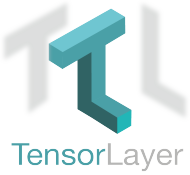
TensorLayer Documentation

Release 1.7.3

TensorLayer contributors

May 17, 2018

Contents



Good News: We won the **Best Open Source Software Award** @ACM Multimedia (MM) 2017.

TensorLayer is a Deep Learning (DL) and Reinforcement Learning (RL) library extended from Google TensorFlow. It provides popular DL and RL modules that can be easily customized and assembled for tackling real-world machine learning problems. More details can be found here.

Note: If you got problem to read the docs online, you could download the repository on GitHub, then go to `/docs/_build/html/index.html` to read the docs offline. The `_build` folder can be generated in `docs` using `make html`.

CHAPTER 1

User Guide

The TensorLayer user guide explains how to install TensorFlow, CUDA and cuDNN, how to build and train neural networks using TensorLayer, and how to contribute to the library as a developer.

1.1 Installation

TensorLayer has some prerequisites that need to be installed first, including TensorFlow , numpy and matplotlib. For GPU support CUDA and cuDNN are required.

If you run into any trouble, please check the TensorFlow installation instructions which cover installing the TensorFlow for a range of operating systems including Mac OX, Linux and Windows, or ask for help on tensorlayer@gmail.com or FQA.

1.1.1 Step 1 : Install dependencies

TensorLayer is build on the top of Python-version TensorFlow, so please install Python first.

Note: We highly recommend python3 instead of python2 for the sake of future.

Python includes `pip` command for installing additional modules is recommended. Besides, a virtual environment via `virtualenv` can help you to manage python packages.

Take Python3 on Ubuntu for example, to install Python includes `pip`, run the following commands:

```
sudo apt-get install python3
sudo apt-get install python3-pip
sudo pip3 install virtualenv
```

To build a virtual environment and install dependencies into it, run the following commands: (You can also skip to Step 3, automatically install the prerequisites by TensorLayer)

```
virtualenv env
env/bin/pip install matplotlib
env/bin/pip install numpy
env/bin/pip install scipy
env/bin/pip install scikit-image
```

To check the installed packages, run the following command:

```
env/bin/pip list
```

After that, you can run python script by using the virtual python as follow.

```
env/bin/python *.py
```

1.1.2 Step 2 : TensorFlow

The installation instructions of TensorFlow are written to be very detailed on TensorFlow website. However, there are something need to be considered. For example, TensorFlow officially supports GPU acceleration for Linux, Mac OX and Windows at present.

Warning: For ARM processor architecture, you need to install TensorFlow from source.

1.1.3 Step 3 : TensorLayer

The simplest way to install TensorLayer is as follow, it will also install the numpy and matplotlib automatically.

```
[stable version] pip install tensorlayer
[master version] pip install git+https://github.com/zsdonghao/tensorlayer.git
```

However, if you want to modify or extend TensorLayer, you can download the repository from Github and install it as follow.

```
cd to the root of the git tree
pip install -e .
```

This command will run the `setup.py` to install TensorLayer. The `-e` reflects editable, then you can edit the source code in `tensorlayer` folder, and `import` the edited TensorLayer.

1.1.4 Step 4 : GPU support

Thanks to NVIDIA supports, training a fully connected network on a GPU, which may be 10 to 20 times faster than training them on a CPU. For convolutional network, may have 50 times faster. This requires an NVIDIA GPU with CUDA and cuDNN support.

CUDA

The TensorFlow website also teach how to install the CUDA and cuDNN, please see TensorFlow GPU Support.

Download and install the latest CUDA is available from NVIDIA website:

- CUDA download and install

If CUDA is set up correctly, the following command should print some GPU information on the terminal:

```
python -c "import tensorflow"
```

cuDNN

Apart from CUDA, NVIDIA also provides a library for common neural network operations that especially speeds up Convolutional Neural Networks (CNNs). Again, it can be obtained from NVIDIA after registering as a developer (it take a while):

Download and install the latest cuDNN is available from NVIDIA website:

- cuDNN download and install

To install it, copy the `*.h` files to `/usr/local/cuda/include` and the `lib*` files to `/usr/local/cuda/lib64`.

1.1.5 Windows User

TensorLayer is built on the top of Python-version TensorFlow, so please install Python first. NoteWe highly recommend installing Anaconda. The lowest version requirements of Python is py35.

Anaconda download

GPU support

Thanks to NVIDIA supports, training a fully connected network on a GPU, which may be 10 to 20 times faster than training them on a CPU. For convolutional network, may have 50 times faster. This requires an NVIDIA GPU with CUDA and cuDNN support.

1. Installing Microsoft Visual Studio

You should preinstall Microsoft Visual Studio (VS) before installing CUDA. The lowest version requirements is VS2010. We recommend installing VS2015 or VS2013. CUDA7.5 supports VS2010, VS2012 and VS2013. CUDA8.0 also supports VS2015.

2. Installing CUDA

Download and install the latest CUDA is available from NVIDIA website:

CUDA download

We do not recommend modifying the default installation directory.

3. Installing cuDNN

The NVIDIA CUDA® Deep Neural Network library (cuDNN) is a GPU-accelerated library of primitives for deep neural networks. Download and extract the latest cuDNN is available from NVIDIA website:

cuDNN download

After extracting cuDNN, you will get three folders (bin, lib, include). Then these folders should be copied to CUDA installation. (The default installation directory is *C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v8.0*)

Installing TensorLayer

You can easily install Tensorlayer using pip in CMD

```
pip install tensorflow        #CPU version
pip install tensorflow-gpu    #GPU version (GPU version and CPU version just choose one)
pip install tensorlayer       #Install tensorlayer
```

Test

Enter "python" in CMD. Then:

```
import tensorlayer
```

If there is no error and the following output is displayed, the GPU version is successfully installed.

```
successfully opened CUDA library cublas64_80.dll locally
successfully opened CUDA library cuDNN64_5.dll locally
successfully opened CUDA library cufft64_80.dll locally
successfully opened CUDA library nvcuda.dll locally
successfully opened CUDA library curand64_80.dll locally
```

If there is no error, the CPU version is successfully installed.

1.1.6 Issue

If you get the following output when import tensorlayer, please read FQA.

```
_tkinter.TclError: no display name and no $DISPLAY environment variable
```

1.2 Tutorial

For deep learning, this tutorial will walk you through building handwritten digits classifiers using the MNIST dataset, arguably the "Hello World" of neural networks. For reinforcement learning, we will let computer learns to play Pong game from the original screen inputs. For nature language processing, we start from word embedding, and then describe language modeling and machine translation.

This tutorial includes all modularized implementation of Google TensorFlow Deep Learning tutorial, so you could read TensorFlow Deep Learning tutorial as the same time [en] [cn] .

Note: For experts: Read the source code of `InputLayer` and `DenseLayer`, you will understand how TensorLayer work. After that, we recommend you to read the codes on Github directly.

1.2.1 Before we start

The tutorial assumes that you are somewhat familiar with neural networks and TensorFlow (the library which TensorLayer is built on top of). You can try to learn the basic of neural network from the Deeplearning Tutorial.

For a more slow-paced introduction to artificial neural networks, we recommend Convolutional Neural Networks for Visual Recognition by Andrej Karpathy et al., Neural Networks and Deep Learning by Michael Nielsen.

To learn more about TensorFlow, have a look at the TensorFlow tutorial. You will not need all of it, but a basic understanding of how TensorFlow works is required to be able to use TensorLayer. If you're new to TensorFlow, going through that tutorial.

1.2.2 TensorLayer is simple

The following code shows a simple example of TensorLayer, see `tutorial_mnist_simple.py` . We provide a lot of simple functions like `fit()` , `test()`), however, if you want to understand the details and be a machine learning expert, we suggest you to train the network by using TensorFlow's methods like `sess.run()`, see `tutorial_mnist.py` for more details.

```
import tensorflow as tf
import tensorlayer as tl

sess = tf.InteractiveSession()

# prepare data
X_train, y_train, X_val, y_val, X_test, y_test = \
                                tl.files.load_mnist_dataset(shape=(-1,784))

# define placeholder
x = tf.placeholder(tf.float32, shape=[None, 784], name='x')
y_ = tf.placeholder(tf.int64, shape=[None, ], name='y_')

# define the network
network = tl.layers.InputLayer(x, name='input_layer')
network = tl.layers.DropoutLayer(network, keep=0.8, name='drop1')
network = tl.layers.DenseLayer(network, n_units=800,
                                act = tf.nn.relu, name='relu1')
network = tl.layers.DropoutLayer(network, keep=0.5, name='drop2')
network = tl.layers.DenseLayer(network, n_units=800,
                                act = tf.nn.relu, name='relu2')
network = tl.layers.DropoutLayer(network, keep=0.5, name='drop3')
# the softmax is implemented internally in tl.cost.cross_entropy(y, y_, 'cost') to
# speed up computation, so we use identity here.
# see tf.nn.sparse_softmax_cross_entropy_with_logits()
network = tl.layers.DenseLayer(network, n_units=10,
                                act = tf.identity,
                                name='output_layer')
# define cost function and metric.
y = network.outputs
cost = tl.cost.cross_entropy(y, y_, 'cost')
correct_prediction = tf.equal(tf.argmax(y, 1), y_)
acc = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
y_op = tf.argmax(tf.nn.softmax(y), 1)

# define the optimizer
train_params = network.all_params
train_op = tf.train.AdamOptimizer(learning_rate=0.0001, beta1=0.9, beta2=0.999,
                            epsilon=1e-08, use_locking=False).minimize(cost, var_
↪list=train_params)

# initialize all variables in the session
tl.layers.initialize_global_variables(sess)
```

(continues on next page)

(continued from previous page)

```
# print network information
network.print_params()
network.print_layers()

# train the network
tl.utils.fit(sess, network, train_op, cost, X_train, y_train, x, y_,
            acc=acc, batch_size=500, n_epoch=500, print_freq=5,
            X_val=X_val, y_val=y_val, eval_train=False)

# evaluation
tl.utils.test(sess, network, acc, X_test, y_test, x, y_, batch_size=None, cost=cost)

# save the network to .npz file
tl.files.save_npz(network.all_params , name='model.npz')
sess.close()
```

1.2.3 Run the MNIST example

In the first part of the tutorial, we will just run the MNIST example that's included in the source distribution of TensorLayer. MNIST dataset contains 60000 handwritten digits that is commonly used for training various image processing systems, each of digit has 28x28 pixels.

We assume that you have already run through the *Installation*. If you haven't done so already, get a copy of the source tree of TensorLayer, and navigate to the folder in a terminal window. Enter the folder and run the `tutorial_mnist.py` example script:

```
python tutorial_mnist.py
```

If everything is set up correctly, you will get an output like the following:

```
tensorlayer: GPU MEM Fraction 0.300000
Downloading train-images-idx3-ubyte.gz
Downloading train-labels-idx1-ubyte.gz
Downloading t10k-images-idx3-ubyte.gz
Downloading t10k-labels-idx1-ubyte.gz

X_train.shape (50000, 784)
y_train.shape (50000,)
X_val.shape (10000, 784)
```

(continues on next page)

(continued from previous page)

```
y_val.shape (10000,)
X_test.shape (10000, 784)
y_test.shape (10000,)
X float32   y int64

[TL] InputLayer   input_layer (?, 784)
[TL] DropoutLayer drop1: keep: 0.800000
[TL] DenseLayer   relu1: 800, relu
[TL] DropoutLayer drop2: keep: 0.500000
[TL] DenseLayer   relu2: 800, relu
[TL] DropoutLayer drop3: keep: 0.500000
[TL] DenseLayer   output_layer: 10, identity

param 0: (784, 800) (mean: -0.000053, median: -0.000043 std: 0.035558)
param 1: (800,)     (mean:  0.000000, median:  0.000000 std: 0.000000)
param 2: (800, 800) (mean:  0.000008, median:  0.000041 std: 0.035371)
param 3: (800,)     (mean:  0.000000, median:  0.000000 std: 0.000000)
param 4: (800, 10)  (mean:  0.000469, median:  0.000432 std: 0.049895)
param 5: (10,)      (mean:  0.000000, median:  0.000000 std: 0.000000)
num of params: 1276810

layer 0: Tensor("dropout/mul_1:0", shape=(?, 784), dtype=float32)
layer 1: Tensor("Relu:0", shape=(?, 800), dtype=float32)
layer 2: Tensor("dropout_1/mul_1:0", shape=(?, 800), dtype=float32)
layer 3: Tensor("Relu_1:0", shape=(?, 800), dtype=float32)
layer 4: Tensor("dropout_2/mul_1:0", shape=(?, 800), dtype=float32)
layer 5: Tensor("add_2:0", shape=(?, 10), dtype=float32)

learning_rate: 0.000100
batch_size: 128

Epoch 1 of 500 took 0.342539s
  train loss: 0.330111
  val loss: 0.298098
  val acc: 0.910700
Epoch 10 of 500 took 0.356471s
  train loss: 0.085225
  val loss: 0.097082
  val acc: 0.971700
Epoch 20 of 500 took 0.352137s
  train loss: 0.040741
  val loss: 0.070149
  val acc: 0.978600
Epoch 30 of 500 took 0.350814s
  train loss: 0.022995
  val loss: 0.060471
  val acc: 0.982800
Epoch 40 of 500 took 0.350996s
  train loss: 0.013713
  val loss: 0.055777
  val acc: 0.983700
...
```

The example script allows you to try different models, including Multi-Layer Perceptron, Dropout, Dropconnect, Stacked Denoising Autoencoder and Convolutional Neural Network. Select different models from `if __name__ == '__main__':`.

```
main_test_layers(model='relu')
main_test_denoise_AE(model='relu')
main_test_stacked_denoise_AE(model='relu')
main_test_cnn_layer()
```

1.2.4 Understand the MNIST example

Let's now investigate what's needed to make that happen! To follow along, open up the source code.

Preface

The first thing you might notice is that besides TensorLayer, we also import numpy and tensorflow:

```
import tensorflow as tf
import tensorlayer as tl
from tensorlayer.layers import set_keep
import numpy as np
import time
```

As we know, TensorLayer is built on top of TensorFlow, it is meant as a supplement helping with some tasks, not as a replacement. You will always mix TensorLayer with some vanilla TensorFlow code. The `set_keep` is used to access the placeholder of keeping probabilities when using Denoising Autoencoder.

Loading data

The first piece of code defines a function `load_mnist_dataset()`. Its purpose is to download the MNIST dataset (if it hasn't been downloaded yet) and return it in the form of regular numpy arrays. There is no TensorLayer involved at all, so for the purpose of this tutorial, we can regard it as:

```
X_train, y_train, X_val, y_val, X_test, y_test = \
                    tl.files.load_mnist_dataset(shape=(-1,784))
```

`X_train.shape` is `(50000, 784)`, to be interpreted as: 50,000 images and each image has 784 pixels. `y_train.shape` is simply `(50000,)`, which is a vector the same length of `X_train` giving an integer class label for each image – namely, the digit between 0 and 9 depicted in the image (according to the human annotator who drew that digit).

For Convolutional Neural Network example, the MNIST can be load as 4D version as follow:

```
X_train, y_train, X_val, y_val, X_test, y_test = \
            tl.files.load_mnist_dataset(shape=(-1, 28, 28, 1))
```

`X_train.shape` is `(50000, 28, 28, 1)` which represents 50,000 images with 1 channel, 28 rows and 28 columns each. Channel one is because it is a grey scale image, every pixel have only one value.

Building the model

This is where TensorLayer steps in. It allows you to define an arbitrarily structured neural network by creating and stacking or merging layers. Since every layer knows its immediate incoming layers, the output layer (or output layers) of a network double as a handle to the network as a whole, so usually this is the only thing we will pass on to the rest of the code.

As mentioned above, `tutorial_mnist.py` supports four types of models, and we implement that via easily exchangeable functions of the same interface. First, we'll define a function that creates a Multi-Layer Perceptron (MLP) of a fixed architecture, explaining all the steps in detail. We'll then implement a Denosing Autoencoder (DAE), after that we will then stack all Denoising Autoencoder and supervised fine-tune them. Finally, we'll show how to create a Convolutional Neural Network (CNN). In addition, a simple example for MNIST dataset in `tutorial_mnist_simple.py`, a CNN example for CIFAR-10 dataset in `tutorial_cifar10_tfrecord.py`.

Multi-Layer Perceptron (MLP)

The first script, `main_test_layers()`, creates an MLP of two hidden layers of 800 units each, followed by a softmax output layer of 10 units. It applies 20% dropout to the input data and 50% dropout to the hidden layers.

To feed data into the network, TensofFlow placeholders need to be defined as follow. The `None` here means the network will accept input data of arbitrary batchsize after compilation. The `x` is used to hold the `X_train` data and `y_` is used to hold the `y_train` data. If you know the batchsize beforehand and do not need this flexibility, you should give the batchsize here – especially for convolutional layers, this can allow TensorFlow to apply some optimizations.

```
x = tf.placeholder(tf.float32, shape=[None, 784], name='x')
y_ = tf.placeholder(tf.int64, shape=[None, ], name='y_')
```

The foundation of each neural network in TensorLayer is an *InputLayer* instance representing the input data that will subsequently be fed to the network. Note that the `InputLayer` is not tied to any specific data yet.

```
network = tl.layers.InputLayer(x, name='input_layer')
```

Before adding the first hidden layer, we'll apply 20% dropout to the input data. This is realized via a *DropoutLayer* instance:

```
network = tl.layers.DropoutLayer(network, keep=0.8, name='drop1')
```

Note that the first constructor argument is the incoming layer, the second argument is the keeping probability for the activation value. Now we'll proceed with the first fully-connected hidden layer of 800 units. Note that when stacking a *DenseLayer*.

```
network = tl.layers.DenseLayer(network, n_units=800, act = tf.nn.relu, name='relu1')
```

Again, the first constructor argument means that we're stacking `network` on top of `network`. `n_units` simply gives the number of units for this fully-connected layer. `act` takes an activation function, several of which are defined in `tensorflow.nn` and *tensorlayer.activation*. Here we've chosen the rectifier, so we'll obtain ReLUs. We'll now add dropout of 50%, another 800-unit dense layer and 50% dropout again:

```
network = tl.layers.DropoutLayer(network, keep=0.5, name='drop2')
network = tl.layers.DenseLayer(network, n_units=800, act = tf.nn.relu, name='relu2')
network = tl.layers.DropoutLayer(network, keep=0.5, name='drop3')
```

Finally, we'll add the fully-connected output layer which the `n_units` equals to the number of classes. Note that, the softmax is implemented internally in `tf.nn.sparse_softmax_cross_entropy_with_logits()` to speed up computation, so we used identity in the last layer, more details in `tl.cost.cross_entropy()`.

```
network = tl.layers.DenseLayer(network,
                               n_units=10,
                               act = tf.identity,
                               name='output_layer')
```

As mentioned above, each layer is linked to its incoming layer(s), so we only need the output layer(s) to access a network in TensorLayer:

```
y = network.outputs
y_op = tf.argmax(tf.nn.softmax(y), 1)
cost = tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(y, y_))
```

Here, `network.outputs` is the 10 identity outputs from the network (in one hot format), `y_op` is the integer output represents the class index. While `cost` is the cross-entropy between target and predicted labels.

Denoising Autoencoder (DAE)

Autoencoder is an unsupervised learning model which is able to extract representative features, it has become more widely used for learning generative models of data and Greedy layer-wise pre-train. For vanilla Autoencoder see Deeplearning Tutorial.

The script `main_test_denoise_AE()` implements a Denoising Autoencoder with corrosion rate of 50%. The Autoencoder can be defined as follow, where an Autoencoder is represented by a `DenseLayer`:

```
network = tl.layers.InputLayer(x, name='input_layer')
network = tl.layers.DropoutLayer(network, keep=0.5, name='denoising1')
network = tl.layers.DenseLayer(network, n_units=200, act=tf.nn.sigmoid, name='sigmoid1
↪')
recon_layer1 = tl.layers.ReconLayer(network,
                                    x_recon=x,
                                    n_units=784,
                                    act=tf.nn.sigmoid,
                                    name='recon_layer1')
```

To train the `DenseLayer`, simply run `ReconLayer.pretrain()`, if using denoising Autoencoder, the name of corrosion layer (a `DropoutLayer`) need to be specified as follow. To save the feature images, set `save` to `True`. There are many kinds of pre-train metrices according to different architectures and applications. For sigmoid activation, the Autoencoder can be implemented by using KL divergence, while for rectifer, L1 regularization of activation outputs can make the output to be sparse. So the default behaviour of `ReconLayer` only provide KLD and cross-entropy for sigmoid activation function and L1 of activation outputs and mean-squared-error for rectifing activation function. We recommend you to modify `ReconLayer` to achieve your own pre-train metrice.

```
recon_layer1.pretrain(sess,
                      x=x,
                      X_train=X_train,
                      X_val=X_val,
                      denoise_name='denoising1',
                      n_epoch=200,
                      batch_size=128,
                      print_freq=10,
                      save=True,
                      save_name='w1pre_')
```

In addition, the script `main_test_stacked_denoise_AE()` shows how to stacked multiple Autoencoder to one network and then fine-tune.

Convolutional Neural Network (CNN)

Finally, the `main_test_cnn_layer()` script creates two CNN layers and max pooling stages, a fully-connected hidden layer and a fully-connected output layer. More CNN examples can be found in the tutorial scripts, like `tutorial_cifar10_tfrecord.py`.

At the begin, we add a *Conv2dLayer* with 32 filters of size 5x5 on top, follow by max-pooling of factor 2 in both dimensions. And then apply a `Conv2dLayer` with 64 filters of size 5x5 again and follow by a max_pool again. After that, flatten the 4D output to 1D vector by using `FlattenLayer`, and apply a dropout with 50% to last hidden layer. The `?` represents arbitrary batch_size.

Note, `tutorial_mnist.py` introduces the simplified CNN API for beginner.

```
network = tl.layers.InputLayer(x, name='input_layer')
network = tl.layers.Conv2dLayer(network,
                        act = tf.nn.relu,
                        shape = [5, 5, 1, 32],  # 32 features for each 5x5 patch
                        strides=[1, 1, 1, 1],
                        padding='SAME',
                        name ='cnn_layer1')     # output: (?, 28, 28, 32)
network = tl.layers.PoolLayer(network,
                        ksize=[1, 2, 2, 1],
                        strides=[1, 2, 2, 1],
                        padding='SAME',
                        pool = tf.nn.max_pool,
                        name ='pool_layer1',)   # output: (?, 14, 14, 32)
network = tl.layers.Conv2dLayer(network,
                        act = tf.nn.relu,
                        shape = [5, 5, 32, 64], # 64 features for each 5x5 patch
                        strides=[1, 1, 1, 1],
                        padding='SAME',
                        name ='cnn_layer2')     # output: (?, 14, 14, 64)
network = tl.layers.PoolLayer(network,
                        ksize=[1, 2, 2, 1],
                        strides=[1, 2, 2, 1],
                        padding='SAME',
                        pool = tf.nn.max_pool,
                        name ='pool_layer2',)   # output: (?, 7, 7, 64)
network = tl.layers.FlattenLayer(network, name='flatten_layer')
                                                # output: (?, 3136)
network = tl.layers.DropoutLayer(network, keep=0.5, name='drop1')
                                                # output: (?, 3136)
network = tl.layers.DenseLayer(network, n_units=256, act = tf.nn.relu, name='relu1')
                                                # output: (?, 256)
network = tl.layers.DropoutLayer(network, keep=0.5, name='drop2')
                                                # output: (?, 256)
network = tl.layers.DenseLayer(network, n_units=10,
                act = tf.identity, name='output_layer')
                                                # output: (?, 10)
```

Note: For experts: `Conv2dLayer` will create a convolutional layer using `tensorflow.nn.conv2d`, TensorFlow's default convolution.

Training the model

The remaining part of the `tutorial_mnist.py` script copes with setting up and running a training loop over the MNIST dataset by using cross-entropy only.

Dataset iteration

An iteration function for synchronously iterating over two numpy arrays of input data and targets, respectively, in mini-batches of a given number of items. More iteration function can be found in `tensorlayer.iterate`

```
tl.iterate.minibatches(inputs, targets, batchsize, shuffle=False)
```

Loss and update expressions

Continuing, we create a loss expression to be minimized in training:

```
y = network.outputs
y_op = tf.argmax(tf.nn.softmax(y), 1)
cost = tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(y, y_))
```

More cost or regularization can be applied here, take `main_test_layers()` for example, to apply max-norm on the weight matrices, we can add the following line:

```
cost = cost + tl.cost.maxnorm_regularizer(1.0)(network.all_params[0]) +
              tl.cost.maxnorm_regularizer(1.0)(network.all_params[2])
```

Depending on the problem you are solving, you will need different loss functions, see *`tensorlayer.cost`* for more.

Having the model and the loss function defined, we create update expressions for training the network. TensorLayer do not provide many optimizers, we used TensorFlow's optimizer instead:

```
train_params = network.all_params
train_op = tf.train.AdamOptimizer(learning_rate, beta1=0.9, beta2=0.999,
    epsilon=1e-08, use_locking=False).minimize(cost, var_list=train_params)
```

For training the network, we fed data and the keeping probabilities to the `feed_dict`.

```
feed_dict = {x: X_train_a, y_: y_train_a}
feed_dict.update( network.all_drop )
sess.run(train_op, feed_dict=feed_dict)
```

While, for validation and testing, we use slightly different way. All dropout, dropconnect, corrosion layers need to be disable. `tl.utils.dict_to_one` set all `network.all_drop` to 1.

```
dp_dict = tl.utils.dict_to_one( network.all_drop )
feed_dict = {x: X_test_a, y_: y_test_a}
feed_dict.update(dp_dict)
err, ac = sess.run([cost, acc], feed_dict=feed_dict)
```

As an additional monitoring quantity, we create an expression for the classification accuracy:

```
correct_prediction = tf.equal(tf.argmax(y, 1), y_)
acc = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

What Next?

We also have a more advanced image classification example in `tutorial_cifar10_tfrecord.py`. Please read the code and notes, figure out how to generate more training data and what is local response normalization. After that, try to implement Residual Network (Hint: you may want to use the Layer.outputs).

1.2.5 Run the Pong Game example

In the second part of the tutorial, we will run the Deep Reinforcement Learning example that is introduced by Karpathy in Deep Reinforcement Learning: Pong from Pixels.

```
python tutorial_atari_pong.py
```

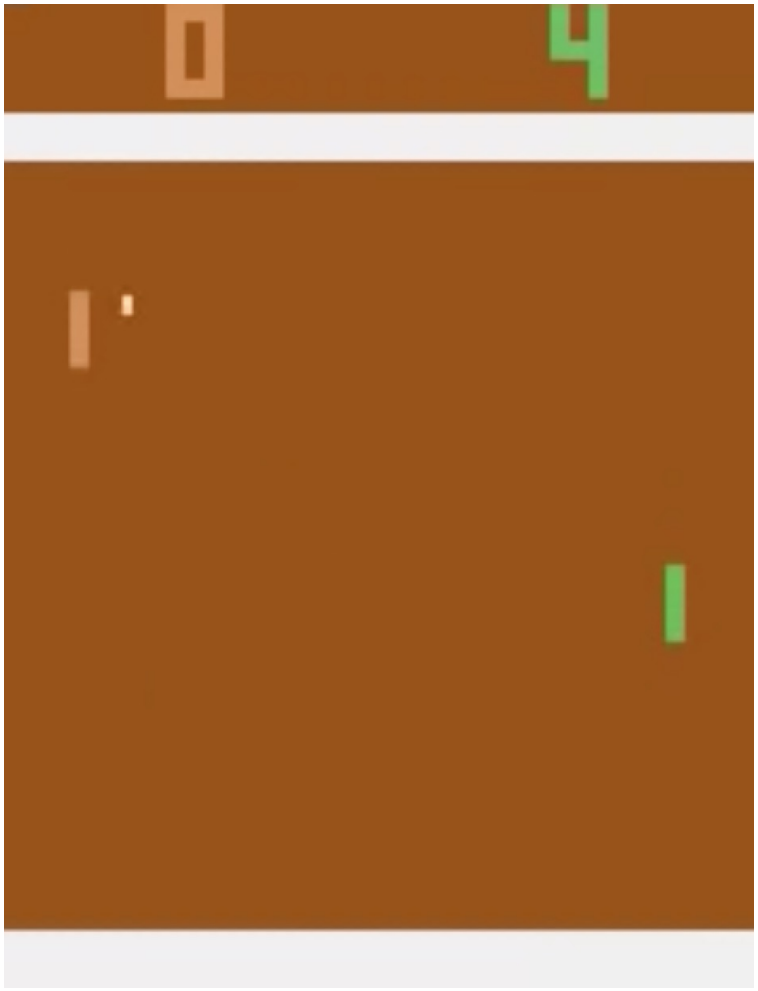
Before running the tutorial code, you need to install OpenAI gym environment which is a benchmark for Reinforcement Learning. If everything is set up correctly, you will get an output like the following:

```
[2016-07-12 09:31:59,760] Making new env: Pong-v0
  [TL] InputLayer input_layer (?, 6400)
  [TL] DenseLayer relu1: 200, relu
  [TL] DenseLayer output_layer: 3, identity
  param 0: (6400, 200) (mean: -0.000009, median: -0.000018 std: 0.017393)
  param 1: (200,) (mean: 0.000000, median: 0.000000 std: 0.000000)
  param 2: (200, 3) (mean: 0.002239, median: 0.003122 std: 0.096611)
  param 3: (3,) (mean: 0.000000, median: 0.000000 std: 0.000000)
  num of params: 1280803
  layer 0: Tensor("Relu:0", shape=(?, 200), dtype=float32)
  layer 1: Tensor("add_1:0", shape=(?, 3), dtype=float32)
episode 0: game 0 took 0.17381s, reward: -1.000000
episode 0: game 1 took 0.12629s, reward: 1.000000  !!!!!!!!
episode 0: game 2 took 0.17082s, reward: -1.000000
episode 0: game 3 took 0.08944s, reward: -1.000000
episode 0: game 4 took 0.09446s, reward: -1.000000
episode 0: game 5 took 0.09440s, reward: -1.000000
episode 0: game 6 took 0.32798s, reward: -1.000000
episode 0: game 7 took 0.74437s, reward: -1.000000
episode 0: game 8 took 0.43013s, reward: -1.000000
episode 0: game 9 took 0.42496s, reward: -1.000000
episode 0: game 10 took 0.37128s, reward: -1.000000
episode 0: game 11 took 0.08979s, reward: -1.000000
episode 0: game 12 took 0.09138s, reward: -1.000000
episode 0: game 13 took 0.09142s, reward: -1.000000
episode 0: game 14 took 0.09639s, reward: -1.000000
episode 0: game 15 took 0.09852s, reward: -1.000000
episode 0: game 16 took 0.09984s, reward: -1.000000
episode 0: game 17 took 0.09575s, reward: -1.000000
episode 0: game 18 took 0.09416s, reward: -1.000000
episode 0: game 19 took 0.08674s, reward: -1.000000
episode 0: game 20 took 0.09628s, reward: -1.000000
resetting env. episode reward total was -20.000000. running mean: -20.000000
episode 1: game 0 took 0.09910s, reward: -1.000000
episode 1: game 1 took 0.17056s, reward: -1.000000
episode 1: game 2 took 0.09306s, reward: -1.000000
episode 1: game 3 took 0.09556s, reward: -1.000000
episode 1: game 4 took 0.12520s, reward: 1.000000  !!!!!!!!
episode 1: game 5 took 0.17348s, reward: -1.000000
episode 1: game 6 took 0.09415s, reward: -1.000000
```

This example allow computer to learn how to play Pong game from the screen inputs, just like human behavior. After training for 15,000 episodes, the computer can win 20% of the games. The computer win 35% of the games at 20,000 episode, we can seen the computer learn faster and faster as it has more winning data to train. If you run it for 30,000 episode, it start to win.

```
render = False
resume = False
```

Setting `render` to `True`, if you want to display the game environment. When you run the code again, you can set `resume` to `True`, the code will load the existing model and train the model basic on it.



1.2.6 Understand Reinforcement learning

Pong Game

To understand Reinforcement Learning, we let computer to learn how to play Pong game from the original screen inputs. Before we start, we highly recommend you to go through a famous blog called Deep Reinforcement Learning: Pong from Pixels which is a minimalistic implementation of Deep Reinforcement Learning by using python-numpy and OpenAI gym environment.

```
python tutorial_atari_pong.py
```

Policy Network

In Deep Reinforcement Learning, the Policy Network is the same with Deep Neural Network, it is our player (or "agent") who output actions to tell what we should do (move UP or DOWN); in Karpathy's code, he only defined 2 actions, UP and DOWN and using a single simgoid output; In order to make our tutorial more generic, we defined 3 actions which are UP, DOWN and STOP (do nothing) by using 3 softmax outputs.

```
# observation for training
states_batch_pl = tf.placeholder(tf.float32, shape=[None, D])

network = tl.layers.InputLayer(states_batch_pl, name='input_layer')
network = tl.layers.DenseLayer(network, n_units=H,
                               act = tf.nn.relu, name='relu1')
network = tl.layers.DenseLayer(network, n_units=3,
                       act = tf.identity, name='output_layer')
probs = network.outputs
sampling_prob = tf.nn.softmax(probs)
```

Then when our agent is playing Pong, it calculates the probabilities of different actions, and then draw sample (action) from this uniform distribution. As the actions are represented by 1, 2 and 3, but the softmax outputs should be start from 0, we calculate the label value by minus 1.

```
prob = sess.run(
    sampling_prob,
    feed_dict={states_batch_pl: x}
)
# action. 1: STOP  2: UP  3: DOWN
action = np.random.choice([1,2,3], p=prob.flatten())
...
ys.append(action - 1)
```

Policy Gradient

Policy gradient methods are end-to-end algorithms that directly learn policy functions mapping states to actions. An approximate policy could be learned directly by maximizing the expected rewards. The parameters of a policy function (e.g. the parameters of a policy network used in the pong example) could be trained and learned under the guidance of the gradient of expected rewards. In other words, we can gradually tune the policy function via updating its parameters, such that it will generate actions from given states towards higher rewards.

An alternative method to policy gradient is Deep Q-Learning (DQN). It is based on Q-Learning that tries to learn a value function (called Q function) mapping states and actions to some value. DQN employs a deep neural network to represent the Q function as a function approximator. The training is done by minimizing temporal-difference errors. A neurobiologically inspired mechanism called "experience replay" is typically used along with DQN to help improve its stability caused by the use of non-linear function approximator.

You can check the following papers to gain better understandings about Reinforcement Learning.

- Reinforcement Learning: An Introduction. Richard S. Sutton and Andrew G. Barto
- Deep Reinforcement Learning. David Silver, Google DeepMind
- UCL Course on RL

The most successful applications of Deep Reinforcement Learning in recent years include DQN with experience replay to play Atari games and AlphaGO that for the first time beats world-class professional GO players. AlphaGO used the policy gradient method to train its policy network that is similar to the example of Pong game.

- Atari - Playing Atari with Deep Reinforcement Learning
- Atari - Human-level control through deep reinforcement learning
- AlphaGO - Mastering the game of Go with deep neural networks and tree search

Dataset iteration

In Reinforcement Learning, we consider a final decision as an episode. In Pong game, a episode is a few dozen games, because the games go up to score of 21 for either player. Then the batch size is how many episode we consider to update the model. In the tutorial, we train a 2-layer policy network with 200 hidden layer units using RMSProp on batches of 10 episodes.

Loss and update expressions

Continuing, we create a loss expression to be minimized in training:

```
actions_batch_pl = tf.placeholder(tf.int32, shape=[None])
discount_rewards_batch_pl = tf.placeholder(tf.float32, shape=[None])
loss = tl.rein.cross_entropy_reward_loss(probs, actions_batch_pl,
                                         discount_rewards_batch_pl)
...
...
sess.run(
    train_op,
    feed_dict={
        states_batch_pl: epx,
        actions_batch_pl: epy,
        discount_rewards_batch_pl: disR
    }
)
```

The loss in a batch is relate to all outputs of Policy Network, all actions we made and the corresponding discounted rewards in a batch. We first compute the loss of each action by multiplying the discounted reward and the cross-entropy between its output and its true action. The final loss in a batch is the sum of all loss of the actions.

What Next?

The tutorial above shows how you can build your own agent, end-to-end. While it has reasonable quality, the default parameters will not give you the best agent model. Here are a few things you can improve.

First of all, instead of conventional MLP model, we can use CNNs to capture the screen information better as Playing Atari with Deep Reinforcement Learning describe.

Also, the default parameters of the model are not tuned. You can try changing the learning rate, decay, or initializing the weights of your model in a different way.

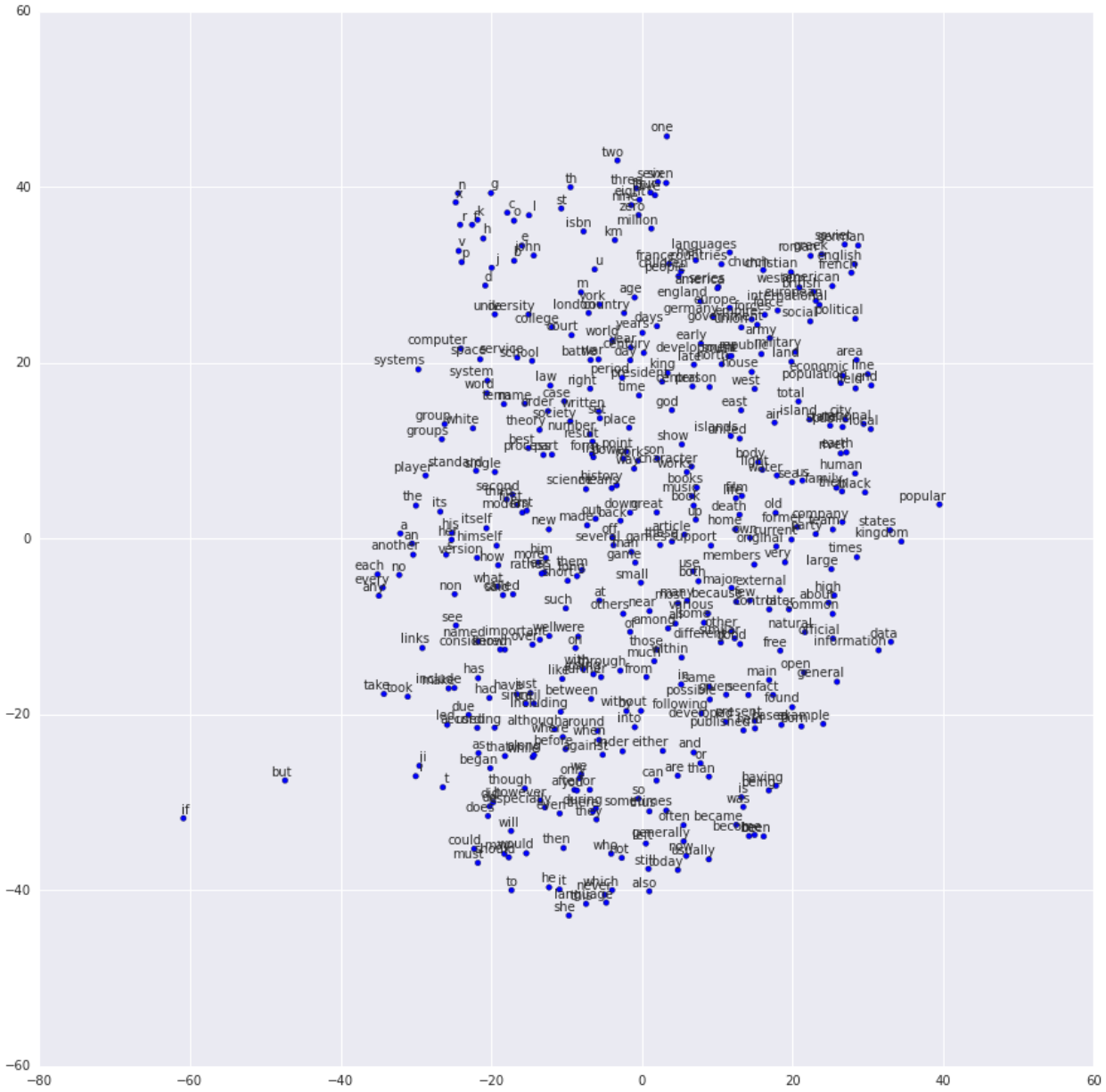
Finally, you can try the model on different tasks (games).

1.2.7 Run the Word2Vec example

In this part of the tutorial, we train a matrix for words, where each word can be represented by a unique row vector in the matrix. In the end, similar words will have similar vectors. Then as we plot out the words into a two-dimensional plane, words that are similar end up clustering nearby each other.

```
python tutorial_word2vec_basic.py
```

If everything is set up correctly, you will get an output in the end.



1.2.8 Understand Word Embedding

Word Embedding

We highly recommend you to read Colah's blog Word Representations to understand why we want to use a vector representation, and how to compute the vectors. (For chinese reader please click. More details about word2vec can be found in Word2vec Parameter Learning Explained.

Bascially, training an embedding matrix is an unsupervised learning. As every word is refected by an unique ID, which is the row index of the embedding matrix, a word can be converted into a vector, it can better represent the meaning. For example, there seems to be a constant male-female difference vector: `woman  man = queen  king`, this means one dimension in the vector represents gender.

The model can be created as follow.

```
# train_inputs is a row vector, a input is an integer id of single word.
# train_labels is a column vector, a label is an integer id of single word.
# valid_dataset is a column vector, a valid set is an integer id of single word.
train_inputs = tf.placeholder(tf.int32, shape=[batch_size])
train_labels = tf.placeholder(tf.int32, shape=[batch_size, 1])
valid_dataset = tf.constant(valid_examples, dtype=tf.int32)

# Look up embeddings for inputs.
emb_net = tl.layers.Word2vecEmbeddingInputlayer(
        inputs = train_inputs,
        train_labels = train_labels,
        vocabulary_size = vocabulary_size,
        embedding_size = embedding_size,
        num_sampled = num_sampled,
        nce_loss_args = {},
        E_init = tf.random_uniform_initializer(minval=-1.0, maxval=1.0),
        E_init_args = {},
        nce_W_init = tf.truncated_normal_initializer(
                            stddev=float(1.0/np.sqrt(embedding_size))),
        nce_W_init_args = {},
        nce_b_init = tf.constant_initializer(value=0.0),
        nce_b_init_args = {},
        name ='word2vec_layer',
    )
```

Dataset iteration and loss

Word2vec uses Negative Sampling and Skip-Gram model for training. Noise-Contrastive Estimation Loss (NCE) can help to reduce the computation of loss. Skip-Gram inverts context and targets, tries to predict each context word from its target word. We use `tl.nlp.generate_skip_gram_batch` to generate training data as follow, see `tutorial_generate_text.py`.

```
# NCE cost expression is provided by Word2vecEmbeddingInputlayer
cost = emb_net.nce_cost
train_params = emb_net.all_params

train_op = tf.train.AdagradOptimizer(learning_rate, initial_accumulator_value=0.1,
          use_locking=False).minimize(cost, var_list=train_params)

data_index = 0
while (step < num_steps):
  batch_inputs, batch_labels, data_index = tl.nlp.generate_skip_gram_batch(
                data=data, batch_size=batch_size, num_skips=num_skips,
                skip_window=skip_window, data_index=data_index)
  feed_dict = {train_inputs : batch_inputs, train_labels : batch_labels}
  _, loss_val = sess.run([train_op, cost], feed_dict=feed_dict)
```

Restore existing Embedding matrix

In the end of training the embedding matrix, we save the matrix and corresponding dictionaries. Then next time, we can restore the matrix and directories as follow. (see `main_restore_embedding_layer()` in `tutorial_generate_text.py`)

```
vocabulary_size = 50000
embedding_size = 128
model_file_name = "model_word2vec_50k_128"
batch_size = None

print("Load existing embedding matrix and dictionaries")
all_var = tl.files.load_npy_to_any(name=model_file_name+'.npy')
data = all_var['data']; count = all_var['count']
dictionary = all_var['dictionary']
reverse_dictionary = all_var['reverse_dictionary']

tl.nlp.save_vocab(count, name='vocab_'+model_file_name+'.txt')

del all_var, data, count

load_params = tl.files.load_npz(name=model_file_name+'.npz')

x = tf.placeholder(tf.int32, shape=[batch_size])
y_ = tf.placeholder(tf.int32, shape=[batch_size, 1])

emb_net = tl.layers.EmbeddingInputlayer(
                inputs = x,
                vocabulary_size = vocabulary_size,
                embedding_size = embedding_size,
                name ='embedding_layer')

tl.layers.initialize_global_variables(sess)

tl.files.assign_params(sess, [load_params[0]], emb_net)
```

1.2.9 Run the PTB example

Penn TreeBank (PTB) dataset is used in many LANGUAGE MODELING papers, including "Empirical Evaluation and Combination of Advanced Language Modeling Techniques", "Recurrent Neural Network Regularization". It consists of 929k training words, 73k validation words, and 82k test words. It has 10k words in its vocabulary.

The PTB example is trying to show how to train a recurrent neural network on a challenging task of language modeling.

Given a sentence "I am from Imperial College London", the model can learn to predict "Imperial College London" from "from Imperial College". In other word, it predict the next word in a text given a history of previous words. In the previous example , `num_steps` (sequence length) is 3.

```
python tutorial_ptb_lstm.py
```

The script provides three settings (small, medium, large), where a larger model has better performance. You can choose different settings in:

```
flags.DEFINE_string(
    "model", "small",
    "A type of model. Possible options are: small, medium, large.")
```

If you choose the small setting, you can see:

```
Epoch: 1 Learning rate: 1.000
0.004 perplexity: 5220.213 speed: 7635 wps
0.104 perplexity: 828.871 speed: 8469 wps
```

(continues on next page)

(continued from previous page)

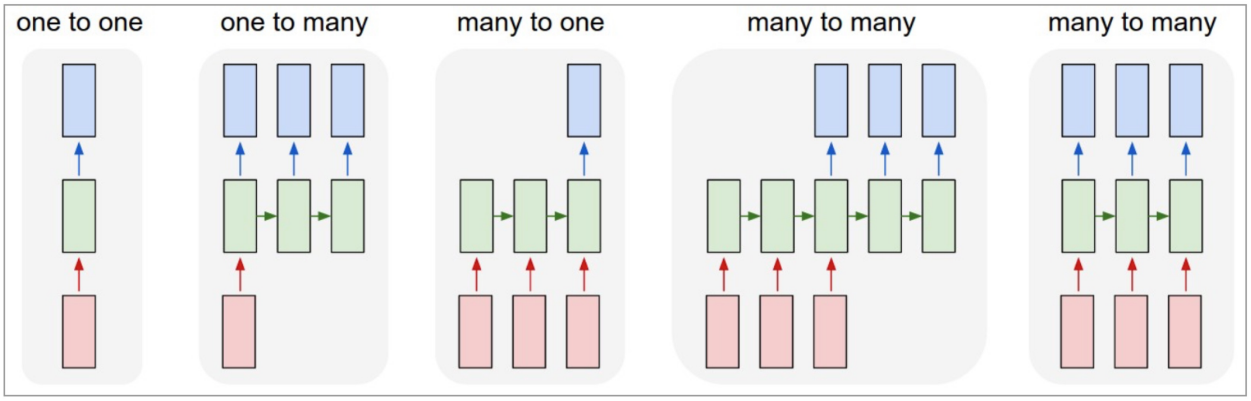
```
0.204 perplexity: 614.071 speed: 8839 wps
0.304 perplexity: 495.485 speed: 8889 wps
0.404 perplexity: 427.381 speed: 8940 wps
0.504 perplexity: 383.063 speed: 8920 wps
0.604 perplexity: 345.135 speed: 8920 wps
0.703 perplexity: 319.263 speed: 8949 wps
0.803 perplexity: 298.774 speed: 8975 wps
0.903 perplexity: 279.817 speed: 8986 wps
Epoch: 1 Train Perplexity: 265.558
Epoch: 1 Valid Perplexity: 178.436
...
Epoch: 13 Learning rate: 0.004
0.004 perplexity: 56.122 speed: 8594 wps
0.104 perplexity: 40.793 speed: 9186 wps
0.204 perplexity: 44.527 speed: 9117 wps
0.304 perplexity: 42.668 speed: 9214 wps
0.404 perplexity: 41.943 speed: 9269 wps
0.504 perplexity: 41.286 speed: 9271 wps
0.604 perplexity: 39.989 speed: 9244 wps
0.703 perplexity: 39.403 speed: 9236 wps
0.803 perplexity: 38.742 speed: 9229 wps
0.903 perplexity: 37.430 speed: 9240 wps
Epoch: 13 Train Perplexity: 36.643
Epoch: 13 Valid Perplexity: 121.475
Test Perplexity: 116.716
```

The PTB example shows that RNN is able to model language, but this example did not do something practically interesting. However, you should read through this example and "Understand LSTM" in order to understand the basics of RNN. After that, you will learn how to generate text, how to achieve language translation, and how to build a question answering system by using RNN.

1.2.10 Understand LSTM

Recurrent Neural Network

We personally think Andrey Karpathy's blog is the best material to Understand Recurrent Neural Network , after reading that, Colah's blog can help you to Understand LSTM Network [chinese] which can solve The Problem of Long-Term Dependencies. We will not describe more about the theory of RNN, so please read through these blogs before you go on.



Each rectangle is a vector and arrows represent functions (e.g. matrix multiply). Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state (more on this soon). From left to right: **(1)** Vanilla mode of processing without RNN, from fixed-sized input to fixed-sized output (e.g. image classification). **(2)** Sequence output (e.g. image captioning takes an image and outputs a sentence of words). **(3)** Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment). **(4)** Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French). **(5)** Synced sequence input and output (e.g. video classification where we wish to label each frame of the video). Notice that in every case are no pre-specified constraints on the lengths sequences because the recurrent transformation (green) is fixed and can be applied as many times as we like.

Image by Andrey Karpathy

Synced sequence input and output

The model in PTB example is a typical type of synced sequence input and output, which was described by Karpathy as "(5) Synced sequence input and output (e.g. video classification where we wish to label each frame of the video). Notice that in every case there are no pre-specified constraints on the lengths of sequences because the recurrent transformation (green) can be applied as many times as we like."

The model is built as follows. Firstly, we transfer the words into word vectors by looking up an embedding matrix. In this tutorial, there is no pre-training on the embedding matrix. Secondly, we stack two LSTMs together using dropout between the embedding layer, LSTM layers, and the output layer for regularization. In the final layer, the model provides a sequence of softmax outputs.

The first LSTM layer outputs `[batch_size, num_steps, hidden_size]` for stacking another LSTM after it. The second LSTM layer outputs `[batch_size*num_steps, hidden_size]` for stacking a DenseLayer after it. Then the DenseLayer computes the softmax outputs of each example `n_examples = batch_size*num_steps`).

To understand the PTB tutorial, you can also read TensorFlow PTB tutorial.

(Note that, TensorLayer supports DynamicRNNLayer after v1.1, so you can set the input/output dropouts, number of RNN layers in one single layer)

```
network = tl.layers.EmbeddingInputlayer(
            inputs = x,
            vocabulary_size = vocab_size,
            embedding_size = hidden_size,
            E_init = tf.random_uniform_initializer(-init_scale, init_scale),
            name ='embedding_layer')
if is_training:
    network = tl.layers.DropoutLayer(network, keep=keep_prob, name='drop1')
network = tl.layers.RNNLayer(network,
            cell_fn=tf.contrib.rnn.BasicLSTMCell,
```

(continues on next page)

(continued from previous page)

```
            cell_init_args={'forget_bias': 0.0},
            n_hidden=hidden_size,
            initializer=tf.random_uniform_initializer(-init_scale, init_scale),
            n_steps=num_steps,
            return_last=False,
            name='basic_lstm_layer1')
lstm1 = network
if is_training:
    network = tl.layers.DropoutLayer(network, keep=keep_prob, name='drop2')
network = tl.layers.RNNLayer(network,
            cell_fn=tf.contrib.rnn.BasicLSTMCell,
            cell_init_args={'forget_bias': 0.0},
            n_hidden=hidden_size,
            initializer=tf.random_uniform_initializer(-init_scale, init_scale),
            n_steps=num_steps,
            return_last=False,
            return_seq_2d=True,
            name='basic_lstm_layer2')
lstm2 = network
if is_training:
    network = tl.layers.DropoutLayer(network, keep=keep_prob, name='drop3')
network = tl.layers.DenseLayer(network,
            n_units=vocab_size,
            W_init=tf.random_uniform_initializer(-init_scale, init_scale),
            b_init=tf.random_uniform_initializer(-init_scale, init_scale),
            act = tf.identity, name='output_layer')
```

Dataset iteration

The `batch_size` can be seen as the number of concurrent computations we are running. As the following example shows, the first batch learns the sequence information by using items 0 to 9. The second batch learn the sequence information by using items 10 to 19. So it ignores the information from items 9 to 10 !n If only if we set `batch_size = 1`, it will consider all the information from items 0 to 20.

The meaning of `batch_size` here is not the same as the `batch_size` in the MNIST example. In the MNIST example, `batch_size` reflects how many examples we consider in each iteration, while in the PTB example, `batch_size` is the number of concurrent processes (segments) for accelerating the computation.

Some information will be ignored if `batch_size` > 1, however, if your dataset is "long" enough (a text corpus usually has billions of words), the ignored information would not affect the final result.

In the PTB tutorial, we set `batch_size = 20`, so we divide the dataset into 20 segments. At the beginning of each epoch, we initialize (reset) the 20 RNN states for the 20 segments to zero, then go through the 20 segments separately.

An example of generating training data is as follows:

```
train_data = [i for i in range(20)]
for batch in tl.iterate.ptb_iterator(train_data, batch_size=2, num_steps=3):
    x, y = batch
    print(x, '\n',y)
```

```
... [[ 0  1  2] <---x                       1st subset/ iteration
...  [10 11 12]]
... [[ 1  2  3] <---y
...  [11 12 13]]
```

(continues on next page)

(continued from previous page)

```
...
... [[ 3  4  5]  <--- 1st batch input       2nd subset/ iteration
...  [13 14 15]] <--- 2nd batch input
... [[ 4  5  6]  <--- 1st batch target
...  [14 15 16]] <--- 2nd batch target
...
... [[ 6  7  8]                            3rd subset/ iteration
...  [16 17 18]]
... [[ 7  8  9]
...  [17 18 19]]
```

Note: This example can also be considered as pre-training of the word embedding matrix.

Loss and update expressions

The cost function is the average cost of each mini-batch:

```
# See tensorlayer.cost.cross_entropy_seq() for more details
def loss_fn(outputs, targets, batch_size, num_steps):
    # Returns the cost function of Cross-entropy of two sequences, implement
    # softmax internally.
    # outputs : 2D tensor [batch_size*num_steps, n_units of output layer]
    # targets : 2D tensor [batch_size, num_steps], need to be reshaped.
    # n_examples = batch_size * num_steps
    # so
    # cost is the average cost of each mini-batch (concurrent process).
    loss = tf.nn.seq2seq.sequence_loss_by_example(
        [outputs],
        [tf.reshape(targets, [-1])],
        [tf.ones([batch_size * num_steps])])
    cost = tf.reduce_sum(loss) / batch_size
    return cost

# Cost for Training
cost = loss_fn(network.outputs, targets, batch_size, num_steps)
```

For updating, truncated backpropagation clips values of gradients by the ratio of the sum of their norms, so as to make the learning process tractable.

```
# Truncated Backpropagation for training
with tf.variable_scope('learning_rate'):
    lr = tf.Variable(0.0, trainable=False)
tvars = tf.trainable_variables()
grads, _ = tf.clip_by_global_norm(tf.gradients(cost, tvars),
                                  max_grad_norm)
optimizer = tf.train.GradientDescentOptimizer(lr)
train_op = optimizer.apply_gradients(zip(grads, tvars))
```

In addition, if the epoch index is greater than `max_epoch`, we decrease the learning rate by multipling `lr_decay`.

```
new_lr_decay = lr_decay ** max(i - max_epoch, 0.0)
sess.run(tf.assign(lr, learning_rate * new_lr_decay))
```

At the beginning of each epoch, all states of LSTMs need to be reseted (initialized) to zero states. Then after each iteration, the LSTMs' states is updated, so the new LSTM states (final states) need to be assigned as the initial states of the next iteration:

```
# set all states to zero states at the beginning of each epoch
state1 = tl.layers.initialize_rnn_state(lstm1.initial_state)
state2 = tl.layers.initialize_rnn_state(lstm2.initial_state)
for step, (x, y) in enumerate(tl.iterate.ptb_iterator(train_data,
                                                      batch_size, num_steps)):
    feed_dict = {input_data: x, targets: y,
                 lstm1.initial_state: state1,
                 lstm2.initial_state: state2,
                 }
    # For training, enable dropout
    feed_dict.update( network.all_drop )
    # use the new states as the initial state of next iteration
    _cost, state1, state2, _ = sess.run([cost,
                                         lstm1.final_state,
                                         lstm2.final_state,
                                         train_op],
                                         feed_dict=feed_dict
                                         )
    costs += _cost; iters += num_steps
```

Predicting

After training the model, when we predict the next output, we no long consider the number of steps (sequence length), i.e. `batch_size, num_steps` are set to `1`. Then we can output the next word one by one, instead of predicting a sequence of words from a sequence of words.

```
input_data_test = tf.placeholder(tf.int32, [1, 1])
targets_test = tf.placeholder(tf.int32, [1, 1])
...
network_test, lstm1_test, lstm2_test = inference(input_data_test,
                        is_training=False, num_steps=1, reuse=True)
...
cost_test = loss_fn(network_test.outputs, targets_test, 1, 1)
...
print("Evaluation")
# Testing
# go through the test set step by step, it will take a while.
start_time = time.time()
costs = 0.0; iters = 0
# reset all states at the beginning
state1 = tl.layers.initialize_rnn_state(lstm1_test.initial_state)
state2 = tl.layers.initialize_rnn_state(lstm2_test.initial_state)
for step, (x, y) in enumerate(tl.iterate.ptb_iterator(test_data,
                                          batch_size=1, num_steps=1)):
    feed_dict = {input_data_test: x, targets_test: y,
                 lstm1_test.initial_state: state1,
                 lstm2_test.initial_state: state2,
                 }
    _cost, state1, state2 = sess.run([cost_test,
                                      lstm1_test.final_state,
                                      lstm2_test.final_state],
                                      feed_dict=feed_dict
```

(continues on next page)

(continued from previous page)

```
                                        )
    costs += _cost; iters += 1
test_perplexity = np.exp(costs / iters)
print("Test Perplexity: %.3f took %.2fs" % (test_perplexity, time.time() - start_
↪time))
```

What Next?

Now, you have understood Synced sequence input and output. Let's think about Many to one (Sequence input and one output), so that LSTM is able to predict the next word "English" from "I am from London, I speak ..".

Please read and understand the code of `tutorial_generate_text.py`. It shows you how to restore a pre-trained Embedding matrix and how to learn text generation from a given context.

Karpathy's blog : "(3) Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment). "

1.2.11 More Tutorials

In Example page, we provide many examples include Seq2seq, different type of Adversarial Learning, Reinforcement Learning and etc.

1.2.12 More info

For more information on what you can do with TensorLayer, just continue reading through readthedocs. Finally, the reference lists and explains as follow.

layers (*tensorlayer.layers*),

activation (*tensorlayer.activation*),

natural language processing (*tensorlayer.nlp*),

reinforcement learning (*tensorlayer.rein*),

cost expressions and regularizers (*tensorlayer.cost*),

load and save files (*tensorlayer.files*),

operating system (*tensorlayer.ops*),

helper functions (*tensorlayer.utils*),

visualization (*tensorlayer.visualize*),

iteration functions (*tensorlayer.iterate*),

preprocessing functions (*tensorlayer.prepro*),

1.3 Example

1.3.1 Basics

- Multi-layer perceptron (MNIST). Classification task, see tutorial_mnist_simple.py.

- Multi-layer perceptron (MNIST). Classification using Iterator, see method1 and method2.

1.3.2 Computer Vision

- Denoising Autoencoder (MNIST). Classification task, see tutorial_mnist.py.
- Stacked Denoising Autoencoder and Fine-Tuning (MNIST). A MLP classification task, see tutorial_mnist.py.
- Convolutional Network (MNIST). Classification task, see tutorial_mnist.py.
- Convolutional Network (CIFAR-10). Classification task, see tutorial_cifar10.py and tutorial_cifar10_tfrecord.py.
- VGG 16 (ImageNet). Classification task, see tutorial_vgg16.py.
- VGG 19 (ImageNet). Classification task, see tutorial_vgg19.py.
- InceptionV3 (ImageNet). Classification task, see tutorial_inceptionV3_tfslim.py.
- Wide ResNet (CIFAR) by ritchieng.
- More CNN implementations of TF-Slim can be connected to TensorLayer via SlimNetsLayer.
- Spatial Transformer Networks by zsdonghao.
- U-Net for brain tumor segmentation by zsdonghao.
- Variational Autoencoder (VAE) for (CelebA) by yzwxx.
- Variational Autoencoder (VAE) for (MNIST) by BUPTLdy.
- Image Captioning - Reimplementation of Google's im2txt by zsdonghao.

1.3.3 Natural Language Processing

- Recurrent Neural Network (LSTM). Apply multiple LSTM to PTB dataset for language modeling, see tutorial_ptb_lstm_state_is_tuple.py.
- Word Embedding (Word2vec). Train a word embedding matrix, see tutorial_word2vec_basic.py.
- Restore Embedding matrix. Restore a pre-train embedding matrix, see tutorial_generate_text.py.
- Text Generation. Generates new text scripts, using LSTM network, see tutorial_generate_text.py.
- Chinese Text Anti-Spam by pakrchen.
- Chatbot in 200 lines of code for Seq2Seq.
- FastText Sentence Classification (IMDB), see tutorial_imdb_fasttext.py by tomtung.

1.3.4 Adversarial Learning

- DCGAN (CelebA). Generating images by Deep Convolutional Generative Adversarial Networks by zsdonghao.
- Generative Adversarial Text to Image Synthesis by zsdonghao.
- Unsupervised Image to Image Translation with Generative Adversarial Networks by zsdonghao.
- Improved CycleGAN with resize-convolution by luoxier.
- Super Resolution GAN by zsdonghao.
- DAGAN: Fast Compressed Sensing MRI Reconstruction by nebulaV.

1.3.5 Reinforcement Learning

- Policy Gradient / Network (Atari Ping Pong), see tutorial_atari_pong.py.
- Deep Q-Network (Frozen lake), see tutorial_frozenlake_dqn.py.
- Q-Table learning algorithm (Frozen lake), see tutorial_frozenlake_q_table.py.
- Asynchronous Policy Gradient using TensorDB (Atari Ping Pong) by nebulaV.
- AC for discrete action space (Cartpole), see tutorial_cartpole_ac.py.
- A3C for continuous action space (Bipedal Walker), see tutorial_bipedalwalker_a3c*.py.
- DAGGER for (Gym Torcs) by zsdonghao.
- TRPO for continuous and discrete action space by jjkke88.

1.3.6 Special Examples

- Distributed Training. tutorial_mnist_distributed.py by jorgemf.
- Merge TF-Slim into TensorLayer. tutorial_inceptionV3_tfslim.py.
- Merge Keras into TensorLayer. tutorial_keras.py.
- Data augmentation with TFRecord. Effective way to load and pre-process data, see tutorial_tfrecord*.py and tutorial_cifar10_tfrecord.py.
- Data augmentation with TensorLayer, see tutorial_image_preprocess.py.
- TensorDB by fangde see here.
- A simple web service - TensorFlask by JoelKronander.
- Float 16 half-precision model, see tutorial_mnist_float16.py.

1.4 Development

TensorLayer is a major ongoing research project in Data Science Institute, Imperial College London. The goal of the project is to develop a compositional language while complex learning systems can be build through composition of neural network modules. The whole development is now participated by numerous contributors on Release. As an open-source project by we highly welcome contributions! Every bit helps and will be credited.

1.4.1 What to contribute

Your method and example

If you have a new method or example in term of Deep learning and Reinforcement learning, you are welcome to contribute.

- Provide your layer or example, so everyone can use it.
- Explain how it would work, and link to a scientific paper if applicable.
- Keep the scope as narrow as possible, to make it easier to implement.

Report bugs

Report bugs at the GitHub, we normally will fix it in 5 hours. If you are reporting a bug, please include:

- your TensorLayer, TensorFlow and Python version.
- steps to reproduce the bug, ideally reduced to a few Python commands.
- the results you obtain, and the results you expected instead.

If you are unsure whether the behavior you experience is a bug, or if you are unsure whether it is related to TensorLayer or TensorFlow, please just ask on our mailing list first.

Fix bugs

Look through the GitHub issues for bug reports. Anything tagged with "bug" is open to whoever wants to implement it. If you discover a bug in TensorLayer you can fix yourself, by all means feel free to just implement a fix and not report it first.

Write documentation

Whenever you find something not explained well, misleading, glossed over or just wrong, please update it! The *Edit on GitHub* link on the top right of every documentation page and the *[source]* link for every documented entity in the API reference will help you to quickly locate the origin of any text.

1.4.2 How to contribute

Edit on GitHub

As a very easy way of just fixing issues in the documentation, use the *Edit on GitHub* link on the top right of a documentation page or the *[source]* link of an entity in the API reference to open the corresponding source file in GitHub, then click the *Edit this file* link to edit the file in your browser and send us a Pull Request. All you need for this is a free GitHub account.

For any more substantial changes, please follow the steps below to setup TensorLayer for development.

Documentation

The documentation is generated with Sphinx. To build it locally, run the following commands:

```
pip install Sphinx
sphinx-quickstart

cd docs
make html
```

If you want to re-generate the whole docs, run the following commands:

```
cd docs
make clean
make html
```

To write the docs, we recommend you to install Local RTD VM.

Afterwards, open `docs/_build/html/index.html` to view the documentation as it would appear on readthedocs. If you changed a lot and seem to get misleading error messages or warnings, run `make clean html` to force Sphinx to recreate all files from scratch.

When writing docstrings, follow existing documentation as much as possible to ensure consistency throughout the library. For additional information on the syntax and conventions used, please refer to the following documents:

- reStructuredText Primer
- Sphinx reST markup constructs
- A Guide to NumPy/SciPy Documentation

Testing

TensorLayer has a code coverage of 100%, which has proven very helpful in the past, but also creates some duties:

- Whenever you change any code, you should test whether it breaks existing features by just running the test scripts.
- Every bug you fix indicates a missing test case, so a proposed bug fix should come with a new test that fails without your fix.

Sending Pull Requests

When you're satisfied with your addition, the tests pass and the documentation looks good without any markup errors, commit your changes to a new branch, push that branch to your fork and send us a Pull Request via GitHub's web interface.

All these steps are nicely explained on GitHub: https://guides.github.com/introduction/flow/

When filing your Pull Request, please include a description of what it does, to help us reviewing it. If it is fixing an open issue, say, issue #123, add *Fixes #123*, *Resolves #123* or *Closes #123* to the description text, so GitHub will close it when your request is merged.

1.5 More

1.5.1 FQA

How to effectively learn TensorLayer

No matter what stage you are in, we recommend you to spend just 10 minutes to read the source code of TensorLayer and the Understand layer / Your layer in this website, you will find the abstract methods are very simple for everyone. Reading the source codes helps you to better understand TensorFlow and allows you to implement your own methods easily. For discussion, we recommend Gitter, Help Wanted Issues, QQ group and Wechat group.

Beginner

For people who new to deep learning, the contirbutors provided a number of tutorials in this website, these tutorials will guide you to understand autoencoder, convolutional neural network, recurrent neural network, word embedding and deep reinforcement learning and etc. If your already understand the basic of deep learning, we recommend you to skip the tutorials and read the example codes on Github , then implement an example from scratch.

Engineer

For people from industry, the contirbutors provided mass format-consistent examples covering computer vision, natural language processing and reinforcement learning. Besides, there are also many TensorFlow users already implemented product-level examples including image captioning, semantic/instance segmentation, machine translation, chatbot and etc, which can be found online. It is worth noting that a wrapper especially for computer vision Tf-Slim can be connected with TensorLayer seamlessly. Therefore, you may able to find the examples that can be used in your project.

Researcher

For people from academic, TensorLayer was originally developed by PhD students who facing issues with other libraries on implement novel algorithm. Installing TensorLayer in editable mode is recommended, so you can extend your methods in TensorLayer. For researches related to image such as image captioning, visual QA and etc, you may find it is very helpful to use the existing Tf-Slim pre-trained models with TensorLayer (a specially layer for connecting Tf-Slim is provided).

Exclude some layers from training

You may need to get the list of variables you want to update, TensorLayer provides two ways to get the variables list.

The first way is to use the all_params of a network, by default, it will store the variables in order. You can print the variables information via `tl.layers.print_all_variables(train_only=True)` or `network.print_params(details=False)`. To choose which variables to update, you can do as below.

```
train_params = network.all_params[3:]
```

The second way is to get the variables by a given name. For example, if you want to get all variables which the layer name contain `dense`, you can do as below.

```
train_params = tl.layers.get_variables_with_name('dense', train_only=True,␣
↪printable=True)
```

After you get the variable list, you can define your optimizer like that so as to update only a part of the variables.

```
train_op = tf.train.AdamOptimizer(0.001).minimize(cost, var_list= train_params)
```

Visualization

Cannot Save Image

If you run the script via SSH control, sometime you may find the following error.

```
_tkinter.TclError: no display name and no $DISPLAY environment variable
```

If happen, use `import matplotlib` and `matplotlib.use('Agg')` before `import tensorlayer as tl`. Alternatively, add the following code into the top of `visualize.py` or in your own code.

```
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
```

Install Master Version

To use all new features of TensorLayer, you need to install the master version from Github. Before that, you need to make sure you already installed git.

```
[stable version] pip install tensorlayer
[master version] pip install git+https://github.com/zsdonghao/tensorlayer.git
```

Editable Mode

- 1. Download the TensorLayer folder from Github.
- 2. Before editing the TensorLayer `.py` file.
  - If your script and TensorLayer folder are in the same folder, when you edit the `.py` inside TensorLayer folder, your script can access the new features.
  - If your script and TensorLayer folder are not in the same folder, you need to run the following command in the folder contains `setup.py` before you edit `.py` inside TensorLayer folder.

```
pip install -e .
```

Load Model

Note that, the `tl.files.load_npz()` can only able to load the npz model saved by `tl.files.save_npz()`. If you have a model want to load into your TensorLayer network, you can first assign your parameters into a list in order, then use `tl.files.assign_params()` to load the parameters into your TensorLayer model.

1.5.2 Recruitment

TensorLayer Contributors

TensorLayer contributors are from Imperial College, Tsinghua University, Carnegie Mellon University, Google, Microsoft, Bloomberg and etc. There are many functions need to be contributed such as Maxout, Neural Turing Machine, Attention, TensorLayer Mobile and etc. Please push on GitHub, every bit helps and will be credited. If you are interested in working with us, please contact us.

Data Science Institute, Imperial College London

Data science is therefore by nature at the core of all modern transdisciplinary scientific activities, as it involves the whole life cycle of data, from acquisition and exploration to analysis and communication of the results. Data science is not only concerned with the tools and methods to obtain, manage and analyse data: it is also about extracting value from data and translating it from asset to product.

Launched on 1st April 2014, the Data Science Institute at Imperial College London aims to enhance Imperial's excellence in data-driven research across its faculties by fulfilling the following objectives.

The Data Science Institute is housed in purpose built facilities in the heart of the Imperial College campus in South Kensington. Such a central location provides excellent access to collabroators across the College and across London.

- To act as a focal point for coordinating data science research at Imperial College by facilitating access to funding, engaging with global partners, and stimulating cross-disciplinary collaboration.

- To develop data management and analysis technologies and services for supporting data driven research in the College.
- To promote the training and education of the new generation of data scientist by developing and coordinating new degree courses, and conducting public outreach programmes on data science.
- To advise College on data strategy and policy by providing world-class data science expertise.
- To enable the translation of data science innovation by close collaboration with industry and supporting commercialization.

If you are interested in working with us, please check our vacancies and other ways to get involved , or feel free to contact us.

CHAPTER 2

API Reference

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

2.1 API - Layers

To make TensorLayer simple, we minimize the number of layer classes as much as we can. So we encourage you to use TensorFlow's function. For example, we provide layer for local response normalization, but user can still apply `tf.nn.lrn` on `network.outputs`. More functions can be found in TensorFlow API.

2.1.1 Understand Basic layer

All TensorLayer layers have a number of properties in common:

- `layer.outputs` : a Tensor, the outputs of current layer.
- `layer.all_params` : a list of Tensor, all network variables in order.
- `layer.all_layers` : a list of Tensor, all network outputs in order.
- `layer.all_drop` : a dictionary of {placeholder : float}, all keeping probabilities of noise layer.

All TensorLayer layers have a number of methods in common:

- `layer.print_params()` : print the network variables information in order (after `tl.layers.initialize_global_variables(sess)`). alternatively, print all variables by `tl.layers.print_all_variables()`.
- `layer.print_layers()` : print the network layers information in order.
- `layer.count_params()` : print the number of parameters in the network.

The initialization of a network is done by input layer, then we can stacked layers as follow, a network is a `Layer` class. The most important properties of a network are `network.all_params`, `network.all_layers` and `network.all_drop`. The `all_params` is a list which store all pointers of all network parameters in order, the following script define a 3 layer network, then:

`all_params` = [W1, b1, W2, b2, W_out, b_out]

To get specified variables, you can use `network.all_params[2:3]` or `get_variables_with_name()`. As the `all_layers` is a list which store all pointers of the outputs of all layers, in the following network:

`all_layers` = [drop(?,784), relu(?,800), drop(?,800), relu(?,800), drop(?,800)], identity(?,10)]

where `?` reflects any batch size. You can print the layer information and parameters information by using `network.print_layers()` and `network.print_params()`. To count the number of parameters in a network, run `network.count_params()`.

```
sess = tf.InteractiveSession()

x = tf.placeholder(tf.float32, shape=[None, 784], name='x')
y_ = tf.placeholder(tf.int64, shape=[None, ], name='y_')

network = tl.layers.InputLayer(x, name='input_layer')
network = tl.layers.DropoutLayer(network, keep=0.8, name='drop1')
network = tl.layers.DenseLayer(network, n_units=800,
                                act = tf.nn.relu, name='relu1')
network = tl.layers.DropoutLayer(network, keep=0.5, name='drop2')
network = tl.layers.DenseLayer(network, n_units=800,
                                act = tf.nn.relu, name='relu2')
network = tl.layers.DropoutLayer(network, keep=0.5, name='drop3')
network = tl.layers.DenseLayer(network, n_units=10,
                                act = tl.activation.identity,
                                name='output_layer')

y = network.outputs
y_op = tf.argmax(tf.nn.softmax(y), 1)

cost = tl.cost.cross_entropy(y, y_)

train_params = network.all_params

train_op = tf.train.AdamOptimizer(learning_rate, beta1=0.9, beta2=0.999,
                            epsilon=1e-08, use_locking=False).minimize(cost, var_list␣
↪= train_params)

tl.layers.initialize_global_variables(sess)

network.print_params()
network.print_layers()
```

In addition, `network.all_drop` is a dictionary which stores the keeping probabilities of all noise layer. In the above network, they are the keeping probabilities of dropout layers.

So for training, enable all dropout layers as follow.

```
feed_dict = {x: X_train_a, y_: y_train_a}
feed_dict.update( network.all_drop )
loss, _ = sess.run([cost, train_op], feed_dict=feed_dict)
feed_dict.update( network.all_drop )
```

For evaluating and testing, disable all dropout layers as follow.

```
feed_dict = {x: X_val, y_: y_val}
feed_dict.update(dp_dict)
print("   val loss: %f" % sess.run(cost, feed_dict=feed_dict))
```

(continues on next page)

(continued from previous page)

```
print("   val acc: %f" % np.mean(y_val ==
                        sess.run(y_op, feed_dict=feed_dict)))
```

For more details, please read the MNIST examples on Github.

2.1.2 Customized layer

A Simple layer

To implement a custom layer in TensorLayer, you will have to write a Python class that subclasses Layer and implement the `outputs` expression.

The following is an example implementation of a layer that multiplies its input by 2:

```
class DoubleLayer(Layer):
    def __init__(
        self,
        layer = None,
        name ='double_layer',
    ):
        # check layer name (fixed)
        Layer.__init__(self, name=name)

        # the input of this layer is the output of previous layer (fixed)
        self.inputs = layer.outputs

        # operation (customized)
        self.outputs = self.inputs * 2

        # get stuff from previous layer (fixed)
        self.all_layers = list(layer.all_layers)
        self.all_params = list(layer.all_params)
        self.all_drop = dict(layer.all_drop)

        # update layer (customized)
        self.all_layers.extend( [self.outputs] )
```

Your Dense layer

Before creating your own TensorLayer layer, let's have a look at Dense layer. It creates a weights matrix and biases vector if not exists, then implement the output expression. At the end, as a layer with parameter, we also need to append the parameters into `all_params`.

```
class MyDenseLayer(Layer):
  def __init__(
      self,
      layer = None,
      n_units = 100,
      act = tf.nn.relu,
      name ='simple_dense',
  ):
      # check layer name (fixed)
      Layer.__init__(self, name=name)
```

(continues on next page)

(continued from previous page)

```
        # the input of this layer is the output of previous layer (fixed)
        self.inputs = layer.outputs

        # print out info (customized)
        print("  MyDenseLayer %s: %d, %s" % (self.name, n_units, act))

        # operation (customized)
        n_in = int(self.inputs._shape[-1])
        with tf.variable_scope(name) as vs:
            # create new parameters
            W = tf.get_variable(name='W', shape=(n_in, n_units))
            b = tf.get_variable(name='b', shape=(n_units))
            # tensor operation
            self.outputs = act(tf.matmul(self.inputs, W) + b)

        # get stuff from previous layer (fixed)
        self.all_layers = list(layer.all_layers)
        self.all_params = list(layer.all_params)
        self.all_drop = dict(layer.all_drop)

        # update layer (customized)
        self.all_layers.extend( [self.outputs] )
        self.all_params.extend( [W, b] )
```

Modifying Pre-train Behaviour

Greedy layer-wise pretraining is an important task for deep neural network initialization, while there are many kinds of pre-training methods according to different network architectures and applications.

For example, the pre-train process of Vanilla Sparse Autoencoder can be implemented by using KL divergence (for sigmoid) as the following code, but for Deep Rectifier Network, the sparsity can be implemented by using the L1 regularization of activation output.

```
# Vanilla Sparse Autoencoder
beta = 4
rho = 0.15
p_hat = tf.reduce_mean(activation_out, reduction_indices = 0)
KLD = beta * tf.reduce_sum( rho * tf.log(tf.div(rho, p_hat))
        + (1- rho) * tf.log((1- rho)/ (tf.sub(float(1), p_hat))) )
```

There are many pre-train methods, for this reason, TensorLayer provides a simple way to modify or design your own pre-train method. For Autoencoder, TensorLayer uses `ReconLayer.__init__()` to define the reconstruction layer and cost function, to define your own cost function, just simply modify the `self.cost` in `ReconLayer.__init__()`. To creat your own cost expression please read Tensorflow Math. By default, `ReconLayer` only updates the weights and biases of previous 1 layer by using `self.train_params = self.all _params[-4:]`, where the 4 parameters are `[W_encoder, b_encoder, W_decoder, b_decoder]`, where `W_encoder, b_encoder` belong to previous DenseLayer, `W_decoder, b_decoder` belong to this ReconLayer. In addition, if you want to update the parameters of previous 2 layers at the same time, simply modify `[-4:]` to `[-6:]`.

```
ReconLayer.__init__(...):
    ...
    self.train_params = self.all_params[-4:]
```

(continues on next page)

(continued from previous page)

```
    ...
    self.cost = mse + L1_a + L2_w
```

2.1.3 Layer list

get_variables_with_name(name[, train_only, ...])	Get variable list by a given name scope.
get_layers_with_name([network, name, printable])	Get layer list in a network by a given name scope.
set_name_reuse([enable])	Enable or disable reuse layer name.
print_all_variables([train_only])	Print all trainable and non-trainable variables without tl.layers.initialize_global_variables(sess)
initialize_global_variables([sess])	Excute `sess.run(tf.global_variables_initializer())` for TF 0.12+ or `sess.run(tf.initialize_all_variables())` for TF 0.11.
Layer([inputs, name])	The *Layer* class represents a single layer of a neural network.
InputLayer([inputs, name])	The *InputLayer* class is the starting layer of a neural network.
OneHotInputLayer([inputs, depth, on_value, ...])	The *OneHotInputLayer* class is the starting layer of a neural network, see `tf.one_hot`.
Word2vecEmbeddingInputlayer([inputs, ...])	The *Word2vecEmbeddingInputlayer* class is a fully connected layer, for Word Embedding.
EmbeddingInputlayer([inputs, ...])	The *EmbeddingInputlayer* class is a fully connected layer, for Word Embedding.
AverageEmbeddingInputlayer(inputs, ...[, ...])	The *AverageEmbeddingInputlayer* averages over embeddings of inputs, can be used as the input layer for models like DAN[1] and FastText[2].
DenseLayer([layer, n_units, act, W_init, ...])	The *DenseLayer* class is a fully connected layer.
ReconLayer([layer, x_recon, name, n_units, act])	The *ReconLayer* class is a reconstruction layer *DenseLayer* which use to pre-train a *DenseLayer*.
DropoutLayer([layer, keep, is_fix, ...])	The *DropoutLayer* class is a noise layer which randomly set some values to zero by a given keeping probability.
GaussianNoiseLayer([layer, mean, stddev, ...])	The *GaussianNoiseLayer* class is noise layer that adding noise with normal distribution to the activation.
DropconnectDenseLayer([layer, keep, ...])	The *DropconnectDenseLayer* class is `DenseLayer` with DropConnect behaviour which randomly remove connection between this layer to previous layer by a given keeping probability.
Conv1dLayer([layer, act, shape, stride, ...])	The *Conv1dLayer* class is a 1D CNN layer, see tf.nn.convolution.
Conv2dLayer([layer, act, shape, strides, ...])	The *Conv2dLayer* class is a 2D CNN layer, see tf.nn.conv2d.
DeConv2dLayer([layer, act, shape, ...])	The *DeConv2dLayer* class is deconvolutional 2D layer, see tf.nn.conv2d_transpose.

Continued on next page

Table 1 – continued from previous page

Conv3dLayer([layer, act, shape, strides, ...])	The Conv3dLayer class is a 3D CNN layer, see tf.nn.conv3d.
DeConv3dLayer([layer, act, shape, ...])	The DeConv3dLayer class is deconvolutional 3D layer, see tf.nn.conv3d_transpose.
PoolLayer([layer, ksize, strides, padding, ...])	The PoolLayer class is a Pooling layer, you can choose tf.nn.max_pool and tf.nn.avg_pool for 2D or tf.nn.max_pool3d and tf.nn.avg_pool3d for 3D.
PadLayer([layer, paddings, mode, name])	The PadLayer class is a Padding layer for any modes and dimensions.
UpSampling2dLayer([layer, size, is_scale, ...])	The UpSampling2dLayer class is upSampling 2d layer, see tf.image.resize_images.
DownSampling2dLayer([layer, size, is_scale, ...])	The DownSampling2dLayer class is downSampling 2d layer, see tf.image.resize_images.
DeformableConv2dLayer([layer, act, ...])	The DeformableConv2dLayer class is a Deformable Convolutional Networks .
AtrousConv1dLayer(net[, n_filter, ...])	Wrapper for AtrousConv1dLayer, if you don't understand how to use Conv1dLayer, this function may be easier.
AtrousConv2dLayer([layer, n_filter, ...])	The AtrousConv2dLayer class is Atrous convolution (a.k.a.
Conv1d(net[, n_filter, filter_size, stride, ...])	Wrapper for Conv1dLayer, if you don't understand how to use Conv1dLayer, this function may be easier.
Conv2d(net[, n_filter, filter_size, ...])	Wrapper for Conv2dLayer, if you don't understand how to use Conv2dLayer, this function may be easier.
DeConv2d(net[, n_out_channel, filter_size, ...])	Wrapper for DeConv2dLayer, if you don't understand how to use DeConv2dLayer, this function may be easier.
MaxPool1d(net, filter_size, strides[, ...])	Wrapper for tf.layers.max_pooling1d .
MeanPool1d(net, filter_size, strides[, ...])	Wrapper for tf.layers.average_pooling1d .
MaxPool2d(net[, filter_size, strides, ...])	Wrapper for PoolLayer.
MeanPool2d(net[, filter_size, strides, ...])	Wrapper for PoolLayer.
MaxPool3d(net, filter_size, strides[, ...])	Wrapper for tf.layers.max_pooling3d .
MeanPool3d(net, filter_size, strides[, ...])	Wrapper for tf.layers.average_pooling3d
DepthwiseConv2d([layer, channel_multiplier, ...])	Separable/Depthwise Convolutional 2D, see tf.nn.depthwise_conv2d.
SubpixelConv1d(net[, scale, act, name])	One-dimensional subpixel upsampling layer.
SubpixelConv2d(net[, scale, n_out_channel, ...])	It is a sub-pixel 2d upsampling layer, usually be used for Super-Resolution applications, see example code.
SpatialTransformer2dAffineLayer([layer, ...])	The SpatialTransformer2dAffineLayer class is a Spatial Transformer Layer for 2D Affine Transformation.
transformer(U, theta, out_size[, name])	Spatial Transformer Layer for 2D Affine Transformation , see SpatialTransformer2dAffineLayer class.
batch_transformer(U, thetas, out_size[, name])	Batch Spatial Transformer function for 2D Affine Transformation.

Continued on next page

Table 1 – continued from previous page

BatchNormLayer([layer, decay, epsilon, act, ...])	The BatchNormLayer class is a normalization layer, see tf.nn.batch_normalization and tf.nn.moments.
LocalResponseNormLayer([layer, ...])	The LocalResponseNormLayer class is for Local Response Normalization, see tf.nn.local_response_normalization or tf.nn.lrn for new TF version.
InstanceNormLayer([layer, act, epsilon, ...])	The InstanceNormLayer class is a for instance normalization.
LayerNormLayer([layer, center, scale, act, ...])	The LayerNormLayer class is for layer normalization, see tf.contrib.layers.layer_norm.
ROIPoolingLayer([layer, rois, pool_height, ...])	The ROIPoolingLayer class is Region of interest pooling layer.
TimeDistributedLayer([layer, layer_class, ...])	The TimeDistributedLayer class that applies a function to every timestep of the input tensor.
RNNLayer([layer, cell_fn, cell_init_args, ...])	The RNNLayer class is a RNN layer, you can implement vanilla RNN, LSTM and GRU with it.
BiRNNLayer([layer, cell_fn, cell_init_args, ...])	The BiRNNLayer class is a Bidirectional RNN layer.
ConvRNNCell	Abstract object representing an Convolutional RNN Cell.
BasicConvLSTMCell(shape, filter_size, ... [, ...])	Basic Conv LSTM recurrent network cell.
ConvLSTMLayer([layer, cell_shape, ...])	The ConvLSTMLayer class is a Convolutional LSTM layer, see Convolutional LSTM Layer .
advanced_indexing_op(input, index)	Advanced Indexing for Sequences, returns the outputs by given sequence lengths.
retrieve_seq_length_op(data)	An op to compute the length of a sequence from input shape of [batch_size, n_step(max), n_features], it can be used when the features of padding (on right hand side) are all zeros.
retrieve_seq_length_op2(data)	An op to compute the length of a sequence, from input shape of [batch_size, n_step(max)], it can be used when the features of padding (on right hand side) are all zeros.
DynamicRNNLayer([layer, cell_fn, ...])	The DynamicRNNLayer class is a Dynamic RNN layer, see tf.nn.dynamic_rnn.
BiDynamicRNNLayer([layer, cell_fn, ...])	The BiDynamicRNNLayer class is a RNN layer, you can implement vanilla RNN, LSTM and GRU with it.
Seq2Seq([net_encode_in, net_decode_in, ...])	The Seq2Seq class is a Simple DynamicRNNLayer based Seq2seq layer without using tl.contrib.seq2seq.
PeekySeq2Seq([net_encode_in, net_decode_in, ...])	Waiting for contribution.
AttentionSeq2Seq([net_encode_in, ...])	Waiting for contribution.
FlattenLayer([layer, name])	The FlattenLayer class is layer which reshape high-dimension input to a vector.
ReshapeLayer([layer, shape, name])	The ReshapeLayer class is layer which reshape the tensor.
TransposeLayer([layer, perm, name])	The TransposeLayer class transpose the dimension of a teneor, see tf.transpose() .
LambdaLayer([layer, fn, fn_args, name])	The LambdaLayer class is a layer which is able to use the provided function.
ConcatLayer([layer, concat_dim, name])	The ConcatLayer class is layer which concat (merge) two or more tensor by given axis..

Continued on next page

Table 1 – continued from previous page

ElementwiseLayer([layer, combine_fn, name])	The *ElementwiseLayer* class combines multiple *Layer* which have the same output shapes by a given elemwise-wise operation.
ExpandDimsLayer([layer, axis, name])	The *ExpandDimsLayer* class inserts a dimension of 1 into a tensor's shape, see tf.expand_dims() .
TileLayer([layer, multiples, name])	The *TileLayer* class constructs a tensor by tiling a given tensor, see tf.tile() .
StackLayer([layer, axis, name])	The *StackLayer* class is layer for stacking a list of rank-R tensors into one rank-(R+1) tensor, see tf.stack().
UnStackLayer([layer, num, axis, name])	The *UnStackLayer* is layer for unstacking the given dimension of a rank-R tensor into rank-(R-1) tensors., see tf.unstack().
EstimatorLayer([layer, model_fn, args, name])	The *EstimatorLayer* class accepts `model_fn` that described the model.
SlimNetsLayer([layer, slim_layer, ...])	The *SlimNetsLayer* class can be used to merge all TF-Slim nets into TensorLayer.
KerasLayer([layer, keras_layer, keras_args, ...])	The *KerasLayer* class can be used to merge all Keras layers into TensorLayer.
PReluLayer([layer, channel_shared, a_init, ...])	The *PReluLayer* class is Parametric Rectified Linear layer.
MultiplexerLayer([layer, name])	The *MultiplexerLayer* selects one of several input and forwards the selected input into the output, see *tutorial_mnist_multiplexer.py*.
EmbeddingAttentionSeq2seqWrapper(..., [, ...])	Sequence-to-sequence model with attention and for multiple buckets (Deprecated after TF0.12).
flatten_reshape(variable[, name])	Reshapes high-dimension input to a vector.
clear_layers_name()	Clear all layer names in set_keep['_layers_name_list'], enable layer name reuse.
initialize_rnn_state(state[, feed_dict])	Returns the initialized RNN state.
list_remove_repeat([l])	Remove the repeated items in a list, and return the processed list.
merge_networks([layers])	Merge all parameters, layers and dropout probabilities to a *Layer*.

2.1.4 Name Scope and Sharing Parameters

These functions help you to reuse parameters for different inference (graph), and get a list of parameters by given name. About TensorFlow parameters sharing click here.

Get variables with name

tensorlayer.layers.**get_variables_with_name**(*name*, *train_only=True*, *printable=False*)

Get variable list by a given name scope.

Examples

```
>>> dense_vars = tl.layers.get_variable_with_name('dense', True, True)
```

Get layers with name

tensorlayer.layers.**get_layers_with_name**(*network=None*, *name=''*, *printable=False*)
Get layer list in a network by a given name scope.

Examples

```
>>> layers = tl.layers.get_layers_with_name(network, "CNN", True)
```

Enable layer name reuse

tensorlayer.layers.**set_name_reuse**(*enable=True*)
Enable or disable reuse layer name. By default, each layer must has unique name. When you want two or more input placeholder (inference) share the same model parameters, you need to enable layer name reuse, then allow the parameters have same name scope.

Parameters

enable [boolean, enable name reuse. (None means False).]

Examples

```
>>> def embed_seq(input_seqs, is_train, reuse):
>>>     with tf.variable_scope("model", reuse=reuse):
>>>         tl.layers.set_name_reuse(reuse)
>>>         network = tl.layers.EmbeddingInputlayer(
...                     inputs = input_seqs,
...                     vocabulary_size = vocab_size,
...                     embedding_size = embedding_size,
...                     name = 'e_embedding')
>>>         network = tl.layers.DynamicRNNLayer(network,
...                     cell_fn = tf.contrib.rnn.BasicLSTMCell,
...                     n_hidden = embedding_size,
...                     dropout = (0.7 if is_train else None),
...                     initializer = w_init,
...                     sequence_length = tl.layers.retrieve_seq_length_op2(input_
↪seqs),
...                     return_last = True,
...                     name = 'e_dynamicrnn')
>>>     return network
>>>
>>> net_train = embed_seq(t_caption, is_train=True, reuse=False)
>>> net_test = embed_seq(t_caption, is_train=False, reuse=True)
```

- see `tutorial_ptb_lstm.py` for example.

Print variables

tensorlayer.layers.**print_all_variables**(*train_only=False*)
Print all trainable and non-trainable variables without tl.layers.initialize_global_variables(sess)

Parameters

train_only [boolean] If True, only print the trainable variables, otherwise, print all variables.

Initialize variables

tensorlayer.layers.**initialize_global_variables**(*sess=None*)

Excute `sess.run(tf.global_variables_initializer())` for TF 0.12+ or `sess.run(tf.initialize_all_variables())` for TF 0.11.

Parameters

sess [a Session]

2.1.5 Basic layer

class tensorlayer.layers.**Layer**(*inputs=None*, *name='layer'*)

The *Layer* class represents a single layer of a neural network. It should be subclassed when implementing new types of layers. Because each layer can keep track of the layer(s) feeding into it, a network's output *Layer* instance can double as a handle to the full network.

Parameters

inputs [a *Layer* instance] The *Layer* class feeding into this layer.

name [a string or None] An optional name to attach to this layer.

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

2.1.6 Input layer

class tensorlayer.layers.**InputLayer**(*inputs=None*, *name='input_layer'*)

The *InputLayer* class is the starting layer of a neural network.

Parameters

inputs [a placeholder or tensor] The input tensor data.

name [a string or None] An optional name to attach to this layer.

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

2.1.7 One-hot layer

class tensorlayer.layers.**OneHotInputLayer**(*inputs=None*, *depth=None*, *on_value=None*, *off_value=None*, *axis=None*, *dtype=None*, *name='input_layer'*)

The *OneHotInputLayer* class is the starting layer of a neural network, see `tf.one_hot`.

Parameters

inputs [a placeholder or tensor] The input tensor data.

name [a string or None] An optional name to attach to this layer.

depth [If the input indices is rank N, the output will have rank N+1. The new axis is created at dimension axis (default: the new axis is appended at the end).]

on_value [If on_value is not provided, it will default to the value 1 with type dtype.] default, None

off_value [If off_value is not provided, it will default to the value 0 with type dtype.] default, None

axis [default, None]

dtype [default, None]

Methods

`count_params`()	Return the number of parameters in the network
`print_layers`()	Print all info of layers in the network
`print_params`([details, session])	Print all info of parameters in the network

2.1.8 Word Embedding Input layer

Word2vec layer for training

class tensorlayer.layers.**Word2vecEmbeddingInputlayer**(*inputs=None*, *train_labels=None*, *vocabulary_size=80000*, *embedding_size=200*, *num_sampled=64*, *nce_loss_args={}*, *E_init=<tensorflow.python.ops.init_ops.RandomUniform object>*, *E_init_args={}*, *nce_W_init=<tensorflow.python.ops.init_ops.Truncated object>*, *nce_W_init_args={}*, *nce_b_init=<tensorflow.python.ops.init_ops.Constant object>*, *nce_b_init_args={}*, *name='word2vec_layer'*)

The *Word2vecEmbeddingInputlayer* class is a fully connected layer, for Word Embedding. Words are input as integer index. The output is the embedded word vector.

Parameters

inputs [placeholder] For word inputs. integer index format.

train_labels [placeholder] For word labels. integer index format.

vocabulary_size [int] The size of vocabulary, number of words.

embedding_size [int] The number of embedding dimensions.

num_sampled [int] The Number of negative examples for NCE loss.

nce_loss_args [a dictionary] The arguments for tf.nn.nce_loss()

E_init [embedding initializer] The initializer for initializing the embedding matrix.

E_init_args [a dictionary] The arguments for embedding initializer

nce_W_init [NCE decoder biases initializer] The initializer for initializing the nce decoder weight matrix.

nce_W_init_args [a dictionary] The arguments for initializing the nce decoder weight matrix.

nce_b_init [NCE decoder biases initializer] The initializer for tf.get_variable() of the nce decoder bias vector.

nce_b_init_args [a dictionary] The arguments for tf.get_variable() of the nce decoder bias vector.

name [a string or None] An optional name to attach to this layer.

References

- tensorflow/examples/tutorials/word2vec/word2vec_basic.py

Examples

- Without TensorLayer : see tensorflow/examples/tutorials/word2vec/word2vec_basic.py

```
>>> train_inputs = tf.placeholder(tf.int32, shape=[batch_size])
>>> train_labels = tf.placeholder(tf.int32, shape=[batch_size, 1])
>>> embeddings = tf.Variable(
...     tf.random_uniform([vocabulary_size, embedding_size], -1.0, 1.0))
>>> embed = tf.nn.embedding_lookup(embeddings, train_inputs)
>>> nce_weights = tf.Variable(
...     tf.truncated_normal([vocabulary_size, embedding_size],
...                         stddev=1.0 / math.sqrt(embedding_size)))
>>> nce_biases = tf.Variable(tf.zeros([vocabulary_size]))
>>> cost = tf.reduce_mean(
...    tf.nn.nce_loss(weights=nce_weights, biases=nce_biases,
...                   inputs=embed, labels=train_labels,
...                   num_sampled=num_sampled, num_classes=vocabulary_size,
...                   num_true=1))
```

- With TensorLayer : see tutorial_word2vec_basic.py

```
>>> train_inputs = tf.placeholder(tf.int32, shape=[batch_size])
>>> train_labels = tf.placeholder(tf.int32, shape=[batch_size, 1])
>>> emb_net = tl.layers.Word2vecEmbeddingInputlayer(
...         inputs = train_inputs,
...         train_labels = train_labels,
...         vocabulary_size = vocabulary_size,
...         embedding_size = embedding_size,
```

(continues on next page)

(continued from previous page)

```
...          num_sampled = num_sampled,
...         name ='word2vec_layer',
...     )
>>> cost = emb_net.nce_cost
>>> train_params = emb_net.all_params
>>> train_op = tf.train.GradientDescentOptimizer(learning_rate).minimize(
...                                                 cost, var_list=train_params)
>>> normalized_embeddings = emb_net.normalized_embeddings
```

Attributes

nce_cost [a tensor] The NCE loss.

outputs [a tensor] The outputs of embedding layer.

normalized_embeddings [tensor] Normalized embedding matrix

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

Embedding Input layer

class tensorlayer.layers.**EmbeddingInputlayer**(*inputs=None*, *vocabulary_size=80000*, *embedding_size=200*, *E_init=<tensorflow.python.ops.init_ops.RandomUniform object>*, *E_init_args={}*, *name='embedding_layer'*)

The *EmbeddingInputlayer* class is a fully connected layer, for Word Embedding. Words are input as integer index. The output is the embedded word vector.

If you have a pre-train matrix, you can assign the matrix into it. To train a word embedding matrix, you can used class:*Word2vecEmbeddingInputlayer*.

Note that, do not update this embedding matrix.

Parameters

inputs [placeholder] For word inputs. integer index format. a 2D tensor : [batch_size, num_steps(num_words)]

vocabulary_size [int] The size of vocabulary, number of words.

embedding_size [int] The number of embedding dimensions.

E_init [embedding initializer] The initializer for initializing the embedding matrix.

E_init_args [a dictionary] The arguments for embedding initializer

name [a string or None] An optional name to attach to this layer.

Examples

```
>>> vocabulary_size = 50000
>>> embedding_size = 200
>>> model_file_name = "model_word2vec_50k_200"
>>> batch_size = None
...
>>> all_var = tl.files.load_npy_to_any(name=model_file_name+'.npy')
>>> data = all_var['data']; count = all_var['count']
>>> dictionary = all_var['dictionary']
>>> reverse_dictionary = all_var['reverse_dictionary']
>>> tl.files.save_vocab(count, name='vocab_'+model_file_name+'.txt')
>>> del all_var, data, count
...
>>> load_params = tl.files.load_npz(name=model_file_name+'.npz')
>>> x = tf.placeholder(tf.int32, shape=[batch_size])
>>> y_ = tf.placeholder(tf.int32, shape=[batch_size, 1])
>>> emb_net = tl.layers.EmbeddingInputlayer(
...                inputs = x,
...                vocabulary_size = vocabulary_size,
...                embedding_size = embedding_size,
...                name ='embedding_layer')
>>> tl.layers.initialize_global_variables(sess)
>>> tl.files.assign_params(sess, [load_params[0]], emb_net)
>>> word = b'hello'
>>> word_id = dictionary[word]
>>> print('word_id:', word_id)
... 6428
...
>>> words = [b'i', b'am', b'hao', b'dong']
>>> word_ids = tl.files.words_to_word_ids(words, dictionary)
>>> context = tl.files.word_ids_to_words(word_ids, reverse_dictionary)
>>> print('word_ids:', word_ids)
... [72, 1226, 46744, 20048]
>>> print('context:', context)
... [b'i', b'am', b'hao', b'dong']
...
>>> vector = sess.run(emb_net.outputs, feed_dict={x : [word_id]})
>>> print('vector:', vector.shape)
... (1, 200)
>>> vectors = sess.run(emb_net.outputs, feed_dict={x : word_ids})
>>> print('vectors:', vectors.shape)
... (4, 200)
```

Attributes

outputs [a tensor] The outputs of embedding layer. the outputs 3D tensor : [batch_size, num_steps(num_words), embedding_size]

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

Average Embedding Input layer

class tensorlayer.layers.**AverageEmbeddingInputlayer**(*inputs*, *vocabulary_size*, *embedding_size*, *pad_value=0*, *name='average_embedding_layer'*, *embeddings_initializer=<tensorflow.python.ops.init_ops.Rando object>*, *embeddings_kwargs=None*)

The *AverageEmbeddingInputlayer* averages over embeddings of inputs, can be used as the input layer for models like DAN[1] and FastText[2].

Parameters

inputs [input placeholder or tensor]

vocabulary_size [an integer, the size of vocabulary]

embedding_size [an integer, the dimension of embedding vectors]

pad_value [an integer, the scalar pad value used in inputs]

name [a string, the name of the layer]

embeddings_initializer [the initializer of the embedding matrix]

embeddings_kwargs [kwargs to get embedding matrix variable]

References

- [1] Iyyer, M., Manjunatha, V., Boyd-Graber, J., & Daum'e III, H. (2015). Deep Unordered Composition Rivals Syntactic Methods for Text Classification. In Association for Computational Linguistics.
- [2] Joulin, A., Grave, E., Bojanowski, P., & Mikolov, T. (2016).'Bag of Tricks for Efficient Text Classification. <http://arxiv.org/abs/1607.01759>'_

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

2.1.9 Dense layer

Dense layer

class tensorlayer.layers.**DenseLayer**(*layer=None*, *n_units=100*, *act=<function identity>*, *W_init=<tensorflow.python.ops.init_ops.TruncatedNormal object>*, *b_init=<tensorflow.python.ops.init_ops.Constant object>*, *W_init_args={}*, *b_init_args={}*, *name='dense_layer'*)

The *DenseLayer* class is a fully connected layer.

Parameters

layer [a *Layer* instance] The *Layer* class feeding into this layer.

n_units [int] The number of units of the layer.

act [activation function] The function that is applied to the layer activations.

W_init [weights initializer] The initializer for initializing the weight matrix.

b_init [biases initializer or None] The initializer for initializing the bias vector. If None, skip biases.

W_init_args [dictionary] The arguments for the weights tf.get_variable.

b_init_args [dictionary] The arguments for the biases tf.get_variable.

name [a string or None] An optional name to attach to this layer.

Notes

If the input to this layer has more than two axes, it need to flatten the input by using *FlattenLayer* in this case.

Examples

```
>>> network = tl.layers.InputLayer(x, name='input_layer')
>>> network = tl.layers.DenseLayer(
...                 network,
...                 n_units=800,
...                 act = tf.nn.relu,
...                 W_init=tf.truncated_normal_initializer(stddev=0.1),
...                 name ='relu_layer'
...                 )
```

```
>>> Without TensorLayer, you can do as follow.
>>> W = tf.Variable(
...     tf.random_uniform([n_in, n_units], -1.0, 1.0), name='W')
>>> b = tf.Variable(tf.zeros(shape=[n_units]), name='b')
>>> y = tf.nn.relu(tf.matmul(inputs, W) + b)
```

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

Reconstruction layer for Autoencoder

class tensorlayer.layers.**ReconLayer**(*layer=None*, *x_recon=None*, *name='recon_layer'*, *n_units=784*, *act=<function softplus>*)

The *ReconLayer* class is a reconstruction layer *DenseLayer* which use to pre-train a *DenseLayer*.

Parameters

layer [a *Layer* instance] The *Layer* class feeding into this layer.

x_recon [tensorflow variable] The variables used for reconstruction.

name [a string or None] An optional name to attach to this layer.

n_units [int] The number of units of the layer, should be equal to x_recon

act [activation function] The activation function that is applied to the reconstruction layer. Normally, for sigmoid layer, the reconstruction activation is sigmoid; for rectifying layer, the reconstruction activation is softplus.

Notes

The input layer should be *DenseLayer* or a layer has only one axes. You may need to modify this part to define your own cost function. By default, the cost is implemented as follow: - For sigmoid layer, the implementation can be UFLDL - For rectifying layer, the implementation can be Glorot (2011). Deep Sparse Rectifier Neural Networks

Examples

```
>>> network = tl.layers.InputLayer(x, name='input_layer')
>>> network = tl.layers.DenseLayer(network, n_units=196,
...                                act=tf.nn.sigmoid, name='sigmoid1')
>>> recon_layer1 = tl.layers.ReconLayer(network, x_recon=x, n_units=784,
...                                act=tf.nn.sigmoid, name='recon_layer1')
>>> recon_layer1.pretrain(sess, x=x, X_train=X_train, X_val=X_val,
...                           denoise_name=None, n_epoch=1200, batch_size=128,
...                           print_freq=10, save=True, save_name='w1pre_')
```

Methods

pretrain(self, sess, x, X_train, X_val, denoise_name=None, n_epoch=100, batch_size=128, print_freq=10, save=True, save_name='w1pre_')	Start to pre-train the parameters of previous DenseLayer.

2.1.10 Noise layer

Dropout layer

class tensorlayer.layers.**DropoutLayer**(*layer=None*, *keep=0.5*, *is_fix=False*, *is_train=True*, *seed=None*, *name='dropout_layer'*)

The *DropoutLayer* class is a noise layer which randomly set some values to zero by a given keeping probability.

Parameters

layer [a *Layer* instance] The *Layer* class feeding into this layer.

keep [float] The keeping probability, the lower more values will be set to zero.

is_fix [boolean] Default False, if True, the keeping probability is fixed and cannot be changed via feed_dict.

is_train [boolean] If False, skip this layer, default is True.

seed [int or None] An integer or None to create random seed.

name [a string or None] An optional name to attach to this layer.

Notes

- A frequent question regarding `DropoutLayer` is that why it donot have *is_train* like `BatchNormLayer`.

In many simple cases, user may find it is better to use one inference instead of two inferences for training and testing seperately, `DropoutLayer` allows you to control the dropout rate via *feed_dict*. However, you can fix the keeping probability by setting *is_fix* to True.

Examples

- Define network

```
>>> network = tl.layers.InputLayer(x, name='input_layer')
>>> network = tl.layers.DropoutLayer(network, keep=0.8, name='drop1')
>>> network = tl.layers.DenseLayer(network, n_units=800, act = tf.nn.relu, name=
↪'relu1')
>>> ...
```

- For training, enable dropout as follow.

```
>>> feed_dict = {x: X_train_a, y_: y_train_a}
>>> feed_dict.update( network.all_drop )      # enable noise layers
>>> sess.run(train_op, feed_dict=feed_dict)
>>> ...
```

- For testing, disable dropout as follow.

```
>>> dp_dict = tl.utils.dict_to_one( network.all_drop ) # disable noise layers
>>> feed_dict = {x: X_val_a, y_: y_val_a}
>>> feed_dict.update(dp_dict)
>>> err, ac = sess.run([cost, acc], feed_dict=feed_dict)
>>> ...
```

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

Gaussian noise layer

class tensorlayer.layers.**GaussianNoiseLayer**(*layer=None*, *mean=0.0*, *stddev=1.0*, *is_train=True*, *seed=None*, *name='gaussian_noise_layer'*)

The `GaussianNoiseLayer` class is noise layer that adding noise with normal distribution to the activation.

Parameters

layer [a *Layer* instance] The *Layer* class feeding into this layer.

mean [float]

stddev [float]

is_train [boolean] If False, skip this layer, default is True.

seed [int or None] An integer or None to create random seed.

name [a string or None] An optional name to attach to this layer.

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

Dropconnect + Dense layer

class tensorlayer.layers.**DropconnectDenseLayer**(*layer=None*, *keep=0.5*, *n_units=100*, *act=<function identity>*, *W_init=<tensorflow.python.ops.init_ops.TruncatedNormal object>*, *b_init=<tensorflow.python.ops.init_ops.Constant object>*, *W_init_args={}*, *b_init_args={}*, *name='dropconnect_layer'*)

The *DropconnectDenseLayer* class is DenseLayer with DropConnect behaviour which randomly remove connection between this layer to previous layer by a given keeping probability.

Parameters

layer [a *Layer* instance] The *Layer* class feeding into this layer.

keep [float] The keeping probability, the lower more values will be set to zero.

n_units [int] The number of units of the layer.

act [activation function] The function that is applied to the layer activations.

W_init [weights initializer] The initializer for initializing the weight matrix.

b_init [biases initializer] The initializer for initializing the bias vector.

W_init_args [dictionary] The arguments for the weights tf.get_variable().

b_init_args [dictionary] The arguments for the biases tf.get_variable().

name [a string or None] An optional name to attach to this layer.

References

- Wan, L. (2013). Regularization of neural networks using dropconnect

Examples

```
>>> network = tl.layers.InputLayer(x, name='input_layer')
>>> network = tl.layers.DropconnectDenseLayer(network, keep = 0.8,
...         n_units=800, act = tf.nn.relu, name='dropconnect_relu1')
>>> network = tl.layers.DropconnectDenseLayer(network, keep = 0.5,
...         n_units=800, act = tf.nn.relu, name='dropconnect_relu2')
>>> network = tl.layers.DropconnectDenseLayer(network, keep = 0.5,
...         n_units=10, act = tl.activation.identity, name='output_layer')
```

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

2.1.11 Convolutional layer (Pro)

1D Convolution

class tensorlayer.layers.**Conv1dLayer**(*layer=None, act=<function identity>, shape=[5, 1, 5], stride=1, dilation_rate=1, padding='SAME', use_cudnn_on_gpu=None, data_format='NWC', W_init=<tensorflow.python.ops.init_ops.TruncatedNormal object>, b_init=<tensorflow.python.ops.init_ops.Constant object>, W_init_args={}, b_init_args={}, name='cnn_layer'*)

The *Conv1dLayer* class is a 1D CNN layer, see tf.nn.convolution.

Parameters

layer [a *Layer* instance] The *Layer* class feeding into this layer, [batch, in_width, in_channels].

act [activation function, None for identity.]

shape [list of shape] shape of the filters, [filter_length, in_channels, out_channels].

stride [an int.] The number of entries by which the filter is moved right at each step.

dilation_rate [an int.] Specifies the filter upsampling/input downsampling rate.

padding [a string from: "SAME", "VALID".] The type of padding algorithm to use.

use_cudnn_on_gpu [An optional bool. Defaults to True.]

data_format [As it is 1D conv, default is 'NWC'.]

W_init [weights initializer] The initializer for initializing the weight matrix.

b_init [biases initializer or None] The initializer for initializing the bias vector. If None, skip biases.

W_init_args [dictionary] The arguments for the weights tf.get_variable().

b_init_args [dictionary] The arguments for the biases tf.get_variable().

name [a string or None] An optional name to attach to this layer.

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

2D Convolution

class tensorlayer.layers.**Conv2dLayer**(*layer=None, act=<function identity>, shape=[5, 5, 1, 100], strides=[1, 1, 1, 1], padding='SAME', W_init=<tensorflow.python.ops.init_ops.TruncatedNormal object>, b_init=<tensorflow.python.ops.init_ops.Constant object>, W_init_args={}, b_init_args={}, use_cudnn_on_gpu=None, data_format=None, name='cnn_layer'*)

The *Conv2dLayer* class is a 2D CNN layer, see tf.nn.conv2d.

Parameters

layer [a *Layer* instance] The *Layer* class feeding into this layer.

act [activation function] The function that is applied to the layer activations.

shape [list of shape] shape of the filters, [filter_height, filter_width, in_channels, out_channels].

strides [a list of ints.] The stride of the sliding window for each dimension of input.

It Must be in the same order as the dimension specified with format.

padding [a string from: "SAME", "VALID".] The type of padding algorithm to use.

W_init [weights initializer] The initializer for initializing the weight matrix.

b_init [biases initializer or None] The initializer for initializing the bias vector. If None, skip biases.

W_init_args [dictionary] The arguments for the weights tf.get_variable().

b_init_args [dictionary] The arguments for the biases tf.get_variable().

use_cudnn_on_gpu [bool, default is None.]

data_format [string "NHWC" or "NCHW", default is "NHWC"]

name [a string or None] An optional name to attach to this layer.

Notes

- shape = [h, w, the number of output channel of previous layer, the number of output channels]
- the number of output channel of a layer is its last dimension.

Examples

```
>>> x = tf.placeholder(tf.float32, shape=[None, 28, 28, 1])
>>> network = tl.layers.InputLayer(x, name='input_layer')
>>> network = tl.layers.Conv2dLayer(network,
```

(continues on next page)

(continued from previous page)

```
...                     act = tf.nn.relu,
...                     shape = [5, 5, 1, 32],  # 32 features for each 5x5 patch
...                     strides=[1, 1, 1, 1],
...                     padding='SAME',
...                     W_init=tf.truncated_normal_initializer(stddev=5e-2),
...                     W_init_args={},
...                     b_init = tf.constant_initializer(value=0.0),
...                     b_init_args = {},
...                     name ='cnn_layer1')     # output: (?, 28, 28, 32)
>>> network = tl.layers.PoolLayer(network,
...                     ksize=[1, 2, 2, 1],
...                     strides=[1, 2, 2, 1],
...                     padding='SAME',
...                     pool = tf.nn.max_pool,
...                     name ='pool_layer1',)   # output: (?, 14, 14, 32)
```

```
>>> Without TensorLayer, you can implement 2d convolution as follow.
>>> W = tf.Variable(W_init(shape=[5, 5, 1, 32], ), name='W_conv')
>>> b = tf.Variable(b_init(shape=[32], ), name='b_conv')
>>> outputs = tf.nn.relu( tf.nn.conv2d(inputs, W,
...                         strides=[1, 1, 1, 1],
...                         padding='SAME') + b )
```

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

2D Deconvolution

class tensorlayer.layers.**DeConv2dLayer**(*layer=None, act=<function identity>, shape=[3, 3, 128, 256], output_shape=[1, 256, 256, 128], strides=[1, 2, 2, 1], padding='SAME', W_init=<tensorflow.python.ops.init_ops.TruncatedNormal object>, b_init=<tensorflow.python.ops.init_ops.Constant object>, W_init_args={}, b_init_args={}, name='decnn2d_layer'*)

The *DeConv2dLayer* class is deconvolutional 2D layer, see tf.nn.conv2d_transpose.

Parameters

layer [a *Layer* instance] The *Layer* class feeding into this layer.

act [activation function] The function that is applied to the layer activations.

shape [list of shape] shape of the filters, [height, width, output_channels, in_channels], filter's in_channels dimension must match that of value.

output_shape [list of output shape] representing the output shape of the deconvolution op.

strides [a list of ints.] The stride of the sliding window for each dimension of the input tensor.

padding [a string from: "SAME", "VALID".] The type of padding algorithm to use.

W_init [weights initializer] The initializer for initializing the weight matrix.

b_init [biases initializer] The initializer for initializing the bias vector. If None, skip biases.

W_init_args [dictionary] The arguments for the weights initializer.

b_init_args [dictionary] The arguments for the biases initializer.

name [a string or None] An optional name to attach to this layer.

Notes

- shape = [h, w, the number of output channels of this layer, the number of output channel of previous layer]
- output_shape = [batch_size, any, any, the number of output channels of this layer]
- the number of output channel of a layer is its last dimension.

Examples

- A part of the generator in DCGAN example

```
>>> batch_size = 64
>>> inputs = tf.placeholder(tf.float32, [batch_size, 100], name='z_noise')
>>> net_in = tl.layers.InputLayer(inputs, name='g/in')
>>> net_h0 = tl.layers.DenseLayer(net_in, n_units = 8192,
...                               W_init = tf.random_normal_initializer(stddev=0.02),
...                               act = tf.identity, name='g/h0/lin')
>>> print(net_h0.outputs._shape)
... (64, 8192)
>>> net_h0 = tl.layers.ReshapeLayer(net_h0, shape = [-1, 4, 4, 512], name='g/h0/
↪reshape')
>>> net_h0 = tl.layers.BatchNormLayer(net_h0, act=tf.nn.relu, is_train=is_train,␣
↪name='g/h0/batch_norm')
>>> print(net_h0.outputs._shape)
... (64, 4, 4, 512)
>>> net_h1 = tl.layers.DeConv2dLayer(net_h0,
...                               shape = [5, 5, 256, 512],
...                               output_shape = [batch_size, 8, 8, 256],
...                               strides=[1, 2, 2, 1],
...                               act=tf.identity, name='g/h1/decon2d')
>>> net_h1 = tl.layers.BatchNormLayer(net_h1, act=tf.nn.relu, is_train=is_train,␣
↪name='g/h1/batch_norm')
>>> print(net_h1.outputs._shape)
... (64, 8, 8, 256)
```

- U-Net

```
>>> ....
>>> conv10 = tl.layers.Conv2dLayer(conv9, act=tf.nn.relu,
...        shape=[3,3,1024,1024], strides=[1,1,1,1], padding='SAME',
...        W_init=w_init, b_init=b_init, name='conv10')
>>> print(conv10.outputs)
... (batch_size, 32, 32, 1024)
>>> deconv1 = tl.layers.DeConv2dLayer(conv10, act=tf.nn.relu,
...         shape=[3,3,512,1024], strides=[1,2,2,1], output_shape=[batch_size,64,
↪64,512],
...         padding='SAME', W_init=w_init, b_init=b_init, name='devcon1_1')
```

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

3D Convolution

class tensorlayer.layers.**Conv3dLayer**(*layer=None, act=<function identity>, shape=[2, 2, 2, 64, 128], strides=[1, 2, 2, 2, 1], padding='SAME', W_init=<tensorflow.python.ops.init_ops.TruncatedNormal object>, b_init=<tensorflow.python.ops.init_ops.Constant object>, W_init_args={}, b_init_args={}, name='cnn3d_layer'*)

The *Conv3dLayer* class is a 3D CNN layer, see tf.nn.conv3d.

Parameters

layer [a *Layer* instance] The *Layer* class feeding into this layer.

act [activation function] The function that is applied to the layer activations.

shape [list of shape] shape of the filters, [filter_depth, filter_height, filter_width, in_channels, out_channels].

strides [a list of ints. 1-D of length 4.] The stride of the sliding window for each dimension of input. Must be in the same order as the dimension specified with format.

padding [a string from: "SAME", "VALID".] The type of padding algorithm to use.

W_init [weights initializer] The initializer for initializing the weight matrix.

b_init [biases initializer] The initializer for initializing the bias vector.

W_init_args [dictionary] The arguments for the weights initializer.

b_init_args [dictionary] The arguments for the biases initializer.

name [a string or None] An optional name to attach to this layer.

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

3D Deconvolution

class tensorlayer.layers.**DeConv3dLayer**(*layer=None, act=<function identity>, shape=[2, 2, 2, 128, 256], output_shape=[1, 12, 32, 32, 128], strides=[1, 2, 2, 2, 1], padding='SAME', W_init=<tensorflow.python.ops.init_ops.TruncatedNormal object>, b_init=<tensorflow.python.ops.init_ops.Constant object>, W_init_args={}, b_init_args={}, name='decnn3d_layer'*)

The *DeConv3dLayer* class is deconvolutional 3D layer, see tf.nn.conv3d_transpose.

Parameters

layer [a *Layer* instance] The *Layer* class feeding into this layer.

act [activation function] The function that is applied to the layer activations.

shape [list of shape] shape of the filters, [depth, height, width, output_channels, in_channels], filter's in_channels dimension must match that of value.

output_shape [list of output shape] representing the output shape of the deconvolution op.

strides [a list of ints.] The stride of the sliding window for each dimension of the input tensor.

padding [a string from: "SAME", "VALID".] The type of padding algorithm to use.

W_init [weights initializer] The initializer for initializing the weight matrix.

b_init [biases initializer] The initializer for initializing the bias vector.

W_init_args [dictionary] The arguments for the weights initializer.

b_init_args [dictionary] The arguments for the biases initializer.

name [a string or None] An optional name to attach to this layer.

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

2D UpSampling

class tensorlayer.layers.**UpSampling2dLayer**(*layer=None*, *size=[]*, *is_scale=True*, *method=0*, *align_corners=False*, *name='upsample2d_layer'*)

The *UpSampling2dLayer* class is upSampling 2d layer, see tf.image.resize_images.

Parameters

layer [a layer class with 4-D Tensor of shape [batch, height, width, channels] or 3-D Tensor of shape [height, width, channels].]

size [a tuple of int or float.] (height, width) scale factor or new size of height and width.

is_scale [boolean, if True (default), size is scale factor, otherwise, size is number of pixels of height and width.]

method [0, 1, 2, 3. ResizeMethod. Defaults to ResizeMethod.BILINEAR.]

- ResizeMethod.BILINEAR, Bilinear interpolation.
- ResizeMethod.NEAREST_NEIGHBOR, Nearest neighbor interpolation.
- ResizeMethod.BICUBIC, Bicubic interpolation.
- ResizeMethod.AREA, Area interpolation.

align_corners [bool. If true, exactly align all 4 corners of the input and output. Defaults to false.]

name [a string or None] An optional name to attach to this layer.

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

2D DownSampling

class tensorlayer.layers.**DownSampling2dLayer**(*layer=None, size=[], is_scale=True, method=0, align_corners=False, name='downsample2d_layer'*)

The *DownSampling2dLayer* class is downSampling 2d layer, see tf.image.resize_images.

Parameters

layer [a layer class with 4-D Tensor of shape [batch, height, width, channels] or 3-D Tensor of shape [height, width, channels].]

size [a tupe of int or float.] (height, width) scale factor or new size of height and width.

is_scale [boolean, if True (default), size is scale factor, otherwise, size is number of pixels of height and width.]

method [0, 1, 2, 3. ResizeMethod. Defaults to ResizeMethod.BILINEAR.]

- ResizeMethod.BILINEAR, Bilinear interpolation.
- ResizeMethod.NEAREST_NEIGHBOR, Nearest neighbor interpolation.
- ResizeMethod.BICUBIC, Bicubic interpolation.
- ResizeMethod.AREA, Area interpolation.

align_corners [bool. If true, exactly align all 4 corners of the input and output. Defaults to false.]

name [a string or None] An optional name to attach to this layer.

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

2D Deformable Conv

class tensorlayer.layers.**DeformableConv2dLayer**(*layer=None, act=<function identity>, offset_layer=None, shape=[3, 3, 1, 100], name='deformable_conv_2d_layer', W_init=<tensorflow.python.ops.init_ops.TruncatedNormal object>, b_init=<tensorflow.python.ops.init_ops.Constant object>, W_init_args={}, b_init_args={}*)

The *DeformableConv2dLayer* class is a Deformable Convolutional Networks .

Parameters

layer [TensorLayer layer.]

offset_layer [TensorLayer layer, to predict the offset of convolutional operations. The shape of its output should be (batchsize, input height, input width, 2*(number of element in the convolutional kernel))] e.g. if apply a 3*3 kernel, the number of the last dimension should be 18 (2*3*3)

channel_multiplier [int, The number of channels to expand to.]

filter_size [tuple (height, width) for filter size.]

strides [tuple (height, width) for strides. Current implementation fix to (1, 1, 1, 1)]

act [None or activation function.]

shape [list of shape] shape of the filters, [filter_height, filter_width, in_channels, out_channels].

W_init [weights initializer] The initializer for initializing the weight matrix.

b_init [biases initializer or None] The initializer for initializing the bias vector. If None, skip biases.

W_init_args [dictionary] The arguments for the weights tf.get_variable().

b_init_args [dictionary] The arguments for the biases tf.get_variable().

name [a string or None] An optional name to attach to this layer.

Notes

- The stride is fixed as (1, 1, 1, 1).
- The padding is fixed as 'SAME'.
- The current implementation is memory-inefficient, please use carefully.

References

- The deformation operation was adapted from the implementation in https://github.com/felixlaumon/deform-conv

Examples

```
>>> network = tl.layers.InputLayer(x, name='input_layer')
>>> offset_1 = tl.layers.Conv2dLayer(layer=network, act=act, shape=[3, 3, 3, 18],␣
↪strides=[1, 1, 1, 1],padding='SAME', name='offset_layer1')
>>> network = tl.layers.DeformableConv2dLayer(layer=network, act=act, offset_
↪layer=offset_1,  shape=[3, 3, 3, 32],  name='deformable_conv_2d_layer1')
>>> offset_2 = tl.layers.Conv2dLayer(layer=network, act=act, shape=[3, 3, 32, 18],
↪ strides=[1, 1, 1, 1], padding='SAME', name='offset_layer2')
>>> network = tl.layers.DeformableConv2dLayer(layer=network, act = act, offset_
↪layer=offset_2, shape=[3, 3, 32, 64], name='deformable_conv_2d_layer2')
```

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

1D Atrous convolution

tensorlayer.layers.**AtrousConv1dLayer**(*net*, *n_filter=32*, *filter_size=2*, *stride=1*, *dilation=1*, *act=None*, *padding='SAME'*, *use_cudnn_on_gpu=None*, *data_format='NWC'*, *W_init=<tensorflow.python.ops.init_ops.TruncatedNormal object>*, *b_init=<tensorflow.python.ops.init_ops.Constant object>*, *W_init_args={}*, *b_init_args={}*, *name='conv1d'*)

Wrapper for *AtrousConv1dLayer*, if you don't understand how to use *Conv1dLayer*, this function may be easier.

Parameters

net [TensorLayer layer.]

n_filter [number of filter.]

filter_size [an int.]

stride [an int.]

dilation [an int, filter dilation size.]

act [None or activation function.]

others [see *Conv1dLayer*.]

2D Atrous convolution

class tensorlayer.layers.**AtrousConv2dLayer**(*layer=None*, *n_filter=32*, *filter_size=(3, 3)*, *rate=2*, *act=None*, *padding='SAME'*, *W_init=<tensorflow.python.ops.init_ops.TruncatedNormal object>*, *b_init=<tensorflow.python.ops.init_ops.Constant object>*, *W_init_args={}*, *b_init_args={}*, *name='atrou2d'*)

The *AtrousConv2dLayer* class is Atrous convolution (a.k.a. convolution with holes or dilated convolution) 2D layer, see tf.nn.atrous_conv2d.

Parameters

layer [a layer class with 4-D Tensor of shape [batch, height, width, channels].]

filters [A 4-D Tensor with the same type as value and shape [filter_height, filter_width, in_channels, out_channels]. filters' in_channels dimension must match that of value. Atrous convolution is equivalent to standard convolution with upsampled filters with effective height filter_height + (filter_height - 1) * (rate - 1) and effective width filter_width + (filter_width - 1) * (rate - 1), produced by inserting rate - 1 zeros along consecutive elements across the filters' spatial dimensions.]

n_filter [number of filter.]

filter_size [tuple (height, width) for filter size.]

rate [A positive int32. The stride with which we sample input values across the height and width dimensions. Equivalently, the rate by which we upsample the filter values by inserting zeros across the height and width dimensions. In the literature, the same parameter is sometimes called input stride or dilation.]

act [activation function, None for linear.]

padding [A string, either 'VALID' or 'SAME'. The padding algorithm.]

W_init [weights initializer. The initializer for initializing the weight matrix.]

b_init [biases initializer or None. The initializer for initializing the bias vector. If None, skip biases.]

W_init_args [dictionary. The arguments for the weights tf.get_variable().]

b_init_args [dictionary. The arguments for the biases tf.get_variable().]

name [a string or None, an optional name to attach to this layer.]

Methods

`count_params`()	Return the number of parameters in the network
`print_layers`()	Print all info of layers in the network
`print_params`([details, session])	Print all info of parameters in the network

2.1.12 Convolutional layer (Simplified)

For users don't familiar with TensorFlow, the following simplified functions may easier for you. We will provide more simplified functions later, but if you are good at TensorFlow, the professional APIs may better for you.

1D Convolution

`tensorlayer.layers.`**`Conv1d`**(*net*, *n_filter=32*, *filter_size=5*, *stride=1*, *dilation_rate=1*, *act=None*, *padding='SAME'*, *use_cudnn_on_gpu=None*, *data_format='NWC'*, *W_init=<tensorflow.python.ops.init_ops.TruncatedNormal object>*, *b_init=<tensorflow.python.ops.init_ops.Constant object>*, *W_init_args={}*, *b_init_args={}*, *name='conv1d'*)

Wrapper for `Conv1dLayer`, if you don't understand how to use `Conv1dLayer`, this function may be easier.

Parameters

net [TensorLayer layer.]

n_filter [number of filter.]

filter_size [an int.]

stride [an int.]

dilation_rate [As it is 1D conv, the default is "NWC".]

act [None or activation function.]

others [see `Conv1dLayer`.]

Examples

```
>>> x = tf.placeholder(tf.float32, [batch_size, width])
>>> y_ = tf.placeholder(tf.int64, shape=[batch_size,])
>>> n = InputLayer(x, name='in')
>>> n = ReshapeLayer(n, [-1, width, 1], name='rs')
>>> n = Conv1d(n, 64, 3, 1, act=tf.nn.relu, name='c1')
>>> n = MaxPool1d(n, 2, 2, padding='valid', name='m1')
>>> n = Conv1d(n, 128, 3, 1, act=tf.nn.relu, name='c2')
>>> n = MaxPool1d(n, 2, 2, padding='valid', name='m2')
>>> n = Conv1d(n, 128, 3, 1, act=tf.nn.relu, name='c3')
>>> n = MaxPool1d(n, 2, 2, padding='valid', name='m3')
>>> n = FlattenLayer(n, name='f')
>>> n = DenseLayer(n, 500, tf.nn.relu, name='d1')
>>> n = DenseLayer(n, 100, tf.nn.relu, name='d2')
>>> n = DenseLayer(n, 2, tf.identity, name='o')
```

2D Convolution

tensorlayer.layers.**Conv2d**(*net*, *n_filter=32*, *filter_size=(3, 3)*, *strides=(1, 1)*, *act=None*, *padding='SAME'*, *W_init=<tensorflow.python.ops.init_ops.TruncatedNormal object>*, *b_init=<tensorflow.python.ops.init_ops.Constant object>*, *W_init_args={}*, *b_init_args={}*, *use_cudnn_on_gpu=None*, *data_format=None*, *name='conv2d'*)

Wrapper for *Conv2dLayer*, if you don't understand how to use *Conv2dLayer*, this function may be easier.

Parameters

net [TensorLayer layer.]

n_filter [number of filter.]

filter_size [tuple (height, width) for filter size.]

strides [tuple (height, width) for strides.]

act [None or activation function.]

others [see *Conv2dLayer*.]

Examples

```
>>> w_init = tf.truncated_normal_initializer(stddev=0.01)
>>> b_init = tf.constant_initializer(value=0.0)
>>> inputs = InputLayer(x, name='inputs')
>>> conv1 = Conv2d(inputs, 64, (3, 3), act=tf.nn.relu, padding='SAME', W_init=w_
↪init, b_init=b_init, name='conv1_1')
>>> conv1 = Conv2d(conv1, 64, (3, 3), act=tf.nn.relu, padding='SAME', W_init=w_
↪init, b_init=b_init, name='conv1_2')
>>> pool1 = MaxPool2d(conv1, (2, 2), padding='SAME', name='pool1')
>>> conv2 = Conv2d(pool1, 128, (3, 3), act=tf.nn.relu, padding='SAME', W_init=w_
↪init, b_init=b_init, name='conv2_1')
>>> conv2 = Conv2d(conv2, 128, (3, 3), act=tf.nn.relu, padding='SAME', W_init=w_
↪init, b_init=b_init, name='conv2_2')
>>> pool2 = MaxPool2d(conv2, (2, 2), padding='SAME', name='pool2')
```

2D Deconvolution

tensorlayer.layers.**DeConv2d**(*net*, *n_out_channel=32*, *filter_size=(3, 3)*, *out_size=(30, 30)*, *strides=(2, 2)*, *padding='SAME'*, *batch_size=None*, *act=None*, *W_init=<tensorflow.python.ops.init_ops.TruncatedNormal object>*, *b_init=<tensorflow.python.ops.init_ops.Constant object>*, *W_init_args={}*, *b_init_args={}*, *name='decnn2d'*)

Wrapper for *DeConv2dLayer*, if you don't understand how to use *DeConv2dLayer*, this function may be easier.

Parameters

net [TensorLayer layer.]

n_out_channel [int, number of output channel.]

filter_size [tuple of (height, width) for filter size.]

out_size [tuple of (height, width) of output.]

batch_size [int or None, batch_size. If None, try to find the batch_size from the first dim of net.outputs (you should tell the batch_size when define the input placeholder).]

strides [tuple of (height, width) for strides.]

act [None or activation function.]

others [see *DeConv2dLayer*.]

1D Max pooling

tensorlayer.layers.**MaxPool1d**(*net*, *filter_size*, *strides*, *padding='valid'*, *data_format='channels_last'*, *name=None*)

Wrapper for tf.layers.max_pooling1d .

Parameters

net [TensorLayer layer, the tensor over which to pool. Must have rank 3.]

filter_size (pool_size) [An integer or tuple/list of a single integer, representing the size of the pooling window.]

strides [An integer or tuple/list of a single integer, specifying the strides of the pooling operation.]

padding [A string. The padding method, either 'valid' or 'same'. Case-insensitive.]

data_format [A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch, length, channels) while channels_first corresponds to inputs with shape (batch, channels, length).]

name [A string, the name of the layer.]

Returns

- A :class:'Layer' which the output tensor, of rank 3.

1D Mean pooling

tensorlayer.layers.**MeanPool1d**(*net*, *filter_size*, *strides*, *padding='valid'*, *data_format='channels_last'*, *name=None*)

Wrapper for tf.layers.average_pooling1d .

Parameters

net [TensorLayer layer, the tensor over which to pool. Must have rank 3.]

filter_size (pool_size) [An integer or tuple/list of a single integer, representing the size of the pooling window.]

strides [An integer or tuple/list of a single integer, specifying the strides of the pooling operation.]

padding [A string. The padding method, either 'valid' or 'same'. Case-insensitive.]

data_format [A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch, length, channels) while channels_first corresponds to inputs with shape (batch, channels, length).]

name [A string, the name of the layer.]

Returns

- A :class:'Layer' which the output tensor, of rank 3.

2D Max pooling

tensorlayer.layers.**MaxPool2d**(*net*, *filter_size=(2, 2)*, *strides=None*, *padding='SAME'*, *name='maxpool'*)

Wrapper for *PoolLayer*.

Parameters

net [TensorLayer layer.]

filter_size [tuple of (height, width) for filter size.]

strides [tuple of (height, width). Default is the same with filter_size.]

others [see *PoolLayer*.]

2D Mean pooling

tensorlayer.layers.**MeanPool2d**(*net*, *filter_size=(2, 2)*, *strides=None*, *padding='SAME'*, *name='meanpool'*)

Wrapper for *PoolLayer*.

Parameters

net [TensorLayer layer.]

filter_size [tuple of (height, width) for filter size.]

strides [tuple of (height, width). Default is the same with filter_size.]

others [see *PoolLayer*.]

3D Max pooling

tensorlayer.layers.**MaxPool3d**(*net*, *filter_size*, *strides*, *padding='valid'*, *data_format='channels_last'*, *name=None*)

Wrapper for tf.layers.max_pooling3d .

Parameters

net [TensorLayer layer, the tensor over which to pool. Must have rank 5.]

filter_size (pool_size) [An integer or tuple/list of 3 integers: (pool_depth, pool_height, pool_width) specifying the size of the pooling window. Can be a single integer to specify the same value for all spatial dimensions.]

strides [An integer or tuple/list of 3 integers, specifying the strides of the pooling operation. Can be a single integer to specify the same value for all spatial dimensions.]

padding [A string. The padding method, either 'valid' or 'same'. Case-insensitive.]

data_format [A string. The ordering of the dimensions in the inputs. channels_last (default) and channels_first are supported. channels_last corresponds to inputs with shape (batch, depth, height, width, channels) while channels_first corresponds to inputs with shape (batch, channels, depth, height, width).]

name [A string, the name of the layer.]

3D Mean pooling

tensorlayer.layers.**MeanPool3d**(*net*, *filter_size*, *strides*, *padding='valid'*, *data_format='channels_last'*, *name=None*)

Wrapper for tf.layers.average_pooling3d

Parameters

net [TensorLayer layer, the tensor over which to pool. Must have rank 5.]

filter_size (pool_size) [An integer or tuple/list of 3 integers: (pool_depth, pool_height, pool_width) specifying the size of the pooling window. Can be a single integer to specify the same value for all spatial dimensions.]

strides [An integer or tuple/list of 3 integers, specifying the strides of the pooling operation. Can be a single integer to specify the same value for all spatial dimensions.]

padding [A string. The padding method, either 'valid' or 'same'. Case-insensitive.]

data_format [A string. The ordering of the dimensions in the inputs. channels_last (default) and channels_first are supported. channels_last corresponds to inputs with shape (batch, depth, height, width, channels) while channels_first corresponds to inputs with shape (batch, channels, depth, height, width).]

name [A string, the name of the layer.]

2D Depthwise/Separable Conv

class tensorlayer.layers.**DepthwiseConv2d**(*layer=None*, *channel_multiplier=3*, *shape=(3, 3)*, *strides=(1, 1)*, *act=None*, *padding='SAME'*, *W_init=<tensorflow.python.ops.init_ops.TruncatedNormal object>*, *b_init=<tensorflow.python.ops.init_ops.Constant object>*, *W_init_args={}*, *b_init_args={}*, *name='depthwise_conv2d'*)

Separable/Depthwise Convolutional 2D, see tf.nn.depthwise_conv2d.

Input: 4-D Tensor [batch, height, width, in_channels].

Output: 4-D Tensor [batch, new height, new width, in_channels * channel_multiplier].

Parameters

net [TensorLayer layer.]

channel_multiplier [int, The number of channels to expand to.]

filter_size [tuple (height, width) for filter size.]

strides [tuple (height, width) for strides.]

act [None or activation function.]

padding [a string from: "SAME", "VALID".] The type of padding algorithm to use.

W_init [weights initializer] The initializer for initializing the weight matrix.

b_init [biases initializer or None] The initializer for initializing the bias vector. If None, skip biases.

W_init_args [dictionary] The arguments for the weights tf.get_variable().

b_init_args [dictionary] The arguments for the biases tf.get_variable().

name [a string or None] An optional name to attach to this layer.

References

- tflearn's grouped_conv_2d
- keras's separableconv2d

Examples

```
>>> t_im = tf.placeholder("float32", [None, 256, 256, 3])
>>> net = InputLayer(t_im, name='in')
>>> net = DepthwiseConv2d(net, 32, (3, 3), (1, 1, 1, 1), tf.nn.relu, padding="SAME
↪", name='dep')
>>> print(net.outputs.get_shape())
... (?, 256, 256, 96)
```

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

2.1.13 Super-Resolution layer

1D Subpixel Convolution

tensorlayer.layers.**SubpixelConv1d**(*net*, *scale=2*, *act=<function identity>*, *name='subpixel_conv1d'*)

One-dimensional subpixel upsampling layer. Calls a tensorflow function that directly implements this functionality. We assume input has dim (batch, width, r)

Parameters

net [TensorLayer layer.]

scale [int, upscaling ratio, a wrong setting will lead to Dimension size error.]

act [activation function.]

name [string.] An optional name to attach to this layer.

References

- Audio Super Resolution Implementation.

Examples

```
>>> t_signal = tf.placeholder('float32', [10, 100, 4], name='x')
>>> n = InputLayer(t_signal, name='in')
>>> n = SubpixelConv1d(n, scale=2, name='s')
>>> print(n.outputs.shape)
... (10, 200, 2)
```

2D Subpixel Convolution

tensorlayer.layers.**SubpixelConv2d**(*net*, *scale=2*, *n_out_channel=None*, *act=<function identity>*, *name='subpixel_conv2d'*)

It is a sub-pixel 2d upsampling layer, usually be used for Super-Resolution applications, see example code.

Parameters

net [TensorLayer layer.]

scale [int, upscaling ratio, a wrong setting will lead to Dimension size error.]

n_out_channel [int or None, the number of output channels.] Note that, the number of input channels == (scale x scale) x The number of output channels. If None, automatically set n_out_channel == the number of input channels / (scale x scale).

act [activation function.]

name [string.] An optional name to attach to this layer.

References

- Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network

Examples

```
>>> # examples here just want to tell you how to set the n_out_channel.
>>> x = np.random.rand(2, 16, 16, 4)
>>> X = tf.placeholder("float32", shape=(2, 16, 16, 4), name="X")
>>> net = InputLayer(X, name='input')
>>> net = SubpixelConv2d(net, scale=2, n_out_channel=1, name='subpixel_conv2d')
>>> y = sess.run(net.outputs, feed_dict={X: x})
>>> print(x.shape, y.shape)
... (2, 16, 16, 4) (2, 32, 32, 1)
```

(continues on next page)

(continued from previous page)

```
>>>
>>> x = np.random.rand(2, 16, 16, 4*10)
>>> X = tf.placeholder("float32", shape=(2, 16, 16, 4*10), name="X")
>>> net = InputLayer(X, name='input2')
>>> net = SubpixelConv2d(net, scale=2, n_out_channel=10, name='subpixel_conv2d2')
>>> y = sess.run(net.outputs, feed_dict={X: x})
>>> print(x.shape, y.shape)
... (2, 16, 16, 40) (2, 32, 32, 10)
>>>
>>> x = np.random.rand(2, 16, 16, 25*10)
>>> X = tf.placeholder("float32", shape=(2, 16, 16, 25*10), name="X")
>>> net = InputLayer(X, name='input3')
>>> net = SubpixelConv2d(net, scale=5, n_out_channel=None, name='subpixel_conv2d3
↪')
>>> y = sess.run(net.outputs, feed_dict={X: x})
>>> print(x.shape, y.shape)
... (2, 16, 16, 250) (2, 80, 80, 10)
```

2.1.14 Spatial Transformer

2D Affine Transformation

class tensorlayer.layers.**SpatialTransformer2dAffineLayer**(*layer=None, theta_layer=None, out_size=[40, 40], name='sapatial_trans_2d_affine'*)

The *SpatialTransformer2dAffineLayer* class is a Spatial Transformer Layer for 2D Affine Transformation.

Parameters

layer [a layer class with 4-D Tensor of shape [batch, height, width, channels]]

theta_layer [a layer class for the localisation network.] In this layer, we will use a *DenseLayer* to make the theta size to [batch, 6], value range to [0, 1] (via tanh).

out_size [tuple of two ints.] The size of the output of the network (height, width), the feature maps will be resized by this.

References

- Spatial Transformer Networks
- TensorFlow/Models

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

2D Affine Transformation function

tensorlayer.layers.**transformer**(*U*, *theta*, *out_size*, *name='SpatialTransformer2dAffine'*, ***kwargs*)

Spatial Transformer Layer for 2D Affine Transformation , see *SpatialTransformer2dAffineLayer* class.

Parameters

U [float] The output of a convolutional net should have the shape [num_batch, height, width, num_channels].

theta: float The output of the localisation network should be [num_batch, 6], value range should be [0, 1] (via tanh).

out_size: tuple of two ints The size of the output of the network (height, width)

Notes

- To initialize the network to the identity transform init.

```
>>> ``theta`` to
>>> identity = np.array([[1., 0., 0.],
...                      [0., 1., 0.]])
>>> identity = identity.flatten()
>>> theta = tf.Variable(initial_value=identity)
```

References

- Spatial Transformer Networks
- TensorFlow/Models

Batch 2D Affine Transformation function

tensorlayer.layers.**batch_transformer**(*U*, *thetas*, *out_size*, *name='BatchSpatialTransformer2dAffine'*)

Batch Spatial Transformer function for 2D Affine Transformation.

Parameters

U [float] tensor of inputs [batch, height, width, num_channels]

thetas [float] a set of transformations for each input [batch, num_transforms, 6]

out_size [int] the size of the output [out_height, out_width]

Returns: float Tensor of size [batch * num_transforms, out_height, out_width, num_channels]

2.1.15 Pooling layer

Pooling layer for any dimensions and any pooling functions.

class tensorlayer.layers.**PoolLayer**(*layer=None, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME', pool=<function max_pool>, name='pool_layer'*)

The *PoolLayer* class is a Pooling layer, you can choose `tf.nn.max_pool` and `tf.nn.avg_pool` for 2D or `tf.nn.max_pool3d` and `tf.nn.avg_pool3d` for 3D.

Parameters

layer [a *Layer* instance] The *Layer* class feeding into this layer.

ksize [a list of ints that has length >= 4.] The size of the window for each dimension of the input tensor.

strides [a list of ints that has length >= 4.] The stride of the sliding window for each dimension of the input tensor.

padding [a string from: "SAME", "VALID".] The type of padding algorithm to use.

pool [a pooling function]

- see TensorFlow pooling APIs
- class `tf.nn.max_pool`
- class `tf.nn.avg_pool`
- class `tf.nn.max_pool3d`
- class `tf.nn.avg_pool3d`

name [a string or None] An optional name to attach to this layer.

Examples

- see *Conv2dLayer*.

Methods

`count_params`()	Return the number of parameters in the network
`print_layers`()	Print all info of layers in the network
`print_params`([details, session])	Print all info of parameters in the network

2.1.16 Padding

Padding layer for any modes.

class tensorlayer.layers.**PadLayer**(*layer=None, paddings=None, mode='CONSTANT', name='pad_layer'*)

The *PadLayer* class is a Padding layer for any modes and dimensions. Please see tf.pad for usage.

Parameters

layer [a *Layer* instance] The *Layer* class feeding into this layer.

padding [a Tensor of type int32.]

mode [one of "CONSTANT", "REFLECT", or "SYMMETRIC" (case-insensitive)]

name [a string or None] An optional name to attach to this layer.

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

2.1.17 Normalization layer

For local response normalization as it does not have any weights and arguments, you can also apply `tf.nn.lrn` on `network.outputs`.

Batch Normalization

class tensorlayer.layers.**BatchNormLayer**(*layer=None*, *decay=0.9*, *epsilon=1e-05*, *act=<function identity>*, *is_train=False*, *beta_init=<class 'tensorflow.python.ops.init_ops.Zeros'>*, *gamma_init=<tensorflow.python.ops.init_ops.RandomNormal object>*, *name='batchnorm_layer'*)

The *BatchNormLayer* class is a normalization layer, see `tf.nn.batch_normalization` and `tf.nn.moments`.

Batch normalization on fully-connected or convolutional maps.

Parameters

layer [a *Layer* instance] The *Layer* class feeding into this layer.

decay [float, default is 0.9.] A decay factor for ExponentialMovingAverage, use larger value for large dataset.

epsilon [float] A small float number to avoid dividing by 0.

act [activation function.]

is_train [boolean] Whether train or inference.

beta_init [beta initializer] The initializer for initializing beta

gamma_init [gamma initializer] The initializer for initializing gamma

dtype [tf.float32 (default) or tf.float16]

name [a string or None] An optional name to attach to this layer.

References

- Source
- stackoverflow

Methods

count_params()	Return the number of parameters in the network

Continued on next page

Table 25 – continued from previous page

print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

Local Response Normalization

class tensorlayer.layers.**LocalResponseNormLayer**(*layer=None*, *depth_radius=None*, *bias=None*, *alpha=None*, *beta=None*, *name='lrn_layer'*)

The *LocalResponseNormLayer* class is for Local Response Normalization, see tf.nn.local_response_normalization or tf.nn.lrn for new TF version. The 4-D input tensor is treated as a 3-D array of 1-D vectors (along the last dimension), and each vector is normalized independently. Within a given vector, each component is divided by the weighted, squared sum of inputs within depth_radius.

Parameters

layer [a layer class. Must be one of the following types: float32, half. 4-D.]

depth_radius [An optional int. Defaults to 5. 0-D. Half-width of the 1-D normalization window.]

bias [An optional float. Defaults to 1. An offset (usually positive to avoid dividing by 0).]

alpha [An optional float. Defaults to 1. A scale factor, usually positive.]

beta [An optional float. Defaults to 0.5. An exponent.]

name [A string or None, an optional name to attach to this layer.]

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

Instance Normalization

class tensorlayer.layers.**InstanceNormLayer**(*layer=None*, *act=<function identity>*, *epsilon=1e-05*, *scale_init=<tensorflow.python.ops.init_ops.TruncatedNormal object>*, *offset_init=<tensorflow.python.ops.init_ops.Constant object>*, *name='instan_norm'*)

The *InstanceNormLayer* class is a for instance normalization.

Parameters

layer [a *Layer* instance] The *Layer* class feeding into this layer.

act [activation function.]

epsilon [float] A small float number.

scale_init [beta initializer] The initializer for initializing beta

offset_init [gamma initializer] The initializer for initializing gamma

name [a string or None] An optional name to attach to this layer.

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

Layer Normalization

class tensorlayer.layers.**LayerNormLayer**(*layer=None*, *center=True*, *scale=True*, *act=<function identity>*, *reuse=None*, *variables_collections=None*, *outputs_collections=None*, *trainable=True*, *begin_norm_axis=1*, *begin_params_axis=-1*, *name='layernorm'*)

The *LayerNormLayer* class is for layer normalization, see tf.contrib.layers.layer_norm.

Parameters

layer [a *Layer* instance] The *Layer* class feeding into this layer.

act [activation function] The function that is applied to the layer activations.

others [see tf.contrib.layers.layer_norm]

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

2.1.18 Object Detection

ROI layer

class tensorlayer.layers.**ROIPoolingLayer**(*layer=None*, *rois=None*, *pool_height=2*, *pool_width=2*, *name='roipooling_layer'*)

The *ROIPoolingLayer* class is Region of interest pooling layer.

Parameters

layer [a *Layer* instance] The *Layer* class feeding into this layer, the feature maps on which to perform the pooling operation

rois [list of regions of interest in the format (feature map index, upper left, bottom right)]

pool_width [int, size of the pooling sections.]

pool_width [int, size of the pooling sections.]

Notes

- This implementation is from Deepsense-AI .
- Please install it by the instruction HERE.

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

2.1.19 Time distributed layer

class tensorlayer.layers.**TimeDistributedLayer**(*layer=None*, *layer_class=None*, *args={}*, *name='time_distributed'*)

The *TimeDistributedLayer* class that applies a function to every timestep of the input tensor. For example, if using *DenseLayer* as the layer_class, inputs [batch_size , length, dim] outputs [batch_size , length, new_dim].

Parameters

layer [a *Layer* instance] The *Layer* class feeding into this layer, [batch_size , length, dim]

layer_class [a *Layer* class]

args [dictionary] The arguments for the layer_class.

name [a string or None] An optional name to attach to this layer.

Examples

```
>>> batch_size = 32
>>> timestep = 20
>>> input_dim = 100
>>> x = tf.placeholder(dtype=tf.float32, shape=[batch_size, timestep,  input_dim],
↪ name="encode_seqs")
>>> net = InputLayer(x, name='input')
>>> net = TimeDistributedLayer(net, layer_class=DenseLayer, args={'n_units':50,
↪'name':'dense'}, name='time_dense')
... [TL] InputLayer  input: (32, 20, 100)
... [TL] TimeDistributedLayer time_dense: layer_class:DenseLayer
>>> print(net.outputs._shape)
... (32, 20, 50)
>>> net.print_params(False)
... param   0: (100, 50)          time_dense/dense/W:0
... param   1: (50,)              time_dense/dense/b:0
... num of params: 5050
```

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

2.1.20 Fixed Length Recurrent layer

All recurrent layers can implement any type of RNN cell by feeding different cell function (LSTM, GRU etc).

RNN layer

class tensorlayer.layers.**RNNLayer**(*layer=None*, *cell_fn=None*, *cell_init_args={}*, *n_hidden=100*, *initializer=<tensorflow.python.ops.init_ops.RandomUniform object>*, *n_steps=5*, *initial_state=None*, *return_last=False*, *return_seq_2d=False*, *name='rnn_layer'*)

The *RNNLayer* class is a RNN layer, you can implement vanilla RNN, LSTM and GRU with it.

Parameters

layer [a *Layer* instance] The *Layer* class feeding into this layer.

cell_fn [a TensorFlow's core RNN cell as follow (Note TF1.0+ and TF1.0- are different).]

- see RNN Cells in TensorFlow

cell_init_args [a dictionary] The arguments for the cell initializer.

n_hidden [an int] The number of hidden units in the layer.

initializer [initializer] The initializer for initializing the parameters.

n_steps [an int] The sequence length.

initial_state [None or RNN State] If None, initial_state is zero_state.

return_last [boolean]

- If True, return the last output, "Sequence input and single output"
- If False, return all outputs, "Synced sequence input and output"
- In other word, if you want to apply one or more RNN(s) on this layer, set to False.

return_seq_2d [boolean]

- When return_last = False
- If True, return 2D Tensor [n_example, n_hidden], for stacking DenseLayer after it.
- If False, return 3D Tensor [n_example/n_steps, n_steps, n_hidden], for stacking multiple RNN after it.

name [a string or None] An optional name to attach to this layer.

Notes

Input dimension should be rank 3 : [batch_size, n_steps, n_features], if no, please see *ReshapeLayer*.

References

- Neural Network RNN Cells in TensorFlow
- tensorflow/python/ops/rnn.py
- tensorflow/python/ops/rnn_cell.py
- see TensorFlow tutorial `ptb_word_lm.py`, TensorLayer tutorials `tutorial_ptb_lstm*.py` and `tutorial_generate_text.py`

Examples

- For words

```
>>> input_data = tf.placeholder(tf.int32, [batch_size, num_steps])
>>> net = tl.layers.EmbeddingInputlayer(
...                 inputs = input_data,
...                 vocabulary_size = vocab_size,
...                 embedding_size = hidden_size,
...                 E_init = tf.random_uniform_initializer(-init_scale, init_
↪scale),
...                 name ='embedding_layer')
>>> net = tl.layers.DropoutLayer(net, keep=keep_prob, is_fix=True, is_train=is_
↪train, name='drop1')
>>> net = tl.layers.RNNLayer(net,
...             cell_fn=tf.contrib.rnn.BasicLSTMCell,
...             cell_init_args={'forget_bias': 0.0},# 'state_is_tuple': True},
...             n_hidden=hidden_size,
...             initializer=tf.random_uniform_initializer(-init_scale, init_
↪scale),
...             n_steps=num_steps,
...             return_last=False,
...             name='basic_lstm_layer1')
>>> lstm1 = net
>>> net = tl.layers.DropoutLayer(net, keep=keep_prob, is_fix=True, is_train=is_
↪train, name='drop2')
>>> net = tl.layers.RNNLayer(net,
...             cell_fn=tf.contrib.rnn.BasicLSTMCell,
...             cell_init_args={'forget_bias': 0.0}, # 'state_is_tuple': True},
...             n_hidden=hidden_size,
...             initializer=tf.random_uniform_initializer(-init_scale, init_
↪scale),
...             n_steps=num_steps,
...             return_last=False,
...             return_seq_2d=True,
...             name='basic_lstm_layer2')
>>> lstm2 = net
>>> net = tl.layers.DropoutLayer(net, keep=keep_prob, is_fix=True, is_train=is_
↪train, name='drop3')
>>> net = tl.layers.DenseLayer(net,
...             n_units=vocab_size,
...             W_init=tf.random_uniform_initializer(-init_scale, init_scale),
...             b_init=tf.random_uniform_initializer(-init_scale, init_scale),
...             act = tl.activation.identity, name='output_layer')
```

- For CNN+LSTM

```
>>> x = tf.placeholder(tf.float32, shape=[batch_size, image_size, image_size, 1])
>>> net = tl.layers.InputLayer(x, name='input_layer')
>>> net = tl.layers.Conv2dLayer(net,
...                         act = tf.nn.relu,
...                         shape = [5, 5, 1, 32],  # 32 features for each 5x5␣
↪patch
...                         strides=[1, 2, 2, 1],
...                         padding='SAME',
...                         name ='cnn_layer1')
```

(continues on next page)

(continued from previous page)

```
>>> net = tl.layers.PoolLayer(net,
...                           ksize=[1, 2, 2, 1],
...                           strides=[1, 2, 2, 1],
...                           padding='SAME',
...                           pool = tf.nn.max_pool,
...                           name ='pool_layer1')
>>> net = tl.layers.Conv2dLayer(net,
...                           act = tf.nn.relu,
...                           shape = [5, 5, 32, 10], # 10 features for each 5x5␣
↪patch
...                           strides=[1, 2, 2, 1],
...                           padding='SAME',
...                           name ='cnn_layer2')
>>> net = tl.layers.PoolLayer(net,
...                           ksize=[1, 2, 2, 1],
...                           strides=[1, 2, 2, 1],
...                           padding='SAME',
...                           pool = tf.nn.max_pool,
...                           name ='pool_layer2')
>>> net = tl.layers.FlattenLayer(net, name='flatten_layer')
>>> net = tl.layers.ReshapeLayer(net, shape=[-1, num_steps, int(net.outputs._
↪shape[-1])])
>>> rnn1 = tl.layers.RNNLayer(net,
...                           cell_fn=tf.nn.rnn_cell.LSTMCell,
...                           cell_init_args={},
...                           n_hidden=200,
...                           initializer=tf.random_uniform_initializer(-0.1, 0.1),
...                           n_steps=num_steps,
...                           return_last=False,
...                           return_seq_2d=True,
...                           name='rnn_layer')
>>> net = tl.layers.DenseLayer(rnn1, n_units=3,
...                           act = tl.activation.identity, name='output_layer')
```

Attributes

outputs [a tensor] The output of this RNN. return_last = False, outputs = all cell_output, which is the hidden state.

cell_output.get_shape() = (?, n_hidden)

final_state [a tensor or StateTuple] When state_is_tuple = False, it is the final hidden and cell states, states.get_shape() = [?, 2 * n_hidden].

When state_is_tuple = True, it stores two elements: (c, h), in that order. You can get the final state after each iteration during training, then feed it to the initial state of next iteration.

initial_state [a tensor or StateTuple] It is the initial state of this RNN layer, you can use it to initialize your state at the begining of each epoch or iteration according to your training procedure.

batch_size [int or tensor] Is int, if able to compute the batch_size, otherwise, tensor for `?`.

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

Bidirectional layer

class tensorlayer.layers.**BiRNNLayer**(*layer=None*, *cell_fn=None*, *cell_init_args={'state_is_tuple': True, 'use_peepholes': True}*, *n_hidden=100*, *initializer=<tensorflow.python.ops.init_ops.RandomUniform object>*, *n_steps=5*, *fw_initial_state=None*, *bw_initial_state=None*, *dropout=None*, *n_layer=1*, *return_last=False*, *return_seq_2d=False*, *name='birnn_layer'*)

The *BiRNNLayer* class is a Bidirectional RNN layer.

Parameters

layer [a *Layer* instance] The *Layer* class feeding into this layer.

cell_fn [a TensorFlow's core RNN cell as follow (Note TF1.0+ and TF1.0- are different).]

- see RNN Cells in TensorFlow

cell_init_args [a dictionary] The arguments for the cell initializer.

n_hidden [an int] The number of hidden units in the layer.

initializer [initializer] The initializer for initializing the parameters.

n_steps [an int] The sequence length.

fw_initial_state [None or forward RNN State] If None, initial_state is zero_state.

bw_initial_state [None or backward RNN State] If None, initial_state is zero_state.

dropout [*tuple* of *float*: (input_keep_prob, output_keep_prob).] The input and output keep probability.

n_layer [an int, default is 1.] The number of RNN layers.

return_last [boolean]

- If True, return the last output, "Sequence input and single output"
- If False, return all outputs, "Synced sequence input and output"
- In other word, if you want to apply one or more RNN(s) on this layer, set to False.

return_seq_2d [boolean]

- When return_last = False
- If True, return 2D Tensor [n_example, n_hidden], for stacking DenseLayer after it.
- If False, return 3D Tensor [n_example/n_steps, n_steps, n_hidden], for stacking multiple RNN after it.

name [a string or None] An optional name to attach to this layer.

Notes

- Input dimension should be rank 3 : [batch_size, n_steps, n_features], if no, please see *ReshapeLayer*.
- For predicting, the sequence length has to be the same with the sequence length of training, while, for normal

RNN, we can use sequence length of 1 for predicting.

References

- Source

Attributes

outputs [a tensor] The output of this RNN. return_last = False, outputs = all cell_output, which is the hidden state.

cell_output.get_shape() = (?, n_hidden)

fw(bw)_final_state [a tensor or StateTuple] When state_is_tuple = False, it is the final hidden and cell states, states.get_shape() = [?, 2 * n_hidden].

When state_is_tuple = True, it stores two elements: (c, h), in that order. You can get the final state after each iteration during training, then feed it to the initial state of next iteration.

fw(bw)_initial_state [a tensor or StateTuple] It is the initial state of this RNN layer, you can use it to initialize your state at the begining of each epoch or iteration according to your training procedure.

batch_size [int or tensor] Is int, if able to compute the batch_size, otherwise, tensor for `?`.

Methods

`count_params`()	Return the number of parameters in the network
`print_layers`()	Print all info of layers in the network
`print_params`([details, session])	Print all info of parameters in the network

2.1.21 Recurrent Convolutional layer

Conv RNN Cell

class `tensorlayer.layers.`**ConvRNNCell**

Abstract object representing an Convolutional RNN Cell.

Attributes

output_size Integer or TensorShape: size of outputs produced by this cell.

state_size size(s) of state(s) used by this cell.

Methods

`__call__`(inputs, state[, scope])	Run this RNN cell on inputs, starting from the given state.
`zero_state`(batch_size, dtype)	Return zero-filled state tensor(s).

Basic Conv LSTM Cell

class tensorlayer.layers.**BasicConvLSTMCell**(*shape*, *filter_size*, *num_features*, *forget_bias=1.0*, *input_size=None*, *state_is_tuple=False*, *activation=<function tanh>*)

Basic Conv LSTM recurrent network cell.

Parameters

shape [int tuple thats the height and width of the cell]

filter_size [int tuple thats the height and width of the filter]

num_features [int thats the depth of the cell]

forget_bias [float, The bias added to forget gates (see above).]

input_size [Deprecated and unused.]

state_is_tuple [If True, accepted and returned states are 2-tuples of] the *c_state* and *m_state*. If False, they are concatenated along the column axis. The latter behavior will soon be deprecated.

activation [Activation function of the inner states.]

Attributes

output_size Number of units in outputs.

state_size State size of the LSTMStateTuple.

Methods

`__call__`(inputs, state[, scope])	Long short-term memory cell (LSTM).
`zero_state`(batch_size, dtype)	Return zero-filled state tensor(s).

Conv LSTM layer

class tensorlayer.layers.**ConvLSTMLayer**(*layer=None*, *cell_shape=None*, *feature_map=1*, *filter_size=(3, 3)*, *cell_fn=<class 'tensorlayer.layers.BasicConvLSTMCell'>*, *initializer=<tensorflow.python.ops.init_ops.RandomUniform object>*, *n_steps=5*, *initial_state=None*, *return_last=False*, *return_seq_2d=False*, *name='convlstm_layer'*)

The *ConvLSTMLayer* class is a Convolutional LSTM layer, see Convolutional LSTM Layer .

Parameters

layer [a *Layer* instance] The *Layer* class feeding into this layer.

cell_shape [tuple, the shape of each cell width*height]

filter_size [tuple, the size of filter width*height]

cell_fn [a Convolutional RNN cell as follow.]

feature_map [a int] The number of feature map in the layer.

initializer [initializer] The initializer for initializing the parameters.

n_steps [a int] The sequence length.

initial_state [None or ConvLSTM State] If None, initial_state is zero_state.

return_last [boolen]

- If True, return the last output, "Sequence input and single output"
- If False, return all outputs, "Synced sequence input and output"
- In other word, if you want to apply one or more ConvLSTM(s) on this layer, set to False.

return_seq_2d [boolen]

- When return_last = False
- If True, return 4D Tensor [n_example, h, w, c], for stacking DenseLayer after it.
- If False, return 5D Tensor [n_example/n_steps, h, w, c], for stacking multiple ConvLSTM after it.

name [a string or None] An optional name to attach to this layer.

Methods

`count_params`()	Return the number of parameters in the network
`print_layers`()	Print all info of layers in the network
`print_params`([details, session])	Print all info of parameters in the network

2.1.22 Advanced Ops for Dynamic RNN

These operations usually be used inside Dynamic RNN layer, they can compute the sequence lengths for different situation and get the last RNN outputs by indexing.

Output indexing

`tensorlayer.layers.`**`advanced_indexing_op`**(*input*, *index*)

Advanced Indexing for Sequences, returns the outputs by given sequence lengths. When return the last output *DynamicRNNLayer* uses it to get the last outputs with the sequence lengths.

Parameters

input [tensor for data] [batch_size, n_step(max), n_features]

index [tensor for indexing, i.e. sequence_length in Dynamic RNN.] [batch_size]

References

- Modified from TFlearn (the original code is used for fixed length rnn), references.

Examples

```
>>> batch_size, max_length, n_features = 3, 5, 2
>>> z = np.random.uniform(low=-1, high=1, size=[batch_size, max_length, n_
↪features]).astype(np.float32)
>>> b_z = tf.constant(z)
>>> sl = tf.placeholder(dtype=tf.int32, shape=[batch_size])
>>> o = advanced_indexing_op(b_z, sl)
>>>
>>> sess = tf.InteractiveSession()
>>> tl.layers.initialize_global_variables(sess)
>>>
>>> order = np.asarray([1,1,2])
>>> print("real",z[0][order[0]-1], z[1][order[1]-1], z[2][order[2]-1])
>>> y = sess.run([o], feed_dict={sl:order})
>>> print("given",order)
>>> print("out", y)
... real [-0.93021595  0.53820813] [-0.92548317 -0.77135968] [ 0.89952248  0.
↪19149846]
... given [1 1 2]
... out [array([[-0.93021595,  0.53820813],
...              [-0.92548317, -0.77135968],
...              [ 0.89952248,  0.19149846]], dtype=float32)]
```

Compute Sequence length 1

tensorlayer.layers.**retrieve_seq_length_op**(*data*)

An op to compute the length of a sequence from input shape of [batch_size, n_step(max), n_features], it can be used when the features of padding (on right hand side) are all zeros.

Parameters

data [tensor] [batch_size, n_step(max), n_features] with zero padding on right hand side.

References

- Borrow from TFlearn.

Examples

```
>>> data = [[[1],[2],[0],[0],[0]],
...         [[1],[2],[3],[0],[0]],
...         [[1],[2],[6],[1],[0]]]
>>> data = np.asarray(data)
>>> print(data.shape)
... (3, 5, 1)
>>> data = tf.constant(data)
>>> sl = retrieve_seq_length_op(data)
>>> sess = tf.InteractiveSession()
>>> tl.layers.initialize_global_variables(sess)
>>> y = sl.eval()
... [2 3 4]
```

- Multiple features

```
>>> data = [[[1,2],[2,2],[1,2],[1,2],[0,0]],
...         [[2,3],[2,4],[3,2],[0,0],[0,0]],
...         [[3,3],[2,2],[5,3],[1,2],[0,0]]]
>>> print(sl)
... [4 3 4]
```

Compute Sequence length 2

tensorlayer.layers.**retrieve_seq_length_op2**(*data*)

An op to compute the length of a sequence, from input shape of [batch_size, n_step(max)], it can be used when the features of padding (on right hand side) are all zeros.

Parameters

data [tensor] [batch_size, n_step(max)] with zero padding on right hand side.

Examples

```
>>> data = [[1,2,0,0,0],
...         [1,2,3,0,0],
...         [1,2,6,1,0]]
>>> o = retrieve_seq_length_op2(data)
>>> sess = tf.InteractiveSession()
>>> tl.layers.initialize_global_variables(sess)
>>> print(o.eval())
... [2 3 4]
```

2.1.23 Dynamic RNN layer

RNN layer

class tensorlayer.layers.**DynamicRNNLayer**(*layer=None*, *cell_fn=None*, *cell_init_args={'state_is_tuple': True}*, *n_hidden=256*, *initializer=<tensorflow.python.ops.init_ops.RandomUniform object>*, *sequence_length=None*, *initial_state=None*, *dropout=None*, *n_layer=1*, *return_last=False*, *return_seq_2d=False*, *dynamic_rnn_init_args={}*, *name='dyrnn_layer'*)

The *DynamicRNNLayer* class is a Dynamic RNN layer, see `tf.nn.dynamic_rnn`.

Parameters

layer [a *Layer* instance] The *Layer* class feeding into this layer.

cell_fn [a TensorFlow's core RNN cell as follow (Note TF1.0+ and TF1.0- are different).]

- see RNN Cells in TensorFlow

cell_init_args [a dictionary] The arguments for the cell initializer.

n_hidden [an int] The number of hidden units in the layer.

initializer [initializer] The initializer for initializing the parameters.

sequence_length [a tensor, array or None. The sequence length of each row of input data, see `Advanced Ops for Dynamic RNN`.]

- If None, it uses `retrieve_seq_length_op` to compute the sequence_length, i.e. when the features of padding (on right hand side) are all zeros.
- If using word embedding, you may need to compute the sequence_length from the ID array (the integer features before word embedding) by using `retrieve_seq_length_op2` or `retrieve_seq_length_op`.
- You can also input an numpy array.
- More details about TensorFlow dynamic_rnn in Wild-ML Blog.

initial_state [None or RNN State] If None, initial_state is zero_state.

dropout [*tuple* of *float*: (input_keep_prob, output_keep_prob).] The input and output keep probability.

n_layer [an int, default is 1.] The number of RNN layers.

return_last [boolean]

- If True, return the last output, "Sequence input and single output"
- If False, return all outputs, "Synced sequence input and output"
- In other word, if you want to apply one or more RNN(s) on this layer, set to False.

return_seq_2d [boolean]

- When return_last = False
- If True, return 2D Tensor [n_example, n_hidden], for stacking DenseLayer or computing cost after it.
- If False, return 3D Tensor [n_example/n_steps(max), n_steps(max), n_hidden], for stacking multiple RNN after it.

name [a string or None] An optional name to attach to this layer.

Notes

Input dimension should be rank 3 : [batch_size, n_steps(max), n_features], if no, please see *ReshapeLayer*.

References

- Wild-ML Blog
- dynamic_rnn.ipynb
- tf.nn.dynamic_rnn
- tflearn rnn
- `tutorial_dynamic_rnn.py`

Examples

```
>>> input_seqs = tf.placeholder(dtype=tf.int64, shape=[batch_size, None], name=
↪"input_seqs")
>>> net = tl.layers.EmbeddingInputlayer(
...             inputs = input_seqs,
...             vocabulary_size = vocab_size,
...             embedding_size = embedding_size,
...             name = 'seq_embedding')
>>> net = tl.layers.DynamicRNNLayer(net,
...             cell_fn = tf.contrib.rnn.BasicLSTMCell, # for TF0.2 tf.nn.rnn_
↪cell.BasicLSTMCell,
...             n_hidden = embedding_size,
...             dropout = 0.7,
...             sequence_length = tl.layers.retrieve_seq_length_op2(input_seqs),
...             return_seq_2d = True,     # stack denselayer or compute cost␣
↪after it
...             name = 'dynamic_rnn')
... net = tl.layers.DenseLayer(net, n_units=vocab_size,
...             act=tf.identity, name="output")
```

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

Bidirectional layer

class tensorlayer.layers.**BiDynamicRNNLayer**(*layer=None*, *cell_fn=None*, *cell_init_args={'state_is_tuple': True}*, *n_hidden=256*, *initializer=<tensorflow.python.ops.init_ops.RandomUniform object>*, *sequence_length=None*, *fw_initial_state=None*, *bw_initial_state=None*, *dropout=None*, *n_layer=1*, *return_last=False*, *return_seq_2d=False*, *dynamic_rnn_init_args={}*, *name='bi_dyrnn_layer'*)

The *BiDynamicRNNLayer* class is a RNN layer, you can implement vanilla RNN, LSTM and GRU with it.

Parameters

layer [a *Layer* instance] The *Layer* class feeding into this layer.

cell_fn [a TensorFlow's core RNN cell as follow (Note TF1.0+ and TF1.0- are different).]

- see RNN Cells in TensorFlow

cell_init_args [a dictionary] The arguments for the cell initializer.

n_hidden [an int] The number of hidden units in the layer.

initializer [initializer] The initializer for initializing the parameters.

sequence_length [a tensor, array or None.]

The sequence length of each row of input data, see `Advanced Ops for Dynamic RNN`.

- If None, it uses `retrieve_seq_length_op` to compute the sequence_length, i.e. when the features of padding (on right hand side) are all zeros.
- If using word embedding, you may need to compute the sequence_length from the ID array (the integer features before word embedding) by using `retrieve_seq_length_op2` or `retrieve_seq_length_op`.
- You can also input an numpy array.
- More details about TensorFlow dynamic_rnn in Wild-ML Blog.

fw_initial_state [None or forward RNN State] If None, initial_state is zero_state.

bw_initial_state [None or backward RNN State] If None, initial_state is zero_state.

dropout [*tuple* of *float*: (input_keep_prob, output_keep_prob).] The input and output keep probability.

n_layer [an int, default is 1.] The number of RNN layers.

return_last [boolean] If True, return the last output, "Sequence input and single output"

If False, return all outputs, "Synced sequence input and output"

In other word, if you want to apply one or more RNN(s) on this layer, set to False.

return_seq_2d [boolean]

- When return_last = False
- If True, return 2D Tensor [n_example, 2 * n_hidden], for stacking DenseLayer or computing cost after it.
- If False, return 3D Tensor [n_example/n_steps(max), n_steps(max), 2 * n_hidden], for stacking multiple RNN after it.

name [a string or None] An optional name to attach to this layer.

Notes

Input dimension should be rank 3 : [batch_size, n_steps(max), n_features], if no, please see *`ReshapeLayer`*.

References

- Wild-ML Blog
- bidirectional_rnn.ipynb

Attributes

outputs [a tensor] The output of this RNN. return_last = False, outputs = all cell_output, which is the hidden state.

cell_output.get_shape() = (?, 2 * n_hidden)

fw(bw)_final_state [a tensor or StateTuple] When state_is_tuple = False, it is the final hidden and cell states, states.get_shape() = [?, 2 * n_hidden].

When state_is_tuple = True, it stores two elements: (c, h), in that order. You can get the final state after each iteration during training, then feed it to the initial state of next iteration.

fw(bw)_initial_state [a tensor or StateTuple] It is the initial state of this RNN layer, you can use it to initialize your state at the begining of each epoch or iteration according to your training procedure.

sequence_length [a tensor or array, shape = [batch_size]] The sequence lengths computed by Advanced Opt or the given sequence lengths.

Methods

`count_params`()	Return the number of parameters in the network
`print_layers`()	Print all info of layers in the network
`print_params`([details, session])	Print all info of parameters in the network

2.1.24 Sequence to Sequence

Simple Seq2Seq

class tensorlayer.layers.**Seq2Seq**(*net_encode_in=None*, *net_decode_in=None*, *cell_fn=None*, *cell_init_args={'state_is_tuple': True}*, *n_hidden=256*, *initializer=<tensorflow.python.ops.init_ops.RandomUniform object>*, *encode_sequence_length=None*, *decode_sequence_length=None*, *initial_state_encode=None*, *initial_state_decode=None*, *dropout=None*, *n_layer=1*, *return_seq_2d=False*, *name='seq2seq'*)

The *Seq2Seq* class is a Simple *DynamicRNNLayer* based Seq2seq layer without using tl.contrib.seq2seq. See Model and Sequence to Sequence Learning with Neural Networks.

- Please check the example Chatbot in 200 lines of code.
- The Author recommends users to read the source code of *DynamicRNNLayer* and *Seq2Seq*.

Parameters

net_encode_in [a *Layer* instance] Encode sequences, [batch_size, None, n_features].

net_decode_in [a *Layer* instance] Decode sequences, [batch_size, None, n_features].

cell_fn [a TensorFlow's core RNN cell as follow (Note TF1.0+ and TF1.0- are different).]

- see RNN Cells in TensorFlow

cell_init_args [a dictionary] The arguments for the cell initializer.

n_hidden [an int] The number of hidden units in the layer.

initializer [initializer] The initializer for initializing the parameters.

encode_sequence_length [tensor for encoder sequence length, see *DynamicRNNLayer* .]

decode_sequence_length [tensor for decoder sequence length, see *DynamicRNNLayer* .]

initial_state_encode [None or RNN state (from placeholder or other RNN).] If None, initial_state_encode is of zero state.

initial_state_decode [None or RNN state (from placeholder or other RNN).] If None, initial_state_decode is of the final state of the RNN encoder.

dropout [*tuple* of *float*: (input_keep_prob, output_keep_prob).] The input and output keep probability.

n_layer [an int, default is 1.] The number of RNN layers.

return_seq_2d [boolean]

- When return_last = False
- If True, return 2D Tensor [n_example, n_hidden], for stacking DenseLayer or computing cost after it.
- If False, return 3D Tensor [n_example/n_steps(max), n_steps(max), n_hidden], for stacking multiple RNN after it.

name [a string or None] An optional name to attach to this layer.

Notes

- How to feed data: Sequence to Sequence Learning with Neural Networks
- input_seqs : `['how', 'are', 'you', '<PAD_ID'>]`
- decode_seqs : `['<START_ID>', 'I', 'am', 'fine', '<PAD_ID'>]`
- target_seqs : `['I', 'am', 'fine', '<END_ID', '<PAD_ID'>]`
- target_mask : `[1, 1, 1, 1, 0]`
- related functions : tl.prepro <pad_sequences, precess_sequences, sequences_add_start_id, sequences_get_mask>

Examples

```
>>> from tensorlayer.layers import *
>>> batch_size = 32
>>> encode_seqs = tf.placeholder(dtype=tf.int64, shape=[batch_size, None], name=
↪"encode_seqs")
>>> decode_seqs = tf.placeholder(dtype=tf.int64, shape=[batch_size, None], name=
↪"decode_seqs")
>>> target_seqs = tf.placeholder(dtype=tf.int64, shape=[batch_size, None], name=
↪"target_seqs")
>>> target_mask = tf.placeholder(dtype=tf.int64, shape=[batch_size, None], name=
↪"target_mask") # tl.prepro.sequences_get_mask()
>>> with tf.variable_scope("model"):
...     # for chatbot, you can use the same embedding layer,
...     # for translation, you may want to use 2 seperated embedding layers
>>>     with tf.variable_scope("embedding") as vs:
>>>         net_encode = EmbeddingInputlayer(
...                 inputs = encode_seqs,
...                 vocabulary_size = 10000,
...                 embedding_size = 200,
...                 name = 'seq_embedding')
>>>         vs.reuse_variables()
>>>         tl.layers.set_name_reuse(True)
>>>         net_decode = EmbeddingInputlayer(
```

(continues on next page)

(continued from previous page)

```
...                 inputs = decode_seqs,
...                 vocabulary_size = 10000,
...                 embedding_size = 200,
...                 name = 'seq_embedding')
>>>     net = Seq2Seq(net_encode, net_decode,
...             cell_fn = tf.contrib.rnn.BasicLSTMCell,
...             n_hidden = 200,
...             initializer = tf.random_uniform_initializer(-0.1, 0.1),
...             encode_sequence_length = retrieve_seq_length_op2(encode_seqs),
...             decode_sequence_length = retrieve_seq_length_op2(decode_seqs),
...             initial_state_encode = None,
...             dropout = None,
...             n_layer = 1,
...             return_seq_2d = True,
...             name = 'seq2seq')
>>> net_out = DenseLayer(net, n_units=10000, act=tf.identity, name='output')
>>> e_loss = tl.cost.cross_entropy_seq_with_mask(logits=net_out.outputs, target_
↪seqs=target_seqs, input_mask=target_mask, return_details=False, name='cost')
>>> y = tf.nn.softmax(net_out.outputs)
>>> net_out.print_params(False)
```

Attributes

outputs [a tensor] The output of RNN decoder.

initial_state_encode [a tensor or StateTuple] Initial state of RNN encoder.

initial_state_decode [a tensor or StateTuple] Initial state of RNN decoder.

final_state_encode [a tensor or StateTuple] Final state of RNN encoder.

final_state_decode [a tensor or StateTuple] Final state of RNN decoder.

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

PeekySeq2Seq

class tensorlayer.layers.**PeekySeq2Seq**(*net_encode_in=None*, *net_decode_in=None*, *cell_fn=None*, *cell_init_args={'state_is_tuple': True}*, *n_hidden=256*, *initializer=<tensorflow.python.ops.init_ops.RandomUniform object>*, *in_sequence_length=None*, *out_sequence_length=None*, *initial_state=None*, *dropout=None*, *n_layer=1*, *return_seq_2d=False*, *name='peeky_seq2seq'*)

Waiting for contribution. The *PeekySeq2Seq* class, see Model and Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation .

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

AttentionSeq2Seq

class tensorlayer.layers.**AttentionSeq2Seq**(*net_encode_in=None*, *net_decode_in=None*, *cell_fn=None*, *cell_init_args={'state_is_tuple': True}*, *n_hidden=256*, *initializer=<tensorflow.python.ops.init_ops.RandomUniform object>*, *in_sequence_length=None*, *out_sequence_length=None*, *initial_state=None*, *dropout=None*, *n_layer=1*, *return_seq_2d=False*, *name='attention_seq2seq'*)

Waiting for contribution. The *AttentionSeq2Seq* class, see Model and Neural Machine Translation by Jointly Learning to Align and Translate .

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

2.1.25 Shape layer

Flatten layer

class tensorlayer.layers.**FlattenLayer**(*layer=None*, *name='flatten_layer'*)

The *FlattenLayer* class is layer which reshape high-dimension input to a vector. Then we can apply DenseLayer, RNNLayer, ConcatLayer and etc on the top of it.

[batch_size, mask_row, mask_col, n_mask] —> [batch_size, mask_row * mask_col * n_mask]

Parameters

layer [a *Layer* instance] The *Layer* class feeding into this layer.

name [a string or None] An optional name to attach to this layer.

Examples

```
>>> x = tf.placeholder(tf.float32, shape=[None, 28, 28, 1])
>>> net = tl.layers.InputLayer(x, name='input_layer')
>>> net = tl.layers.Conv2dLayer(net,
...                     act = tf.nn.relu,
...                     shape = [5, 5, 32, 64],
...                     strides=[1, 1, 1, 1],
...                     padding='SAME',
```

(continues on next page)

(continued from previous page)

```
...                     name ='cnn_layer')
>>> net = tl.layers.Pool2dLayer(net,
...                     ksize=[1, 2, 2, 1],
...                     strides=[1, 2, 2, 1],
...                     padding='SAME',
...                     pool = tf.nn.max_pool,
...                     name ='pool_layer',)
>>> net = tl.layers.FlattenLayer(net, name='flatten_layer')
```

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

Reshape layer

class tensorlayer.layers.**ReshapeLayer**(*layer=None*, *shape=[]*, *name='reshape_layer'*)

The *ReshapeLayer* class is layer which reshape the tensor.

Parameters

layer [a *Layer* instance] The *Layer* class feeding into this layer.

shape [a list] The output shape.

name [a string or None] An optional name to attach to this layer.

Examples

- The core of this layer is `tf.reshape`.
- Use TensorFlow only :

```
>>> x = tf.placeholder(tf.float32, shape=[None, 3])
>>> y = tf.reshape(x, shape=[-1, 3, 3])
>>> sess = tf.InteractiveSession()
>>> print(sess.run(y, feed_dict={x:[[1,1,1],[2,2,2],[3,3,3],[4,4,4],[5,5,5],[6,6,
↪6]]}))
... [[[ 1.  1.  1.]
... [ 2.  2.  2.]
... [ 3.  3.  3.]]
... [[ 4.  4.  4.]
... [ 5.  5.  5.]
... [ 6.  6.  6.]]]
```

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network

Continued on next page

Table 42 – continued from previous page

print_params([details, session])	Print all info of parameters in the network

Transpose layer

class tensorlayer.layers.**TransposeLayer**(*layer=None*, *perm=None*, *name='transpose'*)

The *TransposeLayer* class transpose the dimension of a teneor, see tf.transpose() .

Parameters

layer [a *Layer* instance] The *Layer* class feeding into this layer.

perm: list, a permutation of the dimensions Similar with numpy.transpose.

name [a string or None] An optional name to attach to this layer.

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

2.1.26 Lambda layer

class tensorlayer.layers.**LambdaLayer**(*layer=None*, *fn=None*, *fn_args={}*, *name='lambda_layer'*)

The *LambdaLayer* class is a layer which is able to use the provided function.

Parameters

layer [a *Layer* instance] The *Layer* class feeding into this layer.

fn [a function] The function that applies to the outputs of previous layer.

fn_args [a dictionary] The arguments for the function (option).

name [a string or None] An optional name to attach to this layer.

Examples

```
>>> x = tf.placeholder(tf.float32, shape=[None, 1], name='x')
>>> net = tl.layers.InputLayer(x, name='input_layer')
>>> net = LambdaLayer(net, lambda x: 2*x, name='lambda_layer')
>>> y = net.outputs
>>> sess = tf.InteractiveSession()
>>> out = sess.run(y, feed_dict={x : [[1],[2]]})
... [[2],[4]]
```

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network

Continued on next page

Table 44 – continued from previous page

print_params([details, session])	Print all info of parameters in the network

2.1.27 Merge layer

Concat layer

class tensorlayer.layers.**ConcatLayer**(*layer=[]*, *concat_dim=1*, *name='concat_layer'*)

The *ConcatLayer* class is layer which concat (merge) two or more tensor by given axis..

Parameters

layer [a list of *Layer* instances] The *Layer* class feeding into this layer.

concat_dim [int] Dimension along which to concatenate.

name [a string or None] An optional name to attach to this layer.

Examples

```
>>> sess = tf.InteractiveSession()
>>> x = tf.placeholder(tf.float32, shape=[None, 784])
>>> inputs = tl.layers.InputLayer(x, name='input_layer')
>>> net1 = tl.layers.DenseLayer(inputs, 800, act=tf.nn.relu, name='relu1_1')
>>> net2 = tl.layers.DenseLayer(inputs, 300, act=tf.nn.relu, name='relu2_1')
>>> net = tl.layers.ConcatLayer([net1, net2], 1, name ='concat_layer')
...     [TL] InputLayer input_layer (?, 784)
...     [TL] DenseLayer relu1_1: 800, relu
...     [TL] DenseLayer relu2_1: 300, relu
...     [TL] ConcatLayer concat_layer, 1100
>>> tl.layers.initialize_global_variables(sess)
>>> net.print_params()
...     param 0: (784, 800) (mean: 0.000021, median: -0.000020 std: 0.035525)
...     param 1: (800,)     (mean: 0.000000, median: 0.000000  std: 0.000000)
...     param 2: (784, 300) (mean: 0.000000, median: -0.000048 std: 0.042947)
...     param 3: (300,)     (mean: 0.000000, median: 0.000000  std: 0.000000)
...     num of params: 863500
>>> net.print_layers()
...     layer 0: ("Relu:0", shape=(?, 800), dtype=float32)
...     layer 1: Tensor("Relu_1:0", shape=(?, 300), dtype=float32)
```

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

Element-wise layer

class tensorlayer.layers.**ElementwiseLayer**(*layer=[]*, *combine_fn=<function minimum>*, *name='elementwise_layer'*)

The *ElementwiseLayer* class combines multiple *Layer* which have the same output shapes by a given elemwise-wise operation.

Parameters

layer [a list of *Layer* instances] The *Layer* class feeding into this layer.

combine_fn [a TensorFlow elemwise-merge function] e.g. AND is `tf.minimum` ; OR is `tf.maximum` ; ADD is `tf.add` ; MUL is `tf.multiply` and so on. See TensorFlow Math API .

name [a string or None] An optional name to attach to this layer.

Examples

- AND Logic

```
>>> net_0 = tl.layers.DenseLayer(net_0, n_units=500,
...                              act = tf.nn.relu, name='net_0')
>>> net_1 = tl.layers.DenseLayer(net_1, n_units=500,
...                              act = tf.nn.relu, name='net_1')
>>> net_com = tl.layers.ElementwiseLayer(layer = [net_0, net_1],
...                               combine_fn = tf.minimum,
...                               name = 'combine_layer')
```

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

2.1.28 Extend layer

Expand dims layer

class tensorlayer.layers.**ExpandDimsLayer**(*layer=None*, *axis=None*, *name='expand_dims'*)

The *ExpandDimsLayer* class inserts a dimension of 1 into a tensor's shape, see tf.expand_dims() .

Parameters

layer [a *Layer* instance] The *Layer* class feeding into this layer.

axis [int, 0-D (scalar).] Specifies the dimension index at which to expand the shape of input.

name [a string or None] An optional name to attach to this layer.

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

Tile layer

class tensorlayer.layers.**TileLayer**(*layer=None*, *multiples=None*, *name='tile'*)

The *TileLayer* class constructs a tensor by tiling a given tensor, see tf.tile() .

Parameters

layer [a *Layer* instance] The *Layer* class feeding into this layer.

multiples: a list of int Must be one of the following types: int32, int64. 1-D. Length must be the same as the number of dimensions in input

name [a string or None] An optional name to attach to this layer.

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

2.1.29 Stack layer

Stack layer

class tensorlayer.layers.**StackLayer**(*layer=[]*, *axis=0*, *name='stack'*)

The *StackLayer* class is layer for stacking a list of rank-R tensors into one rank-(R+1) tensor, see tf.stack().

Parameters

layer [a list of *Layer* instances] The *Layer* class feeding into this layer.

axis [an int] Dimension along which to concatenate.

name [a string or None] An optional name to attach to this layer.

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

Unstack layer

tensorlayer.layers.**UnStackLayer**(*layer=None*, *num=None*, *axis=0*, *name='unstack'*)

The *UnStackLayer* is layer for unstacking the given dimension of a rank-R tensor into rank-(R-1) tensors., see tf.unstack().

Parameters

layer [a list of *Layer* instances] The *Layer* class feeding into this layer.

num [an int] The length of the dimension axis. Automatically inferred if None (the default).

axis [an int] Dimension along which to concatenate.

name [a string or None] An optional name to attach to this layer.

Returns

The list of layer objects unstacked from the input.

2.1.30 Estimator layer

class tensorlayer.layers.**EstimatorLayer**(*layer=None*, *model_fn=None*, *args={}*, *name='estimator_layer'*)

The *EstimatorLayer* class accepts `model_fn` that described the model. It is similar with *KerasLayer*, see tutorial_keras.py. This layer will be deprecated soon as *LambdaLayer* can do the same thing.

Parameters

layer [a *Layer* instance] The *Layer* class feeding into this layer.

model_fn [a function that described the model.]

args [dictionary] The arguments for the model_fn.

name [a string or None] An optional name to attach to this layer.

Methods

`count_params`()	Return the number of parameters in the network
`print_layers`()	Print all info of layers in the network
`print_params`([details, session])	Print all info of parameters in the network

2.1.31 Connect TF-Slim

Yes ! TF-Slim models can be connected into TensorLayer, all Google's Pre-trained model can be used easily , see Slim-model .

class tensorlayer.layers.**SlimNetsLayer**(*layer=None*, *slim_layer=None*, *slim_args={}*, *name='tfslim_layer'*)

The *SlimNetsLayer* class can be used to merge all TF-Slim nets into TensorLayer. Models can be found in slim-model, see Inception V3 example on Github.

Parameters

layer [a *Layer* instance] The *Layer* class feeding into this layer.

slim_layer [a slim network function] The network you want to stack onto, end with `return net, end_points`.

slim_args [dictionary] The arguments for the slim model.

name [a string or None] An optional name to attach to this layer.

Notes

The due to TF-Slim stores the layers as dictionary, the `all_layers` in this network is not in order ! Fortunately, the `all_params` are in order.

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

2.1.32 Connect Keras

Yes ! Keras models can be connected into TensorLayer! see tutorial_keras.py .

class tensorlayer.layers.**KerasLayer**(*layer=None*, *keras_layer=None*, *keras_args={}*, *name='keras_layer'*)

The *KerasLayer* class can be used to merge all Keras layers into TensorLayer. Example can be found here tutorial_keras.py. This layer will be deprecated soon as *LambdaLayer* can do the same thing.

Parameters

layer [a *Layer* instance] The *Layer* class feeding into this layer.

keras_layer [a keras network function]

keras_args [dictionary] The arguments for the keras model.

name [a string or None] An optional name to attach to this layer.

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

2.1.33 Parametric activation layer

class tensorlayer.layers.**PReluLayer**(*layer=None*, *channel_shared=False*, *a_init=<tensorflow.python.ops.init_ops.Constant object>*, *a_init_args={}*, *name='prelu_layer'*)

The *PReluLayer* class is Parametric Rectified Linear layer.

Parameters

x [A *Tensor* with type *float*, *double*, *int32*, *int64*, *uint8*,] *int16*, or *int8*.

channel_shared [*bool*. Single weight is shared by all channels]

a_init [alpha initializer, default zero constant.] The initializer for initializing the alphas.

a_init_args [dictionary] The arguments for the weights initializer.

name [A name for this activation op (optional).]

References

- Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

2.1.34 Flow control layer

class tensorlayer.layers.**MultiplexerLayer**(*layer=[]*, *name='mux_layer'*)

The *MultiplexerLayer* selects one of several input and forwards the selected input into the output, see *tutorial_mnist_multiplexer.py*.

Parameters

layer [a list of *Layer* instances] The *Layer* class feeding into this layer.

name [a string or None] An optional name to attach to this layer.

References

- See tf.pack() for TF0.12 or tf.stack() for TF1.0 and tf.gather() at TensorFlow - Slicing and Joining

Examples

```
>>> x = tf.placeholder(tf.float32, shape=[None, 784], name='x')
>>> y_ = tf.placeholder(tf.int64, shape=[None, ], name='y_')
>>> # define the network
>>> net_in = tl.layers.InputLayer(x, name='input_layer')
>>> net_in = tl.layers.DropoutLayer(net_in, keep=0.8, name='drop1')
>>> # net 0
>>> net_0 = tl.layers.DenseLayer(net_in, n_units=800,
...                                 act = tf.nn.relu, name='net0/relu1')
>>> net_0 = tl.layers.DropoutLayer(net_0, keep=0.5, name='net0/drop2')
>>> net_0 = tl.layers.DenseLayer(net_0, n_units=800,
...                                 act = tf.nn.relu, name='net0/relu2')
>>> # net 1
>>> net_1 = tl.layers.DenseLayer(net_in, n_units=800,
...                                 act = tf.nn.relu, name='net1/relu1')
>>> net_1 = tl.layers.DropoutLayer(net_1, keep=0.8, name='net1/drop2')
>>> net_1 = tl.layers.DenseLayer(net_1, n_units=800,
...                                 act = tf.nn.relu, name='net1/relu2')
>>> net_1 = tl.layers.DropoutLayer(net_1, keep=0.8, name='net1/drop3')
>>> net_1 = tl.layers.DenseLayer(net_1, n_units=800,
...                                 act = tf.nn.relu, name='net1/relu3')
>>> # multiplexer
>>> net_mux = tl.layers.MultiplexerLayer(layer = [net_0, net_1], name='mux_layer')
>>> network = tl.layers.ReshapeLayer(net_mux, shape=[-1, 800], name='reshape_layer
↪') #
>>> network = tl.layers.DropoutLayer(network, keep=0.5, name='drop3')
>>> # output layer
>>> network = tl.layers.DenseLayer(network, n_units=10,
...                                 act = tf.identity, name='output_layer')
```

Methods

count_params()	Return the number of parameters in the network
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network

2.1.35 Wrapper

Embedding + Attention + Seq2seq

class tensorlayer.layers.**EmbeddingAttentionSeq2seqWrapper**(*source_vocab_size*, *target_vocab_size*, *buckets*, *size*, *num_layers*, *max_gradient_norm*, *batch_size*, *learning_rate*, *learning_rate_decay_factor*, *use_lstm=False*, *num_samples=512*, *forward_only=False*, *name='wrapper'*)

Sequence-to-sequence model with attention and for multiple buckets (Deprecated after TF0.12).

This example implements a multi-layer recurrent neural network as encoder, and an attention-based decoder. This is the same as the model described in this paper: - Grammar as a Foreign Language please look there for details, or into the seq2seq library for complete model implementation. This example also allows to use GRU cells in addition to LSTM cells, and sampled softmax to handle large output vocabulary size. A single-layer version of this model, but with bi-directional encoder, was presented in - Neural Machine Translation by Jointly Learning to Align and Translate The sampled softmax is described in Section 3 of the following paper. - On Using Very Large Target Vocabulary for Neural Machine Translation

Parameters

source_vocab_size [size of the source vocabulary.]

target_vocab_size [size of the target vocabulary.]

buckets [a list of pairs (I, O), where I specifies maximum input length] that will be processed in that bucket, and O specifies maximum output length. Training instances that have inputs longer than I or outputs longer than O will be pushed to the next bucket and padded accordingly. We assume that the list is sorted, e.g., [(2, 4), (8, 16)].

size [number of units in each layer of the model.]

num_layers [number of layers in the model.]

max_gradient_norm [gradients will be clipped to maximally this norm.]

batch_size [the size of the batches used during training;] the model construction is independent of batch_size, so it can be changed after initialization if this is convenient, e.g., for decoding.

learning_rate [learning rate to start with.]

learning_rate_decay_factor [decay learning rate by this much when needed.]

use_lstm [if true, we use LSTM cells instead of GRU cells.]

num_samples [number of samples for sampled softmax.]

forward_only [if set, we do not construct the backward pass in the model.]

name [a string or None] An optional name to attach to this layer.

Methods

count_params()	Return the number of parameters in the network
get_batch(data, bucket_id[, PAD_ID, GO_ID, ...])	Get a random batch of data from the specified bucket, prepare for step.
print_layers()	Print all info of layers in the network
print_params([details, session])	Print all info of parameters in the network
step(session, encoder_inputs, ...)	Run a step of the model feeding the given inputs.

get_batch(*data*, *bucket_id*, *PAD_ID=0*, *GO_ID=1*, *EOS_ID=2*, *UNK_ID=3*)
Get a random batch of data from the specified bucket, prepare for step.

To feed data in step(..) it must be a list of batch-major vectors, while data here contains single length-major cases. So the main logic of this function is to re-index data cases to be in the proper format for feeding.

Parameters

data [a tuple of size len(self.buckets) in which each element contains] lists of pairs of input and output data that we use to create a batch.

bucket_id [integer, which bucket to get the batch for.]

PAD_ID [int] Index of Padding in vocabulary

GO_ID [int] Index of GO in vocabulary

EOS_ID [int] Index of End of sentence in vocabulary

UNK_ID [int] Index of Unknown word in vocabulary

Returns

The triple (encoder_inputs, decoder_inputs, target_weights) for

the constructed batch that has the proper format to call step(...) later.

step(*session*, *encoder_inputs*, *decoder_inputs*, *target_weights*, *bucket_id*, *forward_only*)
Run a step of the model feeding the given inputs.

Parameters

session [tensorflow session to use.]

encoder_inputs [list of numpy int vectors to feed as encoder inputs.]

decoder_inputs [list of numpy int vectors to feed as decoder inputs.]

target_weights [list of numpy float vectors to feed as target weights.]

bucket_id [which bucket of the model to use.]

forward_only [whether to do the backward step or only forward.]

Returns

A triple consisting of gradient norm (or None if we did not do backward),

average perplexity, and the outputs.

Raises

ValueError [if length of encoder_inputs, decoder_inputs, or] target_weights disagrees with bucket size for the specified bucket_id.

2.1.36 Helper functions

Flatten tensor

tensorlayer.layers.**flatten_reshape**(*variable*, *name=''*)

Reshapes high-dimension input to a vector. [batch_size, mask_row, mask_col, n_mask] —> [batch_size, mask_row * mask_col * n_mask]

Parameters

variable [a tensorflow variable]

name [a string or None] An optional name to attach to this layer.

Examples

```
>>> W_conv2 = weight_variable([5, 5, 100, 32])    # 64 features for each 5x5 patch
>>> b_conv2 = bias_variable([32])
>>> W_fc1 = weight_variable([7 * 7 * 32, 256])
```

```
>>> h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
>>> h_pool2 = max_pool_2x2(h_conv2)
>>> h_pool2.get_shape()[:].as_list() = [batch_size, 7, 7, 32]
...          [batch_size, mask_row, mask_col, n_mask]
>>> h_pool2_flat = tl.layers.flatten_reshape(h_pool2)
...          [batch_size, mask_row * mask_col * n_mask]
>>> h_pool2_flat_drop = tf.nn.dropout(h_pool2_flat, keep_prob)
...
```

Permanent clear existing layer names

tensorlayer.layers.**clear_layers_name**()

Clear all layer names in set_keep['_layers_name_list'], enable layer name reuse.

Examples

```
>>> network = tl.layers.InputLayer(x, name='input_layer')
>>> network = tl.layers.DenseLayer(network, n_units=800, name='relu1')
...
>>> tl.layers.clear_layers_name()
>>> network2 = tl.layers.InputLayer(x, name='input_layer')
>>> network2 = tl.layers.DenseLayer(network2, n_units=800, name='relu1')
...
```

Initialize RNN state

tensorlayer.layers.**initialize_rnn_state**(*state*, *feed_dict=None*)

Returns the initialized RNN state. The inputs are LSTMStateTuple or State of RNNCells and an optional feed_dict.

Parameters

state [a RNN state.]

feed_dict [None or a dictionary for initializing the state values (optional).] If None, returns the zero state.

Remove repeated items in a list

tensorlayer.layers.**list_remove_repeat**(*l=None*)

Remove the repeated items in a list, and return the processed list. You may need it to create merged layer like Concat, Elementwise and etc.

Parameters

l [a list]

Examples

```
>>> l = [2, 3, 4, 2, 3]
>>> l = list_remove_repeat(l)
... [2, 3, 4]
```

Merge networks attributes

tensorlayer.layers.**merge_networks**(*layers=[]*)

Merge all parameters, layers and dropout probabilities to a *Layer*.

Parameters

layer [list of *Layer* instance] Merge all parameters, layers and dropout probabilities to the first layer in the list.

Examples

```
>>> n1 = ...
>>> n2 = ...
>>> n1 = merge_networks([n1, n2])
```

2.2 API - Cost

To make TensorLayer simple, we minimize the number of cost functions as much as we can. So we encourage you to use TensorFlow's function. For example, you can implement L1, L2 and sum regularization by `tf.nn.l2_loss`, `tf.contrib.layers.l1_regularizer`, `tf.contrib.layers.l2_regularizer` and `tf.contrib.layers.sum_regularizer`, see TensorFlow API.

2.2.1 Your cost function

TensorLayer provides a simple way to create you own cost function. Take a MLP below for example.

```
network = InputLayer(x, name='input')
network = DropoutLayer(network, keep=0.8, name='drop1')
network = DenseLayer(network, n_units=800, act=tf.nn.relu, name='relu1')
network = DropoutLayer(network, keep=0.5, name='drop2')
network = DenseLayer(network, n_units=800, act=tf.nn.relu, name='relu2')
network = DropoutLayer(network, keep=0.5, name='drop3')
network = DenseLayer(network, n_units=10, act=tf.identity, name='output')
```

The network parameters will be `[W1, b1, W2, b2, W_out, b_out]`, then you can apply L2 regularization on the weights matrix of first two layer as follow.

```
cost = tl.cost.cross_entropy(y, y_)
cost = cost + tf.contrib.layers.l2_regularizer(0.001)(network.all_params[0]) + tf.
↪contrib.layers.l2_regularizer(0.001)(network.all_params[2])
```

Besides, TensorLayer provides a easy way to get all variables by a given name, so you can also apply L2 regularization on some weights as follow.

```
l2 = 0
for w in tl.layers.get_variables_with_name('W_conv2d', train_only=True,␣
↪printable=False):
    l2 += tf.contrib.layers.l2_regularizer(1e-4)(w)
cost = tl.cost.cross_entropy(y, y_) + l2
```

Regularization of Weights

After initializing the variables, the informations of network parameters can be observed by using `network.print_params()`.

```
tl.layers.initialize_global_variables(sess)
network.print_params()
```

```
param 0: (784, 800) (mean: -0.000000, median: 0.000004 std: 0.035524)
param 1: (800,) (mean: 0.000000, median: 0.000000 std: 0.000000)
param 2: (800, 800) (mean: 0.000029, median: 0.000031 std: 0.035378)
param 3: (800,) (mean: 0.000000, median: 0.000000 std: 0.000000)
param 4: (800, 10) (mean: 0.000673, median: 0.000763 std: 0.049373)
param 5: (10,) (mean: 0.000000, median: 0.000000 std: 0.000000)
num of params: 1276810
```

The output of network is `network.outputs`, then the cross entropy can be defined as follow. Besides, to regularize the weights, the `network.all_params` contains all parameters of the network. In this case, `network.all_params = [W1, b1, W2, b2, Wout, bout]` according to param 0, 1 ... 5 shown by `network.print_params()`. Then max-norm regularization on W1 and W2 can be performed as follow.

```
y = network.outputs
# Alternatively, you can use tl.cost.cross_entropy(y, y_) instead.
cross_entropy = tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(y, y_))
cost = cross_entropy
cost = cost + tl.cost.maxnorm_regularizer(1.0)(network.all_params[0]) +
          tl.cost.maxnorm_regularizer(1.0)(network.all_params[2])
```

In addition, all TensorFlow's regularizers like `tf.contrib.layers.l2_regularizer` can be used with TensorLayer.

Regularization of Activation outputs

Instance method `network.print_layers()` prints all outputs of different layers in order. To achieve regularization on activation output, you can use `network.all_layers` which contains all outputs of different layers. If you want to apply L1 penalty on the activations of first hidden layer, just simply add `tf.contrib.layers.l2_regularizer(lambda_l1)(network.all_layers[1])` to the cost function.

```
network.print_layers()
```

```
layer 0: Tensor("dropout/mul_1:0", shape=(?, 784), dtype=float32)
layer 1: Tensor("Relu:0", shape=(?, 800), dtype=float32)
layer 2: Tensor("dropout_1/mul_1:0", shape=(?, 800), dtype=float32)
layer 3: Tensor("Relu_1:0", shape=(?, 800), dtype=float32)
layer 4: Tensor("dropout_2/mul_1:0", shape=(?, 800), dtype=float32)
layer 5: Tensor("add_2:0", shape=(?, 10), dtype=float32)
```

cross_entropy(output, target[, name])	It is a softmax cross-entropy operation, returns the TensorFlow expression of cross-entropy of two distributions, implement softmax internally.
sigmoid_cross_entropy(output, target[, name])	It is a sigmoid cross-entropy operation, see `tf.nn.sigmoid_cross_entropy_with_logits`.
binary_cross_entropy(output, target[, ...])	Computes binary cross entropy given *output*.
mean_squared_error(output, target[, ...])	Return the TensorFlow expression of mean-square-error (L2) of two batch of data.
normalized_mean_square_error(output, target)	Return the TensorFlow expression of normalized mean-square-error of two distributions.
absolute_difference_error(output, target[, ...])	Return the TensorFlow expression of absolute difference error (L1) of two batch of data.
dice_coe(output, target[, loss_type, axis, ...])	Soft dice (Sørensen or Jaccard) coefficient for comparing the similarity of two batch of data, usually be used for binary image segmentation i.e.
dice_hard_coe(output, target[, threshold, ...])	Non-differentiable Sørensen–Dice coefficient for comparing the similarity of two batch of data, usually be used for binary image segmentation i.e.
iou_coe(output, target[, threshold, axis, ...])	Non-differentiable Intersection over Union (IoU) for comparing the similarity of two batch of data, usually be used for evaluating binary image segmentation.
cross_entropy_seq(logits, target_seqs[, ...])	Returns the expression of cross-entropy of two sequences, implement softmax internally.
cross_entropy_seq_with_mask(logits, ...[, ...])	Returns the expression of cross-entropy of two sequences, implement softmax internally.
cosine_similarity(v1, v2)	Cosine similarity [-1, 1], wiki.
li_regularizer(scale[, scope])	li regularization removes the neurons of previous layer, *i* represents *inputs*.
lo_regularizer(scale[, scope])	lo regularization removes the neurons of current layer, *o* represents *outputs*
maxnorm_regularizer([scale, scope])	Max-norm regularization returns a function that can be used to apply max-norm regularization to weights.

Continued on next page

Table 56 – continued from previous page

maxnorm_o_regularizer(scale, scope)	Max-norm output regularization removes the neurons of current layer.
maxnorm_i_regularizer(scale[, scope])	Max-norm input regularization removes the neurons of previous layer.

2.2.2 Softmax cross entropy

tensorlayer.cost.**cross_entropy**(*output*, *target*, *name=None*)

It is a softmax cross-entropy operation, returns the TensorFlow expression of cross-entropy of two distributions, implement softmax internally. See `tf.nn.sparse_softmax_cross_entropy_with_logits`.

Parameters

output [Tensorflow variable] A distribution with shape: [batch_size, n_feature].

target [Tensorflow variable] A batch of index with shape: [batch_size,].

name [string] Name of this loss.

References

- About cross-entropy: wiki.
- The code is borrowed from: here.

Examples

```
>>> ce = tl.cost.cross_entropy(y_logits, y_target_logits, 'my_loss')
```

2.2.3 Sigmoid cross entropy

tensorlayer.cost.**sigmoid_cross_entropy**(*output*, *target*, *name=None*)

It is a sigmoid cross-entropy operation, see `tf.nn.sigmoid_cross_entropy_with_logits`.

2.2.4 Binary cross entropy

tensorlayer.cost.**binary_cross_entropy**(*output*, *target*, *epsilon=1e-08*, *name='bce_loss'*)

Computes binary cross entropy given *output*.

For brevity, let *x* = *output*, *z* = *target*. The binary cross entropy loss is

loss(x, z) = - sum_i (x[i] * log(z[i]) + (1 - x[i]) * log(1 - z[i]))

Parameters

output [tensor of type *float32* or *float64*.]

target [tensor of the same type and shape as *output*.]

epsilon [float] A small value to avoid output is zero.

name [string] An optional name to attach to this layer.

References

- DRAW

2.2.5 Mean squared error (L2)

tensorlayer.cost.**mean_squared_error**(*output*, *target*, *is_mean=False*, *name='mean_squared_error'*)

Return the TensorFlow expression of mean-square-error (L2) of two batch of data.

Parameters

output [2D, 3D or 4D tensor i.e. [batch_size, n_feature], [batch_size, w, h] or [batch_size, w, h, c].]

target [2D, 3D or 4D tensor.]

is_mean [boolean, if True, use `tf.reduce_mean` to compute the loss of one data, otherwise, use `tf.reduce_sum` (default).]

References

- Wiki Mean Squared Error

2.2.6 Normalized mean square error

tensorlayer.cost.**normalized_mean_square_error**(*output*, *target*)

Return the TensorFlow expression of normalized mean-square-error of two distributions.

Parameters

output [2D, 3D or 4D tensor i.e. [batch_size, n_feature], [batch_size, w, h] or [batch_size, w, h, c].]

target [2D, 3D or 4D tensor.]

2.2.7 Absolute difference error (L1)

tensorlayer.cost.**absolute_difference_error**(*output*, *target*, *is_mean=False*)

Return the TensorFlow expression of absolute difference error (L1) of two batch of data.

Parameters

output [2D, 3D or 4D tensor i.e. [batch_size, n_feature], [batch_size, w, h] or [batch_size, w, h, c].]

target [2D, 3D or 4D tensor.]

is_mean [boolean, if True, use `tf.reduce_mean` to compute the loss of one data, otherwise, use `tf.reduce_sum` (default).]

2.2.8 Dice coefficient

tensorlayer.cost.**dice_coe**(*output, target, loss_type='jaccard', axis=[1, 2, 3], smooth=1e-05*)

Soft dice (Sørensen or Jaccard) coefficient for comparing the similarity of two batch of data, usually be used for binary image segmentation i.e. labels are binary. The coefficient between 0 to 1, 1 means totally match.

Parameters

output [tensor] A distribution with shape: [batch_size, ...], (any dimensions).

target [tensor] A distribution with shape: [batch_size, ...], (any dimensions).

loss_type [string] `jaccard` or `sorensen`, default is `jaccard`.

axis [list of integer] All dimensions are reduced, default `[1,2,3]`.

smooth [float] This small value will be added to the numerator and denominator. If both output and target are empty, it makes sure dice is 1. If either output or target are empty (all pixels are background), dice = `smooth/(small_value + smooth)`, then if smooth is very small, dice close to 0 (even the image values lower than the threshold), so in this case, higher smooth can have a higher dice.

References

- Wiki-Dice

Examples

```
>>> outputs = tl.act.pixel_wise_softmax(network.outputs)
>>> dice_loss = 1 - tl.cost.dice_coe(outputs, y_)
```

2.2.9 Hard Dice coefficient

tensorlayer.cost.**dice_hard_coe**(*output, target, threshold=0.5, axis=[1, 2, 3], smooth=1e-05*)

Non-differentiable Sørensen–Dice coefficient for comparing the similarity of two batch of data, usually be used for binary image segmentation i.e. labels are binary. The coefficient between 0 to 1, 1 if totally match.

Parameters

output [tensor] A distribution with shape: [batch_size, ...], (any dimensions).

target [tensor] A distribution with shape: [batch_size, ...], (any dimensions).

threshold [float] The threshold value to be true.

axis [list of integer] All dimensions are reduced, default `[1,2,3]`.

smooth [float] This small value will be added to the numerator and denominator, see `dice_coe`.

References

- Wiki-Dice

2.2.10 IOU coefficient

tensorlayer.cost.**iou_coe**(*output, target, threshold=0.5, axis=[1, 2, 3], smooth=1e-05*)

Non-differentiable Intersection over Union (IoU) for comparing the similarity of two batch of data, usually be used for evaluating binary image segmentation. The coefficient between 0 to 1, 1 means totally match.

Parameters

output [tensor] A distribution with shape: [batch_size, ...], (any dimensions).

target [tensor] A distribution with shape: [batch_size, ...], (any dimensions).

threshold [float] The threshold value to be true.

axis [list of integer] All dimensions are reduced, default `[1,2,3]`.

smooth [float] This small value will be added to the numerator and denominator, see `dice_coe`.

Notes

- IoU cannot be used as training loss, people usually use dice coefficient for training, IoU and hard-dice for evaluating.

2.2.11 Cross entropy for sequence

tensorlayer.cost.**cross_entropy_seq**(*logits*, *target_seqs*, *batch_size=None*)

Returns the expression of cross-entropy of two sequences, implement softmax internally. Normally be used for Fixed Length RNN outputs.

Parameters

logits [Tensorflow variable] 2D tensor, `network.outputs`, [batch_size*n_steps (n_examples), number of output units]

target_seqs [Tensorflow variable] target : 2D tensor [batch_size, n_steps], if the number of step is dynamic, please use `cross_entropy_seq_with_mask` instead.

batch_size [None or int.] If not None, the return cost will be divided by batch_size.

Examples

```
>>> see PTB tutorial for more details
>>> input_data = tf.placeholder(tf.int32, [batch_size, num_steps])
>>> targets = tf.placeholder(tf.int32, [batch_size, num_steps])
>>> cost = tl.cost.cross_entropy_seq(network.outputs, targets)
```

2.2.12 Cross entropy with mask for sequence

tensorlayer.cost.**cross_entropy_seq_with_mask**(*logits*, *target_seqs*, *input_mask*, *return_details=False*, *name=None*)

Returns the expression of cross-entropy of two sequences, implement softmax internally. Normally be used for Dynamic RNN outputs.

Parameters

logits [network identity outputs] 2D tensor, `network.outputs`, [batch_size, number of output units].

target_seqs [int of tensor, like word ID.] [batch_size, ?]

input_mask [the mask to compute loss] The same size with target_seqs, normally 0 and 1.

return_details [boolean]

- If False (default), only returns the loss.
- If True, returns the loss, losses, weights and targets (reshape to one vetcor).

Examples

- see Image Captioning Example.

2.2.13 Cosine similarity

`tensorlayer.cost.`**`cosine_similarity`**(*v1*, *v2*)

Cosine similarity [-1, 1], wiki.

Parameters

v1, v2 [tensor of [batch_size, n_feature], with the same number of features.]

Returns

a tensor of [batch_size,]

2.2.14 Regularization functions

For `tf.nn.l2_loss`, `tf.contrib.layers.l1_regularizer`, `tf.contrib.layers.l2_regularizer` and `tf.contrib.layers.sum_regularizer`, see TensorFlow API.

Maxnorm

`tensorlayer.cost.`**`maxnorm_regularizer`**(*scale=1.0*, *scope=None*)

Max-norm regularization returns a function that can be used to apply max-norm regularization to weights. About max-norm: wiki.

The implementation follows TensorFlow contrib.

Parameters

scale [float] A scalar multiplier *Tensor*. 0.0 disables the regularizer.

scope: An optional scope name.

Returns

A function with signature 'mn(weights, name=None)' that apply Lo regularization.

Raises

ValueError [If scale is outside of the range [0.0, 1.0] or if scale is not a float.]

Special

tensorlayer.cost.**li_regularizer**(*scale*, *scope=None*)
li regularization removes the neurons of previous layer, *i* represents *inputs*.

Returns a function that can be used to apply group li regularization to weights.

The implementation follows TensorFlow contrib.

Parameters

scale [float] A scalar multiplier *Tensor*. 0.0 disables the regularizer.

scope: An optional scope name for TF12+.

Returns

A function with signature 'li(weights, name=None)' that apply Li regularization.

Raises

ValueError [if scale is outside of the range [0.0, 1.0] or if scale is not a float.]

tensorlayer.cost.**lo_regularizer**(*scale*, *scope=None*)
lo regularization removes the neurons of current layer, *o* represents *outputs*

Returns a function that can be used to apply group lo regularization to weights.

The implementation follows TensorFlow contrib.

Parameters

scale [float] A scalar multiplier *Tensor*. 0.0 disables the regularizer.

scope: An optional scope name for TF12+.

Returns

A function with signature 'lo(weights, name=None)' that apply Lo regularization.

Raises

ValueError [If scale is outside of the range [0.0, 1.0] or if scale is not a float.]

tensorlayer.cost.**maxnorm_o_regularizer**(*scale*, *scope*)
Max-norm output regularization removes the neurons of current layer.

Returns a function that can be used to apply max-norm regularization to each column of weight matrix.

The implementation follows TensorFlow contrib.

Parameters

scale [float] A scalar multiplier *Tensor*. 0.0 disables the regularizer.

scope: An optional scope name.

Returns

A function with signature 'mn_o(weights, name=None)' that apply Lo regularization.

Raises

ValueError [If scale is outside of the range [0.0, 1.0] or if scale is not a float.]

tensorlayer.cost.**maxnorm_i_regularizer**(*scale*, *scope=None*)
Max-norm input regularization removes the neurons of previous layer.

Returns a function that can be used to apply max-norm regularization to each row of weight matrix.

The implementation follows TensorFlow contrib.

Parameters

scale [float] A scalar multiplier *Tensor*. 0.0 disables the regularizer.

scope: An optional scope name.

Returns

A function with signature 'mn_i(weights, name=None)' that apply Lo regularization.

Raises

ValueError [If scale is outside of the range [0.0, 1.0] or if scale is not a float.]

2.3 API - Preprocessing

We provide abundant data augmentation and processing functions by using Numpy, Scipy, Threading and Queue. However, we recommend you to use TensorFlow operation function like `tf.image.central_crop`, more TensorFlow data augmentation method can be found here and `tutorial_cifar10_tfrecord.py`. Some of the code in this package are borrowed from Keras.

threading_data([data, fn, thread_count])	Return a batch of result by given data.
rotation(x[, rg, is_random, row_index, ...])	Rotate an image randomly or non-randomly.
rotation_multi(x[, rg, is_random, ...])	Rotate multiple images with the same arguments, randomly or non-randomly.
crop(x, wrg, hrg[, is_random, row_index, ...])	Randomly or centrally crop an image.
crop_multi(x, wrg, hrg[, is_random, ...])	Randomly or centrally crop multiple images.
flip_axis(x[, axis, is_random])	Flip the axis of an image, such as flip left and right, up and down, randomly or non-randomly,
flip_axis_multi(x, axis[, is_random])	Flip the axises of multiple images together, such as flip left and right, up and down, randomly or non-randomly,
shift(x[, wrg, hrg, is_random, row_index, ...])	Shift an image randomly or non-randomly.
shift_multi(x[, wrg, hrg, is_random, ...])	Shift images with the same arguments, randomly or non-randomly.
shear(x[, intensity, is_random, row_index, ...])	Shear an image randomly or non-randomly.
shear_multi(x[, intensity, is_random, ...])	Shear images with the same arguments, randomly or non-randomly.
shear2(x[, shear, is_random, row_index, ...])	Shear an image randomly or non-randomly.
shear_multi2(x[, shear, is_random, ...])	Shear images with the same arguments, randomly or non-randomly.
swirl(x[, center, strength, radius, ...])	Swirl an image randomly or non-randomly, see scikit-image swirl API and example.
swirl_multi(x[, center, strength, radius, ...])	Swirl multiple images with the same arguments, randomly or non-randomly.
elastic_transform(x, alpha, sigma[, mode, ...])	Elastic deformation of images as described in [Simard2003] .
elastic_transform_multi(x, alpha, sigma[, ...])	Elastic deformation of images as described in [Simard2003].
zoom(x[, zoom_range, is_random, row_index, ...])	Zoom in and out of a single image, randomly or non-randomly.

Continued on next page

Table 57 – continued from previous page

zoom_multi(x[, zoom_range, is_random, ...])	Zoom in and out of images with the same arguments, randomly or non-randomly.
brightness(x[, gamma, gain, is_random])	Change the brightness of a single image, randomly or non-randomly.
brightness_multi(x[, gamma, gain, is_random])	Change the brightness of multiply images, randomly or non-randomly.
illumination(x[, gamma, contrast, ...])	Perform illumination augmentation for a single image, randomly or non-randomly.
rgb_to_hsv(rgb)	Input RGB image [0~255] return HSV image [0~1].
hsv_to_rgb(hsv)	Input HSV image [0~1] return RGB image [0~255].
adjust_hue(im[, hout, is_offset, is_clip, ...])	Adjust hue of an RGB image.
imresize(x[, size, interp, mode])	Resize an image by given output size and method.
pixel_value_scale(im[, val, clip, is_random])	Scales each value in the pixels of the image.
samplewise_norm(x[, rescale, ...])	Normalize an image by rescale, samplewise centering and samplewise centering in order.
featurewise_norm(x[, mean, std, epsilon])	Normalize every pixels by the same given mean and std, which are usually compute from all examples.
channel_shift(x, intensity[, is_random, ...])	Shift the channels of an image, randomly or non-randomly, see numpy.rollaxis.
channel_shift_multi(x, intensity[, ...])	Shift the channels of images with the same arguments, randomly or non-randomly, see numpy.rollaxis .
drop(x[, keep])	Randomly set some pixels to zero by a given keeping probability.
transform_matrix_offset_center(matrix, x, y)	Return transform matrix offset center.
apply_transform(x, transform_matrix[, ...])	Return transformed images by given transform_matrix from transform_matrix_offset_center.
projective_transform_by_points(x, src, dst)	Projective transform by given coordinates, usually 4 coordinates.
array_to_img(x[, dim_ordering, scale])	Converts a numpy array to PIL image object (uint8 format).
find_contours(x[, level, fully_connected, ...])	Find iso-valued contours in a 2D array for a given level value, returns list of (n, 2)-ndarrays see skimage.measure.find_contours .
pt2map([list_points, size, val])	Inputs a list of points, return a 2D image.
binary_dilation(x[, radius])	Return fast binary morphological dilation of an image.
dilation(x[, radius])	Return greyscale morphological dilation of an image, see skimage.morphology.dilation.
binary_erosion(x[, radius])	Return binary morphological erosion of an image, see skimage.morphology.binary_erosion.
erosion(x[, radius])	Return greyscale morphological erosion of an image, see skimage.morphology.erosion.
obj_box_coord_rescale([coord, shape])	Scale down one coordinates from pixel unit to the ratio of image size i.e.
obj_box_coords_rescale([coords, shape])	Scale down a list of coordinates from pixel unit to the ratio of image size i.e.
obj_box_coord_scale_to_pixelunit(coord[, shape])	Convert one coordinate [x, y, w (or x2), h (or y2)] in ratio format to image coordinate format.
obj_box_coord_centroid_to_upleft_butright([coord])	Convert one coordinate [x_center, y_center, w, h] to [x1, y1, x2, y2] in up-left and botton-right format.

Continued on next page

Table 57 – continued from previous page

obj_box_coord_upleft_butright_to_centroid(coord)	Convert one coordinate [x1, y1, x2, y2] to [x_center, y_center, w, h].
obj_box_coord_centroid_to_upleft(coord)	Convert one coordinate [x_center, y_center, w, h] to [x, y, w, h].
obj_box_coord_upleft_to_centroid(coord)	Convert one coordinate [x, y, w, h] to [x_center, y_center, w, h].
parse_darknet_ann_str_to_list(annotation)	Input string format of class, x, y, w, h, return list of list format.
parse_darknet_ann_list_to_cls_box(annotation)	Input list of [[class, x, y, w, h], ...], return two list of [class ...] and [[x, y, w, h], ...].
obj_box_left_right_flip(im[, coords, ...])	Left-right flip the image and coordinates for object detection.
obj_box_imresize(im[, coords, size, interp, ...])	Resize an image, and compute the new bounding box coordinates.
obj_box_crop(im[, classes, coords, wrg, ...])	Randomly or centrally crop an image, and compute the new bounding box coordinates.
obj_box_shift(im[, classes, coords, wrg, ...])	Shift an image randomly or non-randomly, and compute the new bounding box coordinates.
obj_box_zoom(im[, classes, coords, ...])	Zoom in and out of a single image, randomly or non-randomly, and compute the new bounding box coordinates.
pad_sequences(sequences[, maxlen, dtype, ...])	Pads each sequence to the same length: the length of the longest sequence.
remove_pad_sequences(sequences[, pad_id])	Remove padding.
process_sequences(sequences[, end_id, ...])	Set all tokens(ids) after END token to the padding value, and then shorten (option) it to the maximum sequence length in this batch.
sequences_add_start_id(sequences[, ...])	Add special start token(id) in the beginning of each sequence.
sequences_add_end_id(sequences[, end_id])	Add special end token(id) in the end of each sequence.
sequences_add_end_id_after_pad(sequences[, ...])	Add special end token(id) in the end of each sequence.
sequences_get_mask(sequences[, pad_val])	Return mask for sequences.

2.3.1 Threading

tensorlayer.prepro.**threading_data**(*data=None*, *fn=None*, *thread_count=None*, ***kwargs*)

Return a batch of result by given data. Usually be used for data augmentation.

Parameters

data [numpy array, file names and etc, see Examples below.]

thread_count [the number of threads to use]

fn [the function for data processing.]

more args [the args for fn, see Examples below.]

References

- python queue

- run with limited queue

Examples

- Single array

```
>>> X --> [batch_size, row, col, 1] greyscale
>>> results = threading_data(X, zoom, zoom_range=[0.5, 1], is_random=True)
... results --> [batch_size, row, col, channel]
>>> tl.visualize.images2d(images=np.asarray(results), second=0.01, saveable=True,␣
↪name='after', dtype=None)
>>> tl.visualize.images2d(images=np.asarray(X), second=0.01, saveable=True, name=
↪'before', dtype=None)
```

- List of array (e.g. functions with `multi`)

```
>>> X, Y --> [batch_size, row, col, 1]  greyscale
>>> data = threading_data([_ for _ in zip(X, Y)], zoom_multi, zoom_range=[0.5, 1],
↪ is_random=True)
... data --> [batch_size, 2, row, col, 1]
>>> X_, Y_ = data.transpose((1,0,2,3,4))
... X_, Y_ --> [batch_size, row, col, 1]
>>> tl.visualize.images2d(images=np.asarray(X_), second=0.01, saveable=True, name=
↪'after', dtype=None)
>>> tl.visualize.images2d(images=np.asarray(Y_), second=0.01, saveable=True, name=
↪'before', dtype=None)
```

- Single array split across `thread_count` threads (e.g. functions with `multi`)

```
>>> X, Y --> [batch_size, row, col, 1]  greyscale
>>> data = threading_data(X, zoom_multi, 8, zoom_range=[0.5, 1], is_random=True)
... data --> [batch_size, 2, row, col, 1]
>>> X_, Y_ = data.transpose((1,0,2,3,4))
... X_, Y_ --> [batch_size, row, col, 1]
>>> tl.visualize.images2d(images=np.asarray(X_), second=0.01, saveable=True, name=
↪'after', dtype=None)
>>> tl.visualize.images2d(images=np.asarray(Y_), second=0.01, saveable=True, name=
↪'before', dtype=None)
```

- Customized function for image segmentation

```
>>> def distort_img(data):
...     x, y = data
...     x, y = flip_axis_multi([x, y], axis=0, is_random=True)
...     x, y = flip_axis_multi([x, y], axis=1, is_random=True)
...     x, y = crop_multi([x, y], 100, 100, is_random=True)
...     return x, y
>>> X, Y --> [batch_size, row, col, channel]
>>> data = threading_data([_ for _ in zip(X, Y)], distort_img)
>>> X_, Y_ = data.transpose((1,0,2,3,4))
```

2.3.2 Images

- These functions only apply on a single image, use `threading_data` to apply multiple threading see `tutorial_image_preprocess.py`.
- All functions have argument `is_random`.
- All functions end with *multi* , usually be used for image segmentation i.e. the input and output image should be matched.

Rotation

`tensorlayer.prepro.`**`rotation`**(*x*, *rg=20*, *is_random=False*, *row_index=0*, *col_index=1*, *channel_index=2*, *fill_mode='nearest'*, *cval=0.0*, *order=1*)

Rotate an image randomly or non-randomly.

Parameters

x [numpy array] An image with dimension of [row, col, channel] (default).

rg [int or float] Degree to rotate, usually 0 ~ 180.

is_random [boolean, default False] If True, randomly rotate.

row_index, col_index, channel_index [int] Index of row, col and channel, default (0, 1, 2), for theano (1, 2, 0).

fill_mode [string] Method to fill missing pixel, default 'nearest', more options 'constant', 'reflect' or 'wrap'

- scipy ndimage affine_transform

cval [scalar, optional] Value used for points outside the boundaries of the input if mode='constant'. Default is 0.0

order [int, optional] The order of interpolation. The order has to be in the range 0-5. See `apply_transform`.

- scipy ndimage affine_transform

Examples

```
>>> x --> [row, col, 1] greyscale
>>> x = rotation(x, rg=40, is_random=False)
>>> tl.visualize.frame(x[:,:,0], second=0.01, saveable=True, name='temp',cmap=
↪'gray')
```

`tensorlayer.prepro.`**`rotation_multi`**(*x*, *rg=20*, *is_random=False*, *row_index=0*, *col_index=1*, *channel_index=2*, *fill_mode='nearest'*, *cval=0.0*, *order=1*)

Rotate multiple images with the same arguments, randomly or non-randomly. Usually be used for image segmentation which x=[X, Y], X and Y should be matched.

Parameters

x [list of numpy array] List of images with dimension of [n_images, row, col, channel] (default).

others [see `rotation`.]

Examples

```
>>> x, y --> [row, col, 1]  greyscale
>>> x, y = rotation_multi([x, y], rg=90, is_random=False)
>>> tl.visualize.frame(x[:,:,0], second=0.01, saveable=True, name='x',cmap='gray')
>>> tl.visualize.frame(y[:,:,0], second=0.01, saveable=True, name='y',cmap='gray')
```

Crop

tensorlayer.prepro.**crop**(*x*, *wrg*, *hrg*, *is_random=False*, *row_index=0*, *col_index=1*, *channel_index=2*)

Randomly or centrally crop an image.

Parameters

x [numpy array] An image with dimension of [row, col, channel] (default).

wrg [int] Size of width.

hrg [int] Size of height.

is_random [boolean, default False] If True, randomly crop, else central crop.

row_index, col_index, channel_index [int] Index of row, col and channel, default (0, 1, 2), for theano (1, 2, 0).

tensorlayer.prepro.**crop_multi**(*x*, *wrg*, *hrg*, *is_random=False*, *row_index=0*, *col_index=1*, *channel_index=2*)

Randomly or centrally crop multiple images.

Parameters

x [list of numpy array] List of images with dimension of [n_images, row, col, channel] (default).

others [see `crop`.]

Flip

tensorlayer.prepro.**flip_axis**(*x*, *axis=1*, *is_random=False*)

Flip the axis of an image, such as flip left and right, up and down, randomly or non-randomly,

Parameters

x [numpy array] An image with dimension of [row, col, channel] (default).

axis [int]

- 0, flip up and down
- 1, flip left and right
- 2, flip channel

is_random [boolean, default False] If True, randomly flip.

tensorlayer.prepro.**flip_axis_multi**(*x*, *axis*, *is_random=False*)

Flip the axises of multiple images together, such as flip left and right, up and down, randomly or non-randomly,

Parameters

x [list of numpy array] List of images with dimension of [n_images, row, col, channel] (default).

others [see `flip_axis`.]

Shift

tensorlayer.prepro.**shift**(*x*, *wrg=0.1*, *hrg=0.1*, *is_random=False*, *row_index=0*, *col_index=1*, *channel_index=2*, *fill_mode='nearest'*, *cval=0.0*, *order=1*)

Shift an image randomly or non-randomly.

Parameters

x [numpy array] An image with dimension of [row, col, channel] (default).

wrg [float] Percentage of shift in axis x, usually -0.25 ~ 0.25.

hrg [float] Percentage of shift in axis y, usually -0.25 ~ 0.25.

is_random [boolean, default False] If True, randomly shift.

row_index, col_index, channel_index [int] Index of row, col and channel, default (0, 1, 2), for theano (1, 2, 0).

fill_mode [string] Method to fill missing pixel, default 'nearest', more options 'constant', 'reflect' or 'wrap'.

- scipy ndimage affine_transform

cval [scalar, optional] Value used for points outside the boundaries of the input if mode='constant'. Default is 0.0.

order [int, optional] The order of interpolation. The order has to be in the range 0-5. See `apply_transform`.

- scipy ndimage affine_transform

tensorlayer.prepro.**shift_multi**(*x*, *wrg=0.1*, *hrg=0.1*, *is_random=False*, *row_index=0*, *col_index=1*, *channel_index=2*, *fill_mode='nearest'*, *cval=0.0*, *order=1*)

Shift images with the same arguments, randomly or non-randomly. Usually be used for image segmentation which x=[X, Y], X and Y should be matched.

Parameters

x [list of numpy array] List of images with dimension of [n_images, row, col, channel] (default).

others [see `shift`.]

Shear

tensorlayer.prepro.**shear**(*x*, *intensity=0.1*, *is_random=False*, *row_index=0*, *col_index=1*, *channel_index=2*, *fill_mode='nearest'*, *cval=0.0*, *order=1*)

Shear an image randomly or non-randomly.

Parameters

x [numpy array] An image with dimension of [row, col, channel] (default).

intensity [float] Percentage of shear, usually -0.5 ~ 0.5 (is_random==True), 0 ~ 0.5 (is_random==False), you can have a quick try by shear(X, 1).

is_random [boolean, default False] If True, randomly shear.

row_index, col_index, channel_index [int] Index of row, col and channel, default (0, 1, 2), for theano (1, 2, 0).

fill_mode [string] Method to fill missing pixel, default 'nearest', more options 'constant', 'reflect' or 'wrap'.

- scipy ndimage affine_transform

cval [scalar, optional] Value used for points outside the boundaries of the input if mode='constant'. Default is 0.0.

order [int, optional] The order of interpolation. The order has to be in the range 0-5. See `apply_transform`.

- scipy ndimage affine_transform

References

- Affine transformation

`tensorlayer.prepro.`**`shear_multi`**(*x*, *intensity=0.1*, *is_random=False*, *row_index=0*, *col_index=1*, *channel_index=2*, *fill_mode='nearest'*, *cval=0.0*, *order=1*)

Shear images with the same arguments, randomly or non-randomly. Usually be used for image segmentation which x=[X, Y], X and Y should be matched.

Parameters

x [list of numpy array] List of images with dimension of [n_images, row, col, channel] (default).

others [see `tl.prepro.shear`.]

Shear V2

`tensorlayer.prepro.`**`shear2`**(*x*, *shear=(0.1, 0.1)*, *is_random=False*, *row_index=0*, *col_index=1*, *channel_index=2*, *fill_mode='nearest'*, *cval=0.0*, *order=1*)

Shear an image randomly or non-randomly.

Parameters

x [numpy array] An image with dimension of [row, col, channel] (default).

shear [tuple of two floats] Percentage of shear for height and width direction (0, 1).

is_random [boolean, default False] If True, randomly shear.

row_index, col_index, channel_index [int] Index of row, col and channel, default (0, 1, 2), for theano (1, 2, 0).

fill_mode [string] Method to fill missing pixel, default 'nearest', more options 'constant', 'reflect' or 'wrap'.

- scipy ndimage affine_transform

cval [scalar, optional] Value used for points outside the boundaries of the input if mode='constant'. Default is 0.0.

order [int, optional] The order of interpolation. The order has to be in the range 0-5. See `apply_transform`.

- scipy ndimage affine_transform

References

- Affine transformation

tensorlayer.prepro.**shear_multi2**(*x*, *shear=(0.1, 0.1)*, *is_random=False*, *row_index=0*, *col_index=1*, *channel_index=2*, *fill_mode='nearest'*, *cval=0.0*, *order=1*)

Shear images with the same arguments, randomly or non-randomly. Usually be used for image segmentation which x=[X, Y], X and Y should be matched.

Parameters

x [list of numpy array] List of images with dimension of [n_images, row, col, channel] (default).

others [see `tl.prepro.shear2`.]

Swirl

tensorlayer.prepro.**swirl**(*x*, *center=None*, *strength=1*, *radius=100*, *rotation=0*, *output_shape=None*, *order=1*, *mode='constant'*, *cval=0*, *clip=True*, *preserve_range=False*, *is_random=False*)

Swirl an image randomly or non-randomly, see scikit-image swirl API and example.

Parameters

x [numpy array] An image with dimension of [row, col, channel] (default).

center [(row, column) tuple or (2,) ndarray, optional] Center coordinate of transformation.

strength [float, optional] The amount of swirling applied.

radius [float, optional] The extent of the swirl in pixels. The effect dies out rapidly beyond radius.

rotation [float, (degree) optional] Additional rotation applied to the image, usually [0, 360], relates to center.

output_shape [tuple (rows, cols), optional] Shape of the output image generated. By default the shape of the input image is preserved.

order [int, optional] The order of the spline interpolation, default is 1. The order has to be in the range 0-5. See skimage.transform.warp for detail.

mode [{'constant', 'edge', 'symmetric', 'reflect', 'wrap'}, optional] Points outside the boundaries of the input are filled according to the given mode, with 'constant' used as the default. Modes match the behaviour of numpy.pad.

cval [float, optional] Used in conjunction with mode 'constant', the value outside the image boundaries.

clip [bool, optional] Whether to clip the output to the range of values of the input image. This is enabled by default, since higher order interpolation may produce values outside the given input range.

preserve_range [bool, optional] Whether to keep the original range of values. Otherwise, the input image is converted according to the conventions of img_as_float.

is_random [boolean, default False]

If True, random swirl.

- random center = [(0 ~ x.shape[0]), (0 ~ x.shape[1])]
- random strength = [0, strength]
- random radius = [1e-10, radius]
- random rotation = [-rotation, rotation]

Examples

```
>>> x --> [row, col, 1] greyscale
>>> x = swirl(x, strength=4, radius=100)
```

tensorlayer.prepro.**swirl_multi**(*x*, *center=None*, *strength=1*, *radius=100*, *rotation=0*, *output_shape=None*, *order=1*, *mode='constant'*, *cval=0*, *clip=True*, *preserve_range=False*, *is_random=False*)

Swirl multiple images with the same arguments, randomly or non-randomly. Usually be used for image segmentation which x=[X, Y], X and Y should be matched.

Parameters

x [list of numpy array] List of images with dimension of [n_images, row, col, channel] (default).

others [see `swirl`.]

Elastic transform

tensorlayer.prepro.**elastic_transform**(*x*, *alpha*, *sigma*, *mode='constant'*, *cval=0*, *is_random=False*)

Elastic deformation of images as described in [Simard2003] .

Parameters

x [numpy array, a greyscale image.]

alpha [scalar factor.]

sigma [scalar or sequence of scalars, the smaller the sigma, the more transformation.] Standard deviation for Gaussian kernel. The standard deviations of the Gaussian filter are given for each axis as a sequence, or as a single number, in which case it is equal for all axes.

mode [default constant, see scipy.ndimage.filters.gaussian_filter.]

cval [float, optional. Used in conjunction with mode 'constant', the value outside the image boundaries.]

is_random [boolean, default False]

References

- Github.
- Kaggle

Examples

```
>>> x = elastic_transform(x, alpha = x.shape[1] * 3, sigma = x.shape[1] * 0.07)
```

tensorlayer.prepro.**elastic_transform_multi**(*x*, *alpha*, *sigma*, *mode='constant'*, *cval=0*, *is_random=False*)

Elastic deformation of images as described in [Simard2003].

Parameters

x [list of numpy array]

others [see `elastic_transform`.]

Zoom

`tensorlayer.prepro.`**`zoom`**(*x*, *zoom_range=(0.9, 1.1)*, *is_random=False*, *row_index=0*, *col_index=1*, *channel_index=2*, *fill_mode='nearest'*, *cval=0.0*, *order=1*)

Zoom in and out of a single image, randomly or non-randomly.

Parameters

x [numpy array] An image with dimension of [row, col, channel] (default).

zoom_range [list or tuple]

- If is_random=False, (h, w) are the fixed zoom factor for row and column axies, factor small than one is zoom in.
- If is_random=True, it is (min zoom out, max zoom out) for x and y with different random zoom in/out factor.

e.g (0.5, 1) zoom in 1~2 times.

is_random [boolean, default False] If True, randomly zoom.

row_index, col_index, channel_index [int] Index of row, col and channel, default (0, 1, 2), for theano (1, 2, 0).

fill_mode [string] Method to fill missing pixel, default 'nearest', more options 'constant', 'reflect' or 'wrap'.

- scipy ndimage affine_transform

cval [scalar, optional] Value used for points outside the boundaries of the input if mode='constant'. Default is 0.0.

order [int, optional] The order of interpolation. The order has to be in the range 0-5. See `apply_transform`.

- scipy ndimage affine_transform

`tensorlayer.prepro.`**`zoom_multi`**(*x*, *zoom_range=(0.9, 1.1)*, *is_random=False*, *row_index=0*, *col_index=1*, *channel_index=2*, *fill_mode='nearest'*, *cval=0.0*, *order=1*)

Zoom in and out of images with the same arguments, randomly or non-randomly. Usually be used for image segmentation which x=[X, Y], X and Y should be matched.

Parameters

x [list of numpy array] List of images with dimension of [n_images, row, col, channel] (default).

others [see `zoom`.]

Brightness

`tensorlayer.prepro.`**`brightness`**(*x*, *gamma=1*, *gain=1*, *is_random=False*)

Change the brightness of a single image, randomly or non-randomly.

Parameters

x [numpy array] An image with dimension of [row, col, channel] (default).

gamma [float, small than 1 means brighter.] Non negative real number. Default value is 1, smaller means brighter.

- If is_random is True, gamma in a range of (1-gamma, 1+gamma).

gain [float] The constant multiplier. Default value is 1.

is_random [boolean, default False]

- If True, randomly change brightness.

References

- skimage.exposure.adjust_gamma
- chinese blog

`tensorlayer.prepro.`**`brightness_multi`**(*x*, *gamma=1*, *gain=1*, *is_random=False*)

Change the brightness of multiply images, randomly or non-randomly. Usually be used for image segmentation which x=[X, Y], X and Y should be matched.

Parameters

x [list of numpy array] List of images with dimension of [n_images, row, col, channel] (default).

others [see `brightness`.]

Brightness, contrast and saturation

`tensorlayer.prepro.`**`illumination`**(*x*, *gamma=1.0*, *contrast=1.0*, *saturation=1.0*, *is_random=False*)

Perform illumination augmentation for a single image, randomly or non-randomly.

Parameters

x [numpy array] an image with dimension of [row, col, channel] (default).

gamma [change brightness (the same with `tl.prepro.brightness`)]

- if is_random=False, one float number, small than one means brighter, greater than one means darker.
- if is_random=True, tuple of two float numbers, (min, max).

contrast [change contrast]

- if is_random=False, one float number, small than one means blur.
- if is_random=True, tuple of two float numbers, (min, max).

saturation [change saturation]

- if is_random=False, one float number, small than one means unsaturation.
- if is_random=True, tuple of two float numbers, (min, max).

is_random [whether the parameters are randomly set.]

Examples

- Random

```
>>> x = illumination(x, gamma=(0.5, 5.0), contrast=(0.3, 1.0), saturation=(0.7, 1.
↪0), is_random=True)
- Non-random
>>> x = illumination(x, 0.5, 0.6, 0.8, is_random=False)
```

RGB to HSV

tensorlayer.prepro.**rgb_to_hsv**(*rgb*)

Input RGB image [0~255] return HSV image [0~1].

Parameters

rgb [should be a numpy arrays with values between 0 and 255.]

HSV to RGB

tensorlayer.prepro.**hsv_to_rgb**(*hsv*)

Input HSV image [0~1] return RGB image [0~255].

Parameters

hsv [should be a numpy arrays with values between 0.0 and 1.0]

Adjust Hue

tensorlayer.prepro.**adjust_hue**(*im*, *hout=0.66*, *is_offset=True*, *is_clip=True*, *is_random=False*)

Adjust hue of an RGB image. This is a convenience method that converts an RGB image to float representation, converts it to HSV, add an offset to the hue channel, converts back to RGB and then back to the original data type. For TF, see tf.image.adjust_hue and tf.image.random_hue.

Parameters

im [should be a numpy arrays with values between 0 and 255.]

hout [float.]

- If is_offset is False, set all hue values to this value. 0 is red; 0.33 is green; 0.66 is blue.
- If is_offset is True, add this value as the offset to the hue channel.

is_offset [boolean, default True.]

is_clip [boolean, default True.]

- If True, set negative hue values to 0.

is_random [boolean, default False.]

References

- tf.image.random_hue.
- tf.image.adjust_hue.
- StackOverflow: Changing image hue with python PIL.

Examples

- Random, add a random value between -0.2 and 0.2 as the offset to every hue values.

```
>>> im_hue = tl.prepro.adjust_hue(image, hout=0.2, is_offset=True, is_
↪random=False)
```

- Non-random, make all hue to green.

```
>>> im_green = tl.prepro.adjust_hue(image, hout=0.66, is_offset=False, is_
↪random=False)
```

Resize

tensorlayer.prepro.**imresize**(*x, size=[100, 100], interp='bicubic', mode=None*)

Resize an image by given output size and method. Warning, this function will rescale the value to [0, 255].

Parameters

x [numpy array] An image with dimension of [row, col, channel] (default).

size [int, float or tuple (h, w)]

- int, Percentage of current size.
- float, Fraction of current size.
- tuple, Size of the output image.

interp [str, optional] Interpolation to use for re-sizing ('nearest', 'lanczos', 'bilinear', 'bicubic' or 'cubic').

mode [str, optional] The PIL image mode ('P', 'L', etc.) to convert arr before resizing.

Returns

imresize [ndarray]

The resized array of image.

References

- scipy.misc.imresize

Pixel value scale

tensorlayer.prepro.**pixel_value_scale**(*im, val=0.9, clip=[], is_random=False*)

Scales each value in the pixels of the image.

Parameters

im [numpy array for one image.]

val [float.]

- If is_random=False, multiply this value with all pixels.
- If is_random=True, multiply a value between [1-val, 1+val] with all pixels.

Examples

- Random

```
>>> im = pixel_value_scale(im, 0.1, [0, 255], is_random=True)
```

- Non-random

```
>>> im = pixel_value_scale(im, 0.9, [0, 255], is_random=False)
```

Normalization

tensorlayer.prepro.**samplewise_norm**(*x*, *rescale=None*, *samplewise_center=False*, *samplewise_std_normalization=False*, *channel_index=2*, *epsilon=1e-07*)

Normalize an image by rescale, samplewise centering and samplewise centering in order.

Parameters

x [numpy array] An image with dimension of [row, col, channel] (default).

rescale [rescaling factor.] If None or 0, no rescaling is applied, otherwise we multiply the data by the value provided (before applying any other transformation)

samplewise_center [set each sample mean to 0.]

samplewise_std_normalization [divide each input by its std.]

epsilon [small position value for dividing standard deviation.]

Notes

When samplewise_center and samplewise_std_normalization are True.

- For greyscale image, every pixels are subtracted and divided by the mean and std of whole image.
- For RGB image, every pixels are subtracted and divided by the mean and std of this pixel i.e. the mean and std of a pixel is 0 and 1.

Examples

```
>>> x = samplewise_norm(x, samplewise_center=True, samplewise_std_
↪normalization=True)
>>> print(x.shape, np.mean(x), np.std(x))
... (160, 176, 1), 0.0, 1.0
```

tensorlayer.prepro.**featurewise_norm**(*x*, *mean=None*, *std=None*, *epsilon=1e-07*)

Normalize every pixels by the same given mean and std, which are usually compute from all examples.

Parameters

x [numpy array] An image with dimension of [row, col, channel] (default).

mean [value for subtraction.]

std [value for division.]

epsilon [small position value for dividing standard deviation.]

Channel shift

tensorlayer.prepro.**channel_shift**(*x*, *intensity*, *is_random=False*, *channel_index=2*)
Shift the channels of an image, randomly or non-randomly, see numpy.rollaxis.

Parameters

x [numpy array] An image with dimension of [row, col, channel] (default).

intensity [float] Intensity of shifting.

is_random [boolean, default False] If True, randomly shift.

channel_index [int] Index of channel, default 2.

tensorlayer.prepro.**channel_shift_multi**(*x*, *intensity*, *is_random=False*, *channel_index=2*)
Shift the channels of images with the same arguments, randomly or non-randomly, see numpy.rollaxis . Usually be used for image segmentation which x=[X, Y], X and Y should be matched.

Parameters

x [list of numpy array] List of images with dimension of [n_images, row, col, channel] (default).

others [see `channel_shift`.]

Noise

tensorlayer.prepro.**drop**(*x*, *keep=0.5*)
Randomly set some pixels to zero by a given keeping probability.

Parameters

x [numpy array] An image with dimension of [row, col, channel] or [row, col].

keep [float (0, 1)] The keeping probability, the lower more values will be set to zero.

Transform matrix offset

tensorlayer.prepro.**transform_matrix_offset_center**(*matrix*, *x*, *y*)
Return transform matrix offset center.

Parameters

matrix [numpy array] Transform matrix

x, y [int] Size of image.

Examples

- See `rotation`, `shear`, `zoom`.

Apply affine transform by matrix

tensorlayer.prepro.**apply_transform**(*x*, *transform_matrix*, *channel_index=2*, *fill_mode='nearest'*, *cval=0.0*, *order=1*)

Return transformed images by given transform_matrix from `transform_matrix_offset_center`.

Parameters

x [numpy array] An image with dimension of [row, col, channel] (default).

transform_matrix [numpy array] Transform matrix (offset center), can be generated by `transform_matrix_offset_center`

channel_index [int] Index of channel, default 2.

fill_mode [string] Method to fill missing pixel, default 'nearest', more options 'constant', 'reflect' or 'wrap'

- scipy ndimage affine_transform

cval [scalar, optional] Value used for points outside the boundaries of the input if mode='constant'. Default is 0.0

order [int, optional] The order of interpolation. The order has to be in the range 0-5:

- 0 Nearest-neighbor
- 1 Bi-linear (default)
- 2 Bi-quadratic
- 3 Bi-cubic
- 4 Bi-quartic
- 5 Bi-quintic
- scipy ndimage affine_transform

Examples

- See `rotation`, `shift`, `shear`, `zoom`.

Projective transform by points

tensorlayer.prepro.**projective_transform_by_points**(*x*, *src*, *dst*, *map_args={}*, *output_shape=None*, *order=1*, *mode='constant'*, *cval=0.0*, *clip=True*, *preserve_range=False*)

Projective transform by given coordinates, usually 4 coordinates. see scikit-image.

Parameters

x [numpy array] An image with dimension of [row, col, channel] (default).

src [list or numpy] The original coordinates, usually 4 coordinates of (width, height).

dst [list or numpy] The coordinates after transformation, the number of coordinates is the same with src.

map_args [dict, optional] Keyword arguments passed to inverse_map.

output_shape [tuple (rows, cols), optional] Shape of the output image generated. By default the shape of the input image is preserved. Note that, even for multi-band images, only rows and columns need to be specified.

order [int, optional] The order of interpolation. The order has to be in the range 0-5:

- 0 Nearest-neighbor
- 1 Bi-linear (default)
- 2 Bi-quadratic
- 3 Bi-cubic
- 4 Bi-quartic
- 5 Bi-quintic

mode [{'constant', 'edge', 'symmetric', 'reflect', 'wrap'}, optional] Points outside the boundaries of the input are filled according to the given mode. Modes match the behaviour of numpy.pad.

cval [float, optional] Used in conjunction with mode 'constant', the value outside the image boundaries.

clip [bool, optional] Whether to clip the output to the range of values of the input image. This is enabled by default, since higher order interpolation may produce values outside the given input range.

preserve_range [bool, optional] Whether to keep the original range of values. Otherwise, the input image is converted according to the conventions of img_as_float.

References

- scikit-image : geometric transformations
- scikit-image : examples

Examples

```
>>> Assume X is an image from CIFAR 10, i.e. shape == (32, 32, 3)
>>> src = [[0,0],[0,32],[32,0],[32,32]]     # [w, h]
>>> dst = [[10,10],[0,32],[32,0],[32,32]]
>>> x = projective_transform_by_points(X, src, dst)
```

Numpy and PIL

tensorlayer.prepro.**array_to_img**(*x*, *dim_ordering=(0, 1, 2)*, *scale=True*)

Converts a numpy array to PIL image object (uint8 format).

Parameters

x [numpy array] A image with dimension of 3 and channels of 1 or 3.

dim_ordering [list or tuple of 3 int] Index of row, col and channel, default (0, 1, 2), for theano (1, 2, 0).

scale [boolean, default is True] If True, converts image to [0, 255] from any range of value like [-1, 2].

References

- PIL Image.fromarray

Find contours

tensorlayer.prepro.**find_contours**(*x*, *level=0.8*, *fully_connected='low'*, *positive_orientation='low'*)

Find iso-valued contours in a 2D array for a given level value, returns list of (n, 2)-ndarrays see skimage.measure.find_contours .

Parameters

x [2D ndarray of double. Input data in which to find contours.]

level [float. Value along which to find contours in the array.]

fully_connected [str, {'low', 'high'}. Indicates whether array elements below the given level value are to be considered fully-connected (and hence elements above the value will only be face connected), or vice-versa. (See notes below for details.)]

positive_orientation [either 'low' or 'high'. Indicates whether the output contours will produce positively-oriented polygons around islands of low- or high-valued elements. If 'low' then contours will wind counter-clockwise around elements below the iso-value. Alternately, this means that low-valued elements are always on the left of the contour.]

Points to Image

tensorlayer.prepro.**pt2map**(*list_points=[]*, *size=(100, 100)*, *val=1*)

Inputs a list of points, return a 2D image.

Parameters

list_points [list of [x, y].]

size [tuple of (w, h) for output size.]

val [float or int for the contour value.]

Binary dilation

tensorlayer.prepro.**binary_dilation**(*x*, *radius=3*)

Return fast binary morphological dilation of an image. see skimage.morphology.binary_dilation.

Parameters

x [2D array image.]

radius [int for the radius of mask.]

Greyscale dilation

tensorlayer.prepro.**dilation**(*x*, *radius=3*)

Return greyscale morphological dilation of an image, see skimage.morphology.dilation.

Parameters

x [2D array image.]

radius [int for the radius of mask.]

Binary erosion

tensorlayer.prepro.**binary_erosion**(*x*, *radius=3*)

Return binary morphological erosion of an image, see skimage.morphology.binary_erosion.

Parameters

x [2D array image.]

radius [int for the radius of mask.]

Greyscale erosion

tensorlayer.prepro.**erosion**(*x*, *radius=3*)

Return greyscale morphological erosion of an image, see skimage.morphology.erosion.

Parameters

x [2D array image.]

radius [int for the radius of mask.]

2.3.3 Object detection

Tutorial for Image Aug

Hi, here is an example for image augmentation on VOC dataset.

```
import tensorlayer as tl

## download VOC 2012 dataset
imgs_file_list, _, _, _, classes, _, _,\
    _, objs_info_list, _ = tl.files.load_voc_dataset(dataset="2012")

## parse annotation and convert it into list format
ann_list = []
for info in objs_info_list:
    ann = tl.prepro.parse_darknet_ann_str_to_list(info)
    c, b = tl.prepro.parse_darknet_ann_list_to_cls_box(ann)
    ann_list.append([c, b])

# read and save one image
idx = 2  # you can select your own image
image = tl.vis.read_image(imgs_file_list[idx])
tl.vis.draw_boxes_and_labels_to_image(image, ann_list[idx][0],
     ann_list[idx][1], [], classes, True, save_name='_im_original.png')

# left right flip
im_flip, coords = tl.prepro.obj_box_left_right_flip(image,
        ann_list[idx][1], is_rescale=True, is_center=True, is_random=False)
tl.vis.draw_boxes_and_labels_to_image(im_flip, ann_list[idx][0],
        coords, [], classes, True, save_name='_im_flip.png')

# resize
```

(continues on next page)

(continued from previous page)

```
im_resize, coords = tl.prepro.obj_box_imresize(image,
        coords=ann_list[idx][1], size=[300, 200], is_rescale=True)
tl.vis.draw_boxes_and_labels_to_image(im_resize, ann_list[idx][0],
        coords, [], classes, True, save_name='_im_resize.png')

# crop
im_crop, clas, coords = tl.prepro.obj_box_crop(image, ann_list[idx][0],
         ann_list[idx][1], wrg=200, hrg=200,
         is_rescale=True, is_center=True, is_random=False)
tl.vis.draw_boxes_and_labels_to_image(im_crop, clas, coords, [],
         classes, True, save_name='_im_crop.png')

# shift
im_shfit, clas, coords = tl.prepro.obj_box_shift(image, ann_list[idx][0],
        ann_list[idx][1], wrg=0.1, hrg=0.1,
        is_rescale=True, is_center=True, is_random=False)
tl.vis.draw_boxes_and_labels_to_image(im_shfit, clas, coords, [],
        classes, True, save_name='_im_shift.png')

# zoom
im_zoom, clas, coords = tl.prepro.obj_box_zoom(image, ann_list[idx][0],
        ann_list[idx][1], zoom_range=(1.3, 0.7),
        is_rescale=True, is_center=True, is_random=False)
tl.vis.draw_boxes_and_labels_to_image(im_zoom, clas, coords, [],
        classes, True, save_name='_im_zoom.png')
```

In practice, you may want to use threading method to process a batch of images as follows.

```
import tensorlayer as tl
import random

batch_size = 64
im_size = [416, 416]
n_data = len(imgs_file_list)
jitter = 0.2
def _data_pre_aug_fn(data):
    im, ann = data
    clas, coords = ann
    ## change image brightness, contrast and saturation randomly
    im = tl.prepro.illumination(im, gamma=(0.5, 1.5),
             contrast=(0.5, 1.5), saturation=(0.5, 1.5), is_random=True)
    ## flip randomly
    im, coords = tl.prepro.obj_box_left_right_flip(im, coords,
             is_rescale=True, is_center=True, is_random=True)
    ## randomly resize and crop image, it can have same effect as random zoom
    tmp0 = random.randint(1, int(im_size[0]*jitter))
    tmp1 = random.randint(1, int(im_size[1]*jitter))
    im, coords = tl.prepro.obj_box_imresize(im, coords,
            [im_size[0]+tmp0, im_size[1]+tmp1], is_rescale=True,
             interp='bicubic')
    im, clas, coords = tl.prepro.obj_box_crop(im, clas, coords,
             wrg=im_size[1], hrg=im_size[0], is_rescale=True,
             is_center=True, is_random=True)
    ## rescale value from [0, 255] to [-1, 1] (optional)
    im = im / 127.5 - 1
    return im, [clas, coords]
```

(continues on next page)

(continued from previous page)

```
# randomly read a batch of image and the corresponding annotations
idexs = tl.utils.get_random_int(min=0, max=n_data-1, number=batch_size)
b_im_path = [imgs_file_list[i] for i in idexs]
b_images = tl.prepro.threading_data(b_im_path, fn=tl.vis.read_image)
b_ann = [ann_list[i] for i in idexs]

# threading process
data = tl.prepro.threading_data([_ for _ in zip(b_images, b_ann)],
              _data_pre_aug_fn)
b_images2 = [d[0] for d in data]
b_ann = [d[1] for d in data]

# save all images
for i in range(len(b_images)):
    tl.vis.draw_boxes_and_labels_to_image(b_images[i],
              ann_list[idexs[i]][0], ann_list[idexs[i]][1], [],
              classes, True, save_name='_bbox_vis_%d_original.png' % i)
    tl.vis.draw_boxes_and_labels_to_image((b_images2[i]+1)*127.5,
              b_ann[i][0], b_ann[i][1], [], classes, True,
              save_name='_bbox_vis_%d.png' % i)
```

Coordinate pixel unit to percentage

tensorlayer.prepro.**obj_box_coord_rescale**(*coord=[], shape=[100, 200]*)

Scale down one coordinates from pixel unit to the ratio of image size i.e. in the range of [0, 1]. It is the reverse process of `obj_box_coord_scale_to_pixelunit`.

Parameters

coords [list of list for coordinates [[x, y, w, h], [x, y, w, h], ...]]

shape [list of 2 integers for [height, width] of the image.]

Examples

```
>>> coord = obj_box_coord_rescale(coord=[30, 40, 50, 50], shape=[100, 100])
... [[0.3, 0.4, 0.5, 0.5]]
```

Coordinates pixel unit to percentage

tensorlayer.prepro.**obj_box_coords_rescale**(*coords=[], shape=[100, 200]*)

Scale down a list of coordinates from pixel unit to the ratio of image size i.e. in the range of [0, 1].

Parameters

coords [list of list for coordinates [[x, y, w, h], [x, y, w, h], ...]]

shape [list of 2 integers for [height, width] of the image.]

Examples

```
>>> coords = obj_box_coords_rescale(coords=[[30, 40, 50, 50], [10, 10, 20, 20]],␣
↪shape=[100, 100])
>>> print(coords)
... [[0.3, 0.4, 0.5, 0.5], [0.1, 0.1, 0.2, 0.2]]
>>> coords = obj_box_coords_rescale(coords=[[30, 40, 50, 50]], shape=[50, 100])
>>> print(coords)
... [[0.3, 0.8, 0.5, 1.0]]
>>> coords = obj_box_coords_rescale(coords=[[30, 40, 50, 50]], shape=[100, 200])
>>> print(coords)
... [[0.15, 0.4, 0.25, 0.5]]
```

Coordinate percentage to pixel unit

tensorlayer.prepro.**obj_box_coord_scale_to_pixelunit**(*coord*, *shape=(100, 100, 3)*)

Convert one coordinate [x, y, w (or x2), h (or y2)] in ratio format to image coordinate format. It is the reverse process of `obj_box_coord_rescale`.

Parameters

coord [list of float, [x, y, w (or x2), h (or y2)] in ratio format, i.e value range [0~1].]

shape [tuple of (height, width, channel (optional))]

Examples

```
>>> x, y, x2, y2 = obj_box_coord_scale_to_pixelunit([0.2, 0.3, 0.5, 0.7],␣
↪shape=(100, 200, 3))
... (40, 30, 100, 70)
```

Coordinate [x_center, x_center, w, h] to up-left button-right

tensorlayer.prepro.**obj_box_coord_centroid_to_upleft_butright**(*coord*, *to_int=False*)

Convert one coordinate [x_center, y_center, w, h] to [x1, y1, x2, y2] in up-left and botton-right format.

Examples

```
>>> coord = obj_box_coord_centroid_to_upleft_butright([30, 40, 20, 20])
... [20, 30, 40, 50]
```

Coordinate up-left button-right to [x_center, x_center, w, h]

tensorlayer.prepro.**obj_box_coord_upleft_butright_to_centroid**(*coord*)

Convert one coordinate [x1, y1, x2, y2] to [x_center, y_center, w, h]. It is the reverse process of `obj_box_coord_centroid_to_upleft_butright`.

Coordinate [x_center, x_center, w, h] to up-left-width-high

tensorlayer.prepro.**obj_box_coord_centroid_to_upleft**(*coord*)

Convert one coordinate [x_center, y_center, w, h] to [x, y, w, h]. It is the reverse process of obj_box_coord_upleft_to_centroid.

Coordinate up-left-width-high to [x_center, x_center, w, h]

tensorlayer.prepro.**obj_box_coord_upleft_to_centroid**(*coord*)

Convert one coordinate [x, y, w, h] to [x_center, y_center, w, h]. It is the reverse process of obj_box_coord_centroid_to_upleft.

Darknet format string to list

tensorlayer.prepro.**parse_darknet_ann_str_to_list**(*annotation*)

Input string format of class, x, y, w, h, return list of list format.

Darknet format split class and coordinate

tensorlayer.prepro.**parse_darknet_ann_list_to_cls_box**(*annotation*)

Input list of [[class, x, y, w, h], ...], return two list of [class ...] and [[x, y, w, h], ...].

Image Aug - Flip

tensorlayer.prepro.**obj_box_left_right_flip**(*im*, *coords=[]*, *is_rescale=False*, *is_center=False*, *is_random=False*)

Left-right flip the image and coordinates for object detection.

Parameters

im [numpy array] An image with dimension of [row, col, channel] (default).

coords [list of list for coordinates [[x, y, w, h], [x, y, w, h], ...]]

is_rescale [boolean, default False] Set to True, if the input coordinates are rescaled to [0, 1].

is_center [boolean, default False] Set to True, if the x and y of coordinates are the centroid. (i.e. darknet format)

is_random [boolean, default False] If True, randomly flip.

Examples

```
>>> im = np.zeros([80, 100])    # as an image with shape width=100, height=80
>>> im, coords = obj_box_left_right_flip(im, coords=[[0.2, 0.4, 0.3, 0.3], [0.1,␣
↪0.5, 0.2, 0.3]], is_rescale=True, is_center=True, is_random=False)
>>> print(coords)
... [[0.8, 0.4, 0.3, 0.3], [0.9, 0.5, 0.2, 0.3]]
>>> im, coords = obj_box_left_right_flip(im, coords=[[0.2, 0.4, 0.3, 0.3]], is_
↪rescale=True, is_center=False, is_random=False)
>>> print(coords)
... [[0.5, 0.4, 0.3, 0.3]]
>>> im, coords = obj_box_left_right_flip(im, coords=[[20, 40, 30, 30]], is_
↪rescale=False, is_center=True, is_random=False)
```

(continues on next page)

(continued from previous page)

```
>>> print(coords)
... [[80, 40, 30, 30]]
>>> im, coords = obj_box_left_right_flip(im, coords=[[20, 40, 30, 30]], is_
↪rescale=False, is_center=False, is_random=False)
>>> print(coords)
... [[50, 40, 30, 30]]
```

Image Aug - Resize

tensorlayer.prepro.**obj_box_imresize**(*im, coords=[], size=[100, 100], interp='bicubic', mode=None, is_rescale=False*)

Resize an image, and compute the new bounding box coordinates.

Parameters

im [numpy array] An image with dimension of [row, col, channel] (default).

coords [list of list for coordinates [[x, y, w, h], [x, y, w, h], ...]]

size, interp, mode [see `tl.prepro.imresize` for details.]

is_rescale [boolean, default False] Set to True, if the input coordinates are rescaled to [0, 1], then return the original coordinates.

Examples

```
>>> im = np.zeros([80, 100, 3])    # as an image with shape width=100, height=80
>>> _, coords = obj_box_imresize(im, coords=[[20, 40, 30, 30], [10, 20, 20, 20]],␣
↪size=[160, 200], is_rescale=False)
>>> print(coords)
... [[40, 80, 60, 60], [20, 40, 40, 40]]
>>> _, coords = obj_box_imresize(im, coords=[[20, 40, 30, 30]], size=[40, 100],␣
↪is_rescale=False)
>>> print(coords)
... [20, 20, 30, 15]
>>> _, coords = obj_box_imresize(im, coords=[[20, 40, 30, 30]], size=[60, 150],␣
↪is_rescale=False)
>>> print(coords)
... [30, 30, 45, 22]
>>> im2, coords = obj_box_imresize(im, coords=[[0.2, 0.4, 0.3, 0.3]], size=[160,␣
↪200], is_rescale=True)
>>> print(coords, im2.shape)
... [0.2, 0.4, 0.3, 0.3] (160, 200, 3)
```

Image Aug - Crop

tensorlayer.prepro.**obj_box_crop**(*im, classes=[], coords=[], wrg=100, hrg=100, is_rescale=False, is_center=False, is_random=False, thresh_wh=0.02, thresh_wh2=12.0*)

Randomly or centrally crop an image, and compute the new bounding box coordinates. Objects outside the cropped image will be removed.

Parameters

im [numpy array] An image with dimension of [row, col, channel] (default).

classes [list of class ID (int).]

coords [list of list for coordinates [[x, y, w, h], [x, y, w, h], ...]]

wrg, hrg, is_random [see `tl.prepro.crop` for details.]

is_rescale [boolean, default False] Set to True, if the input coordinates are rescaled to [0, 1].

is_center [boolean, default False] Set to True, if the x and y of coordinates are the centroid. (i.e. darknet format)

thresh_wh [float] Threshold, remove the box if its ratio of width(height) to image size less than the threshold.

thresh_wh2 [float] Threshold, remove the box if its ratio of width to height or vice verse higher than the threshold.

Image Aug - Shift

`tensorlayer.prepro.`**`obj_box_shift`**(*im*, *classes=[]*, *coords=[]*, *wrg=0.1*, *hrg=0.1*, *row_index=0*, *col_index=1*, *channel_index=2*, *fill_mode='nearest'*, *cval=0.0*, *order=1*, *is_rescale=False*, *is_center=False*, *is_random=False*, *thresh_wh=0.02*, *thresh_wh2=12.0*)

Shift an image randomly or non-randomly, and compute the new bounding box coordinates. Objects outside the cropped image will be removed.

Parameters

im [numpy array] An image with dimension of [row, col, channel] (default).

classes [list of class ID (int).]

coords [list of list for coordinates [[x, y, w, h], [x, y, w, h], ...]]

wrg, hrg, row_index, col_index, channel_index, is_random, fill_mode, cval, order [see `tl.prepro.shift`.]

is_rescale [boolean, default False] Set to True, if the input coordinates are rescaled to [0, 1].

is_center [boolean, default False] Set to True, if the x and y of coordinates are the centroid. (i.e. darknet format)

thresh_wh [float] Threshold, remove the box if its ratio of width(height) to image size less than the threshold.

thresh_wh2 [float] Threshold, remove the box if its ratio of width to height or vice verse higher than the threshold.

Image Aug - Zoom

`tensorlayer.prepro.`**`obj_box_zoom`**(*im*, *classes=[]*, *coords=[]*, *zoom_range=(0.9, 1.1)*, *row_index=0*, *col_index=1*, *channel_index=2*, *fill_mode='nearest'*, *cval=0.0*, *order=1*, *is_rescale=False*, *is_center=False*, *is_random=False*, *thresh_wh=0.02*, *thresh_wh2=12.0*)

Zoom in and out of a single image, randomly or non-randomly, and compute the new bounding box coordinates. Objects outside the cropped image will be removed.

Parameters

im [numpy array] An image with dimension of [row, col, channel] (default).

classes [list of class ID (int).]

coords [list of list for coordinates [[x, y, w, h], [x, y, w, h], ...]]

zoom_range, row_index, col_index, channel_index, is_random, fill_mode, cval, order [see `tl.prepro.zoom`.]

is_rescale [boolean, default False] Set to True, if the input coordinates are rescaled to [0, 1].

is_center [boolean, default False] Set to True, if the x and y of coordinates are the centroid. (i.e. darknet format)

thresh_wh [float] Threshold, remove the box if its ratio of width(height) to image size less than the threshold.

thresh_wh2 [float] Threshold, remove the box if its ratio of width to height or vice verse higher than the threshold.

2.3.4 Sequence

More related functions can be found in `tensorlayer.nlp`.

Padding

`tensorlayer.prepro.`**`pad_sequences`**(*sequences*, *maxlen=None*, *dtype='int32'*, *padding='post'*, *truncating='pre'*, *value=0.0*)

Pads each sequence to the same length: the length of the longest sequence. If maxlen is provided, any sequence longer than maxlen is truncated to maxlen. Truncation happens off either the beginning (default) or the end of the sequence. Supports post-padding and pre-padding (default).

Parameters

sequences [list of lists where each element is a sequence]

maxlen [int, maximum length]

dtype [type to cast the resulting sequence.]

padding ['pre' or 'post', pad either before or after each sequence.]

truncating ['pre' or 'post', remove values from sequences larger than] maxlen either in the beginning or in the end of the sequence

value [float, value to pad the sequences to the desired value.]

Returns

x [numpy array with dimensions (number_of_sequences, maxlen)]

Examples

```
>>> sequences = [[1,1,1,1,1],[2,2,2],[3,3]]
>>> sequences = pad_sequences(sequences, maxlen=None, dtype='int32',
...                           padding='post', truncating='pre', value=0.)
... [[1 1 1 1 1]
...  [2 2 2 0 0]
...  [3 3 0 0 0]]
```

Remove Padding

tensorlayer.prepro.**remove_pad_sequences**(*sequences*, *pad_id=0*)

Remove padding.

Parameters

sequences [list of list.]

pad_id [int.]

Examples

```
>>> sequences = [[2,3,4,0,0], [5,1,2,3,4,0,0,0], [4,5,0,2,4,0,0,0]]
>>> print(remove_pad_sequences(sequences, pad_id=0))
... [[2, 3, 4], [5, 1, 2, 3, 4], [4, 5, 0, 2, 4]]
```

Process

tensorlayer.prepro.**process_sequences**(*sequences*, *end_id=0*, *pad_val=0*, *is_shorten=True*, *remain_end_id=False*)

Set all tokens(ids) after END token to the padding value, and then shorten (option) it to the maximum sequence length in this batch.

Parameters

sequences [numpy array or list of list with token IDs.] e.g. [[4,3,5,3,2,2,2,2], [5,3,9,4,9,2,2,3]]

end_id [int, the special token for END.]

pad_val [int, replace the end_id and the ids after end_id to this value.]

is_shorten [boolean, default True.] Shorten the sequences.

remain_end_id [boolean, default False.] Keep an end_id in the end.

Examples

```
>>> sentences_ids = [[4, 3, 5, 3, 2, 2, 2, 2],  <-- end_id is 2
...                  [5, 3, 9, 4, 9, 2, 2, 3]]  <-- end_id is 2
>>> sentences_ids = precess_sequences(sentences_ids, end_id=vocab.end_id, pad_
↪val=0, is_shorten=True)
... [[4, 3, 5, 3, 0], [5, 3, 9, 4, 9]]
```

Add Start ID

tensorlayer.prepro.**sequences_add_start_id**(*sequences*, *start_id=0*, *remove_last=False*)

Add special start token(id) in the beginning of each sequence.

Examples

```
>>> sentences_ids = [[4,3,5,3,2,2,2,2], [5,3,9,4,9,2,2,3]]
>>> sentences_ids = sequences_add_start_id(sentences_ids, start_id=2)
... [[2, 4, 3, 5, 3, 2, 2, 2, 2], [2, 5, 3, 9, 4, 9, 2, 2, 3]]
>>> sentences_ids = sequences_add_start_id(sentences_ids, start_id=2, remove_
↪last=True)
... [[2, 4, 3, 5, 3, 2, 2, 2], [2, 5, 3, 9, 4, 9, 2, 2]]
```

- For Seq2seq

```
>>> input = [a, b, c]
>>> target = [x, y, z]
>>> decode_seq = [start_id, a, b] <-- sequences_add_start_id(input, start_id,␣
↪True)
```

Add End ID

tensorlayer.prepro.**sequences_add_end_id**(*sequences*, *end_id=888*)

Add special end token(id) in the end of each sequence.

Parameters

sequences [list of list.]

end_id [int.]

Examples

```
>>> sequences = [[1,2,3],[4,5,6,7]]
>>> print(sequences_add_end_id(sequences, end_id=999))
... [[1, 2, 3, 999], [4, 5, 6, 999]]
```

Add End ID after pad

tensorlayer.prepro.**sequences_add_end_id_after_pad**(*sequences*, *end_id=888*, *pad_id=0*)

Add special end token(id) in the end of each sequence.

Parameters

sequences [list of list.]

end_id [int.]

pad_id [int.]

Examples

```
>>> sequences = [[1,2,0,0], [1,2,3,0], [1,2,3,4]]
>>> print(sequences_add_end_id_after_pad(sequences, end_id=99, pad_id=0))
... [[1, 2, 99, 0], [1, 2, 3, 99], [1, 2, 3, 4]]
```

Get Mask

tensorlayer.prepro.**sequences_get_mask**(*sequences*, *pad_val=0*)
Return mask for sequences.

Examples

```
>>> sentences_ids = [[4, 0, 5, 3, 0, 0],
...                  [5, 3, 9, 4, 9, 0]]
>>> mask = sequences_get_mask(sentences_ids, pad_val=0)
... [[1 1 1 1 0 0]
...  [1 1 1 1 1 0]]
```

2.4 API - Iteration

Data iteration.

minibatches([inputs, targets, batch_size, ...])	Generate a generator that input a group of example in numpy.array and their labels, return the examples and labels by the given batchsize.
seq_minibatches(inputs, targets, batch_size, ...)	Generate a generator that return a batch of sequence inputs and targets.
seq_minibatches2(inputs, targets, ...)	Generate a generator that iterates on two list of words.
ptb_iterator(raw_data, batch_size, num_steps)	Generate a generator that iterates on a list of words, see PTB tutorial.

2.4.1 Non-time series

tensorlayer.iterate.**minibatches**(*inputs=None*, *targets=None*, *batch_size=None*, *shuffle=False*)
Generate a generator that input a group of example in numpy.array and their labels, return the examples and labels by the given batchsize.

Parameters

inputs [numpy.array]

24. The input features, every row is a example.

targets [numpy.array]

25. The labels of inputs, every row is a example.

batch_size [int] The batch size.

shuffle [boolean] Indicating whether to use a shuffling queue, shuffle the dataset before return.

Notes

- If you have two inputs, e.g. X1 (1000, 100) and X2 (1000, 80), you can ‘‘np.hstack((X1, X2))

into (1000, 180) and feed into `inputs`, then you can split a batch of X1 and X2.

Examples

```
>>> X = np.asarray([['a','a'], ['b','b'], ['c','c'], ['d','d'], ['e','e'], ['f','f
↪']])
>>> y = np.asarray([0,1,2,3,4,5])
>>> for batch in tl.iterate.minibatches(inputs=X, targets=y, batch_size=2,␣
↪shuffle=False):
>>>     print(batch)
... (array([['a', 'a'],
...         ['b', 'b']],
...         dtype='<U1'), array([0, 1]))
... (array([['c', 'c'],
...         ['d', 'd']],
...         dtype='<U1'), array([2, 3]))
... (array([['e', 'e'],
...         ['f', 'f']],
...         dtype='<U1'), array([4, 5]))
```

2.4.2 Time series

Sequence iteration 1

`tensorlayer.iterate.`**`seq_minibatches`**(*inputs*, *targets*, *batch_size*, *seq_length*, *stride=1*)

Generate a generator that return a batch of sequence inputs and targets. If `batch_size = 100`, `seq_length = 5`, one return will have `500` rows (examples).

Examples

- Synced sequence input and output.

```
>>> X = np.asarray([['a','a'], ['b','b'], ['c','c'], ['d','d'], ['e','e'], ['f','f
↪']])
>>> y = np.asarray([0, 1, 2, 3, 4, 5])
>>> for batch in tl.iterate.seq_minibatches(inputs=X, targets=y, batch_size=2,␣
↪seq_length=2, stride=1):
>>>     print(batch)
... (array([['a', 'a'],
...         ['b', 'b'],
...          ['b', 'b'],
...          ['c', 'c']],
...         dtype='<U1'), array([0, 1, 1, 2]))
... (array([['c', 'c'],
...          ['d', 'd'],
...          ['d', 'd'],
...          ['e', 'e']],
...         dtype='<U1'), array([2, 3, 3, 4]))
...
...
```

- Many to One

```
>>> return_last = True
>>> num_steps = 2
>>> X = np.asarray([['a','a'], ['b','b'], ['c','c'], ['d','d'], ['e','e'], ['f','f
↪']])
>>> Y = np.asarray([0,1,2,3,4,5])
>>> for batch in tl.iterate.seq_minibatches(inputs=X, targets=Y, batch_size=2,␣
↪seq_length=num_steps, stride=1):
>>>     x, y = batch
>>>     if return_last:
>>>         tmp_y = y.reshape((-1, num_steps) + y.shape[1:])
>>>     y = tmp_y[:, -1]
>>>     print(x, y)
... [['a' 'a']
... ['b' 'b']
... ['b' 'b']
... ['c' 'c']] [1 2]
... [['c' 'c']
... ['d' 'd']
... ['d' 'd']
... ['e' 'e']] [3 4]
```

Sequence iteration 2

tensorlayer.iterate.**seq_minibatches2**(*inputs*, *targets*, *batch_size*, *num_steps*)

Generate a generator that iterates on two list of words. Yields (Returns) the source contexts and the target context by the given batch_size and num_steps (sequence_length), see PTB tutorial. In TensorFlow's tutorial, this generates the batch_size pointers into the raw PTB data, and allows minibatch iteration along these pointers.

- Hint, if the input data are images, you can modify the code as follow.

```
from
data = np.zeros([batch_size, batch_len)
to
data = np.zeros([batch_size, batch_len, inputs.shape[1], inputs.shape[2], inputs.
↪shape[3]])
```

Parameters

inputs [a list] the context in list format; note that context usually be represented by splitting by space, and then convert to unique word IDs.

targets [a list] the context in list format; note that context usually be represented by splitting by space, and then convert to unique word IDs.

batch_size [int] the batch size.

num_steps [int] the number of unrolls. i.e. sequence_length

Yields

Pairs of the batched data, each a matrix of shape [batch_size, num_steps].

Raises

ValueError [if batch_size or num_steps are too high.]

Examples

```
>>> X = [i for i in range(20)]
>>> Y = [i for i in range(20,40)]
>>> for batch in tl.iterate.seq_minibatches2(X, Y, batch_size=2, num_steps=3):
...     x, y = batch
...     print(x, y)
...
... [[  0.   1.   2.]
... [ 10.  11.  12.]]
... [[ 20.  21.  22.]
... [ 30.  31.  32.]]
...
... [[  3.   4.   5.]
... [ 13.  14.  15.]]
... [[ 23.  24.  25.]
... [ 33.  34.  35.]]
...
... [[  6.   7.   8.]
... [ 16.  17.  18.]]
... [[ 26.  27.  28.]
... [ 36.  37.  38.]]
```

PTB dataset iteration

tensorlayer.iterate.**ptb_iterator**(*raw_data*, *batch_size*, *num_steps*)

Generate a generator that iterates on a list of words, see PTB tutorial. Yields (Returns) the source contexts and the target context by the given batch_size and num_steps (sequence_length).

see `PTB tutorial`.

e.g. x = [0, 1, 2] y = [1, 2, 3] , when batch_size = 1, num_steps = 3, raw_data = [i for i in range(100)]

In TensorFlow's tutorial, this generates batch_size pointers into the raw PTB data, and allows minibatch iteration along these pointers.

Parameters

raw_data [a list] the context in list format; note that context usually be represented by splitting by space, and then convert to unique word IDs.

batch_size [int] the batch size.

num_steps [int] the number of unrolls. i.e. sequence_length

Yields

Pairs of the batched data, each a matrix of shape [batch_size, num_steps].

The second element of the tuple is the same data time-shifted to the

right by one.

Raises

ValueError [if batch_size or num_steps are too high.]

Examples

```
>>> train_data = [i for i in range(20)]
>>> for batch in tl.iterate.ptb_iterator(train_data, batch_size=2, num_steps=3):
>>>     x, y = batch
>>>     print(x, y)
... [[ 0  1  2] <---x                       1st subset/ iteration
...  [10 11 12]]
... [[ 1  2  3] <---y
...  [11 12 13]]
...
... [[ 3  4  5]  <--- 1st batch input       2nd subset/ iteration
...  [13 14 15]] <--- 2nd batch input
... [[ 4  5  6]  <--- 1st batch target
...  [14 15 16]] <--- 2nd batch target
...
... [[ 6  7  8]                            3rd subset/ iteration
...  [16 17 18]]
... [[ 7  8  9]
...  [17 18 19]]
```

2.5 API - Utility

fit(sess, network, train_op, cost, X_train, ...)	Traing a given non time-series network by the given cost function, training data, batch_size, n_epoch etc.
test(sess, network, acc, X_test, y_test, x, ...)	Test a given non time-series network by the given test data and metric.
predict(sess, network, X, x, y_op[, batch_size])	Return the predict results of given non time-series network.
evaluation([y_test, y_predict, n_classes])	Input the predicted results, targets results and the number of class, return the confusion matrix, F1-score of each class, accuracy and macro F1-score.
class_balancing_oversample([X_train, ...])	Input the features and labels, return the features and labels after oversampling.
get_random_int([min, max, number, seed])	Return a list of random integer by the given range and quantity.
dict_to_one([dp_dict])	Input a dictionary, return a dictionary that all items are set to one, use for disable dropout, dropconnect layer and so on.
list_string_to_dict(string)	Inputs `['a', 'b', 'c']`, returns `{'a': 0, 'b': 1, 'c': 2}`.
flatten_list([list_of_list])	Input a list of list, return a list that all items are in a list.

2.5.1 Training, testing and predicting

Training

tensorlayer.utils.**fit**(*sess*, *network*, *train_op*, *cost*, *X_train*, *y_train*, *x*, *y_*, *acc=None*, *batch_size=100*, *n_epoch=100*, *print_freq=5*, *X_val=None*, *y_val=None*, *eval_train=True*, *tensorboard=False*, *tensorboard_epoch_freq=5*, *tensorboard_weight_histograms=True*, *tensorboard_graph_vis=True*)

Traing a given non time-series network by the given cost function, training data, batch_size, n_epoch etc.

Parameters

sess [TensorFlow session] sess = tf.InteractiveSession()

network [a TensorLayer layer] the network will be trained

train_op [a TensorFlow optimizer] like tf.train.AdamOptimizer

X_train [numpy array] the input of training data

y_train [numpy array] the target of training data

x [placeholder] for inputs

y_ [placeholder] for targets

acc [the TensorFlow expression of accuracy (or other metric) or None] if None, would not display the metric

batch_size [int] batch size for training and evaluating

n_epoch [int] the number of training epochs

print_freq [int] display the training information every `print_freq` epochs

X_val [numpy array or None] the input of validation data

y_val [numpy array or None] the target of validation data

eval_train [boolean] if X_val and y_val are not None, it refects whether to evaluate the training data

tensorboard [boolean] if True summary data will be stored to the log/ direcory for visualization with tensorboard. See also detailed tensorboard_X settings for specific configurations of features. (default False) Also runs tl.layers.initialize_global_variables(sess) internally in fit() to setup the summary nodes, see Note:

tensorboard_epoch_freq [int] how many epochs between storing tensorboard checkpoint for visualization to log/ directory (default 5)

tensorboard_weight_histograms [boolean] if True updates tensorboard data in the logs/ directory for visulaization of the weight histograms every tensorboard_epoch_freq epoch (default True)

tensorboard_graph_vis [boolean] if True stores the graph in the tensorboard summaries saved to log/ (default True)

Notes

If tensorboard=True, the global_variables_initializer will be run inside the fit function in order to initalize the automatically generated summary nodes used for tensorboard visualization, thus tf.global_variables_initializer().run() before the fit() call will be undefined.

Examples

```
>>> see tutorial_mnist_simple.py
>>> tl.utils.fit(sess, network, train_op, cost, X_train, y_train, x, y_,
...            acc=acc, batch_size=500, n_epoch=200, print_freq=5,
...            X_val=X_val, y_val=y_val, eval_train=False)
>>> tl.utils.fit(sess, network, train_op, cost, X_train, y_train, x, y_,
...            acc=acc, batch_size=500, n_epoch=200, print_freq=5,
...            X_val=X_val, y_val=y_val, eval_train=False,
...            tensorboard=True, tensorboard_weight_histograms=True, tensorboard_
↪graph_vis=True)
```

Evaluation

tensorlayer.utils.**test**(*sess*, *network*, *acc*, *X_test*, *y_test*, *x*, *y_*, *batch_size*, *cost=None*)

Test a given non time-series network by the given test data and metric.

Parameters

sess [TensorFlow session] sess = tf.InteractiveSession()

network [a TensorLayer layer] the network will be trained

acc [the TensorFlow expression of accuracy (or other metric) or None] if None, would not display the metric

X_test [numpy array] the input of test data

y_test [numpy array] the target of test data

x [placeholder] for inputs

y_ [placeholder] for targets

batch_size [int or None] batch size for testing, when dataset is large, we should use minibatche for testing. when dataset is small, we can set it to None.

cost [the TensorFlow expression of cost or None] if None, would not display the cost

Examples

```
>>> see tutorial_mnist_simple.py
>>> tl.utils.test(sess, network, acc, X_test, y_test, x, y_, batch_size=None,␣
↪cost=cost)
```

Prediction

tensorlayer.utils.**predict**(*sess*, *network*, *X*, *x*, *y_op*, *batch_size=None*)

Return the predict results of given non time-series network.

Parameters

sess [TensorFlow session] sess = tf.InteractiveSession()

network [a TensorLayer layer] the network will be trained

X [numpy array] the input

x [placeholder] for inputs

y_op [placeholder] the argmax expression of softmax outputs

batch_size [int or None] batch size for prediction, when dataset is large, we should use mini-batche for prediction. when dataset is small, we can set it to None.

Examples

```
>>> see tutorial_mnist_simple.py
>>> y = network.outputs
>>> y_op = tf.argmax(tf.nn.softmax(y), 1)
>>> print(tl.utils.predict(sess, network, X_test, x, y_op))
```

2.5.2 Evaluation functions

tensorlayer.utils.**evaluation**(*y_test=None*, *y_predict=None*, *n_classes=None*)

Input the predicted results, targets results and the number of class, return the confusion matrix, F1-score of each class, accuracy and macro F1-score.

Parameters

y_test [numpy.array or list] target results

y_predict [numpy.array or list] predicted results

n_classes [int] number of classes

Examples

```
>>> c_mat, f1, acc, f1_macro = evaluation(y_test, y_predict, n_classes)
```

2.5.3 Class balancing functions

tensorlayer.utils.**class_balancing_oversample**(*X_train=None*, *y_train=None*, *printable=True*)

Input the features and labels, return the features and labels after oversampling.

Parameters

X_train [numpy.array] Features, each row is an example

y_train [numpy.array] Labels

Examples

- One X

```
>>> X_train, y_train = class_balancing_oversample(X_train, y_train,␣
↪printable=True)
```

- Two X

```
>>> X, y = tl.utils.class_balancing_oversample(X_train=np.hstack((X1, X2)), y_
↪train=y, printable=False)
>>> X1 = X[:, 0:5]
>>> X2 = X[:, 5:]
```

2.5.4 Random functions

tensorlayer.utils.**get_random_int**(*min=0*, *max=10*, *number=5*, *seed=None*)

Return a list of random integer by the given range and quantity.

Examples

```
>>> r = get_random_int(min=0, max=10, number=5)
... [10, 2, 3, 3, 7]
```

2.5.5 Dictionary and list

Set all items in dictionary to one

tensorlayer.utils.**dict_to_one**(*dp_dict={}*)

Input a dictionary, return a dictionary that all items are set to one, use for disable dropout, dropconnect layer and so on.

Parameters

dp_dict [dictionary] keeping probabilities

Examples

```
>>> dp_dict = dict_to_one( network.all_drop )
>>> dp_dict = dict_to_one( network.all_drop )
>>> feed_dict.update(dp_dict)
```

Convert list of string to dictionary

tensorlayer.utils.**list_string_to_dict**(*string*)

Inputs `['a', 'b', 'c']`, returns `{'a': 0, 'b': 1, 'c': 2}`.

Flatten a list

tensorlayer.utils.**flatten_list**(*list_of_list=[[], []]*)

Input a list of list, return a list that all items are in a list.

Parameters

list_of_list [a list of list]

Examples

```
>>> tl.utils.flatten_list([[1, 2, 3],[4, 5],[6]])
... [1, 2, 3, 4, 5, 6]
```

2.6 API - Natural Language Processing

Natural Language Processing and Word Representation.

generate_skip_gram_batch(data, batch_size, ...)	Generate a training batch for the Skip-Gram model.
sample([a, temperature])	Sample an index from a probability array.
sample_top([a, top_k])	Sample from `top_k` probabilities.
SimpleVocabulary(vocab, unk_id)	Simple vocabulary wrapper, see create_vocab().
Vocabulary(vocab_file[, start_word, ...])	Create Vocabulary class from a given vocabulary and its id-word, word-id convert, see create_vocab() and `tutorial_tfrecord3.py`.
process_sentence(sentence[, start_word, ...])	Converts a sentence string into a list of string words, add start_word and end_word, see `create_vocab()` and `tutorial_tfrecord3.py`.
create_vocab(sentences, word_counts_output_file)	Creates the vocabulary of word to word_id, see create_vocab() and `tutorial_tfrecord3.py`.
simple_read_words([filename])	Read context from file without any preprocessing.
read_words([filename, replace])	File to list format context.
read_analogies_file([eval_file, word2id])	Reads through an analogy question file, return its id format.
build_vocab(data)	Build vocabulary.
build_reverse_dictionary(word_to_id)	Given a dictionary for converting word to integer id.
build_words_dataset([words, ...])	Build the words dictionary and replace rare words with 'UNK' token.
save_vocab([count, name])	Save the vocabulary to a file so the model can be reloaded.
words_to_word_ids([data, word_to_id, unk_key])	Given a context (words) in list format and the vocabulary, Returns a list of IDs to represent the context.
word_ids_to_words(data, id_to_word)	Given a context (ids) in list format and the vocabulary, Returns a list of words to represent the context.
basic_tokenizer(sentence[, _WORD_SPLIT])	Very basic tokenizer: split the sentence into a list of tokens.
create_vocabulary(vocabulary_path, ..., [, ...])	Create vocabulary file (if it does not exist yet) from data file.
initialize_vocabulary(vocabulary_path)	Initialize vocabulary from file, return the word_to_id (dictionary) and id_to_word (list).
sentence_to_token_ids(sentence, vocabulary)	Convert a string to list of integers representing token-ids.
data_to_token_ids(data_path, target_path, ...)	Tokenize data file and turn into token-ids using given vocabulary file.
moses_multi_bleu(hypotheses, references[, ...])	Calculate the bleu score for hypotheses and references using the MOSES ulti-bleu.perl script.

2.6.1 Iteration function for training embedding matrix

tensorlayer.nlp.**generate_skip_gram_batch**(*data*, *batch_size*, *num_skips*, *skip_window*, *data_index=0*)

Generate a training batch for the Skip-Gram model.

Parameters

data [a list] To present context.

batch_size [an int] Batch size to return.

num_skips [an int] How many times to reuse an input to generate a label.

skip_window [an int] How many words to consider left and right.

data_index [an int] Index of the context location. without using yield, this code use data_index to instead.

Returns

batch [a list] Inputs

labels [a list] Labels

data_index [an int] Index of the context location.

References

- TensorFlow word2vec tutorial

Examples

- Setting num_skips=2, skip_window=1, use the right and left words.

  In the same way, num_skips=4, skip_window=2 means use the nearby 4 words.

```
>>> data = [1,2,3,4,5,6,7,8,9,10,11]
>>> batch, labels, data_index = tl.nlp.generate_skip_gram_batch(data=data, batch_
↪size=8, num_skips=2, skip_window=1, data_index=0)
>>> print(batch)
... [2 2 3 3 4 4 5 5]
>>> print(labels)
... [[3]
... [1]
... [4]
... [2]
... [5]
... [3]
... [4]
... [6]]
```

2.6.2 Sampling functions

Simple sampling

tensorlayer.nlp.**sample**(*a=[]*, *temperature=1.0*)

Sample an index from a probability array.

Parameters

a [a list] List of probabilities.

temperature [float or None] The higher the more uniform.

When a = [0.1, 0.2, 0.7],

temperature = 0.7, the distribution will be sharpen [0.05048273 0.13588945 0.81362782]

temperature = 1.0, the distribution will be the same [0.1 0.2 0.7]

temperature = 1.5, the distribution will be filtered [0.16008435 0.25411807 0.58579758]

If None, it will be `np.argmax(a)`

Notes

- No matter what is the temperature and input list, the sum of all probabilities will be one.

Even if input list = [1, 100, 200], the sum of all probabilities will still be one. - For large vocabulary_size, choice a higher temperature to avoid error.

Sampling from top k

tensorlayer.nlp.**sample_top**(*a=[]*, *top_k=10*)

Sample from `top_k` probabilities.

Parameters

a [a list] List of probabilities.

top_k [int] Number of candidates to be considered.

2.6.3 Vector representations of words

Simple vocabulary class

class tensorlayer.nlp.**SimpleVocabulary**(*vocab*, *unk_id*)

Simple vocabulary wrapper, see create_vocab().

Parameters

vocab [A dictionary of word to word_id.]

unk_id [Id of the special 'unknown' word.]

Methods

word_to_id(word)	Returns the integer id of a word string.

Vocabulary class

class tensorlayer.nlp.**Vocabulary**(*vocab_file*, *start_word='<S>'*, *end_word='</S>'*, *unk_word='<UNK>'*, *pad_word='<PAD>'*)

Create Vocabulary class from a given vocabulary and its id-word, word-id convert, see create_vocab() and tutorial_tfrecord3.py.

Parameters

vocab_file [File containing the vocabulary, where the words are the first] whitespace-separated token on each line (other tokens are ignored) and the word ids are the corresponding line numbers.

start_word [Special word denoting sentence start.]

end_word [Special word denoting sentence end.]

unk_word [Special word denoting unknown words.]

Attributes

vocab [a dictionary from word to id.]

reverse_vocab [a list from id to word.]

start_id [int of start id]

end_id [int of end id]

unk_id [int of unk id]

pad_id [int of padding id]

Methods

id_to_word(word_id)	Returns the word string of an integer word id.
word_to_id(word)	Returns the integer word id of a word string.

Process sentence

tensorlayer.nlp.**process_sentence**(*sentence*, *start_word='<S>'*, *end_word='</S>'*)

Converts a sentence string into a list of string words, add start_word and end_word, see create_vocab() and tutorial_tfrecord3.py.

Returns

A list of strings; the processed caption.

Notes

- You have to install the following package.
- Installing NLTK

- Installing NLTK data

Examples

```
>>> c = "how are you?"
>>> c = tl.nlp.process_sentence(c)
>>> print(c)
... ['<S>', 'how', 'are', 'you', '?', '</S>']
```

Create vocabulary

tensorlayer.nlp.**create_vocab**(*sentences*, *word_counts_output_file*, *min_word_count=1*)

Creates the vocabulary of word to word_id, see create_vocab() and `tutorial_tfrecord3.py`.

The vocabulary is saved to disk in a text file of word counts. The id of each word in the file is its corresponding 0-based line number.

Parameters

sentences [a list of lists of strings.]

word_counts_output_file [A string] The file name.

min_word_count [a int] Minimum number of occurrences for a word.

Returns

- tl.nlp.SimpleVocabulary object.

Notes

- See more `tl.nlp.build_vocab()`

Examples

```
>>> captions = ["one two , three", "four five five"]
>>> processed_capts = []
>>> for c in captions:
>>>     c = tl.nlp.process_sentence(c, start_word="<S>", end_word="</S>")
>>>     processed_capts.append(c)
>>> print(processed_capts)
...[['<S>', 'one', 'two', ',', 'three', '</S>'], ['<S>', 'four', 'five', 'five', '
↪</S>']]
```

```
>>> tl.nlp.create_vocab(processed_capts, word_counts_output_file='vocab.txt', min_
↪word_count=1)
...   [TL] Creating vocabulary.
...   Total words: 8
...   Words in vocabulary: 8
...   Wrote vocabulary file: vocab.txt
>>> vocab = tl.nlp.Vocabulary('vocab.txt', start_word="<S>", end_word="</S>", unk_
↪word="<UNK>")
... INFO:tensorflow:Initializing vocabulary from file: vocab.txt
```

(continues on next page)

(continued from previous page)

```
... [TL] Vocabulary from vocab.txt : <S> </S> <UNK>
... vocabulary with 10 words (includes start_word, end_word, unk_word)
...     start_id: 2
...     end_id: 3
...     unk_id: 9
...     pad_id: 0
```

2.6.4 Read words from file

Simple read file

tensorlayer.nlp.**simple_read_words**(*filename='nietzsche.txt'*)

Read context from file without any preprocessing.

Parameters

filename [a string] A file path (like .txt file)

Returns

The context in a string

Read file

tensorlayer.nlp.**read_words**(*filename='nietzsche.txt', replace=['\n', '<eos>']*)

File to list format context. Note that, this script can not handle punctuations. For customized read_words method, see `tutorial_generate_text.py`.

Parameters

filename [a string] A file path (like .txt file)

replace [a list] [original string, target string], to disable replace use [‘’, ‘’]

Returns

The context in a list, split by space by default, and use ‘‘<eos>‘‘ to represent ‘‘n‘‘,

e.g. ‘‘[… ‘how’, ‘useful’, ‘it’, “‘s” …]‘‘.

References

- tensorflow.models.rnn.ptb.reader

2.6.5 Read analogy question file

tensorlayer.nlp.**read_analogies_file**(*eval_file='questions-words.txt', word2id={}*)

Reads through an analogy question file, return its id format.

Parameters

eval_data [a string] The file name.

word2id [a dictionary] Mapping words to unique IDs.

Returns

analogy_questions [a [n, 4] numpy array containing the analogy question's] word ids. questions_skipped: questions skipped due to unknown words.

Examples

```
>>> eval_file should be in this format :
>>> : capital-common-countries
>>> Athens Greece Baghdad Iraq
>>> Athens Greece Bangkok Thailand
>>> Athens Greece Beijing China
>>> Athens Greece Berlin Germany
>>> Athens Greece Bern Switzerland
>>> Athens Greece Cairo Egypt
>>> Athens Greece Canberra Australia
>>> Athens Greece Hanoi Vietnam
>>> Athens Greece Havana Cuba
...
```

```
>>> words = tl.files.load_matt_mahoney_text8_dataset()
>>> data, count, dictionary, reverse_dictionary =                 tl.nlp.build_
↪words_dataset(words, vocabulary_size, True)
>>> analogy_questions = tl.nlp.read_analogies_file(                eval_file=
↪'questions-words.txt', word2id=dictionary)
>>> print(analogy_questions)
... [[ 3068  1248  7161  1581]
... [ 3068  1248 28683  5642]
... [ 3068  1248  3878   486]
... ...,
... [ 1216  4309 19982 25506]
... [ 1216  4309  3194  8650]
... [ 1216  4309   140   312]]
```

2.6.6 Build vocabulary, word dictionary and word tokenization

Build dictionary from word to id

tensorlayer.nlp.**build_vocab**(*data*)

Build vocabulary. Given the context in list format. Return the vocabulary, which is a dictionary for word to id. e.g. {'campbell': 2587, 'atlantic': 2247, 'aoun': 6746 .... }

Parameters

data [a list of string] the context in list format

Returns

word_to_id [a dictionary] mapping words to unique IDs. e.g. {'campbell': 2587, 'atlantic': 2247, 'aoun': 6746 .... }

References

- tensorflow.models.rnn.ptb.reader

Examples

```
>>> data_path = os.getcwd() + '/simple-examples/data'
>>> train_path = os.path.join(data_path, "ptb.train.txt")
>>> word_to_id = build_vocab(read_txt_words(train_path))
```

Build dictionary from id to word

tensorlayer.nlp.**build_reverse_dictionary**(*word_to_id*)

Given a dictionary for converting word to integer id. Returns a reverse dictionary for converting a id to word.

Parameters

word_to_id [dictionary] mapping words to unique ids

Returns

reverse_dictionary [a dictionary] mapping ids to words

Build dictionaries for id to word etc

tensorlayer.nlp.**build_words_dataset**(*words=[]*, *vocabulary_size=50000*, *printable=True*, *unk_key='UNK'*)

Build the words dictionary and replace rare words with 'UNK' token. The most common word has the smallest integer id.

Parameters

words [a list of string or byte] The context in list format. You may need to do preprocessing on the words, such as lower case, remove marks etc.

vocabulary_size [an int] The maximum vocabulary size, limiting the vocabulary size. Then the script replaces rare words with 'UNK' token.

printable [boolean] Whether to print the read vocabulary size of the given words.

unk_key [a string] Unknown words = unk_key

Returns

data [a list of integer] The context in a list of ids

count [a list of tuple and list] count[0] is a list : the number of rare words

count[1:] are tuples : the number of occurrence of each word

e.g. [['UNK', 418391], (b'the', 1061396), (b'of', 593677), (b'and', 416629), (b'one', 411764)]

dictionary [a dictionary] word_to_id, mapping words to unique IDs.

reverse_dictionary [a dictionary] id_to_word, mapping id to unique word.

References

- tensorflow/examples/tutorials/word2vec/word2vec_basic.py

Examples

```
>>> words = tl.files.load_matt_mahoney_text8_dataset()
>>> vocabulary_size = 50000
>>> data, count, dictionary, reverse_dictionary = tl.nlp.build_words_
↪dataset(words, vocabulary_size)
```

Save vocabulary

tensorlayer.nlp.**save_vocab**(*count=[]*, *name='vocab.txt'*)

Save the vocabulary to a file so the model can be reloaded.

Parameters

count [a list of tuple and list] count[0] is a list : the number of rare words

count[1:] are tuples : the number of occurrence of each word

e.g. [['UNK', 418391], (b'the', 1061396), (b'of', 593677), (b'and', 416629), (b'one', 411764)]

Examples

```
>>> words = tl.files.load_matt_mahoney_text8_dataset()
>>> vocabulary_size = 50000
>>> data, count, dictionary, reverse_dictionary =     ...     tl.nlp.build_words_
↪dataset(words, vocabulary_size, True)
>>> tl.nlp.save_vocab(count, name='vocab_text8.txt')
>>> vocab_text8.txt
... UNK 418391
... the 1061396
... of 593677
... and 416629
... one 411764
... in 372201
... a 325873
... to 316376
```

2.6.7 Convert words to IDs and IDs to words

These functions can be done by `Vocabulary` class.

List of Words to IDs

tensorlayer.nlp.**words_to_word_ids**(*data=[]*, *word_to_id={}*, *unk_key='UNK'*)

Given a context (words) in list format and the vocabulary, Returns a list of IDs to represent the context.

Parameters

data [a list of string or byte] the context in list format

word_to_id [a dictionary] mapping words to unique IDs.

unk_key [a string] Unknown words = unk_key

Returns

A list of IDs to represent the context.

References

- tensorflow.models.rnn.ptb.reader

Examples

```
>>> words = tl.files.load_matt_mahoney_text8_dataset()
>>> vocabulary_size = 50000
>>> data, count, dictionary, reverse_dictionary =     ...         tl.nlp.build_
↪words_dataset(words, vocabulary_size, True)
>>> context = [b'hello', b'how', b'are', b'you']
>>> ids = tl.nlp.words_to_word_ids(words, dictionary)
>>> context = tl.nlp.word_ids_to_words(ids, reverse_dictionary)
>>> print(ids)
... [6434, 311, 26, 207]
>>> print(context)
... [b'hello', b'how', b'are', b'you']
```

List of IDs to Words

`tensorlayer.nlp.`**`word_ids_to_words`**(*data*, *id_to_word*)

Given a context (ids) in list format and the vocabulary, Returns a list of words to represent the context.

Parameters

data [a list of integer] the context in list format

id_to_word [a dictionary] mapping id to unique word.

Returns

A list of string or byte to represent the context.

Examples

```
>>> see words_to_word_ids
```

2.6.8 Functions for translation

Word Tokenization

`tensorlayer.nlp.`**`basic_tokenizer`**(*sentence*, *_WORD_SPLIT=re.compile(b'([.,!?"':;)(])')*)

Very basic tokenizer: split the sentence into a list of tokens.

Parameters

sentence [tensorflow.python.platform.gfile.GFile Object]

_WORD_SPLIT [regular expression for word spliting.]

References

- Code from /tensorflow/models/rnn/translation/data_utils.py

Examples

```
>>> see create_vocabulary
>>> from tensorflow.python.platform import gfile
>>> train_path = "wmt/giga-fren.release2"
>>> with gfile.GFile(train_path + ".en", mode="rb") as f:
>>>    for line in f:
>>>       tokens = tl.nlp.basic_tokenizer(line)
>>>       print(tokens)
>>>       exit()
... [b'Changing', b'Lives', b'|', b'Changing', b'Society', b'|', b'How',
...   b'It', b'Works', b'|', b'Technology', b'Drives', b'Change', b'Home',
...   b'|', b'Concepts', b'|', b'Teachers', b'|', b'Search', b'|', b'Overview',
...   b'|', b'Credits', b'|', b'HHCC', b'Web', b'|', b'Reference', b'|',
...   b'Feedback', b'Virtual', b'Museum', b'of', b'Canada', b'Home', b'Page']
```

Create or read vocabulary

tensorlayer.nlp.**create_vocabulary**(*vocabulary_path, data_path, max_vocabulary_size, tokenizer=None, normalize_digits=True, _DIGIT_RE=re.compile(b'\\d'), _START_VOCAB=[b'_PAD', b'_GO', b'_EOS', b'_UNK']*)

Create vocabulary file (if it does not exist yet) from data file.

Data file is assumed to contain one sentence per line. Each sentence is tokenized and digits are normalized (if normalize_digits is set). Vocabulary contains the most-frequent tokens up to max_vocabulary_size. We write it to vocabulary_path in a one-token-per-line format, so that later token in the first line gets id=0, second line gets id=1, and so on.

Parameters

vocabulary_path [path where the vocabulary will be created.]

data_path [data file that will be used to create vocabulary.]

max_vocabulary_size [limit on the size of the created vocabulary.]

tokenizer [a function to use to tokenize each data sentence.] if None, basic_tokenizer will be used.

normalize_digits [Boolean] if true, all digits are replaced by 0s.

References

- Code from /tensorflow/models/rnn/translation/data_utils.py

tensorlayer.nlp.**initialize_vocabulary**(*vocabulary_path*)

Initialize vocabulary from file, return the word_to_id (dictionary) and id_to_word (list).

We assume the vocabulary is stored one-item-per-line, so a file:

dog

cat

will result in a vocabulary {“dog”: 0, “cat”: 1}, and this function will also return the reversed-vocabulary [“dog”, “cat”].

Parameters

vocabulary_path [path to the file containing the vocabulary.]

Returns

vocab [a dictionary] Word to id. A dictionary mapping string to integers.

rev_vocab [a list] Id to word. The reversed vocabulary (a list, which reverses the vocabulary mapping).

Raises

ValueError [if the provided vocabulary_path does not exist.]

Examples

```
>>> Assume 'test' contains
... dog
... cat
... bird
>>> vocab, rev_vocab = tl.nlp.initialize_vocabulary("test")
>>> print(vocab)
>>> {b'cat': 1, b'dog': 0, b'bird': 2}
>>> print(rev_vocab)
>>> [b'dog', b'cat', b'bird']
```

Convert words to IDs and IDs to words

tensorlayer.nlp.**sentence_to_token_ids**(*sentence*, *vocabulary*, *tokenizer=None*, *normalize_digits=True*, *UNK_ID=3*, *_DIGIT_RE=re.compile(b'\\d')*)

Convert a string to list of integers representing token-ids.

For example, a sentence “I have a dog” may become tokenized into [“I”, “have”, “a”, “dog”] and with vocabulary {“I”: 1, “have”: 2, “a”: 4, “dog”: 7”} this function will return [1, 2, 4, 7].

Parameters

sentence [tensorflow.python.platform.gfile.GFile Object] The sentence in bytes format to convert to token-ids.

see basic_tokenizer(), data_to_token_ids()

vocabulary [a dictionary mapping tokens to integers.]

tokenizer [a function to use to tokenize each sentence;] If None, basic_tokenizer will be used.

normalize_digits [Boolean] If true, all digits are replaced by 0s.

Returns

A list of integers, the token-ids for the sentence.

tensorlayer.nlp.**data_to_token_ids**(*data_path*, *target_path*, *vocabulary_path*, *tokenizer=None*, *normalize_digits=True*, *UNK_ID=3*, *_DIGIT_RE=re.compile(b'\\d')*)

Tokenize data file and turn into token-ids using given vocabulary file.

This function loads data line-by-line from data_path, calls the above sentence_to_token_ids, and saves the result to target_path. See comment for sentence_to_token_ids on the details of token-ids format.

Parameters

data_path [path to the data file in one-sentence-per-line format.]

target_path [path where the file with token-ids will be created.]

vocabulary_path [path to the vocabulary file.]

tokenizer [a function to use to tokenize each sentence;] if None, basic_tokenizer will be used.

normalize_digits [Boolean; if true, all digits are replaced by 0s.]

References

- Code from /tensorflow/models/rnn/translation/data_utils.py

2.6.9 Metrics

BLEU

tensorlayer.nlp.**moses_multi_bleu**(*hypotheses*, *references*, *lowercase=False*)

Calculate the bleu score for hypotheses and references using the MOSES ulti-bleu.perl script.

Parameters

hypotheses [A numpy array of strings where each string is a single example.]

references [A numpy array of strings where each string is a single example.]

lowercase [If true, pass the "-lc" flag to the multi-bleu script]

Returns

The BLEU score as a float32 value.

References

- Google/seq2seq/metric/bleu

Examples

```
>>> hypotheses = ["a bird is flying on the sky"]
>>> references = ["two birds are flying on the sky", "a bird is on the top of the␣
↪tree", "an airplane is on the sky",]
>>> score = tl.nlp.moses_multi_bleu(hypotheses, references)
```

2.7 API - Reinforcement Learning

Reinforcement Learning.

discount_episode_rewards([rewards, gamma, mode])	Take 1D float array of rewards and compute discounted rewards for an episode.
cross_entropy_reward_loss(logits, actions, ...)	Calculate the loss for Policy Gradient Network.
log_weight(probs, weights[, name])	Log weight.
choice_action_by_probs([probs, action_list])	Choice and return an an action by given the action probability distribution.

2.7.1 Reward functions

tensorlayer.rein.**discount_episode_rewards**(*rewards=[]*, *gamma=0.99*, *mode=0*)

Take 1D float array of rewards and compute discounted rewards for an episode. When encount a non-zero value, consider as the end a of an episode.

Parameters

rewards [numpy list] a list of rewards

gamma [float] discounted factor

mode [int] if mode == 0, reset the discount process when encount a non-zero reward (Ping-pong game). if mode == 1, would not reset the discount process.

Examples

```
>>> rewards = np.asarray([0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1])
>>> gamma = 0.9
>>> discount_rewards = tl.rein.discount_episode_rewards(rewards, gamma)
>>> print(discount_rewards)
... [ 0.72899997  0.81        0.89999998  1.          0.72899997  0.81
... 0.89999998  1.          0.72899997  0.81        0.89999998  1.        ]
>>> discount_rewards = tl.rein.discount_episode_rewards(rewards, gamma, mode=1)
>>> print(discount_rewards)
... [ 1.52110755  1.69011939  1.87791049  2.08656716  1.20729685  1.34144104
... 1.49048996  1.65610003  0.72899997  0.81        0.89999998  1.        ]
```

2.7.2 Cost functions

Weighted Cross Entropy

tensorlayer.rein.**cross_entropy_reward_loss**(*logits*, *actions*, *rewards*, *name=None*)

Calculate the loss for Policy Gradient Network.

Parameters

logits [tensor] The network outputs without softmax. This function implements softmax inside.

actions [tensor/ placeholder] The agent actions.

rewards [tensor/ placeholder] The rewards.

Examples

```
>>> states_batch_pl = tf.placeholder(tf.float32, shape=[None, D])
>>> network = InputLayer(states_batch_pl, name='input')
>>> network = DenseLayer(network, n_units=H, act=tf.nn.relu, name='relu1')
>>> network = DenseLayer(network, n_units=3, name='out')
>>> probs = network.outputs
>>> sampling_prob = tf.nn.softmax(probs)
>>> actions_batch_pl = tf.placeholder(tf.int32, shape=[None])
>>> discount_rewards_batch_pl = tf.placeholder(tf.float32, shape=[None])
>>> loss = tl.rein.cross_entropy_reward_loss(probs, actions_batch_pl, discount_
↪rewards_batch_pl)
>>> train_op = tf.train.RMSPropOptimizer(learning_rate, decay_rate).minimize(loss)
```

Log weight

tensorlayer.rein.**log_weight**(*probs*, *weights*, *name='log_weight'*)

Log weight.

Parameters

probs [tensor] If it is a network output, usually we should scale it to [0, 1] via softmax.

weights [tensor]

2.7.3 Sampling functions

tensorlayer.rein.**choice_action_by_probs**(*probs=[0.5, 0.5]*, *action_list=None*)

Choice and return an an action by given the action probability distribution.

Parameters

probs [a list of float.] The probability distribution of all actions.

action_list [None or a list of action in integer, string or others.] If None, returns an integer range between 0 and len(probs)-1.

Examples

```
>>> for _ in range(5):
>>>     a = choice_action_by_probs([0.2, 0.4, 0.4])
>>>     print(a)
... 0
... 1
... 1
... 2
... 1
>>> for _ in range(3):
>>>     a = choice_action_by_probs([0.5, 0.5], ['a', 'b'])
>>>     print(a)
... a
... b
... b
```

2.8 API - Files

Load benchmark dataset, save and restore model, save and load variables. TensorFlow provides `.ckpt` file format to save and restore the models, while we suggest to use standard python file format `.npz` to save models for the sake of cross-platform.

```
## save model as .ckpt
saver = tf.train.Saver()
save_path = saver.save(sess, "model.ckpt")
# restore model from .ckpt
saver = tf.train.Saver()
saver.restore(sess, "model.ckpt")

## save model as .npz
tl.files.save_npz(network.all_params , name='model.npz')
# restore model from .npz (method 1)
load_params = tl.files.load_npz(name='model.npz')
tl.files.assign_params(sess, load_params, network)
# restore model from .npz (method 2)
tl.files.load_and_assign_npz(sess=sess, name='model.npz', network=network)

## you can assign the pre-trained parameters as follow
# 1st parameter
tl.files.assign_params(sess, [load_params[0]], network)
# the first three parameters
tl.files.assign_params(sess, load_params[:3], network)
```

load_mnist_dataset([shape, path])	Automatically download MNIST dataset and return the training, validation and test set with 50000, 10000 and 10000 digit images respectively.
load_cifar10_dataset([shape, path, ...])	The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class.
load_ptb_dataset([path])	Penn TreeBank (PTB) dataset is used in many LANGUAGE MODELING papers, including "Empirical Evaluation and Combination of Advanced Language Modeling Techniques", "Recurrent Neural Network Regularization".
load_matt_mahoney_text8_dataset([path])	Download a text file from Matt Mahoney's website if not present, and make sure it's the right size.
load_imdb_dataset([path, nb_words, ...])	Load IMDB dataset
load_nietzsche_dataset([path])	Load Nietzsche dataset.
load_wmt_en_fr_dataset([path])	It will download English-to-French translation data from the WMT'15 Website (10^9-French-English corpus), and the 2013 news test from the same site as development set.
load_flickr25k_dataset([tag, path, ...])	Returns a list of images by a given tag from Flick25k dataset, it will download Flickr25k from the official website at the first time you use it.
load_flickr1M_dataset([tag, size, path, ...])	Returns a list of images by a given tag from Flickr1M dataset, it will download Flickr1M from the official website at the first time you use it.

Continued on next page

Table 64 – continued from previous page

load_cyclegan_dataset([filename, path])	Load image data from CycleGAN's database, see this link.
load_celebA_dataset([dirpath])	Automatically download celebA dataset, and return a list of image path.
load_voc_dataset([path, dataset, ...])	Pascal VOC 2007/2012 Dataset has 20 objects : aeroplane, bicycle, bird, boat, bottle, bus, car, cat, chair, cow, diningtable, dog, horse, motorbike, person, pottedplant, sheep, sofa, train, tvmonitor and additional 3 classes : head, hand, foot for person.
download_file_from_google_drive(id, destination)	Download file from Google Drive, see `load_celebA_dataset` for example.
save_npz([save_list, name, sess])	Input parameters and the file name, save parameters into .npz file.
load_npz([path, name])	Load the parameters of a Model saved by tl.files.save_npz().
assign_params(sess, params, network)	Assign the given parameters to the TensorLayer network.
load_and_assign_npz([sess, name, network])	Load model from npz and assign to a network.
save_npz_dict([save_list, name, sess])	Input parameters and the file name, save parameters as a dictionary into .npz file.
load_and_assign_npz_dict([name, sess])	Restore the parameters saved by `tl.files.save_npz_dict()`.
save_ckpt([sess, mode_name, save_dir, ...])	Save parameters into ckpt file.
load_ckpt([sess, mode_name, save_dir, ...])	Load parameters from ckpt file.
save_any_to_npy([save_dict, name])	Save variables to .npy file.
load_npy_to_any([path, name])	Load .npy file.
file_exists(filepath)	Check whether a file exists by given file path.
folder_exists(folderpath)	Check whether a folder exists by given folder path.
del_file(filepath)	Delete a file by given file path.
del_folder(folderpath)	Delete a folder by given folder path.
read_file(filepath)	Read a file and return a string.
load_file_list([path, regx, printable])	Return a file list in a folder by given a path and regular expression.
load_folder_list([path])	Return a folder list in a folder by given a folder path.
exists_or_mkdir(path[, verbose])	Check a folder by given name, if not exist, create the folder and return False, if directory exists, return True.
maybe_download_and_extract(filename, ...[, ...])	Checks if file exists in working_directory otherwise tries to dowload the file, and optionally also tries to extract the file if format is ".zip" or ".tar"
natural_keys(text)	Sort list of string with number in human order.
npz_to_W_pdf([path, regx])	Convert the first weight matrix of .npz file to .pdf by using tl.visualize.W().

2.8.1 Load dataset functions

MNIST

tensorlayer.files.**load_mnist_dataset**(*shape=(-1, 784)*, *path='data'*)

Automatically download MNIST dataset and return the training, validation and test set with 50000, 10000 and 10000 digit images respectively.

Parameters

shape [tuple] The shape of digit images, defaults is (-1,784)

path [string] The path that the data is downloaded to, defaults is `data/mnist/`.

Examples

```
>>> X_train, y_train, X_val, y_val, X_test, y_test = tl.files.load_mnist_
↪dataset(shape=(-1,784))
>>> X_train, y_train, X_val, y_val, X_test, y_test = tl.files.load_mnist_
↪dataset(shape=(-1, 28, 28, 1))
```

CIFAR-10

`tensorlayer.files.`**`load_cifar10_dataset`**(*shape=(-1, 32, 32, 3)*, *path='data'*, *plotable=False*, *second=3*)

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

Parameters

shape [tupe] The shape of digit images: e.g. (-1, 3, 32, 32) and (-1, 32, 32, 3).

plotable [True, False] Whether to plot some image examples.

second [int] If `plotable` is True, `second` is the display time.

path [string] The path that the data is downloaded to, defaults is `data/cifar10/`.

References

- CIFAR website
- Data download link
- Code references

Examples

```
>>> X_train, y_train, X_test, y_test = tl.files.load_cifar10_dataset(shape=(-1,␣
↪32, 32, 3))
```

Penn TreeBank (PTB)

`tensorlayer.files.`**`load_ptb_dataset`**(*path='data'*)

Penn TreeBank (PTB) dataset is used in many LANGUAGE MODELING papers, including "Empirical Evaluation and Combination of Advanced Language Modeling Techniques", "Recurrent Neural Network Regular-

ization". It consists of 929k training words, 73k validation words, and 82k test words. It has 10k words in its vocabulary.

Parameters

path [: string] The path that the data is downloaded to, defaults is `data/ptb/`.

Returns

train_data, valid_data, test_data, vocabulary size

References

- `tensorflow.models.rnn.ptb import reader`
- Manual download

Examples

```
>>> train_data, valid_data, test_data, vocab_size = tl.files.load_ptb_dataset()
```

Matt Mahoney's text8

`tensorlayer.files.`**`load_matt_mahoney_text8_dataset`**(*path='data'*)

Download a text file from Matt Mahoney's website if not present, and make sure it's the right size. Extract the first file enclosed in a zip file as a list of words. This dataset can be used for Word Embedding.

Parameters

path [: string] The path that the data is downloaded to, defaults is `data/mm_test8/`.

Returns

word_list [a list] a list of string (word).

e.g. [.... 'their', 'families', 'who', 'were', 'expelled', 'from', 'jerusalem', ...]

Examples

```
>>> words = tl.files.load_matt_mahoney_text8_dataset()
>>> print('Data size', len(words))
```

IMBD

`tensorlayer.files.`**`load_imdb_dataset`**(*path='data'*, *nb_words=None*, *skip_top=0*, *maxlen=None*, *test_split=0.2*, *seed=113*, *start_char=1*, *oov_char=2*, *index_from=3*)

Load IMDB dataset

Parameters

path [: string] The path that the data is downloaded to, defaults is `data/imdb/`.

References

- Modified from keras.

Examples

```
>>> X_train, y_train, X_test, y_test = tl.files.load_imdb_dataset(
...                                 nb_words=20000, test_split=0.2)
>>> print('X_train.shape', X_train.shape)
... (20000,)  [[1, 62, 74, ... 1033, 507, 27],[1, 60, 33, ... 13, 1053, 7]..]
>>> print('y_train.shape', y_train.shape)
... (20000,)  [1 0 0 ..., 1 0 1]
```

Nietzsche

tensorlayer.files.**load_nietzsche_dataset**(*path='data'*)

Load Nietzsche dataset. Returns a string.

Parameters

path [string] The path that the data is downloaded to, defaults is `data/nietzsche/`.

Examples

```
>>> see tutorial_generate_text.py
>>> words = tl.files.load_nietzsche_dataset()
>>> words = basic_clean_str(words)
>>> words = words.split()
```

English-to-French translation data from the WMT'15 Website

tensorlayer.files.**load_wmt_en_fr_dataset**(*path='data'*)

It will download English-to-French translation data from the WMT'15 Website (10^9-French-English corpus), and the 2013 news test from the same site as development set. Returns the directories of training data and test data.

Parameters

path [string] The path that the data is downloaded to, defaults is `data/wmt_en_fr/`.

Notes

Usually, it will take a long time to download this dataset.

References

- Code modified from /tensorflow/models/rnn/translation/data_utils.py

Flickr25k

tensorlayer.files.**load_flickr25k_dataset**(*tag='sky'*, *path='data'*, *n_threads=50*, *printable=False*)

Returns a list of images by a given tag from Flick25k dataset, it will download Flickr25k from the official website at the first time you use it.

Parameters

tag [string or None] If you want to get images with tag, use string like 'dog', 'red', see Flickr Search. If you want to get all images, set to `None`.

path [string] The path that the data is downloaded to, defaults is `data/flickr25k/`.

n_threads [int, number of thread to read image.]

printable [bool, print infomation when reading images, default is `False`.]

Examples

- Get images with tag of sky

```
>>> images = tl.files.load_flickr25k_dataset(tag='sky')
```

- Get all images

```
>>> images = tl.files.load_flickr25k_dataset(tag=None, n_threads=100,␣
↪printable=True)
```

Flickr1M

tensorlayer.files.**load_flickr1M_dataset**(*tag='sky'*, *size=10*, *path='data'*, *n_threads=50*, *printable=False*)

Returns a list of images by a given tag from Flickr1M dataset, it will download Flickr1M from the official website at the first time you use it.

Parameters

tag [string or None] If you want to get images with tag, use string like 'dog', 'red', see Flickr Search. If you want to get all images, set to `None`.

size [int 1 to 10.] 1 means 100k images ... 5 means 500k images, 10 means all 1 million images. Default is 10.

path [string] The path that the data is downloaded to, defaults is `data/flickr25k/`.

n_threads [int, number of thread to read image.]

printable [bool, print infomation when reading images, default is `False`.]

Examples

- Use 200k images

```
>>> images = tl.files.load_flickr1M_dataset(tag='zebra', size=2)
```

- Use 1 Million images

```
>>> images = tl.files.load_flickr1M_dataset(tag='zebra')
```

CycleGAN

tensorlayer.files.**load_cyclegan_dataset**(*filename='summer2winter_yosemite'*, *path='data'*)

Load image data from CycleGAN's database, see this link.

Parameters

filename [string] The dataset you want, see this link.

path [string] The path that the data is downloaded to, defaults is *data/cyclegan*

Examples

```
>>> im_train_A, im_train_B, im_test_A, im_test_B = load_cyclegan_dataset(filename=
↪'summer2winter_yosemite')
```

CelebA

tensorlayer.files.**load_celebA_dataset**(*dirpath='data'*)

Automatically download celebA dataset, and return a list of image path.

VOC 2007/2012

tensorlayer.files.**load_voc_dataset**(*path='data'*, *dataset='2012'*, *contain_classes_in_person=False*)

Pascal VOC 2007/2012 Dataset has 20 objects : aeroplane, bicycle, bird, boat, bottle, bus, car, cat, chair, cow, diningtable, dog, horse, motorbike, person, pottedplant, sheep, sofa, train, tvmonitor and additional 3 classes : head, hand, foot for person.

Parameters

path [string] The path that the data is downloaded to, defaults is `data/VOC`.

dataset [string, 2012, 2007, 2007test or 2012test.] The VOC dataset version, we usually train model on 2007+2012 and test it on 2007test.

contain_classes_in_person [If True, dataset will contains labels of head, hand and foot.]

Returns

imgs_file_list [list of string.] Full paths of all images.

imgs_semseg_file_list [list of string.] Full paths of all maps for semantic segmentation. Note that not all images have this map!

imgs_insseg_file_list [list of string.] Full paths of all maps for instance segmentation. Note that not all images have this map!

imgs_ann_file_list [list of string.] Full paths of all annotations for bounding box and object class, all images have this annotations.

classes [list of string.] Classes in order.

classes_in_person [list of string.] Classes in person.

classes_dict [dictionary.] Class label to integer.

n_objs_list [list of integer] Number of objects in all images in "imgs_file_list' in order.

objs_info_list [list of string.] Darknet format for the annotation of all images in `imgs_file_list` in order. `[class_id x_centre y_centre width height]` in ratio format.

objs_info_dicts [dictionary.] `{imgs_file_list : dictionary for annotation}`, the annotation of all images in `imgs_file_list`, format from TensorFlow/Models/object-detection.

References

- Pascal VOC2012 Website.
- Pascal VOC2007 Website.
- TensorFlow/Models/object-detection.

Examples

```
>>> imgs_file_list, imgs_semseg_file_list, imgs_insseg_file_list, imgs_ann_file_
↪list,
>>>     classes, classes_in_person, classes_dict,
>>>     n_objs_list, objs_info_list, objs_info_dicts = tl.files.load_voc_
↪dataset(dataset="2012", contain_classes_in_person=False)
>>> idx = 26
>>> print(classes)
... ['aeroplane', 'bicycle', 'bird', 'boat', 'bottle', 'bus', 'car', 'cat', 'chair
↪', 'cow', 'diningtable', 'dog', 'horse', 'motorbike', 'person', 'pottedplant',
↪'sheep', 'sofa', 'train', 'tvmonitor']
>>> print(classes_dict)
... {'sheep': 16, 'horse': 12, 'bicycle': 1, 'bottle': 4, 'cow': 9, 'sofa': 17,
↪'car': 6, 'dog': 11, 'cat': 7, 'person': 14, 'train': 18, 'diningtable': 10,
↪'aeroplane': 0, 'bus': 5, 'pottedplant': 15, 'tvmonitor': 19, 'chair': 8, 'bird
↪': 2, 'boat': 3, 'motorbike': 13}
>>> print(imgs_file_list[idx])
... data/VOC/VOC2012/JPEGImages/2007_000423.jpg
>>> print(n_objs_list[idx])
... 2
>>> print(imgs_ann_file_list[idx])
... data/VOC/VOC2012/Annotations/2007_000423.xml
>>> print(objs_info_list[idx])
... 14 0.173 0.461333333333 0.142 0.496
... 14 0.828 0.542666666667 0.188 0.594666666667
>>> ann = tl.prepro.parse_darknet_ann_str_to_list(objs_info_list[idx])
>>> print(ann)
... [[14, 0.173, 0.461333333333, 0.142, 0.496], [14, 0.828, 0.542666666667, 0.188,
↪ 0.594666666667]]
```

(continues on next page)

(continued from previous page)

```
>>> c, b = tl.prepro.parse_darknet_ann_list_to_cls_box(ann)
>>> print(c, b)
... [14, 14] [[0.173, 0.461333333333, 0.142, 0.496], [0.828, 0.542666666667, 0.
↪188, 0.594666666667]]
```

Google Drive

tensorlayer.files.**download_file_from_google_drive**(*id*, *destination*)

Download file from Google Drive, see `load_celebA_dataset` for example.

Parameters

id [driver ID]

destination [string, save path.]

2.8.2 Load and save network

Save network into list (npz)

tensorlayer.files.**save_npz**(*save_list=[]*, *name='model.npz'*, *sess=None*)

Input parameters and the file name, save parameters into .npz file. Use tl.utils.load_npz() to restore.

Parameters

save_list [a list] Parameters want to be saved.

name [a string or None] The name of the .npz file.

sess [None or Session]

Notes

If you got session issues, you can change the value.eval() to value.eval(session=sess)

References

- Saving dictionary using numpy

Examples

- Save model to npz

```
>>> tl.files.save_npz(network.all_params, name='model.npz', sess=sess)
- Load model from npz (Method 1)
>>> load_params = tl.files.load_npz(name='model.npz')
>>> tl.files.assign_params(sess, load_params, network)
- Load model from npz (Method 2)
>>> tl.files.load_and_assign_npz(sess=sess, name='model.npz', network=network)
```

Load network from list (npz)

tensorlayer.files.**load_npz**(*path=''*, *name='model.npz'*)
Load the parameters of a Model saved by tl.files.save_npz().

Parameters

path [a string] Folder path to .npz file.

name [a string or None] The name of the .npz file.

Returns

params [list] A list of parameters in order.

References

- Saving dictionary using numpy

Examples

- See `save_npz`

Assign a list of parameters to network

tensorlayer.files.**assign_params**(*sess*, *params*, *network*)
Assign the given parameters to the TensorLayer network.

Parameters

sess [TensorFlow Session. Automatically run when sess is not None.]

params [a list] A list of parameters in order.

network [a `Layer` class] The network to be assigned

Returns

ops [list] A list of tf ops in order that assign params. Support sess.run(ops) manually.

References

- Assign value to a TensorFlow variable

Examples

- Save model to npz

```
>>> tl.files.save_npz(network.all_params, name='model.npz', sess=sess)
- Load model from npz (Method 1)
>>> load_params = tl.files.load_npz(name='model.npz')
>>> tl.files.assign_params(sess, load_params, network)
- Load model from npz (Method 2)
>>> tl.files.load_and_assign_npz(sess=sess, name='model.npz', network=network)
```

Load and assign a list of parameters to network

tensorlayer.files.**load_and_assign_npz**(*sess=None*, *name=None*, *network=None*)

Load model from npz and assign to a network.

Parameters

sess [TensorFlow Session]

name [string] Model path.

network [a `Layer` class] The network to be assigned

Returns

Returns False if faild to model is not exist.

Examples

```
>>> tl.files.save_npz(net.all_params, name='net.npz', sess=sess)
>>> tl.files.load_and_assign_npz(sess=sess, name='net.npz', network=net)
```

Save network into dict (npz)

tensorlayer.files.**save_npz_dict**(*save_list=[]*, *name='model.npz'*, *sess=None*)

Input parameters and the file name, save parameters as a dictionary into .npz file. Use `tl.files.load_and_assign_npz_dict()` to restore.

Parameters

save_list [a list to tensor for parameters] Parameters want to be saved.

name [a string] The name of the .npz file.

sess [Session]

Load network from dict (npz)

tensorlayer.files.**load_and_assign_npz_dict**(*name='model.npz'*, *sess=None*)

Restore the parameters saved by `tl.files.save_npz_dict()`.

Parameters

name [a string] The name of the .npz file.

sess [Session]

Save network into ckpt

tensorlayer.files.**save_ckpt**(*sess=None*, *mode_name='model.ckpt'*, *save_dir='checkpoint'*, *var_list=[]*, *global_step=None*, *printable=False*)

Save parameters into ckpt file.

Parameters

sess [Session.]

mode_name [string, name of the model, default is `model.ckpt`.]

save_dir [string, path / file directory to the ckpt, default is `checkpoint`.]

var_list [list of variables, if not given, save all global variables.]

global_step [int or None, step number.]

printable [bool, if True, print all params info.]

Examples

- see `tl.files.load_ckpt()`.

Load network from ckpt

`tensorlayer.files.`**`load_ckpt`**(*sess=None*, *mode_name='model.ckpt'*, *save_dir='checkpoint'*, *var_list=[]*, *is_latest=True*, *printable=False*)

Load parameters from ckpt file.

Parameters

sess [Session.]

mode_name [string, name of the model, default is `model.ckpt`.] Note that if `is_latest` is True, this function will get the `mode_name` automatically.

save_dir [string, path / file directory to the ckpt, default is `checkpoint`.]

var_list [list of variables, if not given, save all global variables.]

is_latest [bool, if True, load the latest ckpt, if False, load the ckpt with the name of `mode_name.]

printable [bool, if True, print all params info.]

Examples

- Save all global parameters.

```
>>> tl.files.save_ckpt(sess=sess, mode_name='model.ckpt', save_dir='model',␣
↪printable=True)
- Save specific parameters.
>>> tl.files.save_ckpt(sess=sess, mode_name='model.ckpt', var_list=net.all_params,
↪ save_dir='model', printable=True)
- Load latest ckpt.
>>> tl.files.load_ckpt(sess=sess, var_list=net.all_params, save_dir='model',␣
↪printable=True)
- Load specific ckpt.
>>> tl.files.load_ckpt(sess=sess, mode_name='model.ckpt', var_list=net.all_params,
↪ save_dir='model', is_latest=False, printable=True)
```

2.8.3 Load and save variables

Save variables as .npy

`tensorlayer.files.`**`save_any_to_npy`**(*save_dict={}*, *name='file.npy'*)

Save variables to .npy file.

Examples

```
>>> tl.files.save_any_to_npy(save_dict={'data': ['a','b']}, name='test.npy')
>>> data = tl.files.load_npy_to_any(name='test.npy')
>>> print(data)
... {'data': ['a','b']}
```

Load variables from .npy

tensorlayer.files.**load_npy_to_any**(*path=''*, *name='file.npy'*)
Load .npy file.

Examples

- see save_any_to_npy()

2.8.4 Folder/File functions

Check file exists

tensorlayer.files.**file_exists**(*filepath*)
Check whether a file exists by given file path.

Check folder exists

tensorlayer.files.**folder_exists**(*folderpath*)
Check whether a folder exists by given folder path.

Delete file

tensorlayer.files.**del_file**(*filepath*)
Delete a file by given file path.

Delete folder

tensorlayer.files.**del_folder**(*folderpath*)
Delete a folder by given folder path.

Read file

tensorlayer.files.**read_file**(*filepath*)
Read a file and return a string.

Examples

```
>>> data = tl.files.read_file('data.txt')
```

Load file list from folder

tensorlayer.files.**load_file_list**(*path=None*, *regx='\\.npz'*, *printable=True*)

Return a file list in a folder by given a path and regular expression.

Parameters

path [a string or None] A folder path.

regx [a string] The regx of file name.

printable [boolean, whether to print the files infomation.]

Examples

```
>>> file_list = tl.files.load_file_list(path=None, regx='w1pre_[0-9]+\.(npz)')
```

Load folder list from folder

tensorlayer.files.**load_folder_list**(*path=''*)

Return a folder list in a folder by given a folder path.

Parameters

path [a string or None] A folder path.

Check and Create folder

tensorlayer.files.**exists_or_mkdir**(*path*, *verbose=True*)

Check a folder by given name, if not exist, create the folder and return False, if directory exists, return True.

Parameters

path [a string] A folder path.

verbose [boolean] If True, prints results, deaults is True

Returns

True if folder exist, otherwise, returns False and create the folder

Examples

```
>>> tl.files.exists_or_mkdir("checkpoints/train")
```

Download or extract

tensorlayer.files.**maybe_download_and_extract**(*filename*, *working_directory*, *url_source*, *extract=False*, *expected_bytes=None*)

Checks if file exists in working_directory otherwise tries to dowload the file, and optionally also tries to extract the file if format is ".zip" or ".tar"

Parameters

filename [string] The name of the (to be) dowloaded file.

working_directory [string] A folder path to search for the file in and dowload the file to

url [string] The URL to download the file from

extract [bool, defaults is False] If True, tries to uncompress the dowloaded file is ".tar.gz/.tar.bz2" or ".zip" file

expected_bytes [int/None] If set tries to verify that the downloaded file is of the specified size, otherwise raises an Exception, defaults is None which corresponds to no check being performed

Returns

filepath to dowloaded (uncompressed) file

Examples

```
>>> down_file = tl.files.maybe_download_and_extract(filename = 'train-images-idx3-
↪ubyte.gz',
                                                    working_directory = 'data/',
                                                    url_source = 'http://yann.
↪lecun.com/exdb/mnist/')
>>> tl.files.maybe_download_and_extract(filename = 'ADEChallengeData2016.zip',
                                        working_directory = 'data/',
                                        url_source = 'http://sceneparsing.csail.
↪mit.edu/data/',
                                        extract=True)
```

2.8.5 Sort

List of string with number in human order

tensorlayer.files.**natural_keys**(*text*)

Sort list of string with number in human order.

References

alist.sort(key=natural_keys) sorts in human order http://nedbatchelder.com/blog/200712/human_sorting.html (See Toothy's implementation in the comments)

Examples

```
>>> l = ['im1.jpg', 'im31.jpg', 'im11.jpg', 'im21.jpg', 'im03.jpg', 'im05.jpg']
>>> l.sort(key=tl.files.natural_keys)
... ['im1.jpg', 'im03.jpg', 'im05', 'im11.jpg', 'im21.jpg', 'im31.jpg']
>>> l.sort() # that is what we dont want
... ['im03.jpg', 'im05', 'im1.jpg', 'im11.jpg', 'im21.jpg', 'im31.jpg']
```

2.8.6 Visualizing npz file

tensorlayer.files.**npz_to_W_pdf**(*path=None*, *regx='w1pre_[0-9]+\\.(npz)'*)
Convert the first weight matrix of .npz file to .pdf by using tl.visualize.W().

Parameters

path [a string or None] A folder path to npz files.

regx [a string] Regx for the file name.

Examples

```
>>> Convert the first weight matrix of w1_pre...npz file to w1_pre...pdf.
>>> tl.files.npz_to_W_pdf(path='/Users/.../npz_file/', regx='w1pre_[0-9]+\.(npz)')
```

2.9 API - Visualization

TensorFlow provides TensorBoard to visualize the model, activations etc. Here we provide more functions for data visualization.

read_image(image[, path])	Read one image.
read_images(img_list[, path, n_threads, ...])	Returns all images in list by given path and name of each image file.
save_image(image[, image_path])	Save one image.
save_images(images, size[, image_path])	Save mutiple images into one single image.
draw_boxes_and_labels_to_image(image[, ...])	Draw bboxes and class labels on image.
W([W, second, saveable, shape, name, fig_idx])	Visualize every columns of the weight matrix to a group of Greyscale img.
CNN2d([CNN, second, saveable, name, fig_idx])	Display a group of RGB or Greyscale CNN masks.
frame([I, second, saveable, name, cmap, fig_idx])	Display a frame(image).
images2d([images, second, saveable, name, ...])	Display a group of RGB or Greyscale images.
tsne_embedding(embeddings, reverse_dictionary)	Visualize the embeddings by using t-SNE.

2.9.1 Save and read images

Read one image

tensorlayer.visualize.**read_image**(*image*, *path=''*)
Read one image.

Parameters

images [string, file name.]

path [string, path.]

Read multiple images

tensorlayer.visualize.**read_images**(*img_list*, *path=''*, *n_threads=10*, *printable=True*)

Returns all images in list by given path and name of each image file.

Parameters

img_list [list of string, the image file names.]

path [string, image folder path.]

n_threads [int, number of thread to read image.]

printable [bool, print infomation when reading images, default is True.]

Save one image

tensorlayer.visualize.**save_image**(*image*, *image_path=''*)

Save one image.

Parameters

images [numpy array [w, h, c]]

image_path [string.]

Save multiple images

tensorlayer.visualize.**save_images**(*images*, *size*, *image_path=''*)

Save mutiple images into one single image.

Parameters

images [numpy array [batch, w, h, c]]

size [list of two int, row and column number.] number of images should be equal or less than size[0] * size[1]

image_path [string.]

Examples

```
>>> images = np.random.rand(64, 100, 100, 3)
>>> tl.visualize.save_images(images, [8, 8], 'temp.png')
```

Save image for object detection

tensorlayer.visualize.**draw_boxes_and_labels_to_image**(*image*, *classes=[]*, *coords=[]*, *scores=[]*, *classes_list=[]*, *is_center=True*, *is_rescale=True*, *save_name=None*)

Draw bboxes and class labels on image. Return or save the image with bboxes, example in the docs of `tl.prepro`.

Parameters

image [RGB image in numpy.array, [height, width, channel].]

classes [a list of class ID (int).]

coords [a list of list for coordinates.]

- Should be [x, y, x2, y2] (up-left and botton-right format)
- If [x_center, y_center, w, h] (set is_center to True).

scores [a list of score (float). (Optional)]

classes_list [list of string, for converting ID to string on image.]

is_center [boolean, defalt is True.] If coords is [x_center, y_center, w, h], set it to True for converting [x_center, y_center, w, h] to [x, y, x2, y2] (up-left and botton-right). If coords is [x1, x2, y1, y2], set it to False.

is_rescale [boolean, defalt is True.] If True, the input coordinates are the portion of width and high, this API will scale the coordinates to pixel unit internally. If False, feed the coordinates with pixel unit format.

save_name [None or string] The name of image file (i.e. image.png), if None, not to save image.

References

- OpenCV rectangle and putText.
- scikit-image.

2.9.2 Visualize model parameters

Visualize weight matrix

tensorlayer.visualize.**W**(*W=None, second=10, saveable=True, shape=[28, 28], name='mnist', fig_idx=2396512*)

Visualize every columns of the weight matrix to a group of Greyscale img.

Parameters

W [numpy.array] The weight matrix

second [int] The display second(s) for the image(s), if saveable is False.

saveable [boolean] Save or plot the figure.

shape [a list with 2 int] The shape of feature image, MNIST is [28, 80].

name [a string] A name to save the image, if saveable is True.

fig_idx [int] matplotlib figure index.

Examples

```
>>> tl.visualize.W(network.all_params[0].eval(), second=10, saveable=True, name=
↪'weight_of_1st_layer', fig_idx=2012)
```

Visualize CNN 2d filter

tensorlayer.visualize.**CNN2d**(*CNN=None*, *second=10*, *saveable=True*, *name='cnn'*, *fig_idx=3119362*)

Display a group of RGB or Greyscale CNN masks.

Parameters

CNN [numpy.array] The image. e.g: 64 5x5 RGB images can be (5, 5, 3, 64).

second [int] The display second(s) for the image(s), if saveable is False.

saveable [boolean] Save or plot the figure.

name [a string] A name to save the image, if saveable is True.

fig_idx [int] matplotlib figure index.

Examples

```
>>> tl.visualize.CNN2d(network.all_params[0].eval(), second=10, saveable=True,␣
↪name='cnn1_mnist', fig_idx=2012)
```

2.9.3 Visualize images

Image by matplotlib

tensorlayer.visualize.**frame**(*I=None*, *second=5*, *saveable=True*, *name='frame'*, *cmap=None*, *fig_idx=12836*)

Display a frame(image). Make sure OpenAI Gym render() is disable before using it.

Parameters

I [numpy.array] The image

second [int] The display second(s) for the image(s), if saveable is False.

saveable [boolean] Save or plot the figure.

name [a string] A name to save the image, if saveable is True.

cmap [None or string] 'gray' for greyscale, None for default, etc.

fig_idx [int] matplotlib figure index.

Examples

```
>>> env = gym.make("Pong-v0")
>>> observation = env.reset()
>>> tl.visualize.frame(observation)
```

Images by matplotlib

tensorlayer.visualize.**images2d**(*images=None*, *second=10*, *saveable=True*, *name='images'*, *dtype=None*, *fig_idx=3119362*)

Display a group of RGB or Greyscale images.

Parameters

images [numpy.array] The images.

second [int] The display second(s) for the image(s), if saveable is False.

saveable [boolean] Save or plot the figure.

name [a string] A name to save the image, if saveable is True.

dtype [None or numpy data type] The data type for displaying the images.

fig_idx [int] matplotlib figure index.

Examples

```
>>> X_train, y_train, X_test, y_test = tl.files.load_cifar10_dataset(shape=(-1,␣
↪32, 32, 3), plotable=False)
>>> tl.visualize.images2d(X_train[0:100,:,:,:], second=10, saveable=False, name=
↪'cifar10', dtype=np.uint8, fig_idx=20212)
```

2.9.4 Visualize embeddings

tensorlayer.visualize.**tsne_embedding**(*embeddings*, *reverse_dictionary*, *plot_only=500*, *second=5*, *saveable=False*, *name='tsne'*, *fig_idx=9862*)

Visualize the embeddings by using t-SNE.

Parameters

embeddings [a matrix] The images.

reverse_dictionary [a dictionary] id_to_word, mapping id to unique word.

plot_only [int] The number of examples to plot, choice the most common words.

second [int] The display second(s) for the image(s), if saveable is False.

saveable [boolean] Save or plot the figure.

name [a string] A name to save the image, if saveable is True.

fig_idx [int] matplotlib figure index.

Examples

```
>>> see 'tutorial_word2vec_basic.py'
>>> final_embeddings = normalized_embeddings.eval()
>>> tl.visualize.tsne_embedding(final_embeddings, labels, reverse_dictionary,
...                   plot_only=500, second=5, saveable=False, name='tsne')
```

2.10 API - Operation System

Operation system, more functions can be found in TensorFlow API.

exit_tf([sess, port])	Close TensorFlow session, TensorBoard and Nvidia-process if available.
open_tb([logdir, port])	Open Tensorboard.
clear_all([printable])	Clears all the placeholder variables of keep prob, including keeping probabilities of all dropout, denoising, dropconnect etc.
set_gpu_fraction([sess, gpu_fraction])	Set the GPU memory fraction for the application.
disable_print()	Disable console output, suppress_stdout is recommended.
enable_print()	Enable console output, suppress_stdout is recommended.
suppress_stdout()	Temporarily disable console output.
get_site_packages_directory()	Print and return the site-packages directory.
empty_trash()	Empty trash folder.

2.10.1 TensorFlow functions

Close TF session and associated processes

tensorlayer.ops.**exit_tf**(*sess=None*, *port=6006*)

Close TensorFlow session, TensorBoard and Nvidia-process if available.

Parameters

sess [a session instance of TensorFlow] TensorFlow session

tb_port [an integer] TensorBoard port you want to close, 6006 as default.

Open TensorBoard

tensorlayer.ops.**open_tb**(*logdir='/tmp/tensorflow'*, *port=6006*)

Open Tensorboard.

Parameters

logdir [a string] Directory where your tensorboard logs are saved

port [an integer] TensorBoard port you want to open, 6006 is tensorboard default

Delete placeholder

tensorlayer.ops.**clear_all**(*printable=True*)

Clears all the placeholder variables of keep prob, including keeping probabilities of all dropout, denoising, dropconnect etc.

Parameters

printable [boolean] If True, print all deleted variables.

2.10.2 GPU functions

tensorlayer.ops.**set_gpu_fraction**(*sess=None*, *gpu_fraction=0.3*)

Set the GPU memory fraction for the application.

Parameters

sess [a session instance of TensorFlow] TensorFlow session

gpu_fraction [a float] Fraction of GPU memory, (0 ~ 1]

References

- TensorFlow using GPU

2.10.3 Console display

Disable print

`tensorlayer.ops.`**`disable_print`**`()`
Disable console output, `suppress_stdout` is recommended.

Examples

```
>>> print("You can see me")
>>> tl.ops.disable_print()
>>> print(" You can't see me")
>>> tl.ops.enable_print()
>>> print("You can see me")
```

Enable print

`tensorlayer.ops.`**`enable_print`**`()`
Enable console output, `suppress_stdout` is recommended.

Examples

- see tl.ops.disable_print()

Temporary disable print

`tensorlayer.ops.`**`suppress_stdout`**`()`
Temporarily disable console output.

References

- stackoverflow

Examples

```
>>> print("You can see me")
>>> with tl.ops.suppress_stdout():
>>>     print("You can't see me")
>>> print("You can see me")
```

2.10.4 Site packages information

tensorlayer.ops.**get_site_packages_directory**()
Print and return the site-packages directory.

Examples

```
>>> loc = tl.ops.get_site_packages_directory()
```

2.10.5 Trash

tensorlayer.ops.**empty_trash**()
Empty trash folder.

2.11 API - Activations

To make TensorLayer simple, we minimize the number of activation functions as much as we can. So we encourage you to use TensorFlow's function. TensorFlow provides `tf.nn.relu`, `tf.nn.relu6`, `tf.nn.elu`, `tf.nn.softplus`, `tf.nn.softsign` and so on. More TensorFlow official activation functions can be found here. For parametric activation, please read the layer APIs.

The shortcut of `tensorlayer.activation` is `tensorlayer.act`.

2.11.1 Your activation

Customizes activation function in TensorLayer is very easy. The following example implements an activation that multiplies its input by 2. For more complex activation, TensorFlow API will be required.

```
def double_activation(x):
    return x * 2
```

identity(x[, name])	The identity activation function, Shortcut is `linear`.
ramp([x, v_min, v_max, name])	The ramp activation function.
leaky_relu([x, alpha, name])	The LeakyReLU, Shortcut is `lrelu`.
swish(x[, name])	The Swish function, see Swish: a Self-Gated Activation Function.
pixel_wise_softmax(output[, name])	Return the softmax outputs of images, every pixels have multiple label, the sum of a pixel is 1.

2.11.2 Identity

tensorlayer.activation.**identity**(*x*, *name=None*)
The identity activation function, Shortcut is `linear`.

Parameters

x [a tensor input] input(s)

Returns

A 'Tensor' with the same type as 'x'.

2.11.3 Ramp

tensorlayer.activation.**ramp**(*x=None*, *v_min=0*, *v_max=1*, *name=None*)
The ramp activation function.

Parameters

x [a tensor input] input(s)

v_min [float] if input(s) smaller than v_min, change inputs to v_min

v_max [float] if input(s) greater than v_max, change inputs to v_max

name [a string or None] An optional name to attach to this activation function.

Returns

A 'Tensor' with the same type as 'x'.

2.11.4 Leaky Relu

tensorlayer.activation.**leaky_relu**(*x=None*, *alpha=0.1*, *name='lrelu'*)
The LeakyReLU, Shortcut is `lrelu`.

Modified version of ReLU, introducing a nonzero gradient for negative input.

Parameters

x [A *Tensor* with type *float*, *double*, *int32*, *int64*, *uint8*,] *int16*, or *int8*.

alpha [*float*. slope.]

name [a string or None] An optional name to attach to this activation function.

References

- Rectifier Nonlinearities Improve Neural Network Acoustic Models, Maas et al. (2013)

Examples

```
>>> network = tl.layers.DenseLayer(network, n_units=100, name = 'dense_lrelu',
...                 act= lambda x : tl.act.lrelu(x, 0.2))
```

2.11.5 Swish

tensorlayer.activation.**swish**(*x*, *name='swish'*)

The Swish function, see Swish: a Self-Gated Activation Function.

Parameters

x [a tensor input] input(s)

Returns

A 'Tensor' with the same type as 'x'.

2.11.6 Pixel-wise softmax

tensorlayer.activation.**pixel_wise_softmax**(*output*, *name='pixel_wise_softmax'*)

Return the softmax outputs of images, every pixels have multiple label, the sum of a pixel is 1. Usually be used for image segmentation.

Parameters

output [tensor]

- For 2d image, 4D tensor [batch_size, height, weight, channel], channel >= 2.
- For 3d image, 5D tensor [batch_size, depth, height, weight, channel], channel >= 2.

References

- tf.reverse

Examples

```
>>> outputs = pixel_wise_softmax(network.outputs)
>>> dice_loss = 1 - dice_coe(outputs, y_, epsilon=1e-5)
```

2.11.7 Parametric activation

See tensorlayer.layers.

2.12 API - Distribution (alpha)

Helper sessions and methods to run a distributed training. Check this minst example.

TaskSpecDef([type, index, trial, ps_hosts, ...])	Specification for the distributed task with the job name, index of the task, the parameter servers and the worker servers.
TaskSpec()	Returns the a *TaskSpecDef* based on the environment variables for distributed training.
DistributedSession([task_spec, ...])	Creates a distributed session.

2.12.1 Distributed training

TaskSpecDef

tensorlayer.distributed.**TaskSpecDef**(*type='master'*, *index=0*, *trial=None*, *ps_hosts=None*, *worker_hosts=None*, *master=None*)

Specification for the distributed task with the job name, index of the task, the parameter servers and the worker servers. If you want to use the last worker for continuous evaluation you can call the method *user_last_worker_as_evaluator* which returns a new `TaskSpecDef` object without the last worker in the cluster specification.

Parameters

type [A string with the job name, it will be *master*, *worker* or *ps*.]

index [The zero-based index of the task. Distributed training jobs will have a single] master task, one or more parameter servers, and one or more workers.

trial [The identifier of the trial being run.]

ps_hosts [A string with a coma separate list of hosts for the parameter servers] or a list of hosts.

worker_hosts [A string with a coma separate list of hosts for the worker servers] or a list of hosts.

master [A string with the master hosts]

References

- ML-engine trainer considerations

Create TaskSpecDef from environment variables

tensorlayer.distributed.**TaskSpec**()

Returns the a `TaskSpecDef` based on the environment variables for distributed training.

References

- ML-engine trainer considerations
- TensorPort Distributed Computing

Distributed session object

tensorlayer.distributed.**DistributedSession**(*task_spec=None*, *checkpoint_dir=None*, *scaffold=None*, *hooks=None*, *chief_only_hooks=None*, *save_checkpoint_secs=600*, *save_summaries_steps=<object object>*, *save_summaries_secs=<object object>*, *config=None*, *stop_grace_period_secs=120*, *log_step_count_steps=100*)

Creates a distributed session. It calls *MonitoredTrainingSession* to create a `MonitoredSession` for distributed training.

Parameters

task_spec [TaskSpecDef. The task spec definition from TaskSpec()]

checkpoint_dir [A string. Optional path to a directory where to restore] variables.

scaffold [A *Scaffold* used for gathering or building supportive ops. If] not specified, a default one is created. It's used to finalize the graph.

hooks [Optional list of *SessionRunHook* objects.]

chief_only_hooks [list of *SessionRunHook* objects. Activate these hooks if] *is_chief==True*, ignore otherwise.

save_checkpoint_secs [The frequency, in seconds, that a checkpoint is saved] using a default checkpoint saver. If *save_checkpoint_secs* is set to *None*, then the default checkpoint saver isn't used.

save_summaries_steps [The frequency, in number of global steps, that the] summaries are written to disk using a default summary saver. If both *save_summaries_steps* and *save_summaries_secs* are set to *None*, then the default summary saver isn't used. Default 100.

save_summaries_secs [The frequency, in secs, that the summaries are written] to disk using a default summary saver. If both *save_summaries_steps* and *save_summaries_secs* are set to *None*, then the default summary saver isn't used. Default not enabled.

config [an instance of *tf.ConfigProto* proto used to configure the session.] It's the *config* argument of constructor of *tf.Session*.

stop_grace_period_secs [Number of seconds given to threads to stop after] *close()* has been called.

log_step_count_steps [The frequency, in number of global steps, that the] global step/sec is logged.

References

- MonitoredTrainingSession

Examples

A simple example for distributed training where all the workers use the same dataset:

```
>>> task_spec = TaskSpec()
>>> with tf.device(task_spec.device_fn()):
>>>      tensors = create_graph()
>>> with tl.DistributedSession(task_spec=task_spec,
...                            checkpoint_dir='/tmp/ckpt') as session:
>>>      while not session.should_stop():
>>>           session.run(tensors)
```

An example where the dataset is shared among the workers (see https://www.tensorflow.org/programmers_guide/datasets):

```
>>> task_spec = TaskSpec()
>>> # dataset is a :class:`tf.data.Dataset` with the raw data
>>> dataset = create_dataset()
```

(continues on next page)

(continued from previous page)

```
>>> if task_spec is not None:
>>>     dataset = dataset.shard(task_spec.num_workers, task_spec.shard_index)
>>> # shuffle or apply a map function to the new sharded dataset, for example:
>>> dataset = dataset.shuffle(buffer_size=10000)
>>> dataset = dataset.batch(batch_size)
>>> dataset = dataset.repeat(num_epochs)
>>> # create the iterator for the dataset and the input tensor
>>> iterator = dataset.make_one_shot_iterator()
>>> next_element = iterator.get_next()
>>> with tf.device(task_spec.device_fn()):
>>>     # next_element is the input for the graph
>>>     tensors = create_graph(next_element)
>>> with tl.DistributedSession(task_spec=task_spec,
...                            checkpoint_dir='/tmp/ckpt') as session:
>>>     while not session.should_stop():
>>>         session.run(tensors)
```

Data sharding

In some cases we want to shard the data among all the training servers and not use all the data in all servers. TensorFlow >=1.4 provides some helper classes to work with data that support data sharding: Datasets

It is important in sharding that the shuffle or any non deterministic operation is done after creating the shards:

```
from tensorflow.contrib.data import TextLineDataset
from tensorflow.contrib.data import Dataset

task_spec = TaskSpec()
files_dataset = Dataset.list_files(files_pattern)
dataset = TextLineDataset(files_dataset)
dataset = dataset.map(your_python_map_function, num_threads=4)
if task_spec is not None:
    dataset = dataset.shard(task_spec.num_workers, task_spec.shard_index)
dataset = dataset.shuffle(buffer_size)
dataset = dataset.batch(batch_size)
dataset = dataset.repeat(num_epochs)
iterator = dataset.make_one_shot_iterator()
next_element = iterator.get_next()
with tf.device(task_spec.device_fn()):
    tensors = create_graph(next_element)
with tl.DistributedSession(task_spec=task_spec,
                           checkpoint_dir='/tmp/ckpt') as session:
    while not session.should_stop():
        session.run(tensors)
```

Logging

We can use task_spec to log only in the master server:

```
while not session.should_stop():
    should_log = task_spec.is_master() and your_conditions
    if should_log:
        results = session.run(tensors_with_log_info)
        logging.info(...)
```

(continues on next page)

(continued from previous page)

```
        else:
            results = session.run(tensors)
```

Continuous evaluation

You can use one of the workers to run an evaluation for the saved checkpoints:

```
import tensorflow as tf
from tensorflow.python.training import session_run_hook
from tensorflow.python.training.monitored_session import SingularMonitoredSession

class Evaluator(session_run_hook.SessionRunHook):
        def __init__(self, checkpoints_path, output_path):
            self.checkpoints_path = checkpoints_path
            self.summary_writer = tf.summary.FileWriter(output_path)
            self.lastest_checkpoint = ''

        def after_create_session(self, session, coord):
            checkpoint = tf.train.latest_checkpoint(self.checkpoints_path)
            # wait until a new check point is available
            while self.lastest_checkpoint == checkpoint:
                time.sleep(30)
                checkpoint = tf.train.latest_checkpoint(self.checkpoints_path)
            self.saver.restore(session, checkpoint)
            self.lastest_checkpoint = checkpoint

        def end(self, session):
            super(Evaluator, self).end(session)
            # save summaries
            step = int(self.lastest_checkpoint.split('-')[-1])
            self.summary_writer.add_summary(self.summary, step)

        def _create_graph():
            # your code to create the graph with the dataset

        def run_evaluation():
            with tf.Graph().as_default():
                summary_tensors = create_graph()
                self.saver = tf.train.Saver(var_list=tf_variables.trainable_variables())
                hooks = self.create_hooks()
                hooks.append(self)
                if self.max_time_secs and self.max_time_secs > 0:
                    hooks.append(StopAtTimeHook(self.max_time_secs))
                # this evaluation runs indefinitely, until the process is killed
                while True:
                    with SingularMonitoredSession(hooks=[self]) as session:
                        try:
                            while not sess.should_stop():
                                self.summary = session.run(summary_tensors)
                        except OutOfRangeError:
                            pass
                        # end of evaluation

task_spec = TaskSpec().user_last_worker_as_evaluator()
if task_spec.is_evaluator():
```

(continues on next page)

(continued from previous page)

```
    Evaluator().run_evaluation()
else:
    # run normal training
```

2.12.2 Session hooks

TensorFlow provides some Session Hooks to do some operations in the sessions. We added more to help with common operations.

Stop after maximum time

tensorlayer.distributed.**StopAtTimeHook**(*time_running*)

Hook that requests stop after a specified time.

Parameters

time_running: Maximum time running in seconds

Initialize network with checkpoint

tensorlayer.distributed.**LoadCheckpoint**(*saver*, *checkpoint*)

Hook that loads a checkpoint after the session is created.

```
>>> from tensorflow.python.ops import variables as tf_variables
>>> from tensorflow.python.training.monitored_session import␣
↪SingularMonitoredSession
>>>
>>> tensors = create_graph()
>>> saver = tf.train.Saver(var_list=tf_variables.trainable_variables())
>>> checkpoint_hook = LoadCheckpoint(saver, my_checkpoint_file)
>>> with tf.SingularMonitoredSession(hooks=[checkpoint_hook]) as session:
>>>     while not session.should_stop():
>>>         session.run(tensors)
```

2.13 API - Database

This is the alpha version of database management system. If you have trouble, you can ask for help on fangde.liu@imperial.ac.uk .

Note: We are writing up the documentation, please wait in patient.

2.13.1 Why TensorDB

TensorLayer is designed for production, aiming to be applied large scale machine learning application. TensorDB introduce the database infracture to address the many challenges in large scale machine learning project, such as:

1. How to mangage the training data and load the training datasets
2. When the dataset is so large that beyonds the storage limitation of one computer

3. How shoud we managment different models and version, and comparing different models.
4. How to automate the whole training, evaluaiton and deploy machine learning model automatically.

In TensorLayer system, we introduce the database technology to the issues above.

TensorDB is designed by following three principles.

Everything is Data

TensorDB is a data warehouse that stores that capture the whole machine learning development process. the data inside tensordb can be catagloried as:

1. Data and Labels: Which includes all the data for training, validation and prediction. The labels can be manually labelled or generated by machine
2. Model Architecture: This group store the different model architecture, which user can select to use
3. Model Parameters: This tables stores all the model parameters of echo in the training step.
4. Jobs: All the computation is cutted into several jobs. Each jobs constains some computing work load. for training , the jobs includes training data , the model parameter, the model architecture, how many epochs the training want to do. Similarity are the validation jobs and inference jobs.
5. Logs: The logs store all the step time and accuracy and other metric of each training steps and also the time stamps.

TensorDB in principal is a key-word based search engine. Each model, parameters, or training data are assigned many tags. The data are stored in two layers. On the top, there is the index layer, which instore the blob storage reference with all the tags assigned to the data, which is implemented based on NoSQL document database such as Mongodb. The second layer is used store big chunk of data, such as videos, medical images or image mask, which is usually implemented as file system. Our open source implementation is implemented based MongoDB. The blob data is in store in the gridfs while the tag index is stored in the documents.

Everying is identified by Query

Within TensorDB framework, any entity within the data warehouse, such as the data, model or jobs are specified by the database query language. The first advantage is the query is more efficient in space and can specify multiple objects in a concise way. The advantage such a design is to enable a highly flexible software system. data, model architecture and training are interchangeable. Many work can be implemented by simply rewire different components. This enable us to develop many new application just by change the query without change any applicaition code.

An pulling based Stream processing pipeline.

Also with a large dataset, we can assume that the data is unlimited. TensorDB provides a streaming interface, implemented in python as generators, it keeps return the new data during training. Also the training system have no clue of epochs, instead, it knows batchize and store parameters after how many steps.

Many techniques are introduced behind the streaming interface. The stream is implemented based on the database cursor technology, so for every search, only the cursors are returned, not the actual data. Only when the generator is evaluated, the acutal data is loaded. The data loading is further optimise:

1. Data are compressed and decompressed,
2. The dataloaded in bulk model to optimise the IO traffic
3. The argumentation or random sample are computed on the fly after the data are loaded into the local computer.
4. To optimise the space, the will also be a cache system that only store the recent blob data.

Based on streaming interface, TensorLayer can be implemented as a continuous machine learning. On the distributed system, the model training, validation and deployment can be running on different computers which all running continuously. The trainer can keeps on optimising the models, the evaluation keeps evaluating the recent added models and the deployment system keeps pulling the best models from the TensorDB warehouse.

2.13.2 Preparation

In principle, TensorDB is can be implemented on any documents NoSQL database system. The exisitng implementation is based on Mongodb. Further implementaiton on other database will be released depends on progress. It will be stragihtford to port the tensorDB system to google cloud , aws and azure.

The following tutorials are based on the MongoDb implmenetation.

Install MongoDB

The installation instruction of Mongodb can be found at MongoDB Docs there are also managed mongodb service from amazon or gcp, or mongo atlas from mongodb

User can also user docker, which is a powerful tool for deploy software .

After install mongodb, a mongod db management tool with graphic user interface will be extremely valuale.

Users can install the Studio3T(mongochef), which is free for none commerical user interface. studio3t

Start MongoDB service

After mongodb is installed, you shoud start the database.

`mongod start`

You can specificy the path the database files with `-d` flag

2.13.3 Quick Start

A fully working example with mnist training set is the _TensorLabDemo.ipnb_

Connect to database

To use TensorDB mongodb implmentaiton, you need pymongo client.

you can install it by

```
pip install pymongo
pip install lz4
```

it is very strateford to connected to the TensorDB system. you can try the following code

```
from tensorlayer.db import TensorDB
db = TensorDB(ip='127.0.0.1', port=27017, db_name='your_db', user_name=None,␣
↪password=None, studyID='ministMLP')
```

The `ip` is the ip address of the database, and `port` number is number of mongodb. You may need to specificy the database name and studyid. The study id is an unique identifier for an experiement.

TensorDB stores different study in one data warehouse. This has pros and cons, the benefits is that suppose the each study we try a different model architecutre, it is very easy for us to evaluate different model architecture.

log and parameters

The basic application is use TensorDB to save the model parameters and training/evaluation/testing logs. to use tensorDB, this can be easily done by replacing the print function by the db.log function

For save the trainning log, we have `db.train_log`

and

`db. save_parameter`

methods

Suppose we save the log each step and save the parameters each epoch, we can have the code like this

```
for epoch in range(0,epoch_count):
   [~,ac]=sess.run([train_op,loss],feed_dict({x:x,y:y_}
   db.train_log({'accuracy':ac})
db.save_parameter(sess.run(network.all_parameters),{'acc':ac})
```

the code for save validation log and test log are similar.

Model Architecture and Jobs

TensorDb also supporting the model architecture and jobs system in the current version, both the model architecture and job are just simply strings. it is up to the user to specifiy how to convert the string back to models or job. for example, in many our our cases, we just simpliy specify the python code.

```
code= '''
print "hello
'''
db.save_model_architecutre(code,{'name':'print'}

c,fid = db.find_model_architecutre({'name':'print'})
exec c

db.push_job(code,{'type':'train'})

## worker
code = db.pop_job()
exec code
```

Database Interface

The trainning set is managed by a seperate database. each application has its own database. However, all the database interface should support two interface, 1. find_data, 2. data_generator

and example for minist dataset is include in the TensorLabDemo code

Data Importing

With a database, the development workflow is very flexible. As long as the comtent in the database in the same, user can use whatever tools to write into the database

the TesorLabDemo has an import data interface, which allow the user to injecting data in future

user can import data by the following code

```
db.import_data(X,y,{'type':'train'})
```

2.13.4 Application Framework

In fact, in real application, we rarely code everything from scrach and using the tensorDB interface directly. as demostrate in the TensorLabDemo

we implemented 4 class each with a well defined interace. 1. The dataset. 2. The TensorDb 3. The Model, model is loggically a full compoment can be trained, evaluate and deployed. It has property like parameters 4. The DBLogger, which is connecttor from model to tensorDB, which is implemented as callback functions, automatically called at each batch_step and each epoch.

users can based on the TensorLabDemo code, overrite the interface to suits their own applicaions needs.

when training, the overall archtiecture is first, find a data generator from the dataset module

```
g=datase.data_generator({"type":[your type]})
```

then intialize a model with a name

```
m=model('mytes')
```

during training, connected the db logger and tensordb togehter

```
m.fit_generator(g,dblogger(tensordb,m),1000,100)
```

if the work is distributed, we have to save the model archtiecture and reload and excute it

```
db.save_model_architecture(code,{'name':'mlp'})
db.push_job({'name':'mlp'},{'type':XXXX},{'batch:1000','epoch':100)
```

the worker will run the job as the following code

```
j=job.pop
g=dataset.data_generator(j.filter)
c=tensordb.load_model_architecutre(j.march)
exec c
m=model()
m.fit_generator(g,dblooger(tensordb,m),j.bach_size,j.epoch}
```

Experimental Database Management System.

Latest Version

class tensorlayer.db.**TensorDB**(*ip='localhost'*, *port=27017*, *db_name='db_name'*, *user_name=None*, *password='password'*, *studyID=None*)

TensorDB is a MongoDB based manager that help you to manage data, network topology, parameters and logging.

Parameters

ip [string, localhost or IP address.]

port [int, port number.]

db_name [string, database name.]

user_name [string, set to None if it donnot need authentication.]

password [string.]

Methods

save_params([params, args])	Save parameters into MongoDB Buckets, and save the file ID into Params Collections.

del_job	
del_params	
del_test_log	
del_train_log	
del_valid_log	
find_all_params	
find_one_job	
find_one_params	
load_model_architecture	
peek_job	
push_job	
run_job	
save_job	
save_model_architecture	
test_log	
train_log	
valid_log	

save_params (*params=[]*, *args={}*)

Save parameters into MongoDB Buckets, and save the file ID into Params Collections.

Parameters

params [a list of parameters]

args [dictionary, item meta data.]

Returns

f_id [the Buckets ID of the parameters.]

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

t

Index

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

Z