TensorLayer Documentation

Release 1.2.8

TensorLayer contributors

May 17, 2018

Contents

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.

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 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_) 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_)
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
sess.run(tf.initialize_all_variables())

# 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,
```

(continues on next page)

(continued from previous page)

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

tensorlayer:Instantiate InputLayer   input_layer (?, 784)
tensorlayer:Instantiate DropoutLayer drop1: keep: 0.800000
tensorlayer:Instantiate DenseLayer   relu1: 800, relu
```

(continues on next page)

(continued from previous page)

```
tensorlayer:Instantiate DropoutLayer drop2: keep: 0.500000
tensorlayer:Instantiate DenseLayer   relu2: 800, relu
tensorlayer:Instantiate DropoutLayer drop3: keep: 0.500000
tensorlayer:Instantiate 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 a unsupervised learning models which 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.

```
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
```

(continues on next page)

(continued from previous page)

```
                        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 optimizer, 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
```

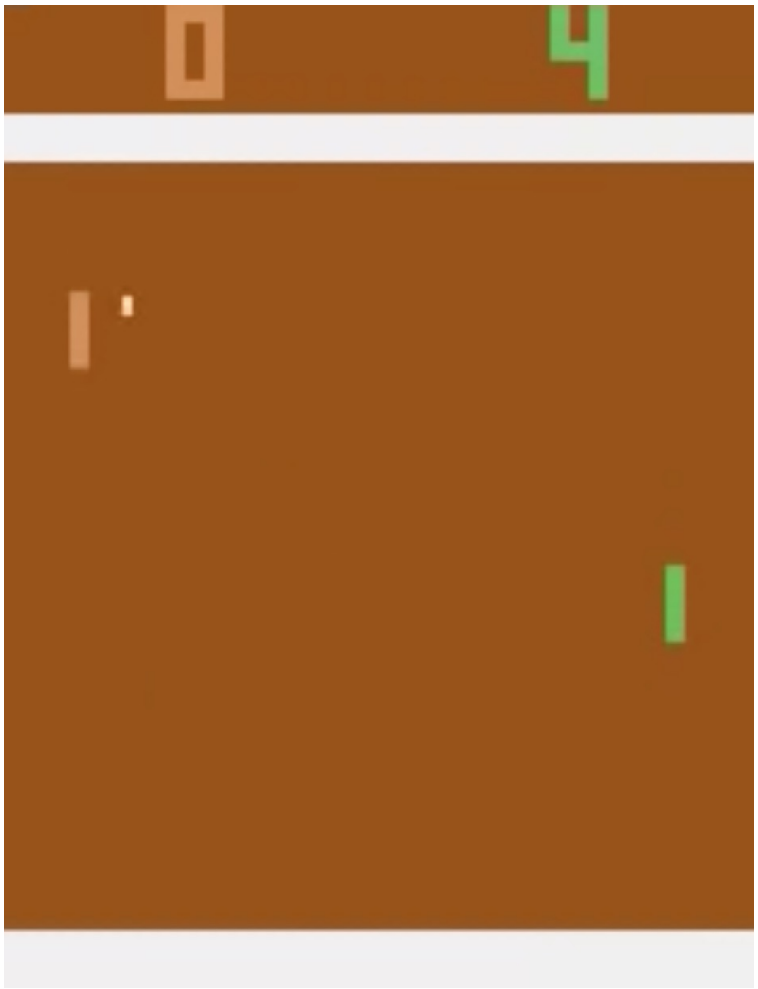
Before running the tutorial code, you need to install OpenAI gym environment which is a benchmark for Reinforcement Learning. If everything is set up correctly, you will get an output like the following:

```
[2016-07-12 09:31:59,760] Making new env: Pong-v0
  tensorlayer:Instantiate InputLayer input_layer (?, 6400)
  tensorlayer:Instantiate DenseLayer relu1: 200, relu
  tensorlayer:Instantiate 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])
```

(continues on next page)

(continued from previous page)

```
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.

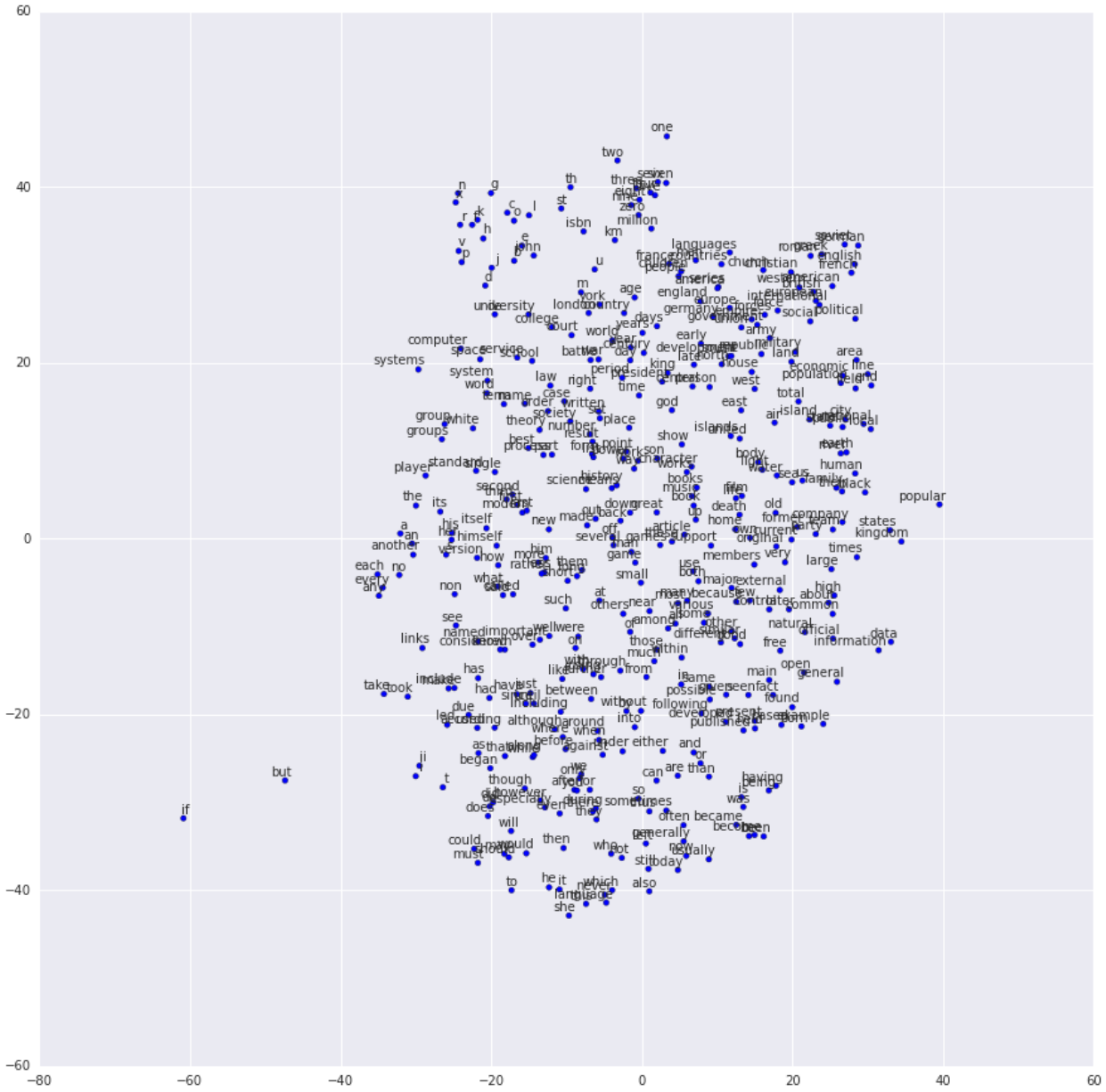
Finally, you can try the model on different tasks (games).

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')

sess.run(tf.initialize_all_variables())

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 next words in a text given a history of previous words. In previous example , `num_steps (sequence length)` is 3.

```
python tutorial_ptb_lstm.py
```

The script provides three settings (small, medium, large), larger model has better performance, you can choice different setting in:

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

If you choice 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)

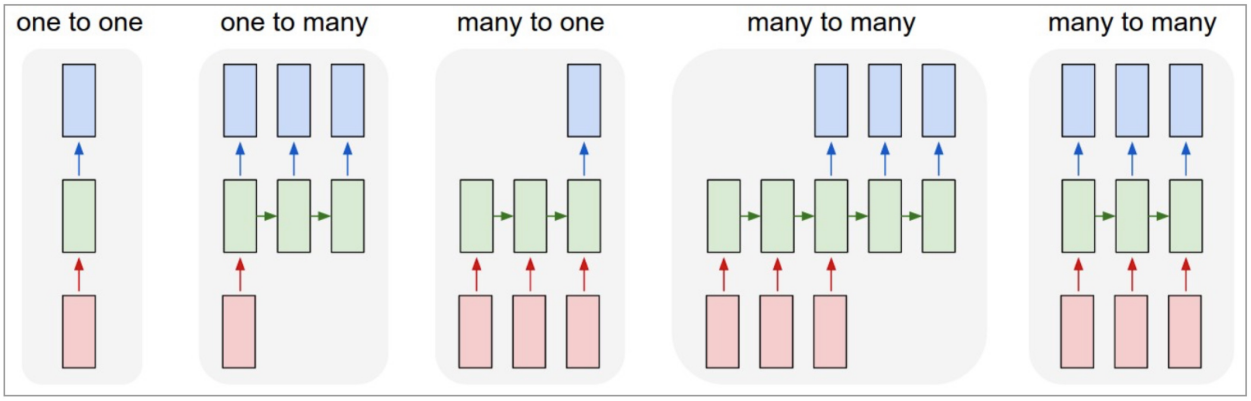
```
0.204 perplexity: 614.071 speed: 8839 wps
0.304 perplexity: 495.485 speed: 8889 wps
0.404 perplexity: 427.381 speed: 8940 wps
0.504 perplexity: 383.063 speed: 8920 wps
0.604 perplexity: 345.135 speed: 8920 wps
0.703 perplexity: 319.263 speed: 8949 wps
0.803 perplexity: 298.774 speed: 8975 wps
0.903 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 proves RNN is able to modeling language, but this example did not do something practical. However, you should read through this example and "Understand LSTM" in order to understand the basic of RNN. After that, you learn how to generate text, how to achieve language translation and how to build a questions 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 do not describe more about RNN, 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 typically 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 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."

The model is built as follow. Firstly, transfer the words into word vectors by looking up an embedding matrix, in this tutorial, no pre-training on embedding matrix. Secondly, we stacked two LSTMs together use dropout among the embedding layer, LSTM layers and output layer for regularization. In the last 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 DenseLayer after it, then compute 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 layer 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.nn.rnn_cell.BasicLSTMCell,
            cell_init_args={'forget_bias': 0.0},
```

(continues on next page)

(continued from previous page)

```
            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.nn.rnn_cell.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 seem as how many concurrent computations. As the following example shows, the first batch learn the sequence information by using 0 to 9. The second batch learn the sequence information by using 10 to 19. So it ignores the information from 9 to 10 !n If only if we set the batch_size = 1, it will consider all information from 0 to 20.

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

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

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

A example of generating training data as follow:

```
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 averaged 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 averaged 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, this example decreases the initial learning rate after several epochs (defined by `max_epoch`), by multiplying a `lr_decay`. In addition, 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))
```

If the epoch index greater than `max_epoch`, 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 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 `1`. Then we can output the next word step by step, instead of predict 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 understand Synced sequence input and output. Let think about Many to one (Sequence input and one output), 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 show 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 Run the Translation example

```
python tutorial_translate.py
```

This script is going to training a neural network to translate English to French. If everything is correct, you will see.

- Download WMT English-to-French translation data, includes training and testing data.
- Create vocabulary files for English and French from training data.
- Create the tokenized training and testing data from original training and testing data.

```
Prepare raw data
Load or Download WMT English-to-French translation > wmt
Training data : wmt/giga-fren.release2
Testing data : wmt/newstest2013

Create vocabularies
Vocabulary of French : wmt/vocab40000.fr
Vocabulary of English : wmt/vocab40000.en
Creating vocabulary wmt/vocab40000.fr from data wmt/giga-fren.release2.fr
  processing line 100000
  processing line 200000
  processing line 300000
  processing line 400000
  processing line 500000
  processing line 600000
  processing line 700000
  processing line 800000
  processing line 900000
  processing line 1000000
  processing line 1100000
  processing line 1200000
  ...
  processing line 22500000
Creating vocabulary wmt/vocab40000.en from data wmt/giga-fren.release2.en
  processing line 100000
  ...
```

(continues on next page)

(continued from previous page)

```
  processing line 22500000

...
```

Firstly, we download English-to-French translation data from the WMT'15 Website. The training and testing data as follow. The training data is used to train the model, the testing data is used to evaluate the model.

```
wmt/training-giga-fren.tar  <-- Training data for English-to-French (2.6GB)
                                giga-fren.release2.* are extracted from it.
wmt/dev-v2.tgz              <-- Testing data for different language (21.4MB)
                                newstest2013.* are extracted from it.

wmt/giga-fren.release2.fr   <-- Training data of French   (4.57GB)
wmt/giga-fren.release2.en   <-- Training data of English  (3.79GB)

wmt/newstest2013.fr         <-- Testing data of French    (393KB)
wmt/newstest2013.en         <-- Testing data of English   (333KB)
```

As `giga-fren.release2.*` are the training data, the context of `giga-fren.release2.fr` look as follow.

```
Il a transformé notre vie | Il a transformé la société | Son fonctionnement | La␣
↪technologie, moteur du changement Accueil | Concepts | Enseignants | Recherche |␣
↪Aperçu | Collaborateurs | Web HHCC | Ressources | Commentaires Musée virtuel du␣
↪Canada
Plan du site
Rétroaction
Crédits
English
Qu'est-ce que la lumière?
La découverte du spectre de la lumière blanche Des codes dans la lumière Le spectre␣
↪électromagnétique Les spectres d'émission Les spectres d'absorption Les années-
↪lumière La pollution lumineuse
Le ciel des premiers habitants La vision contemporaine de l'Univers L'astronomie pour␣
↪tous
Bande dessinée
Liens
Glossaire
Observatoires
...
```

While `giga-fren.release2.en` look as follow, we can see words or sentences are separated by `|` or `\n`.

```
Changing Lives | Changing Society | How It Works | Technology Drives Change Home |␣
↪Concepts | Teachers | Search | Overview | Credits | HHCC Web | Reference | Feedback␣
↪Virtual Museum of Canada Home Page
Site map
Feedback
Credits
Français
What is light ?
The white light spectrum Codes in the light The electromagnetic spectrum Emission␣
↪spectra Absorption spectra Light-years Light pollution
The sky of the first inhabitants A contemporary vison of the Universe Astronomy for␣
↪everyone
Cartoon
Links
```

(continues on next page)

(continued from previous page)

```
Glossary
Observatories
```

The testing data `newstest2013.en` and `newstest2013.fr` look as follow.

```
newstest2013.en :
A Republican strategy to counter the re-election of Obama
Republican leaders justified their policy by the need to combat electoral fraud.
However, the Brennan Centre considers this a myth, stating that electoral fraud is␣
↪rarer in the United States than the number of people killed by lightning.

newstest2013.fr :
Une stratégie républicaine pour contrer la réélection d'Obama
Les dirigeants républicains justifièrent leur politique par la nécessité de lutter␣
↪contre la fraude électorale.
Or, le Centre Brennan considère cette dernière comme un mythe, affirmant que la␣
↪fraude électorale est plus rare aux États-Unis que le nombre de personnes tuées par␣
↪la foudre.
```

After downloading the dataset, it start to create vocabulary files, `vocab40000.fr` and `vocab40000.en` from the training data `giga-fren.release2.fr` and `giga-fren.release2.en`, usually it will take a while. The number `40000` reflects the vocabulary size.

The `vocab40000.fr` (381KB) stores one-item-per-line as follow.

```
_PAD
_GO
_EOS
_UNK
de
,
.
'
la
et
des
les
à
le
du
l
en
)
d
0
(
00
pour
dans
un
que
une
sur
au
0000
a
par
```

The `vocab40000.en` (344KB) stores one-item-per-line as follow as well.

```
_PAD
_GO
_EOS
_UNK
the
.
,
of
and
to
in
a
)
(
0
for
00
that
is
on
The
0000
be
by
with
or
:
as
"
000
are
;
```

And then, we start to create the tokenized training and testing data for both English and French. It will take a while as well.

```
Tokenize data
Tokenizing data in wmt/giga-fren.release2.fr  <-- Training data of French
  tokenizing line 100000
  tokenizing line 200000
  tokenizing line 300000
  tokenizing line 400000
  ...
  tokenizing line 22500000
Tokenizing data in wmt/giga-fren.release2.en  <-- Training data of English
  tokenizing line 100000
  tokenizing line 200000
  tokenizing line 300000
  tokenizing line 400000
  ...
  tokenizing line 22500000
Tokenizing data in wmt/newstest2013.fr        <-- Testing data of French
Tokenizing data in wmt/newstest2013.en        <-- Testing data of English
```

In the end, all files we have as follow.

```
wmt/training-giga-fren.tar  <-- Compressed Training data for English-to-French (2.6GB)
                                giga-fren.release2.* are extracted from it.
wmt/dev-v2.tgz              <-- Compressed Testing data for different language (21.
↪4MB)
                                newstest2013.* are extracted from it.

wmt/giga-fren.release2.fr   <-- Training data of French   (4.57GB)
wmt/giga-fren.release2.en   <-- Training data of English  (3.79GB)

wmt/newstest2013.fr         <-- Testing data of French    (393KB)
wmt/newstest2013.en         <-- Testing data of English   (333KB)

wmt/vocab40000.fr           <-- Vocabulary of French      (381KB)
wmt/vocab40000.en           <-- Vocabulary of English     (344KB)

wmt/giga-fren.release2.ids40000.fr   <-- Tokenized Training data of French (2.81GB)
wmt/giga-fren.release2.ids40000.en   <-- Tokenized Training data of English (2.38GB)

wmt/newstest2013.ids40000.fr         <-- Tokenized Testing data of French (268KB)
wmt/newstest2013.ids40000.en         <-- Tokenized Testing data of English (232KB)
```

Now, read all tokenized data into buckets and compute the number of data of each bucket.

```
Read development (test) data into buckets
dev data: (5, 10) [[13388, 4, 949], [23113, 8, 910, 2]]
en word_ids: [13388, 4, 949]
en context: [b'Preventing', b'the', b'disease']
fr word_ids: [23113, 8, 910, 2]
fr context: [b'Pr\xc3\xa9venir', b'la', b'maladie', b'_EOS']

Read training data into buckets (limit: 0)
  reading data line 100000
  reading data line 200000
  reading data line 300000
  reading data line 400000
  reading data line 500000
  reading data line 600000
  reading data line 700000
  reading data line 800000
  ...
  reading data line 22400000
  reading data line 22500000
train_bucket_sizes: [239121, 1344322, 5239557, 10445326]
train_total_size: 17268326.0
train_buckets_scale: [0.013847375825543252, 0.09169638099257565, 0.3951164693091849,␣
↪1.0]
train data: (5, 10) [[1368, 3344], [1089, 14, 261, 2]]
en word_ids: [1368, 3344]
en context: [b'Site', b'map']
fr word_ids: [1089, 14, 261, 2]
fr context: [b'Plan', b'du', b'site', b'_EOS']

the num of training data in each buckets: [239121, 1344322, 5239557, 10445326]
the num of training data: 17268326
train_buckets_scale: [0.013847375825543252, 0.09169638099257565, 0.3951164693091849,␣
↪1.0]
```

Start training by using the tokenized bucket data, the training process can only be terminated by stop the program.

When `steps_per_checkpoint = 10` you will see.

```
Create Embedding Attention Seq2seq Model

global step 10 learning rate 0.5000 step-time 22.26 perplexity 12761.50
  eval: bucket 0 perplexity 5887.75
  eval: bucket 1 perplexity 3891.96
  eval: bucket 2 perplexity 3748.77
  eval: bucket 3 perplexity 4940.10
global step 20 learning rate 0.5000 step-time 20.38 perplexity 28761.36
  eval: bucket 0 perplexity 10137.01
  eval: bucket 1 perplexity 12809.90
  eval: bucket 2 perplexity 15758.65
  eval: bucket 3 perplexity 26760.93
global step 30 learning rate 0.5000 step-time 20.64 perplexity 6372.95
  eval: bucket 0 perplexity 1789.80
  eval: bucket 1 perplexity 1690.00
  eval: bucket 2 perplexity 2190.18
  eval: bucket 3 perplexity 3808.12
global step 40 learning rate 0.5000 step-time 16.10 perplexity 3418.93
  eval: bucket 0 perplexity 4778.76
  eval: bucket 1 perplexity 3698.90
  eval: bucket 2 perplexity 3902.37
  eval: bucket 3 perplexity 22612.44
global step 50 learning rate 0.5000 step-time 14.84 perplexity 1811.02
  eval: bucket 0 perplexity 644.72
  eval: bucket 1 perplexity 759.16
  eval: bucket 2 perplexity 984.18
  eval: bucket 3 perplexity 1585.68
global step 60 learning rate 0.5000 step-time 19.76 perplexity 1580.55
  eval: bucket 0 perplexity 1724.84
  eval: bucket 1 perplexity 2292.24
  eval: bucket 2 perplexity 2698.52
  eval: bucket 3 perplexity 3189.30
global step 70 learning rate 0.5000 step-time 17.16 perplexity 1250.57
  eval: bucket 0 perplexity 298.55
  eval: bucket 1 perplexity 502.04
  eval: bucket 2 perplexity 645.44
  eval: bucket 3 perplexity 604.29
global step 80 learning rate 0.5000 step-time 18.50 perplexity 793.90
  eval: bucket 0 perplexity 2056.23
  eval: bucket 1 perplexity 1344.26
  eval: bucket 2 perplexity 767.82
  eval: bucket 3 perplexity 649.38
global step 90 learning rate 0.5000 step-time 12.61 perplexity 541.57
  eval: bucket 0 perplexity 180.86
  eval: bucket 1 perplexity 350.99
  eval: bucket 2 perplexity 326.85
  eval: bucket 3 perplexity 383.22
global step 100 learning rate 0.5000 step-time 18.42 perplexity 471.12
  eval: bucket 0 perplexity 216.63
  eval: bucket 1 perplexity 348.96
  eval: bucket 2 perplexity 318.20
  eval: bucket 3 perplexity 389.92
global step 110 learning rate 0.5000 step-time 18.39 perplexity 474.89
  eval: bucket 0 perplexity 8049.85
  eval: bucket 1 perplexity 1677.24
  eval: bucket 2 perplexity 936.98
```

(continues on next page)

(continued from previous page)

```
  eval: bucket 3 perplexity 657.46
global step 120 learning rate 0.5000 step-time 18.81 perplexity 832.11
  eval: bucket 0 perplexity 189.22
  eval: bucket 1 perplexity 360.69
  eval: bucket 2 perplexity 410.57
  eval: bucket 3 perplexity 456.40
global step 130 learning rate 0.5000 step-time 20.34 perplexity 452.27
  eval: bucket 0 perplexity 196.93
  eval: bucket 1 perplexity 655.18
  eval: bucket 2 perplexity 860.44
  eval: bucket 3 perplexity 1062.36
global step 140 learning rate 0.5000 step-time 21.05 perplexity 847.11
  eval: bucket 0 perplexity 391.88
  eval: bucket 1 perplexity 339.09
  eval: bucket 2 perplexity 320.08
  eval: bucket 3 perplexity 376.44
global step 150 learning rate 0.4950 step-time 15.53 perplexity 590.03
  eval: bucket 0 perplexity 269.16
  eval: bucket 1 perplexity 286.51
  eval: bucket 2 perplexity 391.78
  eval: bucket 3 perplexity 485.23
global step 160 learning rate 0.4950 step-time 19.36 perplexity 400.80
  eval: bucket 0 perplexity 137.00
  eval: bucket 1 perplexity 198.85
  eval: bucket 2 perplexity 276.58
  eval: bucket 3 perplexity 357.78
global step 170 learning rate 0.4950 step-time 17.50 perplexity 541.79
  eval: bucket 0 perplexity 1051.29
  eval: bucket 1 perplexity 626.64
  eval: bucket 2 perplexity 496.32
  eval: bucket 3 perplexity 458.85
global step 180 learning rate 0.4950 step-time 16.69 perplexity 400.65
  eval: bucket 0 perplexity 178.12
  eval: bucket 1 perplexity 299.86
  eval: bucket 2 perplexity 294.84
  eval: bucket 3 perplexity 296.46
global step 190 learning rate 0.4950 step-time 19.93 perplexity 886.73
  eval: bucket 0 perplexity 860.60
  eval: bucket 1 perplexity 910.16
  eval: bucket 2 perplexity 909.24
  eval: bucket 3 perplexity 786.04
global step 200 learning rate 0.4901 step-time 18.75 perplexity 449.64
  eval: bucket 0 perplexity 152.13
  eval: bucket 1 perplexity 234.41
  eval: bucket 2 perplexity 249.66
  eval: bucket 3 perplexity 285.95
...
global step 980 learning rate 0.4215 step-time 18.31 perplexity 208.74
  eval: bucket 0 perplexity 78.45
  eval: bucket 1 perplexity 108.40
  eval: bucket 2 perplexity 137.83
  eval: bucket 3 perplexity 173.53
global step 990 learning rate 0.4173 step-time 17.31 perplexity 175.05
  eval: bucket 0 perplexity 78.37
  eval: bucket 1 perplexity 119.72
  eval: bucket 2 perplexity 169.11
  eval: bucket 3 perplexity 202.89
```

(continues on next page)

(continued from previous page)

```
global step 1000 learning rate 0.4173 step-time 15.85 perplexity 174.33
  eval: bucket 0 perplexity 76.52
  eval: bucket 1 perplexity 125.97
  eval: bucket 2 perplexity 150.13
  eval: bucket 3 perplexity 181.07
...
```

After training the model for 350000 steps, you can play with the translation by switch `main_train()` to `main_decode()`. You type in a English sentence, the program will outputs a French sentence.

```
Reading model parameters from wmt/translate.ckpt-350000
>  Who is the president of the United States?
Qui est le président des États-Unis ?
```

1.2.12 Understand Translation

Seq2seq

Sequence to sequence model is commonly be used to translate a language to another. Actually it can do many thing you can't imagine, we can translate a long sentence into short and simple sentence, for example, translation going from Shakespeare to modern English. With CNN, we can also translate a video into a sentence, i.e. video captioning.

If you just want to use Seq2seq but not going to design a new algorithm, the only think you need to consider is the data format including how to split the words, how to tokenize the words etc. In this tutorial, we described a lot about data formating.

Basics

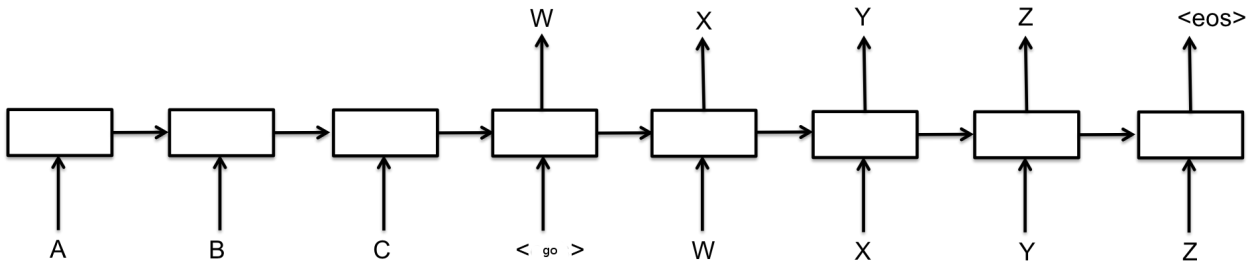
Sequence to sequence model is a type of "Many to many" but different with Synced sequence input and output in PTB tutorial. Seq2seq generates sequence output after feeding all sequence inputs. The following two methods can improve the accuracy:

- Reversing the inputs
- Attention mechanism

To speed up the computation, we used:

- Sampled softmax

Karpathy's blog described Seq2seq as: "(4) Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French)."



As the above figure shows, the encoder inputs, decoder inputs and targets are:

```
encoder_input =  A    B    C
decoder_input =  <go> W    X    Y    Z
targets       =  W    X    Y    Z    <eos>

Note: in the code, the size of targets is one smaller than the size
of decoder_input, not like this figure. More details will be show later.
```

Papers

The English-to-French example implements a multi-layer recurrent neural network as encoder, and an Attention-based decoder. It is the same as the model described in this paper:

- Grammar as a Foreign Language

The example uses sampled softmax to handle large output vocabulary size. In this example, as `target_vocab_size=4000`, for vocabularies smaller than `512`, it might be a better idea to just use a standard softmax loss. Sampled softmax is described in Section 3 of the this paper:

- On Using Very Large Target Vocabulary for Neural Machine Translation

Reversing the inputs and Multi-layer cells have been successfully used in sequence-to-sequence models for translation has beed described in this paper:

- Sequence to Sequence Learning with Neural Networks

Attention mechanism allows the decoder more direct access to the input, it was described in this paper:

- Neural Machine Translation by Jointly Learning to Align and Translate

Alternatively, the model can also be implemented by a single-layer version, but with Bi-directional encoder, was presented in this paper:

- Neural Machine Translation by Jointly Learning to Align and Translate

Implementation

Bucketing and Padding

(Note that, TensorLayer supports Dynamic RNN layer after v1.2, so bucketing is not longer necessary in many cases)

Bucketing is a method to efficiently handle sentences of different length. When translating English to French, we will have English sentences of different lengths `L1` on input, and French sentences of different lengths `L2` on output. We should in principle create a seq2seq model for every pair `(L1, L2+1)` (prefixed by a GO symbol) of lengths of an English and French sentence.

To minimize the number of buckets and find the closest bucket for each pair, then we could just pad every sentence with a special PAD symbol in the end if the bucket is bigger than the sentence

We use a number of buckets and pad to the closest one for efficiency. In this example, we used 4 buckets as follow.

```
buckets = [(5, 10), (10, 15), (20, 25), (40, 50)]
```

If the input is an English sentence with `3` tokens, and the corresponding output is a French sentence with `6` tokens, then they will be put in the first bucket and padded to length `5` for encoder inputs (English sentence), and length `10` for decoder inputs. If we have an English sentence with 8 tokens and the corresponding French sentence has 18 tokens, then they will be fit into `(20, 25)` bucket.

In other word, bucket `(I, O)` is `(encoder_input_size, decoder_inputs_size)`.

Given a pair of `[["I", "go", "."], ["Je", "vais", "."]]` in tokenized format, we fit it into bucket `(5, 10)`. The training data of encoder inputs representing `[PAD PAD "." "go" "I"]` and decoder inputs `[GO "Je" "vais" "." EOS PAD PAD PAD PAD PAD]`. The targets are decoder inputs shifted by one. The `target_weights` is the mask of `targets`.

```
bucket = (I, O) = (5, 10)
encoder_inputs = [PAD PAD "." "go" "I"]                      <-- 5  x batch_size
decoder_inputs = [GO "Je" "vais" "." EOS PAD PAD PAD PAD PAD] <-- 10 x batch_size
target_weights = [1   1     1     1   0 0 0 0 0 0 0]          <-- 10 x batch_size
targets        = ["Je" "vais" "." EOS PAD PAD PAD PAD PAD]    <-- 9  x batch_size
```

In this example, one sentence is represented by one column, so assume `batch_size = 3`, `bucket = (5, 10)` the training data will look like:

```
encoder_inputs     decoder_inputs     target_weights     targets
0    0    0        1    1    1        1    1    1        87   71   16748
0    0    0        87   71   16748    1    1    1        2    3    14195
0    0    0        2    3    14195    0    1    1        0    2    2
0    0    3233     0    2    2        0    0    0        0    0    0
3    698  4061     0    0    0        0    0    0        0    0    0
                   0    0    0        0    0    0        0    0    0
                   0    0    0        0    0    0        0    0    0
                   0    0    0        0    0    0        0    0    0
                   0    0    0        0    0    0        0    0    0
                   0    0    0        0    0    0

where 0 : _PAD    1 : _GO     2 : _EOS      3 : _UNK
```

During training, the decoder inputs are the targets, while during prediction, the next decoder input is the last decoder output.

Special vocabulary symbols, punctuations and digits

The special vocabulary symbols in this example are:

```
_PAD = b"_PAD"
_GO = b"_GO"
_EOS = b"_EOS"
_UNK = b"_UNK"
PAD_ID = 0      <-- index (row number) in vocabulary
GO_ID = 1
EOS_ID = 2
UNK_ID = 3
_START_VOCAB = [_PAD, _GO, _EOS, _UNK]
```

```
        ID    MEANINGS
_PAD    0     Padding, empty word
_GO     1     1st element of decoder_inputs
_EOS    2     End of Sentence of targets
_UNK    3     Unknown word, words do not exist in vocabulary will be marked as 3
```

For digits, the `normalize_digits` of creating vocabularies and tokenized dataset must be consistent, if `normalize_digits=True` all digits will be replaced by `0`. Like `123` to `000`, *9* to *0* and *1990-05* to *0000-00*, then *000*, *0* and *0000-00* etc will be the words in the vocabulary (see `vocab40000.en`).

Otherwise, if `normalize_digits=False`, different digits will be seem in the vocabulary, then the vocabulary

size will be very big. The regular expression to find digits is `_DIGIT_RE = re.compile(br"\d")`. (see `tl.nlp.create_vocabulary()` and `tl.nlp.data_to_token_ids()`)

For word split, the regular expression is `_WORD_SPLIT = re.compile(b"([.,!?\"':;)(])")`, this means use the symbols like `[ . , ! ? " ' : ; ) ( ]` and space to split the sentence, see `tl.nlp.basic_tokenizer()` which is the default tokenizer of `tl.nlp.create_vocabulary()` and `tl.nlp.data_to_token_ids()`.

All punctuation marks, such as `. , ) (` are all reserved in the vocabularies of both English and French.

Sampled softmax

Sampled softmax is a method to reduce the computation of cost so as to handle large output vocabulary. Instead of compute the cross-entropy of large output, we compute the loss from samples of `num_samples`.

Dataset iteration

The iteration is done by `EmbeddingAttentionSeq2seqWrapper.get_batch`, which get a random batch of data from the specified bucket, prepare for step. The data

Loss and update expressions

The `EmbeddingAttentionSeq2seqWrapper` has built in SGD optimizer.

What Next?

Try other applications.

1.2.13 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). A multi-layer perceptron implementation for MNIST classification task, see `tutorial_mnist_simple.py` on GitHub.

1.3.2 Computer Vision

- Denoising Autoencoder (MNIST). A multi-layer perceptron implementation for MNIST classification task, see `tutorial_mnist.py` on GitHub.
- Stacked Denoising Autoencoder and Fine-Tuning (MNIST). A multi-layer perceptron implementation for MNIST classification task, see `tutorial_mnist.py` on GitHub.
- Convolutional Network (MNIST). A Convolutional neural network implementation for classifying MNIST dataset, see `tutorial_mnist.py` on GitHub.
- Convolutional Network (CIFAR-10). A Convolutional neural network implementation for classifying CIFAR-10 dataset, see `tutorial_cifar10.py` and ‘‘tutorial_cifar10_tfrecord.py‘‘on GitHub.
- VGG 16 (ImageNet). A Convolutional neural network implementation for classifying ImageNet dataset, see `tutorial_vgg16.py` on GitHub.
- VGG 19 (ImageNet). A Convolutional neural network implementation for classifying ImageNet dataset, see `tutorial_vgg19.py` on GitHub.
- InceptionV3 (ImageNet). A Convolutional neural network implementation for classifying ImageNet dataset, see `tutorial_inceptionV3_tfslim.py` on GitHub.
- Wild ResNet (CIFAR) by ritchieng.
- More CNN implementations of TF-Slim can be connected to TensorLayer via SlimNetsLayer.

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` on GitHub.
- Word Embedding - Word2vec. Train a word embedding matrix, see `tutorial_word2vec_basic.py` on GitHub.
- Restore Embedding matrix. Restore a pre-train embedding matrix, see `tutorial_generate_text.py` on GitHub.
- Text Generation. Generates new text scripts, using LSTM network, see `tutorial_generate_text.py` on GitHub.
- Machine Translation (WMT). Translate English to French. Apply Attention mechanism and Seq2seq to WMT English-to-French translation data, see `tutorial_translate.py` on GitHub.

1.3.4 Reinforcement Learning

- Deep Reinforcement Learning - Pong Game. Teach a machine to play Pong games, see `tutorial_atari_pong.py` on GitHub.

1.3.5 Applications

- Image Captioning - Reimplementation of Google's im2txt by zsdonghao.
- DCGAN - Generating images by Deep Convolutional Generative Adversarial Networks by zsdonghao.

1.3.6 Special Examples

- Merge TF-Slim into TensorLayer. `tutorial_inceptionV3_tfslim.py` on GitHub.
- MultiplexerLayer. `tutorial_mnist_multiplexer.py` on GitHub.
- Data augmentation with TFRecord. Effective way to load and pre-process data, see `tutorial_tfrecord*.py` and `tutorial_cifar10_tfrecord.py` on GitHub.
- Data augmentation with TensorLayer, see `tutorial_image_preprocess.py` on GitHub.

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, `import matplotlib matplotlib.use('Agg')` before import `tensorlayer as tl`. Alternatively, add the following code into the top of `visualize.py`.

```
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 we still suggest users to 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 `sess.run(tf.initialize_all_variables())`). 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)

sess.run(tf.initialize_all_variables())

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 Understand 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 DenseLayer(Layer):
    """
    The :class:`DenseLayer` class is a fully connected layer.

    Parameters
    ----------
    layer : a :class:`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
        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.
    """
    def __init__(
        self,
        layer = None,
        n_units = 100,
        act = tf.nn.relu,
        W_init = tf.truncated_normal_initializer(stddev=0.1),
        b_init = tf.constant_initializer(value=0.0),
        W_init_args = {},
        b_init_args = {},
        name ='dense_layer',
    ):
        Layer.__init__(self, name=name)
        self.inputs = layer.outputs
        if self.inputs.get_shape().ndims != 2:
            raise Exception("The input dimension must be rank 2")
        n_in = int(self.inputs._shape[-1])
        self.n_units = n_units
        print("  tensorlayer:Instantiate DenseLayer %s: %d, %s" % (self.name, self.n_
↪units, act))
        with tf.variable_scope(name) as vs:
            W = tf.get_variable(name='W', shape=(n_in, n_units), initializer=W_init,␣
↪**W_init_args )
```

(continues on next page)

(continued from previous page)

```
            if b_init:
                b = tf.get_variable(name='b', shape=(n_units), initializer=b_init,␣
↪**b_init_args )
                self.outputs = act(tf.matmul(self.inputs, W) + b)#, name=name)
            else:
                self.outputs = act(tf.matmul(self.inputs, W))

        # Hint : list(), dict() is pass by value (shallow).
        self.all_layers = list(layer.all_layers)
        self.all_params = list(layer.all_params)
        self.all_drop = dict(layer.all_drop)
        self.all_layers.extend( [self.outputs] )
        if b_init:
           self.all_params.extend( [W, b] )
        else:
           self.all_params.extend( [W] )
```

2.1.3 Your 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',
    ):
        Layer.__init__(self, name=name)
        self.inputs = layer.outputs
        self.outputs = self.inputs * 2

        self.all_layers = list(layer.all_layers)
        self.all_params = list(layer.all_params)
        self.all_drop = dict(layer.all_drop)
        self.all_layers.extend( [self.outputs] )
```

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
```

(continues on next page)

(continued from previous page)

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

2.1.4 Layer list

get_variables_with_name(name[, train_only, ...])	Get variable list 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 initialize_all_variables()
Layer([inputs, name])	The *Layer* class represents a single layer of a neural network.
InputLayer([inputs, n_features, name])	The *InputLayer* class is the starting layer of a neural network.
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.
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, name])	The *DropoutLayer* class is a noise layer which randomly set some values to zero by a given keeping probability.
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, strides, ...])	The *Conv1dLayer* class is a 1D CNN layer, see tf.nn.conv1d.
Conv2dLayer([layer, act, shape, strides, ...])	The *Conv2dLayer* class is a 2D CNN layer, see tf.nn.conv2d.

Continued on next page

Table 1 – continued from previous page

DeConv2dLayer([layer, act, shape, ...])	The *DeConv2dLayer* class is deconvolutional 2D layer, see tf.nn.conv2d_transpose.
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.
UpSampling2dLayer([layer, size, is_scale, ...])	The *UpSampling2dLayer* class is upSampling 2d layer, see tf.nn.conv3d_transpose.
AtrousConv2dLayer([layer, n_filter, ...])	The *AtrousConv2dLayer* class is Atrous convolution (a.k.a.
LocalResponseNormLayer([layer, ...])	The *LocalResponseNormLayer* class is for Local Response Normalization, see `tf.nn.local_response_normalization`.
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.
MaxPool2d(net[, filter_size, strides, ...])	Wrapper for *PoolLayer*.
MeanPool2d(net[, filter_size, strides, ...])	Wrapper for *PoolLayer*.
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`.
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.
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`.
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.
LambdaLayer([layer, fn, fn_args, name])	The *LambdaLayer* class is a layer which is able to use the provided function.

Continued on next page

Table 1 – continued from previous page

ConcatLayer([layer, concat_dim, name])	The *ConcatLayer* class is layer which concat (merge) two or more *DenseLayer* to a single class:*DenseLayer*.
ElementwiseLayer([layer, combine_fn, name])	The *ElementwiseLayer* class combines multiple *Layer* which have the same output shapes by a given elemwise-wise operation.
SlimNetsLayer([layer, slim_layer, ...])	The *SlimNetsLayer* class can be used to merge all TF-Slim nets 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.
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)	Return the initialized RNN state.
list_remove_repeat([l])	Remove the repeated items in a list, and return the processed list.

2.1.5 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 = get_variable_with_name('dense', True, 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.]

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.nn.rnn_cell.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 initialize_all_variables()

Parameters

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

2.1.6 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])	Print all info of parameters in the network

2.1.7 Input layer

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

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

Parameters

inputs [a TensorFlow placeholder] The input tensor data.

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

n_features [a int] The number of features. If not specify, it will assume the input is with the shape of [batch_size, n_features], then select the second element as the n_features. It is used to specify the matrix size of next layer. If apply Convolutional layer after InputLayer, n_features is not important.

Methods

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

Methods

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

(continues on next page)

(continued from previous page)

```
...                 name ='embedding_layer')
>>> sess.run(tf.initialize_all_variables())
>>> 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)
```

Methods

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

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])	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])	Print all info of parameters in the network

2.1.11 Convolutional layer (Pro)

1D Convolutional layer

class tensorlayer.layers.**Conv1dLayer**(*layer=None, act=<function identity>, shape=[5, 5, 1], strides=[1, 1, 1], padding='SAME', use_cudnn_on_gpu=None, data_format=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='cnn_layer'*)

The *Conv1dLayer* class is a 1D CNN layer, see tf.nn.conv1d.

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].

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.

use_cudnn_on_gpu [An optional bool. Defaults to True.]

data_format [An optional string from "NHWC", "NCHW". Defaults to "NHWC", the data is stored in the order of [batch, in_width, in_channels]. The "NCHW" format stores data as [batch, in_channels, in_width].]

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])	Print all info of parameters in the network

2D Convolutional layer

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={}, 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().

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,
...                     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])	Print all info of parameters in the network

2D Deconvolutional layer

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, is_train=is_train, name='g/h0/batch_
↪norm')
```

(continues on next page)

(continued from previous page)

```
>>> net_h0.outputs = tf.nn.relu(net_h0.outputs, name='g/h0/relu')
>>> 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, is_train=is_train, name='g/h1/batch_
↪norm')
>>> net_h1.outputs = tf.nn.relu(net_h1.outputs, name='g/h1/relu')
>>> 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])	Print all info of parameters in the network

3D Convolutional layer

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])	Print all info of parameters in the network

3D Deconvolutional layer

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

Continued on next page

Table 13 – continued from previous page

print_params([details])	Print all info of parameters in the network

2D UpSampling layer

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.nn.conv3d_transpose.

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.] (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])	Print all info of parameters in the network

2D Atrous convolutional layer

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])	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.

2D Convolutional layer

`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={}*, *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 Deconvolutional layer

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 *Conv2dLayer*.]

2D Max pooling layer

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 *Conv2dLayer*.]

2D Mean pooling layer

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 *Conv2dLayer*.]

2.1.13 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])	Print all info of parameters in the network

2.1.14 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=0.999*, *epsilon=1e-05*, *act=<function identity>*, *is_train=None*, *beta_init=<class 'tensorflow.python.ops.init_ops.Zeros'>*, *gamma_init=<class 'tensorflow.python.ops.init_ops.Ones'>*, *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] A decay factor for ExponentialMovingAverage.

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

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

2.1.15 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=<class 'tensorflow.python.ops.rnn_cell_impl.BasicRNNCell'>*, *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.]

- see RNN Cells in TensorFlow
- class `tf.nn.rnn_cell.BasicRNNCell`
- class `tf.nn.rnn_cell.BasicLSTMCell`
- class `tf.nn.rnn_cell.GRUCell`

- class `tf.nn.rnn_cell.LSTMCell`

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

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

initializer [initializer] The initializer for initializing the parameters.

n_steps [a 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])
>>> network = 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')
>>> if is_training:
```

(continues on next page)

(continued from previous page)

```
>>>     network = tl.layers.DropoutLayer(network, keep=keep_prob, name='drop1')
>>> network = tl.layers.RNNLayer(network,
...             cell_fn=tf.nn.rnn_cell.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 = network
>>> if is_training:
>>>     network = tl.layers.DropoutLayer(network, keep=keep_prob, name='drop2')
>>> network = tl.layers.RNNLayer(network,
...             cell_fn=tf.nn.rnn_cell.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 = 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 = tl.activation.identity, name='output_layer')
```

- For CNN+LSTM

```
>>> x = tf.placeholder(tf.float32, shape=[batch_size, image_size, image_size, 1])
>>> 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, 2, 2, 1],
...                         padding='SAME',
...                         name ='cnn_layer1')
>>> 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')
>>> network = tl.layers.Conv2dLayer(network,
...                         act = tf.nn.relu,
...                         shape = [5, 5, 32, 10], # 10 features for each 5x5␣
↪patch
...                         strides=[1, 2, 2, 1],
...                         padding='SAME',
```

(continues on next page)

(continued from previous page)

```
...                           name ='cnn_layer2')
>>> 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')
>>> network = tl.layers.FlattenLayer(network, name='flatten_layer')
>>> network = tl.layers.ReshapeLayer(network, shape=[-1, num_steps, int(network.
↪outputs._shape[-1])])
>>> rnn1 = tl.layers.RNNLayer(network,
...                           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')
>>> network = tl.layers.DenseLayer(rnn1, n_units=3,
...                           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])	Print all info of parameters in the network

Bidirectional layer

class tensorlayer.layers.**BiRNNLayer**(*layer=None*, *cell_fn=<class 'tensorflow.python.ops.rnn_cell_impl.LSTMCell'>*, *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.]

- see RNN Cells in TensorFlow
- class tf.nn.rnn_cell.BasicRNNCell
- class tf.nn.rnn_cell.BasicLSTMCell
- class tf.nn.rnn_cell.GRUCell
- class tf.nn.rnn_cell.LSTMCell

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

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

initializer [initializer] The initializer for initializing the parameters.

n_steps [a 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 [a 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

Methods

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

2.1.16 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()
>>> sess.run(tf.initialize_all_variables())
>>>
>>> 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()
>>> sess.run(tf.initialize_all_variables())
>>> 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]]]
>>> 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()
>>> sess.run(tf.initialize_all_variables())
>>> print(o.eval())
... [2 3 4]
```

2.1.17 Dynamic RNN layer

class tensorlayer.layers.**DynamicRNNLayer**(*layer=None*, *cell_fn=<class 'tensorflow.python.ops.rnn_cell_impl.LSTMCell'>*, *cell_init_args={'state_is_tuple': True}*, *n_hidden=64*, *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*, *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.]

- see RNN Cells in TensorFlow
- class `tf.nn.rnn_cell.BasicRNNCell`
- class `tf.nn.rnn_cell.BasicLSTMCell`
- class `tf.nn.rnn_cell.GRUCell`
- class `tf.nn.rnn_cell.LSTMCell`

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

n_hidden [a 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 [a 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")
>>> network = tl.layers.EmbeddingInputlayer(
...             inputs = input_seqs,
...             vocabulary_size = vocab_size,
...             embedding_size = embedding_size,
...             name = 'seq_embedding')
>>> network = tl.layers.DynamicRNNLayer(network,
...             cell_fn = 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',)
... network = tl.layers.DenseLayer(network, 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])	Print all info of parameters in the network

2.1.18 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])
>>> network = tl.layers.InputLayer(x, name='input_layer')
>>> network = tl.layers.Conv2dLayer(network,
...                     act = tf.nn.relu,
...                     shape = [5, 5, 32, 64],
...                     strides=[1, 1, 1, 1],
...                     padding='SAME',
...                     name ='cnn_layer')
>>> network = tl.layers.Pool2dLayer(network,
...                     ksize=[1, 2, 2, 1],
...                     strides=[1, 2, 2, 1],
...                     padding='SAME',
...                     pool = tf.nn.max_pool,
...                     name ='pool_layer',)
>>> network = tl.layers.FlattenLayer(network, 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])	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
print_params([details])	Print all info of parameters in the network

2.1.19 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')
>>> network = tl.layers.InputLayer(x, name='input_layer')
>>> network = LambdaLayer(network, lambda x: 2*x, name='lambda_layer')
>>> y = network.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
print_params([details])	Print all info of parameters in the network

2.1.20 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 *DenseLayer* to a single class:*DenseLayer*.

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, n_units=800, act = tf.nn.relu, name=
↪'relu1_1')
>>> net2 = tl.layers.DenseLayer(inputs, n_units=300, act = tf.nn.relu, name=
↪'relu2_1')
>>> network = tl.layers.ConcatLayer(layer = [net1, net2], name ='concat_layer')
...     tensorlayer:Instantiate InputLayer input_layer (?, 784)
...     tensorlayer:Instantiate DenseLayer relu1_1: 800, <function relu at␣
↪0x1108e41e0>
...     tensorlayer:Instantiate DenseLayer relu2_1: 300, <function relu at␣
↪0x1108e41e0>
...     tensorlayer:Instantiate ConcatLayer concat_layer, 1100
...
>>> sess.run(tf.initialize_all_variables())
>>> network.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
>>> network.print_layers()
...     layer 0: Tensor("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

Continued on next page

Table 25 – continued from previous page

print_layers()	Print all info of layers in the network
print_params([details])	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.mul` 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])	Print all info of parameters in the network

2.1.21 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='InceptionV3'*)

The *SlimNetsLayer* class can be used to merge all TF-Slim nets into TensorLayer. Model can be found in slim-model , more about slim see slim-git .

Parameters

layer [a list of *Layer* instances] 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.

Examples

- see Inception V3 example on Github

Methods

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

2.1.22 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])	Print all info of parameters in the network

2.1.23 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()` 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])	Print all info of parameters in the network

2.1.24 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.

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])	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.

2.1.25 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*)

Return the initialized RNN state. The input is LSTMStateTuple or State of RNNCells.

Parameters

state [a RNN 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]
```

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

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()`.

```
sess.run(tf.initialize_all_variables())
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])	Returns the TensorFlow expression of cross-entropy of two distributions, implement softmax internally.
binary_cross_entropy(output, target[, ...])	Computes binary cross entropy given *output*.
mean_squared_error(output, target)	Return the TensorFlow expression of mean-squre-error of two distributions.

Continued on next page

Table 31 – continued from previous page

dice_coe(output, target[, epsilon])	Sørensen–Dice coefficient for comparing the similarity of two distributions, usually be used for binary image segmentation i.e.
dice_hard_coe(output, target[, epsilon])	Non-differentiable Sørensen–Dice coefficient for comparing the similarity of two distributions, usually be used for binary image segmentation i.e.
iou_coe(output, target[, threshold, epsilon])	Non-differentiable Intersection over Union, 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)	li regularization removes the neurons of previous layer, *i* represents *inputs*.
lo_regularizer(scale)	lo regularization removes the neurons of current layer, *o* represents *outputs*
maxnorm_regularizer([scale])	Max-norm regularization returns a function that can be used to apply max-norm regularization to weights.
maxnorm_o_regularizer(scale)	Max-norm output regularization removes the neurons of current layer.
maxnorm_i_regularizer(scale)	Max-norm input regularization removes the neurons of previous layer.

2.2.2 Cross entropy

tensorlayer.cost.**cross_entropy**(*output*, *target*, *name='cross_entropy_loss'*)

Returns the TensorFlow expression of cross-entropy of two distributions, implement softmax internally.

Parameters

output [Tensorflow variable] A distribution with shape: [batch_size, n_feature].

target [Tensorflow variable] A batch of index with shape: [batch_size,].

References

- About cross-entropy: wiki.
- The code is borrowed from: here.

Examples

```
>>> ce = tl.cost.cross_entropy(y_logits, y_target_logits)
```

2.2.3 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

2.2.4 Mean squared error

`tensorlayer.cost.`**`mean_squared_error`**(*output*, *target*)

Return the TensorFlow expression of mean-squre-error of two distributions.

Parameters

output [tensorflow variable] A distribution with shape: [batch_size, n_feature].

target [tensorflow variable] A distribution with shape: [batch_size, n_feature].

2.2.5 Dice coefficient

`tensorlayer.cost.`**`dice_coe`**(*output*, *target*, *epsilon=1e-10*)

Sørensen–Dice coefficient for comparing the similarity of two distributions, usually be used for binary image segmentation i.e. labels are binary. The coefficient = [0, 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).

epsilon [float] An optional name to attach to this layer.

References

- wiki-dice

Examples

```
>>> outputs = tl.act.pixel_wise_softmax(network.outputs)
>>> dice_loss = 1 - tl.cost.dice_coe(outputs, y_, epsilon=1e-5)
```

2.2.6 Hard Dice coefficient

tensorlayer.cost.**dice_hard_coe**(*output*, *target*, *epsilon=1e-10*)

Non-differentiable Sørensen–Dice coefficient for comparing the similarity of two distributions, usually be used for binary image segmentation i.e. labels are binary. The coefficient = [0, 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).

epsilon [float] An optional name to attach to this layer.

References

- wiki-dice

Examples

```
>>> outputs = pixel_wise_softmax(network.outputs)
>>> dice_loss = 1 - dice_coe(outputs, y_, epsilon=1e-5)
```

2.2.7 IOU coefficient

tensorlayer.cost.**iou_coe**(*output*, *target*, *threshold=0.5*, *epsilon=1e-10*)

Non-differentiable Intersection over Union, usually be used for evaluating binary image segmentation. The coefficient = [0, 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.

epsilon [float] A small value to avoid zero denominator when both output and target output nothing.

Notes

- IOU cannot be used as training loss, people usually use dice coefficient for training, and IOU for evaluating.

Examples

```
>>> outputs = tl.act.pixel_wise_softmax(network.outputs)
>>> iou = tl.cost.iou_coe(outputs[:,:,:,0], y_[:,:,:,0])
```

2.2.8 Cross entropy for sequence

tensorlayer.cost.**cross_entropy_seq**(*logits*, *target_seqs*, *batch_size=1*, *num_steps=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 [a int, default is 1] RNN batch_size, number of concurrent processes, divide the loss by batch_size.

num_steps [a int] sequence length

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 = tf.cost.cross_entropy_seq(network.outputs, targets, batch_size, num_
↪steps)
```

2.2.9 Cross entropy with mask for sequence

tensorlayer.cost.**cross_entropy_seq_with_mask**(*logits*, *target_seqs*, *input_mask*, *return_details=False*)

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.

2.2.10 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,]

2.2.11 Regularization functions

For `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*)
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.

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*)
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.

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*)
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.

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*)
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.

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*)
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.

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])	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.
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.
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.
imresize(x[, size, interp, mode])	Resize an image by given output size and method.
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 .
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).

Continued on next page

Table 32 – continued from previous page

pad_sequences(sequences[, maxlen, dtype, ...])	Pads each sequence to the same length: the length of the longest sequence.
distorted_images([images, height, width])	Distort images for generating more training data.
crop_central_whiten_images([images, height, ...])	Crop the central of image, and normailize it for test data.

2.3.1 Threading

tensorlayer.prepro.**threading_data**(*data=None*, *fn=None*, ***kwargs*)

Return a batch of result by given data. Usually be used for data augmentation.

Parameters

data [numpy array or zip of numpy array, see Examples below.]

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)
```

- 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*)

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

- 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*)

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 [float] Size of weight.

hrg [float] 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*, *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 zoom.

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*)

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.

- 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*)

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*)

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.

- scipy ndimage affine_transform

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*)

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 `shear`.]

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*)

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, (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', 're-flect' 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.

- 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*)

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.

- 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`.]

Resize

`tensorlayer.prepro.`**`imresize`**(*x, size=[100, 100], interp='bilinear', 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

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*, *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`.]

Manual transform

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`.

tensorlayer.prepro.**apply_transform**(*x*, *transform_matrix*, *channel_index=2*, *fill_mode='nearest'*, *cval=0.0*)
Return transformed images by given transform_matrix from `transform_matrix_offset_center`.

Parameters

x [numpy array] Batch of images with dimension of 3, [batch_size, row, col, channel].

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

- scipy ndimage affine_transform

Examples

- See `rotation`, `shift`, `shear`, `zoom`.

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 (x, y).

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]]
>>> 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

2.3.3 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]]
```

2.3.4 Tensor Opt

Note: These functions will be deprecated, see `tutorial_cifar10_tfrecord.py` for new information.

`tensorlayer.prepro.distorted_images`(*images=None*, *height=24*, *width=24*)

Distort images for generating more training data.

Parameters

images [4D Tensor] The tensor or placeholder of images

height [int] The height for random crop.

width [int] The width for random crop.

Returns

result [tuple of Tensor] (Tensor for distorted images, Tensor for while loop index)

Notes

- The first image in 'distorted_images' should be removed.

References

- tensorflow.models.image.cifar10.cifar10_input

Examples

```
>>> X_train, y_train, X_test, y_test = tl.files.load_cifar10_dataset(shape=(-1,␣
↪32, 32, 3), plotable=False)
>>> sess = tf.InteractiveSession()
>>> batch_size = 128
>>> x = tf.placeholder(tf.float32, shape=[batch_size, 32, 32, 3])
>>> distorted_images_op = tl.preprocess.distorted_images(images=x, height=24,␣
↪width=24)
>>> sess.run(tf.initialize_all_variables())
>>> feed_dict={x: X_train[0:batch_size,:,:,:]}
>>> distorted_images, idx = sess.run(distorted_images_op, feed_dict=feed_dict)
>>> tl.visualize.images2d(X_train[0:9,:,:,:], second=2, saveable=False, name=
↪'cifar10', dtype=np.uint8, fig_idx=20212)
>>> tl.visualize.images2d(distorted_images[1:10,:,:,:], second=10, saveable=False,
↪ name='distorted_images', dtype=None, fig_idx=23012)
```

tensorlayer.prepro.**crop_central_whiten_images**(*images=None*, *height=24*, *width=24*)

Crop the central of image, and normailize it for test data.

They are cropped to central of height * width pixels.

Whiten (Normalize) the images.

Parameters

images [4D Tensor] The tensor or placeholder of images

height [int] The height for central crop.

width [int] The width for central crop.

Returns

result [tuple Tensor] (Tensor for distorted images, Tensor for while loop index)

Notes

The first image in 'central_images' should be removed.

Examples

```
>>> X_train, y_train, X_test, y_test = tl.files.load_cifar10_dataset(shape=(-1,␣
↪32, 32, 3), plotable=False)
>>> sess = tf.InteractiveSession()
>>> batch_size = 128
>>> x = tf.placeholder(tf.float32, shape=[batch_size, 32, 32, 3])
>>> central_images_op = tl.preprocess.crop_central_whiten_images(images=x,␣
↪height=24, width=24)
>>> sess.run(tf.initialize_all_variables())
>>> feed_dict={x: X_train[0:batch_size,:,:,:]}
>>> central_images, idx = sess.run(central_images_op, feed_dict=feed_dict)
>>> tl.visualize.images2d(X_train[0:9,:,:,:], second=2, saveable=False, name=
↪'cifar10', dtype=np.uint8, fig_idx=20212)
>>> tl.visualize.images2d(central_images[1:10,:,:,:], second=10, saveable=False,␣
↪name='central_images', dtype=None, fig_idx=23012)
```

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.

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):
```

(continues on next page)

(continued from previous page)

```
>>>     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)	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.
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.
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*)

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

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)
```

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*)

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

y_op [placeholder] the argmax expression of softmax outputs

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

```
>>> X_train, y_train = class_balancing_oversample(X_train, y_train,␣
↪printable=True)
```

2.5.4 Helper functions

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)
```

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.

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>'*)

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.]

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.

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.

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)
...   tensorlayer.nlp: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>")
...   tensorlayer.nlp:Instantiate Vocabulary from vocab.txt : <S> </S> <UNK>
...   vocabulary with 9 words (includes unk_word)
```

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 ‘‘’

‘‘, e.g. `[... 'how', 'useful', 'it', "'s" ... ]`.

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 .... }

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.

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.

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`

2.7 API - Reinforcement Learning

Reinforcement Learning.

discount_episode_rewards([rewards, gamma])	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.

2.7.1 Reward functions

`tensorlayer.rein.`**`discount_episode_rewards`**(*rewards=[]*, *gamma=0.99*)

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

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.        ]
```

2.7.2 Cost functions

tensorlayer.rein.**cross_entropy_reward_loss**(*logits*, *actions*, *rewards*)

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])   # observation␣
↪for training
>>> 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 = tl.activation.
↪identity, name='output_layer')
>>> 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 = cross_entropy_reward_loss(probs, actions_batch_pl, discount_rewards_
↪batch_pl)
>>> train_op = tf.train.RMSPropOptimizer(learning_rate, decay_rate).minimize(loss)
```

2.8 API - Load, Save Model and Data

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")
```

(continues on next page)

(continued from previous page)

```
# 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
load_params = tl.files.load_npz(path='', name='model.npz')
tl.files.assign_params(sess, load_params, 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])	Automatically download MNIST dataset and return the training, validation and test set with 50000, 10000 and 10000 digit images respectively.
load_cifar10_dataset([shape, plotable, second])	The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class.
load_ptb_dataset()	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()	Download a text file from Matt Mahoney's website if not present, and make sure it's the right size.
load_imbd_dataset([path, nb_words, ...])	Load IMDB dataset
load_nietzsche_dataset()	Load Nietzsche dataset.
load_wmt_en_fr_dataset([data_dir])	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.
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.
save_any_to_npy([save_dict, name])	Save variables to .npy file.
load_npy_to_any([path, name])	Load .npy file.
npz_to_W_pdf([path, regx])	Convert the first weight matrix of .npz file to .pdf by using tl.visualize.W().
load_file_list([path, regx, printable])	Return a file list in a folder by given a path and regular expression.

2.8.1 Load dataset functions

MNIST

tensorlayer.files.**load_mnist_dataset**(*shape=(-1, 784)*)

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

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)*, *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) , (-1, 32, 32, 3) , (-1, 32*32*3)

plotable [True, False] Whether to plot some image examples.

second [int] If `plotable` is True, `second` is the display time.

Notes

CIFAR-10 images can only be display without color change under uint8. >>> X_train = np.asarray(X_train, dtype=np.uint8) >>> plt.ion() >>> fig = plt.figure(1232) >>> count = 1 >>> for row in range(10): >>> for col in range(10): >>> a = fig.add_subplot(10, 10, count) >>> plt.imshow(X_train[count-1], interpolation='nearest') >>> plt.gca().xaxis.set_major_locator(plt.NullLocator()) # (tick) >>> plt.gca().yaxis.set_major_locator(plt.NullLocator()) >>> count = count + 1 >>> plt.draw() >>> plt.pause(3)

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), plotable=True)
```

Penn TreeBank (PTB)

tensorlayer.files.**load_ptb_dataset**()

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.

In "Recurrent Neural Network Regularization", they trained regularized LSTMs of two sizes; these are denoted the medium LSTM and large LSTM. Both LSTMs have two layers and are unrolled for 35 steps. They initialize the hidden states to zero. They then use the final hidden states of the current minibatch as the initial hidden state of the subsequent minibatch (successive minibatches sequentially traverse the training set). The size of each minibatch is 20.

The medium LSTM has 650 units per layer and its parameters are initialized uniformly in [0.05, 0.05]. They apply 50% dropout on the non-recurrent connections. They train the LSTM for 39 epochs with a learning rate of 1, and after 6 epochs they decrease it by a factor of 1.2 after each epoch. They clip the norm of the gradients (normalized by minibatch size) at 5.

The large LSTM has 1500 units per layer and its parameters are initialized uniformly in [0.04, 0.04]. We apply 65% dropout on the non-recurrent connections. They train the model for 55 epochs with a learning rate of 1; after 14 epochs they start to reduce the learning rate by a factor of 1.15 after each epoch. They clip the norm of the gradients (normalized by minibatch size) at 10.

Returns

train_data, valid_data, test_data, vocabulary size

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**()

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.

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_imbd_dataset**(*path='imdb.pkl'*, *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

References

- Modify from keras.

Examples

```
>>> X_train, y_train, X_test, y_test = tl.files.load_imbd_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**()

Load Nietzsche dataset. Returns a string.

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**(*data_dir='wmt'*)

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

data_dir [a string] The directory to store the dataset.

Notes

Usually, it will take a long time to download this dataset.

References

- Code modified from /tensorflow/models/rnn/translation/data_utils.py

2.8.2 Load and save network

Save network as .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

```
>>> tl.files.save_npz(network.all_params, name='model_test.npz', sess=sess)
... File saved to: model_test.npz
>>> load_params = tl.files.load_npz(name='model_test.npz')
... Loading param0, (784, 800)
... Loading param1, (800,)
... Loading param2, (800, 800)
... Loading param3, (800,)
... Loading param4, (800, 10)
... Loading param5, (10,)
>>> put parameters into a TensorLayer network, please see assign_params()
```

Load network from .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 and assign_params

Assign parameters to network

`tensorlayer.files.`**`assign_params`**(*sess*, *params*, *network*)

Assign the given parameters to the TensorLayer network.

Parameters

sess [TensorFlow Session]

params [a list] A list of parameters in order.

network [a `Layer` class] The network to be assigned

References

- Assign value to a TensorFlow variable

Examples

```
>>> Save your network as follow:
>>> tl.files.save_npz(network.all_params, name='model_test.npz')
>>> network.print_params()
...
... Next time, load and assign your network as follow:
>>> sess.run(tf.initialize_all_variables()) # re-initialize, then save and assign
>>> load_params = tl.files.load_npz(name='model_test.npz')
>>> tl.files.assign_params(sess, load_params, network)
>>> network.print_params()
```

2.8.3 Load and save variables

Save variables as .npy

`tensorlayer.files.`**`save_any_to_npy`**(*save_dict={}*, *name='any.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='any.npy'*)

Load .npy file.

Examples

- see save_any_to_npy()

2.8.4 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)')
```

2.8.5 Helper functions

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)')
```

2.9 API - Visualize Model and Data

TensorFlow provides TensorBoard to visualize the model, activations etc. Here we provide more functions for data visualization.

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 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)
```

2.9.2 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)
```

2.9.3 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])	Close tensorboard and nvidia-process if available
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.

Continued on next page

Table 41 – continued from previous page

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

Kill nvidia process

`tensorlayer.ops.`**`exit_tf`**(*sess=None*)

Close tensorboard and nvidia-process if available

Parameters

sess [a session instance of TensorFlow] TensorFlow session

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`.
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='LeakyReLU'*)

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 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)
```

2.11.6 Parametric activation

See `tensorlayer.layers`.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

t

Index

G

I

L

M

N

P

R

S

T

U

V

W

Z