pytorch-nlp-tutorial-sf2017 Documentation

Release

Brian McMahan and Delip Rao

Sep 18, 2017

Extra Resources

CHAPTER 1

Getting the Data

In this training, there are two options of participating.

Option 1: Download and Setup things on your laptop

The first option is to download the data below, setup the environment, and download the notebooks when we make them available. If you choose this options but do not download the data before the first day, we will have several flash drives with the data on it.

Please visit this link to download the data.

Option 2: Use O'Reilly's online resource through your browser

The second option is to use an online resource provided by O'Reilly. On the first day of this training, you will be provided with a link to a JupyterHub instance where the environment will be pre-made and ready to go! If you choose this option, you do not have to do anything until you arrive on Sunday. You are still required to bring your laptop.

CHAPTER 2

Environment Setup

On this page, you will find not only the list of dependencies to install for the tutorial, but a description of how to install them. This tutorial assumes you have a laptop with OSX or Linux. If you use Windows, you might have to install a virtual machine to get a UNIX-like environment to continue with the rest of this instruction. A lot of this instruction is more verbose than needed to accomodate participants of different skill levels.

Please note that these are only optional. On the first day of this training, you will be provided with a link to a JupyterHub instance where the environment will be pre-made and ready to go!

0. Get Anaconda

Anaconda is a Python (and R) distribution that aims to provide everything needed for common scientific and machine learning situations out-of-the-box. We chose Anaconda for this tutorial as it significantly simplifies Python dependency management.

In practice, Anaconda can be used to manage different environment and packages. This setup document will assume that you have Anaconda installed as your default Python distribution.

You can download Anaconda here: https://www.continuum.io/downloads

After installing Anaconda, you can access its command-line interface with the `conda` command.

1. Create a new environment

Environments are a tool for sanitary software development. By this, we mean that you can install specific versions of packages without worrying that it breaks a dependency elsewhere.

Here is how you can create an environment with Anaconda

```
conda create -n dl4nlp python=3.6
```

2. Install Dependencies

2a. Activate the environment

After creating the environment, you need to **activate** the environment:

```
source activate dl4nlp
```

After an environment is activated, it might prepend/append itself to your console prompt to let you know it is active.

With the environment activated, any installation commands (whether it is `pip install X`, `python setup.py install` or using Anaconda's install command `conda install X`) will only install inside the environment.

2b. Install IPython and Jupyter

Two core dependencies are IPython and Jupyter. Let's install them first:

```
conda install ipython
conda install jupyter
```

To allow a jupyter notebooks to use this environment as their kernel, it needs to be linked:

```
python -m ipykernel install --user --name dl4nlp
```

2c. Installing CUDA (optional)

NOTE: CUDA is currently not supported out of the conda package control manager. Please refer to pytorch's github repository for compilation instructions.

If you have a CUDA compatible GPU, it is worthwhile to take advantage of it as it can significantly speedup training and make your PyTorch experimentation more enjoyable.

To install CUDA:

1. Download CUDA appropriate to your OS/Arch from here.
2. Follow installation steps for your architecture/OS. For Ubuntu/x86_64, see here.
3. Download and install CUDNN from here.

Make sure you have the latest CUDA (8.0) and CUDNN (7.0).

2d. Install PyTorch

There are instructions on http://pytorch.org which detail how to install it. If you have been following along so far and have Anaconda installed with CUDA enabled, you can simply do:

```
conda install pytorch torchvision cuda80 -c soumith
```

The widget on PyTorch.org will let you select the right command line for your specific OS/Arch. Make sure you have PyTorch 2.0 or higher.

2e. Clone (or Download) Repository

At this point, you may have already cloned the tutorial repository. But if you have not, you will need it for the next step.

```
git clone https://github.com/joosthub/pytorch-nlp-tutorial-sf2017.git
```

If you do not have git or do not want to use it, you can also download the repository as a zip file

2f. Install Dependencies from Repository

Assuming the you have cloned (or downloaded and unzipped) the repository, please navigate to the directory in your terminal. Then, you can do the following:

```
pip install -r requirements.txt
```

CHAPTER 3

Frequency Asked Questions

On this page, you will find a list of questions that we either anticipate people will ask or that we have been asked previously. They are intended to be the first stop for any confusion or trouble that might occur.

Do I Need to have a NVIDIA GPU enabled laptop?

Nope! While having a NVIDIA GPU enabled laptop will make the training run faster, we provide instructions for people who do not have one.

If you are plan on working on Natural Language Processing/Deep Learning in the future, a GPU enabled laptop might be a good investment.

CHAPTER 4

Recipes and PyTorch patterns

In this section, you will find a set of recipes for doing various things with PyTorch.

Loading Pretrained Vectors

It can be extremely useful to make a model which had as advantageous starting point.

To do this, we can set the values of the embedding matrix.

```
# we give an example of this function in the day 1, word vector notebook
word_to_index, word_vectors, word_vector_size = load_word_vectors()


# now, we want to iterate over our vocabulary items
for word, emb_index in vectorizer.word_vocab.items():
    # if the word is in the loaded glove vectors
    if word.lower() in word_to_index:
         # get the index into the glove vectors
         glove_index = word_to_index[word.lower()]
         # get the glove vector itself and convert to pytorch structure
         glove_vec = torch.FloatTensor(word_vectors[glove_index])

         # this only matters if using cuda :)
         if settings.CUDA:
             glove_vec = glove_vec.cuda()

         # finally, if net is our network, and emb is the embedding layer:
         net.emb.weight.data[emb_index, :].set_(glove_vec)
```

Compute Convolution Sizes

```
import math

def conv_shape_helper_1d(input_seq_len, kernel_size, stride=1, padding=0, dilation=1):
    kernel_width = dilation * (kernel_size - 1) + 1
    tensor_size = input_seq_len + 2 * padding
    return math.floor((tensor_size - kernel_width) / stride + 1)
```

CHAPTER 5

Take-Home Exercises

Exercise 1

Implement Deep Continuous Bag-of-Words (CBOW). Here is a link to the paper!

Exercise 2

Complete ConvNet example to do evaluation (in the same manner as the MLP example).

Exercise 3

Implement a convnet classifier to classify names

Things to try: with and without padding and changing strides.

CHAPTER 6

Warm Up Exercise

To get you back into the PyTorch groove, let's do some easy exercises. You will have 10 minutes. See how far you can get.

1. Use `torch.randn` to create two tensors of size (29, 30, 32) and and (32, 100).
2. Use `torch.matmul` to matrix multiply the two tensors.
3. Use `torch.sum` on the resulting tensor, passing the optional argument of `dim=1` to sum across the 1st dimension. Before you run this, can you predict the size?
4. Create a new long tensor of size (3, 10) from the `np.random.randint` method.
5. Use this new long tensor to index into the tensor from step 3.
6. Use `torch.mean` to average across the last dimension in the tensor from step 5.

CHAPTER 7

Choose Your Own Adventures

Exercise: Interpolating Between Vectors

One fun option for the conditional generation code is to interpolate between the learned hidden vectors.

To do this, first look at the code for sampling given a specific nationality:

```
def sample_n_for_nationality(nationality, n=10, temp=0.8):
    assert nationality in vectorizer.nationality_vocab.keys(), 'not a nationality we␣
↪trained on'
    keys = [nationality] * n
    init_vector = long_variable([vectorizer.nationality_vocab[key] for key in keys])
    init_vector = net.conditional_emb(init_vector)
    samples = decode_matrix(vectorizer,
                            sample(net.emb, net.rnn, net.fc,
                                   init_vector,
                                   make_initial_x(n, vectorizer),
                                   temp=temp))
    return list(zip(keys, samples))
```

As you can see, we create a list of keys that is the length of the number of samples we want (n). And we use that list to retrieve the correct index from the vocabulary. Finally, we use that index in the conditional embedding inside the network to get the initial hidden state for the sampler.

To do this exercise, write a function that has the following signature:

```
def interpolate_n_samples_from_two_nationalities(nationality1, nationality2, weight,␣
↪n=10, temp=0.8):
    print('awesome stuff here')
```

This should retrieve the `init_vectors` for two different nationalities. Then, using the weight, combine the init vectors as `weight * init_vector1 + (1 - weight) * init_vector2`.

For fun, after you finish this function, write a for loop which loops over the weight from 0.1 to 0.9 to see how it affects the generation.

Exercise: Fast Lookups for Encoded Sequences

Let's suppose that you want to embed or encode something that you want to look up at a later date. For example, you could be embedded things that need to be identified (such as a song). Or maybe you want to just find the neighbors of a new data point.

In any case, using the approximate nearest neighbors libraries are wonderful for this. For this exercise, we will use Spotify's annoy library (we saw this on day 1, in the pretrained word vector notebook). You should aim to complete the following steps:

1. **Load the network from the Day 2, 01 notebook using the pre-trained weights.**
    - You could use the 02 notebook, but we want to get a single vector per each sequence.
    - So, to use 02, you would need to port the `column_gather` function.
    - One reason why you might be interested in doing this is because the 02 objective function learned a better final vector representation.
2. **Given a loaded network with pre-trained weights, write a function which does nearly exactly what the forward function d**
    - This is because we want the feature vector just before the fully connected.
    - it is common to assume that the penultimate layer has learned more generalizable features than the final layer (which is used in softmax computations and is this used to being normalize inducing a probability distribution).
    - The code for this shoud look something like:

```
def get_penultimate(net, x_in, x_lengths=None):
    x_in = net.emb(x_in)
    x_mid = net.conv(x_in.permute(0, 2, 1)).permute(0, 2, 1)
    y_out = net.rnn(x_in)

    if x_lengths is not None:
        y_out = column_gather(y_out, x_lengths)
    else:
        y_out = y_out[:, -1, :]

    return y_out
```

3. As you get penultimate vectors for each datapoint, store them in spotify's annoy. This requires specifying some label for the vector. Using `vectorizer.surname_vocab.lookup` is how you can retrieve the character for each index value in the network inputs. There are some 'decode' functions in the day 2 02 and 03 notebooks.
4. Once everything is added to spotify's annoy, you can then look up any surname and find the set of nearest neighbors! Kind of cool! this is one way to do the k nearest neighbor classification rule.

A New Load-Vectorize-Generate

In this exercise, you should look into the two datasets that are not included in the exercises. There are two datasets to work with. The first is the Amazon Review dataset.

```
from local_settings import settings
import pandas as pd
```

```
data = pd.read_csv(settings.AMAZON_FILENAME, names=['rating', 'title', 'review'])
print(data.head())
```

The Amazon Reviews Dataset does not come with a precompute train-test split. One thing that would be important is to select a subset to do that.

The other is the first names dataset. You can load with:

```
from local_settings import settings
import pandas as pd

data = pd.read_csv(settings.FIRSTNAMES_CSV)
print(data.head())
```

For these two datasets, you should write a Raw dataset which loads the data. Then, you should write a Vectorizer which creates the relevant vocabularies from the 'fit' method and transforms a raw dataset into a vectorized dataset using the 'transform' method. Finally, you should write a Vectorized datset which implements the required `__len__` and `__getitem__` methods.

The make_generator can be reused.

CHAPTER 8

NN Patterns

Attention

Attention is a useful pattern for when you want to take a collection of vectors—whether it be a sequence of vectors representing a sequence of words, or an unordered collections of vectors representing a collection of attributes—and summarize them into a single vector. This has similar analogs to the CBOW examples we saw on Day 1, but instead of just averaging or using max pooling, we are learning a function which learns to compute the weights for each of the vectors before summing them together.

Importantly, the weights that the attention module is learning is a valid probability distribution. This means that weighting the vectors by the value the attention module learns can additionally be seen as computing the Expection. Or, it could as interpolating. In any case, attention's main use is to select 'softly' amongst a set of vectors.

The attention vector has several different published forms. The one below is very simple and just learns a single vector as the attention mechanism.

Using the `new_parameter` function we have been using for the RNN notebooks:

```
def new_parameter(*size):
    out = Parameter(FloatTensor(*size))
    torch.nn.init.xavier_normal(out)
    return out
```

We can then do:

```
class Attention(nn.Module):
    def __init__(self, attention_size):
        super(Attention, self).__init__()
        self.attention = new_parameter(attention_size, 1)

    def forward(self, x_in):
        # after this, we have (batch, dim1) with a diff weight per each cell
        attention_score = torch.matmul(x_in, self.attention).squeeze()
        attention_score = F.softmax(attention_score).view(x_in.size(0), x_in.size(1),␣
↪1)
```

```
        scored_x = x_in * attention_score

        # now, sum across dim 1 to get the expected feature vector
        condensed_x = torch.sum(scored_x, dim=1)

        return condensed_x



attn = Attention(100)
x = Variable(torch.randn(16,30,100))
attn(x).size() == (16,100)
```

Hello! This is a directory of resources for a training tutorial to be given at the O'Reilly AI Conference in San Francisco on September 17 and 18, 2017.

Please read below for general information. You can find the github repository at this link. Please note that there are two ways to engage in this training (desribed below).

More information will be added to this site as the training progresses. Specifically, we will be adding a 'recipes' section, 'errata' section, and a 'bonus exercise' section as the training progresses!

CHAPTER 9

General Information

Prerequisites:

- A working knowledge of Python and the command line
- Familiarity with precalc math (multiply matrices, dot products of vectors, etc.) and derivatives of simple functions (If you are new to linear algebra, this video course is handy.)
- A general understanding of machine learning (setting up experiments, evaluation, etc.) (useful but not required)

Hardware and/or installation requirements:

- **There are two options:**
    1. **Using O'Reilly's online resources**. For this, you only needs a laptop; on the first day, we will provide you with credentials and a URL to use an online computing resource (a JupyterHub instance) provided by O'Reilly. You will be able to access Jupyter notebooks through this and they will persist until the end of the second day of training (September 18th). This option is not limited by what operating system you have. You will need to have a browser installed.
    2. **Setting everything up locally**. For this, you need a laptop with the PyTorch environment set up. This is only recommended if you want to have the environment locally or have a laptop with a GPU. (If you have trouble following the provided instructions or if you find any mistakes, please file an issue here.) This option is limited to Macs and Linux users only (sorry Windows users!). Be sure you check the LOCAL_RUN_README.md.