Project: TensorFlow

Crisan Adrian

30.05.2018

Chapter 1

Final Report

1.1 Introduction

TensorFlow is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. The flexible architecture allows you to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API. TensorFlow was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well.

1.2 Instalation

TensorFlow can be installed on Ubuntu, macOS or Windows. I will present the installation steps for the Windows OS. First of all, you must decide which version of TensorFlow you will install, the CPU version or the GPU version, used for performance-critical applications, because it uses the NVIDIA GPU of the system. If you are installing TensorFlow with GPU support using one of the mechanisms described in this guide, then the following NVIDIA software must be installed on your system:

- CUDA Toolkit 9.0. For details, see NVIDIA's documentation Ensure that you append the relevant Cuda pathnames to the PATH environment variable as described in the NVIDIA documentation.
- The NVIDIA drivers associated with CUDA Toolkit 9.0
- cuDNN v7.0. For details, see NVIDIA's documentation. Note that cuDNN is typically installed in a different location from the other CUDA DLLs. Ensure that you add the directory where you installed the cuDNN DLL to your %PATH% environment variable.
- GPU card with CUDA Compute Capability 3.0 or higher for building from source and 3.5 or higher for our binaries. See NVIDIA documentation for

a list of supported GPU cards.

You can install TensorFlow using "native" pip or Anaconda. I will present the steps of the "native" pip installation of the CPU version of TensorFlow. Native pip installs TensorFlow directly on your system without going through a virtual environment. Since a native pip installation is not walled-off in a separate container, the pip installation might interfere with other Python-based installations on your system. However, if you understand pip and your Python environment, a native pip installation often entails only a single command! Furthermore, if you install with native pip, users can run TensorFlow programs from any directory on the system.
If you don't have Python 3.5.x or 3.6.x 64-bit installed on your system, you can install it from python.org. To install TensorFlow, start a terminal. Then issue the appropriate pip3 install command in that terminal. To install the CPU-only version of TensorFlow, enter the following command:

```
C:> pip3 install –upgrade tensorflow
```

Now you must validate your installation. Start a terminal.Invoke python from your shell as follows:

```
$ python
```

Enter the following short program inside the python interactive shell:

```
>>> import tensorflow as tf
>>> hello = tf.constant('Hello, TensorFlow!')
>>> sess = tf.Session()
>>> print(sess.run(hello))
```

If the system outputs the following, then you are ready to begin writing TensorFlow programs:

```
Hello, TensorFlow!
```

1.3 Running an example

1.3.1 Prerequisites

Prior to using the sample code in this document, you'll need to do the following:

- TensorFlow installed.
- Install or upgrade pandas by issuing the following command:
  ```
  pip install pandas
  ```

1.3.2 Getting the sample code (optional)

Take the following steps to get the sample code we'll be going through:

Clone the TensorFlow Models repository from GitHub by entering the following command:

```
git clone https://github.com/tensorflow/models
```

Change directory within that branch to the location containing the examples used in this document:

```
cd models/samples/core/get_started/
```

The program described in this document is premade_estimator.py. This program uses iris_data.py to fetch its training data.
If you wish to use another program, just skip this step and go straight to Running the program.

1.3.3 Running the program

You run TensorFlow programs as you would run any Python program. For example:

```
python premade_estimator.py
```

The program should output training logs followed by some predictions against the test set. For example, the first line in the following output shows that the model thinks there is a 99.6% chance that the first example in the test set is a Setosa. Since the test set expected Setosa, this appears to be a good prediction.

```
...
Prediction is "Setosa" (99.6%), expected "Setosa"

Prediction is "Versicolor" (99.8%), expected "Versicolor"

Prediction is "Virginica" (97.9%), expected "Virginica"
```

1.4 Theoretical Aspects

1.4.1 Data Representation

For text classification algorithms, we can represent the data as a bag of words. A bag of words is just a model where you ID each unique word in your corpus. The bag of words idea brings us to the idea of using a LEXICON. LEXICON as mentioned earlier is a dictionary of all the important words (you can choose the importance) and it follows something called as a hot-array notation. Hot array notations are really crucial to the functioning of a text mining project if youre going down the NLTK road and employing the Neural Network analysis methodology. Something called as a one-hot array notation is also used where it works like a switch-one component is on while the rest remain off.

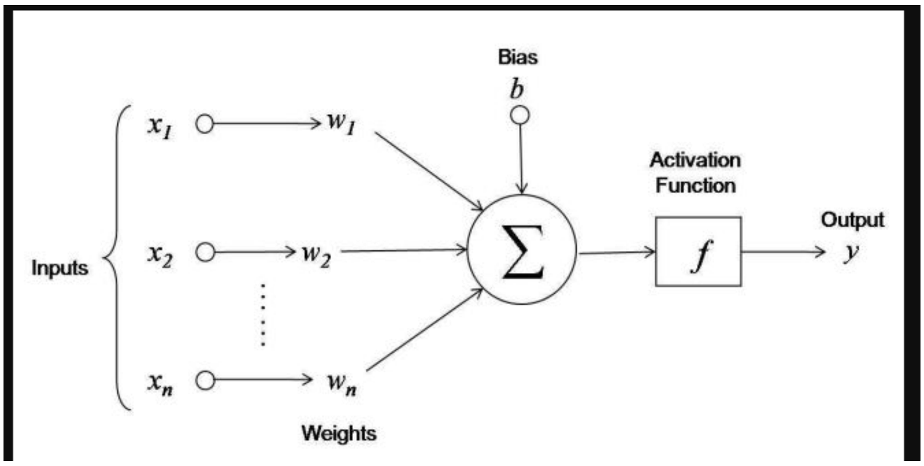
The dataset used consist of text lines, positive and negative sentences. We create a set of features where all the contents i.e all the words are zero initially. We do this because as of now, we have no idea as to which word is falling in the lexicon and which word is not. Once they are initialized to zero and the list of current words is full of all words initialized to zero, we move ahead word by word to check if the word is in lexicon. If the word happens to be in the lexicon, the index of that word in the lexicon is stored and appended into the features till the very last word is taken care off. In this manner, we create our featureset, which consists of a set of features along with their classification (of the sentiment) either positive or negative.

1.4.2 Algorithm

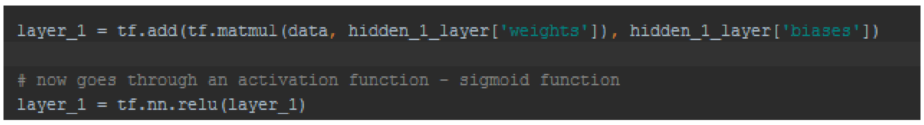
The algorithm consists of several steps:
1. Reading the data.
2. Building the dictionary.
3. Generating a training batch for the model.
4. Building and training the model.
5. Beginning the training.
6. Visualising the embeddings.

A neural network is used, using a few hidden layers. With each layer, we have a weight as well as a bias associated with it. If you look closely, you can see that the output associated with the first layer serves as the input for the second layer. Output for the second layer serves as the input for the third layer so on and so forth.

Once you have defined the hidden layers and the output layer, you go ahead and define each layer separately based on the neural network equation:



For instance, the following equation for layer-1 translates the above visuals into a clean equation:



```
layer_1 = tf.add(tf.matmul(data, hidden_1_layer['weights']), hidden_1_layer['biases'])

# now goes through an activation function - sigmoid function
layer_1 = tf.nn.relu(layer_1)
```

The weights are multiplied with the input data. Then, together with the bias, we sum the combined weight obtained from above. Finally, nn.relu is the sigmoidal activation function which comes inbuilt with the TensorFlow package. We pass our combined value of layer 1 through this activation function.

1.5 Proposed Problem

1.5.1 Specification

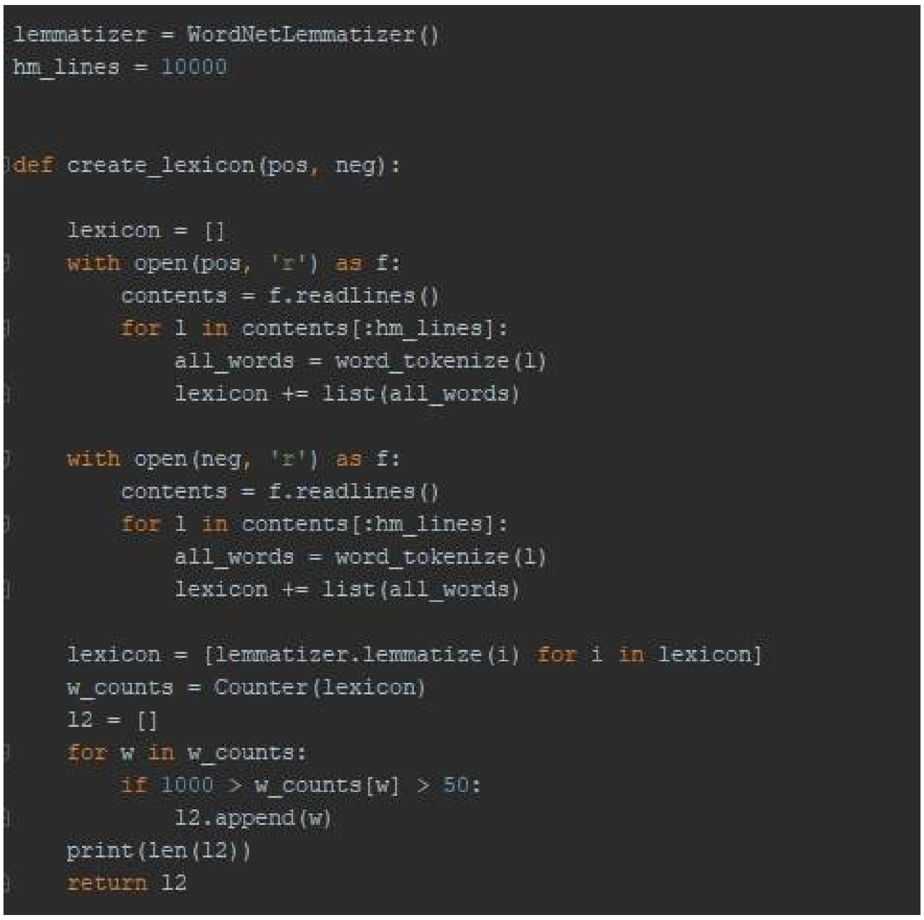
Using the NLTK library, TensorFlow and a neural network i will try to predict the general meaning of a sentence, a positive or negative meaning. The dataset I will be using will consist of two text files each containing 5000 positive/negative sentences.

1.5.2 Solution

In order to solve the problem as simple as I can, I used the NLTK library which offers the Lemmatizer, which will be explained in a later section, the numpy library in order to process numbers and the TensorFlow package that offers the possibility to work with neural networks. I used a model of a feed-forwarding neural network containing 3 hidden layers and an output layer, each hidden layer having 500 nodes. I have also used the Rectified Linear Unit sigmoidal activation function which comes inbuilt with the TensorFlow package. The first set is to parse the dataset and create a featureset and then labels. The parsed dataset results in a lexicon containing the most common used words, but not too common, for example pronouns and prepositions are not included because they could influence the result in an unwanted manner. Then we take the dataset and for each sentence we create an array with the length of the lexicon, and if we find a word in the sentence that is present in the lexicon, we increment the corresponding position. The final set of features consists of an array of arrays, each array pair being the above mentioned array containing the words, and a label, [0,1] for negative sentences and [1,0] for positive sentences. This final set of features is then used to train the neural network. After the training, we define a function that takes as an input a sentence and using the trained neural network returns the classification of the sentence. The accuracy is about 60%, the dataset is small but easy to use.

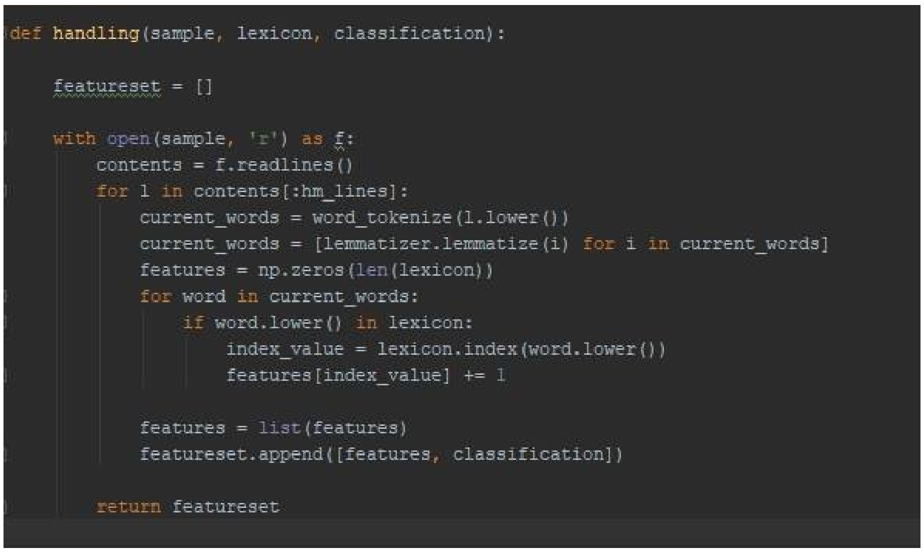
1.5.3 Implementation

In this section I will provide the snippets of code used for each step of the algorithm.
The creation of the lexicon:



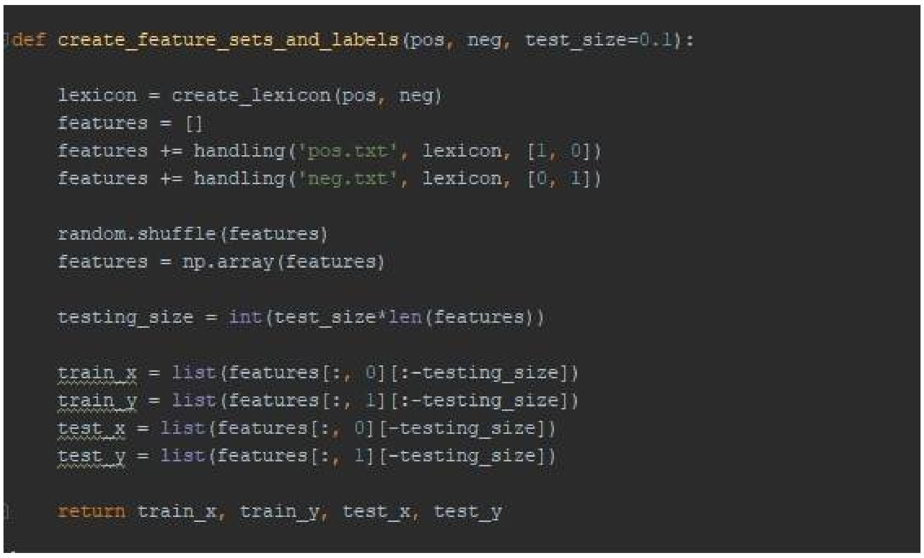
```python
lemmatizer = WordNetLemmatizer()
hm_lines = 10000


def create_lexicon(pos, neg):

    lexicon = []
    with open(pos, 'r') as f:
        contents = f.readlines()
        for l in contents[:hm_lines]:
            all_words = word_tokenize(l)
            lexicon += list(all_words)

    with open(neg, 'r') as f:
        contents = f.readlines()
        for l in contents[:hm_lines]:
            all_words = word_tokenize(l)
            lexicon += list(all_words)

    lexicon = [lemmatizer.lemmatize(i) for i in lexicon]
    w_counts = Counter(lexicon)
    l2 = []
    for w in w_counts:
        if 1000 > w_counts[w] > 50:
            l2.append(w)
    print(len(l2))
    return l2
```

The creation of the featureset:



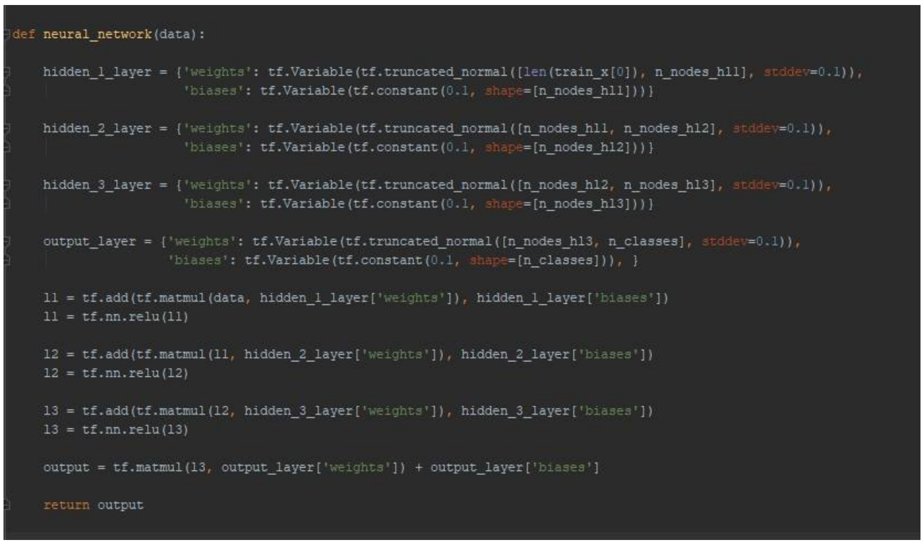
```python
def handling(sample, lexicon, classification):

    featureset = []

    with open(sample, 'r') as f:
        contents = f.readlines()
        for l in contents[:hm_lines]:
            current_words = word_tokenize(l.lower())
            current_words = [lemmatizer.lemmatize(i) for i in current_words]
            features = np.zeros(len(lexicon))
            for word in current_words:
                if word.lower() in lexicon:
                    index_value = lexicon.index(word.lower())
                    features[index_value] += 1

            features = list(features)
            featureset.append([features, classification])

        return featureset
```

The creation of the final set of features:



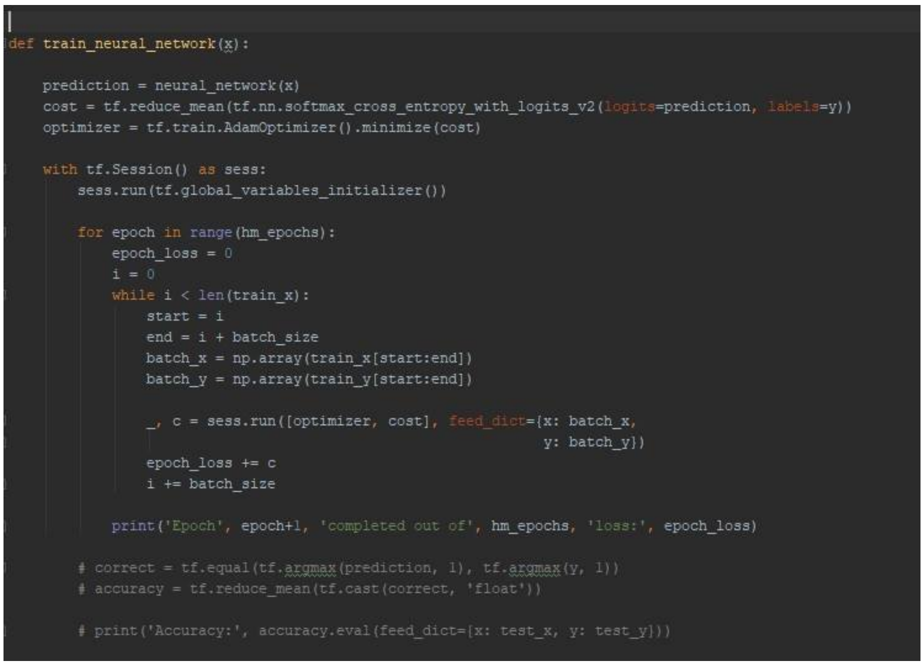
```python
def create_feature_sets_and_labels(pos, neg, test_size=0.1):

    lexicon = create_lexicon(pos, neg)
    features = []
    features += handling('pos.txt', lexicon, [1, 0])
    features += handling('neg.txt', lexicon, [0, 1])

    random.shuffle(features)
    features = np.array(features)

    testing_size = int(test_size*len(features))

    train_x = list(features[:, 0][:-testing_size])
    train_y = list(features[:, 1][:-testing_size])
    test_x = list(features[:, 0][-testing_size])
    test_y = list(features[:, 1][-testing_size])

    return train_x, train_y, test_x, test_y
```

The neural network model:



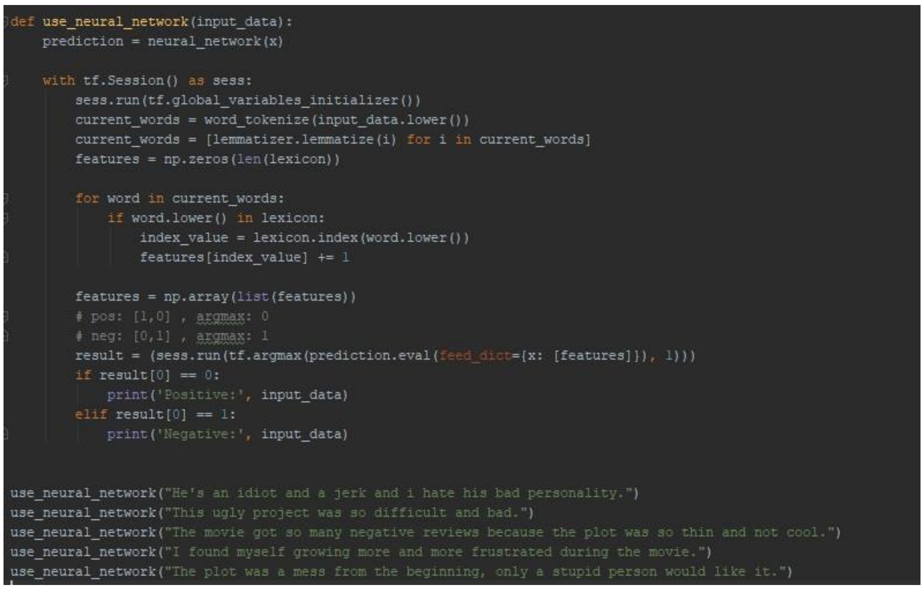
```python
def neural_network(data):

    hidden_1_layer = {'weights': tf.Variable(tf.truncated_normal([len(train_x[0]), n_nodes_hl1], stddev=0.1)),
                      'biases': tf.Variable(tf.constant(0.1, shape=[n_nodes_hl1]))}

    hidden_2_layer = {'weights': tf.Variable(tf.truncated_normal([n_nodes_hl1, n_nodes_hl2], stddev=0.1)),
                      'biases': tf.Variable(tf.constant(0.1, shape=[n_nodes_hl2]))}

    hidden_3_layer = {'weights': tf.Variable(tf.truncated_normal([n_nodes_hl2, n_nodes_hl3], stddev=0.1)),
                      'biases': tf.Variable(tf.constant(0.1, shape=[n_nodes_hl3]))}

    output_layer = {'weights': tf.Variable(tf.truncated_normal([n_nodes_hl3, n_classes], stddev=0.1)),
                    'biases': tf.Variable(tf.constant(0.1, shape=[n_classes])), }

    l1 = tf.add(tf.matmul(data, hidden_1_layer['weights']), hidden_1_layer['biases'])
    l1 = tf.nn.relu(l1)

    l2 = tf.add(tf.matmul(l1, hidden_2_layer['weights']), hidden_2_layer['biases'])
    l2 = tf.nn.relu(l2)

    l3 = tf.add(tf.matmul(l2, hidden_3_layer['weights']), hidden_3_layer['biases'])
    l3 = tf.nn.relu(l3)

    output = tf.matmul(l3, output_layer['weights']) + output_layer['biases']

    return output
```

The neural network training function:



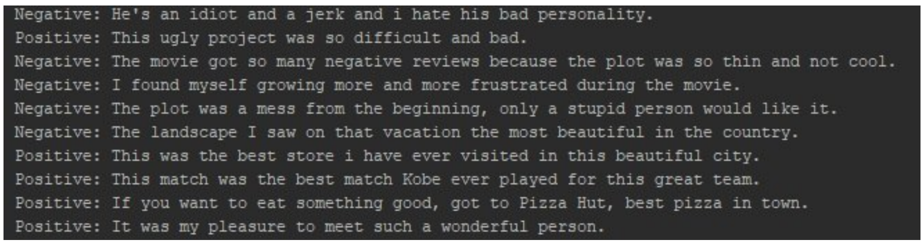
```python
def train_neural_network(x):

    prediction = neural_network(x)
    cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=prediction, labels=y))
    optimizer = tf.train.AdamOptimizer().minimize(cost)

    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())

        for epoch in range(hm_epochs):
            epoch_loss = 0
            i = 0
            while i < len(train_x):
                start = i
                end = i + batch_size
                batch_x = np.array(train_x[start:end])
                batch_y = np.array(train_y[start:end])

                _, c = sess.run([optimizer, cost], feed_dict={x: batch_x,
                                                              y: batch_y})
                epoch_loss += c
                i += batch_size

            print('Epoch', epoch+1, 'completed out of', hm_epochs, 'loss:', epoch_loss)

        # correct = tf.equal(tf.argmax(prediction, 1), tf.argmax(y, 1))
        # accuracy = tf.reduce_mean(tf.cast(correct, 'float'))

        # print('Accuracy:', accuracy.eval(feed_dict={x: test_x, y: test_y}))
```

The function that returns the sentiment of a sentence:



```python
def use_neural_network(input_data):
    prediction = neural_network(x)

    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        current_words = word_tokenize(input_data.lower())
        current_words = [lemmatizer.lemmatize(i) for i in current_words]
        features = np.zeros(len(lexicon))

        for word in current_words:
            if word.lower() in lexicon:
                index_value = lexicon.index(word.lower())
                features[index_value] += 1

        features = np.array(list(features))
        # pos: [1,0] , argmax: 0
        # neg: [0,1] , argmax: 1
        result = (sess.run(tf.argmax(prediction.eval(feed_dict={x: [features]}), 1)))
        if result[0] == 0:
            print('Positive:', input_data)
        elif result[0] == 1:
            print('Negative:', input_data)


use_neural_network("He's an idiot and a jerk and i hate his bad personality.")
use_neural_network("This ugly project was so difficult and bad.")
use_neural_network("The movie got so many negative reviews because the plot was so thin and not cool.")
use_neural_network("I found myself growing more and more frustrated during the movie.")
use_neural_network("The plot was a mess from the beginning, only a stupid person would like it.")
```

Final results:



```
Negative: He's an idiot and a jerk and i hate his bad personality.
Positive: This ugly project was so difficult and bad.
Negative: The movie got so many negative reviews because the plot was so thin and not cool.
Negative: I found myself growing more and more frustrated during the movie.
Negative: The plot was a mess from the beginning, only a stupid person would like it.
Negative: The landscape I saw on that vacation the most beautiful in the country.
Positive: This was the best store i have ever visited in this beautiful city.
Positive: This match was the best match Kobe ever played for this great team.
Positive: If you want to eat something good, got to Pizza Hut, best pizza in town.
Positive: It was my pleasure to meet such a wonderful person.
```