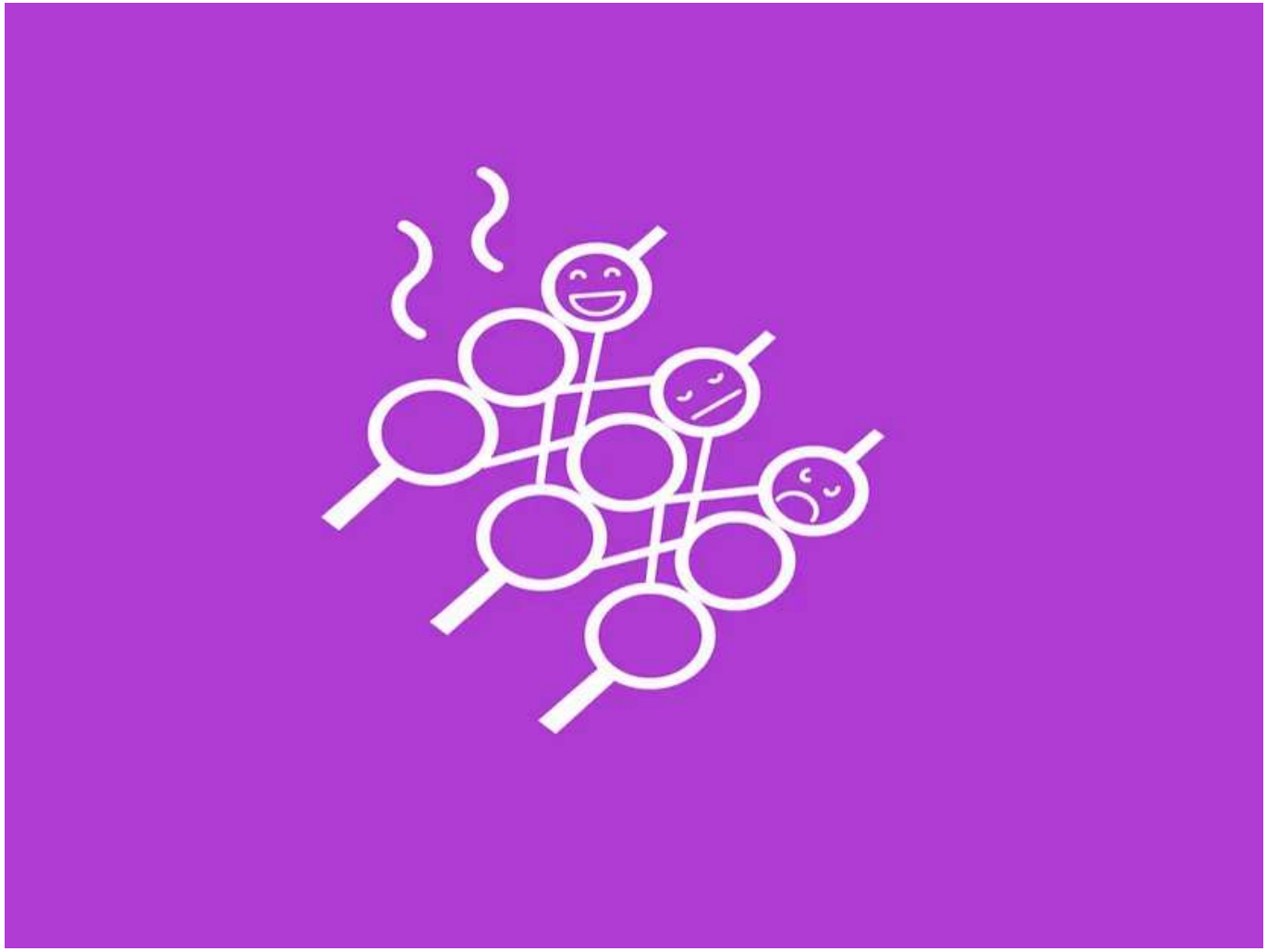
Sentiment Classification using Feed Forward Neural Network in PyTorch

Dipika Baad · Follow
Published in The Startup · 12 min read · Mar 23, 2020

Implementing Sentiment Classification For Restaurant Reviews Taken From Yelp using Feed Forward Neural Network in PyTorch



Sentiment Classification using Feed Forward Neural Network in PyTorch by Dipika Baad

In this article, I will explain how the Feed forward neural network can be used for text classification problems and how to define the neural network using PyTorch. You will understand how to build a custom feed forward neural network in PyTorch for a sentiment classification problem.

In my previous post, I introduced the basics of PyTorch and how to implement Logistic Regression for Sentiment Classification. You can refer to that if you are new to PyTorch. I have explained in the previous posts other methods for Sentiment Classification using BOW, TF-IDF, Word2Vec and Doc2Vec vectors using Decision Tree Classifier, which will be compared at the end as well. Let's start with loading the data now!

Restaurant Reviews by Sentiment Example by Dipika Baad

Load the data

Yelp restaurant review dataset can be downloaded from their site and the format of the data present there is JSON. The data provided is actually not in correct json format readable for python. Each row is dictionary but for it to be a valid json format, a square bracket should be at the start and end of the file with , being added at end of each row. Define the `INPUT_FOLDER` as folder path in your local directory where yelp review.json file is present. Declare `OUTPUT_FOLDER` as a path where you want to write the output from the following function. Loading of json data and writing the top 100,000 rows is done in the following function:

```python
import pandas as pd

def load_yelp_orig_data():
    PATH_TO_YELP_REVIEWS = INPUT_FOLDER + '/review.json'

    # read the entire file into a python array
    with open(PATH_TO_YELP_REVIEWS, 'r') as f:
        data = f.readlines()

    # remove the trailing "\n" from each line
    data = map(lambda x: x.rstrip(), data)

    data_json_str = "[" + ','.join(data) + "]"

    # now, load it into pandas
    data_df = pd.read_json(data_json_str)

    data_df.head(100000).to_csv(OUTPUT_FOLDER + '/output_reviews_top.csv')

load_yelp_orig_data()
```

load_yelp_reviews.py hosted with ❤ by GitHub view raw

Once the above function has been run, you are ready to load it in pandas dataframe for the next steps. For the experiment, only small amount of data is taken so that it can be run faster to see the results.

Exploring data

After the data is loaded, new column for sentiment indication is created. It is not always the situation that some column with the prediction label you want to do is present in the original dataset. This can be a derived column in most of the cases. For this case, `stars` column in the data is used to derive sentiment.

```
top_data_df = pd.read_csv(INPUT_FOLDER + 'output_reviews_top.csv')
print("Columns in the original dataset:\n")
print(top_data_df.columns)
```

loading_in_dataframe.py hosted with ❤ by GitHub view raw

Output:

```
Columns in the original dataset:

Index(['Unnamed: 0', 'business_id', 'cool', 'date', 'funny', 'review_id',
       'stars', 'text', 'useful', 'user_id'],
      dtype='object')
```

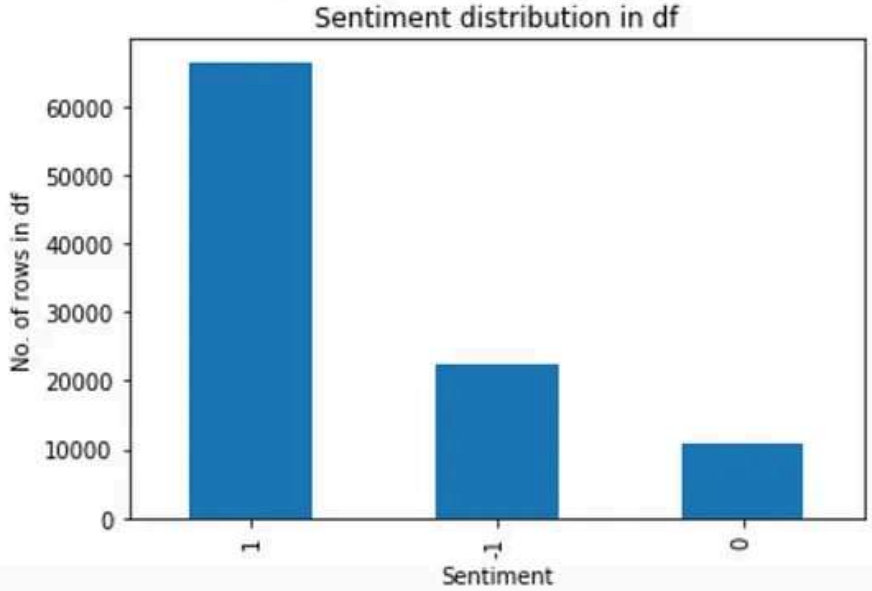
After the data is available, mapping from stars to sentiment is done and distribution for each sentiment is plotted.

```python
import matplotlib.pyplot as plt

print("Number of rows per star rating:")
print(top_data_df['stars'].value_counts())

# Function to map stars to sentiment
def map_sentiment(stars_received):
    if stars_received <= 2:
        return -1
    elif stars_received == 3:
        return 0
    else:
        return 1
# Mapping stars to sentiment into three categories
top_data_df['sentiment'] = [ map_sentiment(x) for x in top_data_df['stars']]
# Plotting the sentiment distribution
plt.figure()
pd.value_counts(top_data_df['sentiment']).plot.bar(title="Sentiment distribution in df")
plt.xlabel("Sentiment")
plt.ylabel("No. of rows in df")
plt.show()
```

mapping_stars_to_sentiment.py hosted with ❤️ by GitHub view raw

Output:

```
Number of rows per star rating:
5    44367
4    22103
1    14585
3    10970
2     7975
Name: stars, dtype: int64
```



Once that is done, number of rows for each sentiment is checked. Sentiment Classes are as follows:

1. Positive : 1
2. Negative: -1
3. Neutral: 0

Number of rows are not equally distributed across these three sentiments. In this post, problem of imbalanced classes won't be dealt that is why, simple function to retrieve the top few records for each sentiment is written. In this example, `top_n` is 10000 which means total of 30,000 records will be taken.

```
# Function to retrieve top few number of each category
def get_top_data(top_n = 5000):
    top_data_df_positive = top_data_df[top_data_df['sentiment'] == 1].head(top_n)
    top_data_df_negative = top_data_df[top_data_df['sentiment'] == -1].head(top_n)
    top_data_df_neutral = top_data_df[top_data_df['sentiment'] == 0].head(top_n)
    top_data_df_small = pd.concat([top_data_df_positive, top_data_df_negative, top_data_df_neutra
    return top_data_df_small

# Function call to get the top 10000 from each sentiment
top_data_df_small = get_top_data(top_n=10000)

# After selecting top few samples of each sentiment
print("After segregating and taking equal number of rows for each sentiment:")
print(top_data_df_small['sentiment'].value_counts())
top_data_df_small.head(10)
```

load_top_reviews.py hosted with ❤ by GitHub view raw

Output:

```
After segregating and taking equal number of rows for each sentiment:
-1    10000
 1    10000
 0    10000
Name: sentiment, dtype: int64
```

	Unnamed: 0	business_id	cool	date	funny	review_id	stars	text	useful	user_id	sentiment
1	1	NZnhc2sEQy3RmzKTZnqtwQ	0	2017-01-14 21:30:33	0	GJXCdrto3ASJOqKeVWPi6Q	5	I *adore* Travis at the Hard Rock's new Kelly ...	0	yXQM5uF2jS6es16SJzNHfg	1
2	2	WTqjgwHlXbSFevF32_DJVw	0	2016-11-09 20:09:03	0	2TzJjDVDEuAW6MR5Vuc1ug	5	I have to say that this office really has it t...	3	n6-Gk65cPZL6Uz8qRm3NYw	1
3	3	ikCg8xy5JIg_NGPx-MSIDA	0	2018-01-09 20:56:38	0	yi0R0Ugj_xUx_Nek0-_Qig	5	Went in for a lunch. Steak sandwich was delici...	0	dacAIZ6fTM6mqwW5uxkskg	1
5	5	eU_713ec6fTGNO4BegRaww	0	2013-01-20 13:25:59	0	fdiNeiN_hoCxCMy2wTRW9g	4	I'll be the first to admit that I was not exci...	0	w31MKYsNFMrjhWxxAb5wIw	1
10	10	8mIrX_LrOnAqWsB5JrOojQ	0	2011-11-30 02:11:15	0	kbtscdyz6lvrtGjD1quQTg	4	Like walking back in time, every Saturday morn...	0	Fik4lQQu1eTe2EpzQ4xhBA	1
12	12	FxLfqxdYPA6Z85PFKaqLrg	0	2016-05-07 01:36:53	0	Z7wgXp98wYB57QdRY3HQ3w	4	Wow. So surprised at the one and two star revi...	0	GYNnVehQeXjty0xH7-6Fhw	1
13	13	LUN6swQYa4xJKaM_UEUOEw	0	2018-04-27 20:25:26	0	qIXw1JQ0UodW7qrmVgwCXw	4	Michael from Red Carpet VIP is amazing ! I rea...	0	bAhqAPoWaZYcyYi7bs024Q	1

How to preprocess text data?

Preprocessing involves many steps like tokenization, removing stop words, stemming/lemmatization etc. These commonly used techniques were explained in detail in my previous post of BOW. Here, only the necessary steps are explained in the next phase.

Why do you need to preprocess this text? — Not all the information is useful in making predictions or doing classifications. Reducing the number of words will reduce the input dimension to your model. The way the language is written, it contains lot of information which is grammar specific. Thus when converting to numeric format, word specific characteristics like capitalisation, punctuations, suffixes/prefixes etc. are redundant. Cleaning

the data in a way that similar words map to single word and removing the grammar relevant information from text can tremendously reduce the vocabulary. Which methods to apply and which ones to skip depends on the problem at hand.

1. Removal of Stop Words

Stop words are the words which are commonly used and removed from the sentence as pre-step in different Natural Language Processing (NLP) tasks. Example of stop words are: ‘a’, ‘an’, ‘the’, ‘this’, ‘not’ etc. Every tool uses a bit different set of stop words list that it removes but this technique is avoided in cases where phrase structure matters like in this case of Sentiment Analysis.

Example of removing stop words:

```
# Removing the stop words
from gensim.parsing.preprocessing import remove_stopwords
print(remove_stopwords("Restaurant had a really good service!!"))
print(remove_stopwords("I did not like the food!!"))
print(remove_stopwords("This product is not good!!"))
```

stop_words_removal.py hosted with ❤ by GitHub view raw

Output:

```
Restaurant good service!!
I like food!!
This product good!!
```

As it can be seen from the output, removal of stop words removes necessary words required to get the sentiment and sometimes it can totally change the meaning of the sentence. In the examples printed by above piece of code, it is clear that it can convert a negative statement into positive sentence. Thus, this step is skipped for Sentiment Classification.

2. Tokenization

Tokenization is the process in which the sentence/text is split into array of words called tokens. This helps to do transformations on each words separately and this is also required to transform words to numbers. There are different ways of performing tokenization. I have explained these ways in my previous post under Tokenization section, so if you are interested you can check it out.

remove punctuations. It has `min` and `max` length parameters as well which help to filter out rare words and most commonly words which will fall in that range of lengths.

Here, `simple_preprocess` is used to get the tokens for the dataframe as it does most of the preprocessing already for us. Let's apply this method to get the tokens for the dataframe:

Output:

```
1     [adore, travis, at, the, hard, rock, new, kell...
2     [have, to, say, that, this, office, really, ha...
3     [went, in, for, lunch, steak, sandwich, was, d...
5     [ll, be, the, first, to, admit, that, was, not...
10    [like, walking, back, in, time, every, saturda...
12    [wow, so, surprised, at, the, one, and, two, s...
13    [michael, from, red, carpet, vip, is, amazing,...
15    [you, can, really, find, anything, wrong, with...
16    [great, lunch, today, staff, was, very, helpfu...
18    [we, ve, been, huge, slim, fan, since, they, o...
Name: tokenized_text, dtype: object
```

3. Stemming

Stemming process reduces the words to its' root word. Unlike Lemmatization which uses grammar rules and dictionary for mapping words to root form, stemming simply removes suffixes/prefixes. Stemming is widely used in the application of SEOs, Web search results, and information retrieval since as long as the root matches in the text somewhere it helps to retrieve all the related documents in the search.

There are different algorithms used to do the stemming. PorterStammer(1979), LancasterStammer (1990), and SnowballStemmer (can add custom rules). NLTK or Gensim package can be used for implementing these algorithms for stemming. Lancaster is bit slower than Porter so we can use it according to size and response time required. Snowball stemmer is a slightly improved version of the Porter stemmer and

is usually preferred over the latter. It is not very clear which one will produce accurate results, so one has to experiment different methods and choose the one that gives better results. In this example, Porter Stemmer is used which is simple and speedy. Following code shows how to implement stemming on dataframe and new column `stemmed_tokens` is created:

Output:

```
1     [ador, travi, at, the, hard, rock, new, kelli,...
2     [have, to, sai, that, thi, offic, realli, ha, ...
3     [went, in, for, lunch, steak, sandwich, wa, de...
5     [ll, be, the, first, to, admit, that, wa, not,...
10    [like, walk, back, in, time, everi, saturdai, ...
12    [wow, so, surpris, at, the, on, and, two, star...
13    [michael, from, red, carpet, vip, is, amaz, re...
15    [you, can, realli, find, anyth, wrong, with, t...
16    [great, lunch, todai, staff, wa, veri, help, i...
18    [we, ve, been, huge, slim, fan, sinc, thei, op...
Name: stemmed_tokens, dtype: object
```

Splitting into Train and Test Sets:

Train data would be used to train the model and test data is the data on which the model would predict the classes and it will be compared with original labels to check the accuracy or other model test metrics.

- Train data (Subset of data for training ML Model) ~70%
- Test data (Subset of data for testing ML Model trained from the train data) ~30%

Try to balance the number of classes in both the sets so that the results are not biased or one of the reasons for insufficient model training. This is a crucial part of machine learning model. In real-world problems, there are cases of imbalanced classes which needs using techniques like oversampling

minority class, undersampling majority class (Resample function from scikit-learn packaged or generating synthetic samples using SMOTE functionality in Imblearn package .

For this case, the data is split into two parts, train and test with 70% in train and 30% in test. While making the splitting, it is better to have equal distribution of classes in both train and test data. Here, function train_test_split from scikit-learn package is used.

Output:

```
Value counts for Train sentiments
 1    7036
-1    7008
 0    6956
Name: sentiment, dtype: int64
Value counts for Test sentiments
 0    3044
-1    2992
 1    2964
Name: sentiment, dtype: int64
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.series.Series'>
   index  ...                                    stemmed_tokens
0  34963  ...  [wait, minut, onli, to, get, to, front, of, th...
1  21265  ...  [chicken, bacon, ranch, pizza, wa, pretti, bom...
2  36472  ...  [as, sketchi, as, hell, thi, place, strike, me...
3  36589  ...  [last, time, tri, bubbl, tea, and, it, wa, dis...
4   3117  ...  [so, for, the, game, last, week, wa, visit, th...

[5 rows x 11 columns]
```

As it can be seen from the above output, data is distributed for each classes proportionately. Number of rows for each sentiment in train and test are printed.

Getting Started with PyTorch

Basics of PyTorch and different functions in PyTorch are explained in my previous post, so I will keep it short here but you can refer to my previous article of Logistic Regression with PyTorch if you are new to PyTorch.

We will start with importing the necessary libraries and setting the `torch.device` to whichever processor is available i.e. wither `cpu` or `gpu`.

Main advantage of using PyTorch is that you can run the computations on gpu for faster speed.

Main libraries needed to be included and how the current device is identified is shown in the following code. Where to load the tensor and do the computation is decided with a `device` parameter in different functions used in neural network layers. I have used Google Colab for the experiment and set the runtime to have GPU in hardware accelerator, that is why I can see that `torch.cuda.is_available` is true in my case.

Output:

```
Device available for running:
cuda
```

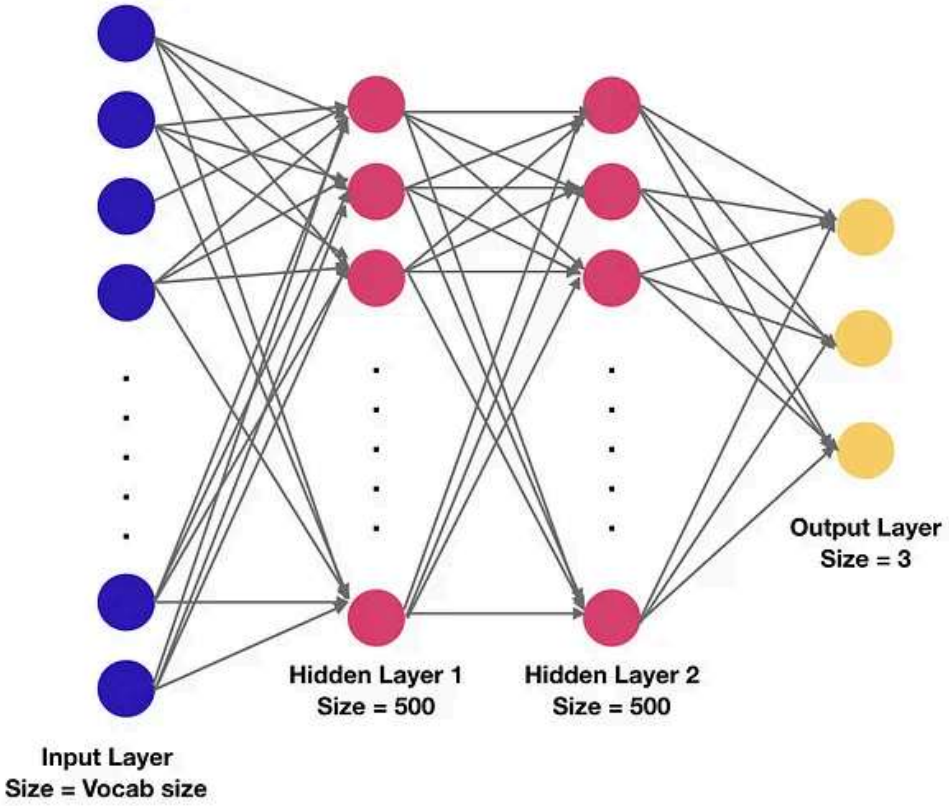
It is hard to go into detail about how neural network works in a short article. In order to get basic information needed to understand the training process, you can read it here. In short, the neural network (NN) definition and training process is as follows for PyTorch.

Steps in PyTorch for NN Model

- Define the NN model
- Override the forward function
- Initialise Optimisation and loss function for training
- Iterate over dataset of inputs
- Compute the loss
- Propagate gradients back into the network's parameters
- Update the weights and biases

Defining Feed Forward Neural Network (FFNN) Model

FFNN model is the simplest form of artificial neural network. Information flows in one direction from first input layer to hidden layer to output layer. You can have any number of hidden layers with different sizes. Output layer in case of classification will be the same size as that of number of classes (for this case 3). Here, I have chosen 2 hidden layers with size 500. You can use different activation functions, here `nn.Relu` is used. Softmax is used for the last layer. You can create a custom network with different functions and different hidden layers to see which one fits the given input data. Let's start with defining the network.



Feed Forward Neural Network architecture for Sentiment Classification by Dipika Baad

Above code shows, how to define the FFNN. Non-linear activation function is used on each hidden layer. Softmax layer is the output layer to get the probabilities for each class and the maximum of that will be the predicted class. As one can see, the class needs to be inherited from `nn.module` and constructor has to be initialised. In the next steps, we will see how to use this and train it.

Generating input and label tensor

First step would be to have functions that can create input tensor and corresponding label that is output tensor which are fed to network for training. For Feed Forward neural network, we will be using BOW vector as the input which is nothing but array with size of vocabulary in the corpus and values are the frequencies of words in corpus with index being the unique id of the word. We will get unique id from the dictionary built using `corpora.Dictionary` of Gensim package. This is similar to what I did in BOW post but I am adding another parameter called padding which will be used in other tasks like CNN where you want to use word embeddings for each words in document. For this example, padding is turned off.

After that you are ready to create bow vector function as follows. Vocab size is `30056`. You can see how I have assigned the device while creating tensor:

For creating the output tensor, mapping of label to positive values has to be done. Currently we had -1 for negative, this is not possible in neural network. Three neurons in the output layer will give probabilities for each label so we just need mapping to positive numbers. Function is as follows:

Training FFNN Model

Now we are ready to start training, before that we will initialize the model. Here I have shown the best result got from different learning rates I experimented. At the end, I will compare the results.

Now, we can start the training. It will run for 100 epochs. The loss at each step will be recorded and written to a file. Loss plot will give us better idea of how well the model is learning and if we need to do early stopping. This graph can give us a better idea of number of epochs to run. You can plot training loss along with validation loss to find if the model is overfitting as well. Let's understand the training loss graph in this article.

Output:

```
Epoch completed: 25
Epoch completed: 50
Epoch completed: 75
Epoch completed: 100
```

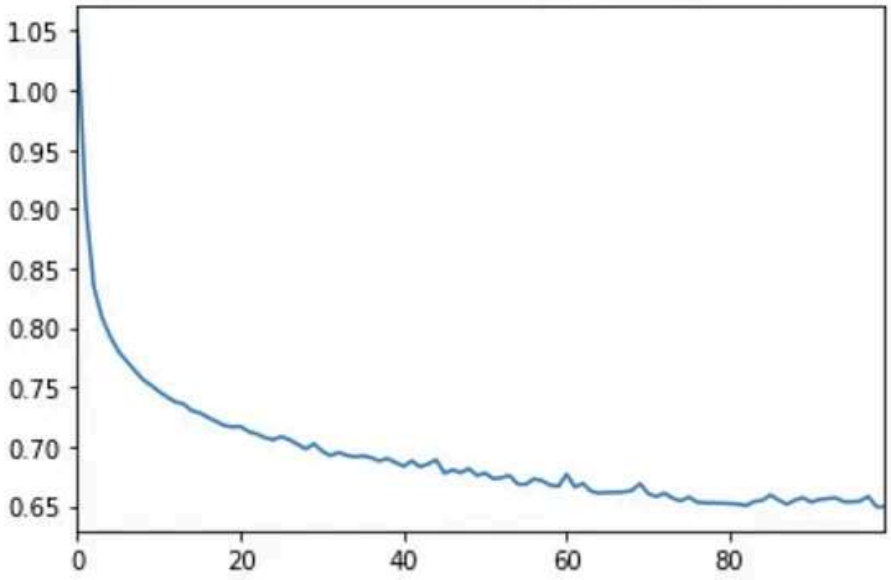
Testing the model

Testing the model code is shown as follows. Loss graph is also plotted and code for saving the plot. This is useful when you are doing multiple experiments and wan to compare results after all combinations of different hyper-parameters.

Output:

	precision	recall	f1-score	support
0	0.80	0.72	0.76	2992
1	0.64	0.69	0.66	3044
2	0.79	0.81	0.80	2964
accuracy			0.74	9000
macro avg	0.75	0.74	0.74	9000
weighted avg	0.74	0.74	0.74	9000

```
100
Index(['iter', ' loss'], dtype='object')
```

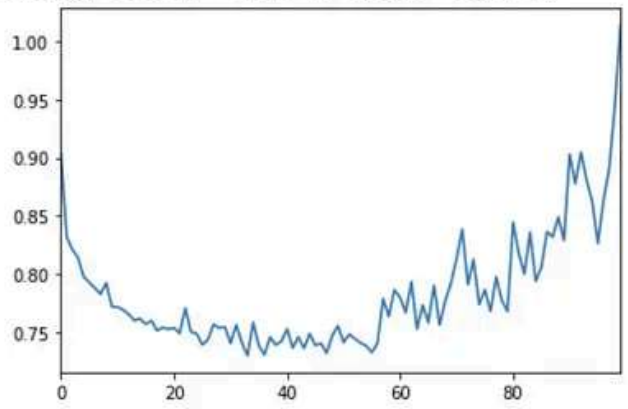


Average accuracy of `0.74` is really good and it is the best accuracy that I obtained compared to other methods done in my previous posts of

Sentiment classification. Accuracy of positive and negative sentiments is higher than the neutral sentiment but that is possible since it doesn't have specific words that can distinguishably used while expressing neutral emotions. Loss graph as you can see that it is steadily decreasing which is a good sign and it is smooth as well. As one can see that almost 40–50 is not reducing drastically and is smoothening out. You can choose the number of epochs based on resources you are willing to spend and how often you are training the models.

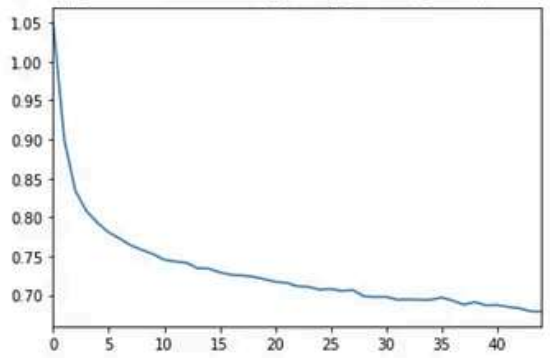
I had run this with learning rate `0.01` which is more than the one above, during which I got plot as shown below when ran for 100 epochs and average accuracy was `0.40` so you can see if the learning rate is more, it is overshooting the local minima and loss is not decreasing. This is when you decide to lower the rate and see the loss graph to see how the model is behaving.

```
Index(['iter', ' loss'], dtype='object')
```



I ran the training for only 45 considering that as the cut point with learning rate `0.001` and the training accuracy was same as 100 epochs. But you can choose according to your preferences of tradeoffs between accuracy and training resources.

```
45
Index(['iter', ' loss'], dtype='object')
```



This accuracy is better than the methods implemented in my previous posts where using Decision Tree classifier was used to classify based on BOW, TF-IDF, Word2Vec and Doc2Vec vectors as input. This shows that neural networks implementing simple feed forward neural network can perform better with simple BOW vectors trained for many epochs. It can pick up the relations between words and sentiments and classify better.

So now you can easily experiment for your own dataset with this method! I hope this helped you to understand how to use PyTorch to build neural network model to do the sentiment analysis on restaurant reviews data. Feel free to extend this code! This is applicable to any other text classification problems where multiple classes are there. If I can think about improving this model, I would use different learning rate, epochs, other non-linear activation functions like `tanh`, `sigmoid` etc. , other optimization algorithms like `Adam`, `RMSProp` etc. and try using different input types other than just BOW. You can try TFIDF, Word2Vec, Doc2Vec and see what results you get. Preprocessing can be changed to use lemmatization or other stemming algorithms to see how the results change. There is lot of room for experimenting for your project.

As always — Happy experimenting and learning :)

NLP Deep Learning Pytorch Sentiment Classification Restaurant Review

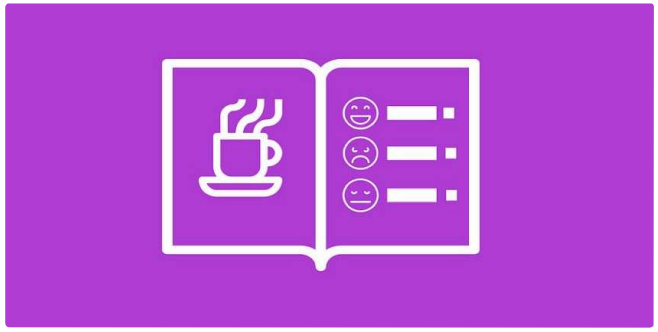
Written by Dipika Baad

312 Followers · Writer for The Startup

Follow

Big Data Consultant @Netlight | CoFounder @HuskyCodes | Web developer | Passionate about coding, dancing, reading

More from Dipika Baad and The Startup



Dipika Baad in The Startup

Sentiment Classification using Word Embeddings (Word2Vec)

Background to Word Embeddings and Implementing Sentiment Classification on...

Mar 2, 2020 290 1

Anangsha Alammyan in The Startup

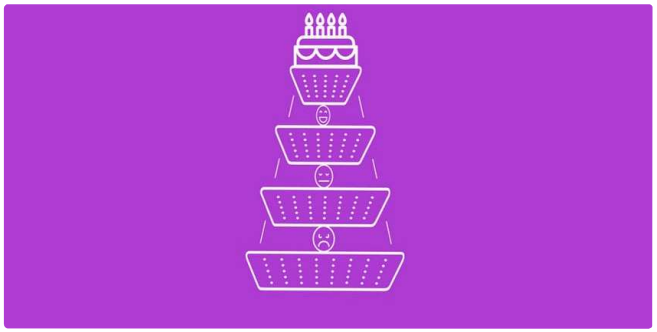
You're Not Making Money As A Writer Because You're Focusing on...

People often ask me how long it took to make money as a complete beginner.

Aug 22 11.2K 318

Zulie Rane in The Startup

If You Want to Be a Creator, Delete All (But Two) Social Media Platforms



Dipika Baad in Towards Data Science

Sentiment Classification using CNN in PyTorch

In October 2022, during the whole Elon Musk debacle, I finally deleted Twitter from my...

Apr 19, 2023 53K 1308

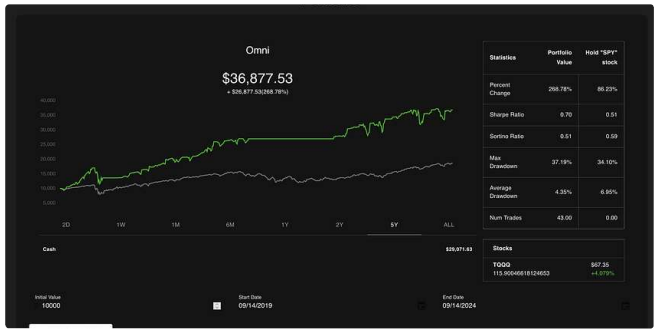
Implementing Convolutional Neural Network (CNN) with word2vec embeddings as input t...

Mar 30, 2020 142 4

See all from Dipika Baad

See all from The Startup

Recommended from Medium



Austin Starks in DataDrivenInvestor

I used OpenAI's o1 model to develop a trading strategy. It is...

Nutan

Text Classification with Recurrent Neural Network

It literally took one try. I was shocked.

Sep 16 5K 136

In this blog, we will train a recurrent neural network on the IMDB large movie review...

May 12 204

Lists

Natural Language Processing

1780 stories · 1384 saves

The New Chatbots: ChatGPT, Bard, and Beyond

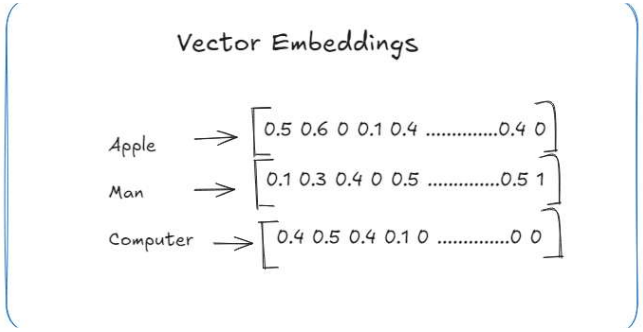
12 stories · 491 saves

data science and AI

40 stories · 271 saves

Practical Guides to Machine Learning

10 stories · 1978 saves



Mdabdullahalhasib in Towards AI

Deepankar Singh in AI-Enthusiast

A Complete Guide to Embedding For NLP & Generative AI/LLM

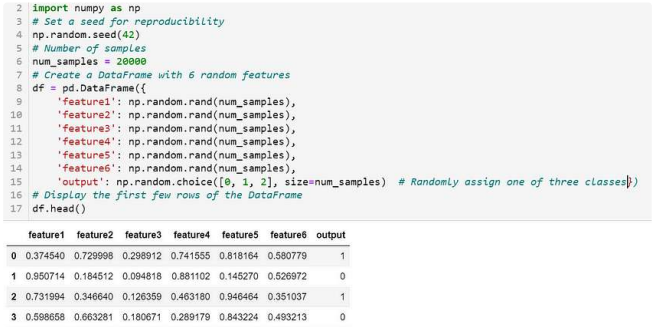
Understand the concept of vector embedding, why it is needed, and...

Oct 19 83

Building a Multi-Class Text Classifier with BERT: A Step-by-Step Guide...

Unlock the power of BERT for multi-class text classification! Dive into its architecture, fine-...

Sep 7 14 2



```
import numpy as np
# Set a seed for reproducibility
np.random.seed(42)
# Number of samples
num_samples = 20000
# Create a DataFrame with 6 random features
df = pd.DataFrame({
    'feature1': np.random.rand(num_samples),
    'feature2': np.random.rand(num_samples),
    'feature3': np.random.rand(num_samples),
    'feature4': np.random.rand(num_samples),
    'feature5': np.random.rand(num_samples),
    'feature6': np.random.rand(num_samples),
    'output': np.random.choice([0, 1, 2], size=num_samples)  # Randomly assign one of three classes})
# Display the first few rows of the DataFrame
df.head()
```

	feature1	feature2	feature3	feature4	feature5	feature6	output
0	0.374540	0.729998	0.298912	0.741555	0.818164	0.580779	1
1	0.950714	0.184512	0.094818	0.881102	0.145270	0.526972	0
2	0.731994	0.346640	0.126359	0.463180	0.946464	0.351037	1
3	0.598658	0.663281	0.180671	0.289179	0.843224	0.493213	0

Sarthak Agarwal

Exploring LSTM Architectures for Multi-Class Classification with...

This article dives into two common LSTM model architectures implemented using...

May 18 26

Emmanuel Ikogho

Data Science is dying; here's why

Why 85% of data science projects fail

Sep 3 1.8K 87

See more recommendations