CS 6320.001: Natural Language Processing
Fall 2019
Project 1 Evaluation sheet

Please attach this sheet to the beginning of the project report.
Name: Rishita Bansal
NetID: rxb180044

| **Task no** | **Points** | **Comments** |
|---|---|---|
| | | |
| 1 | | |
| 2 | | |
| 3 | | |
| 4 | | |
| 5 | | |
| 6 | | |
| 7 | | |
| Report | | |
| | | |
| TOTAL | | |

# Sentiment Classification with Deep Learning

Rishita Bansal
The University of Texas at Dallas,
Richardson, TX- 75080 USA
rishita.bansal@utdallas.edu

**Abstract**
We often find movie reviews with different sentiments. Some of them are positive while others are negative. There is a need to identify the sentiments behind these reviews in order to get the quality of the movie. Thus we can perform sentiment Classification using Deep Learning.

## Introduction

We are given many movie reviews classified as negative and positive. We need to train our Recurrent Neural Network model on this data, so that our model can recognize the sentiments behind any movie review.

## Pre-processing of data

We first need to get all the reviews in a list form. We will also make a list of labels corresponding to these reviews. The labels list will store 0 for all negative reviews and 1 for all positive reviews. Then we will create a vocabulary dictionary containing all unique words in the reviews, each having a unique number. We then encode our review list by replacing all the words by the corresponding numbers from our vocabulary dictionary. We will obtain a list of lists containing a list of encodings for each sentence. We also need to pad the encoded list with zeros and ensure that all the encoded lists are of equal length.

## Loading embeddings

We will use a pre-trained Word2Vec embedding to convert our vocabulary dictionary in to an embedding dictionary. We first read the word2vec file using the gensim library provided by pytorch. Then we will match words from our vocabulary dictionary and associates the integers in our dictionary with the tensors in the embeddings.

## Tensor Dataloader

We create tensor dataset and group it according to batch size. Then we can train our data in batches to get good accuracy and manage resources in a better manner.

**RNN Model**
We start with an embedding layer and fully connected layer. Then we add RNN layers and experiment.

**Obeservations from the Models**

Baseline Model:
The baseline model has one Embedding Layer and one Fully Convolutional Layer.

Results:
Epochs = 20
Test loss: 3.001
Test accuracy: 0.570

Adding RNN layer to the baseline model

We can add different types of RNNs between the two layers in the baseline model. We can try Vanilla RNN, GRU and LSTM. We can also try making RNN bi-directional.

LSTM Result:

Epochs = 10
Test loss: 0.693
Test accuracy: 0.600

LSTM Result (Bi- directional):

Epochs = 10
Test loss: 0.693
```
Test accuracy: 0.500
```

GRU Result(Bi- directional):

Epochs = 10

Test loss: 0.693

```
Test accuracy: 0.470
```

GRU Result:

Epochs = 10

```
Test loss: 0.693
```

```
Test accuracy: 0.530
```

Vanilla RNN Results:

Epochs = 10

Test loss: 0.696

```
Test accuracy: 0.500
```

Vanilla RNN Results(Bidirectional):

Epochs = 10

Test loss: 0.695

```
Test accuracy: 0.490
```

Replacing the RNN by self-attention

We have implemented the neural network by replacing RNN with self-attention layer. This will help us reduce the computational complexity using the attention scores. It also makes it easy to implement with parallelization.

Results:
Epochs = 20
Test loss: 13.816
Test accuracy: 0.440

**Convolutional Neural Network(CNN) (Step 8 – Get Creative)**

We can implement this by replacing the RNN layer by a CNN layer.

Result for CNN:

Epochs = 10

Test loss: 0.693

```
Test accuracy: 0.560
```

**Accuracy Table**

| **Model name** | **No of Epochs** | **Test Loss** | **Test Accuracy** |
|---|---|---|---|
| Baseline | 20 | 3.001 | 0.570 |
| RNN - LSTM | 10 | 0.693 | 0.600 |
| RNN - GRU | 10 | 0.693 | 0.530 |
| RNN - Vanilla | 10 | 0.696 | 0.500 |
| RNN – LSTM (Bi-directional) | 10 | 0.693 | 0.500 |
| RNN – GRU (Bi-directional) | 10 | 0.693 | 0.470 |
| RNN – Vanilla (Bi-directional) | 10 | 0.695 | 0.490 |
| Self Attention | 20 | 13.816 | 0.440 |
| CNN | 10 | 0.693 | 0.560 |

**Conclusion**

We can implement different types of layers from the Recurrent Neural Networks and analyzed their performances and compared their performance with different hyperparameters. LSTM gives highest accuracy and lowest error among all the RNN types. We also observe that when we make RNN bi-directional, the accuracy decreases. Self-attention method yields least accuracy and a high errror. We have ovbserved that adding RNN to baseline model increases accuracy and decrease error by a great value. We also observe that the accuracy of CNN is more than GRU and Vanilla RNN but yet less than LSTM RNN. Thus we can say LSTM RNN gives most accuracy and least error.

## References

[1] Bo Pang and Lillian Lee. A sentimental education: Sentiment analysis using subjectivity. In Proceedings of ACL, pages 271–278, 2004.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Advances in neural information processing systems, pages 5998–6008, 2017.