# CS6320 Assignment 2

https://github.com/pritika292/NLP-Assignment-2

Group07 Pritika Priyadarshini PXP210104 Gauri Sagane GXS220013

## 1 Introduction and Data

The goal of this project is to train neural language models on yelp reviews for various establishments using feed forward neural networks and recurrent neural networks. The training data set consists of a list of reviews each having a text component which is the description of the review and its associated star rating. Using our models, we aim to predict the star rating (on a scale of 0 to 4) for our validation data set.

**Data Pre-processing :**

1. Cleaning of data - Convert all text to lowercase as it improves consistency of the words in the vocabulary and prevents unnecessarily large vocabulary. The goal is to remove any potential inconsistencies that arise from using mixed case words.
2. Standardize punctuations - Using punctuations of a single format ensures that text data is consistent and hence easier to compare and process. Punctuations are not removed altogether to avoid loss of context as they might convey some sentiment - such as exclamation marks which emphasize on a particular sentiment.

**Data analysis :** We have training data consisting of 8000 examples, validation data which consists of 800 examples and test data which consists of 800 examples.

| | Count | Max review length | Min review length | Avg review length |
|---|---|---|---|---|
| Training data | 8000 | 989 | 2 | 141 |
| Validation data | 800 | 980 | 5 | 140 |
| Test data | 800 | 782 | 9 | 110 |

Table 1: Data set analysis

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| Training data | 3200 | 3200 | 1600 | 0 | 0 |
| Validation data | 320 | 320 | 160 | 0 | 0 |
| Test data | 0 | 0 | 160 | 320 | 320 |

Table 2: Number of ratings in datasets

## 2 Implementations

### 2.1 FFNN

The code defines a FFNN using Pytorch modules with parameters such as **number of hidden units (h), W1 (weights for the input layer), W2 (weights for the hidden layer), activation function (eg relu), output dimensions** (5, since we have ratings from 0-4) and **loss function (Cross-entropy)**. After analyzing the dataset, we create a vocabulary based on the training data and use it to construct a bag of words representation of the text reviews.

We complete the forward function which defines how the input data is passed through each layer of the neural network model and transformed to give us the predicted label.

- First, the input vector representations of the reviews are multiplied with the weights W1 and activation function is applied on it to introduce non-linearity which helps the model in learning complex data.

- Next, this output from the hidden layer is then multiplied with weights W2 and softmax function is applied to it to obtain the predicted vector for ratings. Since we have multiple classes at the output layer, we use a softmax function to draw a probability distribution over the possible ratings.

```
def forward(self, input_vector):
    # [to fill] obtain first hidden layer representation
    hidden = self.activation(self.W1(input_vector))

    # [to fill] obtain output layer representation
    output = self.W2(hidden)

    # [to fill] obtain probability dist.
    predicted_vector = self.softmax(output)

    return predicted_vector
```

We are using **Stochastic gradient descent as the optimizer** to learn weights based on the loss function. The model is trained on input data with batch size of 16. We compute the average loss for every batch, calculate the gradients for back propagation algorithm and update the weights (after observing all examples in the batch) based on that using **optimizer.step()**.

After training the model, we use it on the validation data set to predict the ratings for reviews and compute the accuracy of the model (number of correct predictions / total predictions).
*Reference material* - https://www.kaggle.com/code/uvinir/a-neural-network-for-sentiment-analysis-pytorch.

### 2.2 RNN

The code defines a RNN model using Pytorch modules with parameters such as **number of hidden units(h)**, **number of layers**, W (weights for the linear layer), **activation function (tanh)** and **loss function (Negative log likelihood in this case).** We use word embeddings which are pre-trained representations of words in vector space, to represent the words from the text reviews.
- To fill in the required code in forward function, we first define the initial hidden state which is a tensor of zeros.
- The RNN then processes the input sequence and hidden will contain the final hidden state after processing the input sequence (i.e last token of the text). This is representative of the entire sequence as RNNs can in principle represent words from the beginning of the sequence.
- The output of this hidden layer is then multiplied with the weights and aggregated. These weights determine how the network makes use of past context in calculating the output for the current input.
- Finally, softmax function is applied on the output representation to obtain the predicted vector for ratings.

```
def forward(self, inputs):
    # [to fill] obtain hidden layer representation
    h0 = torch.zeros(1*self.numOfLayer, 1, self.h)
    _, hidden = self.rnn(inputs, h0)
    # [to fill] obtain output layer representations
    output = self.W(hidden)
    # [to fill] sum over output
    output_sum = torch.sum(output, dim=1)
    # [to fill] obtain probability dist.
    predicted_vector = self.softmax(output_sum)

    return predicted_vector
```

The part that is functionally different from FFNN for the forward pass is that for sentiment classification task, RNN only considers the output of the last token's hidden unit (which in turn depends on previous hidden layer output). This output is multiplied with weights and a class is chosen via softmax distribution over the possible classes. This allows RNNs to have context of the entire input sequence whereas FFNNs have limited window of context. Also, we use word embeddings for RNN whereas for FFNN we use the bag of word representation for each review.
*Reference material* - https://www.kaggle.com/code/kanncaa1/recurrent-neural-network-with-pytorch.

## 3 Experiments and Results

**Evaluations** Here we evaluate the models using accuracy which is simply the measure of how often the classifier correctly predicts. Using accuracy as a metric for evaluation of classification tasks is generally not a good idea because accuracy does not work well when the classes are unbalanced (as they are here). Its a good practice to look at other metrics of the confusion matrix such as Precision, Recall and F1 score. Precision is useful in the cases where False Positive is a higher concern than False Negatives and Recall in cases where False Negative is of higher concern than False Positive.

**Results**
1. **FFNN** - We get the best validation accuracy of **0.59625** when number of **hidden units(h) is 10** and the model is trained for **5 epochs**. We also try multiple variations by changing the hidden unit sizes. On reducing the hidden unit size to 5, we see a dip in the validation accuracy as the capacity of the model might reduce and it does not learn the complex features well. On increasing the hidden unit size to say 20, we see don't see any improvement in the validation set accuracy as the model is prone to over-fitting and doesn't generalize well to new unseen data.

| Hidden unit size(h) | Validation Accuracy |
|---|---|
| 5 | 0.54125 |
| 10 | 0.59625 |
| 20 | 0.5825 |

2. **RNN** - We get the best validation accuracy of **0.45875** when number of **hidden units(h) is 10** and the model is trained for **2 epochs**. We also try multiple variations by changing the hidden unit sizes, for which we have similar observations as FFNN for lesser or greater number of hidden units.

| Hidden unit size(h) | Validation Accuracy |
|---|---|
| 16 | 0.44625 |
| 32 | 0.45875 |
| 64 | 0.4575 |

**RNN Variation** - We tried a variation of the RNN model which uses a random initial hidden state as opposed to the previous implementation which has zero weights for the initial hidden state. The results for this variation with 32 hidden units and 2 epochs is as follows:

| Hidden unit size(h) | Validation Accuracy |
|---|---|
| 32 | 0.45375 |

The output of this variation varies for each implementation as the initial hidden state is randomly initialized as results in different outputs. The above implementation did not improve our validation accuracy as compared to our default implementation. This might be due to vanishing gradient problem during training and the network may have difficulty learning meaningful representations.

## 4 Analysis

- Learning curve for FFNN with 10 hidden units and trained for 5 epochs is given below -

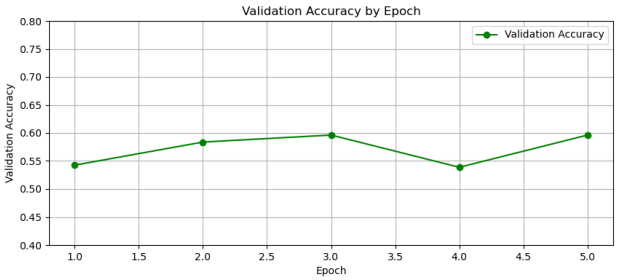


Figure 1: FFNN validation accuracy by epoch.

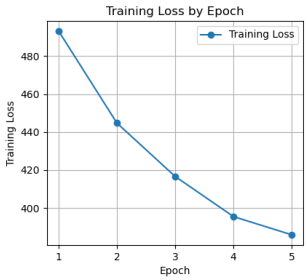


Figure 2: FFNN training loss by epoch.

- Learning curve for RNN with 10 hidden units and trained for 2 epochs is given below -

**Error analysis**

- Here is an example of a review whose rating was incorrectly predicted -

*" I am upgrading my review because the General Manager reached out to us to address our disappointment in a highly professional manner. We will give them another try on our next trip and hope that our experience is more consistent with what others rave about."*

This review is predicted to be of rating 0 (1-star) when its true label is 2 (3-star). The model does not understand the context of this review correctly. It may be focusing on certain keywords like "disappointment" and "hope" and hence it predicts a negative sentiment. It fails to recognize the positive sentiment such as the General Manager's professional response which led the customer to give it another try.

From these observations, we can improve our model in the following ways -

1. *Domain specific jargon* - Here, the model receives reviews from a variety of industries as training data. It might be beneficial to fine tune the model based on more examples from a specific industry such as restaurant/food reviews.
2. *Contextual knowledge* - Using Transformer-based models like BERT, GPT-3 can capture the nuances of language and context better, even capturing neutral reviews will improve the prediction system.

## 5 Conclusion and Others

### 5.1 Brief description of the contributions of each group member

***Pritika*** - Initial data analysis to gain insights on the distribution of data sets and review lengths. Tried appropriate pre-processing methods to clean the text data obtained from reviews. Reading about Feed Forward Neural networks, going through the boiler plate code and understanding PyTorch functions to complete the forward function. Tried multiple variations of the model by varying the hidden unit sizes. Performed error analysis on some of the incorrectly classified labels and researched on ways to improve the model.

***Gauri*** - Implemented the forward function of Recurrent Neural Networks by understanding the concept and compared the results of the algorithm for different number of hidden units. Reported

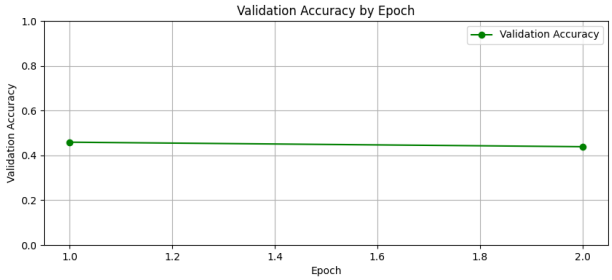


Figure 3: RNN validation accuracy by epoch.

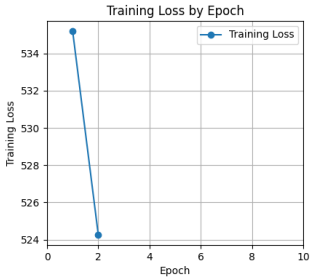


Figure 4: RNN training loss by epoch.

the best model and its accuracy. Also, implemented a variation of RNN which utilizes a different initial hidden state and reported the result for the same.

Combined work on the report to illustrate our findings and understanding of the problems solved in this assignment.

### 5.2 Feedback for the project

This assignment provided us with the opportunity to gain a deeper understanding of neural network models - FFNN and RNN models by implementing a sentiment analysis task. It was of medium difficulty as we were provided with most of the boilerplate code to get started and learning to implement the forward function using PyTorch and understanding the effect of different parameters added to the challenge. We took almost two weeks gain in depth knowledge and implement this assignment. This was a great way of applying theoretical knowledge learnt in class to real world data.