# CS6320 Assignment 2

Github link

Group 10 Sree Vidya Alivelu sxa200150 Likhitha Emmadi lxe220001

## 1. Introduction and Data

For this assignment, we performed 5-class sentiment analysis on a set of Yelp reviews with Recurrent Neural Network (RNN) and a Feed Forward Neural Network (FFNN) using Pytorch.

This assignment's primary goal is to use both FFNN and RNN algorithms to conduct sentiment analysis on a data set of Yelp reviews divided into 5 classes {1, 2, 3, 4, 5}. We were provided with the training data, validation data, and test data in json format. The number of examples in each json file are as shown in the following table.

| Data type and provided | File Name | No. of examples |
|---|---|---|
| Training | training.json | 16000 |
| Validation | validation.json | 800 |
| Test | test.json | 800 |

Each json object has 2 key - value pairs: text, and stars. Text has the review and stars have a number from the set {1,2,3,4,5}.

## 2. Implementations

### 2.1 FFNN

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

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

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

    return predicted_vector
```

The FFNN takes in a fixed amount of input data simultaneously and produces a fixed amount of output simultaneously. In the forward propagation, weights at a specific layer are multiplied with the input vector at that layer. To add non-linearity, the result is run through an activation function (like ReLU). The weights of the following layer are multiplied by the output of this. In order to produce a normalized probability distribution over the output, we have incorporated a soft-max function at the end.

**Understanding the Code:**
SGD and ADAM optimizers have been used here. The optimizer iteratively adjusts the model's parameters to find the values that minimize the loss function and increase the accuracy. We referred to the Pytorch documentation to get a deeper understanding of the optimizers (https://pytorch.org/docs/stable/optim.html)

The class FFNN represents the neural network model. It includes the constructor which Initializes the FFNN with input dimensions, a hidden dimension (h), and output dimensions (set to 5 for a 5-class classification task). compute_Loss method computes the loss. Utility functions - “make_vocab”, “make_indices”, and “convert_to_vector_representation” are used to work with the vocabulary and convert text data to a vector representation. Training and validation data is loaded from JSON files and converted into lists of pairs, where each pair consists of a tokenized document and an integer label.

The code parses command-line arguments using argparse, specifying parameters such as hidden dimension, number of epochs, and file paths for training, validation, and test data. Random seeds are set for reproducibility and initialize the learning rate. Then it Loads and preprocesses data, including creating a vocabulary, mapping words to indices, and converting documents into vector representations. Initializes the FFNN model and an optimizer (SGD, ADAM).
The code then enters the training loop, where it trains the FFNN for the specified number of epochs. Within each epoch, it shuffles the training data, iterates through mini-batches of data, calculating the loss and performing backpropagation, and calculates and monitors training accuracy. After training, the code proceeds to validate the model on the validation data, calculating validation accuracy.

### 2.2 RNN

```
def forward(self, inputs):
    # [to fill] obtain hidden layer representation (https://pytorch.org/doc
    _, hidden = self.rnn(inputs,torch.zeros(self.numOfLayer, 1, self.h))

    # [to fill] obtain output layer representations
    output = hidden[:,-1,:]

    # [to fill] sum over output
    output1 = self.W(output)

    # [to fill] obtain probability dist.
    predicted_vector = self.softmax(output1)

    return predicted_vector
```

In the first step, a hidden state will be seeded as a matrix of zeros of dimensions (1, 1, h) where h is hidden dimension taken as input. The result of this will then be passed through an activation function ( a tanh function) to introduce non-linearity. Finally, a soft-max function is introduced to provide normalized probability distribution over the output.

Inside the training loop, data is shuffled, and the model is trained for multiple epochs. The training process includes preprocessing input text data, computing the forward pass through the RNN model. Calculating the loss and performing backpropagation. The training loop continues until the stopping condition is met (to avoid overfitting). The stopping condition for RNN is if validation accuracy of the current epoch is less than the validation accuracy of the last epoch and training accuracy of the current epoch is greater than the training accuracy of the last epoch training is done to avoid overfitting.

RNN uses the hyperbolic tangent (tanh) activation function in its recurrent layers whereas FFNN uses ReLU (Rectified Linear Unit) activation function in its hidden layers. In RNN input data is represented as sequences of word embeddings, and the model maintains memory of previous inputs. whereas in FFNN it is a vector where each dimension corresponds to a word or token in the vocabulary and uses a bag-of-words representation. In both RNN and FFNN training can be done using optimization algorithms like Adam or SGD.

## 3. Experiments and Results

### 3.1 Evaluations

Accuracy metric is used to evaluate the models.The number of properly predicted results over the total number of results gives the value of accuracy.

```
print("Validation accuracy for epoch {}: {}".format(epoch + 1, correct / total))
```

### 3.2 Results

**FFNN Results:** The following table shows the accuracy values of the FFNN model by varying the Hidden size, learning rate and the optimizer.

SGD Optimizer

| Hidden size | Learning rate | epochs | Accuracy |
|---|---|---|---|
| 10 | 0.01 | 10 | 0.566 |
| 20 | 0.01 | 10 | 0.576 |
| 10 | 0.1 | 10 | 0.40 |
| 20 | 0.1 | 10 | 0.411 |
| 10 | 0.001 | 10 | 0.586 |
| 20 | 0.001 | 10 | 0.587 |

ADAM Optimizer

| Hidden size | Learning rate | epochs | Accuracy |
|---|---|---|---|
| 10 | 0.01 | 10 | 0.586 |
| 20 | 0.01 | 10 | 0.600 |
| 10 | 0.1 | 10 | 0.407 |
| 20 | 0.1 | 10 | 0.464 |
| 10 | 0.001 | 10 | 0.611 |
| 20 | 0.001 | 10 | 0.605 |

**RNN Results:** The following table shows the accuracy values of the RNN model by varying the Hidden size, learning rate and the optimizers.

SGD Optimizer

| Hidden size | Learning rate | Accuracy |
|---|---|---|
| 10 | 0.01 | 0.456 |
| 20 | 0.01 | 0.438 |
| 10 | 0.1 | 0.385 |
| 20 | 0.1 | 0.423 |
| 10 | 0.001 | 0.44 |
| 20 | 0.001 | 0.445 |

ADAM Optimizer

| Hidden size | Learning rate | Accuracy |
|---|---|---|
| 10 | 0.01 | 0.457 |
| 20 | 0.01 | 0.437 |
| 10 | 0.1 | 0.427 |
| 20 | 0.1 | 0.403 |
| 10 | 0.001 | 0.47 |
| 20 | 0.001 | 0.46 |

### Observations and Analysis:

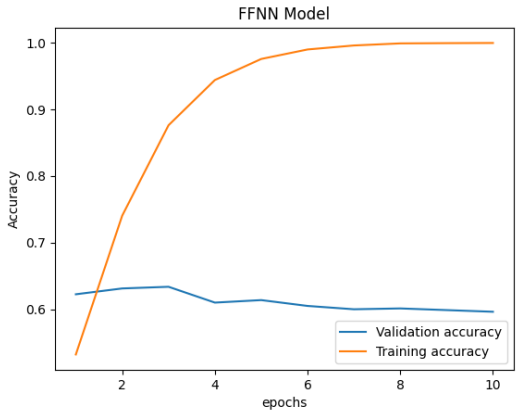
- Learning Rate:
    - The Accuracy of both the models increases as the learning rate decreases.
    - Large values of learning rates will cause the model to diverge and decrease the performance of the model. And very small values for learning rate will lead to slow convergence.
    - Accuracy generally increases as the learning rate is decreased till it reaches an optimal point.
- Hidden Dimension:

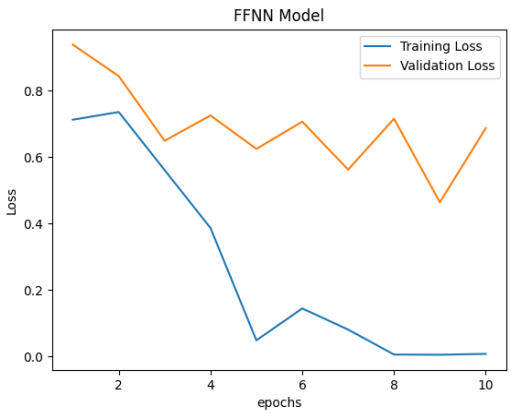
    - The Accuray of both the models increases when you increase the size of hidden dimensions.
    - With a small hidden dimension the model might not be able to capture the complex patterns in data and a larger value of hidden dimension might lead to overfitting.
    - Accuracy generally increases as the hidden dimensions increase until it reaches the point of overfitting.
- Optimizer:
    - The ADAM optimizer gives better results compared to the SGD optimizer.
    - ADAM optimizer performs slightly better than the SGD as it converges faster compared to SGD.
    - ADAM optimizer requires less manual hyperparameter tuning but requires additional memory.
    - SGD performs better if we are fine-tuning pre-trained models.
- Optimal values for these hyperparameters need to be selected by conducting multiple experiments on the dataset.

# 4. Analysis

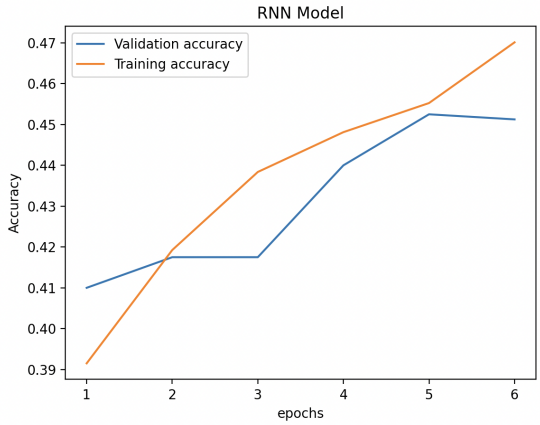
## 4.1 Plots

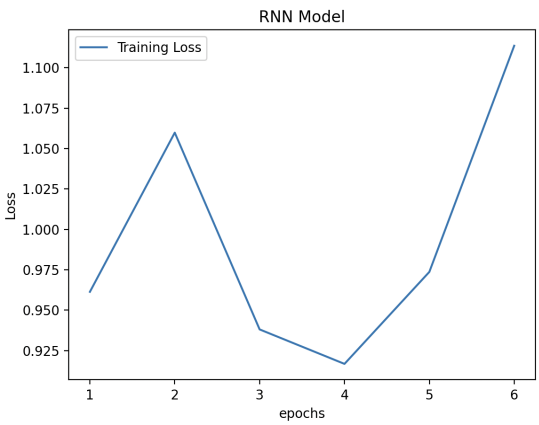
The FFNN model gives the highest accuracy of 0.611 with ADAM optimizer, hidden dimension 10 and learning rate 0.001. The below graphs show the learning curve of this FFNN model




The RNN model gives the highest accuracy of 0.47 with ADAM optimizer, hidden dimension 10 and learning rate 0.001. The below graphs show the learning curve of this RNN model



## 4.2 Error Analysis

FFNN error example

```
##################
predicted_label tensor(1)
gold label 2
input_vector tensor([ 0.,  0.,  0.,  ...,  0.,  0., 23.])
predicted_vector tensor([ -3.4386,  -0.4610,  -1.0870, -24.0397, -26.1464],
       grad_fn=<LogSoftmaxBackward0>)
##################
predicted_label tensor(0)
gold label 2
input_vector tensor([0., 0., 0.,  ..., 0., 0., 1.])
predicted_vector tensor([-1.0781, -1.1325, -1.1700, -4.0538, -4.6247],
       grad_fn=<LogSoftmaxBackward0>)
##################
predicted_label tensor(1)
gold label 2
input_vector tensor([0., 0., 0.,  ..., 0., 0., 1.])
predicted_vector tensor([-1.2316, -1.0400, -1.0894, -4.4873, -4.9569],
       grad_fn=<LogSoftmaxBackward0>)
##################
```

RNN error example

```
predicted_label tensor(0)
gold label 2
input_vector ['Overly', 'priced', 'they', 'do', 'not', 'clean', 'under', 'nails', 'Very
################
predicted_label tensor(1)
gold label 2
input_vector ['a', 'recent', 'weekend', 'dinner', 'found', 'the', 'place', 'almost', 'em
ed', 'room', 'after', 'we', 'had', 'perused', 'the', 'menu', 'for', 'awhile', 'made', 'o
was', 'out', 'of', 'and', 'what', 'the', 'specials', 'were', 'we', 'ordered', 'wine', 'w
she', 'looked', 'on', 'blankly', 'we', 'suggested', 'a', 'decanter', 'she', 'brought',
t', 'was', 'gone', 'was', 'good', 'there', 'were', 'bread', 'plates', 'on', 'the', 'tabl
 'while', 'my', 'tofu', 'could', 'have', 'had', 'more', 'flavor', 'the', 'dish', 'accomp
################
predicted_label tensor(1)
gold label 2
input_vector ['I', 'just', 'got', 'two', 'tires', 'replaced', 'here', 'today', 'I', 'dea
hey', 'promised', 'my', 'car', 'would', 'be', 'done', 'in', 'an', 'hour', 'and', 'they',
'have', 'a', 'warranty', 'on', 'their', 'tires', 'for', 'an', 'extra', '10', 'that', 'wi
', 'your', 'fault', 'up', 'to', '65000', 'miles', 'I', 'thought', 'that', 'was', 'intere
t', 'it', 'was', 'included', 'in', 'the', 'price', 'without', 'telling', 'me', 'All', 'i
heap', 'tire']
```

The above figures show the wrongly predicted examples using the FFNN and RNN model.

- Some of the reasons for the misclassification of examples might be insufficient and noisy data.
- We can improve the system by applying suitable hyperparameter tuning, choosing a better optimizer, implementing learning rate schedules like learning rate decay etc.
- Replace standard RNN cells with Long Short-Term Memory (LSTM) cells or Gated Recurrent Unit (GRU) cells. These architectures have better memory capabilities, helping the model capture long-range dependencies in the data.
- Adding more layers to RNN. Stacking multiple layers can enable the model to learn more complex patterns.

## 4.3 Other Analysis

On the Test data, the FFNN model gives an accuracy of 0.09 with ADAM optimizer, hidden dimension 10 and learning rate 0.001. The RNN model gives an accuracy of 0.010 with ADAM optimizer, hidden dimension 10 and learning rate 0.001. We think the accuracy can be improved by implementing the suggestions made above (in 4.2 Error Analysis).

# 5. Conclusion and Others

### 3.1 Libraries Installed

Numpy, torch, tqdm, matplotlib

### 3.2 Contribution of team members

Sree Vidya - RNN and report
Likhitha - FFNN and report

### 3.3 Feedback for the project

The Assignment was a good learning experience. It helped us understand how the implementation of the Neural networks is done for sentiment analysis and how the hyperparameters affect the performance of the model.