CS6320 Assignment 2

https://github.com/srikar-kotra/Feedforward-Neural-Network-FFNN-and-a-RNN

Group 9 | Shivanee Ramesh SXR230004 | Srikar Satya SXS230164 | Trishala Reddy TXR220017

1 Introduction and Data (5pt)

In this assignment, we conducted a 5-class sentiment analysis on Yelp reviews using PyTorch, employing both a Feed Forward Neural Network (FFNN) and a Recurrent Neural Network (RNN). The primary task is to classify Yelp reviews into five categories - [1, 2, 3, 4, 5] - utilizing both FFNN and RNN models. The dataset includes training, test, and validation sets in JSON format, with the number of examples in each set detailed in Table 1.

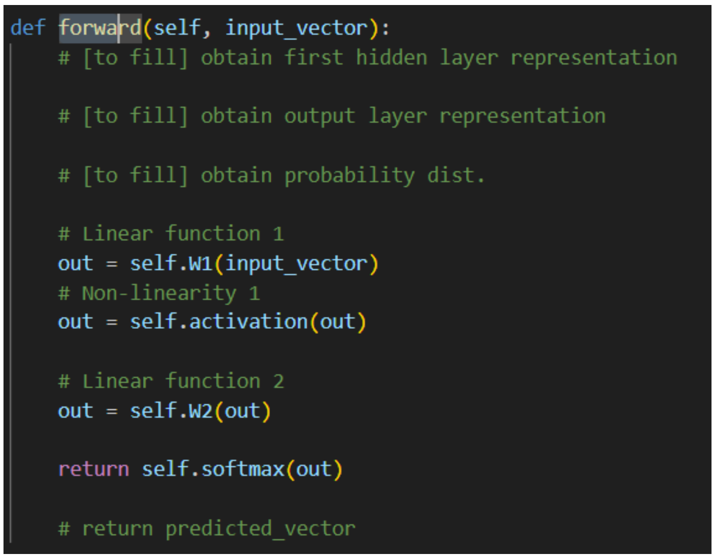
Data Type	No. of Examples
Training	16,000
Test	800
Validation	800

Table 1: Dataset statistics

During pre-processing, the text data is tokenized using whitespace as a delimiter and transformed into vector embeddings through a pre-trained word embedding file. All text is converted to lowercase, and the data is shuffled at each epoch.

2 Implementations (45pt)

2.1 FFNN (20pt)



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

    # [to fill] obtain output layer representation

    # [to fill] obtain probability dist.

    # Linear function 1
    out = self.W1(input_vector)
    # Non-linearity 1
    out = self.activation(out)

    # Linear function 2
    out = self.W2(out)

    return self.softmax(out)

    # return predicted_vector
```

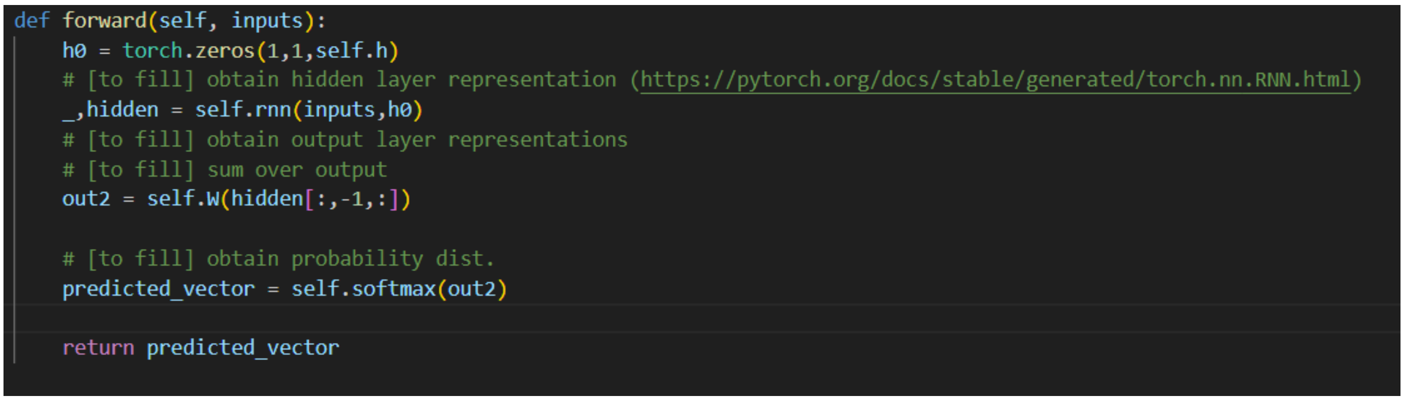
Figure 1: FFNN Code

The Feedforward Neural Network (FFNN) implemented here takes in a vectorized representation of text data and processes it to classify inputs into one of five categories. The FFNN takes in a fixed amount of input data all at the same time and produces a fixed amount of output at the same time. The network is built using PyTorch and contains an input layer, one hidden layer, and an output layer. In the forward pass, the input is multiplied by the weights of the hidden layer, and the result is passed through a ReLU activation function to introduce non-linearity. This output is then fed to the final layer, where it is again transformed by another set of weights. Finally, a Softmax function is applied to produce a normalized probability distribution over the possible classes.

The model is trained using Stochastic Gradient Descent (SGD) with momentum for more stable updates. The training loop iterates over the specified number of epochs, and in each epoch, data is processed in mini-batches for efficient computation. After each mini-batch, gradients are calculated, and the optimizer updates the weights to minimize the negative log-likelihood loss. Validation is performed after each epoch to evaluate the model's performance on unseen data, providing training and validation accuracy at each step.

Additionally, seed setting ensures reproducibility, and data is shuffled before each epoch to avoid any order biases. The results of the training, including training loss and both training and validation accuracy, are plotted using Matplotlib. This setup enables us to visualize the model's performance over epochs and observe trends in learning progress.

2.2 RNN (25pt)



```python
def forward(self, inputs):
    h0 = torch.zeros(1,1,self.h)
    # [to fill] obtain hidden layer representation (https://pytorch.org/docs/stable/generated/torch.nn.RNN.html)
    _,hidden = self.rnn(inputs,h0)
    # [to fill] obtain output layer representations
    # [to fill] sum over output
    out2 = self.W(hidden[:,-1,:])

    # [to fill] obtain probability dist.
    predicted_vector = self.softmax(out2)

    return predicted_vector
```

Figure 2: RNN Code

The RNN processes input sequences one vector at a time in sequence, updating its hidden state with each step. The hidden state output is combined with the following input in sequence to produce the following output. For sentiment analysis, the model learns to predict the review rating by sequentially processing words in the review text. The final hidden state, which has integrated information from all prior words, provides an informed prediction based on the entire sequence. In this setup, we are performing sentiment analysis to predict a review's rating, which can be derived from the final cell of the RNN. This is because the last output incorporates information from all prior computations and inputs. Initially, the hidden state is typically initialized as a zero matrix with dimensions (1, 1, h), where h represents the specified hidden dimension.

During each step, the RNN computes a new hidden state using an activation function, such as tanh, which introduces non-linearity and enables the model to capture complex patterns in the data. After processing the full sequence, the model passes the final hidden state through a softmax layer, which transforms it into a normalized probability distribution across possible ratings. This enables the RNN to predict the most likely rating for the input review based on the learned context.

3 Experiments and Results (45pt)

3.1 Evaluations (15pt)

Accuracy is used as the metric for the model evaluation which is a measure of the number of correctly predicted results over the total number of results.

3.2 Results (30pt)

Table 2 and Table 3 show the accuracy for different hyper parameters that we tested our implementation with for FFNN and RNN respectively.

Hidden Size	Optimizer	Epochs	Accuracy
50	SGD	5	0.6
100	SGD	5	0.65
10	SGD	5	0.65
32	SGD	5	0.56
64	SGD	3	0.56
32	Adam	3	0.86

Table 2: FFNN: Validation Accuracy with different parameters (0.01 learning rate)

Hidden Size	Optimizer	Epochs	Accuracy
32	Adam	5	0.375
50	Adam	5	0.316
100	Adam	2	0.335
10	SGD	3	0.36
32	SGD	3	0.37375
50	Adam	1	0.3375

Table 3: RNN: Validation Accuracy with different parameters (learning rate : 0.01)

4 Analysis (bonus: 5pt)

4.1 Plot the learning curve of your best system.

For the learning curve plots given below for our best system, the blue coloured lines represent the training accuracy and the orange lines represent the validation accuracy.

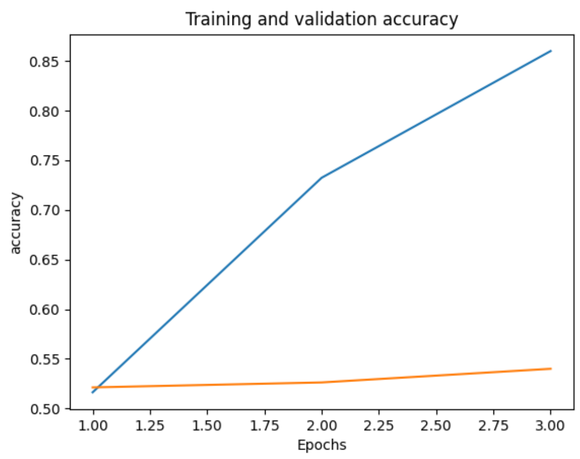


Figure 3: FFNN Training and Validation Accuracy

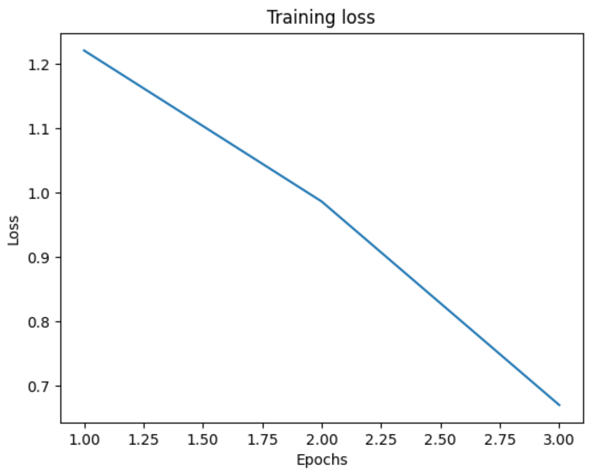


Figure 4: FFNN Training Loss

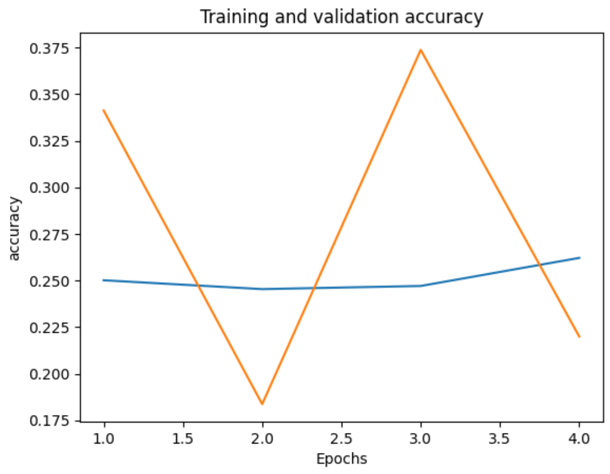


Figure 5: RNN Training and Validation Accuracy

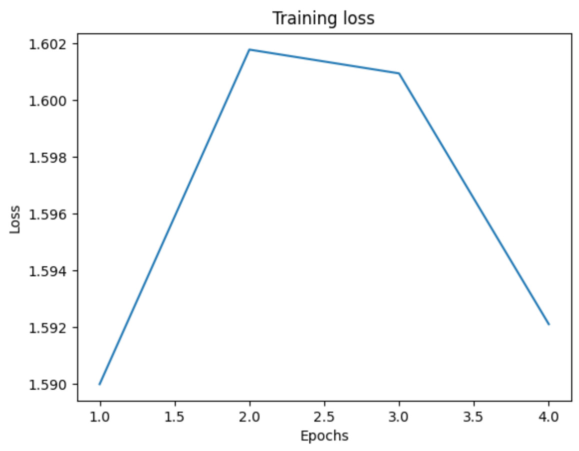


Figure 6: RNN Training Loss

4.2 Error analysis

The FFNN yielded varying results with a learning rate of 0.01 across different configurations. Using the SGD optimizer, a hidden size of 100 achieved the highest validation accuracy of 0.65 after 5 epochs, while smaller hidden sizes, such as 32 and 64, resulted in lower accuracies around 0.56. The model showed a substantial improvement when using the Adam optimizer, with a hidden size of 32 reaching a validation accuracy of 0.86 after 3 epochs, suggesting that Adam was more effective in optimizing the FFNN at this learning rate.

The RNN, with a consistent learning rate of 0.01, displayed varied accuracy levels across configurations, showing lower overall performance than the FFNN. The highest validation accuracy achieved by the RNN was 0.375 with a hidden size of 32 using Adam over 5 epochs. Configurations with SGD showed similar accuracy but with more stability across epochs. The differences in performance between the optimizers suggest that the RNN may benefit from further tuning to reach consistent accuracy improvements, particularly with the Adam optimizer.

5 Conclusion and Others (5pt)

In this assignment, we successfully implemented sentiment analysis using both FFNN and RNN models, experimenting with various hyperparameter configurations for analysis and comparison.

The highest accuracy achieved was 0.86 for FFNN and 0.37375 for RNN.

5.1 Individual member contribution:

While the entire team collaborated on the initial planning and workflow design, once the layout was finalized, we divided the implementation tasks equally among team members as follows:

- Trishala: RNN and report.
- Srikar: Overall improvement, analysis, and report.
- Shivanee: FFNN and report.

5.2 Assignment Feedback

Completing this assignment was a positive experience. However, we felt that the coding component was limited, and more hands-on coding could enhance the learning process. With this, we could explore different neural models and learnt the comparison quickly.