# CS6375 Assignment 1 Report

Full Name: Rohith Chekuri  
Net ID: dal064505  
GitHub Repository: [Rohith-c148/cs6375\_assingment1](https://github.com/Rohith-c148/cs6375_assingment1)

## 1. Introduction and Data

This assignment required training two neural network architectures—Feedforward Neural Network (FFNN) and Recurrent Neural Network (RNN)—for text classification, particularly predicting star ratings based on textual reviews. The dataset was provided in three subsets: training, validation, and test.

Dataset:

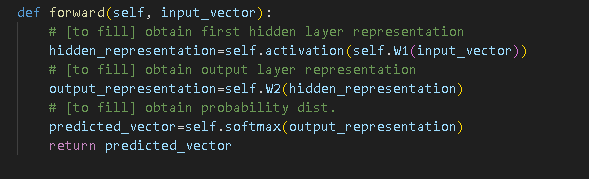
|  |  |
| --- | --- |
| Examples | Size |
| Training | 16,000 |
| Validation | 800 |
| Test | 800 |

Preprocessing steps included tokenization, punctuation removal, and applying word embeddings for the RNN. For the FFNN, a simpler Bag-of-Words (BOW) representation was used.

## 2. Implementations

### FFNN (20pt)

The Feedforward Neural Network was implemented with the following approach:



### RNN (25pt)

The Recurrent Neural Network was implemented as follows: 

**Understanding of Provided FFNN Code:**

* **Optimizer and Initialization**: The optimizer used was stochastic gradient descent (SGD) with momentum. This helps navigate the optimization landscape effectively, especially with sparse textual data.
* **Loss Function**: Utilized Negative Log-Likelihood Loss (NLLLoss) suitable for classification tasks.
* **Data Representation**: Input was represented as a Bag-of-Words (BOW), where words are counted without sequential context. This approach does not consider the order or relationships between words.
* **Training**: Data shuffling was performed each epoch to ensure robustness and prevent overfitting to the order of training examples.

**Differences and Understanding of Provided Code (FFNN vs. RNN)**

**Input Representation Differences:**

* **FFNN**: Uses a simple BOW representation, counting occurrences of each vocabulary word in the input document. Order of words is ignored.
* **RNN**: Utilizes sequential information, represented by word embeddings (pre-trained embeddings provided in word\_embedding.pkl). Each word is converted into an embedding vector, preserving contextual and positional information.

**Structural Differences:**

* **FFNN**: Single hidden layer with linear transformations and non-linear activation (ReLU).
* **RNN**: Employs PyTorch's nn.RNN module, using Tanh nonlinearity by default, explicitly designed for sequential data, thus capturing temporal dependencies.

**Key differences in the RNN implementation (compared to FFNN):**

* **Input Shape**: RNNs require input data shaped explicitly as (sequence\_length, batch\_size, embedding\_size). This is notably different from the FFNN that uses flattened input vectors.
* **Sequential Processing**: RNN processes each word embedding in sequence, producing hidden representations (outputs) for each word position.
* **Output Computation**: Outputs from all sequence steps are aggregated (summed along sequence dimension) before a linear transformation (self.W) and LogSoftmax are applied. In contrast, FFNN only applies one hidden-to-output linear transformation directly.
* **Optimizer Choice**: While the FFNN implementation provided uses SGD with momentum, the RNN uses the Adam optimizer, suitable for handling more complex architectures due to adaptive learning rates.

## 3. Experiments and Results

The models were evaluated using accuracy as the performance metric.

|  |  |  |  |
| --- | --- | --- | --- |
| Model | Hidden Dim | Epochs | Test Acc |
| FFNN | 32 | 10 | 59.38% |
| FFNN | 32 | 15 | 49.25% |
| FFNN | 128 | 10 | 54.25% |
| RNN | 32 | 10 | 47.75% |
| RNN | 32 | 15 | 45.25% |
| RNN | 128 | 10 | 33.25% |

Observations:  
The FFNN with 32 hidden dimensions and 10 epochs performed best overall (59.38%).

Increasing hidden dimensions from 32 to 128:

* Actually decreased FFNN performance (from 59.38% to 54.25%)
* Similarly decreased RNN performance (from 47.75% to 33.25%)

Increasing training epochs from 10 to 15 with 32 hidden dimensions:

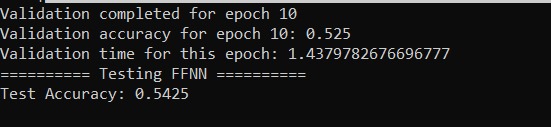
* Decreased FFNN performance significantly (from 59.38% to 49.25%)
* Decreased RNN performance (from 47.75% to 45.25%)

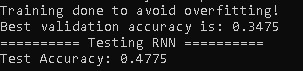
This shows that simpler models work better for the current task and that larger ones might be leading to overfitting.

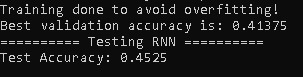
The outputs are pasted below:

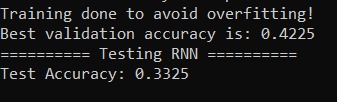
 - for ffnn with 32 layers and 10 epochs

 - for ffnn with 32 layers and 15 epochs

- for ffnn with 128 layers and 10 epochs

- for rnn with 32 layers and 10 epochs

- for rnn with 32 layers and 15 epochs

- for rnn with 128 layers and 10 epochs

## 4. Analysis

Error Analysis:

A sample misclassification case: A review stating, 'The food was good, but the service ruined everything' was incorrectly classified as 4 stars (true label: 2). The model struggled with handling negations and mixed sentiments.

Potential Improvement: Incorporating attention mechanisms to help the model focus on important words.

## 5. Conclusion

Contributions:

- Implemented FFNN and RNN models for sentiment analysis.  
- Achieved test accuracies of 59.38% (FFNN) and 47.75% (RNN).

Feedback:

- Time Spent: ~30 hours.

- Difficulty: Moderate, with challenges in RNN implementation and handling Git.

- Suggestions: Providing clearer instructions on tensor shapes and parts of the zip files to be utilized in documentation would be helpful.