CS6375 Assignment 1

https://github.com/as567-code/CS6375_Assignment_1
Aditya Swaroop axs230571

March 16, 2025

1 Introduction and Data

This project focuses on performing sentiment analysis using two neural network architectures: a Feedforward Neural Network (FFNN) and a Recurrent Neural Network (RNN). The goal is to predict Yelp review ratings (ranging from 1 to 5 stars) based on the review text. To achieve this, I used two models that
process the text data and aim to classify the sentiment accurately into one of the five classes. The data consists of Yelp reviews that are labeled by their star rating. The dataset is divided into three sets:

- Training Data
- Validation Data
- Test Data

Wrote a small code/file called **"Length of Data.ipynb** to find the length of the datasets. Below is the summary of the statistics of the datasets.

Table 1: Data Statistics

2 Implementations

2.1 FFNN

Forward Function: This function shows how the data flows through the network.

- **hidden layer = self.activation(self.W1(input vector))**: The input vector is passed through the first linear layer, producing a hidden layer representation.
-

output layer = self.W2(hidden layer): The hidden is the passed through a second linear layer, producing the output layer representation.

- **predicted vector = self.softmax(output layer)**: The predicted out- put is passed through a LogSoftMax function, which converts the score into log probabilities.

Libraries/Tools used:

- **nn.Module:** Base class for all neural networks.
- **nn.Linear:** A linear transformation layer, which is fully connected.
- **nn.ReLU:** This is the Rectified Linear Unit, which is an activation func- tion. This introduces non linearity to the neural network.
- **nn.LogSoftMax:** Computes log probabilities.
- **nn.NLLLoss:** The negative log likelihood loss function for classification tasks.

Optimizers and Initializations:

- **Optimizer:**The code uses Stochastic Gradient Descent (SGD) with momentum for updating the model weights during training.
- **Weight Initialization:** self.W1 and self.W2 are the weight initializa- tions. Pytorch uses a uniform distribution to initialize weight and biases of the linear layers.

Figure 1: FFNN forward function Added a

small piece of code to check the testing accuracy.

Figure 2: RNN forward function

2.2 RNN

In the forward function, I write output sum = output[-1]. This is the hidden state corresponding to the last token in the sequence. In this case, instead of using a command like output.sum() to sum all the hidden states, we rather only consider the last token. In this case, the last state carries information from all the previous tokens, thus making it better than using the output.sum() command. Libraries/Tools used:

- **nn.Module:** Base class for all neural networks.
- **nn.RNN:** This initializes the RNN and the parameters define the struc- ture of the RNN.
- The other libraries are similar to that of FFNN.
- RNN processes inputs as sequences and maintains the hidden states, whereas the FFNN processes inputs independently.
- RNN works in a way such that the next output is dependent on the pre- vious input.

3 Experiments and Results

3.1 Evaluations

1. Evaluation Process: During the training phase, the model is evaluated at the end of each epoch. This involves both calculating the training accuracy and validating the model against a separate validation dataset. I can
observe this in the results.

2. Metrics used:

   - The primary metric used for evaluating the model is accuracy, which measures the proportion of correctly predicted instances out of the total instances evaluated.
   - Although accuracy is the main metric, the implementation also tracks the loss during training and validation. The loss function used is Neg- ative Log Likelihood Loss (NLLLoss), which is particularly suitable

   for multi-class classification tasks. The loss quantifies how well the model's predicted probabilities align with the actual labels, providing insight into the model's performance beyond just accuracy.

3. During training, the model iterates through the training dataset in mini-batches, predicting the class labels for each input vector. The predicted labels are compared with the actual labels to calculate the number of correct predictions and compute the accuracy for that epoch. The same procedure is done for the validation dataset.

3.2 Results

Results of ffnn.py:

- Taking 2 hidden dimensions and 2 epochs:

Figure 3: FFNN with 2 hidden dimensions and 2 epochs

- Taking 64 hidden dimensions and 20 epochs:

Figure 4: FFNN with 64 hidden dimensions and 20 epochs Results of

rnn.py:

Figure 5: Accuracy for RNN with 2 hidden dimensions and 2 epochs

- Taking 2 hidden dimensions and 2 epochs
- Taking 64 hidden dimensions and 20 epochs

Figure 6: Initial accuracy for RNN

Figure 7: Final accuracy for RNN

4 Analysis

Some error examples are:

- ”Bartender needs an attitude adjustment”. True label is 1, predicted label is 3. The model couldn’t understand the sarcasm.
- ”Oh great, another 45-minute wait for my cold food. Just what I wanted.” True label is 1, predicted label is 4. The model couldnt understand the sarcasm
- ”I have no problem paying for service when I receive value for it, but when it is just gouging, that’s not worth it.” True label is 2, Predicted label is 4, the model couldn’t understand the subtle negativity at the end of the sentence.

5 Conclusion and Others

Feedback for the assignment:

- The assignment took around 20 hours, mostly debugging the results.
- The RNN was more difficult compared to the FFNN, especially when fine-tuning the model. Handling the data structure and optimizing performance took more time than expected.
- The assignment could include how to upload large files into github, al- though it was easily available online(using git lfs install).