CS6320 Assignment 2

https://github.com/phsaikiran/CS6320-Fall23-Group30

Group30 Hima Sai Kiran Prudhivi HXP220011 Abhilash Rajesh Bagalkoti AXB220071

1 Introduction and Data

The project involves building and evaluating two different neural network models for a 5-class Sentiment Analysis task. The two models are a Feed Forward Neural Network (FFNN) and a Recurrent Neural Network (RNN). The task is to predict the sentiment rating (y) for Yelp reviews, where y can take values in the set {1, 2, 3, 4, 5}.

Dataset	# of examples
Training	16000
Validation	800
Test	800

Table 1: Number of examples

We will compare multiple variations of hyper-parameters based on accuracy. We will briefly explain the implementation, analyze the findings and report the results.

2 Implementations

2.1 FFNN and RNN

Libraries We used `mlflow` library to experiment with different hyper-parameters. MLflow facilitates tracking and organizing various aspects of this ML project, including parameters, metrics, and models.

Extra code We added extra code to calculate test accuracy using the test set and print out a few wrongly predicted examples. We also added code for saving the model for best validation accuracy.

Modifying implementation For RNN, we modified the implementation to train using multiple hidden layers. We calculated the test accuracy for various hidden layer configurations.

Experimenting with parameters We experimented with `ADAM` and `SGD` optimizers, `ReLU` and `Softmax` activation functions and multiple values for learning rate, minibatch size and hidden layer dimensions.

Debugging We used IntelliJ IDE and VSCode to implement and debugged the code by adding debug points.

2.2 FFNN

The `forward()` function calculates the FFNN's prediction and consists of the following steps:

1. Calculate the hidden layer representation by applying a linear transformation and an activation function to the input vector.

```python
def forward(self, input_vector):
    # Obtain first hidden layer representation
    hidden = self.W1(input_vector)
    hidden = self.activation(hidden)
    # Obtain output layer representation
    output = self.W2(hidden)
    # Obtain probability dist.
    predicted_vector = self.softmax(output)
    return predicted_vector
```

Listing 1: forward() function for FFNN

2. Obtain the output layer representation by applying another linear transformation to the hidden layer representation.
3. Compute the probability distribution over the classes by applying the softmax function to the output.
4. Return the predicted vector representing class probabilities.

The implementation parses command-line arguments to configure the training process, such as hidden dimension, number of epochs, and data paths. Loads the training, validation, and test data. Generates a vocabulary, word-to-index, and index-to-word mappings. Iterates through hyperparameters (learning rate, momentum, minibatch size) for model training and logs results using MLflow. Selects the best model based on validation accuracy. Logs the best model to MLflow and computes the test accuracy.

2.3 RNN

```python
def forward(self, inputs):
    # Obtain hidden layer representation
    _, hidden = self.rnn(inputs)
    # Obtain output layer representations
    output = self.W(hidden)
    # Sum over output
    summed_output = torch.sum(output, dim=0)
    # Obtain probability dist.
    predicted_vector = self.softmax(summed_output)
    return predicted_vector
```

Listing 2: forward() function for RNN

1. **RNN forward function** The RNN model processes the input sequence to extract a hidden state, which is then linearly transformed and summed over time steps to create a single representation. This representation is passed through a softmax function to produce a probability distribution over classes, serving as the model's prediction for the input sequence.
2. **Early Stopping**: The training process includes early stopping to prevent overfitting.
3. **Testing**: If provided, the code can load a pre-trained model and evaluate its performance on the test data.
4. **Pretrained Word Embeddings**: The code incorporates word embeddings loaded from a pickle file for text representation.

5. **Logging**: The code uses MLflow for tracking and logging training and validation metrics and model hyperparameters.

2.4 FFNN vs RNN

Aspect	RNN	FFNN
Architecture	Designed for sequential data, uses recurrent layers to capture temporal dependencies.	Processes data in a feedforward manner, not designed for sequential data.
Data Handling	Processes data sequentially, updating hidden states at each time step to understand context and order.	Treats data as a static bag of words, ignoring word order.
Word Embeddings	Utilizes word embeddings like pre-trained vectors to understand word meanings in context.	Uses one-hot encoding for words, lacking inherent word meaning understanding.

3 Experiments and Results

Evaluations We used accuracy to evaluate the FFNN and RNN models.
$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions}$

Results We compared multiple variations in the optimizer, activation function, hidden layer dimensions, learning rate, momentum and minibatch size. We reported the one which gave a better accuracy, as shown in the table 2. Once the parameters were finalized, various hidden dimensions were tested as shown in table 2.

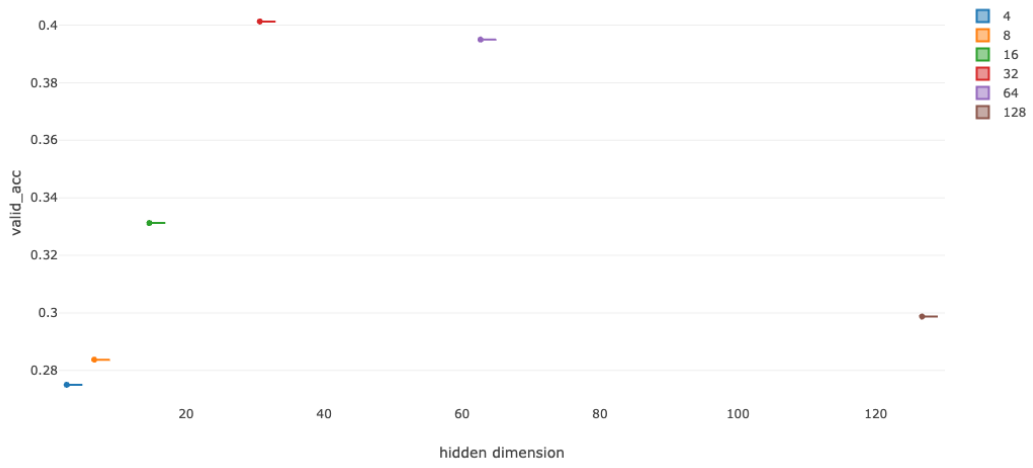

Figure 1: Hidden dimension vs validation accuracy for RNN

The final test accuracies for FFNN and RNN are as follows. The hyperparameters chosen are present in table 2.

FFNN Hidden layers: 16 and 5 Epochs: **53.375%**

RNN Hidden layers: 32 and 8 Epochs: **31.875%**

Bonus We also changed the default code for RNN to accommodate multiple hidden layers and the results are shown in table 3 and Plot 1.

Parameter	Variations	FFNN	RNN
Optimizer	`SGD`, `ADAM`	`SGD`	`ADAM`
Activation function	`ReLU`, `Softmax`	`ReLU`	`ReLU`
Learning rate	[0.01, 0.005, 0.0025,0.0015,0.0010,0.0005,0.0001]	0.01	0.0015
Momentum	[0.9, 0.99]	0.9	-
Minibatch size	[2, 4, 8, 16, 32, 64]	32	8
Hidden layer dimensions	[4, 8, 16, 32, 64, 128]	16	32

Table 2: Parameter variations

Layer	Test accuracy
1	31.875%
2	36.375%
3	30.875%
4	31.05%

Table 3: Multiple hidden layers for RNN

4 Analysis

Train and validation plots We got the best results for the hidden layer dimension of 16 for FFNN and 32 for RNN as shown. We plotted training and validation curves in the Plots 2 and 3. We also plotted the training loss in the Plots 2 and 3.

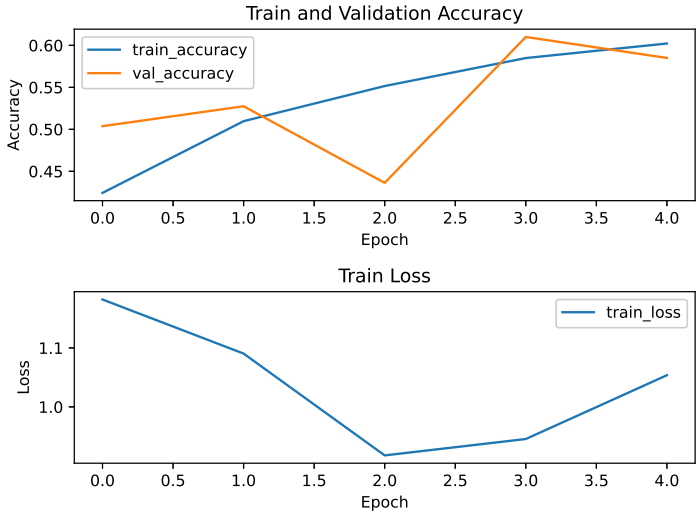

Figure 2: FFNN Train and validation accuracy, Training loss

Error analysis Consider the following wrong prediction:

```
"Stumbled into this place, and nooooo!! fellow yelpers, I was not
drunk. I would think I was after paying for this meal, talk about still
being hungry. A simple chicken single taco with only a few bits of
chicken and I think I piece of onion was $5.50, and that's because it
was on the special menu. My har gow, which is one of my favorite dish
was not worth $16.00, I should have walked over to Earls Sandwich,
which was my first choice, but me being a foodie/foodsnob, trying Jose
Andres restaurant sounded like a good idea. Oh wells, I ended up having
CPK at the airport for a filling BBQ salad. Since I had a 3 hour
delay!! Augh!!"
```

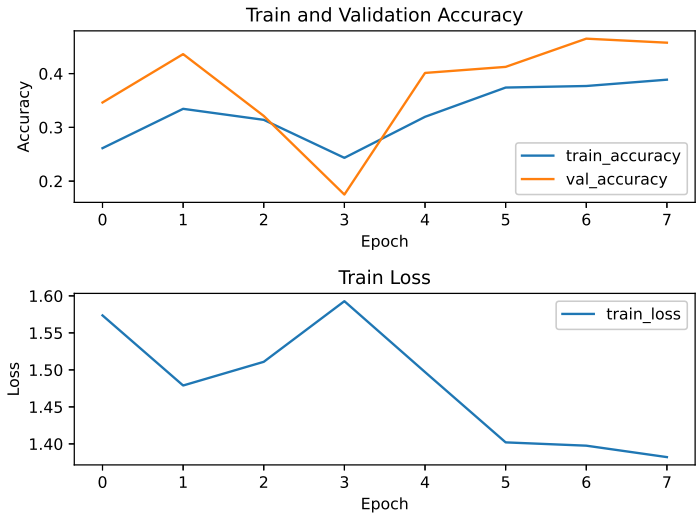

Figure 3: RNN Train and validation accuracy, Training loss

The correct value is 2 but it was predicted as 0. The prediction likely went wrong because of the complex and lengthy nature of the text. Longer sentences often introduce more ambiguity and complexity, making it challenging for models to correctly classify them. It's possible that the model struggled to capture the correct sentiment or meaning within the sentence, leading to the wrong prediction. Additionally, the presence of unusual or informal words like "nooooo!!" and complex sentence structures may have added to the difficulty in prediction.
To improve the performance, we can:

- Improve text preprocessing by removing noise words, such as "mmmmMmm," and lowercasing the entire text.
- Consider increasing the number of layers in your model architecture to capture more complex patterns.
- Ensure data consistency across different sets of labels to maintain the quality of training and validation data.
- When working with small hidden layer sizes, be cautious with long sentences, as they might lead to incorrect predictions.
- Increase the size of the training dataset to provide more diverse examples. Train for more epochs to improve model performance, ensuring it doesn't overfit.
- Changing the number of feed-forward NN layers.

5 Conclusion and Others

Our experiments and analyses shed light on the performance of FFNN and RNN models in sentiment analysis. We systematically explored a range of hyperparameters, allowing us to identify optimal settings for both models. We showcased the significance of hidden layer dimensions, demonstrating that the choice of this parameter significantly impacted model accuracy. We discuss the challenges in sentiment analysis when dealing with complex and lengthy texts. We provided recommendations for improving model performance in this context.
Contributions:
Team member 1: Hima Sai Kiran Prudhivi: Implementation and reporting for FFNN
Team member 2: Abhilash Rajesh Bagalkoti: Implementation and reporting for RNN
Difficulty Level: The project's difficulty level was just right for our current level of understanding and skills.
Time Spent: Our team spent approximately 25-30 hours collectively on the project over 2 weeks.