CS4375 Assignment 2

<https://github.com/rtv416/ML4375HW2>

Rachel Vincent

RTV200000

1 Introduction and Data (5pt)

The purpose of this project is to classify Yelp reviews from 1 to 5 stars using a Feedforward Neural Network and a Recurrent Neural Network. Both models are trained to predict the star rating based on the review text content. The dataset is divided into training, validation, and test sets, shown in the table below. The task that was to be done was a multi-class classification where the model predicts a rating based on the content of the review.

TABLE

DataSet	#
Training Model	Around 800
Testing Model	Around 800
Validation Model	Around 800

2 Implementations (45pt)

2.1 FFNN (20pt)

```python
def forward(self, input_vector):
    # Obtain first hidden layer representation
    h = self.activation(self.W1(input_vector))
    # Obtain output layer representation
    z = self.W2(h)
    # Ensure z is 2D by adding a batch dimension if necessary
    if z.dim() == 1:  # Check if z is 1D
        z = z.unsqueeze(0)  # Add a batch dimension
    # Apply softmax along the last dimension (dim=-1)
    predicted_vector = self.softmax(z)
    return predicted_vector
```

In the forward() function of ffnn.py, the input vector is transformed through a hidden layer using a fully connected layer followed by a ReLU activation function. The hidden representation is passed through an output layer to produce class scores. To ensure compatibility for the softmax operation, an additional dimension is added if the output is 1D. Finally, the scores are converted into probabilities using LogSoftmax, which allows the model to interpret outputs as class probabilities. I also used PyTorch and a Negative Likelihood Loss Function.

2.2 RNN (25pt)

```python
def forward(self, inputs):
    # Obtain hidden layer representation from the RNN layer
    _, hidden = self.rnn(inputs)
    # Pass the hidden state through the output layer to get scores
    z = self.W(hidden[-1])
    # Sum over output for the final vector and apply softmax
    predicted_vector = self.softmax(z)
    return predicted_vector
```

In the forward() function of rnn.py, the input sequence is processed by an RNN Layer, which produces hidden states at each time step. The final hidden state is used to make a summary of the input sequence, thus capturing information from the entire sequence. This final hidden state is passed through a fully connected layer to generate class scores which are converted into probabilities using LogSoftmax. I used Pytorch to provide RNN layers, linear transformations, and softmax operations, and used an ADAM optimizer for better performance, as well as a loss function (NLLLoss).

3 Experiments and Results (45pt)

3.1 Evaluations (15pt)

The models are evaluated based on their accuracy, which measures the proportion of correctly predicted labels out of the total examples. During training, both training and validation accuracies are tracked across epochs to assess how well the models learn from the data and generalize to unseen examples. Accuracy is calculated by comparing the predicted class with the actual label for each example, with separate metrics reported for the training and validation datasets. These metrics provide insight into the models' performance, allowing for comparisons across different hyperparameter configurations and helping to identify overfitting or underfitting tendencies.

3.2 Results (30pt)

The performance of the FFNN and RNN models was evaluated based on training and validation accuracy over multiple epochs, with results presented in tables and diagrams for clarity. The FFNN exhibited higher training accuray but showed limited improvement in validation accuracy, indicating potential overfitting with larger hidden layers. In contrast, the RNN maintained consistent validation accuracy, showing its stability when processing sequential data. These results demonstrate that while both models effectively learn patterns in the training data, there are different strengths and limitations of how each model handles the tasks.

EPOCH	Training Accuracy (FFNN)	Validation Accuracy (FFNN)	Training Accuracy (RNN)	Validation Accuracy (RNN)
1	0.60125	0.59025	0.3967	0.42875
2	0.888275	0.596225	0.39	0.4025
3	0.889775	0.585575	0.39175	0.4325
4	0.883775	0.5835	0.388575	0.40125
5	0.888775	0.584075	0.4025	0.40125
6	0.888075	0.566075	0.39975	0.40125
7	0.888275	0.567025	0.40125	0.40125
8	0.889675	0.583525	0.405	0.40125
9	0.887575	0.584513	0.393625	0.40125
10	0.878125	0.58785	0.4065	0.40125

Conclusion and Others (5pt)

For this assignment, I completed all tasks independently, including implementing and evaluating both FFNN and RNN models and analyzing results. While the assignment was slightly on the more difficult side, the primary challenge was a slight misunderstanding of the general direction on where to go at the start, which required extra effort to understand requirements and how to proceed. However, the tasks were manageable. Overall the assignment was a good learning experience for neural network models.