CS4395 Assignment 2

https://github.com/justi-lai/CS_4395_Personal

Justin Lai
JXL210110

1 Introduction and Data (5pt)

This project is a comparison of the effectiveness of feed-forward and recurrent neural networks on a five-class sentiment analysis task over Yelp reviews. The goal of each model is to predict the star-rating of a given Yelp review.

The data was provided in a train, validation, and test set, with each being shuffled as input to the model with a total of 16000, 800, and 800 valid inputs for training, validation, and testing.

What was observed in the end was a testing accuracy in favor of the feed-forward network. The predicted main factor is based on the nature of the task. For five class sentiment analysis, it may be better to base the rating off the density of positive and negative words, which is represented well by the FFNN. On the other hand, the order of words may not matter as much, so the RNN may not use the recurrent nature well and may in fact be a detriment due to a vanishing or exploding gradient.

2 Implementations (45pt)

2.1 FFNN (20pt)

The vector representation of the input to the model is the count of each word in the vocabulary. This fixes the input length to be a consistent number, allowing for a feed-forward network to take in a variable length input.

The feed-forward network passes this input to a linear layer that maps the input dimension to the hidden dimension and passes that through a Rectified Linear Unit. This is then passed to the output layer where it maps the hidden layer output to an output layer of size five, which goes through a softmax function to get the probability distribution.

```
def forward(self, input_vector):
    # [to fill] obtain first hidden layer representation
    hidden_layer = self.W1(input_vector)
    hidden_layer = self.activation(hidden_layer)

    # [to fill] obtain output layer representation
    output_layer = self.W2(hidden_layer)

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

    return predicted_vector
```

This output is evaluated using a negative log-likelihood loss and is optimized using standard gradient descent with a learning rate of 0.01. For this project, the feed-forward network is only trained through the entire training data set once before being evaluated. The batch size is 16 for each input.

2.2 RNN (25pt)

The Recurrent Neural Network represents each document as a list of word embeddings (the word embeddings are provided, and the dimension is unknown) before passing it to the model. This is done to make use of the recurrent structure of the model, which allows variable input length, as

opposed to the feed-forward network from before, which used a vocabulary count to represent each document. This list of embeddings is passed to the recurrent layer (with a tanh activation function instead of a ReLU), which creates a hidden layer at every "time stamp" that is passed through a linear layer that combines the hidden layer dimension to just five for the final output. Lastly, every time stamp's output is summed and then softmaxed to predict the star rating based on all hidden layers of the input.

```
def forward(self, inputs):
    # [to fill] obtain hidden layer representation (https://pytorch.org/docs/stable/generated/torch.nn.RNN.html)
    output, hidden = self.rnn(inputs)

    # [to fill] obtain output layer representations
    output_layer = self.W(output)

    # [to fill] sum over output
    output_layer = torch.sum(output_layer, dim=0)

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

    return predicted_vector
```

The input, validation, and testing sets are the same and still shuffled before every epoch. The loss function is still the NLLLoss. However, the optimizer is changed from standard gradient descent to the Adam optimizing function. This time, the model is set to run until overfitting is detected. Overfitting is detected by finding whether the previous validation accuracy is higher than the current, and the current training accuracy is higher than the previous. Each pass where the validation accuracy is higher than the previous, the model is saved (implemented separately from the provided code) and therefore, the highest validation accuracy weights are loaded when overfitting is detected.

3 Experiments and Results (45pt)

Evaluations (15pt)

Each model was tested using a shuffled testing set that was never seen before by the models. To evaluate a model's accuracy, each input from the testing set is input individually, and the accuracy of the model is the *total correct / total inputs*.

Results (30pt)

Model (Hidden Size)	***Accuracy***
FNN (5)	31.38%
FFNN (10)	52.88%
FFNN (16)	40.25%
FFNN (32)	46.13%
RNN (8)	26.75%
RNN (16)	39.13%

RNN (32)	37.25%
RNN (64)	11.88%
RNN (128)	14.38%

Overall, the performance of the feed-forward networks heavily outperformed the recurrent neural networks.

4 Analysis (bonus: 1pt)

In theory, the RNN should be able to understand the nature of the text in a deeper manner. The possible reasons as for why the accuracy was much lower could be:

- **Poor Embedding Quality:** As the word embeddings were provided, there is a chance that the embeddings didn't represent the words well. A way to fix this is to load in the embeddings as a layer that is able to be fine tuned for the purpose of the task.
- **Vanishing/Exploding Gradient:** The longer the input is for a RNN, the gradient of the recurrent layer will constantly multiply previous gradients onto itself, causing some weights from earlier time-steps to be either effectively zero (not considered) or "explode" to a NAN output. This is most likely not a large issue with this model, as we are taking the output at every time-step already, but it may affect the later time-step predictions more. This can be combatted by using GRU or LSTM networks instead, or by adding drop-out layers.
- **Small Dataset:** The dataset provided may have been too small for the RNN to understand the inputs correctly, causing a non-converging model. This could have been a problem with this project, as often the recurrent networks would be overfit within the second epoch, meaning that either it converged within the first two epochs, or that it failed to understand the data. The fix would be to just increase the data input to the model.

5 Conclusion and Others (5pt)

The project could be configured to use the built in pytorch dataset and dataloader libraries, which may help students understand more intuitively about what is happening during loading the data. I think that it would also help if we could fine tune the word embeddings ourselves, or create have a method that creates a byte-encoding from the training set for students to better grasp how that would look.

While doing the project, it felt like I spent more time trying to understand what the code was doing and wondering why the RNN was performing worse despite it being a more revolutionary model compared to FFNN. That lead to a lot of time trying to adjust the RNN model to make it perform better, but most of it was not successful. I think it would be better if the project could show situations where the FFNN or RNN are better options compared to the other.