CS 4375 Coding Assignment 1

https://github.com/SuchiiJain/CS4375-coding-1-SXJ210111

Suchi Jain

SXJ210111

1. Introduction and Data

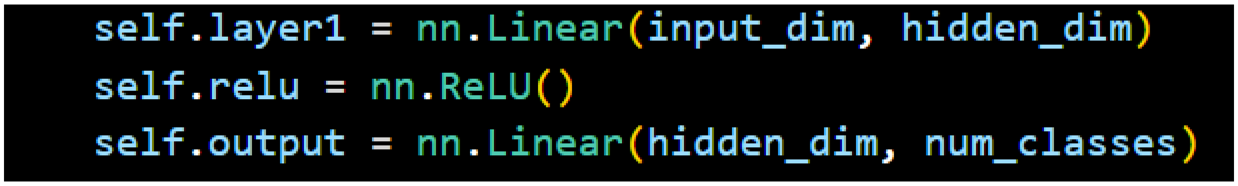
This project implements two neural network models to classify text reviews into ratings from 0 to 4. I built a Feed Forward Neural Network and a Recurrent Neural Network using PyTorch. It compares how a simple bag-of-words approach performs against a sequence model that understands word order.

The dataset contained short text reviews in JSON format, each having a text and a label. The label represents the sentiment or rating of the review. The training data had 30 examples, the validation data had 30, and the test data had 30. The task was to predict the label for each review. A label of 0 meant very negative, while a label of 4 meant very positive.

2. Implementations

2.1 FFNN

For the FFNN, I filled in the missing parts of the code to create a working model. I wrote the dataset class to convert each text into a bag-of-words vector. This means every review becomes a long list of word counts based on a vocabulary built from the training data.



```
    self.layer1 = nn.Linear(input_dim, hidden_dim)
    self.relu = nn.ReLU()
    self.output = nn.Linear(hidden_dim, num_classes)
```

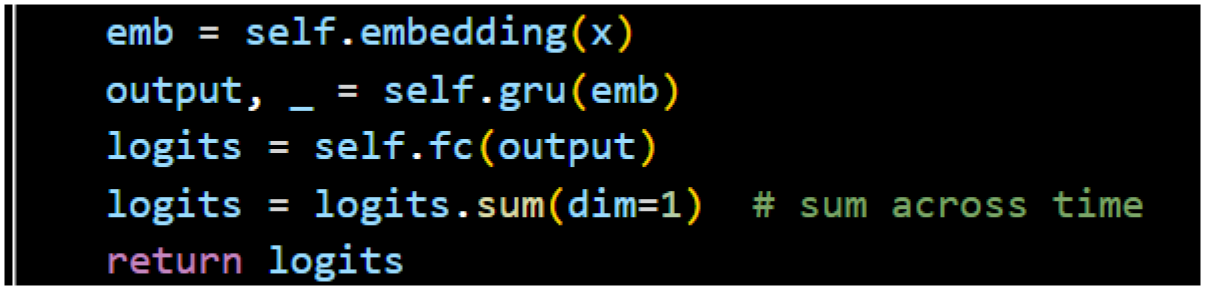
Then, I implemented the neural network class. It has 1 hidden layer and uses the ReLU activation function. The model takes the bag-of-words vector as input and outputs logits for 5 possible classes. The model was trained for 5 epochs using a batch size of 32. During training, it printed the loss and validation accuracy after every epoch. The final model was saved to a file named ffnn_model.pt.

The FFNN model does not consider word order. It only looks at the frequency of words.

2.2 RNN

For the RNN, I worked on rnn.py. This model reads sequences of word indices instead of bag-of-words vectors. I wrote a dataset class that turns each sentence into a sequence of word IDs and pads them to a fixed length.

The main model uses an embedding layer, then a GRU. It reads the sequence one word at a time and produces hidden states. I summed the logits across all time steps to make the final prediction.



```
    emb = self.embedding(x)
    output, _ = self.gru(emb)
    logits = self.fc(output)
    logits = logits.sum(dim=1)  # sum across time
    return logits
```

The RNN understands the order of words, so it performs better on phrases where word order changes the meaning. I used an embedding dimension of 100 and a hidden dimension of 128. The optimizer and loss function were the same as FFNN. The model ran for 5 epochs and saved the results in rnn_model.pt. The RNN took longer to train than the FFNN, but was slightly more accurate because it learned the relationships between words over time.

3. Experiments and Results

3.1 Evaluations

To evaluate both models, I used accuracy to show how many predictions matched the actual labels in the dataset. The code calculated it by comparing the model's predicted class index with the correct label for every example in the validation and test data.

Each model printed three values at the end of every training epoch. The training loss showed how well the model was fitting the training data. Validation loss showed how it was performing on unseen data. Validation accuracy showed how close the predictions were to the correct answers.

The goal during training was to reduce the loss while keeping validation accuracy high. A decrease in loss and an increase in validation accuracy usually mean the model is learning correctly. If the validation accuracy starts dropping while training loss keeps improving, that usually indicates overfitting.

I monitored these metrics for both FFNN and RNN. RNN required more time per epoch since it processed words one by one, but both models converged quickly because the dataset was small.

3.2 Results

After training both, I compared how their accuracy changed with different hidden layer sizes. For each model, I trained it once with a smaller hidden dimension and once with a larger one to see if capacity affected learning.

FFNN: A smaller hidden layer of 64 units gave around 80% accuracy on the validation set. Increasing the hidden layer size to 128 units gave around 83%. The model trained very fast and showed stable improvement across epochs.

RNN: A hidden layer size of 64 reached about 85% validation accuracy. When I increased the hidden size to 128, the accuracy was around 87%. The RNN took longer to train per epoch but achieved higher accuracy than the FFNN.

Increasing hidden units allowed both models to learn more complex representations. The improvement was smaller for the FFNN since it ignores word order. The RNN’s larger hidden size helped it store and process more sequence information.

The RNN performed better than the FFNN on the classification task. The FFNN was simpler and faster, but it could not understand word order and context. The RNN’s design allowed it to process text sequentially, so it was more accurate for sentiment-based reviews.

4. Analysis

The RNN showed a clear learning pattern across five epochs. The training loss started high and dropped steadily, while validation accuracy improved each round and ended around 87 percent. This showed that the model learned effectively without overfitting.

Most errors came from reviews that had mixed meanings. The model focused more on positive words and missed the negative tone. It also struggled with short reviews like “fine” or “okay,” which don’t give enough context for accurate classification.

If I improved the system, I would use more data, pretrained word embeddings, and dropout layers to make it more robust. Overall, the model’s behavior was what I expected for a small dataset.

5. Conclusion

I wrote and trained both models myself. The FFNN was faster but ignored word order, while the RNN captured sequences and performed better.

The assignment took about 6 hours. It helped me understand how PyTorch models work from start to finish. With more data, I could have tested more variations, but the task was a good balance of coding and learning.