CS6375 Assignment 1

https://github.com/VladimirASokolov/CS6375Assignment1

Vladimir Sokolov
VAS180007

1 Introduction and Data

The project was about classifying a set of reviews using RNN and FFNN to determine customer sentiment automatically. The data included reviews with their corresponding rating given. The ratings were 1 to 5 stars, with only integer ratings allowed. The skews for the training, test, and validation were interesting. Training had 8000 examples with a split of 40% 1 star, 40% 2 stars, and 20% 3 stars with no examples of the others. Test had 800 examples with a split of 20% 3 stars, 40% 4 stars, and 40% 5 stars with no examples of the others. Validation had 800 examples with a split of 40% 1 star, 40% 2 stars, and 20% 3 stars with no examples of the others. The main goal seemed to be to overcome the test set skewing the accuracy of the model against the validation set.

File	1 star	2 stars	3 stars	4 stars	5 stars	Total examples
training.json	3200	3200	1600	0	0	8000
test.json	0	0	160	320	320	800
validation.json	320	320	160	0	0	800

2 Implementations

2.1 FFNN

```
def forward(self, input_vector):
    # [to fill] obtain first hidden layer representation
    hidden = self.activation(self.W1(input_vector))
    # [to fill] obtain output layer representation
    output = self.W2(hidden)
    # [to fill] obtain probability dist.
    predicted_vector = self.softmax(output)
    return predicted_vector
```

The main parts to fill in were in the forward part which controls how the data passes through the model. The first line applies ReLU on the input vector which essentially deletes negative probabilities. The second line maps the previous result to the 5 output classes aka 1-5 stars. The final line applies softmax which converts the output scores into log probabilities.

The overall neural network uses two linear layers w1 and w2 with ReLU activation, log probability distribution using softmax, and NLLoss function for measuring how far off it is from the validation set. The JSON data is processed into a vocabulary set and vectors. For training it uses SGD optimizer with a learning rate of 0.01 and momentum of 0.9 which seem very general. FFNN didn't have any early stopping for overfitting.

2.2 RNN (25pt)

```
def forward(self, inputs):
    # [to fill] obtain hidden layer representation
    _, hidden = self.rnn(inputs)
    # [to fill] obtain output layer representations
    output_rep = self.W(hidden.squeeze(0))
    # [to fill] sum over output
    # seems unnecessary
    # [to fill] obtain probability dist.
    predicted_vector = self.softmax(output_rep)

    return predicted_vector
```

```
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
# optimizer = optim.Adam(model.parameters(), lr=0.001)
```

The main parts to fill in were in the forward part which controls how the data passes through the model. The first line passes the input through the RNN. The second line maps the previous result to the 5 output classes aka 1-5 stars. The final line applies softmax which converts the output scores into log probabilities.

The overall neural network uses a linear layer to map the outputs, log probability distribution using softmax, and a NLLoss function for measuring how far off it is from the validation set. The JSON data is processed into a vocabulary set and vectors. For training it gave the option to use Adam and SGD. For Adam I tried using a more normal learning rate of 0.001 but I'm not sure if it was better than the provided rate of 0.01. For SGD I used the same lr and momentum as RNN. RNN had an early stopping condition unlike FFNN when training accuracy increases while validation accuracy decreases to reduce overfitting.

3 Experiments and Results

I mainly evaluated the models based on the validation accuracy as the goal was to see how well the model worked on the validation set.The best result that I got with FFNN was with a low number of epochs, 5, and a middling number of hidden dimensions, 32. The lower number of epochs made sense as the test set was aiming to skew the results incorrectly so decreasing the count made the model a bit more accurate. The best RNN result I got was with Adam using a different lr than was given initially as most online sources used an lr of 0.001. Its result was 0.47125. With a bit more fine tuning I could have tried to find better parameters but it was time consuming to test. Overall, the models performed decently well considering how the skewed data was working against it and with how people can rate things differently than how they describe things. The exact parameters used were “optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)” and “optimizer = optim.Adam(model.parameters(), lr=0.001)”.

Table for FFNN

Hidden Dim	Epochs	Validation Accuracy
128	10	0.59375
128	20	0.53125
64	10	0.61875
64	20	0.59125
16	5	0.6125
8	5	0.57375
32	5	0.63125
64	5	0.5975

Table for RNN

Hidden Dim	Validation Accuracy ADAM	Validation Accuracy SGD
4	0.45	0.45375
8	0.46375	0.44125
16	0.47125	0.45

4 Conclusion and Others

Having some starting points and guidelines for the parameters would be nice since testing can be time consuming especially for RNN.