Assignment 4 Writeup

Murray Kang

Problem 1.1

I used the Glove embeddings of 50 dimensions in this part.

- Dog – cat (0.9218)
- Whale – whales (0.8987)
- Before – after (0.9511)
- However – although (0.9801)
- Fabricate – fabricating (0.7595)

Problem 1.2

I still used the Glove embeddings of 50 dimensions in this part.

dog : puppy :: cat: ?

- puppies : 0.7629
- scaredy : 0.7438
- kitten : 0.7406

speak : speaker :: sing : ?

- sang : 0.6226
- nateq : 0.6217
- lyricist : 0.6019

France : French :: England : ?

- scottish : 0.8679
- english : 0.8374
- welsh : 0.8057

France : wine :: England : ?

- orchard : 0.6624
- tasting : 0.6325
- tea : 0.6155

Problem 1.3

I used IMDE dataset in torchtext.datasets.IMDB for classification and Glove embeddings with 300 dimensions as pre-trained embeddings.

URL: http://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz

This dataset has 251639 text vocabulary, and the vector size of text vocabulary is [251639, 300].

Firstly, I used a torchtext.legacy.data.Field to store texts, and a LabelField to store corresponding labels. Then, I used BucketIterator.splits() to split train, validation, and test datasets with passing parameter (batch_size=32, repeat=False, shuffle=True, sort_key=lambda x: len(text)).

In terms of constructing the model, I used one nn.Embedding layer, one nn.LSTM layer, and one nn.Linear layer. I used self.word_embeddings.weight = nn.Parameter to input the Glove pre-trained embeddings into the embedding layer. Before problem 5, when I need to use the pre-trained embedding frozen, I passed requires_grad=False in the nn.Parameter, which enables the pre-trained embeddings not to change when training. Otherwise, in problem 5, I switched this to requires_grad=True.

In the forward method, the parameters I used are input_sentence, and batch_size=None, that input_sentence of shape = (batch_size, num_sequences) and batch_size used only for predication on a single sentence after training (batch_size = 1). And it would return the output of the linear layer containing logits for positive & negative class which receives its input as the final_hidden_state of the LSTM (final_output.shape = (batch_size, output_size)).

In the main file, I firstly construct a clip_gradient method which filters the parameters and performs gradient clipping. I used params = list(filter(lambda p: p.grad is not None, model.parameters())) to filter the parameters that we need to perform gradient clipping on.

In the training method, I firstly initialized three total epoch numbers for loss, accuracy, and f1 score. Then, I used an Adam optimizer which passed a filter(lambda p: p.requires_grad, model.parameters()), which enables to freeze some pre-trained word embeddings parameters before problem 4. Then, I turned on the training mode of the model. For each mini-batch, I firstly zero all gradients, and made the predication for the given texts in the batch. The, I counted the loss between prediction and target. To get my real prediction of labels, I used torch.max(prediction, 1)[1].view(target.size()).data to find the result according to the maximum relative values on label 0 and 1. Then, I can calculate the number of corrects to calculate the accuracy for the batch and calculate the recall and precision to calculate the f1 score on each label. After that I took an average on those two labels to get the macro-averaged f1 scores. Next, I would call loss.backward(), clip_gradient(model, 1e-1), optim.step(), steps += 1 to start next batch. After each epoch, I would sum up the loss, accuracy, and f1 score, and finally I would divide the sum by the number of batches to get the final results for the training set.

In the evaluation process, I also initialized three numbers for total epoch loss, accuracy, and f1 score. After turning on the evaluation mode of the model, I used with torch.no_grad() to disable gradient calculation. Then, I used the same way to get the real predictions of texts on labels by using torch.max(model(text), 1)[1].view(target.size()).data. Then, I can calculate and sum up the accuracies, f1 scores of every batch of validation or test set. Then, I divided them by the length of iteration (number of batches) to get the final metrics.

Finally, I called the training method without exceeding the max number of epochs and stopped training when the validation accuracy was not increasing.

After tuning the hyperparameter, I found the optimums as follows.

- learning_rate = 5e-5
- batch_size = 32
- output_size = 2
- hidden_size = 256
- embedding_length = 300
- max_epochs = 20
- loss_fn = F.cross_entropy

Here are my results for using pretrained embeddings (before problem 5).

	Training Set	Validation Set	Test Set
Accuracy	80.16%	79.57%	76.06%
F1 score	0.795	0.790	0.652

	Train Accuracy	Train F1 Score	Valid. Accuracy	Valid. F1 Score	Training Time
Epoch 1	51.03%	0.42	53.56%	0.43	10.87s
Epoch 2	55.87%	0.51	61.21%	0.54	8.45s
Epoch 3	66.16%	0.64	74.6%	0.74	8.41s
Epoch 4	76.44%	0.76	78.05%	0.77	8.46s
Epoch 5	78.23%	0.78	79.64%	0.79	8.43s
Epoch 6	79.85%	0.79	80.09%	0.8	8.40s

For problem 1.5, I just turned on the ‘required_grad=True’ in the ‘self.word_embedding.weights =nn.parameter()’ to perform finetuning. Here are my results for problem 5 (update all word embedding during training).

	Train Accuracy	Train F1 Score	Valid. Accuracy	Valid. F1 Score	Training Time
Epoch 1	51.67%	0.45	51.56%	0.39	24.16s
Epoch 2	55.63%	0.52	61.4%	0.54	21.43s
Epoch 3	70.71%	0.69	75.45%	0.74	20.87s
Epoch 4	79.71%	0.79	80.44%	0.8	20.80s
Epoch 5	82.88%	0.82	82.07%	0.81	20.94s
Epoch 6	84.93%	0.85	82.82%	0.82	20.98s
Epoch 7	87.29%	0.87	83.8%	0.83	20.88s
Epoch 8	88.82%	0.89	84.48%	0.84	20.81s
Epoch 9	90.85%	0.91	85.31%	0.85	20.80s
Epoch 10	92.37%	0.92	85.43%	0.85	20.78s

	Training Set	Validation Set	Test Set
Accuracy	89.21%	82.82%	81.79%
F1 score	0.872	0.812	0.701

Problem 2

$$score(good) = \theta \cdot x_{good}$$

$$score(bad) = \theta \cdot x_{bad}$$

$$score(not\ good) = \theta \cdot x_{not} + \theta \cdot x_{good}$$

$$score(not\ bad) = \theta x_{not} + \theta x_{bad}$$

$$score(not\ good) - score(good) = \theta \cdot x_{not}$$

$$= score(not\ bad) - score(bad)$$

Thus, if $score(good) > score(not\ good)$

$$\Rightarrow \theta \cdot x_{not} < 0 \Rightarrow score(bad) > score(not\ bad)$$

Similarly,

If $score(bad) < score(not\ bad)$

$$\Rightarrow \theta \cdot x_{not} > 0 \Rightarrow score(good) < score(not\ good)$$

Thus, the two inequalities can not both hold.

① $\text{score}(\text{very very good}) > \text{score}(\text{not very good})$

② $\text{score}(\text{not very bad}) > \text{score}(\text{very very bad})$

Suppose ① hold,

$$\frac{1}{3}(\theta \cdot X_{very} + \theta \cdot X_{very} + \theta \cdot X_{good}) - \frac{1}{3}(\theta \cdot X_{not} + \theta \cdot X_{very} + \theta \cdot X_{good}) > 0$$

$$\Rightarrow \quad X_{very} > X_{not}$$

$$\text{score}(\text{not very bad}) = \frac{1}{3}\theta \cdot (X_{not} + X_{very} + X_{bad})$$

$$\text{score}(\text{very very bad}) = \frac{1}{3}\theta \cdot (X_{very} + X_{very} + X_{bad})$$

Given $X_{very} > X_{not}$

$$\frac{1}{3}\theta \cdot (X_{not} + X_{very} + X_{bad}) < \frac{1}{3}\theta \cdot (X_{very} + X_{very} + X_{bad})$$

Thus, ② does not hold.

Similarly, suppose ② hold,

$$X_{not} > X_{very}$$

Then, $\text{score}(\text{very very good}) = \frac{1}{3}\theta(X_{very} + X_{very} + X_{good})$

$\text{score}(\text{not very good}) = \frac{1}{3}\theta(X_{not} + X_{very} + X_{good})$

$$\Rightarrow \quad \text{score}(\text{very very good}) < \text{score}(\text{not very good})$$

Thus, ① does not hold.

In conclusion, ① and ② cannot both hold.

Problem 3

$$\Theta = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad X_{good} = \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \end{bmatrix} \quad X_{bad} = \begin{bmatrix} 0 \\ -1 \\ -2 \\ -3 \end{bmatrix}$$

$$X_{not} = \begin{bmatrix} 1 \\ -10 \\ -10 \\ -10 \end{bmatrix}$$

$$\text{Score (good)} = \Theta \cdot \text{reLU}\left(\begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \end{bmatrix}\right) = 6$$

$$\text{Score (not good)} = \Theta \cdot \text{reLU}\left(\begin{bmatrix} 1 \\ -10 \\ -10 \\ -10 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = 1$$

$$\text{score (bad)} = \Theta \cdot \text{reLU}\left(\begin{bmatrix} 0 \\ -1 \\ -2 \\ -3 \end{bmatrix}\right) = 0$$

$$\text{Score (not bad)} = \Theta \cdot \text{reLU}\left(\begin{bmatrix} 1 \\ -10 \\ -10 \\ -10 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \\ -2 \\ -3 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = 1$$

Thus, score (good) > score (not good) and score (bad) < score (not bad)

Problem 4

$$\mathcal{L} = \sum_{i\in\vec{V}} \sum_{j\in\vec{e}} count(i,j) \cdot \log\sigma(\vec{u}_i, \vec{v}_j) + \sum_{i\in\vec{V}} \sum_{j\in\vec{e}} \sum_{i'\in W_{neg}} \log(1-\sigma(\vec{u}_{i'}, \vec{v}_j))$$

$$= \sum_{i\in\vec{V}} \sum_{j\in\vec{e}} count(i,j) \cdot \log\sigma(\vec{u}_i, \vec{v}_j) + \sum_{i\in\vec{V}} Count(\vec{u}_i) \cdot count(W_{neg}) \cdot E_{W_{neg}}(\log\sigma(-\vec{u}_{i'}, \vec{v}_j))$$

Expectation term:

$$E_{W_{neg}}(\log\sigma(-\vec{u}_{i'} \cdot \vec{v}_j)) = \sum_{j\in e} \hat{P}(i) \log\sigma(-\vec{u}_{i'}, \vec{v}_j)$$

$$\mathcal{L}(\sigma(\vec{u}_i, \vec{v}_j)) = Count(\vec{u}_i, \vec{v}_j) \cdot \log\sigma(\vec{u}_i, \vec{v}_j) + count(W_{neg}) \cdot Count(\vec{u}_i) \cdot \hat{P}(i) \log\sigma(-\vec{u}_i, \vec{v}_j)$$

Let $x = \vec{u}_i \cdot \vec{v}_j$

$$\frac{\partial \mathcal{L}}{\partial x} = count(\vec{u}_i, \vec{v}_j) \cdot \sigma(-x) - count(W_{neg}) \cdot count(\vec{u}_i) \cdot \hat{P}(i) \cdot \sigma(x)$$

Comparing the derivative to 0,

$$e^{2x} - \left(\frac{count(\vec{u}_i, \vec{v}_j)}{count(W_{neg}) count(\vec{u}_i) \hat{P}(i)} - 1\right) e^x - \frac{count(\vec{u}_i, \vec{v}_j)}{count(W_{neg})\, count(\vec{u}_i) \cdot \hat{P}(i)} = 0$$

Let $y = e^x$, and solve the quadratic function of y.

$$y = \frac{count(\vec{u}_i, \vec{v}_j)}{count(W_{neg}) \cdot count(\vec{u}_i) \cdot \hat{P}(i)}$$

Substitute e^x back, and $x = \vec{u_i} \cdot \vec{v_i}$

$$\vec{u_i} \cdot \vec{v_i} = \log\left(\frac{\text{count}(\vec{u_i}, \vec{v_j})}{\text{count}(\vec{u_i}) \cdot \hat{p}(i) \cdot \text{count}(v_{neg})}\right) = \log\left(\frac{\text{count}(\vec{u_i}, \vec{v_j})}{\text{count}(\vec{u_i}) \cdot \hat{p}(i)}\right) - \log(\text{count}(v_{neg}))$$

$\log\left(\frac{\text{count}(\vec{u_i}, \vec{v_j})}{\text{count}(\vec{u_i}) \cdot \hat{p}(i)}\right)$ is the PMI of pair $(\vec{u_i}, \vec{v_j})$

$$M_{ij}^{SGNS} = \vec{u_i} \cdot \vec{v_j} = PMI(\vec{u_i}, \vec{v_j}) - \log(\text{count}(v_{neg}))$$