Mini 2 Sentiment Classification Report

Chenkuan Liu
Natural Language Processing
UT Austin
Mar 2021

1 Introduction

This project is using deep neural network with PyTorch for sentiment classification. The dataset comes from Rotten Tomato movie reviews. Each input is a tokenized sentence and the output is simplified to be either 0 or 1, where 0 denotes negative review and 1 denotes positive review.

2 Method 1: Deep Averaging Network

We first use deep averaging network for this task. We average all the word embeddings from an input sentence, and use that as a fixed-length input to the neural network. The word embeddings are pretrained and each word has a corresponding 300-dimensional embedding vector. For an unknown word, the embedding will be a 300-dimensional zero vector. After experimenting with different setups, I constructed two fully-connected hidden layers of size 60 and 10 each with no activation functions inside. The output is a 2-dimension vector, which are then softmaxed to compute cross-entropy loss and perform backpropagation.

The values inside the matrices are initialized using Xavier initialization. The initial learning rate is 0.001 and the optimizer is Adam. During each epoch, we randomly shuffle the input training sentences. After running for 20 epochs, I get 0.81 (5618 / 6920) accuracy on training set and 0.78 (680 / 872) on dev set. The results may vary depending on random initialization and shuffling, but the dev accuracy will generally surpass 0.77 and sometimes get around 0.78.

3 Method 2: LSTM and CNN

In the network above, we are averaging the embeddings of each word. As a result, the model in method 1 is unable to capture the change of output due to different word ordering. To solve this, we employ recurrent neural network in method 2.

I am using LSTM on the input sentence as the first step. Since the longest sentence has length 60, the input to LSTM is a $(60, 1, 300)$ tensor. Note that for sentence with length less than 60, we need to pack them with zero vectors. Each word is transformed to its corresponding 300-dimensional pretrained embedding. I set the `hidden_size` in LSTM to be 64. After going through LSTM, I take the output from each state and pool them into a fully-connected layer. So the output is of size $(60 \times 1 \times 64, 1)$. After this, since many sentences have length much less than 60, the LSTM output will be a sparse vector. To better capture the information from the $(60 \times 1 \times 64, 1)$ fully-connected layer, I add two 1d-convolution layers after it. The first convolution layer takes the $(60 \times 1 \times 64, 1)$ FC layer as an input filter. It has 64 output filters. The kernel size is 640, which corresponds to 10 LSTM hidden states. The stride is 64, which is the same as the hidden size. The padding is 640 to better capture the beginning or end of the sentence. Therefore the output of the first convolution layer is 64×71. The second convolution layer takes these 64 input filters and also output 64 filters. The kernel size is 8 and stride is 2. Relu is applied for both convolution layers. So, the output after the 2nd convolution layer is 64×32. After reshaping it to an 2048×1 FC layer, I add two hidden FC layers with size 512 and 64 respectively, which uses Relu activation and dropout rate 0.25. The final output layer is a 2-dimensional vector.

The learning rate, optimizer and sentence shuffling are applied in the same way as method 1. After running for 3 epochs, the training accuracy will get above 0.90 and the dev accuracy will be around 81.0∼82.0. The results will vary slightly depending on random initialization. The implementation that is used to predict and write the labels into the blind test set `eng.testb.out` has 0.91 (6297 / 6920) accuracy on training set and 0.82 (713 / 872) accuracy on dev set.

4 Appendix

4.1. Method 1 network structure, no activation or dropout inside:

```
FFNN(
  (V1): Linear(in_features=300, out_features=60, bias=True)
  (W2): Linear(in_features=60, out_features=10, bias=True)
  (W3): Linear(in_features=10, out_features=2, bias=True)
  (log_softmax): LogSoftmax(dim=0)
)
```

4.2. Method 2 network structure, relu is applied after conv1, conv2, fc1 and fc2, and dropout is applied only after fc1 and fc2:

```
FFNN_Fancy(
  (lstm): LSTM(300, 64, batch_first=True)
  (conv1): Conv1d(1, 64, kernel_size=(640,), stride=(64,), padding=(640,))
  (conv2): Conv1d(64, 64, kernel_size=(8,), stride=(2,))
  (fc1): Linear(in_features=2048, out_features=512, bias=True)
  (fc2): Linear(in_features=512, out_features=64, bias=True)
  (fc3): Linear(in_features=64, out_features=2, bias=True)
  (log_softmax): LogSoftmax(dim=0)
  (relu): ReLU()
  (drop): Dropout(p=0.25, inplace=False)
)
```