NLP HW2 Sentiment Analysis

109550003 Mao-Siang Chen

1 Methodology

Preprocess Data

First, I read the CSV files and change the emotion labels into numbers 0-6. In order to enhance the accuracy of the emotions in small scale, I duplicate these data several times to average the distribution. Second, I use nltk package for sentence tokenization and remove the stopwords with lowercase sentences as input. Third, encode the words in the sentences with integers defined in `word2idx` dictionary, which is simply the order of the word's first appearance. For the validation and test data, we encode a word as 0 if it doesn't appear in training data. Finally, pad the encoded sequences with 0 to a fixed length since the LSTM model needs a fixed input length.

Build Model

In this assignment, I choose bidirectional LSTM as the basic model. First, the input will be transformed into pretrained word embedding. Second, it will go through the single layer LSTM. Third, a linear layer will transfrom the output of LSTM into a vector with size 7 to represent the probability of each class. While predicting labels, it will be processed with a softmax function. The hyperparameters are 20 epochs with batch size 256, learning rate 2e-4 and 24 hidden layers in LSTM. Then, with attention implemented, the output will pass the attention and linear layers as final output.

```
RNN(
  (embedding): Embedding(5845, 100)
  (lstm): LSTM(100, 24, dropout=0.1, bidirectional=True)
  (linear1): Linear(in_features=48, out_features=7, bias=True)
)
```

Figure 1: Model Structure

```
def attention(self, lstm_output, hidden_output, input):
    # lstm_output = (len(sentence), batch_size, 2*n_hidden)
    lstm_output = lstm_output.permute(1, 0, 2)
    merged_state = torch.cat([s for s in hidden_output], 1)

    merged_state = merged_state.squeeze(0).unsqueeze(2)
    weights = torch.bmm(lstm_output, merged_state)
    weights = F.softmax(weights.squeeze(2), dim=1).unsqueeze(2)
    '''…
    return torch.bmm(torch.transpose(lstm_output, 1, 2), weights).squeeze(2)
```

Figure 2: Attention Mechanism

2 Questions

Have you tried pretrained word embedding?(e.g. Glove or Word2vec).What is the influence of the result after you using them?

I have tried the Glove pretrained embedding with 100 dimensions, and the model gets a lower loss and tends to converge quickly than those without a pretrained embedding. However, since my model have randomness, I cannot make sure that it will lead to a higher f1-score or accuracy of small scale emotions.

Have you tried attention on your model? What is the influence of the result after you use them? Which text your model attention on when it predict the emotion?

I have tried attention on my LSTM model, and the f1-score of my submission improves a lot while the overall accuracy and loss do not have significant improvements. However, since most of the word in the test data are not in the training data, the attention mechanism focus on `<unk>` or some punctuation in most cases. However, there are some examples that focus on verbs or interjections which makes more sense.

```
start to predict test set...
['okay', 'go', 'left', 'left', '!', 'left', '!']
focus on: left
['monica', 'still', 'turn', 'lights', 'bedroom', '?', '<unk>']
focus on: <unk>
['oh', 'thinking', 'people', 'game', '<unk>', '<unk>', '<unk>']
focus on: <unk>
['ohh', 'know', 'one', 'thing', '!', '<unk>', '<unk>']
focus on: know
['<unk>', '!', '<unk>', '<unk>', '<unk>', '<unk>', '<unk>']
focus on: <unk>
['well', 'think', "'s", 'perfect', '<unk>', '<unk>', '<unk>']
focus on: <unk>
['ahh', 'thank', '!', '<unk>', '<unk>', '<unk>', '<unk>']
focus on: <unk>
['uh', "'s", '<unk>', 'morning', '!', '<unk>', '<unk>']
focus on: <unk>
['okay', '?', '<unk>', '<unk>', '<unk>', '<unk>', '<unk>']
focus on: <unk>
['kidding', "n't", 'ever', 'disagree', 'okay', "'m", 'kidding']
focus on: n't
['oh', 'good', 'god', '<unk>', '<unk>', '<unk>', '<unk>']
focus on: oh
['yeah', '<unk>', '<unk>', '<unk>', '<unk>', '<unk>', '<unk>']
focus on: <unk>
['see', '<unk>', 'friend', 'rachel', 'wants', 'set', '<unk>']
focus on: rachel
['whoa-whoa-whoa', '!', 'emily', 'honey', 'okay', '?', '<unk>']
focus on: whoa-whoa-whoa
Done
```

Figure 3: Highest Attention word when predicting

Have you used other information from dataset to improve your model performance?(e.g. Speaker) What is the influence of the result after you use them?

I have tried using Speaker to improve my model performance, but the change is not significant enough to determine whether the influence is due to the model randomness or the Speaker label.