

# Report of Natural Language Processing, hw2

109550027 紀竺均

1. Describe how you build your model ? How did you do to preprocess your data from dataset? The distribution of the emotion is unbalance, what did you do to improve the accuracy on those emotion which are in small scale?(30%)

- Model:

I use Pytorch to build a multi-layer long short-term memory (LSTM) RNN. Followed by an attention layer that connected the output and hidden layer of the LSTM and finally the output layer.

Some parameters:

```
dimension_model = 300
num_layers = 5
hidden_size = 200
linear_hidden_size = 30
dropout = 0.2
attention_width = 15
```

Details of constructing the model:

For LSTM layer, I simply call 'torch.nn.LSTM'. For attention layer, First, I iterated through all the input. Then, calculate the attention score from query and source and use softmax function to get the probability distribution. I think the most complicated part is to deal with matrix multiplication using torch.

- Data preprocess:

- (1) text.lower()
- (2) Use collections.Counter and nltk.tokenize.word\_tokenize to calculate the frequency of each word in training data set.
- (3) Remove low frequency word ( exist = 1 ) -> **lower score**
- (4) Encode each utterance into a vector of indexes that represent the words.
- (5) Use stemming and lemmatization -> **lower score**

I remove (3) and (5) in final version since using this methods result in lower score. I didn't remove stopwords because I think the length of the sentence also determined its emotions.

- Deal with unbalance data:

Because of the lack of some emotion in training data, I merged the training dataset

and validation dataset to get a larger dataset and use K-fold cross validation with k=5 to repeatedly resampling the data and introduced randomness to the dataset. The function `cross_val()` return the *i*th dataset (for *i* = 1~k) with 80% training data and 20% validation data. After applying cross validation, I get a higher score.

2. Have you tried pretrain word embedding?( e.g. Glove or Word2vec).What is the influence of the result after you using them?(30%)

Yes, I have tried to use 'glove.6B.300d', 'glove.42B.300d', and 'glove.840B.300d' to pretrain my embedding.

'glove.840B.300d' matches the most vocabulary in word2index list (6266 / 6933), but **the result score of using 'glove.42B.300d' is the highest of all and undoubtedly beat the model without GloVe.**

Model	Matched-word	F1-score
without Glove	0	0.27811
'glove.6B.300d'	4234/ 6933	0.2833
'glove.42B.300d'	4701 / 6933	<b>0.3362</b>
'glove.840B.300d'	<b>6266 / 6933</b>	0.30045

```
class LSTM(torch.nn.Module):
    def __init__(self):
        super(LSTM, self).__init__()
        self.embed =
            torch.nn.Embedding.from_pretrained(torch.from_numpy(embs_npa).float())
```

note: `embs_npa` is a numpy array that I save previously with the embeddings of the word in the order the same as “word2index” list.

**My conclusion:** We can be sure that **pretrain the embedding have a better performance than doing nothing**, but since there are some randomness generated during the training process, we can't be 100% sure that 42B is a better pretrain model than 840B. Besides, the larger pretrain model sometimes results in memory error.

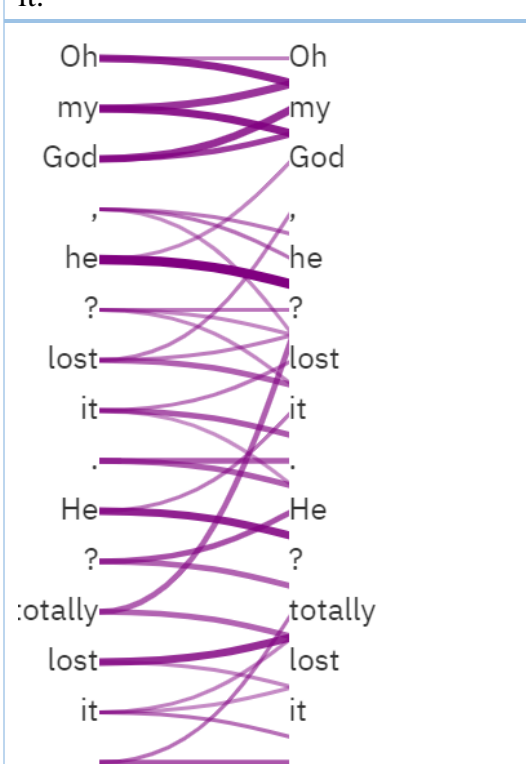
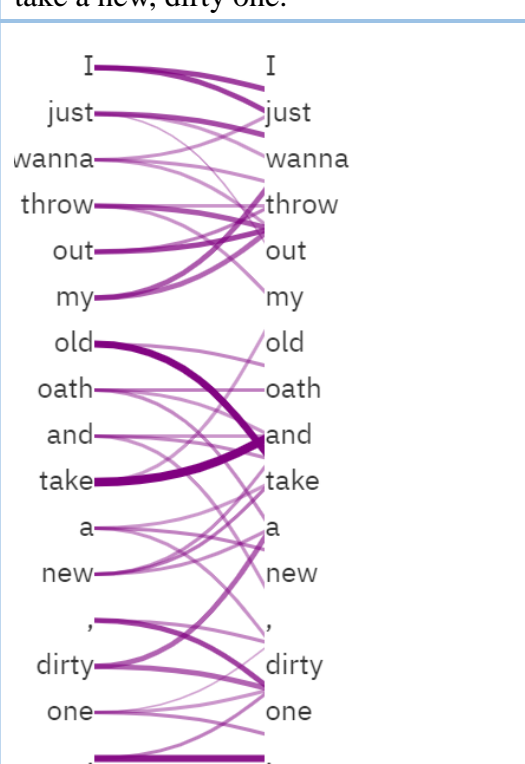
3. Have you tried attention on your model? What is the influence of the result after you using them? Which text your model attention on when it predict the emotion?(30%)

Yes, I add an attention layer after the LSTM layer.

At first, the score gets lower, but after I modified “attention\_width” parameter from 5 to 15, the **performance of the network get better ant beats my origin model.**

Model	F1-score
without attention	0.27722
Attention_width = 5	0.19301
Attention_width = 12	0.29811
Attention_width = 15	0.30955

The attention layer calculates the attention score by multiplicative attention function for each input timestep, and use it to get the weighted output. So, **the attention algorithm attention on the part of input that it thinks will directly influence the output.** Below I’ll use the dataset to show a few examples:

Oh my God, he? lost it. He? totally lost it.	I just wanna throw out my old oath and take a new, dirty one.
	

4. Have you used other information from dataset to improve your model performance? (e.g. Speaker) What is the influence of the result after you using them? (10%)

Yes, I take 'Speaker attribute' into consideration by turning it into an index and insert it in the front of encoded vector. This results in a **small increase in f1-score in simple LSTM model, but limited the validation accuracy in the model that I add attention layer.**

Model	F1-score
Without speaker	0.23844
Add speaker	0.24401
Without speaker + attention	0.29074
Add speaker + attention	0.27955

Note: I only take the six main characters into consideration.

```
s2idx = {"Chandler":0, "Joey":1, "Rachel":2, "Monica":3, "Phoebe":4, "Ross":5}
if(speaker in s2idx):
    idx = s2idx[speaker]
else:
    idx = 6
encoded.insert(0,idx)
```