ADL 2019 HW2 Report

Q1: Describe your ELMo model

1. Training corpus processing
    a. Tokenization
        i. special: unk, pad, bos, eos
        ii. use the ta's tokenization with simple split
    b. Vocabulary building
        i. word vocab: catch the top 135000 by TA's suggestion
        ii. char vocab: just use the charset in string.printable
    c. Data split
        i. Split the sentence into each chunk with size 64 to prevent the lstm from predict long distance information.
2. Model architecture and implementation details

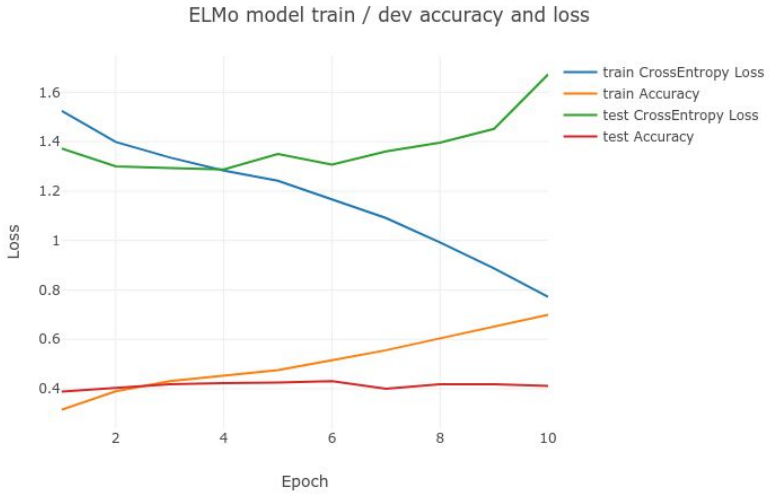
```
def forward(self, f, b, f_y=None, b_y=None, f_mask=None, b_mask=None):
    f1 = self.char_embedding(f)
    b1 = self.char_embedding(b)

    f2 = self.forward_lm_1(f1)[0]
    f2 = self.forward_lp_1(f2)
    b2 = self.backward_lm_1(b1)[0]
    b2 = self.backward_lp_1(b2)

    f3 = self.forward_lm_2(f2)[0]
    f3 = self.forward_lp_2(f3)
    b3 = self.backward_lm_2(b2)[0]
    b3 = self.backward_lp_2(b3)

    if f_y is None:
        return (f1.data.cpu().numpy(), f2.data.cpu().numpy(), f3.data.cpu().numpy()),\
               (b1.data.cpu().numpy(), b2.data.cpu().numpy(), b3.data.cpu().numpy())

    dim = f3.size(0)*f3.size(1)
    f = f3.view(dim, -1)
    f_y = f_y.view(-1)
    f_mask = f_mask.view(-1)

    b = b3.view(dim, -1)
    b_y = b_y.view(-1)
    b_mask = b_mask.view(-1)

    return (self.loss(f[f_mask],f_y[f_mask]).loss + self.loss(b[b_mask],b_y[b_mask]).loss)
```

- each operation with lp is a Linear(4*embedding_dim, embedding_dim)
- each operation with lm is a LSTM(embedding_dim, 4*embedding_dim)
- char_embedding is the one that TA provides.

3. Hyperparameters of your ELMo model
    a. char_embedding:
        i. 2 highway network
        ii. conv filters: [(1,32), (2,64), (3,128), (4,128), (5,256), (6,256), (7,512)]

b. BiLM
    i. 2 layers of LSTM for each
    ii. hidden_dim: 2048
    iii. a projection for 2048 -> 512
c. common
    i. optimizer: Adam
    ii. learning rate: 1e-3
    iii. batch size: 64

4. Plot the score on train/dev set while training



- It's quite strange that the validation set accuracy is only 0.43 at epoch 6, but the public leaderboard score is 0.46244.

5. Show the performance of the BCN model with and without ELMo on the public leaderboard
    a. BCN without ELMO: 0.4434
    b. BCN with ELMO: 0.46244

Q2: Compare different settings (Require public leaderboard score)

1. Different number of training steps
    a. Performance
        i. 20000 steps: 0.46063 (public score), 0.4396 (dev)
        ii. 40000 steps: 0.46696 (public score), 0.4314 (dev)
        iii. 60000 steps: 0.46244 (public score), 0.4314 (dev)
    b. Discussion
        i. In general, ELMo may help the task. However, it's really weird that the performance improvement for each steps are almost the same. I have observed that the training NLL loss is already lower than 4 after 10000 steps. This may imply that maybe the task vocabulary set is much smaller than ELMo training set. As a result, there isn't any obvious difference between 20000 steps, 40000 steps and 60000 steps.
2. Different hyperparameters
    a. Performance
        i. embedding_dim: 512, 0.46244 (public score) (The one in Q1)
        ii. embedding_dim: 256, 0.42805 (public score)
    b. From the result, we could infer that the contextual space is quite complex. The 256 case doesn't seem to encode the information effectively. It's even worse than the one without contextual embedding.

Q3: Describe your model that passes strong baseline

1. Input of your model (word embedding? character embedding?)
    a. Bert
2. Model architecture
    a. BertForSequenceClassification.from_pretrained('bert-base-uncased',num_labels=5)
    b.

```
BertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): BertLayerNorm()
      (dropout): Dropout(p=0.1)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0): BertLayer
        (1): BertLayer
        (2): BertLayer
        (3): BertLayer
        (4): BertLayer
        (5): BertLayer
        (6): BertLayer
        (7): BertLayer
        (8): BertLayer
        (9): BertLayer
        (10): BertLayer
        (11): BertLayer
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (dropout): Dropout(p=0.1)
  (classifier): Linear(in_features=768, out_features=5, bias=True)
)
```

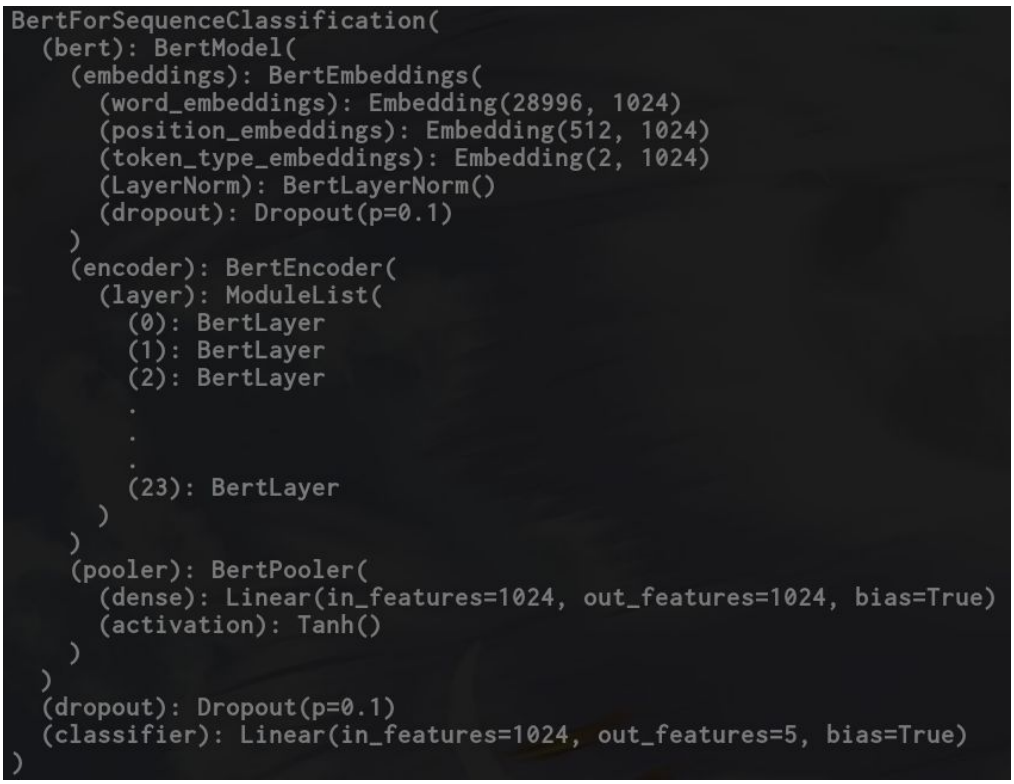
    c. BertLayer

```
(0): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): BertLayerNorm()
      (dropout): Dropout(p=0.1)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): BertLayerNorm()
    (dropout): Dropout(p=0.1)
  )
)
```

3. Hyperparameters of your model
    a. optimizer Adam
        i. lr: 1e-5
        ii. weight_decy: 1e-8
    b. batch_size: 4
    c. uncased
    d. without [BOS] and [EOS]
4. Performance: 0.5095 (public score)

Q4: Describe your best model

1. Describe your best model (0.5%)
    a. Input to your model
        i. Raw text -> Bert Tokenizer -> BertForSequenceClassification.from_pretrained('bert-large-cased')
    b. Model architecture
        i.



ii. BertLayer

c. Hyperparameters
   i. Adam
      1. lr=1e-5
   ii. batch_size: 6
d. Performance 0.51403 (public score)

2. Describe the reason you think why your best model performs better than other models
   a. Using cased instead of uncased. Cased may contain some information such as the strength of the sentiment.
   b. Large vs small. The large Bert model has more parameter size. That is to say, it may model better than the small one when solving the masked language task. As a result, the stuff bert large catched may be more reasonable.

Q5: Compare different input embeddings

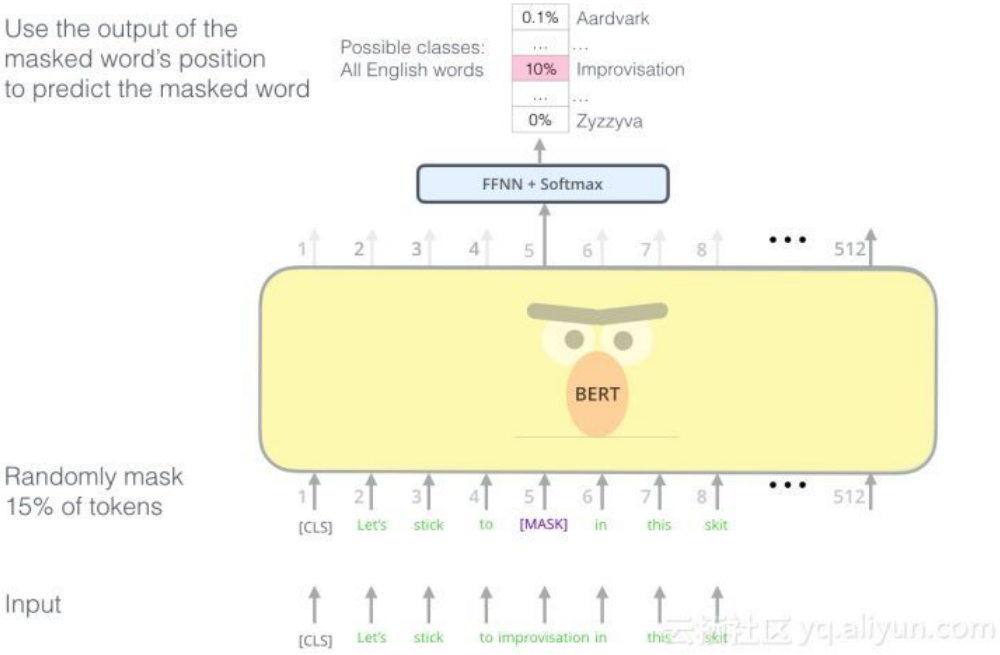
In this home work, you may encounter models taking inputs in different forms.

Please compare the pros and cons of using character embedding, word embedding and byte pair encoding.

- character embedding
    - pros
        - reasonable to handle OOV
            - words share root and prefix (字根, 字首)
        - model complexity low -> faster (compare to word embedding)
        - tolerate
    - cons
        - not applicable to language such as chinese, since the smallest component in chinese is word instead of character.
- word embedding (word2vec)
    - pros
        - compare to one-hot encoding, word embedding saves lots of memory and save the word information in a smaller vector.
        - the vectors have some special geometry property
            - e.g. "queen" - "king" + "man" = "woman"
    - cons
        - cannot handle OOV
            - only <unk>
        - model complexity high
- byte pair
    - pros
        - effectively reduce vocabulary set size
    - cons
        - not applicable to language such as chinese, since the smallest component in chinese is word instead of character.

Q6: BERT

1. Please describe the tasks that BERT used for pre-training. What do they have comparing to language model pretraining used in ELMo?
    a. ELMo trains the embedding by predicting the next word with two directions.
    b. BERT tries to solve the masked language task. As ELMo the input would be a sentence. Then, some of the words would be randomly masked. BERT tries to predict the origin sentence by the masked sentence.



2. Please describe in detail how you would formulate the problem and apply BERT on it.
    a. First, we could load pre-trained models of BERT. Then, we could simply concat all the output of BERT and connect it with a DNN which maps the dimension to the tasks dimension. By backpropagation, the pretrained weight may have slightly difference. But as we know, the DNN part would update more fastly then the pre-trained part.

Q7: Bonus

1. Compare more than 2 contextualized embedding on this task
    a. ELMo: 0.46244 (public score)
    b. BERT BertForSequenceClassification
        i. bert-large-cased 0.5142 (public score)
        ii. bert-small-uncased 0.5095 (public score)
        iii. bert-large-uncased 0.5104
2. Apply your ELMo embedding to other tasks.
    a. I choose Stanford Question Answering Dataset (SQuAD) 2.0
    b. Baseline Model : BiDAF + Cross Attention(https://github.com/chrischute/squad)
        i. F1: 60.41
        ii. EM: 57.07
    c. ELMo Appended: Directly append ELMo to the glove embedding
        i. F1: 61.37
        ii. EM: 58.45
    d. Discussion
        i. It looks like the ELMo embedding has some help to the SQuAD task. But in general, the baseline I used is really quite weak such that it's weaker than the weakest one on the SQuAD scoreboard.
3. Apply any kind of contextualized embedding method on HW1 and report the performance
    a. Original: 0.938666 (public score), 0.944571 (private score)
    b. ELMo 20000 step: 0.94466 (public score), 0.94928 (private score)
    c. Discussion
        i. It doesn't have improvement. In general, the reason may be that the distribution of the word in HW1 is quite different from our training corpus. While doing homework 1, we could find that there is a lot of strange word like "goodddddddddddddddddddddddddddd" or "rm -rf". It's quite different from the training corpus.