ADL Homework 1 Report

b05902002 資工三 李栢淵

Q1: Describe your ELMo model. (2%)

1. Training corpus processing. (tokenization, vocabulary building) (0.4%)
    - tokenization: Has been finished by TA, using spacy tokenize.
    - I use the first 1000K sentences without any filtering and constrict their length to 64.
    - Only use the Top 80000 frequent word in vocabulary, the others are considered $[UNK]$

2. Model architecture and implementation details. (0.4%)

    Consider TA's suggestion, I used the CharEmbedding offered by TA, which contains HighwayNetwork using CNN to catch the relation between words and characters.

    After get the char embedding, I put the sequence to one-direction LSTM and then do the projection to the lower dimension on the output. And then do it again, including one-direction LSTM and projection.

    FInally, considering the large vocabulary set, I use the AdaptiveSoftmax as the final layer to accelerate.

    - Prepared Data: $[SOS], w_1, w_2, \ldots, w_{64}, [EOS]$
    - For Forword:
        - Input_f: $[SOS], w_1, w_2, \ldots, w_{64}$, Labels_f: $w_1, w_2, \ldots, w_{64}, [EOS]$
        - Input_f -> CharEmbedding -> LSTM1_f -> Projection1_f ->

          LSTM2_f -> Projection2_f -> AdaptiveSoftmax
    - For Backword:
        - Input_b: $[EOS], w_{64}, w_{63}, \ldots, w_1$, Labels_f: $w_{64}, w_{63}, \ldots, w_1, [SOS]$
        - Input_b -> CharEmbedding -> LSTM1_b -> Projection1_b ->

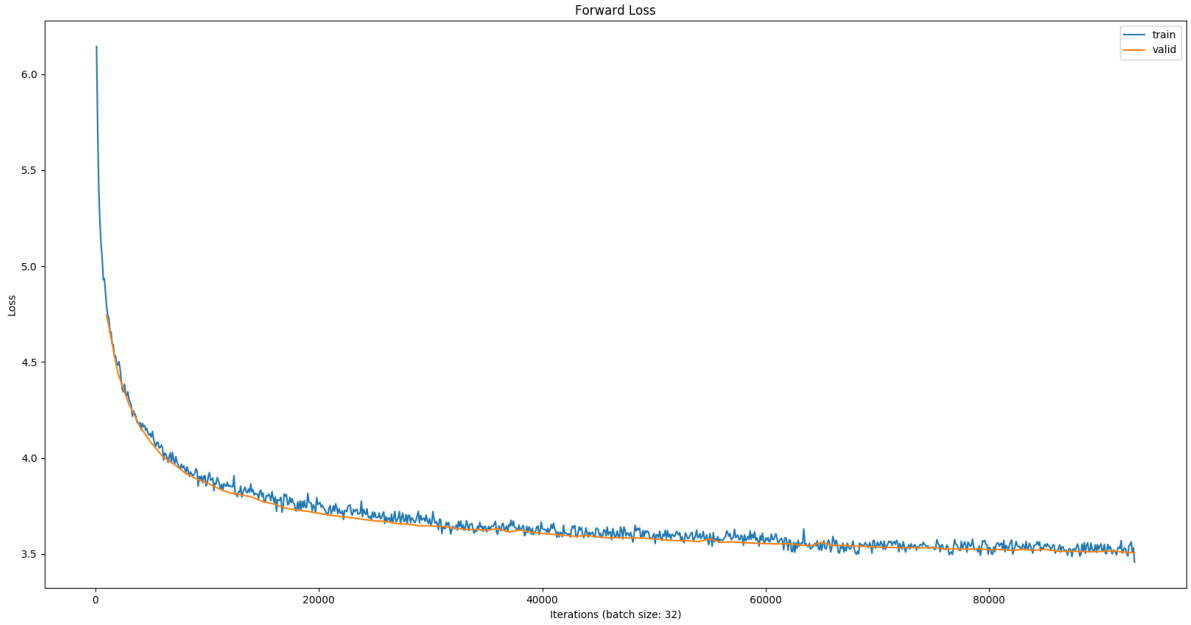
          LSTM2_b -> Projection2_b -> AdaptiveSoftmax

3. Hyperparameters of your ELMo model. (number of layers, hidden dimension, output dimension, optimization algorithm, learning rate and batch size) (0.4%)
    - CharEmbedding

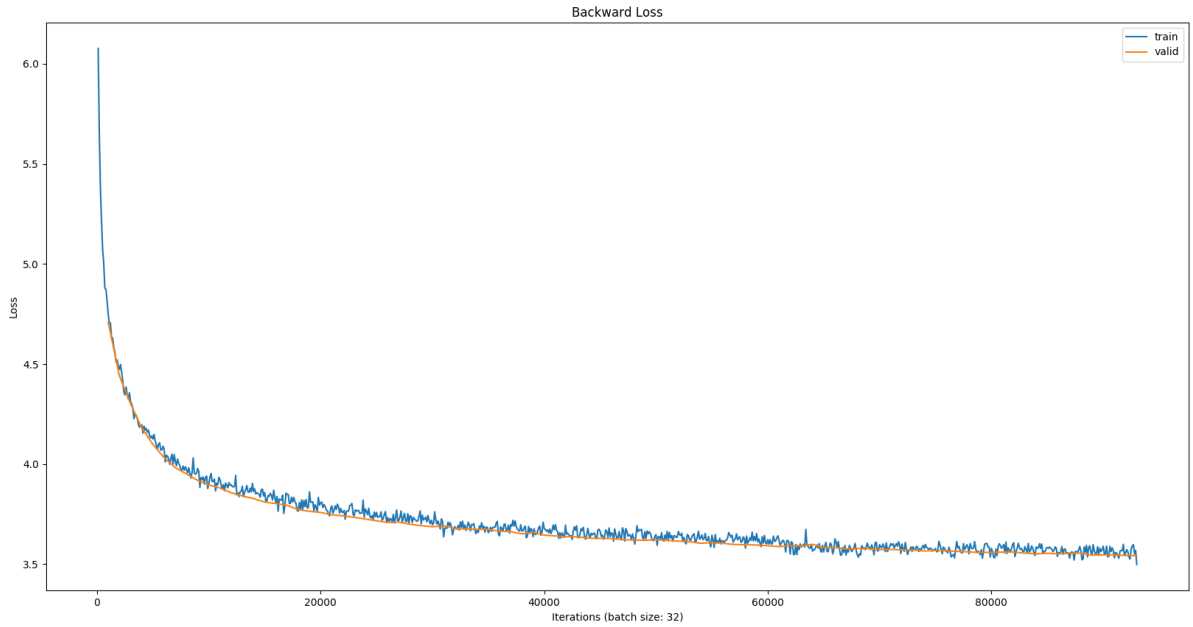
        - num of embeddings: 260 (consider all char(0-255) and $[SOS], [EOS], [PAD], [UNK]$)
        - embedding dimension: 16
        - convolution filters: [(1, 32), (2, 64), (3, 128), (4, 128), (5, 256), (6, 256), (7, 512)]
        - n highways: 2
        - projection dimension: 512
    - ELMo
        - number of layers: 2
        - hidden dimension: 2048
        - projection dimension: 512
        - output dimension: 80000
        - cutoffs in AdaptiveLogSoftmaxWithLoss: [20,200,1000,10000]
    - others
        - optimization algorithm: Adam
        - learning rate: 1e-3
        - batch size: 32

4. Plot the perplexity score or loss on train/dev set while training. (0.4%)

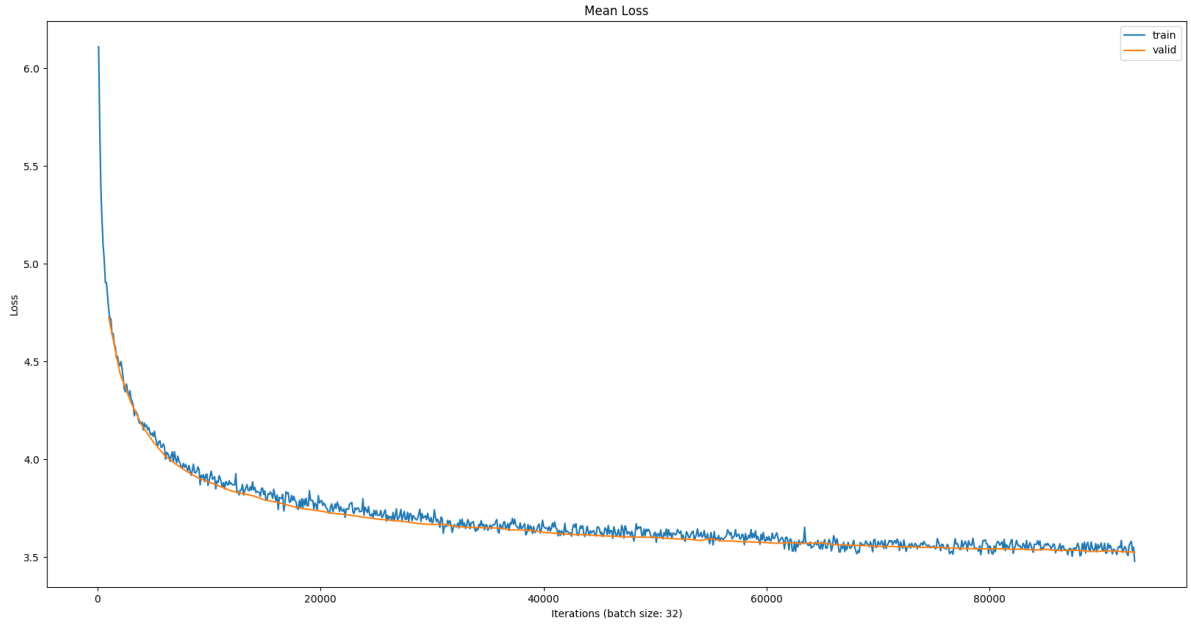
   Forward:



Backward:



Mean:



5. Show the performance of the BCN model with and without ELMo on the public leaderboard. (0.4%)

   The same settings:

   - random_seed: 126
   - iters: 93000

Model	**Validation Acc.**	**Public Acc.**	**Private Acc.**
With ELMo	**0.4496**	**0.46696**	**0.49049**
Without ELMo	0.4323	0.45429	0.45882

Q2: Compare different settings for ELMo. (2%)

1. Different number of training steps. (1%)

   You can train one model for large number of training steps, then take the intermediate checkpoints for this problem.

   The same settings:

   - random_seed: 126

Iters	Validation Acc.	Public Acc.	Private Acc.
24000 iters	**0.4559**	0.45520	0.46515
38000 iters	0.4478	0.46334	0.44434
58000 iters	0.4432	**0.46787**	0.48144
74000 iters	0.4550	0.45882	0.47782
93000 iters	0.4496	0.46696	**0.49049**

2. Different hyperparameters. (1%)

   The Same Setting:

   - First 1000K sentences
   - vocabulary size: 80000
   - epochs 2

   Compared (Small Size Elmo):

   - CharEmbedding
     - projection dimension: 128
   - ELMo
     - hidden dimension: 512
     - projection dimension: 128

Model	Validation Acc.	Public Acc.	Private Acc.
Origin	**0.4496**	0.46696	**0.49049**
Compared (small)	0.4441	**0.46787**	0.47330

Q3: Describe your model that passes strong baseline. (1%)

1. Input of your model. (0.4%)

   byte pair encoding

2. Model architecture. (0.4%)

   BERT embedding -> Pooling -> Linear to 5 classes

3. Hyperparameters of your model. (0.2%)
   - optimization: Adam
   - learning rate: 5e-6
   - batch size: 16
   - max length: 64

I used bert-large-cased model at first due to the better performance on the publice score board. I just used the model, optimizer and tokenizer offered by pytorch's version BERT, and personal preprocessing.

The performance and dsicussion will be compared with Q4, please see the following answer.

Q4: Describe your best model. (1%)

1. Describe your best model (0.5%)
2. 1. Input to your model

      byte pair encoding

   2. Model architecture

      Like the model above, but use uncased's version

   3. Hyperparameters
      - optimization: Adam
      - learning rate: 5e-6
      - batch size: 16
      - max length: 64
3. Describe the reason you think why your best model performs better than other models. (0.5%)

   Different with above the better model on this task is uncased version. See the following tables.

Model	Validation Acc.	Public Acc.	Private Acc.
Cased (64)	0.52091	0.52217	0.53484
Unased(64)	0.54182	0.54027	0.55022

   I also find that the max length of each sentence is critical, so after deadline, I do another experiment. Cause the limiation of GPU memory, the batch size is changed to 8.

   I find that the uncased's version of model have the better peroformance, but the double max length likely have no great improvement.

Model	Validation Acc.	Public Acc.	Private Acc.
Cased (128)	0.52636	0.52850	0.53755
Unased (128)	0.51818	0.52307	0.54841

For the cased v.s. uncased, during the training I find that the cased's version model would easily overfit, but uncased's version did not. I though that the cased word may confused the model and cased imformation may be not important at all. And the uncased's version model may have more robustness.

For the difference of max length of sentence, the large max length will improve the performance on longer sentence. But the short sentence performance may be lower. So, after experiment, the difference of max length did not a critical factor.

Q5: Compare different input embeddings. (1%)

In this homework, 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**.

	Pros	Cons
character embedding	1. Can handle nknown words and rare words (OOV). 2. Can catch the meaning of prefix, root and suffix. 3. For the low frequency words, it gets more imformation.	1. Subwords may be confused by model, like "her" in "where" and "superhero".
word embedding	1. Compress the word imformation and have numeric word meaning. 2. Widly used in many cased and work with other embedding together.	1. Can not handle unknown words and rare words 2. There are lots of the words have similar meanings, like run, ran, running. 3. Ther is not a perfect tokenize ways for all languages.
byte pair encoding	1. Split the unseen word into word piece, solving the OOV problem.	2. The word piece may not contain the related meaning of the origin words.

Q6: BERT (2%)

1. Please describe the tasks that BERT used for pre-training. What benefits do they have comparing to language model pretraining used in ELMo? (1%)

    - BERT can be fine-tuned on specified task, but ELMo should be fixed.
    - BERT use Transformer to encoding and use mask prediction to fixed the problem of bi-directional LSTM in ELMo, which may peek the next word.
    - Used only attention model without RNN or CNN, which can use parallel computing, optimize the speed of training.

2. Please describe in detail how you would formulate the problem in HW1 and apply BERT on it. (1%)

    The HW1 problem can be considered as a Next Sentence Prediction problem.

    So we can dirctly use the model in [pytorch-pretrained-BERT](), and concat the utterence and option with `"SEP"` and add `[CLS]` in the beginning to be a single sentence. ANd then, do the binary classification with `IsNext` and `NotNext` . The other details remain the same, like negative sampling, preprocessing ...

Q7: Bonus

1. Compare more than 2 contextualized embedding on this task. (1%)

Model	Validation Acc.	Public Acc.	Private Acc.
ELMo (Mine)	0.4496	0.46696	0.49049
ELMo (allennlp)	0.48047	0.48506	0.49321
BERT Cased (128)	0.52636	0.52850	0.53755
BERT Unased (64)	0.54182	0.54027	0.55022

2. Apply your ELMo embedding to other tasks. (1%)
    - CoNLL-2003 Named Entity Recognition
    - Reference:
        - https://github.com/yongyuwen/PyTorch-Elmo-BiLSTMCRF
        - https://github.com/juand-r/entity-recognition-datasets

Model	Validation Acc.	Public Acc.	Private Acc.
With ELMo (allennlp)	**0.4496**	**0.46696**	**0.49049**
With ELMo (mine)			
Without ELMo	0.4323	0.45429	0.45882

3. Apply any kind of contextualized embedding method on HW1 and report the performance. (1%)

   I use the ELMo embedding which training on the given corpus on HW1 instead of GloVe.

   But the speed of generating ELMo embedding is too slow, and the embbeding is too large that I can't save it temporaly.

   For the reference, I train the model on the GTX 1080, and it cost about 1day to train 1 epoch.

   And the accuracy in training data during this epoch is not as high as my origin best model.

   As a result, considering the time consuming and performance in first epoch, I thougth the performance is poor.