ADL HW1 Report

B04705026 資管四 林彥廷

- ## Q1: Data Usage

  1. Tokenization
     - Spacy tokenizer.
     - To speed up the pipeline, except for tokenizer, all the other function (ner, dependency parsing, etc…) are disabled.
     - If a sentence is empty or all words are discarded by tokenizer, the sentence is padded with “OOV”.
  2. Number of negative samples
     - # Positive : # Negative = **1:4**
  3. Truncation length of the utterances and the options.
     - Max number of tokens in **conversation**: **300**
     - Max number of tokens in **options**: **50**
     - Utterances are all concatenated
     - Did **NOT** include speaker information.
     - If “max number of tokens” not exceeded or the last “max number of tokens” were taken.
  4. Pre-trained embedding
     - FastText official pre-trained word embedding on Common Crawl with 2M word vectors and 300 dimension.

- # Q2: RNN w/o attention

    1. ## RNN w/o Attention Model Design
        - Context and option passed through **seperated rnn layers.**
        - To product fixed length output of rnn, max pooling on last gru output along gru dimension is needed.
        - In my experiments, **max pooling performed better** than avg pooling on rnn **at the cost of longer convergence time.**
        - Last hidden state of rnn is concatenated to capture whole sentence information.

BS = Batch Size
ML = max length of contexts/options in mini-batch
E = Embedding Dimension (300 with fasttext embedding)

Layer	Layer Dimension	Output Dimension		Dropout	Note
		Context	Option		
Input		BS x ML	BS x ML	x	Padded into fixed length
Embedding	300	BS x ML x E	BS x ML x E	0	Weight Freezed
Bi-GRU #1	256	BS x ML x 512	BS x ML x 512	0.2	Ignore hidden
Bi-GRU #2	256	BS x ML x 512	BS x ML x 512	0	
Max Pool		BS x 512	BS x 512	0	Pooling along GRU dimension
GRU #2 Hidden		BS x 512	BS x 512	0	Hidden state of last gru layer
Concat		BS x 1024	BS x 1024	0.2	Concat hidden and output after pooling
Bi-Linear	(1024, 1024)	BS x 1 (logit)		0	

2. Model Performance

Metrics \ Dataset	Training	Validation	Public LB	Private LB
Recall @ 10	1	0.3586	**0.7233**	**0.74**
Recall @ 5	1	0.3246	x	x
Recall @ 1	0.8159	0.2954	x	x
Loss (BCE)	0.2484	0.1464	x	x

3. Loss function

- **Binary Cross Entropy**

4. Optimization Algorithm, learning rate, batch size

Optimizer	Learning Rate	Batch Size	# Epoch converges to best val recall	Convergence time
Adam*	1e-3	100	5 epochs	~30min

* all the other optimizer parameters were set as default. (without Amsgrad)

- # Q3: RNN w/ Attention

1. ## RNN w/ Attention Model Design

Built upon rnn w/o attention with the following differences.

- Context and option passed through **independent rnn layers.**
- **Attention mechanism was applied on the outputs of second rnn layer.**
- **Option-aware context** and **context-aware option** went through the downstream rnn and bi-linear layers to calculate logits.
- **Scaled Inner-product** attention was adopt.

BS = Batch Size
ML = max length of contexts/options in mini-batch
E = Embedding Dimension (300 with fasttext embedding)

Layer	Layer Dimension	Output Dimension: Context	Output Dimension: Option	Dropout	Note
Input		BS x ML	BS x ML	x	Padded into fixed length
Embedding	300	BS x ML x E	BS x ML x E	0	Weight Freezed
Bi-GRU #1	256	BS x ML x 512	BS x ML x 512	0.2	Ignore hidden
Bi-GRU #2	256	BS x ML x 512	BS x ML x 512	0.2	
Attention					Scaled inner product Both way were attended
Bi-GRU #3	256	BS x ML x 512	BS x ML x 512	0	
Max Pool		BS x 512	BS x 512	0	Pooling along GRU dimension
GRU #3 Hidden		BS x 512	BS x 512	0	Hidden state of last gru layer
Concat		BS x 1024	BS x 1024	0.2	Concat hidden and output after pooling
Bi-Linear	(1024, 1024)	BS x 1 (logit)		0	

2. Model Performance

Metrics \ Dataset	Training	Validation	Public LB	Private LB
Recall @ 10	1	0.4866	**0.77**	**0.7833**
Recall @ 5	1	0.4524	x	x
Recall @ 1	0.8556	0.4302	x	x
Loss (BCE)	0.2142	0.1234	x	x

3. Loss function

- **Binary Cross Entropy**

4. Optimization Algorithm, learning rate, batch size

Optimizer	Learning Rate	Batch Size	# Epoch converges to best val recall	Convergence time
Adam*	1e-3	128	5 epochs	~100min

* all the other optimizer parameters were set as default. (without Amsgrad)

- # Q4: Best model

1. RNN w/ Attention Model Design

   Built upon rnn w/ attention with the following differences.

   - **ONLY Option-aware context** went through the downstream rnn and linear layers to calculate logits.
   - **Tri-linear attention function** was adopt.

BS = Batch Size
ML = max length of contexts/options in mini-batch
E = Embedding Dimension (300 with fasttext embedding)

Layer	Layer Dimension	Output Dimension		Dropout	Note
		Context	Option		
Input		BS x ML	BS x ML	x	Padded into fixed length
Embedding	300	BS x ML x E	BS x ML x E	0	Weight Freezed
Bi-GRU #1	256	BS x ML x 512	BS x ML x 512	0.2	Ignore hidden
Bi-GRU #2	256	BS x ML x 512	BS x ML x 512	0.2	
Attention					**Tri-linear attention**
Bi-GRU #3	256	BS x ML x 512	x	0	
Max Pool		BS x 512	x	0	Pooling along GRU dimension
GRU #3 Hidden		BS x 512	x	0	Hidden state of last gru layer
Concat		BS x 1024	x	0.2	Concat hidden and output after pooling
Linear	(1024, 1)	BS x 1 (logit)		0	

2. Model Performance

Metrics \ Dataset	Training	Validation	Public LB	Private LB
Recall @ 10	1	0.4878	**0.7933**	**0.8042**
Recall @ 5	1	0.4572	x	x
Recall @ 1	0.8544	0.4336	x	x
Loss (BCE)	0.2138	0.1606	x	x

3. Loss function
   - **Binary Cross Entropy**

4. Optimization Algorithm, learning rate, batch size

Optimizer	Learning Rate	Batch Size	# Epoch converges to best val recall	Convergence time
Adam*	1e-3	128	**6** epochs	~120min

* all the other optimizer parameters were set as default. (without Amsgrad)

5. Reason for Better Performance

Metric Comparison	Main Difference	Recall @ 1	Recall @ 10		
		Training	Validation *(relative)*	Public LB	Private LB
w/o Attention		0.8159	0.3586	0.7233	0.74
w/ Attention	*Attention*	0.8556	0.4866 *(+35%)*	0.77	0.7833
Best	*Tri-linear func.*	0.8544	0.4878 *(+0.24%)*	0.7933	0.8042

- w/ attention vs. w/o attention

Attention mechanism hugely improved model performance
since attention function interacted between context and option to incorporate
bi-directional information and attended to useful features.

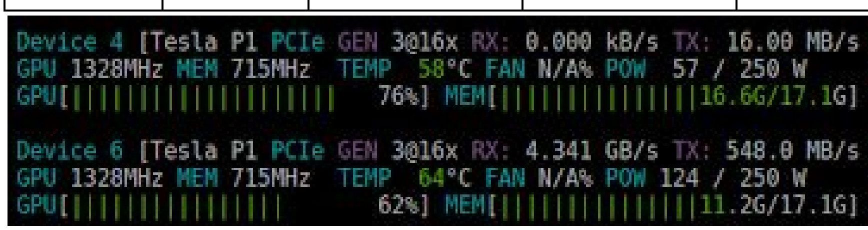
■ Best vs. w/ attention

The main reason for improvement was better attention flow. Tri-linear attention function were first adopt in [Bi-Daf](), a question answering ([SQuAD 1]()) model which was similar to our task (handing context and query relation).

• Q5: Compare GRU and LSTM model

Common Configuration		
Hardware	GPU	Nvidia Tesla P100
	CUDA	10.0
	CuDNN	7.4
Training	Base Model	RNN w/o Attention
	Batch Size	512
	# RNN Layers	2
	Positive: Negative	1:4

Difference						
RNN Type	Memory Usage	Training Speed *(sec/batch)*	Inference Speed *(sec/batch)*	Recall@10		
				Validation	Public LB	Private LB
LSTM	11.2G	0.9708	2.86	0.3094	0.7133	0.6942
GRU	16.6G	0.9900	2.33	0.3496	0.6933	0.7442



1. Recall@10 score

    Both model passed baseline score. GRU model outperformed LSTM model on validation set and private leaderboard.
    In my experience in HW1, GRU model usually **performed better** on validation set and **converged much faster** (2~3 epoch faster).

2. Required GPU Memory Usage

    Surprisingly, GRU consumed **more gpu memory** even with **less parameters**!
    I found an issues on Pytorch github regarding the same observation.
    It was not caused by bugs or memory leaks. Instead, ***"GRU has to store 6 values per input value for LSTM just 4 is enough"*** seemed to be an reasonable explanation supported by the fact that the memory usage ratio in our case was close to 6:4.

    No significant difference on GPU utilization rate.

3. Training & Testing Speed

    Both *by a little margin*, LSTM was faster on training and GRU was faster on inference.

- # Q6: Visualize the attention weights

    1. Take one example in the validation set and visualize the attention weights (after softmax).

        Applying softmax to attention weight matrix **row by row**.
        Padding token were discarded.

        The figure is presented below.
        Original figure was too long to fit, so only a little arrangement was made.

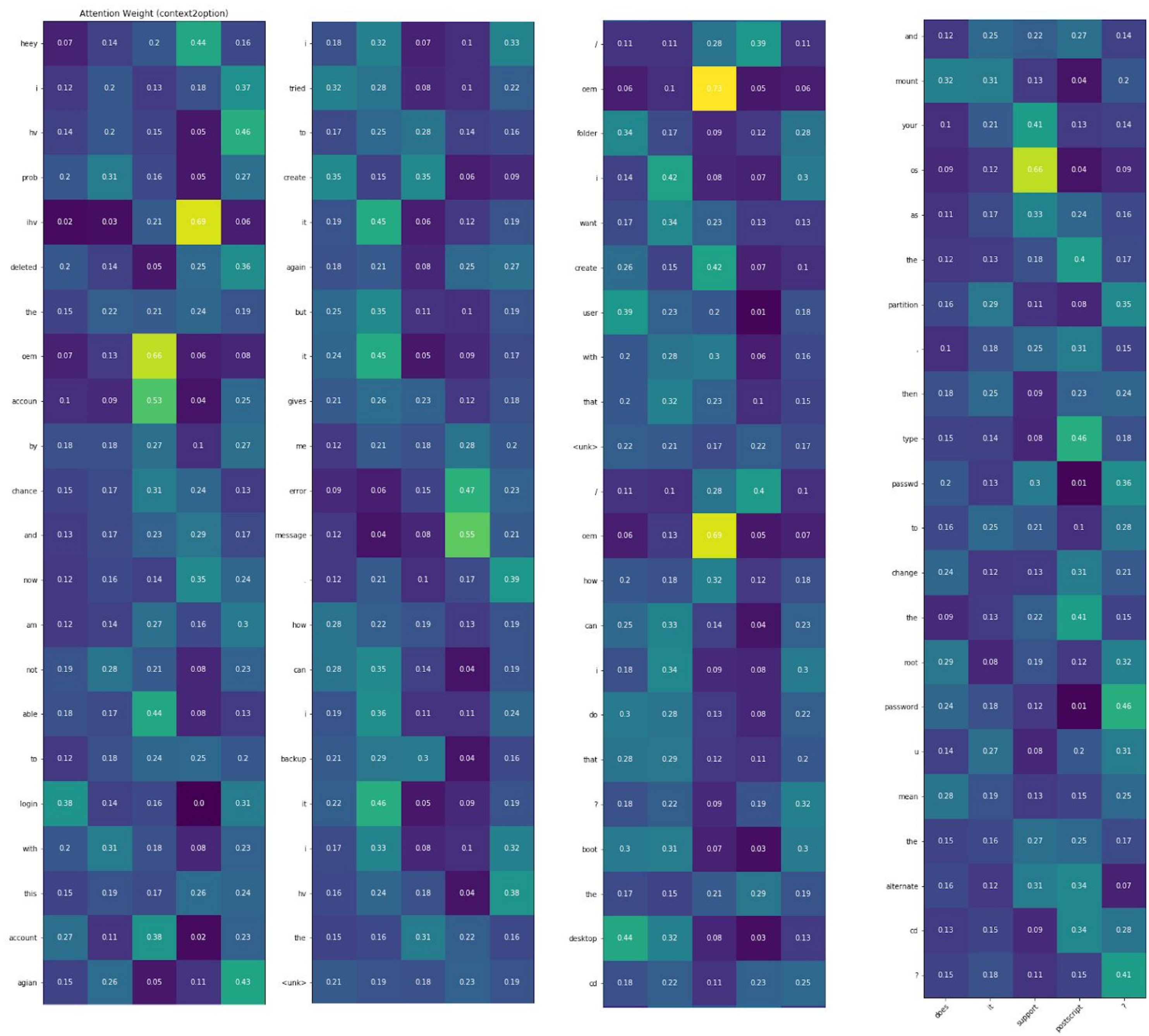
    2. Describe your findings. (1%)

        The option sentence is "Does it support postscript ?".

        The "Does", "it" and "?" are clearly *common stopwords* of little value for semantics understanding, and the **none of context tokens paid much attention to these common words**.

On the other hand, **“support” and “postscript” are supportive keywords** and are strongly attended by context tokens like “os”, “oem”, “error”.

In sum, the attention model did learn some critical relationship between keywords and even proprietaries and pay less attention to the unrelated words and stopwords.



- # Q7: Compare training result with different settings

Base architecture is rnn w/attention with only one layer encoding rnn and no dropout.

1. different reasonable loss functions. (1%)
   - Cross entropy calculates loss **with all samples logits**, while binary cross entropy treats all samples **independently**.
   - Convergence time of BCE is shorter.

Loss \ Recall@10	Training (@1)	Validation	#Epoch converges
Binary Cross Entropy	0.7223	0.4302	2
Cross Entropy	0.5622	0.2284	3

2. different number of negative samples. (1%)
   - Larger number of negative samples linearly increase training time.
   - Higher positive sample ratio handle the *class imbalance problem* better

#Negative \ Recall@10	Training (@1)	Validation	Train Time /Epoch
4	0.7223	0.4302	22min
9	0.4478	0.4048	42min

3. different number of utterances in a dialog.
   - Training time increase linearly with number of tokens
   - About 80%+ context length are within 150 tokens.
   - Lower context length reduced noise and decrease training time and GPU memory usage.

Max Context Length \ Recall@10	Training (@1)	Validation	Train Time /Epoch
300	0.7223	0.4302	22min
150	0.7596	0.4448	14min