ADL 2019 HW1 Report

r07922001 陳佳佑

Q1: Data preprocessing

I use the same preprocessing method for rnn.sh, attention.sh and best.sh.

Tokenization

- I just call the nltk word_tokenize

Negative sample ratio

- 4 : 1 (Negative : Positive)
- I have also tried 9 : 1, but it seems the same.

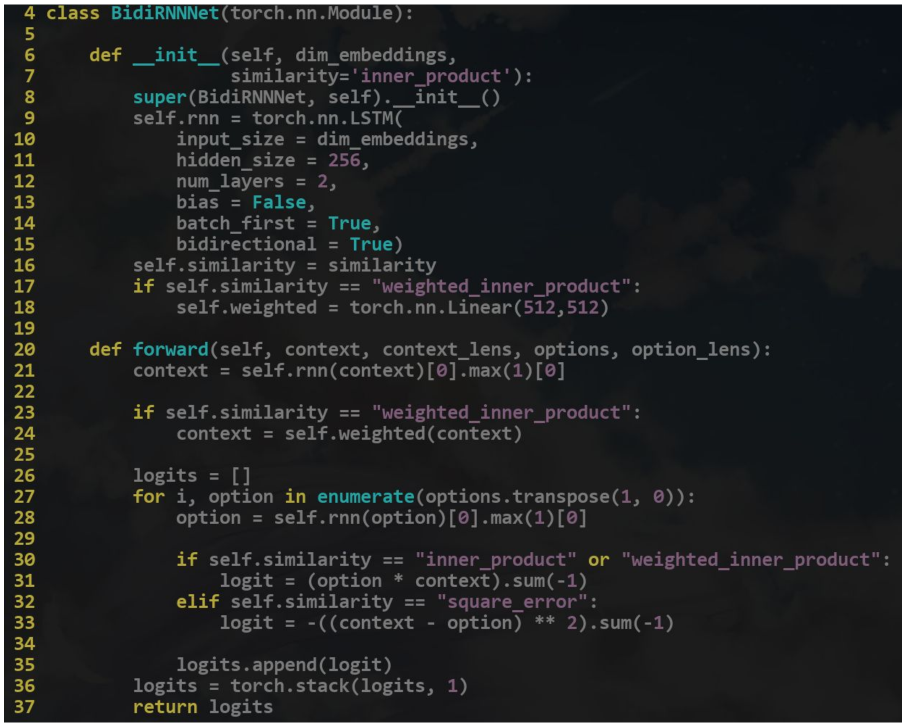
Truncation length

- 300 (context)
- 100 (option)

Pre-trained Embedding

- crawl-300d-2M.vec (based on fastText)

Q2: RNN w/o attention model



```python
class BidiRNNNet(torch.nn.Module):

    def __init__(self, dim_embeddings,
                 similarity='inner_product'):
        super(BidiRNNNet, self).__init__()
        self.rnn = torch.nn.LSTM(
            input_size = dim_embeddings,
            hidden_size = 256,
            num_layers = 2,
            bias = False,
            batch_first = True,
            bidirectional = True)
        self.similarity = similarity
        if self.similarity == "weighted_inner_product":
            self.weighted = torch.nn.Linear(512,512)

    def forward(self, context, context_lens, options, option_lens):
        context = self.rnn(context)[0].max(1)[0]

        if self.similarity == "weighted_inner_product":
            context = self.weighted(context)

        logits = []
        for i, option in enumerate(options.transpose(1, 0)):
            option = self.rnn(option)[0].max(1)[0]

            if self.similarity == "inner_product" or "weighted_inner_product":
                logit = (option * context).sum(-1)
            elif self.similarity == "square_error":
                logit = -((context - option) ** 2).sum(-1)

            logits.append(logit)
        logits = torch.stack(logits, 1)
        return logits
```

The above image is my implementation of rnn w/o attention model. I just use a simple simple LSTM with bidirectional to encode both the context and option. Then, I use the bilinear scoring function to get the logit of each option.

Performance

- 0.68 (validation set)
- 9.5 (public scoreboard)

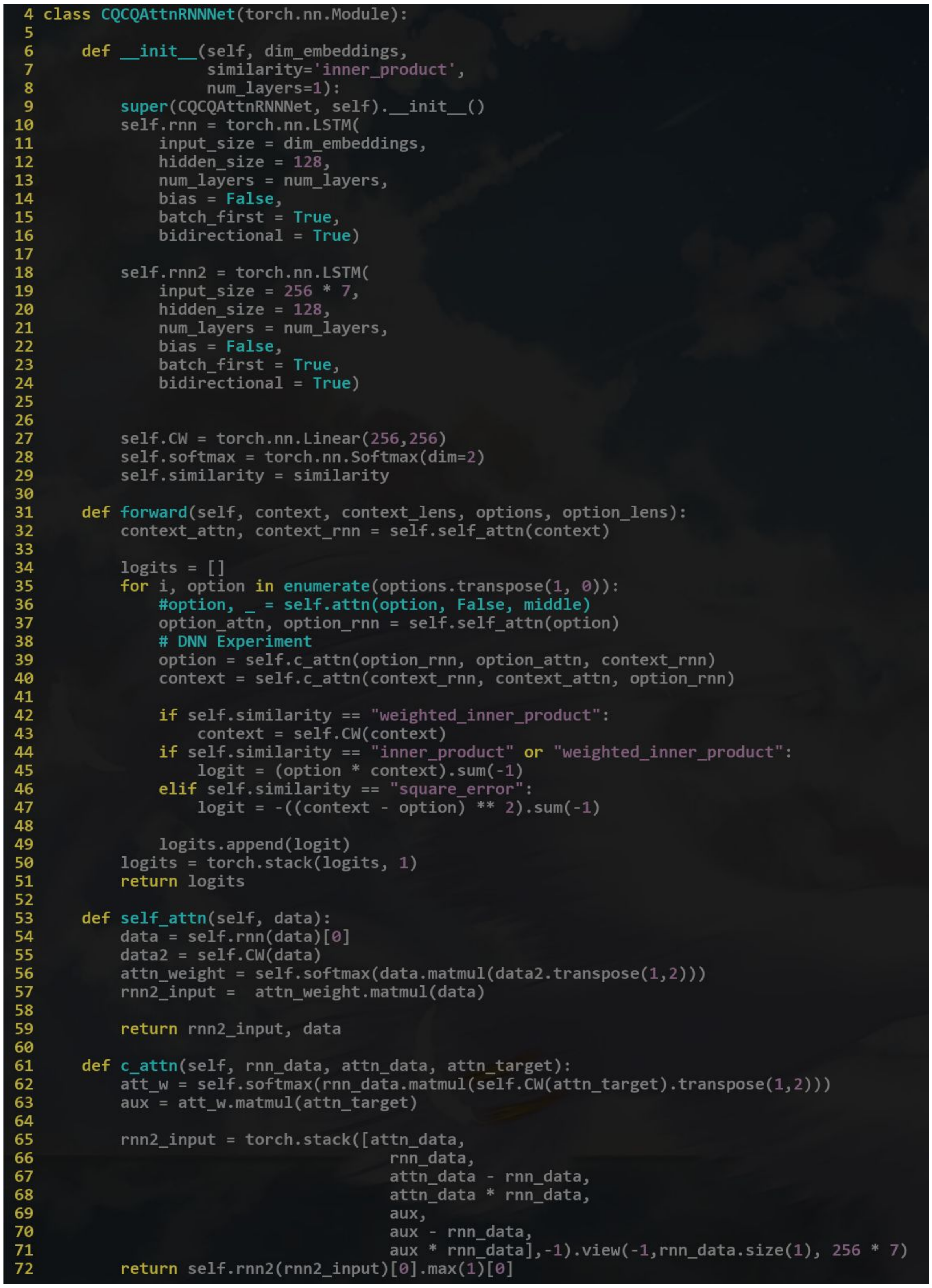
Loss function

- BCE Loss

Hyperparameter

- Adam optimizer, lr : 1e-3
- Epoch: 7
- Batch size: 64
- Context option scoring function: Bilinear

Q3: RNN w attention model



```python
class CQCQAttnRNNNet(torch.nn.Module):

    def __init__(self, dim_embeddings,
                 similarity='inner_product',
                 num_layers=1):
        super(CQCQAttnRNNNet, self).__init__()
        self.rnn = torch.nn.LSTM(
            input_size = dim_embeddings,
            hidden_size = 128,
            num_layers = num_layers,
            bias = False,
            batch_first = True,
            bidirectional = True)

        self.rnn2 = torch.nn.LSTM(
            input_size = 256 * 7,
            hidden_size = 128,
            num_layers = num_layers,
            bias = False,
            batch_first = True,
            bidirectional = True)


        self.CW = torch.nn.Linear(256,256)
        self.softmax = torch.nn.Softmax(dim=2)
        self.similarity = similarity

    def forward(self, context, context_lens, options, option_lens):
        context_attn, context_rnn = self.self_attn(context)

        logits = []
        for i, option in enumerate(options.transpose(1, 0)):
            #option, _ = self.attn(option, False, middle)
            option_attn, option_rnn = self.self_attn(option)
            # DNN Experiment
            option = self.c_attn(option_rnn, option_attn, context_rnn)
            context = self.c_attn(context_rnn, context_attn, option_rnn)

            if self.similarity == "weighted_inner_product":
                context = self.CW(context)
            if self.similarity == "inner_product" or "weighted_inner_product":
                logit = (option * context).sum(-1)
            elif self.similarity == "square_error":
                logit = -((context - option) ** 2).sum(-1)

            logits.append(logit)
        logits = torch.stack(logits, 1)
        return logits

    def self_attn(self, data):
        data = self.rnn(data)[0]
        data2 = self.CW(data)
        attn_weight = self.softmax(data.matmul(data2.transpose(1,2)))
        rnn2_input =  attn_weight.matmul(data)

        return rnn2_input, data

    def c_attn(self, rnn_data, attn_data, attn_target):
        att_w = self.softmax(rnn_data.matmul(self.CW(attn_target).transpose(1,2)))
        aux = att_w.matmul(attn_target)

        rnn2_input = torch.stack([attn_data,
                                  rnn_data,
                                  attn_data - rnn_data,
                                  attn_data * rnn_data,
                                  aux,
                                  aux - rnn_data,
                                  aux * rnn_data],-1).view(-1,rnn_data.size(1), 256 * 7)
        return self.rnn2(rnn2_input)[0].max(1)[0]
```

The above part is my implementation of attention model. At first, I have tried the model with only self-attention on both context and option. However, the result is just same as the bidirectional rnn. As a result, I concat the result of self attention and cross attention with some affine transformation. The result is quite great. However, cross-attention may break lots of parallelism. As a result, it requires much more time for it to work.

Performance

- 0.75 (validation set)
- 9.38666 (public scoreboard)

Loss function

- BCE Loss

Hyperparameter

- Adam optimizer, lr : 1e-3
- Epoch: 3-6 (epoch ensemble)
- Batch size: 64
- Context option scoring function: Bilinear
- Layer: 2

Q4: Best Model

My best model is as the same as the attention one. I have tried lots of other model without attention. However, none of it beats the performance of the one with cross attention. As a result, all the following one is same as the attention one.

Performance

- 0.75 (validation set)
- 9.386 (public scoreboard)

Loss function

- BCE Loss

Hyperparameter

- Adam optimizer, lr : 1e-3
- Epoch: 3-6 (epoch ensemble)
- Batch size: 64
- Context option scoring function: Bilinear

Why attention model beats the vanilla rnn one?

- As the finding in Q3, self-attention may not help the rnn model but cross attention may not in this task. This implies that context/option getting "leaked" information from the other one could help a lot. In other words, in original rnn model, context and option only interacts with each other in the last bilinear scoring function. And this may highly reduce the ability to fit.

Some other try but fail

- I have tried to change the bilinear scoring function to other functions such as mlp, inner-product, negative rmse, cosine-similarity. However, all of them fail and gives out really poor performance.
    - The fail for inner-product is quite trivial, since it's the bilinear scoring function with W as the identity matrix.
- I have also tried to extend the model complexity of W in the bilinear scoring function. However, the performance doesn't improve and the speed slows a lot.

Q5: GRU vs LSTM RNN Model

For this comparison, I reuse the model in Q2 and simply change the LSTM GRU to compare.

Performance

- 0.67 (GRU epoch 6, validation set)
- 0.68 (LSTM epoch 7, validation setc)

GPU Memory

- 1883 MB (GRU)
- 1635 MB (LSTM)

Training Speed (per epoch)

- 3 min 42 sec (GRU)
- 4 min 20 sec (LSTM)

Testing Speed

- 8 sec (GRU)
- 9 sec (LSTM)

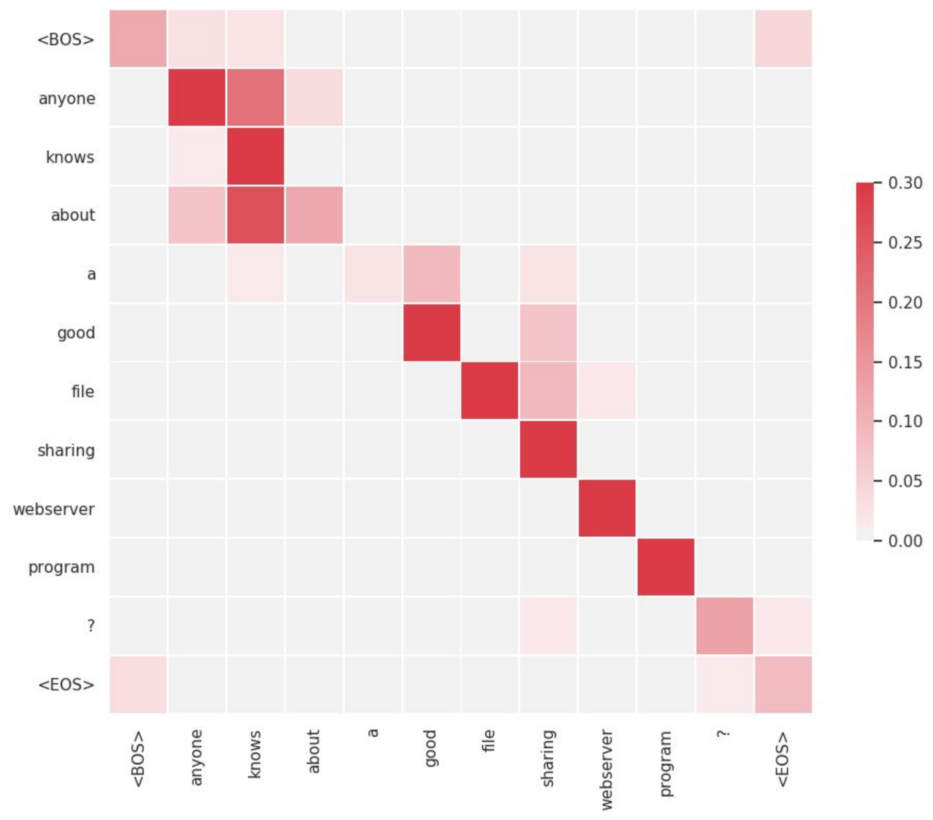
Hyperparameter

- Adam optimizer, lr : 1e-3
- Batch size: 64
- Context option scoring function: Bilinear

Summary

- Since GRU has less gates than LSTM, it's faster than LSTM. And surprisingly, they have almost the same performance. In the other words, GRU may be better than LSTM in this task. By the way, it's quite strange that LSTM cost less GPU memory than GRU. I'm not sure that isn't it any special optimization in the implementation in pytorch. However, the difference is pretty small.

Q6: Visualize the attention weight



Findings

- BOS and EOS contribute to each other. It quite makes sense, since in the grammar <BOS> and <EOS> has similar meaning that represents the location of a sentence.
- Almost all the attention weight contribute to themselves. However, 'a' doesn't contribute quite much too itself. The possible reason maybe that such article (a, the, an) doesn't matter the meaning of the sentence. For example, lots of non native speaker may forget to add the article (a, the, an) in their writing. However, we could still understand what they are talking about.

Reference: **https://seaborn.pydata.org/examples/many_pairwise_correlations.html**

Q7: Compare different settings

Different loss function

- KLDivLoss: Doesn't converge
    - KLDivLoss assumes that the logits and the labels are two probability distribution and measures the distance between them.
- HingeLoss: Doesn't converge
    - HingeLoss is widely used in multi-class multi-label settings, however, it doesn't seem to work well in this task.

Different number of negative sample

- 9 : 1
    - Much more slower, performance (0.66 @ RNN in Q2) is the same.
- 4 : 1
    - Faster, same performance (0.66 @ RNN in Q2) as 9 : 1.
- Conclusion
    - 4:1 is good enough, it's faster and have the same performance.

Different number of utterances in dialog

- 300 - 100
    - 0.66 @RNN in Q2 (pubic scoreboard 9.5)
- 600 - 100
    - 0.69 @ RNN in Q2 (Poor performance on public scoreboard 9.6)

Different pre-trained word embedding

- Crawl-300d-2M.vec
    - 0.66 (RNN in Q2
- Wiki-news-300d-1M.vec
    - 0.66 (RNN in Q2
- It looks like they have the same performance. Though 'Crawl-300d-2M.vec' has much more vocabulary, it seems like 'wiki-news-300d-1M.vec' has covered those needed in this task.