CSC413: Programming Assignment 3

Tianyu Du (1003801647)

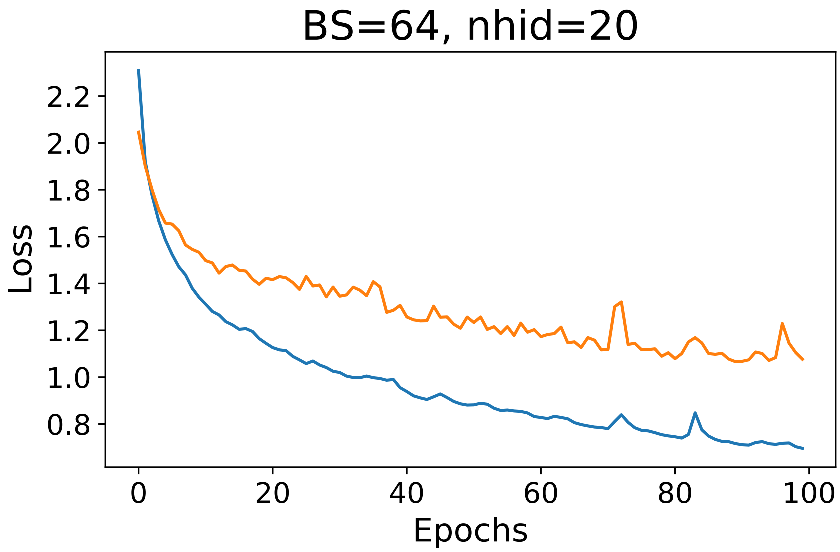
2020/03/16 at 22:51:38

1 Part 1: Gated Recurrent Units

1.1

```python
class MyGRUCell(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(MyGRUCell, self).__init__()

        self.input_size = input_size
        self.hidden_size = hidden_size

        # ------------
        # FILL THIS IN
        # ------------
        ## Input linear layers
        self.Wiz = nn.Linear(input_size, hidden_size, bias=False)
        self.Wir = nn.Linear(input_size, hidden_size, bias=False)
        self.Win = nn.Linear(input_size, hidden_size, bias=False)

        ## Hidden linear layers
        self.Whz = nn.Linear(hidden_size, hidden_size, bias=True)
        self.Whr = nn.Linear(hidden_size, hidden_size, bias=True)
        self.Whn = nn.Linear(hidden_size, hidden_size, bias=True)


    def forward(self, x, h_prev):
        # ------------
        # FILL THIS IN
        # ------------
        r = torch.sigmoid(self.Wir(x) + self.Whr(h_prev))
        z = torch.sigmoid(self.Wiz(x) + self.Whz(h_prev))
        g = torch.tanh(self.Win(x) + r * self.Whn(h_prev))
        h_new = (1 - z) * g + z * h_prev
        return h_new
```

1.2



1.3

Failure Type 1 Long vocabularies. The model fails to translate long vocabularies, the middle part of vocabularies get messed up.

```
source:     computer science
translated: opcorchyway ipenceway
```

Failure Type 2 Vocabularies containing dashes ("-"). The model fails to distinguish parts of word before and after the dash. Sometime, the dash is missing after translation.

```
source:     electric-powered
translated: elcercorstentway
================================
source:     to-buy
translated: otay-othay
```

2 Additive Attention

2.1

$$\tilde{\alpha}_i^{(t)} = f(Q_i, K_i) = W_2 \ \mathrm{ReLU}(W_1[Q_i, K_i] + b_1) + b_2 \tag{2.1}$$

$$\alpha_i^{(t)} = \mathrm{softmax}(\tilde{\alpha}^{(t)})_i = \frac{\exp(\tilde{\alpha}_i^{(t)})}{\sum_{t=1}^{\texttt{seq\_len}} \exp(\tilde{\alpha}_i^{(t)})} \tag{2.2}$$

$$c_t = \sum_{t=1}^{\texttt{seq\_len}} \alpha_i^{(t)} K_i \tag{2.3}$$

2.2

```python
class RNNAttentionDecoder(nn.Module):
    def __init__(self, vocab_size, hidden_size, attention_type='scaled_dot'):
        super(RNNAttentionDecoder, self).__init__()
        self.vocab_size = vocab_size
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding(vocab_size, hidden_size)

        self.rnn = MyGRUCell(input_size=hidden_size*2, hidden_size=hidden_size)
        if attention_type == 'additive':
          self.attention = AdditiveAttention(hidden_size=hidden_size)
        elif attention_type == 'scaled_dot':
          self.attention = ScaledDotAttention(hidden_size=hidden_size)

        self.out = nn.Linear(hidden_size, vocab_size)


    def forward(self, inputs, annotations, hidden_init):
        """Forward pass of the attention-based decoder RNN.

        Arguments:
            inputs: Input token indexes across a batch for all the time step. (batch_size x decoder_seq_len)
            annotations: The encoder hidden states for each step of the input.
                         sequence. (batch_size x seq_len x hidden_size)
            hidden_init: The final hidden states from the encoder, across a batch. (batch_size x hidden_size)

        Returns:
            output: Un-normalized scores for each token in the vocabulary, across a batch for all the decoding time steps. (batch_size x decoder_seq_len x vocab_size)
            attentions: The stacked attention weights applied to the encoder annotations (batch_size x encoder_seq_len x decoder_seq_len)
        """

        batch_size, seq_len = inputs.size()
        embed = self.embedding(inputs)  # batch_size x seq_len x hidden_size

        hiddens = []
        attentions = []
        h_prev = hidden_init
        for i in range(seq_len):
            # ------------
            # FILL THIS IN - START
            # ------------
            embed_current = embed[:,i,:]  # Get the current time step, across the whole batch
            context, attention_weights = self.attention(
                h_prev, # queries @ (bs, hidden_size)
                annotations, # keys @ (bs, sl, hs)
                annotations # values @ (bs, sl, hs)
            )  # @ (batch_size, 1,  hidden_size) and (batch_size, seq_len, 1)
            embed_and_context = torch.cat((
                embed_current.view(batch_size, -1),
                context.view(batch_size, -1)),
                dim=1
            )  # batch_size x (2*hidden_size)
```

```
            h_prev = self.rnn(embed_and_context, h_prev)  # batch_size x hidden_size
            # ------------
            # FILL THIS IN - END
            # ------------

            hiddens.append(h_prev)
            attentions.append(attention_weights)

        hiddens = torch.stack(hiddens, dim=1) # batch_size x seq_len x hidden_size
        attentions = torch.cat(attentions, dim=2) # batch_size x seq_len x seq_len

        output = self.out(hiddens)  # batch_size x seq_len x vocab_size
        return output, attentions
```

3 Scaled Dot Product Attention

3.1 Implementations

3.1.1 ScaledDotAttention

```
class ScaledDotAttention(nn.Module):
    def __init__(self, hidden_size):
        ...

    def forward(self, queries, keys, values):
        """..."""

        # ------------
        # FILL THIS IN
        # ------------
        hidden_size = self.hidden_size
        batch_size = queries.shape[0]
        d = hidden_size
        # Convert tensor to 3D.
        # k is the number of queries.
        queries = queries.view(batch_size, -1, hidden_size)
        num_queries = queries.shape[1]
        seq_len = keys.shape[1]
        # Expand.
        # keys = keys.expand(batch_size, seq_len, hidden_size)
        # keys = torch.transpose(keys, dim0=0, dim1=1)

        q = self.Q(queries) # @ (batch_size, k, hidden_size)
        k = self.K(keys) # @ (batch_size, seq_len, hidden_size)
        v = self.V(values) # @ (batch_size, seq_len, hidden_size)
        q = torch.transpose(q, 1, 2) # @ (batch_size, hidden_size, k)
        # print("q @", q.shape)
        # print("k @", k.shape)
        unnormalized_attention = torch.bmm(k, q) * self.scaling_factor
        # unnormalized_attention @ (batch_size, seq_len, k)
        # print(unnormalized_attention.shape)

        attention_weights = self.softmax(unnormalized_attention).transpose(1, 2) # @ (
    batch_size, k, seq_len)
        context = torch.bmm(attention_weights, v) # @ (batch_size, k, hidden_size)
        attention_weights = attention_weights.transpose(1, 2) # @ (batch_size, seq_len, k)
        return context, attention_weights
```

3.1.2 CausalScaledDotAttention

```python
class CausalScaledDotAttention(nn.Module):
    def __init__(self, hidden_size):
        ...

    def forward(self, queries, keys, values):
        """..."""

        # ------------
        # FILL THIS IN
        # ------------
        hidden_size = self.hidden_size
        batch_size = queries.shape[0]
        d = hidden_size
        # Convert tensor to 3D.
        # k is the number of queries.
        queries = queries.view(batch_size, -1, hidden_size)
        num_queries = queries.shape[1]
        seq_len = keys.shape[1]
        # keys = keys.expand(batch_size, seq_len, hidden_size)
        # keys = torch.transpose(keys, dim0=0, dim1=1)

        q = self.Q(queries) # @ (batch_size, k, hidden_size)
        k = self.K(keys) # @ (batch_size, seq_len, hidden_size)
        v = self.V(values) # @ (batch_size, seq_len, hidden_size)
        q = torch.transpose(q, 2, 1) # @ (batch_size, hidden_size, k)
        # print("q @", q.shape)
        # print("k @", k.shape)
        unnormalized_attention = torch.bmm(k, q) * self.scaling_factor
        # unnormalized_attention @ (batch_size, seq_len, k)
        # print(unnormalized_attention.shape)
        # ==== Enforce Casual ====
        mask = torch.tril(torch.ones_like(unnormalized_attention)) * self.neg_inf
        unnormalized_attention += mask
        # ==== End ====
        attention_weights = self.softmax(unnormalized_attention).transpose(1, 2) # @ (
    batch_size, k, seq_len)
        context = torch.bmm(attention_weights, v) # @ (batch_size, k, hidden_size)
        attention_weights = attention_weights.transpose(1, 2) # @ (batch_size, seq_len, k)
        return context, attention_weights
```

3.1.3 TransformerEncoder

```python
class TransformerEncoder(nn.Module):
    def __init__(self, vocab_size, hidden_size, num_layers, opts):
        ...

    def forward(self, inputs):
        """..."""

        batch_size, seq_len = inputs.size()
        # ------------
        # FILL THIS IN - START
        # ------------
        encoded = self.embedding(inputs)  # @ (batch_size, seq_len, hidden_size)
        for i in range(self.num_layers):
            new_annotations, self_attention_weights = self.self_attentions[i](
                annotations, annotations, annotations
```

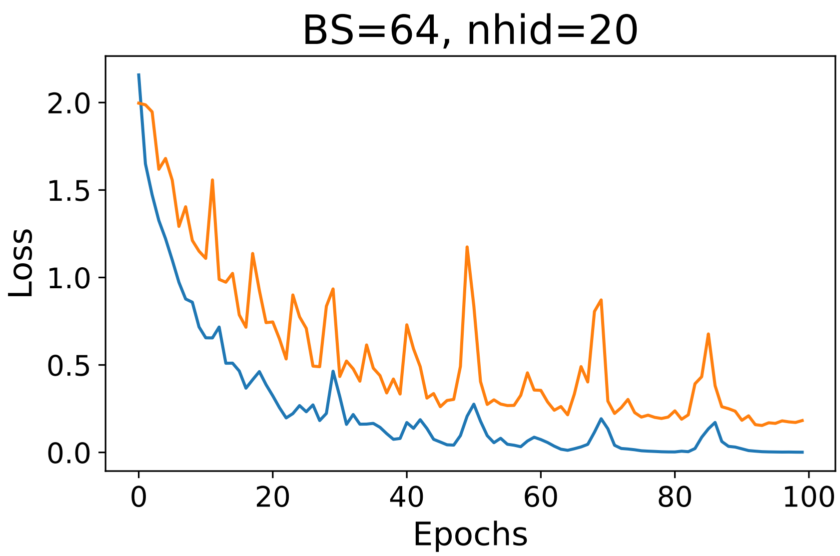
```
            )  # batch_size x seq_len x hidden_size
            # annotation with residual added.
            residual_annotations = annotations + new_annotations
            new_annotations = self.attention_mlps[i](residual_annotations)
            # Update annotations, the output of this layer.
            annotations = residual_annotations + new_annotations
        # ------------
        # FILL THIS IN - END
        # ------------

        # Transformer encoder does not have a last hidden layer.
        return annotations, None

    def create_positional_encodings(self, max_seq_len=1000):
        ...
```

3.1.4 TransformerDecoder

```
class TransformerDecoder(nn.Module):
    def __init__(self, vocab_size, hidden_size, num_layers):
        ...

    def forward(self, inputs, annotations, hidden_init):
        """..."""

        batch_size, seq_len = inputs.size()
        embed = self.embedding(inputs)  # batch_size x seq_len x hidden_size

        # THIS LINE WAS ADDED AS A CORRECTION.
        embed = embed + self.positional_encodings[:seq_len]

        encoder_attention_weights_list = []
        self_attention_weights_list = []

        # Decoder: the input fed to the first layer.
        contexts = embed # batch_size x seq_len x hidden_size
        for i in range(self.num_layers):
            # ------------
            # FILL THIS IN - START
            # ------------
            new_contexts, self_attention_weights = self.self_attentions[i](
                contexts, contexts, contexts
            ) # batch_size x seq_len x hidden_size
            residual_contexts = contexts + new_contexts
            new_contexts, encoder_attention_weights = self.encoder_attentions[i](
                residual_contexts, annotations, annotations
            ) # batch_size x seq_len x hidden_size
            residual_contexts = residual_contexts + new_contexts
            new_contexts = self.attention_mlps[i](residual_contexts)
            contexts = residual_contexts + new_contexts

            # ------------
            # FILL THIS IN - END
            # ------------

            encoder_attention_weights_list.append(encoder_attention_weights)
            self_attention_weights_list.append(self_attention_weights)

        output = self.out(contexts)
        encoder_attention_weights = torch.stack(encoder_attention_weights_list)
```

```python
        self_attention_weights = torch.stack(self_attention_weights_list)

        return output, (encoder_attention_weights, self_attention_weights)

    def create_positional_encodings(self, max_seq_len=1000):
        ...
```

3.2 Question 5: Training and Validation Plots



Training logs of the last few steps from three models are reported below, the validation loss of transformer is significantly lower than the previous two decoders. However, the translation results from additive attention is overall better. Additive attention model failed to translate one word (conditioning), but the transformer only translated conditioning correctly, it could be that additive attention works better for short words (because it's recurrent and might suffers from gradient vanishing/exploding problems) but transformer works better for long vocabularies (because it reads the entire sequence the same time).

```
================ GRU ================
Epoch:  95 | Train loss: 0.650 | Val loss: 1.069 | Gen: ethay airway onintoidingsday isway oulgefray
Epoch:  96 | Train loss: 0.647 | Val loss: 1.048 | Gen: ethay ariway onsidtoingray isway oulfrway
Epoch:  97 | Train loss: 0.647 | Val loss: 1.120 | Gen: ethay aringpay ondintingshingbay isway orkingway
Epoch:  98 | Train loss: 0.670 | Val loss: 1.123 | Gen: ethay aringpay onsidtenfay-onsay isway orkgingway
Epoch:  99 | Train loss: 0.673 | Val loss: 1.053 | Gen: ethay aisray onsiditiongray issway oulfreday
================ Additive Attention ================
Epoch:  95 | Train loss: 0.006 | Val loss: 0.138 | Gen: ethay airway onditioningcay isway orkingway
Epoch:  96 | Train loss: 0.006 | Val loss: 0.133 | Gen: ethay airway onditioningcay isway orkingway
Epoch:  97 | Train loss: 0.006 | Val loss: 0.137 | Gen: ethay airway onditioningcay isway orkingway
Epoch:  98 | Train loss: 0.056 | Val loss: 1.192 | Gen: ethay airway ondicecgcay isway orkiwway
```

```
Epoch:  99 | Train loss: 0.123 | Val loss: 0.332 | Gen: ethay airway onditionwway isway orkingway
================ Transformer (Enforcing Causal) ================
Epoch:  95 | Train loss: 0.002 | Val loss: 0.166 | Gen: ethhay iarway onditioningcay iseway orkingway
Epoch:  96 | Train loss: 0.002 | Val loss: 0.180 | Gen: ethhay iirway onditioningcay isiiiiiiiiiissssacy orkingwaay
Epoch:  97 | Train loss: 0.002 | Val loss: 0.175 | Gen: ethhay iirway onditioningcay iswway orkingway
Epoch:  98 | Train loss: 0.002 | Val loss: 0.171 | Gen: ethhay iirway onditioningcay iswway orkingway
Epoch:  99 | Train loss: 0.001 | Val loss: 0.182 | Gen: ethhay iirway onditioningcay iswway orkingway
```

3.3 Question 6: Non-causal Decoder

The outputs from the last few training iterations suggested the modified transformer achieves both lower training and validation loss compared with the original transformer. However, the generated translation is non-sense compared with the transformer with causal decoder. The can be resulted from the fact that, without enforcing causal mask, we allow the model to peak into the future, which discourages the decoder from learning the sequential structure of sentences (i.e., the model failed to learn the importance of character orders).

```
================ Output From Transformer with Causal Decoder ================
Epoch:  95 | Train loss: 0.002 | Val loss: 0.166 | Gen: ethhay iarway onditioningcay iseway orkingway
Epoch:  96 | Train loss: 0.002 | Val loss: 0.180 | Gen: ethhay iirway onditioningcay isiiiiiiiiiissssacy orkingwaay
Epoch:  97 | Train loss: 0.002 | Val loss: 0.175 | Gen: ethhay iirway onditioningcay iswway orkingway
Epoch:  98 | Train loss: 0.002 | Val loss: 0.171 | Gen: ethhay iirway onditioningcay iswway orkingway
Epoch:  99 | Train loss: 0.001 | Val loss: 0.182 | Gen: ethhay iirway onditioningcay iswway orkingway
================ Output From Transformer with Normal Decoder ================
Epoch:  95 | Train loss: 0.000 | Val loss: 0.001 | Gen: - - - - -
Epoch:  96 | Train loss: 0.000 | Val loss: 0.001 | Gen: - - - - -
Epoch:  97 | Train loss: 0.000 | Val loss: 0.001 | Gen: - - - - -
Epoch:  98 | Train loss: 0.000 | Val loss: 0.001 | Gen: - - - - -
Epoch:  99 | Train loss: 0.000 | Val loss: 0.001 | Gen: - - - - -
```

3.4 Question 7: Advantages and Disadvantages of Additive Attentions and Scaled Dot Product Attention

It seems that the scaled dot attention is better at translating long vocabularies, since a transformer takes the entire sequence of characters once, and can better exploit the correlation between characters distant apart from each other. Additive attention models is based on recurrent neural networks, and RNNs may suffer from vanishing and exploding gradient problems, depends on the specific types of RNN cell used. Therefore, RNN together with additive attentions works better for short vocabularies.

4 BERT

4.1 Question 1

The `BertCSC413_MLP` class uses 512 hidden neurones (in contrast to the 768 hidden neurones in the original implementation), and a sigmoid activation function (in contrast to the ReLU activation).

4.2 Question 2

4.3 Question 3

```
[72]  what_is("twelve minus fourteen")
```
negative

```
[73]  what_is("twelve plus fourteen")
```
positive

```
[74]  what_is("eight plus thousand")
```
positive

```
[75]  what_is("eight minus thousand")
```
negative

```
[76]  what_is("thousand minus eight")
```
positive

```
[77]  what_is("eight minus thousand")
```
negative

```
[78]  what_is("1 minus 14")
```
negative

```
[79]  what_is("1 minus two") # interesting.
```
negative

```
[80]  what_is("one minus two")
```
negative

```
[81]  what_is("three minus two minus eight")
```
negative

```
[82]  what_is("three minus two")
```
positive

```
[83]  what_is("one minus one minus one")
```
positive

```
[84]  what_is("one minus one minus one plus ten")
```
positive

```
[85]  what_is("one minus one plus ten minus one")
```
positive

```
[86]  what_is("minus three plus eight")
```
positive

These inference tasks involves both standard usages of binary operator (i.e., number + operator + number) and longer compound usages (i.e., using multiple binary operators consecutively). Moreover, three types of representations of numbers are used: plain English(e.g., three), plain numerical (e.g., 3), and English multipliers (e.g., thousand). Interestingly, the model processes ambiguous compound operations differently, for example `"three minus two minus eight"` is interpreted as $3 - 2 - 8 < 0$, but `"one minus one minus one"` as $1 - (1 - 1) > 0$.

4.4 Question 4

I changed some hyper-parameters to the training of `model_finetune_bert` by reducing the learning rate while increasing the number of training epochs. Specifically, learning rate is changed to `5e-6` (originally `2e-5`) and the model is now trained for 6 epochs (originally 4 epochs). The (overall) validation accuracy improved from 97% to 98%. The number of incorrect predictions in validation set reduced from two instances to one instance. The detailed training logs and validation reports are attached below:

```
==========================================================
                  Old Hyperparameters
==========================================================
======== Epoch 1 / 4 ========
Training...

  Average training loss: 1.08
  Training epcoh took: 0:01:20
Running Validation...
  Accuracy: 0.88
  Validation took: 0:00:01

======== Epoch 2 / 4 ========
Training...

  Average training loss: 0.79
  Training epcoh took: 0:01:19
Running Validation...
  Accuracy: 0.98
  Validation took: 0:00:01

======== Epoch 3 / 4 ========
Training...

  Average training loss: 0.59
  Training epcoh took: 0:01:19
Running Validation...
  Accuracy: 0.97
  Validation took: 0:00:01

======== Epoch 4 / 4 ========
Training...

  Average training loss: 0.53
  Training epcoh took: 0:01:19
Running Validation...
  Accuracy: 0.97
  Validation took: 0:00:01

Training complete!

Predicting labels for 160 test sentences...
```

```
Number of expressions with negative result 47
 45  predicted correctly , accuracy  0.9574468085106383

Number of expressions with 0 result 2
 0  predicted correctly , accuracy  0.0

Number of expressions with positive result 111
 111  predicted correctly , accuracy  1.0

==========================================================
                  New Hyperparameters
==========================================================
======== Epoch 1 / 6 ========
Training...

  Average training loss: 0.49
  Training epcoh took: 0:01:27
Running Validation...
  Accuracy: 0.98
  Validation took: 0:00:01

======== Epoch 2 / 6 ========
Training...

  Average training loss: 0.45
  Training epcoh took: 0:01:26
Running Validation...
  Accuracy: 0.98
  Validation took: 0:00:01

======== Epoch 3 / 6 ========
Training...

  Average training loss: 0.41
  Training epcoh took: 0:01:26
Running Validation...
  Accuracy: 0.98
  Validation took: 0:00:01

======== Epoch 4 / 6 ========
Training...

  Average training loss: 0.38
  Training epcoh took: 0:01:28
Running Validation...
  Accuracy: 0.98
  Validation took: 0:00:01

======== Epoch 5 / 6 ========
Training...

  Average training loss: 0.37
  Training epcoh took: 0:01:29
Running Validation...
  Accuracy: 0.98
  Validation took: 0:00:01

======== Epoch 6 / 6 ========
Training...
```

```
  Average training loss: 0.36
  Training epcoh took: 0:01:28
Running Validation...
  Accuracy: 0.98
  Validation took: 0:00:01

Training complete!

Predicting labels for 160 test sentences...
Number of expressions with negative result 47
 47  predicted correctly , accuracy  1.0

Number of expressions with 0 result 2
 0  predicted correctly , accuracy  0.0

Number of expressions with positive result 111
 110  predicted correctly , accuracy  0.990990990990991
```