CSC413 Programming Assignment 3

TONGFEI ZHOU, Student #: 1004738448

March 15th, 2022

Part 1: Neural machine translation (NMT)

1.

(a) The code is as follows:

```
[6]  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)
             self.Wir = nn.Linear(input_size, hidden_size)
             self.Wih = nn.Linear(input_size, hidden_size)

             # Hidden linear layers
             self.Whz = nn.Linear(hidden_size, hidden_size)
             self.Whr = nn.Linear(hidden_size, hidden_size)
             self.Whh = nn.Linear(hidden_size, hidden_size)
```

```
        def forward(self, x, h_prev):
            """Forward pass of the GRU computation for one time step.

            Arguments
                x: batch_size x input_size
                h_prev: batch_size x hidden_size

            Returns:
                h_new: batch_size x hidden_size
            """

            # ------------
            # FILL THIS IN
            # ------------
            z = torch.sigmoid(self.Wiz(x) + self.Whz(h_prev))
            r = torch.sigmoid(self.Wir(x) + self.Whr(h_prev))
            g = torch.tanh(self.Wih(x) + self.Whh(r * h_prev))
            h_new = (1.0-z) * g + z * h_prev
            return h_new
```

(b) Neither model performs significantly better than the other one. The reason is that we are using *teacher-forcing*, and this would lead to the situation that, in the training part, the decoder can train with the target, but in the testing part, the decoder can only generate the text from the previous word. This leads to the situation that the decoder is working under different distribution in training and testing time, and so increasing the dataset size does not reduce the impacts created by the teacher-forcing, and so the model performs with barely no difference.

2. After translating several sentences, such as "*the air conditioning is working*", "*the air streaming is working*", "*the fair conding is working*", "*the pair performing is working*", I noticed that one distinct failure mode would be that the model performs very badly when translating the word ends with "*ing*".

3. (a) Total number of parameters of the LSTM encoder: $4(DH + H^2)$
   (b) Total number of parameters of the GRU encoder: $3(DH + H^2)$

Additive Attention

1. Answer to Question 3: The training speed is slower than the RNNs without attention because we are introducing the MLP, and so more parameters to train with.

Scaled Dot Product Attention

1. Screenshots of my `ScaledDotProduct`

```
# ------------
# FILL THIS IN
# ------------
batch_size = keys.size(0)
if queries.dim() == 2:
  queries = queries.unsqueeze(queries, 1)
q = self.Q(queries)
k = self.K(keys)
v = self.V(values)
unnormalized_attention = torch.bmm(k, q.transpose(2,1)) * self.scaling_factor
attention_weights = self.softmax(unnormalized_attention)
context = torch.bmm(attention_weights.transpose(2,1), v)
return context, attention_weights
```

2. Screenshots of my `CausalScaledDotProduct`

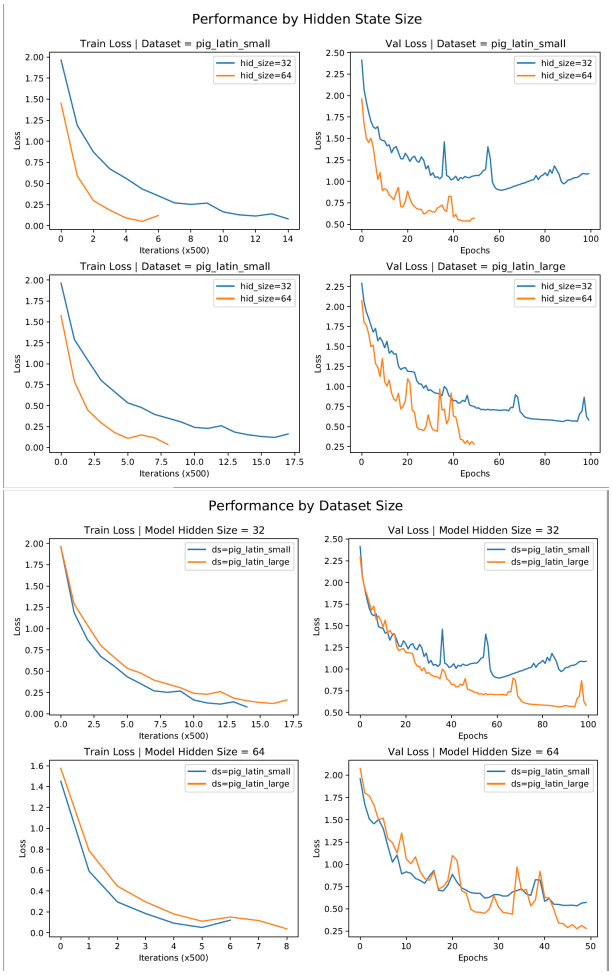
```
# ------------
# FILL THIS IN
# ------------
batch_size = keys.size(0)
if queries.dim() == 2:
  queries = queries.unsqueeze(queries, 1)
q = self.Q(queries)
k = self.K(keys)
v = self.V(values)
unnormalized_attention = torch.bmm(k, q.transpose(2,1)) * self.scaling_factor
mask = torch.triu(unnormalized_attention)
mask[mask == 0] = self.neg_inf
attention_weights = self.softmax(mask)
context = torch.bmm(attention_weights.transpose(2,1), v)
return context, attention_weights
```

3. The performance of the model using `ScaledDotAttention` is not better than the `RNNAttention` model. The reason might be that the cosine similarity is not appropriate for the *character-level* NMT model, and instead, the additive attention can reflect the similarity better in the *character-level* perspective.

4. The reason to represent the position of each word is that the order of the sequence conveys important information for the machine translation tasks. For instance, *"I love cat"* and *"Cat loves me"* convey completely different meaning. The advantages of using this positional encoding method are that, first, we can adjust the encoding by the hyperparameter d_{model}, and second, the range of sine and cosine functions is $[-1, 1]$, and so we have larger capacity comparing to the one-hot encoding.

5. In all three models, `RNNAttention` performs best with the lowest validation loss being approximately 0.18496574, while `Transformer` being the second best, with validation loss approximately 0.8973. The single-block `Attention` is the worst, with validation loss only around 1.153. However, the `Transformer` model only makes very slightly more mistakes than `RNNAttention`, and the `RNNAttention` model has a very tiny training error with tiny increasing validation error at the end of the training, which might cause the over-fitting issue.

6. The corresponding plots are as follows:



The lowest validation loss obtained by are:

(a) `pig_latin_small`, `hidden_size` = 32: 0.8972819642651648

(b) `pig_latin_large`, `hidden_size` = 32: 0.5634385246251311

(c) `pig_latin_small`, `hidden_size` = 64: 0.534538182285836

(d) `pig_latin_large`, `hidden_size` = 64: 0.27657217867149764

As the hidden size increases, the number of gradient descent iterations required is halved when holding the dataset size same, and the validation loss is halved as well, which means that increasing the hidden size improves the generalization of the model a lot. As the dataset size increases, one can notice that neither the gradient descent iterations nor the validation loss is improved significantly. These results are in my expectations since increasing the hidden size is increasing the complexity of the model by introducing more parameters to train, and so the training and validation loss should be increased. However, since we are training the model based on the *teacher-forcing*, the dataset size does not have much effect on improving the model's performance.

Part 3: Fine-tuning Pretrained Language Models (LMs)

1. Screenshots of the code for `BertForSentenceClassification` are following:

```python
class BertForSentenceClassification(BertModel):
    def __init__(self, config):
        super().__init__(config)

        ##### START YOUR CODE HERE #####
        # Add a linear classifier that map BERTs [CLS] token representation to the unnormalized
        # output probabilities for each class (logits).
        # Notes:
        #  * See the documentation for torch.nn.Linear
        #  * You do not need to add a softmax, as this is included in the loss function
        #  * The size of BERTs token representation can be accessed at config.hidden_size
        #  * The number of output classes can be accessed at config.num_labels
        self.classifier = nn.Linear(config.hidden_size, config.num_labels)
        ##### END YOUR CODE HERE #####
        self.loss = torch.nn.CrossEntropyLoss()
```

```python
    def forward(self, labels=None, **kwargs):
        outputs = super().forward(**kwargs)
        ##### START YOUR CODE HERE #####
        # Pass BERTs [CLS] token representation to this new classifier to produce the logits.
        # Notes:
        #  * The [CLS] token representation can be accessed at outputs.pooler_output
        cls_token_repr = outputs.pooler_output
        logits = self.classifier(cls_token_repr)
        ##### END YOUR CODE HERE #####
        if labels is not None:
            outputs = (logits, self.loss(logits, labels))
        else:
            outputs = (logits,)
        return outputs
```

2. (a) Compared to fine-tuning BERT, when BERTs' weights are frozen, the training time decreases a lot since we do not need to train the parameters in the BERTs' weights any longer, but only the classifier we add for our downstream task.

   (b) Compared to fine-tuning BERT, when BERTs' weights are frozen, the validation accuracy has decreased quite a lot to only around 0.3. The reason is that we only add a classifier to the output of the BERT, and so the weights of the BERT model are not adjusted appropriately for our downstream task.

3. Compared to fine-tuning BERT with the pretrained weights from MathBERT, we can see that the validation accuracy for BERTweet after fine-tuning the weights is much higher than the "frozen weight" model, but is lower than the MathBERT. The difference in the validation accuracy is because that MathBERT is trained on a large mathematical corpus ranging from pre-kindergarten to college graduate level mathematical content, and BERTweet is trained on hundreds of millions of tweets. Therefore, MathBERT is more appropriate for our downstream task.

Part 4: Connecting Text and Images with CLIP

1. The process of finding the caption is relevant easy, as one includes the dominant features in the picture. For instance, here we can find the picture easily by inputting the caption containing the word "butterfly" and "purple", as I inputted "butterfly on the purple"