CSC413 – Neural Networks and Deep Learning
Ziyang Lin | 1003246551
Assignment 3

Gated Recurrent Unit

1. *MyGRUC*ell implementation as below:

```
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.Win = 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.Whn = nn.Linear(hidden_size, hidden_size)
```

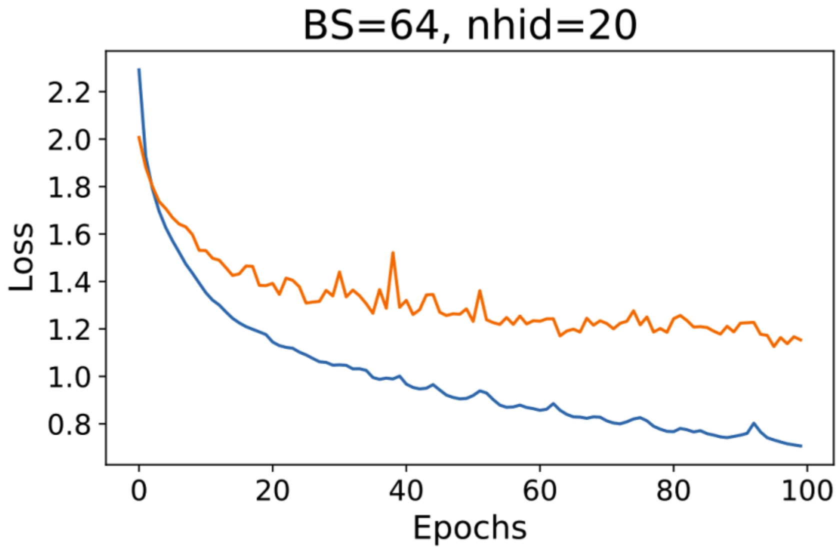
```
def forward(self, x, h_prev):
    """Forward pass of the GRU computation for one time step.

    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 = F.sigmoid(self.Wiz(x) + self.Whz(h_prev))
    r = F.sigmoid(self.Wir(x) + self.Whr(h_prev))
    g = F.tanh(self.Win(x) + r * self.Whn(h_prev))
    h_new = (1 - z) * g + z * h_prev
    return h_new
```

2. Plot for training (blue) and validation loss (orange) as below:



3. One input and output sentence is shown in below. We observe that even though shorter words are mostly fine, but longer words are not translated correctly as the letters after the first one are not copied correctly to the translation (e.g. *"conditioning → "onditingventway"*, *"functioning → "untorrionsway"*). Also, words that starts with consonant pairs are not translated correctly as the second consonant is missing (e.g. *"shake" → "akesway"*, *"thack" → "acktray"*)

```
source:          but the heat pump is not functioning
translated:      utway ehay eathay umppay isway otay untorrionsway
source:          shake thack ay way sss iii hyperparameters
translated:      akesway acktray ayway ayway fsay iriway uscalicationcay
```

Additive Attention

1. Equations for $\tilde{\alpha}_i^{(t)}, \alpha_i^{(t)}, c_t$ are as below:

$$\tilde{\alpha}_i^{(t)} = W_2(\max(W_1[Q_t, K_i], 0) + b_1) + b_2$$

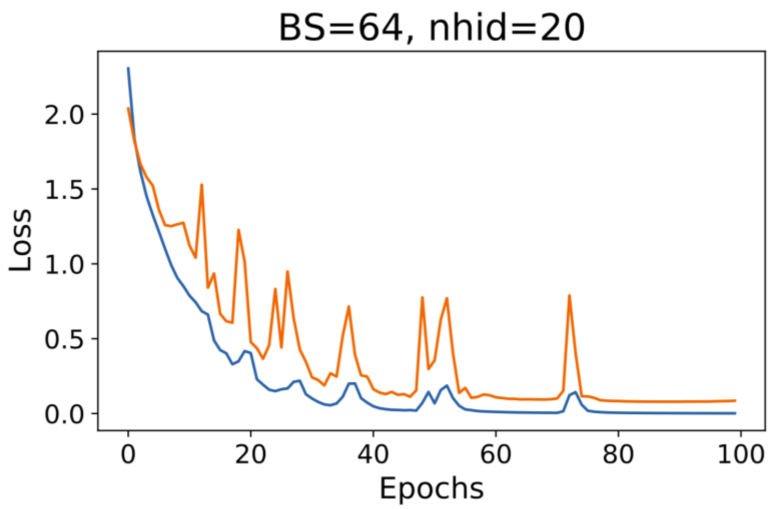
$$\alpha_i^{(t)} = \text{softmax}\left(\tilde{\alpha}_i^{(t)}\right)$$

$$c_t = \sum_{i=1}^{T} \alpha_i^{(t)} V_i$$

2. *RNNAttentionDecoder self.forward* method implementations as below:

```
# ------------
# FILL THIS IN - START
# ------------
embed_current = embed[:,i,:]  # Get the current time step, across the whole batch
context, attention_weights = self.attention(h_prev, annotations, annotations)  # b
embed_and_context = torch.cat((embed_current, torch.squeeze(context, 1)), dim=1)
h_prev = self.rnn(embed_and_context, h_prev)  # batch_size x hidden_size
# ------------
# FILL THIS IN - END
# ------------
```

3. Plot for training (blue) and validation loss (orange) as below:



Scaled Dot Product Attention

1. Implementation of the *ScaledDotAttention forward* method:

```
# ------------
# FILL THIS IN
# ------------
batch_size = keys.size(0)
seq_len = keys.size(1)
hidden_size = keys.size(2)
if queries.dim() < 3:
    queries = queries.unsqueeze(1)

q = self.Q(queries) # batch * k * hidden
k = self.K(keys) # batch * seq_len * hidden
v = self.V(values) # batch * seq_len * hidden
unnormalized_attention = (k @ q.transpose(1, 2)) * self.scaling_factor
attention_weights = self.softmax(unnormalized_attention)
context = attention_weights.transpose(1, 2) @ v
return context, attention_weights
```

2. Implementation of the *CasualScaledDotAttention forward* method:

```
# ------------
# FILL THIS IN
# ------------
batch_size = keys.size(0)
seq_len = keys.size(1)
hidden_size = keys.size(2)
if queries.dim() < 3:
    queries = queries.unsqueeze(1)
q = self.Q(queries) # batch * k * hidden
k = self.K(keys) # batch * seq_len * hidden
v = self.V(values) # batch * seq_len * hidden
unnormalized_attention = (k @ q.transpose(1, 2)) * self.scaling_factor
mask = torch.tril(torch.ones((batch_size, seq_len, seq_len),
                             dtype=torch.uint8)).transpose(1, 2)
unnormalized_attention[mask == 0] = self.neg_inf
attention_weights = self.softmax(unnormalized_attention)
context = attention_weights.transpose(1, 2) @ v
return context, attention_weights
```

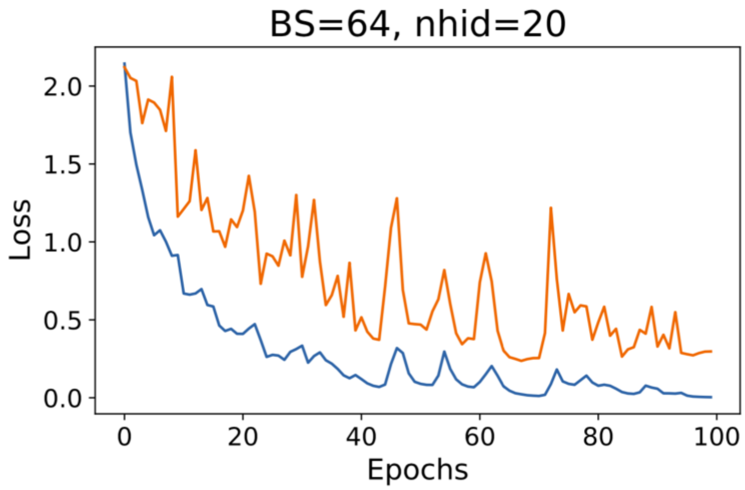
3. Implementation of the *TransformerEncoder forward* method:

```
# ------------
# FILL THIS IN - START
# ------------
encoded = self.embedding(inputs) + self.positional_encodings[:seq_len]
annotations = encoded
for i in range(self.num_layers):
    new_annotations, self_attention_weights = self.self_attentions[i](annotations,
                                                                      annotations,
                                                                      annotations)
    residual_annotations = annotations + new_annotations
    new_annotations = self.attention_mlps[i](residual_annotations)
    annotations = residual_annotations + new_annotations
# ------------
# FILL THIS IN - END
# ------------
```

4. Implementation of the *TransformerDecoder forward* method:

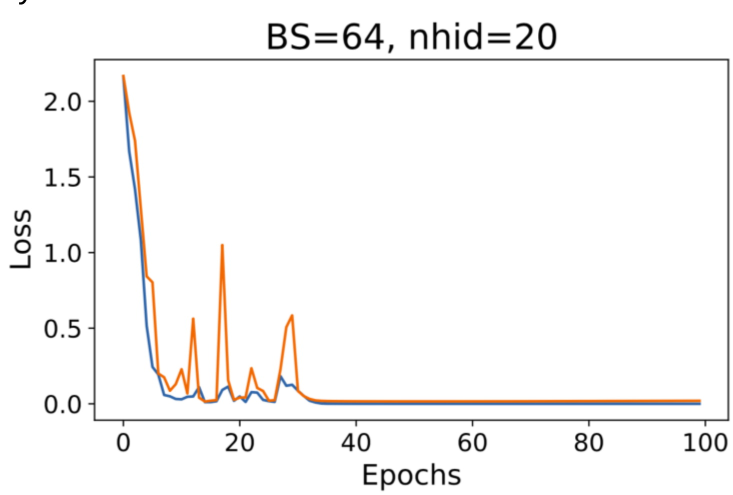
```
for i in range(self.num_layers):
  # ------------
  # FILL THIS IN - START
  # ------------
    new_contexts, self_attention_weights = self.self_attentions[i](contexts,
                                                                   contexts,
                                                                   contexts)
    residual_contexts = contexts + new_contexts
    new_contexts, encoder_attention_weights = self.encoder_attentions[i](residual_contexts,
                                                                         annotations,
                                                                         annotations)
    residual_contexts = residual_contexts + new_contexts
    new_contexts = self.attention_mlps[i](residual_contexts)
    contexts = residual_contexts + new_contexts
  # ------------
  # FILL THIS IN - END
  # ------------
```

5. Plot for training (blue) and validation loss (orange) in below, and we also used the same inputs for translation as before. We notice that the model performs better than the GRU model but worse than the additive one in terms of validation error. However, it is able to handle words that start with consonant pairs and it also performs better in longer words (e.g. *"hyperparameters"* → *"yperparametersay"* vs. *"yzarparametersshay"* in *AdditiveAttention* model)




```
source:          but the heat pump is not functioning
translated:      utbay ethay eathay umppay isway otnay unctioningfay
source:          shake thack ay way sss iii hyperparameters
translated:      akeshay ackthay ayway ayway ssssay iiiway yperparametersay
```

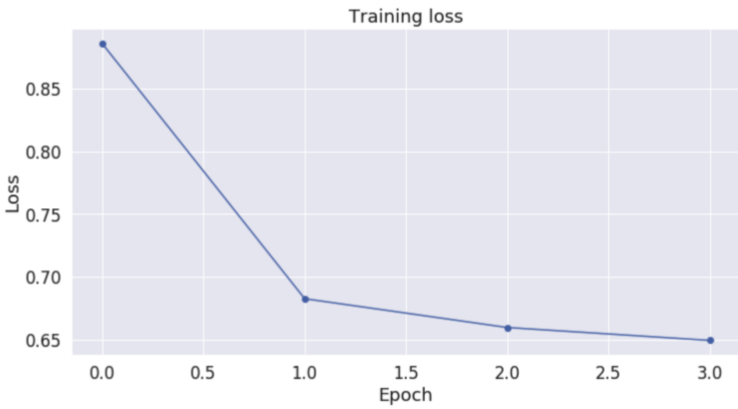
6. We see that after modifying the transformer decoder, the training loss goes to 0 and the validation error is also very small, but the translations are completely nonsensical. This is because of the teacher forcing during training as the correct labels are also fed to the model as inputs. But when testing, the input will be the prediction in the previous time step, so if one prediction is incorrect, all followings will be very likely to be incorrect as well.



```
source:          but the heat pump is not functioning
translated:        aaaaaaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaaaaaa   aaaaaaaaaaaaaaaaaaaa
source:          shake thack ay way sss iii hyperparameters
translated:      aaaaaaaaaaaaaaaaaaaa      aaaaaaaaaaaaaaaaaaaa
```

BERT for Arithmetic Sentiment Analysis

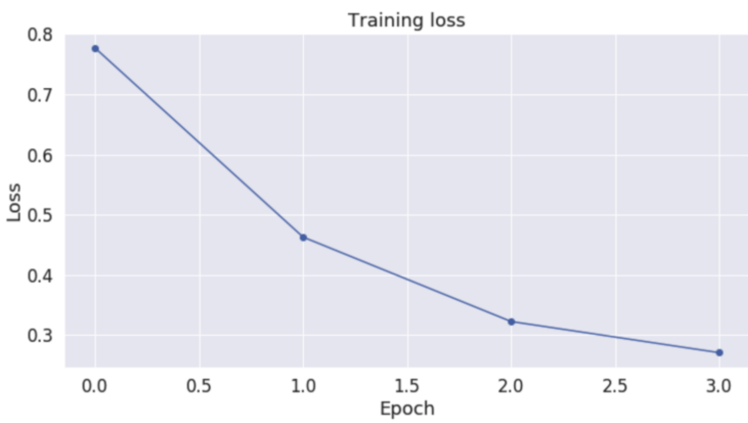
1. The sentence classifier I used has a $tanh$ activation instead of the $ReLU$.
2. Training error curves for freeze (upper) and fine-tuned (lower) model as follow:




```
Predicting labels for 160 test sentences...
Number of expressions with negative result 47
 0  predicted correctly , accuracy  0.0

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
```




```
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
 109  predicted correctly , accuracy  0.981981981
```

   We observe that even though the training loss gradually decrease, but the validation accuracy of the freeze model does not change during training, and that the fine-tuned model leads to smaller training error and much higher validation accuracy of 0.98. The fine-tuned model also generalizes better as it performs better in test set. The model fails to predict the zero cases may be because that there is not enough data to train from.

3. We found that the fine-tuned model is quite powerful as it can extrapolate to unspecific quantities (thousand / hundred), Arabian numbers, numbers with minus sign, and even longer sentences and multiplication and division calculation. However, the model always fails to predict the case of zero. Some of the inference results of some examples are shown below.

Extrapolating to Unspecific Quantities (i.e. thousand / hundred)

```
what_is("eight plus thousand")        positive
what_is("eight minus hundred")        negative
```

Arabian Numbers Instead of Words

```
what_is("1 minus 14")        negative
what_is("1 minus two")       negative
what_is("-1 minus two")      negative
what_is("-1 plus 3")         positive
```

Longer Sentences with Multiple Operations

```
what_is("three minus two minus eight")          negative
what_is("three minus two plus eight")           positive
what_is("one minus one minus one")              negative
what_is("one minus one minus one plus ten")     positive
```

Add the Word "Minus" to Express Negative Numbers

```
what_is("minus one minus two")      negative
what_is("minus 3 plus four")        positive
```

Multiplication and Division

```
what_is("eight divided by 5")          positive
what_is("minus eight divided by 5")    negative
what_is("3 times zero")                positive
what_is("3 times minus six")           negative
```

4. We trained the fine-tuned model with different settings for each hyperparameter: smaller batch size (16), more epochs (8), larger learning rate (2e-4) while holding others the same as our original models for each.
    a. With batch size 16, the training loss lowers to 0.18, but the validation accuracy fluctuates around 0.95 among epochs, didn't exhibit improvement.
    b. With number of epochs 8, achieved same validation accuracy and 0.15 training loss, didn't exhibit improvement.
    c. With learning rate 2e-4, training takes nearly three times longer but only achieved a validation accuracy of 0.73, performing worse.