Exercise Solution

Exercise 11 - NLP

Iterable Dataset – parse_file()

```
##########################################################################
# TODO:                                                                  #
#   Task 1:                                                              #
#       - Loop through all chunks in reader                              #
#       - Loop through all rows in chunk                                 #
#       - 'return' a dictionary: {'source': source_data,                 #
#                                 'target': target_data}                 #
# Hints:                                                                 #
#       - Use iterrows() to iterate through all rows! Have a look at     #
#         at pandas implementation of this function and see what it     #
#         returns!                                                       #
#       - The dataframe we are reading in this case has two columns:     #
#         'source' and 'target'. You can index them using something      #
#         like row['source'].                                            #
#         Dont use return ;)                                             #
##########################################################################

for chunk in reader:
    for _, row in chunk.iterrows():
        yield {'source': row['source'], 'target': row['target']}

##########################################################################
#                            END OF YOUR CODE                            #
##########################################################################
```

Scaled Dot Attention – __init__()

```
########################################################################
# TODO:                                                                #
#   Task 2: Initialize the softmax layer (torch.nn implementation)     #
#   Task 13: Initialize the dropout layer (torch.nn implementation)    #
########################################################################

self.softmax = nn.Softmax(dim=-1)
self.dropout = nn.Dropout(p=dropout)

########################################################################
#                           END OF YOUR CODE                           #
########################################################################
```

Scaled Dot Attention – forward()

```
# Hint 2:                                                              #
#       - torch.transpose(x, dim_1, dim_2) swaps the dimensions dim_1  #
#         and dim_2 of the tensor x!                                   #
#       - Later we will insert more dimensions into *, so how could    #
#         index these dimensions to always get the right ones?         #
#       - Also dont forget to scale the scores as discussed!           #
# Hint 8:                                                              #
#       - Have a look at Tensor.masked_fill_() or use torch.where()    #
########################################################################

scores = torch.matmul(q, k.transpose(-2, -1)) / (self.d_k ** 0.5)

if mask is not None:
    scores.masked_fill_(~mask, -torch.inf)

scores = self.softmax(scores)

scores = self.dropout(scores)

outputs = torch.matmul(scores, v)

########################################################################
#                         END OF YOUR CODE                             #
########################################################################
```

Multi Head Attention - __inti__()

```
##########################################################################
# TODO:                                                                  #
#   Task 3:                                                              #
#       -Initialize all weight layers as linear layers                   #
#       -Initialize the ScaledDotAttention                               #
#       -Initialize the projection layer as a linear layer               #
#  Task 13:                                                              #
#       -Initialize the dropout layer (torch.nn implementation)          #
#                                                                        #
# Hints 3:                                                               #
#       - Instead of initializing several weight layers for each head,  #
#         you can create one large weight matrix. This speed up          #
#         the forward pass, since we dont have to loop through all       #
#         heads!                                                         #
#       - All linear layers should only be a weight without a bias!     #
##########################################################################

self.weights_q = nn.Linear(in_features=d_model, out_features=n_heads * d_k, bias=False)
self.weights_k = nn.Linear(in_features=d_model, out_features=n_heads * d_k, bias=False)
self.weights_v = nn.Linear(in_features=d_model, out_features=n_heads * d_v, bias=False)

self.attention = ScaledDotAttention(d_k=d_k, dropout=dropout)

self.project = nn.Linear(in_features=n_heads * d_v, out_features=d_model, bias=False)
self.dropout = nn.Dropout(p=dropout)

##########################################################################
#                           END OF YOUR CODE                             #
##########################################################################
```

Multi Head Attention - forward()

```
# Hints 8:                                                              #
#       - Use unsqueeze() to add dimensions at the correct location    #
#########################################################################

q = self.weights_q(q)
k = self.weights_k(k)
v = self.weights_v(v)

q = q.reshape(batch_size, sequence_length_queries, self.n_heads, self.d_k)
q = q.transpose(-3, -2)

k = k.reshape(batch_size, sequence_length_keys, self.n_heads, self.d_k)
k = k.transpose(-3, -2)

v = v.reshape(batch_size, sequence_length_keys, self.n_heads, self.d_v)
v = v.transpose(-3, -2)

if mask is not None:
    mask = mask.unsqueeze(1)
outputs = self.attention(q, k, v, mask)


outputs = outputs.transpose(-3, -2)
outputs = outputs.reshape(batch_size, sequence_length_queries, self.n_heads * self.d_v)

outputs = self.project(outputs)
outputs = self.dropout(outputs)

#########################################################################
#                            END OF YOUR CODE                           #
#########################################################################
```

Feed Forward Neural Network - __init__()

```
##########################################################################
# TODO:                                                                  #
#   Task 5: Initialize the feed forward network                          #
#   Task 13: Initialize the dropout layer (torch.nn implementation)      #
#                                                                        #
##########################################################################

self.linear_1 = nn.Linear(in_features=d_model, out_features=d_ff)
self.relu = nn.ReLU()
self.linear_2 = nn.Linear(in_features=d_ff, out_features=d_model)
self.dropout = nn.Dropout(p=dropout)

##########################################################################
#                           END OF YOUR CODE                             #
##########################################################################
```

Feed Forward Neural Network – forward()

```
##########################################################################
# TODO:                                                                  #
#   Task 5: Implement forward pass of feed forward layer                 #
#   Task 13: Pass the output through a dropout layer as a final step    #
#                                                                        #
##########################################################################

outputs = self.linear_1(inputs)
outputs = self.relu(outputs)
outputs = self.linear_2(outputs)
outputs = self.dropout(outputs)

##########################################################################
#                             END OF YOUR CODE                           #
##########################################################################
```

Encoder Block - __init__()

```
########################################################################
# TODO:                                                                #
#   Task 6: Initialize an Encoder Block                                #
#           You will need:                                             #
#                            - Multi-Head Self-Attention layer         #
#                            - Layer Normalization                     #
#                            - Feed forward neural network layer       #
#                            - Layer Normalization                     #
#                                                                      #
# Hint 6: Check out the pytorch layer norm module                      #
########################################################################

self.multi_head = MultiHeadAttention(d_model=d_model, d_k=d_k, d_v=d_v, n_heads=n_heads, dropout=dropout)
self.layer_norm1 = nn.LayerNorm(normalized_shape=d_model)
self.ffn = FeedForwardNeuralNetwork(d_model=d_model, d_ff=d_ff, dropout=dropout)
self.layer_norm2 = nn.LayerNorm(normalized_shape=d_model)

########################################################################
#                            END OF YOUR CODE                          #
########################################################################
```

Encoder Block – forward()

```
########################################################################
# TODO:                                                                #
#   Task 6: Implement the forward pass of the encoder block            #
#   Task 12: Pass on the padding mask                                  #
#                                                                      #
# Hint 6: Don't forget the residual connection! You can forget about   #
#         the pad_mask for now!                                        #
########################################################################

outputs = self.multi_head(q=inputs, k=inputs, v=inputs, mask=pad_mask) + inputs
outputs = self.layer_norm1(outputs)
outputs = self.ffn(outputs) + outputs
outputs = self.layer_norm2(outputs)

########################################################################
#                           END OF YOUR CODE                           #
########################################################################
```

Decoder Block - __init__()

```
##########################################################################
# TODO:                                                                  #
#   Task 9: Initialize an Decoder Block                                  #
#            You will need:                                              #
#                            - Causal Multi-Head Self-Attention layer    #
#                            - Layer Normalization                       #
#                            - Multi-Head Cross-Attention layer          #
#                            - Layer Normalization                       #
#                            - Feed forward neural network layer         #
#                            - Layer Normalization                       #
#                                                                        #
# Hint 9: Check out the pytorch layer norm module                        #
##########################################################################

self.causal_multi_head = MultiHeadAttention(d_model=d_model, d_k=d_k, d_v=d_v, n_heads=n_heads, dropout=dropout)
self.layer_norm1 = nn.LayerNorm(normalized_shape=d_model)
self.cross_multi_head = MultiHeadAttention(d_model=d_model, d_k=d_k, d_v=d_v, n_heads=n_heads, dropout=dropout)
self.layer_norm2 = nn.LayerNorm(normalized_shape=d_model)
self.ffn = FeedForwardNeuralNetwork(d_model=d_model, d_ff=d_ff, dropout=dropout)
self.layer_norm3 = nn.LayerNorm(normalized_shape=d_model)

##########################################################################
#                            END OF YOUR CODE                            #
##########################################################################
```

Decoder Block – forward()

```
#########################################################################
# TODO:                                                                 #
#   Task 9: Implement the forward pass of the decoder block            #
#   Task 12: Pass on the padding mask                                   #
#                                                                       #
# Hint 9:                                                               #
#       - Don't forget the residual connections!                        #
#       - Remember where we need the causal mask, forget about the      #
#         other mask for now!                                           #
# Hints 12:                                                             #
#       - We have already combined the causal_mask with the pad_mask    #
#         for you, all you have to do is pass it on to the "other"      #
#         module                                                        #
#########################################################################

outputs = self.causal_multi_head(q=inputs, k=inputs, v=inputs, mask=causal_mask) + inputs
outputs = self.layer_norm1(outputs)
outputs = self.cross_multi_head(q=outputs, k=context, v=context, mask=pad_mask) + outputs
outputs = self.layer_norm2(outputs)
outputs = self.ffn(outputs) + outputs
outputs = self.layer_norm3(outputs)

#########################################################################
#                           END OF YOUR CODE                            #
#########################################################################
```

Transformer - __init__()

```
# Hint 11: Have a look at the output shape of the decoder and the      #
#          output shape of the transformer model to figure out the     #
#          dimensions of the output layer! We will not need a bias!    #
########################################################################

self.embedding = Embedding(vocab_size=self.vocab_size,
                           d_model=self.d_model,
                           max_length=self.max_length,
                           dropout=self.dropout)

self.encoder = Encoder(d_model=self.d_model,
                       d_k=self.d_k,
                       d_v=self.d_v,
                       n_heads=self.n_heads,
                       d_ff=self.d_ff,
                       n=self.n,
                       dropout=self.dropout)

self.decoder = Decoder(d_model=self.d_model,
                       d_k=self.d_k,
                       d_v=self.d_v,
                       n_heads=self.n_heads,
                       d_ff=self.d_ff,
                       n=self.n,
                       dropout=self.dropout)

self.output_layer = nn.Linear(in_features=self.d_model,
                              out_features=self.vocab_size,
                              bias=False)

########################################################################
#                            END OF YOUR CODE                          #
########################################################################
```

Transformer – forward()

```
##########################################################################
# TODO:                                                                  #
#   Task 11: Implement the forward pass of the transformer!              #
#            You will need to:                                           #
#               - Compute the encoder embeddings                         #
#               - Compute the forward pass through the encoder           #
#               - Compute the decoder embeddings                         #
#               - Compute the forward pass through the decoder           #
#               - Compute the output logits                              #
#   Task 12: Pass on the encoder and decoder padding masks!              #
#                                                                        #
# Hints 12: Have a look at the forward pass of the encoder and decoder   #
#           to figure out which masks to pass on!                        #
##########################################################################

encoder_inputs = self.embedding(encoder_inputs)
encoder_outputs = self.encoder(encoder_inputs,
                               encoder_mask=encoder_mask)

decoder_inputs = self.embedding(decoder_inputs)
decoder_outputs = self.decoder(decoder_inputs,
                               encoder_outputs,
                               decoder_mask=decoder_mask,
                               encoder_mask=encoder_mask)

outputs = self.output_layer(decoder_outputs)

##########################################################################
#                           END OF YOUR CODE                             #
##########################################################################
```