Exercise 12 — Solution Reference

I2DL — Introduction to Deep Learning

exercise_code/data/dataset.py

```
        ########################################################################
        # 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                           #
        ########################################################################
```

exercise_code/network/attention.py

```
        ########################################################################
        # TODO:                                                                #
        #   Task 11: Initialize the dropout layer (torch.nn implementation)    #
        #                                                                      #
        ########################################################################

        self.dropout = nn.Dropout(p=dropout)

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

```
##########################################################################
# TODO:                                                                  #
#   Task 6:                                                              #
#       - Add a negative infinity mask if a mask is given                #
#                                                                        #
# Hint 6:                                                                #
#       - Have a look at Tensor.masked_fill_() or use torch.where()      #
##########################################################################

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

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

```
##########################################################################
# TODO:                                                                  #
#   Task 11:                                                             #
#       - Add dropout to the scores                                      #
##########################################################################

scores = self.dropout(scores)

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

exercise_code/network/decoder_block.py

```
##########################################################################
# TODO:                                                                  #
#   Task 7: Initialize the 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 7: 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                                #
##############################################################################
```

```
##############################################################################
# TODO:                                                                      #
#   Task 7: Implement the forward pass of the decoder block                  #
#   Task 10: Pass on the padding mask                                        #
#                                                                            #
# Hint 7:                                                                    #
#       - Don't forget the residual connections!                             #
#       - Remember where we need the causal mask, forget about the           #
#         other mask for now!                                                #
# Hints 10:                                                                  #
#       - 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                                #
##############################################################################
```

exercise_code/network/embedding.py

```
##############################################################################
# TODO:                                                                      #
#   Task 11: Initialize the dropout layer (torch.nn implementation)          #
#                                                                            #
##############################################################################

self.dropout = nn.Dropout(p=dropout)

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

```
##############################################################################
# TODO:                                                                      #
#   Task 11:                                                                 #
#       - Add dropout to the outputs                                         #
##############################################################################
```

```
outputs = self.dropout(outputs)

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

exercise_code/network/encoder_block.py

```
##############################################################################
# TODO:                                                                      #
#   Task 4: Initialize the Encoder Block                                     #
#           You will need:                                                   #
#                            - Multi-Head Self-Attention layer               #
#                            - Layer Normalization                           #
#                            - Feed forward neural network layer             #
#                            - Layer Normalization                           #
#                                                                            #
# Hint 4: 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                                #
##############################################################################
```

```
##############################################################################
# TODO:                                                                      #
#   Task 4: Implement the forward pass of the encoder block                  #
#   Task 10: Pass on the padding mask                                        #
#                                                                            #
# Hint 4: 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                                #
##############################################################################
```

exercise_code/network/feed_forward_nn.py

```
########################################################################
# TODO:                                                                #
#   Task 3: Initialize the feed forward network                        #
#   Task 11: 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                          #
########################################################################
```

```
########################################################################
# TODO:                                                                #
#   Task 3: Implement forward pass of feed forward layer               #
#   Task 11: 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                          #
########################################################################
```

exercise_code/network/multi_head_attention.py

```
########################################################################
# TODO:                                                                #
#  Task 11:                                                            #
#       -Initialize the dropout layer (torch.nn implementation)        #
#                                                                      #
########################################################################

self.dropout = nn.Dropout(p=dropout)

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

```
################################################################################
# TODO:                                                                        #
#   Task 6:                                                                    #
#       - If a mask is given, add an empty dimension at dim=1                  #
#       - Pass the mask to the ScaledDotAttention layer                        #
#                                                                              #
# Hints 6:                                                                     #
#       - Use unsqueeze() to add dimensions at the correct location            #
################################################################################

if mask is not None:
    mask = mask.unsqueeze(1)

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

```
################################################################################
# TODO:                                                                        #
#  Task 11:                                                                    #
#       - Add dropout as a final step after the projection layer               #
#                                                                              #
################################################################################

outputs = self.dropout(outputs)

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

exercise_code/network/transformer.py

```
################################################################################
# TODO:                                                                        #
#   Task 9: Initialize the transformer!                                        #
#            You will need:                                                    #
#               - An embedding layer                                           #
#               - An encoder                                                   #
#               - A decoder                                                    #
#               - An output layer                                              #
#                                                                              #
# Hint 9: 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                              #
##############################################################################
```

```
##############################################################################
# TODO:                                                                      #
#   Task 9: 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 10: Pass on the encoder and decoder padding masks!                  #
#                                                                            #
# Hints 10: 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                              #
##############################################################################
```

exercise_code/trainer.py

```
########################################################################
# TODO:                                                                #
#   Task 13:                                                           #
#       - Unpack the batch                                             #
#       - Move all tensors to self.device                              #
#       - Compute the outputs of self.model                            #
#       - Compute the loss using self.loss_func                        #
#                                                                      #
# Hints: Inspect the outputs of collate method  - __call__()  - in     #
#        data/collator.py                                              #
#        Inspect the inputs of the SmoothCrossEntropy                  #
#        Make sure to pass all masks to the model!                     #
########################################################################

encoder_inputs = batch['encoder_inputs'].to(self.device)
decoder_inputs = batch['decoder_inputs'].to(self.device)
encoder_mask = batch['encoder_mask'].to(self.device)
decoder_mask = batch['decoder_mask'].to(self.device)

outputs = self.model(encoder_inputs, decoder_inputs, encoder_mask, decoder_mask)

labels = batch['labels'].to(self.device)
label_mask = batch['label_mask'].to(self.device)
lengths = batch['label_length'].to(self.device)

loss = self.loss_func(outputs, labels, label_mask, lengths)

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

2_transformer.ipynb

```
########################################################################
# TODO:                                                                #
#   Initialize your tokenizer. You train your own tokenizer and load   #
#   load it like we did in the first notebook, or load the pretrained  #
#   version!                                                           #
#                                                                      #
# Hint: Scroll up a couple of cells for the default tokenizer          #
########################################################################

tokenizer = load_pretrained_fast()

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

```
################################################################################
# TODO:                                                                        #
#   Implement you model here                                                   #
#                                                                              #
################################################################################

hparams = {
    'd_model': 256,
    'd_k': 32,
    'd_v': 32,
    'd_ff': 512,
    'n_heads': 3,
    'n': 2,
    'dropout': 0
}

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

```
################################################################################
# TODO:                                                                        #
#   Define the optimizer and optionally scheduler - not really needed  #
#                                                                              #
################################################################################

optimizer = Adam(model.parameters(), lr=1e-3)

epochs = 200
batch_size = 20

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