Exercise 1: Using BERT

Trying out a BERT model from the Transformers library

If we have first computed the output from a BERT model like this:

```
bert_output = example_bert_model(input_ids=test_input.input_ids,
                                 attention_mask=test_input.attention_mask,
                                 output_attentions=True,
                                 output_hidden_states=True)
```

then

```
bert_output.last_hidden_state
```

contains the BERT representations from the top layer or all tokens in the batch. If we use the example given in the notebook, then the shape is (1, 7, 768), where 1 is the number of documents in the batch, 7 the length of the document (including dummy tokens) and 768 the size of the BERT representation.

```
bert_output.hidden_states
```

is a tuple including 13 elements. This stores the output from all layers in the BERT model: first the embedding layer, and then 12 transformer blocks.

```
bert_output.attentions
```

is a tuple containing 12 elements. Each element is a tensor storing the attention scores for all heads in one layer. In our case, each such tensor has the shape (1, 12, 7, 7). Here, 1 is again for the number of documents in the batch, 12 for the number of attention heads. Then there are 7x7 attention scores: for each of the 7 tokens, we compute attention scores over all tokens.

Masked language modeling

```
masked_sentence = tokenizer('The Germans like to drink [MASK] .',
return_tensors='pt')

# Find the position of the masked token
mask_position =
list(masked_sentence.input_ids[0]).index(tokenizer.mask_token_id)

# What is the missing word? We compute the probabilities over the
# vocabulary. (The softmax is optional.)
output_probs = torch.softmax(mlm(masked_sentence.input_ids).logits[0,
mask_position], 0)
```

```
# Alternative 1: print the highest-scoring guess.
print(tokenizer.decode(output_probs.argmax()))
print()

# Alternative 2: print the 5 highest-scoring guesses.
topk = output_probs.topk(5)
for i, logit in zip(topk.indices, topk.values):
    word = tokenizer.decode(i)
    print(f'{word}: {logit.item():.3f}')
```

Fine-tuning a BERT model for document classification

In `__init__`(at the end):

```
# Output unit:
self.output_head = nn.Linear(hidden_size, nbr_classes)
```

In `__forward__`:

```
# We get the BERT output at the top layer. This gives us a tensor of
# the shape (n_docs, max_length, hidden_size).
bert_output = self.bert_model(Xbatch,
                              attention_mask=Xmask).last_hidden_state

# We then extract the token representations at the first position in
# each document, corresponding to the initial [CLS] dummy.
# The shape of this tensor is (n_docs, hidden_size)
cls_representations = bert_output[:, 0]

# Apply the output unit. The shape is now (n_docs, n_classes)
logits = self.output_head(cls_representations)

return logits
```

Exercise 2: Generation exercise

Summarization:

We can apply the usual machinery for generation. Typically, the scores are somewhat higher when using beam search. The following shows a solution. This snippet should be inserted inside the for loop after computing the `input_ids` tensor.

```
output_ids = model.generate(
      input_ids,
      num_beams=4,
      max_length=100,
      no_repeat_ngram_size=2
)
t5_summary = tokenizer.decode(output_ids[0], skip_special_tokens=True)
```

Open-domain dialogue:

The following shows a modified implementation that takes the conversation history into account. This solution uses the regular greedy decoding but we could also include sampling etc if we think that the generated answers are too bland.

```
import torch

chat_history_ids = tokenizer.encode(tokenizer.eos_token,
return_tensors='pt')

while True:
   user_input = input('>> User: ')

   input_ids = tokenizer(user_input + tokenizer.eos_token,
return_tensors='pt').input_ids

   all_input_ids = torch.cat([chat_history_ids, input_ids], dim=-1)

   chat_history_ids = model.generate(all_input_ids,
                             max_length=200,
                             #top_k=5,
                             #do_sample=True,
                             #num_beams=4,
                             pad_token_id=tokenizer.eos_token_id)
```

```
    # As in the GPT-2 example above, the generated text includes the
"prompt", so
    # we remove it. Then we decode as above.
    input_len = all_input_ids.shape[1]
    print("DialoGPT: {}".format(tokenizer.decode(chat_history_ids[:,
input_len:][0],
                                skip_special_tokens=True)))
```