Solution to the BERT exercise

Fine-tuning a BERT model for document classification

```
class TextClassifierUsingBERT(nn.Module):

    def __init__(self, model_name, n_labels):
        super().__init__()
        self.loss_fn = nn.CrossEntropyLoss()
        print('Loading pre-trained model...')
        self.bert_model = AutoModel.from_pretrained(model_name)
        hidden_size = self.bert_model.config.hidden_size
        self.classifier_head = nn.Linear(in_features=hidden_size,
                                         out_features=n_labels)

    def forward(self, input_ids, attention_mask, labels=None):
         # shape of input_ids: batch_size, number of tokens
         # shape of bert_outputs:
         #    batch size, number of tokens, representation dim
         bert_outputs = self.bert_model(input_ids,
            attention_mask=attention_mask).last_hidden_state
         # shape of document_representations:
         #    batch_size, representation dim
        document_representations = bert_outputs[:, 0]
         # shape of logits:
         #    batch_size, n_labels
        logits = self.classifier_head(document_representations)

        # Return format expected by HuggingFace Trainer.
        if labels is not None:
          return { 'loss': self.loss_fn(logits, labels),
                   'logits': logits }
        else:
          return { 'logits': logits }
```

Masked language modeling

```
import torch
masked_sentence = tokenizer('I went to the [MASK] to learn how to code .',
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, prob in zip(topk.indices, topk.values):
  word = tokenizer.decode(i)
  print(f'{word}: {prob.item():.3f}')
```