lab06

June 1, 2023

0.1 PyTorch tutorial copy paste

[1]:
```
from torchtext.data.utils import get_tokenizer
from torchtext.vocab import build_vocab_from_iterator
from torchtext.datasets import multi30k, Multi30k
from typing import Iterable, List


# We need to modify the URLs for the dataset since the links to the original␣
 ↪dataset are broken
# Refer to https://github.com/pytorch/text/issues/1756#issuecomment-1163664163␣
 ↪for more info
multi30k.URL["train"] = "https://raw.githubusercontent.com/neychev/
 ↪small_DL_repo/master/datasets/Multi30k/training.tar.gz"
multi30k.URL["valid"] = "https://raw.githubusercontent.com/neychev/
 ↪small_DL_repo/master/datasets/Multi30k/validation.tar.gz"

SRC_LANGUAGE = 'de'
TGT_LANGUAGE = 'en'

# Place-holders
token_transform = {}
vocab_transform = {}
```

[ ]:
```
%pip install spacy sacrebleu torchdata -U
!python -m spacy download en_core_web_sm
!python -m spacy download de_core_news_sm
```

[3]:
```
token_transform[SRC_LANGUAGE] = get_tokenizer('spacy',␣
 ↪language='de_core_news_sm')
token_transform[TGT_LANGUAGE] = get_tokenizer('spacy',␣
 ↪language='en_core_web_sm')


# helper function to yield list of tokens
def yield_tokens(data_iter: Iterable, language: str) -> Iterable[str]:
    language_index = {SRC_LANGUAGE: 0, TGT_LANGUAGE: 1}
```

```
    for data_sample in data_iter:
        yield token_transform[language](data_sample[language_index[language]])

# Define special symbols and indices
UNK_IDX, PAD_IDX, BOS_IDX, EOS_IDX = 0, 1, 2, 3
# Make sure the tokens are in order of their indices to properly insert them in␣
 ↪vocab
special_symbols = ['<unk>', '<pad>', '<bos>', '<eos>']

for ln in [SRC_LANGUAGE, TGT_LANGUAGE]:
    # Training data Iterator
    train_iter = list(Multi30k(split='train', language_pair=(SRC_LANGUAGE,␣
 ↪TGT_LANGUAGE)))
    # Create torchtext's Vocab object
    vocab_transform[ln] = build_vocab_from_iterator(yield_tokens(train_iter,␣
 ↪ln),
                                                    min_freq=1,
                                                    specials=special_symbols,
                                                    special_first=True)

# Set ``UNK_IDX`` as the default index. This index is returned when the token␣
 ↪is not found.
# If not set, it throws ``RuntimeError`` when the queried token is not found in␣
 ↪the Vocabulary.
for ln in [SRC_LANGUAGE, TGT_LANGUAGE]:
  vocab_transform[ln].set_default_index(UNK_IDX)
```

[4]:
```
from torch import Tensor
import torch
import torch.nn as nn
from torch.nn import Transformer
import math
DEVICE = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# helper Module that adds positional encoding to the token embedding to␣
 ↪introduce a notion of word order.
class PositionalEncoding(nn.Module):
    def __init__(self,
                 emb_size: int,
                 dropout: float,
                 maxlen: int = 5000):
        super(PositionalEncoding, self).__init__()
        den = torch.exp(- torch.arange(0, emb_size, 2)* math.log(10000) /␣
 ↪emb_size)
        pos = torch.arange(0, maxlen).reshape(maxlen, 1)
```

```
        pos_embedding = torch.zeros((maxlen, emb_size))
        pos_embedding[:, 0::2] = torch.sin(pos * den)
        pos_embedding[:, 1::2] = torch.cos(pos * den)
        pos_embedding = pos_embedding.unsqueeze(-2)

        self.dropout = nn.Dropout(dropout)
        self.register_buffer('pos_embedding', pos_embedding)

    def forward(self, token_embedding: Tensor):
        return self.dropout(token_embedding + self.pos_embedding[:
 ↪token_embedding.size(0), :])

# helper Module to convert tensor of input indices into corresponding tensor of␣
 ↪token embeddings
class TokenEmbedding(nn.Module):
    def __init__(self, vocab_size: int, emb_size):
        super(TokenEmbedding, self).__init__()
        self.embedding = nn.Embedding(vocab_size, emb_size)
        self.emb_size = emb_size

    def forward(self, tokens: Tensor):
        return self.embedding(tokens.long()) * math.sqrt(self.emb_size)

# Seq2Seq Network
class Seq2SeqTransformer(nn.Module):
    def __init__(self,
                 num_encoder_layers: int,
                 num_decoder_layers: int,
                 emb_size: int,
                 nhead: int,
                 src_vocab_size: int,
                 tgt_vocab_size: int,
                 dim_feedforward: int = 512,
                 dropout: float = 0.1):
        super(Seq2SeqTransformer, self).__init__()
        self.transformer = Transformer(d_model=emb_size,
                                       nhead=nhead,
                                       num_encoder_layers=num_encoder_layers,
                                       num_decoder_layers=num_decoder_layers,
                                       dim_feedforward=dim_feedforward,
                                       dropout=dropout)
        self.generator = nn.Linear(emb_size, tgt_vocab_size)
        self.src_tok_emb = TokenEmbedding(src_vocab_size, emb_size)
        self.tgt_tok_emb = TokenEmbedding(tgt_vocab_size, emb_size)
        self.positional_encoding = PositionalEncoding(
            emb_size, dropout=dropout)
```

```
    def forward(self,
                src: Tensor,
                trg: Tensor,
                src_mask: Tensor,
                tgt_mask: Tensor,
                src_padding_mask: Tensor,
                tgt_padding_mask: Tensor,
                memory_key_padding_mask: Tensor):
        src_emb = self.positional_encoding(self.src_tok_emb(src))
        tgt_emb = self.positional_encoding(self.tgt_tok_emb(trg))
        outs = self.transformer(src_emb, tgt_emb, src_mask, tgt_mask, None,
                                src_padding_mask, tgt_padding_mask,␣
↪memory_key_padding_mask)
        return self.generator(outs)

    def encode(self, src: Tensor, src_mask: Tensor):
        return self.transformer.encoder(self.positional_encoding(
                            self.src_tok_emb(src)), src_mask)

    def decode(self, tgt: Tensor, memory: Tensor, tgt_mask: Tensor):
        return self.transformer.decoder(self.positional_encoding(
                          self.tgt_tok_emb(tgt)), memory,
                          tgt_mask)
```

[5]:
```
def generate_square_subsequent_mask(sz):
    mask = (torch.triu(torch.ones((sz, sz), device=DEVICE)) == 1).transpose(0,␣
↪1)
    mask = mask.float().masked_fill(mask == 0, float('-inf')).masked_fill(mask␣
↪== 1, float(0.0))
    return mask


def create_mask(src, tgt):
    src_seq_len = src.shape[0]
    tgt_seq_len = tgt.shape[0]

    tgt_mask = generate_square_subsequent_mask(tgt_seq_len)
    src_mask = torch.zeros((src_seq_len, src_seq_len),device=DEVICE).type(torch.
↪bool)

    src_padding_mask = (src == PAD_IDX).transpose(0, 1)
    tgt_padding_mask = (tgt == PAD_IDX).transpose(0, 1)
    return src_mask, tgt_mask, src_padding_mask, tgt_padding_mask
```

[6]:
```
torch.manual_seed(0)

SRC_VOCAB_SIZE = len(vocab_transform[SRC_LANGUAGE])
```

```
TGT_VOCAB_SIZE = len(vocab_transform[TGT_LANGUAGE])
EMB_SIZE = 512
NHEAD = 8
FFN_HID_DIM = 512
BATCH_SIZE = 128
NUM_ENCODER_LAYERS = 3
NUM_DECODER_LAYERS = 3

transformer = Seq2SeqTransformer(NUM_ENCODER_LAYERS, NUM_DECODER_LAYERS,␣
 ↪EMB_SIZE,
                                 NHEAD, SRC_VOCAB_SIZE, TGT_VOCAB_SIZE,␣
 ↪FFN_HID_DIM)

for p in transformer.parameters():
    if p.dim() > 1:
        nn.init.xavier_uniform_(p)

transformer = transformer.to(DEVICE)

loss_fn = torch.nn.CrossEntropyLoss(ignore_index=PAD_IDX)

optimizer = torch.optim.Adam(transformer.parameters(), lr=0.0001, betas=(0.9, 0.
 ↪98), eps=1e-9)
```

[7]:
```
from torch.nn.utils.rnn import pad_sequence

# helper function to club together sequential operations
def sequential_transforms(*transforms):
    def func(txt_input):
        for transform in transforms:
            txt_input = transform(txt_input)
        return txt_input
    return func

# function to add BOS/EOS and create tensor for input sequence indices
def tensor_transform(token_ids: List[int]):
    return torch.cat((torch.tensor([BOS_IDX]),
                      torch.tensor(token_ids),
                      torch.tensor([EOS_IDX])))

# ``src`` and ``tgt`` language text transforms to convert raw strings into␣
 ↪tensors indices
text_transform = {}
for ln in [SRC_LANGUAGE, TGT_LANGUAGE]:
    text_transform[ln] = sequential_transforms(token_transform[ln],␣
 ↪#Tokenization
```

```
                                                vocab_transform[ln],␣
 ↪#Numericalization
                                                tensor_transform) # Add BOS/EOS␣
 ↪and create tensor


# function to collate data samples into batch tensors
def collate_fn(batch):
    src_batch, tgt_batch = [], []
    for src_sample, tgt_sample in batch:
        src_batch.append(text_transform[SRC_LANGUAGE](src_sample.rstrip("\n")))
        tgt_batch.append(text_transform[TGT_LANGUAGE](tgt_sample.rstrip("\n")))

    src_batch = pad_sequence(src_batch, padding_value=PAD_IDX)
    tgt_batch = pad_sequence(tgt_batch, padding_value=PAD_IDX)
    return src_batch, tgt_batch
```

[8]:
```
from torch.utils.data import DataLoader

def train_epoch(model, optimizer):
    model.train()
    losses = 0
    train_iter = Multi30k(split='train', language_pair=(SRC_LANGUAGE,␣
 ↪TGT_LANGUAGE))
    train_dataloader = DataLoader(train_iter, batch_size=BATCH_SIZE,␣
 ↪collate_fn=collate_fn)

    for src, tgt in train_dataloader:
        src = src.to(DEVICE)
        tgt = tgt.to(DEVICE)

        tgt_input = tgt[:-1, :]

        src_mask, tgt_mask, src_padding_mask, tgt_padding_mask =␣
 ↪create_mask(src, tgt_input)

        logits = model(src, tgt_input, src_mask, tgt_mask,src_padding_mask,␣
 ↪tgt_padding_mask, src_padding_mask)

        optimizer.zero_grad()

        tgt_out = tgt[1:, :]
        loss = loss_fn(logits.reshape(-1, logits.shape[-1]), tgt_out.
 ↪reshape(-1))
        loss.backward()
```

```
        optimizer.step()
        losses += loss.item()

    return losses / len(list(train_dataloader))


def evaluate(model):
    model.eval()
    losses = 0

    val_iter = Multi30k(split='valid', language_pair=(SRC_LANGUAGE,␣
 ↪TGT_LANGUAGE))
    val_dataloader = DataLoader(val_iter, batch_size=BATCH_SIZE,␣
 ↪collate_fn=collate_fn)

    for src, tgt in val_dataloader:
        src = src.to(DEVICE)
        tgt = tgt.to(DEVICE)

        tgt_input = tgt[:-1, :]

        src_mask, tgt_mask, src_padding_mask, tgt_padding_mask =␣
 ↪create_mask(src, tgt_input)

        logits = model(src, tgt_input, src_mask, tgt_mask,src_padding_mask,␣
 ↪tgt_padding_mask, src_padding_mask)

        tgt_out = tgt[1:, :]
        loss = loss_fn(logits.reshape(-1, logits.shape[-1]), tgt_out.
 ↪reshape(-1))
        losses += loss.item()

    return losses / len(list(val_dataloader))
```

[35]:
```
from timeit import default_timer as timer
NUM_EPOCHS = 18

for epoch in range(1, NUM_EPOCHS+1):
    start_time = timer()
    train_loss = train_epoch(transformer, optimizer)
    end_time = timer()
    val_loss = evaluate(transformer)
    print((f"Epoch: {epoch}, Train loss: {train_loss:.3f}, Val loss: {val_loss:.
 ↪3f}, "f"Epoch time = {(end_time - start_time):.3f}s"))
```

```
/Users/quentinfisch/Documents/EPITA/ING2/SCIA/S8/NLP1/.venv/lib/python3.9/site-
packages/torch/nn/functional.py:4999: UserWarning: Support for mismatched
```

```
key_padding_mask and attn_mask is deprecated. Use same type for both instead.
  warnings.warn(
/Users/quentinfisch/Documents/EPITA/ING2/SCIA/S8/NLP1/.venv/lib/python3.9/site-
packages/torch/utils/data/datapipes/iter/combining.py:297: UserWarning: Some
child DataPipes are not exhausted when __iter__ is called. We are resetting the
buffer and each child DataPipe will read from the start again.
  warnings.warn("Some child DataPipes are not exhausted when __iter__ is called.
We are resetting "

Epoch: 1, Train loss: 5.342, Val loss: 4.107, Epoch time = 343.495s
Epoch: 2, Train loss: 3.760, Val loss: 3.308, Epoch time = 319.152s
Epoch: 3, Train loss: 3.156, Val loss: 2.893, Epoch time = 323.218s
Epoch: 4, Train loss: 2.765, Val loss: 2.631, Epoch time = 328.611s
Epoch: 5, Train loss: 2.478, Val loss: 2.436, Epoch time = 329.120s
Epoch: 6, Train loss: 2.249, Val loss: 2.300, Epoch time = 338.191s
Epoch: 7, Train loss: 2.057, Val loss: 2.189, Epoch time = 339.081s
Epoch: 8, Train loss: 1.893, Val loss: 2.123, Epoch time = 335.729s
Epoch: 9, Train loss: 1.754, Val loss: 2.052, Epoch time = 333.173s
Epoch: 10, Train loss: 1.628, Val loss: 2.005, Epoch time = 336.888s
Epoch: 11, Train loss: 1.519, Val loss: 1.975, Epoch time = 330.446s
Epoch: 12, Train loss: 1.417, Val loss: 1.956, Epoch time = 329.546s
Epoch: 13, Train loss: 1.331, Val loss: 1.965, Epoch time = 328.958s
Epoch: 14, Train loss: 1.249, Val loss: 1.960, Epoch time = 321.621s
Epoch: 15, Train loss: 1.171, Val loss: 1.915, Epoch time = 320.837s
Epoch: 16, Train loss: 1.100, Val loss: 1.916, Epoch time = 309.242s
Epoch: 17, Train loss: 1.036, Val loss: 1.913, Epoch time = 311.059s
Epoch: 18, Train loss: 0.976, Val loss: 1.922, Epoch time = 316.606s
```

[ ]:
```python
# save model
torch.save(transformer.state_dict(), 'model.pt')
```

[13]:
```python
# load model.pt
transformer.load_state_dict(torch.load('model.pt'))
```

[13]:
```
<All keys matched successfully>
```

[10]:
```python
# function to generate output sequence using greedy algorithm
def greedy_decode(model, src, src_mask, max_len, start_symbol):
    src = src.to(DEVICE)
    src_mask = src_mask.to(DEVICE)

    memory = model.encode(src, src_mask)
    ys = torch.ones(1, 1).fill_(start_symbol).type(torch.long).to(DEVICE)
    for i in range(max_len-1):
        memory = memory.to(DEVICE)
        tgt_mask = (generate_square_subsequent_mask(ys.size(0))
                    .type(torch.bool)).to(DEVICE)
        out = model.decode(ys, memory, tgt_mask)
```

```
        out = out.transpose(0, 1)
        prob = model.generator(out[:, -1])
        _, next_word = torch.max(prob, dim=1)
        next_word = next_word.item()

        ys = torch.cat([ys,
                        torch.ones(1, 1).type_as(src.data).fill_(next_word)],␣
 ↪dim=0)
        if next_word == EOS_IDX:
            break
    return ys


# actual function to translate input sentence into target language
def translate(model: torch.nn.Module, src_sentence: str):
    model.eval()
    src = text_transform[SRC_LANGUAGE](src_sentence).view(-1, 1)
    num_tokens = src.shape[0]
    src_mask = (torch.zeros(num_tokens, num_tokens)).type(torch.bool)
    tgt_tokens = greedy_decode(
        model,  src, src_mask, max_len=num_tokens + 5, start_symbol=BOS_IDX).
 ↪flatten()
    return " ".join(vocab_transform[TGT_LANGUAGE].lookup_tokens(list(tgt_tokens.
 ↪cpu().numpy()))).replace("<bos>", "").replace("<eos>", "")
```

[11]:
```
print(translate(transformer, "Eine Gruppe von Menschen steht vor einem Iglu ."))
```

```
A group of people stand in front of an igloo .
```

0.2 Theoritical questions answer

- In the positional encoding, why are we using a combination of sinus and cosinus?
- In the Seq2SeqTransformer class,
  - What is the parameter nhead for?
  - What is the point of the generator?
- Describe the goal of the create_mask function. Why does it handle differently the source and target masks?

In the positional encoding, why are we using a combination of sinus and cosinus? The positional encoding is a way to encode the position of the words in the sentence. We use sinus and cosinus to encode the position of the words in the sentence. These functions are bounded so the modification of the embedding is only a little, to not change the nature of it. We combine a sinus and a cosinus because they have different periods, which allows us to have a different encoding for each position, thus avoiding the problem that we could have two words with the same position because of the periodicity of the sinus and cosinus.

In the Seq2SeqTransformer class, What is the parameter nhead for? The parameter nhead is the number of heads in the multihead attention. It is the number of parallel attention

layers. It is used to increase the representational power of the model.

In the Seq2SeqTransformer class, What is the point of the generator? The generator is a linear layer that takes the output of the decoder and returns the logits. It maps the output of the decoder to the vocabulary.

Describe the goal of the create_mask function. Why does it handle differently the source and target masks? The goal of the create_mask function is to create a mask for the source and the target. The source mask is used to mask the padding tokens. The target mask is used to mask the padding tokens and the future tokens.

0.3 Decoding functions

[12]:
```python
def top_k_sampling_with_temperature(model: torch.nn.Module,
                                    src: torch.Tensor,
                                    src_mask: torch.Tensor,
                                    max_len: int,
                                    start_symbol: int,
                                    k: int,
                                    temperature: float) -> torch.Tensor:
    """
    Top K sampling algorithm with temperature

    -------
    Args:
    model: torch.nn.Module
        Transformer model
    src: torch.Tensor
        Source tensor
    src_mask: torch.Tensor
        Source mask tensor
    max_len: int
        Maximum length of the output sequence
    start_symbol: int
        Start symbol
    k: int
        Top K
    temperature: float
        Temperature

    -------
    Returns:
    ys: torch.Tensor
        Output tensor
    """
    src = src.to(DEVICE)
    src_mask = src_mask.to(DEVICE)
```

```
    memory = model.encode(src, src_mask)
    ys = torch.ones(1, 1).fill_(start_symbol).type(torch.long).to(DEVICE)
    for i in range(max_len-1):
        memory = memory.to(DEVICE)
        tgt_mask = (generate_square_subsequent_mask(ys.size(0))
                    .type(torch.bool)).to(DEVICE)
        out = model.decode(ys, memory, tgt_mask)
        out = out.transpose(0, 1)
        prob = model.generator(out[:, -1])
        prob = torch.div(prob, temperature)
        prob = torch.softmax(prob, dim=1)

        # Choose top k words from the probability distribution
        top_k_prob, top_k_idx = torch.topk(prob, k=k, dim=1)

        # Choose next word from remaining words using multinomial distribution
        next_word = torch.multinomial(top_k_prob, num_samples=1)
        next_word = top_k_idx[torch.arange(top_k_idx.size(0)), next_word.
 ↪squeeze(1)]
        next_word = next_word.item()

        ys = torch.cat([ys,
                        torch.ones(1, 1).type_as(src.data).fill_(next_word)],␣
 ↪dim=0)
        if next_word == EOS_IDX:
            break
    return ys

def translate_top_k(model: torch.nn.Module, src_sentence: str, k: int,␣
 ↪temperature: float) -> str:
    """
    Translate source sentence using top k sampling algorithm with temperature

    -------
    Args:
    model: torch.nn.Module
        Transformer model
    src_sentence: str
        Source sentence
    k: int
        Top K
    temperature: float
        Temperature

    -------
    Returns:
```

```
    str
        The translated sentence
    """
    model.eval()
    src = text_transform[SRC_LANGUAGE](src_sentence).view(-1, 1)
    num_tokens = src.shape[0]
    src_mask = (torch.zeros(num_tokens, num_tokens)).type(torch.bool)
    tgt_tokens = top_k_sampling_with_temperature(
        model,  src, src_mask, max_len=num_tokens + 5, start_symbol=BOS_IDX,␣
↪k=k, temperature=temperature).flatten()
    return " ".join(vocab_transform[TGT_LANGUAGE].lookup_tokens(list(tgt_tokens.
↪cpu().numpy()))).replace("<bos>", "").replace("<eos>", "")
```

[13]:
```
def top_p_sampling_with_temperature(model: torch.nn.Module,
                                    src: torch.Tensor,
                                    src_mask: torch.Tensor,
                                    max_len: int,
                                    start_symbol: int,
                                    p: int,
                                    temperature: float) -> torch.Tensor:
    """
    Top P sampling algorithm with temperature

    -------
    Args:
    model: torch.nn.Module
        Transformer model
    src: torch.Tensor
        Source tensor
    src_mask: torch.Tensor
        Source mask tensor
    max_len: int
        Maximum length of the output sequence
    start_symbol: int
        Start symbol
    p: int
        Top P
    temperature: float
        Temperature

    -------
    Returns:
    ys: torch.Tensor
        Output tensor
    """
    src = src.to(DEVICE)
    src_mask = src_mask.to(DEVICE)
```

```
    memory = model.encode(src, src_mask)
    ys = torch.ones(1, 1).fill_(start_symbol).type(torch.long).to(DEVICE)
    for i in range(max_len-1):
        memory = memory.to(DEVICE)
        tgt_mask = (generate_square_subsequent_mask(ys.size(0))
                    .type(torch.bool)).to(DEVICE)
        out = model.decode(ys, memory, tgt_mask)
        out = out.transpose(0, 1)
        prob = model.generator(out[:, -1])
        prob = torch.div(prob, temperature)
        prob = torch.softmax(prob, dim=1)
        sorted_prob, sorted_idx = torch.sort(prob, descending=True, dim=1)

        # Calculate cumulative probabilities and remove tokens with cumulative␣
 ↪probability above p
        cumulative_prob = torch.cumsum(sorted_prob, dim=1)
        sorted_idx_to_remove = cumulative_prob > p
        sorted_idx_to_remove[:, 1:] = sorted_idx_to_remove[:, :-1].clone()
        sorted_idx_to_remove[:, 0] = 0
        sorted_idx_to_remove = sorted_idx_to_remove.type(torch.bool)
        sorted_idx[sorted_idx_to_remove] = -1
        sorted_idx = sorted_idx[sorted_idx != -1]
        sorted_idx = sorted_idx.type(torch.float)

        # Choose next word from remaining words using multinomial distribution
        next_word = torch.multinomial(sorted_idx, num_samples=1)
        next_word = sorted_idx[next_word]
        next_word = next_word.item()

        ys = torch.cat([ys,
                        torch.ones(1, 1).type_as(src.data).fill_(next_word)],␣
 ↪dim=0)
        if next_word == EOS_IDX:
            break
    return ys

def translate_top_p(model: torch.nn.Module, src_sentence: str, p: int,␣
 ↪temperature: float) -> str:
    """
    Translate source sentence using top p sampling algorithm with temperature

    -------
    Args:
    model: torch.nn.Module
        Transformer model
    src_sentence: str
```

```
        Source sentence
    p: int
        Top P
    temperature: float
        Temperature

    -------
    Returns:
    str
        The translated sentence
    """
    model.eval()
    src = text_transform[SRC_LANGUAGE](src_sentence).view(-1, 1)
    num_tokens = src.shape[0]
    src_mask = (torch.zeros(num_tokens, num_tokens)).type(torch.bool)
    tgt_tokens = top_p_sampling_with_temperature(
        model, src, src_mask, max_len=num_tokens + 5, start_symbol=BOS_IDX,␣
↪p=p, temperature=temperature).flatten()
    return " ".join(vocab_transform[TGT_LANGUAGE].lookup_tokens(list(tgt_tokens.
↪cpu().numpy()))).replace("<bos>", "").replace("<eos>", "")
```

[161]:
```
print("Greedy translate:")
print(translate(transformer, "Eine Gruppe von Menschen steht vor einem Iglu ."))

print("\nTop K sampling with temperature (k=5, temperature 1.0, 0.5 and 0.1):")
print(translate_top_k(transformer, "Eine Gruppe von Menschen steht vor einem␣
↪Iglu .", k=5, temperature=1.0))
print(translate_top_k(transformer, "Eine Gruppe von Menschen steht vor einem␣
↪Iglu .", k=5, temperature=0.5))
print(translate_top_k(transformer, "Eine Gruppe von Menschen steht vor einem␣
↪Iglu .", k=5, temperature=0.1))

print("\nTop P sampling with temperature (p=0.9, temperature 1.0, 0.5 and 0.1):
↪")
print(translate_top_p(transformer, "Eine Gruppe von Menschen steht vor einem␣
↪Iglu .", p=0.9, temperature=1.0))
print(translate_top_p(transformer, "Eine Gruppe von Menschen steht vor einem␣
↪Iglu .", p=0.9, temperature=0.5))
print(translate_top_p(transformer, "Eine Gruppe von Menschen steht vor einem␣
↪Iglu .", p=0.9, temperature=0.1))

print("\nTop K sampling with temperature (k=10, temperature 1.0, 0.5 and 0.1):")
print(translate_top_k(transformer, "Eine Gruppe von Menschen steht vor einem␣
↪Iglu .", k=10, temperature=1.0))
print(translate_top_k(transformer, "Eine Gruppe von Menschen steht vor einem␣
↪Iglu .", k=10, temperature=0.5))
```

```
print(translate_top_k(transformer, "Eine Gruppe von Menschen steht vor einem␣
 ↪Iglu .", k=10, temperature=0.1))

print("\nTop P sampling with temperature (p=0.5, temperature 1.0, 0.5 and 0.1):
 ↪")
print(translate_top_p(transformer, "Eine Gruppe von Menschen steht vor einem␣
 ↪Iglu .", p=0.5, temperature=1.0))
print(translate_top_p(transformer, "Eine Gruppe von Menschen steht vor einem␣
 ↪Iglu .", p=0.5, temperature=0.5))
print(translate_top_p(transformer, "Eine Gruppe von Menschen steht vor einem␣
 ↪Iglu .", p=0.5, temperature=0.1))

print("\nTop K sampling with temperature (k=50, temperature 1.0, 0.5 and 0.1):")
print(translate_top_k(transformer, "Eine Gruppe von Menschen steht vor einem␣
 ↪Iglu .", k=50, temperature=1.0))
print(translate_top_k(transformer, "Eine Gruppe von Menschen steht vor einem␣
 ↪Iglu .", k=50, temperature=0.5))
print(translate_top_k(transformer, "Eine Gruppe von Menschen steht vor einem␣
 ↪Iglu .", k=50, temperature=0.1))

print("\nTop K sampling with temperature (k=100, temperature 1.0, 0.5 and 0.1):
 ↪")
print(translate_top_k(transformer, "Eine Gruppe von Menschen steht vor einem␣
 ↪Iglu .", k=100, temperature=1.0))
print(translate_top_k(transformer, "Eine Gruppe von Menschen steht vor einem␣
 ↪Iglu .", k=100, temperature=0.5))
print(translate_top_k(transformer, "Eine Gruppe von Menschen steht vor einem␣
 ↪Iglu .", k=100, temperature=0.1))
```

```
Greedy translate:
 A group of people stand in front of an igloo .

Top K sampling with temperature (k=5, temperature 1.0, 0.5 and 0.1):
 A group of people stand in front of an igloo
 A group of people stand in front of an igloo .
 A group of people stand in front of an igloo .

Top P sampling with temperature (p=0.9, temperature 1.0, 0.5 and 0.1):
 A group of people standing in front an pledge Artists .
 A group of people stand in front of an overpass .
 A group of people stand in front of an igloo .

Top K sampling with temperature (k=10, temperature 1.0, 0.5 and 0.1):
 A group of people stand in front of an igloo .
 A group of people stand in front of an igloo .
 A group of people stand in front of an igloo .
```

```
Top P sampling with temperature (p=0.5, temperature 1.0, 0.5 and 0.1):
 A group of people stand in front of an Ohio calligraphy .
 A group of people stand in front of an igloo .
 A group of people stand in front of an igloo .

Top K sampling with temperature (k=50, temperature 1.0, 0.5 and 0.1):
 A group of people stand in front of an auditorium .
 A group of people stand in front of an olive .
 A group of people standing in front of an igloo .

Top K sampling with temperature (k=100, temperature 1.0, 0.5 and 0.1):
 A group of people stand in front of an unidentified room .
 A group of people stand in front of an igloo .
 A group of people stand in front of an igloo .
```

[162]:
```
print("Greedy translate:")
print(translate(transformer, "Während der Mann in der Küche ist, sitzt die Frau␣
 ↪im Wohnzimmer ."))

print("\nTop K sampling with temperature (k=5, temperature 1.0, 0.5 and 0.1):")
print(translate_top_k(transformer, "Während der Mann in der Küche ist, sitzt␣
 ↪die Frau im Wohnzimmer .", k=5, temperature=1.0))
print(translate_top_k(transformer, "Während der Mann in der Küche ist, sitzt␣
 ↪die Frau im Wohnzimmer .", k=5, temperature=0.5))
print(translate_top_k(transformer, "Während der Mann in der Küche ist, sitzt␣
 ↪die Frau im Wohnzimmer .", k=5, temperature=0.1))

print("\nTop P sampling with temperature (p=0.9, temperature 1.0, 0.5 and 0.1):
 ↪")
print(translate_top_p(transformer, "Während der Mann in der Küche ist, sitzt␣
 ↪die Frau im Wohnzimmer .", p=0.9, temperature=1.0))
print(translate_top_p(transformer, "Während der Mann in der Küche ist, sitzt␣
 ↪die Frau im Wohnzimmer .", p=0.9, temperature=0.5))
print(translate_top_p(transformer, "Während der Mann in der Küche ist, sitzt␣
 ↪die Frau im Wohnzimmer .", p=0.9, temperature=0.1))

print("\nTop K sampling with temperature (k=10, temperature 1.0, 0.5 and 0.1):")
print(translate_top_k(transformer, "Während der Mann in der Küche ist, sitzt␣
 ↪die Frau im Wohnzimmer .", k=10, temperature=1.0))
print(translate_top_k(transformer, "Während der Mann in der Küche ist, sitzt␣
 ↪die Frau im Wohnzimmer .", k=10, temperature=0.5))
print(translate_top_k(transformer, "Während der Mann in der Küche ist, sitzt␣
 ↪die Frau im Wohnzimmer .", k=10, temperature=0.1))

print("\nTop P sampling with temperature (p=0.5, temperature 1.0, 0.5 and 0.1):
 ↪")
```

```
print(translate_top_p(transformer, "Während der Mann in der Küche ist, sitzt␣
 ↪die Frau im Wohnzimmer .", p=0.5, temperature=1.0))
print(translate_top_p(transformer, "Während der Mann in der Küche ist, sitzt␣
 ↪die Frau im Wohnzimmer .", p=0.5, temperature=0.5))
print(translate_top_p(transformer, "Während der Mann in der Küche ist, sitzt␣
 ↪die Frau im Wohnzimmer .", p=0.5, temperature=0.1))

print("\nTop K sampling with temperature (k=50, temperature 1.0, 0.5 and 0.1):")
print(translate_top_k(transformer, "Während der Mann in der Küche ist, sitzt␣
 ↪die Frau im Wohnzimmer .", k=50, temperature=1.0))
print(translate_top_k(transformer, "Während der Mann in der Küche ist, sitzt␣
 ↪die Frau im Wohnzimmer .", k=50, temperature=0.5))
print(translate_top_k(transformer, "Während der Mann in der Küche ist, sitzt␣
 ↪die Frau im Wohnzimmer .", k=50, temperature=0.1))

print("\nTop K sampling with temperature (k=100, temperature 1.0, 0.5 and 0.1):
 ↪")
print(translate_top_k(transformer, "Während der Mann in der Küche ist, sitzt␣
 ↪die Frau im Wohnzimmer .", k=100, temperature=1))
print(translate_top_k(transformer, "Während der Mann in der Küche ist, sitzt␣
 ↪die Frau im Wohnzimmer .", k=10, temperature=0.5))
print(translate_top_k(transformer, "Während der Mann in der Küche ist, sitzt␣
 ↪die Frau im Wohnzimmer .", k=100, temperature=0.1))
```

```
Greedy translate:
 While the man in a kitchen is sitting in the living room .

Top K sampling with temperature (k=5, temperature 1.0, 0.5 and 0.1):
 While sitting in the kitchen in a kitchen , the woman in a living room .
 The man in the kitchen is sitting in the living room .
 While sitting in the kitchen in a kitchen with the woman in her living room .

Top P sampling with temperature (p=0.9, temperature 1.0, 0.5 and 0.1):
 Skiers in the kitchen in the kitchen sitting in a warehouse .
 While seated in the kitchen , the man sits in a living room .
 While sitting in the kitchen in a living room .

Top K sampling with temperature (k=10, temperature 1.0, 0.5 and 0.1):
 The man in the kitchen is sitting in a room .
 A man in a kitchen is sitting in the living room .
 The man in the kitchen is sitting in the living room .

Top P sampling with temperature (p=0.5, temperature 1.0, 0.5 and 0.1):
 While sitting in the kitchen in a living room .
 While sitting in the kitchen in a living room .
 While the man in a kitchen is sitting in the living room .
```

```
Top K sampling with temperature (k=50, temperature 1.0, 0.5 and 0.1):
 The Man in the kitchen is sitting in a living room .
 While sitting in the kitchen in a kitchen with the woman in his living room .
 While the man in a kitchen is sitting in the living room .

Top K sampling with temperature (k=100, temperature 1.0, 0.5 and 0.1):
 On the black kitchen , the man in a kitchen in the living room .
 While seated in the kitchen in a kitchen with the woman in his living room .
 The man in the kitchen is sitting in the living room .
```

[163]:
```
print("Greedy translate:")
print(translate(transformer, "Die Frau geht in den Wald und sammelt Pilze ."))

print("\nTop K sampling with temperature (k=5, temperature 1.0, 0.5 and 0.1):")
print(translate_top_k(transformer, "Die Frau geht in den Wald und sammelt Pilze␣
 ↪.", k=5, temperature=1.0))
print(translate_top_k(transformer, "Die Frau geht in den Wald und sammelt Pilze␣
 ↪.", k=5, temperature=0.5))
print(translate_top_k(transformer, "Die Frau geht in den Wald und sammelt Pilze␣
 ↪.", k=5, temperature=0.1))

print("\nTop P sampling with temperature (p=0.9, temperature 1.0, 0.5 and 0.1):
 ↪")
print(translate_top_p(transformer, "Die Frau geht in den Wald und sammelt Pilze␣
 ↪.", p=0.9, temperature=1.0))
print(translate_top_p(transformer, "Die Frau geht in den Wald und sammelt Pilze␣
 ↪.", p=0.9, temperature=0.5))
print(translate_top_p(transformer, "Die Frau geht in den Wald und sammelt Pilze␣
 ↪.", p=0.9, temperature=0.1))

print("\nTop K sampling with temperature (k=10, temperature 1.0, 0.5 and 0.1):")
print(translate_top_k(transformer, "Die Frau geht in den Wald und sammelt Pilze␣
 ↪.", k=10, temperature=1.0))
print(translate_top_k(transformer, "Die Frau geht in den Wald und sammelt Pilze␣
 ↪.", k=10, temperature=0.5))
print(translate_top_k(transformer, "Die Frau geht in den Wald und sammelt Pilze␣
 ↪.", k=10, temperature=0.1))

print("\nTop P sampling with temperature (p=0.5, temperature 1.0, 0.5 and 0.1):
 ↪")
print(translate_top_p(transformer, "Die Frau geht in den Wald und sammelt Pilze␣
 ↪.", p=0.5, temperature=1.0))
print(translate_top_p(transformer, "Die Frau geht in den Wald und sammelt Pilze␣
 ↪.", p=0.5, temperature=0.5))
print(translate_top_p(transformer, "Die Frau geht in den Wald und sammelt Pilze␣
 ↪.", p=0.5, temperature=0.1))
```

```
print("\nTop K sampling with temperature (k=50, temperature 1.0, 0.5 and 0.1):")
print(translate_top_k(transformer, "Die Frau geht in den Wald und sammelt Pilze.", k=50, temperature=1.0))
print(translate_top_k(transformer, "Die Frau geht in den Wald und sammelt Pilze.", k=50, temperature=0.5))
print(translate_top_k(transformer, "Die Frau geht in den Wald und sammelt Pilze.", k=50, temperature=0.1))

print("\nTop K sampling with temperature (k=100, temperature 1.0, 0.5 and 0.1):")
print(translate_top_k(transformer, "Die Frau geht in den Wald und sammelt Pilze.", k=100, temperature=1))
print(translate_top_k(transformer, "Die Frau geht in den Wald und sammelt Pilze.", k=10, temperature=0.5))
print(translate_top_k(transformer, "Die Frau geht in den Wald und sammelt Pilze.", k=100, temperature=0.1))
```

```
Greedy translate:
 The woman walks in the woods and picking up books .

Top K sampling with temperature (k=5, temperature 1.0, 0.5 and 0.1):
 The woman is walking in the forest and gathering .
 The lady walks in the forest and picking up lunch .
 The woman walks in the woods and picking up books .

Top P sampling with temperature (p=0.9, temperature 1.0, 0.5 and 0.1):
 The lady walks in the woods strums everywhere .
 The woman walks in the forest examining properly time .
 The woman walks in the forest and picking up books .

Top K sampling with temperature (k=10, temperature 1.0, 0.5 and 0.1):
 The woman walks in the woods while picking up him .
 The woman is walking in the woods while picking time .
 The woman walks in the woods and picking up sale .

Top P sampling with temperature (p=0.5, temperature 1.0, 0.5 and 0.1):
 The woman walks in the woods gestures and wonderful .
 The woman walks in the woods and picking up customers .
 The woman walks in the woods and picking up books .

Top K sampling with temperature (k=50, temperature 1.0, 0.5 and 0.1):
 The woman walks in the forest as he picks books .
 The woman walks in the woods as if picking up .
 The woman walks in the woods and picking up books .

Top K sampling with temperature (k=100, temperature 1.0, 0.5 and 0.1):
```

```
The lady walks in the woods with steam and figures .
The woman is walking in the forest and picking up books .
The woman walks in the woods and picking up books .
```

When looking at these 3 examples, we note some interesting behavior.

The first example always gives the same translation for the main part of the sentence ("A group of people standing in front an…" or "A group of people stand in front an…", which have the same meaning in this case). However, the nomial group at the end of the sentence varies depending on the decoding method and parameters we use. Most of the time, the translation is correct and ends with "igloo", but we also observe funny endings such as "Ohio calligraphy".

The other two examples are also interesting, because none of the configurations manage to translate the sentence correctly. In the second example, the "woman" seems to be an issue for the model. Everytime it manages to translate it and adds it to the sentence, the "man" at the beginning disappears and some words are duplicated. Other translations that don't include the "woman" are correct about the "man", but the end of the sentence doesn't make sense… The third example always give a pretty accurate translation of the first part of the sentence, but the end is always wrong. None of the configuration get "mushrooms" right.

In general, it seems like a smaller temperature gives better results, but it's hard to see shapes of better success with k or p. The fact that we see better results using a smaller temperature comes from the fact that the smaller the temperature, the less random the sampling is. The smaller the temperature, the more the model will choose the most probable word, which is the one that is the most likely to be correct. However, if we use a temperature that is too small, the model will always choose the same word, which is not what we want. We want the model to be able to choose different words, but to choose the most probable ones. In our case, `0.1` seems to be a good temperature overall.

0.4 Compute the BLEU score of the model

[15]:
```
from sacrebleu import corpus_bleu, BLEU

def compute_bleu(model: torch.nn.Module,
                 data_loader: torch.utils.data.DataLoader,
                 translate_method=translate,
                 params = {}) -> BLEU:
    """
    Compute the sentence-level BLEU score

    -------
    Args:
    model: torch.nn.Module
        Transformer model
    data_loader: torch.utils.data.DataLoader
        DataLoader for the dataset
    translate_method: Callable
        Translation function
    params: dict
```

```
        Parameters for the translation function

    -------
    Returns:
    BLEUScore
        Sentence-level BLEU score
    """
    model.eval()
    refs = []
    preds = []
    with torch.no_grad():
        for src, tgt in data_loader:
            translated_sentence = translate_method(model, src, **params)
            refs.append(tgt)
            preds.append(translated_sentence)
    return corpus_bleu(preds, refs)
```

[17]:
```
test_iter = Multi30k(split='test', language_pair=(SRC_LANGUAGE, TGT_LANGUAGE))
print(compute_bleu(transformer, test_iter))
print(compute_bleu(transformer, test_iter, translate_method=translate_top_k,␣
 ↪params={"k": 5, "temperature": 1.0}))
print(compute_bleu(transformer, test_iter, translate_method=translate_top_k,␣
 ↪params={"k": 5, "temperature": 0.5}))
print(compute_bleu(transformer, test_iter, translate_method=translate_top_k,␣
 ↪params={"k": 5, "temperature": 0.1}))
```

```
BLEU = 0.32 9.3/0.2/0.1/0.1 (BP = 1.000 ratio = 13.286 hyp_len = 279 ref_len =
21)

/Users/quentinfisch/Documents/EPITA/ING2/SCIA/S8/NLP1/.venv/lib/python3.9/site-
packages/torch/utils/data/datapipes/iter/combining.py:297: UserWarning: Some
child DataPipes are not exhausted when __iter__ is called. We are resetting the
buffer and each child DataPipe will read from the start again.
  warnings.warn("Some child DataPipes are not exhausted when __iter__ is called.
We are resetting "

BLEU = 0.31 8.4/0.2/0.1/0.1 (BP = 1.000 ratio = 13.667 hyp_len = 287 ref_len =
21)
BLEU = 0.32 9.1/0.2/0.1/0.1 (BP = 1.000 ratio = 13.571 hyp_len = 285 ref_len =
21)
BLEU = 0.33 9.4/0.2/0.1/0.1 (BP = 1.000 ratio = 13.190 hyp_len = 277 ref_len =
21)
```

[174]:
```
test_iter = Multi30k(split='test', language_pair=(SRC_LANGUAGE, TGT_LANGUAGE))
print(compute_bleu(transformer, test_iter))
print(compute_bleu(transformer, test_iter, translate_method=translate_top_k,␣
 ↪params={"k": 5, "temperature": 1.0}))
```

```
print(compute_bleu(transformer, test_iter, translate_method=translate_top_k,␣
 ↪params={"k": 5, "temperature": 0.5}))
print(compute_bleu(transformer, test_iter, translate_method=translate_top_k,␣
 ↪params={"k": 5, "temperature": 0.1}))

print(compute_bleu(transformer, test_iter, translate_method=translate_top_p,␣
 ↪params={"p": 0.9, "temperature": 1.0}))
print(compute_bleu(transformer, test_iter, translate_method=translate_top_p,␣
 ↪params={"p": 0.9, "temperature": 0.5}))
print(compute_bleu(transformer, test_iter, translate_method=translate_top_p,␣
 ↪params={"p": 0.9, "temperature": 0.1}))

print(compute_bleu(transformer, test_iter, translate_method=translate_top_k,␣
 ↪params={"k": 10, "temperature": 1.0}))
print(compute_bleu(transformer, test_iter, translate_method=translate_top_k,␣
 ↪params={"k": 10, "temperature": 0.5}))
print(compute_bleu(transformer, test_iter, translate_method=translate_top_k,␣
 ↪params={"k": 10, "temperature": 0.1}))

print(compute_bleu(transformer, test_iter, translate_method=translate_top_p,␣
 ↪params={"p": 0.5, "temperature": 1.0}))
print(compute_bleu(transformer, test_iter, translate_method=translate_top_p,␣
 ↪params={"p": 0.5, "temperature": 0.5}))
print(compute_bleu(transformer, test_iter, translate_method=translate_top_p,␣
 ↪params={"p": 0.5, "temperature": 0.1}))

print(compute_bleu(transformer, test_iter, translate_method=translate_top_k,␣
 ↪params={"k": 50, "temperature": 1.0}))
print(compute_bleu(transformer, test_iter, translate_method=translate_top_k,␣
 ↪params={"k": 50, "temperature": 0.5}))
print(compute_bleu(transformer, test_iter, translate_method=translate_top_k,␣
 ↪params={"k": 50, "temperature": 0.1}))

print(compute_bleu(transformer, test_iter, translate_method=translate_top_k,␣
 ↪params={"k": 100, "temperature": 1.0}))
print(compute_bleu(transformer, test_iter, translate_method=translate_top_k,␣
 ↪params={"k": 100, "temperature": 0.5}))
print(compute_bleu(transformer, test_iter, translate_method=translate_top_k,␣
 ↪params={"k": 100, "temperature": 0.1}))
```

```
BLEU = 0.32 9.3/0.2/0.1/0.1 (BP = 1.000 ratio = 13.286 hyp_len = 279 ref_len =
21)
BLEU = 0.32 8.8/0.2/0.1/0.1 (BP = 1.000 ratio = 13.476 hyp_len = 283 ref_len =
21)
BLEU = 0.33 10.1/0.2/0.1/0.1 (BP = 1.000 ratio = 13.143 hyp_len = 276 ref_len =
21)
```

```
BLEU = 0.32 9.2/0.2/0.1/0.1 (BP = 1.000 ratio = 13.476 hyp_len = 283 ref_len =
21)
BLEU = 0.33 7.3/0.2/0.1/0.1 (BP = 1.000 ratio = 12.333 hyp_len = 259 ref_len =
21)
BLEU = 0.31 8.7/0.2/0.1/0.1 (BP = 1.000 ratio = 13.667 hyp_len = 287 ref_len =
21)
BLEU = 0.33 9.4/0.2/0.1/0.1 (BP = 1.000 ratio = 13.190 hyp_len = 277 ref_len =
21)
BLEU = 0.32 9.8/0.2/0.1/0.1 (BP = 1.000 ratio = 13.667 hyp_len = 287 ref_len =
21)
BLEU = 0.32 9.1/0.2/0.1/0.1 (BP = 1.000 ratio = 13.571 hyp_len = 285 ref_len =
21)
BLEU = 0.32 9.2/0.2/0.1/0.1 (BP = 1.000 ratio = 13.429 hyp_len = 282 ref_len =
21)
BLEU = 0.31 8.8/0.2/0.1/0.1 (BP = 1.000 ratio = 13.571 hyp_len = 285 ref_len =
21)
BLEU = 0.32 9.2/0.2/0.1/0.1 (BP = 1.000 ratio = 13.429 hyp_len = 282 ref_len =
21)
BLEU = 0.32 9.3/0.2/0.1/0.1 (BP = 1.000 ratio = 13.333 hyp_len = 280 ref_len =
21)
BLEU = 0.31 8.8/0.2/0.1/0.1 (BP = 1.000 ratio = 13.524 hyp_len = 284 ref_len =
21)
BLEU = 0.31 9.1/0.2/0.1/0.1 (BP = 1.000 ratio = 13.667 hyp_len = 287 ref_len =
21)
BLEU = 0.33 9.4/0.2/0.1/0.1 (BP = 1.000 ratio = 13.143 hyp_len = 276 ref_len =
21)
BLEU = 0.31 8.7/0.2/0.1/0.1 (BP = 1.000 ratio = 13.714 hyp_len = 288 ref_len =
21)
BLEU = 0.30 8.7/0.2/0.1/0.1 (BP = 1.000 ratio = 14.190 hyp_len = 298 ref_len =
21)
BLEU = 0.32 9.6/0.2/0.1/0.1 (BP = 1.000 ratio = 13.429 hyp_len = 282 ref_len =
21)
```

Let's take the first BLEU score computation to see what the different parameters mean (greedy decoder):

We obtain the following BLEU score: 0.32 9.3/0.2/0.1/0.1 (BP = 1.000 ratio = 13.286 hyp_len = 279 ref_len = 21) Here is a description of the different scores: * BLEU score: 0.32 * BLEU score for 1-grams: 9.3 * BLEU score for 2-grams: 0.2 * BLEU score for 3-grams: 0.1 * BLEU score for 4-grams: 0.1 * BP: 1.000 (Brevity Penalty) * ratio: 13.286 (ratio of the length of the candidate translation to the length of the reference translation) * hyp_len: 279 (total length of the candidate translation) * ref_len: 21 (total length of the reference translation)

Use part of the test set to perform an hyperparameters search on the value of temperature, k, and p. Note that, normally, this should be done on a validation set, not the test set.

[23]:
```
temperatures = [i / 10 for i in range(1, 11)]
ks = [5, 10, 50, 100]
```

```python
# use a small subset of the test set for hyperparameter search
test_iter = Multi30k(split='test', language_pair=(SRC_LANGUAGE, TGT_LANGUAGE))
test_iter = list(test_iter)[:100]

best_bleu_score = 0
best_temperature = 0
best_k = 0

for temperature in temperatures:
    for k in ks:
        bleu_score = compute_bleu(transformer, test_iter,␣
 ↪translate_method=translate_top_k, params={"k": k, "temperature":␣
 ↪temperature})
        bleu_score = float(str(bleu_score).split()[2])
        if bleu_score > best_bleu_score:
            best_bleu_score = bleu_score
            best_temperature = temperature
            best_k = k
        print(f"BLEU score: {bleu_score}, temperature: {temperature}, k: {k}")

print(f"Best BLEU score: {best_bleu_score}, temperature: {best_temperature}, k:␣
 ↪{best_k}")
```

```
BLEU score: 0.26, temperature: 0.1, k: 5
BLEU score: 0.26, temperature: 0.1, k: 10
BLEU score: 0.26, temperature: 0.1, k: 50
BLEU score: 0.26, temperature: 0.1, k: 100
BLEU score: 0.26, temperature: 0.2, k: 5
BLEU score: 0.26, temperature: 0.2, k: 10
BLEU score: 0.25, temperature: 0.2, k: 50
BLEU score: 0.26, temperature: 0.2, k: 100
BLEU score: 0.26, temperature: 0.3, k: 5
BLEU score: 0.26, temperature: 0.3, k: 10
BLEU score: 0.25, temperature: 0.3, k: 50
BLEU score: 0.26, temperature: 0.3, k: 100
BLEU score: 0.25, temperature: 0.4, k: 5
BLEU score: 0.25, temperature: 0.4, k: 10
BLEU score: 0.26, temperature: 0.4, k: 50
BLEU score: 0.26, temperature: 0.4, k: 100
BLEU score: 0.25, temperature: 0.5, k: 5
BLEU score: 0.25, temperature: 0.5, k: 10
BLEU score: 0.25, temperature: 0.5, k: 50
BLEU score: 0.25, temperature: 0.5, k: 100
BLEU score: 0.25, temperature: 0.6, k: 5
BLEU score: 0.26, temperature: 0.6, k: 10
BLEU score: 0.24, temperature: 0.6, k: 50
BLEU score: 0.26, temperature: 0.6, k: 100
```

```
BLEU score: 0.24, temperature: 0.7, k: 5
BLEU score: 0.25, temperature: 0.7, k: 10
BLEU score: 0.25, temperature: 0.7, k: 50
BLEU score: 0.25, temperature: 0.7, k: 100
BLEU score: 0.26, temperature: 0.8, k: 5
BLEU score: 0.26, temperature: 0.8, k: 10
BLEU score: 0.26, temperature: 0.8, k: 50
BLEU score: 0.25, temperature: 0.8, k: 100
BLEU score: 0.24, temperature: 0.9, k: 5
BLEU score: 0.26, temperature: 0.9, k: 10
BLEU score: 0.26, temperature: 0.9, k: 50
BLEU score: 0.25, temperature: 0.9, k: 100
BLEU score: 0.26, temperature: 1.0, k: 5
BLEU score: 0.25, temperature: 1.0, k: 10
BLEU score: 0.25, temperature: 1.0, k: 50
BLEU score: 0.25, temperature: 1.0, k: 100
Best BLEU score: 0.26, temperature: 0.1, k: 5
```

It looks like the best BLEU score (0.26) is obtained multiple times with different configurations, especially with temperature=0.1 and k=5

Let's do the same for p

[24]:
```
ps = [i / 100 for i in range(80, 100)]

best_bleu_score = 0
best_temperature = 0
best_p = 0

for temperature in temperatures:
    for p in ps:
        bleu_score = compute_bleu(transformer, test_iter,␣
 ↪translate_method=translate_top_p, params={"p": p, "temperature":␣
 ↪temperature})
        bleu_score = float(str(bleu_score).split()[2])
        if bleu_score > best_bleu_score:
            best_bleu_score = bleu_score
            best_temperature = temperature
            best_p = p
        print(f"BLEU score: {bleu_score}, temperature: {temperature}, p: {p}")

print(f"Best BLEU score: {best_bleu_score}, temperature: {best_temperature}, p:␣
 ↪{best_p}")
```

```
BLEU score: 0.26, temperature: 0.1, p: 0.8
BLEU score: 0.26, temperature: 0.1, p: 0.81
BLEU score: 0.26, temperature: 0.1, p: 0.82
BLEU score: 0.26, temperature: 0.1, p: 0.83
BLEU score: 0.26, temperature: 0.1, p: 0.84
```

BLEU score: 0.26, temperature: 0.1, p: 0.85
BLEU score: 0.26, temperature: 0.1, p: 0.86
BLEU score: 0.26, temperature: 0.1, p: 0.87
BLEU score: 0.26, temperature: 0.1, p: 0.88
BLEU score: 0.26, temperature: 0.1, p: 0.89
BLEU score: 0.26, temperature: 0.1, p: 0.9
BLEU score: 0.25, temperature: 0.1, p: 0.91
BLEU score: 0.26, temperature: 0.1, p: 0.92
BLEU score: 0.26, temperature: 0.1, p: 0.93
BLEU score: 0.25, temperature: 0.1, p: 0.94
BLEU score: 0.26, temperature: 0.1, p: 0.95
BLEU score: 0.25, temperature: 0.1, p: 0.96
BLEU score: 0.25, temperature: 0.1, p: 0.97
BLEU score: 0.26, temperature: 0.1, p: 0.98
BLEU score: 0.26, temperature: 0.1, p: 0.99
BLEU score: 0.26, temperature: 0.2, p: 0.8
BLEU score: 0.26, temperature: 0.2, p: 0.81
BLEU score: 0.26, temperature: 0.2, p: 0.82
BLEU score: 0.25, temperature: 0.2, p: 0.83
BLEU score: 0.26, temperature: 0.2, p: 0.84
BLEU score: 0.26, temperature: 0.2, p: 0.85
BLEU score: 0.26, temperature: 0.2, p: 0.86
BLEU score: 0.25, temperature: 0.2, p: 0.87
BLEU score: 0.25, temperature: 0.2, p: 0.88
BLEU score: 0.26, temperature: 0.2, p: 0.89
BLEU score: 0.25, temperature: 0.2, p: 0.9
BLEU score: 0.25, temperature: 0.2, p: 0.91
BLEU score: 0.26, temperature: 0.2, p: 0.92
BLEU score: 0.27, temperature: 0.2, p: 0.93
BLEU score: 0.25, temperature: 0.2, p: 0.94
BLEU score: 0.26, temperature: 0.2, p: 0.95
BLEU score: 0.26, temperature: 0.2, p: 0.96
BLEU score: 0.26, temperature: 0.2, p: 0.97
BLEU score: 0.25, temperature: 0.2, p: 0.98
BLEU score: 0.25, temperature: 0.2, p: 0.99
BLEU score: 0.26, temperature: 0.3, p: 0.8
BLEU score: 0.26, temperature: 0.3, p: 0.81
BLEU score: 0.25, temperature: 0.3, p: 0.82
BLEU score: 0.25, temperature: 0.3, p: 0.83
BLEU score: 0.25, temperature: 0.3, p: 0.84
BLEU score: 0.25, temperature: 0.3, p: 0.85
BLEU score: 0.26, temperature: 0.3, p: 0.86
BLEU score: 0.26, temperature: 0.3, p: 0.87
BLEU score: 0.26, temperature: 0.3, p: 0.88
BLEU score: 0.26, temperature: 0.3, p: 0.89
BLEU score: 0.26, temperature: 0.3, p: 0.9
BLEU score: 0.25, temperature: 0.3, p: 0.91
BLEU score: 0.26, temperature: 0.3, p: 0.92

BLEU score: 0.26, temperature: 0.3, p: 0.93
BLEU score: 0.26, temperature: 0.3, p: 0.94
BLEU score: 0.26, temperature: 0.3, p: 0.95
BLEU score: 0.25, temperature: 0.3, p: 0.96
BLEU score: 0.26, temperature: 0.3, p: 0.97
BLEU score: 0.25, temperature: 0.3, p: 0.98
BLEU score: 0.26, temperature: 0.3, p: 0.99
BLEU score: 0.26, temperature: 0.4, p: 0.8
BLEU score: 0.25, temperature: 0.4, p: 0.81
BLEU score: 0.25, temperature: 0.4, p: 0.82
BLEU score: 0.26, temperature: 0.4, p: 0.83
BLEU score: 0.26, temperature: 0.4, p: 0.84
BLEU score: 0.25, temperature: 0.4, p: 0.85
BLEU score: 0.26, temperature: 0.4, p: 0.86
BLEU score: 0.25, temperature: 0.4, p: 0.87
BLEU score: 0.25, temperature: 0.4, p: 0.88
BLEU score: 0.25, temperature: 0.4, p: 0.89
BLEU score: 0.26, temperature: 0.4, p: 0.9
BLEU score: 0.25, temperature: 0.4, p: 0.91
BLEU score: 0.25, temperature: 0.4, p: 0.92
BLEU score: 0.25, temperature: 0.4, p: 0.93
BLEU score: 0.26, temperature: 0.4, p: 0.94
BLEU score: 0.25, temperature: 0.4, p: 0.95
BLEU score: 0.26, temperature: 0.4, p: 0.96
BLEU score: 0.25, temperature: 0.4, p: 0.97
BLEU score: 0.26, temperature: 0.4, p: 0.98
BLEU score: 0.25, temperature: 0.4, p: 0.99
BLEU score: 0.26, temperature: 0.5, p: 0.8
BLEU score: 0.25, temperature: 0.5, p: 0.81
BLEU score: 0.25, temperature: 0.5, p: 0.82
BLEU score: 0.25, temperature: 0.5, p: 0.83
BLEU score: 0.25, temperature: 0.5, p: 0.84
BLEU score: 0.25, temperature: 0.5, p: 0.85
BLEU score: 0.25, temperature: 0.5, p: 0.86
BLEU score: 0.26, temperature: 0.5, p: 0.87
BLEU score: 0.25, temperature: 0.5, p: 0.88
BLEU score: 0.25, temperature: 0.5, p: 0.89
BLEU score: 0.26, temperature: 0.5, p: 0.9
BLEU score: 0.25, temperature: 0.5, p: 0.91
BLEU score: 0.25, temperature: 0.5, p: 0.92
BLEU score: 0.25, temperature: 0.5, p: 0.93
BLEU score: 0.26, temperature: 0.5, p: 0.94
BLEU score: 0.26, temperature: 0.5, p: 0.95
BLEU score: 0.26, temperature: 0.5, p: 0.96
BLEU score: 0.24, temperature: 0.5, p: 0.97
BLEU score: 0.26, temperature: 0.5, p: 0.98
BLEU score: 0.26, temperature: 0.5, p: 0.99
BLEU score: 0.26, temperature: 0.6, p: 0.8

BLEU score: 0.25, temperature: 0.6, p: 0.81
BLEU score: 0.26, temperature: 0.6, p: 0.82
BLEU score: 0.25, temperature: 0.6, p: 0.83
BLEU score: 0.25, temperature: 0.6, p: 0.84
BLEU score: 0.26, temperature: 0.6, p: 0.85
BLEU score: 0.25, temperature: 0.6, p: 0.86
BLEU score: 0.25, temperature: 0.6, p: 0.87
BLEU score: 0.25, temperature: 0.6, p: 0.88
BLEU score: 0.25, temperature: 0.6, p: 0.89
BLEU score: 0.26, temperature: 0.6, p: 0.9
BLEU score: 0.25, temperature: 0.6, p: 0.91
BLEU score: 0.26, temperature: 0.6, p: 0.92
BLEU score: 0.27, temperature: 0.6, p: 0.93
BLEU score: 0.26, temperature: 0.6, p: 0.94
BLEU score: 0.25, temperature: 0.6, p: 0.95
BLEU score: 0.26, temperature: 0.6, p: 0.96
BLEU score: 0.24, temperature: 0.6, p: 0.97
BLEU score: 0.26, temperature: 0.6, p: 0.98
BLEU score: 0.25, temperature: 0.6, p: 0.99
BLEU score: 0.26, temperature: 0.7, p: 0.8
BLEU score: 0.25, temperature: 0.7, p: 0.81
BLEU score: 0.26, temperature: 0.7, p: 0.82
BLEU score: 0.26, temperature: 0.7, p: 0.83
BLEU score: 0.26, temperature: 0.7, p: 0.84
BLEU score: 0.25, temperature: 0.7, p: 0.85
BLEU score: 0.26, temperature: 0.7, p: 0.86
BLEU score: 0.28, temperature: 0.7, p: 0.87
BLEU score: 0.26, temperature: 0.7, p: 0.88
BLEU score: 0.26, temperature: 0.7, p: 0.89
BLEU score: 0.26, temperature: 0.7, p: 0.9
BLEU score: 0.25, temperature: 0.7, p: 0.91
BLEU score: 0.26, temperature: 0.7, p: 0.92
BLEU score: 0.26, temperature: 0.7, p: 0.93
BLEU score: 0.25, temperature: 0.7, p: 0.94
BLEU score: 0.25, temperature: 0.7, p: 0.95
BLEU score: 0.25, temperature: 0.7, p: 0.96
BLEU score: 0.25, temperature: 0.7, p: 0.97
BLEU score: 0.25, temperature: 0.7, p: 0.98
BLEU score: 0.25, temperature: 0.7, p: 0.99
BLEU score: 0.26, temperature: 0.8, p: 0.8
BLEU score: 0.27, temperature: 0.8, p: 0.81
BLEU score: 0.26, temperature: 0.8, p: 0.82
BLEU score: 0.26, temperature: 0.8, p: 0.83
BLEU score: 0.26, temperature: 0.8, p: 0.84
BLEU score: 0.27, temperature: 0.8, p: 0.85
BLEU score: 0.25, temperature: 0.8, p: 0.86
BLEU score: 0.26, temperature: 0.8, p: 0.87
BLEU score: 0.26, temperature: 0.8, p: 0.88

BLEU score: 0.26, temperature: 0.8, p: 0.89
BLEU score: 0.26, temperature: 0.8, p: 0.9
BLEU score: 0.25, temperature: 0.8, p: 0.91
BLEU score: 0.25, temperature: 0.8, p: 0.92
BLEU score: 0.25, temperature: 0.8, p: 0.93
BLEU score: 0.25, temperature: 0.8, p: 0.94
BLEU score: 0.27, temperature: 0.8, p: 0.95
BLEU score: 0.26, temperature: 0.8, p: 0.96
BLEU score: 0.26, temperature: 0.8, p: 0.97
BLEU score: 0.24, temperature: 0.8, p: 0.98
BLEU score: 0.24, temperature: 0.8, p: 0.99
BLEU score: 0.27, temperature: 0.9, p: 0.8
BLEU score: 0.26, temperature: 0.9, p: 0.81
BLEU score: 0.26, temperature: 0.9, p: 0.82
BLEU score: 0.26, temperature: 0.9, p: 0.83
BLEU score: 0.26, temperature: 0.9, p: 0.84
BLEU score: 0.26, temperature: 0.9, p: 0.85
BLEU score: 0.25, temperature: 0.9, p: 0.86
BLEU score: 0.27, temperature: 0.9, p: 0.87
BLEU score: 0.26, temperature: 0.9, p: 0.88
BLEU score: 0.26, temperature: 0.9, p: 0.89
BLEU score: 0.26, temperature: 0.9, p: 0.9
BLEU score: 0.27, temperature: 0.9, p: 0.91
BLEU score: 0.27, temperature: 0.9, p: 0.92
BLEU score: 0.25, temperature: 0.9, p: 0.93
BLEU score: 0.26, temperature: 0.9, p: 0.94
BLEU score: 0.24, temperature: 0.9, p: 0.95
BLEU score: 0.25, temperature: 0.9, p: 0.96
BLEU score: 0.24, temperature: 0.9, p: 0.97
BLEU score: 0.25, temperature: 0.9, p: 0.98
BLEU score: 0.23, temperature: 0.9, p: 0.99
BLEU score: 0.27, temperature: 1.0, p: 0.8
BLEU score: 0.26, temperature: 1.0, p: 0.81
BLEU score: 0.25, temperature: 1.0, p: 0.82
BLEU score: 0.25, temperature: 1.0, p: 0.83
BLEU score: 0.26, temperature: 1.0, p: 0.84
BLEU score: 0.26, temperature: 1.0, p: 0.85
BLEU score: 0.25, temperature: 1.0, p: 0.86
BLEU score: 0.26, temperature: 1.0, p: 0.87
BLEU score: 0.24, temperature: 1.0, p: 0.88
BLEU score: 0.26, temperature: 1.0, p: 0.89
BLEU score: 0.24, temperature: 1.0, p: 0.9
BLEU score: 0.25, temperature: 1.0, p: 0.91
BLEU score: 0.26, temperature: 1.0, p: 0.92
BLEU score: 0.24, temperature: 1.0, p: 0.93
BLEU score: 0.24, temperature: 1.0, p: 0.94
BLEU score: 0.26, temperature: 1.0, p: 0.95
BLEU score: 0.24, temperature: 1.0, p: 0.96

```
BLEU score: 0.23, temperature: 1.0, p: 0.97
BLEU score: 0.22, temperature: 1.0, p: 0.98
BLEU score: 0.21, temperature: 1.0, p: 0.99
Best BLEU score: 0.28, temperature: 0.7, p: 0.87
```

For the top P decoding, the best BLEU score we have is 0.28 with a temperature of 0.7 and a p of 0.87.

The computed BLEU scores are not very good, even though the model seems to be trained enough.