Московский государственный технический университет им. Н.Э. Баумана
Факультет «Информатика и системы управления»
Кафедра «Системы обработки информации и управления»

Отчет
Лабораторная работа № 8
По курсу «Проектирование интеллектуальных систем»

Вариант 9

ИСПОЛНИТЕЛЬ:
Попов Илья Андреевич
Группа ИУ5-23М

____

"__"___________2022 г.

ПРЕПОДАВАТЕЛЬ:
Канев А.И.

____

"__"___________2022 г.

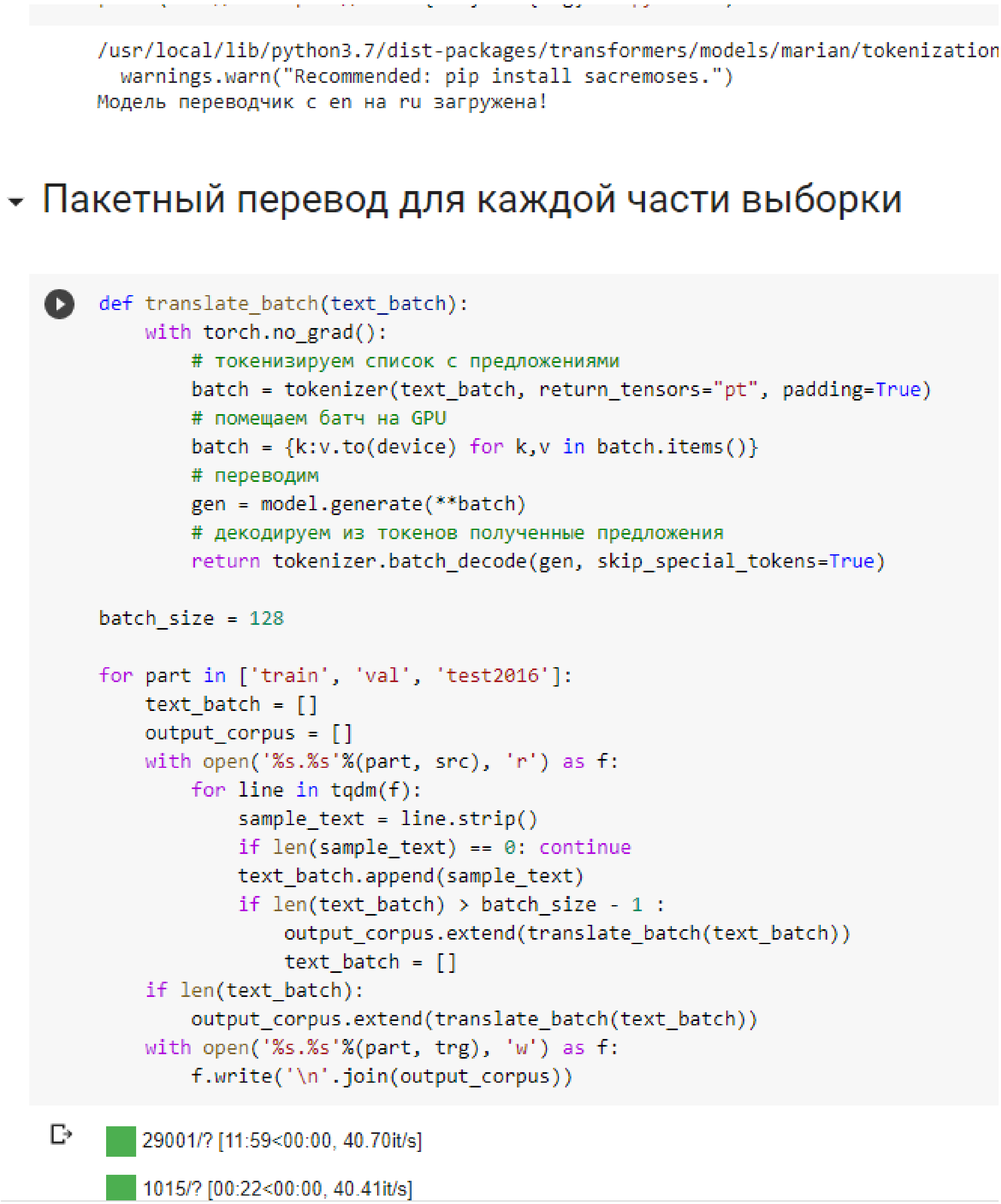
Москва 2022

Задание

Необходимо выполнить машинный перевод тектового корпуса для формирования набора данных. Обучить на сформированном наборе данных модель трасформер.

Выполнить предсказание с помощью обученной модели. Проанализировать метрику BLEU

Выполнение



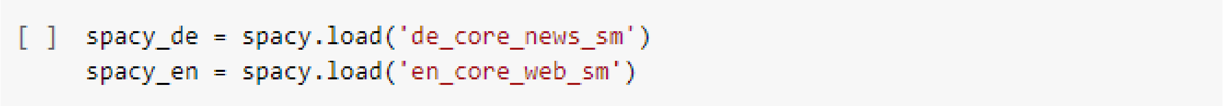
```
/usr/local/lib/python3.7/dist-packages/transformers/models/marian/tokenization
  warnings.warn("Recommended: pip install sacremoses.")
Модель переводчик с en на ru загружена!
```

Пакетный перевод для каждой части выборки

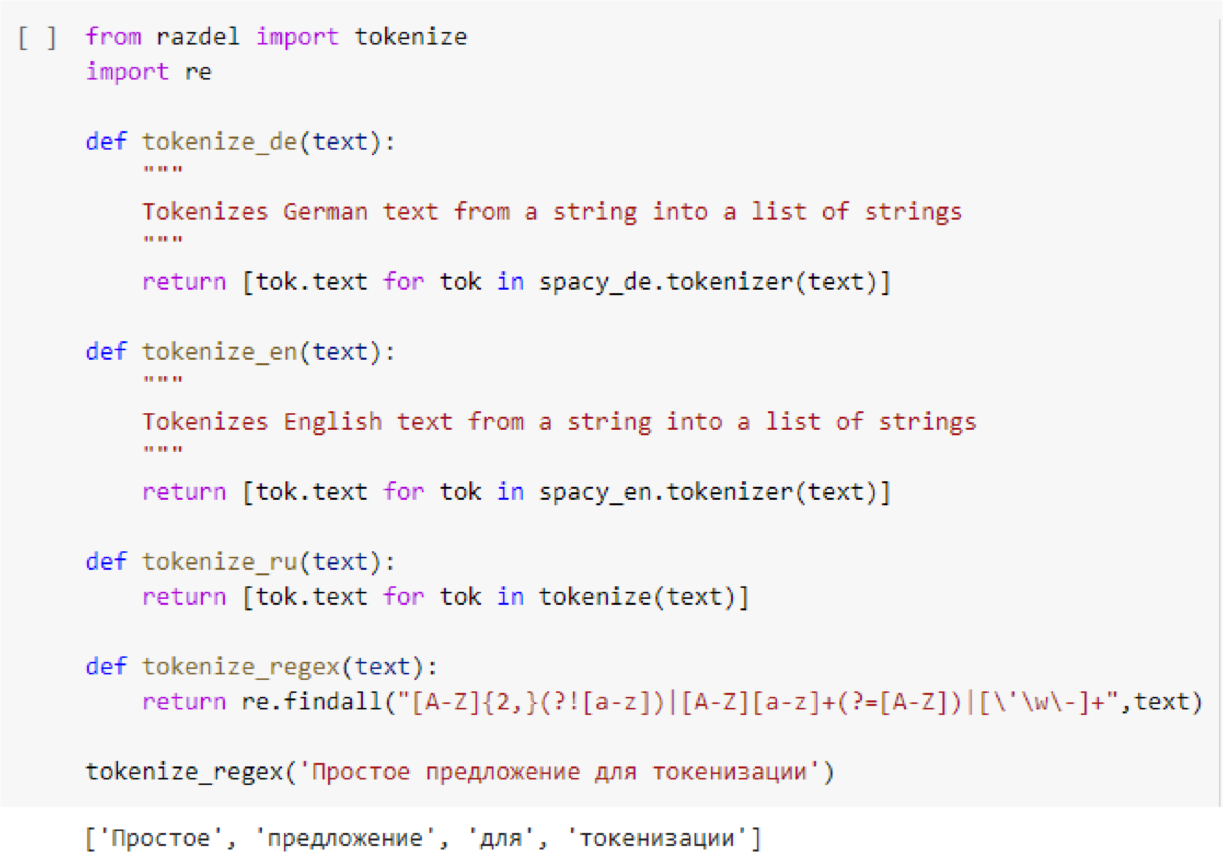
```
def translate_batch(text_batch):
    with torch.no_grad():
        # токенизируем список с предложениями
        batch = tokenizer(text_batch, return_tensors="pt", padding=True)
        # помещаем батч на GPU
        batch = {k:v.to(device) for k,v in batch.items()}
        # переводим
        gen = model.generate(**batch)
        # декодируем из токенов полученные предложения
        return tokenizer.batch_decode(gen, skip_special_tokens=True)

batch_size = 128

for part in ['train', 'val', 'test2016']:
    text_batch = []
    output_corpus = []
    with open('%s.%s'%(part, src), 'r') as f:
        for line in tqdm(f):
            sample_text = line.strip()
            if len(sample_text) == 0: continue
            text_batch.append(sample_text)
            if len(text_batch) > batch_size - 1 :
                output_corpus.extend(translate_batch(text_batch))
                text_batch = []
    if len(text_batch):
        output_corpus.extend(translate_batch(text_batch))
    with open('%s.%s'%(part, trg), 'w') as f:
        f.write('\n'.join(output_corpus))
```

```
🟩 29001/? [11:59<00:00, 40.70it/s]

🟩 1015/? [00:22<00:00, 40.41it/s]
```

Загружаем модель переводчика

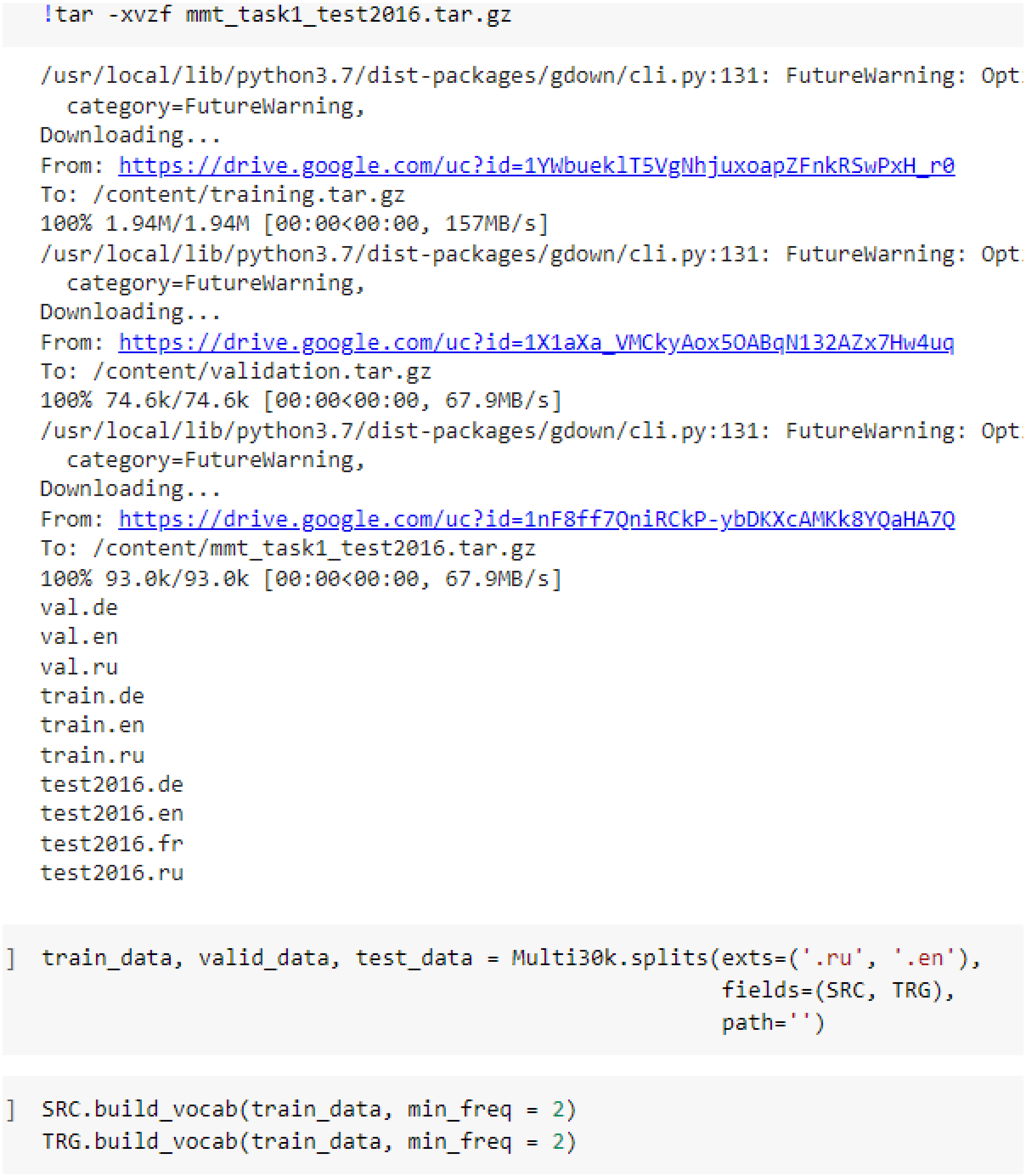


```
[ ]  spacy_de = spacy.load('de_core_news_sm')
     spacy_en = spacy.load('en_core_web_sm')
```



```
[ ]  from razdel import tokenize
     import re

     def tokenize_de(text):
         """
         Tokenizes German text from a string into a list of strings
         """
         return [tok.text for tok in spacy_de.tokenizer(text)]

     def tokenize_en(text):
         """
         Tokenizes English text from a string into a list of strings
         """
         return [tok.text for tok in spacy_en.tokenizer(text)]

     def tokenize_ru(text):
         return [tok.text for tok in tokenize(text)]

     def tokenize_regex(text):
         return re.findall("[A-Z]{2,}(?![a-z])|[A-Z][a-z]+(?=[A-Z])|[\'\w\-]+",text)

     tokenize_regex('Простое предложение для токенизации')

     ['Простое', 'предложение', 'для', 'токенизации']
```

Создаём токенизаторы



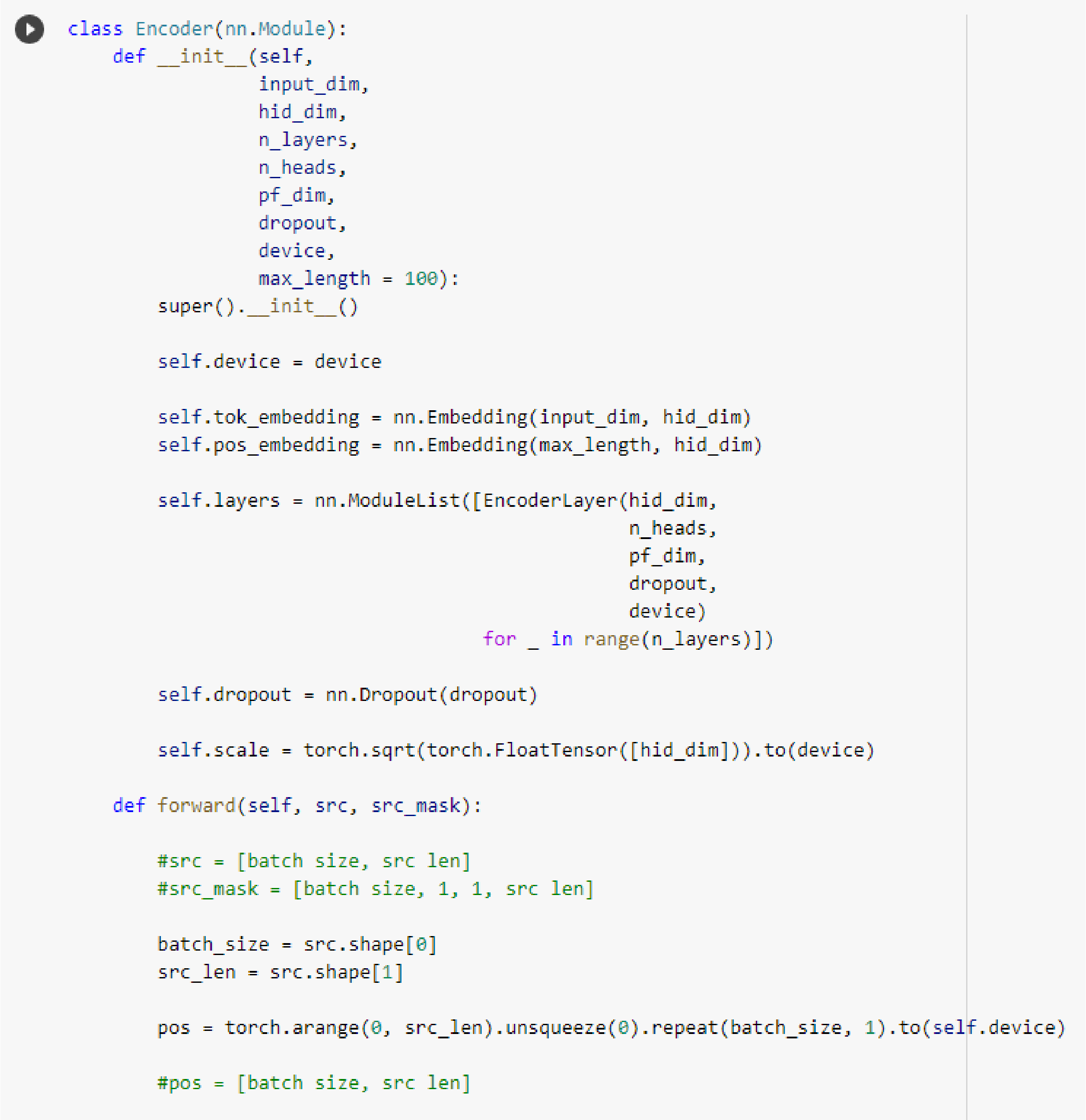
```
!tar -xvzf mmt_task1_test2016.tar.gz
```

```
/usr/local/lib/python3.7/dist-packages/gdown/cli.py:131: FutureWarning: Opt
  category=FutureWarning,
Downloading...
From: https://drive.google.com/uc?id=1YWbueklT5VgNhjuxoapZFnkRSwPxH_r0
To: /content/training.tar.gz
100% 1.94M/1.94M [00:00<00:00, 157MB/s]
/usr/local/lib/python3.7/dist-packages/gdown/cli.py:131: FutureWarning: Opt
  category=FutureWarning,
Downloading...
From: https://drive.google.com/uc?id=1X1aXa_VMCkyAox5OABqN132AZx7Hw4uq
To: /content/validation.tar.gz
100% 74.6k/74.6k [00:00<00:00, 67.9MB/s]
/usr/local/lib/python3.7/dist-packages/gdown/cli.py:131: FutureWarning: Opt
  category=FutureWarning,
Downloading...
From: https://drive.google.com/uc?id=1nF8ff7QniRCkP-ybDKXcAMKk8YQaHA7Q
To: /content/mmt_task1_test2016.tar.gz
100% 93.0k/93.0k [00:00<00:00, 67.9MB/s]
val.de
val.en
val.ru
train.de
train.en
train.ru
test2016.de
test2016.en
test2016.fr
test2016.ru
```

```
] train_data, valid_data, test_data = Multi30k.splits(exts=('.ru', '.en'),
                                                     fields=(SRC, TRG),
                                                     path='')
```

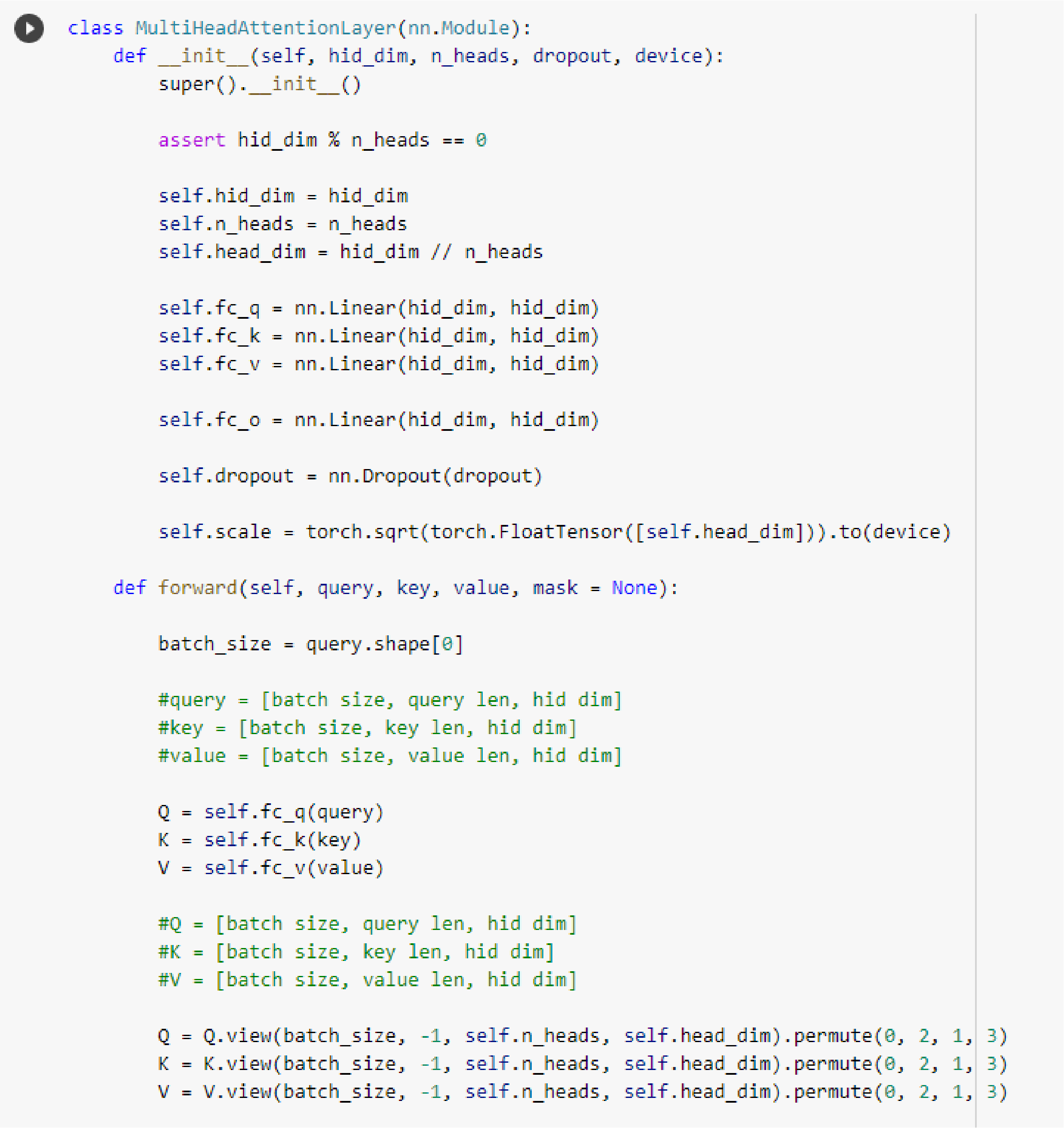
```
] SRC.build_vocab(train_data, min_freq = 2)
  TRG.build_vocab(train_data, min_freq = 2)
```

Загружаем текст



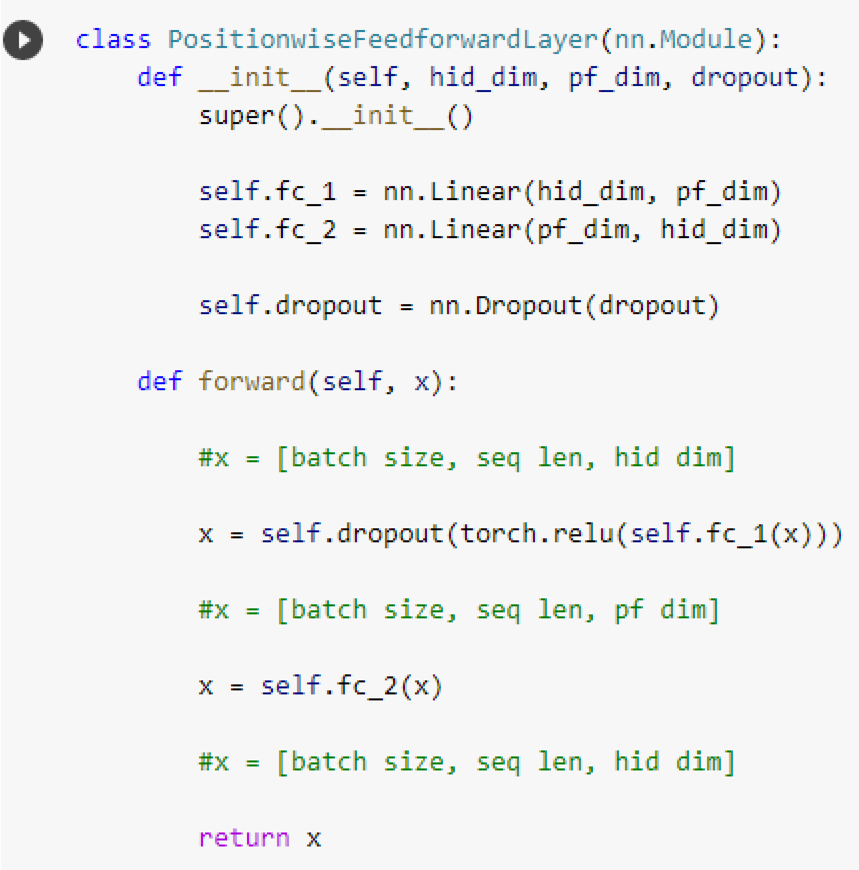
```
class Encoder(nn.Module):
    def __init__(self,
                 input_dim,
                 hid_dim,
                 n_layers,
                 n_heads,
                 pf_dim,
                 dropout,
                 device,
                 max_length = 100):
        super().__init__()

        self.device = device

        self.tok_embedding = nn.Embedding(input_dim, hid_dim)
        self.pos_embedding = nn.Embedding(max_length, hid_dim)

        self.layers = nn.ModuleList([EncoderLayer(hid_dim,
                                                  n_heads,
                                                  pf_dim,
                                                  dropout,
                                                  device)
                                     for _ in range(n_layers)])

        self.dropout = nn.Dropout(dropout)

        self.scale = torch.sqrt(torch.FloatTensor([hid_dim])).to(device)

    def forward(self, src, src_mask):

        #src = [batch size, src len]
        #src_mask = [batch size, 1, 1, src len]

        batch_size = src.shape[0]
        src_len = src.shape[1]

        pos = torch.arange(0, src_len).unsqueeze(0).repeat(batch_size, 1).to(self.device)

        #pos = [batch size, src len]
```

Определяем энкодер трансформера



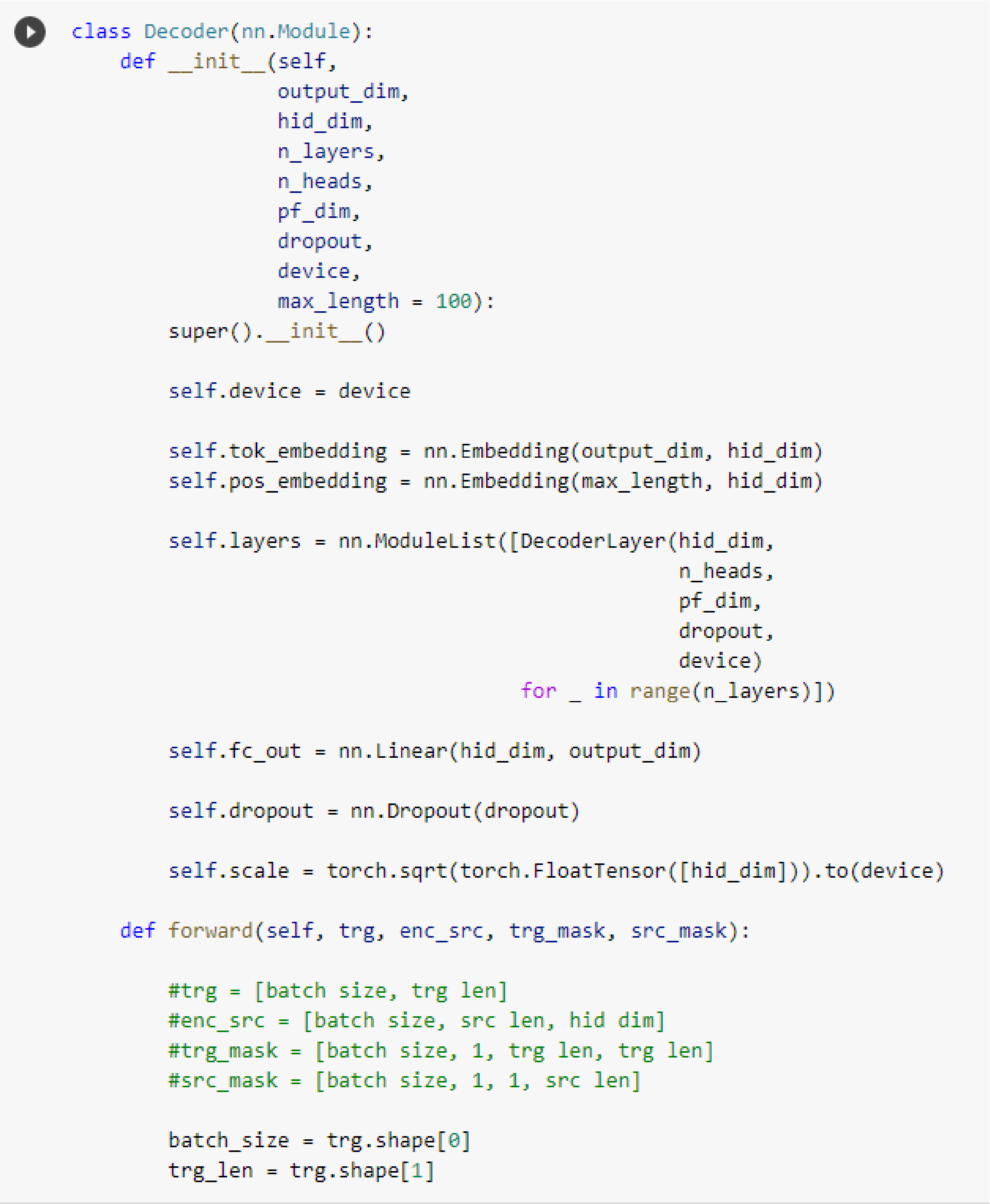
```python
class MultiHeadAttentionLayer(nn.Module):
    def __init__(self, hid_dim, n_heads, dropout, device):
        super().__init__()

        assert hid_dim % n_heads == 0

        self.hid_dim = hid_dim
        self.n_heads = n_heads
        self.head_dim = hid_dim // n_heads

        self.fc_q = nn.Linear(hid_dim, hid_dim)
        self.fc_k = nn.Linear(hid_dim, hid_dim)
        self.fc_v = nn.Linear(hid_dim, hid_dim)

        self.fc_o = nn.Linear(hid_dim, hid_dim)

        self.dropout = nn.Dropout(dropout)

        self.scale = torch.sqrt(torch.FloatTensor([self.head_dim])).to(device)

    def forward(self, query, key, value, mask = None):

        batch_size = query.shape[0]

        #query = [batch size, query len, hid dim]
        #key = [batch size, key len, hid dim]
        #value = [batch size, value len, hid dim]

        Q = self.fc_q(query)
        K = self.fc_k(key)
        V = self.fc_v(value)

        #Q = [batch size, query len, hid dim]
        #K = [batch size, key len, hid dim]
        #V = [batch size, value len, hid dim]

        Q = Q.view(batch_size, -1, self.n_heads, self.head_dim).permute(0, 2, 1, 3)
        K = K.view(batch_size, -1, self.n_heads, self.head_dim).permute(0, 2, 1, 3)
        V = V.view(batch_size, -1, self.n_heads, self.head_dim).permute(0, 2, 1, 3)
```

Голова внимания



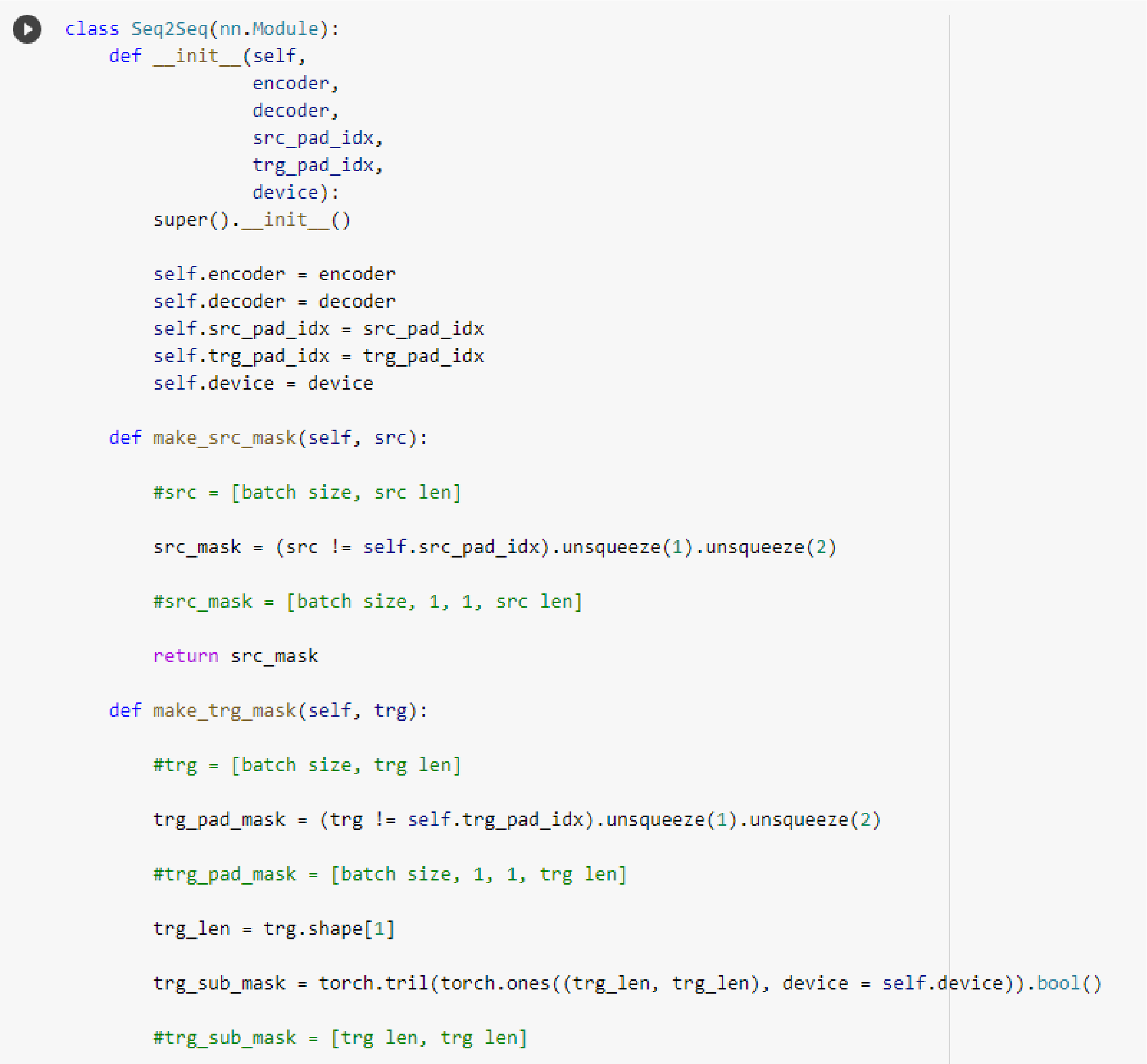
```
class PositionwiseFeedforwardLayer(nn.Module):
    def __init__(self, hid_dim, pf_dim, dropout):
        super().__init__()

        self.fc_1 = nn.Linear(hid_dim, pf_dim)
        self.fc_2 = nn.Linear(pf_dim, hid_dim)

        self.dropout = nn.Dropout(dropout)

    def forward(self, x):

        #x = [batch size, seq len, hid dim]

        x = self.dropout(torch.relu(self.fc_1(x)))

        #x = [batch size, seq len, pf dim]

        x = self.fc_2(x)

        #x = [batch size, seq len, hid dim]

        return x
```

Полносвязный слой



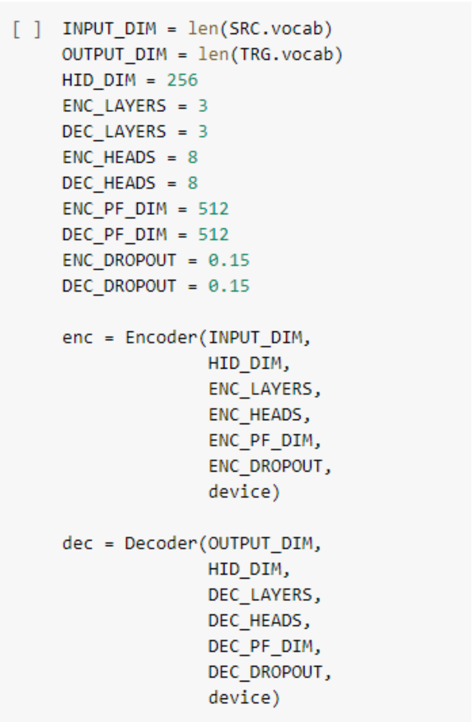
```
class Decoder(nn.Module):
    def __init__(self,
                 output_dim,
                 hid_dim,
                 n_layers,
                 n_heads,
                 pf_dim,
                 dropout,
                 device,
                 max_length = 100):
        super().__init__()

        self.device = device

        self.tok_embedding = nn.Embedding(output_dim, hid_dim)
        self.pos_embedding = nn.Embedding(max_length, hid_dim)

        self.layers = nn.ModuleList([DecoderLayer(hid_dim,
                                                  n_heads,
                                                  pf_dim,
                                                  dropout,
                                                  device)
                                     for _ in range(n_layers)])

        self.fc_out = nn.Linear(hid_dim, output_dim)

        self.dropout = nn.Dropout(dropout)

        self.scale = torch.sqrt(torch.FloatTensor([hid_dim])).to(device)

    def forward(self, trg, enc_src, trg_mask, src_mask):

        #trg = [batch size, trg len]
        #enc_src = [batch size, src len, hid dim]
        #trg_mask = [batch size, 1, trg len, trg len]
        #src_mask = [batch size, 1, 1, src len]

        batch_size = trg.shape[0]
        trg_len = trg.shape[1]
```

Слой Декодера

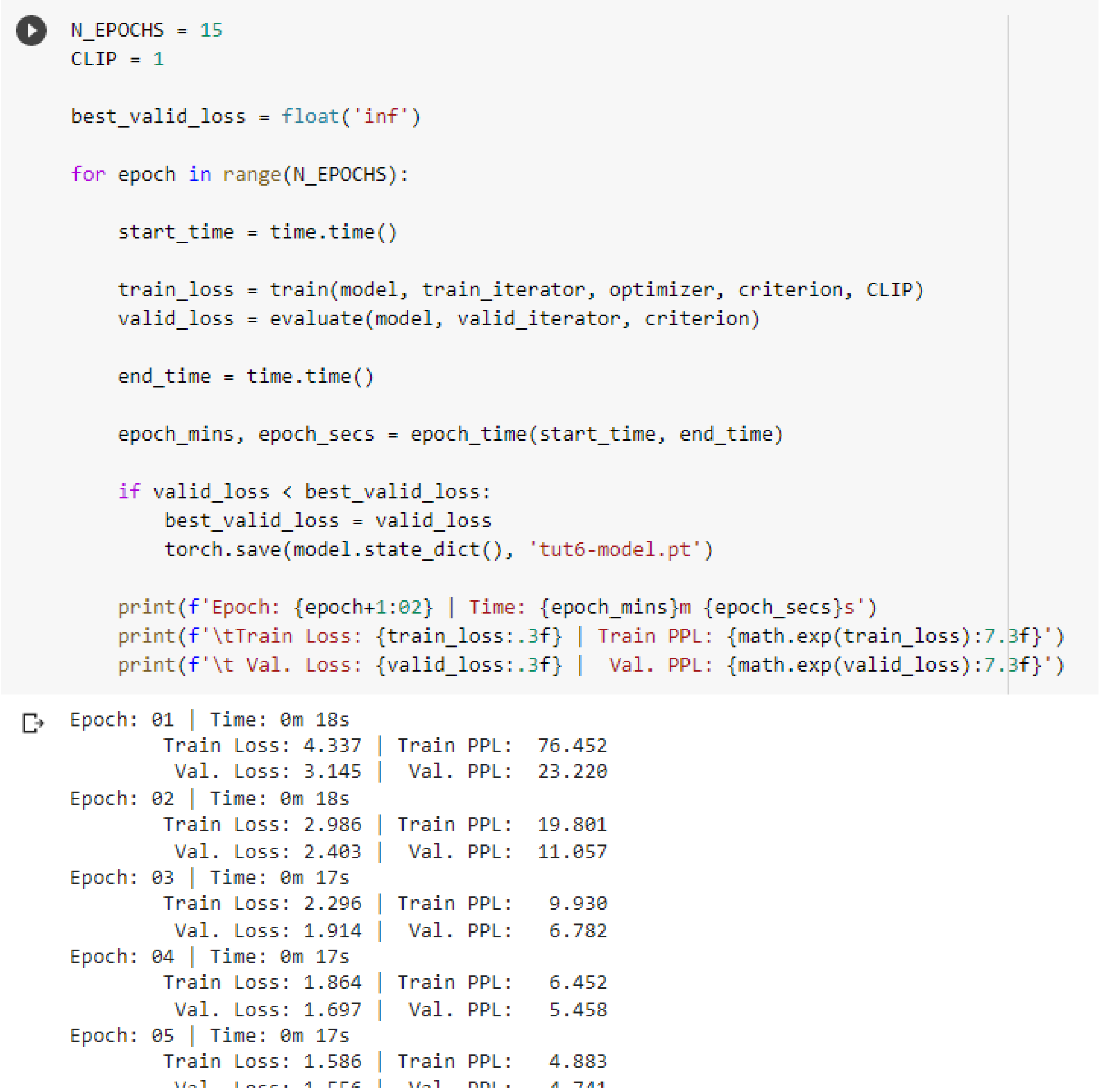


```
class Seq2Seq(nn.Module):
    def __init__(self,
                 encoder,
                 decoder,
                 src_pad_idx,
                 trg_pad_idx,
                 device):
        super().__init__()

        self.encoder = encoder
        self.decoder = decoder
        self.src_pad_idx = src_pad_idx
        self.trg_pad_idx = trg_pad_idx
        self.device = device

    def make_src_mask(self, src):

        #src = [batch size, src len]

        src_mask = (src != self.src_pad_idx).unsqueeze(1).unsqueeze(2)

        #src_mask = [batch size, 1, 1, src len]

        return src_mask

    def make_trg_mask(self, trg):

        #trg = [batch size, trg len]

        trg_pad_mask = (trg != self.trg_pad_idx).unsqueeze(1).unsqueeze(2)

        #trg_pad_mask = [batch size, 1, 1, trg len]

        trg_len = trg.shape[1]

        trg_sub_mask = torch.tril(torch.ones((trg_len, trg_len), device = self.device)).bool()

        #trg_sub_mask = [trg len, trg len]
```

Формируем трансформер с помощью модели Seq2Seq



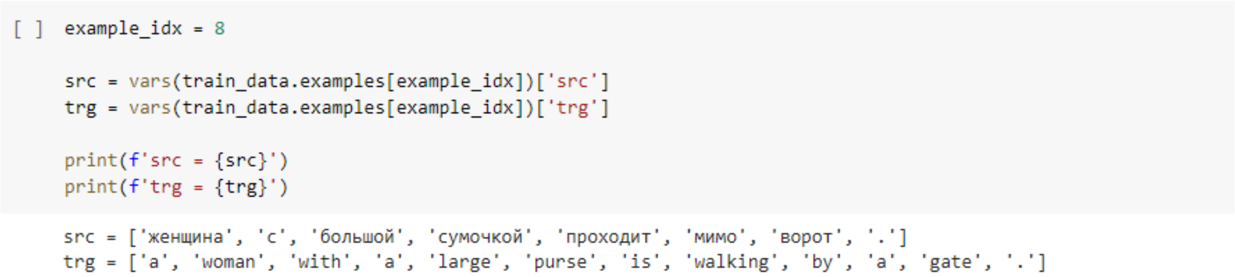
```
[ ] INPUT_DIM = len(SRC.vocab)
    OUTPUT_DIM = len(TRG.vocab)
    HID_DIM = 256
    ENC_LAYERS = 3
    DEC_LAYERS = 3
    ENC_HEADS = 8
    DEC_HEADS = 8
    ENC_PF_DIM = 512
    DEC_PF_DIM = 512
    ENC_DROPOUT = 0.15
    DEC_DROPOUT = 0.15

    enc = Encoder(INPUT_DIM,
                  HID_DIM,
                  ENC_LAYERS,
                  ENC_HEADS,
                  ENC_PF_DIM,
                  ENC_DROPOUT,
                  device)

    dec = Decoder(OUTPUT_DIM,
                  HID_DIM,
                  DEC_LAYERS,
                  DEC_HEADS,
                  DEC_PF_DIM,
                  DEC_DROPOUT,
                  device)
```



```
N_EPOCHS = 15
CLIP = 1

best_valid_loss = float('inf')

for epoch in range(N_EPOCHS):

    start_time = time.time()

    train_loss = train(model, train_iterator, optimizer, criterion, CLIP)
    valid_loss = evaluate(model, valid_iterator, criterion)

    end_time = time.time()

    epoch_mins, epoch_secs = epoch_time(start_time, end_time)

    if valid_loss < best_valid_loss:
        best_valid_loss = valid_loss
        torch.save(model.state_dict(), 'tut6-model.pt')

    print(f'Epoch: {epoch+1:02} | Time: {epoch_mins}m {epoch_secs}s')
    print(f'\tTrain Loss: {train_loss:.3f} | Train PPL: {math.exp(train_loss):7.3f}')
    print(f'\t Val. Loss: {valid_loss:.3f} |  Val. PPL: {math.exp(valid_loss):7.3f}')
```

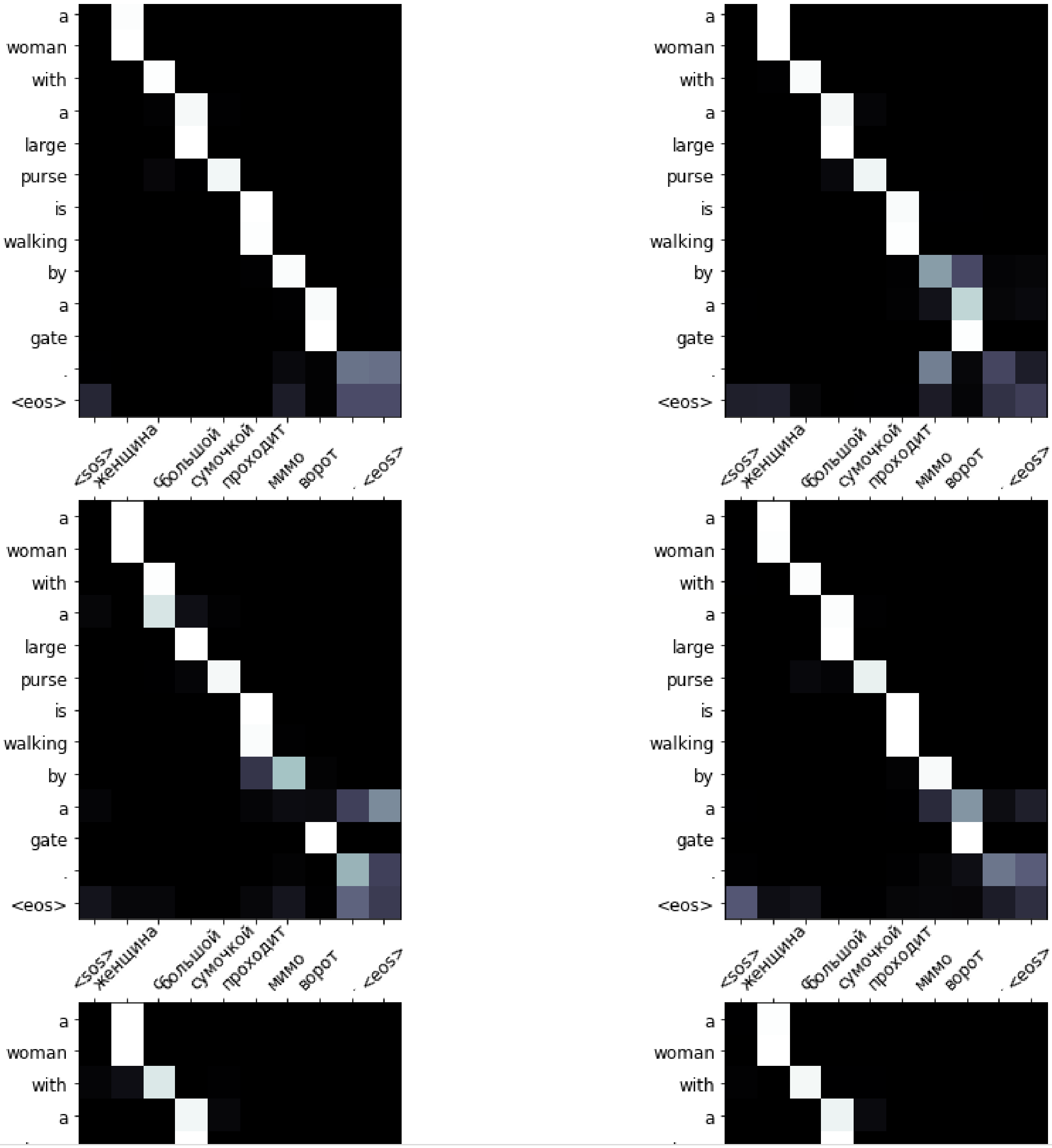
```
Epoch: 01 | Time: 0m 18s
        Train Loss: 4.337 | Train PPL:  76.452
         Val. Loss: 3.145 |  Val. PPL:  23.220
Epoch: 02 | Time: 0m 18s
        Train Loss: 2.986 | Train PPL:  19.801
         Val. Loss: 2.403 |  Val. PPL:  11.057
Epoch: 03 | Time: 0m 17s
        Train Loss: 2.296 | Train PPL:   9.930
         Val. Loss: 1.914 |  Val. PPL:   6.782
Epoch: 04 | Time: 0m 17s
        Train Loss: 1.864 | Train PPL:   6.452
         Val. Loss: 1.697 |  Val. PPL:   5.458
Epoch: 05 | Time: 0m 17s
        Train Loss: 1.586 | Train PPL:   4.883
         Val. Loss: 1.556 |  Val. PPL:   4.741
```

Обучение модели

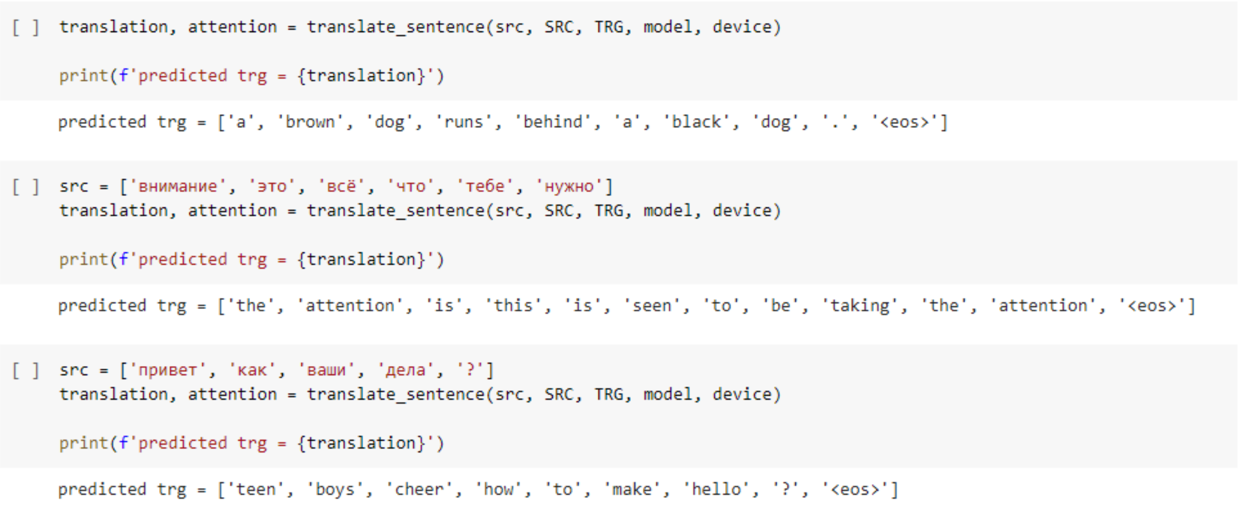


```
[ ]  example_idx = 8

     src = vars(train_data.examples[example_idx])['src']
     trg = vars(train_data.examples[example_idx])['trg']

     print(f'src = {src}')
     print(f'trg = {trg}')

     src = ['женщина', 'с', 'большой', 'сумочкой', 'проходит', 'мимо', 'ворот', '.']
     trg = ['a', 'woman', 'with', 'a', 'large', 'purse', 'is', 'walking', 'by', 'a', 'gate', '.']
```

Результат: пример из обучающего набора



Сосредоточение внимания каждой из голов

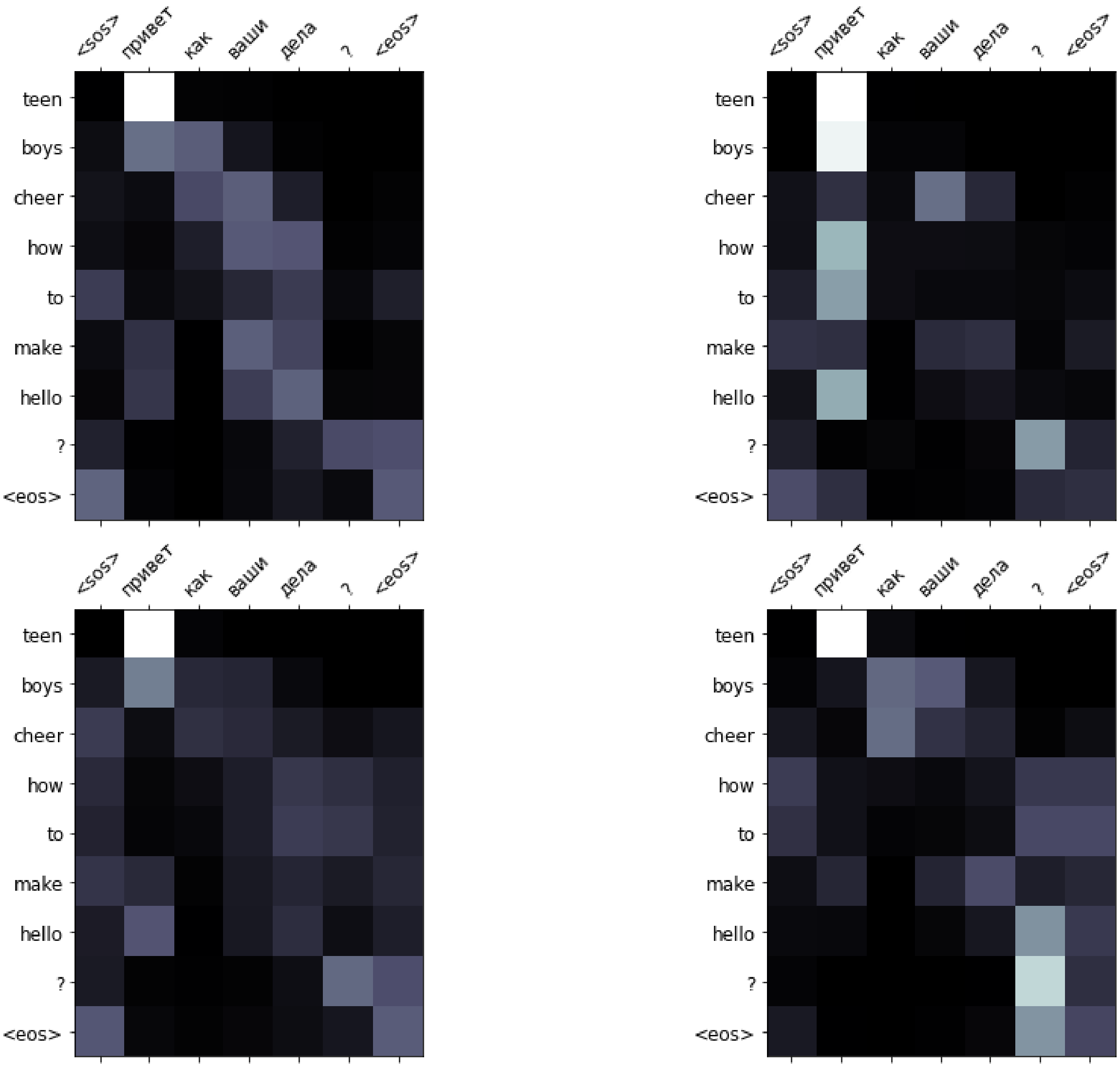


```
[ ]  translation, attention = translate_sentence(src, SRC, TRG, model, device)

     print(f'predicted trg = {translation}')

     predicted trg = ['a', 'brown', 'dog', 'runs', 'behind', 'a', 'black', 'dog', '.', '<eos>']
```

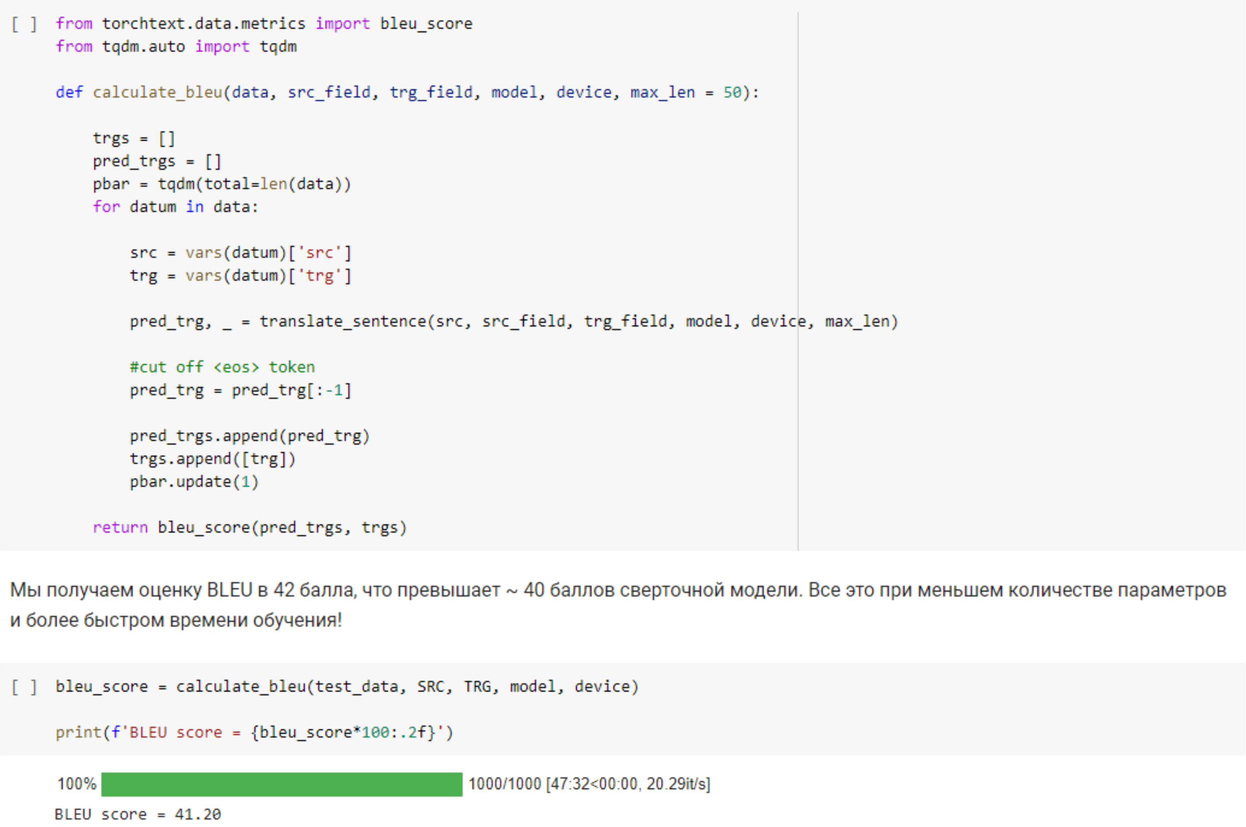
```
[ ]  src = ['внимание', 'это', 'всё', 'что', 'тебе', 'нужно']
     translation, attention = translate_sentence(src, SRC, TRG, model, device)

     print(f'predicted trg = {translation}')

     predicted trg = ['the', 'attention', 'is', 'this', 'is', 'seen', 'to', 'be', 'taking', 'the', 'attention', '<eos>']
```

```
[ ]  src = ['привет', 'как', 'ваши', 'дела', '?']
     translation, attention = translate_sentence(src, SRC, TRG, model, device)

     print(f'predicted trg = {translation}')

     predicted trg = ['teen', 'boys', 'cheer', 'how', 'to', 'make', 'hello', '?', '<eos>']
```

Несколько собственных примеров перевода



Сосредоточение внимания



```
[ ] from torchtext.data.metrics import bleu_score
    from tqdm.auto import tqdm

    def calculate_bleu(data, src_field, trg_field, model, device, max_len = 50):

        trgs = []
        pred_trgs = []
        pbar = tqdm(total=len(data))
        for datum in data:

            src = vars(datum)['src']
            trg = vars(datum)['trg']

            pred_trg, _ = translate_sentence(src, src_field, trg_field, model, device, max_len)

            #cut off <eos> token
            pred_trg = pred_trg[:-1]

            pred_trgs.append(pred_trg)
            trgs.append([trg])
            pbar.update(1)

        return bleu_score(pred_trgs, trgs)
```

Мы получаем оценку BLEU в 42 балла, что превышает ~ 40 баллов сверточной модели. Все это при меньшем количестве параметров и более быстром времени обучения!

```
[ ] bleu_score = calculate_bleu(test_data, SRC, TRG, model, device)

    print(f'BLEU score = {bleu_score*100:.2f}')
```

```
100% 1000/1000 [47:32<00:00, 20.29it/s]
BLEU score = 41.20
```

Расчёт оценки BELU