```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.nn.utils.rnn import pad_sequence
from torch.utils.data import TensorDataset, DataLoader, RandomSampler, SequentialSampler
from torchtext import datasets

import numpy as np
from sklearn.metrics import classification_report

import random

from gensim.models import FastText
from nltk import word_tokenize
from nltk.stem import PorterStemmer
from sklearn.metrics import accuracy_score, f1_score
```

```
SEED = 1234

random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
torch.backends.cudnn.deterministic = True
```

```
train_data, _, test_data = datasets.UDPOS()
train_data = [d for d in train_data]
test_data = [d for d in test_data]

train_tokens = [ [w.lower() for w in d[0]] for d in train_data]
train_tags = [ d[1] for d in train_data]

test_tokens = [[w.lower() for w in d[0]] for d in test_data]
test_tags = [d[1] for d in test_data]

tag2num = { t:i for i, t in enumerate(np.unique([tag for tags in train_tags for tag in tag
```

```
    100%|██████████| 688k/688k [00:00<00:00, 16.4MB/s]
```

```
print(train_data[0][0])
print(train_data[0][1])

print(train_tokens[0])
print(train_tags[0])
```

```
    ['Al', '-', 'Zaman', ':', 'American', 'forces', 'killed', 'Shaikh', 'Abdullah', 'al',
    ['PROPN', 'PUNCT', 'PROPN', 'PUNCT', 'ADJ', 'NOUN', 'VERB', 'PROPN', 'PROPN', 'PROPN
    ['al', '-', 'zaman', ':', 'american', 'forces', 'killed', 'shaikh', 'abdullah', 'al',
    ['PROPN', 'PUNCT', 'PROPN', 'PUNCT', 'ADJ', 'NOUN', 'VERB', 'PROPN', 'PROPN', 'PROPN
```

```
ft = FastText(sentences=train_tokens, size=100, window=5, min_count=1, min_n=1, workers=4)
```

```
max_len = 20
pad_inds = len(tag2num)


def prepare_data(all_tokens, all_tags, ft, tag2num, max_len, pad_tags):
  '''
    Из массива слов all_tokens получим тензор векторов, где каждое слово представлено вект
    А целевую переменную классов all_tags преобразуем в числа.
    Все строки не длиннее max_len.
    Пустые значения заполняются нулями или pad_tags
  '''

    # укорачиваем токены
  all_tokens = [tokens[:max_len] for tokens in all_tokens]
  all_tags = [tags[:max_len] for tags in all_tags]

  # переводим теги в числа
  all_tags = [np.array([tag2num[tag]  for tag in tags]) for tags in all_tags]

  # all_ids = []
  # for tokens in all_tokens:
  #     ids = prepare_sequence(tokens, word_to_ix)
  #     all_ids.append(ids)

  X_vecs = []
  Y_vecs = []

  for tokens, tags in zip(all_tokens, all_tags):
      X_vecs.append(torch.tensor(np.row_stack([ft.wv[w] for w in tokens])))
      Y_vecs.append(torch.tensor(tags, dtype=torch.long))

  # в качестве заполнителя X используем новый индекс len(word_to_ix)
  X = pad_sequence(X_vecs, batch_first=True)

  # в качестве заполнителя Y используем pad_tags
  Y = pad_sequence(Y_vecs, batch_first=True, padding_value=pad_tags)

  return X, Y

X_train, Y_train = prepare_data(train_tokens, train_tags, ft, tag2num, max_len, pad_inds)

# X_train.size(), Y_train.size()

X_test, Y_test = prepare_data(test_tokens, test_tags, ft, tag2num, max_len, pad_inds)

# X_test.size(), Y_test.size()
```

```
X_train.size(), Y_train.size()
```

```
(torch.Size([12543, 20, 100]), torch.Size([12543, 20]))
```

```
X_test.size(), Y_test.size()
```

```
(torch.Size([2077, 20, 100]), torch.Size([2077, 20]))
```

```
from torch.utils.data import TensorDataset, DataLoader, RandomSampler, SequentialSampler

bs = 128
data = TensorDataset(X_train, Y_train)
dataloader = DataLoader(data, sampler=SequentialSampler(data), batch_size=bs)
```

```
class BiLSTMPOSTagger(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim, n_layers, bidirectional, dropout

        super().__init__()

        self.output_dim = output_dim
        self.input_size = input_dim

        self.lstm = nn.LSTM(input_dim, hidden_dim, num_layers=n_layers, bidirectional=bidi

        self.fc = nn.Linear(hidden_dim * 2 if bidirectional else hidden_dim, output_dim)

        self.dropout = nn.Dropout(dropout)

    def forward(self, sentence):
    # sentence = [batch size, sent len, emb dim]
      sentence = sentence.view(sentence.shape[1], sentence.shape[0], self.input_size)
    # sentence = [sent len, batch size, emb dim]

      outputs, (hidden, cell) = self.lstm(sentence)

      predictions = self.fc(self.dropout(outputs))

      # predictions = [sent len, batch size, output dim]
      predictions = predictions.view(predictions.shape[1],predictions.shape[0], self.outpu
      # predictions = [batch size, sent len, output dim]

      # raise NotImplementedException()
      return predictions
```

```
def train_on_epoch(model, dataloader, optimizer):
    model.train()
    for batch in dataloader:
        batch = tuple(t.to(device) for t in batch)
        b_input, b_tags = batch

        model.zero_grad()
        outputs = model(b_input)

        # outputs = [batch size, sent len, out dim]
        outputs = outputs.view(-1, outputs.shape[-1])
        # outputs = [batch size * sent len, out dim]

        # b_tags = [batch size, sent len]
        b_tags = b_tags.view(-1)
        # b_tags = [batch size * sent len]

        loss = criterion(outputs, b_tags)
        loss.backward()
        optimizer.step()


def predict_on_dataloader(model, dataloaded):
    model.eval()

    all_outputs = []
    all_tags = []
    for batch in dataloaded:
        batch = tuple(t.to(device) for t in batch)
        b_input, b_tags = batch
        outputs = model(b_input)

        outputs = outputs.view(-1, outputs.shape[-1])
        b_tags = b_tags.view(-1)

        all_outputs.append(outputs)
        all_tags.append(b_tags)

    all_outputs = torch.cat(all_outputs)
    all_tags = torch.cat(all_tags)

    return all_outputs, all_tags
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(device)
```

```
cuda
```

```
INPUT_DIM = 100
HIDDEN_DIM = 128
OUTPUT_DIM = len(tag2num)
N_LAYERS = 2
BIDIRECTIONAL = True
DROPOUT = 0.25

model = BiLSTMPOSTagger(INPUT_DIM, HIDDEN_DIM, OUTPUT_DIM, N_LAYERS, BIDIRECTIONAL, DROPOU
model.to(device)

criterion = nn.CrossEntropyLoss(ignore_index=pad_inds)
optimizer = optim.Adam(model.parameters())
```

```
epochs = 50
for e in range(epochs):
    train_on_epoch(model, dataloader, optimizer)

    all_outputs, all_tags = predict_on_dataloader(model, dataloader)
    loss = criterion(all_outputs, all_tags).item()
    all_outputs = all_outputs.detach().cpu().numpy()
    all_tags = all_tags.detach().cpu().numpy()

    mask = all_tags != pad_inds
    loss = loss/len(all_tags[mask])
    all_tags = all_tags[mask]
    all_preds = np.argmax(all_outputs, axis=1)[mask]

    print(f"{e}:\tLoss {loss}, "
          f"accuracy: {accuracy_score(all_tags, all_preds)}, "
          f"f1-macro: {f1_score(all_tags, all_preds, average='macro')}")
```

```
0:      Loss 1.0015787127812761e-05, accuracy: 0.48026716360080446, f1-macro: 0.25125
1:      Loss 8.605620787400416e-06, accuracy: 0.5540446625706501, f1-macro: 0.3588759
2:      Loss 7.738153386824405e-06, accuracy: 0.5951462819117204, f1-macro: 0.4304601
3:      Loss 7.077609564454969e-06, accuracy: 0.6303437332792118, f1-macro: 0.4783707
4:      Loss 6.623119794917055e-06, accuracy: 0.6454670135365351, f1-macro: 0.4937901
5:      Loss 6.311222655833707e-06, accuracy: 0.6566787825113625, f1-macro: 0.5072632
6:      Loss 6.073827198705909e-06, accuracy: 0.6675399915127586, f1-macro: 0.5193816
7:      Loss 5.869439980562564e-06, accuracy: 0.677866135291549, f1-macro: 0.52703248
8:      Loss 5.71169847315859e-06, accuracy: 0.6825894696704121, f1-macro: 0.52991742
9:      Loss 5.58071466630315e-06, accuracy: 0.686242673604064, f1-macro: 0.533624975
10:     Loss 5.464418580021986e-06, accuracy: 0.6914580219807254, f1-macro: 0.5401216
11:     Loss 5.332528956471036e-06, accuracy: 0.6975466952034786, f1-macro: 0.5567457
12:     Loss 5.226545837043296e-06, accuracy: 0.7033832112523601, f1-macro: 0.5792283
13:     Loss 5.129611353516807e-06, accuracy: 0.7082541498305627, f1-macro: 0.5892182
14:     Loss 5.044750515754521e-06, accuracy: 0.7131312385837377, f1-macro: 0.5985769
15:     Loss 4.964695895189664e-06, accuracy: 0.715861916271518, f1-macro: 0.60115755
16:     Loss 4.8954472902258876e-06, accuracy: 0.7185064915096835, f1-macro: 0.606478
17:     Loss 4.810539163773307e-06, accuracy: 0.7227070610158859, f1-macro: 0.6159342
18:     Loss 4.74201711837671e-06, accuracy: 0.7255299913282532, f1-macro: 0.62670118
19:     Loss 4.681432974527906e-06, accuracy: 0.7288264851135015, f1-macro: 0.6392520
20:     Loss 4.626134166622395e-06, accuracy: 0.7313234561523275, f1-macro: 0.6510890
21:     Loss 4.56079439347741e-06, accuracy: 0.735450223558602, f1-macro: 0.66328558
22:     Loss 4.5238402981957155e-06, accuracy: 0.7357392817825668, f1-macro: 0.662310
23:     Loss 4.481206427972602e-06, accuracy: 0.7373198767504936, f1-macro: 0.6632319
24:     Loss 4.406753134444333e-06, accuracy: 0.741520446256696, f1-macro: 0.67154360
25:     Loss 4.363998293153021e-06, accuracy: 0.7431071913995954, f1-macro: 0.6741347
26:     Loss 4.322358585163103e-06, accuracy: 0.7458378690873755, f1-macro: 0.6771111
27:     Loss 4.271823837870648e-06, accuracy: 0.7477997749035961, f1-macro: 0.6807209
28:     Loss 4.240151046163306e-06, accuracy: 0.7494787726710825, f1-macro: 0.6829080
29:     Loss 4.18690874641717e-06, accuracy: 0.7520925970343856, f1-macro: 0.68718359
30:     Loss 4.135663569033294e-06, accuracy: 0.754915527346753, f1-macro: 0.69284251
31:     Loss 4.092793988516691e-06, accuracy: 0.75714189068679, f1-macro: 0.695745535
32:     Loss 4.066884918340519e-06, accuracy: 0.7585379804055425, f1-macro: 0.7000830
33:     Loss 4.026599353156436e-06, accuracy: 0.7608627465451392, f1-macro: 0.7020745
34:     Loss 3.976631703568535e-06, accuracy: 0.763304366009212 9, f1-macro: 0.7055221
35:     Loss 3.945131937146617e-06, accuracy: 0.7652724220004059, f1-macro: 0.7072132
36:     Loss 3.896189242427453e-06, accuracy: 0.7676709902396723, f1-macro: 0.7138451
37:     Loss 3.862625269687275e-06, accuracy: 0.7697928006051772, f1-macro: 0.7148866
38:     Loss 3.8197406594318956e-06, accuracy: 0.7727510347669391, f1-macro: 0.720878
39:     Loss 3.7906287885804183e-06, accuracy: 0.7744423328843706, f1-macro: 0.722762
40:     Loss 3.7456737401638717e-06, accuracy: 0.7760782794270497, f1-macro: 0.724956
41:     Loss 3.713377764165811e-06, accuracy: 0.779350172512408, f1-macro: 0.72937697
42:     Loss 3.6746670223422764e-06, accuracy: 0.7805433064570687, f1-macro: 0.732772
43:     Loss 3.640653257175796e-06, accuracy: 0.7825728641979864, f1-macro: 0.7343082
44:     Loss 3.607320962360168e-06, accuracy: 0.7839812542666839, f1-macro: 0.7403553
45:     Loss 3.568171058901173e-06, accuracy: 0.7859677607827943, f1-macro: 0.7427950
46:     Loss 3.5515547662366166e-06, accuracy: 0.7877390111748679, f1-macro: 0.742860
```

```
47:     Loss 3.5106302538650488e-06, accuracy: 0.7896209647164462, f1-macro: 0.74754
48:     Loss 3.496188874375104e-06, accuracy: 0.7896148145414738, f1-macro: 0.746818
49:     Loss 3.4430745106980625e-06, accuracy: 0.7930343118261715, f1-macro: 0.75247
```

```
def count_metrics(model, dataloader):
  y_pred, y_true = predict_on_dataloader(model, dataloader)

  y_pred = y_pred.detach().cpu().numpy()
  y_true = y_true.detach().cpu().numpy()

  mask = y_true != pad_inds
  y_true = y_true[mask]
  y_pred = np.argmax(y_pred, axis=1)[mask]

  print(classification_report(y_true, y_pred))
```

```
count_metrics(model, dataloader)
```

```
              precision    recall  f1-score   support

           0       0.63      0.36      0.46      9962
           1       0.89      0.85      0.87     13578
           2       0.70      0.70      0.70      8547
           3       0.86      0.95      0.90     10404
           4       0.99      0.99      0.99      5202
           5       0.95      0.98      0.96     13014
           6       0.95      0.63      0.76       649
           7       0.58      0.86      0.69     27080
           8       0.94      0.94      0.94      3339
           9       0.76      0.93      0.83      4484
          10       0.95      0.95      0.95     15619
          11       0.76      0.17      0.28     10523
          12       0.99      1.00      0.99     16990
          13       0.77      0.60      0.67      3134
          14       0.86      0.78      0.82       484
          15       0.74      0.70      0.72     18849
          16       0.96      0.14      0.25       739

    accuracy                           0.79    162597
   macro avg       0.84      0.74      0.75    162597
weighted avg       0.80      0.79      0.78    162597
```

```
data = TensorDataset(X_test, Y_test)
test_dataloader = DataLoader(data, sampler=SequentialSampler(data), batch_size=bs)
count_metrics(model, test_dataloader)
```

```
              precision    recall  f1-score   support

           0       0.59      0.35      0.44      1466
           1       0.86      0.83      0.84      1656
           2       0.64      0.64      0.64      1066
           3       0.81      0.91      0.86      1336
           4       0.99      0.99      0.99       599
           5       0.94      0.97      0.96      1607
           6       0.94      0.43      0.59       115
           7       0.51      0.80      0.63      3446
           8       0.91      0.95      0.93       448
           9       0.70      0.83      0.76       546
          10       0.92      0.93      0.92      1923
          11       0.62      0.08      0.15      1773
          12       0.98      0.99      0.98      2467
          13       0.62      0.45      0.52       330
          14       0.67      0.56      0.61        81
          15       0.64      0.68      0.66      2306
          16       0.00      0.00      0.00       114

    accuracy                           0.74     21279
   macro avg       0.73      0.67      0.67     21279
weighted avg       0.74      0.74      0.72     21279

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: Unde
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: Unde
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: Unde
  _warn_prf(average, modifier, msg_start, len(result))
```

✓ 0 сек. выполнено в 14:55