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


# Отчет
# Лабораторная работа № 7
# 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**

Выбрать свой корпус текста и обучить последовательно на нём три архитектуры нейронных сетей на задаче Next Token Prediction (предсказание следующего токена).

Архитектуры:

- Одномерная свёрточная нейронная сеть
- Рекуррентная нейронная сеть
- Трансформер кодировщик
- Визуализировать полученное векторное представление токенов. Сохранить модели в формате onnx.

Дописать код для генерации текста по входной последовательности.

В качестве примера приводится обучение на русском корпусе текста - произведении Ф.М. Достоевского "Преступление и наказание".

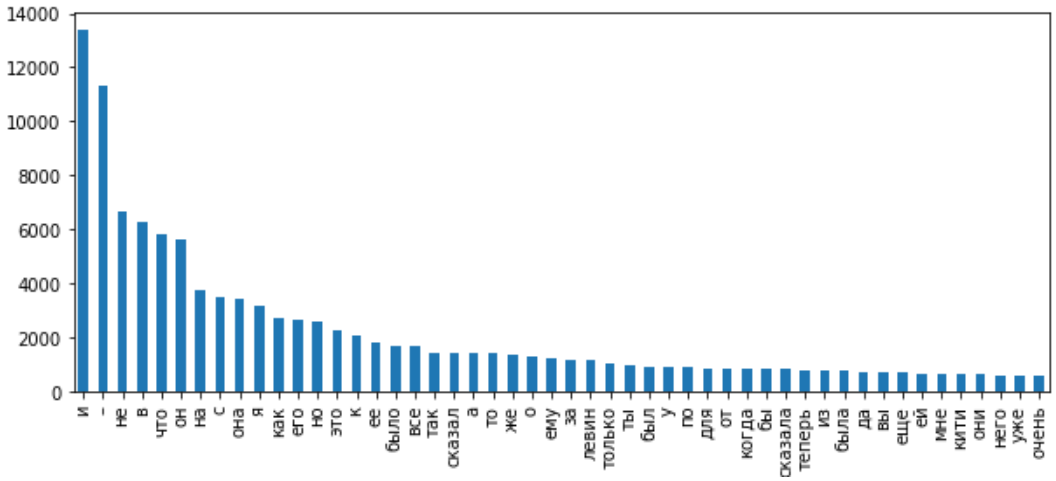
**Выполнение**

```python
def filter_punctuation(x):
    table = str.maketrans('', '', punctuation)
    x = map(lambda x: x.lower(), x)
    x = map(lambda x: x.translate(table), x)
    x = filter(lambda x: len(x)>0, x)
    return list(x)

FILENAME = 'anna-karenina.txt'
corpus = []
# encoding может быть один из ['ascii', 'utf-8', 'cp1251']
with open(FILENAME, 'r',
          encoding='utf-8') as f:
    for line in f:
        corpus.extend(list(filter(lambda x: len(x)>0, line.split())))

corpus = filter_punctuation(corpus)
with open('corpus.txt', 'w') as f:
    f.write('\n'.join(corpus))
corpus_df = pd.Series(corpus)
corpus_df.value_counts().head(50).plot(kind='bar', figsize=(10, 4))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e18c17850>
```




Загрузка корпуса текста (Анна Каренина), очистака от пунктуации, токенизация

Используется алгоритм WordPiece

```
[ ]  spm.SentencePieceTrainer.train(input='corpus.txt', model_prefix='m'
                                    vocab_size=15000)
```

```
corpus_df
```

```
0         annotation
1              «анна
2           каренина»
3               один
4                 из
              ...
295714    васильевка
295715           спб
295716          1904
295717             с
295718         30-31
Length: 295719, dtype: object
```

Обучение токенизатора, размер словоря 15000

```
vocabs = [[sp.id_to_piece(id), id] for id in range(sp.get_piece_size())]
vocabs[:20]
```

```
[['<unk>', 0],
 ['<s>', 1],
 ['</s>', 2],
 ['▁', 3],
 ['▁и', 4],
 ['-', 5],
 ['а', 6],
 ['е', 7],
 ['▁не', 8],
 ['и', 9],
 ['▁в', 10],
 ['▁он', 11],
 ['▁что', 12],
 ['й', 13],
 ['у', 14],
 ['я', 15],
 ['▁с', 16],
 ['▁на', 17],
 ['▁она', 18],
 ['▁я', 19]]
```

Полученные токены

## Обучение в Pytorch

пример основан на

https://github.com/pytorch/examples/tree/master/word_language_model

```
[ ]  class ConvModel(nn.Module):
        """Container module with an encoder, a convolutional module, and a decoder."""

        def __init__(self, ntoken, ninp, nhid, dropout=0.1, tie_weights=False):
            super(ConvModel, self).__init__()
            self.ntoken = ntoken
            self.drop = nn.Dropout(dropout)
            self.encoder = nn.Embedding(ntoken, ninp)
            self.conv = nn.Sequential(
                        nn.Conv1d(ninp, nhid//2, 5, stride=3, padding=2, dilation=1),
                        nn.ReLU(),
                        nn.Conv1d(nhid//2, nhid, 5, stride=3, padding=2, dilation=1),
                        nn.ReLU(),
                        nn.Conv1d(nhid, nhid*2, 5, stride=3, padding=2, dilation=1),
                        nn.ReLU(),
                        )
            self.decoder = nn.Linear(nhid*2, ntoken)

            self.init_weights()

            self.nhid = nhid
            self.model_type = 'Conv'

        def init_weights(self):
            initrange = 0.1
            nn.init.uniform_(self.encoder.weight, -initrange, initrange)
            nn.init.zeros_(self.decoder.bias)
            nn.init.uniform_(self.decoder.weight, -initrange, initrange)

        def forward(self, inputs):
            emb = self.drop(self.encoder(inputs))
            emb = emb.transpose(2, 1)
            output = self.conv(emb)
            #print(output.size())
```

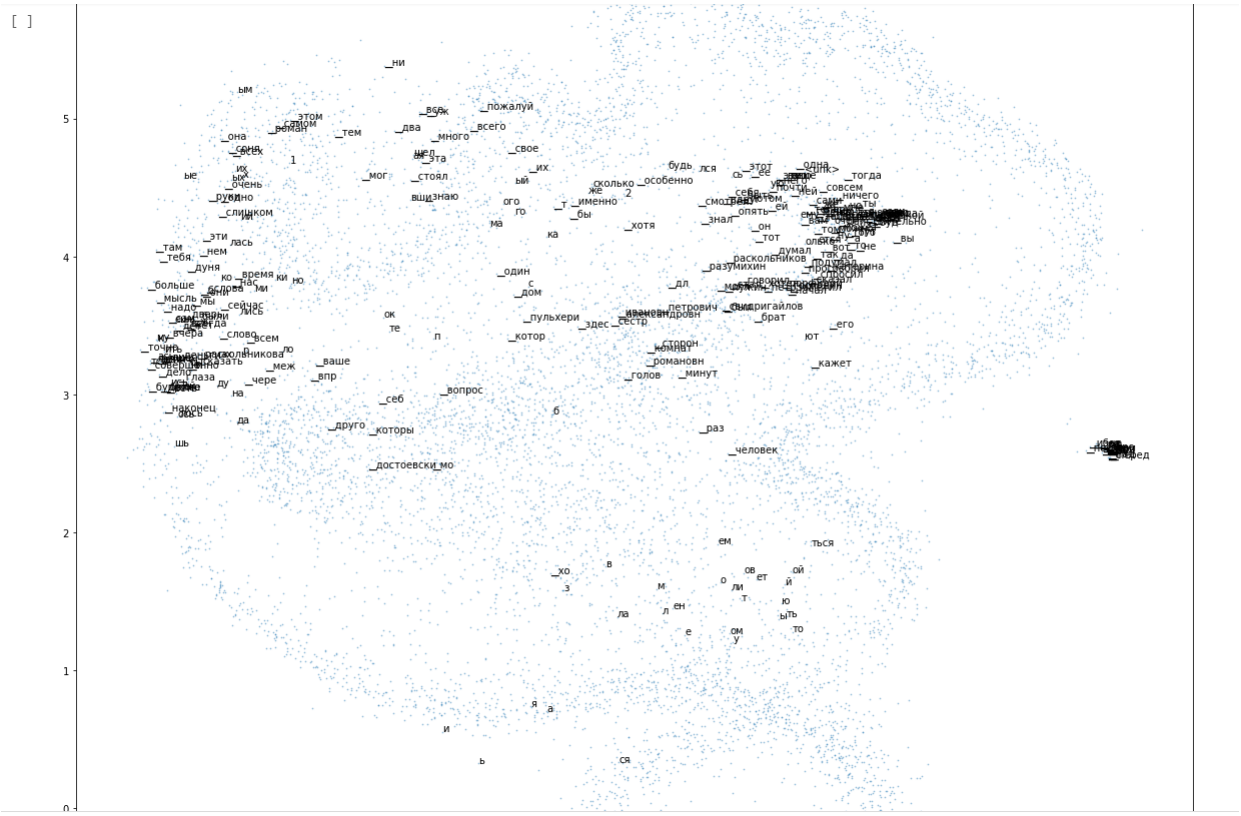
Определение моделей (Одномерная свёрточная, Рекуррентная, Трансформер)

```
# Loop over epochs.
epochs = 40
best_val_loss = None
savefile = '%s_next_token.pt'%model_type

patience_max = 1
patience = 0
# At any point you can hit Ctrl + C to break out of training early.
try:
    for epoch in range(1, epochs+1):
        epoch_start_time = time.time()
        train()
        val_loss = evaluate(y)
        print('-' * 89)
        print('| end of epoch {:3d} | time: {:5.2f}s | valid loss {:5.3f} | '
                'valid ppl {:8.2f}'.format(epoch, (time.time() - epoch_start_time),
                                           val_loss, math.exp(val_loss)))
        print('-' * 89)
        # Save the model if the validation loss is the best we've seen so far.
        if not best_val_loss or val_loss < best_val_loss:
            with open(savefile, 'wb') as f:
                torch.save(model, f)
            best_val_loss = val_loss
            patience = 0
        else:
            patience += 1
            if patience < patience_max: continue
            with open(savefile, 'rb') as f:
                model = torch.load(f)
            # Anneal the learning rate if no improvement has been seen in the validation dataset.
            lr /= 2.0
            patience = 0
except KeyboardInterrupt:
    print('-' * 89)
    print('Exiting from training early')
```

```
| epoch  39 |  3700/ 4408 batches | lr 1.00 | ms/batch 11.358 | loss  4.99 | ppl   147.05
| epoch  39 |  3800/ 4408 batches | lr 1.00 | ms/batch 11.372 | loss  4.95 | ppl   141.26
| epoch  39 |  3900/ 4408 batches | lr 1.00 | ms/batch 11.425 | loss  4.99 | ppl   146.71
| epoch  39 |  4000/ 4408 batches | lr 1.00 | ms/batch 11.424 | loss  4.96 | ppl   142.75
| epoch  39 |  4100/ 4408 batches | lr 1.00 | ms/batch 11.305 | loss  4.97 | ppl   144.42
| epoch  39 |  4200/ 4408 batches | lr 1.00 | ms/batch 11.400 | loss  4.98 | ppl   146.13
| epoch  39 |  4300/ 4408 batches | lr 1.00 | ms/batch 11.437 | loss  4.96 | ppl   142.65
```

Обучение одномерной свёрточной модели



Полученный эмбеддинг

```
model = RNNModel('LSTM', np.unique(y).size, 50, 256, 2, 0.1, False).to(device)
criterion = nn.NLLLoss()
model
```

```
RNNModel(
  (drop): Dropout(p=0.1, inplace=False)
  (encoder): Embedding(14916, 50)
  (rnn): LSTM(50, 256, num_layers=2, dropout=0.1)
  (decoder): Linear(in_features=256, out_features=14916, bias=True)
)
```

```
model_type = 'LSTM'
lr=10
bptt = 1024
eval_batch_size = bptt
clip = 0.25
log_interval = 10
dry_run = False
```

Модель LSTM

```
# Loop over epochs.
epochs = 30
best_val_loss = None
savefile = '%s_next_token.pt'%model_type

# At any point you can hit Ctrl + C to break out of training early.
try:
    for epoch in range(1, epochs+1):
        epoch_start_time = time.time()
        train()
        val_loss = evaluate(y)
        print('-' * 89)
        print('| end of epoch {:3d} | time: {:5.2f}s | valid loss {:5.3f} | '
                'valid ppl {:8.2f}'.format(epoch, (time.time() - epoch_start_time),
                                           val_loss, math.exp(val_loss)))
        print('-' * 89)
        # Save the model if the validation loss is the best we've seen so far.
        if not best_val_loss or val_loss < best_val_loss:
            with open(savefile, 'wb') as f:
                torch.save(model, f)
            best_val_loss = val_loss
        else:
            # Anneal the learning rate if no improvement has been seen in the validation dataset.
            lr /= 2.0
except KeyboardInterrupt:
    print('-' * 89)
    print('Exiting from training early')
```

```
| epoch   6 |    50/  404 batches | lr 10.00 | ms/batch 207.665 | loss  5.76 | ppl   318.92
| epoch   6 |    60/  404 batches | lr 10.00 | ms/batch 207.028 | loss  5.77 | ppl   321.48
| epoch   6 |    70/  404 batches | lr 10.00 | ms/batch 206.674 | loss  5.73 | ppl   307.39
| epoch   6 |    80/  404 batches | lr 10.00 | ms/batch 207.445 | loss  5.75 | ppl   313.73
| epoch   6 |    90/  404 batches | lr 10.00 | ms/batch 206.659 | loss  5.75 | ppl   313.12
| epoch   6 |   100/  404 batches | lr 10.00 | ms/batch 207.246 | loss  5.72 | ppl   304.25
| epoch   6 |   110/  404 batches | lr 10.00 | ms/batch 207.123 | loss  5.81 | ppl   335.10
| epoch   6 |   120/  404 batches | lr 10.00 | ms/batch 207.215 | loss  5.70 | ppl   299.56
| epoch   6 |   130/  404 batches | lr 10.00 | ms/batch 206.625 | loss  5.70 | ppl   298.11
| epoch   6 |   140/  404 batches | lr 10.00 | ms/batch 207.141 | loss  5.73 | ppl   306.95
| epoch   6 |   150/  404 batches | lr 10.00 | ms/batch 207.307 | loss  5.72 | ppl   305.09
```

Обучение


Полученный эмбеддинг

```
model = TransformerModel(np.unique(y).size, 128+32, 4, 192, 2, 0.1).to(device)
# ntoken, ninp, nhead, nhid, nlayers, dropout
criterion = nn.NLLLoss()
model
```

```
TransformerModel(
  (pos_encoder): PositionalEncoding(
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (transformer_encoder): TransformerEncoder(
    (layers): ModuleList(
      (0): TransformerEncoderLayer(
        (self_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=160, out_features=160, bias=True)
        )
        (linear1): Linear(in_features=160, out_features=192, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
        (linear2): Linear(in_features=192, out_features=160, bias=True)
        (norm1): LayerNorm((160,), eps=1e-05, elementwise_affine=True)
        (norm2): LayerNorm((160,), eps=1e-05, elementwise_affine=True)
        (dropout1): Dropout(p=0.1, inplace=False)
        (dropout2): Dropout(p=0.1, inplace=False)
      )
      (1): TransformerEncoderLayer(
        (self_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=160, out_features=160, bias=True)
        )
        (linear1): Linear(in_features=160, out_features=192, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
        (linear2): Linear(in_features=192, out_features=160, bias=True)
        (norm1): LayerNorm((160,), eps=1e-05, elementwise_affine=True)
        (norm2): LayerNorm((160,), eps=1e-05, elementwise_affine=True)
        (dropout1): Dropout(p=0.1, inplace=False)
        (dropout2): Dropout(p=0.1, inplace=False)
      )
    )
```

```
[ ] model_type = 'Transformer'
    lr = 2.0
    bptt = 64
    eval_batch_size = bptt
    clip = 0.25
    log_interval = 100
    dry_run = False
```

Вывод числа параметров

```
print(*[(k, p.view(-1).size()[0]) for k, p in model.named_parameters()],
      sep='\n')
print('Всего: {:,}'.format(sum(p.view(-1).size()[0] for k, p in model.named_parameters())))
```

```
('transformer_encoder.layers.0.self_attn.in_proj_weight', 76800)
('transformer_encoder.layers.0.self_attn.in_proj_bias', 480)
('transformer_encoder.layers.0.self_attn.out_proj.weight', 25600)
('transformer_encoder.layers.0.self_attn.out_proj.bias', 160)
('transformer_encoder.layers.0.linear1.weight', 30720)
('transformer_encoder.layers.0.linear1.bias', 192)
('transformer_encoder.layers.0.linear2.weight', 30720)
('transformer_encoder.layers.0.linear2.bias', 160)
('transformer_encoder.layers.0.norm1.weight', 160)
('transformer_encoder.layers.0.norm1.bias', 160)
('transformer_encoder.layers.0.norm2.weight', 160)
('transformer_encoder.layers.0.norm2.bias', 160)
('transformer_encoder.layers.1.self_attn.in_proj_weight', 76800)
('transformer_encoder.layers.1.self_attn.in_proj_bias', 480)
('transformer_encoder.layers.1.self_attn.out_proj.weight', 25600)
('transformer_encoder.layers.1.self_attn.out_proj.bias', 160)
('transformer_encoder.layers.1.linear1.weight', 30720)
('transformer_encoder.layers.1.linear1.bias', 192)
('transformer_encoder.layers.1.linear2.weight', 30720)
('transformer_encoder.layers.1.linear2.bias', 160)
('transformer_encoder.layers.1.norm1.weight', 160)
('transformer_encoder.layers.1.norm1.bias', 160)
('transformer_encoder.layers.1.norm2.weight', 160)
('transformer_encoder.layers.1.norm2.bias', 160)
('encoder.weight', 2386560)
('decoder.weight', 2386560)
('decoder.bias', 14916)
Всего: 5,118,980
```

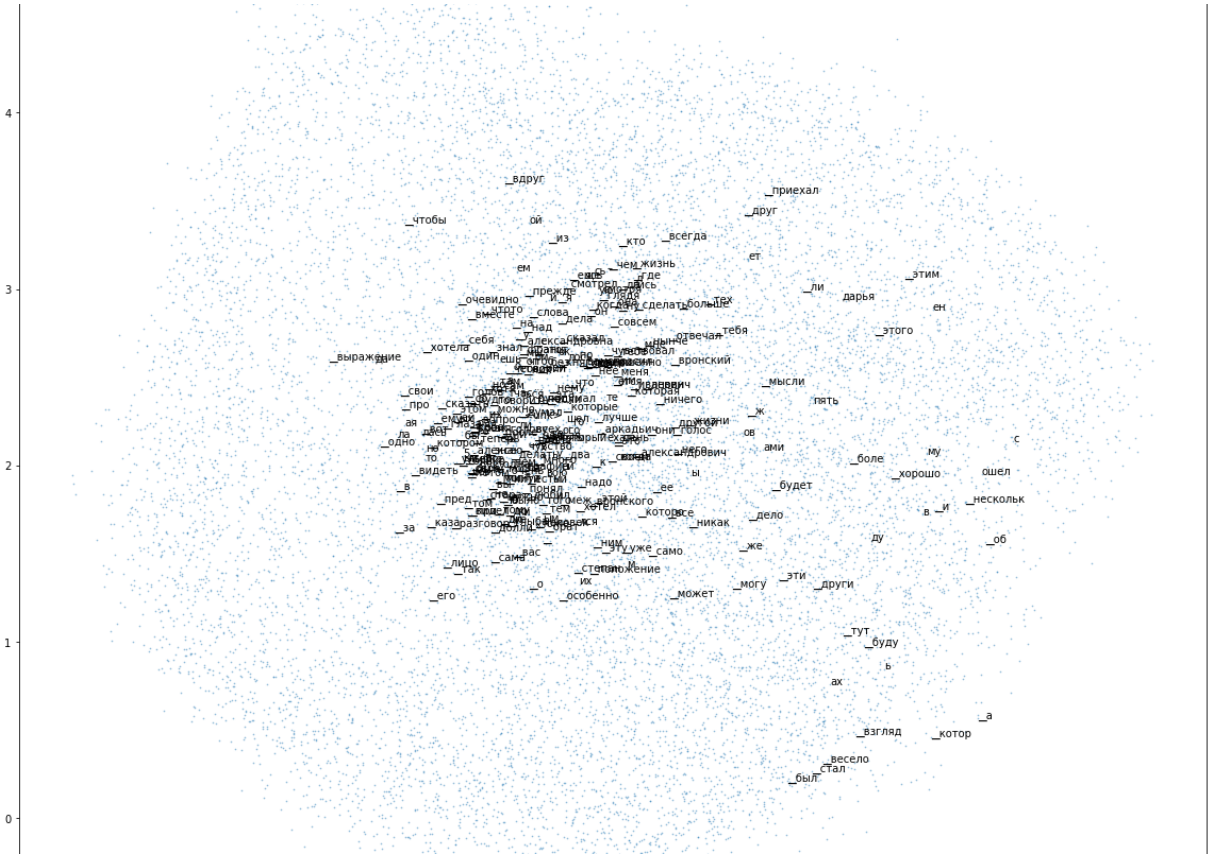
# Модель Трансформер

```
# Loop over epochs.
epochs = 40
best_val_loss = None
savefile = '%s_next_token.pt'%model_type

patience_max = 2
patience = 0
# At any point you can hit Ctrl + C to break out of training early.
try:
    for epoch in range(1, epochs+1):
        epoch_start_time = time.time()
        train()
        val_loss = evaluate(y)
        print('-' * 89)
        print('| end of epoch {:3d} | time: {:5.2f}s | valid loss {:5.3f} | '
                'valid ppl {:8.2f}'.format(epoch, (time.time() - epoch_start_time),
                                           val_loss, math.exp(val_loss)))
        print('-' * 89)
        # Save the model if the validation loss is the best we've seen so far.
        if not best_val_loss or val_loss < best_val_loss:
            with open(savefile, 'wb') as f:
                torch.save(model, f)
            best_val_loss = val_loss
            patience = 0
        else:
            patience += 1
            if patience < patience_max: continue
            with open(savefile, 'rb') as f:
                model = torch.load(f)
            # Anneal the learning rate if no improvement has been seen in the validation dataset.
            lr /= 2.0
            patience = 0
except KeyboardInterrupt:
    print('-' * 89)
    print('Exiting from training early')
```

```
| epoch   1 |   100/ 6464 batches | lr 2.00 | ms/batch 11.824 | loss  7.80 | ppl  2431.27
| epoch   1 |   200/ 6464 batches | lr 2.00 | ms/batch 11.387 | loss  7.10 | ppl  1216.77
| epoch   1 |   300/ 6464 batches | lr 2.00 | ms/batch 11.418 | loss  6.87 | ppl   964.32
| epoch   1 |   400/ 6464 batches | lr 2.00 | ms/batch 11.398 | loss  6.80 | ppl   895.68
| epoch   1 |   500/ 6464 batches | lr 2.00 | ms/batch 11.412 | loss  6.75 | ppl   851.17
| epoch   1 |   600/ 6464 batches | lr 2.00 | ms/batch 11.389 | loss  6.64 | ppl   762.08
```

Обучение



Эмбеддинг

```
sess1 = onnxruntime.InferenceSession('LSTM_next_token.onnx')
text = 'привет, как дела'
inputs = sp.encode(text)[-16:]
inputs = [0]*max(16 - len(inputs), 0) + inputs
finalresult = text
for i in range(100):
    token = sess1.run(None, {'input.1': np.array(inputs, dtype=np.int64).reshape(1, 16),
                      'onnx::Slice_1': np.zeros((2, 16, 256), dtype=np.float32),
                      'onnx::Slice_2': np.zeros((2, 16, 256), dtype=np.float32)})[0]
    inputs.pop(0)
    inputs.append(int(token[-1].argmax()))
    print(sp.decode([int(token[-1].argmax())]))
    finalresult = finalresult +' '+ sp.decode([int(token[-1].argmax())])
print(finalresult)
```

```
и

я

лась

 
а

иванович

а

иванович
```

Работа модели LSTM в качестве генератора следующего слова