

Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»

(национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ _	«Информатика и системы управления»
КАФЕДРА	«Теоретическая информатика и компьютерные технологии»

Домашняя работа №6 по курсу «Теория искусственных нейронных сетей»

«Рекурентные нейронные сети (RNN)»

Студент группы ИУ9-71Б Окутин Д. А.

Преподаватель Каганов Ю. Т.

1 Цель

Ознакомление с основными архитектурами рекурентных нейронных сетей.

2 Задание

- 1. Реализовать нейронную сеть архитектуры RNN и проверить её на датасете IMDB.
- 2. Реализовать нейронную сеть архитектуры LSTM и проверить её на датасете IMDB.
- 3. Реализовать нейронную сеть архитектуры GRU и проверить её на датасете IMDB.

3 Реализация

Исходный код представлен в листинге 1 - 5.

Листинг 1: Подготовка датасета

```
2 import numpy as np
3 import torch
4 import torch.autograd as autograd
5 import torch.nn as nn
6 import torch.nn.functional as F
7 import torch.optim as optim
8 from torch.autograd import Variable
9 from datasets import load_dataset
10 import pandas as pd
11
  from sklearn.model_selection import train_test_split
12
13 from IPython. display import clear output
14 import sys
15
16 import nltk
17 from nltk.corpus import stopwords
18 from nltk.stem import WordNetLemmatizer
19 import re
20 import string
21
22 from sklearn.metrics import classification report, confusion matrix
```

```
23
24 import matplotlib.pyplot as plt
25 import seaborn as sns
26
27 from torch.utils.data import DataLoader, Dataset
28 from tqdm import tqdm
29 import subprocess
30
31 try:
32
                    nltk.data.find('wordnet.zip')
33 except:
34
                    nltk.download('wordnet', download dir='/kaggle/working/')
35
                   command = "unzip /kaggle/working/corpora/wordnet.zip -d /kaggle/
                   working/corpora"
                    subprocess.run(command.split())
36
37
                    nltk.data.path.append('/kaggle/working/')
38
       dataset = load dataset("imdb")
39
40
41 train = pd. DataFrame(dataset["train"])
42 test = pd. DataFrame(dataset["test"])
43
44 df = pd.concat([train, test], ignore index=True)
45 df
46
47 train_text, test_text, train_labels, test_labels= train test split(df['
                  \texttt{text']} \;,\;\; \texttt{df['label']} \;,\;\; \texttt{test\_size} = 0.2, \;\; \texttt{random\_state} = 1024, \\ \texttt{stratify} = \texttt{df['label']} \;,\;\; \texttt{text'} = 0.24, \\ \texttt{stratify} = 0
                  label'])
48
49 print (f"Training samples: {len(train text)}")
        print(f"Testing samples: {len(test text)}")
50
51
52 lemmatizer = WordNetLemmatizer()
53
       stop words = set(stopwords.words('english'))
54
       def clean text(text):
55
                    text = re.sub(r'<.*?>', '', text) # Remove HTML tags
56
57
                    text = re.sub(r'http\S+|www.\S+', '', text) \# Remove URLs
                    text = re.sub(r'\d+', '', text) # Remove numbers
58
59
                    text = text.translate(str.maketrans('', '', string.punctuation)) #
                  Remove punctuation
60
                    text = text.strip() # Remove extra whitespace
                    text = text.lower() # Lowercase
61
62
63
                   #create tokens
64
                    tokens = nltk.word_tokenize(text)
```

```
65
       tokens = [lemmatizer.lemmatize(word) for word in tokens if word not
       in stop words]
       return tokens
66
67
68 train tokens = train text.apply(clean text)
69 test tokens = test text.apply(clean text)
70
71 from collections import Counter
72 all tokens = [token for tokens in train tokens for token in tokens] #
       flatten the list of tokens
73 token_count = Counter(all_tokens) #count the frequency of each token
74 MAX VOCAB SIZE = 40000 \# set the maximum vocab size to the number of
       tokens
75 vocab = ['<PAD>','<UNK>'] + [word for word, freq in token count.
      most common(MAX VOCAB SIZE - 2) | #create the vocab
76 wordtoidx = {word: idx for idx, word in enumerate(vocab)} #create the
       word to index mapping
77 unk idx = wordtoidx['<UNK>']
78 pad idx = wordtoidx['<PAD>']
79
80 def encode tokens (tokens, wordtoidx, max len = 200):
81
       encoded = [wordtoidx.get(token, unk idx) for token in tokens]
       if len(encoded) < max len:
82
83
            encoded += [pad idx] * (max len - len(encoded))
84
       else:
            encoded = encoded [:max_len]
85
86
       return np. array (encoded)
87
88 #encode all the datasets
89 train encoded = np.array([encode tokens(tokens, wordtoidx) for tokens in
        train tokens])
90 test encoded = np.array([encode tokens(tokens, wordtoidx) for tokens in
       test tokens])
91
92 #convert them to numpy array
93 X train = np.array(train encoded.tolist())
94 | X_test = np.array(test_encoded.tolist())
95
96 def code(arr):
97
       result = []
98
99
       for value in arr:
100
            if value == 1:
                result.append([0, 1])
101
102
            elif value == 0:
103
                result.append ([1, 0])
```

```
104
105
        return result
106
107
108
   y_train = np.array(code(train_labels))
   y_test = np.array(code(test_labels))
109
110
   print(f"X train shape: {X train.shape}")
111
   print(f"X test shape: {X test.shape}")
112
113
114
   class IMDBDataset(Dataset):
115
        def __init__(self, texts, labels):
            self.texts = torch.tensor(texts, dtype=torch.long)
116
117
            self.labels = torch.tensor(labels, dtype=torch.float)
118
119
        def \__len__(self):
            return len (self.labels)
120
121
       def \__getitem\__(self, idx):
122
            return self.texts[idx], self.labels[idx]
123
124
125
126
127 train dataset = IMDBDataset(X train, y train)
   test dataset = IMDBDataset(X test, y test)
129
130 train loader = DataLoader(train dataset, batch size=16, shuffle=True)
   test loader = DataLoader(test dataset, batch size=16)
131
132
133 device = 'cuda' if torch.cuda.is available() else 'cpu'
134 device
```

Листинг 2: Функция тренировки

```
1
2 def train (network, train loader, test loader, epochs, loss fn, optim, plot=
      True, verbose=True):
      train loss epochs = []
3
4
       test loss epochs = []
5
      train_accuracy_epochs = []
      test accuracy epochs = []
6
7
8
      try:
9
           for epoch in tqdm(range(epochs)):
10
               losses = []
11
               accuracies = []
12
               for X, y in train_loader:
```

```
13
                   X, y = X. to(device), y. to(device)
14
                    pred = model(X)
15
                    loss batch = loss fn(pred, y)
                    losses.append(loss batch.item())
16
17
                    optim.zero_grad()
18
                    loss batch.backward()
19
                    optim.step()
20
                    pred1 = pred.argmax(dim=1).float()
21
                    y1 = y.argmax(dim=1).float()
22
                    accuracies.append((torch.sum(y1 = pred1)/16).cpu())
23
               train_loss_epochs.append(np.mean(losses))
24
               train accuracy epochs.append(np.mean(accuracies))
25
26
               with torch.no grad():
27
                    losses = []
28
                    accuracies = []
29
                    for X, y in test loader:
                        X, y = X. to(device), y. to(device)
30
31
                        pred = model(X).squeeze(1)
32
                        loss_batch = loss_fn(pred, y)
33
                        losses.append(loss_batch.cpu())
34
                        pred1 = pred.argmax(dim=1).float()
35
                        y1 = y.argmax(dim=1).float()
36
                        accuracies.append((torch.sum(y1 = pred1)/16).cpu())
37
               test loss epochs.append(np.mean(losses))
               test_accuracy_epochs.append(np.mean(accuracies))
38
39
               clear output (True)
40
               if verbose:
41
                    sys.stdout.write('\rEpoch {0}... (Train/Test) Loss:
      \{1:.3f\}/\{2:.3f\}\setminus tAccuracy: \{3:.3f\}/\{4:.3f\}'. format (
42
                                epoch, train_loss_epochs[-1],
      test loss epochs [-1],
43
                                 train accuracy epochs [-1],
      test_accuracy_epochs[-1]))
44
               if plot:
45
                    plt. figure (figsize = (12, 5))
                    plt.subplot(1, 2, 1)
46
47
                    plt.plot(train loss epochs, label='Train')
                    plt.plot(test_loss_epochs, label='Test')
48
49
                    plt.xlabel('Epochs', fontsize=16)
50
                    plt.ylabel('Loss', fontsize=16)
51
                    plt.legend(loc=0, fontsize=16)
52
                    plt.grid('on')
53
                    plt.subplot(1, 2, 2)
                    plt.plot(train accuracy epochs, label='Train accuracy')
54
                    plt.plot(test_accuracy_epochs, label='Test accuracy')
55
```

```
56
                     plt.xlabel('Epochs', fontsize=16)
57
                     plt.ylabel('Accuracy', fontsize=16)
58
                     plt.legend(loc=0, fontsize=16)
59
                     plt.grid('on')
                     plt.show()
60
61
       except KeyboardInterrupt:
62
            pass
63
       return train loss epochs, \
               test loss epochs, \
64
               train\_accuracy\_epochs\;,\;\; \backslash
65
66
               test_accuracy_epochs
```

Листинг 3: RNN

```
1
2
  class MyRNNModel(nn.Module):
3
       def __init__(self , input_size , hidden_size):
4
           super(MyRNNModel, self).__init__()
5
           self.lin1 = nn.Linear(input size, hidden size)
6
7
           self.lin2 = nn.Linear(hidden_size, hidden_size)
8
           self.tanh = nn.Tanh()
           self.hidden_size = hidden_size
10
11
12
       def forward (self, x):
13
           batch_size, seq_len, _ = x.size()
14
           h_t = torch.zeros(batch_size, self.hidden_size)
15
           output = []
           for t in range (seq len):
16
17
               xt = x[:, t, :]
18
               xt.to(device)
19
20
               h_t = self.tanh(self.lin1(xt) + self.lin2(h_t))
               output.append(h_t)
21
22
           output = torch.stack(output)
23
           output = output.transpose(0, 1)
24
25
           return output, h t
26
27
  class SimpleRNN(nn. Module):
28
       def __init__(self, vocab_size, embedding_dim, hidden_dim, output_dim
      , pad idx):
29
           super(SimpleRNN, self).__init__()
30
31
           self.embedding = nn.Embedding(vocab\_size, embedding\_dim,
      padding_idx=pad_idx)
```

```
32
            self.rnn = MyRNNModel(embedding dim, hidden dim)
33
            self.fc = nn.Sequential(
34
                nn. Linear (hidden dim, output dim),
35
            )
36
37
       def forward (self, text):
38
39
            embedded = self.embedding(text)
40
            output, hidden = self.rnn(embedded)
            out = self.fc(output[:, -1,:])
41
42
            return out
43
44
45 | VOCAB SIZE = len(vocab) + 1
46 \mid \text{EMBEDDING DIM} = 128
47 | \text{HIDDEN\_DIM} = 128
48 | OUTPUT DIM = 2
49 | PAD_IDX = pad_idx
```

Листинг 4: LSTM

```
1
2
  class MyLSTMModel(nn.Module):
3
       def init (self, input size, hidden size):
4
           super(MyLSTMModel, self).__init__()
5
6
           self.hidden size = hidden size
7
           self.lin ix = nn.Linear(input size, hidden size)
8
9
           self.lin ih = nn.Linear(hidden size, hidden size)
10
           self.lin fx = nn.Linear(input size, hidden size)
11
           self.lin fh = nn.Linear(hidden size, hidden size)
12
           self.lin_gx = nn.Linear(input_size, hidden_size)
13
           self.lin gh = nn.Linear(hidden size, hidden size)
           self.lin_ox = nn.Linear(input_size, hidden_size)
14
           self.lin oh = nn.Linear(hidden size, hidden size)
15
16
17
           self.tanh = nn.Tanh()
18
           self.sigmoid = nn.Sigmoid()
19
20
       def forward (self, x):
21
           batch_size, seq_len, _ = x.size()
           h t = torch.zeros(batch size, self.hidden size).to(device)
22
23
           c_t = torch.zeros(batch_size, self.hidden_size).to(device)
24
           output = []
25
           for t in range (seq len):
26
               xt = x[:, t, :]
```

```
27
               xt.to(device)
28
29
               i_t = self.sigmoid(self.lin_ix(xt) + self.lin_ih(h_t))
               f t = self.sigmoid(self.lin fx(xt) + self.lin fh(h t))
30
               g_t = self.tanh(self.lin_gx(xt) + self.lin_gh(h_t))
31
               o_t = self.sigmoid(self.lin_ox(xt) + self.lin_oh(h_t))
32
33
               c_t = f_t * c_t + i_t * g_t
34
               h_t = o_t * self.tanh(c_t)
35
36
37
               output.append(h_t)
38
           output = torch.stack(output)
39
40
           output = output.transpose(0, 1)
           return output, (h t, c t)
41
42
43
  class SimpleLSTM(nn.Module):
      def __init__(self, vocab_size, embedding_dim, hidden_dim, output_dim
44
      , pad idx):
45
           super(SimpleLSTM, self).__init__()
46
47
           self.embedding = nn.Embedding(vocab size, embedding dim,
      padding idx=pad idx)
           self.rnn = MyLSTMModel(embedding dim, hidden dim)
48
49
           self.fc = nn.Sequential(
               nn. Linear (hidden dim, output dim),
50
51
52
           self.sigmoid = nn.Sigmoid()
53
54
       def forward (self, text):
55
           embedded = self.embedding(text)
56
           output, hidden = self.rnn(embedded)
57
58
           out = self.fc(output[:, -1,:])
59
           return self.sigmoid(out)
```

Листинг 5: GRU

```
class MyGRUModel(nn.Module):
    def __init__(self , input_size , hidden_size):
        super(GRUModel, self).__init__()

self.hidden_size = hidden_size

self.lin_rx = nn.Linear(input_size , hidden_size)
    self.lin_rh = nn.Linear(hidden_size , hidden_size)
```

```
10
            self.lin zx = nn.Linear(input size, hidden size)
11
            self.lin zh = nn.Linear(hidden size, hidden size)
12
            self.lin nx = nn.Linear(input size, hidden size)
            self.lin nh = nn.Linear(hidden size, hidden size)
13
14
15
            self.tanh = nn.Tanh()
16
            self.sigmoid = nn.Sigmoid()
17
       def forward (self, x):
18
            batch\_size, seq\_len, \_=x.size()
19
20
            h_t = torch.zeros(batch_size, self.hidden_size)
21
            output = []
            for t in range(seq_len):
22
                xt = x[:, t, :]
23
                xt.to(device)
24
25
26
                r t = self.sigmoid(self.lin rx(xt) + self.lin rh(h t))
                z_t = self.sigmoid(self.lin_zx(xt) + self.lin_zh(h_t))
27
                {\tt n\_t} \,=\, {\tt self.tanh} \, (\, {\tt self.lin\_nx} \, (\, {\tt xt} \,) \,+\, {\tt r\_t} \,\, * \,\, {\tt self.lin\_nh} \, (\, {\tt h\_t}) \,)
28
                h_t = (1 - z_t) * n_t + z_t * h_t
29
30
31
                output.append(h t)
32
            output = torch.stack(output)
33
34
            output = output.transpose(0, 1)
35
            return output, h t
36
37
  class SimpleGRU(nn.Module):
38
       def __init__(self, vocab_size, embedding_dim, hidden_dim, output_dim
       , pad idx):
39
            super(SimpleGRU, self).__init__()
40
            self.embedding = nn.Embedding(vocab_size, embedding_dim,
41
      padding idx=pad idx)
42
            self.rnn = nn.GRU(embedding_dim, hidden_dim, batch_first=True)
            self.fc = nn.Sequential(
43
44
                nn.Linear(hidden_dim,output_dim),
45
            )
46
47
48
       def forward (self, text):
49
            embedded = self.embedding(text)
            output, hidden = self.rnn(embedded)
50
            out = self.fc(output[:, -1,:])
51
            return out
52
```

4 Результаты

Результат представлен на рисунках 1 - 5.

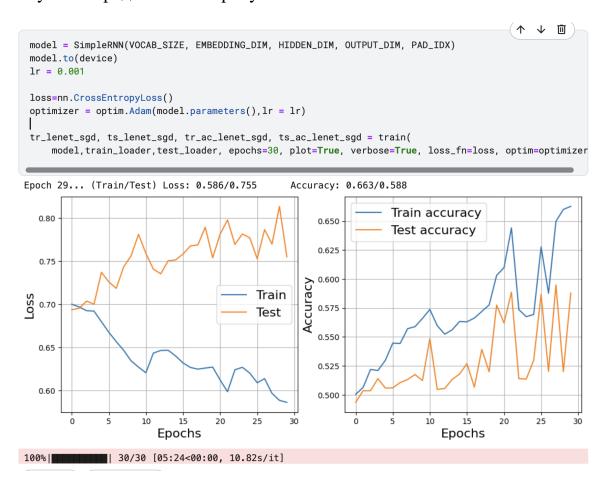


Рис. 1 — RNN Adam

```
model = SimpleRNN(VOCAB_SIZE, EMBEDDING_DIM, HIDDEN_DIM, OUTPUT_DIM, PAD_IDX)
    model.to(device)
    1r = 0.01
    loss=nn.CrossEntropyLoss()
    optimizer = optim.Adagrad(model.parameters(), lr = lr)
     tr_lenet_sgd, ts_lenet_sgd, tr_ac_lenet_sgd, ts_ac_lenet_sgd = train(
                      \verb|model|, \verb|train_loader|, test_loader|, epochs=20, plot= \verb|True|, verbose= \verb|True|, loss_fn= loss|, optim= optimizer| | optim= optimizer| | optim= optim=
Epoch 19... (Train/Test) Loss: 0.639/0.740
                                                                                                                                                                                                                            Accuracy: 0.569/0.534
                                                                                                                                                                                                                                                       0.57
                                                                                                                                                                                                                                                                                                               Train accuracy
                                                                   Train
                                                                    Test
                                                                                                                                                                                                                                                                                                               Test accuracy
           0.72
                                                                                                                                                                                                                                                       0.55
                                                                                                                                                                                                                                         Accuracy
0.53
0.52
            0.70
           0.66
                                                                                                                                                                                                                                                       0.50
           0.64
                                                                                                                              10.0 12.5
                                                                                                                                                                                                                                                                                                                                                                        10.0
                                                                                                             Epochs
                                                                                                                                                                                                                                                                                                                                                         Epochs
                                                           20/20 [03:13<00:00, 9.66s/it]
```

Рис. 2 — RNN Adagrad

```
model = SimpleRNN(VOCAB_SIZE, EMBEDDING_DIM, HIDDEN_DIM, OUTPUT_DIM, PAD_IDX)
model.to(device)
lr = 0.001
loss=nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=lr, momentum=0.8, nesterov=True)
 tr_lenet_sgd, ts_lenet_sgd, tr_ac_lenet_sgd, ts_ac_lenet_sgd = train(
    model,train_loader,test_loader, epochs=20, plot=True, verbose=True, loss_fn=loss, optim=optimizer
Epoch 19... (Train/Test) Loss: 0.685/0.695
                                                  Accuracy: 0.519/0.503
                                                       0.525
                                                                     Train accuracy
  0.694
                                                                     Test accuracy
                                                       0.520
  0.692
                                                       0.515
                                                    Accuracy
  0.690
                                                       0.510
                                                       0.505
  0.688
                Train
                                                       0.500
  0.686
                Test
                             10.0
                                  12.5
                                       15.0
                                                                                  10.0
                                                                                       12.5
                                                                                            15.0
                   5.0
                                                                  2.5
        0.0
             2.5
                                                             0.0
                         Epochs
                                                                              Epochs
             20/20 [02:59<00:00, 9.00s/it]
```

Рис. 3 — RNN Momentum

```
model = SimpleLSTM(VOCAB_SIZE, EMBEDDING_DIM, HIDDEN_DIM, OUTPUT_DIM, PAD_IDX)
         model.to(device)
        1r = 0.001
        loss=nn.CrossEntropyLoss()
        optimizer = optim.Adam(model.parameters(), lr = lr)
         \label{tr_lenet_sgd} \verb|tr_lenet_sgd|, ts_lenet_sgd|, tr_ac_lenet_sgd|, ts_ac_lenet_sgd| = train(
                           \verb|model|, train_loader|, test_loader|, epochs=30, \verb|plot=True|, verbose=True|, loss_fn=loss|, optim=optimizer|, optim=
    Epoch 29... (Train/Test) Loss: 0.010/0.830
                                                                                                                                                                                                                             Accuracy: 0.997/0.860
                                                                                                                                                                                                                                                         1.0
                0.8
                                                                                                                                                                                                                                                         0.9
                0.6
                                                                                                                                                                                                                                           Accuracy 8'0
   O.4
                0.2
                                                                                                                                                                                                                                                         0.6
                                                                                                                                                                                                                                                                                                                                                                                       Train accuracy
                                                                    Train
                                                                    Test
                                                                                                                                                                                                                                                                                                                                                                                       Test accuracy
                0.0
                                                                                                                                                                                                                                                                                                                                                                      15
                                                                                                                              15
                                                                                                                                                                                           25
                                                                                                                                                                                                                                                                                                                                                                                                      20
                                                                                                             Epochs
                                                                                                                                                                                                                                                                                                                                                       Epochs
100%|| 30/30 [06:47<00:00, 13.58s/it]
```

Рис. 4 — LSTM Adam

```
model = SimpleLSTM(VOCAB_SIZE, EMBEDDING_DIM, HIDDEN_DIM, OUTPUT_DIM, PAD_IDX)
 model.to(device)
 lr = 0.001
 loss=nn.CrossEntropyLoss()
 optimizer = optim.Adagrad(model.parameters(), lr = lr)
 tr_lenet_sgd, ts_lenet_sgd, tr_ac_lenet_sgd, ts_ac_lenet_sgd = train(
     model, train\_loader, test\_loader, \ epochs=10, \ plot=True, \ verbose=True, \ loss\_fn=loss, \ optim=optimizer
Epoch 9... (Train/Test) Loss: 0.679/0.683
                                                   Accuracy: 0.582/0.563
                                                                      Train accuracy
   0.692
                                                                       Test accuracy
                                                         0.58
   0.690
                                                      Accuracy
P5'0
P5'0
   0.688
0.686
   0.684
   0.682
                                                         0.52
                 Train
   0.680
                 Test
   0.678
                           Epochs
                                                                                Epochs
100%| 100%| 10/10 [02:05<00:00, 12.57s/it]
```

Рис. 5 — LSTM Adagrad

```
model = SimpleGRU(VOCAB_SIZE, EMBEDDING_DIM, HIDDEN_DIM, OUTPUT_DIM, PAD_IDX)
        model.to(device)
         lr = 0.001
         loss=nn.CrossEntropyLoss()
        optimizer = optim.Adam(model.parameters(), lr = lr)
         {\tt tr\_lenet\_sgd,\ ts\_lenet\_sgd,\ tr\_ac\_lenet\_sgd,\ ts\_ac\_lenet\_sgd\ =\ train(}
                          \verb|model|, \verb|train_loader|, test_loader|, epochs=10, plot=True, verbose=True, loss_fn=loss, optim=optimizer| | test_loader|, test_loader|, epochs=10, plot=True, verbose=True, loss_fn=loss, optim=optimizer|, epochs=10, plot=True, ep
    Epoch 9... (Train/Test) Loss: 0.012/0.720
                                                                                                                                                                                                                             Accuracy: 0.996/0.867
                                                                                                                                                                                                                                                         1.0
               0.7
               0.6
                                                                                                                                                                                                                                                          0.9
                0.5
                                                                                                                                                                                                                                             Accuracy
o.0
    SSOT 0.4
               0.3
               0.2
                                                                                                                                                                                                                                                          0.6
                                                                                                                                                                                                                                                                                                                                                                                        Train accuracy
               0.1
                                                                    Train
                                                                     Test
                                                                                                                                                                                                                                                                                                                                                                                         Test accuracy
                                                                                                                                                                                                                                                          0.5
                 0.0
                                                                                                              Epochs
                                                                                                                                                                                                                                                                                                                                                        Epochs
100%| 100%| 10/10 [01:55<00:00, 11.58s/it]
```

Рис. 6 — GRU Adam

```
model = SimpleGRU(VOCAB_SIZE, EMBEDDING_DIM, HIDDEN_DIM, OUTPUT_DIM, PAD_IDX)
 model.to(device)
 lr = 0.0001
 loss=nn.CrossEntropyLoss()
 optimizer = optim.Adagrad(model.parameters(),lr = lr)
 tr_lenet_sgd, ts_lenet_sgd, tr_ac_lenet_sgd, ts_ac_lenet_sgd = train(
     model,train_loader,test_loader, epochs=10, plot=True, verbose=True, loss_fn=loss, optim=optimizer
Epoch 9... (Train/Test) Loss: 0.193/0.412
                                                 Accuracy: 0.930/0.863
                                          Train
                                                       0.9
                                          Test
   0.6
                                                       0.8
   0.5
                                                    Accuracy
                                                       0.6
   0.3
                                                                                   Train accuracy
                                                                                   Test accuracy
   0.2
                        Epochs
                                                                            Epochs
100%| 100%| 10/10 [01:55<00:00, 11.56s/it]
```

Рис. 7 — GRU Adagrad

5 Выводы

В результе выполнения данной лабораторной работы были реализованы различные архитектуры рекурентных нейронных сетей с помощью библиотеки руtorch. Реализованные архитектуры были протестированы на различных открытых датасетах с использованием различных оптимизаторов.