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)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

Домашняя работа №2

по курсу «Теория искусственных нейронных сетей»

«Разработка многослойного персептрона»

Студент группы ИУ9-71Б Окутин Д. А.

Преподаватель Каганов Ю. Т.

Москва 2024

1 Цель

Цель данной лабораторной работы: Изучение многослойного персептрона, исследование его работы на основе использования различных методов оптимизации и целевых функций.

2 Задание

1. Реализовать на языке высокого уровня многослойный персептрон и проверить его работоспособность на примере данных, выбранных из MNIST dataset.

2. Исследовать работу персептрона на основе использования различных целевых функций. (среднеквадратичная ошибка, перекрестная энтропия, дивергенция Кульбака-Лейблера).

3. Исследовать работу многослойного персептрона с использованием различных методов оптимизации (градиентный, Флетчера-Ривза (FR)).

4. Провести исследование эффективности работы многослойного персептрона при изменении гиперпараметров (количества нейронов и количества слоев).

5. Подготовить отчет с распечаткой текста программы, графиками результатов исследования и анализом результатов.

3 Реализация

Исходный код представлен в листинге 1 - 5.

Листинг 1: Импорт используемых библиотек и скачивание датасета MNIST

```
1
2  from torchvision.datasets import MNIST
3  from torch.utils.data import DataLoader
4  from matplotlib import pyplot as plt
5  import numpy as np
6  from IPython.display import clear_output
7  import sys
8
9  transform = lambda img: np.array(np.asarray(img).flatten())/256
10 train_dataset = MNIST('.', train=True, download=True, transform=
     transform)
```

```
11  test_dataset = MNIST('.', train=False, transform=transform)
12
13  train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
14  test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
15
16  X, y = next(iter(train_loader))
17  X = X.numpy()
18  y = y.numpy()
19
20  plt.figure(figsize=(6, 7))
21  for i in range(25):
22      plt.subplot(5, 5, i+1)
23      plt.imshow(X[i].reshape(28, 28), cmap=plt.cm.Greys_r)
24      plt.title(y[i])
25      plt.axis('off')
```

Листинг 2: Функция train для нейросети

```
1
2   from tqdm import tqdm
3   def train(network,train_loader, test_loader, epochs, learning_rate,
      plot=True,
4             verbose=True, loss=None, optimizer='GD'):
5       loss = loss or MSELoss()
6       train_loss_epochs = []
7       test_loss_epochs = []
8       train_accuracy_epochs = []
9       test_accuracy_epochs = []
10      grad_pre = None
11      b = 0
12      try:
13          for epoch in tqdm(range(epochs)):
14              losses = []
15              accuracies = []
16              for X, y in train_loader:
17                  X = X.view(X.shape[0], -1).numpy()
18                  y = y.numpy()
19                  prediction = network.forward(X)
20                  loss_batch = loss.forward(prediction, y)
21                  losses.append(loss_batch)
22                  dLdx = loss.backward()
23
24                  if optimizer=='GD':
25                      dLdx = dLdx
26                  elif optimizer=='FR':
27                      if grad_pre is None:
28                          grad_pre = dLdx
```

```
29                      else:
30                          dLdx_flat = dLdx.flatten()
31                          grad_pre_flat = grad_pre.flatten()
32                          b = (np.sum(dLdx_flat*dLdx_flat)/np.sum(
    grad_pre_flat*grad_pre_flat))**2
33                          b = max(min(b, 1), 0)
34                      grad_pre = dLdx
35                      dLdx = (dLdx + b*grad_pre)
36
37                  network.backward(dLdx)
38                  network.step(learning_rate)
39                  accuracies.append((np.argmax(prediction, 1)==y).mean()
    )
40              train_loss_epochs.append(np.mean(losses))
41              train_accuracy_epochs.append(np.mean(accuracies))
42              losses = []
43              accuracies = []
44              for X, y in test_loader:
45                  X = X.view(X.shape[0], -1).numpy()
46                  y = y.numpy()
47                  prediction = network.forward(X)
48                  loss_batch = loss.forward(prediction, y)
49                  losses.append(loss_batch)
50                  accuracies.append((np.argmax(prediction, 1)==y).mean()
    )
51              test_loss_epochs.append(np.mean(losses))
52              test_accuracy_epochs.append(np.mean(accuracies))
53              clear_output(True)
54              if verbose:
55                  sys.stdout.write('\rEpoch {0}... (Train/Test) Loss:
    {1:.3f}/{2:.3f}\tAccuracy: {3:.3f}/{4:.3f}'.format(
56                              epoch, train_loss_epochs[-1],
    test_loss_epochs[-1],
57                              train_accuracy_epochs[-1],
    test_accuracy_epochs[-1]))
58              if plot:
59                  plt.figure(figsize=(12, 5))
60                  plt.subplot(1, 2, 1)
61                  plt.plot(train_loss_epochs, label='Train')
62                  plt.plot(test_loss_epochs, label='Test')
63                  plt.xlabel('Epochs', fontsize=16)
64                  plt.ylabel('Loss', fontsize=16)
65                  plt.legend(loc=0, fontsize=16)
66                  plt.grid('on')
67                  plt.subplot(1, 2, 2)
```

```
68                  plt.plot(train_accuracy_epochs, label='Train accuracy
    ')
69                  plt.plot(test_accuracy_epochs, label='Test accuracy')
70                  plt.xlabel('Epochs', fontsize=16)
71                  plt.ylabel('Accuracy', fontsize=16)
72                  plt.legend(loc=0, fontsize=16)
73                  plt.grid('on')
74                  plt.show()
75      except KeyboardInterrupt:
76          pass
77      return train_loss_epochs, \
78             test_loss_epochs, \
79             train_accuracy_epochs, \
80             test_accuracy_epochs
81
82
```

Листинг 3: Функции активации

```
1
2   class ReLU:
3     def __init__(self):
4         pass
5
6     def forward(self, X):
7         self.X = X
8         return np.maximum(X, 0)
9
10    def backward(self, dLdy):
11        return (self.X > 0) * dLdy
12
13    def step(self, learning_rate):
14        pass
15
16  def softmax(z):
17    exp = np.exp(z - np.max(z))
18
19    for i in range(len(z)):
20        exp[i] /= np.sum(exp[i])
21
22    return exp
23
24  class Softmax:
25    def __init__(self):
26        pass
27
28    def forward(self, inputs):
```

```
        self.inputs = inputs
        return softmax(inputs)

    def backward(self, dLdy):
        return dLdy

    def step(self, learning_rate):
        pass


```

Листинг 4: Лосс функции

```

class CELoss:
    def __init__(self):
        pass

    def forward(self, X, y):
        self.p = np.exp(X - np.max(X, axis=1, keepdims=True))
        self.p /= self.p.sum(axis=1, keepdims=True)

        self.y = np.zeros((X.shape[0], X.shape[1]))
        self.y[np.arange(X.shape[0]), y] = 1

        return -np.mean(np.sum(self.y * np.log(self.p + 1e-15), axis=1))

    def backward(self):
        return (self.p - self.y) / self.y.shape[0]

class MSELoss:
    def __init__(self):
        pass

    def forward(self, X, y):
        self.X = X

        self.y = np.zeros((X.shape[0], X.shape[1]))
        self.y[np.arange(X.shape[0]), y] = 1

        return np.mean(np.square(self.X - self.y))

    def backward(self):
        return 2 * (self.X - self.y) / self.y.shape[0]


class KLLoss:
```

```
35      def __init__(self):
36          pass
37
38      def forward(self, pred, target):
39          self.pred = np.clip(pred, 1e-10, 1 - 1e-10)
40
41          self.target = np.zeros((pred.shape[0], pred.shape[1]))
42          self.target[np.arange(pred.shape[0]), target] = 1
43
44          self.target = np.clip(self.target, 1e-10, 1 - 1e-10)
45
46          return np.mean(self.target * np.log(self.target/self.pred))
47
48      def backward(self):
49          self.pred = np.clip(self.pred, 1e-8, 1 - 1e-8)
50          res = self.pred - self.target
51          return res / np.linalg.norm(res)
52
53
```

Листинг 5: Структура Нейронной сети

```
1
2       class Linear:
3       def __init__(self, input_size, output_size):
4           self.W = np.random.randn(input_size, output_size)*0.01
5           self.b = np.zeros(output_size)
6
7       def forward(self, X):
8           self.X = X
9           return X.dot(self.W)+self.b
10
11      def backward(self, dLdy):
12          self.dLdW = self.X.T.dot(dLdy)
13          self.dLdb = dLdy.sum(0)
14          self.dLdx = dLdy.dot(self.W.T)
15          return self.dLdx
16
17      def step(self, learning_rate):
18          self.W = self.W - learning_rate * self.dLdW
19          self.b = self.b - learning_rate * self.dLdb
20
21
22  class NeuralNetwork:
23      def __init__(self, modules):
24          self.modules = modules
25
```

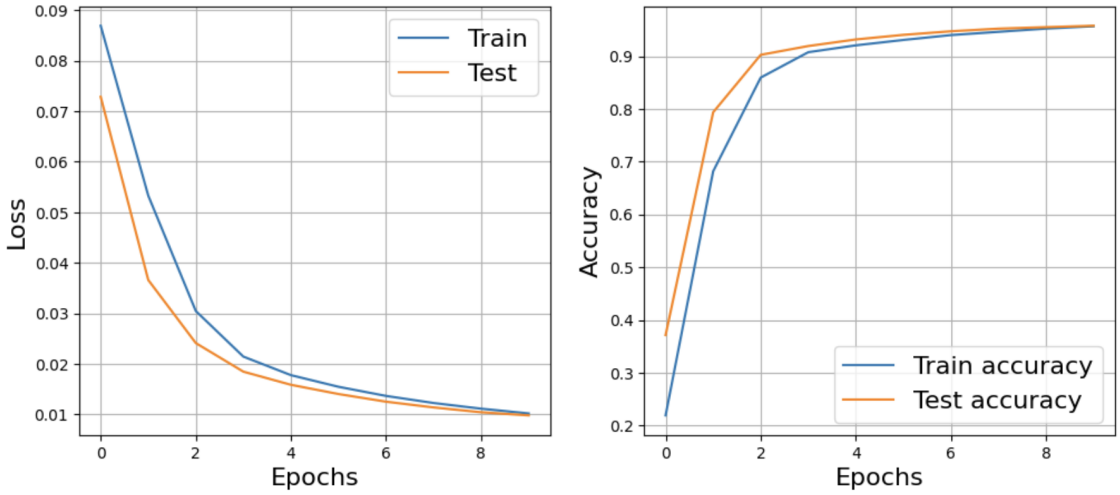
```
26      def forward(self, X):
27          y = X
28          for i in range(len(self.modules)):
29              y = self.modules[i].forward(y)
30          return y
31
32      def backward(self, dLdy):
33          for i in range(len(self.modules))[::-1]:
34              dLdy = self.modules[i].backward(dLdy)
35
36      def step(self, learning_rate):
37          for i in range(len(self.modules)):
38              self.modules[i].step(learning_rate)
39
40
```

4 Результаты

Результаты представлен на рисунках 1 - 2.

```
network = NeuralNetwork([
    Linear(784, 100), ReLU(),
    Linear(100, 100), ReLU(),
    Linear(100, 10)
])
loss = MSELoss()
tr_mse_gd, ts_mse_gd, tr_ac_mse_gd, ts_ac_mse_gd = train(
    network,train_loader,test_loader, 10, 0.01, plot=True, verbose=True, loss=loss, optimizer='GD')
```

```
Epoch 9... (Train/Test) Loss: 0.010/0.010       Accuracy: 0.957/0.958
```



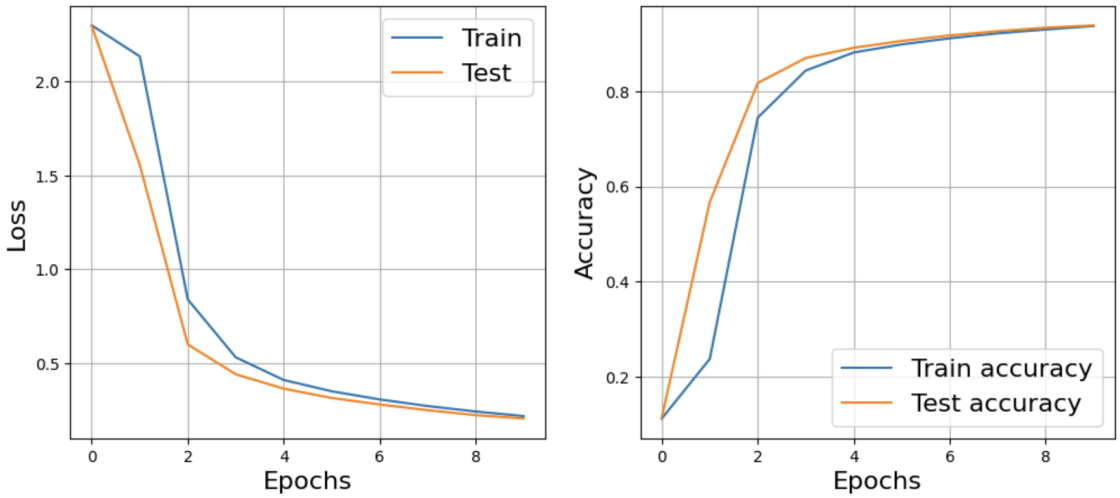

```
100%|██████████| 10/10 [03:02<00:00, 18.23s/it]
```

Рис. 1 — Графики для MSE Loss + GD

```
network = NeuralNetwork([
    Linear(784, 100), ReLU(),
    Linear(100, 100), ReLU(),
    Linear(100, 10)
])
loss = CELoss()
tr_ce_gd, ts_ce_gd, tr_ac_ce_gd, ts_ac_ce_gd = train(
    network,train_loader,test_loader, 10, 0.01, plot=True, verbose=True,loss=loss,optimizer='GD')
```

Notebook editor cells

```
Epoch 9... (Train/Test) Loss: 0.217/0.204       Accuracy: 0.938/0.939
```

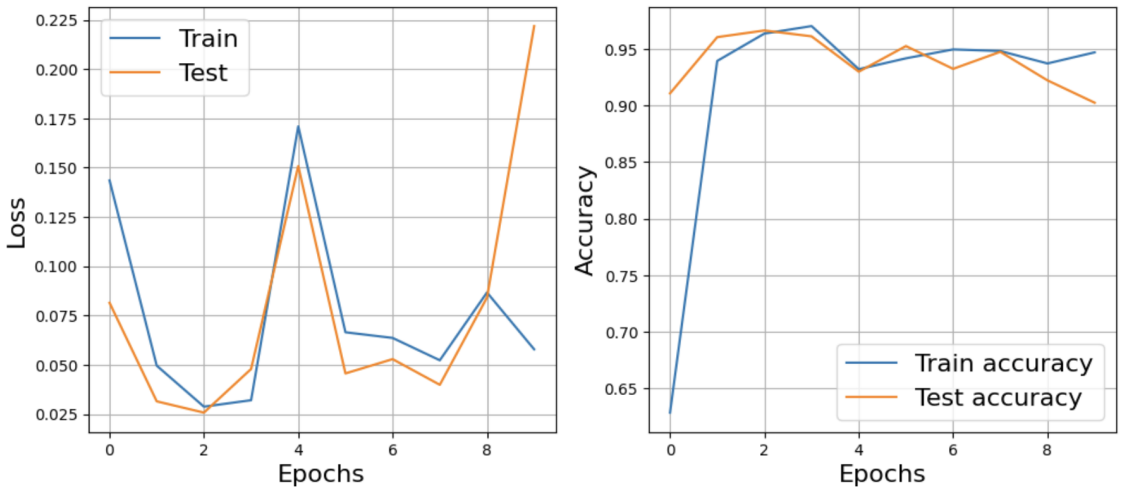



```
100%|██████████| 10/10 [03:01<00:00, 18.17s/it]
```

Рис. 2 — Графики для CE Loss + GD

```
network = NeuralNetwork([
    Linear(784, 100), ReLU(),
    Linear(100, 100), ReLU(),
    Linear(100, 10)
])
loss = KLLoss()
tr_kll_gd, ts_kll_gd, tr_ac_kll_gd, ts_ac_kll_gd = train(
    network,train_loader,test_loader, 10, 0.01, plot=True, verbose=True, loss=loss,optimizer='GD')
```

```
Epoch 9... (Train/Test) Loss: 0.058/0.222       Accuracy: 0.947/0.903
```




```
100%|██████████| 10/10 [03:04<00:00, 18.42s/it]
```

Рис. 3 — Графики для KLL Loss + GD

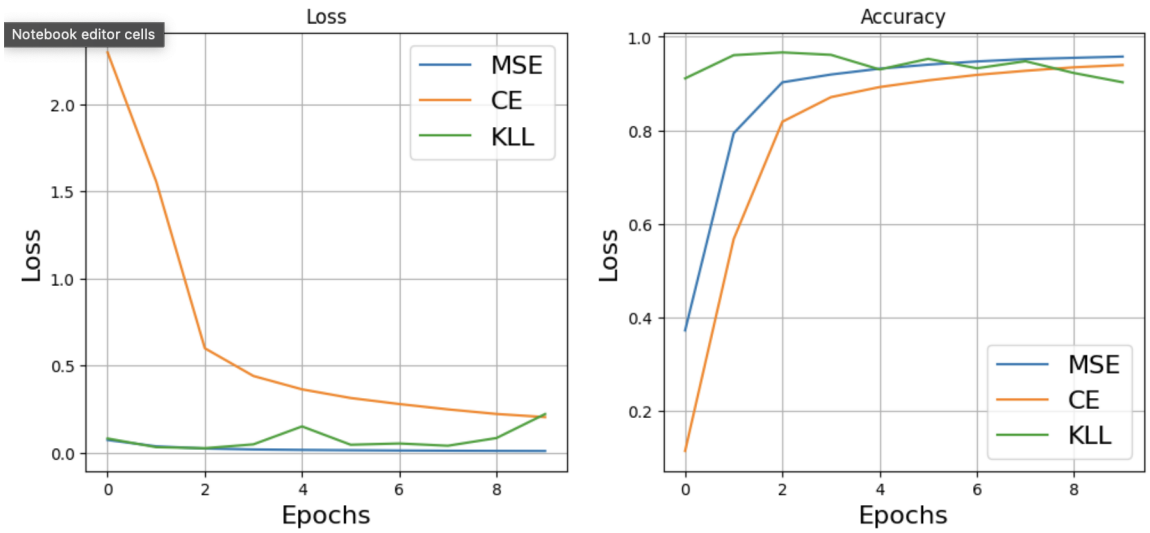


Рис. 4 — Графики сравнения MSE, CE, KLL

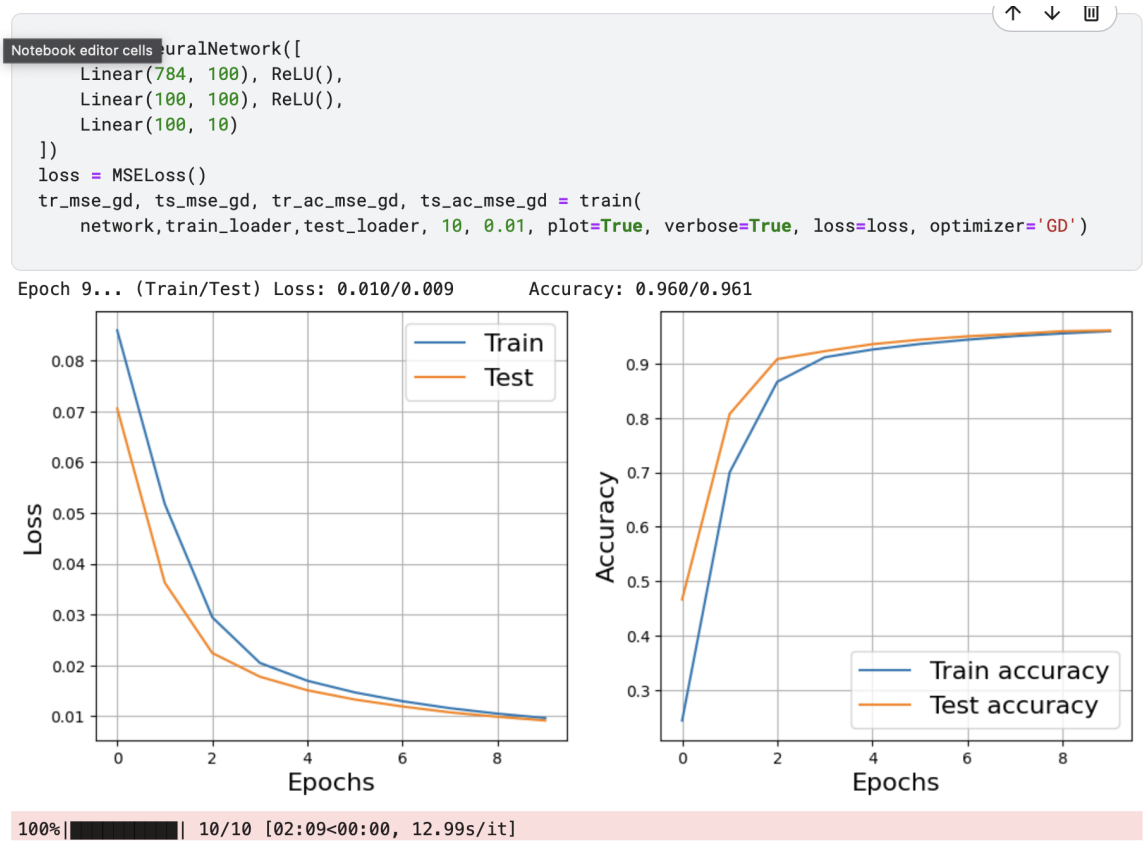
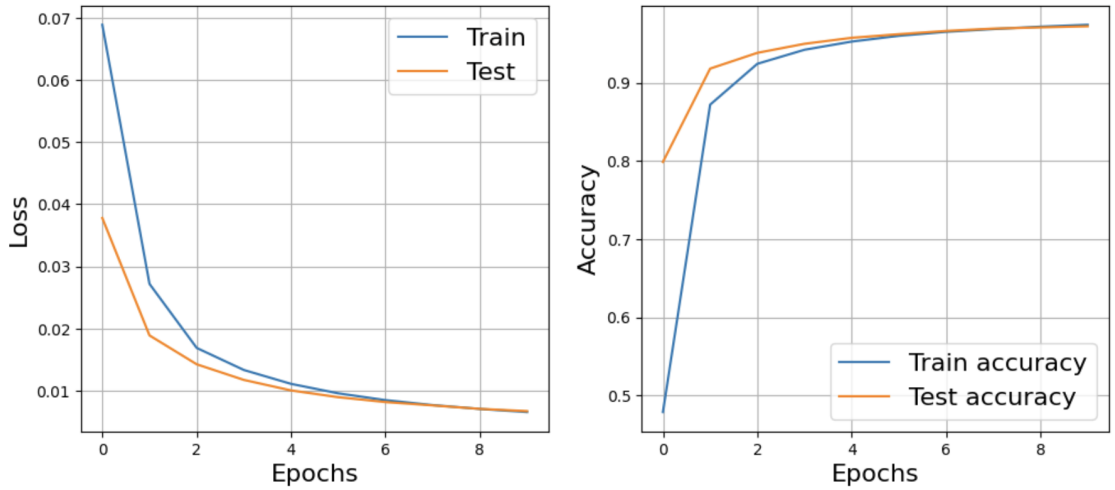


Рис. 5 — Графики для MSE Loss + GD

```
Notebook editor cells
    euralNetwork([
        Linear(784, 100), ReLU(),
        Linear(100, 100), ReLU(),
        Linear(100, 10)
    ])
    loss = MSELoss()
    tr_mse_fr, ts_mse_fr, tr_ac_mse_fr, ts_ac_mse_fr = train(
        network,train_loader,test_loader, 10, 0.01, plot=True, verbose=True, loss=loss, optimizer='FR')
```

```
Epoch 9... (Train/Test) Loss: 0.007/0.007       Accuracy: 0.974/0.972
```




```
100%|██████████| 10/10 [02:10<00:00, 13.08s/it]
```

Рис. 6 — Графики для MSE Loss + FR

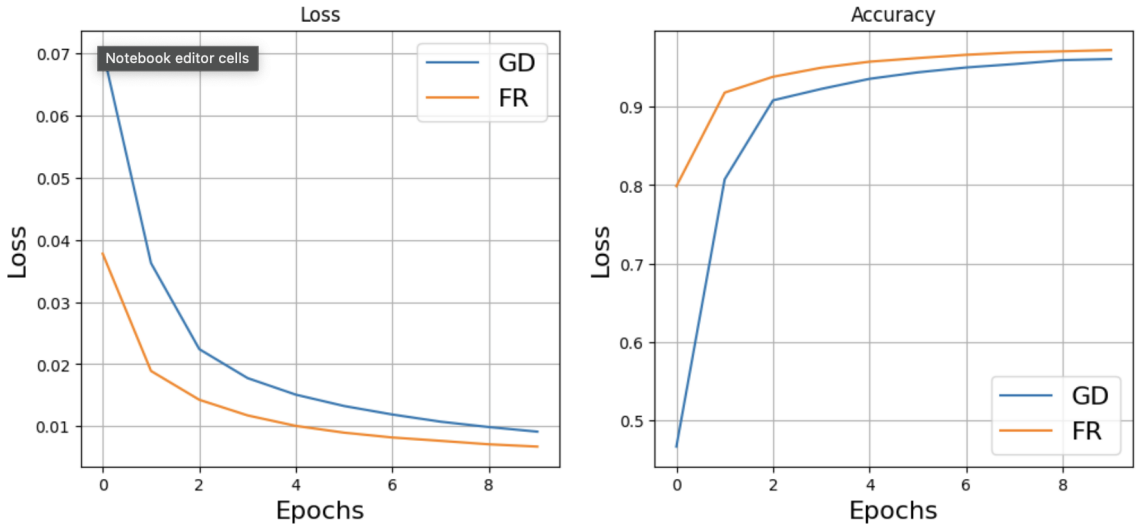
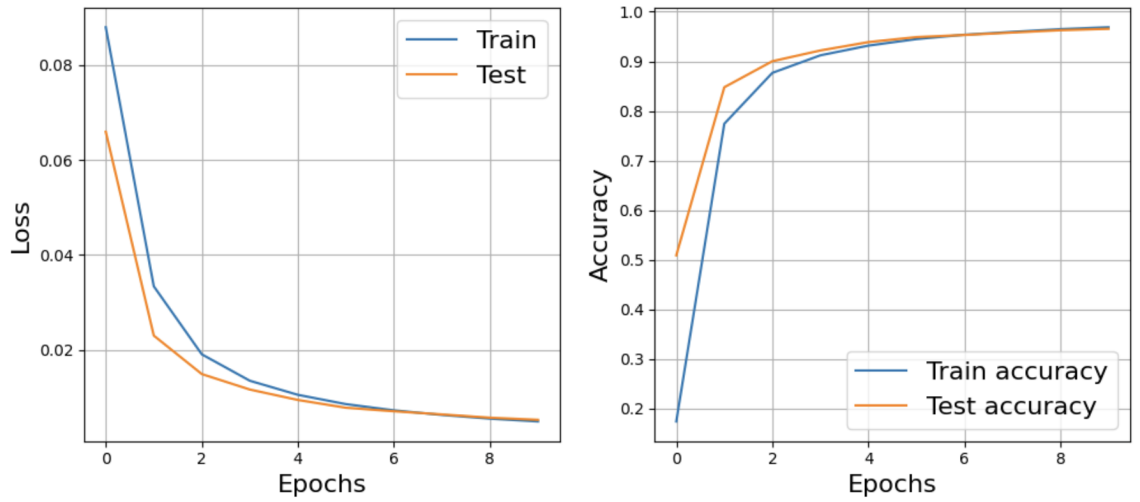


Рис. 7 — Графики сравнения GD и FR

```
network = NeuralNetwork([
    Linear(784, 100), ReLU(),
    Linear(100, 100), ReLU(),
    Linear(100, 10), Softmax()
])
loss = MSELoss()
tr_mse_fr1, ts_mse_fr1, tr_ac_mse_fr1, ts_ac_mse_fr1 = train(
    network,train_loader,test_loader, 10, 0.01, plot=True, verbose=True, loss=loss, optimizer='GD')
```

```
Epoch 9... (Train/Test) Loss: 0.005/0.005       Accuracy: 0.969/0.966
```

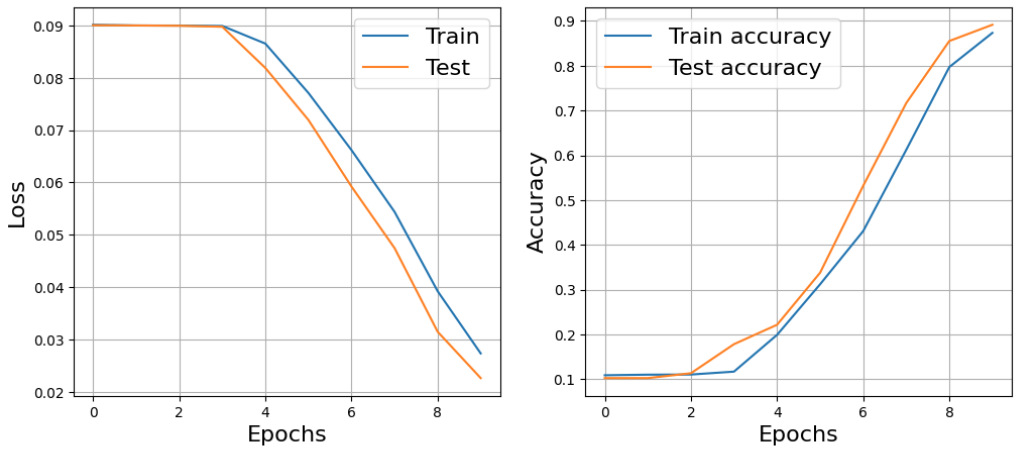



```
100%|██████████| 10/10 [02:21<00:00, 14.11s/it]
```

Рис. 8 — Добавление Softmax

```
network = NeuralNetwork([
    Linear(784, 256), ReLU(),
    Linear(256, 64), ReLU(),
    Linear(64, 64), ReLU(),
    Linear(64, 10),
])
loss = MSELoss()
tr_mse_fr1, ts_mse_fr1, tr_ac_mse_fr1, ts_ac_mse_fr1 = train(
    network,train_loader,test_loader, 10, 0.01, plot=True, verbose=True, loss=loss, optimizer='GD')
```

```
Epoch 9... (Train/Test) Loss: 0.027/0.023       Accuracy: 0.873/0.891
```




```
100%|██████████| 10/10 [03:50<00:00, 23.05s/it]
```

Рис. 9 — Добавление скрытого слоя

5 Выводы

В результате данной лабороторной работы был изучен метод построения архитектуры многослойного персептрона. Были проведены эксперименты с количеством скрытых слоев и размерностью входа/выхода слоя, в результате которых была выбрана архитектура, на которой проводились все дальнейшие действия. 1) Было проведено сравнение различных функций потерь, где наглядно было видно, что, используя KLL Loss нейронная сеть обучается за 3 эпохи, а потом происходит переобучение, остальные функции потерь показали примерно одинаковые результаты. 2) Далее было проведено исследование различных методов оптимизации, для этого была зафиксирована функция потерь, и проведены эксперименты, при которых метод сопряженных градиентов показал себя ожидаемо лучше обычного градиентного спуска. В результате можно сказать, что очень важно подобрать правильное количпество скрытых слоёв модели и их размерность, а также функцию потерь, а уже потом можно эксперементировать с различными оптимизациями.