Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика, искусственный интеллект и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ

по лабораторной работе № 4

по курсу «Основы искусственного интеллекта»

на тему: «Базовая искусственная нейронная сеть»

Студент ИУ7-13М (Группа) _____ (Подпись, дата) Орду М. А. (И. О. Фамилия)

Преподаватель _____ (Подпись, дата) Строганов Ю. В. (И. О. Фамилия)

2025 г.

СОДЕРЖАНИЕ

1 Теоретический раздел 3
1.1 Цель работы 3
1.2 Задачи работы 3
1.3 Нейронные сети и функции активации 3
1.4 Неравенство Чебышева для оценки минимального размера выборки 4

2 Практический раздел 6
2.1 Программная реализация 6
2.2 Результаты 7

ПРИЛОЖЕНИЕ А 10

1 Теоретический раздел

1.1 Цель работы

Целью данной лабораторной работы является изучение методов классификации изображений с помощью нейронных сетей на примере датасета MNIST, а также исследование влияния архитектуры сети и соотношения обучающей и тестовой выборок на качество обучения.

1.2 Задачи работы

1. Создать нейронные сети с различным количеством скрытых слоев (0, 1, 5) с использованием активации ReLU и функции потерь KL Divergence.

2. Обучить сети на различных долях обучающей выборки (10%, 20%, ..., 90%) и оценить точность на обучающей и тестовой выборках.

3. Определить состояния переобучения и недообучения.

4. Рассчитать минимальный размер обучающей выборки с помощью неравенства Чебышёва.

1.3 Нейронные сети и функции активации

Нейронная сеть представляет собой последовательность слоёв, каждый из которых выполняет линейное преобразование и нелинейную активацию. Для скрытых слоёв использовалась активация ReLU (*англ. Rectified Linear Unit*), график которой представлен на рисунке 1.1:

$$f(x) = \max(0, x) \tag{1.1}$$

На выходном слое применялся LogSoftmax для использования с функцией потерь KL Divergence:

$$\text{LogSoftmax}(z_i) = \log \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{1.2}$$

Функция потерь KL Divergence между предсказанным распределени-

ем q и истинным распределением p записывается как:

$$D_{KL}(p \parallel q) = \sum_i p_i \log \frac{p_i}{q_i} \tag{1.3}$$

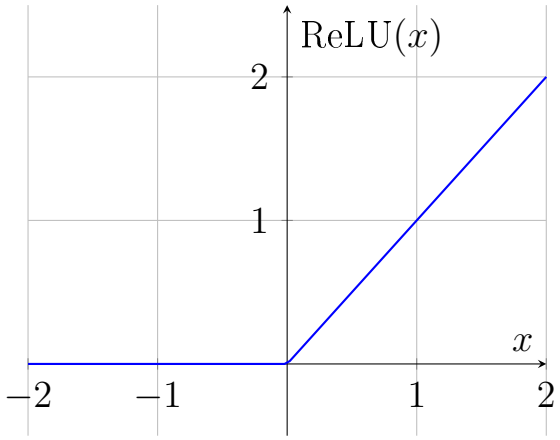

Рисунок 1.1 – График функции ReLU

1.4 Неравенство Чебышева для оценки минимального размера выборки

Для случайной величины X с математическим ожиданием μ и дисперсией σ^2 справедливо неравенство Чебышева:

$$P(|\bar{X} - \mu| \geq \epsilon) \leq \frac{\sigma^2}{N\epsilon^2} \tag{1.4}$$

где $\bar{X}$ — среднее по N наблюдениям, ϵ — допустимая ошибка.

При оценке минимального размера обучающей выборки для задачи классификации удобно использовать дисперсию Бернулли:

$$\sigma^2 = p(1-p) \tag{1.5}$$

где p — ожидаемая точность модели на выборке; $(1-p)$ — вероятность ошибки классификации.

Тогда неравенство Чебышева примет вид:

$$P(|\bar{X} - p| \geq \epsilon) \leq \frac{p(1-p)}{N\epsilon^2} \tag{1.6}$$

Из него выводим минимальный размер выборки:

$$N \geq \frac{p(1-p)}{\delta\,\epsilon^2} \tag{1.7}$$

где $\delta = 1 - P$ — вероятность того, что отклонение превысит ϵ.

2 Практический раздел

2.1 Программная реализация

Программная реализация выполнялась на языке Python с использованием библиотеки PyTorch.

Для создания нейронной сети необходимо задать класс, внутри которого прописаны каждый из слоев сети, как представлено в листинге 2.1.

Листинг 2.1 – Создание класса нейронной сети с одним скрытым слоем

```
class Net1(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Flatten(),
            nn.Linear(784, 128),
            nn.ReLU(),
            nn.Linear(128, 10),
            nn.LogSoftmax(dim=1)
        )
    def forward(self, x):
        return self.model(x)
```

Для создания нейронной сети с пятью скрытыми слоями аналогично создается отдельный класс, как представлено на рисунке 2.2.

Листинг 2.2 – Создание класса нейронной сети с пятью скрытыми слоями

```
class Net5(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Flatten(),
            nn.Linear(784, 256),
            nn.ReLU(),
            nn.Linear(256, 128),
            nn.ReLU(),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, 32),
            nn.ReLU(),
            nn.Linear(32, 16),
            nn.ReLU(),
```

```
            nn.Linear(16, 10),
            nn.LogSoftmax(dim=1)
        )
    def forward(self, x):
        return self.model(x)
```

Для задания KL Divergence в качестве функции потерь, использовался встроенный в библиотеку PyTorch класс KLDivLoss, как представлено на рисунке 2.3.

Листинг 2.3 – Задание KL Divergence в качестве функции потерь

```
loss_fn = nn.KLDivLoss(reduction="batchmean")
```

2.2 Результаты

Количество эпох обучения моделей нейронный сетей было выбрано равным пяти. Зависимость точности обученной модели нейронной сети от доли обучающей выборки представлен на рисунке 2.1.

Как видно из рисунка, при увеличении доли обучающей выборки точность модели растёт. Сети с большим количеством скрытых слоёв демонстрируют более высокую точность на обоих выборках. Причем разница между моделью без скрытых слоев и моделями с хотя бы одним скрытым слоем более существенная, чем разница между моделями с одним и пятью скрытыми слоями - разница между ними на тестовой выборке получилось несущественной.

Для оценки того, какое минимальное количество обучающих примеров необходимо для гарантированного достижения заданной точности, было использовано неравенство Чебышёва, приведенное в отчете ранее.

В рамках эксперимента были использованы значения для модели с наибольшим процентом точности на тестовой выборке (5 скрытых слоев, 80% обучающей выборки):

- точность обученной модели на тестовой выборке: $p = 0.9738$,
- допустимое отклонение: $\epsilon = 0.05$,
- требуемая вероятность достижения точности: $P = 0.95$, следовательно $\delta = 0.05$.

Подставляя значения, получаем:

$$N \geq \frac{0.9738 \cdot (1 - 0.9738)}{0.05 \cdot 0.05^2} \approx 204 \tag{2.1}$$

Таким образом, для гарантии того, что средняя точность модели отклонится от истинной не более чем на $\epsilon = 0.05$ с вероятностью $P = 0.95$, достаточно иметь около:

$$N \approx 204 \text{ обучающих примера} \tag{2.2}$$

Полученное значение означает, что даже относительно небольшая обучающая выборка с высокой вероятностью позволяет получить среднюю точность, отличающуюся от истинной не более чем на 5%.

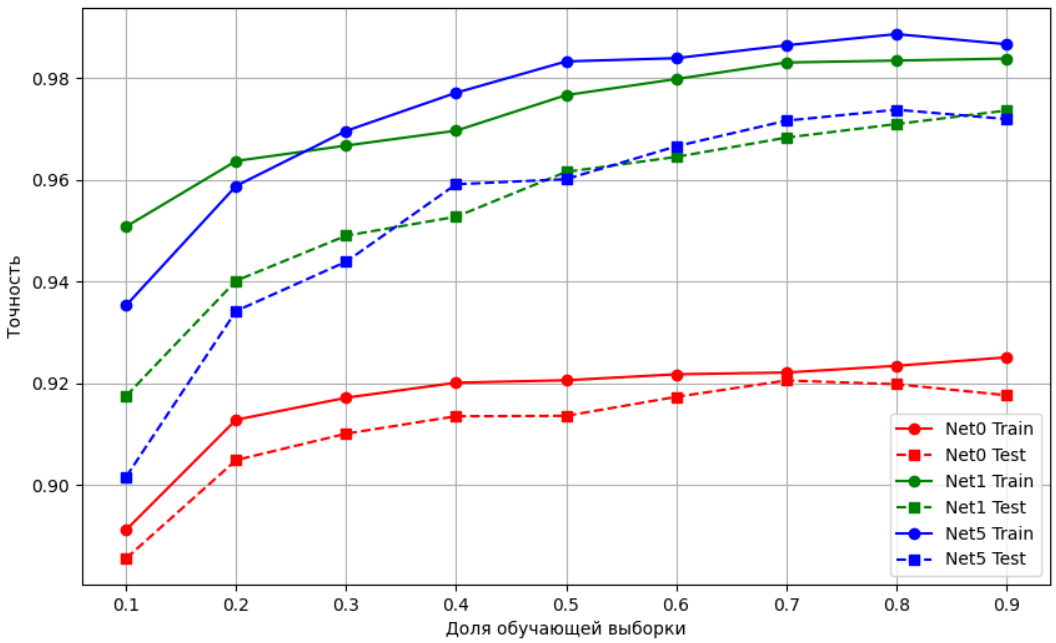

Рисунок 2.1 – Зависимость точности моделей от доли обучающей выборки

ВЫВОДЫ

1. Архитектура сети и количество скрытых слоёв напрямую влияют на точность классификации и риск переобучения.

2. Увеличение доли обучающей выборки повышает точность модели.

3. Полученное по неравенству Чебышёва значение минимального количества обучающих выборок на порядки меньше количества выборок в базе данных MNIST (около 60000). Переобучение модели практически невозможно.

ПРИЛОЖЕНИЕ А

Листинг А.1 – Исходный код программы

```
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, random_split
from torchvision import datasets, transforms
import matplotlib.pyplot as plt

# ------------------------------

transform = transforms.ToTensor()
full_data = datasets.MNIST(root='data', train=True,
    transform=transform, download=True)

# ------------------------------

class Net0(nn.Module):  # без скрытых слоёв
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Flatten(),
            nn.Linear(784, 10),
            nn.LogSoftmax(dim=1)
        )
    def forward(self, x):
        return self.model(x)

class Net1(nn.Module):  # 1 скрытый слой
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Flatten(),
            nn.Linear(784, 128),
            nn.ReLU(),
            nn.Linear(128, 10),
            nn.LogSoftmax(dim=1)
        )
    def forward(self, x):
        return self.model(x)
```

```
37
38 class Net5(nn.Module):  # 5 скрытых слоёв
39     def __init__(self):
40         super().__init__()
41         self.model = nn.Sequential(
42             nn.Flatten(),
43             nn.Linear(784, 256),
44             nn.ReLU(),
45             nn.Linear(256, 128),
46             nn.ReLU(),
47             nn.Linear(128, 64),
48             nn.ReLU(),
49             nn.Linear(64, 32),
50             nn.ReLU(),
51             nn.Linear(32, 16),
52             nn.ReLU(),
53             nn.Linear(16, 10),
54             nn.LogSoftmax(dim=1)
55         )
56     def forward(self, x):
57         return self.model(x)
58
59 # ------------------------------
60
61 def to_one_hot(y, num_classes=10):
62     y_onehot = torch.zeros(len(y), num_classes)
63     y_onehot[torch.arange(len(y)), y] = 1
64     return y_onehot
65
66 def train_epoch(model, train_loader, optimizer, loss_fn):
67     model.train()
68     total_loss = 0
69     correct = 0
70     total = 0
71     for x, y in train_loader:
72         y_onehot = to_one_hot(y)
73         optimizer.zero_grad()
74         preds = model(x)
75         loss = loss_fn(preds, y_onehot)
76         loss.backward()
77         optimizer.step()
```

```
78          total_loss += loss.item()
79          predicted = preds.argmax(dim=1)
80          correct += (predicted == y).sum().item()
81          total += y.size(0)
82      return total_loss, correct/total
83
84  def evaluate(model, loader):
85      model.eval()
86      correct = 0
87      total = 0
88      with torch.no_grad():
89          for x, y in loader:
90              preds = model(x)
91              predicted = preds.argmax(dim=1)
92              correct += (predicted == y).sum().item()
93              total += y.size(0)
94      return correct/total
95
96  def split_dataset(dataset, train_ratio):
97      train_size = int(len(dataset) * train_ratio)
98      test_size = len(dataset) - train_size
99      return random_split(dataset, [train_size, test_size])
100
101 def required_sample_size(p, epsilon, confidence):
102     """
103     p - ожидаемая точность (accuracy)
104     epsilon - допустимая ошибка
105     confidence - вероятность успешного выполнения (например,
            0.95)
106     """
107     delta = 1 - confidence
108     sigma2 = p * (1 - p)
109     N = sigma2 / (delta * epsilon**2)
110     return int(N)
111
112 # ------------------------------
113
114 splits = [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]
115 models = {'Net0': Net0, 'Net1': Net1, 'Net5': Net5}
116 colors = {'Net0':'r', 'Net1':'g', 'Net5':'b'}
117
```

```
118 plt.figure(figsize=(10,6))
119
120 # Параметры для Чебышёва
121 epsilon = 0.05
122 confidence = 0.95
123
124 for name, model_class in models.items():
125     train_accuracies = []
126     test_accuracies = []
127     chebyshev_sizes = []
128
129     for train_ratio in splits:
130         train_set, test_set = split_dataset(full_data,
              train_ratio)
131         train_loader = DataLoader(train_set, batch_size=64,
              shuffle=True)
132         test_loader  = DataLoader(test_set, batch_size=64)
133
134         model = model_class()
135         optimizer = torch.optim.Adam(model.parameters(),
              lr=0.001)
136         loss_fn = nn.KLDivLoss(reduction="batchmean")
137
138         for epoch in range(5):
139             train_epoch(model, train_loader, optimizer,
                  loss_fn)
140
141         train_acc = evaluate(model, train_loader)
142         test_acc  = evaluate(model, test_loader)
143         train_accuracies.append(train_acc)
144         test_accuracies.append(test_acc)
145
146         N_needed = required_sample_size(train_acc, epsilon,
              confidence)
147         chebyshev_sizes.append(N_needed)
148
149         print(f"{name} | train_ratio={train_ratio} |
              train_acc={train_acc:.4f} | test_acc={test_acc:.4f}
              | N_needed={N_needed}")
150
151     plt.plot(splits, train_accuracies, marker='o',
```

```
        color=colors[name], linestyle='-', label=f'{name}
        Train')
    plt.plot(splits, test_accuracies, marker='s',
        color=colors[name], linestyle='--', label=f'{name}
        Test')

plt.xlabel('Доля обучающей выборки')
plt.ylabel('Accuracy')
plt.title('Train/Test Accuracy vs Доля обучающей выборки')
plt.grid(True)
plt.legend()
plt.show()
```