МОСКОВСКИЙ ИНСТИТУТ ЭЛЕКТРОННОЙ ТЕХНИКИ
Институт системной и программной инженерии
и информационных технологий (Институт СПИНТех)

**Лабораторный практикум по курсу**
**"Нейронные сети"**

Лабораторная работа 4.
Распознавание цифр с использованием набора данных MNIST
(Python вариант)

# 1. Основные теоретические сведения

## 1.1. Введение

В данной лабораторной работе ставится задача распознавания рукописных цифр на основе набора данных MNIST.

База данных MNIST (англ. *Modified National Institute of Standards and Technology*) - объёмная база данных образцов рукописного написания цифр. База данных является стандартом, предложенным Национальным институтом стандартов и технологий США (NIST), с целью калибровки и сопоставления методов распознавания изображений с помощью машинного обучения в первую очередь на основе нейронных сетей. Данные состоят из заранее подготовленных примеров изображений, на основе которых проводится обучение и тестирование систем. База данных была создана после переработки оригинального набора чёрно-белых образцов размером 20 x 20 пикселей NIST. Создатели базы данных NIST, в свою очередь, использовали набор образцов из Бюро переписи населения США, к которому были добавлены ещё тестовые образцы, написанные студентами американских университетов. Образцы из набора NIST были нормализированы, прошли сглаживание и приведены к серому полутоновому изображению размером 28x28 пикселей.

Лабораторная работа представлена в двух вариантах, а именно, с использованием двух инструментариев: Matlab и Python. Настоящая версия лабораторной работы содержит сопровождающие скрипты для обучения и оценки модели машинного обучения, подготовленные в среде разработки Python.

## 1.1. Архитектура нейронной сети

Задача распознавания рукописных цифр является классической задачей классификации изображений в области машинного обучения. В рамках данной лабораторной работы предлагается использовать полносвязную нейронную сеть прямого распространения, а именно многослойный перцептрон (англ. Multilayer Perceptron, MLP). При этом не исключается, что в процессе моделирования может быть также применена и другая архитектура, в частности Свёрточная Нейронную Сеть (англ. Convolutional Neural Network, CNN).

## 1.2. Описание данных

Набор данных MNIST состоит из 70 000 изображений размером 28x28 пикселей, каждое из которых представляет собой рукописную цифру от 0 до 9. Изображения представлены в градациях

серого, где 0 - белый цвет, а 255 - черный. Данные разделены на две части: обучающую выборку из 60 000 изображений и тестовую выборку из 10 000 изображений.

### 1.3. Метрики качества модели

Точность, полнота, специфичность и матрица ошибок являются ключевыми метриками оценки производительности модели классификации. Они позволяют измерить, насколько хорошо модель предсказывает истинные метки классов и выявляет типичные ошибки.

1. Точность (англ. precision) определяет долю верно предсказанных положительных примеров среди всех примеров, которые модель определила как положительные.
2. Полнота (англ. recall) определяет долю верно предсказанных положительных примеров из всех примеров, которые являются положительными
3. Специфичность (англ. specificity) определяет долю верно предсказанных отрицательных примеров среди всех примеров, которые являются отрицательными.
4. Доля правильных ответов (англ. accuracy) определяет долю верно предсказанных примеров среди всех примеров. Эта метрика плохо оценивает производительность модели в случае несбалансированности классов.
5. Матрица ошибок (англ. confusion matrix) представляет собой таблицу, которая сопоставляет истинные и предсказанные метки классов

В качестве метрик для оценки модели будем использовать точность и матрицу ошибок.

### 1.4. Полносвязная многослойная нейронно-сетевая сеть модель

В рамках данной лабораторной работы мы будем использовать полносвязную многослойную нейронную сеть (Multilayer Perceptron, MLP) для решения этой задачи.

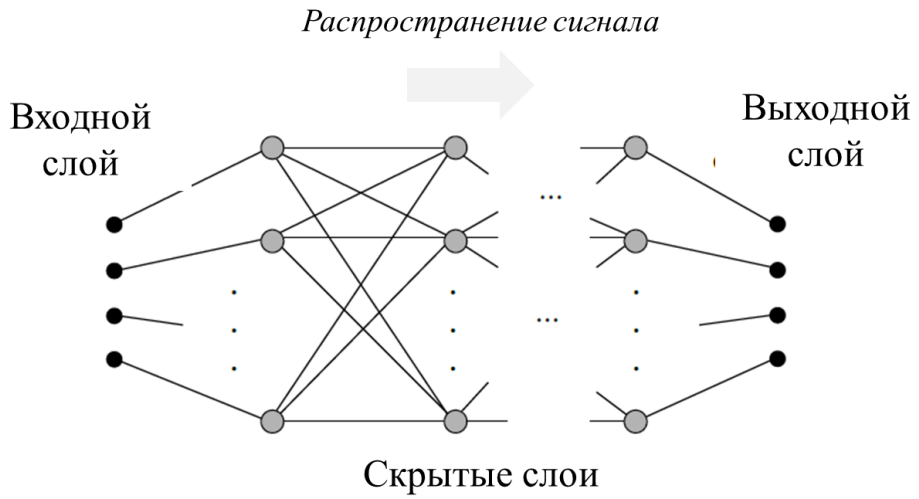



Рис. 1. Общая архитектура многослойной нейронной сети

Многослойный перцептрон - это класс искусственных нейронных сетей, состоящий из нескольких слоёв нейронов, связанных друг с другом. Эта модель была разработана для решения задач классификации и регрессии, таких как распознавание образов, прогнозирование временных рядов и анализ текстов.

Многослойный перцептрон состоит из трёх основных типов слоёв:

- входной слой: представляет собой набор входных переменных (признаков). Количество нейронов на входном слое равно количеству признаков в данных;
- скрытые слои: один или несколько слоёв нейронов, каждый из которых обрабатывает информацию, полученную от предыдущего слоя. Скрытые слои выполняют основную часть вычислений в сети;
- выходной слой: представляет собой набор нейронов, соответствующих классам или значениям, которые необходимо предсказать. Количество нейронов на выходном слое зависит от количества классов (для задач классификации) или равно единице (для задач регрессии).

В многослойном перцептроне каждый нейрон содержит *функцию активации*, которая определяет выходной сигнал данного нейрона. Функции активации могут быть линейными или нелинейными. Нелинейные функции активации позволяют MLP аппроксимировать сложные зависимости в данных. Некоторые популярные функции активации включают сигмоидальную (логистическую), гиперболический тангенс (tanh) и ReLU (англ. Rectified Linear Unit) и т.д. Некоторые из этих функций моделировались в лабораторной работе 1.

Многослойный перцептрон содержит полносвязные слои, где каждый нейрон связан со всеми нейронами предыдущего слоя. Таким образом, веса связей между слоями можно представить в виде матрицы, где каждый элемент матрицы соответствует весу связи между двумя нейронами. Пусть у нас есть два слоя: предыдущий слой с $n$ нейронами и текущий слой с $m$ нейронами. Матрица весов для этих двух слоёв будет иметь размерность $m$ x $n$. В процессе обучения мы подбираем значения элементов этой матрицы таким образом, чтобы минимизировать функцию потерь на обучающей выборке. В дополнение к матрице весов для каждого полносвязного слоя также добавляется вектор смещения (англ. bias), размерность которого равна количеству нейронов в текущем слое ($m$). Вектор смещения также подбирается в процессе обучения и служит для учета смещения активационной функции. Таким образом, полносвязные слои в нейронных сетях могут быть представлены в виде матриц весов и векторов смещения, значения которых подбираются в процессе обучения сети.

Обучение многослойного перцептрона обычно выполняется с использованием алгоритма обратного распространения ошибки (англ. backpropagation) и градиентного спуска. В процессе обучения веса сети настраиваются таким образом, чтобы минимизировать ошибку между предсказанными и истинными значениями на обучающей выборке.

Преимущества многослойного перцептрона заключаются в его универсальности и способности аппроксимировать сложные функции. Однако, для успешного об учения MLP необходимо определить оптимальную архитектуру сети, что может быть непростой задачей. Кроме того, MLP чувствителен к масштабированию признаков и требует предварительной нормализации данных.

Недостатки многослойного перцептрона включают:

- высокую вычислительную сложность: обучение MLP может быть затратным по времени и ресурсам, особенно для больших наборов данных и сложных архитектур сети;

- риск переобучения: если модель имеет слишком много слоев или нейронов, она может стать слишком сложной и подвержена переобучению, что приводит к плохой обобщающей способности;
- локальные минимумы: градиентный спуск в MLP может застрять в локальных минимумах функции потерь, что может привести к неоптимальным решениям.

При использовании многослойного перцептрона важно экспериментировать с архитектурой сети, функциями активации и гиперпараметрами обучения, чтобы найти наилучшую модель для конкретной задачи. В машинном обучении гиперпараметрами называют параметры алгоритмов, значения которых устанавливаются перед запуском процесса обучения. В этом смысле они и отличаются от обычных параметров, вычисляемых в процессе обучения. Гиперпараметры используются для управления процессом обучения. В последние годы разработано множество методов и техник, направленных на улучшение процесса обучения и уменьшение вышеуказанных недостатков, таких как использование методов регуляризации (например, англ. dropout), адаптивных алгоритмов оптимизации (например, англ. Adam) и техник инициализации весов.

### 1.5. Функция потерь

В качестве функции потерь (оптимизируемой функции) мы выберем кросс-энтропию. Кросс-энтропия является мерой различия между двумя вероятностными распределениями и часто используется в качестве функции потерь в задачах классификации машинного обучения, включая нейронные сети. Предполагается, что модель выдает вероятностное распределение для каждого класса. В случае нейронных сетей это обычно достигается с помощью softmax-функции на выходном слое для многоклассовой классификации или сигмоиды для бинарной классификации:

$$L_{CE}(t, p) = -\sum_{i=1}^{C} t_i \log(p_i),$$

где $t$ – это истинное распределение, $p$ – предсказанное распределение, $C$ – количество классов. Функция потерь на основе кросс-энтропии штрафует модель за неверные предсказания и вознаграждает за верные. Когда модель правильно предсказывает класс с высокой уверенностью, значение кросс-энтропии стремится к нулю. Наоборот, если модель дает неверный класс с высокой уверенностью, значение кросс-энтропии возрастает. Кросс энтропия является непрерывной дифференцируемой функцией и позволяет вычислить градиенты для минимизации.

### 1.6. Алгоритм оптимизации

Для поиска значений гиперпараметров сети будем использовать один из широко известных методов оптимизации: стохастический градиентный спуск (англ. Stochastic Gradient Descent, SGD). SGD — это оптимизационный алгоритм, используемый для минимизации функции потерь в задачах машинного обучения, особенно в нейронных сетях. Он является вариацией классического градиентного спуска, который адаптирован для более эффективного обучения на больших наборах данных. Основные принципы стохастического градиентного спуска:

- обновление весов на каждом шаге происходит с использованием только одного случайно выбранного обучающего примера (или небольшого подмножества примеров, называемого мини-пакетом, англ. mini-batch), в отличие от классического градиентного спуска, где градиенты вычисляются на основе всех обучающих примеров;

- в процессе обучения алгоритм делает множество проходов по обучающей выборке (эпох), обновляя веса модели на каждом шаге с учетом градиента функции потерь, вычисленного для текущего обучающего примера или мини-пакета;
- веса обновляются с использованием скорости обучения, которая определяет величину шага в направлении антиградиента функции потерь. Подбор подходящей скорости обучения является важным аспектом для эффективной работы алгоритма.

Преимущества стохастического градиентного спуска:

- вычислительная эффективность: так как SGD использует только один обучающий пример или небольшой мини-пакет на каждом шаге, вычисления становятся более быстрыми и позволяют обучать модели на больших наборах данных;
- улучшенная сходимость: вариативность, связанная со случайным выбором обучающих примеров, может помочь алгоритму "выбираться" из локальных минимумов функции потерь, что может привести к более оптимальным решениям.

Однако стохастический градиентный спуск имеет и некоторые недостатки, такие как более шумные и менее стабильные обновления весов, что может привести к медленной сходимости и необходимости тонкой настройки параметров алгоритма, таких как скорость обучения и размер мини-пакета. Среди недостатков SGD можно перечислить:

- зашумленные обновления весов: так как на каждом шаге используется только один обучающий пример или небольшой мини-пакет, обновления весов могут быть искажёнными и менее стабильными по сравнению с обновлениями, основанными на полном наборе данных;
- сходимость: из-за шумных обновлений весов, сходимость алгоритма может быть медленной и требовать больше эпох для достижения оптимальных значений весов.

Необходимость настройки гиперпараметров: подбор подходящей скорости обучения и размера мини-пакета может быть сложной задачей, так как эти параметры существенно влияют на сходимость и эффективность алгоритма.

В последние годы были разработаны улучшенные варианты стохастического градиентного спуска, такие как AdaGrad, RMSprop, Adam и другие, которые включают адаптивные скорости обучения и моменты для улучшения сходимости и стабильности обучения.

Стохастический градиентный спуск является эффективным и широко используемым алгоритмом оптимизации для обучения моделей машинного обучения, особенно нейронных сетей. Он обеспечивает быстрое обучение на больших наборах данных, однако требует тонкой настройки гиперпараметров и внимания к деталям для достижения оптимальных результатов.

## 2. Выполнение работы

### 2.1. Установка необходимых библиотек

```
pip install opencv-python
pip3 install torch torchvision
pip install -U scikit-learn
```

### 2.2. Импорт необходимых библиотек

```
import torch
import torchvision
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import DataLoader, random_split
from torchvision.transforms import ToTensor
from sklearn.metrics import confusion_matrix, accuracy_score
```

### 2.3. Загрузка данных

```
train_data = torchvision.datasets.MNIST(root='./data', train=True,
transform=ToTensor(), download=True)
test_data = torchvision.datasets.MNIST(root='./data', train=False,
transform=ToTensor(), download=True)
```

### 2.4. Разделение данных на обучающую тестовую и валидационную выборки

Разделение всей выборки на обучающую и валидационную необходимо для оценки обобщающей способности модели и предотвращения переобучения. Это является необходимым шагом в процессе обучения.

1. Оценка обобщающей способности. В процессе обучения модель подстраивается под обучающую выборку, пытаясь наилучшим образом аппроксимировать зависимости между признаками и целевыми переменными. Разделение выборки позволяет оценить, насколько хорошо модель способна предсказывать результаты на новых, ранее не виденных данных, которые представлены тестовой выборкой.
2. Предотвращение переобучения. Переобучение происходит, когда модель слишком сильно подстраивается под обучающую выборку, захватывая шум и специфичные особенности данных, что приводит к плохой обобщающей способности. Разделение выборки на обучающую и валидационную позволяет обнаружить переобучение, сравнивая показатели качества модели на обучающей и тестовой выборках. Если модель показывает хорошие результаты на обучающей выборке, но плохие на тестовой, это может указывать на переобучение.
3. Настройка гиперпараметров и выбор модели. Разделение выборки также может быть использовано для настройки гиперпараметров модели или для выбора наилучшей модели среди нескольких кандидатов. В этих случаях обычно используется дополнительное разделение на тестовую выборку, которая никак не участвует в процессе обучения и валидации модели и служит для финальной оценки обученных моделей. После того, как оптимальные параметры и модель выбраны, итоговая оценка качества производится на тестовой выборке.

```
train_data, val_data = random_split(train_data, [50000, 10000])
train_loader = DataLoader(train_data, batch_size=64, shuffle=True)
val_loader = DataLoader(val_data, batch_size=64, shuffle=False)
test_loader = DataLoader(test_data, batch_size=64, shuffle=False)
```

### 2.5. Описание модели

Модель состоит из оного скрытого слоя fc1 и выходного слоя fc2. Размер входных параметров 28*28, для фрагментов изображений 28 на 28 пикселей в градациях серого. Скрытый

слой имеет размерность 128, а на выходе имеем вероятности принадлежности к одному из 10 классов. Соответственно, модель «внутри» реализована в виде двух матриц, первая – 784 × 128, и вторая - 128 × 10 элементов.

```
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(28*28, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = x.view(-1, 28*28)
        x = F.relu(self.fc1(x))
        x = F.softmax(self.fc2(x), dim=1)
        return x
```

### 2.6. Инициализация модели и определение функций обучения и валидации

```
model = MLP()
optimizer = optim.SGD(model.parameters(), lr=0.01)

def train_epoch(model, data_loader, optimizer, loss_fn):
    model.train()
    for X, Y in data_loader:
        optimizer.zero_grad()
        Y_pred = model(X)
        loss = loss_fn(Y_pred, Y)
        loss.backward()
        optimizer.step()

def evaluate_model(model, data_loader, loss_fn):
    model.eval()
    total_loss = 0
    with torch.no_grad():
        for X, Y in data_loader:
            Y_pred = model(X)
            loss = loss_fn(Y_pred, Y)
            total_loss += loss.item()
    return total_loss / len(data_loader)
```

### 2.7. Обучение

```
num_epochs = 20
loss_fn = nn.CrossEntropyLoss()

for epoch in range(num_epochs):
    train_epoch(model, train_loader, optimizer, loss_fn)
    val_loss = evaluate_model(model, val_loader, loss_fn)
    print(f"Epoch {epoch+1}, Validation Loss: {val_loss:.4f}")
```

Ниже демонстрируется ход процесса обучения нейронной сети.

```
Epoch 1, Validation Loss: 2.2809
Epoch 2, Validation Loss: 2.1983
Epoch 3, Validation Loss: 2.0234
```

```
Epoch 4, Validation Loss: 1.8871
Epoch 5, Validation Loss: 1.8235
Epoch 6, Validation Loss: 1.7884
Epoch 7, Validation Loss: 1.7675
Epoch 8, Validation Loss: 1.7537
Epoch 9, Validation Loss: 1.7277
Epoch 10, Validation Loss: 1.6968
Epoch 11, Validation Loss: 1.6827
Epoch 12, Validation Loss: 1.6616
Epoch 13, Validation Loss: 1.6409
Epoch 14, Validation Loss: 1.6274
Epoch 15, Validation Loss: 1.6177
Epoch 16, Validation Loss: 1.6102
Epoch 17, Validation Loss: 1.6044
Epoch 18, Validation Loss: 1.5998
Epoch 19, Validation Loss: 1.5959
Epoch 20, Validation Loss: 1.5929
```

### 2.8. Оценка полученной модели

```
def predict(model, loader):
    model.eval()
    all_preds = []
    with torch.no_grad():
        for X, _ in loader:
            preds = model(X)
            _, pred_labels = torch.max(preds, 1)
            all_preds.extend(pred_labels.numpy())
    return all_preds

Y_pred = predict(model, test_data)
accuracy = accuracy_score(test_data.targets.numpy(), Y_pred)
cm = confusion_matrix(test_data.targets.numpy(), Y_pred)
print(f"Точность: {accuracy:.4f}")
print("Матрица ошибок:")
print(cm)
```

Точность обучения в примере – 0.9027, а матрица ошибок имеет следующий вид:

```
[[ 963    0    4    2    0    3    5    1    2    0]
 [   0 1107    2    6    0    0    5    2   13    0]
 [  15    3  902   14   16    1   19   21   34    7]
 [   3    0   23  913    1   23    3   16   20    8]
 [   2    3    5    1  904    0   17    2    7   41]
 [  27    6   10   57   26  663   27   16   52    8]
 [  21    3    4    0   10   14  902    0    4    0]
 [   6   17   34    2   11    1    0  926    8   23]
 [   9    8    9   17   12   20   16   14  856   13]
 [  12    7    3   10   40   14    1   20   11  891]]
```

### 2.9. Визуализация

```
import matplotlib.pyplot as plt
import numpy as np

def visualize_predictions(images, labels, preds, num_samples=10):
    idxs = np.random.choice(len(images), size=num_samples, replace=False)
    fig, axes = plt.subplots(1, num_samples, figsize=(15, 3))
    for i, ax in zip(idxs, axes):
        ax.imshow(images[i].numpy().squeeze(), cmap='gray')
```

```
            ax.set_title(f"{preds[i]} (true: {labels[i]})")
            ax.axis('off')
        plt.show()


test_images, test_labels = next(iter(test_loader))
model.eval()
all_preds = []
with torch.no_grad():
        for X in test_images:
            preds = model(X)
            _, pred_labels = torch.max(preds, 1)
            all_preds.extend(pred_labels.numpy())
visualize_predictions(test_images, test_labels.numpy(), all_preds)
```

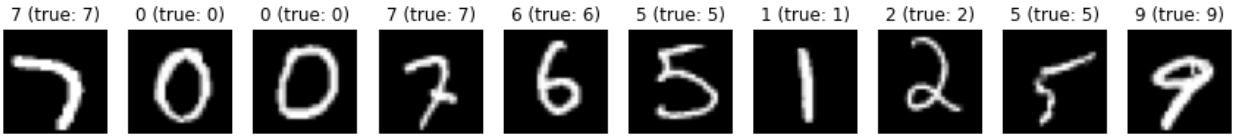



Рис. 2. Распознавание рукописных цифр обученной нейронной сетью

## 3. Задание

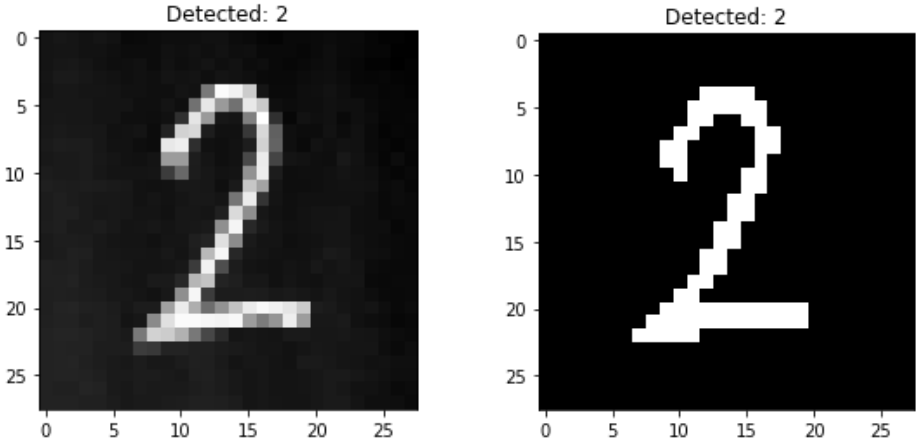
### 3.1. Обработка фотографии рукописного символа

3.1.1. Изобразите произвольный рукописный цифровой символ и сделайте фото.
3.1.2. Примените обученную модель к цифровой версии полученного фото.
3.1.3. Продемонстрируйте распознавание символа обученной нейронной сетью.
3.1.4. Оцените вероятности принадлежности фрагмента к тому или иному классу.

Замечание 1. Для того, чтобы применить обученную модель к собственной фотографии рукописного символа, необходимо произвести предварительную предобработку фотографии. Эта предобработка включает в себя: a) выделение квадратного фрагмента изображения, содержащего цифру; b) нормализацию изображения, т.е. приведение типа изображения к градациям серого и к типу с плавающей точкой (так как обычно изображения состоят из значений беззнакового байтового, одному пикселю цветного изображения соответствует 3 беззнаковых байтовых числа); c) интерполяцию фрагмента изображения в размер, соответствующий размеру входного слоя сети, т. е. 28 × 28 пикселей и инвертирование, то есть преобразование чёрного цвета в белый, а белого цвета – в чёрный.

Замечание 2. Оценка вероятности принадлежности фрагмента к тому или иному классу вычисляется с помощью применения обученной модели к фрагменту, преобразованному в тип torch.Tensor.

Рис. 3. Примеры цифр




(а) (б)

Рис. 4. Визуализация результата предварительной предобработки фотографии: а) полутоновое и б) бинарное изображение

## 4. Литература

1. Рычагов М.Н. Лекции по предмету «Нейронные сети». МИЭТ, 2023 г. Лекции 1, 7.