## Описание метода обратного распространения ошибки

В задаче реализуется классификация изображений из датасета MNIST.

—входной вектор из 784 значений, является преобразованным изображением из датасета 28 на 28 пикселей;

— выходной вектор из 10 правильных значений;

— вектор предсказанных значений;

– матрица весов между входным и скрытым слоями;

– смещение для весов скрытого слоя;

– матрица весов между скрытым и выходным слоями;

– смещение для весов скрытого слоя.

Использовалась функция активации Softmax

и функция потерь категориальная кросс-энтропия

Алгоритм работы сети обратного распространения ошибки:

1. Прямое распространение.
2. Вычисление потерь.
3. Обратное распространение ошибки  
    – ошибка на выходе

– градиенты для весов и смещений выхожного слоя

– ошибка на скрытом слое

*–* градиенты для весов и смещений скрытого слоя

1. Обновление весов.

# Реализация программы

Для начала загрузим библиотеки необходимые для выполнения работы

import numpy as np

import matplotlib.pyplot as plt

from tensorflow.keras.datasets import mnist

from tensorflow.keras.utils import to\_categorical

numpy для работ с матрицами

matplotlib для работы с рисунками

tensorflow для загрузки датасета MNIST и формулы для преобразования вектора в one-hot метки

Реализуем функции потерь и ошибок

def relu(z):

    return np.maximum(0, z)

def relu\_derivative(z):

    return np.where(z > 0, 1, 0)

def softmax(z):

    exp\_z = np.exp(z - np.max(z, axis=1, keepdims=True))

    return exp\_z / np.sum(exp\_z, axis=1, keepdims=True)

def categorical\_cross\_entropy(y\_pred, y\_true):

    m = y\_true.shape[0] # количество примеров

    log = -np.log(y\_pred[np.arange(m), np.argmax(y\_true, axis=1)])

    loss = np.sum(log) / m

    return loss

Реализуем функцию для загрузки датасета

def load\_mnist():

    (x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

    # Нормализация изображений

    x\_train = x\_train.astype('float32') / 255.0

    x\_test = x\_test.astype('float32') / 255.0

    x\_train = x\_train.reshape(x\_train.shape[0], -1)

    x\_test = x\_test.reshape(x\_test.shape[0], -1)

    y\_train\_one\_hot = to\_categorical(y\_train, 10)

    y\_test\_one\_hot = to\_categorical(y\_test, 10)

     # y\_test для расчета accuracy

    return x\_train, y\_train\_one\_hot, x\_test, y\_test\_one\_hot, y\_test

Реализуем класс нейронной сети согласно описанным выше формулам

class NeuralNetwork:

    def \_\_init\_\_(self, input\_size, hidden\_size, output\_size):

        # Инициализация весов и смещений

        self.W1 = np.random.randn(input\_size, hidden\_size)

        self.b1 = np.zeros((1, hidden\_size))

        self.W2 = np.random.randn(hidden\_size, output\_size)

        self.b2 = np.zeros((1, output\_size))

        self.hist = {}

    def forward(self, X):

        # X: (batch\_size, input\_size)

        # Скрытый слой

        self.hist['Z1'] = X @ self.W1 + self.b1 # (batch\_size, hidden\_size)

        self.hist['A1'] = relu(self.hist['Z1']) # (batch\_size, hidden\_size)

        # Выходной слой

        self.hist['Z2'] = self.hist['A1'] @ self.W2 + self.b2 # (batch\_size, output\_size)

        A2 = softmax(self.hist['Z2']) # (batch\_size, output\_size)

        self.hist['A2'] = A2

        return A2

    def backward(self, X, y\_true):

        # y\_true: (batch\_size, output\_size)

        m = X.shape[0]

        A1 = self.hist['A1']

        A2 = self.hist['A2']

        Z1 = self.hist['Z1']

        # dL/dZ2 = A2 - Y

        dZ2 = A2 - y\_true # (batch\_size, output\_size)

        # dL/dW2 = dL/dZ2 \* dZ2/dW2 = dL/dZ2 \* A1.T

        self.dW2 = (1/m) \* (A1.T @ dZ2) # (hidden\_size, output\_size)

        # dL/db2 = dL/dZ2 \* dZ2/db2 = dL/dZ2 \* 1

        self.db2 = (1/m) \* np.sum(dZ2, axis=0, keepdims=True) # (1, output\_size)

        # dL/dA1 = dL/dZ2 \* dZ2/dA1 = dZ2 @ W2.T

        dA1 = dZ2 @ self.W2.T # (batch\_size, hidden\_size)

        # dL/dZ1 = dL/dA1 \* dA1/dZ1 = dA1 \* relu\_derivative(Z1)

        dZ1 = dA1 \* relu\_derivative(Z1) # (batch\_size, hidden\_size)

        # dL/dW1 = dL/dZ1 \* dZ1/dW1 = X.T @ dZ1

        self.dW1 = (1/m) \* (X.T @ dZ1) # (input\_size, hidden\_size)

        # dL/db1 = dL/dZ1 \* dZ1/db1 = dL/dZ1 \* 1

        self.db1 = (1/m) \* np.sum(dZ1, axis=0, keepdims=True) # (1, hidden\_size)

    def update\_parameters(self, learning\_rate):

        self.W1 -= learning\_rate \* self.dW1

        self.b1 -= learning\_rate \* self.db1

        self.W2 -= learning\_rate \* self.dW2

        self.b2 -= learning\_rate \* self.db2

    def predict(self, X):

        A2 = self.forward(X)

        return np.argmax(A2, axis=1)

    def evaluate(self, X\_test, y\_test\_orig):

        predictions = self.predict(X\_test)

        accuracy = np.mean(predictions == y\_test\_orig)

        return accuracy

Обучим модель

INPUT\_SIZE = 784  # 28x28

HIDDEN\_SIZE = 128 # Количество нейронов в скрытом слое

OUTPUT\_SIZE = 10  # 10 классов (цифры 0-9)

LEARNING\_RATE = 0.1

EPOCHS = 20

BATCH\_SIZE = 64

print("Загрузка MNIST")

x\_train, y\_train\_one\_hot, x\_test, y\_test\_one\_hot, y\_test\_orig = load\_mnist()

print(f"Размер обуч выборки: X={x\_train.shape}, Y={y\_train\_one\_hot.shape}")

print(f"Размер тест выборки: X={x\_test.shape}, Y={y\_test\_one\_hot.shape}")

model = NeuralNetwork(INPUT\_SIZE, HIDDEN\_SIZE, OUTPUT\_SIZE)

train\_losses = []

test\_accuracies = []

print(f"\nНачало обучения: {EPOCHS} эпох, размер батча {BATCH\_SIZE}, скорость обучения {LEARNING\_RATE}")

for epoch in range(EPOCHS):

    epoch\_loss = 0

    cur = np.random.permutation(x\_train.shape[0])

    x\_train\_shuffled = x\_train[cur]

    y\_train\_shuffled = y\_train\_one\_hot[cur]

    for i in range(0, x\_train.shape[0], BATCH\_SIZE):

        X\_batch = x\_train\_shuffled[i:i+BATCH\_SIZE]

        y\_batch = y\_train\_shuffled[i:i+BATCH\_SIZE]

        y\_pred\_batch = model.forward(X\_batch)

        loss = categorical\_cross\_entropy(y\_pred\_batch, y\_batch)

        epoch\_loss += loss \* X\_batch.shape[0]

        model.backward(X\_batch, y\_batch)

        model.update\_parameters(LEARNING\_RATE)

    avg\_epoch\_loss = epoch\_loss / x\_train.shape[0]

    train\_losses.append(avg\_epoch\_loss)

    accuracy = model.evaluate(x\_test, y\_test\_orig)

    test\_accuracies.append(accuracy)

    print(f"Эпоха {epoch+1}/{EPOCHS} - Потери: {avg\_epoch\_loss:.4f} - Точность на тесте: {accuracy:.4f}")

final\_accuracy = model.evaluate(x\_test, y\_test\_orig)

print(f"\nИтоговая точность на тестовом наборе: {final\_accuracy:.4f}")

Итоговая точность на тестовом наборе: 0.9368

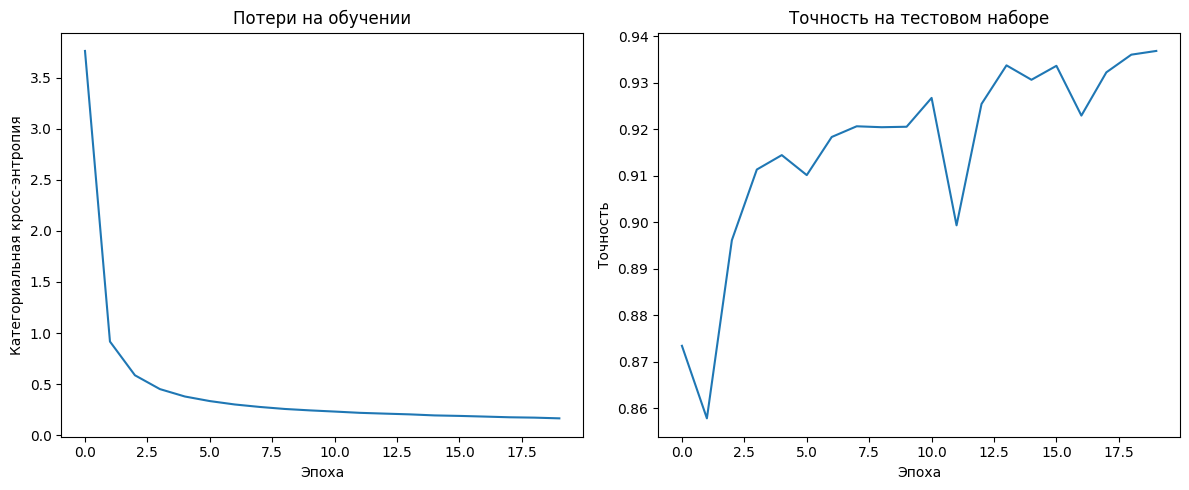


Рисунок – Графики обучения модели

Проверим работу нейронной сети на нескольких изображениях

# Визуализация

num\_examples\_to\_show = 5

random\_indices = np.random.choice(x\_test.shape[0], num\_examples\_to\_show, replace=False)

example\_images = x\_test[random\_indices]

example\_labels\_orig = y\_test\_orig[random\_indices]

predictions = model.predict(example\_images)

plt.figure(figsize=(10, 4))

for i in range(num\_examples\_to\_show):

    plt.subplot(1, num\_examples\_to\_show, i + 1)

    plt.imshow(example\_images[i].reshape(28, 28), cmap='gray')

    plt.title(f"Предсказание: {predictions[i]}\nОригинал: {example\_labels\_orig[i]}")

    plt.axis('off')

plt.tight\_layout()

plt.show()



Рисунок – Тестирование работы нейросети

Готовый код можно потрогать руками если перейти по ссылке https://github.com/indall108302/MNT взять файлик LR1.ipynb и запустить его в google colab