Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**ЛАБОРАТОРНАЯ РАБОТА №6**

**Дисциплина: Алгоритмы цифровой обработки мультимедиа**

**Тема: Введение в нейронные сети**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. А. Эзри

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.А. Крамаренко

**Цель работы:** провести классификацию изображений с помощью нейронных сетей.

**Задания:**

**Задание 1.** Построить многослойный персептрон средствами библиотеки Keras языка Python (или соответствующей библиотеки на другом языке), позволяющий распознавать цифры. Для обучения и тестирования использовать базу MNIST. Возможно использовать любую функции активации, любую структуру сети и любой алгоритм обучения.

**Задание 2.** Исследовать нейронную сеть, построенную в задании 1. Подобрать количество эпох обучения, необходимое для выбранного предела корректной работы на тестовой базе. Сравнить несколько значений количества эпох по параметрам скорость обучения, скорость работы сети, процент корректной работы на тестовой базе.

**Задание 3.** Построить свёрточную нейронную сеть, решающую ту же задачу классификации. Рассмотреть несколько архитектур, подобрать оптимальные параметры. Сравнить результаты с предыдущей архитектурой.

**Ход работы:**

Искусственная нейронная сеть состоит из нескольких слоев нейронов, включая входной слой, скрытые слои и выходной слой. Каждый нейрон в сети связан с нейронами в соседних слоях через веса.

Работа нейронной сети заключается в передаче информации от входного слоя через скрытые слои и, наконец, к выходному слою. Каждый нейрон в сети применяет взвешенные суммы к данным, а затем применяет функцию активации для определения своего выхода. В процессе обучения веса в нейронной сети подстраиваются так, чтобы минимизировать ошибку предсказания.

Преимущество нейронных сетей состоит в их способности решать сложные задачи, такие как распознавание образов, обработка естественного языка, синтез речи и другие. Они являются ключевой технологией в области глубокого обучения.

Примеры типовых архитектур нейронных сетей включают в себя многослойные перцептроны (MLP), свёрточные нейронные сети (CNN) для обработки изображений и рекуррентные нейронные сети (RNN) для обработки последовательных данных.

**Многослойный перцептрон**

Установим библиотеку TensorFlow и фреймворк Keras.

Загрузим данные MNIST – датасет рукописных цифр. x\_train и x\_test содержат изображения рукописных цифр, а y\_train и y\_test - соответствующие метки классов (цифры).

Тренировочный датасет состоит из 60000 изображений, тестовый – из 10000 изображений. Размер изображений: 28x28. Эти данные получены при помощи свойства shape элементов x\_train, y\_train, x\_test, y\_test.

Преобразуем 2D массивы изображений в 1D массивы размером 784, преобразуем тип чисел в float32 и поделим на 255, чтобы значения пикселей находились в диапазоне от 0 до 1.

Затем используем метод to\_categorical, чтобы целевые классы преобразовать в массивы, где под индексом, равным номеру класса, стоит единица, а остальные элементы – нули (этот формат называют one-hot encoding). Например, число 7 соответствует классу «7», в результате получится массив [0, 0, 0, 0, 0, 0, 0, 1, 0, 0], означающий, что картинка относится к классу «7» с вероятностью 100%. Это нужно чтобы сопоставить вероятности классов с нейронами выходного слоя для обучения сети решению задачи многоклассовой классификации.

Создадим модель последовательной нейронной сети, к которой будут поочередно добавляться слои. Добавим входной слой, 2 скрытых слоя с 512 нейронами и функцией активации ReLU (эти слои обеспечивают способность сети извлекать признаки из данных) и выходной слой с 10 нейронами с функцией активации Softmax для определения вероятности принадлежности к каждому из десяти классов. После каждого скрытого слоя будем рандомно занулять (отбрасывать) 20% нейронов.

Модель компилируется с использованием оптимизатора Adam и категориальной функции потери для обучения модели для многоклассовой классификации. Также определяются метрики для оценки производительности модели, в данном случае – точность классификации.

Далее обучаем модель течение 50 эпох (полных проходов через данные). Обновление весов происходит после каждых 128 образцов (это задаётся через batch size). После каждой эпохи измеряется точность на валидационной выборке (10% от обучающей выборки), и, если точность повысилась, специальная callback-функция сохраняет веса в файл.

Результаты обучения на каждой эпохе представлены ниже:

|  |  |
| --- | --- |
| **Номер эпохи** | **Точность на валидационной выборке** |
| **1** | **0,9872** |
| 2 | 0,9867 |
| 3 | 0,9857 |
| 4 | 0,9855 |
| **5** | **0,9875** |
| 6 | 0,9858 |
| **7** | **0,9878** |
| 8 | 0,9870 |
| 9 | 0,9865 |
| 10 | 0,9868 |
| 11 | 0,9850 |
| 12 | 0,9872 |
| 13 | 0,9877 |
| 14 | 0,9850 |
| 15 | 0,9865 |
| 16 | 0,9868 |
| 17 | 0,9875 |
| 18 | 0,9865 |
| 19 | 0,9858 |
| **20** | **0,9882** |
| 21 | 0,9867 |
| 22 | 0,9878 |
| 23 | 0,9858 |
| 24 | 0,9862 |
| 25 | 0,9862 |
| 26 | 0,9868 |
| 27 | 0,9875 |
| 28 | 0,9873 |
| 29 | 0,9872 |
| 30 | 0,9870 |
| 31 | 0,9870 |
| 32 | 0,9875 |
| 33 | 0,9868 |
| **34** | **0,9883** |
| **35** | **0,9892** |
| … | … |
| 50 | 0.9870 |

Эпохи после 35-й пропущены, т.к. на них точность не увеличивалась. Эпохи, на которых точность увеличивалась и которые были сохранены, выделены жирным.

В итоге удалось добиться точности 98,9% на валидационной выборке. При этом на тестовой выборке модель имела точность 98,5%. 9851 из 10000 изображений было классифицировано верно.

Была протестирована модель с добавленным третьим слоем, но это не помогло, точность стала чуть ниже, в пределах статистической погрешности, возникшей из-за случайной инициализации весов.

**Свёрточная нейросеть**

**Сверточная нейронная сеть** (Convolutional Neural Network, CNN) – это тип искусственной нейронной сети, разработанный специально для обработки и анализа структурированных данных, таких как изображения и видео. Она имеет революционное значение в области компьютерного зрения, и она может успешно выполнять задачи, связанные с распознаванием образов, классификацией объектов, сегментацией изображений, детекцией объектов и многими другими задачами, требующими анализа визуальных данных.

Рассмотрим подробно описание основных слоёв свёрточной нейронной сети:

1. Свёрточные слои извлекают признаки из изображений. Сверточные фильтры перемещаются по входным данным и выполняют операцию свертки, что позволяет выявить локальные паттерны, такие как грани, углы, текстуры и другие характеристики. Каждый свёрточный слой может иметь несколько фильтров.
2. Пулинговый слой призван снижать размерность изображения. Исходное изображение делится на блоки размером w×h и для каждого блока вычисляется некоторая функция. Чаще всего используется функция максимума (англ. max pooling) или (взвешенного) среднего (англ. (weighted) average pooling). Обучаемых параметров у этого слоя нет. Основной целью пулингового слоя является уменьшение изображения, чтобы последующие свёртки оперировали над большей областью исходного изображения.
3. Слой Dropout – помогает избежать переобучение, отбрасывая случайные параметры.
4. Слой выравнивания (Flatten) – выпрямляет многомерные матрицы в одномерные вектора.
5. Слой Dense – простой полносвязный слой.

Как и в прошлый раз, загружаем данные, обрабатываем их и кодируем метки классов.

Создаем модель свёрточной нейронной сети с несколькими свёрточными слоями (Conv2D), слоями пулинга (MaxPooling2D), слоем выравнивания (Flatten) и полносвязным выходным слоем (Dense).

Модель компилируется с использованием оптимизатора Adam и категориальной функции потери. Также определяются метрики для оценки производительности модели.

Далее обучаем модель течение 32 эпох. Обновление весов происходит после каждых 128 образцов (это задаётся через batch size). После каждой эпохи измеряется точность на валидационной выборке, и, если точность повысилась, специальная callback-функция сохраняет веса в файл.

Было протестировано несколько комбинаций слоёв и их параметров. Наилучшую точность удалось достигнуть со следующей конфигурацией модели:

model = keras.Sequential([

    keras.layers.Input(shape=(width, height, depth)),

    keras.layers.Conv2D(filters=32, kernel\_size=(3, 3), activation="relu"),

    keras.layers.Conv2D(filters=32, kernel\_size=(3, 3), activation="relu"),

    keras.layers.MaxPooling2D(pool\_size=(2, 2)),

    keras.layers.Conv2D(filters=64, kernel\_size=(3, 3), activation="relu"),

    keras.layers.Conv2D(filters=64, kernel\_size=(3, 3), activation="relu"),

    keras.layers.MaxPooling2D(pool\_size=(2, 2)),

    keras.layers.Flatten(),

    keras.layers.Dropout(0.5),

    keras.layers.Dense(n\_classes, activation="softmax")

])

Результаты обучения на каждой эпохе представлены ниже:

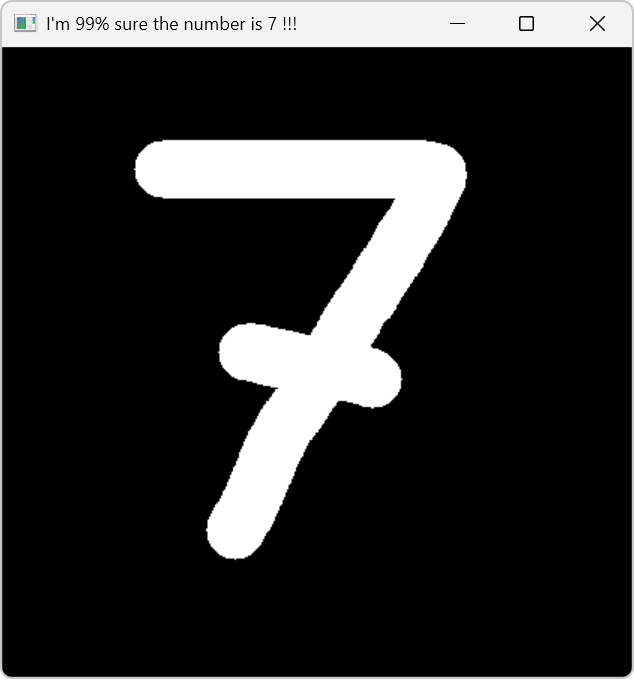
|  |  |
| --- | --- |
| **Номер эпохи** | **Точность на валидационной выборке** |
| **1** | **0.9858** |
| **2** | **0.9887** |
| **3** | **0.9927** |
| 4 | 0.9917 |
| 5 | 0.9925 |
| 6 | 0.9927 |
| **7** | **0.9935** |
| **8** | **0.9937** |
| **9** | **0.9940** |
| 10 | 0.9927 |
| 11 | 0.9922 |
| **12** | **0.9942** |
| 13 | 0.9938 |
| 14 | 0.9940 |
| 15 | 0.9940 |
| 16 | 0.9933 |
| **17** | **0.9955** |
| 18 | 0.9942 |
| 19 | 0.9940 |
| 20 | 0.9948 |
| … | … |
| 32 | 0.9942 |

В таблице указано только первые 20 эпох и последняя. Наивысшая точность была достигнута на 17-й эпохе, а далее пошла на спад. Эпохи, на которых точность увеличивалась и которые были сохранены, выделены жирным.

В итоге удалось добиться точности 99,55% на валидационной выборке. При этом на тестовой выборке модель имела точность 99,54%. 9954 из 10000 изображений было классифицировано верно.

**One more thing**

Мною была разработана программа, в которой пользователь рисует цифру от 0 до 9, а обученная свёрточная нейросеть её угадывает :)

**Листинг программы**

Файл Lab\_6\_MLP.ipynb

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  import keras  from keras.api.datasets import mnist  from keras.api.utils import to\_categorical |

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

|  |
| --- |
| # print(x\_train)  # print(y\_train)  np.set\_printoptions(linewidth=120)  print(x\_train[0])  print(y\_train[0])  for i in range(9):      plt.subplot(3, 3, i+1)      plt.imshow(x\_train[i], cmap='gray')      plt.title(f"Class {y\_train[i]}") |

|  |
| --- |
| print("x\_train shape", x\_train.shape)  print("y\_train shape", y\_train.shape)  print("x\_test shape", x\_test.shape)  print("y\_test shape", y\_test.shape)  n\_train\_images = x\_train.shape[0]  n\_test\_images = x\_test.shape[0] |

|  |
| --- |
| height = 28  width = 28  depth = 1  # Преобразуем изображения в векторы, изменим формат на float и нормализуем  x\_train = x\_train.reshape(n\_train\_images, height \* width \* depth).astype(np.float32) / 255  x\_test = x\_test.reshape(n\_test\_images, height \* width \* depth).astype(np.float32) / 255  print(x\_train.shape)  print(x\_test.shape) |

|  |
| --- |
| n\_classes = 10  print(y\_train[0:5])  # Закодировать номер выходного класса массивом,  # где под индексом, равным номеру класса,  # стоит единица, а остальные элементы - нули  # (этот формат ещё называют one-hot)  y\_train = to\_categorical(y\_train, n\_classes)  y\_test = to\_categorical(y\_test, n\_classes)  print(y\_train[0:5]) |

|  |
| --- |
| # Построение нейросети  hidden\_size = 512  # размер скрытых слоёв  model = keras.Sequential()  model.add(keras.layers.Input(shape=(width \* height,)))          # входной слой  model.add(keras.layers.Dense(hidden\_size, activation='relu'))   # 1-й скрытый слой (функция активации Rectified Linear Unit заменяет отрицательные значения нулём)  model.add(keras.layers.Dropout(0.2))                            # часть нейронов зануляем рандомно, это поможет от переобучения  model.add(keras.layers.Dense(hidden\_size, activation='relu'))   # 2-й скрытый слой  model.add(keras.layers.Dropout(0.2))  model.add(keras.layers.Dense(n\_classes, activation='softmax'))  # функция softmax преобразует полученные числа в распределение вероятностей (сумма = 1) |

|  |
| --- |
| # Компиляция модели с заданеим функции потерь и оптимизации  # adam = optimizer=keras.optimizers.Adam(learning\_rate=0.001)  model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy']) |

|  |
| --- |
| batch\_size = 128  n\_epochs = 50  save\_path = f'checkpoints/MLP/ckpt\_{len(model.layers)}\_{hidden\_size}' + '\_epoch\_{epoch:02d}.keras'  save\_callback = keras.callbacks.ModelCheckpoint(save\_path, verbose=1, monitor='val\_accuracy', mode='max', save\_best\_only=True)  # Обучение модели  model.fit(x\_train,            y\_train,            batch\_size=batch\_size,            epochs=n\_epochs,            verbose=1,            validation\_split=0.1,            callbacks = [save\_callback])  # Сохранение весов в файл  # model.save\_weights(f'MLP\_{len(model.layers)}\_layers\_{n\_epochs}\_epochs\_{hidden\_size}\_hidden\_size.weights.h5') |

|  |
| --- |
| # Оценка точности на тестовом датасете  model = keras.saving.load\_model("checkpoints/MLP/ckpt\_5\_512\_epoch\_35.keras")  score = model.evaluate(x\_test, y\_test, verbose=0, return\_dict=True)  print(score)  predicted\_classes = model.predict(x\_test)  correct\_count = 0  print("Ошибки:")  for index, image in enumerate(predicted\_classes):  probability = max(image)  actual\_number = np.where(y\_test[index] == 1)[0][0]  prediction = np.where(image == probability)[0][0]  if actual\_number != prediction:  print(f'Реальное число: {actual\_number}, распознанное число: {prediction}, {probability \* 100}%')  else:  correct\_count += 1  print(f'\n{correct\_count} из {n\_test\_images} распознано верно') |

Файл Lab\_6\_CNN.ipynb

|  |
| --- |
| import numpy as np  import keras  from keras.api.datasets import mnist  from keras.api.utils import to\_categorical |

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

|  |
| --- |
| print("x\_train shape", x\_train.shape)  print("y\_train shape", y\_train.shape)  print("x\_test shape", x\_test.shape)  print("y\_test shape", y\_test.shape)  n\_train\_images = x\_train.shape[0]  n\_test\_images = x\_test.shape[0] |

|  |
| --- |
| height = 28  width = 28  depth = 1  # Добавим третье измерение (глубину), изменим формат на float и нормализуем  x\_train = x\_train.reshape(n\_train\_images, height, width, depth).astype(np.float32) / 255  x\_test = x\_test.reshape(n\_test\_images, height, width, depth).astype(np.float32) / 255  print(x\_train.shape)  print(x\_test.shape) |

|  |
| --- |
| n\_classes = 10  print(y\_train[0:5])  # Закодировать номер выходного класса массивом,  # где под индексом, равным номеру класса,  # стоит единица, а остальные элементы - нули  # (этот формат ещё называют one-hot)  y\_train = to\_categorical(y\_train, n\_classes)  y\_test = to\_categorical(y\_test, n\_classes)  print(y\_train[0:5]) |

|  |
| --- |
| # Построение нейросети  model = keras.Sequential([  keras.layers.Input(shape=(width, height, depth)),  keras.layers.Conv2D(filters=32, kernel\_size=(3, 3), activation="relu"),  keras.layers.Conv2D(filters=32, kernel\_size=(3, 3), activation="relu"),  keras.layers.MaxPooling2D(pool\_size=(2, 2)),  keras.layers.Conv2D(filters=64, kernel\_size=(3, 3), activation="relu"),  keras.layers.Conv2D(filters=64, kernel\_size=(3, 3), activation="relu"),  keras.layers.MaxPooling2D(pool\_size=(2, 2)),  keras.layers.Flatten(),  keras.layers.Dropout(0.5),  keras.layers.Dense(n\_classes, activation="softmax")  ])  # Компиляция модели с заданеим функции потерь и оптимизации  # adam = optimizer=keras.optimizers.Adam(learning\_rate=0.001)  model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy']) |

|  |
| --- |
| batch\_size = 128  n\_epochs = 32  attempt = 3  save\_path = f'checkpoints/CNN/attempt\_{attempt}/ckpt\_{batch\_size}' + '\_epoch\_{epoch:02d}.keras'  save\_callback = keras.callbacks.ModelCheckpoint(save\_path, verbose=1, monitor='val\_accuracy', mode='max', save\_best\_only=True)  # Обучение модели  model.fit(x\_train,  y\_train,  batch\_size=batch\_size,  epochs=n\_epochs,  verbose=1,  validation\_split=0.1,  callbacks = [save\_callback]) |

|  |
| --- |
| # Оценка точности на тестовом датасете  model = keras.saving.load\_model("checkpoints/CNN/attempt\_3/ckpt\_128\_epoch\_17.keras")  score = model.evaluate(x\_test, y\_test, verbose=0, return\_dict=True)  print(score)  predicted\_classes = model.predict(x\_test)  correct\_count = 0  print("Ошибки:")  for index, image in enumerate(predicted\_classes):  probability = max(image)  actual\_number = np.where(y\_test[index] == 1)[0][0]  prediction = np.where(image == probability)[0][0]  if actual\_number != prediction:  print(f'Реальное число: {actual\_number}, распознанное число: {prediction}, {probability \* 100}%')  else:  correct\_count += 1  print(f'\n{correct\_count} из {n\_test\_images} распознано верно') |

Программа для рисования и отгадывания нарисованных цифр

import cv2

import numpy as np

import os

os.environ['TF\_CPP\_MIN\_LOG\_LEVEL'] = '3'

import keras

import pygame

model = keras.saving.load\_model("../checkpoints/CNN/attempt\_3/ckpt\_128\_epoch\_17.keras")

model\_type = 'CNN'

window\_name = 'Draw a number ;)'

scale = 15

drawing = False  # true if mouse is pressed

pt1\_x, pt1\_y = 0, 0

pygame.mixer.init()

sounds = [pygame.mixer.Sound(f'audio/{sound\_id}.wav') for sound\_id in range(10)]

# mouse callback function

def line\_drawing(event, x, y, flags, param):

    global drawing, pt1\_x, pt1\_y

    if event == cv2.EVENT\_LBUTTONDOWN:

        drawing = True

        pt1\_x, pt1\_y = x, y

    elif event == cv2.EVENT\_MOUSEMOVE:

        if drawing == True:

            cv2.line(img, (pt1\_x, pt1\_y), (x, y), color=(255, 255, 255), thickness=int(2.5 \* scale))

            pt1\_x, pt1\_y = x, y

    elif event == cv2.EVENT\_LBUTTONUP:

        drawing = False

        cv2.line(img, (pt1\_x, pt1\_y), (x, y), color=(255, 255, 255), thickness=int(2.5 \* scale))

img = np.zeros((28 \* scale, 28 \* scale), np.uint8)

cv2.namedWindow(window\_name)

cv2.setMouseCallback(window\_name, line\_drawing)

while cv2.getWindowProperty(window\_name, cv2.WND\_PROP\_VISIBLE) == 1:

    cv2.imshow(window\_name, img)

    key = cv2.waitKey(10) & 0xFF

    if key == 27:  # escape

        break

    elif key == 0 or key == 8:  # del or backspace

        img[:,:] = 0

        cv2.setWindowTitle(window\_name, window\_name)

    elif key == 13 or key == 32:  # enter or space

        if img.max() > 0:

            model\_input = img.copy()

            model\_input = cv2.resize(model\_input, (28, 28))

            if model\_type.lower() == 'mlp':

                model\_input = model\_input.reshape(1, 784)

            elif model\_type.lower() == 'cnn':

                model\_input = model\_input[np.newaxis, ..., np.newaxis]

            else:

                raise Exception('Неверно указан тип модели. Допустимые значения: MLP, CNN')

            model\_input = model\_input.astype(np.float32) / 255

            pred = model.predict(model\_input, verbose=0, batch\_size=1)[0]

            result = np.where(pred == max(pred))[0][0]

            cv2.setWindowTitle(window\_name, f'I\'m {int(max(pred) \* 100)}% sure the number is {result} !!!')

            sounds[result].play()

cv2.destroyAllWindows()

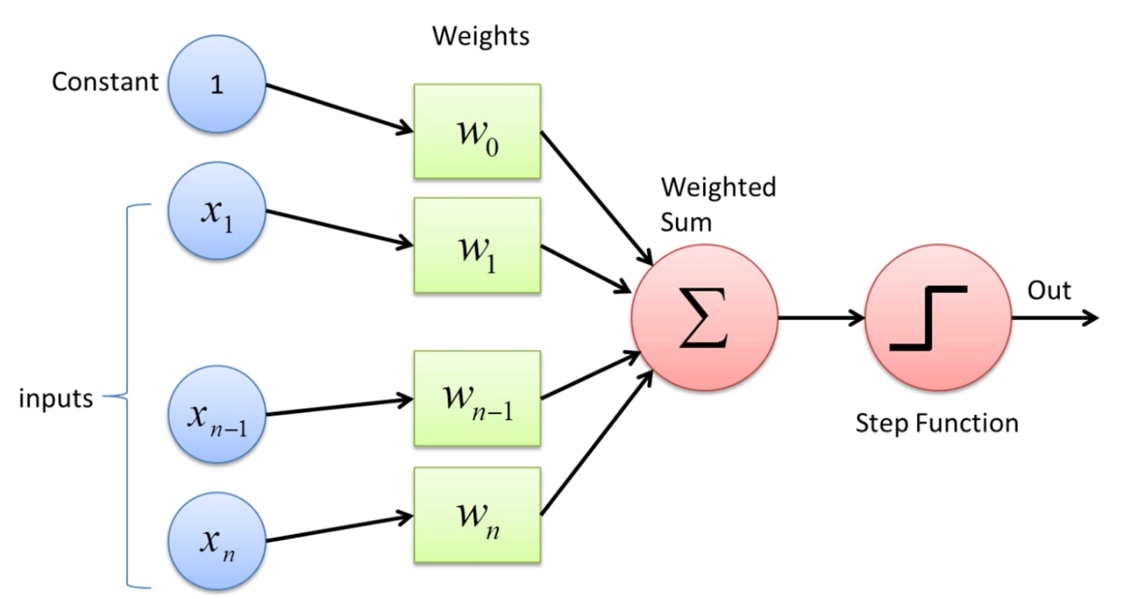
**Ответы на вопросы**

1. Опишите принцип обучения нейронной сети с учителем.

Обучение нейронной сети с учителем заключается в обучении модели на основе пар входных данных и соответствующих этим данным выходных целевых значений. Модель старается научиться преобразовывать входные данные в соответствующие целевые значения. Процесс обучения включает в себя следующие шаги: подача входных данных в модель, вычисление прогнозов модели, вычисление ошибки между прогнозами и целевыми значениями, и обновление параметров (весов) модели с целью минимизации этой ошибки. Этот процесс может повторяться на протяжении нескольких эпох до достижения приемлемого уровня точности.

1. Опишите структуру нейрона.

Нейрон состоит из трёх основных компонентов: входов, весов и функции активации. Входы представляют входные сигналы, веса – параметры, которые моделируют силу связей между входами и нейроном, и функция активации определяет, какой выход будет сгенерирован на основе входов и весов. Математически выход нейрона можно выразить как взвешенную сумму входов, обработанную функцией активации: .



1. Опишите структуру персептрона.

Персептрон состоит из входного и выходного слоёв, а также, возможно, скрытых слоёв между ними. На каждом слое находится определённое число нейронов. Нейроны соседних слоёв связаны между собой, и эти связи характеризуются весами. Нейроны получают числа с предыдущего слоя, умножают их на соответствующие веса, применяют функцию активации и передают итоговый результат нейронам следующего слоя. Функция активации определяет, будет ли нейрон активирован и передаст ли сигнал на выход.

1. Опишите смысл применения оптимизационных методов в задачах обучения нейронной сети.

Оптимизационные методы используются для настройки параметров нейронной сети с целью минимизации функции потерь. Это позволяет нейронной сети адаптироваться к данным и улучшать свои прогнозы. Примеры оптимизационных методов включают в себя градиентный спуск (Gradient Descent) и его различные вариации, такие как Adam, RMSprop, и другие. Они регулируют скорость обучения и направление обновления весов сети на основе градиента функции потерь.

1. Что такое аугментация?

Аугментация данных – это методика, которая заключается в генерации дополнительных обучающих примеров путем применения различных преобразований к существующим данным. Эти преобразования могут включать в себя повороты, отражения, смещения, изменение яркости и многое другое. Аугментация данных помогает расширить датасет, что может улучшить обобщающую способность нейронной сети и сделать её более устойчивой к различным условиям входных данных.

1. Опишите принцип пакетного и последовательного обучения? Что такое batch\_size?

При пакетном обучении (batch learning) модель обновляет свои параметры после обработки заданного количества образцов за одну итерацию обучения. Это позволяет улучшить эффективность вычислений и использовать параллелизм на многопроцессорных системах. Гиперпараметр batch\_size определяет размер пакета.

1. Покажите известные Вам функции активации.

* Сигмоидная функция (Sigmoid):
* Гиперболический тангенс (Tanh):
* ReLU (Rectified Linear Unit): – простая нелинейная функция, которая выполняет следующее преобразование для каждого входного значения: если входное значение больше или равно нулю, то ReLU оставляет его без изменений, иначе ReLU заменяет его на ноль.
* Softmax: используется на выходном слое классификационных моделей для вычисления вероятностей принадлежности к каждому классу. Она преобразует вектор значений (чаще всего выходов последнего слоя нейронной сети) в вероятностное распределение по классам. Softmax принимает входной вектор, и для каждого элемента вектора вычисляет вероятность, что данный элемент принадлежит к одному из классов. Затем эти вероятности нормализуются так, чтобы их сумма равнялась 1.