ФЕДЕРАЛЬНОЕ АГЕНТСТВО ВОЗДУШНОГО ТРАНСПОРТА
(РОСАВИАЦИЯ)

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ГРАЖДАНСКОЙ АВИАЦИИ» (МГТУ ГА)

Кафедра вычислительных машин, комплексов, систем и сетей

Н.И. Романчева

ПОСТРОЕНИЕ НЕЙРОННЫХ СЕТЕЙ РАЗЛИЧНОЙ АРХИТЕКТУРЫ

Учебно-методическое пособие
по выполнению лабораторных работ № 3–4
направления 09.03.01
и лабораторных работ № 1–2
направления 25.03.03

для студентов II курса
очной формы обучения

Москва
ИД Академии Жуковского
2025

УДК 004.032.26
ББК 6Ф7.3
Р69

Рецензент:
Терентьев А.И. – канд. техн. наук, доцент

Романчева Н.И.

Р69 Построение нейронных сетей различной архитектуры [Текст] : учебно-методическое пособие по выполнению лабораторных работ № 3–4 направления 09.03.01 и лабораторных работ № 1–2 направления 25.03.03 / Н.И. Романчева. – М.: ИД Академии Жуковского, 2025. – 40 с.

Данное учебно-методическое пособие издается в соответствии с рабочими программами учебных дисциплин «Методы машинного обучения и нейронные сети» и «Нейронные сети» по учебным планам направлений подготовки 09.03.01 и 25.03.03 для студентов очной формы обучения.

Рассмотрено и одобрено на заседаниях кафедры 25.01.2025 г. и методического совета 28.01.2025 г.

УДК 004.032.26
ББК 6Ф7.3

В авторской редакции

Подписано в печать 14.05.2025 г.
Формат 60x84/16 Печ. л. 2,5 Усл. печ. л. 2,325
Заказ № 1073/0325-УМП10 Тираж 25 экз.

Московский государственный технический университет ГА
125993, Москва, Кронштадтский бульвар, д. 20

Издательский дом Академии имени Н. Е. Жуковского
125167, Москва, 8-го Марта 4-я ул., д. 6А
Тел.: (499) 755-55-43
E-mail: zakaz@itsbook.ru

С О Д Е Р Ж А Н И Е

1. ОСНОВНЫЕ ТРЕБОВАНИЯ И ПОРЯДОК ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

Настоящее учебно-методическое пособие предназначено для студентов направления подготовки 09.03.01 (направленность (профиль) Интеллектуальные системы обработки и анализа данных), выполняющих лабораторные работы по дисциплине Методы машинного обучения и нейронные сети и направления подготовки 25.03.03 (направленность (профиль) Эксплуатация беспилотных авиационных систем), выполняющих лабораторные работы по дисциплине Нейронные сети в соответствии с ФГОС3++. В данное учебно-методическое пособие включены материалы для выполнения лабораторных работ соответственно № 3-4 и №1-2.

Продолжительность каждой лабораторной работы - 4 часа.

Целью проведения лабораторных работ является закрепление основных теоретических положений, изложенных в лекциях по указанным дисциплинам.

В процессе выполнения лабораторных работ осуществляется:

- закрепление основных теоретических положений, изложенных в лекциях на примере широко используемых архитектур нейронных сетей;
- получение навыков и приемами создания и обучения нейронных сетей;
- использование технологии программирования на Python.

Лабораторная работа состоит из следующих этапов:

1) домашняя подготовка;

2) выполнение работы на компьютере в соответствии с заданием;

3) сдача выполненной работы преподавателю на персональном компьютере;

4) распечатка результатов работы на принтере;

5) оформление отчета;

6) защита лабораторной работы.

В процессе домашней подготовки студент:

- изучает лекционный материал,
- материал по теме лабораторной работы данного учебно-методического пособия и дополнительной литературы;
- знакомится с заданием на выполнение лабораторной работы;
- готовит проект отчета по выполнению лабораторной работы.

ВЫПОЛНЕНИЕ лабораторной работы проводится во время занятий в компьютерном классе МГТУ ГА в присутствии преподавателя в соответствии с расписанием, утвержденным проректором по УМР и МП МГТУ ГА. В процессе выполнения лабораторной работы студент последовательно выполняет задание. По завершению работы - демонстрирует преподавателю результаты.

СДАЧА РАБОТЫ преподавателю на персональном компьютере заключается в демонстрации выполненной работы и выполнении

непосредственно при преподавателе индивидуального дополнительного задания.

ОТЧЕТ по каждой лабораторной работе должен содержать: название работы; цель лабораторной работы; задание на выполнение лабораторной работы; краткие комментарии по выполнению лабораторной работы; распечатки файлов результатов или диаграмм, подписанные преподавателем.

ЗАЩИТА лабораторной работы преподавателю проводится по контрольным вопросам и при наличии оформленного отчета (распечатки результатов выполнения лабораторной работы должны быть аккуратно приклеены). После защиты лабораторной работы преподавателем делается соответствующая запись на отчете студента.

РЕЙТИНГОВЫЙ КОНТРОЛЬ по лабораторным работам производится при их сдаче во время лабораторных занятий. Перед выполнением лабораторной работы производится экспресс-опрос для определения готовности студентов к выполнению работы (знания теоретического материала, целей работы и т.д.).

1) Допуск к ЛР

Допуск к выполнению ЛР происходит в виде устного опроса (при условии наличия у студента печатной версии титульного листа отчета по лабораторной работе).

Студент, не прошедший устный опрос к выполнению лабораторной работы не допускается.

2) Выполнение и защита ЛР

Минимальная оценка выставляется за выполненную и защищенную лабораторную работу, а дополнительными баллами оценивается качество выполненной лабораторной работы и полнота знаний, показанная студентами при ее защите. Лабораторная работа считается сделанной, если она выполнена полностью в соответствии с заданием.

Сроки сдачи лабораторных работ

Номер лабораторной работы	Лабораторная работа	Срок сдачи*
1	Лабораторная работа №3(1)	Лабораторная работа №3(1)
2	Лабораторная работа №4(2)	Лабораторная работа №4(2)

**Срок сдачи лабораторной работы – в соответствии с расписанием проведения лабораторных работ.*

3) Отчет по ЛР

Отчет по лабораторной работе представляется в печатном виде в формате, предусмотренном шаблоном отчета по лабораторной работе.

4) Защита ЛР

Отчет не может быть принят и подлежит доработке в случае:

- некорректной обработки результатов выполнения лабораторной работы;
- небрежного выполнения;
- низкого качества представленного материала (неполное решение);
- отсутствия необходимых разделов отчета;
- отсутствия необходимого графического материала;
- отсутствия выводов по работе и т.п.

2. ЛАБОРАТОРНАЯ РАБОТА №3(1) СОЗДАНИЕ НЕЙРОСЕТИ ДЛЯ РАСПОЗНАВАНИЯ РУКОПИСНЫХ ЦИФР ИЗ НАБОРА ДАННЫХ MNIST С ПОМОЩЬЮ KERAS

2.1. Цель работы

Целью данной работы является получение практических навыков применения основных методов первичной обработки данных и нейронных сетей с помощью библиотеки Keras (python).

2.2. Задание на выполнение работы

1. Написать нейросеть для распознавания рукописных цифр датасета. MNIST с помощью библиотеки Keras.
2. Вывести: истории обучения модели, график точности; матрицу ошибок.
3. Дополнительное задание: Разработать модель определения марки автомобиля по фотографии. Датасет предоставлен. Необходимо разработать пилотный проект, где созданная и обученная нейросеть будет различать (с точностью 60%) 3 марки автомобилей.

2.3. Порядок выполнения

1) Подключить библиотеки Keras.
2) Загрузить набор данных с рукописными цифрами.
3) Преобразовать данные для обработки.
4) Создать и скомпилировать нейронную сеть.
5) Обучить нейронную сеть.
6) Сохранить обученную нейронную сеть.
7) Использовать сеть для распознавания рукописных цифр.
8) Провести анализ и сделать выводы по следующим данным:
   - истории обучения модели;
   - график точности;
   - матрица ошибок.

2.4. Основные теоретические сведения и приемы работы

2.4.1. Введение

Классической задачей, решаемой нейронной сетью традиционно считается распознавание изображений. Данная задача относится к задаче классификации.

Теория распознавания рукописных цифр основана на использовании свёрточных нейронных сетей (CNN). Это специальная архитектура искусственных нейронных сетей, предложенная Яном Лекуном в 1988 году и нацеленная на эффективное распознавание образов.

Convolutional neural network (CNN, ConvNet), или сверточная нейронная сеть - класс глубоких нейронных сетей, часто применяемый в анализе визуальных образов. Сверточные нейронные сети являются разновидностью многослойного перспептрона с использованием операций свёртки. Сверточные слои используют операцию свертки для входных данных и передают результат в следующий слой, т.е. сеть глубже с меньшим количеством параметров. Сверточные сети нашли применение в распознавании изображений и видео, рекомендательных системах, классификации изображений, NLP (natural language processing) и анализе временных рядов.

В качестве входных данных сверточная нейронная сеть в основном принимает цветные изображения в формате RGB. В случае с рукописными цифрами на вход приходит чёрно-белое изображение, а значит, канал цветности будет только один.

В сверточных нейронных сетях**,** используемых для обработки изображений, информация проходит через несколько слоёв обработки. Каждый слой выполняет определённую функцию по обработке сигнала и извлечению признаков изображения (рис.1).

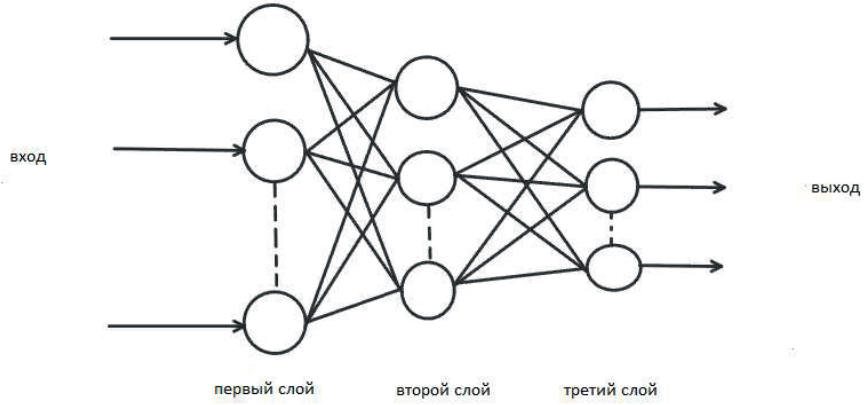


Рисунок 1 – Структура нейронной сети

Задача обучения нейронной сети состоит в нахождении весовых коэффициентов, обеспечивающих правильное распознавание цифр. Эта задача решается методом обратного распространения ошибки.

После обучения модели сверточной нейронной сети выполняется оценка точности модели с помощью метрик классификации для каждой из цифр отдельно.

Одним из современных подходов в распознавании геометрических образов является подход, основанный на вписывании неявных полиномов в облако точек. Плюсом данного подхода является его универсальность, и кроме

того, для его применения не нужно проводить тщательную предобработку данных. Непосредственное развитие в итоговом алгоритме классификации данный подход не получил, и рассматривается скорее как перспективное направление дальнейшего исследования. Для большинства цифр без предобработки такой метод даёт неосмысленный результат (рис.2).

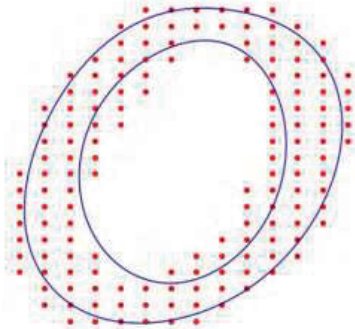


Рисунок 2 - Описание цифры 0 полиномом 4 степени

Для преодоления недостатков метода, описанного выше, можно воспользоваться его модификацией – 3L алгоритмом. Смысл этого алгоритма в том, что делается попытка разделить объект на два класса и ищется такая кривая, которая наилучшим образом отделяла бы эти два класса. Этому методу фактически с точки зрения визуального совпадения необходимо указать «коридор», внутри которого прошёл полином.

Результаты применения этого метода представлены на рис.3.

Критичным для данного метода является необходимость указывать алгоритму набор внешних и внутренних точек. Такой алгоритм усложняется непропорционально улучшению качества классификации, который этот метод потенциально может обеспечить. Кроме того, некоторые цифры, такие например как «3», плохо опишутся полиномом таким способом, так как полином функция гладкая, а у «3» серединную часть представляется сложным описать гладко.

Непосредственная классификация с помощью данного метода могла бы проводиться, благодаря скрещиванию его с цепным кодированием: после нахождения полинома, можно применить цепную кодировку уже не для исходного объекта, а для полинома, его описывающего. Такая кодировка будет более «гладкой», а, следовательно, все объекты одного класса будут больше похожи на свой класс и сильнее отличаться от объектов не своего класса.

В данной лабораторной работе используются библиотеки TensorFlow и Keras для распознавания рукописных цифр из известного датасета MNIST.

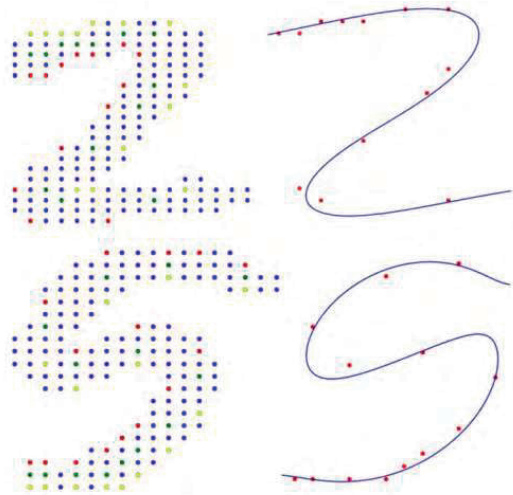


Рисунок 3 - Результат применения 3L алгоритма к цифрам 2 и 5

Датасет MNIST содержит 70 000 изображений рукописных цифр, каждое из которых представляет собой черно-белое изображение размером 28x28 пикселей. Он разделен на две части: 60 000 образцов для обучения и 10 000 для тестирования модели. Датасет загружается напрямую через библиотеку Keras, поэтому не требуется установка дополнительных файлов. Данные уже разделены на обучающую и тестовую выборки, что позволяет сделать акцент на создании и обучении модели.

После загрузки и нормализации данных создается модель. Для этого используется Sequential API из Keras для построения модели. Разрабатываемая нейронная сеть будет состоять из входного слоя, который "выпрямляет" изображения из 2D в вектор, двух скрытых слоев с 128 и 64 нейронами соответственно и функцией активации ReLU, и выходного слоя с 10 нейронами (по количеству классов цифр) и функцией активации softmax для получения вероятностей классов.

Обучение модели - это процесс, в котором модель учится распознавать правильные паттерны в данных. В нашем случае - обучаем ее распознавать рукописные цифры. При обучении используем метод fit. Результаты получаем после заданного количества эпох.

Работа с базой данных MNIST. Для обучения нейронной сети будем использовать архив http://deeplearning.net/data/mnist/mnist.pkl.gz с сайта Лаборатории машинного обучения Университета Монреаля, сформированный

на основе базы данных MNIST, который содержит 70 000 изображений рукописных цифр, разделенных на три набора:

1) training_data – набор из 50 000 изображений предназначен для обучения нейронных сетей;

2) validation_data – набор из 10 000 изображений предназначен для текущей оценки работы алгоритма обучения и подбора параметров обучения
3) test_data – набор из 10 000 изображений предназначен для проверки работы нейронной сетей;

Каждый набор состоит из двух списков: списка изображений (в градациях серого) и соответствующего списка цифр в диапазоне от 0 до 9. Изображение представлено в виде одномерного numpy-массива размера 784 = 28 × 28 значений от 0 до 1, где 0 соответствует черному цвету пиксела, а 1 – белому.

2.4.2. Приемы работы

1. Сначала импортируются необходимые библиотеки TensorFlow и подмодули из TensorFlow.keras.
2. Далее данные MNIST загружаются и нормализуются путем деления на 255 для масштабирования значений пикселей в диапазон от 0 до 1.
3. Создается модель Sequential, состоящая из слоев Flatten, Dense и Dropout, а также всего два слоя:
   - скрытый полносвязный слой;
   - выходной слой с функциями активации ReLU и Softmax соответственно.
4. Модель компилируется:
   - с оптимизатором "adam",
   - функцией потерь "sparse_categorical_crossentropy",
   - и метрикой "accuracy".
5. Далее происходит этап обучения модели на обучающих данных с 5 эпохами, и процесс обучения включает валидацию на тестовых данных.
6. Оценивается качество модели на тестовых данных путем вычисления потерь и точности, и выводится значение точности на экран
7. Идет обработка ввода пользователем картинки и распознавание на ней текста.

С помощью обученной модели идет предсказание цифры на изображении.

8. Заключительный этап обработки ошибок.

Код, является имплементацией нейронной сети для обучения на датасете MNIST с использованием библиотеки TensorFlow. Процесс начинается с загрузки и предобработки данных, а затем модель нейронной сети создается, компилируется, обучается, и оценивается на тестовых данных.

1. Сначала, импортируется TensorFlow и его подмодули, используя следующее выражение:

```
import tensorflow as tf
from tensorflow.keras import layers, models
```

Здесь импортируем TensorFlow под именем tf и некоторые из его компонентов из модуля keras, такие как layers и models.

2. Затем, загружаем и предобрабатываем датасет MNIST при помощи следующего кода:

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```

Этот код загружает датасет MNIST, который состоит из изображений рукописных цифр и их меток, и затем нормализует значения пикселей в диапазоне от 0 до 1 путем деления на 255.

3. Следующий шаг - создание модели нейросети:

```
model = models.Sequential([
    layers.Flatten(input_shape=(28, 28)),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.2),
    layers.Dense(10, activation='softmax')
        ])
```

Этот код создает последовательную модель нейросети, которая включает в себя слои для преобразования двумерного массива в одномерный (Flatten), полносвязного слоя с функцией активации ReLU, Dropout слоя для предотвращения переобучения, и заключительного полносвязного слоя с функцией активации Softmax для предсказания вероятностей для каждой из 10 классов цифр.

4. Компиляция модели:

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
```

Этот блок кода компилирует модель с оптимизатором 'adam', функцией потерь 'sparse_categorical_crossentropy' и метрикой точности ('accuracy').

5. Обучение модели:

```
model.fit(x_train, y_train, epochs=20, validation_data=(x_test, y_test))
```

В этом блоке кода модель обучается на обучающих данных с 5 эпохами обучения, и дополнительно происходит валидация на тестовых данных.

6. Оценка качества модели:

```
test_loss, test_acc = model.evaluate(x_test, y_test)
print("Test accuracy:", test_acc)
```

7. Обработка ввода от пользователя и предсказание:

Цикл while True для ввода пути к изображению.

```
Image.open(path).convert('L')#Открытие изображения и преобразование его в оттенки серого.
img.resize((28, 28)) #Изменение размера изображения до 28x28.
np.array(img) / 255.0 #Преобразование изображения в массив NumPy и его нормализация
np.expand_dims(img_array, axis=0) #Добавление измерения для предсказания.
model.predict(img_array) #Предсказание класса на изображении.
```

np.argmax(prediction)#Получение индекса класса с наибольшей вероятностью.
print("Предсказанная цифра:", digit) #Вывод предсказанной цифры.

8. Обработка возможных ошибок:

except Exception as e # Обработка исключений.
print("Ошибка:", e) #Вывод сообщения об ошибке, если произошло исключение.

Таким образом, оцениваем качество модели на тестовых данных.

Один из вариантов кода программы приведен ниже.

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
import numpy as np
from PIL import Image

# Загрузка и предобработка данных
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0  # Нормализация данных

# Создание модели нейросети
model = models.Sequential([
    layers.Flatten(input_shape=(28, 28)),  # Преобразование двумерного массива в одномерный
    layers.Dense(128, activation='relu'),  # Полносвязный слой с функцией активации ReLU
    layers.Dropout(0.2),  # Dropout для предотвращения переобучения
    layers.Dense(10, activation='softmax')  # Выходной слой с функцией активации Softmax для предсказания вероятностей
])

# Компиляция модели
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Обучение модели
model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test))

# Оценка качества модели на тестовых данных
test_loss, test_acc = model.evaluate(x_test, y_test)
print("Test accuracy:", test_acc)
```

```
# Обработка ввода от пользователя и предсказание
while True:
    path = input("Введите путь к изображению с рукописной цифрой (или 'exit' для выхода): ")
    if path.lower() == 'exit':
        break
    try:
        img = Image.open(path).convert('L')  # Преобразование в оттенки серого
        img = img.resize((28, 28))  # Изменение размера изображения до 28x28
        img_array = np.array(img) / 255.0  # Преобразование в numpy массив и нормализация
        img_array = np.expand_dims(img_array, axis=0)  # Добавление размерности для предсказания
        prediction = model.predict(img_array)  # Предсказание класса
        digit = np.argmax(prediction)  # Получение индекса класса с максимальной вероятностью
        print("Предсказанная цифра:", digit)
    except Exception as e:
        print("Ошибка:", e)

    Вывод истории обучения модели
    print("Training history:")
    print("Train Loss:", history.history['loss'])
    print("Train Accuracy:", history.history['accuracy'])
    print("Test Loss:", history.history['val_loss'])
    print("Test Accuracy:", history.history['val_accuracy'])

    График точности
    plt.subplot(1, 2, 2)

    plt.plot(history.history['accuracy'], label='Точность тренировочного набора',color='g')
    plt.plot(history.history['val_accuracy'], label='Точность тестового набора',color='y')
    plt.title('Точность')
    plt.xlabel('Эпохи')
    plt.ylabel('Точность')
    plt.legend()
    plt.tight_layout()
    plt.show()
```

Матрица ошибок

```
y_pred = np.argmax(model.predict(x_test), axis=-1)
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
plt.imshow(cm, cmap=plt.cm.Wistia)
plt.title('Матрица')
plt.colorbar()
plt.xticks(np.arange(10))
plt.yticks(np.arange(10))
plt.xlabel('Предсказанные')
plt.ylabel('Настоящие')

for i in range(10):
    for j in range(10):
        plt.text(j, i, str(cm[i, j]), ha='center', va='center')

plt.show()
```

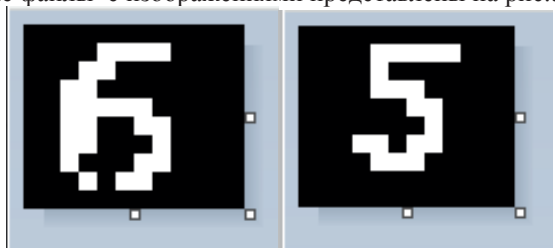
Результат работы программы приведен на рис.4.

```
Epoch 1/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8615 - loss: 0.4747 - val_accuracy: 0.9589 - val_loss: 0.1421
Epoch 2/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9549 - loss: 0.1505 - val_accuracy: 0.9693 - val_loss: 0.1049
Epoch 3/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9677 - loss: 0.1055 - val_accuracy: 0.9737 - val_loss: 0.0847
Epoch 4/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9749 - loss: 0.0844 - val_accuracy: 0.9766 - val_loss: 0.0776
Epoch 5/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9771 - loss: 0.0755 - val_accuracy: 0.9783 - val_loss: 0.0736
Epoch 6/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9798 - loss: 0.0621 - val_accuracy: 0.9799 - val_loss: 0.0671
Epoch 7/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9814 - loss: 0.0562 - val_accuracy: 0.9797 - val_loss: 0.0741
```

```
Epoch 8/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9838 - loss: 0.0489 - val_accuracy: 0.9797 - val_loss: 0.0699
Epoch 9/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9849 - loss: 0.0469 - val_accuracy: 0.9806 - val_loss: 0.0682
Epoch 10/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9864 - loss: 0.0422 - val_accuracy: 0.9797 - val_loss: 0.0704
Epoch 11/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9866 - loss: 0.0408 - val_accuracy: 0.9795 - val_loss: 0.0753
Epoch 12/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9881 - loss: 0.0362 - val_accuracy: 0.9785 - val_loss: 0.0782
Epoch 13/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9883 - loss: 0.0358 - val_accuracy: 0.9806 - val_loss: 0.0675
Epoch 14/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9903 - loss: 0.0302 - val_accuracy: 0.9804 - val_loss: 0.0765
Epoch 15/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9897 - loss: 0.0291 - val_accuracy: 0.9821 - val_loss: 0.0739
Epoch 16/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9898 - loss: 0.0311 - val_accuracy: 0.9806 - val_loss: 0.0770
Epoch 17/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9911 - loss: 0.0261 - val_accuracy: 0.9803 - val_loss: 0.0811
Epoch 18/20
Epoch 18/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9904 - loss: 0.0297 - val_accuracy: 0.9809 - val_loss: 0.0806
Epoch 19/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9914 - loss: 0.0243 - val_accuracy: 0.9804 - val_loss: 0.0821
Epoch 20/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9905 - loss: 0.0265 - val_accuracy: 0.9801 - val_loss: 0.0852
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9770 - loss: 0.1004
Введите путь к изображению с рукописной цифрой (или 'exit' для выхода): C:\Users\user\Desktop\Проекты\5.png
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 86ms/step
Предсказанная цифра: 3
Введите путь к изображению с рукописной цифрой (или 'exit' для выхода): C:\Users\user\Desktop\Проекты\6.png
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
Предсказанная цифра: 6
Введите путь к изображению с рукописной цифрой (или 'exit' для выхода):
```

Рисунок 4 – Результаты работы программы

Использованные файлы с изображениями представлены на рис.5:



6.png 5.png

Рисунок 5 – Использованные изображения

В результате работы программы, нейронная сеть обработала два изображение, одно из которых было обработано верно и на картинке с цифрой 6, сеть распознала 6. Так же нейронная сеть увидела в картинке с цифрой 5, распознала 3. Действительно, на картинке 5 очень похожа на 3, и этот результат можно считать положительным.

На рис.6 приведен фрагмент истории обучения модели и оценка качества модели.

```
Epoch 18/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9984 - loss: 0.0297 - val_accuracy: 0.9809 - val_loss: 0.0806
Epoch 19/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9914 - loss: 0.0243 - val_accuracy: 0.9804 - val_loss: 0.0821
Epoch 20/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9905 - loss: 0.0265 - val_accuracy: 0.9801 - val_loss: 0.0852
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9770 - loss: 0.1004
Test accuracy: 0.9808999760627747
Training history:
Train Loss: [0.2968253357887268, 0.142188934548378, 0.18483663529157639, 0.08593350648888005, 0.07409971207388295, 0.06481489675845013, 0.0587555
Train Accuracy: [0.9141499996185303, 0.9575833082199097, 0.9681166410446167, 0.9744166731834412, 0.9771333336830139, 0.97931665182113365, 0.980449
Test Loss: [0.14213676750659943, 0.10487484186887741, 0.08469078689813614, 0.07755922526212114, 0.07356680929660797, 0.06714369356632233, 0.074131
Test Accuracy: [0.958899974822998, 0.9692999720573425, 0.9736999869346619, 0.9765999913215637, 0.9782999753952026, 0.9799000024795532, 0.97970002
```

Рисунок 6 - Фрагмент истории обучения модели и оценка качества модели

На рис.7 приведены соответственно графики точности тренировочного и тестового наборов

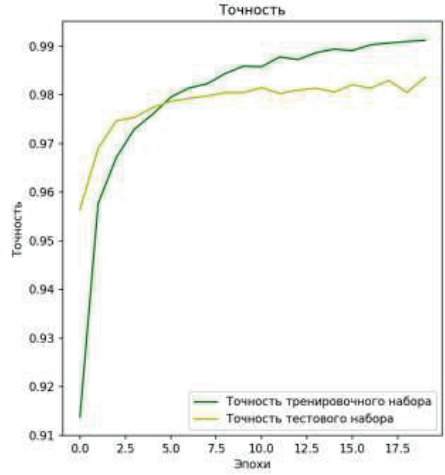


Рисунок 7- Графики точности модели созданной нейросети

На рис.8 приведена матрица ошибок распознавания изображений нейросетью.

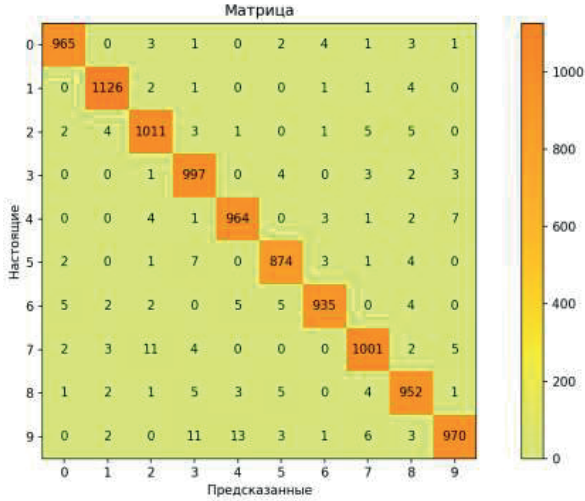


Рисунок 8 - Матрица ошибок распознавания изображений

Дополнительное задание

1. Импортируем проект, где представлены необходимые модули для решения задания. Создаём demo_ai - объект класса TerraIntensive() для обращения к нужным функциям.

В качестве содержимого в файле представлены необходимые бибилотеки для создания графиков, слоёв для нейронных сетей, работы с массивами и описаны необходимые функции для решения задач.

2. Загружаем датасет с автомобилями с помощью функции load_dataset(). В параметрах она принимает строковое значение – наименование базы.

3. Смотрим примеры объектов для обучения с помощью функции samples() (рис.9).

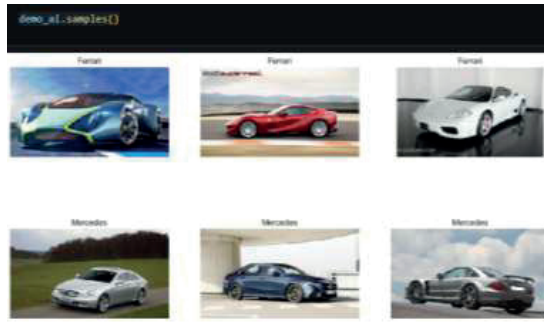


Рисунок 9 - Примеры фотографий для обучения

4. Создаём обучающую и проверочную выборки с метками для дальнейшего обучения нейронной сети с помощью функции create_sets() (рис.10).

```
demo_ai.create_sets()

Создание наборов данных для обучения модели Ok

Размер созданных выборок:
  Обучающая выборка: (3086, 54, 96, 3)
  Метки обучающей выборки: (3086,)
  Проверочная выборка: (341, 54, 96, 3)
  Метки проверочной выборки: (341,)
```

Рисунок 10 - Создание выборок

Графическое представление распределения по классам выборок приведено на рис.11.

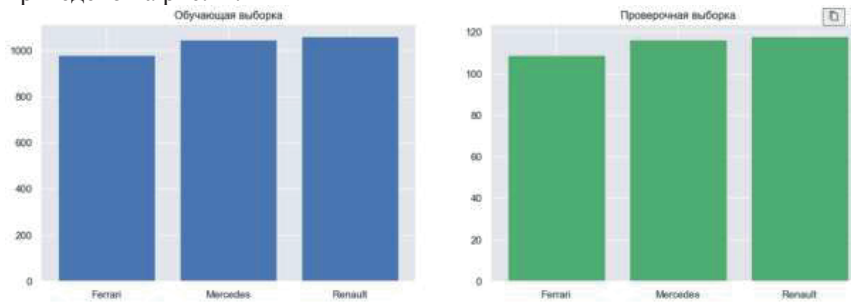


Рисунок 11.- Графическое представление распределения по классам выборок

5. Задаём слои для нейронной сети и создаём модель с помощью функции create_model(), где в параметрах указана строка особого формата – последовательность слоёв.

Обратить внимание на содержимое файла intensivdayone.py и посмотреть, что происходит. Содержимое функции create_model().

Функция create_layers, которая по заданным параметрам создаёт нужные слои нейронной сети.

6. Содержимое функции create_sets() - представлены следующие типы слоёв:

- входной слой (для поступления входных параметров),
- полносвязный слой (для принятия решений),
- выравнивающий слой(для коррекции результатов),
- свертойный2д слой (для извлечения признаков),
- слой максимального объединения (для оптимизации размерности, его мы дописали сами).

Использовать функцию активации Relu.

Опытным путём найти последовательности слоёв.

Например, *"сверточный2д-32-3 объединение сверточный2д-64-3 объединение выравнивающий полносвязный-128 полносвязный -3».*

7. Выполнить тренировку модели с помощью функции train_model(), где в параметрах содержится целое число – количество эпох обучения.

8. Протестировать модель с помощью функции test_model().

Примеры графика представление динамики обучения приведены соответственно на рис.12.

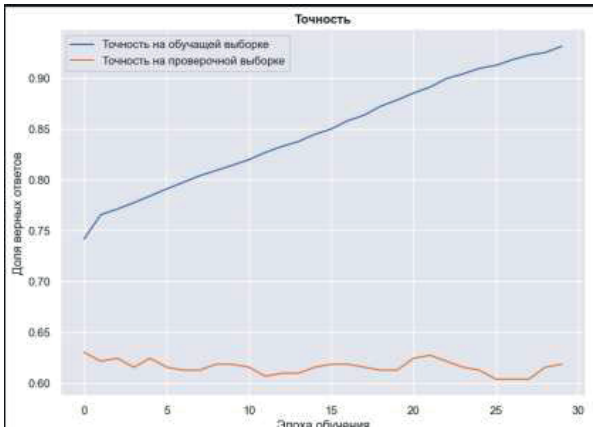
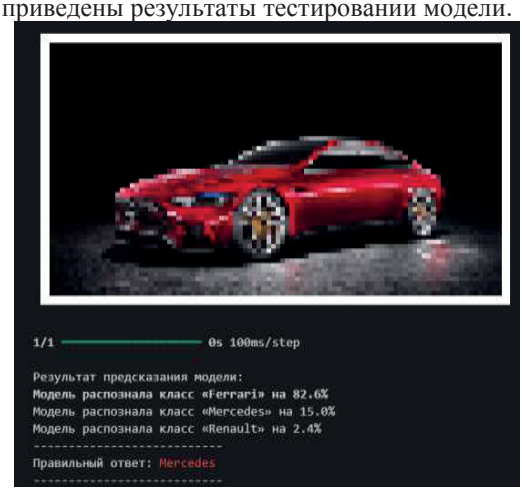
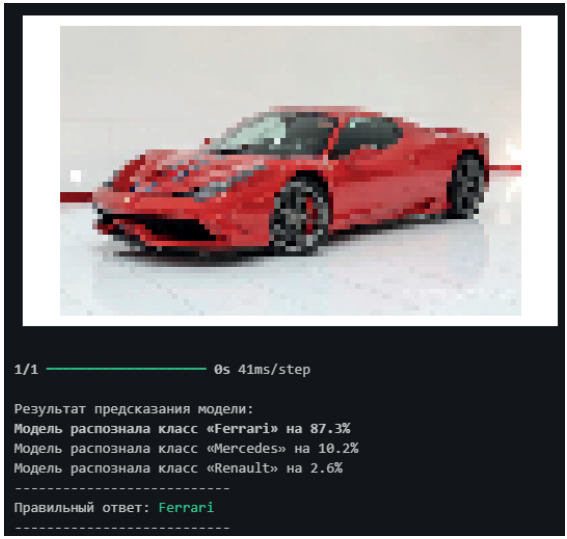


Рисунок 12 - Графическое представление динамики обучения

На рис.13а, б приведены результаты тестировании модели.





б)

Рисунок 13 -Результаты тестировании модели

Примерно в 40% случаев нейросеть ошибается в своих прогнозах. Связано это с тем, что сами по себе фото, в большинстве случаев, похожи, различные отличающие знаки при сжатии картинок хуже отслеживаются при обучении

2.5. Вопросы к защите лабораторной работы

1) Что понимается под термином «нейронная сеть»?
2) Перечислите наборы средств для создания, инициализации, обучения сети.
3) Перечислите эпохи машинного обучения нейронной сети.
4) Какие факторы необходимо принимать во внимание при создании системы на основе нейронной сети?
5) Поясните алгоритм распознавания изображения.
6) Какие параметры и особенности работы сверточных сетей обеспечивают эффективность их работы при решении задач по распознаванию объектов?
7) Что такое batch_size и как он влияет на точность обучения?
8) Как следует делить данные на обучающую и тестовую выборку?
9) Назначение полносвязанного слоя Dense.
10) Для чего используется функция активации?
11) Перечислите активационные функции.

12) Чем они различаются?
13) One hot encoding как и для чего применяется?
14) Что такое сеть прямого распространения (Sequential)?
15) Поясните результаты распознавания картинок.
16) По каким параметрам можно оценить эффективность работы сетей, обученных по разным датасетам?

2.6. Список рекомендуемой литературы

1) Хайкин С. Нейронные сети. Полный курс – 2-е изд. Пер. с англ. – М.: Издательский дом Вильямс, 2006. – 1104 с. [Эл. ресурс]URL: https://books.google.ru/books?id=LPMr0iA0muwC&printsec=copyright&hl=ru&source=gbs_pub_info_r#v=onepage&q&f=false Режим доступа: свободный .
2. Рутковская Д., Пилиньский М., Рутковский Л. Нейронные сети, генетические алгоритмы и нечеткие системы. Пер. с польского И.Д. Рудинского – 2-е изд., стереотип.-М.: Горячая линия - Телеком. 2014.-384 с.
3. Каллан Р. Основные концепции нейронных сетей – Издательство: Вильямс, 2002 г.
4. Романчева Н.И. Машинное обучение и нейронные сети. Нейронные сети/ Учебное пособие.- М.: МГТУ ГА, 2025.- 80с.

3. ЛАБОРАТОРНАЯ РАБОТА №4(2) ОБУЧЕНИЕ С ПОДКРЕПЛЕНИЕМ. АЛГОРИТМ Q-LEARNING НА ПРИМЕРЕ РЕШЕНИЯ КЛАССИЧЕСКОЙ ЗАДАЧИ CART POLE

3.1. Цель работы

Целью данной работы является получение практических навыков реализации алгоритма Q-обучения для решения классической задачи управления "CartPole" с помощью OpenAI Gym.

3.2. Задание на выполнение работы

1. Написать и отладить программу реализации алгоритма Q-обучения для решения классической задачи управления "CartPole" с помощью OpenAI Gym:

- агенту необходимо управлять тележкой, чтобы поддерживать вертикальный шест на ней в равновесии;
- среда предоставляет информацию о состоянии системы, включая положение и скорость тележки, а также угол и угловую скорость шеста;
- агент выбирает между двумя действиями: толкать тележку влево или вправо.

2. Для реализации программы использовать библиотеки gym, numpy, math, matplotlib и pandas.

3. Начальные значения Q-таблицы устанавливаются на нуле.

4. Применяется алгоритм Q-обучения, включая выбор действия по эпсилон-жадной стратегии, обновление Q-значений и постепенное уменьшение параметров (эпсилон и скорость обучения) со временем.

3.3. Основные теоретические сведения и приемы работы

3.3.1. Основные понятия

Алгоритм Q-обучения - это один из методов обучения с подкреплением, который позволяет агенту изучать оптимальные стратегии поведения в заданной среде.

Обучение с подкреплением (англ. reinforcement learning) - способ машинного обучения, при котором система обучается, взаимодействуя с некоторой средой

В обучении с подкреплением существует агент (agent) взаимодействует с окружающей средой (environment), предпринимая действия (actions).

Окружающая среда:

- дает награду (reward) за эти действия,
- а агент продолжает их предпринимать.

Среда формулируется как марковский процесс принятия решений (МППР) с конечным множеством состояний. Следовательно, алгоритмы обучения с подкреплением тесно связаны с динамическим программированием Вероятности выигрышей и перехода состояний в МППР обычно являются величинами случайными, но стационарными в рамках задачи.

При обучении с подкреплением не предоставляются верные пары "входные данные-ответ", а принятие субоптимальных решений (дающих локальный экстремум) не ограничивается явно.

Обучение с подкреплением пытается найти компромисс между исследованием неизученных областей и применением имеющихся знаний (exploration vs exploitation).

3.3.2. Алгоритм взаимодействие агента со средой

Формально простейшая модель обучения с подкреплением состоит из:

-множества состояний окружения (states) S;

-множества действий (actions) A;

-множества вещественозначных скалярных "выигрышей" (rewards) (рис.14).

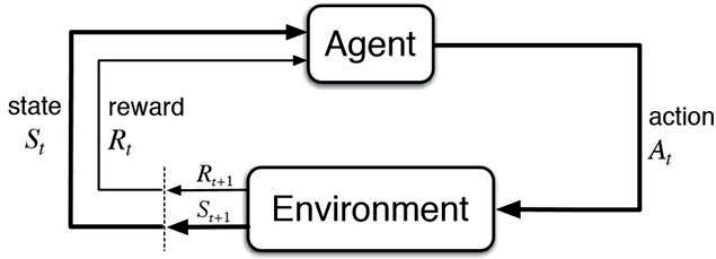


Рисунок 14 - Простейшая модель обучения с подкреплением

В произвольный момент времени t агент характеризуется состоянием $s_t \in S$,

и множеством возможных действий $A(s_t)$.

Выбирая действие

$$a \in A(s_t)$$

он переходит в состояние s_{t+1}, и получает выигрыш r_t .

Основываясь на таком взаимодействии с окружающей средой, агент, обучающийся с подкреплением, должен выработать стратегию

$$\pi: S \to A,$$

которая максимизирует величину в случае МППР, имеющего терминальное состояние:

$$R = r_0 + r_1 + \cdots + r_n,$$

или величину для МППР без терминальных состояний

$$\mathrm{R} = \sum_t \gamma^t r_t,$$

где $0 \le \gamma \le 1$ - дисконтирующий множитель для "предстоящего выигрыша".

Таким образом, обучение с подкреплением особенно хорошо подходит для решения задач, связанных с выбором между долгосрочной и краткосрочной выгодой.

Наивный подход к решению этой задачи подразумевает следующие шаги:

- опробовать все возможные стратегии;
- выбрать стратегию с наибольшим ожидаемым выигрышем

Первая проблема такого подхода заключается в том, что количество доступных стратегий может быть очень велико или бесконечно.

Вторая проблема возникает, если выигрыши стохастические -чтобы точно оценить выигрыш от каждой стратегии потребуется многократно применить каждую из них.

Этих проблем можно избежать, если допустить некоторую структуризацию и, возможно, позволить результатам, полученным от пробы одной стратегии, влиять на оценку для другой.

Двумя основными подходами для реализации этих идей являются оценка функций полезности и прямая оптимизация стратегий.

Подход с использованием функции полезности использует множество оценок ожидаемого выигрыша только для одной стратегии π (либо текущей, либо оптимальной). При этом пытаются оценить либо ожидаемый выигрыш, начиная с состояния S , при дальнейшем следовании стратегии π:

$$\mathrm{V(s)} = \mathrm{E[R|s,}\ \pi]$$

Либо ожидаемый выигрыш, при принятии решения a в состоянии s и дальнейшем наблюдении π,

$$\mathrm{Q(s,a)} = \mathrm{E[R|s,}\ \pi\mathrm{,\ a]}.$$

Если для выбора оптимальной стратегии используется функция полезности Q, то оптимальные действия всегда можно выбрать как действия, максимизирующие полезность. Если используется функция V, то необходимо либо иметь модель окружения в виде вероятностей

$$\mathrm{P[s`|s,a]},$$

что позволяет построить функцию полезности вида

$$Q(s,a) = \sum_{s`} \mathrm{V(s`)\,P(s`|\,s,a)}$$

3.3.3. Q-обучение

Q -learning - это безмодельный алгоритм обучения с подкреплением, позволяющий узнать ценность действия в определенном состоянии.

На основе получаемого от среды вознаграждения агент формирует функцию полезности, что впоследствии дает ему возможность уже не случайно выбирать стратегию поведения, а учитывать опыт предыдущего взаимодействия со средой.

Одно из преимуществ Q-обучения - возможность сравнить ожидаемую полезность доступных действий, не формируя модели окружающей среды. Применяется для ситуаций, которые можно представить в виде МППР.

Таким образом, алгоритм - это функция качества от состояния и действия:

$$Q:S \times A \rightarrow \| R.$$

Перед обучением Q инициализируется случайными значениями. После этого в каждый момент времени t агент выбирает действие a_t, получает награду r_t , переходит в новое состояние s_{t+1}, которое может зависеть от предыдущего состояния s_t и выбранного действия, и обновляет функцию Q.

Обновление функции использует взвешенное среднее между старым и новым значениями:

$$Q^{new}(s_t, \alpha_t) \leftarrow (1-\alpha) \bullet Q(s_t, \alpha_t) + \alpha \bullet (r_t + \gamma \bullet \max_\alpha Q(s_{t+1}, \alpha))$$

где r_t - это награда, полученная при переходе из состояния s_t в состояние s_{t+1}; α - это скорость обучения ($0 < \alpha \leq 1$); $Q(s_t, \alpha_t)$ - старое значение; γ - коэффициент дисконтирования - ускоряет сходимость RL-алгоритмов, ставя в приоритет краткосрочные вознаграждения; $\max_\alpha Q(s_{t+1}, \alpha)$ - оценка оптимальной будущей награды, $r_t + \gamma \bullet \max_\alpha Q(s_{t+1}, \alpha)$ - усвоенная ценность.

Алгоритм заканчивается, когда агент переходит в терминальное состояние s_{t+1}.

3.3.4. Жадная (greedy) стратегия

Эпсилон-жадный алгоритм - это базовый подход к обучению с подкреплением, который позволяет агентам ИИ ориентироваться в сложных средах, балансируя между исследованием и использованием. Это достигается за счёт введения случайности (с вероятностью эпсилон) в процесс принятия решений, при этом в большинстве случаев используются наиболее известные действия. Такой подход обеспечивает адаптивность, эффективность обучения и оптимальное принятие решений, что делает его важным инструментом для современных систем ИИ-агентов.

В отличие от статичных стратегий принятия решений, этот алгоритм позволяет агентам ИИ обучаться динамически. Он сочетает в себе попытки

новых действий (исследование) с использованием известных оптимальных действий (эксплуатация) для максимизации совокупного вознаграждения

Эпсилон-жадная стратегия (Greedy strategy) - метод выбора действий в алгоритмах обучения с подкреплением, который помогает сбалансировать исследование неизвестных действий с эксплуатацией уже известных действий, которые приносят большую награду.

Эта стратегия особенно эффективна в динамичных средах, таких как онлайн-реклама, игры и персонализированные рекомендации, обеспечивая надёжное обучение и адаптацию с течением времени.

Алгоритм «Эпсилон – жадность» использует простую, но эффективную стратегию принятия решений, объединяющую исследование и использование:

- *Исследование:* с вероятностью эпсилон агент намеренно выбирает случайное действие. Этот шаг позволяет агенту исследовать новые возможности и потенциально находить лучшие варианты, которые, возможно, ещё не рассматривались.
- *Эксплуатация:* с вероятностью 1-эпсилон агент полагается на свои существующие знания, выбирая наиболее известное действие на основе прошлого опыта. Это гарантирует, что агент сосредоточится на максимизации вознаграждения за стратегии, которые уже доказали свою эффективность.

Балансируя между этими стратегиями, алгоритм позволяет избежать неоптимальных решений и обеспечивает постоянное совершенствование.

Алгоритм Epsilon Greedy популярен из-за его простоты и эффективности в балансировании между исследованием и эксплуатацией. Однако, как и любой другой метод, он имеет свои недостатки. В табл.1 приведены основные преимущества и недостатки.

Таблица 1. – Обзор преимуществ и недостатков алгоритма Epsilon Greedy

Преимущества	**Недостатки**
Простой в реализации и понимании	Требуется тщательная настройка параметра epsilon.
Эффективен в динамичных средах	Может привести к краткосрочной неэффективности в ходе разведки.
Обеспечивает хороший баланс между разведкой и эксплуатацией	Риск локальных оптимумов без достаточной разведки.

Несмотря на свою эффективность, алгоритм «Эпсилон-жадность» можно усовершенствовать, чтобы добиться ещё более высоких результатов:

- Уменьшение эпсилон: постепенно снижает скорость исследования с течением времени по мере накопления опыта агентом;

- Оптимизация инициализации: для стимулирования первоначального шага сначала используется оптимальная оценка;
- Адаптивные стратегии: для динамического поиска на основе неопределённости использовать такие методы, как верхняя граница достоверности (UCB).

В лабораторной работе данная стратегия реализуется с использованием параметра, который определяет вероятность того, что агент выберет случайное действие вместо действия, максимизирующего ожидаемую награду.

Подход работает следующим образом:

- с вероятностью агент выбирает случайное действие из всего доступного множества действий. Это способствует исследованию, позволяя агенту узнать о возможных наградах от действий, которые он ранее не выбирал;

- с вероятностью агент выбирает действие, которое имеет максимальное ожидаемое значение награды на основе текущих оценок, полученных из Q-таблицы или другой функции оценки. Это называется эксплуатацией, поскольку агент использует свои текущие знания для получения максимальной награды (рис.15).

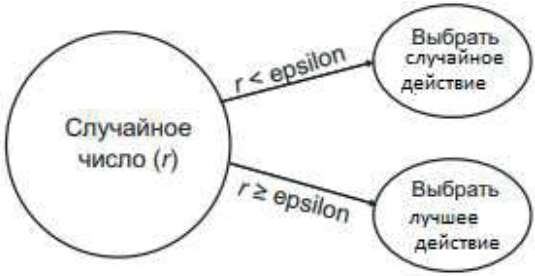


Рисунок 15 – Эпсилон- жадная стратегия

В контексте Q-обучения (Q-learning) жадная стратегия играет ключевую роль в процессе обучения агента, помогая ему находить баланс между исследованием (exploration) и использованием (exploitation) доступной информации об окружающей среде.

Эта стратегия помогает агенту:

- не только опираться на уже известные стратегии для максимизации награды,
- но и исследовать новые действия, которые могут привести к ещё лучшим результатам в долгосрочной перспективе.

3.3.5. Задача «Инвертированный маятник на тележке» (Cart Pole)

Задача Cart Pole - классической задачей обучения с подкреплением. Это модельная задача, представляющая собой систему, где тележка движется без

трения по горизонтальной оси и поддерживает вертикальный стержень (маятник), который может свободно вращаться вокруг точки сочленения.

Цель состоит в том, чтобы поддерживать маятник в вертикальном положении, двигая тележку влево или вправо (рис. 16).

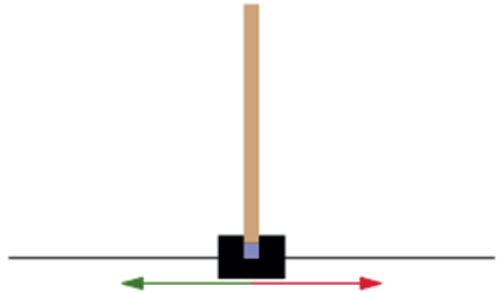


Рисунок 16 - Задача Cart Pole

Рассмотрим основные условия и переменные данной задачи:

- Тележка - может двигаться влево и вправо по горизонтальной оси;
- Маятник - установлен на тележке таким образом, что может свободно вращаться вокруг оси, проходящей через точку сочленения;
- Цель - стабилизировать маятник в вертикальном положении над тележкой, предотвращая его падение;
- Награда – агент получает награду за каждый шаг времени, когда маятник остается в вертикальном положении;
- Среда задается состоянием, действием, наградой, начальным состоянием и флагом завершения эпизода;
- Действия – это ndarray(N-размерный массив), который может принимать значения {0, 1}, указывающие на направление фиксированной силы, с которой толкается тележка:

Num	Action
0	Толкайте тележку влево
1	Толкайте тележку вправо

- Состояние - представляет собой ndarray (N-размерный массив) со значениями:

Num	Наблюдаемые состояния	Min	Max
0	Положение тележки	-4.8	4.8
1	Скорость тележки		
2	Угол обратного маятника	-0.418rad (-24°)	0.418rad (-24°)
3	Угловая скорость обратного маятника	- ∞	∞

Вознаграждение: присваивается 1 за каждый сделанный шаг, включая завершающий;

Начальное состояние - начальное состояние задается при помощи датчика равномерно распределенных случайных чисел в диапазоне (-0,05, 0,05);

Завершение эпизода: - угол шеста вышел из диапазона [-12°, 12°]; позиция тележки вышла из допустимого диапазона [-2.4, 2.4];

длина эпизода превышает 500.

3.3.5. Порядок работы

1. Импортировать необходимые библиотеки:

- gym, numpy, math, matplotlib.pyplot и pandas для работы со средой, массивами, математическими операциями, построения графиков и работы с данными соответственно.

2. Создание среды CartPole:

- Инициализировали среду CartPole-v1 для обучения агента.

3. Дискретизация пространства состояний:

- для удобства использования в Q-таблице дискретизировать пространство состояний среды CartPole.

4. Инициализация Q-таблицы:

- создать Q-таблицу, где каждая строка соответствует дискретизированному состоянию, а каждый столбец - возможному действию.

5. Параметры обучения:

- определить параметры обучения: эпсилон для эпсилон-жадной стратегии, α для скорости обучения и γ для коэффициента дисконтирования.

6. Функция для дискретизации состояний:

- написать функцию, которая принимает непрерывные значения состояний и возвращает их дискретизированные индексы для использования в Q-таблице.

7. Алгоритм Q-обучения:

- в цикле обучения агент выбирает действие согласно эпсилон-жадной стратегии, применяет его к среде, наблюдал новое состояние и награду, обновляет Q-значение согласно алгоритму Q-обучения.

8. Закрытие среды:

-после завершения обучения закрыть среду.

9. Построение графиков:

-Построить графики изменения значения эпсилон и награды за эпизод во время обучения.

10. Вывод Q-таблицы:

-создать DataFrame для отображения Q-таблицы в удобном виде.

Дополнение:

1. Программа использует библиотеку Gym для создания среды "CartPole" и взаимодействия с ней.

2. Состояние среды дискретизируется, чтобы преобразовать непрерывное пространство состояний в дискретное.

3. Q-таблица инициализируется нулями.

4. Применяется алгоритм Q-обучения, включая выбор действия с помощью эпсилон-жадной стратегии, обновление Q-значений и уменьшение параметров (эпсилон и скорость обучения) с течением времени.

5. Значения эпсилон и общие награды за эпизод сохраняются для последующего анализа.

Текст программы приведен ниже.

```
import gym
import numpy as np
import math
import matplotlib.pyplot as plt
import pandas as pd

# Создаем среду CartPole
env = gym.make('CartPole-v1')

# Дискретизируем пространство состояний
n_buckets = (1, 1, 6, 3)  # Интервалы дискретизации для (положение, скорость, угол, угловая скорость)
#(Кол-во ячеек в таблице считается как (1*1*6*3)*2(кол-во действий) = 18*2 = 36)
state_bounds = list(zip(env.observation_space.low, env.observation_space.high))
state_bounds[1] = [-0.5, 0.5]  # Корректируем границы скорости
state_bounds[3] = [-math.radians(50), math.radians(50)]  # Корректируем границы угловой скорости
```

```
# Инициализируем Q-таблицу
n_actions = env.action_space.n # Действия
q_table = np.zeros(n_buckets + (n_actions,))

# Параметры обучения
# Определение эпсилон-жадной стратегии
epsilon = lambda i: max(0.01, min(1, 1.0 - math.log10((i + 1) / 25)))  # Уменьшение эпсилона с течением времени
# Определение скорости обучения
alpha = lambda i: max(0.01, min(0.5, 1.0 - math.log10((i + 1) / 25)))  # Уменьшение скорости обучения с течением времени
gamma = 0.99  # Коэффициент дисконтирования

# Функция для дискретизации непрерывного пространства состояний
def discretize_state(obs):
    # Преобразование непрерывных значений состояний в дискретные индексы корзин
    ratios = [(obs[i] + abs(state_bounds[i][0])) / (state_bounds[i][1] - state_bounds[i][0]) for i in range(len(obs))]
    discretized_obs = [int(round((n_buckets[i] - 1) * ratios[i])) for i in range(len(obs))]  # Перевод в индексы корзин
    discretized_obs = [min(n_buckets[i] - 1, max(0, discretized_obs[i])) for i in range(len(obs))]  # Гарантирование, что индексы в пределах заданного диапазона
    return tuple(discretized_obs)  # Возврат дискретизированного состояния как кортежа

# Алгоритм Q-обучения
num_episodes = 1000
rewards_per_episode = []  # Список для хранения наград за каждый эпизод
epsilon_values = []  # Список для хранения значений эпсилон
for episode in range(num_episodes):
    state = discretize_state(env.reset())  # Начальное состояние
    done = False
    total_reward = 0  # Общая награда за эпизод
    while not done:
        # Выбор действия с помощью эпсилон-жадной стратегии
        if np.random.random() < epsilon(episode):
            action = env.action_space.sample()  # Случайное действие для исследования
        else:
```

```
            action = np.argmax(q_table[state])  # Использование предсказаний Q-таблицы

        # Применение действия и наблюдение за следующим состоянием и наградой
        next_state, reward, done, _ = env.step(action)
        next_state = discretize_state(next_state)

        # Обновление Q-значения
        best_next_action = np.argmax(q_table[next_state])
        q_table[state][action] += alpha(episode) * (reward + gamma * q_table[next_state][best_next_action] - q_table[state][action])

        state = next_state  # Переход к следующему состоянию
        total_reward += reward

    # Сохранение значения эпсилон и общей награды для построения графиков
    epsilon_values.append(epsilon(episode))
    rewards_per_episode.append(total_reward)

    # Вывод информации о эпизоде
    if (episode + 1) % 100 == 0:
        print("Episode:", episode + 1)

# Закрытие среды
env.close()

# Построение графика значений эпсилон
plt.plot(range(num_episodes), epsilon_values)
plt.title('Epsilon Decay')
plt.xlabel('Episode')
plt.ylabel('Epsilon Value')
plt.show()

# Построение графика наград за эпизод
plt.plot(range(num_episodes), rewards_per_episode)
plt.title('Rewards per Episode')
plt.xlabel('Episode')
plt.ylabel('Total Reward')
plt.show()

# Создание DataFrame для Q-таблицы
```

```
q_table_df = pd.DataFrame(q_table.reshape(-1, n_actions), columns=[f"Action {i}"
for i in range(n_actions)])
q_table_df.index.name = 'State'
print("\nQ-table:")
print(q_table_df)
```

Результаты работы программы представлены ниже.

График изменения значения эпсилон (рис.17) демонстрирует уменьшение вероятности случайного выбора действия в процессе обучения. Это связано с тем, что агент с течением времени больше полагается на свои предсказания.

После каждого эпизода высчитываем общую награду для агента исходя из итоговых параметров системы. График показывает какую награду получил агент за каждый отдельный эпизод

Агент стремиться держать шест в равновесии, чтобы получить максимальную награду (500).

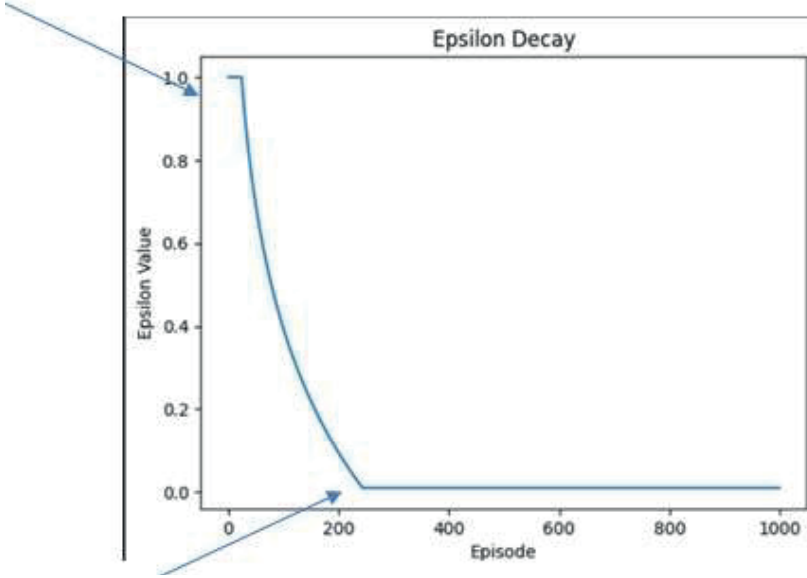


Рисунок 17 – График изменения эпсилон

Снижение наклона (эпсилон) шеста происходит с каждым эпизодом, благодаря эпсилоно-жадной стратегии, используемой в коде

График наград за эпизод отображает, как общая награда за эпизод изменяется в процессе обучения. Цель состоит в том, чтобы максимизировать

эту награду, что указывает на улучшение производительности агента в задаче (рис.18).

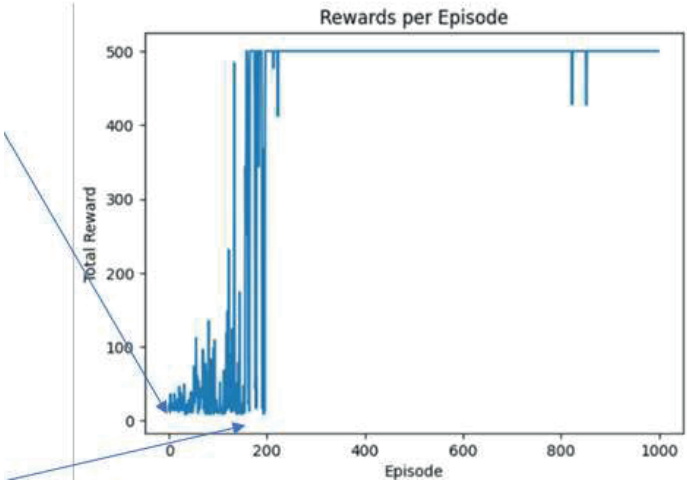


Рисунок 18 – График наград

Этот график показывает, как много наград получал агент на разных этапах. График имеет множество резких скачков около 200-го эпизода. Это связано с тем, что модель почти подобрала веса для состояния равновесия (500), но иногда она его теряет при подборе этих параметров (0).

Из рис.18 видно, что до 160(примерно), он получал очень маленькие награды, что говорит о его случайных и не обдуманных действиях.

В процессе обучения, начиная с примерно 160, агент начинает подбирать более правильные действия и получает за это всё большую награду, но даже, когда он доходит до максимальной награды, имеются выбросы. Это происходит из-за того, что вероятность случайного действия близка к 0, но не равна 0.

Для 3000 эпизодов результаты приведены на рис.19 и рис.20.

В течение 3000 эпизодов алгоритм достигает пика своей производительности примерно после 200 эпизода, однако продолжает исследовать окружающую среду до примерно 2500 эпизода. Это может свидетельствовать о переобучении, поскольку агент продолжает исследовать, несмотря на достижение оптимальной стратегии.

Для 6000 общих наград результаты приведены на рис.20.

Сравнить динамику стратегии обучения при уменьшении числа общих наград (с 6000 до 500) можно, анализируя рис.20 и рис. 21.

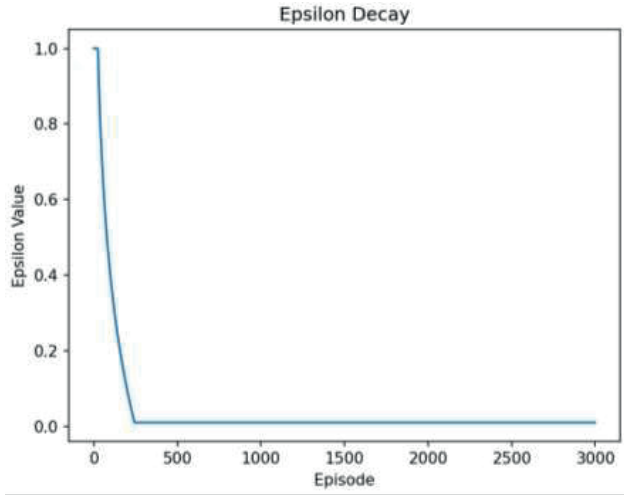


Рисунок 19. – График изменения эпсилон для 3000 эпизодов

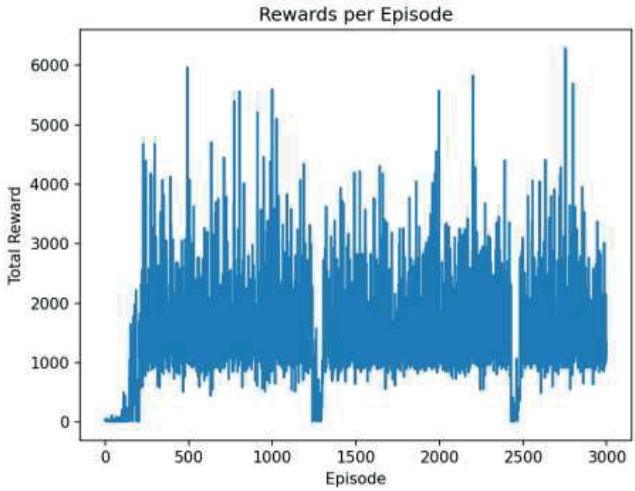


Рисунок 20. – График наград для 3000 эпизодов при общем количестве наград 6000

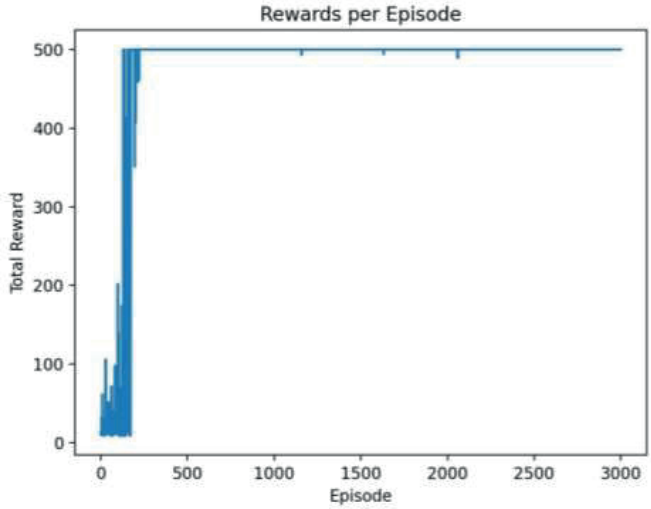


Рисунок 21. – График наград для 3000 эпизодов при общем количестве наград 500

Для 5000 эпизодов результаты представлены соответственно на рис.22. и рис.23.

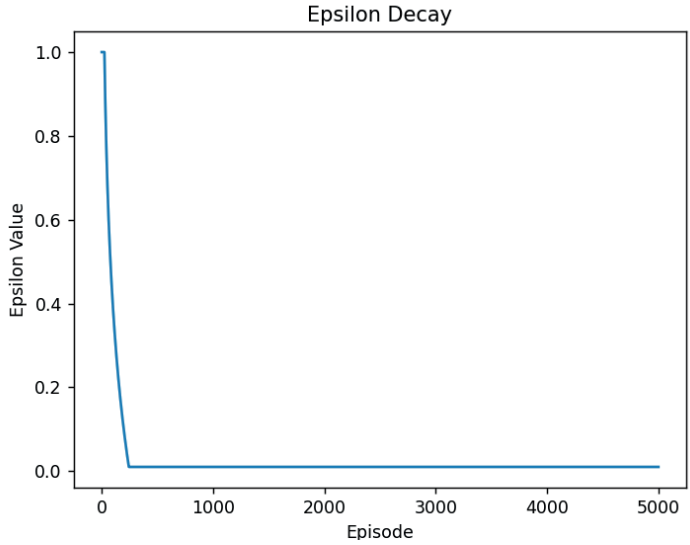


Рисунок 22. – График изменения эпсилон для 5000 эпизодов

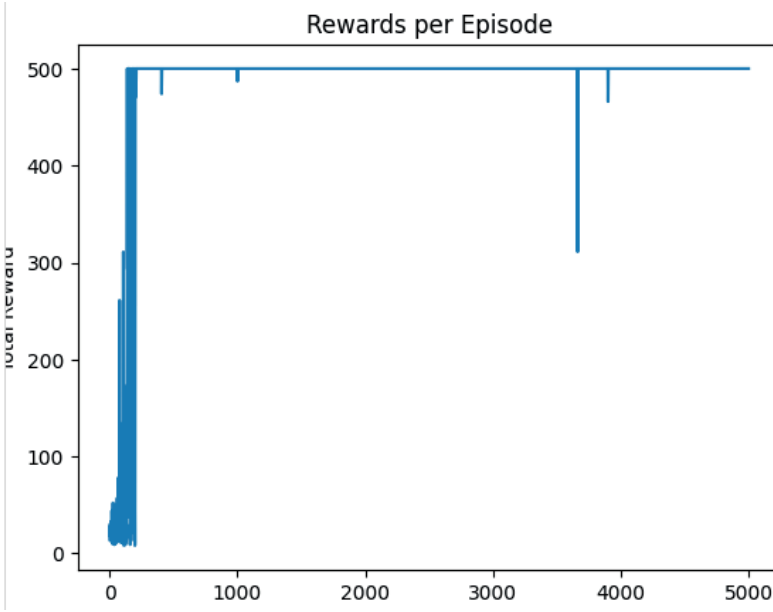


Рисунок 23. – График наград для 5000 эпизодов при общем

На рис. 24 приведена Q- таблица. Таблица показывает возможные состояния параметров системы (State) и действия, предпринимаемые агентом в каждом из состояний, т.е. толчок влево или вправо (Action 0 или Action 1). DataFrame Q-таблицы позволяет визуализировать оценки Q-значений для каждого состояния и каждого возможного действия. Это позволяет анализировать, какие действия предпочтительнее для каждого состояния.

Это можно увидеть на каждом из графиков, что система в этот момент поняла, как поддерживать равновесие.

Модель обучалась путём подбора необходимых действие в каждом из состояний, в котором она могла оказаться.

Таким образом, она может из любого состояния выйти в состояние равновесия путём действий, который она подобрала и занесла в Q-таблицу.

Q-table:

State	Action0	Action1
0	0.000000	0.000000
1	0.000000	0.000000
2	0.000000	0.000000
3	22.388370	29.202378
4	8.293941	26.792655
5	0.000000	0.000000
6	99.999840	99.194777
7	99.999845	99.998925
8	99.933841	99.999844
9	99.999844	99.923182
10	99.998707	99.999845
11	99.406640	99.999840
12	1.116897	0.000000
13	40.765859	30.051761
14	38.670990	41.109025
15	0.000000	0.000000
16	0.000000	0.000000
17	0.000000	0.000000

Рисунок 24 – Q- таблица

3.4. Вопросы к защите лабораторной работы

1) Поясните алгоритм классической задачи обучения с подкреплением.
2) С какой целью проводится анализ процесса обучения сетей и какие действия могут быть предприняты по результатам такого анализа?
3) Перечислите алгоритмы обучения нейронных сетей.
4) Поясните стратегию алгоритма «Эпсилон – жадность».
5) Для чего выполняется дискретизация пространства состояний?
6) Какие параметры используются для оценки качества работы сети?
7) Каковы особенности использования количественных и точностных метрик?
8) Когда применяется метрика и какие она имеет ограничения?
9) Перечислите основные условия и переменные данной задачи.\
10) Поясните назначение использованных библиотек gym, numpy, math, matplotlib и pandas

11) Перечислите параметры запуска созданной программы.
12) Объясните, на что влияют выбранные значения основных параметров.
13) В каком случае проводится процесс дообучения модели сети? В чем он состоит?
14) По каким параметрам можно оценить эффективность обучения сетей разной архитектуры?
15) Как влияет размер датасета на качество обучения нейронных сетей?

3.5. Список рекомендуемой литературы

1) Хайкин С. Нейронные сети. Полный курс – 2-е изд. Пер. с англ. – М.: Издательский дом Вильямс, 2006. – 1104 с. [Эл. ресурс]URL: https://books.google.ru/books?id=LPMr0iA0muwC&printsec=copyright&hl=ru&source=gbs_pub_info_r#v=onepage&q&f=false Режим доступа: свободный .
2. Рутковская Д., Пилиньский М., Рутковский Л. Нейронные сети, генетические алгоритмы и нечеткие системы. Пер. с польского И.Д. Рудинского – 2-е изд., стереотип.-М.: Горячая линия - Телеком. 2014.-384 с.
3. Каллан Р. Основные концепции нейронных сетей – Издательство: Вильямс, 2002 г.
4. Романчева Н.И. Машинное обучение и нейронные сети. Нейронные сети/ Учебное пособие.- М.: МГТУ ГА, 2025.- 80с.