Задача 1

Import tensorflow as tf

From tensorflow.keras.datasets import mnist

*Импорт библиотеки мнист*

**ssl._create_default_https_context = ssl._create_unverified_context**

Этот код устанавливает значение по умолчанию для контекста HTTPS соединения в модуле ssl. Он устанавливает контекст без проверки сертификата, и это может быть полезно в случаях, когда вы хотите установить ненадежное HTTPS соединение. Однако, следует использовать эту функцию только в ограниченных случаях, так как это снижает безопасность вашего приложения.

**(x_train, y_train), (x_test, y_test) = mnist.load_data(path='mnist.npz')**

Этот код загружает набор данных MNIST, который содержит изображения рукописных цифр (0-9) и их соответствующие метки.

Функция mnist.load_data загружает набор данных MNIST из файла mnist.npz и разделяет его на две части: тренировочную (x_train, y_train) и тестовую (x_test, y_test).

Переменная x_train содержит набор изображений для обучения модели, а y_train содержит соответствующие метки (цифры) для этих изображений.

Переменные x_test и y_test содержат набор тестовых изображений и их меток, которые будут использоваться для проверки качества работы модели.

**x_train, x_test = x_train / 255.0, x_test / 255.0**

Этот код выполняет операцию деления. В данном случае, переменные x_train и x_test делятся на

200.0 и 100.0 соответственно.

Таким образом, все элементы массива x_train разделяются на 200.0, а все элементы массива x_test разделяются на 100.0. Это может быть полезно, если необходимо нормализовать данные, чтобы они находились в определенном диапазоне или чтобы ускорить обучение модели машинного обучения.

```
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
# Нормализуем их, преобразуем в нужный формат (например, масштабирование от 0 до 1)
train_images = train_images / 255.0
test_images = test_images / 255.0
```

*Категориальный формат:*

Train_labels=tf.keras.utils.to_**categorials**(train_labels)

Teat_labels=tf.keras.utils.to_**categorials**(test_labels)

Этот код выполняет следующие действия:

1. Импортирует необходимые библиотеки: TensorFlow, NumPy и SSL для работы с SSL сертификатами.

2. Фиксирует ошибку с SSL сертификатами с помощью `_create_default_https_context = _create_unverified_context`, чтобы избежать проблем при загрузке данных из интернета.

3. Загружает датасет MNIST с помощью `mnist.load_data()`. MNIST - это классический датасет, состоящий из изображений рукописных цифр от 0 до 9, каждая цифра представлена как черно-белое изображение размером 28x28 пикселей.

4. Предобрабатывает данные, нормируя значения пикселей изображений к диапазону от 0 до 1. Это делается с помощью деления всех значений на 255.0, так как изначально пиксели имеют значения от 0 до 255 (черный - 0, белый - 255).

Как результат, переменные `x_train` и `x_test` содержат нормализованные значения пикселей изображений, а `y_train` и `y_test` содержат метки классов соответствующих изображений.

**Шаг 2**

```
# Компилируем модель
model.compile(optimizer='adam',- оптимизатор Адам
              loss='sparse_categorical_crossentropy',- функция потерь
              metrics=['accuracy'])- функция метрика
```

**tf.keras.models.Sequential** - это класс, который позволяет создавать последовательные модели, где каждый слой следует за предыдущим в определенном порядке. Последовательная модель - это объект, который представляет собой линейный стек слоев.

Таким образом, данная строка кода создает объект модели типа Sequential, который будет использоваться для добавления слоев в нейронную сеть последовательно.

**tf.keras.layers.Flatten(input_shape=(28, 28)),**

Этот код представляет слой Flatten в модели нейронной сети, который используется для преобразования двумерного массива входных данных (28x28 пикселей, например, изображения) в одномерный массив. В данном случае, указывается input_shape=(28, 28), что означает, что входные данные имеют размерность 28x28. После применения этого слоя, данные будут преобразованы в одномерный массив размером 784 элемента (28 * 28). Это позволяет передать данные в следующие слои нейронной сети, которые требуют одномерный вектор в качестве входа.

**tf.keras.layers.Dense(128, activation='relu'),**

tf.keras.layers.Dense(128, activation='relu') — это команда на языке программирования Python, использующая библиотеку TensorFlow, в

частности ее модуль Keras, для создания полносвязного слоя нейронной сети, который также известен как плотный слой (Dense layer).

Вот что означает каждый компонент этой команды:

- tf.keras.layers.Dense: Это функция, которая создает плотный слой. Плотные слои — это основные блоки для построения слоев нейронной сети, где каждый нейрон связан со всеми нейронами в предыдущем слое.

- 128: Это первый аргумент функции и он определяет количество нейронов в слое. В данном случае слой будет содержать 128 нейронов.

- activation='relu': Это второй аргумент, который определяет функцию активации для нейронов слоя. relu означает "rectified linear unit" и является одной из наиболее популярных функций активации. Она преобразует входное значение в выходное следующим образом: если входное значение положительное, то оно передается без изменений; если отрицательное — заменяется на ноль.

В результате, tf.keras.layers.Dense(128, activation='relu') создает слой из 128 нейронов, каждый из которых использует функцию активации ReLU. Этот слой можно добавить в модель нейронной сети, чтобы помочь в обучении на сложных данных, таких как изображения, звук или текст.

**tf.keras.layers.Dense(10, activation='softmax')**

Команда tf.keras.layers.Dense(10, activation='softmax') используется в библиотеке TensorFlow для создания плотного слоя в нейронной сети, который часто применяется в качестве выходного слоя для классификации.

Вот что означает каждый компонент этой команды:

- tf.keras.layers.Dense: Это функция для создания плотного слоя, где каждый нейрон связан со всеми нейронами в предыдущем слое.

- 10: Это число нейронов в слое. В данном случае, слой будет содержать 10 нейронов, что обычно соответствует количеству классов в задаче классификации.

- activation='softmax': Это функция активации, используемая в выходном слое для многоклассовой классификации. Softmax преобразует вектор чисел в вектор вероятностей, где каждое число представляет вероятность

принадлежности к соответствующему классу. Сумма всех вероятностей в векторе равна 1.

Итак, tf.keras.layers.Dense(10, activation='softmax') создает выходной слой с 10 нейронами, каждый из которых представляет класс, и использует функцию активации softmax для преобразования выходных значений в вероятности. Этот слой обычно используется в конце нейронной сети для задач классификации.

**Задача 3**

**model.compile(optimizer='adam',**

**loss='sparse_categorical_crossentropy',**

**metrics=['accuracy'])**

Команда `**model.compile**` в TensorFlow используется для конфигурации процесса обучения модели. Давайте разберем каждый параметр, указанный в вашем коде:

- `**optimizer='adam'**`: Оптимизатор 'adam' — это один из методов стохастической оптимизации, который используется для обновления весов в нейронной сети на основе обучающих данных. Adam оптимизатор сочетает в себе лучшие свойства алгоритмов AdaGrad и RMSProp, чтобы эффективно находить оптимальные веса.

- `**loss='sparse_categorical_crossentropy**'`: Функция потерь 'sparse_categorical_crossentropy' используется, когда имеются целочисленные метки классов. Она вычисляет ошибку между предсказанными вероятностями и фактическими метками классов и стремится минимизировать эту ошибку в процессе обучения. Это стандартная функция потерь для задач многоклассовой классификации.

- **`metrics=['accuracy']`:** Метрика 'accuracy' используется для оценки производительности модели. В контексте классификации это доля правильных ответов модели на обучающем наборе данных.

Вместе, эти параметры настраивают модель для обучения с использованием оптимизатора Adam, минимизации функции потерь sparse categorical crossentropy и отслеживания точности как показателя производительности модели.

**Задача 4**

**history = model.fit(x_train, y_train, epochs=5)**

Команда model.fit используется в TensorFlow для обучения модели на определенном наборе данных. Вот что означает каждый аргумент в коде:

- x_train: Это данные, на которых модель будет обучаться. Обычно это массив признаков, который модель использует для прогнозирования меток.
- y_train: Это метки классов для x_train, которые модель пытается предсказать. В контексте надзорного обучения это "истинные" значения, которые модель стремится правильно предсказать.
- epochs=5: Это количество эпох обучения. Одна эпоха — это один проход через весь набор данных. Здесь указано, что модель должна пройти через весь набор данных пять раз.

Когда выполняем эту команду, модель начинает процесс обучения, используя данные x_train и метки y_train, пытаясь минимизировать функцию потерь, определенную в model.compile, и улучшая свою точность, также определенную в model.compile.

**Задача 5**

```
# Оценка качества модели
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print(f'Точность модели на тестовом наборе данных: {test_accuracy}')
```

В этом фрагменте кода происходит оценка качества обученной модели на тестовых данных.

Метод **evaluate** используется для оценки функции потерь и метрик модели на тестовых данных. В данном случае тестовые данные (x_test) и соответствующие им метки (y_test) передаются в метод evaluate.

Результатом выполнения этого метода является список, который содержит значения функции потерь и всех метрик модели на тестовых данных. В данном случае возвращаемые значения сохраняются в переменных test_loss и test_accuracy.

**test_loss** - это значение функции потерь модели на тестовых данных. Меньшее значение функции потерь указывает на лучшее качество модели.

**test_accuracy** - это значение метрики точности модели на тестовых данных. Это доля правильно классифицированных примеров. Большее значение точности указывает на лучшее качество модели.

- **Вывод точности модели на тестовом наборе данных**

**print(f'Точность модели на тестовом наборе данных: {test_accuracy}')**

**Точность модели на тестовом наборе данных: 0.9763000011444092**

**Задача 6**

```
# Анализ результатов
import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.show()
```

**Данный график показывает, как менялась точность модели на каждой эпохе обучения (по оси X - эпохи, по оси Y - точность)**

5 лаба

**import tensorflow as tf**

**from tensorflow.keras import datasets, layers, models**

- *`import tensorflow as tf`:* Эта строка импортирует библиотеку TensorFlow под псевдонимом `tf`, позволяя использовать функции и классы TensorFlow в коде.

- *`from tensorflow.keras import datasets, layers, models`:* Здесь мы импортируем модули `datasets`, `layers` и `models` из TensorFlow Keras API. Эти модули позволяют загружать данные, создавать слои нейронных сетей и модели соответственно.

**(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()**

- Эта строка загружает датасет CIFAR-10, который содержит изображения объектов, разделенные на 10 классов. Загрузка данных происходит в переменные `train_images`, `train_labels`, `test_images` и `test_labels`, где `train_images` и `train_labels` содержат обучающие изображения и их метки, а `test_images` и `test_labels` содержат тестовые изображения и их метки.

**train_images, test_images = train_images / 255.0, test_images / 255.0**

- В этой строке выполняется нормализация пикселей изображений, приводя их значения к диапазону от 0 до 1 путем деления на 255.

**model = models.Sequential()**

**model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))**

**model.add(layers.MaxPooling2D((2, 2)))**

**model.add(layers.Conv2D(64, (3, 3), activation='relu'))**

**model.add(layers.MaxPooling2D((2, 2)))**

Этот код создает сверточную нейронную сеть с тремя сверточными слоями и два полносвязных слоя. Первый сверточный слой имеет 32 фильтра, каждый размер которого 3x3, следующий слой имеет 64 фильтра. После трех сверточных слоев следует слой подвыборки (MaxPooling2D), который уменьшает размерность выходных данных. Затем данные передаются в два полносвязных слоя перед выходом.

После компиляции модели мы обучаем ее на обучающем наборе данных и оцениваем ее на тестовом наборе данных.

- В этих строках создается модель нейронной сети с использованием Sequential API. Модель состоит из трех *сверточных слоев* (`Conv2D`), каждый из которых с последующим слоем *пулинга* (`MaxPooling2D`). Функция активации в сверточных слоях - ReLU.

Количество сверточных слоев и слоев пулинга в архитектуре нейронной сети обычно выбирается на основе нескольких факторов, таких как сложность задачи, размер входных данных, доступные вычислительные ресурсы и предыдущий опыт. Оптимальное количество слоев зависит от конкретной задачи и контекста.

**model.add(layers.Flatten())**

**model.add(layers.Dense(64, activation='relu'))**

**model.add(layers.Dense(10))**

- Здесь добавляются два полносвязанных слоя (`Dense`). Первый слой плоскости (`Flatten`) преобразует данные из формы матрицы в одномерный массив. Затем добавляется скрытый слой с 64 нейронами и функцией активации ReLU, а также выходной слой с 10 нейронами (по числу классов в датасете CIFAR-10).

**model.compile(optimizer='adam',**

**loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),**

**metrics=['accuracy'])**

1. **Adam (Adaptive Moment Estimation)**: Оптимизатор, который комбинирует идеи из методов Adagrad и RMSprop, используя как экспоненциально убывающее среднее градиентов, так и экспоненциально убывающее среднее квадратов градиентов.

- Здесь модель компилируется с оптимизатором Adam, функцией потерь SparseCategoricalCrossentropy (поскольку метки в датасете представлены в виде целых чисел) и метрикой точности.

**history = model.fit(train_images, train_labels, epochs=10,**

**validation_data=(test_images, test_labels))**

- В этой строке модель обучается на обучающем датасете в течение 10 эпох, с использованием данных валидации из тестового датасета для оценки производительности модели на каждой эпохе. Результаты обучения сохраняются в переменную `history`.

**test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)**

**print('\nTest accuracy:', test_acc)**

- Эта строка оценивает точность модели на тестовом датасете и выводит ее на экран.

Этот код относится к оценке производительности модели машинного обучения на тестовом наборе данных. Давайте разберем его по частям:

1. `model.evaluate`: Это метод модели машинного обучения, который используется для оценки её производительности.

2. `test_images` и `test_labels`: Это тестовые изображения и соответствующие им метки (целевые значения), которые используются для оценки модели. Тестовый набор данных обычно отделен от обучающего набора и используется для проверки модели на данных, которые она не видела во время обучения.

3. `test_loss` и `test_acc`: Это переменные, в которые будут сохранены значения потерь (loss) и точности (accuracy) модели на тестовом наборе данных после оценки.

Таким образом, этот код оценивает модель на тестовом наборе данных и сохраняет значения потерь и точности в переменные `test_loss` и `test_acc` соответственно

**Verbose"** (от английского "verbose" — многословный, излишне подробный) - это параметр или режим работы в программном обеспечении, который определяет уровень подробности вывода информации или сообщений об ошибках, процессах или операциях.

Например, когда вы используете программу или скрипт с параметром "verbose", она может выводить дополнительные сообщения или подробности о том, что происходит во время выполнения. Это может быть полезно для отладки, мониторинга или просто для получения большего количества информации о работе программы.

В контексте машинного обучения, параметр "verbose" часто используется при обучении моделей, чтобы контролировать уровень вывода информации о прогрессе обучения, например, количество эпох, функции потерь, точность и т. д. В зависимости от значения этого параметра, обучающий алгоритм может выводить различные уровни информации, от минимального до максимального.

**import matplotlib.pyplot as plt**

**plt.plot(history.history['accuracy'])**

**plt.title('Model accuracy')**

**plt.ylabel('Accuracy')**

**plt.xlabel('Epoch')**

**plt.show()**

- Здесь используется библиотека Matplotlib для визуализации точности модели во время обучения. График показывает изменение точности модели на обучающем наборе данных на протяжении каждой эпохи.