Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Уральский федеральный университет имени первого Президента России Б. Н. Ельцина»

УТВЕРЖДАЮ
Директор по образовательной деятельности
_____ С.Т. Князев
«7» сентября 2023 г.

Инструменты решения задач искусственного интеллекта

Учебно-методические материалы по направлению подготовки
09.03.03 Прикладная информатика
Образовательная программа «Прикладной искусственный интеллект»

Екатеринбург

РАЗРАБОТЧИКИ УЧЕБНО-МЕТОДИЧЕСКИХ МАТЕРИАЛОВ

Доцент Базовой кафедры «Аналитика больших данных и методы видеоанализа» _____ М.А. Медведев

СОДЕРЖАНИЕ

Лекция №1 Введение в инструменты и фреймворки искусственного интеллекта. Методы оптимизации моделей ИИ

Искусственный интеллект — это область компьютерных наук, которая стремится создать интеллектуальные агенты, способные выполнять задачи, обычно требующие умственного усилия со стороны человека. Важным аспектом разработки искусственного интеллекта является использование подходящих инструментов и фреймворков, которые облегчают процесс создания и обучения моделей, а также ускоряют их внедрение в реальные приложения.

Роль инструментов и фреймворков в разработке искусственного интеллекта

Инструменты и фреймворки играют ключевую роль в разработке искусственного интеллекта по нескольким причинам:

Упрощение разработки

Искусственный интеллект - сложная область, и разработка алгоритмов и моделей может быть трудоемкой задачей. Использование инструментов и фреймворков позволяет снизить сложность задач и предоставляет разработчикам удобные интерфейсы для создания моделей с минимальными усилиями.

Ускорение процесса обучения

Искусственный интеллект требует множество итераций и обучения моделей на больших объемах данных. Фреймворки предоставляют оптимизированные реализации алгоритмов и использование параллельных вычислений, что значительно ускоряет процесс обучения.

Переносимость и масштабируемость

Использование фреймворков позволяет создавать модели, которые легко переносить на различные платформы и масштабировать для работы с большими объемами данных.

Общие решения и сообщество

Популярные фреймворки имеют активное сообщество разработчиков и большое количество общих решений, что помогает быстрее решать проблемы и находить ответы на вопросы.

Обзор популярных фреймворков искусственного интеллекта

Существует множество фреймворков для разработки искусственного интеллекта, но давайте рассмотрим некоторые из самых популярных:

TensorFlow

TensorFlow разработан компанией Google и является одним из самых популярных фреймворков для глубокого обучения. Он обладает богатой функциональностью и хорошо подходит для обучения сложных нейронных сетей.

PyTorch

PyTorch, разработанный Facebook, также предоставляет мощные инструменты для создания нейронных сетей. Он отличается динамическим вычислением, что упрощает процесс отладки моделей.

Scikit-learn

Scikit-learn является одной из наиболее популярных библиотек машинного обучения в Python. Он предоставляет множество алгоритмов и утилит для обработки данных и создания моделей машинного обучения.

Keras

Keras - это высокоуровневый API, построенный поверх TensorFlow, который облегчает создание нейронных сетей с простым синтаксисом.

MXNet

MXNet предоставляет гибкую и эффективную платформу для обучения моделей и работает на различных архитектурах, включая GPU и CPU.

Давайте рассмотрим подробнее преимущества и недостатки каждого из упомянутых фреймворков: TensorFlow, PyTorch, Scikit-learn, Keras и MXNet.

TensorFlow

Преимущества:

1. Поддержка различных аппаратных платформ: TensorFlow обладает высокой переносимостью и может использоваться на различных аппаратных устройствах, включая CPU, GPU и TPU (Tensor Processing Units).
2. Большое сообщество и документация: TensorFlow имеет огромное и активное сообщество разработчиков, что упрощает поиск ответов на вопросы и решение проблем.
3. Расширяемость: Фреймворк предоставляет возможность создавать и переиспользовать пользовательские операции и слои, что делает его гибким инструментом для исследования и разработки.

Недостатки:

1. Более сложный синтаксис: TensorFlow имеет несколько более громоздкий синтаксис по сравнению с некоторыми другими фреймворками, что может сделать его использование более сложным для начинающих.
2. Версионная совместимость: Обновления TensorFlow иногда могут приводить к проблемам совместимости с предыдущими версиями кода.

PyTorch

Преимущества:

1. Динамическое вычисление: Одним из главных преимуществ PyTorch является его поддержка динамического вычисления, которое облегчает отладку и экспериментирование с моделями.

2. Простота использования: PyTorch имеет интуитивный и простой синтаксис, что делает его более привлекательным для начинающих и исследователей.
3. Красивая визуализация графов вычислений: В PyTorch легко визуализировать графы вычислений, что упрощает понимание работы моделей.

Недостатки:

1. Относительно меньшее сообщество: Несмотря на активное развитие, сообщество PyTorch все еще меньше, чем у TensorFlow.
2. Ограниченная поддержка для некоторых аппаратных платформ: PyTorch менее распространен на некоторых аппаратных устройствах, чем TensorFlow.

Scikit-learn

Преимущества:

1. Простота и удобство использования: Scikit-learn предоставляет простой и легкий в использовании интерфейс для обучения моделей машинного обучения.
2. Обширная коллекция алгоритмов: Фреймворк содержит большой набор классических алгоритмов машинного обучения, что делает его полезным для базовых задач искусственного интеллекта.

Недостатки:

1. Ограниченные возможности глубокого обучения: Scikit-learn не обладает таким широким набором возможностей для глубокого обучения, как TensorFlow или PyTorch.
2. Менее оптимизированный для больших объемов данных: Некоторые алгоритмы в Scikit-learn могут быть менее эффективными при работе с большими объемами данных.

Keras

Преимущества:

1. Простота и высокий уровень абстракции: Keras предоставляет высокоуровневый API, что делает его идеальным для быстрого прототипирования и простых задач.
2. Хорошая поддержка предобученных моделей: Керас имеет широкий выбор предобученных моделей, что позволяет быстро создавать и применять модели для различных задач.

Недостатки:

1. Менее гибкий по сравнению с TensorFlow и PyTorch: Keras может быть менее подходящим для сложных исследований и настройки моделей по сравнению с другими фреймворками.

MXNet

Преимущества:

1. Высокая производительность: MXNet обладает хорошей оптимизацией, что делает его высокопроизводительным фреймворком для обучения моделей на больших объемах данных.
2. Гибкость и масштабируемость: MXNet поддерживает различные аппаратные платформы и облегчает масштабирование моделей на большие кластеры серверов.

Недостатки:

1. Менее популярный: MXNet имеет меньшее сообщество разработчиков и пользователей по сравнению с TensorFlow и PyTorch.
2. Некоторые ограничения API: Интерфейс MXNet не всегда настолько удобен и интуитивен, как у некоторых других фреймворков.

H2O

Описание:

H2O - это открытый и распределенный фреймворк с отличной поддержкой машинного обучения и искусственного интеллекта. Он спроектирован таким образом, чтобы упростить и ускорить процесс создания моделей и их внедрение в продакшн. H2O предоставляет широкий набор алгоритмов машинного и глубокого обучения.

Преимущества:

1. Масштабируемость: H2O обладает отличной масштабируемостью и может легко обрабатывать большие объемы данных на кластерах серверов.
2. Интуитивный пользовательский интерфейс: H2O предоставляет удобный интерфейс для создания и обучения моделей, что делает его привлекательным для начинающих и опытных разработчиков.
3. Поддержка различных языков программирования: H2O поддерживает несколько языков программирования, включая Python, R и Java.

Недостатки:

1. Ограниченное количество алгоритмов для глубокого обучения: Несмотря на широкий набор алгоритмов машинного обучения, H2O имеет небольшое количество алгоритмов для глубокого обучения по сравнению с TensorFlow или PyTorch.

CNTK (Microsoft Cognitive Toolkit)

Описание:

CNTK, также известный как Microsoft Cognitive Toolkit, является открытым фреймворком глубокого обучения, разработанным Microsoft. Он спроектирован с упором на высокую производительность и предоставляет эффективные инструменты для создания и обучения глубоких нейронных сетей.

Преимущества:

1. Производительность: CNTK обладает отличной производительностью, что делает его предпочтительным выбором для задач, требующих обработки больших объемов данных.

2. Распределенное обучение: Фреймворк поддерживает обучение на нескольких устройствах и кластерах серверов, что позволяет ускорить процесс обучения на больших объемах данных.
3. Простота использования: CNTK предоставляет удобный API и простой синтаксис, что упрощает создание и настройку моделей.

Недостатки:

1. Ограниченное сообщество: CNTK имеет меньшее сообщество разработчиков и пользователей по сравнению с TensorFlow и PyTorch, что может сказаться на доступности документации и поддержке.
2. Ограниченный выбор предобученных моделей: Количество предобученных моделей для CNTK менее обширно по сравнению с другими популярными фреймворками.

Каждый из этих фреймворков обладает своими уникальными особенностями и подходит для различных задач и контекстов. Выбор фреймворка зависит от ваших потребностей и опыта в работе с ним. Вы можете экспериментировать с разными фреймворками, чтобы найти тот, который лучше всего соответствует целям и задачам проекта.

Методы оптимизации моделей исскуственного интеллекта

Существует довольно большое количество методов оптимизации моделей искусственного интеллекта, которые широко применяются в области машинного обучения и глубокого обучения. Ниже перечислены некоторые из них:

1. Градиентный спуск (Gradient Descent): Это один из наиболее популярных методов оптимизации. Он используется для обновления параметров модели, исходя из значения градиента функции потерь по параметрам. Градиентный спуск сходится к минимуму функции потерь, если правильно настроены шаг обучения и другие параметры.
2. Стохастический градиентный спуск (Stochastic Gradient Descent, SGD): Этот метод является вариантом градиентного спуска, который обновляет параметры модели на каждом примере из обучающего набора данных. Он более быстро сходится, но может быть менее стабильным.
3. Adam (Adaptive Moment Estimation): Adam является комбинацией методов градиентного спуска с моментом и адаптивного шага обучения. Он эффективно адаптирует скорость обучения для каждого параметра на основе наблюдаемой статистики градиентов.
4. RMSprop (Root Mean Square Propagation): Этот метод оптимизации также адаптивно регулирует скорость обучения для каждого параметра, используя среднеквадратичные значения градиентов.
5. Адаптивный градиентный спуск (Adagrad): Этот метод адаптивно масштабирует скорость обучения для каждого параметра на основе истории градиентов этого параметра. Он предназначен для обработки разреженных данных.
6. Адаптивный моментный адаград (Adadelta): Это расширение метода Adagrad, которое старается устранить недостаток ограничения обучающей скорости.
7. Нестеровский ускоренный градиентный спуск (Nesterov Accelerated Gradient, NAG): Этот метод градиентного спуска предварительно смещает градиенты перед обновлением параметров, что позволяет быстрее сходиться к оптимуму.

8. AdamW: Это модификация метода Adam, которая добавляет регуляризацию весов (weight decay) для более стабильного обучения.
9. Генетический алгоритм (Genetic Algorithm): Это эволюционный метод оптимизации, основанный на принципах биологической эволюции. Он может использоваться для глобальной оптимизации параметров моделей искусственного интеллекта.
10. Эволюционное программирование (Evolutionary Programming): Еще один эволюционный метод оптимизации, который применяется для настройки параметров моделей.

И в искусственном интеллекте и машинном обучении существует множество других методов, каждый из которых имеет свои преимущества и недостатки, и может быть применим в различных сценариях. Выбор метода оптимизации зависит от конкретной задачи и структуры модели.

Лабораторная работа №1 Поиск в сети Интернет и исследование фреймворков для работы с искусственным интеллектом

Задание: Поиск в сети Интернет и исследование фреймворков для работы с искусственным интеллектом

Цель: Ознакомиться с различными фреймворками и инструментами, используемыми для разработки искусственного интеллекта.

Необходимые шаги:

1. Найти в сети Интернет популярные фреймворки для работы с искусственным интеллектом за исключением тех, которые представлены в лекции №1.
2. Изучить каждый из фреймворков.
3. Провести исследование и сравнение фреймворков по следующим критериям:
   - Основные характеристики и возможности фреймворка (поддерживаемые языки программирования, функциональность, типы моделей, алгоритмы и т. д.).
   - Преимущества и недостатки каждого фреймворка.
   - Реальные примеры применения фреймворка в различных сферах (например, медицина, финансы, робототехника и т. д.).
   - Различия между фреймворками, которые делают их уникальными и подходящими для определенных задач.
4. Подготовить отчет или презентацию, в которой представить полученные результаты и выводы о каждом фреймворке.
5. Провести обсуждение и обмен мнениями с другими студентами о том, какой фреймворк может быть наиболее подходящим для различных типов задач и проектов.

Ресурсы:

- Официальные сайты фреймворков.
- Документация и руководства пользователя.
- Научные статьи, блоги и сообщества разработчиков, где можно найти реальные примеры применения фреймворков и отзывы пользователей.

Примечание: При выполнении задания студенты могут использовать как англоязычные, так и русскоязычные ресурсы. Важно, чтобы результаты были описаны собственными словами без копирования источников в сети.

Лекция №2 TensorFlow

TensorFlow — это открытый и распределенный фреймворк для машинного обучения и глубокого обучения, разработанный командой Google Brain. Он был представлен в 2015 году и с тех пор стал одним из наиболее популярных инструментов в области искусственного интеллекта.

Основные функции и приложение фреймворка TensorFlow:

- Машинное обучение: TensorFlow предоставляет широкий набор инструментов и алгоритмов для обучения различных типов моделей, включая нейронные сети, линейные модели, решающие деревья, и другие. Он поддерживает как обучение с учителем, так и без учителя.
- Глубокое обучение: TensorFlow предоставляет гибкие инструменты для создания и обучения глубоких нейронных сетей различных архитектур, включая сверточные нейронные сети (CNN), рекуррентные нейронные сети (RNN), и трансформеры, используемые в задачах обработки естественного языка.
- Вычислительные графы: TensorFlow использует вычислительные графы для представления и выполения операций. Это позволяет оптимизировать и распределить вычисления на различных устройствах, таких как CPU, GPU и TPU, что обеспечивает высокую производительность.
- Распределенное обучение: TensorFlow поддерживает обучение на нескольких устройствах и на кластерах серверов, что позволяет ускорить процесс обучения на больших объемах данных.

Варианты использования и особенности TensorFlow:

TensorFlow может быть использован для широкого спектра задач, включая:

- Классификация и распознавание образов: TensorFlow часто используется для создания моделей, которые классифицируют и распознают объекты на изображениях и видео.
- Обработка естественного языка (Natural Language Processing, NLP): TensorFlow применяется в задачах анализа текста, машинного перевода, сентимент-анализа и других NLP-задач.
- Рекомендательные системы: TensorFlow может использоваться для создания персонализированных рекомендательных систем, предлагая пользователям контент или продукты на основе их предпочтений и поведения.
- Обучение с подкреплением: TensorFlow применяется для разработки алгоритмов обучения с подкреплением, где модель обучается на основе опыта, полученного взаимодействием с окружающей средой.

Особенности TensorFlow:

- Кроссплатформенность: TensorFlow поддерживает различные операционные системы, включая Windows, macOS и Linux.
- Поддержка различных языков программирования: TensorFlow предоставляет API для Python, C++, Java, Go и других языков программирования.
- Коммьюнити и поддержка: TensorFlow имеет огромное и активное сообщество разработчиков, что обеспечивает доступ к обширной документации, учебным материалам и библиотекам предобученных моделей.

Функциональность, типы моделей и алгоритмы в TensorFlow:

- Функциональность: TensorFlow предоставляет API для создания, обучения и оценки моделей, включая методы для работы с данными, оптимизацией моделей, сохранением и загрузкой весов моделей и многое другое.
- Типы моделей: TensorFlow поддерживает различные типы моделей, включая плотные нейронные сети, сверточные нейронные сети, рекуррентные нейронные сети, а также модели для обработки графов и последовательностей.
- Алгоритмы: TensorFlow предоставляет широкий набор алгоритмов оптимизации, активации, функций потерь и других инструментов, которые используются для создания и обучения моделей.

TensorFlow является мощным фреймворком с широким набором функциональности и обширным сообществом разработчиков. Он позволяет создавать различные типы моделей для разнообразных задач и является одним из наиболее популярных инструментов в области искусственного интеллекта и машинного обучения.

Реализация алгоритмов с использованием TensorFlow

Ниже представлены несколько примеров реализации различных алгоритмов с использованием TensorFlow:

1. Линейная регрессия:

```
import tensorflow as tf
import numpy as np

# Генерируем сгенерированные данные
np.random.seed(0)
X_train = np.linspace(0, 1, 100)
y_train = 2 * X_train + 1 + np.random.normal(0, 0.1, size=100)

# Создаем граф вычислений в TensorFlow
X = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)

# Задаем переменные модели (веса и смещение)
W = tf.Variable(np.random.randn(), dtype=tf.float32)
b = tf.Variable(np.random.randn(), dtype=tf.float32)

# Задаем модель (линейная регрессия)
y_pred = tf.add(tf.multiply(X, W), b)

# Задаем функцию потерь (средняя квадратичная ошибка)
loss = tf.reduce_mean(tf.square(y_pred - y))

# Задаем оптимизатор (градиентный спуск)
```

```
learning_rate = 0.01
optimizer = tf.train.GradientDescentOptimizer(learning_rate)
train_op = optimizer.minimize(loss)

# Инициализируем граф
init = tf.global_variables_initializer()

# Запускаем сессию TensorFlow
with tf.Session() as sess:
    # Инициализируем переменные
    sess.run(init)

    # Обучение модели
    epochs = 1000
    for epoch in range(epochs):
        _, loss_val = sess.run([train_op, loss], feed_dict={X:
X_train, y: y_train})
        if epoch % 100 == 0:
            print(f"Эпоха {epoch}, функция потерь: {loss_val:.4f}")

    # Получаем обученные значения весов и смещения
    trained_W, trained_b = sess.run([W, b])

# Выводим результаты и график
print(f"Обученные веса: {trained_W:.4f}, обученное смещение:
{trained_b:.4f}")
```

2. Логистическая регрессия:

```
import tensorflow as tf
import numpy as np

# Генерируем сгенерированные данные
np.random.seed(0)
X_train = np.random.randn(100, 2)
y_train = (X_train[:, 0] + X_train[:, 1] > 0).astype(int)

# Создаем граф вычислений в TensorFlow
X = tf.placeholder(tf.float32, shape=[None, 2])
y = tf.placeholder(tf.int32, shape=[None])

# Задаем переменные модели (веса и смещение)
W = tf.Variable(np.random.randn(2, 1), dtype=tf.float32)
b = tf.Variable(np.random.randn(), dtype=tf.float32)
```

```
# Задаем модель (логистическая регрессия)
logits = tf.matmul(X, W) + b
y_pred = tf.squeeze(tf.round(tf.sigmoid(logits)))

# Задаем функцию потерь (кросс-энтропия)
loss =
tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(labels=tf.cast(
y, dtype=tf.float32), logits=logits))

# Задаем оптимизатор (градиентный спуск)
learning_rate = 0.01
optimizer = tf.train.GradientDescentOptimizer(learning_rate)
train_op = optimizer.minimize(loss)

# Инициализируем граф
init = tf.global_variables_initializer()

# Запускаем сессию TensorFlow
with tf.Session() as sess:
    # Инициализируем переменные
    sess.run(init)

    # Обучение модели
    epochs = 1000
    for epoch in range(epochs):
        _, loss_val = sess.run([train_op, loss], feed_dict={X:
X_train, y: y_train})
        if epoch % 100 == 0:
            print(f"Эпоха {epoch}, функция потерь: {loss_val:.4f}")

    # Получаем обученные значения весов и смещения
    trained_W, trained_b = sess.run([W, b])

# Выводим результаты
print(f"Обученные веса: {trained_W}, обученное смещение: {trained_b}")
```

3. Кластеризация с помощью K-means:

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

# Генерируем сгенерированные данные
```

```
np.random.seed(0)
X = np.vstack([np.random.randn(100, 2) + [2, 2],
               np.random.randn(100, 2) + [-2, -2],
               np.random.randn(100, 2) + [2, -2]])

# Создаем граф вычислений в TensorFlow
num_clusters = 3
points = tf.constant(X)
centroids = tf.Variable(tf.slice(tf.random_shuffle(points), [0, 0],
[num_clusters, -1]))

# Расчет ближайших кластеров
expanded_points = tf.expand_dims(points, 0)
expanded_centroids = tf.expand_dims(centroids, 1)
distances = tf.reduce_sum(tf.square(tf.subtract(expanded_points,
expanded_centroids)), 2)
assignments = tf.argmin(distances, 0)

# Обновление кластеров
means = tf.concat([tf.reduce_mean(tf.gather(points,
tf.reshape(tf.where(tf.equal(assignments, c)), [1, -1])),
                                  reduction_indices=[1]) for c in
range(num_clusters)], 0)
update_centroids = tf.assign(centroids, means)

# Инициализируем граф
init = tf.global_variables_initializer()

# Запускаем сессию TensorFlow
with tf.Session() as sess:
    # Инициализируем переменные
    sess.run(init)

    # Обучение модели
    num_iterations = 100
    for step in range(num_iterations):
        _, centroid_values, assignment_values =
sess.run([update_centroids, centroids, assignments])

# Выводим результаты и график
plt.scatter(X[:, 0], X[:, 1], c=assignment_values, s=50, alpha=0.5)
plt.scatter(centroid_values[:, 0], centroid_values[:, 1], c='red',
s=200)
plt.show()
```

В этих примерах мы реализуем линейную регрессию, логистическую регрессию и кластеризацию с помощью K-means с использованием TensorFlow. Каждый пример демонстрирует создание графа вычислений, определение моделей, определение функций потерь и оптимизаторов, обучение моделей и вывод результатов. TensorFlow обеспечивает гибкость и эффективность для реализации различных алгоритмов машинного обучения и глубокого обучения.

Далее рассмотрим более сложные примеры использования TensotFlow для машинного обучения.

Пример использования TensorFlow для обучения простой линейной регрессии на сгенерированных данных.

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

# Создаем сгенерированные данные
np.random.seed(0)
X_train = np.linspace(0, 1, 100)
y_train = 2 * X_train + 1 + np.random.normal(0, 0.1, size=100)

# Создаем граф вычислений в TensorFlow
# Задаем входные данные
X = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)

# Задаем переменные модели (веса и смещение)
W = tf.Variable(np.random.randn(), dtype=tf.float32)
b = tf.Variable(np.random.randn(), dtype=tf.float32)

# Задаем модель (линейная регрессия)
y_pred = tf.add(tf.multiply(X, W), b)

# Задаем функцию потерь (средняя квадратичная ошибка)
loss = tf.reduce_mean(tf.square(y_pred - y))

# Задаем оптимизатор (градиентный спуск)
learning_rate = 0.01
optimizer = tf.train.GradientDescentOptimizer(learning_rate)
train_op = optimizer.minimize(loss)

# Инициализируем граф
init = tf.global_variables_initializer()
```

```
# Запускаем сессию TensorFlow
with tf.Session() as sess:
    # Инициализируем переменные
    sess.run(init)

    # Обучение модели
    epochs = 1000
    for epoch in range(epochs):
        _, loss_val = sess.run([train_op, loss], feed_dict={X:
X_train, y: y_train})
        if epoch % 100 == 0:
            print(f"Эпоха {epoch}, функция потерь: {loss_val:.4f}")

    # Получаем обученные значения весов и смещения
    trained_W, trained_b = sess.run([W, b])

# Выводим результаты и график
print(f"Обученные веса: {trained_W:.4f}, обученное смещение:
{trained_b:.4f}")

plt.scatter(X_train, y_train, label='Исходные данные')
plt.plot(X_train, X_train * trained_W + trained_b, 'r',
label='Модель')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.show()
```

В этом примере мы создаем сгенерированные данные с помощью NumPy, а затем используем TensorFlow для создания графа вычислений, определения модели линейной регрессии, определения функции потерь и оптимизатора. Затем мы обучаем модель на тренировочных данных, используя градиентный спуск. В конце выводим обученные значения весов и смещения, а также строим график сгенерированных данных и обученной модели.

Примечание: Этот пример преднамеренно простой, чтобы продемонстрировать основы использования TensorFlow. В реальных приложениях вы обычно будете работать с более сложными моделями и данными.

Пример классификации изображений с использованием сверточных нейронных сетей (CNN):

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

```
# Загрузка датасета CIFAR-10
(train_images, train_labels), (test_images, test_labels) =
datasets.cifar10.load_data()

# Масштабирование данных и преобразование меток в one-hot векторы
train_images, test_images = train_images / 255.0, test_images / 255.0
train_labels = tf.keras.utils.to_categorical(train_labels,
num_classes=10)
test_labels = tf.keras.utils.to_categorical(test_labels,
num_classes=10)

# Создание модели CNN
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)))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

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

# Обучение модели
model.fit(train_images, train_labels, epochs=10, batch_size=64,
validation_data=(test_images, test_labels))
```

В этом примере мы используем TensorFlow и его высокоуровневый API Keras для создания сверточной нейронной сети (CNN) для классификации изображений из датасета CIFAR-10. Мы загружаем данные, масштабируем их, создаем и компилируем модель, а затем обучаем ее на тренировочных данных.

Пример обработки естественного языка (NLP) с использованием рекуррентных нейронных сетей (RNN):

```
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

```
# Примеры текстовых данных
texts = ['Это пример предложения 1.',
         'А это - пример предложения 2.',
         'И это - пример предложения 3.']

# Создание токенизатора и преобразование текста в последовательности
чисел
tokenizer = Tokenizer()
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)

# Добавление дополнительного нулевого значения для сравнения длины
последовательностей
padded_sequences = pad_sequences(sequences)

# Создание и компиляция простой RNN модели
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(input_dim=len(tokenizer.word_index)+1,
output_dim=8, input_length=len(padded_sequences[0])),
    tf.keras.layers.SimpleRNN(16),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Обучение модели
labels = [1, 0, 1]
model.fit(padded_sequences, labels, epochs=10)
```

В этом примере мы используем TensorFlow и его высокоуровневый API Keras для обработки естественного языка (NLP) с помощью рекуррентных нейронных сетей (RNN). Мы преобразуем текстовые данные в числовые последовательности, добавляем дополнительное значение для выравнивания длины последовательностей, создаем и компилируем модель RNN, а затем обучаем ее на метках классов.

Обратите внимание, что в реальных приложениях модели обычно будут более сложными, а данные - более разнообразными и объемными. TensorFlow предоставляет множество инструментов и гибкость для работы с различными типами данных и задачами машинного обучения.

Лабораторная работа №2 Реализация модели линейной регрессии с использованием TensorFlow Core

Цель работы: Научиться использовать фреймворк «TensorFlow Core» для реализации простой модели линейной регрессии.

Задание:

1. Изучить теоретические основы линейной регрессии и принцип работы TensorFlow Core.
2. Написать Python-скрипт для генерации сгенерированных данных (например, случайные точки) или загрузить подходящий датасет для линейной регрессии.
3. Импортировать библиотеки TensorFlow Core и NumPy в скрипт.
4. Создать граф вычислений в TensorFlow Core для модели линейной регрессии.
5. Определить переменные модели (веса и смещение) и функцию потерь (среднеквадратичную ошибку).
6. Задать оптимизатор (например, градиентный спуск) для обновления весов модели.
7. Разделить данные на тренировочный и тестовый наборы.
8. Обучить модель на тренировочных данных с использованием TensorFlow Core и оценить качество модели на тестовых данных.
9. Вывести обученные значения весов и смещения модели.
10. Построить график, на котором изображены исходные данные и прямая, соответствующая обученной модели линейной регрессии.
11. Проанализировать результаты, сделать выводы и подготовить отчет о выполненной лабораторной работе.

Ресурсы:

- Документация TensorFlow Core для Python (https://www.tensorflow.org/api_docs/python/)
- Руководства и учебные материалы по линейной регрессии и TensorFlow Core.
- По желанию, студенты могут использовать собственные данные для лабораторной работы или выбрать подходящий датасет для линейной регрессии из доступных открытых источников.

Примечание: Важно обратить внимание на правильное создание графа вычислений, передачу данных с помощью feed_dict и использование оптимизатора для обновления весов модели.

Входные данные

Для реализации лабораторной работы по модели линейной регрессии вы можете использовать различные открытые источники данных. Вот несколько популярных мест, где вы можете найти подходящие датасеты:

1. Kaggle (https://www.kaggle.com/): Kaggle - это платформа для соревнований по анализу данных и машинному обучению. Здесь вы найдете огромное количество датасетов различных тематик, включая датасеты для линейной регрессии.
2. UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/index.php): Это репозиторий машинного обучения, где собраны множество датасетов, используемых для тестирования и оценки алгоритмов машинного обучения. Здесь также можно найти датасеты для линейной регрессии.

3. scikit-learn datasets (https://scikit-learn.org/stable/datasets/index.html): Библиотека scikit-learn для Python содержит некоторые встроенные датасеты, которые можно использовать для экспериментов с моделями машинного обучения, включая линейную регрессию.
4. Также, если вы работаете над специфическим проектом, связанным с линейной регрессией, вы можете собрать свои собственные данные, предварительно обработав их, чтобы они подходили для использования в модели линейной регрессии.

При выборе датасета убедитесь, что он соответствует вашим потребностям и задаче линейной регрессии. Обратите внимание на размер данных, типы признаков и меток, а также целевую переменную, которую вы хотите предсказать с помощью модели линейной регрессии.

Ниже, для реализации лабораторной работы будут сгенерированы синтетические данные.

Пишем необходимый код для реализации поставленной задачи

Шаг 1: Установка Python и редактора кода. Перед началом работы убедитесь, что у вас установлен Python (версия 3.x) на вашем компьютере. Также вам потребуется установить редактор кода. Рекомендуемыми редакторами для Python являются Visual Studio Code или PyCharm. Выберите и установите один из них.

Шаг 2: Начало работы с проектом. Откройте редактор кода и создайте новый файл. Назовите его, например, "linear_regression.py".

Шаг 3: Установка TensorFlow Откройте терминал или командную строку и установите TensorFlow с помощью pip:

```
pip install tensorflow
```

Шаг 4: Установка дополнительных пакетов:

```
python -m pip install -U matplotlib
```

Шаг 5: Написание кода. Теперь давайте напишем код для реализации модели линейной регрессии с использованием TensorFlow Core:

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

# Генерация синтетических данных
np.random.seed(0)
X_train = np.linspace(0, 1, 100)
y_train = 2 * X_train + 1 + np.random.normal(0, 0.1, size=100)

# Создание графа вычислений в TensorFlow 2.x
X = tf.constant(X_train, dtype=tf.float32)
y = tf.constant(y_train, dtype=tf.float32)
```

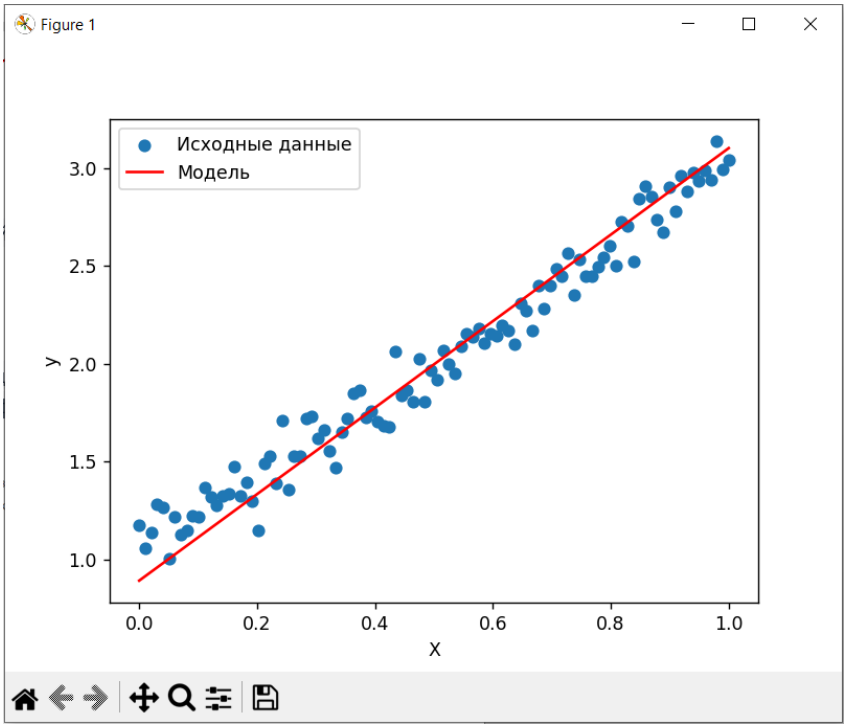
```
# Задание переменных модели (весов и смещения)
W = tf.Variable(np.random.randn(), dtype=tf.float32)
b = tf.Variable(np.random.randn(), dtype=tf.float32)

# Задание модели (линейная регрессия)
def linear_regression(x):
    return W * x + b

# Задание функции потерь (среднеквадратичная ошибка)
def loss_fn(y_true, y_pred):
    return tf.reduce_mean(tf.square(y_true - y_pred))

# Задание оптимизатора (градиентный спуск)
learning_rate = 0.01
optimizer = tf.optimizers.SGD(learning_rate)

# Обучение модели
epochs = 1000
for epoch in range(epochs):
    with tf.GradientTape() as tape:
        y_pred = linear_regression(X)
        loss = loss_fn(y, y_pred)

    gradients = tape.gradient(loss, [W, b])
    optimizer.apply_gradients(zip(gradients, [W, b]))

    if epoch % 100 == 0:
        print(f"Эпоха {epoch}, функция потерь: {loss:.4f}")

# Вывод результатов и построение графика
trained_W, trained_b = W.numpy(), b.numpy()
print(f"Обученные веса: {trained_W:.4f}, обученное смещение:
{trained_b:.4f}")

plt.scatter(X_train, y_train, label='Исходные данные')
plt.plot(X_train, X_train * trained_W + trained_b, 'r',
label='Модель')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.show()
```

Шаг 6: Запуск программы. Сохраните файл и запустите его с помощью команды в терминале:

```
python linear_regression.py
```

После запуска скрипта вы должны увидеть в окне браузера вывод результатов обученных весов и смещения, а также график сгенерированных данных и обученной модели.

```
PS C:\Users\MaxPC\Desktop\New folder (2)> python linear_regression.py
2023-08-01 17:37:49.635293: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in
 performance-critical operations.
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFlow with the appr
opriate compiler flags.
Эпоха 0, функция потерь: 5.8293
Эпоха 100, функция потерь: 0.0995
Эпоха 200, функция потерь: 0.0528
Эпоха 300, функция потерь: 0.0426
Эпоха 400, функция потерь: 0.0350
Эпоха 500, функция потерь: 0.0291
Эпоха 600, функция потерь: 0.0246
Эпоха 700, функция потерь: 0.0212
Эпоха 800, функция потерь: 0.0186
Эпоха 900, функция потерь: 0.0166
Обученные веса: 2.2103, обученное смещение: 0.8922
```



В случае, если имеется набор данных (dataset), который записан в файл (например, dataset.csv), мы можем переписать код, чтобы алгоритм использовал этот файл для загрузки данных:

Установим еще одну необходимую библиотеку:

```
pip install pandas
```

Далее пишем код:

```
import tensorflow as tf
import numpy as np
```

```
import matplotlib.pyplot as plt
import pandas as pd

# Загрузка датасета из файла "dataset.csv"
df = pd.read_csv("dataset.csv")
X_train = df['X'].values
y_train = df['y'].values

# Создание графа вычислений в TensorFlow 2.x
X = tf.constant(X_train, dtype=tf.float32)
y = tf.constant(y_train, dtype=tf.float32)

# Задание переменных модели (весов и смещения)
W = tf.Variable(np.random.randn(), dtype=tf.float32)
b = tf.Variable(np.random.randn(), dtype=tf.float32)

# Задание модели (линейная регрессия)
def linear_regression(x):
    return W * x + b

# Задание функции потерь (среднеквадратичная ошибка)
def loss_fn(y_true, y_pred):
    return tf.reduce_mean(tf.square(y_true - y_pred))

# Задание оптимизатора (градиентный спуск)
learning_rate = 0.01
optimizer = tf.optimizers.SGD(learning_rate)

# Обучение модели
epochs = 1000
for epoch in range(epochs):
    with tf.GradientTape() as tape:
        y_pred = linear_regression(X)
        loss = loss_fn(y, y_pred)

    gradients = tape.gradient(loss, [W, b])
    optimizer.apply_gradients(zip(gradients, [W, b]))

    if epoch % 100 == 0:
        print(f"Эпоха {epoch}, функция потерь: {loss:.4f}")

# Вывод результатов и построение графика
trained_W, trained_b = W.numpy(), b.numpy()
```

```
print(f"Обученные веса: {trained_W:.4f}, обученное смещение:
{trained_b:.4f}")

plt.scatter(X_train, y_train, label='Исходные данные')
plt.plot(X_train, X_train * trained_W + trained_b, 'r',
label='Модель')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.show()
```

Запускаем код с помощью команды в терминале:

```
python linear_regression.py
```

В коде выше мы загружаем датасет из файла «dataset.csv» с помощью библиотеки «pandas» и используем его для обучения модели линейной регрессии. Обратите внимание, что предполагается, что файл "dataset.csv" содержит две колонки: "X" с входными данными и "y" с соответствующими метками. Пожалуйста, убедитесь, что ваш файл данных имеет аналогичную структуру.

Лекция №3 PyTorch

PyTorch – это популярный открытый фреймворк для глубокого обучения, разработанный Facebook's AI Research (FAIR) Lab. Он предоставляет высокоуровневые функции для создания и обучения нейронных сетей, а также предоставляет множество возможностей для выполнения сложных операций на графических процессорах. PyTorch является одним из наиболее популярных фреймворков для глубокого обучения благодаря своей гибкости, простоте использования и отличной документации.

Преимущества и варианты использования PyTorch

Преимущества PyTorch:

1. Динамический граф вычислений: PyTorch использует динамический граф вычислений, который позволяет оптимизировать модель на лету, менять ее структуру и отладку. Это делает разработку и исследования гораздо более гибкими.
2. Простота использования: Синтаксис PyTorch напоминает стандартный Python, что делает его доступным для новичков и опытных разработчиков. Структура PyTorch также интуитивно понятна.
3. Автоматическое дифференцирование: PyTorch предоставляет автоматическое дифференцирование для всех операций, что облегчает обратное распространение ошибки и обучение моделей.
4. Широкая поддержка графических процессоров: PyTorch позволяет легко выполнять вычисления на графических процессорах, что ускоряет процесс обучения нейронных сетей.

Варианты использования PyTorch:

1. Обучение нейронных сетей: PyTorch позволяет легко определять и обучать сложные модели нейронных сетей для различных задач, таких как классификация изображений, обработка естественного языка и многие другие.
2. Исследования в области глубокого обучения: Благодаря своей гибкости, PyTorch становится популярным выбором для исследователей, которые хотят быстро прототипировать и проверить новые идеи в области глубокого обучения.
3. Производственные системы: PyTorch предоставляет возможности для развертывания обученных моделей в производственных системах, что делает его полезным для реальных приложений.

Плюсы и минусы PyTorch

Плюсы:

1. Гибкость и простота: PyTorch предоставляет гибкие инструменты для определения моделей и легкий в освоении синтаксис, что упрощает процесс разработки.
2. Автоматическое дифференцирование: Встроенное автоматическое дифференцирование упрощает обратное распространение ошибки и обучение моделей.
3. Активное сообщество: PyTorch имеет широкое и активное сообщество, которое поддерживает разработчиков, предоставляя множество документации, уроков и примеров кода.

Минусы:

1. Производительность: Несмотря на то, что PyTorch является быстрым, в сравнении с некоторыми другими фреймворками, такими как TensorFlow, может быть незначительное различие в производительности.
2. Ограниченная поддержка мобильных платформ: На момент написания этой лекции, поддержка мобильных платформ в PyTorch не так широка, как у некоторых других фреймворков.

Начало работы с PyTorch

Установка PyTorch:

Для установки PyTorch можно воспользоваться инструкциями с официального сайта PyTorch (https://pytorch.org) в соответствии с вашей операционной системой и конфигурацией.

Настройка среды разработки:

- Установите Python и pip (если еще не установлены).
- Создайте виртуальное окружение (рекомендуется) для изоляции проекта.
- Установите PyTorch и его зависимости с помощью pip.

Пример кода для установки PyTorch в виртуальном окружении:

```
# Создание и активация виртуального окружения (необязательно)
python -m venv myenv
# На Windows: myenv\Scripts\activate
source myenv/Scripts/activate.bat

# Установка PyTorch
pip install torch torchvision
```

В основе фреймворка PyTorch стоят несколько ключевых идей, которые делают его мощным инструментом для глубокого обучения. Рассмотрим эти идеи пошагово, начиная с тензоров и вычислительных графов и заканчивая переменными классами и функциональностью автоматического дифференцирования.

Тензоры

В PyTorch основной единицей данных является тензор - многомерный массив, подобный массивам NumPy. Тензоры в PyTorch обеспечивают удобное представление данных, с которыми работают нейронные сети. Основной тип тензора - это `torch.Tensor`, который поддерживает множество операций и вычислений.

Вычислительные графы

В основе PyTorch лежит концепция вычислительных графов. Вычислительный граф - это абстрактное представление вычислений, которые выполняются в нейронных сетях. PyTorch использует динамические вычислительные графы, что означает, что граф строится и оптимизируется на лету, при выполнении операций. Это делает PyTorch гибким для определения сложных моделей с переменной структурой.

Переменные (Variables)

Когда вы создаете тензоры в PyTorch, они автоматически оборачиваются в класс «torch.Tensor», представляющие собой переменные. Переменные позволяют автоматически отслеживать историю операций и вычислений, которые происходят с данными. Это важно для реализации функциональности автоматического дифференцирования, что позволяет эффективно обучать нейронные сети методом обратного распространения ошибки.

Автоматическое дифференцирование

Одной из основных преимуществ PyTorch является его встроенная функциональность автоматического дифференцирования. Когда операции выполняются над тензорами, PyTorch автоматически создает вычислительный граф и отслеживает производные операций относительно переменных (тензоров). Таким образом, PyTorch может автоматически вычислять градиенты функции по входным данным.

Это делает процесс обучения нейронных сетей значительно проще, поскольку программистам не нужно вручную вычислять градиенты и писать сложные алгоритмы обратного распространения ошибки.

PyTorch представляет собой мощный фреймворк для глубокого обучения, который базируется на концепциях тензоров, вычислительных графов, переменных и автоматического дифференцирования. Эти ключевые идеи делают его гибким, удобным в использовании и эффективным инструментом для разработки и обучения нейронных сетей.

Сравнение фреймворков TensorFlow и PyTorch

Сравнение фреймворков TensorFlow и PyTorch имеет большое значение для разработчиков машинного обучения, так как эти фреймворки являются двумя наиболее популярными инструментами для создания и обучения нейронных сетей. Оба фреймворка предоставляют мощные инструменты для глубокого обучения, но они имеют некоторые различия в архитектуре и функциональности. Давайте рассмотрим основные аспекты сравнения TensorFlow и PyTorch:

1. Архитектура и стиль программирования:
   - TensorFlow: TensorFlow является более графическим фреймворком. Пользователи создают вычислительные графы, определяющие структуру модели, а затем выполняют вычисления внутри сессий. Это делает TensorFlow более подходящим для статических моделей, которые не меняют свою структуру в процессе обучения.
   - PyTorch: PyTorch работает на более динамическом вычислительном графе. Это позволяет пользователям определять и изменять структуру модели на лету. Такой подход делает PyTorch более удобным для экспериментов и разработки новых идей.

2. Гибкость и простота использования:
   - TensorFlow: TensorFlow, особенно в версиях до 2.0, часто считался менее интуитивным и более сложным в использовании для новичков. Однако с выпуском TensorFlow 2.0, его интерфейс стал более удобным и дружелюбным к пользователю.
   - PyTorch: PyTorch обладает простой и понятной синтаксической структурой, что делает его более простым в использовании и более пригодным для обучения и экспериментов.
3. Обучение:
   - TensorFlow: В TensorFlow используется концепция сессий для обучения моделей, что может быть несколько сложнее для новичков в сравнении с PyTorch.
   - PyTorch: PyTorch использует динамический вычислительный граф и более простую систему обратного распространения ошибки, что делает процесс обучения более интуитивным и менее запутанным.
4. Экосистема и поддержка:
   - TensorFlow: TensorFlow имеет широкую поддержку со стороны сообщества и обширную экосистему, что позволяет использовать его в различных областях машинного обучения и искусственного интеллекта. Он также активно используется в промышленных проектах.
   - PyTorch: PyTorch набирает популярность и имеет активное сообщество разработчиков, но его экосистема может быть несколько меньше, чем у TensorFlow. Однако, PyTorch становится все более популярным, особенно в научных и исследовательских кругах.
5. Распределенное обучение:
   - TensorFlow: TensorFlow предоставляет более развитые инструменты для распределенного обучения и масштабирования на кластерах серверов.
   - PyTorch: PyTorch также поддерживает распределенное обучение, но его инструменты могут быть менее развитыми по сравнению с TensorFlow.
6. Производительность:
   - TensorFlow: TensorFlow имеет хорошую производительность на GPU и TPU. Он широко используется в индустрии и на производственных системах.
   - PyTorch: PyTorch также обладает хорошей производительностью на GPU и TPU. В последние годы он улучшил свою производительность и стабильность.

В итоге, выбор между TensorFlow и PyTorch зависит от ваших предпочтений и конкретных задач. TensorFlow хорошо подходит для промышленных проектов и распределенного обучения, в то время как PyTorch предпочтителен для исследовательской работы и прототипирования. Оба фреймворка являются мощными инструментами, которые позволяют решать широкий спектр задач в области глубокого обучения.

Пример создания нейронной сети с использованием PyTorch

Далее рассмотрим пошаговый пример создания и обучения простой нейронной сети для задачи классификации изображений с использованием PyTorch. В этом примере мы будем использовать набор данных CIFAR-10, который содержит изображения 10 различных классов, таких как автомобили, самолеты, кошки и т.д.

Прежде чем начать, убедитесь, что у вас установлен PyTorch и torchvision. Если их нет, установите их с помощью pip:

```
pip install torch torchvision
```

Шаг 1: Импорт необходимых библиотек

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
```

Шаг 2: Подготовка данных

```
# Задаем преобразования для нормализации и аугментации данных
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

# Загружаем обучающий и тестовый наборы данных CIFAR-10
trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True,
transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=4,
                                          shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True,
transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=4,
                                         shuffle=False, num_workers=2)

# Определяем классы для классификации
classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

Шаг 3: Определение нейронной сети

```
# Определяем простую сверточную нейронную сеть
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
```

```
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1)  # "Распрямляем" тензор для
полносвязного слоя
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

net = Net()
```

Шаг 4: Определение функции потерь и оптимизатора

```
# Функция потерь
criterion = nn.CrossEntropyLoss()
# Оптимизатор SGD
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

Шаг 5: Обучение модели

```
device = torch.device("cuda:0" if torch.cuda.is_available() else
"cpu")
net.to(device)

for epoch in range(10):  # Проводим обучение в течение 10 эпох
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        inputs, labels = data[0].to(device), data[1].to(device)

        optimizer.zero_grad()  # Обнуляем градиенты параметров

        outputs = net(inputs)  # Получаем выход модели
        loss = criterion(outputs, labels)  # Вычисляем функцию потерь
        loss.backward()  # Обратное распространение ошибки
        optimizer.step()  # Обновление параметров

        running_loss += loss.item()
        if i % 2000 == 1999:  # Печатаем статистику каждые 2000 мини-
батчей
```

```
            print(f'Эпоха {epoch + 1}, Мини-батч {i + 1}, Потеря:
{running_loss / 2000:.3f}')
            running_loss = 0.0

print('Обучение завершено!')
```

Шаг 6: Тестирование модели

```
correct = 0
total = 0
with torch.no_grad():  # Отключаем вычисление градиентов для тестовой
части
    for data in testloader:
        images, labels = data[0].to(device), data[1].to(device)
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f'Точность сети на тестовых данных: {100 * correct /
total:.2f}%')
```

Это простой пример обучения сверточной нейронной сети с помощью PyTorch для классификации изображений. Обратите внимание, что это лишь базовый пример, и реальные нейронные сети могут иметь более сложные архитектуры и разнообразные улучшения.

Лабораторная работа №3: Создание нейронной сети с использованием фреймворка PyTorch

Цель работы: Освоить базовые принципы создания и обучения нейронных сетей с использованием фреймворка PyTorch.

Задачи:

1. Изучить основные понятия PyTorch: тензоры, вычислительные графы, переменные и автоматическое дифференцирование.
2. Подготовить и предобработать набор данных для обучения и тестирования.
3. Определить архитектуру нейронной сети, включая слои, функции активации и выходные слои.
4. Создать класс модели нейронной сети, унаследованный от torch.nn.Module.
5. Определить функцию потерь и оптимизатор для обучения модели.
6. Настроить гиперпараметры обучения: скорость обучения, количество эпох, размер мини-батча и т.д.
7. Обучить нейронную сеть на обучающем наборе данных и отслеживать процесс обучения.
8. Протестировать обученную модель на тестовом наборе данных и оценить ее точность.
9. Проанализировать результаты и оценить, насколько хорошо модель справилась с задачей классификации.
10. (Дополнительно) Провести эксперименты с различными архитектурами нейронных сетей и гиперпараметрами обучения для определения наилучшего решения.

Инструменты:

- Python с установленными библиотеками PyTorch и torchvision.
- Интерфейс Jupyter Notebook или среда разработки Python, поддерживающая выполнение скриптов.

Материалы:

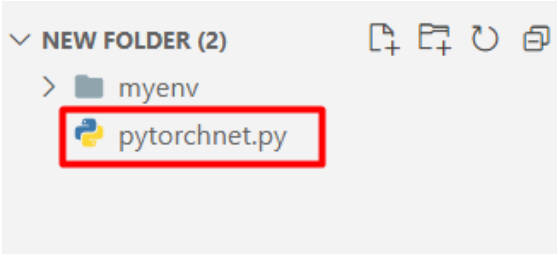
- Набор данных для классификации изображений, например, CIFAR-10.
- Документация PyTorch и примеры кода для руководства.

Ожидаемые результаты: Студенты должны создать и обучить нейронную сеть для классификации изображений, используя фреймворк PyTorch. Они должны оценить точность модели на тестовом наборе данных и провести анализ результатов. В отчете должны быть представлены графики обучения, описание архитектуры модели, выбранные гиперпараметры и выводы о производительности модели.

Необходимые шаги для решения поставленной задачи

Ниже приведен пошаговый пример по созданию и обучению нейронной сети для классификации изображений с использованием PyTorch. Код снабжен комментариями и описанием основных моментов алгоритма.

```
# Создадим папку на рабочем столе. Откроем созданную папку из
командной строки.

# Создадим виртуальное окружение
python -m venv myenv

# Активируем виртуальное окружение:
myenv/Scripts/activate.bat

# Установим PyTorch
pip install torch torchvision
# Откроем проект через VS Code.
# Создадим новый терминал:
Menu File -> Terminal -> New Terminal

# Создадим новый файл «pytorchnet.py»
```




```
# Добавим в созданный файл код, представленный ниже:

# Шаг 1: Импорт необходимых библиотек
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms

if __name__ == '__main__':
    # Шаг 2: Подготовка данных
    # Задаем преобразования для нормализации и аугментации данных
    transform = transforms.Compose(
        [transforms.ToTensor(),                                    #
Преобразуем изображение в тензор
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])  #
Нормализуем тензор средними и стандартными отклонениями

    # Загружаем обучающий и тестовый наборы данных CIFAR-10
    trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
```

```
                                            download=True,
transform=transform)  # Загрузка обучающего набора
    trainloader = torch.utils.data.DataLoader(trainset, batch_size=4,
                                            shuffle=True,
num_workers=2)       # Создание итератора для обучающего набора

    testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                        download=True,
transform=transform)  # Загрузка тестового набора
    testloader = torch.utils.data.DataLoader(testset, batch_size=4,
                                            shuffle=False,
num_workers=2)       # Создание итератора для тестового набора

    # Определяем классы для классификации
    classes = ('plane', 'car', 'bird', 'cat',
            'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

    # Шаг 3: Определение нейронной сети
    class Net(nn.Module):
        def __init__(self):
            super(Net, self).__init__()
            self.conv1 = nn.Conv2d(3, 6, 5)   # Первый сверточный
слой: 3 входных канала (RGB), 6 выходных каналов, размер ядра 5x5
            self.pool = nn.MaxPool2d(2, 2)    # Пулинг слой: размер
пулинга 2x2, с шагом 2
            self.conv2 = nn.Conv2d(6, 16, 5)  # Второй сверточный
слой: 6 входных каналов (из первого слоя), 16 выходных каналов, размер
ядра 5x5
            self.fc1 = nn.Linear(16 * 5 * 5, 120)  # Полносвязный
слой: 16*5*5 входных нейронов (из второго слоя), 120 выходных нейронов
            self.fc2 = nn.Linear(120, 84)    # Второй полносвязный
слой: 120 входных нейронов (из первого полносвязного слоя), 84
выходных нейрона
            self.fc3 = nn.Linear(84, 10)     # Выходной слой: 84
входных нейрона (из второго полносвязного слоя), 10 выходных нейронов
(количество классов)

        def forward(self, x):
            x = self.pool(nn.functional.relu(self.conv1(x)))   #
Применяем первый сверточный слой, затем функцию активации ReLU и
пулинг
            x = self.pool(nn.functional.relu(self.conv2(x)))   #
Применяем второй сверточный слой, затем функцию активации ReLU и
пулинг
```

```
            x = torch.flatten(x, 1)  # "Распрямляем" тензор для
полносвязного слоя
            x = nn.functional.relu(self.fc1(x))  # Применяем первый
полносвязный слой и функцию активации ReLU
            x = nn.functional.relu(self.fc2(x))  # Применяем второй
полносвязный слой и функцию активации ReLU
            x = self.fc3(x)  # Применяем выходной слой
            return x

    net = Net()

    # Шаг 4: Определение функции потерь и оптимизатора
    criterion = nn.CrossEntropyLoss()  # Функция потерь
CrossEntropyLoss: используется для задачи классификации
    optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)  #
Оптимизатор SGD с импульсом: используется для обновления параметров
модели

    # Шаг 5: Обучение модели
    device = torch.device("cuda:0" if torch.cuda.is_available() else
"cpu")  # Определяем устройство для обучения (GPU, если доступен,
иначе CPU)
    net.to(device)  # Переносим модель на устройство (GPU или CPU)

    for epoch in range(10):  # Проводим обучение в течение 10 эпох
        running_loss = 0.0
        for i, data in enumerate(trainloader, 0):
            inputs, labels = data[0].to(device), data[1].to(device)  #
Переносим данные на устройство

            optimizer.zero_grad()  # Обнуляем градиенты параметров

            outputs = net(inputs)  # Получаем выход модели
            loss = criterion(outputs, labels)  # Вычисляем функцию
потерь
            loss.backward()  # Обратное распространение ошибки для
вычисления градиентов
            optimizer.step()  # Обновление параметров модели на основе
градиентов

            running_loss += loss.item()
            if i % 2000 == 1999:  # Печатаем статистику каждые 2000
мини-батчей
```

```
                print(f'Эпоха {epoch + 1}, Мини-батч {i + 1}, Потеря:
{running_loss / 2000:.3f}')
                running_loss = 0.0

    print('Обучение завершено!')

    # Шаг 6: Тестирование модели
    correct = 0
    total = 0
    with torch.no_grad():  # Отключаем вычисление градиентов для
тестовой части
        for data in testloader:
            images, labels = data[0].to(device), data[1].to(device)  #
Переносим данные на устройство
            outputs = net(images)  # Получаем выход модели
            _, predicted = torch.max(outputs.data, 1)  # Определяем
предсказанные классы
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    print(f'Точность сети на тестовых данных: {100 * correct /
total:.2f}%')
```

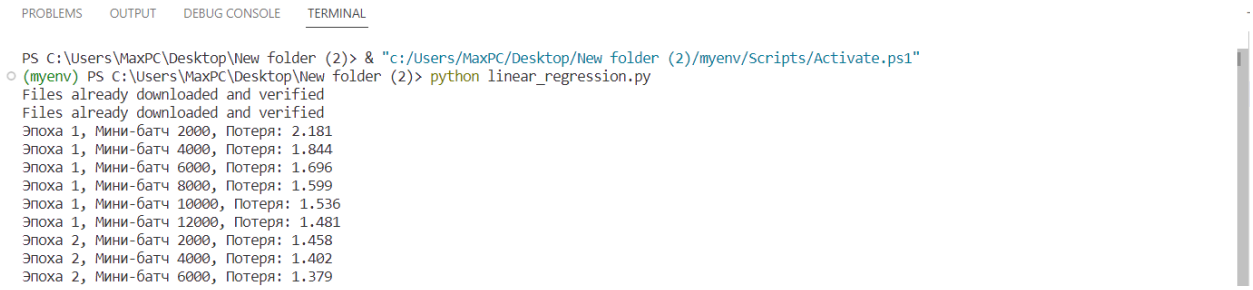
Описание алгоритма:

1. Импортируем необходимые библиотеки, включая PyTorch и torchvision для работы с нейронными сетями и наборами данных.
2. Подготавливаем данные: загружаем обучающий и тестовый наборы данных CIFAR-10, а также определяем преобразования для нормализации и аугментации данных.
3. Определяем класс нейронной сети, состоящий из двух сверточных слоев, пулинга и полносвязных слоев. Каждый слой описан в комментарии.
4. Определяем функцию потерь и оптимизатор для обучения модели. В данном примере мы используем кросс-энтропийную функцию потерь и оптимизатор стохастического градиентного спуска (SGD) с импульсом.
5. Обучаем модель на обучающем наборе данных в течение 10 эпох, используя оптимизатор SGD и функцию потерь. Во время обучения мы отслеживаем потери на каждом мини-батче и выводим их на экран.
6. После завершения обучения тестируем обученную модель на тестовом наборе данных и вычисляем ее точность.

```
# Запустим код на выполнение следующей командой:
python pytorchnet.py
```

После запуска кода, необходимо некоторое время (порядка 5-10 минут) для создания, обучения и тестирования нейронной сети.

Вывод в консоль будет иметь следующий вид:



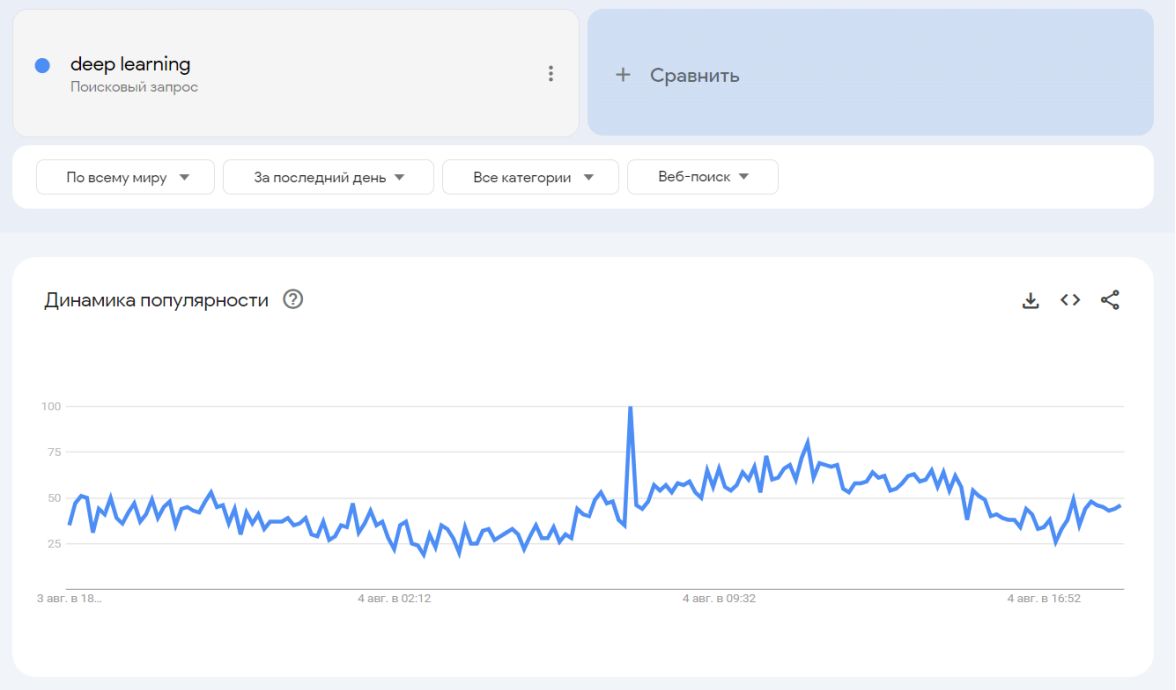
...

Как результат, вы получите обученную модель нейронной сети, способную классифицировать изображения на тестовом наборе данных CIFAR-10.

Лекция №4 Keras. Глубокое обучение нейронных сетей

Глубокое обучение стало модным трендом в области искусственного интеллекта (ИИ). В течение многих лет мы использовали машинное обучение (ML) для наделения машин интеллектом. В последнее время глубокое обучение стало более популярным благодаря своему превосходству в предсказаниях по сравнению с традиционными методами ML.

Глубокое обучение, по сути, означает обучение искусственной нейронной сети (ANN) огромному объему данных. При глубоком обучении сеть обучается сама по себе и, таким образом, требует огромных объемов данных для обучения. В то время как традиционное машинное обучение - это, по сути, набор алгоритмов, которые анализируют данные и извлекают из них уроки. Затем они использовали это обучение для принятия разумных решений.



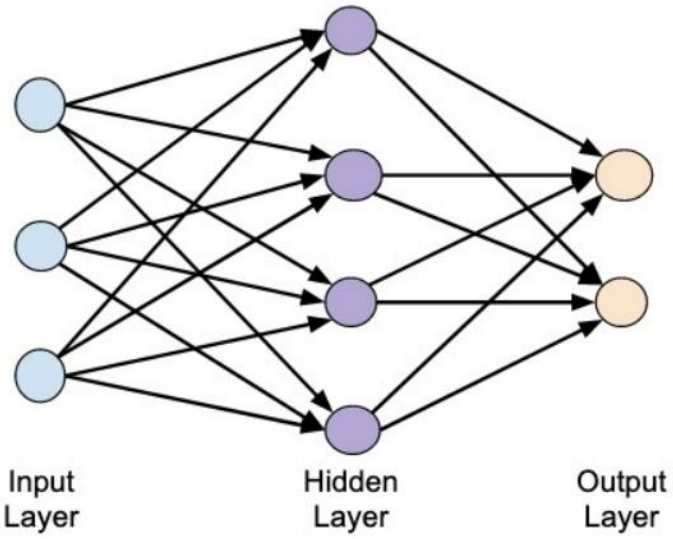
Чтобы понять, на сколько популярно сегодня глубокое обучение, выше приведен скриншот Google Trends. Из диаграммы видно, что интерес к глубокому не падает и даже немного растет в течение последних нескольких лет. Существует множество областей, таких как компьютерное зрение, обработка естественного языка, распознавание речи, биоинформатика, разработка лекарств и т.д., где глубокое обучение успешно применяется.

Глубокое обучение

Глубокое обучение - это процесс обучения искусственной нейронной сети огромному объему данных. После обучения сеть сможет давать нам прогнозы на основе невидимых данных. Прежде чем продолжить говорить о глубоком обучении, давайте быстро разберем некоторые термины, используемые при обучении нейронной сети.

Искусственные нейронные сети

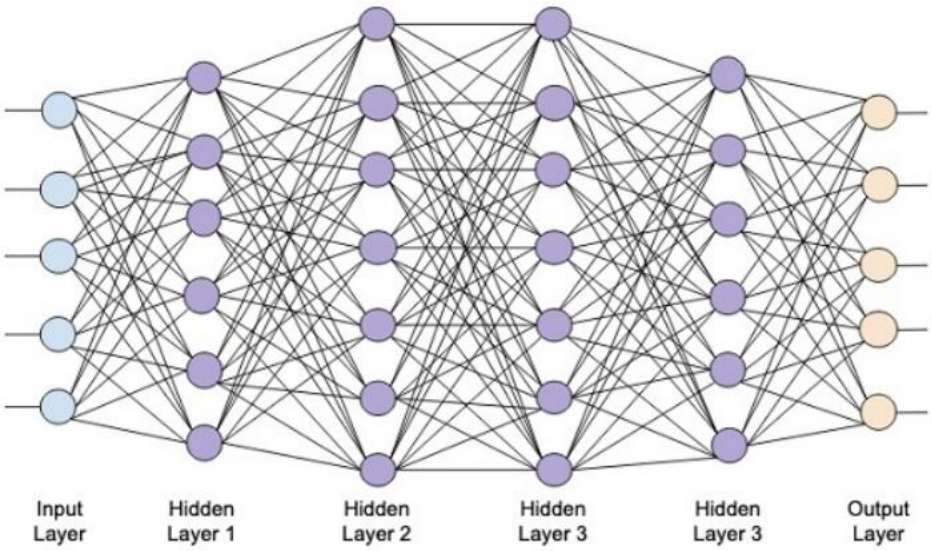
Идея искусственной нейронной сети была заимствована из нейронных сетей в нашем мозге. Типичная нейронная сеть состоит из трех слоев — входного, выходного и скрытого, как показано на рисунке ниже.



Это также называется неглубокой нейронной сетью, поскольку она содержит только один скрытый слой. Можно добавить больше скрытых слоев в приведенную выше архитектуру, чтобы создать более сложную архитектуру.

Глубокие сети

На следующей диаграмме показана глубокая сеть, состоящая из четырех скрытых слоев, входного слоя и выходного слоя.



По мере увеличения количества скрытых слоев в сети, ее обучение становится более сложным с точки зрения требуемых ресурсов и времени, необходимого для полного обучения сети.

Сетевое обучение (Network Training)

После того как вы определяете архитектуру сети, вы обучаете ее для выполнения определенных видов прогнозов. Обучение сети - это процесс нахождения правильных весовых коэффициентов для каждого звена в сети. Во время обучения данные передаются из входных слоев в выходные через различные скрытые слои. Поскольку данные всегда перемещаются в одном направлении от входа к выходу, мы называем эту сеть сетью прямой связи, а распространение данных - прямым распространением (Forward Propagation).

Функция активации (Activation Function)

На каждом уровне мы вычисляем взвешенную сумму входных данных и передаем ее в функцию активации. Функция активации привносит нелинейность в сеть. Это просто некоторая математическая функция, которая дискретизирует выходные данные. Некоторыми из наиболее часто используемых функций активации являются sigmoid, hyperbolic, tangent (tanh), ReLU and Softmax.

Обратное распространение (Backpropagation)

Обратное распространение - это алгоритм для контролируемого обучения. При обратном распространении ошибки распространяются в обратном направлении от выходного уровня к входному. Учитывая функцию ошибки, мы вычисляем градиент функции ошибки относительно весов, присвоенных каждому соединению. Вычисление градиента выполняется в обратном порядке по сети. Градиент последнего слоя весов вычисляется первым, а градиент первого слоя весов вычисляется последним.

На каждом слое частичные вычисления градиента повторно используются при вычислении градиента для предыдущего слоя. Это называется градиентным спуском.

Keras – это высокоуровневый API для нейронных сетей, который работает поверх TensorFlow - сквозной платформы машинного обучения с открытым исходным кодом. Используя Keras, вы легко определяете сложные архитектуры ANN для экспериментов с большими данными. Keras также поддерживает графический процессор, который становится необходимым для обработки огромных объемов данных и разработки моделей машинного обучения.

Для чего используется Keras?

Keras используется для разработки и обучения нейронных сетей. Он поддерживает быструю и простую разработку моделей для различных задач машинного обучения, таких как классификация, регрессия, сегментация, обработка естественного языка и многое другое. Фреймворк Keras также способствует переносимости кода, что означает, что вы можете легко переносить модели между различными бэкендами, такими как TensorFlow и Microsoft Cognitive Toolkit (CNTK).

Особенности работы с фреймворком Keras

1. Простота и интуитивность: Keras обладает чистым и легко понимаемым API, что делает его отличным выбором для новичков и исследователей.
2. Модульность: Keras позволяет строить модели с помощью последовательных и функциональных API, что обеспечивает гибкость в проектировании сложных нейронных сетей.
3. Поддержка множества бэкендов: Keras позволяет использовать различные библиотеки для работы с тензорами, такие как TensorFlow, CNTK или Theano, что дает вам возможность выбрать оптимальный бэкенд для вашей задачи.
4. Автоматическое дифференцирование: Keras обеспечивает автоматическое дифференцирование, что позволяет легко оптимизировать параметры модели в процессе обучения.

Преимущества и недостатки Keras

Преимущества:

1. **Гибкая работа с бэкендом**: Keras обеспечивает абстракцию уровня бэкенда, что позволяет легко переключаться между различными библиотеками для работы с тензорами, такими как TensorFlow, Theano или CNTK. Это дает пользователям возможность выбирать наиболее подходящий бэкенд для их задач и аппаратного обеспечения.
2. **Удобство**: Keras был разработан с упором на простоту использования. Он предоставляет простой и интуитивно понятный API, который позволяет создавать и обучать нейронные сети с минимальными усилиями. Это особенно ценно для новичков и исследователей, которые могут быстро прототипировать модели и экспериментировать с различными архитектурами.
3. **Модульность**: Keras позволяет строить модели с помощью последовательных и функциональных API. Это дает пользователю гибкость и контроль при проектировании сложных нейронных сетей. Вы можете легко создавать множество ветвлений, объединений и сложных архитектур, что делает Keras мощным инструментом для реализации разнообразных моделей.
4. **Расширяемость**: Keras предоставляет возможность создавать собственные слои, функции активации, функции потерь и оптимизаторы. Это дает пользователю возможность создавать собственные модули и расширять функциональность Keras согласно своим потребностям.
5. **Высокая скорость работы**: Как высокоуровневый фреймворк, Keras стремится обеспечить быстродействие. Он использует оптимизированные бэкенды, такие как TensorFlow, чтобы обеспечить высокую производительность при выполнении операций с тензорами и обучении нейронных сетей.
6. **Популярность**: Keras является одним из самых популярных и широко используемых фреймворков глубокого обучения. Он имеет активное сообщество пользователей и разработчиков, что обеспечивает поддержку, обновления и расширение функциональности. Большое количество обучающих материалов и ресурсов делает Keras привлекательным выбором для многих исследователей и разработчиков.

Объединение всех этих преимуществ делает Keras мощным инструментом для быстрого прототипирования, разработки и обучения нейронных сетей. Он подходит как для новичков, так и для опытных исследователей, и его популярность продолжает расти в сообществе машинного обучения и искусственного интеллекта.

Недостатки:

1. **Неуниверсальность**: Несмотря на то, что Keras обладает гибкостью выбора бэкенда и поддерживает несколько популярных библиотек для работы с тензорами, он все равно ограничен теми бэкендами, которые поддерживает. Это означает, что некоторые существующие или новые библиотеки не могут быть легко интегрированы с Keras.
2. **Отсутствие обратной совместимости**: В связи с постоянным развитием и улучшением Keras, обратная совместимость между различными версиями может быть нарушена. Это означает, что код, написанный для одной версии Keras, может не работать без изменений в другой версии, что может вызвать неудобства при обновлении фреймворка.
3. **Зависимость от движка**: Keras представляет собой высокоуровневую оболочку для работы с бэкендами, такими как TensorFlow или Theano. Это означает, что при использовании Keras вы становитесь зависимыми от выбранного бэкенда, и если в будущем решите перейти на другой бэкенд, это потребует изменения вашего кода.
4. **Сложность обучения**: Несмотря на удобство и простоту использования, обучение нейронных сетей всегда остается сложной задачей, и Keras не исключение. Для успешного обучения моделей в Keras все равно требуется глубокое понимание основных принципов и алгоритмов машинного обучения, а также опыт в настройке гиперпараметров и оптимизации моделей.
5. **Ограниченность функциональности**: В сравнении с другими фреймворками глубокого обучения, такими как TensorFlow или PyTorch, Keras может быть ограничен в своей функциональности. Некоторые продвинутые возможности и оптимизации могут быть легче реализовать с помощью других фреймворков, которые предоставляют более низкоуровневый доступ к тензорам и операциям.
6. **Большой объем моделей и слоев**: Поскольку Keras старается быть простым и удобным, некоторые пользователи могут столкнуться с ограничениями в объеме доступных предустановленных моделей и слоев. В этом случае придется самостоятельно создавать и настраивать более сложные архитектуры.

Несмотря на эти недостатки, Keras остается популярным фреймворком глубокого обучения, который предоставляет простой и удобный способ создания и обучения нейронных сетей. Он может быть отличным выбором для быстрого прототипирования и исследований, а также для тех, кто только начинает свой путь в мире искусственного интеллекта.

Сравнение с другими фреймворками

Когда дело доходит до сравнения с другими фреймворками, выбор зависит от конкретных потребностей и целей. Если простота использования и быстрая разработка прототипов являются приоритетами, то Keras является прекрасным выбором. Однако, если вам нужна более высокая гибкость и контроль, вы можете обратить внимание на TensorFlow, PyTorch или MXNet.

Пример создания модели в Keras с использованием Jupyter Notebook

Давайте создадим простую нейронную сеть для классификации рукописных цифр из набора данных MNIST с использованием Keras в Jupyter Notebook:

Шаг 1: Установка Python и Jupyter Notebook

Если у вас уже установлен Python, убедитесь, что у вас также установлен Jupyter Notebook. Если нет, установите его с помощью следующей команды:

```
pip install jupyter
```

Шаг 2: Создание виртуальной среды (необязательно, но рекомендуется)

Создание виртуальной среды поможет изолировать зависимости этого проекта от других проектов на вашей системе. Выполните следующие команды для создания и активации виртуальной среды:

```
# Создание виртуальной среды с именем 'keras_env'
python -m venv keras_env

# Активация виртуальной среды в Windows
keras_env\Scripts\activate
```

Шаг 3: Установка необходимых библиотек

Установите библиотеки TensorFlow и Matplotlib (для визуализации) с помощью следующих команд:

```
pip install tensorflow
pip install matplotlib
```

Шаг 4: Запуск Jupyter Notebook

Запустите Jupyter Notebook с помощью команды:

```
jupyter notebook
```

Jupyter Notebook откроется в вашем веб-браузере. Создайте новый ноутбук и начните кодирование примера.

Шаг 5: Пример создания модели в Keras

Теперь давайте перейдем к коду примера. Поместите код после каждого комментария последовательно в ячейки Jupyter Notebook.

```
# Импорт необходимых библиотек
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
```

```
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt

# Загрузка данных MNIST
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Нормализация данных
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

# Определение архитектуры модели
model = Sequential([
    Flatten(input_shape=(28, 28)),    # Преобразование изображений
28x28 в одномерный массив размерности 784
    Dense(128, activation='relu'),     # Полносвязный слой с 128
нейронами и функцией активации ReLU
    Dense(10, activation='softmax')    # Выходной слой с 10 нейронами
и функцией активации Softmax для классификации по 10 классам
])

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

# Обучение модели
history = model.fit(X_train, y_train, epochs=5, batch_size=32,
validation_data=(X_test, y_test))

# Оценка модели на тестовых данных
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f'Точность на тестовых данных: {test_acc}')
```

Теперь вы можете запустить каждую ячейку кода в Jupyter Notebook, чтобы пошагово выполнить пример. Убедитесь, что виртуальная среда активирована, и установлены все необходимые библиотеки перед запуском кода.

Лабораторная работа №4 Создание модели с использованием фреймворка Keras

В текущей лабораторной работе мы создадим нейронную сеть глубокого обучения, которая будет обучена распознаванию рукописных цифр. В любом проекте машинного обучения первой задачей является сбор данных. Особенно для сетей глубокого обучения, нам нужны огромные объемы данных. К счастью, для задачи, которую мы попытаемся решить, кто-то уже создал набор данных для обучения. Он называется «mnist», доступен как часть библиотек Keras. Набор данных состоит из нескольких изображений рукописных цифр размером 28x28 пикселей. Вы будете обучать свою модель на большей части этого набора данных, а остальные данные будут использоваться для проверки обученной модели.

Шаг 1: Установка Python и Jupyter Notebook

Если у вас уже установлен Python, убедитесь, что у вас также установлен Jupyter Notebook. Если нет, установите его с помощью следующей команды:

```
pip install jupyter
```

Шаг 2: Создание виртуальной среды (необязательно, но рекомендуется)

Создайте новую папку на рабочем столе и откройте ее через командную строку.

Создание виртуальной среды поможет изолировать зависимости этого проекта от других проектов на вашей системе. Выполните следующие команды для создания и активации виртуальной среды:

```
# Создание виртуальной среды с именем 'keras_env'
python -m venv keras_env

# Откройте редактор кода VS Code. Откройте новый терминал и
активируйет созданную на прошлом шаге виртуальную среду

# Активация виртуальной среды в Windows
keras_env\Scripts\activate
```

Шаг 3: Установка необходимых библиотек

Установите библиотеки TensorFlow и Matplotlib (для визуализации) с помощью следующих команд:

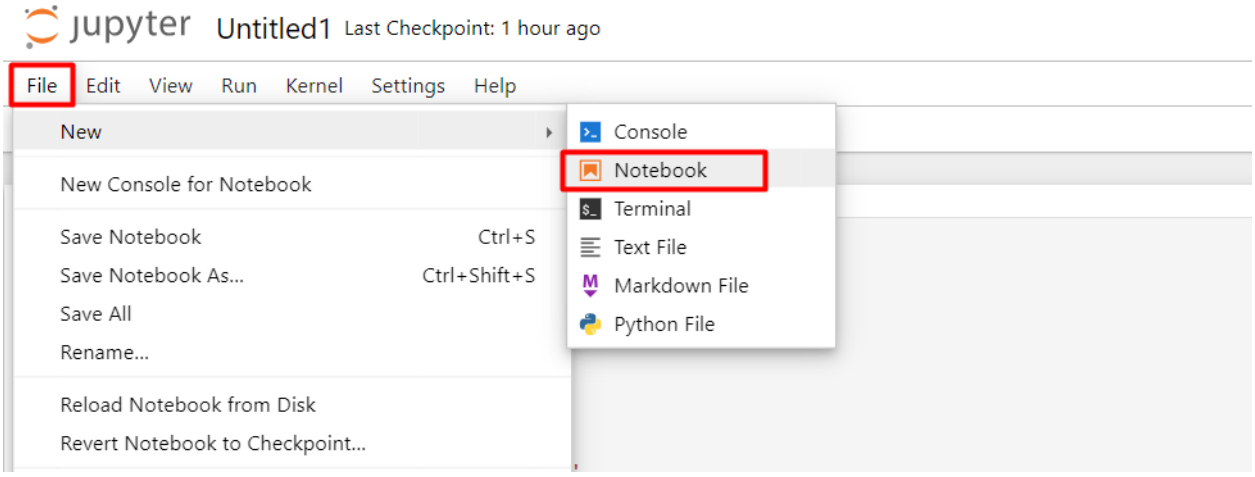
```
pip install tensorflow
pip install matplotlib
```

Шаг 4: Запуск Jupyter Notebook

Запустите Jupyter Notebook с помощью команды:

```
jupyter notebook
```

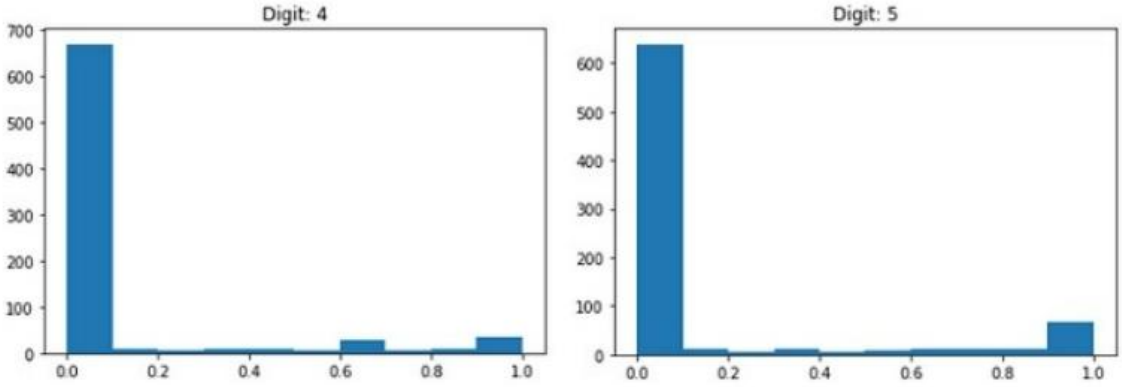
Jupyter Notebook откроется в вашем веб-браузере. Создайте новый ноутбук и начните кодирование примера.



Описание проекта

Набор данных «mnist» состоит из 70000 изображений рукописных цифр. Несколько примеров изображений показаны ниже:

Каждое изображение имеет размер 28 x 28 пикселей, что составляет в общей сложности 768 пикселей с различными уровнями оттенков серого. Большинство пикселей имеют тенденцию к черному оттенку, в то время как лишь немногие из них имеют тенденцию к белому. Мы поместим распределение этих пикселей в массив или вектор. Например, распределение пикселей для типичного изображения цифр 4 и 5 показано на рисунке ниже.



Из рисунка видно, что распределение пикселей (особенно тех, которые имеют тенденцию к белому тону) различается и это отличает цифры, которые они представляют. Мы передадим это распределение в 784 пикселя в нашу сеть в качестве входных данных. Выходные данные сети будут состоять из 10 категорий, представляющих цифру от 0 до 9.

Сеть будет состоять из 4 слоев — одного входного слоя, одного выходного слоя и двух скрытых слоев. Каждый скрытый слой будет содержать 512 узлов. Каждый слой полностью соединен со следующим слоем. Когда мы обучим сеть, мы будем вычислять веса для каждого соединения. Мы будем обучать сеть, применяя обратное распространение и градиентный спуск, которые обсуждали в лекции №4.

Импорт библиотек

Сначала мы импортируем различные библиотеки, необходимые для реализации проекта.

Обработка массивов и построение графиков

Обычно, мы используем «numpy» для обработки массива и matplotlib для построения графиков. Эти библиотеки импортируются в проект с использованием следующих инструкций импорта:

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plot
import tensorflow
import keras
```

Подавление предупреждений

Поскольку и Tensorflow, и Keras продолжают развиваться, если вы не синхронизируете их соответствующие версии в проекте, во время выполнения вы увидите множество предупреждений. Мы отключим предупреждения в этом проекте. Это делается с помощью следующих строк кода:

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL']='3'
import warnings
warnings.filterwarnings('ignore')
```

```
from tensorflow.python.util import deprecation
deprecation._PRINT_DEPRECATION_WARNINGS = False
```

Используем библиотеки фреймворка Keras для импорта набора данных. Мы будем использовать набор данных «mnist» для рукописных цифр. Импортируем необходимый пакет, используя следующую инструкцию:

```
from keras.datasets import mnist
```

Определим нейронную сеть глубокого обучения с использованием пакетов Keras. Для этого импортируем пакеты Sequential, Dense, Dropout и Activation для определения сетевой архитектуры. Также используем np_utils для нескольких утилит, которые необходимы в проекте. Импорты выполняются с помощью следующих инструкций:

```
from keras.models import Sequential, load_model
from tensorflow.python.keras.layers.core import Dense, Dropout,
Activation
from tensorflow.python.keras.utils import np_utils
```

В итоге, ячейка с импортами будет выглядеть следующим образом:

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plot
import tensorflow
import keras

# silent all warnings
import os
os.environ['TF_CPP_MIN_LOG_LEVEL']='3'
import warnings
warnings.filterwarnings('ignore')
from tensorflow.python.util import deprecation
deprecation._PRINT_DEPRECATION_WARNINGS = False

from keras.datasets import mnist
from keras.models import Sequential, load_model
from tensorflow.python.keras.layers.core import Dense, Dropout, Activation
from tensorflow.python.keras.utils import np_utils
```

Создание модели глубокого обучения

Модель нейронной сети будет состоять из линейного набора слоев. Чтобы определить такую модель, мы вызываем последовательную функцию:

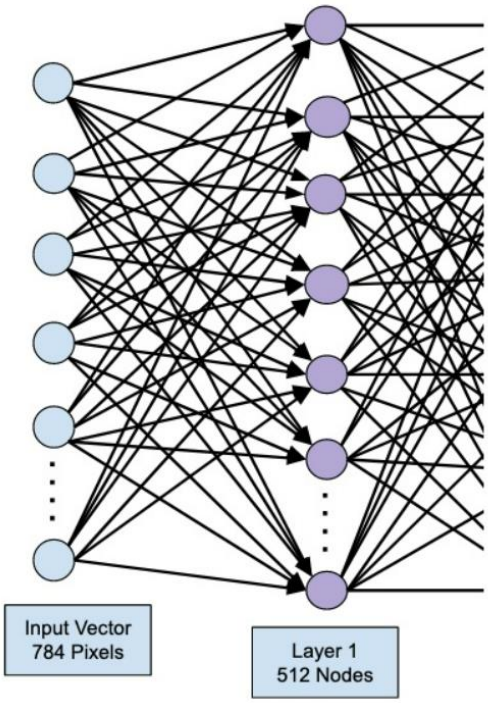
```
model = Sequential()
```

Входной слой

Определяем входной уровень, который является первым уровнем в нейронной сети, используя следующую инструкцию:

```
model.add(Dense(512, input_shape=(784,)))
```

Она создает слой с 512 узлами (нейронами) и 784 входными узлами, как показано на рисунке ниже:



Обратите внимание, что все входные узлы полностью подключены к уровню 1, то есть каждый входной узел подключен ко всем 512 узлам уровня 1.

Далее нам нужно добавить функцию активации для вывода уровня 1. Мы будем использовать «ReLU» в качестве активации. Функция активации добавляется с помощью следующей инструкции:

```
model.add(Activation('relu'))
```

Затем мы добавляем Dropout 20%, используя приведенную ниже инструкцию. Dropout — это метод, используемый для предотвращения переобучения модели.

```
model.add(Dropout(0.2))
```

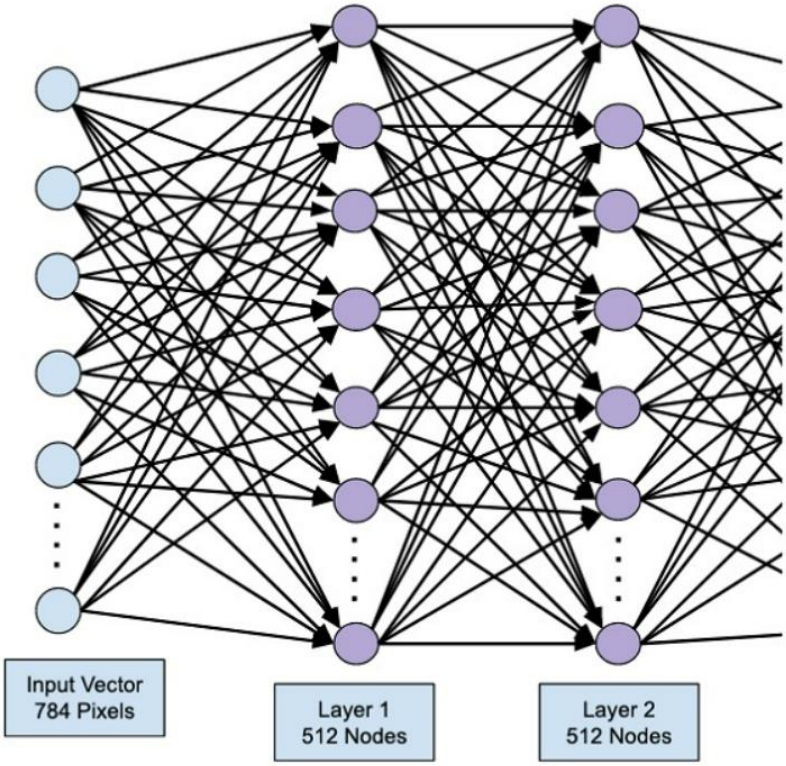
Скрытый слой

На данный момент входной слой полностью определен. Далее мы добавим скрытый слой.

Скрытый слой будет состоять из 512 узлов. Входные данные для скрытого слоя идут из ранее определенного входного слоя. Все узлы полностью связаны, как и в предыдущем случае. Выходные данные скрытого слоя перейдут к следующему слою в сети, который станет последним и выходным слоем. Мы будем использовать ту же активацию «ReLU», что и для предыдущего слоя, и Dropout 20%. Код показан ниже:

```
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.2))
```

Сеть на этом этапе можно визуализировать следующим образом:



Далее добавим последний слой в нейронную сеть, который является выходным слоем. Обратите внимание, что вы можете добавить любое количество скрытых слоев, используя код, аналогичный тому, который был использован выше. Добавление дополнительных слоев сделало бы сеть сложной для обучения; тем не менее, это дает определенное преимущество в виде лучших результатов во многих, хотя и не во всех, случаях.

Выходной слой

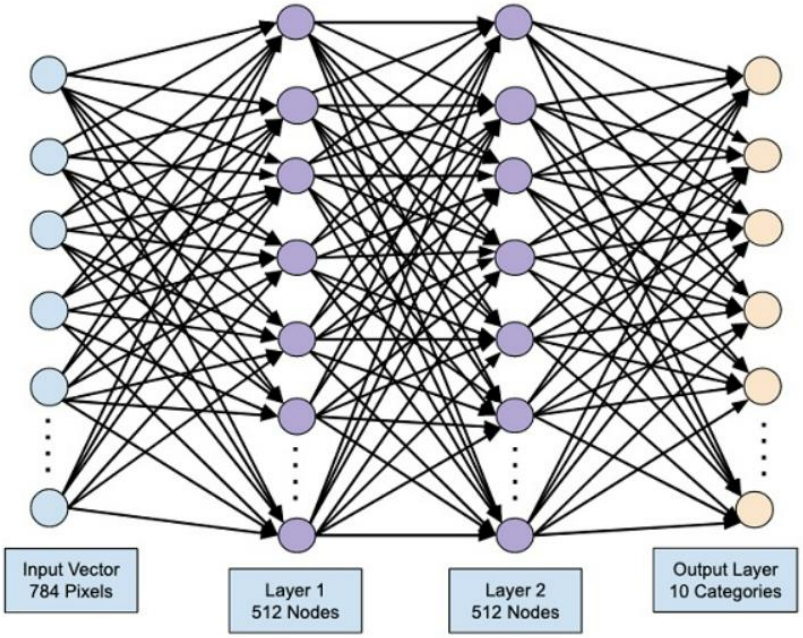
Выходной слой состоит всего из 10 узлов, так как мы хотим классифицировать данные изображения по 10 различным цифрам. Добавляем этот слой следующим образом:

```
model.add(Dense(10))
```

Поскольку мы хотим классифицировать вывод по 10 различным единицам, мы используем активацию «softmax». В случае «ReLU» вывод двоичный. Добавим активацию:

```
model.add(Activation('softmax'))
```

На данный момент нашу сеть можно визуализировать, как показано на рисунке ниже:



Полный код ячейки будет выглядеть следующим образом:

```
model = Sequential()
model.add(Dense(512, input_shape=(784,)))
model.add(Activation('relu'))
model.add(Dropout(0.2))

model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.2))

model.add(Dense(10))
model.add(Activation('softmax'))
```

Далее нам нужно скомпилировать модель.

Компиляция модели

Компиляция выполняется с использованием одного единственного вызова метода, который называется «compile»:

```
model.compile(loss='categorical_crossentropy', metrics=['accuracy'],
optimizer='adam')
```

Метод компиляции требует несколько параметров. Параметр потерь имеет тип «categorical_crossentropy». Параметр метрики установлен на «accuracy», и, наконец, мы используем оптимизатор Adam (Adaptive Moment Estimation) для обучения сети.

Теперь мы готовы передать данные в нейронную сеть.

Загрузка данных

Как было сказано ранее, мы будем использовать набор данных «mnist», предоставленный Keras. Когда мы загружаем данные в систему, мы разделяем их на обучающие и тестовые данные. Данные загружаются путем вызова метода «load_data» следующим образом:

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

Код в ячейке будет выглядеть следующим образом:

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

Изучим структуру загруженного набора данных

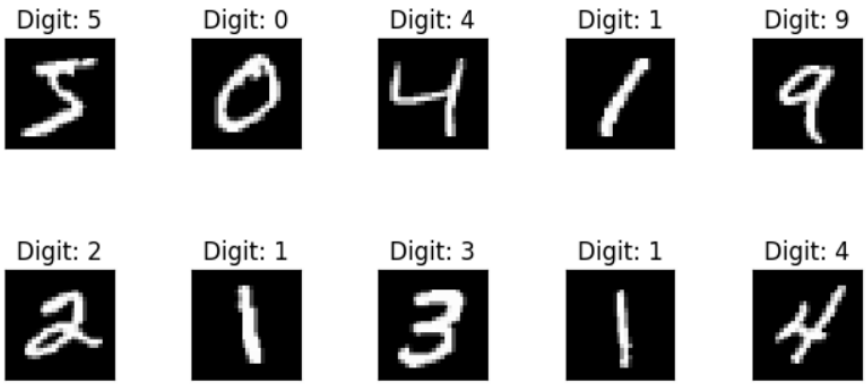
Данные, которые нам предоставляются, представляют собой графические изображения размером 28 x 28 пикселей, каждое из которых содержит одну цифру от 0 до 9. Отобразим первые десять изображений в консоли. Код показан ниже:

```
# printing first 10 images
for i in range(10):

    plot.subplot(3,5,i+1)
    plot.tight_layout()
    plot.imshow(X_train[i], cmap='gray', interpolation='none')
    plot.title("Digit: {}".format(y_train[i]))
    plot.xticks([])
    plot.yticks([])
```

```
# printing first 10 images
for i in range(10):

    plot.subplot(3,5,i+1)
    plot.tight_layout()
    plot.imshow(X_train[i], cmap='gray', interpolation='none')
    plot.title("Digit: {}".format(y_train[i]))
    plot.xticks([])
    plot.yticks([])
```

В цикле из 10 шагов мы создаем подграфик на каждой итерации и показываем в нем изображение из вектора «X_train». Мы даем название каждому изображению из соответствующего вектора «y_train». Обратите внимание, что вектор «y_train» содержит фактические значения для соответствующего изображения в векторе «X_train». Удаляем маркировку осей «x» и «y», вызывая два метода «xticks» и «yticks» с нулевым аргументом.

После запуска ячейки вывод будет следующим:



Подготовим данные для загрузки их в нейронную сеть

Прежде чем мы отправим данные в сеть, их необходимо преобразовать в формат, требуемый сетью. Этот этап называется подготовкой данных для сети. Обычно он состоит из преобразования многомерных входных данных в одномерный вектор и нормализации точек данных.

Преобразование входного вектора

Изображения в нашем наборе данных состоят из 28 x 28 пикселей. Его необходимо преобразовать в одномерный вектор размером 28 * 28 = 784 для подачи в нашу сеть. Мы делаем это, вызывая метод «reshape» для вектора:

```
X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)
```

Теперь обучающий вектор будет состоять из 60 000 точек данных, каждая из которых состоит из одномерного вектора размера 784. Точно так же наш тестовый вектор будет состоять из 10 000 точек данных одномерного вектора размера 784.

Нормализация данных

Данные, которые содержит входной вектор, в настоящее время имеют дискретное значение от 0 до 255 — уровни шкалы серого. Нормализация этих значений пикселей между 0 и 1 помогает ускорить обучение. Поскольку мы собираемся использовать стохастический градиентный спуск, нормализация данных также поможет уменьшить вероятность застревания в локальных оптимумах.

Чтобы нормализовать данные, мы представляем их как тип с плавающей точкой и делим на 255, как показано в следующем фрагменте кода:

```
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
```

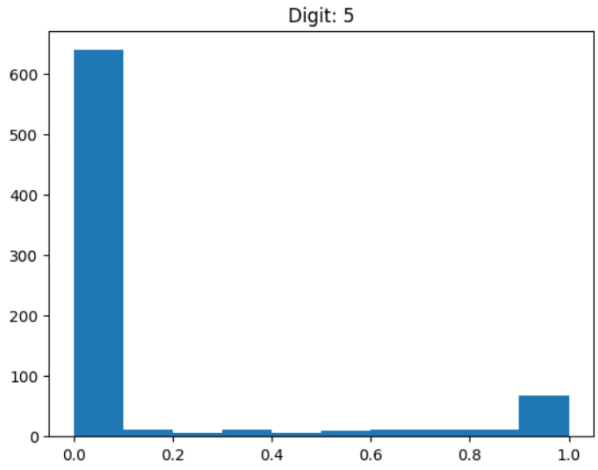
Давайте теперь посмотрим, как выглядят нормализованные данные.

Изучение нормализованных данных

Чтобы просмотреть нормализованные данные, вызовем функцию гистограммы, как показано ниже:

```
plot.hist(X_train[0])
plot.title("Digit: {}".format(y_train[0]))
```

Здесь мы строим гистограмму первого элемента вектора «X_train». Мы также печатаем цифру, представленную этой точкой данных. Результат выполнения приведенного выше кода показан ниже:



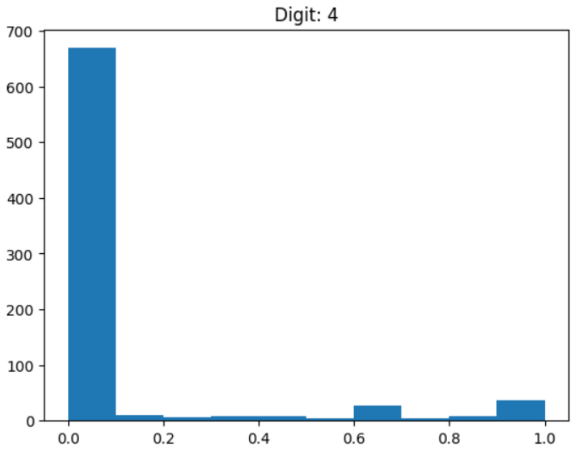
Полный код ячейки будет выглядеть следующим образом:

```
X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)

X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255

plot.hist(X_train[0])
plot.title("Digit: {}".format(y_train[0]))
```

Из рисунка видно, что имеется густая плотность точек, имеющих значение, близкое к нулю. Это черные точки на изображении, которые, очевидно, составляют большую часть изображения. Остальные точки серой шкалы, близкие к белому цвету, представляют цифру. Можно проверить распределение пикселей для другой цифры. Код ниже (его можно разместить в следующей ячейке) печатает гистограмму цифры с индексом 2 в наборе обучающих данных:

```
plot.hist(X_train[2])
plot.title("Digit: {}".format(y_train[2])
```

Результат выполнения вышеуказанного кода показан ниже:



Сравнив два приведенных выше рисунка, можно заметить, что распределение белых пикселей на двух изображениях различается, что указывает на представление другой цифры — «5» и «4» на двух приведенных выше изображениях.

Далее рассмотрим распределение данных в нашем полном наборе обучающих данных.

Изучение распределения данных

Прежде чем мы обучим модель машинного обучения на наборе данных, мы должны знать распределение уникальных цифр в нем. Изображения представляют 10 различных цифр в диапазоне от 0 до 9. Мы хотели бы знать количество цифр 0, 1 и т. д. в нашем наборе данных. Мы можем получить эту информацию, используя уникальный метод «Numpy»:

Используйте следующую команду, чтобы напечатать количество уникальных значений и количество вхождений каждого из них (вставьте код в следующую ячейку):

```
print(np.unique(y_train, return_counts=True))
```

```
print(np.unique(y_train, return_counts=True))
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=uint8), array([5923, 6742, 5958, 6131, 5842, 5421, 5918, 6265, 5851, 5949],
      dtype=int64))
```

Он показывает, что существует 10 различных значений — от 0 до 9. Цифра 0 встречается 5923 раза, цифра 1 встречается 6742 раза и так далее.

Кодирование данных

У нас есть десять категорий в наборе данных. Таким образом, мы будем кодировать вывод в этих десяти категориях, используя однократное кодирование. Мы используем метод «to_categorial» утилиты «Numpy» для выполнения кодирования. После кодирования выходных данных каждая точка данных будет преобразована в одномерный вектор размера 10. Например, цифра 5 теперь будет представлена как [0,0,0,0,0,1,0,0,0 ,0].

Закодируем данные, используя следующий фрагмент кода:

```
n_classes = 10
Y_train = np_utils.to_categorical(y_train, n_classes)
```

Мы можем проверить результат кодирования, напечатав первые 5 элементов категоризированного вектора «Y_train».

Используйте следующий код для печати первых 5 векторов:

```
for i in range(5):
   print (Y_train[i])
```

Код ячейки и вывод показан ниже:

```
n_classes = 10
Y_train = np_utils.to_categorical(y_train, n_classes)

for i in range(5):
   print (Y_train[i])
```

```
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
[0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
```

Первый элемент представляет цифру 5, второй представляет цифру 0 и так далее.

Наконец, нам также придется классифицировать тестовые данные, что делается с помощью следующей инструкции (в следующей ячейке):

```
Y_test = np_utils.to_categorical(y_test, n_classes)
```

На этом этапе данные полностью готовы к вводу в нейронную сеть.

Далее наступает самая важная часть — обучение сетевой модели.

Обучение модели

Обучение модели выполняется одним вызовом метода под названием «fit», который принимает несколько параметров, как показано в коде ниже:

```
history = model.fit(X_train, Y_train, batch_size=128, epochs=20,
verbose=2, validation_data=(X_test, Y_test))
```

```
history = model.fit(X_train, Y_train, batch_size=128, epochs=20, verbose=2, validation_data=(X_test, Y_test))
```

Первые два параметра метода подгонки определяют функции и выходные данные обучающего набора данных. Эпохи установлены на 20; мы предполагаем, что обучение сойдется максимум за 20 эпох — итераций. Обученная модель проверяется на тестовых данных, как указано в последнем параметре.

Частичный вывод выполнения вышеуказанной команды показан ниже:

```
Epoch 1/20
469/469 - 7s - loss: 0.2480 - accuracy: 0.9248 - val_loss: 0.0996 - val_accuracy: 0.9686 - 7s/epoch - 15ms/step
Epoch 2/20
469/469 - 6s - loss: 0.1006 - accuracy: 0.9691 - val_loss: 0.0743 - val_accuracy: 0.9766 - 6s/epoch - 13ms/step
Epoch 3/20
469/469 - 6s - loss: 0.0723 - accuracy: 0.9774 - val_loss: 0.0718 - val_accuracy: 0.9781 - 6s/epoch - 13ms/step
Epoch 4/20
469/469 - 6s - loss: 0.0563 - accuracy: 0.9815 - val_loss: 0.0687 - val_accuracy: 0.9783 - 6s/epoch - 14ms/step
Epoch 5/20
469/469 - 6s - loss: 0.0445 - accuracy: 0.9857 - val_loss: 0.0655 - val_accuracy: 0.9797 - 6s/epoch - 14ms/step
Epoch 6/20
469/469 - 6s - loss: 0.0404 - accuracy: 0.9866 - val_loss: 0.0710 - val_accuracy: 0.9798 - 6s/epoch - 14ms/step
Epoch 7/20
469/469 - 6s - loss: 0.0331 - accuracy: 0.9889 - val_loss: 0.0684 - val_accuracy: 0.9809 - 6s/epoch - 13ms/step
Epoch 8/20
469/469 - 6s - loss: 0.0305 - accuracy: 0.9898 - val_loss: 0.0678 - val_accuracy: 0.9821 - 6s/epoch - 13ms/step
Epoch 9/20
469/469 - 6s - loss: 0.0286 - accuracy: 0.9903 - val_loss: 0.0609 - val_accuracy: 0.9832 - 6s/epoch - 13ms/step
Epoch 10/20
469/469 - 6s - loss: 0.0273 - accuracy: 0.9902 - val_loss: 0.0683 - val_accuracy: 0.9808 - 6s/epoch - 13ms/step
```

…

Теперь, когда модель обучена на обучающем наборе, мы оценим ее производительность.

Оценка производительности модели

Чтобы оценить производительность модели, вызываем метод оценки следующим образом:

```
loss_and_metrics = model.evaluate(X_test, Y_test, verbose=2)
```

Мы напечатаем потери и точность, используя следующие две инструкции:

```
print("Test Loss", loss_and_metrics[0])
print("Test Accuracy", loss_and_metrics[1])
```

Вывод и полный код ячейки показан ниже:

```
loss_and_metrics = model.evaluate(X_test, Y_test, verbose=2)
print("Test Loss", loss_and_metrics[0])
print("Test Accuracy", loss_and_metrics[1])
```

```
313/313 - 1s - loss: 0.0699 - accuracy: 0.9848 - 1s/epoch - 4ms/step
Test Loss 0.0699140653014183
Test Accuracy 0.9847999811172485
```

Выше мы можем видеть точность теста 98%, что должно быть приемлемым для нас. Что означает, что в 2% случаев рукописные цифры не будут правильно классифицированы. Мы также построим метрики точности и потерь, чтобы увидеть, как модель работает на тестовых данных.

Построение показателей точности

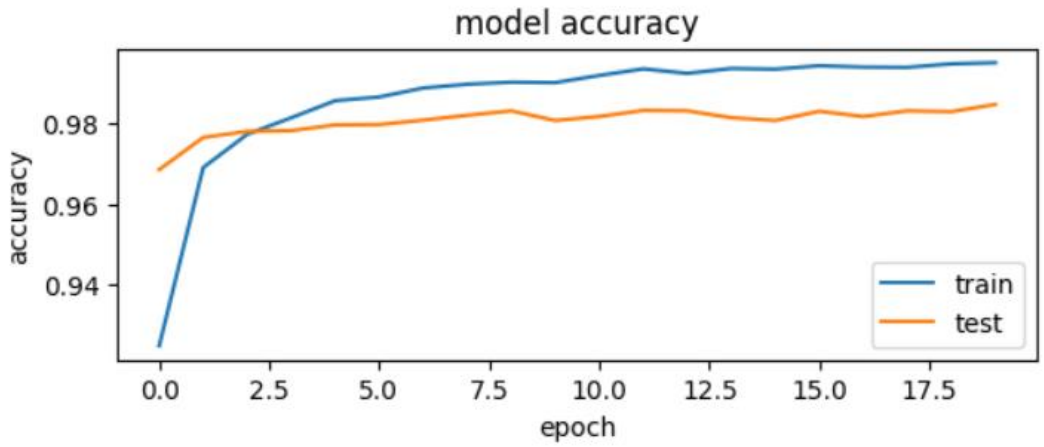
Мы используем записанную историю во время нашего обучения, чтобы получить график показателей точности. Следующий код отображает точность для каждой эпохи. Мы выбираем точность данных обучения («accuracy») и точность данных проверки («val_ accuracy») для построения графика.

```
plot.subplot(2,1,1)
plot.plot(history.history['accuracy'])
plot.plot(history.history['val_accuracy'])
plot.title('model accuracy')
plot.ylabel('accuracy')
plot.xlabel('epoch')
plot.legend(['train', 'test'], loc='lower right')
```

Вывод и полный код ячейки показан ниже:

```
plot.subplot(2,1,1)
plot.plot(history.history['accuracy'])
plot.plot(history.history['val_accuracy'])
plot.title('model accuracy')
plot.ylabel('accuracy')
plot.xlabel('epoch')
plot.legend(['train', 'test'], loc='lower right')
```

```
<matplotlib.legend.Legend at 0x20e003252d0>
```



Как вы можете видеть на диаграмме, точность быстро увеличивается в первые две эпохи, что указывает на быстрое обучение сети. После этого кривая выравнивается, указывая на то, что для дальнейшего обучения модели требуется не слишком много эпох. Как правило, если точность обучающих данных («accuracy») продолжает улучшаться, а точность проверочных данных («val_ accuracy») ухудшается, вы сталкиваетесь с переобучением. Это указывает на то, что модель начинает запоминать данные.

Мы также построим метрики потерь, чтобы проверить производительность нашей модели.

График показателей потерь

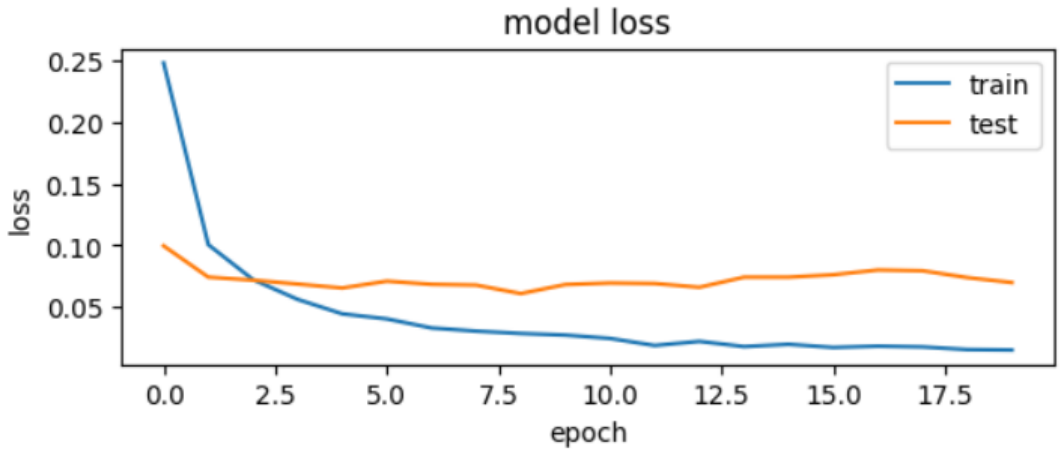
Опять же, мы наносим на график потери как на обучающие, так и на тестовые данные. Это делается с помощью следующего кода:

```
plot.subplot(2,1,2)
plot.plot(history.history['loss'])
plot.plot(history.history['val_loss'])
plot.title('model loss')
plot.ylabel('loss')
plot.xlabel('epoch')
plot.legend(['train', 'test'], loc='upper right')
```

Вывод и полный код ячейки показан ниже:

```
plot.subplot(2,1,2)
plot.plot(history.history['loss'])
plot.plot(history.history['val_loss'])
plot.title('model loss')
plot.ylabel('loss')
plot.xlabel('epoch')
plot.legend(['train', 'test'], loc='upper right')
```

```
<matplotlib.legend.Legend at 0x20e769fca90>
```



Как вы можете видеть на диаграмме, потери на тренировочном наборе быстро уменьшаются для первых двух эпох. Для тестового набора потери не уменьшаются с той же скоростью, что и для тренировочного набора, но остаются почти неизменными в течение нескольких эпох. Это означает, что наша модель хорошо обобщает невидимые данные.

Теперь мы будем использовать нашу обученную модель для прогнозирования цифр в тестовых данных.

Прогнозирование тестовых данных

Предсказать цифры в невидимых данных достаточно просто. Необходимо вызвать метод модели «model.predict»:

```
predictions = np.argmax(model.predict(X_test),axis=1)
```

Вызов метода возвращает прогнозы в виде вектора, который можно проверить на наличие нулей и единиц в сравнении с фактическими значениями. Это делается с помощью следующих двух инструкций:

```
correct_predictions = np.nonzero(predictions == y_test)[0]
incorrect_predictions = np.nonzero(predictions != y_test)[0]
```

Наконец, мы напечатаем количество правильных и неправильных прогнозов, используя следующие две инструкции:

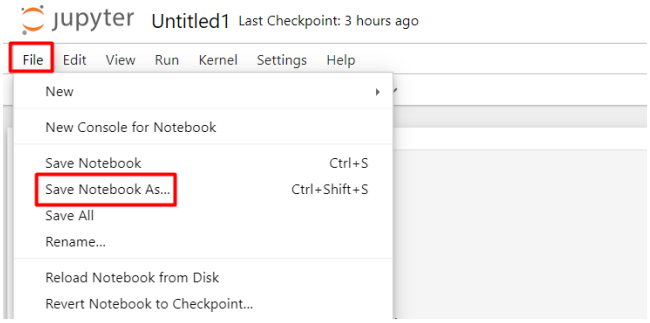
```
print(len(correct_predictions)," classified correctly")
print(len(incorrect_predictions)," classified incorrectly")
```

Вывод и полный код ячейки показан ниже:

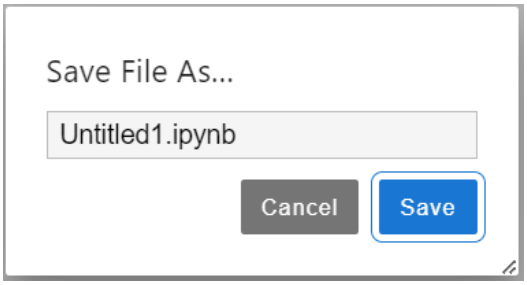
```
predictions = np.argmax(model.predict(X_test),axis=1)
correct_predictions = np.nonzero(predictions == y_test)[0]
incorrect_predictions = np.nonzero(predictions != y_test)[0]
print(len(correct_predictions)," classified correctly")
print(len(incorrect_predictions)," classified incorrectly")
```

```
313/313 [==============================] - 1s 3ms/step
9848  classified correctly
152  classified incorrectly
```

Далее мы можем сохранить проект для того, чтобы использовать его в будущем на других данных.



Меню File -> Save Notebook As…



Введите имя файла и нажмите «Save»

Файл будет сохранен в папке проекта. Вы можете его найти в корневом каталоге проекта VS Code.

Лекция №5. Scikit-learn для задач машинного обучения

Что такое Scikit-learn?

Scikit-learn (также известный как sklearn) - это библиотека с открытым исходным кодом для машинного обучения в Python. Он предоставляет простой и эффективный набор инструментов для различных задач машинного обучения, таких как классификация, регрессия, кластеризация, обработка данных и многое другое. Scikit-learn основан на других популярных библиотеках Python, таких как NumPy, SciPy и Matplotlib, что делает его мощным и гибким инструментом для исследования и решения задач машинного обучения.

Приложение Scikit-learn

Scikit-learn широко используется для решения различных задач машинного обучения и статистического анализа данных. Его гибкий API позволяет легко создавать и обучать модели машинного обучения. Некоторые из основных приложений Scikit-learn включают:

1. **Классификация**: Решение задачи классификации, где модель предсказывает категорию или класс объекта на основе его признаков. Примеры включают классификацию писем на спам и не спам, определение типа цветка на основе его характеристик и другие.
2. **Регрессия**: Решение задачи регрессии, где модель предсказывает численное значение или непрерывный выход на основе входных данных. Примеры включают прогнозирование цен на недвижимость, предсказание дохода клиента и т.д.
3. **Кластеризация**: Группировка объектов в кластеры на основе их схожести. Это может быть полезным для обнаружения паттернов в данных, а также для сжатия их представления.
4. **Обработка текстовых данных**: Scikit-learn предоставляет удобные инструменты для векторизации текстовых данных и обработки их с использованием методов машинного обучения.
5. **Измерение производительности моделей**: Scikit-learn предоставляет метрики и инструменты для оценки производительности моделей и выбора наилучшей модели для задачи.

Установка Scikit-learn

Установка Scikit-learn производится с помощью менеджера пакетов pip:

```
pip install scikit-learn
```

Scikit-learn также может быть установлен совместно с другими пакетами для анализа данных, такими как NumPy и SciPy.

Особенности Scikit-learn

1. Простой и интуитивно понятный API: Scikit-learn предоставляет простой и логичный API, который позволяет быстро прототипировать и обучать модели.
2. Отличная документация: Библиотека обладает подробной документацией и множеством примеров, что делает процесс изучения и использования библиотеки более простым.

3. Расширяемость: Scikit-learn позволяет расширять функциональность с помощью собственных классов и функций, что делает его гибким инструментом для решения различных задач.

Плюсы использования Scikit-learn

- Простота использования и прототипирования моделей.
- Богатый выбор алгоритмов машинного обучения и статистических методов.
- Отличная поддержка для измерения производительности моделей.
- Расширяемость и возможность создания собственных методов и функций.

Минусы использования Scikit-learn

- Ограниченность в некоторых продвинутых возможностях, доступных в других фреймворках.
- Не все алгоритмы машинного обучения доступны в Scikit-learn.
- Некоторые методы обработки данных могут быть не слишком эффективными для очень больших наборов данных.

Как уже было сказано выше, Scikit-learn предоставляет мощные инструменты для решения различных задач машинного обучения, таких как классификация, регрессия и кластеризация. Рассмотрим примеры применения Scikit-learn для каждой из этих задач.

1. Классификация:

Классификация - это задача машинного обучения, в которой модель пытается отнести объекты к определенным категориям или классам на основе их признаков. Примером задачи классификации может быть определение, является ли письмо спамом или не спамом, или определение типа цветка на основе его характеристик.

Пример кода для задачи классификации с использованием логистической регрессии в Scikit-learn:

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Подготовка данных
X, y = load_dataset()  # Загрузка данных, X - признаки, y - метки
классов
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Создание модели и обучение
model = LogisticRegression()
model.fit(X_train, y_train)
```

```
# Предсказание на тестовых данных
y_pred = model.predict(X_test)

# Оценка производительности модели
accuracy = accuracy_score(y_test, y_pred)
print("Точность классификации:", accuracy)
```

2. Регрессия:

Регрессия - это задача машинного обучения, в которой модель пытается предсказать численное значение или непрерывный выход на основе входных данных. Примерами задач регрессии могут быть прогнозирование цен на недвижимость или предсказание дохода клиента на основе его характеристик.

Пример кода для задачи регрессии с использованием линейной регрессии в Scikit-learn:

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Подготовка данных
X, y = load_dataset()  # Загрузка данных, X - признаки, y - значения
целевой переменной
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Создание модели и обучение
model = LinearRegression()
model.fit(X_train, y_train)

# Предсказание на тестовых данных
y_pred = model.predict(X_test)

# Оценка производительности модели
mse = mean_squared_error(y_test, y_pred)
print("Среднеквадратичная ошибка:", mse)
```

3. Кластеризация:

Кластеризация - это задача машинного обучения, в которой модель пытается группировать объекты в кластеры на основе их схожести. Кластеризация может быть полезна для обнаружения паттернов в данных и сжатия их представления.

Пример кода для задачи кластеризации с использованием метода K-средних в Scikit-learn:

```
from sklearn.cluster import KMeans
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Подготовка данных
X = load_dataset()  # Загрузка данных, X - признаки
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Уменьшение размерности данных с помощью метода главных компонент
(PCA)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Создание модели кластеризации и обучение
model = KMeans(n_clusters=3)
model.fit(X_scaled)

# Предсказание кластеров для данных
clusters = model.predict(X_scaled)

# Визуализация результатов
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=clusters)
plt.xlabel('Главная компонента 1')
plt.ylabel('Главная компонента 2')
plt.show()
```

Scikit-learn - мощный и популярный фреймворк для машинного обучения в Python. Он обладает простым и эффективным API, множеством алгоритмов и инструментов для решения различных задач. Он облегчает разработку и применение моделей на различных типах данных. Несмотря на некоторые ограничения, Scikit-learn остается одним из наиболее популярных выборов для начинающих и опытных исследователей машинного обучения и анализа данных.

Лабораторная работа №5 Применение Scikit-learn для задач классификации, регрессии и кластеризации

Цель работы:

Целью данной лабораторной работы является знакомство с библиотекой Scikit-learn и применение ее инструментов для решения задач классификации, регрессии и кластеризации на различных наборах данных.

Шаги выполнения работы:

1. Установка и импорт библиотеки: Установите библиотеку Scikit-learn с помощью pip, если она не установлена. Импортируйте необходимые модули из библиотеки.
2. Загрузка данных: Выберите три различных набора данных, подходящих для задач классификации, регрессии и кластеризации. Возможно, вы можете использовать наборы данных, доступные в Scikit-learn или загрузить свои собственные данные.
3. Задача классификации: Используйте модель классификации из Scikit-learn (например, логистическую регрессию или метод опорных векторов) для решения задачи классификации на одном из выбранных наборов данных. Оцените производительность модели с помощью соответствующих метрик, таких как точность или F1-мера.
4. Задача регрессии: Используйте модель регрессии из Scikit-learn (например, линейную регрессию или метод ближайших соседей) для решения задачи регрессии на другом выбранном наборе данных. Оцените качество модели с помощью метрик, таких как среднеквадратичная ошибка или коэффициент детерминации (R^2).
5. Задача кластеризации: Используйте алгоритм кластеризации из Scikit-learn (например, метод K-средних или DBSCAN) для разделения данных на кластеры на третьем выбранном наборе данных. Визуализируйте результаты кластеризации для наглядности.
6. Выводы: Сделайте выводы о производительности моделей и алгоритмов, а также о применимости Scikit-learn для решения различных задач машинного обучения.

Примечание:

Не забудьте документировать ваш код, включить пояснения к каждому шагу и предоставить графики для наглядности результатов. В отчете по лабораторной работе также укажите, какие метрики были использованы для оценки производительности моделей и какие выводы можно сделать из результатов экспериментов.

Реализация задания на лабораторную работу

Создайте новую папку на рабочем столе и откройте ее через командную строку.

```
# Создание виртуальной среды с именем 'myenv'
python -m venv myenv

# Откройте редактор кода VS Code. Откройте новый терминал и
активируйет созданную на прошлом шаге виртуальную среду

# Активация виртуальной среды в Windows
```

```
myenv\Scripts\activate.bat
```

Шаг 1: Установка необходимых библиотек

Для начала убедитесь, что у вас установлен Python и менеджер пакетов pip. Затем установите необходимые пакеты:

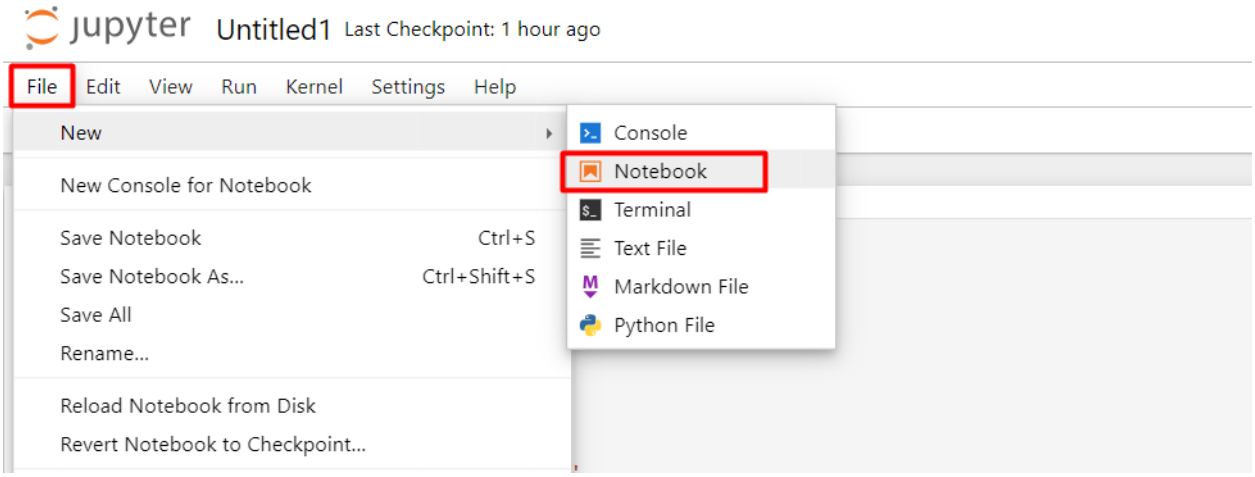
```
pip install scikit-learn numpy pandas matplotlib
```

Шаг 2: Запуск Jupyter Notebook

Запустите Jupyter Notebook с помощью команды:

```
jupyter notebook
```

Jupyter Notebook откроется в вашем веб-браузере. Создайте новый ноутбук и начните кодирование примера.



Шаг 3: Импорт необходимых модулей

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

from sklearn.datasets import load_iris, fetch_california_housing,
make_blobs
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score, mean_squared_error
```

Шаг 4: Загрузка данных

В этой лабораторной работе мы будем использовать три набора данных из библиотеки Scikit-learn:

а) Набор данных Iris для задачи классификации

б) Набор данных Boston для задачи регрессии

в) Набор данных для кластеризации (сгенерируем его с помощью функции make_blobs())

```
# Загрузка наборов данных
iris = load_iris()
housing = fetch_california_housing()
X_blob, _ = make_blobs(n_samples=300, centers=4, random_state=42)
```

Здесь мы загружаем три различных набора данных из библиотеки Scikit-learn: набор данных Iris, набор данных Boston и сгенерированный набор данных для кластеризации.

Шаг 5: Задача классификации

Для задачи классификации мы будем использовать набор данных Iris и модель логистической регрессии.

Мы подготавливаем данные для задачи классификации, разделяя их на обучающий и тестовый наборы с использованием функции «train_test_split».

```
# Подготовка данных для классификации
X_class, y_class = iris.data, iris.target
X_class_train, X_class_test, y_class_train, y_class_test =
train_test_split(X_class, y_class, test_size=0.2, random_state=42)

# Нормализация данных
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_class_train_scaled = scaler.fit_transform(X_class_train)
X_class_test_scaled = scaler.transform(X_class_test)
```

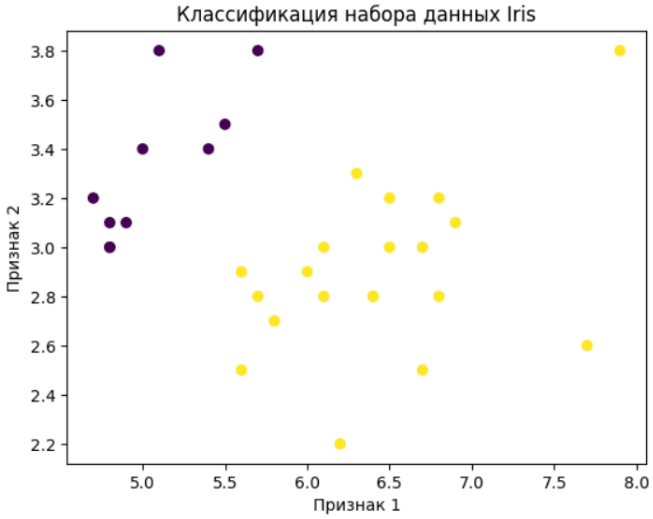
Здесь мы создаем и обучаем модель логистической регрессии на обучающих данных.

```
# Создание и обучение модели классификации с увеличенным числом
итераций
clf = LogisticRegression(max_iter=1000)
clf.fit(X_class_train_scaled, y_class_train)

# Предсказание на тестовых данных для задачи классификации
y_class_pred = clf.predict(X_class_test)
```

Модель выполняет предсказание на тестовых данных для задачи классификации.

Далее мы оцениваем производительность модели классификации, сравнивая предсказанные метки с истинными метками тестового набора и выводим точность классификации.

```
# Оценка производительности модели классификации
accuracy = accuracy_score(y_class_test, y_class_pred)
print("Точность классификации:", accuracy)

# Визуализация результатов классификации
plt.scatter(X_class_test[:, 0], X_class_test[:, 1], c=y_class_pred,
cmap='viridis')
plt.xlabel('Признак 1')
plt.ylabel('Признак 2')
plt.title('Классификация набора данных Iris')
plt.show()
```



Шаг 6: Задача регрессии

Для задачи регрессии мы будем использовать набор данных Boston и модель линейной регрессии.

```
# Подготовка данных для регрессии
X_reg, y_reg = housing.data, housing.target
X_reg_train, X_reg_test, y_reg_train, y_reg_test =
train_test_split(X_reg, y_reg, test_size=0.2, random_state=42)
```

Мы подготавливаем данные для задачи регрессии, разделяя их на обучающий и тестовый наборы с использованием функции «train_test_split».

Создаем и обучаем модель линейной регрессии на обучающих данных.

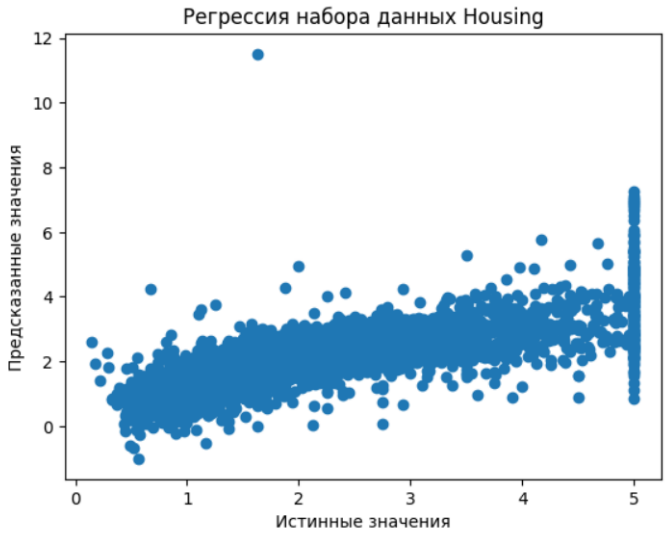
```
# Создание и обучение модели регрессии
reg = LinearRegression()
reg.fit(X_reg_train, y_reg_train)
```

Модель выполняет предсказание на тестовых данных для задачи регрессии.

```
# Предсказание на тестовых данных для задачи регрессии
y_reg_pred = reg.predict(X_reg_test)
```

Мы оцениваем качество модели регрессии с помощью среднеквадратичной ошибки и выводим результат.

```
# Оценка качества модели регрессии
mse = mean_squared_error(y_reg_test, y_reg_pred)
print("Среднеквадратичная ошибка:", mse)

# Визуализация результатов регрессии
plt.scatter(y_reg_test, y_reg_pred)
plt.xlabel('Истинные значения')
plt.ylabel('Предсказанные значения')
plt.title('Регрессия набора данных Housing')
plt.show()
```



Шаг 7: Задача кластеризации

Для задачи кластеризации мы будем использовать сгенерированный набор данных X_blob и алгоритм K-средних.

Создаем и обучаем модель кластеризации методом K-средних на сгенерированных данных.

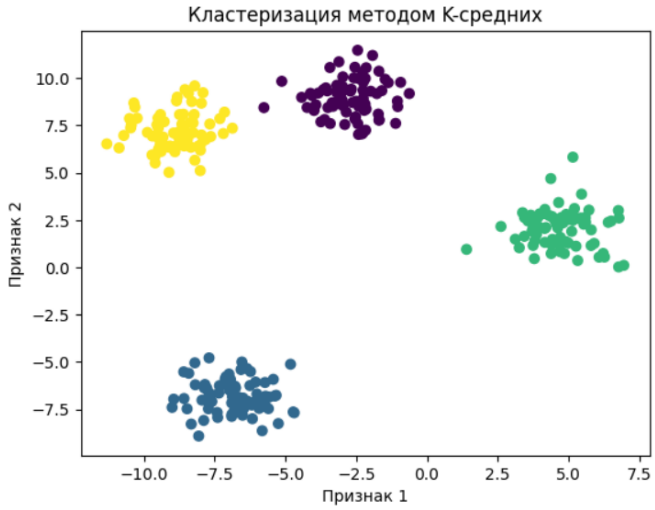
```
# Создание модели кластеризации и обучение с явным указанием параметра
n_init
kmeans = KMeans(n_clusters=4, random_state=42, n_init=10)
kmeans.fit(X_blob)
```

Модель выполняет предсказание кластеров для сгенерированных данных.

```
# Предсказание кластеров для данных
clusters = kmeans.predict(X_blob)
```

Визуализируем результаты кластеризации на графике для лучшего понимания.

```
# Визуализация результатов кластеризации
plt.scatter(X_blob[:, 0], X_blob[:, 1], c=clusters, cmap='viridis')
plt.xlabel('Признак 1')
plt.ylabel('Признак 2')
plt.title('Кластеризация методом K-средних')
plt.show()
```



Выше мы применили библиотеку Scikit-learn для решения задач классификации, регрессии и кластеризации на различных наборах данных. В ходе работы были использованы модели логистической регрессии и линейной регрессии, а также алгоритм K-средних для кластеризации. Каждая задача была выполнена с помощью соответствующих метрик для оценки производительности моделей и алгоритмов.

Выполнение каждого из этих блоков кода в Jupyter Notebook отдельно позволит вам увидеть результаты задачи классификации, регрессии и кластеризации, а также соответствующие графики. Таким образом, вы сможете более детально изучить каждую задачу и ее решение.

Примечание: Для задачи классификации может возникнуть ошибка: «ConvergenceWarning: lbfgs failed to converge (status=1): STOP: TOTAL NO. of ITERATIONS REACHED LIMIT». Ошибка «ConvergenceWarning» возникает, когда модель логистической регрессии не сходится к оптимальному решению за ограниченное количество итераций. Это может быть вызвано различными причинами, такими как нарушение масштабирования данных, неправильный выбор параметров или недостаточное количество итераций.

Для решения этой проблемы можно увеличить максимальное количество итераций для алгоритма оптимизации. А также провести нормализацию данные перед обучением модели с помощью метода «StandardScaler», чтобы улучшить сходимость модели. Также, мы можем увеличить максимальное количество итераций для алгоритма оптимизации до 1000 с помощью «max_iter=1000». Это должно помочь избежать предупреждения «ConvergenceWarning» и успешно обучить модель.

Лекция №6 H20. Изучение возможностей метода AutoML для автоматического выбора модели и настройки

H2O.ai - это компания, которая разработала фреймворк H2O для машинного обучения и искусственного интеллекта. Компания H2O.ai была основана в 2011 году в Санта-Кларе, Калифорния. Основатели компании - Sri Ambati, Cliff Click и Arno Candel. H2O.ai стремится предоставить открытые и доступные инструменты для применения искусственного интеллекта в различных областях и помочь компаниям использовать данные для принятия более умных решений.

Кроме фреймворка H2O, у H2O.ai есть несколько других продуктов и решений, которые дополняют и расширяют возможности работы с данными и искусственным интеллектом:

1. H2O Driverless AI: Это интеллектуальный платформа автоматизации машинного обучения. Она предлагает автоматическую подготовку данных, оптимизацию параметров моделей, обучение различных моделей машинного обучения и создание полностью оптимизированных пайплайнов для решения задач машинного обучения.
2. H2O Q: Это интерактивный инструмент визуализации данных, который позволяет аналитикам и исследователям быстро визуализировать и исследовать данные. Он обеспечивает мощные функции визуализации и анализа данных, которые упрощают понимание данных и выявление важных закономерностей.
3. H2O MLOps: Это решение для управления жизненным циклом моделей машинного обучения. Оно помогает командам машинного обучения разрабатывать, развертывать и масштабировать модели в производственные системы с минимальными усилиями.

Все эти продукты и решения от H2O.ai обладают хорошей совместимостью и могут интегрироваться друг с другом для создания совокупных решений и обеспечения полного цикла разработки и внедрения моделей машинного обучения. Это упрощает процесс работы с данными, обучения моделей и реализации искусственного интеллекта в реальных приложениях. Благодаря этому компании и организациям становится легче и быстрее использовать машинное обучение для решения сложных задач и повышения эффективности бизнес-процессов.

Что такое H2O?

H2O - это фреймворк машинного обучения с открытым исходным кодом, разработанный для работы с большими объемами данных и высокой производительности. Он написан на языке Java, что позволяет использовать его на различных платформах. H2O предоставляет реализации множества алгоритмов машинного обучения, таких как градиентный бустинг, случайный лес, нейронные сети и многое другое.

Особенности и преимущества H2O:

- Распределенные вычисления: H2O обеспечивает возможность распределенных вычислений, что позволяет работать с большими объемами данных и эффективно использовать ресурсы кластера.
- Простой интерфейс: H2O предоставляет простой и интуитивно понятный интерфейс для создания, обучения и оценки моделей машинного обучения.

- Поддержка различных языков программирования: H2O предоставляет API для языков программирования, таких как Python, R, Java, Scala, что делает его доступным для широкого круга разработчиков.
- Автоматическая оптимизация: H2O поддерживает автоматическую оптимизацию параметров моделей, что позволяет находить наилучшие параметры без необходимости вручную настраивать модели.

Область применения

Фреймворк H2O широко применяется в различных сферах и областях, где требуется обработка данных и построение моделей машинного обучения. Вот некоторые из сфер использования H2O:

1. **Финансы**: В финансовой отрасли H2O может использоваться для прогнозирования рисков, кредитного скоринга, определения мошенничества, прогнозирования финансовых рынков и оптимизации портфелей.
2. **Здравоохранение**: В медицине H2O применяется для диагностики заболеваний на основе медицинских изображений, прогнозирования заболеваемости, применения персонализированной медицины и анализа медицинских данных.
3. **Розничная торговля**: В розничной торговле H2O может использоваться для персонализированных рекомендаций продуктов, прогнозирования спроса, анализа покупательского поведения и оптимизации цен.
4. **Производство**: В производственных компаниях H2O может применяться для предотвращения отказов оборудования, оптимизации производственных процессов, прогнозирования сбоев в работе и оптимизации качества продукции.
5. **Интернет вещей**: В сфере Интернета вещей H2O может применяться для анализа и обработки данных от датчиков, оптимизации работы устройств и предотвращения сбоев.
6. **Аналитика и бизнес-интеллект**: H2O используется для различных задач аналитики данных и бизнес-интеллекта, таких как прогнозирование продаж, сегментация клиентов, анализ текстовых данных и определение важных признаков.
7. **Энергетика**: В энергетической отрасли H2O может применяться для прогнозирования потребления энергии, оптимизации работы энергосистем и предотвращения сбоев в работе оборудования.
8. **Телекоммуникации**: H2O может использоваться для анализа данных о поведении клиентов, прогнозирования нагрузки на сети, оптимизации качества обслуживания и предотвращения отказов в сети.

Алгоритм работы с фреймворком H2O включает следующие основные шаги:

1. Установка и настройка H2O:

Установите H2O на свой компьютер или сервер. H2O поддерживает Python, R, Java и Scala, поэтому выберите подходящую версию для своей среды программирования.

Запустите H2O, инициализируя его с помощью соответствующего API для выбранного языка программирования.

2. Загрузка данных:

Загрузите данные, с которыми вы хотите работать, используя функции H2O для импорта данных из файлов или других источников данных.

3. Подготовка данных:

Проведите предварительный анализ данных, обработайте пропущенные значения и выполните другие необходимые преобразования для подготовки данных к обучению модели.

4. Разделение данных:

Разделите данные на обучающую и тестовую выборки. Это важный шаг для оценки производительности модели на новых данных.

5. Обучение модели:

Выберите алгоритм машинного обучения, который подходит для вашей задачи, и создайте модель с помощью H2O API.

Обучите модель на обучающих данных с использованием функций H2O для обучения моделей.

6. Оценка модели:

Оцените производительность модели на тестовой выборке с помощью метрик оценки качества, таких как точность, среднеквадратичная ошибка и другие.

7. Настройка модели (опционально):

Если необходимо, настройте параметры модели для достижения лучшей производительности. H2O предоставляет функции автоматической оптимизации параметров, которые можно использовать.

8. Прогнозирование (применение) модели:

Используйте обученную модель для прогнозирования на новых данных или в реальном времени.

9. Сохранение и загрузка модели:

При необходимости сохраните обученную модель на диск для последующего использования, либо загрузите сохраненную модель для прогнозирования.

10. Завершение работы с H2O:

Завершите работу с H2O, освободив ресурсы и память после завершения работы с моделями.

Введение в AutoML

AutoML (Automatic Machine Learning) - это функционал H2O, который позволяет автоматически настраивать и выбирать модели машинного обучения без участия человека. Это очень полезная функция, особенно когда у нас нет опыта в выборе оптимальных параметров моделей или, когда времени на настройку моделей ограничено.

Возможности AutoML в H2O:

- Автоматический выбор алгоритмов: AutoML позволяет автоматически выбирать оптимальные алгоритмы машинного обучения для решения задачи на основе данных и типа предсказания (классификация или регрессия).
- Автоматическая настройка параметров: AutoML выполняет автоматическую оптимизацию параметров моделей для достижения наилучшего качества предсказаний.
- Стекинг моделей: AutoML способен объединять несколько моделей в стек и использовать их вместе для улучшения качества предсказаний.
- Поддержка временных рядов: AutoML может работать с данными временных рядов, что делает его удобным инструментом для анализа и прогнозирования временных зависимостей.

Пример использования AutoML в H2O

Давайте рассмотрим простой пример использования AutoML в H2O для решения задачи классификации. Мы загрузим данные, разделим их на обучающую и тестовую выборки, а затем запустим AutoML для настройки моделей и выбора лучшей модели для нашей задачи.

```
# Пример использования AutoML в Python с H2O
import h2o
from h2o.automl import H2OAutoML

# Инициализация H2O и загрузка данных
h2o.init()
data = h2o.import_file("data.csv")

# Разделение данных на обучающую и тестовую выборки
train, test = data.split_frame(ratios=[0.8])

# Определение целевой переменной и признаков
x = data.columns[:-1]
y = 'target'

# Запуск AutoML
automl = H2OAutoML(max_models=10, seed=42)
automl.train(x=x, y=y, training_frame=train)

# Получение наилучшей модели
best_model = automl.leader
print("Наилучшая модель:", best_model)
```

H2O - это мощный фреймворк для машинного обучения, который обладает отличной поддержкой масштабируемости и предоставляет удобный интерфейс для работы с данными и моделями. AutoML в H2O делает процесс выбора и настройки моделей максимально

автоматизированным, что позволяет сосредоточиться на самой задаче и повысить производительность разработки искусственного интеллекта.

Лабораторная работа №6 Использование AutoML в H2O для автоматизации выбора моделей и подбора гиперпараметров.

Цель работы: Изучить и применить функционал AutoML в H2O для автоматического выбора и оптимизации моделей машинного обучения с использованием различных алгоритмов и гиперпараметров.

Задачи работы:

1. Ознакомление с фреймворком H2O и его возможностями для машинного обучения и искусственного интеллекта.
2. Подготовка данных: выбор и загрузка датасета для проведения экспериментов с AutoML.
3. Разделение данных на обучающую и тестовую выборки для оценки производительности моделей.
4. Изучение и настройка параметров AutoML в H2O для обеспечения оптимальной автоматизации выбора моделей и подбора гиперпараметров.
5. Запуск AutoML с использованием различных алгоритмов машинного обучения и оценка производительности полученных моделей.
6. Анализ результатов работы AutoML и выбор наилучшей модели для решения поставленной задачи.
7. Оценка и интерпретация результатов: сравнение метрик производительности моделей и анализ важности признаков, полученных с помощью AutoML.
8. Применение наилучшей модели для прогнозирования на новых данных и оценка ее точности.
9. Составление отчета о выполненной работе, включающего описание проведенных экспериментов, анализ результатов и выводы об эффективности AutoML в H2O для автоматизации выбора моделей и подбора гиперпараметров.

Примечание: В ходе работы студентам предлагается провести эксперименты с различными датасетами, настроить параметры AutoML, изучить метрики производительности моделей и сравнить результаты для выбора оптимальной модели для решения задачи машинного обучения. Лабораторная работа позволит студентам познакомиться с автоматизированным подходом к выбору моделей и оптимизации гиперпараметров, что упростит процесс работы с машинным обучением и повысит его эффективность.

Реализация задания на лабораторную работу

Создайте новую папку на рабочем столе и откройте ее через командную строку.

```
# Создание виртуальной среды с именем 'myenv'
python -m venv myenv

# Откройте редактор кода VS Code. Откройте новый терминал и
активируйте созданную на прошлом шаге виртуальную среду

# Активация виртуальной среды в Windows
myenv\Scripts\activate.bat
```

Установка необходимых библиотек

Для начала убедитесь, что у вас установлен Python, менеджер пакетов pip, а также Java JDK. Затем установите необходимые пакеты:

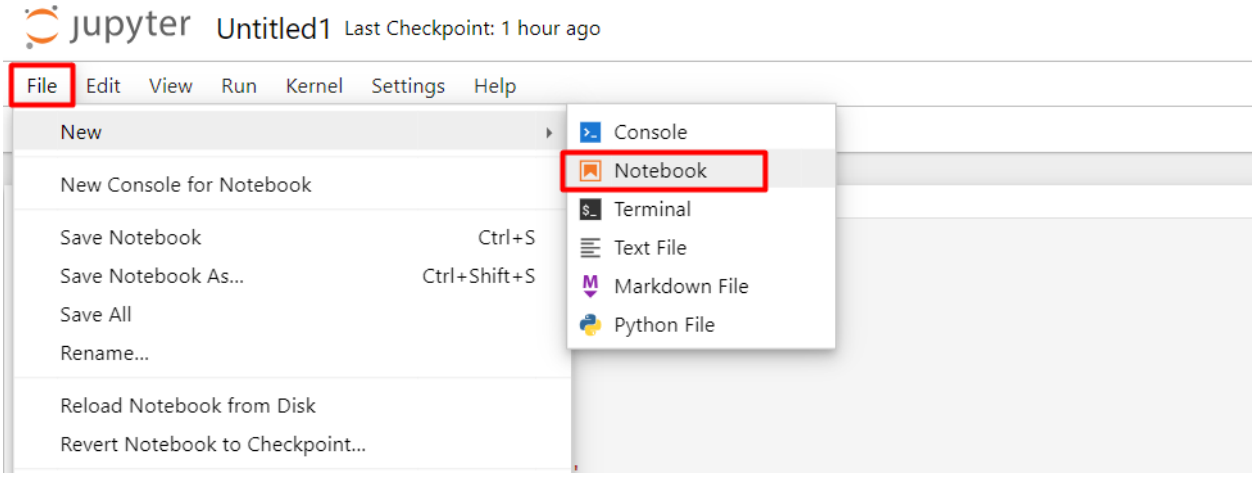
```
pip install h2o pandas numpy scikit-learn seaborn jupyter
```

Запуск Jupyter Notebook

Запустите Jupyter Notebook с помощью команды:

```
jupyter notebook
```

Jupyter Notebook откроется в вашем веб-браузере. Создайте новый ноутбук и начните кодирование примера.



Шаг 1: Установка и импорт необходимых пакетов

```
import h2o
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
```

Шаг 2: Инициализация H2O

```
h2o.init()
```

```
h2o.init()
```

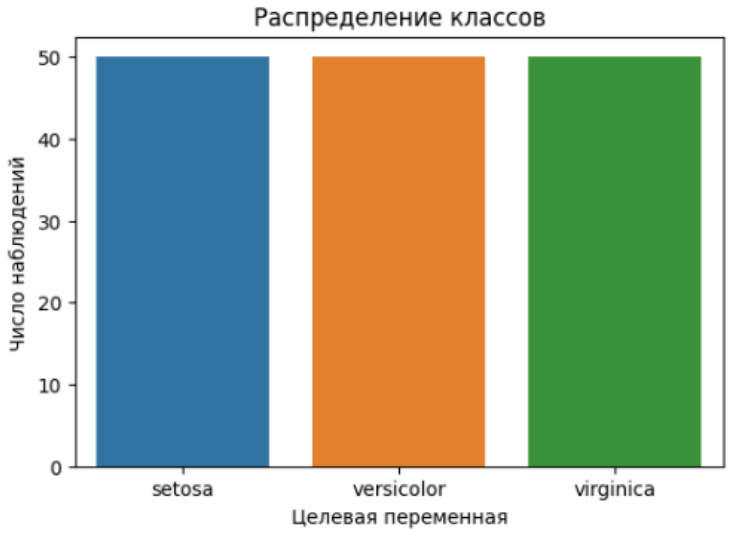
```
Checking whether there is an H2O instance running at http://localhost:54321. connected.
```

H2O_cluster_uptime:	17 mins 15 secs
H2O_cluster_timezone:	Asia/Yekaterinburg
H2O_data_parsing_timezone:	UTC
H2O_cluster_version:	3.42.0.2
H2O_cluster_version_age:	13 days
H2O_cluster_name:	H2O_from_python_MaxPC_4pqhed
H2O_cluster_total_nodes:	1
H2O_cluster_free_memory:	1.847 Gb
H2O_cluster_total_cores:	4
H2O_cluster_allowed_cores:	4
H2O_cluster_status:	locked, healthy
H2O_connection_url:	http://localhost:54321
H2O_connection_proxy:	{"http": null, "https": null}
H2O_internal_security:	False
Python_version:	3.10.3 final

Шаг 3: Загрузка и подготовка датасета

```
# Загрузка датасета "iris"
data = load_iris()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target_names[data.target]

# Преобразование pandas DataFrame в H2O Frame
h2o_df = h2o.H2OFrame(df)

# Визуализация первых 5 строк датасета
print("Первые 5 строк датасета:")
print(h2o_df.head())

# Визуализация распределения классов целевой переменной
plt.figure(figsize=(6, 4))
sns.countplot(x='target', data=df)
plt.title("Распределение классов")
plt.xlabel("Целевая переменная")
plt.ylabel("Число наблюдений")
plt.show()
```

```
Parse progress: |████████████████████████████████████████████████████████| (done) 100%
Первые 5 строк датасета:
  sepal length (cm)    sepal width (cm)    petal length (cm)    petal width (cm)  target
                5.1                 3.5                  1.4                0.2  setosa
                4.9                 3                    1.4                0.2  setosa
                4.7                 3.2                  1.3                0.2  setosa
                4.6                 3.1                  1.5                0.2  setosa
                5                   3.6                  1.4                0.2  setosa
                5.4                 3.9                  1.7                0.4  setosa
                4.6                 3.4                  1.4                0.3  setosa
                5                   3.4                  1.5                0.2  setosa
                4.4                 2.9                  1.4                0.2  setosa
                4.9                 3.1                  1.5                0.1  setosa
[10 rows x 5 columns]
```



Шаг 4: Разделение данных на обучающую и тестовую выборки

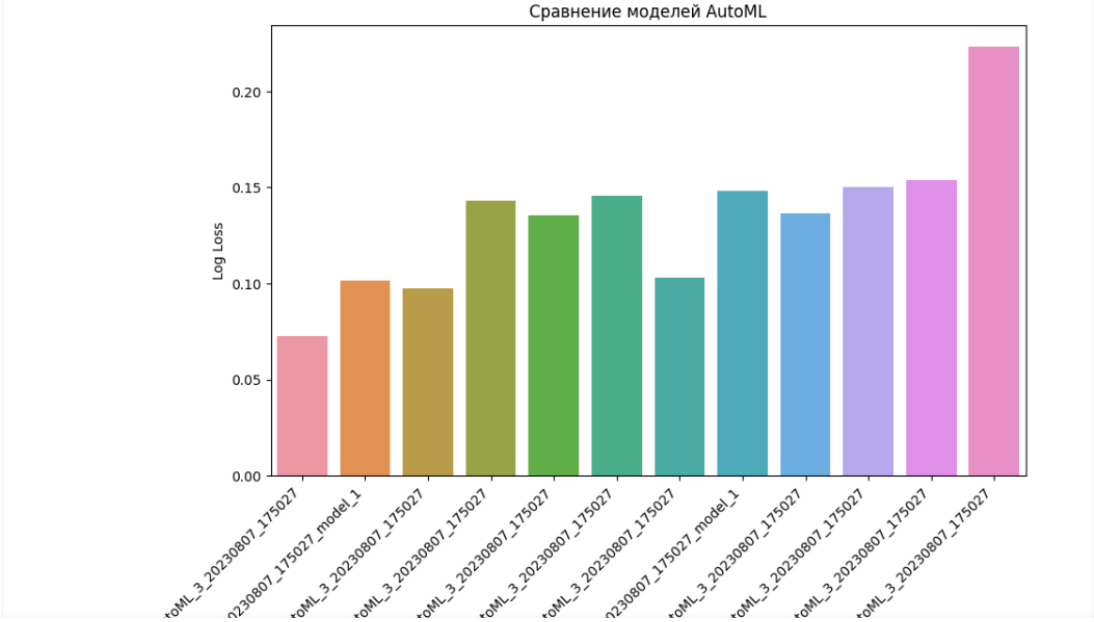
```
# Разделение данных на обучающую и тестовую выборки
train, test = h2o_df.split_frame(ratios=[0.8])

# Определение целевой переменной и признаков
x = h2o_df.columns[:-1]
y = 'target'
```

Шаг 5: Использование AutoML для выбора и настройки моделей

```
from h2o.automl import H2OAutoML

# Запуск AutoML с ограничением по времени выполнения и максимальному количеству моделей
automl = H2OAutoML(max_runtime_secs=3600, max_models=10, seed=42)
automl.train(x=x, y=y, training_frame=train)

# Получение наилучшей модели
best_model = automl.leader
print("Наилучшая модель:", best_model)
```

```
# Визуализация результатов AutoML
automl_leaderboard = automl.leaderboard.as_data_frame()
plt.figure(figsize=(10, 6))
sns.barplot(x='model_id', y='logloss', data=automl_leaderboard)
plt.xticks(rotation=45, ha='right')
plt.title("Сравнение моделей AutoML")
plt.xlabel("Модель")
plt.ylabel("Log Loss")
plt.show()
```

```
Variable Importances:
variable           relative_importance    scaled_importance    percentage
-----------------  ---------------------  -------------------  ------------
petal length (cm)  12.2422                1                    0.393435
petal width (cm)   11.4613                0.936214             0.368339
sepal width (cm)   3.96443                0.323833             0.127407
sepal length (cm)  3.44826                0.28167              0.110819
```



Шаг 6: Оценка производительности модели на тестовой выборке

```
# Оценка производительности на тестовой выборке
perf = best_model.model_performance(test_data=test)
print(perf)
```

```
ModelMetricsMultinomialGLM: glm
** Reported on test data. **

MSE: 0.0229700774098234 62
RMSE: 0.15155882491568567
LogLoss: 0.06243427093155967
Null degrees of freedom: 32
Residual degrees of freedom: 18
Null deviance: 73.34306313692458
Residual deviance: 4.120661881482938
AUC table was not computed: it is either disabled (model parameter 'auc_type' was set to AUTO or NONE) or the domain size exceeds the limit (maximum is
50 domains).
AUCPR table was not computed: it is either disabled (model parameter 'auc_type' was set to AUTO or NONE) or the domain size exceeds the limit (maximum i
s 50 domains).

Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
setosa    versicolor    virginica    Error      Rate
--------  ------------  -----------  ---------  ------
10        0             0            0          0 / 10
0         9             0            0          0 / 9
0         1             13           0.0714286  1 / 14
10        10            13           0.030303   1 / 33
```

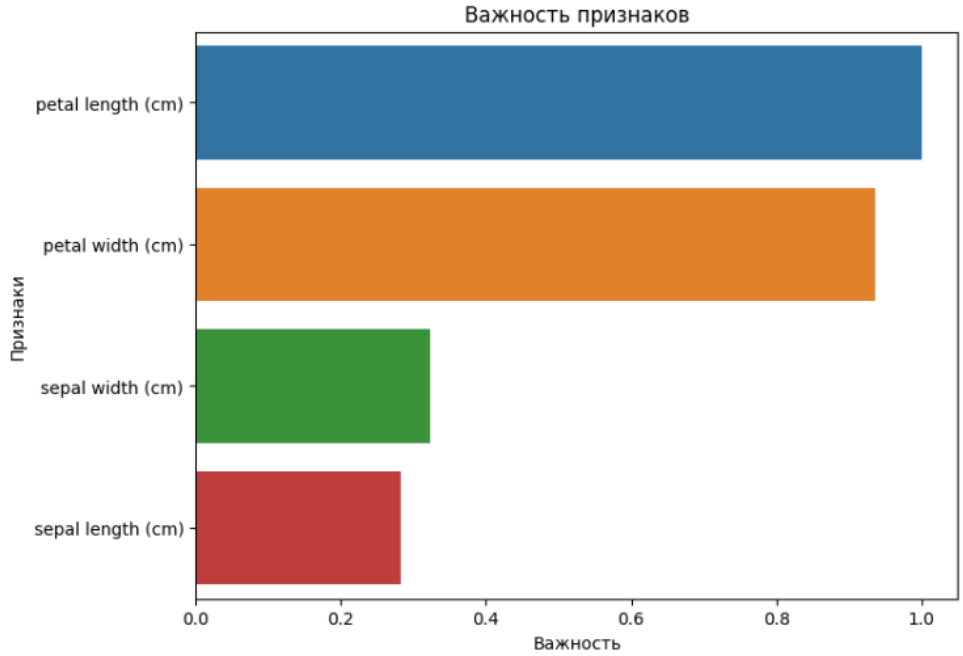
…

```
# Визуализация оценки производительности модели
plt.figure(figsize=(6, 4))
sns.barplot(x='model_id', y='logloss', data=perf)
plt.title("Производительность модели на тестовой выборке")
plt.xlabel("Модель")
plt.ylabel("Log Loss")
plt.show()
```

Шаг 7: Анализ результатов и выводы

```
# Анализ важности признаков
feature_importance = best_model.varimp(use_pandas=True)
print("Важность признаков:")
print(feature_importance)

# Визуализация важности признаков
plt.figure(figsize=(8, 6))
sns.barplot(x='scaled_importance', y='variable',
data=feature_importance)
plt.title("Важность признаков")
plt.xlabel("Важность")
plt.ylabel("Признаки")
plt.show()
```

```
Важность признаков:
             variable  relative_importance  scaled_importance  percentage
0  petal length (cm)            12.242188           1.000000    0.393435
1   petal width (cm)            11.461313           0.936214    0.368339
2   sepal width (cm)             3.964426           0.323833    0.127407
3  sepal length (cm)             3.448259           0.281670    0.110819
```



В текущей лабораторной работе используется базовый датасет "iris". Обратите внимание, что использование встроенного датасета из scikit-learn позволяет нам пропустить шаги по загрузке и предобработке данных, так как они уже содержатся в датасете.

Код может отличаться в зависимости от формата данных, используемых метрик оценки качества и других параметров AutoML. Также, возможно, вам понадобится провести дополнительные настройки и анализ результатов в соответствии с вашим конкретным датасетом и задачей машинного обучения.

Лекция 7: Microsoft Cognitive Toolkit (CNTK)

Microsoft Cognitive Toolkit (CNTK) - один из популярных фреймворков для разработки и обучения глубоких нейронных сетей. В этой лекции мы рассмотрим основные аспекты CNTK, его сравнение с другими фреймворками, а также рассмотрим его особенности, слабые и сильные стороны, и алгоритм использования.

1. Введение в Microsoft Cognitive Toolkit (CNTK)

1.1 Что такое CNTK?

Microsoft Cognitive Toolkit, или сокращенно CNTK, это глубокий фреймворк с открытым исходным кодом, разработанный Microsoft для разработки и обучения глубоких нейронных сетей. Он был создан с упором на эффективность и масштабируемость, позволяя обрабатывать большие объемы данных и ускорять процесс обучения на многих процессорах и графических ускорителях (GPU).

1.2 Особенности CNTK.

CNTK обладает рядом особенностей, которые делают его привлекательным для разработчиков и исследователей:

- Эффективность: CNTK оптимизирован для обучения и выполнения глубоких нейронных сетей на больших объемах данных.
- Масштабируемость: Фреймворк может работать на множестве устройств, включая CPU и GPU, что позволяет ускорить процесс обучения.
- Гибкость: CNTK предоставляет широкий набор функций для создания и обучения различных архитектур нейронных сетей.
- Поддержка различных языков программирования: CNTK поддерживает Python и C++, что облегчает разработку и интеграцию с существующими проектами.
- Визуализация: CNTK предоставляет средства для визуализации графов нейронных сетей, что помогает понять структуру модели и ее производительность.

2. Внедрение глубоких нейронных сетей с использованием CNTK

2.1 Создание нейронной сети с помощью CNTK. Для создания нейронной сети с использованием CNTK, необходимо импортировать библиотеку в выбранном языке программирования (Python или C++). Затем определить архитектуру сети, задать функцию потерь и оптимизатор, и запустить процесс обучения на тренировочных данных.

2.2 Обучение нейронной сети. Обучение нейронной сети в CNTK включает подачу обучающих данных в модель, вычисление функции потерь и оптимизацию параметров модели с помощью заданного оптимизатора. Процесс обучения может быть многократно повторен (эпохи), чтобы улучшить производительность модели.

2.3 Применение обученной модели. После завершения процесса обучения, модель может быть использована для предсказания на новых данных. Это может быть важной частью внедрения модели в реальные приложения.

3. Сравнение CNTK с другими фреймворками

3.1 Сравнение с TensorFlow и PyTorch. CNTK, TensorFlow и PyTorch являются популярными фреймворками для глубокого обучения. В сравнении с TensorFlow, CNTK обладает более высокой производительностью на многих задачах, особенно при использовании больших объемов данных. PyTorch и CNTK также имеют схожую производительность, но выбор между ними может зависеть от предпочтений программиста и характера проекта.

3.2 Сравнение с Keras. Keras - это высокоуровневый API для создания нейронных сетей, который может быть построен поверх различных бекендов, включая TensorFlow и CNTK. В этом смысле CNTK и Keras могут использоваться вместе, сочетая преимущества CNTK в производительности с удобством использования Keras.

4. Сильные и слабые стороны CNTK

4.1 Сильные стороны CNTK:

- Высокая производительность на многих задачах, особенно с использованием GPU.
- Эффективность и масштабируемость на больших объемах данных.
- Гибкость в выборе архитектуры нейронных сетей и оптимизаторов.
- Поддержка Python и C++.

4.2 Слабые стороны CNTK:

- Относительно меньшее количество документации и сообщества, по сравнению с TensorFlow и PyTorch.
- Ограниченные инструменты для работы с некоторыми сложными архитектурами нейронных сетей, которые могут быть доступны в других фреймворках.

5. Алгоритм использования CNTK

5.1 Выбор задачи и данных. Первый шаг - определить задачу, которую вы хотите решить с помощью нейронной сети. Затем подготовьте данные для обучения и тестирования модели.

5.2 Создание и обучение модели. Определите архитектуру нейронной сети с помощью CNTK и укажите функцию потерь и оптимизатор. Обучите модель на обучающих данных и отслеживайте производительность на тестовых данных.

5.3 Оценка и настройка. Оцените производительность модели на тестовых данных. Если необходимо, настройте параметры модели и архитектуру, чтобы улучшить результаты.

5.4 Применение модели. После завершения обучения и настройки, модель может быть использована для предсказания на новых данных в реальных приложениях.

Microsoft Cognitive Toolkit (CNTK) - мощный фреймворк для разработки и обучения глубоких нейронных сетей. Он обладает высокой производительностью и масштабируемостью, что делает его привлекательным для различных задач и проектов. Хотя CNTK имеет некоторые слабые стороны, его гибкость и эффективность позволяют разработчикам создавать мощные и точные модели глубокого обучения.

Лабораторная работа №7. Разработка нейронной сети с рекуррентной архитектурой

Цель лабораторной работы: Целью данной лабораторной работы является разработка и обучение нейронной сети с рекуррентной архитектурой для решения задачи временного ряда или последовательности данных.

Задачи:

1. Изучение принципов работы рекуррентных нейронных сетей (RNN) и их применение в задачах анализа временных рядов и последовательностей.
2. Ознакомление с различными типами рекуррентных ячеек, такими как LSTM (Long Short-Term Memory) и GRU (Gated Recurrent Unit).
3. Подготовка обучающего набора данных для обучения и тестирования нейронной сети. Можно использовать как собственные данные, так и открытые наборы данных, подходящие для выбранной задачи.
4. Реализация рекуррентной нейронной сети с использованием выбранного фреймворка (например, TensorFlow, PyTorch, CNTK или Keras).
5. Настройка гиперпараметров модели, таких как количество слоев, количество нейронов, скорость обучения и функции активации, для достижения оптимальной производительности.
6. Обучение разработанной модели на обучающих данных и оценка ее производительности на тестовых данных.
7. Анализ результатов и сравнение с базовыми моделями или другими типами нейронных сетей (по желанию).
8. Визуализация прогнозов или выходов модели для наглядного представления результатов.

Инструменты: Для выполнения лабораторной работы можно использовать один из популярных фреймворков для глубокого обучения, таких как TensorFlow, PyTorch, CNTK или Keras. Также необходимо использовать язык программирования, с которым вы знакомы или хотите познакомиться (Python, C++, и т.д.).

Результаты: Ожидается, что студенты представят отчет о выполненной лабораторной работе, включающий:

1. Описание выбранной задачи и набора данных.
2. Подход к разработке архитектуры нейронной сети.
3. Использованные гиперпараметры и аргументы для обучения модели.
4. Анализ результатов обучения и производительности разработанной нейронной сети.
5. Выводы о применимости и эффективности рекуррентных нейронных сетей в данной задаче.

Примечание: При выполнении лабораторной работы студенты могут обратиться за помощью к преподавателю или советам из открытых источников, но оригинальность и самостоятельность в подходе к разработке модели будут оцениваться выше.

Далее, рассмотрим пример выполнения лабораторной работы по разработке рекуррентной нейронной сети с использованием фреймворка CNTK для решения задачи прогнозирования временного ряда. В этом примере мы будем использовать данные о ежедневных температурах для прогнозирования температуры на следующий день.

Для установки CNTK, вы можете использовать pip (если у вас установлен Python) или conda (если вы используете среду Anaconda).

С помощью pip:

```
pip install cntk
```

С помощью conda:

```
conda install cntk -c pytorch
```

Установка дополнительных модулей:

В приведенном выше примере, мы также используем библиотеки NumPy, Pandas и Matplotlib для работы с данными и визуализации. Если вы еще не установили их, вам нужно установить их также:

С помощью pip:

```
pip install numpy pandas matplotlib
```

С помощью conda:

```
conda install numpy pandas matplotlib
```

Шаг 1: Импорт необходимых библиотек

```
import cntk as C
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Шаг 2: Подготовка данных

Для этого примера используем искусственно сгенерированные данные, представляющие среднесуточные температуры в течение 365 дней.

```
np.random.seed(42)
num_days = 365
temps = np.random.randint(-10, 30, size=num_days)  # Сгенерируем
случайные температуры
```

Шаг 3: Создание обучающего и тестового наборов данных

Для простоты разделим данные на обучающий и тестовый наборы в пропорции 80/20.

```
train_data = temps[:int(num_days * 0.8)]
test_data = temps[int(num_days * 0.8):]
```

Шаг 4: Подготовка данных для обучения

Мы хотим преобразовать последовательность температур в пары вход-выход, чтобы обучить рекуррентную нейронную сеть предсказывать температуру на следующий день.

```
def create_sequences(data, seq_length):
    X, Y = [], []
    for i in range(len(data) - seq_length):
        X.append(data[i:i + seq_length])
        Y.append(data[i + seq_length])
    return np.array(X), np.array(Y)

sequence_length = 10  # Длина последовательности
train_X, train_Y = create_sequences(train_data, sequence_length)
test_X, test_Y = create_sequences(test_data, sequence_length)
```

Шаг 5: Создание модели рекуррентной нейронной сети

В этом примере мы будем использовать простую рекуррентную нейронную сеть с одним слоем LSTM.

```
input_dim = 1  # Размер входа (температуры)
hidden_dim = 10  # Размер скрытого слоя
output_dim = 1  # Размер выхода (прогнозируемая температура на
следующий день)

# Определение входного тензора и целевого тензора
input_var = C.sequence.input_variable(input_dim)
target_var = C.sequence.input_variable(output_dim)

# Создание LSTM-ячейки
lstm = C.layers.LSTM(hidden_dim)

# Построение архитектуры нейронной сети
output = C.layers.Dense(output_dim)(lstm(input_var))

# Определение функции потерь и метрики
loss = C.squared_error(output, target_var)
metric = C.squared_error(output, target_var)
```

Шаг 6: Обучение модели

```
# Создание функций обучения и тестирования
learning_rate = 0.01
trainer = C.Trainer(output, (loss, metric), [C.sgd(output.parameters,
lr=learning_rate)])

# Обучение модели
```

```
num_epochs = 50
batch_size = 32

for epoch in range(num_epochs):
    epoch_loss = 0
    num_batches = len(train_X) // batch_size
    for batch in range(num_batches):
        start_idx = batch * batch_size
        end_idx = (batch + 1) * batch_size
        x_batch = train_X[start_idx:end_idx]
        y_batch = train_Y[start_idx:end_idx]
        trainer.train_minibatch({input_var: x_batch, target_var:
y_batch})
        epoch_loss += trainer.previous_minibatch_loss_average

    epoch_loss /= num_batches
    print(f"Epoch {epoch + 1}/{num_epochs}, Loss: {epoch_loss:.4f}")

# Тестирование модели
test_metric = C.squared_error(output, target_var)
test_metric.eval({input_var: test_X, target_var: test_Y})
```

Шаг 7: Визуализация результатов

```
# Визуализация прогнозов и реальных значений
plt.plot(test_Y, label='True')
plt.plot(predictions, label='Predicted')
plt.xlabel('Time')
plt.ylabel('Temperature')
plt.legend()
plt.show()
```

В этом примере мы рассмотрели основные шаги разработки рекуррентной нейронной сети с помощью фреймворка CNTK для прогнозирования температуры на следующий день на основе искусственно сгенерированных данных. Вы можете адаптировать этот пример для своей задачи и данных, изменяя архитектуру сети и параметры обучения.

Лекция №8. Трансферное обучение и обучение с подкреплением

Трансферное обучение (Transfer Learning) - это метод машинного обучения, который позволяет использовать знания, полученные в процессе решения одной задачи, для улучшения производительности в другой задаче. Основная идея заключается в том, чтобы передать (трансферировать) знания из одной модели (обычно обученной на большом наборе данных) в другую модель (модель с меньшим объемом данных) с целью улучшения её обучения и общей производительности.

Процесс трансферного обучения обычно происходит следующим образом:

1. Подготовка предобученной модели: Первоначально обучается модель на большом наборе данных и задаче (например, распознавание объектов на ImageNet). Эта модель называется "предобученной моделью".
2. Извлечение признаков: После обучения предобученной модели, её сверточные слои (или часть сети) можно использовать для извлечения высокоуровневых признаков из входных данных.
3. Приспособление модели к новой задаче: Затем предобученную модель "замораживают", чтобы сохранить веса полученных признаков, и добавляют новые слои (обычно полносвязные слои) для решения новой задачи.
4. Обучение на новых данных: Затем "размораживают" несколько верхних слоёв предобученной модели и переобучают эти слои на новом наборе данных (набор данных новой задачи).

Преимущества трансферного обучения:

- Улучшение производительности: За счет использования знаний из предобученной модели, обучение на новых данных может быть более эффективным и быстрым, чем начинать обучение с нуля.
- Обучение на небольших данных: Трансферное обучение особенно полезно, когда у нас есть ограниченное количество данных для новой задачи.
- Предотвращение переобучения: Предобученные модели уже обучены на огромном наборе данных, что помогает предотвратить переобучение на новых данных.

Обучение с подкреплением (Reinforcement Learning) - это раздел машинного обучения, в котором агент (например, робот или программа) взаимодействует с окружающей средой и выполняет действия, чтобы максимизировать некоторую награду или кумулятивную награду на протяжении времени. В этом процессе агент осуществляет действия в среде и получает обратную связь в виде награды или штрафа от среды в зависимости от его действий.

Основные компоненты обучения с подкреплением:

1. Агент: Это сущность, которая принимает решения и выполняет действия в среде.
2. Среда: Это мир, с которым агент взаимодействует и где он выполняет свои действия.
3. Действия (Actions): В каждый момент времени агент выполняет определенное действие из пространства возможных действий.
4. Награды (Rewards): Агент получает награду или штраф от среды в зависимости от его действий. Цель агента - максимизировать награду на протяжении времени.

5. Политика (Policy): Это стратегия, которая определяет, как агент выбирает действия в зависимости от состояния среды.

Процесс обучения с подкреплением проходит следующим образом:

1. Агент взаимодействует с средой, выполняет действия и получает награды.
2. Агент использует полученные награды для обновления своей политики таким образом, чтобы максимизировать ожидаемую награду.
3. Обучение происходит на основе методов оптимизации, таких как Q-learning, Deep Q-Networks (DQN), Policy Gradient и других.

Обучение с подкреплением широко применяется в задачах, где не существует явных правил для достижения определенной цели, например, в играх, управлении роботами, финансовой торговле и других сферах, где агент должен учиться оптимальным стратегиям взаимодействия с окружающей средой.

Некоторые фреймворки машинного обучения и искусственного интеллекта предоставляют возможности для реализации методов трансферного обучения и обучения с подкреплением. Ниже приведены некоторые из наиболее популярных фреймворков, которые поддерживают эти методы:

1. TensorFlow: TensorFlow - это один из наиболее популярных и мощных фреймворков для машинного обучения и искусственного интеллекта. Он предоставляет богатые возможности для трансферного обучения и обучения с подкреплением, включая предобученные модели (например, модели из библиотеки TensorFlow Hub), инструменты для реализации различных алгоритмов обучения с подкреплением и поддержку создания пользовательских моделей и агентов.
2. PyTorch: PyTorch - это еще один популярный и гибкий фреймворк для глубокого обучения. Он также обладает хорошей поддержкой для трансферного обучения, включая возможность использования предобученных моделей из библиотеки TorchVision. Кроме того, PyTorch предоставляет инструменты для обучения с подкреплением, такие как библиотеку PyTorch RL.
3. Keras: Keras - это простой и удобный фреймворк для создания нейронных сетей, который построен поверх TensorFlow. Keras предоставляет набор предобученных моделей из библиотеки Keras Applications, что делает трансферное обучение более простым. Он также поддерживает создание моделей обучения с подкреплением.
4. OpenAI Gym: OpenAI Gym - это библиотека, предоставляющая среды и задачи для обучения с подкреплением. Она предоставляет набор сред, в которых можно обучать агентов и оценивать их производительность. Это один из наиболее популярных фреймворков для исследования и реализации обучения с подкреплением.
5. Stable Baselines: Stable Baselines - это библиотека, основанная на TensorFlow, предназначенная специально для обучения с подкреплением. Она предоставляет реализации популярных алгоритмов обучения с подкреплением, таких как DDPG, PPO, A2C и другие.
6. Microsoft CNTK (Cognitive Toolkit): CNTK - это фреймворк, разработанный Microsoft для обучения глубоких нейронных сетей и поддерживает трансферное обучение и обучение с подкреплением.

7. Apache MXNet: MXNet - это другой гибкий фреймворк для машинного обучения и глубокого обучения, который поддерживает методы трансферного обучения и обучения с подкреплением.

Давайте рассмотрим некоторые из предоставляемых библиотек более подробно.

PyTorch RL:

Описание: PyTorch RL - это библиотека, которая предоставляет инструменты и алгоритмы для обучения агентов с использованием обучения с подкреплением на базе фреймворка PyTorch.

Возможности: Библиотека содержит различные реализации алгоритмов обучения с подкреплением, таких как DDPG (Deep Deterministic Policy Gradients), PPO (Proximal Policy Optimization), A2C (Advantage Actor-Critic) и других. Она также предоставляет удобный API для создания собственных агентов и сред обучения с подкреплением.

Особенности работы: PyTorch RL построена поверх фреймворка PyTorch, что делает её удобной для исследования и реализации новых алгоритмов. Библиотека обеспечивает простоту и гибкость для создания и обучения агентов с подкреплением.

Плюсы:

- Простота и гибкость создания собственных агентов и сред обучения.
- Поддержка различных алгоритмов обучения с подкреплением.
- Интеграция с фреймворком PyTorch, что позволяет использовать всю его мощь и возможности.

***Минусы*:**

- Некоторые алгоритмы могут быть менее оптимизированы и масштабируемы по сравнению с другими библиотеками.

Пример использования: Обучение агента для игры в Atari с использованием алгоритма PPO и PyTorch RL.

Keras Applications:

Описание: Keras Applications - это часть фреймворка Keras, которая предоставляет набор предобученных моделей для классификации изображений. Он включает известные архитектуры, такие как VGG, ResNet, Inception и другие.

Возможности: Библиотека предоставляет доступ к предобученным моделям, обученным на больших наборах данных, таких как ImageNet. Они могут быть использованы для классификации, извлечения признаков или дальнейшего дообучения на других задачах.

Особенности работы: Keras Applications предоставляет удобные интерфейсы для загрузки предобученных моделей и использования их в своих проектах. Модели можно обращаться как с полной архитектурой, так и с возможностью загрузки весов из ImageNet.

Плюсы:

- Простота использования предобученных моделей для классификации и извлечения признаков.

- Возможность дообучения моделей на своих данных для более специфичных задач.

Минусы:

- Ограничение выбора предобученных моделей только на классификацию изображений.

Пример использования: Использование предобученной модели VGG для классификации изображений с помощью Keras Applications.

OpenAI Gym:

Описание: OpenAI Gym - это библиотека, разработанная OpenAI, которая предоставляет среды и задачи для обучения с подкреплением.

Возможности: Библиотека включает множество сред для обучения агентов с подкреплением, включая задачи контроля, игры и другие сценарии. Она также предоставляет удобный API для взаимодействия с средами и обучения агентов.

Особенности работы: OpenAI Gym предоставляет интерфейс для создания среды, выполнения действий агента и получения награды. Библиотека позволяет оценивать производительность агентов и сравнивать различные алгоритмы обучения с подкреплением.

Плюсы:

- Большой выбор сред и задач для обучения с подкреплением.
- Удобный API для взаимодействия с средами и обучения агентов.
- Широко используется в исследованиях обучения с подкреплением.

Минусы:

- Ограниченные возможности для создания собственных сред и задач.

Пример использования: Обучение агента с помощью алгоритма DQN для игры в Atari с использованием OpenAI Gym.

Stable Baselines:

Описание: Stable Baselines - это библиотека, предназначенная для обучения агентов с подкреплением. Она предоставляет реализации популярных алгоритмов, таких как DDPG, PPO, A2C и других.

Возможности: Библиотека предоставляет готовые реализации алгоритмов обучения с подкреплением, что делает процесс обучения более простым. Она также поддерживает параллельное обучение и оптимизацию для повышения эффективности.

Особенности работы: Stable Baselines предоставляет простой API для создания агентов, обучения и взаимодействия с средами. Она строится на фреймворке TensorFlow и обеспечивает стабильное обучение агентов с подкреплением.

Плюсы:

- Готовые реализации популярных алгоритмов обучения с подкреплением.
- Поддержка параллельного обучения для ускорения процесса.

Минусы:

- Может быть менее гибкой в сравнении с другими библиотеками для создания собственных агентов.

Пример использования: Обучение агента с использованием алгоритма PPO с помощью Stable Baselines.

MXNet:

Описание: Apache MXNet (произносится "mix-net") - это гибкий фреймворк глубокого обучения, разработанный Apache Software Foundation. Он предоставляет высокую производительность и эффективность для обучения нейронных сетей и поддерживает различные алгоритмы обучения.

Возможности: MXNet поддерживает широкий спектр функциональности для глубокого обучения, включая обучение с подкреплением и трансферное обучение. Он имеет API для создания различных типов моделей и реализации различных алгоритмов оптимизации.

Особенности работы: MXNet может работать на различных платформах и поддерживает распределенное обучение для больших моделей и наборов данных. Он предоставляет высокую производительность благодаря оптимизации и использованию GPU.

Плюсы:

- Высокая производительность и эффективность.
- Поддержка различных алгоритмов обучения и типов моделей.

Минусы:

- Менее популярен и менее развит в сообществе по сравнению с TensorFlow и PyTorch.

Пример использования: Обучение агента с помощью алгоритма DDPG с использованием фреймворка MXNet.

Лабораторная работа №8 Применение методов оптимизации моделей искусственного интеллекта

Цель работы: Изучить и применить различные методы оптимизации для улучшения производительности и точности моделей искусственного интеллекта.

Задачи:

1. Изучить основные понятия и теоретические аспекты оптимизации моделей искусственного интеллекта, включая градиентные методы, стохастические методы и эволюционные алгоритмы.
2. Провести анализ и выбрать подходящую задачу для оптимизации модели искусственного интеллекта. Это может быть задача классификации, регрессии, кластеризации или другие.
3. Реализовать выбранную модель искусственного интеллекта с использованием одной из популярных библиотек машинного обучения, таких как TensorFlow, PyTorch, или scikit-learn.
4. Применить несколько методов оптимизации для обучения модели. Включить варианты градиентного спуска, стохастического градиентного спуска, методы оптимизации с моментом, адаптивные методы оптимизации (например, Adam, RMSprop), а также методы глобальной оптимизации, такие как генетические алгоритмы.
5. Сравнить производительность модели с различными методами оптимизации в терминах скорости сходимости, точности и устойчивости к переобучению.
6. Провести анализ результатов и сделать выводы о применимости каждого метода оптимизации в зависимости от типа задачи и структуры модели.
7. Оформить отчет о лабораторной работе, включающий в себя введение, описание методологии, полученные результаты, обсуждение и заключение.

Примечание: При выполнении лабораторной работы можно использовать различные наборы данных и изменять параметры модели, чтобы провести более глубокий анализ эффективности методов оптимизации.

Пример использования методов оптимизации на задачах кластеризации и регрессии

Пример применения методов оптимизации моделей искусственного интеллекта на задаче кластеризации с использованием метода оптимизации «K-Means». В данном примере мы будем использовать библиотеку «Scikit-learn» для кластеризации данных с помощью алгоритма «K-Means».

Пример применения метода кластеризации «K-Means»:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

# Генерируем синтетические данные для кластеризации
n_samples = 300
```

```
n_features = 2
n_clusters = 3
X, _ = make_blobs(n_samples=n_samples, n_features=n_features,
centers=n_clusters, random_state=42)

# Инициализируем модель K-Means с использованием метода оптимизации
Lloyd's (по умолчанию)
kmeans = KMeans(n_clusters=n_clusters, random_state=42)

# Обучаем модель на данных
kmeans.fit(X)

# Получаем прогнозированные метки кластеров для каждого образца
labels = kmeans.labels_

# Визуализируем результаты кластеризации
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,
1], c='red', marker='x', s=200)
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('K-Means Clustering')
plt.show()

# Вычисляем оценку силуэта для кластеризации
silhouette_avg = silhouette_score(X, labels)
print(f"Average Silhouette Score: {silhouette_avg:.4f}")
```

В этом примере мы генерируем синтетические данные для задачи кластеризации с помощью «make_blobs», инициализируем модель «K-Means» с помощью класса «KMeans» из «Scikit-learn» и обучаем её на данных. Затем мы получаем прогнозированные метки кластеров для каждого образца и визуализируем результаты кластеризации.

Мы также вычисляем оценку силуэта для оценки качества кластеризации. Оценка силуэта предоставляет информацию о том, насколько хорошо объекты внутри кластеров похожи друг на друга и насколько различны они от объектов в других кластерах.

Убедитесь, что у вас установлены библиотеки «numpy», «matplotlib», «scikit-learn» перед запуском кода.

Пример применения методов оптимизации моделей искусственного интеллекта на задаче регрессии. В данном примере мы будем использовать библиотеку Scikit-learn для создания и обучения модели регрессии с использованием метода оптимизации «SGD» (Stochastic Gradient Descent).

Пример применения метода оптимизации «SGD» для задачи регрессии:

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import SGDRegressor
from sklearn.metrics import mean_squared_error

# Генерируем синтетические данные для задачи регрессии
np.random.seed(0)
X = np.random.rand(100, 1)
y = 2 * X + 1 + 0.1 * np.random.randn(100, 1)

# Разделяем данные на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Масштабируем данные
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Инициализируем модель регрессии с использованием метода SGD
regressor = SGDRegressor(learning_rate='constant', eta0=0.01,
max_iter=1000, random_state=42)

# Обучаем модель
regressor.fit(X_train_scaled, y_train.ravel())

# Предсказываем значения на тестовой выборке
y_pred = regressor.predict(X_test_scaled)

# Вычисляем среднеквадратичную ошибку
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse:.4f}")
```

В этом примере мы используем синтетические данные для задачи регрессии, создаем модель регрессии с помощью «SGDRegressor» из «Scikit-learn» и обучаем её на обучающей выборке. Затем мы предсказываем значения на тестовой выборке и вычисляем среднеквадратичную ошибку в качестве метрики производительности.

Обратите внимание, что параметры «learning_rate», «eta0» и «max_iter» настроены на значения, которые могут потребовать дополнительной настройки в зависимости от данных и задачи.

Важно помнить, что выбор метода оптимизации и его параметров может зависеть от специфики данных, задачи и типа модели, поэтому рекомендуется провести эксперименты с различными параметрами для достижения наилучших результатов.

Пример использования предобученной модели «VGG» для классификации изображений с помощью библиотеки «Keras Applications»:

```
import numpy as np
from keras.applications.vgg16 import VGG16, preprocess_input,
decode_predictions
from keras.preprocessing import image

# Загружаем предобученную модель VGG16
model = VGG16(weights='imagenet')

# Загружаем изображение для классификации
img_path = 'path_to_your_image.jpg'  # Укажите путь к изображению
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

# Получаем предсказания с помощью модели
predictions = model.predict(x)

# Декодируем предсказания в текстовый формат с помощью
decode_predictions
decoded_predictions = decode_predictions(predictions, top=3)[0]

# Выводим результаты классификации
for label, description, score in decoded_predictions:
    print(f"{description} ({label}): {score:.2f}")
```

В этом примере мы используем предобученную модель «VGG16» из библиотеки «Keras Applications» для классификации изображения. Мы загружаем изображение, предварительно обрабатываем его для соответствия входным данным модели, и получаем предсказания классов с их вероятностями. Затем мы используем функцию «decode_predictions» для перевода числовых меток классов в человекочитаемые описания и выводим результаты.

Убедитесь, что у вас установлены библиотеки «Keras» и «TensorFlow» (или TensorFlow backend) перед запуском кода. Вы также должны заменить 'path_to_your_image.jpg' на путь к изображению, которое вы хотите классифицировать.

Пример реализации алгоритма обучения с подкреплением с использованием библиотеки «OpenAI Gym» и алгоритма «Q-learning». В этом примере мы будем использовать среду "FrozenLake-v0" из библиотеки Gym и реализуем алгоритм «Q-learning» для обучения агента.

Пример реализации «Q-learning» с обучением с подкреплением:

```
import numpy as np
import gym

# Создаем среду FrozenLake
env = gym.make('FrozenLake-v0')

# Определяем параметры обучения
num_episodes = 10000
learning_rate = 0.1
discount_factor = 0.99
exploration_prob = 1.0
exploration_decay = 0.995
min_exploration_prob = 0.01

# Инициализируем Q-таблицу
num_states = env.observation_space.n
num_actions = env.action_space.n
Q = np.zeros((num_states, num_actions))

# Обучение с подкреплением с использованием алгоритма Q-learning
for episode in range(num_episodes):
    state = env.reset()
    done = False

    while not done:
        if np.random.rand() < exploration_prob:
            action = env.action_space.sample()  # Исследование
(случайное действие)
        else:
            action = np.argmax(Q[state, :])  # Использование текущих
знаний

        next_state, reward, done, _ = env.step(action)

        # Обновляем значение Q-таблицы
        Q[state, action] = (1 - learning_rate) * Q[state, action] + \
                           learning_rate * (reward + discount_factor *
np.max(Q[next_state, :]))
```

```
            state = next_state

        # Уменьшаем вероятность исследования по мере обучения
        exploration_prob = max(exploration_prob * exploration_decay,
min_exploration_prob)

# Оценка обученной стратегии
num_episodes_eval = 1000
num_wins = 0

for _ in range(num_episodes_eval):
    state = env.reset()
    done = False

    while not done:
        action = np.argmax(Q[state, :])
        state, reward, done, _ = env.step(action)

        if done and reward == 1:
            num_wins += 1

win_rate = num_wins / num_episodes_eval
print(f"Win rate: {win_rate:.2f}")
```

В этом примере мы используем среду «FrozenLake-v0» из библиотеки «OpenAI Gym» для обучения агента с использованием алгоритма «Q-learning». Мы инициализируем Q-таблицу и научим агента принимать решения на основе текущих оценок Q-значений. По мере обучения мы уменьшаем вероятность случайных действий, чтобы агент начал больше использовать свои знания.

Этот пример представляет базовую реализацию «Q-learning» и может потребовать дополнительной настройки для достижения лучших результатов. Убедитесь, что у вас установлена библиотека gym перед запуском кода.

Лабораторная работа №9. Интеграция моделей искусственного интеллекта в веб-приложение.

Интеграция моделей искусственного интеллекта (ИИ) в веб-приложения может значительно улучшить их функциональность, автоматизировать процессы и предоставить более персонализированный опыт пользователям. Ниже представены некоторые ключевые аспекты интеграции:

Выбор подходящей модели ИИ: Выбор подходящей модели зависит от задачи, которую вы хотите решить. Например, для обработки естественного языка (Natural Language Processing, NLP) можно использовать модели, обученные на текстовых данных, такие как GPT-3 или BERT. Для компьютерного зрения (Computer Vision) - модели, способные анализировать изображения, например, сверточные нейронные сети (Convolutional Neural Networks, CNN).

API для ИИ: Большинство ИИ-моделей предоставляют API для интеграции в другие приложения. Например, OpenAI предоставляет API для доступа к своим моделям GPT. Это облегчает интеграцию ИИ в ваше веб-приложение.

Архитектура приложения: Решите, как будете интегрировать ИИ-модель в архитектуру вашего веб-приложения. Модель может быть интегрирована напрямую на сервере или взаимодействовать с клиентом через API.

Безопасность и конфиденциальность: Если ваша модель обрабатывает чувствительные данные, обязательно обеспечьте надежные механизмы защиты данных и обработки запросов. Работайте с экспертами по безопасности, чтобы убедиться, что риски минимизированы.

Производительность: ИИ-модели могут быть вычислительно интенсивными. Учтите это при интеграции, чтобы обеспечить хорошую производительность вашего веб-приложения. Вы можете использовать кэширование, оптимизировать запросы и выбирать подходящие аппаратные ресурсы.

Тестирование и обучение: Перед внедрением ИИ-модели удостоверьтесь, что она корректно работает и дает ожидаемые результаты. Проводите тестирование на различных данных и сценариях.

Масштабируемость: Подумайте о масштабируемости вашего приложения при увеличении числа пользователей и нагрузки. Кластеризация, балансировка нагрузки и горизонтальное масштабирование могут быть необходимы для обеспечения стабильной работы.

Мониторинг и обслуживание: После интеграции ИИ-модели важно мониторить ее работу и производительность. Выявление проблем и оперативное вмешательство позволят поддерживать качество обслуживания на высоком уровне.

Документация: Создайте подробную документацию по интеграции ИИ-модели, чтобы другие разработчики и администраторы могли легко понять, как использовать и обслуживать эту функциональность.

Интеграция ИИ в веб-приложения может сделать их более интеллектуальными и ценными для пользователей. Однако это также требует тщательного планирования, проектирования и обеспечения стабильности и безопасности системы.

Рассмотрим ШАБЛОН интеграции модели машинного обучения в React-приложение. Для этого мы предположим, что у нас есть предварительно обученная модель, способная классифицировать тексты на позитивные и негативные отзывы.

Код ниже является лишь ШАБЛОНОМ приложения React.js. Для реального запуска подобного приложения необходимо создать некотороую предобученную модель, которая должна иметь функционал доступа через API.

Шаг 1: Установка зависимостей

Первым шагом будет создание React-приложения и установка необходимых зависимостей:

```
npx create-react-app ai-integration-example
cd ai-integration-example
npm install axios
```

Мы устанавливаем пакет «axios», чтобы отправлять HTTP-запросы к нашему серверу с моделью.

Шаг 2: Создание сервера с моделью

Для простоты давайте предположим, что у нас есть сервер с предварительно обученной моделью, доступной по API.

Шаг 3: Интеграция в React-приложение

Создайте новый файл «ReviewClassifier.js» в папке «src». Этот компонент будет отвечать за интеграцию с моделью и вывод результата на экран.

```
// src/ReviewClassifier.js
import React, { useState } from 'react';
import axios from 'axios';

function ReviewClassifier() {
  const [text, setText] = useState('');
  const [prediction, setPrediction] = useState('');

  const classifyReview = async () => {
    try {
      const response = await axios.post('http://your-server-
url/api/classify', { text });
      setPrediction(response.data.prediction);
    } catch (error) {
      console.error('Error classifying review:', error);
    }
  };

  return (
    <div>
```

```
      <h2>Review Classifier</h2>
      <textarea
        rows="4"
        cols="50"
        value={text}
        onChange={(e) => setText(e.target.value)}
        placeholder="Enter a review text..."
      />
      <button onClick={classifyReview}>Classify</button>
      {prediction && <p>Prediction: {prediction}</p>}
    </div>
  );
}

export default ReviewClassifier;
```

Шаг 4: Использование компонента в приложении

Отредактируйте файл «src/App.js», чтобы использовать созданный компонент «ReviewClassifier».

```
// src/App.js
import React from 'react';
import './App.css';
import ReviewClassifier from './ReviewClassifier';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <h1>AI Integration Example</h1>
        <ReviewClassifier />
      </header>
    </div>
  );
}

export default App;
```

Шаг 5: Запуск приложения

Запустите React-приложение:

```
npm start
```

Приложение будет открыто в браузере. Компонент «ReviewClassifier» используется для того, чтобы вводить тексты отзывов и видеть их классификацию, полученную с помощью модели.

Обратите внимание, что в реальных приложениях интеграция модели может быть более сложной, например, требовать управления состоянием, обработки ошибок, безопасности и оптимизации. Также, в зависимости от задачи и выбранной модели, API-запросы и обработка ответов могут отличаться.

Если у нас нет своего сервера с предобученной моделью, мы можем воспользоваться публичными AI-сервисами, которые предоставляют API для выполнения различных задач машинного обучения. Один из таких популярных сервисов - это OpenAI, который предоставляет доступ к моделям GPT-3 через API.

Рассмотрим **пример** интеграции GPT-3 от OpenAI в ваше React-приложение. Для этого вы будете использовать библиотеку openai для работы с API OpenAI.

Шаг 1: Создание React-приложения и установка зависимостей

Сначала создайте новое React-приложение и установите необходимые библиотеки:

```
npx create-react-app gpt3-integration-example
cd gpt3-integration-example
npm install openai axios
```

Шаг 2: Интеграция с API OpenAI

Создайте файл ChatGPT.js в папке src. В этом файле вы будете использовать библиотеку openai для взаимодействия с API GPT-3.

```
// src/ChatGPT.js
import React, { useState, useEffect } from 'react';
import axios from 'axios';

function ChatGPT() {
  const [messages, setMessages] = useState([]);
  const [inputText, setInputText] = useState('');

  const sendMessage = () => {
    if (inputText.trim() === '') return;

    setMessages([...messages, { role: 'user', text: inputText }]);
    setInputText('');
  };

  useEffect(() => {
    const generateResponse = async () => {
      if (messages.length === 0) return;
```

```
      try {
        const response = await axios.post(
          'https://api.openai.com/v1/engines/davinci/completions',
          {
            prompt: messages.map(message => message.text).join('\n') +
'\nUser:',
            max_tokens: 50,
          },
          {
            headers: {
              'Content-Type': 'application/json',
              'Authorization': 'Bearer sk-
fPBr5luUDGTpIAeiVEx3T3BlbkFJb2wQNJ1aujnuUxPOdlmN', // Замените на ваш
ключ API
            },
          }
        );

        const aiResponse = response.data.choices[0].text.trim();
        setMessages([...messages, { role: 'ai', text: aiResponse }]);
      } catch (error) {
        console.error('Error sending message:', error);
      }
    };

    generateResponse();
  }, [messages]);

  return (
    <div>
      <h2>Chat with GPT-3</h2>
      <div className="chat">
        {messages.map((message, index) => (
          <div key={index} className={`message ${message.role}`}>
            {message.text}
          </div>
        ))}
      </div>
      <input
        type="text"
        value={inputText}
        onChange={(e) => setInputText(e.target.value)}
        onKeyDown={(e) => {
```

```
          if (e.key === 'Enter') {
            sendMessage();
          }
        }}
        placeholder="Type your message..."
      />
      <button onClick={sendMessage}>Send</button>
    </div>
  );
}

export default ChatGPT;
```

Шаг 3: Использование компонента в приложении

Измените файл src/App.js, чтобы использовать компонент ChatGPT.

```
// src/App.js
import React from 'react';
import './App.css';
import ChatGPT from './ChatGPT';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <h1>ChatGPT Integration Example</h1>
        <ChatGPT />
      </header>
    </div>
  );
}

export default App;
```

Шаг 4: Запуск приложения

Запустите React-приложение:

```
npm start
```

Теперь мы можем использовать компонент «Gpt3Integration» для ввода текстовых сообщений и генерации ответов с помощью модели «GPT-3».

Обратите внимание, что для использования API OpenAI вам потребуется получить ключ API OpenAI. Вам также следует ознакомиться с документацией OpenAI для более подробной информации о доступных параметрах и функциях.

Пожалуйста, помните о конфиденциальности данных и ограничениях использования API.

Помимо GPT-3 существует множество других открытых моделей искусственного интеллекта, которые могут быть интегрированы в веб-приложения. Ниже перечислены некоторые из них:

BERT (Bidirectional Encoder Representations from Transformers): Это модель, разработанная Google, специализирующаяся на обработку естественного языка. Она позволяет понимать контекст и смысл слов в предложении.

Transformers: Кроме GPT-3, семейство моделей "Transformers" также включает в себя другие модели, такие как BERT, RoBERTa, T5 и многие другие, которые могут быть полезны для различных задач NLP.

FastText: Это модель для обработки текстов, разработанная Facebook. Она специализируется на быстрой обработке текстов и поддерживает множество языков.

YOLO (You Only Look Once): Это модель компьютерного зрения, которая способна быстро и точно обнаруживать, и классифицировать объекты на изображении.

Spacy: Это библиотека для обработки текстов, которая включает предобученные модели для различных задач NLP, таких как извлечение именованных сущностей и анализ синтаксиса.

VGGNet: Это классическая модель для компьютерного зрения, которая может использоваться для классификации изображений.

DenseNet: Эта модель также применяется в компьютерном зрении и обеспечивает хорошее качество классификации и сегментации изображений.

OpenNMT: Это библиотека для машинного перевода, которая предоставляет модели для перевода текстов на разные языки.

Scikit-learn: Это библиотека для машинного обучения, включающая различные модели и алгоритмы, которые могут быть интегрированы в веб-приложения для решения разнообразных задач.

CycleGAN: Это модель генеративных нейронных сетей, используемая для преобразования изображений из одного стиля в другой.

Выбор модели зависит от задачи, которую вы хотите решить. Каждая модель имеет свои особенности, а также требования к данным и ресурсам для обучения и интеграции.

В следующем примере мы будем использовать открытый «API Clarifai», которое предоставляет возможность классифицировать изображения с помощью предварительно обученных моделей.

Шаг 1: Создание React-приложения и установка зависимостей

Сначала создайте новое React-приложение и установите библиотеку axios для выполнения HTTP-запросов:

```
npx create-react-app image-classification-app
cd image-classification-app
npm install axios @material-ui/core
```

Шаг 2: Создание компонента для загрузки и классификации изображений

Создайте файл ImageClassification.js в папке src.

```
// src/ImageClassification.js
import React, { useState } from 'react';
import { Button, CircularProgress, Container, Paper, TextField,
Typography } from '@material-ui/core';
import axios from 'axios';

function ImageClassification() {
  const [imageUrl, setImageUrl] = useState('');
  const [predictions, setPredictions] = useState([]);
  const [loading, setLoading] = useState(false);

  const classifyImage = async () => {
    setLoading(true);

    try {
      const response = await axios.post(
        'https://api.clarifai.com/v2/models/e0be3b9d6a454f0493ac3a3078
4001ff/outputs',
        {
          inputs: [
            {
              data: {
                image: {
                  url: imageUrl,
                },
              },
            },
          ],
        },
        {
          headers: {
            'Content-Type': 'application/json',
            'Authorization': 'Bearer YOUR_CLARIFAI_API_KEY', //
Замените на ваш ключ API
```

```
          },
        }
      );

      const concepts = response.data.outputs[0].data.concepts;
      setPredictions(concepts);
    } catch (error) {
      console.error('Error classifying image:', error);
    } finally {
      setLoading(false);
    }
  };

  return (
    <Container>
      <h2>Image Classification with Clarifai API</h2>
      <TextField
        label="Image URL"
        variant="outlined"
        value={imageUrl}
        onChange={(e) => setImageUrl(e.target.value)}
        fullWidth
        style={{ marginBottom: '10px' }}
      />
      <Button variant="contained" color="primary"
onClick={classifyImage}>
        Classify
      </Button>
      {loading && <CircularProgress />}
      {predictions.length > 0 && (
        <Paper elevation={3} style={{ marginTop: '20px', padding:
'20px' }}>
          <Typography variant="h6">Predictions:</Typography>
          <ul>
            {predictions.map((prediction) => (
              <li key={prediction.id}>{prediction.name}
({(prediction.value * 100).toFixed(2)}%)</li>
            ))}
          </ul>
        </Paper>
      )}
    </Container>
  );
}
```

```
export default ImageClassification;
```

Шаг 3: Использование компонента в приложении

Измените файл src/App.js, чтобы использовать компонент ImageClassification.

```
// src/App.js
import React from 'react';
import './App.css';
import ImageClassification from './ImageClassification';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <h1>Image Classification App</h1>
        <ImageClassification />
      </header>
    </div>
  );
}

export default App;
```

Шаг 4: Запуск приложения

Запустите React-приложение:

```
npm start
```

В этом примере мы используем открытый «API Clarifai» для классификации изображений. Вы должны получить ключ API Clarifai и подставить его вместо 'YOUR_CLARIFAI_API_KEY'.

Обратите внимание, что это лишь базовый пример. В реальном приложении вы также можете добавить дополнительные функции, обработку ошибок и стилизацию для улучшения пользовательского опыта.

Лабораторные работы №10-15. Проектная работа

Задание на проект: Разработка комплексного приложения на основе искусственного интеллекта с использованием фреймворка машинного обучения

Цель проекта: Разработать полноценное приложение, использующее технологии искусственного интеллекта на базе выбранного фреймворка машинного обучения. Проект должен решать практическую задачу и демонстрировать умение применять методы и техники машинного обучения для создания полезных и инновационных решений.

Тема проекта: (Тема, предложенная преподавателем, или выбранная студентом, например, обнаружение объектов на изображениях, автоматический перевод текстов, рекомендательная система и т.д.)

Задачи проекта:

1. **Исследование и планирование (Лаб. работа №10)**: Изучить тему проекта, провести обзор существующих решений и методов, выбрать подходящий фреймворк машинного обучения (например, TensorFlow, PyTorch, scikit-learn и др.). Подробно определить, какую задачу вы планируете решить с использованием искусственного интеллекта.
2. **Сбор и подготовка данных (Лаб. работа №11)**: Собрать или подготовить набор данных, необходимых для обучения и оценки модели. Произвести очистку, аугментацию (при необходимости) и предобработку данных.
3. **Обучение модели (Лаб. работа №12)**: Разработать и обучить модель машинного обучения, на основе выбранного фреймворка. Произвести настройку гиперпараметров, выбор архитектуры и техники обучения.
4. **Разработка приложения (Лаб. работа №13)**: Создать веб-приложение или мобильное приложение, которое интегрирует разработанную модель машинного обучения. Реализовать интерфейс для взаимодействия пользователя с моделью.
5. **Тестирование и оценка (Лаб. работа №14)**: Протестировать работу приложения с использованием реальных данных. Произвести оценку производительности, точности и других метрик модели и приложения.
6. **Документирование (Лаб. работа №15)**: Создать документацию, которая включает описание задачи, использованных данных, архитектуры модели, методов обучения, интерфейса приложения и инструкции по развертыванию.

Ожидаемые результаты:

- Полноценное приложение, использующее искусственный интеллект для решения практической задачи.
- Обученная и настроенная модель машинного обучения.
- Документация, описывающая проект и его компоненты.

Критерии оценки:

Оценка проекта будет проводиться на основе следующих критериев:

- Актуальность выбранной темы и задачи.
- Качество исследования и выбора подходов.

- Качество обработки и предобработки данных.
- Качество разработанной модели машинного обучения.
- Работоспособность и эффективность приложения.
- Качество документации и оформления проекта.

Контрольная работа

Контрольная работа №1 Тема: Фреймворки для работы с нейронными сетями и их применение

Часть 1. TensorFlow

1. TensorFlow разработан .....

A. Команда IBM

B. Команда Microsoft

C. Команда Google Brain

D. Ничего из вышеперечисленного

Q2. Имеются..... основной тип тензора, который вы можете создать в TensorFlow.

A. 2

Б.3

С.4

Д.5

B3. В чем преимущество TensorFlow?

A. Он имеет отличную поддержку сообщества.

B.It предназначен для использования различного серверного программного обеспечения (графические процессоры, ASIC) и т. д. а также очень параллельно.

C.It обладает уникальным подходом, который позволяет отслеживать прогресс обучения наших моделей и отслеживать несколько метрик.

D. Все вышеперечисленное

Q4. Создайте необходимый конвейер данных с помощью TensorFlow Problem Учитывая следующий набор данных, создайте конвейер данных, который: - Фильтруйте данные только по положительным значениям. - Умножает значение данных на 2. - Перемешайте данные с buffer_size = 2. Распечатайте начальный и конечный элементы набора данных, чтобы проверить результаты. набор данных = [12,15,67,-56,78,90,25,-890,-45,67,90,45,34,-100,300]

Мой ответ: 1. Распечатайте исходный импорт тензорфлоу кадра данных в формате tf tf_dataset = tf.data.Dataset.from_tensor_slices(набор данных) для i в tf_dataset: print(i.numpy()) 2. Создайте конвейер данных и распечатайте окончательный набор данных tf_dataset_new = tf_dataset.filter(lambda x: x>0).map(lambda a: a*2).shuffle(2) для i в tf_dataset_new: print(i.numpy())

Q5. Какой метод инициализации переменных используется по умолчанию в tf.get_variable()? Ответ: Если инициализатор имеет значение None, будет использоваться инициализатор по умолчанию, переданный в области переменных. Если это тоже Нет, будет использоваться glorot_uniform_initializer. Функция glorot_uniform_initializer инициализирует значения из равномерного распределения.

B6. Что делают функции набора данных TensorFlow cache() и prefetch()?

Ответ: Преобразование tf.data.Dataset.cache может кэшировать набор данных либо в памяти, либо в локальном хранилище. Это избавит некоторые операции (например, открытие файлов и чтение данных) от выполнения в течение каждой эпохи. Следующие эпохи будут повторно использовать данные, кэшированные преобразованием кэша. Предварительная выборка перекрывает предварительную обработку и выполнение модели обучающего шага. Пока модель выполняет обучающий шаг s, входной конвейер считывает данные для шага s+1. Это сокращает время шага до максимума (в отличие от суммы) обучения и время, необходимое для извлечения данных.

Часть 2. PyTorch

Q1. Что такое PyTorch?

Ответ: PyTorch — это фреймворк глубокого обучения, который популярен благодаря простоте использования и гибкости. Он используется как в исследовательских, так и в производственных целях и был принят многими компаниями.

B2. Как определить тензор в PyTorch?

Ответ: Тензор — это многомерный массив, используемый для численных вычислений в PyTorch. Тензоры могут быть созданы из списков Python или кортежей с помощью фонарика. Функция Tensor().

B3. В чем разница между модулями nn и nn.functional в PyTorch?

Ответ: Модуль nn в PyTorch предназначен для построения нейронных сетей, а модуль nn.functional содержит ряд функций для работы с нейронными сетями. Основное отличие состоит в том, что модуль nn предоставляет API более высокого уровня для работы с нейронными сетями, в то время как модуль nn.functional предоставляет более низкоуровневые функции.

B4. Как найти производные функции в PyTorch?

Ответ: Производные функции вычисляются с помощью градиента. Есть четыре простых шага, с помощью которых мы можем легко вычислить производные. Эти шаги заключаются в следующем: - Инициализация функции, для которой мы будем вычислять производные. - Установите значение переменной, которая используется в функции. - Вычислите производную функции с помощью метода backward (). - Выведите значение производной с помощью grad.

Часть 3. Керас Q1.

Какая серверная функция в Keras возвращает соглашение о формате данных изображения по умолчанию? 1. Image_format_data() 2. image_format() 3. image_data_format() 4. Format_image()

B2. Поддерживает ли Keras сверточные или рекуррентные нейронные сети?

1. Да, он поддерживает оба

2. Да, он поддерживает только сверточную сеть

3. Да, он поддерживает только рекуррентную нейронную сеть

4. Нет Q3.

Кто разработал Керас?

1. Франсуа Шолле

2. Пит Шиннерс

3. Уэс МакКинни

Q4. Чтобы вернуться к текущему имени серверной части, какую из следующих серверных функций вы бы использовали в Keras?

1. Бэкенд()

2. keras.backend.backend()

3. Backend.keras()

Q5. В каком из следующих слоев входные данные преобразуются в стандартизированную форму? 1. Уровень нормализации

2. Объединение уровня

3. Уровень шума

4. Рекуррентный слой

Q6. В Керасе «последовательный» относится к ____?

1. Несколько слоев

2. Однослойный

3. Вся модель

Часть 4. H2O, AutoML Q1.

Что такое AutoML в Python?

Ответ: AutoML — это методы автоматического и быстрого обнаружения хорошо работающего конвейера модели машинного обучения для задачи прогнозного моделирования. ... Тремя наиболее популярными библиотеками AutoML для Scikit-Learn являются Hyperopt-Sklearn, Auto-Sklearn и TPOT.

B2. Кто изобрел AutoML?

Ответ: Quoc Le За AutoML стоит его движок под названием Neural Architecture Search, изобретенный Куок Ле, пионером в области искусственного интеллекта. Куок Ле стал соучредителем Google Brain в 2011 году вместе с Эндрю Нг и Джеффом Дином. В 2012 году Ле опубликовал знаменитую «кошачью» статью, в которой распознал кошек на основе 10 миллионов изображений.

B3. Для чего H2O.ai используется?

Ответ: H2O — это полностью открытая распределенная платформа машинного обучения в памяти с линейной масштабируемостью. H2O поддерживает наиболее широко используемые алгоритмы статистики и машинного обучения, включая машины с

градиентным ускорением, обобщенные линейные модели, глубокое обучение и многое другое.

В4. Как запустить H2O в Python?

Ответ: Откройте окно терминала и запустите записную книжку jupyter. Создайте новую записную книжку Python, нажав кнопку «Создать» в верхнем левом углу. На этом этапе вы можете начать использовать Jupyter Notebook для выполнения команд H2O Python

Домашняя работа

Домашняя работа №1 Тема: Фреймворки для работы с нейронными сетями и их применение

Часть 1: Теоретическая часть

1. Объясните, что такое нейронные сети и какова их структура. Какие типы слоев обычно применяются в нейронных сетях?
2. Опишите важность использования фреймворков для создания нейронных сетей. Какие преимущества дает использование фреймворков?
3. Расскажите о различных фреймворках для создания нейронных сетей, таких как TensorFlow, PyTorch, Keras, MXNet и другие. Какие особенности у каждого из них?
4. Объясните, что такое трансферное обучение и какие фреймворки предоставляют возможности для его реализации? Приведите примеры сценариев, в которых трансферное обучение может быть полезным.
5. Что такое обучение с подкреплением и какие фреймворки поддерживают его реализацию? Какие алгоритмы обучения с подкреплением можно применять с использованием этих фреймворков?

Часть 2: Практическая часть

1. Выберите один из основных фреймворков (например, TensorFlow или PyTorch) и реализуйте нейронную сеть для решения задачи классификации на известном наборе данных, например, MNIST или CIFAR-10. Опишите структуру модели, процесс обучения и полученные результаты.
2. Продолжите предыдущий пункт, добавив применение метода трансферного обучения. Загрузите предобученную модель (например, из библиотеки Keras Applications) и дообучите её на вашем наборе данных. Сравните производительность модели до и после трансферного обучения.
3. Выберите библиотеку для обучения с подкреплением (например, OpenAI Gym или Stable Baselines) и реализуйте агента для решения задачи обучения с подкреплением. Опишите выбранный алгоритм, процесс обучения и результаты.

По результатам работы необходимо оформить отчет, который должен содержать следующие моменты:

- ответы на теоретические вопросы в текстовой форме, предоставив ясные и краткие объяснения. Используйте технические термины и определения, чтобы дать полное понимание понятий и особенностей фреймворков.
- код реализации нейронной сети для задачи классификации. Включите в комментарии описание структуры модели, функции потерь, метод оптимизации и метрики оценки производительности.
- графики или таблицы с результатами обучения вашей модели на выбранном наборе данных (например, график изменения функции потерь и точности на обучающем и проверочном наборах данных).

- код реализации метода трансферного обучения, если это входит в задание. Покажите, как вы загрузили и дообучили предобученную модель, и предоставьте результаты сравнения до и после трансферного обучения.
- код реализации агента для обучения с подкреплением, если это входит в задание. Покажите, как вы создали агента, использовали библиотеки для обучения с подкреплением и предоставьте результаты агента в среде.

Домашняя работа № 2 Реализация проекта

1.Обнаружение спама с помощью TensorFlow

Если вы когда-либо пользовались Gmail, вы должны быть знакомы с его системой uber vigilant для обнаружения спама. Эта функция, которая, как утверждается, блокирует более 99,9 % фишинговых писем и вредоносных программ от попадания в ваш почтовый ящик, сделала пакет Google Suite еще более желанным для его пользователей. Таким образом, первым проектом TensorFlow и, возможно, самым известным в списке будет построение вашей модели обнаружения спама!

Хотя эта модель построения может не соответствовать стандартам Google, создать базовую модель обнаружения спама с помощью TensorFlow, которая составляет основу детектора спама Google, довольно просто. Рекуррентная нейронная сеть, обученная с использованием набора данных, такого как классификация текстовых сообщений со спамом, должна хорошо служить этой цели.

2. Классификация изображений с помощью TensorFlow

Классификация изображений, по сути, включает в себя разделение изображения на фиксированное количество классов на основе его содержимого. Airbnb, в котором представлены миллионы домов для путешественников, использует классификацию изображений, чтобы гарантировать, что их гости получат то, что они ищут, классифицируя объявления с фотографиями по разным типам номеров, чтобы убедиться, что объявления соответствуют стандартам, и проверить правильность информации. Вы могли бы рассмотреть возможность начать с моделей глубокой нейронной сети (DNN), таких как VGG, ResNet и Inception, для построения модели классификации изображений. В TensorFlow доступно множество предварительно обученных глубоких нейронных сетей.

Однако вам нужно будет рассмотреть возможность использования методов переноса обучения или переобучения всей модели в зависимости от имеющегося в вашем распоряжении набора данных. В любом случае вам нужно будет модифицировать последние несколько слоев нейронной сети в соответствии с вашими требованиями к выходным размерам и в некоторой степени переобучить DNN, пока он не сможет достичь удовлетворительной производительности. Для этого проекта TensorFlow вы могли бы сразу перейти к многоклассовой задаче классификации с помощью этого набора данных или начать с простой задачи классификации кошек и собак, используя этот набор данных.

Вот интересный решаемый проект TensorFlow о том, как создать аналогичный поисковик изображений, который вам, возможно, будет интересно проверить.

3. Оптическое распознавание символов с помощью TensorFlow

Оптическое распознавание символов или OCR имеет множество применений, поскольку позволяет извлекать текст из изображений (даже рукописный) в текстовый формат. Вы, должно быть, сталкивались со многими программными приложениями, которые обеспечивают деформацию документа, обнаружение границ и даже оптическое распознавание символов в режиме онлайн и офлайн. Kingsoft WPS, созданный с использованием TensorFlow, является одним из инструментов на рынке, который обладает всеми этими функциональными возможностями.

Хотя на их основе существуют целые приложения, по иронии судьбы, создать собственную систему оптического распознавания символов не так уж сложно. На самом деле, для этого проекта вы могли бы пойти дальше и попробовать распознавание рукописного текста с использованием модели ResNet в Keras / TensorFlow с популярным общедоступным набором данных MNIST.

Ознакомьтесь с этим решенным проектом, чтобы узнать, как создать OCR с нуля?

4. AR-фильтры для лица с использованием TensorFlow

ModiFace, который группа L'Oreal приобрела в 2018 году, позволяет пользователям видеть, как продукт будет смотреться на них перед покупкой, а не просто просматривать фотографии продукта. Таким образом, первый промежуточный проект TensorFlow заключается в настройке собственного простого приложения для определения ориентиров на лице. Для этого вы могли бы просто использовать модель Media Pipe Facemesh и создавать свои пользовательские фильтры и классные AR-эффекты (сложность будет зависеть от того, насколько сложными вы их создадите ...). Вы можете либо использовать статические изображения (например https://www.kaggle.com/laurentmih/aisegmentcom-matting-human-datasets /), либо (что еще лучше) работать с фронтальной камерой для применения эффектов в режиме реального времени.

5. Системы рекомендаций с использованием TensorFlow

Будь то веб-сайты для покупок или потоковые приложения, сортировка товаров по релевантности или предоставление их в разделе ‘Вам может понравиться это" становится все более распространенным, настолько, что вы можете не знать сервис, который этого не делает (в конце концов, хорошие рекомендательные системы могут создать или разорить компанию).

Всеми любимый Twitter не новичок в этих системах ранжирования. Отойдя от вековой практики представления твитов в обратном хронологическом порядке, Twitter теперь использует платформу TensorFlow для ранжирования твитов, чтобы убедиться, что их пользователи никогда не пропустят наиболее релевантные твиты. Они учитывают комбинацию факторов, таких как содержание твита, его недавность, количество ретвитов или лайков, автор твита и связь пользователя с ними, и даже активность (как часто и насколько интенсивно кто-то использует Twitter), чтобы убедиться, что вы ничего не пропустите.

Оснащен арсеналом постоянно используемых API-интерфейсов Twitter для разработчиков (https://developer.twitter.com/en) и TensorFlow, вы могли бы реализовать это с помощью подхода, основанного на глубоком обучении, возможно, используя рекуррентные нейронные сети (или библиотеку TensorFlow Recommenders (TFRS), специализированную для построения моделей рекомендательных систем) или даже изучить деревья решений с градиентным усилением (которые, по наблюдениям самого Twitter, становились все более популярными на RecSys 2020).). Однако действуйте здесь осторожно, поскольку на

создание рекомендательных систем и их модификацию уходит много внимания и усилий, и вы можете себе представить, почему…

5. Автоматическая классификация товаров для покупок с помощью TensorFlow

Хотя задачи классификации обычно считаются довольно простыми, сложность этого проекта обусловлена характером данных или их отсутствием. Naver Shopping - это торговый портал, который использует TensorFlow для автоматической классификации более десятка миллионов новых товаров в день по заранее определенным категориям, что в противном случае было бы огромной задачей, если не невыполнимой. Однако добиться этого было непросто, поскольку категории продавцов и детали не всегда совпадают с категориями, определенными Naver. Поэтому это вынудило их использовать комбинацию методов, таких как модель CNN-LSTM для названия продукта, MobileNet для изображения продукта и даже простую сверточную модель нейронной сети, среди прочего.

Чтобы начать с вашей собственной задачи по классификации продуктов, вы могли бы рассмотреть возможность использования нескольких общедоступных наборов данных (https://www.kaggle.com/c/retail-products-classification/data), а затем решить, какая модель или алгоритм (или их совокупность) лучше всего подойдет для ваших нужд.

Проекты NLP TensorFlow

1. Распознавание речи

Повсеместного распространения систем распознавания речи, таких как Siri от Apple, Alexa от Amazon и Microsoft Cortana, должно быть достаточно, чтобы подчеркнуть важность систем распознавания речи. Проведя небольшое исследование, внедрить собственную базовую систему распознавания речи должно быть довольно просто.

Одним из возможных способов достижения этой цели является обучение CNN с помощью спектрограмм MFCC, полученных на основе необработанных данных. Для этой цели вы можете использовать этот общедоступный обучающий набор данных: https://www.kaggle.com/c/TensorFlow-speech-recognition-challenge/data.

ПРИМЕЧАНИЕ: Этот проект, возможно, был упрощен здесь. Однако, если вы готовы взяться за более сложную задачу (например, фактическую обработку естественного языка), вы заметите, что в этой области предстоит изучить гораздо больше.

2. Распознавание намерений с помощью TensorFlow

Распознавание намерений становится все более важным, поскольку мы наслаждаемся цифровым пространством почти так же, как и физическим миром вокруг нас. Например, Facebook разработала RoBERTa, модифицированную версию модели BERT, для решения проблемы модерации контента и заявила, что смогла увеличить блокировку вредоносных сообщений на 70%. С появлением BERT распознавание намерений было радикально упрощено и улучшено, однако его ценность ни в коем случае не уменьшилась. Поэтому разработка вашей собственной системы распознавания намерений станет блестящим проектом. Вы можете начать с модели 12/768 (BERT-Base), которую можно загрузить по этой ссылке - https://storage.googleapis.com/bert_models/2020_02_20/uncased_L-12_H-768_A-12.zip

3. Анализ настроений

Этот проект предполагает создание системы, которая может использовать фреймворк Tensorflow для построения модели, которая может классифицировать настроение данного текста как положительное или отрицательное. Вы можете подготовить набор данных самостоятельно, просмотрев в Twitter обзоры фильмов по вашему выбору. Как только у вас будет готовый набор данных, внедрите методы NLP для полировки набора данных, прежде чем использовать его в качестве входных данных для модели обучения. Для выполнения задачи классификации вы можете использовать модель глубокого обучения, такую как модель классификации нейронных сетей.

4. DeepSpeech

DeepSpeech - это проект TensorFlow, разработанный Mozilla, целью которого является создание механизма автоматического распознавания речи (ASR) с открытым исходным кодом, доступного каждому. В проекте используются методы глубокого обучения для обучения нейронной сети, которая может преобразовывать произносимые слова в текст.

Это мощный и гибкий движок ASR, который можно обучать на любом языке и адаптировать к различным доменам и приложениям. Проект с открытым исходным кодом, что означает, что любой может внести свой вклад в его разработку и использовать его для своих собственных проектов. Использование TensorFlow упрощает обучение и развертывание моделей на различных платформах, включая настольные компьютеры, серверы и мобильные устройства.

Проекты TensorFlow с использованием наборов данных Kaggle

1. Распознавание эмоций

У Kaggle есть набор данных Facial Expression Recognition Challenge Dataset, который вы можете использовать для создания системы распознавания эмоций на лице. Набор данных содержит изображения лиц людей в натуральную величину с различными эмоциями, такими как гнев, отвращение, страх, радость, грусть, удивление и нейтральность. Эмоции представлены числами от 0 до 6. Используя TensorFlow, можно обучить модель глубокого обучения распознавать эмоции по изображениям лиц, а затем использовать ее для прогнозирования. Можно использовать такую систему для понимания опыта клиентов по выражению их лиц.

2. Распознавания эмоций речи

Kaggle предоставляет различные наборы данных для распознавания эмоций речи, такие как аудиовизуальная база данных эмоциональной речи Ryerson и база данных песенt. Этот набор данных содержит аудиозаписи людей, говорящих с различными эмоциями, такими как нейтральность, спокойствие, радость, грусть, гнев, страх, отвращение и удивление. Цель этого проекта - использовать TensorFlow для построения моделей глубокого обучения, способных распознавать эмоции по речевым сигналам. Эти модели могут найти применение в различных областях, таких как здравоохранение и обслуживание клиентов, где понимание человеческих эмоций может иметь решающее значение.

3. Прогнозирование временных рядов

У Kaggle есть набор данных о поездках на такси в Нью-Йорке, который содержит данные о поездках на такси в Нью-Йорке с 2009 по 2015 год. Набор данных включает в себя такую

информацию, как места получения и высадки, время получения и высадки, расстояние поездки и сумму тарифа. Этот набор данных часто используется для построения моделей машинного обучения, которые могут прогнозировать стоимость проезда в такси или количество посадок / высадок такси за определенный период времени. Вы можете использовать TensorFlow для построения моделей глубокого обучения, которые могут прогнозировать количество остановок такси в определенном районе или стоимость проезда за данную поездку на основе мест посадки и высадки пассажиров и других переменных.

Зачет по дисциплине

Зачет по дисциплине проводится в форме презентации проекта "Разработка комплексного приложения на основе искусственного интеллекта", который был реализован в лаб. работах №10-15.

Формат представления проекта:

Презентация: Подготовить презентацию, включающую следующие разделы:

- Введение в тему и актуальность выбранной задачи.
- Обзор используемых фреймворков и методов.
- Описание собранного набора данных и предобработки.
- Процесс обучения модели и выбранную архитектуру.
- Демонстрацию работы приложения и интерфейса.
- Результаты тестирования и оценку точности модели.
- Выводы и возможные направления дальнейшего развития.

Отчет по проекту: Подготовить письменный отчет, включающий следующие разделы:

- Введение в тему и постановку задачи.
- Описание выбранного фреймворка и методов обучения.
- Описание набора данных и предобработки.
- Подробное описание архитектуры модели и процесса обучения.
- Описание разработанного приложения и его функциональных возможностей.
- Результаты тестирования, включая метрики точности и производительности.
- Выводы о выполненной работе и достигнутых результатах.

Ожидаемые результаты:

- Готовое веб-приложение для обнаружения объектов на изображениях.
- Презентация, демонстрирующая работу приложения и результаты обучения модели.
- Письменный отчет, описывающий все этапы выполнения проекта.

Критерии оценки:

Оценка проекта будет проводиться на основе следующих критериев:

- Качество обученной модели и ее производительность.
- Качество разработанного приложения и его интерфейса.
- Глубина исследования темы и выбора методов.
- Оценка точности и стабильности модели.
- Оформление презентации и отчета.

Список ссылок и литературных источников

Общие ресурсы по машинному обучению и искусственному интеллекту:

1. "Python Machine Learning" by Sebastian Raschka and Vahid Mirjalili.
2. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow" by Aurélien Géron.
3. "Deep Learning" by Ian Goodfellow, Yoshua Bengio, and Aaron Courville.
4. Coursera: Machine Learning by Andrew Ng - Ссылка
5. Fast.ai - Бесплатные курсы по глубокому обучению - Ссылка

Фреймворки для глубокого обучения:

1. TensorFlow - Официальный сайт: https://www.tensorflow.org/
   - TensorFlow Tutorials: https://www.tensorflow.org/tutorials
   - TensorFlow 2.x Guide: https://www.tensorflow.org/guide
2. PyTorch - Официальный сайт: https://pytorch.org/
   - PyTorch Tutorials: https://pytorch.org/tutorials
   - Deep Learning with PyTorch: A 60 Minute Blitz: https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html
3. Keras - Официальный сайт: https://keras.io/
   - Keras Documentation: https://keras.io/api/
   - Keras Tutorials: https://keras.io/guides/

Фреймворки для обработки данных и машинного обучения:

1. Scikit-learn - Официальный сайт: https://scikit-learn.org/
   - Scikit-learn User Guide: https://scikit-learn.org/stable/user_guide.html
   - Scikit-learn Tutorials: https://scikit-learn.org/stable/tutorial/index.html
2. XGBoost - Официальный сайт: https://xgboost.readthedocs.io/
   - XGBoost Tutorials: https://xgboost.readthedocs.io/en/latest/tutorials/index.html

Обработка изображений и компьютерное зрение:

1. OpenCV - Официальный сайт: https://opencv.org/
   - OpenCV Tutorials: https://docs.opencv.org/master/d9/df8/tutorial_root.html
2. TensorFlow Object Detection API: https://github.com/tensorflow/models/tree/master/research/object_detection
3. YOLO (You Only Look Once) - https://pjreddie.com/darknet/yolo/

Обработка естественного языка (NLP) и генерация текстов:

1. Natural Language Processing with Python - https://www.nltk.org/book/
2. spaCy - Официальный сайт: https://spacy.io/
   - spaCy Tutorials: https://spacy.io/usage/tutorials

Необходимое программное обеспечение:

1. Visual Studio Code (VS Code): https://code.visualstudio.com/
2. Python: https://www.python.org/
3. React.js: https://reactjs.org/
4. Material-UI: https://mui.com/
5. Clarifai API: https://www.clarifai.com/

6. GitHub: https://github.com/
7. Jupyter Notebook: https://jupyter.org/
8. OpenAI ChatGPT: https://beta.openai.com/signup/
9. Операционная система: Windows, macOS или Linux.