ЛАБОРАТОРНАЯ РАБОТА № 5

«Исследование методов ИИ с использованием JavaScript»

1. Цель работы

Изучение методов разработки ПО с использованием методов Machine Learning и DeepLearning, получение практических навыков разработки ПО с использованием JavaScript.

2 Краткие теоретические сведения

Машинное обучение (Machine Learning) и глубокое обучение (Deep Learning) — это два взаимосвязанных, но отличающихся подхода в области искусственного интеллекта.

Machine Learning (ML) – Область искусственного интеллекта, которая фокусируется на создании алгоритмов, способных обучаться на данных и делать предсказания или принимать решения. ML включает в себя множество методов и моделей, таких как регрессия, классификация, кластеризация и методы снижения размерности.

Deep Learning (DL) – Подмножество машинного обучения, которое использует многослойные искусственные нейронные сети для моделирования и решения сложных задач. DL особенно эффективен при работе с большими объемами данных и сложными структурами данных, такими как изображения, аудио и текст.

Структура моделей:

- ML использует более простые модели, такие как линейная регрессия, логистическая регрессия, деревья решений, случайные леса, методы опорных

векторов и другие алгоритмы. Эти модели часто требуют ручного подбора признаков и инженерии признаков.

- DL использует многослойные нейронные сети, включая свёрточные нейронные сети (CNN) для обработки изображений, рекуррентные нейронные сети (RNN) для последовательных данных и глубокие полностью связанные сети. Эти модели могут автоматически извлекать иерархические признаки из данных, минимизируя необходимость ручного подбора признаков.

Объем данных:

- ML эффективен при работе с меньшими и средними наборами данных. Производительность моделей может существенно зависеть от качества и подготовки данных.
- DL требует больших объемов данных для эффективного обучения. Глубокие нейронные сети обладают большим числом параметров, которые необходимо настроить, и поэтому они лучше работают с большими наборами данных.

Аппаратные ресурсы:

- ML может работать на обычных компьютерах и не требует значительных вычислительных ресурсов. Обучение моделей ML может быть менее ресурсоемким.
- DL требует мощных аппаратных ресурсов, таких как графические процессоры (GPU) или тензорные процессоры (TPU), из-за высокой вычислительной сложности обучения нейронных сетей.

Работа с признаками:

- В ML значительное внимание уделяется ручному выбору и обработке признаков. Качество модели сильно зависит от качества и релевантности выбранных признаков.

- В DL нейронные сети могут автоматически извлекать сложные признаки из сырых данных, что снижает необходимость в ручной обработке признаков.

Применение:

- Machine Learning используется для более простых задач, где объем данных относительно небольшой или средний, и задача не требует сложной структуры. Примеры включают кредитный скоринг, диагностику заболеваний, прогнозирование временных рядов, кластеризацию клиентов и другие.
- Deep Learning применяется для более сложных задач, где объем данных велик и требуется анализ сложных структур. Примеры включают распознавание изображений и лиц, обработку естественного языка (NLP), синтез речи, автоматический перевод, управление автономными транспортными средствами и другие.

Примеры библиотек и инструментов

- Machine Learning: Scikit-learn, XGBoost, LightGBM, CatBoost, H2O.ai.
- Deep Learning: TensorFlow, Keras, PyTorch, Caffe, MXNet.

TensorFlow.js и его использование

TensorFlow.js — это библиотека для машинного обучения в JavaScript. Библиотека предоставляет высокоуровневые API для создания и обучения нейронных сетей, а также низкоуровневые операции для выполнения сложных вычислений.

Для создания нейронной сети в TensorFlow.js используется последовательная модель (sequential model), состоящая из слоев. Основные типы слоев включают:

- Dense (Полносвязный слой): Каждый нейрон связан с каждым нейроном предыдущего слоя.

- Conv2D (Сверточный слой): Применяется фильтры к входным данным для выделения пространственных признаков.
- MaxPooling2D (Макспулинг): Уменьшает размерность данных, сохраняя важные признаки.

3 Порядок выполнения лабораторной работы

3.1 Установка необходимых компонентов и инициализация проекта.

Node.js — это платформа для разработки серверных и сетевых приложений, которая основана на движке V8 от Google. Установить можно с официального сайта.

После установки node.js, создайте дирректорию проекта и выполните команду для инициализации проекта:

```
npm init -y
```

Далее нужно установить необходимые библиотеки:

```
npm install @tensorflow/tfjs @tensorflow/tfjs-node
npm install mnist
npm install canvas
```

3.2 Обучение модели.

В корне проекта создайте файл train.js:

```
const tf = require('@tensorflow/tfjs-node');
const mnist = require('mnist');

// Загрузка данных MNIST
const set = mnist.set(8000, 2000);
const trainingSet = set.training;
const testSet = set.test;

// Подготовка данных
const prepareData = (data) => {
  const images = data.map(item => item.input);
  const labels = data.map(item => item.output);
  return {
    images: tf.tensor2d(images, [images.length, 784]),
    labels: tf.tensor2d(labels, [labels.length, 10])
  };
};

const trainData = prepareData(trainingSet);
const testData = prepareData(testSet);

// Определение модели
```

```
const model = tf.sequential();
model.add(tf.layers.dense({ units: 128, activation: 'relu', inputShape: [784] }));
model.add(tf.layers.dense({ units: 64, activation: 'relu' }));
model.add(tf.layers.dense({ units: 10, activation: 'softmax' }));

// Компиляция модели
model.compile({
  optimizer: 'adam',
  loss: 'categoricalCrossentropy',
  metrics: ['accuracy']
});

// Обучение модели
(async () => {
  await model.fit(trainData.images, trainData.labels, {
    epochs: 20,
    validationData: [testData.images, testData.labels],
    callbacks: {
      onEpochEnd: (epoch, logs) => {
        console.log(`Epoch ${epoch + 1}: loss = ${logs.loss.toFixed(4)}, accuracy
= ${logs.acc.toFixed(4)}`);
      }
    }
  });

  // Сохранение модели
  await model.save('file://./mnist-model');
})();
```

Описание кода train.js

Импорт библиотек: Импортируются TensorFlow.js и библиотека mnist.

Загрузка данных MNIST: Загружаются тренировочный и тестовый наборы данных MNIST.

Подготовка данных: Данные преобразуются в тензоры, которые могут быть использованы TensorFlow.js.

Определение модели: Создается последовательная модель, добавляются три полносвязных слоя.

Компиляция модели: Модель компилируется с оптимизатором 'adam' и функцией потерь 'categoricalCrossentropy'.

Обучение модели: Модель обучается на тренировочных данных, в процессе выводятся метрики потерь и точности после каждой эпохи.

Сохранение модели: После обучения модель сохраняется в локальное хранилище.

Для обучения модели запустите команду:

```
node train.js
```

3.3 Использование обученной модели.

Создайте файл predict.js для загрузки и использования модели:

```
const tf = require('@tensorflow/tfjs-node');
const mnist = require('mnist');
const fs = require('fs');
const { createCanvas } = require('canvas');

// Загрузка данных MNIST
const set = mnist.set(0, 10);
const testSet = set.test;

// Подготовка данных
const prepareData = (data) => {
  const images = data.map(item => item.input);
  return tf.tensor2d(images, [images.length, 784]);
};

// Загрузка и предсказание
(async () => {
  const model = await tf.loadLayersModel('file://./mnist-model/model.json');
  const testImages = prepareData(testSet);
  const predictions = model.predict(testImages).argMax(-1).arraySync();

  // Визуализация результатов
  const canvas = createCanvas(280, 280);
  const ctx = canvas.getContext('2d');

  for (let i = 0; i < 10; i++) {
    const imageData = tf.tensor2d(testSet[i].input, [28, 28]).mul(255).toInt().ar-
raySync();
    const predictedLabel = predictions[i];
    const trueLabel = testSet[i].output.indexOf(1);

    ctx.clearRect(0, 0, 280, 280);
    for (let y = 0; y < 28; y++) {
      for (let x = 0; x < 28; x++) {
        const color = imageData[y][x];
        ctx.fillStyle = `rgb(${color}, ${color}, ${color})`;
        ctx.fillRect(x * 10, y * 10, 10, 10);
      }
    }
    ctx.fillStyle = 'red';
    ctx.font = '20px Arial';
    ctx.fillText(`Predicted: ${predictedLabel}`, 10, 260);
    ctx.fillText(`True: ${trueLabel}`, 150, 260);

    fs.writeFileSync(`output${i + 1}.png`, canvas.toBuffer('image/png'));
  }
})();
```

Описание кода predict.js

Импорт библиотек: Импортируются TensorFlow.js, mnist для загрузки данных, fs для работы с файловой системой и canvas для создания изображений.

Загрузка данных MNIST: Загружаются тестовые данные MNIST.

Подготовка данных: Данные преобразуются в тензоры для использования TensorFlow.js.

Загрузка и предсказание: Загружается обученная модель, выполняются предсказания на тестовых данных.

Визуализация результатов: Создаются изображения предсказанных и истинных меток, сохраняются в файлы.

Для использования модели, после того как отработал скрипт train.js и в папку mnist-model сохранилась модель, выполните команду:

```
node predict.js
```

3.4 Эксперименты с моделью.

Попробуйте изменить количество и размер слоев модели, используйте другие типы слоев. Экспериментируйте с различными значениями гиперпараметров, таких как количество эпох, размер батча, тип оптимизатора.

3.5 Выводы.

Сформулируйте выводы, проанализировав разницу работы модели при тех или иных экспериментах. Оформите отчет по проделанной работе.