|  |  |
| --- | --- |
|  | **Российский государственный социальный университет** |

**Лабораторная работа № 2.**

**по дисциплине «Интеллектуальные информационные системы»**

|  |  |
| --- | --- |
| **ФИО студента** | Салов Артём Владимирович |
| **Направление подготовки** | Программная инженерия |
| **Группа** | ПИН-Б-О-Д-2021-1 |

# **Москва 2023**

# Привести пример доступного средства.

В качестве доступного средства для разработки исскуственной информационной системы, в данной работе был выбран язык программирования C# и доступные для него дополнения.

1.1. Основы и история C#

C# — это объектно-ориентированный язык программирования, разработанный Microsoft. Он был впервые выпущен в 2000 году и стал основным языком для платформы .NET. C# предоставляет мощные инструменты для разработки современных приложений, таких как Windows-приложения, веб-сайты и игры.

В сравнении с C и C++, C# является более безопасным и проще в использовании. Он имеет множество инструментов для создания отличных и эффективных приложений. C# поддерживает объектно-ориентированную парадигму, многоязычность, задействует сборку мусора и имеет множество библиотек.

2. Библиотеки для работы с ИИ на C#

При создании приложений для работы с ИИ на C# используются следующие библиотеки:

— ML.NET: это библиотека для машинного обучения, разработанная Microsoft. Она работает в среде .NET и поддерживает множество задач, начиная от классификации до кластеризации и регрессии. ML.NET поддерживает конвейерную модель машинного обучения и предоставляет инструменты для работы с TensorFlow, ONNX и другими библиотеками машинного обучения.

— Accord.NET: это библиотека для обработки аудио-, видео- и изображений. Она имеет множество инструментов для машинного обучения и алгоритмов распознавания. Accord.NET также поддерживает научные вычисления и разработку информационных систем.

— AForge.NET: это библиотека для обработки изображений и видео. Она включает методы для обработки изображений, распознавания образов, обучения нейронных сетей и обнаружения объектов.

— TensorFlow.NET: это библиотека для работы с библиотекой машинного обучения TensorFlow. Она позволяет использовать TensorFlow на платформе .NET и имеет множество инструментов для машинного обучения и обработки изображений.

— Keras.NET: это библиотека для работы с библиотекой машинного обучения Keras. Keras.NET позволяет использовать Keras на платформе .NET и имеет множество инструментов для машинного обучения и обработки изображений.

# Привести примеры использования описанного средства.

Работа с OpenAI на C#: создание и использование моделей искусственного интеллекта

Современные технологии искусственного интеллекта создают новые возможности для разных областей науки и техники. Мы можем использовать искусственный интеллект для выполнения задач, которые раньше были возможны только для человека. Сегодня мы рассмотрим, как можно работать с OpenAI на C#: создание и использование моделей искусственного интеллекта.

OpenAI – это компания, которая занимается исследованиями в области искусственного интеллекта. Они создали набор библиотек и инструментов, которые позволяют использовать искусственный интеллект в своих проектах. В частности, OpenAI создала библиотеку GPT-3 (Generative Pre-training Transformer 3), которая позволяет создавать тексты высокого качества.

Создание модели искусственного интеллекта

Для того чтобы начать работу с OpenAI на C#, нужно создать модель искусственного интеллекта. Модель – это набор правил, которые позволяют искусственному интеллекту делать предсказания или принимать решения в конкретной области.

Прежде всего, вам нужно зарегистрироваться на сайте OpenAI и получить ключ API. Этот ключ позволит вам использовать библиотеки GPT-3 для создания модели искусственного интеллекта.

Чтобы создать модель, нужно использовать библиотеку AI Dungeon. Для этого нужно установить библиотеку через NuGet:

|  |  |
| --- | --- |
| 1 | Install-Package ai-dungeon-csharp |

После установки библиотеки, нужно создать класс Gpt3Bot, который будет использоваться для создания текстов. Класс должен содержать методы для создания и обновления модели:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57 | public class Gpt3Bot  {      private readonly string \_apiKey;      private readonly string \_model;        public Gpt3Bot(string apiKey, string model)      {          \_apiKey = apiKey;          \_model = model;      }        public async Task<string> GenerateStringAsync(string prompt)      {          using (var httpClient = new HttpClient())          {              httpClient.DefaultRequestHeaders.Add("Authorization", $"Bearer {\_apiKey}");                var requestData = new              {                  context = \_model,                  prompt = prompt              };                var requestJson = JsonConvert.SerializeObject(requestData);              var content = new StringContent(requestJson, Encoding.UTF8, "application/json");                var response = await httpClient.PostAsync("https://api.openai.com/v1/engine/" + \_model + "/completions", content);                var jsonString = await response.Content.ReadAsStringAsync();                var responseJson = JObject.Parse(jsonString);                var responseText = responseJson["choices"][0]["text"].ToString();                return responseText;          }      }        public async Task UpdateModelAsync(string example)      {          using (var httpClient = new HttpClient())          {              httpClient.DefaultRequestHeaders.Add("Authorization", $"Bearer {\_apiKey}");                var requestData = new              {                  model = \_model,                  documents = new List<string> { example }              };                var requestJson = JsonConvert.SerializeObject(requestData);              var content = new StringContent(requestJson, Encoding.UTF8, "application/json");                var response = await httpClient.PostAsync("https://api.openai.com/v1/engines/" + \_model + "/documents/create", content);          }      }  } |

В конструкторе класса вы передаете ключ API и имя модели, которую вы будете использовать. Метод GenerateStringAsync принимает параметр prompt – это строка, которая будет использоваться для генерации текста. Метод UpdateModelAsync принимает новый пример (текст), который будет использоваться для обновления модели.

Использование модели

После того, как вы создали класс для работы с моделью, можно использовать его для создания текста. Например, можно использовать модель для создания продолжения предложения. Для этого нужно вызвать метод GenerateStringAsync и передать часть предложения в параметр prompt:

|  |  |
| --- | --- |
| 1  2  3 | var bot = new Gpt3Bot("your\_api\_key", "your\_model");  var prompt = "The quick brown fox jumps over the lazy dog.";  var result = await bot.GenerateStringAsync(prompt); |

В этом примере мы создаем объект класса Gpt3Bot и вызываем его метод GenerateStringAsync с параметром prompt. В этом параметре мы передаем часть предложения, которую мы хотим завершить. Метод GenerateStringAsync отправляет запрос к библиотеке GPT-3, которая генерирует продолжение предложения на основе обученной модели. Результат хранится в переменной result.

Еще один пример использования модели – создание краткого описания для статьи. Для этого нужно создать объект класса Gpt3Bot и вызвать его метод GenerateStringAsync с параметром title. В качестве параметра мы передаем заголовок статьи:

|  |  |
| --- | --- |
| 1  2  3 | var bot = new Gpt3Bot("your\_api\_key", "your\_model");  var title = "Working with OpenAI in C#: Creating and Using Artificial Intelligence Models";  var result = await bot.GenerateStringAsync(title); |

Метод GenerateStringAsync возвращает краткое описание статьи, сгенерированное на основе обученной модели.

Обновление модели

Чтобы модель работала лучше, ее нужно периодически обновлять новыми примерами. Для этого нужно использовать метод UpdateModelAsync. Метод принимает новый текст, который будет использоваться для обновления модели.

|  |  |
| --- | --- |
| 1  2  3 | var bot = new Gpt3Bot("your\_api\_key", "your\_model");  var example = "The quick brown fox jumps over the lazy dog.";  await bot.UpdateModelAsync(example); |

Пример добавляется в обучающую выборку модели, чтобы позволить искусственному интеллекту работать с новыми данными.

Вывод

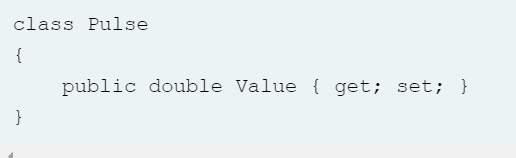
OpenAI – это мощный инструмент для создания интеллектуальных приложений. Библиотека GPT-3 позволяет создавать тексты, которые не отличаются от текстов, написанных человеком. Чтобы работать с OpenAI на C#, нужно сначала создать модель искусственного интеллекта, которая будет использоваться для генерации текста или принятия решений в конкретной области. Затем можно использовать эту модель для создания расширенных функций своего приложения.

# Привести свой пример решения задачи с помощью описанного средства

Реализация простой нейронной сети на C # .NET

Пульс

Импульс - это электрический сигнал, проходящий через дендрит нейрона, который составляет основу данных (значение, хранящееся в двойном типе данных). Ниже представлена ​​простая форма Pulse, в которой один или несколько символов, которые в данном случае являются Value



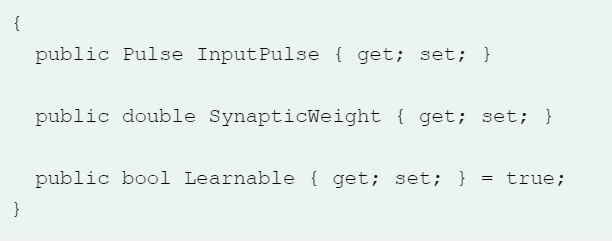
Дендрит

Простая проекция нейрона, которая получает импульсный вход и передает его в сам нейрон для вычислений. Я определил три свойства дендрита

InputPulse: входной сигнал.

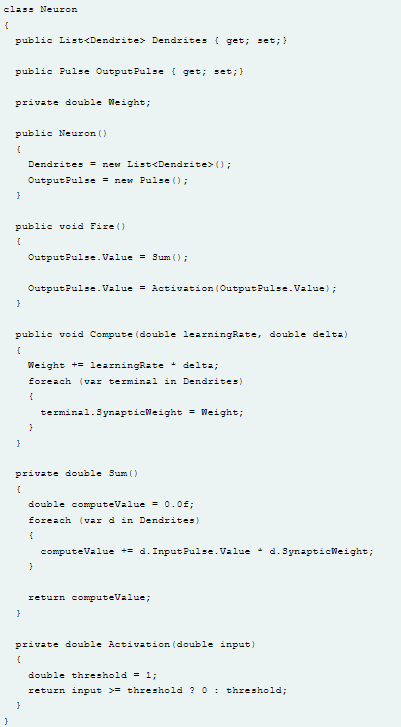
SynapticWeight: сила связи между дендритом и синапсисом двух нейронов.

Обучаемость: просто логический тип, используемый во время обучения для настройки значения SynapticWeight.



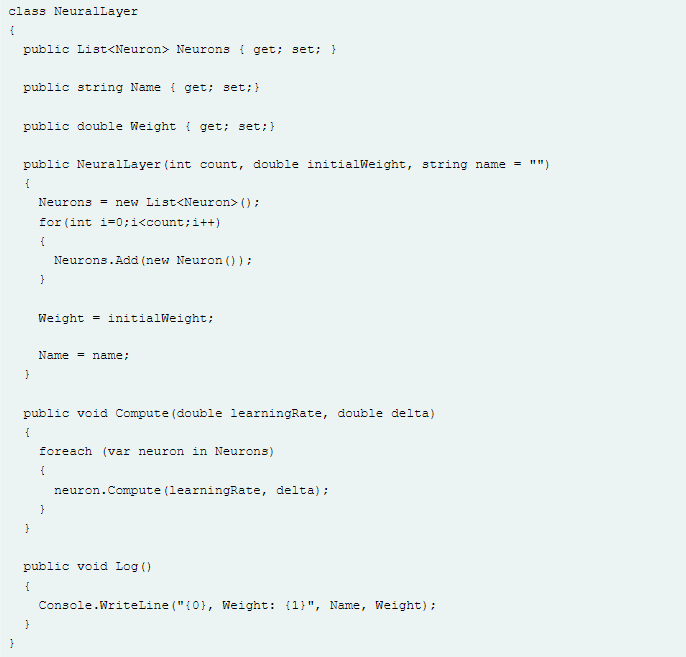
Нейрон

Сам нейрон в сочетании со своей основной функцией «Огонь» и Дендрит. Рабочий процесс нейрона заключается в том, что он получает входные значения, производит взвешенную сумму всех входных сигналов от всех дендритов и передает их функции активации, которая в данном случае является функцией активации Step. Выход Activation назначается OutputPulse, который проходит через аксон нейрона.



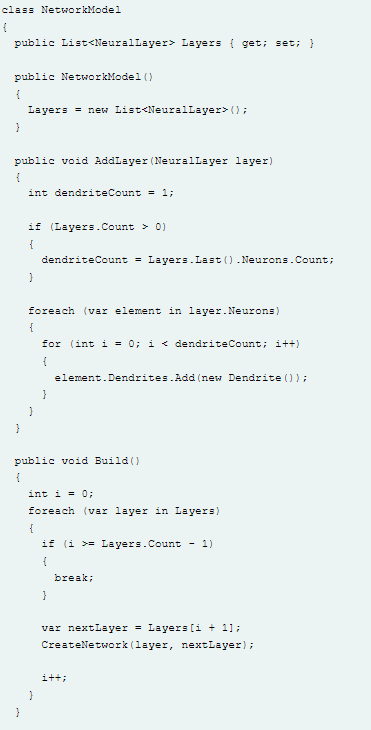
NeuralLayer

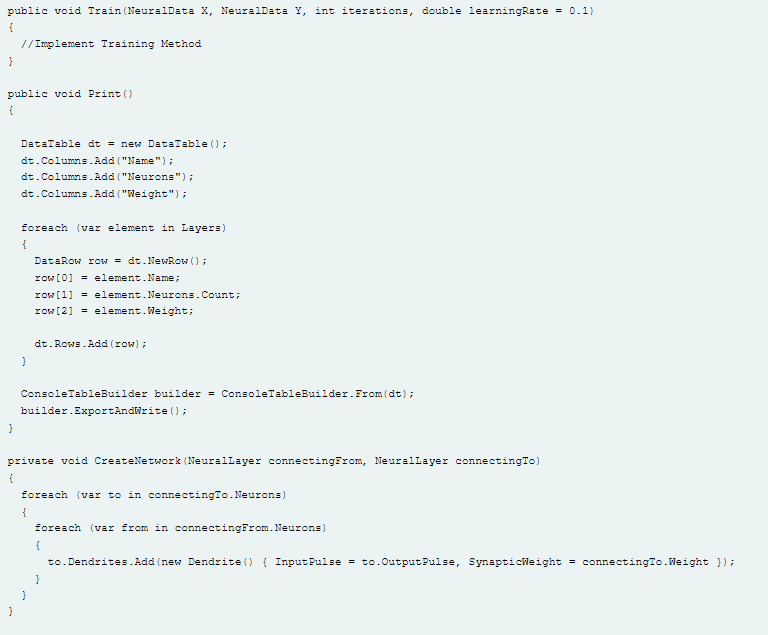
Как вы видите на изображении ниже, мы складываем группу нейронов в виде слоев для построения сложной сети. Я создал еще один класс, который сначала создаст слой.



NetworkModel

NetworkModel - это полная архитектура со слоями нейронов и функцией Train.





Вызовите NetworkModel из метода Main в консольном проекте, как показано ниже. Это создаст сеть из трех слоев с начальными весами.

