**Федеральное государственное автономное образовательное учреждение высшего образования**

**«РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ» (РУДН)**

Основное учебное подразделение ФФМиЕН

Направление/специальность 02.03.02 «Фундаментальная информатика и информаци-

онные технологии»

**ОТЧЕТ**

**o прохождении учебной практики**

**«Анализ управления манипулятором с помощью алгоритма обучения с подкреплением»**

Ломакина София Васильевна

Курс, группа 3 курс, группа НФИбд 02-19

Место прохождения практики «Отдел информационно-технического обеспечения естественно-научных факультетов УИТО и СТС РУДН»

Сроки прохождения с «18» апреля 2022 г. по «19» июня 2022 г.

Руководители практики:

от РУДН Фомин М.Б., к.ф.-м.н, доцент, доцент

кафедры информационных технологий

от организации (предприятия)

Киселёв Г. А., к.т.н., ст. преподаватель

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва 2022 г.

**Содержание**

[Введение 3](#_2et92p0)

[Цель исследования 3](#_i7uvlutfsecz)

[Методы машинного обучения 4](#_tyjcwt)

[Обучение с подкреплением (Reinforcement learning) 5](#_3dy6vkm)

[Q-Learning 6](#_4d34og8)

[Deep Q Network (DQN) 8](#_2s8eyo1)

[Deep Deterministic Policy Gradient (DDPG) 11](#_17dp8vu)

[Выбор ПО для работы с манипулятором 12](#_3rdcrjn)

[Подготовка среды 12](#_26in1rg)

[Pick and Place 15](#_lnxbz9)

[Обзор понимания объекта 19](#_35nkun2)

[Полносвязные слои нейронных сетей в машинном обучении 19](#_1ksv4uv)

[DQN 21](#_44sinio)

[Заключение 29](#)

[Библиография 30](#_2jxsxqh)

# Введение

Современная технология и промышленность быстро развиваются, что приводит к непрерывному изменению требований к технике и машинам. Робототехнические системы не являются исключением из этого правила – постоянно идет работа над улучшением их функциональных возможностей. В настоящее время одним из наиболее перспективных направлений в данной области являются алгоритмы и системы машинного обучения, такие как DQN (Deep Q-network).

DQN – это алгоритм, основанный на нейронных сетях, который способен обучаться в условиях неопределенности и нестабильности. Он был создан для решения задач, требующих принятия решений с учетом текущей ситуации и наличия информации о результате этих решений. Применение DQN на практике может быть очень широким – от игр и различных приложений, до управления роботами и другими техническими системами.

В данной исследовательской работе будет рассмотрена модель манипулятора, в которой будет использован алгоритм DQN для управления им. Манипуляторы широко используются в промышленности и робототехнике и представляют собой механические устройства с различными видами эквивалентных звеньев, которые позволяют перемещать, переносить и манипулировать объектами в пространстве. Управление манипулятором посредством алгоритмов машинного обучения является важной проблемой, и в данной работе будет представлен один из подходов к ее решению.

# Цель исследования

Целью данной работы является разработка и имплементация алгоритма DQN для управления манипулятором, а также анализ его результатов по сравнению с другими методами управления. В ходе исследования будет проведен анализ различных факторов, влияющих на эффективность управления манипулятором, таких как количество входных данных, скорость принятия решений и другие. Результаты исследования могут быть применены для улучшения управления манипуляторами и другими робототехническими системами.

В качестве входных данных для исследования выступают параметры манипулятора, такие как размеры, масса, количество звеньев, а также информация о задачах, которые должен выполнять манипулятор. На выходе получается оптимальное управление манипулятором для выполнения конкретной задачи.

# Методы машинного обучения

Машинное обучение - это раздел искусственного интеллекта, который изучает методы, алгоритмы и технологии, позволяющие компьютерной системе извлекать закономерности из набора данных, обучаться на основе этих данных и делать предсказания или принимать решения на основе полученных знаний. Машинное обучение применяется во многих областях, таких как обработка естественного языка, компьютерное зрение, рекомендательные системы и др.

Центральной идеей машинного обучения является существующая математическая связь между любой комбинацией входных и выходных данных. Модель машинного обучения не имеет сведений об этой взаимосвязи заранее, но может сгенерировать их, если будет предоставлено достаточное количество наборов данных. Это означает, что каждый алгоритм машинного обучения строится вокруг модифицируемой математической функции.

Пример:

* Мы «обучаем» алгоритм, давая ему следующие комбинации ввода-вывода [input / output (i,o)]: (2,10), (5,19) и (9,31).
* Алгоритм вычисляет соотношение между входом и выходом следующим образом: o = 3 × i + 4.
* Затем мы задаем ввод 7 и просим предсказать результат. Алгоритм может автоматически определить выход как 25.

Хотя это базовое понимание, машинное обучение фокусируется на том принципе, что все сложные точки данных могут быть математически связаны компьютерными системами, если у них достаточно данных и вычислительной мощности для обработки этих данных. Следовательно, точность выходных данных прямо пропорциональна величине входных данных.

Алгоритмы можно разделить на четыре стиля обучения в зависимости от ожидаемого результата и типа ввода.

1. Машинное обучение с учителем
2. Машинное обучение без учителя

# Обучение с подкреплением (Reinforcement learning)

Машинное обучение с подкреплением - это раздел машинного обучения, который изучает, каким образом агент может обучаться, взаимодействуя с окружающей средой и принимая решения на основе полученного опыта. В отличие от других видов машинного обучения, где данные обучения передаются агенту напрямую, в машинном обучении с подкреплением агенту приходится экспериментировать со своим окружением, чтобы получить награду или штраф.

Машинное обучение с подкреплением связано с изучением оптимального способа поведения агента в конкретной среде. Агент находится в конкретном состоянии, на что среда отвечает определенной реакцией. Кроме того, среда может сообщать агенту некую ценность, которую он может использовать для принятия решений. Агент выполняет определенные действия и получает награду или штраф в зависимости от результатов своих действий в текущем состоянии.

Машинное обучение с подкреплением схоже с обучением с учителем. Разница в том, что алгоритм обучения с подкреплением не получает полную информацию о результатах своих действий, а только получает награду или штраф в зависимости от выбранной им стратегии. В случае обучения с учителем система обучается на основе ответов учителя на предъявленные ей данные.

Одним из главных преимуществ машинного обучения с подкреплением является возможность обучения агента на основе эмпирических данных. В качестве примера можно привести использование этой технологии в медицине. Машинное обучение с подкреплением может использоваться для определения оптимального лечения, учитывая возможные побочные эффекты и другие факторы, которые могут влиять на здоровье пациента.

Машинное обучение с подкреплением также может использоваться при разработке роботов. Роботы используют текущее состояние окружающей среды для принятия решений, когда они выполняют определенную задачу. Алгоритмы машинного обучения с подкреплением могут помочь роботам оптимизировать задачи, чтобы они могли завершать их более эффективно и быстро.

Q-Learning, DQN и DDPG являются наиболее распространенными алгоритмами обучения с подкреплением.

# Q-Learning

Q-Learning - это алгоритм обучения с подкреплением, который может быть использован для обучения агентов принимать оптимальные решения в динамических окружениях. Суть алгоритма заключается в том, что агент анализирует текущую ситуацию, выбирает наилучшее действие и осуществляет его. Затем агент наблюдает за состоянием окружающей среды, которое изменяется в результате совершенного действия, и получает награду. Данные полученные из такого взаимодействия между окружающим миром и агентом, используются для обновления внутренней информации агента о том, какие действия приводят к каким наградам в каждой из ситуаций.

В Q-Learning каждый раз агент выбирает действие, которое максимизирует значение функции качества действий (Q-функции), которая хранит информацию о том, какой же выигрыш ожидается от каждого действия в каждом состоянии. Значение Q-функции может быть вычислено известными формулами обновления, которые используют количество полученной награды, ожидаемый будущий выигрыш и текущее значение Q-функции в состоянии и совершенном действии.

Первоначально Q-функция заполняется случайными значениями, однако с каждым новым опытом она настраивается для получения наилучших значений. Этот процесс может продолжаться на протяжении многих итераций, в ходе которых агент пытается найти оптимальные решения в различных ситуациях.

С математической точки зрения Q-learning это без модельный алгоритм ОП, основанный уравнении Беллмана:

Уравнение утверждает, что значение Q для определенной пары состояние-действие должно быть наградой, полученной при переходе в новое состояние (путем выполнения этого действия), добавленной к значению наилучшего действия в следующем состоянии.

Алгоритм Q-Learning можно выразить следующими шагами:

1. Инициализировать Q-функцию на случайных значениях
2. Наблюдать текущее состояние и выбрать действие, основываясь на текущем значении Q-функции
3. Совершить выбранное действие и получить награду
4. Обновить значение Q-функции для текущего состояния и действия с использованием формулы обновления
5. Продолжать процесс до тех пор, пока не будет достигнут нужный уровень оптимальности

Преимущества Q-Learning заключаются в его способности адаптироваться к динамическим условиям, таким как изменение правил игры или состояния окружающей среды. Этот алгоритм также может быстро обучить агента решать задачи в сложных средах, так как он может быстро адаптироваться к изменениям.

Однако, Q-Learning также имеет недостатки. В частности, этот алгоритм может страдать от проблемы "проклятия размерности", которая возникает при работе в пространствах с большим количеством переменных. Кроме того, в случае наличия множества состояний и действий, Q-Learning может потребовать большое количество времени для обучения агента.

В целом, Q-Learning является эффективным алгоритмом для обучения агентов принимать решения в динамических окружениях. Он позволяет адаптироваться к изменению условий и быстро находить оптимальные решения в сложных средах.

# Deep Q Network (DQN)

Deep Q-Network (DQN) - это алгоритм глубокого обучения обратной связи, который был разработан для решения проблемы продвинутых игр на Atari. Он использует нейронную сеть для оценки ценности действий в окружении и обновляет ее, чтобы улучшить качество принимаемых решений. DQN является одним из наиболее известных алгоритмов обучения с подкреплением и позволяет создавать агентов, способных самостоятельно обучаться и принимать лучшие решения в сложных средах.

Алгоритм DQN построен на основе технологии Q-обучения (Q-learning), применяемой в обучении с подкреплением. Он использует нейронную сеть, которая принимает состояние среды, выполняет преобразование и возвращает значения для каждого возможного действия. Нейронная сеть обучается обновлять Q-функцию, которая выражает значение каждого действия в окружении на основе текущего состояния. Эта обученная Q-функция используется для выбора действия, которое должен выполнить агент в следующем шаге.

DQN включает ряд технических усовершенствований, которые решают проблемы, возникающие при обучении с подкреплением с использованием нейронных сетей. Например, он использует методы реплея (replay) и фиксированной Q-цели (fixed Q-target), чтобы уменьшить нестабильность обучения и предотвратить очень большие изменения ценности действий, которые могут привести к выбору неправильного действия.

Метод реплея предполагает сохранение копий всех состояний, которые агент проходил на протяжении игры. Эти состояния воспроизводятся случайным образом во время процесса обучения, что позволяет нейронной сети рассматривать каждое состояние регулярно и избежать зацикливания на определенных шаблонах.

Метод фиксированной Q-цели заключается в использовании отдельной целевой Q-сети для вычисления ценности действий в окружении. Это предотвращает схлопывание (collapse) сети, которое может произойти при обучении с подкреплением, когда она использует свои же значения Q-функции во время обучения.

Математически формулы DQN алгоритма выглядят следующим образом:

Функция ценности Q(s, a): это функция, которая оценивает ожидаемую сумму дисконтированных наград для действия "а" в состоянии "s". Она определяется как:

,где - награда полученная за действие в состоянии , - состояние, в которое перешел агент после выполнения действия в состоянии , - параметр дисконтирования будущих наград.

Политика действий "" для состояния "", π(s): это функция, которая выбирает действие "" в состоянии "" на основе функции ценности . Она определяется как:

,где argmax - оператор выбора максимального значения функции ценности.

Оптимизация функции потерь: для обновления функции ценности используется метод градиентного спуска. Функция потерь определяется как среднеквадратичная ошибка между оценкой Q(s, a) и целевым значением:

Целевое значение определяется как:

,где argmax определяет действие, которое было бы выбрано в состоянии , если бы политика действий π использовалась вместо .

Таким образом, для обновления функции ценности Q используется следующий градиентный спуск:

,где - градиент функции потери по функции ценности в состоянии st и на действии at.

Алгоритм DQN используется в широком спектре задач машинного обучения, от игр до управления роботами и финансового анализа. Он позволяет агентам обучаться самостоятельно и принимать оптимальные решения в сложных и динамических средах, открывая новые возможности для интеллектуальных систем во многих областях.

# Deep Deterministic Policy Gradient (DDPG)

Алгоритм Deep Deterministic Policy Gradient (DDPG) является алгоритмом глубокого обучения с подкреплением, который использует Deep Q Network (DQN) для обучения оценки ценности действий (Q-значений) и определения лучшего действия для каждого состояния окружающей среды.

Однако, в отличие от DQN, который может использовать только дискретные действия, DDPG позволяет работать с непрерывными пространствами действий. Кроме того, DDPG использует две нейронные сети - Actor и Critic, для определения лучшей стратегии поведения агента и оценки ценности действий.

В Actor-сети используется функция, которая на вход принимает состояние окружающей среды и выходит с выбранным действием. Critic-сеть используется для оценки ценности действий в данном состоянии.

DDPG использует off-policy метод, который позволяет обучаться на данных, которые были собраны другим агентом или использовать ранее собранные данные. Это делает алгоритм стабильнее и более эффективным по сравнению с алгоритмами on-policy, такими как REINFORCE и A2C.

В целом, DDPG - это продвинутый алгоритм глубокого обучения, который позволяет обучаться в сложных средах и работать с непрерывными пространствами действий там, где DQN не может быть использован.

# Выбор ПО для работы с манипулятором

Для работы с моделированием и программированием манипулятора была выбрана среда V-REP (Virtual Robot Experimentation Platform). Это трёхмерный роботизированный симулятор, основанный на распределенной архитектуре управления: управляющие программы (или сценарии) могут быть непосредственно прикреплены к объектам сцены и выполняться одновременно в потоковом или непоточном режиме. Это делает V-REP очень универсальным и идеальным инструментом для мультироботов и позволяет пользователям моделировать роботизированные системы таким же образом, как и в реальности, где управление распределено большую часть времени.

В самой программе V-REP можно использовать скриптовый язык программирования Lua, но в данной работе было решено использовать язык программирования Python, так как он позволяет работать с дополнительными библиотеками и имеет большую вариативность действий.

# Подготовка среды

Так как для работы был выбран не внутренний скриптовый язык, то взаимодействовать с V-REP получится через предоставляемый им Remote API. V-REP предоставляет средства контроля симуляцией из внешнего приложения или внешнего устройства (например с настоящего робота, удалённого компьютера). API содержит сотни функций, которые могут быть вызваны через коммуникационные сокеты, с минимальной задержкой и нагрузкой на сеть. Всё это происходит в скрытом от пользователя виде. По средствам удалённого API можно взаимодействовать с V-REP в синхронном и асинхронных режимах (по умолчанию действует асинхронный режим), и даже удалённо управлять симулятором (удалённо загружать сцену, начинать и останавливать работу симуляции). Для этого необходимо скопировать несколько файлов из директории с установленной программой. В ней содержится папка programming, затем в remoteApiBindings, из которой нужно скопировать несколько файлов в созданную ранее папку для проекта. А именно файлы из папки python (vrep.py, vrepConst.py, simpleTest.py). И файлы из папки lib в соответсвии с разрядностью ОС (это может быть remoteApi.so или remoteApi.dll).

Для начала работы с Remote API нужно активировать его на стороне сервера, то есть в самой среде V-REP. Для этого нужно активировать прослушку порта в любом активном потоково зависимом скрипте (например дописав команду в имеющийся Lua-скрипт или добавив какой-нибудь примитив, тот же куб, и прикрепив скрипт к нему). После чего рядом с именем нашего куба появится иконка зависимого скрипта и его настроек в иерархии сцены. В панеле Скрипты (Scripts) также появится соответствующий пункт. По нажатию на иконку скрипта в текстовом редакторе выведется стандартный каркас потоковых скриптов, в него же следует добавить строку открытия и активации порта на языке Lua.

SimExtRemoteApiStart(19999)

Далее необходимо соединиться с удалённым сервером на том порту, что открыт в скрипте. Так как V-REP запущен локально, то адрес 127.0.0.1. Согласно документации по Remote Api для Python, аргументы данной функции обозначают следующее:

clientID = vrep.simxStart('127.0.0.1', 19999, True, True, 5000, 5)

* **connectionAddress**: IP адрес, где сервер находится (то есть адрес сервера с V-REP).
* **connectionPort**: Порт, с которым происходит соединение.
* **waitUntilConnected**: Если True, тогда функция блокируется на время соединения или пока не закончится время тайм-аута.
* **doNotReconnectOnceDisconnected**: Если True, тогда коммуникационный поток не будет пытаться переподсоединиться в случае потери соединения.
* **timeOutInMs**: Тайм-аут соединения в милисекундах (для первого соединения).
* **commThreadCycleInMs**: Указывает на то, как часто должен происходить обмен пакетами данных. Уменьшение этого числа повышает чувствительность, а в роли значения по умолчанию рекомендуется 5.

Благодаря этим настройкам появляется возможность устанавливать удаленное соединение для подключения Python скриптов для моделей роботов.

Для работы с Remote API разработчики приложения V-REP (CoppeliaSim) предоставили открытый доступ к руководству с функциями, необходимыми для взаимодействия со средой - Legacy remote API functions (Python).

Legacy remote API для V-Rep включает в себя набор функций, доступных для использования из Python, которые позволяют управлять симуляцией V-Rep. Некоторые из основных функций Legacy remote API для Python включают:

* **simxStart**: устанавливает соединение между кодом Python и V-Rep.
* **simxFinish**: закрывает соединение между кодом Python и V-Rep.
* **simxGetObjectHandle**: получает идентификатор объекта в V-Rep на основе его имени.
* **simxGetObjectPosition**: получает координаты объекта в V-Rep (три значения x, y, z).
* **simxSetObjectPosition**: устанавливает новые координаты объекта в V-Rep.
* **simxGetObjectOrientation**: получает ориентацию объекта в V-Rep (три угла поворота).
* **simxSetObjectOrientation**: устанавливает новую ориентацию объекта в V-Rep.
* **simxGetVisionSensorImage**: получает изображение с видеокамеры в V-Rep.
* **simxEraseFile**: удаляет файл на V-Rep сервере.

Это только некоторые из функций, доступных в Legacy remote API для Python. Эти функции могут быть полезны для управления роботами, объектами, камерами и другими элементами в симуляции V-Rep из Python.

# Pick and Place

Алгоритм Pick and Place для робота-манипулятора - это метод автоматизации работы робота, который включает в себя процедуру подбора предмета (Pick) и его переноса на другое место (Place).

Для реализации данного алгоритма робот должен быть оснащен системой восприятия (камерой, сенсорами), блоком управления, механизмом захвата и системой передачи манипулятора.

Основное предназначение алгоритма Pick and Place - это автоматизация манипуляций с объектами, при которых требуется их перемещение с одного места на другое. Примеры таких задач могут включать в себя комплектацию упаковок, сортировку продуктов или складскую логистику.

Алгоритм генерируется на основе данных, которые робот получает от системы восприятия. Сначала он должен определить местонахождение предмета на поверхности, используя камеру или другие сенсоры. Затем робот оценивает свою позицию и выбирает оптимальный путь для перемещения манипулятора к предмету.

Далее, используя механизм захвата, робот схватывает предмет и перемещает его на нужное место. При этом он должен следить, чтобы объект был установлен правильно и не падал в процессе передвижения.

Алгоритм Pick and Place может выполняться как в режиме автоматического выполнения, так и контролируемого оператором. В первом случае робот может работать самостоятельно, сканируя заранее заданные участки, выбирая предметы и перемещая их туда, где нужно. Во втором - оператор контролирует процесс и направляет робота при необходимости.

Это один из самых простых алгоритмов для манипуляторов, благодаря которому можно настроить, например, конвейерную работу. Можно реализовать разные настройки, например, чтобы манипулятор захватывал объект из определенных заданных координат и перемещал его в такую же заданную точку, но также можно настроить манипулятор на поиск объекта, находящегося рядом, с помощью различных датчиков. Этот алгоритм идет как основа для более сложных (например, для алгоритма обучения с подкреплением), т. к. этот алгоритм не требует от манипулятора запоминания своих прошлых действий, а просто позволяет механизму взаимодействовать с окружающей средой и объектами.



Рисунок 1. Пример работы стандартного алгоритма Pick and Place, выполненного в программе V-REP

Одна из наиболее подходящих библиотек для работы с V-REP на Python это библиотека PyRep. PyRep доступна на GitHub и может быть установлена через pip.

Для решения поставленной задачи, необходимо выполнить следующие шаги:

1. Загрузить в среду V-REP модель.
2. Написать функцию определения расстояния между манипулятором и объектами в среде.
3. Написать функцию обнаружения куба и перемещения его в заданную точку, используя манипулятор.

Далее с помощью библиотеки PyRep написать ряд основных функций:

#Функция определения расстояния между манипулятором и объектами в среде

def distance\_to\_object(obj):

robot\_pos = robot.get\_tip().get\_position()

obj\_pos = obj.get\_position()

return np.sqrt(np.sum((robot\_pos — obj\_pos)\*\*2))

#Функция обнаружения и перемещения куба

def pick\_and\_place(cube, target\_pos):

# Определение положения куба и целевой точки

cube\_pos = cube.get\_position()

target = Shape.create(type=PrimitiveShape.CUBOID, size=[0.05, 0.05, 0.05], color=[1.0, 1.0, 0.0])

target.set\_position(target\_pos)

#Нахождение расстояния между кубом и манипулятором

distance = distance\_to\_object(cube)

#Перемещение манипулятора над кубом

robot.arm.set\_joint\_target\_positions([cube\_pos[0], cube\_pos[1], cube\_pos[2] + 0.1, 0, 0, 0])

pr.step()

robot.arm.wait\_until\_arrived()

#Захват куба

robot.gripper.close()

pr.step()

robot.gripper.wait\_until\_closed()

#Перемещение куба в целевую точку

robot.arm.set\_joint\_target\_positions([target\_pos[0], target\_pos[1], target\_pos[2] + 0.1, 0, 0, 0])

pr.step()

robot.arm.wait\_until\_arrived()

#Освобождение куба

robot.gripper.open()

pr.step()

robot.gripper.wait\_until\_open()

#Возврат манипулятора в начальное положение

robot.arm.set\_joint\_target\_positions(robot.arm.get\_joint\_positions())

pr.step()

robot.arm.wait\_until\_arrived()

#Удаление целевой точки

target.remove()

#Поиск куба рядом с манипулятором

visible\_objects = [Shape('Cuboid{}'.format(i)) for i in range(10)]

nearby\_cubes = [obj for obj in visible\_objects if obj.get\_position()[2] < 0.1]

cube = None

for obj in nearby\_cubes:

if obj.get\_color()[1] >= 0.9:

cube = obj

break

if cube is not None:

#Целевая точка для перемещения куба

target\_pos = [-0.2, 0.2, 0.025]

#Выполнение операции Pick and Place

pick\_and\_place(cube, target\_pos)

Данный код представлен на моей GitHub: <https://github.com/soflom/arm>

Данная программа выполняет простой алгоритм:

1. Определяет функцию distance\_to\_object, которая измеряет расстояние между манипулятором и объектами в среде.
2. Определяет функцию pick\_and\_place, которая находит куб рядом с манипулятором, перемещает манипулятор над кубом, захватывает его, перемещает его в заданную точку, освобождает куб, и возвращает манипулятор в начальное положение.
3. Использует библиотеку PyRep для поиска куба рядом с манипулятором, определения целевой точки и выполнения операции Pick and Place.

С помощью датчика у манипулятора есть возможность обнаружить рядом объект типа кубоид и переместить его в заданную программой целевую точку. Подобный алгоритм подходит для целей по типу конвейерного производства или сортировки объектов (перемещение объектов разных типов в разные целевые точки). Данный алгоритм подойдет для немобильных манипуляторов.

# Обзор понимания объекта

Цель использования датчиков (например, зрения и осязания) — помочь во время взаимодействия между роботом манипулятором и объектом в пределах рабочего пространства робота. Глобальная и локальная информация может быть получена с датчиков. Глобальная информация предоставляется датчиками видения и используется для определения местоположения объектов в окружающей среде. Манипулятор может использовать глобальную информацию, чтобы избежать нежелательных препятствий или успешно переместить конечный эффектор к его цели. Между тем, локальная информация относится к тому, как робот взаимодействует с объектами в окружающей среде, и она предоставляется сенсорными датчиками. Локальная информация может быть использована роботом для манипулирования контактирующими объектами или изучения и извлечения свойств поверхности объектов. Глубокое обучение используется для обучения больших искусственных нейронных сетей.

# **Полносвязные слои нейронных сетей в машинном обучении**

Полносвязные слои являются одним из типов слоев нейронных сетей, которые используются в машинном обучении для решения различных задач. Они представляют собой распределенную сеть из взаимно связанных нейронов, где каждый нейрон получает на вход сигналы от всех нейронов предыдущего слоя и передает сигналы следующему слою в качестве своего выхода.

Функция полносвязного слоя заключается в том, чтобы принимать один или несколько входных сигналов, преобразовывать их и производить выходные сигналы. Эти слои могут быть использованы для классификации объектов, предсказания значений, обработки естественного языка и других задач машинного обучения.

Каждый нейрон полносвязного слоя вычисляет взвешенную сумму всех входных сигналов, используя свои весовые коэффициенты. Весовые коэффициенты определяют, насколько важна каждая связь между нейронами. Затем, вычисленное значение подвергается функции активации, которая определяет, какой уровень активации будет выдан на выходе из нейрона.

Функция активации является очень важным компонентом в работе полносвязного слоя. Она определяет, как нейрон будет реагировать на вычисленную взвешенную сумму, и тем самым, как сильно нейрон будет активирован. Существует несколько типов функция активации, но наиболее распространенными являются сигмоидальная, гиперболический тангенс (tanh), ReLU и Softmax.

Стоит отметить, что количество нейронов в полносвязном слое может быть довольно велико, и это часто приводит к проблемам с переобучением. Для решения этой проблемы, часто используется регуляризация, например, Dropout или L1/L2 регуляризация.

Во многих задачах машинного обучения полносвязный слой - это основной компонент нейронной сети. Однако, в некоторых случаях, использование полносвязных слоев может быть неэффективным, например, когда необходимо обрабатывать данные с неравномерной структурой. В таких случаях, часто используются другие типы слоев, такие как сверточные слои или рекуррентные слои.

В заключение, полносвязные слои - это важный элемент нейронных сетей. Они позволяют решать многие задачи машинного обучения, но требуют тщательной настройки параметров и регуляризации. Для более эффективного решения некоторых задач, могут быть необходимы другие типы слоев.

# DQN

Для работы с алгоритмом обучения с подкреплением понадобятся несколько других Python библиотек, а именно TensorFlow и Keras.

TensorFlow - это библиотека для численных вычислений, которая позволяет создавать и обучать нейронные сети и другие модели машинного обучения. Она была разработана командой Google Brain и предназначена для использования в задачах искусственного интеллекта, машинного обучения, обработки естественного языка, компьютерного зрения и других областях.

TensorFlow использует вычислительный граф для описания математических операций и трансформаций данных для создания моделей машинного обучения.

Достоинства TensorFlow включают гибкость, масштабируемость и объективность. Библиотека имеет множество встроенных функций и возможностей для создания, обучения и тестирования моделей. TensorFlow также является открытым исходным кодом, что позволяет сообществу разработчиков улучшать и расширять его возможности.

TensorFlow также имеет подробную документацию и обширную базу знаний, которая может помочь начинающим и опытным разработчикам в использовании библиотеки и решении сложных задач машинного обучения. Кроме того, на основе TensorFlow были разработаны такие инструменты, как Keras, TensorFlow.js и TFLite, которые расширяют функциональность библиотеки и делают ее доступнее для использования в различных приложениях.

Keras с версии 2.3 — это надстройка над библиотекой TensorFlow, которая нужна для машинного обучения. TensorFlow выполняет все низкоуровневые вычисления и преобразования и служит своеобразным движком, математическим ядром. Keras же управляет моделями, по которым проходят вычисления.

До версии 2.3Keras мог использовать в качестве движка вычислительные библиотеки Theano или CNTK. Но в новых версиях поддержка прекратилась, теперь библиотека работает только с TensorFlow.

Процесс настройки алгоритма DQN для манипулятора в среде V-REP связан с необходимостью использования библиотеки TensorFlow и глубинного обучения искусственных нейронных сетей.

В алгоритме DQN используется две нейронные сети: main и target. На каждом шаге эпизода на вход main-сети подаётся состояние окружения, и она вычисляет значения Q-функции для каждого действия в этом состоянии. Агент выбирает действие с наибольшим значением Q-функции и выполняет его, перейдя в новое состояние. Оценка этого действия записывается в буфер памяти.

После каждого шага обучения происходит обновление параметров main-сети. Для этого из буфера памяти случайным образом выбирается mini-batch, содержащий состояние, действие, награду и следующее состояние. Выполняя минимизацию функции потерь, значения целевой функции (target-сети), которая вычисляет значения Q-функции для следующего состояния, используются для обновления весов main-сети.

Таким образом, обучение DQN происходит в два этапа: на первом этапе с помощью main-сети вычисляются значения Q-функции; на втором этапе используются значения целевой функции и выполняется обучение сети. Эта процедура повторяется до тех пор, пока среднее значение функции потерь не достигнет заданного порога.

После обучения модели мы тестируем ее на задаче захвата объекта манипулятором. Мы используем обученную модель, чтобы позиционировать манипулятор и выполнить определенные действия, пока объект не будет захвачен.

Математически, DQN использует принцип Беллмана оптимальности для поиска оптимальной Q-функции:

Q\*(s,a) = E[ r + gamma \* max Q\*(s',a') | s,a ]

где

* s - состояние среды
* a - выбранное действие
* r - награда, полученная за выполнение действия a в состоянии s
* s' - новое состояние, в которое перешла среда после выполнения действия a в состоянии s
* a' - наилучшее действие, которое можно выполнить в состоянии s'
* gamma - коэффициент дисконтирования награды

Возможные состояния среды и выбор действий в каждом состоянии хранятся в памяти агента. DQN также использует технику реплея (experience replay), которая сохраняет опыт агента в буфере памяти и применяет мини-пакеты из случайных примеров опыта для обновления нейронной сети Q. Это помогает увеличить стабильность обучения и предотвратить "зацикливание" на определенных комбинациях состояний и действий.

Используя DQN, агент может обучаться в среде, позволяя ему находить оптимальную стратегию действий, максимизирующую сумму наград на протяжении всего времени выполнения задачи.

Алгоритм будет состоять из следующих шагов:

1. Импортируем необходимые библиотеки и классы.

2. Создаем класс DQNAgent для определения параметров нейронной сети и настройки алгоритма:

class DQNAgent:

def \_\_init\_\_(self, state\_size, action\_size):

self.state\_size = state\_size

self.action\_size = action\_size

self.memory = []

self.gamma = 0.95

self.epsilon = 1.0

self.epsilon\_min = 0.01

self.epsilon\_decay = 0.995

self.learning\_rate = 0.001

self.model = self.\_build\_model()

*Происходит определение класса DQNAgent, который будет использоваться для обучения алгоритмов Q-обучения и DQN.*

*В конструкторе класса определены переменные состояния и размер действия, память агента, параметры гамма, эпсилон, скорости обучения. Кроме того, модель DQN также создается, используя вспомогательный метод \_build\_model.*

def \_build\_model(self):

model = Sequential()

model.add(Dense(24, input\_dim=self.state\_size, activation='relu')) #

model.add(Dense(24, activation='relu'))

model.add(Dense(self.action\_size, activation='linear'))

model.compile(loss='mse', optimizer=Adam(lr=self.learning\_rate))

return model

*Объявление класса и метода \_build\_model (private метод класса) для создания нейронной сети.*

*Внутри метода объявляется модель Sequential(), которая описывает последовательную структуру нейронной сети.*

*Затем с помощью model.add() добавляются слои в модель. В данном случае добавлены три слоя Dense (полносвязных слоя) с разными параметрами:*

* *первый слой с 24 нейронами, принимающий на вход вектор размерности self.state\_size и использующий функцию активации ReLU*
* *второй слой также с 24 нейронами и функцией активации ReLU*
* *третий слой с self.action\_size выходов и активацией linear (функция активации, позволяющая получать значения на выходе без ограничений)*

*После добавления всех слоев модель компилируется с помощью model.compile(), указывая параметры критерия оптимизации (функция потерь - mse) и метода оптимизации (Adam с коэффициентом обучения self.learning\_rate).*

*В результате метод \_build\_model() возвращает созданную модель.*

def remember(self, state, action, reward, next\_state, done):

self.memory.append((state, action, reward, next\_state, done))

*Происходит определение метода remember в классе self, который добавляет в память терминалы, связанные с определенным состоянием (state), действием (action), наградой (reward), следующим состоянием (next\_state) и флагом окончания (done). Каждый терминал (5 элементов) представляет собой процессию, и все такие процессии добавляются в список (self.memory) для последующей обработки в обучении.*

def act(self, state):

if np.random.rand() <= self.epsilon:

return random.randrange(self.action\_size)

act\_values = self.model.predict(state)

return np.argmax(act\_values[0])

*Это метод act, который принимает на вход текущее состояние state, и возвращает выбранное действие. Он использует epsilon-greedy стратегию для выбора действия - с вероятностью epsilon будет произведен случайный выбор действия из заданного пространства действий action\_size, а с вероятностью 1-epsilon будет выбрано действие с наибольшим предсказанным значением (на основе текущей модели обучения), примененным к переданному состоянию state. Функция predict модели принимает входные данные и возвращает предсказание модели.*

def replay(self, batch\_size):

minibatch = random.sample(self.memory, batch\_size)

for state, action, reward, next\_state, done in minibatch:

target = reward

if not done:

target = (reward + self.gamma \*np.amax(self.model.predict(next\_state)[0]))

target\_f = self.model.predict(state)

target\_f[0][action] = target

self.model.fit(state, target\_f, epochs=1, verbose=0)

if self.epsilon > self.epsilon\_min:

self.epsilon \*= self.epsilon\_decay

Создаем функцию для получения состояния агента, которая будет считывать текущее состояние манипулятора и объекта (куба) из V-REP:

def get\_state(clientID, joint\_handles, obj\_handle):

\_, cube\_pos = vrep.simxGetObjectPosition(clientID, obj\_handle, -1, vrep.simx\_opmode\_blocking)

\_, joint\_angles = vrep.simxGetJointPosition(clientID, joint\_handles[0], vrep.simx\_opmode\_blocking)

state = np.concatenate((joint\_angles, cube\_pos))

return state.reshape(1, state\_size)

*Происходит определение функции get\_state, которая принимает три аргумента: clientID - идентификатор подключения к V-REP, joint\_handles - список идентификаторов управляющих сочленений робота, и obj\_handle - идентификатор объекта, в данном случае кубика.*

*Внутри функции get\_state вызываются две функции vrep.simxGetObjectPosition и vrep.simxGetJointPosition, которые взаимодействуют с V-REP и получают позицию кубика и углы поворота сочлениий. Затем полученные значения объединяются с помощью функции np.concatenate и формируется одномерный массив state.*

*В последней строке функция state.reshape(1, state\_size) изменяет форму массива state на матрицу размером 1xstate\_size. Функция возвращает эту матрицу.*

Создаем основной цикл для обучения агента, который будет выполнять следующие шаги.

Определяем начальные значения состояния агента и параметров:

vrep.simxFinish(-1)

clientID = vrep.simxStart('127.0.0.1', 19997, True, True, 5000, 5)

if clientID != -1:

print("Connected to remote API server")

else:

print("Connection failed")

state\_size = 4

action\_size = 4

agent = DQNAgent(state\_size, action\_size)

joint\_names = ['joint1', 'joint2', 'joint3', 'joint4']

joint\_handles = [vrep.simxGetObjectHandle(clientID, name, vrep.simx\_opmode\_blocking)[1] for name in joint\_names]

\_, obj\_handle = vrep.simxGetObjectHandle(clientID, 'cube', vrep.simx\_opmode\_blocking)

done = False

batch\_size = 32

Получаем текущее состояние агента и определяем следующее действие:

state = get\_state(clientID, joint\_handles, obj\_handle)

action = agent.act(state)

Выполняем действие и получаем награду:

if action == 0:

\_, \_ = vrep.simxSetJointTargetVelocity(clientID, joint\_handles[0], 0.5, vrep.simx\_opmode\_blocking)

elif action == 1:

\_, \_ = vrep.simxSetJointTargetVelocity(clientID, joint\_handles[0], -0.5, vrep.simx\_opmode\_blocking)

elif action == 2:

\_, \_ = vrep.simxSetJointTargetVelocity(clientID, joint\_handles[1], 0.5, vrep.simx\_opmode\_blocking)

elif action == 3:

\_, \_ = vrep.simxSetJointTargetVelocity(clientID, joint\_handles[1], -0.5, vrep.simx\_opmode\_blocking)

\_, \_, \_ = vrep.simxGetPingTime(clientID)

time.sleep(0.01)

next\_state = get\_state(clientID, joint\_handles, obj\_handle)

reward = 0

if all(np.abs(next\_state[0, -3:]) < 0.05):

reward = 10

done = True

agent.remember(state, action, reward, next\_state, done)

state = next\_state

if len(agent.memory) > batch\_size:

agent.replay(batch\_size)

Уменьшаем значение параметра epsilon для уменьшения вероятности случайного действия:

if agent.epsilon > agent.epsilon\_min:

agent.epsilon \*= agent.epsilon\_decay

Если объект был захвачен, заканчиваем обучение:

if done:

print("Cube is captured!")

break

В данном алгоритме DQN используется нейронную сеть с тремя полносвязными слоями для предсказания оптимального действия агента в заданном состоянии. Мы также использовали хранилище для сохранения результатов прошлых действий, чтобы обучать нейронную сеть на их основе. Веса нейронной сети обновлялись после каждого выполнения действия, чтобы аппроксимировать оптимальную функцию ценности.

# Заключение

Цель данного исследования заключалась в разработке эффективного алгоритма управления манипулятором на основе DQN,

В качестве входных данных для исследования выступали параметры манипулятора, такие как размеры, масса, количество звеньев, а также информация о задачах, которые должен выполнять манипулятор. На выходе получилось оптимальное управление манипулятором для выполнения конкретной задачи.

В рамках данной работы были выполнены поставленные задачи:

1. Исследована литература по машинному обучению.
2. Исследованы существующие алгоритмы обучения с подкреплением.
3. Выбрано и настроено ПО для реализации задачи.
4. Проведен анализ работы с манипулятором в выбранной среде и разработаны алгоритмы для его использования

# Библиография

1. Marwan Qaid Mohammed, L.C. Kwek, Shing Chyi Chua. Review of Deep Reinforcement Learning-Based Object Grasping: Techniques, Open Challenges, and Recommendations //
2. Marco Bertini. Open source column: deep learning with Keras //
3. Саттон Р. С., Барто Э. Дж. Обучение с подкреплением: Введение. 2-е изд. / пер. с англ. А. А. Слинкина. – М.: ДМК Пресс, 2020. – 552 с. //
4. Official Developer guides from Keras. Available at: <https://keras.io/guides/> //
5. PyRep — Jan 21, 2023 //
6. Лаура Г., Лун К.В. Глубокое обучение с подкреплением: теория и практика на языке Python. — СПб.: Питер, 2022. — 416 с.: ил. — (Серия «Библиотека программиста»). //