

МИНОБРНАУКИ РОССИИ

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Казанский национальный исследовательский технологический университет»**

**(ФГБОУ ВО «КНИТУ»)**

Кафедра Интеллектуальных систем и управления информационными ресурсами

Направление специальность: 01.03.02 Прикладная математика и информатика

Тема курсового проекта по дисциплине «Машинное обучение»: Обучение нейронной сети играть в Pac-Man.

**КУРСОВОЙ ПРОЕКТ**

Заведующий кафедрой\_\_\_ Заведующий кафедры ИСУИР Герасимов А.В.

Руководитель работы\_\_\_\_\_\_\_\_ Доцент кафедры ИСУИР Тартыгашева А.М.

Студент Андрианов А.В.

Казань 2023 г

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 2](#_Toc153227353)

[ЗАДАНИЕ 3](#_Toc153227354)

[ЛИСТ НОРМОКОНТРОЛЕРА 4](#_Toc153227355)

[1. СРЕДА PAC-MAN 5](#_Toc153227356)

[2. ГРАДИЕНТНЫЙ СПУСК 7](#_Toc153227357)

[3. POLICY GRADIENT 8](#_Toc153227358)

[4. ПОЛНОСВЯЗНЫЕ НЕЙРОННЫЕ СЕТИ 11](#_Toc153227359)

[5. СВЕРТОЧНАЯ НЕЙРОННАЯ СЕТЬ 14](#_Toc153227360)

[6. АРХИТЕКТУРА НЕЙРОННОЙ СЕТИ 16](#_Toc153227361)

[7. БИБЛИОТЕКИ 19](#_Toc153227362)

[8. РЕАЛИЗАЦИЯ КОДА 20](#_Toc153227363)

[9. ГРАФИКИ И ОБУЧЕНИЕ 23](#_Toc153227364)

[ВЫВОД 24](#_Toc153227365)

[СПИСОК ЛИТЕРАТУРЫ 25](#_Toc153227366)

[ПРИЛОЖЕНИЕ 26](#_Toc153227367)

# ВВЕДЕНИЕ

В современном мире искусственный интеллект (AI) играет важную роль, проникая во все сферы нашей жизни. Для решения сложных задач, которые не могут быть решены традиционными методами машинного обучения, существует обучение с подкреплением. Обучение с подкреплением – метод обучения нейронных сетей для нахождения оптимального решения сложных задач. Обучение основано на взаимодействии агента со средой. В качестве агента выступает нейросеть. В процессе обучения нейросеть получает награду за свои действия и информацию о состоянии среды. Награда нужна для корректировки нейросетью своих действий. Нейросеть к концу обучения должна достичь оптимального пути решения задачи, в результате которого получает максимальное вознаграждение. В курсовой работе нейросеть обучается взаимодействовать в среде Pac-man. Pac-Man – это классическая аркадная игра, выпущенная в 1980 году компанией Namco. Задача агента собрать все точки в лабиринте и избежать столкновения с привидениями. Для реализации проекта будет использован язык Python. В качестве среды для проведения экспериментов будет использована библиотека Gym. Нейросеть будет написана с применением библиотеки PyTorch. Для обучения агента будет использован метод Policy Gradient. Данный метод основан на том, чтобы оптимизировать политику агента для получения максимального выигрыша. Политика определяет действия агента из какого-то состояния среды. Целью данной курсовой работы является изучение процесса обучения нейросети игре Pac-Man и анализ полученных результатов. Для достижения данной цели необходимо изучить основные принципы работы полносвязных, свёрточных слоёв нейросети, реализовать алгоритм обучения, подобрать эмпирическим путём архитектуру и гиперпараметры нейросети.

ЗАДАНИЕ

На курсовую работу (проект) студенту кафедры: Интеллектуальных систем и управления информационными ресурсами Андрианову Алексею Вениаминовичу\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Тема работы (проекта): курсовой проект по дисциплине «Машинное обучение». Тема проекта: «Обучение нейронной сети играть в Pac-Man»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Исходные данные к проекту: предметная область \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Содержание расчетно - пояснительной записки (включая перечень подлежащих разработке вопросов, включая вопросы стандартизации и контроля качества): Введение, Задание, Лист нормоконтролера, Раздел 1. Предметная область, Раздел 2. Теоретическая часть, 2.1 Выбор языка программирования, 2.2 Выбор библиотек, Раздел 3. Разработка приложения, 3.1 Установка библиотеки, 3.2 Чтение изображения, 3.3 Обнаружение контуров, 3.4 Функция поиска геометрической фигуры, 3.5 Вывод изображения 4. Результаты, Заключение, Список литературы, Приложение \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Перечень графического материала (схемной документации): презентационный материал\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Консультанты по проекту (с указанием относящихся к ним разделов): Тартыгашева А.М.. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата выдачи задания «\_\_15\_\_\_» \_\_\_\_\_\_\_сентября\_\_\_\_\_\_\_\_\_\_\_\_2023 г.

Руководитель проекта \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Тартыгашева А.М.

ЛИСТ НОРМОКОНТРОЛЕРА

1. Лист является обязательным приложением к пояснительной записке дипломного (курсового) проекта.
2. Нормоконтролер имеет право возвращать документацию без рассмотрения в случаях:

-нарушения установленной комплектности,

-отсутствия обязательных подписей,

-нечеткого выполнения текстового и графического материала.

1. Устранение ошибок, указанных нормоконтролером, обязательно.

**ПЕРЕЧЕНЬ**

замечаний и предложений нормоконтролера по дипломному (курсовому) проекту, студента

4301-22, Андрианов Алексей Вениаминович

(группа, инициалы, фамилия)

|  |  |  |
| --- | --- | --- |
| Лист  (страница) | Условное обозна-чение (код ошибок) | Содержание замечаний и предложений со ссылкой на нормативный документ, стандарт или типовую документацию |
|  |  |  |

Дата\_\_22 декабря 2023 г.\_\_ Нормоконтролер\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# СРЕДА PAC-MAN

В процессе игры агент нейросеть-агент управляет персонажем по имени Pac-man, который должен собирать точки и избегать столкновения с привидениями. Процесс, как агент взаимодействует с окружение, в виде блок-схемы представлен на рисунке 1. Агент может совершать действия в 9 направлениях: бездействие, вверх, вправо, влево, вниз, вверх-вправо, вверх-влево, вниз-вправо, вниз-влево. За собранную точку агент получает награду в размере 10 единиц. Есть особые точки, съев которые, агент может начать поглощать призраков. За особые точки агент получает награду в 15 единиц. За съеденного призрака агент получает награду в 5 единиц. Если агент столкнулся с призраком, то получает штраф в размере -20 единиц. За “пустые” действия агент получает 0 единиц. “Пустым” действием считается ход, совершенный в ячейку, в которой нет ни точки, ни призрака, или в стену игрового мира. У агента в распоряжении есть 2 жизни, если жизни заканчиваются, то агент проигрывает, и игра начинается заново. Эпизодом игры считается последовательностью действий, совершаемых с начала игры до терминального состояния.

Агент, совершая действия, среда возвращает состояние, которое представлено в виде RGB-изображения с шириной 210 пикселей, высотой 160 пикселей, глубиной 3. Пример необработанного изображения представлен на рисунке 2. На вход для принятия решения в нейросеть подаётся предварительно обработанное изображение. Изображение обрезается до размеров ширины, высоты, глубины 86, 80, 1 соответственно. Пример обработанного изображения представлен на рисунке 3. Для динамики игрового процесса нейросети передаётся информация о двух предыдущих состояниях игры, благодаря чему нейросеть получает возможность принимать решение на основе контекста и предыдущего опыта. Таким образом, входные данные будут иметь размеры ширины 86 пикселей, высоты 80, глубины 3.

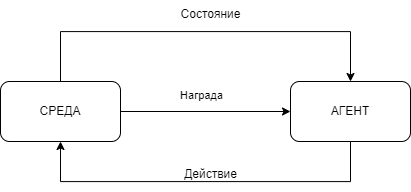


Рисунок 1 – Процесс взаимодействия агента со средой

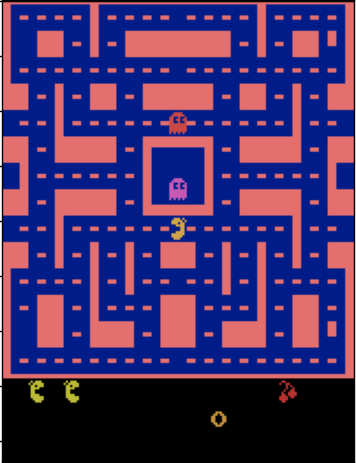


Рисунок 2 – Необработанное изображение

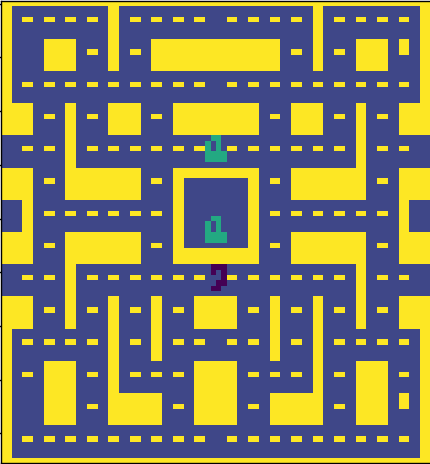


Рисунок 3 – Обработанное изображение

# ГРАДИЕНТНЫЙ СПУСК

Градиентный спуск является ключевым понятием в обучении с подкреплением. Агент взаимодействует с окружающей средой с целью максимизации награды. На каждой итерации агент принимает решение, какое действие совершить из данного состояния, и получает награду. Градиентный спуск используется для обновления весов нейронной сети агента. То есть задача оптимизации или обучения нейронной сети заключается в нахождении оптимальных весов, которые максимизируют целевую функцию. Оптимизация проходит с помощью градиентного спуска. Формула градиентного спуска представлена на рисунке 4. Градиентный спуск – это итерационный алгоритм для нахождения минимума или максимума функции. Суть данного алгоритма заключается в том, что на каждой итерации спуск идёт в наибольшем убывании функции, то есть в направлении антиградиента. Таким образом, после достаточно большого количества итераций алгоритм должен сойтись в точке оптимума.

Рисунок 4 - Формула обновления весов с помощью градиентного спуска

Гиперпараметр λ регулирует скорость, шаг обучения нейронной сети или скорость схождения алгоритма. - градиент целевой функции, либо функция ошибки по весам нейронной сети. Функция ошибки должна высчитывать то, как нейронная сеть хороша справляется с задачей. У градиентного спуска, как и у других методов оптимизации, существует проблема застревания в локальных оптимумах. Это проблема решается с запуском заново алгоритма обучения с новыми начальными весами нейронной сети, либо сделать скорость обучения больше, чтобы спуск мог проскочить локальный оптимум, но тут тоже не надо делать шаг слишком большой.

# POLICY GRADIENT

Методы обучения с подкреплением делятся на те, которые оптимизируют функцию стоимости и функцию политики. Метод оптимизирующие функцию стоимости направлены на предсказание предполагаемого будущего выигрыша действия, совершенного из текущего состояния среды. Оптимизация функции политики направлена сразу на предсказание оптимального действия для агента в заданной окружающей среде. Политика представлена нейронной сетью. Самый простой и распространённый метод оптимизации политики является Policy Gradient. Рассмотрим математическую основу данного алгоритма обучения.

Общее между двумя подходами то, что направлены на получении максимальной кумулятивной дисконтированной награды. На более техническом уровне означает, что алгоритм должен найти такие веса нейросети, которые дают максимальную кумулятивную дисконтированную награду. Назовём её целевой функцией. Формула представлена на рисунке 5.

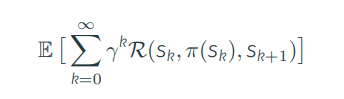


Рисунок 5 - Кумулятивная дисконтированная награда, γ - коэффициент дисконтирования, R - награда, S - состояние, π - политика, k - шаг.

Чтобы агент имел представление о будущей награде введем понятие дисконтируемая награда. Это методика, которая учитывает временную стоимость награды, то есть насколько важна награда в будущем по сравнению с наградой в настоящем. Действия, которые совершил агент в конце эпизода, будут иметь меньшее значение, чем в текущий момент. Коэффициент дисконтирование принимает значения от нуля до единицы. Преимущественно берётся значение ближе к единице. Таким образом, оптимизация заключается в том, чтобы предполагаемая суммарная награда увеличивалась в ходе обучения.

Политика принимает на вход состояние среды и выдаёт распределение вероятности действий. Формула показана на рисунке 6. В среде Pac-Man пространство действия дискретное и имеет 9 возможных действий. Соответственно нейронная сеть будет возвращать вероятность для каждого действия. Действия образуют полную группу по теории вероятности, следовательно, сумма вероятностей равна единице. С каждым шагом оптимизации нейронная сеть должна возвращать оптимальное распределение вероятностей, которое приносит наибольший выигрыш. Выбор действия из распределения вероятности частично решает проблему с исследованием и эксплуатацией среды, так как действия имеют стохастическую основу. Проблема с исследованием и эксплуатацией среды заключается в том, что агент мало исследует незнакомую часть среды и стремится переобучиться на уже знакомой части среды. Методы оптимизации политики частично решают данную проблему.

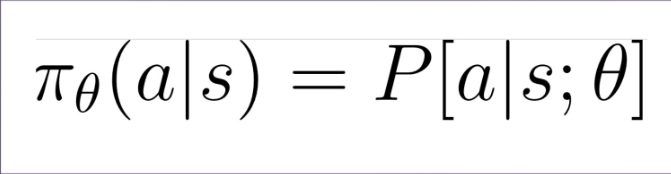


Рисунок 6 - Политика

Что же представляет из себя целевая функция? На рисунке 7 представлена формула целевой функции.

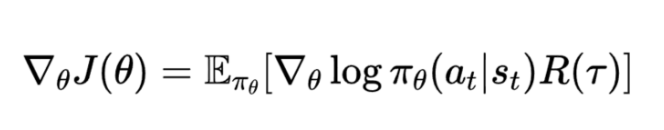


Рисунок 8 - Целевая функция

Агент совершает за эпизод N действий и назовем эту последовательность траекторией τ. Все совершаемые действия и полученные награды накапливаются для расчета целевой функции. После эпизода по наградам рассчитывается дисконтированная награда R для каждого шага. Чтобы целевая функция была дифференцируемой, от вероятности, которую возвращает политика, берется натуральный логарифм. Можно интерпретировать данную функцию следующим образом: если из состояния S совершено действие a и это действие привело к большой награде, то надо увеличить вероятность наступление данного действие, иначе снизить наступление данного действия в будущем, потому что действие не ведет к большой награде.

Можем изменить немного целевую функцию на так называемую градиент политики Монте-Карло. В данной политике вместо математического ожидания используется суммарная итоговая оценка за эпизод игры. На рисунке 8 представлена формула.

Рисунок 8 – Градиент политики Монте – Карло

# ПОЛНОСВЯЗНЫЕ НЕЙРОННЫЕ СЕТИ

Главное единицей нейронной сети является нейрон. Нейрон представляет собой математическую модель, у которой есть несколько входов и один выход. Каждый вход x оснащён своим весом ω. Сумма произведения входных признаков и весов X передаётся в функцию активации f(X). Пример как выглядит схематично нейрон показан на рисунке 12.

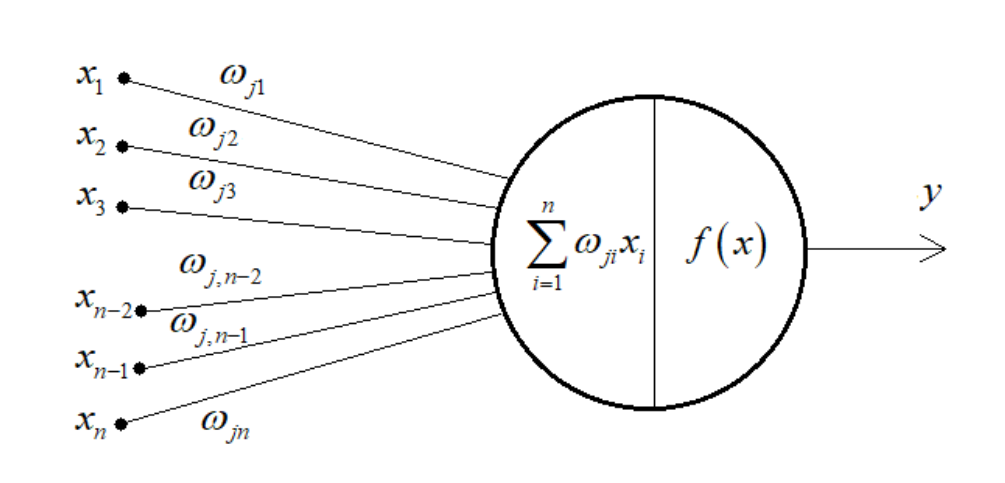


Рисунок 12 – Нейрон

Одного нейрона будет мало для построения полноценной мощной сети. Полносвязная нейронная сеть – это тип нейронной сети, где каждый нейрон предыдущего слоя связан с каждым нейроном следующего слоя. Первый слой в данной сети идёт входным слоем и представляет собой входные данные. Из входного слоя переходит в скрытые слои, которых может быть несколько штук, где каждый полносвязный слой со своим количеством нейронов и функцией активацией. Пример полносвязной нейронной сети показан на рисунке 13. Если количество нейронов увеличивается из слоя в слой, то нейронная сеть способна найти более сложные структуры и зависимости в данных, но сильное усложнение и избыточность архитектуры ведёт к переобучению нейронной сети.

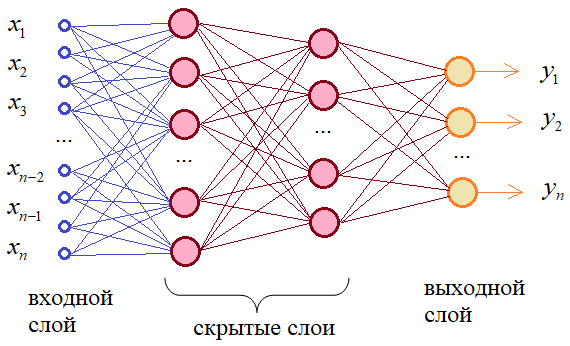


Рисунок 13 - Полносвязная нейронная сеть

Есть разные виды функций активации, в том числе: линейная, ReLu, Sigmoid, SoftMax.

Есть разные виды функций активации, в том числе:

1. линейная функция:
2. ReLu: *f(x) = max(x, 0)*
3. Sigmoid:
4. SoftMax: , где *i* - номер класса, n- количество классов.

Важной особенностью является использование функции активации, которая позволяет нейронам работать нелинейно и обрабатывать сложные зависимости. Чтобы последующая оптимизация весов была проще, используют ReLu, так как легко дифференцируема и не вызывает затухание градиента в отличие от sigmoid-функции. Линейная функция используется для задач регрессии, функция также не вызывает затухание градиента. SoftMax функция используется для представления выхода слоя нейронной сети в виде распределения вероятностей. Данная функция используется для классификации не пересекающихся классов.

# СВЕРТОЧНАЯ НЕЙРОННАЯ СЕТЬ

Сверточные нейронные сети (СНС) – это тип нейронные сетей, используемых в области компьютерного зрения и обработки 2D-массивов. Сверточная сеть позволяет извлекать признаки из входных данных, наподобие, формы, фигуры. Для уменьшения размера изображения используются Pooling – слой. Также применяется слои активации для нахождения нелинейности. После сверточных нейронных сетей применяются полносвязные нейронные сети для задачи классификации или регрессии.

Основная операция СНС является свертка и ядро фильтр. Ядро представляет из себя матрицу NxN размера, где N гиперпараметр. Для примера возьмём N=3. Пример как выглядит ядро фильтр показан на рисунке 14. Параметр ω0 является свободным и предназначен для смещения и улучшения результатов оптимизации. Параметры ω подбираются в ходе обучения нейронной сети. Ядро фильтр пропускает через свою матрицу часть входного изображения NxN размера. Делает сдвиг по входному изображению X и пропускает через фильтр следующую часть. Формула операции свертки показана на рисунке 15.

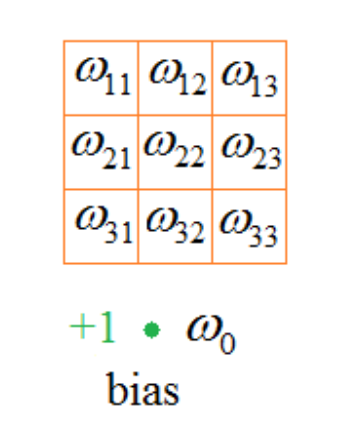


Рисунок 14 – Ядро фильтр

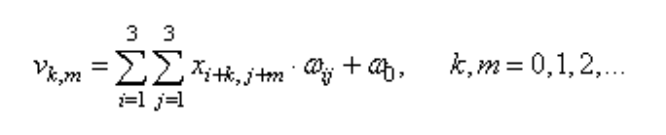


Рисунок 15 – Операция свертки

Для изменения размера изображения применяется Pooling слой. Для Pooling слоя задаётся гиперпараметр размер окна, которое будет пробегаться по изображению, и шага. На каждом шаге значения, которые попали в окно, будет производиться одна из следующих операций:

1. MaxPooling – наибольшее значение
2. MinPooling – наименьшее значение
3. AveragePooling – среднее значение

Таким образом, операция Pooling слоя позволяет отсеять ненужные признаки из изображения, позволяя углубиться в поиск более важных признаков.

# АРХИТЕКТУРА НЕЙРОННОЙ СЕТИ

В данной главе рассматривается архитектура нейронной сети, использованная в процессе обучения искусственного агента в видеоигре Pac-Man. Для этого используется нейронная сеть, состоящая из двух сверточных слоев и плоского (Flatten) слоя. Размер ядра – это параметр сверточного слоя нейронной сети, определяющий размер области входного сигнала, которая участвует в вычислении одного значения выходного сигнала. Шаг – это параметр сверточного слоя, определяющий расстояние между соседними областями входного сигнала. Первый сверточный слой обладает размером ядра 4 и шагом 2, второй слой– размером ядра 3 и шагом 1. В обоих слоях используется функцию активации ReLu. После прохождения первого сверточного слоя, применяется MaxPooling с параметрами ядра 3 и шага 3. Аналогично, после второго сверточного слоя тоже используется MaxPooling, но уже со значениями ядра 3 и шага 2. Это позволяет уменьшить размерность входных данных. Чтобы преобразовать выходное изображение в одномерную матрицу, применяется Flatten слой. На вход нейронная сеть получает матрицу размерности 86x80x3 после прохождения матрицы через сверточные слои и Flatten слой изображение преобразуется в 960×1 матрицу. Сверточные слои позволяют извлекать из входных изображений наиболее значимые признаки, такие как края, углы и другие геометрические примитивы. Это их основное преимущество то, что они способны автоматически обнаруживать важные признаки в данных, без необходимости ручного кодирования этих признаков. После извлечения признаков из входных данных, признаки передаются в полносвязные слои. В работе используется 1 скрытый полносвязный слой с 128 нейронами. Полносвязные слои используются для нахождения закономерностей между признаками и для принятия дальнейшего решения. В качестве активационной функции для скрытых слоев используется ReLu, так как легко дифференцируема и показывает хорошие результаты при обучении. Для выходного слоя используется активационная функции Softmax. Данная функция представляет выход в виде распределения вероятностей. Так как нейронной сети должна предсказывать какое из 9 действие в данном состоянии принять, удобнее будет выход нейронной сети сделать в 9 нейронов, где каждый нейрон будет представлять вероятность. Значение вероятности ближе к 1 означает, что действие приводит к максимальной награде, а значение ближе к 0, наоборот. В процессе обучения нейросеть должна корректировать веса, в результате которых распределение вероятностей всё больше подстраиваются для получения максимального выигрыша. Инициализация архитектуры нейронной сети представлена на рисунке 9.



Рисунок 9 – Инициализация нейронной сети

Важным аспектом является начальная инициализация весов нейронной сети, так как это влияет на сходимость алгоритма обучения. Начальные веса определяют начальные позиции, откуда градиентный спуск начнёт обновлять веса, и если веса инициализированы близко к точке оптимуме, то алгоритм быстро сойдётся. Если начальные веса будут находиться на плато, то веса будут слабо изменяться и обучение займёт много времени. Для активационной функции ReLu существует нормальное Кайминг инициализация, где веса представляют из нормального распределения N (0, σ2). Формула данного метода инициализации представлен на рисунке 10.

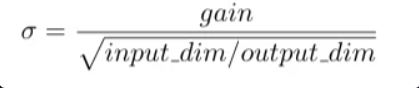


Рисунок 10 – Нормальное распределение Кайминга, где gain – коэффициент масштабирования

Для активационной функции SoftMax применялось равномерное распределение Ксавье, где веса также выбираются из нормального распределения N(0, σ2). Формула данного метода инициализации представлен на рисунке 11.

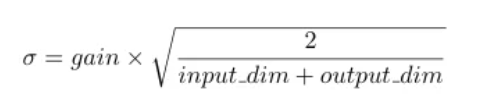


Рисунок 11 – Нормальное распределение Ксавье, где gain – коэффициент масштабирования

# БИБЛИОТЕКИ

Использовались основные библиотеки для написания программы:

Numpy – библиотека для работы с числовыми данными, массивами.

Torch – библиотека для создания, обучения нейронных сетей и работы с численными массивами.

matplotlib.pyplot — Библиотека для визуализации данных.

Torch.optim – оптимизатор Adam, используемый для обучения нейронных сетей.

Gym – библиотека, предоставляющая готовые среды для обучения, включая Pac-Man.

Collections – библиотека для работы с коллекциями.

Torch.distributions - библиотека вероятностных распределений.

Torch.nn – библиотека содержит слои нейронных сетей и другие вспомогательные классы.

1. РЕАЛИЗАЦИЯ КОДА

Для обучения нейронной сети нужны данные, которые агент собирает в процессе игры. Процесс игры завершается, если агент совершил максимальное количество шагов или игра была завершена по причине проигрыша. Состояние среды проходит предобработку и заключается в том, что состояние обрезается 80 пикселей на 86 пикселей. В нейросеть для прогноза действия передаётся ещё два предыдущих состояния, что в итоге конечное входное состояние матрица 3x80x86 пикселей. Код предобработки состояния показан на рисунке 16. В процессе игры собираются данные награда за действия, логарифм вероятности действия, а также достигла ли игра терминального состояния. После сбора идёт предобработка данных и обучение. В предобработку входит расчёт дисконтированной награды. Функция расчета награды показана на рисунке 17.

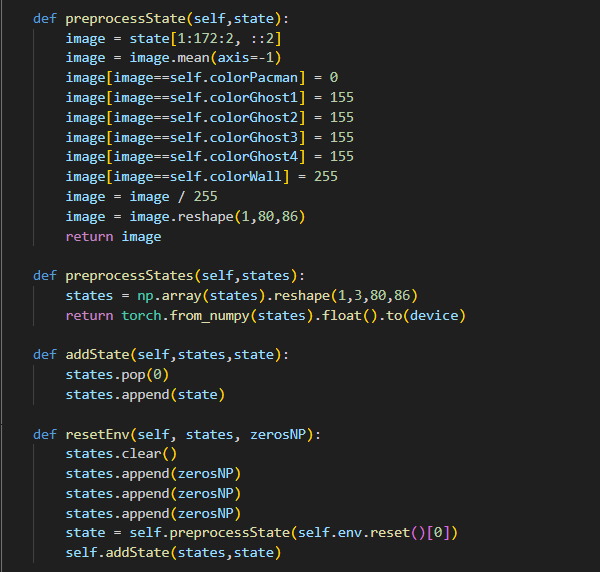


Рисунок 16 – Предобработка состояния

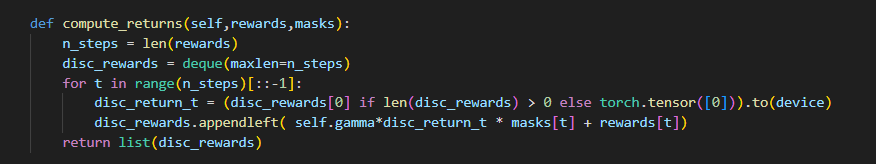


Рисунок 17 – Функция дисконтированной награды

Так же для более стабильного обучения дисконтированная награда стандартизируется, то есть распределение награды становиться нормальным с математическим ожиданием ноль и стандартным отклонением 1. На рисунке 18 показан код стандартизации.



Рисунок 18 – Стандартизация награды

Высчитывается функция ошибки по методу Монте-Карло. Entropy – это мера хаоса, которая отражает неопределенность. Добавляется, чтобы агент больше совершал ошибки и исследовал среду. Функция ошибки показан на рисунке 19.

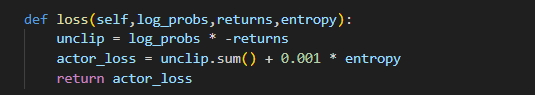


Рисунок 19 – Функция ошибки

По ошибке вычисляется градиент с помощью метода обратного распространения ошибки, и оптимизатор делает шаг, используя вычисленный градиент. Шаг обучения показан на рисунке 20.

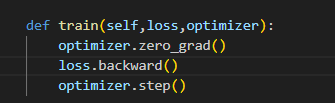


Рисунок 20 – Вычисляется градиент и делается шаг оптимизации

# ГРАФИКИ И ОБУЧЕНИЕ

На рисунке 21 показаны суммарные награды, которые агент получал за каждый сыгранный эпизод. На графике видно, что агент за первые 700 эпизодов увеличивает получаемую награду, но после изменения в награде не наблюдаются, агент вышел на среднюю 600 награду и выше не поднимается.

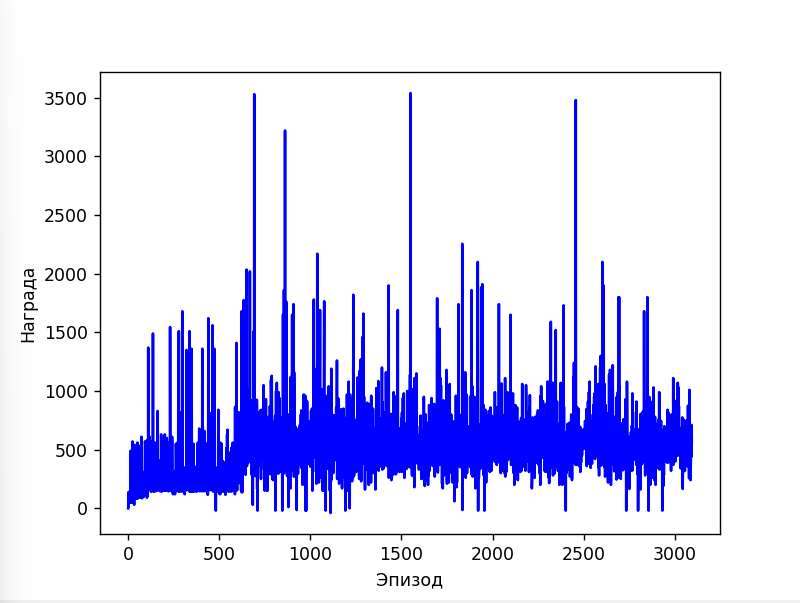


Рисунок 21 – График наград

# ВЫВОД

В данной курсовой работе была реализована нейронная сеть для игры Pac-Man. В процессе обучения наблюдалась склонность нейронной сети к переобучению, повторению одного и того же паттерна поведения без возможности к обобщению. Не было реализовано параллелизма для ускорения процесса обучению, что в разы сократило бы время обучения. В ходе работы было изучено много информации и закреплены навыки работы с библиотеками python, таких как PyTorch, GYM. В дальнейших проектах надо заранее предусмотреть параллелизм и сбор данных для отчётности про проделанной работе.

# СПИСОК ЛИТЕРАТУРЫ

1. Васильев, А. Н. Принципы и техника нейросетевого моделирования / А.Н. Васильев, Д.А. Тархов. - Москва: СПб. [и др.] : Питер, 2022. - 218 c.
2. Визильтер, Юрий Алексеевич Обработка и анализ цифровых изображений с примерами на LabVIEW и IMAQ Vision / Визильтер Юрий Алексеевич. - М.: ДМК Пресс, 2023. - 837 c.
3. Глория, Буэно Гарсия Обработка изображений с помощью OpenCV / Глория Буэно Гарсия. - М.: ДМК Пресс, 2022. - 489 c.
4. Лутц, Марк Программирование на Python. Том 2: моногр. / Марк Лутц. - М.: Символ-плюс, 2023. - 992 c.
5. Таганов, Александр Иванович Нейросетевые системы искусственного интеллекта в задачах обработки изображений / Таганов Александр Иванович. - М.: Горячая линия - Телеком, 2022. - 660 c.
6. Хайкин, Саймон Нейронные сети. Полный курс / Саймон Хайкин. - М.: Вильямс, 2022. - 107 c "Машинное обучение. Современные методы и технологии" А. Г. Абрамян (2019) — стр. 1-520
7. Пушкин А. С. Нейронные сети / А. С. Пушкин. – Москва : Эксмо, 2022. – 300 с.
8. Аггарвал Ч. Нейронные сети и глубокое обучение: учебный курс.: Пер. с англ. / Ч. Аггарвал. – СПб. : Диалектика, 2020. – 752 с. – ISBN 978-5-907203-01-3
9. "Машинное обучение. Современные методы и технологии" А. Г. Левин (2019) — стр. 1-520 Эти ресурсы предоставят вам хорошее понимание основных концепций и техник, связанных с нейронными сетями и машинным обучением.
10. Чару, А. Нейронные сети и глубокое обучение: учебный курс. / А. Чару,. – Санкт-Петербург : Диалектика, 2020. – 752 с.

# ПРИЛОЖЕНИЕ

Листинг программы

import numpy as np

import torch,time

import torch.nn as nn

import torch.optim as optim

from torch.distributions import Categorical

from collections import deque

import gym

import matplotlib.pyplot as plt

device = torch.device("cuda:0" if torch.cuda.is\_available() else "cpu")

# device = torch.device('cpu')

class Policy(nn.Module):

    def \_\_init\_\_(self,eps=0.2):

        super(Policy, self).\_\_init\_\_()

        self.eps = eps

        self.policy = nn.Sequential(

            nn.Sequential(nn.Conv2d(kernel\_size=4,stride=2, in\_channels=3,out\_channels=64,padding="same",device=device),nn.ReLU(),nn.MaxPool2d(kernel\_size=3,stride=3)),

            nn.Sequential(nn.Conv2d(kernel\_size=3,stride=1, in\_channels=64,out\_channels=128,padding="same",device=device),nn.ReLU(),nn.MaxPool2d(kernel\_size=2,stride=2)),

            nn.Flatten(),

            nn.Sequential(nn.Linear(in\_features=128\*6\*5, out\_features=256,device=device), nn.Dropout(p=0.3),nn.ReLU()),

            nn.Sequential(nn.Linear(in\_features=256, out\_features=9,device=device), nn.Softmax(dim=-1))

        )

        nn.init.kaiming\_normal\_(self.policy[0][0].weight,mode='fan\_out',nonlinearity='relu')

        nn.init.kaiming\_normal\_(self.policy[1][0].weight,mode='fan\_out',nonlinearity='relu')

        nn.init.kaiming\_normal\_(self.policy[3][0].weight,mode='fan\_out',nonlinearity='relu'),

        nn.init.xavier\_uniform\_(self.policy[4][0].weight)

    def forward(self, states):

        probs = self.policy(states)

        dist = Categorical(probs)

        actions = dist.sample()

        return actions, dist.log\_prob(actions),dist.entropy().mean()

    def loss(self,log\_probs,returns,entropy):

        unclip = log\_probs \* -returns

        actor\_loss = unclip.sum() - 0.001 \* entropy

        return actor\_loss

    def train(self,loss,optimizer):

        optimizer.zero\_grad()

        loss.backward()

        optimizer.step()

class PolicyGradient:

    def \_\_init\_\_(self,lr = 1e-4,gamma=0.9,render\_mode='rgb\_array',weight\_decay = None):

        self.gamma=gamma

        self.policy = Policy().to(device)

        self.optimizer = optim.Adam(self.policy.parameters(), lr=lr,weight\_decay=weight\_decay)

        # self.scheldure = optim.lr\_scheduler.StepLR(self.optimizer,step\_size=300,gamma=0.8)

        print(sum([p.numel() for p in self.policy.parameters() if p.requires\_grad]))

        # self.backup\_model = copy.deepcopy(self.policy)

        self.env = gym.make('MsPacman-v4',render\_mode=render\_mode)

        self.colorPacman = np.array([210, 164, 74]).mean()

        self.colorWall = np.array([228,111,111]).mean()

        self.colorGhost1 = np.array([200, 72, 72]).mean()

        self.colorGhost2 = np.array([198, 89, 179]).mean()

        self.colorGhost3 = np.array([180, 122, 48]).mean()

        self.colorGhost4 = np.array([84, 184, 153]).mean()

        self.statictics = Statictics()

    def writeModel(self):

        state  = {

            'model' : self.policy.state\_dict(),

            'optimizer': self.optimizer.state\_dict()

        }

        torch.save(state, 'Policy/Model.pt')

    def readModel(self):

        try:

            state = torch.load('Policy/Model.pt')

            self.policy.load\_state\_dict(state['model'])

            self.optimizer.load\_state\_dict(state['optimizer'])

            print(sum([p.numel() for p in self.policy.parameters() if p.requires\_grad]))

            print(self.policy.policy[5][0].weight)

        except Exception as e:

            print(e)

    def compute\_returns(self,rewards,masks):

        n\_steps = len(rewards)

        disc\_rewards = deque(maxlen=n\_steps)

        for t in range(n\_steps)[::-1]:

            disc\_return\_t = (disc\_rewards[0] if len(disc\_rewards) > 0 else torch.tensor([0])).to(device)

            disc\_rewards.appendleft( self.gamma\*disc\_return\_t \* masks[t] + rewards[t])

        return list(disc\_rewards)

    def preprocessState(self,state):

        image = state[1:172:2, ::2]

        image = image.mean(axis=-1)

        image[image==self.colorPacman] = 0

        image[image==self.colorGhost1] = 155

        image[image==self.colorGhost2] = 155

        image[image==self.colorGhost3] = 155

        image[image==self.colorGhost4] = 155

        image[image==self.colorWall] = 255

        image = image / 255

        image = image.reshape(1,80,86)

        return image

    def preprocessStates(self,states):

        states = np.array(states).reshape(1,3,80,86)

        return torch.from\_numpy(states).float().to(device)

    def addState(self,states,state):

        states.pop(0)

        states.append(state)

    def resetEnv(self, states, zerosNP):

        states.clear()

        states.append(zerosNP)

        states.append(zerosNP)

        states.append(zerosNP)

        state = self.preprocessState(self.env.reset()[0])

        self.addState(states,state)

    def train(self, n\_training\_episodes,n\_steps,skip\_episodes):

        self.readModel()

        self.statictics.readTotalReward()

        try:

            terminated = False

            states = []

            # ratio = []

            log\_probs = []

            masks = []

            rewards   = []

            zerosNP = np.zeros((1,80,86))

            self.resetEnv(states,zerosNP)

            lives = 3

            for i\_episode in range(1, n\_training\_episodes+1):

                entropy = 0

                reward\_episod = 0

                # start = time.time()

                if terminated:

                    lives = 3

                    self.resetEnv(states, zerosNP)

                for step in range(1, n\_steps):

                    pre\_states = self.preprocessStates(states)

                    action,log\_prob,ent = self.policy.forward(pre\_states)

                    # action2,log\_prob\_old, ent2 = self.backup\_model.forward(pre\_states)

                    next\_state, reward, terminated, truncated , info = self.env.step(action)

                    if lives != info['lives']:

                        lives = info['lives']

                        reward = -20

                    elif reward == 200:

                        reward = 5

                    elif reward == 50:

                        reward = 15

                    reward\_episod += reward

                    entropy += ent

                    # ratio.append(torch.exp(log\_prob-log\_prob\_old.detach()))

                    log\_probs.append(log\_prob)

                    rewards.append(torch.tensor([reward],device=device))

                    masks.append(torch.tensor(1-terminated,device=device))

                    state = self.preprocessState(next\_state)

                    self.addState(states,state)

                    if terminated:

                        break

                self.statictics.total\_rewards = np.append(self.statictics.total\_rewards, reward\_episod)

                if (i\_episode > skip\_episodes):

                    returns = torch.cat(self.compute\_returns(rewards,masks))

                    log\_prob\_tensor    = torch.cat(log\_probs)

                    eps = np.finfo(np.float32).eps.item()

                    returns = ((returns - returns.mean()) / (returns.std() + eps)).detach()

                    # self.backup\_model = copy.deepcopy(self.policy)

                    loss = self.policy.loss(log\_prob\_tensor,returns,entropy.detach())

                    self.policy.train(loss,self.optimizer)

                    del rewards[:]

                    del log\_probs[:]

                    del masks[:]

                    # end = time.time() - start

                    # print(end)

                if i\_episode % 10 == 0:

                    print(f'Эпизод = {i\_episode} Последняя награда = {int(reward\_episod)} Средняя = {int(self.statictics.total\_rewards[-10:].mean())} Откл = {int(self.statictics.total\_rewards[-10:].std())}')

                if i\_episode % 200 == 0:

                    print(self.policy.policy[5][0].weight)

                    self.writeModel()

                    self.statictics.writeTotalReward()

                    print("SAVE MODEL")

        except Exception as e:

            print(e)

        finally:

            self.writeModel()

            self.statictics.writeTotalReward()

            print('SAVE MODEL')

    def evaluateAgent(self,n\_eval\_episodes, max\_steps):

        self.readModel()

        try:

            terminated = False

            states = []

            zerosNP = np.zeros((1,80,86))

            self.resetEnv(states,zerosNP)

            with torch.no\_grad():

                for i\_episode in range(1, n\_eval\_episodes+1):

                    reward\_episod = 0

                    if terminated:

                        self.resetEnv(states, zerosNP)

                    for step in range(1, max\_steps):

                        pre\_states = self.preprocessStates(states)

                        action,log\_prob,ent = self.policy.forward(pre\_states)

                        next\_state, reward, terminated, truncated , info = self.env.step(action)

                        state = self.preprocessState(next\_state)

                        self.addState(states,state)

                        if terminated:

                            break

                    print(f'Эпизод = {i\_episode} Награда = {int(reward\_episod)}')

        except Exception as ex:

            print(ex)

class Statictics:

    def \_\_init\_\_(self):

        self.total\_rewards = []

    def writeTotalReward(self):

        with open('Policy/total\_reward.txt','w') as f:

            np.savetxt(f,self.total\_rewards,fmt='%f')

    def readTotalReward(self):

        with open('Policy/total\_reward.txt','r') as f:

            self.total\_rewards = np.loadtxt(f,dtype=float)

    def oneGraph(self,x,xlabel,ylabel,color):

        plt.xlabel(xlabel)

        plt.ylabel(ylabel)

        plt.plot(range(x.shape[0]),x,color = color)

        plt.show()

    def twoGraph(self,x1,x2,xlabel,ylabel,colorx1,colorx2):

        plt.xlabel(xlabel)

        plt.ylabel(ylabel)

        plt.plot(range(len(x1)),x1,color = colorx1,label = 'loss')

        plt.plot(range(len(x2)),x2,color = colorx2,label = 'val\_loss')

        plt.legend(loc = 'upper left')

        plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

    pacman\_hyperparameters = {

        "n\_training\_episodes" : 100000,

        "n\_evaluation\_episodes" : 10,

        'n\_steps' : 1500,

        "gamma" : 0.99,

        "lr" : 0.0001,

        'weight\_decay' : 1e-2,

        'render\_mode' : 'rgb\_array',

        'skip\_episodes' : 1

    }

    policyGrad = PolicyGradient(pacman\_hyperparameters['lr'],pacman\_hyperparameters['gamma'],pacman\_hyperparameters['render\_mode'],pacman\_hyperparameters['weight\_decay'])

    policyGrad.train(pacman\_hyperparameters["n\_training\_episodes"],pacman\_hyperparameters['n\_steps'],pacman\_hyperparameters['skip\_episodes'])