МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ «ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра МОЭВМ

ОТЧЕТ

по «Производственной» практике

Тема: Реализация решения задачи построения маршрута с использованием алгоритмов обучения с подкреплением

Студент гр. 8310 ____ Пилипцевич Д.Д.

Руководитель ____ Азаревич А.Д.

Санкт-Петербург

2023

ЗАДАНИЕ

НА УЧЕБНУЮ ПРАКТИКУ

Студент Пилипцевич Д.Д.

Группа 8310

Тема практики: Реализация решения задачи построения маршрута с использованием алгоритмов обучения с подкреплением

Задание на практику:

Реализовать алгоритм обучения с подкреплением и внедрить его в теоретическую среду, в которой машина решит поставленную задачу построения маршрута.

Дата сдачи отчета: 20.12.2023

Дата защиты отчета:

Студент _____ Пилипцевич Д.Д.

Руководитель _____ Азаревич А.Д

АННОТАЦИЯ

Реализация решения задачи построения маршрута с использованием алгоритмов обучения с подкреплением, состоит из нескольких этапов: изучение предметной области построения маршрута, анализ существующих решений и изучение особенностей реализации алгоритмов построения маршрута на основе обучения с подкреплением. В данной работе представлено изучение различных методов, а также реализация алгоритма обучения с подкреплением, внедренным в созданную ранее среду в виде игры «прохождение трассы», в которой искусственный интеллект будет прокладывать маршрут к финишу.

SUMMARY

The implementation of a solution to the route construction problem using reinforcement learning algorithms consists of several stages: studying the subject area of route construction, analyzing existing solutions and studying the features of the implementation of route construction algorithms based on reinforcement learning. This paper presents the study of various methods, as well as the implementation of a reinforcement learning algorithm implemented in a previously created environment in the form of a “track passing” game, in which artificial intelligence will plot the route to the finish line.

Оглавление

Введение

Обучение с подкреплением - это метод машинного обучения, который обучается путем взаимодействия с окружающей средой. Агент использует методы обучения с подкреплением для обучения, то есть для получения знаний из последовательности действий, полученных в результате исследования. Его выборочные данные не существуют, что означает, что они отличаются от контролируемого процесса обучения. После того, как агент выполнит действие, он получит обратную связь от окружающей среды. Эта обратная связь представляет собой оценку действий, выполняемых окружающей средой, и представляет собой процесс “проб и ошибок”. Оценка действия, совершенного окружающей средой, - это немедленное вознаграждение, получаемое Агентом. Немедленная награда - это усиленный сигнал, который указывает на влияние выполнения этого действия на результат. Чем больше значение, тем лучше эффект, в противном случае это будет иметь плохое влияние.

Цель работы – реализовать алгоритм обучения с подкреплением и внедрить его в разработанную ранее среду.

Объект исследования – методы реализации алгоритма обучения с подкреплением и его практическое внедрение в задачу построения маршрута.

Предмет исследования – идейные особенности реализации алгоритма обучения с подкреплением, опирающиеся на исходную задачу построения маршрута

Задачи данной работы:

1. Изучить способы реализации алгоритма обучения с подкреплением.
2. Изучение различных архитектур, которые используются вместе с алгоритмом обучения с подкреплением.
3. Внедрить алгоритм обучения с подкреплением в реализованную ранее среду «прохождение трассы».

Практическая ценность работы: результатом данной работы будет практическая среда, выполненная в виде игры «прохождение трассы», в которую

внедрен искусственный интеллект, обучающийся методом «проб и ошибок» с помощью алгоритма обучения с подкреплением. Полученный продукт будет являться следующим этапом проекта «Реализация решения задачи построения маршрута с использованием алгоритмов обучения с подкреплением».

Следующим этапом данного проекта будет обучение искусственного интеллекта, сохранение его данных и помещение в различные карты разработанной ранее среды «прохождение трассы» для тестирования и отладки алгоритма.

Результат работы в осеннем семестре

1. Исследование распространенных алгоритмов обучения с подкреплением и их характеристик.

В ходе изучения данной темы, были рассмотрены популярные алгоритмы обучения с подкреплением, а именно:

1. Q-обучение (Q-Learning)
2. SARSA (State-Action-Reward-State-Action):
3. DQN (Deep Q-Network):
4. Policy Gradient (REINFORCE)
5. Actor-Critic
6. Dueling DQN
7. TRPO (Trust Region Policy Optimization)

После первичного изучения и анализа возможностей и вычислительных мощностей, было принято решение более детально изучить в рамках данной задачи Q-обучение, SARSA и DQN.

1.1. Q-обучение (Q-Learning)

Алгоритм Q-обучения (Q-Learning) представляет собой метод обучения с подкреплением, используемый для обучения агента в среде, где он принимает решения и взаимодействует с окружающей средой с целью максимизации суммарной награды. Он основан на концепции оценочной функции действия Q, которая оценивает ценность выполнения конкретного действия в конкретном состоянии. Результатом данной функции является Q-значение, которое представляет ожидаемую сумму вознаграждений, которую агент получит, выбрав определенное действие в определенном состоянии, а затем следуя определенной политике. Политика - это стратегия, определяющая, какие действия необходимо предпринять в каждом состоянии

Принцип работы алгоритма Q-обучения (Q-Learning):

1. **Определение Q-функции**. Q-функция (или оценочная функция действия) в контексте алгоритма Q-обучения представляет собой функцию, которая оценивает ценность выполнения конкретного действия *a* в конкретном состоянии *s*.

Она играет ключевую роль в процессе принятия решений агентом в среде обучения с подкреплением. Значение $Q(s,a)$ можно интерпретировать как суммарную ожидаемую награду, которую агент получит, начиная с состояния s, выполнив действие a, и следуя оптимальной стратегии вплоть до конца эпизода.

2. **Инициализация Q-таблицы.** Инициализация Q-таблицы является важным этапом при применении алгоритма Q-обучения. Q-таблица представляет собой структуру данных, где хранятся оценочные значения Q-функции для каждой комбинации состояния и действия. Начальные значения Q-таблицы могут быть установлены произвольно, но часто используются небольшие случайные числа. Это помогает во избежание симметрии и способствует более равномерному исследованию стратегий в начальной фазе обучения. Размерность Q-таблицы определяется количеством возможных состояний и действий в среде. Каждая ячейка таблицы соответствует определенному состоянию и действию. Например, если агент управляет роботом в пространстве с двумя возможными действиями (движение вперед или назад), и существует 100 различных состояний, Q-таблица будет иметь размерность 100x2. Начальные значения для каждой ячейки могут быть установлены, например, в диапазоне от 0 до 1 или в другом интервале в зависимости от требований задачи.

3. **Использование Q-значений для принятия решений.** Использование Q-значений для принятия решений является центральным моментом алгоритма Q-обучения. Q-значения предоставляют информацию об оценочной ценности каждого действия в каждом состоянии. В каждом состоянии агент выбирает действие, которое максимизирует Q-значение для этого состояния. Формально, агент выполняет следующее: $a=\text{argmax}_a Q(s,a)$. Где a - выбранное оптимальное действие, s - текущее состояние, a - возможное действие. Для обеспечения баланса между исследованием новых стратегий (эксплорацией) и использованием изученных стратегий (эксплуатацией) может быть введен параметр "эпсилон". С вероятностью ε агент выбирает случайное действие (исследование), а с вероятностью $1-\varepsilon$ выбирает оптимальное действие согласно Q-значениям (эксплуатация).

4. **Исполнение действия.** Выбранное действие a фактически выполняется

в среде. Агент воздействует на среду, и как результат, происходит переход из текущего состояния s в новое состояние s', а также агент получает награду r от среды за выполнение выбранного действия.

5. **Обновление Q-значения.** Обновление Q-значения происходит в результате выполнения действия a в состоянии s и наблюдения нового состояния s' с получением награды r. Формула обновления Q-значения выглядит следующим образом: $Q(s,a) \leftarrow (1-\alpha) \cdot Q(s,a) + \alpha \cdot (r + \gamma \cdot \max a' Q(s',a'))$

Где:

- α - коэффициент обучения, который контролирует степень, с которой новая информация заменяет старую. Маленькие значения α делают обновления Q-значений более стабильными, но медленными, в то время как большие значения α могут привести к более быстрой адаптации, но с возможностью нестабильности.
- r - полученная награда от среды за выполнение действия a в состоянии s.
- γ - дисконт-фактор, который учитывает влияние будущих наград. Значение γ должно быть между 0 и 1, и чем ближе к 1, тем больший вес придается будущим наградам.
- $\max a' Q(s',a')$ - максимальное Q-значение для всех возможных действий a' в новом состоянии s'.

Формула отражает баланс между текущим Q-значением $Q(s,a)$ и новой информацией, полученной из награды r и максимального Q-значения для следующего состояния. Это обновление дает агенту представление о том, насколько хорошо выбранное действие в текущем состоянии соответствует его целям.

6. **Повторение процесса.** Шаги 3-5 повторяются в течение нескольких эпох или до достижения критерия завершения обучения.

7. **Сходимость.** Сходимость в контексте алгоритма Q-обучения относится к тому, как обновленные Q-значения со временем стремятся к оптимальным значениям. Сходимость - это процесс, в результате которого алгоритм достигает стабильного состояния, где его результаты не изменяются или изменяются в пределах заданной погрешности. В идеале, сходимость означает, что Q-значения в

таблице приближаются к оптимальным значениям для всех состояний и действий. Оптимальные Q-значения представляют собой оценки максимальной суммарной награды, которую агент может достичь, начиная с определенного состояния и выбирая оптимальные действия. Постепенно, с повторением цикла выбора действия, его выполнения, обновления Q-значения и наблюдения за новым состоянием, алгоритм должен стать стабильным. Это означает, что обновления Q-значений не сильно меняются, и агент приобретает устойчивость к новой информации. Сходимость может быть оценена посредством различных критериев остановки. Например, обучение может считаться завершенным, когда изменения Q-значений становятся достаточно малыми или когда агент достигает определенного уровня производительности. Применение эпсилон-жадной стратегии также может влиять на сходимость. С увеличением числа эпох уменьшается вероятность случайного исследования (выбора случайного действия), что усиливает эксплуатацию изученных стратегий. Контроль параметров, таких как коэффициент обучения (α), дисконт-фактор (γ) и уровень исследования (ε в эпсилон-жадной стратегии), также влияет на сходимость. Выбор оптимальных значений этих параметров может ускорить или замедлить процесс сходимости.

1.2. SARSA

SARSA - это алгоритм обучения с подкреплением, который обновляет Q-значение для текущего состояния-действия пары, учитывая текущее действие и действие, которое будет выбрано в следующем состоянии. Данный алгоритм основным принципом своей работы идентичен алгоритму Q-обучения, но имеет отличия, которые необходимо рассмотреть подробнее:

1. **Определение последовательных действий в SARSA.** SARSA обновляет Q-значения, учитывая текущее действие (A) и следующее действие (A'), выбранные агентом в последовательных состояниях. Это отличает SARSA от Q-обучения, где обновление Q-значений происходит, ориентируясь на максимальное Q-значение для следующего состояния, независимо от выбранного действия. Формула обновления Q-значения в SARSA выглядит следующим образом: $Q(S,A) \leftarrow Q(S,A) + \alpha \cdot (R + \gamma \cdot Q(S',A') - Q(S,A))$

Где S - текущее состояние, A - текущее выбранное действие, R - полученная награда, S' - следующее состояние, A' - следующее выбранное действие, α - коэффициент обучения, γ - дисконт-фактор.

2. **Использование текущего и следующего действий.** Обновление Q-значения включает в себя оценку разницы между текущим Q-значением и комбинацией текущей награды, дисконтированного Q-значения для следующего состояния и следующего выбранного действия. Таким образом, SARSA учитывает последовательность действий, используя информацию о том, какие действия агент реально выбрал в процессе взаимодействия со средой.

1.3. DQN (Deep Q-Network)

Алгоритм Deep Q-Network (DQN) представляет собой метод обучения с подкреплением, который объединяет в себе идеи Q-обучения и глубокого обучения. Этот алгоритм позволяет обучать агента принимать решения в пространствах состояний и действий большой размерности, что делает его особенно эффективным в задачах, связанных с комплексными окружающими условиями. Ключевые шаги DQN:

1. **Нейронная сеть Q-функции.** Нейронная сеть Q-функции в алгоритме DQN (Deep Q-Network) играет центральную роль в оценке ценности действий в различных состояниях среды. Её цель — аппроксимировать Q-значения для всех возможных действий в зависимости от текущего состояния. Обычно нейронная сеть Q-функции состоит из входного слоя, скрытых слоев и выходного слоя. Входной слой принимает на вход текущее состояние, а выходной слой выдает Q-значения для всех действий. Скрытые слои выполняют промежуточные вычисления для обучения. Нейронная сеть обучается аппроксимировать Q-функцию, которая оценивает ценность каждого возможного действия в текущем состоянии. Функция Q(s, a), где s - состояние, a - действие, представляет ожидаемую суммарную награду, которую агент получит, начиная с состояния s, выбрав действие a, и далее действуя оптимально.

2. **Опытный буфер (Replay Buffer).** Опытный буфер (Replay Buffer) в алгоритме DQN (Deep Q-Network) представляет собой механизм хранения и

повторного использования предыдущего опыта агента при обучении с подкреплением. Этот компонент играет важную роль в улучшении стабильности обучения и снижении корреляции между последовательными обновлениями весов нейронной сети. Опытный буфер сохраняет переходы (или сэмплы) из взаимодействия агента со средой. Каждый переход представляет собой четверку данных: текущее состояние s, выбранное действие a, полученную награду r, и новое состояние s'. Размер опытного буфера ограничен и фиксирован заранее. Как только буфер заполняется до максимального размера, новые переходы переписывают старые, обеспечивая циркуляцию опыта. Вместо использования последовательных блоков данных, DQN выбирает случайную подвыборку из опытного буфера для обновления нейронной сети. Это позволяет уменьшить корреляцию между последовательными переходами и стабилизировать обучение

3. **Эпсилон-жадная стратегия.** Эпсилон (ε) представляет собой параметр, который определяет вероятность того, что агент будет исследовать новые действия вместо выбора действия, которое в данный момент считается наилучшим согласно текущей стратегии. С вероятностью $1-\varepsilon$ агент выбирает действие, которое считается наилучшим на основе текущей стратегии. Это позволяет агенту эксплуатировать уже известные, хорошие действия для максимизации награды в текущей ситуации. Выбор оптимального значения для параметра ε зависит от конкретной задачи и требований. Если ε слишком велико, агент будет часто исследовать, упуская возможность эксплуатации уже известных выгодных действий. Если ε слишком мало, агент может застрять в локальных оптимумах, не исследуя новые возможности. Часто применяется стратегия динамического уменьшения значения эпсилона с течением времени или эпох обучения. Например, начиная с высокого значения ε для активного исследования в начале обучения, постепенно уменьшают его до более низкого уровня для более фокусированной эксплуатации. Такая стратегия позволяет эффективно обучаться в неопределенных средах, обеспечивая агенту возможность исследовать и одновременно использовать опыт для достижения оптимальных результатов.

4. **Обновление параметров нейронной сети.** Обновление параметров

нейронной сети в алгоритме DQN (Deep Q-Network) происходит в процессе обучения с использованием градиентного спуска. Этот процесс направлен на минимизацию функции потерь и улучшение аппроксимации Q-функции. Функция потерь определяет разницу между предсказанными Q-значениями и целевыми Q-значениями. Обозначим функцию потерь как L. Она обычно выражается среднеквадратичной ошибкой (MSE) между предсказанными и целевыми Q-значениями. Целевые Q-значения (Qtarget) вычисляются с использованием целевой нейронной сети, которая является копией основной сети с некоторой задержкой. Для каждой пары состояние-действие (s,a) целевое Q-значение вычисляется по формуле: $Q\text{target}(s,a)=r+\gamma\cdot\max_{a'}Q(s',a')$ где r - полученная награда, s' - новое состояние, γ - дисконт-фактор. Обновление параметров нейронной сети осуществляется с использованием градиентного спуска. Градиент функции потерь по отношению к параметрам сети ($\nabla\theta L$, где θ - параметры сети) вычисляется с помощью обратного распространения ошибки (backpropagation). Затем происходит изменение весов в направлении, противоположном градиенту, с учетом коэффициента обучения (α). Обновление весов происходит с использованием следующей формулы: $\theta\leftarrow\theta-\alpha\cdot\nabla\theta L$ где θ - параметры нейронной сети, α - коэффициент обучения, $\nabla\theta L$ - градиент функции потерь по отношению к параметрам сети. Для реализации градиентного спуска часто используются различные оптимизаторы, такие как оптимизатор Adam, RMSProp, или SGD (стохастический градиентный спуск). Оптимизаторы могут варьироваться в своей способности адаптации шага обучения и обработки градиентов.

5. **Формула обновления Q-значения.** Формула обновления Q-значения в алгоритме Q-обучения (Q-Learning) описывает, как агент пересчитывает оценки ценности действий в соответствии с полученной наградой и максимальным Q-значением в новом состоянии. Пусть $Q(s,a)$ - Q-значение для состояния s и действия a. Формула обновления Q-значения в Q-обучении выглядит следующим образом:

$$Q(s,a)\leftarrow(1-\alpha)\cdot Q(s,a)+\alpha\cdot(r+\gamma\cdot\max_{a'}Q(s',a'))$$

где:

- s - текущее состояние,

- a - выбранное действие в состоянии s,
- r - полученная награда после выполнения действия a в состоянии s,
- α - коэффициент обучения (learning rate), который определяет, насколько сильно новая информация замещает предыдущие знания ($0 \leq \alpha \leq 1$),
- γ - дисконт-фактор, который определяет важность будущих вознаграждений ($0 \leq \gamma \leq 1$),
- s' - новое состояние, в которое переходит среда после выполнения действия a,
- a' - возможные действия в новом состоянии s'.

Сложение двух компонентов даёт обновленное Q-значение для состояния s и действия a.

6. **Замораживание целевой нейронной сети.** Этот прием используется для стабилизации обучения и предотвращения колебаний весов, которые могут возникнуть из-за динамических изменений в целевых Q-значениях. В алгоритме DQN используются две нейронные сети: основная сеть (online network) и целевая сеть (target network). Основная сеть используется для выбора действий в текущем состоянии, а целевая сеть используется для вычисления целевых Q-значений. Замораживание целевой сети означает, что параметры (веса) этой сети копируются из основной сети с определенной периодичностью. На протяжении нескольких итераций обучения параметры целевой сети остаются неизменными. Частота обновлений целевой сети определяется гиперпараметром, который называется "период обновлений" или "шаг замораживания". Например, если этот параметр равен 1000, то целевая сеть замораживается и обновляется каждые 1000 шагов обучения. Замораживание целевой сети способствует стабилизации обучения, поскольку это предотвращает частые и резкие изменения весов, которые могут привести к неустойчивости в обучении. Стабильные целевые Q-значения помогают более надежно оценивать ценность действий.

1.4. Результат исследования.

На основании проведенного изучения было принято решение использовать DQN (Deep Q-Network) так, как ИИ необходимо активно взаимодействовать с

окружающей средой для нахождения решения поставленной задачи. Использование Dueling DQN(улучшенная версия DQN) было расценено нецелесообразным, так как на порядок труднее в реализации и требует больше вычислительных мощностей, но в некоторых задачах увеличение результативности либо крайне незначительное, либо отсутствует.

2. Описание реализации алгоритма обучения с подкреплением

2.1. Реализация класса DDDQNNet

Класс **DDDQNNet** представляет собой реализацию нейронной сети для Double Deep Q-Learning (DDDQN) в рамках обучения на задаче управления агентом в среде, предоставляемой библиотекой Pygame. Класс обеспечивает функциональность для создания и обучения глубокой нейронной сети, используемой в алгоритме DDDQN для принятия решений в процессе взаимодействия агента со средой.

1.1. Параметры класса:

- `state_size`: Размер входного состояния агента.
- `action_size`: Количество возможных действий агента.
- `learning_rate`: Скорость обучения для оптимизатора RMSProp.
- `name`: Имя сети, используемое для переменных TensorFlow.

1.2. Методы класса:

- `__init__(self, state_size, action_size, learning_rate, name)`: Инициализирует объект класса DDDQNNet.
- `update_target_graph()`: Обновляет веса целевой сети с весами основной сети.

1.3. Нейронная сеть состоит из следующих слоев:

- Входной слой (`self.inputs_`): Принимает состояние агента в качестве входных данных.
- Скрытый слой 1 (`self.dense1`): Полносвязный слой с функцией активации ELU.
- Скрытый слой 2 (`self.dense2`): Второй полносвязный слой с функцией активации ELU.
- Слой для оценки значения (`self.value`): Полносвязный слой для оценки значения состояния.
- Слой для оценки преимущества (`self.advantage`): Полносвязный слой для оценки преимуществ действий.
- Выходной слой (`self.output`): Выходной слой, объединяющий

оценки значения и преимущества.

1.4. Процесс обучения

- Нейронная сеть использует технику Double DQN, разделяя выбор действия и оценку его ценности.
- Используется experience replay для хранения и повторного использования предыдущих опытов агента.
- Обновление весов происходит с использованием градиентного спуска и оптимизатора RMSProp.
- Применяется техника Importance Sampling для управления весами при обновлении сети.

1.5. Гиперпараметры

- `state_size`: Размер входного состояния агента.
- `action_size`: Количество возможных действий агента.
- `learning_rate`: Скорость обучения для оптимизатора RMSProp.
- `total_episodes`: Общее количество эпизодов для обучения.
- `max_steps`: Максимальное количество шагов в одном эпизоде.
- `batch_size`: Размер мини-пакета для обновления весов сети.
- `max_tau`: Параметр, определяющий, с какой периодичностью обновлять веса целевой сети.
- `explore_start`: Начальная вероятность исследования (epsilon-greedy).
- `explore_stop`: Минимальная вероятность исследования (epsilon-greedy).
- `decay_rate`: Экспоненциальная скорость уменьшения вероятности исследования.
- `gamma`: Коэффициент дисконтирования будущих вознаграждений.
- `memory_size`: Размер памяти для хранения опыта агента.
- `pretrain_length`: Начальная длина опыта в памяти перед началом обучения.

- `training`: Флаг, указывающий, производится ли обучение.
- `episode_render`: Флаг, указывающий, производится ли визуализация эпизодов во время обучения.

1.6. RMSProp (Root Mean Square Propagation)

RMSProp (Root Mean Square Propagation) - это оптимизационный алгоритм, применяемый в глубоком обучении для обновления весов нейронных сетей в процессе обучения. Он был предложен Георгем Хинтоном в 2012 году и представляет собой модификацию стохастического градиентного спуска (SGD).

1.7. Основные идеи RMSProp:

- Адаптивная скорость обучения - основной принцип RMSProp заключается в том, чтобы адаптировать скорость обучения для каждого параметра сети. Это позволяет учитывать разные шаги обновления для различных параметров.
- Использование квадрата градиентов - алгоритм использует квадраты градиентов для каждого параметра. Это позволяет алгоритму подстраиваться под изменения градиента в процессе обучения.
- Экспоненциальное сглаживание - RMSProp использует экспоненциальное сглаживание (exponential smoothing) для обновления среднего квадрата градиентов. Это позволяет алгоритму быстро реагировать на изменения градиента и приспосабливаться к динамике обучения

Алгоритм обновления весов в RMSProp

Для каждого параметра θ алгоритм вычисляет:

- $E[g2]t$ - экспоненциально сглаженное значение квадрата градиента для параметра на текущем шаге t.
- α - параметр сглаживания, обычно близкий к 1.
- ϵ - небольшое значение, добавленное для численной стабильности.
- gt - градиент по параметру на текущем шаге t.

Обновление весов происходит следующим образом:

$$E[g^2]_t = \alpha E[g^2]_{t-1} + (1-\alpha) g^2_{-t}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} * g_t$$

Преимущества RMSProp:

- RMSProp позволяет автоматически адаптировать скорость обучения для каждого параметра, что может ускорить сходимость.
- Алгоритм хорошо справляется с задачами, в которых данные разрежены, так как он адаптируется к различным масштабам градиентов.
- Использование экспоненциального сглаживания позволяет алгоритму стабильно реагировать на изменения в градиентах.

RMSProp является популярным выбором в области глубокого обучения и часто применяется вместе с другими оптимизационными методами для достижения лучших результатов в обучении нейронных сетей.

2.2. Реализация класса SumTree

Класс **SumTree** базируется на бинарном дереве суммы, представляющем собой иерархическую структуру с узлами и листьями. Он оптимизирован для эффективного распределения и поиска опыта с учетом их приоритета.

Основные Компоненты:

1. **Капаситет и Память.** Капаситет в структуре SumTree представляет собой заранее определенный лимит по количеству хранимых элементов опыта. Этот параметр фиксирует максимальное количество данных, которые система может вместить. Он играет роль границы, контролируя объем информации, доступной для обучения агента. Капаситет определяет, насколько далеко раскиданы узлы дерева и формирует основу для управления приоритетами. Для хранения информации SumTree использует два основных массива - `tree` и `data`. Первый, `tree`, представляет собой бинарное дерево, где каждый узел содержит информацию о суммарном приоритете. Эти узлы организованы таким образом, чтобы обеспечивать быстрый поиск и доступ к данным опыта. Второй массив, `data`, служит для хранения собственно опыта - пары (состояние, действие, награда, следующее состояние). Таким образом, память SumTree представляет собой совокупность этих данных, размещенных в соответствии с их временным порядком.

2. **Добавление и Обновление:**

- Метод `add(priority, data)` принимает новый опыт и его

приоритет. Процесс добавления нового опыта в SumTree начинается с приема данных о состоянии, совершенном действии, полученной награде и следующем состоянии. Эти параметры формируют запись опыта, которая затем помещается в структуру памяти SumTree. Новый опыт интегрируется в дерево, учитывая его временной порядок. После добавления нового опыта система SumTree пересчитывает приоритеты всех связанных с ним узлов. Изменения приоритетов основаны на значимости добавленного опыта и его влиянии на общий приоритет в дереве. Происходит пересчет весов, учитывающих актуальность данных.

- Метод `update(tree_index, priority)` обновляет значение приоритета в узле дерева, а затем корректирует суммы вверх по дереву. Добавление нового опыта также влияет на общий приоритет дерева. Общий приоритет пересчитывается, учитывая новый опыт. Этот параметр является ключевым, поскольку он служит ориентиром для агента при выборе данных для обучения. Обновление общего приоритета предоставляет системе актуальную картину значимости доступного опыта. Добавление новых данных также связано с контролем капаситета. Если текущий объем опыта превышает установленный лимит, система принимает меры по удалению старых записей. Этот механизм обеспечивает поддержание структуры данных в пределах заданного капаситета, предотвращая избыточное накопление информации.

3. **Выбор и Поиск.**

Метод `get_leaf(v)` позволяет эффективно выбирать опыт на основе случайной величины **v**, сравнивая ее с приоритетами в дереве. Выбор данных из SumTree важен для обучения агента, поэтому этот процесс основан на высокоэффективных механизмах. В центре выбора лежит общий приоритет дерева, который выступает в роли вероятности выбора конкретного опыта. Чем выше приоритет, тем больше вероятность того, что опыт будет выбран. Приоритеты каждого опыта в дереве формируют распределение вероятностей, исходя из которого осуществляется выбор. Это означает, что опыты с более высоким приоритетом имеют больше шансов быть выбранными для обучения. Это структура стимулирует агента сфокусироваться на более важных данных, максимизируя их

влияние на обучение. Важно отметить, что процесс выбора в SumTree учитывает не только абсолютный приоритет, но и стремится поддерживать разнообразие данных. Для этого механизмы выбора иногда могут варьировать вероятности в соответствии с текущими приоритетами, обеспечивая баланс между приоритетами и разнообразием данных.

4. **Общий Приоритет.**

Атрибут `total_priority` предоставляет суммарный приоритет корневого узла. Это важное значение, отражающее общую значимость всего опыта в памяти. Общий приоритет в SumTree является ключевым понятием, лежащим в основе эффективного выбора данных для обучения агента. Этот параметр представляет собой численное значение, отражающее важность каждого опыта в дереве. Чем выше значение, тем выше приоритет и, следовательно, больше вероятность выбора данного опыта. Общий приоритет формируется на основе оценки значимости каждого опыта для обучения агента. Эта оценка может включать в себя факторы, такие как степень ошибки или влияние на предсказания агента. Опыты, вносящие больший вклад в улучшение стратегии, получают более высокие приоритеты. Важно отметить, что общий приоритет поддерживается механизмами динамического обновления. Система реагирует на изменения в данных и обучении, пересчитывая приоритеты в реальном времени. Это обеспечивает адаптивность и эффективное использование ресурсов обучения. Приоритеты напрямую связаны с ошибками, допущенными агентом в предсказаниях. Опыты, в которых модель совершает большие ошибки, получают повышенный приоритет, что стимулирует агента фокусироваться на улучшении своих слабых сторон. Такая стратегия способствует более эффективному обучению. Система также поддерживает уравновешивание приоритетов, чтобы избежать чрезмерной фокусировки на ограниченном подмножестве данных. Это обеспечивает разнообразие обучающих сценариев и повышает обобщающую способность агента.

2.3. Реализация класса Memory.

Класс Memory представляет собой ключевой элемент структуры данных, используемой в контексте обучения с подкреплением. Эта структура предназначена

для эффективного хранения и управления опытом, собранным агентом в ходе взаимодействия с окружающей средой. Роль класса Memory заключается в обеспечении агенту доступа к предыдущим состояниям, действиям и наградам для последующего обучения.

Архитектурные Компоненты Класса Memory:

1. **Хранение Состояний и Действий -** аккумулирует и хранит информацию о состояниях, в которых находился агент, и действиях, которые он предпринимал. Это позволяет агенту анализировать свое прошлое поведение и строить оптимальные стратегии на основе опыта.

2. **Награды и Ошибки -** класс также учитывает полученные агентом награды и вычисленные ошибки предсказания модели. Эти данные имеют важное значение при обновлении внутренних представлений агента, направленных на улучшение стратегии.

3. **Управление Приоритетами -** может использовать механизмы управления приоритетами, определяя, какой опыт является более важным для последующего обучения. Это способствует эффективной настройке агента на основе опыта с более высоким обучающим значением.

4. **Память в Контексте Задач -** класс динамически адаптируется к различным задачам обучения с подкреплением. Это включает в себя управление размером памяти, выбором критериев важности опыта и другие аспекты, обеспечивающие оптимальное использование ресурсов.

Добавление информации в Memory начинается с захвата текущего состояния агента, регистрации совершенного действия и зафиксированной награды. Эти элементы объединяются в опыт, который затем интегрируется в память. Процесс этот подразумевает не только хранение данных, но и определение их важности для дальнейшего обучения.

2.4. Цикл обучения

Процесс обучения модели в данном коде реализуется через функцию `train_network()`, которая включает в себя ряд шагов. Предлагаю рассмотреть каждый из них:

1. **Выбор мини-пакета из памяти.** В начале функции выбирается случайный мини-пакет из памяти с использованием функции `memory.sample(batch_size)`. Этот мини-пакет представляет собой случайные опыты агента из прошлых эпизодов.

2. **Подготовка данных.** Полученные из памяти состояния, действия, награды, следующие состояния и флаги завершения эпизода разбираются и подготавливаются для передачи в нейронную сеть.

3. **Вычисление Q-значений.** С использованием текущей сети **DQNetwork** расчитываются Q-значения для текущих состояний. Это делается с помощью `sess.run(DQNetwork.output, feed_dict={DQNetwork.inputs_: states_mb})`, где `states_mb` - это текущие состояния из мини-пакета.

4. **Вычисление Q-значений для следующих состояний.** Также с использованием **DQNetwork** расчитываются Q-значения для следующих состояний (важно, что используется текущая сеть). Это выполняется с помощью `sess.run(DQNetwork.output, feed_dict={DQNetwork.inputs_: next_states_mb})`, где `next_states_mb` - следующие состояния из мини-пакета.

5. **Вычисление Q-значений для следующих состояний в Target-сети.** Также расчитываются Q-значения для следующих состояний, но уже с использованием **TargetNetwork**. Это делается с помощью `sess.run(TargetNetwork.output, feed_dict={TargetNetwork.inputs_: next_states_mb})`.

6. **Расчет целевых Q-значений.** Для каждого опыта в мини-пакете рассчитываются целевые Q-значения с учетом формулы двойного Q-обновления.

7. **Обновление весов сети.** Производится обновление весов сети (обучение) с использованием оптимизатора. Это выполняется через `sess.run([DQNetwork.optimizer, DQNetwork.loss, DQNetwork.absolute_errors], feed_dict={...})`, где передаются входные данные, целевые Q-значения и другие параметры.

8. **Обновление приоритетов в памяти.** После обучения рассчитываются

абсолютные ошибки (по модулю) и происходит обновление приоритетов в памяти с использованием `memory.batch_update(tree_idx, absolute_errors)`.

Таким образом, весь процесс обучения заключается в выборе случайного мини-пакета опыта, расчете Q-значений, обновлении весов сети, итеративном повторении этого процесса на протяжении нескольких эпизодов.

План работ на весенний семестр

- Исправление «багов» и ошибок в коде
- Улучшение визуальной составляющей(исправление проблемы с разрешением экрана)
- Создание дополнительных карт, куда в дальнейшем будет помещен обученный ИИ для нахождения пути.
- Проведение исследования и анализ сильных и слабых сторон реализованного алгоритма обучения с подкреплением.

Заключение

В ходе данной работы были изучены распространенные алгоритмы обучения с подкреплением. Был выбран самый подходящий для текущей задачи алгоритм и внедрен в разработанную ранее игру «прохождение трассы», в которой можно наблюдать за тем, как искусственный интеллект обучается находить путь, тем самым выполняя основную цель данного проекта. В дальнейшем будет доработан интерфейс и созданы дополнительные среды, в которых можно будет проверить результат обучения искусственного интеллекта.

Список использованных источников

1. Арулкумаран К., Дейзенрот М. П., Брандейдж М. и Бхарат А. А. (2017). Глубокое обучение с подкреплением: краткий обзор. 26-38. (дата обращения: 20.12.2023)

2. Боттеги, Н., Сирмачек, Б., Мустафа, К. А., Поэль, М. и Страмиджоли, С. (2020). О формировании вознаграждения за навигацию мобильного робота: подход, основанный на обучении с подкреплением и SLAM. 1025-1037 (дата обращения: 16.12.2023)

3. Ван, Б., Лю, З., Ли, К. и Пророк, А. (2020). Планирование пути мобильного робота в динамических средах посредством глобального обучения с подкреплением. 22-29 (дата обращения: 15.12.2023)

4. Артур Деларю, Росс Андерсон и Кристиан Тяндраатмаджа. Обучение с подкреплением комбинаторные действия: приложение для маршрутизации транспортных средств. Достижения в области нейронных систем обработки информации, 33, 2020. (дата обращения: 18.12.2023)

5. Леррел Пинто, Джеймс Дэвидсон, Рахул Суктанкар и Абхинав Гупта. Надежное состязательное обучение с подкреплением. В материалах 34-й Международной конференции по машинному обучению, том 70, стр. 2817–2826. JMLR. Орг, 2017(дата обращения: 05.12.2023)

6. Документация по библиотеке pygame языка программирования Python [Электронный ресурс]. URL: https://www.pygame.org/docs/ (дата обращения: 29.11.2023)

7. Документация по библиотеке pygame языка программирования Python [Электронный ресурс]. URL:https://pyglet.readthedocs.io/en/latest/ (дата обращения: 01.12.2023)

8. Документация по библиотеке tensorflow языка программирования Python [Электронный ресурс]. URL:https://www.tensorflow.org/guide?hl=ru (дата обращения: 10.12.2023)

Приложение А

Ссылка на полный код проекта: https://github.com/PiliptsevichDanil/Research-work-II