ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
"МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ
имени М.В.ЛОМОНОСОВА"
ФИЗИЧЕСКИЙ ФАКУЛЬТЕТ
КАФЕДРА ФИЗИКО-МАТЕМАТИЧЕСКИХ МЕТОДОВ
УПРАВЛЕНИЯ

Выпускная дипломная работа

Обучение с подкреплением в задаче поиска пути в лабиринте

Выполнил студент IV курса:
Завгородний Игорь Викторович

Научный руководитель:
Галяев А.А.

Москва
2019

1. Введение

Данная работа посвящена применению метода обучения с подкреплением (Reinforcement Learning) в задаче поиска оптимального пути в трёхмерном лабиринте. Построеннаые математические и програмные модели применимы для описания движения агентов в различных физических системах. Например, описание движения беспилотного летательного аппарата (БПЛА), выполняющего задачи в различных слоях атмосферы, описание движения автономного подводного судна, выполнящего исследования на разной глубине, и так далее.

В результате работы был создан и протестирован алгоритм, позволяющий осуществлять оптимальное управление агентом в трёхмерном лабиринте, имитирующим атмосферу. Метод обучения с подкреплением показал эффективность при обучении агента на заданных лабиринтах, где данные не меняются с течением времени, что, безусловно, отличается от реальных процессов.

Оглавление

2. Теоретическое введение

Современные задачи науки и техники требуют применения современных методов, позволяющих быстро и корректно обратывать большие объёмы данных, ежесекундно поступающих с многочисленных датчиков. Более того, с увеличением объёма задач, стоящих перед кибернетическии агентами, усложняется их поведение. Традиционные методы программирования исчерпывают себя, делая решение современных задач неэффективным по затрачиваемому времени и используемой памяти.

Данные проблемы призван преодолеть метод машинного обучения (Machine Learning), фундаментальные основы которого были заложены еще в 1940-1950-х годах прошлого века. Однако бурное развитие подобных методов началось лишь в 1990-х годах вместе с ростом вычислительных мощностей компьютеров. Достоинством данного метода является отсутсвие необходимости создавать детерменированные алогритмы, полностью покрывающие необходимые сценарии поведения агентов. Машинное обучение позволяет создать агентов нового типа, способных обучаться и строить оптимальные алгоритмы при минимальном воздействии человека.

Существует две основных концепции машинного обучения: обучение с учителем, в котором агент обучается производить определённые действия на основании предварительно подготовленных выборок, и обучение без учителя, в котором агент самостоятельно формирует стратегию поведения, опираясь на изменения, производимые его действиями. Обучение с подкреплением принадлежит ко второму типу машинного обучения. Агент перебирает все варианты действий и из всех возможных действий выбирает те, которые принесут ему наибольшее итоговое вознаграждение. Перечисленные концепции называются "методом проб и ошибок"и "отсроченным поощрением" они лежат в основе обучения с подкреплением.

В данной работе для решения задачи поиска пути в лабиринте применяется метод обучения с подкреплением. Как было сказано ранее, одной из особенностей метода является то, что обучение агента

происходит благодаря взаимодействию с окружающей средой. Лабиринт - это и есть среда, предназначенная для экспериментального исследования, в которой движется управляемый агент. Задача поиска пути в лабиринте является одной из ключевых задач в робототехнике, решение которой позволяет создавать системы управления движением автономных роботов (дронов).

Метод обучения с подкреплением в общем виде можно представить в качестве марковского процесса принятия решений:

$$(S, A, P_a(s, s'), R_a(s, s')), \text{ где:}$$

1. S - множество возможных состояний среды,
2. A - множество возможных действий агента над средой,
3. $P_a(s, s') = P(s_{t+1} = s' \mid s_t = s,\ a_t = a)$ - вероятность, что состояние s под действием a во время t перейдёт в состояние s' ко времени $t + 1$,
4. $R_a(s, s') = R(s_{t+1} = s' \mid s_t = s,\ a_t = a)$ - вознаграждение, получаемое после перехода в состояние s' из состояния s с вероятностью $P_a(s, s')$.

Поведение агента описывается следующей цепочкой действий:

$$\text{состояние} \to \text{действие} \to \text{поощрение} \to \text{состояние} \to$$
$$\to \text{действие} \to \text{поощрение} \to \ldots$$

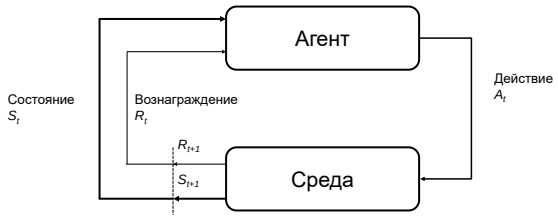

Рис. 1: SARSA-модель

В англоязычной литературе данный процесс носит название «SARSA» («State-Action-Reward-State-Action-...»).

Вводится некоторая политика (англ. *policy*):

$$\pi \ : \ S \times A \to [0, 1]$$

$\pi(a \mid s) = P(a_t = a \mid s_t = s)$ - вероятность действия a в состоянии s.

Цель агента - выбрать такую оптимальную политику π, обозначающую вероятность выбора действия a в состоянии s, чтобы при следовании ей сумма вознаграждений, получаемых от среды, была максимальна. Ожидаемая награда в момент времени t определяется как:

$$R_t = E[r_t \ + \ \gamma r_{t+1} \ + \ \gamma^2 r_{t+2} \ + \ ...] = E\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k}\right],$$

где $E[\cdot]$ - математическое ожидание, $\gamma \in (0, 1)$ - коэффициент дисконтирования (англ. *discount rate*).

Долгосрочная стратегия агента в общем случае не подразумевает преследование максимальной выгоды на каждом ромежуточном шаге. Непосредственный выбор стратегии может осуществляться множеством способов. Введем функцию $Q(s, a)$, которая парам состояние-действие ставит в соотвествие число. Данное число называется ценностью состояния-действия. Также на каждом временном шаге t агент получает вознаграждение r_t:

$$Q^{\pi}(s, a) = E_{\pi}[R_t | \ s_t = s, a_t = a] = E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, a_t = a\right],$$

где индекс π означает выбор действий в соотвествии с некоторой политикой (*policy*).

Эта функция характеризует ожидаемую награду, получаемую агентом стартуя из состояния $s, s \in S$ совершая действие $a, a \in A$, и в дальнейшем действуя в соответсвии с определенной политикой π.

Отсюда мы можем получить рекурсивную формулу для оценки данной функции:

$$Q^{\pi}_{i+1}(s,a) = E_{\pi}\left[r_t + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right] = \\ E_{\pi}\left[r_t + \gamma Q^{\pi}_i(s_{t+1} = s', a_{t+1} = a') \mid s_t = s, a_t = a\right]$$

Однако целью агента является - нахождение оптимальной политики π, на которой достигается максимальная ожидаемая награда. Таким образом, мы должны найти такую π^*, которая в результате нам дает максимальное значение action-value функции $Q^*(s,a)$ среди всех существующих политик. Формула для оценки оптимального значения action-value функции определяется следующим образом:

$$Q_{i+1}(s,a) = E[r_t + \gamma max_{a'} Q_i(s',a') \mid s,a]$$

При $i \to \infty$ следует, что $Q_i(s,a) \to Q^*(s,a)$. Данный процесс называется *алгоритмом итерации значений* (англ. *value iteration algorithm*).

3. Постановка и формализация задачи

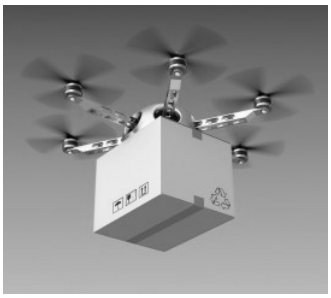

Рис. 2: Беспилотный аппарат

Рассмотрим движение беспилотного аппарата (агента) в атмосфере (испытательной среде, лабиринте). Задачей агента является сбор грузов в различных точках пространтва и их доставка до точек выгрузки с наименьшими затратами топлива.

Для описания системы используются следующие величины:

1. $\vec{r}\ (x,y,z)$ - координата беспилотного аппарата в трёхмерном пространстве: $x \in [0; l_x]$, $y \in [0; l_y]$, $z \in [0; l_z]$;
2. $\rho(\vec{r})$ - плотность атмосферы в точке пространства: $\rho \in [\rho_{min}; \rho_{max}]$;

3. F - статус наличия груза: $F \in \{0; 1\}$.

Множество действий, доступных агентов состоит из восьми элементов:

1. *south* - увеличение координаты y на 1 (движение на юг);
2. *north* - уменьшение координаты y на 1 (движение на север);
3. *east* - увеличение координаты x на 1 (движение на восток);
4. *west* - уменьшение координаты x на 1 (движение на запад);
5. *east* - увеличение координаты z на 1 (движение вверх);
6. *west* - уменьшение координаты z на 1 (движение вниз);
7. *pickup* - изменение статуса F до 1 (сбор груза);
8. *dropoff* - изменение статуса F до 0 (сброс груза).

Таким образом, состояние среды описывается четыремя непрерывными вещественным

4. Описание используемых методов

Перед тем как описывать способы определения ценностей для пар состояний-действий, стоит указать используемые стратегии выбора действий. В данной задаче есть два принципиально разных класса состояний, следовательно, и стратегии для них тоже должны быть разные. В случае, когда производится выбор карты для озвучивания, карта выбирается следующим образом. В руке находятся наименования с наибольшим количеством карт, далее из них случайно выбирается одно наименование. Во втором случае используется -жадная стратегия, она заключается в жадном выборе действия (действие, которое максимизируетQ(s, a) с вероятностью , в остальных случаях действие выбирается случайно. Все используемые в работе методы построения оценки функции ценности пар состояний-действий основаны на методе временных различий (TD — Temporal-Difference). В TD-методах процесс обучения основывается на опыте взаимодействия агента со средой без использования модели среды. Расчетные оценки состояний (в случае задачи

управления состояний-действий) в TD-методах обновляются, основываясь на других полученных оценках, т.е. они самонастраиваются [2]. Классический TD-метод используют для построения оценок ценности состояния среды. Опишем его, перед тем как перейти к случаю управления. В данной работе будут использоваться идеи многошагового TD-метода, так же известного как метод TD, и одношагового метода, или метода TD(0), который является частным случаем многошагового. В многошаговом методе имеется переменная памяти e(s) , соответствующая каждому состоянию. Она называется следом приемлемости [2]. На каждом временном шаге следы приемлемости для всех состояний, кроме текущего, убывают с коэффициентом, а след приемлемости для посещаемого на данном шаге состояния увеличивается на параметр затухания следа, — коэффициент приведения. След приемлемости все время регистрирует, посещение каких состояний имело место недавно, где смысл понятия "недавно"определяется с помощью коэффициента. Процесс оценки состояний проходит следующим образом. Во время обучения при переходе из состояний st в состояние st+1 вычисляется величина где V(st) — функция ценности состояния, аналогичная функции ценности пар состояний-действий Q(s, a) . Далее для всех состояний производится корректировка их ценности с использованием следов приемлемости где — коэффициент обучения. Соответственно, в случае одношагового метода никаких следов приемлемости нет, т.к. = 0 , поэтому на каждом шаге производится только корректировка ценности состояния st, что можно записать в виде Одним из наиболее важных достижений в обучении с подкреплением стало развитие управления по TD-методу с разделенной оценкой ценности стратегий, известного как Q-обучение. В данной работе используется простейший одношаговый алгоритм корректировки ценностей пар состояние- действие, который основывается на одношаговом методе TD, (4) с штрихами здесь состояния и действия$st+1at+1$, без штрихов stiat . В этом случае искомая функция ценности действия Q непосредственно аппроксимирует оптимальную функцию ценности действий, независимо от применяющейся стратегии. [3]. Альтернативой методам Q-обучения является метод SARSA (State-Action- Reward-State-Action), который основывается на модели обобщенной итерации по стратегиям с использованием TD-метода в оценочной или предсказательной части. В данной работе используется TD-метод управления с интегрированной оценкой ценности

стратегий. Последовательность действий в методе SARSA() базируется на двух шагах. Первый шаг заключается в изучении функции ценности действий. Для этого необходимо оценить функцию Q(s, a) для состояния s и всех действий a. Далее выбирается действие a и производится переход в следующее состояние. Второй шаг повторяет первый, только в конце шага вместо перехода производится корректировка ценностей всех пар состояний-действий [4]. По аналогии с методом TD() находится величина, а далее для всех пар состояний-действий производится корректировка оценок и корректировка всех следов приемлемости.

5. Программная реализация

Основой для решения послужила библиотека Gym от OpenAI. Библиотека содержала, рассмотренную мной задачу в упрощённом виде: обучение с подкреплением использовалось для оптимизации обработки заказов и движения такси в двумерном лабиринте.
Несмотря на кажущуюся схожесть с задачей управления беспилотным аппаратом, требовалась серьёзная доработка существующего решения:

- Требовалось обощить задачу на случай движения в трёх измерениях;
- Требовалось изменить постановку задачи так, чтобы добавить физический и прикладной смыслы.

Обе задачи были выполнены.

Основные компоненты:

- main.py - основной файл, в котором реализовано обучение агента с помощью Q-learning, заданы параметры обучения (количетсво эпизодов, максимальное количетсво шагов в эпизоде, параметры Q-learning т.д.)
- labyrinth.py - файл, в котором реализованы "правила"взаимодействия агента со средой.
- map_generation.py - файл, который содержит необходимые функции для построения символьного поля среды, в которой будет происходить обучение агента.

- discrete.py - вспомогательный файл, который был разработан OpenAI для обучения с подкреплением.

Код каждого файла (будет) представлен в Приложении. Теперь рассмотрим каждую часть более подробно.

5.1 Создание среды

Как отмечалось выше, программа была реализована с помощью библиотеки gym от OpenAI, так же был использован пакет numpy для более удобной работы с матрицами.
В моей задаче, среда - это параллелепипед, задаваемый тремя параметрами (длина, ширина, высота). Создание символьного поля производится в файле map_generation.py (см. Приложение), в котором с помощью символов +, -, | и : формируется среда, а так же случайным образом расставляются пункты назначения для агента (R(ed), G(green), B(lue), Y(ellow)) (см. рисунок ниже).

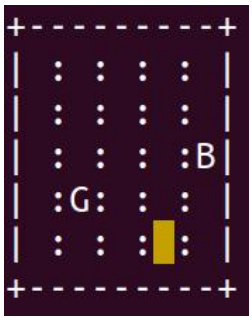

Рис. 3: Слой в какой-то момент времени.

Основная цель данного символьного поля - провизуализировать перемещение агента в среде. Как отмечалось в разделе "Постановка и формализация задачи агенту доступны следующие действия (actions):

- Двигаться на юг (move south)
- Двигаться на север (move north)
- Двигаться на восток (move east)
- Двигаться на запад (move west)
- Двигаться наверх (move up)

- Двигаться вниз (move down)
- Подобрать объект (pickup)
- Положить объект (dropoff)

За каждое действие, агент получается очки (rewards). Они могут быть как очки вознаграждения, когда агент доставил объект из одной точки в другую, так и очки штрафы, когда агент сделал неправильное действие, например, доставил объект не в то место или врезался в препятствие. Кроме того, за каждое перемещение агент теряет очки (топливо). И в зависимости от того, в какой ячейке находится агент, он зтрачивает различное количество очков. За то, сколько необходимо потратить на перемещение отвечает функционал, который каждому набору данных в ячейке ставит в соответствие вознаграждение. В самом простом случае, в ячейке хранится уровень слоя, но в моей реализации среда также может учитывать плотность, давление и споротивление воздуха на данной высоте. При проведении различных испытаний, связанных с изменнием количества эпизодов обучения агента, размеров среды и т.д. учитывается только высота слоя, поэтому вознаграждние рассчитывается следюущим образом:

$$\text{reward} = \text{lay\_reward}(\text{lay}),$$

где lay_reward - это структура данных "ключ-значение где ключ - это номер слоя, а значение - очки на этом слое. В моём случае, нулевой слой соответсвует вознаграждению -1, а n-ый слой вознаграждению -n.

Теперь рассмотрим количетсво возможных состояний в данной задаче. Всю среду можно представить в виде трехмерной сетки size_x*size_y*size_z. Количетсво ячеек этой сетки равно количетсву возможных расположений агента. В среде так же расположены 4 возможных места назначения. Если еще учесть одно состояние объекта: объект находится у агента, то можно подсчитать общее количество состояний в нашей среде для обучения агента. Итого, четыре возможных расположений пунктов назначений и 5 возможных расположений для объекта. Следовательно, в нашей среде насчитывается

$$\text{N} = \text{size\_x}*\text{size\_y}*\text{size\_z}*5*4$$

возможных состояний для агента. Агент взаимодейтсвует с одним из этих состояний и предпринимает решение, какое действие ему принять дальше.

После того, как было задано количество состояний, нужно учесть границы среды, чтобы в дальнейшем агент не смог за них выйти. Основную часть данного файла занимает шестивложенный цикл по следующим парметрам:

- 3 пространственных параметра (lay, row, column)
- 2 по состояним объекта и пунктов назначения
- 1 по возможным действиям

Внутри данного шестивложенного цикла происходит заполнение первичной таблицы вознаграждений под названием P. Данная таблица является матрицей, в котором количество столбцов соотвествует числу возможных действий, а количество строк соотвествует количеству состойний. На рисунке ниже представлена данная матрица P при рандомном индексе 442.

```
(0, [(1.0, 442, -10, False)])
(1, [(1.0, 342, -5.0, False)])
(2, [(1.0, 442, -5.0, False)])
(3, [(1.0, 442, -5.0, False)])
(4, [(1.0, 942, -5.0, False)])
(5, [(1.0, 442, -10, False)])
(6, [(1.0, 442, -10, False)])
(7, [(1.0, 442, -10, False)])
```

Рис. 4: P[442]

Как интерпретировать эти данные?

(action, [(probability, nextstate, reward, done)]),

Причем,

- значения 0 - 7 соответсвуют действиям (south, north, east, west, move up, move down, pickup, dropoff)

- done характеризует результат доставки объекта в пункт назначения

Каким же образом заполняется данная таблица? В шестивложенном цикле существует проверка на то, какое действие совершается и в зависимости от результата дейтсвия агент получает опредленное количество очков. Новое состояние получается при помощи функции encode и пяти параметров:

- 3 пространственных (new_lay, new_row, new_col)
- расположения объекта (new_pass_idx)
- расположения пункта назначения (desc_idx)

В данном файле так же представлена функция render(), которая реализуют 2D-отрисовку перемещения агента в среде. В render используется decode(), которая преобразует входные данные в расположение агента, объекта и пунктов назначения при визуалиции. Ниже представлено несколько последовательных расположений агента в среде в виде куба со стороной 5:

Задача для агента формулируется следующим образом: "Доставить объект из пункта А в пункт Б с минимальными затратами топлива."

5.2 Основной алгоритм

Прежде всего, нужно заметить, что данную задачу можно решить другим способом без машинного обучения. С помощью цикла while можно написать алгоритм, который реализовывал бы доставку объекта и одной точки локации в другую, но, очевидно, что данный алгоритм был бы совершенно не эффективен на сетках любой размерности. Теперь перейдем к описанию самого алгоритма.

Используя среду, которую я описал в предыдущем подпункте и gym, я реализовал Q-learning алгоритм для поставленной задачи. Перед тем, как описывать реализацию алгоритма необходимо описать несколько полезных функций, которые были разработаны OpenAI. Прежде всего, env = gym.make() - это сердце OpenAI Gym, представляет собой интерфейс среды. У env есть несколько полезных методов:

- env.step(action) - продвигает развитие оружающей среды на один шаг по времени
- env.reset - перезапускает среду, то есть перезапускает исходную среду и возвращает новое случайное исходное состояние

Напомню основные детали Q-learning метода. Среда вознаграждает агента за постепенное обучение и за то, что в конкретном состоянии он совершает наиболее оптимальный шаг. В предыдущем подпункте я вводил таблицу P, по которой будет учиться агент. Опираясь на таблицу вознаграждений, он выбирает следующее действие в зависимости от того, насколько оно затратно, а затем обновляет величину, именуемую Q-значением. В результате создается новая таблица (Q-таблица), отображаемая на комбинацию (State, Action). Если Q-значения оказываются лучше, то получаются более оптимизированные вознаграждения. Например, если агент с объейктом находится в точке, в которой нужно выложить объект, то Q-значение для "dropoff"оказывается выше, чем для остальных действий. При взаимодейтсвии со средой Q-значение в Q-таблицы обновляется на основе следующей формулы:

$$Q(state, action) = Q(state, action) + \\ + \alpha * [R(state, action) + \gamma * maxQ(state', action') - Q(state, action)],$$

где α, γ - параметры Q-learning. α - это темп обучения, а γ - дисконтирующий множитель. Гамма определяет, какую мы хотим придать важность вознаграждениям, ождиюащим нас в перпективе.

Так же, чтобы агент был "любопытным"вводится параметр ϵ, отвечающий за так называемый exploration, то есть за исследование среды.

5.3 Вспомогательный файл от OpenAI

6. Результаты работы

7. Вывод

8. Список используемой литературы

[1] Комаров А. Ю., Метод обучения с подкреплением для архитектуры вероятностных автоматов.
[2] Князятов С.А., Малинецкий Г.Г., Решение задачи распознавания блефа в игре «верю – не верю» с помощью алгоритмов обучения с подкреплением // Препринты ИПМ им. М.В.Келдыша. 2018. No 170. 21 с.
[3] André Barreto, Will Dabney, Rémi Munos, Jonathan J. Hunt, Tom Schaul, Hado van Hasselt, David Silver, Successor Features for Transfer in Reinforcement Learning
[4] André Barreto, Will Dabney, Rémi Munos, Jonathan J. Hunt, Tom Schaul, Hado van Hasselt, David Silver, Successor Features for Transfer in Reinforcement Learning
[5] Romain Laroche, Merwan Barlier, Transfer Reinforcement Learning with Shared Dynamics
[6] Саттон Р., Барто Э. Обучение с подкреплением – Бином. Лаборатория знаний, 2012. – 400 с.