Федеральное государственное бюджетное образовательное учреждение
высшего образования «Московский Государственный Университет
им. М. В. Ломоносова»
Механико-математический факультет
Кафедра теоретической механики и мехатроники

Выпускная квалификационная работа (Дипломная работа) специалиста

«Применение генетического алгоритма для управления подвижным экипажем»

Выполнил студент
622 группы
Кафанов Олег Игоревич

Научные руководители:
к. ф.-м. н. А. В. Шокуров

д. ф.-м. н. И. И. Косенко

Москва
2019

Содержание

Введение

Актуальность

Обучение с подкреплением набирает всё большую популярность в задачах, где сложно построить математическую модель. Или в реальных задачах, где необходимо, чтобы система работала на реальных данных, подстраиваясь под изменчивую окружающую среду. Одной из таких задач является пилотирование автомобиля.

Беспилотные автомобили - актуальная задача на сегодняшний день. Согласно официальным данным, по состоянию на 1 января 2017 года испытаниями беспилотных автомобилей на дорогах общего пользования в Калифорнии занималась 21 компания. Среди них Google, Tesla, Bosh, Mercedes-Benz и другие. Яндекс.Такси уже запустили в Иннополисе и Сколково беспилотные автомобили, которыми может воспользоваться любой желающий через приложение.

Краткое содержание работы

Работа состоит из четырех разделов.

В первом разделе приводится постановка задачи обучения с подкреплением. Описывается геймплей, среда и агент.

Второй раздел содержит краткий обзор существующих решений и алгоритмов для аналогичных задач.

Третий раздел посвящен описанию разработанного алгоритма. Структуры нейросети и методу обучения с помощью генетического алгоритма.

В четвертом разделе описываются эксперименты. Приводятся графики. Оцениваются плюсы и минусы алгоритма.

1. Постановка задачи

Для обучения и тестирования алгоритма выбрем OpenAI Gym - это популярный репозиторий с открытым исходным кодом для обучения с подкреплением (RL). Его набор проблем и простота использования сделали его стандартным инструментом для разработки алгоритмов RL. Кроме того, его задачи предназначены для вычисления на современном оборудовании потребительского уровня.

1.1. Обучение с подкреплением

В OpenAI Gym имплементирована среда по принципу классического цикла агент-среда (Рис.1). Агент и Среда играют ключевые роли в алгоритме обучения с подкреплением. Среда – это тот мир, в котором приходится выживать Агенту. Кроме того, Агент получает от Среды подкрепляющие сигналы (вознаграждение): это число, характеризующее, насколько хорошим или плохим можно считать текущее состояние мира. Цель Агента — совершая действия, максимизировать совокупное вознаграждение.

1.2. Геймплей и вознаграждение

Задача среды "CarRacing-v0" - пилотировать автомобиль через случайно сгенерированный двумерный мир трека, травы и границ, достигнув конца трассы за максимально короткое время. Трек разбит на фрагменты - плитки. Среда поощряет движение по дороге и позволяет избежать границ (окончание эпизода и вознаграждение -100 при столкновении с границами); Вознаграждение составляет $-0,1$ за каждый кадр и $+1000/N$ за каждый посещенный фрагмент трека (плитку), где N - об-

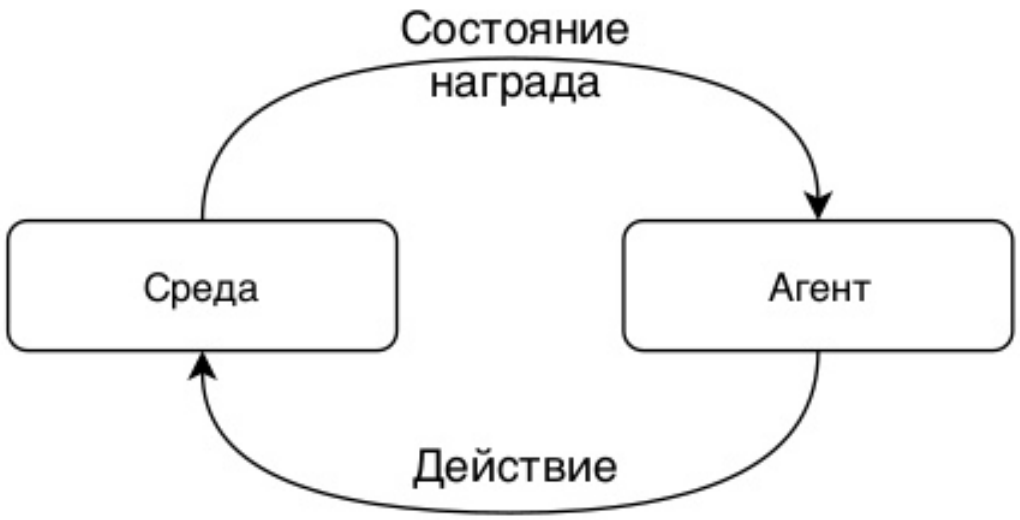


Рис. 1. Классический цикл агент-среда

щее количество фрагментов в треке. Например, если вы финишировали на 732 кадре, ваше вознаграждение составит $1000 - 0,1 * 732 = 926,8$ балла. Эпизод заканчивается, когда все плитки посещены. На траве машина, как правило, выходит из-под контроля, заставляя много времени тратить впустую, поэтому я внес небольшие изменения: окончание эпизода и вознаграждение -50 в случае, если машина долго (100 шагов) получает отрицательное вознаграждение.

1.3. Пространство состояний

Пространство состояний представляет собой последовательность кадров для игрового экрана, каждый представлен в виде сетки $96 \times 96 \times 3$ значений RGB. При отсутствии упрощающих предположений, размер пространства состояний очень велик $256^{3 \cdot 96^2}$. Можно значительно сокра-

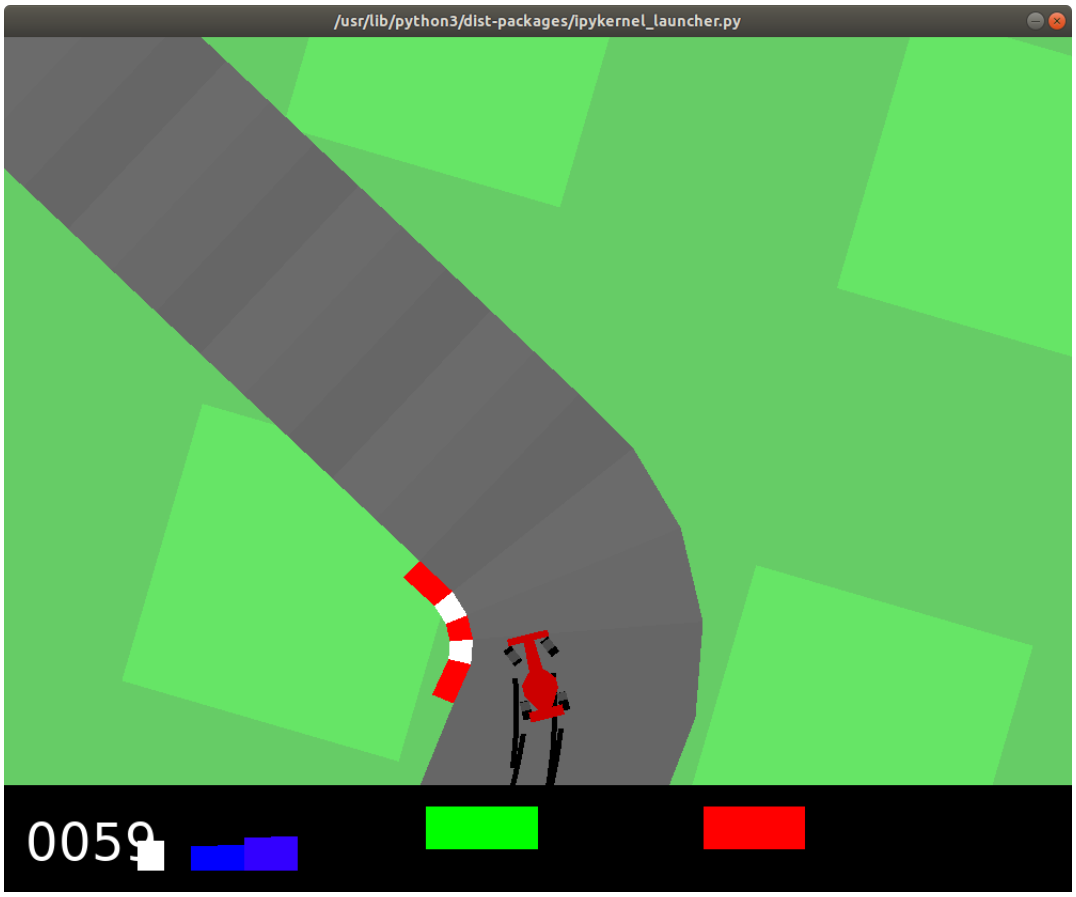


Рис. 2. Геймплей

тить его, предварительно обработав изображение, не жертвуя при этом практически никакой информацией, относящейся к «водителю».

Методы для этого, используемые далее в §3.1., перечислены здесь:

- Уменьшение разрешения и обрезка изображения.
- Классификация пикселей только как «дорожные» или «не дорожные», используя их цвет/интенсивность. Это уменьшает основание степени до 2.

1.4. Пространство действий

Пространство действий представляет собой множество троек $(s, a, d) \in [-1, 1] \times [0, 1] \times [0, 1]$, где рулевое управление коэффициент s колеблется

от крайнего левого до крайнего правого, газ a от нулевого до полного ускорения, и тормоз d колеблется от нуля до полного торможения.

Чтение исходного кода показывает, что в человеческой версии игры (управляется стрелками) действия дискретизированы, стрелка влево: $s = -1$, стрелка вниз: $d = 1$ и т. д. Так как можно добиться хороших результатов с помощью клавиш со стрелками, то есть возможность существенно ограничить пространство действий
$A = \{(-1, 0, 0), (1, 0, 0), (0, 1, 0), (0, 0, 0.8), (0, 0, 0)\}$ - влево, вправо, ускорение, замедление, ничего.

2. Существующие методы

2.1. q-learning и другие табличные методы

Размер нашего пространства состояний немедленно исключает несколько категорий алгоритмов RL:

• Методы, основанные на максимальном правдоподобии, исключены, поскольку для любого момента времени очень вероятно, что мы никогда раньше не наблюдали точно такой же экран, а это означает, что подавляющее большинство наших оценок будет неопределенным.

• Также исключаются методы, основанные на таблицах, такие как Q-learning и Sarsa [6]. Пространство пар «состояние-действие» (s, a) слишком велико для хранения в табличной форме. Использование разреженных матричных представлений для Q-таблиц не поможет, так как не решает основную проблему - подавляющее большинство значений $Q(s, a)$ (функции отражающей ценность каждого возможного действия a агента для текущего состояния s, в котором сейчас находится симуляция)

никогда не дождутся обновления.

2.2. Методы Policy Gradient

Цель обучения с подкреплением - найти оптимальную стратегию поведения для агента, чтобы получить максимальное вознаграждение. Методы Policy Gradient нацелены на моделирование и оптимизацию политики напрямую. Политика обычно моделируется параметризованной функцией $\pi_\theta(a, s)$ относительно параметра θ. Политика – это то, что определяет поведение системы в данный момент времени. $\pi_\theta(a|s)$ - стохастическая политика (стратегия поведения агента, вероятность совершить действие a в состоянии s). Значение функции вознаграждения зависит от политики. Для получения наибольшего вознаграждения могут быть применены различные алгоритмы оптимизации по θ .

Функция вознаграждения определяется как:

$$J(\theta) = \sum_{s \in S} d^\pi(s) V^\pi(s) = \sum_{s \in S} d^\pi(s) \sum_{a \in A} \pi_\theta(a|s) Q^\pi(s, a)$$

где $d^\pi(s)$ - стационарное распределение цепи Маркова для π_θ, $V^\pi(s)$ - функция значения состояния измеряет ожидаемый возврат состояния s при условии следования политике, $Q^\pi(s, a)$ - функция «действие-значение» аналогична функции $V^\pi(s)$, но она оценивает ожидаемый возврат пары состояния и действия (s, a), S - пространство состояний, A - пространство действий.

Представьте, что вы можете путешествовать по состояниям цепи Маркова вечно, и в конце концов, вероятность того, что вы окажетесь в одном состоянии, со временем станет неизменной - это стационарная вероятность для π_θ. $d^\pi(s) = \lim_{t->\infty} P(s_t = s|s_0, \pi_\theta)$ - это вероятность

того, что $s_t = s$ при запуске с s_0 и следовании политике π_θ для t шагов.

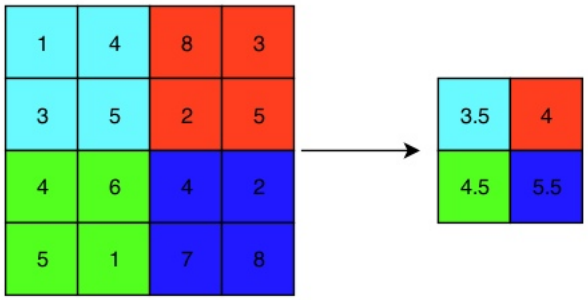
Естественно ожидать, что основанные на политике методы более полезны в непрерывном пространстве. Поскольку существует бесконечное число действий и (или) состояний для оценки значений, и, следовательно, подходы, основанные на значениях, являются слишком дорогими в вычислительном отношении в непрерывном пространстве. Мы можем найти наилучшее θ для π_θ используя градиентный спуск по $J(\theta)$ или при помощи генетического алгоритма. Вид $J(\theta)$ может различаться в зависимости от метода.

3. Генетический алгоритм

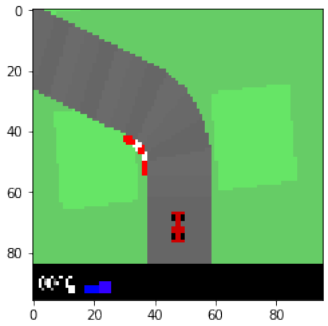
3.1. Структура нейросети

Для сокращения размера $256^{3 \cdot 96^2}$ пространства состояний проведем следующие операции: медианный пулинг 10x10 (Область $N \times N$ пикселей заменяется на один пиксель с медианным значением. Пример медианного пулинга 2x2 представлен на Рис.3), свертку цветного изображения в черно-белое, разделим пиксели на «дорожные» и «не дорожные» по интенсивности цвета, обрежем нижнюю полосу пикселей, не несущую теперь никакой информации, получив новое пространство с количеством элементов 2^{72} (Рис.3). Так как внизу картинки были показания датчиков, но инфомация с них была потеряна в результате преобразований, то показания передадим из среды отдельным каналом.

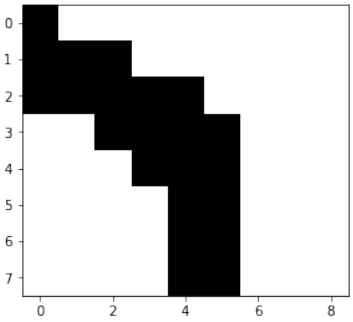
Итак, после обработки исходного изображения, получили изображение 8×9 или, иначе говоря, массив из 72 чисел. Эти числа (а также показания датчиков), подаются на входной слой перцептрона (Рис.4).



Медианный пулинг 2x2



до обработки

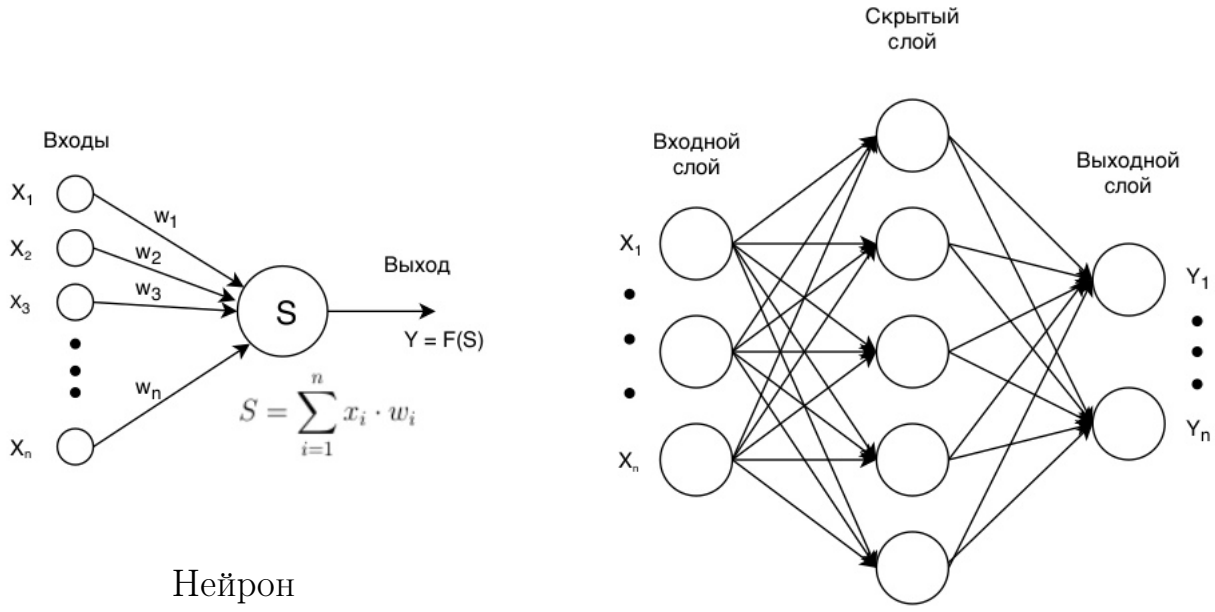


после обработки

Рис. 3. Сокращение пространства состояний

В случае дискретного пространства действий, количество выходов сети равно количеству возможных действий (Например, для $A = \{(-1, 0, 0), (1, 0, 0), (0, 1, 0), (0, 0, 0.8), (0, 0, 0)\}$ равно 5: $(Y_1, Y_2, Y_3, Y_4, Y_5)$) и выбирается действие соотвествующее наибольшему значению на выходе - $argmax(Y_1, Y_2, Y_3, Y_4, Y_5)$

В случае непрерывного пространства действий, т.е. всего множества $[-1, 1] \times [0, 1] \times [0, 1] \ni (s, a, d)$ к трем выходам Y_1, Y_2, Y_3 применяется сигмоида $\sigma(x) = \frac{1}{1+e^{-x}}$. Так как $\sigma(x) \in (0, 1)$, то $(2\sigma(Y_1) - 1, \sigma(Y_2), \sigma(Y_3)) \in [-1, 1] \times [0, 1] \times [0, 1]$



Нейрон

Однослойный перцептрон

Рис. 4. Схема нейрона и однослойного перцептрона

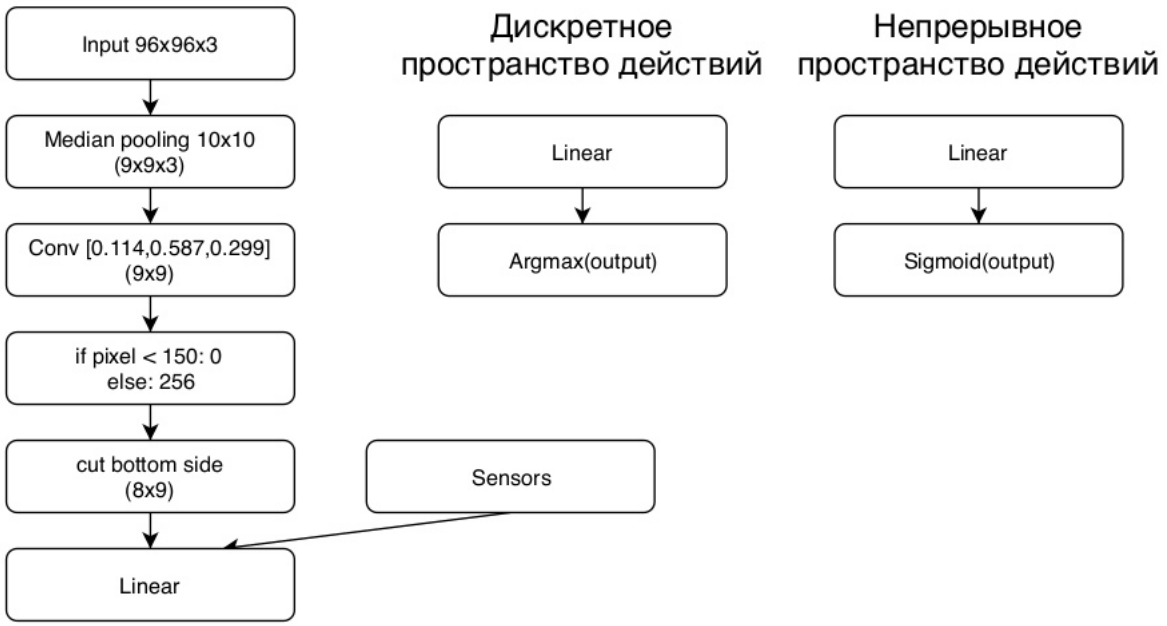


Рис. 5. Схема нейросети

3.2. Метод обучения

Для обучения перцептрона применяется генетический алгоритм. Геномом является вышеописанная сеть, полная схема которой представлена на Рис.5 (Linear - однослойный линейный перцептрон Рис.4).

Для каждого генома из поколения запускается среда и считается функция приспособленности (суммарное вознаграждение). Затем в новое поколение выбираются представители с наибольшим значением ФП: геном с номером i берется с вероятностью $\frac{n-i}{n}$, где n - количество геномов в поколении, i - порядковый номер элемента в списке, отсортированном по убыванию ФП.

Недостающие до n геномы добираются потомками от выживших (причем снова с большей вероятностью от тех, у которых больше ФП)

Всё повторяется для нового поколения

```
1: procedure GENETIC ALGORITHM
2:     for all generation do
3:         for all genome do
4:             run simulation
5:             for all step do
6:                 totalReward+ = reward
7:             end for
8:             fitness ← totalReward
9:         end for
10:        CreateNewGeneration()
11:    end for
12: end procedure
```

```
1: procedure CREATE NEW GENERATION
2:     sort population
3:     n ← length(population)
4:     for i < n do
5:         if random(0, 1) < (n−i)/n then
6:             nextGen.append(population[i])
7:         end if
8:     end for
9:     while length(nextGen) < n do
10:        nn1, nn2 ← select from nextGen
11:        nextGen.append(CreateChiled(nn1, nn2))
12:    end while
13:    population ← nextGen
14: end procedure
```

Потомок создается от пары выживших представителей. Обозначим r_m (mutation rate) - вероятность мутации, F_1, F_2 - значения функции приспособленности для первого и второго предствителя.

Для потомка каждый вес выбирается по следующему правилу: с вероятностью r_m берется случайный вес, с вероятностью $(1 - r_m) \cdot \frac{F_1}{F_1+F_2}$ выбирается вес первого родителя на соответствующей позиции, с вероятностью $(1 - r_m) \cdot \frac{F_2}{F_1+F_2}$ - вес второго.

```
1: procedure CREATE CHILD(nn1, nn2)
2:     init all weights as random
3:     for all weight do
4:         if random(0, 1) > mutation_rate then
5:             if random(0, 1) < nn1.fitness/(nn1.fitness+nn2.fitness) then
```

6: $weight \leftarrow nn1.weight$

7: **else**

8: $weight \leftarrow nn2.weight$

9: **end if**

10: **end if**

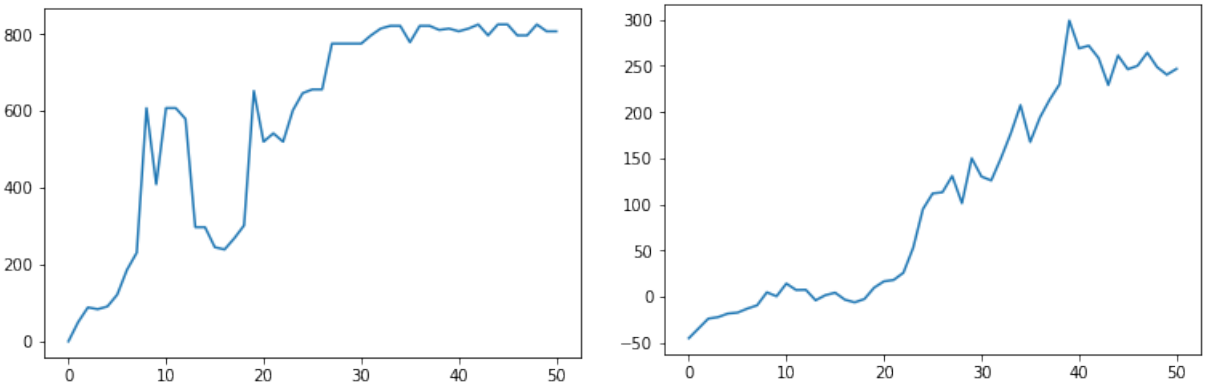
11: **end for**

12: **end procedure**

4. Эксперименты

На графиках (Рис.6) предствлена зависимость функции приспособленности (суммарного вознаграждения) от поколения, максимальная и средняя по всем представителям поколения.

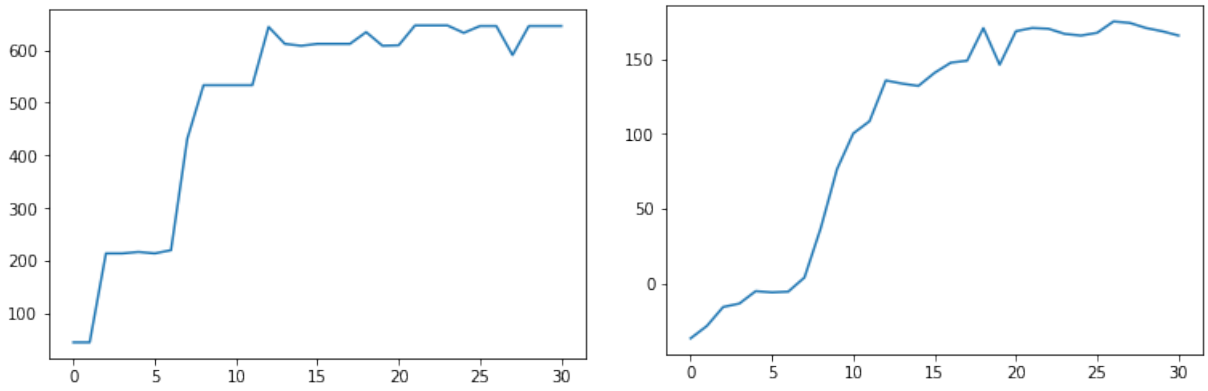
Количество геномов в популяции - 100. Пространство действий - непрерывное.



max fitness avg fitness

Рис. 6

Видно, что алгоритм сошелся к довольно хорошему качеству. Учитывая, что масимально возможное вознаграждение чуть больше 900 и

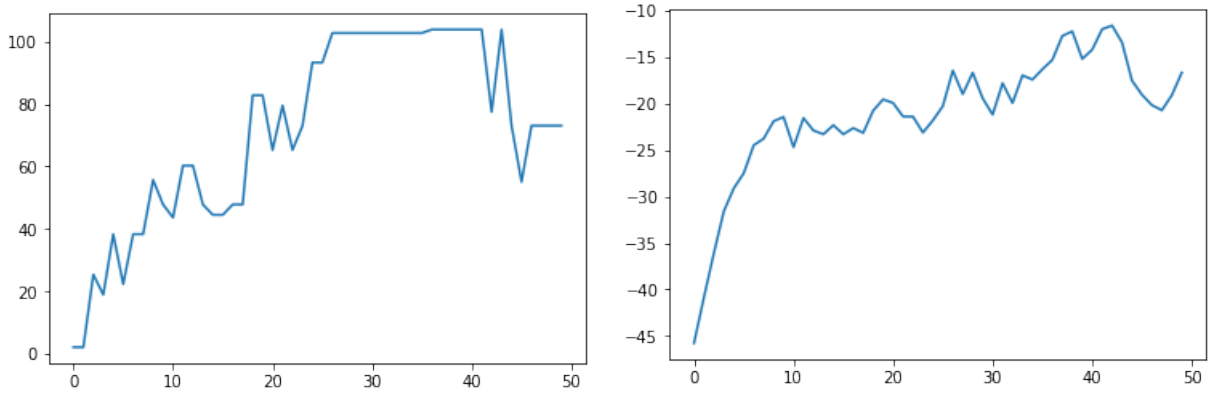


max fitness avg fitness

Рис. 7

так как эпизод не был закончен - дополнительные -50. Просматривая повтор на видео, можно увидеть, что автомобиль пропустил лишь несколько плиток трека.

Но не всегда удается сойтись к глобальному максимуму, и как можно заметить на Рис.7,8 алгоритм застрял в локальном максимуме.



max fitness avg fitness

Рис. 8

Выбор непрерывного или дискретного пространства действий отражается на "стиле вождения". В случае непрерывного у "водителя"большая вариативность действий, но при этом он совершает много "лишних дви-

жений". В дискретном случае движения плавнее.

4.1. Сравнение с Proximal Policy Optimization [5]

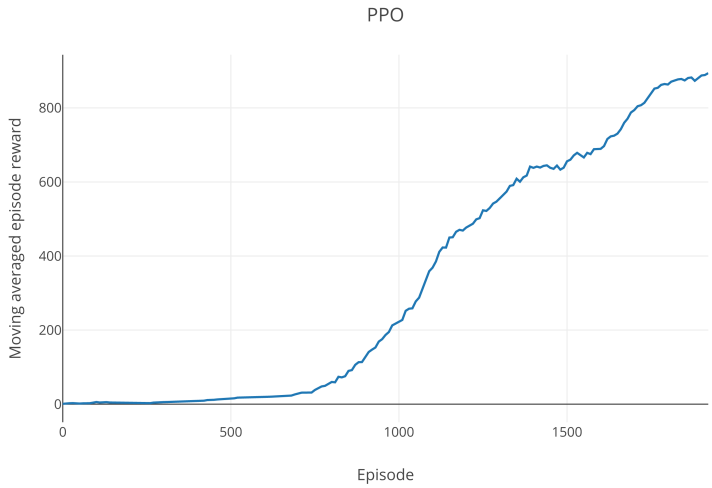


Рис. 9. avg fitness PPO

Сравнивая графики Рис.6 и Рис.9 можно сделать следующие выводы:

В лучшем случае достигаемое качество почти не различается.

Для 100 геномов в поколении скорость работы у генетического алгоритма ниже - к 40 поколению было запущено $40 \cdot 100 = 4000$ эпизодов. Тогда как PPO сошелся у 2000 эпизодов.

Но генетический алгоритм можно хорошо распараллелить на этапе генерации трека, симуляции и подсчете функции приспособленности, что сделает его в разы быстрее PPO.

Также плюсом является то, что ГА не использует производную функции приспособленности, но лишь саму функцию. Это может быть полезно в задачах, где производную вычислить проблемно.

Главный недостаток генетического алгоритма - застревает в локальных максимумах. Сильно зависит от выбора начальных весов.

Заключение

Данная работа посвящена описанию способа применения генетического алгоритма в задаче обучения с подкреплением. Был описан метод и приведены результаты экспериментов. Также приведен пример другого алгоритма, решающего данную задачу и проведено сравнение.

Список литературы

1. Pablo Aldape Samuel Sowell *Reinforcement Learning for a Simple Racing Game*
   Department of Statistics Stanford University, Department of Electrical Engineering Stanford University, December 8, 2018
2. Henry Teigar, Miron Storožev, Janar Saks *2D Racing game using reinforcement learning and supervised learning*
   University of Tartu, Institute of Computer Science Neural Networks
3. John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov *Proximal Policy Optimization Algorithms*
   OpenAI, 28 Aug 2017, arXiv:1707.06347v2
4. Zhangbo Liu *A Guided Genetic Algorithm for the Planning in Lunar Lander Games*
   Department of Computer Science University of British Columbia
5. *Proximal Policy Optimization*
   https://github.com/xtma/pytorch_car_caring
6. Richard S. Sutton and Andrew G. Barto A Bradford Book
   Reinforcement Learning: An Introduction
   The MIT Press Cambridge, Massachusetts London, England
7. Szasz Janos *The algorithmicx package*

The April 27, 2005

8. Генетические алгоритмы как конкурентная альтернатива для обучения глубоких нейросетей
   https://habr.com/ru/post/345950/
9. Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, Joelle Pineau *An Introduction to Deep Reinforcement Learning*
   arXiv:1811.12560v2 [cs.LG] 3 Dec 2018