Преддипломен проект

Решаване на оптимизационната задача FrozenLake с невробиологичен симулатор по метода поощрение/наказание

**Факултет по Математика и Информатика**  
**Студент:** Борислав Стоянов Марков  
**Факултетен номер:** **0MI3400048**  
**Учебен план:** **Изкуствен Интелект  
Научен ръководител**: Проф. Петя Копринкова-Христова, Институт по Информационни и Комуникационни Технологии (ИИКТ), БАН

София: 01.2023г.

# 1. Съдържание

[1. Съдържание 2](#_Toc125403432)

[2. Увод 2](#_Toc125403433)

[3. Средата „FrozenLake“[4] в Gym 3](#_Toc125403434)

[4. Неврологични мрежи 4](#_Toc125403435)

[4.1 Основи на неврологията 4](#_Toc125403436)

[4.2 Математически апарат на невроните 5](#_Toc125403437)

[4. Алгоритъм Актьор-критика 6](#_Toc125403438)

[5. Реализация на проекта 9](#_Toc125403439)

[5.1 Анализ на резултатите 10](#_Toc125403440)

[6. Идеи за бъдещо развитие и подобрения 12](#_Toc125403441)

[7. Източници и използвана литература 12](#_Toc125403442)

[Приложения 12](#_Toc125403443)

[1. Сорс код (Source code) 12](#_Toc125403444)

# 2. Увод

Невробиологията все повече набира скорост в света на изкуствения интелект. Има все повече изследвания на функционирането на нервни клетки, които са довели до създаването на биологично обоснованите spike timing модели на невроните, както и много знания за структурната организация и функционирането на мозъка на бозайниците при вземане на решения. Доказано е, че много от решенията се вземат по метода на поощрението и наказанието (Reinforcement Learning).

Целта на преддипломния проект е да се разработи модел на биологично обоснована (spike timing) невронна мрежа посредством библиотеката NEST Simulator, която е в състояние да решава оптимизационната задача за преминаване на агент през известната среда FrozenLake от пакета Gym посредством reinforcement обучение. Задачата е с дискретни състояния на средата и 4 възможни действия на агента.

Проектът ще включва кратък обзор в областта на Spike Timing Neural Networks, описание на теоретичната постановка, код на Python с използване на библиотеката NEST Simulator и анализ на резултатите. В процеса на изработка на проекта ще се изпробват различни параметри на биологично подобните неврони и решението ще бъде илюстрирано с подходящи визуализации и графики съпътстващи обучителния процес.

# 3. Средата „FrozenLake“[4] в Gym

Средата Frozen Lake е част от подпакета а текстови игри. Това са много опростени игри с визуализация като текст. Имат малък на брой дискретни състояния и контрола на агента е също дискретен с малък брой дествия. Този пакет е разработен за обучителни цели по метода на поощрението и наказанието (Reinforcement Learning). Целта е да се сравняват различни решения при една и съща среда и да може да се прави сравнение на резултатите. На следващата фигура е показана примерна визуализация на тази среда. Имаме езеро, правоъгълна фигура, разграфена на квадранти, например 4х4 с кодове латинските букви S,F,H,G.

|  |  |
| --- | --- |
|  |  |

Фигура 3.1 – Примерна визуализация на FrozenLake вляво и текстов еквивалент вдясно.

Значенията на квадрантите са старт-S, заледен-F, дупка-H, цел-G. Човече тръгва от позиция „S” и трябва да стигне до крайната цел „G”, преминавайки по заледени участъци „F“. Някои от квадрантите имат дупки “H” и там текущият опит приключва неуспешно. Поощрението и наказанието е както следва:

* При достигане на крайната цел „G” награда от 1 точка и експериментът приключва.
* При попадане в дупка “H” награда „0“ и експериментът приключва.
* При попадане на заледен участък „F“ награда „0“.

Има една особеност на средата, че на заледените участъци агентът може да се подхлъзне и да не отиде в избраната посока, тоест имаме хлъзгане. Хлъзгането зависи с какви настройки е пусната средата: is\_slippery=False|True. Основна информация за средата са дадени в следващата таблица.

|  |  |
| --- | --- |
| Пространство на действията | Discrete(4) |
| Вектор на наблюдението | Discrete(16) |
| Импортиране в Питон | gym.make("FrozenLake-v0") |

Таблица 3.1. Общи параметри за средата

Човечето може да се придвижва в четири посоки, всяка с код от 0 до 3 включително, а именно: наляво-0, надолу-1, надясно-2, нагоре-3.

# 4. Въведение в неврологичните симулации

Неврологията е дисциплина която може да засяга един или друг аспект свързан с работата на нервната система при живите организми. За нашите цели по-интересно е математическият апарат, отколкото биологичната основа. Най-забележимият аспект на нервната система при човека и при други живи организми е начинът на вземане на решения и поразяващи имплементации на алгоритми с поощрение и наказание. Връзката между реинфорсмънт обучението (Reinforcement learning) и новрологичните науки се крие в химично имплементираната награда – допаминът ([1] глава 15). Допаминът пренася темпоралната грешка (TD error) до структурите в мозъка, които са отговорни за вземане на решение.

## 4.1 Основи на неврологията

Невроните са основните компоненти на нервната система. Това са клетки специализирани в пренасяне и обработка на информация посредством електрохимически и химически сигнали [1]. Невроните са различни по вид, но обикновено имат тяло, дендрити и един аксон. Дендритите са разклонения от тялото, чрез които клетката се свързва с аксони от други неврони или са сензори, в случай на сензорни неврони.

Невронът събира импулси от много входове и когато сумарно тези входове станат забележително големи и преминат някаква граница, невронът изстрелва потенциал(action potential) или тъй наречения импулс (spike). Това е и фундаменталният начин на комуникация между невроните ([2], Ch.2). Изходният сигнал на неврона са електрически импулси, пътуващи по аксона, наречени спайкове (spikes).



Фиг.4.1.1. Рисунка на два свързани неврони и ин-витро записан спайк  
Адаптирано от [3], Глава 1.1

## 4.2 Математически апарат на невроните

Към момента невроните се моделират по много различни начини и има описани десетки видове диференциални уравнения на различни неврони. Един от първите математически формализми на неврони е на Ходжкин и Хъксли, описан през петдесетте години на миналия век. Уравнението се оказва доста сложно за решаване на практически задачи и затова по-късно са предложени опростени модели. Integrate-and-fire са фамилия модели от които най-популярният към момента е leaky integrate-and-fire неврон. Това е идеализация на неврон с утечки(по закона на Ом) , който е суматор на токове.

|  |  |
| --- | --- |
|  | (4.2.1) |

Тук формализма е взет от [3], C е капацитетът на мембраната на неврона, V е мембранния потенциал, gleak e функция на утечка, Eleak e средно напрежение на мембраната в покой. На следващата фигура се онагледява действието на неврона.



Фиг. 4.2.1 Потенциална диаграма на суматорен неврон с утечка (Leaky integrate-and-fire neuron). Адаптация от [3].

На фигурата се вижда, че при сумарно напрежение надвишаващо Ethresh се произвежда токов импулс (spike) и напрежението се връща до стойност Ek. При липса на входни токове или шум се забелязва как напрежението намалява експоненциално, стремейки се към Eleak.

## 4.3 Невробиологичен симулатор NEST

NEST е симулатор за невронни мрежи основани на спайкове (spiking neural network models или SNN) и може да послужи за: модели за обработка на информация, модели на мрежова динамика, модели на обучение и синаптична пластичност [5]. Симулаторът представлява библиотека на Python и може да се вгради в по-голямо приложение. Структурата на програмата е например:

|  |
| --- |
| 1. Подготовка на невронните групи и шумогенератори 2. Свързване на групите неврони с определено тегло 3. Поставяне на виртуални измервателни уреди и свързване с интересуващите ни групи неврони; 4. Симулиране с метода nest.Simulate(t), който проиграва една евентуална симулация за време t милисекунди. 5. След това може да се наблюдават измервателните уреди(волтметри, детектори на импулси), да се променят теглата на връзките и други параметри и може отново да се премине към стъпка 4 или към край на програмата. |

Таблица 4.3.1 Структура на програма използваща NEST simulator

Можем да разгледаме примерен код на програма, състояща се от един неврон, на който подаваме постоянен ток от 376 [pA] и замерваме напрежението на мембраната, както и генерираните импулси.

|  |
| --- |
| import nest  import matplotlib.pyplot as plt  import nest  nest.ResetKernel()  neuron = nest.Create("iaf\_psc\_alpha", {"I\_e":376.0})  voltmeter = nest.Create("voltmeter")  spikerecorder = nest.Create("spike\_recorder")  nest.Connect(voltmeter, neuron)  nest.Connect(neuron, spikerecorder)  nest.Simulate(1000.0)  plt.rcParams["figure.figsize"] = (5,2)  nest.voltage\_trace.from\_device(voltmeter)  plt.show()  nest.raster\_plot.from\_device(spikerecorder, hist=False, title="spikerecorder")  plt.show() |

Таблица 4.3.2 Примерна програма използваща NEST simulator с един неврон и симулация от 1000 милисекунди

Изходът от програмата са две графики изобразяващи скоковете в напрежението и генерираните импулси.

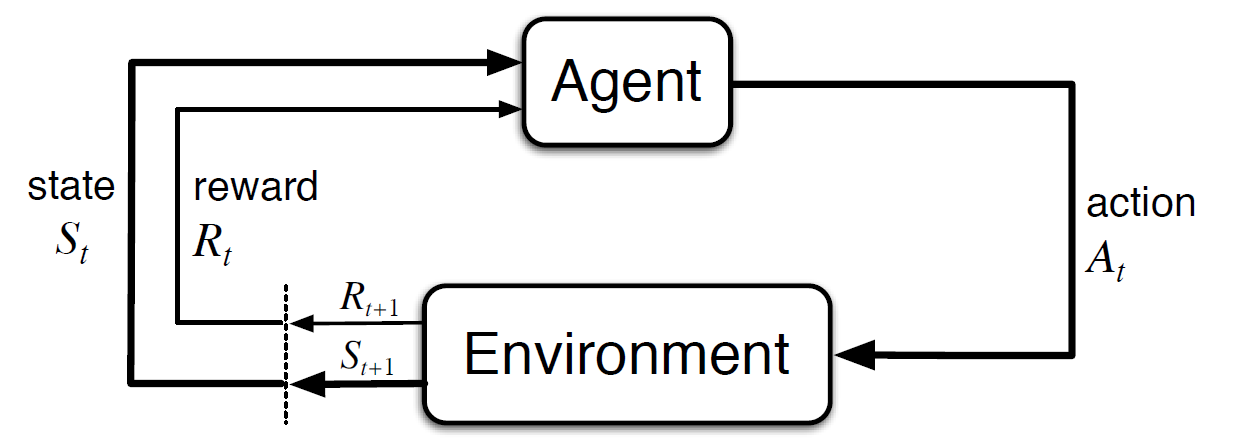
|  |  |
| --- | --- |
|  |  |

Фигура 4.3.1 Напрежение на мембраната и импулси от неврона

При по-голям ток, импулсите ще следват по-бързо, при по-малък ток на неврона импулсите ще намалеят или ще изчезнат.

# Подход за решаване на задачата

FrozenLake е задача, което съвпада с формализма въведен за Reinforcement Learning и можем да кажем, че имаме агент и среда, взаимодействащи си чрез марковски процес. След всяко действие на агентът At средата ни отговаря с ново състояние St+1 и се дава съответната награда Rt+1.



Фиг. 5.1. Взаимодействие Агент-Среда при марковски процес на решението, вж.[1]Глава 3.1

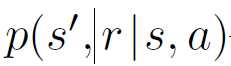
Динамиката на марковският процес на решенията(MDP) е определен от следната функция, даваща връзка между състоянията, наградите и действията:

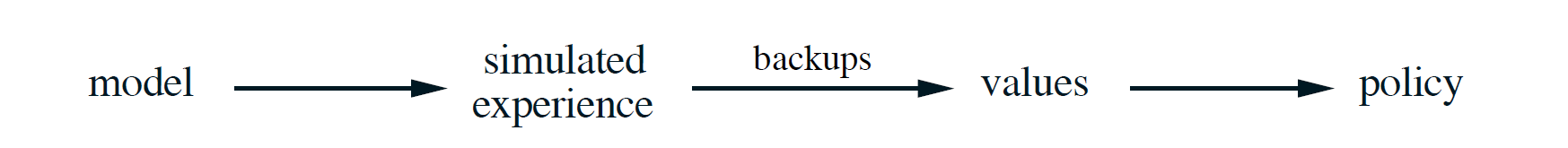
|  |  |
| --- | --- |
|  | (5.1) |

Функцията „p“ е плътност на вероятност при условие, че средата се намира в състояние „s“ и агентът е предприел действие „а“ то средата да премине в ново състояние „s‘” и да има награда ”r”. Знакът за равенство с точка означава „по дефиниция“. Тук интуитивно можем да съпоставим клетка от таблица на всеки квадрант от FrozenLake. Така ще имаме за всяка посока изчислени стойности какво би ни донесло всяко действие по дадената посока.

За някои приложения е подходящо да се направи извадков модел от опита, за други задачи е подходящо да се направи дистрибуционен модел. От многото алгоритми за решаване на подобна задача, подходящо е да се построи дистрибуционния модел MDP(Марковски процес на решенията), тъй като имаме изброими състояния на средата и можем да намерим функцията „p” от (5.1) и ще можем въз основа на нея да дефининираме полица на действията “π”. В случая подходящ алгоритъм е Q-learning от [1]. В началото на пускане на агента не знаем решението и агента трябва да стигне по случаен начин до решението. От това следва, че имаме случаен елемент или т.нар “exploration”, т.е. агента ще действа само на базата случайности. След научаване на правилното решение ще можем да кажем от къде е минал агента и ще можем да следваме модела или ще имаме т.нар. “exploitation”.

# Q-learning

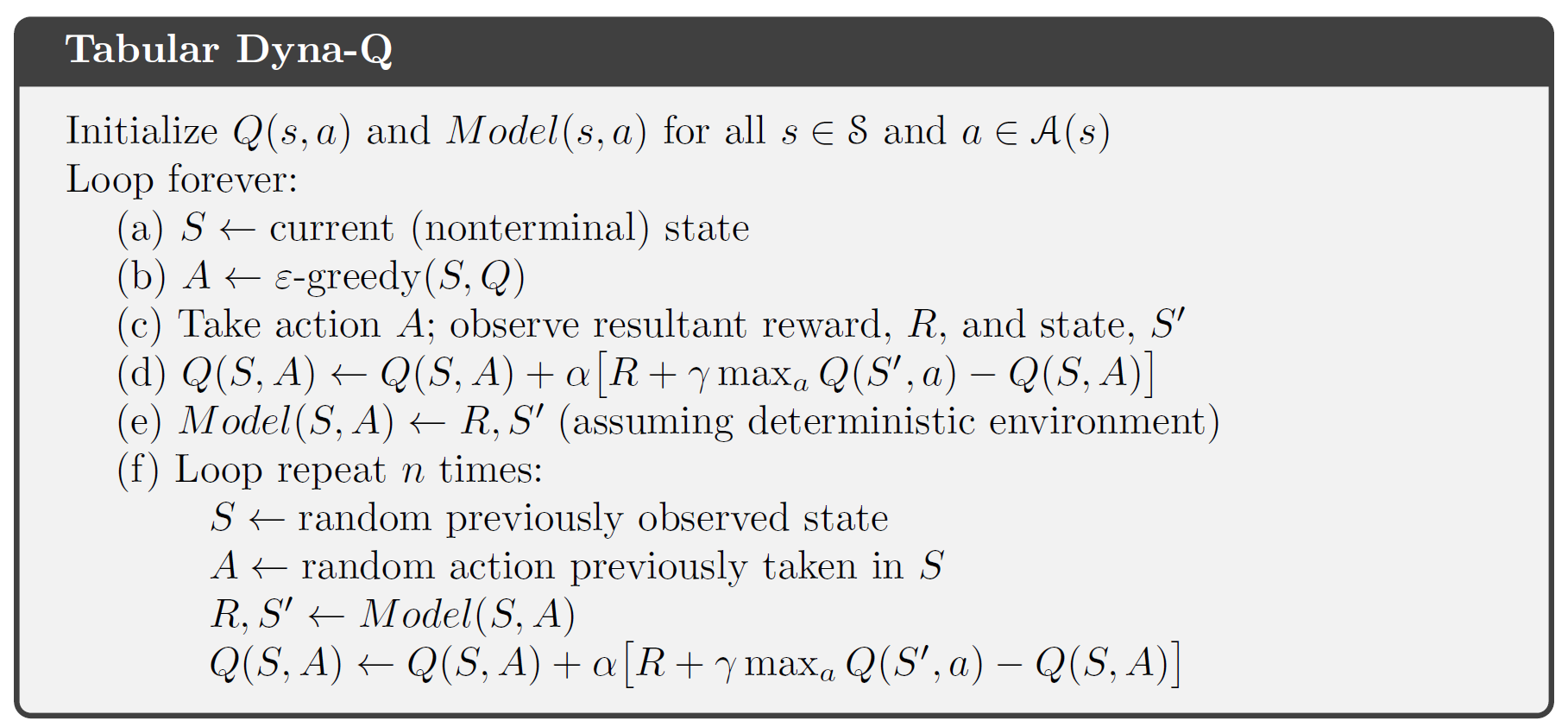
Q-learning са фамилия от алгоритми използващи динамично програмиране за да стигнат до оптималния дистрибуционен модел MDP описан с функцията  от (5.1). Имаме варианти на този алгоритъм, например с модел или без модел. Моделът ни позволява да симулираме средата преди агентът да е действал и да получим симулирана награда.



Фиг.5.1.1 Модел на планиране, вж. [1] глава 8.1

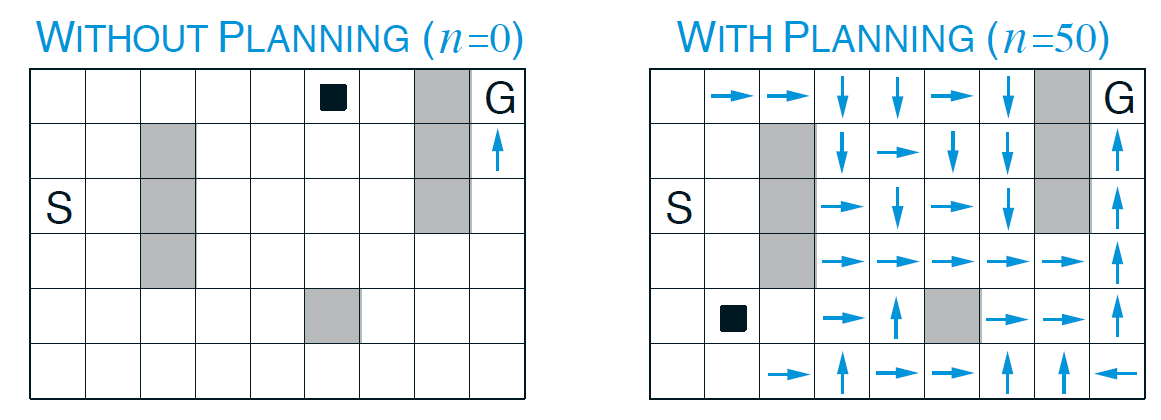
Динамичното програмиране пасва на тази структура като при обхождане на състоянията се генерира дистрибуция на вероятностите от съответните преходи в различни посоки.

Ще разгледаме по-сложният вариант на Q-learning с планиране изложен на следващата фигура.



Фиг. 5.1.2 Алгоритъм на Q-learning с модел, вж. [1] глава 8.1

При n=0 имаме опростеният Q-learning без модел. Барто и Сътън разглеждат в книгата си [1] подобна задача на FrozenLake, която се нарича с общото име Maze. В случая те дават пример на табличното решаване на задачата с модел и без модел.



Фиг.5.1.3 Таблично решаване на задачата Maze без планиране и с планиране, вж. [1] глава 8.1

На по-горната фигура черното квадратче е позицията на агента, стрелките означават действието (ляво,дясно,горе, долу), което максимизира общата награда в края на еизода. В няшият случай FrozenLake може да пусната като детерминирана среда без хлъзгави участъци, но може да бъде пусната и като недетерминирана среда с хлъзгане. Недетерминирана означава, че въпреки, че сме посочили посока, като например надясно, агентът може да премине и надолу в определени случаи, което прави решението много по-трудно.

## 5.2 Победителят печели всичко

При динамичните системи изборът на различни действия при различни параметри понякога може да се окаже проблем, тъй като невронните групи навлизат в стабилен еквилибриум, и не могат да бъдат изместени от него. Това би се изродило в нашият случай като например агентът да отива само надясно да кажем. За тези положения при динамични системи от един неврон се споменава Изикевич в [3] в глава 3.2.6. Наистина за системи с повече неврони теоретичното изчисление на практика е невъзможно и единствено симулациите са водещото, тъй като няма добре установена математическа теория за това. Ще разгледаме механичната интерпретация според [3] на стабилен и нестабилен еквилибриум на следващата фигура.

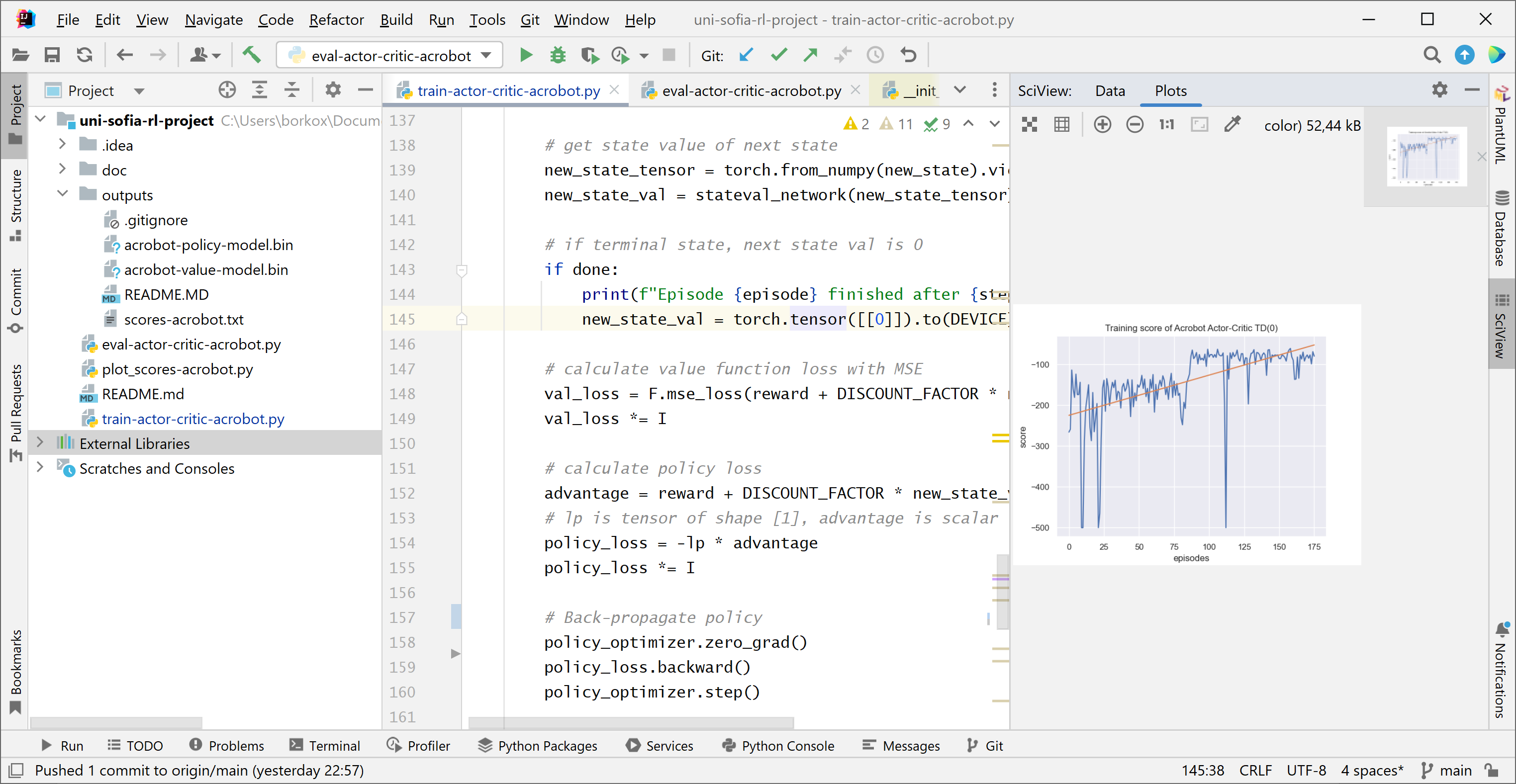


Фиг. 5.2.1 Механична интерпретация на стабилен и нестабилен еквилибриум. Вж. [3].

Топката на фигурата няма маса (без инерция) и се движи към възможно най-ниската точка със скорост пропорционална на наклона. Механичната диаграма се променя по време на симулацията. Искаме да има 4 различни устоичиви положения (по една за четирите възможни действия на агента) тръгвайки от една нестабилна точка. За целта ще използваме WTA (Winner Takes All) схема, което представлява начин на свързване на невронни групи и като цяло е известен подход в невронните мрежи. WTA има един изход от K възможни. Всеки изход е преставен от група неврони с положителна връзка (excitatory neurons).

# 5. Реализация на проекта

Проектът е реализиран като github публичен проект и може да се види дори и през браузър(виж Приложения). За да се пусне локално се изисква инсталация на Python, конкретно 3.9.8 е използван тук. Линк към сорс кода е качен в гитхъб (Вж. Приложение 1) и е неразделна част от този документ. Структурата на приложението е дадена на фигура 5.1. Използваната среда за текстообработка и работа с git е IntelliJ .



Фигура 5.1. Обща структура на проекта

Подробни инструкции на са дадени в README.md файла.

В централната папка има скрипт “train-actor-critic-acrobot.py” на програмния език Python. С него се стартира процеса на обучение. По време на обучение резултатите от точките (поощрението) се записват във файл „outputs/scores-acrobot.txt“ за последваща визуализация. Скриптът „plot\_scores-acrobot.py“ ще ни визуализира картинка с резултатите след текущото обучение. След като сме обучили невронните мрежи можем да пуснем друг скрипт „eval-actor-critic-acrobot.py“, който ще ни визуализира готовото решение и ще проиграе няколко епизода за демонстрация. За край на обучение се приема момента, когато средно аритметичната награда от последните 100 епизода е над -100.

## 5.1 Анализ на резултатите

Като анализ можем да разгледаме графики при различни коефициенти на α(коефициент на обучение) и γ(коефициент на отстъпка). При този тип експерименти точната повторяемост на експериментите е невъзможна, поради случайния характер на актьора π (изборът на действие е с вероятности), случайната инициализация на теглата на невронните мрежи, както и поради случйната подредба на параметрите на средата в Gym при всяко пускане. Следва таблица с графики за различни стойности на параметрите.

|  |  |
| --- | --- |
| α=0.001, γ=0.99 |  |
| α=0.001, γ=0.95 |  |
| α=0.002, γ=0.95 |  |

Таблица 5.1.1

Оранжевата линия представлява линейна апроксимация на всички точки през епизодите. Тя ни показва тренда на обучение. Виждаме, че при α=0.002, γ=0.95 резултатите са най добри и обучението е само за 95 епизода и съответно няма катастрофални забравяния както в първия случай. Наистина този алгоритъм е доста надежден и дава решение в много от случаите. Тук дори не се интересуваме какви са връщаните стойности на състоянието, алгоритъмът би работил с произволни дължини на вектора. Единствено би се наложила промяна при преминаване в непрекъснато пространство на действията.

# 6. Идеи за бъдещо развитие и подобрения

# 7. Източници и използвана литература

[1] Reinforcement Learning: An Introduction, 2018, Richard S. Sutton and Andrew G. Barto, [PDF] <http://www.incompleteideas.net/book/the-book-2nd.html>

[2] Computational Cognitive Neuroscience, Randall C. O’Reilly et al., 2020, Open Textbook, freely available, <https://ccnlab.org/>

[3] Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting, Eugene M. Izhikevich, 2005, The MIT Press, (<https://www.izhikevich.org/publications/dsn.pdf>)

[4] FrozenLake, OpenGym, <https://www.gymlibrary.dev/environments/toy_text/frozen_lake/>

[5] NEST simulator, https://nest-simulator.readthedocs.io/en/v3.3/index.html

[6] Neuronal Dynamics, Wulfram Gerstner, M.Kistler, R.Naud, L.Paninski, 2014, Cambridge university press

# Приложения

## 1. Сорс код (Source code)

<https://github.com/borkox/uni-sofia-rl-project>

В таблици са дадени скриптовете на Питон.

|  |
| --- |
| train-actor-critic-acrobot.py |