# Содержание

Введение

Глава 1. Устройство нейронных сетей

* 1. Общее определение нейросети. ……………………………………………………
  2. Устройство нейросети в этом случае. ...……………………………………………

Глава 2. Создание нейронной сети.

2.1. Реализация нейросети……………………………………………………………2-8

2.2. Реализация среды обучения……………………………………………………......

* 1. Эксперименты с обучением………………………………………………………..
  2. Написание статьи, публикация кода в open source……………………………….

Заключение………………………………………………………………………………….......

Список используемой литературы…………………………………………………………….

Приложение…………………………………………………………………………………10-14

# Введение

**Актуальность:**

Темой для своего проекта я выбрал «Нейронные сети и их обучение». Почему это актуально? За нейросетями будущее: нейросети сейчас применяются в программировании, решении задач компьютерного зрения, data science, аналитической медицине и еще во множестве направлений, а в будущем сфера их применения будет только расти. Возможностей масса: любую задачу, которую может решить человек, может решить и хорошо обученная нейросеть с достаточным количеством нейронов. В общем, нейронные сети сейчас – чуть ли не самое перспективное направление в It. Не смотря на большое количество книг и гайдов по этой теме, нейросети остаются одной из самых сложных тем для понимания, что является **противоречием**. Собственно главной **проблемой** является высокий порог входа в создание нейросетей.

**Цель работы:**

Исследовать процесс конструирования нейронных сетей и их обучения, сконструировать свою сеть, симулирующую деятельность одноклеточного организма.

**Задачи:**

1. Изучить литературу и проанализировать методы инженерии нейронных сетей.
2. Продумать конструкцию примитивной нейронной сети.
3. Реализовать самописную библиотеку для удобной работы с нейросетью.
4. Реализовать среду обучения.
5. Построить графики, демонстрирующие процесс обучения, провести несколько экспериментов.

**Используемые методы:**

Изучение, анализ, синтез и использование изученной информации.

**Описание ожидаемого продукта:**

Нейронная сеть со скрытым слоем, которая будет обучаться за счет положительного и отрицательного подкрепления.

# Глава 1. Устройство нейронных сетей

**1.1 Общее определение нейросети**

Чтобы написать свою нейронную сеть, нужно понять, что это такое. Для начала обратимся к определению: *«нейронная сеть - математическая модель, построенная по принципу организации и функционирования биологических нейронных сетей — сетей нервных клеток живого организма».* Если говорить более понятно и подробно: нейронная сеть – несколько слоев, состоящих из искусственных нейронов и синапсов, которые их соединяют. Нулевой слой называется входным (input), а крайний – выходным (output), между ними находятся «скрытые слои», наглядно это продемонстрировано в приложении**.** Значение нейрона формируется из активированной суммы дочерних нейронов, умноженных на вес их синапсов*(см. гр. 0).* Первый (следующий после нулевого) слой формируется из активированных входных данных, тоже умноженных на веса синапсов. Веса синапсов изначально генерируются случайно, а потом корректируются в зависимости от процесса обучения. «Активированное значение» - значение, которое преобразовано с помощью выбранной функции активации, которая, например, возвращает число от -1 до 1 (например, гиперболический тангенс, который использую я).

**1.2 Устройство нейросети в этом случае**

Задумавшись над задачей для нейронной сети, я решил выбрать что-нибудь подходящее под три критерия: наглядность, чтобы выходом было какое-то графическое действие, обучение с подкреплением, потому что коррекцию ошибки сложно объяснить в биологических терминах, и количество нейронов не более 5—10 млн, ибо имеющаяся у меня в данный момент вычислительная мощность не позволяет быстро обучать нейросети такого объема. После длительного отбора идей, я вспомнил статью *The scientist®* про эксперименты над обучением одноклеточных организмов и пришел к выводу, что правильным решением будет создать примитивную нейросеть, которая будет выполнять роль клетки в чашке Петри. Предварительный анализ задачи показал, что логичней будет ограничить поле зрения: я выбрал поле 5 на 5 вокруг клетки, чтобы входной слой ограничился 25-ю нейронами и нейросеть не запуталась в данных. В итоге я решил сделать нейронную сеть, имеющую входной слой в 25 нейронов, скрытый в 16 и выходной слой в 14. Почему именно столько? В конструировании нейросетей нет четких правил, но количество нейронов в скрытом слою, принято делать между количеством во входном и выходном, а дальше корректировать, в зависимости от эмпирических данных, так что спустя несколько попыток, я выбрал именно 16. Систему обучения я выбрал изначально - подкрепление для нашей задачи подходит идеально. Реализовать обучение с подкреплением для нейросети не сложно: для положительного подкрепления необходимо увеличивать веса синапсов активных нейронов, ответственных за правильное решение на n, а для отрицательного уменьшать. Ещё нужна функция активации, чтобы значение нейрона варьировалось между -1 и 1. Как я уже говорил, я выбираю стандартный гиперболический тангенс, который на самом деле является модифицированной экспонентой.

# Практика

Переходим к написанию кода:

2.1.1. Нам не понадобятся сторонние библиотеки, так что импортируем встроенные *random* и *math:*

import random  
import math

2.1.2. Обычно для нейросетей используют библиотеку NumPy и многомерные массивы, но я в этот раз пойду другим путем. Сначала создаю класс нейрона. У нейрона должны быть 3 переменных: out – выход нейрона, weight – вес синапса, связывающий этот нейрон и родительский, childs – массив дочерних нейронов.

class Neuron(object):  
 def \_\_init\_\_(self, childs, weight,isultra):  
 self.childs = childs  
 self.weight = weight  
 self.isultra=isultra  
 self.out=None

2.1.3. Также нужен класс самой нейронной сети, в ней нужен только массив выходов.

class Mind(object):  
 def \_\_init\_\_(self, childs):  
 self.outs = childs

2.1.4.1. Теперь нужно реализовать создание сети, для этого создаем функцию:

def create\_network(layers,p):  
 mind=Mind([])  
 for i in range(p):  
 mind.outs.append(create\_neuron(layers))  
 return(mind)

*layers* – массив слоев(точнее массив количеств нейронов в слою), не считая выходного

*p* – выходной слой

Что такое *create\_neuron()*? Это функция, которую я создаю далее.

2.1.4.2. Функция *create\_neuron()*:

def create\_neuron(layers):  
 if len(layers)==1:  
 neuron=Neuron([],random.uniform(-1,1),True)  
 for j in range(layers[-1]):  
 neuron.childs.append(Neuron(None,random.uniform(-1,1),False))  
 return neuron  
 else:  
 neuron=Neuron([],random.uniform(-1, 1),False)  
 for j in range(layers[-1]):  
 neuron.childs.append(create\_neuron(layers[:-1]))  
 return neuron

Эта функция – рекурсивная, это означает, что она вызывает сама себя, в этом случае она работает так: Если слой, который необходимо создать, – не предпоследний, то сначала создается нейрон со случайным весом синапса(*random.uniform(-1,1)* – функция, возвращающая псевдослучайное число от -1 до 1), а потом с помощью этой же функции создаются дочерние нейроны этого нейрона, иначе создается нейрон и сразу дочерние нейроны к нему.

2.1.5.1. Создание самой нейронной сети готово, теперь перейдем к получению выхода, для этого создаем самую интересную функцию *out()* классу *Mind*:

class Mind(object):  
 def \_\_init\_\_(self, childs):  
 self.outs = childs  
  
 def out(self,input):  
 maxx=-float('inf')  
 maxxlist=list()  
 for i in range(len(self.outs)):  
 now=self.outs[i].getout(input)  
 if now==maxx:  
 maxxlist.append(i)  
 if now> maxx:  
 maxx=now  
 maxxlist=[i]  
 return(random.choice(maxxlist))

Эта функция принимает параметр *input* – массив входных значений. По сути, эта функция возвращает номер самого активного выходного нейрона или случайного из самых активных. Выход дочернего нейрона возвращается функцией *Neuron.getout(input)*, о которой дальше:

2.1.5.2. *Neuron.getout():*

Перед этой функцией нужно добавить еще одну – функцию активации – гиперболический тангенс:

def act(num):  
 return(math.tanh(num))

Смысл создавать отдельную функцию, а не просто использовать *math.tanh()*, в том, чтобы удобнее было ее заменить, в случае, если я решу, что другая будет эффективней. Далее пишем саму функцию :

class Neuron(object):  
 def \_\_init\_\_(self, childs, weight,isultra):  
 self.childs = childs  
 self.weight = weight  
 self.isultra=isultra  
 self.out=None  
  
 def getout(self,input):  
 if self.isultra:  
 out=0  
 for i in range(len(input)):  
 self.childs[i].out=input[i]  
 out+=act(self.childs[i].out\*self.childs[i].weight)  
 self.out=act(out)  
 return(self.out)  
 else:  
 out=0  
 for i in range(len(self.childs)):  
 out+=act(self.childs[i].getout(input)\*self.childs[i].weight)  
 self.out=act(out)  
 return(self.out)

Эта функция работает следующим образом:

Если на нейрон, из которого вызвали эту функцию находится не на предпоследнем слое – его выход вычисляется по формуле , иначе - . На данном этапе мы уже можем запустить нейросеть со случайным входным значением и увидеть, что все работает и нейросеть выдает случайное значение:

>>> from neurocell import \*  
>>> mind=create\_network([25,16],4)  
>>> print(mind.out([random.uniform(-1,1) for i in range(25)]))  
0  
>>> print(mind.out([random.uniform(-1,1) for i in range(25)]))  
2

Осталось лишь добавить возможность обучения и можно будет переходить к реализации внешней среды.

2.1.6.1. Добавляем классу *Mind* функции *good* и *bad*:

class Mind(object):  
 def \_\_init\_\_(self, childs):  
 self.outs = childs  
  
 def out(self,input):  
 maxx=-float('inf')  
 maxxlist=list()  
 for i in range(len(self.outs)):  
 now=self.outs[i].getout(input)  
 if now==maxx:  
 maxxlist.append(i)  
 if now> maxx:  
 maxx=now  
 maxxlist=[i]  
 return(random.choice(maxxlist))  
  
 def bad(self,out,cof):  
 if out ==-1:  
 for i in range(len(self.outs)):  
 self.outs[i].chweight(-0.001\*cof/len(self.outs))  
 else:  
 self.outs[out].chweight(-0.001\*cof)  
 return   
 def good(self,out,cof):  
 if out ==-1:  
 for i in range(len(self.outs)):  
 self.outs[i].chweight(0.001\*cof/len(self.outs))  
 else:  
 self.outs[out].chweight(0.001\*cof)  
 return

Функции *good* и *bad* меняют веса выбранного нейрона на определенное значение с помощью функции *Neuron.chweight()*. На практике, как следует из названия, good – положительное подкрепление, а bad – отрицательное. Как это работает рассмотрим далее.

2.1.6.2. *Neuron.chweight()*.

class Neuron(object):  
 def \_\_init\_\_(self, childs, weight,isultra):  
 self.childs = childs  
 self.weight = weight  
 self.isultra=isultra  
 self.out=None  
  
 def getout(self,input):  
 if self.isultra:  
 out=0  
 for i in range(len(input)):  
 self.childs[i].out=input[i]  
 out+=act(self.childs[i].out\*self.childs[i].weight)  
 self.out=act(out)  
 return(self.out)  
 else:  
 out=0  
 for i in range(len(self.childs)):  
 out+=act(self.childs[i].getout(input)\*self.childs[i].weight)  
 self.out=act(out)  
 return(self.out)  
  
 def chweight(self,mlt):  
 if self.isultra:  
 for i in range(len(self.childs)):  
 if self.childs[i].out>=0.4:  
 self.childs[i].weight+=mlt  
 if self.childs[i].out<=-0.4:  
 self.childs[i].weight-=mlt   
 if self.out>=0.4:  
 self.weight+=mlt  
 if self.out<=-0.4:  
 self.weight-=mlt   
 else:  
 for i in range(len(self.childs)):  
 self.childs[i].chweight(mlt)  
 if self.out>=0.4:  
 self.weight+=mlt  
 if self.out<=-0.4:  
 self.weight-=mlt   
 return

Что же делает chweight()? По сути это реализация альфа-системы подкрепления: по определению *«Альфа-системой подкрепления называется система подкрепления, при которой веса всех активных связей, которые ведут к элементу, изменяются на одинаковую величину, а веса неактивных связей за это время не изменяются.»* Надо оговориться, что у меня считаются «активными связями» все нейроны, модуль выхода которых, больше, либо равен 0.4, а вес синапса может быть отрицательным.

2.1.7. Полный код:

import random  
import math  
  
defch=float(input("Введите число(дефолт - 0.01):"))  
  
def okr(num):  
 #num = int(num + (0.5 if num > 0 else -0.5))  
 return num  
  
def act(num):  
 return(math.tanh(num))  
  
  
class Neuron(object):  
 def \_\_init\_\_(self, childs, weight,isultra):  
 self.childs = childs  
 self.weight = weight  
 self.isultra=isultra  
 self.out=None  
  
 def getout(self,input):  
 if self.isultra:  
 out=0  
 for i in range(len(input)):  
 self.childs[i].out=input[i]  
 out+=act(self.childs[i].out\*self.childs[i].weight)  
 self.out=act(out)  
 return(self.out)  
 else:  
 out=0  
 for i in range(len(self.childs)):  
 out+=act(self.childs[i].getout(input)\*self.childs[i].weight)  
 self.out=act(out)  
 return(self.out)  
  
 def chweight(self,mlt):  
 if self.isultra:  
 for i in range(len(self.childs)):  
 if self.childs[i].out>=0.4:  
 self.childs[i].weight+=mlt  
 if self.childs[i].out<=-0.4:  
 self.childs[i].weight-=mlt  
 if self.out>=0.4:  
 self.weight+=mlt  
 if self.out<=-0.4:  
 self.weight-=mlt  
 else:  
 for i in range(len(self.childs)):  
 self.childs[i].chweight(mlt)  
 if self.out>=0.4:  
 self.weight+=mlt  
 if self.out<=-0.4:  
 self.weight-=mlt  
 return  
  
class Mind(object):  
 def \_\_init\_\_(self, childs):  
 self.outs = childs  
  
 def out(self,input):  
 maxx=-float('inf')  
 maxxlist=list()  
 for i in range(len(self.outs)):  
 now=self.outs[i].getout(input)  
 if now==maxx:  
 maxxlist.append(i)  
 if now> maxx:  
 maxx=now  
 maxxlist=[i]  
 return(random.choice(maxxlist))  
  
 def bad(self,out,cof):  
 if out ==-1:  
 for i in range(len(self.outs)):  
 self.outs[i].chweight(-defch\*cof/len(self.outs))  
 else:  
 self.outs[out].chweight(-defch\*cof)  
 return  
 def good(self,out,cof):  
 if out ==-1:  
 for i in range(len(self.outs)):  
 self.outs[i].chweight(defch\*cof/len(self.outs))  
 else:  
 self.outs[out].chweight(defch\*cof)  
 return  
  
def create\_neuron(layers):  
 if len(layers)==1:  
 neuron=Neuron([],random.uniform(-1,1),True)  
 for j in range(layers[-1]):  
 neuron.childs.append(Neuron(None,random.uniform(-1,1),False))  
 return neuron  
 else:  
 neuron=Neuron([],random.uniform(-1, 1),False)  
 for j in range(layers[-1]):  
 neuron.childs.append(create\_neuron(layers[:-1]))  
 return neuron  
  
def create\_network(layers,p):  
 mind=Mind([])  
 for i in range(p):  
 mind.outs.append(create\_neuron(layers))  
 return(mind)

2.2.1. На этом сама нейросеть окончательно закончена, пора приступать к разработке экспериментальной среды

2.2.2. Подробное описание процесса разработки среды не имеет ценности для темы проекта, так что я просто опишу принцип работы среды:

Изначально создается массив, который является картой среды. Массив изначально состоит из 0.1 (при работе с нейросетями это эффективнее, чем 0), а потом каждый ход наполняется 1 и -1 случайным образом. Также создается клетка, которая управляется нейросетью, которой на вход подается массив из значений полей в квадрате 5\*5, а на выходе число от 1 до 4, обозначающие ход (1- шаг вверх, 2 - вниз, 3 - вправо, 4 – влево). Проверяется по одной клетке вокруг клетки и если находится 1 – то по этому направлению применяется положительное подкрепление, а если -1 – то отрицательное. Чтобы клетка не стояла на месте, если 0.1, то тоже применяется отрицательное подкрепление, но в меньшем количестве, чем при -1. Также я добавил к этому графический интерфейс.

2.2.3. Таким образом происходит обучение, что наглядно видно на графике*(см. гр. 1-3)*, который строится автоматически. График строится на основе значений положительного и отрицательного подкрепления за ход. Рост графика означает преобладание положительного подкрепления над отрицательным.

2.3.1. Итак, наконец переходим к самой интересной части: процесс обучения. Запускаем среду и видим, что клетка совершает хаотичные действия и большей частью остается на месте и лишь изредка совершает длительные передвижения. График в это время показывает убывание. Спустя некоторое время график начинает приобретать некую стабильность, а клетка иногда совершать правильные ходы, а иногда стоять на месте. Еще спустя немного времени обучения нейросеть понимает, что нужно не стоять на месте и обходить опасность. График начинает резко расти и преодолевает ноль. Это означает, что все работает и нейросеть обучилась. Итак, настало время опробовать различные модификации. В изначальной версии 1 очко подкрепления = 0.001 единиц изменения весов и график преодолевает значение нуля примерно за 150 ходов обучения*(см. гр. 1)*. Увеличим количество подкрепления в 10 раз*(см. гр. 2)*. Смотрим на графики, ого, существенное улучшение, скорость обучения увеличилась в несколько раз. Теперь пробуем увеличить количество подкрепления еще в 10 раз*(см. гр. 3)*. Внезапно скорость обучения стала хуже изначальной. Почему так происходит? Без понятия. Но пока четких объяснений нет – будем оставлять последнее слово за экспериментом и вернемся к значению 0.01.

2.3.2. Попробуем проверить, что будет, если изменить правила на середине процесса? Пробуем скажем на двухтысячном ходу поменять местами положительное и отрицательное подкрепления*(см. гр. 4)*. Как и ожидалось график показывает возрастание до 2 тыс., потом происходит резкое падение, а затем резкое возрастание. На данном этапе я могу сделать проверяемое предсказание, что чем раньше мы меняем правила, тем проще нейросети адаптироваться. Итак, запускаем среду с изменением правил на тысячном ходу*(см. гр. 5)* и наблюдаем: график после тысячи не резко падает, а проседает и почти сразу переходит в стабильность и спустя некоторое время происходит резкое возрастание.

2.4. Я сторонник open source, так что свои проекты обычно выкладываю на свой аккаунт в github. Сделав это, я решил также написать статью по теме проекта на habr, чтобы получить профит от проекта и получить конструктивную критику.

# Заключение

Создание индивидуального проекта – интересный и полезный опыт, давший мне замечательную возможность проявить мои давно сформированные навыки сбора, обработки и анализации информации, креативность и критическое мышление. Во время разработки нейронной сети я применял накопленные знания, изучал новое и практиковал написание текстов в научном/официально-деловом стиле, что наверняка поможет мне в будущем.

Формулируя свои выводы, я скажу, что нейронные сети – очень интересная для меня тема и я обязательно продолжу развиваться в ней в будущем и скорее всего в ближайшее время постараюсь сделать что-нибудь крупное с их участием.

# Список используемой литературы

1. wikipedia.org - the free encyclopedia (определения)

И на этом месте я понял, что все мной написанное я сформировал из своих знаний, полученных мною, когда я всерьез заинтересовался своей темой (этим летом), так что укажу также и литературу по которой я учился еще тогда:

2. habr.com (на нем я прочитал очень много статей про нейросети, что в общем сформировало мои знания.).

3. the-scientist.com (тут преимущественно я читал статьи про нейробиологию и обучение, что также помогло мне при создании проекта).

# Приложение

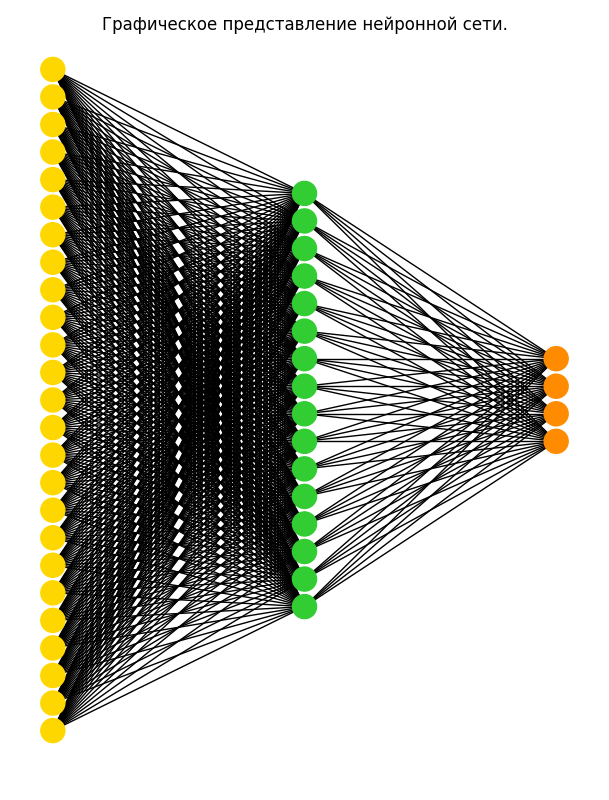


график 0

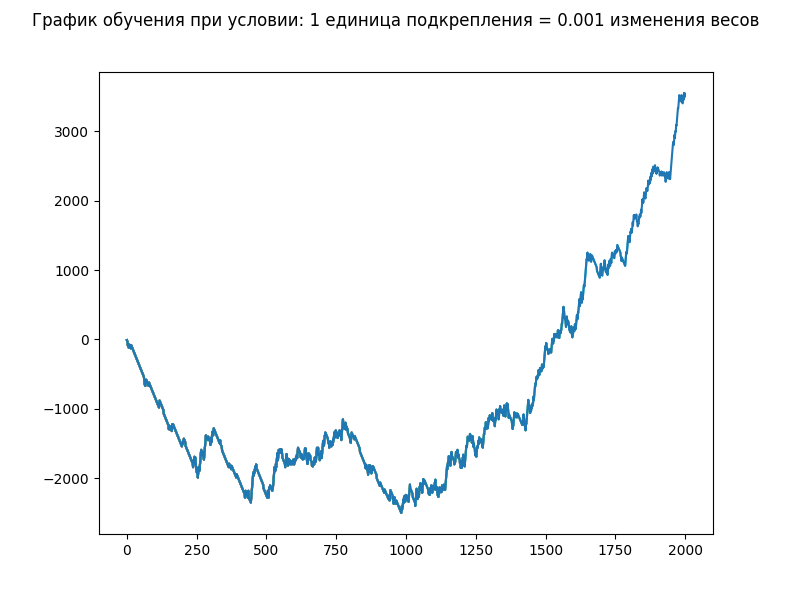


график 1

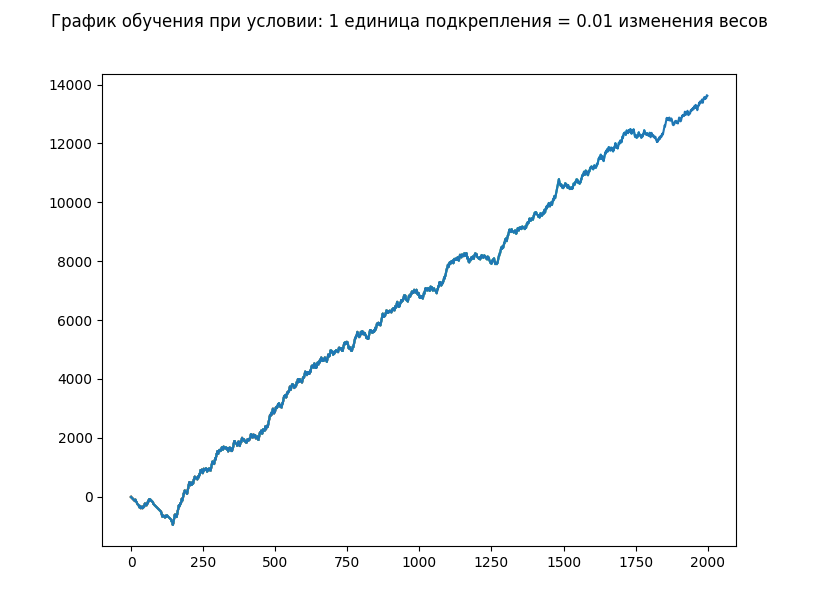


график 2

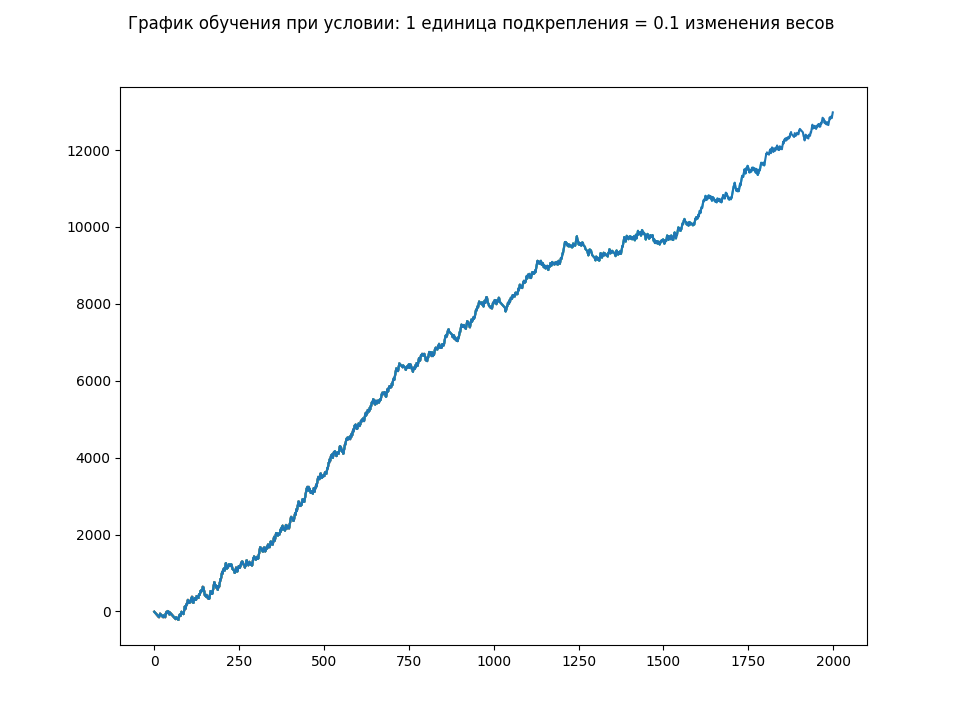


график 3

[Ссылка](https://youtu.be/QoiFizHaxLc) на запись процесса обучения.

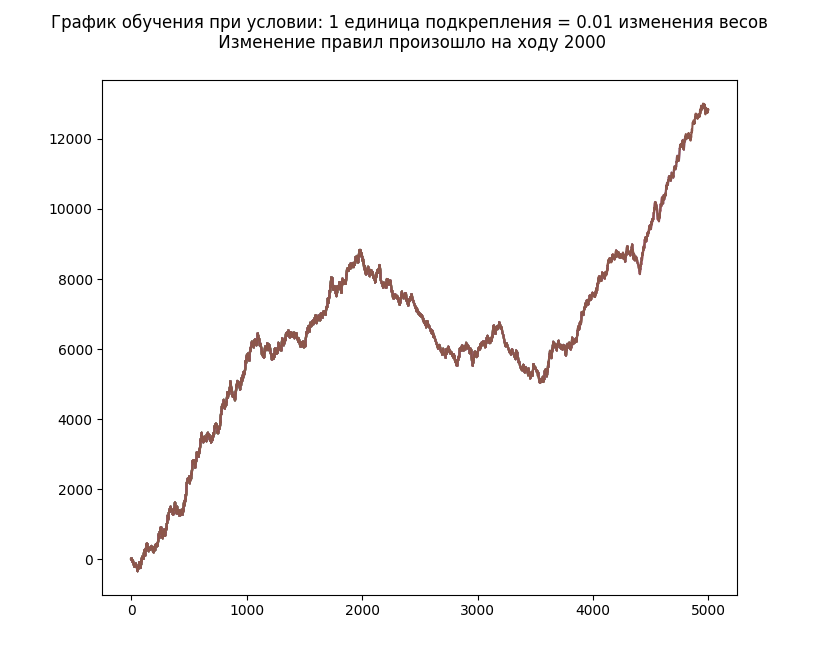


график 4

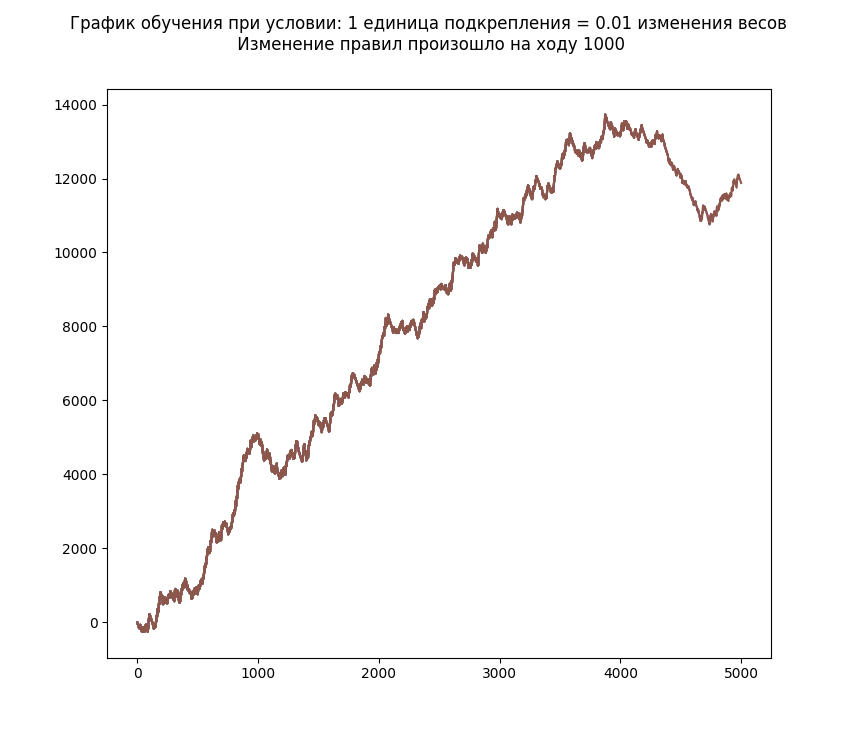


график 5

[Ссылка](https://habr.com/ru/post/577348/) на статью на habr.com.

[Ссылка](https://github.com/leo-need-more-coffee/simple-neural-net) на исходный код нейросети и среды обучения.

[Ссылка](https://github.com/leo-need-more-coffee/simple-neural-net/releases/tag/relisssss) на релиз(есть .exe файл).

Запуск продукта

(при условии .exe файла в папке с проектом)