**НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ УКРАИНЫ**

**«КИЕВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**ИМЕНИ ИГОРЯ СИКОРСКОГО»**

**ФАКУЛЬТЕТ ИНФОРМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ**

**КАФЕДРЫ ТЕХНИЧЕСКОЙ КИБЕРНЕТИКИ**

**НЕЙРОТЕХНОЛОГИИ И НЕЙРОКОМПЬЮТЕРНЫЕ СИСТЕМЫ**

Индивидуальное задание

|  |  |
| --- | --- |
| **проверил:**  **Старший преподаватель кафедры,**  **к.т.н**  **Олейник В.В** | Выполнил:  слушатель Макиян С. А.  зач. книжка № ЗПІ-ЗП6115  гр. ЗПІ-зп61 |

**КИЕВ – 2019**

**Задача.**

**Постановка задачи.** В рамках данной работы было решено создать самообучающую искусственную нейронную сеть, которая сама бы могла обучаться и проходить различные компьютерные игры. Для примера была выбрана игра “Catch”.

**Мотивация.** Не смотря на то что ИИ для данной игры написать можно использовав машинное зрение и несколько условий ветвления, для прохождения игр с более сложными условиями написать ИИ уже очень сложно. Создание данной нейронной сети, позволит в будущем с небольшими изменениями научить ее проходить другие игры. Отсутствие явного старого набора данных позволяет абстрагировать от конкретной игры, меняя только правила игры и условия оценивания. Конечной целью, является создание алгоритма аналога Alpha Zero, является на данный момент самой совершенной программой для игры в Go и Шахматы, а так же способна играть в другие подобные игры.

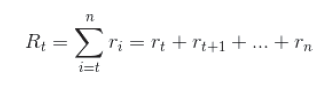
**Формализация задачи.**

* найти готовое решение игры “Catch”, подготовить ее для тренировки нейросети;
* создать нейросеть с оптическим разспознаванием картинки на экране;
* обучить нейросеть проходить игру “Catch”.

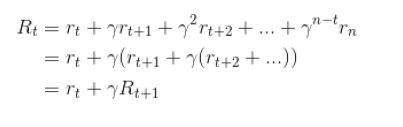
**Модель нейросети.**

**Тип нейросети.** Игра “Catch”, является простой игрой, где сверху падаюют мячи, а снизу пользователь должен их поймать на платформу.Основной сложностью данной нейросети, является отсутствие подкрепления обучения за после каждого шага. Все шаги нейросети оцениваются только в конце каждой эпохи, где определяется была ли данная последовательность удачно или нет. Данный тип нейросети хорошо описывается алгоритмом MDP (markov decision process).

Подкрепление выигрыша: целью данной нейросети, является поймать на платформу мячики, при этом уменьшив процент игр завершающихся неудачей. Сумму всех выигрышей можно переставить формулой:



или в развернутом виде



**Q-обучение (Q-learning)** — метод, применяемый в искусственном интеллекте при агентном подходе. Применяется для ситуаций, которые можно представить в виде марковского процесса принятия решений. Относится к экспериментам вида oбучение с подкреплением. На основе получаемого от среды вознаграждения агент формирует функцию полезности Q, что впоследствии дает ему возможность уже не случайно выбирать стратегию поведения, а учитывать опыт предыдущего взаимодействия со средой. Одно из преимуществ Q-обучения — то, что оно в состоянии сравнить ожидаемую полезность доступных действий, не формируя модели окружающей среды.

Короткий вид Q ­– функции представлен на картинке ниже.

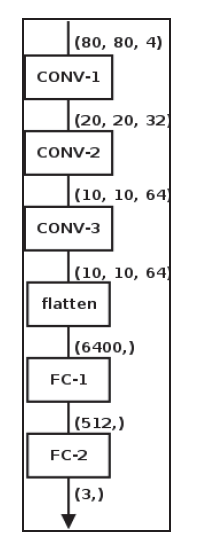


После нахождения значения Q функции, можно найти поведение.

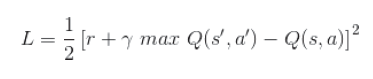


**Представление нейросети как Q-функции.** Теперь необходимо определится с типом нейросети. На вход мы будем получать картинку 80х80 (где каждый «пиксель» представлен в виде квадрата из 4-х пикселей), что дает нам огромное число вариантов. Для этого целесообразно выбрать сверточную модель нейросети, так как каждый нейрон будет закреплен за отдельным регионом экрана.

Структуру будущей нейросети можно представить схематически:



В данной игре Q- функция на выходе должна выдать вероятность оптимального решения для 3-х вариантов действия (стоять на месте, подвинуть платформу вправо, подвинуть платформу влево), то для нормализации решения и средней квадратично ошибки, необходимо применить оптимизатор вида



Фронтендом для создания нейросети я выбрал Keras. К преимуществам данного фронтэнда можно отнести:

* язык python (нет необходимости учить специальный новый язык программирования для нейросети + сама игра Catch написана на пайтон);
* легкая заменяемость бэкндов;
* хорошая документация и большое количество примеров.

**Ход работы.**

Для начала скачаем реализацию игры “Catch” на python [листинг game.py] Убедимся что игра работает с клавиатуры. После этого игру необходимо подготовить к тестированию - написать обертку [листинг wrapped\_game.py], в которой: игра проходится с куда меньшей задержкой; управление с клавиатуры заменено на управление нейросетью; будет возвращен результат и состояние игры (игра закончилась успешно/неуспешно, игра не закончилась).

После этого необходимо непосредственно создать нейросеть [листинг network-traine.py].

Ключевые моменты для создания нейросети:

model = Sequential()

model.add(Conv2D(32, kernel\_size=8, strides=4,

kernel\_initializer="normal",

padding="same",

input\_shape=(80, 80, 4)))

model.add(Activation("relu"))

model.add(Conv2D(64, kernel\_size=4, strides=2,

kernel\_initializer="normal",

padding="same"))

model.add(Activation("relu"))

model.add(Conv2D(64, kernel\_size=3, strides=1,

kernel\_initializer="normal",

padding="same"))

model.add(Activation("relu"))

model.add(Flatten())

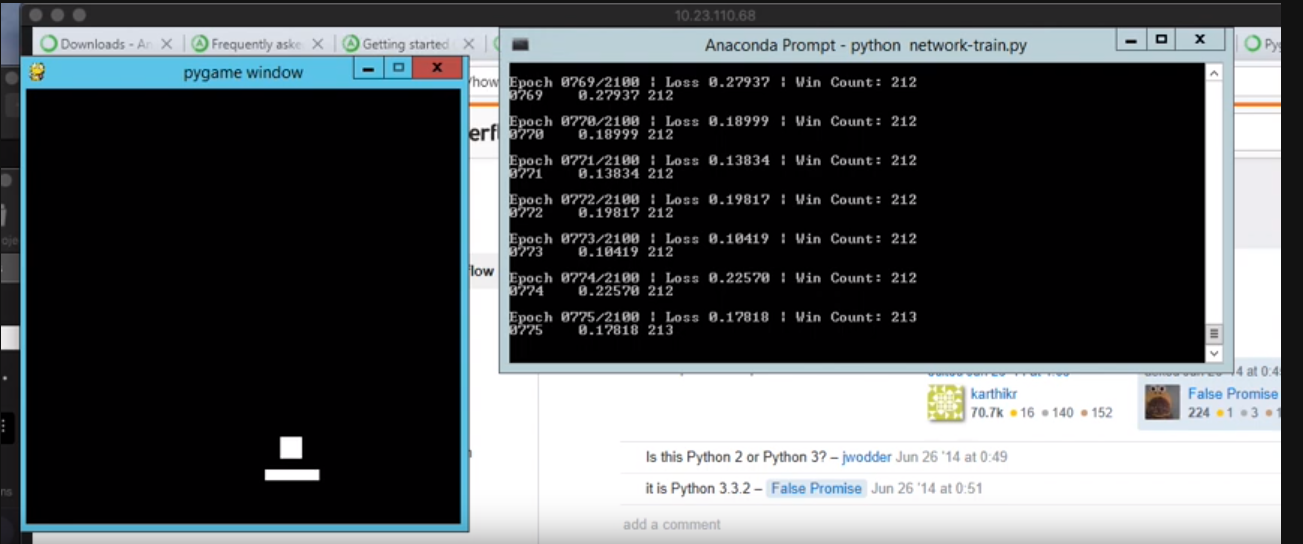
model.add(Dense(512, kernel\_initializer="normal"))

model.add(Activation("relu"))

model.add(Dense(3, kernel\_initializer="normal"))

model.compile(optimizer=Adam(lr=1e-6), loss="mse")

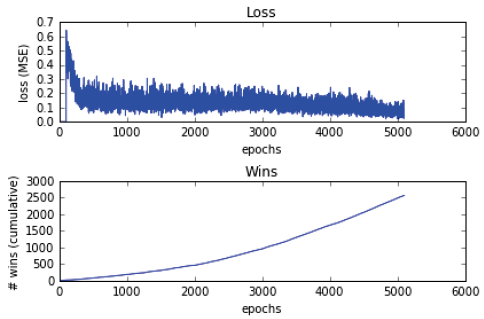
После того как написана программа для обучение нейросети, необходимо запустить непосредственно процесс обучения.



Я выбрал для начала 2100 эпох (партий) для обучения за один раз, из которых первые 100 действия выбираются в случайном порядке, а начиная с 101 партии, за результат действия, начинает отвечать нейросеть. Все успешные последовательности шагов, будут записаны в память нейросети и использованы для дальнейшего обучения.

Результаты обучения нейросети: через несколько тысяч эпох нейросеть начинает проходить игру, с очень низким количеством проигрышей, хотя результаты работы Q-функции необходимо еще оптимизировать, так как платформа иногда просто так двигается влево и вправо, ожидании мячика.

Отчет по процессу обучения: график выигрышей и процента проигрышей.



Для тысячи сессий игры нейросеть имеет 42% выигрышей; для 2000 - 74%, для 5000 – 87%, что наглядно показывает успешность обучения нашей нейронной сети.

**Приложение:**

**Листинг игры game.py**

#!/usr/bin/env python2

# -\*- coding: utf-8 -\*-

from \_\_future\_\_ import division, print\_function

import collections

import numpy as np

import pygame

import random

class MyGame(object):

def \_\_init\_\_(self):

pygame.init()

pygame.key.set\_repeat(10, 100)

# set constants

self.COLOR\_WHITE = (255, 255, 255)

self.COLOR\_BLACK = (0, 0, 0)

self.GAME\_WIDTH = 400

self.GAME\_HEIGHT = 400

self.BALL\_WIDTH = 20

self.BALL\_HEIGHT = 20

self.PADDLE\_WIDTH = 50

self.PADDLE\_HEIGHT = 10

self.GAME\_FLOOR = 350

self.GAME\_CEILING = 10

self.BALL\_VELOCITY = 10

self.PADDLE\_VELOCITY = 20

self.FONT\_SIZE = 30

self.MAX\_TRIES\_PER\_GAME = 100

self.CUSTOM\_EVENT = pygame.USEREVENT + 1

self.font = pygame.font.SysFont("Comic Sans MS", self.FONT\_SIZE)

# we want to start the game remotely, so moving this block

# to its own function

self.frames = collections.deque(maxlen=4)

self.game\_over = False

# initialize positions

self.paddle\_x = self.GAME\_WIDTH // 2

self.game\_score = 0

self.ball\_x = random.randint(0, self.GAME\_WIDTH)

self.ball\_y = self.GAME\_CEILING

self.num\_tries = 0

# set up display, clock, etc

self.screen = pygame.display.set\_mode(

(self.GAME\_WIDTH, self.GAME\_HEIGHT))

self.clock = pygame.time.Clock()

def play(self):

# game loop

while not self.game\_over:

if self.num\_tries > self.MAX\_TRIES\_PER\_GAME:

self.game\_over = True

break

for event in pygame.event.get():

if event.type == pygame.QUIT:

break

if event.type == pygame.KEYDOWN and event.key == pygame.K\_ESCAPE:

self.game\_over = True

break

if event.type == pygame.KEYDOWN and event.key == pygame.K\_LEFT:

self.paddle\_x -= self.PADDLE\_VELOCITY

if self.paddle\_x < 0:

self.paddle\_x = 0

if event.type == pygame.KEYDOWN and event.key == pygame.K\_RIGHT:

self.paddle\_x += self.PADDLE\_VELOCITY

if self.paddle\_x > self.GAME\_WIDTH - self.PADDLE\_WIDTH:

self.paddle\_x = self.GAME\_WIDTH - self.PADDLE\_WIDTH

self.screen.fill(self.COLOR\_BLACK)

score\_text = self.font.render("Score: {:d}/{:d}, Ball: {:d}"

.format(self.game\_score, self.MAX\_TRIES\_PER\_GAME,

self.num\_tries), True, self.COLOR\_WHITE)

# self.screen.blit(score\_text,

# ((self.GAME\_WIDTH - score\_text.get\_width()) // 2,

# (self.GAME\_FLOOR + self.FONT\_SIZE // 2)))

# update ball position

self.ball\_y += self.BALL\_VELOCITY

ball = pygame.draw.rect(self.screen, self.COLOR\_WHITE,

pygame.Rect(self.ball\_x, self.ball\_y,

self.BALL\_WIDTH,

self.BALL\_HEIGHT))

# update paddle position

paddle = pygame.draw.rect(self.screen, self.COLOR\_WHITE,

pygame.Rect(self.paddle\_x,

self.GAME\_FLOOR,

self.PADDLE\_WIDTH,

self.PADDLE\_HEIGHT))

# check for collision

if self.ball\_y >= self.GAME\_FLOOR - self.BALL\_WIDTH // 2:

if ball.colliderect(paddle):

self.game\_score += 1

else:

self.game\_score -= 1

self.ball\_x = random.randint(0, self.GAME\_WIDTH)

self.ball\_y = self.GAME\_CEILING

self.num\_tries += 1

pygame.display.flip()

# save last 4 frames

self.frames.append(pygame.surfarray.array2d(self.screen))

self.clock.tick(30)

# save last 4 frames

S = np.array(self.frames)

with open("game-screenshots.npy", "wb") as fscreenshot:

np.save(fscreenshot, S)

def get\_state(self):

return np.array(list(self.frames))

def set\_action(self, action):

pass

if \_\_name\_\_ == "\_\_main\_\_":

game = MyGame()

game.play()

**Листинг обертки к игре game\_wrapper.py**

# -\*- coding: utf-8 -\*-

from \_\_future\_\_ import division, print\_function

import collections

import numpy as np

import pygame

import random

import os

class MyWrappedGame(object):

def \_\_init\_\_(self):

# run pygame in headless mode

# os.environ["SDL\_VIDEODRIVER"] = "dummy"

pygame.init()

pygame.key.set\_repeat(10, 100)

# set constants

self.COLOR\_WHITE = (255, 255, 255)

self.COLOR\_BLACK = (0, 0, 0)

self.GAME\_WIDTH = 400

self.GAME\_HEIGHT = 400

self.BALL\_WIDTH = 20

self.BALL\_HEIGHT = 20

self.PADDLE\_WIDTH = 50

self.PADDLE\_HEIGHT = 10

self.GAME\_FLOOR = 350

self.GAME\_CEILING = 10

# based on experimentation, the ball tends to move 4 times

# between each paddle movement. Since here we alternate ball

# and paddle movement, we make ball move 4x faster.

self.BALL\_VELOCITY = 10

self.PADDLE\_VELOCITY = 20

self.FONT\_SIZE = 30

# self.MAX\_TRIES\_PER\_GAME = 100

self.MAX\_TRIES\_PER\_GAME = 1

self.CUSTOM\_EVENT = pygame.USEREVENT + 1

self.font = pygame.font.SysFont("Comic Sans MS", self.FONT\_SIZE)

def reset(self):

self.frames = collections.deque(maxlen=4)

self.game\_over = False

# initialize positions

self.paddle\_x = self.GAME\_WIDTH // 2

self.game\_score = 0

self.reward = 0

self.ball\_x = random.randint(0, self.GAME\_WIDTH)

self.ball\_y = self.GAME\_CEILING

self.num\_tries = 0

# set up display, clock, etc

self.screen = pygame.display.set\_mode(

(self.GAME\_WIDTH, self.GAME\_HEIGHT))

self.clock = pygame.time.Clock()

def step(self, action):

pygame.event.pump()

if action == 0: # move paddle left

self.paddle\_x -= self.PADDLE\_VELOCITY

if self.paddle\_x < 0:

# bounce off the wall, go right

self.paddle\_x = self.PADDLE\_VELOCITY

elif action == 2: # move paddle right

self.paddle\_x += self.PADDLE\_VELOCITY

if self.paddle\_x > self.GAME\_WIDTH - self.PADDLE\_WIDTH:

# bounce off the wall, go left

self.paddle\_x = self.GAME\_WIDTH - self.PADDLE\_WIDTH - self.PADDLE\_VELOCITY

else: # dont move paddle

pass

self.screen.fill(self.COLOR\_BLACK)

score\_text = self.font.render("Score: {:d}/{:d}, Ball: {:d}"

.format(self.game\_score, self.MAX\_TRIES\_PER\_GAME,

self.num\_tries), True, self.COLOR\_WHITE)

# self.screen.blit(score\_text,

# ((self.GAME\_WIDTH - score\_text.get\_width()) // 2,

# (self.GAME\_FLOOR + self.FONT\_SIZE // 2)))

# update ball position

self.ball\_y += self.BALL\_VELOCITY

ball = pygame.draw.rect(self.screen, self.COLOR\_WHITE,

pygame.Rect(self.ball\_x, self.ball\_y,

self.BALL\_WIDTH,

self.BALL\_HEIGHT))

# update paddle position

paddle = pygame.draw.rect(self.screen, self.COLOR\_WHITE,

pygame.Rect(self.paddle\_x,

self.GAME\_FLOOR,

self.PADDLE\_WIDTH,

self.PADDLE\_HEIGHT))

# check for collision and update reward

self.reward = 0

if self.ball\_y >= self.GAME\_FLOOR - self.BALL\_WIDTH // 2:

if ball.colliderect(paddle):

self.reward = 1

else:

self.reward = -1

self.game\_score += self.reward

self.ball\_x = random.randint(0, self.GAME\_WIDTH)

self.ball\_y = self.GAME\_CEILING

self.num\_tries += 1

pygame.display.flip()

# save last 4 frames

self.frames.append(pygame.surfarray.array2d(self.screen))

if self.num\_tries >= self.MAX\_TRIES\_PER\_GAME:

self.game\_over = True

self.clock.tick(30)

return self.get\_frames(), self.reward, self.game\_over

def get\_frames(self):

return np.array(list(self.frames))

if \_\_name\_\_ == "\_\_main\_\_":

game = MyWrappedGame()

NUM\_EPOCHS = 10

for e in range(NUM\_EPOCHS):

print("Epoch: {:d}".format(e))

game.reset()

input\_t = game.get\_frames()

game\_over = False

while not game\_over:

action = np.random.randint(0, 3, size=1)[0]

input\_tp1, reward, game\_over = game.step(action)

print(action, reward, game\_over)

**Листинг функции создании скриншотов game\_screechot.py**

#!/usr/bin/env python2

# -\*- coding: utf-8 -\*-

from \_\_future\_\_ import division, print\_function

import matplotlib.pyplot as plt

import numpy as np

X = np.load("game-screenshots.npy")

print(X.shape)

sidx = 140

for soff in range(4):

plt.subplot(sidx + soff + 1)

plt.imshow(X[soff].T, cmap="gray")

plt.xticks([])

plt.yticks([])

plt.tight\_layout()

plt.show()

**Листинг скрипта для тренировки нейросети network-train.py**

# -\*- coding: utf-8 -\*-

from \_\_future\_\_ import division, print\_function

from keras.models import Sequential

from keras.layers.core import Activation, Dense, Flatten

from keras.layers.convolutional import Conv2D

from keras.optimizers import Adam

from scipy.misc import imresize

import collections

import numpy as np

import os

import wrapped\_game

def preprocess\_images(images):

if images.shape[0] < 4:

# single image

x\_t = images[0]

x\_t = imresize(x\_t, (80, 80))

x\_t = x\_t.astype("float")

x\_t /= 255.0

s\_t = np.stack((x\_t, x\_t, x\_t, x\_t), axis=2)

else:

# 4 images

xt\_list = []

for i in range(images.shape[0]):

x\_t = imresize(images[i], (80, 80))

x\_t = x\_t.astype("float")

x\_t /= 255.0

xt\_list.append(x\_t)

s\_t = np.stack((xt\_list[0], xt\_list[1], xt\_list[2], xt\_list[3]),

axis=2)

s\_t = np.expand\_dims(s\_t, axis=0)

return s\_t

def get\_next\_batch(experience, model, num\_actions, gamma, batch\_size):

batch\_indices = np.random.randint(low=0, high=len(experience),

size=batch\_size)

batch = [experience[i] for i in batch\_indices]

X = np.zeros((batch\_size, 80, 80, 4))

Y = np.zeros((batch\_size, num\_actions))

for i in range(len(batch)):

s\_t, a\_t, r\_t, s\_tp1, game\_over = batch[i]

X[i] = s\_t

Y[i] = model.predict(s\_t)[0]

Q\_sa = np.max(model.predict(s\_tp1)[0])

if game\_over:

Y[i, a\_t] = r\_t

else:

Y[i, a\_t] = r\_t + gamma \* Q\_sa

return X, Y

############################# main ###############################

# initialize parameters

DATA\_DIR = "."

NUM\_ACTIONS = 3 # number of valid actions (left, stay, right)

GAMMA = 0.99 # decay rate of past observations

INITIAL\_EPSILON = 0.1 # starting value of epsilon

FINAL\_EPSILON = 0.0001 # final value of epsilon

MEMORY\_SIZE = 50000 # number of previous transitions to remember

NUM\_EPOCHS\_OBSERVE = 100

NUM\_EPOCHS\_TRAIN = 2000

BATCH\_SIZE = 32

NUM\_EPOCHS = NUM\_EPOCHS\_OBSERVE + NUM\_EPOCHS\_TRAIN

# build the model

model = Sequential()

model.add(Conv2D(32, kernel\_size=8, strides=4,

kernel\_initializer="normal",

padding="same",

input\_shape=(80, 80, 4)))

model.add(Activation("relu"))

model.add(Conv2D(64, kernel\_size=4, strides=2,

kernel\_initializer="normal",

padding="same"))

model.add(Activation("relu"))

model.add(Conv2D(64, kernel\_size=3, strides=1,

kernel\_initializer="normal",

padding="same"))

model.add(Activation("relu"))

model.add(Flatten())

model.add(Dense(512, kernel\_initializer="normal"))

model.add(Activation("relu"))

model.add(Dense(3, kernel\_initializer="normal"))

model.compile(optimizer=Adam(lr=1e-6), loss="mse")

# train network

game = wrapped\_game.MyWrappedGame()

experience = collections.deque(maxlen=MEMORY\_SIZE)

fout = open(os.path.join(DATA\_DIR, "rl-network-results.tsv"), "wb")

num\_games, num\_wins = 0, 0

epsilon = INITIAL\_EPSILON

for e in range(NUM\_EPOCHS):

loss = 0.0

game.reset()

# get first state

a\_0 = 1 # (0 = left, 1 = stay, 2 = right)

x\_t, r\_0, game\_over = game.step(a\_0)

s\_t = preprocess\_images(x\_t)

while not game\_over:

s\_tm1 = s\_t

# next action

if e <= NUM\_EPOCHS\_OBSERVE:

a\_t = np.random.randint(low=0, high=NUM\_ACTIONS, size=1)[0]

else:

if np.random.rand() <= epsilon:

a\_t = np.random.randint(low=0, high=NUM\_ACTIONS, size=1)[0]

else:

q = model.predict(s\_t)[0]

a\_t = np.argmax(q)

# apply action, get reward

x\_t, r\_t, game\_over = game.step(a\_t)

s\_t = preprocess\_images(x\_t)

# if reward, increment num\_wins

if r\_t == 1:

num\_wins += 1

# store experience

experience.append((s\_tm1, a\_t, r\_t, s\_t, game\_over))

if e > NUM\_EPOCHS\_OBSERVE:

# finished observing, now start training

# get next batch

X, Y = get\_next\_batch(experience, model, NUM\_ACTIONS,

GAMMA, BATCH\_SIZE)

loss += model.train\_on\_batch(X, Y)

# reduce epsilon gradually

if epsilon > FINAL\_EPSILON:

epsilon -= (INITIAL\_EPSILON - FINAL\_EPSILON) / NUM\_EPOCHS

print("Epoch {:04d}/{:d} | Loss {:.5f} | Win Count: {:d}"

.format(e + 1, NUM\_EPOCHS, loss, num\_wins))

fout.write("{:04d}\t{:.5f}\t{:d}\n"

.format(e + 1, loss, num\_wins))

if e % 100 == 0:

model.save(os.path.join(DATA\_DIR, "rl-network.h5"), overwrite=True)

fout.close()

model.save(os.path.join(DATA\_DIR, "rl-network.h5"), overwrite=True)

**Листинг визуализации отчетов network-wiz.py**

# -\*- coding: utf-8 -\*-

from \_\_future\_\_ import division, print\_function

import numpy as np

import matplotlib.pyplot as plt

import os

DATA\_DIR = "."

xs, loss\_ys, wins\_ys = [], [], []

fin = open(os.path.join(DATA\_DIR, "rl-network-results-5100.tsv"), "rb")

lno = 0

for line in fin:

# if lno < 1000:

# lno += 1

# continue

cols = line.strip().split("\t")

epoch = int(cols[0])

loss = float(cols[1])

num\_wins = int(cols[2])

xs.append(epoch)

loss\_ys.append(loss)

wins\_ys.append(num\_wins)

lno += 1

fin.close()

plt.subplot(211)

plt.plot(xs, loss\_ys)

plt.title("Loss")

plt.xlabel("epochs")

plt.ylabel("loss (MSE)")

plt.subplot(212)

plt.plot(xs, wins\_ys)

plt.title("Wins")

plt.xlabel("epochs")

plt.ylabel("# wins (cumulative)")

plt.tight\_layout()

plt.show()

**Листинг скрипта для тестирования network-test.py**

# -\*- coding: utf-8 -\*-

from \_\_future\_\_ import division, print\_function

from keras.models import load\_model

from keras.optimizers import Adam

from scipy.misc import imresize

import numpy as np

import os

import wrapped\_game

def preprocess\_images(images):

if images.shape[0] < 4:

# single image

x\_t = images[0]

x\_t = imresize(x\_t, (80, 80))

x\_t = x\_t.astype("float")

x\_t /= 255.0

s\_t = np.stack((x\_t, x\_t, x\_t, x\_t), axis=2)

else:

# 4 images

xt\_list = []

for i in range(images.shape[0]):

x\_t = imresize(images[i], (80, 80))

x\_t = x\_t.astype("float")

x\_t /= 255.0

xt\_list.append(x\_t)

s\_t = np.stack((xt\_list[0], xt\_list[1], xt\_list[2], xt\_list[3]),

axis=2)

s\_t = np.expand\_dims(s\_t, axis=0)

return s\_t

############################# main ###############################

DATA\_DIR = "."

BATCH\_SIZE = 32

NUM\_EPOCHS = 100

model = load\_model(os.path.join(DATA\_DIR, "rl-network-4100.h5"))

model.compile(optimizer=Adam(lr=1e-6), loss="mse")

# train network

game = wrapped\_game.MyWrappedGame()

num\_games, num\_wins = 0, 0

for e in range(NUM\_EPOCHS):

loss = 0.0

game.reset()

# get first state

a\_0 = 1 # (0 = left, 1 = stay, 2 = right)

x\_t, r\_0, game\_over = game.step(a\_0)

s\_t = preprocess\_images(x\_t)

while not game\_over:

s\_tm1 = s\_t

# next action

q = model.predict(s\_t)[0]

a\_t = np.argmax(q)

# apply action, get reward

x\_t, r\_t, game\_over = game.step(a\_t)

s\_t = preprocess\_images(x\_t)

# if reward, increment num\_wins

if r\_t == 1:

num\_wins += 1

num\_games += 1

print("Game: {:03d}, Wins: {:03d}".format(num\_games, num\_wins), end="\r")

print("")