`

Chapter – 3 (implementation – Part 1 - Environment Setup)

**Artificial Intelligence**

**AI-app (CNN-DQ): DOOM Game-play** (part 1)

Environment Setup & Install

Folder Structure & Libraries

**3.1** step 1: **Environment Setup & Install**

Please find below the commands you need to run in a terminal to install everything required for Doom (make sure you run them in the same order):

**Installing Open AI Gym and ppaquette**

* Install OpenAI (link <https://gym.openai.com/docs>):

git clone https://github.com/openai/gym

cd gym

pip install -e . # minimal install

* Install the full set of environments(link <https://github.com/openai/gym#installing-everything>):
* Mac OSX:

brew install cmake boost boost-python sdl2 swig wget

pip install -e '.[all]'

* Linux (or Windows with Ubuntu):

apt-get install -y python-numpy python-dev cmake zlib1g-dev libjpeg-dev xvfb libav-tools xorg-dev python-opengl libboost-all-dev libsdl2-dev swig

pip install -e '.[all]'

* Install ppaquette to visualize Doom (link <https://github.com/ppaquette/gym-doom/blob/master/README.md>):

pip install ppaquette-gym-doom

* Install ffmpeg to get the videos in a folder (link <https://anaconda.org/conda-forge/ffmpeg>):

conda install -c conda-forge ffmpeg=3.2.4

**Method 1: Installing Open AI Gym**

* Install Open-Ai GYM in MAC:

Use commands and what they for

* Install dependencies first and then
* Clone the Open-Ai Gym and install other requirements
* Install doom.py and vizdoom with ppaquette, GymDoom.
* Use Home-brew to install the environment. Brew install command:

brew install cmake boost boost-python sdl2 swig wget

* Clone the Open-Ai Gym:

git clone htpps://github.com/openai/gym

* Change directory to 'gym', **'-e'** means install everything
* Install Open-Ai GYM in Linux: cd gym
* Install everything in Open-Ai GYM: pip install -e '.[all]'
* Clone the Doom.py:

cd

git clone https://github.com/openai/doom-py

* Install doom.py: cd doom-py

pip install –e .

don't forget the '.'

* Install vizdoom:

cd

pip install vizdoom

* Install ppaquette:

pip install ppaquette-gym-doom

* Note: these environment runs under **python 2.7**

**Common Debugging Solutions**

1. Download VirtualBox Ubuntu 17.10 Artful 64bits disk image from osboxes:

http://www.osboxes.org/ubuntu/#ubuntu-1710-vbox

Create the VM with 4GB of memory (2GB should be more than enough)

Run Guest Additions CD (optional)

1. Download Anaconda (latest version) from:

https://www.anaconda.com/download/#linux

Install Anacoda:

bash Anaconda3-5.0.1-Linux-x86\_64.sh

Close terminal and open a new one

1. Downgrade to Python 3.5

conda install python=3.5

1. Install PyTorch

conda install pytorch torchvision -c soumith

1. Install OpenAI Gym:

sudo apt install git

git clone https://github.com/openai/gym

cd gym

pip install -e .

sudo apt-get install -y python3-numpy python3-dev python-numpy pythondev cmake zlib1g-dev libjpeg-dev xvfb libav-tools xorg-dev pythonopengl libboost-all-dev libsdl2-dev swig pip install -e '.[all]'

1. Install FFmpeg:

conda install -c conda-forge ffmpeg

1. Install OpenCV:

conda install -c menpo opencv

1. Now open the Breakout source code files using Spyder, Change **model.py**, line 12 to:

out \*= std / torch.sqrt(out.pow(2).sum(1, keepdim=True).expand\_as(out))

# thanks to this initialization, we have var(out) = std^2

1. Change **test.py**, line 37 to:

state, reward, done, \_ = env.step(action[0]) # done = done or

episode\_length >= params.max\_episode\_length

* The training of the agent took around half an hour to achieve the first rewards (so don't panic if you don't see any results during the first minutes), and a few hours to get some good results.
* By the way, for those that want to save more videos than the "standard" behaviour, just change envs.py, line 17 to:

env = wrappers.Monitor(env, 'test', force=True, video\_callable=lambda episode\_id: True) # save all episodes

or

env = wrappers.Monitor(env, 'test', force=True, video\_callable=lambda

episode\_id: episode\_id%10==0) # save every 10th, etc

* Probably for the Doom environment all we need is to install **ppaquette** before **FFmpeg** (didn't test it in this script since I have the Doom environment working on another VM):

pip install ppaquette-gym-doom

* Environment installation:
* Go to OpenAi Gym's website and then go to 'Environments' tab.
* <https://www.gymlibrary.dev/environments/third_party_environments/index.html>
* <https://github.com/Farama-Foundation/ViZDoom>

There (first link) you have all the environments for games. For each of these games you can implement an AI to beat the game.

* For each of those games you have to specify which goal you have to accomplish what are the actions and what are the rewards.
* For most of the games the input states are actually the input images (screen-buffer).
* i.e. it's not like before **with** the **self-driving car** where we had ***a vector encoding one state*** of the environment, for most of the **OpenAI-gym games** the *input state* is instead the *image*.
* It's more like an AI with eyes like a humans.
* This AI we're about to implement is to be the closest to a human brain because it is going to observe a series of images then it is going to detect where it has to do in the images and then using a classic neural network it'll know which action it should play.

**Game Description**

* Doom Environment: We have several versions of doom. These are just different maps with different levels.
* **Level 1:** **DoomBasiC-v0 (experimental) (by @ppaquette)**
* This map is rectangular with gray walls, ceiling and floor. You are spawned in the center of the longer wall, and a red circular monster is spawned randomly on the opposite wall. You need to kill the monster (one bullet is enough).

|  |  |
| --- | --- |
| * Goal: 10 points * Kill the monster in 3 secs with 1 shot Rewards: * Plus 101 pts for killing the monster * Minus 5 pts for missing a shot * Minus 1 pts every 0.028 secs * Ends when: * Monster is dead * Player is dead * Timeout (10 seconds - 350 frames) * Allowed actions: * ATTACK * MOVE\_RIGHT * MOVE\_LEFT   DoomBasic-v0 defines 'solving' as getting average reward of 10.0 over 100 consecutive trials. |  |

* Goal:For this level the **goal** is to **kill the monster** in *three seconds* with *one shot*.
* The rewards: It'll get **+101** points for killing the monster. You get a bad reward with **-5** points for missing a shot. Also get **-1** point on every **0.028** seconds.
* How the game ends: It ends when the Monster is dead or the player is dead or when there is time out.
* Actions: You have three actions for this one attack/shoot, move right and move left.

But it's not the game we're going to play. This one is too simple. We want our AI to move *forward, backward, turn left, turn right* and *shoot* *many monsters*. So we select different level.

* **Level 2:** **DoomCorridor-v0 (experimental) (by @ppaquette)**
* This map is designed to improve your navigation. There is a vest at the *end of the corridor*, with **6** enemies (3 groups of 2). Your goal is to get to the vest as soon as possible, without being killed.

|  |  |
| --- | --- |
| * Goal: 1,000 points. Reach the vest (or get very close to it) * Rewards: * **Plus distance** for getting closer to the vest * **Minus distance** for getting further from the vest * **Minus 100** pts for getting killed * Ends when: * Player touches vest * Player is dead * Timeout (1 minutes - 2,100 frames) * Allowed actions: * ATTACK (shoot) * MOVE\_RIGHT * MOVE\_LEFT * MOVE\_FORWARD * TURN\_RIGHT * TURN\_LEFT |  |

**DoomCorridor-v0** defines 'solving' or getting average reword of **1000.0** over **100** consecutive trials.

* Our goal of course is to not only to shoot the monsters but also reach to the vest and that gets us 1000 points.
* We have to kill the monsters because if we don't kill the monsters we will get killed and we will not be able to reach the vest. (However our AI won't kill them but move too fast to reach the vest).
* The *positive reward is a continuous* reward. If we are getting closer to the vest we'll get **+distance** rewords.
* There are two types of negative rewards, first is a *continuous negative reward* if we move further from the vest, another reward is **-100** points for getting killed.
* The game ends when the player touches the vest or gets killed or timeout.
* We have 6 actions to play the game. In this version, we can move forward, left or right and turn left, turn right and we can shoot. We also have many more monsters.

We're going to build an **AI** to beat that game and to do this we will implement a **Deep Convolutional Neural Network**.

**3.2** step 2: **Folder Structure & Libraries**

First we ***set*** the right folder ***as working directory***. Then **restart kernel**. In this directory, we have 3 files and 1 folder.

* ai.py: it will contain our artificial intelligence and that's nothing else. In this file we'll build our ***Deep Convolutional Q learning model***. We'll implement everything that is related to building the AI.
* experience\_replay.py: This time we put the experience-replay separately because we already implemented it and now we want to focus on the new concepts, techniques and tricks.
* image\_preprocessing.py: This python file will take care of images pre-processing. Since our ***AI*** will have ***eyes*** and the ***input states*** are ***no longer*** encoded by a ***vector***. This time the ***input states*** are the ***images***.
* So the first layer of the **Big Neural Network** that we're about to make will be the eyes and that will be the convolutional layers of the CNN to make sure these images can be accepted as inputs of the Convolutional Neural Network.
* Before *feed* those image, we need to *pre-preprocess* them. So this **image\_preprocessing.py** will take care of pre-processing of these images so that they can go into the *neural network*.
* We used this separate file because it's not directly related to AI.
* videos: The videos folder. Right now this folder is empty.
* When we execute the code, some videos of the AI-playing-Doom will be added to this folder. We can see in the videos how well the AI is doing.
* We will literally see that the AI killing the monsters and trying to run towards the goal.
* Of course the first videos will be very bad because the AI need some times to be trained. After some training, the AI will get better and better and eventually it will manage to kill some monsters by not getting killed.
* Essential libraries: In our **ai.py** file we'll use following libraries.

# *Importing the libraries*

**import** numpy **as** np

**import** torch

**import** torch.nn **as** nn

**import** torch.nn.functional **as** F

**import** torch.optim **as** optim

**from** torch.autograd **import** Variable

# *Importing the packages for OpenAI and Doom*

**import** gym

**from** gym.wrappers **import** SkipWrapper

**from** ppaquette\_gym\_doom.wrappers.action\_space **import** ToDiscrete

# *Importing the other Python files*

**import** experience\_replay, image\_preprocessing

* Libraries to build Neural Network: Following are the essential libraries, will be used to build our AI's Neural Network.
* **numpy:** we'll be working with arrays.
* **torch:** because we're implementing the AI with a torch.
* **torch.nn**: To implement a neural network. It contains all the tools to build a NN. For example, it contain the convolutional layers.
* **torch.nn.functional**: Contains all the functions (typically the activation functions) that are used in a Neural Network. We'll be using some **rectifier-activation** functions but also some **max-pooling** function for the CNN.
* **torch.optim**: for optimizer, we'll be using an Adam-optimizer.
* From **torch.autograd** we'll import **Variable**. It's the real power of **PyTorch** because it contains the dynamic graphs allowing us to perform very *fast computations* of the *gradients* even the *gradients of composition functions*.
* Packages for OpenAI and Doom: We need to import some packages related to open-AI and Doom.

**import** gym

**from** gym.wrappers **import** SkipWrapper

* First we import **gym** then we import some **wrappers** module of the **gym** library and one of these wrappers is **SkipWrapper**. That's basically to *import all the tools* and *environments* of **gym**.
* We have following package that is directly related to Doom.

**from** ppaquette\_gym\_doom.wrappers.action\_space **import** ToDiscrete

* And that's **action\_space** and **ToDiscrete** of the Doom wrapper that basically contains the environment of doom and more specifically the actions that can be played, the number of actions for this specific game we're going to play.
* There are six actions: ATTACK (shoot), MOVE\_RIGHT, MOVE\_LEFT, MOVE\_FORWARD, TURN\_RIGHT, TURN\_LEFT
* Importing the other Python files: Finally we need to import our two internal **py** files, experience.

**import** experience\_replay, image\_preprocessing

* **experience\_replay.py** will import experience-replay and
* **image\_preprocessing.py** will import image preprocessing mechanism. This image preprocessing includes:
* getting the **images** of the **screen**.
* **converted** those image into **NumPy arrays** then **reshaped** to a certain format and then they will go to the **CNN**.
* Implementation structure: We will be implementing this app in three parts.
* The first part will be about building the AI. We'll make the AI's brain i.e. the neural network. It'll be a big CNN composed of some ***convolutional layers*** and then some ***fully connected*** ***layers*** to predict the outputs. The outputs are the Q values.
* Then we'll make the body of the AI.
* The body is the part that will tell the AI ***how to play*** the ***action***.
* The brain will detects the images and predicts the Q-values.
* But then you need to specify how the AI should play the action inside the AI-body. Here we will specify the method of playing the action.
* As for example with our self-driving car (previous app), the ***brain*** was a ***neural network*** and we made the body inside the **DQN class**, where we ***defined*** how the ***action*** was played, inside the method **select\_action()** using **softmax**.
* We'll do the same for this Doom-Playing-AI, we're going to make a brain and we're going to make a body which will play the action. We'll call this **SoftmaxBody**.
* Thirdly, we'll ***implement*** the ***Deep-convolutional Q-learning model*** and there again will have different sections.
* And one of them will be of course to ***train the AI***.
* Then we'll ***test*** the AI.

Next we're going to start by implementing the brain of our AI.

**Al code at once: experience\_replay.py**

# *Experience Replay*

# *Importing the libraries*

**import** numpy **as** np

**from** collections **import** namedtuple, deque

# *Defining one Step*

Step = **namedtuple**('Step', ['state', 'action', 'reward', 'done'])

# *Making the AI progress on several (n\_step) steps*

**class** NStepProgress:

**def** **\_\_init\_\_**(self, env, ai, n\_step):

        self.ai = ai

        self.rewards = []

        self.env = env

        self.n\_step = n\_step

**def** **\_\_iter\_\_**(self):

        state = self**.env.reset**()

        history = **deque**()

        reward = 0.0

**while** **True**:

            action = self**.ai**(**np.array**([state]))[0][0]

            next\_state, r, is\_done, \_ = self**.env.step**(action)

            reward += r

**history.append**(**Step**(state = state, action = action, reward = r, done = is\_done))

**while** **len**(history) **>** self.n\_step + 1:

**history.popleft**()

**if** **len**(history) **==** self.n\_step + 1:

**yield** **tuple**(history)

            state = next\_state

**if** is\_done:

**if** **len**(history) **>** self.n\_step + 1:

**history.popleft**()

**while** **len**(history) **>=** 1:

**yield** **tuple**(history)

**history.popleft**()

                self**.rewards.append**(reward)

                reward = 0.0

                state = self**.env.reset**()

**history.clear**()

**def** **rewards\_steps**(self):

        rewards\_steps = self.rewards

        self.rewards = []

**return** rewards\_steps

# *Implementing Experience Replay*

**class** ReplayMemory:

**def** **\_\_init\_\_**(self, n\_steps, capacity = 10000):

        self.capacity = capacity

        self.n\_steps = n\_steps

        self.n\_steps\_iter = **iter**(n\_steps)

        self.buffer = **deque**()

**def** **sample\_batch**(self, batch\_size): # *creates an iterator that returns random batches*

        ofs = 0

        vals = **list**(self.buffer)

**np.random.shuffle**(vals)

**while** (ofs+1)\*batch\_size **<=** **len**(self.buffer):

**yield** vals[ofs\*batch\_size:(ofs+1)\*batch\_size]

            ofs += 1

**def** **run\_steps**(self, samples):

**while** samples **>** 0:

            entry = **next**(self.n\_steps\_iter) # *10 consecutive steps*

            self**.buffer.append**(entry) # *we put 200 for the current episode*

            samples -= 1

**while** **len**(self.buffer) **>** self.capacity: # *we accumulate no more than the capacity (10000)*

            self**.buffer.popleft**()

**Al code at once: image\_preprocessing.py**

# *Image Preprocessing*

# *Importing the libraries*

**import** numpy **as** np

**from** scipy.misc **import** imresize

**from** gym.core **import** ObservationWrapper

**from** gym.spaces.box **import** Box

# *Preprocessing the Images*

**class** PreprocessImage(ObservationWrapper):

**def** **\_\_init\_\_**(self, env, height = 64, width = 64, grayscale = **True**, crop = **lambda** img: img):

**super**(PreprocessImage, self).**\_\_init\_\_**(env)

        self.img\_size = (height, width)

        self.grayscale = grayscale

        self.crop = crop

        n\_colors = 1 **if** self.grayscale **else** 3

        self.observation\_space = **Box**(0.0, 1.0, [n\_colors, height, width])

**def** **\_observation**(self, img):

        img = self**.crop**(img)

        img = **imresize**(img, self.img\_size)

**if** self.grayscale:

            img = **img.mean**(-1, keepdims = **True**)

        img = **np.transpose**(img, (2, 0, 1))

        img = **img.astype**('float32') / 255.

**return** img