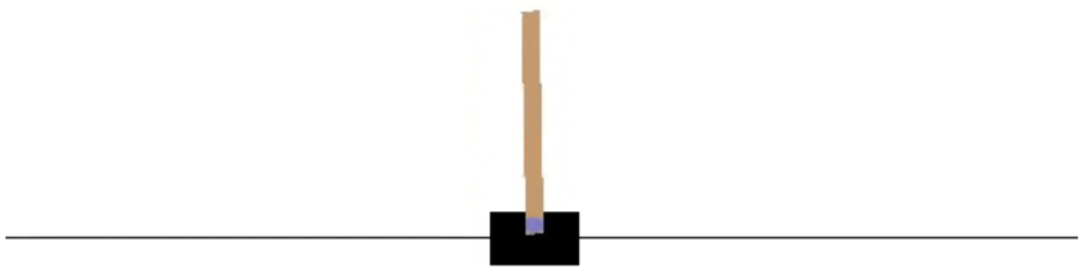
Machine Learning
Lab 10

Implementing Deep Q-Learning in Python using Keras & OpenAI Gym

We will make an agent that can play a game called CartPole.



CartPole is one of the simplest environments in the OpenAI gym (a game simulator). As you can see in the above animation, the goal of CartPole is to balance a pole that's connected with one joint on top of a moving cart.

Instead of pixel information, there are four kinds of information given by the state (such as the angle of the pole and position of the cart). An agent can move the cart by performing a series of actions of 0 or 1, pushing the cart left or right.

We will use the *keras-rl* library here which lets us implement deep Q-learning out of the box.

Step 1: Install keras-rl library

```
%pip install keras-rl2
```

Step 2: Install dependencies for the CartPole environment

Assuming you have pip installed, you need to install the following libraries:

```
%pip install h5py
%pip install gym
```

If you are using Colab, use the following codes to initialise visualization

```
%pip install pygame
```

Step 3: Let's get started!

First, we have to import the necessary modules:

```
import numpy as np
import gym
import tensorflow as tf

from keras.models import Sequential
from keras.layers import Dense, Activation, Flatten
from keras import __version__
tf.keras.__version__ = __version__

from tensorflow.keras.optimizers.legacy import Adam

from rl.agents.dqn import DQNAgent
from rl.policy import EpsGreedyQPolicy
from rl.memory import SequentialMemory
```

For Colab:

```
import os
os.environ['SDL_VIDEODRIVER']='dummy'
import pygame
pygame.display.set_mode((640,480))
```

Then, set the relevant variables:

```
ENV_NAME = 'CartPole-v0'

# Get the environment and extract the number of actions available in the Cartpole problem
env = gym.make(ENV_NAME)
np.random.seed(123)
env.reset(seed=123)
nb_actions = env.action_space.n
```

Next, we will build a very simple single hidden layer neural network model:

```
model = Sequential()
model.add(Flatten(input_shape=(1,) + env.observation_space.shape))
model.add(Dense(16))
model.add(Activation('relu'))
model.add(Dense(nb_actions))
model.add(Activation('linear'))
```

```
print(model.summary())
```

Now, configure and compile our agent. We will set our policy as Epsilon Greedy and our memory as Sequential Memory because we want to store the result of actions we performed and the rewards we get for each action.

```
policy = EpsGreedyQPolicy()
memory = SequentialMemory(limit=50000, window_length=1)
dqn = DQNAgent(model=model, nb_actions=nb_actions, memory=memory,
nb_steps_warmup=10,target_model_update=1e-2, policy=policy)
dqn.compile(Adam(learning_rate=1e-3), metrics=['mae'])
```

Okay, now it's time to learn something! We visualize the training here for show, but this slows down training quite a lot.

```
dqn.fit(env, nb_steps=5000, visualize=True, verbose=2)
```

Test our reinforcement learning model:

```
dqn.test(env, nb_episodes=5, visualize=True)
```

Check the output of our model!

Activity 1

OpenAI gym provides several environments fusing DQN on Atari games. Those who have worked with computer vision problems might intuitively understand this since the input for these are direct frames of the game at each time step, the model comprises of convolutional neural network based architecture.

There are some more advanced Deep RL techniques, such as Double DQN Networks, Dueling DQN and Prioritized Experience replay which can further improve the learning process. These techniques give us better scores using an even lesser number of episodes. I will be covering these concepts in future articles.

I encourage you to try the DQN algorithm on at least 1 environment other than CartPole to practice and understand how you can tune the model to get the best results

Activity 2

Prepare for Essay Assessment!

Please check tutorial ‘Cite Them Right Tutorial’ in Assessment and start your exploration for essay writing!

Extended Reading: Tips for Deep Q Learning

Here are some of the tips and tricks that really helped.

- Having the right model parameter update frequency is important. If you update model weights too often (e.g. after every step), the algorithm will learn very slowly when not much has changed.
- Setting the correct frequency to copy weights from the Main Network to the Target Network also helps improve learning performance. We initially tried to update the Target Network every N episodes which turned
- Using the Huber loss function rather than the Mean Squared Error loss function also helps the agent to learn. The Huber loss function weighs outliers less than the Mean Squared Error loss function.
- The right initialization strategy seems to help. In this case, we use He Initialization for initializing network weights. He Initialization is a good initialization strategy for networks that use the Relu activation function.