Intro to Reinforcement Learning

VTHacks 8

Patrick Riley (rileyp@vt.edu) and Andrew Farabow (aafarabow@vt.edu)

Plan for Today

Today we will:

- Describe what deep reinforcement learning is
- Describe and implement the DQN algorithm
- Solve a simple game using pytorch and OpenAIGym
- Suggest some next steps to continue your learning!

Link to Github Repo with the notebook for today

What is Reinforcement Learning?

Area of machine learning that deals with with maximizing some kind of reward by choosing actions to take in an environment

Key terms:

- Agent - whatever is taking the actions
- Environment - an interactive setting that the agent interacts with. Think of this like a game. The environment could be an Atari game, a robotics simulator, or a power grid
- Reward - a scalar value that measures how well the agent is doing

Generally, an RL system is comprised of an agent that interacts with the environment, receives a reward, and through trial and error works to maximize that reward

What is Reinforcement Learning? (part 2)

Some old-school RL algorithms use a table that stores the reward from every state/action combination experienced by the algorithm

At any given state, they refer to the table to find the best action.

This becomes infeasible in environments with many possible states and actions - aka every possible problem we would be interested in.

Solution: find some way to approximate the value of unseen states and actions based on similar ones seen in the past

What is Deep Learning?

Deep learning, also known as deep neural networks, provide a way to do exactly that.

Neural networks approximate functions. For a function f(x), you give it an x, it takes a guess at f(x), then you show it f(x) and it improves itself. Repeat for all the data you have.

Specifics could fill a whole talk, so we won’t go there.

For the RL algorithm we are going to study, the function is called Q(s, a) where s is a state, a is an action, and Q(s, a) is the reward received when action a is taken is state s

Tools/Setup

Requirements:

- Python 3.0 or higher
  - Jupyter
  - Notebook
  - Gym
  - Torch
  - Numpy
- Use the requirements.txt file and follow the instructions in the github repo to streamline this process

Interactivity

- We have omitted five lines of code from our notebook
- At various points in the workshop, we will ask you to write one of more of these lines based on a short description

WE DO NOT EXPECT YOU TO GET ALL OF THESE RIGHT

- The goal is to make you think about the material instead of having it just be read to you
- You may wish to consult the Python, PyTorch or OpenAI Gym documentation
- You can run the notebook to test your implementation
- We will give you the correct solution after you have considered it for a few minutes

Let’s Go!

Hyperparameters

- These control some of the higher-level elements of our training
  - ε (epsilon) - Controls the exploration vs exploitation balance as we train, which allows our agent to escape poor solutions
  - γ (gamma) - Reward discount factor, so that more recent events are considered more important
  - τ (tau) - Controls how quickly we update the target network, which determines how rapidly the network updates
  - batch_size - How many past states, actions, and rewards to look at for each update from the replay buffer
  - max_ep - Number of games to play during the training session
- These are often what you will change to improve your model’s performance and efficiency once you implement a particular algorithm

Q Network

- Where the “deep” part comes in
- Approximates a function that takes the state of the game and an action as input, and then outputs the expected reward
- In practice, we only give Q the state, and it outputs a vector of rewards for every possible action.
    - Write this as Q(s)
    - We determine the optimal action by finding the one corresponding to the biggest reward
        - write this as formally as argmax(Q(s, a), a) or in code as torch.argmax(Q(s))
- Pytorch allows you to easily create a neural network and customize the number of layers, the number of nodes in each layer, and the types of layers
- We use fully connected layers, and then activation function layers
    - Activation functions allow the network to learn non-linear functions
- forward() function is what get called when you want to pass values through the network

Replay Buffer

- Just a fancy list
- We sample from this when it's time to update our current network
- Pass in current state, action, reward, done, and next state at each step of training and exploring

Exploration

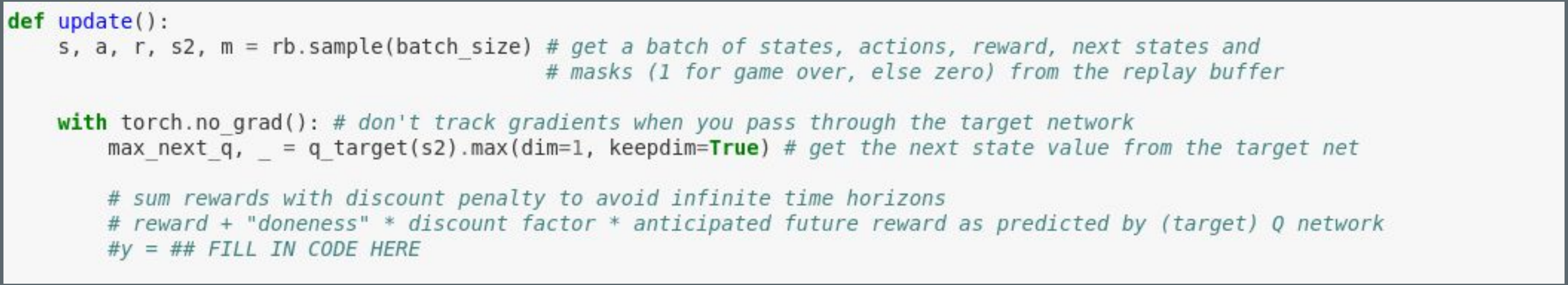
- This step is critical for many different RL algorithms
- Fills up the replay buffer
- Allows the agent to experience different parts of the state space, reducing the chance of it falling into a local minimum
- Also increases training speed by providing more states to sample from when updating the network
- Agent simply takes random actions for a specified number of steps

Update

- Where the Q network improves itself
- Gets a batch of states, actions, reward, next states, and masks from the replay buffer
    - Masks simply tell whether the game ended on that state
- Get the reward and the (target) Q network’s estimate of the reward for the next state
- Sum that reward and expected future reward, but discount the expected future reward by multiplying by hyperparameter gamma <1
- Calculate the loss between that and the network’s estimate, Q(s)
- Perform backpropagation and update the network (handled by PyTorch)
- Update the target network to be more like the actual network

Update Code #1

- The line we want you to write defines a variable called y
- y is calculated from the data we sampled from the replay buffer
- We want to train the Q network to predict y
- If the game is over, y = rewards, otherwise, y = rewards + discount factor * anticipated future rewards
  - Hint: discount factor is in a constant called “gamma”
  - Hint: use the “m” variable to calculate this without an if else statement. m=0 if the game is



```
def update():
    s, a, r, s2, m = rb.sample(batch_size) # get a batch of states, actions, reward, next states and
                                           # masks (1 for game over, else zero) from the replay buffer

    with torch.no_grad(): # don't track gradients when you pass through the target network
        max_next_q, _ = q_target(s2).max(dim=1, keepdim=True) # get the next state value from the target net

        # sum rewards with discount penalty to avoid infinite time horizons
        # reward + "doneness" * discount factor * anticipated future reward as predicted by (target) Q network
        #y = ## FILL IN CODE HERE
```

Update Code #1 - Solution

```
y = r + m * gamma * max_next_q
```

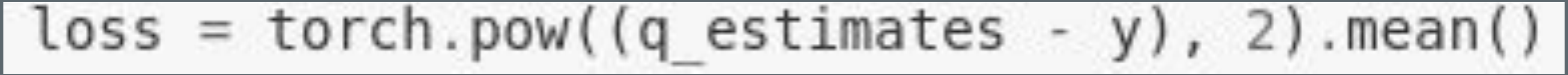
Update Code #2

- To evaluate how good of an estimate for y q_estimates is, you need to compute the mean squared error loss between y and q_estimtaes
- If q_estimates is close to y, the loss will be low

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

- Remember: q_estimates and y are tensors and every element is a different value that was sampled from the replay buffer
- Useful PyTorch functions: torch.pow(base, power) and .mean()

Update Code #2 - Solution



```
loss = torch.pow((q_estimates - y), 2).mean()
```

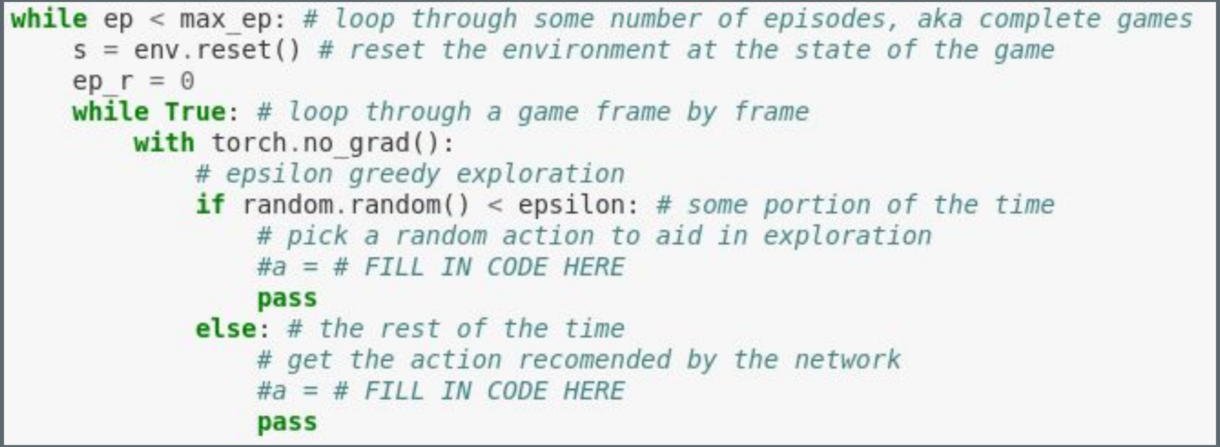
You could also use PyTorch’s built in MSE Loss function

Training

- Where the model “learns”
- Explore first
- Loop for the desired number of games
  - Play until you lose or the game ends
  - At each step, either take the action recommended by the network or take a random one (more exploration)
  - Record that info in the replay buffer, and repeat
- After each game, update the network as described previously
- That’s it!

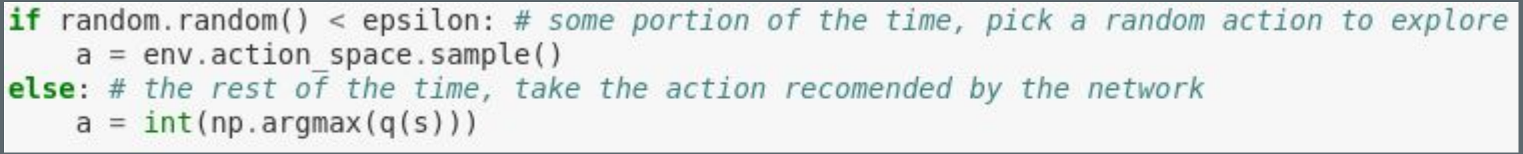
Training Code #1-2

1. Exploration case: look up the OpenAI Gym API and try to figure out how to get an action randomly (hint: look at the explore method, we do this there too) OR use Python's built-in features to randomly set a to either 0 or 1
2. Greedy case: pass the neural network the input it expects. It will output a vector with two dimensions. Set a to 0 or 1, depending on which position in the vector has a higher value



```
while ep < max_ep: # loop through some number of episodes, aka complete games
    s = env.reset() # reset the environment at the state of the game
    ep_r = 0
    while True: # loop through a game frame by frame
        with torch.no_grad():
            # epsilon greedy exploration
            if random.random() < epsilon: # some portion of the time
                # pick a random action to aid in exploration
                #a = # FILL IN CODE HERE
                pass
            else: # the rest of the time
                # get the action recomended by the network
                #a = # FILL IN CODE HERE
                pass
```

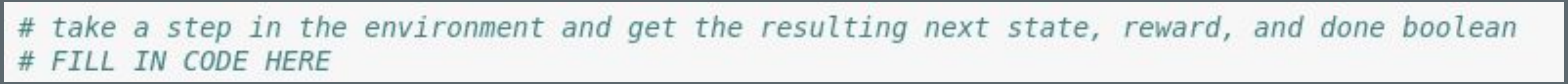
Training Code #1 - Solution



```python
if random.random() < epsilon: # some portion of the time, pick a random action to explore
    a = env.action_space.sample()
else: # the rest of the time, take the action recomended by the network
    a = int(np.argmax(q(s)))
```

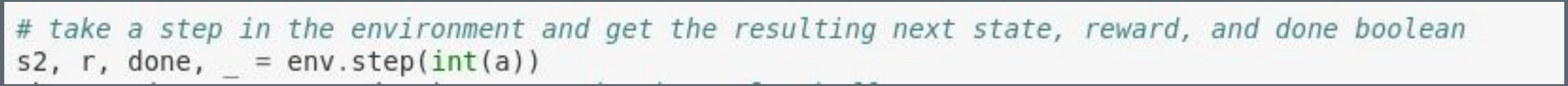
Training Code #3

- Take a step in the environment. You will have to look this up, or find somewhere else in the code where we did it.
- This produces a state, reward, boolean representing whether the game is done, and a dictionary of info. You can discard the info, but the rest are used.
  - Look at the context and guess what these values are called
  - We will tell you right now that the state is called s2, because that is confusing
  - In Python, functions can return multiple outputs. Ex: “var1, var2, var3 = return_three()”



```
# take a step in the environment and get the resulting next state, reward, and done boolean
# FILL IN CODE HERE
```

Training Code #3 - Solution



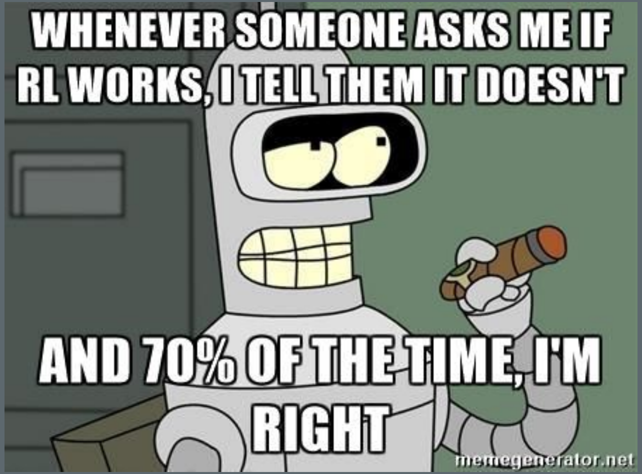
```
# take a step in the environment and get the resulting next state, reward, and done boolean
s2, r, done, _ = env.step(int(a))
```

Next Steps

- Read the research paper and dig into the math: https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf
- Check out other algorithms like Policy Gradients and see how they compare
- Read more about PyTorch
- Read Andrew’s blog posts (shameless-self promotion)
- Read Sutton and Barto's Reinforcement Learning, second edition: An Introduction (for the mathematically inclined, this is THE RL textbook)

Questions?

Something not get answered in this workshop? Ping either of us on Discord!


WHENEVER SOMEONE ASKS ME IF RL WORKS, I TELL THEM IT DOESN'T
AND 70% OF THE TIME, I'M RIGHT
memegenerator.net

Thanks for attending, and enjoy the rest of the event!