Reinforcement Learning

StartOnAI Deep Learning Tutorial 6

This tutorial covers Reinforcement Learning and its applications

What is Reinforcement Learning and how is it different than supervised and unsupervised learning?

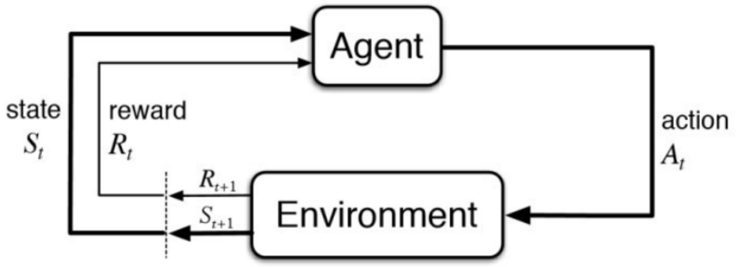
In the field of machine learning, reinforcement learning is the process in which a mathematical/algorithmic model makes decisions in a dynamically changing environment in some optimal sequence. This is drastically different from both supervised and unsupervised learning. In supervised learning algorithms a model is given a dataset with corresponding labels and the model tries to optimize its parameters to predict the labels of a different set of data. In this type of learning we try to train our model to predict the “right” answer. In unsupervised learning the goal is to find features and trends inside the data.

Using a reinforcement learning model is more desirable than using a supervised learning model when it's hard to label a “right” answer in the dataset. For example in a game of checkers, the amount of ways a player can win given a certain board configuration is endless. It is much more reasonable for the model to gain intuition through extended play-outs of trial and error rather than “labeling” what move should be carried out in advance. Unsupervised learning is usually used for very different problems since it simply tries to observe patterns and trends rather than maximize some reward.

Reinforcement learning attempts to find the optimal reward through some agent which seeks to carry out actions in an environment in response to some state. Before we go on further lets define some important terms related to reinforcement learning(credit for definitions go to this article):

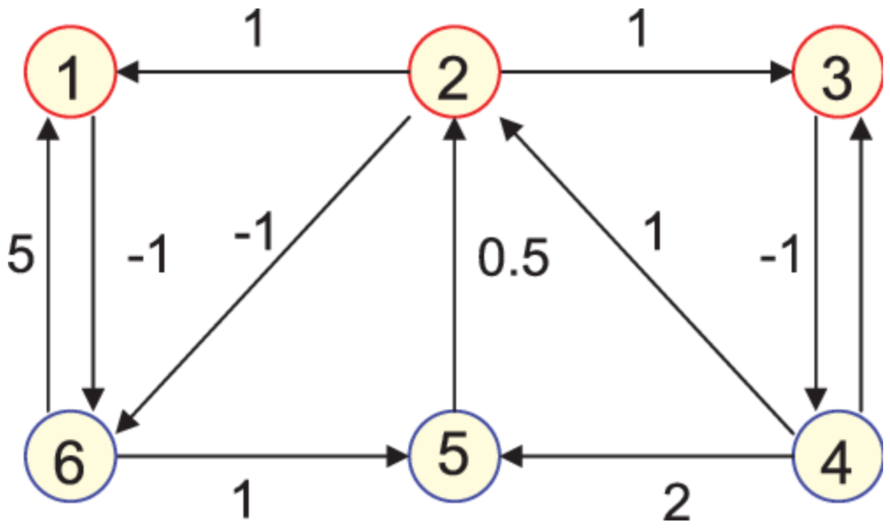
- Action (A): All the possible moves that the agent can take
- State(S): Current situation returned by the environment
- Reward(R): An immediate return send back from the environment to evaluate the last action.
- Policy(π) : The strategy that the agent employs to determine the next action based on the current state.

- Value (V): The expected long-term return with discount, as opposed to the short-term reward R. *Vπ(s)* is defined as the expected long-term return of the current state under policy π.
- Q-value or action value(Q): Q-value is similar to Value, except that it takes an extra parameter, the current action *a*. *Qπ(s, a)* refers to the long-term return of the current state *s*, taking action a under policy π.



Flowchart of an RL algorithm (Source: https://i.stack.imgur.com/eoeSq.png)

The reinforcement learning process described above is also defined as a **Markov Decision Process.** Our action A incites something in an environment which returns two things: a new state S and a reward R that accompanies the action. We can try to visualize this process with a weighted and directed graph:



If we represent our reinforcement model as a graph then these are some properties of a graph that would correspond to the model:

- Nodes: States of the model
- Edges: Possible actions between states
- Weights: Rewards of the actions between the states. Positive weights represent rewards, while negative weights represent punishment
- Selecting the path to travel: The policy which decides what actions to carry out

If we are trying to gain the maximum reward by traversing this graph, then arguably the most difficult part is choosing the path. One method for choosing such as path is called the **epsilon greedy** which is essentially a greedy algorithm that chooses the local optimal path while traversing the graph.There are many different strategies we can choose to maximize reward:

- **Policy based model:** we want to find the policy that maximizes the reward
- **Value based model:** we simply focus on maximizing the total reward in the long run
- **Action based model:** we try to find the best type of actions to take in a particular state

Introduction to the Q-Learning technique:

In Q-learning we use a reward function Q with parameters s and a. Thus Q(s,a) represents the reward that can be achieved in state s after carrying out action a. The Q function can be simplified to :

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma max_{a'} Q(s',a') - Q(s,a)]$$

Updated Q-Learning Equation

In the formula r represents the reward gained after carrying out action a. Furthermore s' and a' are future states and actions. α is a hyperparameter that represents the learning rate. It is usually set to a value between 0 and 1.γ is also a hyper parameter between 0 and 1 and is called the discount factor. This helps to account for the fact that immediate actions result in higher rewards than rewards gained by actions in the future.

On an algorithmic level this image generalizes the process of Q-learning:

```
Initialize Q(s, a) arbitrarily
Repeat (for each episode):
   Initialize s
   Repeat (for each step of episode):
      Choose a from s using policy derived from Q
         (e.g., $\varepsilon$-greedy)
      Take action a, observe r, s'
      $Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma \max_{\alpha'} Q(s', a') - Q(s, a)]$
      s <-- s';
   until s is terminal
```

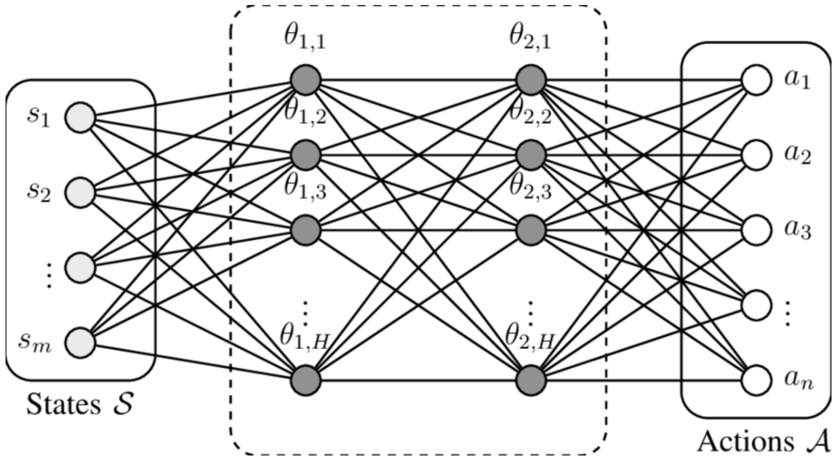
SARSA:

This is simply a slight variation of Q-learning. It employs the same concept of state action pairs but uses a slightly different transition formula. Instead of simply taking the maximum reward state-action pair when selecting s' and a', it uses the same policy when s and a were selected(eg. Epsilon greedy) for s' and a'. Here is the algorithm:

```
Initialize Q(s, a) arbitrarily
Repeat (for each episode):
   Initialize s
   Choose a from s using policy derived from Q
         (e.g., ε-greedy)
   Repeat (for each step of episode):
      Take action a, observe r, s'
      Choose a' from s' using policy derived from Q
         (e.g., ε-greedy)
      Q(s, a) <-- Q(s, a) + α[r + γQ(s', a')- Q(s, a)]
      s <-- s'; a <-- a';
   until s is terminal
```

DeepQ learning:

Sometimes the number of given states in a problem exceeds the memory capacity of a machine and thus storing every state Q(s,a) is unfeasible. Thus we need a very good approximation of the Q-function that does not take up so much memory. Using a Deep Q network is a feasible option which uses a neural network of a state in order to approximate the best action to take in the given situation. The neural network would use a loss function that would compare the true value of Q(s,a) and the predicted value.



Monte Carlo methods:

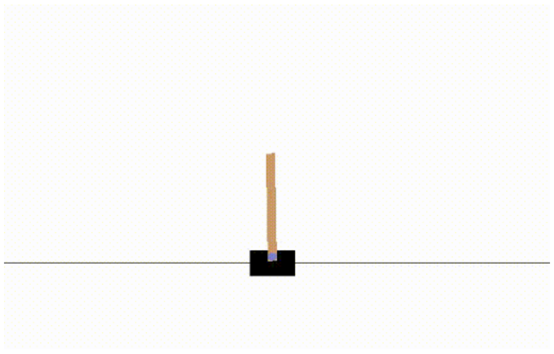
A definition from here:

"The Monte Carlo method for reinforcement learning learns directly from episodes of experience without any prior knowledge of MDP transitions. Here, the random component is the return or reward."

Introduction to Cartpole problem and applications of Q-learning:

Here is a good explanation of the cart pole problem from this article:

"The cart pole problem is a famous problem in dynamics and control theory, with a pendulum whose center of gravity is above the pivot point. This naturally creates an unstable system and the pendulum will typically remain vertically downward without any force of swing or dynamic control being applied. The cartpole has one degree of freedom upon its axis, and the system has no vertical movement. The goal of most cartpole systems is to effectively keep the cartpole balanced by applying various forces on the pivot point and its axis of movement (the horizontal direction)."



The distinct parameters of the states in this problem would be the velocity, position, angle, and pole velocity which is represented by (θ, l, x, $x2$). The action space would be to either exert a constant left force or constant right force. Since states are shifting after a very small portion of time we must resort to a deep q-network to save memory. After training our network we essentially constantly feed the current state of the cart pole to the network and obtain the corresponding action after a forward pass. The rewards of the states are high if the angle of the cartpole is relatively farther from the surface and low if it goes out of bounds or is very stretched.

More resources:

https://www.geeksforgeeks.org/what-is-reinforcement-learning/

https://www.youtube.com/watch?v=zR11FLZ-O9M
https://medium.com/@zsalloum/monte-carlo-in-reinforcement-learning-the-easy-way-564c53010511
https://pathmind.com/wiki/deep-reinforcement-learning
https://towardsdatascience.com/applications-of-reinforcement-learning-in-real-world-1a94955bcd12

Citations:

Sharma, Siddharth. “The Ultimate Beginner's Guide to Reinforcement Learning.” *Medium*, Towards Data Science, 5 Apr. 2020, towardsdatascience.com/the-ultimate-beginners-guide-to-reinforcement-learning-588c071af1ec.

“Reinforcement Learning.” *Reinforcement Learning - Algorithms*, www.cse.unsw.edu.au/~cs9417ml/RL1/algorithms.html.

Choudhary, Ankit, and IIT Bombay Graduate. “Reinforcement Learning: Monte Carlo Learning Using OpenAI Gym.” *Analytics Vidhya*, 6 May 2019, www.analyticsvidhya.com/blog/2018/11/reinforcement-learning-introduction-monte-carlo-learning-openai-gym/.

Shaikh, Faizan, and Faizan. “Beginner's Guide to Reinforcement Learning & Its Implementation in Python.” *Analytics Vidhya*, 24 June 2019, www.analyticsvidhya.com/blog/2017/01/introduction-to-reinforcement-learning-implementation/.