Simple Beginner's guide to Reinforcement Learning & its implementation

Introduction

One of the most fundamental question for scientists across the globe has been – "How to learn a new skill?". The desire to understand the answer is obvious – if we can understand this, we can enable human species to do things we might not have thought before. Alternately, we can train machines to do more "human" tasks and create true artificial intelligence.

While we don't have a complete answer to the above question yet, there are a few things which are clear. Irrespective of the skill, we first learn by interacting with the environment. Whether we are learning to drive a car or whether it an infant learning to walk, the learning is based on the interaction with the environment. Learning from interaction is the foundational underlying concept for all theories of learning and intelligence.

Re-inforcement Learning

Today, we will explore Reinforcement Learning – a goal-oriented learning based on interaction with environment. Reinforcement Learning is said to be the hope of true artificial intelligence. And it is rightly said so, because the potential that Reinforcement Learning possesses is immense.

Reinforcement Learning is growing rapidly, producing wide variety of learning algorithms for different applications. Hence it is important to be familiar with the techniques of reinforcement learning. If you are not familiar with reinforcement learning, I will suggest you to go through my previous article on introduction to reinforcement learning and the open source RL platforms.

Once you have an understanding of underlying fundamentals, proceed with this article. By the end of this article you will have a thorough understanding of Reinforcement Learning and its practical implementation.

P.S. *For implementation, we assume that you have basic knowledge of Python. If you don't know Python, you should first go through this tutorial*

Table of Content

1. Formulating a Reinforcement Learning Problem

Reinforcement Learning is learning what to do and how to map situations to actions. The end result is to maximize the numerical reward signal. The learner is not told which action to take, but instead must discover which action will yield the maximum reward. Let's understand this with a simple example below.

Consider an example of a child learning to walk.

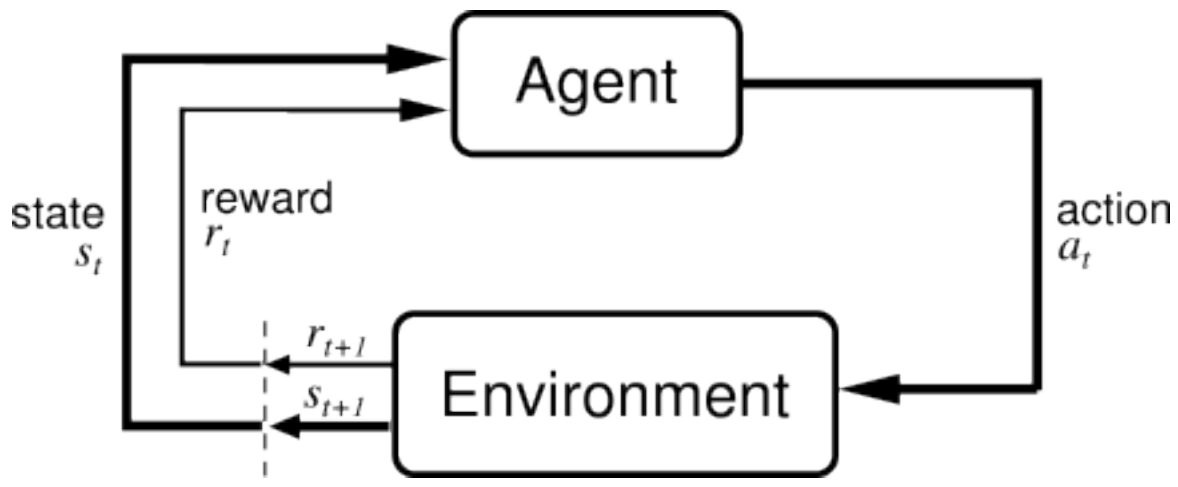
Here are the steps a child will take while learning to walk:

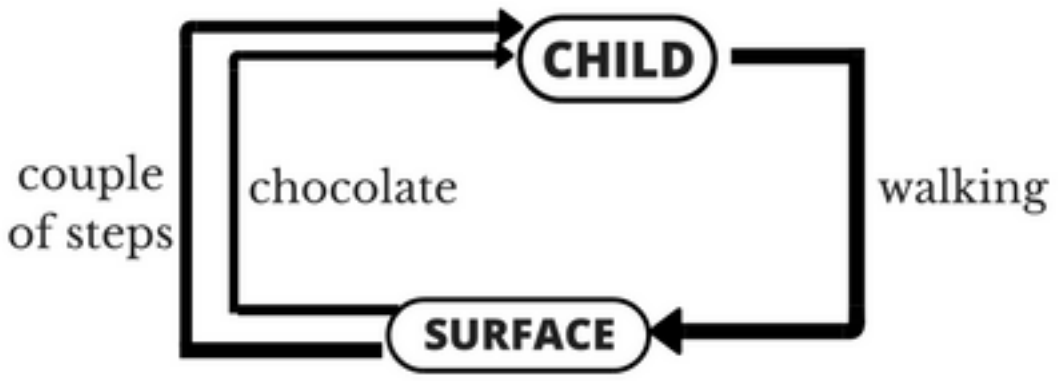
1. The first thing the child will observe is to **notice** how you are walking. You use two legs, taking a step at a time in order to walk. Grasping this concept, the child tries to replicate you.

2. But soon he/she will understand that before walking, the child has to stand up! This is a challenge that comes along while trying to walk. So now the child **attempts to get up,** staggering and slipping but still determinant to get up.
3. Then there's another challenge to cope up with. Standing up was easy, but to **remain still** is another task altogether! Clutching thin air to find support, the child manages to stay standing.
4. Now the real task for the child is to start walking. But it's easy to say than actually do it. There are so **many things to keep in mind**, like balancing the body weight, deciding which foot to put next and where to put it.

Sounds like a difficult task right? It actually is a bit challenging to get up and start walking, but you have become so use to it that you are not fazed by the task. But now you can get the gist of how difficult it is for a child.

Let's formalize the above example, the "problem statement" of the example is **to walk**, where **the child is an agent** trying to manipulate the **environment (which is the surface on which it walks)** by **taking actions (viz walking)** and he/she tries to go from one **state (viz each step he/she takes)** to another. The child gets a **reward (let's say chocolate)** when he/she accomplishes a **submodule of the task (viz taking couple of steps)** and will not receive any chocolate **(a.k.a negative reward)** when he/she is not able to walk. This is a simplified description of a reinforcement learning problem.

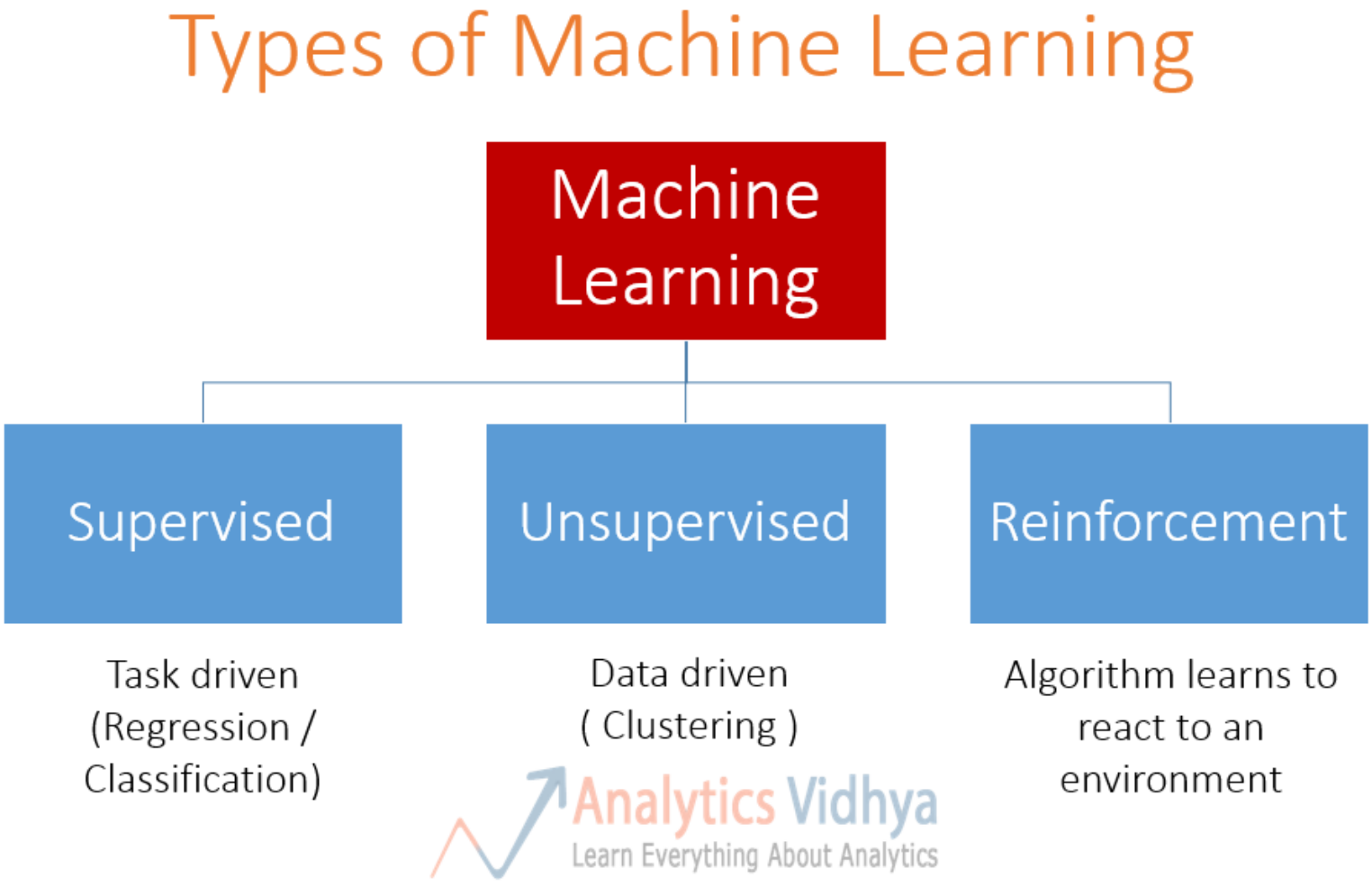




Here's a good introductory video on Reinforcement Learning.

2. Comparison with other machine learning methodologies

Reinforcement Learning belongs to a bigger class of machine learning algorithm. Below is the description of types of machine learning methodologies.



Let’s see a comparison between RL and others:

- **Supervised vs Reinforcement Learning**: In supervised learning, there’s an

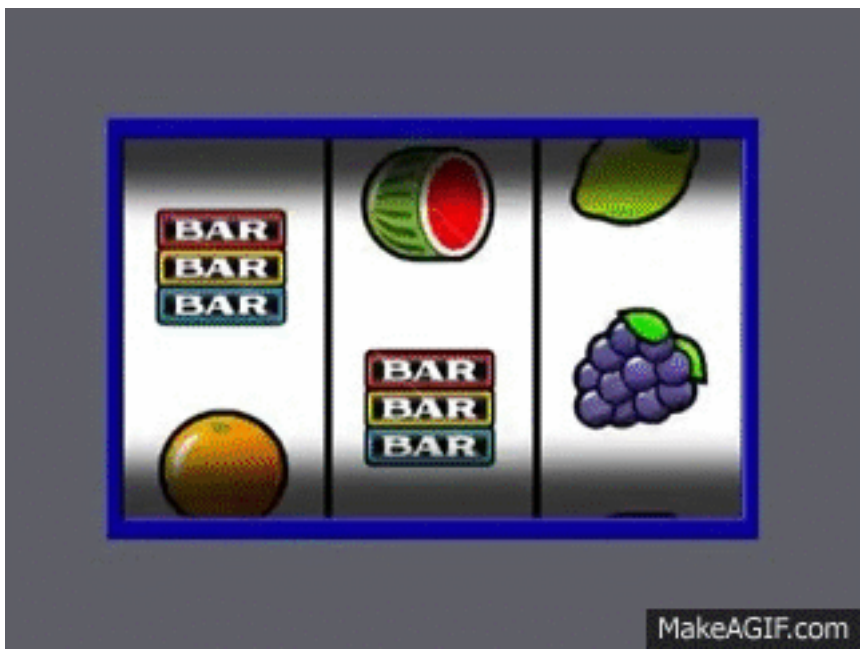
external “supervisor”, which has knowledge of the environment and who shares it with the agent to complete the task. But there are some problems in which there are so many combinations of subtasks that the agent can perform to achieve the objective. So that creating a “supervisor” is almost impractical. For example, in a chess game, there are tens of thousands of moves that can be played. So creating a knowledge base that can be played is a tedious task. In these problems, it is more feasible to learn from one’s own experiences and gain knowledge from them. This is the main difference that can be said of reinforcement learning and supervised learning. In both supervised and reinforcement learning, there is a mapping between input and output. But in reinforcement learning, there is a reward function which acts as a feedback to the agent as opposed to supervised learning.

- **Unsupervised vs Reinforcement Leanring**: In reinforcement learning, there’s a mapping from input to output which is not present in unsupervised learning. In unsupervised learning, the main task is to find the underlying patterns rather than the mapping. For example, if the task is to suggest a news article to a user, an unsupervised learning algorithm will look at similar articles which the person has previously read and suggest anyone from them. Whereas a reinforcement learning algorithm will get constant feedback from the user by suggesting few news articles and then build a “knowledge graph” of which articles will the person like.

There is also a fourth type of machine learning methodology called **semi-supervised** learning, which is essentially a combination of supervised and unsupervised learning. It differs from reinforcement learning as similar to supervised and semi-supervised learning has direct mapping whereas reinforcement does not.

3. Framework for solving Reinforcement Learning Problems

To understand how to solve a reinforcement learning problem, let’s go through a classic example of reinforcement learning problem – Multi-Armed Bandit Problem. First, we would understand the fundamental problem of exploration vs exploitation and then go on to define the framework to solve RL problems.



Suppose you have many slot machines with random payouts. A slot machine would look something like this.

Now you want to do is get the maximum bonus from the slot machines as fast as possible. What would you do?

One naive approach might be to select only one slot machine and keep pulling the lever all day long. Sounds boring, but it may give you “some” payouts. With this approach, you might hit the jackpot (with a probability close to 0.00000....1) but most of the time you may just be sitting in front of the slot machine losing money. Formally, this can be defined as a **pure exploitation** approach. Is this the optimal choice? The answer is NO.

Let’s look at another approach. We could pull a lever of each & every slot machine and pray to God that at least one of them would hit the jackpot. This is another naive approach which would keep you pulling levers all day long, but give you sub-optimal payouts. Formally this approach is a **pure exploration** approach.

Both of these approaches are not optimal, and we have to find a proper balance

between them to get maximum reward. This is said to be **exploration vs exploitation dilemma** of reinforcement learning.

First, we formally define the framework for reinforcement learning problem and then list down the probable approaches to solve the problem.

Markov Decision Process:

The mathematical framework for defining a solution in reinforcement learning scenario is called **Markov Decision Process**. This can be designed as:

- Set of states, S
- Set of actions, A
- Reward function, R
- Policy, π
- Value, V

We have to take an action (A) to transition from our start state to our end state (*S*). In return getting rewards (R) for each action we take. Our actions can lead to a positive reward or negative reward.

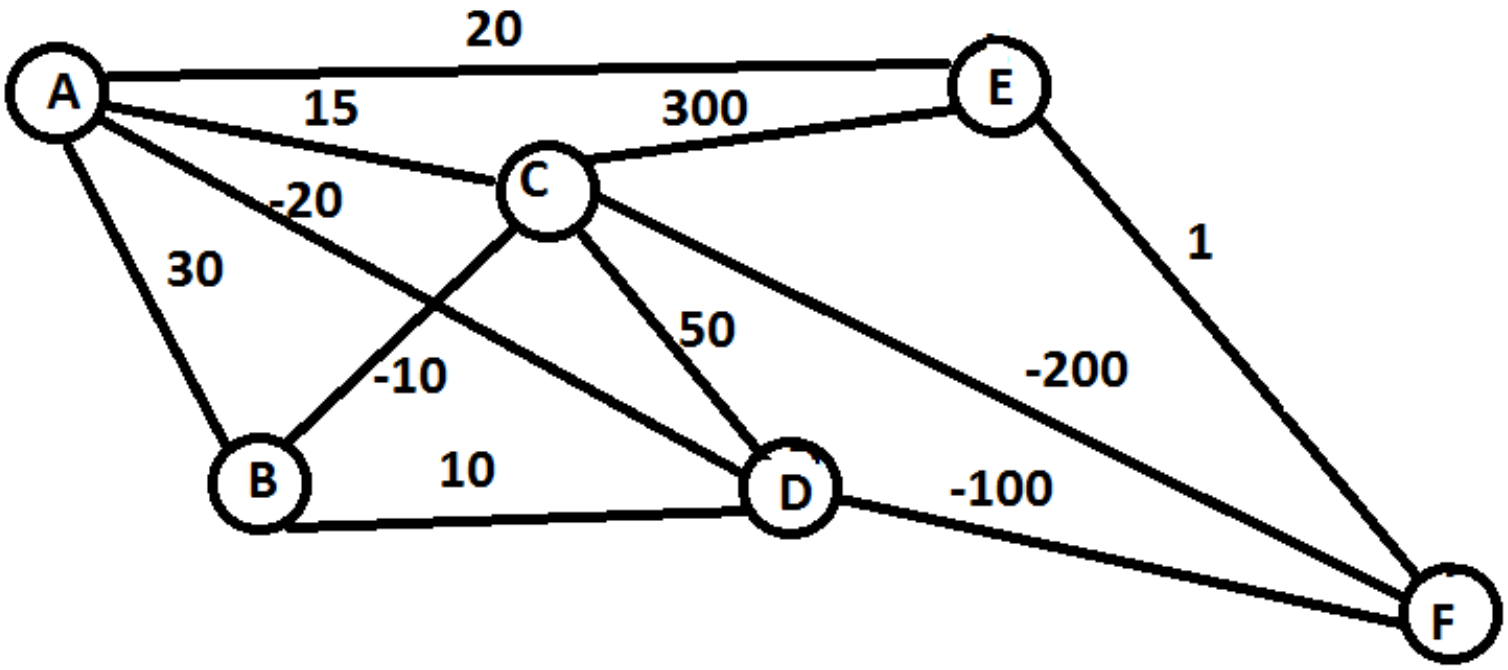
The set of actions we took define our policy (π) and the rewards we get in return defines our value (V). Our task here is to maximize our rewards by choosing the correct policy. So we have to maximize

$$E(r_t \mid \pi, s_t)$$

for all possible values of *S* for a time t.

Travelling Salesman Problem

Let me take you through another example to make it clear.



This is a representation of a traveling salesman problem. The task is to go from place A to place F, with as low cost as possible. The numbers at each edge between two places represent the cost taken to traverse the distance. The negative cost are actually some earnings on the way. We define Value is the total cumulative reward when you do a policy.

Here,

- The set of states are the nodes, viz {A, B, C, D, E, F}
- The action to take is to go from one place to other, viz {A -> B, C -> D, etc}
- The reward function is the value represented by edge, i.e. cost
- The policy is the “way” to complete the task, viz {A -> C -> F}

Now suppose you are at place A, the only visible path is your next destination and anything beyond that is not known at this stage (a.k.a observable space).

You can take a greedy approach and take the best possible next step, which is going from {A -> D} from a subset of {A -> (B, C, D, E)}. Similarly now you are at place D and want to go to place F, you can choose from {D -> (B, C, F)}. We see that {D -> F} has the lowest cost and hence we take that path.

So here, our policy was to take {A -> D -> F} and our Value is -120.

Congratulations! You have just implemented a reinforcement learning algorithm. This algorithm is known as **epsilon greedy,** which is literally a greedy approach to solving the problem. Now if you (the salesman) want to go from place A to place F again, you would always choose the same policy.

Other ways of travelling?

Can you guess which category does our policy belong to i.e. (pure exploration vs pure exploitation)?

Notice that the policy we took is not an optimal policy. We would have to “explore” a little bit to find the optimal policy. The approach which we took here is policy based learning, and our task is to find the optimal policy among all the possible policies. There are different ways to solve this problem, I’ll briefly list down the major categories

- **Policy based,** where our focus is to find optimal policy
- **Value based,** where our focus is to find optimal value, i.e. cumulative reward
- **Action based,** where our focus is on what optimal actions to take at each step

I would try to cover in-depth reinforcement learning algorithms in future articles. Till then, you can refer to this paper on a survey of reinforcement learning algorithms.

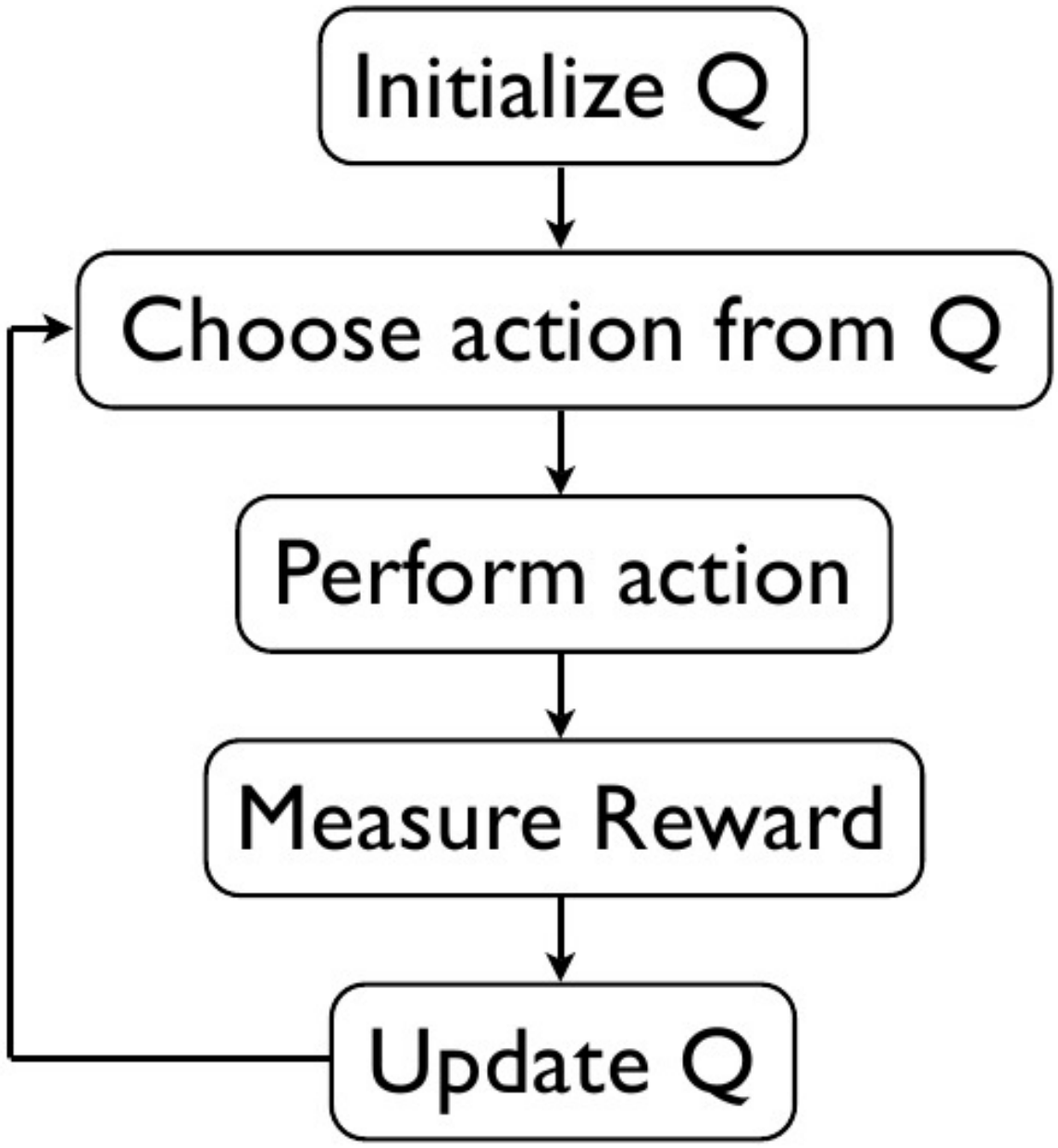
4. An implementation of Reinforcement Learning

We will be using Deep Q-learning algorithm. Q-learning is a policy based learning algorithm with the function approximator as a neural network. This algorithm was used by Google to beat humans at Atari games!

Let’s see a pseudocode of Q-learning:

1. Initialize the Values table ‘**Q(s, a)’**.
2. Observe the current state ‘**s’**.
3. Choose an action **‘a’** for that state based on one of the action selection policies (eg. epsilon greedy)
4. Take the action, and observe the reward **‘r’** as well as the new state **‘s’**.
5. Update the Value for the state using the observed reward and the maximum reward possible for the next state. The updating is done according to the formula and parameters described above.
6. Set the state to the new state, and repeat the process until a terminal state is reached.

A simple description of Q-learning can be summarized as follows:



We will first see what Cartpole problem is then go on to coding up a solution

When I was a kid, I remember that I would pick a stick and try to balance it on one hand. Me and my friends used to have this competition where whoever balances it for more time would get a “reward”, a chocolate!

Here’s a short video description of a real cart-pole system

Let’s code it up!

To setup our code, we need to first install a few things,

Step 1: Install keras-rl library

From terminal, run the following commands:

```
git clone https://github.com/matthiasplappert/keras-rl.git
cd keras-rl
python setup.py install
```

Step 2: Install dependencies for CartPole environment

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

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

Step 3: lets get started!

First we have to import modules that are necessary

```
import numpy as np
import gym

from keras.models import Sequential
```

```
from keras.layers import Dense, Activation, Flatten
from keras.optimizers import Adam

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

Then set the relevant variables

```
ENV_NAME = 'CartPole-v0'

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

Next, we 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())
```

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

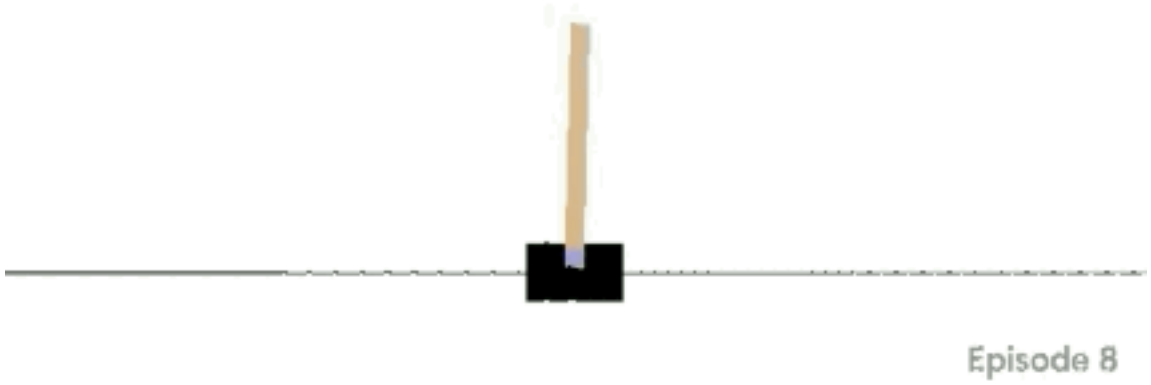
```
policy = EpsGreedyQPolicy()
memory = SequentialMemory(limit=50000, window_length=1)
dqn = DQNAgent(model=model, nb_actions=nb_actions, memory=memory, nb_steps_warmup=10,
target_model_update=1e-2, policy=policy)
dqn.compile(Adam(lr=1e-3), metrics=['mae'])

# Okay, now it's time to learn something! We visualize the training here for show, but t
dqn.fit(env, nb_steps=5000, visualize=True, verbose=2)
```

Now we test our reinforcement learning model

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

This will be the output of our model:



And Voila! You have just built a reinforcement learning bot!

5. Increasing the complexity

Now that you have seen a basic implementation of Re-inforcement learning, let us start moving towards a few more problems, increasing the complexity little bit every time.

Problem – Towers of Hanoi

For those, who don't know the game – it was invented in 1883 and consists of 3 rods along with a number of sequentially-sized disks (3 in the figure above) starting at the leftmost rod. The objective is to move all the disks from the leftmost rod to the rightmost rod **with the least number of moves**. (You can read more on wikipedia)

If we have to map this problem, let us start with states:

- **Starting state** – All 3 disks in leftmost rod (in order 1, 2 and 3 from top to bottom)
- **End State** – All 3 disks in rightmost rod (in order 1, 2 and 3 from top to bottom)

All possible states:

Here are our 27 possible states:

All disks in a rod	One disk in a Rod	(13) disks in a rod	(23) disks in a rod	(12) disks in a rod
(123)**	321	(13)2*	(23)1*	(12)3*
(123)	312	(13)*2	(23)*1	(12)*3
**(123)	231	2(13)*	1(23)*	3(12)*
	132	*(13)2	*(23)1	*(12)3
	213	2*(13)	1*(23)	3*(12)
	123	*2(13)	*1(23)	*3(12)

Where (12)3* represents disks 1 and 2 in leftmost rod (top to bottom) 3 in middle rod and * denotes an empty rightmost rod

Numerical Reward:

Since we want to solve the problem in least number of steps, we can attach a reward of -1 to each step.

Policy:

Now, without going in any technical details, we can map possible transitions between above states. For example (123)** -> (23)1* with reward -1. It can also go to (23)*1

If you can now see a parallel, each of these 27 states mentioned above can represent a graph similar to that of travelling Salesman above and we can find the most optimal solutions by experimenting various states and paths.

Problem – 3 x 3 Rubix Cube

While I can solve this for you as well, I would want you to do this by yourself. Follow the same line of thought I used above and you should be good.

Start by defining the Starting state and the end state. Next, define all possible states and their transitions along with reward and policy. Finally, you should be able to create a solution for solving a rubix cube using the same approach.

6. A Peek into Recent Advancements in Reinforcement Learning

As you would realize that the complexity of this Rubix Cube is many folds higher than the Towers of Hanoi. You can also understand how the possible number of options have increased in number. Now, think of number of states and options in a game of Chess and then in Go! Google DeepMind recently created a deep reinforcement learning algorithm which defeated Lee Sedol!

With the recent success in Deep Learning, now the focus is slowly shifting to applying deep learning to solve reinforcement learning problems. The news recently has been flooded with the defeat of Lee Sedol by a deep reinforcement learning algorithm developed by Google DeepMind. Similar breakthroughs are being seen in video games, where the algorithms developed are achieving human-level accuracy and beyond. Research is still at par, with both industrial and academic masterminds working together to accomplish the goal of building better self-learning robots

Source

Some major domains where RL has been applied are as follows:

- Game Theory and Multi-Agent Interaction
- Robotics
- Computer Networking
- Vehicular Navigation
- Medicine and
- Industrial Logistic.

There are so many things unexplored and with the current craze of deep learning applied to reinforcement learning, there certainly are breakthroughs incoming!

Here is one of the recent news:

> Excited to share an update on #AlphaGo!
>
> — Demis Hassabis (@demishassabis) 8:30 PM - 4 Jan 2017

7. Additional Resources

I hope now you have in-depth understanding of how reinforcement learning works. Here are some additional resources to help you explore more about reinforcement learning

- Videos on Reinforcement Learning
- Book on Introduction to Reinforcement Learning
- Awesome Reinforcement Learning Github repo
- Course on Reinforcement Learning by David Silver

End notes

I hope you liked reading this article. If you have any doubts or questions, feel free to post them below. If you have worked with Reinforcement Learning before then share your experience below. Through this article I wanted to provide you an overview of reinforcement learning with its practical implementation. Hope you make found it useful.

Learn, compete, hack and get hired