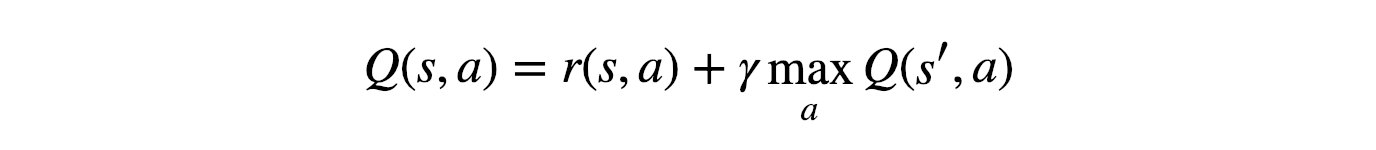
**Q Learning**

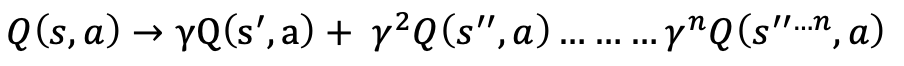
Let’s say we know the expected reward of each action at every step. This would essentially be like a cheat sheet for the agent! Our agent will know exactly which action to perform.

It will perform the sequence of actions that will eventually generate the maximum total reward. This total reward is also called the Q-value and we will formalise our strategy as:



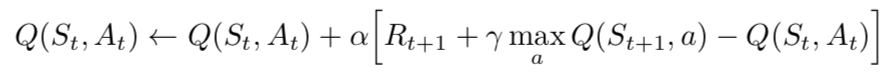
The above equation states that the Q-value yielded from being at state *s* and performing action *a* is the immediate reward r(s,a) plus the highest Q-value possible from the next state *s’*. Gamma here is the discount factor which controls the contribution of rewards further in the future.

Q(s’,a) again depends on Q(s”,a) which will then have a coefficient of gamma squared. So, the Q-value depends on Q-values of future states as shown here:



Adjusting the value of gamma will diminish or increase the contribution of future rewards.

Since this is a recursive equation, we can start with making arbitrary assumptions for all q-values. With experience, it will converge to the optimal policy. In practical situations, this is implemented as an update:



where alpha is the learning rate or step size. This simply determines to what extent newly acquired information overrides old information.

**Why ‘Deep’ Q-Learning?**

Q-learning is a simple yet quite powerful algorithm to create a cheat sheet for our agent. This helps the agent figure out exactly which action to perform.

But what if this cheatsheet is too long? Imagine an environment with 10,000 states and 1,000 actions per state. This would create a table of 10 million cells. Things will quickly get out of control!

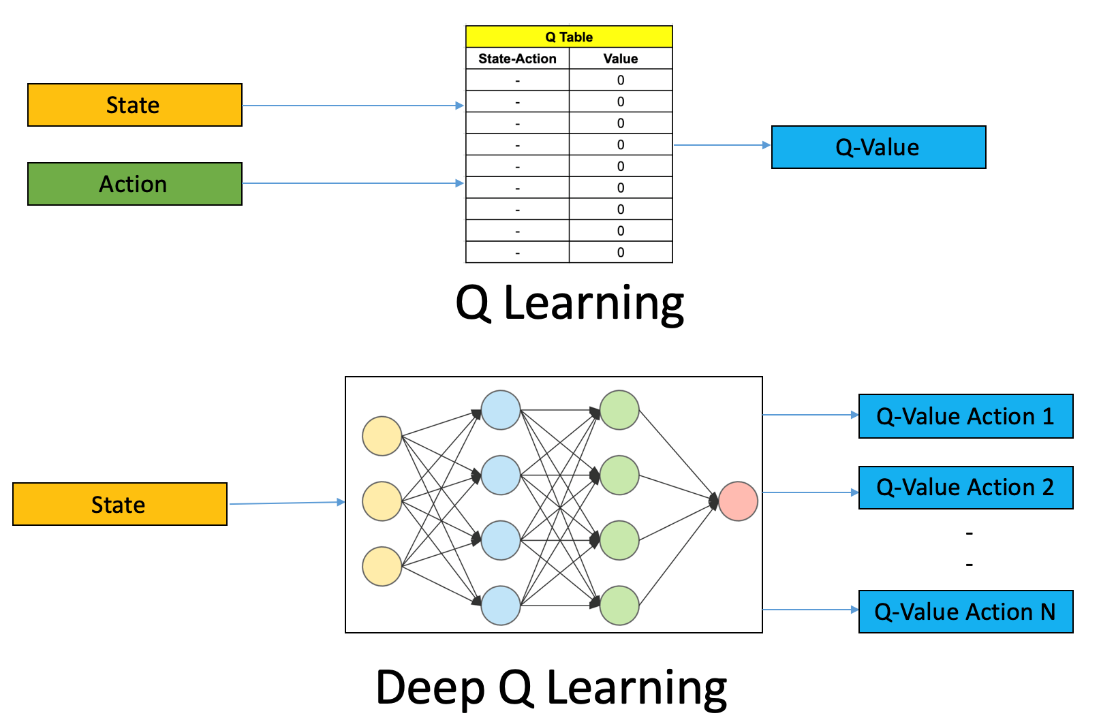
It is pretty clear that we can’t infer the Q-value of new states from already explored states. This presents two problems:

* First, the amount of memory required to save and update that table would increase as the number of states increases
* Second, the amount of time required to explore each state to create the required Q-table would be unrealistic

Here’s a thought – what if we approximate these Q-values with machine learning models such as a neural network? Well, this was the idea behind DeepMind’s algorithm that led to its acquisition by Google for 500 million dollars!

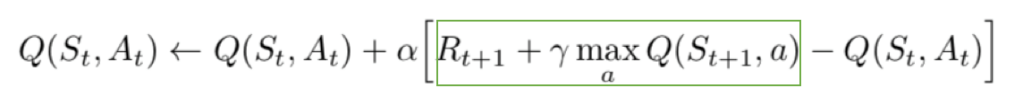
**Deep Q-Networks**

In deep Q-learning, we use a neural network to approximate the Q-value function. The state is given as the input and the Q-value of all possible actions is generated as the output. The comparison between Q-learning & deep Q-learning is wonderfully illustrated below:

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2019/04/Screenshot-2019-04-16-at-5.46.01-PM.png)

So, what are the steps involved in reinforcement learning using deep Q-learning networks (DQNs)?

1. All the past experience is stored by the user in memory
2. The next action is determined by the maximum output of the Q-network
3. The loss function here is mean squared error of the predicted Q-value and the target Q-value – Q\*. This is basically a regression problem. However, we do not know the target or actual value here as we are dealing with a reinforcement learning problem. Going back to the Q-value update equation derived fromthe Bellman equation. we have:



The section in green represents the target. We can argue that it is predicting its own value, but since R is the unbiased true reward, the network is going to update its gradient using backpropagation to finally converge.