Neural Networks Applying Deep Q-Learning

Gaizka Calvo and Gentzane Pastor

1 Introduction

Nowadays, machine learning is getting a huge impact in the video games industry. Many games use these techniques in order to achieve realistic AI behaviors. In this article we will explain some algorithms we have used in order to make a simple AI learn how to play the traditional snake game.

The first algorithm we are going to explain is Neural Network, which is a technique used for learning some complex patterns such as face recognition or hand writings. Later, we will explain Q learning as well as deep Q learning. This last one is used for learning by interacting with the environment.

2 Neural Network

Neural Network is a machine learning technique inspired by the biological Neural Network. This technique can be used in different areas; Supervised Learning, where here the AI learns about the task by giving him inputs and the desired outputs, Unsupervised Learning, here the AI learns about the task by interacting with the environment and learning from it, and finally Deep Reinforcement Learning, that is a combination of the previous ones.

In our case we choose a Deep Reinforcement Learning called Deep Q-learning, that combines the Neural Network and Q-learning algorithms. As we previously mentioned, Neural Networks is inspired by a biological Neural Network, so first, we will take a look into Neurons, which is, what it is composed of.

Neurons is the basic unit of a Network, and it works as simply as getting some inputs and giving some outputs. All this process is done through a function called activation function.

Activation function: $f(x_1, x_2) = f(w_1 * x_1 + w_2 * x_2 + b)$

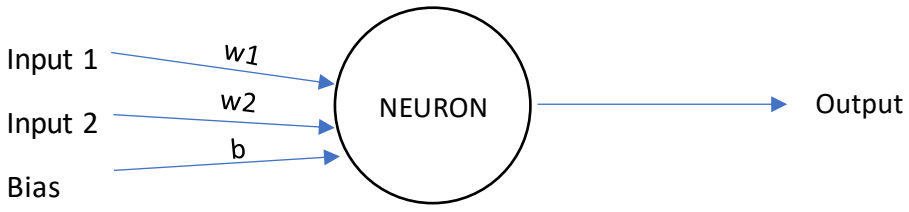

Figure 1 Representation of a Neuron.

As we can see in the Figure 1, a Neuron is composed by connections to other Neurons that give to each other their respective outputs and taking them as inputs. The connections between Neurons have a value called weight, that is used to multiply and change the value of the given input to obtain the expected output. Finally, we have a last input that is called the Bias, this one is used to shift the activation function a constant amount.

There are 3 common activation function used in Neural Network: Sigmoid, Tanh and ReLU. For our project, we used the Sigmoid activation function:

Sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$

Input range: (-∞, ∞)

Output range: (0,1)

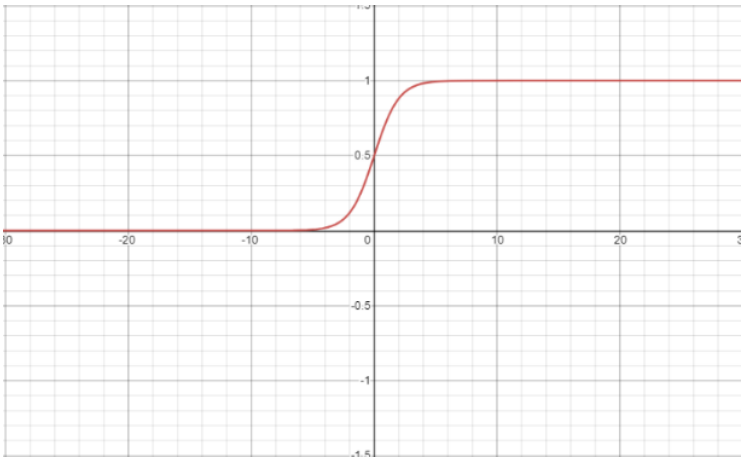

Figure 2 Graph of the Sigmoid function.

As we previously said, Neural Networks are composed by Neurons, and each of those neurons form different type of layers with different functions. The type of Network we are using for this assignment is a multi-layer perceptron which adds a hidden layer between the input and output layers, as we can see in the following Figure 3.

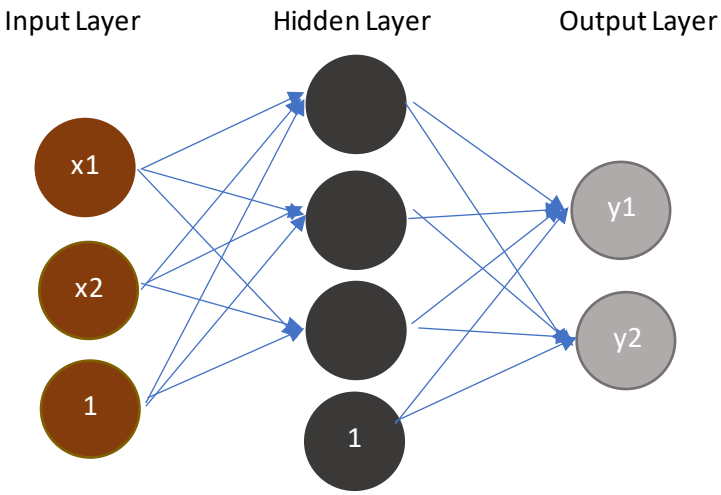

Figure 3 Multi-Layer Perceptron network.

The input layer is used to get the inputs we need, in the same way, the output layer represents the output values of the networks and finally the hidden layer which is used to add more complexity to the Neural Network.

Knowing all these, now we will explain how the neural network gives outputs and learns from those. The process to obtain the output of the Neural Network is called Forward propagation. Here, each layer calculates their outputs, using the activation function with the given inputs. The last layer will represent the outputs of the neural network.

In the forward propagation, the Neural Network tries to predict the output we expect, but it is not learning yet. The process of learning is called backward propagation. Here, given real outputs, the Neural Network will correct the weights to approach more to the wanted results.

3 Q-Learning

Q-Learning is a Reinforcement Learning Algorithm, which means that this algorithm can learn patterns without the requirement of giving it outputs. Q-Learning uses a mathematical framework for modeling decision making that comes from the Markov Decision Process (MDP).

In MDP, we have an agent with different states and inside each state we will encounter multiple actions. Here, the agent will perform an action and will change to another state. At the same time, the environment where the agent is, will also change and here we will reward the agent in a positive or a negative way depending on if the action of the AI is the desired one or not. With this the agent will learn what it has done correct or incorrect.

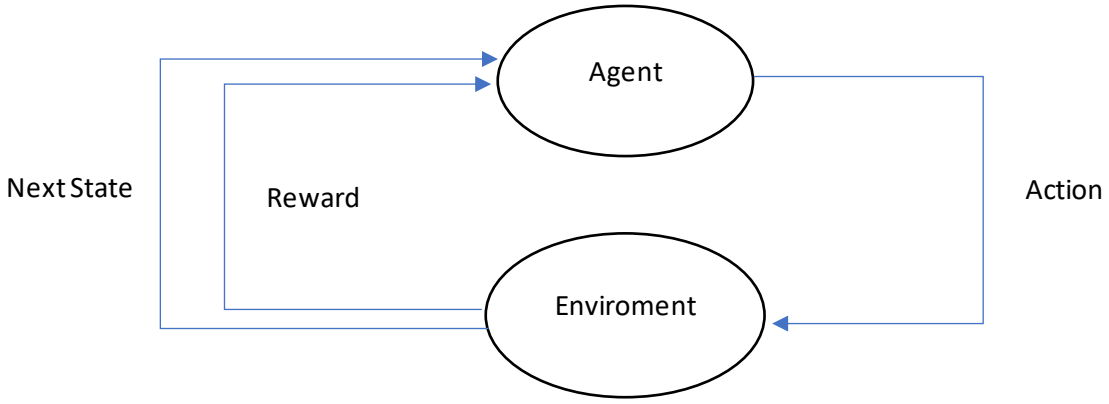

Figure 4 MDP framework.

In the case of our environment, we will reward our agent, the snake, for each movement that gets closer to the apple, and we will punish it when it gets further. As the same way, we will reward the snake if it gets an apple, and we will punish it if he dies. With this stated, the snake will learn to go towards the apple.

For all this to work, we will also need a decision process called the policy function π(s). We will need to find a good policy for our snake to learn properly. The policy is the function that taking a state, will choose an action that will give the AI the best reward.

For a policy function we have the Q-learning algorithm. This algorithm chooses the best action in each state considering future rewards. This is done by choosing the highest value that one of the possible actions will have. As we can see in the table 1, we will choose our highest value in what is called the Q table, which is composed by Q values.

Table 1 Q-table.

States	Q (S, Move up)	Q (S, Move down)
S1	0	0
S2	0	0
S3	0	0

This Q-table values will be initialized to 0 and later for each action that it does it will learn from it and update the Q-values. We will compute our Q-values with the Bellman's equation: $Q(s,a) = R(s_{next}) + \gamma * max_{a_{next}} * Q(s_{next}, a_{next})$.

Knowing all this, now we will explain how the algorithms learn. The AI will have a phase where it will perform random actions to learn from those, this process is called exploration. By the time goes on the algorithm will perform less random actions and will choose the best action possible. This process is called exploitation. This behavior can be gathered using Decay Epsilon Greedy (shown in figure 5), where we will have a value between 0 and 1 that will decrease over time and each time that we need to choose an action, considering a random value between 0 and 1 we will explore or exploit.

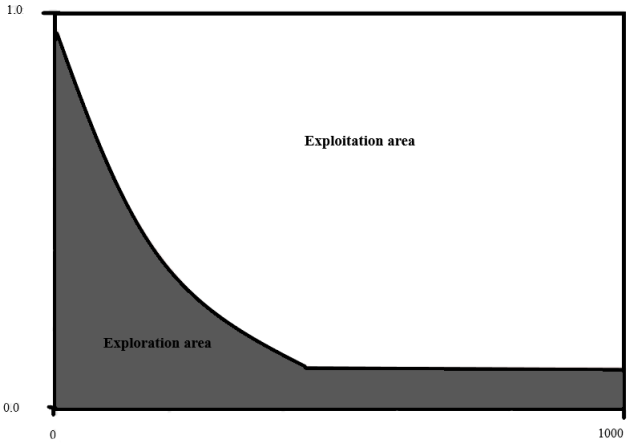

Figure 5 Epsilon greedy method.

4 Deep Q-Learning

Some AI may have too complex and too many states to be able for us to use Q-Learning and here is where Deep Q-Networks come us handy. Here we will use our Neural Network to estimate the Q-function, which means that we will replace our Q-table for our Neural Network. Our Neural Network will take as inputs, the different states of our AI and will return the Q-values of each action. You can see the difference of these two methods clearly in the Figure 6.

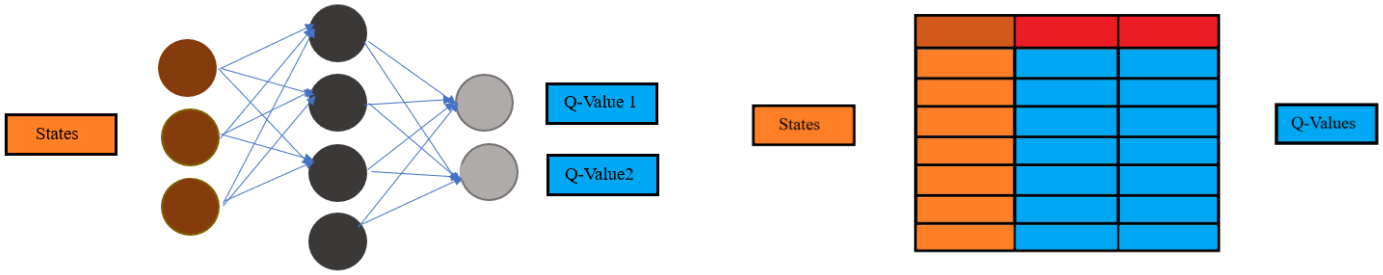

Figure 6 Difference between Deep Q and Q Learning.

Deep reinforcement learning combines what we have seen that is Deep Learning using Neural Networks and Reinforcement Learning which is Q-Learning. In this algorithm, the Neural Network, corrects the error that generates using the following cost/error function.

$$E = \left(\left(R(S_{next}) + \gamma * max_{a_{next}} * Q_{network\ target}(s_{next}, a_{next})\right) - Q_{network\ main}(s, a)\right)^2$$

As you can see, both sides of the equations are similar to Bellman's equation. As you could also notice, in this algorithm we will use to Neural Network, a target Network and a main Network. We will use the main network to compute the Q-value of the next state and the target network to calculate the error in the main network. With each action we will train the main network and the target network will remain the same, but after some iterations, all the weights of the main network will be copied to the target network.

Finally, we also implemented in deep neural network the experience replay, which is a buffer of past experiences that we use to refresh what has the Neural Network learned. Each time the AI makes an action, we will save some information and train the neural network with previous actions saved into the buffer. Let us put an example of our game for a better understanding. Imagine that the first problem that our snake

encounter is avoiding walls, the snake will learn to avoid those walls and it will stay in the middle area. In that area, it may learn that eating apples is the way to go and when it encounters a wall again, he may have ended up forgetting what he learned about that past situation and dies again with the walls. What we do in the experience replay, is saving some information when the AI encounters a wall and when later on it encounters a wall again, it will know what to do because of that past experience.

6 Conclusion

In this article, we have introduced you a technique of machine learning that it may be too complex to fully understand it with only the given information. We encourage you to check out a bit more about Neural Network, Markov Decision Process, Q Learning and finally after getting a good knowledge in these, Deep Q Learning.

This deep reinforcement learning algorithm can come handily to solve that problem where the reinforcement learning gives us a lot of trouble. When implementing this algorithm, it needs to be taken into account the number of parameters that influence the learning of the AI and that the expected behavior can be achieved with constant iterations and tweaking of those.

7 References

[DigiPen Institute of Technology Bilbao] CS397 Machine Learning Slides
Neural Network, Neural Network Types, Reinforcement Learning and Deep Q-Learning.

[Wikipedia] Deep reinforcement learning
https://en.wikipedia.org/wiki/Deep_reinforcement_learning

[Wikipedia] Markov decision process
https://en.wikipedia.org/wiki/Markov_decision_process