# Dots and Boxes Using Q-Learning

Shivang Patel

## Assignment Overview:

In this assignment, a reinforcement learning approach was used to tackle the game “Dots and Boxes”. Reinforcement learning within the machine learning domain is a process by which an agent learns to act by interacting with its environment. It does this by experiencing “rewards” for taking given actions. A useful analogy, albeit a naive one, could be the process training a dog. If a dog does something correct, you reward that dog with a treat. If that dog does something incorrect, it may receive a scolding. To translate that metaphor to the problem of game playing; when the agent performs actions that result in a ‘win’, it receives a reward, and actions that result in a loss are punished with a negative reward. Ultimately, the program should seek to maximize its reward by picking the best actions in the current state of the environment.

Specifically, Q-learning model was to be used to tackle this problem. Also, need to solve using both the method, Q-table and Q-nets (functional approximation). Later, performance of both the method is been compared.

## Game Description

The game “Dots and Boxes” is played on a game board as shown in figure 1. The players take turns drawing lines between two adjacent dots. When a player completes a box by drawing the fourth wall, they get a point and get to draw again. The game continues until all lines are filled, and the winner is the player who has the most points at end-game.

## Q-Learning

Q-Learning is a popular method of determining the value of state-action pairs in reinforcement learning. The Q-Value, represented by Q(S, a) where S is the state and a is the action, is derived from the reward received for taking action a, plus the discounted sum of all future rewards. The generic formula for updating a Q-Value can be represented by the equation in figure

The ‘learned value’ in the formula above can be thought of as the target. It is comprised of the reward (the value of the state-action pair), the discount factor (discussed in 2.4.6), and the estimate of optimal future value. This means the value of choosing the right action, according to the current policy, at every state going forward. This ‘target’ is essentially what the Q-value should be as dictated by the reward from the environment and beliefs about the future. The learning rate adjusts the current Q-Value so that it moves towards the target value by some

amount. This parameter is between 0 and 1. In this assignment, the reward will be received via winning (reward = 5) or losing (reward = 0) a game in the environment.

## Methodology (Q-Table)

## Methodology (Q-Learning)

Very little preprocessing is needed for this method as the states are in the form of array and we can directly feed it to neural network.

Convolution Model:

The architecture of the convolutional network is described as follows (all activations are elu (exponential linear unit) unless otherwise specified).

INPUTS -> Layer 1 -> Layer 2 -> Layer 3 -> Layer 4 -> Output

Layer 1: Contains 3x3 convolution and 1x1 convolution with 16 filters (concatenated)
Layer 2: 3x3 convolution and 1x1 convolution with 32 filters (concatenated)
Layer 3: Fully Connected Layer with 256 Neurons
Layer 4: Fully Connected Layer with 256 Neurons
Output: Fully Connected layer with 40 neurons and tanh activation

Rather than a Q-Function that takes a state-action pair and returns a value, this function takes a state and returns values for every state-action pair. This allows for far more efficiency, as this way only one forward pass is needed to find all Q-Values, rather than a pass for every possible action. The ELU activation was chosen to help avoid dead neurons, which frequently occurred while using RELU activations.

Training Procedure:

The training process for the learning agent was fairly straightforward. Two agents with a DQN ‘mind’ were initialized, which will be referred to as the ‘learning agent’ and the ‘target agent’. The target agent did not learn while it played the games. These agents played games against each other, and the learning agent would update their Q-Function as the games went on. At some interval, the learning agent would save it’s Q-Function and that function would be loaded into the target agent. This allows the ‘opponent’ to improve as the agent improves.

100, 1000 and 10,000 games were played against the target agent from start to finish.
The interval at which the target agent was initialized at 10 for 100 and 100 for remaining games, however this did not remain constant.

Testing Procedure:

After 10 or 100 of games of training against an opponent, the learning agent was then tested against the random agent benchmark agent. The learning agent was set so that it wouldn't update it's Q-Function or record information for the replay table during these games. One thousand games were played against each agent, and the metrics were recorded in a log file. Half of these games were played as the first player, and the other half as second player, to ensure that the tests were robust.

## Results