

Deep Q-Learning for Tower Defense

With a huge state space regular Q-learning can't learn or explore all combinations of state-action pairs. Deep Q-Learning can be useful here as instead of a big lookup table, it uses a neural network to approximate the Q-function. The network takes a state as input and outputs Q-values for all possible actions.

- Q-learning
 - look up the state in a table → get Q-value
- Deep Q-learning
 - feed state into neural network → get Q-values for all actions at once

It helps us solve problems which are learning a huge state space by being able to learn from patterns rather than specific positions. Similar states (enemy at 13 HP vs 12 HP) produce similar Q-values because the network learns patterns, not individual values

How can we implement it?

For our 5×5 grid, we can use a Convolutional Neural Network (CNN) because tower placement strategy involves spatial patterns. A tower at (2,2) vs (2,3) has similar strategic value. CNNs naturally learn these spatial relationships.

Experience replay is important here. Instead of learning from experiences in the order they happen, we store all experiences (state, action, reward) in a replay buffer, randomly sample a mini-batch from this buffer when training, and train the network on this random batch. This breaks up correlations between consecutive experiences which would otherwise mess up training.

We can use 2 networks. Policy Network updated every step, used to select actions and Target Network updated only every N steps by copying policy network, used only to compute target Q-values.

3 versions of deep Q-learning we can try as well and compare with

1. vanilla deep q-learning
 - the basic version with experience replay and target networks. We can start with this.
2. double deep q-learning
 - Vanilla DQN tends to overestimate Q-values because of noise in the Q-value estimates, max tends to pick actions whose values are overestimated by random chance. Over time, this leads to systematic overestimation.
 - the fix is to use the main network to select the action, but the target network to evaluate it
 - It's supposed to be 2-3 lines of code different from vanilla DQN and give ` better performance.
3. dueling deep q-learning

- it splits the Q-value into $V(s)$: "How good is this state in general?" (state value) and $A(s, a)$: "How much better is action a compared to average?" (advantage)

Things to do

- normalize all health values to $[0, 1]$ range. Neural networks train much better when inputs are in the range $[0, 1]$ rather than $[0, 56]$ for HP values.
- Deep Q-learning class
 - Methods to include
 - Network
 - replay buffer
 - training
- set up the training loop that stores experiences and trains on batches