# 2.3.2 Problem description

As described in the paper, using an infinite space for the elements in the state vector, it is impossible to determine the near-optimal policy, when using the Q learning approach. This is because it would need infinite search power, as the Q table would grow infinitely. Thus, the paper proposes to discretize the state space to some coded values. However, in the context of the beer game, relying discretized states can be restrictive, as this would mean information loss. It would in consequence be better to represent the state variable, ranging from negative infinity to positive infinity.

Using DQN permits us to handle these scenarios where the number of states aligns more realistically with real-world situations. In our specific case this implies the adoption of a continuous state space for the actors. Instead of the building of a Q table, we would need to find a function to approximate the Q values. This approximation can be achieved through the use of a neural network, that predicts a Q value based on a current state.

To understand better what needs to be changed on the current implementation of the inventory optimization, we will explain how the DQN works.

To begin with, a replay buffer interacts with the environment. In these interactions, it collects data by following an ε-greedy policy for specific state-action pairs. This data collection process is focused on obtaining both the reward and the state that follows a particular action. Once acquired, this data is used as training data, for later use. From training data amassed, a sample is made to create a subset. This subset, becomes the basis for training of a neural network, called the Q neural network. This network is used to predict the Q value of the current state. In parallel, a second neural network, the Target network, predicts the Q value of the next state, that is linked to the action undertaken after the current state. With the neural network architecture in place, the loss is computed in the next step. This metric measures the discrepancy between the predicted Q value from the Q neural network and the target Q value. The target Q value is computed by adding the reward associated with the current action and the next states Q value. This discrepancy could for example be calculated through the Mean Squared Error (MSE). This loss is then used for backpropagation, a process to adjust the neural networks weights with the goal to minimize the loss between the current Q value and the target Q value. Backpropagation works by plotting the gradients of the loss, with respect to the network weights, which in turn indicates how much the weights have to be adjusted to reduce the loss. The extent of the update is controlled by the learning rate α, ensuring a balanced convergence toward more accurate Q value predictions. However, the update of the weights is only carried out for the Q NN and not the target NN, to ensure stability in the learning process. By keeping the target NN's weights fixed for a set number of iterations, we provide a more stable and consistent target for the Q NN to learn from. Although the update of the weights of the target NN happen periodically, by matching the weights to the weights of the Q NN. This whole process is repeated for each time step until a specified maximum number of iterations is reached.

Following this logic, we can now understand what needs to be changed in our implementation to apply the DQN. As already mentioned, the state space doesn’t have to be encoded anymore and we can just assume the infinite state space. Thus, the state can be represented as a vector of continuous values [S1, S2, S3, S4]. This is also recommended as it wouldn’t make sense to use DQN if the state and action space stays the same, as the training of the NN takes more resources, when the state and action space is small.

For the action space, it has to remain discretized. This is because the neural network needs a defined number of output neurons, each corresponding to a specific action. Nevertheless, DQN allows for a broader action space without employing too many resources, where in contrast, the increase of actions would increase the number of operations in a Q table exponentially.

Most importantly, we wouldn’t need a Q table anymore. We could just replace it with the neural network that reliably predicts the Q values. To train that NN, we would have to implement a new algorithm that updates the weights for the NN iteratively, to get more realistic Q values.

We would also need to be implemented is that of replay buffer. In this, the state action combination, paired with their reward and next state is stored for each step. This could be stored as a list with multiple “experiences” stored in them. As we have multiple actors in the beer game, we need to consider that each experience stored should capture the combined actions and states of all actors. Thus, each experience should be represented by a vector of states, action, rewards and next states. Additionally, the replay buffer should also include information about if the next state is terminal or not, as if the next state is terminal, there is no future reward to consider, and this would signify the end of an episode (35 weeks) and a new one can start. From this list we should be able to sample a batch of those experiences.

On top of the hyperparameters that we have in Q learning, we will get some more parameters to tune, such as the size of the replay buffer, the batch size of the subset for training the NN, the update frequency for the target network and the hyperparameters of the neural networks, including the learning rate for the backpropagation.

Although some of the elements wouldn’t need to be changed. Exploration/exploitation stays the same. We still use an ε-greedy strategy with ε decreasing over time, leading to exploitation in contrast to exploration.