# Q-Learning with Neural Networks for Robotic Arm Control

Pablo Sánchez Orozco, Alberto Arath Figueroa Salomon
Nicolás Every Alvarez

March 7, 2025

**Abstract**

This report explores the use of Q-learning with neural networks (NN) for training a robotic arm model in a physics-accurate simulation. The objective is to demonstrate the advantages of using NN-based Q-learning over traditional Q-tables, particularly in high-dimensional robotic control problems. The Mujoco-based model and the Menagie physics engine were employed for accurate physics simulation.

## 1 Introduction

The intention of this project was to train a robotic arm using Q-learning with neural networks (NN) instead of a conventional Q-table. The complexity of the robot's state-action space necessitates a function approximator, which NN provides efficiently.

## 2 Methodology

The robotic arm was simulated using Mujoco, with Menagie serving as the physics engine. The model consists of multiple articulated joints, each actuated by motors. The XML model features:

- Six degrees of freedom: Base rotation, shoulder pitch, elbow movement, wrist pitch and roll, and jaw control.
- Defined friction loss and armature parameters for realistic motion.
- Motor constraints with position-based control.
- Contact modeling to simulate object interactions.

The Q-learning algorithm with NN was used to approximate the Q-value function, enabling generalization over large state-action spaces.

## 3 Mathematical Formulation

The Q-learning update rule is given by:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[ r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right] \tag{1}$$

where:

- $Q(s,a)$ is the Q-value for state $s$ and action $a$.
- $\alpha$ is the learning rate.
- $\gamma$ is the discount factor.
- $r$ is the received reward.
- $s'$ is the next state.

Instead of storing $Q(s,a)$ in a table, we approximate it using a neural network:

$$Q(s,a;\theta) \approx f(s,a;\theta) \tag{2}$$

where $\theta$ represents the NN parameters trained via backpropagation.

## 4 Neural Network Design

The neural network used for Q-learning is designed as follows:

- **Input**: The state vector containing the current angles of all joints.
- **Output**: Q-values corresponding to the two possible actions (increase or decrease angle) for each joint.
- **Size**: If the robotic arm has $n$ joints, the output layer consists of $2n$ neurons.

The reward mechanism is defined based on the Euclidean distance between the tip of the claw and a target point in space. The objective is to minimize this distance, guiding the robotic arm towards the target.

## 5 Why Use NN Instead of Q-table?

A traditional Q-table is infeasible for a high-dimensional state space as it requires an entry for each state-action pair. Given the continuous action space of the robotic arm, NN provides a better alternative by enabling function approximation and generalization, reducing memory requirements and training time.

## 6 Results and Discussion

While the implementation of Q-learning with neural networks showed potential, the model requires further refinement. One of the primary issues was the selection mechanism using the $\arg\max$ function. Instead of selecting the best Q-value for each joint separately, it was only choosing a single action across the entire output vector. Since each joint has two possible actions, the correct approach should involve selecting the best index within each joint's respective Q-value pair. This misalignment caused suboptimal policy execution, leading to ineffective control of the robotic arm. Future improvements should focus on correctly implementing $\arg\max$ across joint-specific action pairs to ensure proper action selection per joint. Additionally, hyperparameter tuning using a grid search approach should be explored to optimize learning efficiency and convergence.

## 7 References

Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine Learning*, **8**(3-4), 279-292.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, **518**(7540), 529-533.

Mujoco Physics Engine: `https://mujoco.org/`

Mendeliere Repository: `https://github.com/mendeliere`