Human-Level Control with Deep Q-Networks (DQN)

A Guide for Bioengineers

Yange Sun

(CID: 02059079)

January 11, 2026

Note:

This tutorial aims to explain DQN in an accessible way for bioengineers with a basic background in Reinforcement Learning and Bioengineering. For beginners or enthusiasts seeking a more accessible introduction, please refer to this link: https://medium.com/@1933476828/human-level-control-with-deep-q-networks-dqn-b9461a143ebb?postPublishedType=repub

Abstract

High-dimensional state spaces present significant challenges for traditional reinforcement learning algorithms, often rendering them computationally intractable. Deep Q-Networks (DQN) revolutionized the field by integrating deep convolutional neural networks for robust function approximation from raw inputs. This study offers a systematic tutorial on the architecture and implementation of DQN. We elucidate core concepts, including Experience Replay and Target Networks, and present a functional implementation utilizing 1D Convolutional Neural Networks for sEMG-based gesture recognition. The efficacy of the proposed method is validated through experimental analysis, providing clear visualization of the learning progress.

*I acknowledge the use of **Gemini 3 pro** (Google, https://gemini.google.com/app) to help me with language polishing and coding (implement the algorithm I designed myself). I confirm that no content generated by AI has been presented as my own work.*

1 Introduction: Why DQN Matters?

As detailed by Sutton and Barto [1], Reinforcement Learning (RL) relies on continuous agent-environment interaction to learn optimal behaviors. Among foundational algorithms, **Q-learning** is the most prominent. As an off-policy Temporal-Difference (TD) control algorithm, it is widely adopted for its ability to guarantee convergence in finite environments by learning independently of the agent's exploratory actions.

However, traditional tabular methods fail in high-dimensional environments (e.g., images or biological signals) due to the "Curse of Dimensionality" [3]. The fundamental challenge here is **generalization**: agents must estimate values for states never encountered before. Consequently, lookup tables must be replaced by **function approximation** to infer values from limited experience [1].

In 2015, Mnih et al. introduced the **Deep Q-Network (DQN)**, shattering this barrier [2]. By using Convolutional Neural Networks to approximate Q-values directly from raw pixels, DQN achieved **human-level control** in Atari games without handcrafted features. Although later research identified limitations such as value overestimation (addressed by Double DQN [4]), this tutorial focuses on the foundational DQN architecture.

For bioengineers, particularly in the domain of myoelectric control, DQN is transformative. **Surface Electromyography (sEMG)** signals are inherently high-dimensional, stochastic, and non-stationary, making precise gesture decoding a complex challenge. DQN offers a robust framework to learn optimal classification policies directly from these noisy, multi-channel time-series data. By bypassing the limitations of manual feature engineering, this approach opens new frontiers in adaptive prosthetic control and intuitive Human-Machine Interaction (HMI).

2 Theoretical Framework: From Q-Learning to DQN

To understand Deep Q-Networks, we must first revisit the fundamental algorithm it is built upon: **Q-learning**, and then address the instability issues that arise when combining it with deep neural networks.

2.1 The Foundation: Q-Learning and the Bellman Equation

The goal of the agent is to learn the optimal action-value function, $Q^*(s,a)$, which represents the maximum expected future reward achievable starting from state s and taking action a. This theoretical optimum satisfies the **Bellman Optimality Equation**:

$$Q^*(s,a) = E\left[r + \gamma \max_{a'} Q^*(s',a')\right] \tag{1}$$

To estimate this, traditional Q-learning iteratively updates the values in a table using the **update rule**:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right] \tag{2}$$

where α is the learning rate and γ is the discount factor.

2.2 Bridging the Gap: Function Approximation

Before introducing DQN, it is crucial to understand the mathematical transition from tables to networks.

2.2.1 The Problem with Tables

In standard Q-learning, the agent uses a **Q-table** where every state s corresponds to a specific row. However, if an agent encounters a state that is not in the table, it cannot make a decision. This lack of **generalization** is a major bottleneck for complex environments with vast state spaces.

2.2.2 The Solution: Function Approximator

To solve this, we replace the discrete table with a continuous function f, parameterized by a vector of weights θ. Instead of looking up a value, the agent *calculates* it:

$$Q(s, a) \approx f(s, a; \theta) \tag{3}$$

This function is called a **Function Approximator**.

- In simple cases, this could be a linear equation: $f(s) = \theta^T s$.
- In complex cases (like Atari games), we need a non-linear function capable of understanding patterns. This is where **Neural Networks** come in.

2.2.3 Convolutional Neural Networks (CNN)

A Convolutional Neural Network (CNN) is a specialized deep learning architecture designed to process grid-like data (such as images) by using learnable filters to automatically extract spatial features directly from raw pixels. DQN utilizes a **CNN** with weights θ as its function approximator to estimate the optimal Q-values:

$$Q(s, a; \theta) \approx Q^*(s, a) \tag{4}$$

The network takes raw pixel data (state s) as input and outputs a vector of Q-values, one for each possible action.

2.3 Key Mechanisms for Stability

Naively applying Q-learning to neural networks is unstable due to correlated data and non-stationary targets. Mnih et al. [2] introduced two novel mechanisms to stabilize training:

2.3.1 Experience Replay

In standard RL, observations occur sequentially (s_t, a_t, r_t, s_{t+1}), creating strong temporal correlations that can cause the network to overfit and diverge.

Solution: DQN stores transitions in a cyclic buffer called **Experience Replay** ($\mathcal{D}$). During training, the agent samples a random *mini-batch* of transitions from $\mathcal{D}$. This breaks the temporal correlation between samples and allows the agent to learn from past experiences multiple times, significantly improving data efficiency.

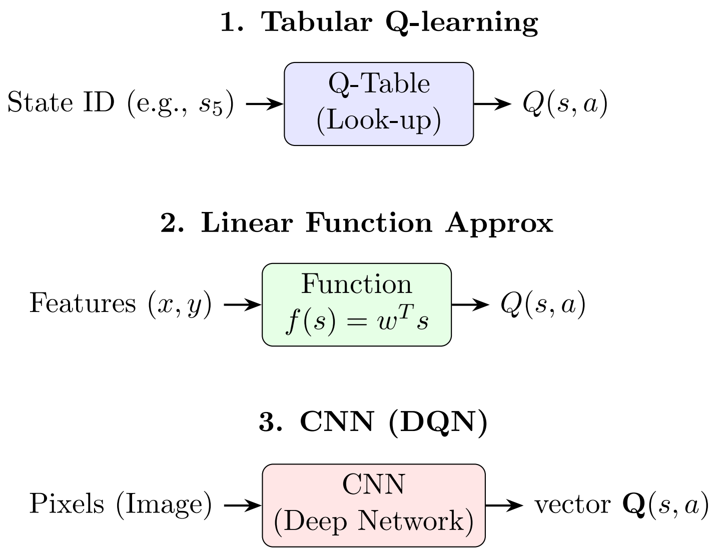


Figure 1: Evolution of Q-value estimation: (1) Direct table lookup, (2) Simple linear function approximation, and (3) Deep Convolutional Neural Network (DQN) as a powerful function approximator.

2.3.2 Fixed Target Network

The standard Q-learning update uses the same network parameters θ to calculate both the predicted value and the target value. This leads to a "chasing a moving target" problem, causing oscillations.

Solution: DQN employs two separate networks:

- **Policy Network (θ):** Selects actions and is updated at every step.
- **Target Network (θ^-):** Used solely to generate the target Q-values. Its weights are periodic copies of the Policy Network (e.g., $\theta^- \leftarrow \theta$ every C steps) but remain frozen in between.

2.4 The Loss Function

Combining these components, the DQN agent is trained by minimizing the loss function $L(\theta)$ via Stochastic Gradient Descent. The loss is defined as the squared difference between the target Q-value and the predicted Q-value:

$$L(\theta) = E_{(s,a,r,s')\sim\mathcal{D}}\left[\left(\underbrace{r + \gamma \max_{a'} Q(s', a'; \theta^-)}_{\text{TD Target (using frozen } \theta^-)} - \underbrace{Q(s,a;\theta)}_{\text{Prediction}}\right)^2\right] \tag{5}$$

By minimizing this error, the neural network gradually learns to approximate the optimal action-value function, allowing the agent to perform human-level control.

3 Code Implementation

3.1 Background and Motivation

Surface Electromyography (sEMG) is a non-invasive technique that records the electrical activity of skeletal muscles, serving as a critical interface for prosthetic control and Human-Computer Interaction (HCI). Despite its utility, sEMG signals are inherently complex, characterized by high dimensionality, stochasticity, and non-stationarity. Traditional pattern recognition methods often rely on manual feature extraction (e.g., Root Mean Square), which requires domain expertise and may fail to capture latent temporal dependencies in the data. Consequently, there is a growing demand for data-driven approaches capable of learning robust representations directly from raw, noisy signals without manual intervention.

3.2 Objectives and Proposed Framework

To address these challenges, this project implements a **Deep Reinforcement Learning (DRL)** framework for sEMG-based gesture recognition. The primary objective is to validate the efficacy of an end-to-end learning agent that maps raw signal inputs directly to gesture classes.

The proposed system utilizes a **Deep Q-Network (DQN)**, which reformulates the classification task as a sequential decision-making problem. The core innovation lies in the integration of two key components:

- **1D Convolutional Neural Network (1D CNN):** Serves as the function approximator to automatically extract local temporal features from high-dimensional sEMG time windows.
- **Environment Wrapper:** Transforms the static dataset into an interactive environment, where the agent receives rewards for correct classifications, enabling learning through trial-and-error.

By combining the feature extraction power of CNNs with the decision-making capability of Q-learning, this code aims to demonstrate that RL agents can effectively master complex bio-signal patterns, offering a robust alternative to traditional supervised learning paradigms. **Note:** For the complete code, includes all scripts for generating visualizations and the original data, please click this link, where you can find detailed comments for each line of codes: `https://github.com/1933476828-ship-it/Gesture-Prediction-Using-DQN-Based-on-sEMG-Signals.git`. Later in this article, I will summarize my algorithm using pseudocode.

To validate the efficacy of this approach, the experimental simulation is driven by data from the **Ninapro DB5** database [5], specifically focusing on **Subject 01**. Using this benchmark dataset allows us to evaluate the DQN agent's performance in a realistic environment characterized by high-dimensional and non-stationary sEMG signals. The framework and structure of this data will be presented in Chapter 4, Results and Visualization.

3.3 Methodology

3.3.1 Replay Buffer

To mitigate temporal correlations in sequential sEMG data, we utilize a Replay Buffer $\mathcal{B}$ with capacity N. It stores transitions (s, a, r, s', d) using a FIFO policy and supports uniform random batch sampling. This mechanism stabilizes training by breaking serial correlations, ensuring the data distribution remains approximately independent and identically distributed (i.i.d.).

Algorithm 1 Replay Buffer Logic

```
1: **Initialize:** Buffer $\mathcal{B}$ with capacity $N$
2: **procedure** PUSH($s, a, r, s', d$)
3:     **if** $|\mathcal{B}| \geq N$ **then**
4:         Remove oldest transition (FIFO)
5:     **end if**
6:     Add $(s, a, r, s', d)$ to $\mathcal{B}$
7: **end procedure**
8: **procedure** SAMPLE(BatchSize $K$)
9:     Select $K$ transitions uniformly at random from $\mathcal{B}$
10:    **return** Batch $(S, A, R, S', D)$
11: **end procedure**
```

3.3.2 1D CNN Architecture

We employ a 1D Convolutional Neural Network to extract temporal features from the multi-channel sEMG input of shape $(C \times W)$. The network consists of two convolutional layers followed by Global Max Pooling to reduce dimensionality while retaining salient features. These features are processed by fully connected layers to output Q-values for each action, enabling the model to learn robust spatial-temporal representations.

Algorithm 2 1D CNN Q-Network Architecture

```
1: **procedure** FORWARD($x$)                    ▷ Input shape: (Batch, Window, Channels)
2:     $x \leftarrow$ Permute($x$)                 ▷ Reshape to (Batch, Channels, Window)
3:     $x \leftarrow$ ReLU(Conv1d($x$))
4:     $x \leftarrow$ ReLU(Conv1d($x$))
5:     $x \leftarrow$ GlobalMaxPool($x$)           ▷ Max over time dimension
6:     $x \leftarrow$ ReLU(Linear$_1$($x$))
7:     $Q \leftarrow$ Linear$_2$($x$)              ▷ Output Q-values for all actions
8:     **return** $Q$
9: **end procedure**
```

3.3.3 Environment Wrapper

We encapsulate the static dataset into an interactive environment where the state s_t represents a sliding window of EMG signals. The environment compares the agent's action a_t against the ground truth label y_{true}, assigning a reward $r_t = 1$ for correct predictions and $r_t = -1$ otherwise. The window then advances by a fixed step to generate s_{t+1}, formulating the classification task as a sequential decision-making process.

Algorithm 3 Environment Step Function

```
 1: Input: Action $a$, Step Size $k$
 2: Global: Current Index $idx$, Data $X$, Labels $Y$
 3: procedure STEP($a$)
 4:     $L_{idx} \leftarrow idx + \text{WindowSize} - 1$
 5:     $y_{true} \leftarrow Y[L_{idx}]$                              ▷ Ground truth label
 6:     if $a = y_{true}$ then
 7:         $r \leftarrow 1.0$
 8:     else
 9:         $r \leftarrow -1.0$
10:     end if
11:     $idx \leftarrow idx + k$                                      ▷ Slide window forward
12:     $s' \leftarrow X[idx : idx + \text{WindowSize}]$
13:     $done \leftarrow (idx + \text{WindowSize} \geq \text{TotalSamples})$
14:     return $s', r, done$
15: end procedure
```

3.3.4 Training Loop

The agent interacts with the environment using an ϵ-greedy strategy to balance exploration and exploitation. Network parameters θ are updated by minimizing the Mean Squared Error (MSE) loss derived from the Bellman equation:

$$L(\theta) = E_{\mathcal{B}}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)\right)^2\right] \tag{6}$$

where γ is the discount factor. This iterative process allows the Q-network to converge towards the optimal policy for gesture classification.

Algorithm 4 DQN Training Algorithm for EMG

```
 1: Initialize: Q-Network $Q_\theta$, Replay Buffer $\mathcal{B}$, Epsilon $\epsilon$
 2: for episode = 1 to $M$ do
 3:     $s \leftarrow$ Env.Reset()
 4:     while not done do
 5:         Select Action $a$:
 6:         if random() $< \epsilon$ then
 7:             $a \leftarrow$ random action
 8:         else
 9:             $a \leftarrow \arg\max_a Q_\theta(s)$
10:         end if
11:         $s', r, done \leftarrow$ Env.Step($a$)
12:         $\mathcal{B}$.Push($s, a, r, s', done$)
13:         $s \leftarrow s'$
14:         if $|\mathcal{B}| >$ BatchSize then
15:             Sample batch $(s_j, a_j, r_j, s'_j, d_j)$ from $\mathcal{B}$
16:             $y_j \leftarrow r_j + \gamma \max_{a'} Q_\theta(s'_j)$          ▷ Target Q calculation
17:             $L \leftarrow \frac{1}{K}\sum(y_j - Q_\theta(s_j, a_j))^2$
18:             Perform Gradient Descent on $L$
19:         end if
20:     end while
21:     Decay $\epsilon$
22: end for
```

4 Results and Visualisation

4.1 Structure of the data.

The Ninapro database contains various demographic metadata, such as gender, age, and weight [5], which can introduce significant inter-subject variability. To control for these physiological differences and focus exclusively on the relationship between signal patterns and gesture intent, we adopt a **subject-specific approach**. By utilizing data recorded from a single participant (Subject 01), we eliminate confounding variables, ensuring that fluctuations in the data are attributable solely to muscle activation changes rather than anatomical differences.

Within the dataset, the training process relies primarily on two data arrays:

- **’emg’ (Input Signals):** This array contains the raw time-series data captured by the surface electromyography sensors. It represents the electrical potential generated by muscle cells, serving as the high-dimensional state input for the neural network.
- **’restimulus’ (Ground Truth Labels):** This vector represents the “re-stimulus” indices, acting as the ground truth labels. It provides the discrete class ID for the specific gesture the subject was instructed to perform at each timestamp (e.g., 0 for rest, $1 \ldots K$ for active gestures).

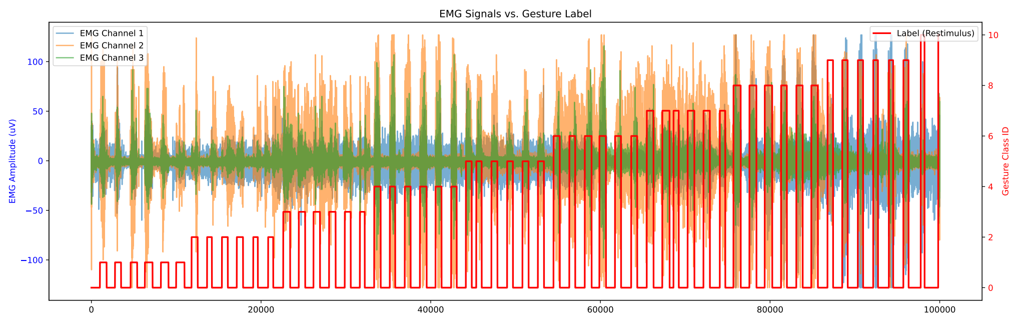


Figure 2: **Visualization of raw sEMG signals synchronized with gesture labels.** The plot depicts a segment of time-series data from Subject 01 (Ninapro DB5). The left y-axis represents the EMG amplitude (μV) for the **first three distinct channels** (blue, orange, green), illustrating the stochastic nature of the muscle activity. The right y-axis (red line) indicates **the corresponding ground truth gesture class** (Restimulus). Distinct bursts in EMG activity correlate with non-zero gesture labels, verifying the temporal alignment between the input signals and the target actions.

4.2 State Representation via Sliding Window

To transform the continuous sEMG time-series into a discrete Markov Decision Process (MDP), we employ a sliding window technique, as illustrated in Figure 3. At each discrete time step t, the state s_t is defined as a matrix of the most recent W samples across C channels:

$$s_t \in R^{C \times W} \tag{7}$$

This formulation ensures that the DQN agent receives sufficient temporal context to distinguish between static gestures and dynamic transitions, while maintaining a fixed input dimension required by the 1D Convolutional Neural Network.

The window slides forward by a fixed stride k at each step. As visually demonstrated in the yellow highlighted region of Figure 3, this window captures the local temporal dynamics of muscle activation immediately preceding the decision point, enabling the agent to infer the user's intent based solely on the provided context.

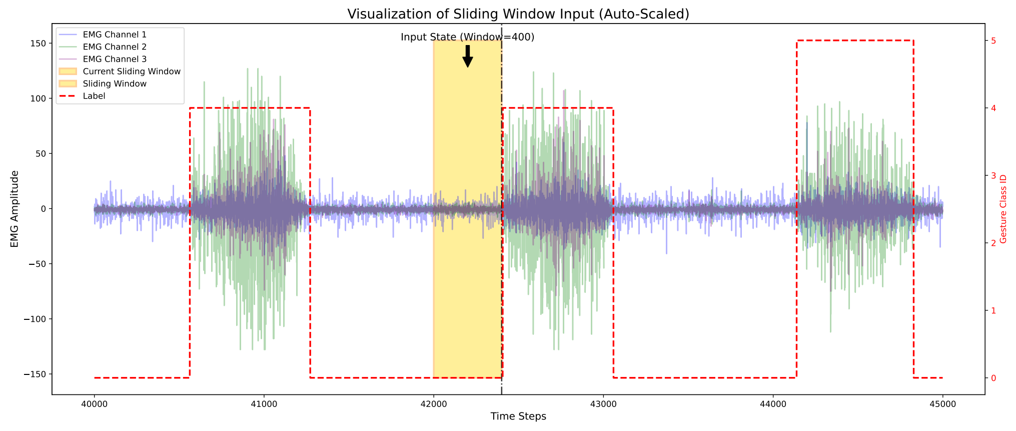


Figure 3: **Illustration of the Sliding Window State Formulation.** The plot visualizes how the continuous sEMG stream is segmented into discrete states for the DQN agent. The multi-channel raw sEMG signals (background waveforms) exhibit distinct activity bursts. The red dashed line represents the ground truth gesture labels. The **yellow highlighted region** indicates the current input state s_t (a sliding window of fixed length, e.g., $W = 400$) fed into the neural network at time step t. The agent must infer the user's intent based solely on the temporal context provided within this window.

4.3 Hyperparameter Configuration

To ensure reproducibility and optimal performance, the specific hyperparameters used in this study are detailed in Table 1. The configuration is categorized into three critical components:

1. **Data and Environment Configuration:** This category defines the structure of the state space. We utilized a *Window Size* of $W = 50$ to capture the immediate temporal context of the gesture. Crucially, a *Step Size* of 50 was selected. Unlike dense sampling, this stride length (equal to the window size) results in non-overlapping windows, which significantly reduces the correlation between consecutive training samples and accelerates the training throughput. The input complexity is defined by the *Channels* parameter, set to $1 - 16$, leveraging data from all 16 active sensors to capture high-resolution spatial patterns of muscle activation. *Max Samples* is set to 20,000 to define the experimental scope and out put dimension.

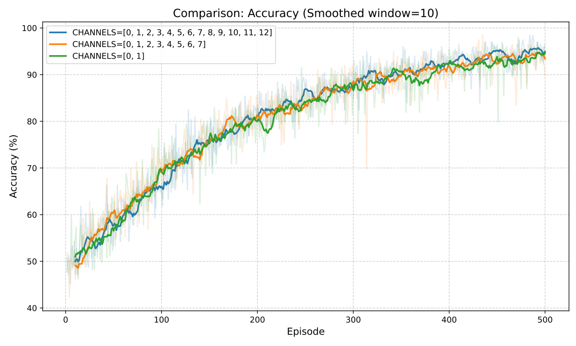
2. **DQN Training Parameters:** These parameters govern the optimization dynamics over a total of 500 *Number of Episodes*, ensuring sufficient iterations for the policy to converge. A *Batch Size* of 64 is utilized; compared to larger batches, this smaller size offers a balance between gradient estimation accuracy and computational efficiency, allowing for more frequent weight updates. The *Learning Rate* is maintained at 0.001. Notably, the *Discount Factor* is set to a high value of $\gamma = 0.995$. This encourages the agent to prioritize long-term cumulative rewards over immediate gains, which is essential for maintaining stable gesture classification over time. The *Replay Buffer* capacity remains at 10,000 transitions.

3. **Exploration Strategy (ϵ-greedy):** To manage the exploration-exploitation trade-off, we implement a decaying ϵ-greedy policy. The agent begins with complete exploration ($\epsilon_{start} = 1.0$) to gather diverse experiences. This probability decays multiplicatively by a factor of 0.995 per episode until it reaches a minimum floor of $\epsilon_{min} = 0.01$, ensuring the agent progressively relies on its learned policy.

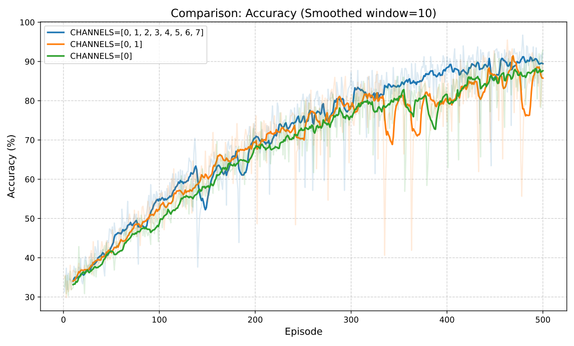
In the following control experiments, modifications are restricted to the *Data and Environment Configuration.* This experimental design is intended to characterize how the DQN architecture generalizes to changes in input dimensionality and scale. Our goal is to understand the model's behavior under different data constraints, as opposed to performing extensive hyperparameter tuning on a static dataset.

4.3.1 Impact of Input Dimensionality (Channel Count)

To evaluate the sensitivity of the DQN agent to spatial information density, we conducted comparative experiments by varying the number of active sEMG channels. The input dimensionality directly determines the spatial resolution of the muscle activation map available to the agent. We compared low-dimensional inputs (2 channels) against high-dimensional inputs (8 and 13 channels) across datasets of different Output dimensions (2 and 3 dimensions). The results are illustrated in Figure 4.



((a)) 2 output dimensions (10k Samples)



((b)) 3 output dimensions (20k Samples)

Figure 4: **Performance comparison across different input dimensions.** Subfigure (a) compares 2, 8, and 13 channels on a smaller dataset, showing similar convergence rates. Subfigure (b) compares 2 vs. 8 channels on a larger dataset, where the high-dimensional input (blue) exhibits superior stability and higher peak reward compared to the low-dimensional input (orange and green).

Observations and Analysis:

- **Convergence and Robustness.** As observed in Figure 4, the DQN agent demonstrates remarkable robustness. Even with minimal spatial information (Channels=[0]), the agent successfully learns the policy, achieving an accuracy exceeding 90% after 400 episodes. This suggests that the 1D CNN is highly effective at extracting *temporal* features from individual sensors.

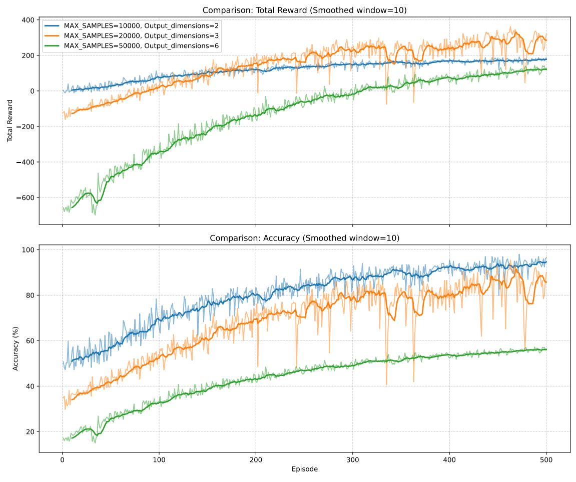
- **Stability and Peak Performance.** However, the advantage of higher dimensionality becomes evident in the stability of the learned policy. In Figure 4(b), the 8-channel configuration (blue line) maintains a stable high-reward trajectory in the late training phase (Episodes 300-500). In contrast, the 2-channel configuration (orange line) and 1-channel configuration (green line) exhibit significant variance and occasional performance drops (spikes in the reward curve).

- **Summary.** This phenomenon indicates that while limited channels are sufficient for basic classification, complex gestures may produce similar temporal signatures on a small subset of muscles. The additional channels provide critical spatial disambiguation, reducing state aliasing and allowing the agent to distinguish between subtle gesture variations with higher confidence. Therefore, while the algorithm is functional with low dimensions, utilizing the full sensor array (8+ channels) is recommended for optimal reliability.

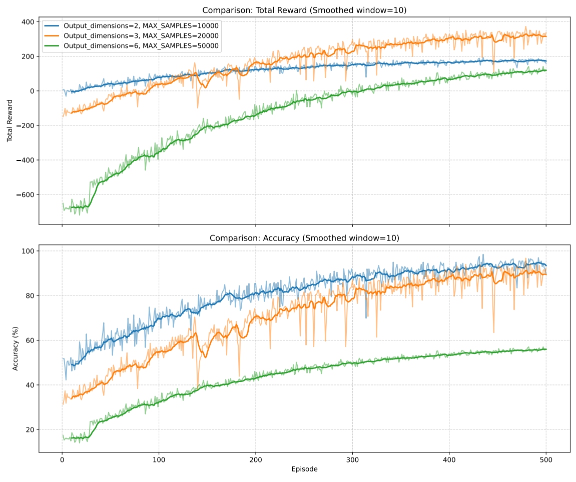
In the next section, we will explore how the output dimension affects the final performance of DQN.

4.3.2 Impact of Output Dimensionality (Action Space Complexity)

The output dimensionality, defined by the number of target gesture classes, represents the complexity of the agent's action space. To evaluate how task complexity influences the learning process, we compared configurations with varying numbers of gesture classes (e.g., 2, 3, and 6 classes). The results, illustrating the relationship between output dimensions and model performance, are shown in Figure 5.



((a)) Impact of 1 vs. 2 output dimensions.



((b)) Impact of 2, 3, and 6 output dimensions.

Figure 5: **Performance comparison across different output dimensions.** The plots show a clear inverse correlation between the number of gesture classes and the convergence rate. As the action space expands, the agent faces a more challenging exploration task, leading to lower accuracy and slower reward growth.

Observations and Analysis:

- **Action Space Explosion and Exploration Difficulty.** As illustrated in Figure 5(b), the number of output dimensions is the primary determinant of the learning curve's slope. With *Output_dimensions*=2 (blue line), the agent rapidly achieves over 90% accuracy. However, when the dimensionality increases to 6 (green line), the accuracy drops drastically to approximately 55%. This phenomenon occurs because a larger action space exponentially increases the difficulty of the exploration phase; the probability of the agent receiving a positive reward through random ϵ-greedy actions diminishes, leading to sparse reinforcement signals that hinder the weight updates in the early stages.

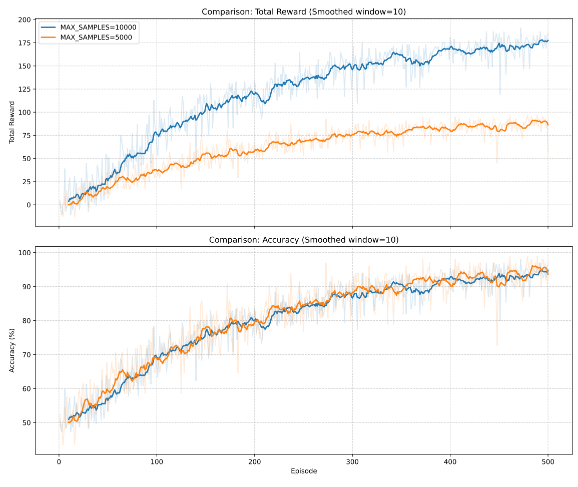
- **Classification Ambiguity and State Aliasing.** Increasing the output dimensionality also introduces higher inter-class similarity, particularly in myoelectric control where different gestures may activate overlapping muscle groups. In Figure 5(a), increasing the dimension from 1.0 to 2.0 (orange line) results in a noticeable delay in convergence and higher variance in the reward. This suggests that as more classes are added, the "boundaries" between gestures in the sEMG feature space become more crowded, requiring the 1D CNN to extract much finer temporal patterns to avoid misclassification.

- **Summary.** The experimental results demonstrate that increasing the output dimensionality significantly impedes the learning efficiency of the DQN agent, requiring a substantially higher number of episodes to identify optimal behaviors. Furthermore, a clear degradation in the final convergence accuracy is observed as the complexity of the action space grows.

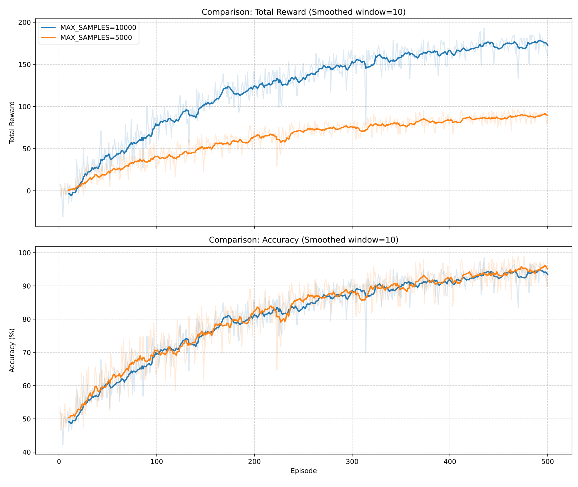
However, a critical observation from the current data is the concurrent increase in the training data volume (*MAX_SAMPLES*) alongside the rise in output dimensions. To isolate these effects, the subsequent section will implement a controlled variable approach, maintaining a constant output dimensionality to specifically investigate how the volume of data independently influences the model's ultimate performance.

4.3.3 Impact of Data Volume (Sample Size)

In this control experiment, the output dimensionality was held constant to isolate the effects of data volume. We compared two scales: *MAX_SAMPLES* = 5,000 and 10,000. As illustrated in Figures 6(b) and 6(a), the impact of data volume on the agent's performance is **minimal**.



((a)) Data volume impact (2 Channels).



((b)) Data volume impact (8 Channels).

Figure 6: **Performance comparison across different data volumes.** The plots indicate that while the *Total Reward* increases with more samples due to longer episodes, the *Accuracy (%)* remains consistently high across both 5,000 and 10,000 sample scales, demonstrating the model's robustness to data volume fluctuations.

Observations and Analysis:

- **Accuracy Convergence:** In both the 8-channel and 2-channel configurations, the *Accuracy (%)* curves for both data scales are nearly identical, with both eventually converging to a high-performance level above 90%. This suggests that for a fixed task complexity, even a smaller sample set (5,000) provides sufficient representative patterns for the 1D CNN to learn effectively.

- **Total Reward Variance:** While the *Total Reward* is higher for the 10,000-sample scale, this is a mathematical artifact of the cumulative reward calculation over a longer episode duration, rather than an indication of superior learning efficacy.

In conclusion, once the action space is fixed, the DQN agent is robust to variations in data volume at this scale, indicating that the complexity of the gestures (Output Dimensions) is a far more critical factor than the absolute number of training samples.

5 Conclusion

This research presents a comprehensive evaluation of a **DQN-based framework for gesture recognition using sEMG signal**s. By analyzing the synthesized results in Figure 7, we draw the following definitive conclusions regarding the interplay between system architecture and task complexity:

- **Dominance of Task Complexity:** The experimental data reveals that *Output Dimensionality* (the number of gesture classes) is the single most significant factor affecting performance. As shown in the overall comparison, while 2-class tasks consistently achieve near 100% accuracy, the 6-class tasks (green curves) struggle to surpass 60% accuracy, even with a fivefold increase in training data (50,000 samples). This confirms that the difficulty of the exploration space in reinforcement learning grows non-linearly with the number of possible actions.
- **Synergy between Input Channels and Stability:** The "Overall" plot underscores the importance of spatial information. Configurations with higher *Channel Counts* (e.g., 8 or 13 channels) exhibit much smoother learning trajectories and higher reward stability compared to single-channel or dual-channel setups. While the 1D CNN can extract features from limited sources, the additional spatial resolution provided by more electrodes is essential for mitigating signal noise and distinguishing between similar gesture patterns.
- **Marginal Gains from Data Volume:** A critical finding of this study is that simply increasing *Data Volume* cannot compensate for high task complexity. The overlap between 5,000 and 10,000 sample curves in low-dimensional tasks suggests that the model reaches saturation quickly. Consequently, future optimizations should focus on enhancing the agent's exploration strategy (e.g., Prioritized Experience Replay) rather than merely expanding the dataset size.

In conclusion, the proposed 1D CNN-DQN architecture is highly effective for low-to-medium complexity gesture recognition – it almost reached the level of a normal human being. So, this theory can subsequently be applied to the development of simple prosthetics, exoskeletons, and other medical devices.

However, for high-dimensional gesture sets, the research points toward a need for more sophisticated algorithm, state representation and reward shaping and greater computing power to overcome the bottlenecks identified in this study.

A Hyperparameters table

Table 1: List of Hyperparameters and Configuration

Hyperparameter	**Value**	**Description & Potential Impact**
Data & Environment Configuration		
Window Size (W)	50	Input sequence length. Determines the temporal context visible to the agent.
Step Size	50	Stride for the sliding window. Smaller steps yield more training samples but higher correlation.
Channels	1-16	Selected sEMG sensor channels (total 16 active sensors).
Max Samples	20000	Data truncation limit for faster experimentation.
DQN Training Parameters		
Number of Episodes	500	Total number of training iterations. Defines the overall duration of the learning process.
Learning Rate (α)	0.001	Step size for the Adam optimizer. Controls convergence speed.
Batch Size	64	Number of transitions sampled per gradient update. Larger batches stabilize gradients.
Discount Factor (γ)	0.995	Importance of future rewards. High γ encourages long-term foresight.
Buffer Capacity	10000	Maximum number of transitions in the Replay Buffer.
Exploration Strategy (ϵ-greedy)		
Epsilon Start (ϵ_{start})	1.0	Initial probability of random exploration.
Epsilon Decay	0.995	Multiplicative decay factor per episode.
Epsilon Min (ϵ_{min})	0.01	Minimum floor for exploration probability.

B Overall graph:

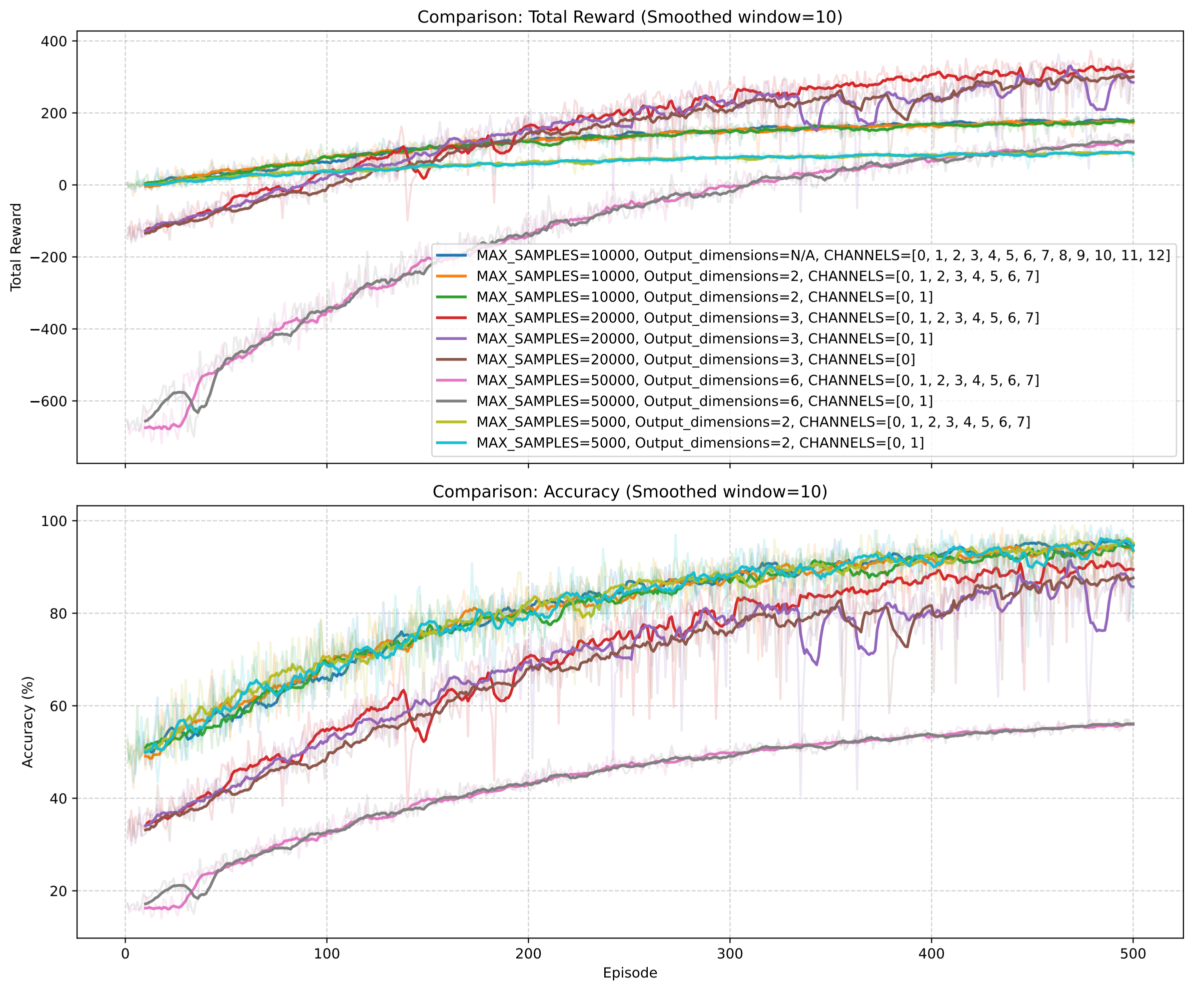


Figure 7: **Comprehensive performance comparison across all experimental variables.** This plot synthesizes the effects of channel counts (2, 8, 13), output dimensions (2, 3, 6), and data volumes (5k, 10k, 20k, 50k). It clearly illustrates the inverse relationship between action space complexity and convergence efficiency.

References

[1] Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT Press.

[2] Mnih, V., Kavukcuoglu, K., Silver, D. et al. Human-level control through deep reinforcement learning. Nature 518, 529–533 (2015). https://doi.org/10.1038/nature14236

[3] Bellman, R. E. (1957). *Dynamic Programming*. Princeton University Press.

[4] https://arxiv.org/abs/1509.06461

[5] https://ninapro.hevs.ch/instructions/DB5.html