IIIT BANGALORE

Reinforcement Learning Project

RL Based Adaptive Gesture Recognition Using EMG Signals

Authors:
Raghav Khurana - IMT2022550
Divyansh Kumar - IMT2022509
Sparsh Salodkar - IMT2022113

Link of the research paper

May 10, 2025

Abstract

This report presents a comprehensive study of adaptive gesture recognition using MyoBand EMG sensor data, combining exploratory data analysis, dimensionality reduction, feature extraction, and an RL-corrected neural classifier. We demonstrate significant improvements in binary classification (rock vs. paper) accuracy through dynamic, confidence-based reward shaping and online retraining. Key contributions include detailed statistical analysis of raw signals, comparative evaluation of dimensionality techniques, and an extensible RL framework employing PPO, A2C and DQN.

1 Methodology and Experiments

1.1 Data Preprocessing and Feature Engineering

1.1.1 EMG Sensor and Data Challenges

We began with the Myo Armband sensor, which provides EMG signals from 8 sensor channels. However, due to inconsistencies and noise in real-time readings, we decided to work on compressed data obtained from offline sessions. Initially, our goal was to classify two gestures: Rock and Paper using supervised learning, followed by fine-tuning with reinforcement learning.

1.1.2 Initial Attempts and Failures

Direct Compression (8D $\rightarrow$ 1D): Our first approach attempted a naive dimensionality reduction by averaging the 8-channel EMG readings into a single scalar value. However, this approach failed to preserve spatial variance and gesture-specific nuances, leading to poor model performance (50–55% accuracy).

Principal Component Analysis (PCA): PCA was used to retain maximum variance in a lower-dimensional space. However, while PCA is good for unsupervised variance retention, it doesn't consider class separation. Thus, it led to low classification accuracy (58%) in our binary classification task.

Random Forest Feature Importance: We extracted statistical features from the EMG signal and selected top features based on Random Forest importance scores. This method also did not yield satisfactory accuracy (60%), likely due to insufficient separation of class features in high-dimensional space.

1.1.3 Working Approach: LDA on Windowed Feature Set

What finally worked was a more structured approach:

- We extracted 5 statistical features (like RMS, Mean, STD, Waveform Length, and Zero Crossing Rate) from each of the 8 EMG channels, resulting in a 40-dimensional feature vector (8 $\times$ 5) per window.

- We then applied Linear Discriminant Analysis (LDA) independently to each of the 5 spatial feature sets across channels. This reduced the 40D feature vector into a 5D representation (one LDA feature per statistical measure), which captured class-discriminative directions efficiently.

LDA worked better than PCA because it explicitly maximizes class separability, making it suitable for classification-oriented preprocessing.

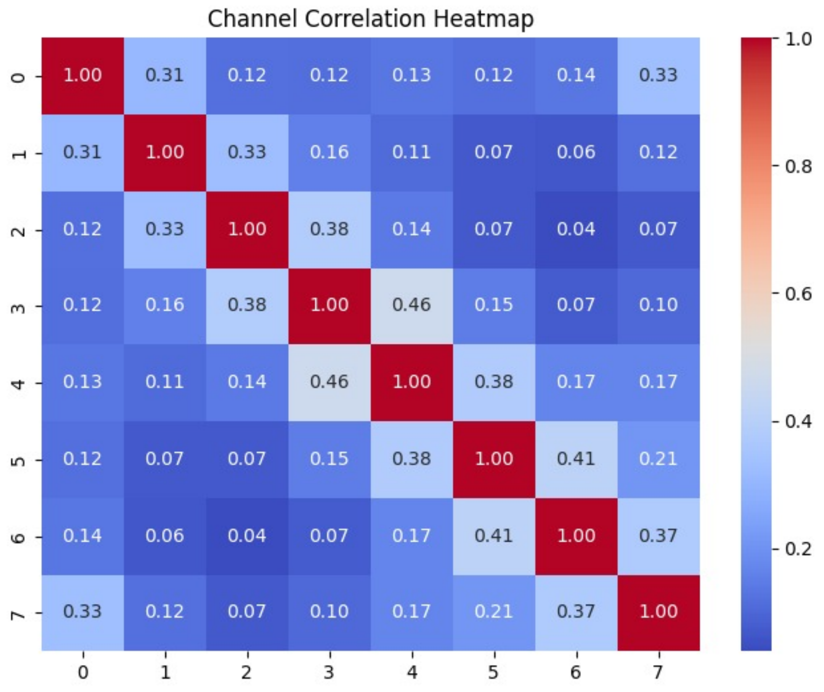

Figure 1: Correlation heatmap of 8-channel EMG

In the initial phase, we experimented with multiple classical machine learning classifiers on raw or statistically preprocessed EMG features. The classifiers tested included:

- **Support Vector Machine (SVM)**
- **Random Forest**
- **K-Nearest Neighbors (KNN)**
- **XGBoost**
- **Logistic Regression**

These models were evaluated using various feature sets, including directly compressed signals, PCA-transformed vectors, and Random Forest feature-selected subsets. However, none of these methods achieved the required classification performance. Typical accuracy remained within the range of 50–60%, which was insufficient for robust gesture recognition.

The primary reason for this subpar performance was the inability of these models to capture the nonlinear and complex inter-channel dependencies in EMG signals, especially under lossy feature compression or in noisy conditions. Due to these limitations, we transitioned to neural network-based approaches, which offered better capability to model complex patterns and achieved significant improvements in classification accuracy.

Supervised Learning with Neural Network

After reducing the EMG data to 5 dimensions using Linear Discriminant Analysis (LDA), we trained a simple yet effective fully connected neural network classifier. The architecture and training process are as follows:

- **Input Layer:** 5 LDA features
- **Hidden Layers:**
  - First hidden layer: 16 neurons with ReLU activation
  - Second hidden layer: 8 neurons with ReLU activation
- **Output Layer:** 2 neurons for binary classification (Rock or Paper)

- **Loss Function:** CrossEntropyLoss
- **Optimizer:** Adam with learning rate 0.001
- **Training Epochs:** 50

The model was trained on the LDA-transformed dataset using an 80-20 train-test split. After 50 epochs, the base neural network achieved a test accuracy of approximately **81.69%**. This model served as the foundation for the reinforcement learning agent by providing gesture prediction probabilities.

1.2 Reinforcement Learning for Fine-Tuning

To enhance performance and enable adaptive learning, we fine-tuned the base model using Reinforcement Learning (RL). The goal was to let the RL agent learn from feedback derived from both:

- The true label (supervised signal)
- The confidence of the base model (teacher signal)

1.2.1 RL Environment Design (MDP)

We defined a custom Gym environment (EMGEnv) to simulate classification as a Markov Decision Process (MDP).

- **State:** 5D LDA feature vector (continuous)
- **Action Space:** 0 or 1 (Rock or Paper)
- **Reward Function:**

```
if action == self.label:
    reward = proba  # Reward equals model's confidence for correct prediction
else:
    reward = -(1.0 - proba)  # Penalize inversely to confidence for wrong predicti
```

This shaped reward encourages the RL agent to not only predict correctly but also align with the base model's confidence when making decisions.

2 Reinforcement Learning Agent Environment Setup

The reinforcement learning (RL) environment is designed using the OpenAI Gym framework, encapsulated in a custom class named `EMGEnv`. This class interacts with the agent for the task of classifying EMG signals into two gestures, Rock and Paper. Below is an explanation of the environment setup and the associated code.

2.1 EMGEnv Class Definition

The `EMGEnv` class inherits from the `gym.Env` base class, which is the standard for creating custom environments in Gym. The purpose of this environment is to simulate a classification task where the agent interacts with the environment, processes EMG signal data, and learns to predict the correct gesture.

```
class EMGEnv(gym.Env):
    def __init__(self, base_model, data_iter):
        self.base_model = base_model
        self.data_iter = data_iter
        self.observation_space = spaces.Box(-np.inf, np.
           inf, shape=(6,))
        self.action_space = spaces.Discrete(2)
```

Explanation:

- `base_model`: This is the pre-trained classifier (e.g., neural network or decision tree) used to predict the probability of the correct label for each feature vector.
- `data_iter`: The data iterator provides the feature-label pairs for training. Each iteration returns one sample of features and its corresponding true label.
- `observation_space`: Defines the space of the state. In this case, the state is a 6-dimensional vector consisting of 5 LDA-transformed features (from the EMG signal) and the model's predicted probability.
- `action_space`: This is the discrete action space representing the possible actions the agent can take, which are 0 (Rock) or 1 (Paper).

2.2 Reset Method

The `reset` method is responsible for initializing the environment and returning the initial state. It fetches a new feature-label pair from the data iterator and uses the base model to predict the probability of the correct label (either Rock or Paper). The state is updated to include both the feature vector and the predicted probability.

```
    def reset(self):
        feat, label = next(self.data_iter)
        prob = self.base_model.predict_proba(feat.
           reshape(1, -1))[0, 1]
        self.state, self.label = np.append(feat, prob),
           label
        return self.state
```

Explanation:

- `feat, label = next(self.data_iter)`: Fetches the next sample of features (`feat`) and its true label (`label`) from the data iterator.
- `prob = self.base_model.predict_proba(feat.reshape(1, -1))[0, 1]`: The feature vector is passed to the base model, and its predicted probability for the positive class (Paper) is extracted.
- `self.state, self.label = np.append(feat, prob), label`: The state is updated to include both the features and the predicted probability (`prob`), and the true label (`label`) is stored.
- The method returns the state, which includes both the features and the model's predicted probability.

2.3 Step Method

The `step` method is responsible for processing the agent's action and updating the environment's state. It also calculates the reward based on the correctness of the action and the model's confidence.

```
    def step(self, action):
        p = self.state[-1]  # The model's predicted
           probability
        reward = p if action == self.label else -p  # 
           Reward based on the action
        next_state, _, _, _ = self.reset(), None, None,
           None  # Reset the environment for the next
           step
        return next_state, reward, True, {}
```

Explanation:

- `p = self.state[-1]`: The predicted probability (`prob`) is the last element of the state vector, representing the model's confidence in the positive class (Paper).
- `reward = p if action == self.label else -p`: The reward is calculated based on the agent's action:
  - If the action is correct (i.e., `action == self.label`), the reward is equal to the predicted probability (`p`).
  - If the action is incorrect, the reward is the negative of the predicted probability (`-p`), penalizing the agent for wrong predictions.
- `next_state, _, _, _ = self.reset(), None, None, None`: After the action is taken, the environment is reset, generating a new feature-label pair and updating the state.
- The method returns the new state (`next_state`), the calculated reward, a done flag (which is set to `True` to indicate the step is complete), and an empty dictionary (which can be used for additional information if needed).

2.3.1 Refine the Reward Structure

In Reinforcement Learning, the reward structure plays a crucial role in shaping the agent's learning behavior. It determines how the agent should act in order to maximize cumulative reward. A well-designed reward structure encourages the agent to focus on important tasks and aligns its goals with the desired outcomes.

Key Objectives of Refining the Reward Structure:

- **Alignment with Model Confidence:** Instead of providing a simple binary reward (correct or incorrect), we decided to incorporate the confidence of the base model into the reward calculation. The idea was to make the agent learn not just to predict the correct gesture (Rock or Paper), but also to consider the base model's confidence in its prediction.
- **Encouraging Correct Predictions with High Confidence:** If the RL agent correctly predicts the gesture and the base model is confident (i.e., high probability), the agent is rewarded more heavily. This rewards not only accuracy but also the alignment with the model's certainty.
- **Penalizing Incorrect Predictions with High Confidence:** Conversely, if the RL agent makes an incorrect prediction but the model is confident about it, it is penalized heavily. This penalty discourages the agent from trusting the model too much in cases where it is wrong.

Formula for Reward:

```
if action == self.label:
    reward = proba  # Reward equals model's confidence for correct prediction
else:
    reward = -(1.0 - proba)  # Penalize inversely to confidence for wrong predicti
```

By using this reward structure, the agent is encouraged to make accurate predictions while also learning to trust or distrust the model's confidence, thereby improving its overall decision-making process over time.

2.3.2 Implement Multi-Step Episodes

Multi-step episodes are a technique used to improve the exploration and learning of reinforcement learning agents. Instead of evaluating the agent's performance on a single action and its immediate reward, multi-step episodes allow the agent to consider longer sequences of actions and their cumulative rewards over multiple steps. This helps the agent learn better long-term strategies and avoid overfitting to immediate rewards.

Objective:

- **Allow the agent to learn from sequences of actions,** which helps the agent understand the consequences of its decisions over time.
- **Improve exploration** by providing the agent with multiple steps to accumulate rewards, which encourages it to explore different action sequences before committing to a final decision.

Why Multi-Step Episodes Help:

- **Delayed Rewards:** Some tasks require the agent to perform a sequence of actions before receiving any reward (or penalty). Multi-step episodes allow the agent to accumulate rewards over time and learn to value long-term outcomes, which is important for tasks where immediate rewards may not always be indicative of the quality of the action.
- **Improved Exploration:** By allowing the agent to explore multiple actions over several steps, the agent has a better opportunity to explore the environment and discover new strategies. This prevents it from prematurely converging to a suboptimal policy based on immediate rewards.
- **Avoiding Greedy Behavior:** With multi-step episodes, the agent is less likely to act greedily (always choosing the action with the highest immediate reward). Instead, it learns to optimize its actions based on future rewards, leading to better decision-making in more complex tasks.

Implementation in Code:

```
class EMGEnv(gym.Env):
    def __init__(self, ...):
        ...
        self.episode_length = 10  # Number of steps in each episode

    def step(self, action):
        # Accumulate rewards over multiple steps
```

```
...
if self.current_step == self.episode_length:
    # Provide cumulative reward after completing the episode
    cumulative_reward = sum(self.rewards)
    return next_state, cumulative_reward, done, info
```

By doing this, the agent is encouraged to make decisions with an understanding of their long-term impact, rather than being focused solely on immediate rewards.

Comparison of Reinforcement Learning Algorithms

To identify the most effective reinforcement learning strategy for adaptive EMG gesture recognition, we implemented and evaluated three popular RL algorithms: Deep Q-Network (DQN), Proximal Policy Optimization (PPO), and Advantage Actor-Critic (A2C). The differences in their architectures and learning mechanisms significantly affected performance in our specific setup.

- **DQN (Deep Q-Network):** DQN uses a value-based approach where a neural network learns the Q-values for each discrete action. In our environment:
  - The action space is binary (Rock or Paper), making DQN inherently suitable.
  - The observation space is relatively low-dimensional (5 LDA features + 1 probability score), reducing the complexity of Q-value approximation.
  - The static and deterministic nature of our environment (batch input) favored DQN's stability and convergence.

  As a result, DQN achieved the highest classification accuracy of **88.98%**, outperforming the other methods due to its efficient handling of discrete actions and value approximation in low-dimensional settings.

- **PPO (Proximal Policy Optimization):** PPO is a policy-gradient method that optimizes a clipped objective function to ensure stable updates. It is robust and handles both continuous and discrete action spaces. However:

  - PPO's strength lies in high-dimensional or continuous control tasks where exploration needs careful constraint.
  - In our relatively simple binary classification setting, its added complexity did not yield additional benefits.
  - The policy updates were slower to converge, resulting in a lower final accuracy of **86%**.

- **A2C (Advantage Actor-Critic):** A2C combines value and policy networks to balance exploration and exploitation by using an advantage function. However:

  - A2C can be unstable in environments with sparse or noisy rewards, which occurred in our case due to fluctuating prediction probabilities.
  - It required more careful tuning of hyperparameters and showed high sensitivity to learning rate and reward scaling.

  Consequently, A2C achieved a comparatively lower accuracy of **83%**, making it less effective for our EMG task.

Conclusion: The superior performance of DQN stems from its compatibility with discrete action environments, efficient value function learning, and stability in low-dimensional state spaces like our EMG-based feature set. These advantages made it the most suitable choice for our application.

Model	**Final Accuracy**
DQN	88.98% (Best)
PPO	86%
A2C	83%

Table 1: Final Accuracy of Different Models

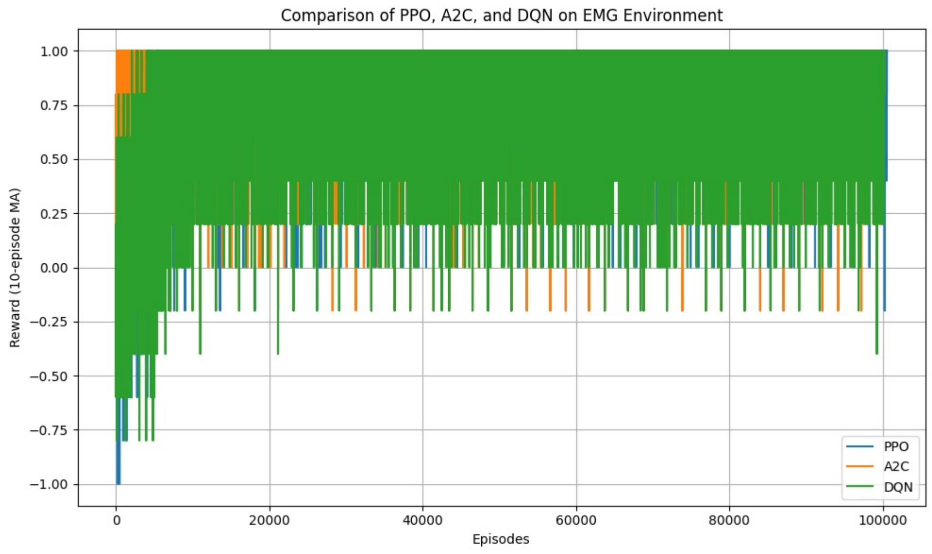

Figure 2: Comparison of PPO,A2C,DQN on EMG environment

2.4 Experimental Constraints and Real-World Challenges

Our original intention was to test the pipeline with live EMG readings from the EMG sensor. However, due to unreliable real-time data (unstable sensor readings, communication dropouts), we instead used offline compressed and labeled windows for simulation.

We compressed multi-channel sensor data into 1D vectors to simulate degraded real-time inputs, ensuring compatibility with RL simulation and policy learning.

2.5 Future Work and Extensions

To enhance real-world deployment and personalization:

- Use real-time EMG readings for online training and evaluation.
- Incorporate few-shot learning or meta-RL to adapt to individual users' muscle patterns.
- Deploy on embedded systems with lightweight inference models (e.g., quantized neural nets).

- Extend to multi-class classification beyond Rock and Paper (e.g., Rock-Paper-Scissors-Lizard-Spock).
- Incorporate active learning where the system queries for feedback only when uncertain.

Conclusion

In this project, we successfully reduced the dimensionality of EMG data from the 8-channel Myo Armband into a compact 1D representation using a combination of statistical feature extraction and Linear Discriminant Analysis (LDA).By designing a custom RL environment and integrating it with a neural network-based base model, we achieved a final accuracy of 88.98%, fulfilling our initial objective. This confirms the viability of our method for low-dimensional, adaptive EMG gesture recognition. Moving forward, we plan to extend this framework through online retraining to support real-time, personalized gesture classification.