Square Nut Assembly via Behavior Cloning

Maanas Gantla
Henry Samueli School of Engineering
University of California, Los Angeles
gantlamr@g.ucla.edu

Abstract: The purpose of this project is to successfully complete the robosuite Nut Assembly Square task by learning a low-dimensional, behavior-cloned policy that maps a 10-dimensional state (robot end effector position, square nut position, square nut orientation) to 7D joint velocity commands. The neural network is a two-layer multilayer perceptron (MLP) trained on demonstration data. After evaluating over 50 trials, the neural network with the optimal hyperparameters used during training achieved an 82% insertion success rate. This paper discusses some of the limitations, methodologies, results, and future directions for improving success rate via reward-based finetuning.

Keywords: Robots, Nut Assembly Square, Behavior Cloning, Deep Learning

1 Problem Statement

Consider the robosuite Nut Assembly Square environment (panda arm and square nut), where the orientation and the position of the square nut are randomized on each trial. The goal is to learn a closed-loop policy $\pi : \mathbb{R}^{10} \to \mathbb{R}^7$ from demonstrations alone that reliably inserts the nut within a 2,500 timestep limit using only 3D end-effector position, 3D nut position, and 4D nut quaternion.

2 System Design / Methodology

2.1 Data Pre-processing

To generalize, the data extractor, with the help of the provided `load_data.py` [1], loads N demonstrations from `demos.npz` [2] or the specified path towards the given data, each recording:

- end effector position: $\mathbf{e}_t \in \mathbb{R}^3$,
- square nut pose: $(\mathbf{p}_t \in \mathbb{R}^3,\ \mathbf{q}_t \in \mathbb{R}^4)$,
- action: $\mathbf{a}_t \in \mathbb{R}^7$.

From here, concatenate each member of the demonstration to form the following vector: $s_t = [\mathbf{e}_t; \mathbf{p}_t; \mathbf{q}_t] \in \mathbb{R}^{10}$, standardize each dimension to have zero-mean and unit variance, and assemble the dataset for all N demonstrations in the form: $\{(s_t, a_t)\}$, where t is the index of a particular demonstration.

2.2 Neural Network Architecture and Training

The policy network is a 2-layer MLP, represented by the following equation:

$$f(s) = W_2\,\mathrm{ReLU}(W_1\,s + b_1) + b_2, \quad W_1 \in \mathbb{R}^{128\times 10},\ W_2 \in \mathbb{R}^{7\times 128}.$$

The optimal hyperparameters used in training the neural network were 100 epochs with Mean Squared Error loss and Adam optimizer with learning rate of $\eta = 10^{-3}$, 90/10 train/validation split, batch size of 256, and step learning rate halving every 20 epochs.

2.3 Inference

At runtime (see `deep_inference.py`), normalize the observed 10D state using saved statistics from the data extraction process, forward it through the MLP, and apply the prediction, in this case, the 7D output as joint-velocity commands, until the environment returns reward $= 1$ (success) or the 2,500 time step limit is exceeded (in which case the reward $= 0$ (failure)).

3 Evaluation Results

Evaluating over 50 randomized trials yields:

- **Behavior cloning success:** 41/50 (82%).
- **Learning curves:** The training loss approached ≈ 0.035, while the validation loss approached ≈ 0.020.

  The corresponding graph is listed in Figure 1 for the loss measured against the number of epochs.

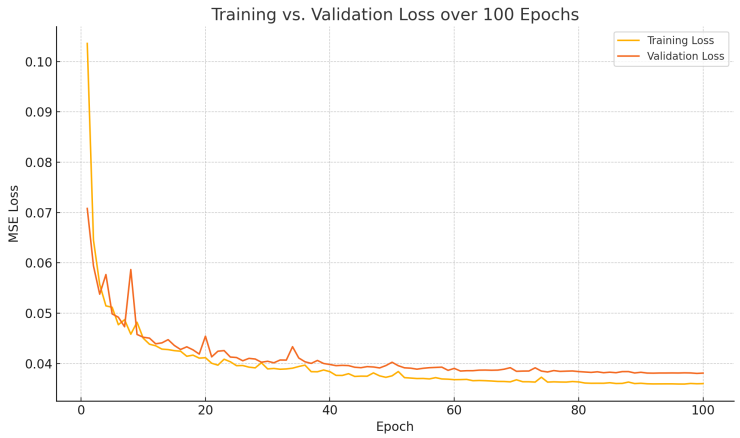


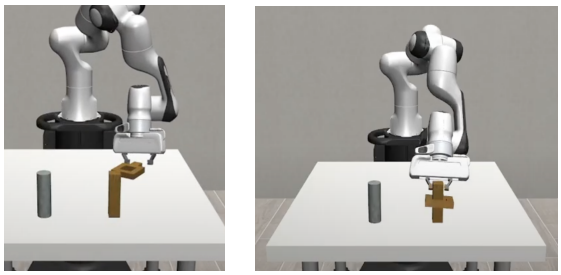
Figure 1: Training (orange) vs. Validation (red) Mean Squared Error loss over 100 epochs for the two-layer MLP.

3.1 Hyperparameter Tuning

After performing a hyperparameter sweep over hidden-layer size, batch size, and learning rate, the following results were measured:

Hidden dim	Batch size	Learning rate	Success rate
64	128	1×10^{-3}	68%
64	256	1×10^{-3}	70%
128	128	1×10^{-3}	75%
128	256	1×10^{-3}	**82%**
128	256	5×10^{-4}	78%

4 Discussion and Reflections



(a) Failed trial (b) Successful trial

Figure 2: (a) A failed trial, where the gripper fails to properly grasp the nut. (b) A successful trial.

In terms of strategies and mechanisms that were successful in completing the task, the behavior cloning on the low-dimensional state is simple to implement, trains fast, and achieves a high success rate (up to 82%). It uses a relatively limited amount of compute, as the two-layer MLP converged in a few minutes on CPU, making it very feasible to utilize as a strategy. Initially, a nearest-neighbors strategy was tested among the 200 demonstrations, where the demonstration with the closest end effector and target poses would have its joint command executed in the current trial. The issue with this approach was that it did not generalize as well as the neural network policy, since some of the random poses of the square nut were not represented well in the demonstration data. In the initial processes of deciding which methods to choose, a Dynamic Movement Primitive (DMP) based controller was used in order to replay the trajectories of inserting the nut. However, it was difficult for the DMP-based approach to capture the orientation changes of the square nut and align the square nut with the peg, and the neural network did a relatively better job at these parts of the insertion task compared to the DMP.

Some issues with the neural network approach were that there was overfitting at times when a larger hidden dimension size was used, like 256 neurons. In addition, some learning rates were quite small and achieved very minimal training loss, but they seemed to cause overfitting as their validation error was relatively high. An issue with the neural network approach is that there are only 200 demonstrations provided in the given dataset. This may not be enough for the neural network to truly learn the mapping between positions and orientations to the appropriate action to take. Neural networks tend to interpolate well but struggle with extrapolation, so quite a bit of variation in the results could be noticed depending on the randomized position of the nut. If the nut's pose was closer to a pose represented in the training data, there was likely a higher chance that the policy would have more success on this trial relative to one in which the nut's pose is not captured well in the training data.

Another factor to consider is that the reward in the environment ($r = 1$ only on success) offers no gradient signal to recover. Fine-tuning via proximal policy optimization (PPO) on top of the MLP model could potentially help the policy better explore different insertion pathways, which would be the future direction of this research, given more time to conduct these reinforcement learning experiments.

5 Team Contribution Division

- **Maanas Gantla:** Data pre-processing, neural network design, creating training scripts, hyperparameter optimization, experiment evaluation, report writing, demonstration video.

References

[1] Y. Cui. load_data.py: Npz reconstructor for robosuite demonstrations. GitHub repository / local script, 2025. URL https://github.com/MaanasGantla/CS188-Final-Project/blob/main/load_data.py. Accessed: 2025-06-08.

[2] CS188 final project demonstrations dataset. demos.npz (UCLA CS 188 Course Materials), 2025. Provided by course staff.