Implementation of iLQR to Solve Gymnasium Tasks

Ganga Nair B
M.Tech RAS, 1st Year

E2 335 – Topics in Artificial Intelligence, 2025

Contents

Motivation

- Reinforcement Learning (RL) aims to find optimal policies maximizing cumulative reward.
- Optimal control seeks actions that minimize cost while achieving goals.
- RL has deep roots in control theory.
- One such method is the Iterative Linear Quadratic Regulator (iLQR).

Optimal Control Methods

- iLQR provides near-optimal control policies analytically with fewer iterations.
- Offers theoretical guarantees, interpretability, and low computational cost.
- Assumes accurate dynamics of the system are known.

Aim

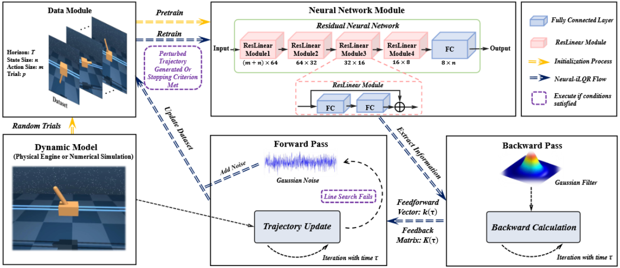
To solve benchmark control problems in the Gymnasium environment using model-based RL methods, and extend the framework to unknown system dynamics.

Scope

- Apply iLQR to classical Gym tasks:
  - Mountain Car
  - Swing-Up Cart Pole
- Learn dynamics using a neural network from scratch.
- Use learned dynamics in iLQR to control the system.

Neural-iLQR: Trajectory Optimization

- **Objective:** Overcome model inaccuracies in conventional iLQR for nonlinear systems
- **Solution:**
  - Integrates 2 layered NN to learn local dynamics online
  - Alternates between policy optimization & network training



Literature Review: Neural-iLQR & Related Works

Physics-Informed Vehicle Tracking

- **Objective:** Improve trajectory tracking accuracy with physical consistency
- **Solution:**
  - Combines PINN with Kinematic Bicycle Model
  - ILQR controller used for control

What is Linear Quadratic Regulator (LQR)?

- Solves constrained optimization using dynamic programming.
- Backward pass computes gain matrices K and k.
- Forward pass generates control trajectory using the optimized gains.

Linear Quadratic Regulator (LQR)

Assumptions:

- Linear time-varying system dynamics:

$$\mathbf{x}_{t+1} = \mathbf{F}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \mathbf{f}_t$$

- Quadratic cost function:

$$c(\mathbf{x}_t, \mathbf{u}_t) = \frac{1}{2} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{C}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{c}_t$$

LQR Problem Statement

Objective:

$$\min_{\mathbf{u}_0,\ldots,\mathbf{u}_{T-1}} \sum_{t=0}^{T-1} c(\mathbf{x}_t, \mathbf{u}_t)$$

Subject to:

$$\mathbf{x}_{t+1} = \mathbf{F}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \mathbf{f}_t$$

LQR with Dynamic Programming

- **Cost optimization:** Minimize $\sum_{t=0}^{T-1} c(x_t, u_t)$ iteratively (Bellman-style).
- Expressed via **Value** (V) and **Action-Value** (Q) functions:

Q-Function Definition

$$Q_t = \text{Current cost} + V(\text{Next step})$$

$$Q(x_{T-1}, u_{T-1}) = \text{const} + \frac{1}{2}\begin{bmatrix} x_{T-1} \\ u_{T-1} \end{bmatrix}^T C_{T-1} \begin{bmatrix} x_{T-1} \\ u_{T-1} \end{bmatrix} + \begin{bmatrix} x_{T-1} \\ u_{T-1} \end{bmatrix}^T c_{T-1} + V(x_T)$$

- **Key idea:** Solve backward from $T-1$ to 0 using dynamic programming.

LQR Backward Pass

- **Iterate from last step to first step**
  1. Substitute Q_{T-1} with V_T expressed in terms of x_T (where $x_T = Ax_{T-1} + Bu_{T-1}$).
  2. Optimize Q by solving $\nabla Q_{T-1} = 0$.
  3. Compute feedback gains $\mathbf{K}_{T-1}$ and $\mathbf{k}_{T-1}$ to get control:

$$u_{T-1} = \mathbf{K}_{T-1}x_{T-1} + \mathbf{k}_{T-1}$$

  4. Substitute u_{T-1} back into Q_{T-1} to obtain V_{T-1} (used for step $T-2$).

iLQR Backward Pass: Algorithm Overview

Initialization (Terminal Step)

$$V_x = Q_{\text{terminal}}(x_T - x_{\text{target}}), \quad V_{xx} = Q_{\text{terminal}}$$

Action-Value Function $Q(x_t, u_t)$

$$Q(x_t, u_t) = \underbrace{\frac{1}{2}\begin{bmatrix} x_t \\ u_t \end{bmatrix}^T \begin{bmatrix} Q_{xx} & Q_{xu} \\ Q_{ux} & Q_{uu} \end{bmatrix} \begin{bmatrix} x_t \\ u_t \end{bmatrix}}_{\text{Quadratic term}} + \underbrace{\begin{bmatrix} Q_x & Q_u \end{bmatrix} \begin{bmatrix} x_t \\ u_t \end{bmatrix}}_{\text{Linear term}} + \text{const}$$

Value Function $V(x_t)$ (After Optimization)

$$V(x_t) = \underbrace{\frac{1}{2} x_t^T V_{xx} x_t}_{\text{Quadratic term}} + \underbrace{V_x^T x_t}_{\text{Linear term}} + \text{const}$$

LQR Backward Pass: Optimization

Key Quantities

$$Q_x = A_t^\top V_x + C_x, \quad Q_u = B_t^\top V_x + Ru_t$$
$$Q_{xx} = A_t^\top V_{xx} A_t + C_{xx}, \quad Q_{uu} = B_t^\top V_{xx} B_t + R$$
$$Q_{ux} = B_t^\top V_{xx} A_t + C_{ux}$$

Control Update

Solve $\nabla_u Q = 0$ to get feedback law:

$$k_t = -Q_{uu}^{-1} Q_u, \quad K_t = -Q_{uu}^{-1} Q_{ux}$$

Value Function Update

$$V_x \leftarrow Q_x + K_t^\top Q_{uu} k_t + K_t^\top Q_u$$
$$V_{xx} \leftarrow Q_{xx} + K_t^\top Q_{uu} K_t + K_t^\top Q_{ux} + Q_{ux}^\top K_t$$

- **Output:** Gain matrices K_t and offsets k_t for all t.

iLQR Overview

- Instead of pretending your nonlinear system is globally linear, you locally linearize it around a "guess" trajectory, solve an LQR problem there, then update your guess and repeat.

iLQR Mathematics: Nonlinear System Approximation

Nonlinear System Assumptions

- Dynamics: $x_{t+1} = f(x_t, u_t)$
- Cost: $c_t = \ell(x_t, u_t)$

Local Approximations (1st/2nd Order)

- **Dynamics (1st Order):**

$$\delta x_{t+1} = \underbrace{\nabla_{x_t} f(\hat{x}_t, \hat{u}_t)}_{A_t} \delta x_t + \underbrace{\nabla_{u_t} f(\hat{x}_t, \hat{u}_t)}_{B_t} \delta u_t$$

  where $\delta x_t = x_t - \hat{x}_t$, $\delta u_t = u_t - \hat{u}_t$

- **Cost (2nd Order):**

$$\delta c_t = \underbrace{\nabla \ell(\hat{x}_t, \hat{u}_t)}_{c_t}^{T} \begin{bmatrix} \delta x_t \\ \delta u_t \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \delta x_t \\ \delta u_t \end{bmatrix}^T \underbrace{\nabla^2 \ell(\hat{x}_t, \hat{u}_t)}_{C_t} \begin{bmatrix} \delta x_t \\ \delta u_t \end{bmatrix}$$

Iterative Linear Quadratic Regulator (iLQR)

Motivation:

- LQR assumes linear dynamics; not suitable for nonlinear systems.
- iLQR extends LQR to handle nonlinear dynamics through iteration.

Approach:

1. Initialize with a nominal trajectory.
2. Linearize dynamics and quadratize cost around the nominal trajectory.
3. Apply LQR to compute control updates.
4. Update nominal trajectory and repeat until convergence.

Iterative LQR (iLQR) Algorithm

WHILE not converged

1. Linearization (Current Trajectory $\hat{x}_t, \hat{u}_t$):

- Dynamics Jacobian:

$$F_t = \nabla_{x_t,u_t} f(\hat{x}_t, \hat{u}_t) = \begin{bmatrix} A_t & B_t \end{bmatrix}$$

- Cost gradients/Hessians:

$$c_t = \nabla_{x_t,u_t} c(\hat{x}_t, \hat{u}_t), \quad C_t = \nabla^2_{x_t,u_t} c(\hat{x}_t, \hat{u}_t) = \begin{bmatrix} Q_{xx} & Q_{xu} \\ Q_{ux} & Q_{uu} \end{bmatrix}$$

2. Backward Pass (LQR on Deviations):

- Solve for $\delta u_t = K_t \delta x_t + k_t$ where:

$$\delta x_t = x_t - \hat{x}_t, \quad \delta u_t = u_t - \hat{u}_t$$

3. Forward Pass (Nonlinear Rollout):

- Apply control with feedback:

$$u_t = K_t(x_t - \hat{x}_t) + k_t + \hat{u}_t$$

- Update nominal trajectory $(\hat{x}_t, \hat{u}_t)$ from rollout.

What If Dynamics Are Unknown?

- Estimate dynamics using a neural network.
- Train NN with randomly collected data.
- Improve the model iteratively as more data is collected.

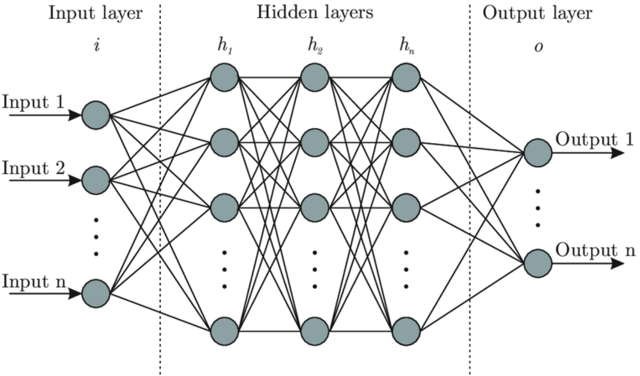
Model-Based RL with NN based Learning

Proposed Improvement:

- **Data Collection:** Generate dataset from iLQR-based rollouts, storing transitions (s_t, a_t, r_t, s_{t+1}).
- **Neural Network for System Dynamics:** Train NN to predict next state: $s_{t+1} = f_\theta(s_t, a_t)$.
- **Control Optimization:** Use learned NN in iLQR instead of true model.

- A Neural Network (NN) acts as a function approximator here.
- It maps inputs **x** to outputs **y** via layers of *neurons*.

Basic Structure:



- **Input Layer**: Receives state/control vectors.
- **Hidden Layers**: Perform successive linear transforms + nonlinear

Training Neural Networks

1. **Data Collection**: Gather state–action–next-state tuples (x_t, u_t, x_{t+1})t.
2. **Loss Function**: Measure prediction error, e.g.

$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N} \|x_{t+1}^{(i)} - \hat{f}_\theta(x_t^{(i)}, u_t^{(i)})\|^2.$$

3. **Optimization**: Use backpropagation + Optimization (like SGD) to adjust weights θ.
4. **Regularization**: Techniques like dropout, weight decay to avoid overfitting.

Why Use Neural Networks to Estimate Dynamics for iLQR

- **Expressiveness**: Can approximate complex, nonlinear system dynamics better than fixed polynomial functions.
- **Data Efficiency**: Learns from observed transitions, capturing unmodeled effects (friction, aerodynamic drag).
- **Analytic Derivatives**: Modern frameworks provide $\partial \hat{f} / \partial x$ and $\partial \hat{f} / \partial u$ via autodiff for local linearization.
- **Modular Integration**: Use the learned model in iLQR algorithm:
    - *Forward pass*: simulate with $\hat{f}_\theta$.
    - *Backward pass*: compute Jacobians A_t, B_t for LQR update.

Implementation: Solving Gym Classic Control Tasks

- Evaluate our approach with implementation on Mountain Car.

Mountain Car Continuos Environment

Problem Description

- **Goal:** Drive underpowered car up steep mountain
- **Challenge:** Must build momentum to climb
- **State:** Position $x \in [-1.2, 0.6]$, Velocity $\dot{x} \in [-0.07, 0.07]$
- **Actions:** $a \in [-1, 1]$

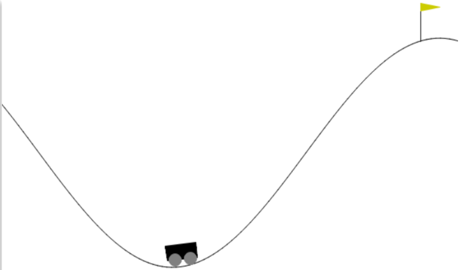
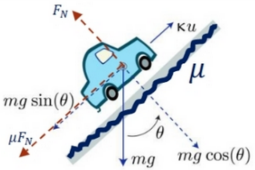


Illustration of MountainCar environment

MountainCar Dynamics & Gym Simplifications



Real-World Physics (x-direction)

$$\sum F_x = \underbrace{F_{\text{engine}}}_{\text{control}} - \underbrace{mg\sin\theta}_{\text{gravity}} - \underbrace{\mu mg\cos\theta}_{\text{friction}}$$

Mountain Car Task

Dynamics

- Physics model:

$$\dot{x}_{t+1} = \dot{x}_t + 0.001a_t - 0.0025\cos(3x_t)$$

$$x_{t+1} = x_t + \dot{x}_{t+1}$$

- **Reward:** -1 per timestep until goal ($x \geq 0.5$)
- **Termination:** Reach flag or 200 steps

- Nonlinear dynamics make LQR inapplicable.
- Applied iLQR to control task.

Gym Implementation

Gym Environment Assumptions

- **Friction ignored:** $\mu = 0$
- **Slope relation:** $\theta = 3x_t$ $\quad (y = \sin(3x))$
- **Simplified gravity term:** $mg \sin\theta \to 0.0025 \cos(3x_t)$
- **Engine acceleration:** $F_{\text{engine}}/m \to \text{action} \times 0.001$

$$\boxed{\begin{aligned} v_{t+1} &= v_t + \underbrace{\text{action} \times 0.001}_{\text{engine}} - \underbrace{0.0025 \cos(3x_t)}_{\text{gravity}} \\ x_{t+1} &= x_t + v_{t+1} \end{aligned}}$$

Gravity term $\cos(3x_t)$ creates a complex energy landscape:

Harder to climb as x increases

Attempt 1: LQR Implementation on MountainCar

Approach

- Implements classic Linear Quadratic Regulator (LQR) control.
- Linearizes the nonlinear dynamics at each timestep:

$$x_{t+1} \approx Ax_t + Bu_t$$

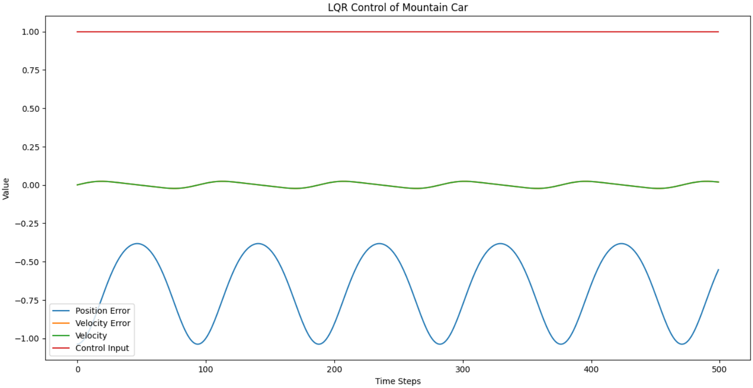
- Solves the discrete-time Riccati equation for feedback gain K.
- Control input:

$$u_t = -Kx_t$$

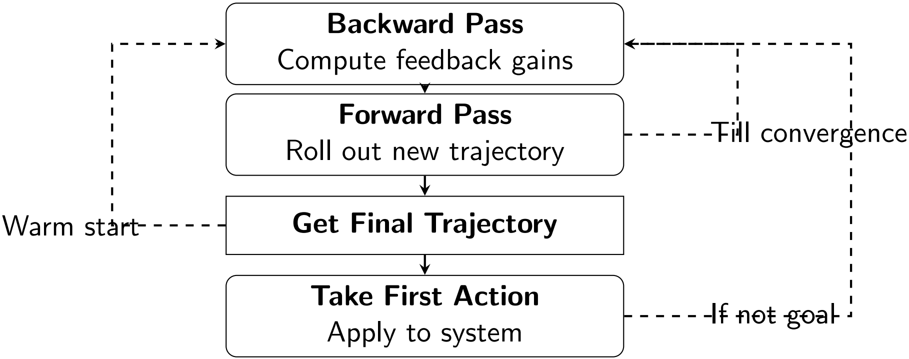
Limitations

- Assumes linear dynamics — MountainCar is inherently nonlinear.
- No consideration of $\cos(3x)$ term during linearization.
- Controller performs poorly near steep slopes.

Result


LQR Control of Mountain Car
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Value
0
100
200
300
400
500
Time Steps
Position Error
Velocity Error
Velocity
Control Input

Iterative Control Loop Overview

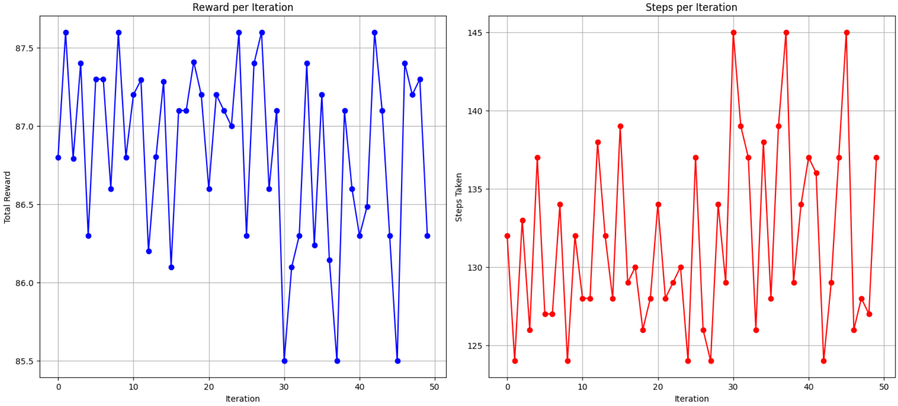


Mountain Car implementation details

Hyperparameters:

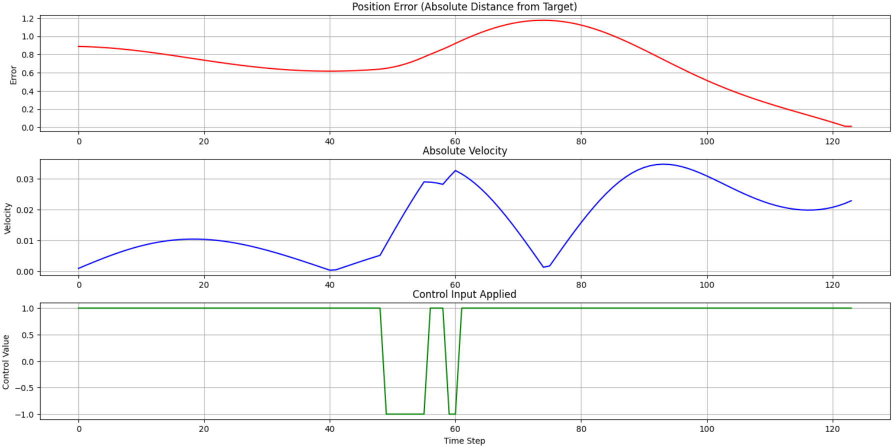
- Horizon: 40
- Constant number of iterations of LQR: 5
- $Q_{terminal} = \begin{bmatrix} 10 & 0 \\ 0 & 0 \end{bmatrix}$
- $R = [0.1]$
  The known dynamics is provided.
- Backward pass uses linearised dynamics A, B
- Forward pass uses actual non linear dynamics

Results: iLQR with Known Model



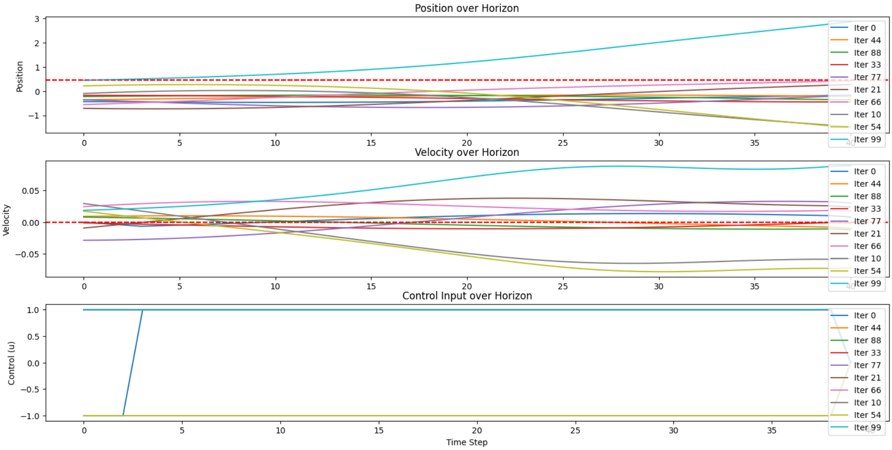
Rewards obtained and steps needed to reach goal over 50 iterations.

Results: iLQR with Known Model



Position error, Velocity and control action in one run

Results: iLQR with Known Model



10 different LQR optimised trajectories within the optimisation loop

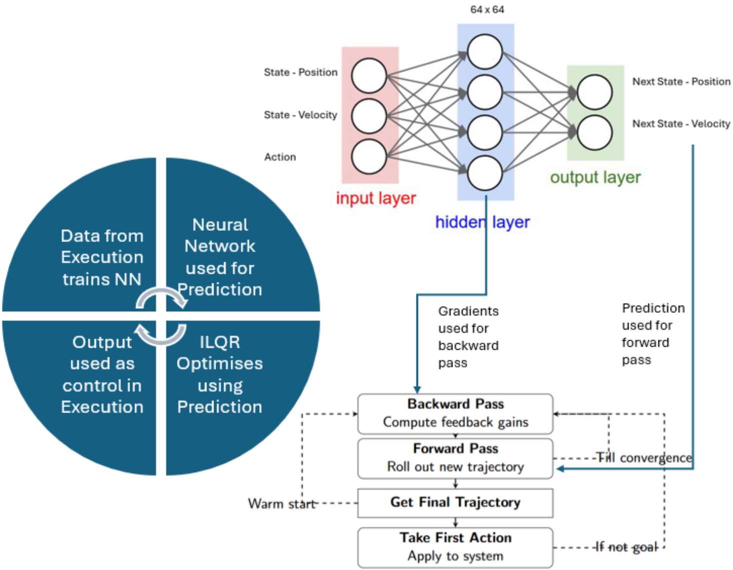
What if Dynamics are not known?

Using a Neural Network to predict dyanmics

What if Dynamics are not known?

- Use a feedforward neural network to approximate unknown system dynamics.
- Architecture:
  - **Input:** State (x) and action (u)
  - **Network:** Two hidden layers with 64 ReLU units each
    Input $\rightarrow$ [64 neurons] $\rightarrow$ [64 neurons] $\rightarrow$ Output
  - **Output:** Next state prediction (x_{t+1})
- Model: `MLPRegressor` from `sklearn`
- Trained online using data collected from environment rollouts
- Mini-batch gradient descent using the Adam optimizer
- Initially random exploration phase helps initilaise NN.

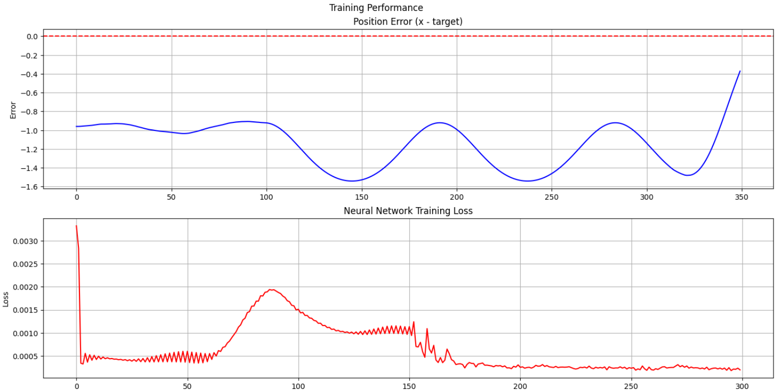
iLQR with NN implementation



Implementation Details for Neural Network

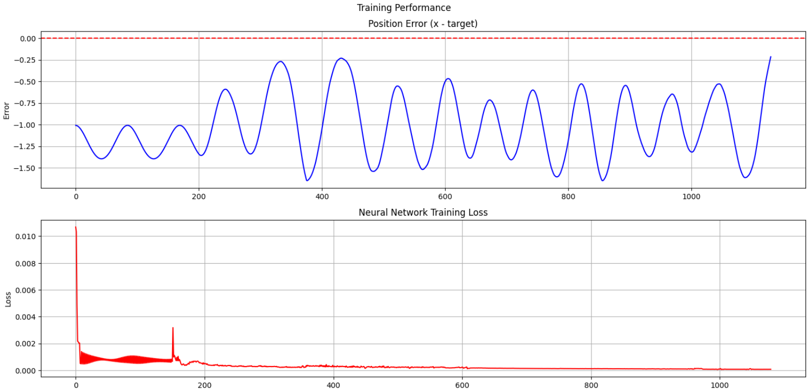
1. **Random Exploration Phase** to initialize the network:
   - Approximately 200 steps of random exploration.
   - To encourage exploration across the state space: Apply action $= +1$ if position > 0, else -1.
2. **Planning Horizon**: Set to 50 time steps.
3. **Training Schedule**: The neural network is updated every 50 steps.
4. **Gradient Estimation for Linearization**: Finite differences used to estimate Jacobians:
   - $A[:, i] = \frac{f(x+\varepsilon_i, u) - f(x, u)}{\varepsilon}$
   - Approximates $\partial f / \partial x$ and $\partial f / \partial u$ for use in iLQR backward pass.

Results: iLQR with NN



Results from partially integrated iLQr with gradients still obtained from known model

Results: iLQR with NN

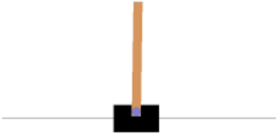


Results from model in which known model completely removed

Swing-Up CartPole Environment

Problem Description

- **Goal:** Swing and balance the pole in upright position on a moving cart.
- **Challenge:** Starts in downward position; must swing up and stabilize.
- **State:** Position x, Velocity $\dot{x}$, Angle θ, Angular Velocity $\dot{\theta}$
- **Actions:** Continuous force a
in [-10, 10]



Gym environment

Note: A harder problem than Mountain Car due to underactuation and nonlinear dynamics.

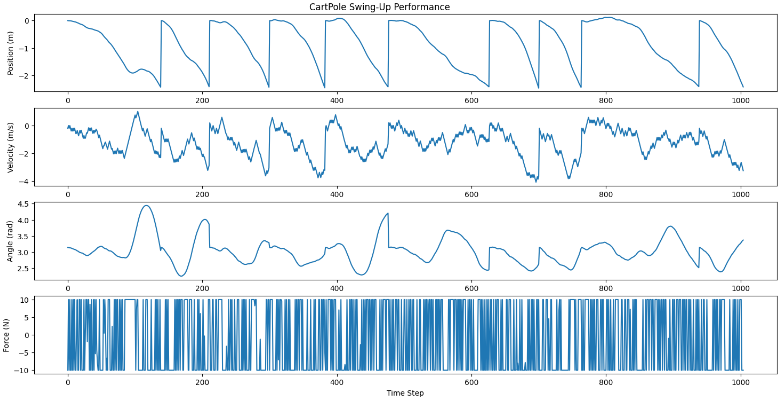
Modifications to Classic CartPole for Swing-Up Task

- **Initial Condition:**
  - Pole starts hanging downward — requires active swing-up to reach upright.
- **Reward Function Modified:**
  - Reward is based on `upright = cos()`, giving:
    - Maximum reward when pole is upright ($\theta = 0$)
    - Zero reward when hanging down ($\theta = \pi$)
- **No Early Termination:**
  - Classic CartPole ends episode when angle deviates too much.
  - The wrapper ensures episode ends only when cart moves beyond threshold.
- **Continuos Actions and Force Scaling:**
  - Supports continuous control input.
  - Action made continuoss clipped to range: $[-10, 10]$

Adapting the Algorithm for Swing-Up Pendulum

- **Modified Neural Network Architecture:**
  - Input dimension increased to 4: $(x, \dot{x}, \theta, \dot{\theta})$
  - Network expanded to hidden layer with **256 neurons**
- **Increased Exploration Steps:**
  - More episodes (100) run with random actions for better coverage of state space
- **Extended iLQR Planning Horizon:**
  - Longer horizon used to account for the swing-up and balance phases
- **More iLQR Optimization Iterations:**
  - Increased inner loop iterations for better trajectory refinement

Results: ILQR + NN base implementation



Very short runs, sub optimal result

Core Technical Improvements

1. **Experience Replay Buffer**
   - Stores 100K transitions
   - Batched training (64 samples)
   - Breaks temporal correlation

2. **PyTorch Dynamics Model**
   - Automatic differentiation
   - GPU acceleration support
   - Precise Jacobians via autograd

3. **Residual Network**
   - Output og NN is difference from the previous state [Delta x].
   - Helps converrge faster and give more smoother dynamics.

4. Enhanced Architecture

- 256 hidden units
- ReLU activation
- ADAM optimizer
- MSE loss

5. Training Protocol

1. Collect episode data
2. Train model (50 epochs)
3. Optimize with iLQR
4. Repeat

6. Adaptive Line Search

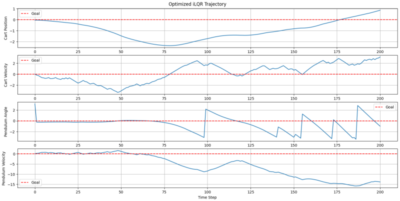
$$u_t^{\text{new}} = u_t + \alpha k_t + K_t(x_t - x_t^{\text{ref}})$$

- Guarantees cost reduction
- Backtracking with $\alpha = 0.5 \rightarrow 0.7$

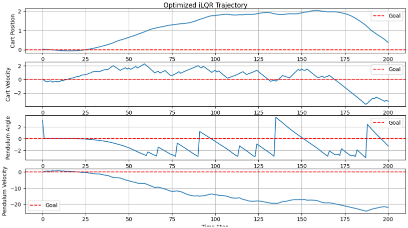
7. Regularized Cost Function

$$J = \underbrace{10\theta^2}_{\text{Upright}} + \underbrace{0.1x^2}_{\text{Position}} + \underbrace{0.1v^2}_{\dot{x}} + \underbrace{0.1x^2}_{\dot{\theta}}$$

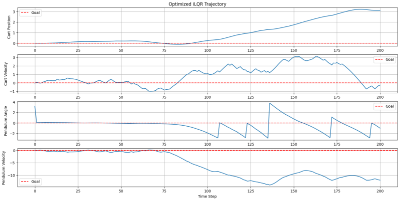
Comparison of Results from Different Runs



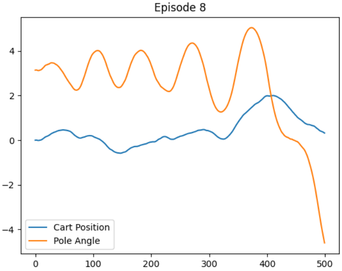
Run 1



Run 2



Run 3



Run 4

Conclusion

- Successfully implemented **iLQR** to solve classic control tasks in the Gymnasium framework.
- Demonstrated effective use of a **neural network-based dynamics model** to handle unknown system dynamics.
- Applied the method to both **Mountain Car** and the more challenging **Swing-Up CartPole** environment.
- Incorporated key adaptations: deeper neural networks, extended horizons, and enhanced iLQR optimization to improve performance on complex tasks.

Key Takeaway

iLQR combined with learned dynamics can serve as a powerful model-based control strategy even in challenging nonlinear settings.

Future Work

- Learn unknown parameters like mass and gravity explicitly.
- Extend to more complex environments.
- Improve sample efficiency of the training loop.

Thank You!

Questions and Discussion Welcome.