Homework 2: Imitation Learning of Robot Policies

RPL Fall 2025

Outline

- **Problem 1:** Diffusion model vs. Regression models
- **Problem 2:** Flow matching
- Grading
- Environment
- Submission
- Policy

1 Problem: Diffusion Model vs. Regression Model

Imitation learning has been commonly adopted for training robot policies [**?**, **?**, **?**]. Despite its success, to learn effective policies, imitation learning methods have to tackle multiple challenges, including distributional drift [**?**], causal confusion, and most importantly, multi-modality behaviors in demonstrations.

Take the task of holding a mug for example: some users may grasp the handle, some may grasp the body, and others may hold the base of a mug. When developing a robot policy to perform such tasks, you need to know what are the suitable frameworks for modeling multi-modality behaviors.

In this assignment, you will implement a probabilistic model—diffusion models, and compare it against the deterministic counterpart—regression models, on two robot manipulation benchmarks (Push-T and CALVIN). You will observe if the probabilistic model outperforms its deterministic counterpart when the training demonstrations are multi-modality.

References

1. RT-1: Robotics Transformer for Real-World Control at Scale. Brohan et al. arXiv 2022.
2. RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control. Brohan et al. arXiv 2023.
3. OpenVLA: An Open-Source Vision-Language-Action Model. Kim et al. arXiv 2024.
4. End to End Learning for Self-Driving Cars. Bojarski et al. CVPR 2016.

2 Model Architecture

In this assignment, we will use the same convolutional U-Net as Diffusion Policy [?] (see the model script). The model conditions on the current and previous visual observations, takes input as noisy estimate of a robot's end-effector poses, and outputs the noise imposed on the end-effector poses.

The conditional visual observations are encoded to separate 1D feature vectors, using a ResNet18 encoder, and injected to the UNet with Adaptive Layer Norm [?]. The input noisy pose estimates is a tensor of shape `[batch_size, traj_len, dim]`. The UNet consists of multiple 1D Convolutional layers, which downsample the temporal length of the inputs, creating an information bottleneck, and then upsample the temporal length back to the same resolution.

References

1. Diffusion Policy. Chi et al. RSS 2023.
2. Scalable Diffusion Models with Transformers. Peebles and Xie. ICCV 2023.

2.1 Model Architecture Details

The model architecture consists of the following components:

2.1.1 ConditionalResidualBlock1D Class

The `forward` function of the model class takes four input arguments:

- `sample (B, T, input_dim)`: indicates the noisy estimates of a robot's end-effector poses
- `timestep (B,)` or `int`: indicates the diffusion timestep
- `local_cond (B, T, local_cond_dim)`: indicates some local conditional inputs. In the current code base, `local_cond` is always set to `None`.
- `global_cond (B, global_cond_dim)`: indicates some global conditional inputs. In the current code base, `global_cond` is the aggregated feature vector of the current and previous visual observation.

2.2 Push-T Benchmark

Push-T is a 2D manipulation benchmark, where the goal is to push the T-shape object (colored in gray) to the target location (colored in green). The blue dot represents the current location of the robot's end-effector.

3 Problem 1: Implementation Tasks

3.1 Part 1: Diffusion Model Implementation

In part 1, you will implement several key components in the `DiffusionUnetImagePolicy` class located in `diffusion_policy/policy/diffusion_unet_image_policy.py`. Your task is to complete the following TODOs in the code:

3.1.1 Training

Implement the diffusion forward process in the `compute_loss` method by:

1. Sample the noise and add it to the trajectory
2. Sample random timesteps and adjust noise magnitude accordingly
3. Predict the noise residual
4. Calculate the MSE loss between the prediction and target

3.1.2 Inference

Implement the trajectory update in `conditional_sample` method:

1. Predict the model output during the inference process
2. Compute the previous image from the current image during the diffusion process

3.2 Part 2: Regression Model Implementation

In part 2, you will implement several key components in the `RegressionUnetImagePolicy` class located in `diffusion_policy/policy/regression_unet_image_policy.py`. Your task is to complete the following TODOs in the code:

3.2.1 Training

Implement the action prediction and loss computation in the `compute_loss` method by:

1. Set the timestep to 0 and forward-pass the embeddings through the model to get the prediction for the regression task.
2. Compute the loss directly against the ground truth trajectory to train the model effectively.

3.2.2 Inference

Implement the action prediction in the `predict_action` method by:

1. Predict the action by setting the timestep to 0 and forward-passing the embeddings through the model to obtain the predicted actions.

4 Problem 2: Flow Matching

Flow matching is a state-of-the-art technique to map between two distributions. By mapping two distributions, you can generate images from normal distribution like stable diffusion 3 or generate robot policy just like diffusion models do. However, as the shortage of diffusion models, original version of flow matching has to use larger steps to generate high quality samples. Recent works often focus on how to lower the steps while remain the quality.

In this part, you will implement the flow matching to map the source distribution to the target distribution. You have to finish the TODO part for four stages. For better understanding, we ask you to visualize the results by `plot()` in `visualization.py`. We provided a metric to compute Wasserstein distance to measure the performance of the methods and it is optional.

For simplicity, we set x_0 to 3000 fixed points and x_1 to 150 fixed points. Then we randomly augment x_1 by 150 fixed points until the number of x_1 becomes 3000. In reality, you should sample x_0 from known distribution and sample x_1 from dataset.

4.1 Stage 1: Beauty of Flow Matching

If we want to generate a target distribution $p(x_1)$, we can start from a source distribution $p(x_0)$, which is often set as $\mathcal{N}(0, I)$. Our goal is to train a model to predict the instantaneous velocity $v_\theta(x_t, t)$, so that we can use the concept of ordinary differential equation to walk through the path from $p(x_0)$ to $p(x_1)$.

Note that:

$$x_t = t \cdot x_1 + (1 - t) \cdot x_0 \tag{1}$$

4.1.1 Loss Function

To train a model, we have to define a loss function. We know that if we want to predict instantaneous velocity, we have to input x_t and t to the model. For simplicity, we can let $x_t = t \cdot x_1 + (1 - t) \cdot x_0$. And we can let the ground truth velocity $v = x_1 - x_0$.

$\pi(x_0, x_1)$ is joint distribution of $p(x_0)$ and $p(x_1)$. We let $\pi(x_0, x_1) = p(x_0)p(x_1)$. As such, we can sample (x_0, x_1) by $x_0 \sim p(x_0)$, $x_1 \sim p(x_1)$.

$$\mathcal{L}_{\text{FM}}(\theta) = \mathbb{E}_{t\sim U(0,1)}\mathbb{E}_{(x_0,x_1)\sim\pi(x_0,x_1)}\left[\left\| \underbrace{v_\theta(x_t, t)}_{\text{predicted velocity}} - \underbrace{(x_1 - x_0)}_{\text{ground truth velocity}} \right\|^2\right] \tag{2}$$

4.1.2 Coupling Strategies

From the previous formulation, we realize that there are several things we can change:

Coupling Strategies $\pi(x_0, x_1)$ Here $\pi(x_0, x_1)$ is joint distribution of $p(x_0)$ and $p(x_1)$

- **Stage 1:** Flow Matching assumes $\pi(x_0, x_1) = p(x_0)p(x_1)$.
- **Stage 2:** Optimal Coupling by considering optimal transport
- **Stage 3:** Reflow by using x_1 output by model using Algorithm 2

Instantaneous velocity prediction loss

$$\left\| \underbrace{v_\theta(x_t, t)}_{\text{predicted velocity}} - \underbrace{(x_1 - x_0)}_{\text{ground truth velocity}} \right\|^2 \tag{3}$$

Stage 4: Meanflow (Any other ideas?)

Algorithm 1 Flow Matching Algorithm

```
Require: Dataset of paired samples $(x_0, x_1) \sim \pi(x_0, x_1)$, model $v_\theta(x_t, t)$, learning rate $\eta$, number
    of iterations $T$
Ensure: Trained parameters $\theta$
 1: Initialize parameters $\theta$
 2: for $i = 1$ to $T$ do
 3:     Sample a minibatch of pairs $(x_0, x_1)$ from the dataset
 4:     Sample a random time $t \sim \mathcal{U}(0, 1)$
 5:     Compute the interpolated state: $x_t = (1 - t)x_0 + tx_1$
 6:     Define the target velocity: $v = x_1 - x_0$
 7:     Compute the predicted velocity from the model: $v_{\text{pred}} = v_\theta(x_t, t)$
 8:     Evaluate the flow-matching loss:
          $\mathcal{L}_{\text{FM}}(\theta) = \mathbb{E}_{(x_0,x_1),t}[\|v - v_{\text{pred}}\|^2]$
 9:     Update parameters: $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_{\text{FM}}(\theta)$
10: end for
11: return Trained parameters $\theta$
```

Algorithm 2 Sampling with Trained Flow Field

```
Require: Trained model $v_\theta(x_t, t)$, initial distribution $p(x_0)$, number of timesteps $K$
Ensure: Sample $x_1$ approximating the target distribution $p(x_1)$
 1: Sample initial point $x_0 \sim p_0(x)$
 2: Set time step $\Delta t = 1/K$, and initialize $x \leftarrow x_0$
 3: for $k = 0$ to $K - 1$ do
 4:     Compute the current time $t_k = k/K$
 5:     Estimate velocity $v_k = v_\theta(x, t_k)$
 6:     Update the state using Euler integration: $x \leftarrow x + \Delta t \cdot v_k$
 7: end for
 8: return $x_1 = x$
```

4.1.3 Algorithm 1: Flow Matching Algorithm

4.1.4 Algorithm 2: Sampling with Trained Flow Field

4.1.5 Tasks for Stage 1

1. Complete the Stage 1 TODO parts
2. Plot the result with steps = 1000
3. Plot the result with steps = 1
4. Analyze the difference between two results and try to answer the reason

4.2 Stage 2: Optimal Coupling

Instead of using random coupling between x_0 and x_1 as in flow matching, a more natural choice is using optimal coupling. Optimal coupling is a coupling which can minimize the Wasserstein distance between two distributions. Also, it has a property that any two (x_0, x_1) coupling's interpolation x_t will not intersect with each other. Thus, the instantaneous velocity has the only direction.

By optimal coupling, we expect that the instantaneous velocity field will not be high curvature. You can use `sample_plan()` in `optimal_transport.py` to get optimal coupling pair. The rest algorithm is same as flow matching.

4.2.1 Tasks for Stage 2

1. Complete the Stage 2 TODO parts
2. Plot the result with steps = 1000
3. Plot the result with steps = 1
4. Is it better? Are the couplings we find actually global optimal? If not, what happens when the couplings are actually local optimal?

4.3 Stage 3: Alternative - Reflow

There are some shortages with optimal coupling in real world to scale up. Consequently, we can set the coupling to $(x_0, \text{model}(x_0))$ instead of (x_0, x_1). This method is proved to let the flow more straight in the paper. The rest algorithm is same as flow matching. $\text{model}(x_0)$ means Algorithm 2.

Rectified Flow Illustration:

(a) **Linear interpolation:** Shows the initial data input $(X_0, X_1) \sim \pi(x_0, x_1)$ where $X_t = tX_1 + (1-t)X_0$.

(b) **The Rectified Flow Z_t by (X_0, X_1):** Trajectories are "rewired" at intersection points to avoid crossing.

(c) **Linear interpolation of endpoints:** Shows the paths between end points (Z_0, Z_1) of flow Z_t, defined by $Z_t = tZ_1 + (1-t)Z_0$.

(d) **Rectified flow Z_t':** The flow induced from (Z_0, Z_1) follows straight paths.

Note that Z_0 is X_0 and Z_1 is the output of model at $t = 1$ traveling from X_0.

4.3.1 Tasks for Stage 3

1. Complete the Stage 3 TODO parts
2. Plot the result with steps = 1000
3. Plot the result with steps = 1
4. Analyze the results. What is the shortage of Reflow?

4.4 Stage 4: Alternative - MeanFlow

Instead of training model to predict instantaneous velocity, we can train the model to predict average velocity. However, how to derive average velocity from instantaneous velocity is a difficult work and is proved in the paper. We encourage you to see them in the paper. By this method, we can improve the performance for one step generation.

$$u(x_t, r, t) = \frac{1}{t-r}\int_r^t v(x_s, s)\, ds \tag{4}$$

where (r, t) is (start time, end time).

Note: While the instantaneous velocity v determines the tangent direction of the path, the average velocity $u(z, r, t)$ is generally not aligned with v. The average velocity is aligned with the displacement, which is $(t - r)u(z, r, t)$. The field $u(z, r, t)$ is conditioned on both r and t.

4.4.1 Algorithm 3: MeanFlow Training

Algorithm 3 MeanFlow Training

```
1: Note: In PyTorch and JAX, jvp returns the function output and JVP.
2: # fn(z,r,t): function to predict $u$
3: # x: training batch
4: $(t, r) \leftarrow$ sample_t_r()                    ▷ $(t, r)$ is (start time, end time)
5: $e \leftarrow$ random_like(x)                      ▷ $e$ is $x_0$, $x$ is $x_1$, and $z$ is $x_t$
6: $z \leftarrow (1 - t) \cdot x + t \cdot e$         ▷ when $t = 1$, $z = e$
7: $v \leftarrow e - x$
8: $u, \frac{du}{dt} \leftarrow$ jvp(fn, (z,r,t), (v,0,1))
9: $u_{\text{tgt}} \leftarrow v - (t - r) \cdot \frac{du}{dt}$
10: error $\leftarrow u -$ stopgrad$(u_{\text{tgt}})$
11: loss $\leftarrow$ mean(error)
```

4.4.2 Algorithm 4: MeanFlow 1-step Sampling

Algorithm 4 MeanFlow: 1-step Sampling

```
1: $e \leftarrow$ randn(x_shape)
2: $x \leftarrow e -$ fn$(e, r = 0, t = 1)$
```

4.4.3 Tasks for Stage 4

1. Complete the Stage 4 TODO parts
2. Plot the result with steps = 1000
3. Plot the result with steps = 1
4. Analyze the results

5 Grading

5.1 Code for Problem (10%)

5.2 report.md (65%)

5.2.1 How to run your code, requirements, etc.

5.2.2 Push-T Benchmark (30%)

- Show the success rates of the diffusion and regression model
- Show the evaluation rollout videos (gif) of the diffusion model and regression model with 2 different seeds when evaluate

5.2.3 Flow Matching (35%)

- Show the generated trajectory using `plot()` for each stage
- Analysis and answer the questions for each stage
- You should answer the questions including the results you observed and the reasons you guessed

5.3 Advanced Challenge (25% each, capped at 100%)

Completing the advanced challenge will earn the following points. Scores above 100 points will be capped at 100.

5.3.1 Challenge 1: Creative Ideas for Target Distribution (25%)

Can you think of any creative idea to better fit the target distribution in one or few steps?

- Show the results and Wasserstein distance and tell us how you do it
- We will grade based on the creativity and performance

5.3.2 Challenge 2: Implement Flow Policy for Push-T (25%)

Try to solve Push-T by flow matching:

- Flow matching, Reflow, MeanFlow, etc. Choose any flow-based method you want.
- Compare the success rate and inference time or NFE (how many times you call forward to sample one action) with diffusion model

6 Submission

Your GitHub repository `RPL-Fall-2025/hw2-{GitHub_ID}` should include all files:

- All codes
- `report.md` (include result gif and trajectory images)
- Don't upload data, cache files, etc.

7 File Structure

```
rpl-fall-2025-hw2-RPL-HW2-goog-msft-fb-nflx-nvda-aapl
          Problem1
                diffusion_policy
                      codecs
                      common
                      config
                      dataset
                      env
                      env_runner
                      gym_util
                      model
                      policy
                      workspace
                envs
                eval.py
                licenses
                scripts
                setup.py
                train.py
          Problem2
                data.py
                main.py
                model.py
                optimal_transport.py
                pipeline.py
                visualize.py
          README.md
```

8 Hardware Information

8.1 System Information

- **OS:** Ubuntu 22.04.5 LTS
- **Kernel:** Linux 5.15.0-164-generic x86_64
- **GPU:** 8x NVIDIA H200 (143771MiB VRAM each)
- **Driver:** NVIDIA-SMI 550.163.01
- **CUDA Version:** 12.4

9 Packages

Please follow the README to install the required packages and provide usage instructions in the `README.md`.