Combining Imitation Learning with Diffusion Processes on Robot Manipulator

Diffusion Processes

Diffusion processes can be modeled by Stochastic Differential Equations (SDEs) as follows:

$$dX_t = \mu(X_t, t)dt + \sigma(X_t, t)dW_t \tag{1}$$

where:

- X_t is the state of the process at time t.
- $\mu(X_t, t)$ represents the drift term.
- $\sigma(X_t, t)$ represents the diffusion term.
- W_t represents a Wiener process (or Brownian motion).

Imitation Learning

The objective function in imitation learning can be expressed as:

$$\min_{\pi} \mathbb{E}_{(s,a^*)\sim D}[-\log \pi(a^*|s)] \tag{2}$$

where:

- D is a dataset of state-action pairs (s, a^*).
- a^* is the action taken by the expert in state s.
- In this case we created expertise Dataset by using Inverse Kinematics Solver (IK-Solver).

Combining Diffusion Processes with Imitation Learning

$$\min_{\pi} \mathbb{E}_{(s,a^*)\sim D, X_t\sim \text{SDE}}\left[-\log \pi(a^*|s) + \lambda \cdot L(X_t, \pi(s))\right] \tag{3}$$

where:

- $L(X_t, \pi(s))$ represents a loss term under the dynamics X_t.
- λ is a regularization parameter.

Neural Network Model

Given a desired end-effector position $x \in \mathbb{R}^3$, the network computes the joint angles $y \in \mathbb{R}^7$ through a series of transformations.

1. **Layer 1:** $h_1 = \text{ReLU}(W_1 x + b_1)$
2. **Layer 2:** $h_2 = \text{ReLU}(W_2 h_1 + b_2)$
3. **Layer 3:** $h_3 = \text{ReLU}(W_3 h_2 + b_3)$
4. **Output Layer:** $y = W_4 h_3 + b_4$

Where:

- W_i and b_i are the weights and biases of the i-th layer.
- $\text{ReLU}(z) = \max(0, z)$ is the Rectified Linear Unit activation function.

Loss Function

Minimizing the difference between the predicted joint angles and the true joint angles. The loss function used is the Mean Squared Error (MSE), given by:

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \| f(x^{(i)}; \theta) - y^{(i)} \|^2$$

Where:

- N is the number of samples in the dataset.
- $x^{(i)}$ is the i-th desired end-effector position.
- $y^{(i)}$ is the true joint angles for the i-th sample.
- $\| \cdot \|$ denotes the Euclidean norm.

Optimization

The training process seeks to find the optimal parameters θ^* that minimize the loss function $L(\theta)$. This is typically done using gradient-based optimization methods, such as Adam. The update rule for Adam at each iteration t for each parameter θ is as follows:

1. Compute gradients: $g_t = \nabla_\theta L(\theta)$
2. Update biased first moment estimate: $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$
3. Update biased second raw moment estimate: $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
4. Compute bias-corrected first moment estimate: $\hat{m}_t = m_t / (1 - \beta_1^t)$

5. Compute bias-corrected second raw moment estimate: $\hat{v}_t = v_t/(1-\beta_2^t)$
6. Update parameters: $\theta = \theta - \eta\hat{m}_t/(\sqrt{\hat{v}_t} + \epsilon)$

Where:

- β_1 and β_2 are hyperparameters that control the exponential decay rates of the moment estimates.
- In pytorch the deafault value of $\beta_1 = 0.9$ and $\beta_2 = 0.999$
- η is the learning rate and we used $\eta = 0.001$.
- ϵ is a small scalar added to improve numerical stability default value is $\epsilon = 1e^{-}8$.

Training Process

- We Trained for Epochs: 10,000.
- $\eta = 0.001$.
- Total Time taken: 11.919572353363037 seconds

```
Epoch 9500, Loss: 0.002287372248247266
Epoch 9600, Loss: 0.0019192623440176249
Epoch 9700, Loss: 0.0031389121431857347
Epoch 9800, Loss: 0.0029367285314947367
Epoch 9900, Loss: 0.0017199647845700383
Total training time: 11.919572353363037 seconds
Model training complete and saved.
```

Figure 1: loss

–

For Demonstrartion

We used matplotlib python library to draw some Trajectory to imitate.

- Given some Random trajectories

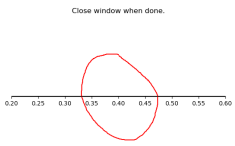

Figure 2: traj

–

Result

Here is a YouTube video http://www.youtube.com/v/CYL4t0xv4y4