Derivation of Denoising Diffusion Policy Optimization Objectives from Scratch

Tanishq Mathew Abraham, Ph.D.

September 17, 2023

1 Introduction

Denoising Diffusion Policy Optimization (DDPO) applies reinforcement learning, and specifically policy gradient methods, to optimize diffusion models against some desired reward signal (ex: aesthetics). A full tutorial and code example is provided as a blog post, along with a short explanation and intuition of how the training works. This document serves specifically as supplemental material to the blog post, deriving the DDPO objectives from scratch for a mathematically inclined beginner to reinforcement learning (but fairly well-versed with diffusion models).

2 Diffusion Model Refresher

A diffusion model is described by a forward and reverse process. The forward process is when we start out with a clean image $\mathbf{x}_0$ and repeatedly add Gaussian noise $\epsilon_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ to give use noisier and noisier images $\mathbf{x}_t$. This is described by the following:

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$$

where β_t is a predefined monotonically increasing variance schedule. The forward process runs for a total of T timesteps and finally ends with pure noise $\mathbf{x}_T$. The reverse process starts with pure noise $\mathbf{x}_T$ and uses a neural network to repeatedly denoise the image giving us $\mathbf{x}_t$. The end of the reverse process gives us back our samples $\mathbf{x}_0$. This is described as follows:

$$p_\theta^{\text{diffusion}}(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \tilde{\beta}_t\mathbf{I})$$

where $\tilde{\beta}_t$ is the variance schedule for the reverse schedule and $\mu_\theta(\mathbf{x}_t, t)$ is the denoising neural network. Note that the denoising neural network can be reparameterized to predict the noise ϵ_t in the image. So instead of predicting the denoised image $\hat{\mathbf{x}}_0$ directly, we can predict the noise in the image and subtract it out to get $\hat{\mathbf{x}}_0$. We train the reparameterized denoising neural network $\epsilon_\theta(\mathbf{x}_t, t)$ in the reverse diffusion process with a simple MSE loss:

$$L_{simple} = \mathbb{E}_{t,\mathbf{x}_t,\epsilon_t}||\epsilon_t - \epsilon_\theta(\mathbf{x}_t, t)||$$

In practice, training and sampling is quite simple. During each training step, a random image $\mathbf{x}_0$ and timestep t is selected, the forward process starts from $\mathbf{x}_0$ till timestep t to get $\mathbf{x}_t$ using the noise ϵ_t, this is passed into our denoising model, and the MSE between the ϵ_t used to calculate $\mathbf{x}_t$ and the predicted $\epsilon_\theta(\mathbf{x}_t, t)$ is optimized. During sampling, we start out with random Gaussian noise $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and the denoising neural network is repeatedly applied to give us $\mathbf{x}_{t-1}$ until we reach our sample $\mathbf{x}_0$.

3 Introduction to Reinforcement Learning

Okay let's now dig into how reinforcement learning works and derive the DDPO objective. What we want to do is to maximize the reward signal. We can mathematically express this as follows:

$$\theta^\star = \arg\max_\theta \mathbb{E}_{\mathbf{x}_0\sim p_\theta(\cdot\mid\mathbf{c})}[r(\mathbf{x}_0 \mid \mathbf{c})]$$

where θ is the weights of our diffusion model, $\mathbf{c}$ is some conditioning for the diffusion model, and $r(\cdot)$ is our reward function.

It would be nice to directly maximize for $r(\cdot)$ and if our model was a single evaluation of a neural network, we could simply backpropagate through the neural network and use an optimizer to update the weights. But that's not what happens with a diffusion model! We have multiple timesteps for which we apply our denoising neural network. This constructs a **trajectory** as its known in the RL literature. In standard RL literature, our trajectory is composed of **states** and **actions**. A model that we are optimizing provides the next action given the current state, and this model is referred to as the **policy**. This framework is known as a **Markov Decision Process (MDP)**. Note that in the general MDP framework, a reward is usually given after each action, and we optimize over the sum of rewards over the whole trajectory.

We can easily describe diffusion models as an MDP, which will allow us to use standard results in RL for diffusion model optimization.

$$\begin{aligned}
&\mathbf{s}_t \triangleq (\mathbf{c}, t, \mathbf{x}_t) \quad && \pi(\mathbf{a}_t \mid \mathbf{s}_t) \triangleq p_\theta^{\text{diffusion}}(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{c}) \quad && p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t) \triangleq \left(\delta_{\mathbf{c}}, \delta_{t-1}, \delta_{\mathbf{x}_{t-1}}\right) \\
&\mathbf{a}_t \triangleq \mathbf{x}_{t-1} && \rho_0(\mathbf{s}_0) \triangleq (p(\mathbf{c}), \delta_T, \mathcal{N}(\mathbf{0}, \mathbf{I})) && R(\mathbf{s}_t, \mathbf{a}_t) \triangleq \begin{cases} r(\mathbf{x}_0, \mathbf{c}) & \text{if } t = 0 \\ 0 & \text{otherwise} \end{cases}
\end{aligned}$$

$$\mathcal{J}(\theta) = \mathbb{E}_{\tau\sim p(\cdot\mid\pi)}\left[\sum_{t=0}^{T} R(\mathbf{s}_t, \mathbf{a}_t)\right] = \mathbb{E}_{\mathbf{x}_0\sim p_\theta(\cdot\mid\mathbf{c})}[r(\mathbf{x}_0 \mid \mathbf{c})]$$

$\mathbf{s}_t$ is the state, which is just the current noisy image $\mathbf{x}_t$ (along with the timestep and condition info). $\mathbf{a}_t$ is the action, which is the slightly less noisy image $\mathbf{x}_{t-1}$. $\pi(\mathbf{a}_t \mid \mathbf{s}_t)$ is the policy that takes the current state and provides the next action, which is our diffusion model $p_\theta^{\text{diffusion}}(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{c})$. $\rho_0(\mathbf{s}_0)$ is the distribution of the initial states, which in this case is the same distribution for $\mathbf{x}_T$, a standard isotropic normal distribution, with the timestep always being T and the conditioning having whatever prior distribution as in the dataset. $p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$ is giving $\mathbf{s}_{t+1}$ given the current state and action, and is basically just saying it always goes to the state associated with $\mathbf{x}_{t-1}$. $R(\mathbf{s}_t, \mathbf{a}_t)$ is the reward after each action/state (which is zero until the very last step when the generation is complete).

Okay, with this framework in place it becomes trivial to apply standard **policy gradient** optimization methods like **REINFORCE** and **proximal policy optimization (PPO)**. Let's derive these two algorithms now.

Let's represent the entire trajectory as τ and the probability density for trajectories as $p_\theta(\mathbf{s}_0, \mathbf{a}_0, \cdots, \mathbf{s}_T, \mathbf{a}_T) = p_\theta(\tau)$.

Here is our objective, note that the expectation is defined as an integral:

$$\mathcal{J}_{\text{RL}}(\pi) = \mathbb{E}_{\tau\sim p_\theta(\tau)}[r(\tau)] = \int p_\theta(\tau) r(\tau) d\tau$$

We want to perform gradient ascent on $\mathcal{J}(\theta)$ in order to maximize the reward:

$$\nabla_\theta \mathcal{J}(\theta) = \int \nabla_\theta p_\theta(\tau) r(\tau) d\tau$$

We can rewrite this using the following identity:

$$\nabla_\theta p_\theta(\tau) = p_\theta(\tau)\frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)} = p_\theta(\tau)\nabla_\theta \log p_\theta(\tau)$$

Rewriting the gradient equation:

$$\int \nabla_\theta p_\theta(\tau) r(\tau) d\tau = \int p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) r(\tau) d\tau = \mathbb{E}_{\tau\sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau) r(\tau)]$$

So this gives:

$$\nabla_\theta \mathcal{J}(\theta) = \mathbb{E}_{\tau\sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau) r(\tau)]$$

Now, note that:

$$p_\theta(\tau) = p_\theta(\mathbf{s}_0, \mathbf{a}_0, \cdots, \mathbf{s}_T, \mathbf{a}_T) = \rho_0(\mathbf{s}_0) \prod_{t=0}^{T} \pi_\theta\left(\mathbf{a}_t \mid \mathbf{s}_t\right) p\left(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t\right)$$

Basically we're mathematically describing that we're taking some action given the current state based on the policy, and then given that action we have some probability for the next state. And since the transitions are independent (it's a *Markov* Decision Process), we can use the product rule for probability and get the probability for the full trajectory.

Okay then we can take the logarithm of both sides:

$$\log p_\theta(\tau) = \log p(\mathbf{s}_0) + \sum_{t=0}^{T} \log \pi_\theta\left(\mathbf{a}_t \mid \mathbf{s}_t\right) + \log p\left(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t\right)$$

Then,

$$\nabla_\theta \log p_\theta(\tau) = \nabla_\theta \left[\log p(\mathbf{s}_0) + \sum_{t=0}^{T} \log \pi_\theta\left(\mathbf{a}_t \mid \mathbf{s}_t\right) + \log p\left(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t\right)\right] = \nabla_\theta \sum_{t=0}^{T} \log \pi_\theta\left(\mathbf{a}_t \mid \mathbf{s}_t\right)$$

(dropping terms that don't depend on θ)

Remembering that $r(\tau) = \sum_{t=0}^{T} R(\mathbf{s}_t, \mathbf{a}_t)$, we can get our **policy gradient**:

$$\nabla_\theta \mathcal{J}(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}\left[\left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta\left(\mathbf{a}_t \mid \mathbf{s}_t\right)\right)\left(\sum_{t=0}^{T} R(\mathbf{s}_t, \mathbf{a}_t)\right)\right]$$

This gradient is referred to as the REINFORCE gradient and is only one type of policy gradient that could be used. Of course, this policy gradient is then used to update the weights of our model using gradient ascent:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \mathcal{J}(\theta)$$

where α is some learning rate.

One implementation point is that the expectation is over the trajectories but of course we can't take and sum over all possible trajectories. The expectation is estimated with just the sampled trajectories in the currenty batch. One other implementation point to mention: we could calculate our gradient and then pass that gradient to our optimizer, or we could let autograd handle the calculation of the gradient by constructing a loss function and treating it as a standard training loop. The latter is what is done in practice even though it is not explicitly mentioned often in the papers. So our loss function is:

$$\mathcal{L}(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}\left[-\left(\sum_{t=0}^{T} \log \pi_\theta\left(\mathbf{a}_t \mid \mathbf{s}_t\right)\right)\left(\sum_{t=0}^{T} R(\mathbf{s}_t, \mathbf{a}_t)\right)\right]$$

Okay so we can simply plug in our diffusion model terms based on how it fits into the MDP framework, which we described earlier. We get:

$$\nabla_\theta \mathcal{J}(\theta) = \mathbb{E}\left[\left(\sum_{t=0}^{T} \nabla_\theta \log p_\theta^{\text{diffusion}}\left(\mathbf{x}_{t-1} \mid \mathbf{c}, t, \mathbf{x}_t\right)\right) r(\mathbf{x}_0, \mathbf{c})\right]$$

This objective and gradient estimator is referred in the paper as $\mathbf{DDPO_{SF}}$.

One challenge with this approach is that for each optimization step, the sampling from the current iteration of the model needs to be performed, we need to re-calculate $\mathbf{x}_t$ as it comes from the current version of the model. This can be computationally expensive and wasteful, as the samples collected with previous iterations of the model cannot be used to learn.

One trick to address this is known as importance sampling. This relies on the following identity (trivial to demonstrate based on the expectation definition):

$$\mathbb{E}_{x\sim p(x)}\left[f(x)\right] = \mathbb{E}_{x\sim q(x)}\left[\frac{p(x)}{q(x)}f(x)\right]$$

In order to take advantage of this, let's rewrite our policy gradient:

$$\begin{aligned}\nabla_\theta \mathcal{J}(\theta) &= \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\left(\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta\left(\mathbf{a}_t \mid \mathbf{s}_t\right)\right)\left(\sum_{t=0}^{T} R(\mathbf{s}_t,\mathbf{a}_t)\right)\right] \\ &= \mathbb{E}_{\mathbf{s}_t\sim p_\theta(\mathbf{s}_t)}\left[\mathbb{E}_{\mathbf{a}_t\sim \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}\left[\left(\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta\left(\mathbf{a}_t \mid \mathbf{s}_t\right)\right)\left(\sum_{t=0}^{T} R(\mathbf{s}_t,\mathbf{a}_t)\right)\right]\right]\end{aligned}$$

This is just saying that the expectation over all the trajectories can be decomposed into an expectation over all states of a conditional expectation over all actions as chosen by the policy. That's a little complicated so let's break it down a bit.

Basically, our trajectory is composed of the states and actions and we have an expectation over all trajectories. Our actions are given by the stochastic policy dependent on the states, so we will have a conditional expectation for the actions. We can then take the expectation over all the states to get the full expectation of all actions. This gives such the expectation over all the trajectories (states and actions). Note that the marginal distribution for the states is itself dependent on θ since the states are given by $p\left(\mathbf{s}_t \mid \mathbf{s}_{t-1},\mathbf{a}_t\right)$ but of course $\mathbf{a}_t \sim \pi_\theta\left(\mathbf{a}_t \mid \mathbf{s}_t\right)$.

Okay now we can apply importance sampling twice here, where we sample from a new distribution given by θ_{old}, old parameters for our policy:

$$\nabla_\theta \mathcal{J}(\theta) = \mathbb{E}_{\mathbf{s}_t\sim p_{\theta_{old}}(\mathbf{s}_t)}\left[\frac{p_\theta\left(\mathbf{s}_t\right)}{p_{\theta_{old}}\left(\mathbf{s}_t\right)}\mathbb{E}_{\mathbf{a}_t\sim \pi_{\theta_{old}}(\mathbf{a}_t|\mathbf{s}_t)}\left[\frac{\pi_\theta\left(\mathbf{a}_t \mid \mathbf{s}_t\right)}{\pi_{\theta_{old}}\left(\mathbf{a}_t \mid \mathbf{s}_t\right)}\left(\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta\left(\mathbf{a}_t \mid \mathbf{s}_t\right)\right)\left(\sum_{t=0}^{T} R(\mathbf{s}_t,\mathbf{a}_t)\right)\right]\right]$$

It can be shown (which I won't go over here for now) that $\frac{p_\theta(\mathbf{s}_t)}{p_{\theta_{old}}(\mathbf{s}_t)} \approx 1$ if $|\pi_\theta\left(\mathbf{a}_t \mid \mathbf{s}_t\right) - \pi_{\theta_{old}}\left(\mathbf{a}_t \mid \mathbf{s}_t\right)| \leq \epsilon$. So this gives:

$$\nabla_\theta \mathcal{J}(\theta) = \mathbb{E}_{\mathbf{s}_t\sim p_{\theta_{old}}(\mathbf{s}_t)}\left[\mathbb{E}_{\mathbf{a}_t\sim \pi_{\theta_{old}}(\mathbf{a}_t|\mathbf{s}_t)}\left[\frac{\pi_\theta\left(\mathbf{a}_t \mid \mathbf{s}_t\right)}{\pi_{\theta_{old}}\left(\mathbf{a}_t \mid \mathbf{s}_t\right)}\left(\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta\left(\mathbf{a}_t \mid \mathbf{s}_t\right)\right)\left(\sum_{t=0}^{T} R(\mathbf{s}_t,\mathbf{a}_t)\right)\right]\right]$$

$$\nabla_\theta \mathcal{J}(\theta) = \mathbb{E}_{\tau\sim p_{\theta_{old}}(\tau)}\left[\left(\sum_{t=0}^{T}\frac{\pi_\theta\left(\mathbf{a}_t \mid \mathbf{s}_t\right)}{\pi_{\theta_{old}}\left(\mathbf{a}_t \mid \mathbf{s}_t\right)}\nabla_\theta \log \pi_\theta\left(\mathbf{a}_t \mid \mathbf{s}_t\right)\right)\left(\sum_{t=0}^{T} R(\mathbf{s}_t,\mathbf{a}_t)\right)\right]$$

Again this can be written down as a loss function that we perform gradient descent on:

$$\mathcal{L}(\theta) = \mathbb{E}_{\tau\sim p_{\theta_{old}}(\tau)}\left[-\left(\sum_{t=0}^{T}\frac{\pi_\theta\left(\mathbf{a}_t \mid \mathbf{s}_t\right)}{\pi_{\theta_{old}}\left(\mathbf{a}_t \mid \mathbf{s}_t\right)}\right)\left(\sum_{t=0}^{T} R(\mathbf{s}_t,\mathbf{a}_t)\right)\right]$$

Just for simpler notation, let's rewrite the loss as an expectation over t:

$$\mathcal{L}(\theta) = \mathbb{E}_t\left[-\frac{\pi_\theta\left(\mathbf{a}_t \mid \mathbf{s}_t\right)}{\pi_{\theta_{old}}\left(\mathbf{a}_t \mid \mathbf{s}_t\right)}\left(\sum_{t=0}^{T} R(\mathbf{s}_t,\mathbf{a}_t)\right)\right]$$

Again, we can plug in the diffusion model terms based on the MDP framework and get this loss function:

$$L(\theta) = \mathbb{E}\left[-\sum_{t=0}^{T}\frac{p_\theta^{\text{diffusion}}(\mathbf{x}_{t-1}|\mathbf{c},t,\mathbf{x}_t)}{p_{\theta_{old}}^{\text{diffusion}}(\mathbf{x}_{t-1}|\mathbf{c},t,\mathbf{x}_t)}r(\mathbf{x}_0,\mathbf{c})\right]$$

Minimization of this loss function is equivalent to gradient with the following gradient:

$$\hat{g} = \mathbb{E}\left[\sum_{t=0}^{T}\frac{p_\theta^{\text{diffusion}}(\mathbf{x}_{t-1}|\mathbf{c},t,\mathbf{x}_t)}{p_{\theta_{old}}^{\text{diffusion}}(\mathbf{x}_{t-1}|\mathbf{c},t,\mathbf{x}_t)}\nabla_\theta p_\theta^{\text{diffusion}}(\mathbf{x}_{t-1}|\mathbf{c},t,\mathbf{x}_t) r(\mathbf{x}_0,\mathbf{c})\right]$$

Note that the reward $r(\mathbf{x}_0, \mathbf{c})$ is usually normalized, and the normalized reward is referred to the advantage $A(\mathbf{x}_0, \mathbf{c})$. So the advantage can be negative if it's less than average.

Note, as mentioned earlier, we can't have the current policy diverge too much from the previous policy. We can apply clipping to the importance sampling ratio to the loss function:

$$L(\theta) = \mathbb{E}\left[-\sum_{t=0}^{T} \min\left(\frac{p_\theta^{\text{diffusion}}(\mathbf{x}_{t-1}|\mathbf{c}, t, \mathbf{x}_t)}{p_{\theta_{old}}^{\text{diffusion}}(\mathbf{x}_{t-1}|\mathbf{c}, t, \mathbf{x}_t)} A(\mathbf{x}_0, \mathbf{c}), \text{clip}\left(\frac{p_\theta^{\text{diffusion}}(\mathbf{x}_{t-1}|\mathbf{c}, t, \mathbf{x}_t)}{p_{\theta_{old}}^{\text{diffusion}}(\mathbf{x}_{t-1}|\mathbf{c}, t, \mathbf{x}_t)}, 1-\epsilon, 1+\epsilon\right) A(\mathbf{x}_0, \mathbf{c})\right)\right]$$

So if the policy diverges too much (the ratio is either much larger or much smaller than 1) the loss function is clipped to a certain value and the gradient will be zero and no updates will be made. Else, it's just sort of equivalent to the REINFORCE with importance sampling.

The loss function can be written in a way that's numerically easier to calculate/more stable (using logs, ignoring the clipping for now):

$$L(\theta) = \mathbb{E}\left[-\sum_{t=0}^{T} \exp\left(\log p_\theta^{\text{diffusion}}(\mathbf{x}_{t-1}|\mathbf{c}, t, \mathbf{x}_t) - \log p_{\theta_{old}}^{\text{diffusion}}(\mathbf{x}_{t-1}|\mathbf{c}, t, \mathbf{x}_t)\right) A(\mathbf{x}_0, \mathbf{c})\right]$$

The objective and gradient estimator described here is referred in the paper as $\mathbf{DDPO_{IS}}$.

4 Additional Resources

- The original DDPO paper
- Sergey Levine's lecture slides on policy gradient
- Sergey Levine's lecture slides on PPO/TRPO
- HuggingFace course unit on PPO
- UWaterloo lecture on PPO
- Pieter Abbeel's lecture on TRPO/PPO
- OpenAI Spinning Up RL course unit on PPO
- Karpathy's blog post on RL
- Blog post on Fundamentals of Policy Gradients
- Policy Gradients Explained blog post