Diffusion for World Modeling: Visual Details Matter in Atari

February 28, 2025

Reference

Eloi Alonso, Adam Jelley, Vincent Micheli, Anssi Kanervisto, Amos Storkey, Tim Pearce, and François Fleuret. Diffusion for world modeling: Visual details matter in atari, 2024. URL `https://arxiv.org/abs/2405.12399`.

Score-based Diffusion Models

Consider a diffusion process $\{\mathbf{x}^\tau\}_{\tau\in[0,\mathcal{T}]}$ with corresponding marginal distributions $\{p^\tau\}_{\tau\in[0,\mathcal{T}]}$, where:

- Diffusion models gradually add noise to data samples until they become pure noise
- This creates a sequence of increasingly noisy versions of the data
- The process is reversible - we can learn to denoise corrupted samples
- Boundary conditions: $p^0 = p^{data}, p^{\mathcal{T}} = p^{prior}$
- p^{data} is our original data distribution
- p^{prior} is a tractable unstructured prior distribution, such as a Gaussian
- The time parameter τ controls the noise level ($\tau = 0$ is clean, $\tau = \mathcal{T}$ is pure noise)

Forward Process SDE

The diffusion process follows a stochastic differential equation:

$$d\mathbf{x} = \mathbf{f}(\mathbf{x}, \tau)d\tau + g(\tau)d\mathbf{w}$$

Where:

- $\mathbf{f}(\mathbf{x}, \tau)d\tau$ is the drift term (deterministic part that guides the overall direction)
- $g(\tau)$ is the diffusion coefficient (controls noise strength over time)
- $\mathbf{w}$ is a standard Wiener process (provides random noise)

Intuition:

- The equation describes how data points move and get corrupted over time
- First term ($\mathbf{f}(\mathbf{x}, \tau)d\tau$) determines the average path
- Second term ($g(\tau)d\mathbf{w}$) adds random noise, gradually making samples more noisy

Reverse Process SDE

It can be shown that the reverse of forward process of the SDE is also an SDE:

$$d\mathbf{x} = \left[f(\mathbf{x}, \tau) - g(\tau)^2 \nabla_{\mathbf{x}} \log p^{\tau}(\mathbf{x}) \right] d\tau + g(\tau) d\bar{w}$$

Where:

- $\bar{\mathbf{w}}$ is the reverse-time Wiener process
- $\nabla_{\mathbf{x}} \log p^{\tau}(\mathbf{x})$ is the (Stein) score function
- The score function is the gradient of the log-marginals with respect to the support

Score Function

Intuition behind the score function $\nabla_{\mathbf{x}} \log p^{\tau}(\mathbf{x})$:

- Tells us which direction to move our noisy sample to make it more "data-like"
- Acts like a guide pointing towards regions of higher probability of being real data
- In image terms: points towards what the clean image should look like

The score function is crucial because it provides the "hint" for how to reverse the noise. However, we don't know the true score function, so we need to estimate it

Learning the Score Function

Challenge: We need to estimate the score function but never observe it directly

Solution: Score Matching - train using only clean images and a known noise process

Key idea: Since we control how noise is added, we can simulate the process

- Generate noisy image $\mathbf{x}^{\tau}$ from clean image $\mathbf{x}^{0}$
- Train model to predict direction of noise removal

Score Matching Loss

The loss function measures how well our model estimates the score:

$$\mathcal{L}(\theta) = \mathbb{E}\left[\left\|\mathbf{S}_\theta(\mathbf{x}, \tau) - \nabla_{\mathbf{x}^\tau} \log p^{0\tau}(\mathbf{x}^\tau \mid \mathbf{x}^0)\right\|^2\right]$$

Where:

- $\mathbf{S}_\theta(\mathbf{x}, \tau)$ is our model's estimate of the score function
- $\nabla_{\mathbf{x}^\tau} \log p^{0\tau}(\mathbf{x}^\tau \mid \mathbf{x}^0)$ is the true score of the Gaussian noise

Gaussian Perturbation Kernel

The distribution $p^{0\tau}$ describes how clean data transforms into noisy data:

$$p^{0\tau}(\mathbf{x}^\tau \mid \mathbf{x}^0) = \mathcal{N}\Big(\mathbf{x}^\tau;\, \alpha(\tau)\mathbf{x}^0,\, \sigma^2(\tau)\mathbf{I}\Big)$$

Where:

- $p^{0\tau}$ is the Gaussian perturbation kernel of the forward process
- $\alpha(\tau)$ scales the clean sample
- $\sigma^2(\tau)$ is the noise variance at time τ
- $\mathbf{I}$ ensures independent noise per dimension

Key insight: We choose a Gaussian transition for analytical tractability

Simplified Reconstruction View (1/2)

The complex score matching loss can be rewritten in a more intuitive form:

Original score matching loss:

$$\mathcal{L}(\theta) = \mathbb{E}\left[\left\|\mathbf{S}_\theta(\mathbf{x}, \tau) - \nabla_{\mathbf{x}^\tau} \log p^{0\tau}(\mathbf{x}^\tau \mid \mathbf{x}^0)\right\|^2\right]$$

Simplified reconstruction loss:

$$\mathcal{L}(\theta) = \mathbb{E}\left[\left\|\mathbf{D}_\theta(\mathbf{x}^\tau, \tau) - \mathbf{x}^0\right\|^2\right]$$

These are equivalent to optimize when we:

- Define a denoising function:

$$\mathbf{D}_\theta(\mathbf{x}^\tau, \tau) = \mathbf{S}_\theta(\mathbf{x}, \tau)\sigma^2(\tau) + \mathbf{x}^\tau$$

- Assume the forward process preserves the clean data mean:

$$\mu(\tau, \mathbf{x}^0) = \mathbf{x}^0$$

Simplified Reconstruction View (2/2)

Key components:

- $\mathbf{D}_\theta$ is our "d"enoising model that tries to recover the clean image $\mathbf{x}^0$ from the noisy image $\mathbf{x}^\tau$
- $\sigma(\tau)$ controls how much noise is present at time τ

This reformulation transforms the abstract score matching problem into a concrete denoising task.

Adapting Diffusion Models for World Modeling

Key differences from standard diffusion:

- Standard diffusion: Generates random samples from data distribution
- World models: Must predict next state given history of states and actions

Modified training objective:

$$\mathcal{L}(\theta) = \mathbb{E}\left[\|\mathbf{D}_\theta(\mathbf{x}_{t+1}^\tau, \tau, \mathbf{x}_{\le t}^0, a_{\le t}) - \mathbf{x}_{t+1}^0\|^2\right]$$

How training works:

- Sample trajectory $(\mathbf{x}_{\le t}^0, a_{\le t}, \mathbf{x}_{t+1}^0)$ from experience replay
- Sample $\mathbf{x}_{t+1}^\tau \sim p^{0\tau}(\mathbf{x}_{t+1}^\tau | \mathbf{x}_{t+1}^0)$ using the perturbation kernel
- Train $\mathbf{D}_\theta$ to denoise $\mathbf{x}_{t+1}^\tau$ conditioned on $(\mathbf{x}_{\le t}^0, a_{\le t})$

Generating Predictions with World Models

To predict the next state in the environment:

- Start with random noise and gradually denoise it
- Use history of states $\mathbf{x}^0_{\leq t}$ and actions $a_{\leq t}$ to guide denoising
- Balance these competing factors:
  - Better predictions (more denoising steps)
  - Faster predictions (fewer denoising steps)
  - Computational resources needed

Diamond Network Architecture (Alonso et al. [2024])

Key insight: Architecture choice affects stability with few denoising steps

- DDPM becomes unstable with few denoising steps
- EDM remains stable even with single-step denoising

The denoising model $\mathbf{D}_\theta$ uses a preconditioned architecture:

$$\mathbf{D}_\theta(\mathbf{x}_{t+1}^\tau, y_t^\tau) = c_{\text{skip}}^\tau \, \mathbf{x}_{t+1}^\tau + c_{\text{out}}^\tau \, \mathbf{F}_\theta\left(c_{\text{in}}^\tau \, \mathbf{x}_{t+1}^\tau, \, y_t^\tau\right)$$

Where:

- $\mathbf{F}_\theta$ is a U-Net backbone
- $y_t^\tau := (c_{\text{noise}}^\tau, \, \mathbf{x}_{\leq t}^0, \, a_{\leq t})$ is the conditioning information
- $c_{\text{skip}}^\tau, c_{\text{in}}^\tau, c_{\text{out}}^\tau$ are preconditioners determined by $\sigma(\tau)$
- c_{noise}^τ is also determined by $\sigma(\tau)$ to ensure stability

Preconditioned Architecture (1/2)

The denoising function combines two paths:

$$\mathbf{D}_\theta(\mathbf{x}_{t+1}^\tau, y_t^\tau) = \underbrace{c_{\text{skip}}^\tau\, \mathbf{x}_{t+1}^\tau}_{\text{skip connection}} + \underbrace{c_{\text{out}}^\tau\, \mathbf{F}_\theta\left(c_{\text{in}}^\tau\, \mathbf{x}_{t+1}^\tau,\ y_t^\tau\right)}_{\text{network prediction}}$$

Skip connection:

- Acts as a shortcut, letting part of the noisy input bypass the network
- Weight c_{skip}^τ chosen based on noise level
- When noise is low, keeps most of the input unchanged

Preconditioned Architecture (2/2)

Network prediction:

- $\mathbf{F}_\theta$ ("F"ilter) learns to predict either the clean image or a residual correction
- Input scaling c_{in}^{τ} maintains consistent input scale across noise levels
- Output scaling c_{out}^{τ} properly scales network output for final prediction

Conditioning information y_t^{τ}:

- Noise level c_{noise}^{τ} guides denoising strength
- Past observations $\mathbf{x}_{\leq t}^{0}$ provide context
- Past actions $a_{\leq t}$ inform dynamics

Benefits of Preconditioning

Why use this architecture?

- Stabilizes training across noise levels
- Adapts network's role based on noise:
  - High noise: Network does heavy lifting
  - Low noise: Network focuses on residual
- Makes learning easier by separating tasks
- Enables stable single-step denoising

Training Implementation

The original training loss for the denoising network:

$$\mathcal{L}(\theta) = \mathbb{E}\left[\|\mathbf{D}_\theta(\mathbf{x}_{t+1}^\tau, \tau, \mathbf{x}_{\le t}^0, a_{\le t}) - \mathbf{x}_{t+1}^0\|^2\right]$$

Can be shown to be equivalent to optimizing:

$$\mathcal{L}(\theta) = \mathbb{E}\left[\left\|\mathbf{F}_\theta(c_{\text{in}}^\tau \mathbf{x}_{t+1}^\tau, y_t^\tau) - \frac{1}{c_{\text{out}}^\tau}\left(\mathbf{x}_{t+1}^0 - c_{\text{skip}}^\tau \mathbf{x}_{t+1}^\tau\right)\right\|^2\right]$$

The training target adapts based on noise level $\sigma(\tau)$:

- High noise ($\sigma(\tau) \gg \sigma_{\text{data}}$):
  - $c_{\text{skip}}^\tau \to 0$
  - Network learns to predict clean signal $\mathbf{x}_{t+1}^0$ directly
- Low noise ($\sigma(\tau) \to 0$):
  - $c_{\text{skip}}^\tau \to 1$
  - Target becomes residual noise (difference between clean and noisy)
  - Prevents training from becoming trivial at low noise levels

Diamond Training Loop

```
Procedure training_loop():
    for epochs do
        collect_experience(steps_collect) ;
        for steps_diffusion_model do
            update_diffusion_model() ;
        for steps_reward_end_model do
            update_reward_end_model() ;
        for steps_actor_critic do
            update_actor_critic() ;
```

Experience Collection

```
Procedure collect_experience($n$):
    $\mathbf{x}_0^0 \leftarrow$ env.reset() ;
    for $t = 0$ to $n - 1$ do
        Sample $a_t \sim \pi_\phi(a_t \mid \mathbf{x}_t^0)$ ;
        $\mathbf{x}_{t+1}^0, r_t, d_t \leftarrow$ env.step($a_t$) ;
        $\mathcal{D} \leftarrow \mathcal{D} \cup \{\mathbf{x}_t^0, a_t, r_t, d_t\}$ ;
        if $d_t = 1$ then
            $\mathbf{x}_{t+1}^0 \leftarrow$ env.reset() ;
```

Diffusion Model Update

```
Procedure update_diffusion_model():
    Sample sequence $(\mathbf{x}^0_{t-L+1}, a_{t-L+1}, \dots, \mathbf{x}^0_t, a_t, \mathbf{x}^0_{t+1}) \sim \mathcal{D}$ ;
    Sample $\log(\sigma) \sim \mathcal{N}(P_{mean}, P^2_{std})$        // log-normal sigma ;
    Define $\tau := \sigma$                                             // identity schedule ;
    Sample $\mathbf{x}^\tau_{t+1} \sim \mathcal{N}(\mathbf{x}^0_{t+1}, \sigma^2\mathbf{I})$          // Add noise ;
    $\hat{\mathbf{x}}^0_{t+1} = \mathbf{D}_\theta(\mathbf{x}^\tau_{t+1}, \tau, \mathbf{x}^0_{t-L+1}, a_{t-L+1}, \dots, \mathbf{x}^0_t, a_t)$ ;
    Compute loss $\mathcal{L}(\theta) = \|\hat{\mathbf{x}}^0_{t+1} - \mathbf{x}^0_{t+1}\|^2$ ;
    Update $\mathbf{D}_\theta$ ;
```

Reward and Termination Model Update

```
Procedure update_reward_end_model():
    Sample indexes $\mathcal{I} := \{t, \dots, t + L + H - 1\}$ // burn-in + horizon ;
    Sample sequence $(\mathbf{x}_i^0, a_i, r_i, d_i)_{i \in \mathcal{I}} \sim \mathcal{D}$ ;
    Initialize $h = c = 0$                    // LSTM states;
    for $i \in \mathcal{I}$ do
        $\hat{r}_i, \hat{d}_i, h, c = R_\psi(\mathbf{x}_i, a_i, h, c)$ ;
    $\mathcal{L}(\psi) = \sum_{i \in \mathcal{I}} \mathrm{CE}(\hat{r}_i, \mathrm{sign}(r_i)) + \mathrm{CE}(\hat{d}_i, d_i)$ ;
    Update $R_\psi$ ;
```

Actor-Critic Update

```
Procedure update_actor_critic():
    Sample initial buffer $(\mathbf{x}^0_{t-L+1}, a_{t-L+1}, \ldots, \mathbf{x}^0_t) \sim \mathcal{D}$ ;
    Burn-in buffer with $R_\psi$, $\pi_\phi$ and $V_\phi$ to initialize LSTM states ;
    for $i = t$ to $t + H - 1$ do
        Sample $a_i \sim \pi_\phi(a_i \mid \mathbf{x}^0_i)$ ;
        Sample reward $r_i$ and termination $d_i$ with $R_\psi$ ;
        Sample next observation $\mathbf{x}^0_{i+1}$ with $\mathbf{D}_\theta$ ;
    Compute $V_\phi(\mathbf{x}_i)$ for $i = t, \ldots, t + H$ ;
    Compute RL losses $\mathcal{L}_V(\phi)$ and $\mathcal{L}_\pi(\phi)$ ;
    Update $\pi_\phi$ and $V_\phi$ ;
```

Experimental Setup & Results

Benchmark: Atari 100k (26 games)

- Only 100k environment steps (2 hours of gameplay)
- 5 random seeds per game, 2.9 days per run on RTX 4090

Performance:

- Mean Human Normalized Score (HNS): 1.46 (>1 is superhuman)
- Superhuman in 11/26 games
- Outperforms STORM, DreamerV3, IRIS, TWM, SimPle

Key Findings

Main Results:

- Sets new state-of-the-art for world model-based agents
- Achieves strong performance with limited training data
- Visual quality matters for world modeling

Implications:

- Diffusion models effectively capture environment dynamics
- High-quality visual predictions enable better planning
- Sample efficiency possible with good architectures

Technical Analysis

EDM Framework Advantages:

- Improved training objective over DDPM
- Stable with single-step denoising
- Adaptive mixing of noisy and denoised predictions

Denoising Strategy:

- Final choice: 3 denoising steps
- Balances prediction quality and computational efficiency
- Adapts to game complexity (1 step for simple, 3+ for complex)

Comparison with IRIS (Previous SOTA)

IRIS Limitations:

- Uses discrete autoencoder + transformer
- Visual inconsistencies in predictions
- Higher computational requirements

DIAMOND's Advantages:

- More consistent visual predictions
- Better preservation of important details
- Lower compute: fewer steps, smaller model