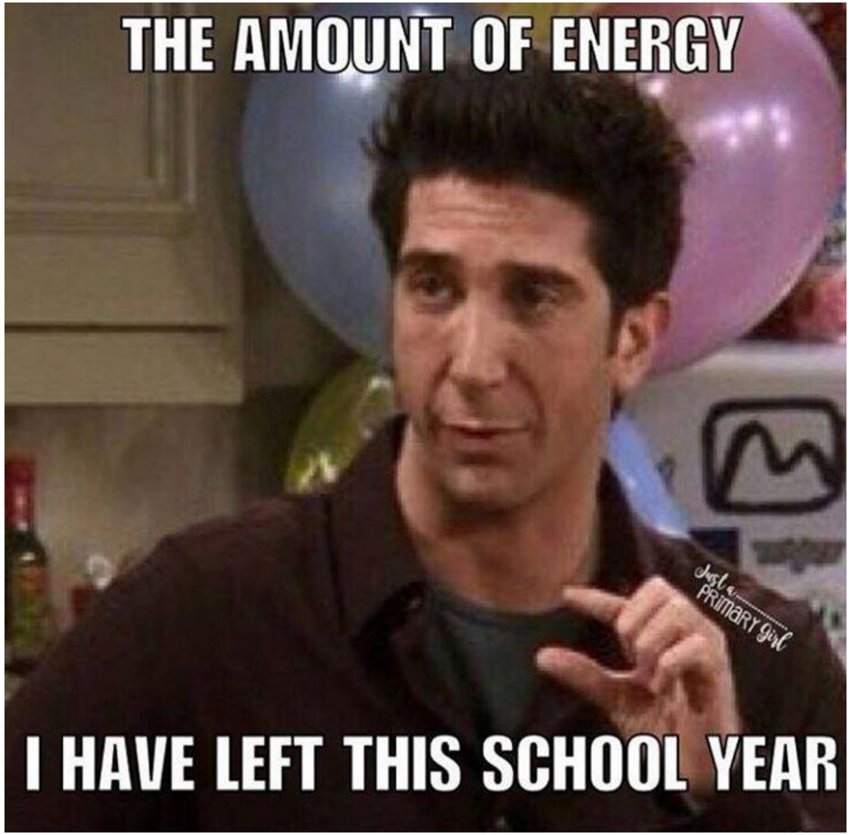

THE AMOUNT OF ENERGY
I HAVE LEFT THIS SCHOOL YEAR
Just a PRIMARY girl

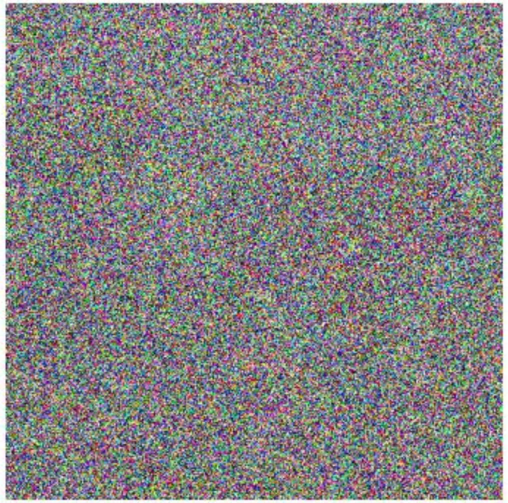
# ⚡Energy-Based Models⚡
# and
# 💯Score Modeling 💯

🔥🔥🔥

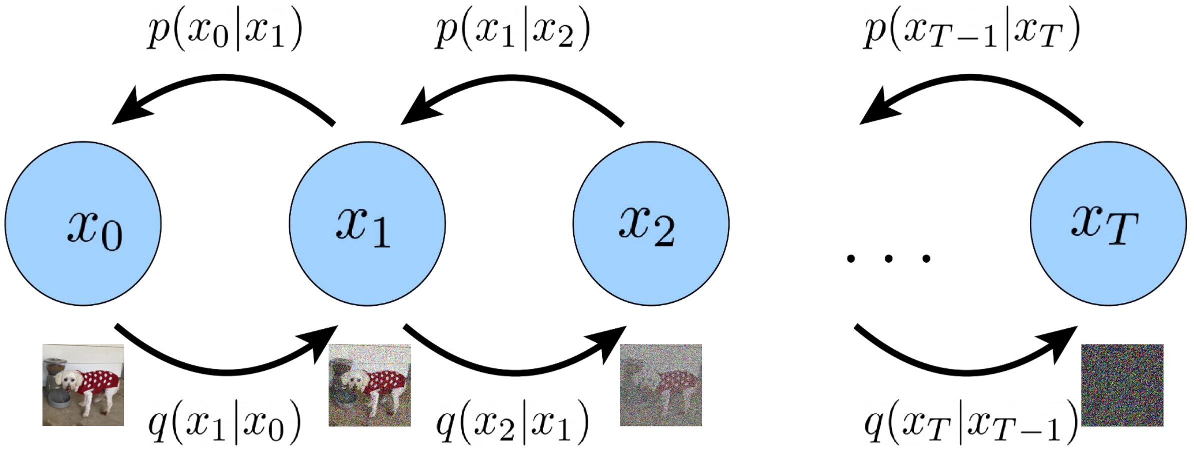
## Diffusion Models: a TLDR

**An observation:** adding steady amounts of Gaussian noise eventually corrupts an image into something indistinguishable from a standard Gaussian sample.

- Diffusion models simply learn to **reverse** this procedure over many timesteps



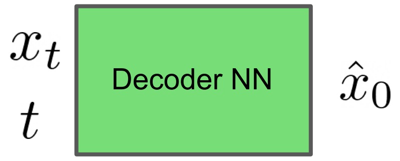
## Diffusion Models: a TLDR

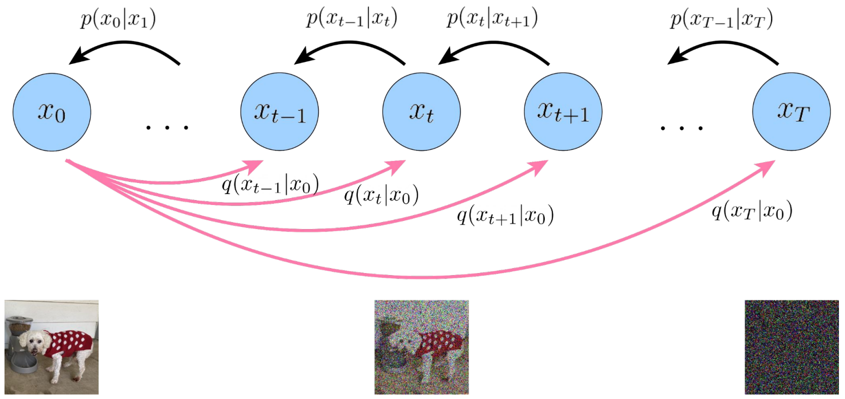


# Diffusion Models: A Quick Review

A Diffusion Model is:

- One NN that predicts a clean image from a noisy version of the image

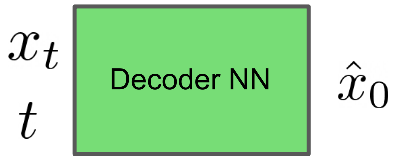


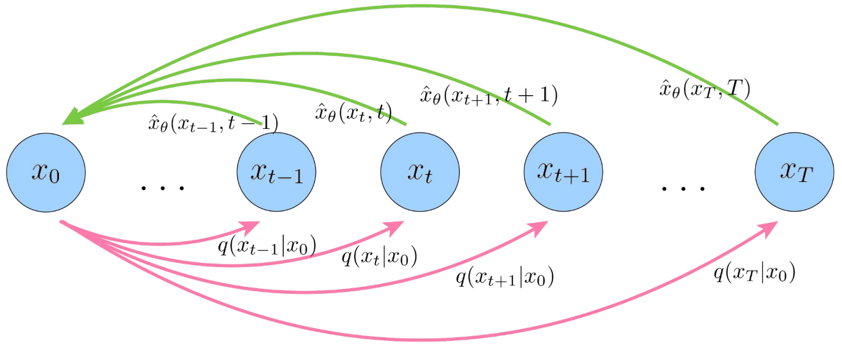


# Diffusion Models: A Quick Review

A Diffusion Model is:

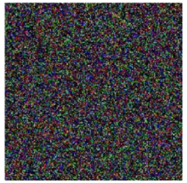
- One NN that predicts a clean image from a noisy version of the image





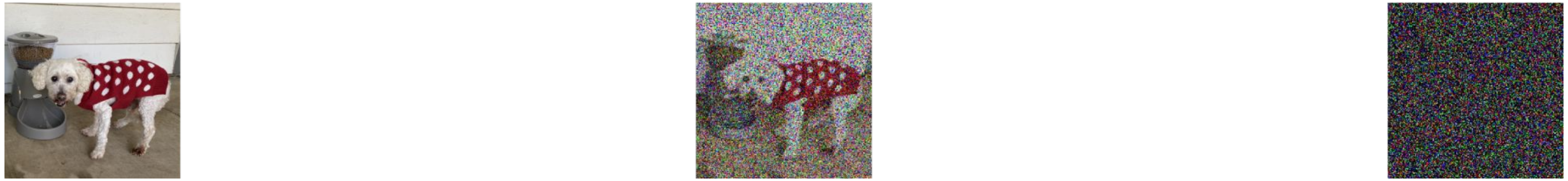
## Sampling

$$\begin{tikzcd}
x_0 & \cdots & x_{t-1} & x_t & x_{t+1} \arrow[llll, bend right, "\hat{x}_\theta(x_{t+1}, t+1)"'] & \cdots & x_T
\end{tikzcd}$$

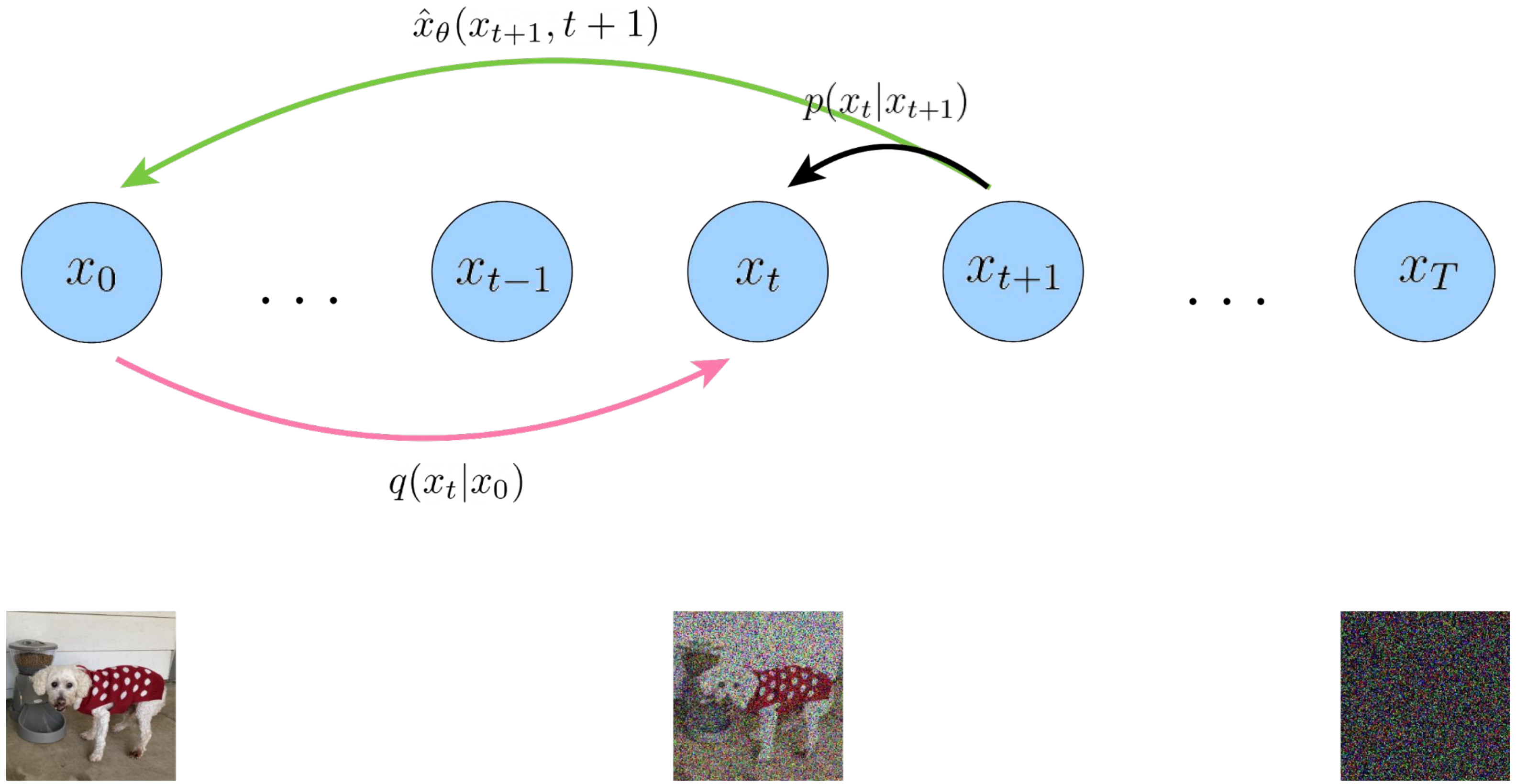


## Sampling

$$\begin{tikzcd}
x_0 \arrow[rrr, bend right, "q(x_t|x_0)"'] & \cdots & x_{t-1} & x_t & x_{t+1} \arrow[llll, bend right, "\hat{x}_\theta(x_{t+1}, t+1)"'] & \cdots & x_T
\end{tikzcd}$$

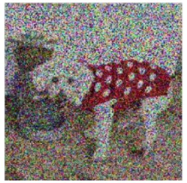


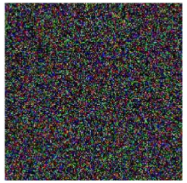
## Sampling



## Sampling

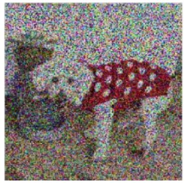
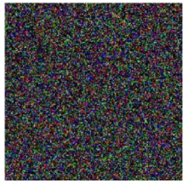
$$\begin{tikzcd}
x_0 & \cdots & x_{t-1} & x_t \arrow[lll, bend right, "\hat{x}_\theta(x_t, t)"'] & x_{t+1} \arrow[l, bend right, "p(x_t|x_{t+1})"'] & \cdots & x_T
\end{tikzcd}$$





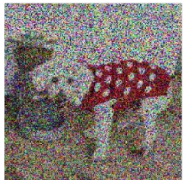
## Sampling

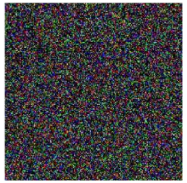
$$\begin{tikzcd}
x_0 \arrow[rr, bend right, "q(x_{t-1}|x_0)"'] & \cdots & x_{t-1} & x_t \arrow[lll, bend right, "\hat{x}_\theta(x_t, t)"'] & x_{t+1} \arrow[l, bend right, "p(x_t|x_{t+1})"'] & \cdots & x_T
\end{tikzcd}$$

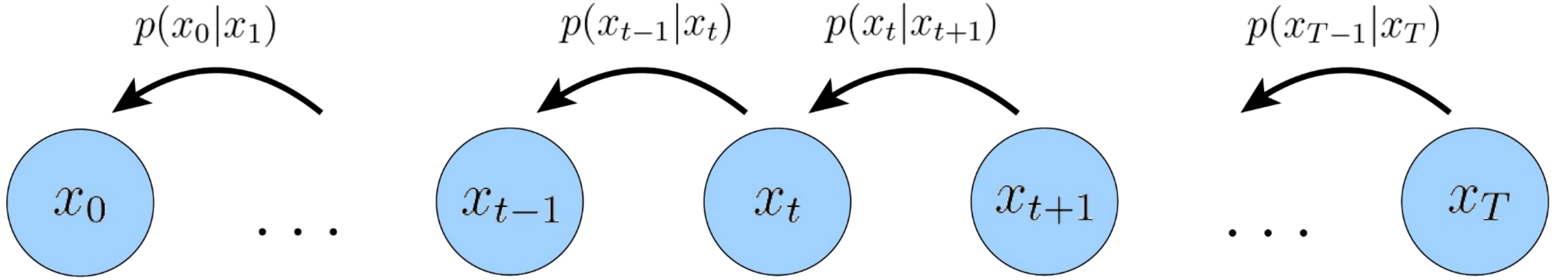
## Sampling

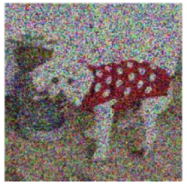
$$\begin{tikzcd}
x_0 \arrow[rr, bend right, "q(x_{t-1}|x_0)"'] & \cdots & x_{t-1} & x_t \arrow[l, bend right, "p(x_{t-1}|x_t)"'] \arrow[lll, bend right, "\hat{x}_\theta(x_t, t)"'] & x_{t+1} \arrow[l, bend right, "p(x_t|x_{t+1})"'] & \cdots & x_T
\end{tikzcd}$$

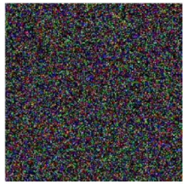




## Sampling



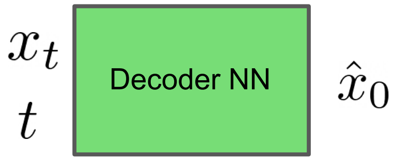


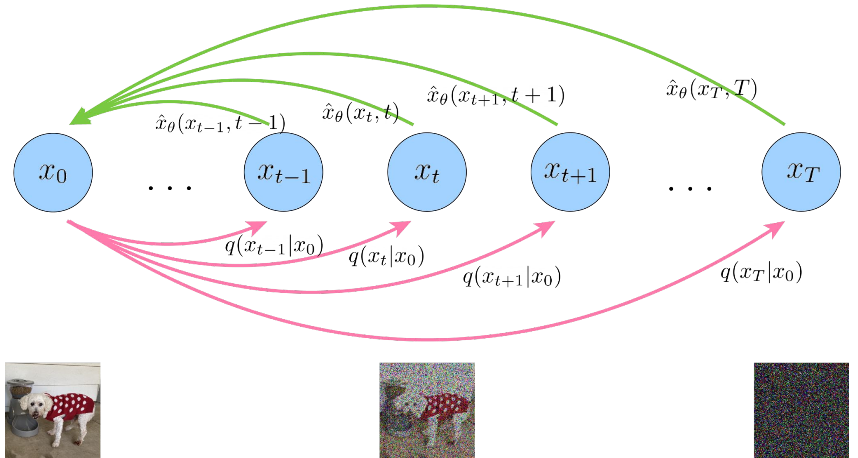


## Diffusion Models: A Quick Review

A Diffusion Model is:

- One NN that predicts a clean image from a noisy version of the image



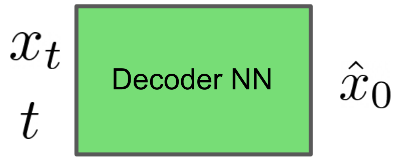


A naive question: *Why don't we just predict one step and be done?*

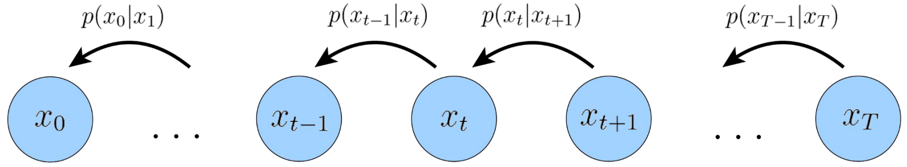
# Diffusion Models: A Quick Review

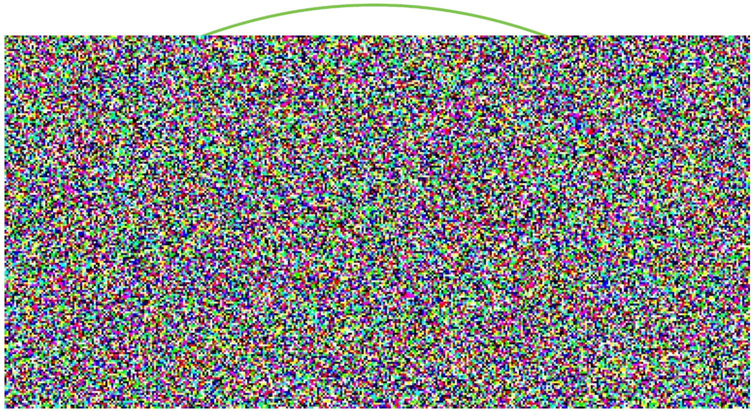
We have learned that a diffusion model is simply one neural network that predicts a clean image from a noisy image.

Objective: $$\arg\min_{\boldsymbol{\theta}} \ ||x_0 - \hat{x}_\theta(x_t, t)||^2$$



Sampling:





## Diffusion Models as a Noise Predictor 🔊

Recall that our objective is to predict $\hat{\boldsymbol{x}}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t) \approx \boldsymbol{x}_0$

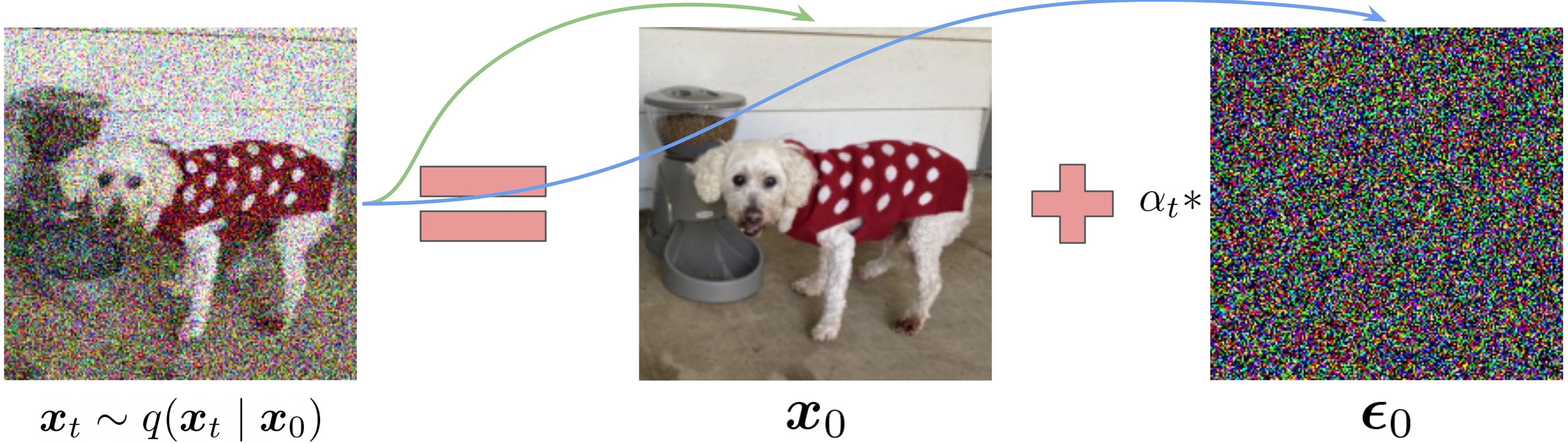
## Image 🖼️ and Noise 🔊?

They are the same!

What does it mean intuitively?

For arbitrary $\boldsymbol{x}_t \sim q(\boldsymbol{x}_t \mid \boldsymbol{x}_0)$, we can rewrite it as $\boldsymbol{x}_t = \boldsymbol{x}_0 + \alpha_t \boldsymbol{\epsilon}_0$

Predicting $\boldsymbol{x}_0$ *determines* $\boldsymbol{\epsilon}_0$ and vice-versa, since they sum to the same thing!



$\alpha_t *$
$\boldsymbol{x}_t \sim q(\boldsymbol{x}_t \mid \boldsymbol{x}_0)$
$\boldsymbol{x}_0$
$\boldsymbol{\epsilon}_0$

## Diffusion Models as a Score Predictor 💯

Recall that our objective is to predict $\hat{\boldsymbol{x}}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t) \approx \boldsymbol{x}_0$

## Three Different Interpretations

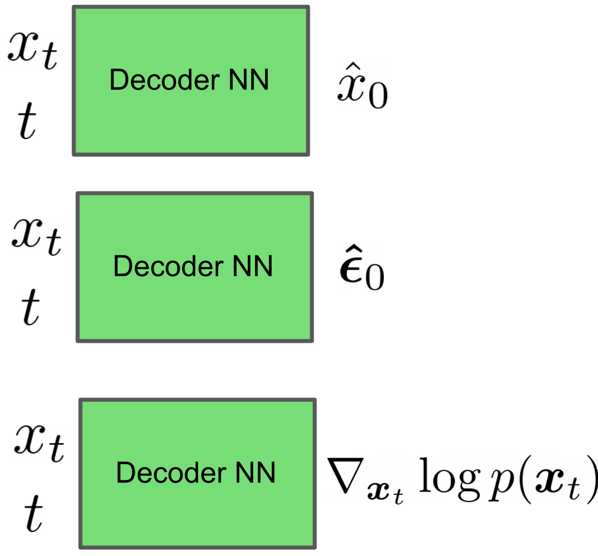
Last class we learned that a DiffModel can be implemented as a neural net that:

- Predicts original image $\hat{\boldsymbol{x}}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t) \approx \boldsymbol{x}_0$
- Predicts noise epsilon $\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t) \approx \boldsymbol{\epsilon}_0$

$$\boldsymbol{x}_t = \boldsymbol{x}_0 + \alpha_t \boldsymbol{\epsilon}_0$$

- Predicts score function $\boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t) \approx \nabla_{\boldsymbol{x}_t} \log p(\boldsymbol{x}_t)$

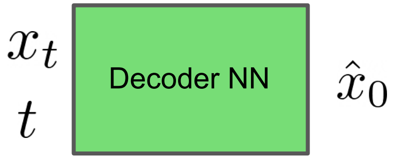
$$\boldsymbol{x}_0 \approx \boldsymbol{x}_t + \alpha_t^2 \nabla_{\boldsymbol{x}_t} \log p(\boldsymbol{x}_t)$$


$x_t$
$t$
Decoder NN
$\hat{x}_0$
$x_t$
$t$
Decoder NN
$\hat{\boldsymbol{\epsilon}}_0$
$x_t$
$t$
Decoder NN
$\nabla_{\boldsymbol{x}_t} \log p(\boldsymbol{x}_t)$

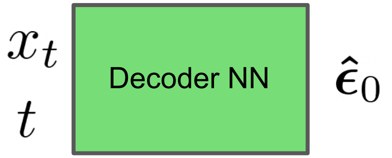
# Three Different Interpretations

Last class we learned that a DiffModel can be implemented as a neural net that:

- 🖼️ Predicts original image $\hat{\boldsymbol{x}}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t) \approx \boldsymbol{x}_0$

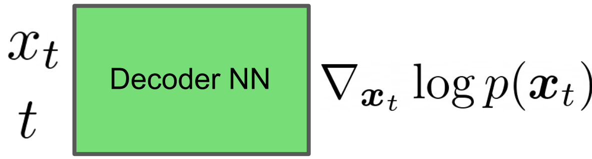


- 🔊 Predicts noise epsilon $\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t) \approx \boldsymbol{\epsilon}_0$

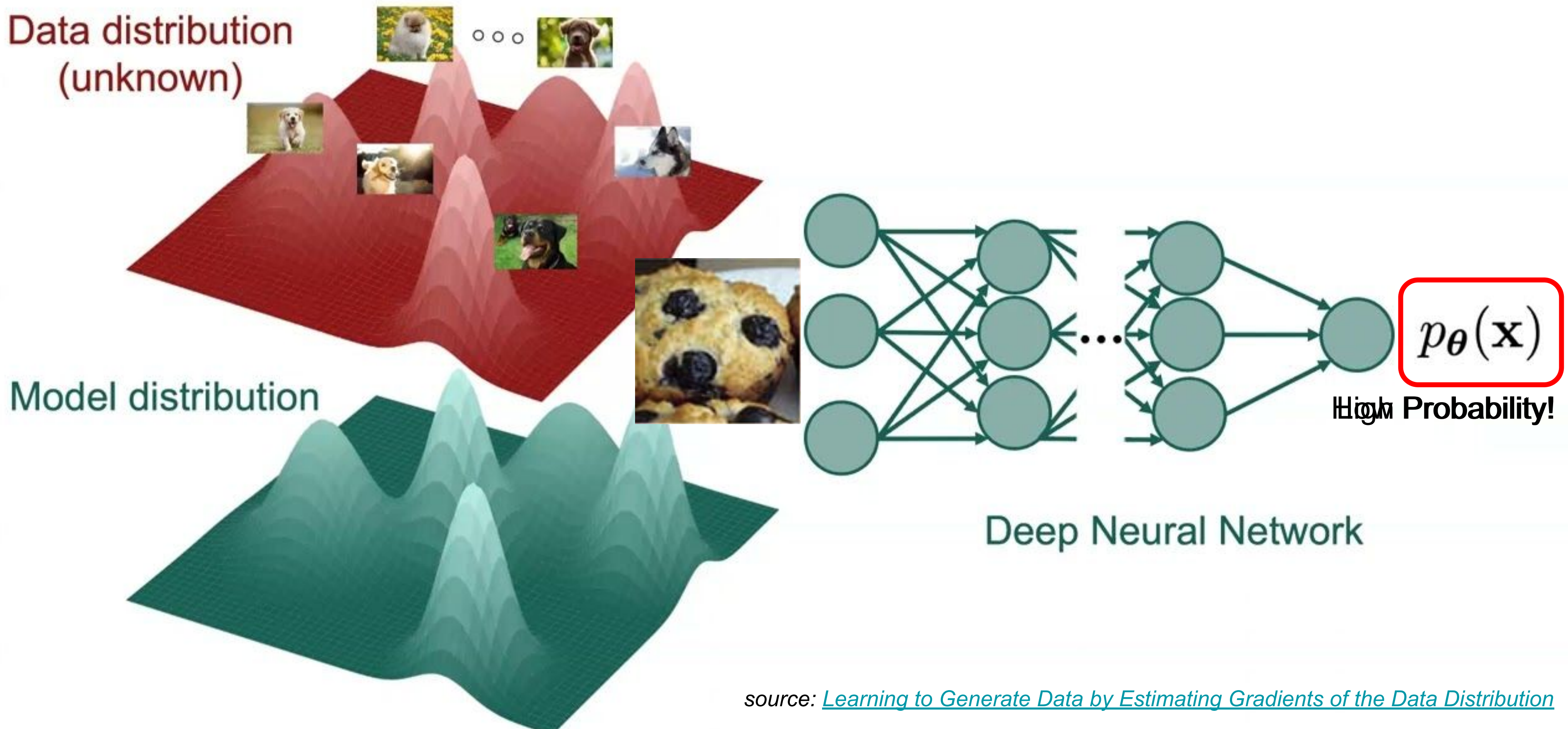


? ? ?

- 💯 Predicts score function $\boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t) \approx \nabla_{\boldsymbol{x}_t} \log p(\boldsymbol{x}_t)$



# Recap: Generative Modeling


Data distribution (unknown)
Model distribution
Deep Neural Network
$p_{\boldsymbol{\theta}}(\mathbf{x})$
High Low Probability!

source: *Learning to Generate Data by Estimating Gradients of the Data Distribution*

## Probability-based Generative Modeling

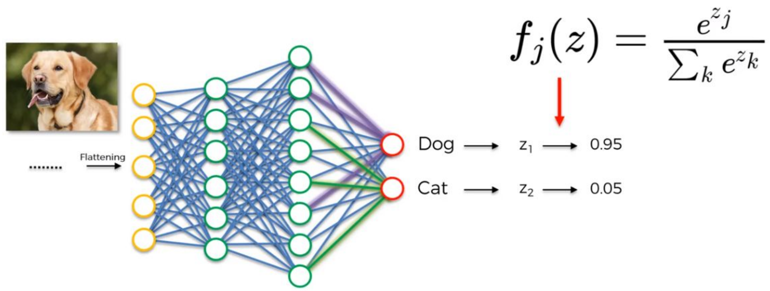
Why is modeling the **probability** hard in GenMo?

# Probability-based **Discriminative** Modeling

What about modeling the **probability** for classifiers?

Probabilities, and therefore model outputs, have to be:

- Non-negative: $0 \leq p_\theta(\boldsymbol{x})$
- Less than or equal to 1: $p_\theta(\boldsymbol{x}) \leq 1$
- Sum to 1 for the **entire** space: $\int_{\boldsymbol{x}} p_\theta(\boldsymbol{x})\, d\boldsymbol{x} = 1$



How did we do this for discriminative modeling?

## Probability-based Generative Modeling

Why is modeling the **probability** hard in GenMo?

Probabilities, and therefore model outputs, have to be:

- Non-negative: $0 \leq p_\theta(\boldsymbol{x})$
- Less than or equal to 1: $p_\theta(\boldsymbol{x}) \leq 1$
- Sum to 1 for the **entire** space: $\int_{\boldsymbol{x}} p_\theta(\boldsymbol{x})\, d\boldsymbol{x} = 1$

In GenMo we do not model the probability over all labels for an image…

We model the **probability of all possible images** - there's no way we can pass everything through our model and softmax over the result!

# Probability-based Generative Modeling

Why is modeling the **probability** hard in GenMo?

Probabilities, and therefore model outputs, have to be:

- Non-negative: $0 \leq p_\theta(\boldsymbol{x})$
- Less than or equal to 1: $p_\theta(\boldsymbol{x}) \leq 1$
- Sum to 1 for the **entire** space: $\int_{\boldsymbol{x}} p_\theta(\boldsymbol{x})\, d\boldsymbol{x} = 1$

This puts a lot of architectural burden on our network, to output ***valid*** probabilities!

## A simple observation…

All probability distributions can be written as:

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boxed{\frac{1}{Z_{\boldsymbol{\theta}}}}\,\boxed{e}^{\boxed{-}f_{\boldsymbol{\theta}}(\boldsymbol{x})}$$

This is a concept from thermodynamics, where the $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ is a flexible, unconstrained value called the ***energy***.

$Z_{\boldsymbol{\theta}}$ is a normalization constant, computed as: $Z_{\boldsymbol{\theta}} = \int_{\boldsymbol{x}} e^{-f_{\boldsymbol{\theta}}(\boldsymbol{x})} d\boldsymbol{x}$

## Energy-based Generative Modeling ⚡

Idea: Let’s just model the energy function $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ using a flexible neural network!

$f_{\boldsymbol{\theta}}(\boldsymbol{x})$ is called an *energy-based model (EBM)* or unnormalized probabilistic model.

How do we train our energy function?

- We can try interpreting it as a probability: $p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{Z_{\boldsymbol{\theta}}} e^{-f_{\boldsymbol{\theta}}(\boldsymbol{x})}$
- Then we can maximize log likelihood as before: $\max_{\theta} \sum_{i=1}^{N} \log p_{\theta}(\mathbf{x}_i)$

What is the problem with this? $Z_{\boldsymbol{\theta}} = \int_{\boldsymbol{x}} e^{-f_{\boldsymbol{\theta}}(\boldsymbol{x})} d\boldsymbol{x}$

Intractable for complex parameterizations!

# Another simple observation…

How do we avoid calculating the normalization constant?

Remember that $Z_{\boldsymbol{\theta}}$ is a *constant* that only depends on parameters $\theta$

Then, if we take the input gradient of the log of the probability:

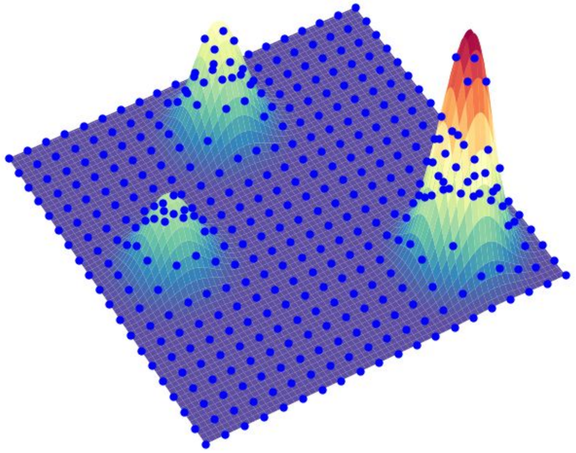
$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \qquad (\frac{1}{Z_{\boldsymbol{\theta}}} e^{-f_{\boldsymbol{\theta}}(\boldsymbol{x})})$$

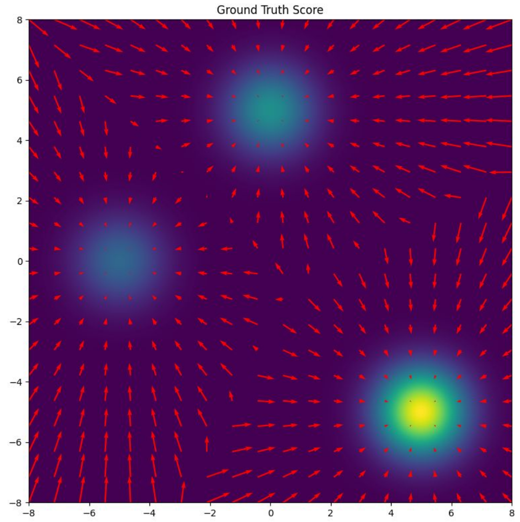
## Score Functions 💯

What are score functions?

$$\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x})$$

Intuitively, it describes how to move in **data space** to improve the (log) likelihood.



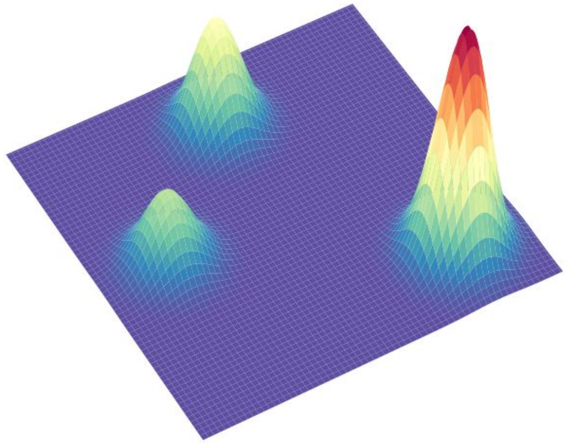


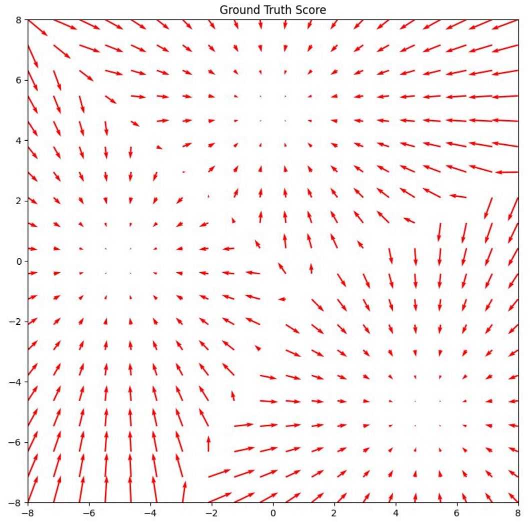
## Score Functions 💯

What are score functions?

$$\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x})$$

Intuitively, it describes how to move in **data space** to improve the (log) likelihood.





## Score-based Generative Modeling 💯

Idea: Let's just model the score function $\boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x})$ using a flexible neural network!

The score is still an unconstrained value, which is attractive to model directly.

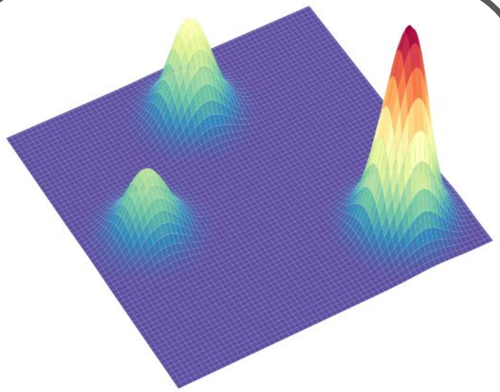
How do we train our score function?

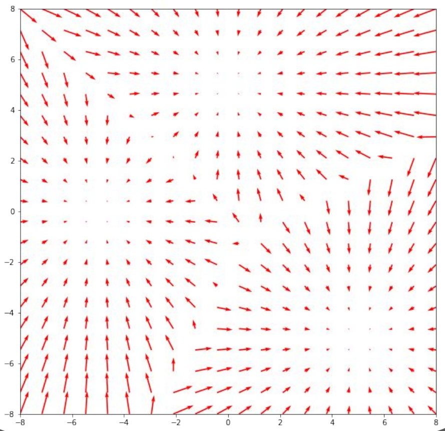
- Minimize the Fisher Divergence between the ground truth and predicted score

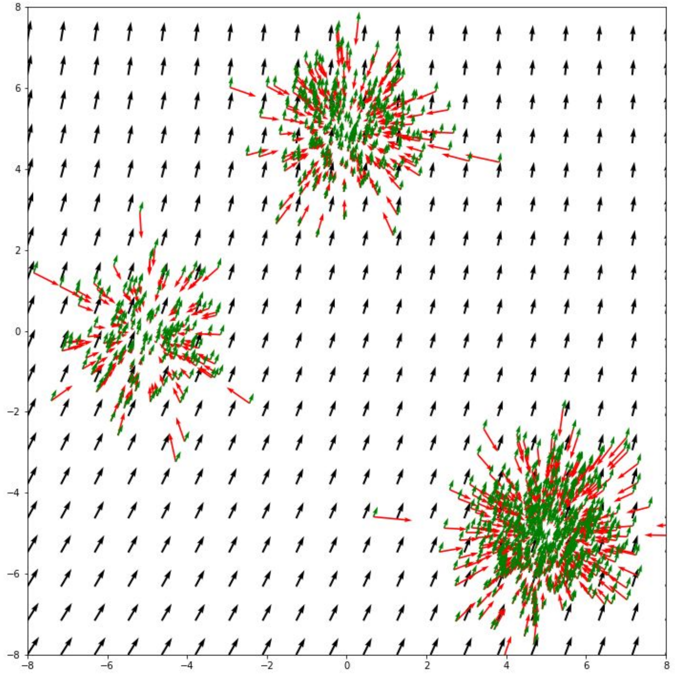
$$\mathbb{E}_{p(\boldsymbol{x})}\left[\|\nabla \log p(\boldsymbol{x}) - \boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x})\|_2^2\right]$$

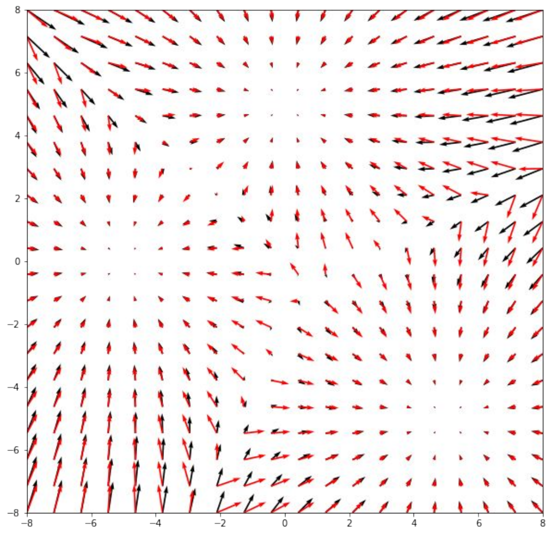
- Intuitively this is simply minimizing the L2-distance between our score model and the ground truth score

## Score-based Generative Modeling 💯





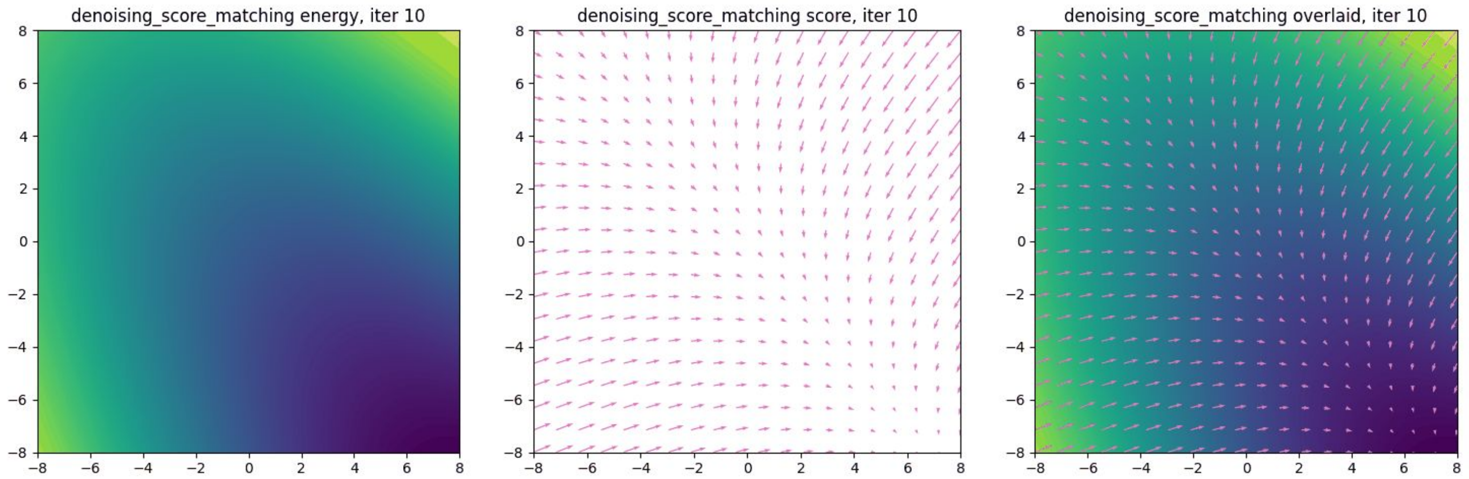




## Score-based Generative Modeling 💯

To drive the point home, score-based generative modeling is a way to implicitly model the energy function $f_{\boldsymbol{\theta}}(\boldsymbol{x})$

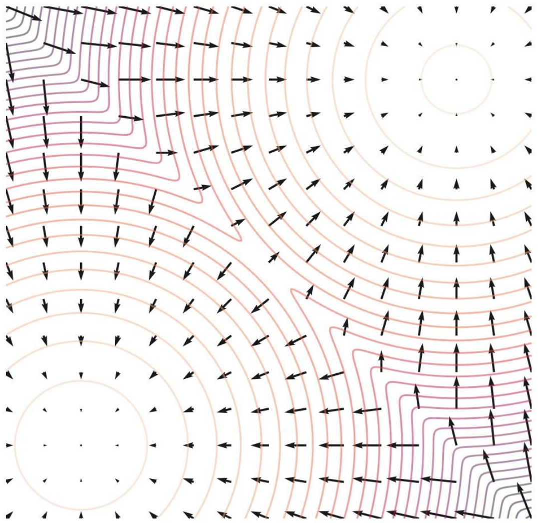
We can visualize the learned energy along with the score estimate below:

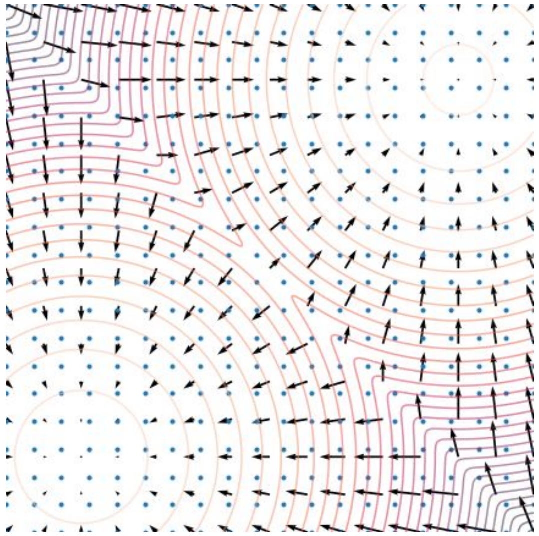


## Score-based Generative Modeling 💯

Once we have trained a score-based model $\boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x}) \approx \nabla_{\boldsymbol{x}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x})$, we can use an iterative procedure called Langevin dynamics to draw samples from it:

$$\boldsymbol{x}_{i+1} \leftarrow \boldsymbol{x}_i + c\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x}) + \sqrt{2c}\, \boldsymbol{\epsilon}_i, \quad i = 0, 1, \cdots, K,$$



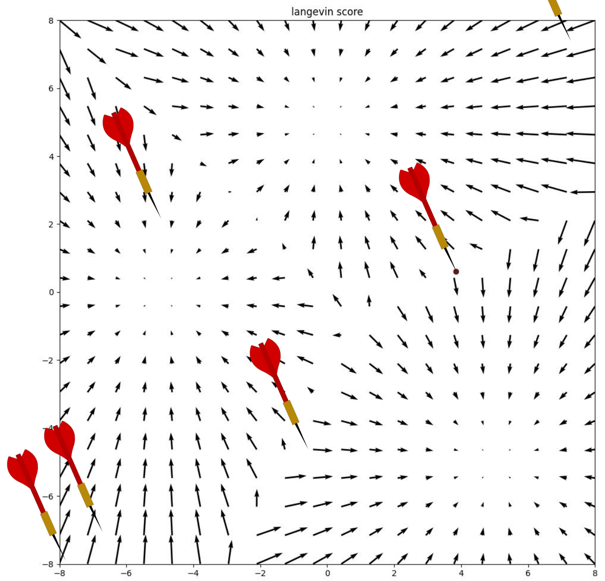


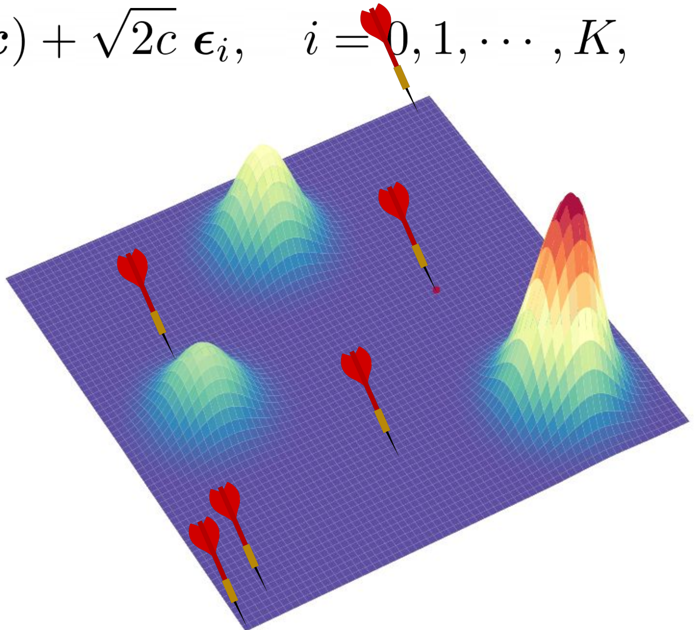
source: Generative Modeling by Estimating Gradients of the Data Distribution

## Score-based Generative Modeling 💯

Once we have trained a score-based model $\boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x}) \approx \nabla_{\boldsymbol{x}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x})$, we can use an iterative procedure called Langevin dynamics to draw samples from it:

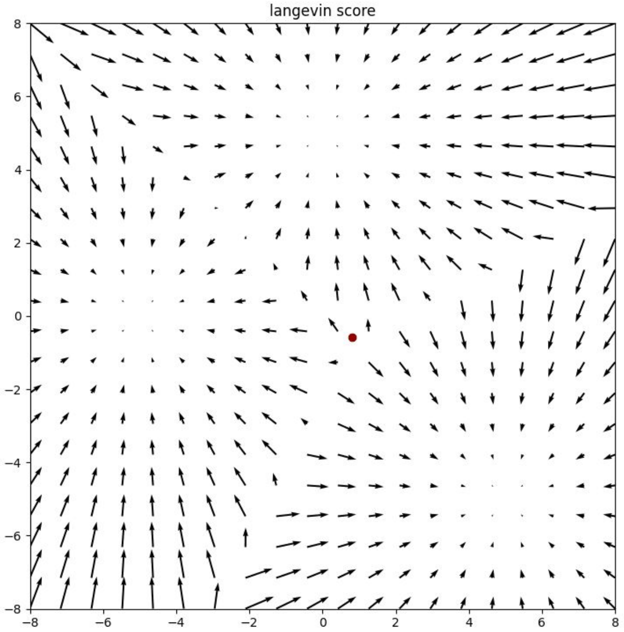
$$\boldsymbol{x}_{i+1} \leftarrow \boldsymbol{x}_i + c\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x}) + \sqrt{2c}\, \boldsymbol{\epsilon}_i, \quad i = 0, 1, \cdots, K,$$

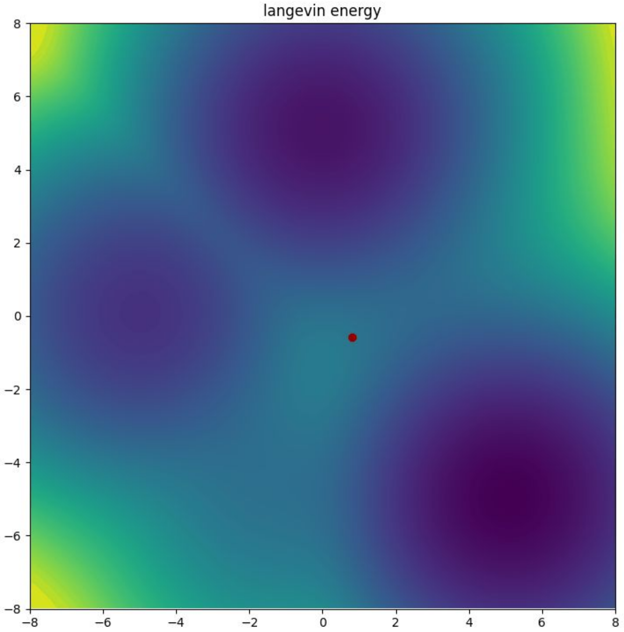


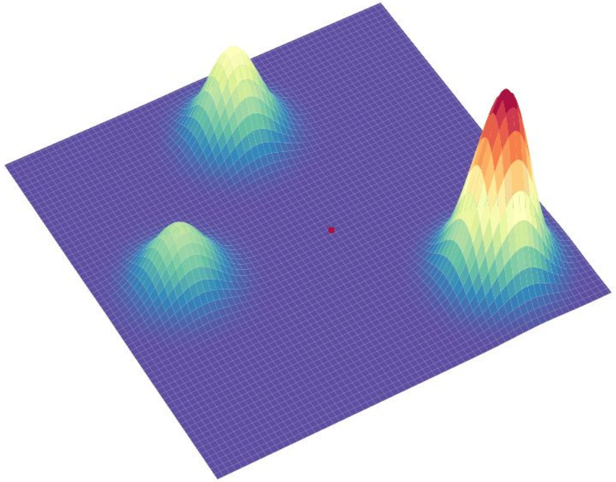


source: Generative Modeling by Estimating Gradients of the Data Distribution

## Score-based Generative Modeling 💯







## Score-based Generative Modeling 💯

What is the problem with this optimization objective?

$$\mathbb{E}_{p(\boldsymbol{x})}\left[\|\boxed{\nabla \log p(\boldsymbol{x})} - \boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x})\|_2^2\right]$$

## Score Matching

Fortunately, there are a class of methods called score matching that minimize the Fisher Divergence without needing to know the ground-truth score!

Ground Truth Score Matching: $\mathbb{E}_{p(\boldsymbol{x})}\left[\|\nabla \log p(\boldsymbol{x}) - \boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x})\|_2^2\right]$

Hyvarinen Score Matching: $\mathbb{E}_{p(\boldsymbol{x})}\left[\operatorname{tr}(\nabla_{\boldsymbol{x}} s_{\boldsymbol{\theta}}(\boldsymbol{x})) + \frac{1}{2}\|s_{\boldsymbol{\theta}}(\boldsymbol{x})\|_2^2\right]$

*Sliced Score Matching*: $\mathbb{E}_{p_{\mathrm{v}}}\mathbb{E}_{p(\boldsymbol{x})}\left[\mathbf{v}^\intercal \nabla_{\boldsymbol{x}} s_{\boldsymbol{\theta}}(\boldsymbol{x})\mathbf{v} + \frac{1}{2}\|s_{\boldsymbol{\theta}}(\boldsymbol{x})\|_2^2\right]$

*Denoising Score Matching*: $\mathbb{E}_{q_\sigma(\tilde{\mathbf{x}}|\mathbf{x})p(\mathbf{x})}[\|\nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) - \mathbf{s}_\theta(\tilde{\mathbf{x}})\|_2^2]$

## Hyvarinen Score Matching

Hyvarinen (2005) utilized integration by parts to remove the unknown $\nabla \log p(\boldsymbol{x})$

$$\mathbb{E}_{p(\boldsymbol{x})}\left[\|\nabla \log p(\boldsymbol{x}) - \boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x})\|_2^2\right]$$

$$L(\theta) = \frac{1}{2}\int p(\boldsymbol{x})\left[s_\theta(\boldsymbol{x})^\top s_\theta(\boldsymbol{x}) - 2s_\theta(\boldsymbol{x})^\top \nabla \log p(\boldsymbol{x}) + \nabla \log p(\boldsymbol{x})^\top \nabla \log p(\boldsymbol{x})\right] d\boldsymbol{x}$$

$$= \frac{1}{2}\int p(\boldsymbol{x})\left[s_\theta(\boldsymbol{x})^\top s_\theta(\boldsymbol{x}) - 2s_\theta(\boldsymbol{x})^\top \nabla \log p(\boldsymbol{x})\right] d\boldsymbol{x}$$

log-deriv trick:

$$p(\boldsymbol{x})\nabla \log p(\boldsymbol{x}) = \nabla p(\boldsymbol{x}$$

$$\int p(\boldsymbol{x}) s_\theta(\boldsymbol{x})^\top \nabla \log p(\boldsymbol{x}) d\boldsymbol{x} = \int s_\theta(\boldsymbol{x})^\top \nabla p(\boldsymbol{x}) d\boldsymbol{x}$$

$$= \left[p(\boldsymbol{x}) s_\theta(\boldsymbol{x})\right]_{-\infty}^{\infty} - \int p(\boldsymbol{x}) \nabla_{\boldsymbol{x}} s_\theta(\boldsymbol{x}) d\boldsymbol{x}$$

$$\mathbb{E}_{p(\boldsymbol{x})}\left[\operatorname{tr}(\nabla_{\boldsymbol{x}} s_{\boldsymbol{\theta}}(\boldsymbol{x})) + \frac{1}{2}\|s_{\boldsymbol{\theta}}(\boldsymbol{x})\|_2^2\right]$$

## Hyvarinen Score Matching

$$\mathbb{E}_{p(\boldsymbol{x})}\left[\operatorname{tr}(\nabla_{\boldsymbol{x}} s_{\boldsymbol{\theta}}(\boldsymbol{x})) + \frac{1}{2}\|s_{\boldsymbol{\theta}}(\boldsymbol{x})\|_2^2\right]$$

We have gotten rid of unknown $\nabla \log p(\boldsymbol{x})$ 🎉

But now we have to compute $\operatorname{tr}(\nabla_{\boldsymbol{x}} s_\theta(\boldsymbol{x}))$

When our data has high dimensionality, this is **not cheap** - many nested backprops!

$$\nabla_{\mathbf{x}} s_\theta(\mathbf{x}) = \begin{pmatrix} \boxed{\frac{\partial s_{\theta,1}(\mathbf{x})}{\partial x_1}} & \frac{\partial s_{\theta,1}(\mathbf{x})}{\partial x_2} & \frac{\partial s_{\theta,1}(\mathbf{x})}{\partial x_3} \\ \frac{\partial s_{\theta,2}(\mathbf{x})}{\partial x_1} & \boxed{\frac{\partial s_{\theta,2}(\mathbf{x})}{\partial x_2}} & \frac{\partial s_{\theta,2}(\mathbf{x})}{\partial x_3} \\ \frac{\partial s_{\theta,3}(\mathbf{x})}{\partial x_1} & \frac{\partial s_{\theta,3}(\mathbf{x})}{\partial x_2} & \boxed{\frac{\partial s_{\theta,3}(\mathbf{x})}{\partial x_3}} \end{pmatrix}$$

*figure link*

# Sliced Score Matching

Song (2019) utilized random projections to estimate the expensive $\operatorname{tr}(\nabla_{\boldsymbol{x}} s_\theta(\boldsymbol{x}))$

Hyvarinen: $\mathbb{E}_{p(\boldsymbol{x})}\left[\operatorname{tr}(\nabla_{\boldsymbol{x}} s_{\boldsymbol{\theta}}(\boldsymbol{x})) + \frac{1}{2}\|s_{\boldsymbol{\theta}}(\boldsymbol{x})\|_2^2\right]$

Sliced Score Matching: $\mathbb{E}_{p_{\mathrm{v}}}\mathbb{E}_{p(\boldsymbol{x})}\left[\mathbf{v}^\intercal \nabla_{\boldsymbol{x}} s_{\boldsymbol{\theta}}(\boldsymbol{x})\mathbf{v} + \frac{1}{2}\|s_{\boldsymbol{\theta}}(\boldsymbol{x})\|_2^2\right]$

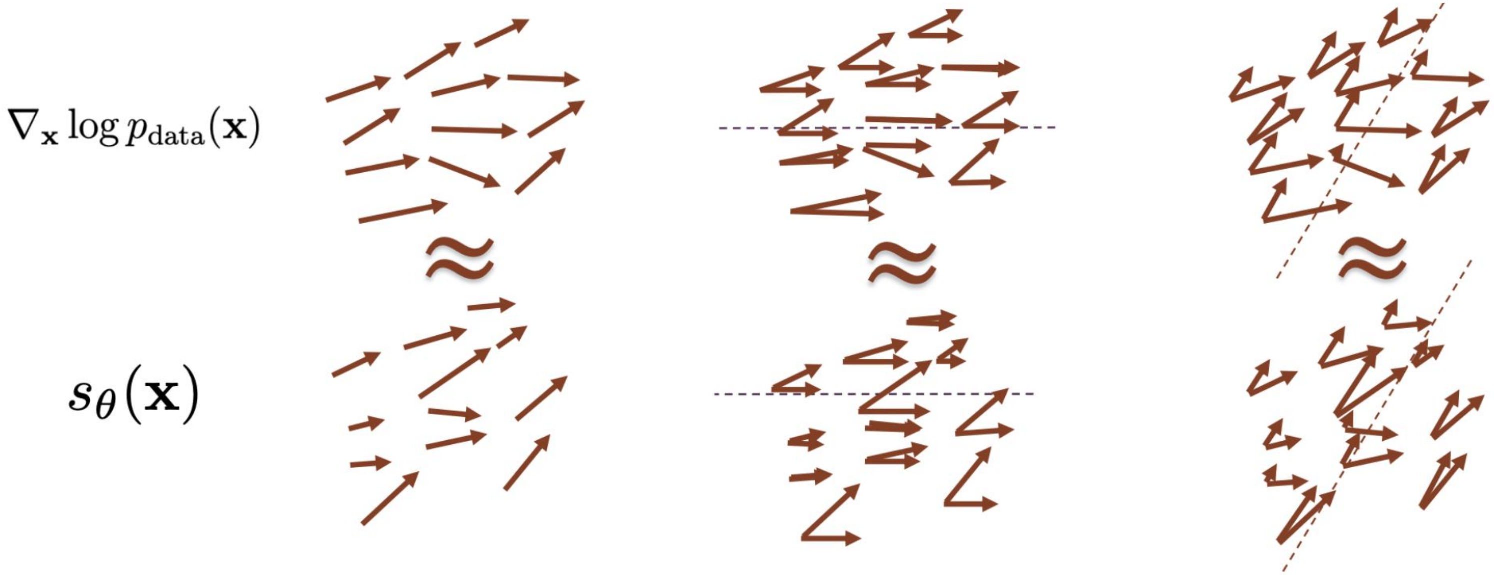
Intuition - when we project to a lower dimension, the problem becomes tractable.

- $p_{\mathbf{v}}$ is a simple distribution of random vectors, e.g. the multivariate std. normal.

# Sliced Score Matching

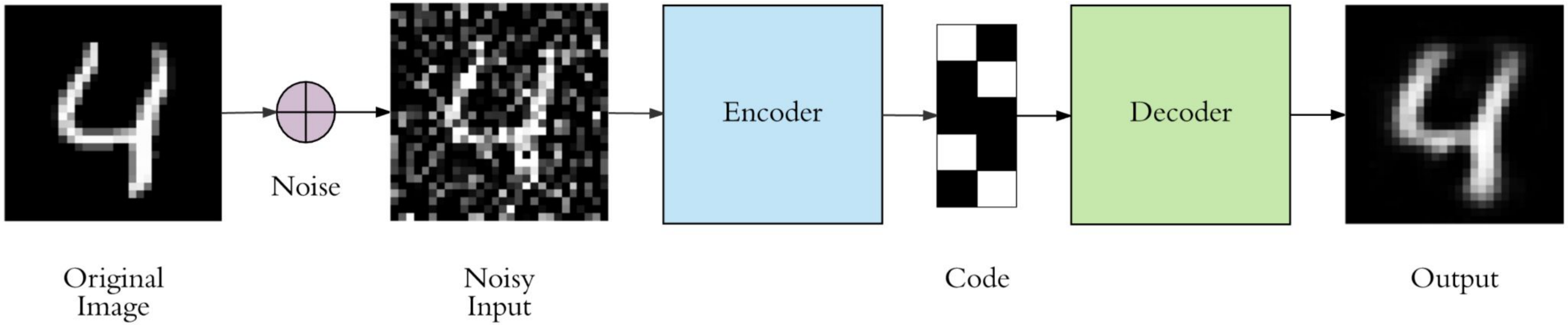
Song (2019) utilized random projections to estimate the expensive $\operatorname{tr}(\nabla_{\boldsymbol{x}} s_\theta(\boldsymbol{x}))$

$$\mathbb{E}_{p_{\mathrm{v}}} \mathbb{E}_{p(\boldsymbol{x})} \left[ \mathbf{v}^\intercal \nabla_{\boldsymbol{x}} s_{\boldsymbol{\theta}}(\boldsymbol{x}) \mathbf{v} + \frac{1}{2} \| s_{\boldsymbol{\theta}}(\boldsymbol{x}) \|_2^2 \right]$$



## Denoising Autoencoders

Denoising Autoencoders work as follows:



The rationale is that this minimizes “memorization” - the input is corrupted from the start!

## Denoising Score Matching

Vincent (2010) proved that matching the score for a noisy perturbation of the input can also minimize the score for the ground truth estimator.

The intuition is that following the gradient of some simple Gaussian perturbation of an input should move us towards the original clean input.

$$\mathbb{E}_{q_\sigma(\tilde{\mathbf{x}}|\mathbf{x})p(\mathbf{x})}[||\nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) - \mathbf{s}_\theta(\tilde{\mathbf{x}})||_2^2]$$

With a simple gaussian for $q_\sigma(\tilde{\boldsymbol{x}}|\boldsymbol{x}) = \mathcal{N}(\tilde{\boldsymbol{x}}; \boldsymbol{x}, \sigma^2)$

We know that indeed: $\nabla_{\tilde{\boldsymbol{x}}} \log q_\sigma(\tilde{\boldsymbol{x}}|\boldsymbol{x}) = \frac{1}{\sigma^2}(\tilde{\boldsymbol{x}} - \boldsymbol{x})$

## Denoising Score Matching

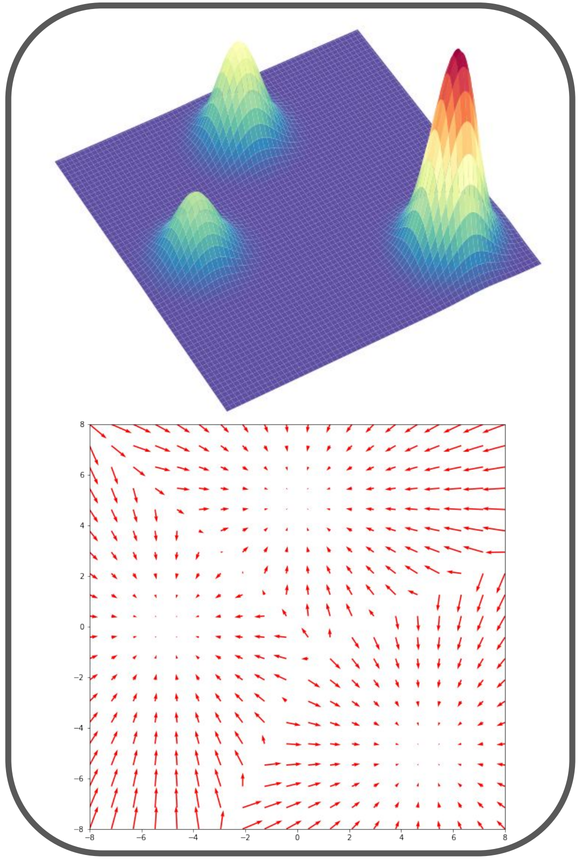
$$\mathbb{E}_{q_\sigma(\tilde{\mathbf{x}}|\mathbf{x})p(\mathbf{x})}[\|\nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) - \mathbf{s}_\theta(\tilde{\mathbf{x}})\|_2^2]$$

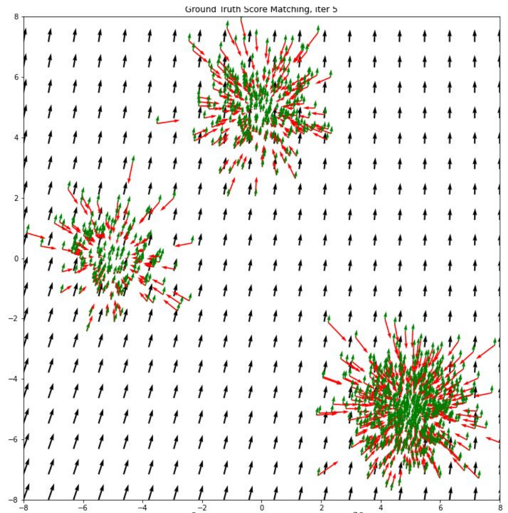
A simple algorithm:

- Take in your input sample
- Perturb it with some Gaussian noise
- Compute the score estimate for the noisy sample
- Compare it with the ground truth score computed by the noising Gaussian

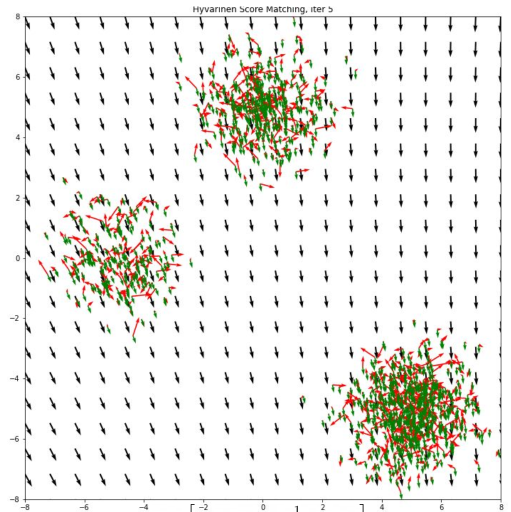
$$\nabla_{\tilde{\boldsymbol{x}}} \log q_\sigma(\tilde{\boldsymbol{x}}|\boldsymbol{x}) = \frac{1}{\sigma^2}(\tilde{\boldsymbol{x}} - \boldsymbol{x})$$

# Score Matching

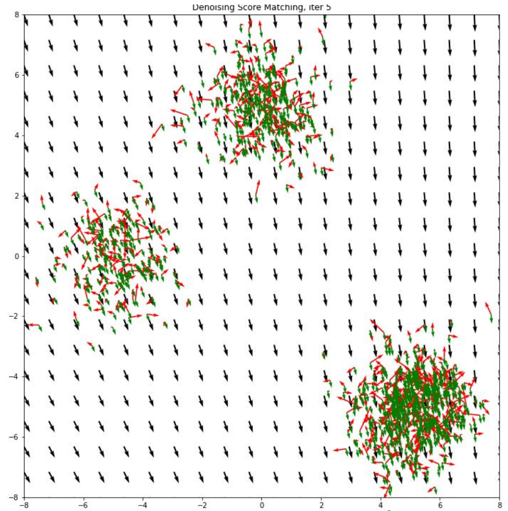




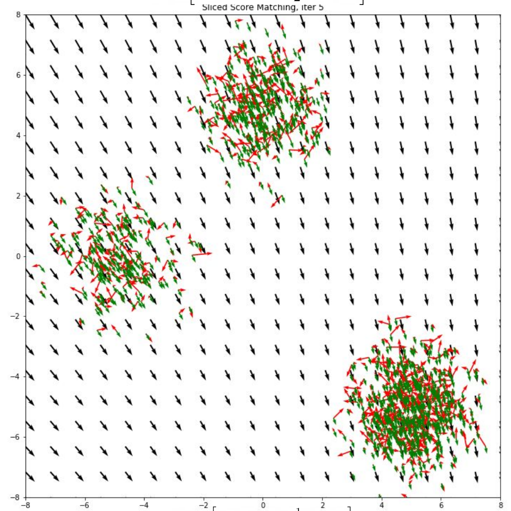
$$\mathbb{E}_{p(\boldsymbol{x})}\left[\|s_{\boldsymbol{\theta}}(\boldsymbol{x})-\nabla_{\boldsymbol{x}}\log p(\boldsymbol{x})\|_2^2\right]$$



$$\mathbb{E}_{p(\boldsymbol{x})}\left[\mathrm{tr}(\nabla_{\boldsymbol{x}} s_{\boldsymbol{\theta}}(\boldsymbol{x}))+\frac{1}{2}\|s_{\boldsymbol{\theta}}(\boldsymbol{x})\|_2^2\right]$$

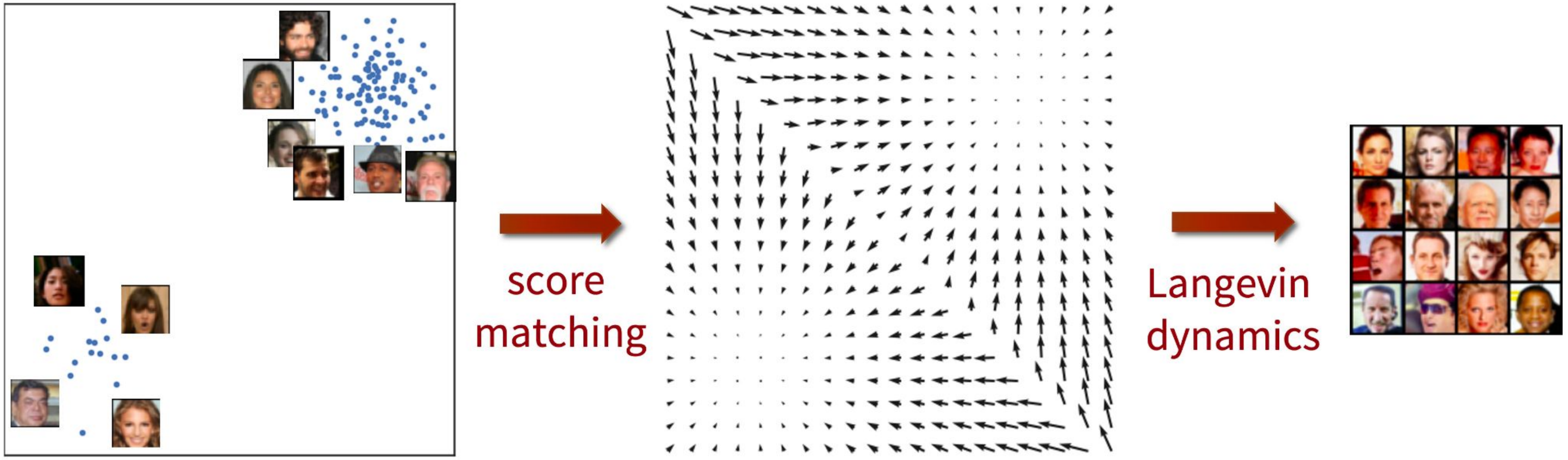


$$\mathbb{E}_{q_\sigma(\tilde{\boldsymbol{x}}|\boldsymbol{x})p(\boldsymbol{x})}\left[\|s_{\boldsymbol{\theta}}(\tilde{\boldsymbol{x}})-\nabla_{\tilde{\boldsymbol{x}}}\log q_\sigma(\tilde{\boldsymbol{x}}|\boldsymbol{x})\|_2^2\right]$$



$$\mathbb{E}_{p_{\mathbf{v}}}\mathbb{E}_{p(\boldsymbol{x})}\left[\mathbf{v}^\intercal\nabla_{\boldsymbol{x}} s_{\boldsymbol{\theta}}(\boldsymbol{x})\mathbf{v}+\frac{1}{2}\|s_{\boldsymbol{\theta}}(\boldsymbol{x})\|_2^2\right]$$

# Score-based Generative Modeling 💯



Data samples

$$\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\} \overset{\text{i.i.d.}}{\sim} p(\mathbf{x})$$

Scores

$$\mathbf{s}_\theta(\mathbf{x}) \approx \nabla_\mathbf{x} \log p(\mathbf{x})$$

New samples

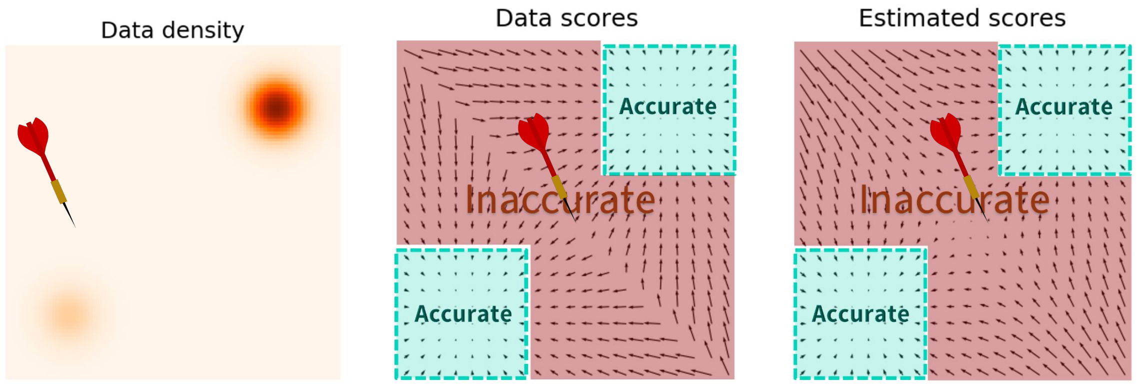
source: Generative Modeling by Estimating Gradients of the Data Distribution

# Score-based Generative Modeling 💯

What is the problem with vanilla score-matching?

$$\mathbb{E}_{p(\boldsymbol{x})}[\|\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x}) - \boldsymbol{s}_\theta(\boldsymbol{x})\|_2^2] = \int p(\boldsymbol{x})\|\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x}) - \boldsymbol{s}_\theta(\boldsymbol{x})\|_2^2 \mathrm{d}\boldsymbol{x}$$

Our model of the score will not learn the low-density regions well

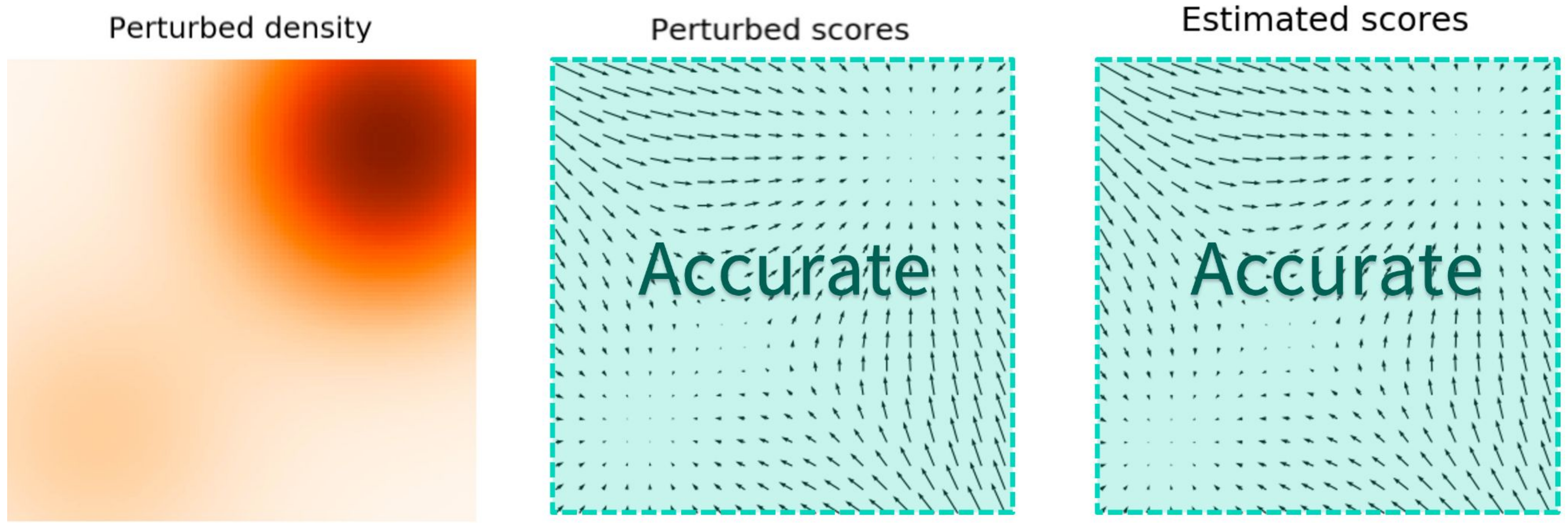


source: Generative Modeling by Estimating Gradients of the Data Distribution

## Score-based Generative Modeling 💯

What is the solution for vanilla score-matching?

- Adding Gaussian noise!

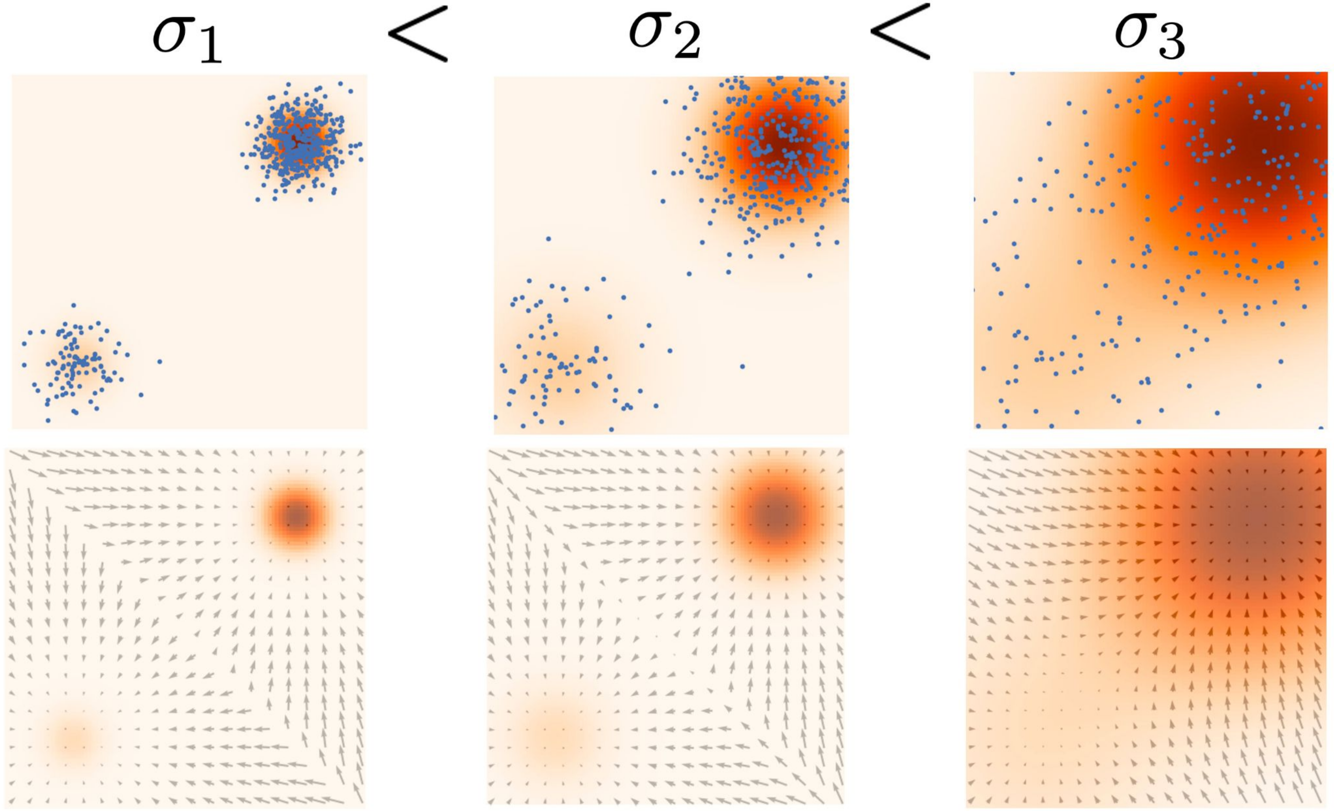


source: [Generative Modeling by Estimating Gradients of the Data Distribution]()

## Score-based Generative Modeling 💯

How do we choose an appropriate noise scale for the perturbation process?

## Score-based Generative Modeling 💯



source: Generative Modeling by Estimating Gradients of the Data Distribution

## Score-based Generative Modeling 💯

Now, we estimate the score function of each noise-perturbed distribution

$$\boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x}, t) \approx \nabla_{\boldsymbol{x}_t} \log p_{\sigma_t}(\boldsymbol{x}_t)$$
$$\text{for all } t = 1, 2, \cdots, T$$

We model it as a neural network, called the **Noise Conditional Score Network**

The training objective is a weighted sum of Fisher divergences for all noise scales:

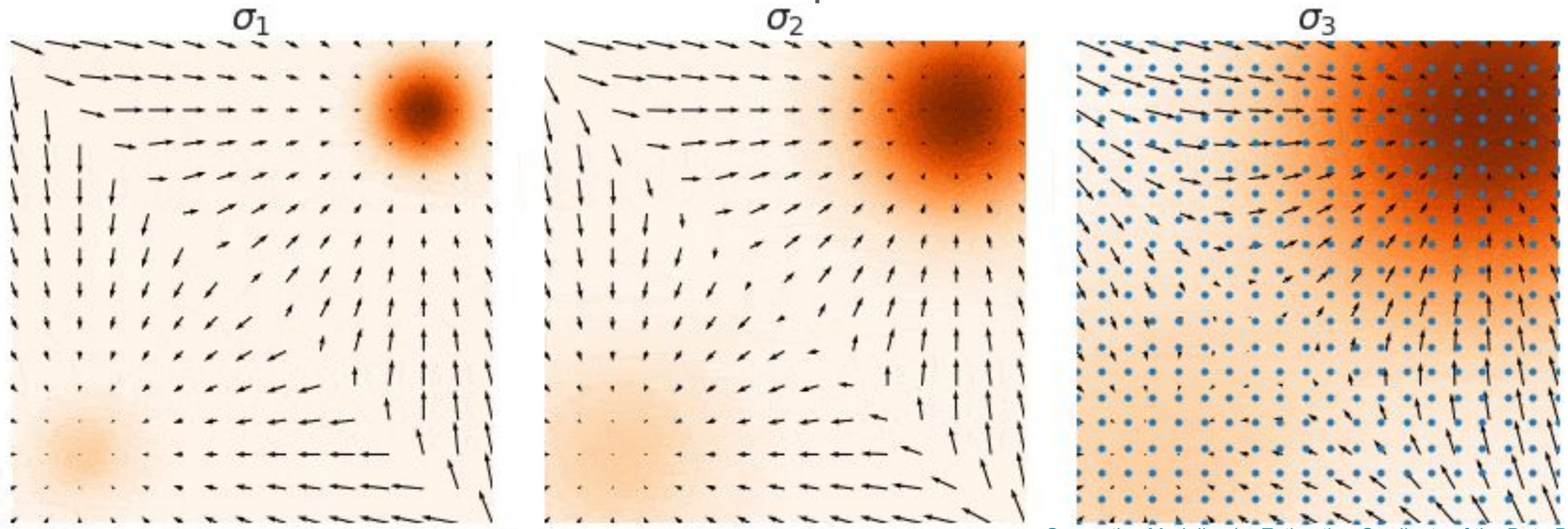
$$\arg\min_{\boldsymbol{\theta}} \sum_{t=1}^{T} \lambda(t) \mathbb{E}_{p_{\sigma_t}(\boldsymbol{x}_t)} \left[ \|\nabla \log p_{\sigma_t}(\boldsymbol{x}_t) - \boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x}, t)\|_2^2 \right]$$

source: Generative Modeling by Estimating Gradients of the Data Distribution

## Score-based Generative Modeling 💯

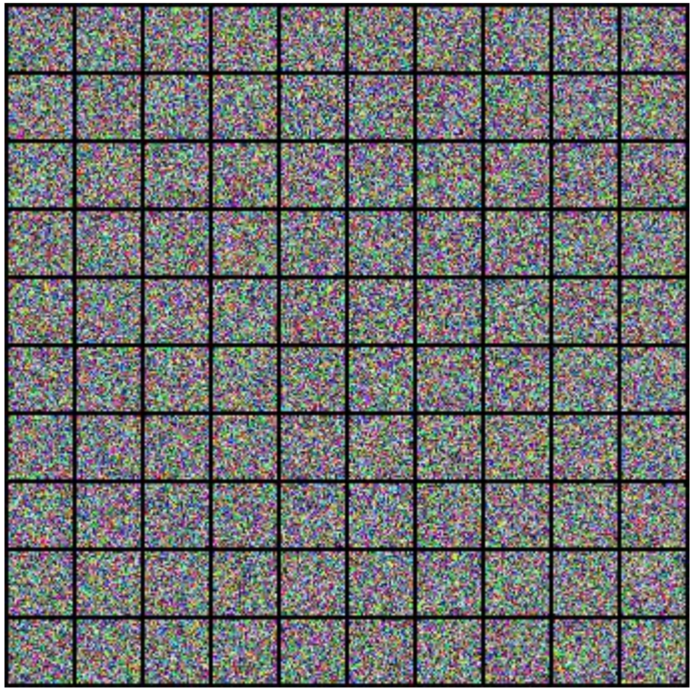
Sampling is done using annealed Langevin Dynamics

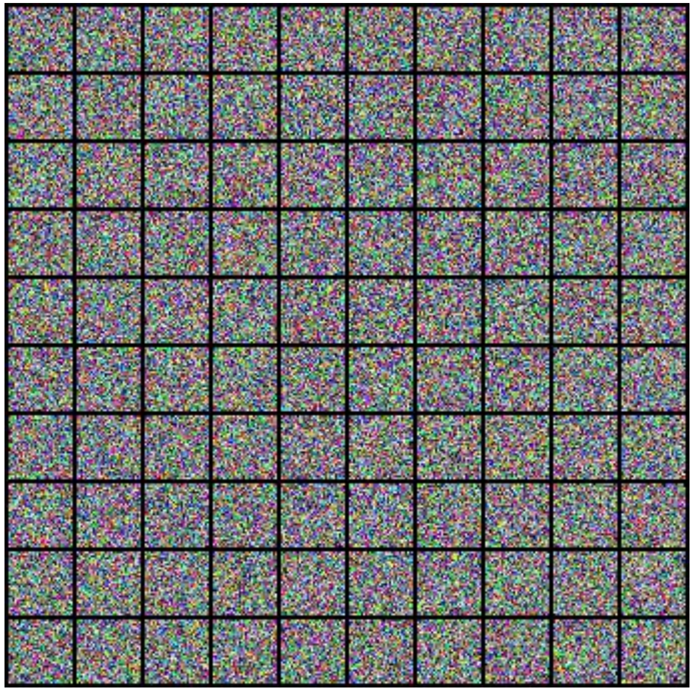
- Running Langevin dynamics for each noise level in sequence, initializing the next noise level with the results of the previous one.



source: Generative Modeling by Estimating Gradients of the Data Distribution

## Examples!

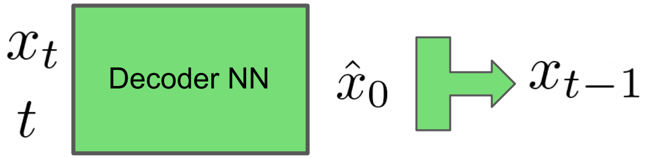

Celeb-A


CIFAR-10

*source: Generative Modeling by Estimating Gradients of the Data Distribution*

# Review: Variational Diffusion Models

We want to learn a denoising decoder:



$$\hat{x}_{t-1} = \hat{x}_0 + \alpha_{t-1} * \epsilon$$

*reparam. trick!*

$\epsilon \sim \mathcal{N}(0, \mathrm{I})$

But what is the form of $x_{t-1}$?

Recall that:

$$\boxed{\begin{aligned} q(x_{t-1}|x_0) &= \mathcal{N}(x_{t-1}|x_0, \alpha_{t-1}^2 \mathrm{I}) \\ \therefore x_{t-1} &= x_0 + \alpha_{t-1} * \epsilon \end{aligned}}$$

*reparam. trick!*

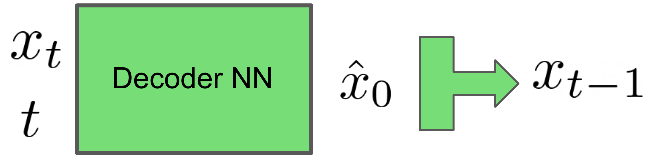
$\epsilon \sim \mathcal{N}(0, \mathrm{I})$

*Do we really need to predict* $\sigma_{dec}$*?*

*What is the ground truth signal for* $\mu_{dec}$*?*

# Review: Variational Diffusion Models

We want to learn a denoising decoder:



where ground truth denoising sample is: $\boldsymbol{x}_{t-1} = \enclose{circle}{\boldsymbol{x}_0} + \alpha_{t-1} * \boldsymbol{\epsilon}$

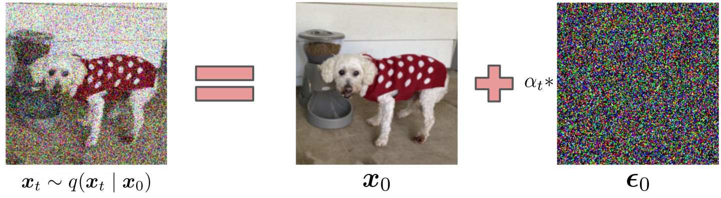
*reparam. trick!*

$\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$

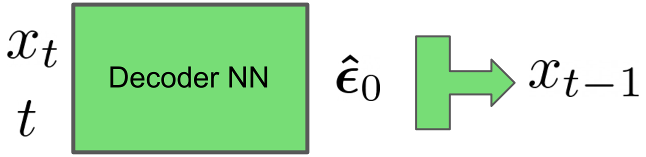
Loss Objective:

$$\arg\min_{\boldsymbol{\theta}} \sum_{t=1}^{T} \| \boldsymbol{x}_0 - \hat{\boldsymbol{x}}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t) \|_2^2$$

# Review: Variational Diffusion Models


$\boldsymbol{x}_t \sim q(\boldsymbol{x}_t \mid \boldsymbol{x}_0)$
$\boldsymbol{x}_0$
$\alpha_t *$
$\boldsymbol{\epsilon}_0$

We want to learn a denoising decoder:



*reparam. trick!*

where ground truth denoising sample is: $\boldsymbol{x}_{t-1} = \boxed{\boldsymbol{x}_0} + \alpha_{t-1} * \boldsymbol{\epsilon}$ $\quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$

which is equivalent to: $\boldsymbol{x}_{t-1} = \boxed{\boldsymbol{x}_t - \alpha_t * \boldsymbol{\epsilon_0}} + \alpha_{t-1} * \boldsymbol{\epsilon}$ (noise prediction)

Loss Objective:

$$\arg\min_{\boldsymbol{\theta}} \sum_{t=1}^{T} \|\boldsymbol{\epsilon}_0 - \hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)\|_2^2$$

## Review: Score 💯 and Noise 🔊?

There is a relationship between the score and the noise, which we can derive by equating Tweedie's formula with the Reparameterization Trick.

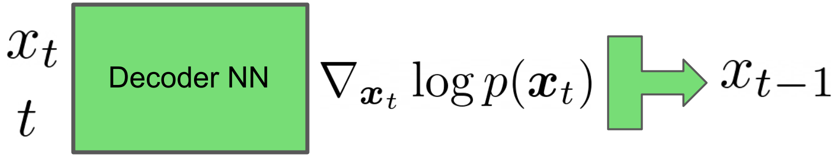
$$\begin{aligned} \boldsymbol{x}_0 = \boldsymbol{x}_t + \alpha_t^2 \nabla \log p(\boldsymbol{x}_t) &= \boldsymbol{x}_t - \alpha_t \boldsymbol{\epsilon}_0 \\ \therefore \alpha_t^2 \nabla \log p(\boldsymbol{x}_t) &= -\alpha_t \boldsymbol{\epsilon}_0 \\ \nabla \log p(\boldsymbol{x}_t) &= -\frac{1}{\alpha_t} \boldsymbol{\epsilon}_0 \end{aligned}$$

Intuitively, the direction to move in data space towards a natural image is the negative noise term that was added.

# Review: Variational Diffusion Models

$\boldsymbol{x}_0 \approx \boldsymbol{x}_t + \alpha_t^2 \nabla_{\boldsymbol{x}_t} \log p(\boldsymbol{x}_t)$

We want to learn a denoising decoder:



*reparam. trick!*

$\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$

where ground truth denoising sample is: $\boldsymbol{x}_{t-1} = \boxed{\boldsymbol{x}_0} + \alpha_{t-1} * \boldsymbol{\epsilon}$

which is equivalent to: $\boldsymbol{x}_{t-1} = \boldsymbol{x}_t - \alpha_t * \boldsymbol{\epsilon}_{\mathbf{0}} + \alpha_{t-1} * \boldsymbol{\epsilon}$

which is equivalent to: $\boldsymbol{x}_{t-1} = \boxed{\boldsymbol{x}_t + \alpha_t^2 * \boxed{\nabla_{\boldsymbol{x}_t} \log p(\boldsymbol{x}_t)}} + \alpha_{t-1} * \boldsymbol{\epsilon}$

(score prediction)

Loss Objective: $\boxed{\arg\min_{\boldsymbol{\theta}} \sum_{t=1}^{T} \|\nabla_{\boldsymbol{x}_t} \log p(\boldsymbol{x}_t) - \hat{\boldsymbol{s}}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)\|_2^2}$

## Unifying Two Interpretations - Training Objective

The Hierarchical VAE interpretation of diffusion models shows that we can use a network to model the score function at arbitrary noise corruptions, learned by:

$$\arg\min_{\boldsymbol{\theta}} \sum_{t=1}^{T} \|\nabla_{\boldsymbol{x}_t} \log p(\boldsymbol{x}_t) - \hat{\boldsymbol{s}}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)\|_2^2$$

Score-based generative modeling also uses a network to model the score function at arbitrary levels of Gaussian noise corruption, learned by:

$$\arg\min_{\boldsymbol{\theta}} \sum_{t=1}^{T} \lambda(t) \mathbb{E}_{p_{\sigma_t}(\boldsymbol{x}_t)} \left[ \|\nabla \log p_{\sigma_t}(\boldsymbol{x}_t) - \boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x}, t)\|_2^2 \right]$$

# Review: Variational Diffusion Models

$\boldsymbol{x}_0 \approx \boldsymbol{x}_t + \alpha_t^2 \nabla_{\boldsymbol{x}_t} \log p(\boldsymbol{x}_t)$

We want to learn a denoising decoder:

$$\begin{matrix} x_t \\ t \end{matrix} \; \boxed{\text{Decoder NN}} \; \nabla_{\boldsymbol{x}_t} \log p(\boldsymbol{x}_t) \Rightarrow x_{t-1}$$

*reparam. trick!*

where ground truth denoising sample is: $\boldsymbol{x}_{t-1} = \boxed{\boldsymbol{x}_0} + \alpha_{t-1} * \boldsymbol{\epsilon}$ $\qquad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$

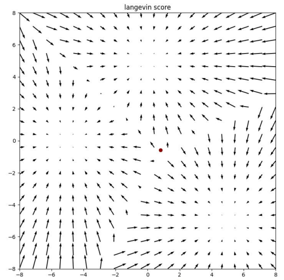
which is equivalent to: $\boldsymbol{x}_{t-1} = \boldsymbol{x}_t - \alpha_t * \boldsymbol{\epsilon_0} + \alpha_{t-1} * \boldsymbol{\epsilon}$

which is equivalent to: $\boxed{\boldsymbol{x}_{t-1} = \boxed{\boldsymbol{x}_t + \alpha_t^2 * \nabla_{\boldsymbol{x}_t} \log p(\boldsymbol{x}_t)} + \alpha_{t-1} * \boldsymbol{\epsilon}}$

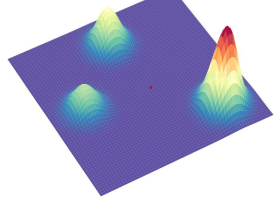
(score prediction)

Loss Objective:

$$\arg\min_{\boldsymbol{\theta}} \sum_{t=1}^{T} \|\nabla_{\boldsymbol{x}_t} \log p(\boldsymbol{x}_t) - \hat{\boldsymbol{s}}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)\|_2^2$$



# Unifying Two Interpretations - Sampling Procedure



Let's take a closer look at Langevin Dynamics:

$$\boldsymbol{x}_{i+1} \leftarrow \boxed{\boldsymbol{x}_i} + c\boxed{\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x})} + \sqrt{2c}\boxed{\boldsymbol{\epsilon}_i,} \quad i = 0, 1, \cdots, K,$$

Recall our denoising transition from the Hierarchical VAE formulation:

$$\boldsymbol{x}_{t-1} = \boxed{\boldsymbol{x}_t} + \alpha_t^2 * \boxed{\nabla_{\boldsymbol{x}_t} \log p(\boldsymbol{x}_t)} + \alpha_{t-1} * \boxed{\boldsymbol{\epsilon}}$$

*reparam. trick!*

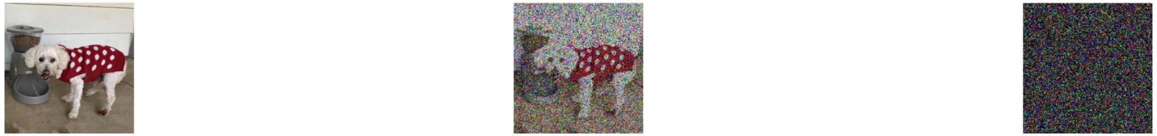
$\epsilon \sim \mathcal{N}(0, \mathrm{I})$

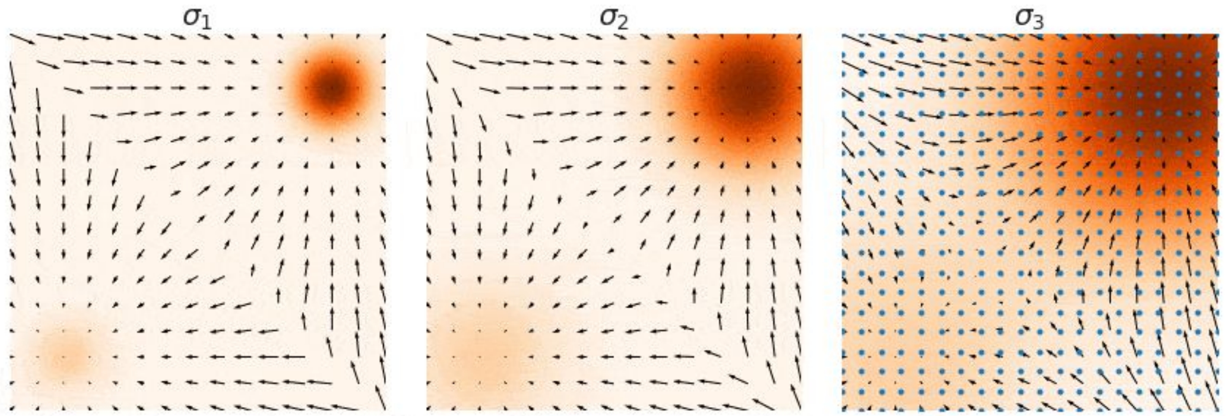
Annealed Langevin Dynamics sampling is **analogous** to Markov Chain procedure.

## Sampling

**HVAEs**

$$\begin{tikzcd}
x_0 & \cdots \arrow[l, bend right, "p(x_0|x_1)"'] & x_{t-1} & x_t \arrow[l, bend right, "p(x_{t-1}|x_t)"'] & x_{t+1} \arrow[l, bend right, "p(x_t|x_{t+1})"'] & \cdots & x_T \arrow[l, bend right, "p(x_{T-1}|x_T)"']
\end{tikzcd}$$





**NCSNs**

## Unifying Two Interpretations

We have shown two equivalently valid ways to describe a diffusion model!

- One takes the perspective of a Markovian Hierarchical VAE, where samples are steadily denoised through “hierarchies”

- Another takes the perspective of energy-based models and score matching where we iteratively refine an input through noise levels.

*they are two sides of the same coin!*

## Conditional Diffusion Models

So far we have been learning an unconditional diffusion model $p_\theta(\boldsymbol{x})$

## Conditional Diffusion Models

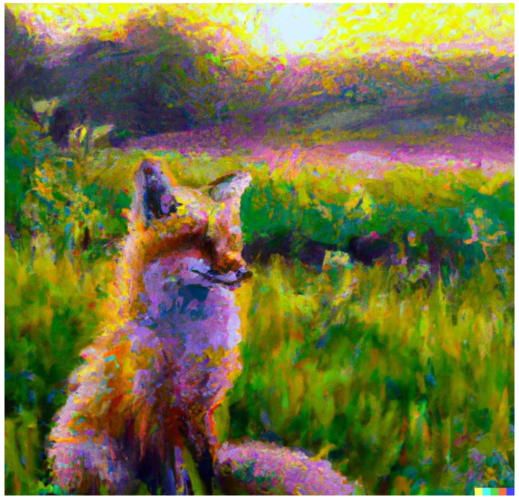
$p(image \mid text\_caption)$

How do we incorporate conditional information, to control data generation?

"a painting of a fox sitting in a field at sunrise in the style of Claude Monet"

Parti (but pretend it is ImageN)

StableDiffusion



Dall-E 2.0

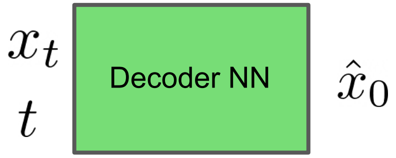
*source: ImageN, StableDiffusion, Dall-E 2.0*

## Conditional Diffusion Models

How do we incorporate conditional information, to control data generation?

Suppose we have conditioning information $y$ and now want to learn $p_\theta(\boldsymbol{x} \mid y)$

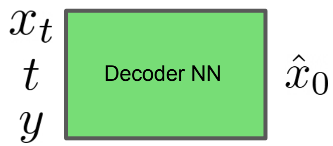
Well an unconditional diffusion model $p_\theta(\boldsymbol{x})$ really is just:



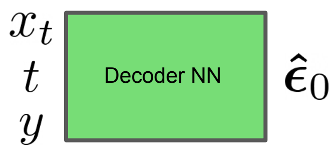
# Three Different Interpretations

It turns out, training a DiffModel can be implemented as a neural net that:

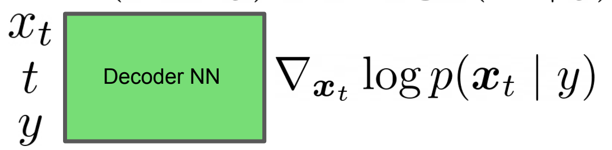
- 🖼️ Predicts original image $\hat{\boldsymbol{x}}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t) \approx \boldsymbol{x}_0 \longrightarrow \hat{\boldsymbol{x}}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t, y) \approx \boldsymbol{x}_0$



- 🔊 Predicts noise epsilon $\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t) \approx \boldsymbol{\epsilon}_0 \longrightarrow \hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t, y) \approx \boldsymbol{\epsilon}_0$



- 💯 Predicts score function $\boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t) \approx \nabla_{\boldsymbol{x}_t} \log p(\boldsymbol{x}_t) \longrightarrow \boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t, y) \approx \nabla \log p(\boldsymbol{x}_t \mid y)$



Caveat: A conditional diffusion model trained by this simple conditioning may ignore or downplay the given conditioning information.

## Guidance

Guidance provides more explicit control on the amount of weight the model gives to the conditioning information, at the cost of sample diversity.

- How much should our generated $x$ match $y$?

Let us take the score-based perspective of diffusion models.

Now, we are interested in learning $\nabla \log p(\boldsymbol{x}_t \mid y)$ rather than unconditional score function $\nabla \log p(\boldsymbol{x}_t)$

# Classifier Guidance

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Score of a conditional diffusion model:

$$\nabla \log p(\boldsymbol{x}_t \mid y) = \nabla \log \left( \frac{p(\boldsymbol{x}_t)p(y \mid \boldsymbol{x}_t)}{p(y)} \right)$$

## Classifier Guidance

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Score of a conditional diffusion model:

$$\begin{aligned}\nabla \log p(\boldsymbol{x}_t \mid y) &= \nabla \log \left( \frac{p(\boldsymbol{x}_t) p(y \mid \boldsymbol{x}_t)}{p(y)} \right) \\ &= \nabla \log p(\boldsymbol{x}_t) + \nabla \log p(y \mid \boldsymbol{x}_t) - \nabla \log p(y)\end{aligned}$$

# Classifier Guidance

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Score of a conditional diffusion model:

$$\begin{aligned}
\nabla \log p(\boldsymbol{x}_t \mid y) &= \nabla \log \left( \frac{p(\boldsymbol{x}_t) p(y \mid \boldsymbol{x}_t)}{p(y)} \right) \\
&= \nabla \log p(\boldsymbol{x}_t) + \nabla \log p(y \mid \boldsymbol{x}_t) - \nabla \log p(y) \\
&= \underbrace{\nabla \log p(\boldsymbol{x}_t)}_{\text{unconditional score}} + \underbrace{\nabla \log p(y \mid \boldsymbol{x}_t)}_{\text{adversarial gradient}}
\end{aligned}$$

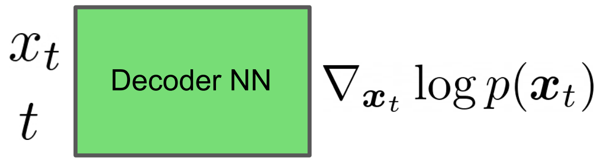
# Classifier Guidance

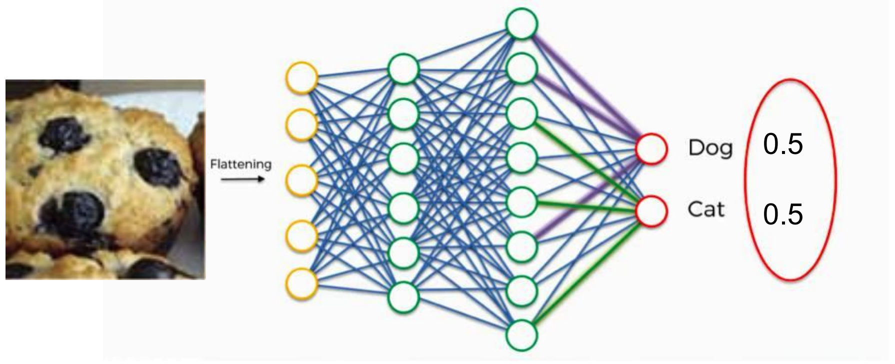
$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Score of a conditional diffusion model:

$$\begin{aligned}\nabla \log p(\boldsymbol{x}_t \mid y) &= \nabla \log \left( \frac{p(\boldsymbol{x}_t)p(y \mid \boldsymbol{x}_t)}{p(y)} \right) \\ &= \nabla \log p(\boldsymbol{x}_t) + \nabla \log p(y \mid \boldsymbol{x}_t) - \nabla \log p(y) \\ &= \underbrace{\nabla \log p(\boldsymbol{x}_t)}_{\text{unconditional score}} + \underbrace{\nabla \log p(y \mid \boldsymbol{x}_t)}_{\text{adversarial gradient}}\end{aligned}$$

It turns out that training a conditional diffusion model is as simple as training an unconditional diffusion model (as before) along with a classifier $p(y \mid \boldsymbol{x}_t)$!





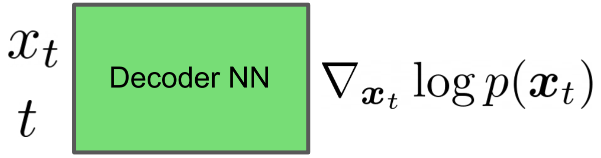
# Classifier Guidance

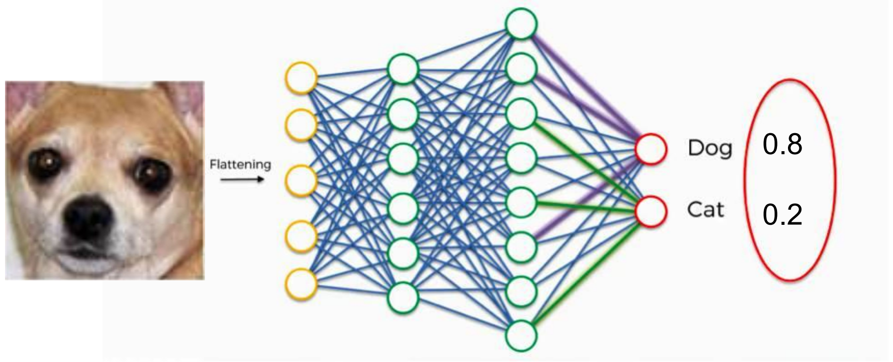
$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Score of a conditional diffusion model:

$$\begin{aligned}\nabla \log p(\boldsymbol{x}_t \mid y) &= \nabla \log \left( \frac{p(\boldsymbol{x}_t) p(y \mid \boldsymbol{x}_t)}{p(y)} \right) \\ &= \nabla \log p(\boldsymbol{x}_t) + \nabla \log p(y \mid \boldsymbol{x}_t) - \nabla \log p(y) \\ &= \underbrace{\nabla \log p(\boldsymbol{x}_t)}_{\text{unconditional score}} + \underbrace{\nabla \log p(y \mid \boldsymbol{x}_t)}_{\text{adversarial gradient}}\end{aligned}$$

It turns out that training a conditional diffusion model is as simple as training an unconditional diffusion model (as before) along with a classifier $p(y \mid \boldsymbol{x}_t)$!





## Classifier Guidance

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Score of a conditional diffusion model:

$$\begin{aligned}\nabla \log p(\boldsymbol{x}_t \mid y) &= \nabla \log \left( \frac{p(\boldsymbol{x}_t)p(y \mid \boldsymbol{x}_t)}{p(y)} \right) \\ &= \nabla \log p(\boldsymbol{x}_t) + \nabla \log p(y \mid \boldsymbol{x}_t) - \nabla \log p(y) \\ &= \underbrace{\nabla \log p(\boldsymbol{x}_t)}_{\text{unconditional score}} + \underbrace{\nabla \log p(y \mid \boldsymbol{x}_t)}_{\text{adversarial gradient}}\end{aligned}$$

It turns out that training a conditional diffusion model is as simple as training an unconditional diffusion model (as before) along with a classifier $p(y \mid \boldsymbol{x}_t)$!

Sampling is then done by querying the learned unconditional score function as well as the adversarial gradient of a classifier.

$\nabla_{\boldsymbol{x}_t} \log p(\boldsymbol{x}_t)$ + ( - )

# Classifier Guidance

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Score of a conditional diffusion model:

$$\begin{aligned}\nabla \log p(\boldsymbol{x}_t \mid y) &= \nabla \log \left( \frac{p(\boldsymbol{x}_t) p(y \mid \boldsymbol{x}_t)}{p(y)} \right) \\ &= \nabla \log p(\boldsymbol{x}_t) + \nabla \log p(y \mid \boldsymbol{x}_t) - \nabla \log p(y) \\ &= \underbrace{\nabla \log p(\boldsymbol{x}_t)}_{\text{unconditional score}} + \underbrace{\nabla \log p(y \mid \boldsymbol{x}_t)}_{\text{adversarial gradient}}\end{aligned}$$

It turns out that training a conditional diffusion model is as simple as training an unconditional diffusion model (as before) along with a classifier $p(y \mid \boldsymbol{x}_t)$!

Sampling is then done by querying the learned unconditional score function as well as the adversarial gradient of a classifier.

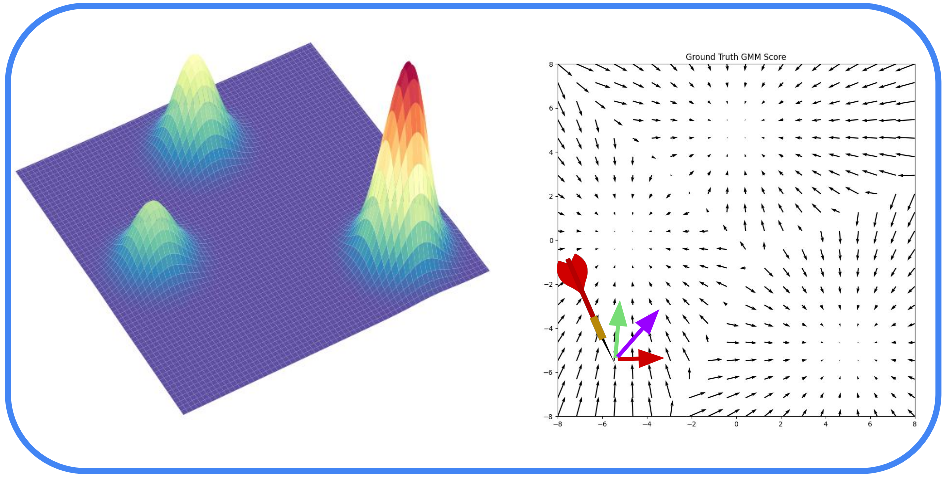
- The classifier attempts to tell us how good the noisy image matches the conditional label. It must be trained for arbitrary noise levels, however!

$$\nabla \log p(\boldsymbol{x}_t \mid y) = \nabla \log p(\boldsymbol{x}_t) + \gamma \nabla \log p(y \mid \boldsymbol{x}_t)$$

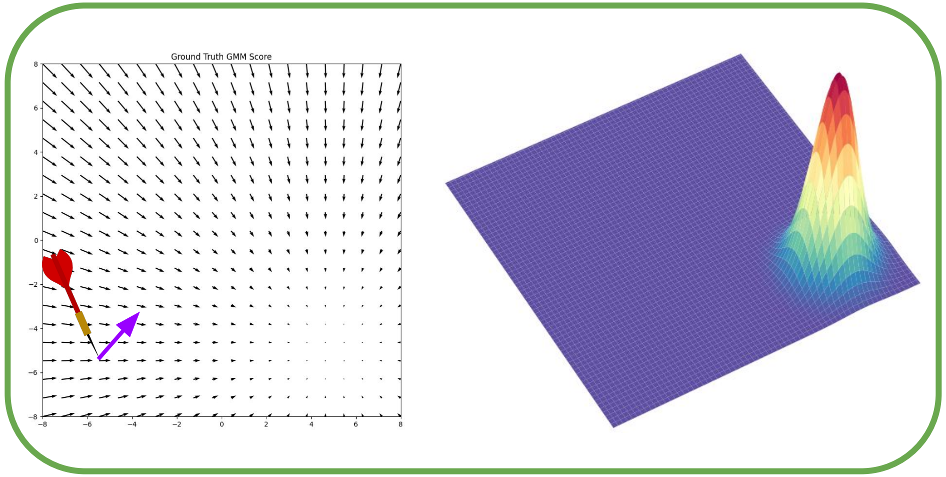
## Classifier Guidance

Sampling is then done by querying the learned unconditional score function as well as the adversarial gradient of a classifier.

$$\boxed{\nabla \log p(\boldsymbol{x}_t \mid y)} = \boxed{\nabla \log p(\boldsymbol{x}_t)} + \gamma \boxed{\nabla \log p(y \mid \boldsymbol{x}_t)}$$



$p(\boldsymbol{x})$

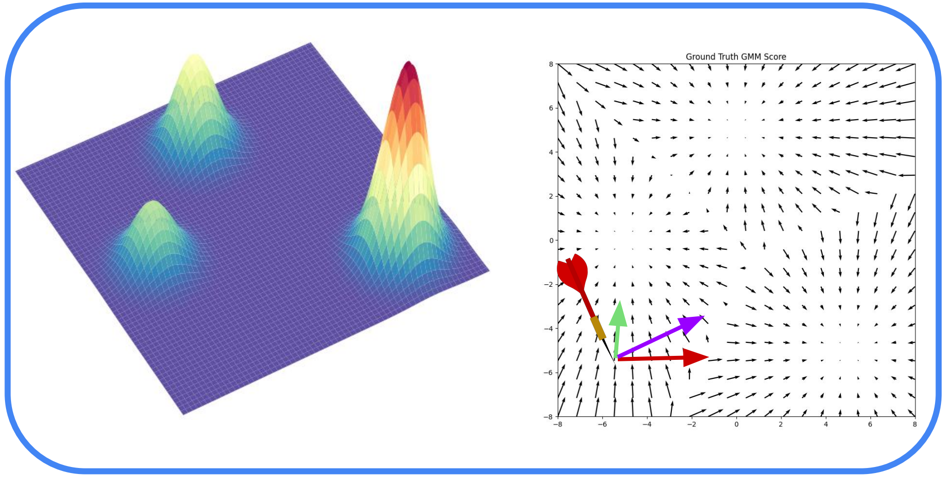


$p(\boldsymbol{x} \mid y = \quad)$

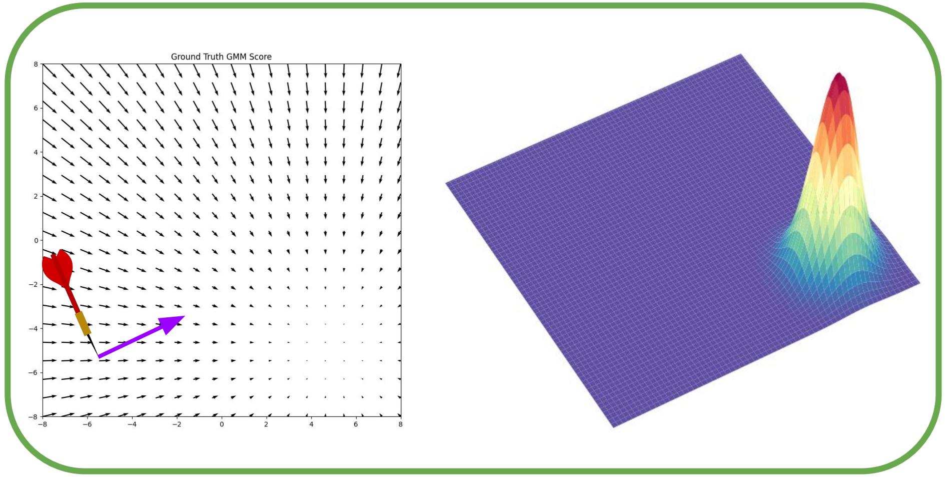
## Classifier Guidance

Sampling is then done by querying the learned unconditional score function as well as the adversarial gradient of a classifier.

$$\boxed{\nabla \log p(\boldsymbol{x}_t \mid y)} = \boxed{\nabla \log p(\boldsymbol{x}_t)} + \gamma \boxed{\nabla \log p(y \mid \boldsymbol{x}_t)}$$



$p(\boldsymbol{x})$

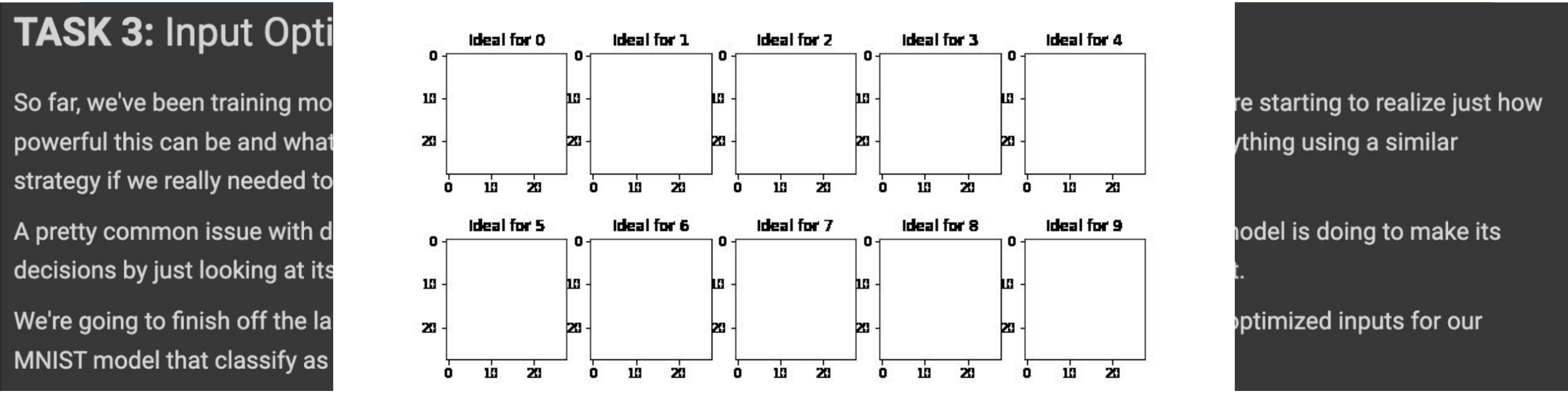


$p(\boldsymbol{x} \mid y = \quad )$

# Classifier Guidance

Sampling is then done by querying the learned unconditional score function as well as the adversarial gradient of a classifier.

Why not just use $\boxed{\nabla \log p(\boldsymbol{x}_t \mid y)} = \boxed{\nabla \log p(y \mid \boldsymbol{x}_t)}$ ?

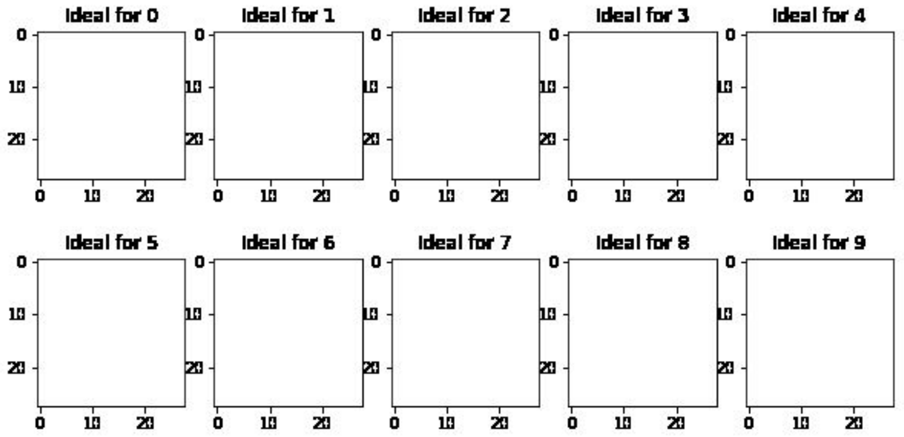


TASK 3: Input Opti

So far, we've been training mo re starting to realize just how
powerful this can be and what ything using a similar
strategy if we really needed to

A pretty common issue with d odel is doing to make its
decisions by just looking at its t.

We're going to finish off the la ptimized inputs for our
MNIST model that classify as



## Classifier-Free Guidance

Let's revisit the score of a conditional diffusion model:

$$\nabla \log p(\boldsymbol{x}_t \mid y) = \nabla \log p(y \mid \boldsymbol{x}_t) + \nabla \log p(\boldsymbol{x}_t)$$

## Classifier-Free Guidance

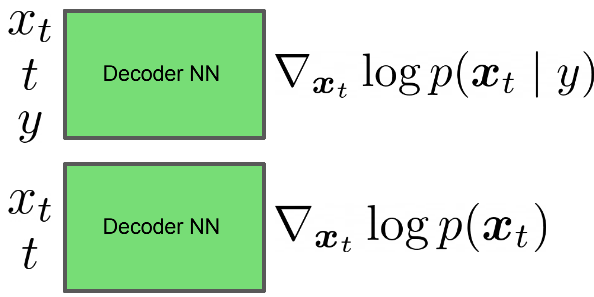
$$\nabla \log p(\boldsymbol{x}_t \mid y) = \underbrace{\gamma \nabla \log p(\boldsymbol{x}_t \mid y)}_{\text{conditional score}} + \underbrace{(1-\gamma)\nabla \log p(\boldsymbol{x}_t)}_{\text{unconditional score}}$$

Questions:

- What happens if $\gamma$ is 1?
- What happens if $\gamma$ is larger than 1?
- How many diffusion models do we need to train?

$$\boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t, y) \approx \nabla \log p(\boldsymbol{x}_t \mid y)$$

$$\boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t) \approx \nabla \log p(\boldsymbol{x}_t)$$


$x_t$
$t$
$y$
Decoder NN
$\nabla_{\boldsymbol{x}_t} \log p(\boldsymbol{x}_t \mid y)$
$x_t$
$t$
Decoder NN
$\nabla_{\boldsymbol{x}_t} \log p(\boldsymbol{x}_t)$

## Classifier-Free Guidance

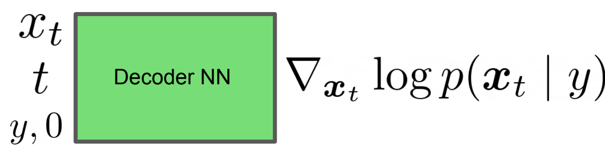
$$\nabla \log p(\boldsymbol{x}_t \mid y) = \underbrace{\gamma \nabla \log p(\boldsymbol{x}_t \mid y)}_{\text{conditional score}} + \underbrace{(1-\gamma)\nabla \log p(\boldsymbol{x}_t)}_{\text{unconditional score}}$$

Questions:

- What happens if $\gamma$ is 1?
- What happens if $\gamma$ is larger than 1?
- How many diffusion models do we need to train?

$$\boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t, y) \approx \nabla \log p(\boldsymbol{x}_t \mid y)$$

$$\boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t, 0) \approx \nabla \log p(\boldsymbol{x}_t)$$



Just one - as long as we train it with dropout on the conditioning info!

## Classifier-Free Guidance

source: 画像生成AI「Stable Diffusion」でどれぐらいプロンプト・呪文の指示に従うかを決める「CFG(classifier-free guidance)」とは一体何なのか？

## Classifier-Free Guidance

A bald eagle made of chocolate powder, mango, and whipped cream.

A photo of a Corgi dog riding a bike in Times Square. It is wearing sunglasses and a beach hat.

A bucket bag made of blue suede. The bag is decorated with intricate golden paisley patterns. The handle of the bag is made of rubies and pearls.

Three spheres made of glass falling into ocean. Water is splashing. Sun is setting.

A photo of a raccoon wearing an astronaut helmet, looking out of the window at night.

The Toronto skyline with Google brain logo written in fireworks.

*source: ImageN*

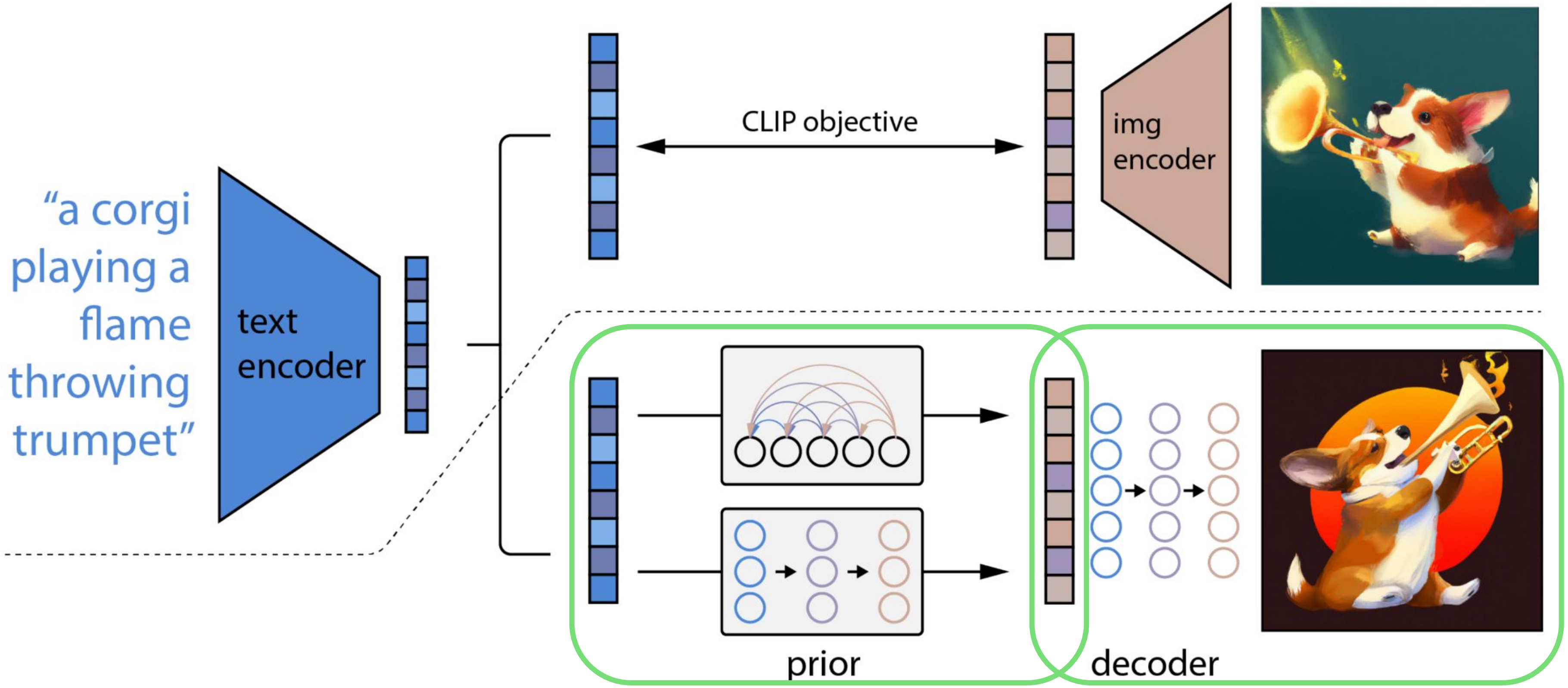
## Models in Practice

Let’s explore some of the state-of-the-art diffusion models in practice:

- DALL-E 2
- ImageN
- StableDiffusion

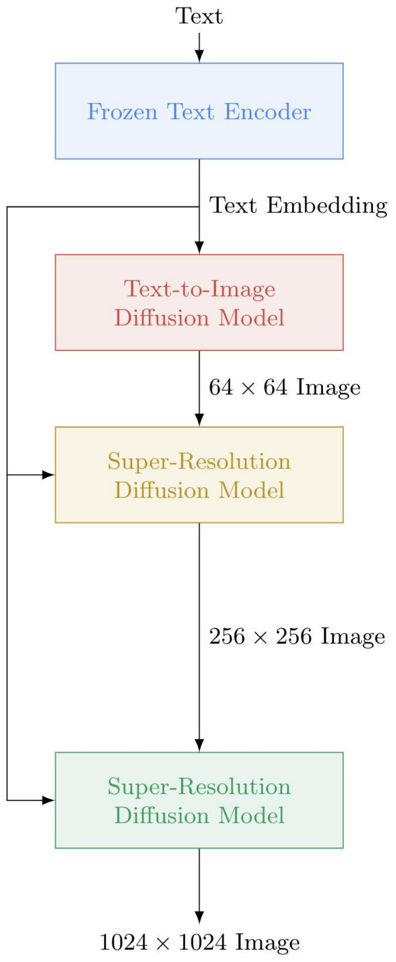
# DALL-E 2

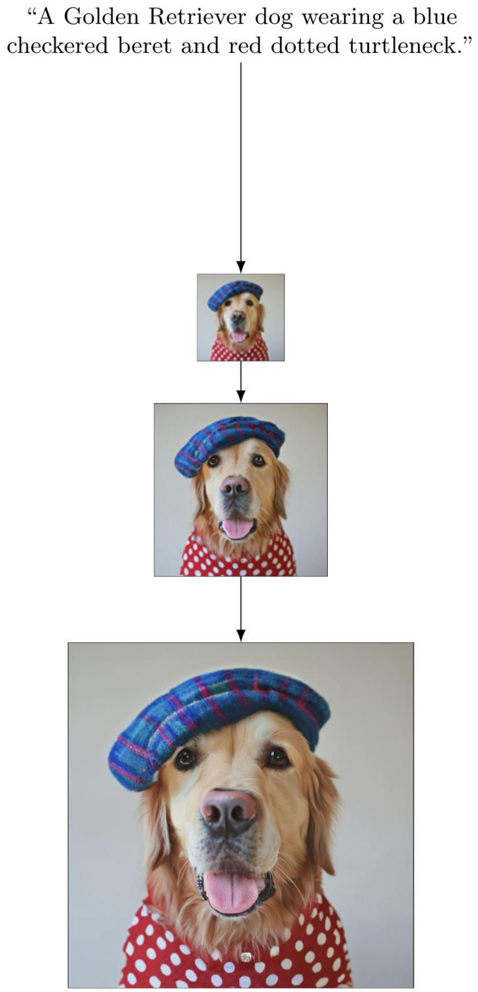
“where is the diffusion model?”



source: Hierarchical Text-Conditional Image Generation with CLIP Latents

# ImageN


Text
Frozen Text Encoder
Text Embedding
Text-to-Image Diffusion Model
64 × 64 Image
Super-Resolution Diffusion Model
256 × 256 Image
Super-Resolution Diffusion Model
1024 × 1024 Image


"A Golden Retriever dog wearing a blue checkered beret and red dotted turtleneck."

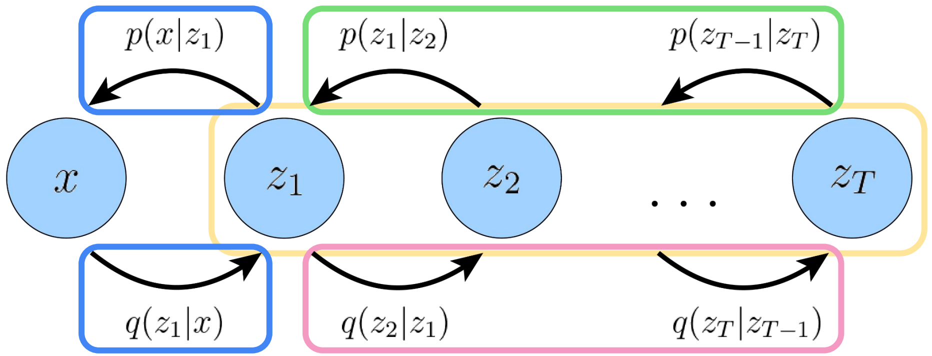
source: Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding

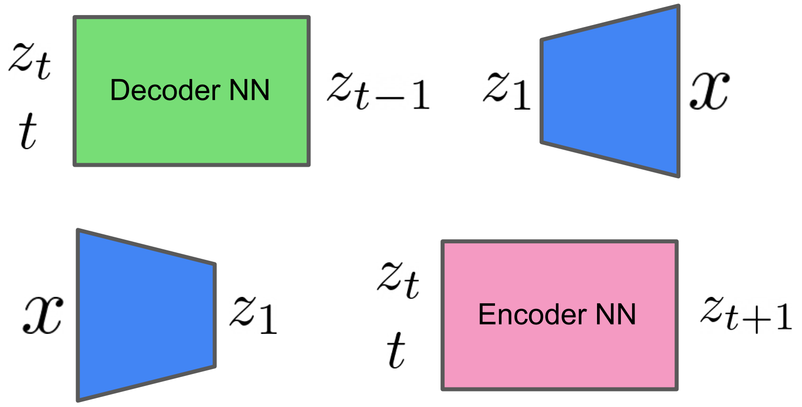
## StableDiffusion

Remember this?

- In a VAE we learn two networks: an encoder and a decoder.
- How many do we need to learn for a Hierarchical VAE?

*…what if we assume all latent dimensions are the same?*

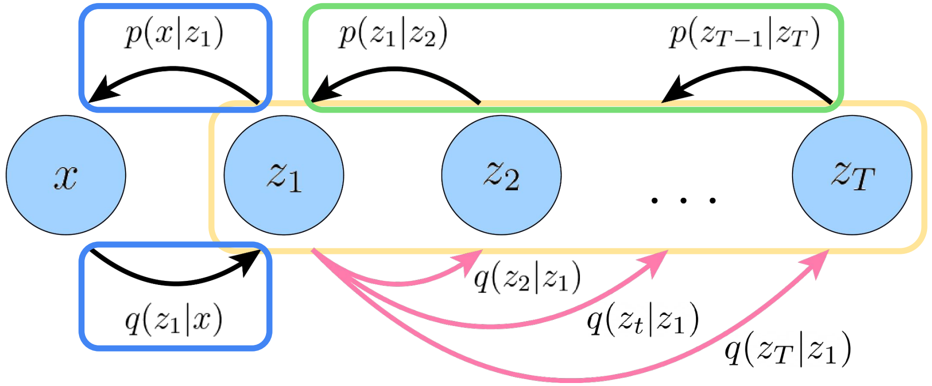


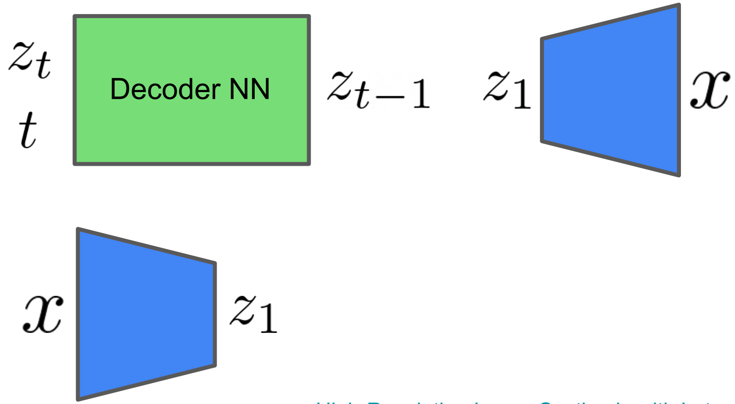


## StableDiffusion

It turns out that this is exactly what Stable/Latent Diffusion is doing!

- We model the latent distribution using a diffusion model



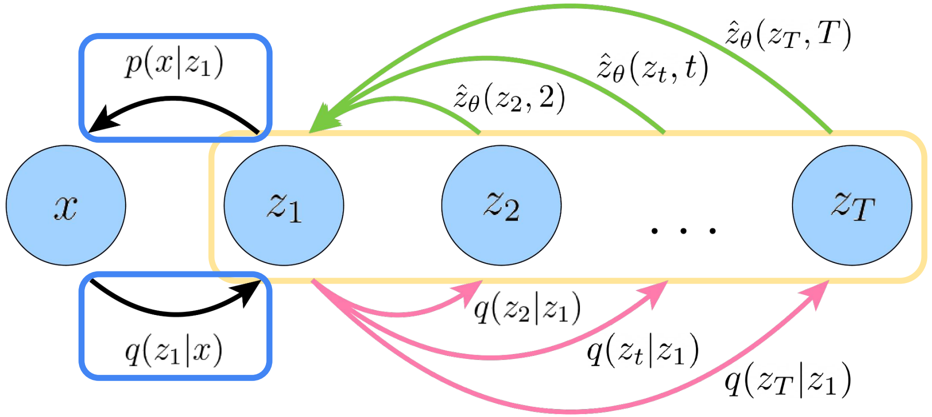


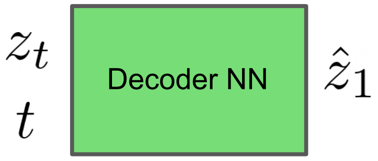
source: High-Resolution Image Synthesis with Latent Diffusion Models

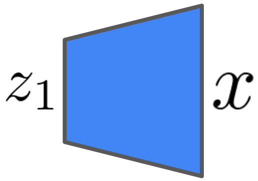
## StableDiffusion

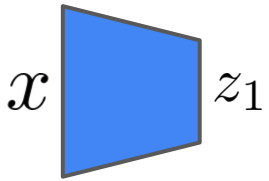
It turns out that this is exactly what Stable/Latent Diffusion is doing!

- Projects the image into a smaller latent space
- Learns the diffusion model for ***only the latent***
- Re-projects the denoised latents back to image space
- Benefits?







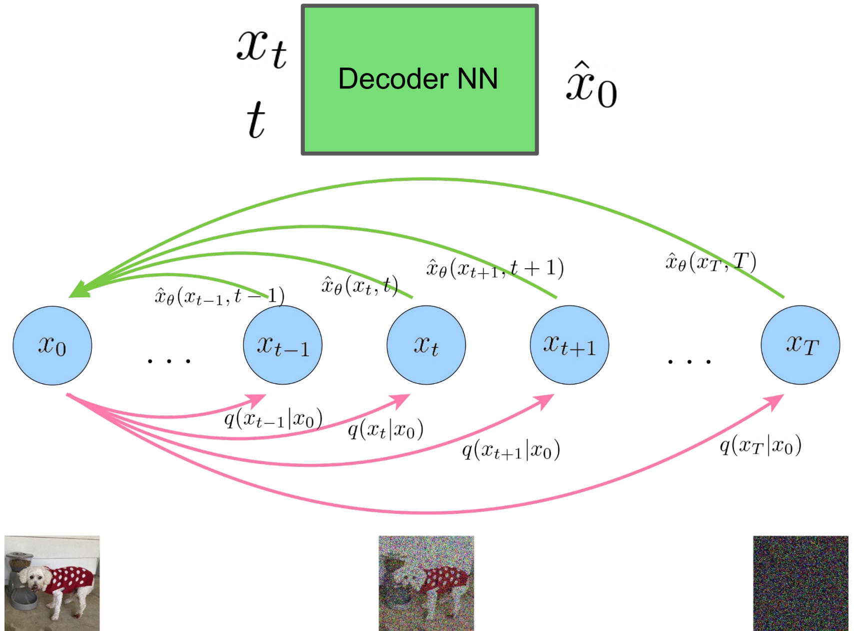


*play with a demo here!*

source: High-Resolution Image Synthesis with Latent Diffusion Models

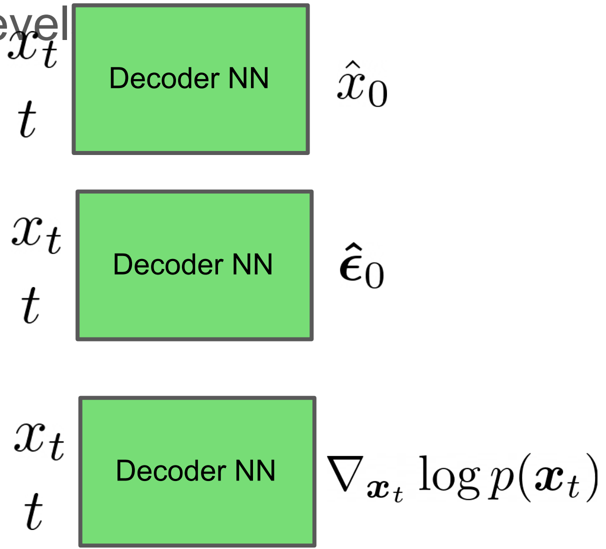
## Summarization

- We show that a Diffusion Model is simply a special case of a Hierarchical VAE



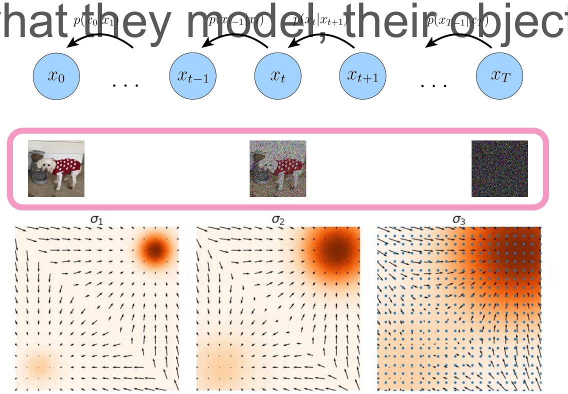
## Summarization

- We show that a Diffusion Model is simply a special case of a Hierarchical VAE
- We show that optimization boils down to learning a network to either predict the original image, the source noise, or the score function at arbitrary Gaussian noise corruption level


$x_t$
$t$
Decoder NN
$\hat{x}_0$
$x_t$
$t$
Decoder NN
$\hat{\boldsymbol{\epsilon}}_0$
$x_t$
$t$
Decoder NN
$\nabla_{\boldsymbol{x}_t} \log p(\boldsymbol{x}_t)$

## Summarization

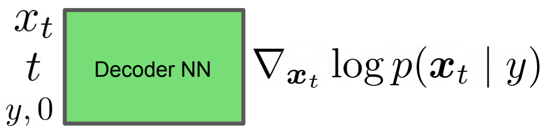
- We show that a Diffusion Model is simply a special case of a Hierarchical VAE
- We show that optimization boils down to learning a network to either predict the original image, the source noise, or the score function at arbitrary Gaussian noise corruption levels.
- We draw an explicit connection to score-based generative modeling and show they are equivalent in what they model, their objective, and sampling process.



## Summarization

- We show that a Diffusion Model is simply a special case of a Hierarchical VAE
- We show that optimization boils down to learning a network to either predict the original image, the source noise, or the score function at arbitrary Gaussian noise corruption levels.
- We draw an explicit connection to score-based generative modeling and show they are equivalent in what they model, their objective, and sampling process.
- We showcase how to build a conditional diffusion model, and apply guidance.

$p(image \mid text\_caption)$



source: ImageN, StableDiffusion, Dall-E 2.0

## Is it Good?

Diffusion models have amazing generative performance!

- Probably state of the art generative model right now
- Absolutely incredible at learning conditional distributions