Deep Learning - Foundations and Concepts

Chapter 20. Diffusion Models

nonlineark@github

April 22, 2025

Outline

Forward encoder

Suppose we take an image from the training set, which we will denote by x, and blend it with Gaussian noise independently for each pixel to give a noise-corrupted image z_1 defined by:

$$z_1 = \sqrt{1-\beta_1}x + \sqrt{\beta_1}\epsilon_1 \qquad \epsilon_1 \sim \mathcal{N}(\epsilon_1; 0, I)$$
$$q(z_1|x) = \mathcal{N}(z_1; \sqrt{1-\beta_1}x, \beta_1 I)$$

where $\beta_1 < 1$ is the variance of the noise distribution.

Forward encoder

We then repeat the process with additional independent Gaussian noise steps to give a sequence of increasingly noisy images $z_1, \ldots, z_T$:

$$z_t = \sqrt{1-\beta_t} z_{t-1} + \sqrt{\beta_t}\epsilon_t \qquad \epsilon_t \sim \mathcal{N}(\epsilon_t; 0, I)$$
$$q(z_t|z_{t-1}) = \mathcal{N}(z_t; \sqrt{1-\beta_t} z_{t-1}, \beta_t I)$$

The values of the variance parameters $\beta_t \in (0, 1)$ are set by hand and are typically chosen such that the variance values increase through the chain according to a prescribed schedule such that $\beta_1 < \cdots < \beta_T$.

Diffusion kernel

Using induction, it's straightforward to verify that:

$$z_t = \sqrt{\alpha_t}x + \sqrt{1-\alpha_t}\epsilon_t \qquad \epsilon_t \sim \mathcal{N}(\epsilon_t; 0, I)$$
$$q(z_t|x) = \mathcal{N}(z_t; \sqrt{\alpha_t}x, (1-\alpha_t)I)$$

where we have defined:

$$\alpha_t = \prod_{\tau=1}^{t}(1-\beta_\tau)$$

We call $q(z_t|x)$ the diffusion kernel. After many steps the image becomes indistinguishable from Gaussian noise, and in the limit $T \to \infty$ we have:

$$q(z_T|x) = \mathcal{N}(z_T; 0, I)$$

Conditional distribution

Our goal is to learn to undo the noise process, and so it is natural to consider the reverse of the conditional distribution $q(z_t|z_{t-1})$:

$$q(z_{t-1}|z_t) = \frac{q(z_t|z_{t-1})q(z_{t-1})}{q(z_t)}$$

But $q(z_{t-1})$ is difficult to calculate:

- Evaluation of the integral $\int q(z_{t-1}|x)p(x)\mathrm{d}x$ is intractable, because we must integrate over the unknown data density $p(x)$.
- If we approximate the integration using samples from the training data set, we obtain a complicated distribution expressed as a mixture of Gaussians.

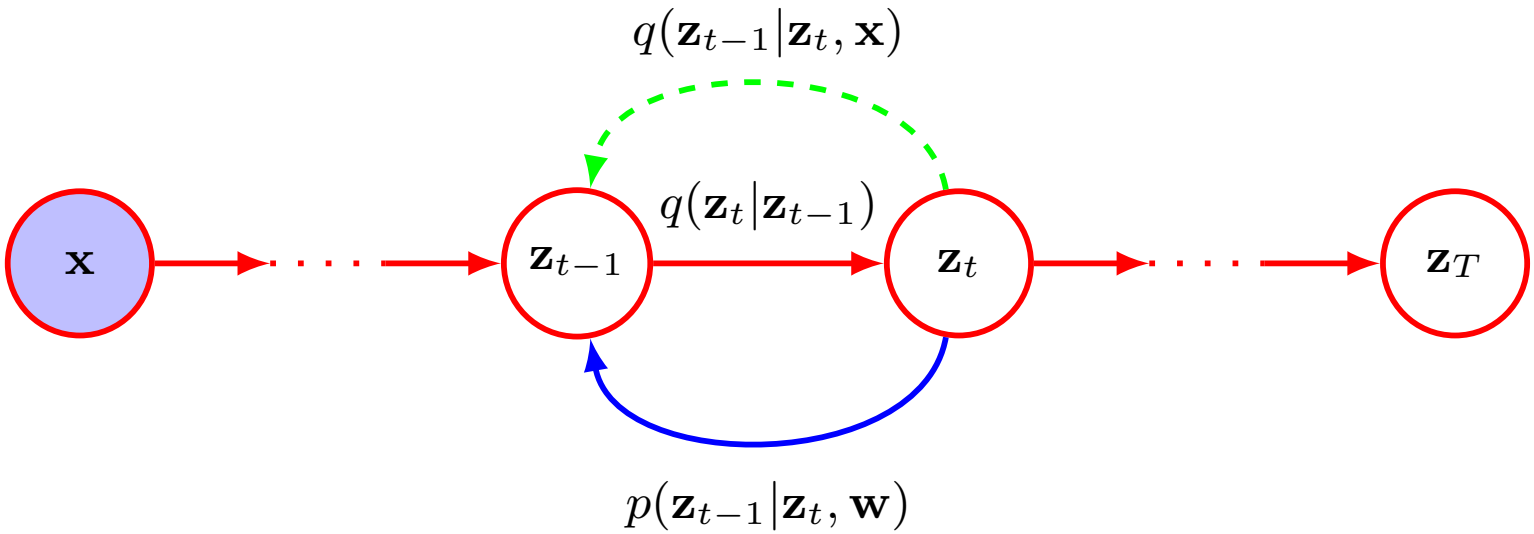
Conditional distribution

Instead, we consider the conditional version of the reverse distribution, conditioned on the data vector x, defined by $q(z_{t-1}|z_t, x)$:

$$
\begin{aligned}
q(z_{t-1}|z_t, x) &= \frac{q(z_t|z_{t-1}, x)q(z_{t-1}|x)}{q(z_t|x)} = \frac{q(z_t|z_{t-1})q(z_{t-1}|x)}{q(z_t|x)} \\
&= \mathcal{N}(z_{t-1}; m_t(x, z_t), \sigma_t^2 I) \\
m_t(x, z_t) &= \frac{(1-\alpha_{t-1})\sqrt{1-\beta_t}z_t + \sqrt{\alpha_{t-1}}\beta_t x}{1-\alpha_t} \\
\sigma_t^2 &= \frac{\beta_t(1-\alpha_{t-1})}{1-\alpha_t}
\end{aligned}
$$

Reverse decoder

- The forward encoder model is defined by a sequence of Gaussian conditional distribution $q(z_t|z_{t-1})$ but inverting this directly leads to a distribution $q(z_{t-1}|z_t)$ that is intractable.
- Instead, we will learn an approximation to the reverse distribution by using a distribution $p(z_{t-1}|z_t; w)$ governed by a deep neural network, where w represents the network weights and biases.



Reverse decoder

For $\beta_t \ll 1$, the distribution $q(z_{t-1}|z_t)$ will be approximately a Gaussian distribution over z_{t-1} (further references). We therefore model the reverse process using a Gaussian distribution of the form:

$$p(z_{t-1}|z_t; w) = \mathcal{N}(z_{t-1}; \mu(z_t, t; w), \beta_t I)$$

where $\mu(z_t, t; w)$ is a deep neural network governed by a set of parameters w. Note:

- The network takes the step index t explicitly as an input so that it can account for the variation of the variance β_t across different steps of the chain.
- It is also possible to learn the covariances of the denoising process by incorporating further outputs in the network to account for the curvature in the distribution $q(z_{t-1})$ in the neighborhood of z_t.

Reverse decoder

The overall reverse denoising process then takes the form of a Markov chain given by:

$$p(x, z_1, \ldots, z_T; w) = p(z_T) \prod_{t=2}^{T} p(z_{t-1}|z_t; w) p(x|z_1; w)$$

$$p(z_T) = q(z_T) = \mathcal{N}(z_T; 0, I)$$

Once the model has been trained, sampling is straightforward, we just follow the chain $z_T, \ldots, z_1, x$ in turn.

Training the decoder

The likelihood function is given by:

$$p(x; w) = \int p(x, z_1, \ldots, z_T; w) \mathrm{d}z_1 \cdots \mathrm{d}z_T$$

We see that the likelihood involves integrating over all possible trajectories by which noise samples could give rise to the observed data point. The integrals are intractable.

Evidence lower bound

- Since the exact likelihood is intractable, we can adopt a similar approach to that used with variational autoencoders and maximize the evidence lower bound.
- With diffusion models, we choose $q(z)$ to be given by the fixed distribution $q(z_1, \ldots, z_T|x)$, and so the only adjustable parameters are those in the model $p(x, z_1, \ldots, z_T; w)$ for the reverse Markov chain.

Evidence lower bound

$$
\begin{aligned}
\mathcal{L}(w) &= \int q(z) \log \frac{p(x, z; w)}{q(z)} \mathrm{d}z \\
&= E_q(\log \frac{p(z_T) \prod_{t=2}^{T} p(z_{t-1}|z_t; w) p(x|z_1; w)}{q(z_1|x) \prod_{t=2}^{T} q(z_t|z_{t-1}, x)}) \\
&= E_q(\log p(z_T) - \log q(z_1|x) + \log p(x|z_1; w) \\
&\quad + \sum_{t=2}^{T} \log \frac{p(z_{t-1}|z_t; w)}{q(z_t|z_{t-1}, x)})
\end{aligned}
$$

Evidence lower bound

- The first and second terms are independent of w and can be omitted.
- The third term can be evaluated by approximating the expectation by a Monte Carlo estimate: $E_q(\log p(x|z_1; w)) \approx \frac{1}{L} \sum_{l=1}^{L} \log p(x|z_1^{(l)}; w)$, where $z_1^{(l)} \sim \mathcal{N}(z_1; \sqrt{1-\beta_1}x, \beta_1 I)$.
- For the fourth term:
  - Although we can sample from $q(z_{t-1}|x)$ and $q(z_t|z_{t-1})$, the use of pairs of sampled values creates very noisy estimates with high variance, so that an unnecessarily large number of samples is required.
  - Instead, we rewrite the ELBO in a form that can be estimated by sampling just one value per term.

Rewriting the ELBO

Following our discussion of the ELBO for the variational autoencoder, our goal here is to write the ELBO in terms of Kullback-Leibler divergences:

$$q(z_t|z_{t-1}, x) = \frac{q(z_{t-1}|z_t, x)q(z_t|x)}{q(z_{t-1}|x)}$$

$$\log \frac{p(z_{t-1}|z_t; w)}{q(z_t|z_{t-1}, x)} = \log \frac{p(z_{t-1}|z_t; w)}{q(z_{t-1}|z_t, x)} + \log \frac{q(z_{t-1}|x)}{q(z_t|x)}$$

The second term is independent of w and can be omitted:

$$\mathcal{L}(w) = E_q(\log p(x|z_1; w)) + \sum_{t=2}^{T} E_q(\log \frac{p(z_{t-1}|z_t; w)}{q(z_{t-1}|z_t, x)})$$

Rewriting the ELBO

$$E_q(\log p(x|z_1;w)) = \int q(z_1|x)p(x|z_1;w)\mathrm{d}z_1 = \text{reconstruction term}$$

$$E_q(\log \frac{p(z_{t-1}|z_t;w)}{q(z_{t-1}|z_t,x)})$$

$$= \int \left(\int \left(\int q(z_1|x) \cdots q(z_{t-1}|z_{t-2})\mathrm{d}z_1 \cdots \mathrm{d}z_{t-2} \right) \right.$$

$$\left. \frac{q(z_{t-1}|z_t,x)}{q(z_{t-1}|x)} \log \frac{p(z_{t-1}|z_t;w)}{q(z_{t-1}|z_t,x)} \mathrm{d}z_{t-1} \right) q(z_t|x)\mathrm{d}z_t$$

$$= -\int \mathrm{KL}(q(z_{t-1}|z_t,x)||p(z_{t-1}|z_t;w))q(z_t|x)\mathrm{d}z_t$$

$$= \text{consistency term}$$

Rewriting the ELBO

The consistency terms are defined between pairs of Gaussian distributions and therefore can be expressed in closed form:

$$\mathrm{KL}(q(z_{t-1}|z_t, x)||p(z_{t-1}|z_t; w)) = \frac{1}{2\beta_t}||m_t(x, z_t) - \mu(z_t, t; w)||^2 + \text{const}$$

$$- \int \mathrm{KL}(q(z_{t-1}|z_t, x)||p(z_{t-1}|z_t; w))q(z_t|x)\mathrm{d}z_t$$

$$= -\frac{1}{2\beta_t}\int ||m_t(x, z_t) - \mu(z_t, t; w)||^2 q(z_t|x)\mathrm{d}z_t + \text{const}$$

$$\approx -\frac{1}{2\beta_t}\frac{1}{L}\sum_{l=1}^{L}||m_t(x, z_t^{(l)}) - \mu(z_t^{(l)}, t; w)||^2 + \text{const}$$

where $z_t^{(l)} \sim \mathcal{N}(z_t; \sqrt{\alpha_t}x, (1-\alpha_t)I)$, and any additive terms that are independent of the network parameters w have been absorbed into the constant term.

Predicting the noise

Let's further simplify the training objective $\mathcal{L}(w)$ by changing the role of the neural network so that instead of predicting the denoised image at each step of the Markov chain it predicts the total noise component that was added to the original image to create the noisy image at that step:

$$z_t = \sqrt{\alpha_t}x + \sqrt{1-\alpha_t}\epsilon_t \implies x = \frac{1}{\sqrt{\alpha_t}}z_t - \frac{\sqrt{1-\alpha_t}}{\sqrt{\alpha_t}}\epsilon_t$$

$$\begin{aligned} m_t(x, z_t) &= \frac{(1-\alpha_{t-1})\sqrt{1-\beta_t}z_t + \sqrt{\alpha_{t-1}}\beta_t x}{1-\alpha_t} \\ &= \frac{1}{\sqrt{1-\beta_t}}(z_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\epsilon_t) \end{aligned}$$

Predicting the noise

We introduce a neural network $g(z_t, t; w)$ that aims to predict the total noise that was added to x to generate z_t:

$$\mu(z_t, t; w) = \frac{1}{\sqrt{1-\beta_t}}(z_t - \frac{\beta_t}{\sqrt{1-\alpha_t}} g(z_t, t; w))$$

We see that:

$$\begin{aligned}\mathrm{KL}(q(z_{t-1}|z_t, x)||p(z_{t-1}|z_t; w)) &= \frac{1}{2\beta_t}||m_t(x, z_t) - \mu(z_t, t; w)||^2 + \text{const} \\ &= \frac{\beta_t}{2(1-\alpha_t)(1-\beta_t)}||g(z_t, t; w) - \epsilon_t||^2 + \text{const}\end{aligned}$$

Predicting the noise

Furthermore, for the reconstruction term, we have:

$$\begin{aligned}\log p(x|z_1; w) &= -\frac{1}{2\beta_1}||x - \mu(z_1, 1; w)||^2 + \text{const} \\ &= -\frac{1}{2(1-\beta_1)}||g(z_1, 1; w) - \epsilon_1||^2 + \text{const}\end{aligned}$$

Now the reconstruction and consistency terms can be combined:

$$\mathcal{L}(w) = -\sum_{t=1}^{T} \frac{\beta_t}{2(1-\alpha_t)(1-\beta_t)} \int ||g(z_t, t; w) - \epsilon_t||^2 q(z_t|x) \mathrm{d}z_t$$

Predicting the noise

It is found empirically that performance is further improved simply by omitting the factor $\frac{\beta_t}{2(1-\alpha_t)(1-\beta_t)}$, so that all steps in the Markov chain have equal weighting. If we only do one sample for each ϵ_t, then we have:

$$\mathcal{L}(w) = -\sum_{t=1}^{T} ||g(\sqrt{\alpha_t}x + \sqrt{1-\alpha_t}\epsilon_t, t; w) - \epsilon_t||^2$$

Predicting the noise

Algorithm 1: Training a denoising diffusion probabilistic model

for $t \leftarrow 1$ **to** T **do**

$\alpha_t \leftarrow \prod_{\tau=1}^{t}(1-\beta_\tau)$;

end

repeat

$x \sim \mathcal{D}$;

$t \sim \{1, \ldots, T\}$;

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

$z_t \leftarrow \sqrt{\alpha_t}x + \sqrt{1-\alpha_t}\epsilon$;

$\mathcal{L}(w) \leftarrow ||g(z_t, t; w) - \epsilon||^2$;

Take optimizer step

until *converged*;

return w;

Generating new samples

Algorithm 2: Sampling from a denoising diffusion probabilistic model

$z_T \sim \mathcal{N}(z_T; 0, I)$;

for $t \leftarrow T$ **to** 2 **do**

$\alpha_t \leftarrow \prod_{\tau=1}^{t}(1-\beta_\tau)$;

$\mu(z_t, t; w) \leftarrow \frac{1}{\sqrt{1-\beta_t}}(z_t - \frac{\beta_t}{\sqrt{1-\alpha_t}} g(z_t, t; w))$;

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

$z_{t-1} \leftarrow \mu(z_t, t; w) + \sqrt{\beta_t}\epsilon$;

end

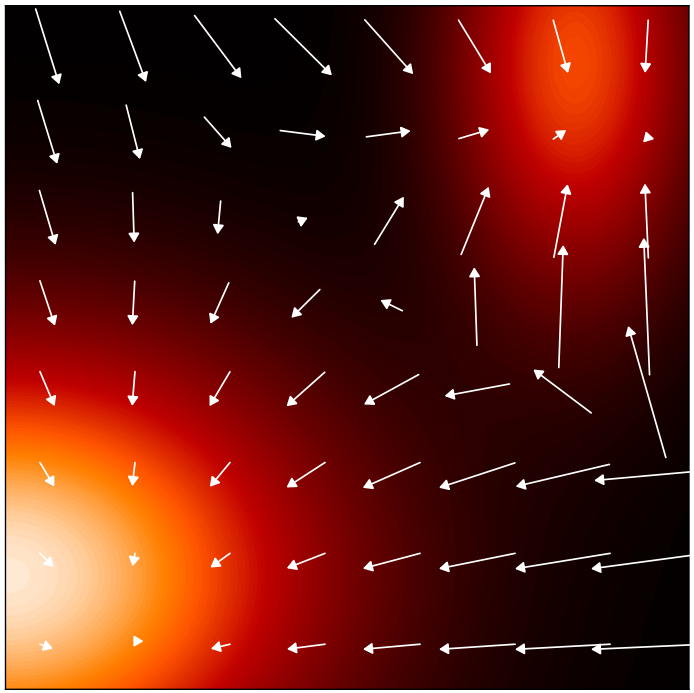
$x \leftarrow \frac{1}{\sqrt{1-\beta_1}}(z_1 - \frac{\beta_1}{\sqrt{1-\alpha_1}} g(z_1, 1; w))$;

return x;

Score function

Score function is defined as the gradient of the log likelihood with respect to the data vector x and is given by:

$$s(x) = \nabla_x \log p(x)$$



Score loss function

To train such a model we need to define a loss function that aims to match the model score function $s(x; w)$ to the score function $\nabla_x \log p(x)$ of the distribution $p(x)$ that generated the data. An example:

$$J(w) = \frac{1}{2} \int ||s(x; w) - \nabla_x \log p(x)||^2 p(x) \mathrm{d}x$$

Score loss function

There are broadly two ways to represent the score function $s(x; w)$ using a deep neural network:

- Because $\nabla_x \log p(x) = \frac{\nabla_x p(x)}{p(x)}$ has the same dimensionality as x, the first approach is to have a network with the same number of outputs as inputs.
  - The learned function may not be a gradient of a scalar function.
- An alternative approach is to have a network with a single output $\phi(x)$ and then to compute $\nabla_x \phi(x)$ using automatic differentiation.
  - This second approach is computationally more expensive.

Modified score loss

We do not know the true data distribution $p(x)$. All we have is the finite data set $\mathcal{D} = \{x_1, \ldots, x_N\}$ from which we can construct an empirical distribution:

$$p_{\mathcal{D}}(x) = \frac{1}{N} \sum_{n=1}^{N} \delta(x - x_n)$$

where δ is the Dirac delta function. Now $p_{\mathcal{D}}$ is not a differentiable function of x, we need to introduce a noise model to smear out the data points:

$$q_\sigma(z) = \int q_\sigma(z|x) p(x) \mathrm{d}x$$

where $q_\sigma(z|x)$ is the noise kernel, and $q_\sigma(z)$ is known as a Parzen estimator.

Modified score loss

We can now use the corresponding loss with respect to the smoothed Parzen density in the form:

$$
\begin{aligned}
J(w) &= \frac{1}{2}\int ||s(z;w) - \nabla_z \log q_\sigma(z)||^2 q_\sigma(z)\mathrm{d}z \\
&= \frac{1}{2}\int (||s(z;w)||^2 q_\sigma(z) - 2(s(z;w)\cdot\nabla_z q_\sigma(z)))\mathrm{d}z + C \\
&= \frac{1}{2}\int \Bigg(\int ||s(z;w)||^2 q_\sigma(z|x)p(x)\mathrm{d}x \\
&\qquad - \int 2(s(z;w)\cdot\nabla_z q_\sigma(z|x))p(x)\mathrm{d}x \Bigg)\mathrm{d}z + C \\
&= \frac{1}{2}\iint (||s(z;w)||^2 - 2(s(z;w)\cdot\nabla_z \log q_\sigma(z|x)))q_\sigma(z|x)p(x)\mathrm{d}x\mathrm{d}z + C \\
&= \frac{1}{2}\iint ||s(z;w) - \nabla_z \log q_\sigma(z|x)||^2 q_\sigma(z|x)p(x)\mathrm{d}x\mathrm{d}z + C
\end{aligned}
$$

Modified score loss

If we substitute for $p(x)$ using the empirical density $p_{\mathcal{D}}(x)$, we obtain:

$$J(w) = \frac{1}{2N} \sum_{n=1}^{N} \int ||s(z; w) - \nabla_z \log q_\sigma(z|x_n)||^2 q_\sigma(z|x_n) \mathrm{d}z + C$$

Modified score loss

For the Gaussian Parzen kernel $q_\sigma(z|x) = \mathcal{N}(z; x, \sigma^2 I)$, the score function becomes:

$$\nabla_z \log q_\sigma(z|x) = -\frac{z-x}{\sigma^2} = -\frac{1}{\sigma}\epsilon \qquad \epsilon \sim \mathcal{N}(\epsilon; 0, I)$$

If we consider the specific noise model $q(z|x) = \mathcal{N}(z; \sqrt{\alpha_t}x, (1-\alpha_t)I)$ then we obtain:

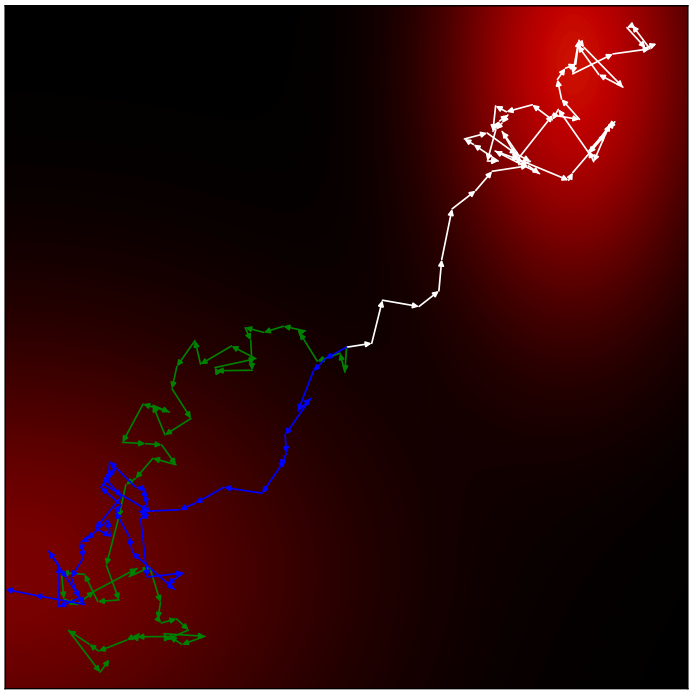
$$\nabla_z \log q(z|x) = -\frac{1}{\sqrt{1-\alpha_t}}\epsilon \qquad \epsilon \sim \mathcal{N}(\epsilon; 0, I)$$

We see that the score loss measures the difference between the neural network prediction and the noise ϵ, and denoising score matching has close connection to denoising diffusion models.

Modified score loss

Having trained a score-based model, we can use Langevin dynamics to draw new samples:

$$z^{(\tau+1)} = z^{(\tau)} + \eta s(z^{(\tau)}; w) + \sqrt{2\eta}\epsilon^{(\tau)} \qquad \epsilon^{(\tau)} \sim \mathcal{N}(\epsilon; 0, I)$$



Noise variance

Potential problems with the current score matching model:

- If the data distribution lies on a manifold of lower dimensionality than the data space, the probability density will be zero at points off the manifold and here the score function is undefined.
- In regions of low data density, the estimate of the score function may be inaccurate, which can lead to poor trajectories when using Langevin sampling.
- The Langevin procedure may not sample correctly if the data distribution comprises a mixture of disjoint distributions.

Noise variance

- All three problems can be addressed by choosing a sufficiently large value for the noise variance σ^2 used in the kernel function.
- However, too large a variance will introduce a significant distortion of the original distribution and this itself introduces inaccuracies in the modelling of the score function.

Noise variance

We can choose a sequence of variance values $\sigma_1^2 < \cdots < \sigma_L^2$, and train a score network that takes the variance as an additional input. For a data vector x_n, the loss function then takes the form:

$$J_n(w) = \frac{1}{2} \sum_{l=1}^{L} \lambda_l \int ||s(z, \sigma_l; w) - \nabla_z \log q_{\sigma_l}(z|x_n)||^2 q_{\sigma_l}(z|x_n) \mathrm{d}z$$

where λ_l are weighting coefficients. Once trained, samples can be generated by running a few steps of Langevin sampling from each of the models for $l = L, \ldots, 1$ in turn.

Further references.

Stochastic differential equations

It is helpful to use a large number of steps, often several thousand, when constructing the noise process for a diffusion model. Taking the limit of an infinite number of steps leads to a formulation of diffusion models for continuous time as stochastic differential equations (SDEs).

Stochastic differential equations

The goal of the forward encoder is to construct a diffusion process indexed by a continuous time variable $t \in [0, T]$, such that $z(0)$ is distributed as the data and $z(T)$ is distributed as the prior. This diffusion process can be modelled as the solution to an SDE:

$$\mathrm{d}z = f(z, t)\mathrm{d}t + g(t)\mathrm{d}w$$

where:

- $f(\cdot, t) : \mathbb{R}^D \to \mathbb{R}^D$ is called the drift coefficient of $z(t)$.
- $g(\cdot) : \mathbb{R} \to \mathbb{R}$ is known as the diffusion coefficient of $z(t)$.
- w is the standard Wiener process.

Stochastic differential equations

The reverse decoder is the reverse of a diffusion process, which is also a diffusion process, running backwards in time and given by the reverse-time SDE:

$$\mathrm{d}z = (f(z,t) - g^2(t)\nabla_z \log p(z))\mathrm{d}t + g(t)\mathrm{d}\bar{w}$$

where:

- We recognize $\nabla_z \log p(z)$ as the score function.
- $\bar{w}$ is a standard Wiener process when time flows backwards from T to 0.

Further references.

Guided diffusion

In many applications, we want to sample from a conditional distribution $p(x|c)$:

- The simplest approach is to treat c as an additional input into the denoising neural network $g(z, c, t; w)$ and then to train the network using matched pairs $\{x_n, c_n\}$.
  - The main limitation of this approach is that the network can give insufficient weight to, or even ignore, the conditioning variables.
- We need a way to control how much weight is given to the conditioning information, this additional pressure to match the conditioning information is called guidance.
- There are two main approaches to guidance depending on whether or not a separate classifier model is used.

Classifier guidance

Suppose that a trained classifier $p(c|x)$ is available:

$$\nabla_x \log p(x|c) = \nabla_x \log \frac{p(x)p(c|x)}{p(c)} = \nabla_x \log p(x) + \nabla_x \log p(c|x)$$

The influence of the classifier can be controlled by introducing a hyperparameter λ, called the guidance scale:

$$\text{score}(x, c) = \nabla_x \log p(x) + \lambda \nabla_x \log p(c|x)$$

Classifier guidance

- One problem with the classifier-based approach to guidance is that a separate classifier must be trained.
- Furthermore, this classifier needs to be able to classify examples with varying degrees of noise.

Classifier-free guidance

We can train a single network to model both $p(x|c)$ and $p(x)$, in which the conditioning variable c is set to a null value, for example $c = 0$, with some probability during training, typically around $10 - 20\%$. Then $p(x)$ is represented by $p(x|c = 0)$. We can rewrite the score function this way:

$$\begin{aligned}\text{score}(x, c) &= \nabla_x \log p(x) + \lambda \nabla_x \log p(c|x) \\ &= \nabla_x \log p(x) + \lambda(\nabla_x \log p(x|c) - \nabla_x \log p(x)) \\ &= \lambda \nabla_x \log p(x|c) + (1 - \lambda)\nabla_x \log p(x)\end{aligned}$$

Classifier-free guidance

Figure: Images generated using the conditioning text "A stained glass window of a panda eating bamboo". Examples on the left were generated with $\lambda = 0$, whereas examples on the right were generated with $\lambda = 3$

Classifier-free guidance

Figure: Image super-resolution can be achieved by denoising a high-resolution sample from a Gaussian using the low-resolution image as a conditioning variable

Classifier-free guidance

More ways of generating images:

- Image denoising is performed at a lower resolution and the result is subsequently up-sampled using a separate network to give a final high-resolution output.
- Latent diffusion models:
  - An autoencoder is trained on noise-free images to obtain a lower-dimensional representation of the images.
  - A U-net architecture is then trained to perform the denoising within the lower-dimensional space.
  - The denoised representation is mapped into the high-resolution image space using the output half of the autoencoder network.