Diffusion Model
Generative models can be used in computational biology to model and simulate biological processes, such as gene expression, protein folding, and signal transduction. Here are some examples of how generative models can be used in computational biology:

1. Gene expression: Generative models can be used to model the expression of genes in response to different stimuli or conditions. These models can be used to predict the expression levels of genes in different conditions or to identify new genes that are involved in a biological process.

2. Protein folding: Generative models can be used to simulate the folding of proteins, which is a critical process that determines their structure and function. These models can be used to predict the structures of new proteins, to identify structural features that are important for protein function, and to design new proteins with specific functions.

3. Signal transduction: Generative models can be used to model the interactions between proteins and other molecules that are involved in signal transduction pathways. These models can be used to predict the effects of perturbations to the system, to identify new targets for drug development, and to design new molecules that can modulate the activity of the pathway.

Overall, generative models can be used to gain insights into complex biological systems and to design new experiments or therapies that target these systems.
Generative model

- More formally, generative models capture the distribution of $P(x)$;
- Why? Because if we know $P(x)$, then we can sample from it and get new data points
- **Problem**: $P(x)$ is intractable
  - e.g., The distribution of all possible images.
Generative model

- **Idea**: use a latent variable $Z$ that can be easily sampled;
- Instead of modeling $P(x)$, we try to find $P(x|z)$
  - Assume $Z \sim N(0, 1)$
  - Then, $P(x) = \int P(x|z) P(z)$
- **How to get $P(x|z)$?** Machine learning!
  - Essentially, we are fitting some function $f(z)$
Diffusion model

- Originally proposed by “Deep Unsupervised Learning using Nonequilibrium Thermodynamics”, Sohl-Dickstein et al., 2015
- **Idea:** Gradually add Gaussian noise to $x_t$, as $t \to \infty$, $x_t \sim N(0, 1)$. Then, learn some distributions $q(x_{t-1} | x_t)$ to reverse the process.
  - After training, we can sample a $x_t$ and get its reconstruction with $q(x_{t-1} | x_t)$. 
\[ p(x_T) = \mathcal{N}(0, I) \]

\[ q(x_T | x_0) \approx \mathcal{N}(0, I) \]

\[ q(x_T | x_{T-1}) \]

\[ p(x_{T-1} | x_T) \]

\[ p(x_1 | x_2) \]

\[ p(x_0 | x_1) \]

\[ x_T \rightarrow x_1 \rightarrow x_0 \]

\[ x_T \leftarrow x_1 \leftarrow x_0 \]

\[ q(x_1 | x_0) \]

\[ q(x_0 | x_1) \]

\[ q(x_T | x_{T-1}) \]

https://www.zhihu.com/question/536012286/answer/2533146567
Deep Unsupervised Learning using Nonequilibrium Thermodynamics”, Sohl-Dickstein et al., 2015
Forward process

- Add small Gaussian noise to the sample;
- \( \bar{\sigma}_t \) are hyperparameters controlling the step sizes.

\[
q(x_t | x_{t-1}) = \mathcal{N}(x_t; \sqrt{1 - \beta_t} x_{t-1}, \beta_t I)
\]

\[
q(x_{1:T} | x_0) = \prod_{t=1}^{T} q(x_t | x_{t-1})
\]
Reverse process

- If $\theta_t$ is small enough, $q(x_{t-1} | x_t)$ will also be Gaussian;
- To define a Gaussian distribution, we need its mean and variance;
  - Unfortunately, there is no simple estimation of the two values;
  - Instead, we learn a model, e.g., a neural net, to predict the values:

$$p_{\theta}(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_{\theta}(x_{t-1} | x_t) \quad p_{\theta}(x_{t-1} | x_t) = \mathcal{N}(x_{t-1} ; \mu_{\theta}(x_t, t), \Sigma_{\theta}(x_t, t))$$

From the model
Loss function

- But what should be the training targets for $\mu_\theta(x_t, t)$ and $\Sigma_\theta(x_t, t)$?
- **Idea:** maximize the likelihood of the data
  - $\log p(x) = \log E_{q(z)} [p_\theta(x, z)/q(z)] \geq E_{q(z)} \log [p_\theta(x, z)/q(z)] = \text{ELBO}$
- For diffusion:
$\mathbb{E}_q \left[ D_{KL}(q(x_T|x_0) \parallel p(x_T)) + \sum_{t>1} D_{KL}(q(x_{t-1}|x_t,x_0) \parallel p_\theta(x_{t-1}|x_t)) - \log p_\theta(x_0|x_1) \right]$
Algorithm 1 Training

1: repeat
2: \( \mathbf{x}_0 \sim q(\mathbf{x}_0) \)
3: \( t \sim \text{Uniform}([1, \ldots, T]) \)
4: \( \epsilon \sim \mathcal{N}(0, I) \)
5: Take gradient descent step on
   \[ \nabla_{\theta} \| \epsilon - \epsilon_{\theta}(\sqrt{\alpha_t} \mathbf{x}_0 + \sqrt{1 - \alpha_t} \epsilon, t) \|^2 \]
6: until converged

Algorithm 2 Sampling

1: \( \mathbf{x}_T \sim \mathcal{N}(0, I) \)
2: for \( t = T, \ldots, 1 \) do
3: \( \mathbf{z} \sim \mathcal{N}(0, I) \) if \( t > 1 \), else \( \mathbf{z} = 0 \)
4: \( \mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left( \mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{1 - \alpha_t}} \epsilon_{\theta}(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z} \)
5: end for
6: return \( \mathbf{x}_0 \)

\[ \mathbf{x}_{t-1} \sim P_{\theta}(\mathbf{x}_{t-1} | \mathbf{x}_t) \]
Conditioned Generation

- Now we can sample a $z$ from $N(0, 1)$ and generate a data point (an image) with the learned model;
- But how to control the generation?
  - e.g., we want an image of a cat
- That is, we want $P(x|y)$
Summary

- Think the forward process as adding Gaussian noise and we are training a model to denoise through the reverse process.
References

“Deep Unsupervised Learning using Nonequilibrium Thermodynamics”, Sohl-Dickstein et al., 2015
