Generative Modelling 11

11.1 Variational Autoencoders

11.1.1 Introduction

Variational Autoencoders (VAE) [9, 10] are one of the most commonly used and efficient approaches for the task of generative modelling. Unlike discriminative models, whose objective is to learn the conditional distribution $p(y|x)$, generative techniques aim to obtain the whole distribution of the data $\underline{x} = x_1, ..., x_n$, i.e. estimate $p(x)$ with the parametric distribution $p(x, \theta)$, typically differentiable with respect to θ.

In this way it becomes possible to sample from such a probability distribution, thus generating new instances similar to the training dataset, or to assign to a given instance the probability of having been generated by the same process as the training dataset.

In general, generative modelling deals with a harder task with respect to discriminative models because in the former there is much more knowledge to learn. Intuitively, this is valid also for humans: by seeing many labelled images of dogs and cats it is much easier to distinguish dogs from cats than to draw a new image of a dog or a cat. In fact, in order to assign a label it is only required to find out some patterns which are different between the categories, while to generate a new instance it is necessary to capture correlations in the dataset, as for example the fact that dogs have two eyes, a tail, etc.. So, the discriminative model has to learn a decision boundary in the data space, while the generative one has to understand how data is placed throughout the space.

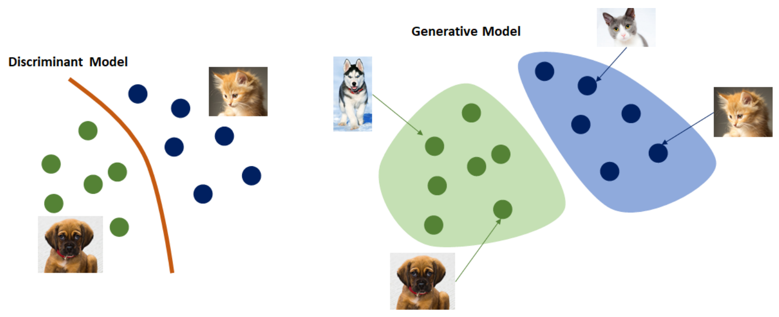


Figure 11.1: Image from *https://medium.com/@jordi299/about-generative-and-discriminative-models-d8958b67ad32*

It may be more convenient to avoid learning directly the distribution of the data (pixels in our example), but to introduce latent features z which can describe the data and rewrite our distribution as:

$$p(x, z|\theta) = p(x|z, \theta)p(z)$$

We assume that the posterior distribution $p(z|x)$ is intractable and that we are working in the big data regime, so that we are forced to consider mini-batches during the training phase. The goals that we would like to achieve are:

- Learning the parameters θ
- Approximating the posterior $p(\boldsymbol{z}|\boldsymbol{x})$ in order to understand which are the latent features extracted from a given instance $\boldsymbol{x}$
- Performing approximate inference on $\boldsymbol{x}$ to be able to fill holes in an instance

In this scenario, where we have a parametric distribution with latent variables, it might occur to us to use Expectation Maximization. But we must remember that the E-step requires being able to evaluate the conditional distribution $p(\boldsymbol{z}|\boldsymbol{x}, \theta_{OLD})$ which we have assumed here to be intractable. Other techniques that we have already seen, such as mean field variational approximation or sampling-based methods, turn out to be too computationally expensive as they do not scale well with large datasets. We will now see how to solve the problem by applying stochastic variational inference together with an encoder-decoder approach.

11.1.2 Autoencoding Variational Bayes (AEVB [11])

Following the idea of black-box variational inference, we start by considering a parametric variational distribution for the latent variables $q(\boldsymbol{z}|\phi)$. We would like to optimize the ELBO jointly on θ and ϕ by stochastic gradient ascent and so we need to compute $\nabla_{\theta,\phi}\mathcal{L}(\phi, \theta)$. We fix the prior distribution of $\boldsymbol{z}$ as a standard Gaussian:

$$p(\boldsymbol{z}) \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$$

and we start by rewriting the lower bound as:

$$\begin{aligned}
\mathcal{L}(\phi, \theta) &= \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x},\phi)}[\log p(\boldsymbol{x}, \boldsymbol{z}|\theta) - \log q(\boldsymbol{z}|\boldsymbol{x}, \phi)] \\
&= \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x},\phi)}[\log p(\boldsymbol{x}|\boldsymbol{z}, \theta) + \log p(\boldsymbol{z}) - \log q(\boldsymbol{z}|\boldsymbol{x}, \phi)] \\
&= \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x},\phi)}[\log p(\boldsymbol{x}|\boldsymbol{z}, \theta)] - \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x},\phi)}\left[\log \frac{q(\boldsymbol{z}|\boldsymbol{x}, \phi)}{p(\boldsymbol{z})}\right] \\
&= \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x},\phi)}[\log p(\boldsymbol{x}|\boldsymbol{z}, \theta)] - KL[q(\boldsymbol{z}|\boldsymbol{x}, \phi)||p(\boldsymbol{z})]
\end{aligned}$$

As we said before, we are not able to work with the entire dataset and thus we have to consider a mini-batch $\boldsymbol{x}_1, ..., \boldsymbol{x}_m$ and approximate:

$$\mathcal{L}(\phi, \theta) \approx \frac{1}{m} \sum_{j=1}^{m} \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x}_j,\phi)}[\log p(\boldsymbol{x}_j|\boldsymbol{z}, \theta)] - KL[q(\boldsymbol{z}|\boldsymbol{x}_j, \phi)||p(\boldsymbol{z})]$$

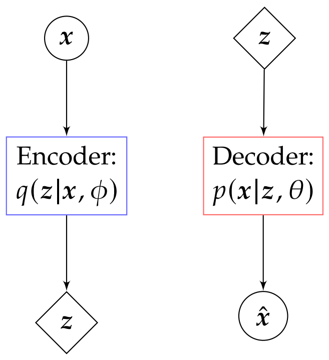
We can try to understand the meaning of this expression:

- The first term is the reconstruction error. It considers how well we are reconstructing $\boldsymbol{x}$ given $\boldsymbol{z}$ sampled from q. This is maximized when $p(\boldsymbol{x}|\boldsymbol{z}, \theta)$ assigns the highest probability to the original instance $\boldsymbol{x}$. We call $p(\boldsymbol{x}|\boldsymbol{z}, \theta)$ **decoder**.

- The second term is a regularization term which encourages the variational distribution to look like a Gaussian distribution and not some kind of identity mapping. This avoids learning a mapping from inputs to latent features that just "stores" the inputs in different regions of the latent space, hence it favours generalization.

Both the terms involve the expectation with respect to the variational distribution of the latent variables that we interpret as features describing the data. So, we call $q(\boldsymbol{z}|\boldsymbol{x}, \phi)$ **encoder**.

We can represent the architecture in the following way:



where the input $\boldsymbol{x}$ is mapped through the encoder into a useful latent space $\boldsymbol{z}$ from which we can reconstruct $\hat{\boldsymbol{x}}$ via the decoder. We would like $\hat{\boldsymbol{x}}$ to be as similar as possible to $\boldsymbol{x}$.

Remark. Notice that the variational parameters are shared across all the datapoints: using global parameters allows us to 'amortize' the cost of inference (**amortized inference**). Instead, in mean-field variational inference we had different parameters for each datapoint. This changes the behaviour of the model when we have a new datapoint: in mean-field VI we need to maximize the ELBO for each new point while here we can keep the global parameters fixed or run the variational inference again (maybe when many points are added).

So far we have obtained an expression for the ELBO, which is the objective function of our optimization. In order to maximize it, we need a good estimate of the gradient and we cannot obtain it directly from the previous expression because the gradient has to be computed on the same variables involved in the expectation. Therefore, it is not possible to swap the gradient and the expectation and this is where the **reparametrization trick** comes in.

We express the variational distribution $q(\boldsymbol{z}|\boldsymbol{x}, \phi)$ through a deterministic transformation $g(\boldsymbol{\varepsilon}, \boldsymbol{x}, \phi)$ which maps a noise variable $\boldsymbol{\varepsilon}$ (distributed accordingly to a distribution we can easily sample from, as $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$) to the distribution of $\boldsymbol{z}$:

$$\boldsymbol{z} = g(\boldsymbol{\varepsilon}, \boldsymbol{x}, \phi)$$

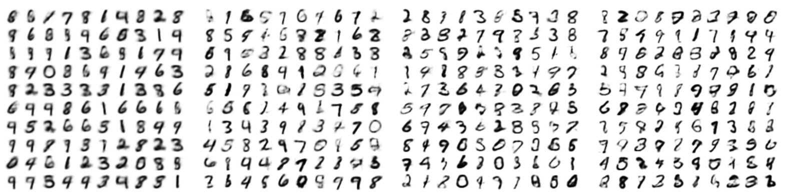

(a) 2-D latent space (b) 5-D latent space (c) 10-D latent space (d) 20-D latent space

Figure 11.2: Random samples from learned generative models (AEVB) of MNIST for different dimensionalities of latent space [11].

Typically, the variational distribution is chosen to be a Gaussian:

$$q(\boldsymbol{z}|\boldsymbol{x},\phi) = \mathcal{N}(\boldsymbol{\mu}_z(\boldsymbol{x},\phi), {\boldsymbol{\sigma}_z}^2(\boldsymbol{x},\phi)\cdot \boldsymbol{I})$$

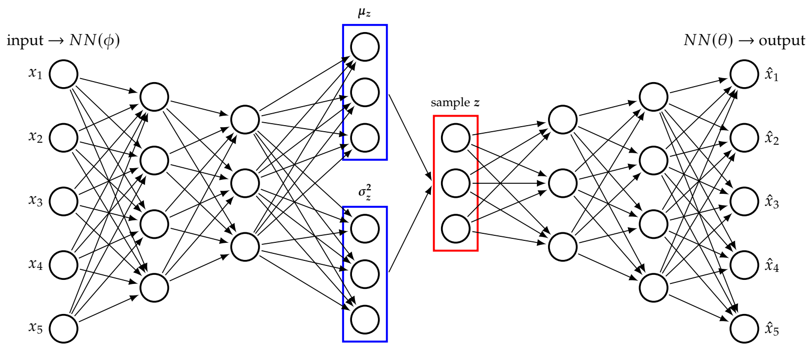
so that we have an analytical expression for $KL[q(\boldsymbol{z}|\boldsymbol{x},\phi)||p(\boldsymbol{z})]$ and we can rewrite $\boldsymbol{z}$ as

$$\boldsymbol{z} = \boldsymbol{\mu}_z(\boldsymbol{x},\phi) + {\boldsymbol{\sigma}_z}^2(\boldsymbol{x},\phi)\cdot \boldsymbol{\varepsilon}$$

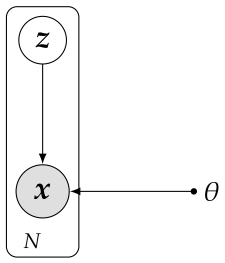
At this point, we can evaluate the gradient of the ELBO as:

$$\nabla_{\theta,\phi}\mathscr{L}(\phi,\theta) = \mathbb{E}_{\varepsilon}[\nabla_{\phi,\theta}\log p(\boldsymbol{x}|g(\boldsymbol{\varepsilon},\boldsymbol{x},\phi),\theta)] - \nabla_{\phi}KL[q(\boldsymbol{z}|\boldsymbol{x},\phi)||p(\boldsymbol{z})]$$

We still have to choose the functions $\boldsymbol{\mu}_z(\boldsymbol{x},\phi)$ and ${\boldsymbol{\sigma}_z}^2(\boldsymbol{x},\phi)$. To have high expressiveness and efficient optimization, we can opt for neural networks: we just built a **variational autoencoder**.

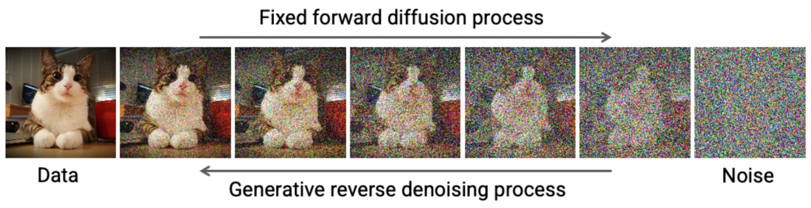


Remark. The variational autoencoder can be interpreted as a directed probabilistic graphical model with latent variables.



11.2 Diffusion Models

Diffusion models are a family of probabilistic generative models that progressively destruct data by injecting noise, then learn to reverse this process for sample generation. In particular, we will discuss denoising diffusion probabilistic models (DDPMs), but many concepts are common to other formulations.



Fixed forward diffusion process
Data
Generative reverse denoising process
Noise

Figure 11.3: Image from [12], *Denoising Diffusion-based Generative Modeling: Foundations and Applications*

11.2.1 Forward Diffusion Process

Given a data point $x_0 \in \mathcal{X}$ sampled from the data distribution $p(x_0)$, consider a Markov chain $p(x_t|x_{t-1})$ with $t \in 1, \dots, T$ satisfying the following properties:

- It is easy to sample from $p(x_t|x_{t-1})$
- For T large enough, $p(x_T)$ is approximately a known distribution from which it is easy to sample and that does not depend on x_0, i.e, $p(x_T) \approx p(x_T|x_0)$. This distribution is referred as the **prior**.

$$\begin{tikzcd} x_0 \arrow[r] & x_1 \arrow[r, dotted, "p(x_t|x_{t-1})"] & x_{T-1} \arrow[r] & x_T \end{tikzcd}$$

Such Markov chain is referred as the **forward diffusion process**, and $p(x_t|x_{t-1})$ is often referred as the **forward transition kernel**. Using the chain rule of probability and the Markov property, we can factorize the joint distribution of $x_0, \dots, x_T$ into:

$$p(x_0, \dots, x_T) = p(x_0) \prod_{t=1}^{T} p(x_t|x_{t-1})$$

In practice, the most used forward process in continuous space is based on the injection of Gaussian noise:

$$p(x_t|x_{t-1}) := \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t \boldsymbol{I})$$

where $\beta_t \in (0, 1)$ regulates the amount of noise that is injected at each step (the larger, the faster the convergence to the prior distribution). A useful property of the above process is that we can sample at any arbitrary time step in a closed form using the reparameterization trick. Given $\alpha_t := 1 - \beta_t$, $\bar{\alpha}_t := \prod_{i=1}^{t} \alpha_i$, and $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ we have

$$\begin{aligned}
\boldsymbol{x}_t &= \sqrt{\alpha_t}\boldsymbol{x}_{t-1} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_{t-1} \\
&= \sqrt{\alpha_t}\left(\sqrt{\alpha_{t-1}}\boldsymbol{x}_{t-2} + \sqrt{1-\alpha_{t-1}}\boldsymbol{\epsilon}_{t-2}\right) + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_{t-1} \\
&= \sqrt{\alpha_t\alpha_{t-1}}\boldsymbol{x}_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\hat{\boldsymbol{\epsilon}}_{t-2} \\
&= \dots \\
&= \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_t}\hat{\boldsymbol{\epsilon}}
\end{aligned}$$

where $\boldsymbol{\epsilon}_{t-2}$ and $\boldsymbol{\epsilon}_{t-1}$ have been merged into $\hat{\boldsymbol{\epsilon}}_{t-2}$.

Hence, we have

$$p(\boldsymbol{x}_t|\boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{x}_t; \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0, 1-\bar{\alpha}_t)$$

This is also useful to show, as $\bar{\alpha}_t \longrightarrow 0$, that

$$\lim_{t\to\infty} p(\boldsymbol{x}_t|\boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{x}_t; \mathbf{0}, \boldsymbol{I})$$

So for a sufficiently large T, $\boldsymbol{x}_T \mathrel{\dot\sim} \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. In other words, this process ends up in corrupting the data into white noise.

11.2.2 Backward Process

Now that we are able to gradually corrupt data points $\boldsymbol{x}_0$ into noise, for generating new data samples we have to revert this process. We have seen that we can easily sample from $p(\boldsymbol{x}_T)$, but sampling from

$$p(\boldsymbol{x}_0, \dots, \boldsymbol{x}_T) = p(\boldsymbol{x}_T)\prod_{t=1}^{T} p(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$$

is non-trivial, since we do not have access to the reverse transition kernels $p(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$* that would allow us to perform ancestral sampling as in the forward process. Therefore, the reverse transition kernels have to be learned. In order to do this, we fit parametric distributions $q_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$, from which it easy to sample and that can be easily computed, for example:

$$q_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) := \mathcal{N}(\boldsymbol{x}_{t-1}; \mu_\theta(\boldsymbol{x}_t, t), \Sigma_\theta(\boldsymbol{x}_t, t))$$

where the mean and covariance functions are neural networks.

The objective is then to find parameters θ such that

$$q_\theta(\boldsymbol{x}_0, \dots, \boldsymbol{x}_T) := p(\boldsymbol{x}_T)\prod_{t=1}^{T} q_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) \approx p(\boldsymbol{x}_0, \dots, \boldsymbol{x}_T)$$

* It is easy to prove that the reverse of a Markov chain is also a Markov chain.

$$\begin{tikzcd} x_0 & x_1 \arrow[l] & x_{T-1} \arrow[l, dotted, "q_\theta(x_{t-1}|x_t)"'] & x_T \arrow[l] \end{tikzcd}$$

The parametric kernels are learned by minimizing the Kullback-Leibler divergence between the forward and backward processes:

$$\begin{aligned} D_{KL}(p(\boldsymbol{x}_0, \dots, \boldsymbol{x}_T) || q_\theta(\boldsymbol{x}_0, \dots, \boldsymbol{x}_T)) &= \mathbb{E}_{p(\boldsymbol{x}_0,\dots,\boldsymbol{x}_T)}\left[\log \frac{p(\boldsymbol{x}_0, \dots, \boldsymbol{x}_T)}{q_\theta(\boldsymbol{x}_0, \dots, \boldsymbol{x}_T)}\right] \\ &= -\mathbb{E}_{p(\boldsymbol{x}_0,\dots,\boldsymbol{x}_T)}\left[\log q_\theta(\boldsymbol{x}_0, \dots, \boldsymbol{x}_T)\right] + \text{const} \\ &= -\mathbb{E}_{p(\boldsymbol{x}_0,\dots,\boldsymbol{x}_T)}\left[\sum_{t=0}^{T} \log q_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)\right] + \text{const} \end{aligned}$$

Remark. The forward process $p(\boldsymbol{x}_0, \dots, \boldsymbol{x}_T)$ was considered constant with respect to the parameters θ. Although parameters of the forward process (as β_t) are usually considered fixed (hyperparameters), it is also possible to learn them. In that case we cannot consider the forward process as constant with respect to the parameters.

Minimizing this inverse KL divergence is equivalent to solving a maximum likelihood estimation problem: we can sample full trajectories starting from samples from the dataset and use them to fit parametric functions q_θ by stochastic optimization, as in a supervised learning problem.

Remark. It can be shown that, if forward transition kernels corrupts the data slowly enough (T is large), then the reverse transition kernels will be approximately Gaussian. This is what makes the approximation of reverse transition kernels possible and approachable as a prediction problem. Conversely, if forward transition kernels add too much noise, the reverse transition probabilities can be multimodal, hence a parametric approximation would be intractable.

11.2.3 Denoising parametrization

There is empirical evidence that learning directly the backward transition kernels $q_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$ is not the most effective strategy. Alternatively, it is possible to write the backward transition kernel in the following way:

$$p(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) = \int_{\boldsymbol{x}_0 \in \mathcal{X}} p(\boldsymbol{x}_{t-1}, \boldsymbol{x}_0|\boldsymbol{x}_t) d\boldsymbol{x}_0 = \int_{\boldsymbol{x}_0 \in \mathcal{X}} p(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0) p(\boldsymbol{x}_0|\boldsymbol{x}_t) d\boldsymbol{x}_0$$

This means that we can sample $\boldsymbol{x}_{t-1}$ if we can sample $\boldsymbol{x}_0 \sim p(\boldsymbol{x}_0|\boldsymbol{x}_t)$ and plug it into $p(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0)$. It is possible to show that for the Gaussian diffusion process introduced above the distribution $p(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0)$ is also a Gaussian and can be computed analytically:

$$p(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{x}_{t-1}; \mu_t(\boldsymbol{x}_t, \boldsymbol{x}_0), \sigma_t \boldsymbol{I})$$

$$\mu_t(\boldsymbol{x}_t, \boldsymbol{x}_0) = \frac{\sqrt{\alpha_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t}\boldsymbol{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1 - \bar{\alpha}_t}\boldsymbol{x}_0$$

$$\sigma_t = \frac{(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\beta_t$$

Instead $p(\boldsymbol{x}_0|\boldsymbol{x}_t)$ (the denoising kernel) has to be learned, again by fitting a parametric distribution $q_\theta(\boldsymbol{x}_0|\boldsymbol{x}_t)$. Interestingly, this works despite $p(\boldsymbol{x}_0|\boldsymbol{x}_t)$ being highly multimodal and non-Gaussian, so only a "mean" approximation is possible.

However, it is common in DDPMs to avoid predicting directly $\boldsymbol{x}_0$ given $\boldsymbol{x}_t$. In fact, a model is trained to predict the noise that was added to $\boldsymbol{x}_0$ in order to obtain $\boldsymbol{x}_t$. Recall the property following from $p(\boldsymbol{x}_t|\boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{x}_t; \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0, 1-\bar{\alpha}_t)$:

$$\boldsymbol{x}_t = \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t \text{ with } \boldsymbol{\epsilon}_t \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$$

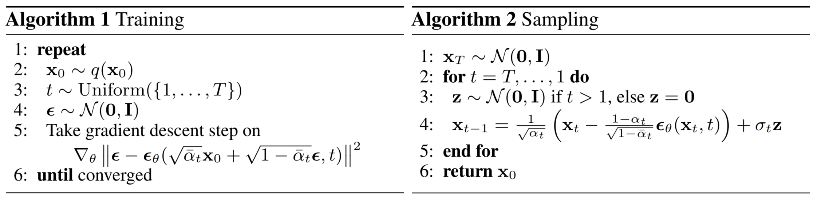
We can rewrite the mean of $p(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0)$, or $p(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{\epsilon}_t)$, as

$$\mu_t(\boldsymbol{x}_t, \boldsymbol{\epsilon}_t) = \frac{1}{\sqrt{\alpha_t}}\left(\boldsymbol{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_t\right)$$

Now we can fit a parametric model $\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t)$ to approximate $p(\boldsymbol{\epsilon}_t|\boldsymbol{x}_t)$. This is usually done in a simplified way by minimizing the squared error:

$$\begin{aligned}\mathcal{L}(\theta) &= \mathbb{E}_{t,\boldsymbol{x}_0,\boldsymbol{\epsilon}_t}\left[\|\boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t)\|^2\right] \\ &= \mathbb{E}_{t,\boldsymbol{x}_0,\boldsymbol{\epsilon}_t}\left[\left\|\boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta\left(\sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t, t\right)\right\|^2\right]\end{aligned}$$

where the time step t is sampled uniformly in $\{1, \ldots, T\}$, $\boldsymbol{x}_0$ is randomly sampled from the dataset, and $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. The training and sampling algorithm are then summarized as follows:



```
Algorithm 1 Training
1: repeat
2:   $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3:   $t \sim \mathrm{Uniform}(\{1, \ldots, T\})$
4:   $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5:   Take gradient descent step on
         $\nabla_\theta \left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t)\right\|^2$
6: until converged
```

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

Figure 11.4: The training and sampling algorithms in DDPMs (image from [13]).

Remark. The empirical advantage of predicting the noise $\boldsymbol{\epsilon}_t$ instead of $\boldsymbol{x}_0$ could be due to the fact that it is easier to train a neural network when the target output is (marginally) distributed according to a normal distribution.

Summary

A DDPM makes use of two Markov chains: a forward chain that perturbs data to noise, and a reverse chain that converts noise back to data. The forward Markov chain is designed with the goal to gradually transform any data distribution into a simple prior distribution (e.g., standard Gaussian). The backward Markov chain instead reverses the former starting from the known distribution and gradually converging to the data distribution. Since the backward Markov chain is unknown, it is learned by a deep neural network using examples generated with the forward process. New data points are subsequently generated by first sampling a random vector from the prior distribution, followed by ancestral sampling through the reverse Markov chain.

11.2.4 Diffusion in discrete space

Despite being originally thought for data in continuous space, it is also possible to define a diffusion process in discrete space [14], even though its usefulness is questionable and subject of current research. Assuming without loss of generality data $\boldsymbol{x} \in \{1, 2, \ldots, c\}^d$, there are two main kinds of discrete diffusion processes:

- **Uniform**: At each time step t, every component x_t^i of $\boldsymbol{x}_t$ switches to a random value with a given small probability ϵ_t. The prior distribution of such diffusion process is the uniform over $\{1, 2, \ldots, c\}^d$.
- **Absorbing**: At each time step t, every component x_t^i of $\boldsymbol{x}_t$ switches to a particular value called the "mask", often referred as the [MASK] token. The prior distribution of such diffusion process is the single point $[\text{MASK}]^d$, that is the absorbing state of the Markov chain.
  A variant of the absorbing diffusion process is when at each time step t the t^{th} component x_t^t is masked. Modelling the corresponding reverse process is then equivalent to fitting an autoregressive model.

11.2.5 Score-based diffusion models

Given the objective to sample from the distribution $p(\boldsymbol{x})$ that generated the data, score-based models models aim at estimating the so called *score* of $p(\boldsymbol{x})$, defined as $\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x})$ and then use sampling techniques that exploit the knowledge of the score of the distribution.

An example of score-based model is diffusion with stochastic differential equations (SDE), a generalization in continuous time of the Markov chain diffusion process discussed above, where the transition kernel $p(\boldsymbol{x}_{t+\epsilon}|\boldsymbol{x}_t)$ is implicitly defined through the following SDE:

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

where $\mathrm{d}\boldsymbol{w} \sim \mathcal{N}(0, \boldsymbol{I}\mathrm{d}t)$ is a Wiener process.

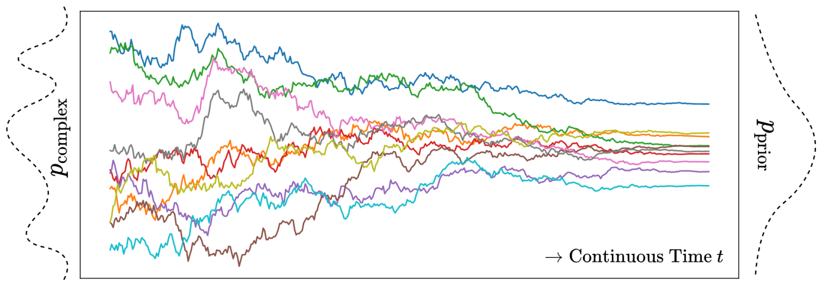


Figure 11.5: Image from [16], *An introduction to Diffusion Probabilistic Models*.

Generation of new data points reduces to solving the inverse-time SDE, that it can be proven to be:

$$\mathrm{d}\boldsymbol{x} = \left[\boldsymbol{f}(\boldsymbol{x}, t) - g^2(t)\nabla_{\boldsymbol{x}} \log p_t(\boldsymbol{x})\right] \mathrm{d}t + g(t)\mathrm{d}\boldsymbol{w}$$

For this we need to estimate the time dependent score $\nabla_{\boldsymbol{x}} \log p_t(\boldsymbol{x})$, for example using a neural network $s_\theta(\boldsymbol{x}, t)$. The most popular method is *denoising score matching*, where the following loss in minimized:

$$\mathbb{E}_{t\sim u(0,1), \boldsymbol{x}_0 \sim p(\boldsymbol{x}_0), \boldsymbol{x}_t \sim p(\boldsymbol{x}_t|\boldsymbol{x}_0)} \left\|\boldsymbol{s}_\theta(\boldsymbol{x}_t, t) - \nabla_{\boldsymbol{x}_t} \ln p(\boldsymbol{x}_t \mid \boldsymbol{x}_0)\right\|$$

that is deeply connected with the denoising loss of DDPMs.