Lab 7: Generative Diffusion Models

Shervin Khalafi and Alejandro Ribeiro*

October 28, 2024

1 Data Probability Distributions

In several of the past chapters we have discussed the difference between the data on which a model is trained and the data on which a model is executed. We have emphasized that these two sets are different but we have not discussed much how they are related. Their relationship is that both of them are sampled from a common probability distribution.

A probability distribution $p(x)$ is a function that assigns values to the likelihood of observing different possible outcomes x. The likelihood is large for signals that are likely and small for signals that are unlikely. As an example, consider a bag that contains all of the possible digits that have, are, or will ever be written. The first image in Figure 1 has a high likelihood as it is a fairly typical handwritten number five. The second image has a smaller likelihood. It is a discernible number five, but not a typical one. The third image has negligible likelihood, maybe even zero likelihood, because it does not look like a digit.

It is convened that likelihoods are normalized so that they integrate to 1,

$$\int p(x)\, d\mathbf{x} = 1. \tag{1}$$

The meaning of this normalization is that when we put our hand in the bag we extract one and only one digit. Thus, the sum (integral) of all likelihoods must be exactly one.

*In alphabetical order.

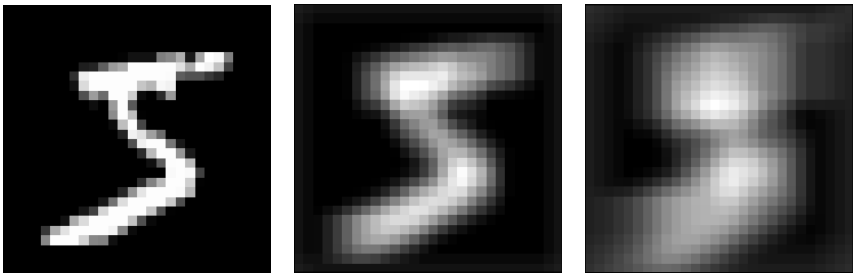

Figure 1. Probability Distributions. A probability distribution $p(x)$ is a function that assigns values to the likelihood of observing different possible outcomes x. In the bag that contains all of the digits that have, are, or will ever be written the first image has high likelihood, the second one intermediate likelihood and the third one small likelihood.

When we train an AI model it is implicit that training samples are extracted from an underlying probability distribution $p(x)$ which is the same distribution from which future samples will be extracted. In digit classification we pull handwritten digits from the bag of all digits that have, are, or will ever be written. We use this digits to train a CNN that we will then execute on any digits that are extracted from this same bag.

In this chapter we are after a more ambitious AI task than in previous chapters. We want to train an AI system that replicates the probability distribution itself. We are not just after classifying digits pulled from the bag of all digits that have, are, or will ever be written. We want to create an artificial bag from where we can pull digits with a distribution of likelihoods equivalent to the distribution of likelihoods of the natural system.

This is the most ambitious AI task we can conceive. If we succeed, we can prescind of reality altogether. Tackling this task requires that we understand distributions a little better and that we tackle the difference between statistical and empirical risk. We do that in the following before discussing generative models in Section 7.

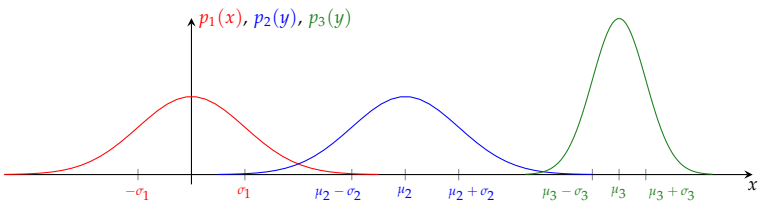

Figure 2. Gaussian (Normal) Probability Distributions in 1 Dimension. Normal distributions have bell shapes and are determined by two parameters, the mean μ and the variance σ^2. The mean is the value with the highest likelihood and the variance is the spread of the likelihood around the mean.

2 Gaussian Distributions

A *Gaussian* probability distribution with mean μ and variance σ^2 is a probability distribution over scalar values x; see Figure 2. This probability distribution has the formula

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(x-\mu^2)}{2\sigma^2}\right]. \tag{2}$$

All Gaussian distributions look like crosssections of a bell. The mean μ determines the center of the bell and the variance σ^2 the concentration of the bell around its mean. Smaller variances imply larger concentrations. Gaussian distributions are also called *normal* distributions.

In Figure 2 the probability distribution $p_1(x)$ is normal and has mean $\mu = 0$ and variance $\sigma^2 = 1$. We see in the figure that values around $x = \mu = 0$ are the most likely to be observed, that values around x and $-x$ are equally likely, and that values much larger that $x = 3\sigma = 3$ are unlikely. Distribution $p_2(x)$ in Figure 2 has mean $\mu = 4$ and variance $\sigma^2 = 1$. It has the same shape of distribution $p_1(x)$ but is shifted to the right. This is because they have the same variance while the mean is of $p_2(x)$ is higher. Distribution $p_3(x)$ has mean $\mu = 8$ and variance $\sigma^2 = 1/4$. In addition to been further shifted to the right, distribution $p_3(x)$ is more concentrated around its mean. This is because its variance is smaller than the variance of $p_1(x)$ and $p_2(x)$.

2.1 Multivariate Gaussian Distributions

Our interest extends to probability distributions across vectors $\mathbf{x} \in \mathbb{R}^n$. In this case we define a vector of means $\boldsymbol{\mu} \in \mathbb{R}^n$ and a covariance matrix $\mathbf{C} \in \mathbb{R}^{n \times n}$. A *multivariate Gaussian* probability distribution assigns likelihoods to samples according to the formula,

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{n/2} \det^{1/2}(\mathbf{C})} \exp\left[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \mathbf{C} (\mathbf{x}-\boldsymbol{\mu})\right] = \mathcal{N}\Big(\mathbf{x}; \boldsymbol{\mu}, \mathbf{C}\Big). \tag{3}$$

In this formula $\det^{1/2}(\mathbf{C})$ is the square root of the determinant of the covariance matrix and the product $(\mathbf{x}-\boldsymbol{\mu})^T \mathbf{C} (\mathbf{x}-\boldsymbol{\mu})$ evaluates to a number that we exponentiate. The matrix $\mathbf{C} = \mathbf{C}^T$ is symmetric.

However complicated, the formula for the normal distribution is just a formula. To save the time of writing (3), we use the shorthand

$$p(\mathbf{x}) = \mathcal{N}\Big(\mathbf{x};\, \boldsymbol{\mu},\, \mathbf{C}\Big), \tag{4}$$

to denote a normal distribution with mean $\boldsymbol{\mu}$ and covariance matrix $\mathbf{C}$ evaluated at $\mathbf{x}$.

Multivariate Gaussian distributions have bell shapes. Or what we imagine a bell is in high dimensions. What we can visualize is that one and two dimensional crosssections of a multivariate Gaussian have bell shapes. In any case, the mean $\boldsymbol{\mu}$ of a multivariate normal distribution represents the vector $\mathbf{x}$ with the highest likelihood. The covariance matrix quantifies the concentration of the distribution around its mean. Since we are in multiple dimensions the concentration can be different in different directions.

An important particular case is when the covariance matrix $\mathbf{C} = \sigma^2 \mathbf{I}$ is a scaled identity. In this case the determinant is $\det(\mathbf{C}) = \sigma^{2n}$ and the product in the exponent is $(\mathbf{x}-\boldsymbol{\mu})^T \mathbf{C} (\mathbf{x}-\boldsymbol{\mu}) = \|\mathbf{x}-\boldsymbol{\mu}\|^2$. These substitutions yield the probability distribution

$$p(\mathbf{x}) \;=\; \frac{1}{(2\pi\sigma^2)^{n/2}} \exp\left[-\frac{1}{2\sigma^2}\|\mathbf{x}-\boldsymbol{\mu}\|^2\right] \;=\; \mathcal{N}\Big(\mathbf{x};\, \boldsymbol{\mu},\, \sigma^2\mathbf{I}\Big). \tag{5}$$

This is a symmetric distribution. The likelihood of observing a vector $\mathbf{x}$ depends on its distance $\|\mathbf{x}-\boldsymbol{\mu}\|$ to the mean only. It is the same in any direction. We call this distribution white and we say that the scalar σ^2 is

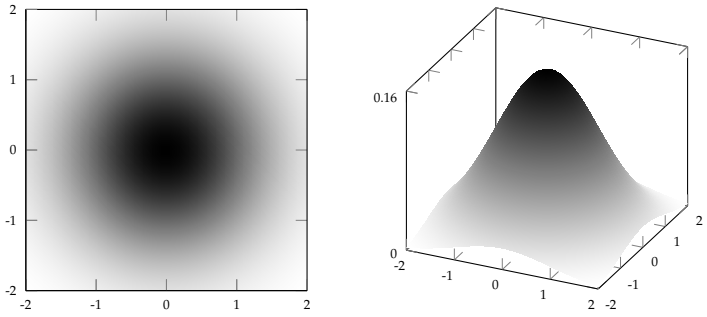

Figure 3. Standard White Multivariate Gaussian (Normal) Distribution in 2 Dimensions. In a white multivariate Gaussian the likelihood of observing $\mathbf{x}$ depends on the distance between $\mathbf{x}$ and the mean $\boldsymbol{\mu}$ but it is otherwise the same in all directions [cf. (5)]. Since this is a standard white distribution the mean is $\boldsymbol{\mu} = \mathbf{0}$ and the covariance is $= \mathbf{I}$.

its variance. In the particular case when the variance is $\sigma^2 = 1$ and the mean is $\boldsymbol{\mu} = \mathbf{0}$ we further say that the distribution is standard. We show a standard white distribution in Figure 3 when $n = 2$.

Distributions that are not white are called colored. Figure 5 depicts colored Gaussian distributions in two dimensions. All of the distributions have zero mean but different covariance matrices. Since we are in two dimensions, the covariance matrix $\mathbf{C}$ has 2 rows and 2 columns. We write it explicitly as

$$\mathbf{C} = \begin{pmatrix} \sigma_{11}^2 & \sigma_{12} \\ \sigma_{12} & \sigma_{22}^2 \end{pmatrix} \tag{6}$$

The first distribution in Figure 5 is more spread along the horizontal axis and more concentrated along the vertical axis. It corresponds to a covariance matrix with $\sigma_{11}^2 = 4$, $\sigma_{22}^2 = 36$ and $\sigma_{12} = 0$. The second distribution in Figure 5 flips the axes. It is more spread along the vertical axis and more concentrated along the horizontal axis. It corresponds to a covariance matrix with $\sigma_{11}^2 = 36$, $\sigma_{22}^2 = 4$ and $\sigma_{12} = 0$. The third distribution is a rotated version of the previous two. It is spread out along the ascending diagonal and concentrated in the descending diagonal. It corresponds to $\sigma_{11}^2 = 20$, $\sigma_{22}^2 = 20$ and $\sigma_{12} = 16$.

Task 1 Generate $N = 10^3$ samples from a standard multivariate white normal distribution in dimension $n = 2$. Each of these samples is a vector

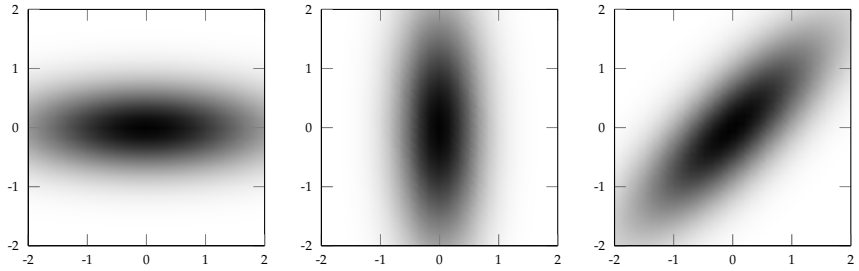

Figure 4. Multivariate Gaussian (Normal) Distributions in 2 Dimensions. Multivariate Gaussians can be skewed. The skew is determined by the entries of the covariance matrix [cf. (6)].

with two entries. Plot these samples on the plane and compare their density to their probability distribution $p(\mathbf{x})$ [cf. (5) with $n = 2$, $\boldsymbol{\mu} = \mathbf{0}$ and $\sigma^2 = 1$]. You should observe that samples accumulate in places where the likelihood $p(\mathbf{x})$ is large.

3 Expectations and Statistical Risks

An expectation is an average weighted by likelihoods. Suppose that we are working with data $\mathbf{x}$ which has a probability distribution $p(\mathbf{x})$. For each data point $\mathbf{x}$ we evaluate a function $f(\mathbf{x})$. The expected value of this function over the data distribution is the average

$$\mathbb{E}_{\mathbf{x}\sim p}(f(\mathbf{x})) = \int f(\mathbf{x})p(\mathbf{x})\,d\mathbf{x}. \tag{7}$$

In this expression, function values $f(\mathbf{x})$ are weighted by the likelihood $p(\mathbf{x})$ of observing $\mathbf{x}$. The expectation is an integral over these weighted function values.

The expectation in (7) is to be contrasted with the sample mean. This is an average over a set of data points drawn from the distribution $p(\mathbf{x})$. Formally, consider N points $\mathbf{x}_n$ drawn from the bag described by $p(\mathbf{x})$.

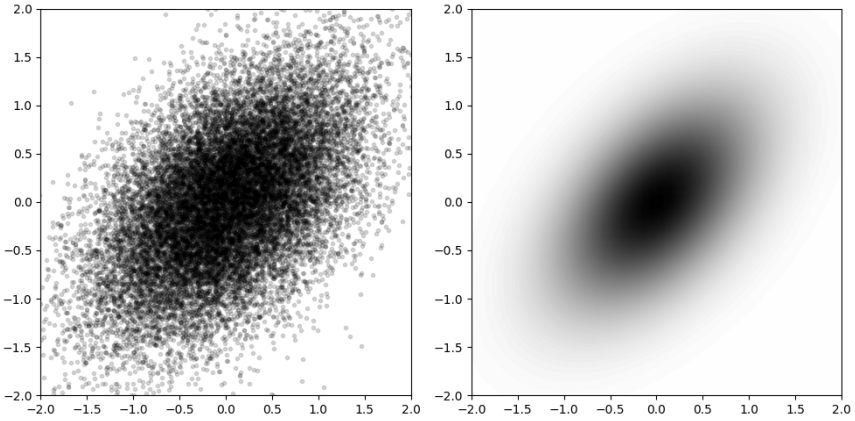

Figure 5. Comparison of a Gaussian probability distribution (right), and a scatter plot of samples from this distribution (left). The regions to which the distribution assigns larger likelihood (darker areas), also have a larger concentration of samples.

The sample mean of this set of points is the simple average

$$\bar{f} = \frac{1}{N} \sum_{n=1}^{N} f(\mathbf{x}_n). \tag{8}$$

The most important result in the theory of probability, some would say the only important result, is the proximity between the expectation $\mathbb{E}(f(\mathbf{x}))$ and the sample mean $\bar{f}$ when the points $\mathbf{x}_n$ are drawn independently. This fact is called the law of large numbers.

The law of large numbers is remarkable because expectations and sample means are two quantities of fundamentally different natures. Whereas the expectation is a property of the probability distribution, the sample mean is a property of the samples. Different sets of data have different sample means. The law of large numbers says that all of these sample means are close to the expectation and, by extension, close to each other.

Ultimately, this is the reason why training on empirical risks is justifiable. When we train a model on the empirical risk,

$$w^*_{\text{ERM}} \;=\; \underset{w}{\text{argmin}} \;\frac{1}{N} \sum_{i=1}^{N} \; c\Big(\Phi(\mathbf{x}_i, w) \Big), \tag{9}$$

the hope is that the solution of empirical risk minimization (ERM) is close to the solution of the statistical risk minimization (SRM) problem

$$w^*_{\text{SRM}} \;=\; \underset{w}{\text{argmin}}\; \mathbb{E}_{\mathbf{x}\sim p}\Big[\, c\Big(\, \Phi(\mathbf{x}, w)\,\Big)\,\Big]. \tag{10}$$

Again, (9) and (10) are fundamentally different problems. If we think of (9) and (10) as models of digit classification problems, the statistical risk is an expectation (an average) over the bag that contains all of the digits that have, are, or will ever be written. The empirical risk is an average over a set of digits pulled out of this bag. The SRM solution w^*_{SRM} is a property of the bag. The ERM solution w^*_{ERM} is a property of the specific set of data samples that are pulled from the bag.

The fundamental result of statistical learning theory is that the solutions of ERM and SRM problems are close. In particular, this result implies that training and execution performance are similar. Both are close to SRM performance. That SRM and ERM is not a ready consequence of the law of large numbers. It is a rather involved consequence which requires restrictions on the choice of the learning parameterization.

It is important to realize that the SRM problem in (10) is the problem that we *would like* to solve whereas the ERM problem in (9) is the problem that we *can* solve. Solving SRM requires knowing the probability distribution $p(\mathbf{x})$ which is in general unknown. In Figure 1 we can ascertain with confidence that likelihoods decrease as we move from left to right. However, we cannot ascertain with any confidence the exact likelihood of drawing any of these samples. What we can do is pull samples from the bag of the digits that have, are, or will ever be written, This is ERM.

4 Generative Models

Given a probability distribution $q(\mathbf{x})$ we want to learn a probability distribution $p(\mathbf{x}, w)$ that imitates $q(\mathbf{x})$. This goal can be subsumed within the definition of artificial intelligence (AI) as the imitation of a natural system. The distribution $q(\mathbf{x})$ is a natural system such as the bag of all digits. We want to learn a distribution $p(\mathbf{x}, w)$ that generates digits according to the same likelihood distribution. We call $p(\mathbf{x}, w)$ a generative model because, once trained, we can use it to generate (artificial) data

that is indistinguishable from the (natural) data that we could sample from $q(\mathbf{x})$ – modulo the accuracy of the training model.

Designing this generative AI requires the selection of a parameterization and the selection of a loss. We discuss in this section the choice of loss.

4.1 Kullback–Leibler (KL) divergence

We compare distributions $q(\mathbf{x})$ and $p(\mathbf{x}, w)$ with the Kullback–Leibler (KL) divergence which we define as

$$D_{\text{KL}}\Big(q(\mathbf{x})\|p(\mathbf{x},w)\Big) = \mathbb{E}_{\mathbf{x}\sim q}\log\left[\frac{q(\mathbf{x})}{p(\mathbf{x},w)}\right] = \int q(\mathbf{x})\log\left[\frac{q(\mathbf{x})}{p(\mathbf{x},w)}\right]d\mathbf{x}. \tag{11}$$

The ratio $p(\mathbf{x}, w)/q(\mathbf{x})$ is a relative comparison of the similarity between the likelihoods $p(\mathbf{x}, w)$ and $q(\mathbf{x})$. This comparison is passed through a logarithm and averaged over the distribution $q(\mathbf{x})$.

To use the KL divergence as a loss we need to make sure that it is indeed a loss. We show in the following proposition that this is true

Proposition 1 *The KL divergence between distributions $q(\mathbf{x})$ and $p(\mathbf{x}, w)$ is nonnegative,*

$$D_{KL}\Big(q(\mathbf{x})\|p(\mathbf{x},w)\Big) \geq 0 \tag{12}$$

with equality attained when $q(\mathbf{x}) = p(\mathbf{x}, w)$ for all x.

Proof: To show that the KL divergence is nonzero rewrite its definition in (11) as

$$\int q(\mathbf{x})\log\left[\frac{q(\mathbf{x})}{p(\mathbf{x},w)}\right]d\mathbf{x} = -\int q(\mathbf{x})\log\left[\frac{p(\mathbf{x},w)}{q(\mathbf{x})}\right]d\mathbf{x}. \tag{13}$$

This fact holds because we reversed the ratio $q(\mathbf{x})/p(\mathbf{x}, w)$ inside the logarithm but compensated by adding a negative sign at the front.

To proceed, remember that the logarithm function satisfies $\log(\alpha) \leq \alpha - 1$ for any $\alpha > 0$. Using this fact in (13) we see that

$$-\int q(\mathbf{x})\log\left[\frac{p(\mathbf{x},w)}{q(\mathbf{x})}\right]d\mathbf{x} \;\geq\; -\int q(\mathbf{x})\left[\frac{p(\mathbf{x},w)}{q(\mathbf{x})} - 1\right]d\mathbf{x} \tag{14}$$

Notice now that in the right hand side we can simplify terms to obtain

$$-\int q(\mathbf{x})\left[\frac{p(\mathbf{x},w)}{q(\mathbf{x})}-1\right]d\mathbf{x} \;=\; -\int p(\mathbf{x},w)+\int q(\mathbf{x}) \;=\; 0. \tag{15}$$

The second equality is true because $p(\mathbf{x},w)$ and $q(\mathbf{x})$ are probability distributions that integrate to 1 [cf. (1)].

To see that equality holds when $q(\mathbf{x}) = p(\mathbf{x},w)$ note that if this is true the ratio $p(\mathbf{x},w)/q(\mathbf{x}) = 1$ and the logarithm of this ratio equals zero. Integrating zeroes over all x is still a zero. ■

Proposition 1 shows that the KL divergence is a valid loss for the problem of imitating the natural distribution $q(\mathbf{x})$ with the distribution $p(\mathbf{x},w)$. The KL divergence is minimized at $D_{\text{KL}}(q(\mathbf{x})\|p(\mathbf{x},w)) = 0$ when $q(\mathbf{x}) = p(\mathbf{x},w)$.

The KL divergence is more than a valid loss to compare distributions $q(\mathbf{x})$ and $p(\mathbf{x},w)$. It is a divergence. The meaning of this distinction is not relevant to us, but it is worth pointing out. A notable observation is that KL divergences are not necessarily symmetric,

$$D_{\text{KL}}\Big(q(\mathbf{x})\|p(\mathbf{x},w)\Big) \not\equiv D_{\text{KL}}\Big(p(\mathbf{x},w)\|q(\mathbf{x})\Big) \tag{16}$$

Comparing $p(\mathbf{x},w))$ to $q(\mathbf{x})$ with KL divregences is not the same as comparing $q(\mathbf{x})$ to $p(x,w)$. The choice we make in (17) of comparing $p(\mathbf{x},w))$ to $q(\mathbf{x})$ is not arbitrary. Its motivation will become clear in the next section.

4.2 Risk Minimization for Generative Models

Having established that the KL divergence is minimized when $q(\mathbf{x}) = p(\mathbf{x},w)$, we formulate the learning of a generative model as the statistical risk minimization problem

$$w^*_{\text{SRM}} \;=\; \operatorname*{argmin}_w D_{\text{KL}}\Big(q(\mathbf{x})\|p(\mathbf{x},w)\Big) \;=\; \operatorname*{argmin}_w \mathbb{E}_{\mathbf{x}\sim q}\log\left[\frac{q(\mathbf{x})}{p(\mathbf{x},w)}\right]. \tag{17}$$

As with any other SRM problem, (17) is a problem that we would like to solve but one that we cannot solve because we do not know the probability distribution $q(\mathbf{x})$. We can, however, resort to the acquisition of data,

the formation of a training set, and the formulation of the empirical risk minimization problem,

$$w^* \;=\; \operatorname*{argmin}_w \frac{1}{N} \sum_{n=1}^{N} \log \left[\frac{q(\mathbf{x}_n)}{p(\mathbf{x}_n, w)} \right]. \tag{18}$$

This ERM formulation is not workable because we do not know $q(\mathbf{x}_n)$. When we pull the samples in Figure 1 from the bag of digits, we do not know how likely they are. If we did know $q(\mathbf{x}_n)$, we would be solving the SRM problem in (17).

This issue has a simple solution. Return to the definition of the KL divergence and split the logarithm of the ratio to write

$$\mathbb{E}_{\mathbf{x}\sim q} \log \left[\frac{q(\mathbf{x})}{p(\mathbf{x}, w)} \right] = \mathbb{E}_{\mathbf{x}\sim q} \Big[\log q(\mathbf{x}) - \log p(\mathbf{x}, w) \Big]. \tag{19}$$

In (19) we sum the logarithm of the natural distribution $\log q(\mathbf{x})$ and the artificial distribution $\log p(\mathbf{x}, w)$. Since the logarithm of the natural distribution $\log q(\mathbf{x})$ is independent of the choice of parameter w, its presence in the SRM problem is moot. The value of $\mathbb{E}_{\mathbf{x}\sim q} \log q(\mathbf{x})$ is the same irrespectively of our choice of the parameter w. We have thus shown that the SRM problem in (17) is equivalent to

$$w^*_{\text{SRM}} \;=\; \operatorname*{argmin}_w -\mathbb{E}_{\mathbf{x}\sim q} \Big[\log p(\mathbf{x}, w) \Big]. \tag{20}$$

This is an SRM problem with loss $-\log p(\mathbf{x}, w)$. This loss is unusual. It can be negative and it is not directly comparing $p(\mathbf{x}, w)$ to $p(x)$. It is just a function of the probability $p(\mathbf{x}, w)$. Its use is nevertheless justified because (20) is equivalent to (17) and we have seen that the KL divergence in (17) is a valid loss. The distribution that minimizes (20) is $p(x)$ and this all that matters in the end.

Since the statistical problems in (17) and (20) are equivalent we can write an empirical version of (20) that will be equivalent to the ERM in (18). This problems takes the form

$$w^* \;=\; \operatorname*{argmin}_w -\frac{1}{N} \sum_{n=1}^{N} \log p(\mathbf{x}_n, w). \tag{21}$$

In this ERM problem the distribution $q(\mathbf{x}_n)$ is not present. We can solve it by pulling samples from the bag of all digits without having to know the

likelihood of the samples. Take a moment to marvel at (21). We are solving a very difficult problem – the imitation of a probability distribution from a set of samples – with the minimization of a very simple objective – a sum of logarithms of the estimated likelihoods of each sample.

5 Generative Normal Distribution Models

In (21), the distribution $p(\mathbf{x}, w)$ is a family of distributions parameterized by w. We choose here to work with white normal distributions with covariance $\mathbf{C} = \mathbf{I}$ and variable mean $\boldsymbol{\mu}$,

$$p(\mathbf{x}, w) = p(\mathbf{x}, \boldsymbol{\mu}) = \frac{1}{(2\pi)^{n/2}} \exp\left[-\frac{1}{2}\|\mathbf{x} - \boldsymbol{\mu}\|^2\right]. \tag{22}$$

We are now given a set of points $\mathbf{x}_n$ drawn from an unknown distribution $q(\mathbf{x})$ and are asked to find the distribution $p(\mathbf{x}, w) = p(\mathbf{x}, \boldsymbol{\mu})$ within the class defined in (22) that is closest to $q(x)$. This goal can be attained by solving the ERM problem in (21) with $p(\mathbf{x}, w) = p(\mathbf{x}, \boldsymbol{\mu})$ as given in (22). After eliminating unnecessary constants this substitution yields the ERM problem

$$\boldsymbol{\mu}^* = \operatorname*{argmin}_{\boldsymbol{\mu}} -\frac{1}{2N} \sum_{n=1}^{N} \|\mathbf{x}_n - \boldsymbol{\mu}\|^2. \tag{23}$$

We say that the probability distribution $p(\mathbf{x}, \boldsymbol{\mu}^*)$ is a generative normal distribution model (GNDM). It is a *generative model* because we can use it to generate data from the same distribution that generated the samples $\mathbf{x}_n$. It is generative *normal distribution* model because it is fitting a normal distribution to the data samples $\mathbf{x}_n$.

GNDM training is a very simple optimization problem to solve. It's simplicity stems from the fact that the Gaussian family of distributions in (22) is not very rich. We are using this simple parameterization because it is a preliminary approach to the diffusion models of Section 6.

Task 2 Let $q(\mathbf{x})$ be a 2D normal distribution with covariance $\mathbf{C} = \mathbf{I}$ and mean $\boldsymbol{\mu} = [1.0; 2.0]$. Generate $m = 10^3$ samples $\mathbf{x}_n$ of the distribution $q(\mathbf{x})$. Solve (23) for this dataset. Compare the learned distribution $p(\mathbf{x}, \boldsymbol{\mu}^*)$ with the dataset. They should match closely.

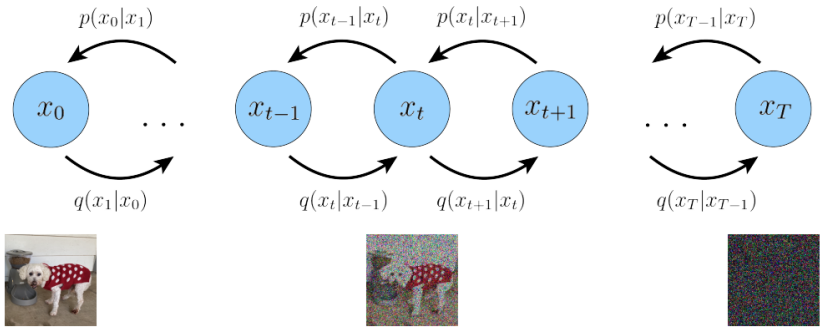

Figure 6. Generative Diffusion Processes.

6 Generative Diffusion Models

In Task 2 we succeed at learning a Gaussian probability distribution $p(\mathbf{x}, \boldsymbol{\mu}^*)$ that matches samples $\mathbf{x}_n$ of an unknown Gaussian probability distribution $q(\mathbf{x})$. In practice, we want to learn distributions $q(\mathbf{x})$ that are more complex than Gaussian distributions. Say, the probability distribution of all the digits that have, are, or will ever be written. To learn these distributions we need a complex parametric family of distributions $p(\mathbf{x}, w)$ to match to our data. Generative diffusion models (GDMs) are such a class. They are based on the parameterization of a backward diffusion process (Section 6.2) whose introduction requires that we first explain forward diffusion processes (Section 6.1)

6.1 Forward Diffusion Process

A forward diffusion process is defined by the recursive mixing of data with samples from a standard white normal distribution. Consider then data samples $\mathbf{x} = \mathbf{x}_0 \in \mathbb{R}^n$ drawn from a data distribution $q(\mathbf{x}_0) = q(\mathbf{x})$,

$$\mathbf{x} = \mathbf{x}_0 \sim q(\mathbf{x}_0) = q(\mathbf{x}). \tag{24}$$

Introduce now a time index $t = 1, \ldots, T$ and associate samples $\boldsymbol{\epsilon}_t \in \mathbb{R}^n$ from a standard white normal distribution $\mathcal{N}(\boldsymbol{\epsilon}, \mathbf{0}, \mathbf{I})$ with each time index t – the standard white distribution is given by (5) with $\boldsymbol{\mu} = \mathbf{0}$ and $\sigma^2 = 1$.

Further consider a sequence of scalar coefficients $\alpha_t < 1$ and define the sequence

$$\mathbf{x}_t = \left(\sqrt{\alpha_t}\right) \times \mathbf{x}_{t-1} + \left(\sqrt{1-\alpha_t}\right) \times \boldsymbol{\epsilon}_t. \tag{25}$$

When adding the sample $\boldsymbol{\epsilon}_t$ to $\mathbf{x}_{t-1}$ we say that we are adding noise. This is because as we can see in Figure 6, samples from white normal distributions look like noise. Although not required, we choose constants $\alpha_t \approx 1$ in practice. This means that at each step we are adding a small amount of noise.

The idea of the recursion in (25) is to add noise progressively. In the first application of (25) we go from the input data sample $\mathbf{x}_0$, which does not have any noise added, to sample $\mathbf{x}_1$. This sample has some amount of noise. We then proceed to add some more noise to create $\mathbf{x}_2$ and even more noise to create $\mathbf{x}_3$. Observe that since $\alpha_1 < 1$ the noise is becoming more prominent while the input data $\mathbf{x}_0$ is being washed out. The goal is that when we get to time T, the signal $\mathbf{x}_T$ is almost the same as a sample from a white normal distribution. That we end up with pure noise is key to construct the backward process in Section 6.3 and its learned version in Section 6.3.

Each noising step can described as a conditional distribution

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{\alpha_t}\mathbf{x}_{t-1}, (1-\alpha_t)\mathbf{I}) \tag{26}$$

In a nutshell we start from initial image $\mathbf{x}_0$ and add a small amount of noise to it in each step until we reach an image $\mathbf{x}_T$ in the final step that's essentially just pure Gaussian noise (see Figure 6). This is called the forward process.

It is relevant that the properties of the Gaussian distribution lets us sample an $\mathbf{x}_t$ directly from $\mathbf{x}_0$ without having to go through all the intermediate steps as follows:

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t} \cdot \mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t} \cdot \boldsymbol{\epsilon}_0 \tag{27}$$

where $\bar{\alpha} = \prod_{i=1}^{t} \alpha_i$ and $\boldsymbol{\epsilon}_0 \sim \mathcal{N}(\boldsymbol{\epsilon}_0; \mathbf{0}, \mathbf{I})$.

6.2 Backward Diffusion Process

Now comes the hard part: How can we start from a pure noise sample $\mathbf{x}_T$ and iteratively denoise the image until we recover an original sample

$\mathbf{x}_0$? Essentially, our goal is to reverse the Forward process. We call this reversed process the backward process.

Recall that each step in the Forward process is essentially sampling from a Gaussian distribution. Luckily, it turns out that if we knew the original sample $\mathbf{x}_0$ that we started from, then each denoising step in the backward process would also be just sampling from a Gaussian,

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_q(\mathbf{x}_t, \mathbf{x}_0), \sigma_q(t)) \tag{28}$$

This is due to properties of the Gaussian Distribution. The mean of this Gaussian, $\mu_q(\mathbf{x}_t, \mathbf{x}_0)$, is essentially the likeliest $\mathbf{x}_{t-1}$ from which $\mathbf{x}_t$ could have been generated and importantly, it depends on $\mathbf{x}_0$ as well as $\mathbf{x}_t$.

$$\mu_q(\mathbf{x}_t, \mathbf{x}_0) = \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})\mathbf{x}_t + \sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)\mathbf{x}_0}{1-\bar{\alpha}_t} \tag{29}$$

In practice, it is usually easier to work with the added noise $\boldsymbol{\epsilon}_0$ rather than the original image $\mathbf{x}_0$. Thus from (27) we can write $\mathbf{x}_0$ in terms of $\boldsymbol{\epsilon}_0$ and $\mathbf{x}_t$ to get:

$$\mu_q(\mathbf{x}_t, \boldsymbol{\epsilon}_0) = \frac{1}{\sqrt{\alpha_t}}\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}\sqrt{\alpha_t}}\boldsymbol{\epsilon}_0 \tag{30}$$

Recall that our goal in generative modeling was to learn a distribution $p(\mathbf{x}, w)$ that imitates $q(\mathbf{x})$. In diffusion models, we have established so far that to get a sample from $q(\mathbf{x})$, we can start from a pure noise sample $\mathbf{x}_T$, and sample from conditional Gaussian distributions $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_q(\mathbf{x}_t, \boldsymbol{\epsilon}_0), \sigma_q(t))$ at each time step from $t = T, T-1, \cdots, 1$, until we reach $\mathbf{x}_0$. Let's call this the "True Backward process". To imitate this process, we will define our model $p(\mathbf{x}, w)$ in the same way i.e. to sample from $p(\mathbf{x}, w)$, we start from a pure noise sample $\mathbf{x}_T$, and sample from conditional Gaussian distributions $p(\mathbf{x}_{t-1}|\mathbf{x}_t, w)$ at each time step until we reach $\mathbf{x}_0$. We call this the learned backward process and discuss it in the next section.

6.3 Learned Backward Diffusion Process

Our goal is for the distribution of samples generated by the learned Backward process, $p(\mathbf{x}_0, w)$, to match the distribution of samples generated by the "true Backward process", $q(\mathbf{x}_0)$, which is in fact the underlying data distribution. The bad news is that unlike the generative normal distribution in part 3 where we could evaluate $p(\mathbf{x}, w)$ for any given $\mathbf{x}, w$, for a diffusion model, evaluating $p(\mathbf{x}_0, w)$ is intractable (both in theory and practice). However, the good news is that if the intermediate denoising steps of the true and learned backward processes match closely, meaning if the Gaussian distributions $q(\mathbf{x}_{t-1}|\mathbf{x}_0, \mathbf{x}_t)$ and $p(\mathbf{x}_{t-1}|\mathbf{x}_t, w)$ are close for all $t = 1, \cdots T$, then the final sample distributions $q(\mathbf{x}_0)$ and $p(\mathbf{x}_0, w)$ will also be close. Therefore we will choose our loss function in a way to ensure these distributions are close in every step. Thus in a similar manner to (17) but over all time steps we write:

$$w^*_{\text{SRM}} \;=\; \underset{w}{\text{argmin}}\, \mathbb{E}_{\mathbf{x}_0}\left[\sum_{t=2}^{T} \mathbb{E}_{\mathbf{x}_t \sim q(\mathbf{x}_t|\mathbf{x}_0)}\left[D_{\text{KL}}\Big(q(\mathbf{x}_{t-1}|\mathbf{x}_0, \mathbf{x}_t) \| p(\mathbf{x}_{t-1}|\mathbf{x}_t, w)\Big)\right]\right] \tag{31}$$

We will now focus on simplifying the loss in (31). Recall that $q(\mathbf{x}_{t-1}|\mathbf{x}_0, \mathbf{x}_t)$ was a Gaussian distribution with mean $\mu_q(\mathbf{x}_t, \mathbf{x}_0)$ and variance $\sigma_q(t)$. To imitate it as closely as possible, it makes sense to choose $p(\mathbf{x}_{t-1}|\mathbf{x}_t, w)$ to also be a Gaussian with similar mean and variance:

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_q(\mathbf{x}_t, \boldsymbol{\epsilon}_0), \sigma_q(t)) \tag{32}$$

$$p(\mathbf{x}_{t-1}|\mathbf{x}_t, w) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_p(\mathbf{x}_t, w), \sigma_q(t)) \tag{33}$$

Note that the variance has been chosen to be exactly the same. We can do this because $\sigma_q(t)$ only depends on the time step and a fixed noise schedule, both of which we have access to in the learned backward process. With this smart choice of $p(\mathbf{x}_{t-1}|\mathbf{x}_t, w)$, we can reduce the loss in (31) to something more familiar:

$$w^*_{\text{SRM}} \;=\; \underset{w}{\text{argmin}}\, \mathbb{E}_{\mathbf{x}_0}\left[\sum_{t=2}^{T} \mathbb{E}_{\mathbf{x}_t \sim q(\mathbf{x}_t|\mathbf{x}_0)}\left[\| \mu_p(\mathbf{x}_t, w) - \mu_q(\mathbf{x}_t, \boldsymbol{\epsilon}_0)\|^2\right]\right] \tag{34}$$

This should remind us of the loss in (23). Our goal has now been simplified to matching $\mu_q(\mathbf{x}_t, \boldsymbol{\epsilon}_0)$ as closely as possible without knowing $\boldsymbol{\epsilon}_0$.

Again we can choose our model smartly based on our knowledge of $\mu_q(\mathbf{x}_t, \boldsymbol{\epsilon}_0)$.

$$\begin{aligned} \mu_q(\mathbf{x}_t, \boldsymbol{\epsilon}_0) &= \frac{1}{\sqrt{\alpha_t}}\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}\sqrt{\alpha_t}} \cdot \boldsymbol{\epsilon}_0 \\ \mu_p(\mathbf{x}_t, w) &= \frac{1}{\sqrt{\alpha_t}}\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}\sqrt{\alpha_t}} \cdot \hat{\boldsymbol{\epsilon}}(\mathbf{x}_t, t, w) \end{aligned} \tag{35}$$

With this choice of $\mu_p(\mathbf{x}_t, w)$ the loss in (34) turns into:

$$w^*_{\text{SRM}} \simeq \underset{w}{\text{argmin}}\, \mathbb{E}_{\mathbf{x}_0}\left[\sum_{t=2}^{T} \mathbb{E}_{\mathbf{x}_t \sim q(\mathbf{x}_t|\mathbf{x}_0)}\left[\| \hat{\boldsymbol{\epsilon}}(\mathbf{x}_t, t, w) - \boldsymbol{\epsilon}_0\|^2\right]\right] \tag{36}$$

We have ignored the time dependent weighting terms $\frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}\sqrt{\alpha_t}}$ as including them doesn't result in any performance gains. At the end of the day, to train our diffusion model, we have to train a model $\hat{\boldsymbol{\epsilon}}(\mathbf{x}_t, t, w)$ to predict the true noise $\boldsymbol{\epsilon}_0$, given the current noisy sample $\mathbf{x}_t$ and time step t. The ERM version of the SRM problem in (36) is:

$$w^*_{\text{ERM}} = \underset{w}{\text{argmin}} \sum_{i=1}^{N}\sum_{t=2}^{T}\sum_{j=1}^{M} \| \hat{\boldsymbol{\epsilon}}(\mathbf{x}_t^{(i,j)}, t, w) - \boldsymbol{\epsilon}_0^{(i,j)}\|^2 \tag{37}$$

where $\boldsymbol{\epsilon}_0^{(i,j)} \sim \mathcal{N}(\boldsymbol{\epsilon}_0^{(i,j)}; \mathbf{0}, \mathbf{I})$, and $\mathbf{x}_t^{(i,j)} = \sqrt{\bar{\alpha}_t} \cdot \mathbf{x}_0^{(i)} + \sqrt{1-\bar{\alpha}_t} \cdot \boldsymbol{\epsilon}_0^{(i,j)}$.

In practice during training, for a batch of samples, e.g. $\mathbf{x}_0^{(1)}, \cdots, \mathbf{x}_0^{(B)}$, we randomly sample B time steps $t^{(1)}, \cdots t^{(B)}$ uniformly from 2 to T, and also sample B points $\boldsymbol{\epsilon}_0^{(1)}, \cdots, \boldsymbol{\epsilon}_0^{(B)}$ from a standard Gaussian. The batch loss would then be:

$$\sum_{i=1}^{B} \| \hat{\boldsymbol{\epsilon}}(\mathbf{x}_t^{(i)}, t^{(i)}, w) - \boldsymbol{\epsilon}_0^{(i)}\|^2 \tag{38}$$

Depending on the type of data that we are generating, we could choose different models as our noise predictor $\hat{\boldsymbol{\epsilon}}(\mathbf{x}_t, t, w)$. For generating images we will use a U-Net architecture as our noise predictor as discussed in the next section

7 Image Generative Diffusion Models

In the remainder of the this lab, we will train a Diffusion model to learn to generate images of Handwritten digits by training it on the MNIST dataset. We start by loading the dataset.

Task 3 Load the MNIST dataset. Since traning on the entire dataset will take very long, we will be training on a subset of it with $m = 3000$ samples. Your training set should have a roughly equal number of samples from each digit class. You should also normalize each image and resize it to be 32×32 pixels.

We now write a function to implement the Forward process.

Task 4 Write a function that takes a batch of images, and a list of timesteps, as inputs. The output is a batch of noisy versions of the input images according to the given timesteps. (See Equation (27)). For the noise schedule, α_t decreases linearly across timesteps from $\alpha_0 = 0.9999$ to $\alpha_T = 0.98$. Choose the total number of diffusion time steps as $T = 500$. Show an example batch with a single image. Plot the noisy images at time steps $t = 100,\ 200,\ 300,\ 400,\ 500$. For $t = 500$ the image should be indistinguishable from pure noise.

It remains to choose the noise predictor model $\hat{\boldsymbol{\epsilon}}(\mathbf{x}_t, t, w)$. We choose a time-conditional U-Net model for this purpose. A U-Net is a convolutional neural network architecture initially used mainly for image segmentation tasks. It has also proven useful for denoising images which is why we use it here to predict the noise added to images. The U-Net features an encoder-decoder structure. The encoder, or contracting path, captures context through convolutional and pooling layers that downsample the input image. The decoder, or expansive path, reconstructs the image using up-sampling layers, which are combined with high-resolution features from the encoder via skip connections. These skip connections help preserve spatial information. The architecture is symmetrical, resembling a U-shape hence the name. For the purposes of this lab, just note that the U-Net consists of Convolutional layers which we know are a good fit for processing images. The U-Net model that you will be using has been provided in the notebook.

We Also need to implement the backward process, that we can later use to sample new images using our trained model.

Task 5 Write a sampling function that takes a trained U-Net, and a number m as inputs and generates m samples by running the backwards process described in (33).

Finally, we train our noise predictor model using gradient descent.

Task 6 Write a Pytorch training loop implementing Gradient Descent with the batch loss given in (38). Train the model for at least 200 epochs. Plot the training loss as a function of the epoch number. After training, generate 64 images using your trained model and the sampling function you wrote for task 5. Display them in an 8 by 8 grid.

8 Evaluating a Diffusion Model

Now that we have trained a diffusion model, it is time to evaluate it quantitatively. In order to do this we use a metric called the Frechet Inception Distance (FID) score. The FID score essentially measures the distance between the mean and covariance of samples generated by the generative model and the 'ground truth' mean and covariance computed from the data. A smaller FID score (therefore smaller distance) means the model has learned to capture the underlying data distribution more accurately. You can think of the FID as something akin to a test error. It is important to note that the distance is not computed in the pixel space but rather in the feature space of features extracted by a neural network.

Task 7 Compute the FID score of your trained model with respect to a large subset of the MNIST dataset (e.g. 10000 samples) using the cleanFID library. The more generated samples you use to compute the FID, the more accurate the score will be. However, generating samples using your trained model can be slow. Try using at least a few thousand generated samples.

9 Report

Do not take much time to prepare a lab report. We do not want you to report your code and we don't want you to report your work. Just give us answers to questions we ask. Specifically give us the following:

Question	**Report deliverable**
Task 1	Plot of generated samples.
Task 2	Parameter μ^*, Distribution vs Dataset plot
Task 3	Do not report.
Task 4	Noisy Images
Task 5	Do not report
Task 6	Loss Plot, Generated Images
Task 7	FID score

We will check that your answers are correct. If they are not, we will get back to you and ask you to correct them. As long as you submit responses, you get an A for the assignment. It counts for 8 points total.