EECS 182 Deep Neural Networks Fall 2025 Anant Sahai and Gireeja Ranade Discussion 13

1. Denoising diffusion models for generation

In lecture, we talked about denoising diffusion models to get samples from a continuous distribution. This problem is about the potentially simpler binary case. We will assume that we have an unknown distribution of black-and-white images $P(\mathbf{x})$ together with a very large number of example images $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$. Formally, each image can be viewed as a binary vector of length m, i.e. $\mathbf{x} \in \{-1, +1\}^m$.

The first conceptual step in setting up a diffusion model is to choose the easy-to-sample distribution that we want to have at the end of the forward diffusion. For this, we choose m iid fair coin tosses (here we think of a fair coin as having a 50% chance of being +1 and a 50% chance of being -1) arranged into a vector.

Next, we need to choose a way to incrementally degrade the images. Let $\mathbf{Y}_0$ start with whatever image sample $\mathbf{x}$ we want to start with. At diffusion stage t, we generate $\mathbf{Y}_t$ from $\mathbf{Y}_{t-1}$ by randomly flipping each pixel of $\mathbf{Y}_{t-1}$ independently with probability δ where δ is a small positive number.

It turns out that this process of rare pixel-flipping can be reinterpreted for easier analysis. For the j-th pixel at diffusion stage t, this process can alternatively be viewed as first flipping an independent coin $R_t[j]$ with a probability 2δ of coming up $+1$ and then, if $R_t[j] = +1$ replacing $Y_{t-1}[j]$ with a freshly drawn independent fair coin $F_t[j]$ that is equally likely to be -1 or $+1$. If $R_t[j] \neq +1$, we leave that pixel alone i.e. $Y_t[j] = Y_{t-1}[j]$.

(a) We need to verify that if we do this and diffuse for sufficiently many stages T, that the resulting distribution is close to looking like m i.i.d. fair coins. **Show that the probability that pixel j has been replaced at some point by an independent fair coin by time T goes to 1 as $T \to \infty$.**

(HINT: It might be helpful to look at the probability that this has not happened...)

Solution: After T stages, consider the probability that the j-th pixel has not been replaced. This requires the T i.i.d. coin tosses to all have come up tails (or -1 if you prefer) and by independence, the probability of that happening is $(1-2\delta)^T$. Since $\delta > 0$, we know that $|1-2\delta| < 1$ and by stability, this implies that $\lim_{T\to\infty}(1-2\delta)^T = 0$. Since the probability of the complementary event is going to zero, we know that the probability that the pixel *has* been replaced by time T is going to 1.

(b) To efficiently do diffusion training, we need a way to be able to quickly sample a realization of $\mathbf{Y}_t$ starting from $\mathbf{Y}_0 = \mathbf{x}_i$. **Give a procedure to sample a realization of $\mathbf{Y}_t$ given $\mathbf{Y}_0$ without having to generate $\mathbf{Y}_1, \mathbf{Y}_2, \ldots, \mathbf{Y}_{t-1}$.**

(HINT: This probability of whether each pixel is $+1$ or -1 can be obtained by flipping at most two (potentially biased) coins.)

Solution: We know from the previous part that that the probability that j-th pixel has not been replaced by a fair coin by time t is $(1-2\delta)^t$. Consequently, we flip a coin with a probability $(1-2\delta)^t$ of coming up tails. (Note: this can be done by sampling a uniform random variable on $[0,1]$ and seeing if it is greater than $(1-2\delta)^t$ and calling that a head.) If it comes up heads, replace the j-th pixel of $\mathbf{Y}_0$ with a freshly tossed independent fair coin (-1 or +1 equally likely) for $y_t[j]$. Otherwise, have $y_t[j] = y_0[j]$.

(c) For the reverse diffusion process that will be used during image generation and is being learned during training, our goal is to approximate $P(\mathbf{Y}_{t-1}|\mathbf{Y}_t)$ with a neural net that has learnable parameters θ.

Suppose I give you a neural net whose input is a binary image $\mathbf{Y}_t$ and whose output is m real numbers that could each in principle be from $-\infty$ to $+\infty$ (for example, these could be the outputs of a linear layer). **Which of the following nonlinear activation functions would be most appropriate to convert them into a probability that we could use to sample whether the pixel in question should be a $+1$?**

- ◯ Sigmoid $\frac{1}{1+\exp(-x)}$

  Solution: The sigmoid is the right answer since since it always returns a number between 0 to 1 which is required for a valid probability.

- ◯ ReLU $\max(0, x)$

  Solution: This is the wrong answer since it can be greater than 1.

- ◯ Tanh $\tanh(x) = \frac{\exp(2x)-1}{\exp(2x)+1}$

  Solution: This is the wrong answer since it can be negative.

(d) The goal of training is to approximate a probability distribution for random denoising. However, we do not actually have access to $P(\mathbf{Y}_{t-1}|\mathbf{Y}_t)$ and decide to use $P(\mathbf{Y}_{t-1}|\mathbf{Y}_t, \mathbf{Y}_0)$ instead as a proxy. Note: this can be interpreted as finding an appropriate target distribution that corresponds to trying to predict $\mathbf{Y}_0$ itself at this stage.

What is $P(Y_{t-1}[j] = +1|\mathbf{Y}_t = \mathbf{y}, \mathbf{Y}_0 = \mathbf{x})$?

For simplicity, just do this calculation for the case $x[j] = +1$. To further help you save some time, you may use the following helper result that comes from Bayes' Rule. If A and B are both binary random variables where the prior probabilities are $P(A = +1) = \rho$ and $P(A = -1) = 1 - \rho$, with B being bit-flipped from A with independent probability δ — that is, $P(B = +1|A = +1) = 1 - \delta$, $P(B = -1|A = +1) = \delta$, $P(B = +1|A = -1) = \delta$, and $P(B = -1|A = -1) = 1 - \delta$ — then the conditional probabilities for A conditioned on B are given by:

$$P(A = +1|B = +1) = \frac{(1-\delta)\rho}{(1-\rho)\delta + (1-\delta)\rho} \tag{1}$$

$$P(A = -1|B = +1) = \frac{(1-\rho)\delta}{(1-\rho)\delta + (1-\delta)\rho} \tag{2}$$

$$P(A = +1|B = -1) = \frac{\delta\rho}{\rho\delta + (1-\delta)(1-\rho)} \tag{3}$$

$$P(A = -1|B = -1) = \frac{(1-\delta)(1-\rho)}{\rho\delta + (1-\delta)(1-\rho)} \tag{4}$$

(HINT: What is the distribution for $Y_{t-1}[j]$ given $Y_0[j]$?)

Solution: Following the strategy urged by the hint, we let A be $Y_{t-1}[j]$ and B be $Y_t[j]$ so we can use the formulas above. Since we are interested in the conditional probability of $Y_{t-1}[j] = +1$, the relevant formulas we want to invoke are (1) and (3).

Conditioning on $\mathbf{Y}_0 = \mathbf{x}$ just tells us that $x[j] = 1$ and by the underlying Markovianity of the evolution from $\mathbf{Y}_{t-1} \to \mathbf{Y}_t$, we know that $P(Y_t[j]|\mathbf{Y}_{t-1}, \mathbf{Y}_0) = P(Y_t[j]|\mathbf{Y}_{t-1}) = P(Y_t[j]|Y_{t-1}[j])$. This means that we are exactly in the setup for the calculation given above. This means that all we need is the $\rho = P(Y_{t-1}[j] = +1|Y_0[j] = x[j] = +1)$.

Since ρ is the probability of a 1, there are two disjoint ways we could have a 1. The original 1 could have survived or the final fair coin toss that replaced it was itself a 1.

$$\rho = (1-2\delta)^{t-1} + (1-(1-2\delta)^{t-1})\frac{1}{2} \tag{5}$$

$$= \frac{1 + (1 - 2\delta)^{t-1}}{2} \tag{6}$$

and plugging this into (1) and (3) gives us:

$$P(Y_{t-1}[j] = +1 | \mathbf{Y}_t = \mathbf{y}, \mathbf{Y}_0 = \mathbf{x}) = \begin{cases} \frac{(1-\delta)\rho}{(1-\rho)\delta + (1-\delta)\rho} & \text{if } y[j] = +1 \\ \frac{\delta\rho}{\rho\delta + (1-\delta)(1-\rho)} & \text{if } y[j] = -1 \end{cases}.$$

Note that if we had $x[j] = -1$, what would happen is that ρ would be $\frac{1-(1-2\delta)^{t-1}}{2}$.

(e) Let the (conditional) probability distribution (on whether each pixel is $+1$) output by our neural net with nonlinearity be $Q_t(\mathbf{Y}_t)$. For training the denoising diffusion model $q(\mathbf{Y}_{t-1}|\mathbf{Y}_t)$, we choose to use SGD loss $D_{KL}(P(\mathbf{Y}_{t-1}|\mathbf{Y}_t, \mathbf{Y}_0 = \mathbf{x}_i)||Q_t(\mathbf{Y}_t))$ where $\mathbf{x}_i$ is the random training image drawn, t is the random time drawn from 1 to T, and $\mathbf{Y}_t$ is the randomly sampled realization of the forward diffusion at time t starting with the image $\mathbf{x}_i$ at time 0. This ends up being a loss on the vector of probabilities coming out of $Q_t(\mathbf{Y}_t)$ that can be written as a sum over the m entries in the vector of probabilities.

Given what you know about KL Divergence, what does this loss penalize most strongly? What does this loss look like at $t = 1$ in particular?

Solution: The KL divergence penalizes most strongly the probability that the neural net says that something is very unlikely when the proxy target distribution has significant probability. This is due to the Q (from the neural net) being the second term in the KL divergence.

At $t = 1$: The target distribution collapses to a point mass on the original image:

$$P(\mathbf{Y}_0 \mid \mathbf{Y}_1, \mathbf{Y}_0 = \mathbf{x}_i) = \delta(\mathbf{Y}_0 = \mathbf{x}_i),$$

so:

$$\begin{aligned} D_{KL}(P(\mathbf{Y}_0|\mathbf{Y}_0 = \mathbf{x}_i)||Q_1(\mathbf{Y}_1)) &= \log \frac{1}{Q_1(\mathbf{x}_i|\mathbf{Y}_1)} \\ &= -\log Q_1(\mathbf{x}_i \mid \mathbf{Y}_1) = \sum_{j=1}^{m} -\log Q_1(x_i[j] \mid \mathbf{Y}_1), \end{aligned}$$

which is exactly the binary cross-entropy loss.

Connection to label smoothing. For larger t, the target distribution for pixel j

$$P(Y_{t-1}[j] = +1 \mid Y_t[j], Y_0[j])$$

moves continuously from being a *hard label* (close to 0 or 1) toward the *uniform distribution* ($1/2$). In classification, *label-smoothing* exactly replaces hard labels with "soft" labels between 0 and 1. Thus, our objective can be considered an instance of label smoothing, where for early t, the labels are still close to 0 or 1, but at late t, the labels essentially become $1/2$. This is the discrete analogue of how in Gaussian DDPMs the target mean moves toward 0 and the variance approaches 1.

Thus the training objective becomes *binary cross-entropy with time-dependent label smoothing*. Under this interpretation, every stage of the reverse diffusion is very much an image denoiser. However, as we increase the level of the noise, we increasingly smooth the labels for the cross-entropy loss function. This explains why weight-sharing across t is natural: every denoising step is solving the *same task* (denoise a noisy image), just with increasingly smoothed targets.

(f) **After training the model, how do we generate a new sample image from scratch? Describe the full sampling procedure, from initialization, to iterative denoising, to getting the final sample**

Solution: Generation begins from pure noise and repeatedly applies the learned reverse diffusion model.

i. **Initialization (analogous to sampling $x_T \sim \mathcal{N}(0, I)$).** Sample $\mathbf{Y}_T$ as m iid fair coins (each $+1$ or -1 with probability $1/2$).

ii. **Reverse diffusion loop (analogous to $p_\theta(x_{t-1} \mid x_t)$).** For $t = T, T-1, \ldots, 1$:

   i. Input $\mathbf{Y}_t$ and the timestep t into the network.

   ii. The network outputs a probability distribution

$$Q_t(\mathbf{Y}_t)[j] = P_\theta(Y_{t-1}[j] = +1 \mid \mathbf{Y}_t).$$

   iii. For each pixel j, sample

$$Y_{t-1}[j] = \begin{cases} +1 & \text{with probability } Q_t(\mathbf{Y}_t)[j], \\ -1 & \text{otherwise.} \end{cases}$$

   This is the discrete analogue of sampling

$$x_{t-1} \sim p_\theta(x_{t-1} \mid x_t)$$

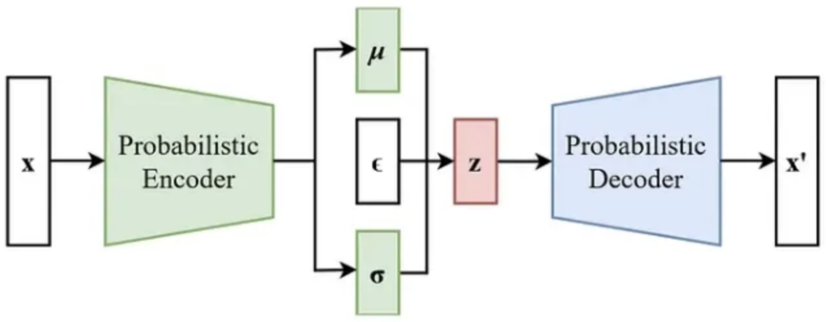
   in a Gaussian DDPM by adding noise.

iii. **Final output.** After reaching $t = 0$, the sample $\mathbf{Y}_0$ is a newly generated binary image drawn from the learned model.

2. Latent Variable Models

(a) Let $q_\phi(z \mid x)$ denote the encoder and $p_\theta(x_i \mid z)$ denote the decoder. **Draw a block diagram for a VAE. In the diagram, show the encoding of input x, sampling of latent z, and decoding of z into reconstruction $\hat{x}$.**

Solution: A possible diagram could be (with $\hat{x}$ instead of x'):



(b) **Describe step-by-step what happens during a forward pass in Variational Autoencoder (VAE) training.**

Solution:

i. **Encode the input.** The encoder network $q_\phi(z \mid x)$ takes x and outputs the parameters

$$\mu_\phi(x), \quad \sigma^2_\phi(x)$$

of a Gaussian distribution approximating the posterior over latent variables.

ii. **Sample a latent vector.** To obtain $z \sim q_\phi(z \mid x)$ in a differentiable way:

$$\epsilon \sim \mathcal{N}(0, I), \qquad z = \mu_\phi(x) + \sigma_\phi(x) \odot \epsilon$$

denoting elementwise product.

iii. **Decode the latent sample.** The decoder $p_\theta(x \mid z)$ outputs parameters of a likelihood distribution over reconstructions. A reconstruction $\hat{x} \sim p_\theta(x \mid z)$ may be sampled from this distribution.

iv. **Compute the VAE loss.**

$$\mathcal{L}_{\text{VAE}}(x) = \underbrace{\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x \mid z)]}_{\text{Reconstruction term}} - \underbrace{D_{\text{KL}}(q_\phi(z \mid x) \| p(z))}_{\text{Regularization term}}$$

where prior $p(z) = \mathcal{N}(0, I)$.

(c) **Describe what the encoder and decoder of the VAE are *respectively* doing** to capture and encode this information into a latent representation of space z. **How is the information bottleneck created in VAE as opposed to a standard Autoencoder.**

Solution:

i. **Encoder** - Encoder maps a high-dimensional input x (like the pixels of an image) and then (most often) outputs the parameters of a Gaussian distribution that specify the hidden variable z. In other words, they output $\mu(x)$ and $\Sigma(x)$ from which $z \sim \mathcal{N}(\mu(x), \Sigma(x))$. We will implement this as a deep neural network, parameterized by ϕ, which computes the probability $q_\phi(z|x)$. We could then sample from this distribution to get noisy values of the representation z.

ii. **Decoder** - Decoder maps the latent representation back to a high dimensional reconstruction, denoted as $\hat{x}$, and outputs the parameters to the probability distribution of the data. We will implement this as another neural network, parametrized by θ, which computes the probability $p_\theta(x|z)$.

The information bottleneck comes from two places. First, the latent space is usually smaller than the input space. Second, there is stochastic sampling from the latent distribution which introduces noise. Due to the pressure from the KL-divergence term in the loss, this latent distribution is encouraged to stay close to the standard normal prior (i.e. have its means driven toward 0 and variance driven toward I, thus constraining how much information about x can be encoded in z).

(d) After training, you notice that when you run a held-out example through the VAE that the predictions were blurry. Please, discuss why that might be.

Solution: The KL term forces each $q_\phi(z \mid x)$ toward $\mathcal{N}(0, I)$, so only the lowest-frequency features survive the information bottleneck. Fine textures and sharp edges get "pruned" to reduce KL cost. Further, because z is drawn stochastically, the decoder must generate reconstructions that work on average over those samples. To minimize expected error, it learns to produce safer, blurrier outputs rather than risk over-fitting to a single sample.

(e) Once the VAE is trained, **how do we use it to generate a new fresh sample from the learned approximation of the data-generating distribution?**

Solution: We can now use only the Decoder network ($p_\theta(x \mid z)$). Here, instead of sampling z from the posterior that we had during training, we sample from our true generative process which is the prior that we had specified ($z \sim \mathcal{N}(0, I)$) and we proceed to use the network to sample $\hat{x}$ from there.

Contributors:

- Anant Sahai.
- Joey Hong.
- Jerome Quenum.
- Ryan Campbell.
- Past CS282 Staff.