DDPMs Course Notes, DL 2024/2025

Mario Prignano

1 DDPMs

DDPMs (Denoising Diffusion Probabilistic Models) `https://arxiv.org/abs/2006.11239` are just a very powerful and specific way to implement a Diffusion Forward and Backward process, tuned for learning deep generative models. In this notes we'll derive the equations that we need to implement DDPMs.

1.1 Forward Process

Since we're gonna train a denoising diffusion model, we need a dataset of noisy images, to then learn a denoising model. Why noise? We choose to slowly map every picture to a pure noise image, because noise is easy to create, once we have a denoising model, we will just create noise, and make our model transform it into an actual image! The forward Process of diffuion can be seen as the process of online (during training) dataset creation.2

Let's start with a clean image from our dataset, say x_0. We want to slowly destroy it by adding noise step-by-step. Imagine taking a picture and throwing a little noise on it. At first, you can still see the picture. But if you keep throwing more noise, eventually it becomes all noise and you won't be able to tellthe difference between that picture and pure noise anymore.

We do this step-by-step: at each step $t = 1, 2, \ldots, T$, we add a bit of Gaussian noise. We call the result x_t, the noisy version of the image at step t.

To make this process smooth, we define a noise schedule $\beta_1, \beta_2, \ldots, \beta_T$, where each β_t is a small number that tells us how much noise we add at step t.

The actual equation to go from x_{t-1} to x_t is:

$$x_t = \sqrt{1 - \beta_t} \cdot x_{t-1} + \sqrt{\beta_t} \cdot \epsilon_t$$

where $\epsilon_t \sim \mathcal{N}(0, I)$ is a random noise vector (the dust we throw). Each x_t becomes noisier and noisier.

It turns out that we can also write the equation for any step t directly in terms of the original image x_0, without going step-by-step:

$$x_t = \sqrt{\bar{\alpha}_t} \cdot x_0 + \sqrt{1 - \bar{\alpha}_t} \cdot \epsilon$$

where $\bar{\alpha}_t = \prod_{s=1}^{t} (1 - \beta_s)$ and $\epsilon \sim \mathcal{N}(0, I)$.

So now, given an image x_0 and a timestep t, we can create the noisy version of it, with t steps layer of noise by just applying this formula, in constant time. You can see this as just interpolating between the actual image and pure noise!

1.2 Backward Process

Now comes the difficult part. We've destroyed the image with noise, but our goal is to learn how to clean it.

If we just invert the last equation we found for the forward process, it seems pretty easy that, if given an image at time step t, x_t, to restore it and bring it back to x_0, we could just reverse the equation, so that:

$$x_0 = \frac{x_t - \sqrt{1-\bar{\alpha}_t} \cdot \epsilon}{\sqrt{\bar{\alpha}_t}}$$

But the problem is that in inference time, we'll start from pure noise just by sampling from a gaussian distribution, and not using the forward equation, so we don't actually know the ε therm of the equation. A solution to this would be to just approximate it, in our case, using a Neural Network. So that the equation now becomes:

$$\hat{x_0} = \frac{x_t - \sqrt{1-\bar{\alpha}_t} \cdot \epsilon_\theta(x_t, t)}{\sqrt{\bar{\alpha}_t}}$$

Where the Neural Network $\epsilon_\theta(x_t, t)$ takes the noisy image x_t and the step t, and guesses the noise ϵ that was added.

The training so is simply:

$$L = \mathbb{E}_{x_0,\epsilon,t}\left[\|\epsilon - \epsilon_\theta(x_t, t)\|^2\right]$$

So the model gets better and better at guessing the TOTAL noise that was added to the image.

1.3 Posterior Derivation

Does all of that mean that we now can just feed the model a totally noisy image and get to a non noisy $\hat{x_0}$ image? Yes and No. Yes because the model we'll try to do its best in order to make this possible, but in practice, it is very hard for the model to do that in a single step.
So why don't we just remove a layer of noise, and then another one, and another one and so on fot T steps For the model would be lot easier, because it basically have a ton of possibilities for correcting itself, for adding details, and its mistakes will weigh less.
So now let's see how to go from a generic x_t to x_{t-1} only knowing a sort of x_0, (the model predicted $\hat{x_0}$)

To do this, we need to understand the posterior distribution:

$$q(x_{t-1} \mid x_t, x_0)$$

This is the probability of getting x_{t-1} if we know x_t and the original image x_0.

It turns out this is still a Gaussian distribution, with mean and variance we can compute using the simple closed formula of the product of two gaussians, if we just use this approximation (valid with big enough T)

$$q(x_{t-1}|x_t, \hat{x_0}) \propto q(x_t|x_{t-1})q(x_{t-1}|\hat{x_0}) \tag{1}$$

Product of two gaussians

$$\mathcal{N}(x;\mu_1,\sigma_1^2) \cdot \mathcal{N}(x;\mu_2,\sigma_2^2) = \mathcal{N}\left(x;\mu,\sigma^2\right) \cdot \underbrace{\mathcal{N}(\mu_1;\mu_2,\sigma_1^2+\sigma_2^2)}_{\text{normalizing constant, we're gonna ignore this one}}$$

$$\text{where:} \quad \sigma^2 = \left(\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}\right)^{-1} \quad \text{and} \quad \mu = \sigma^2\left(\frac{\mu_1}{\sigma_1^2} + \frac{\mu_2}{\sigma_2^2}\right)$$

Back to the Posterior Derivation By inverting the forward equation we get

$$x_t = \sqrt{1-\beta_t} \cdot x_{t-1} + \sqrt{\beta_t} \cdot \epsilon_t$$

we find (recalling $\beta_t = 1 - \alpha_t$)

$$x_{t-1} = \frac{x_t - \sqrt{\beta_t}\,\epsilon_t}{\sqrt{1-\beta_t}}$$

which can be seen a just:

$$\mathcal{N}(\frac{x_t}{\sqrt{\alpha_t}}, \frac{\beta_t}{\alpha_t})$$

so from this part here we get the first part of the (1) that we need: $q(x_t|x_{t-1})$ the other therm, is luckily just the first forward equation:

$$x_t = \sqrt{1-\beta_t} \cdot x_{t-1} + \sqrt{\beta_t} \cdot \epsilon_t$$

but with $t-1$ instead of t:

$$x_{t-1} = \sqrt{1-\beta_{t-1}} \cdot x_{t-1} + \sqrt{\beta_{t-1}} \cdot \epsilon_t$$

which again is just:

$$\mathcal{N}(\bar{\alpha}_{t-1}\hat{x}_0,\ 1-\alpha_{t-1})$$

now by taking the product of this two gaussians using the formula above, we will have our Posterior mean, and Posterior variance:

Final sampling formula: We have:

$$q(x_t \mid x_{t-1}) = \mathcal{N}\left(x_t; \sqrt{\alpha_t}x_{t-1}, (1-\alpha_t)\right) \quad \Rightarrow \quad q(x_{t-1} \mid x_t) = \mathcal{N}\left(x_{t-1}; \frac{x_t}{\sqrt{\alpha_t}}, \frac{\beta_t}{\alpha_t}\right)$$

and:

$$q(x_{t-1} \mid \hat{x}_0) = \mathcal{N}\left(x_{t-1}; \sqrt{\bar{\alpha}_{t-1}}\hat{x}_0, (1-\bar{\alpha}_{t-1})\right)$$

Multiplying these two Gaussians gives:

$$q(x_{t-1} \mid x_t, \hat{x}_0) = \mathcal{N}(x_{t-1}; \mu_t(x_t, \hat{x}_0), \sigma_t^2)$$

where the mean is:

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

and the variance is:

$$\sigma_t^2 = \frac{(1-\bar{\alpha}_{t-1})\beta_t}{1-\bar{\alpha}_t}$$

So in order to get x_{t-1} from x_t we only have to sample from this last distribution

$$\mathcal{N}(\mu_t(x_t, \hat{x}_0), \sigma_t^2)$$

which, again is, in opened form:

$$\mu_t(x_t, \hat{x}_0) + \sigma_t^2 \cdot z$$

where $z \sim \mathcal{N}(0, \mathbf{I})$ So now we only have to create the sampling algorithm to go from a pure noise x_T to a x_0

Algorithm 1 DDPM Sampling Algorithm

```
1: Initialize $x_T \sim \mathcal{N}(0, \mathbf{I})$
2: for $t = T, T-1, \ldots, 1$ do
3:     Predict noise: $\hat{\epsilon}_\theta(x_t, t)$
4:     Estimate $\hat{x}_0$:
           $$\hat{x}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}} \left( x_t - \sqrt{1-\bar{\alpha}_t} \cdot \hat{\epsilon}_\theta(x_t, t) \right)$$
5:     Compute mean:
           $$\mu_t = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\hat{x}_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}x_t$$
6:     Compute variance:
           $$\sigma_t^2 = \frac{(1-\bar{\alpha}_{t-1})\beta_t}{1-\bar{\alpha}_t}$$
7:     if $t > 1$ then
8:         Sample $z \sim \mathcal{N}(0, \mathbf{I})$
9:         $x_{t-1} = \mu_t + \sigma_t z$
10:    else
11:        $x_0 = \mu_1$
12:    end if
13: end for
14: return $x_0$
```

2 Summary

- Forward process = add noise to images $\rightarrow$ with an interpolation
- Backward process = train a neural network (often a UNet) to undo the noise then do it step by step with the derivated posterior.
- After training, we can start from noise and create new images.

It's like learning how to reverse a blender: we mix the image into noise, and then train a model to unmix it!

3 Quickly Looking at Flow matching

An image is just a bunch of numbers which we arrange in a particular way to then be decoded by programs usually in RGB... but again its just a bunch of numbers in a stack of matrices that we can easily think as a big flattened array, namely, a vector. A Vector also can be used to reppreset a point, and with this, our image is just a big dimensional point with a lot of coordinates. So why don't we imagine out image as the position of a particle in a incredibly high dimensional space? You could say its strange, but in reality it gives us the possibilities to use common results in physics.

Imagine to have an ODE, where x is our image, and t is time:

$$\begin{cases} \dot{x}(t) = f(x(t), t) \\ x(t=0) = x_0 \quad \text{our dataset's image} \quad t \in [0,1] \\ x(t=1) = x_1 \sim \mathcal{N}(0, I) \end{cases} \tag{2}$$

This ODE here is the generic trajectory of a particle in physics.

($\dot{x}$ is just another way of saying $\frac{d}{dt}x(t)$) If we force the solution of this ODE for $x(t)$ to be

$$x(t) = x_0 \cdot (1 - t) + x_1 \cdot t$$

See how this is just another time, a simple interpolation between our clean image and total gaussian noise. We can easily find what the f is:

$$\dot{x}(t) = x_1 - x_0 \quad \text{for every t}$$

by simple derivation.
Now we can define this process for all our dataset points, and hope that, by approximating f with a neural network $f_\theta(x(t), t) = v_\theta$. Our Loss can simply be:

$$MSE(x_1 - x_0, v_\theta)$$

But why should we need to approximate f if f is just simply $x_1 - x_0$?
Because now if we just sample a new gaussian noise, we can feed it to the neural network that actually predicts the velocity $\dot{x}$, and in theory we could just go back to x_0:

$$\hat{x_0} \sim x_1 - v_\theta$$

But again, the problem is that the neural Network isn't that powerful, and it fails to predict the velocity field $\dot{x}$ with v_θ So why don't we use the Neural Network N times, and hope that it corrects his errors as we've done with DDPMs Sampling Algorithm? Hope you can see that if we discretize the time interval $[0, 1]$ in N steps, with $\Delta t = \frac{1}{N}$ the equation just become

$$\hat{x}_{t-1} \sim x_t - v_{\theta,t} \cdot \Delta t$$

Now we can just repeat this N time to find a hopefully better approximation of a new, generated x_0 And by doing so, not only we derived the simple Euler Method for solving ODE, but we also made a sampling algorithm for a Flow Matching based generative model. (You can find more on reference [6])

References

[1] Jonathan Ho, Ajay Jain, Pieter Abbeel, *Denoising Diffusion Probabilistic Models*, Advances in Neural Information Processing Systems (NeurIPS), 2020.

[2] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, Surya Ganguli, *Deep Unsupervised Learning using Nonequilibrium Thermodynamics*, International Conference on Machine Learning (ICML), 2015.

[3] Alexander Quinn Nichol, Prafulla Dhariwal, *Improved Denoising Diffusion Probabilistic Models*, International Conference on Machine Learning (ICML), 2021.

[4] Yang Song, Stefano Ermon, *Score-Based Generative Modeling through Stochastic Differential Equations*, International Conference on Learning Representations (ICLR), 2021.

[5] Diederik P. Kingma, Max Welling, *Auto-Encoding Variational Bayes*, arXiv preprint arXiv:1312.6114, 2013.

[6] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, Matt Le, *Flow Matching for Generative Modeling*, arXiv preprint arXiv:2210.02747, 2022.