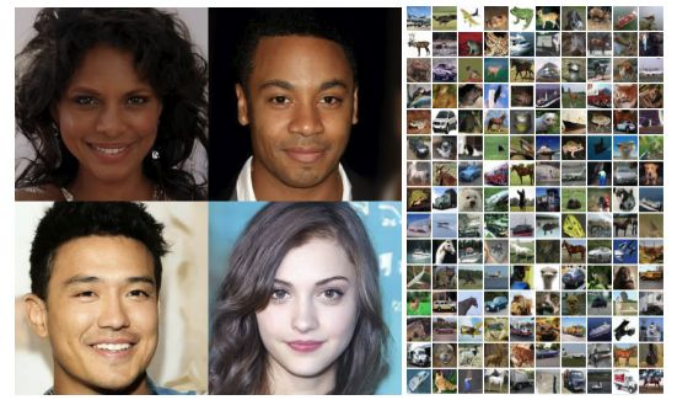
Diffusion Models

Vijay Raghavan

Denoising diffusion probabilistic models

- Imagine you have a photo that is completely noisy, and step by step, this noise is reduced until you get a clear, detailed image.
- The model does something similar:
  - it starts with randomness and gradually refines it into a detailed image, guided by learned patterns from training data.
  - This approach has been found to produce very high-quality images, competing with or even surpassing other methods.
- The paper also presents a simpler way to train these models and a method to make the generation process more efficient, making it easier to understand and use these models for creating images.



Ho, J., Jain, A., & Abbeel, P. (2020). Denoising diffusion probabilistic models. Advances in neural information processing systems, 33, 6840-6851.

Diffusion Models

- These are probabilistic models that define a nonlinear mapping from latent variables to the observed data where both quantities have the same dimension.
- Like variational autoencoders, they approximate the data likelihood using a lower bound based on an encoder that maps to the latent variable.
- However, in diffusion models, this encoder is predetermined; the goal is to learn a decoder that is the inverse of this process and can be used to produce samples.
- Diffusion models are easy to train and can produce very high-quality samples that exceed the realism of those produced by GANs.

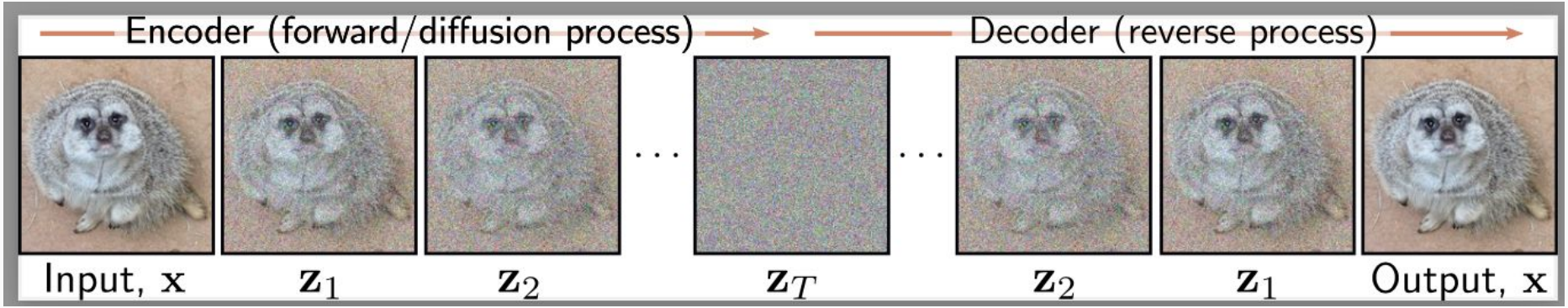
Prince, S. J. (2023). Understanding Deep Learning. MIT press.

Details

- The encoder takes a data sample $\boldsymbol{x}$ and maps it through a series of intermediate latent variables $\boldsymbol{z_1 \ldots z_T}$.
- The decoder reverses this process; it starts with $\boldsymbol{z_T}$ and maps back through $\boldsymbol{z_{T-1}}, \ldots, \boldsymbol{z_1}$ until it finally (re-)creates a data point $\boldsymbol{x}$.
- In both encoder and decoder, the mappings are stochastic rather than deterministic.
- The encoder is prespecified; it gradually blends the input with samples of white noise .
    - With enough steps, the conditional distribution $\boldsymbol{q(z_T|x)}$ and marginal distribution $\boldsymbol{q(z_T)}$ of the final latent variable both become the standard normal distribution.
    - Since this process is prespecified, all the learned parameters are in the decoder.
- In the decoder, a series of networks are trained to map backward between each adjacent pair of latent variables $\boldsymbol{z_T}$ and $\boldsymbol{z_{T-1}}$.
    - The loss function encourages each network to invert the corresponding encoder step.
    - The result is that noise is gradually removed from the representation until a realistic-looking data example remains.
    - To generate a new data example $\boldsymbol{x}$, we draw a sample from $\boldsymbol{q(z_T)}$ and pass it through the decoder.

Prince, S. J. (2023). Understanding Deep Learning. MIT press.

What does look like?



- The encoder (forward, or diffusion process) maps the input x through a series of latent variables $\boldsymbol{z}_1 \ldots \boldsymbol{z}_T$.
- This process is pre- specified and gradually mixes the data with noise until only noise remains.
- The decoder (reverse process) is learned and passes the data back through the latent variables, removing noise at each stage.
- After training, new examples are generated by sampling noise vectors $\boldsymbol{z}_T$ and passing them through the decoder.

Prince, S. J. (2023). Understanding Deep Learning. MIT press.

Diffusion or forward process

The diffusion or forward process maps a data example x through a series of intermediate variables $z_1 \ldots z_T$ with the same size as x according to:

$$z_1 = \sqrt{1-\beta_1} \cdot x + \sqrt{\beta 1} \cdot \varepsilon_1$$

$$z_T = \sqrt{1-\beta_t} \cdot z_{t-1} + \sqrt{\beta t} \cdot \varepsilon_t \qquad \forall\, t \in \{2,\ldots,T,\}, \text{ where } \varepsilon_t \text{ is noise drawn from a standard normal distribution}$$

The first term attenuates the data plus any noise added so far, and the second adds more noise.

The hyperparameters $\beta_t \in [0, 1]$ determine how quickly the noise is blended and are collectively known as the noise schedule. The forward process can equivalently be written as:

$$q(z_1|x) = Norm_{z_1}\left[\sqrt{1-\beta_1}\, x\,,\, \beta 1\, I\right]$$

$$q(z_t|z_{t-1}) = Norm_{z_t}\left[\sqrt{1-\beta_t}\, z_{t-1}\,,\, \beta t\, I\right] \qquad \forall\, t \in \{2,\ldots,T\}.$$

This is a Markov chain because the probability z_t depends only on the value of the immediately preceding variable z_{t-1}. With sufficient steps T, all traces of the original data are removed, and $q(z_T|x) = q(z_T)$ becomes a standard normal distribution

The joint distribution of all of the latent variables $z_1 \ldots z_T$ given input x is:

$$q(\mathbf{z}_{1\ldots T}|\mathbf{x}) = q(\mathbf{z}_1|\mathbf{x}) \prod_{t=2}^{T} q(\mathbf{z}_t|\mathbf{z}_{t-1})$$

Prince, S. J. (2023). Understanding Deep Learning. MIT press.

Intermediate variables and compute

- The diffusion model works by gradually corrupting the original data ***x*** over multiple steps.
- In each step, we add some random noise to get a new corrupted variable.
    - For example:

$$\begin{aligned}\text{Step 1: } & z_1 = x + \text{some noise} \\ \text{Step 2: } & z_2 = z_1 + \text{more noise} \\ \text{Step 3: } & z_3 = z_2 + \text{even more noise}\end{aligned}$$

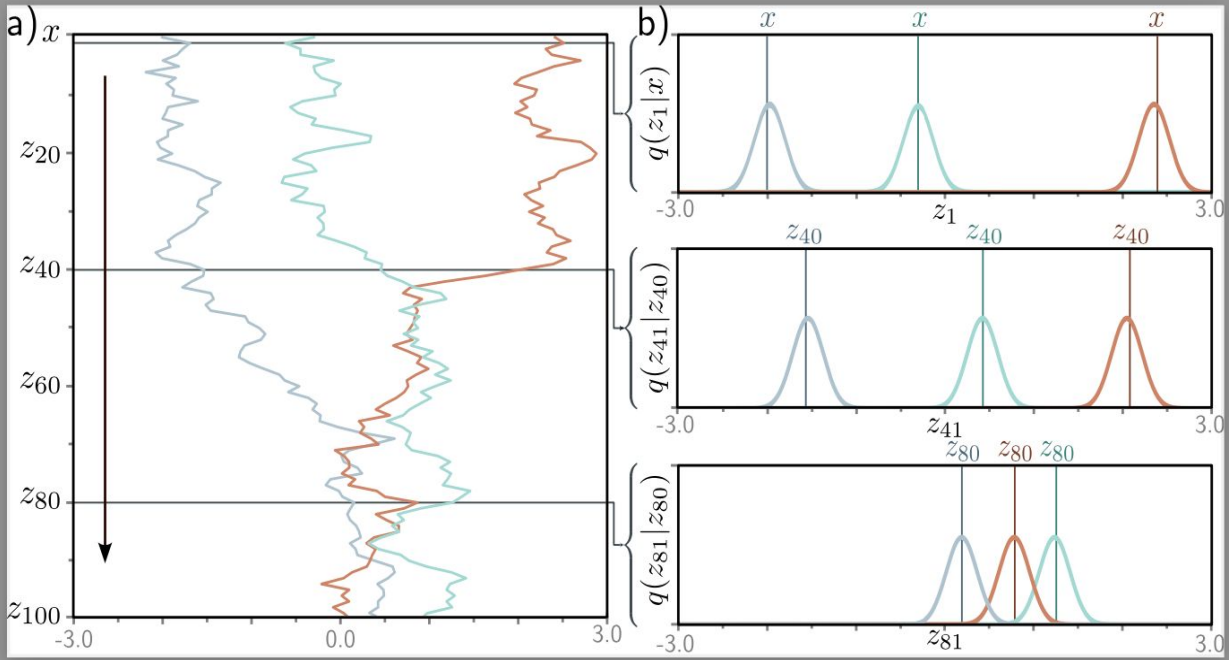
      and so on, for many steps.
- The problem is that sampling all these intermediate variables $z_1, z_2, z_3, z_t \ldots$ from scratch every time is slow.
- We would like a shortcut to directly get a corrupted z_t without having to generate all the intermediate steps.

Prince, S. J. (2023). Understanding Deep Learning. MIT press.

Forward Process

a) We consider one-dimensional data x with T = 100 latent variables $z_1, \ldots, z_{100}$ and $\beta = 0.03$ at all steps.
   - Three values of x (gray, cyan, and orange) are initialized (top row). These are propagated through $z_1, \ldots, z_{100}$.
   - At each step, the variable is updated by attenuating its value by $\sqrt{1-\beta}$ and adding noise with mean zero and variance β.
   - Accordingly, the three examples noisily propagate through the variables with a tendency to move toward zero.

b) The conditional probabilities $Pr(z_1|x)$ and $Pr(z_t|z_{t-1})$ are normal distributions with a mean that is slightly closer to zero than the current point and a fixed variance β_t



Prince, S. J. (2023). Understanding Deep Learning. MIT press.

Diffusion Kernel

- There is a mathematical trick that lets us jump directly to sampling z_t
- We derive a special probability distribution called the "diffusion kernel":

$$q(z_t \mid x) = \mathcal{N}(\sqrt{\alpha_t}x, (1-\alpha_t)I)$$

Where:

$$\alpha_t = \prod_{s=1}^{t}(1-\beta_s)$$

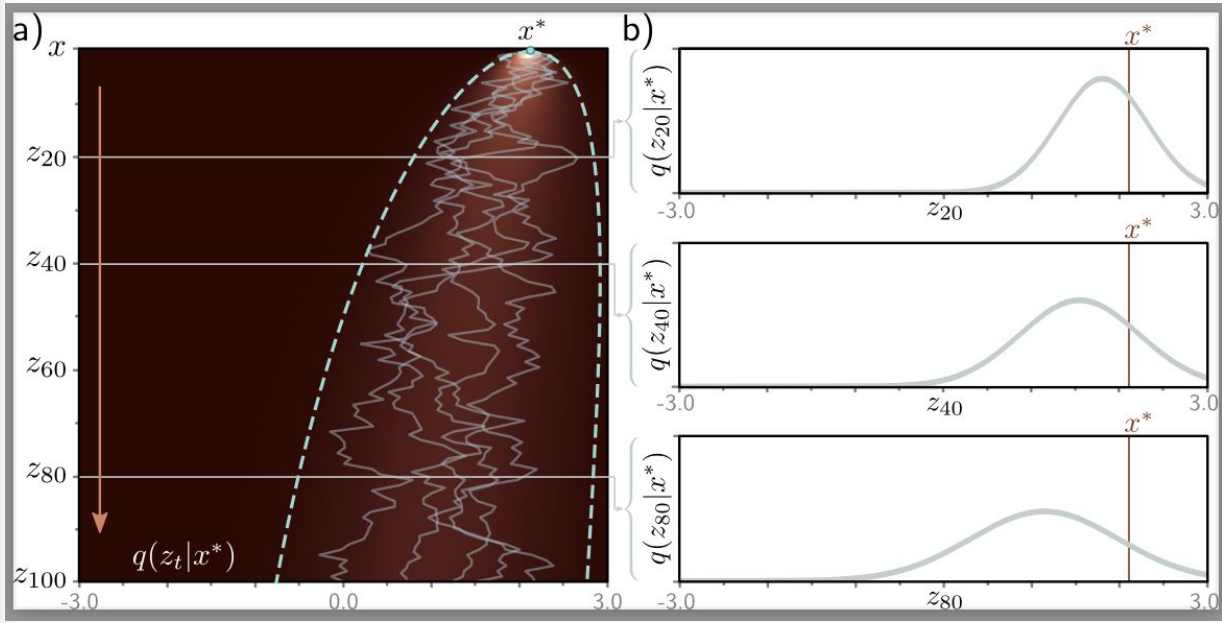
- So to sample a corrupted z_t, we just sample from this Gaussian distribution
- $\beta_t \in [0, 1]$ determine how quickly the noise is blended and are collectively known as the noise schedule.
- The mean starts off as the original *x*, but gets gradually attenuated over more diffusion steps.
    - And the variance starts off low but gets bigger over more steps as we add more noise.
- In simple terms, the diffusion kernel is a handy mathematical shortcut that lets us sample the corrupted z_t in one step, without having to do all the intermediate steps.

Prince, S. J. (2023). Understanding Deep Learning. MIT press.

Diffusion Kernel

a) The point $x^* = 2.0$ is propagated through the latent variables (five paths shown in gray).
   i) The diffusion kernel $q(z_t|x^*)$ is the probability distribution over variable z_t given that we started from x^*.
   ii) It is a normal distribution whose mean moves toward zero and whose variance increases as t increases.
   iii) Heatmap shows $q(z_t|x^*)$ for each variable.
   iv) Cyan lines show ±2 standard deviations from the mean.

b) The diffusion kernel $q(z_t|x^*)$ is shown explicitly for $t = 20, 40, 80$.
   i) In practice, the diffusion kernel allows us to sample a latent variable z_t corresponding to a given x^* without computing the intermediate variables $z_1,\ldots,z_{t-1}$.
   ii) When t becomes very large, the diffusion kernel becomes a standard normal.



Prince, S. J. (2023). Understanding Deep Learning. MIT press.

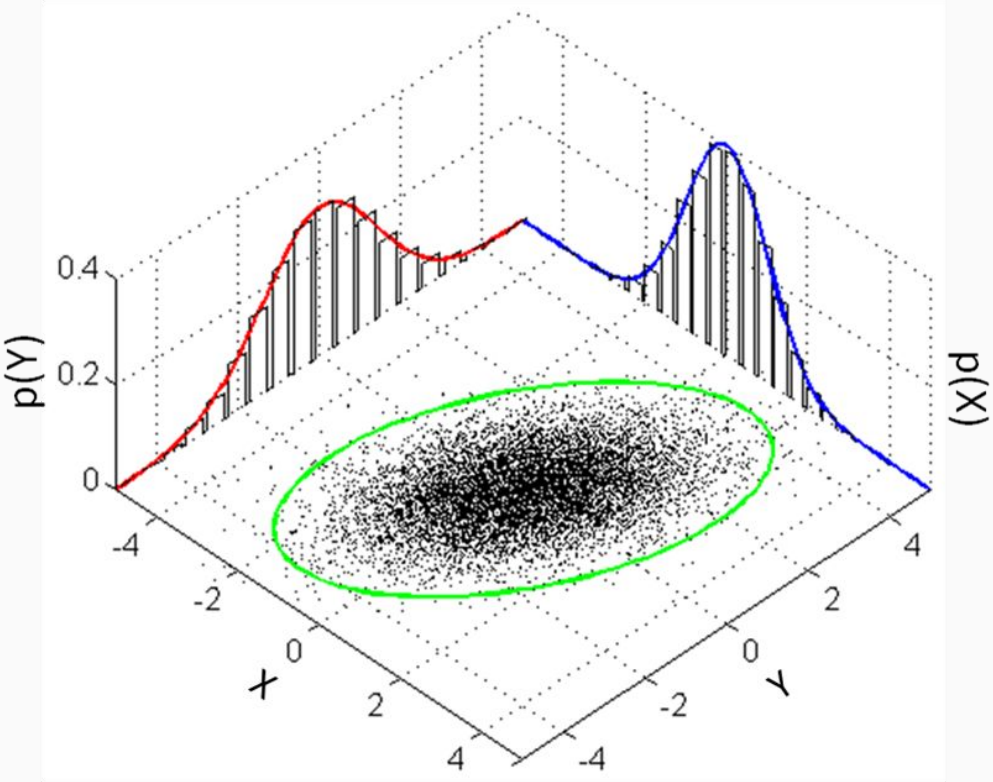
Marginal distribution $q(z_t)$

- The marginal distribution $q(z_t)$ is the probability of observing a value of z_t given the distribution of possible starting points x and the possible diffusion paths for each starting
- For example, imagine we start with a bunch of nice photos.
  - We make copies of each photo.
  - Then we randomly mess up those photocopies by scribbling on them with a black marker.
  - The more we scribble, the more messed up and unrecognizable the photo gets.
  - If we scribble the same amount on every photo copy, then after some amount of scribbling, we'll have:
    - The original nice photos
    - The scribbled messy copies of those photos
- $q(z_t)$ answers the question: *"What is the overall distribution of scribbled messy photos, after scribbling them for time t?"*

Prince, S. J. (2023). Understanding Deep Learning. MIT press.

Key properties of $q(z_t)$

- Mixes together all the messy scribbled photos
- Tells us generally what the messy photos look like
- We can estimate $q(z_t)$ by:
  - Taking lots of nice photos
  - Messing up each photo by scribbling for time *t*
  - Mixing together all those messy t-scribbled photos

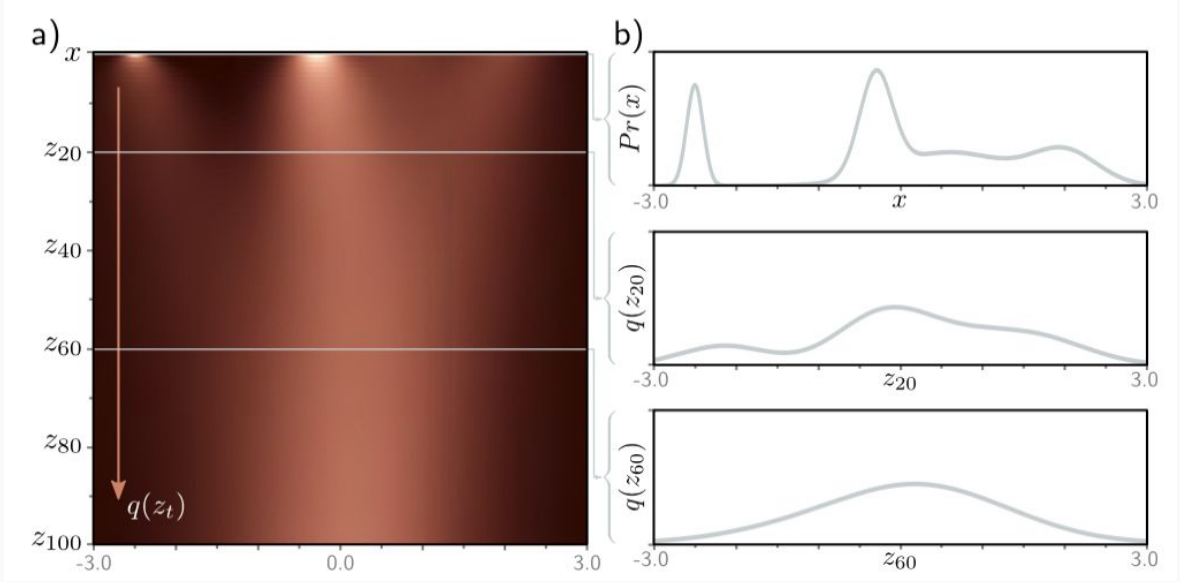


- Many samples from a bivariate normal distribution.
- The marginal distributions are shown in red and blue.
- The marginal distribution of *X* is also approximated by creating a histogram of the *X* coordinates without consideration of the *Y* coordinates.

Marginal distributions

a) Given an initial density *Pr(x)* (top row), the diffusion process gradually blurs the distribution as it passes through the latent variables z_t and moves it toward a standard normal distribution.
   i) Each subsequent horizontal line of heatmap represents a marginal distribution $q(z_t)$.

b) The top graph shows the initial distribution *Pr(x)*.
   i) The other two graphs show the marginal distributions $q(z_{20})$ and $q(z_{60})$, respectively.



Prince, S. J. (2023). Understanding Deep Learning. MIT press.

Conditional distribution $q(z_{t-1} \mid z_t)$

- We defined the conditional probability $q(z_t|z_{t-1})$ as the mixing process .

$$
\begin{aligned}
q(\mathbf{z}_1|\mathbf{x}) &= \text{Norm}_{\mathbf{z}_1}\left[\sqrt{1-\beta_1}\mathbf{x}, \beta_1\mathbf{I}\right] \\
q(\mathbf{z}_t|\mathbf{z}_{t-1}) &= \text{Norm}_{\mathbf{z}_t}\left[\sqrt{1-\beta_t}\mathbf{z}_{t-1}, \beta_t\mathbf{I}\right] \qquad \forall\, t \in \{2, \ldots, T\}.
\end{aligned}
$$

- To reverse this process, we apply Bayes' rule: $q(z_{t-1}|z_t) = [q(z_t|z_{t-1})q(z_{t-1})]/\, q(z_t)$
- This is intractable since we cannot compute the marginal distribution $q(z_{t-1})$ as their form is complex.
- However, in many cases, they are well-approximated by a normal distribution.
- This is important because when we build the decoder, we will approximate the reverse process using a normal distribution.

Conditional Distribution

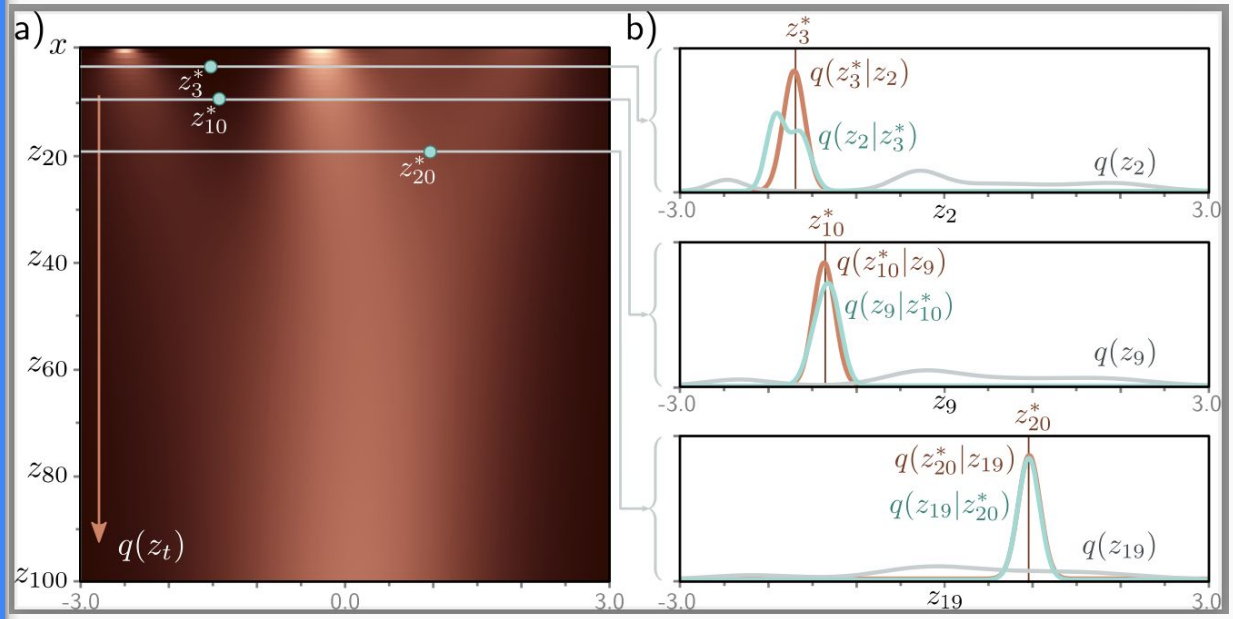
a) The marginal densities $q(z_t)$ with three points z_t^* highlighted.

b) The probability $q(z_{t-1}|z_t^*)$ (cyan curves) is computed via Bayes' rule and is proportional to $q(z_t^*|z_{t-1})q(z_{t-1})$.

   i) In general, it is not normally distributed (top graph), although often the normal is a good approximation (bottom two graphs).

   ii) The first likelihood term $q(z_t^*|z_{t-1})$ is normal in z_{t-1} with a mean that is slightly further from zero than z_t^* (brown curves).

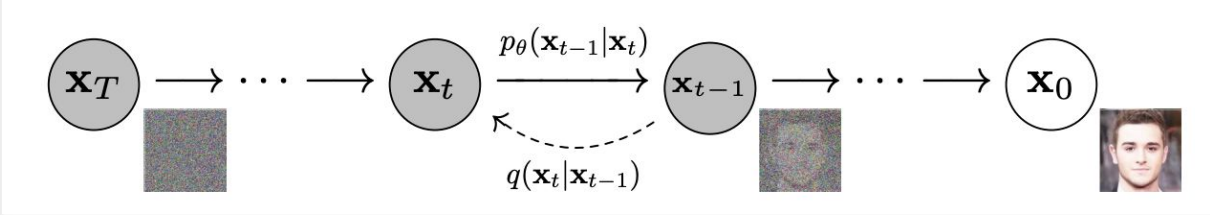
   iii) The second term is the marginal density $q(z_{t-1})$ (gray curves).

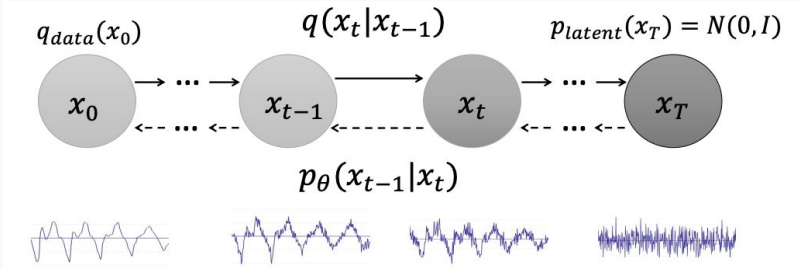


Prince, S. J. (2023). Understanding Deep Learning. MIT press.

Conditional diffusion distribution $q(z_{t-1}|z_t, x)$

- We have been talking about how a data point x diffuses over time into z_t.
- Now we want to go backwards - if we have a diffused point z_t, can we guess what the original x was?
- This reverse diffusion distribution is written $q(z_{t-1}|z_t, x)$.
- In words:
- "Given the current diffused point z_t and the original data x, what was the diffusion variable z_{t-1} at the previous timestep?"





Lu, Y. J., Tsao, Y., & Watanabe, S. (2021, December). A study on speech enhancement based on diffusion probabilistic model. In *2021 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)* (pp. 659-666). IEEE.

Ho, J., Jain, A., & Abbeel, P. (2020). Denoising diffusion probabilistic models. *Advances in neural information processing systems*, *33*, 6840-6851.

Conditional diffusion distribution $q(z_{t-1}|z_t, x)$

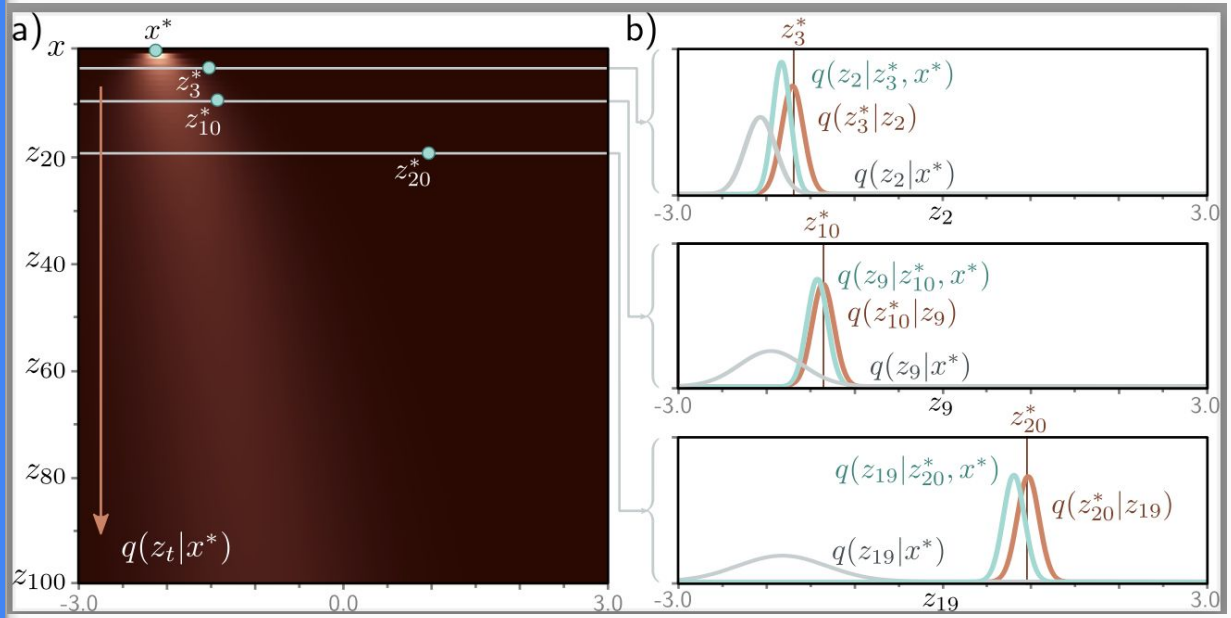
a) Diffusion kernel for x^*= −2.1 with three points z_t^* highlighted.

b) The probability $q(z_{t-1}| z_t^*, x^*)$ is computed via Bayes' rule and is proportional to $q(z_t^*|z_{t-1})q(z_{t-1}|x^*)$.

i) This is normally distributed and can be computed in closed form.

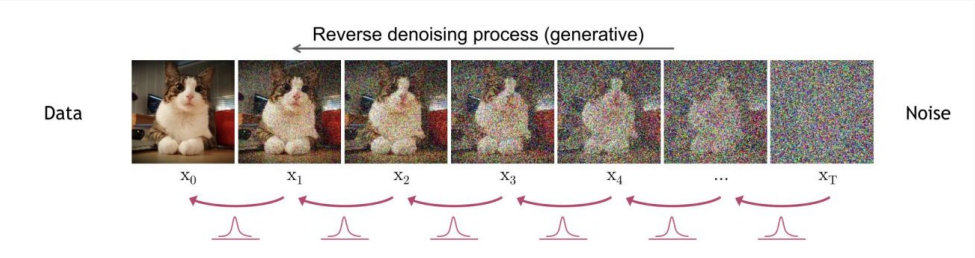
ii) The first likelihood term $q(z_t^*|z_{t-1})$ is normal in z_{t-1} with a mean that is slightly further from zero than z_{t^*} (brown curves).

iii) The second term is the diffusion kernel $q(z_{t-1}|x^*)$ (gray curves).



Prince, S. J. (2023). Understanding Deep Learning. MIT press.

Reverse diffusion distribution



- We couldn't calculate $q(z_{t-1}|z_t)$ before because we didn't know the original x
- However, if we add *x* as a condition, we can derive a formula for this reverse diffusion
- It tells us the distribution over what z_{t-1} could have been to end up at z_t
- The formula turns out to be:
  - $q(z_{t-1}|z_t, x)$= Gaussian over z_{t-1}
  - Mean = weighted combination of z_t and x
  - Variance = gets smaller as we diffuse backwards
- This reverse diffusion distribution lets us probabilistically walk back the diffusion process if we know where it ended up at z_t. The mean interpolates between the end point z_t and original point x.

https://www.cs.rice.edu/~vo9/cv-seminar/slides/aman-diffusion.pdf

Diffusion process - going from blurred points back to clear points

- The reverse mappings are complex, depending on all possible x's and diffusion paths. We approximate each mapping as a Gaussian distribution with:
  - Mean = neural network applied to z_T
  - Variance = pre-defined based on diffusion
- Each step of the process tries to map the current blurred z to what it might have been one step less blurred.
- To generate clear samples:
  - Sample random blurred point z_T
  - Map $z_T \rightarrow z_{T-1}$
  - Map $z_{T-1} \rightarrow z_{T-2} \dots z_T$.
  - Map $z_1 \rightarrow x$. Where x is the original clear data.
  - Going step-by-step reverse until we output a clear x.
- To summarize:
  - we learn a series of neural networks to reverse the blurring, approximated as Gaussians.
  - We then generate clear samples by walking back step-by-step.

Prince, S. J. (2023). Understanding Deep Learning. MIT press.

Example: We start with clear cat photos.

- We randomly blur these cat photos more and more over multiple steps.
- The decoder learns to unblur photos in 10 steps: Each step is a neural network: Input: Blurred photo -> Output: Slightly less blurred photo
    - Step 1: Take very blurred photo, unblur a little
    - Step 2: Take less blurred photo, unblur a little more ……
    - Step 10: Take slightly blurred photo, unblur back to clear cat
- To generate clear cats:

  1. Start with very blurred random photo
  2. Apply Step 10 network: slight unblur
  3. Apply Step 9 network: more unblur….
  10. Apply Step 1: clear photo!
- In simple terms:
    - the decoder learns a series of baby steps to reverse the blurring process.
    - Each neural network undoes a little blurring.
    - When sampling, we start blurred and undo the blurring step-by-step.

Prince, S. J. (2023). Understanding Deep Learning. MIT press.

Decoder model (reverse process)

- We learn a series of probabilistic mappings back from latent variable z_T to z_{T-1}, from z_{T-1} to z_{T-2}, and so on, until we reach the data x.
- The true reverse distributions $q(z_{t-1}|z_t)$ of the diffusion process are complex distributions that depend on the data distribution $Pr(x)$
- We approximate these as normal distributions:

$$\begin{aligned} Pr(\mathbf{z}_T) &= \text{Norm}_{\mathbf{z}_T}[\mathbf{0}, \mathbf{I}] \\ Pr(\mathbf{z}_{t-1}|\mathbf{z}_t, \boldsymbol{\phi}_t) &= \text{Norm}_{\mathbf{z}_{t-1}}\Big[\mathbf{f}_t[\mathbf{z}_t, \boldsymbol{\phi}_t], \sigma_t^2\mathbf{I}\Big] \\ Pr(\mathbf{x}|\mathbf{z}_1, \boldsymbol{\phi}_1) &= \text{Norm}_{\mathbf{x}}\Big[\mathbf{f}_1[\mathbf{z}_1, \boldsymbol{\phi}_1], \sigma_1^2\mathbf{I}\Big], \end{aligned}$$

  - $F_t[z_t, \varphi_t]$ is a neural network that computes the mean of the normal distribution in the estimated mapping from z_t to the preceding latent variable z_{t-1}.
  - The terms $\{\sigma_t^2\}$ are predetermined. If the hyperparameters β_t in the diffusion process are close to zero (and the number of time steps T is large), then this normal approximation will be reasonable.
  - We generate new examples from $Pr(x)$ using ancestral sampling. We start by drawing z_T from $Pr(z_T)$.
  - Then we sample z_{T-1} from $Pr(z_{T-1}|z_T, \varphi_T)$, sample z_{T-2} from $Pr(z_{T-2}|z_{T-1}, \varphi_{T-1})$ and so on until we finally generate x from $Pr(x|z_1, \varphi_1)$.

Prince, S. J. (2023). Understanding Deep Learning. MIT press.

Training

- The goal is to maximize the likelihood of the training data x under our learned decoder model. This tells us how probable the real data is according to our diffusion-reversal process.
- The likelihood involves a marginalization over all possible latent variables $z_1 ... z_T$ for each x:
  - $P(x)$ = Integral over $z_1 ... z_T$ of $P(x, z_1 ... z_T)$
  - This sums up the joint likelihood $P(x, z_1 ... z_T)$ over all sequences that could have diffused into x.
- This integral is intractable to compute directly. So instead, we use a common trick:
  - We lower bound the likelihood using Jensen's inequality
  - This converts the likelihood to an expectation we can estimate
  - We then maximize this lower bound, akin to variational autoencoders
- We optimize the decoder parameters step-by-step over batches:
  - Sample a batch of $z_t \sim q(z_t|x)$ by diffusing the training data
  - Sample $z_t \sim q(z_{t-1}|z_t, x)$ using the reverse process
  - Update the decoder for step t to maximize the likelihood lower bound
  - Repeating this process over all steps and batches trains the full diffusion reversing model.

Prince, S. J. (2023). Understanding Deep Learning. MIT press.

Example: Imagine we have just one cat photo called "Catty"

- We make two copies of Catty. Let's call them "Catty10" and "Catty9".
- We start scribbling black marker all over Catty10 for 10 seconds. It's a mess!
- We scribble marker over Catty9 for only 9 seconds. So it's less messy.
- Now the training starts:
    a. Input the messy 10-second scribbled Catty10 into Net10
    b. Net10 tries its best to clean up Catty10, outputting its guess of what the 9-second scribbled version would look like
    c. Compare Net10's output to the real 9-second scribbled Catty9
    d. Calculate error, update Net10's weights so its output would look more like Catty9
    e. Repeat steps 1-4 many times, each time Net10 gets better at converting a 10-second scribbled cat into a 9-second scribbled cat
- Then we train the Net9 to convert 9-second scribbled cats into 8-second scribbled cats. And so on, until the last network outputs nice and clean again.

Prince, S. J. (2023). Understanding Deep Learning. MIT press.

Loss Function

- The loss has two main terms:

    1. Reconstruction term: This measures how well the model is able to reconstruct the original data point x_i from the latent variable z at the last diffusion step. This encourages the model outputs to match the real data.

    2. Diffusion term: This measures how well the predicted latent variable z at each step matches the target distribution - which is a Gaussian centered at a linear combination of the next timestep's z and the data. This encourages the model to reverse the diffusion process.

- The hyperparameters α and β control the strength of the diffusion process. As t increases, more noise has been added, so α decreases and beta β.
- The loss is summed over all data points and all diffusion steps to train the entire model/diffusion process end-to-end.
- By optimizing this loss, we learn a model that can reverse the artificial diffusion back to the real data distribution.

Prince, S. J. (2023). Understanding Deep Learning. MIT press.

Evidence Lower Bound (ELBO)

- To fit the model, we maximize the ELBO with respect to the parameters $\varphi_{1...T}$.
- We recast this as a minimization by multiplying with minus one and approximating the expectations with samples to give the loss function:

$$L[\boldsymbol{\phi}_{1...T}] = \sum_{i=1}^{I}\Bigg(\overbrace{-\log\Big[\text{Norm}_{\mathbf{x}_i}\left[\mathbf{f}_1[\mathbf{z}_{i1},\boldsymbol{\phi}_1],\sigma_1^2\mathbf{I}\right]\Big]}^{\text{reconstruction term}} + \sum_{t=2}^{T}\frac{1}{2\sigma_t^2}\Bigg\|\underbrace{\frac{1-\alpha_{t-1}}{1-\alpha_t}\sqrt{1-\beta_t}\mathbf{z}_{it}+\frac{\sqrt{\alpha_{t-1}}\beta_t}{1-\alpha_t}\mathbf{x}_i}_{\text{target, mean of } q(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{x})} - \underbrace{\mathbf{f}_t[\mathbf{z}_{it},\boldsymbol{\phi}_t]}_{\text{predicted } \mathbf{z}_{t-1}}\Bigg\|^2\Bigg), \tag{18.29}$$

where $\mathbf{x}_i$ is the i^{th} data point, and $\mathbf{z}_{it}$ is the associated latent variable at diffusion step t.

Prince, S. J. (2023). Understanding Deep Learning. MIT press.

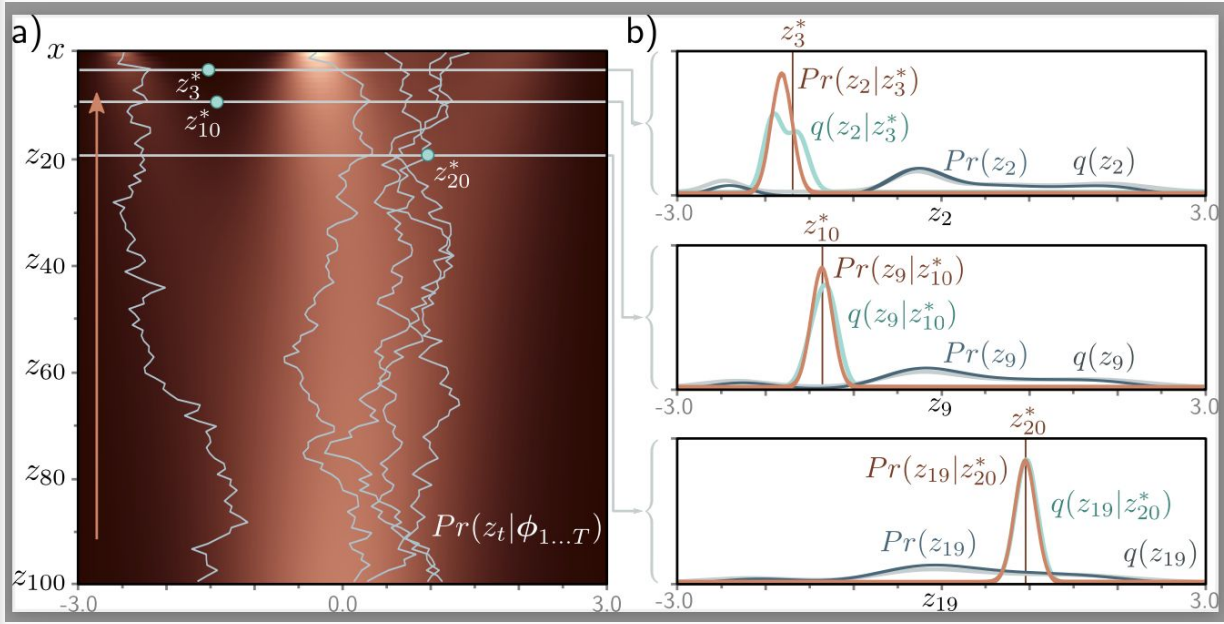
Fitted Model

a) Individual samples can be generated by sampling from the standard normal distribution $Pr(z_T)$ (bottom row) and then sampling z_{T-1} from $Pr(z_{T-1}|z_T) = Norm_{z_{T-1}}[f_T[z_T, \phi_T], \sigma_T^2 I]$ and so on until we reach x (five paths shown).

   i) The estimated marginal densities (heatmap) are the aggregation of these samples and are similar to the true marginal densities .

b) The estimated distribution $Pr(z_{t-1}|z_t)$ (brown curve) is a reasonable approximation to the true posterior of the diffusion model $q(z_{t-1}|z_t)$(cyan curve) .

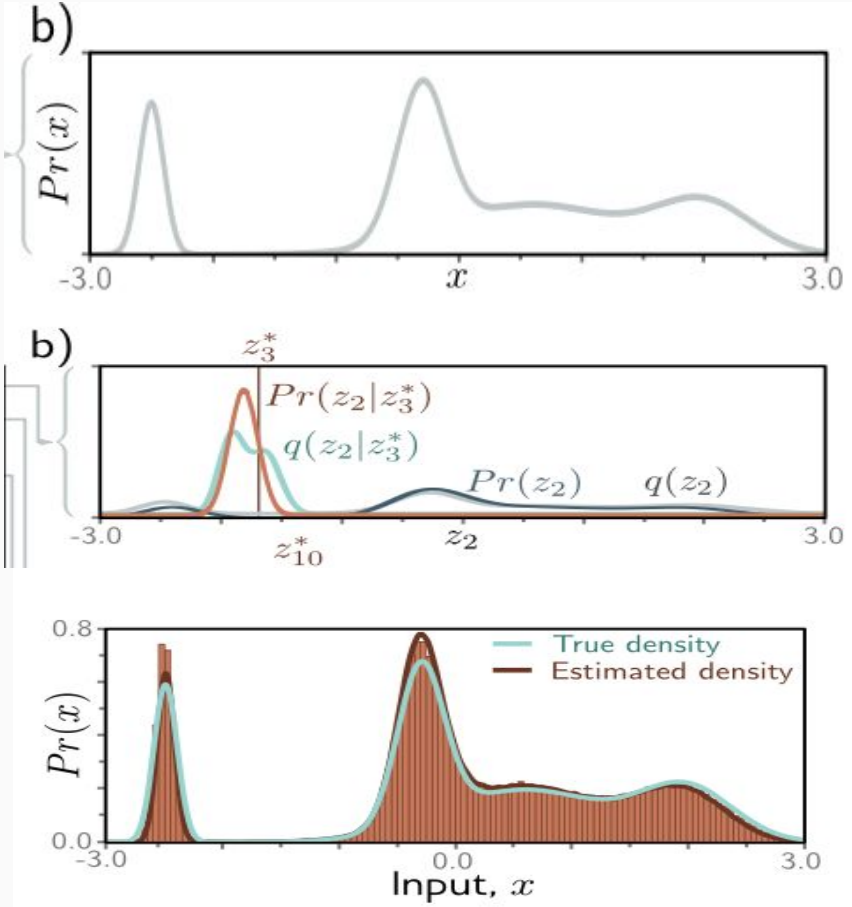
   i) The marginal distributions $Pr(z_t)$ and $q(z_t)$ of the estimated and true models (dark blue and gray curves, respectively) are also similar.



Prince, S. J. (2023). Understanding Deep Learning. MIT press.

Fitted model results

- Cyan and brown curves are original and estimated densities and correspond to the top rows of figures .
- Vertical bars are binned samples from the model, generated by sampling from $Pr(z_T)$ and propagating back through the variables $z_{T-1}, z_{T-2}, \ldots$



Prince, S. J. (2023). Understanding Deep Learning. MIT press.

Reparameterized Loss Function

1. Originally, the diffusion model aimed to predict the next diffused image $\tilde{z}_{t-1}$ from the current diffused image z_t.
2. The target for the model is reparameterized to predict the noise ε that was added to the original image x to create the current diffused image z_t.
3. This means the model now predicts $\tilde{\varepsilon} = g_t(z_t, \phi_t)$ rather than $\tilde{z}_{t-1} = f_t(z_t, \phi_t)$.
4. The loss function is modified to compare the predicted noise $\tilde{\varepsilon}$ to the true previously added noise ε.
5. This reparameterization simplifies the loss function and is found empirically to work better for training diffusion models.
6. The key advantage is that predicting the added noise is an easier learning task than predicting the next diffused image.

In summary, reparameterization changes the model to predict the noise rather than the diffused image, leading to a simpler and more effective loss function.

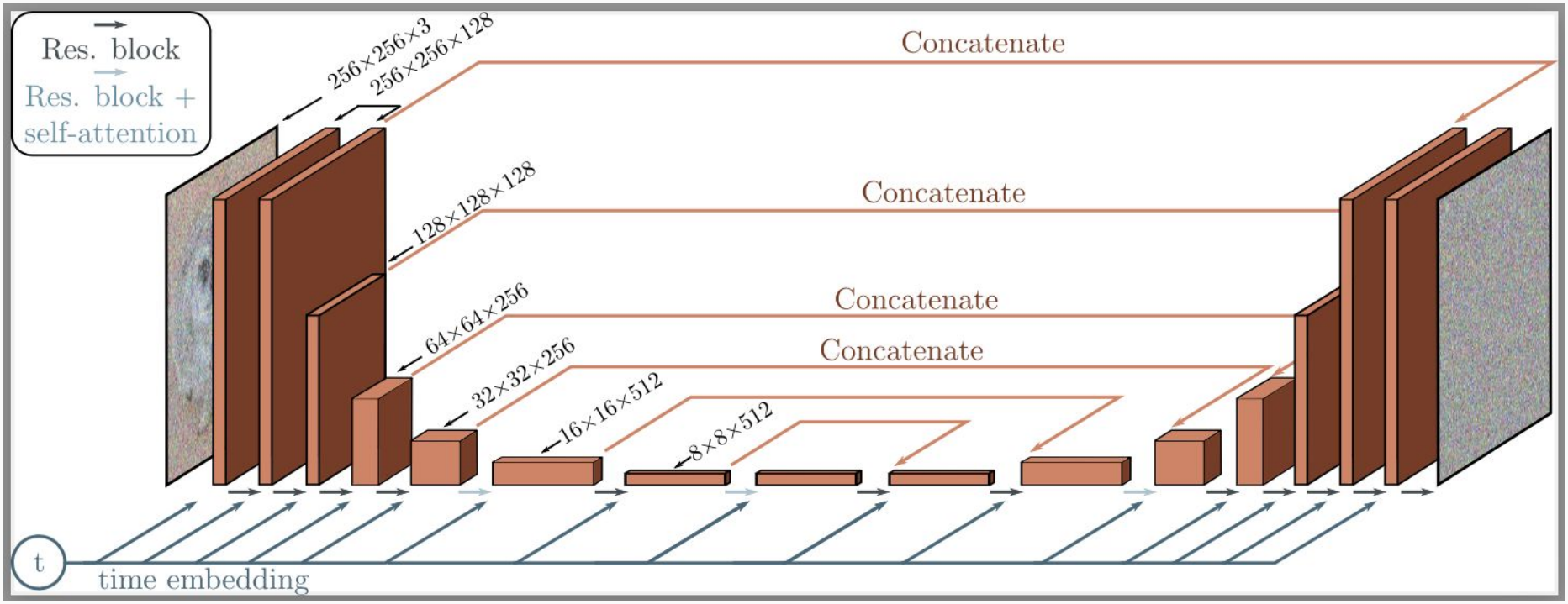
Prince, S. J. (2023). Understanding Deep Learning. MIT press.

U-Net as used in diffusion models for images

- The network aims to predict the noise that was added to the image.
- It consists of an encoder which reduces the scale and increases the number of channels and a decoder which increases the scale and reduces the number of channels.
- The encoder representations are concatenated to their partner in the decoder.
- Connections between adjacent representations consist of residual blocks, and periodic global self-attention in which every spatial position interacts with every other spatial position.
- A single network is used for all time steps, by passing a sinusoidal time embedding through a shallow neural network and adding the result to the channels at every spatial position at every stage of the U-Net.

Prince, S. J. (2023). Understanding Deep Learning. MIT press.

Diffusion UNet



Prince, S. J. (2023). Understanding Deep Learning. MIT press.

Two main approaches for conditional generation

1. Classifier Guidance - Adds an extra term to the denoising process that depends on the gradient of a classifier that predicts the class c from the latent z. This encourages the denoising to make the target class more likely.

2. Classifier-Free Guidance - Incorporates the class information directly into the main diffusion model by adding an embedding for c to the U-Net layers. Can be trained jointly on conditional and unconditional data. Allows continuous trade off between quality and diversity via weighting the class embedding.

- The key idea is to incorporate the class information c into the iterative denoising process to encourage generation of examples from the target class.
- Using a classifier provides an explicit signal, while embedding in the U-Net layers is more direct.
- Both approaches allow controlled conditional generation.

Prince, S. J. (2023). Understanding Deep Learning. MIT press.

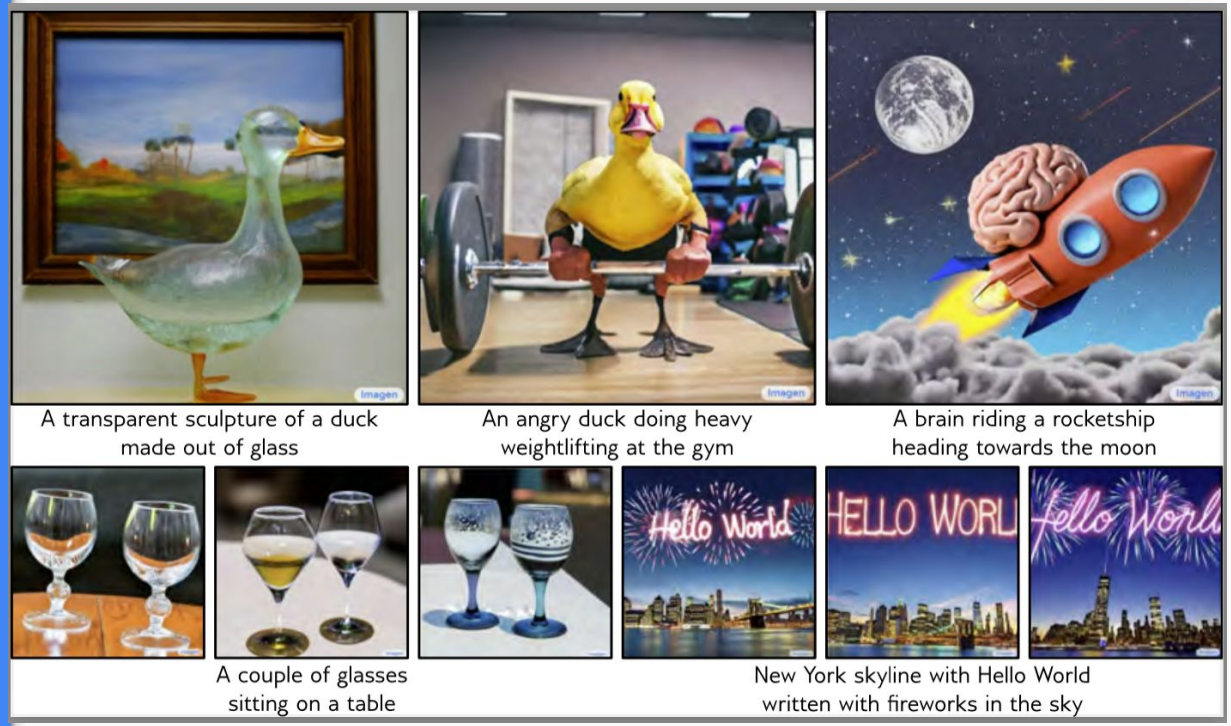
Conditional generation using classifier guidance

- Image samples conditioned on different ImageNet classes.
- The same model produces high quality samples of highly varied image classes.
- Adapted from Dhariwal & Nichol (2021)

Prince, S. J. (2023). Understanding Deep Learning. MIT press.

Conditional generation using text prompts

- Synthesized images from a cascaded generation framework, conditioned on a text prompt encoded by a large language model.
- The stochastic model can produce many different images compatible with the prompt.
- The model can count objects and incorporate text into images.
- Adapted from Saharia et al. (2022b).



Prince, S. J. (2023). Understanding Deep Learning. MIT press.

Diffusion Cascade

Cascaded conditional generation based on a text prompt.

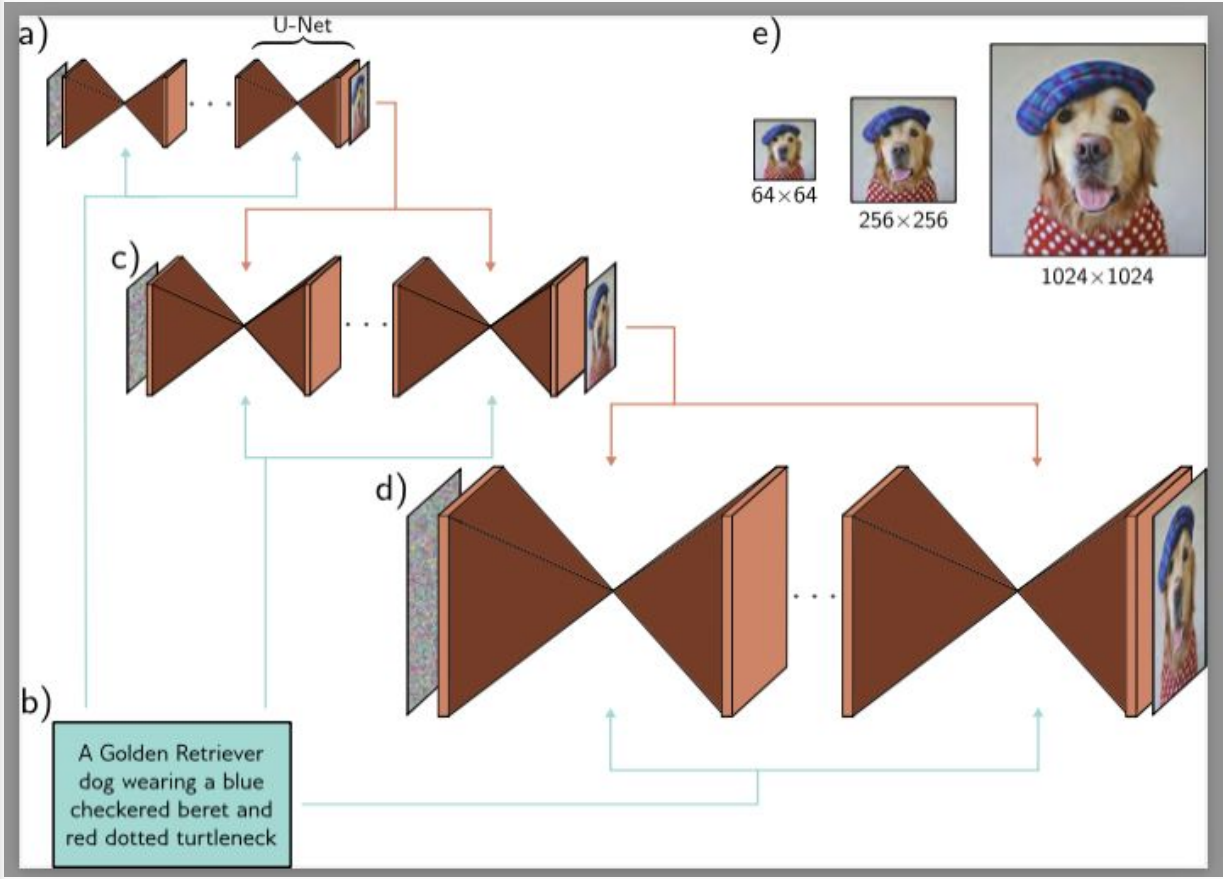
a) A diffusion model consisting of a series of U-Nets is used to generate a 64×64 image.

b) This generation is conditioned on a sentence embedding computed by a language model.

c) A higher resolution 256×256 image is generated and conditioned on the smaller image and the text encoding.

d) This is repeated to create a 1024×1024 image.

e) Final image sequence. Adapted from Saharia et al. (2022b).

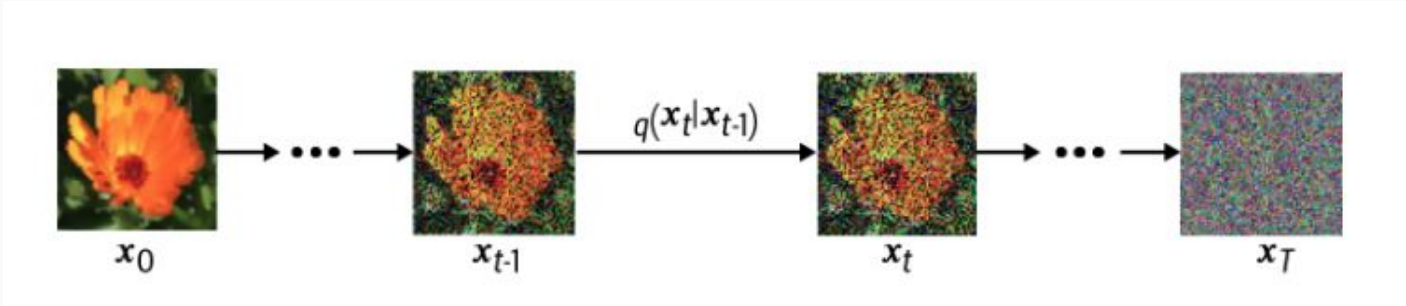


Prince, S. J. (2023). Understanding Deep Learning. MIT press.

Implementation

Forward Diffusion Process

- The goal is to take an image x_0 and gradually corrupt it over many steps until it looks like completely random Gaussian noise.
- This corrupted image can then be used as the starting point for a reverse diffusion process to reconstruct the original image.
- The forward diffusion gradually corrupts the image by adding small controlled amounts of noise, while keeping the total variance fixed.
- After many steps, we get Gaussian noise which can be reversed.



Steps

1. Start with image x_0 that we want to corrupt. Normalize it to have 0 mean and unit variance.
2. Repeatedly apply a function f that adds a small amount of Gaussian noise to the image and scales it.
3. Specifically, if the current image is x_{t-1}, we generate the next image x_t as:
   a. $x_t = \sqrt{(1-\beta)}x_{t-1} + \sqrt{\beta}\,\varepsilon_{t-1}$
   b. where $\varepsilon_{t-1} \sim N(0, I)$ is a vector of standard Gaussian noise and β is a small constant that controls the noise level.
4. By inductively applying the variance formula $Var(aX + bY) = a^2Var(X) + b^2Var(Y)$, it can be shown that if x_{t-1} has unit variance, then x_t will also have unit variance after this update.
5. Therefore, if we apply this diffusion process for many steps T, the final x_T will be a vector with 0 mean and unit variance - i.e. a standard Gaussian as desired.
6. The forward noising process q can also be written as follows:

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}\left(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}\right)$$

The Reparameterization Trick

If we define $\alpha_t = 1 - \beta_t$ and $\overline{\alpha}_t = \prod_{i=1}^{t} \alpha_i$, then we can write the following:

$$\begin{aligned} \mathbf{x}_t &= \sqrt{\alpha_t}\mathbf{x}_{t-1} + \sqrt{1-\alpha_t}\epsilon_{t-1} \\ &= \sqrt{\alpha_t\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\epsilon \\ &= \cdots \\ &= \sqrt{\overline{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\overline{\alpha}_t}\epsilon \end{aligned}$$

- We can add two gaussians to get a new Gaussian
- This helps us to go from x_0 to x_t and we can define a diffusion schedule using α_t instead of β_t
- the interpretation that $\overline{\alpha}_t$ is the variance due to the signal (the original image, $\mathbf{x}_0$) and 1- $\overline{\alpha}_t$ is the variance due to the noise
- The forward diffusion process q can therefore also be written as follows:

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_t; \sqrt{\overline{\alpha}_t}\mathbf{x}_0, (1-\overline{\alpha}_t)\mathbf{I}\right)$$

Diffusion Schedules

- In the original paper (Ho et al., 2020), the authors chose a linear diffusion schedule for β_t and it linearly increases with t from 0.0001 to 0.02. 0.02.
- This ensures that in the early stages of the noising process we take smaller noising steps than in the later stages, when the image is already very noisy
- In a later paper it was found that a cosine diffusion schedule outperformed the linear schedule from the original paper.
- A cosine schedule defines the following values of $\overline{\alpha}_t$:

$$\overline{\alpha}_t = \cos^2\left(\frac{t}{T} \cdot \frac{\pi}{2}\right)$$

The updated equation is therefore as follows (using the trigonometric identity $\cos^2(x) + \sin^2(x) = 1$):

$$\mathbf{x}_t = \cos\left(\frac{t}{T} \cdot \frac{\pi}{2}\right)\mathbf{x}_0 + \sin\left(\frac{t}{T} \cdot \frac{\pi}{2}\right)\epsilon$$

Signal and Noise

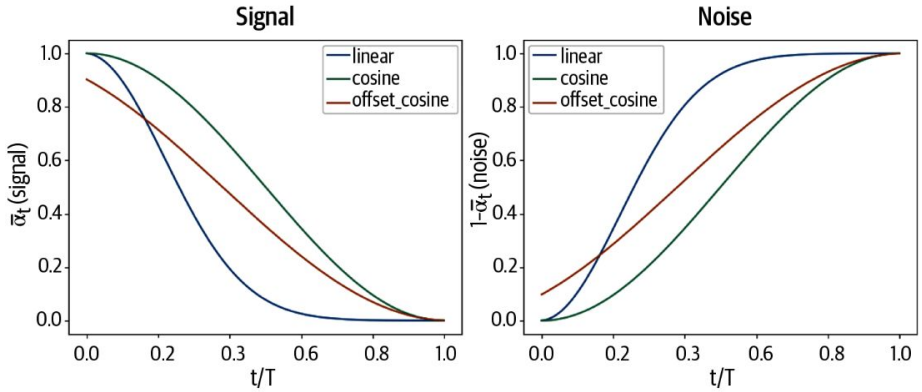


Figure 8-4. The signal and noise at each step of the noising process, for the linear, cosine, and offset cosine diffusion schedules

Figure 8-5. An image being corrupted by the linear (top) and cosine (bottom) diffusion schedules, at equally spaced values of t from 0 to T (source: Ho et al., 2020)

The Reverse Diffusion Process

- We are looking to build a neural network $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ that can undo the noising process i.e., approximate the reverse distribution $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$
- If we can do this, we can sample random noise from $\mathcal{N}(0,\mathbf{I})$and then apply the reverse diffusion process multiple times in order to generate a novel image

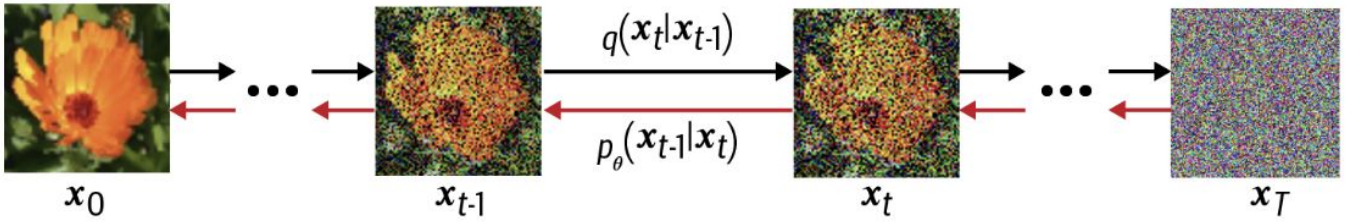


Figure 8-6. The reverse diffusion process $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ tries to undo the noise produced by the forward diffusion process

Training

Algorithm 1 Training

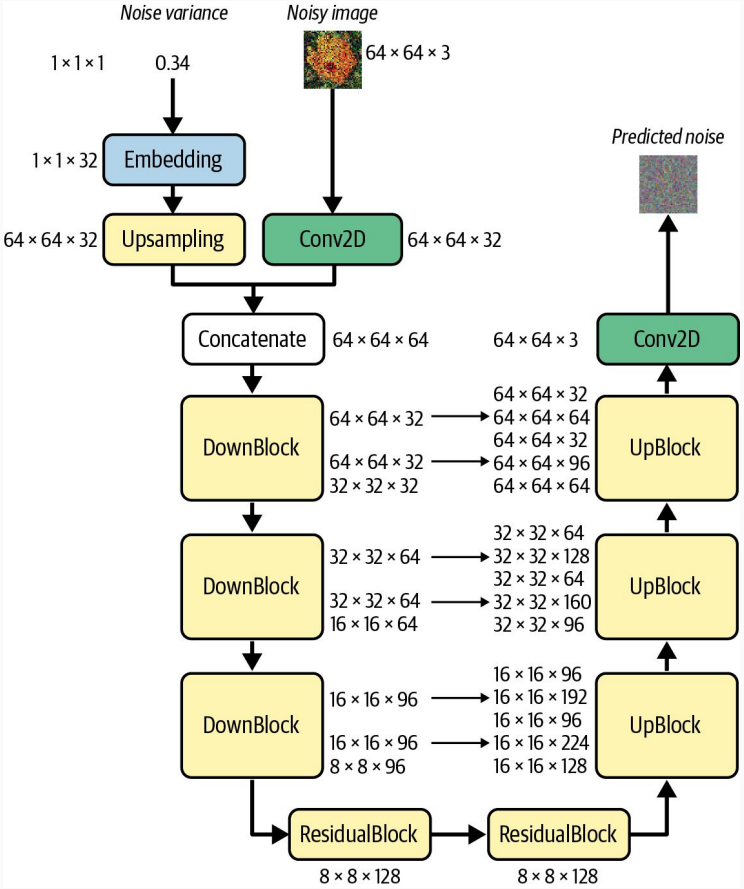
1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on

$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$

6: **until** converged

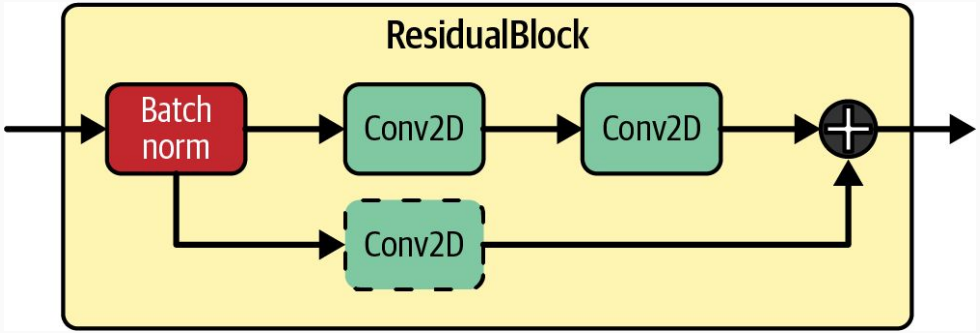
U-Net

- U-Net consists of two halves:
  - the downsampling half, where input images are compressed spatially but expanded channel-wise
  - and the upsampling half, where representations are expanded spatially while the number of channels is reduced.
- There are skip connections in the upsampling and downsampling parts of the network.
- The skip connections allow information to shortcut parts of the network and flow through to later layers.
- A U-Net is particularly useful when we want the output to have the same shape as the input.
- We want to predict the noise added to an image, which has exactly the same shape as the image itself



Residual Block

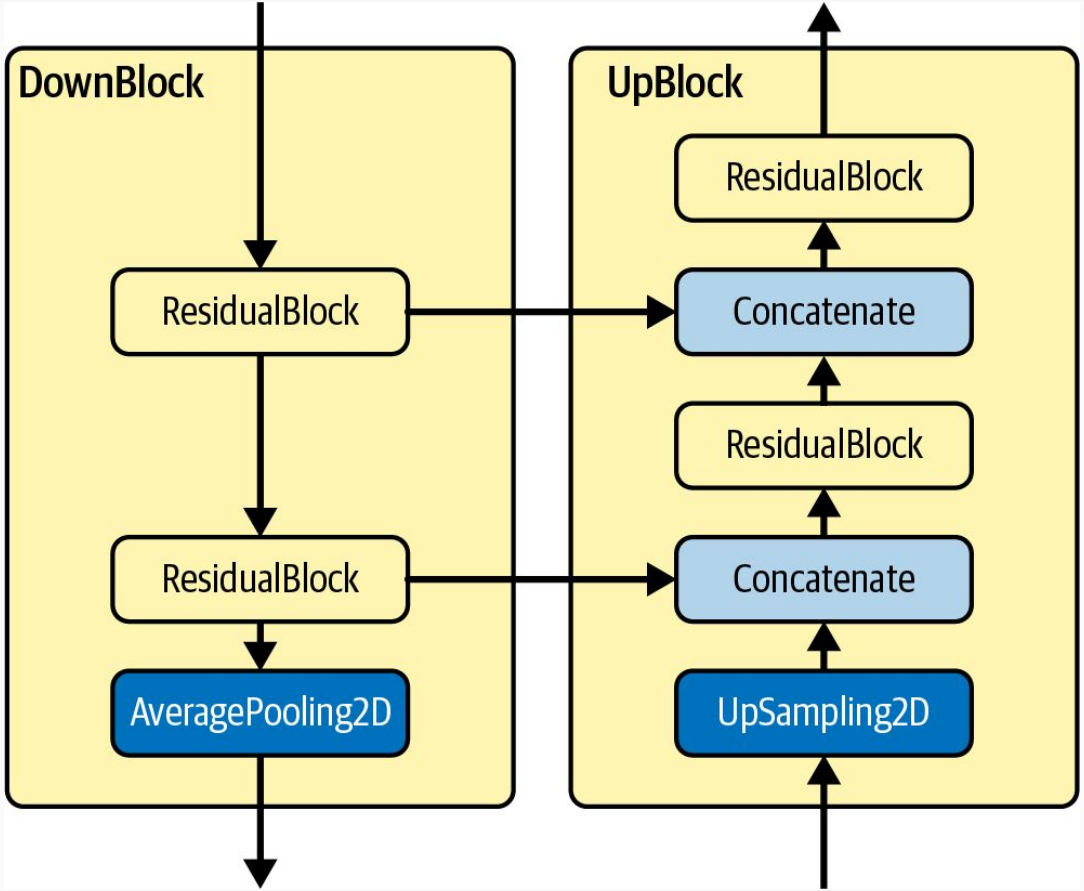
- A group of layers that contains a skip connection that adds the input to the output.
- Residual blocks help us to build deeper networks that can learn more complex patterns without suffering as greatly from vanishing gradient and degradation problems.
- The vanishing gradient problem is the assertion that as the network gets deeper, the gradient propagated through deeper layers is tiny and therefore learning is very slow.
- The degradation problem - accuracy seems to become saturated at a certain depth and then degrade rapidly.



DownBlocks and UpBlocks

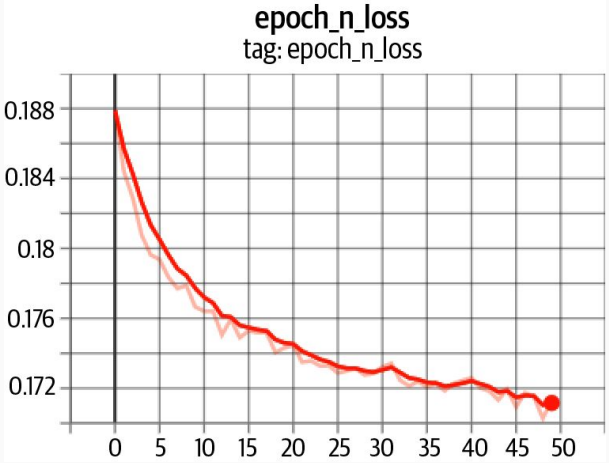
- DownBlock increases the number of channels via block_depth (=2 in our example) ResidualBlocks, while also applying a final AveragePooling2D layer in order to halve the size of the image.
- Each ResidualBlock is added to a list for use later by the UpBlock layers as skip connections across the U-Net.
- An UpBlock first applies an UpSampling2D layer that doubles the size of the image, through bilinear interpolation.
- Each successive UpBlock decreases the number of channels via block_depth (=2) ResidualBlocks, while also concatenating the outputs from the DownBlocks through skip connections across the U-Net.

The DownBlock and corresponding UpBlock in the U-Net



Training

- Instantiate the model.
- Compile the model, using the AdamW optimizer (similar to Adam but with weight decay, which helps stabilize the training process) and mean absolute error loss function.
- Calculate the normalization statistics using the training set.
- Fit the model over 50 epochs.



Sampling from the Denoising Diffusion Model

- The goal is to start with random noise and gradually convert that into a realistic image by undoing the noise over multiple small steps.
- This allows the model to gradually transform the noise into recognizable images.
- The key aspect is that by undoing the noise gradually over multiple steps, the model is able to adjust its predictions and transform random noise into realistic images.
- The rates allow blending noise and signal at each step to get the model outputs to converge properly.

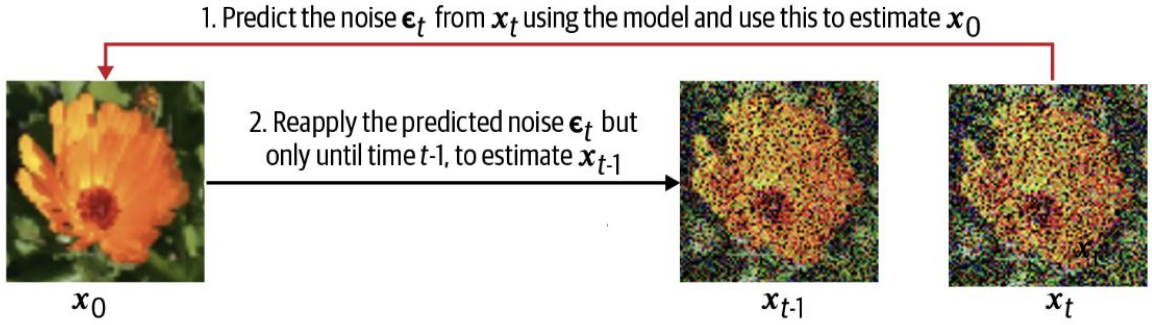


Figure 8-13. One step of the sampling process for our diffusion model

Two steps

- To achieve this, we can jump from x_t to x_{t-1} in two steps
- First by using the model's noise prediction to calculate an estimate for the original image
- and then by reapplying the predicted noise to this image, but only over t-1 timesteps, to produce x_{t-1}
- If we repeat this process over a number of steps, we'll eventually get back to an estimate for that has been guided gradually over many small steps.
- In fact, we are free to choose the number of steps we take, and crucially, it doesn't have to be the same as the large number of steps in the training noising process (i.e., 1,000).
- It can be much smaller—in this example we choose 20

Prediction and Noise

The following equation (Song et al., 2020) this process mathematically:

$$\mathbf{x}_{t-1} = \underbrace{\sqrt{\overline{\alpha}_{t-1}}\left(\frac{\mathbf{x}_t - \sqrt{1-\overline{\alpha}_t}\epsilon_\theta^{(t)}(\mathbf{x}_t)}{\sqrt{\overline{\alpha}_t}}\right)}_{\text{predicted } \mathbf{x}_0} + \underbrace{\sqrt{1-\overline{\alpha}_{t-1}-\sigma_t^2}\cdot\epsilon_\theta^{(t)}(\mathbf{x}_t)}_{\text{direction pointing to } \mathbf{x}_t} + \underbrace{\sigma_t\epsilon_t}_{\text{random noise}}$$

- The first term inside the brackets on the right hand side of the equation is the estimated image x0, calculated using the noise predicted by our network $\epsilon_\theta^{(t)}$
- We then scale this by the t-1 signal rate $\sqrt{\overline{\alpha}_{t-1}}$ and reapply the predicted noise, but this time scaled by the t-1 noise rate $\sqrt{1-\overline{\alpha}_{t-1}-\sigma_t^2}$
- Additional Gaussian random noise $\sigma_t\epsilon_t$ is also added, with the factors σ_t determining how random we want our generation process to be.
- The special case σ_t = 0 for all corresponds to a type of model known as a Denoising Diffusion Implicit Model (DDIM).
- With a DDIM, the generation process is entirely deterministic—that is, the same random noise input will always give the same output.
- This is desirable as then we have a well-defined mapping between samples from the latent space and the generated outputs in pixel space.

Workflow

1. Start with a batch of random noise images (the "latent code" that will be transformed into images).
2. Over a number of steps (e.g. 20):

    a) Use the diffusion model to predict the noise that has been added to the current images. This allows estimating the original denoised image.

    b) Calculate the new "diffusion time", which represents how far along we are in the diffusion/denoising process.

    c) Based on the diffusion time, calculate the noise rate (how much noise to re-add) and signal rate (how much of denoised image to keep).

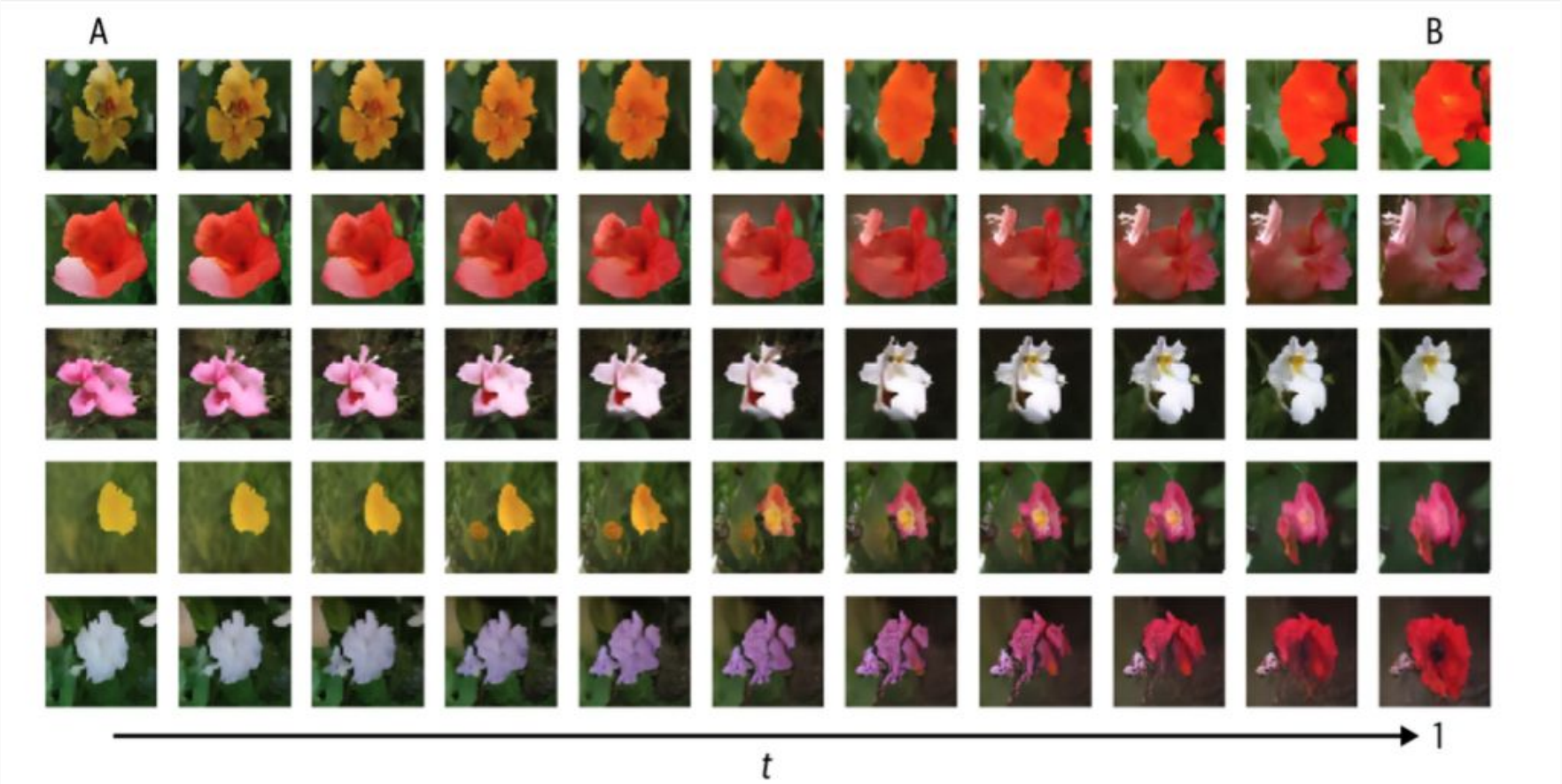
    d) Re-add some noise to the predicted denoised image, according to the calculated rates. This gives the images for the next step.
3. After the set number of steps, the final predicted images represent samples from the diffusion model, starting from just noise.

Interpolating between images

- The denoising diffusion model has a Gaussian latent space. Points in this latent space map to images in pixel space.
- We can interpolate between two points in latent space to smoothly transition between the corresponding images in pixel space.
- To interpolate, we take two randomly sampled Gaussian noise tensors and blend them together.
- Specifically, we use spherical interpolation, where we calculate the interpolated noise as:

  interpolated_noise = s * noise1 + (1-s) * noise2
- Here it goes smoothly from 0 to 1 over the number of interpolation steps.
- This form of interpolation keeps the variance of the noise constant while blending between the two noises.
- By using this interpolated noise as input to the denoising diffusion model and generating images, we can transition smoothly between two randomly sampled images from the model.
- This allows us to explore the latent space and the types of images the model has learned to generate.
- In our example the initial noise map at each step is give by $a \sin\left(\frac{\pi}{2}t\right) + b \cos\left(\frac{\pi}{2}t\right)$ where t = 0 to 1 and a & b are two randomly sampled Gaussian noise tensors.

Interpolating between images using the denoising diffusion model

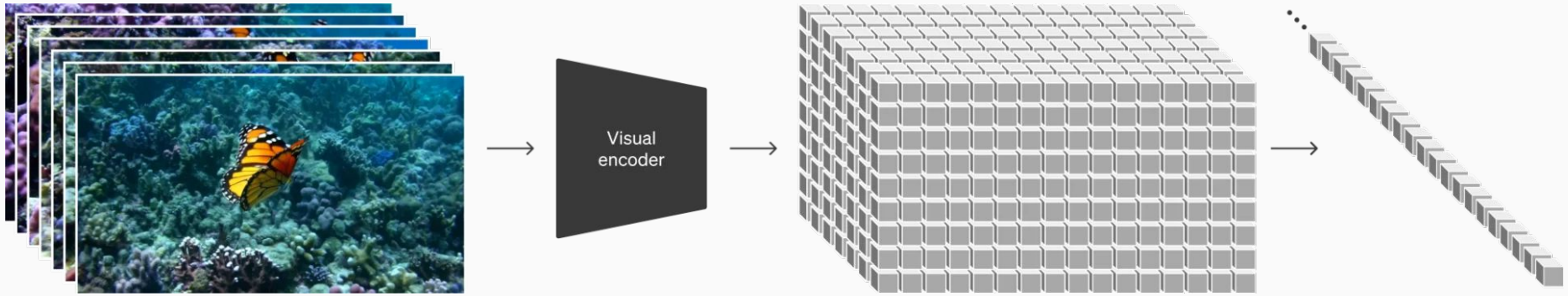


Summary

- Diffusion models sequentially add small amounts of noise to data examples through a series of latent variables. After enough steps, the representations become indistinguishable from white noise.
- The denoising process at each step can be approximated as a normal distribution and predicted by a deep learning model.
- The training loss is based on the evidence lower bound (ELBO) which results in a least-squares formulation.
- For image generation, each denoising step uses a U-Net, which is slow compared to other generative models.
- Sampling can be sped up by using a deterministic formulation and fewer steps.
- Several methods have been proposed to condition the image generation on class labels, input images, and text descriptions.
- Combining conditioning methods enables impressive text-to-image synthesis capabilities.
- In summary, diffusion models can generate high-quality images by gradually corrupting and then denoising data, with flexible conditioning approaches to control the synthesis.

Video generation models as world simulators

Video Patches



Videos into patches by first compressing videos into a lower-dimensional latent space, and subsequently decomposing the representation into spacetime patches

Large-scale training of generative models on video data

- They train text-conditional diffusion models jointly on videos/images of variable durations, resolutions and aspect ratios
- Uses transformer architecture operating on spacetime patches of video/image latent codes
- Largest model Sora can generate 1 minute of high fidelity video
- Scaling video generation models is promising path to building general purpose simulators of physical world

Turning Visual Data into Patches

Draws inspiration from large language models acquiring generalist capabilities by training on internet-scale data using text tokens

Uses visual patches as unified representation for diverse videos/images

First compresses videos into lower-dimensional latent space using trained video compression network

Then decomposes latent representation into spacetime patches used as transformer tokens

Patch-based representation allows training on variable resolution, duration, aspect ratio videos/images

At inference, can control generated video size by arranging randomly initialized patches in appropriately sized grid

Scaling Transformers for Video Generation

- Sora is a diffusion model - predicts clean patches from noisy input patches and conditioning (e.g. text prompts)
- It is a diffusion transformer model
- Shows comparison of fixed seed/input video samples as training compute increases from base to 4x to 32x
- Demonstrates diffusion transformers scale effectively as video models, with sample quality improving markedly with increased training compute

Variable Duration/Resolution/Aspect Ratio Training

Past approaches typically resize/crop/trim to standard size (e.g. 4s 256x256 videos)

Sora instead trains on data at native sizes which provides:

1) Sampling flexibility - can generate wide/tall/square videos for different devices at native aspect ratios + prototype at lower res

2) Improved framing and composition vs always cropping to square

Also leverages GPT to expand short user prompts into detailed captions sent to video model, improving prompt-following

Prompting with Images and Videos

In addition to text prompts, Sora can use image/video prompts to perform editing tasks:

1) Animating static DALL-E images into video

2) Extending generated videos forward or backward in time

3) Creating perfectly looping videos by extending both directions

4) Video-to-video style transfer and environment changes using SDEdit technique

5) Interpolating to smoothly transition between two videos

Image Generation Capabilities

- Sora is also capable of generating images up to 2048 x 2048 resolution
- Does this by arranging noise patches in a spatial 1-frame grid

Emerging Simulation Capabilities

Exhibits interesting emergent capabilities enabling simulation of aspects of real world

These arise from scale without explicit 3D/object inductive biases:

1) 3D consistency - people/objects move consistently in 3D as camera moves

2) Long-range coherence and object permanence - persists people/objects over time, occlusions, and scene cuts

3) Interacting with world - simulates simple state-changing actions like painting or eating

4) Simulating digital worlds - controls Minecraft player with basic policy while rendering world/dynamics

Limitations

- Not accurately modeling physics of many basic interactions like shattering
- Interactions like eating not always causing correct object state changes
- Other failure modes: long-range incoherencies, spontaneous object appearances
- But capabilities demonstrate continued scaling of video models is promising path to development of capable physical/digital world simulators

References

- Foster, D. (2022). *Generative deep learning*. " O'Reilly Media, Inc. - Chapter 8
- *Prince, S. J. (2023). Understanding Deep Learning. MIT press.* - Chapter 18

Homework - 6

- Run this notebook - Please write a summary (in a paragraph or two) of how the code in the notebook works.
- Run this notebook - Please discuss the positives/negatives of the Hugging Face workflow