**Diffusion Model (Denoising)**

**High-Level Overview**

***Introduction***

The essential idea is to systematically and slowly destroy structure in a data distribution through and iterative forward diffusion process.

A collage of different images of the sun

Description automatically generated  
We then learn a reverse diffusion process that restores structure in data, yielding a highly flexible and tractable generative model of the data.

A white line on a black background

Description automatically generated

When the model learns how the reverse diffusion process, to transform a random latent noise into an actual data (image), we can use it to generate other new data with the model by feeding another random latent noise.

A diagram of a model

Description automatically generated

***Forward Diffusion process***

* We don’t employ the same amount of noise in each timestep, t.
  + I.e. for each timestep t where we add noise to the image, it is not constant.
* The variance of the noise added at timestep t, Bt, is regulated by a fixed schedule (some fixed function Bt = f(t, T), where t is the timestep and T is the total number of timesteps
  + Bt, the variance of the noise, will scale the mean and the variance of the distribution of the noise sample at t
  + E.g.: Linear Schedule, Quadratic Schedule, Cosine Schedule

A blurry image of a couple of white objects

Description automatically generated

***Reverse Diffusion process***

Same as the forward diffusion process, the reverse diffusion process iteratively learn how to denoise noisy corrupted images per timestep t

1. Possible Outputs (Predictions)
   1. Mean of Noise
      1. By right, to predict noise 🡨🡪 noise distribution, we want to predict both the mean and the variance of the noise.   
         But a newer paper decided to fix the variance instead of also learning how to predict the variance as they found learning them let to unstable training and lower quality samples
      2. \*\*\* Wait, but the newest OpenAI paper found learning the variance of the noise led to better results, lead to improvements in the log-likelihood (???)
   2. Noise in the Image
      1. Same as predicting the mean of noise, just parameterized differently. It works better
      2. Which can then be subtracted from the noisy image to get a noise-reduced image

Note:   
for the reverse diffusion process, the same noisy image is passed through the model multiple times, ie Xt is passed and Xt-1 is the output, then Xt-1 is passed then Xt-2 is the output, so on and so forth till a good image X0 is obtained.

The reason why the noisy image is not completely tried to change into the good image, ie Xt to Xt-1 to…X0 within one pass through of the model is that it was found to have poor quality results.

***Model***

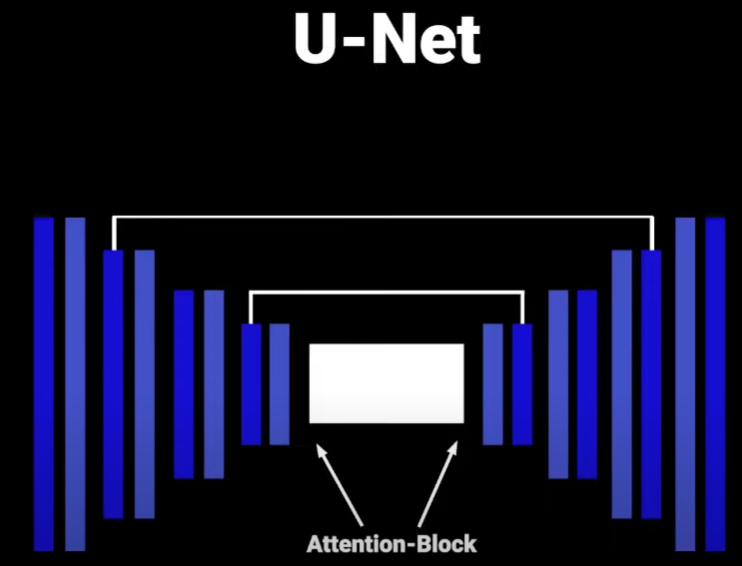
*Recap:*

1. Input: Actual Noise (noisy images from the original images)
2. Output: Predicted Noise in the image (parameterized from the Mean of noise) used to denoise the image step by step in the reverse process

*Architecture:*



1. *(2020 paper)*

**



The model:

* has a bottleneck in the middle
* takes as input an image and using down sample and ResNet blocks, it projects the image to a very small resolution (to a small latent vector)
* after the bottleneck, it tries to project the image back up to the original size, this time using up sample blocks instead
* at certain resolutions there will be;
  + attention blocks and
  + skip connections between layers of the same spatial resolution

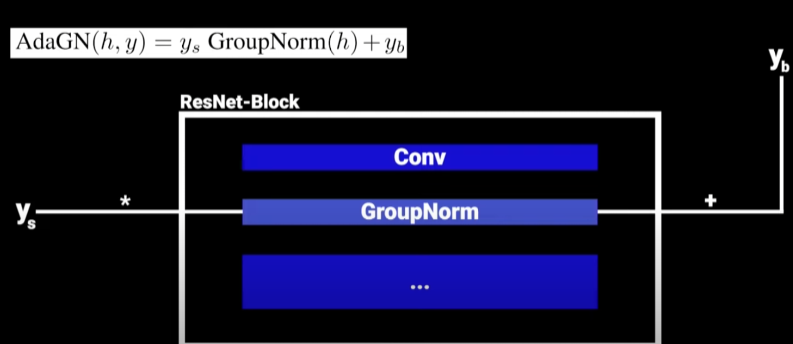
As mentioned previously, for the same noisy image, at each time step t, Xt, we pass through the entire model once. (Each time step = one entire pass through the model):

* The way we tell the model at which time step we are is done using the sinusoidal embeddings from the transformer paper
  + This embedding is projected to each residual block
  + This is important as the forward diffusion uses a schedule for the noise (variance of the noise), which scales the mean and the variance of the noisy images
    - As a result, different amount of noise is applied to the image at different time steps
    - With this information, the model can also account for removing the different amount of noise at the different time steps

1. *(Open AI paper) <Too much, needs to be a separate topic by itself>*

Improvements/Updates:

1. Increase depth, decrease width
2. More attention layers
3. More attention heads
4. Used BigGAN residual blocks for up sampling and down sampling blocks
5. Adaptive Group Normalization
   1. Making the time step and class label slightly different

, apply a GroupNorm after the first convolutional layer in each ResNet block, multiplied by Ys (linear projection of the time step) and add Yb (linear projection of the class labels)

1. Classifier Guidance
   1. Use a separate classifier to help the diffusion model generate a certain class

**Technical In-Depths of Diffusion Model**

***1. Notation***

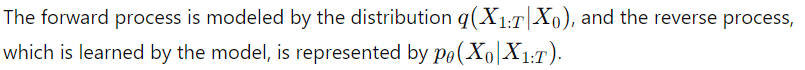
1. Xt = (noisy) image at time t
   1. E.g. X0 = original image at time step 0, no noise added, X42 = noisy image at time step 42 (original image added noise 42 times)
   2. XT is the final image, “pure Gaussian noise”, with distribution ~N(0, I),  
       t = T. T = final time step (Supposed to be infinity or a super large number)
2.  = Forward process (input = Xt-1, output = Xt) 🡨🡪  
    distribution of the noisy data output of the forward process at  
    time step t, conditioned on the data from the previous time step  
    t−1.
   1. Xt-1 = image with lesser noise than Xt (as we add noise with each increasing time step t); Xt = image with more noise than Xt-1
3.  = Reverse process (input = Xt, output = Xt-1) 🡨🡪  
    distribution of the un-noised data output of the reverse process  
    at time step t-1, conditioned on the data from the previous time  
    step t.
4.  🡨🡪 = taking multiple t number of forward process steps, basically the joint distribution of X1 to Xt conditioned on the original image X0
   1. E.g. for taking 3 forward process steps, q(X3 | X0) 🡨🡪 q(X1:3 | X0)

***2. Introduction***

**A black background with white text

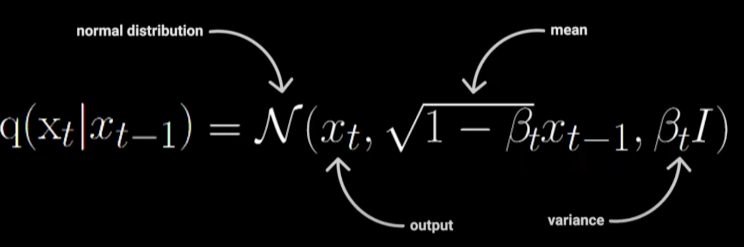
Description automatically generated**

Forward Process : For T time steps, add noise to the original training images X0, after T time steps to Xt

Reverse Process: Model will be tasked to start with the noisy image Xt, and undo the noise

***3. Forward Process***

Definition of Forward Process



A black background with white text and green symbols

Description automatically generated

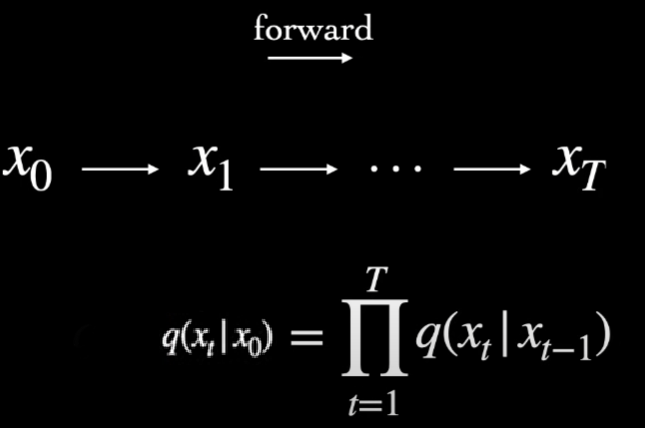
In the case of continuous data, each forward process transition q(Xt | Xt-1) can be parameterized as a diagonal Gaussian.

Bt (variance of noise added at time step t)

* Bt represents the variance of the noise added at time step t.
* Bt is a hyperparameter that typically follows a fixed schedule ( a fixed function wrt to t) for a particular training run.
  + Types of schedules for Bt: Linear schedule, Quadratic schedule, Cosine Schedule…
* Bt-1< Bt, i.e., B1 < B2 < ... < Bt-1 < Bt
* Bt is within the interval (0, 1)
  + Thus, with increasing t, it brings the mean (sqrt(1-Bt)\*xt-1 ) closer to 0.

Applying Multiple Forward steps

By Markov’s chain rule,

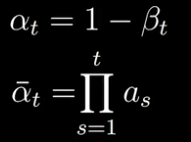
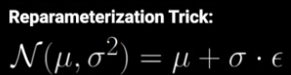


* The distribution of a corrupted noisy image at time t takes the form of a Markov chain and is solely dependent on the distribution of the previous image at time step t-1.
* Thus, we can write the joint distribution of a corrupted sample at time step t, Xt, conditioned on the original datapoint X0, q(Xt | X0) = product of the successive single step (transition) conditions.
  + Eg: Distribution of Xt for t = 3 conditioned on X0 can be written as q(X3 | X0) = q(X3 | X2) \* q(X2 | X1)

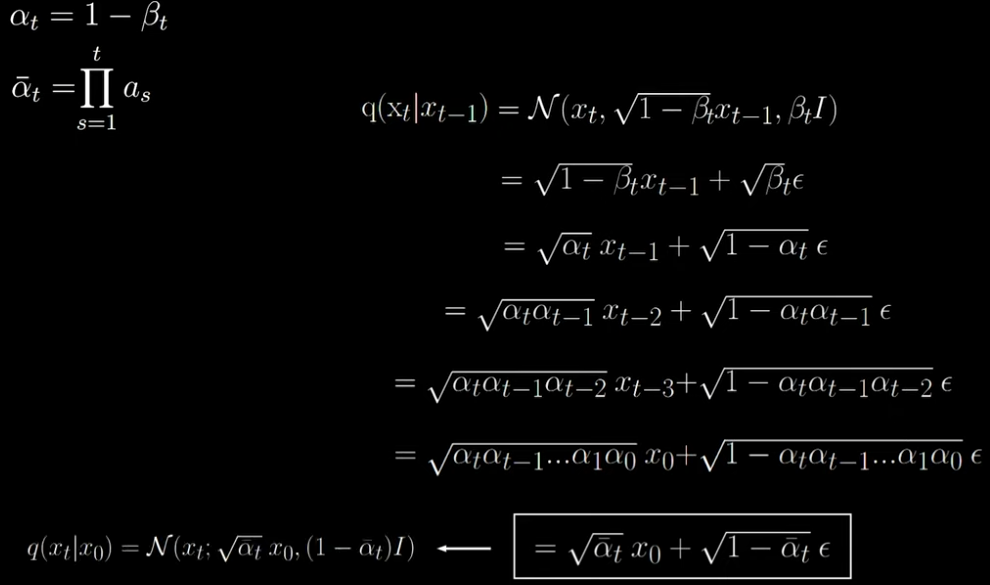
But there is a shortcut, a derivation of a new formula for  that doesn’t require the Markov Chain rule, we can apply an arbitrary number of forward steps  without needing to simulate multiple multiplications of the Markov chain rule

- don’t need to have the Markov chain till time step k to perform k forward steps to get Xk)

1. Define new notations

2. Derivation



Result:

A black background with white text

Description automatically generated

Thus, at training time any term of the objective can be obtained without simulating the entire Markov chain, using forward pass at time step t = 

More info:  
getting reverse process distribution  from forward process distribution

As T -> Infinity, q(Xt | X0 ) is approximately ~N(0, I) \*variance = I, identity matrix,  
meaning that as T -> Infinity, the adding of the noise will cause the final corrupted image’s distribution to be N(0, I), losing all information about the original sample X0.

In practice, the order of T is in thousands.

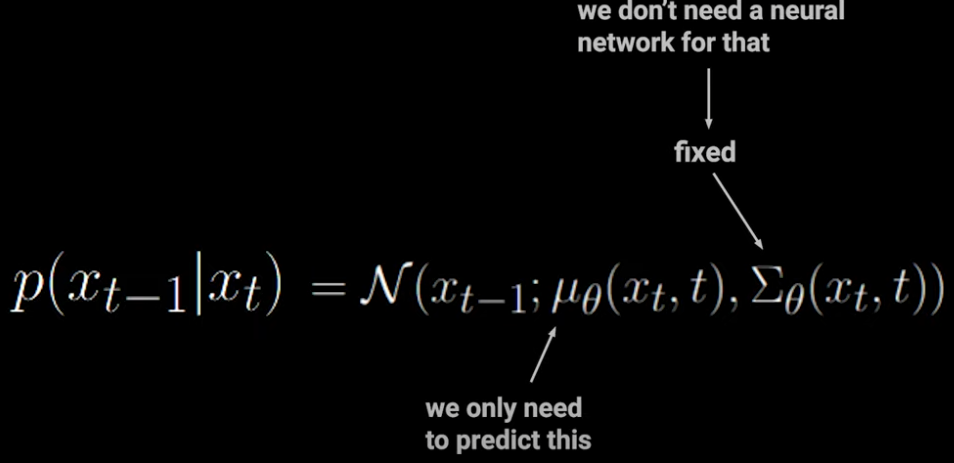
Using such a large finite T, allows us to use really small steps of Bt, setting the individual variances Bt to be very small, Bt << 1, still within (0,1).   
But what is the benefit of having smaller steps of Bt?

* This is because learning how to undo each forward step in the reverse process will not be too difficult.
  + Feller, 1949 - the limit of infinitesimal step sizes will allow the true reverse process to have the same functional form as the forward process
    - This allows the diffusion model to parameterize the reverse process, q(Xt-1 | Xt), also as a unimodal diagonal Gaussian distribution, like the forward process, q ( Xt | Xt-1 ).
      * 🡪 Each Reverse Process transition q(Xt-1 | Xt) to have the same unimodal diagonal Gaussian distribution as the Forward Process transition q( Xt | Xt-1)



***4. Reverse Process / Model***

Definition of Reverse Process

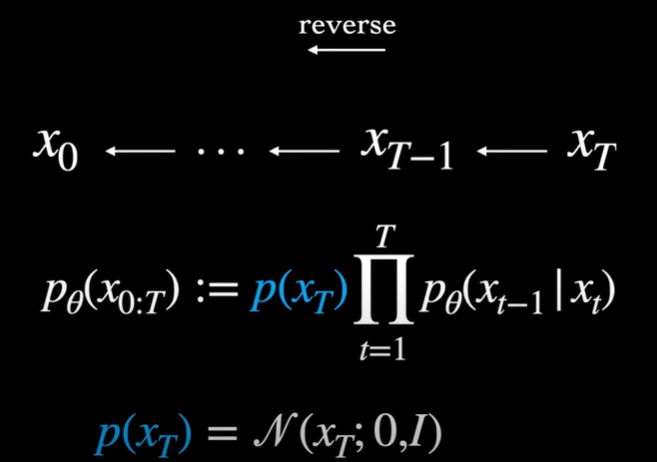
****



The mean and the variance of the reverse process is parameterized by the model (to be learned).

* Already said that variance is to be fixed, so we only need one reverse process model to learn the mean of the reverse process output noisy image distribution.

Distribution of each reverse process output image also follows the Diagonal Normal Distribution , where it also takes t as input, to account for the forward process variance schedule so that the model can undo them individually.

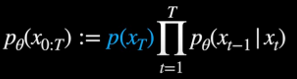
**

Like the Forward Process, we can write the joint probability of a sequence of samples in the reverse process, P(X0toXt | Xt), as a Markov chain.

Except it is not multiplied by P(Xt), the pure noise distribution (the distribution of the last noisy corrupted image Xt and time t), which as mentioned before, should be ~N(0, I)

So, at inference time, to actually generate a sample, we start from the Gaussian N(0, I) and begin sampling from the learned individual steps of the reverse process P(Xt-1 | xt) until we produce X0.

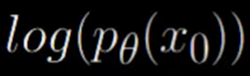
***5. Variational Lower Bound (Loss / Objective Function)***

End Goal: to learn and achieve ****** to be able to successfully denoise the maximal noisy image (Gaussian noise XT) to X0

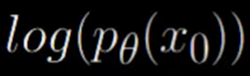
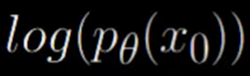
Recall , so to learn , it needs to learn Mean Theta (Xt, t) 🡨🡪 its model parameters theta (variation is fixed), by optimizing its parameter theta with some objective function.

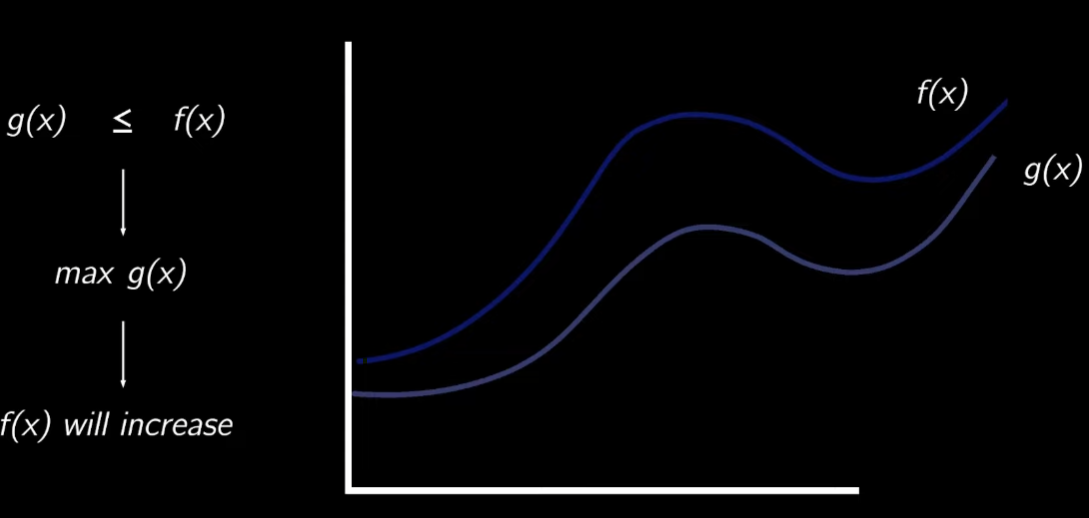
Original objective function

So far, to summarize, the forward process is to push a sample off the data manifold and transform it to noise and the reverse process is trained to produce a trajectory back to the data manifold to get a reasonable sample. But what is the objective we are minimizing?

Intuitively, we want to optimize the models parameters, theta, to maximize the objective function, log likelihood function of producing the good image, .

But , so finding P(X0) involves integrating over all the intermediate latent variables (noisy steps X1, X2, …, XT-1, XT),  
so finding P(X0) is intractable.

So we need to find the next best function to maximize tractably. By knowing its lower bound function, a function that is always lower than , we know that if we maximize that lower bound function, and still pushes the model to maximize the true likelihood .



Intuition for Variational Lower Bound as the objective function

A diagram of a decoder

Description automatically generated

In Variational Auto Encoders (VAE):

1. encoder trained to learn a conditional distribution of latent noise z given input training X q(z | X) (encoding training data into noise) 🡨🡪 How to encode the training samples into latent noise with this distribution and
2. decoder takes in a sample z from the learned q(z | X) and learn how to decode/reconstruct q(Z | X) back into a decoding distribution P(X | z) that reconstructs z back to x.

In Diffusion Models, we can think of the forward process analogous to VAE’s encoder, producing latent noise from data and the reverse process analogous to VAE’s decoder, producing data from latent noise.

But unlike the VAE encoder, the diffusion’s forward process is typically fixed, it is the reverse process that we are fully focused on learning.   
This means that for diffusion, only a single network has to be trained (unlike VAE, where 2 networks are trained simultaneously).  
Thus, we can borrow the basic training objective function used by VAEs, deriving the variational lower bound:

A black background with white text

Description automatically generated

When we have an observed variable x and a latent variable z, we can derive something called a variational lower bound (evidence lower bound). It is the lower bound on the marginal log likelihood P theta (X) 🡨🡪 log P(X | theta), theta being the models parameters.

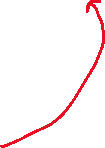
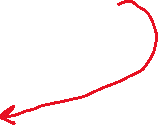
Final and simplified objective function





A black background with white text

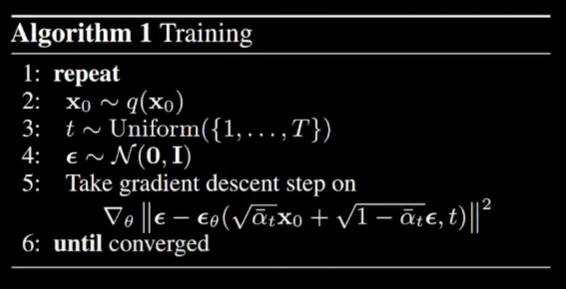
Description automatically generated

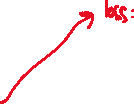


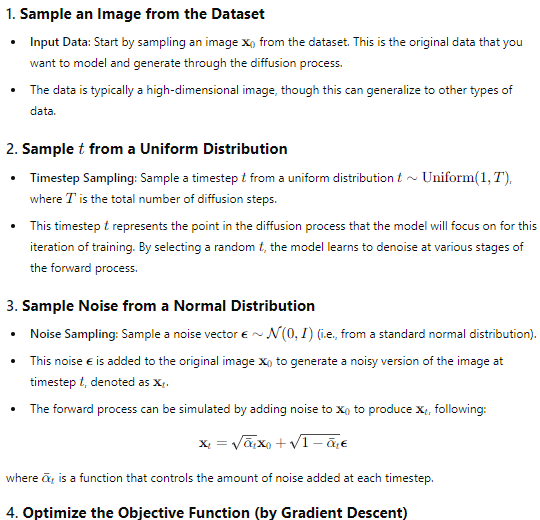
A black background with white text

Description automatically generated

***6. Algorithm***

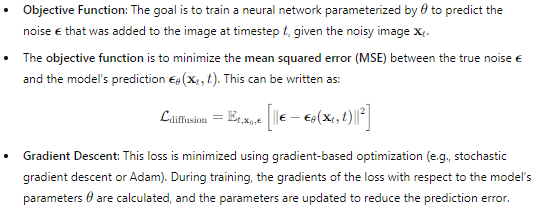


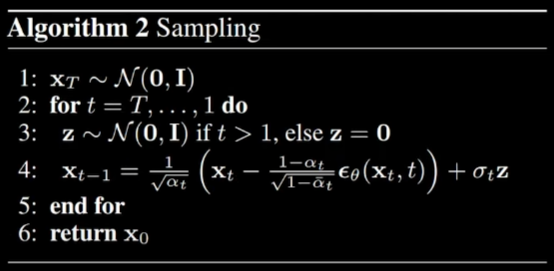


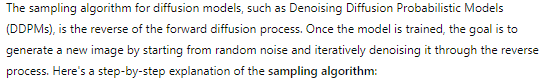


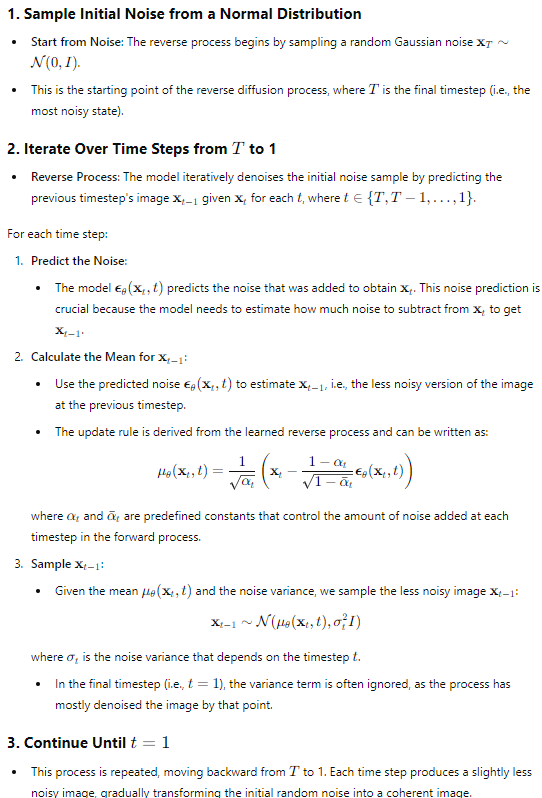




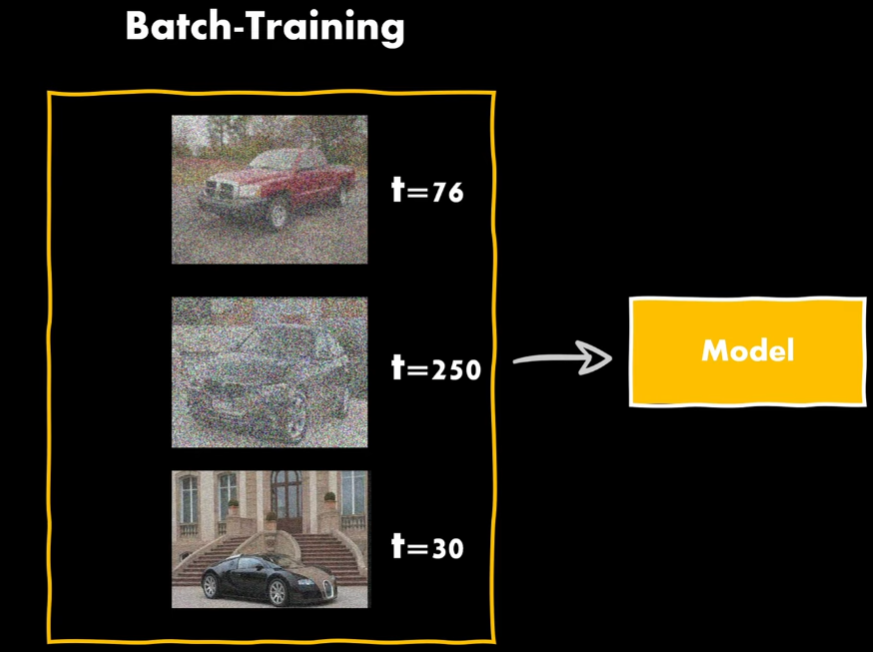








Training Overview



For Batch Training, each batch, we have different images, with different amount of noise steps added (t) {with closed form formula, we can get it instantly without Markov chain}

But we also thus need to pass in the time since the data is randomly sampled, meaning that we have to pass the time t for each training image, for model to know how much noise was added at that time step t.

**Annex**

1. **Heavy Math to reformulate Variational Lower Bound to simplest form**

<https://www.youtube.com/watch?v=HoKDTa5jHvg&t=749s>

https://www.youtube.com/watch?v=H45lF4sUgiE&list=PL8VDJoEXIjpo2S7X-1YKZnbHyLGyESDCe



Variational Lower Bound = likelihood term / construction term (the expectation term), subtracted by a KL divergence term (the DKL term)

A black background with white text

Description automatically generated

Z, the latent noise, can be substituted by X1:t (X1 to Xt) are the corrupted noisy samples  
X, the ground truth, can be substituted by X0, which is the original image at t=0

Recap:   
q is the distribution of the forward processes;   
P theta is the distribution of the reverse processes conditioned on the model parameters theta (only P has theta as we are only learning the reverse process with the model)

Subbing Z = X1:T, X = X0 for the context of diffusion models, we get:



Simplifying the objective function:

The KL Divergence term can be expressed as another Expectation term:

A black background with white text

Description automatically generated

The 2 expectation terms can be combined into one expectation term:

A black background with white text

Description automatically generated

A black background with white text

Description automatically generated

And finally, we can refactor the chain probabilities into a single step:

A black background with white text

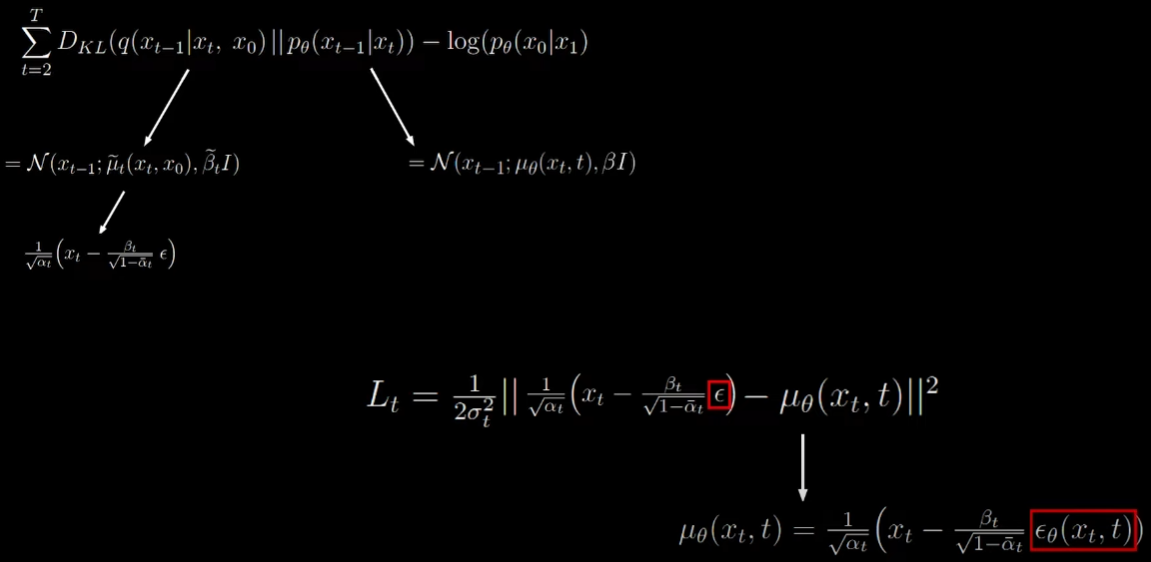
Description automatically generated

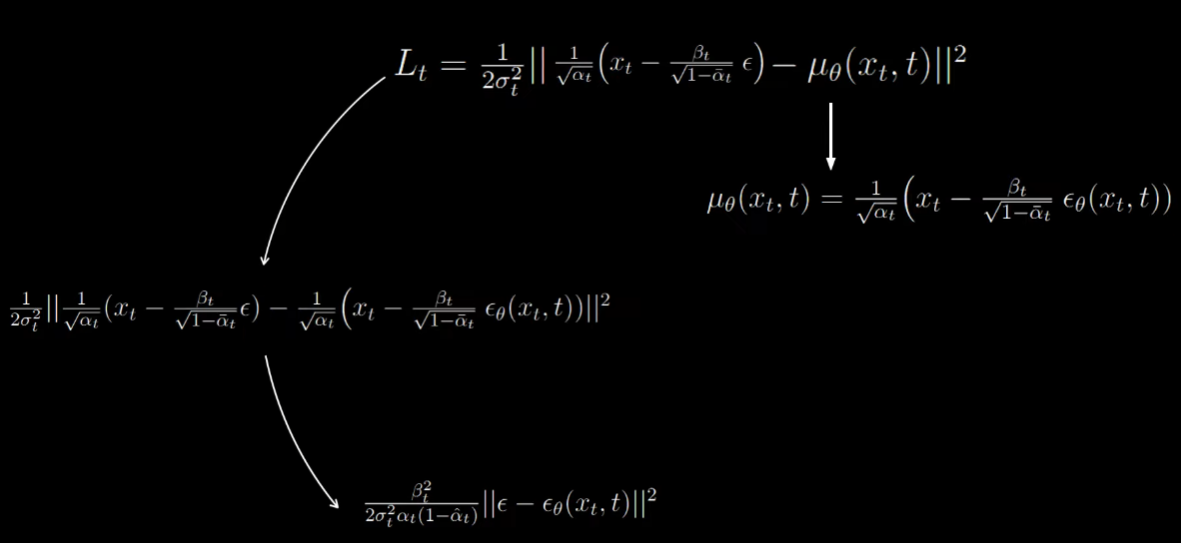
To reduce variance in the process further, we reduce the objective function to its final form:

A black background with white text and orange lines

Description automatically generated

More math…





Authors also found that a simpler version of the variational bound lead to better results by removing the scaling quantity.

A black background with text and symbols

Description automatically generated

Final Result:

A black background with white text

Description automatically generated

***A screenshot of a computer program

Description automatically generated***