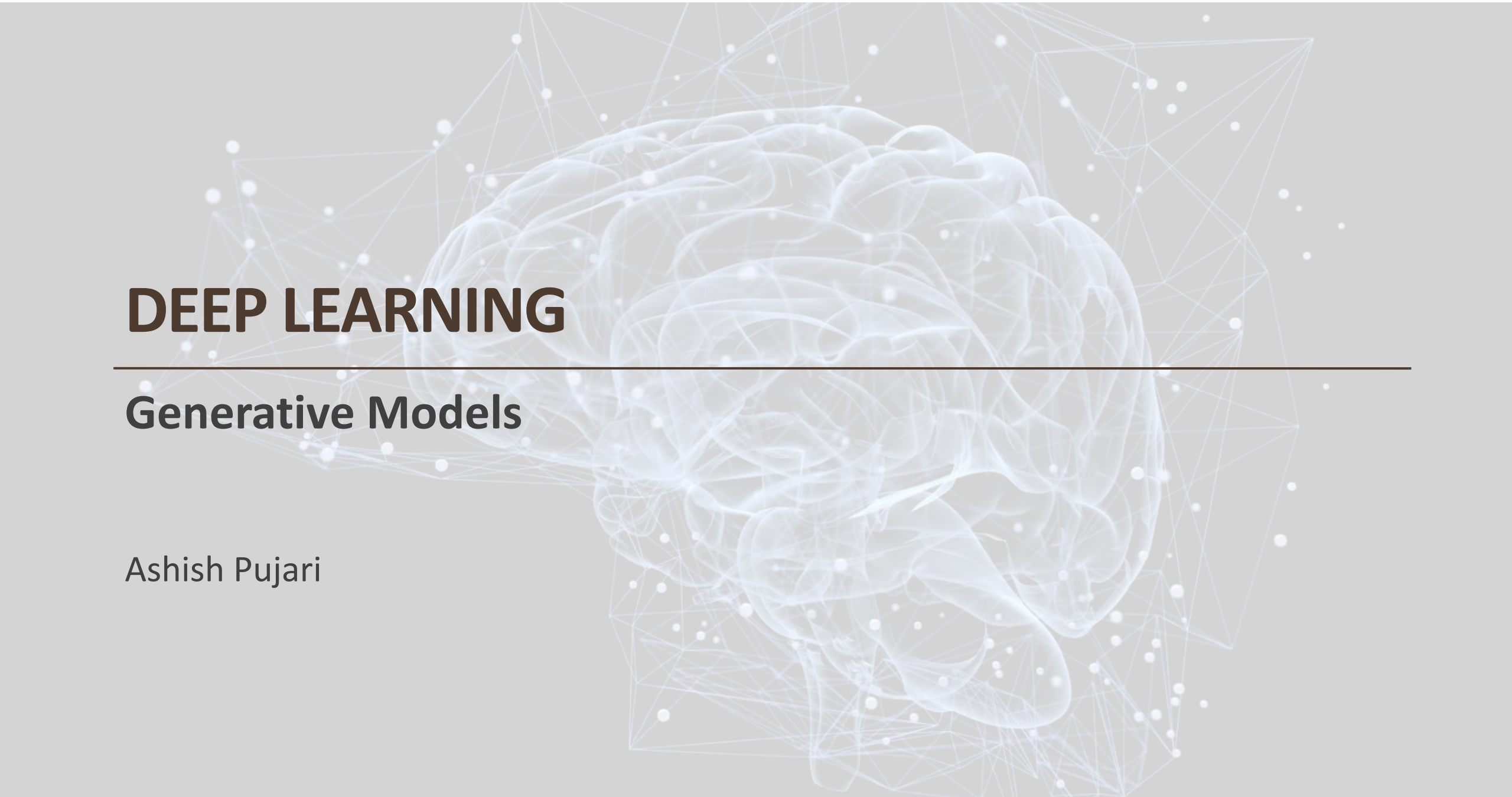


DEEP LEARNING

Generative Models

Ashish Pujari

Generative Models

1. Generative Models
2. Autoencoders
3. Variational Autoencoders (VAE)
4. Generative Adversarial Networks (GAN)
5. Diffusion Models

GENERATIVE MODELS

Generating Data

- How can we generate new/synthetic data using Machine Learning

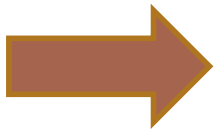
~~Text from images~~ Images from Text !

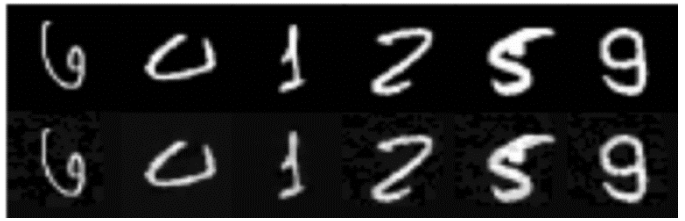
https://arxiv.org/pdf/1612.03242v1.pdf

Generative Models

- Model **how** the data was generated in order to categorize a signal. They learn the joint probability distribution $p(label, data)$
- Take training data and form a representation of a probability distribution that explains where the samples originated from, and generate new samples that distribution

Training Data: $p_{data}(x)$



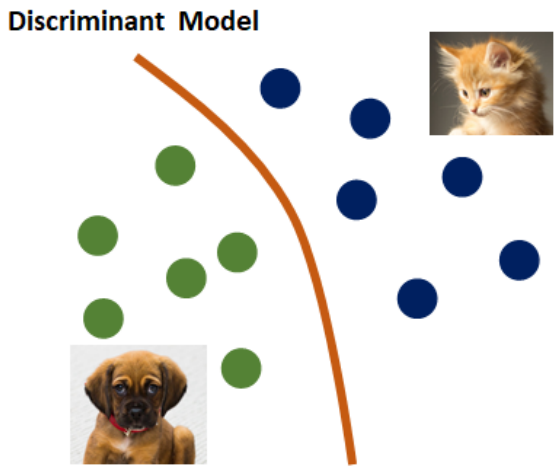


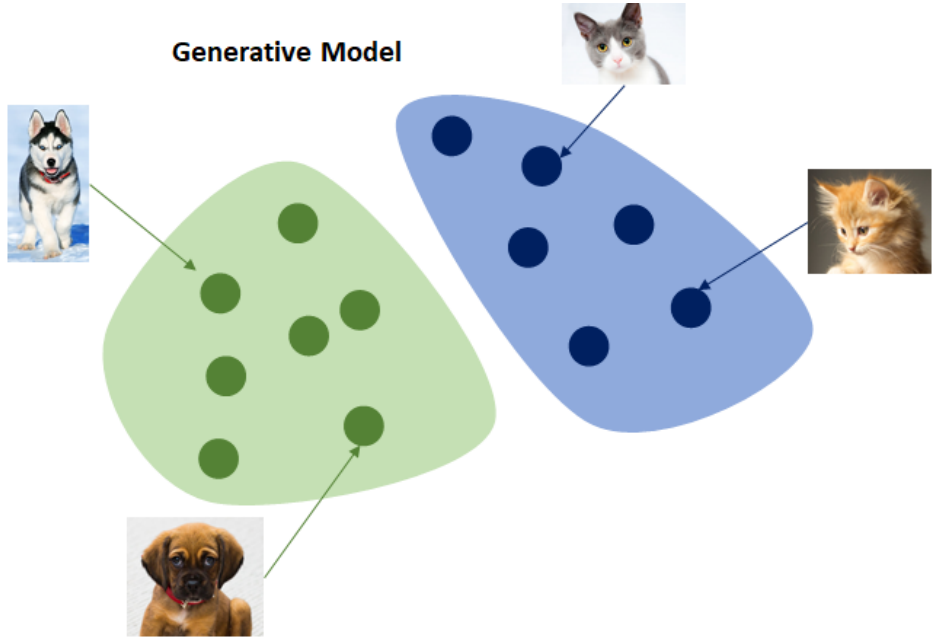
Generated Samples : $p_{model}(x)$

Discriminative vs Generative Models

Discriminative model learns the conditional probability distribution $p(y|x)$

Generative model learns the joint probability distribution $p(y, x)$



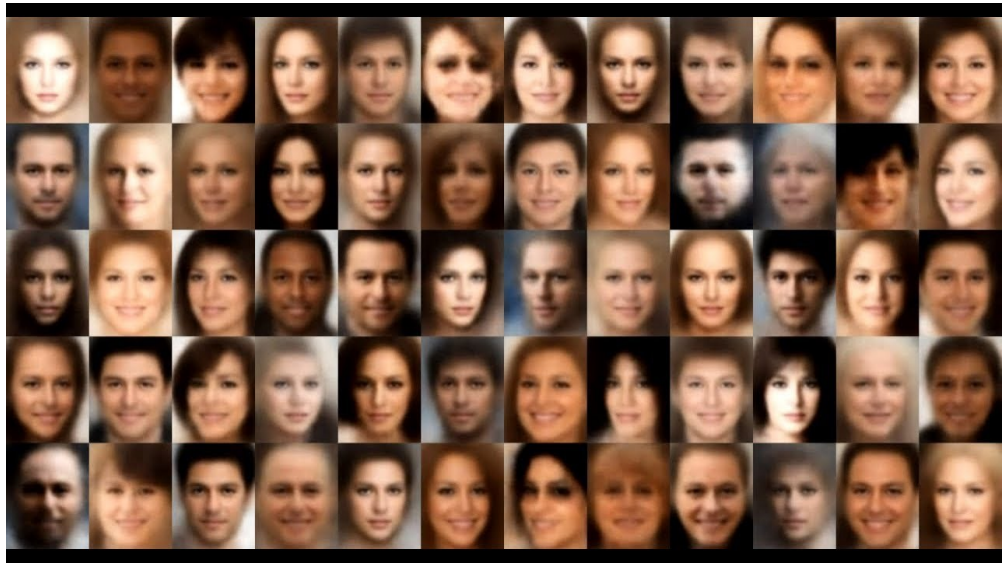


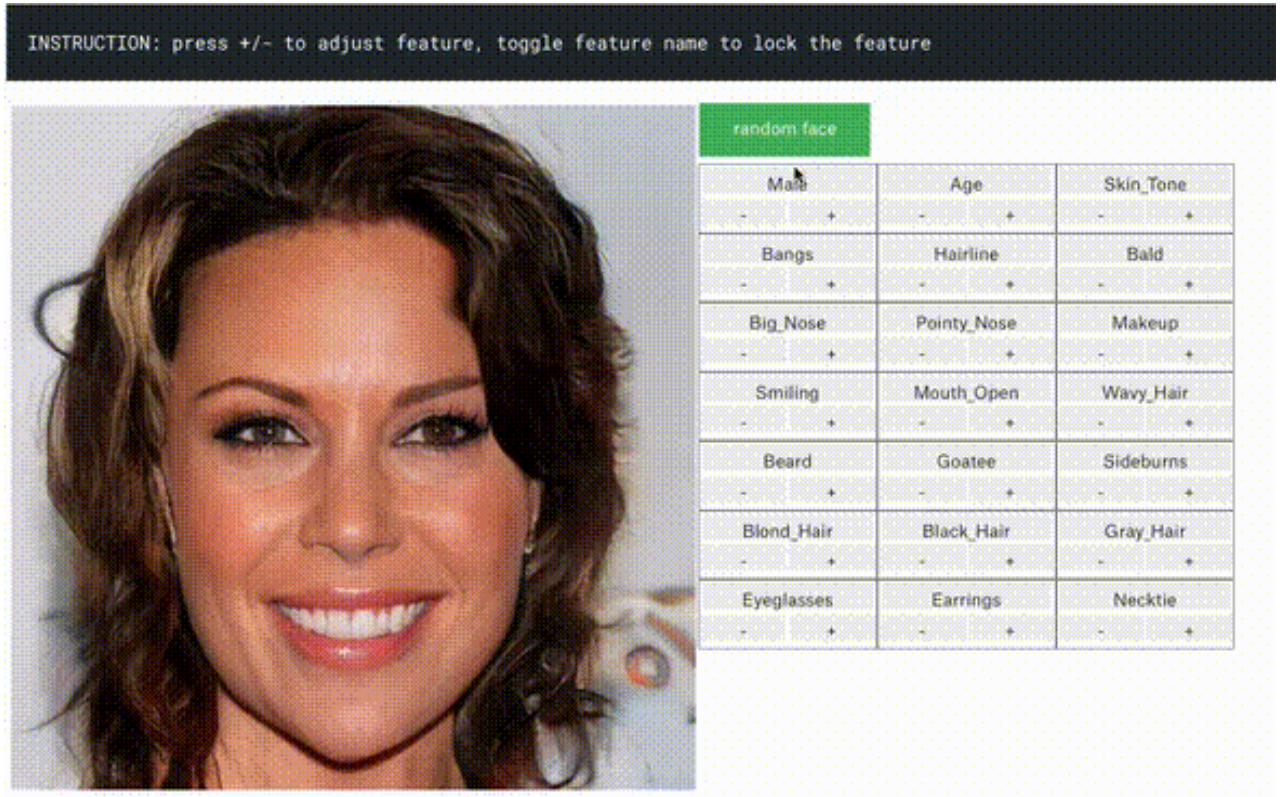
$p(y|x)$ can be inferred from observations $p(y, x)$ by applying Bayes rule $p(y|x) = \frac{p(y,x)}{p(x)}$

Applications

- Data augmentation
  - Generate missing data or semi-supervised learning
  - Model very complex high dimensional distributions
  - Multimodal outputs – generating the next frame in a video
  - Simulate potential futures for learning algorithms
- Content creation
  - Generate high-resolution photo-realistic images from text, sketches into photos
  - Create simulated environments, artificial worlds, visual manipulation
  - Content-aware smart editing: Manipulate real samples with assistance of a generative model
  - Fashion design, engineering, manufacturing, etc.

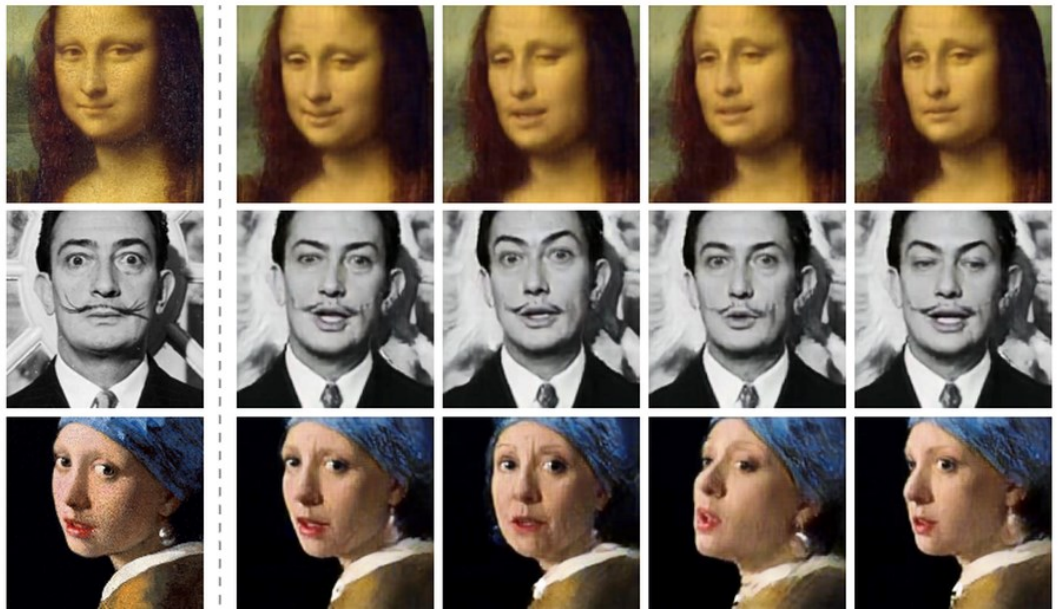
Generated Faces (2018)

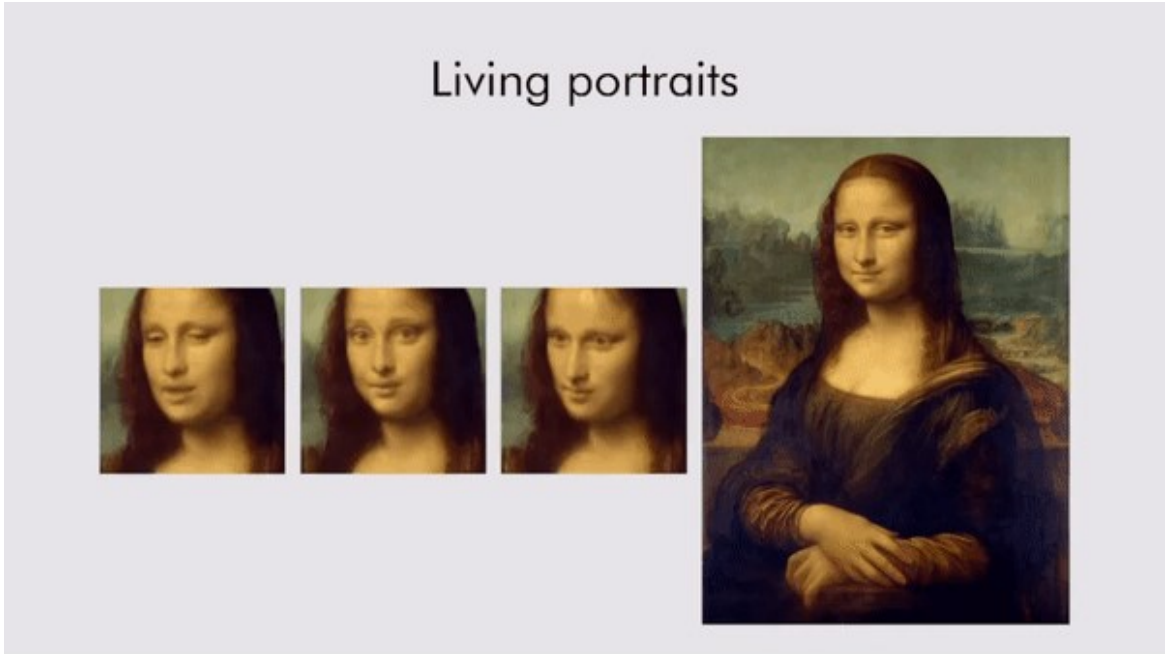




https://blog.insightdatascience.com/generating-custom-photo-realistic-faces-using-ai-d170b1b59255

Samsung DeepFake AI (2019)





Samsung Deepfake AI can fabricate a video clip of you from a single photo

Open AI DALL-E (2021)

vibrant portrait painting of Salvador Dalí with a robotic half face

a shiba inu wearing a beret and black turtleneck

a close up of a handpalm with leaves growing from it

an espresso machine that makes coffee from human souls, artstation

panda mad scientist mixing sparkling chemicals, artstation

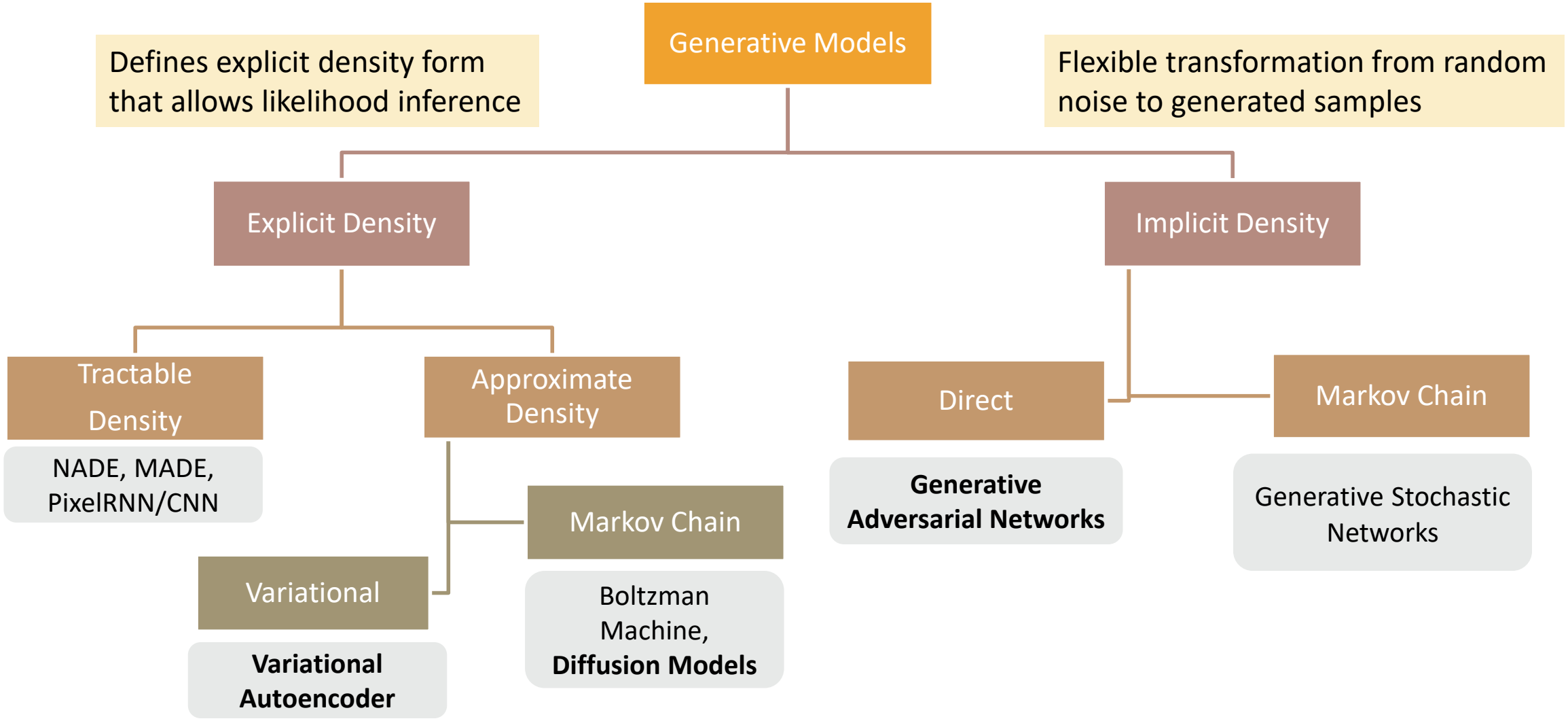
a corgi's head depicted as an explosion of a nebula

https://openai.com/blog/dall-e/

Stable Diffusion (2022)

https://stablediffusionweb.com/

Generative Models Taxonomy

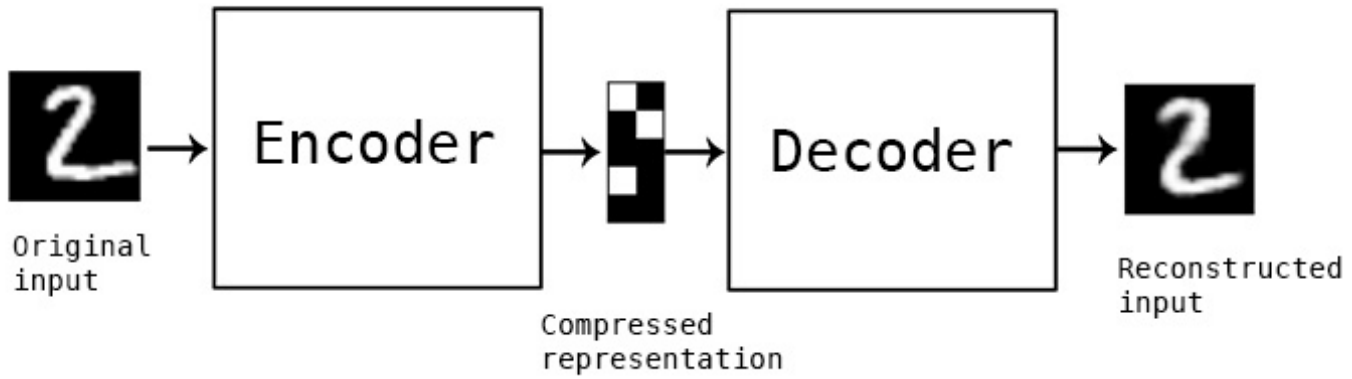


https://arxiv.org/abs/1406.2661

AUTOENCODERS

Autoencoder

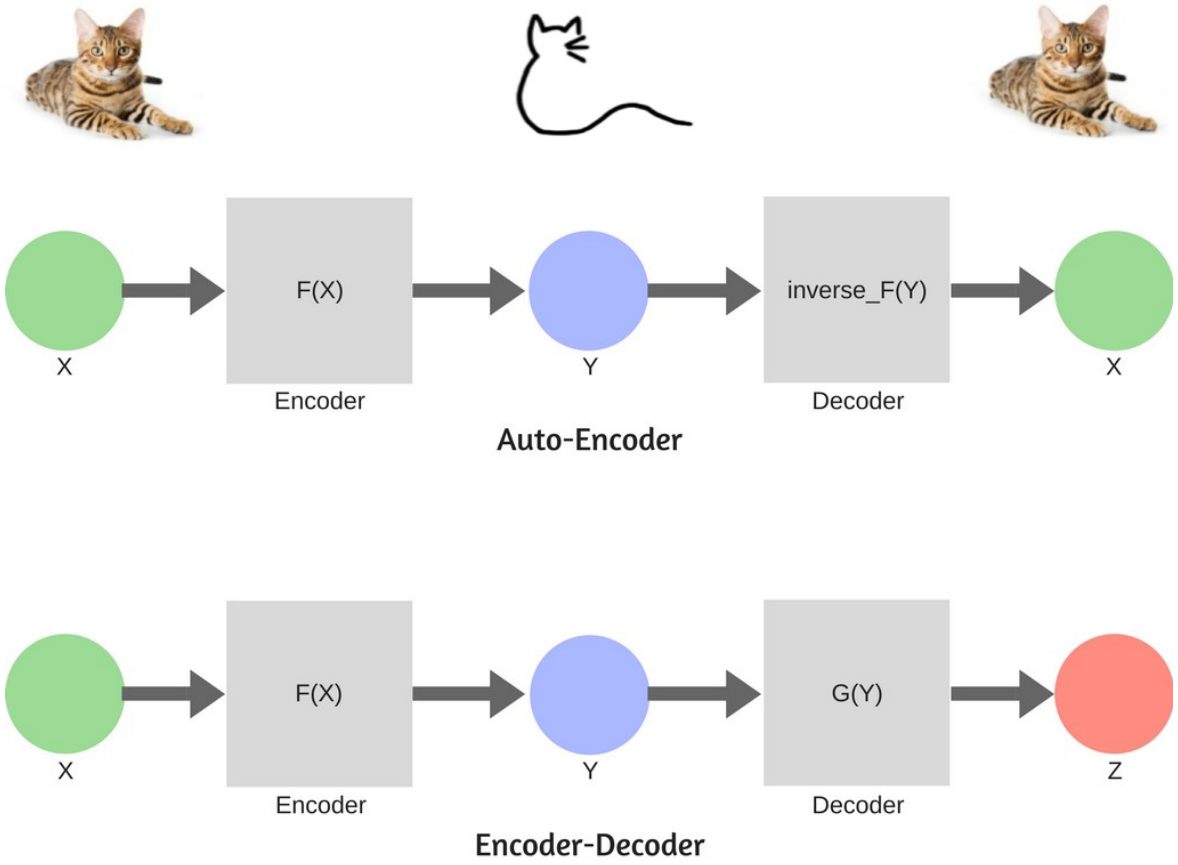
- Autoencoding is a data compression algorithm that is data-specific and lossy
  - Data-specific: only able to compress data similar to what they have been trained on
  - Lossy: Decompressed outputs will be degraded compared to the original inputs
- An un(self)-supervised network that produces a low dimensional representation of a high dimensional input



Input and output layers have the same dimensionality. By limiting the number of hidden units, interesting structures emerge about the data

Encoder-Decoder Model

- The Encoder is a compression algorithm and Decoder a generating algorithm



Autoencoder Components

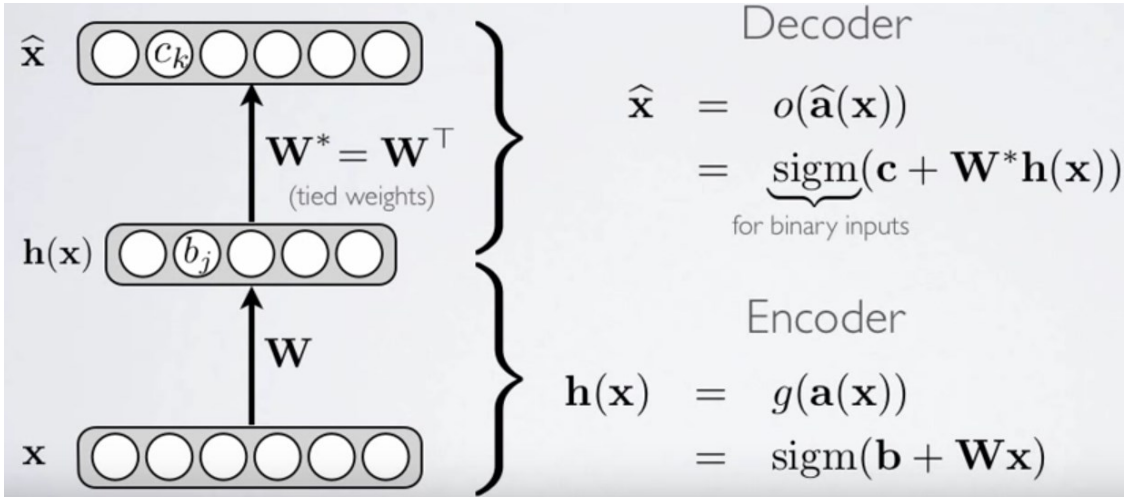
- Encoder
  - The part of the model that computes the hidden layer $h(x)$ from the input x
- Decoder
  - Takes the output of the encoder $h(x)$ and computes the output $\hat{x}$
- Loss Function
  - Distance function between the amount of information loss between the compressed representation and the decompressed representation of the data
- Training
  - Autoencoders are trained using backpropagation

$$\text{output} = \text{decode}\,(\text{encode}(\text{input}))$$

$$\hat{\boldsymbol{x}} = \boldsymbol{g}(\boldsymbol{f}(\boldsymbol{x}))$$

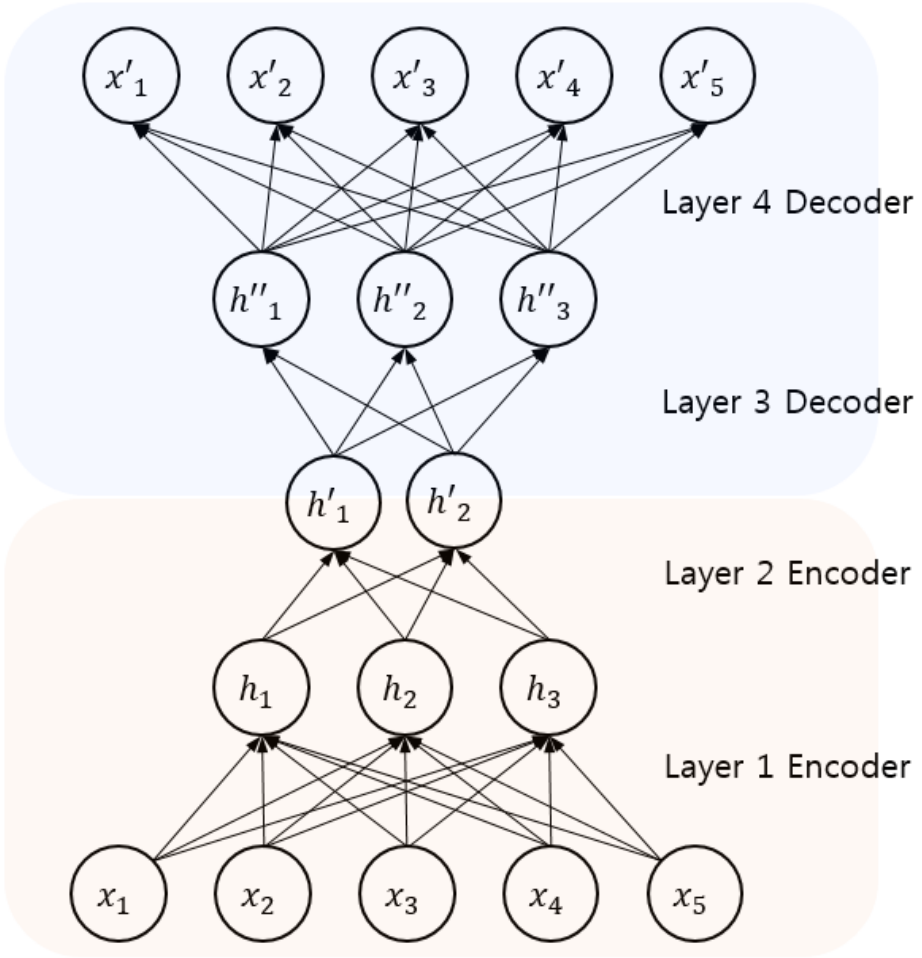
$$\text{Loss Function} = \mathrm{L}(\mathrm{x}, g(f(x)))$$

Learns approximation to the identity function, i.e. target output $\hat{x}$ that is similar to input x



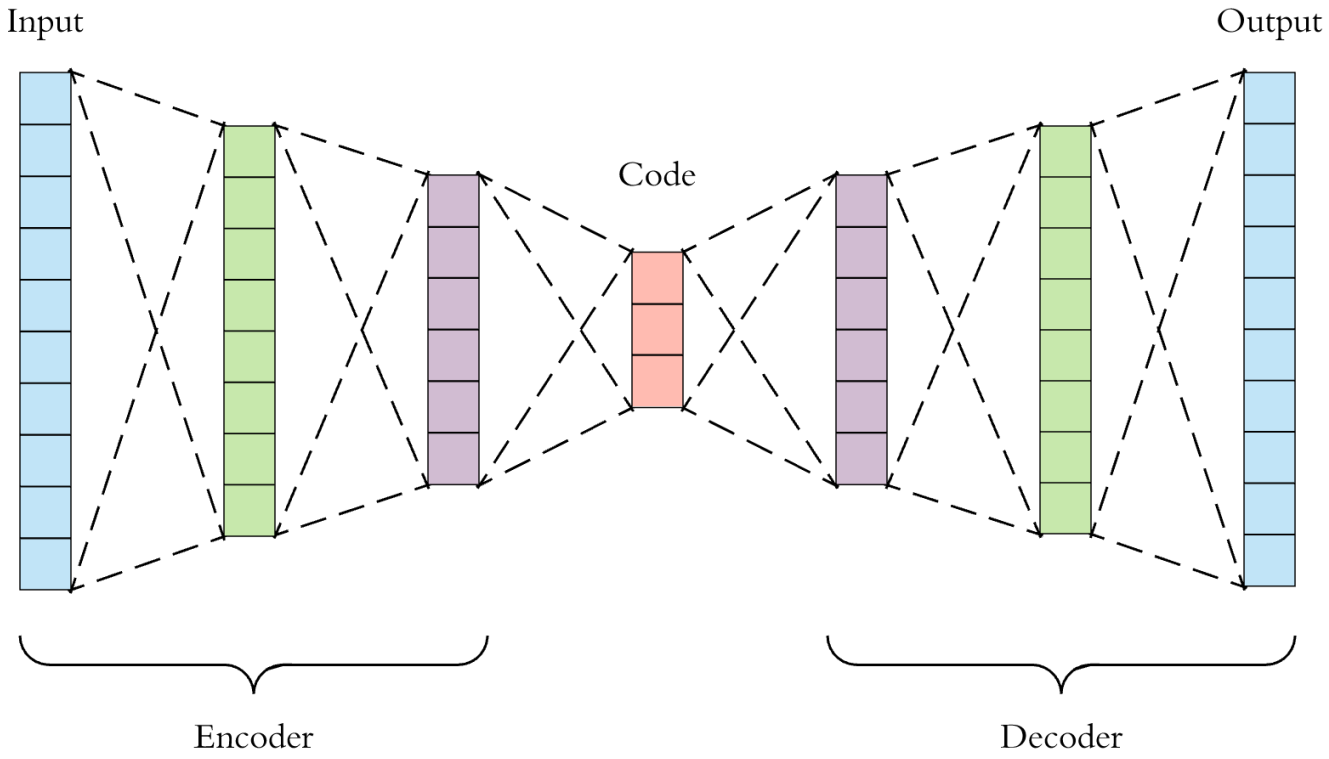
Stacked Autoencoders

- Autoencoder can be stacked to create depth
- Each layer is trained in turn, and used as input for the next layer
- This provides an effective initialization of the network, prior to supervised learning
- Experimentally, deep autoencoders yield much better compression than corresponding shallow or linear autoencoders



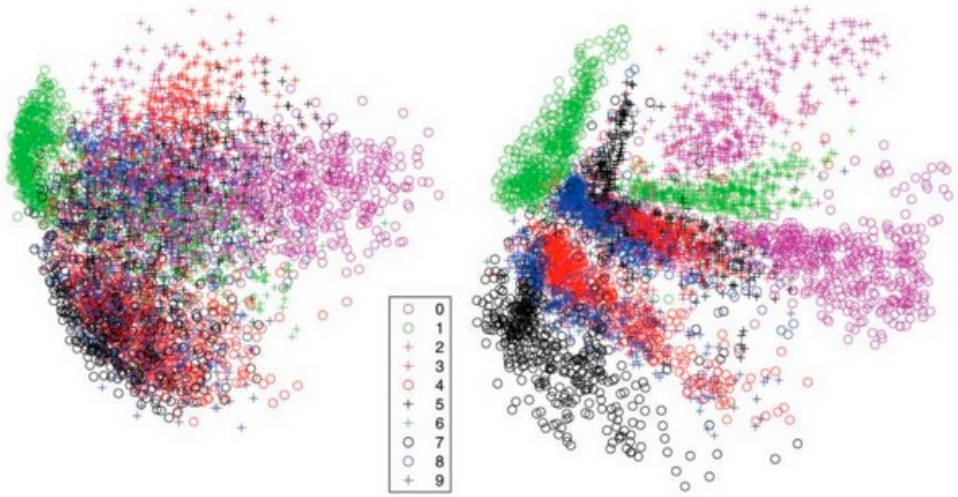
Autoencoder vs Deep Learning

Auto-encoders mostly aim at reducing feature space in order to distill the essential aspects of the data vs more conventional deep learning which blows up the feature space up to capture non-linearities and subtle interactions within the data

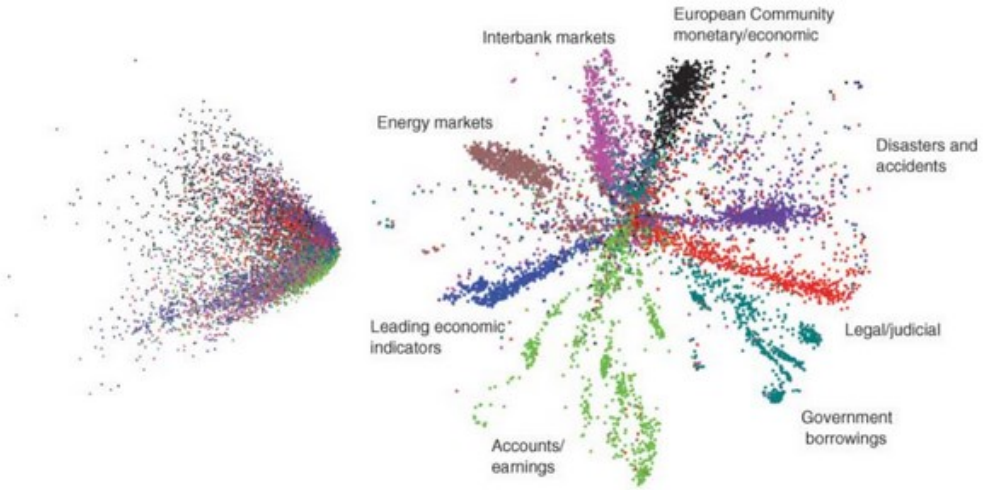


Autoencoder vs PCA

- PCA is a **linear** method which finds the directions of maximal variance in high-dimensional data. Autoencoder with Linear decoder *g* and mean squared error loss is similar to PCA
- Autoencoder with nonlinear encoder and decoder functions can learn a more powerful generalization of PCA



A comparison of separability of 2-dimensional codes generated by an autoencoder (right) and PCA (left) on the MNIST dataset

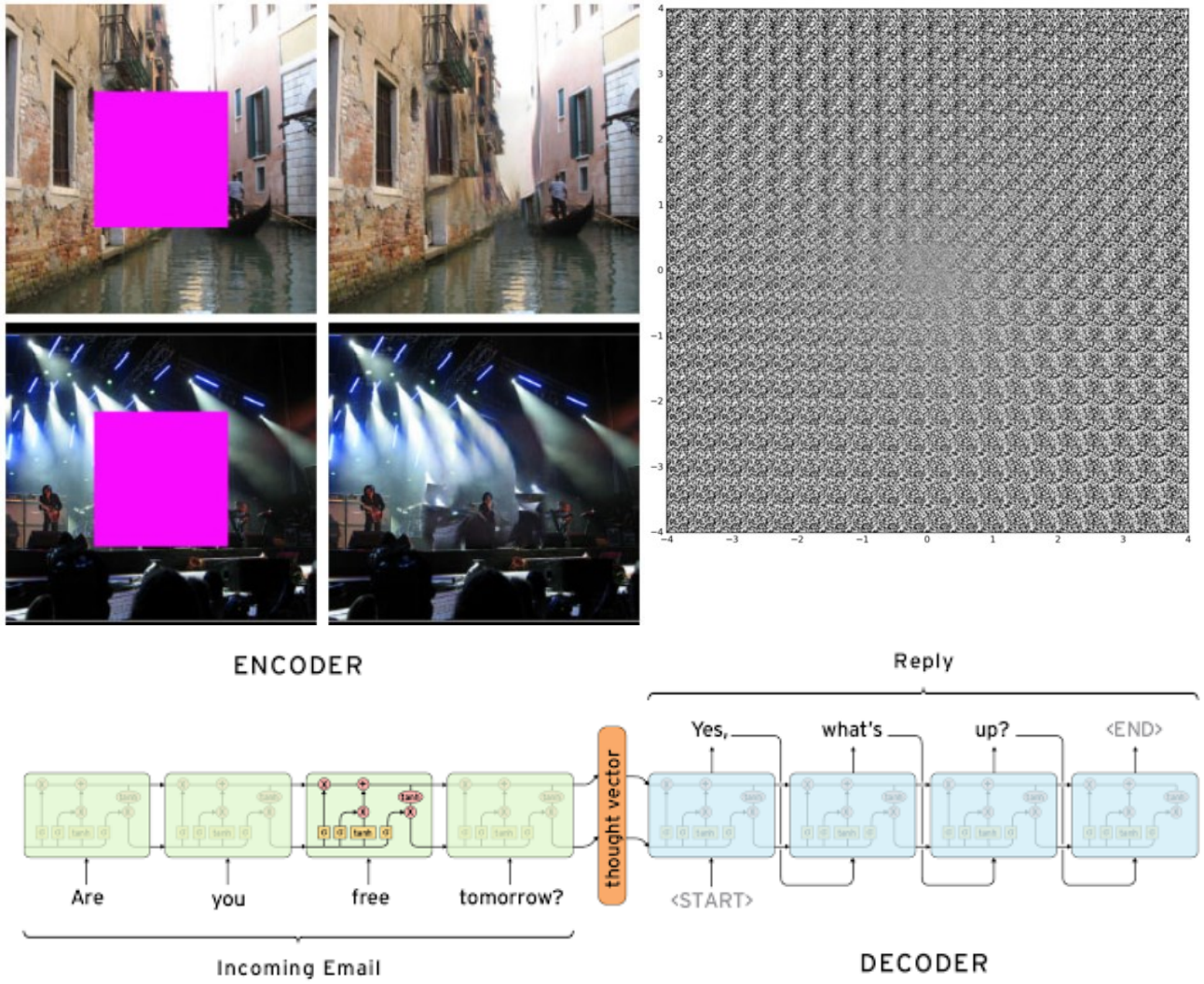


A comparison of separability of 2-dimensional codes generated by an autoencoder (right) and PCA (left) on news stories

http://nikhilbuduma.com/2015/03/10/the-curse-of-dimensionality/

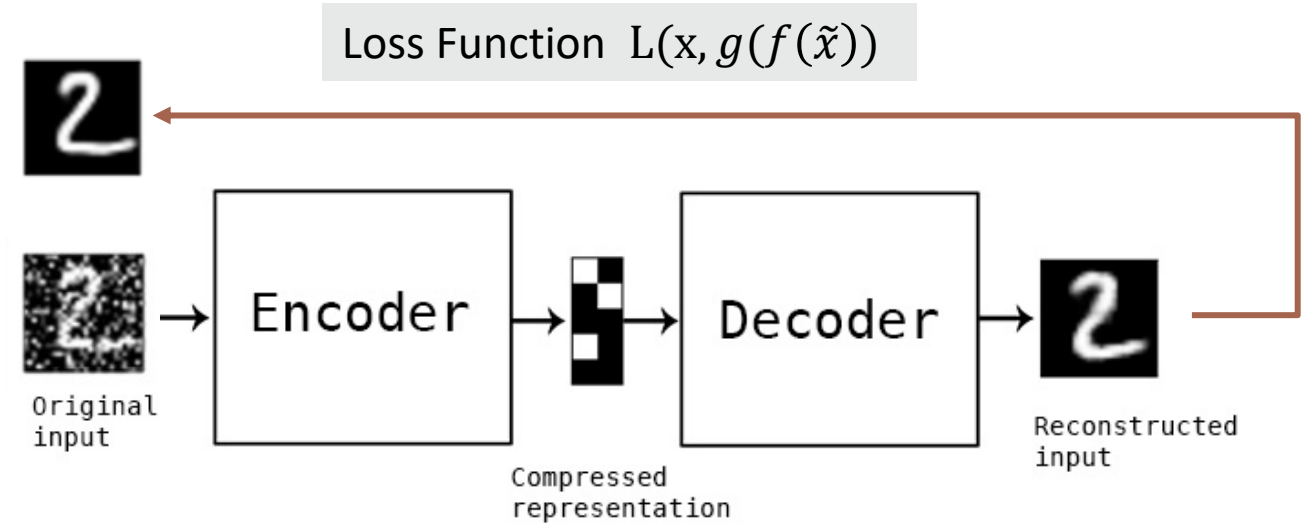
Autoencoder Applications

- Dimensionality reduction
  - Image compaction for training
  - Not better than compression algorithms (JPEG, GIF, etc.)
- Denoising Images
- Anomaly Detection
- Image Inpainting
- Information Retrieval
  - Semantic hashing
- Pre-training DNNs
  - Train stacked AE in an unsupervised manner to obtain weights (not very popular)



Denoising Autoencoders

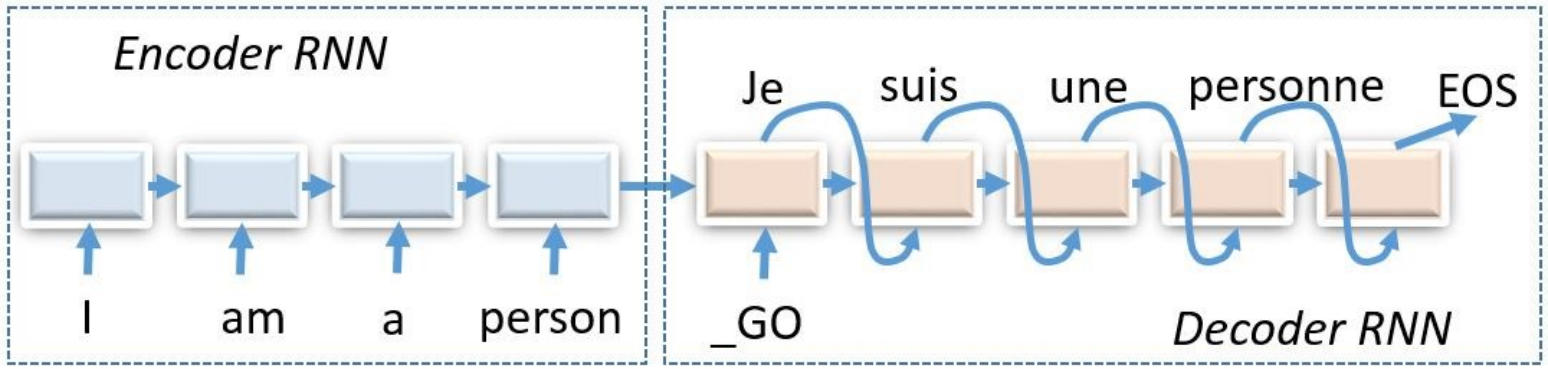
- Stochastic version of the auto-encoder which adds noise to inputs to avoid overfitting
- Denoising training forces *g* and *a* to implicitly learn the structure of *p* data (*x*)
- The goal is to:
  - encode the input (rather than noise)
  - undo the effect of a corruption process stochastically applied to the input of the auto-encoder



https://blog.keras.io/building-autoencoders-in-keras.html

Recurrent Autoencoder

- Special case of sequence-to-sequence (seq2seq) models where encoder and decoder are replaced by RNNs such as LSTMs
- Applications : Neural machine translation (NMT), time series modeling



http://rickyhan.com/jekyll/update/2017/09/14/autoencoders.html

VARIATIONAL AUTOENCODERS (VAE)

Bayesian Inference

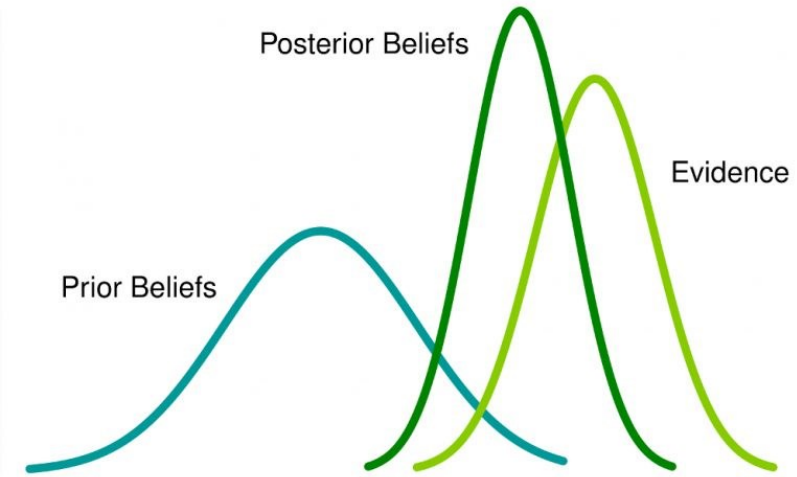
Likelihood of observations

Posterior Probability

Prior Probability

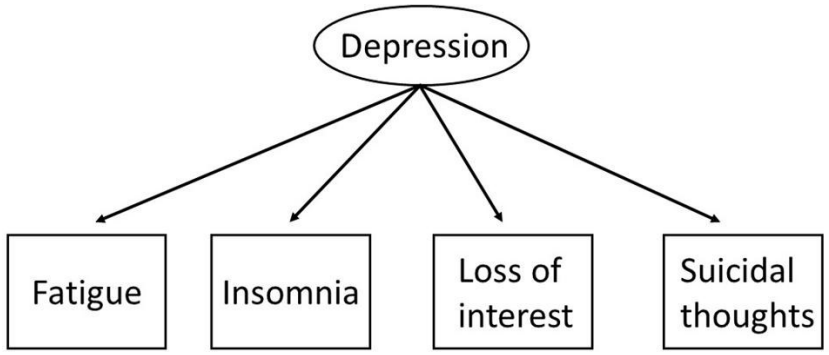
$$P(\theta|X) = \frac{P(X|\theta)P(\theta)}{P(X)}$$

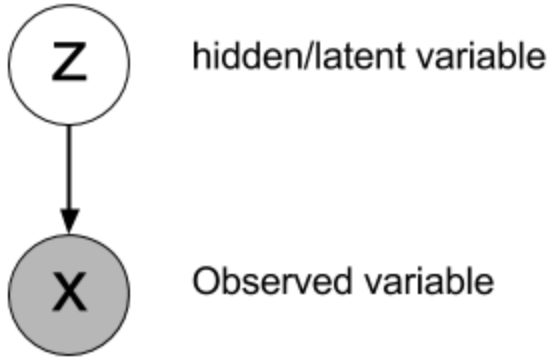
Evidence Marginal Likelihood

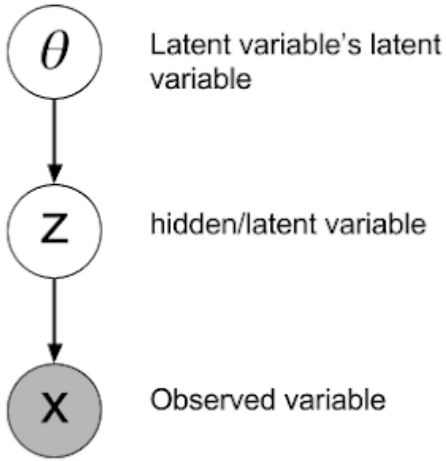


Latent Variables

- Variables that are not directly observed but are rather inferred (through a mathematical model) which we assume to have generated our actual training data.
- Latent variables can store useful information about the type of output the model needs to generate.

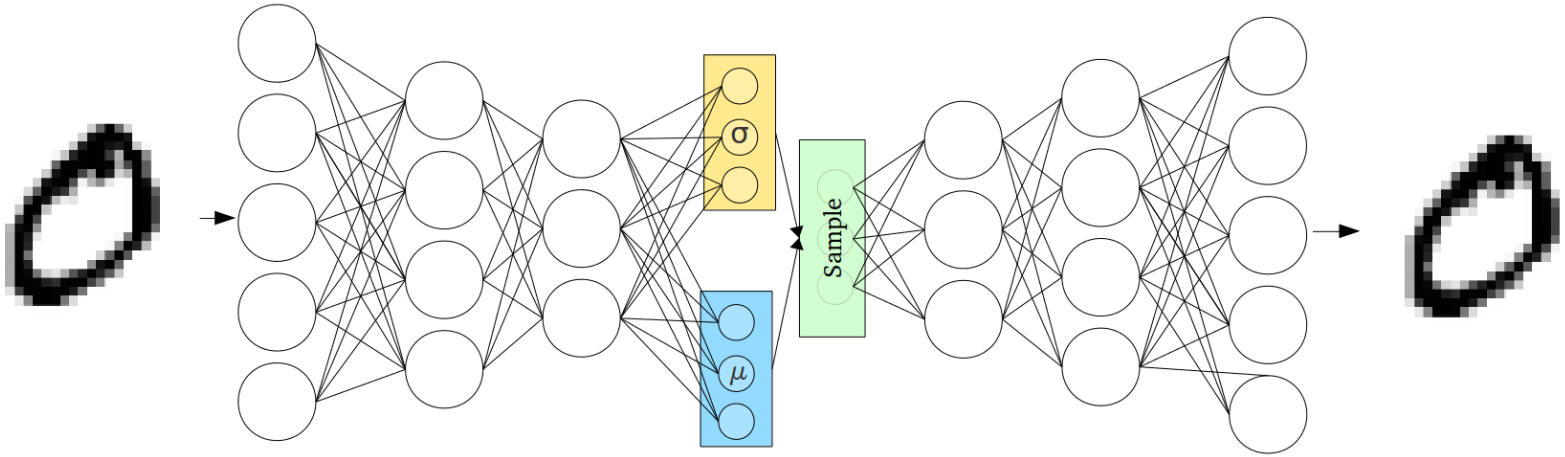






VAE - Unsupervised + Bayesian

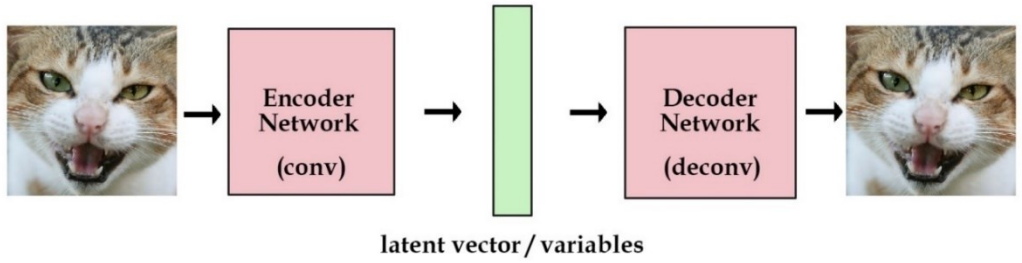
- VAE combines the power of unsupervised deep learning and Bayesian methods
- VAE incorporates regularization by explicitly learning the joint distribution $P(X, z)$ over data X, and a set of latent variables z that is most compatible with observed data points and some designated prior distribution over latent space



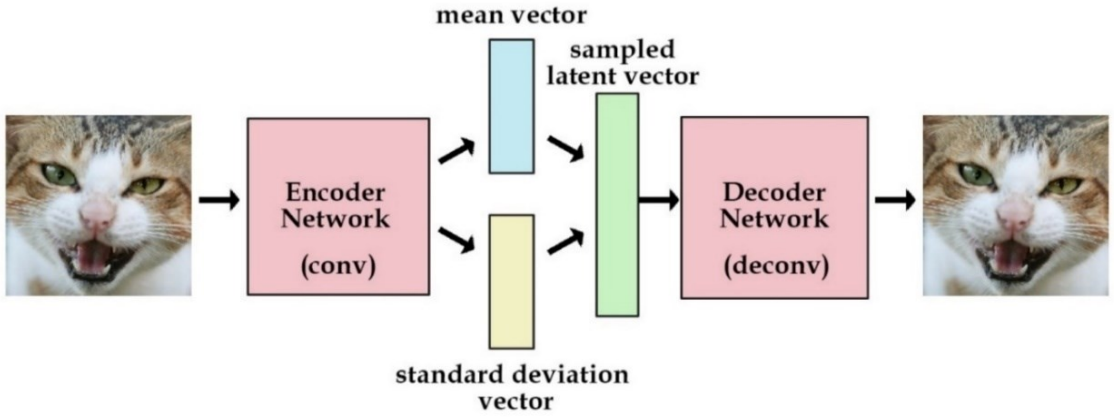
https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf

Autoencoder vs Variational Autoencoder

- Standard autoencoders learn to generate compact representations and reconstruct their inputs well.
- However, for data generation, the latent space they convert their inputs to and where their encoded vectors lie, may not be continuous or allow easy interpolation.



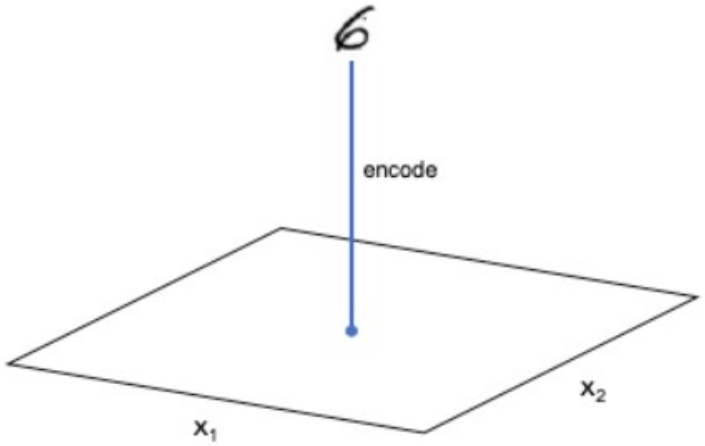
- In Variational Autoencoders (VAEs) their latent spaces are, *by design,* continuous, allowing easy random sampling and interpolation which makes them very useful for generative modeling.



http://kvfrans.com/variational-autoencoders-explained/

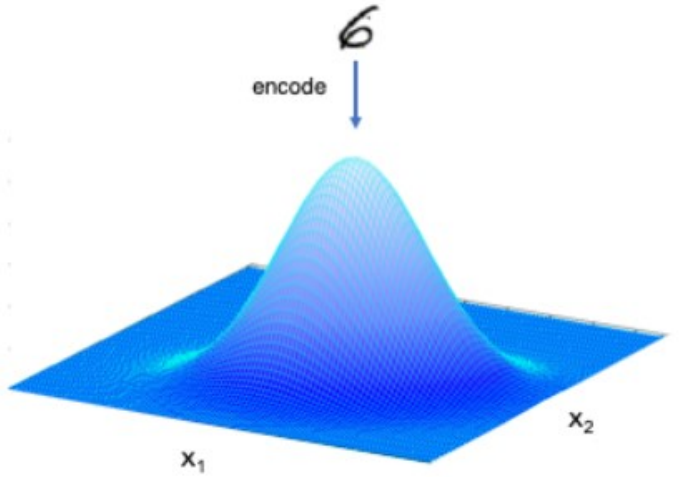
Autoencoder vs Variational Autoencoder

- In an autoencoder, each image is mapped directly to one point in the latent space.



autoencoder

- In a variational autoencoder, each image is instead mapped to a multivariate normal distribution around a point in the latent space

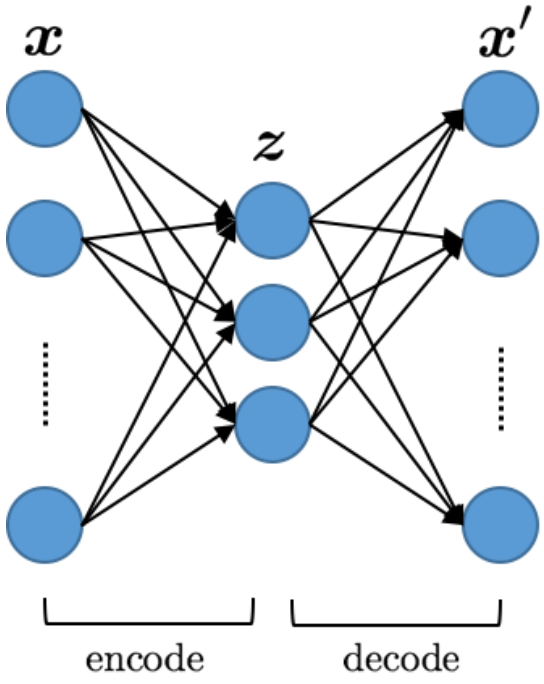


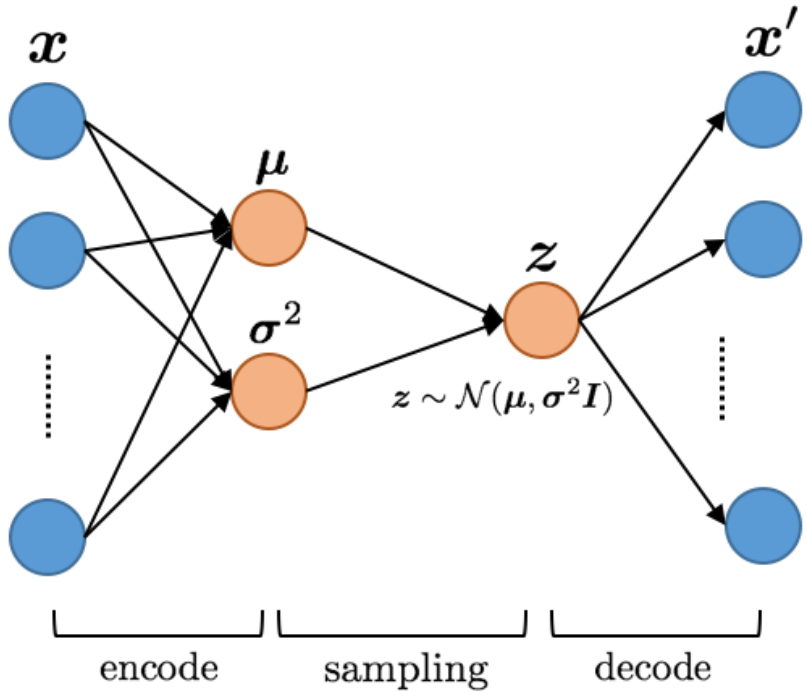
variational autoencoder

Autoencoder vs Variational Autoencoder

- We can sample a point z from a normal distribution with mean μ and standard deviation σ using the following equation:

  $z = \mu + \sigma\epsilon$, where ϵ is sampled from a standard normal distribution.





https://www.renom.jp/notebooks/tutorial/generative-model/VAE/notebook.html

VAE - Loss Function

VAEs have two separate losses:

- **Generative loss**, which is a mean squared error that measures how accurately the network reconstructed the images
- **Latent loss**, which is the KL divergence that measures how closely the latent variables match a unit gaussian

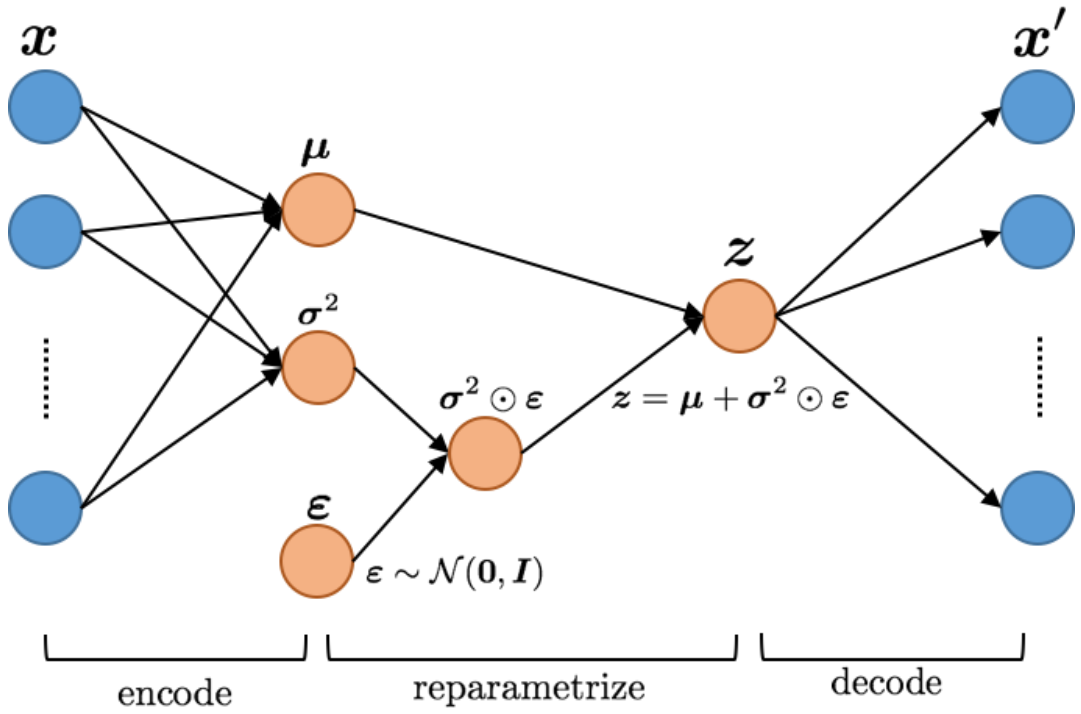
```
generation_loss = mean(square(generated_image - real_image))
latent_loss = KL-Divergence(latent_variable, unit_gaussian)
loss = generation_loss + latent_loss
```

If we add a constraint on the encoding network, that forces it to generate latent vectors that roughly follow a *unit gaussian distribution*. In practice, the tradeoff is between how accurate our network can be and how close its latent variables can match the unit gaussian distribution.

http://kvfrans.com/variational-autoencoders-explained/

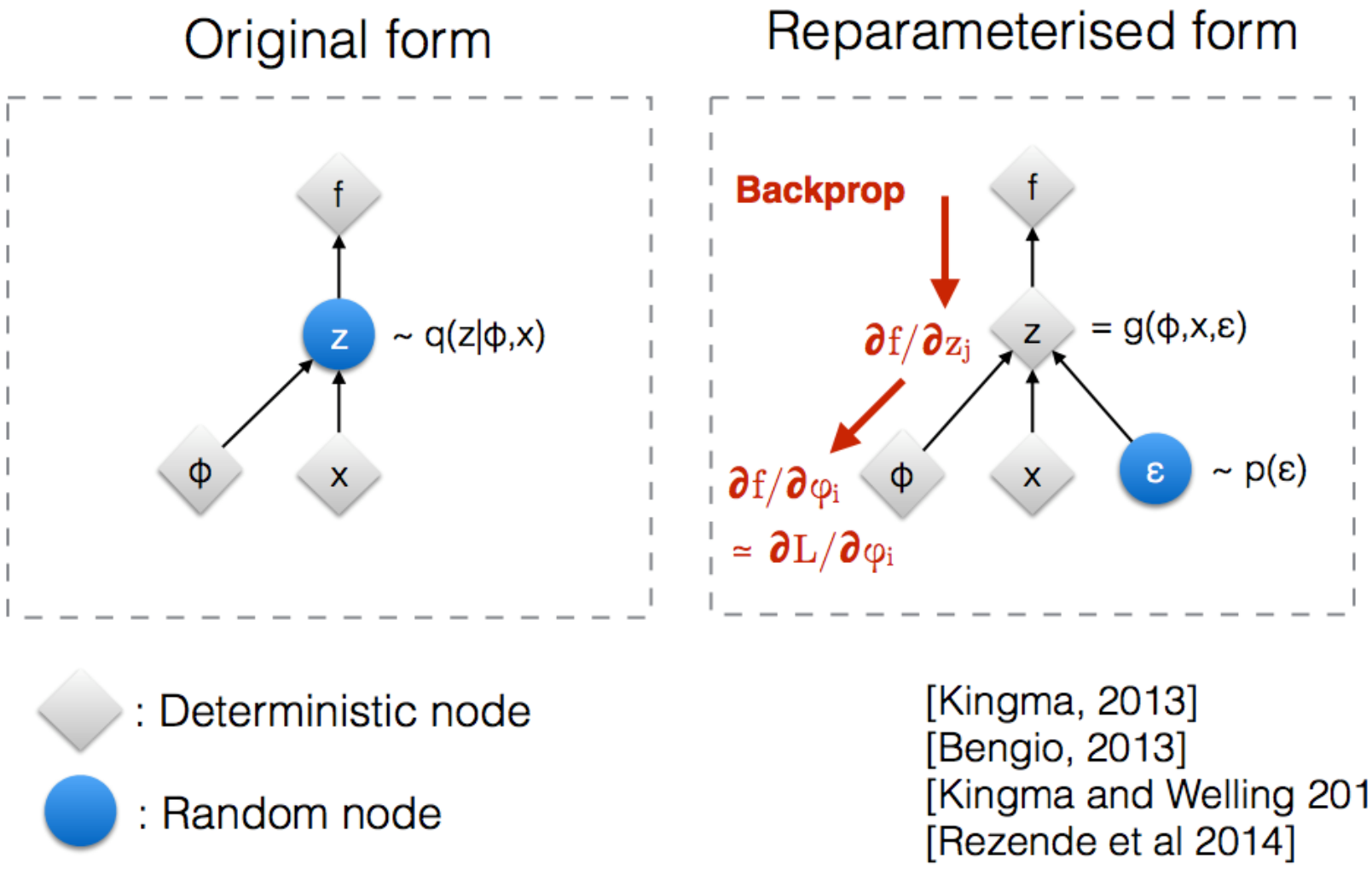
VAE Reparameterization Trick

- Problem:
  - VAEs sample from a random node z which is approximated by the parametric model q(z|ϕ,x) of the true posterior. Backprop cannot flow through a random node.
- Solution:
  - Introducing a new parameter **ϵ** that allows us to re-parameterize z in a way that allows backpropagation to flow through the deterministic nodes.
  - In order to optimize the KL divergence, instead of the encoder generating a vector of real values, it will generate a vector of means and a vector of standard deviations



http://kvfrans.com/variational-autoencoders-explained/

VAE Reparameterization Trick

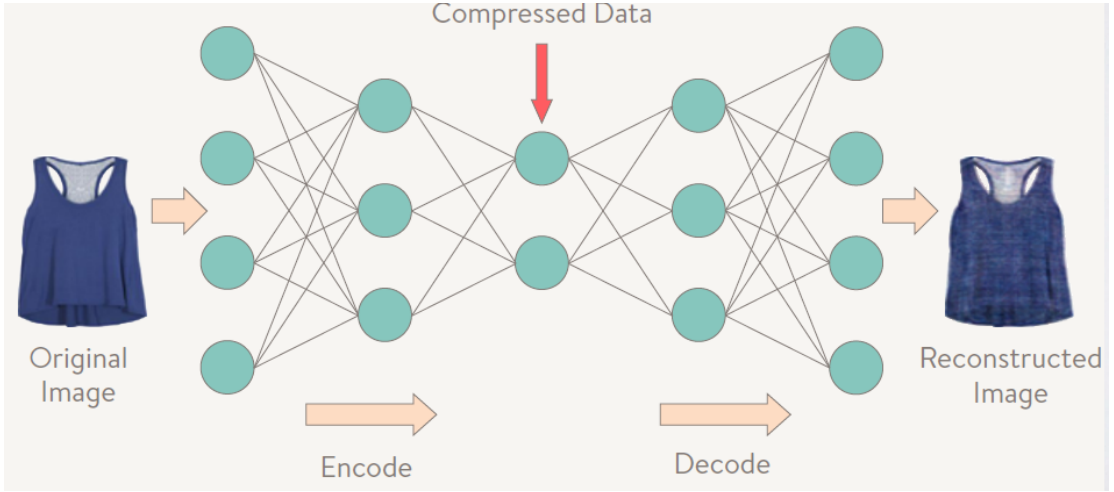


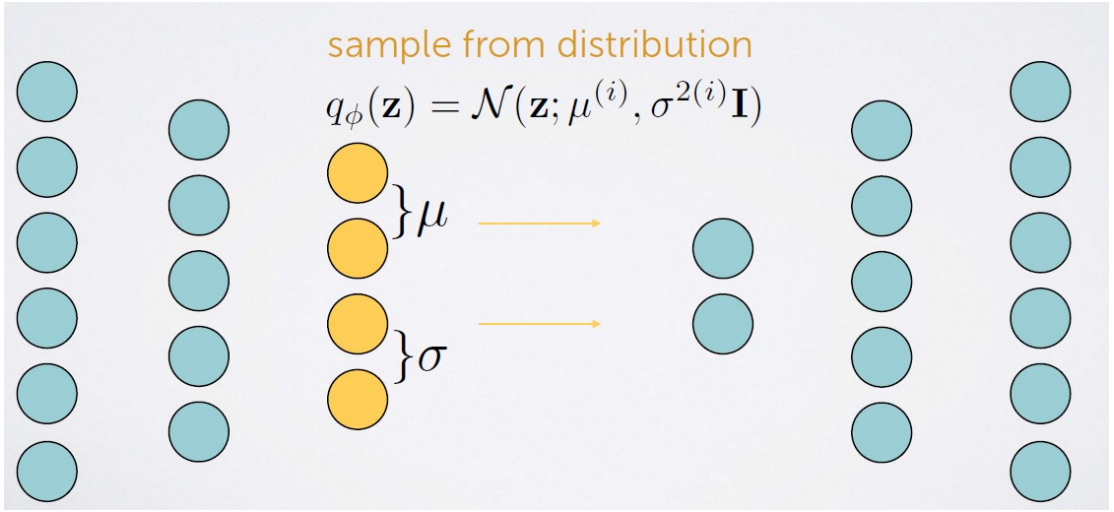
[Kingma, 2013]
[Bengio, 2013]
[Kingma and Welling 2014]
[Rezende et al 2014]

https://stats.stackexchange.com/questions/199605/how-does-the-reparameterization-trick-for-vaes-work-and-why-is-it-important

VAE Sample Application

- Problem:
  - No reliable system of style labels for image data
- Training:
  1. Auto Encoder - Backpropagation
  2. Loss - MSE of original data to reconstructed data
- Issues:
  1. AEs often overfit unless amount of training data is large.
  2. Gradients diminish quickly, thus weight corrections small “far away” from output.
- Solution:
  1. Use variational component to “regularize” training.
  2. Stack auto-encoders and train greedily (DBN)



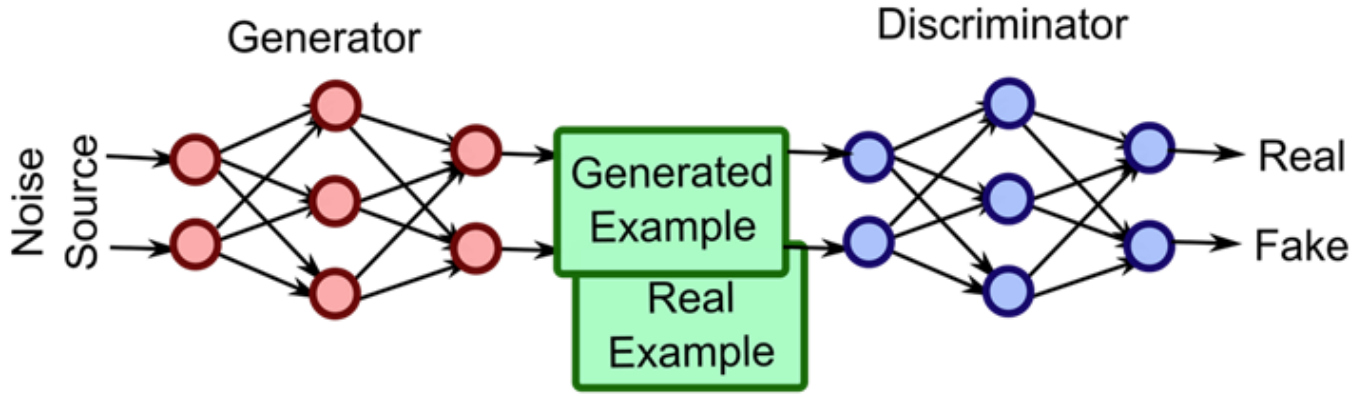


https://multithreaded.stitchfix.com/blog/2015/09/17/deep-style/

GENERATIVE ADVERSARIAL NETWORKS (GAN)

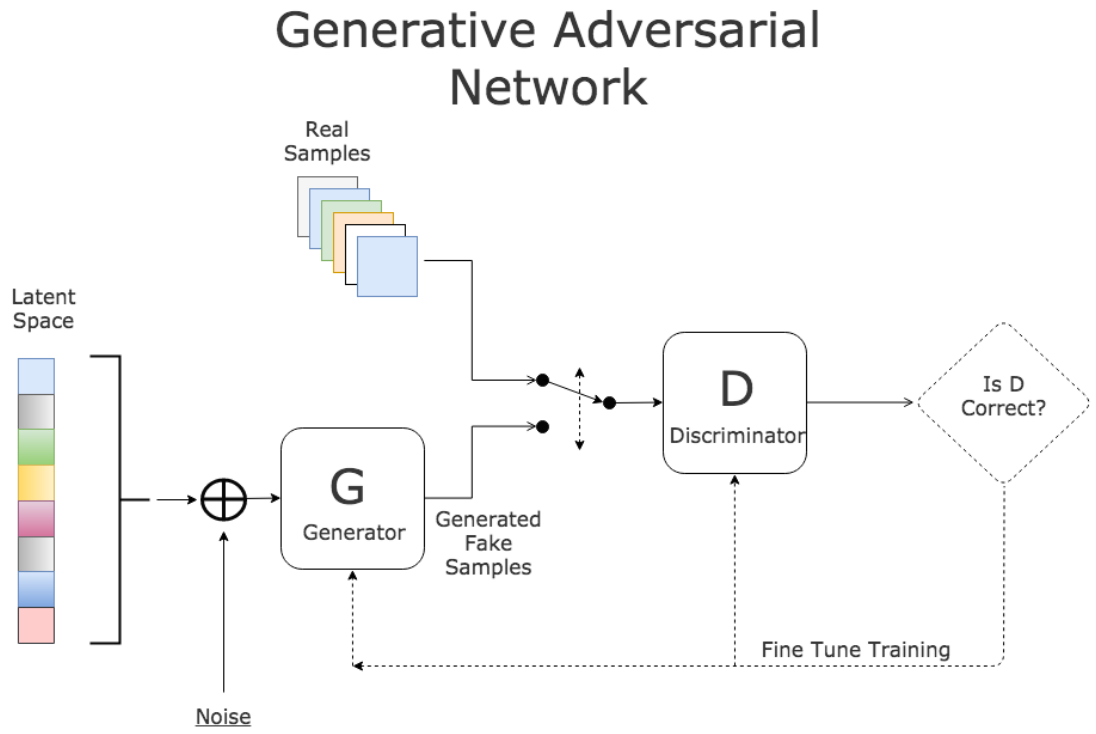
Generative Adversarial Networks (GAN)

- Problem:
  - We would like to sample from a complex, high-dimensional training distribution. But there is no direct way to do this.
- Solution
  - Approach - Instead of an explicit density function, GANs take game-theory approach i.e. learn to generate from training distribution through a two-player minimax game
  - Sample from a simple distribution, e.g. random noise. Learn transformation to training distribution using Neural Networks.



GAN Design

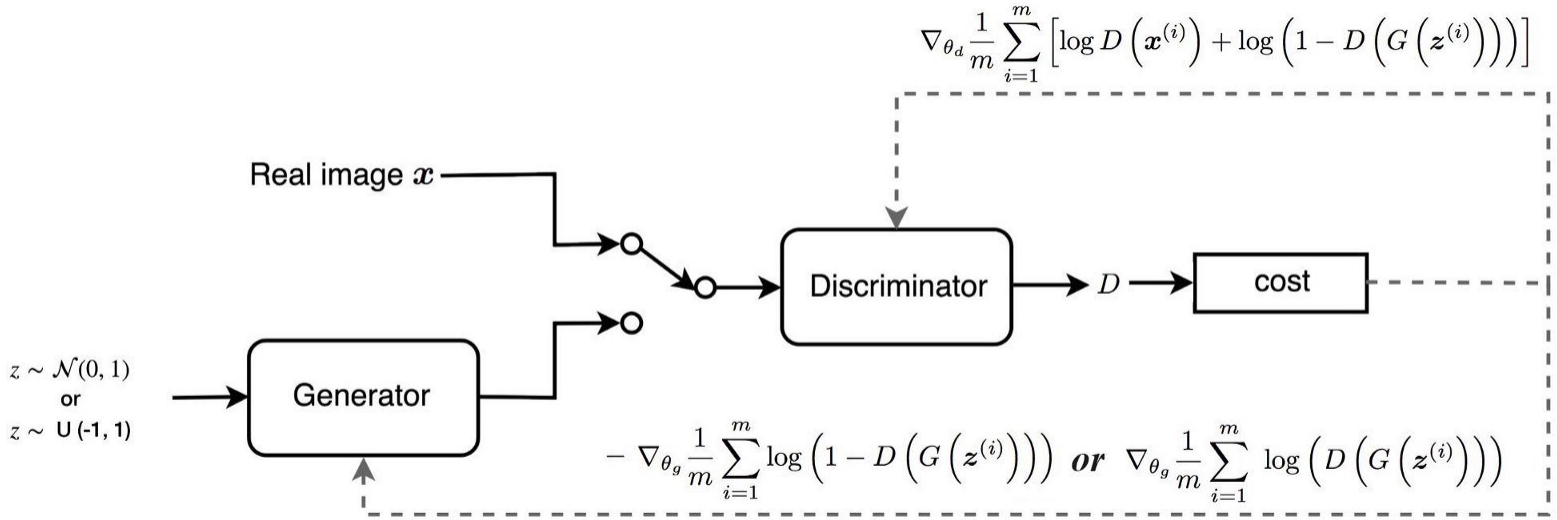
- GANs contain two independent feedforward networks that act as adversaries
- The Generator (G) network is tasked to generate random samples that resemble real samples and tries to fool the discriminator
- The Discriminator (D) network acts as classifier – tries to distinguish between real and fake images



https://arxiv.org/abs/1406.2661

GAN Training

- Generator takes as input a vector of random numbers (*z*), and transforms it into the form of the data we are interested in imitating.
- Discriminator takes as input a set of data, either real (*x*) or generated (*G(z)),* and produces a probability of that data being real (*P(x)*).
- Discriminator is optimized in order to increase the likelihood of giving a high probability to the real data and a low probability to the generated data
- Generator is then optimized in order to increase the probability of the generated data being rated highly



https://arxiv.org/abs/1406.2661

GAN Cost Function

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Discriminator output for real data x

Discriminator output for generated fake data G(z)

- Discriminator (θ_d) wants to **maximize** objective such that D(x) is close to 1 (real) and D(G(z)) is close to 0 (fake)
- Generator (θ_g) wants to **minimize** objective such that D(G(z)) is close to 1 (discriminator is fooled into thinking generated G(z) is real)

GAN Cost Function

Alternate between:

1. *Gradient ascent on Discriminator*

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

The **gradient ascent** expression for the discriminator. The first term corresponds to optimizing the probability that the real data (x) is rated highly. The second term corresponds to optimizing the probability that the generated data G(z) is rated poorly. Notice we apply the gradient to the **discriminator**, not the generator.

2. *Gradient descent on Generator*

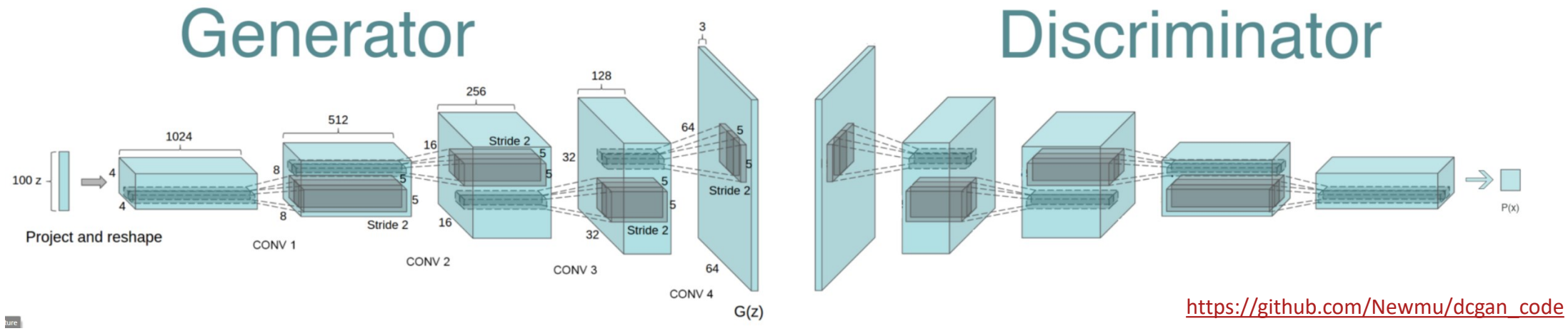
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

The **gradient descent** expression for the generator. The term corresponds to optimizing the probability that the generated data G(z) is rated highly. Notice we apply the gradient to the **generator** network, not the discriminator.

https://arxiv.org/abs/1406.2661

Deep Convolutional GAN (DCGAN)

By alternating gradient optimization between the two networks using these expressions on new batches of real and generated data each time, the GAN will slowly converge to producing data that is as realistic as the network is capable of modeling

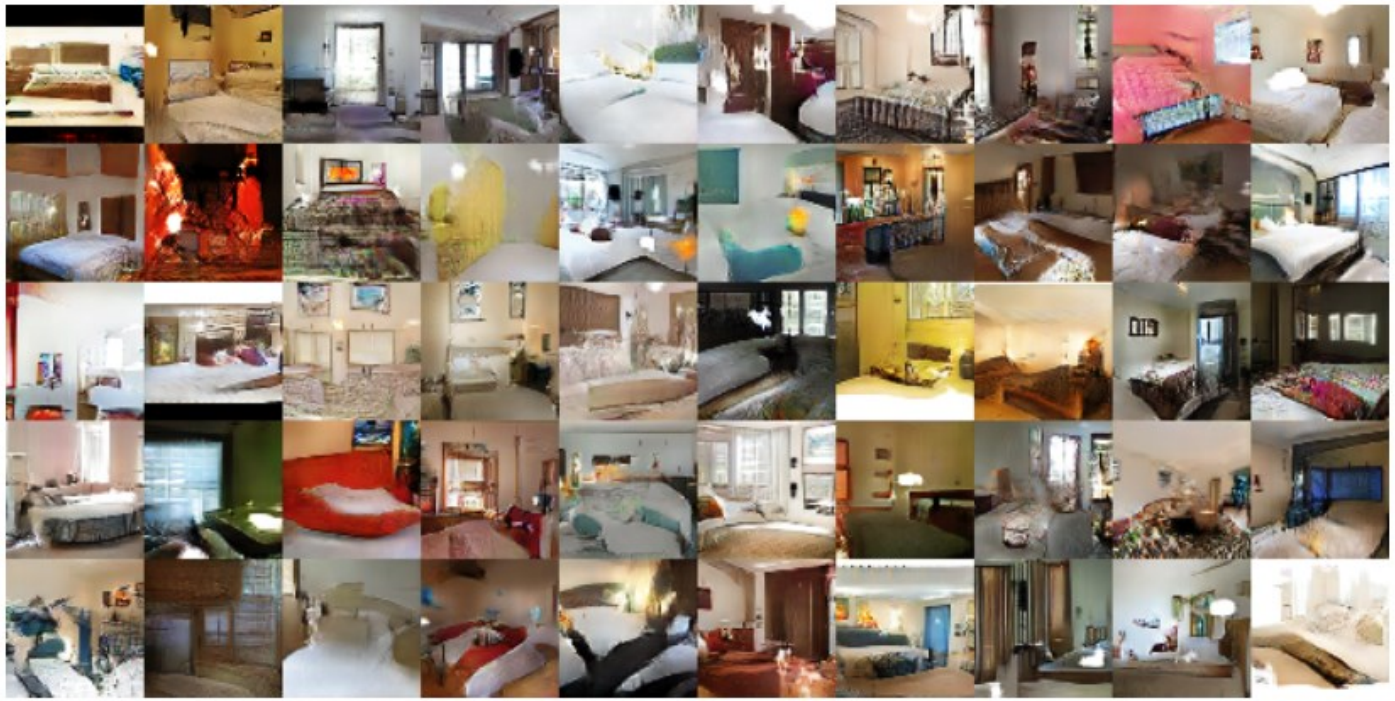


https://github.com/Newmu/dcgan_code

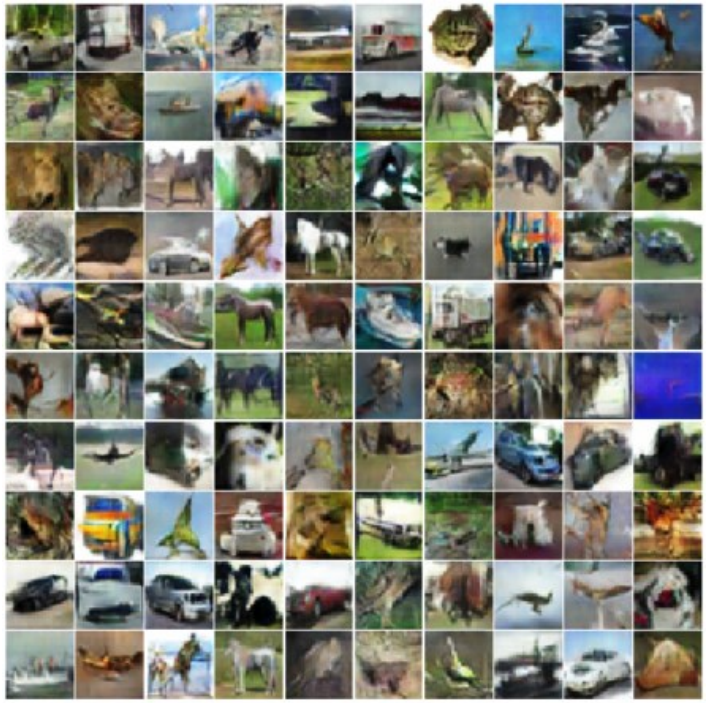
Example architecture for generator and discriminator networks. Both utilize convolutional layers to process visual information. Modified from http://arxiv.org/abs/1511.06434

GAN Applications

- GANs have been primarily applied to modeling natural images
- Now generating images that are significantly sharper than those trained using other leading generative methods based on maximum likelihood training objectives



Generated bedrooms. Source: “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks” https://arxiv.org/abs/1511.06434v2



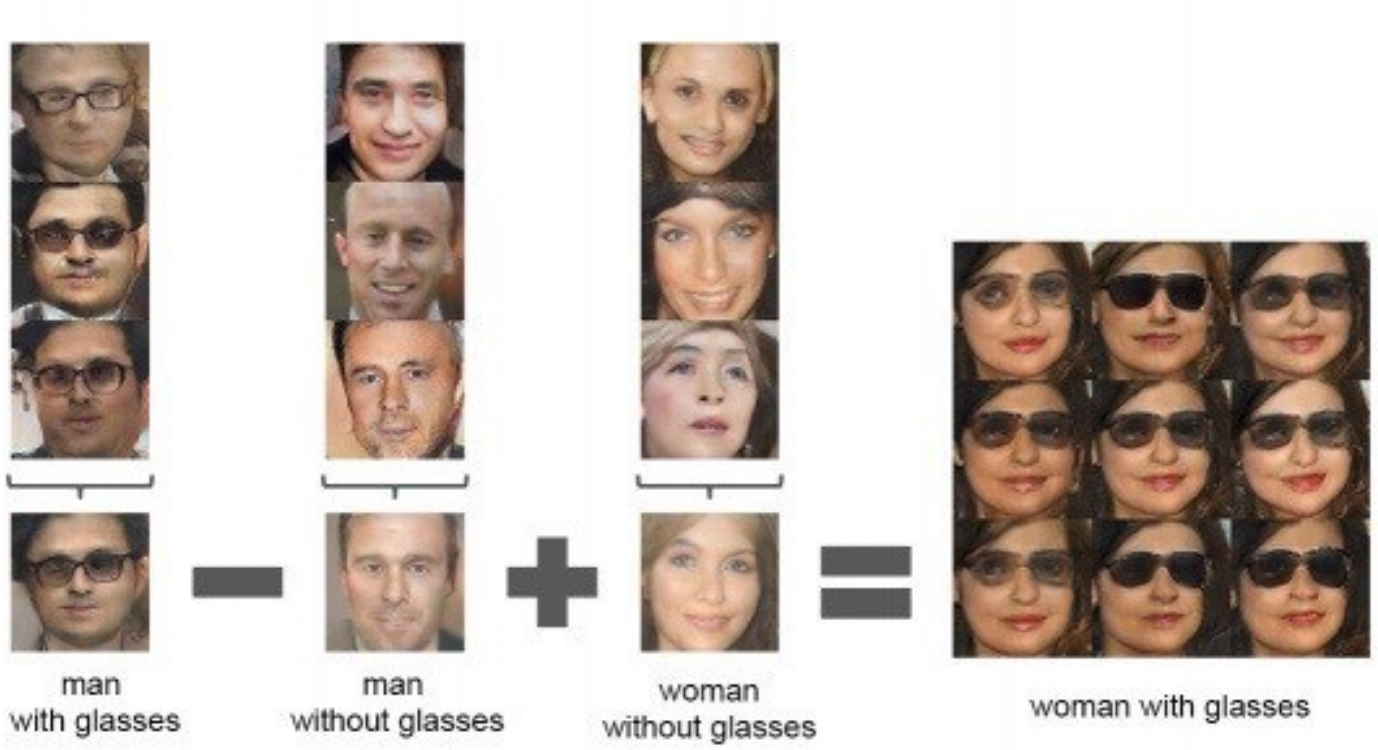
Generated CIFAR-10 samples. Source: “Improved Techniques for Training GANs” https://arxiv.org/abs/1606.03498

Style Transfer CNN vs GAN


https://github.com/junyanz/CycleGAN

Image Arithmetic

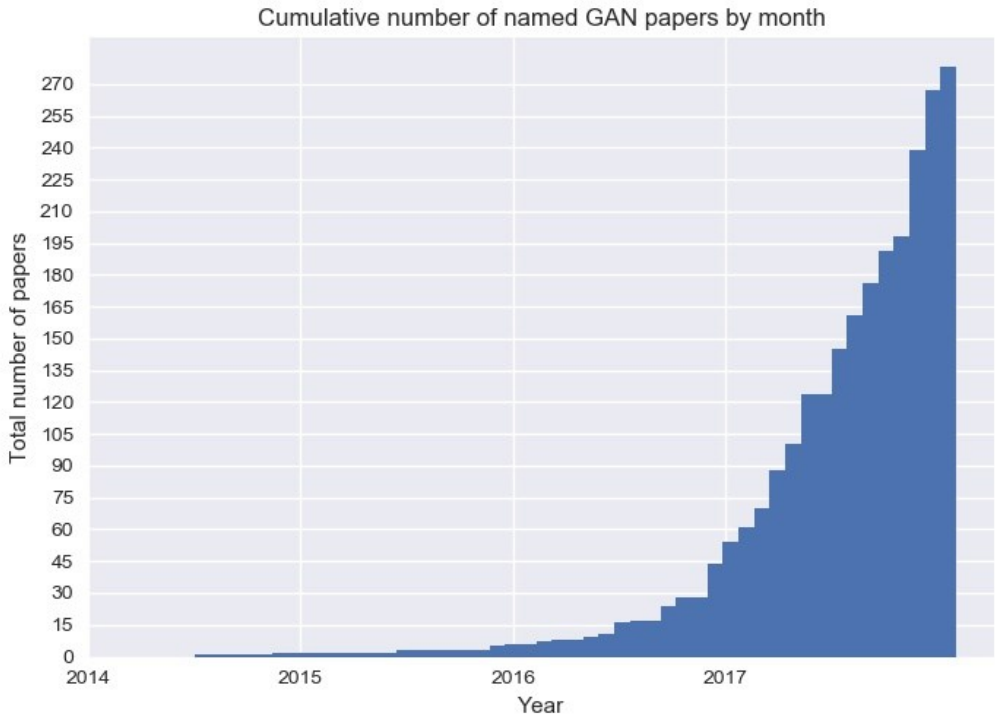
- GANs have been applied to performing arithmetic operations on images



Research

GANs are currently a major research area in Deep learning

3D-ED-GAN - Shape Inpainting using 3D Generative Adversarial Network and Recurrent Cor
3D-GAN - Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversa
3D-IWGAN - Improved Adversarial Systems for 3D Object Generation and Reconstruction (g
3D-RecGAN - 3D Object Reconstruction from a Single Depth View with Adversarial Learning
ABC-GAN - ABC-GAN: Adaptive Blur and Control for improved training stability of Generati
(github)
ABC-GAN - GANs for LIFE: Generative Adversarial Networks for Likelihood Free Inference
AC-GAN - Conditional Image Synthesis With Auxiliary Classifier GANs
acGAN - Face Aging With Conditional Generative Adversarial Networks
ACtuAL - ACtuAL: Actor-Critic Under Adversarial Learning
AdaGAN - AdaGAN: Boosting Generative Models
AdvGAN - Generating adversarial examples with adversarial networks
AE-GAN - AE-GAN: adversarial eliminating with GAN
AEGAN - Learning Inverse Mapping by Autoencoder based Generative Adversarial Nets
AffGAN - Amortised MAP Inference for Image Super-resolution
AL-CGAN - Learning to Generate Images of Outdoor Scenes from Attributes and Semantic L

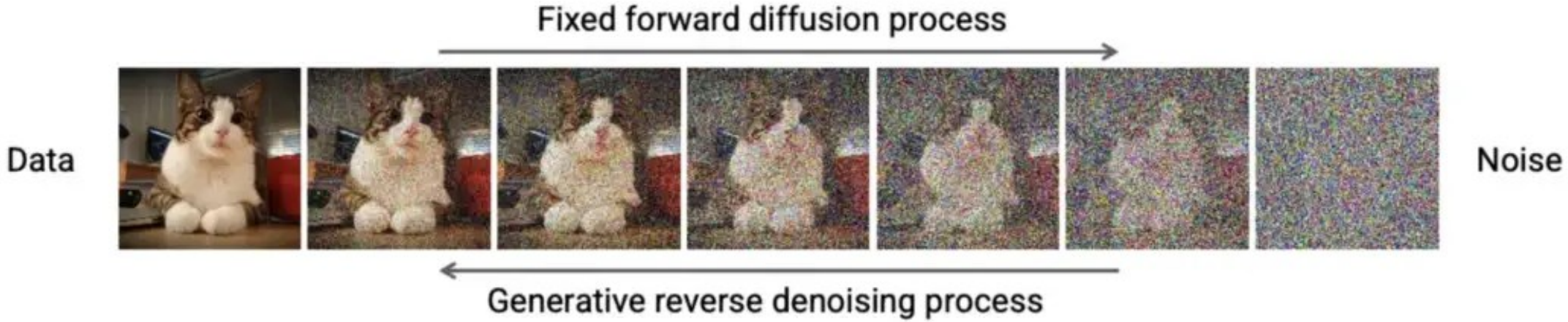


https://github.com/hindupuravinash

DIFFUSION MODELS

Diffusion Models

- If a model can learn the systematic decay of information due to noise, then it should be possible to reverse the process and therefore, recover the information back from the noise
- Diffusion Models work by destroying training data by adding Gaussian noise to it repeatedly, and then learning how to get the data back by reversing this process of adding noise



Forward Diffusion Process

- Forward diffusion process can be defined as a *Markov Chain* and therefore, unlike an encoder in the VAEs it doesn't require a training. Starting with the initial data point , we add Gaussian noise for $\boldsymbol{T}$ successive steps, and obtain a set of noisy samples.
- The prediction of probability density at time $\boldsymbol{t}$ is only dependent on the immediate predecessor at time $\boldsymbol{t-1}$ and therefore, the conditional probability density can be computed as follows

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \quad q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1})$$

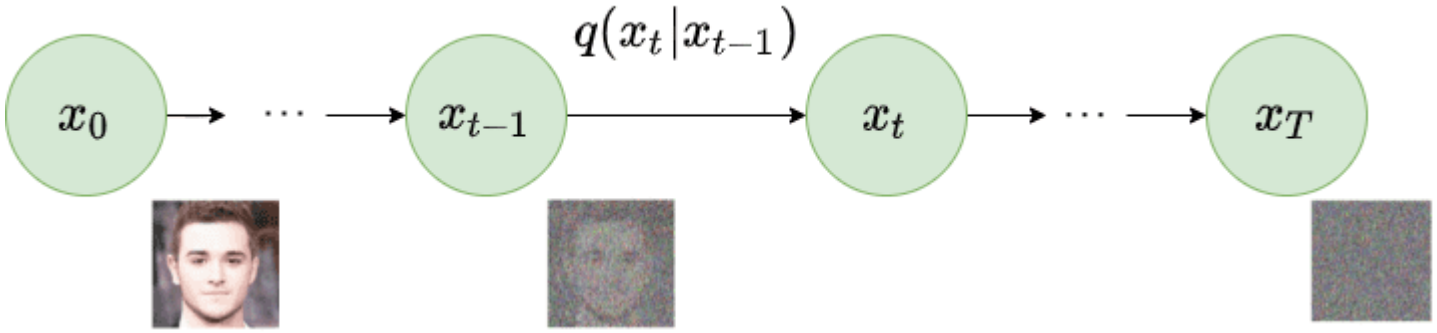
- Here, the mean and variance of the density function depends on a parameter β_t, which is a hyper parameter whose value can either be taken as a constant throughout the process or can be gradually changed in the successive steps.

Backward Diffusion Process (Reconstruction)

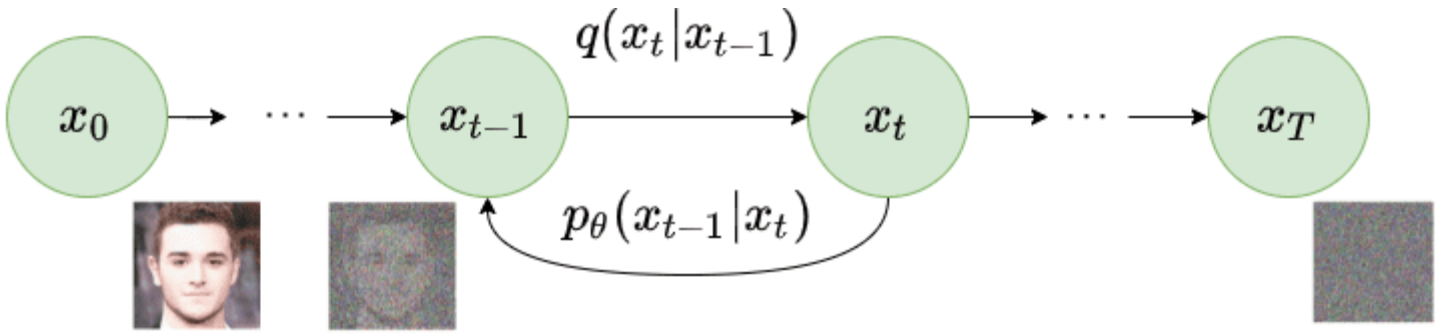
- Unlike forward diffusion, the reverse diffusion process requires training of a neural network model
- Requires the estimation of probability density at an earlier time step given the current state of the system. This means estimating the $\boldsymbol{q}(x_t - 1 \,|x_t)$ when $t = \boldsymbol{T}$ and thereby generating data sample from isotropic Gaussian noise.
- Therefore, we train a neural network model that estimates the $\mathrm{p}_\theta\ (x_t - 1 \,|x_t)$ based on learned weights $\boldsymbol{\theta}$ and the current state at time $\boldsymbol{t}$.
- Loss Function:

$$K = -\mathbb{E}_q[D_{KL}(q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \| p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)) + H_q(\mathbf{X}_T|\mathbf{X}_0) - H_q(\mathbf{X}_1|\mathbf{X}_0) - H_p(\mathbf{X}_T)]$$

Forward and Backward Diffusion

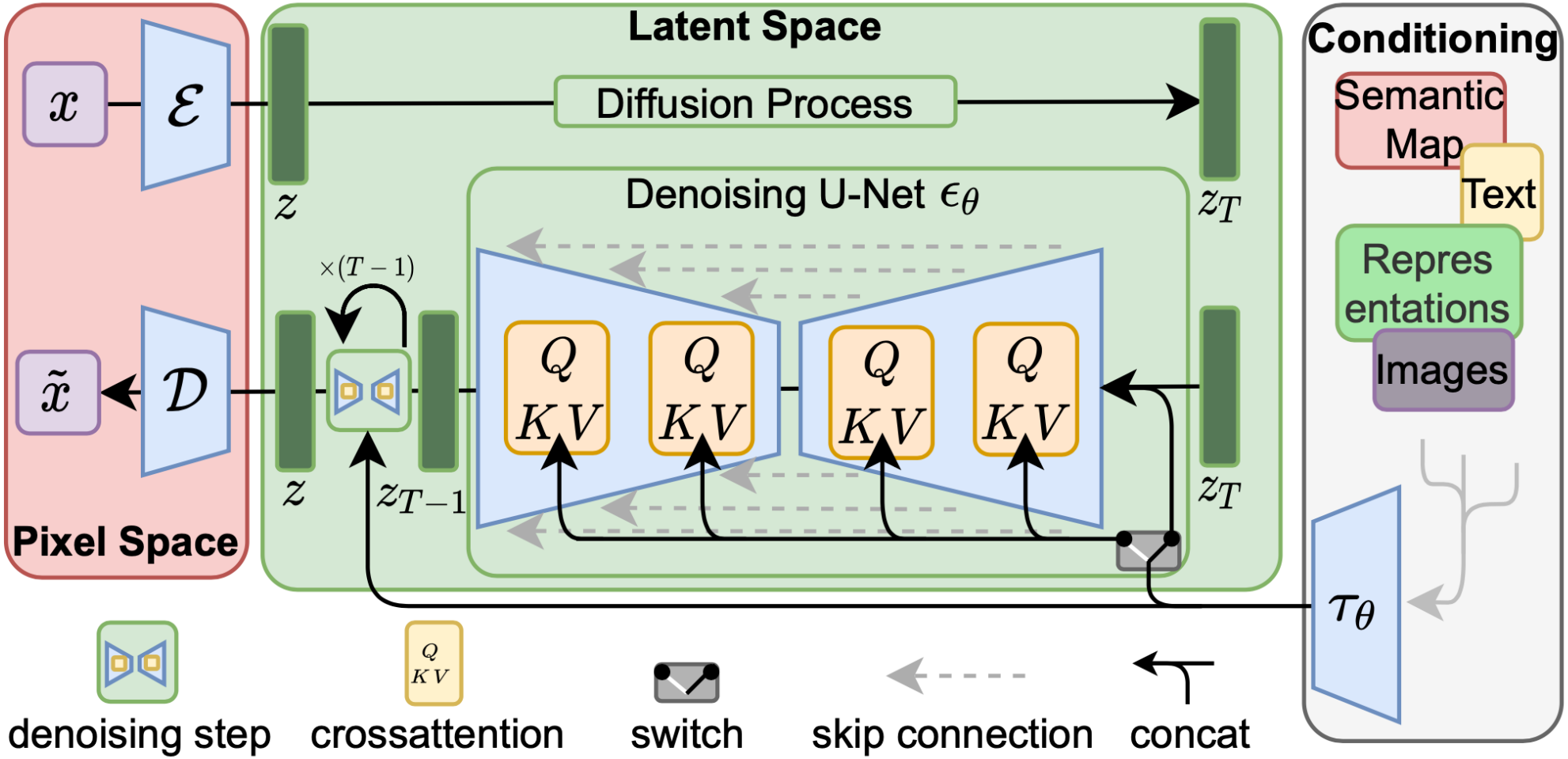


Forward diffusion process

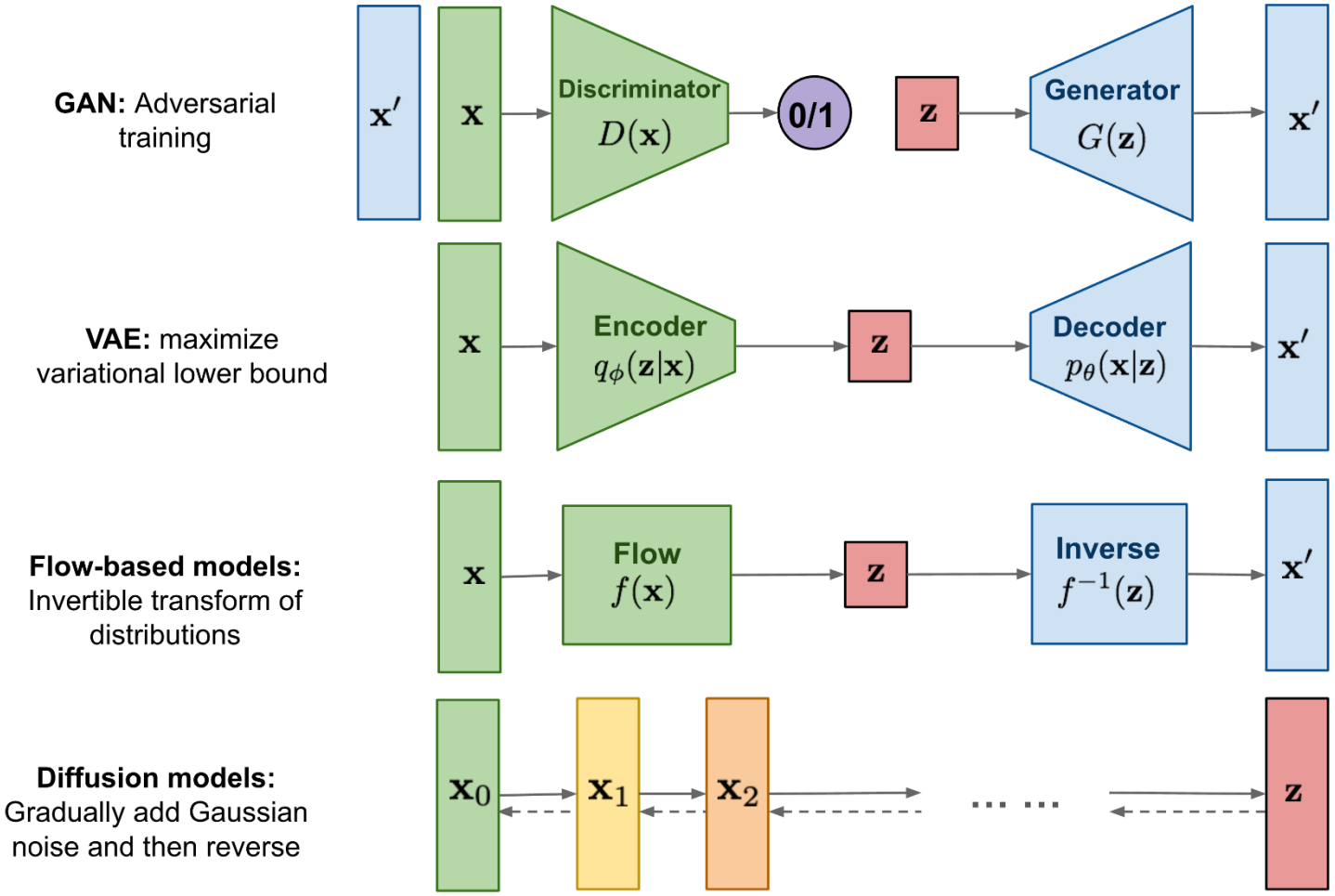


Reverse diffusion process

Denoising Diffusion Architecture



Summary



https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Lab

1. Autoencoder
2. Denoising Autoencoder
3. Variational Autoencoder
4. DCGAN