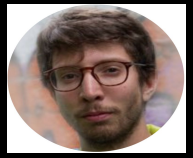


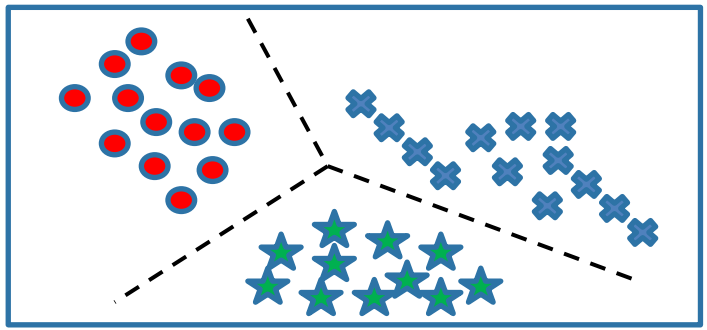
Deep generative methods



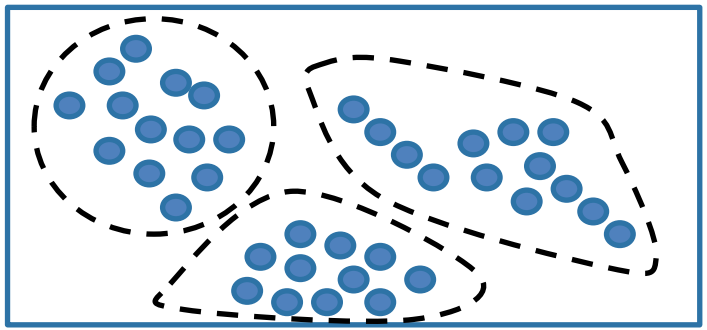
Stéphane Lathuilière
Télécom Paris

Types of Learning

- **Supervised (inductive)** learning
  - Given: training data + desired outputs (labels)
- **Unsupervised** learning
  - Given: training data (without desired outputs)



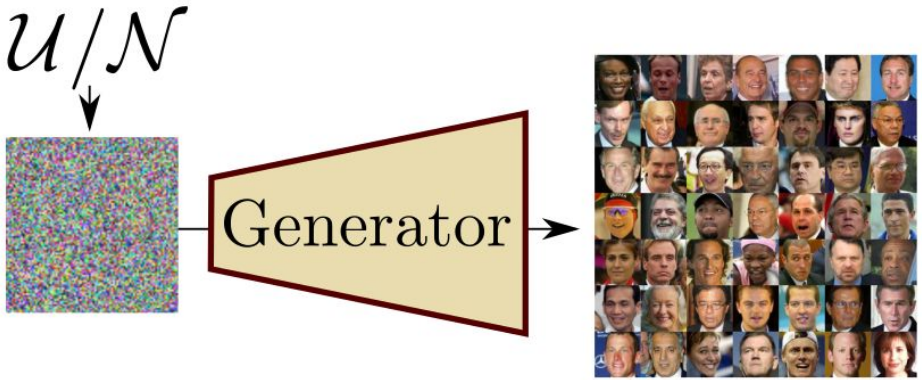
Supervised learning



Unsupervised learning

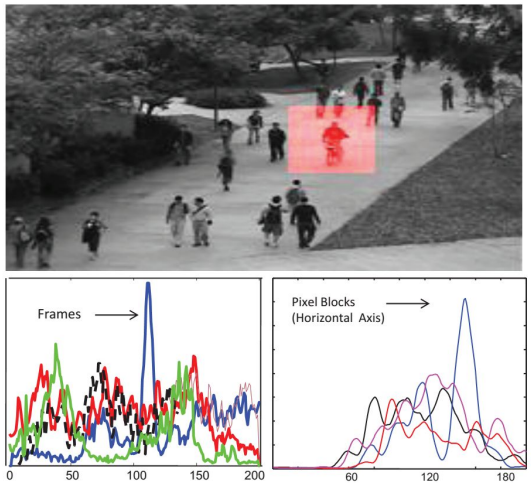
Unsupervised Learning

- Sampling: learn to sample according to $p(\boldsymbol{x})$: $R^d \rightarrow R$



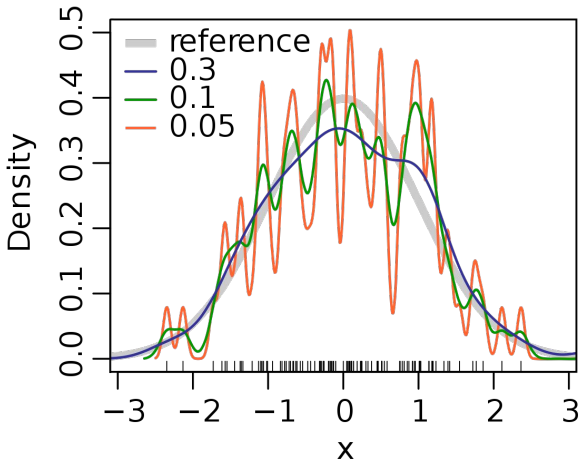
Unsupervised Learning

- **Anomaly detection:** the computer program sifts through a set of events or objects and flags some of them as being unusual or atypical.
- Example: credit card fraud detection, video surveillance.



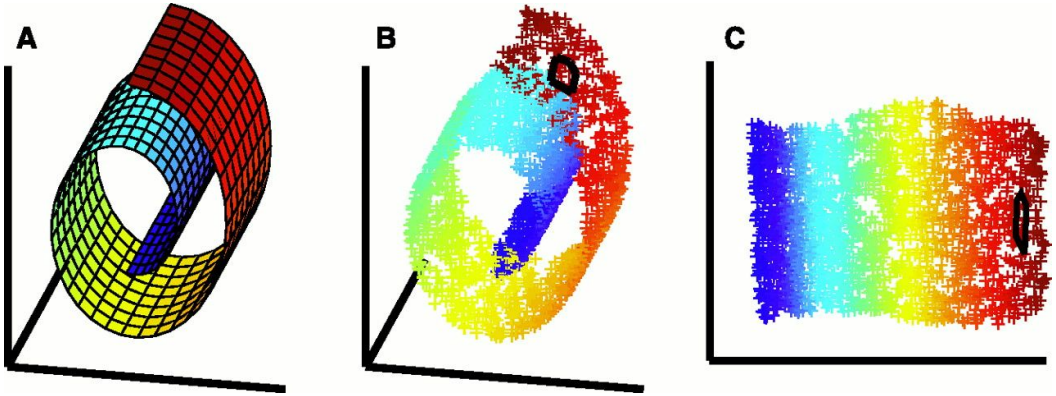
Unsupervised Learning

- Density Estimation: learn a function $p(\boldsymbol{x})$: $R^d \rightarrow R$



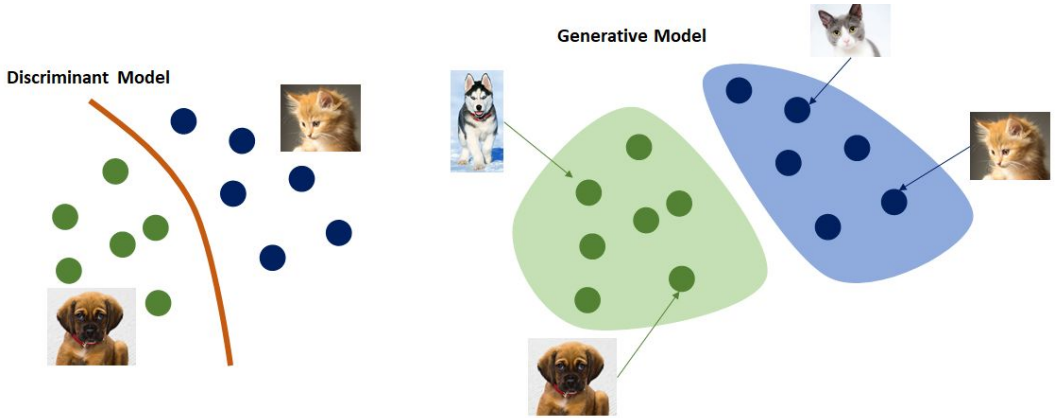
Unsupervised Learning

- Dimensionality Reduction



Supervised Learning: Classification

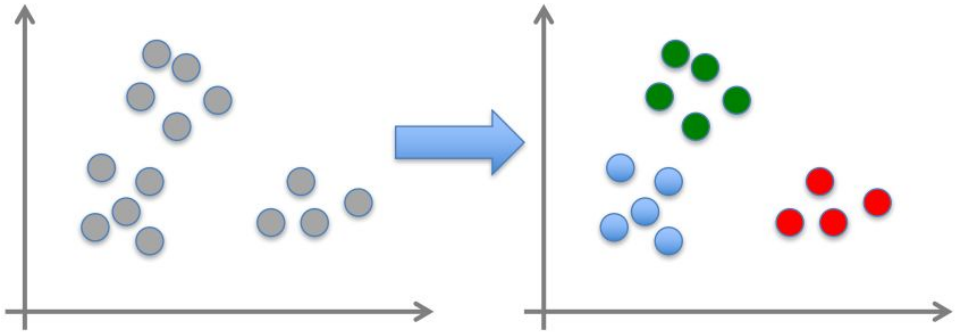
Discriminative vs Generative Algorithms



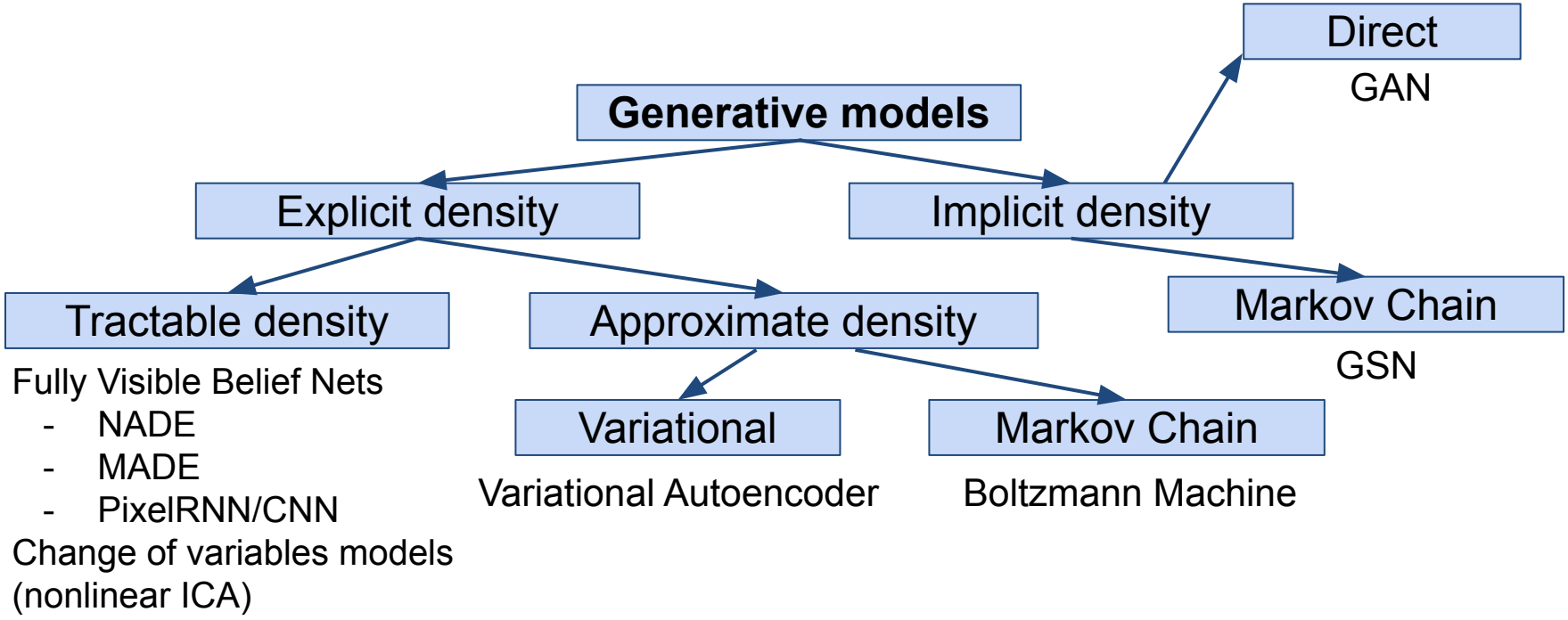
About Generative and Discriminative models - Jordi EsteveSorribas,medium

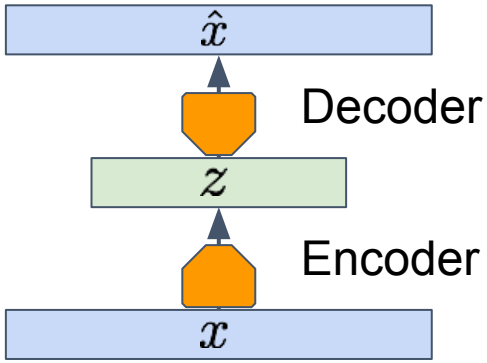
Unsupervised Learning: Clustering

- Given $\mathcal{T} = \{\mathbf{x}_1, \cdots, \mathbf{x}_m\}$ (without labels)
  output hidden structure behind the $\mathbf{x}$'s – e.g., clustering



Taxonomy of Generative Models

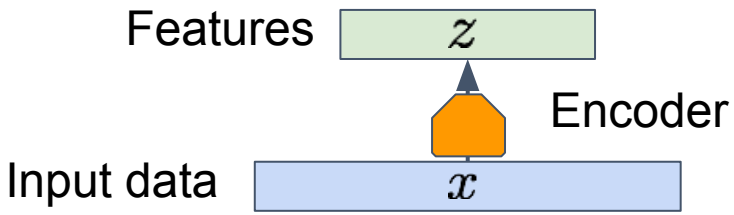




Autoencoders

Autoencoders

Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data



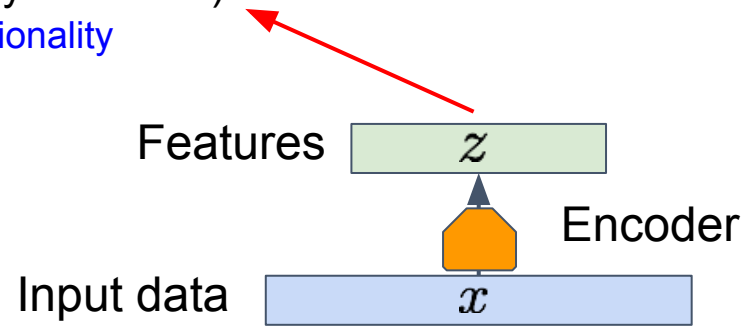
Auto-association by multilayer perceptrons and singular value decomposition, H Bourlard, Y Kamp - Biological cybernetics, 1988

Autoencoders

Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

z usually smaller than **x**
(dimensionality reduction)

Q: Why dimensionality reduction?



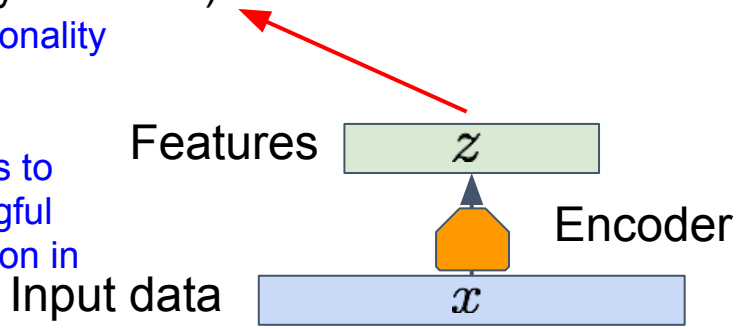
Autoencoders

Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

z usually smaller than **x**
(dimensionality reduction)

Q: Why dimensionality reduction?

A: Want features to capture meaningful factors of variation in data



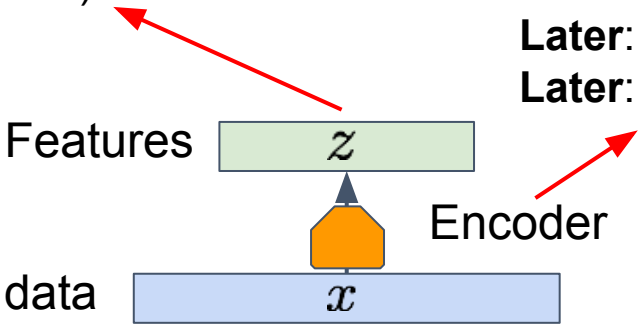
Autoencoders

Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

z usually smaller than **x** (dimensionality reduction)

Q: Why dimensionality reduction?

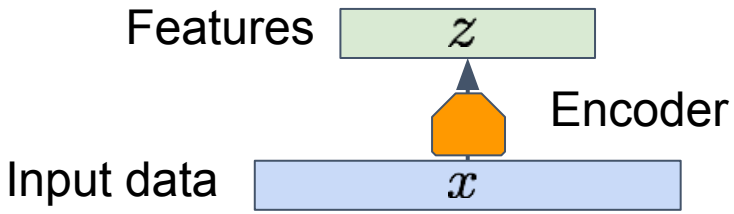
A: Want features to capture meaningful factors of variation in data



Originally: Linear + nonlinearity (sigmoid)
Later: Deep, fully-connected
Later: ReLU CNN

Autoencoders

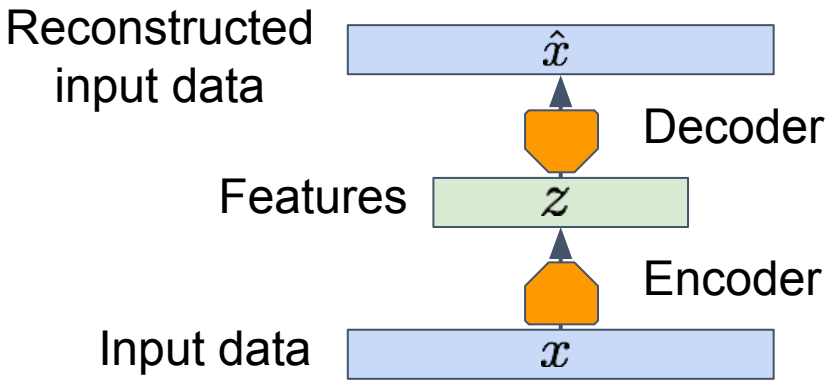
How to learn this feature representation?



Autoencoders

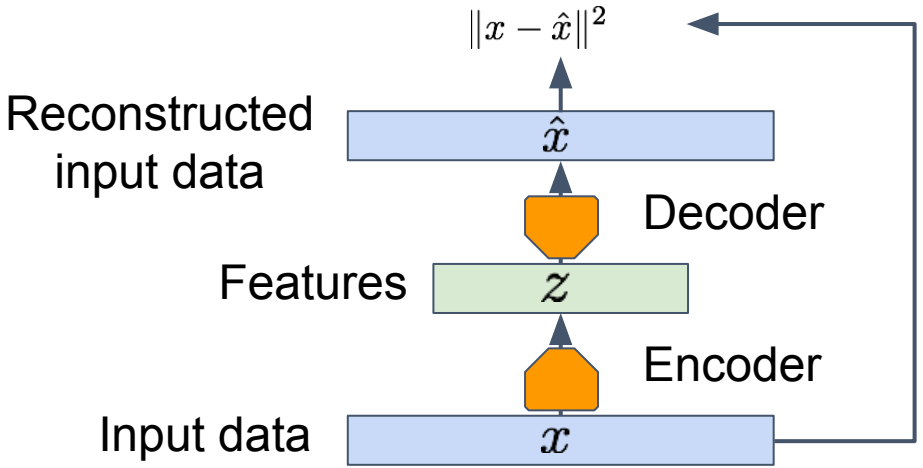
Train such that features can be used to reconstruct original data

“Autoencoding” - encoding itself

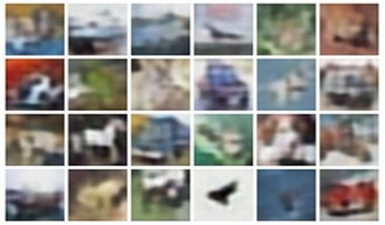


Autoencoders

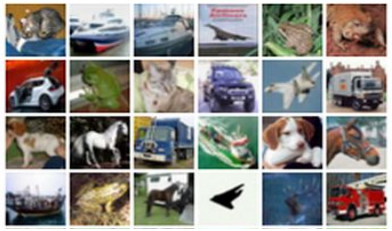
Train such that features can be used to reconstruct original data



Reconstructed data



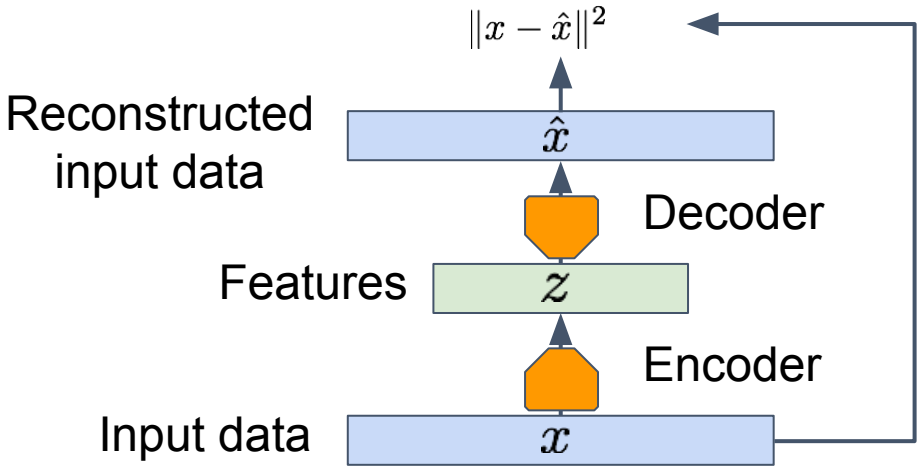
Input data



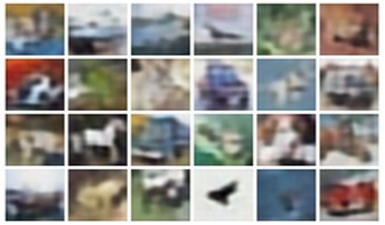
Autoencoders

Train such that features can be used to reconstruct original data

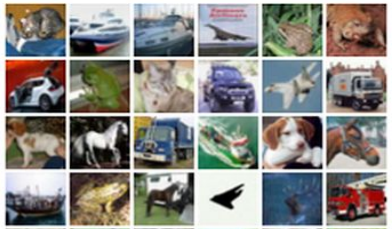
Doesn't use labels!



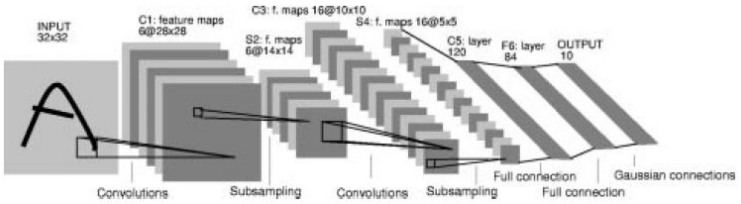
Reconstructed data



Input data

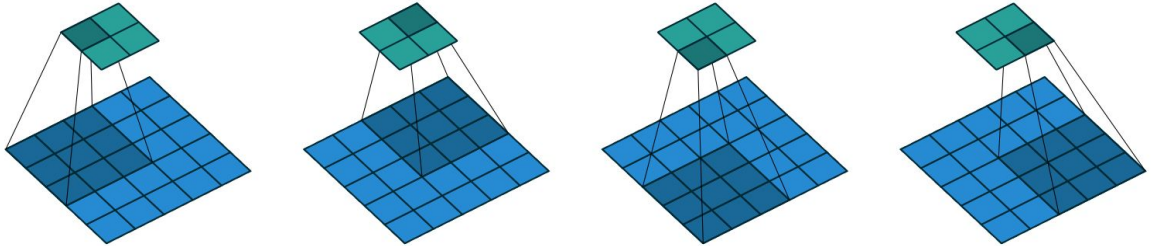


Autoencoders



Architecture:

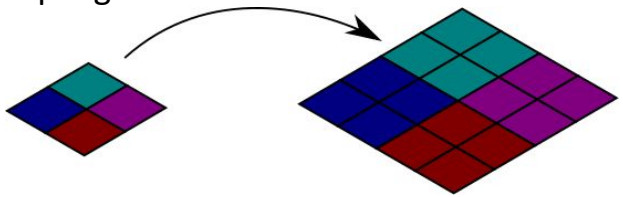
- Encoder: ConvNet architecture



Convolving a 3 × 3 kernel over a 5 × 5 input using 2 × 2 strides

- Decoder: How to go from a vector to an image? How to “deconvolve”?

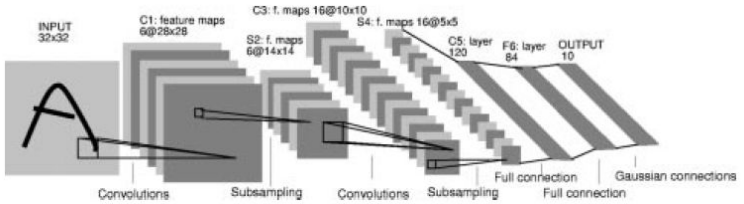
Up-sampling

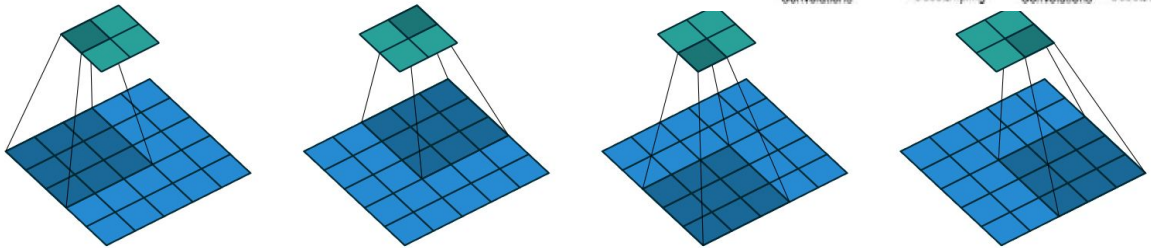


Autoencoders

Architecture:

- Encoder: ConvNet architecture

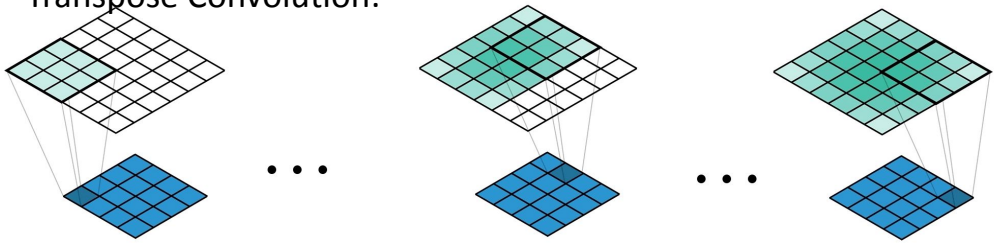




Convolving a 3 × 3 kernel over a 5 × 5 input using 2 × 2 strides

- Decoder: How to go from a vector to an image? How to “deconvolve”?

Transpose Convolution:

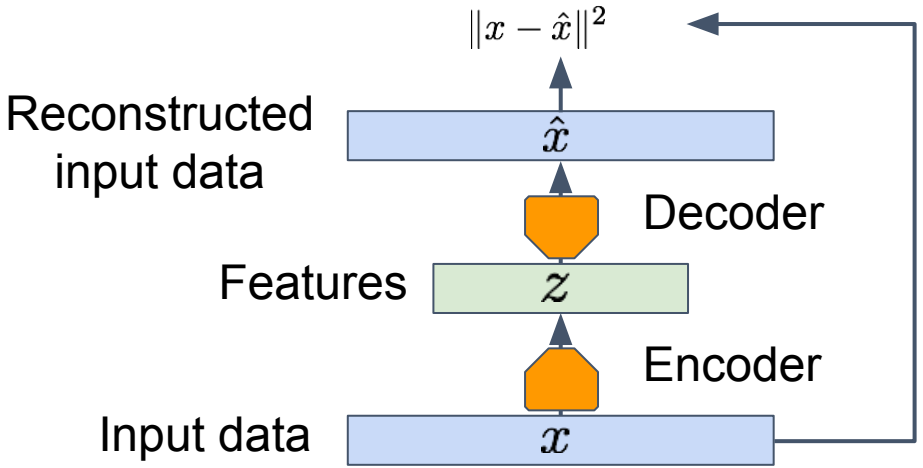


A guide to convolution arithmetic for deep learning ,Vincent Dumoulin, Francesco Visin

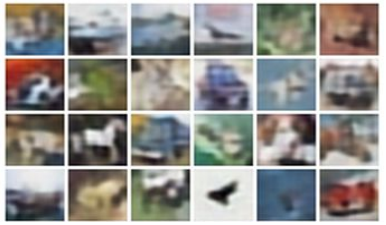
Autoencoders

Train such that features can be used to reconstruct original data

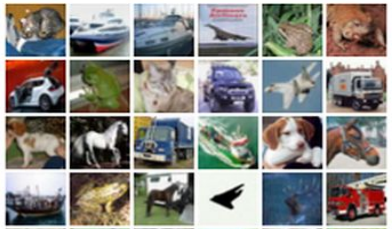
Doesn't use labels!



Reconstructed data



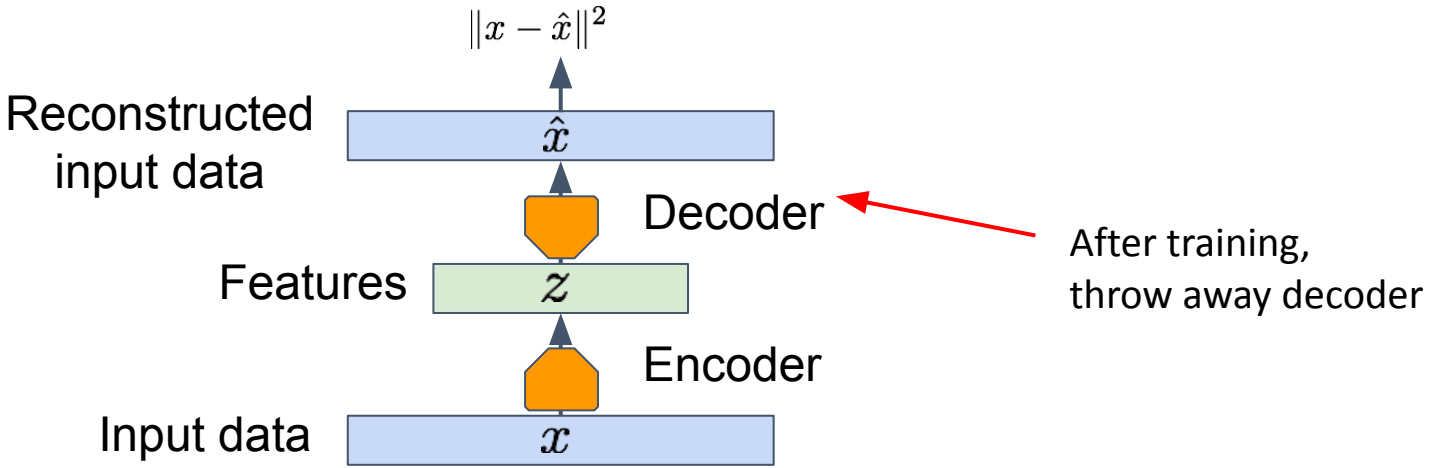
Input data



What can we do now?

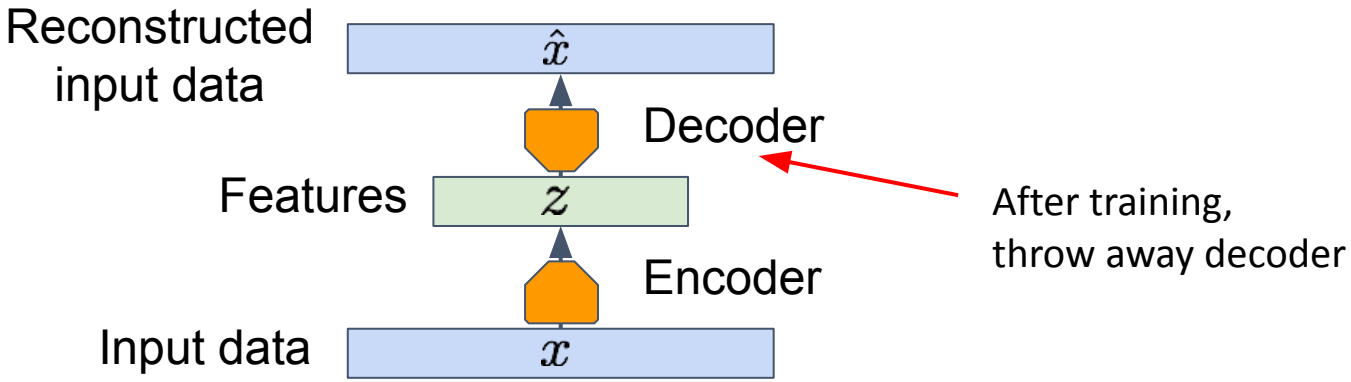
Autoencoders

Encoder can be used to initialize a **supervised** model



Autoencoders

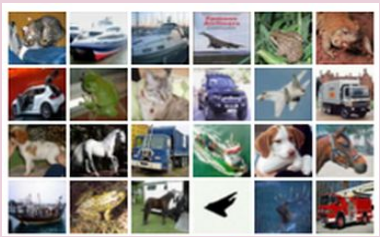
Encoder can be used to initialize a **supervised** model



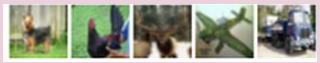
Autoencoders

Encoder can be used to initialize a **supervised** model.

Unlabeled dataset



bird plane
dog deer truck

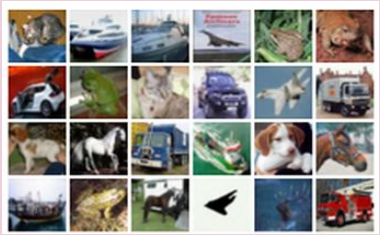


Small annotated dataset

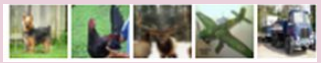
Autoencoders

Encoder can be used to initialize a **supervised** model.

Unlabeled dataset

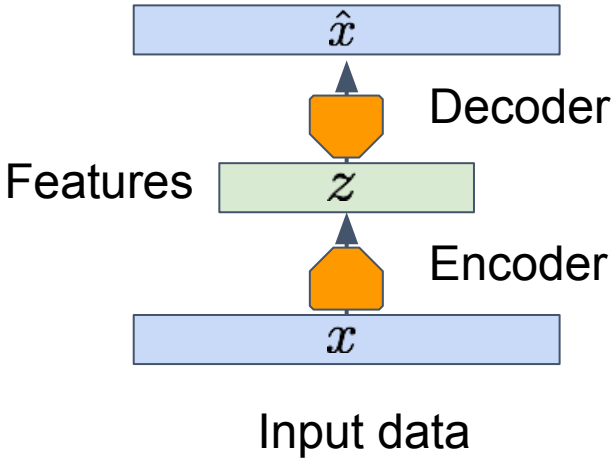


bird plane
dog deer truck



Small annotated dataset

Pretrain an auto-encoder on the unlabeled data.

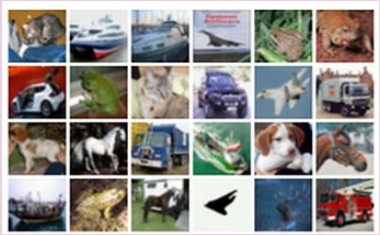


Reconstructed input data

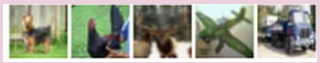
Autoencoders

Encoder can be used to initialize a **supervised** model.

Unlabeled dataset

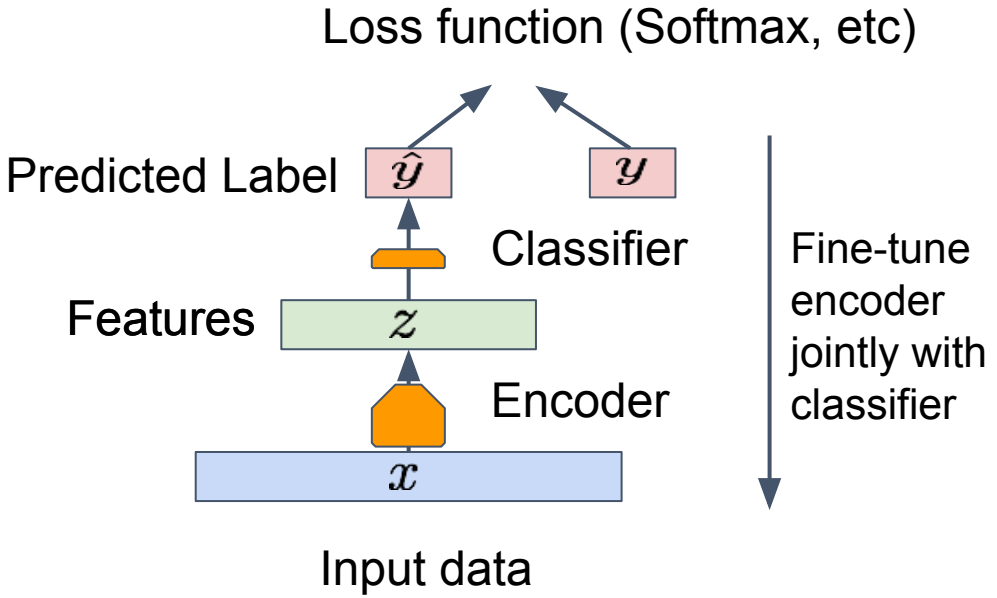


bird plane
dog deer truck



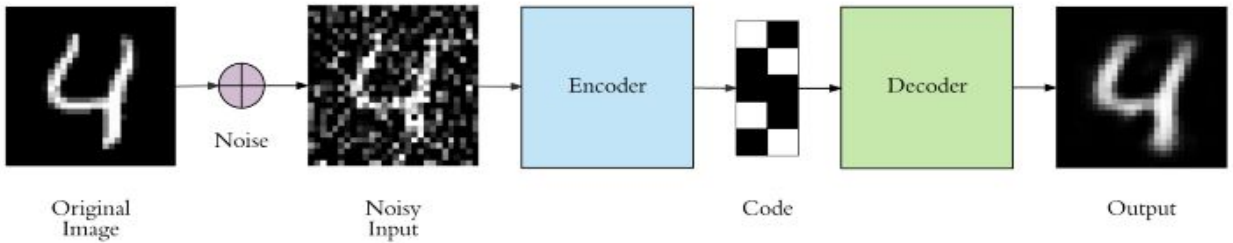
Small annotated dataset

Train a classifier on the labeled data using the learned feature.



Denoising Autoencoders

- Keeping the code layer small (latent representation) forces the autoencoder to learn an intelligent representation of the data.
- Another way to force the autoencoder to learn useful features is adding random noise to its inputs and making it recover the original noise-free data.



Vincent, H. Larochelle Y. Bengio and P.A. Manzagol, Extracting and Composing Robust Features with Denoising Autoencoders, Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08), pages 1096 - 1103, ACM, 2008.

Autoencoders

Auto-Encoder can:

- Reduce dimension
- Unsupervised pre-training
- Denoising
- Detect outlier $\|x - \hat{x}\|^2$

Autoencoders

Auto-Encoder can:

- Reduce dimension
- Unsupervised pre-training
- Denoising
- Detect outlier

Auto-Encoder cannot:

- Estimate the density function p(x)
- Sample according to p(x)

Given training data, generate new samples from same distribution

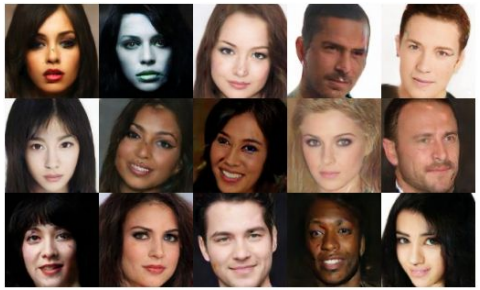
Training data ~ $p_{data}(x)$

Generated samples ~ $p_{model}(x)$

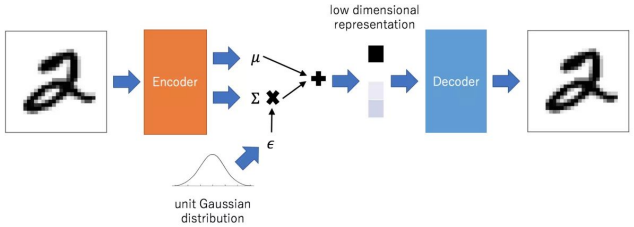
Want to learn $p_{model}(x)$ similar to $p_{data}(x)$

Why Generative Models?

- Realistic samples for artwork, super-resolution, colorization, etc.



- Generative models of time-series data can be used for simulation and planning (reinforcement learning applications!)
- Training generative models can also enable inference of **latent representations** that can be useful as general features



Variational Autoencoders (VAE)

Variational Autoencoders

VAEs are a combination of the following ideas:

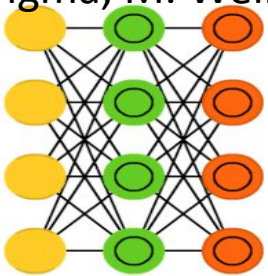
1. Autoencoders

2. Variational Approximation & Variational Lower Bound

3. “Reparameterization” Trick

Variational Autoencoders

D. Kingma, M. Welling, *Auto-Encoding Variational Bayes*, ICLR, 2014



Variational autoencoders (VAE) have the same architecture as AEs but are "taught" something else: an approximated probability distribution of the input samples. It's a bit back to the roots as they are bit more closely related to BMs and RBMs. They do however rely on Bayesian mathematics regarding probabilistic inference and independence, as well as a re-parametrisation trick to achieve this different representation. The inference and independence parts make sense intuitively, but they rely on somewhat complex mathematics. The basics come down to this: take influence into account. If one thing happens in one place and something else happens somewhere else, they are not necessarily related. If they are not related, then the error propagation should consider that. This is a useful approach because neural networks are large graphs (in a way), so it helps if you can rule out influence from some nodes to other nodes as you dive into deeper layers.

Kingma, Diederik P., and Max Welling. "Auto-encoding variational bayes." arXiv preprint arXiv:1312.6114 (2013).

Original Paper PDF

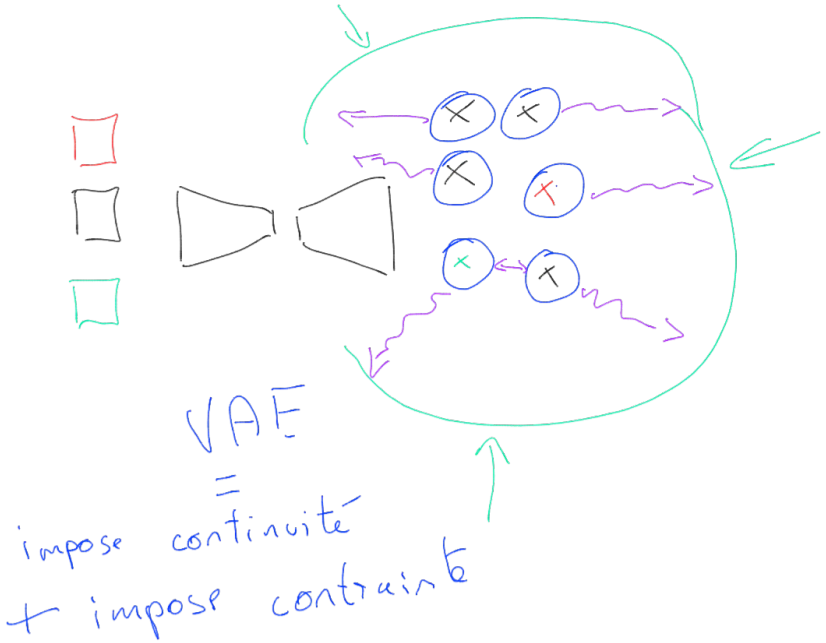
Variational Autoencoders

VAEs are a combination of the following ideas:

1. **Autoencoders**
2. Variational Approximation & Variational Lower Bound
3. “Reparameterization” Trick

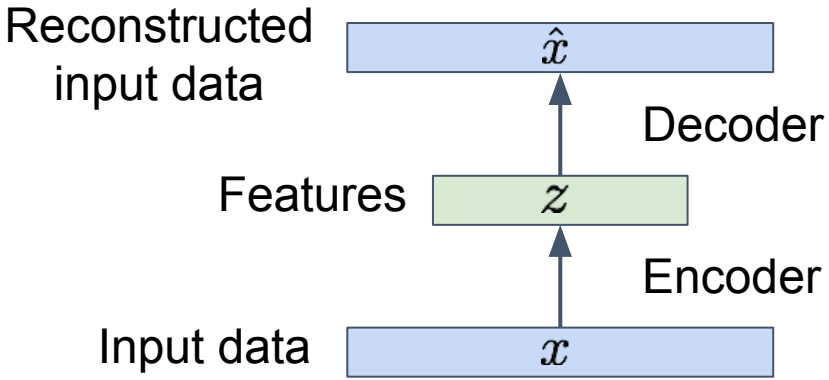
Autoencoders

Can we generate new images from an autoencoder?



Autoencoders

Can we generate new images from an autoencoder?

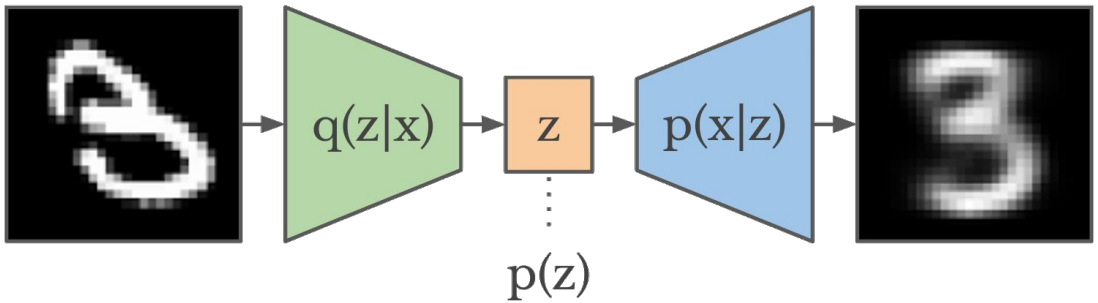


2 problems:

- How to sample Z
- No continuity in z space

Variational Autoencoders

Probabilistic spin on autoencoders - will let us sample from the model to generate data!



Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014
C. Doearsh, Tutorial on Variational Autoencoders, https://arxiv.org/pdf/1606.05908.pdf

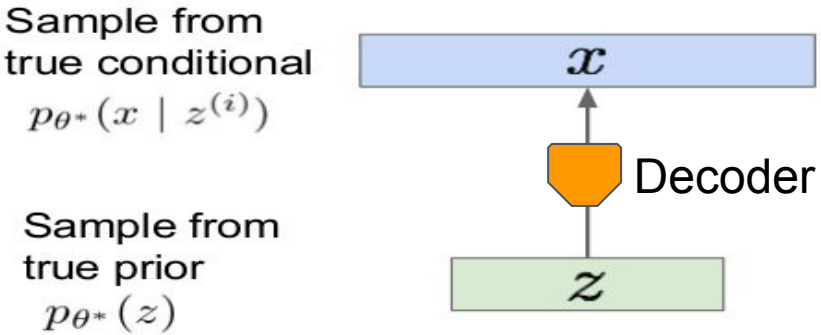
Variational Autoencoders

VAEs are a combination of the following ideas:

1. Autoencoders
2. **Variational Approximation & Variational Lower Bound**
3. “Reparameterization” Trick

Variational Autoencoders

Assume training data representation x is generated from underlying unobserved (latent) z

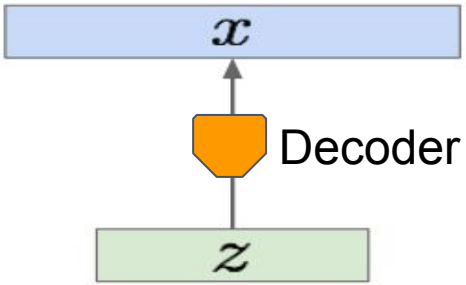


Variational Autoencoders

Assume training data representation x is generated from underlying unobserved (latent) z

Sample from true conditional $p_{\theta^*}(x \mid z^{(i)})$

Sample from true prior $p_{\theta^*}(z)$



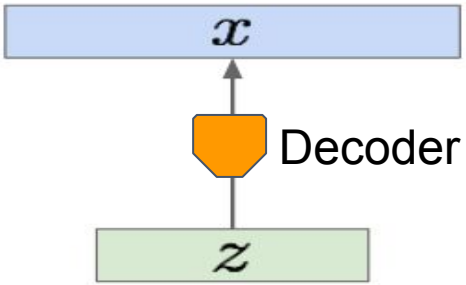
Intuition:
x is an image, z is a latent factor used to generate x: attributes, orientation, etc.

Variational Autoencoders

Assume training data representation x is generated from underlying unobserved (latent) z

Sample from true conditional $p_{\theta^*}(x \mid z^{(i)})$

Sample from true prior $p_{\theta^*}(z)$



Intuition:
x is an image, z is a latent factor used to generate x: attributes, orientation, etc.

We want to estimate the optimal parameters θ^* of this generative model, according to maximum likelihood

But what is a loss? But maximum likelihood

- **Examples with supervised training**
  - Given training samples

$$T=\{(\mathbf{x}_1, \mathbf{t}_1), (\mathbf{x}_2, \mathbf{t}_2), \ldots, (\mathbf{x}_N, \mathbf{t}_N)\}$$

  - Choose an objective function (loss function)
  - Adjust all the weights of the network θ such that a cost function is minimized

$$\min_w = L_{tot}(w)$$
$$= \sum_i L(\mathbf{t}_i, y(\mathbf{x}_i; w))$$

Loss: Regression

The choice of the loss function depends on the problem.

MSE loss:

$$\frac{1}{2}\sum_{n=1}^{N}\|\mathbf{y}(\mathbf{x}_n,\mathbf{w})-\mathbf{t}_n\|^2$$

But why?

Let's assume a gaussian distribution of the error:

$$p(t|\mathbf{x},\mathbf{w})=\mathcal{N}\left(t|y(\mathbf{x},\mathbf{w}),\beta^{-1}\right)$$

Dataset likelihood:

$$p(\mathbf{t}|\mathbf{X},\mathbf{w},\beta)=\prod_{n=1}^{N}p(t_n|\mathbf{x}_n,\mathbf{w},\beta)$$

Loss: Regression but why MSE?

Let's assume a gaussian distribution of the error:

$$p(t|\mathbf{x}, \mathbf{w}) = \mathcal{N}\left(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}\right)$$

Dataset likelihood:

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^{N} p(t_n|\mathbf{x}_n, \mathbf{w}, \beta)$$

We take -log:

$$\frac{\beta}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2 - \frac{N}{2}\ln\beta + \frac{N}{2}\ln(2\pi)$$

Loss: Regression but why MSE?

Maximize the data log-likelihood assuming gaussian errors

$$p(t|\mathbf{x}, \mathbf{w}) = \mathcal{N}\left(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}\right)$$

$$\Leftrightarrow$$

Minimize the MSE:

$$\frac{1}{2}\sum_{n=1}^{N} \|\mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{t}_n\|^2$$

Loss: classification

- Classification problem with `C` classes: negative loglikelihood

$$\text{loss}(x) = -\log\left(prob(c = c_{\text{True}}|x)\right)$$

- How to get probabilities:

$$Softmax(x) = \frac{\exp(x)}{\sum_c \exp(x[c])}$$

$$\sum_{c=0}^{C} Softmax(x)[c] = 1$$

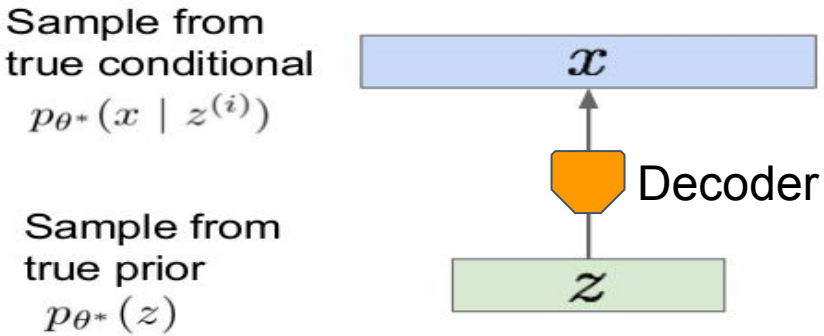
- **Cross-entropy loss**

$$\text{loss}(x) = -\log\left(Softmax(x)[c_{\text{True}}]\right)$$

Variational Autoencoders

How should we **train** this model?

Learn model parameters to maximize likelihood of training data



$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

Variational Autoencoders

Data likelihood: $p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$

Intractable to compute p(x|z) for every z!

Simple Gaussian prior

Decoder neural network

Variational Autoencoders

Data likelihood: $p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$

Intractable to compute p(x|z) for every z!

Simple Gaussian prior

Decoder neural network

Posterior density also intractable: $p_\theta(z|x) = p_\theta(x|z) p_\theta(z) / p_\theta(x)$

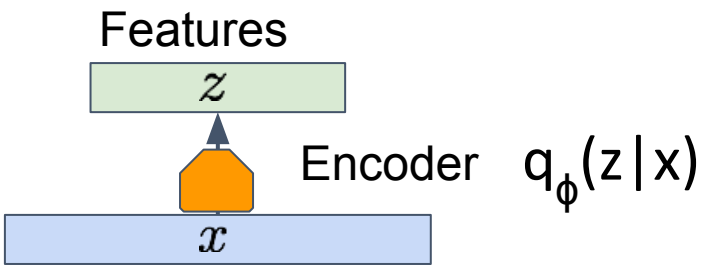
Decoder neural network

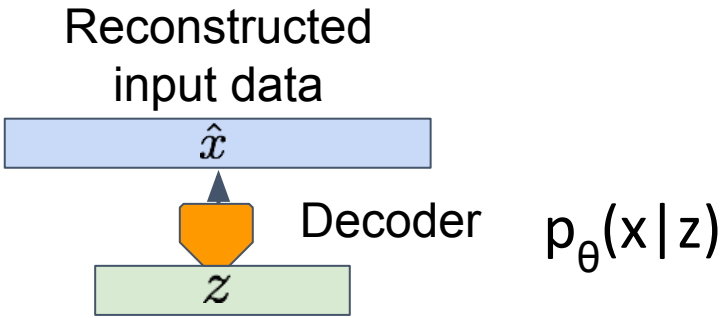
Simple gaussian prior

Intractable data likelihood

Variational Approximation

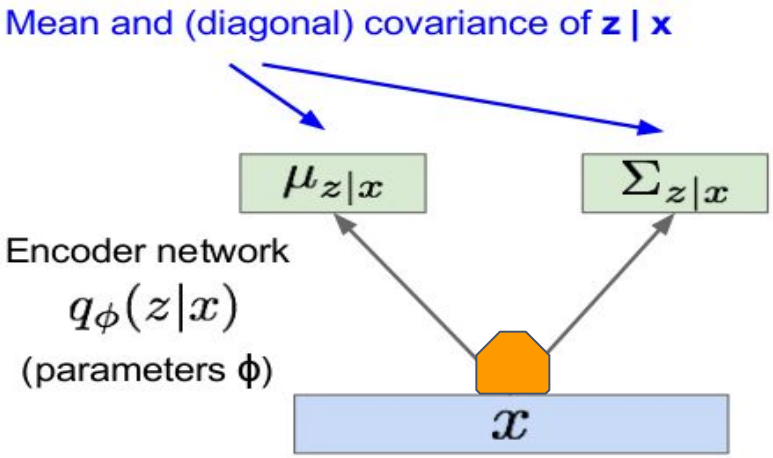
- Solution: In addition to decoder network modeling $p_\theta(x|z)$, define **additional encoder network** $q_\phi(z|x)$ that approximates $p_\theta(z|x)$
- This allows us to derive a **lower bound** on the data likelihood that is tractable, which we can optimize

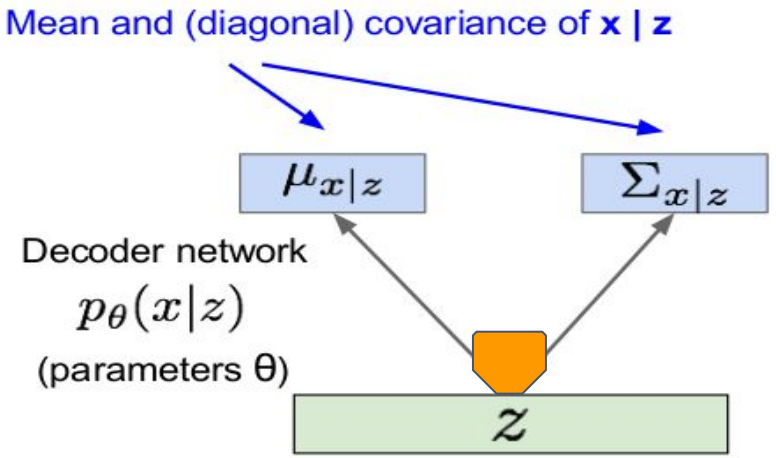




Variational Approximation

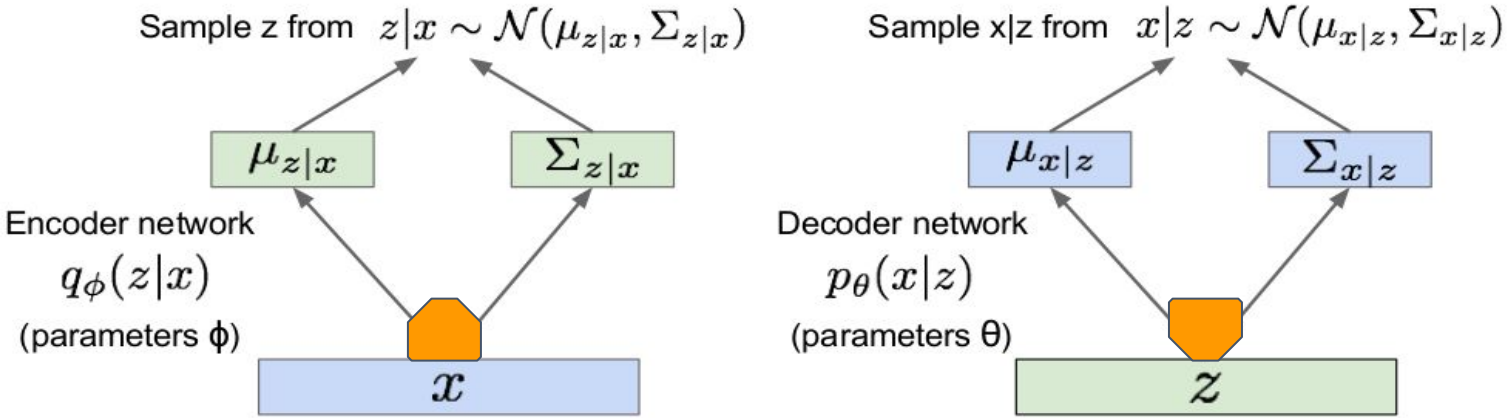
- Since we are modeling probabilistic generation of data, encoder and decoder networks are probabilistic





Variational Approximation

- Since we are modeling probabilistic generation of data, encoder and decoder networks are probabilistic



- Encoder and decoder networks also called “recognition”/“inference” and “generation” networks

Variational Approximation

- Let us consider the log likelihood of the data

Variational Approximation

- Let us consider the log likelihood of the data

$$
\begin{aligned}
\log p_\theta(x^{(i)}) &= \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] && (p_\theta(x^{(i)}) \text{ Does not depend on } z)\\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] && \text{(Bayes' Rule)}\\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] && \text{(Multiply by constant)}\\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] && \text{(Logarithms)}\\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z)) + D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))
\end{aligned}
$$

The expectation wrt. z (using encoder network) let us write nice KL terms

Variational Approximation

- Let us consider the log likelihood of the data

$$\log p_\theta(x^{(i)}) = \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Bayes' Rule)}$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] \quad \text{(Multiply by constant)}$$

$$= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Logarithms)}$$

$$= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z)) + D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))$$

Decoder network gives $p_\theta(x|z)$, can compute estimate of this term through sampling. (Sampling differentiable through reparam. trick, see paper.)

This KL term (between Gaussians for encoder and z prior) has nice closed-form solution!

$p_\theta(z|x)$ intractable (saw earlier), can't compute this KL term :(But we know KL divergence always >= 0.

Variational Approximation

- Let us consider the log likelihood of the data

$$\log p_\theta(x^{(i)}) = \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Bayes' Rule)}$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] \quad \text{(Multiply by constant)}$$

$$= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Logarithms)}$$

$$= \underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)} + \underbrace{D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))}_{\geq 0}$$

We want to maximize the data likelihood

Tractable lower bound which we can take gradient of and optimize! ($p_\theta(x|z)$ differentiable, KL term differentiable)

Variational Approximation

- Let us consider the log likelihood of the data

$$\log p_\theta(x^{(i)}) = \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Bayes' Rule)}$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] \quad \text{(Multiply by constant)}$$

$$= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Logarithms)}$$

$$= \underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)} + \underbrace{D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))}_{>0}$$

We want to maximize the data likelihood

$$\log p_\theta(x^{(i)}) \geq \mathcal{L}(x^{(i)}, \theta, \phi)$$

Variational lower bound ("ELBO")

$$\theta^*, \phi^* = \arg\max_{\theta,\phi} \sum_{i=1}^{N} \mathcal{L}(x^{(i)}, \theta, \phi)$$

Training: Maximize lower bound

Variational Approximation

- Let us consider the log likelihood of the data

$$\log p_\theta(x^{(i)}) = \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Bayes' Rule)}$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] \quad \text{(Multiply by constant)}$$

$$= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Logarithms)}$$

$$= \underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)} + \underbrace{D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))}_{>0}$$

Reconstruct the input data

Make approximate posterior distribution close to prior

$$\log p_\theta(x^{(i)}) \geq \mathcal{L}(x^{(i)}, \theta, \phi)$$

Variational lower bound ("ELBO")

$$\theta^*, \phi^* = \arg\max_{\theta,\phi} \sum_{i=1}^{N} \mathcal{L}(x^{(i)}, \theta, \phi)$$

Training: Maximize lower bound

Variational Autoencoders: recap

- Let us consider the log likelihood of the data

Variational Autoencoders

Maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

Variational Autoencoders

$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

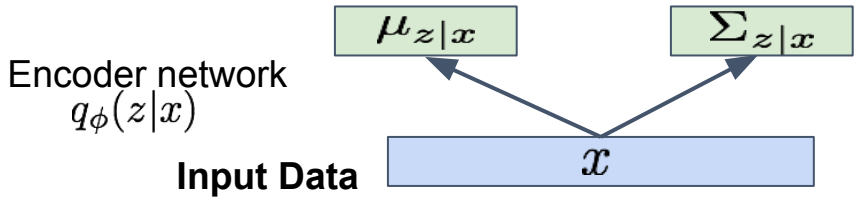
Let's look at computing the bound (forward pass) for a given minibatch of input data

Input Data x

Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

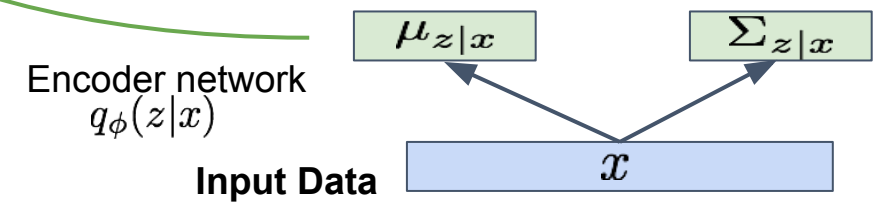


Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

Make approximate posterior distribution close to prior

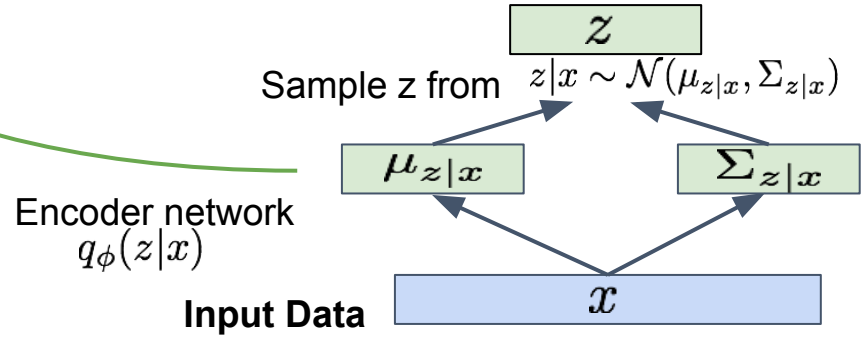


Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

Make approximate posterior distribution close to prior

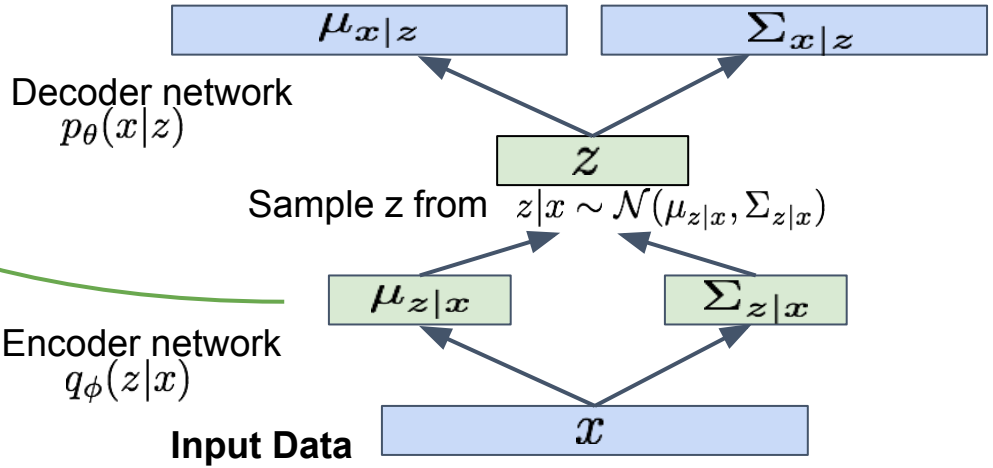


Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

Make approximate posterior distribution close to prior



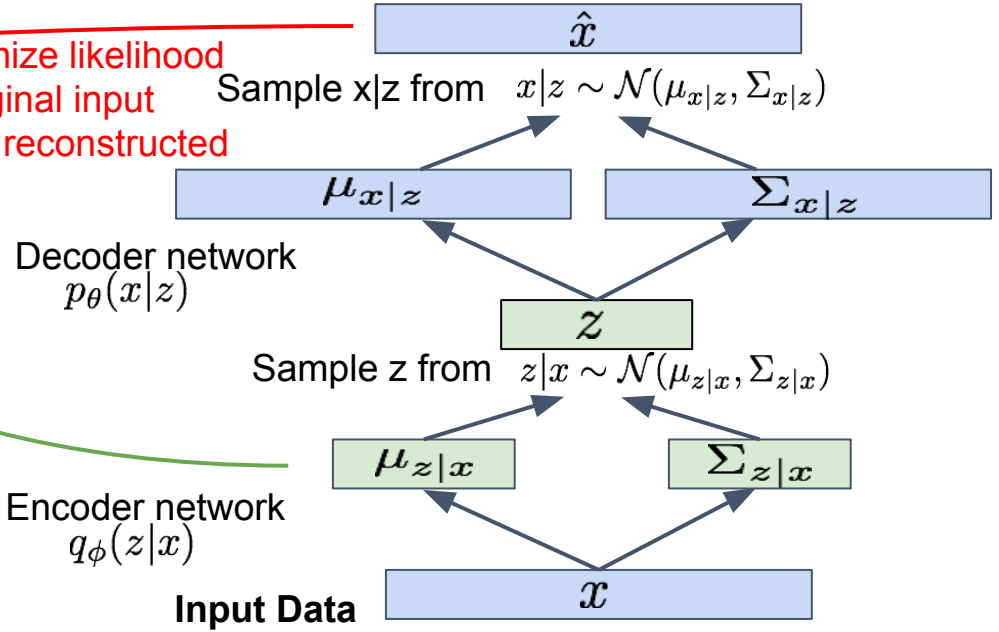
Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

Maximize likelihood of original input being reconstructed

Make approximate posterior distribution close to prior

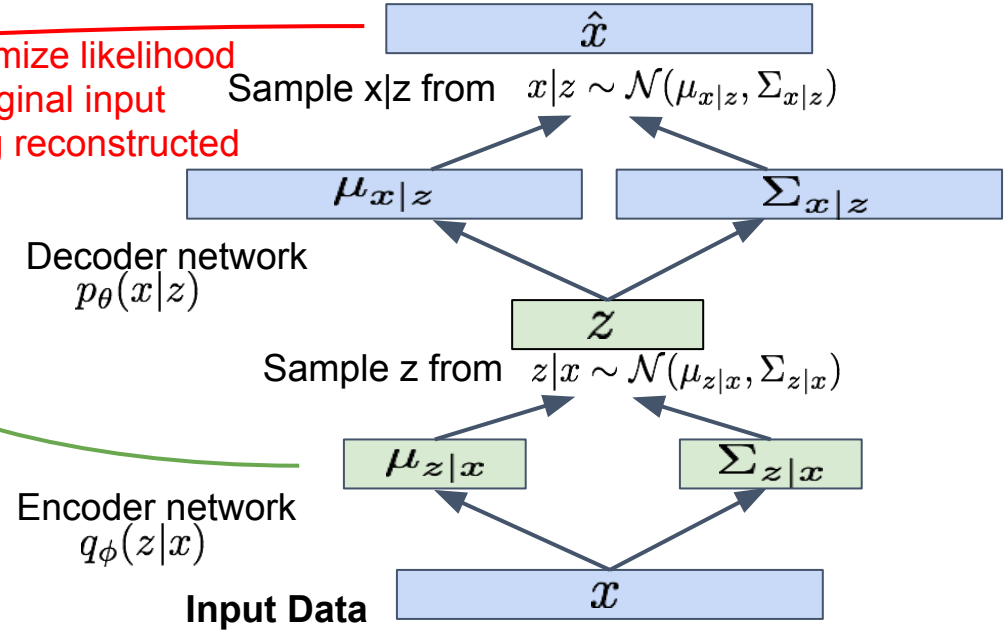


Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

Maximize likelihood of original input being reconstructed

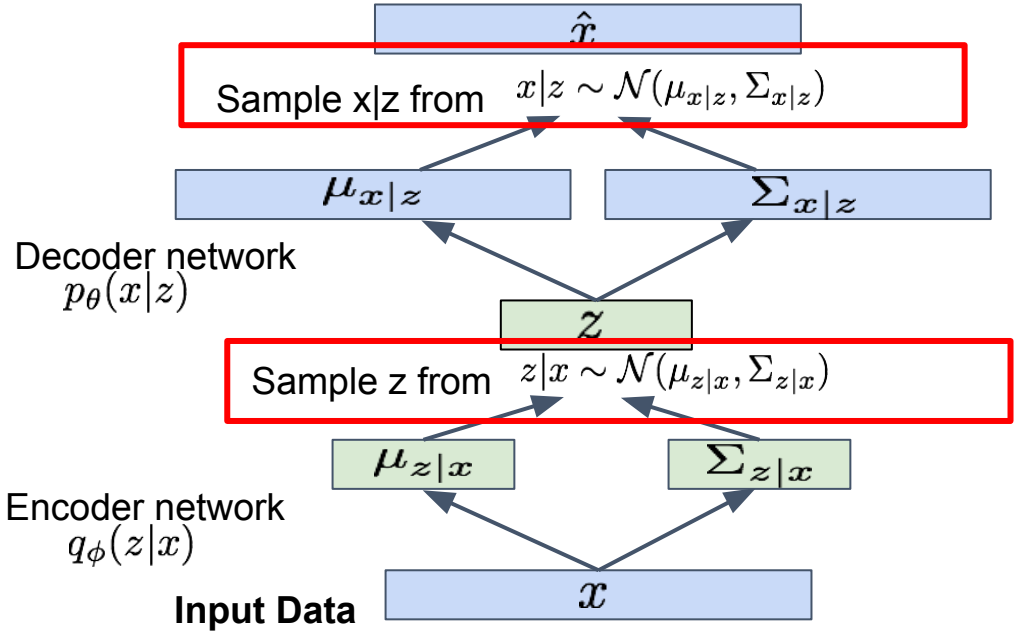


Make approximate posterior distribution close to prior

For every minibatch of input data: compute this forward pass, and then backprop!

Employ "Reparameterization" Trick

Variational Autoencoders



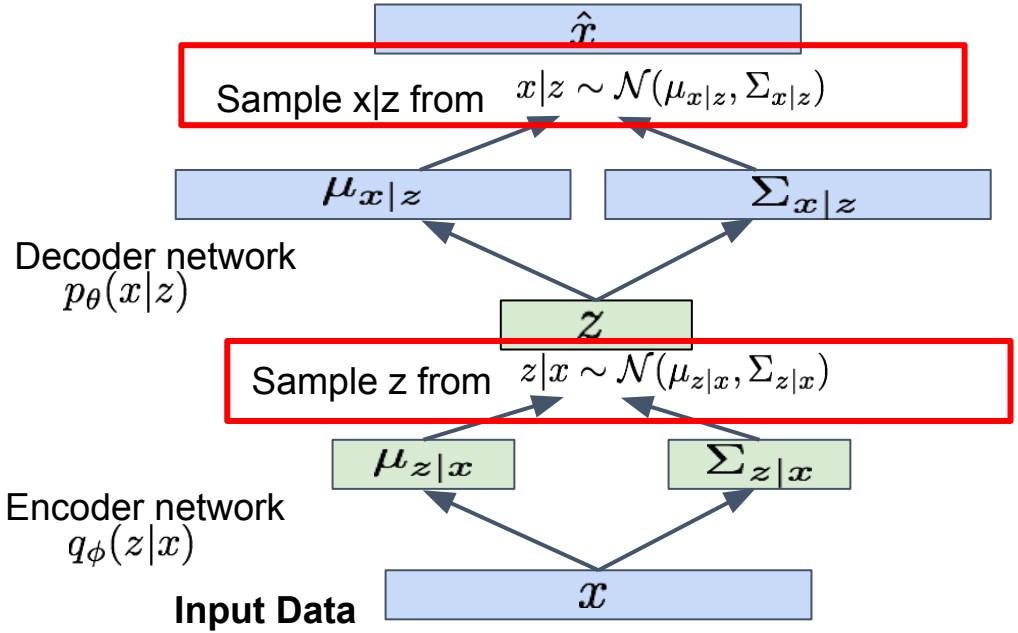
Not differentiable!

Not differentiable!

Re-parametrization trick:

If $z_0 \sim \mathcal{N}(0,1)$, then $z = \mu_{x|z} + z_0 \sqrt{\Sigma_{x|z}} \sim \mathcal{N}(\mu_{x|z}, \Sigma_{x|z})$

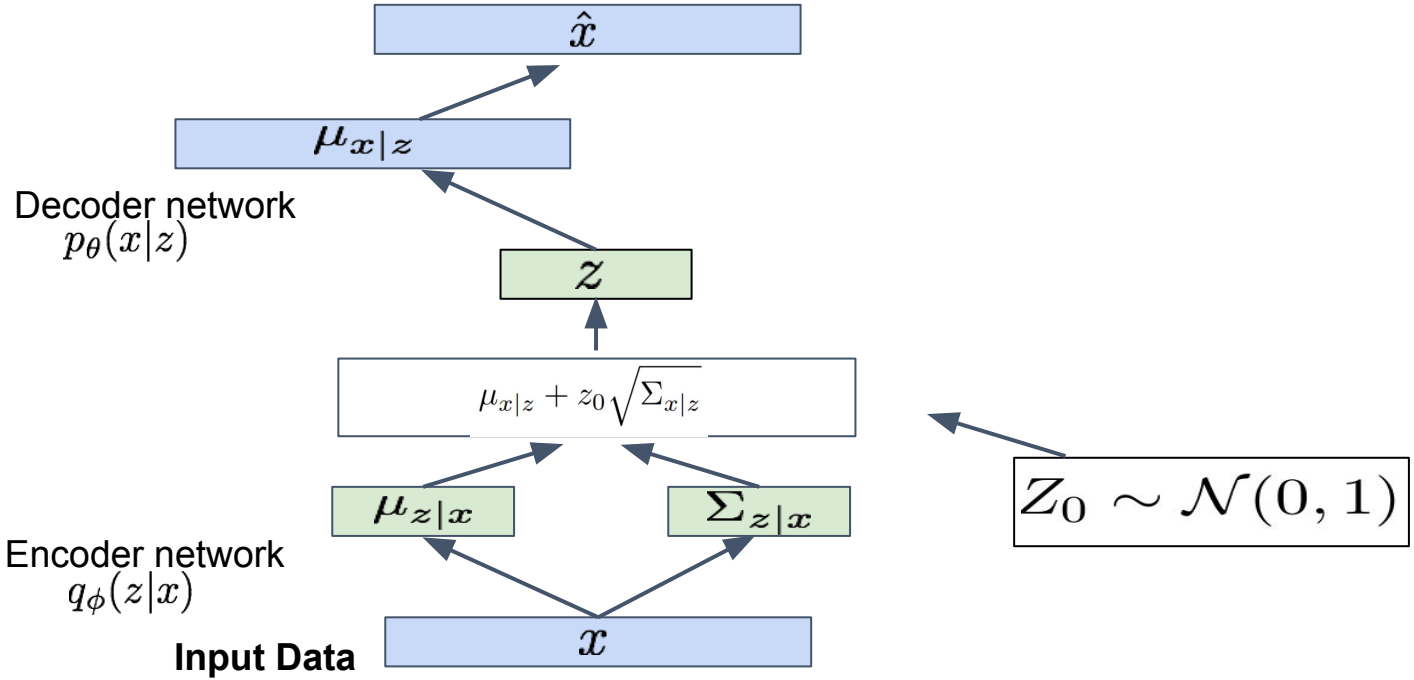
Variational Autoencoders



Not differentiable!

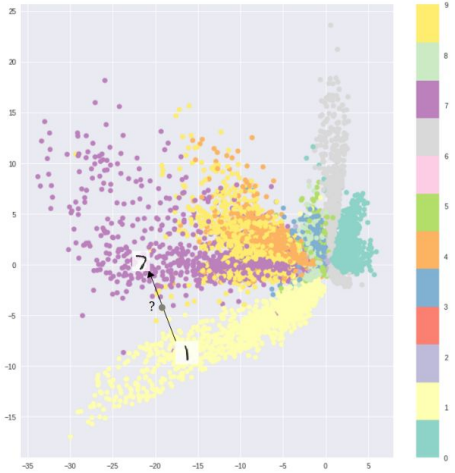
Not differentiable!

Variational Autoencoders

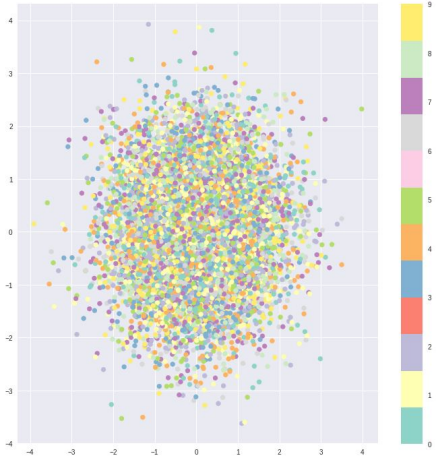


Variational Autoencoders

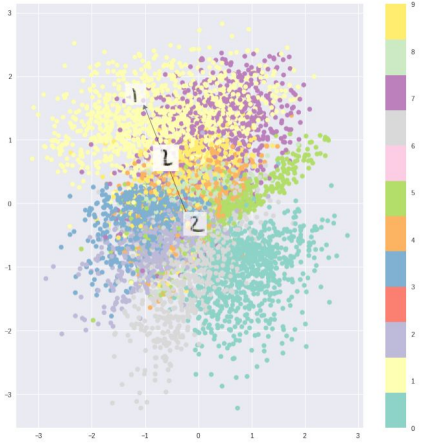
Autoencoders vs Variational Autoencoders



AE (reconstruction)



Only KL term

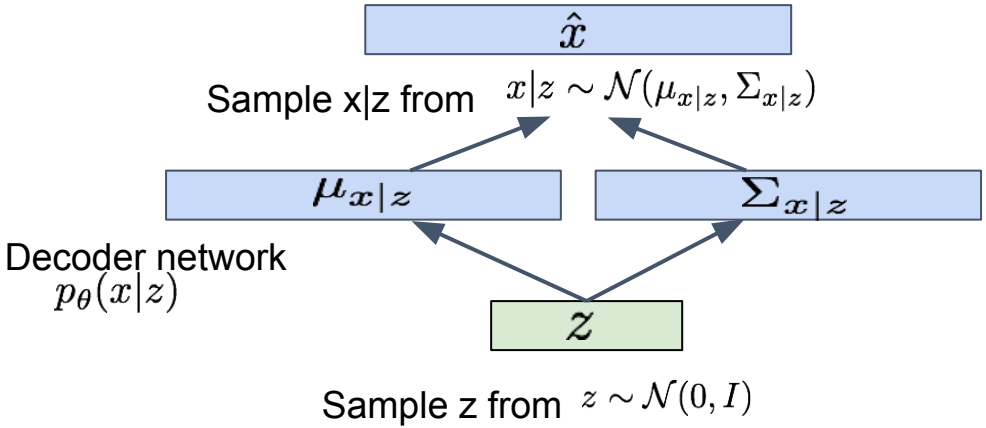


VAE (reconstruction+KL)

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

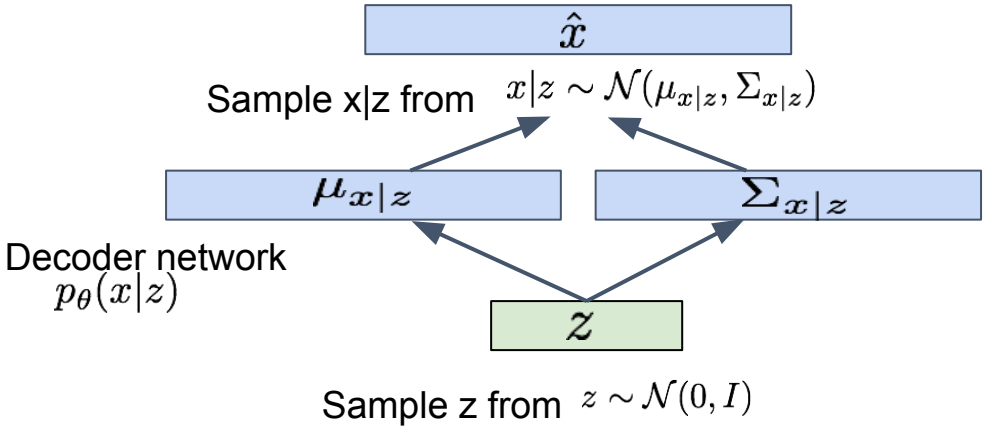
Generating Data

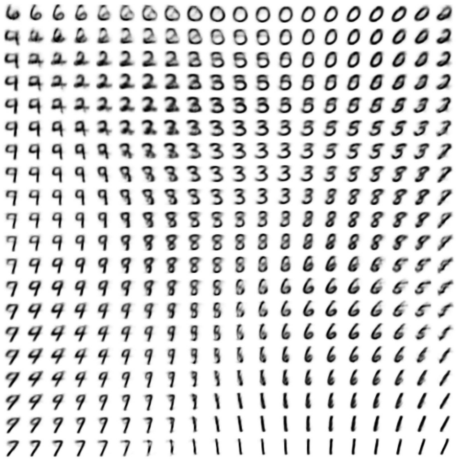
Use decoder network. Now sample z from prior!



Generating Data

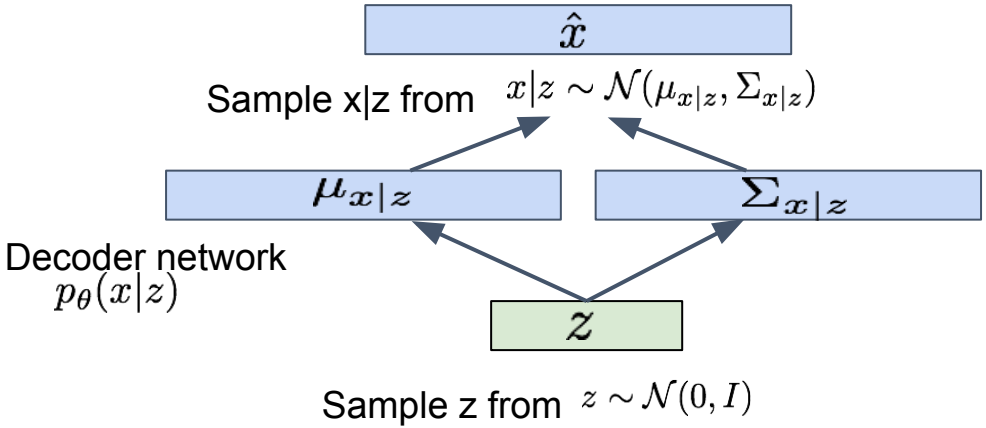
Use decoder network. Now sample z from prior!



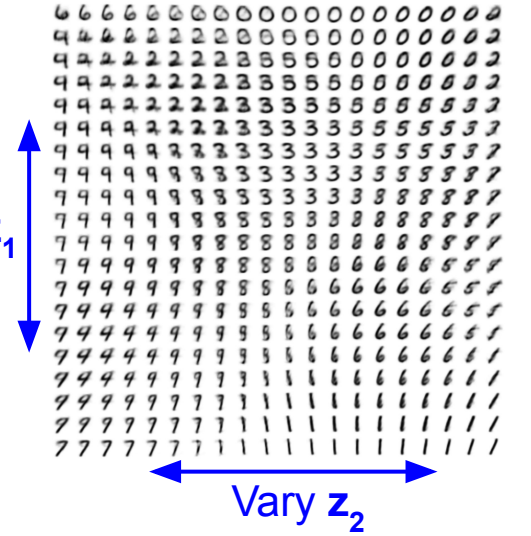


Generating Data

Use decoder network. Now sample z from prior!



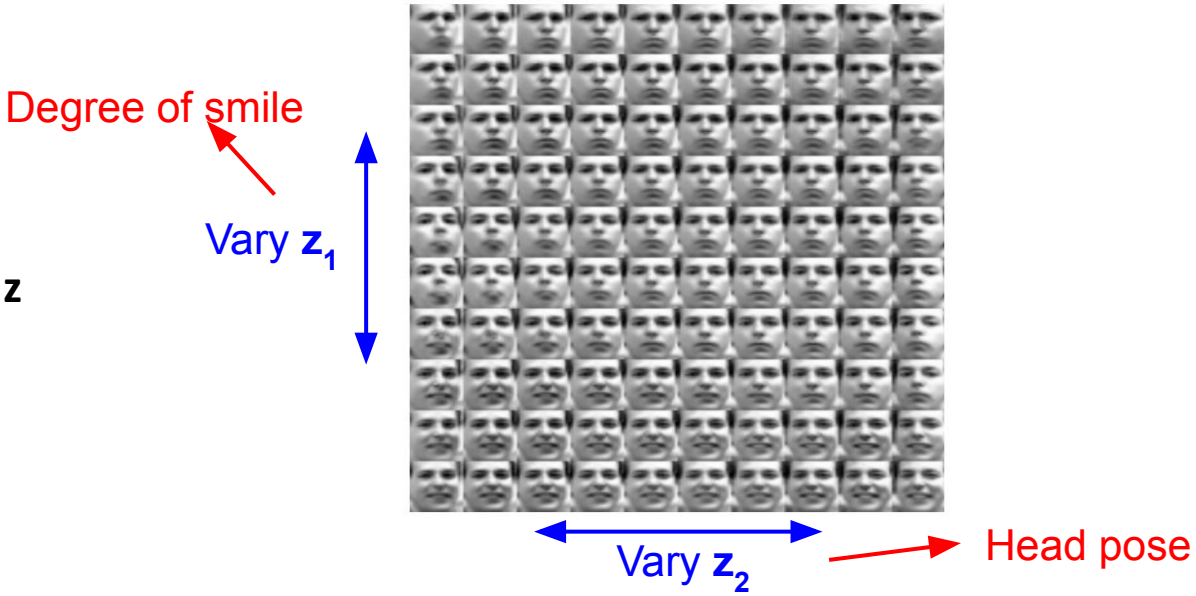
Data manifold for 2-d **z**



Generating Data

Diagonal prior on **z** => independent latent variables

Different dimensions of **z** encode interpretable factors of variation

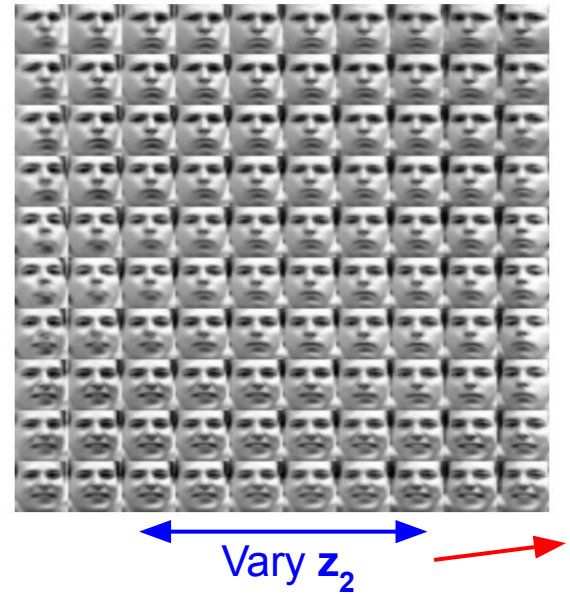


Generating Data

Diagonal prior on **z** => independent latent variables

Different dimensions of **z** encode interpretable factors of variation

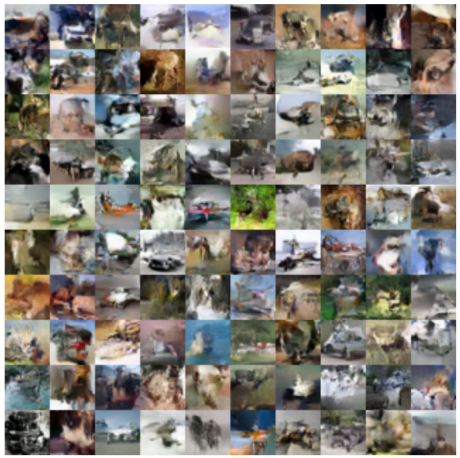
Also good feature representation that can be computed using $q_\phi(z|x)$!



http://www.dpkingma.com/sgvb_mnist_demo/demo.html

Generating Data: interpolation

Generating Data

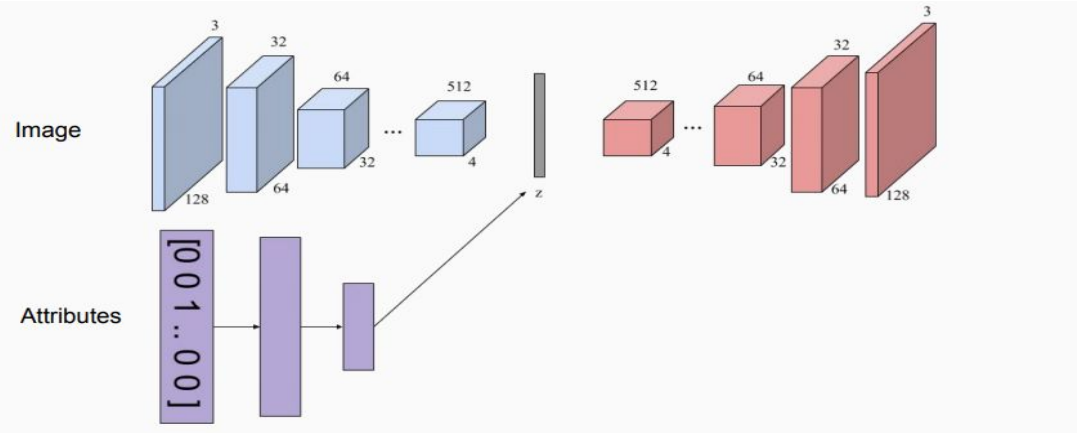


32x32 CIFAR-10

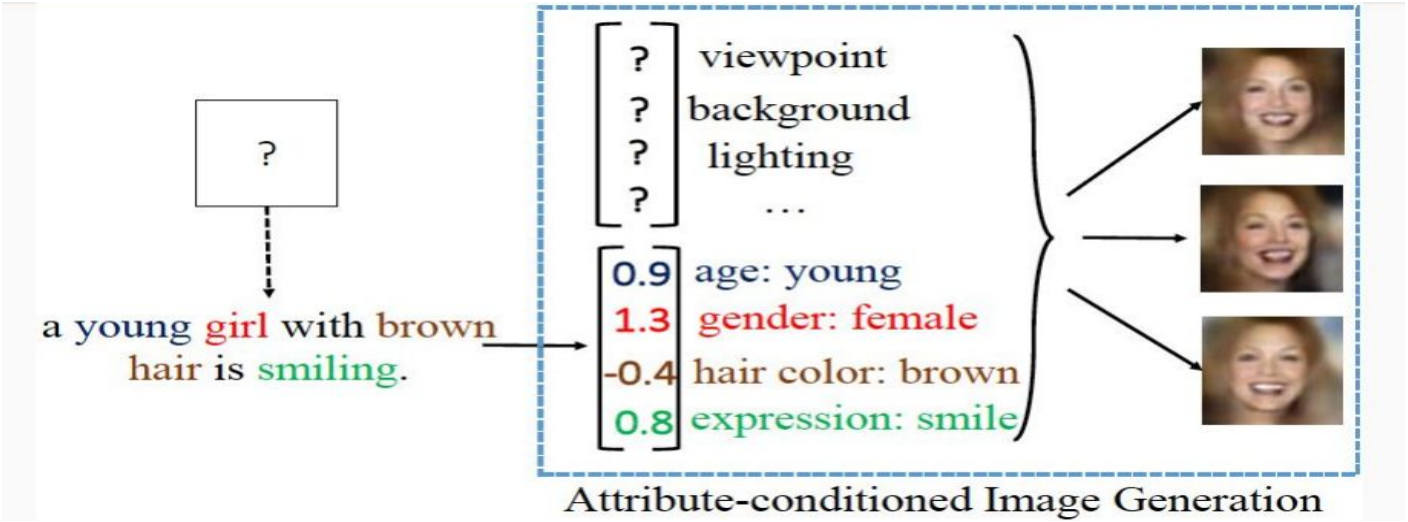
Labeled Faces in the Wild

Conditional VAE

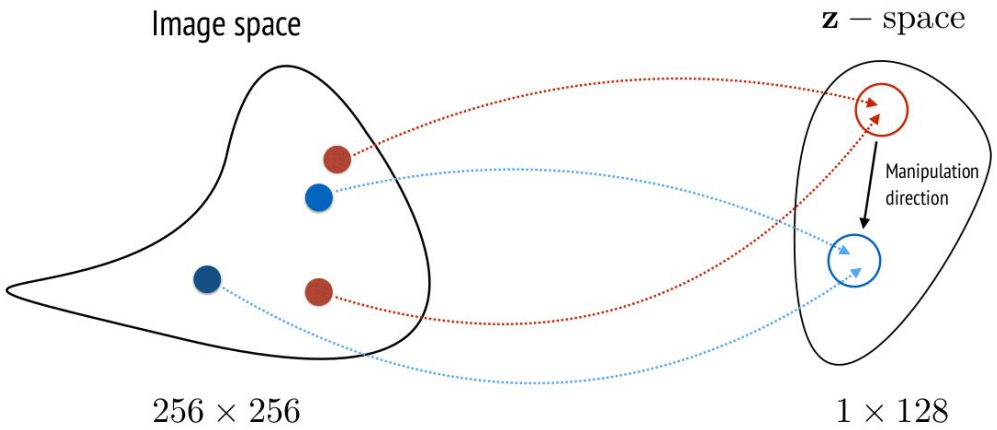
- What if we have labels? (e.g. digit labels or attributes) or other inputs we wish to condition on?
- None of the derivation changes.
- Replace all *p(x|z)* with *p(x|z,y)*.
- Replace all *q(x|z)* with *q(x|z,y)*.
- Go through the same KL divergence procedure, to get the same lower bound.



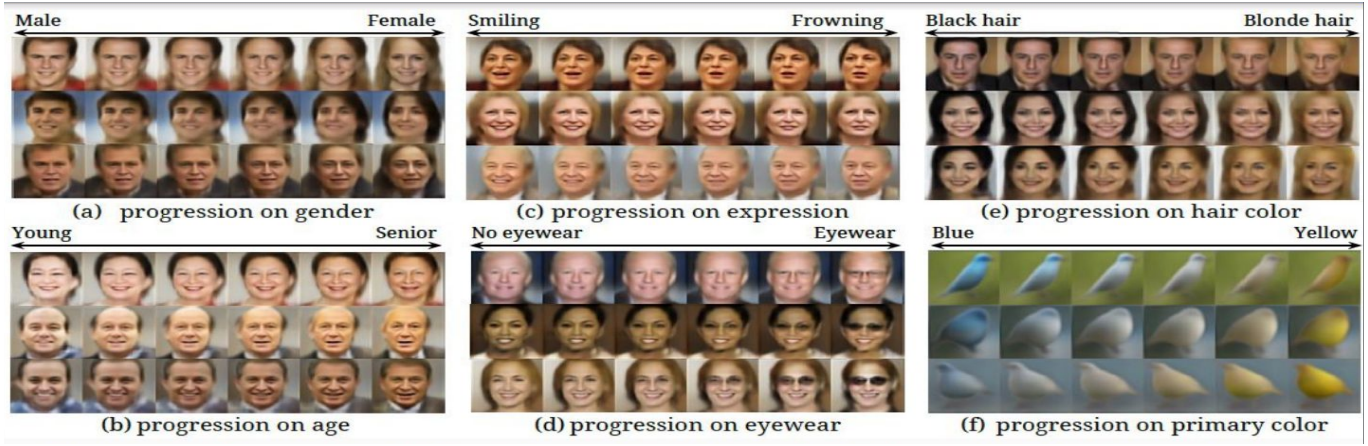
Generating Data: attribute-driven



Generating Data: attribute-driven



Generating Data: attribute-driven

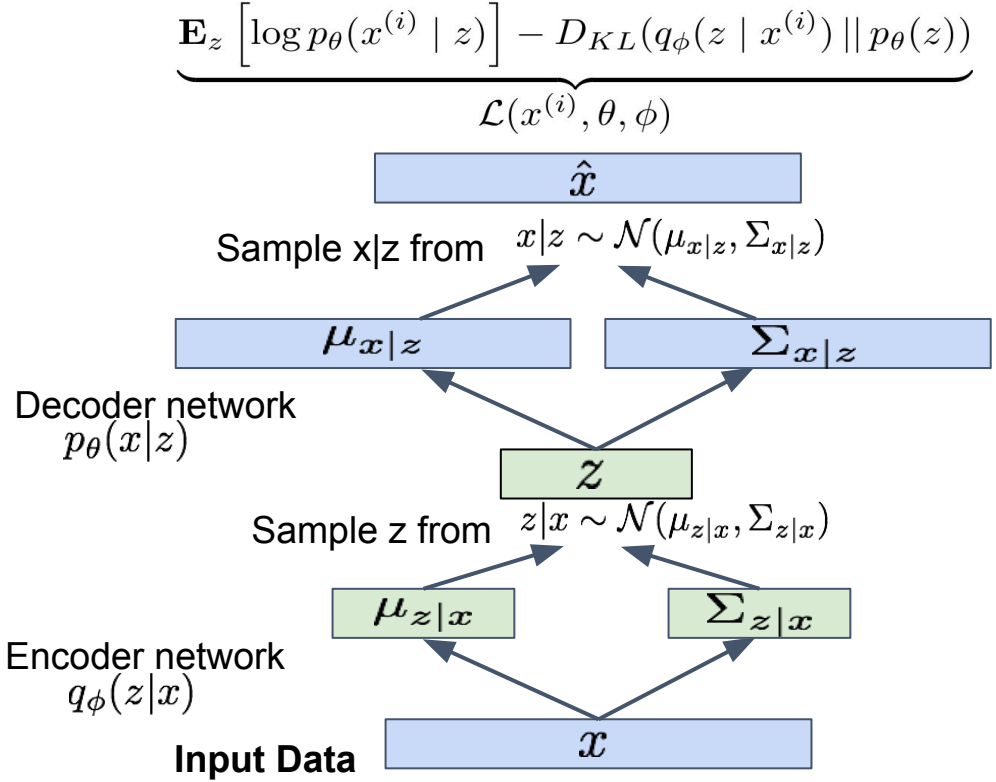


Variational Autoencoders

- Probabilistic spin to traditional autoencoders => allows generating data
- Defines an intractable density => derive and optimize a (variational) lower bound
- **Pros:**
  - Principled approach to generative models
  - Allows inference of q(z|x), can be useful feature representation for other tasks
- **Cons:**
  - Maximizes lower bound of likelihood: okay, but not as good evaluation as PixelRNN/PixelCNN
  - Samples blurrier and lower quality compared to state-of-the-art (GANs)
- **Active areas of research:**
  - More flexible approximations, e.g. richer approximate posterior instead of diagonal Gaussian
  - Incorporating structure in latent variables

Variational Autoencoders: recap

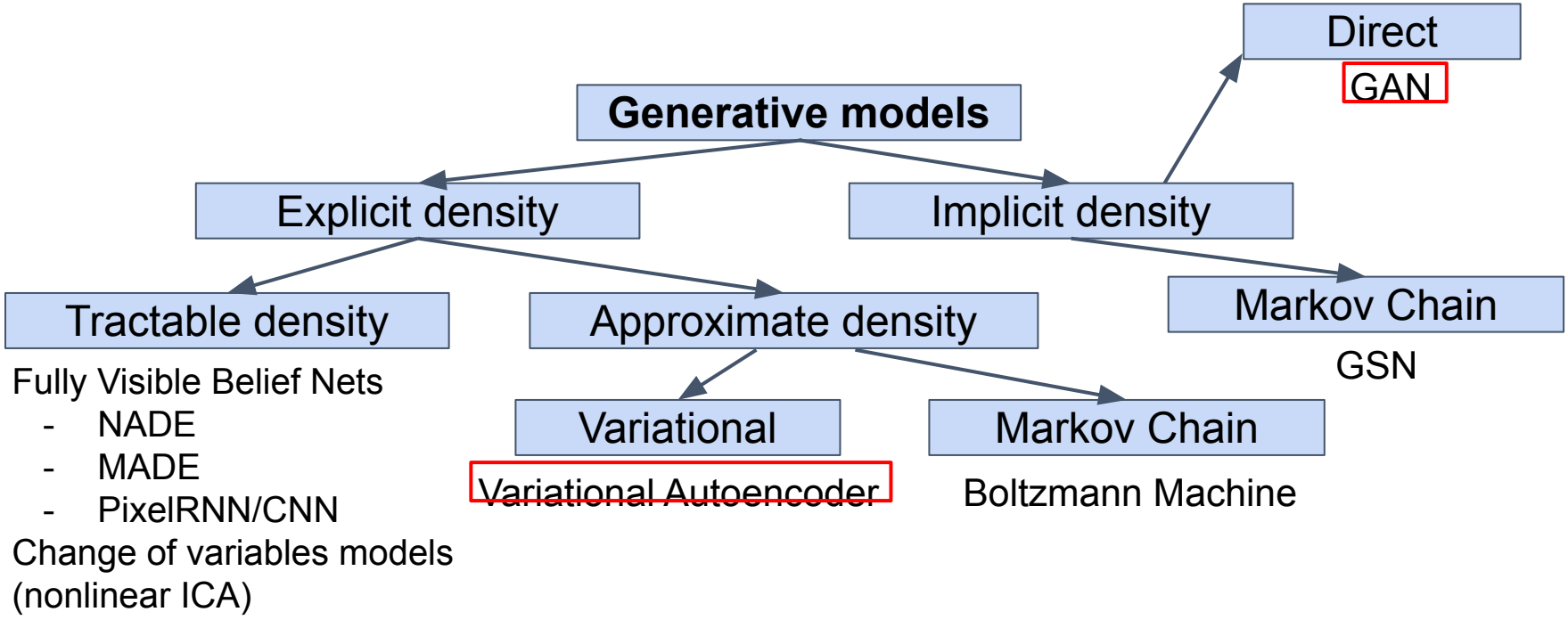
Variational Autoencoders: recap



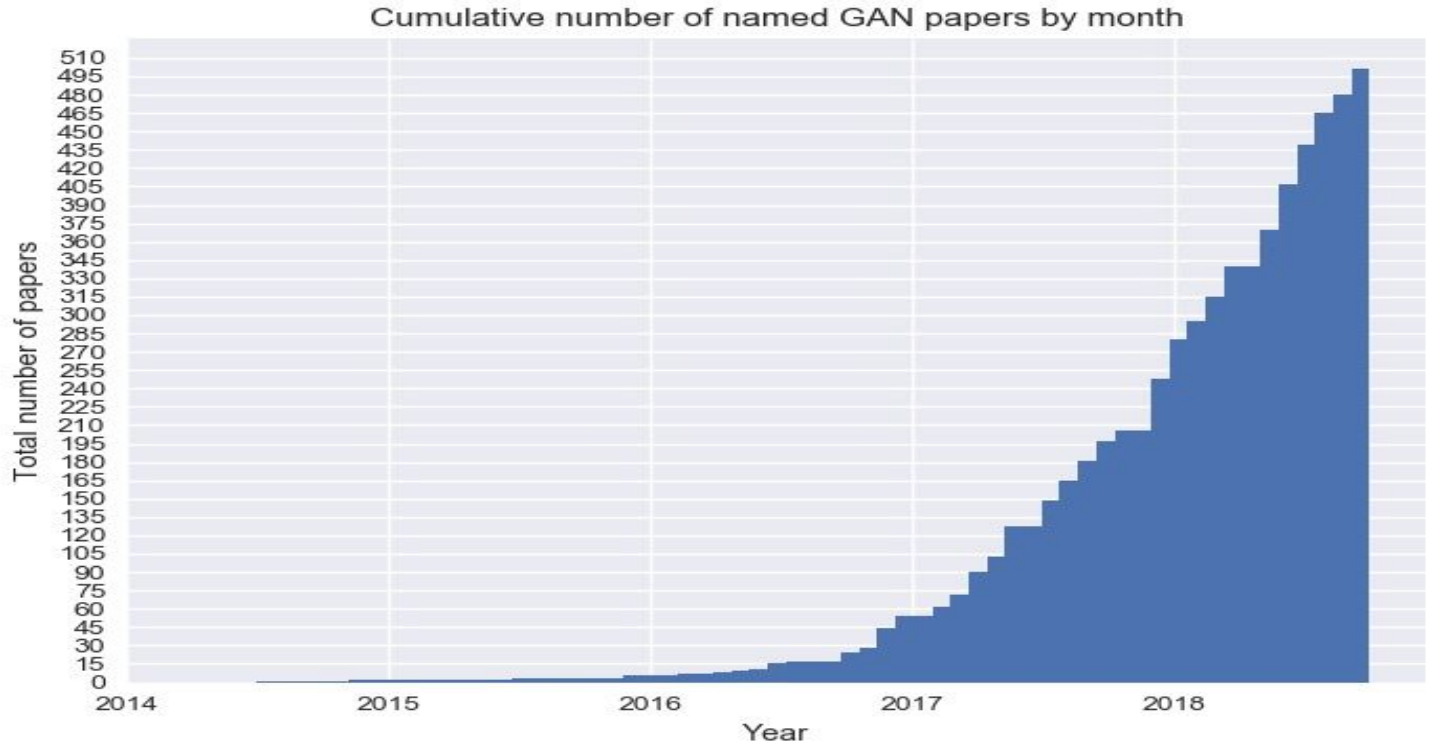
Useful Links

- D. Kingma, M. Welling, *Auto-Encoding Variational Bayes*, ICLR, 2014
- Carl Doersch, *Tutorial on Variational Autoencoders* arXiv, 2016
- Xinchen Yan, Jimei Yang, Kihyuk Sohn, Honglak Lee, *Attribute2Image: Conditional Image Generation from Visual Attributes,* ECCV, 2016
- Jacob Walker, Carl Doersch, Abhinav Gupta, Martial Hebert, *An Uncertain Future: Forecasting from Static Images using Variational Autoencoders*, ECCV, 2016
- Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, Ole Winther, *Autoencoding beyond pixels using a learned similarity metric*, ICML, 2016
- Aditya Deshpande, Jiajun Lu, Mao-Chuang Yeh, David Forsyth, *Learning Diverse Image Colorization*, arXiv, 2016
- Raymond Yeh, Ziwei Liu, Dan B Goldman, Aseem Agarwala, *Semantic Facial Expression Editing using Autoencoded Flow*, arXiv, 2016

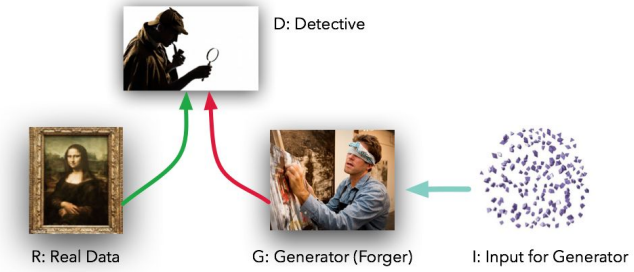
Taxonomy of Generative Models



GAN zoo



https://github.com/hindupuravinash/the-gan-zoo



Generative Adversarial Networks (GAN)

Generative Adversarial Networks

Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

Solution: Sample from a simple distribution, e.g. random noise. Learn transformation to training distribution.

Q: What can we use to represent this complex transformation?

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

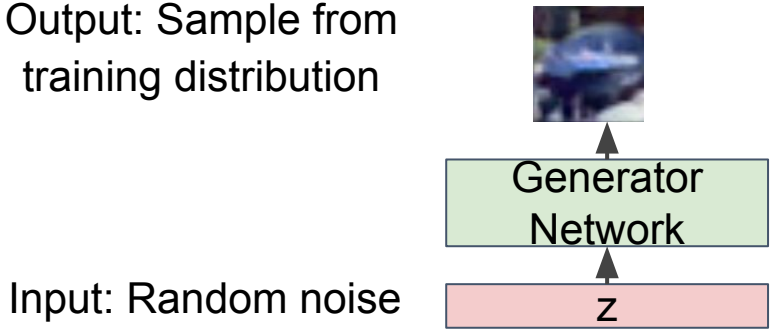
Generative Adversarial Networks

Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

Solution: Sample from a simple distribution, e.g. random noise. Learn transformation to training distribution.

Q: What can we use to represent this complex transformation?

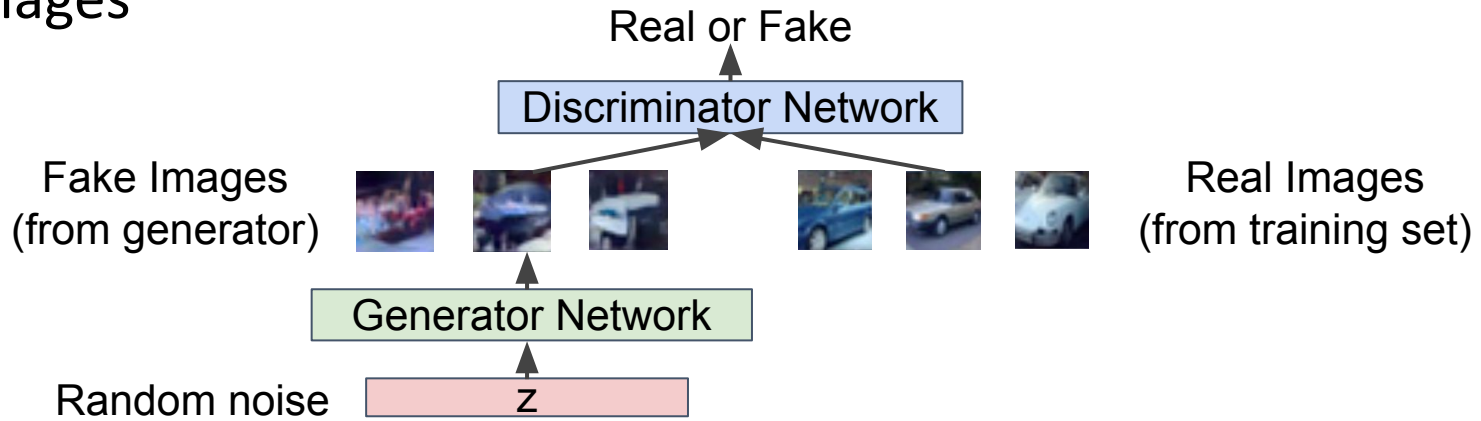
A: A neural network!



Training GANs: Two-player game

Generator network: try to fool the discriminator by generating real-looking images

Discriminator network: try to distinguish between real and fake images



Training GANs: Two-player game

Generator network: try to fool the discriminator by generating real-looking images
Discriminator network: try to distinguish between real and fake images

Train jointly in **minimax game**

Minimax objective function:

Discriminator outputs likelihood in (0,1) of real image

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

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)

Training GANs: Two-player game

Minimax objective function:

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

Alternate between:

1. **Gradient ascent** on discriminator

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

2. **Gradient descent** on generator

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

Training GANs: Two-player game

Minimax objective function:

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

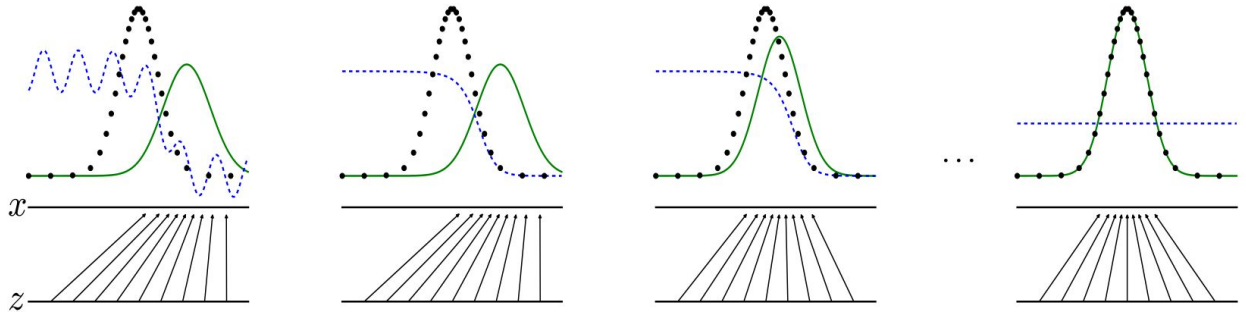
Alternate between:

1. **Gradient ascent** on discriminator

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

2. **Gradient descent** on generator

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$



Training GANs: Two-player game

Minimax objective function:

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

Alternate between:

1. **Gradient ascent** on discriminator

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

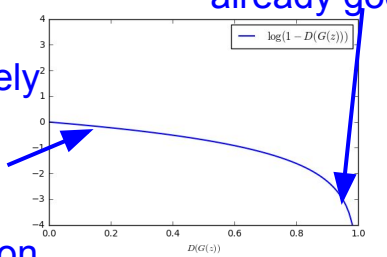
2. **Gradient descent** on generator

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

In practice, optimizing this generator objective does not work well!

Gradient signal dominated by region where sample is already good

When sample is likely fake, want to learn from it to improve generator. But gradient in this region is relatively flat!



Training GANs: Two-player game

Minimax objective function:

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

Alternate between:

1. **Gradient ascent** on discriminator

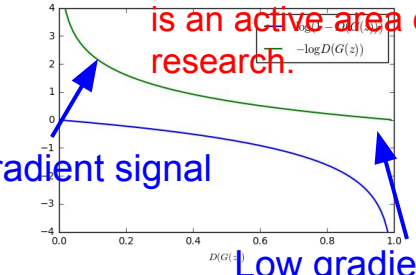
$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **Instead: Gradient ascent** on generator, different objective

$$\max_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(D_{\theta_d}(G_{\theta_g}(z)))$$

Instead of minimizing likelihood of discriminator being correct, now maximize likelihood of discriminator being wrong.
Same objective of fooling discriminator, but now higher gradient signal for bad samples => works much better! Standard in practice.

Aside: Jointly training two networks is challenging, can be unstable. Choosing objectives with better loss landscapes helps training, is an active area of research.



GAN Training Algorithm

- Update discriminator
  - Repeat for k steps:
    - Sample mini-batch of noise samples $z_1, \dots, z_m$ and mini-batch of real samples $x_1, \dots, x_m$
    - Update parameters of D by stochastic gradient ascent on

$$\frac{1}{m}\sum_{m}[\log D(x_m) + \log(1 - D(G(z_m)))]$$

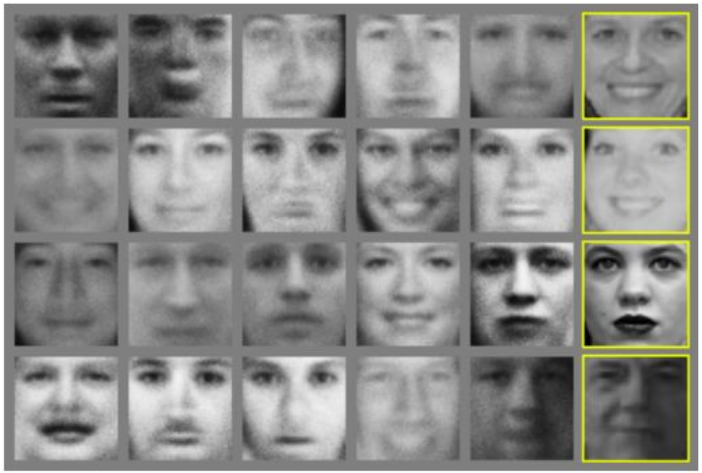
- Update generator
  - Sample mini-batch of noise samples $z_1, \dots, z_m$
  - Update parameters of G by stochastic gradient ascent on

$$\frac{1}{m}\sum_{m}\log D(G(z_m))$$

GAN Results

MNIST digits

Toronto Face Dataset



Nearest real image for sample to the left

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial nets, NIPS 2014

GAN Results

CIFAR-10 (FC networks)

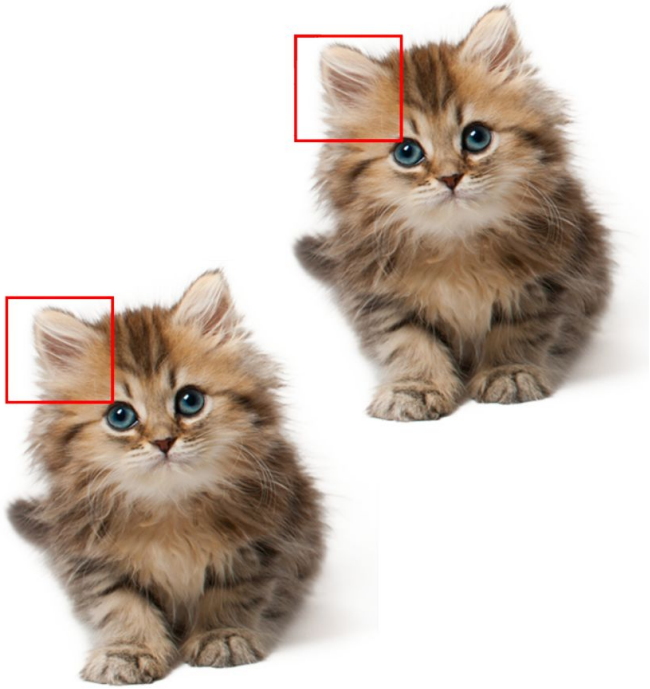
CIFAR-10 (conv networks)

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial nets, NIPS 2014

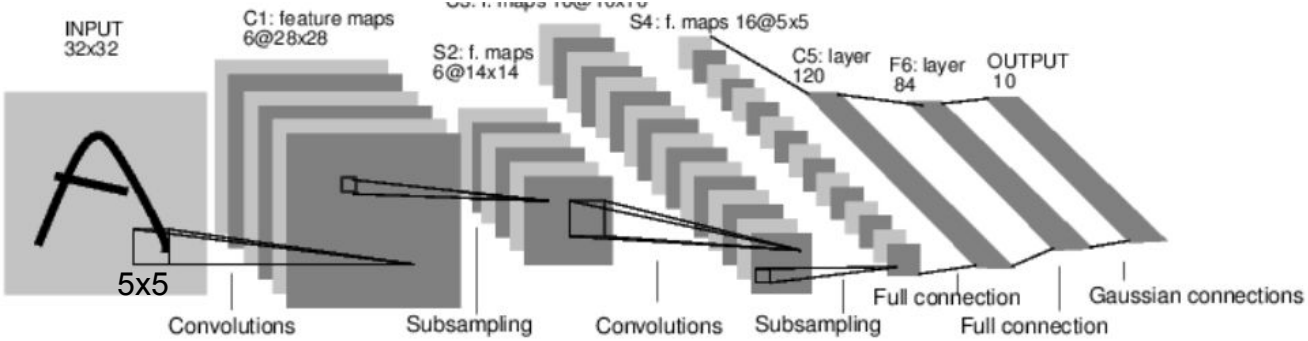
How to improve GAN?

Let’s go to basics!

Layers: convolution



Layers: convolution

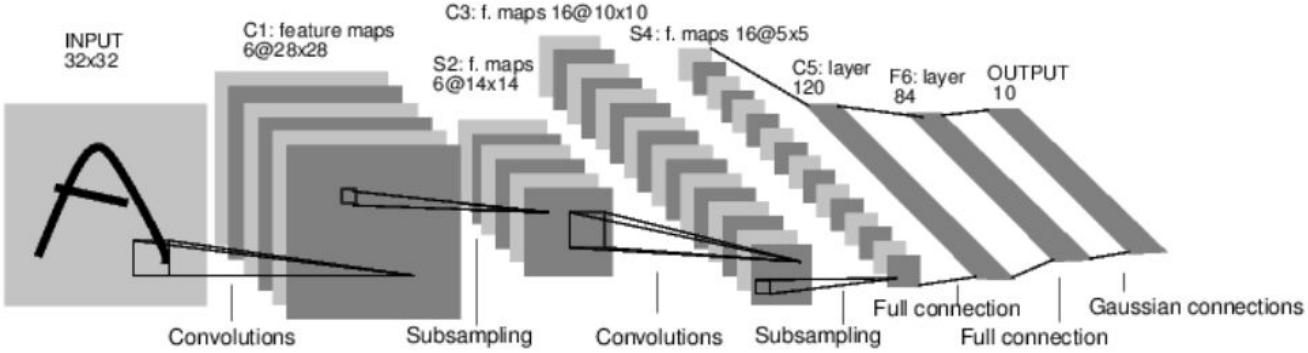


First layer:

- Without weight sharing:
- with weight sharing:

Y. LeCun, et al.: Gradient-Based Learning Applied to Document Recognition,1998

Layers: convolution



First layer:

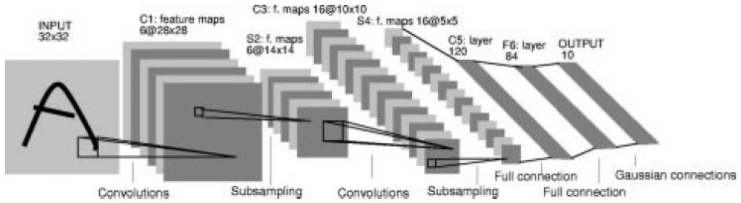
- Without weight sharing: 5x5x6x28x28 = 117600
- with weight sharing: 5x5x6 = 150

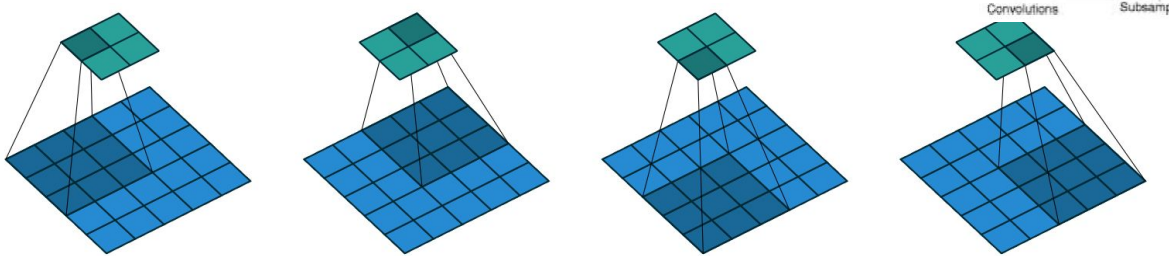
Y. LeCun, et al.: Gradient-Based Learning Applied to Document Recognition,1998

“De-convolution”

Architecture:

- Encoder: ConvNet architecture

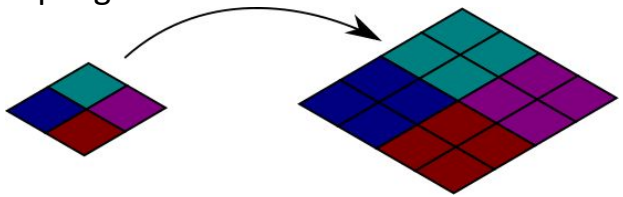




Convolving a 3 × 3 kernel over a 5 × 5 input using 2 × 2 strides

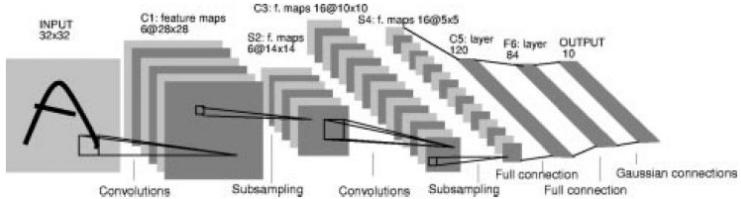
- Decoder: How to go from a vector to an image? How to “deconvolve”?

Up-sampling

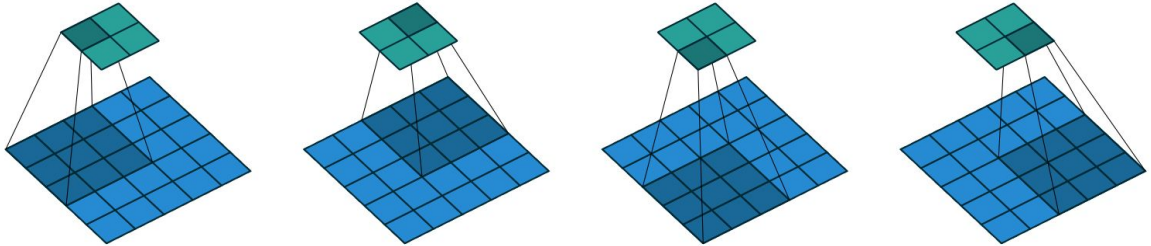


“De-convolution”

Architecture:



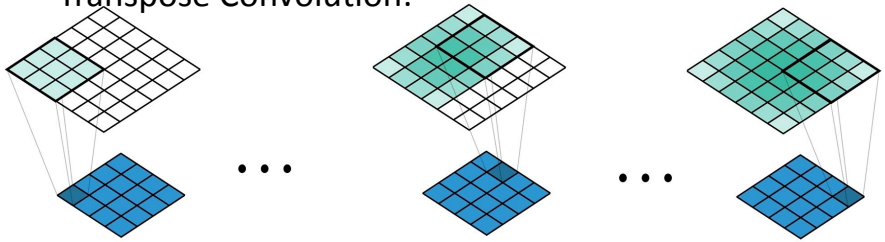
- Encoder: ConvNet architecture



Convolving a 3 × 3 kernel over a 5 × 5 input using 2 × 2 strides

- Decoder: How to go from a vector to an image? How to “deconvolve”?

Transpose Convolution:



A guide to convolution arithmetic for deep learning ,Vincent Dumoulin, Francesco Visin

Gradient descent

- Optimization problem: $\min_\theta = L_{tot}(\theta)$

- Gradient Descent:

  Until convergence:

$$\theta_{n+1} = \theta_n - \gamma \nabla L(\theta_n)$$

- Non convex problem -> Gradient descent does not work!

Gradient descent

- Optimization problem: $\min_\theta = L_{tot}(\theta)$
- Gradient Descent:

  Until convergence: $\theta_{n+1} = \theta_n - \gamma \nabla L(\theta_n)$

- Non convex problem -> Gradient descent does not work!
- Stochastic Gradient Descent:
  - We split the training in mini-batch: $L_{tot}(\theta) = \sum_b L_b(\theta)$

    Until convergence:

    For each batch b: $\theta_{n+1} = \theta_n - \gamma \nabla L_b(\theta_n)$

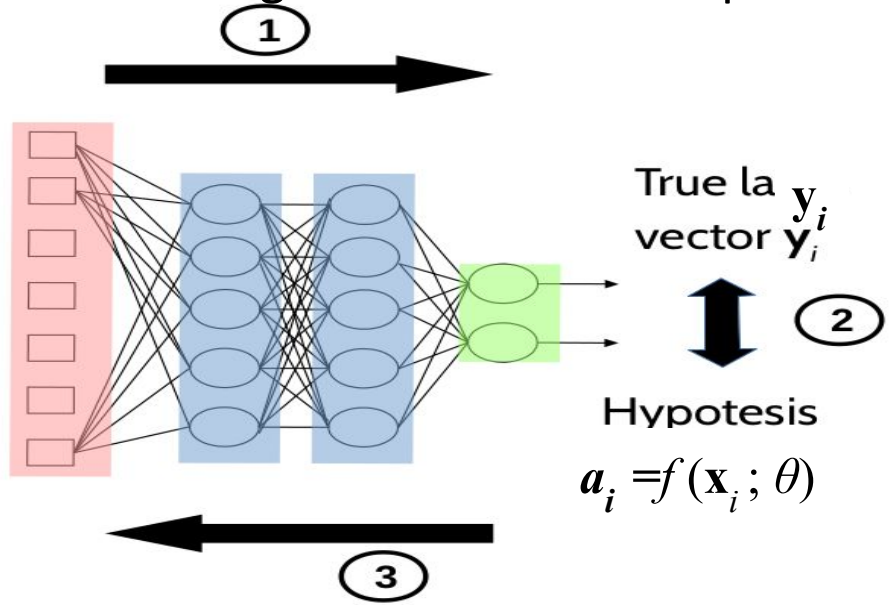
- How to compute the gradient efficiently?

1986 – Backpropagation

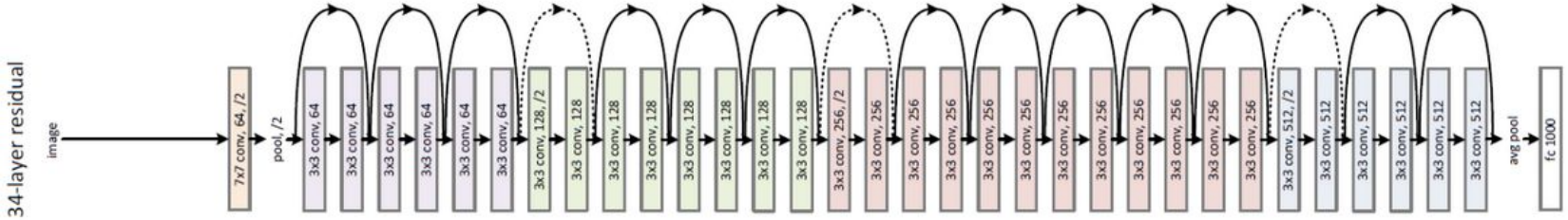
- Backpropagation revitalized the field
- Learning MLP for complicated functions can be solved
- Efficient algorithm which processes “large” training sets
- Allowed for complicated neural network architectures
- Today backpropagation is still at the core of neural network training

Backpropagation in a Nutshell

1. **Forward propagation**: sum inputs, produce activations, feed-forward
2. **Error estimation**.
3. **Back propagate** the error signal and used it to update weights

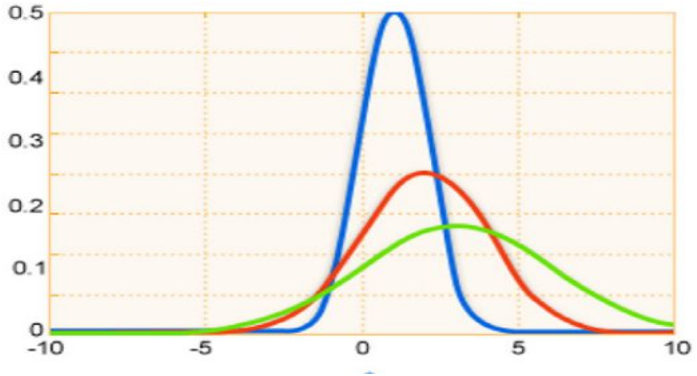


Layers: Batch-normalization

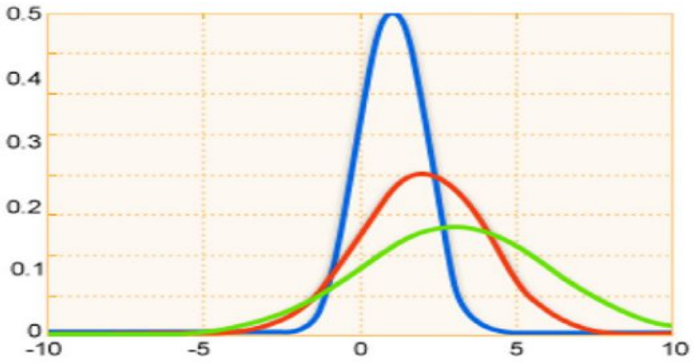


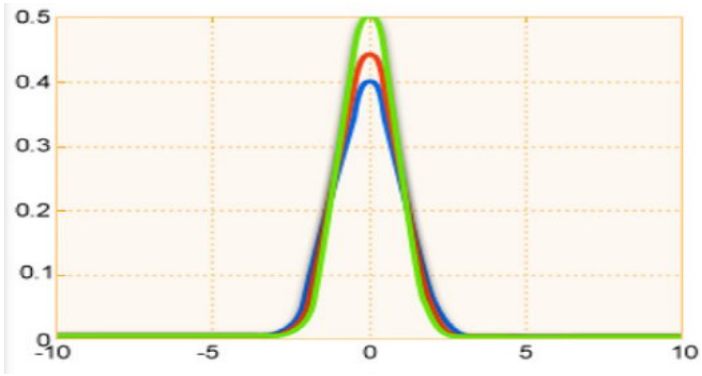
Parameter update:

$$\theta_{n+1} = \theta_n - \gamma \nabla L_b(\theta_n)$$



Layers: Batch-normalization





Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift
Sergey Ioffe, Christian Szegedy

Layers: Batch-normalization

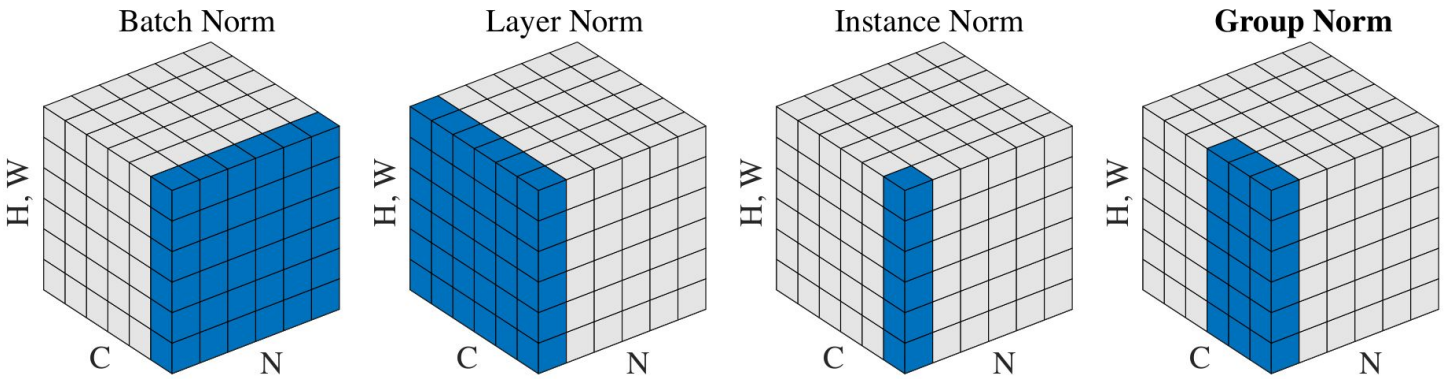
Advantages:

- Faster training
- Better optimization (lower final loss)
- Better generalization
- Specific usages in:
  - few-shot learning
  - domain adaptation
  - generation

Layers: Other normalizations

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

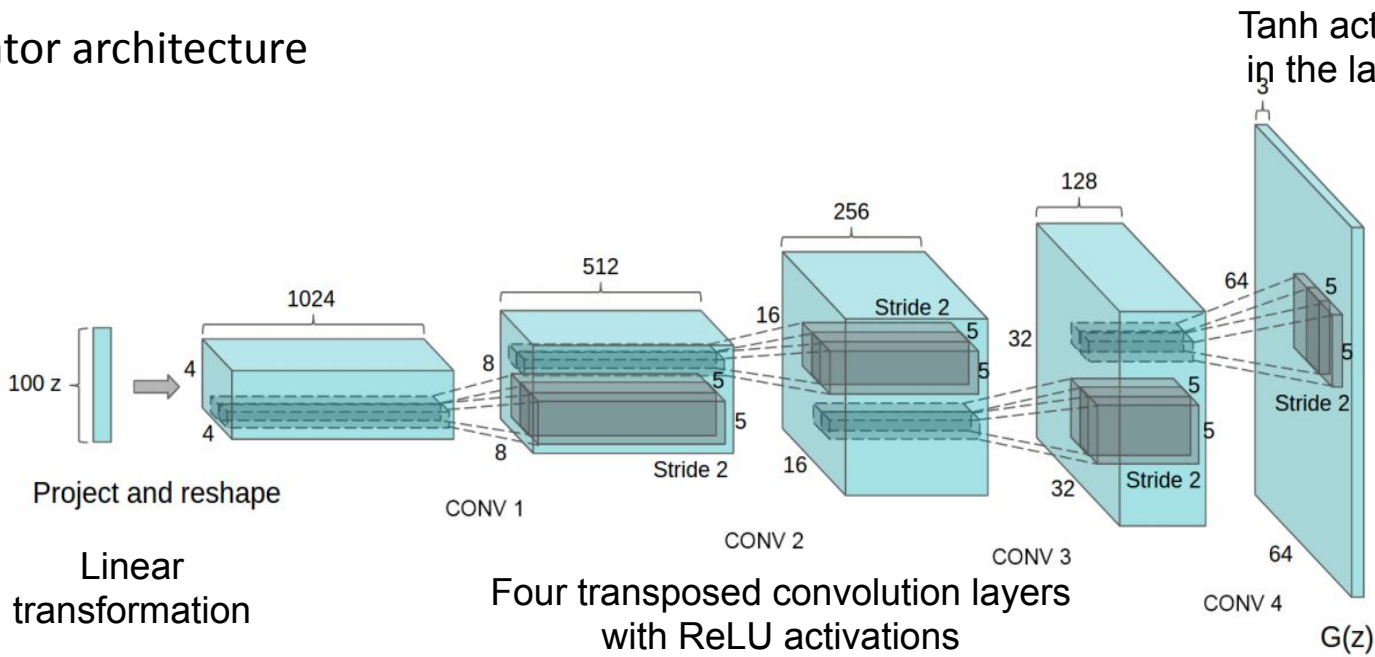
$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$



Group Normalization,Yuxin Wu, Kaiming He, ECCV 2018

DCGAN

- Propose principles for designing convolutional architectures for GANs, first model for large resolution image generation
- Generator architecture



A. Radford, L. Metz, S. Chintala, Unsupervised representation learning with deep convolutional generative adversarial networks, ICLR 2016

DCGAN

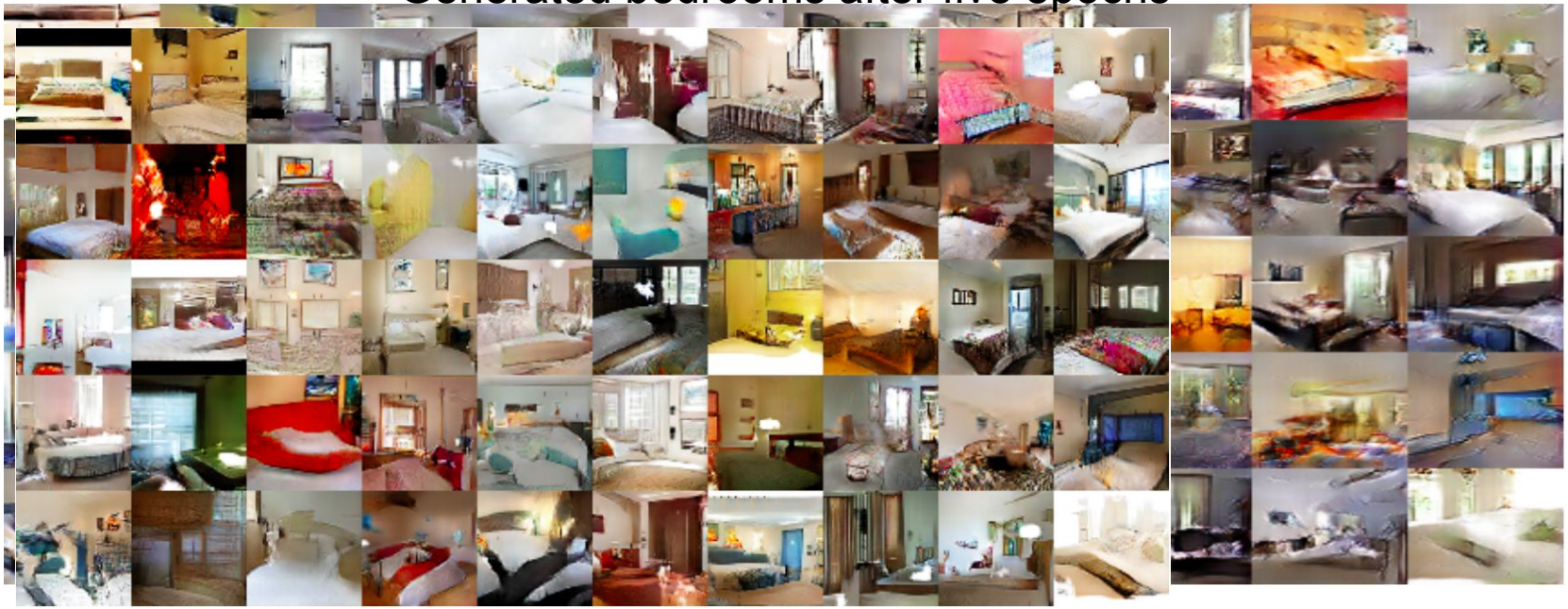
- Discriminator architecture
  - No pooling, only strided convolutions
  - Use Leaky ReLU activations (sparse gradients cause problems for training)
  - Use only one FC layer before the softmax output
  - Use batch normalization after most layers
    (in the generator also)

DCGAN results

Generated bedrooms after one epoch

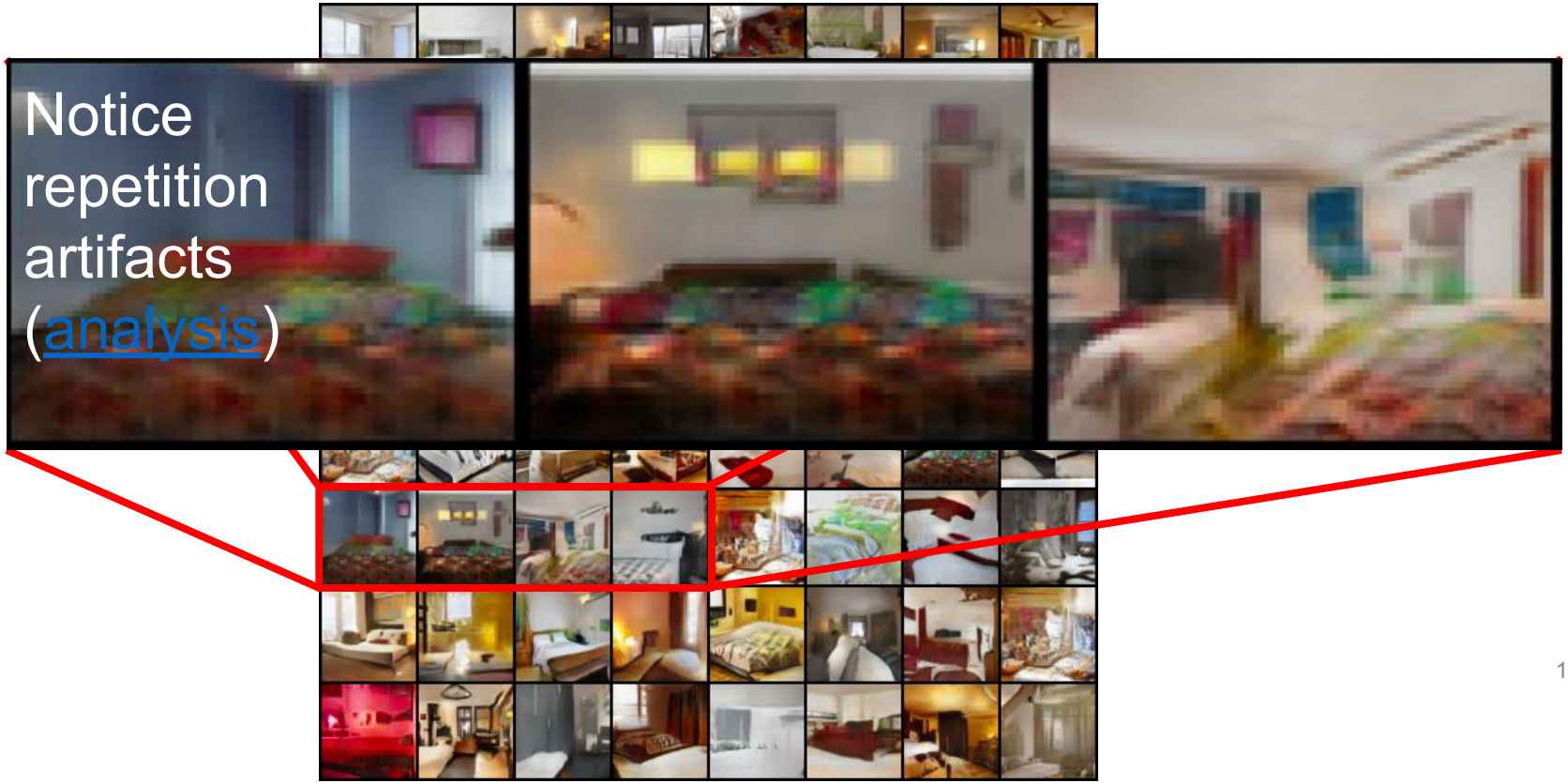
DCGAN results

Generated bedrooms after five epochs

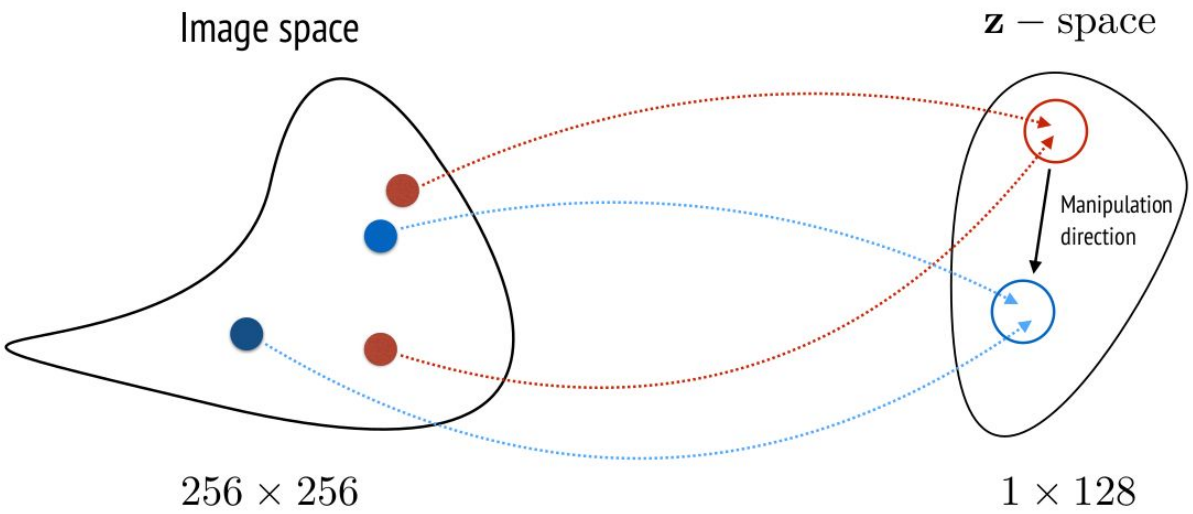


DCGAN results

Generated bedrooms from reference implementation



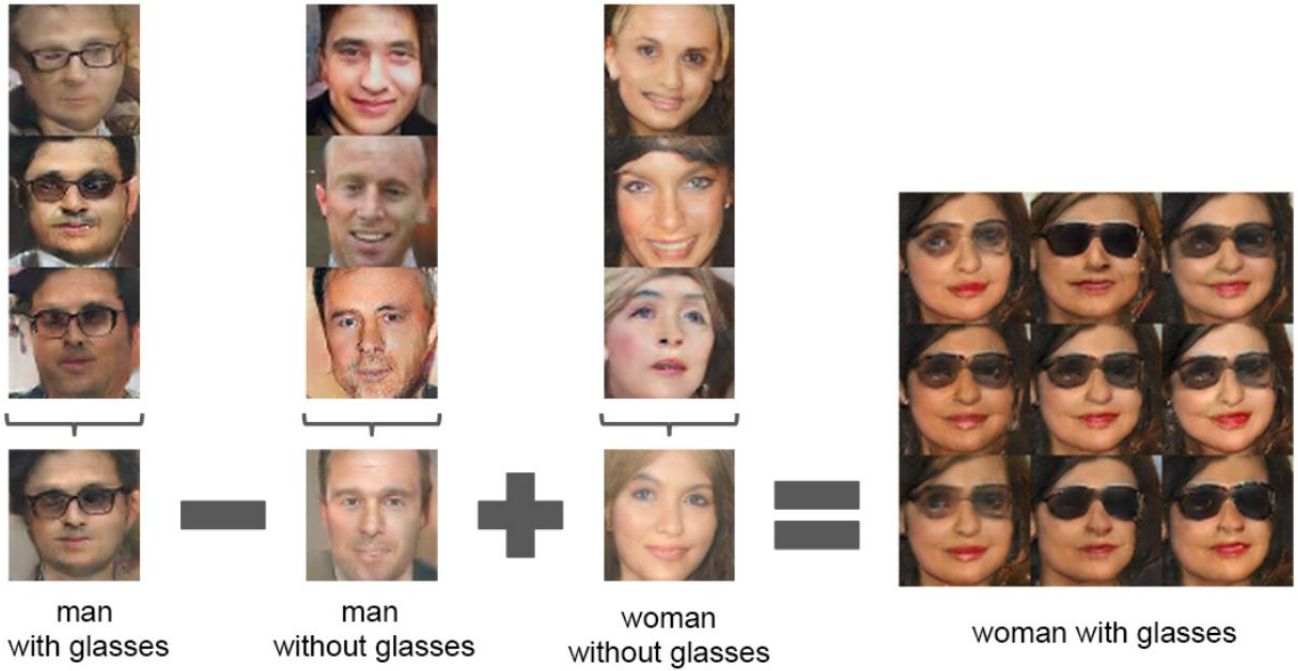
DCGAN results



DCGAN results

For each column, the Z vectors of samples are averaged. Arithmetic was performed on the mean vectors creating a new vector Y . The center sample on the right hand side is produce by feeding Y as input to the generator. To demonstrate the interpolation capabilities of the generator, uniform noise sampled with scale +-0.25 was added to Y to produce the 8 other samples

DCGAN results



First model to bring the concept of **disentanglement**.

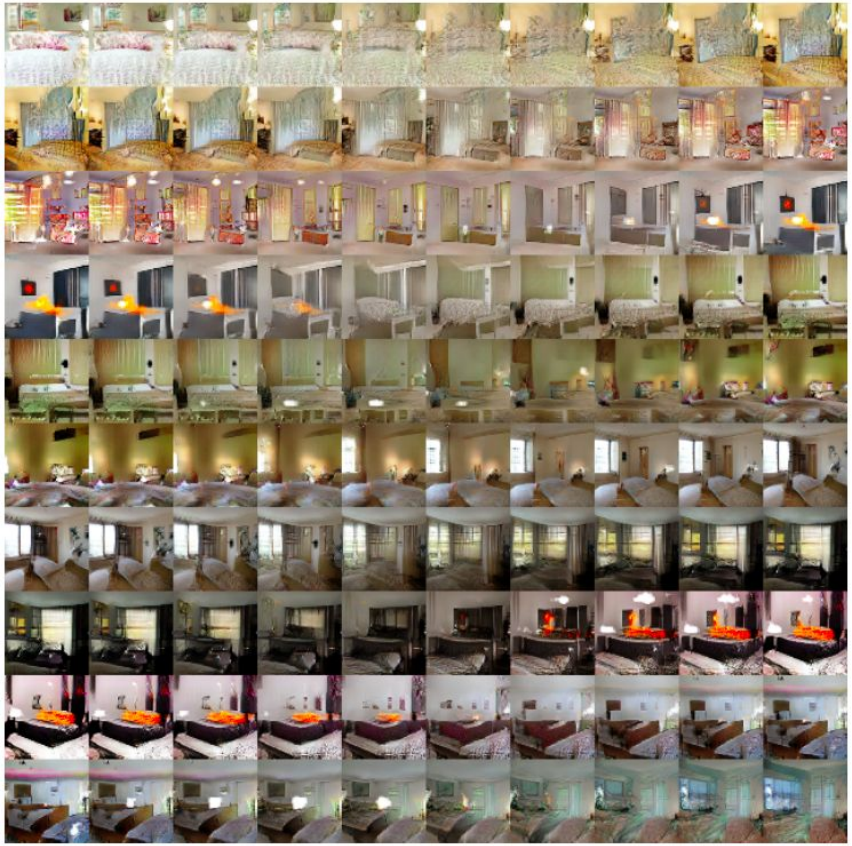
DCGAN results

- 

A ”turn” vector was created from four averaged samples of faces looking left vs looking right. By adding interpolations along this axis to random samples we were able to reliably transform their pose.

DCGAN results

Interpolation between different points in the z space



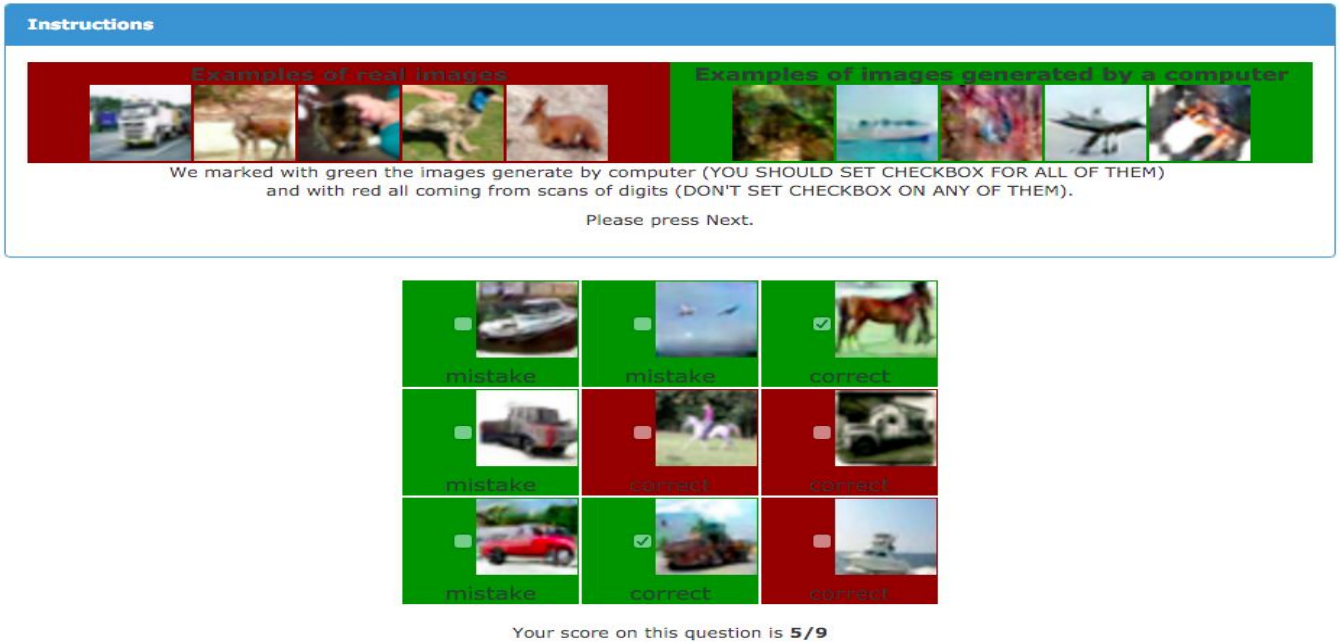
Evaluate GAN performance

- Showing pictures of samples is not enough, especially for simpler datasets like MNIST, CIFAR, faces, bedrooms, etc.
- We cannot directly compute the likelihoods of high-dimensional samples (real or generated) or compare their distributions
- Many GAN approaches claim mainly to improve stability, which is hard to evaluate
- For discussion, see Ian Goodfellow Twitter thread

Theis, L., Oord, A. V. D., & Bethge, M. (2016). A note on the evaluation of generative models. ICLR

Evaluate GAN performance

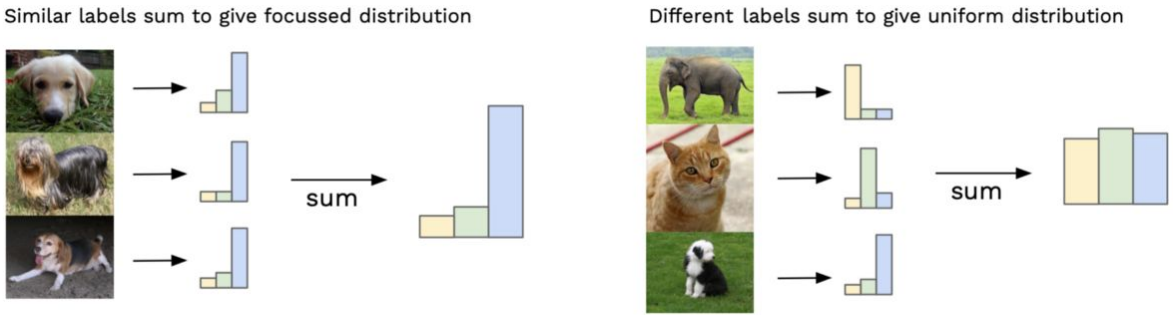
- Human performance



T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, X. Chen, Improved techniques for training GANs, NIPS 2016

Inception Score

- Key idea: generators should produce images with a **variety** of **recognizable** object classes
- Two criteria:
  - A human, looking at a separate image, would be able to confidently determine what's in there (**saliency**).
  - A human, looking at a set of various images, would say that the set has lots of different objects (**diversity**).



T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, X. Chen, Improved techniques for training GANs, NIPS 2016

Inception Score

- Key idea: generators should produce images with a **variety** of **recognizable** object classes
- Defined as:

$$\mathrm{IS}(G) = \exp\left(\mathbb{E}_{\mathbf{x} \sim p_g} D_{KL}(p(y|\mathbf{x}) \parallel p(y)) \right),$$

where $p(y|x)$ is the posterior label distribution returned by an image classifier (*e.g.*, InceptionNet) for sample x

- If the image contains a recognizable object, entropy of $p(y|x)$ should be low
- If generator generates images of diverse objects, the marginal distribution $p(y)$ should have high entropy
- Disadvantage: a GAN that simply memorizes the training data (overfitting) or outputs a single image per class (mode dropping) could still score well

T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, X. Chen, Improved techniques for training GANs, NIPS 2016

Fréchet Inception Distance

- Key idea: fit simple distributions to statistics of feature activations for real and generated data; estimate divergence parametrically
  - Pass generated samples through a network (InceptionNet), compute activations for a chosen layer
  - Estimate multivariate mean and covariance of activations, compute Fréchet distance to those of real data

$$\texttt{FID}(x, g) = ||\mu_x - \mu_g||_2^2 + \text{Tr}(\Sigma_x + \Sigma_g - 2(\Sigma_x\Sigma_g)^{\frac{1}{2}}),$$

- Advantages: correlated with visual quality of samples and human judgment
- Disadvantage: cannot detect overfitting

M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, S. Hochreiter, GANs trained by a two time-scale update rule converge to a local Nash equilibrium, NIPS 2017

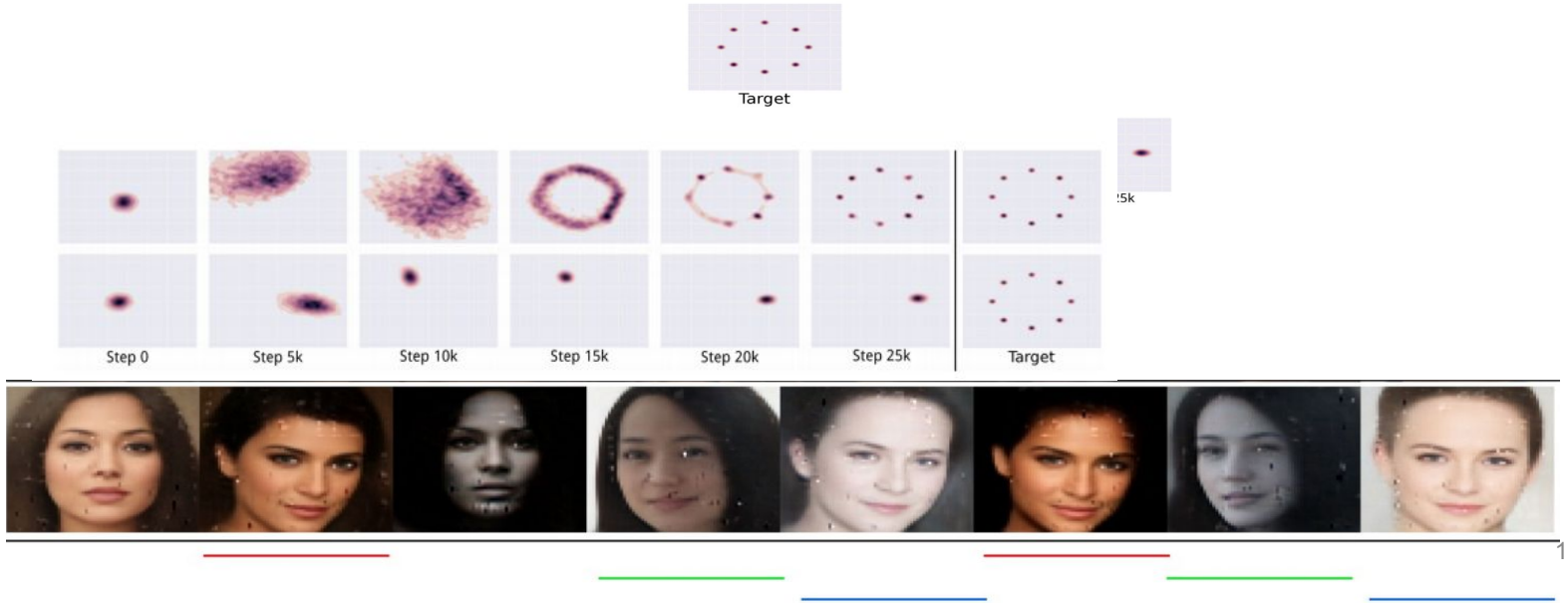
Issues with GAN training

- Stability
  - Parameters can oscillate or diverge, generator loss does not correlate with sample quality
  - Behavior very sensitive to hyper-parameter selection
  - Proposed several improvements, mostly changing the loss function

Tutorial on GAN: summary https://arxiv.org/pdf/1701.00160.pdf
https://medium.com/@jonathan_hui/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b

Issues with GAN training

- Mode collapse
  - Generator ends up modeling only a small subset of the training data



https://medium.com/@jonathan_hui/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b

Training Tricks

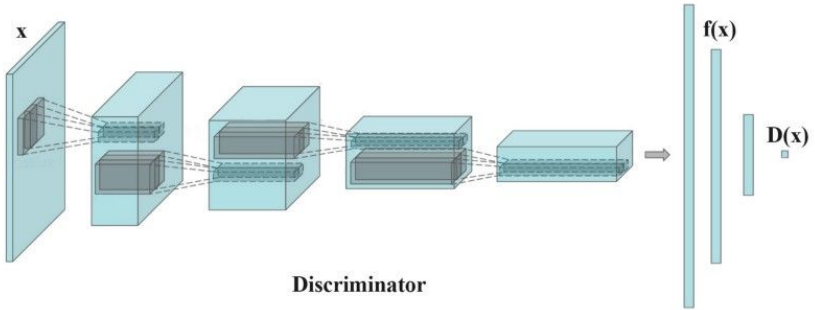
- Feature matching
- Use virtual batch norm
- Mini-batch discrimination (avoid mode collapsing by looking at more samples)
- Use label informations in the discriminator (C+1 categories)
- Label smoothing
- Historical averaging (introducing an additional cost function)

T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, X. Chen, Improved techniques for training GANs, NIPS 2016

Training Tricks

- Feature matching
- Issue:
  - The generator tries to find the best image to fool the discriminator.
  - The "best" image keeps changing when both networks counteract their opponent.
  - The model does not converge and mode collapses.
- Add a loss for feature matching (expands the goal from beating the opponent to matching features in real images)

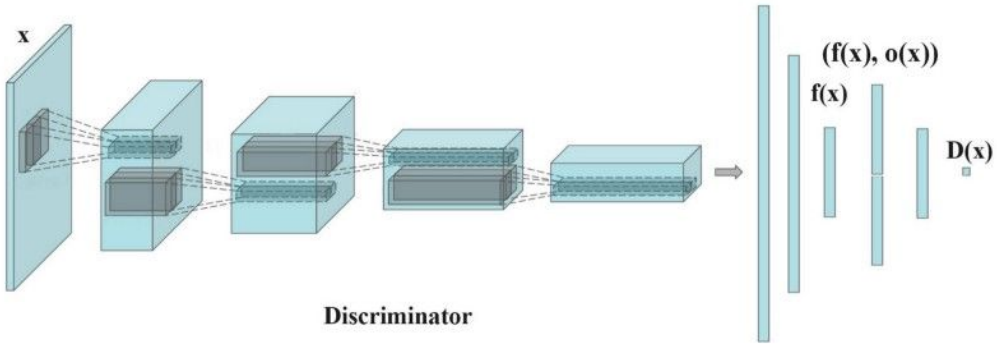
$$||\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\mathbf{f}(\boldsymbol{x}) - \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\mathbf{f}(G(\boldsymbol{z}))||_2^2$$



T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, X. Chen, Improved techniques for training GANs, NIPS 2016

Training Tricks

- Mini-batch discrimination
- Compute the similarity of the image ***x*** with images in the same batch. Append the similarity ***o(x)*** in one of the dense layers in the discriminator to classify whether this image is real or generated.
- If the mode starts to collapse, the similarity of generated images increases. The discriminator can use ***o(x)*** to detect generated images and penalize the generator if mode is collapsing.



T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, X. Chen, Improved techniques for training GANs, NIPS 2016

Training Tricks

- Virtual batch norm
- Issue: Due to batch norm the generated images are not independent
- Idea: samples a reference batch before the training to compute the normalization parameters (μ and σ) and combine a reference batch with the current batch to compute the normalization parameters.

T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, X. Chen, Improved techniques for training GANs, NIPS 2016

Training Tricks

- Feature matching
- Use virtual batch norm
- Mini-batch discrimination (avoid mode collapsing by looking at more samples)
- Use label informations in the discriminator (C+1 categories)
- Label smoothing
- Historical averaging (introducing an additional cost function)

$$||\boldsymbol{\theta} - \frac{1}{t}\sum_{i=1}^{t}\boldsymbol{\theta}[i]||^2$$

T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, X. Chen, Improved techniques for training GANs, NIPS 2016

GAN Results

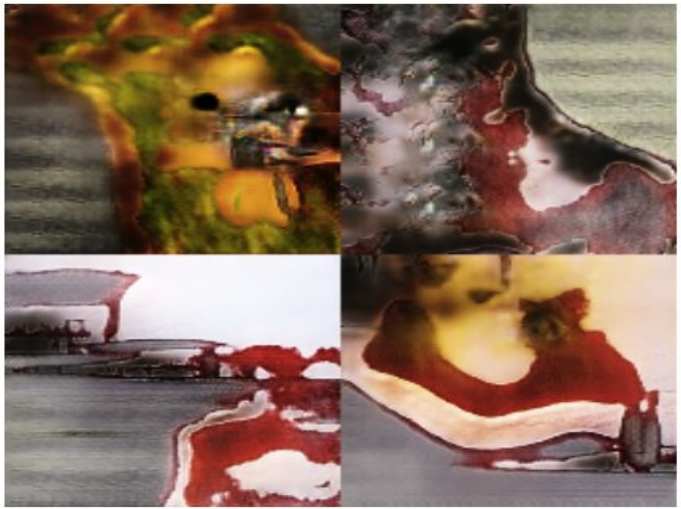
Original CIFAR-10 vs. Generated CIFAR-10 samples

T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, X. Chen, Improved techniques for training GANs, NIPS 2016

GAN Results

- ImageNet Results (128x128 pixels)

DCGAN

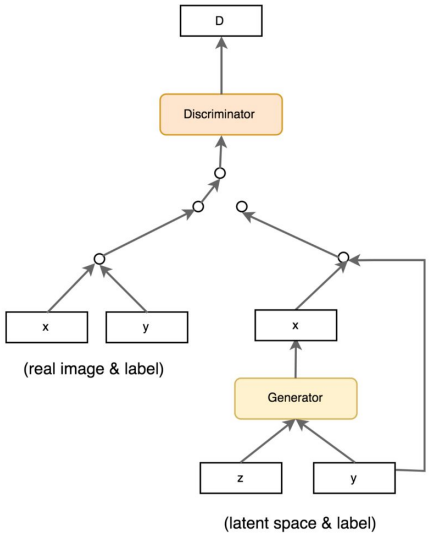


Improved DCGAN

T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, X. Chen, Improved techniques for training GANs, NIPS 2016

CGAN & InfoGAN

- Conditional GAN (CGAN)
  - Same loss as the original GAN
  - Use label info (1-hot encoding) both in generator and discriminator
  - Label as extension to the latent space



$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x}|\boldsymbol{y})] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z}|\boldsymbol{y})))].$$

CGAN & InfoGAN

- Info GAN (label not given, treated as latent factor):
- Information Maximizing Generative Adversarial Nets

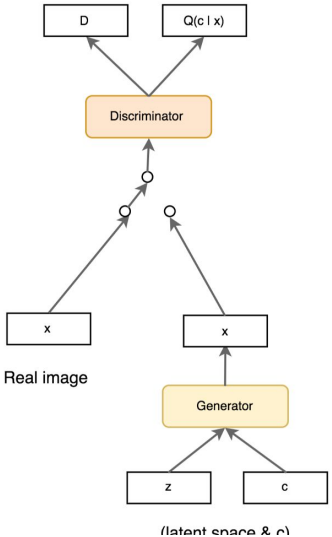
$$I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$$

- X and Y are independent, then I(X; Y) = 0
- X is completely determined by Y, then H(Y|X)=0, I(X; Y) = H(Y)
  Then I is maximized

Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., & Abbeel, P. (2016). Infogan: Interpretable representation learning by information maximizing generative adversarial nets. NIPS

CGAN & InfoGAN

- Info GAN (label not given, treated as latent factor):
  - Additional output for the discriminator $Q(c|x)$
  - Mutual information $I(c;x)$ measure how much we know c if we know x. It should be high.
  - The info in the latent code c should not be lost in the generation.



$$\min_G \max_D V_{infoGAN}(D, G) = V_{GAN}(D, G) - \lambda I(c; G(z, c))$$

$$V_{GAN}(D, G) \equiv \mathbb{E}_{x \sim p_{data}} logD(x) + \mathbb{E}_{z \sim p_z(z)} log(1 - D(G(z, c)))$$

$$\begin{aligned} I(c; G(z, c)) &= H(c) - H(c|G(z, c)) \\ &= \mathbb{E}_{x \sim G(z,c)}[\mathbb{E}_{c \sim P(c|x)}[logP(c|x)]] + H(c) \\ &\geq \mathbb{E}_{x \sim G(z,c)}[\mathbb{E}_{c \sim P(c|x)}[logQ(c|x)]] + H(c) \end{aligned}$$

Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., & Abbeel, P. (2016). Infogan: Interpretable representation learning by information maximizing generative adversarial nets. NIPS.

InfoGAN Results

(a) Varying c_1 on InfoGAN (Digit type)

(b) Varying c_1 on regular GAN (No clear meaning)

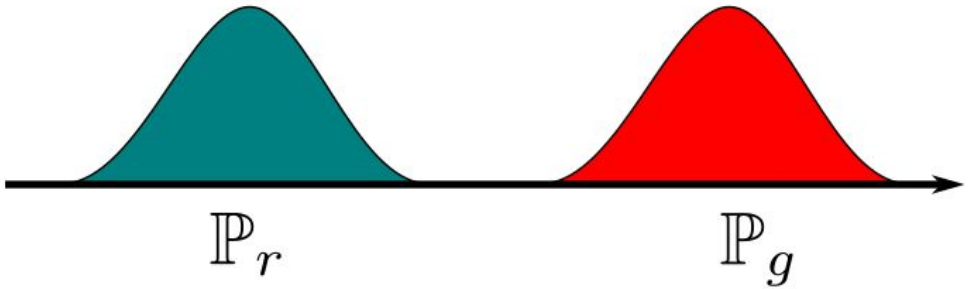
Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., & Abbeel, P. (2016). Infogan: Interpretable representation learning by information maximizing generative adversarial nets. NIPS.

Wasserstein GAN: towards better losses

The *Kullback-Leibler* (KL) divergence

$$KL(\mathbb{P}_r \| \mathbb{P}_g) = \int \log \left(\frac{P_r(x)}{P_g(x)} \right) P_r(x) d\mu(x)$$

Problem when the distributions have $P_g(x)=0$ and $P_r(x)>0$:



$$KL(\mathbb{P}_r || \mathbb{P}_g) = +\infty$$

Wasserstein GAN, Martin Arjovsky, Soumith Chintala, Léon Bottou

Wasserstein GAN: towards better losses

The *Jensen-Shannon* (JS) divergence

$$JS(\mathbb{P}_r, \mathbb{P}_g) = KL(\mathbb{P}_r \| \mathbb{P}_m) + KL(\mathbb{P}_g \| \mathbb{P}_m)$$

where $\mathbb{P}_m$ is the mixture $(\mathbb{P}_r + \mathbb{P}_g)/2$.

Minimizing the GAN objective function with an optimal discriminator is equivalent to minimizing the JS-divergence

Wasserstein GAN, Martin Arjovsky, Soumith Chintala, Léon Bottou

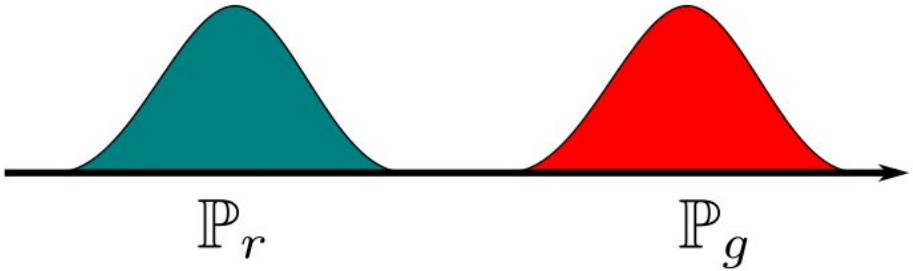
Wasserstein GAN: towards better losses

The *Jensen-Shannon* (JS) divergence

$$JS(\mathbb{P}_r, \mathbb{P}_g) = KL(\mathbb{P}_r \| \mathbb{P}_m) + KL(\mathbb{P}_g \| \mathbb{P}_m)$$

where $\mathbb{P}_m$ is the mixture $(\mathbb{P}_r + \mathbb{P}_g)/2$.

Minimizing the GAN objective function with an optimal discriminator is equivalent to minimizing the JS-divergence



$$JS(\mathbb{P}_r, \mathbb{P}_g) = 2log(2)$$

Wasserstein GAN, Martin Arjovsky, Soumith Chintala, Léon Bottou

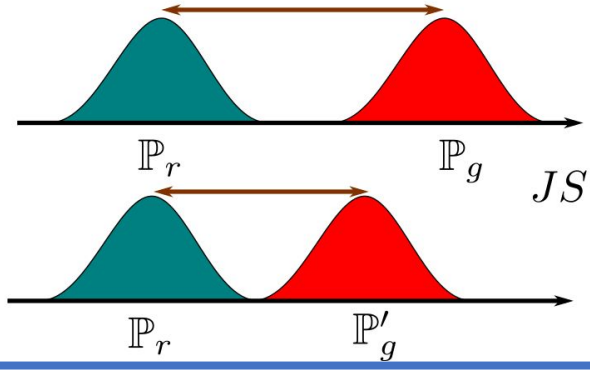
Wasserstein GAN: towards better losses

The *Jensen-Shannon* (JS) divergence

$$JS(\mathbb{P}_r, \mathbb{P}_g) = KL(\mathbb{P}_r \| \mathbb{P}_m) + KL(\mathbb{P}_g \| \mathbb{P}_m)$$

where $\mathbb{P}_m$ is the mixture $(\mathbb{P}_r + \mathbb{P}_g)/2$.

Problem:



$$JS(\mathbb{P}_r, \mathbb{P}_g) = JS(\mathbb{P}_r, \mathbb{P}'_g)$$

The *Earth-Mover* (EM) distance or Wasserstein-1

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}\left[\|x - y\| \right]$$

Wasserstein GAN: towards better losses

The *Earth-Mover* (EM) distance or Wasserstein-1

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}\left[\, \|x - y\| \,\right]$$

We can show:

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_\theta}[f(x)]$$

where the supremum is over all the 1-Lipschitz functions $f : \mathcal{X} \to \mathbb{R}$.

Wasserstein GAN, Martin Arjovsky, Soumith Chintala, Léon Bottou

Cédric Villani.Optimal Transport: Old and New. Grundlehren der mathema-tischen Wissenschaften

Wasserstein GAN: towards better losses

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim \mathbb{P}_r}[f(x)] - \mathbb{E}_{x \sim \mathbb{P}_\theta}[f(x)]$$

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.

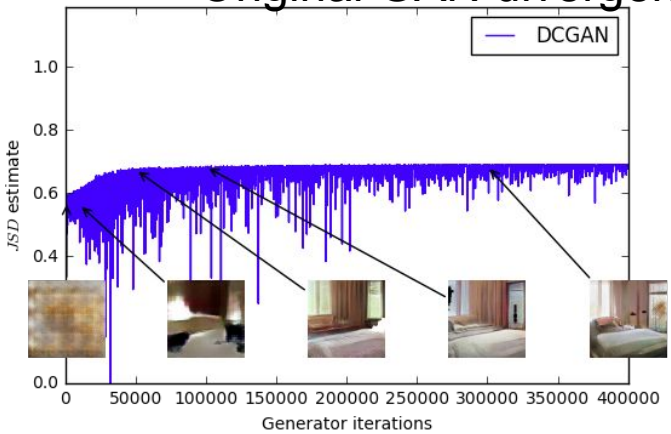
Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^m \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^m \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^m f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^m f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^m \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^m f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

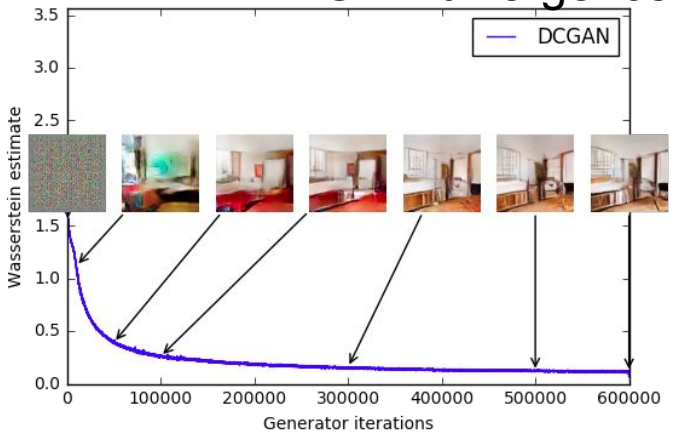
Wasserstein GAN

- Objective function value is more meaningfully related to quality of generator output

Original GAN divergence



WGAN divergence



Wasserstein GAN: Improved Training (WGAN_GP)

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim \mathbb{P}_r}[f(x)] - \mathbb{E}_{x \sim \mathbb{P}_\theta}[f(x)]$$

Weight clipping: $w \leftarrow \text{clip}(w, -c, c)$

Gradient penalty (WGAN-GP):

$$L = \underbrace{\mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}} \sim \mathbb{P}_g}[D(\tilde{\boldsymbol{x}})] - \mathop{\mathbb{E}}_{\boldsymbol{x} \sim \mathbb{P}_r}[D(\boldsymbol{x})]}_{\text{Original critic loss}} + \underbrace{\lambda \mathop{\mathbb{E}}_{\hat{\boldsymbol{x}} \sim \mathbb{P}_{\hat{\boldsymbol{x}}}}\left[(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1)^2\right]}_{\text{Our gradient penalty}}.$$

Improved Training of Wasserstein GANs, Gulrajani et.al. Nips 2017

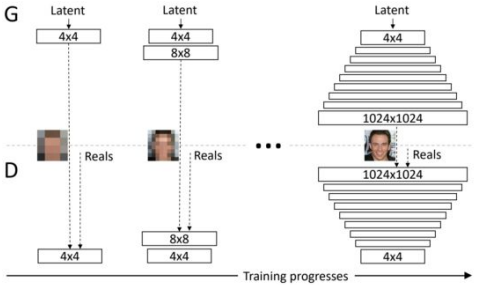
Lost functions

Name	Value Function
GAN	$L_D^{GAN} = E[\log(D(x))] + E[\log(1 - D(G(z)))]$ $L_G^{GAN} = E[\log(D(G(z)))]$
LSGAN	$L_D^{LSGAN} = E[(D(x) - 1)^2] + E[D(G(z))^2]$ $L_G^{LSGAN} = E[(D(G(z)) - 1)^2]$
WGAN	$L_D^{WGAN} = E[D(x)] - E[D(G(z))]$ $L_G^{WGAN} = E[D(G(z))]$ $W_D \leftarrow clip\_by\_value(W_D, -0.01, 0.01)$
WGAN_GP	$L_D^{WGAN\_GP} = L_D^{WGAN} + \lambda E[(\lvert\nabla D(\alpha x - (1 - \alpha G(z)))\rvert - 1)^2]$ $L_G^{WGAN\_GP} = L_G^{WGAN}$
DRAGAN	$L_D^{DRAGAN} = L_D^{GAN} + \lambda E[(\lvert\nabla D(\alpha x - (1 - \alpha x_p))\rvert - 1)^2]$ $L_G^{DRAGAN} = L_G^{GAN}$
CGAN	$L_D^{CGAN} = E[\log(D(x, c))] + E[\log(1 - D(G(z), c))]$ $L_G^{CGAN} = E[\log(D(G(z), c))]$
infoGAN	$L_{D,Q}^{infoGAN} = L_D^{GAN} - \lambda L_I(c, c')$ $L_G^{infoGAN} = L_G^{GAN} - \lambda L_I(c, c')$
ACGAN	$L_{D,Q}^{ACGAN} = L_D^{GAN} + E[P(class = c\lvert x)] + E[P(class = c\lvert G(z))]$ $L_G^{ACGAN} = L_G^{GAN} + E[P(class = c\lvert G(z))]$
EBGAN	$L_D^{EBGAN} = D_{AE}(x) + \max(0, m - D_{AE}(G(z)))$ $L_G^{EBGAN} = D_{AE}(G(z)) + \lambda \cdot PT$
BEGAN	$L_D^{BEGAN} = D_{AE}(x) - k_t D_{AE}(G(z))$ $L_G^{BEGAN} = D_{AE}(G(z))$ $k_{t+1} = k_t + \lambda(\gamma D_{AE}(x) - D_{AE}(G(z)))$

https://towardsdatascience.com/gan-ways-to-improve-gan-performance-acf37f9f59b

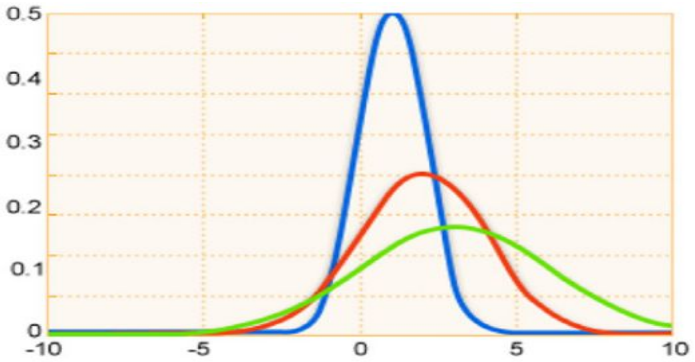
High Resolution

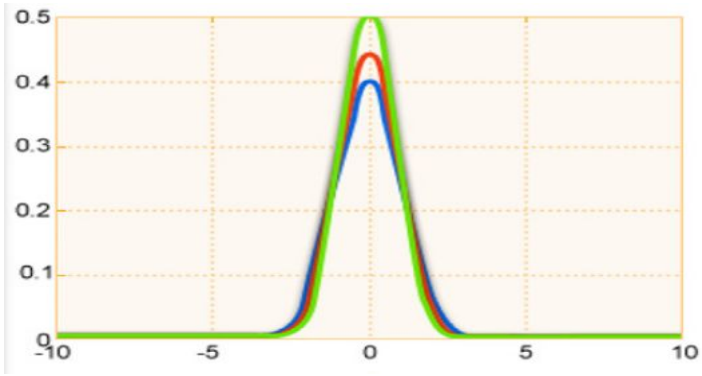
Progressive GAN (2018)



BigGAN (2018)

Inference-normalization





Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

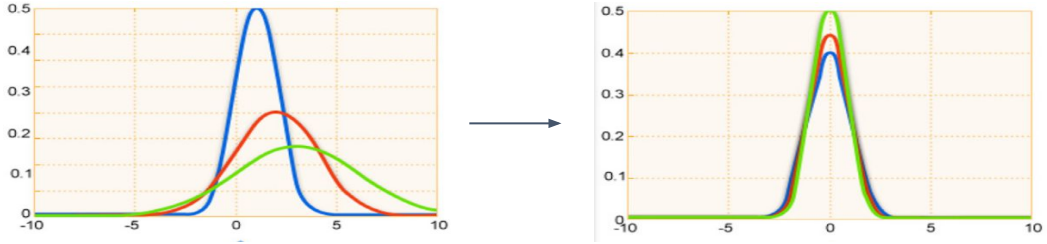
$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

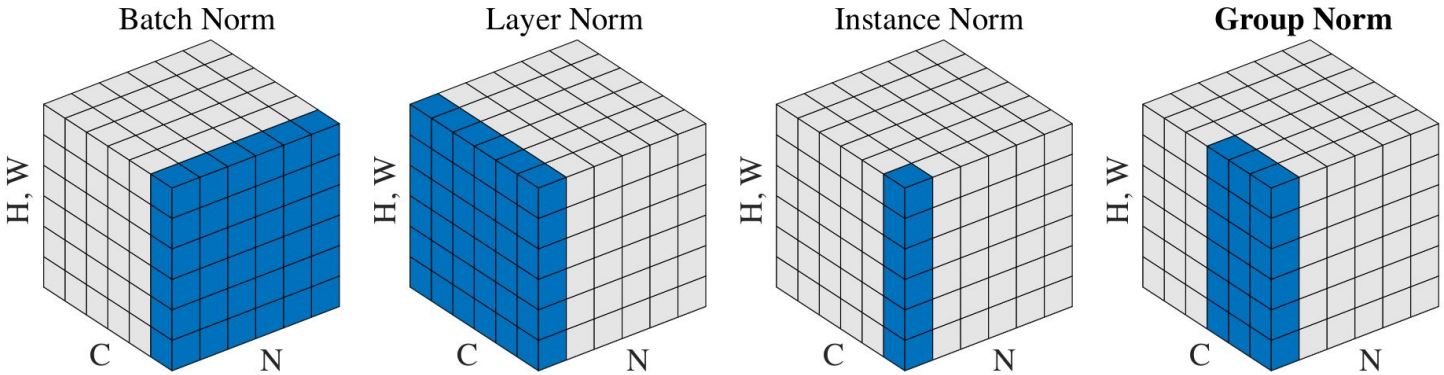
Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift
Sergey Ioffe, Christian Szegedy

Inference-normalization

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$$



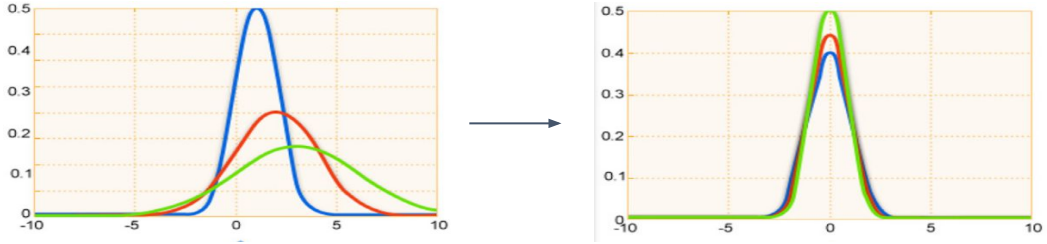


Group Normalization,Yuxin Wu, Kaiming He, ECCV 2018

Why inference-normalization?

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$$



(a) Content image.

(c) Low contrast content image.

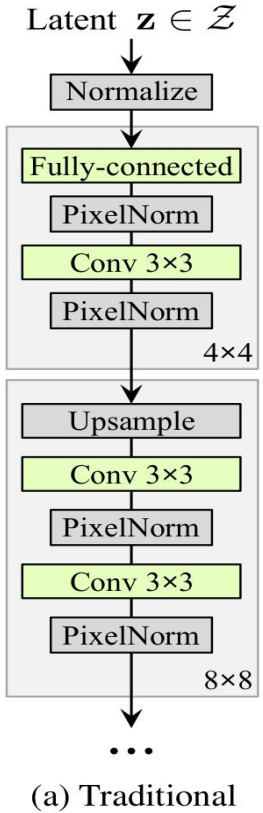
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv$$

The two images are identical after instance normalization. We obtain style invariant representations.

Instance Normalization: The Missing Ingredient for Fast Stylization,Dmitry Ulyanov, Andrea Vedaldi, Victor Lempitsky

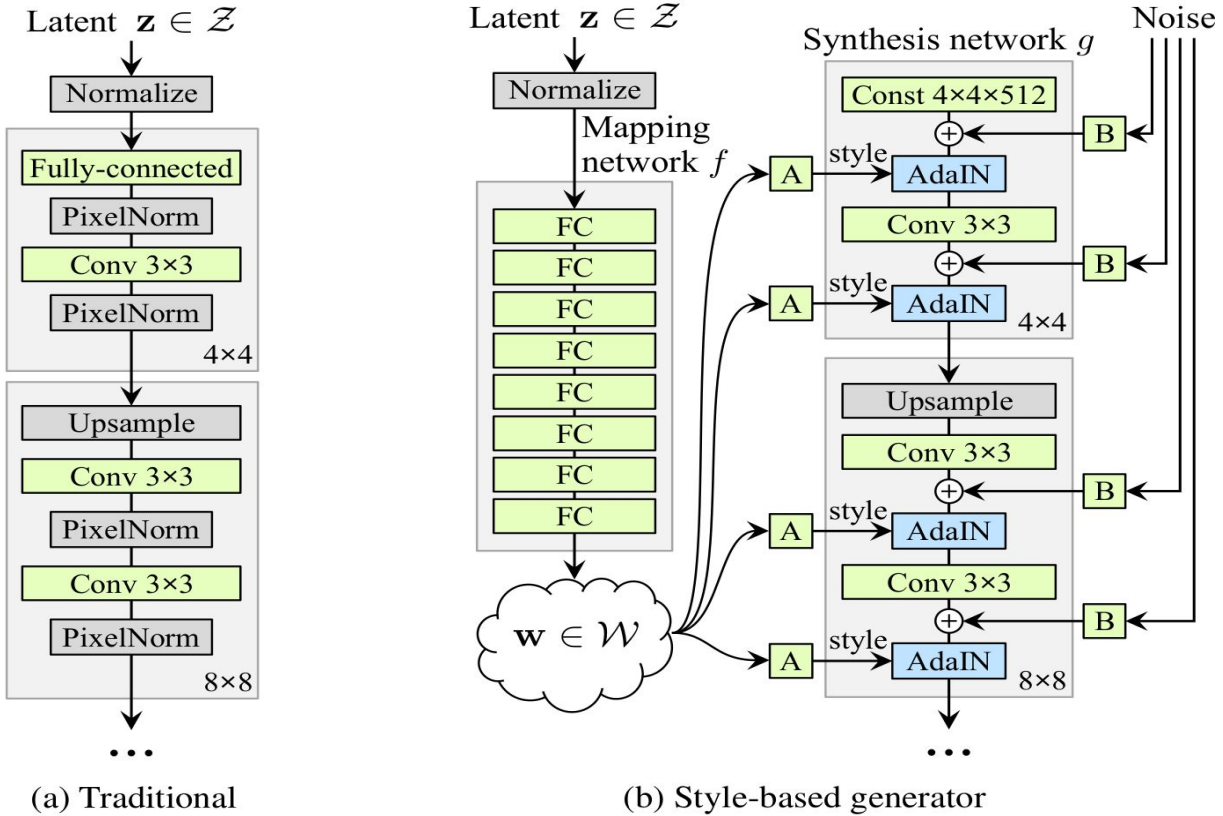
Style-Based GAN



(a) Traditional

A Style-Based Generator Architecture for Generative Adversarial Networks, Karras et.al. CVPR 2019

Style-Based GAN

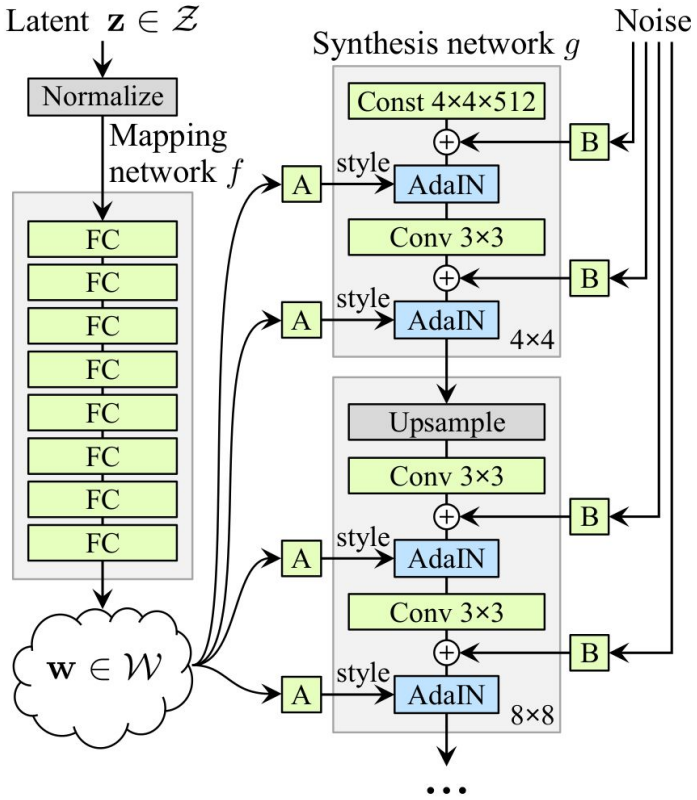


(a) Traditional

(b) Style-based generator

A Style-Based Generator Architecture for Generative Adversarial Networks, Karras et.al. CVPR 2019

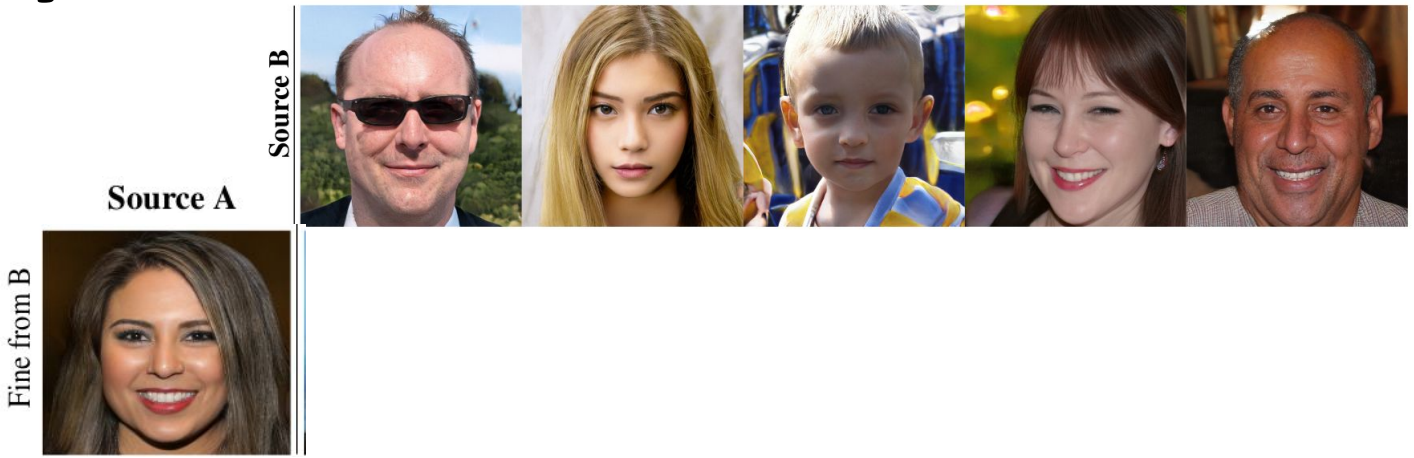
Style-Based GAN



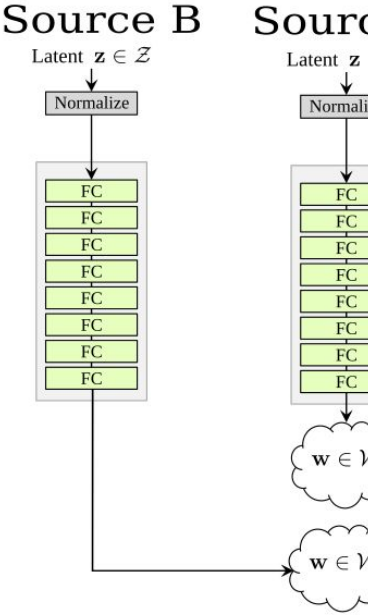
(b) Style-based generator

$$\text{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{s,i} \frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i},$$

Style-Based GAN



Style-Based GAN



Style-Based GAN

Style-Based GAN

Style-Based GAN

Training GANs: Recap

Training GANs: Recap

Minimax objective function:

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

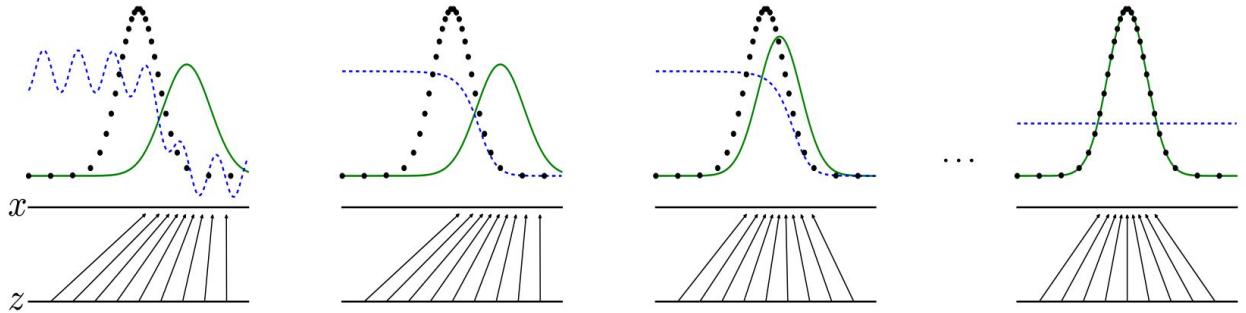
Alternate between:

1. **Gradient ascent** on discriminator

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

2. **Gradient descent** on generator

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

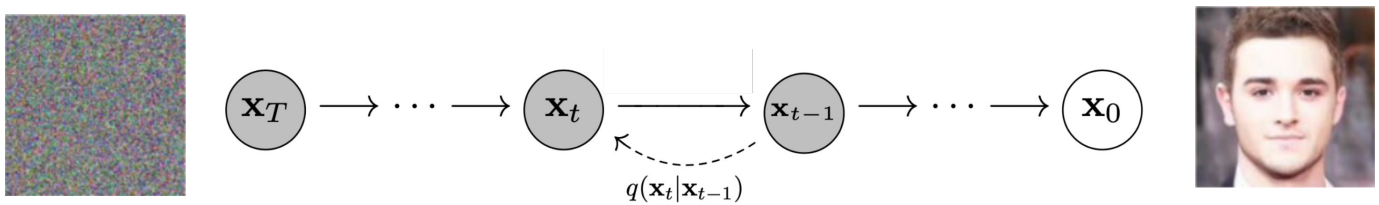


Improving GANs: Recap

- Use convolutions
- Use BN
- Many optimization tricks
- Change the loss (e.g. Wasserstein loss)
- Improve the architecture using inference normalization

Diffusion Models

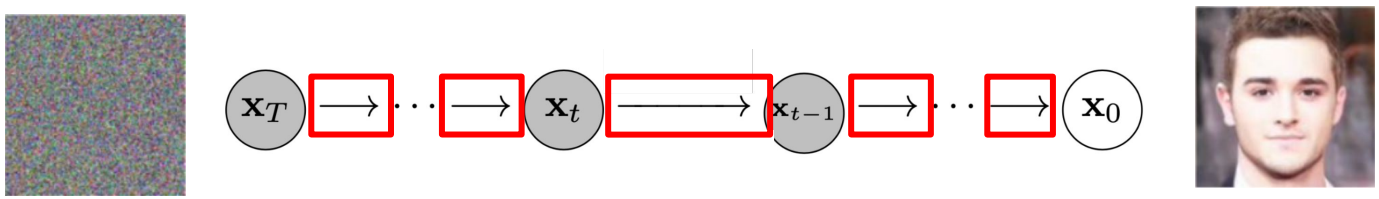
Formulate generation as a denoising problem:



We add Gaussian Noise: $q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$

Images inspired from: Denoising Diffusion Probabilistic Models, Ho et al, Neurips 2020

Diffusion Models



The goal: learn a neural network to approximate the probability distributions in the reverse diffusion process:

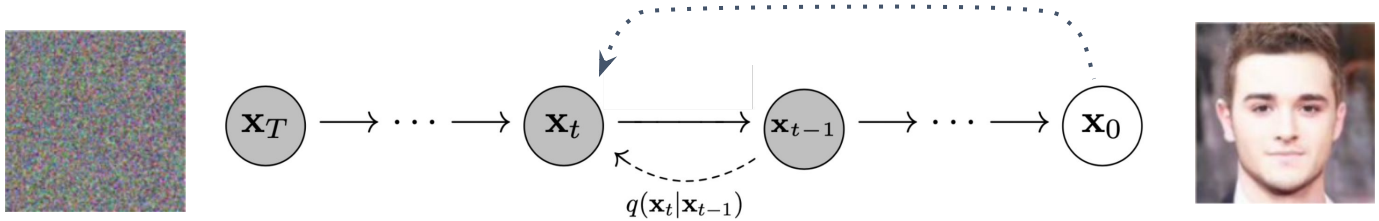
$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$$

Images inspired from: Denoising Diffusion Probabilistic Models, Ho et al, Neurips 2020

Diffusion Models

Formulate generation as a denoising problem:

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I})$$ We can combine the multiple steps.

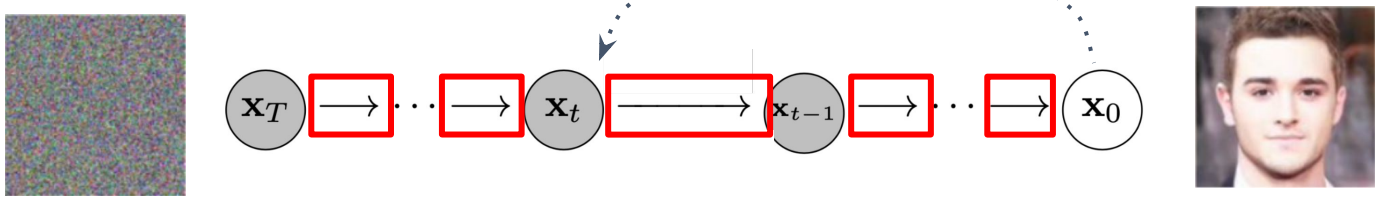


We add Gaussian Noise: $q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$

Images inspired from: Denoising Diffusion Probabilistic Models, Ho et al, Neurips 2020

Diffusion Models

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



Re-parametrization: We predict the noise instead of the image

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$$

$$\boldsymbol{\mu}_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right)$$

Noise estimate

Images inspired from: Denoising Diffusion Probabilistic Models, Ho et al, Neurips 2020

Diffusion Models

Training loss ?

- We can use the variational lower bound of the negative log-likelihood:

$$L = \mathbb{E}_q[\underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_T|\mathbf{x}_0) \parallel p_\theta(\mathbf{x}_T))}_{L_T} + \sum_{t=2}^{T} \underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \parallel p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}} \underbrace{- \log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0}]$$

Images inspired from: Denoising Diffusion Probabilistic Models, Ho et al, Neurips 2020

Diffusion Models

Training loss ?

- We can use the variational lower bound to optimize the negative

$$L = \mathbb{E}_q[\underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_T|\mathbf{x}_0) \parallel p_\theta(\mathbf{x}_T))}_{L_T} + \sum_{t=2}^{T} \underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \parallel p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}} \underbrace{- \log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0}]$$

where:

$$L_t = \mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}_t}\left[\frac{(1-\alpha_t)^2}{2\alpha_t(1-\bar{\alpha}_t)\|\boldsymbol{\Sigma}_\theta\|_2^2}\|\boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t, t)\|^2\right]$$

Images inspired from: Denoising Diffusion Probabilistic Models, Ho et al, Neurips 2020

Diffusion Models

Training loss ?

- We can simplify the loss.

$$L_t = \mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}_t}\left[\frac{(1-\alpha_t)^2}{2\alpha_t(1-\bar{\alpha}_t)\|\boldsymbol{\Sigma}_\theta\|_2^2}\|\boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t, t)\|^2\right]$$

It becomes:

$$L_t^{\text{simple}} = \mathbb{E}_{t\sim[1,T],\mathbf{x}_0,\boldsymbol{\epsilon}_t}\left[\|\boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\|^2\right]$$

Images inspired from: Denoising Diffusion Probabilistic Models, Ho et al, Neurips 2020

Diffusion Models

Training and sampling algorithms ?

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

Sampling

Diffusion Models

Training and sampling algorithms ?

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

Noise estimation error minimization

Images inspired from: Denoising Diffusion Probabilistic Models, Ho et al, Neurips 2020

Diffusion Models

Training and sampling algorithms ?

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\quad \mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Images inspired from: Denoising Diffusion Probabilistic Models, Ho et al, Neurips 2020

Diffusion Models

Training and sampling algorithms ?

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

We sample according to:

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$$

Images inspired from: Denoising Diffusion Probabilistic Models, Ho et al, Neurips 2020

Diffusion Models

Training and sampling algorithms ?

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

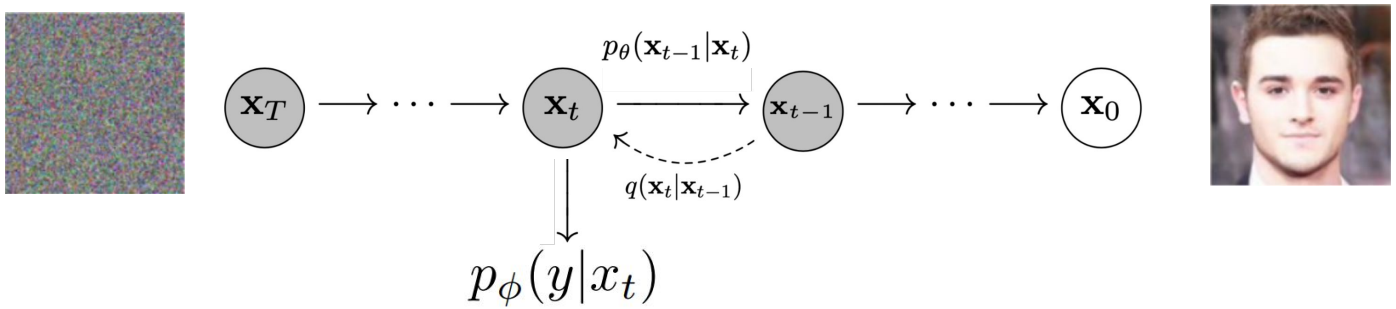
We sample the initial noise.

Images inspired from: Denoising Diffusion Probabilistic Models, Ho et al, Neurips 2020

Classifier guidance

How to condition the output image on y label ?

Let us assume a classif $p_\phi(y|x_t)$

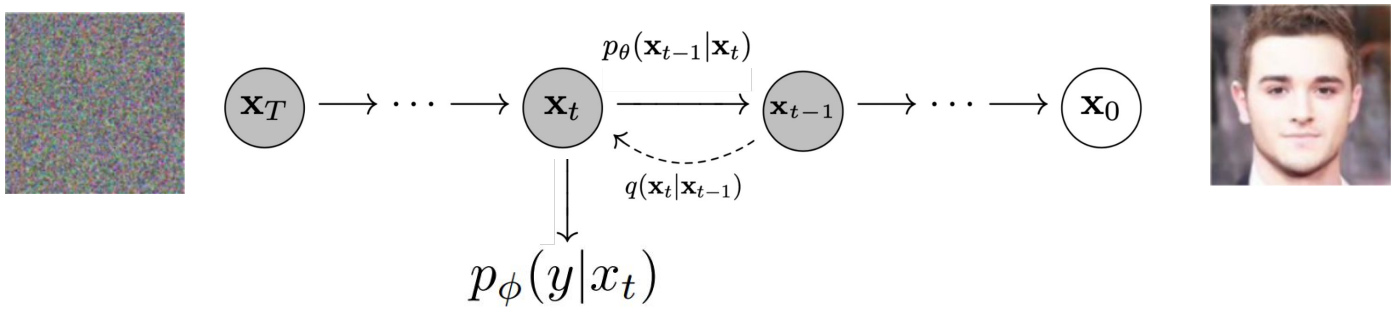


Images inspired from: Denoising Diffusion Probabilistic Models, Ho et al, Neurips 2020
Diffusion models beat GANs on image synthesis, Dhariwal and Nichol, arxiv, 2021

Classifier guidance

How to condition the output image on y label ?

Let us assume a classif $p_\phi(y|x_t)$



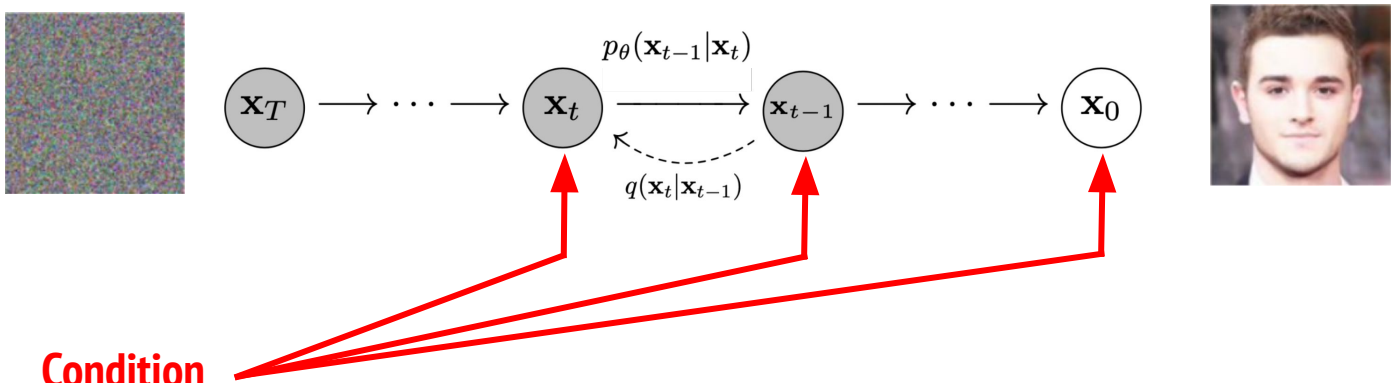
Modified update:

$$\hat{\epsilon}(x_t) := \epsilon_\theta(x_t) - \sqrt{1-\bar{\alpha}_t}\,\nabla_{x_t} \log p_\phi(y|x_t)$$

Images inspired from: Denoising Diffusion Probabilistic Models, Ho et al, Neurips 2020
Diffusion models beat GANs on image synthesis, Dhariwal and Nichol, arxiv, 2021

Classifier-free guidance

How to condition the output image on $y$label ?

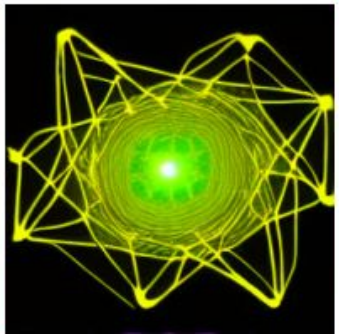


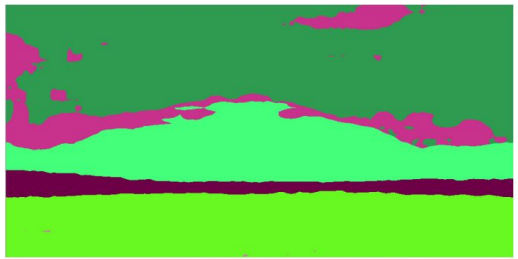
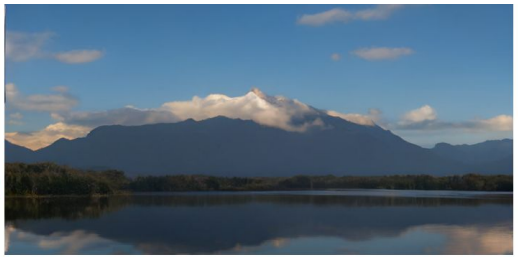
$$\tilde{\epsilon}_\theta(\mathbf{x_t}, y) = (1+w)\epsilon_\theta(\mathbf{x_t}, y) - w\epsilon_\theta(\mathbf{x_t})$$

Images inspired from: Denoising Diffusion Probabilistic Models, Ho et al, Neurips 2020
Diffusion models beat GANs on image synthesis, Dhariwal and Nichol, arxiv, 2021

Latent Diffusion Models

'An illustration of a slightly conscious neural network'	*'A painting of a squirrel eating a burger'*	*'A watercolor painting of a chair that looks like an octopus'*	*'A shirt with the inscription: "I love generative models!" '*

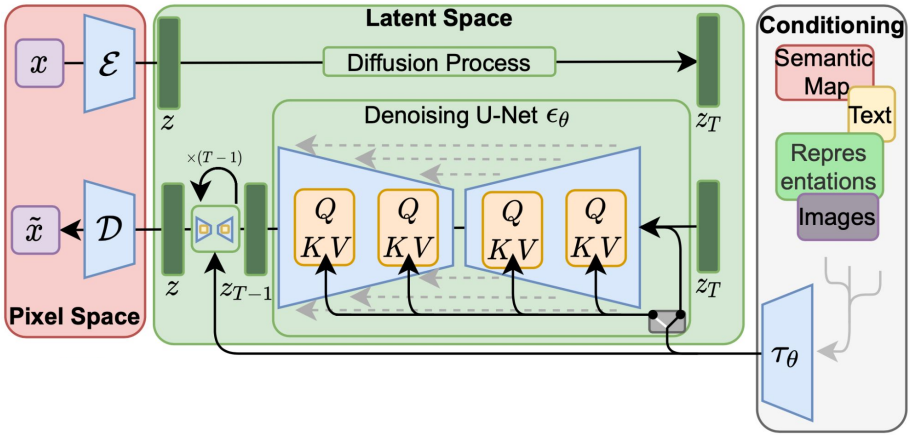



Images inspired from: High-Resolution Image Synthesis With Latent Diffusion Models, Rombach et al, CVPR 2022

Latent Diffusion Models

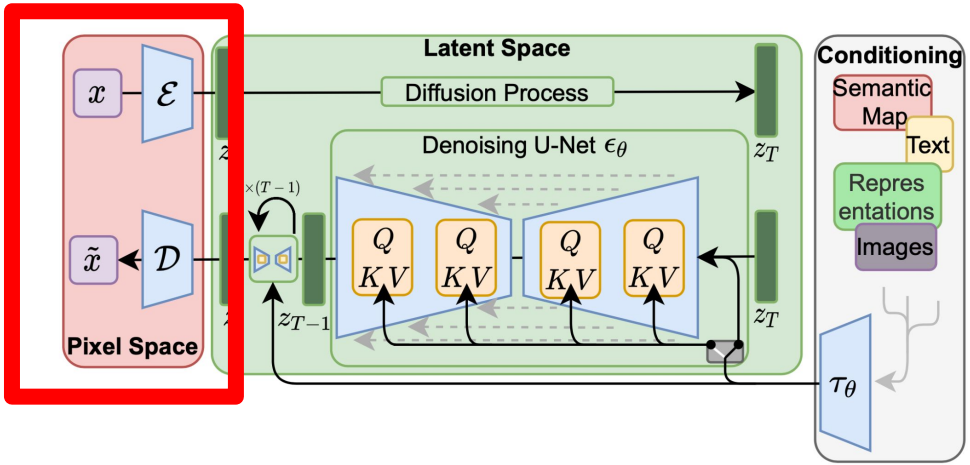
How to reduce computational complexity?



Images inspired from: High-Resolution Image Synthesis With Latent Diffusion Models, Rombach et al, CVPR 2022

Latent Diffusion Models

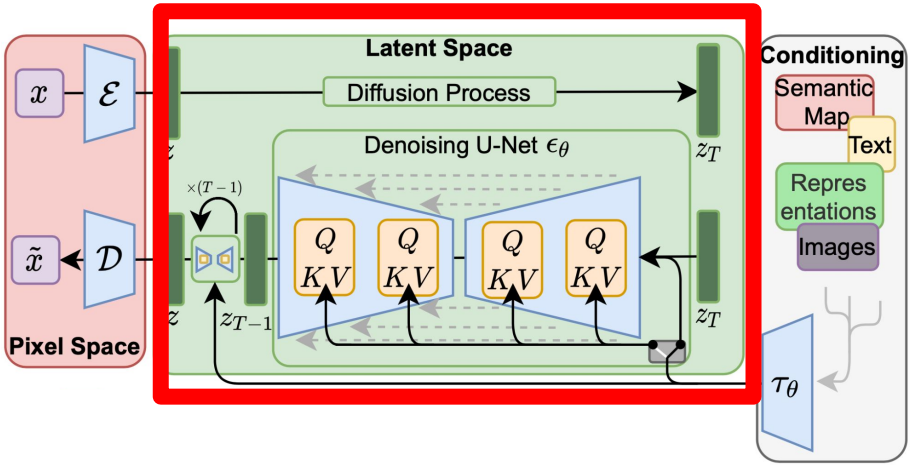
We project in a discrete latent space



Images inspired from: High-Resolution Image Synthesis With Latent Diffusion Models, Rombach et al, CVPR 2022

Latent Diffusion Models

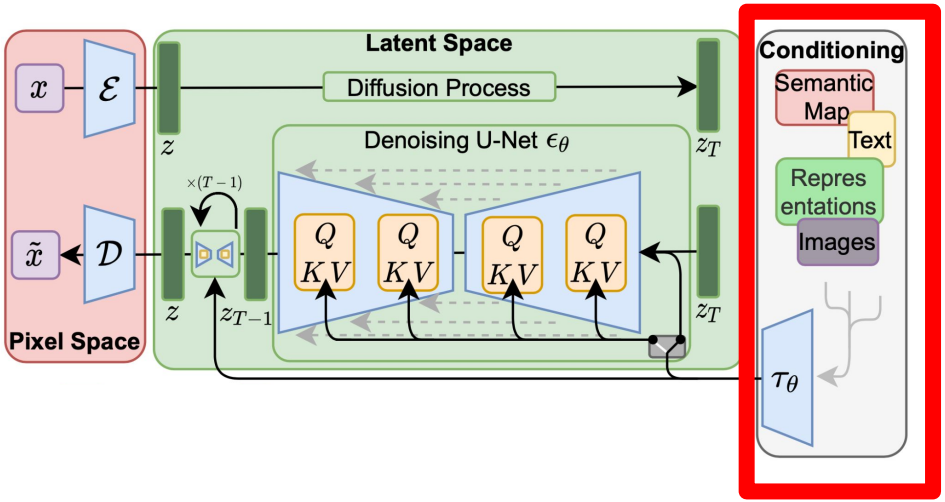
The diffusion process is used in the latent space



Images inspired from: High-Resolution Image Synthesis With Latent Diffusion Models, Rombach et al, CVPR 2022

Latent Diffusion Models

A conditional encoder can be learned



Images inspired from: High-Resolution Image Synthesis With Latent Diffusion Models, Rombach et al, CVPR 2022

Diffusion Models

If you want to know more

- Read the papers in the references.
- Lilian Weng's blog: https://lilianweng.github.io/posts/2021-07-11-diffusion-models/
- CVPR Tutorial on Denoising Diffusion-based Generative Modeling: Foundations and Applications: https://www.youtube.com/watch?v=cS6JQpEY9cs

Why does it change everything?

- Unprecedented image quality and data coverage.
- Fine-grained control with text.
- We have a general-purpose image prior.
- Since the training cost of large models is very high, transfer learning becomes the way to go for many problems.
- Because of the Iterative generation process, many algorithms need to be adapted to be compatible with diffusion models.