中山大學
SUN YAT-SEN UNIVERSITY

Lecture 25. Generative Models

Pattern Recognition and Computer Vision

Guanbin Li,
School of Computer Science and Engineering, Sun Yat-Sen University

Supervised vs Unsupervised Learning

- **Supervised Learning**

- **Data**: (x, y)
- x is data, y is label.

- **Goal**: Learn a *function* to map x -> y.

- **Examples**: Classification, regression, object detection, semantic segmentation, image captioning, etc.

Supervised vs Unsupervised Learning

- **Supervised Learning**

- **Data**: (x, y)
- x is data, y is label.

- **Goal**: Learn a *function* to map x -> y.

- **Examples**: Classification, regression, object detection, semantic segmentation, image captioning, etc.

Cat

Classification

Supervised vs Unsupervised Learning

- **Supervised Learning**

- **Data**: (x, y)
- x is data, y is label.

- **Goal**: Learn a *function* to map x -> y.

- **Examples**: Classification, regression, object detection, semantic segmentation, image captioning, etc.

A cat sitting on a suitcase on the floor

Image captioning

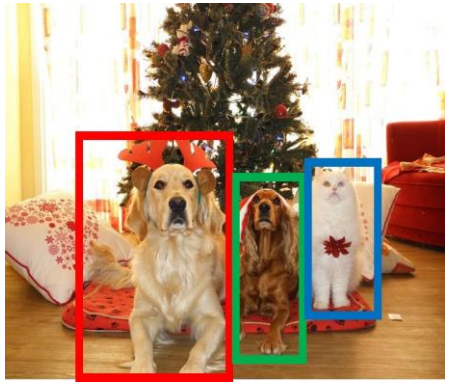
Supervised vs Unsupervised Learning

- **Supervised Learning**

- **Data**: (x, y)
- x is data, y is label.

- **Goal**: Learn a *function* to map x -> y.

- **Examples**: Classification, regression, object detection, semantic segmentation, image captioning, etc.



DOG, **DOG**, **CAT**

Object Detection

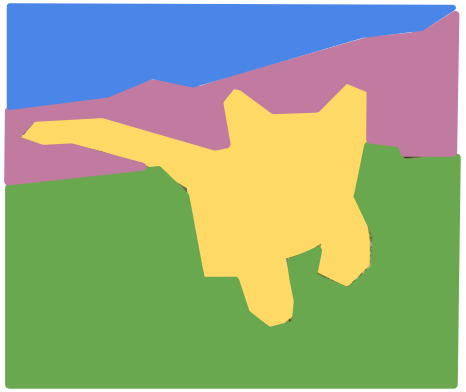
Supervised vs Unsupervised Learning

- **Supervised Learning**

- **Data**: (x, y)
- x is data, y is label.

- **Goal**: Learn a *function* to map x -> y.

- **Examples**: Classification, regression, object detection, semantic segmentation, image captioning, etc.



GRASS, **CAT**, **TREE**, **SKY**

Semantic Segmentation

Supervised vs Unsupervised Learning

- **Unsupervised Learning**

- **Data**: x
- Just data, **no labels!**

- **Goal**: Learn some underlying hidden *structure* of the data.

- **Examples**: Clustering, dimensionality reduction, feature learning, density estimation, etc.

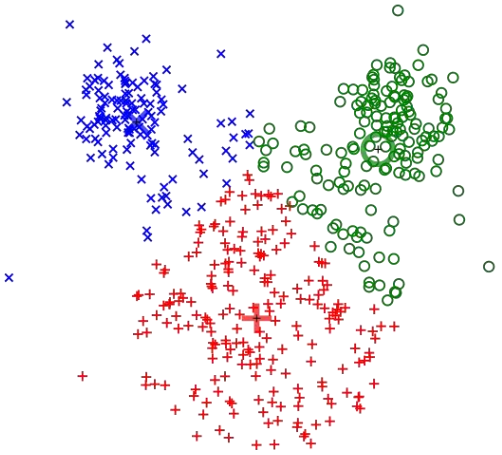
Supervised vs Unsupervised Learning

- **Unsupervised Learning**

- **Data**: x
- Just data, **no labels!**

- **Goal**: Learn some underlying hidden *structure* of the data.

- **Examples**: Clustering, dimensionality reduction, feature learning, density estimation, etc.



K-means clustering

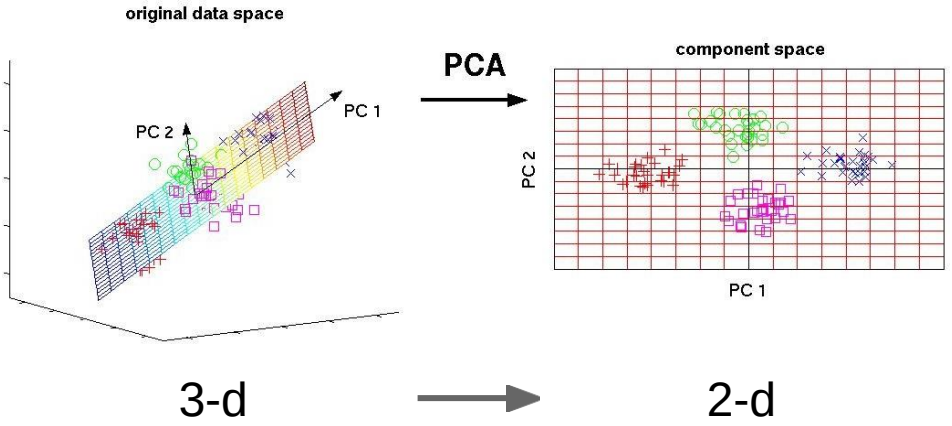
Supervised vs Unsupervised Learning

- **Unsupervised Learning**

- **Data**: x
- Just data, **no labels!**

- **Goal**: Learn some underlying hidden *structure* of the data.

- **Examples**: Clustering, dimensionality reduction, feature learning, density estimation, etc.



Principal Component Analysis
(Dimensionality reduction)

Supervised vs Unsupervised Learning

- **Unsupervised Learning**

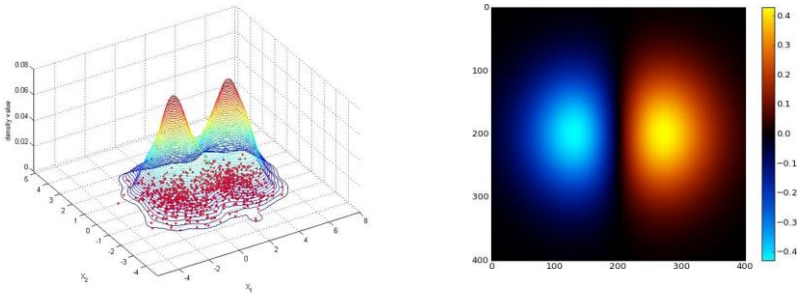
- **Data**: x
- Just data, **no labels!**

- **Goal**: Learn some underlying hidden *structure* of the data.

- **Examples**: Clustering, dimensionality reduction, feature learning, density estimation, etc.

Figure copyright Ian Goodfellow, 2016. Reproduced with permission.

1-d density estimation



2-d density estimation

Modeling p(x)

Supervised vs Unsupervised Learning

- **Supervised Learning**

- **Data**: (x, y)
- x is data, y is label.

- **Goal**: Learn a *function* to map x -> y.

- **Examples**: Classification, regression, object detection, semantic segmentation, image captioning, etc.

- **Unsupervised Learning**

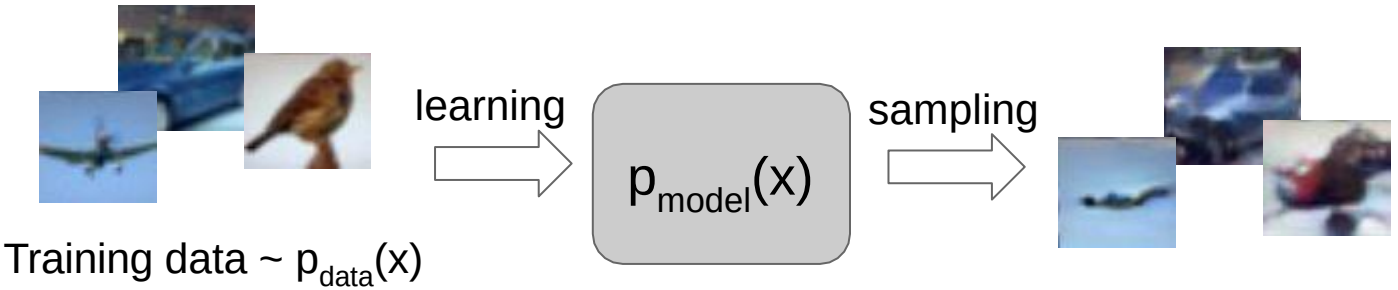
- **Data**: x
- Just data, no labels!

- **Goal**: Learn some underlying hidden *structure* of the data.

- **Examples**: Clustering, dimensionality reduction, feature learning, density estimation, etc.

Generative Modeling

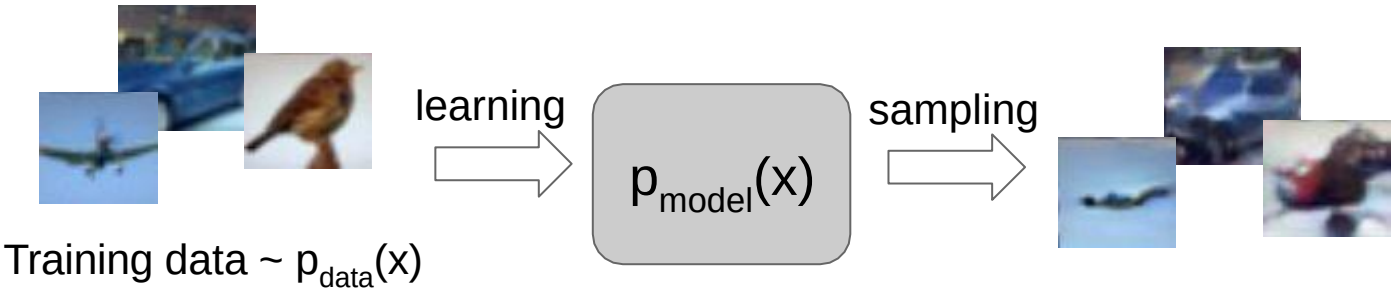
- Given training data, generate new samples from same distribution.



- Objectives:
  - Learn $p_{model}(x)$ that approximates $p_{data}(x)$.
  - **Sample new x from $p_{model}(x)$.**

Generative Modeling

- Given training data, generate new samples from same distribution.



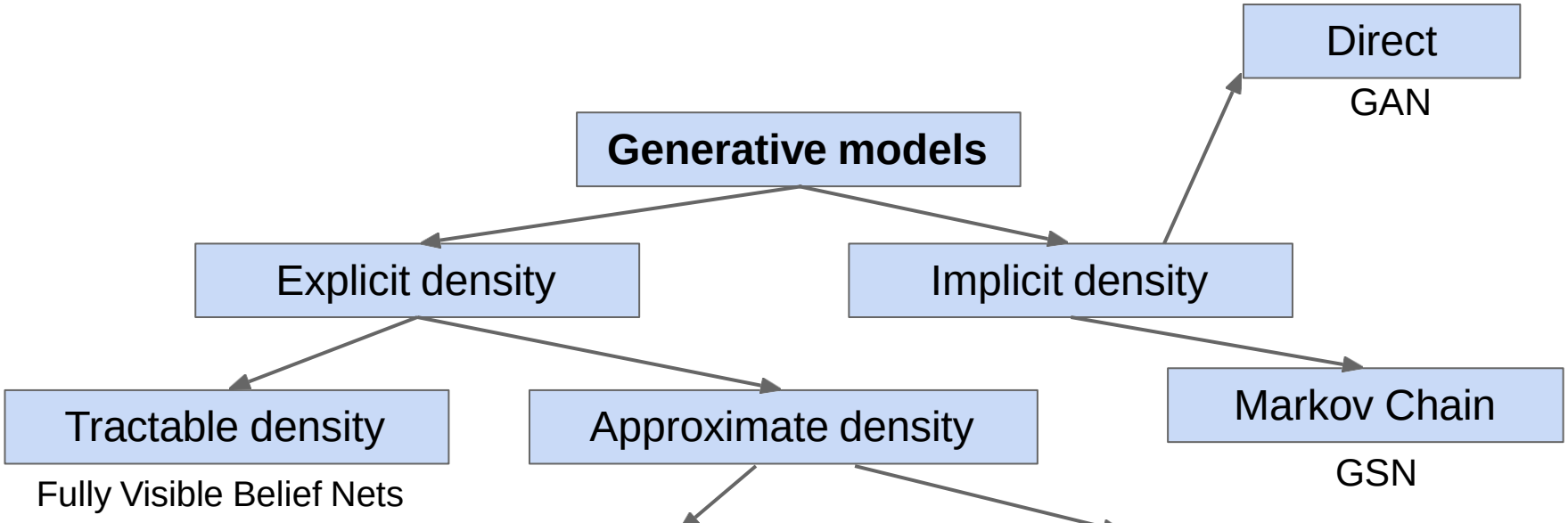
- Formulate as density estimation problems**:**
  - **Explicit density estimation**: explicitly define and solve for $p_{model}(x)$
  - **Implicit density estimation**: learn model that can sample from $p_{model}(x)$ **without explicitly defining it.**

Why Generative Models?



- Realistic samples for artwork, super-resolution, colorization, etc.
- Learn useful features for downstream tasks such as classification.
- Getting insights from high-dimensional data (physics, medical imaging, etc.)
- Modeling physical world for simulation and planning (robotics and reinforcement learning applications)
- Many more ...

Taxonomy of Generative Models

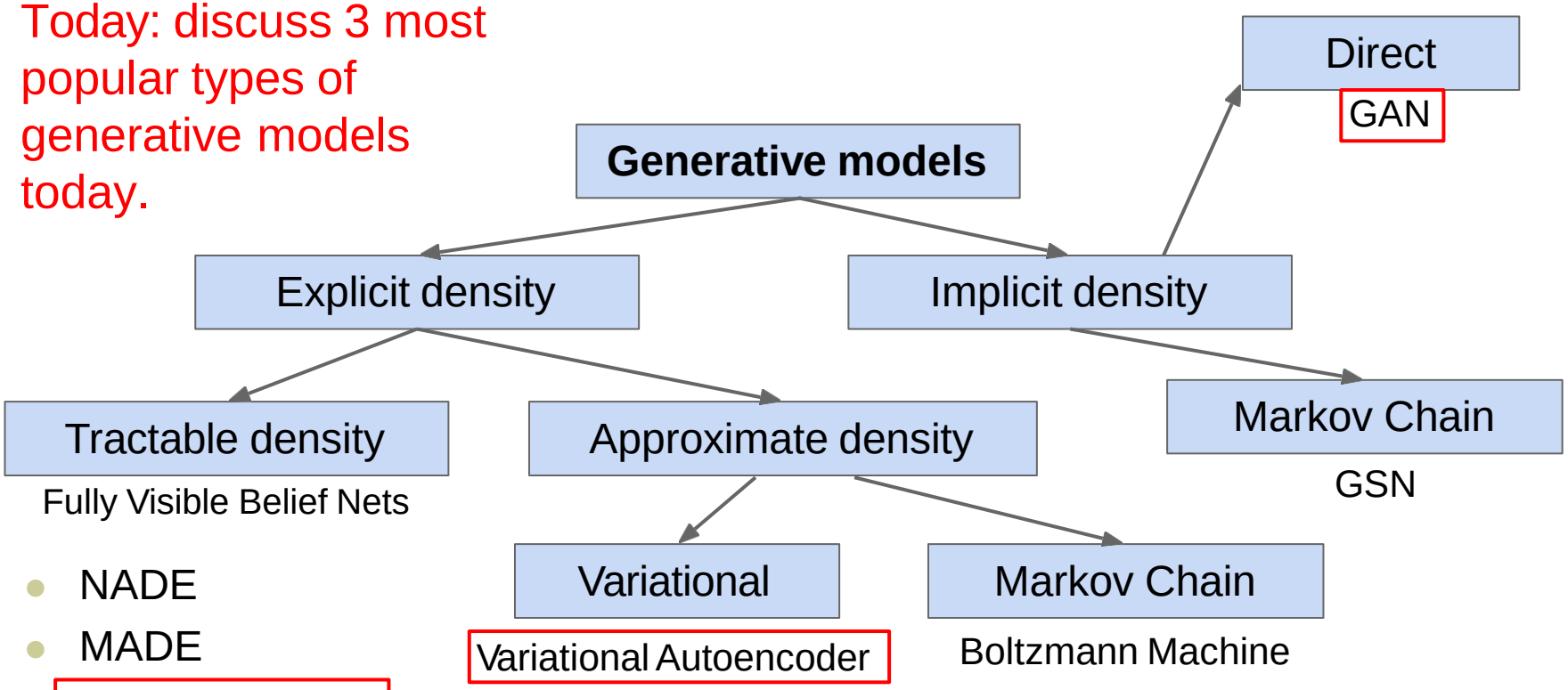


- NADE
- MADE
- PixelRNN/CNN
- NICE / RealNVP
- Glow
- Ffjord

Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Taxonomy of Generative Models

Today: discuss 3 most popular types of generative models today.



- NADE
- MADE
- PixelRNN/CNN
- NICE / RealNVP
- Glow
- Ffjord

Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

PixelRNN and PixelCNN (A very brief overview)

Fully visible belief network (FVBN)

Explicit density model

$$p(x) = p(x_1, x_2, \ldots, x_n)$$

Likelihood of image x

Joint likelihood of each pixel in the image

Fully visible belief network (FVBN)

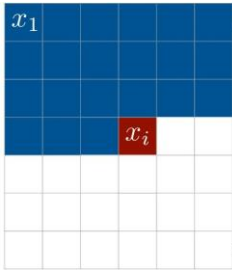
Explicit density model

- Use chain rule to decompose likelihood of an image x into product of 1-d distributions:

$$p(x) = \prod_{i=1}^{n} p(x_i | x_1, \ldots, x_{i-1})$$

Likelihood of image x

Probability of i'th pixel value given all previous pixels



- Then maximize likelihood of training data.

Fully visible belief network (FVBN)

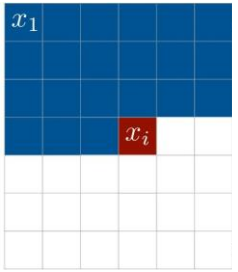
Explicit density model

- Use chain rule to decompose likelihood of an image x into product of 1-d distributions:

$$p(x) = \prod_{i=1}^{n} p(x_i | x_1, \ldots, x_{i-1})$$

Likelihood of image x

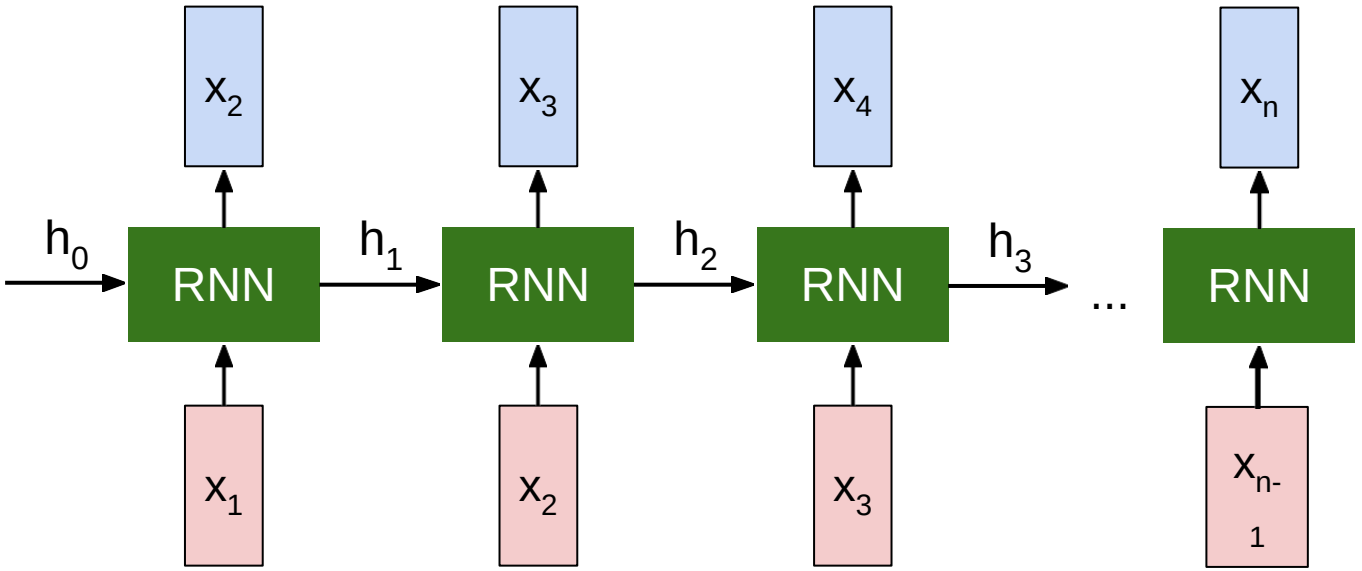
Probability of i'th pixel value given all previous pixels



Complex distribution over pixel values => Express using a neural network!

- Then maximize likelihood of training data.

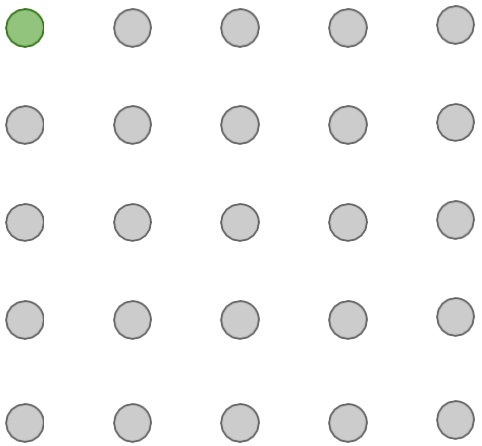
Recurrent Neural Network



$$p(x_i|x_1, \ldots, x_{i-1})$$

PixelRNN *[van der Oord et al. 2016]*

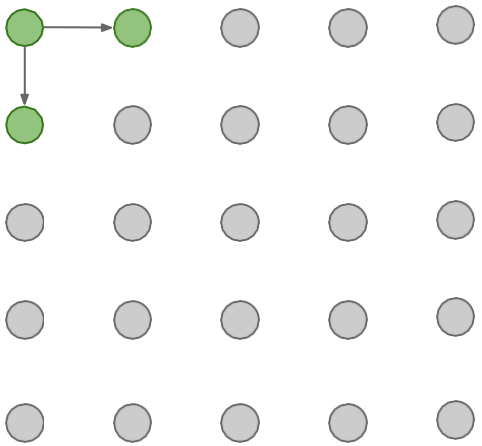
- Generate image pixels starting from corner.
- Dependency on previous pixels modeled using an RNN (LSTM).



PixelRNN *[van der Oord et al. 2016]*

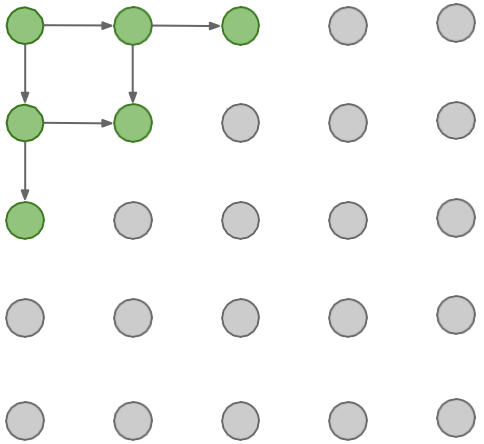
- Generate image pixels starting from corner.

- Dependency on previous pixels modeled using an RNN (LSTM).



PixelRNN *[van der Oord et al. 2016]*

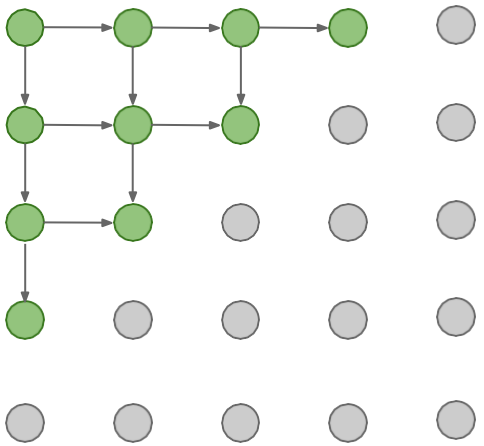
- Generate image pixels starting from corner.
- Dependency on previous pixels modeled using an RNN (LSTM).



PixelRNN *[van der Oord et al. 2016]*

- Generate image pixels starting from corner.

- Dependency on previous pixels modeled using an RNN (LSTM).

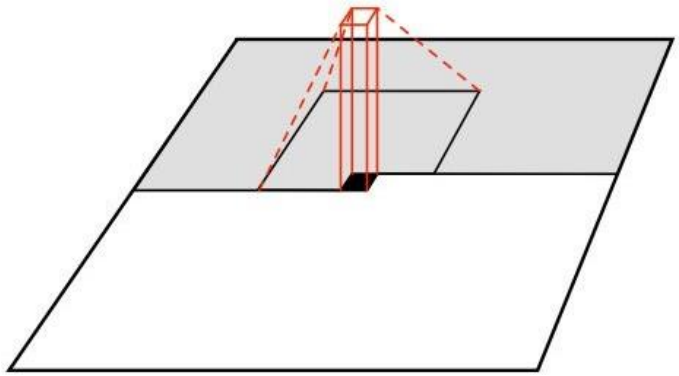


- Drawback: sequential generation is slow in both training and inference!

PixelCNN *[van der Oord et al. 2016]*

- Still generate image pixels starting from corner.

- Dependency on previous pixels now modeled using a CNN over context region **(masked convolution)**.



PixelCNN *[van der Oord et al. 2016]*

- Still generate image pixels starting from corner.

- Dependency on previous pixels now modeled using a CNN over context region (masked convolution).

Training is faster than PixelRNN

(can parallelize convolutions since context region values known from training images).

Generation is still slow:

For a 32x32 image, we need to do forward passes of the network 1024 times for a single image.

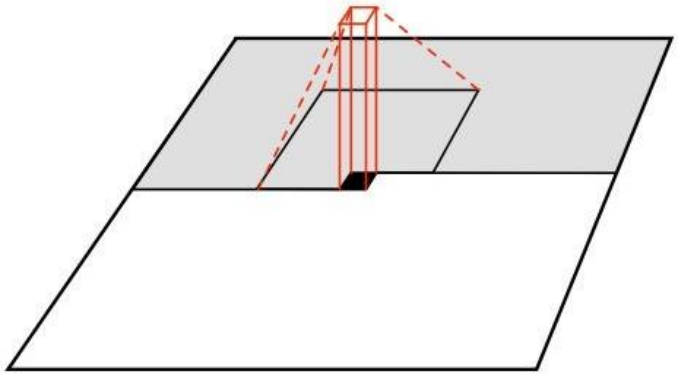
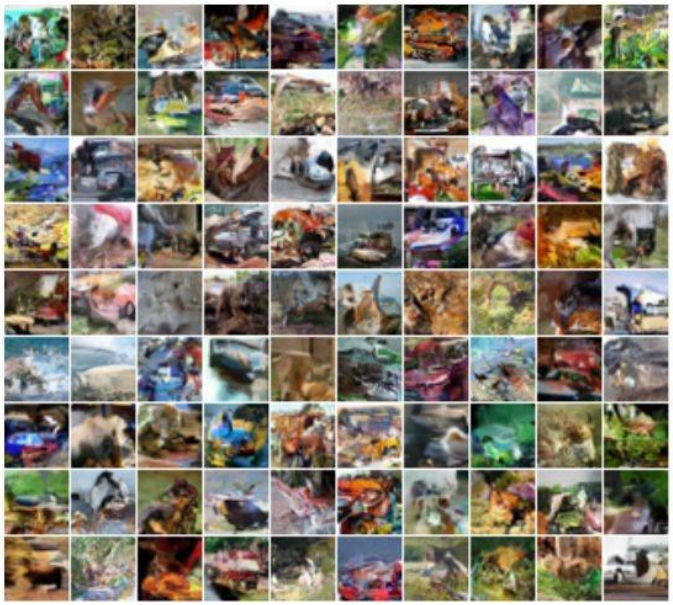
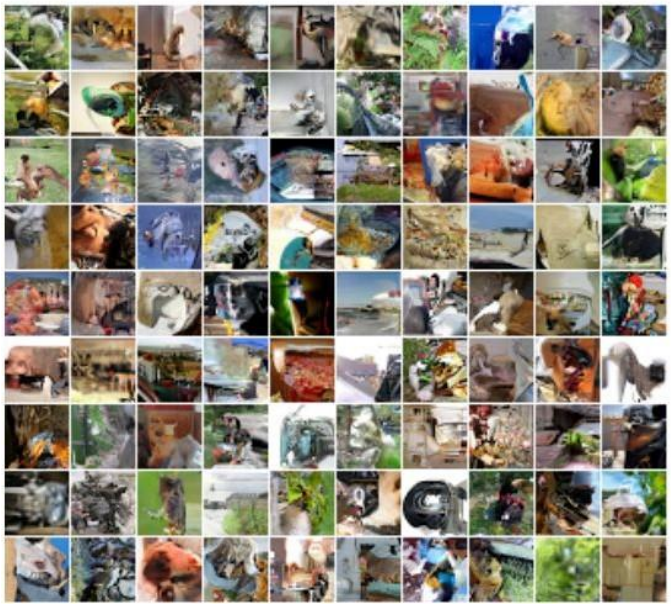


Figure copyright van der Oord et al., 2016. Reproduced with permission.

Generation Samples



32x32 CIFAR-10



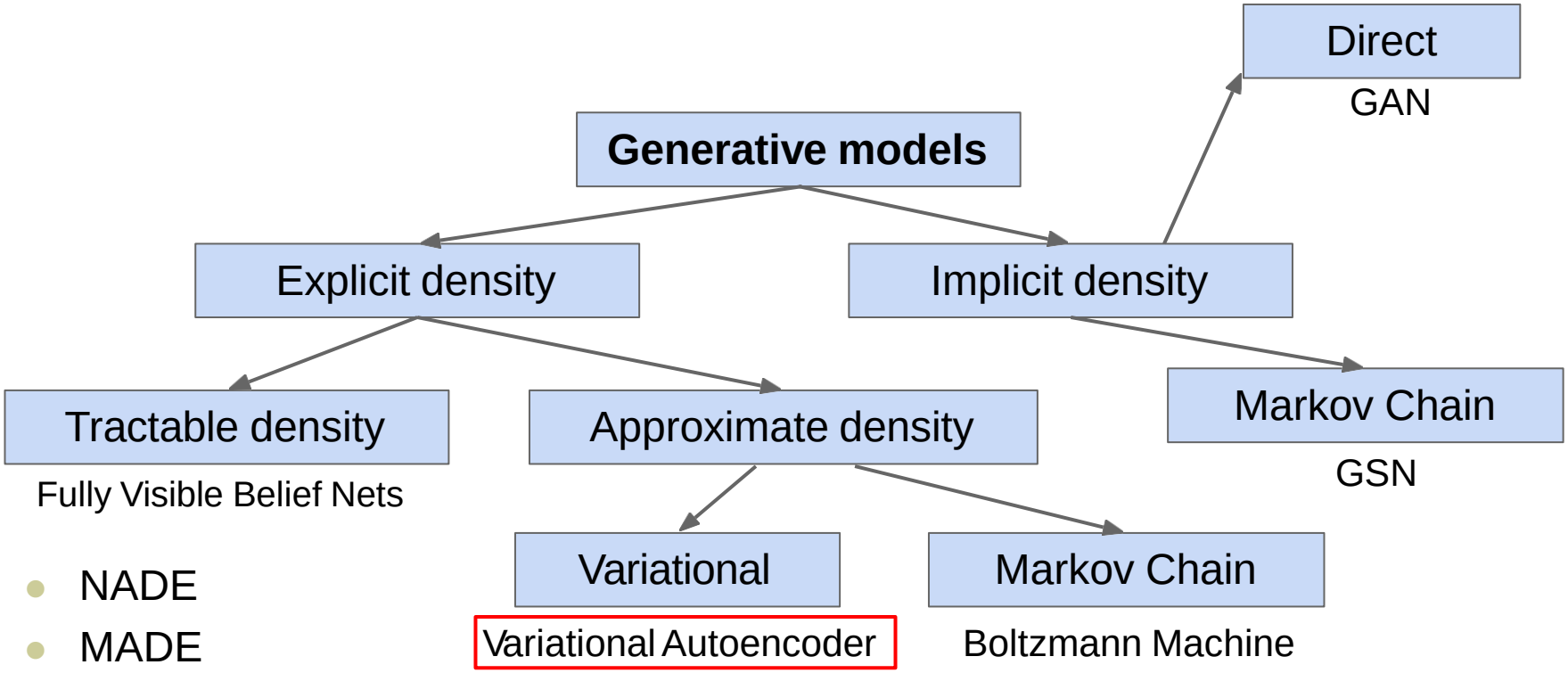
32x32 ImageNet

PixelRNN and PixelCNN

- Pros:
  - Can explicitly compute likelihood p(x)
  - Easy to optimize
  - Good samples
- Con:
  - Sequential generation => slow

- Improving PixelCNN performance
  - Gated convolutional layers
  - Short-cut connections
  - Discretized logistic loss
  - Multi-scale
  - Training tricks
  - Etc…
- See
  - Van der Oord et al. NIPS 2016
  - Salimans et al. 2017 (PixelCNN++)

Taxonomy of Generative Models



- NADE
- MADE
- PixelRNN/CNN
- NICE / RealNVP
- Glow
- Ffjord

Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Variational Autoencoders (VAE)

So far...

- PixelRNN/CNNs define tractable density function, optimize likelihood of training data:

$$p_\theta(x) = \prod_{i=1}^{n} p_\theta(x_i | x_1, ..., x_{i-1})$$

So far...

- PixelRNN/CNNs define tractable density function, optimize likelihood of training data:

$$p_\theta(x) = \prod_{i=1}^{n} p_\theta(x_i | x_1, ..., x_{i-1})$$

- Variational Autoencoders (VAEs) define intractable density function with latent **z**:

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

- No dependencies among pixels, can generate all pixels at the same time!

- Cannot optimize directly, derive and optimize lower bound on likelihood instead.

So far...

- PixelRNN/CNNs define tractable density function, optimize likelihood of training data:

$$p_\theta(x) = \prod_{i=1}^{n} p_\theta(x_i | x_1, ..., x_{i-1})$$

- Variational Autoencoders (VAEs) define intractable density function with latent **z**:

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

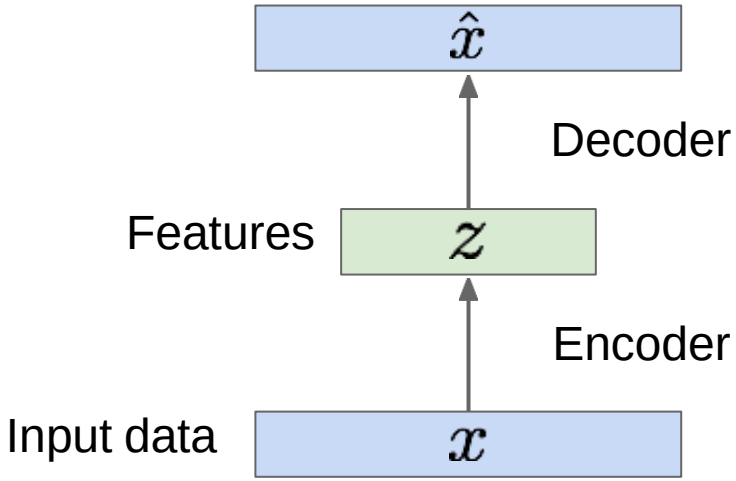
- No dependencies among pixels, can generate all pixels at the same time!

- Cannot optimize directly, derive and optimize lower bound on likelihood instead.

- Why latent z?

Some background first: Autoencoders

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

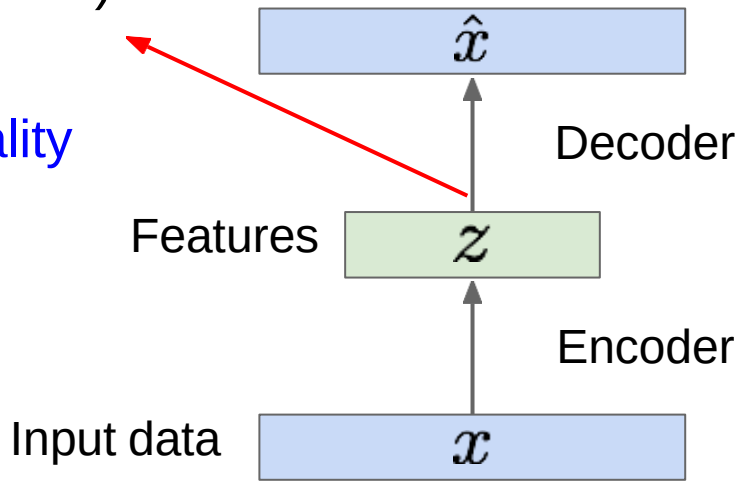


Some background first: Autoencoders

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

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

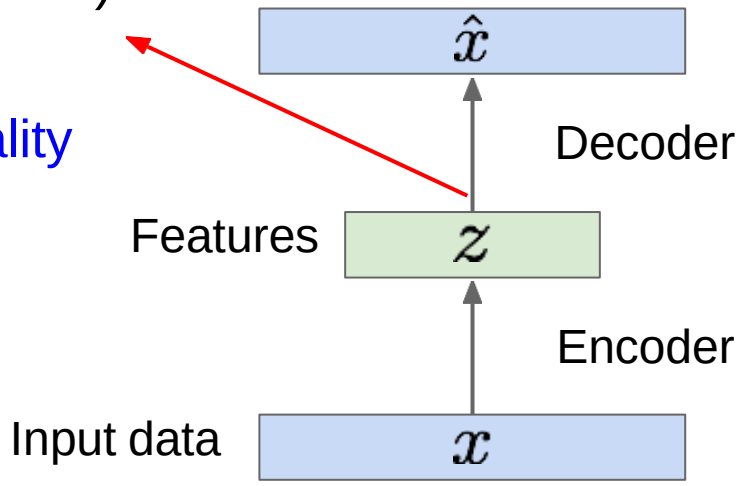
- Q: Why dimensionality reduction?



Some background first: Autoencoders

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

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



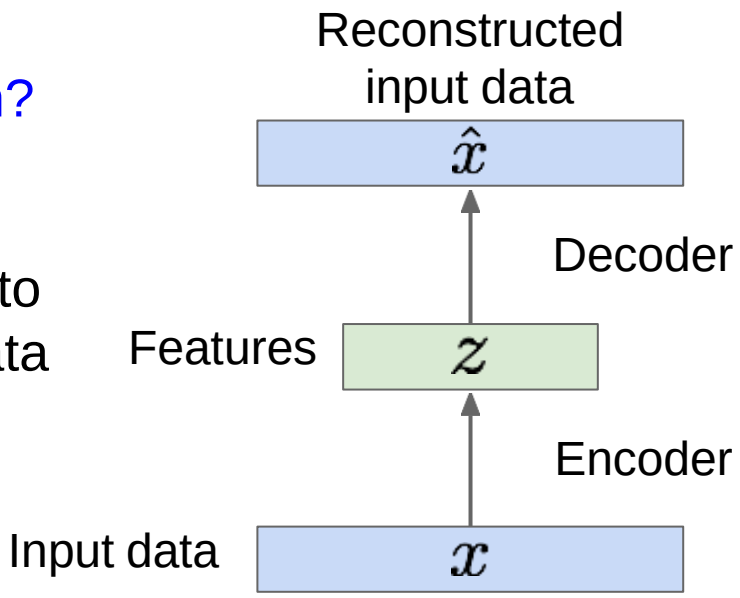
- Q: Why dimensionality reduction?
- A: Want features to capture meaningful factors of variation in data.

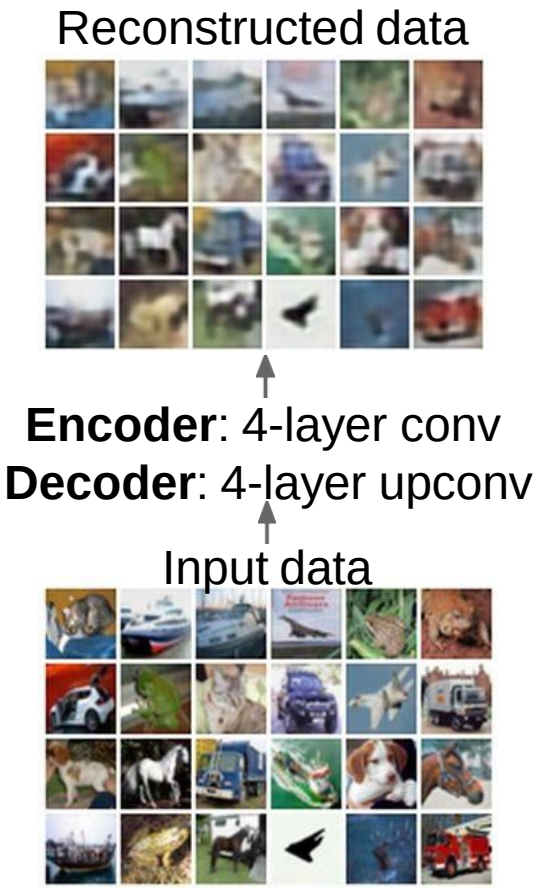
Some background first: Autoencoders

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

- How to learn this feature representation?

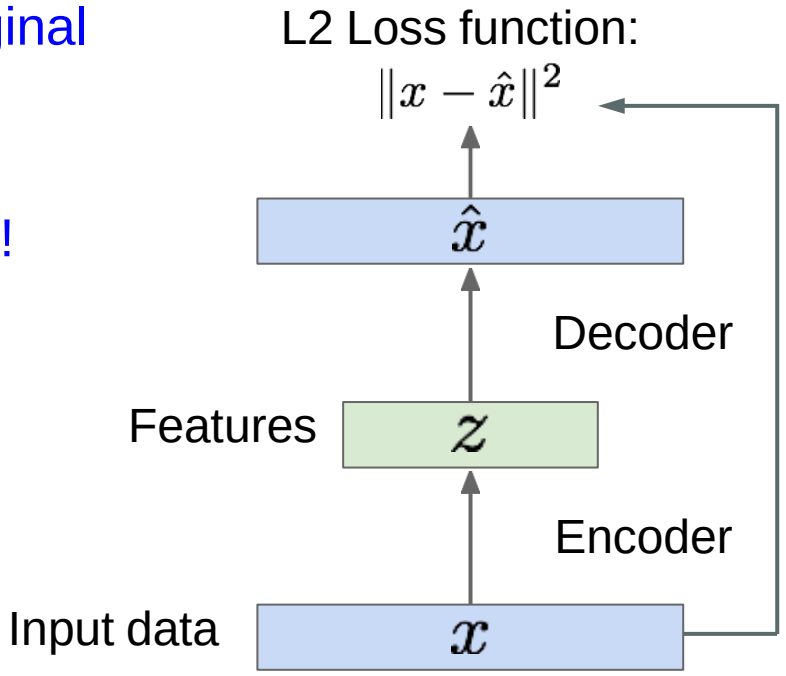
- Train such that features can be used to reconstruct original data
"Autoencoding" – encoding input itself.





Some background first: Autoencoders

- Train such that features can be used to reconstruct original data.
- Doesn't use labels!

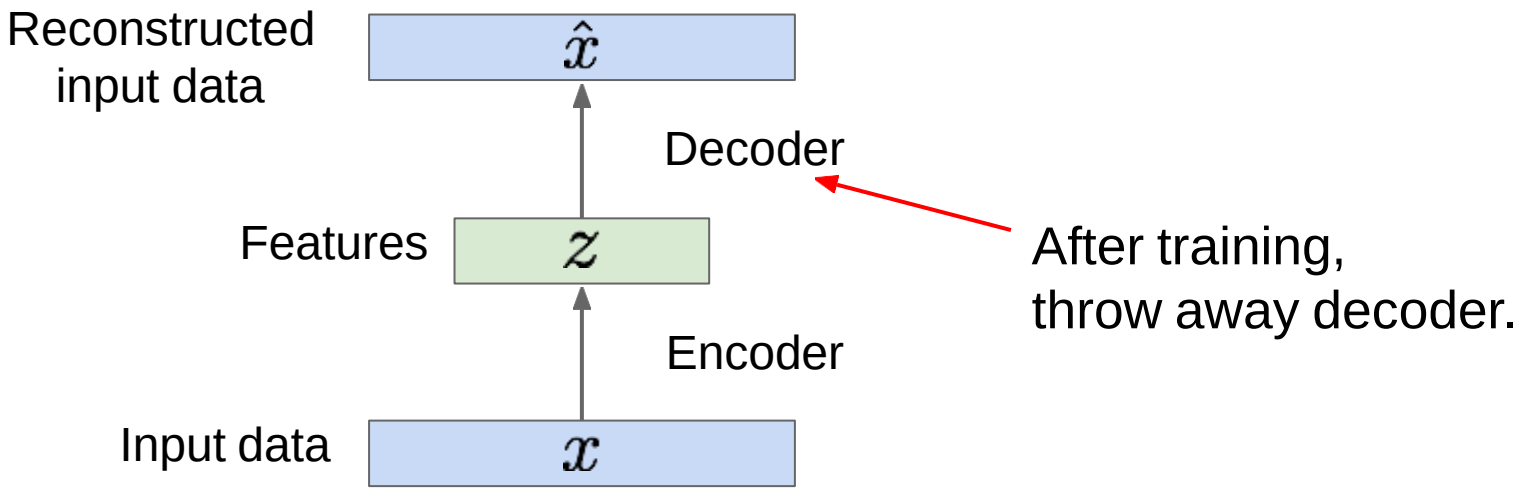


Reconstructed data

Encoder: 4-layer conv
Decoder: 4-layer upconv

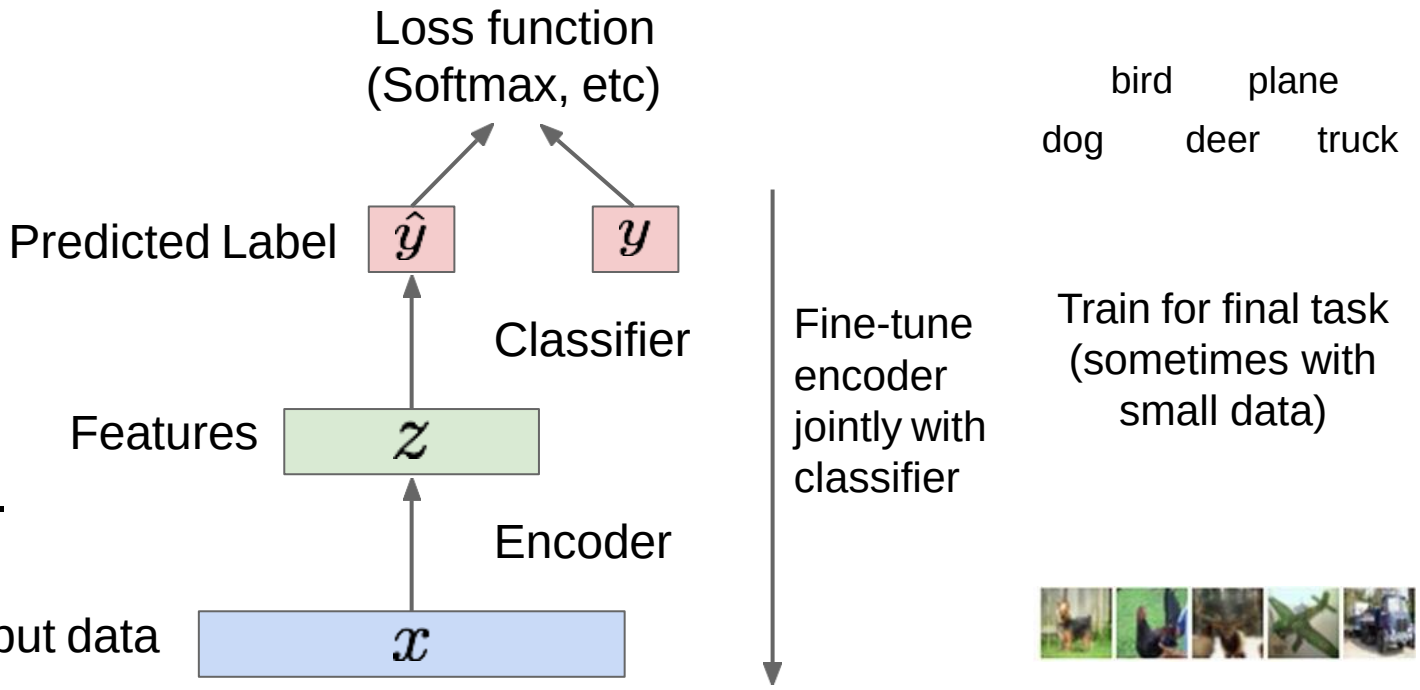
Input data

Some background first: Autoencoders



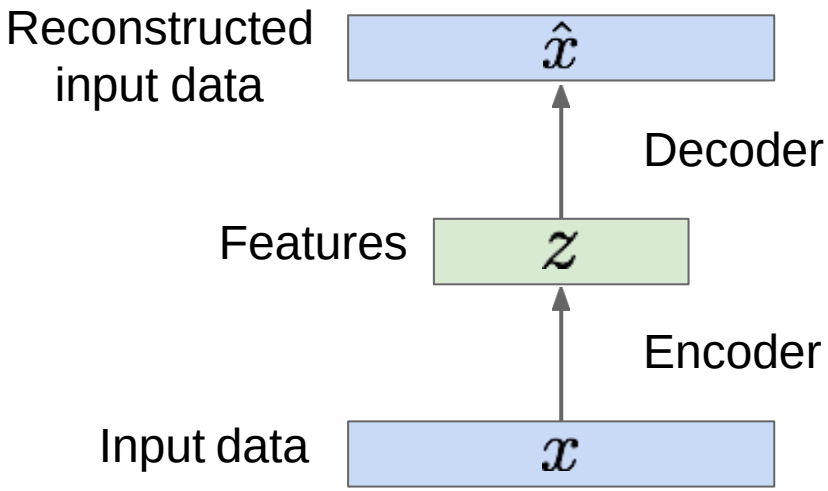
Some background first: Autoencoders

- Transfer from large, unlabeled dataset to small, labeled dataset.



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

Some background first: Autoencoders



- Autoencoders can reconstruct data, and can learn features to initialize a supervised model.
- Features capture factors of variation in training data.
- But we can't generate new images from an autoencoder because we don't know the space of z.
- How do we make autoencoder a **generative model**?

Variational Autoencoders

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

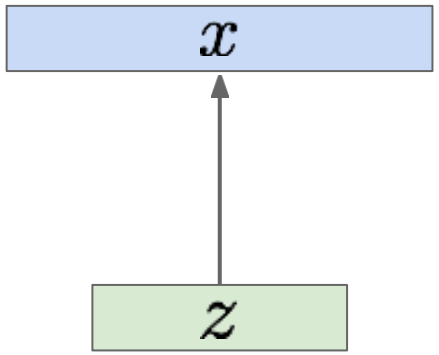
Variational Autoencoders

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

- Assume training data $\{x^{(i)}\}_{i=1}^{N}$ is generated from the distribution of unobserved (latent) representation **z**.

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

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



Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

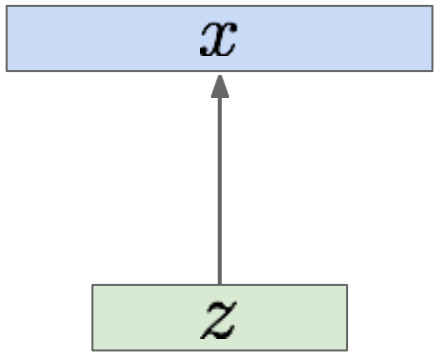
Variational Autoencoders

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

- Assume training data $\{x^{(i)}\}_{i=1}^{N}$ is generated from the distribution of unobserved (latent) representation **z**

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

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



- **Intuition** (remember from autoencoders!): **x** is an image, **z** is latent factors used to generate **x:** attributes, orientation, etc.

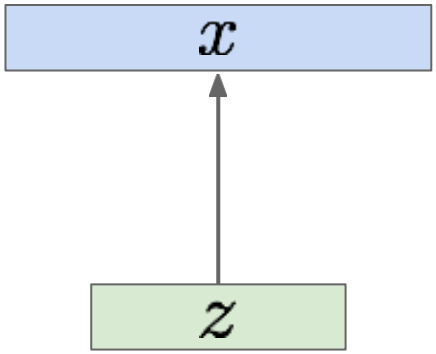
Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoders

- We want to estimate the true parameters θ^* of this generative model given training data x.

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

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



Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

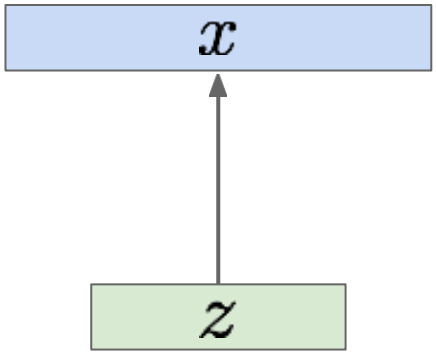
Variational Autoencoders

- We want to estimate the true parameters θ^* of this generative model given training data x.

- How should we represent this model?

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

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



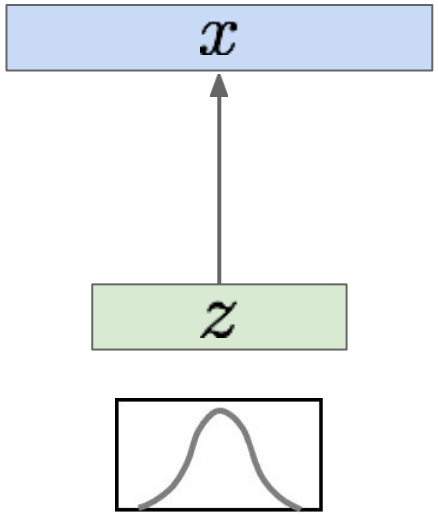
Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoders

- We want to estimate the true parameters θ^* of this generative model given training data x.

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

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



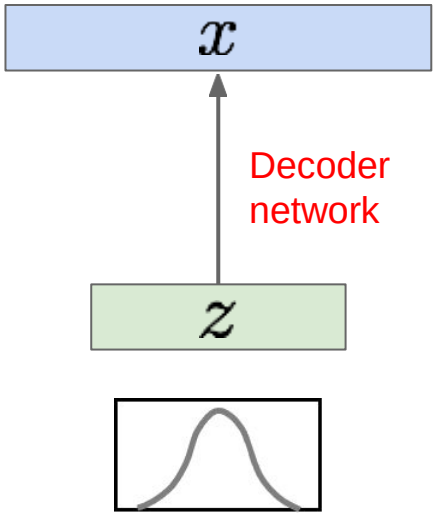
- How should we represent this model?

- Choose prior p(z) to be simple, e.g. Gaussian. Reasonable for latent attributes, e.g. pose, how much smile.

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoders

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



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

- We want to estimate the true parameters θ^* of this generative model given training data x.
- How should we represent this model?
- Choose prior p(z) to be simple, e.g. Gaussian. Reasonable for latent attributes, e.g. pose, how much smile.
- Conditional p(x|z) is complex (generates image) => represent with neural network.

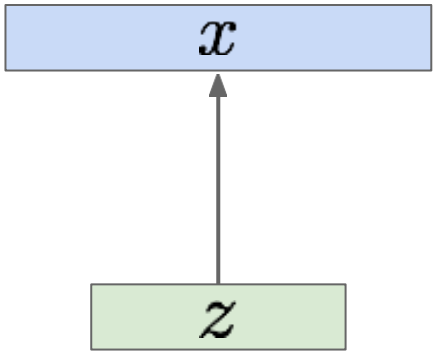
Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoders

- We want to estimate the true parameters θ^* of this generative model given training data x.
- How to train the model?

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

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

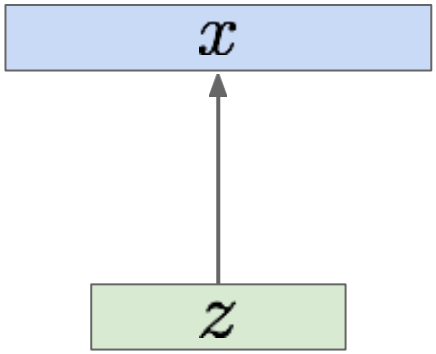


Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoders

- We want to estimate the true parameters θ^* of this generative model given training data x.

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



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

- How to train the model?
- Learn model parameters to maximize likelihood of training data

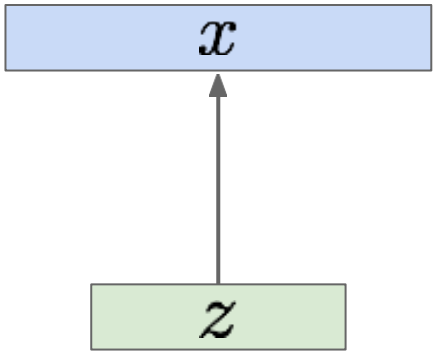
$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoders

- We want to estimate the true parameters θ^* of this generative model given training data x.

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



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

- How to train the model?
- Learn model parameters to maximize likelihood of training data

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

- Q: What is the problem with this? Intractable!

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoders: Intractability

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

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoders: Intractability

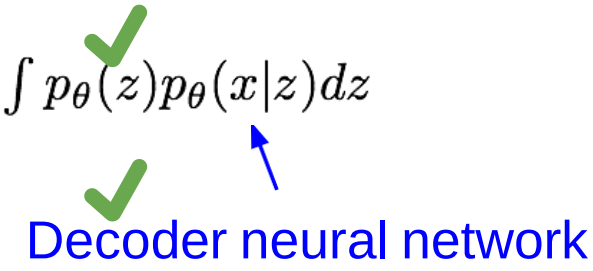
- Data likelihood: $p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$

Simple Gaussian prior

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoders: Intractability

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



Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoders: Intractability

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

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

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoders: Intractability

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

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

$$\log p(x) \approx \log \frac{1}{k} \sum_{i=1}^{k} p(x|z^{(i)}), \text{where } z^{(i)} \sim p(z)$$

Monte Carlo estimation is too high variance

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoders: Intractability

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

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

Intractable data likelihood

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoders: Intractability

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

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

- **Solution**: In addition to modeling $p_\theta(x|z)$, learn $q_\phi(z|x)$ that approximates the true posterior $p_\theta(z|x)$.

- Will see that the approximate posterior allows us to derive a lower bound on the data likelihood that is tractable, which we can optimize.

- **Variational inference** is to approximate the unknown posterior distribution from only the observed data x.

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoders

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

Variational Autoencoders

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

Taking expectation wrt. z (using encoder network) will come in handy later.

Variational Autoencoders

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

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

Variational Autoencoders

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

Variational Autoencoders

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

Variational Autoencoders

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

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

Variational Autoencoders

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

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

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

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

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

Decoder network gives $p_\theta(x|z)$, can compute estimate of this term through sampling (need some trick to differentiate through sampling).

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 Autoencoders

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

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

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

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

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

We want to maximize the data likelihood.

Decoder network gives $p_\theta(x|z)$, can compute estimate of this term through sampling (need some trick to differentiate through sampling).

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 Autoencoders

We want to maximize the data likelihood.

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

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

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

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

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

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

Variational Autoencoders

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

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

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

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

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

Decoder: reconstruct the input data

Encoder: make approximate posterior distribution close to prior

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

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.

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

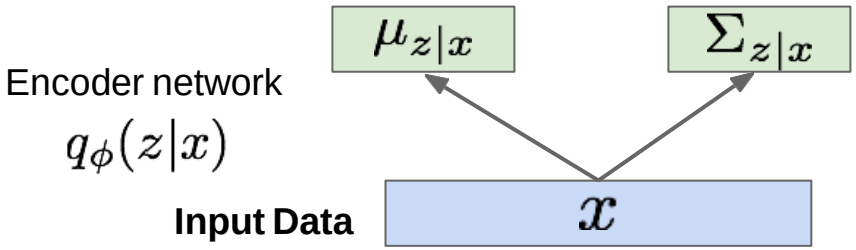
Let's look at computing the KL divergence between the estimated posterior and the prior given some data.

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

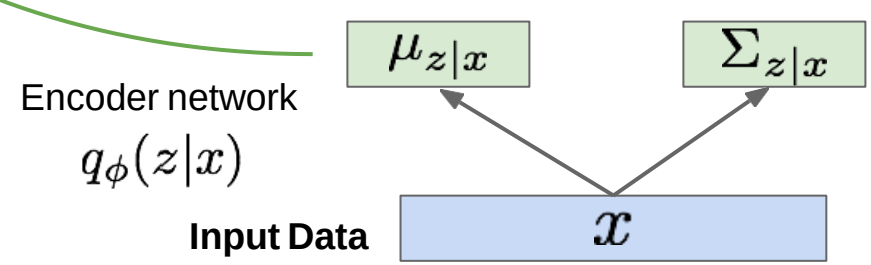
Putting it all together: maximizing the likelihood lower bound.

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

$$D_{KL}(\mathcal{N}(\mu_{z|x}, \Sigma_{z|x}) || \mathcal{N}(0, I))$$

Have analytical solution

Make approximate posterior distribution close to prior.

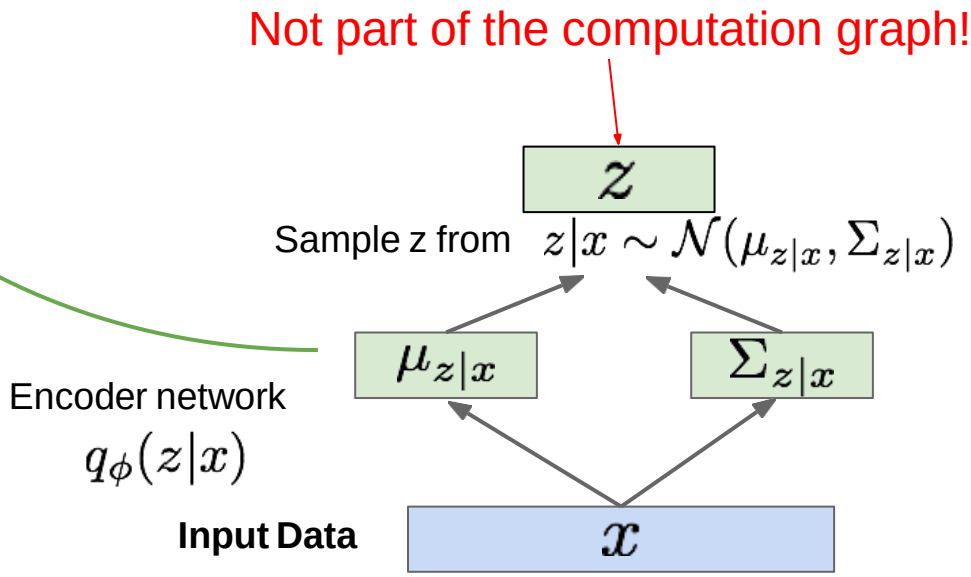


Variational Autoencoders

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

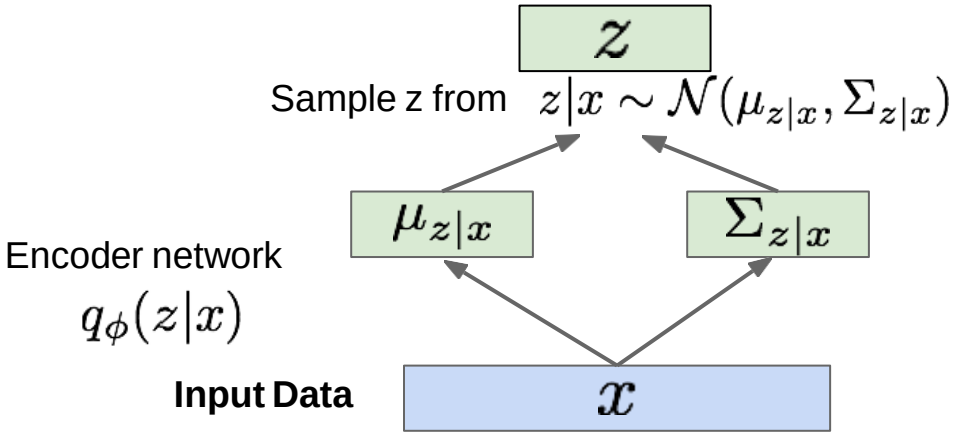
Putting it all together: maximizing the likelihood lower bound.

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

Reparameterization trick to make sampling differentiable:

Sample $\epsilon \sim \mathcal{N}(0, I)$

$$z = \mu_{z|x} + \epsilon\sigma_{z|x}$$



Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound.

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

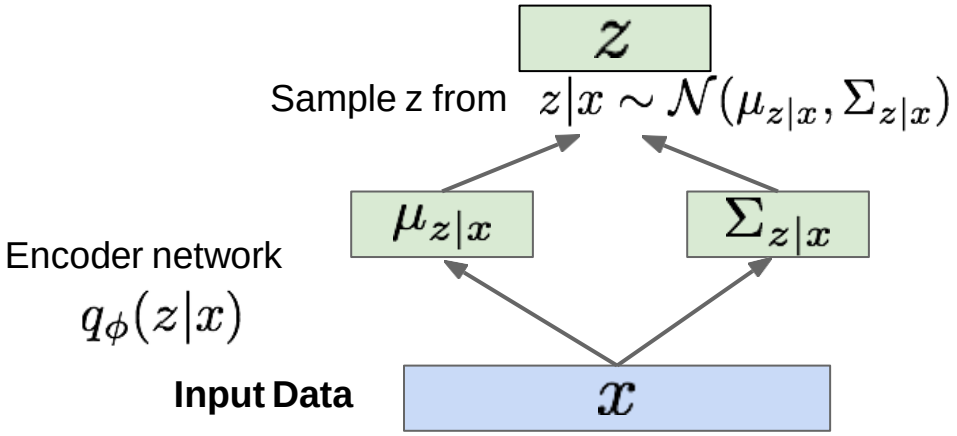
Reparameterization trick to make sampling differentiable:

Sample $\epsilon \sim \mathcal{N}(0, I)$

Input to the graph

$$z = \mu_{z|x} + \epsilon\sigma_{z|x}$$

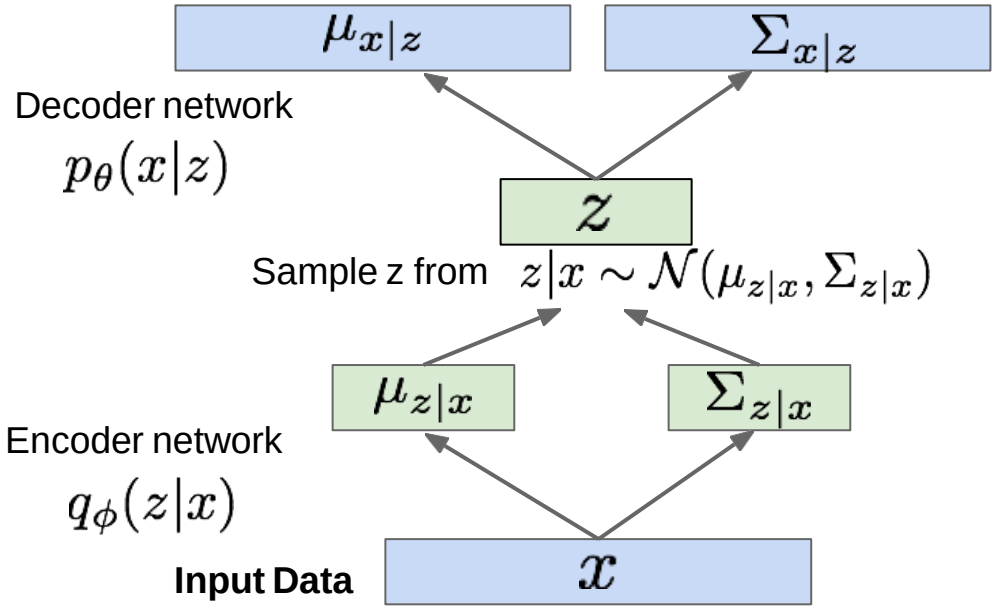
Part of computation graph



Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound.

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

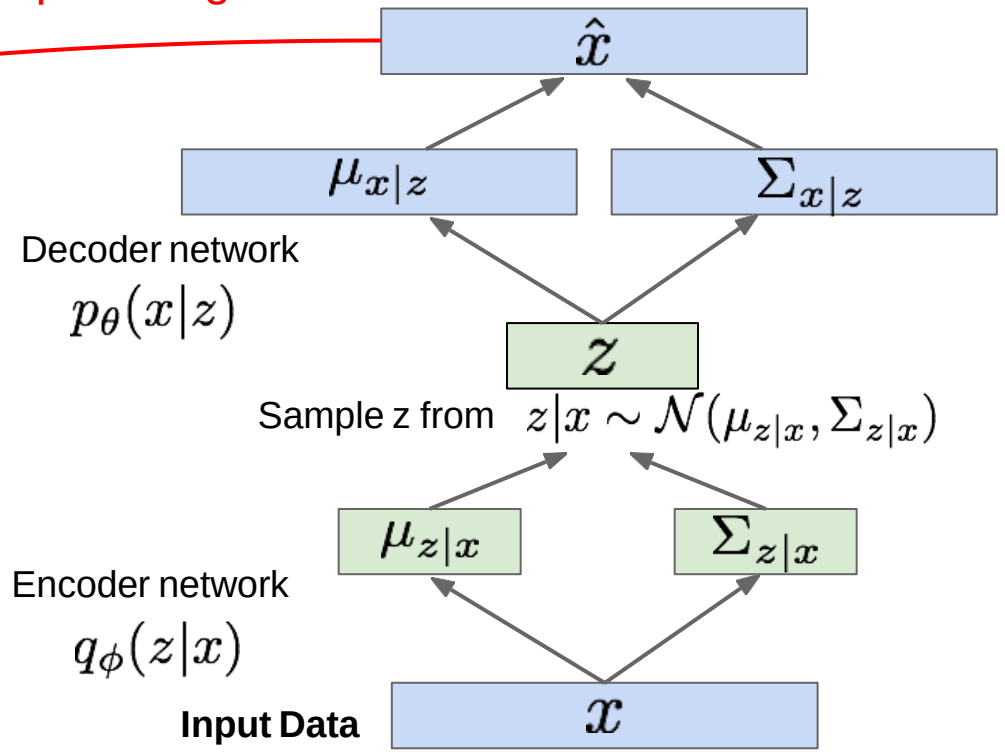


Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound.

Maximize likelihood of original input being reconstructed

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

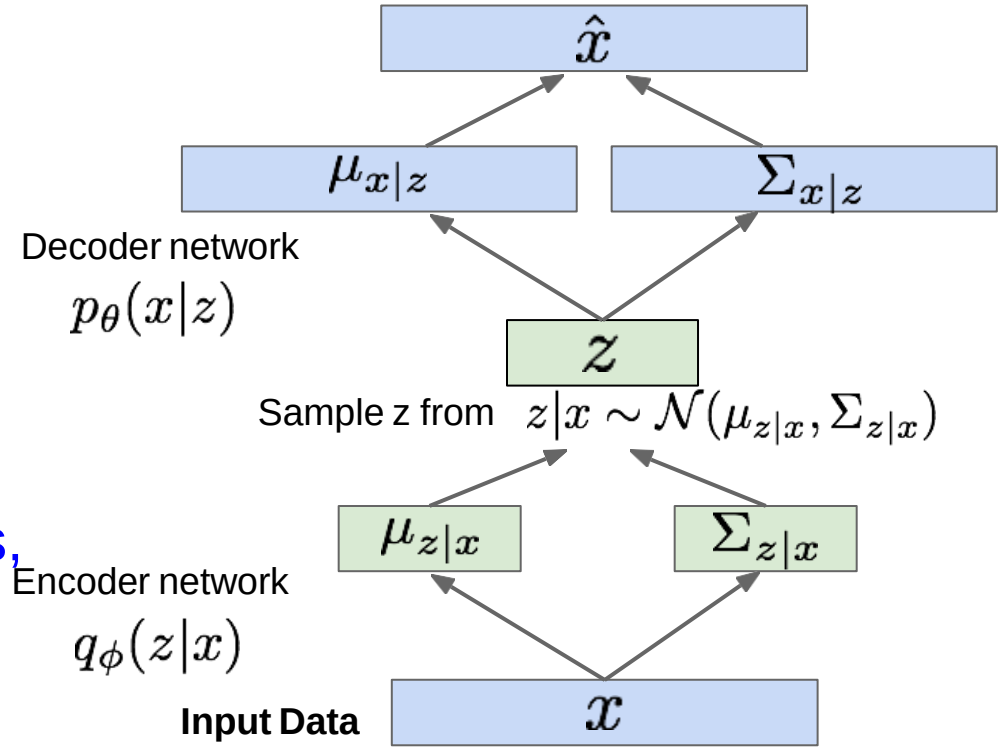


Variational Autoencoders

- Putting it all together: maximizing the likelihood lower bound.

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

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

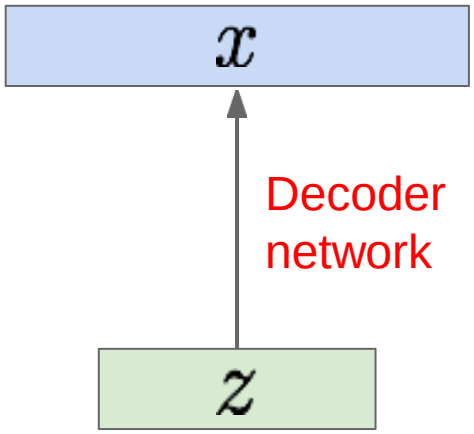


Variational Autoencoders: Generating Data!

- Our assumption about data generation process.

Sample from true conditional

$p_{\theta^*}(x \mid z^{(i)})$



Sample from true prior

$z^{(i)} \sim p_{\theta^*}(z)$

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

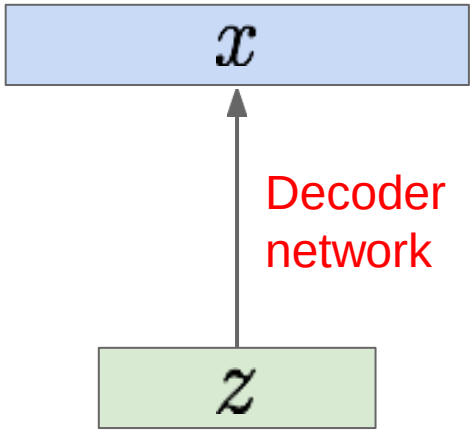
Variational Autoencoders: Generating Data!

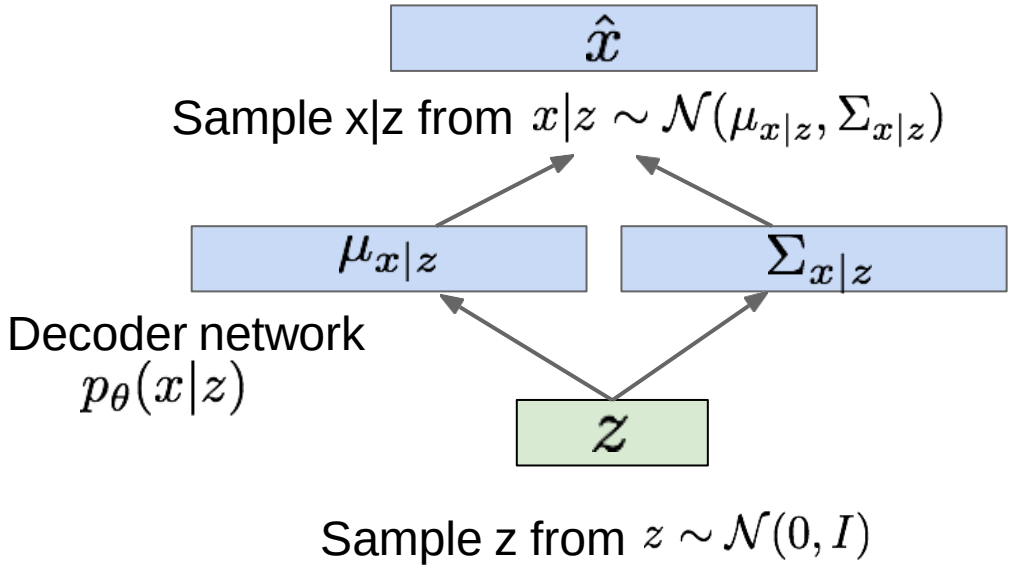
- Our assumption about data generation process.

- Now given a trained VAE: use decoder network & sample z from prior!

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

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

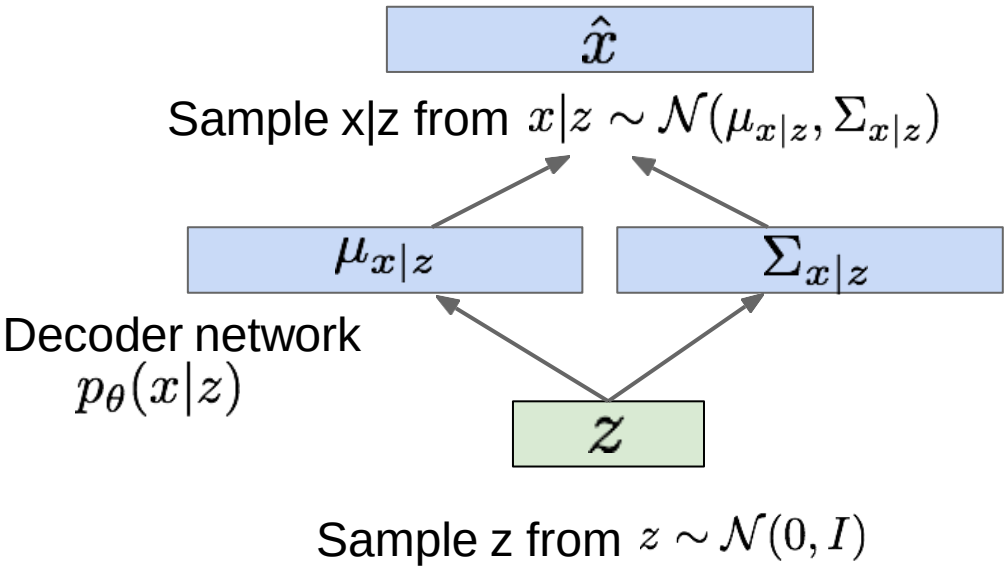


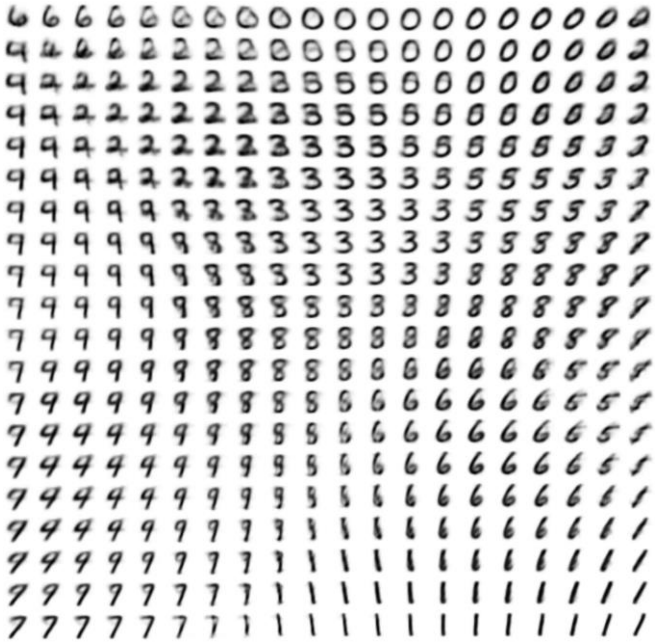


Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoders: Generating Data!

- use decoder network & sample z from prior!

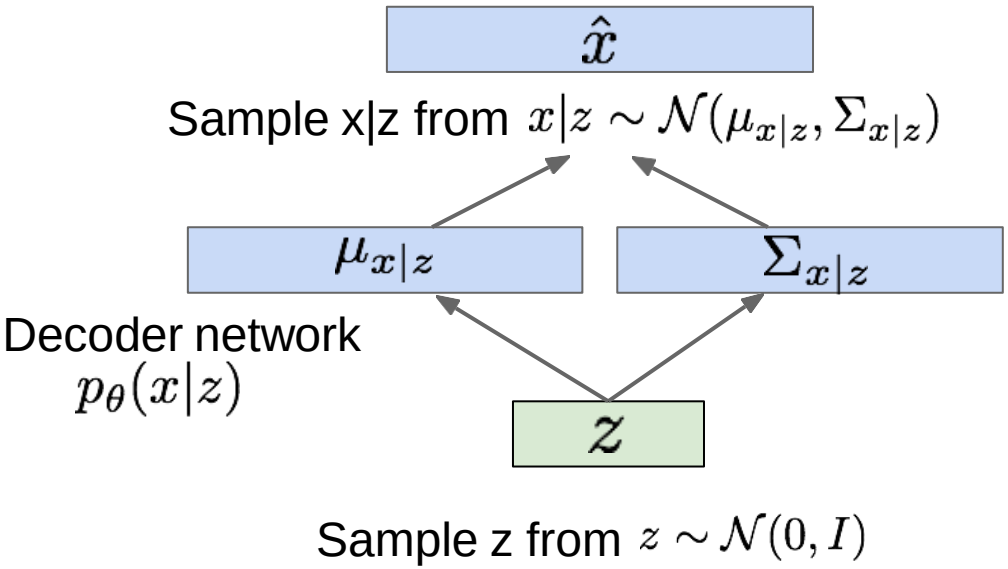


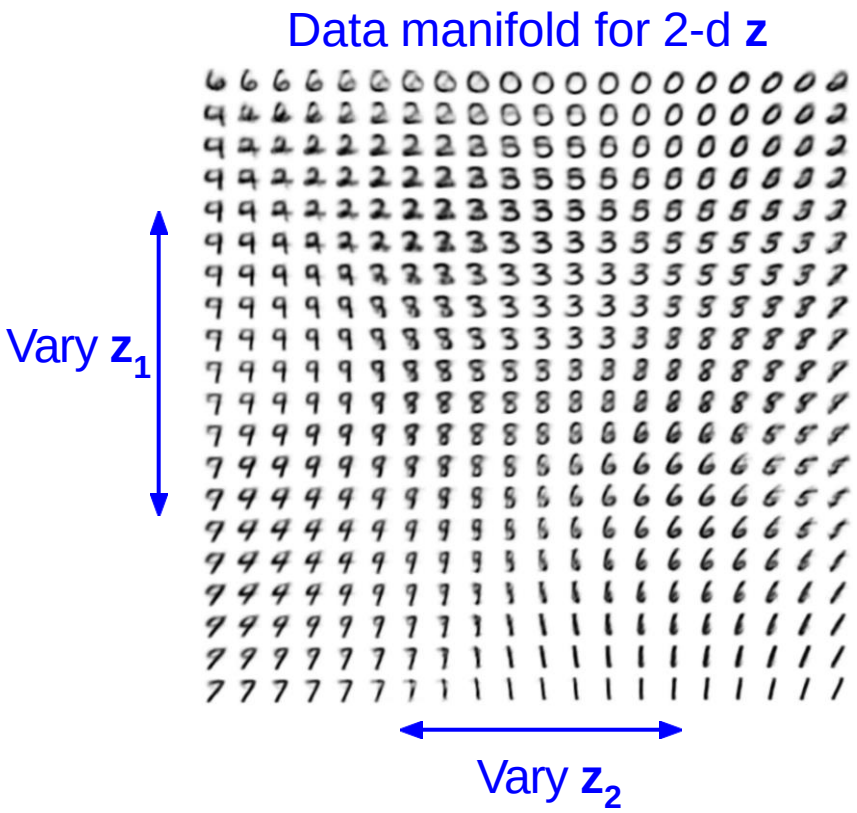


Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoders: Generating Data!

- use decoder network & sample z from prior!





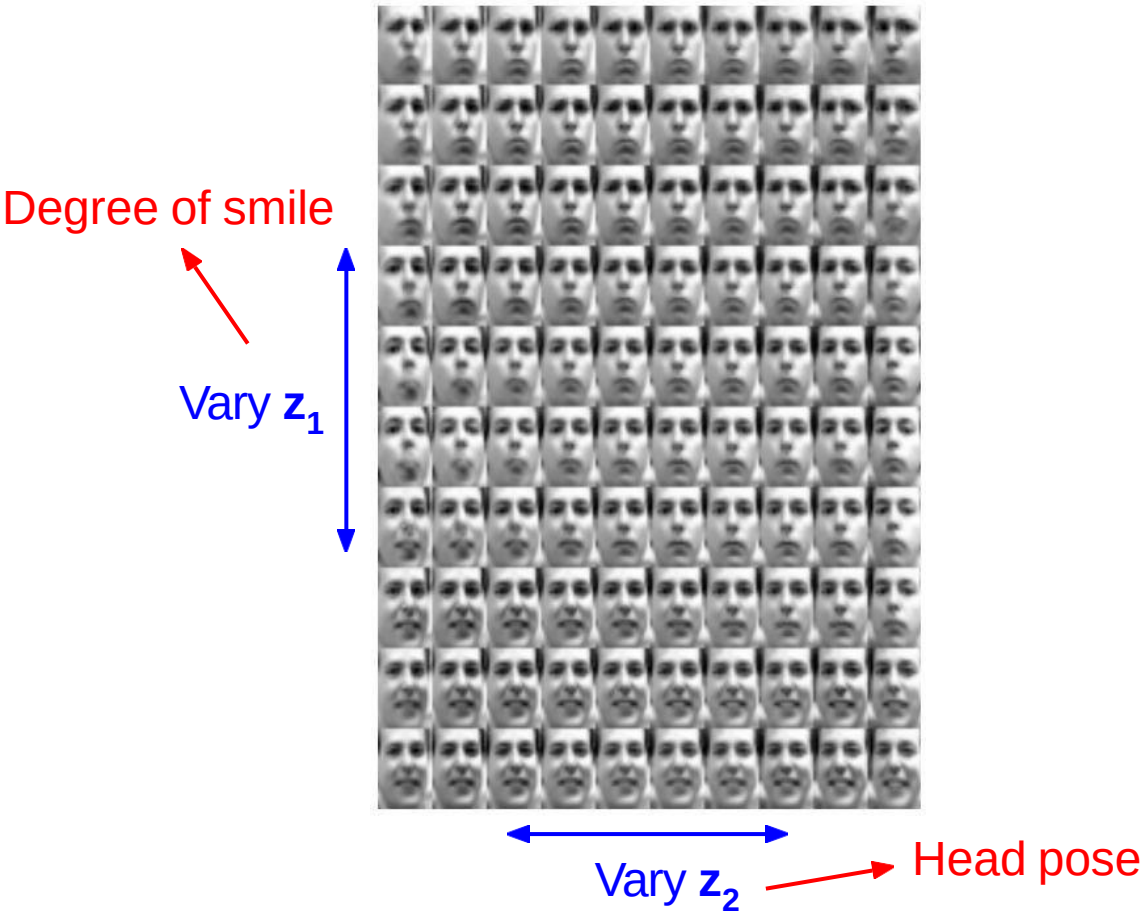
Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoders: Generating Data!

- Diagonal prior on z

=> independent latent variables.

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



Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoders: 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 $q_\phi(z|x)$!

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Variational Autoencoders: Generating Data!

32x32 CIFAR-10

Labeled Faces in the Wild

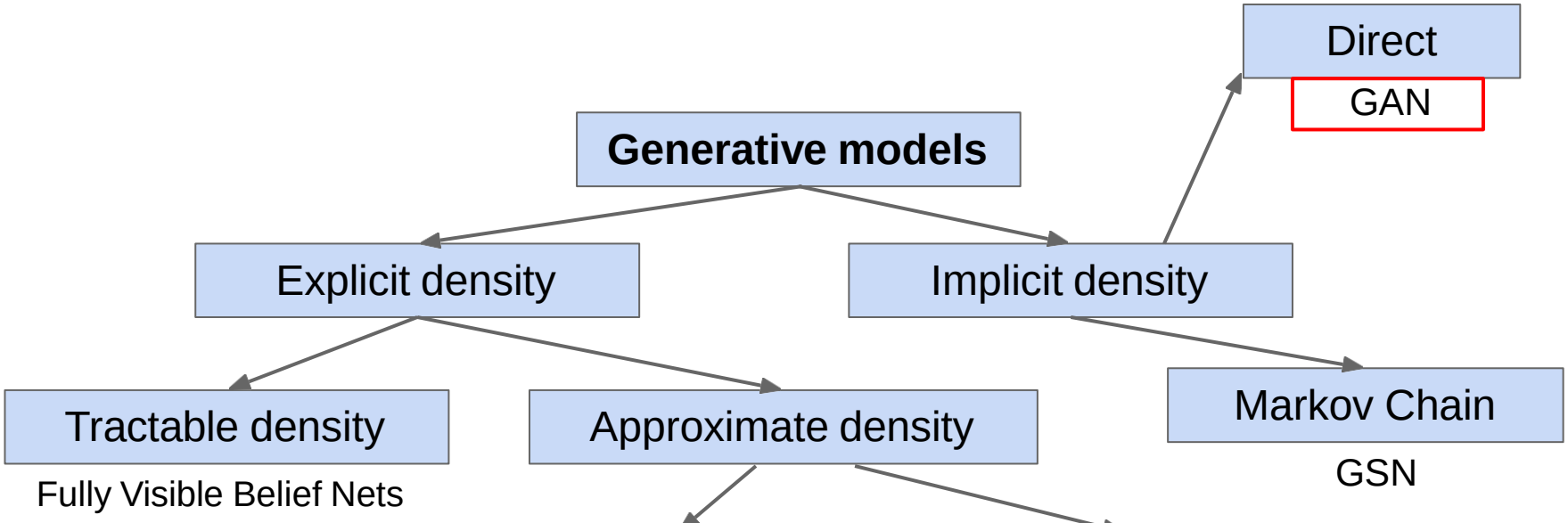
Variational Autoencoders

- 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
  - Interpretable latent space.
  - Allows inference of q(z|x), can be useful feature representation for other tasks.

Variational Autoencoders

- **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, e.g., Gaussian Mixture Models (GMMs), Categorical Distributions.
  - Learning disentangled representations.

Taxonomy of Generative Models



- NADE
- MADE
- PixelRNN/CNN
- NICE / RealNVP
- Glow
- Ffjord

Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Generative Adversarial Networks (GANs)

So far...

- PixelRNN/CNNs define tractable density function, optimize likelihood of training data:

$$p_\theta(x) = \prod_{i=1}^{n} p_\theta(x_i | x_1, ..., x_{i-1})$$

- VAEs define intractable density function with latent **z**:

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

- Cannot optimize directly, derive and optimize lower bound on likelihood instead.

So far...

- PixelRNN/CNNs define tractable density function, optimize likelihood of training data:

$$p_\theta(x) = \prod_{i=1}^{n} p_\theta(x_i | x_1, ..., x_{i-1})$$

- VAEs define intractable density function with latent **z**:

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

- Cannot optimize directly, derive and optimize lower bound on likelihood instead.

- What if we give up on explicitly modeling density, and just want ability to sample?

So far...

- PixelRNN/CNNs define tractable density function, optimize likelihood of training data:

$$p_\theta(x) = \prod_{i=1}^{n} p_\theta(x_i | x_1, ..., x_{i-1})$$

- VAEs define intractable density function with latent **z**:

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

- Cannot optimize directly, derive and optimize lower bound on likelihood instead.

- What if we give up on explicitly modeling density, and just want ability to sample?

- GANs: not modeling any explicit density function!

Generative Adversarial Networks

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

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

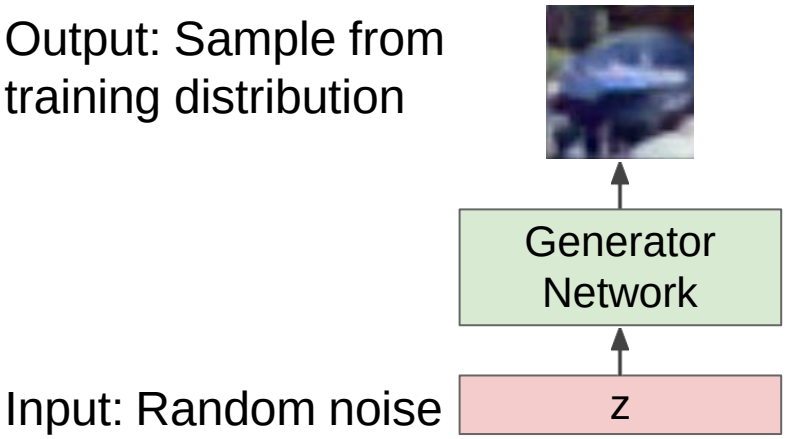
- Solution: Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.

Generative Adversarial Networks

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

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

- Solution: Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.



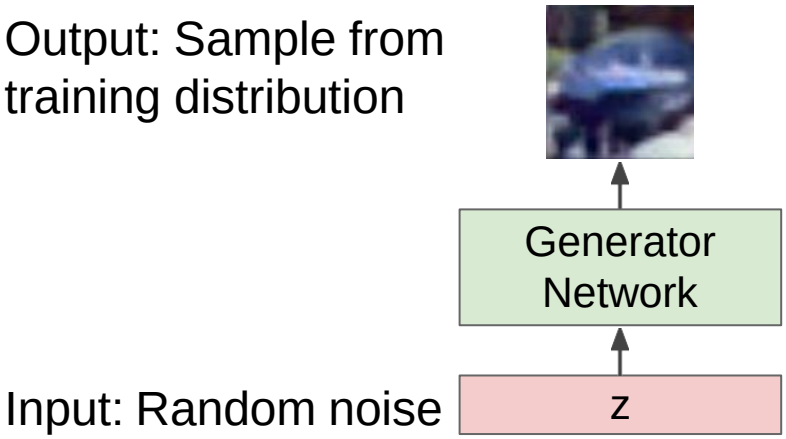
Generative Adversarial Networks

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

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

- Solution: Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.

But we don't know which sample z maps to which training image -> can't learn by reconstructing training images



Generative Adversarial Networks

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

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

- Solution: Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.

But we don't know which sample z maps to which training image -> can't learn by reconstructing training images

Output: Sample from training distribution

Generator Network

Input: Random noise

z

Objective: generated images should look "real"

Generative Adversarial Networks

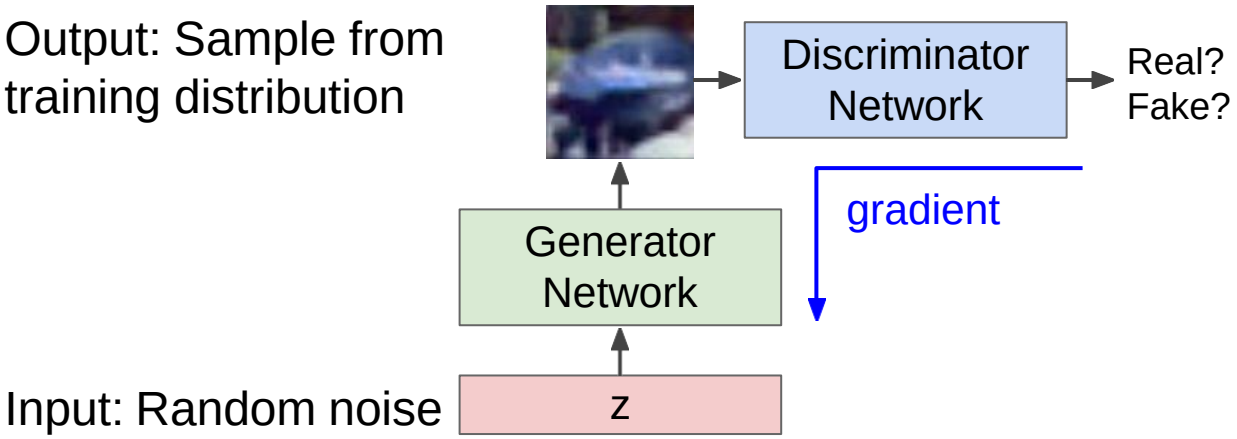
Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

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

- Solution: Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.

But we don't know which sample z maps to which training image -> can't learn by reconstructing training images.

Solution: Use a discriminator network to tell whether the generate image is within data distribution ("real") or not.



Training GANs: Two-player game

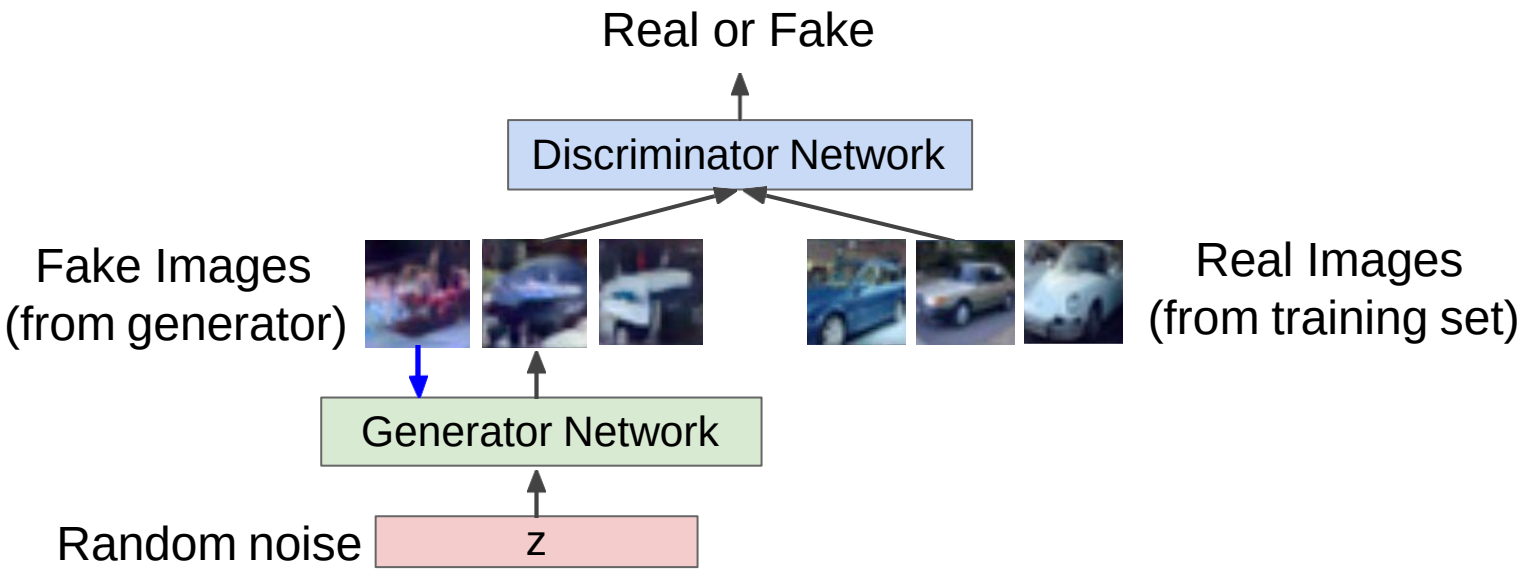
Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

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

Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

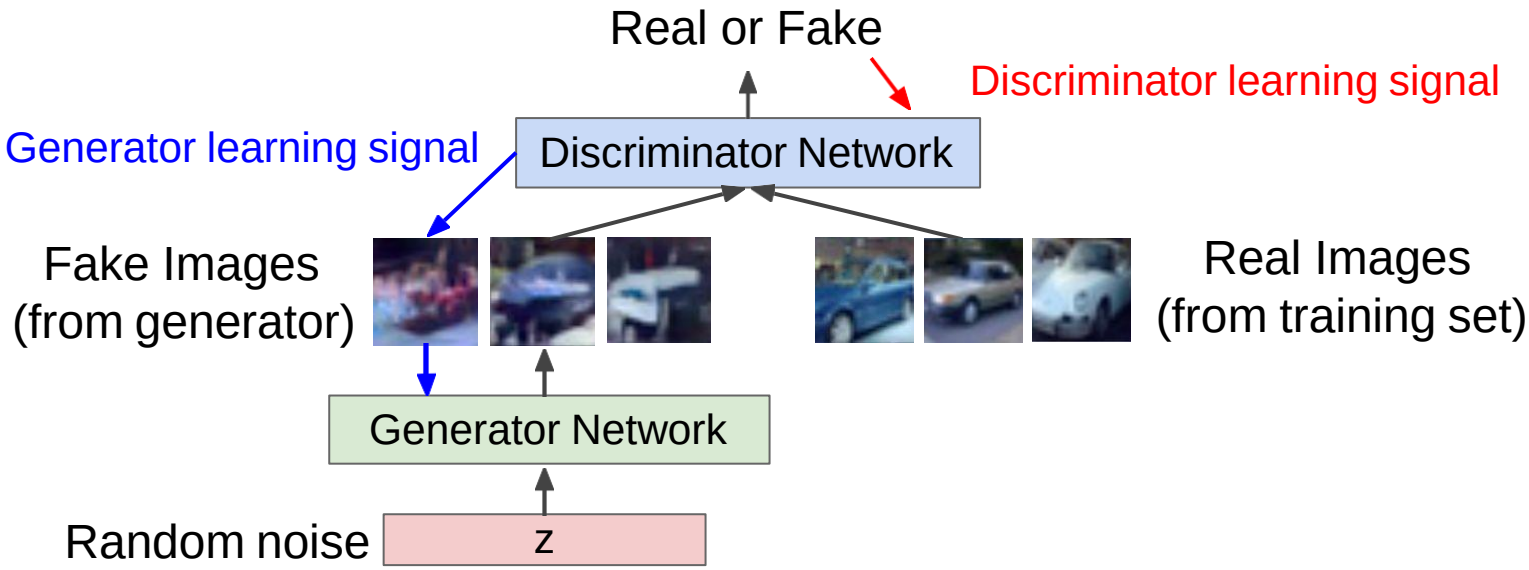
- **Discriminator network**: try to distinguish between real and fake images.
- **Generator network**: try to fool the discriminator by generating real-looking images.



Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

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



Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

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

- Train jointly in **minimax game.**
- Minimax objective function:

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

Generator objective

Discriminator objective

Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

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

Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

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

Training GANs: Two-player game

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

Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

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

- Train jointly in **minimax game.**
- Minimax objective function:

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

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log \underbrace{D_{\theta_d}(x)}_{\text{Discriminator output for real data x}} + \mathbb{E}_{z \sim p(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\text{Discriminator output for generated fake data G(z)}}) \right]$$

- 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

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

- 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 ascent** 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

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

- 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

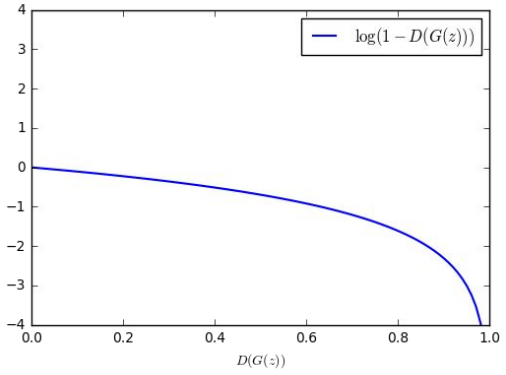
$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

- **2. Gradient ascent** 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!

When sample is likely fake, want to learn from it to improve generator (move to the right on X axis).



Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

- 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 ascent** on generator

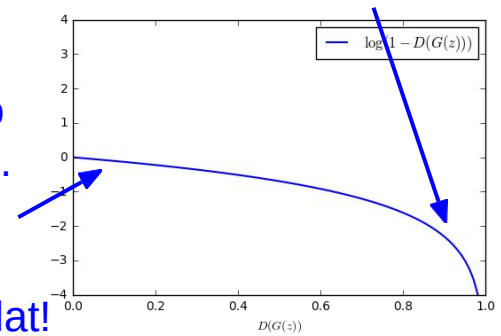
$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

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

When sample is likely fake, want to learn from it to improve generator (move to the right on X axis).

But gradient in this region is relatively flat!

Gradient signal dominated by region where sample is already good.



Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

- Minimax objective function:

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

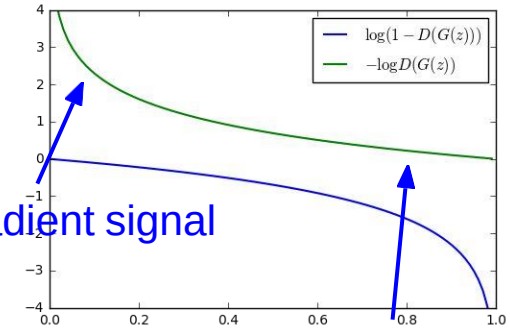
- Alternate between:
- **1. Gradient ascent** on discriminator

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

- **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.



Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

- Putting it together: GAN training algorithm

for number of training iterations **do**
  for k steps **do**
    - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
    - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
    - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$

  end for
  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by ascending its stochastic gradient (improved objective):

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

end for

Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

- Putting it together: GAN training algorithm

for number of training iterations **do**
  for k steps **do**
  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$

  end for
  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by ascending its stochastic gradient (improved objective):

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

end for

Some find k=1 more stable, others use k > 1, no best rule.

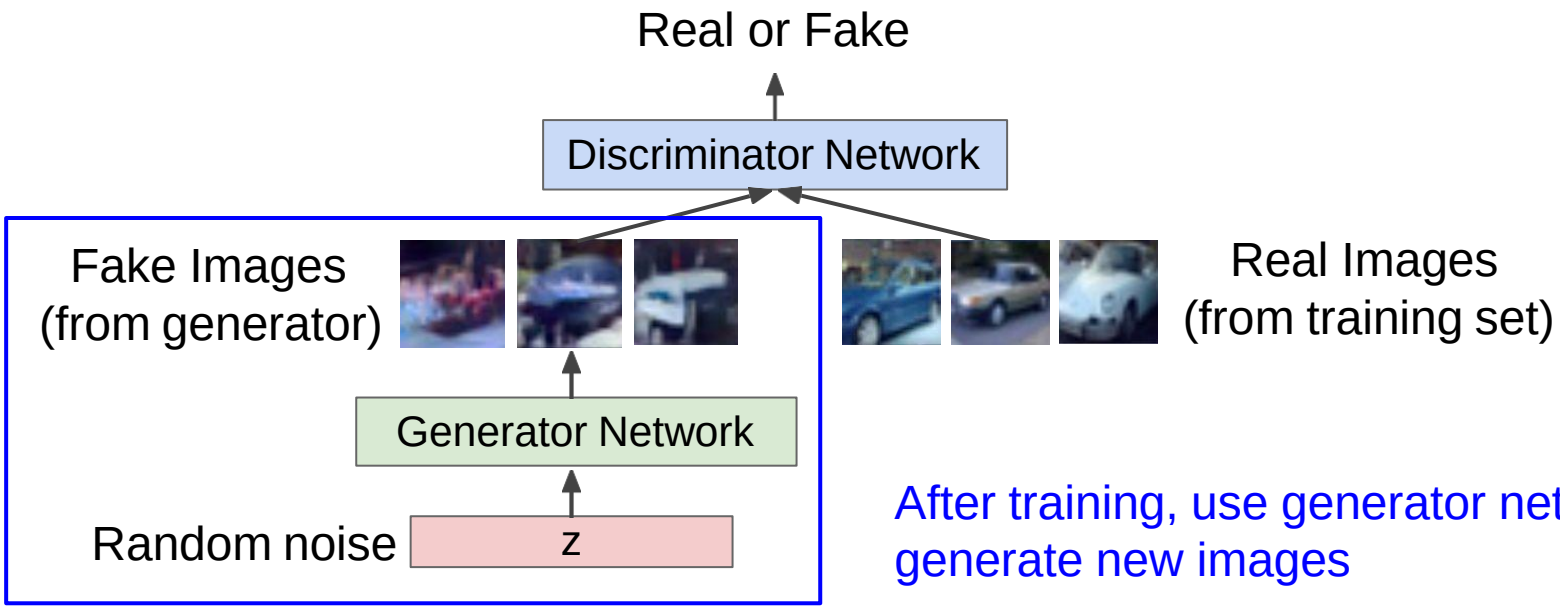
Followup work (e.g. Wasserstein GAN, BEGAN) alleviates this problem, better stability!

Arjovsky et al. "Wasserstein gan." arXiv preprint arXiv:1701.07875 (2017)
Berthelot, et al. "Began: Boundary equilibrium generative adversarial networks." arXiv preprint arXiv:1703.10717 (2017)

Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

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



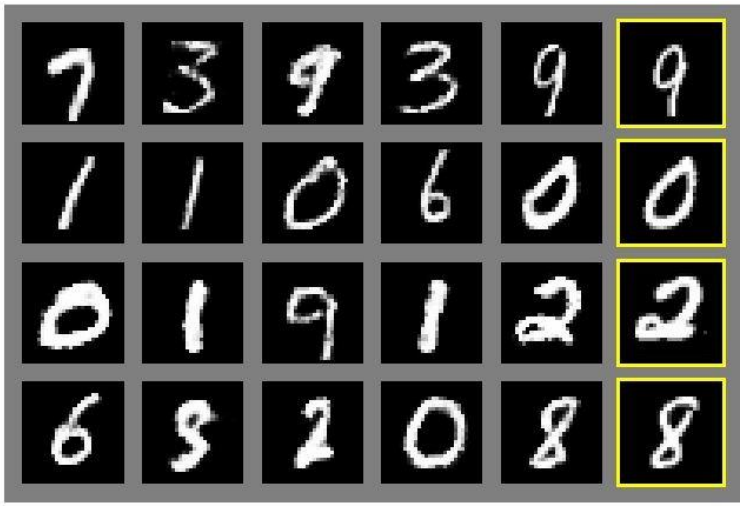
After training, use generator network to generate new images

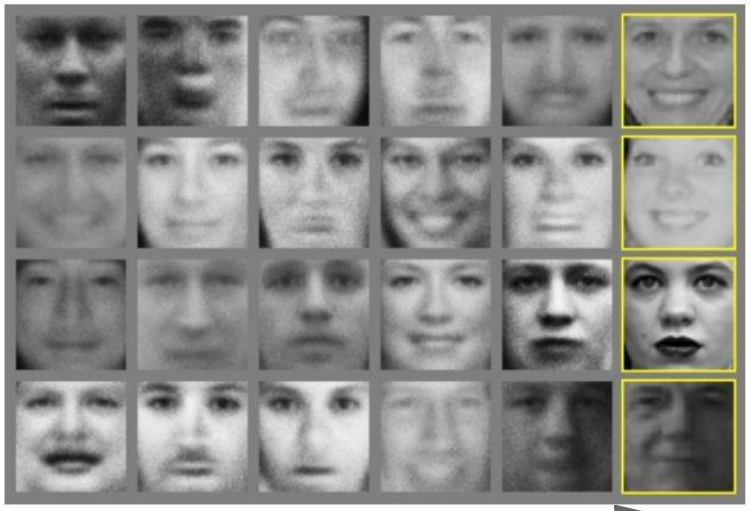
Fake and real images copyright Emily Denton et al. 2015. Reproduced with permission.

Generative Adversarial Nets

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

Generated samples



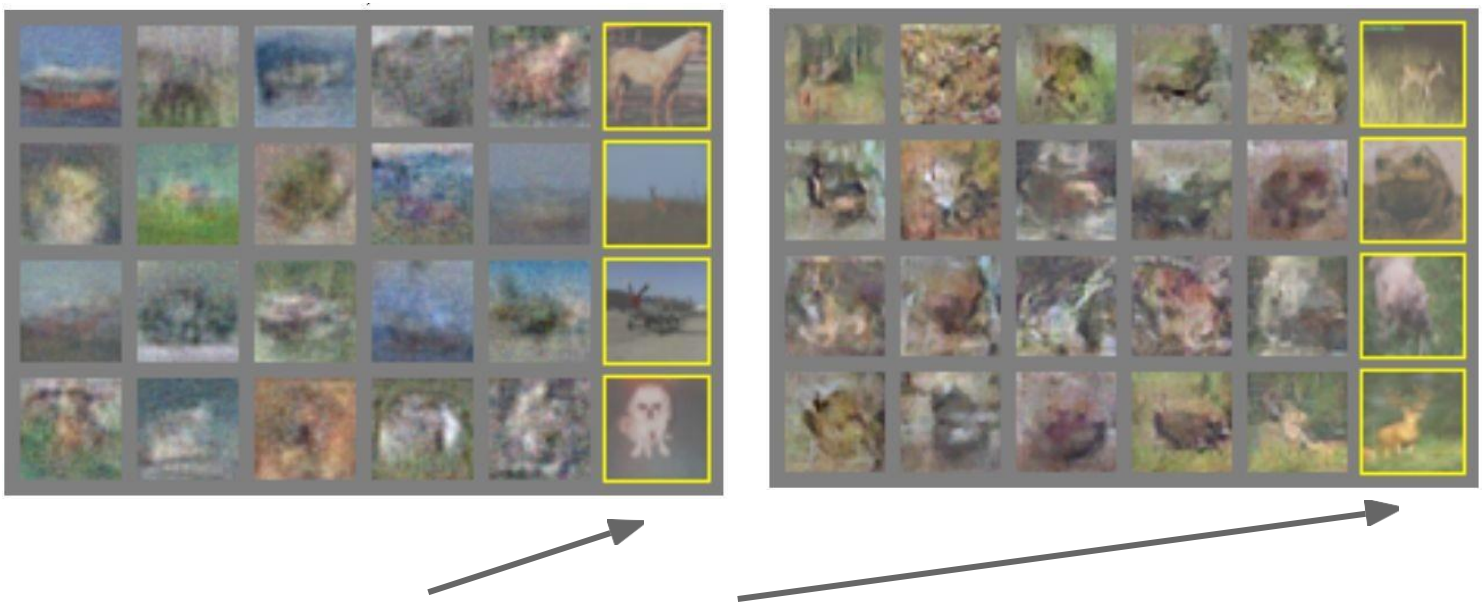


Nearest neighbor from training set

Generative Adversarial Nets

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

Generated samples (CIFAR-10)



Nearest neighbor from training set

Generative Adversarial Nets: Convolutional Architectures

- Generator is an upsampling network with fractionally-strided convolutions Discriminator is a convolutional network.

Architecture guidelines for stable Deep Convolutional GANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

Radford et al, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”, ICLR 2016

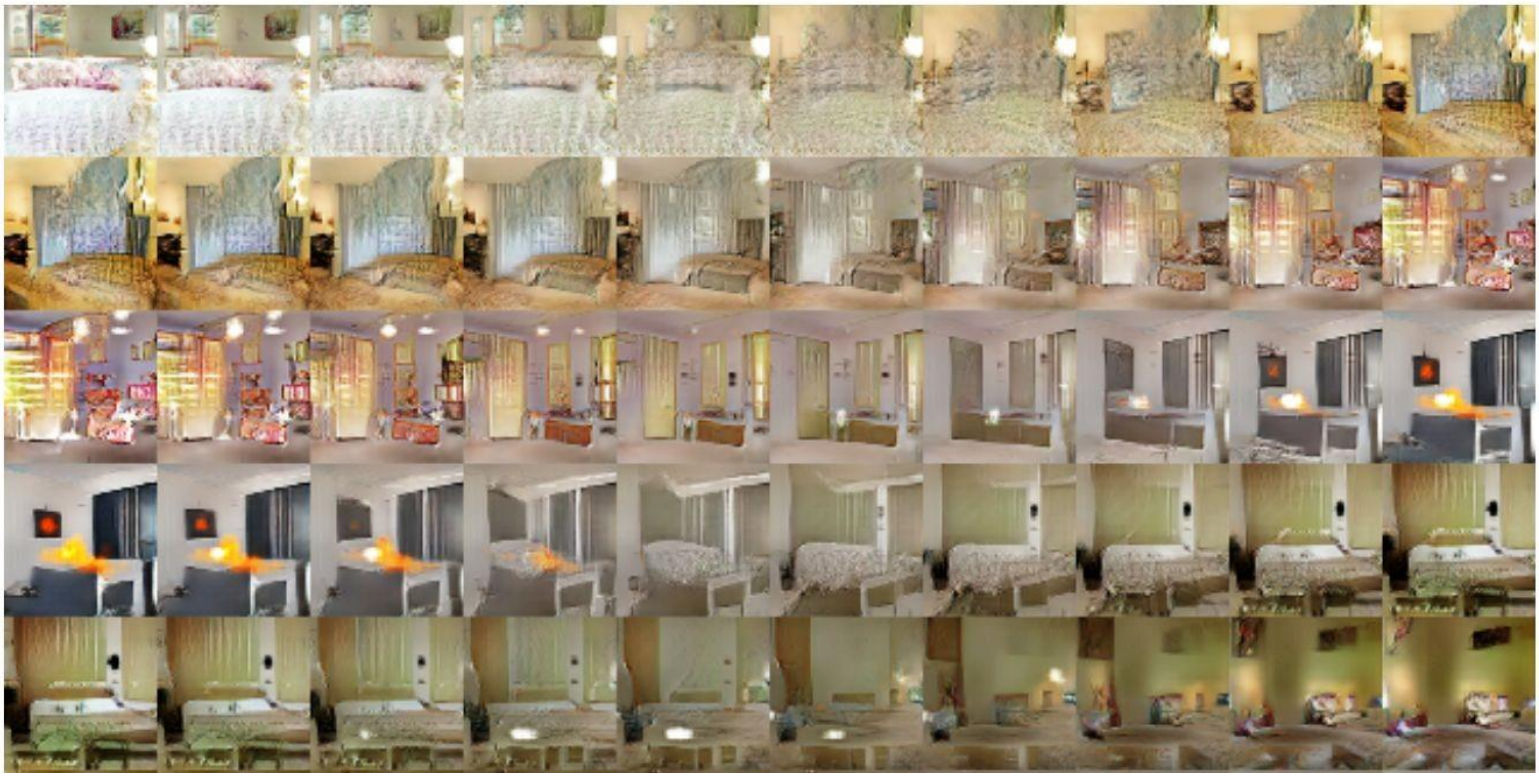
Generative Adversarial Nets: Convolutional Architectures

- Samples from the model look much better!

Radford et al, ICLR 2016

Generative Adversarial Nets: Convolutional Architectures

- Interpolating between random points in latent space.



Radford et al, ICLR 2016

Generative Adversarial Nets: Interpretable Vector Math

Radford et al, ICLR 2016

Smiling woman | Neutral woman | Neutral man

Samples from the model.

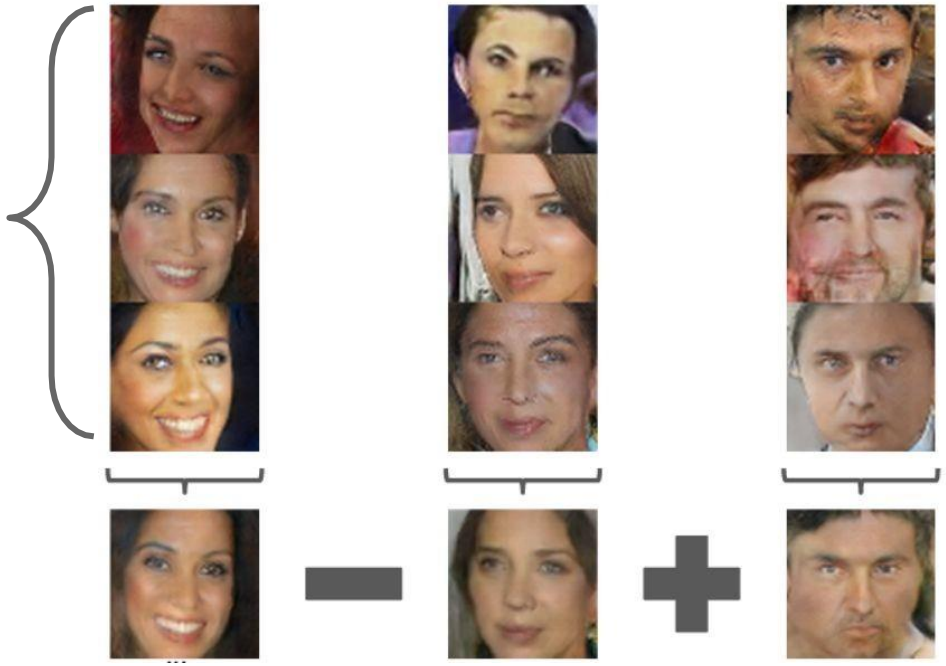
Generative Adversarial Nets: Interpretable Vector Math

Radford et al, ICLR 2016

Smiling woman Neutral woman Neutral man

Samples from the model.

Average Z vectors, do arithmetic.



Generative Adversarial Nets: Interpretable Vector Math

Radford et al, ICLR 2016

Smiling woman | Neutral woman | Neutral man

Samples from the model.

Average Z vectors, do arithmetic.

Smiling Man

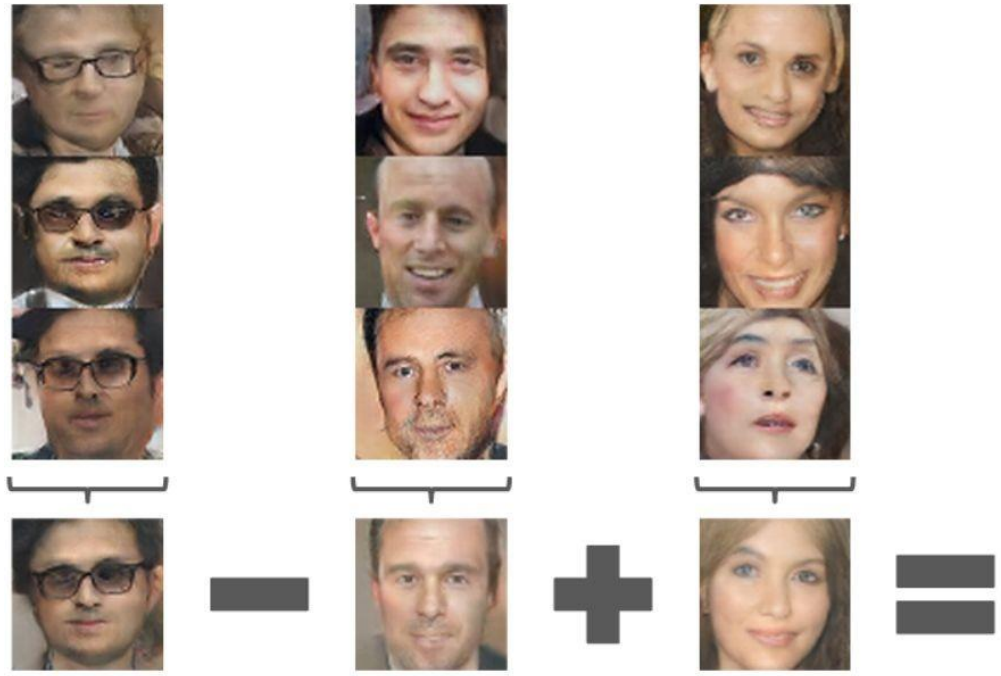
Generative Adversarial Nets: Interpretable Vector Math

Radford et al, ICLR 2016

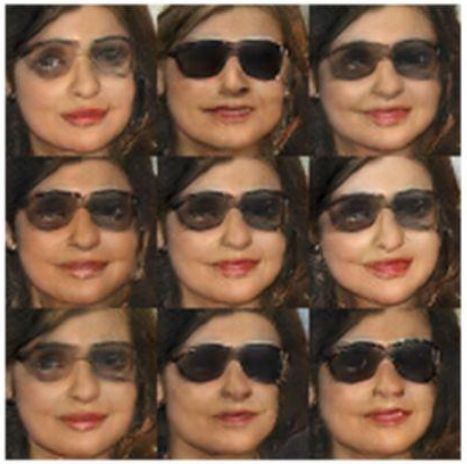
Glasses man

No glasses man

No glasses woman



Woman with glasses



2017: Explosion of GANs

See also: https://github.com/soumith/ganhacks for tips and tricks for trainings GANs

- "The GAN Zoo"

- GAN - Generative Adversarial Networks
- 3D-GAN - Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling
- acGAN - Face Aging With Conditional Generative Adversarial Networks
- AC-GAN - Conditional Image Synthesis With Auxiliary Classifier GANs
- AdaGAN - AdaGAN: Boosting Generative Models
- AEGAN - Learning Inverse Mapping by Autoencoder based Generative Adversarial Nets
- AffGAN - Amortised MAP Inference for Image Super-resolution
- AL-CGAN - Learning to Generate Images of Outdoor Scenes from Attributes and Semantic Layouts
- ALI - Adversarially Learned Inference
- AM-GAN - Generative Adversarial Nets with Labeled Data by Activation Maximization
- AnoGAN - Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery
- ArtGAN - ArtGAN: Artwork Synthesis with Conditional Categorial GANs
- b-GAN - b-GAN: Unified Framework of Generative Adversarial Networks
- Bayesian GAN - Deep and Hierarchical Implicit Models
- BEGAN - BEGAN: Boundary Equilibrium Generative Adversarial Networks
- BiGAN - Adversarial Feature Learning
- BS-GAN - Boundary-Seeking Generative Adversarial Networks
- CGAN - Conditional Generative Adversarial Nets
- CaloGAN - CaloGAN: Simulating 3D High Energy Particle Showers in Multi-Layer Electromagnetic Calorimeters with Generative Adversarial Networks
- CCGAN - Semi-Supervised Learning with Context-Conditional Generative Adversarial Networks
- CatGAN - Unsupervised and Semi-supervised Learning with Categorical Generative Adversarial Networks
- CoGAN - Coupled Generative Adversarial Networks
- Context-RNN-GAN - Contextual RNN-GANs for Abstract Reasoning Diagram Generation
- C-RNN-GAN - C-RNN-GAN: Continuous recurrent neural networks with adversarial training
- CS-GAN - Improving Neural Machine Translation with Conditional Sequence Generative Adversarial Nets
- CVAE-GAN - CVAE-GAN: Fine-Grained Image Generation through Asymmetric Training
- CycleGAN - Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks
- DTN - Unsupervised Cross-Domain Image Generation
- DCGAN - Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks
- DiscoGAN - Learning to Discover Cross-Domain Relations with Generative Adversarial Networks
- DR-GAN - Disentangled Representation Learning GAN for Pose-Invariant Face Recognition
- DualGAN - DualGAN: Unsupervised Dual Learning for Image-to-Image Translation
- EBGAN - Energy-based Generative Adversarial Network
- f-GAN - f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization
- FF-GAN - Towards Large-Pose Face Frontalization in the Wild
- GAWWN - Learning What and Where to Draw
- GeneGAN - GeneGAN: Learning Object Transfiguration and Attribute Subspace from Unpaired Data
- Geometric GAN - Geometric GAN
- GoGAN - Gang of GANs: Generative Adversarial Networks with Maximum Margin Ranking
- GP-GAN - GP-GAN: Towards Realistic High-Resolution Image Blending
- IAN - Neural Photo Editing with Introspective Adversarial Networks
- iGAN - Generative Visual Manipulation on the Natural Image Manifold
- IcGAN - Invertible Conditional GANs for image editing
- ID-CGAN - Image De-raining Using a Conditional Generative Adversarial Network
- Improved GAN - Improved Techniques for Training GANs
- InfoGAN - InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets
- LAGAN - Learning Particle Physics by Example: Location-Aware Generative Adversarial Networks for Physics Synthesis
- LAPGAN - Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks

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

2017: Explosion of GANs

- Better training and generation

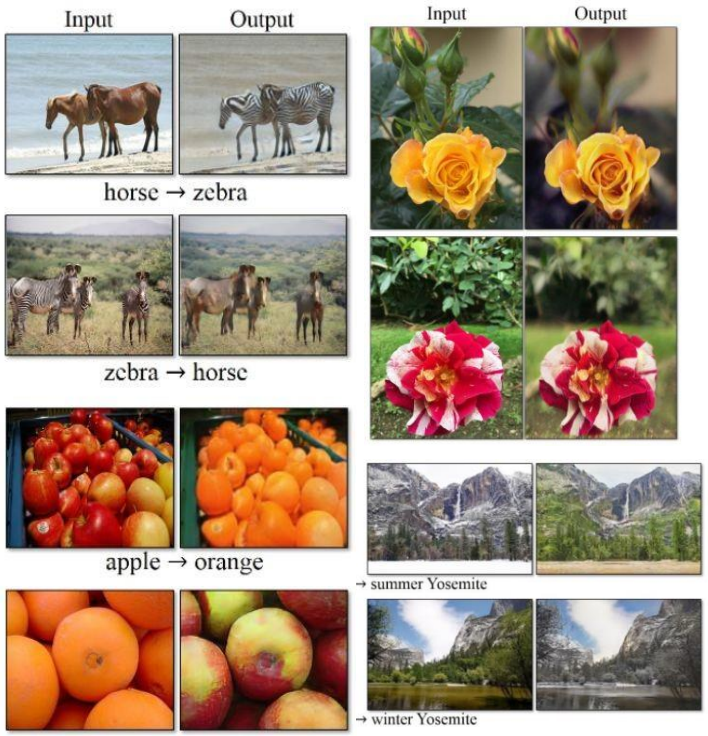
LSGAN, Zhu 2017.

Wasserstein GAN, Arjovsky 2017. Improved Wasserstein GAN, Gulrajani 2017.

Progressive GAN, Karras 2018.

2017: Explosion of GANs

- Source->Target domain transfer



CycleGAN. Zhu et al. 2017.

- Text -> Image Synthesis

Reed et al. 2017.

- Many GAN applications

Pix2pix. Isola 2017. Many examples at https://phillipi.github.io/pix2pix/

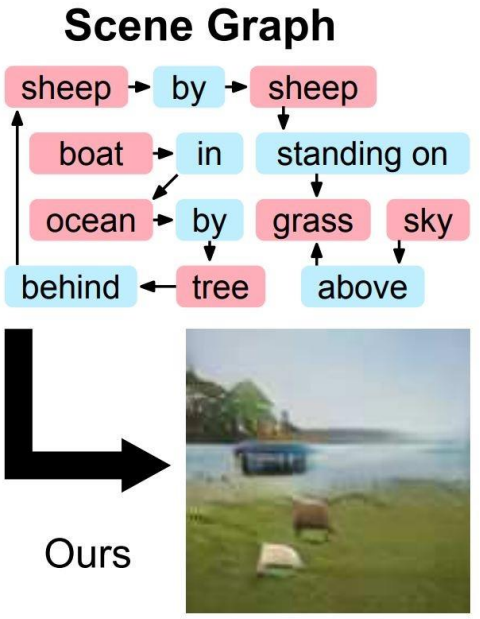
2019: BigGAN

Brock et al., 2019

Scene graphs to GANs

- Specifying exactly what kind of image you want to generate.

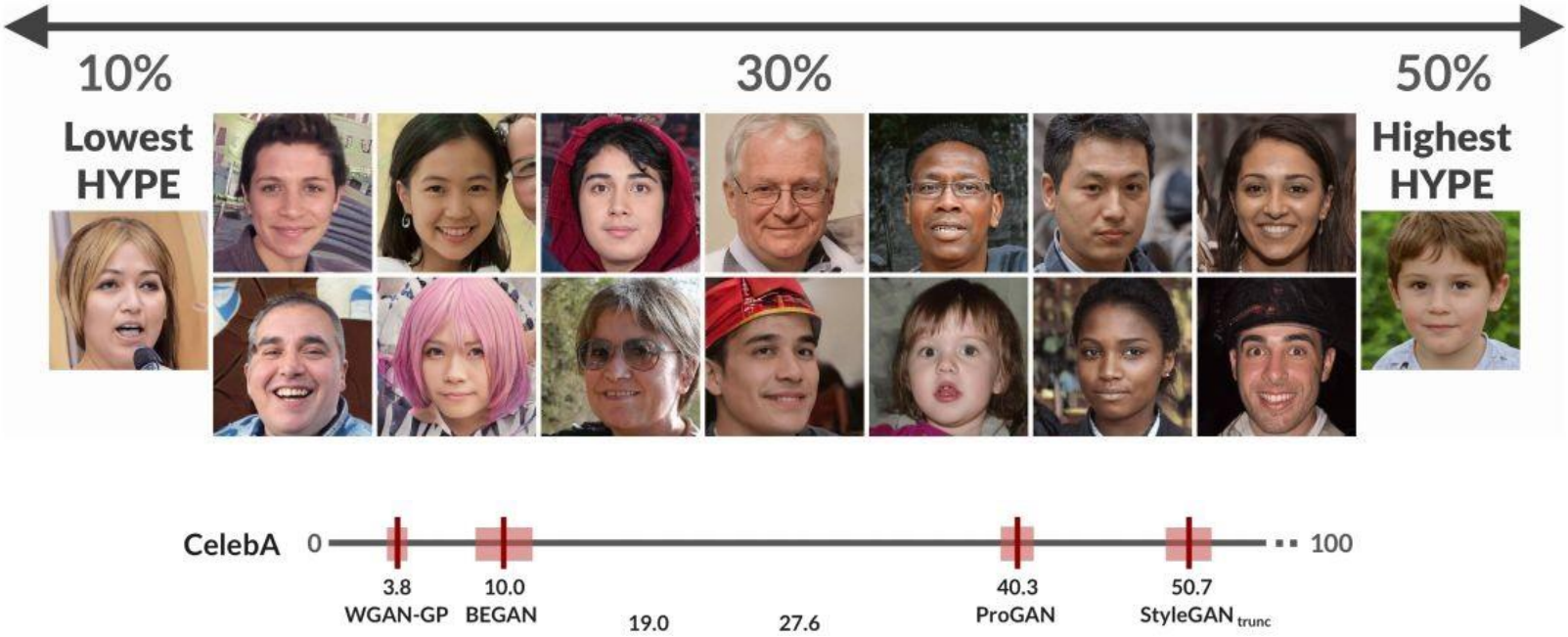
- The explicit structure in scene graphs provides better image generation for complex scenes.



Johnson et al. Image Generation from Scene Graphs, CVPR 2019

HYPE: Human eYe Perceptual Evaluations

hype.stanford.edu



Zhou, Gordon, Krishna et al. HYPE: Human eYe Perceptual Evaluations, NeurIPS 2019

Summary: GANs

- Don’t work with an explicit density function.
- Take game-theoretic approach: learn to generate from training distribution through 2-player game.

- Pros:
  - Beautiful, state-of-the-art samples!

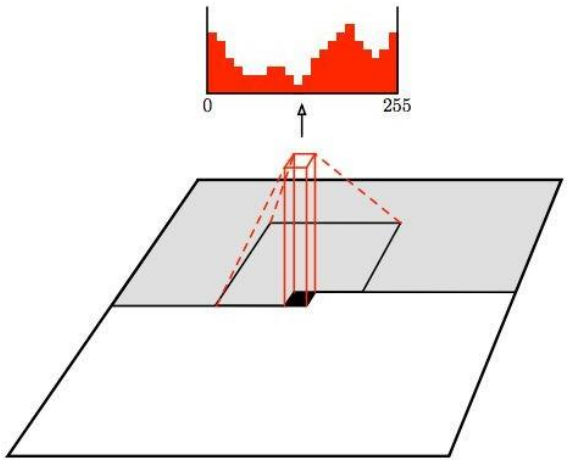
- Cons:
  - Trickier / more unstable to train.
  - Can’t solve inference queries such as p(x), p(z|x).

Summary: GANs

- Active areas of research:
  - Better loss functions, more stable training (Wasserstein GAN, LSGAN, many others).
  - Conditional GANs, GANs for all kinds of applications.

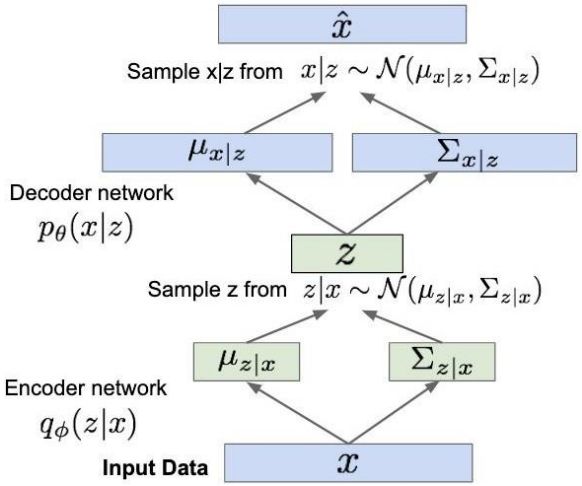
Summary

Autoregressive models: PixelRNN, PixelCNN



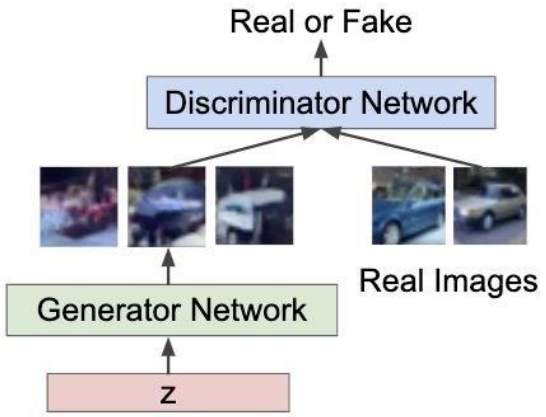
Van der Oord et al, “Conditional image generation with pixelCNN decoders”, NIPS 2016

Variational Autoencoders



Kingma and Welling, “Auto-encoding variational bayes”, ICLR 2013

Generative Adversarial Networks (GANs)



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

Useful Resources on Generative Models

- CS 236: Deep Generative Models (Stanford)
- CS 294-158 Deep Unsupervised Learning (Berkeley)

中山大學
SUN YAT-SEN UNIVERSITY

Next time:

Self-Supervised Learning

Pattern Recognition and Computer Vision

Guanbin Li,
School of Computer Science and Engineering, Sun Yat-Sen University

Most of our slides have been borrowed from the courses of Stanford CS131 and CS231N, Thanks!