Generative Models

Samuel Cheng

School of ECE
University of Oklahoma

Spring, 2018
(Slides credit to Goodfellow, Larochelle, Hinton)

Table of Contents

Review

- We talked about RNN previously. RNN can be treated as a kind of generative models. That is, able to generate samples from the model
- We will look into more generative models:
  - PixelCNN and PixelRNN
  - Generative adversarial networks (GANs)
  - Variational autoencoders

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.

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 13 - 4 May 18, 2017

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

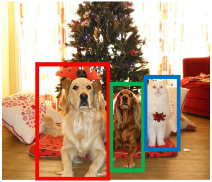
Supervised vs Unsupervised Learning

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.



DOG, DOG, CAT

Object Detection

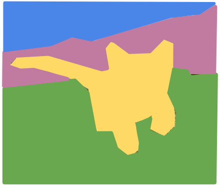
Supervised vs Unsupervised Learning

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.



Semantic Segmentation

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 13 - 7 May 18, 2017

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

Caption generated using neuraltalk2
Image is CC0 Public domain.

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 13 - 8 May 18, 2017

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.

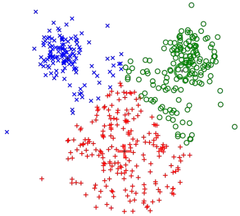
Supervised vs Unsupervised Learning

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

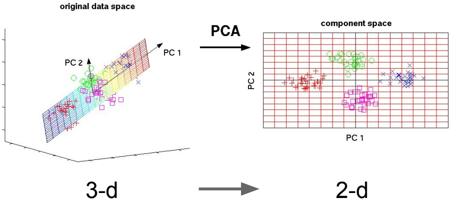
Supervised vs Unsupervised Learning

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)

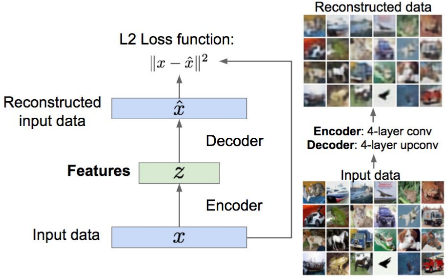
Supervised vs Unsupervised Learning

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.



Autoencoders
(Feature learning)

Supervised vs Unsupervised Learning

Unsupervised Learning

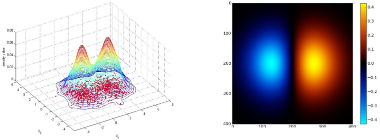
Data: x
Just data, no labels!

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

2-d density images left and right are CC0 public domain

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.

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

Training data is cheap

Data: x
Just data, no labels!

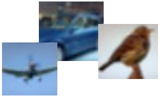
Holy grail: Solve unsupervised learning => understand structure of visual world

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

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

Generative Models

Given training data, generate new samples from same distribution



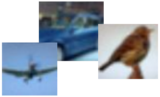
Training data ~ $p_{data}(x)$

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

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

Generative Models

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 $p_{data}(x)$

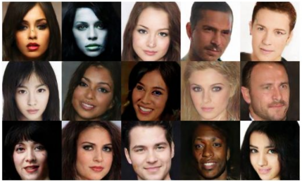
Addresses density estimation, a core problem in unsupervised learning

Several flavors:

- Explicit density estimation: explicitly define and solve for $p_{model}(x)$
- Implicit density estimation: learn model that can sample from $p_{model}(x)$ w/o explicitly defining it

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

Discriminative models vs generative models

- Discriminative models try to discriminate if one input is different from another. But it is not possible to generate samples from the models. Many classifiers are based on discriminative models, for example, support vector machines
- Generative models on the other hand can generate simulated data, for example, PixelCNN
- Many older machine learning problems are classification problems. Discriminative models provide a more direct solution and thus were more attractive
- Generative models have gained quite some attentions in recent years
  - Generate labeled simulation data for semi-supervised learning
  - Simulate data for planning and reinforcement learning

Discriminative models vs generative models

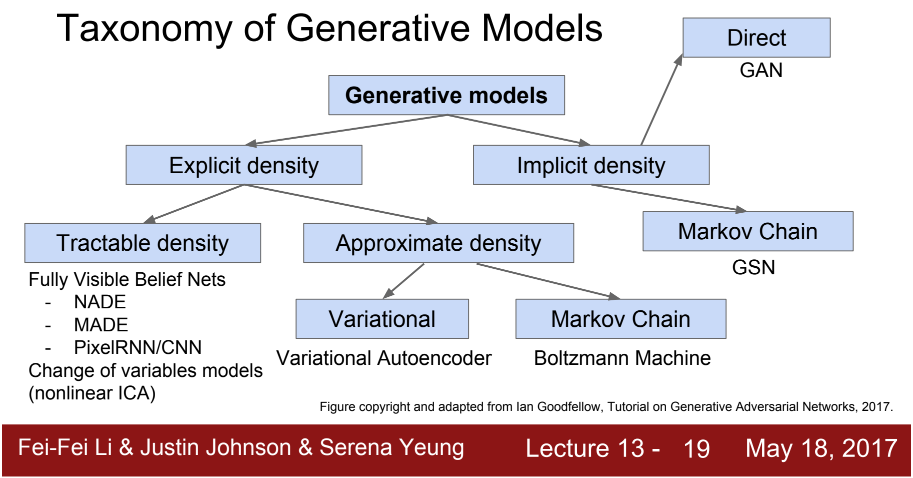
- Discriminative models try to discriminate if one input is different from another. But it is not possible to generate samples from the models. Many classifiers are based on discriminative models, for example, support vector machines
- Generative models on the other hand can generate simulated data, for example, PixelCNN
- Many older machine learning problems are classification problems. Discriminative models provide a more direct solution and thus were more attractive
- Generative models have gained quite some attentions in recent years
    - Generate labeled simulation data for semi-supervised learning
    - Simulate data for planning and reinforcement learning

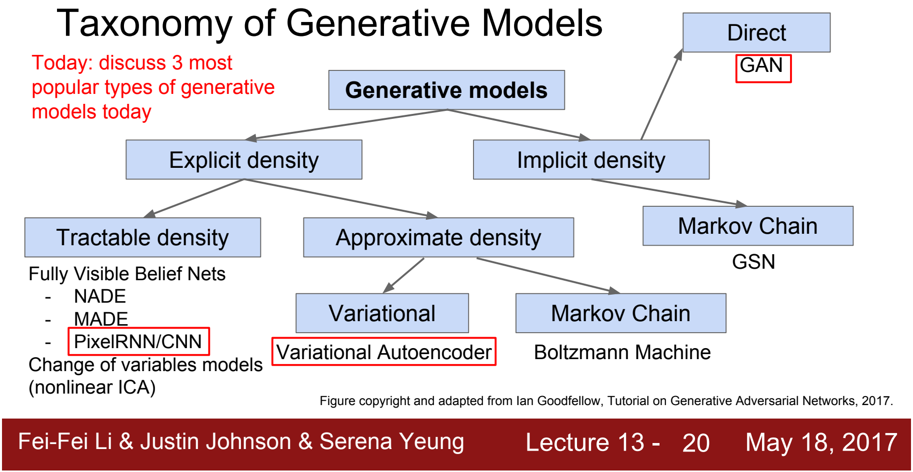
Discriminative models vs generative models

- Discriminative models try to discriminate if one input is different from another. But it is not possible to generate samples from the models. Many classifiers are based on discriminative models, for example, support vector machines
- Generative models on the other hand can generate simulated data, for example, PixelCNN
- Many older machine learning problems are classification problems. Discriminative models provide a more direct solution and thus were more attractive
- Generative models have gained quite some attentions in recent years
  - Generate labeled simulation data for semi-supervised learning
  - Simulate data for planning and reinforcement learning

Discriminative models vs generative models

- Discriminative models try to discriminate if one input is different from another. But it is not possible to generate samples from the models. Many classifiers are based on discriminative models, for example, support vector machines
- Generative models on the other hand can generate simulated data, for example, PixelCNN
- Many older machine learning problems are classification problems. Discriminative models provide a more direct solution and thus were more attractive
- Generative models have gained quite some attentions in recent years
  - Generate labeled simulation data for semi-supervised learning
  - Simulate data for planning and reinforcement learning


Taxonomy of Generative Models
Generative models
Explicit density
Implicit density
Direct
GAN
Markov Chain
GSN
Tractable density
Approximate density
Fully Visible Belief Nets
- NADE
- MADE
- PixelRNN/CNN
Change of variables models (nonlinear ICA)
Variational
Variational Autoencoder
Markov Chain
Boltzmann Machine
Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.
Fei-Fei Li & Justin Johnson & Serena Yeung
Lecture 13 - 19
May 18, 2017


Taxonomy of Generative Models
Today: discuss 3 most popular types of generative models today
Generative models
Explicit density
Implicit density
Direct
GAN
Tractable density
Approximate density
Markov Chain
GSN
Fully Visible Belief Nets
- NADE
- MADE
- PixelRNN/CNN
Change of variables models (nonlinear ICA)
Variational
Variational Autoencoder
Markov Chain
Boltzmann Machine
Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.
Fei-Fei Li & Justin Johnson & Serena Yeung
Lecture 13 - 20
May 18, 2017

PixelRNN and PixelCNN

Fully visible belief network

Explicit density model

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

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

Likelihood of image x

Probability of i'th pixel value given all previous pixels

Then maximize likelihood of training data

Fei-Fei Li & Justin Johnson & Serena Yeung   Lecture 13 - 22   May 18, 2017

Fully visible belief network

Explicit density model

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

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

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

Fully visible belief network

Explicit density model

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

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

Likelihood of image x

Probability of i'th pixel value given all previous pixels

Will need to define ordering of "previous pixels"

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

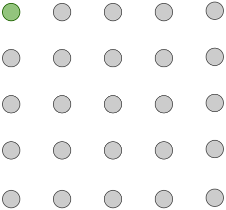
Then maximize likelihood of training data

Fei-Fei Li & Justin Johnson & Serena Yeung   Lecture 13 - 24   May 18, 2017

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

Generate image pixels starting from corner

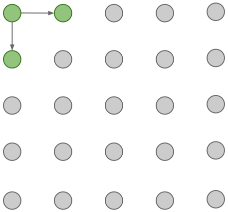
Dependency on previous pixels modeled using an RNN (LSTM)



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

Generate image pixels starting from corner

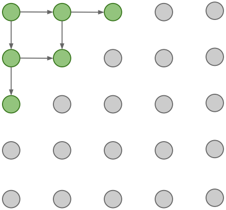
Dependency on previous pixels modeled using an RNN (LSTM)



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

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)

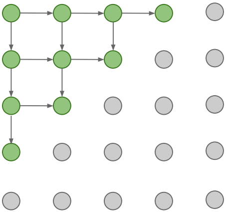


PixelRNN *[van der Oord et al. 2016]*

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)

Drawback: sequential generation is slow!

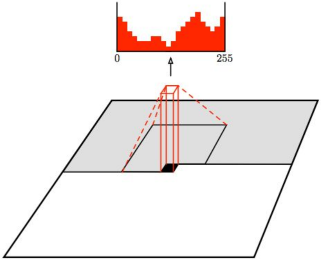


Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 13 - 28 May 18, 2017

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



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

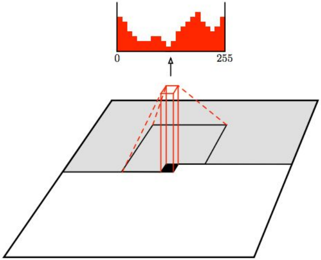
Still generate image pixels starting from corner

Dependency on previous pixels now modeled using a CNN over context region

Training: maximize likelihood of training images

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

Softmax loss at each pixel



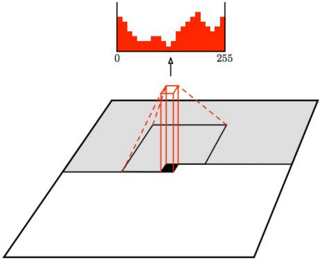
PixelCNN *[van der Oord et al. 2016]*

Still generate image pixels starting from corner

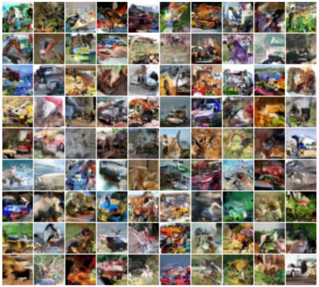
Dependency on previous pixels now modeled using a CNN over context region

Training is faster than PixelRNN
(can parallelize convolutions since context region values known from training images)

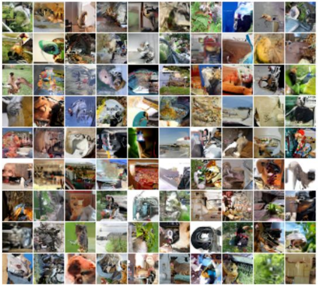
Generation must still proceed sequentially
=> still slow



Generation Samples



32x32 CIFAR-10



32x32 ImageNet

PixelRNN and PixelCNN

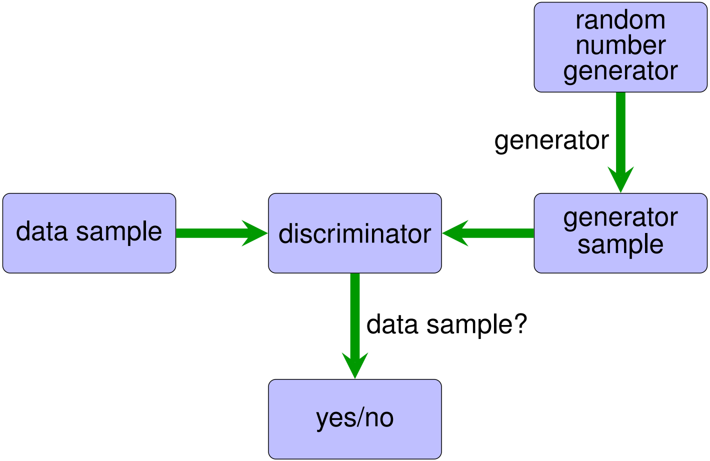
Pros:
- Can explicitly compute likelihood p(x)
- Explicit likelihood of training data gives good evaluation metric
- Good samples

Con:
- Sequential generation => slow

Improving PixelCNN performance
- Gated convolutional layers
- Short-cut connections
- Discretized logistic loss
- Multi-scale
- Training tricks
- Etc…

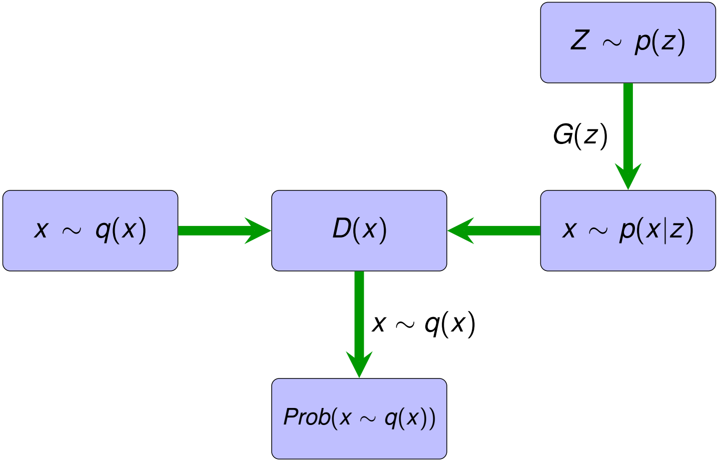
See
- Van der Oord et al. NIPS 2016
- Salimans et al. 2017 (PixelCNN++)

Generative adversarial networks (GANs)
Goodfellow *et al.* 2014



Generative adversarial networks (GANs)

Goodfellow *et al.* 2014



Minimax game of a GAN

- Probability of model data: $p_{model}(x) = \int_z p(z)p(x|z)dz$
- Probability of true data: $p_{data}(x) = q(x)$
- Discriminator wants to catch fake data

$$J^{(D)} = -E_{x\sim p_{data}} \log D(x) - E_z \log(1 - D(G(z)))$$
$$= -E_{x\sim p_{data}} \log D(x) - E_{x\sim p_{model}} \log(1 - D(x))$$

  - N.B. $J^{(D)}$ is just cross-entropy loss for correct classification
- Generator wants to fool the discriminator: $J^{(G)} = -J^{(D)}$
  - Since first term does not depend on $G(\cdot)$, we can simplify $J^{(G)}$ to

$$J^{(G)} = -E_z \log(1 - D(G(z)))$$

Minimax game of a GAN

- Probability of model data: $p_{model}(x) = \int_z p(z)p(x|z)dz$
- Probability of true data: $p_{data}(x) = q(x)$
- Discriminator wants to catch fake data

$$
\begin{aligned}
J^{(D)} &= -E_{x \sim p_{data}} \log D(x) - E_z \log(1 - D(G(z))) \\
&= -E_{x \sim p_{data}} \log D(x) - E_{x \sim p_{model}} \log(1 - D(x))
\end{aligned}
$$

  - N.B. $J^{(D)}$ is just cross-entropy loss for correct classification
- Generator wants to fool the discriminator: $J^{(G)} = -J^{(D)}$
  - Since first term does not depend on $G(\cdot)$, we can simplify $J^{(G)}$ to

$$J^{(G)} = -E_z \log(1 - D(G(z)))$$

Minimax game of a GAN

- Probability of model data: $p_{model}(x) = \int_z p(z)p(x|z)dz$
- Probability of true data: $p_{data}(x) = q(x)$
- Discriminator wants to catch fake data

$$
\begin{aligned}
J^{(D)} &= -E_{x\sim p_{data}} \log D(x) - E_z \log(1 - D(G(z))) \\
&= -E_{x\sim p_{data}} \log D(x) - E_{x\sim p_{model}} \log(1 - D(x))
\end{aligned}
$$

  - N.B. $J^{(D)}$ is just cross-entropy loss for correct classification
- Generator wants to fool the discriminator: $J^{(G)} = -J^{(D)}$
  - Since first term does not depend on $G(\cdot)$, we can simplify $J^{(G)}$ to

$$J^{(G)} = -E_z \log(1 - D(G(z)))$$

Minimax game of a GAN

- Probability of model data: $p_{model}(x) = \int_z p(z)p(x|z)dz$
- Probability of true data: $p_{data}(x) = q(x)$
- Discriminator wants to catch fake data

$$\begin{aligned} J^{(D)} &= -E_{x\sim p_{data}} \log D(x) - E_z \log(1 - D(G(z))) \\ &= -E_{x\sim p_{data}} \log D(x) - E_{x\sim p_{model}} \log(1 - D(x)) \end{aligned}$$

  - N.B. $J^{(D)}$ is just cross-entropy loss for correct classification
- Generator wants to fool the discriminator: $J^{(G)} = -J^{(D)}$
  - Since first term does not depend on $G(\cdot)$, we can simplify $J^{(G)}$ to

$$J^{(G)} = -E_z \log(1 - D(G(z)))$$

Optimal discriminator $D^*(x)$

By calculus of variations, for any $\Delta(x)$,

$$\left.\frac{\partial J^{(D)}(D^*(X)+\lambda\Delta(x))}{\partial\lambda}\right|_{\lambda=0}=0$$

$$\Rightarrow -\left.\frac{\partial E_{x\sim p_{data}}\log(D^*(x)+\lambda\Delta(x))}{\partial\lambda}-\frac{\partial E_{x\sim p_{model}}\log(1-D^*(x)-\lambda\Delta(x))}{\partial\lambda}\right|_{\lambda=0}=0$$

$$\Rightarrow -E_{x\sim p_{data}}\left[\frac{1}{D^*(x)+\lambda\Delta(x)}\right]+E_{x\sim p_{model}}\left.\left[\frac{1}{1-D^*(x)-\lambda\Delta(x)}\right]\right|_{\lambda=0}=0$$

$$\Rightarrow \int_x\left[\frac{p_{data}(x)}{D^*(x)}-\frac{p_{model}(x)}{1-D^*(x)}\right]dx=0$$

$$\Rightarrow D^*(x)=\frac{p_{data}(x)}{p_{data}(x)+p_{model}(x)}$$

Optimal discriminator $D^*(x)$

By calculus of variations, for any $\Delta(x)$,

$$\left.\frac{\partial J^{(D)}(D^*(X)+\lambda\Delta(x))}{\partial\lambda}\right|_{\lambda=0}=0$$

$$\Rightarrow -\left.\frac{\partial E_{x\sim p_{data}}\log(D^*(x)+\lambda\Delta(x))}{\partial\lambda}-\frac{\partial E_{x\sim p_{model}}\log(1-D^*(x)-\lambda\Delta(x))}{\partial\lambda}\right|_{\lambda=0}=0$$

$$\Rightarrow -E_{x\sim p_{data}}\left[\frac{1}{D^*(x)+\lambda\Delta(x)}\right]+E_{x\sim p_{model}}\left.\left[\frac{1}{1-D^*(x)-\lambda\Delta(x)}\right]\right|_{\lambda=0}=0$$

$$\Rightarrow \int_x\left[\frac{p_{data}(x)}{D^*(x)}-\frac{p_{model}(x)}{1-D^*(x)}\right]dx=0$$

$$\Rightarrow D^*(x)=\frac{p_{data}(x)}{p_{data}(x)+p_{model}(x)}$$

Optimal discriminator $D^*(x)$

By calculus of variations, for any $\Delta(x)$,

$$\left.\frac{\partial J^{(D)}(D^*(X)+\lambda\Delta(x))}{\partial\lambda}\right|_{\lambda=0}=0$$

$$\Rightarrow -\frac{\partial E_{x\sim p_{data}}\log(D^*(x)+\lambda\Delta(x))}{\partial\lambda}-\left.\frac{\partial E_{x\sim p_{model}}\log(1-D^*(x)-\lambda\Delta(x))}{\partial\lambda}\right|_{\lambda=0}=0$$

$$\Rightarrow -E_{x\sim p_{data}}\left[\frac{1}{D^*(x)+\lambda\Delta(x)}\right]+E_{x\sim p_{model}}\left.\left[\frac{1}{1-D^*(x)-\lambda\Delta(x)}\right]\right|_{\lambda=0}=0$$

$$\Rightarrow \int_x\left[\frac{p_{data}(x)}{D^*(x)}-\frac{p_{model}(x)}{1-D^*(x)}\right]dx=0$$

$$\Rightarrow D^*(x)=\frac{p_{data}(x)}{p_{data}(x)+p_{model}(x)}$$

Optimal discriminator $D^*(x)$

By calculus of variations, for any $\Delta(x)$,

$$\left.\frac{\partial J^{(D)}(D^*(X)+\lambda\Delta(x))}{\partial\lambda}\right|_{\lambda=0}=0$$

$$\Rightarrow -\left.\frac{\partial E_{x\sim p_{data}}\log(D^*(x)+\lambda\Delta(x))}{\partial\lambda}-\frac{\partial E_{x\sim p_{model}}\log(1-D^*(x)-\lambda\Delta(x))}{\partial\lambda}\right|_{\lambda=0}=0$$

$$\Rightarrow -E_{x\sim p_{data}}\left[\frac{1}{D^*(x)+\lambda\Delta(x)}\right]+E_{x\sim p_{model}}\left.\left[\frac{1}{1-D^*(x)-\lambda\Delta(x)}\right]\right|_{\lambda=0}=0$$

$$\Rightarrow \int_x\left[\frac{p_{data}(x)}{D^*(x)}-\frac{p_{model}(x)}{1-D^*(x)}\right]dx=0$$

$$\Rightarrow D^*(x)=\frac{p_{data}(x)}{p_{data}(x)+p_{model}(x)}$$

Optimal discriminator $D^*(x)$

By calculus of variations, for any $\Delta(x)$,

$$\left.\frac{\partial J^{(D)}(D^*(X)+\lambda\Delta(x))}{\partial\lambda}\right|_{\lambda=0}=0$$

$$\Rightarrow -\left.\frac{\partial E_{x\sim p_{data}}\log(D^*(x)+\lambda\Delta(x))}{\partial\lambda}-\frac{\partial E_{x\sim p_{model}}\log(1-D^*(x)-\lambda\Delta(x))}{\partial\lambda}\right|_{\lambda=0}=0$$

$$\Rightarrow -E_{x\sim p_{data}}\left[\frac{1}{D^*(x)+\lambda\Delta(x)}\right]+\left.E_{x\sim p_{model}}\left[\frac{1}{1-D^*(x)-\lambda\Delta(x)}\right]\right|_{\lambda=0}=0$$

$$\Rightarrow \int_x\left[\frac{p_{data}(x)}{D^*(x)}-\frac{p_{model}(x)}{1-D^*(x)}\right]dx=0$$

$$\Rightarrow D^*(x)=\frac{p_{data}(x)}{p_{data}(x)+p_{model}(x)}$$

Non-saturating cost function

- The discriminator cost function $J^{(D)} = -E_{x \sim p_{data}} \log D(x) - E_z \log(1 - D(G(z)))$ is a very reasonable choice and usually will not be modified
- On the other hand, we have more freedom on choosing the generator cost
    - $E_z \log(1 - D(G(z)))$ is the intuitive choice for $J^{(G)}$ but it has a small gradient when $D(G(z))$ is small for all z
        - That is, generator is not able to fool the discriminator
        - Reasonable when we just started to train the generator
    - Instead, it is better to have $J^{(G)} = -E_z \log D(G(z))$
        - $-\log D(G(z)) \approx 0$ when $D(G(z)) \approx 1$: ignore samples that successfully fool the discriminator
        - $-\log D(G(z)) \gg 0$ when $D(G(z)) \approx 0$: emphasize samples that cannot fool the discriminator
        - When $D(G(z)) \approx 1$ for all z, we may need to switch back to the original cost function. But better yet, we should better train the discriminator

Non-saturating cost function

- The discriminator cost function $J^{(D)} = -E_{x \sim p_{data}} \log D(x) - E_z \log(1 - D(G(z)))$ is a very reasonable choice and usually will not be modified
- On the other hand, we have more freedom on choosing the generator cost
  - $E_z \log(1 - D(G(z)))$ is the intuitive choice for $J^{(G)}$ but it has a small gradient when $D(G(z))$ is small for all z
    - That is, generator is not able to fool the discriminator
    - Reasonable when we just started to train the generator
  - Instead, it is better to have $J^{(G)} = -E_z \log D(G(z))$
    - $-\log D(G(z)) \approx 0$ when $D(G(z)) \approx 1$: ignore samples that successfully fool the discriminator
    - $-\log D(G(z)) \gg 0$ when $D(G(z)) \approx 0$: emphasize samples that cannot fool the discriminator
    - When $D(G(z)) \approx 1$ for all z, we may need to switch back to the original cost function. But better yet, we should better train the discriminator

Non-saturating cost function

- The discriminator cost function $J^{(D)} = -E_{x\sim p_{data}} \log D(x) - E_z \log(1 - D(G(z)))$ is a very reasonable choice and usually will not be modified
- On the other hand, we have more freedom on choosing the generator cost
  - $E_z \log(1 - D(G(z)))$ is the intuitive choice for $J^{(G)}$ but it has a small gradient when $D(G(z))$ is small for all z
    - That is, generator is not able to fool the discriminator
    - Reasonable when we just started to train the generator
  - Instead, it is better to have $J^{(G)} = -E_z \log D(G(z))$
    - $-\log D(G(z)) \approx 0$ when $D(G(z)) \approx 1$: ignore samples that successfully fool the discriminator
    - $-\log D(G(z)) \gg 0$ when $D(G(z)) \approx 0$: emphasize samples that cannot fool the discriminator
    - When $D(G(z)) \approx 1$ for all z, we may need to switch back to the original cost function. But better yet, we should better train the discriminator

Non-saturating cost function

- The discriminator cost function $J^{(D)} = -E_{x\sim p_{data}} \log D(x) - E_z \log(1 - D(G(z)))$ is a very reasonable choice and usually will not be modified
- On the other hand, we have more freedom on choosing the generator cost
  - $E_z \log(1 - D(G(z)))$ is the intuitive choice for $J^{(G)}$ but it has a small gradient when $D(G(z))$ is small for all z
    - That is, generator is not able to fool the discriminator
    - Reasonable when we just started to train the generator
  - Instead, it is better to have $J^{(G)} = -E_z \log D(G(z))$
    - $-\log D(G(z)) \approx 0$ when $D(G(z)) \approx 1$: ignore samples that successfully fool the discriminator
    - $-\log D(G(z)) \gg 0$ when $D(G(z)) \approx 0$: emphasize samples that cannot fool the discriminator
    - When $D(G(z)) \approx 1$ for all z, we may need to switch back to the original cost function. But better yet, we should better train the discriminator

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:

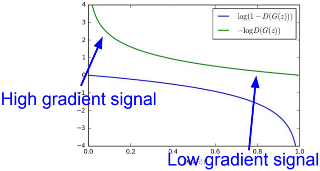
1. **Gradient ascent** on discriminator

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

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.



Some refinements

Training GAN is equivalent of finding the Nash equilibrium of a two-player non-cooperative game, which itself is a very hard problem. We will mention here a couple refinements to help find a better solution. You probably would like to check out Salimans' 16 also

- One-sided label smoothing
- Fixing batch-norm
- Mini-batch features
- Unrolled GAN

One-sided label smoothing

Salimans *et al.* 2016

- Default discriminator cost can also be written as

$$\text{cross\_entropy}(\text{“1”}, \text{discriminator}(\text{data})) + \text{cross\_entropy}(\text{“0”}, \text{discriminator}(\text{samples}))$$

- Experiment shows that one-sided label smoothed cost enhance system stability

$$\text{cross\_entropy}(\text{“0.9”}, \text{discriminator}(\text{data})) + \text{cross\_entropy}(\text{“0”}, \text{discriminator}(\text{samples}))$$

  - Essentially prevent extrapolating effect from extreme samples
  - Generally does not reduce classification accuracy, only confidence

One-sided label smoothing
Salimans *et al.* 2016

- Default discriminator cost can also be written as

$$\text{cross\_entropy}(\text{``1''}, \text{discriminator(data)}) \\ +\text{cross\_entropy}(\text{``0''}, \text{discriminator(samples)})$$

- Experiment shows that one-sided label smoothed cost enhance system stability

$$\text{cross\_entropy}(\text{``0.9''}, \text{discriminator(data)}) \\ +\text{cross\_entropy}(\text{``0''}, \text{discriminator(samples)})$$

  - Essentially prevent extrapolating effect from extreme samples
  - Generally does not reduce classification accuracy, only confidence

One-sided label smoothing
Salimans *et al.* 2016

- Default discriminator cost can also be written as

$$\begin{aligned}&\text{cross\_entropy}(\text{“1”},\text{discriminator(data)})\\ +&\text{cross\_entropy}(\text{“0”},\text{discriminator(samples)})\end{aligned}$$

- Experiment shows that one-sided label smoothed cost enhance system stability

$$\begin{aligned}&\text{cross\_entropy}(\text{“0.9”},\text{discriminator(data)})\\ +&\text{cross\_entropy}(\text{“0”},\text{discriminator(samples)})\end{aligned}$$

  - Essentially prevent extrapolating effect from extreme samples
  - Generally does not reduce classification accuracy, only confidence

One-sided label smoothing
Salimans *et al.* 2016

- It is important not to smooth the negative labels though, i.e., say

$$\begin{aligned} &\text{cross\_entropy}(1-\alpha, \text{discriminator(data)}) \\ +&\text{cross\_entropy}(\beta, \text{discriminator(samples)}) \end{aligned}$$

with $\beta > 0$

- Just follow the same derivation as before, we can get the optimum $D(x)$ as

$$D^*(x) = \frac{(1-\alpha)p_{data}(x) + \beta p_{model}(x)}{p_{data}(x) + p_{model}(x)}$$

- $\beta > 0$ tends to give undesirable bias of the discriminator to data generated by the model

One-sided label smoothing
Salimans *et al.* 2016

- It is important not to smooth the negative labels though, i.e., say

$$\text{cross\_entropy}(1-\alpha,\text{discriminator(data)}) \\ +\text{cross\_entropy}(\beta,\text{discriminator(samples)})$$

  with $\beta > 0$

- Just follow the same derivation as before, we can get the optimum $D(x)$ as

$$D^*(x) = \frac{(1-\alpha)p_{data}(x) + \beta p_{model}(x)}{p_{data}(x) + p_{model}(x)}$$

- $\beta > 0$ tends to give undesirable bias of the discriminator to data generated by the model

One-sided label smoothing
Salimans *et al.* 2016

- It is important not to smooth the negative labels though, i.e., say

$$\text{cross\_entropy}(1-\alpha, \text{discriminator}(\text{data})) \\ +\text{cross\_entropy}(\beta, \text{discriminator}(\text{samples}))$$

  with $\beta > 0$

- Just follow the same derivation as before, we can get the optimum $D(x)$ as

$$D^*(x) = \frac{(1-\alpha)p_{data}(x) + \beta p_{model}(x)}{p_{data}(x) + p_{model}(x)}$$

- $\beta > 0$ tends to give undesirable bias of the discriminator to data generated by the model

Issue on batch normalization
Goodfellow 2016

Batch normalization is preferred and highly recommended. But it can cause strong intra-batch correlation

Fixing batch norm

- Reference batch norm: one possible approach is keep one reference batch and always normalized based on that batch. That is, always subtract mean from that of the reference batch and adjust variance to that of the reference batch
  - Can easily overfit to the particular reference batch
- Virtual batch norm: a partial solution by combining the reference batch norm and conventional batch norm. Fix a reference batch, but every time inputs are normalize to the net mean and variance of the virtual batch containing both inputs and all elements of the reference batch

Fixing batch norm

- Reference batch norm: one possible approach is keep one reference batch and always normalized based on that batch. That is, always subtract mean from that of the reference batch and adjust variance to that of the reference batch
  - Can easily overfit to the particular reference batch
- Virtual batch norm: a partial solution by combining the reference batch norm and conventional batch norm. Fix a reference batch, but every time inputs are normalize to the net mean and variance of the virtual batch containing both inputs and all elements of the reference batch

Balancing G and D

- Usually it is more preferable to have a bigger and deeper D
- Some researchers also run more D steps than G steps. The results are mixed though
- Some take home messages
  - Use non-saturating cost
  - Use label smoothing
- Do not try to limit D from being "too smart"
  - The original theoretical justification is that D is supposed to be perfect

Balancing G and D

- Usually it is more preferable to have a bigger and deeper D
- Some researchers also run more D steps than G steps. The results are mixed though
- Some take home messages
  - Use non-saturating cost
  - Use label smoothing
- Do not try to limit D from being "too smart"
  - The original theoretical justification is that D is supposed to be perfect

Balancing G and D

- Usually it is more preferable to have a bigger and deeper D
- Some researchers also run more D steps than G steps. The results are mixed though
- Some take home messages
  - Use non-saturating cost
  - Use label smoothing
- Do not try to limit D from being "too smart"
  - The original theoretical justification is that D is supposed to be perfect

Balancing G and D

- Usually it is more preferable to have a bigger and deeper D
- Some researchers also run more D steps than G steps. The results are mixed though
- Some take home messages
  - Use non-saturating cost
  - Use label smoothing
- Do not try to limit D from being “too smart”
  - The original theoretical justification is that D is supposed to be perfect

Mode collapse
Metz *et al.* 2016

Below demonstrates why D should be smart.

- Basically the minmax and the minmax problem is not the same and can lead to drastically different solutions

$$\min_G \max_D V(G, D) \neq \max_D \min_G V(G, D)$$

- D in the inner loop: converge to the correct distribution
- G in the inner loop: place all mass on most likely point

Mode collapse

Metz *et al.* 2016

Below demonstrates why D should be smart.

- Basically the minmax and the minmax problem is not the same and can lead to drastically different solutions

$$\min_G \max_D V(G, D) \neq \max_D \min_G V(G, D)$$

- D in the inner loop: converge to the correct distribution
- G in the inner loop: place all mass on most likely point

Minibatch features
Salimans *et al.* 2016

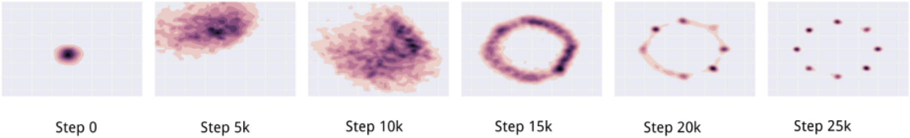
- Mode collapse can lead to low diversity of generated data
- One attempt to mitigate this problem is to introduce the so-called minibatch features
  - Basically classify each example by comparing the features to other members in the minibatch
  - Reject a sample if the feature to close to existing ones

Minibatch features
Salimans *et al.* 2016

- Mode collapse can lead to low diversity of generated data
- One attempt to mitigate this problem is to introduce the so-called minibatch features
  - Basically classify each example by comparing the features to other members in the minibatch
  - Reject a sample if the feature to close to existing ones

Unrolled Gans
Metz *et al.* 2016

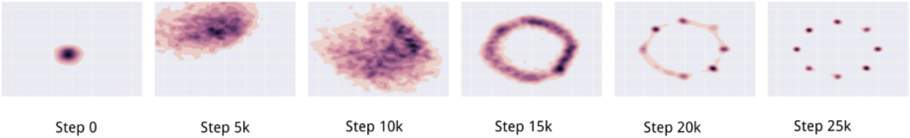
- A more direct approach was proposed by Google brain
- Trying to ensure that the generated sample is a solution of the minmax rather than the maxmin problem
- Have the generator to unroll k future steps and predict what discriminator will think of the current sample
  - Since generator is the one who unrolls, generator is in the outer loop and discriminator is in the inner loop
  - We ensure that we have solution approximating a minmax rather than maxmin problem



Unrolled Gans

Metz *et al.* 2016

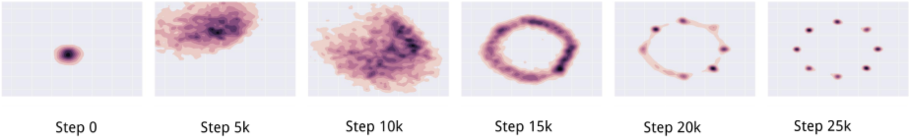
- A more direct approach was proposed by Google brain
- Trying to ensure that the generated sample is a solution of the minmax rather than the maxmin problem
- Have the generator to unroll k future steps and predict what discriminator will think of the current sample
  - Since generator is the one who unrolls, generator is in the outer loop and discriminator is in the inner loop
  - We ensure that we have solution approximating a minmax rather than maxmin problem



Unrolled Gans

Metz *et al.* 2016

- A more direct approach was proposed by Google brain
- Trying to ensure that the generated sample is a solution of the minmax rather than the maxmin problem
- Have the generator to unroll k future steps and predict what discriminator will think of the current sample
  - Since generator is the one who unrolls, generator is in the outer loop and discriminator is in the inner loop
  - We ensure that we have solution approximating a minmax rather than maxmin problem



Deep convolutional GAN (DCGAN)

Generative Adversarial Nets: Convolutional Architectures

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

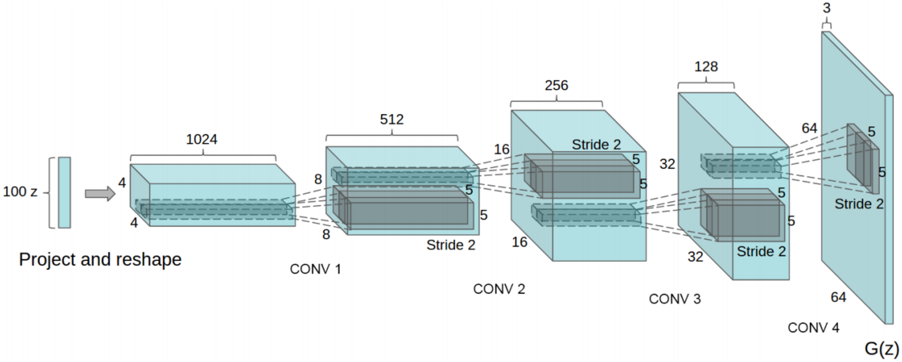
Architecture guidelines for stable Deep Convolutional GANs

- 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

Deep convolutional GAN (DCGAN)

Radford *et al.* 2016

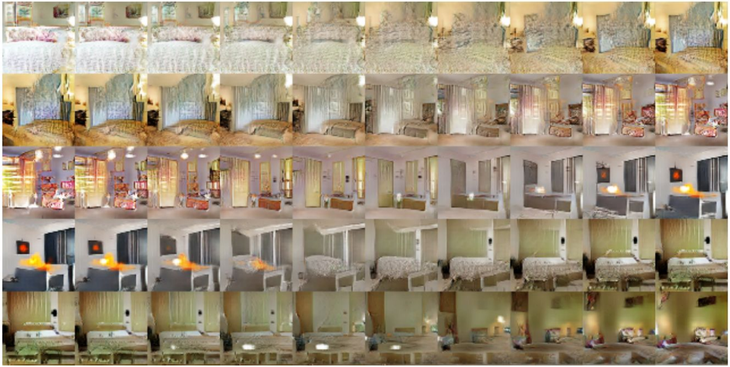


Generated bedroom after 5 epochs (LSUN dataset)

Radford *et al.* 2016

Generative Adversarial Nets: Convolutional Architectures

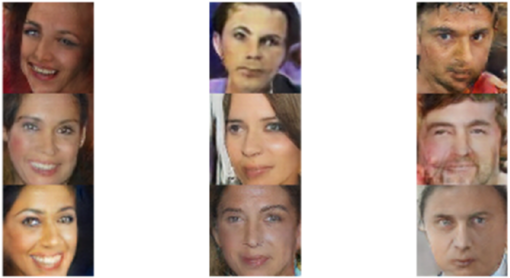
Interpolating between random points in laten space



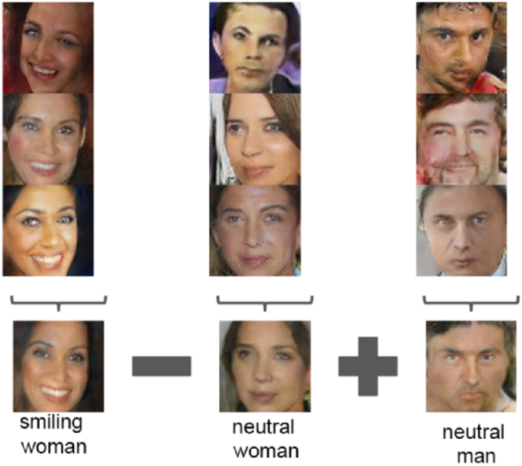
Radford et al, ICLR 2016

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 13 - 121 May 18, 2017

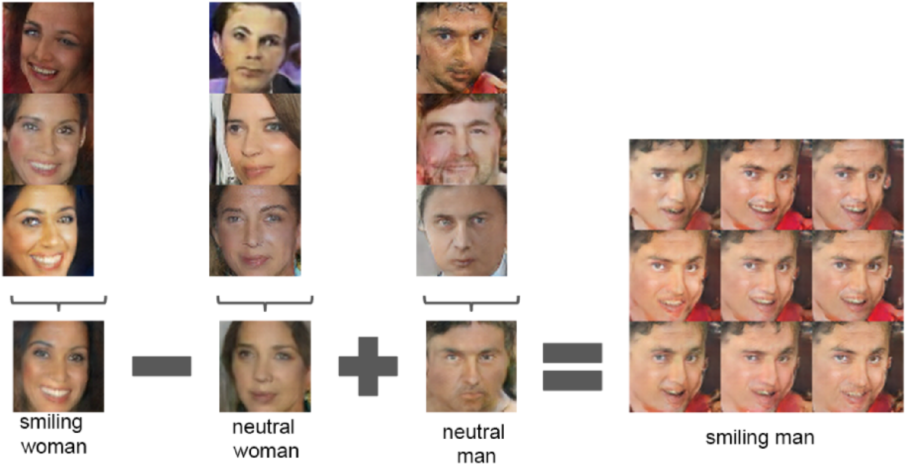
Vector arithmetics
Radford *et al.* 2016



Vector arithmetics
Radford *et al.* 2016

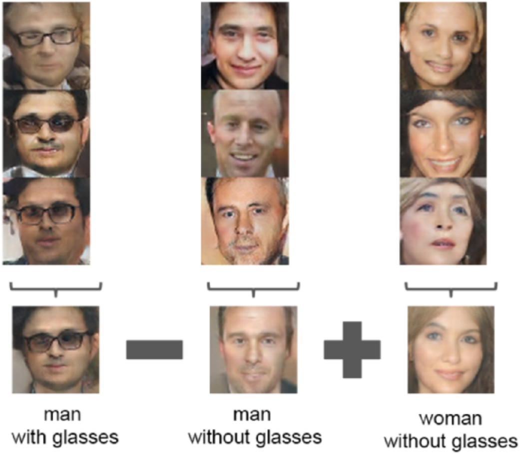


Vector arithmetics
Radford *et al.* 2016

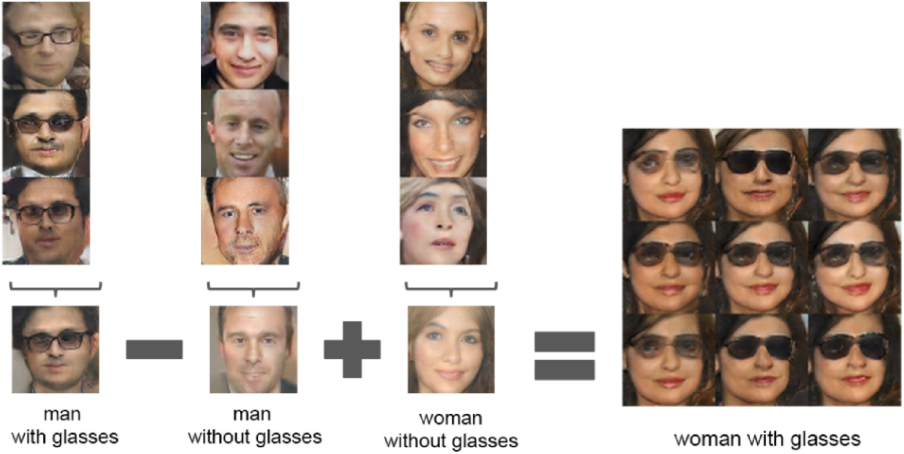


Vector arithmetics
Radford *et al.* 2016

Vector arithmetics
Radford *et al.* 2016

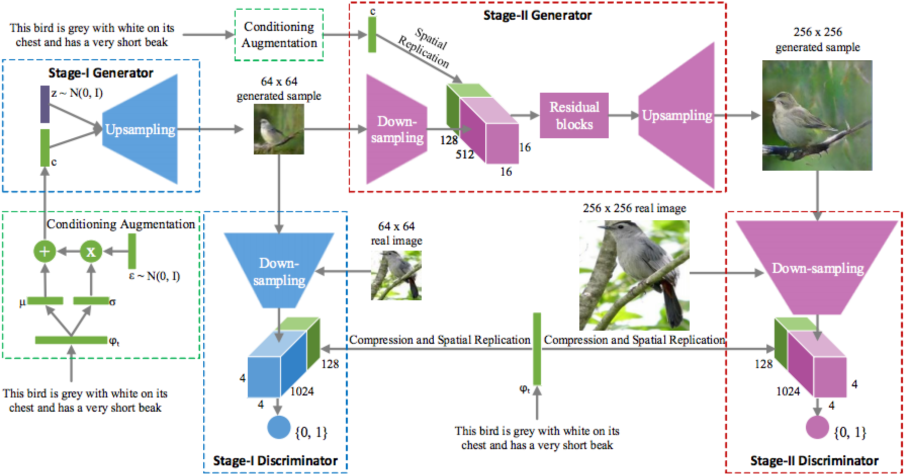


Vector arithmetics
Radford *et al.* 2016

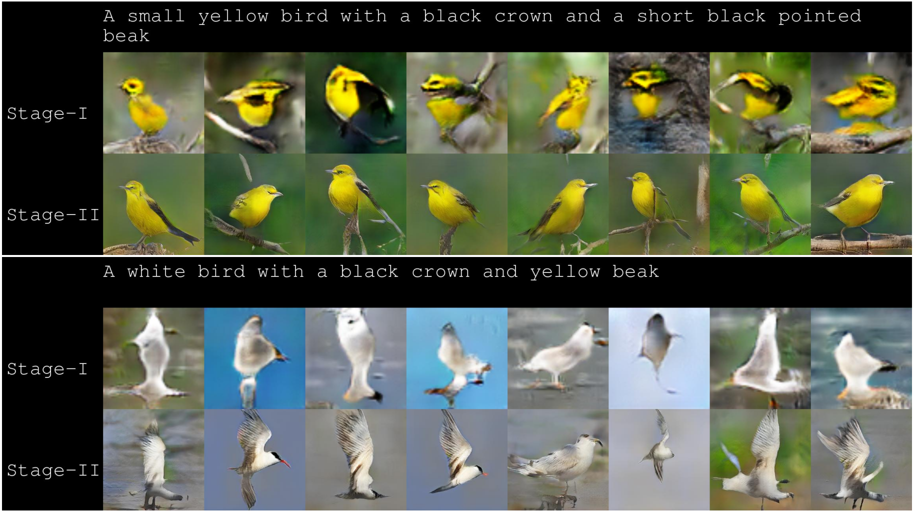


StackGAN

Zhang et al. 2016



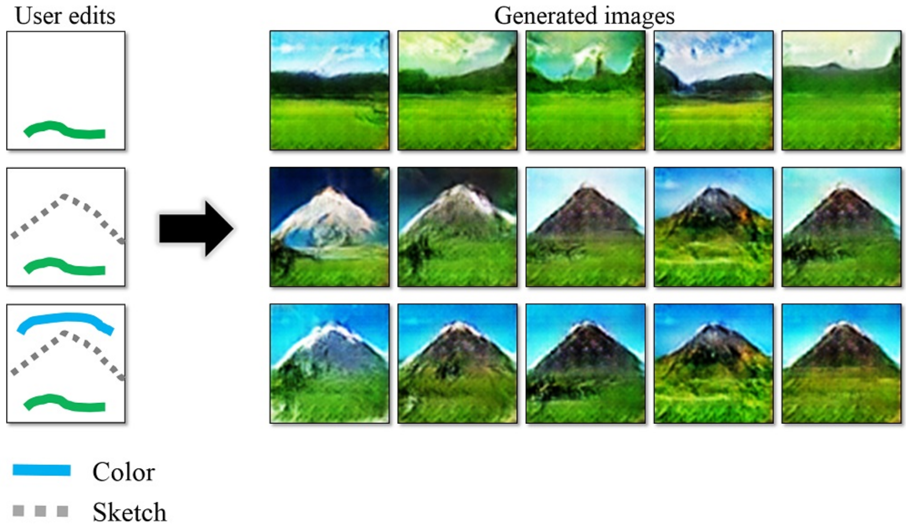
StackGAN


A small yellow bird with a black crown and a short black pointed beak
Stage-I
Stage-II
A white bird with a black crown and yellow beak
Stage-I
Stage-II

StackGAN

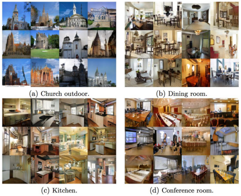

This flower has long thin yellow petals and a lot of yellow anthers in the center
Stage-I
Stage-II
This flower is white, pink, and yellow in color, and has petals that are multi colored
Stage-I
Stage-II

iGAN
Zhu *et al.* 2016

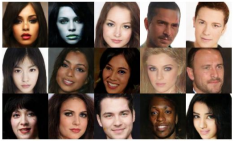


2017: Year of the GAN

Better training and generation

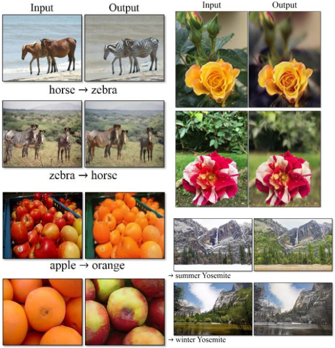


LSGAN. Mao et al. 2017.



BEGAN. Bertholet et al. 2017.

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/

Fei-Fei Li & Justin Johnson & Serena Yeung | Lecture 13 - 127 | May 18, 2017

"The GAN Zoo"

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

- 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

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 13 - 129 May 18, 2017

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)

Active areas of research:

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

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 13 - 130 May 18, 2017

Why autoencoders? Dimension reduction

- As name suggests, the objective of dimension of reduction is to decrease the dimension of input signals to ease later processing
  - It is often a preprocessing step
  - Was commonly used to compress features
- It is a very old problem. The most representative algorithm is the principal component analysis (PCA)

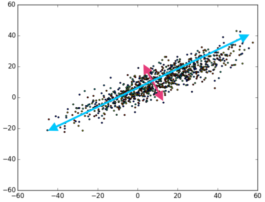
Why autoencoders? Dimension reduction

- As name suggests, the objective of dimension of reduction is to decrease the dimension of input signals to ease later processing
  - It is often a preprocessing step
  - Was commonly used to compress features
- It is a very old problem. The most representative algorithm is the principal component analysis (PCA)

Why autoencoders? Dimension reduction

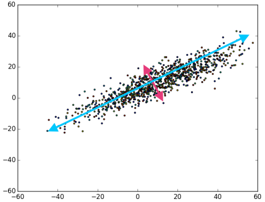
- As name suggests, the objective of dimension of reduction is to decrease the dimension of input signals to ease later processing
  - It is often a preprocessing step
  - Was commonly used to compress features
- It is a very old problem. The most representative algorithm is the principal component analysis (PCA)

Principal component analysis (PCA)



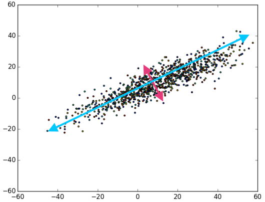
- Take *N*-dimensional data and find the *M* orthogonal directions in which the data have the most variance
  - We can represent an *N*-dimensional datapoint by its projections onto the *M* principal directions (i.e., with highest variances)
  - This loses all information about where the datapoint is located in the remaining orthogonal directions

Principal component analysis (PCA)



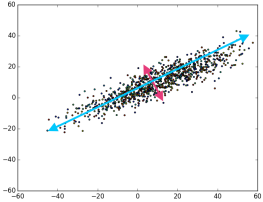
- Take N-dimensional data and find the M orthogonal directions in which the data have the most variance
  - We can represent an N-dimensional datapoint by its projections onto the M principal directions (i.e., with highest variances)
  - This loses all information about where the datapoint is located in the remaining orthogonal directions

Principal component analysis (PCA)



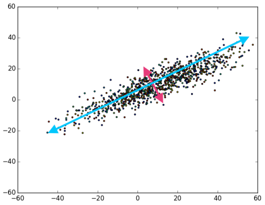
- Take N-dimensional data and find the M orthogonal directions in which the data have the most variance
  - We can represent an N-dimensional datapoint by its projections onto the M principal directions (i.e., with highest variances)
  - This loses all information about where the datapoint is located in the remaining orthogonal directions

PCA reconstruction



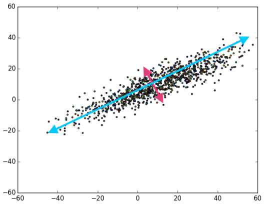
- We reconstruct by using the mean value (over all the data) on the $N - M$ directions that are not represented.
  - The reconstruction error is the sum over the variances over all these unrepresented directions
    - The variances are just eigenvalues of covariance matrix of the data
- PCA is "optimum"
  - Since we keep the largest variance components, on average the distortion is minimum among all linear dimension reduction methods

PCA reconstruction



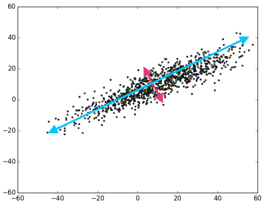
- We reconstruct by using the mean value (over all the data) on the $N - M$ directions that are not represented.
  - The reconstruction error is the sum over the variances over all these unrepresented directions
    - The variances are just eigenvalues of covariance matrix of the data
- PCA is "optimum"
  - Since we keep the largest variance components, on average the distortion is minimum among all linear dimension reduction methods

PCA reconstruction



- We reconstruct by using the mean value (over all the data) on the $N - M$ directions that are not represented.
  - The reconstruction error is the sum over the variances over all these unrepresented directions
    - The variances are just eigenvalues of covariance matrix of the data
- PCA is "optimum"
  - Since we keep the largest variance components, on average the distortion is minimum among all linear dimension reduction methods

PCA reconstruction



- We reconstruct by using the mean value (over all the data) on the $N - M$ directions that are not represented.
  - The reconstruction error is the sum over the variances over all these unrepresented directions
    - The variances are just eigenvalues of covariance matrix of the data
- PCA is "optimum"
  - Since we keep the largest variance components, on average the distortion is minimum among all linear dimension reduction methods

Math review: Singular value decomposition (SVD)

For any $N \times K$ matrix A (assume $K \leq N$), we can decompose it into product of three matrices

$$\begin{pmatrix} \\ \\ A \\ \\ \\ \end{pmatrix} = \begin{pmatrix} \\ \\ U \\ \\ \\ \end{pmatrix} \begin{pmatrix} \\ D \\ \\ \end{pmatrix} \begin{pmatrix} \\ V \\ \\ \end{pmatrix}^T,$$

where U is $N \times K$, D is $K \times K$, and V is $K \times K$. Moreover,

- U is orthonormal, i.e., $U^T U = I$
- D is diagonal
- V is orthonormal, i.e., $V^T V = I$

Has nice geometric interpretation. Roughly speaking, any linear transform can be decompose into rotation, scaling, and rotation again

Math review: Singular value decomposition (SVD)

For any $N \times K$ matrix A (assume $K \leq N$), we can decompose it into product of three matrices

$$\left(\quad A \quad \right) = \left(\quad U \quad \right) \left(\quad D \quad \right) \left(\quad V \quad \right)^T,$$

where U is $N \times K$, D is $K \times K$, and V is $K \times K$. Moreover,

- U is orthonormal, i.e., $U^T U = I$
- D is diagonal
- V is orthonormal, i.e., $V^T V = I$

Has nice geometric interpretation. Roughly speaking, any linear transform can be decompose into rotation, scaling, and rotation again

Math review: Singular value decomposition (SVD)

For any $N \times K$ matrix A (assume $K \leq N$), we can decompose it into product of three matrices

$$\left(\quad A \quad \right) = \left(\quad U \quad \right) \left(\quad D \quad \right) \left(\quad V \quad \right)^T,$$

where U is $N \times K$, D is $K \times K$, and V is $K \times K$. Moreover,

- U is orthonormal, i.e., $U^T U = I$
- D is diagonal
- V is orthonormal, i.e., $V^T V = I$

Has nice geometric interpretation. Roughly speaking, any linear transform can be decompose into rotation, scaling, and rotation again

SVD and PCA

- Let $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_K]$ be the matrix with columns as data vectors. We can decompose $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$ using SVD
- Assume X is zero-mean, the covariance matrix C is just $C \approx \frac{XX^T}{k}$
- Note that $C \sim U\Sigma V^T(U\Sigma V^T)^T = U\Sigma^2 U^T$, thus singular values are just square root of eigenvalues
  - Since PCA is in effect keeping the M largest eigenvalues of the covariance matrix, it is the same as keeping the M largest singular values of X
- One can easily verify that. Let $\hat{X} = U\hat{\Sigma}V^T$, where $\hat{\Sigma}$ only keeps the M largest singular values, then

$$\begin{aligned}
Error =& \sum_i (x-\hat{x})^T(x-\hat{x}) = tr((X-\hat{X})^T(X-\hat{X})) \\
=& tr(V(\Sigma-\hat{\Sigma})U^T U(\Sigma-\hat{\Sigma})V^T) = tr(V(\Sigma-\hat{\Sigma})(\Sigma-\hat{\Sigma})V^T) \\
=& tr(((\Sigma-\hat{\Sigma})V^T)V(\Sigma-\hat{\Sigma})) = tr((\Sigma-\hat{\Sigma})^2) \\
=& \text{Sum of eigenvalues excluding the } M \text{ largest ones}
\end{aligned}$$

SVD and PCA

- Let $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_K]$ be the matrix with columns as data vectors. We can decompose $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$ using SVD
- Assume X is zero-mean, the covariance matrix C is just $C \approx \frac{XX^T}{k}$
- Note that $C \sim U\Sigma V^T (U\Sigma V^T)^T = U\Sigma^2 U^T$, thus singular values are just square root of eigenvalues
    - Since PCA is in effect keeping the M largest eigenvalues of the covariance matrix, it is the same as keeping the M largest singular values of X
- One can easily verify that. Let $\hat{X} = U\hat{\Sigma}V^T$, where $\hat{\Sigma}$ only keeps the M largest singular values, then

$$
\begin{aligned}
Error =& \sum_i (x - \hat{x})^T (x - \hat{x}) = tr((X - \hat{X})^T (X - \hat{X})) \\
=& tr(V(\Sigma - \hat{\Sigma})U^T U(\Sigma - \hat{\Sigma})V^T) = tr(V(\Sigma - \hat{\Sigma})(\Sigma - \hat{\Sigma})V^T) \\
=& tr(((\Sigma - \hat{\Sigma})V^T)V(\Sigma - \hat{\Sigma})) = tr((\Sigma - \hat{\Sigma})^2) \\
=& \text{Sum of eigenvalues excluding the } M \text{ largest ones}
\end{aligned}
$$

SVD and PCA

- Let $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_K]$ be the matrix with columns as data vectors. We can decompose $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$ using SVD
- Assume X is zero-mean, the covariance matrix C is just $C \approx \frac{XX^T}{k}$
- Note that $C \sim U\Sigma V^T(U\Sigma V^T)^T = U\Sigma^2 U^T$, thus singular values are just square root of eigenvalues
  - Since PCA is in effect keeping the M largest eigenvalues of the covariance matrix, it is the same as keeping the M largest singular values of X
- One can easily verify that. Let $\hat{X} = U\hat{\Sigma}V^T$, where $\hat{\Sigma}$ only keeps the M largest singular values, then

$$
\begin{aligned}
Error = &\sum_i (x - \hat{x})^T (x - \hat{x}) = tr((X - \hat{X})^T (X - \hat{X})) \\
= &tr(V(\Sigma - \hat{\Sigma})U^T U(\Sigma - \hat{\Sigma})V^T) = tr(V(\Sigma - \hat{\Sigma})(\Sigma - \hat{\Sigma})V^T) \\
= &tr(((\Sigma - \hat{\Sigma})V^T)V(\Sigma - \hat{\Sigma})) = tr((\Sigma - \hat{\Sigma})^2) \\
= &\text{Sum of eigenvalues excluding the } M \text{ largest ones}
\end{aligned}
$$

SVD and PCA

- Let $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_K]$ be the matrix with columns as data vectors. We can decompose $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$ using SVD
- Assume X is zero-mean, the covariance matrix C is just $C \approx \frac{XX^T}{k}$
- Note that $C \sim U\Sigma V^T(U\Sigma V^T)^T = U\Sigma^2 U^T$, thus singular values are just square root of eigenvalues
  - Since PCA is in effect keeping the M largest eigenvalues of the covariance matrix, it is the same as keeping the M largest singular values of X
- One can easily verify that. Let $\hat{X} = U\hat{\Sigma}V^T$, where $\hat{\Sigma}$ only keeps the M largest singular values, then

$$
\begin{aligned}
Error = \sum_i (x - \hat{x})^T (x - \hat{x}) &= tr((X - \hat{X})^T (X - \hat{X})) \\
&= tr(V(\Sigma - \hat{\Sigma})U^T U(\Sigma - \hat{\Sigma})V^T) = tr(V(\Sigma - \hat{\Sigma})(\Sigma - \hat{\Sigma})V^T) \\
&= tr(((\Sigma - \hat{\Sigma})V^T)V(\Sigma - \hat{\Sigma})) = tr((\Sigma - \hat{\Sigma})^2) \\
&= \text{Sum of eigenvalues excluding the } M \text{ largest ones}
\end{aligned}
$$

SVD and PCA

- Let $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_K]$ be the matrix with columns as data vectors. We can decompose $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$ using SVD
- Assume X is zero-mean, the covariance matrix C is just $C \approx \frac{XX^T}{k}$
- Note that $C \sim U\Sigma V^T(U\Sigma V^T)^T = U\Sigma^2 U^T$, thus singular values are just square root of eigenvalues
    - Since PCA is in effect keeping the M largest eigenvalues of the covariance matrix, it is the same as keeping the M largest singular values of X
- One can easily verify that. Let $\hat{X} = U\hat{\Sigma}V^T$, where $\hat{\Sigma}$ only keeps the M largest singular values, then

$$
\begin{aligned}
Error = \sum_i (x-\hat{x})^T(x-\hat{x}) &= tr((X-\hat{X})^T(X-\hat{X})) \\
&= tr(V(\Sigma-\hat{\Sigma})U^T U(\Sigma-\hat{\Sigma})V^T) = tr(V(\Sigma-\hat{\Sigma})(\Sigma-\hat{\Sigma})V^T) \\
&= tr(((\Sigma-\hat{\Sigma})V^T)V(\Sigma-\hat{\Sigma})) = tr((\Sigma-\hat{\Sigma})^2) \\
&= \text{Sum of eigenvalues excluding the } M \text{ largest ones}
\end{aligned}
$$

SVD and PCA

- Let $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_K]$ be the matrix with columns as data vectors. We can decompose $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$ using SVD
- Assume X is zero-mean, the covariance matrix C is just $C \approx \frac{XX^T}{k}$
- Note that $C \sim U\Sigma V^T (U\Sigma V^T)^T = U\Sigma^2 U^T$, thus singular values are just square root of eigenvalues
    - Since PCA is in effect keeping the M largest eigenvalues of the covariance matrix, it is the same as keeping the M largest singular values of X
- One can easily verify that. Let $\hat{X} = U\hat{\Sigma}V^T$, where $\hat{\Sigma}$ only keeps the M largest singular values, then

$$Error = \sum_i (x - \hat{x})^T (x - \hat{x}) = tr((X - \hat{X})^T (X - \hat{X}))$$

$$= tr(V(\Sigma - \hat{\Sigma})U^T U(\Sigma - \hat{\Sigma})V^T) = tr(V(\Sigma - \hat{\Sigma})(\Sigma - \hat{\Sigma})V^T)$$

$$= tr(((\Sigma - \hat{\Sigma})V^T)V(\Sigma - \hat{\Sigma})) = tr((\Sigma - \hat{\Sigma})^2)$$

= Sum of eigenvalues excluding the M largest ones

SVD and PCA

- Let $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_K]$ be the matrix with columns as data vectors. We can decompose $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$ using SVD
- Assume X is zero-mean, the covariance matrix C is just $C \approx \frac{XX^T}{k}$
- Note that $C \sim U\Sigma V^T (U\Sigma V^T)^T = U\Sigma^2 U^T$, thus singular values are just square root of eigenvalues
  - Since PCA is in effect keeping the M largest eigenvalues of the covariance matrix, it is the same as keeping the M largest singular values of X
- One can easily verify that. Let $\hat{X} = U\hat{\Sigma}V^T$, where $\hat{\Sigma}$ only keeps the M largest singular values, then

$$
\begin{aligned}
Error = &\sum_i (x - \hat{x})^T (x - \hat{x}) = tr((X - \hat{X})^T (X - \hat{X})) \\
= &tr(V(\Sigma - \hat{\Sigma})U^T U(\Sigma - \hat{\Sigma})V^T) = tr(V(\Sigma - \hat{\Sigma})(\Sigma - \hat{\Sigma})V^T) \\
= &tr(((\Sigma - \hat{\Sigma})V^T)V(\Sigma - \hat{\Sigma})) = tr((\Sigma - \hat{\Sigma})^2) \\
= &\text{Sum of eigenvalues excluding the } M \text{ largest ones}
\end{aligned}
$$

SVD and PCA

- Let $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_K]$ be the matrix with columns as data vectors. We can decompose $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$ using SVD
- Assume X is zero-mean, the covariance matrix C is just $C \approx \frac{XX^T}{k}$
- Note that $C \sim U\Sigma V^T (U\Sigma V^T)^T = U\Sigma^2 U^T$, thus singular values are just square root of eigenvalues
  - Since PCA is in effect keeping the M largest eigenvalues of the covariance matrix, it is the same as keeping the M largest singular values of X
- One can easily verify that. Let $\hat{X} = U\hat{\Sigma}V^T$, where $\hat{\Sigma}$ only keeps the M largest singular values, then

$$Error = \sum_i (x - \hat{x})^T (x - \hat{x}) = tr((X - \hat{X})^T (X - \hat{X}))$$

$$= tr(V(\Sigma - \hat{\Sigma})U^T U(\Sigma - \hat{\Sigma})V^T) = tr(V(\Sigma - \hat{\Sigma})(\Sigma - \hat{\Sigma})V^T)$$

$$= tr(((\Sigma - \hat{\Sigma})V^T)V(\Sigma - \hat{\Sigma})) = tr((\Sigma - \hat{\Sigma})^2)$$

$=$Sum of eigenvalues excluding the M largest ones

SVD and PCA

- Let $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_K]$ be the matrix with columns as data vectors. We can decompose $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$ using SVD
- Assume X is zero-mean, the covariance matrix C is just $C \approx \frac{XX^T}{k}$
- Note that $C \sim U\Sigma V^T (U\Sigma V^T)^T = U\Sigma^2 U^T$, thus singular values are just square root of eigenvalues
  - Since PCA is in effect keeping the M largest eigenvalues of the covariance matrix, it is the same as keeping the M largest singular values of X
- One can easily verify that. Let $\hat{X} = U\hat{\Sigma}V^T$, where $\hat{\Sigma}$ only keeps the M largest singular values, then

$$
\begin{aligned}
Error = &\sum_i (x - \hat{x})^T (x - \hat{x}) = tr((X - \hat{X})^T (X - \hat{X})) \\
= &tr(V(\Sigma - \hat{\Sigma})U^T U(\Sigma - \hat{\Sigma})V^T) = tr(V(\Sigma - \hat{\Sigma})(\Sigma - \hat{\Sigma})V^T) \\
= &tr(((\Sigma - \hat{\Sigma})V^T) V(\Sigma - \hat{\Sigma})) = tr((\Sigma - \hat{\Sigma})^2) \\
= &\text{Sum of eigenvalues excluding the } M \text{ largest ones}
\end{aligned}
$$

SVD and PCA

- Let $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_K]$ be the matrix with columns as data vectors. We can decompose $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$ using SVD
- Assume X is zero-mean, the covariance matrix C is just $C \approx \frac{XX^T}{k}$
- Note that $C \sim U\Sigma V^T(U\Sigma V^T)^T = U\Sigma^2 U^T$, thus singular values are just square root of eigenvalues
  - Since PCA is in effect keeping the M largest eigenvalues of the covariance matrix, it is the same as keeping the M largest singular values of X
- One can easily verify that. Let $\hat{X} = U\hat{\Sigma}V^T$, where $\hat{\Sigma}$ only keeps the M largest singular values, then

$$
\begin{aligned}
Error =& \sum_i (x - \hat{x})^T(x - \hat{x}) = tr((X - \hat{X})^T(X - \hat{X})) \\
=& tr(V(\Sigma - \hat{\Sigma})U^T U(\Sigma - \hat{\Sigma})V^T) = tr(V(\Sigma - \hat{\Sigma})(\Sigma - \hat{\Sigma})V^T) \\
=& tr(((\Sigma - \hat{\Sigma})V^T)V(\Sigma - \hat{\Sigma})) = tr((\Sigma - \hat{\Sigma})^2) \\
=& \text{Sum of eigenvalues excluding the } M \text{ largest ones}
\end{aligned}
$$

Optimal linear decoder ⇒ optimal linear encoder

- PCA is optimum when things are "linear"
- Interesting to know that as far as decoding is linear, the optimal encoding is linear (PCA) as well
  - That is, if $\hat{\mathbf{X}} = \mathbf{W}h(\mathbf{X})$ for some optimal $\mathbf{W}$
  - $\Rightarrow h(\mathbf{X}) = \mathbf{T}\mathbf{X}$ for some optimal $\mathbf{T}$

Optimal linear decoder $\Rightarrow$ optimal linear encoder

- PCA is optimum when things are "linear"
- Interesting to know that as far as decoding is linear, the optimal encoding is linear (PCA) as well
  - That is, if $\hat{\mathbf{X}} = \mathbf{W}h(\mathbf{X})$ for some optimal $\mathbf{W}$
  - $\Rightarrow h(\mathbf{X}) = \mathbf{T}\mathbf{X}$ for some optimal $\mathbf{T}$

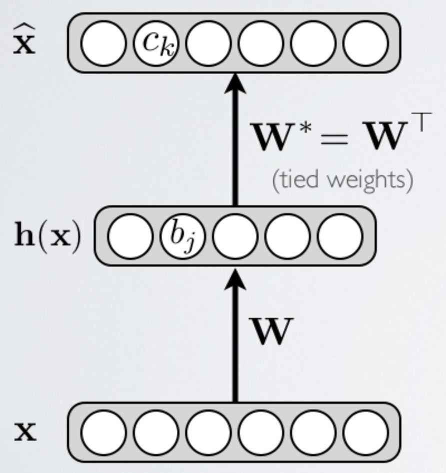
Optimal linear decoder $\Rightarrow$ optimal linear encoder

- PCA is optimum when things are "linear"
- Interesting to know that as far as decoding is linear, the optimal encoding is linear (PCA) as well
  - That is, if $\hat{\mathbf{X}} = \mathbf{W}h(\mathbf{X})$ for some optimal $\mathbf{W}$
  - $\Rightarrow h(\mathbf{X}) = \mathbf{T}\mathbf{X}$ for some optimal $\mathbf{T}$

Optimal linear decoder $\Rightarrow$ optimal linear encoder

- PCA is optimum when things are "linear"
- Interesting to know that as far as decoding is linear, the optimal encoding is linear (PCA) as well
  - That is, if $\hat{\mathbf{X}} = \mathbf{W}h(\mathbf{X})$ for some optimal $\mathbf{W}$
  - $\Rightarrow h(\mathbf{X}) = \mathbf{TX}$ for some optimal $\mathbf{T}$

Autoencoders

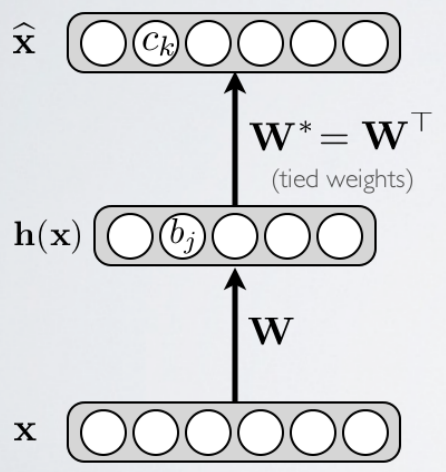


- Autoencoder is a way to perform dimension reduction with neural networks

$$\mathbf{h}(\mathbf{x}) = \text{sigm}(\mathbf{b} + \mathbf{W}\mathbf{x})$$
$$\hat{\mathbf{x}} = \mathbf{c} + \mathbf{W}^*\mathbf{h}(\mathbf{x})$$

- loss = $\|\mathbf{x} - \hat{\mathbf{x}}\|$
- N.B., as the decoder is linear, the optimum autoencoder is just equivalent to PCA

Autoencoders

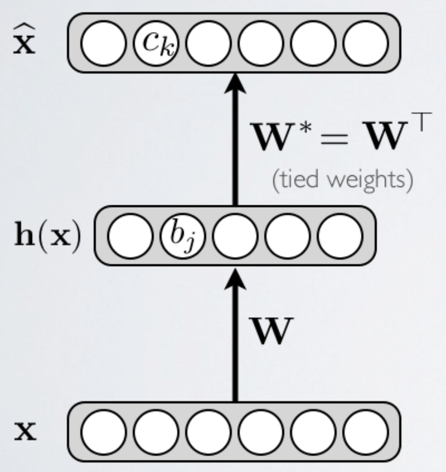


- Autoencoder is a way to perform dimension reduction with neural networks

$$\mathbf{h}(\mathbf{x}) = \text{sigm}(\mathbf{b} + \mathbf{W}\mathbf{x})$$
$$\hat{\mathbf{x}} = \mathbf{c} + \mathbf{W}^*\mathbf{h}(\mathbf{x})$$

- loss = $\|\mathbf{x} - \hat{\mathbf{x}}\|$
- N.B., as the decoder is linear, the optimum autoencoder is just equivalent to PCA

Autoencoders



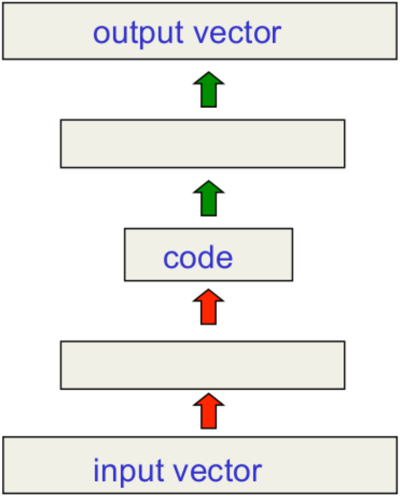
- Autoencoder is a way to perform dimension reduction with neural networks

$$\mathbf{h}(\mathbf{x}) = \mathrm{sigm}(\mathbf{b} + \mathbf{W}\mathbf{x})$$
$$\hat{\mathbf{x}} = \mathbf{c} + \mathbf{W}^*\mathbf{h}(\mathbf{x})$$

- loss = $\|\mathbf{x} - \hat{\mathbf{x}}\|$
- N.B., as the decoder is linear, the optimum autoencoder is just equivalent to PCA

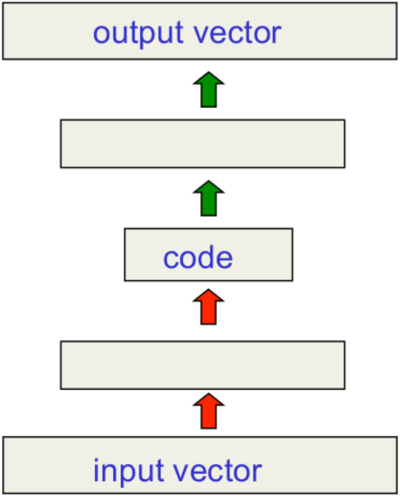
Deep autoencoders

Hinton & Salakhutdinov, Science 2006



- When using multiple layers, PCA is no longer optimal for continuous input
- The introduced nonlinearity can efficiently represent data that lies on a non-linear manifold
- It was an old idea (dated back to 80's) but it was considered to be very hard to train

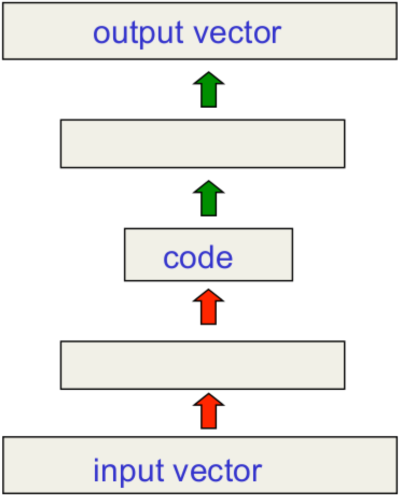
Deep autoencoders
Hinton & Salakhutdinov, Science 2006



- When using multiple layers, PCA is no longer optimal for continuous input
- The introduced nonlinearity can efficiently represent data that lies on a non-linear manifold
- It was an old idea (dated back to 80's) but it was considered to be very hard to train

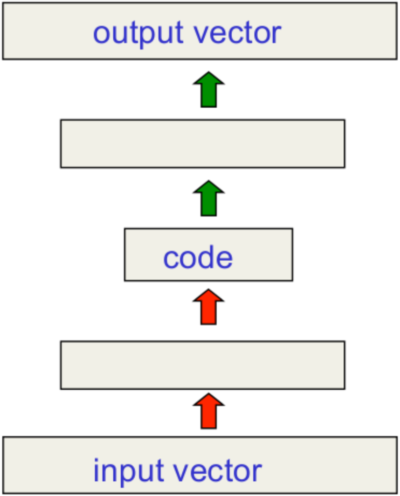
Deep autoencoders

Hinton & Salakhutdinov, Science 2006



- When using multiple layers, PCA is no longer optimal for continuous input
- The introduced nonlinearity can efficiently represent data that lies on a non-linear manifold
- It was an old idea (dated back to 80's) but it was considered to be very hard to train

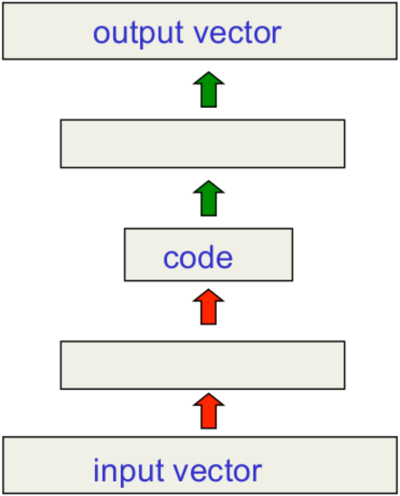
Deep autoencoders
Hinton & Salakhutdinov, Science 2006



- First really successful deep autoencoder was trained in 2006 by Hinton's group
- It uses layer-by-layer RBM pre-training as described in the last lecture
- Just use regular backprob for fine-tuning

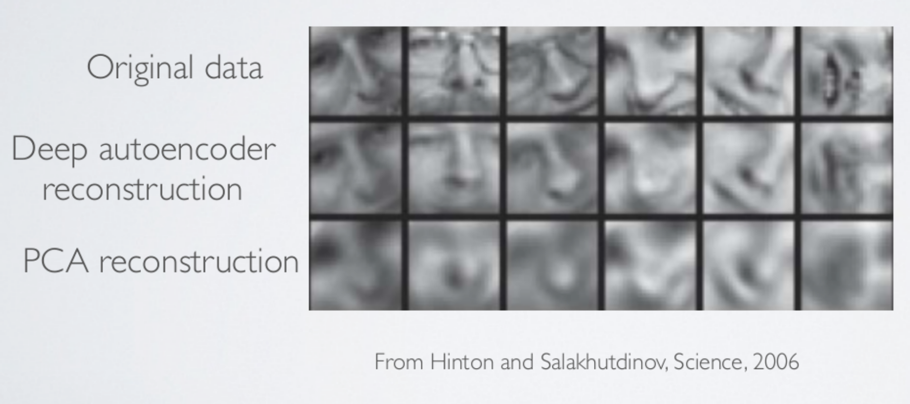
Deep autoencoders

Hinton & Salakhutdinov, Science 2006



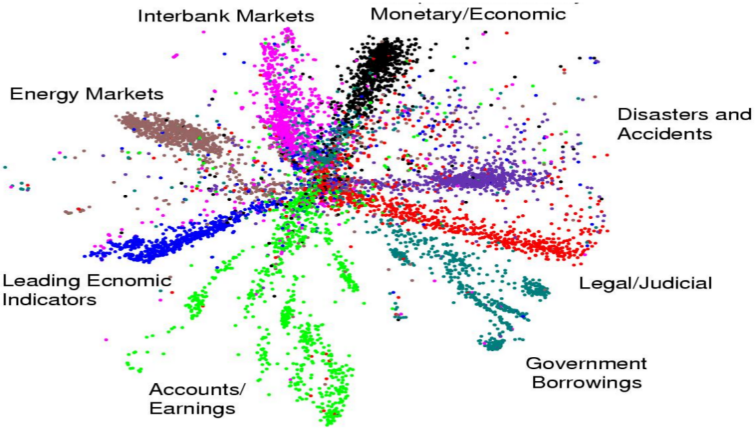
- First really successful deep autoencoder was trained in 2006 by Hinton's group
- It uses layer-by-layer RBM pre-training as described in the last lecture
- Just use regular backprob for fine-tuning

Deep autoencoder vs PCA



Deep autoencoder for 400,000 business documents
Hinton 2006

First compress all documents to 2 numbers using deep auto.
Then use different colors for different document categories



Deep autoencoder for 400,000 image retrieval

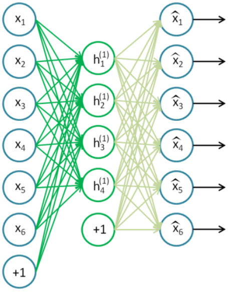
Hinton 2006

Leftmost column is the search image.

Other columns are the images that have the most similar feature activities in the last hidden layer.

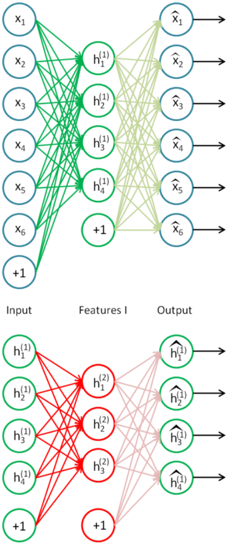
Stacked autoencoders

Alternative pretraining approach



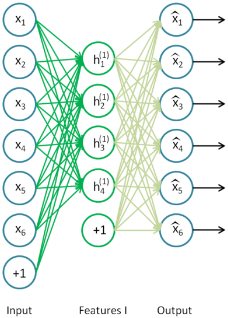
- Besides pre-training using RBMs, we may also “expand” a deep autoencoders as a stack of shallow autoecoders
- Shallow autoencoders are easier to train than RBM

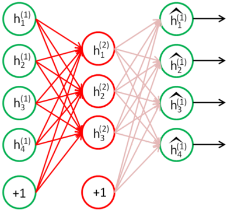
Stacked autoencoders
Alternative pretraining approach

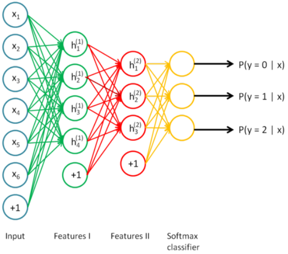


- Besides pre-training using RBMs, we may also “expand” a deep autoencoders as a stack of shallow autoecoders
- Shallow autoencoders are easier to train than RBM

Stacked autoencoders
Alternative pretraining approach

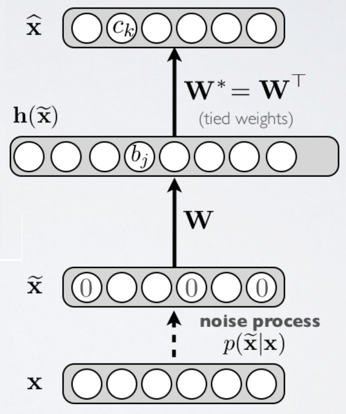






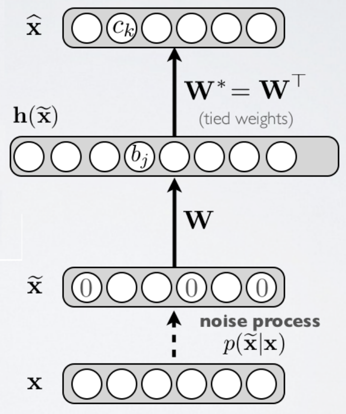
- Besides pre-training using RBMs, we may also “expand” a deep autoencoders as a stack of shallow autoecoders
- Shallow autoencoders are easier to train than RBM

Denoising autoencoders
Vincent *et al.* 2008



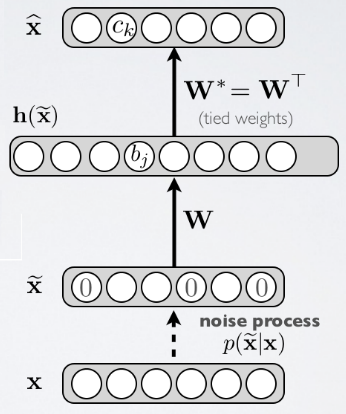
- Idea: representation should be robust to introduction of noise
  - Randomly assign bits to zero for binary case
    - Similar to dropout but for inputs only
  - Gaussian additive noise for continuous case
- Loss function compares $\hat{\mathbf{x}}$ with noiseless input $\mathbf{x}$

Denoising autoencoders
Vincent *et al.* 2008



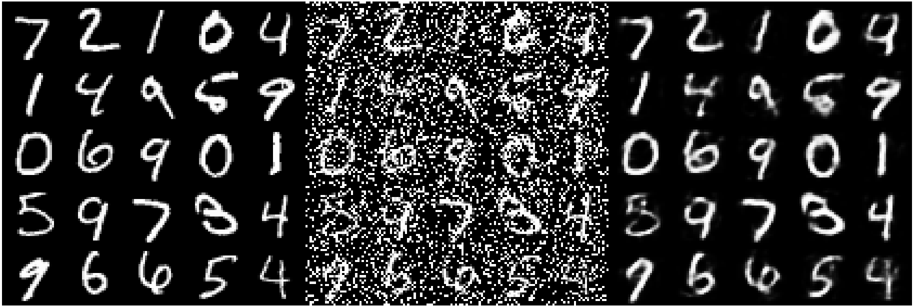
- Idea: representation should be robust to introduction of noise
  - Randomly assign bits to zero for binary case
    - Similar to dropout but for inputs only
  - Gaussian additive noise for continuous case
- Loss function compares $\hat{\mathbf{x}}$ with noiseless input $\mathbf{x}$

Denoising autoencoders
Vincent *et al.* 2008



- Idea: representation should be robust to introduction of noise
  - Randomly assign bits to zero for binary case
    - Similar to dropout but for inputs only
  - Gaussian additive noise for continuous case
- Loss function compares $\hat{\mathbf{x}}$ with noiseless input $\mathbf{x}$

Denoising autoencoders



Contractive autoencoders
Rifai *et al.* 2011

- Idea: encourage robustness of the model by forcing the hidden units to be insensitive to slight change of inputs
- Achieve this by penalizing the squared gradient of each hidden activity w.r.t. the inputs

$$l(\mathbf{x}) \to l(\mathbf{x}) + \lambda \|\nabla_{\mathbf{x}} h(\mathbf{x})\|_F^2$$

- Pros and cons
  - + deterministic gradient ⇒ can use second order optimizers
  - + could be more stable than denoising autoencoder, which needs to use a sampled gradient
  - - Need to compute Jacobian of hidden layer
  - - More complex than denoising autoencoder, which just needs to add one two lines of code

Contractive autoencoders

Rifai *et al.* 2011

- Idea: encourage robustness of the model by forcing the hidden units to be insensitive to slight change of inputs
- Achieve this by penalizing the squared gradient of each hidden activity w.r.t. the inputs

$$l(\mathbf{x}) \to l(\mathbf{x}) + \lambda \|\nabla_{\mathbf{x}} h(\mathbf{x})\|_F^2$$

- Pros and cons
  - + deterministic gradient $\Rightarrow$ can use second order optimizers
  - + could be more stable than denoising autoencoder, which needs to use a sampled gradient
  - - Need to compute Jacobian of hidden layer
  - - More complex than denoising autoencoder, which just needs to add one two lines of code

Contractive autoencoders

Rifai *et al.* 2011

- Idea: encourage robustness of the model by forcing the hidden units to be insensitive to slight change of inputs
- Achieve this by penalizing the squared gradient of each hidden activity w.r.t. the inputs

$$l(\mathbf{x}) \to l(\mathbf{x}) + \lambda \|\nabla_{\mathbf{x}} h(\mathbf{x})\|_F^2$$

- Pros and cons
  - + deterministic gradient ⇒ can use second order optimizers
  - + could be more stable than denoising autoencoder, which needs to use a sampled gradient
  - - Need to compute Jacobian of hidden layer
  - - More complex than denoising autoencoder, which just needs to add one two lines of code

Contractive autoencoders
Rifai *et al.* 2011

- Idea: encourage robustness of the model by forcing the hidden units to be insensitive to slight change of inputs
- Achieve this by penalizing the squared gradient of each hidden activity w.r.t. the inputs

$$l(\mathbf{x}) \to l(\mathbf{x}) + \lambda \|\nabla_{\mathbf{x}} h(\mathbf{x})\|_F^2$$

- Pros and cons
  - + deterministic gradient ⇒ can use second order optimizers
  - + could be more stable than denoising autoencoder, which needs to use a sampled gradient
  - - Need to compute Jacobian of hidden layer
  - - More complex than denoising autoencoder, which just needs to add one two lines of code

Contractive autoencoders

Rifai *et al.* 2011

- Idea: encourage robustness of the model by forcing the hidden units to be insensitive to slight change of inputs
- Achieve this by penalizing the squared gradient of each hidden activity w.r.t. the inputs

$$l(\mathbf{x}) \to l(\mathbf{x}) + \lambda \|\nabla_{\mathbf{x}} h(\mathbf{x})\|_F^2$$

- Pros and cons
  - + deterministic gradient ⇒ can use second order optimizers
  - + could be more stable than denoising autoencoder, which needs to use a sampled gradient
  - - Need to compute Jacobian of hidden layer
  - - More complex than denoising autoencoder, which just needs to add one two lines of code

Contractive autoencoders

Rifai *et al.* 2011

- Idea: encourage robustness of the model by forcing the hidden units to be insensitive to slight change of inputs
- Achieve this by penalizing the squared gradient of each hidden activity w.r.t. the inputs

$$l(\mathbf{x}) \to l(\mathbf{x}) + \lambda \|\nabla_{\mathbf{x}} h(\mathbf{x})\|_F^2$$

- Pros and cons
    - + deterministic gradient $\Rightarrow$ can use second order optimizers
    - + could be more stable than denoising autoencoder, which needs to use a sampled gradient
    - - Need to compute Jacobian of hidden layer
    - - More complex than denoising autoencoder, which just needs to add one two lines of code

Remark on pretraining

What are the disadvantages of pretraining deep neural networks by stacking autoencoders?

Answer | Request | Follow 55 | Comment | Downvote

1 Answer

Yoshua Bengio, My lab has been one of the three that started the deep learning approach, back in 2006, along with Hinton's...

Answered Aug 14, 2014 · Upvoted by Zeeshan Zia, PhD in Computer Vision and Machine Learning and Jason Li, AI researcher.

The same disadvantage as other layer-wise pre-training techniques: it is greedy, i.e., it does not try to tune the lower layers in a way that will make the work of higher layers easier. But that will change soon with a new approach I am working on!

Remark on pretraining

Ian Goodfellow, Lead author of the Deep Learning textbook: http://www.deeplearningbook.org

Answered Sep 28, 2016 · Upvoted by Aaditya Prakash, Graduate student in Computer Vision and Deep Learning and Abhinav Maurya, PhD Student (Machine Learning, Public Policy) at CMU

Autoencoders are useful for some things, but turned out not to be nearly as necessary as we once thought. Around 10 years ago, we thought that deep nets would not learn correctly if trained with only backprop of the supervised cost. We thought that deep nets would also need an unsupervised cost, like the autoencoder cost, to regularize them. When Google Brain built their first very large neural network to recognize objects in images, it was an autoencoder (and it didn't work very well at recognizing objects compared to later approaches). Today, we know we are able to recognize images just by using backprop on the supervised cost as long as there is enough labeled data. There are other tasks where we do still use autoencoders, but they're not the fundamental solution to training deep nets that people once thought they were going to be.

Variational autoencoders

"Generative autoencoders" $\Rightarrow$ variational autoencoders

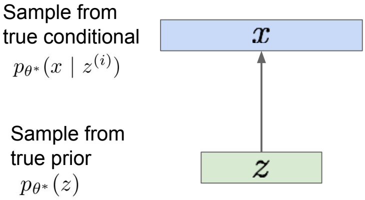
- Instead of spitting out an approximate for the input
- The network spits out parameters of a distribution

Variational autoencoders

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 underlying unobserved (latent) representation **z**



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

Variational autoencoders

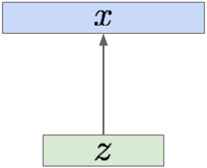
Variational Autoencoders

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 underlying unobserved (latent) representation **z**

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

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



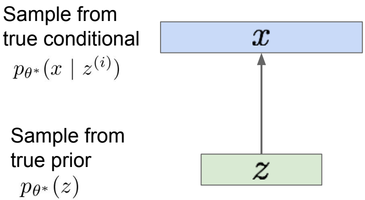
Intuition (remember from autoencoders!): **x** is an image, **z** is latent factors used to generate **x:** attributes, orientation, etc.

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

Variational autoencoders

Variational Autoencoders

We want to estimate the true parameters θ^* of this generative model.



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

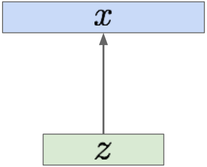
Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 13 - 53 May 18, 2017

Variational autoencoders

Variational Autoencoders

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

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



We want to estimate the true parameters θ^* of this generative model.

How should we represent this model?

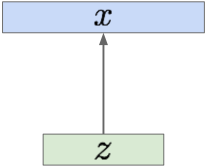
Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Variational autoencoders

Variational Autoencoders

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

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



We want to estimate the true parameters θ^* of this generative model.

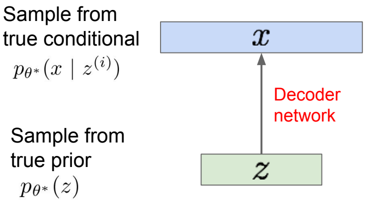
How should we represent this model?

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

Variational Autoencoders



We want to estimate the true parameters θ^* of this generative model.

How should we represent this model?

Choose prior p(z) to be simple, e.g. Gaussian.

Conditional p(x|z) is complex (generates image) => represent with neural network

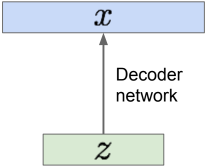
Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Variational autoencoders

Variational Autoencoders

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

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



We want to estimate the true parameters θ^* of this generative model.

How to train the model?

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

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 13 - 57 May 18, 2017

Variational autoencoders

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

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

Variational Autoencoders: Intractability

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

Decoder neural network

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

Variational autoencoders

Variational Autoencoders: Intractability

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

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

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

Variational autoencoders

Variational Autoencoders: Intractability

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

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

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

Variational autoencoders

Variational Autoencoders: Intractability

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

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

Intractable data likelihood

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

Variational autoencoders

Variational Autoencoders: Intractability

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

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

Solution: In addition to decoder network modeling $p_\theta(x|z)$, define additional encoder network $q_\phi(z|x)$ that approximates $p_\theta(z|x)$

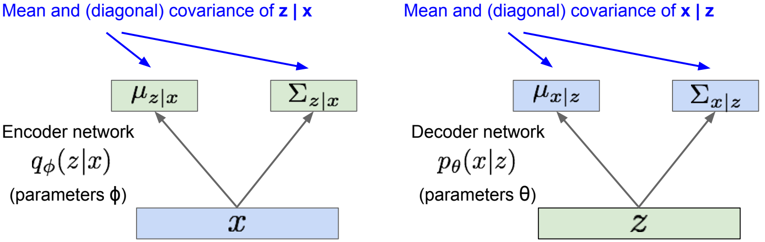
Will see that this allows us to derive a lower bound on the data likelihood that is tractable, which we can optimize

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

Variational autoencoders

Variational Autoencoders

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

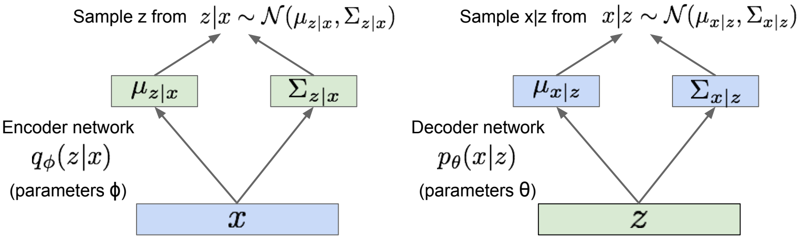


Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Variational autoencoders

Variational Autoencoders

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

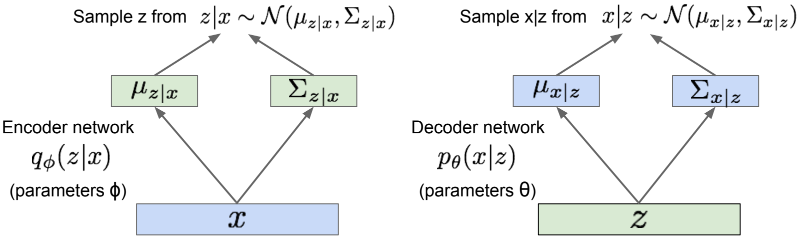


Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Variational autoencoders

Variational Autoencoders

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



Encoder and decoder networks also called "recognition"/"inference" and "generation" networks

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

Variational autoencoders

Variational Autoencoders

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

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

Variational autoencoders

Variational Autoencoders

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

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

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

Variational autoencoders

Variational Autoencoders

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

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

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

Variational autoencoders

Variational Autoencoders

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

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

Variational autoencoders

Variational Autoencoders

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

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

Variational autoencoders

Variational Autoencoders

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

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

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 13 - 77 May 18, 2017

Variational autoencoders

Variational Autoencoders

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

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

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

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

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

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

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

Variational autoencoders

Variational Autoencoders

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

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

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

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

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

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

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 autoencoders

Variational Autoencoders

Now equipped with our encoder and decoder networks, let's work out the (log) 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

Variational Autoencoders

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

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

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

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

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

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

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

Variational lower bound ("ELBO")

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

Training: Maximize lower bound

Fei-Fei Li & Justin Johnson & Serena Yeung | Lecture 13 - 81 | May 18, 2017

Variational autoencoders

Variational Autoencoders

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

$$
\begin{aligned}
\log p_\theta(x^{(i)}) &= \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z) \\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] \quad (\text{Bayes' Rule}) \\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] \quad (\text{Multiply by constant}) \\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] \quad (\text{Logarithms}) \\
&= \underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)},\theta,\phi)} + \underbrace{D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))}_{\geq 0}
\end{aligned}
$$

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

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

Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

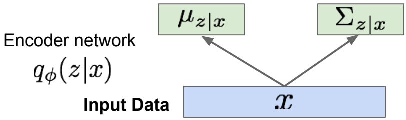
Let’s look at computing the bound (forward pass) for a given minibatch of input data

Variational autoencoders

Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

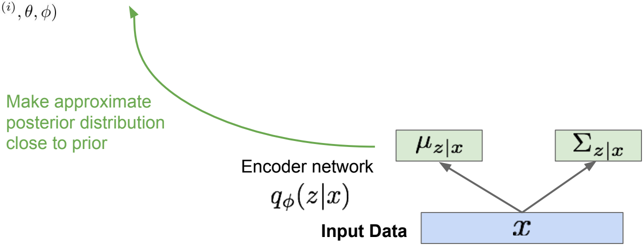


Variational autoencoders

Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

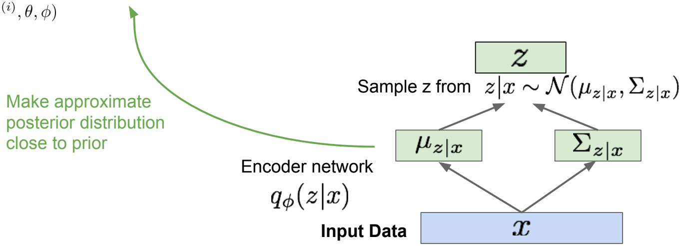


Variational autoencoders

Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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



Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 13 - 87 May 18, 2017

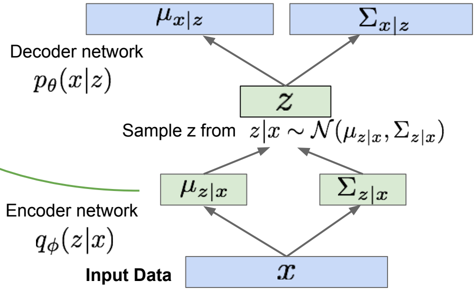
Variational autoencoders

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

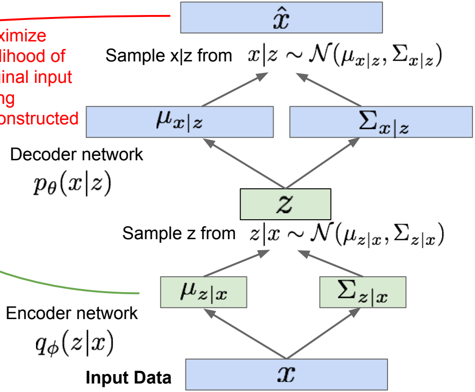
Variational Autoencoders

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

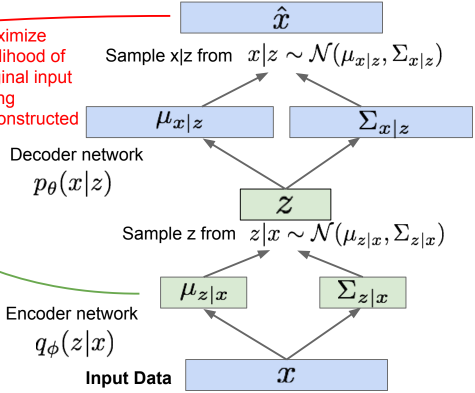
Variational Autoencoders

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



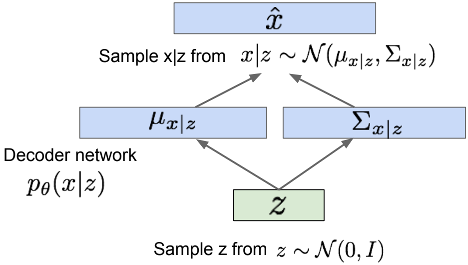
For every minibatch of input data: compute this forward pass, and then backprop!

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 13 - 90 May 18, 2017

Variational autoencoders

Variational Autoencoders: Generating Data!

Use decoder network. Now sample z from prior!

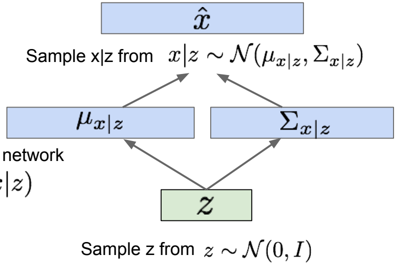


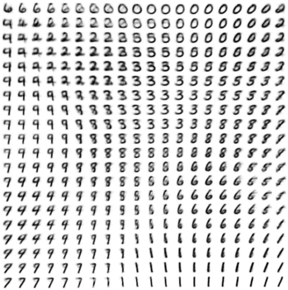
Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Variational autoencoders

Variational Autoencoders: Generating Data!

Use decoder network. Now sample z from prior!



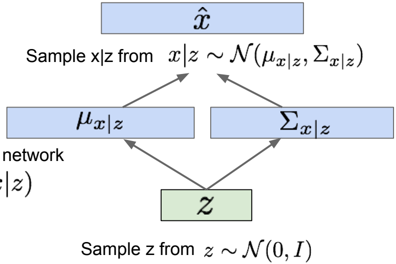


Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Variational autoencoders

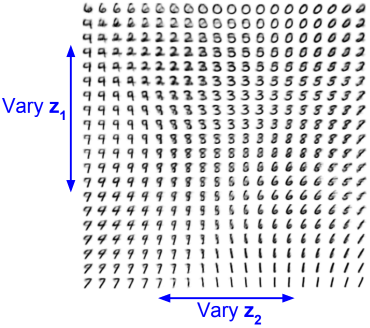
Variational Autoencoders: Generating Data!

Use decoder network. Now sample z from prior!



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

Data manifold for 2-d **z**



Variational autoencoders

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

Variational Autoencoders: Generating Data!

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

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

Degree of smile

Vary $\mathbf{z}_1$

Vary $\mathbf{z}_2$

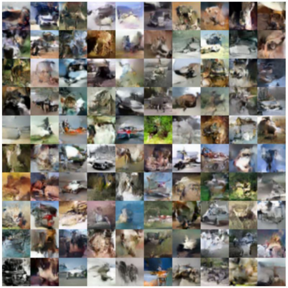
Head pose

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

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

Variational autoencoders

Variational Autoencoders: Generating Data!



32x32 CIFAR-10

Labeled Faces in the Wild

Summary of variational autoencoders

- Probabilistic spin to traditional autoencoders to allow data generation. Use variational lower bound to workaround intractable density estimation

  Pros
  - Principled approach to generative models
  - Allows inference of $q(z|x)$ that can be used for feature representation

  Cons
  - Maximizes lower bound rather than exact cost function. Less direct than say PixelRNN/PixelCNN
  - Samples generated are lower quality compared to the state-of-the-art (GANs)
- Follow-up research:
  - More flexible approximations, e.g., richer model in approximating the posterior (typically just use diagonal Gaussian in the basic model)
  - Incorporating structure in latent variables
  - Disentangled variational autoencoder

Conclusions

- Conventional autoencoders are important tools for dimension reduction and data representation in general
- Generative models are some very exciting hot topics in deep learning
  - Especially useful for datasets with few or no labels
  - Many other possible applications to be discovered
- We discuss two state-of-the-art generative models
  - Variational autoencoders: autoencoders + variational inference
  - Generative adversarial networks (GANs): more recent and gaining lots of interests

Conclusions

- Conventional autoencoders are important tools for dimension reduction and data representation in general
- Generative models are some very exciting hot topics in deep learning
    - Especially useful for datasets with few or no labels
    - Many other possible applications to be discovered
- We discuss two state-of-the-art generative models
    - Variational autoencoders: autoencoders + variational inference
    - Generative adversarial networks (GANs): more recent and gaining lots of interests

Conclusions

- Conventional autoencoders are important tools for dimension reduction and data representation in general
- Generative models are some very exciting hot topics in deep learning
    - Especially useful for datasets with few or no labels
    - Many other possible applications to be discovered
- We discuss two state-of-the-art generative models
    - Variational autoencoders: autoencoders + variational inference
    - Generative adversarial networks (GANs): more recent and gaining lots of interests