Session 6

Deep Generative model

Mini-project topics

Acknowledgements

- The materials majorly derived http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture14.pdf
- OpenAI spinningup https://spinningup.openai.com/en/latest/spinningup/rl_intro.html
- David Sliver

https://www.davidsilver.uk/wp-content/uploads/2020/03/intro_RL.pdf

Overview

- Introduction
- Fully visible belief network
- Boltzmann machine/RBM/DBM
- Autoencoder
- GAN

Supervised vs Unsupervised

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

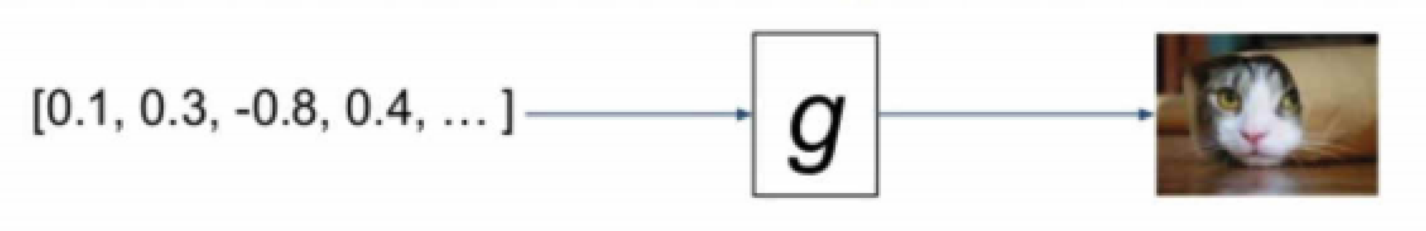
Goal: Learn some underlying hidden *structure* of the data

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

Unsupervised Learning

- Examples:
  - Dimension reduction : PCA
  - Clustering: k-means
  - Density estimation
  - Feature learning

- General framework:
  - Find deterministic function f: z=f(x), x is data, z is the latent



Unsupervised learning vs. Generative model

- $z = f(x)$ vs. $x = g(z)$
- $P(z|x)$ vs. $P(x|z)$
- **Encoder** vs. **Decoder (Generator)**
  - P(x, z) needed. (cf : P(y|x) in supervised learning)
    - $P(z|x) = P(x, z) / P(x)$
    - $P(x|z) = P(x, z) / P(z)$ → P(z) is given. (prior)

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

Taxonomy of Generative Models

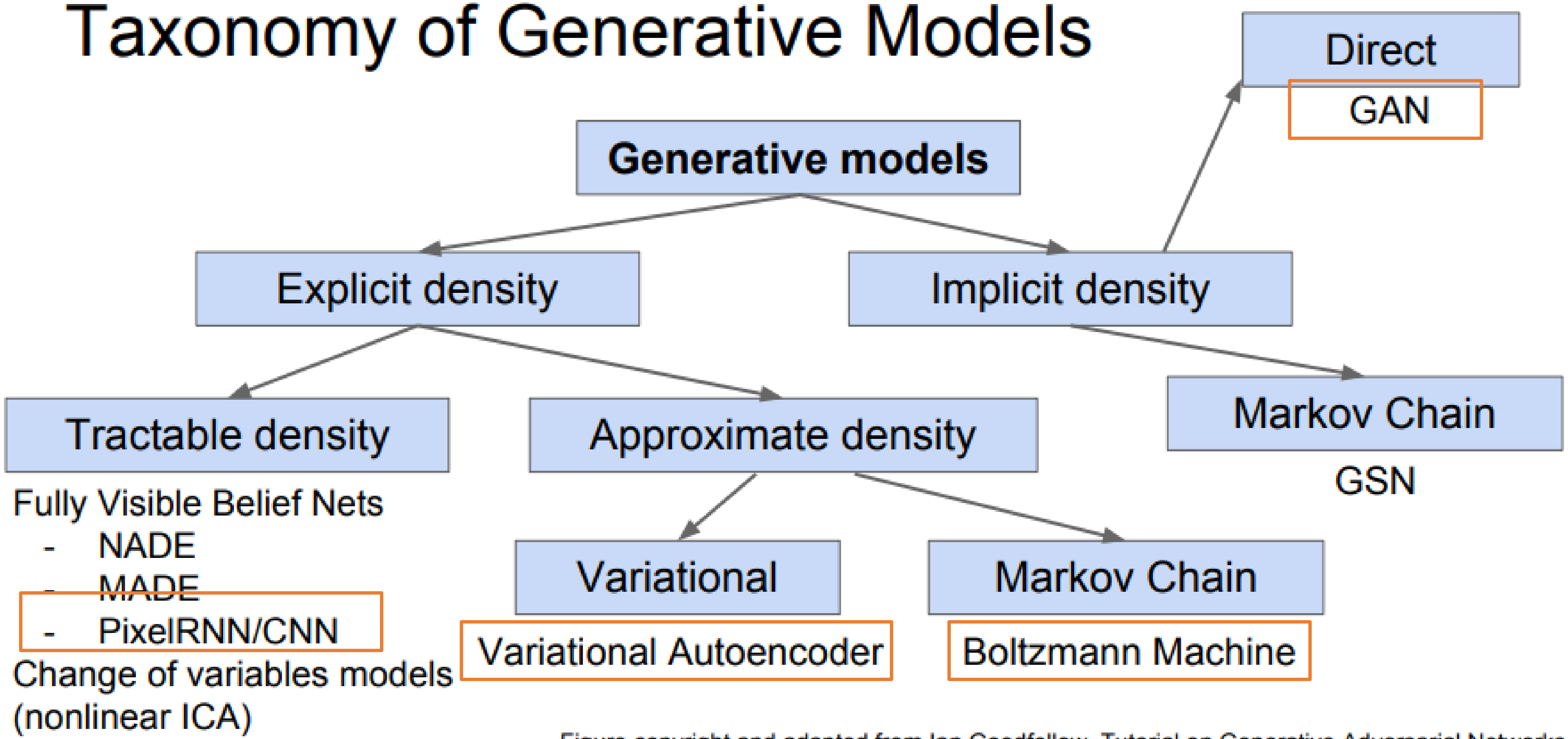


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

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

Will need to define ordering of "previous pixels"

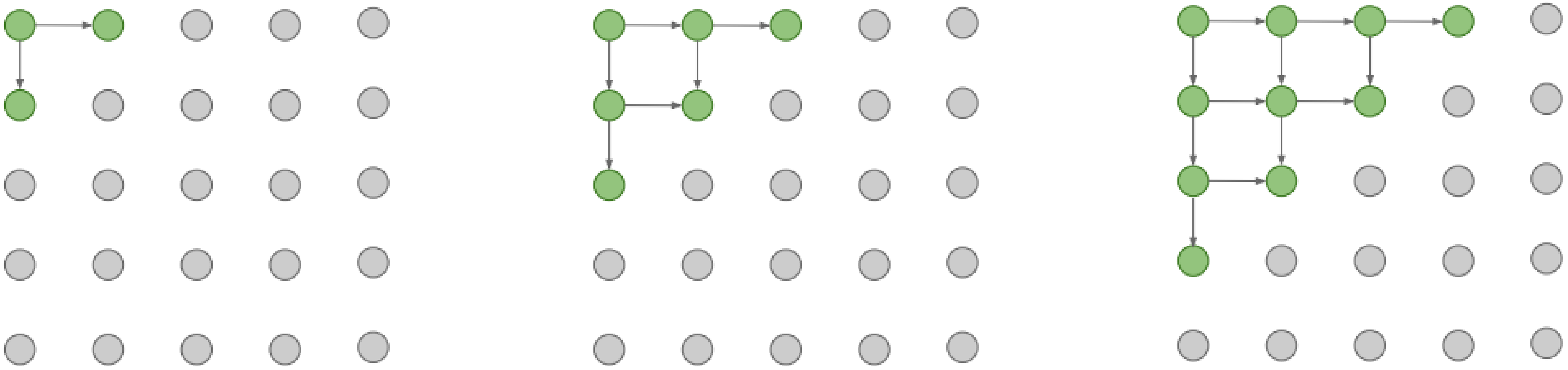
Then maximize likelihood of training data

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

Drawback: sequential generation is slow!

PixelRNN [van der Oord et al. 2016]

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



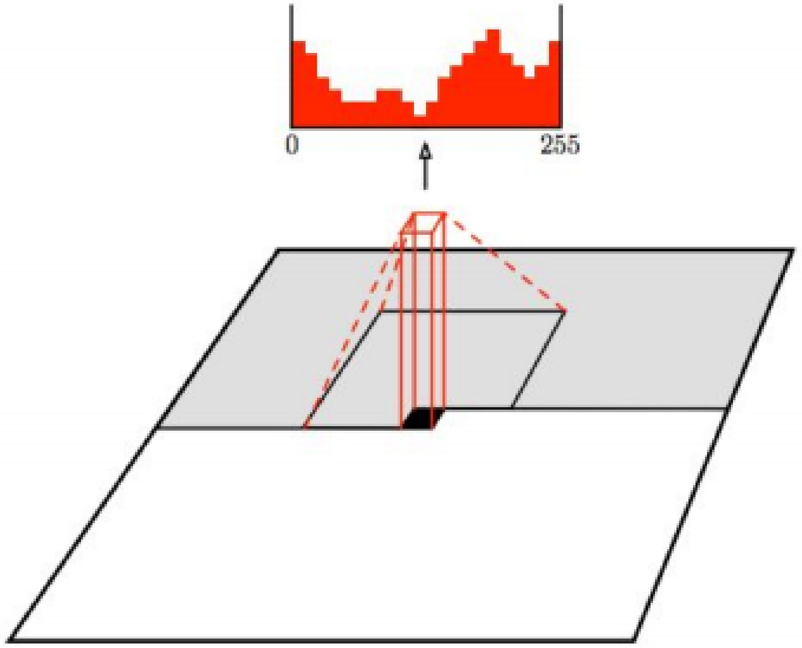
PixelCNN [van der Oord et al. 2016]

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



PixelRNN and PixelCNN

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

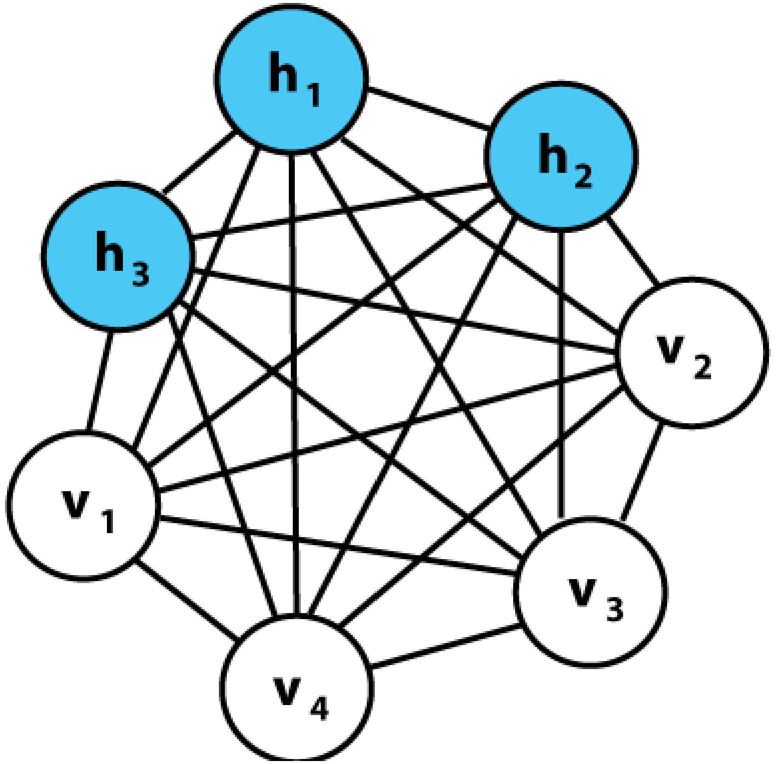
Con:
- Sequential generation => slow

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++)

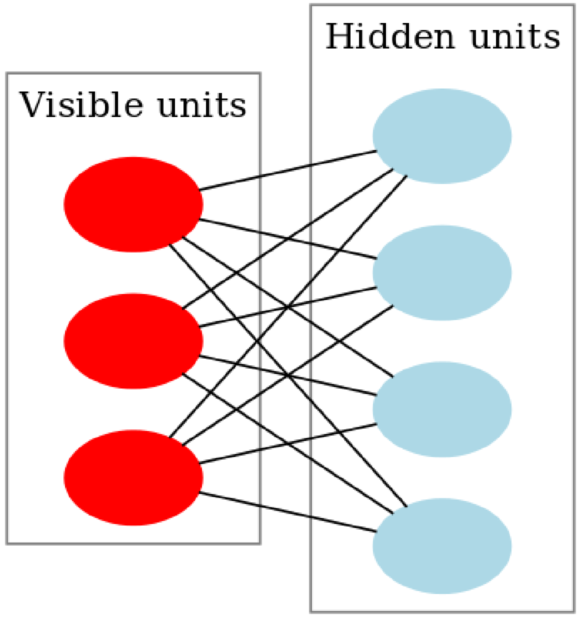
What is Boltzmann machine?

- Boltzmann machines are stochastic and generative neural networks capable of learning internal representations
- They are able to represent (given sufficient time) and solve difficult combinatory problems.
- They were invented in 1985 by Geoffrey Hinton
- non-deterministic (or stochastic) generative Deep Learning models with only two types of nodes: visible (v) and hidden (h)
- Unlike classical neural networks
  - No output nodes
  - Connection between input nodes (v)
- This allows them to share information among themselves and self-generate subsequent data



Restricted Boltzmann machine (RBM)

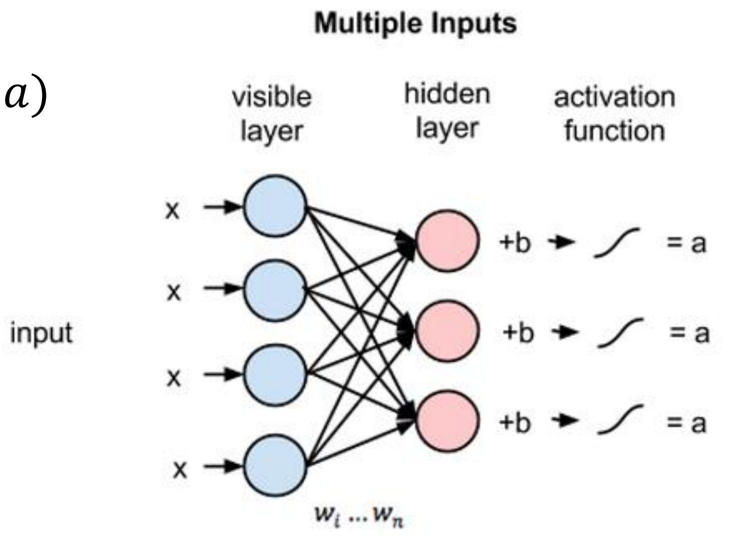
- RBMs are a two-layered artificial neural network with generative capabilities.
- They have the ability to learn a probability distribution over its set of input.
- It can be used for dimensionality reduction, classification, regression, collaborative filtering, feature learning, and topic modeling.
- RBMs are a special class of Boltzmann Machines and they are restricted in terms of the connections between the visible and the hidden units.
- Every node in the visible layer is connected to every node in the hidden layer but no two nodes in the same group are connected to each other.
- This restriction allows for more efficient training algorithms than what is available for the general class of Boltzmann machines, in particular, the gradient-based contrastive divergence algorithm.

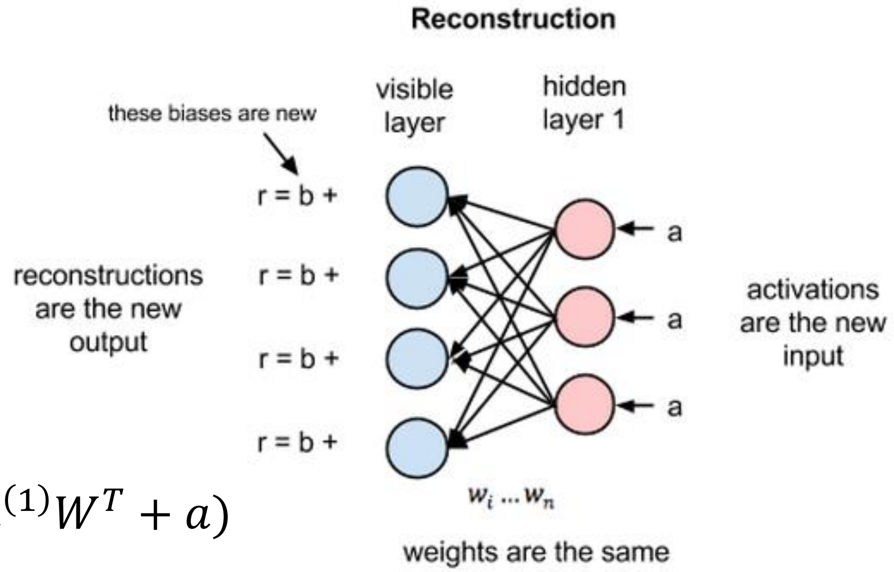


RBM

$$h^1 = S(v^{(0)T}W + a)$$

- RBM is a Stochastic Neural Network which means that each neuron will have some random behavior when activated.
- There are two other layers of bias units (hidden bias and visible bias) in an RBM.
- The hidden bias RBM produce the activation on the forward pass and the visible bias helps RBM to reconstruct the input during a backward pass.
- The reconstructed input is always different from the actual input as there are no connections among the visible units and therefore, no way of transferring information among themselves.





$$v^1 = S(h^{(1)}W^T + a)$$

Learning process of RBM

- Consider the difference v(0)-v(10) as reconstruction error. The weights are adjusted in each iteration so as to minimize this error
- In forward pass: $p\left(h^{(1)} \middle| v^{(0)}; W\right)$
- In backward pass: $p\left(v^{(1)} \middle| h^{(1)}; W\right)$
- It is a joint distribution: p(v,h)
- Assume that we have two normal distributions, one from the input data (denoted by p(x)) and one from the reconstructed input approximation (denoted by q(x)). The difference between these two distributions is our error in the graphical sense and our goal is to minimize it ➔ Kullback-Leibler divergence (KL-divergence)
- KL-divergence measures the non-overlapping areas under the two graphs and the RBM's optimization algorithm tries to minimize this difference by changing the weights so that the reconstruction closely resembles the input.

Contrastive divergence

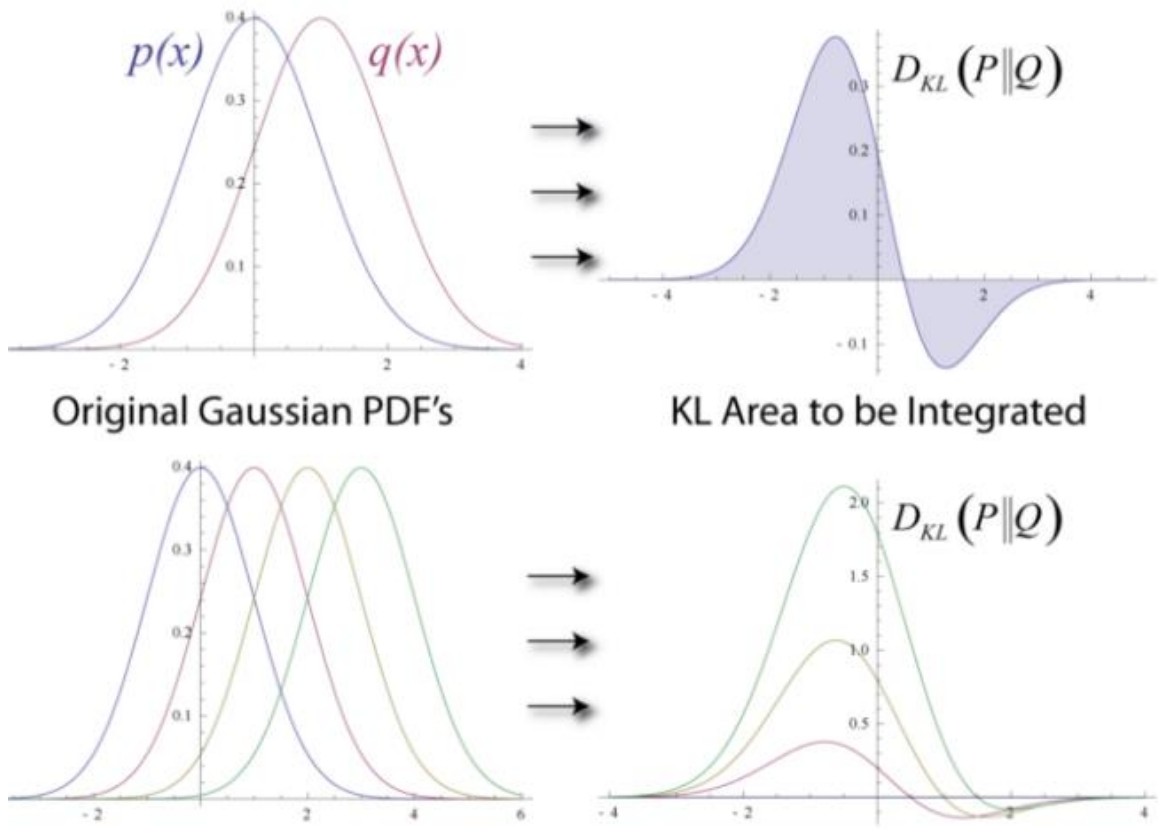


Image by [Mundhenk](#) on [Wikimedia](#)

Boltzmann Machines (and RBMs) are Energy-based models and a joint configuration, (**v,h**) of the visible and hidden units has an energy given by:

$$E(\mathbf{v},\mathbf{h}) = -\sum_{i\in visible} a_i v_i - \sum_{j\in hidden} b_j h_j - \sum_{i,j} v_i h_j w_{ij}$$

The probability that the network assigns to a visible vector, *v*, is given by summing over all possible hidden vectors

$$p(\mathbf{v}) = \frac{\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}}{\sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}}$$

$$\frac{\partial log p(\mathbf{v})}{\partial w_{ij}} = \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}$$

Expectation

reconstruct

$$\Delta w_{ij} = \alpha(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model})$$

Learning rate

Contrastive divergence

- The learning rule is much more closely approximating the gradient of another objective function called the **Contrastive Divergence** which is the difference between two Kullback-Liebler divergences

Algorithm 1. k-step contrastive divergence

Input: RBM $(V_1, \dots, V_m, H_1, \dots, H_n)$, training batch S

Output: gradient approximation Δw_{ij}, Δb_j and Δc_i for $i = 1, \dots, n$, $j = 1, \dots, m$

init $\Delta w_{ij} = \Delta b_j = \Delta c_i = 0$ for $i = 1, \dots, n$, $j = 1, \dots, m$

forall the $\boldsymbol{v} \in S$ **do**

  $\boldsymbol{v}^{(0)} \leftarrow \boldsymbol{v}$

  for $t = 0, \dots, k-1$ **do**

    for $i = 1, \dots, n$ **do** sample $h_i^{(t)} \sim p(h_i \,|\, \boldsymbol{v}^{(t)})$

    for $j = 1, \dots, m$ **do** sample $v_j^{(t+1)} \sim p(v_j \,|\, \boldsymbol{h}^{(t)})$

  for $i = 1, \dots, n$, $j = 1, \dots, m$ **do**

    $\Delta w_{ij} \leftarrow \Delta w_{ij} + p(H_i = 1 \,|\, \boldsymbol{v}^{(0)}) \cdot v_j^{(0)} - p(H_i = 1 \,|\, \boldsymbol{v}^{(k)}) \cdot v_j^{(k)}$

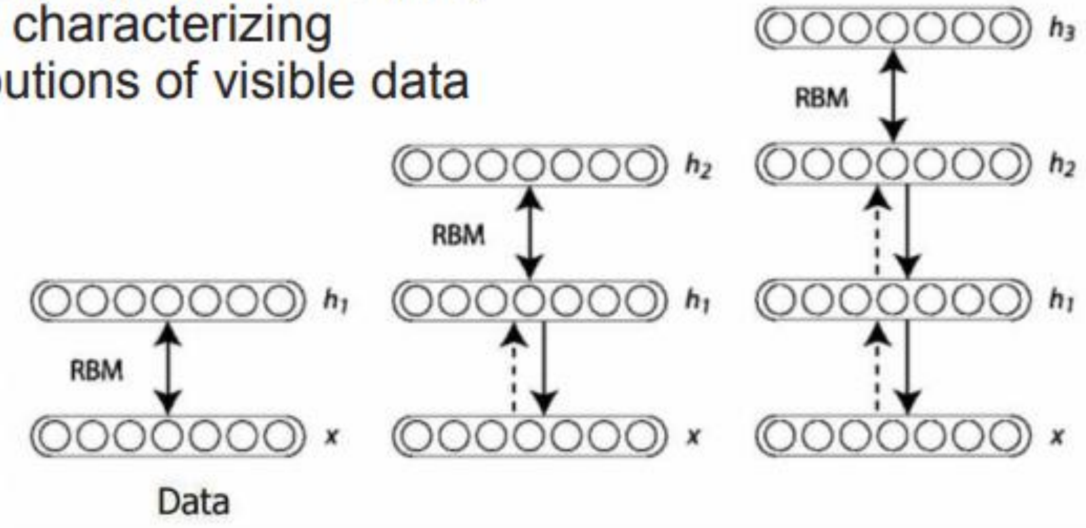
    $\Delta b_j \leftarrow \Delta b_j + v_j^{(0)} - v_j^{(k)}$

    $\Delta c_i \leftarrow \Delta c_i + p(H_i = 1 \,|\, \boldsymbol{v}^{(0)}) - p(H_i = 1 \,|\, \boldsymbol{v}^{(k)})$

Deep belief network (DBM)

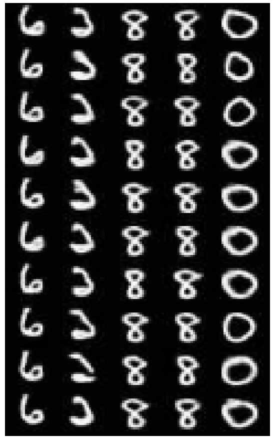
- Multiple RBMs can be stacked and can be fine-tuned through the process of gradient descent and back-propagation.
- One of first Deep-Learning models
- Proposed by G. Hinton in 2006
- Generative probabilistic model (mostly UNSUPERVISED)

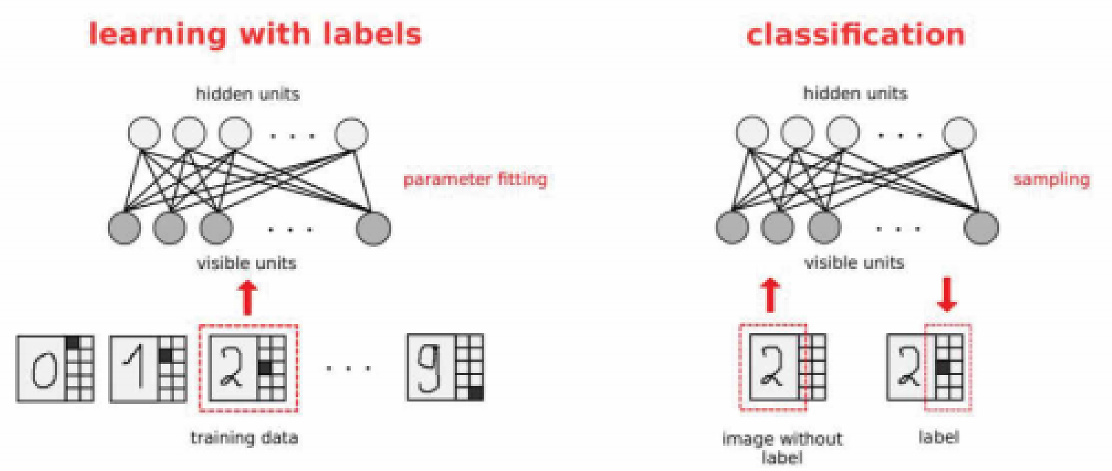
For capturing high-order *correlations* of observed/visible data (→ pattern analysis, or synthesis); and/or characterizing *joint* statistical distributions of visible data



Use of trained RBM

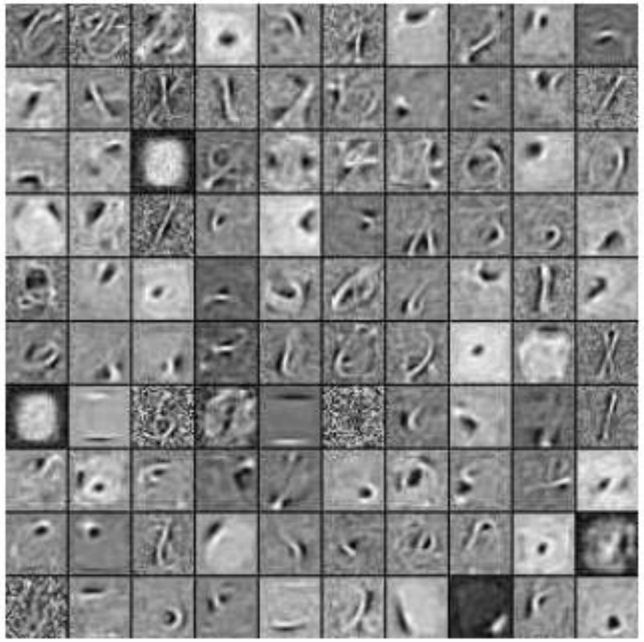
- **Input data "completion" : set some v_i then compute h, and generate compatible full samples**
- **Generating representative samples**
- **Classification if trained with inputs=data+label**

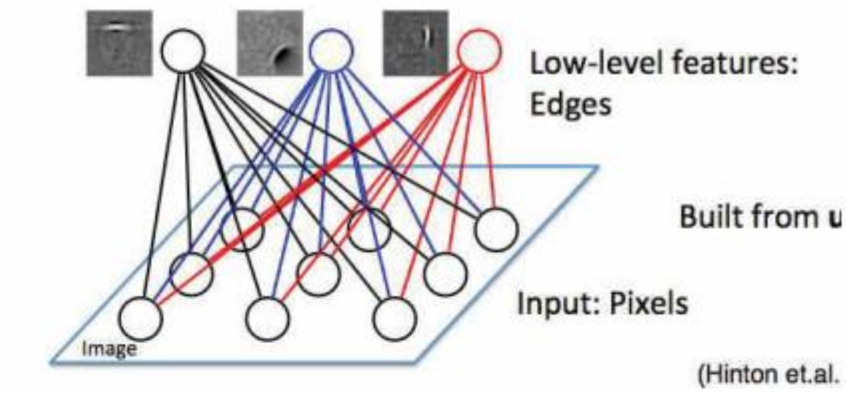




Interpretation of trained RBM hidden layer

- **Look at weights of hidden nodes → low-level features**

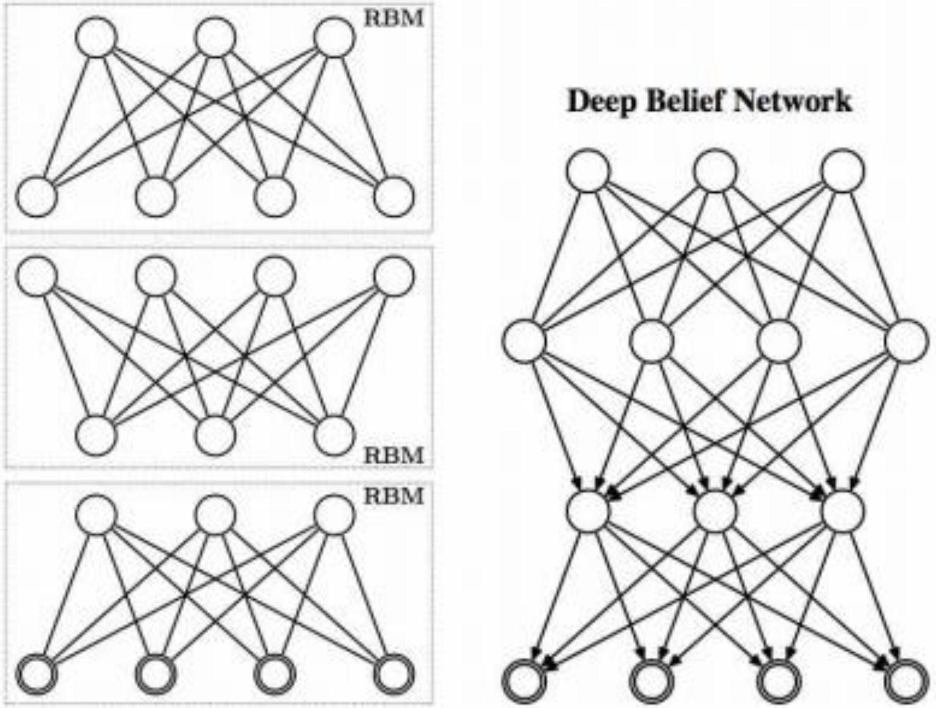




This is why people are inspired to stack the RBMs to get more “abstract” features.

Learning of DBN

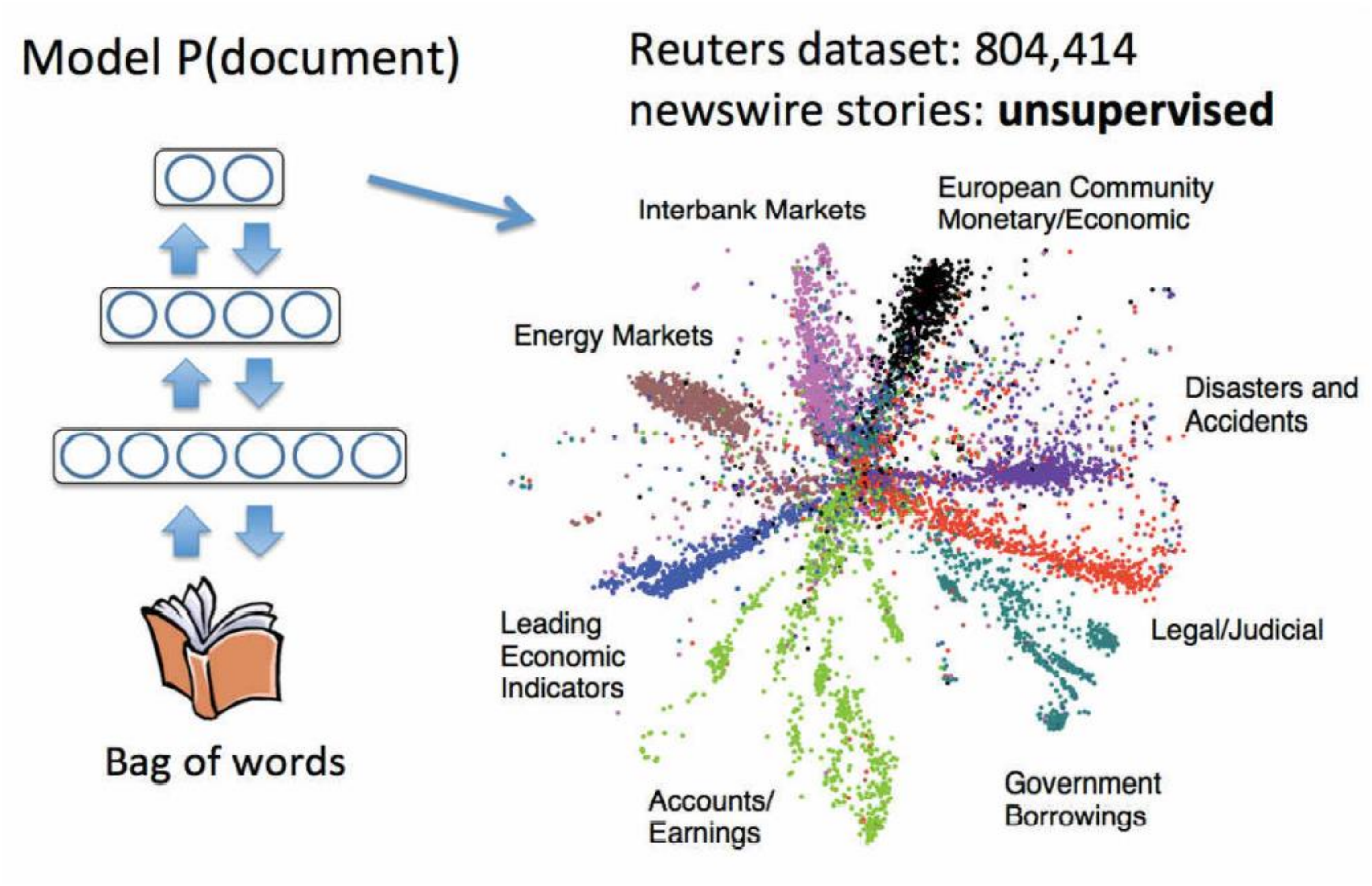
Greedy learning of successive layers



Algorithm 1 Recursive Greedy Learning Procedure for the DBN.

1: Fit parameters W^1 of the 1st layer RBM to data.
2: Freeze the parameter vector W^1 and use samples $\mathbf{h}^1$ from $Q(\mathbf{h}^1|\mathbf{v}) = P(\mathbf{h}^1|\mathbf{v}, W^1)$ as the data for training the next layer of binary features with an RBM.
3: Freeze the parameters W^2 that define the 2nd layer of features and use the samples $\mathbf{h}^2$ from $Q(\mathbf{h}^2|\mathbf{h}^1) = P(\mathbf{h}^2|\mathbf{h}^1, W^2)$ as the data for training the 3rd layer of binary features.
4: Proceed recursively for the next layers.

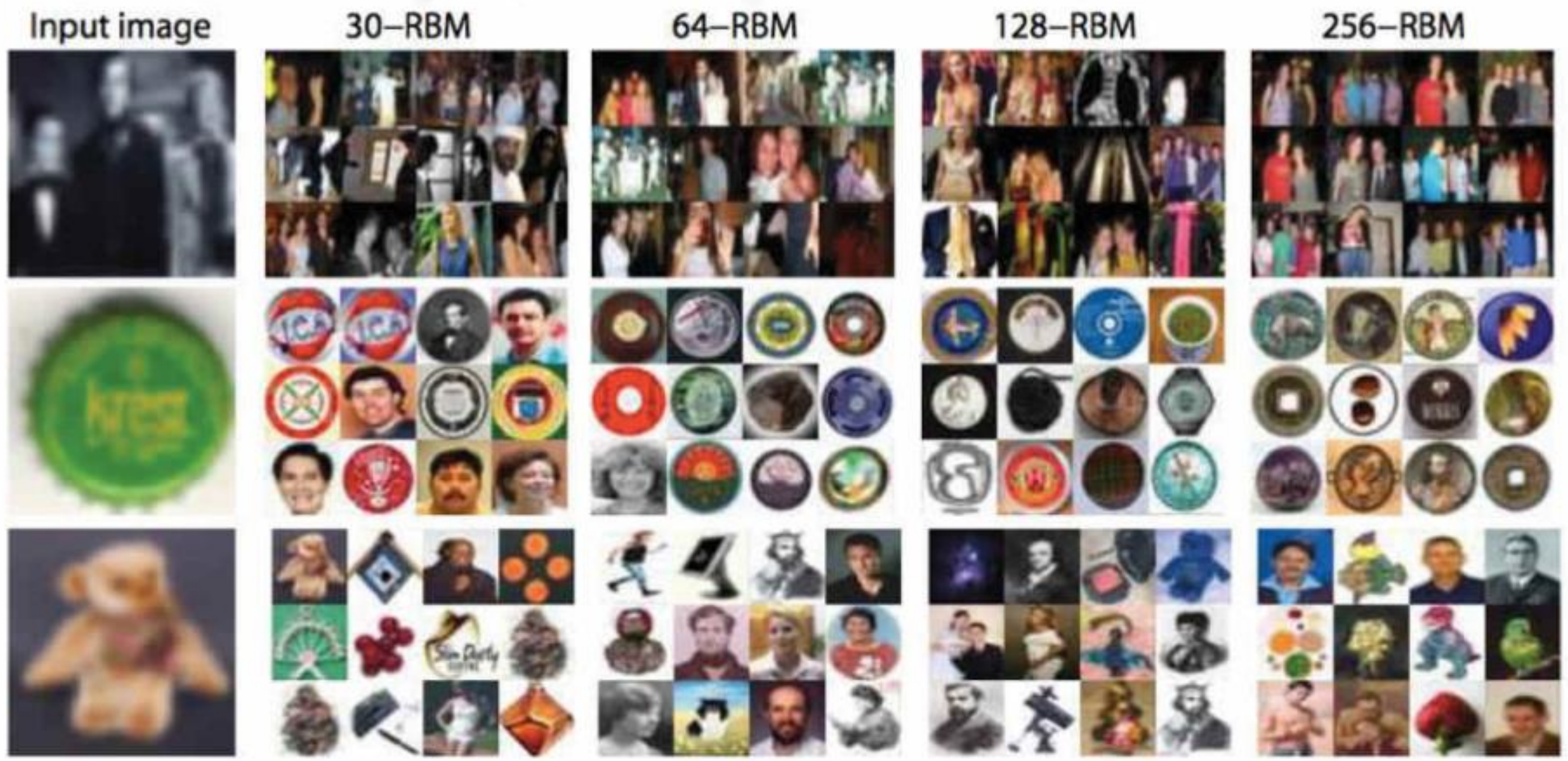
Example application of DBN: Clustering of documents in database



*The goal of this model is to find the internal representation in all the types (classes) of documents provided by Reuters dataset

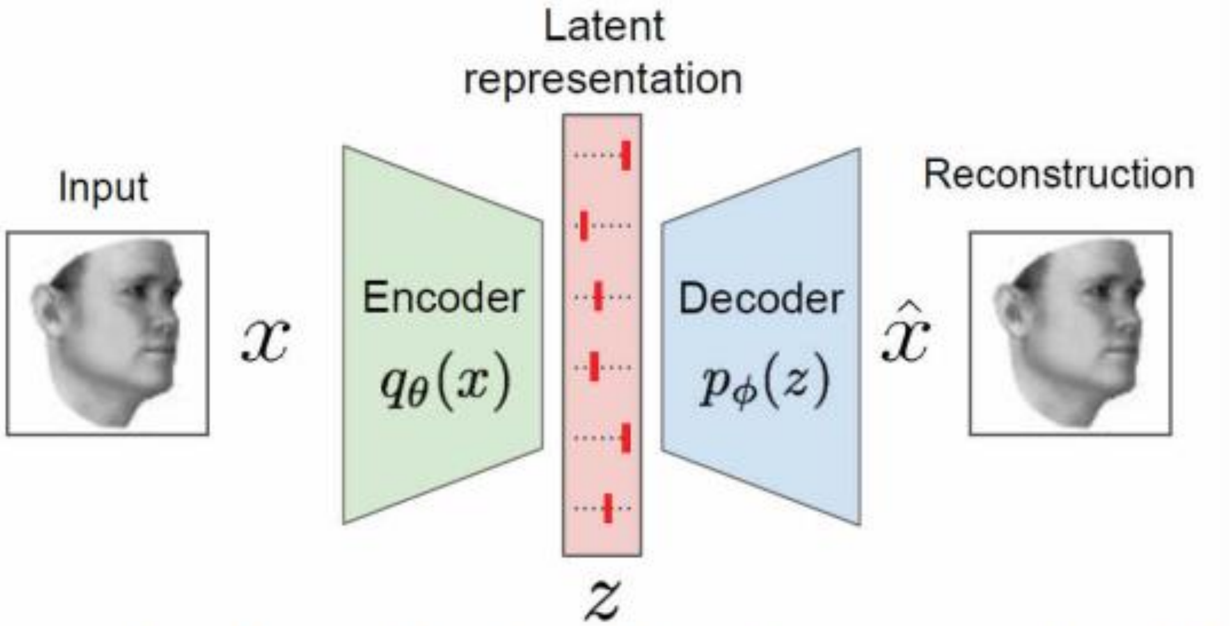
Image Retrieval application example

- Map images in to binary codes fast retrieval



[Small codes, Torralba, Fergus, Weiss, CVPR 2008]

Autoencoders

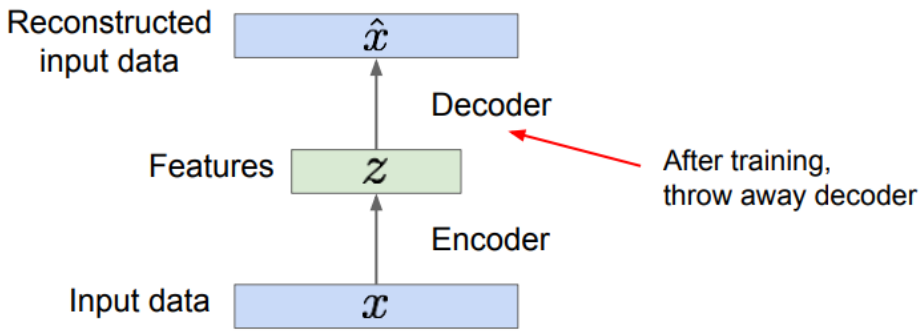


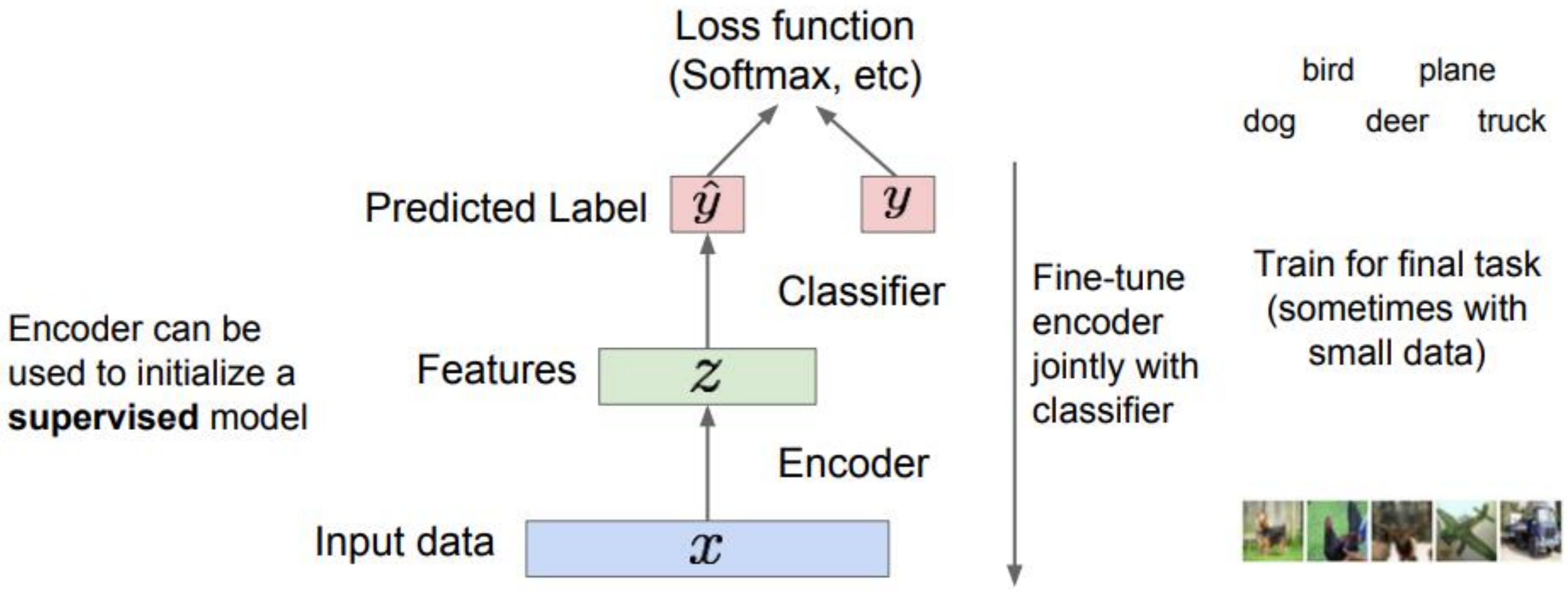
Learn q_θ and p_Φ in order to minimize *reconstruction cost*:

$$Q = \sum_k \|\hat{x}_k - x_k\|^2 = \sum_k \|p_\phi(q_\theta(x_k)) - x_k\|^2$$

→ unsupervised learning of latent variables, and of a generative model

Use case of AutoEncoder

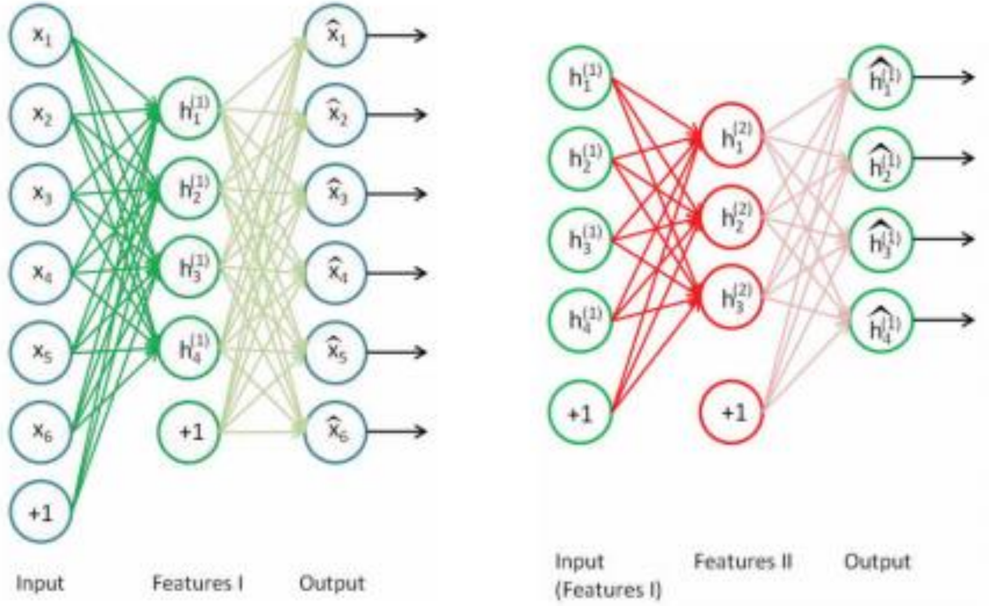




Variants of autoencoders

- Denoising autoencoders
- Sparse autoencoders
- Stochastic autoencoders
- Contractive autoencoders
- VARIATIONAL autoencoders
- …

Training of stacked Autoencoders



etc…

Greedy layerwise training:

for each layer k, use *backpropagation* to minimize
$|| A_k(h^{(k)})-h^{(k)} ||^2$ (+ regularization cost $\lambda \Sigma_{ij} |W_{ij}|^2$)
possibly + additional term for "sparsity"

AutoEncoders (AE)

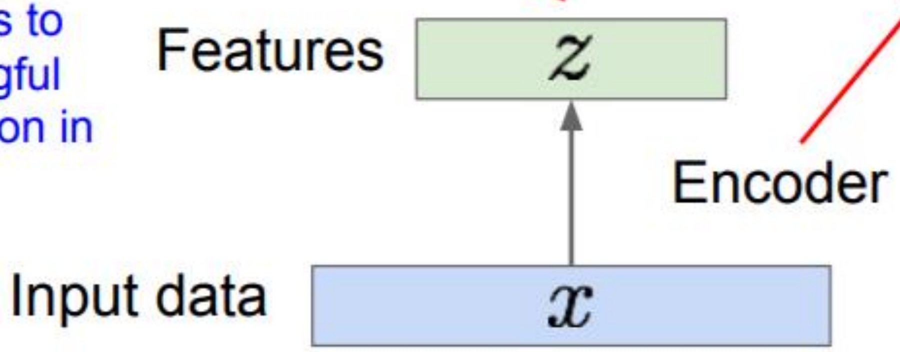
Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

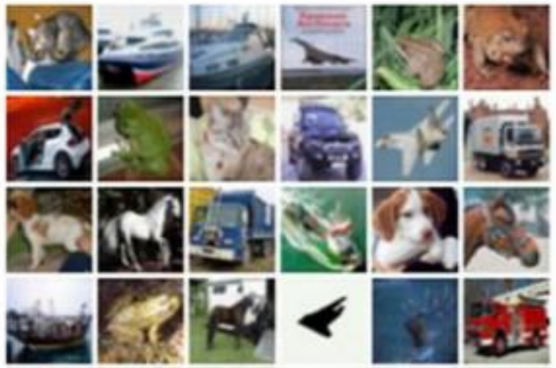
z usually smaller than **x** (dimensionality reduction)

Q: Why dimensionality reduction?

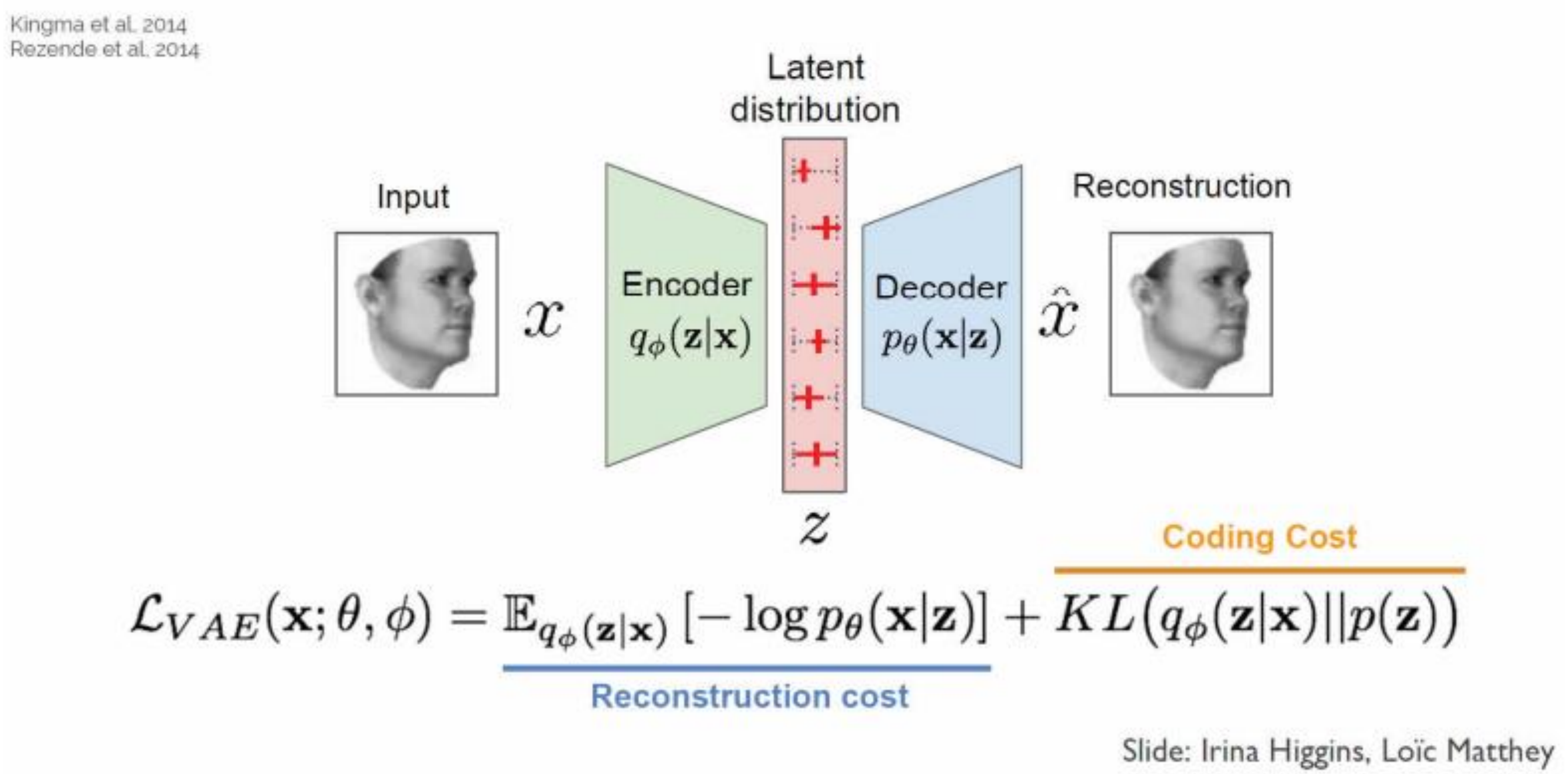
A: Want features to capture meaningful factors of variation in data

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





Variational AutoEncoders (VAE)



_KL_ = Kullback-Leibler divergence (a.k.a. ‘relative entropy’)
KL(Q || P) measures how different are distributions

VAE

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

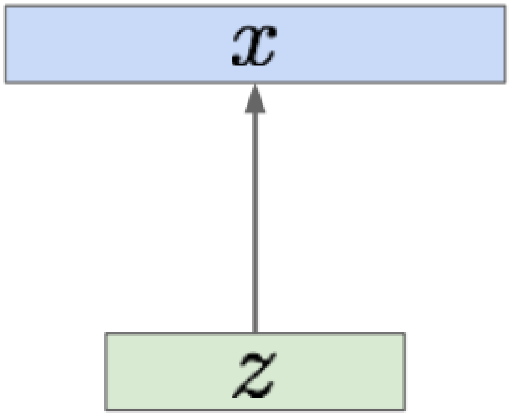
Assume training data $\{x^{(i)}\}_{i=1}^{N}$ is generated from underlying unobserved (latent) representation **z**

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.

How should we represent this model?

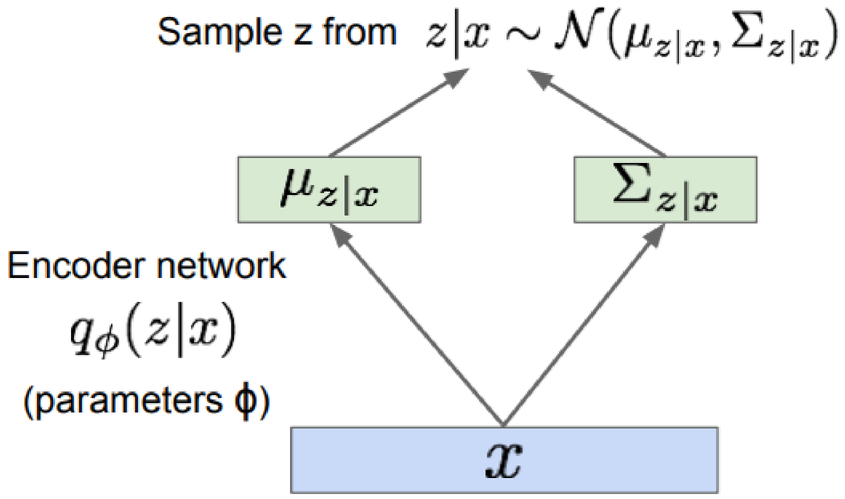
Choose prior p(z) to be simple, e.g. Gaussian.

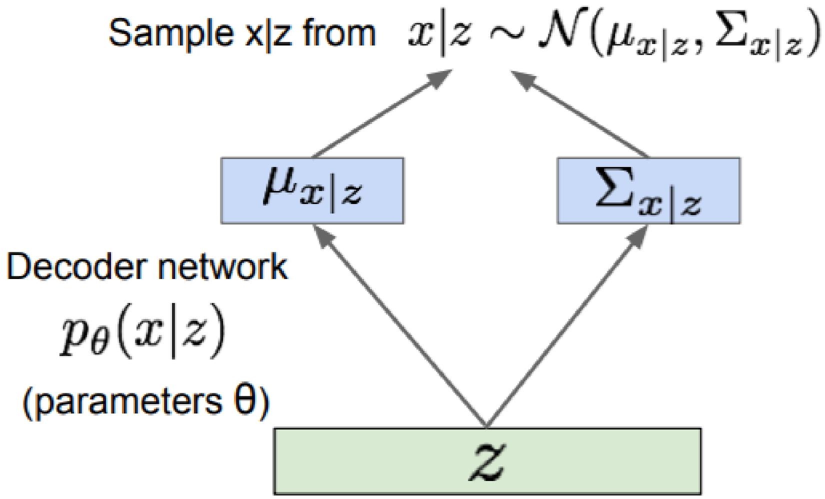
Conditional p(x|z) is complex (generates image) => represent with neural network

VAE

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

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





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.

Generative Adversarial Network (GAN)

GAN

[Introduced in 2014 by Ian Goodfellow et al. (incl. Yoshua Bengio) from University of Montreal]

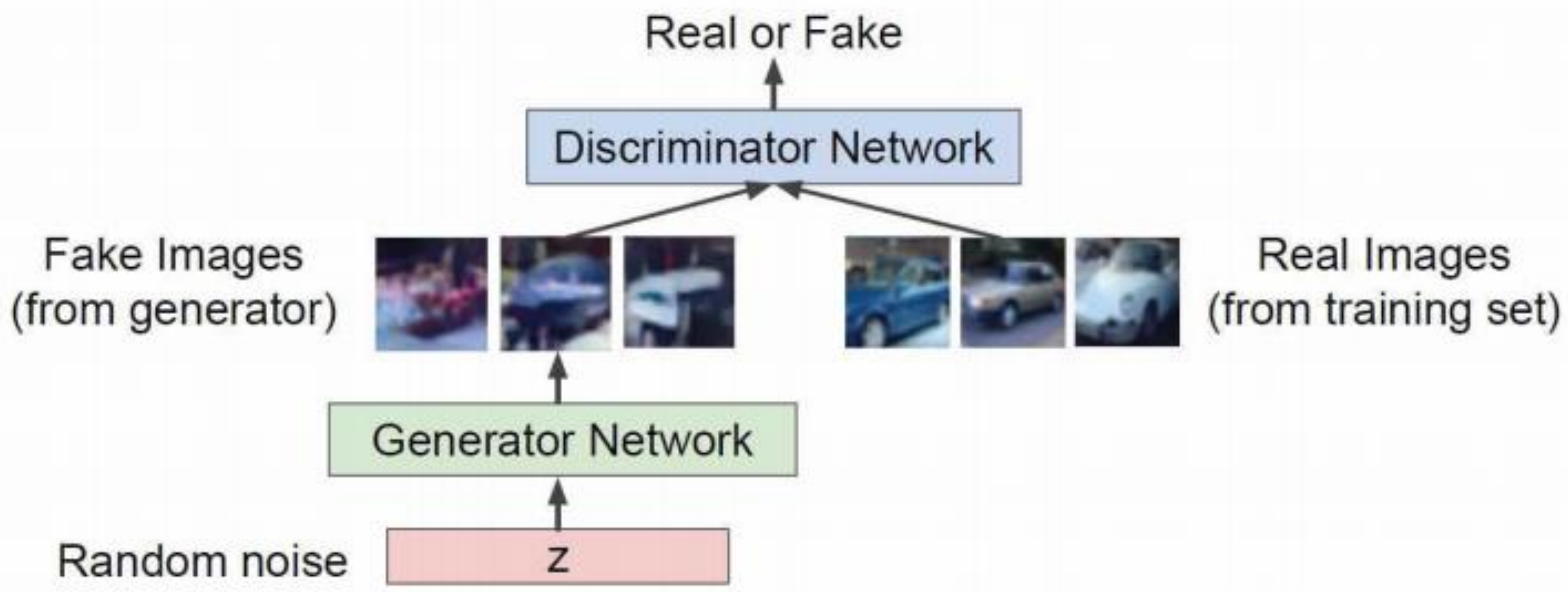
Goal: generate « artificial » but credible examples
credible = sampled from same probability distribution p(x)

Idea: instead of trying to explicitly estimate p(x),

1. **LEARN a transformation G from a simple and known distribution (e.g. random) into X,**
2. **then sampling z → generate realistic samples G(z)**

GAN's architecture

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



(Gaussian/Uniform).
Z ~ latent representation of the image.

GAN training: minimax two-player game!

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

It is formulated as a **minimax game**, where:

- The Discriminator is trying to maximize its reward $\boldsymbol{V(D, G)}$
- The Generator is trying to minimize Discriminator's reward (or maximize its loss)

$$V(D, G) = \boxed{\mathbb{E}_{x \sim p(x)}[\log D(x)]} + \boxed{\mathbb{E}_{z \sim q(z)}[\log(1 - D(G(z)))]}$$

Joint training of D and G

The Nash equilibrium of this particular game is achieved at:

- $\boxed{P_{data}(x) = P_{gen}(x) \;\; \forall x}$
- $\mathrm{D}(x) = \frac{1}{2} \;\; \forall x$

GAN training detail

In practice, alternate Discriminator training (gradient *ascent*) and Generator training:

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

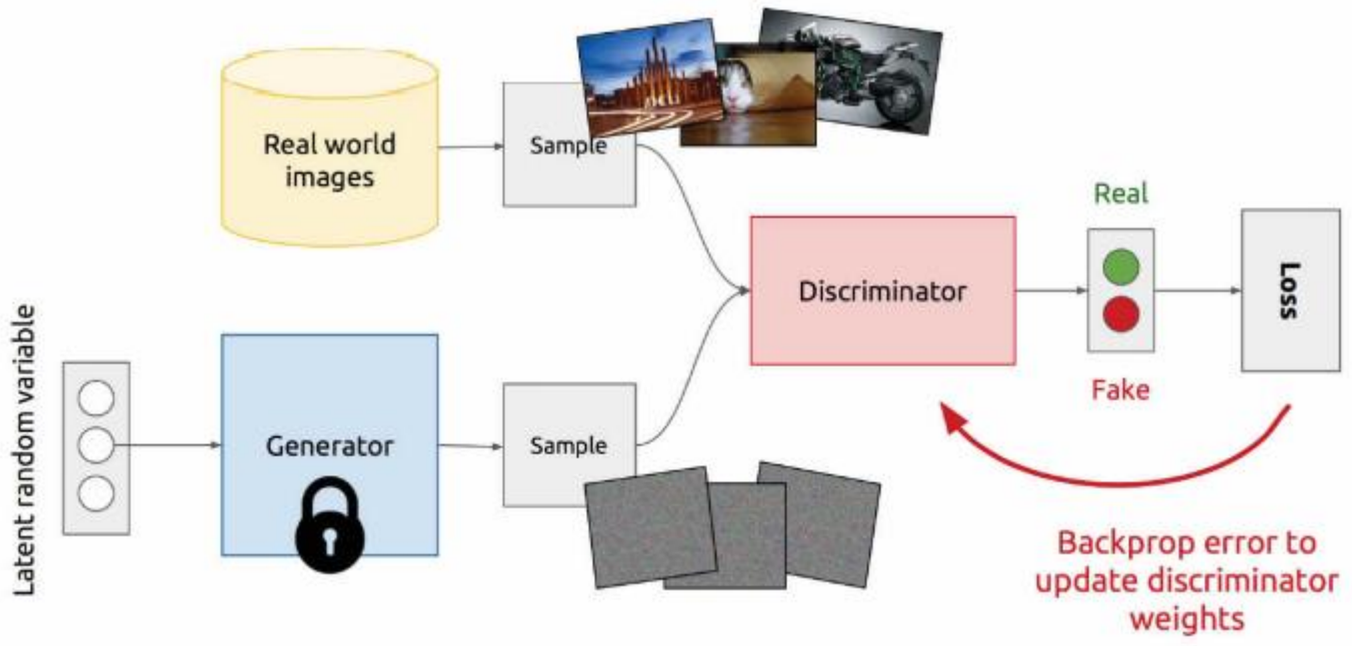
$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$

end for

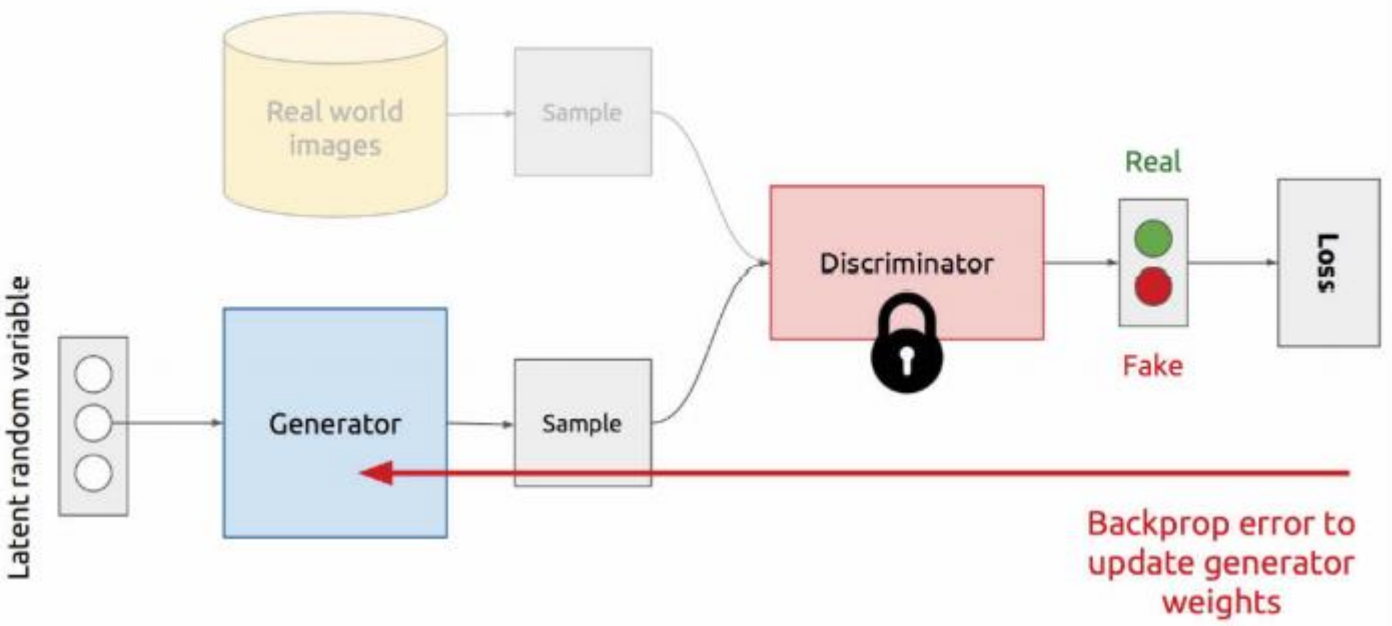
- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by ascending its stochastic gradient (improved objective):

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

end for

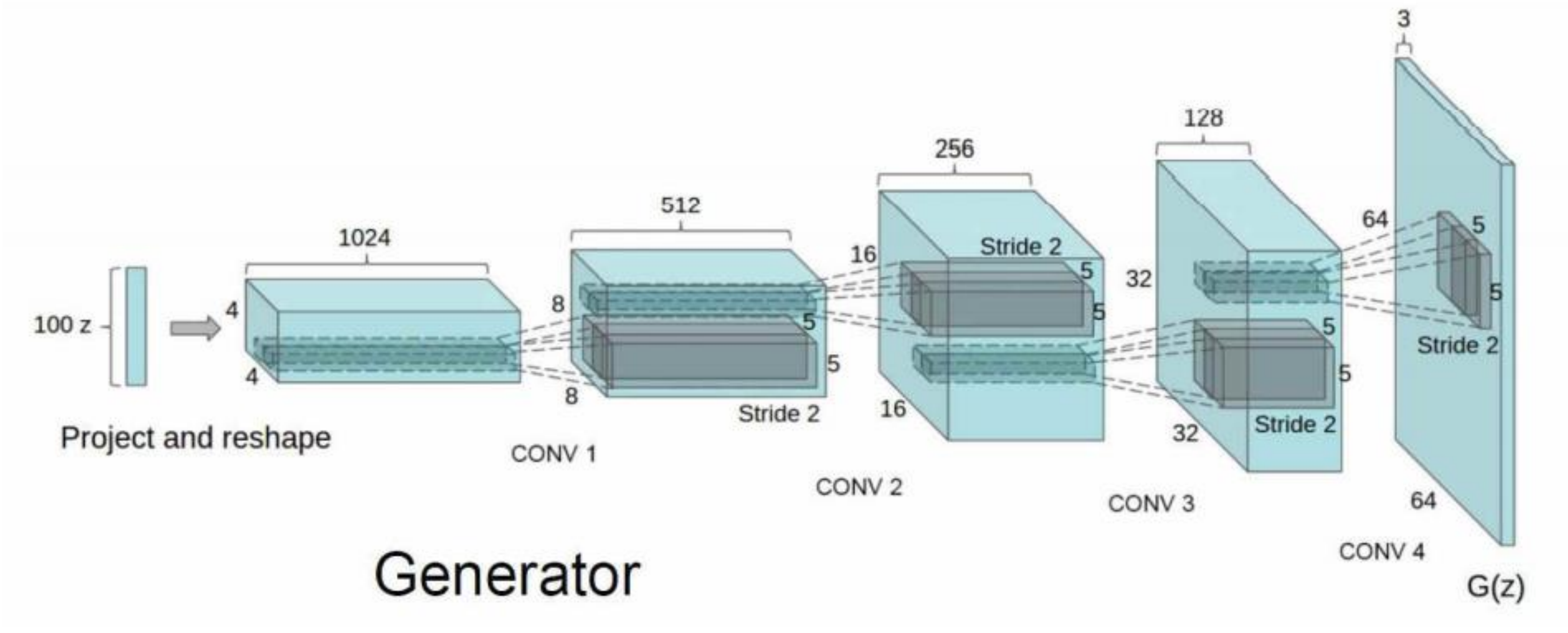


https://www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016



DCGANs

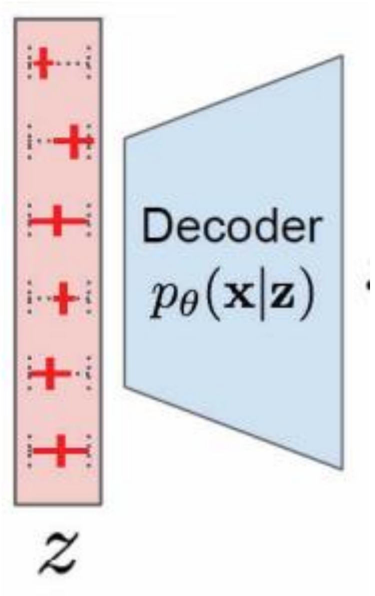
- Deep Convolutional Generative Adversarial Networks



Fake images generated by DCGANs

Trajectory in latent space

- Walking in the latent space (z) can help us understand the landscape of it as well as to reason if the model has learned relevant and interesting representations



Sample code

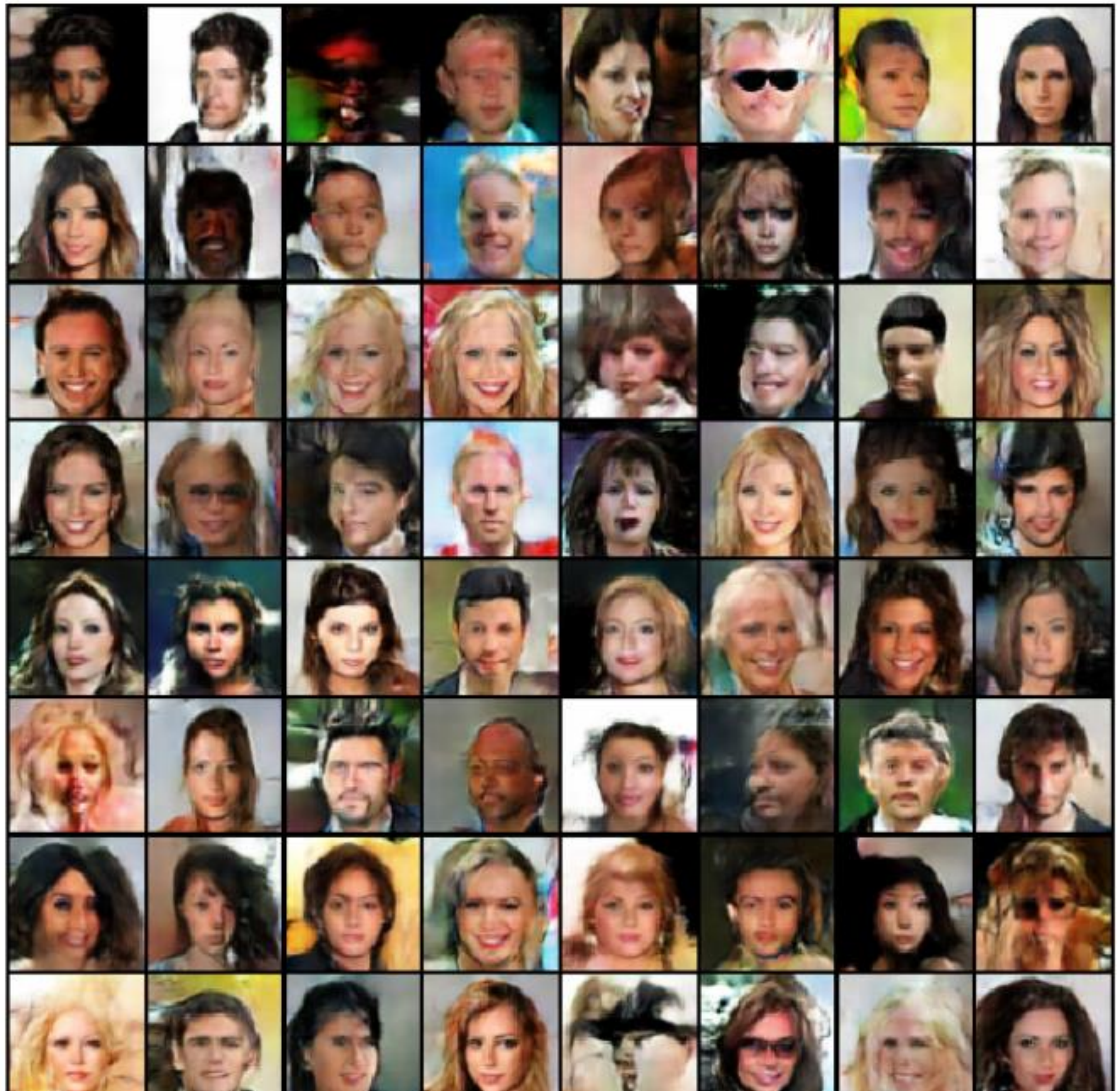
```python
class Generator(nn.Module):
    def __init__(self, ngpu):
        super(Generator, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(
            # input is Z, going into a convolution
            nn.ConvTranspose2d( nz, ngf * 8, 4, 1, 0, bias=False),
            nn.BatchNorm2d(ngf * 8),
            nn.ReLU(True),
            # state size. (ngf*8) x 4 x 4
            nn.ConvTranspose2d(ngf * 8, ngf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf * 4),
            nn.ReLU(True),
            # state size. (ngf*4) x 8 x 8
            nn.ConvTranspose2d( ngf * 4, ngf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf * 2),
            nn.ReLU(True),
            # state size. (ngf*2) x 16 x 16
            nn.ConvTranspose2d( ngf * 2, ngf, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf),
            nn.ReLU(True),
            # state size. (ngf) x 32 x 32
            nn.ConvTranspose2d( ngf, nc, 4, 2, 1, bias=False),
            nn.Tanh()
            # state size. (nc) x 64 x 64
        )

    def forward(self, input):
        return self.main(input)
```

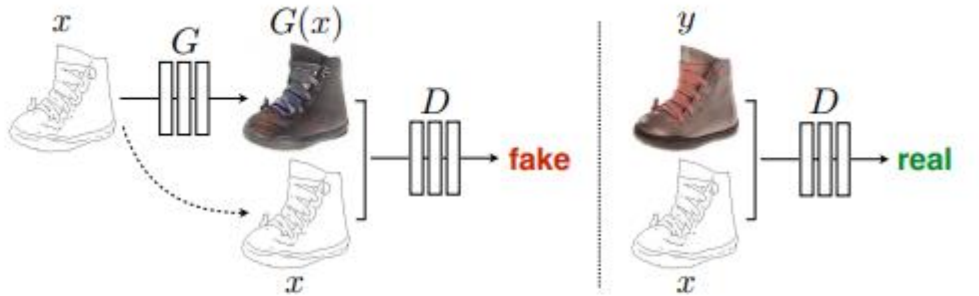
```python
class Discriminator(nn.Module):
    def __init__(self, ngpu):
        super(Discriminator, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(
            # input is (nc) x 64 x 64
            nn.Conv2d(nc, ndf, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf) x 32 x 32
            nn.Conv2d(ndf, ndf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 2),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf*2) x 16 x 16
            nn.Conv2d(ndf * 2, ndf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 4),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf*4) x 8 x 8
            nn.Conv2d(ndf * 4, ndf * 8, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 8),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf*8) x 4 x 4
            nn.Conv2d(ndf * 8, 1, 4, 1, 0, bias=False),
            nn.Sigmoid()
        )

    def forward(self, input):
        return self.main(input)
```

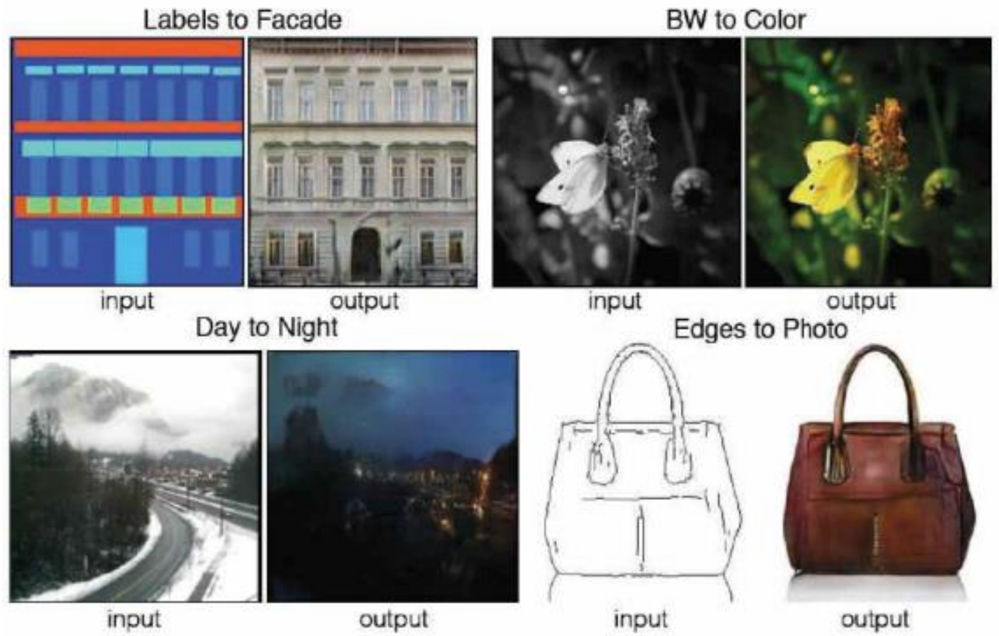
Result of sample code (iteration 8000)



Conditional GAN



- Training a conditional GAN to map edges→photo. The discriminator, D, learns to classify between fake (synthesized by the generator) and real {edge, photo} tuples. The generator, G, learns to fool the discriminator. Unlike an unconditional GAN, both the generator and discriminator observe the input edge map



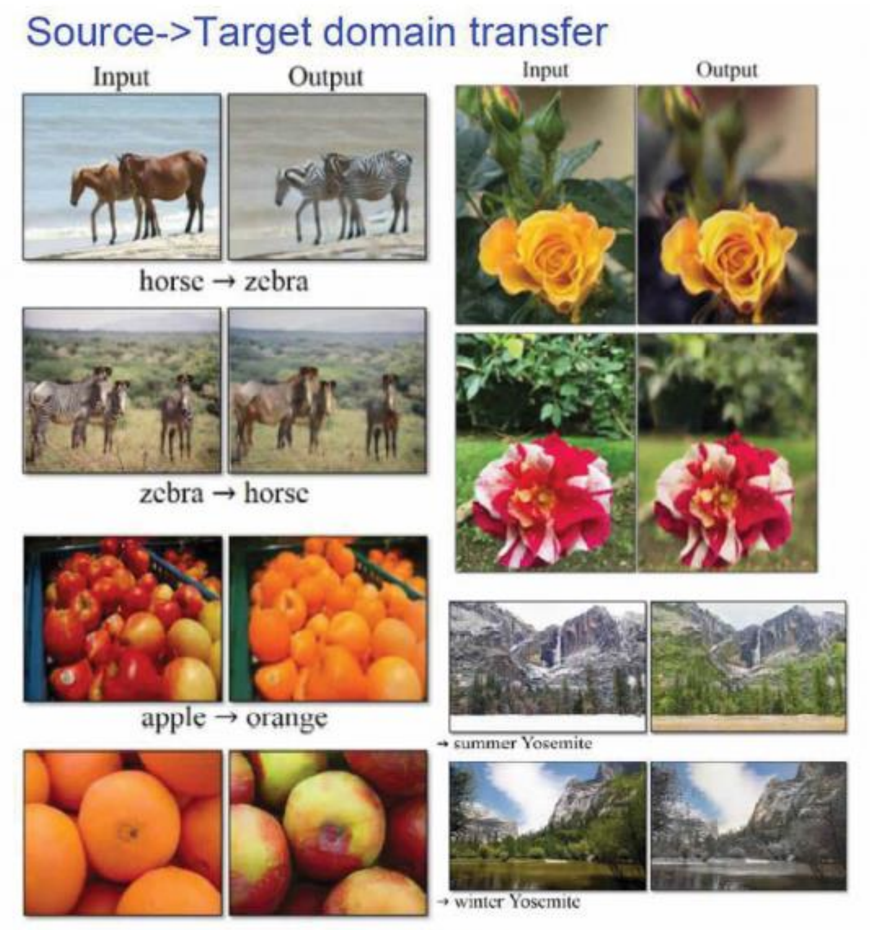
Interactive demo: https://affinelayer.com/pixsrv/

Isola, Phillip, et al. "Image-to-image translation with conditional adversarial networks." 2017.

Video2video Synthesis

"Video-to-Video Synthesis", NeurIPS'2018 [Nvidia+MIT]
Using Generative Adversarial Network (GAN)

CycleGAN

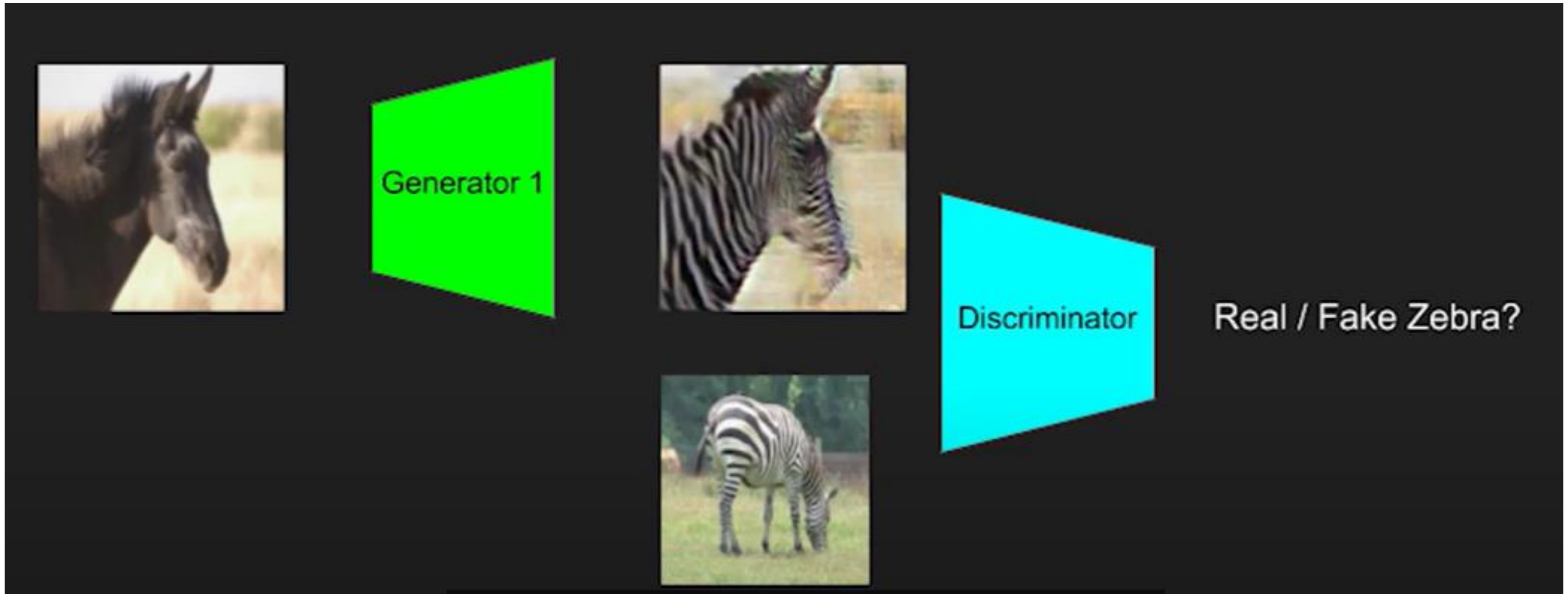


CycleGAN. Zhu et al. 2017.

- CycleGAN is proposed in 2017 (Jun-Yan Zhu, Taesuang Park, et al.) to deal with task with Unpaired Image-to-Image Translation
- The architecture use two generators and on discriminator.
- The Objectives are
  - Ensure the translated image looks like zebra
    - This is trained using the GAN objective with the discriminator
  - Ensure the translated image still looks mostly like the original
    - This is trained using a reconstruction objective with the second generator
    - This is the novel cycle-consistency loss

CycleGAN


Generator 1
Generator 2
Mean Squared Error


Generator 1
Discriminator
Real / Fake Zebra?

"The GAN Zoo"

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

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

GANs short summary

- 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

Summary

- Intrinsically UNSUPERVISED
  - can be used on UNLABELLED DATA
- Impressive results in Image Retrieval
- DBN/DBM/VAE = Generative probabilistic models
- GAN = most promising generative model, with already many remarkable & exciting applications
- Strong potential for enhancement of datasets and for ultra-realistic synthetic data
- Interest for "creative« /artistic computing?

Any QUESTIONS ?