Deep-Learning:

Unsupervised Generative models

Deep Belief Networks
Deep Stacked AutoEncoders
Generative Adversarial Networks

Pr. Fabien MOUTARDE
Center for Robotics
MINES ParisTech
PSL Université Paris

`Fabien.Moutarde@mines-paristech.fr`
`http://people.mines-paristech.fr/fabien.moutarde`

Acknowledgements

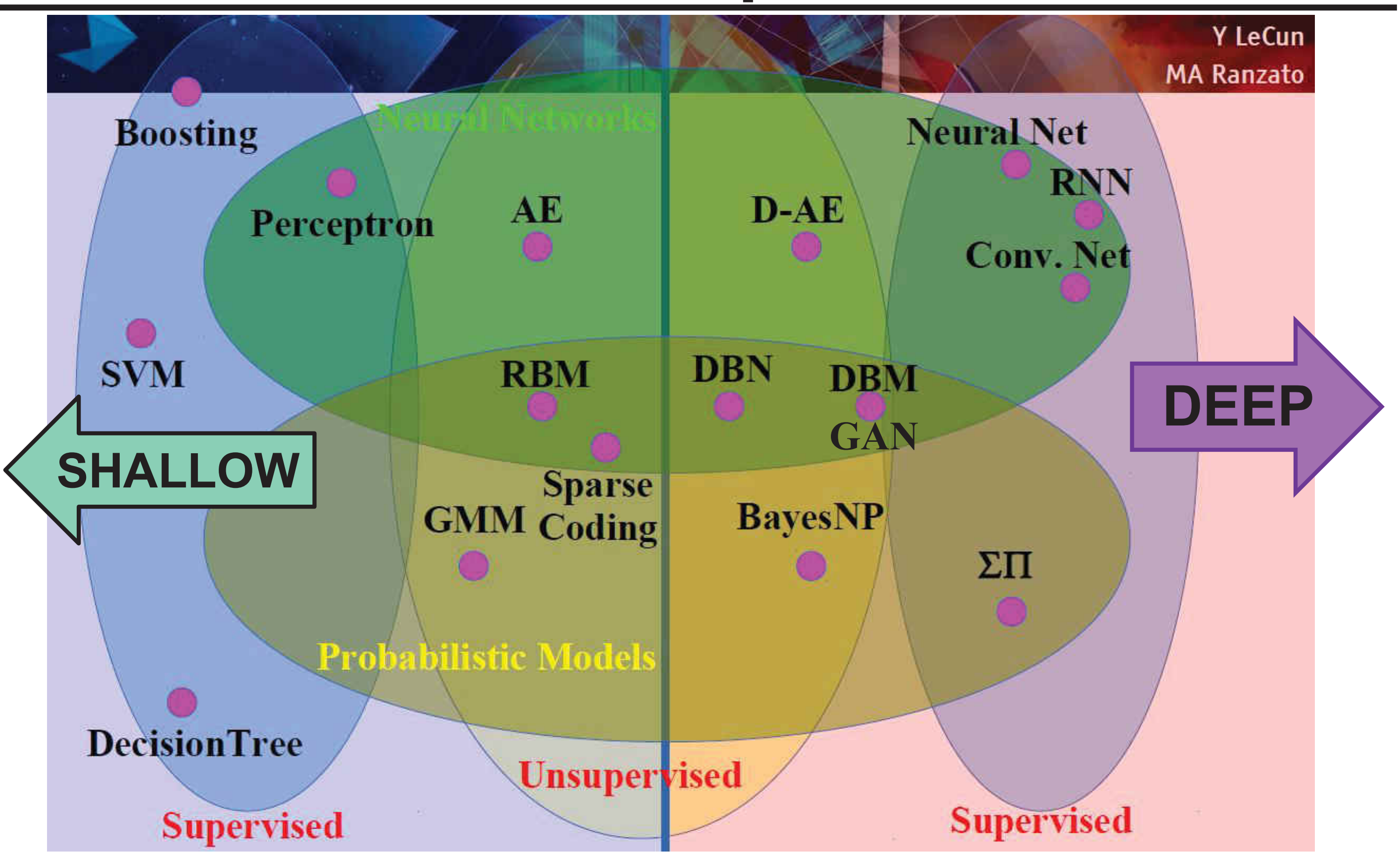
During preparation of these slides, I got inspiration and borrowed some slide content from several sources, in particular:

- Fei-Fei Li & J. Johnson & S. Yeung: course on Generative Models http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf
- I. Kokkinos: slides of a CentraleParis course on Deep Belief Networks http://cvn.ecp.fr/personnel/iasonas/course/DL5.pdf
- I. Goodfellow: NIPS'2016 tutorial on Generative Adversarial Networks (GANs) https://media.nips.cc/Conferences/2016/Slides/6202-Slides.pdf
- Binglin, Shashank & Bhargav: A short tutorial on Generative Adversarial Networks (GANs) http://slazebni.cs.illinois.edu/spring17/lec11_gan.pdf

Outline

- <u>Unsupervised Learning and Generative Models</u>
- Deep Belief Networks (DBN)
  and Deep Boltzman Machine (DBM)
- Autoencoders
- Generative Adversarial Networks (GAN)

Deep vs Shallow Learning techniques overview



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 : data, z : **latent**

Generative **models**

Find generation function ***g*** : $x = g(z)$, x : data, z : **latent**

[0.1, 0.3, -0.8, 0.4, …] → g → 

Unsupervised learning vs. Generative model

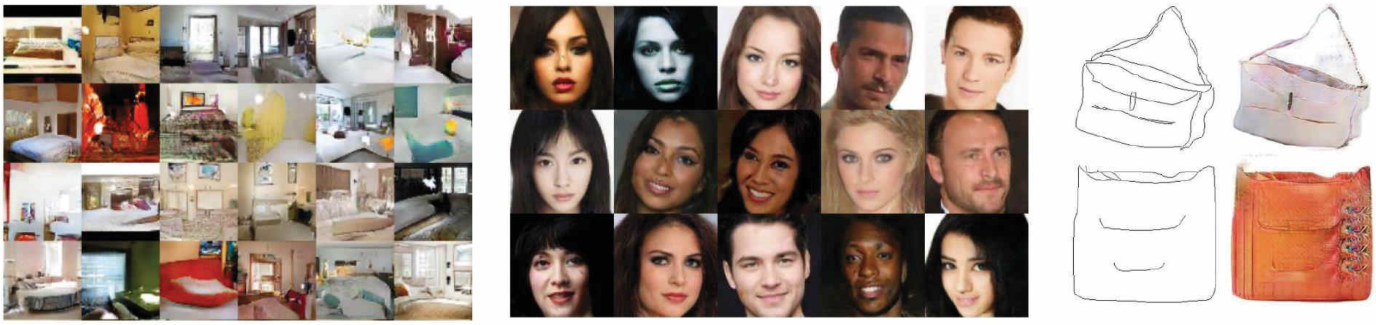
- ***z = f(x)*** vs. ***x = g(z)***
- ***P(z|x)*** vs. ***P(x|z)***
- ***Encoder*** vs. ***Decoder (Generator)***
  - P(x, z) needed. (cf : P(y|x) in supervised learning)
    - P(z|x) = P(x, z) / P(x)
    - P(x|z) = P(x, z) / P(z) → P(z) is given. (prior)

Why Generative?

- Conditional generative models
  - Speech synthesis: Text ⇒ Speech
  - Machine Translation: French ⇒ English
    - French: Si mon tonton tond ton tonton, ton tonton sera tondu.
    - English: If my uncle shaves your uncle, your uncle will be shaved
  - Image ⇒ Image segmentation
- Environment simulator
  - Reinforcement learning
  - Planning
- Leverage unlabeled data

Why generative?

- 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

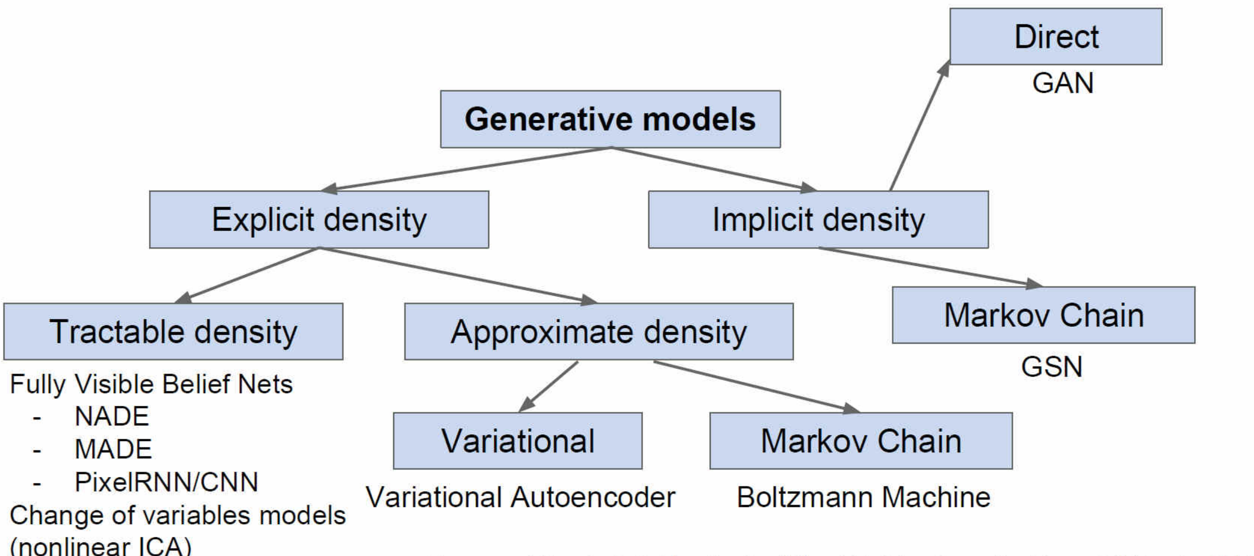


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

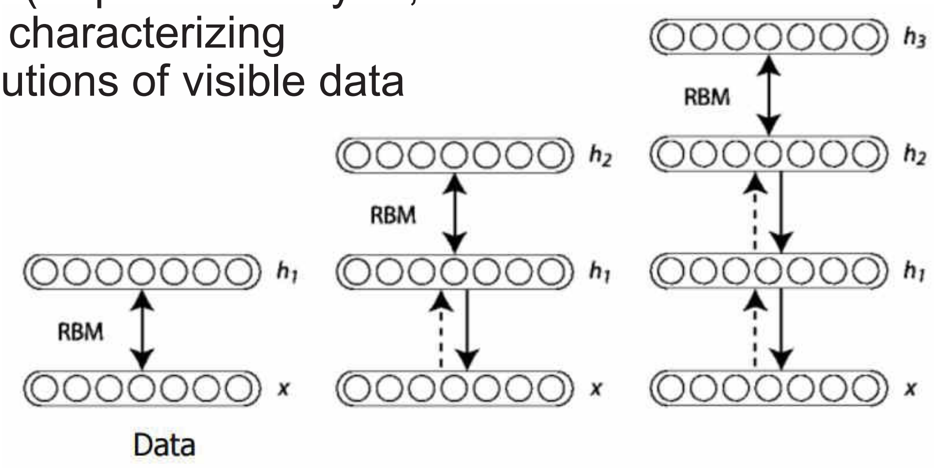
Outline

- Unsupervised Learning and Generative Models
- <u>Deep Belief Networks (DBN)
and Deep Boltzman Machine (DBM)</u>
- Autoencoders
- Generative Adversarial Networks (GAN)

Deep Belief Networks (DBN)

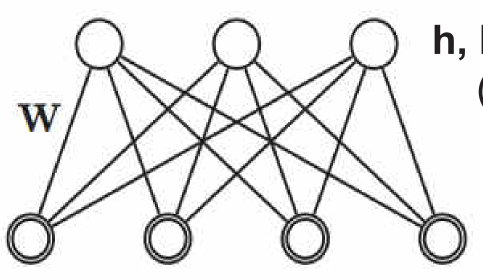
- **One of first Deep-Learning models**
- **Proposed by G. Hinton in 2006**
- **Generative probabilistic model (mostly UNSUPERVISED)**

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



Greedy successive UNSUPERVISED learning of layers of Restricted Boltzmann Machine (RBM)

Restricted Boltzmann Machine (RBM)



h, hidden
(~ latent variables)

v, observed

NB: connections are BI-DIRECTIONAL (with same weight)

Modelling probability distribution as:

$$P(\mathbf{v}, \mathbf{h}; \theta) = \frac{\exp(-E(\mathbf{v}, \mathbf{h}; \theta))}{\sum_{\mathbf{v},\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h}; \theta))}$$

with ***« Energy »*** E given by

$$\begin{aligned} E(\mathbf{v}, \mathbf{h}; \theta) &= -\mathbf{v}^\top W\mathbf{h} - \mathbf{b}^\top \mathbf{v} - \mathbf{a}^\top \mathbf{h} \\ &= -\sum_{i=1}^{D}\sum_{j=1}^{F} W_{ij} v_i h_j - \sum_{i=1}^{D} b_i v_i - \sum_{j=1}^{F} a_j h_j \end{aligned}$$

Training RBM

Finding θ=(W,a,b) **maximizing likelihood** $\prod_{v \in S} p_\theta(v)$ **of dataset S**

⇔ **minimize NegLogLikelihood** $-\sum_{v \in S} \log(p_\theta(v))$

Independance within layers ➔ $p(v|h) = \prod_i p(v_i|h)$ **and** $p(h|v) = \prod_j p(h_j|v)$

So objective = find $\boxed{\theta_* = \underset{\theta}{\mathrm{argMin}} \left(-\sum_{v \in S} \sum_j \log(p_\theta(v_j)) \right)}$

In *binary* input case:

$$p(v_i = 1 \mid h) = \sigma(a_i + W_{:,i} h)$$
$$p(h_j = 1 \mid v) = \sigma(b_j + W_{j,:} v)$$

with $\sigma(u) = \dfrac{e^u}{e^u + 1}$

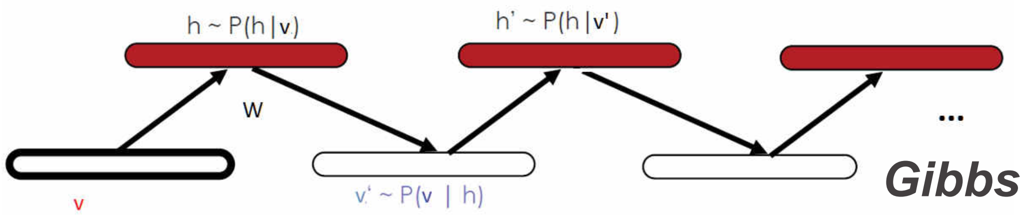
Algo: Contrastive Divergence

≈ Gibbs sampling used inside a gradient descent procedure

Contrastive Divergence algo

Repeat:

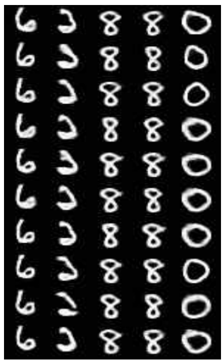
1. Take a training sample *v*, compute $p(h_j = 1 \mid v) = \sigma(b_j + W_{j,:}v)$ and sample a vector *h* from this probability distribution
2. Compute *positive* gradient as outer product $G_+ = v \otimes h = vh^T$
3. From *h*, compute $p(v'_i = 1 \mid h) = \sigma(a_i + W_{:,i}h)$ and sample reconstructed *v'*, then resample *h'* using $p(h'_j = 1 \mid v') = \sigma(b_j + W_{j,:}v')$
   [Gibbs sampling single step; should theoretically be repeated until convergence]
4. Compute *negative* gradient as outer product $G_- = v' \otimes h' = v'h'^T$
5. Update weight matrix by $\delta W = \varepsilon(G_+ - G_-) = \varepsilon(vh^T - v'h'^T)$
6. Update biases *a* and *b* analogously: $\delta a = \varepsilon(v - v')$ and $\delta b = \varepsilon(h - h')$

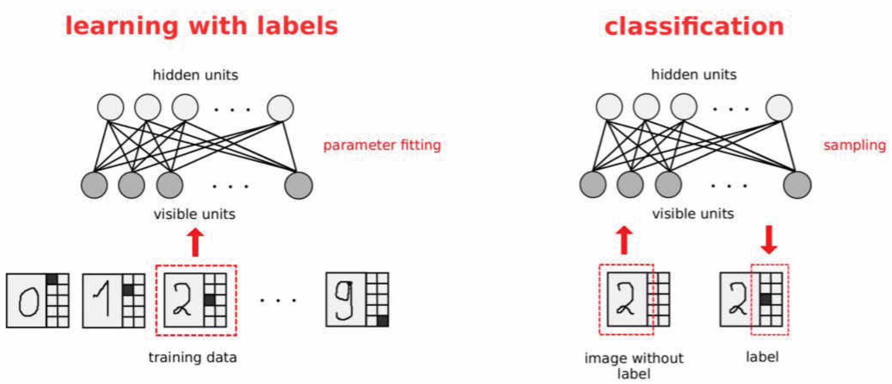


Gibbs sampling

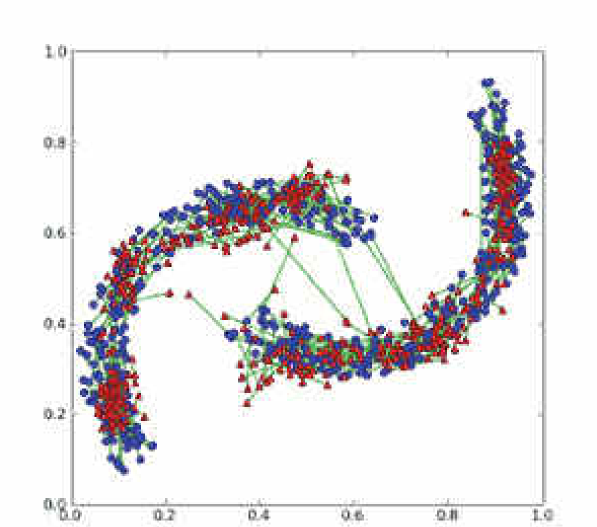
Use of trained RBM

- 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

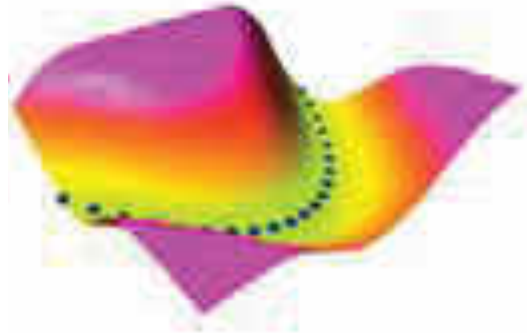




Modeling of input data distribution from trained RBM



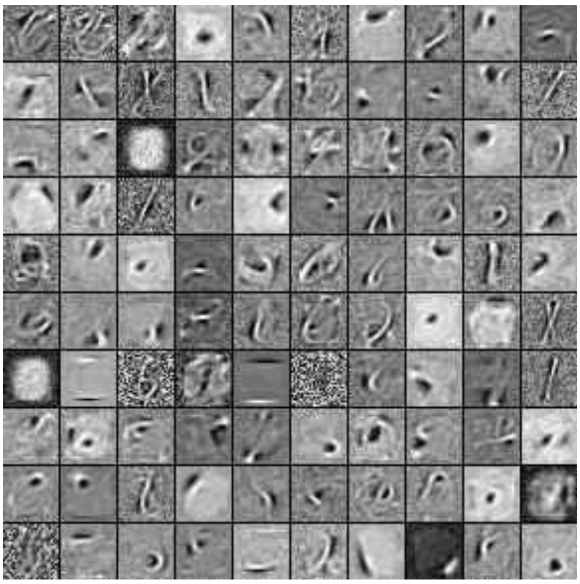
Initial data is in blue, reconstructed in red (and green line connects each data point with reconstructed one).

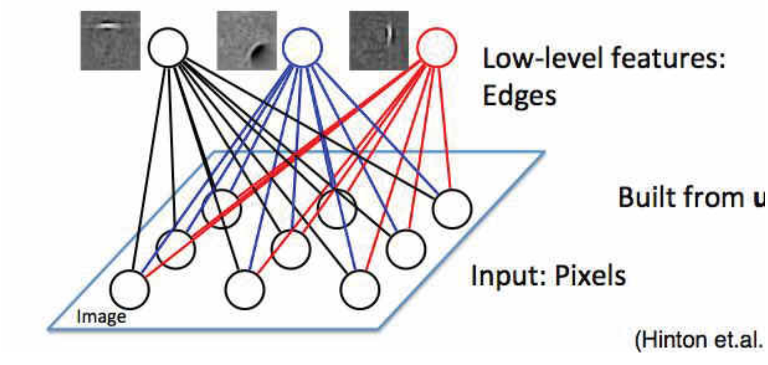


Learnt energy function:
<u>minima</u> created <u>where data points are</u>

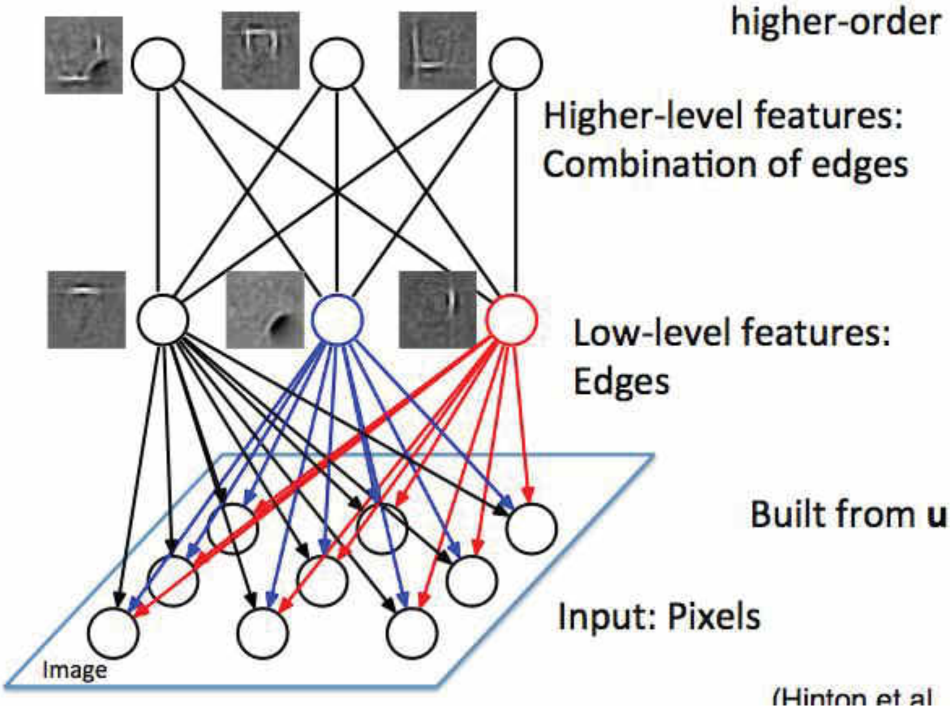
Interpretation of trained RBM hidden layer

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





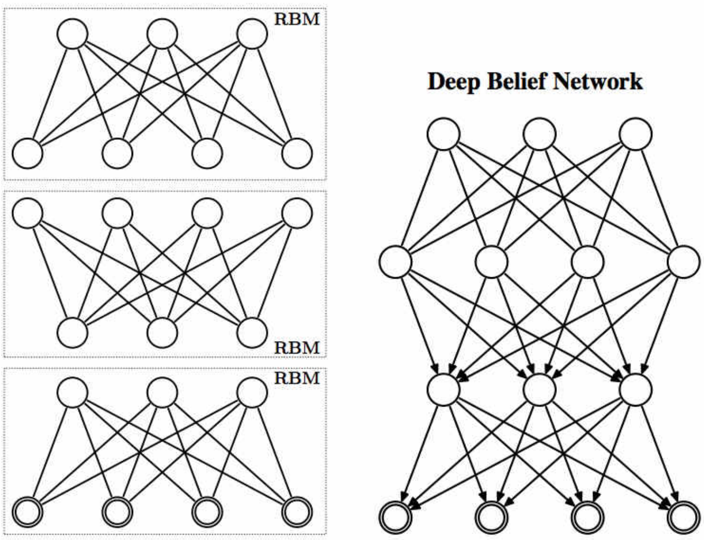
Why go deeper with DBN ?



DBN: upper layers → more « abstract » features

Learning of DBN

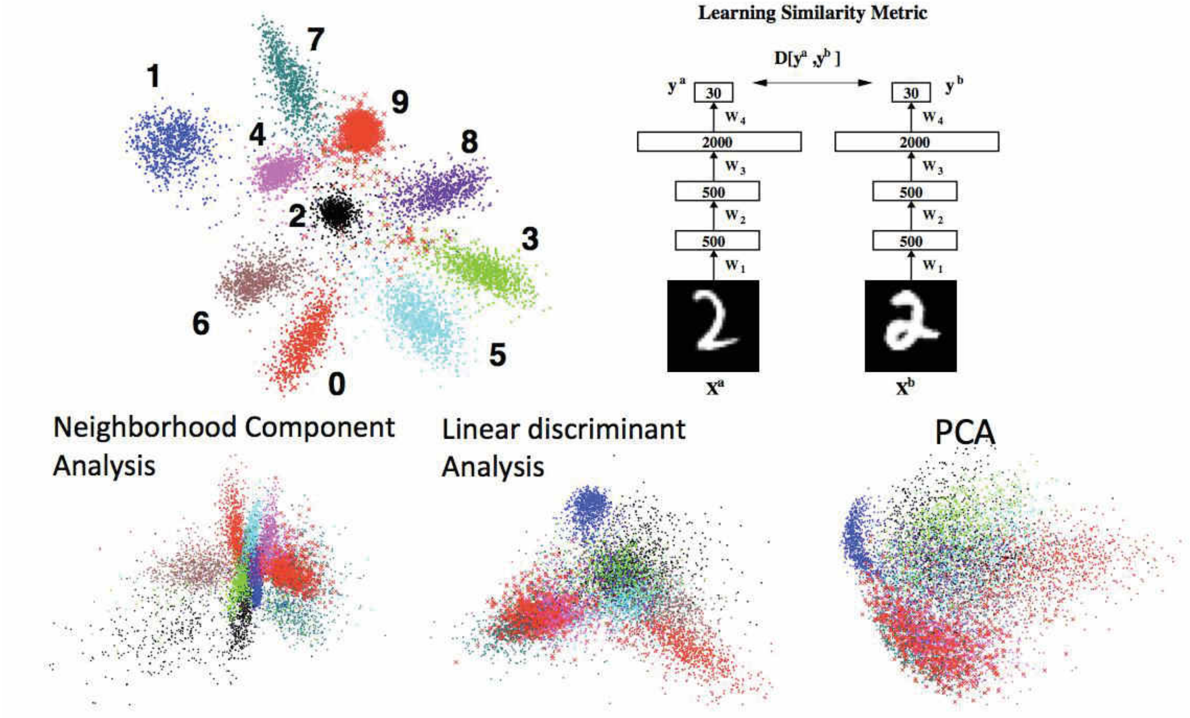
Greedy learning of successive layers



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.

Using low-dim final features for clustering



Much better results than clustering in input space or using other dimension reduction (PCA, etc…)

Example application of DBN: Clustering of documents in database

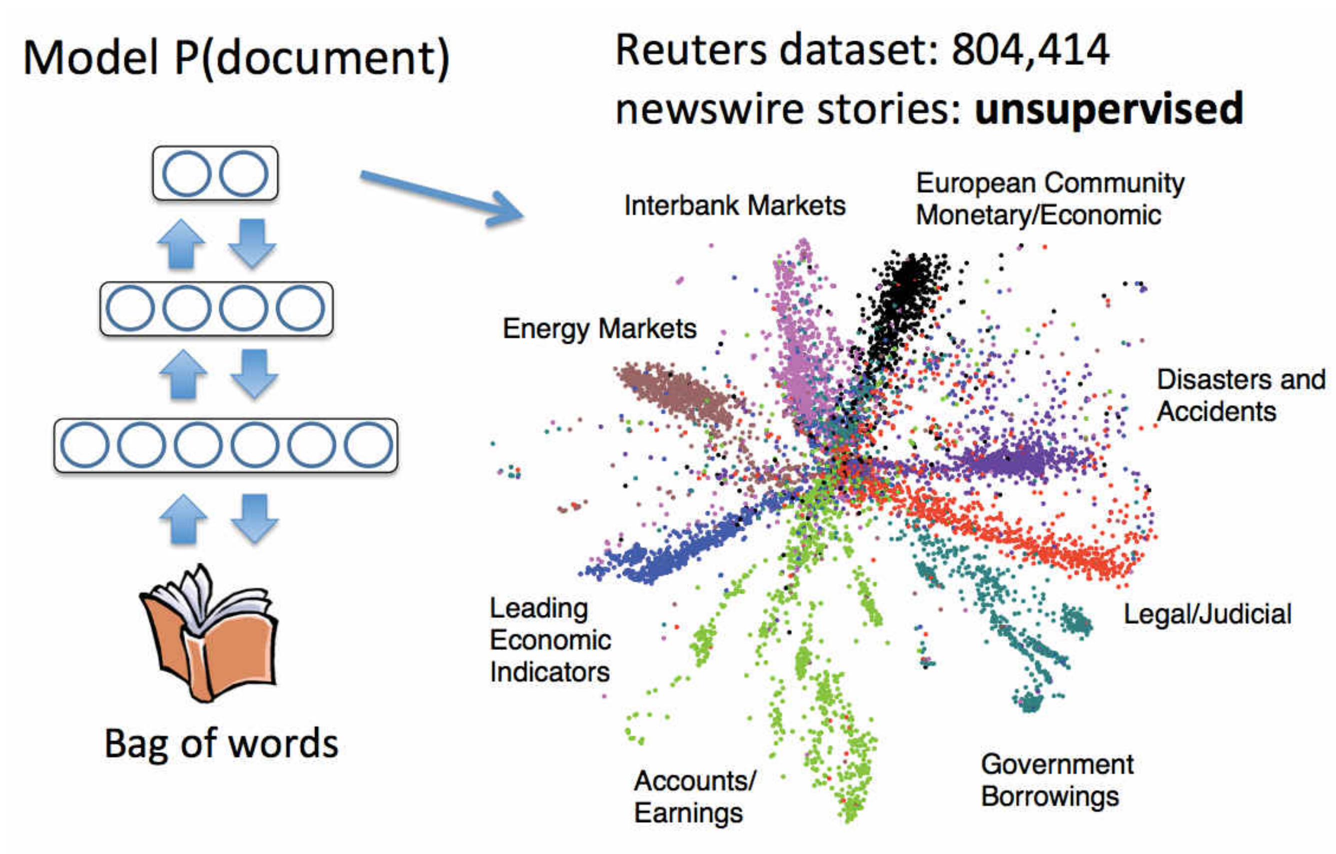
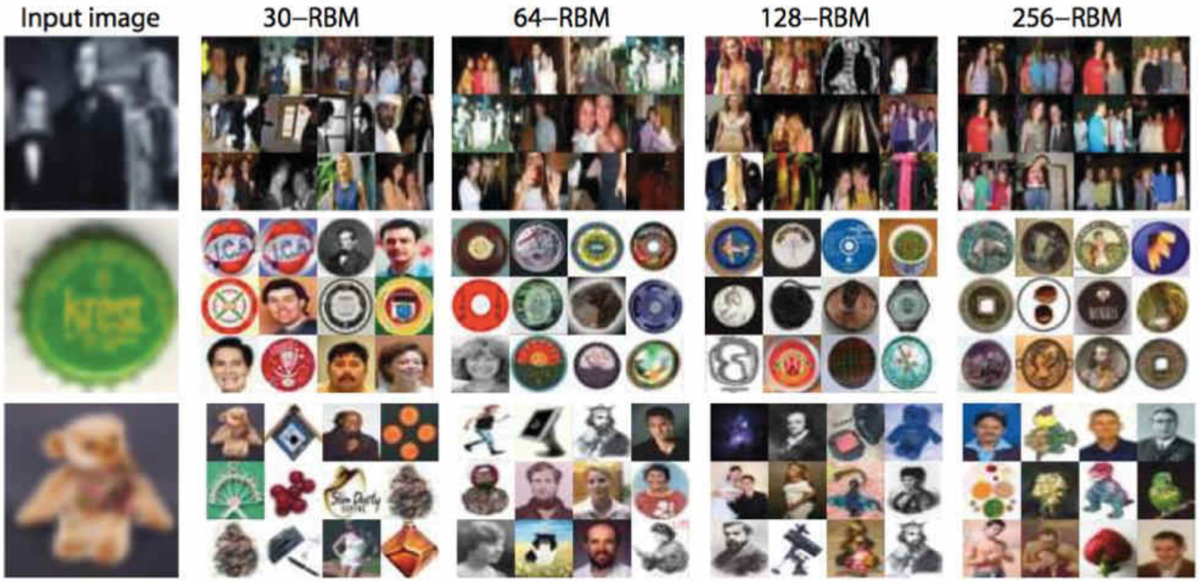


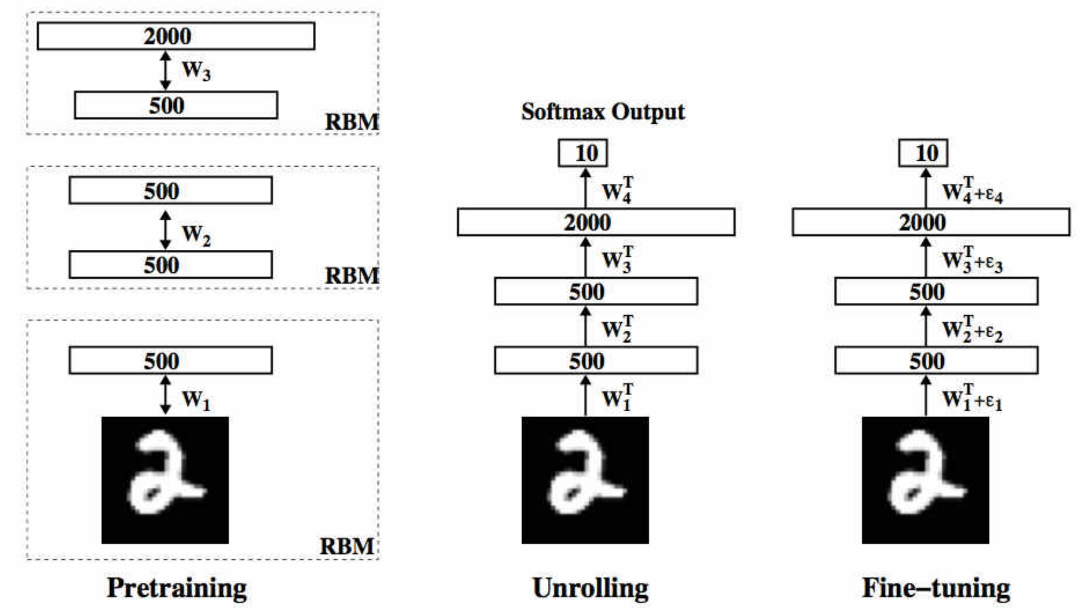
Image Retrieval application example of DBN

- Map images into binary codes for fast retrieval.



- Small Codes, Torralba, Fergus, Weiss, CVPR 2008
- Spectral Hashing, Y. Weiss, A. Torralba, R. Fergus, NIPS 2008
- Kulis and Darrell, NIPS 2009, Gong and Lazebnik, CVPR 20111
- Norouzi and Fleet, ICML 2011,

DBN supervised tuning



UNSUPERVISED

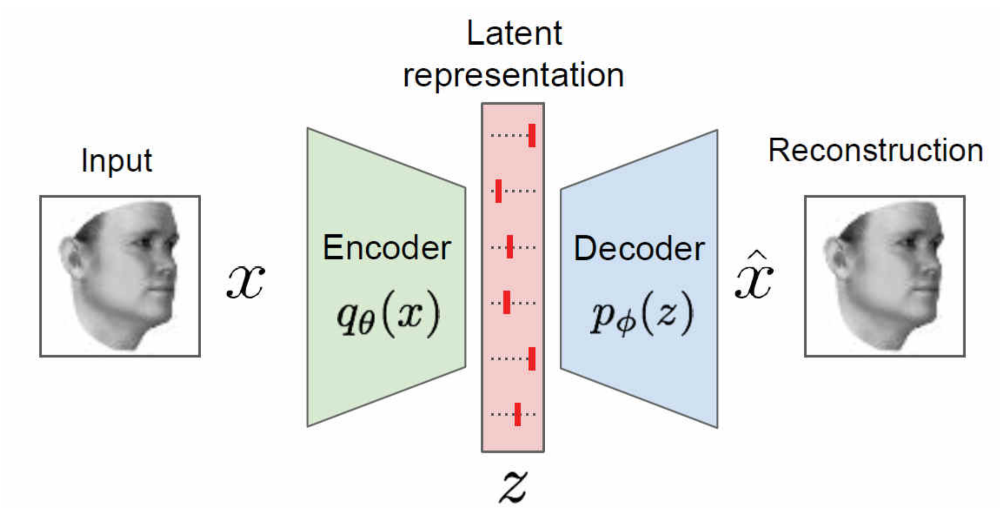
SUPERVISED

- After layer-by-layer **unsupervised pretraining**, discriminative fine-tuning by backpropagation achieves an error rate of 1.2% on MNIST. SVM's get 1.4% and randomly initialized backprop gets 1.6%.

Outline

- Unsupervised Learning and Generative Models
- Deep Belief Networks (DBN)
  and Deep Boltzman Machine (DBM)
- <u>Autoencoders</u>
- Generative Adversarial Networks (GAN)

Autoencoders



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

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

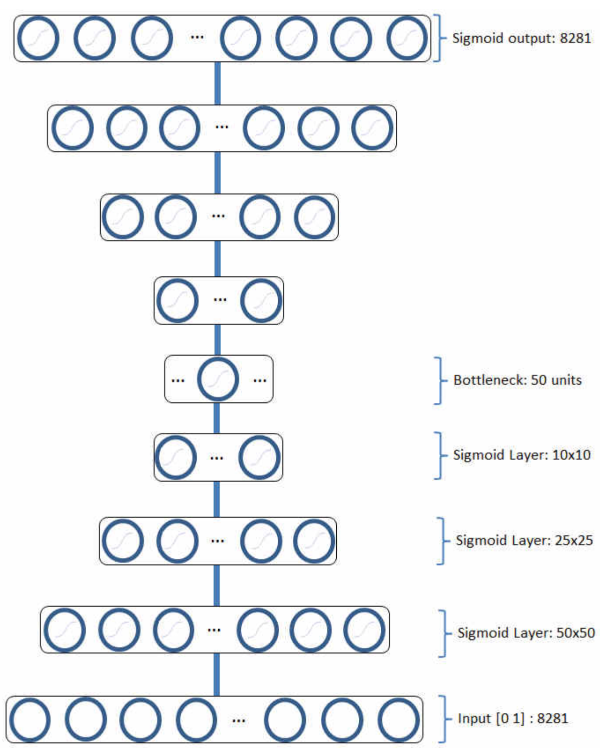
→ unsupervised learning of latent variables, and of a generative model

Variants of autoencoders

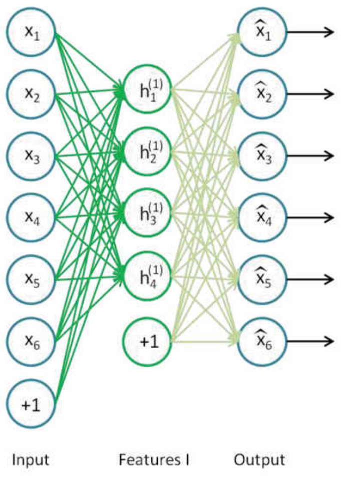
- Denoising autoencoders
- Sparse autoencoders
- Stochastic autoencoders
- Contractive autoencoders
- VARIATIONAL autoencoders
- …

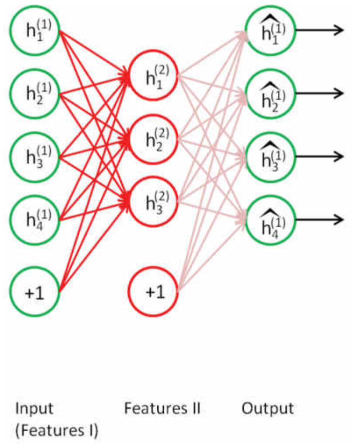
Deep *Stacked* Autoencoders

Proposed by Yoshua Bengio in 2007



Training of Stacked Autoencoers



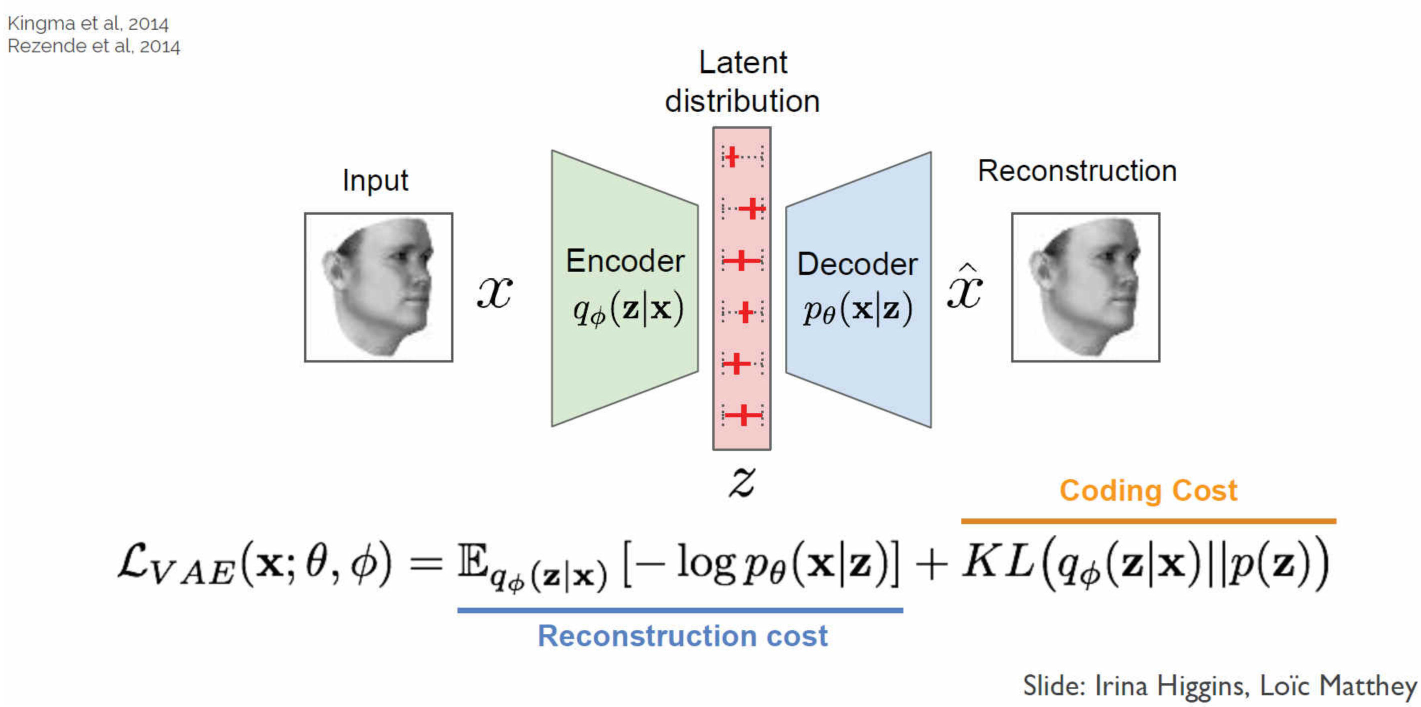


etc…

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"

Variational AutoEncoders (VAE)



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

Outline

- Unsupervised Learning and Generative Models
- Deep Belief Networks (DBN)
- Autoencoders
- Generative Adversarial Networks (GAN)

Generative Adversarial Network

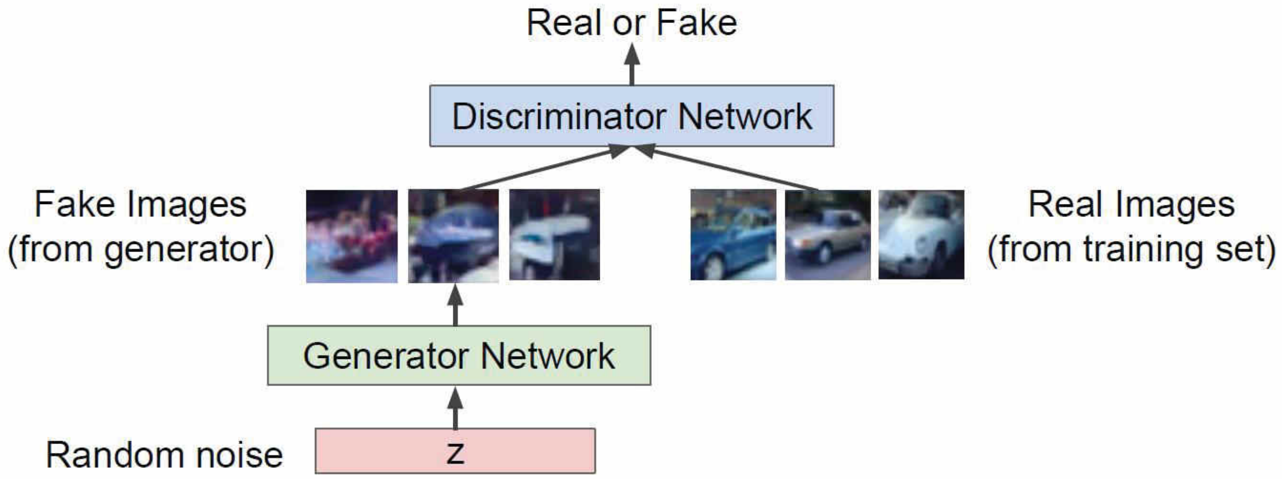
[Introduced in 2014 by Ian Goodfellow et al.
(incl. Yoshua Bengio) from University of Montreal]

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)

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

Joint training of D and G

The Nash equilibrium of this particular game is achieved at:

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

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

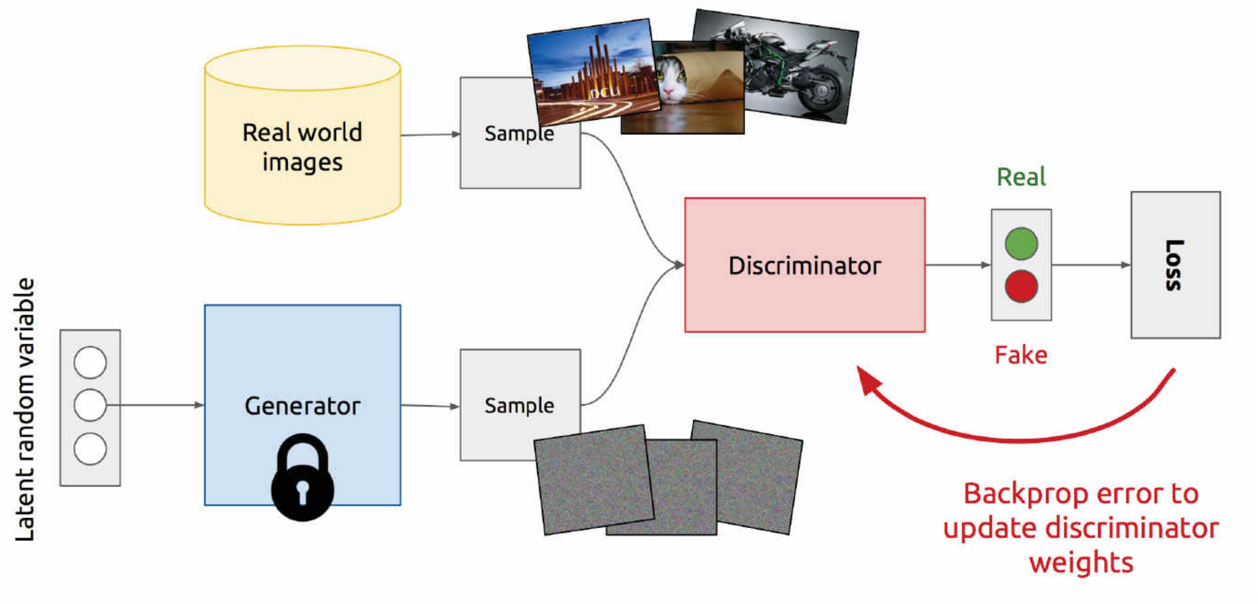
$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$

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

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

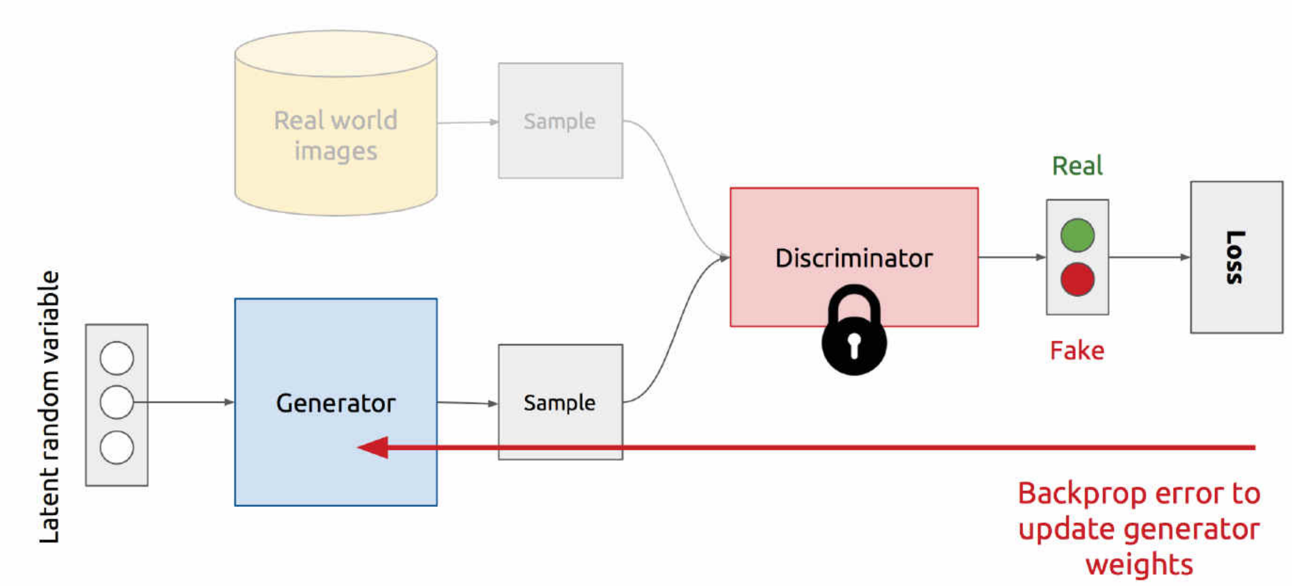
end for

Training the Discriminator

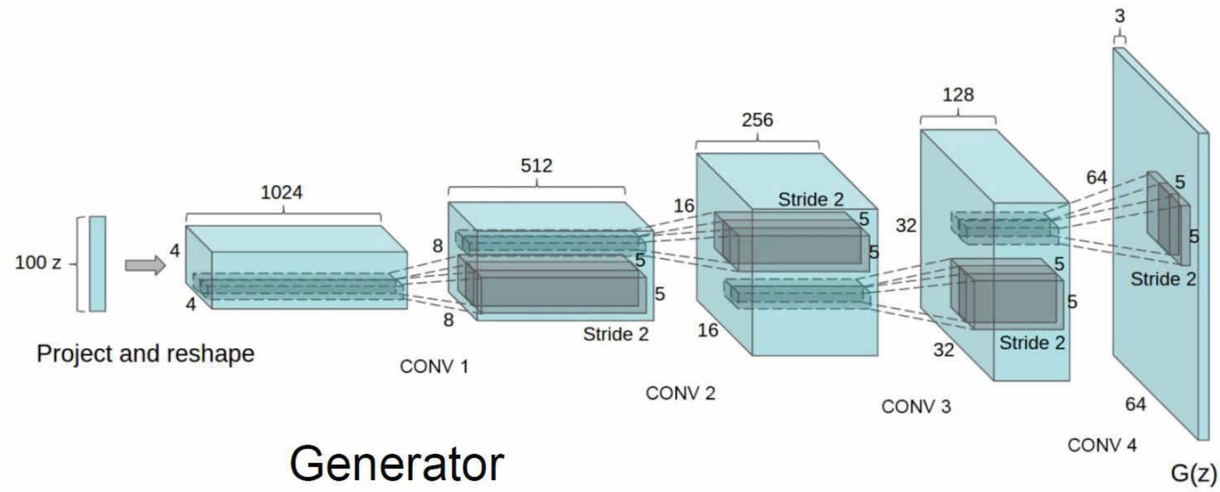


https://www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016

Training the Generator



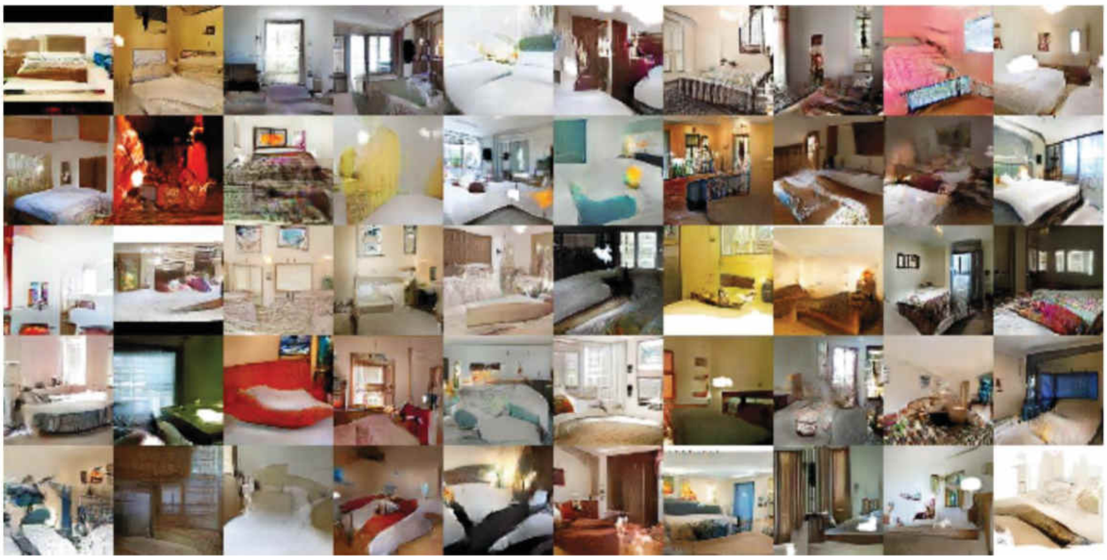
Convolutional Generator for GAN



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

Example of fake samples generated by GAN

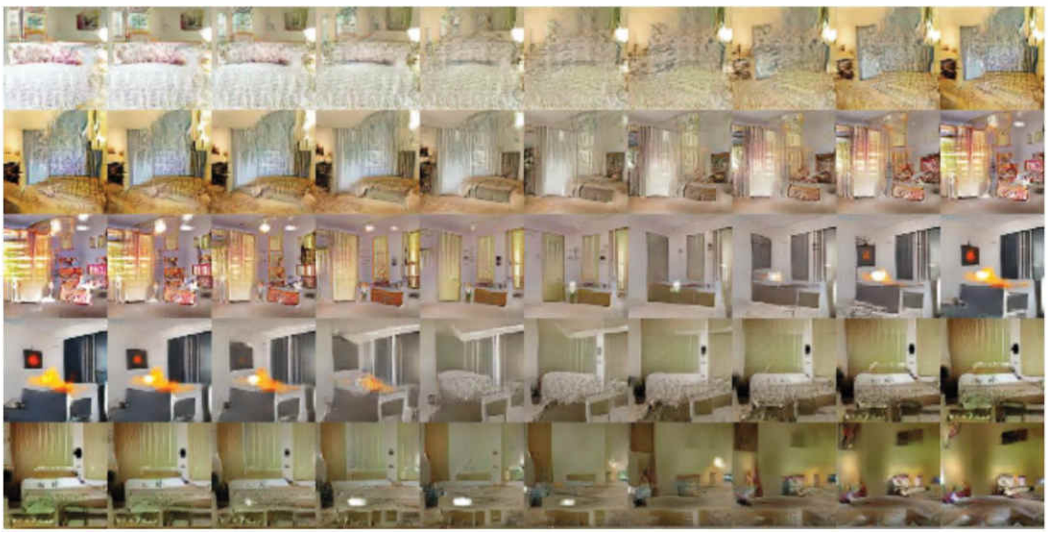
Samples from the model look amazing!



Radford et al, ICLR 2016

Trajectory in latent space → continous image transform

Interpolating between random points in laten space



Radford et al, ICLR 2016

« Arithmetic » of latent vectors

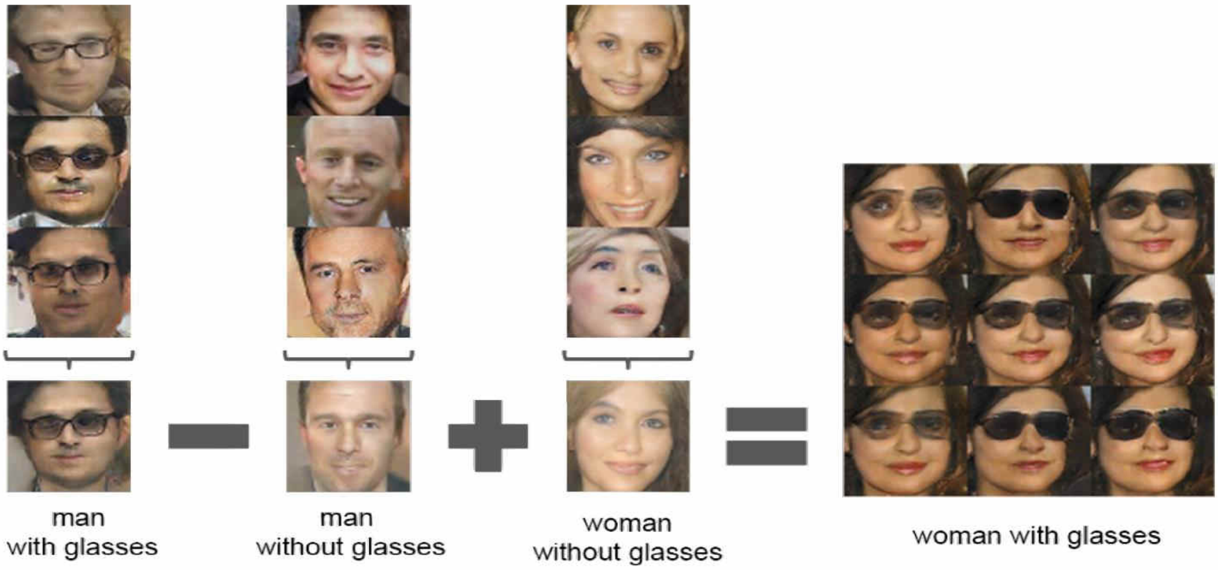


Image-to-Image translation

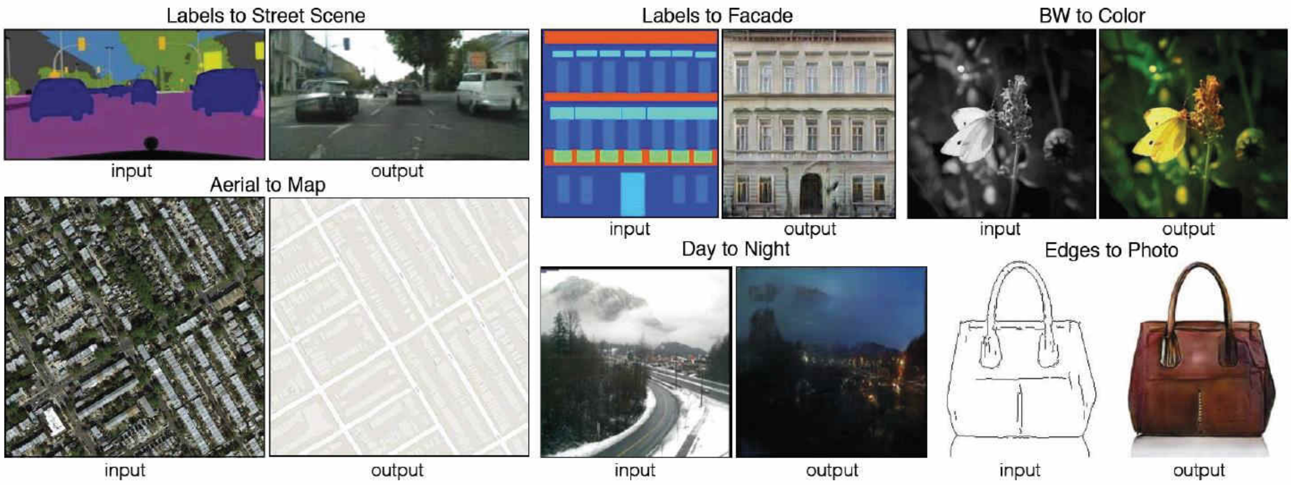


Figure 1 in the original paper.

Link to an interactive demo of this paper

Isola, P., Zhu, J. Y., Zhou, T., & Efros, A. A. "**Image-to-image translation with conditional adversarial networks**". arXiv preprint arXiv:1611.07004. (2016).

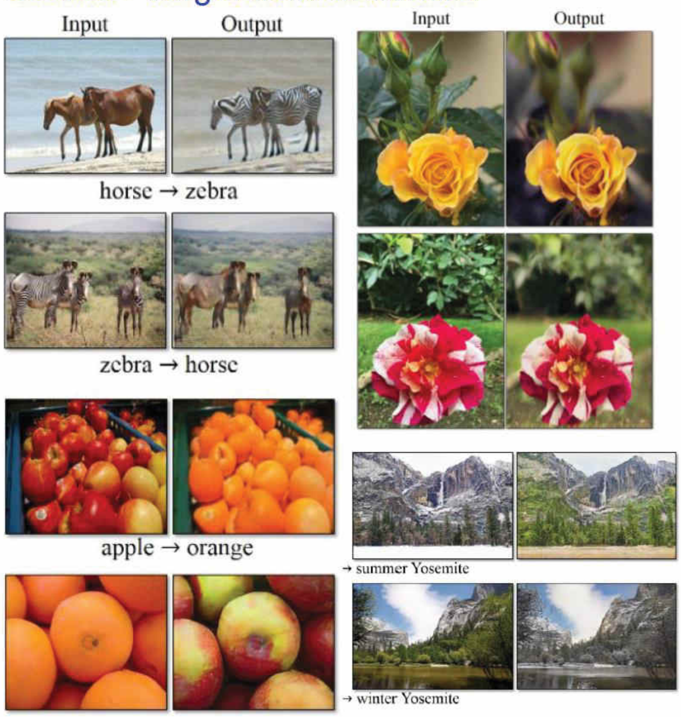
GAN for synthesis of realistic images

"Video-to-Video Synthesis", NeurIPS’2018 [Nvidia+MIT]

Using Generative Adversarial Network (GAN)

Domain transfer!

Source->Target domain transfer



CycleGAN. Zhu et al. 2017.

Summary and perspectives on DBN/DBM/DSA/VAE/GAN

- Intrinsicly 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 ?