Giới thiệu về Mô Hình Học Nhiều Tầng (deep learning models)

Vietnam Institute for Advanced Studies in Mathmeatics
VIASM, Data Science 2017, FIRST

Few Useful Things to Know About Deep Learning

Phùng Quốc Định
Centre for Pattern Recognition and Data Analytics (PRaDA)
Deakin University, Australia
Email: dinh.phung@deakin.edu.au
(published under **Dinh Phung**)

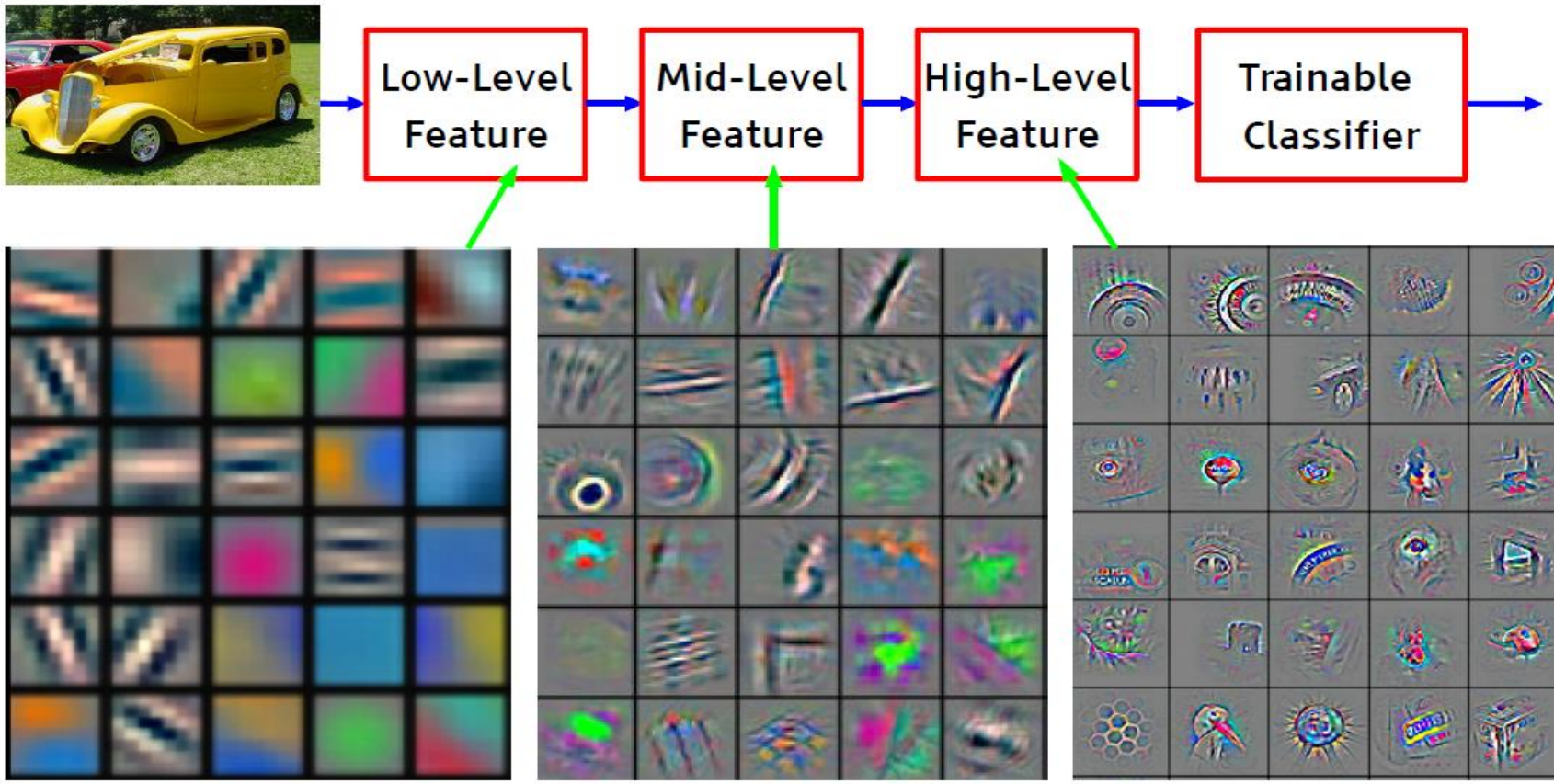
Deep Learning

"Deep Learning: machine learning algorithms based on learning multiple levels of representation and abstraction" - Joshua Bengio

Học nhiều tầng là những thuật toán học máy dựa trên việc học các tầng biểu diễn khác nhau của dữ liệu.

Deep Learning

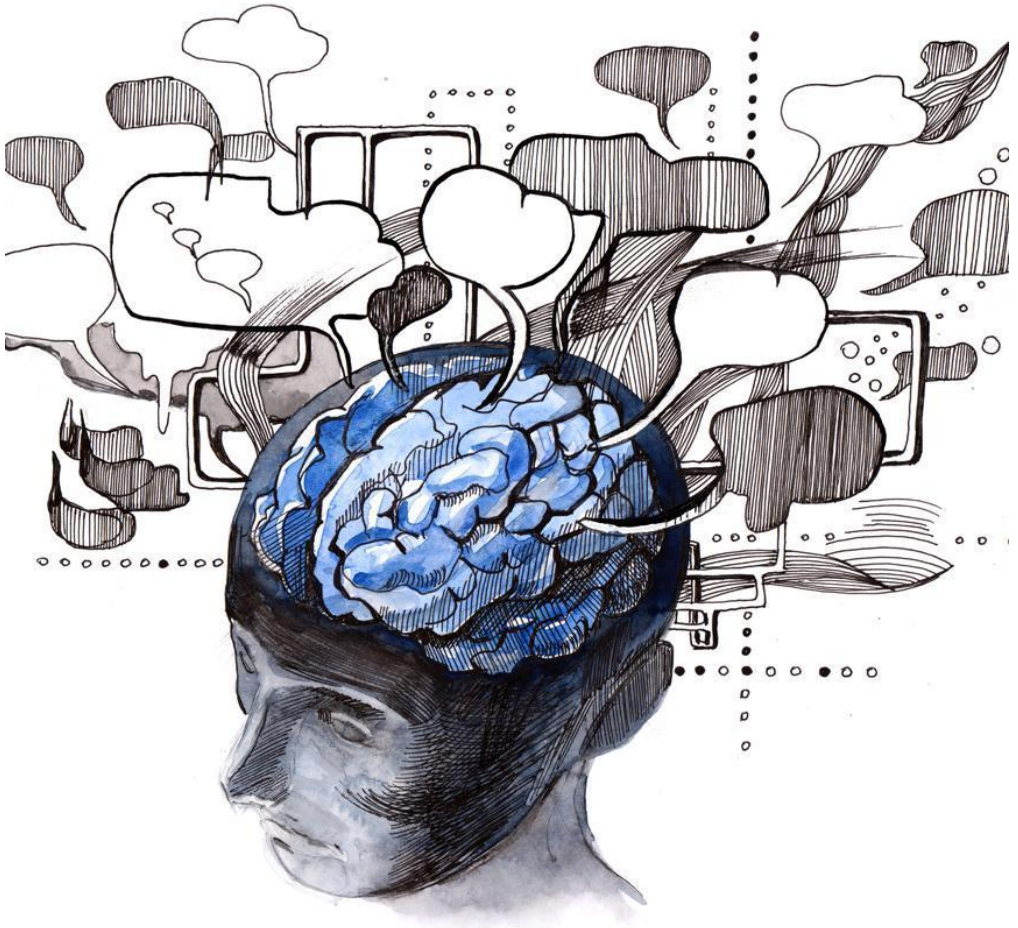
"Deep Learning: machine learning algorithms based on learning multiple levels of representation and abstraction" - Joshua Bengio



Feature visualization of CNN trained on ImageNet [Zeiler and Fergus 2013]

Deep Learning

- A fast moving field! The literature is vast.
- Rooted in machine learning and early pattern recognition theory.
- Raising technology to build Artificial Intelligence systems.
- Goals of this talk:
  - Give the basic foundations, i.e., **it's important to know the basic**.
  - Give an overview on important classes of deep learning models.



[Image courtesy: @DeepLearningIT]

What drives success in DL?

- Flexible models
- Data, data and data
- Distributed and parallel computing
  - CPU: exploit fixed point arithmetic, cache-friendly
  - GPU: high memory bandwidth, no cache
  - Multi-GPU/Multi-core/machines
  - Data parallelism: split data and run in parallel with the same model
    - Fast at test time
    - Asynchronous at training time
  - Model parallelism: one unit of data, but run multiple models (e.g., Gibbs)

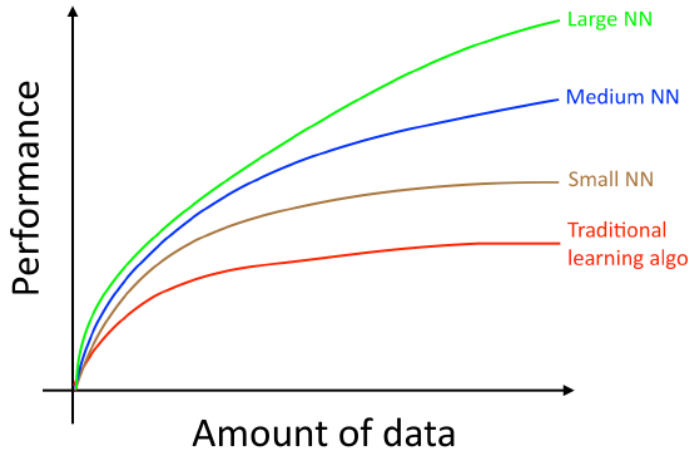
What makes deep learning take off?

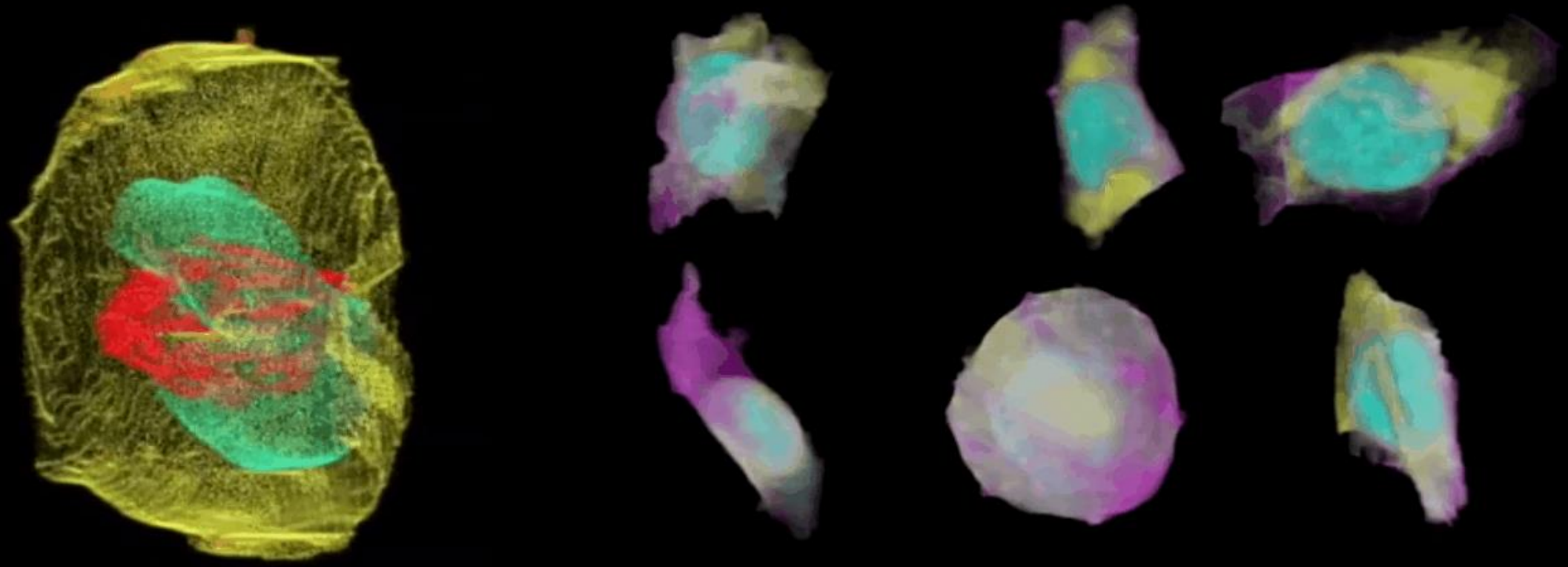
Andrew Ng's Analogy

Mô hình tính toán lớn, ví dụ: deep learning models

Cơ sở hạ tầng tính toán, nhà nghiên cứu, môi trường và chính sách như bệ phóng

Dữ liệu lớn như nguồn năng lượng





Machine learning predicts the look of stem cells, *Nature News*, April 2017

The Allen Cell Explorer Project

"No two stem cells are identical, even if they are genetic clones …. Computer scientists analysed thousands of the images using deep learning programs and found relationships between the locations of cellular structures. They then used that information to predict where the structures might be when the program was given just a couple of clues, such as the position of the nucleus. The program 'learned' by comparing its predictions to actual cells"

Applied DL and AI Systems

Computer vision and data technology

Speech Recognition & Computer Vision

Google IBM Microsoft

Drug discovery & Medical Image Analysis

MERCK NOVARTIS

Recommender System

amazon.com ebay NETFLIX

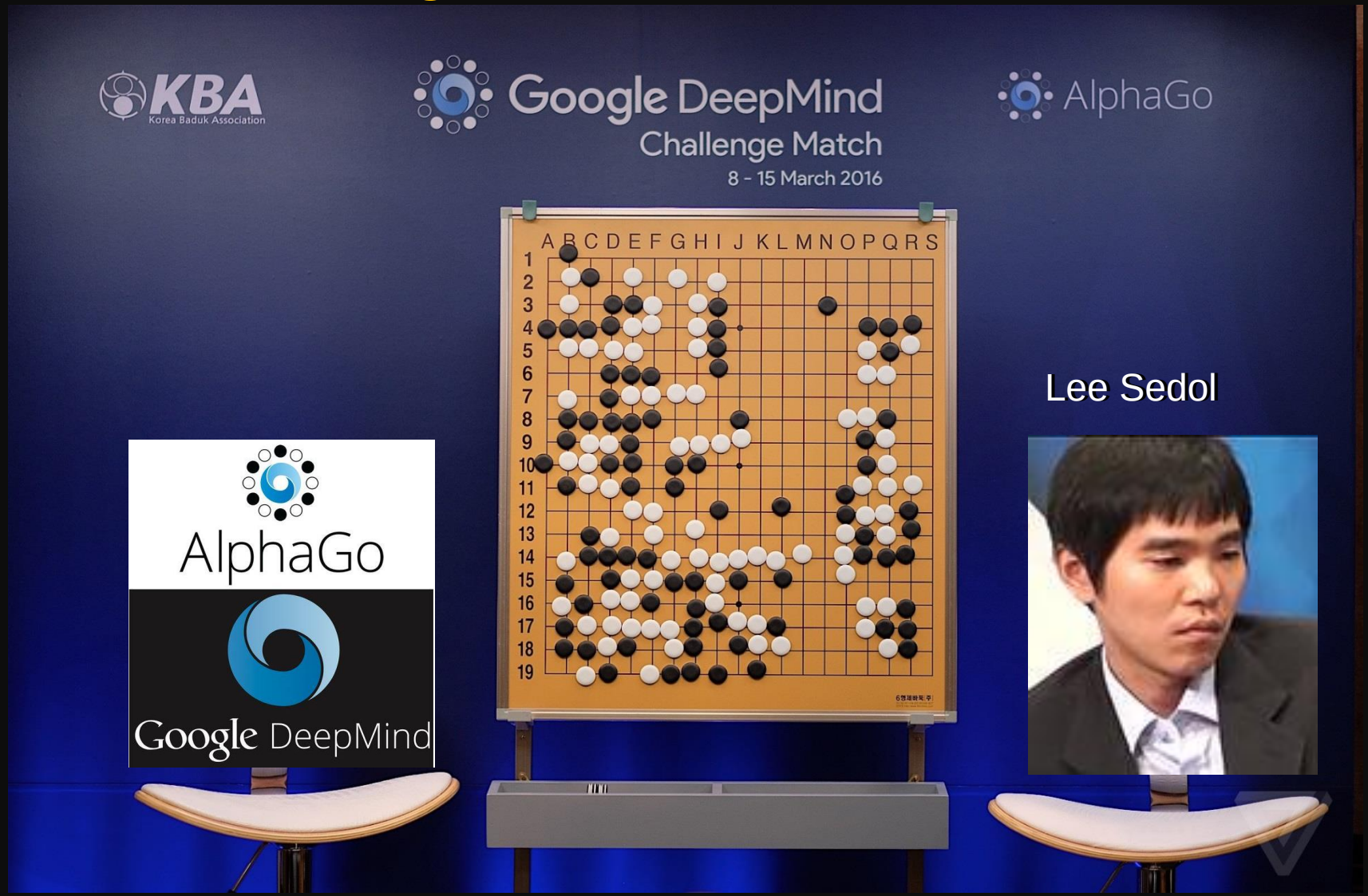
Nature, 2016

Mastering the game of Go with deep neural networks and tree search

David Silver[1]*, Aja Huang[1]*, Chris J. Maddison[1], Arthur Guez[1], Laurent Sifre[1], George van den Driessche[1], Julian Schrittwieser[1], Ioannis Antonoglou[1], Veda Panneershelvam[1], Marc Lanctot[1], Sander Dieleman[1], Dominik Grewe[1], John Nham[2], Nal Kalchbrenner[1], Ilya Sutskever[2], Timothy Lillicrap[1], Madeleine Leach[1], Koray Kavukcuoglu[1], Thore Graepel[1] & Demis Hassabis[1]

Applied DL and AI Systems

Reinforcement Learning



Impact of Deep Learning

Applied DL and AI Systems

Reinforcement Learning

1997

2016

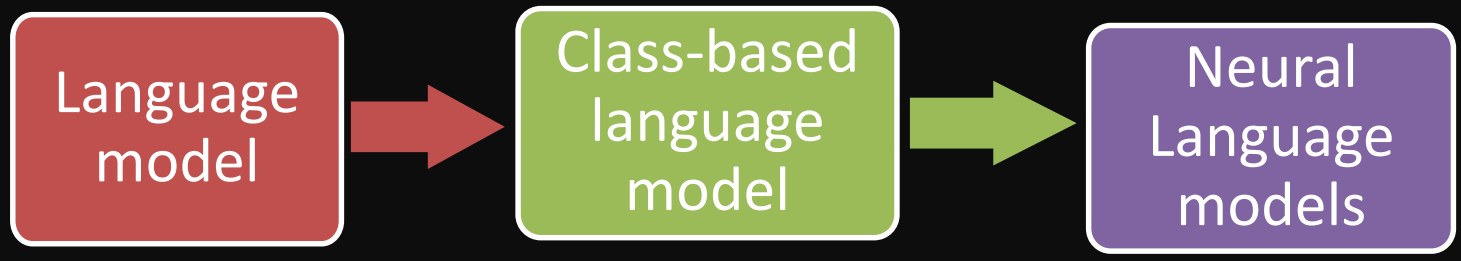
Bản chất vấn đề là gì?

Từ heuristics đến ...

AUTOMATED LEARNING!

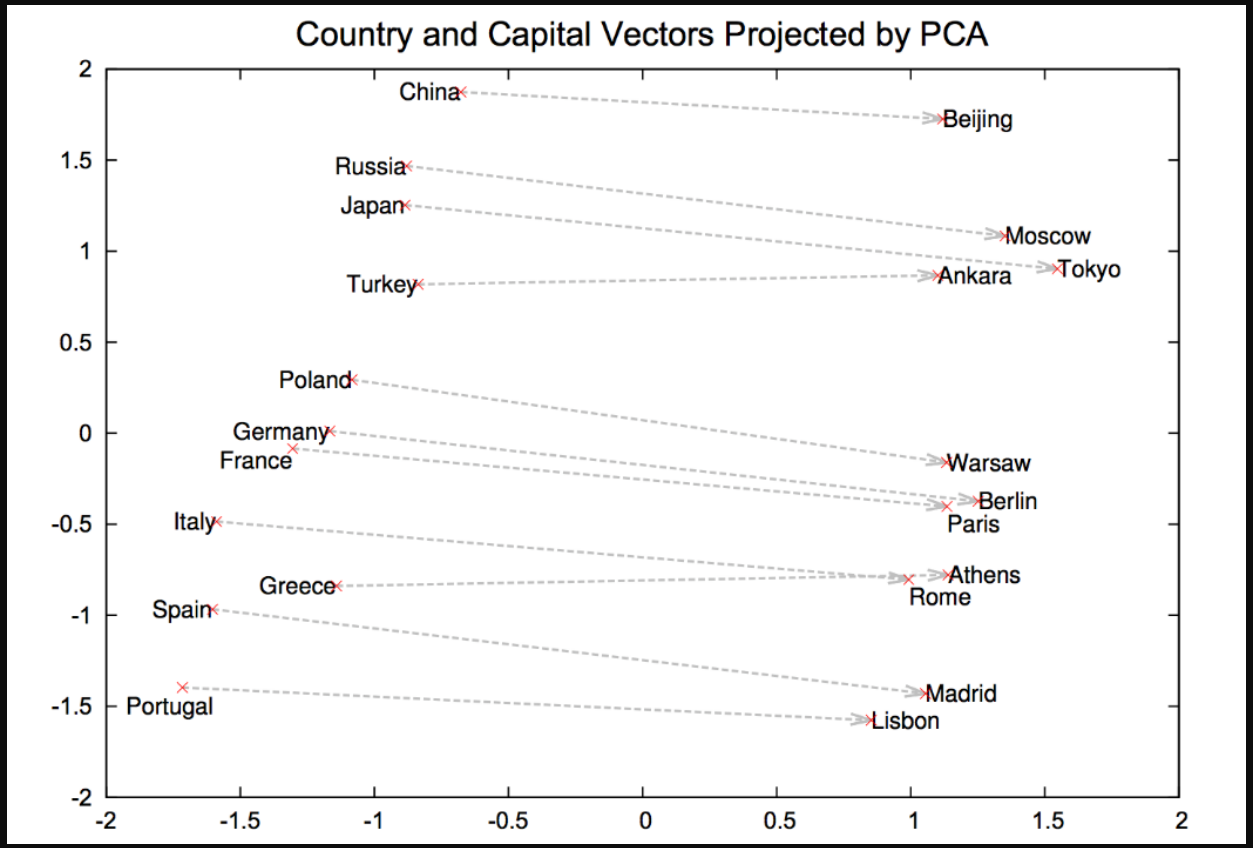
Neural Language NLP

Impact of Deep Learning

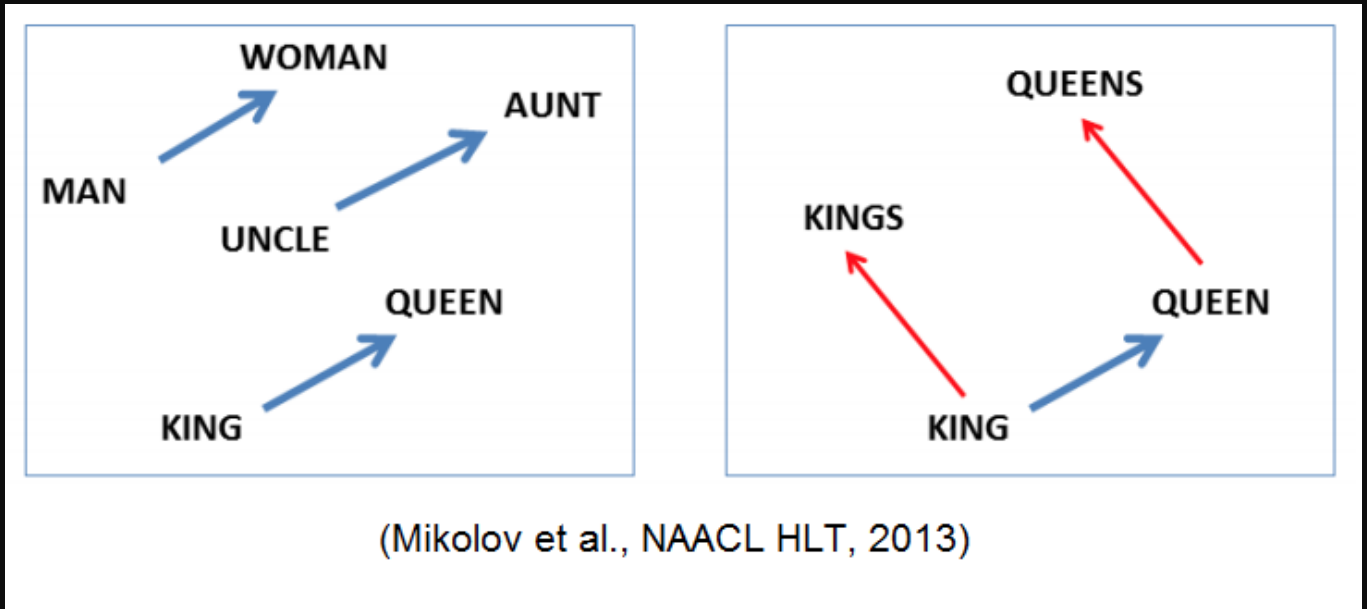


word2vec

[Mikolov et al. 2013]



Neural Language NLP

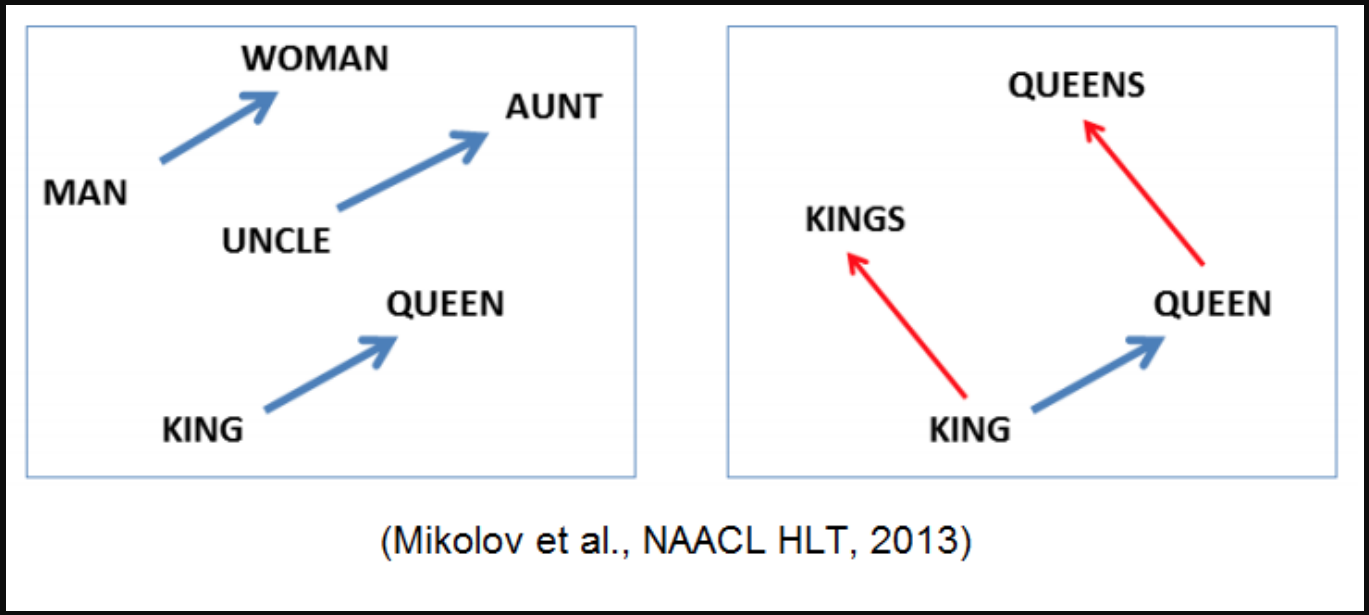


(Mikolov et al., NAACL HLT, 2013)

Analogical reasoning: đàn ông : phụ nữ = hoàng đế : ?

$$\text{argmin}_v \left\| h_{\text{đàn ông}} - h_{\text{phụ nữ}} + h_{\text{hoàng đế}} - h_v \right\|^2$$

Neural Language NLP



(Mikolov et al., NAACL HLT, 2013)

Analogical reasoning: đàn ông : phụ nữ = hoàng đế : nữ hoàng

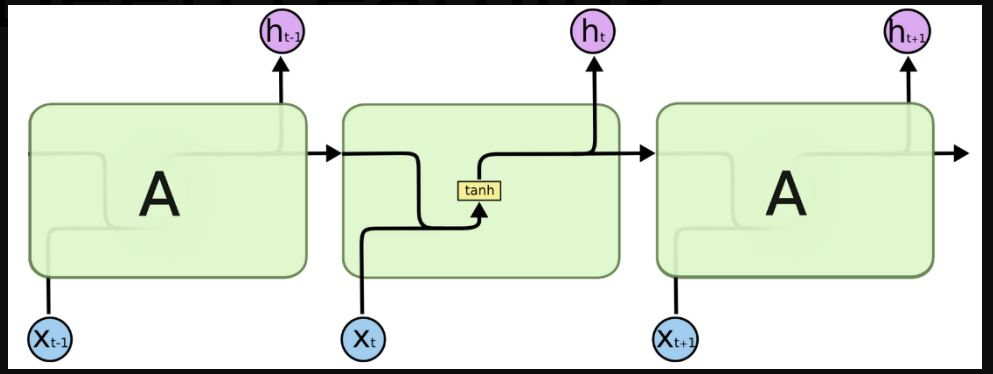
$$\text{argmin}_v \left\| h_{\text{đàn ông}} - h_{\text{phụ nữ}} + h_{\text{hoàng đế}} - v \right\|^2$$

Applied DL and AI Systems

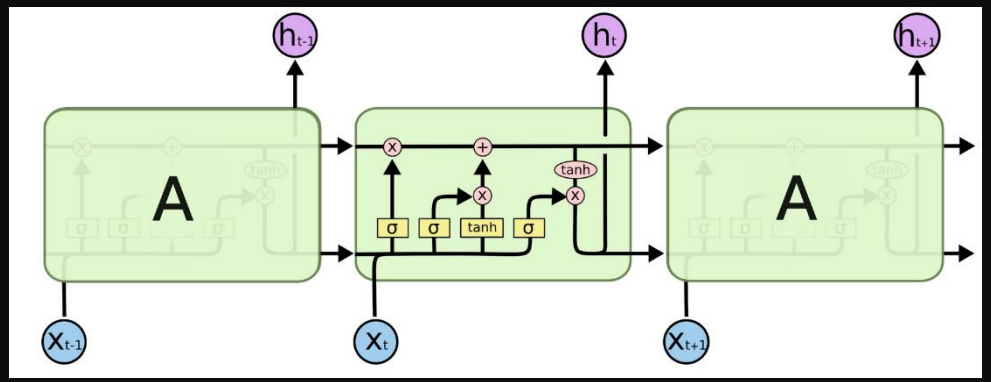
Neural Language NLP

Tí đi vào nhà bếp
Ti lượm được một trái khế
Sau đó Tí chạy ra vườn
Tí làm rớt trái khế
Hỏi: trái khế ở đâu?
Máy tính: ở ngoài vườn

Dư hấu thì tròn
Quả mướp thì dài
Mướp xanh mươn mướt
Mướp và dưa hấu cùng màu
Hỏi: quả dưa hấu màu gì?
Máy tính: màu xanh



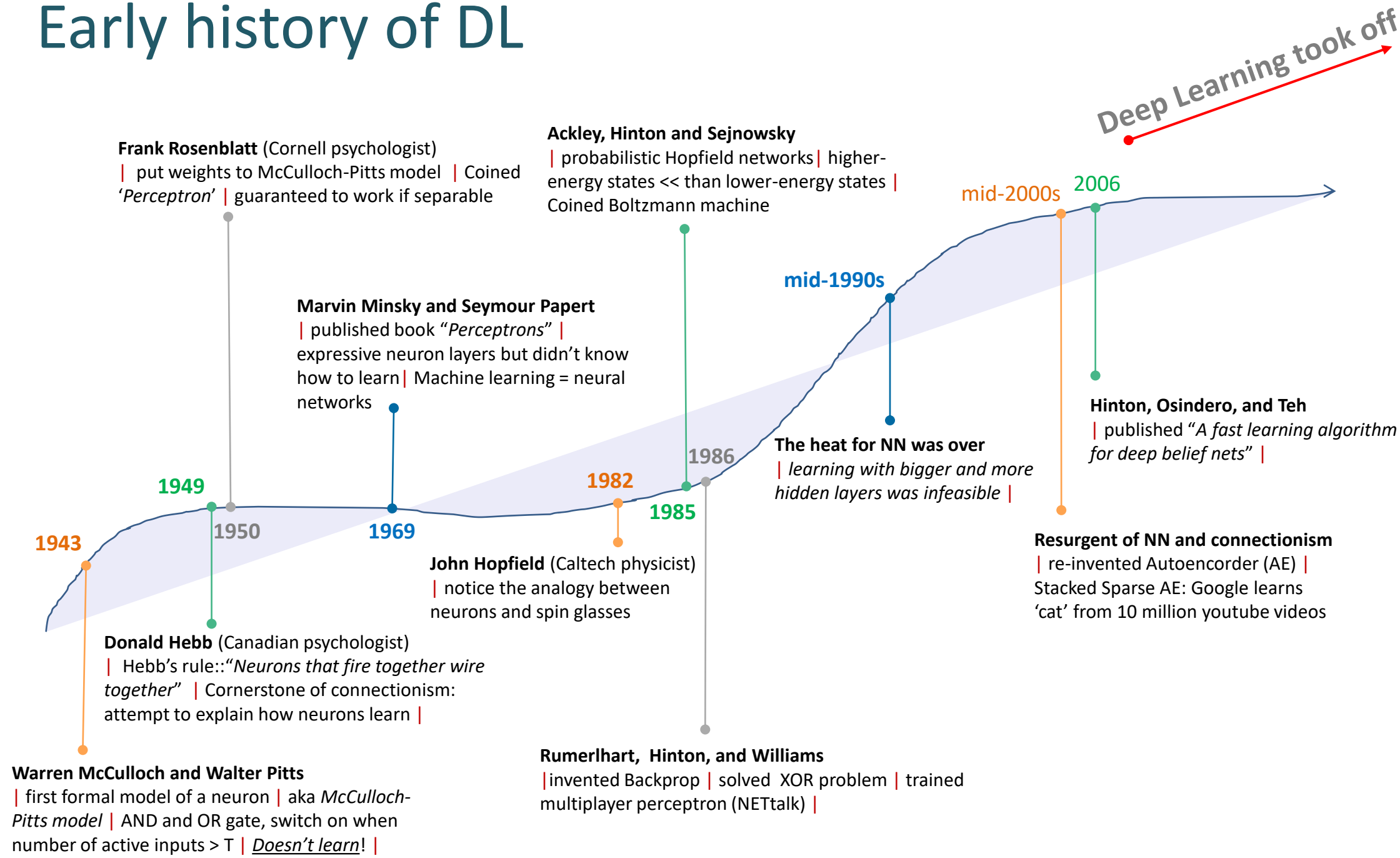
RNNs



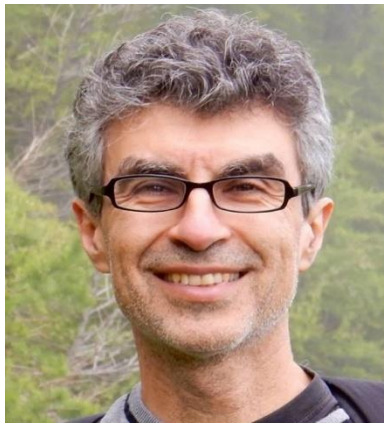
LSTMs

- Neural Machine Translation
  - Sequence to sequence (Sutskever et. al, 14)
  - Attention-based Neural MT (Luong, et al. 15)
- End-to-End Memory Networks
- Neural Turing Machine (Graves et al. 14)

Early history of DL



Yann LeCun
Director of AI Research, Facebook



Yoshua Bengio
Head, Montreal Institute for Learning Algorithms

Geoff Hinton
Google Brain Toronto, University of Torronto

Deep learning

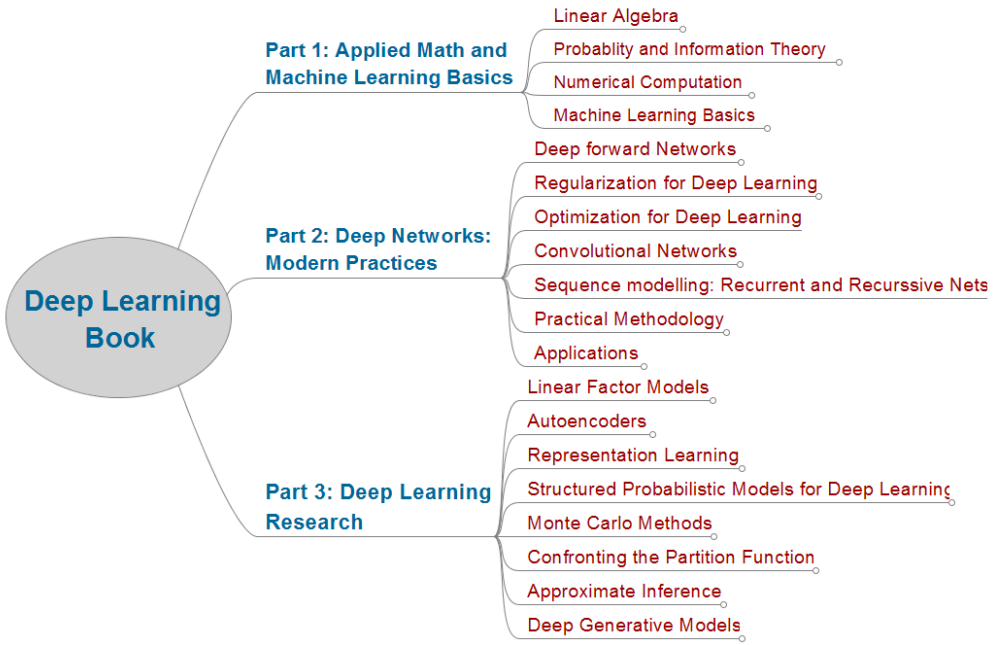
Yann LeCun[1,2], Yoshua Bengio[3] & Geoffrey Hinton[4,5]

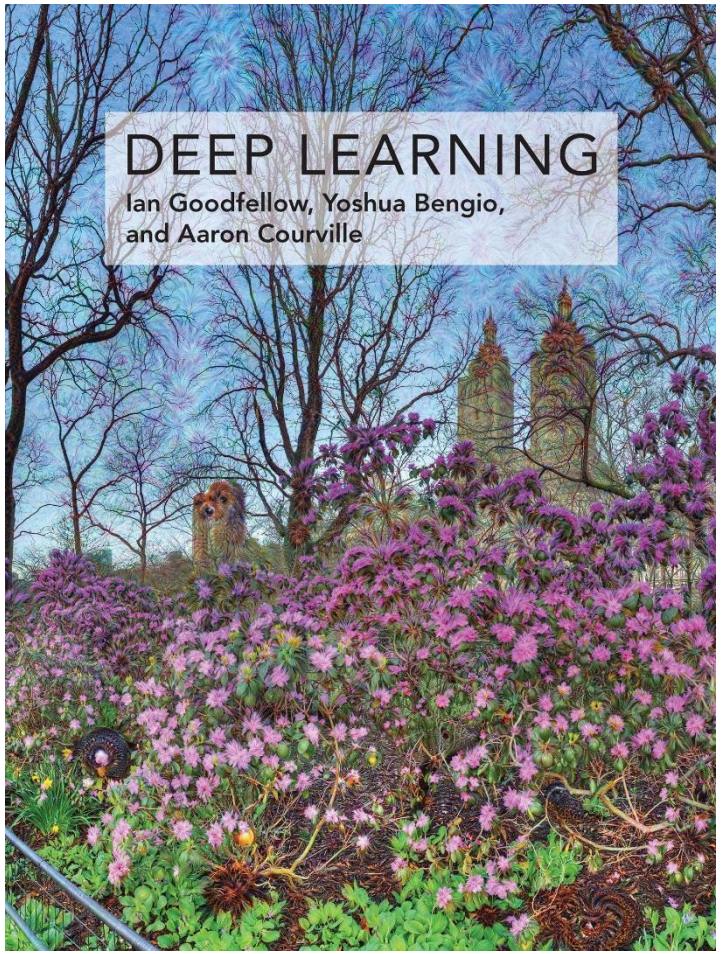
nature

Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection and many other domains such as drug discovery and genomics. Deep learning discovers intricate structure in large data sets by using the backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer. Deep convolutional nets have brought about breakthroughs in processing images, video, speech and audio, whereas recurrent nets have shone light on sequential data such as text and speech.

Books

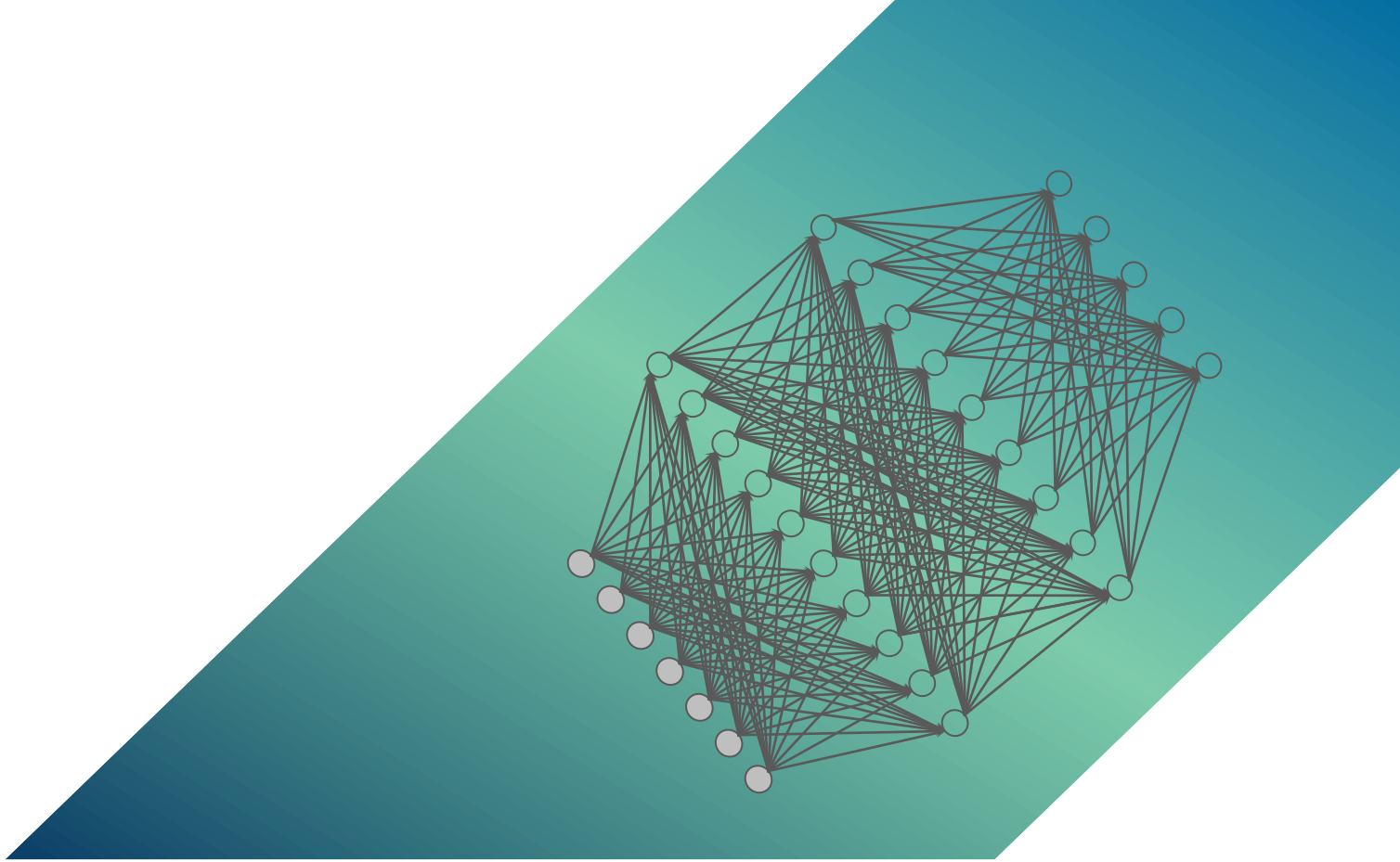
- Currently the (only) most comprehensive book in DL
- What is not covered in this book:
  - Latest development (extremely fast moving field)
  - Lack of practical deployments, codebase, framework – which is important for this field.
  - Represent a very subset of the field, e.g., no discussion of statistical foundation, uncertainty, Bayesian treatment.





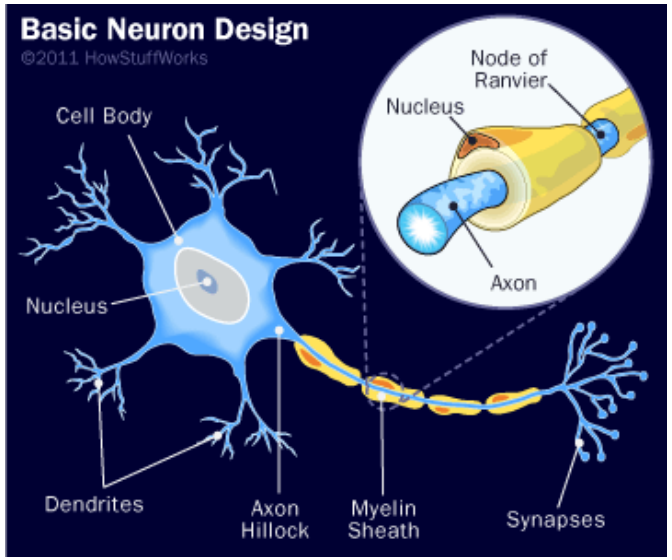
About this talk

- What is deep learning and few application examples.
- Deep neural networks
- Neural embedding
- Deep Generative Models
- TensorFlow
- Few practical tips



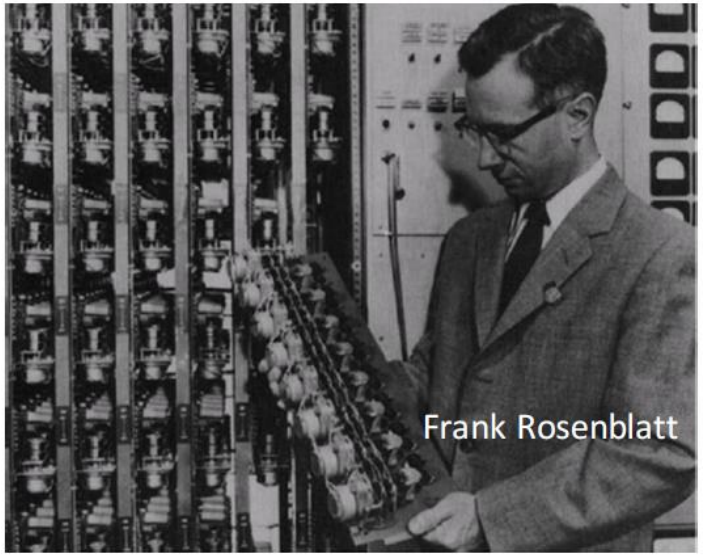
DEEP NEURAL NETWORKS

Some historical motivations

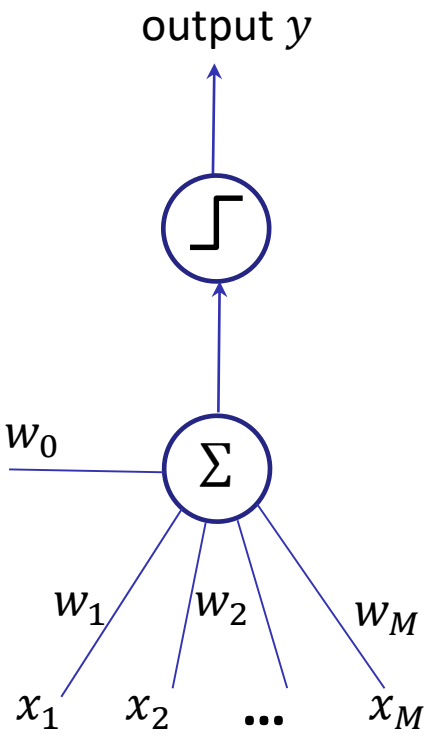


- Our brain is made up by network of inter-connected neurons and have highly-parallel architecture.
- ANN (*artificial neural networks*) are motivated by biological neural systems.
- Two groups of ANN researchers:
  - To use ANN to study/model the brain
  - Use the brain as the motivation to design ANN as an effective learning machine, which might not be the true model of the brain. Deviation from the brain model is not necessarily bad, e.g. airplanes do not fly like birds.
- So far, most, if not all, use the brain architecture as motivation to build computational models.

Perceptron

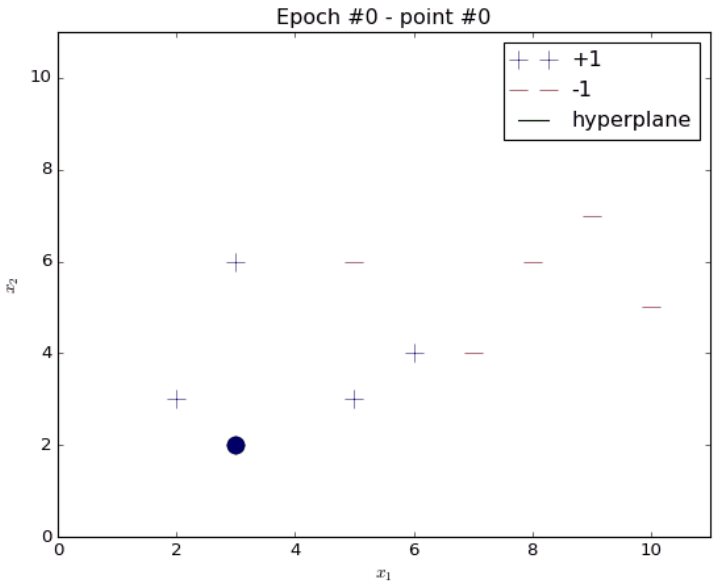


[Rosenblatt, late 50's]



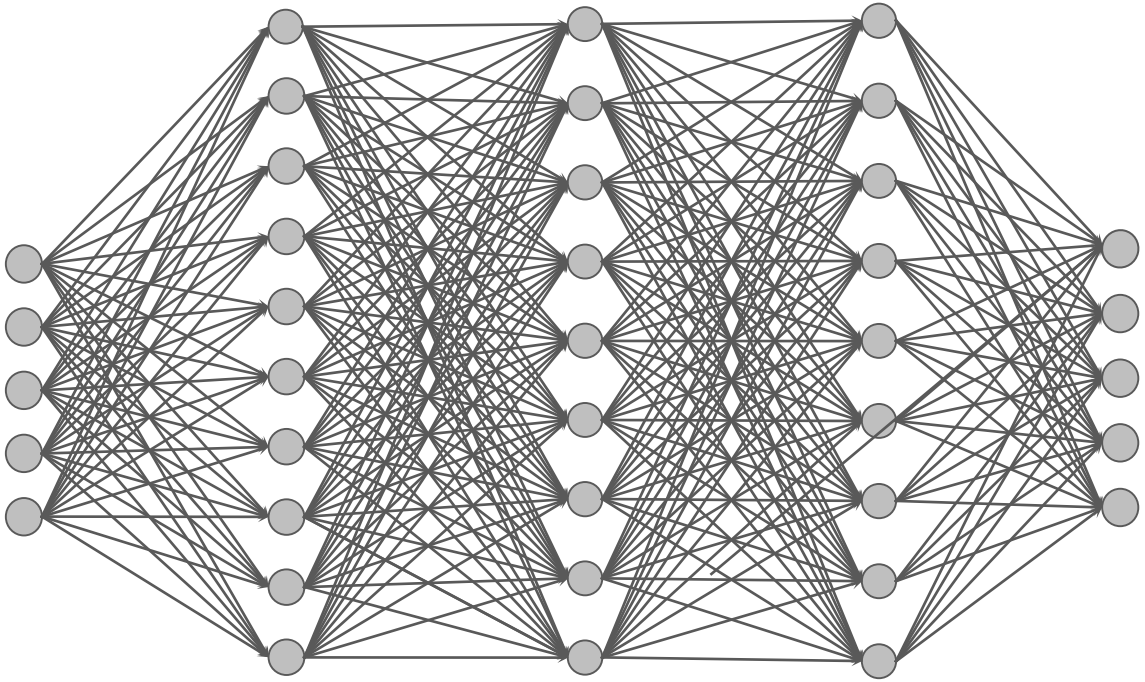
single 'neuron' unit

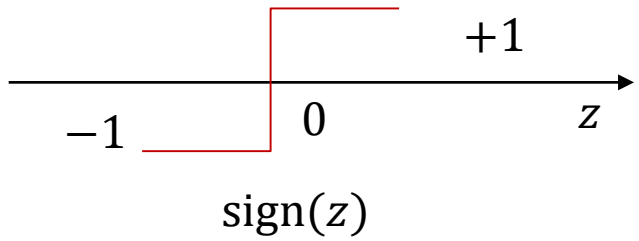
$$\min J(\boldsymbol{w}) = \frac{1}{2}\sum_{t=1}^{n}[y_t - \boldsymbol{w}^{\mathsf{T}}\boldsymbol{x}]^2$$



Feed Forward Neural Networks

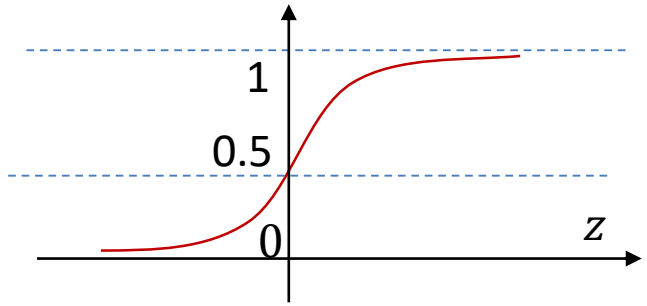
- Perceptron are linear models and too weak in what they can represent.
- Modern ML/Stats deal with high-dimensional data.
  - non-linear decision surfaces





$\text{sign}(z)$

small step, big leap in modelling



$$\sigma(z) = \frac{1}{1+e^{-z}}$$

$$\frac{d\sigma(z)}{dz} = \sigma(z)(1-\sigma(z))$$

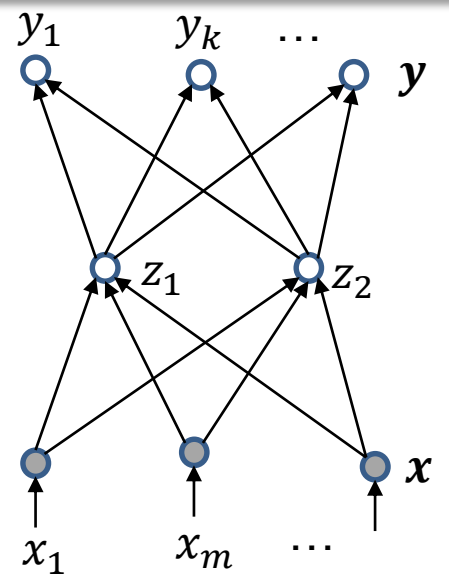
Activation Functions

- Given input $\boldsymbol{x}_t$ and desired output $\boldsymbol{y}_t$, $t = 1, \ldots, n$, find the network weights $\boldsymbol{w}$ such that

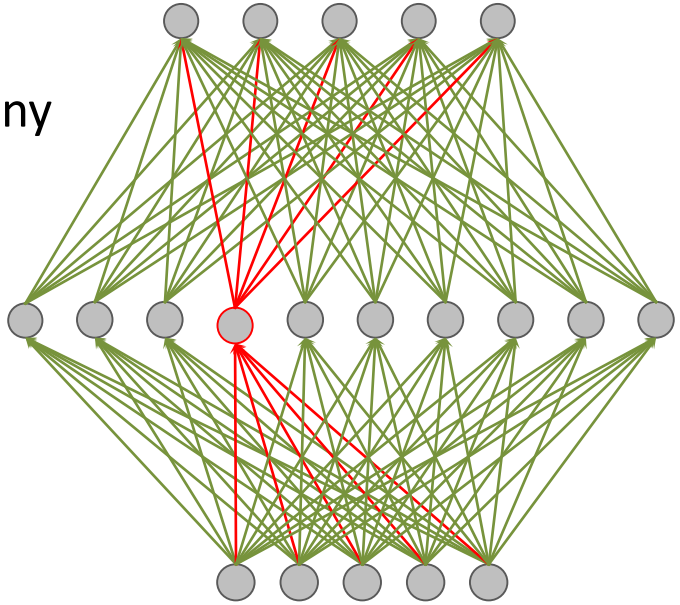
$$\hat{\boldsymbol{y}}_t \approx \boldsymbol{y}_t, \forall t$$

- Stating as an optimization problem: find w to minimize the error function

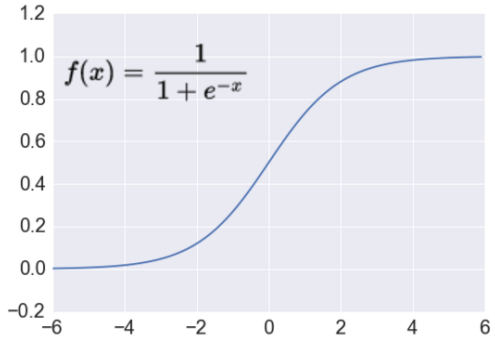
$$J(\boldsymbol{w}) = \frac{1}{2}\sum_{t=1}^{M}\sum_{k=1}^{K}(y_{tk} - \hat{y}_{tk})^2$$



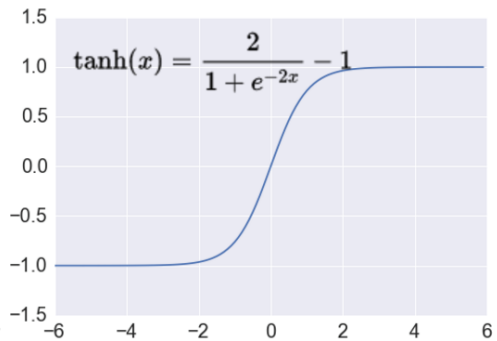
- Neural activation: $h(\boldsymbol{x}) = f(\boldsymbol{w}^\top \boldsymbol{x} + b)$
- $J(\boldsymbol{w})$ is a nonconvex, high-dimensional function with possibly many local minima.



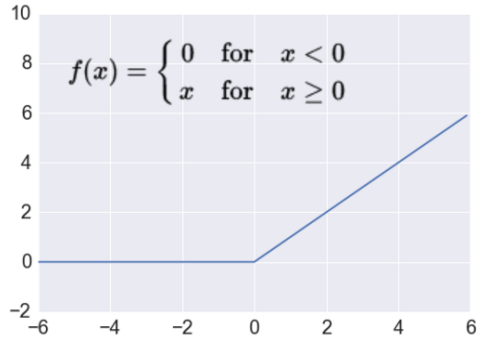
Sigmoid



TanH



ReLU



Gradient-based optimization and search

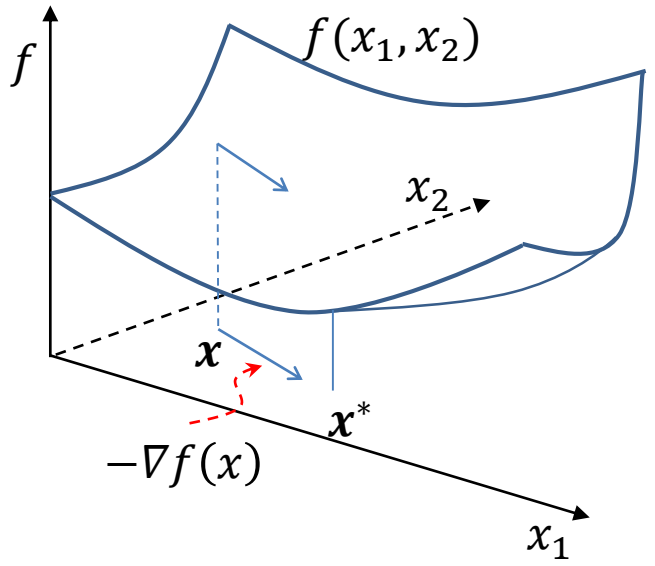
Perceptron

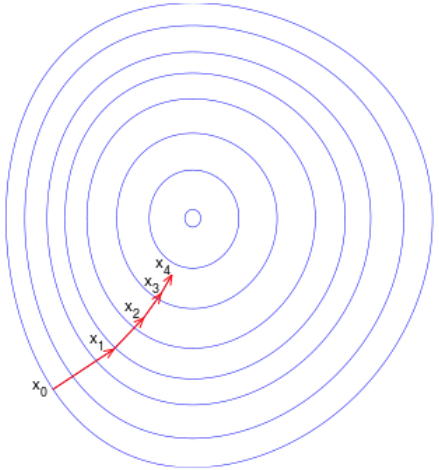
$$\min J(\boldsymbol{w}) = \frac{1}{2}\sum_{t=1}^{n}[y_t - \boldsymbol{w}^\top \boldsymbol{x}]^2$$

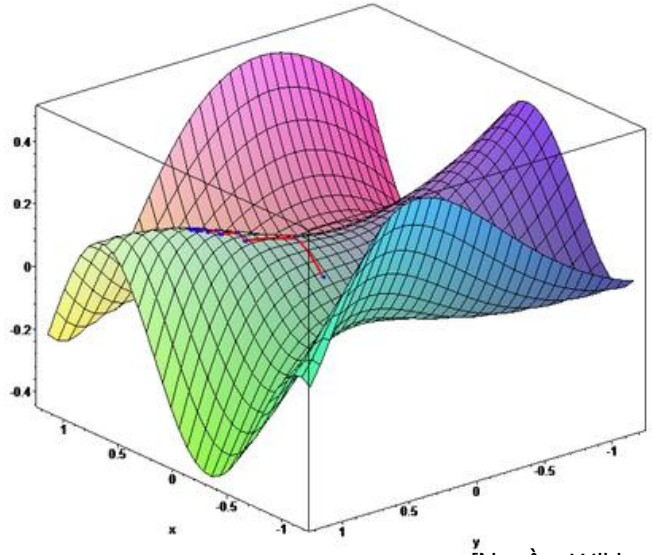
- At a point $\boldsymbol{x} = (x_1, x_2)$, the gradient vector of the function $f(\boldsymbol{x})$ w.r.t $\boldsymbol{x}$ is

$$\nabla f(\boldsymbol{x}) = \left(\frac{\partial f}{\partial\, x_1}, \frac{\partial f}{\partial\, x_2}\right)$$

- $\nabla f(\boldsymbol{x})$ represents the direction that produces steepest increase in f.
- Similarly $-\nabla f(x)$ is the direction of steepest decrease in f







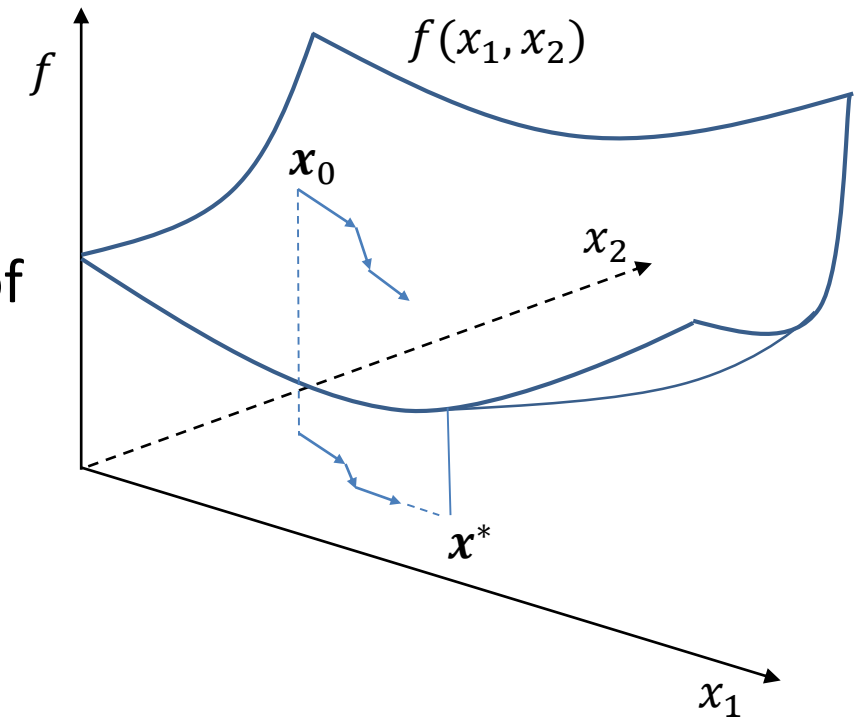
[Nguồn: Wikipedia]

Gradient-based optimization and search

- To minimize a functional $f(\boldsymbol{x})$, use gradient-descent:
  - Initialize random $\boldsymbol{x}_0$
  - Slide down the surface of f in the direction of steepest decrease:

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \eta \times \nabla f(\boldsymbol{x}_t)$$

learning rate



- Similarly, to maximize $f(\boldsymbol{x})$, use gradient-ascent.

Stochastic Gradient-Descent (SGD) Learning

- Also known as delta rule, LMS rule or Widrow-Hoff rule.
- Instead of minimizing $J(\boldsymbol{w})$, minimize the instantaneous approximation of $J(\boldsymbol{w})$:

$$J_t(\boldsymbol{w}) = \frac{1}{2}[y_t - \hat{y}_t]^2$$

$$J(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{n}[\boldsymbol{x}_t \cdot \boldsymbol{w} - y_t]^2$$

$$\frac{\partial J}{\partial w_i} = -\sum_{t=1}^{n}[\hat{y}_t - y(t)]x_{ti}$$

- Partial derivative:

$$\frac{\partial J_t(\boldsymbol{w})}{\partial w_i} = -(y_t - \hat{y}_t)x_{ti}$$

new old

- Update rule (where t denotes the current training sample)

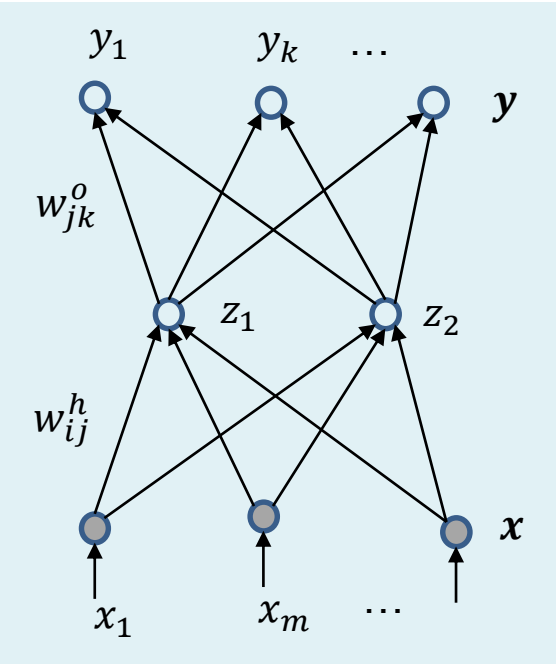
$$w_i \leftarrow w_i + \eta(y_t - \hat{y}_t)x_{ti}$$

- Compare with standard gradient descent:
  - Less computation required in each iteration
  - Approaching the minimum in a stochastic sense
  - Use smaller learning rate η, hence need more iterations

Backpropagation

- Similarly for input → hidden weights

$$J_t(\boldsymbol{w}) = \frac{1}{2}\sum_{k=1}^{K}(\hat{y}_k - y_k)^2 \qquad z_j = \sigma(\bar{z}_j) \text{ where } \bar{z}_j = \sum_{i=1}^{M} x_i w_{ij}$$



$$w_{ij}^h \leftarrow w_{ij}^h - \eta \frac{\partial J_t(\boldsymbol{w})}{\partial w_{ij}^h}$$

$$\frac{\partial J_t(\boldsymbol{w})}{\partial w_{ij}^h} = \frac{\partial J_t}{\partial \bar{z}_j} \times \frac{\partial \bar{z}_j}{\partial w_{ij}^h} \qquad \frac{\partial \bar{z}_j}{\partial w_{ij}^h} = x_i$$

$$\frac{\partial J_t}{\partial \bar{z}_j} = \frac{\partial J_t}{\partial z_j} \times \frac{\partial z_j}{\partial \bar{z}_j} \qquad \frac{\partial z_j}{\partial \bar{z}_j} = z_j(1 - z_j)$$

chain rule

$$\frac{\partial J_t}{\partial z_j} = \sum_{k=1}^{K} \frac{\partial J_t}{\partial \bar{y}_k} \times \frac{\partial \bar{y}_k}{\partial z_j} \qquad \frac{\partial \bar{y}_k}{\partial z_j} = w_{jk}^0 \text{ since } \bar{y}_k = \sum w_{jk}^0 z_j$$

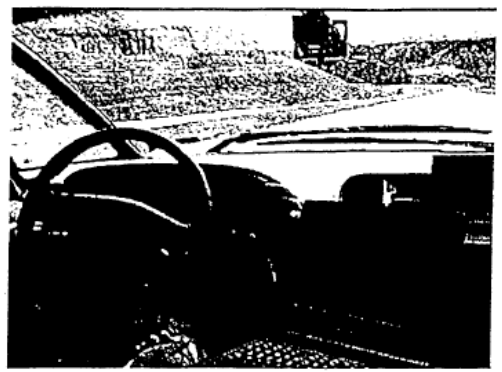
$$\frac{\partial J_t}{\partial \bar{y}_k} = -\delta_k^o$$

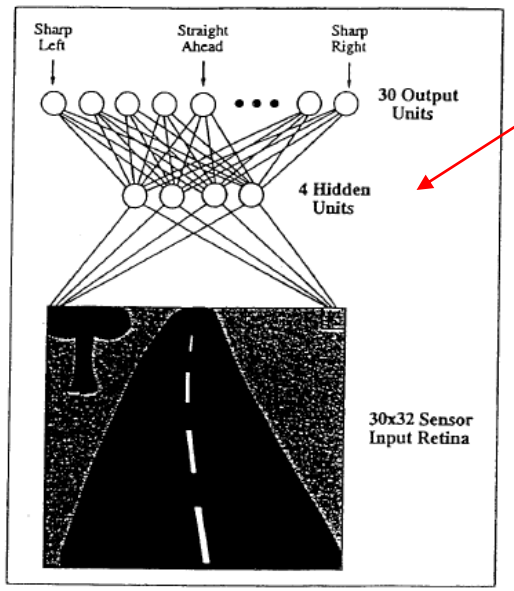
$$\frac{\partial J_t}{\partial \bar{z}_j} = -\delta_j^h = -z_j(1 - z_j)\sum_{k=1}^{K} w_{jk}^o \delta_k^o$$

- Gradient descent update

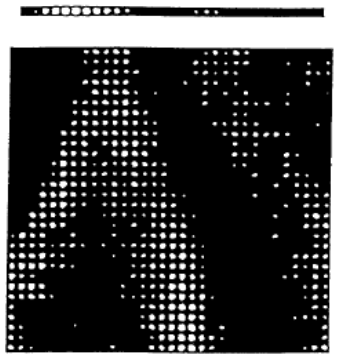
$$w_{ij}^h \leftarrow w_{ij}^h + \eta \delta_j^h x_i$$

Early day of NN application



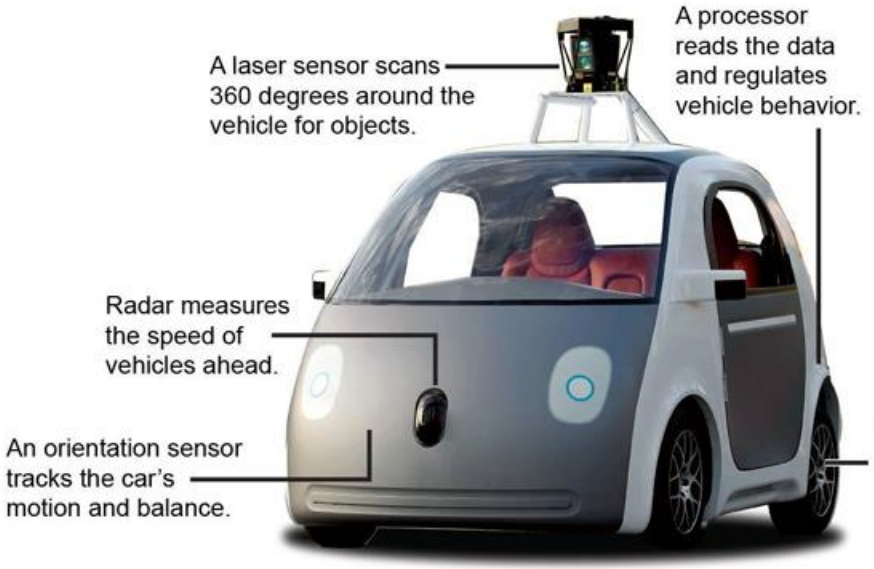


4 nodes

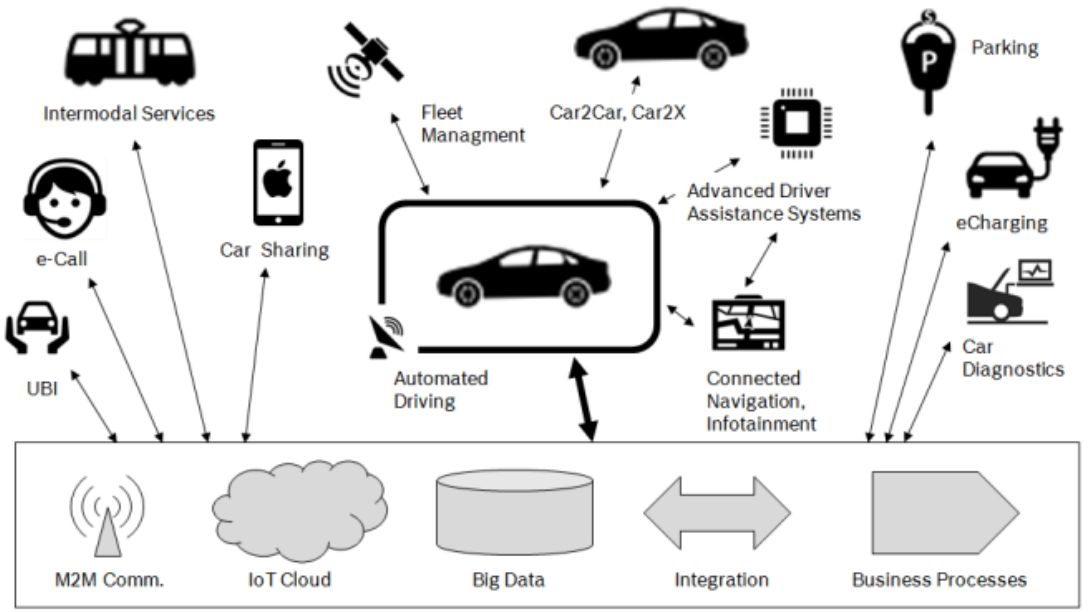


[Nguồn: *Machine Learning*, Tom Mitchell, 1997]

.. and TODAY



Truck,
Train,
Boat,
Uber,
Delivery,
.....



Tricks and tips for applying BP

Sensitivity of learning rate η

- SGD provides *unbiased estimate* of the gradient by taking average gradient on a minibatch.
- Since the SGD gradient estimator introduces a source of noise (random sampling data example), thus gradually decrease/reduce the learning rate over time.
- Let η_k be the learning rate at epoch k. A sufficient condition to guarantee convergence of SGD is:

$$\sum_{k}^{\infty} \eta_k = \infty \quad \text{and} \quad \sum_{k}^{\infty} \eta_k^2 < \infty$$

- In practice, it is common to decay the learning rate linearly until iteration τ:

$$\eta_k = (1 - \alpha)\eta_0 + \alpha\eta_\tau$$

with $\alpha = \frac{k}{\tau}$. After iteration τ, leave η constant.

- Monitor convergence rate
  - Measure the excess error

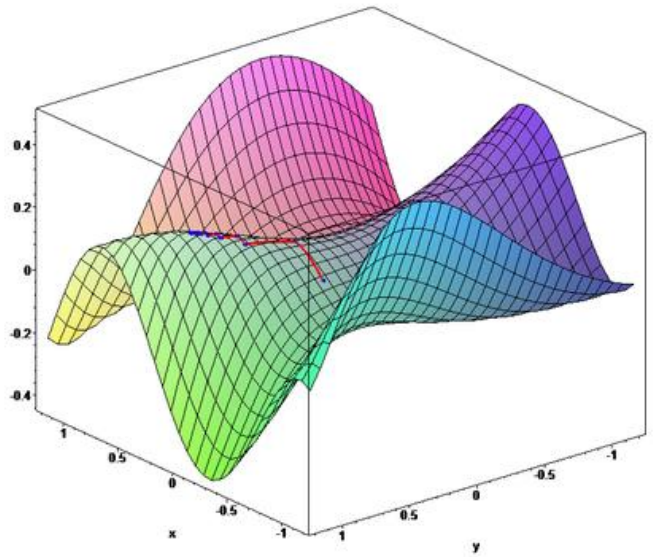
$$J(\boldsymbol{\theta}) - \min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

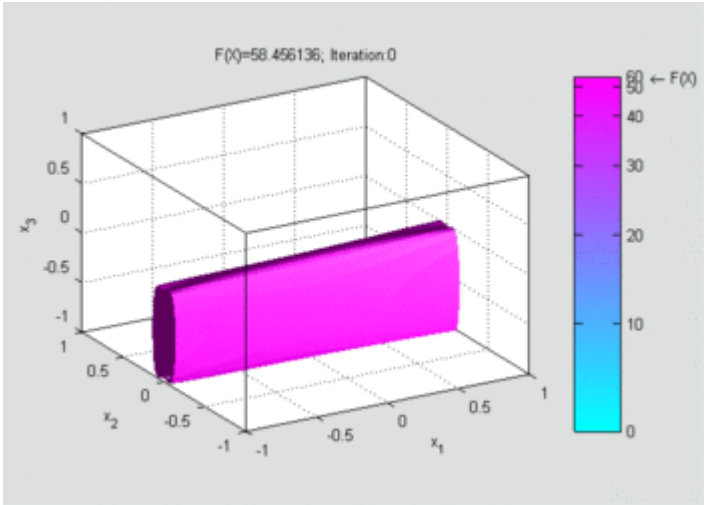
  - In a convex optimization problem: the excess error is $\mathcal{O}\left(\frac{1}{\sqrt{k}}\right)$ after k iterations
  - In a strongly convex optimization problem, excess error is: $\mathcal{O}\left(\frac{1}{k}\right)$

Tricks and tips for applying BP

Sensitivity of learning rate η

- Start with large learning rate (e.g., 0.1)
- Maintain until validation error stops improving
- ***Step decay***: Divide the learning rate by 2 and go back to the second step.
- ***Exponential decay***: $\eta = \eta_0 e^{-kt}$ where k is a hyperparameter and t is the epoch/iteration number
- ***1/t decay***: $\eta = \frac{\eta_0}{1+kt}$ where k is a hyperparameter and t is the epoch/iteration number.





[Image courtesy: Wikipedia]

Strategies to decay learning rate η

Tricks and tips for applying BP

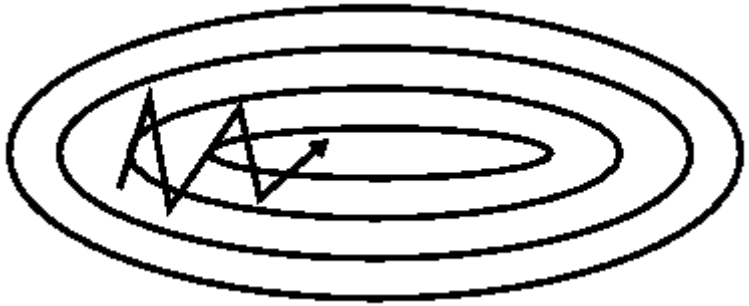
Accelerate learning - Momentum

- SGD can sometimes be slow.
- Momentum (Polyak, 1964) accelerates the learning by accumulating an exponentially decaying moving average of past gradients and then continuing to move in their direction.

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$$

- α is a hyperparameter that indicates how quickly the contributions of previous gradients exponentially decay. In practice, this is usually set to 0.5, 0.9, and 0.99.
- The momentum primarily solves 2 problems: poor conditioning of the Hessian matrix and variance in the stochastic gradient.

normal SGD



SGD with momentum

Momentum helps accelerate SGD to use relevant directions and dampens

[Image courtesy: Sebastian Ruder]

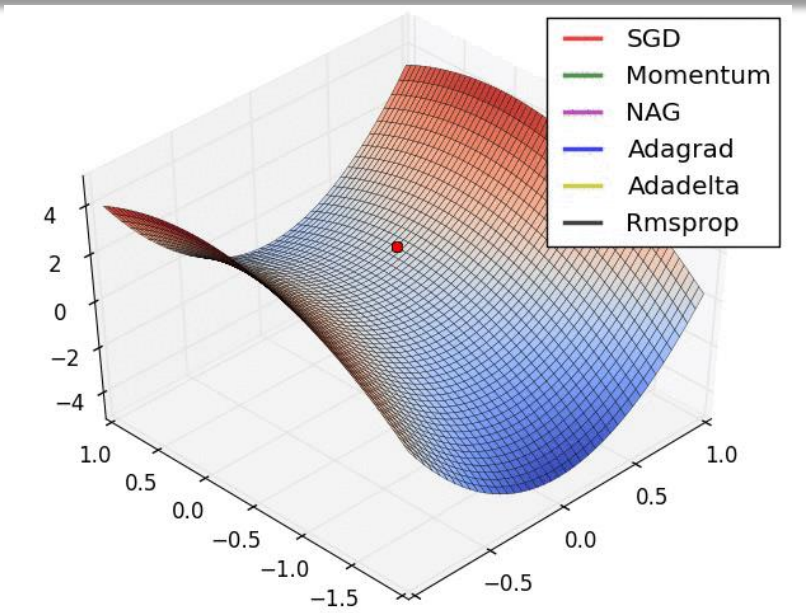
Tricks and tips for applying BP

[Image courtesy: Sebastian Ruder]

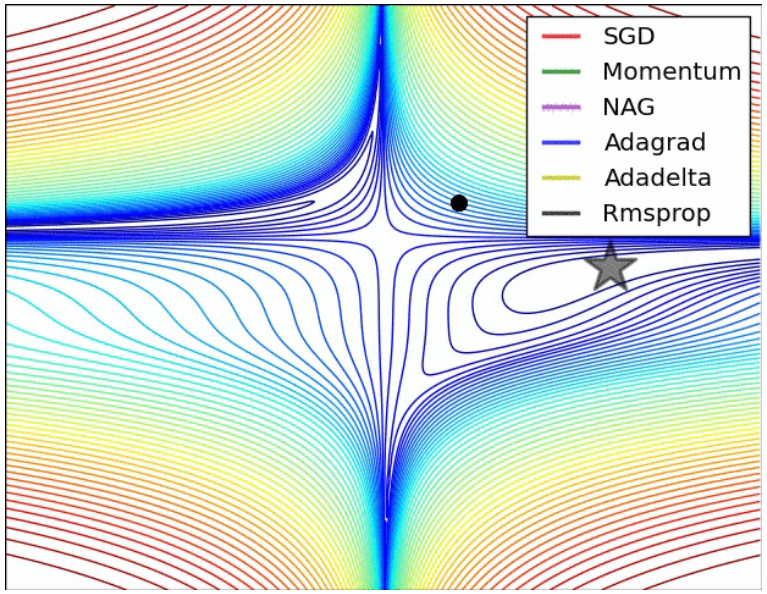
Accelerate learning - Nesterov Momentum

- A variant of the momentum algorithm, inspired by Nesterov's accelerated gradient method (Nesterov, 1983, 2004)

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta} + \alpha \boldsymbol{v})$$
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$$



- The only difference between Nesterov momentum and standard momentum is how the gradient is computed.
- For convex batch gradient, Nesterov momentum improves the convergence rate from $\mathcal{O}(1/k)$ to $\mathcal{O}(1/k^2)$.
- Unfortunately, this result does not hold for stochastic gradient.



Tricks and tips for applying BP

[Image courtesy: Sebastian Ruder]

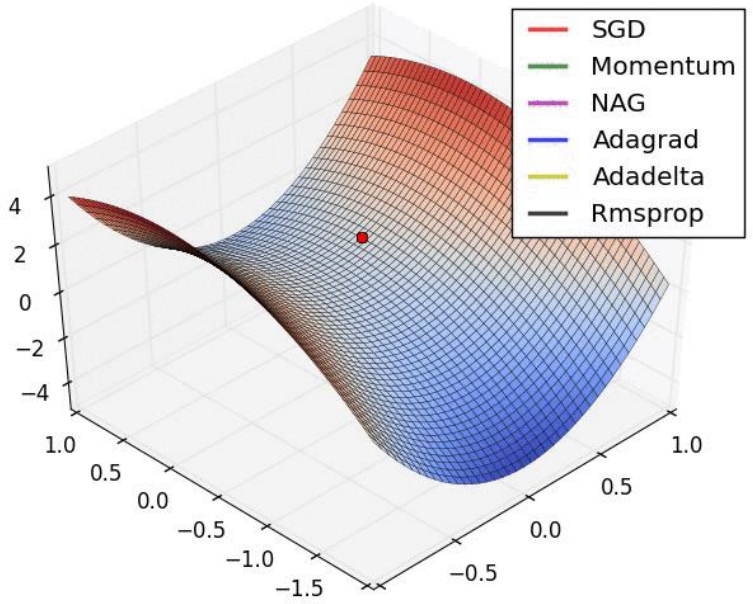
Accelerate learning – AdaGrad (Duchi, 11)

- Learning rates are scaled by the square root of the cumulative sum of squared gradients

$$\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$
$$\boldsymbol{\gamma} \leftarrow \boldsymbol{\gamma} + \boldsymbol{g} \odot \boldsymbol{g}$$
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \frac{\eta}{\delta + \sqrt{\boldsymbol{\gamma}}} \odot \boldsymbol{g}$$

where δ is a small constant (e.g., 10^{-7}) for numerical stability.

- Large partial derivatives
  - Thus, rapid decrease in their learning rates
- Small partial derivatives
  - hence relatively small decrease in their learning rates.
- Weakness: always decrease the learning rate!



Tricks and tips for applying BP

[Image courtesy: Sebastian Ruder]

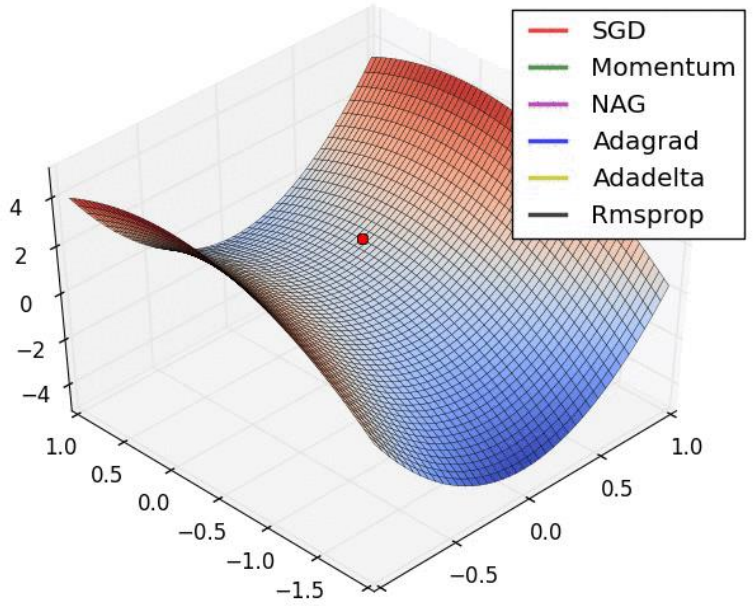
Accelerate learning – RMSProp (Hinton, 12)

- A modification of AdaGrad to work better for non-convex setting.
- Instead of cumulative sum, use exponential moving/smoothing average

$$\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$
$$\boldsymbol{\gamma} \leftarrow \beta\boldsymbol{\gamma} + (1-\beta)\boldsymbol{g} \odot \boldsymbol{g}$$
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \frac{\eta}{\sqrt{\delta+\boldsymbol{\gamma}}} \odot \boldsymbol{g}$$

where δ is a small constant (e.g., 10^{-6}) for numerical stability.

- RMSProp has been shown to be an effective and practical optimization algorithm for DNN. It is currently one of the go-to optimization methods being employed routinely by DL applications.



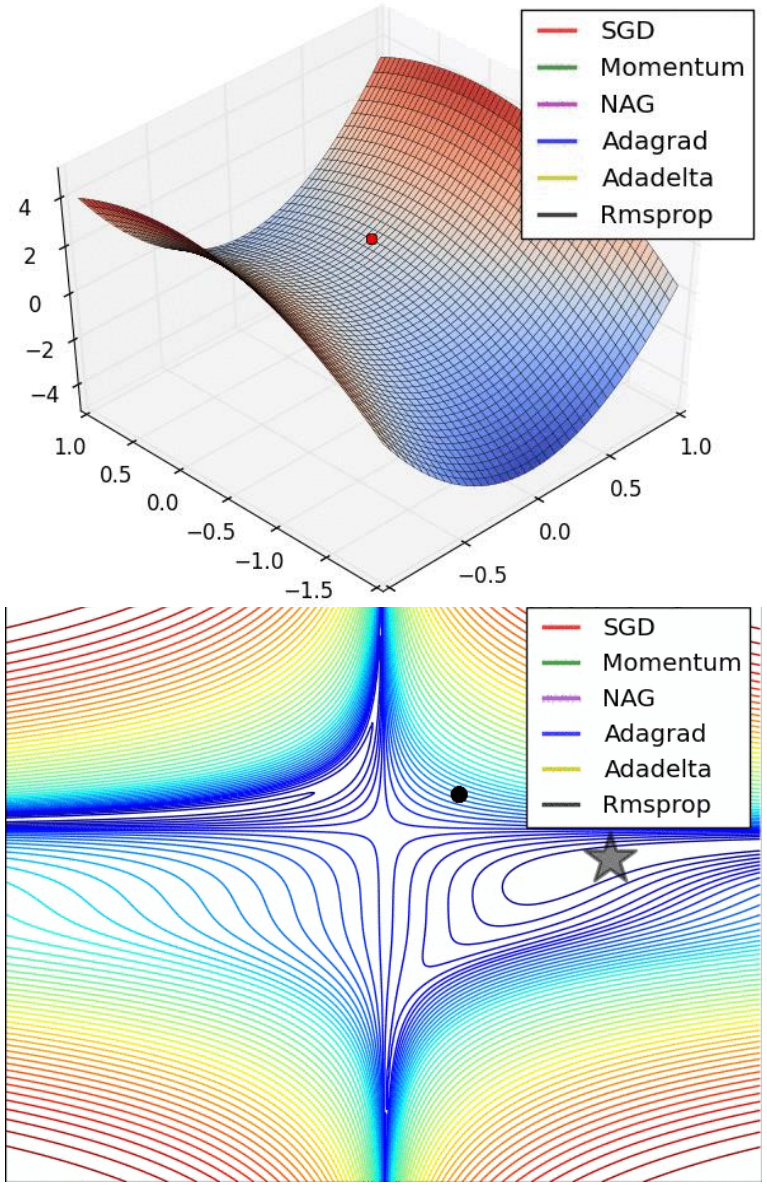
Tricks and tips for applying BP

[Image courtesy: Sebastian Ruder]

Accelerate learning – RMSProp (Hinton, 12)

- RMSProp with Nesterov momentum

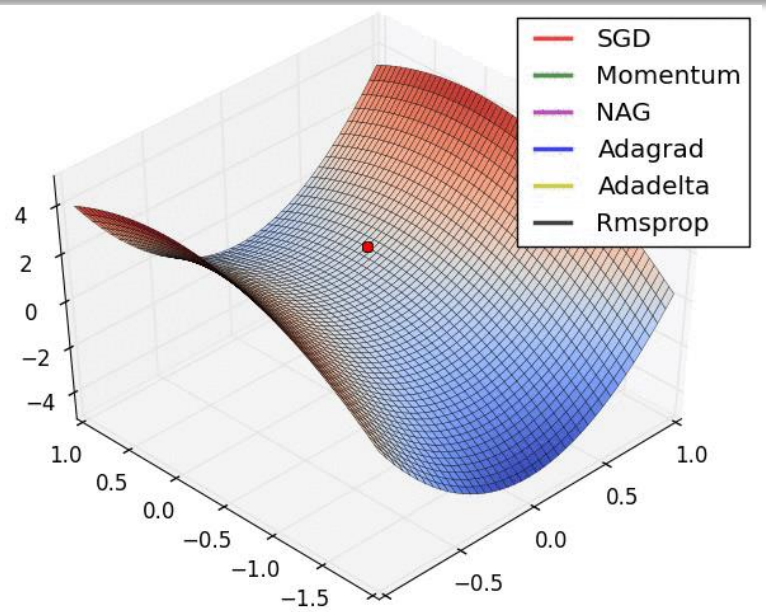
$$\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$$
$$\boldsymbol{g} \leftarrow \nabla_{\tilde{\boldsymbol{\theta}}} J(\tilde{\boldsymbol{\theta}})$$
$$\boldsymbol{\gamma} \leftarrow \beta \boldsymbol{\gamma} + (1-\beta)\boldsymbol{g} \odot \boldsymbol{g}$$
$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \frac{\eta}{\sqrt{\delta + \boldsymbol{\gamma}}} \odot \boldsymbol{g}$$
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$$



Tricks and tips for DNN

[Image courtesy: Sebastian Ruder]

Accelerate learning – Adam (Kingma & Ba, 14)



- The best variant that essentially combines RMSProp with momentum

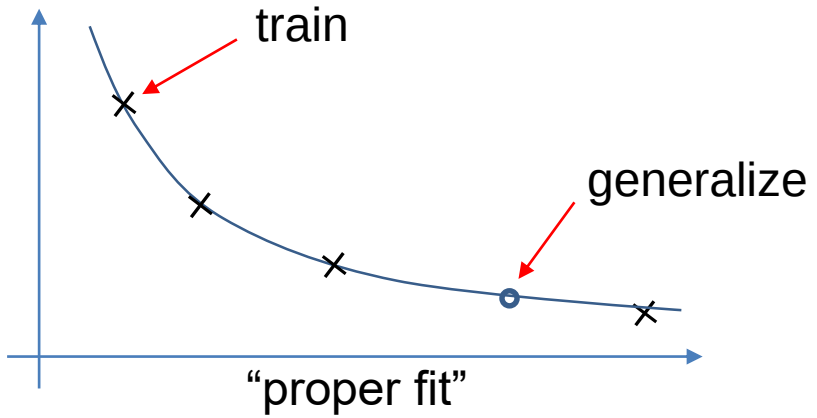
$$t \leftarrow t + 1$$
$$\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$
$$\boldsymbol{s} \leftarrow \beta_1 \boldsymbol{s} + (1 - \beta_1)\boldsymbol{g}$$
$$\boldsymbol{\gamma} \leftarrow \beta_2 \boldsymbol{\gamma} + (1 - \beta_2)\boldsymbol{g} \odot \boldsymbol{g}$$
$$\boldsymbol{s} \leftarrow \frac{\boldsymbol{s}}{1 - \beta_1^t}$$
$$\boldsymbol{\gamma} \leftarrow \frac{\boldsymbol{\gamma}}{1 - \beta_2^t}$$
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \frac{\boldsymbol{s}}{\sqrt{\delta + \boldsymbol{\gamma}}}$$

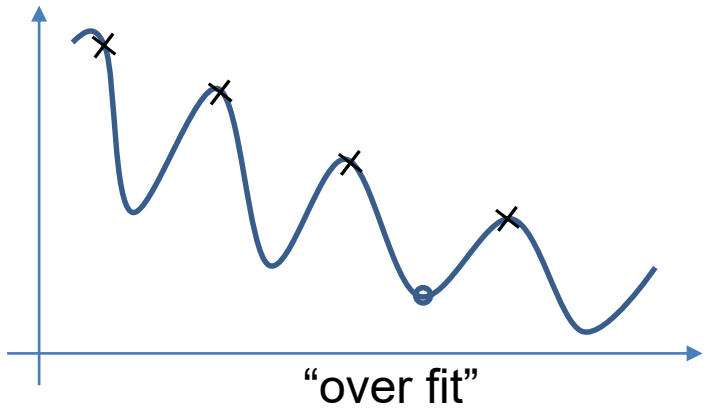
- Suggested default values: $\epsilon = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\delta = 10^{-8}$.
- Should always be used as first choice!

Tricks and tips for DNN

Overfitting problem - quick fixes

- Performance measured by errors on "unseen" data.
- Minimize error alone on training data is not enough
- Causes: too many hidden nodes and overtrained.
- Possible quick fixes:
  - Use cross validation or early stopping, e.g., stop training when validation error starts to grow.
  - Weight decaying: minimize also the magnitude of weights, keeping weights small (since the sigmoid function is almost linear near 0, if weights are small, decision surfaces are less non-linear and smoother)
  - Keep small number of hidden nodes!





Overfit: tendency of the network to "memorize" all training samples, leading to poor generalization

Tricks and tips for DNN

Overfitting problem - regularization

- **L2 regularization**

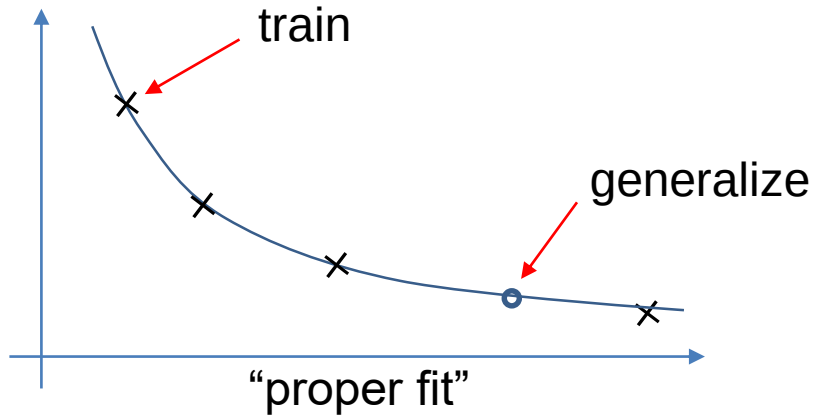
$$\Omega(\boldsymbol{\theta}) = \sum_k \sum_i \left(w_{i,j}^{(k)}\right)^2 = \sum_k \|\boldsymbol{W}^{(k)}\|_F^2$$

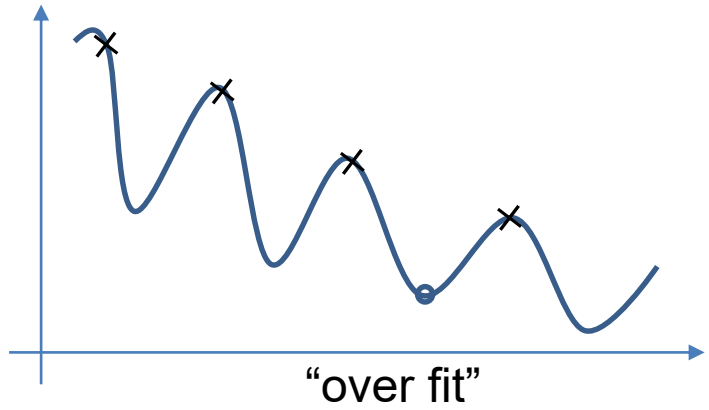
  - Gradient: $\nabla_{\boldsymbol{W}^{(k)}} \Omega(\boldsymbol{\theta}) = 2\boldsymbol{W}^{(k)}$
  - Apply on weights ($\boldsymbol{W}$) only, not on biases ($\boldsymbol{b}$)
  - Bayesian interpretation: the weights follow a Gaussian prior.

- **L1 regularization**

$$\Omega(\boldsymbol{\theta}) = \sum_k \sum_i \|w_{i,j}^{(k)}\|$$

  - Optimization is now much harder – subgradient.
  - Apply on weights ($\boldsymbol{W}$) only, not on biases ($\boldsymbol{b}$)
  - Bayesian interpretation: the weights follow a Laplacian prior.





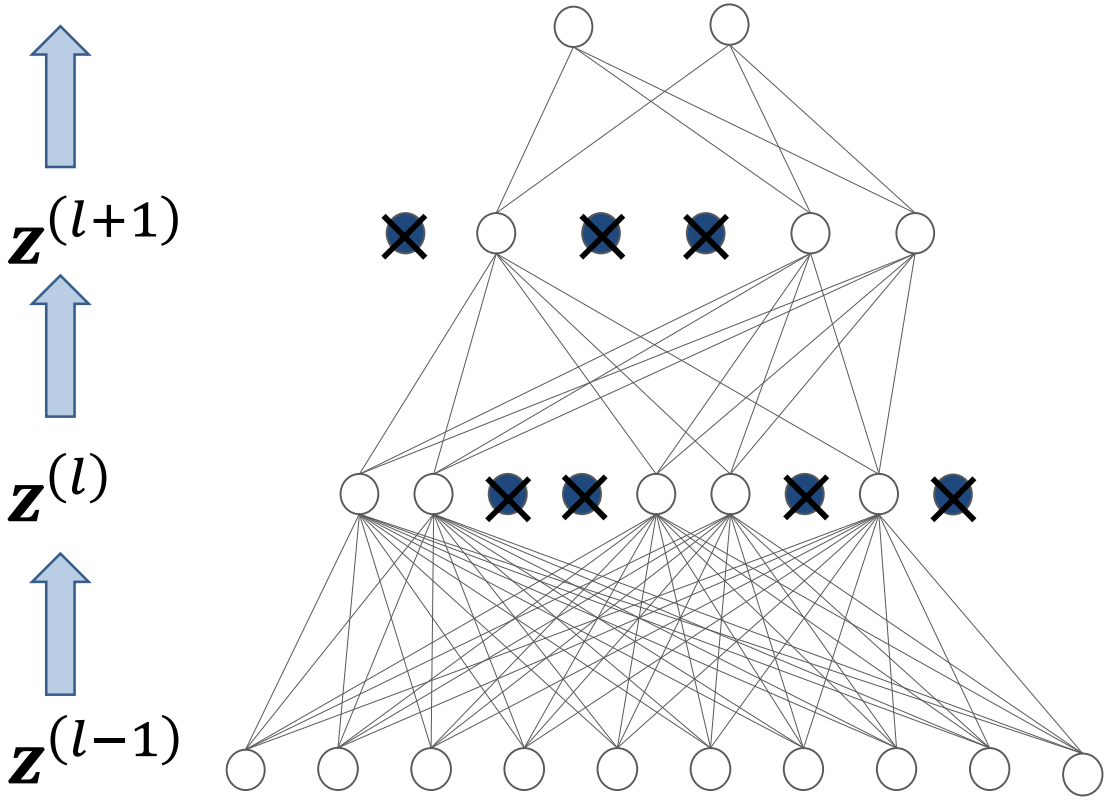
Overfit: tendency of the network to "memorize" all training samples, leading to poor generalization

Tricks and tips for DNN (tự đọc)

Overfitting problem – Dropout (Nitish et. al., 2014)

- Computational inexpensive
- Can be considered a bagged ensemble of an exponential number (2^N) of neural networks.
- Training

$$\boldsymbol{r} \sim \text{Bernoulli}(\mu)$$
$$\tilde{\boldsymbol{z}}^{(l)} = \boldsymbol{z}^{(l)} \odot \boldsymbol{r}$$
$$\boldsymbol{z}^{(l+1)} = f\left(\boldsymbol{w}^{(l)^\top} \tilde{\boldsymbol{z}}^{(l)} + \boldsymbol{b}^{(l)}\right)$$



- Testing

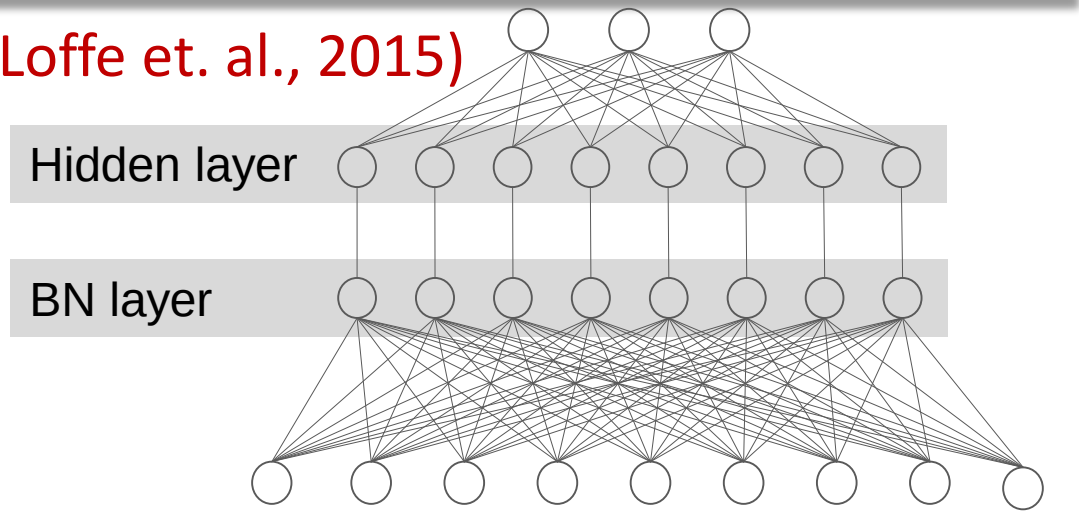
$$\tilde{\boldsymbol{z}}^{(l)} = \boldsymbol{z}^{(l)} \times (1-\mu)$$
$$\boldsymbol{z}^{(l+1)} = f\left(\boldsymbol{w}^{(l)^\top} \tilde{\boldsymbol{z}}^{(l)} + \boldsymbol{b}^{(l)}\right)$$

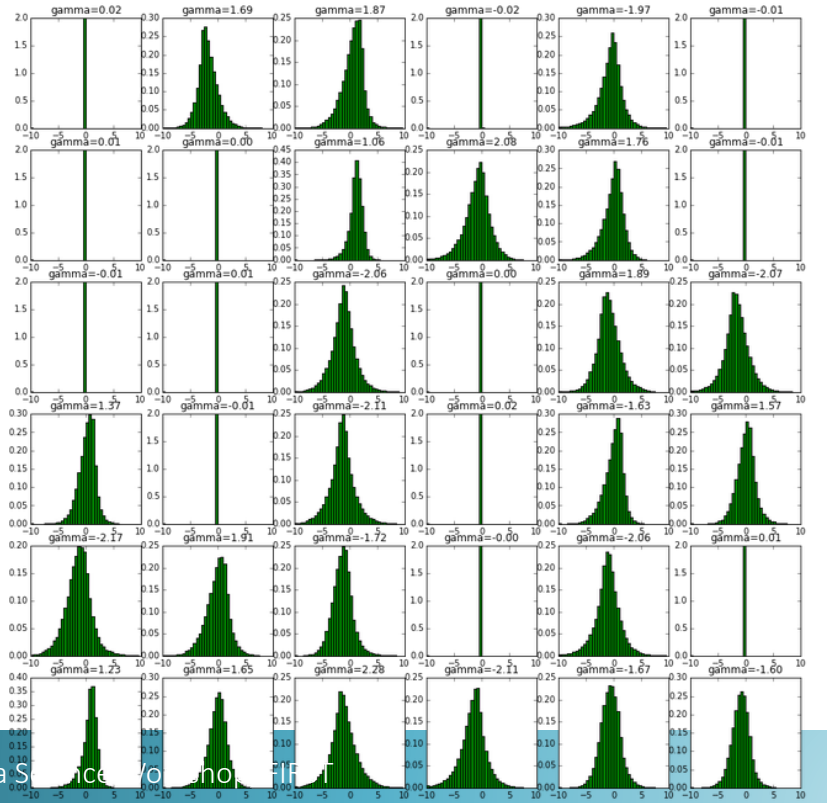
Tricks and tips for DNN (tự đọc)

[Image courtesy: guillaumebrg]

Overfitting problem -Batch Normalization (Loffe et. al., 2015)

- Can help: better optimization, better generalization (regularization).
- Allow to use higher learning rate because of acting as a regularizer.
- It is actually not an optimization algorithm at all. It is a method of *adaptive reparameterization*, motivated by the difficulty of training very deep models.
- Update all of the layers simultaneously using gradient can cause unexpected results, as many functions compose and change together.





Tricks and tips for DNN (tự đọc)

Overfitting problem - Batch Normalization (Ioffe et. al., 2015)

- Apply before activation: $f(\boldsymbol{w}^\top \boldsymbol{h} + \boldsymbol{b}) \rightarrow f\big(BN(\boldsymbol{w}^\top \boldsymbol{x} + \boldsymbol{b})\big)$
- Make a minibatch has mean 0 and variance 1.

$$BN_{init}(\boldsymbol{x}) = \frac{\boldsymbol{x} - \boldsymbol{\mu}_B}{\sqrt{\boldsymbol{\sigma}_B^2 + \epsilon}}$$

- Undo the batch norm by multiplying with a new scale parameter $\boldsymbol{\gamma}$ and adding a new shift parameter $\boldsymbol{\beta}$ (note that $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are learnable)

$$BN(\boldsymbol{x}) = \boldsymbol{\gamma}\left(\frac{\boldsymbol{x} - \boldsymbol{\mu}_B}{\sqrt{\boldsymbol{\sigma}_B^2 + \epsilon}}\right) + \boldsymbol{\beta}$$

- Hai biến thể quan trọng của deep NN
  - Autoencoder
  - Convolution neural networks (CNN)

What is an autoencoder?

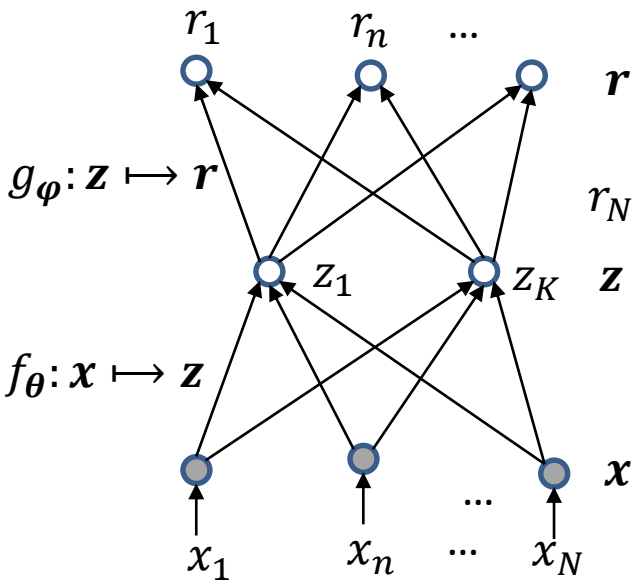
- Simply a neural network that tries to copy its input to its output.
  - Input $\boldsymbol{x} = [x_1, x_2, \dots, x_N]^\top$
  - An encoder function f parameterized by $\boldsymbol{\theta}$
  - A coding representation $\boldsymbol{z} = [z_1, z_2, \dots, z_K]^\top = f_{\boldsymbol{\theta}}(\boldsymbol{x})$
  - A decoder function g parameterized by $\boldsymbol{\varphi}$
  - An output, also called reconstruction

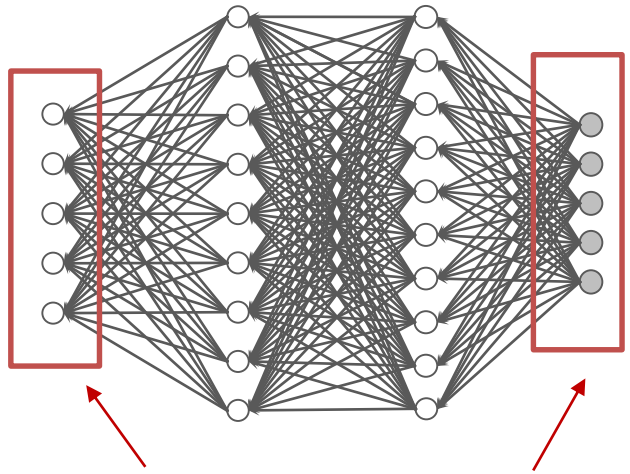
$$\boldsymbol{r} = [r_1, r_2, \dots, r_N]^\top = g_{\boldsymbol{\varphi}}(\boldsymbol{z}) = g_{\boldsymbol{\varphi}}\big(f_{\boldsymbol{\theta}}(\boldsymbol{x})\big)$$

  - A loss function $\mathcal{J}$ that computes a scalar $\mathcal{J}(\boldsymbol{x}, \boldsymbol{r})$ to measure how good of a reconstruction $\boldsymbol{r}$ of the given input $\boldsymbol{x}$, e.g., mean square error loss:

$$\mathcal{J}(\boldsymbol{x}, \boldsymbol{r}) = \sum_{n=1}^{N} (x_n - r_n)^2$$

  - Learning objective: $\underset{\boldsymbol{\theta}, \boldsymbol{\varphi}}{\operatorname{argmin}}\, \mathcal{J}(\boldsymbol{x}, \boldsymbol{r})$

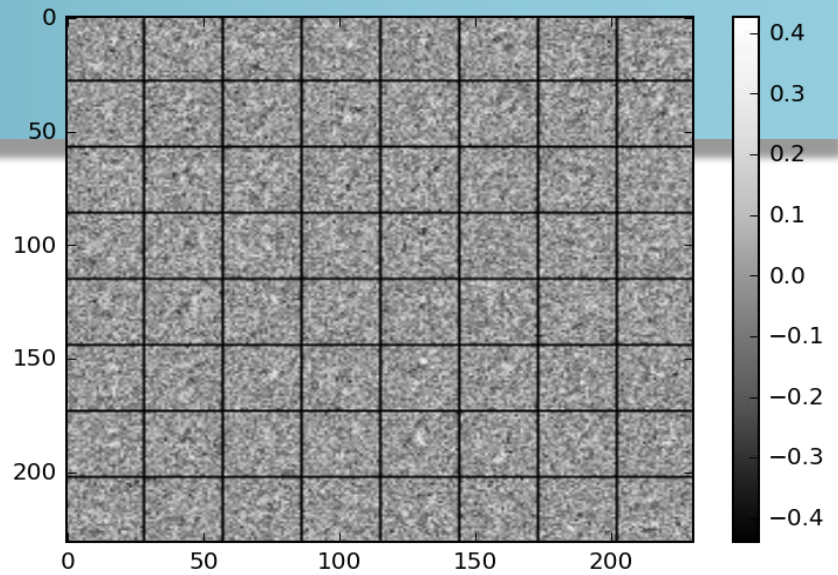




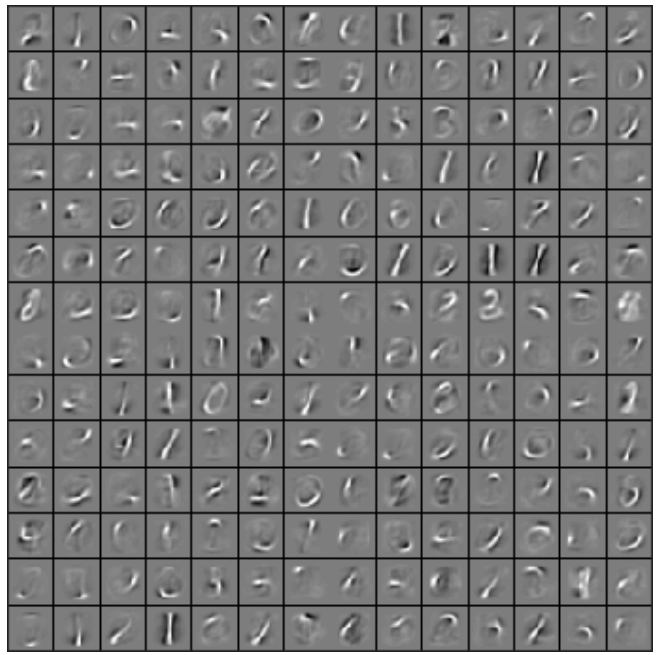
Giá trị đầu vào và ra là giống nhau

Autoencoders

- Hidden layers $K < N$: under complete, otherwise overcomplete.
- Shallow representation works better with overcomplete case.
  - Warning: overfit without regularization; why? one can always recover the exact input!
- Key is to regularize the autoencoder!
  - Sparse coding $\boldsymbol{z}$: $\mathcal{J}(\boldsymbol{x}, \boldsymbol{r}) + \Omega(\boldsymbol{z})$
  - Predictive sparse decomposition
  - Contractive autoencoder
  - Denoising autoencoder



unregularized – useless!

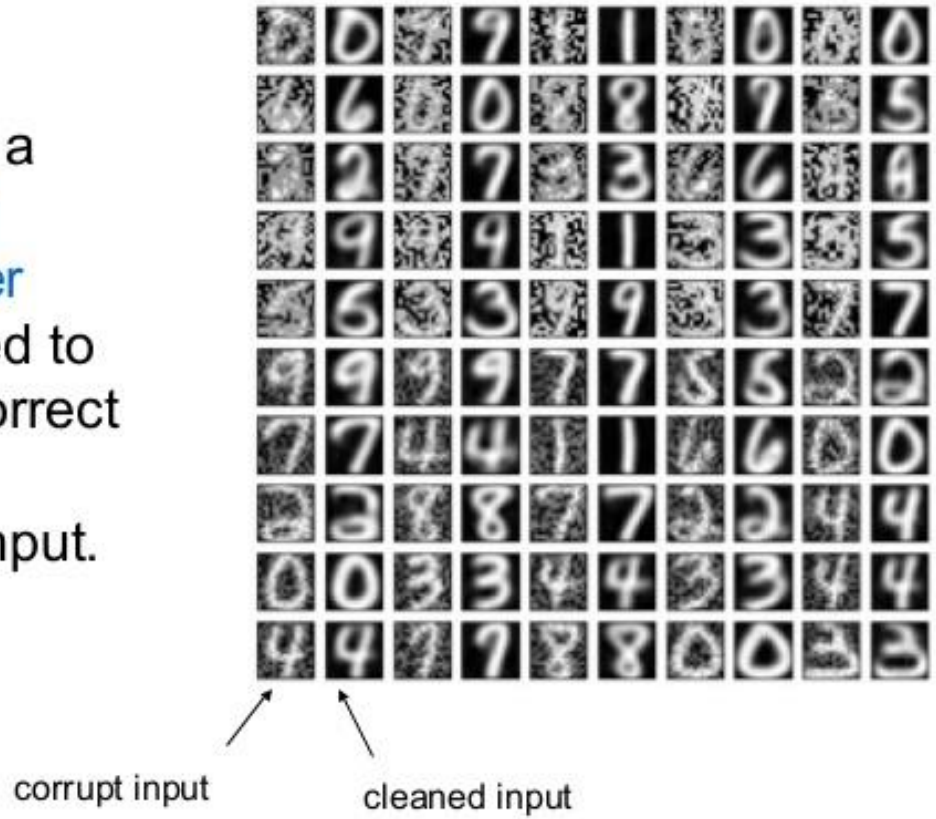


Sparsity regularization

Khử nhiễu ảnh

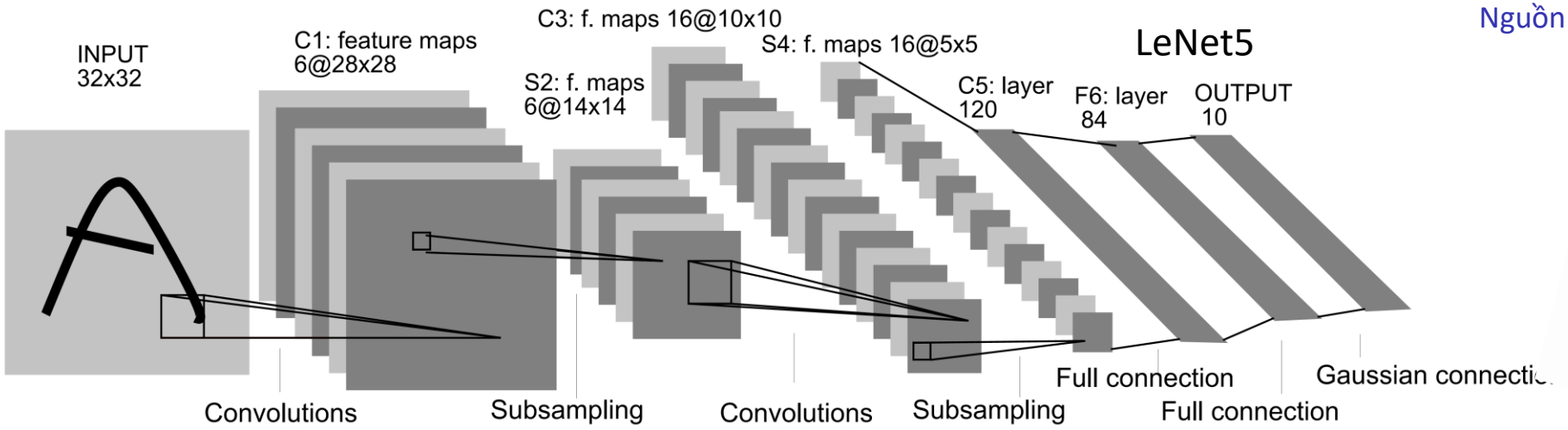
Denoising autoencoder

The image shows how a "denoising" autoencoder may be used to generate correct input from corrupted input.

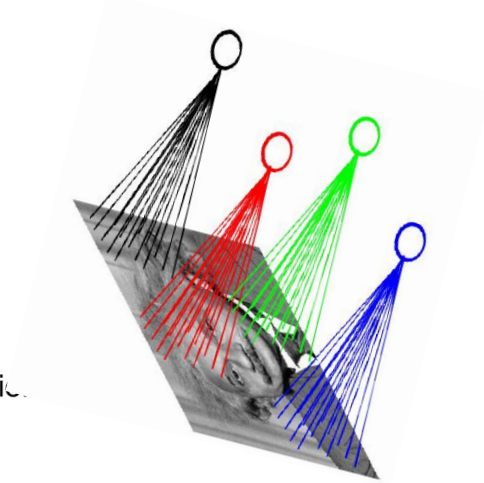


[Vincent et al., JMLR'10]

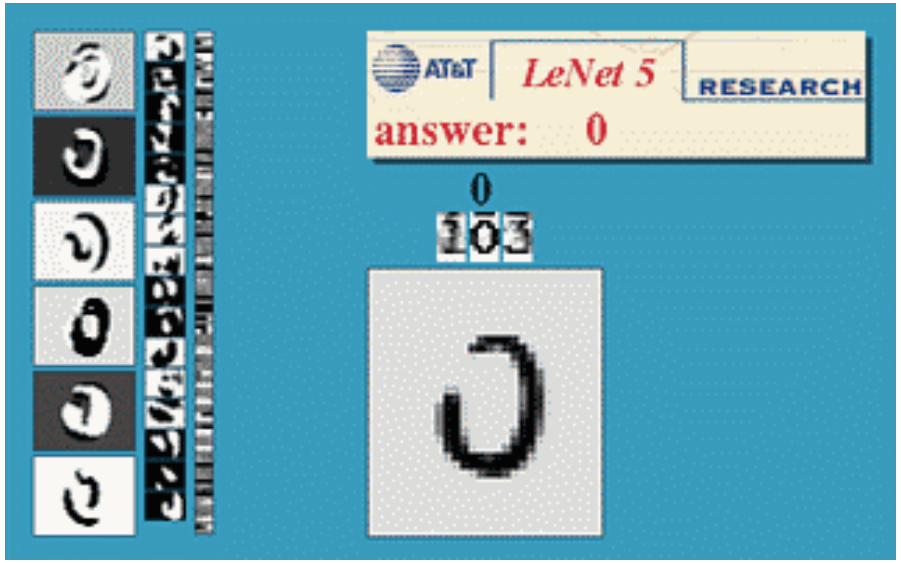
Convolutional Neural Networks (CNN, ConvNets)



Nguồn: http://yann.lecun.com/

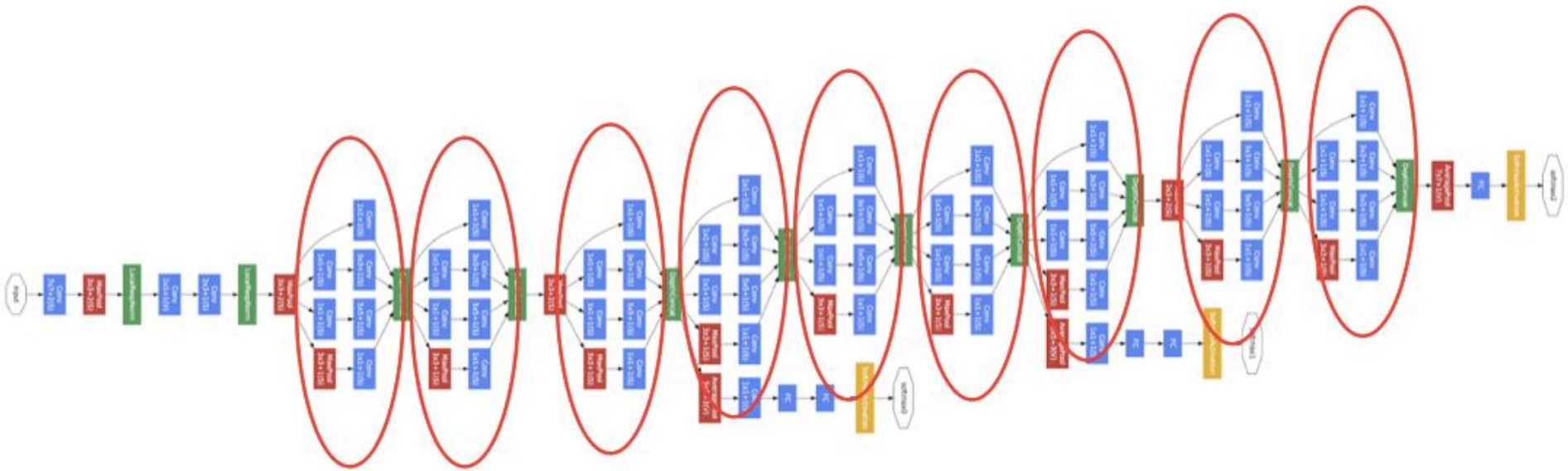


- Motivation: vision processing in the brain is fast
  - Simple cells detect local features
  - Complex cells pool local features
- Technical: sparse interactions and sparse weights within a smaller kernel (e.g., 3x3, 5x5) instead of the whole input -> reduce #params.
  - Parameter sharing: a kernel with same set of weights while applying onto different location
  - Translation invariance.



Networks become thinner and deeper

- A glimpse of GoogLeNet 2014: deeper and thinner …

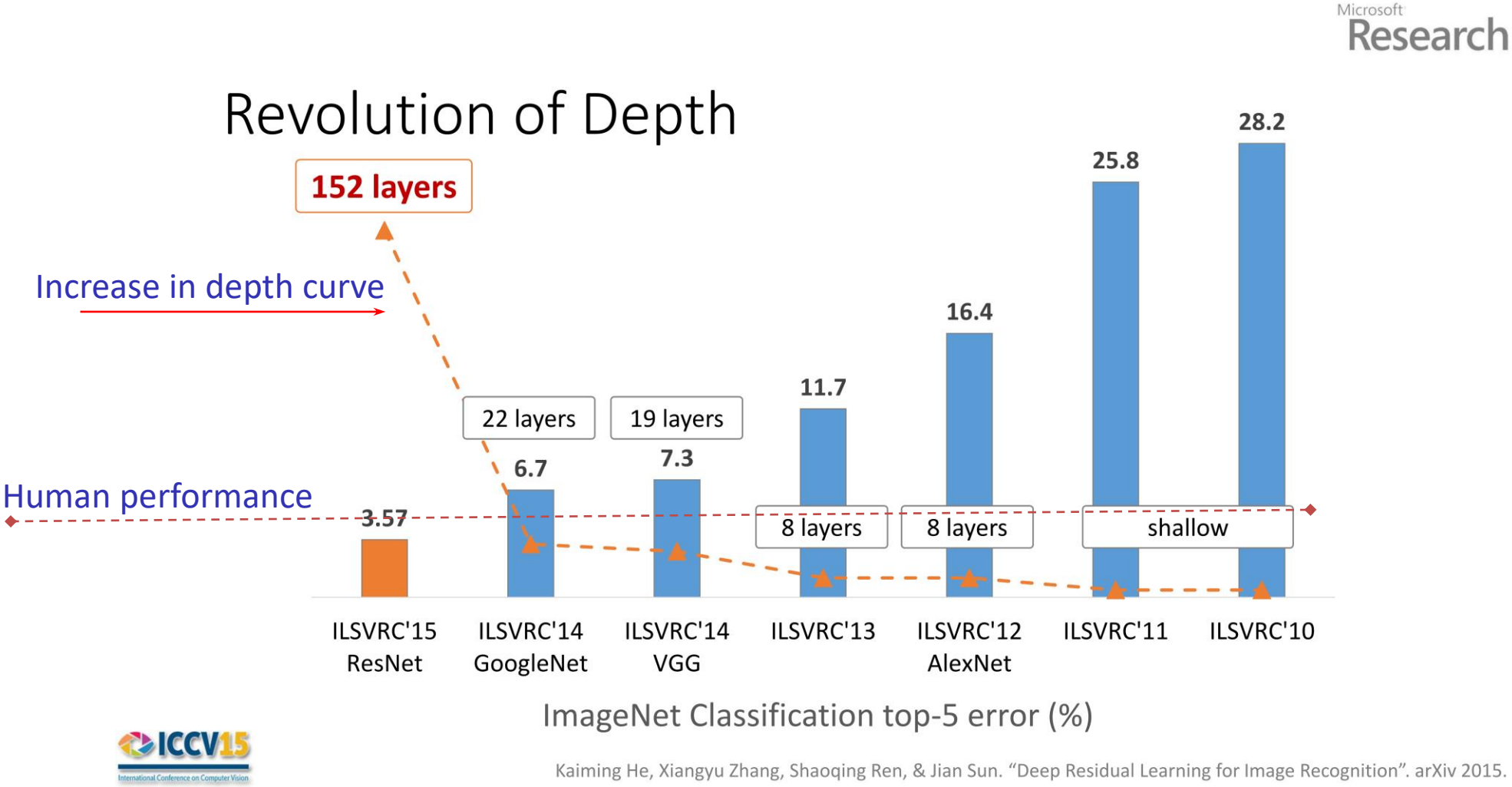


9 **Inception** modules

Network in a network in a network...

Convolution
Pooling
Softmax
Other

- And … ResNet 2015 beats human performance (5.1%)!



ICCV15 International Conference on Computer Vision

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

So what help SOTA results emerge?

- Larger models with new training techniques:
  - Dropout, Maxout, Batchnorm, Maxnorm, …
- Large ImageNet dataset [Fei-Fei et al. 2012]
  - 1.2 million training samples
  - 1000 categories
- Fast graphical processing units (GPU)
  - Capable of 1 trillion operations per second

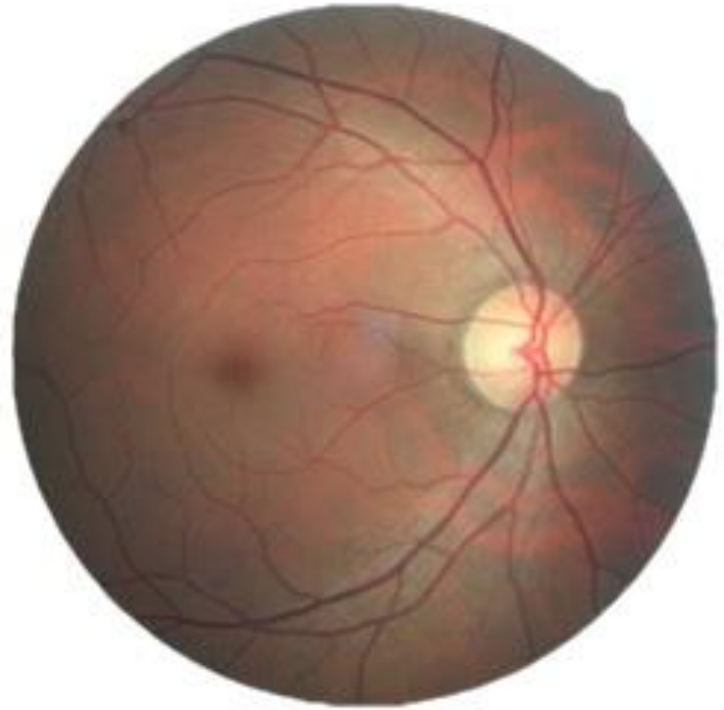






ConvNets won all Computer Vision challenges recently

Galaxy



Diabetes recognition from retina image

- CNN đã gặt hái rất nhiều thành công (nhất là trong computer vision)
- Tuy nhiên nó đã đến lúc bão hòa – nếu có cải thiện, thì chỉ là vụn vặt.
- Next? Mô hình sinh dữ liệu nhiều tầng (deep generative models)

Outline

- What is deep learning and few application examples.
- Deep neural networks
  - Key intuitions and techniques
  - Deep Autoencoders
  - Convolutional networks (CNN)
- Neural embedding
  - Word2vec, recent embedding methods
- Deep Generative Models
- TensorFlow
- Few practical tips

Neural Embedding

"You shall know a word by the company it keeps"

J.R. Firth 1957

Tạm dịch: chúng ta có thể suy diễn nghĩa của một từ nếu biết những từ vựng khác hay dùng kèm với nó.

Advances in NLP

Language model → **Class-based language model** → **Neural Language models**

- (aka n-gram model)
- distributions over sequence of symbolic tokens in a natural language.
- Use probability chain rule.
- Estimating by counting the occurrence of n-gram

- Reduce computation
- Cluster words into categories
- Use category ID for context
- Symbolic representation
- With one-hot representation, the distance between any two words will always be exactly $\sqrt{2}$!

- Use distributed representation for words.
- Similar words will stay closer.

context

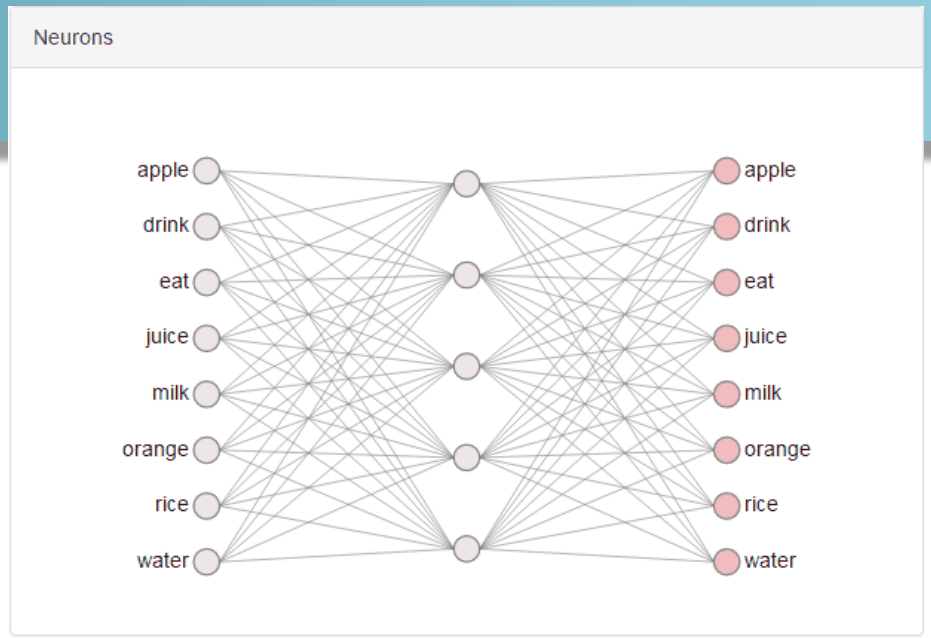
$$\Pr(x_1, \dots, x_n) = \Pr(x_1, \dots, x_{k-1}) \prod_{t=k}^{n} \Pr(x_t \mid x_{t-1}, \dots, x_{t-k+1})$$

$$P(\text{“}Bầu\ trời\ màu\ xanh\text{”}) = P_3(\text{“}Bầu\ trời\ màu\text{”}) P_3(\text{“}trời\ màu\ xanh\text{”}) / P_2(\text{“}trời\ màu\text{”})$$

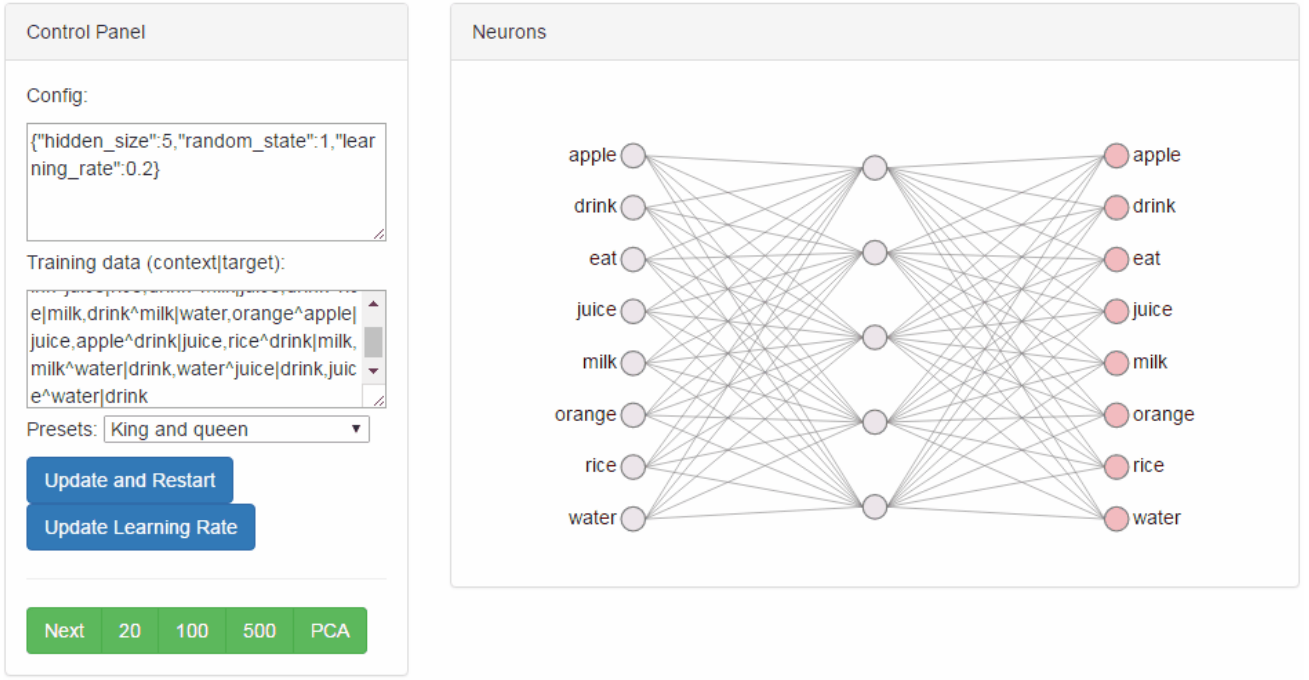
Neural embedding

Word Embedding [Mikolov et al., 2013]

- Representing meaning of words
  - Important for all NLP tasks.
  - WordNet
  - Discrete/one-hot representation
  - Methods: LSA, MDS, LDA
- **Word2vec** (Mikolov et al. 2013)
  - Distributed representation for word.
  - What does it mean? Learn a real-valued vector for each word.
  - Small distances induce similar words.
  - Compositionality induces real-valued vector representation for phrases/sentences.
  - A simple application of NN with exactly 2 layers!
- Fun: https://ronxin.github.io/wevi/



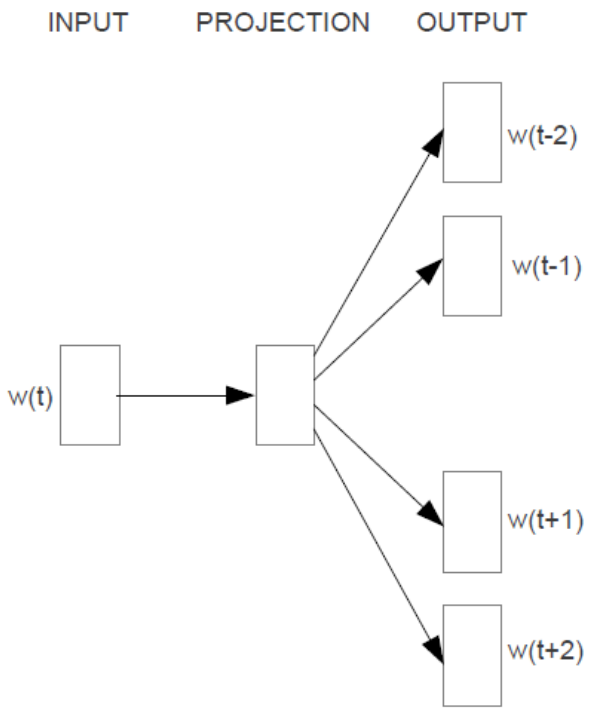
Neural embedding



Neural embedding

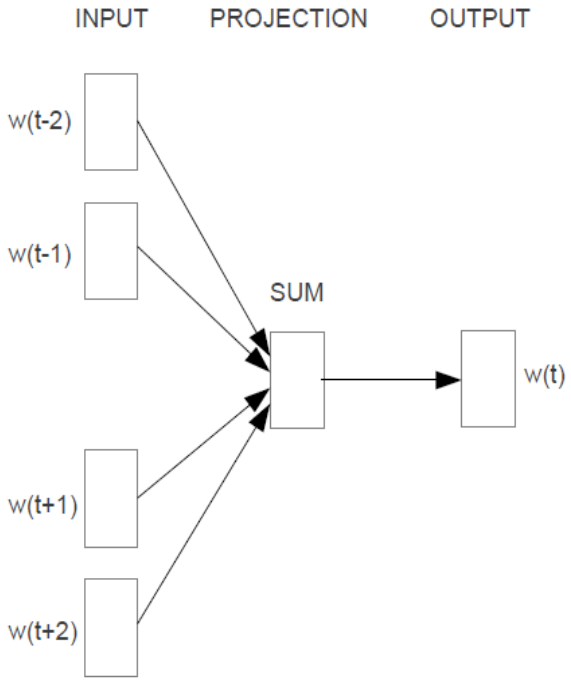
Word Embedding

- Skip n-gram: predict context based on word.
- Continuous-bag-of-word (CBOW): predict word based on the context.



Skip-gram

- Negative sampling: speed up training by sub-sampling (e.g., frequent words).
- Hierarchical softmax: deal with large vocabulary: $O(V)$ to $O(\log V)$.

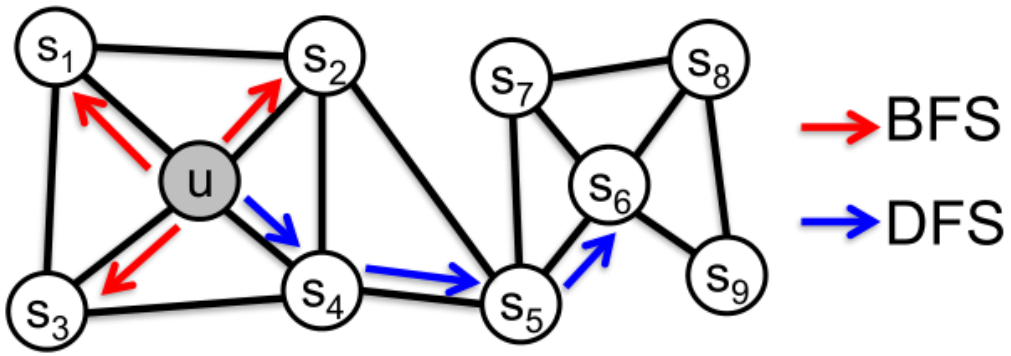


CBOW

Node2vec

[Grover and Leskovec, KDD'16]

- **Motivation:**
  - Learn continuous feature representations for nodes in large networks (e.g., social networks)
  - Define a flexible notion of a node's neighbourhood
- **Key ideas:**
  - Consider a node in a network as a word in a document.
  - Apply Word2vec to learn a distributed representation for a node
  - Question: "How can we define the context (neighbourhood) for a node?"
    - *Homophily*: nodes are organized based on communities they belong to
    - *Structural equivalence*: nodes are organized based on their structural roles
  - Node2vec develops a *biased random walk* approach to explore both types of neighbourhood of a given node



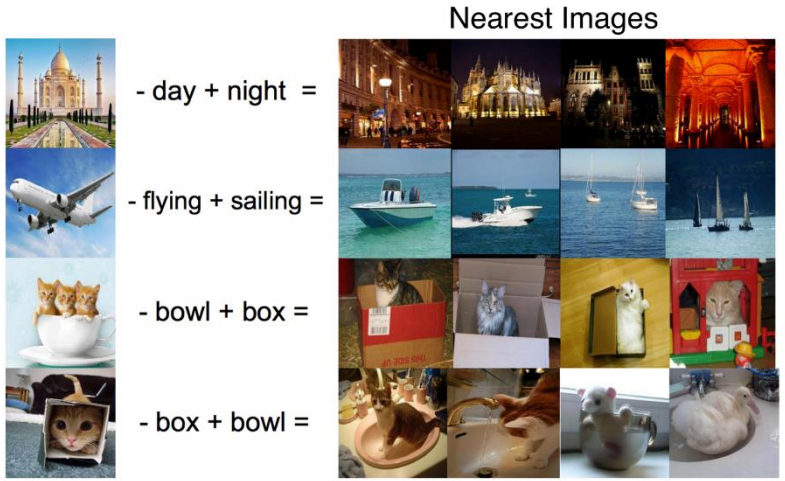
Homophily: neighbourhood(u)={s1, s2, s3, s4}

Structural equivalence: neighbourhood(u)={s6}

Neural embedding

Applications

- Combining n-gram and neural language models.
- Neural Machine Translation.
- Question-Answering system.
- End-to-End systems.
- Multimodal linguistic regularities



(Kiros, Salakhutdinov, Zemel, TACL 2015)

Recent Methods	
word2vec (Mikolov, et al. 2013)	distributed representation for words
doc2vec (Le an Mikolov, 2014)	distributed representation of sentences and documents
topic2vec (Niu and Dai, 2015)	distributed representation for topics
item2vec (Barkan and Koenigstein, 2016)	distributed representation of items in recommender systems
med2vec (Choi et al., 2016)	distributed representations of ICD codes
node2vec (Grover and Leskovec, 2016)	distributed representation for nodes in a network
paper2vec (Ganguly and Pudi, 2017)	distributed representations of textual and graph-based information
sub2vec (Adhikari et al., 2017)	distributed representation for subgraphs
cat2vec (Wen et al., 2017)	distributed representation for categorical values
fis2vec (2017)	distributed representation for frequent itemsets

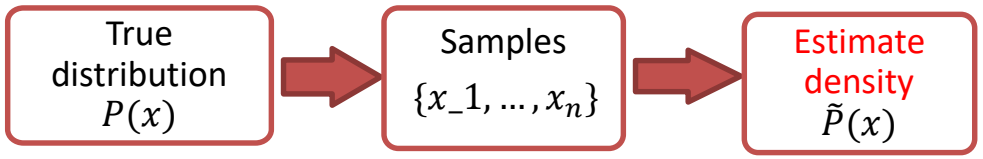
Outline

- What is deep learning and few application examples.
- Deep neural networks
  - Key intuitions and techniques
  - Deep Autoencoders
  - Convolutional networks (CNN)
- Neural embedding
  - Word2vec, recent embedding methods
- Deep Generative Models
- TensorFlow
- Few practical tips

Deep generative models

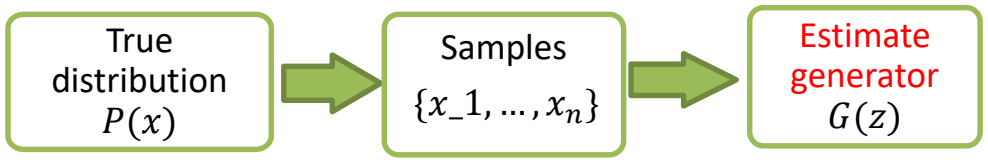
Deep generative models from density estimation view

Explicit density estimation



- Assumption: the true, but unknown, data distribution: $P(x)$
- Data: $\mathrm{D} = \{x_1, x_2, \dots, x_n\}$ where $x_i \sim P(x)$.
- Goal: estimate density $\tilde{P}(x)$ with data D such that $\tilde{P}(x)$ is as 'close' to $P(x)$ as possible.

Implicit density estimation



- Assumption: the true, but unknown, data distribution: $P(x)$
- Data: $\mathrm{D} = \{x_1, x_2, \dots, x_n\}$ where $x_i \sim P(x)$.
- Goal: estimate generator $G(z)$ with data D such that samples generated by $G(z)$ is 'indistinguishable' from samples generated by $P(x)$ where $z \sim \mathbb{N}(0,1)$.

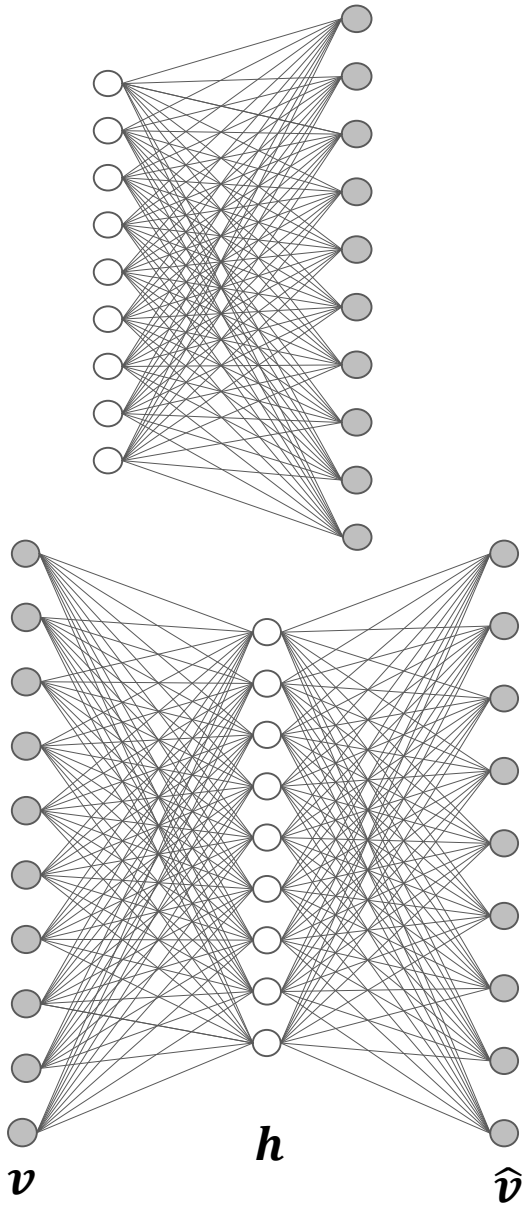
[Goodfellow, NIPS Tuts 2016]

Deep Generative Models

- Explicit Density Estimation
  - Restricted Boltzmann Machines
  - Deep Boltzmann Machines
- Implicit Density Estimation
  - Variational Autoencoders
  - Generative Adversarial Nets

Restricted Boltzmann Machines (RBM)

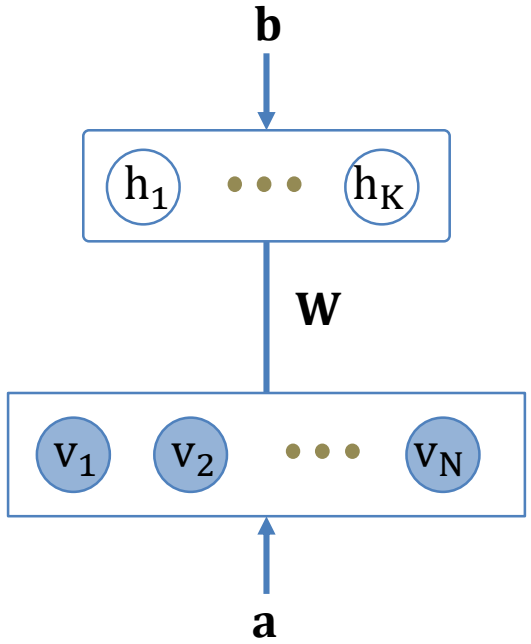
- An important building block for deep generative models.
- It is (undirected) graphical models with two layers.
- Unsupervised learning to:
  - Discover latent features of data
  - Provide new representations of data for other classifiers such as GLM, SVM, Random Forest, …
- Stacking to construct higher-level models
  - Deep Belief Networks
  - Deep Boltzmann Machines
  - Deep AutoEncoders
- Can also be viewed as a stochastic autoencorder.

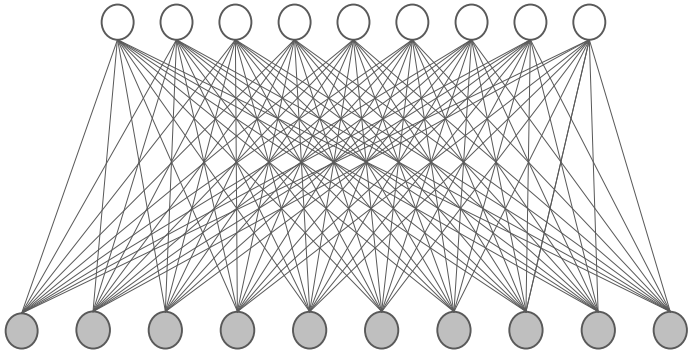


RBM – Model Representation

- Input: data ($\boldsymbol{v}$)
- Output: new representation ($\boldsymbol{h}$)

- Variables
  - $\boldsymbol{v} = [v_n]_N \in \{0,1\}^N$: visible/observed
  - $\boldsymbol{h} = [h_k]_K \in \{0,1\}^K$: hidden/latent
- Parameters
  - $\boldsymbol{a} = [a_n]_N \in \mathbb{R}^N$, $\boldsymbol{b} = [b_k]_K \in \mathbb{R}^K$: visible and hidden biases
  - $\boldsymbol{W} = [w_{nk}]_{N \times K} \in \mathbb{R}^{N \times K}$: connection weights





RBM - Model Representation (tự đọc)

- Energy function

$$E(\boldsymbol{v}, \boldsymbol{h}; \psi) = -(\boldsymbol{a}^\top \boldsymbol{v} + \boldsymbol{b}^\top \boldsymbol{h} + \boldsymbol{v}^\top \boldsymbol{W} \boldsymbol{h})$$

- Tractable and fast factorization

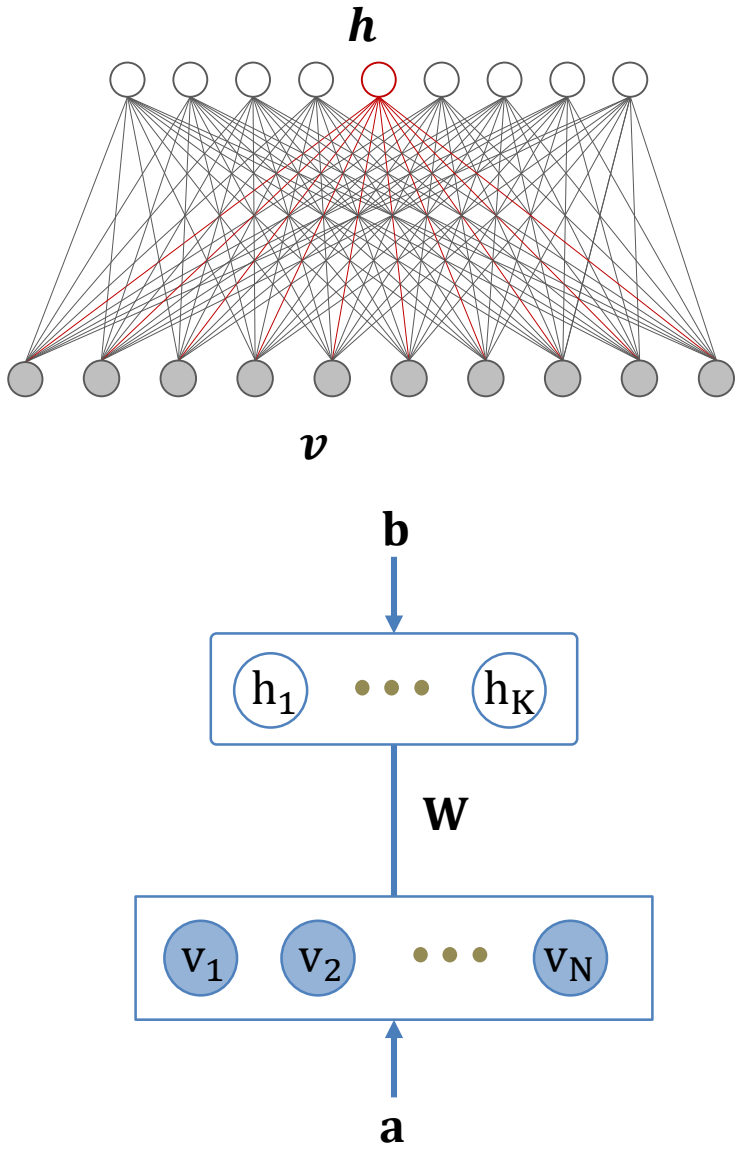
$$p(\boldsymbol{v}|\boldsymbol{h}; \psi) = \prod_{n=1}^{N} p(v_n|\boldsymbol{h}; \psi)$$

$$p(\boldsymbol{h}|\boldsymbol{v}; \psi) = \prod_{k=1}^{K} p(h_k|\boldsymbol{v}; \psi)$$

- Conditional probability

$$p(h_k = 1 \mid \boldsymbol{v}; \psi) = \text{sig}(b_k + \boldsymbol{v}^\top \boldsymbol{w}_{.k})$$

$$p(v_n = 1 \mid \boldsymbol{h}; \psi) = \text{sig}(a_n + \boldsymbol{w}_{n.} \boldsymbol{h})$$



RBM - Parameter Estimation (tự đọc)

- Data log-likelihood

$$\log \mathcal{L}(\boldsymbol{v};\psi) \& = \log p(\boldsymbol{v};\psi)$$
$$= \log \sum_{\boldsymbol{h}} p(\boldsymbol{v};\boldsymbol{h};\psi)$$

- Rewrite the joint distribution:

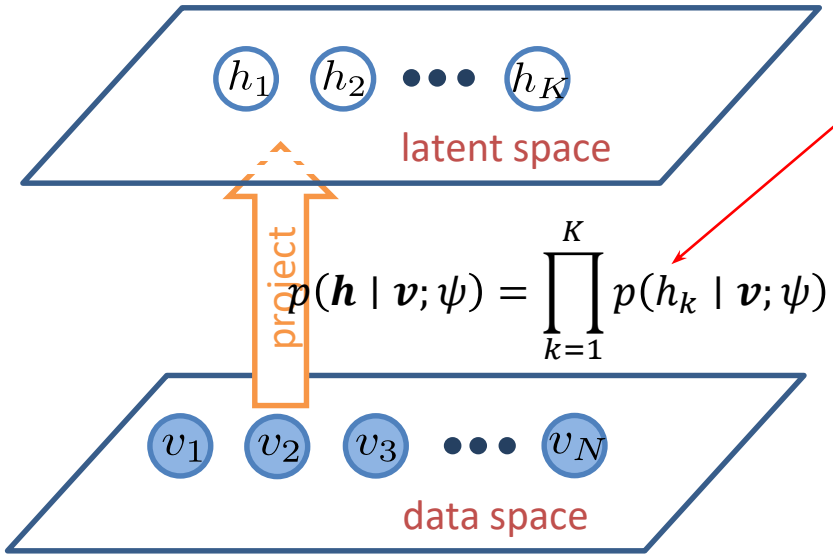
$$p(\boldsymbol{v},\boldsymbol{h};\psi) = \exp\{-E(\boldsymbol{v},\boldsymbol{h};\psi) - A(\psi)\}$$

Exploiting property of exponential family

$$p(\boldsymbol{x};\boldsymbol{\theta}) = h(\boldsymbol{x}) \exp[\boldsymbol{\theta}^{\top}\boldsymbol{\phi}(\boldsymbol{x}) - A(\boldsymbol{\theta})]$$

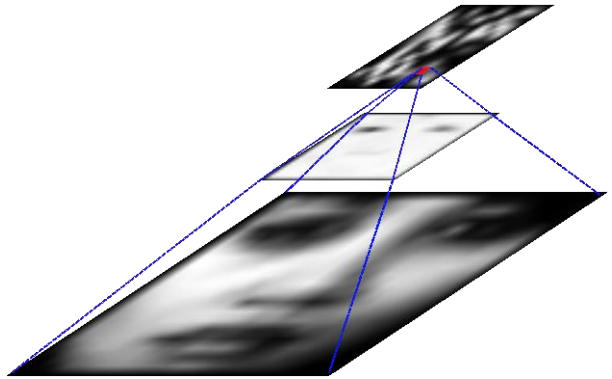
$$\frac{\partial}{\partial\psi}\log \mathcal{L}(\boldsymbol{v};\psi) = \underbrace{\mathbb{E}_{p(\boldsymbol{v},\boldsymbol{h};\psi)}\left[\frac{\partial E(\boldsymbol{v},\boldsymbol{h};\psi)}{\partial\psi}\right]}_{\text{model expectation, this is the problematic term!}} - \underbrace{\mathbb{E}_{p(\boldsymbol{h}|\boldsymbol{v};\psi)}\left[\frac{\partial E(\boldsymbol{v},\boldsymbol{h};\psi)}{\partial\psi}\right]}_{\text{data expectation, can be computed easily!}}$$

RBM - Latent Representation

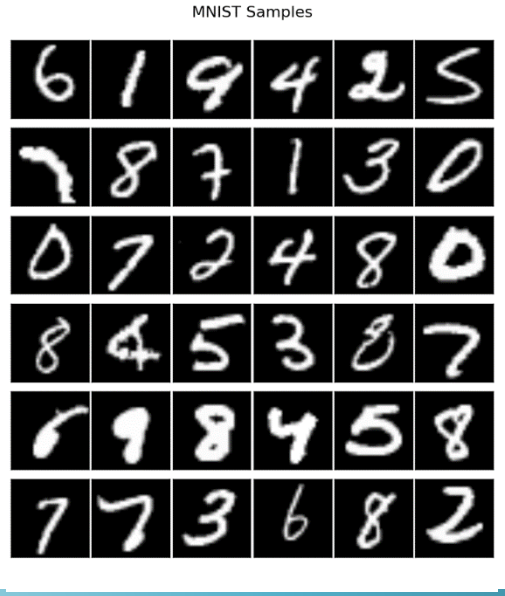


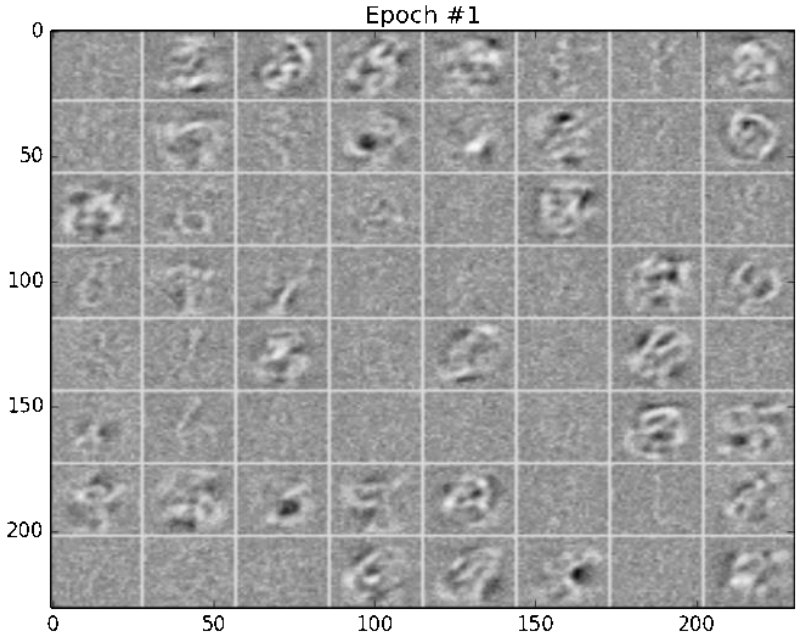
$$p(h_k = 1 \mid \boldsymbol{v}; \psi) = \text{sig}(b_k + \boldsymbol{v}^\top \boldsymbol{w}_{\cdot k})$$

efficient posterior inference



data





latent representation

Applications

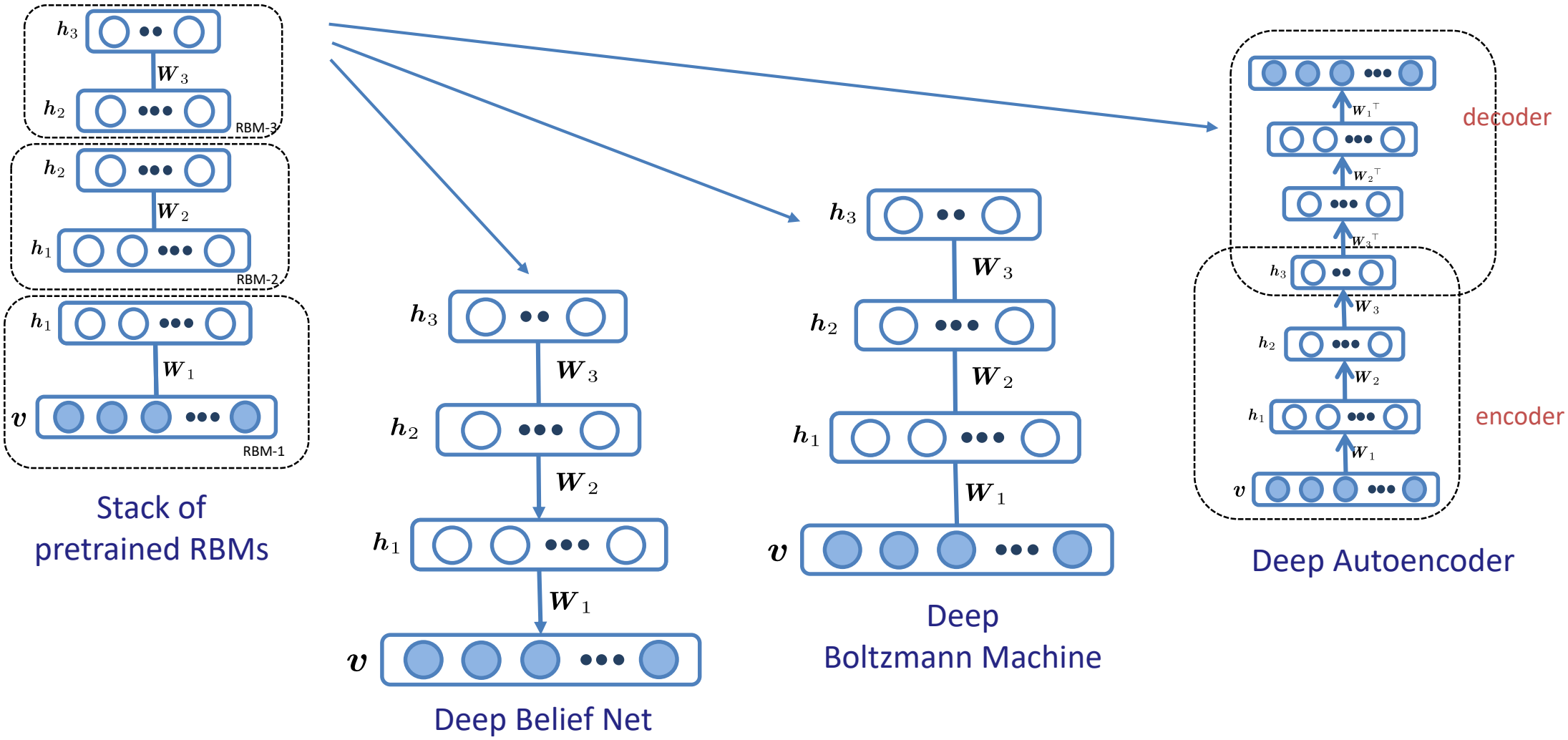
- Image, video modelling
- Speech recognition
- Time series
- Topic modelling
- Recommender systems
- Information retrieval
- Natural language processing
- Neuroimaging
- Deep networks
- Mixed data modelling
- Latent patient profile modelling
- Image retrieval
- Tensor data modelling

- Covariance RBM (cRBM) (Hinton, 2010)
- Mean-covariance RBM (mcRBM) = cRBM + Gaussian RBM (Ranzato et al., 2010)
- The mean-product of student's t-distribution models (mPoT) (Ranzato et al, 2010)
- Spike-and-slab RBM (ssRBM) (Courville et al.,2011)
- Convolutional RBMs (Desjardins and Bengio, 2008)

- Beta-Bernoulli process RBMs (HongLak Lee et al., 2012)
- An infinite RBM (Hugo Larochelle, 2015)
- Infinite RBMs (Jun Zhu group)
- Privacy-preserving RBMs (Li et al., 2014)
- Mixed-Variate RBM, ordinal Boltzmann machine, cumulative RBM (Truyen et al, 12-14)
- Graph-induced RBMs (Tu et. al, 2015)
- Non-negative RBMs (Tu et. al, 2013)

Going Deeper

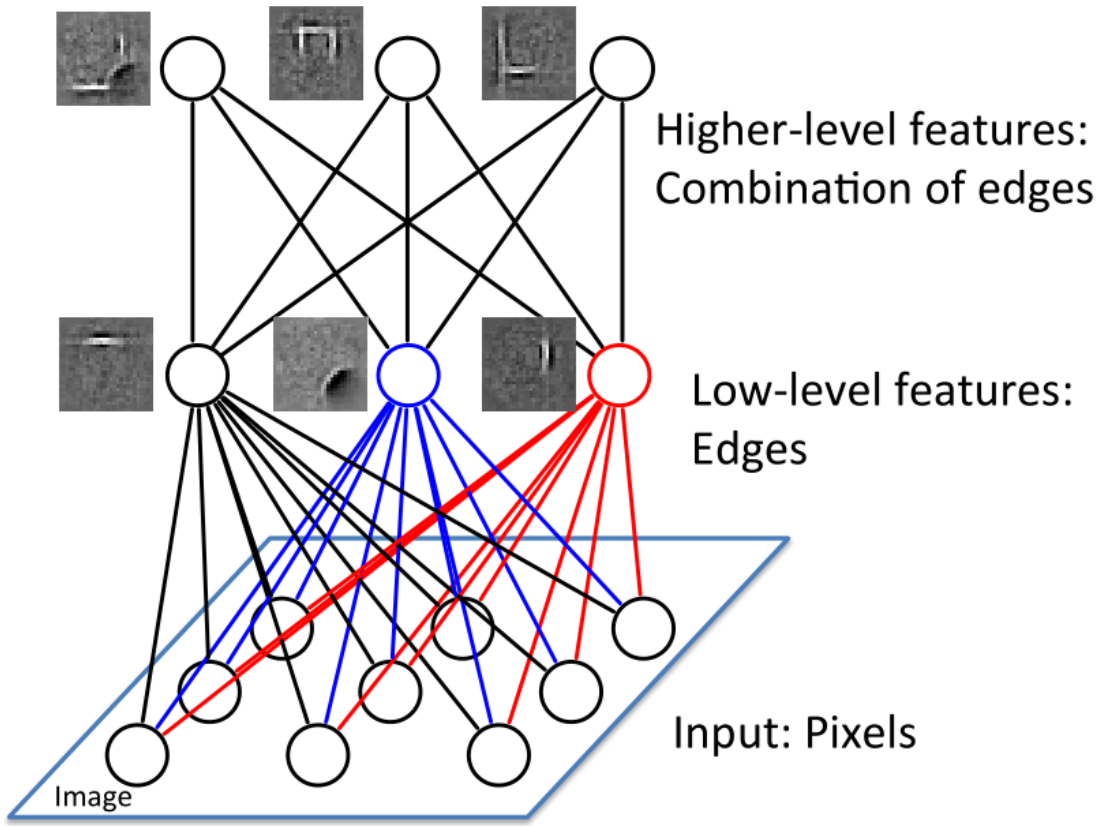
- Layer-wise pre-training for DBN, DBM, DAE

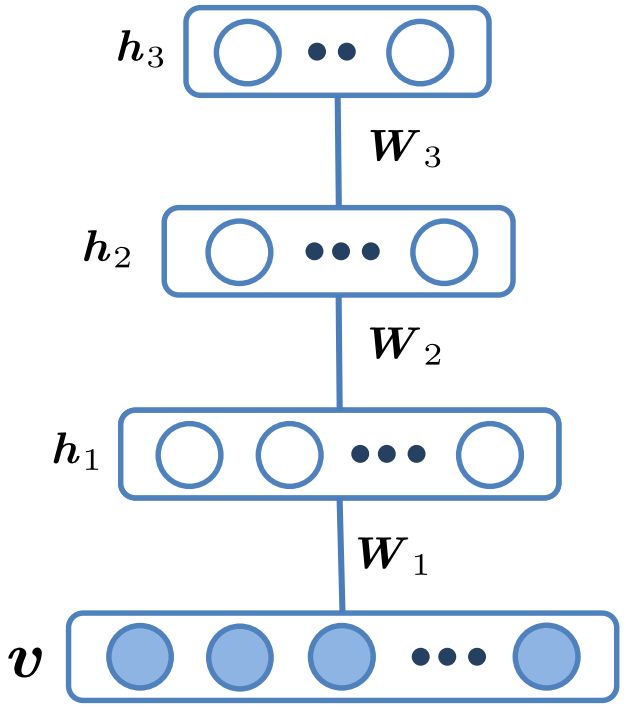


Deep Boltzmann Machine (DBM)

[Salakhudinov, Hinton 2008, 2012]

- Motivation: compose more complex representations from simpler ones.
- How? Expand/stack the latent structures

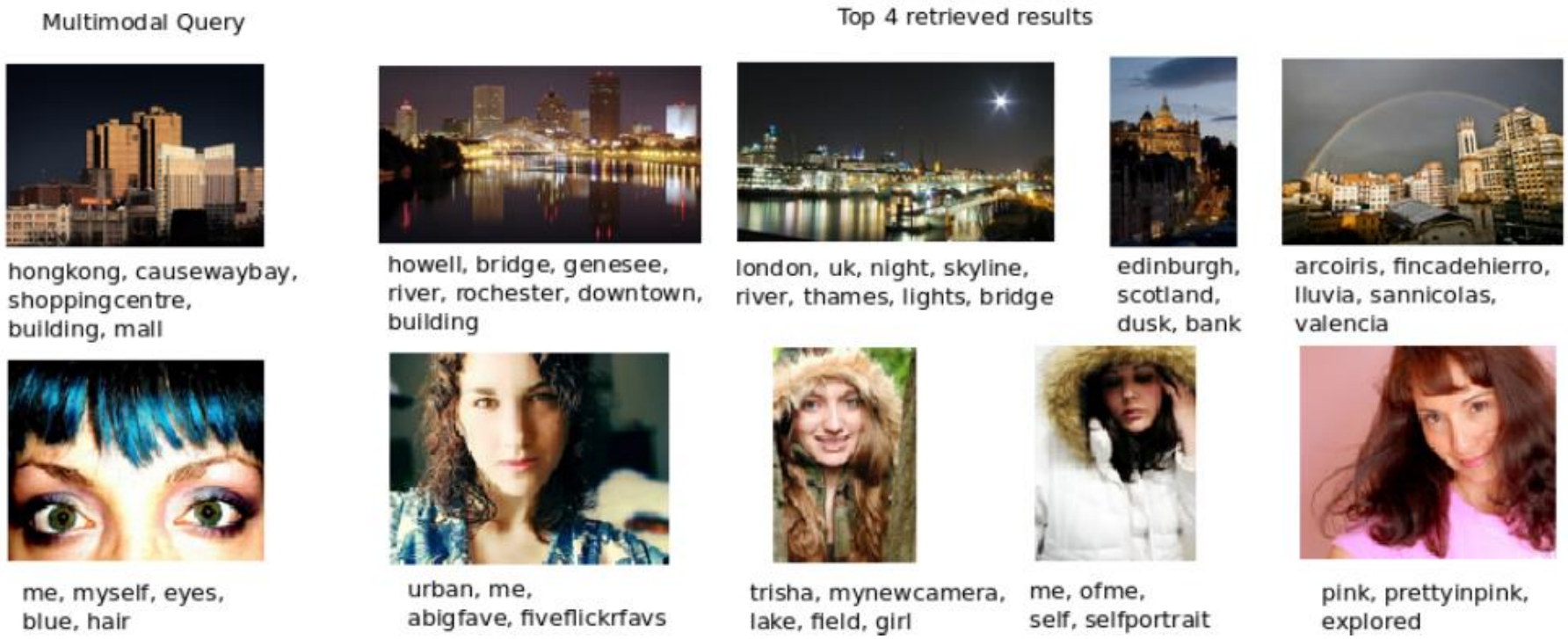




Deep
Boltzmann Machine

DBM Applications

- Document classification [Srivastava 12]
- Image classification [Srivastava and Salakhutdinov 13]
- Automatic caption generation
- Image Retrieval with Multimodal query [Srivastava and Salakhutdinov 12]



DBM Applications

- Text generated from Images
  - Huấn luyện (images, captions)
  - Input: images → suy diễn captions

dog, cat, pet, kitten, puppy, ginger, tongue, kitty, dogs, furry

sea, france, boat, mer, beach, river, Bretagne, plage, brittany

portrait, child, kid, ritratto, kids, children, boy, cute, boys, italy

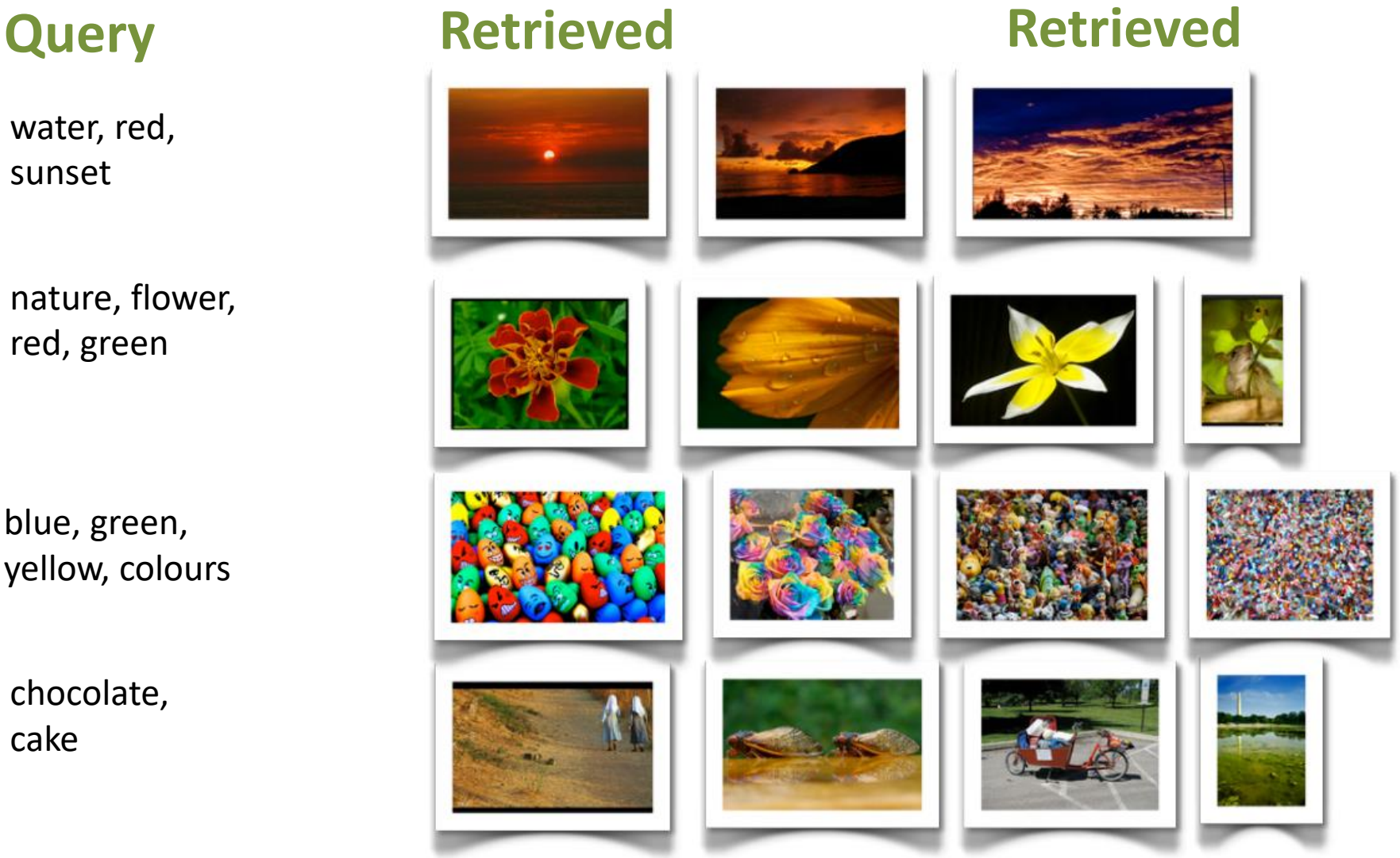
water, glass, beer, bottle, drink, wine, bubbles, splash, drops, drop

portrait, women, army, soldier, mother, postcard, soldiers

DBM Applications

[Ruslan Salakhudinov, Deep Learning Summer School, 2015]

- **Image retrieval from text**



DBM Applications

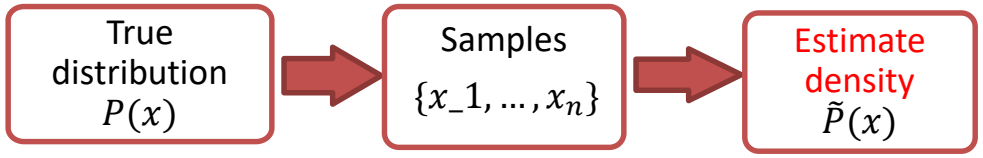
[Ruslan Salakhudinov, Deep Learning Summer School, 2015]

- Story telling

A man skiing down the snow covered mountain with a dark sky in the background

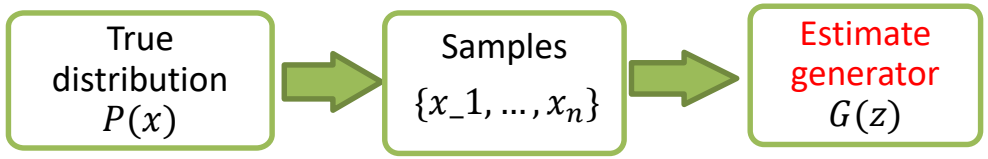
Deep Generative Models

Explicit density estimation



- Assumption: the true, but unknown, data distribution: $P(x)$
- Data: $\mathrm{D} = \{x_1, x_2, \dots, x_n\}$ where $x_i \sim P(x)$.
- Goal: estimate density $\tilde{P}(x)$ with data D such that $\tilde{P}(x)$ is as 'close' to $P(x)$ as possible.

Implicit density estimation



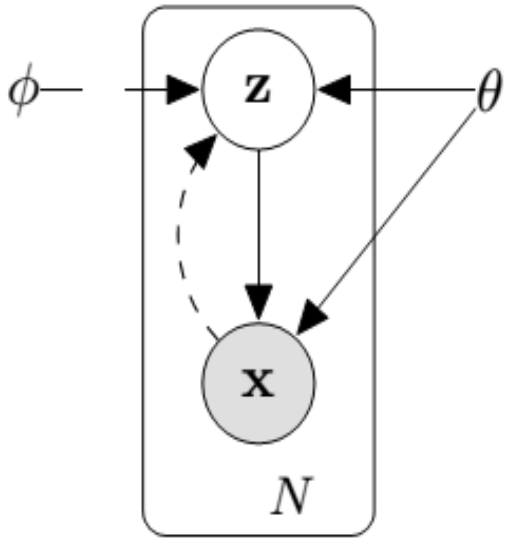
- Assumption: the true, but unknown, data distribution: $P(x)$
- Data: $\mathrm{D} = \{x_1, x_2, \dots, x_n\}$ where $x_i \sim P(x)$.
- Goal: estimate generator $G(z)$ with data D such that samples generated by $G(z)$ is 'indistinguishable' from samples generated by $P(x)$ where $z \sim \mathbb{N}(0,1)$.

Variational Autoencoders (VAE)

- Given a dataset $\boldsymbol{X} = \{\boldsymbol{x}^{(i)}\}_{i=1}^{N}$ wherein $\boldsymbol{x}$ are i.i.d and can be continuous or discrete.
- Latent ***continuous*** random variable $\boldsymbol{z}$ with prior distribution $p_{\boldsymbol{\theta}}(\boldsymbol{z})$, a conditional distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})$ parameterized by $\boldsymbol{\theta}$.
- The generative process:

$$\boldsymbol{z} \sim p_{\boldsymbol{\theta}}(\boldsymbol{z})$$
$$\boldsymbol{x} \sim p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})$$

- Inference latent variable $p(\boldsymbol{z}|\boldsymbol{x})$ for new data sample $\boldsymbol{x}$ is also expensive



SOLUTION:

- Use a recognition model $q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})$ to approximate $p_{\boldsymbol{\theta}}(\boldsymbol{z} \mid \boldsymbol{x})$,
- Learn parameters $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ together,
- Use neural networks to parameterize $q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})$ and $p_{\boldsymbol{\theta}}(\boldsymbol{z} \mid \boldsymbol{x})$ that
  form the *encoder* and *decoder*, thus resemble an *autoencoder*.

Variational Autoencoders (VAE) (tự đọc)

- Marginal likelihood: $p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z}; \boldsymbol{\theta}) p_{\boldsymbol{\theta}}(\boldsymbol{z}) \mathrm{d}\boldsymbol{z}$
  - Intractable to evaluate or differentiate.
- Its bound:

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) = D_{KL}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z} \mid \boldsymbol{x})\right) + \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x})$$

where $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x})$ is the (variational) lower bound:

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \geq \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[-\log q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}) + \log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})\right]$$

- The bound can also be written as:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}) = -D_{KL}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z})\right) + \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})]$$

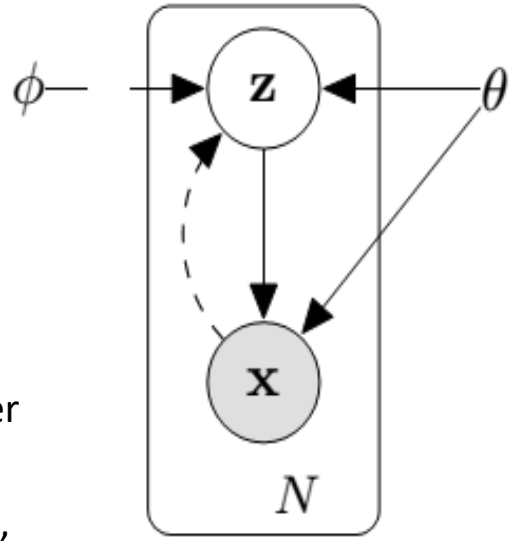
- We want to differentiate and optimize the lower bound w.r.t both the variational parameter $\boldsymbol{\phi}$ and generative parameter $\boldsymbol{\theta}$.
- Gradient $\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x})$ can be derived easily, but $\nabla_{\boldsymbol{\phi}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x})$ is problematic, for example, the approximation:

$$\nabla_{\boldsymbol{\phi}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})] = \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z}) \nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}(\boldsymbol{z})\right]$$

$$\cong \frac{1}{L} \sum_{l=1}^{L} \log p_{\boldsymbol{\theta}}\left(\boldsymbol{x} \mid \boldsymbol{z}^{(l)}\right) \nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}\left(\boldsymbol{z}^{(l)}\right)$$

exhibits very high variance.

- moreover, taking the gradient over the random variable $\boldsymbol{z}$ is infeasible, thus we use ***reparameterization*** trick.



Variational Autoencoders (VAE) (tự đọc)

- Reparametrize the random variable $\boldsymbol{z} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})$ using a differentiable transformation $g_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}, \boldsymbol{x})$ of an (auxiliary) noise variable $\boldsymbol{\epsilon}$:

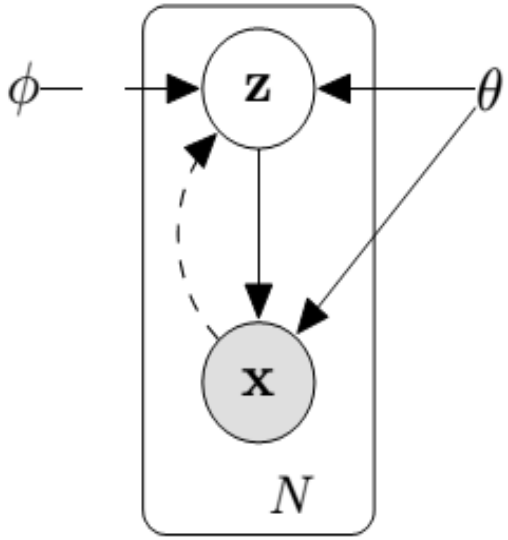
$$\tilde{\boldsymbol{z}} = g_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}, \boldsymbol{x}) \text{ with } \boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon}).$$

- The gradient of approximation of the expectation becomes:

$$\nabla_{\boldsymbol{\phi}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})]$$

$$\cong \frac{1}{L} \sum_{l=1}^{L} \log p_{\boldsymbol{\theta}}\left(\boldsymbol{x} \mid g_{\boldsymbol{\phi}}\left(\boldsymbol{\epsilon}^{(l)}, \boldsymbol{x}\right)\right) \nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}\left(g_{\boldsymbol{\phi}}\left(\boldsymbol{\epsilon}^{(l)}, \boldsymbol{x}\right)\right)$$

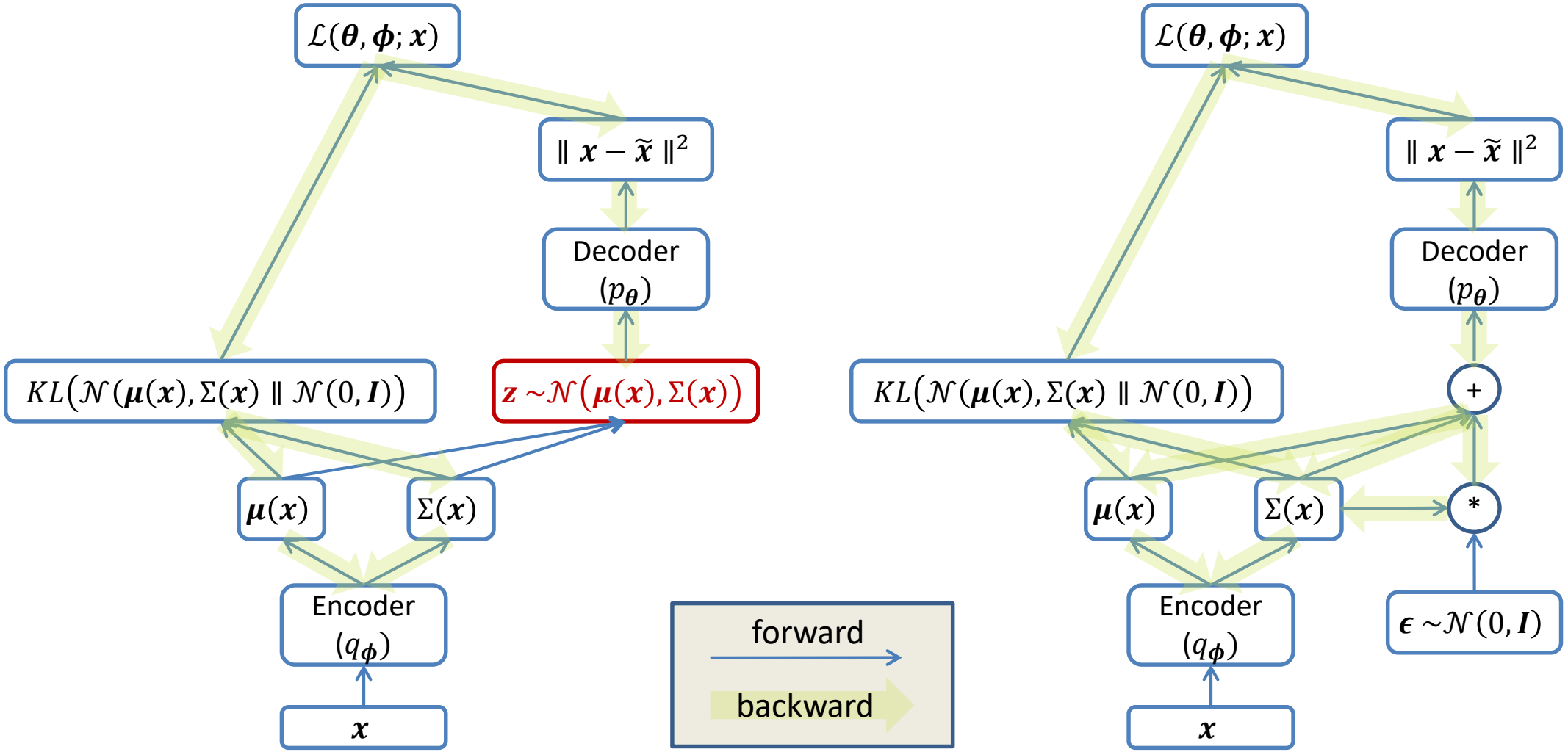
- Now we can take the derivative $\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x})$, and plug the gradient into our favourite optimization method such as SGD, AdaGrad, RMSProp, Adam.



Variational Autoencoders (VAE)

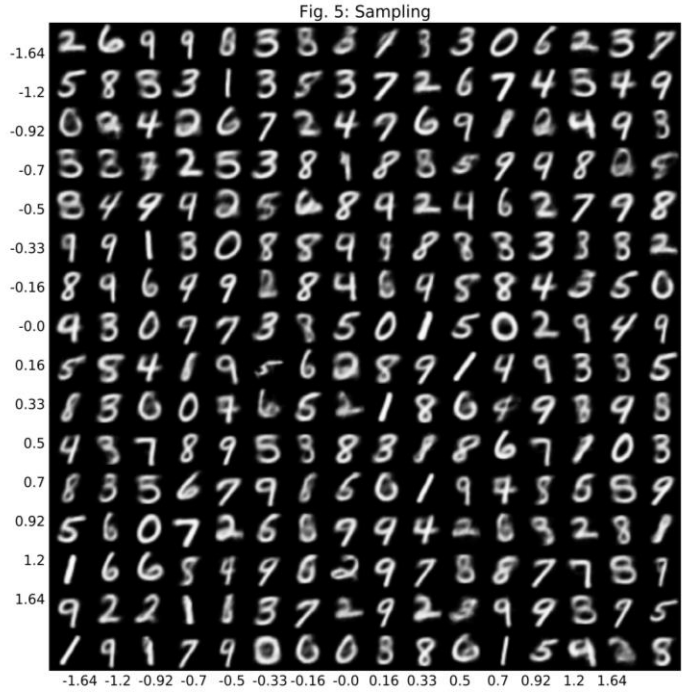
- An illustration of networks p and q with $\boldsymbol{x}$ and $\boldsymbol{z}$ following normal distributions. $\boldsymbol{z}$ (left) blocks the backpropagation.

'Reparameterization trick'



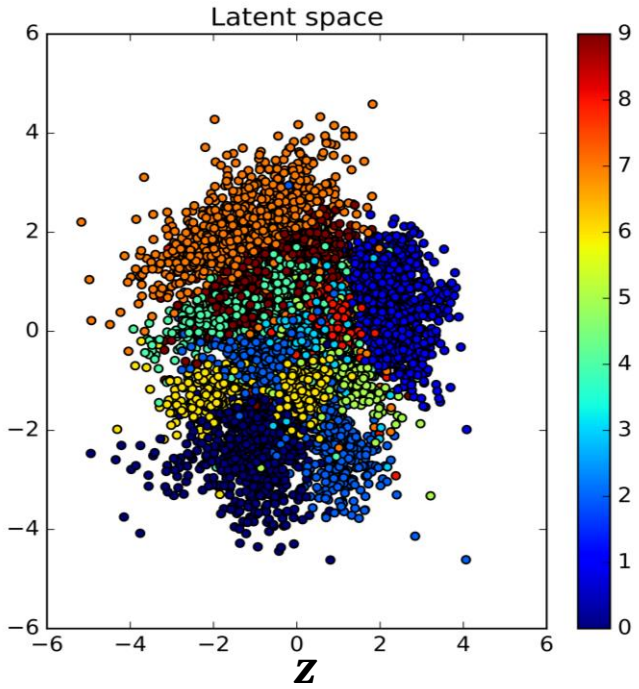
Variational Autoencoders (VAE)

- We run VAE on MNIST data with 2-dimensional latent variable $\boldsymbol{z}$.

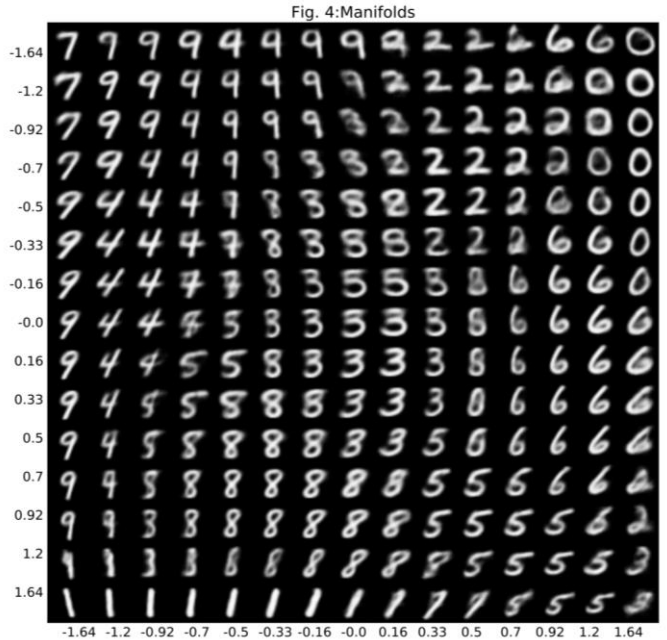


sampling

$p(x \mid z)$



$q(z \mid x)$

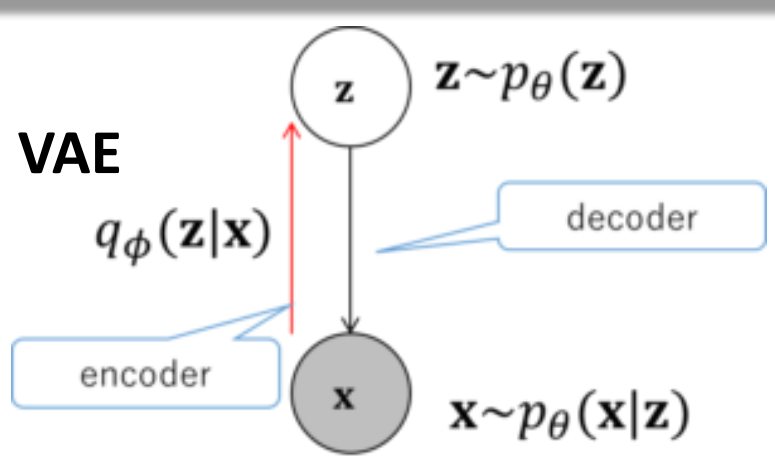


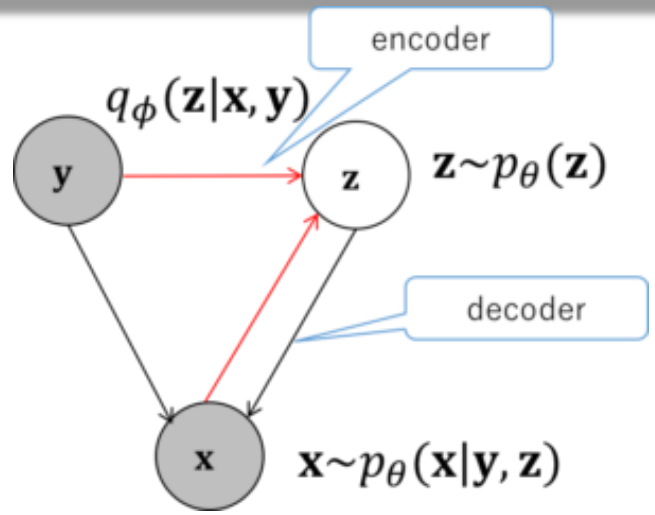
sampling on manifold

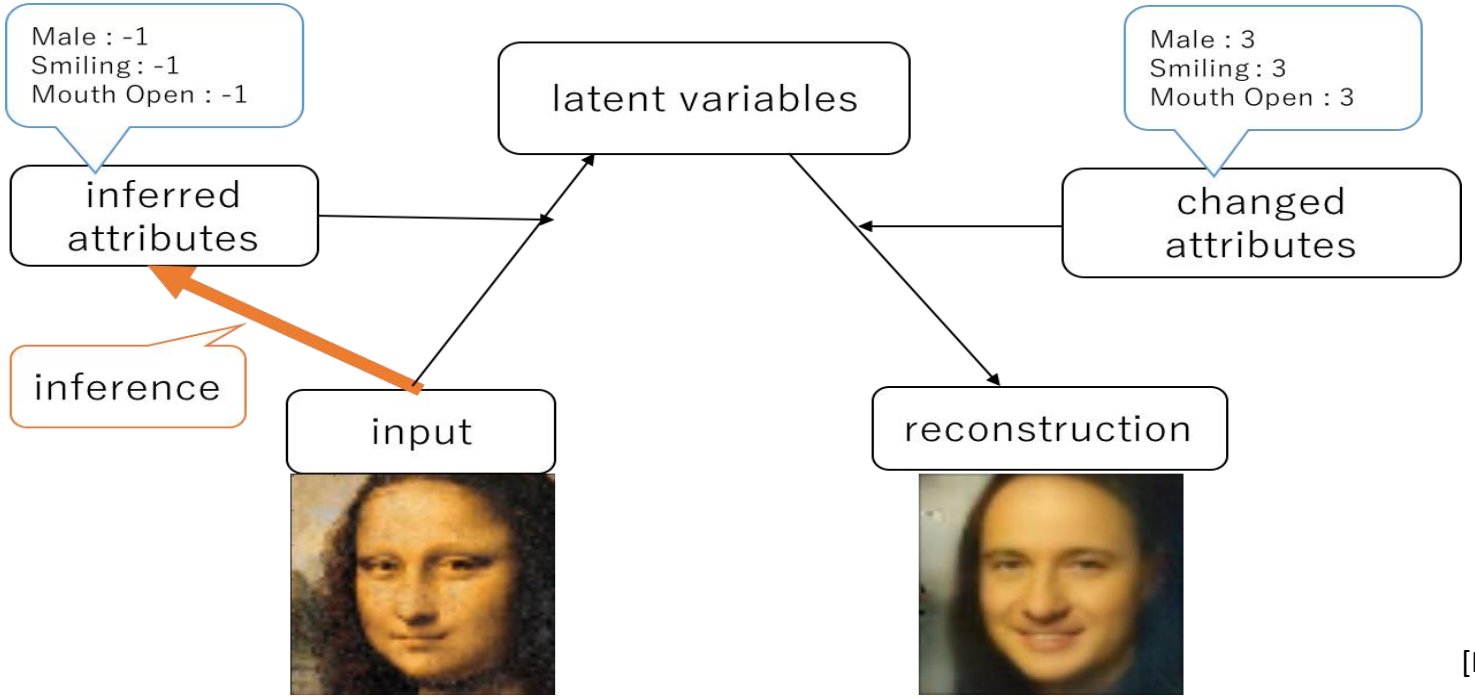
Variational Autoencoders (VAE)

- Nhược điểm của VAE: không tạo ra được hình sắc nét trong trường hợp dữ liệu rất phức tạp (ví dụ hình ảnh thiên nhiên)
- Lý do: ???

Conditional Variational Autoencoders (VAE)







[Nguồn: http://deeplearning.jp/cvae/]

Variational Autoencoders (VAE)

Những biến thể khác

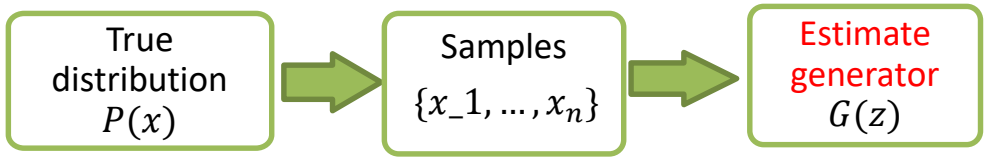
- Recurrent VAE [Fabius et al., ICLR14]
- Ladder VAE [Sønderby et al., arxiv16]
- Importance Weighted Autoencoders [Burda et al., ICLR15]
- Bottleneck VAE (semi-supervised learning) [Shui and Bui, ICML17]

Kết hợp với GANs

- Adversarial Variational Bayes [Mescheder et al., arxiv17]
- Adversarial Autoencoders [Makhzani et al., ICLR16]
- Autoencoding beyond pixels using a learned similarity metric [Larsen et al., ICML16]

GAN

Implicit density estimation



- Assumption: the true, but unknown, data distribution $P(x)$
- Data: $\mathrm{D} = \{x_1, x_2, \dots, x_n\}$ where $x_i \sim P(x)$.
- Goal: estimate generator $G(z)$ with data D such that samples generated by $G(z)$ is 'indistinguishable' from samples generated by $P(x)$ where $z \sim \mathbb{N}(0,1)$.

GAN estimate a generative model via an adversarial process by simultaneously training two models:

1. A generative model G that captures the data distribution.
2. A discriminative model D that estimates the probability that a sample came from the training data rather than G.

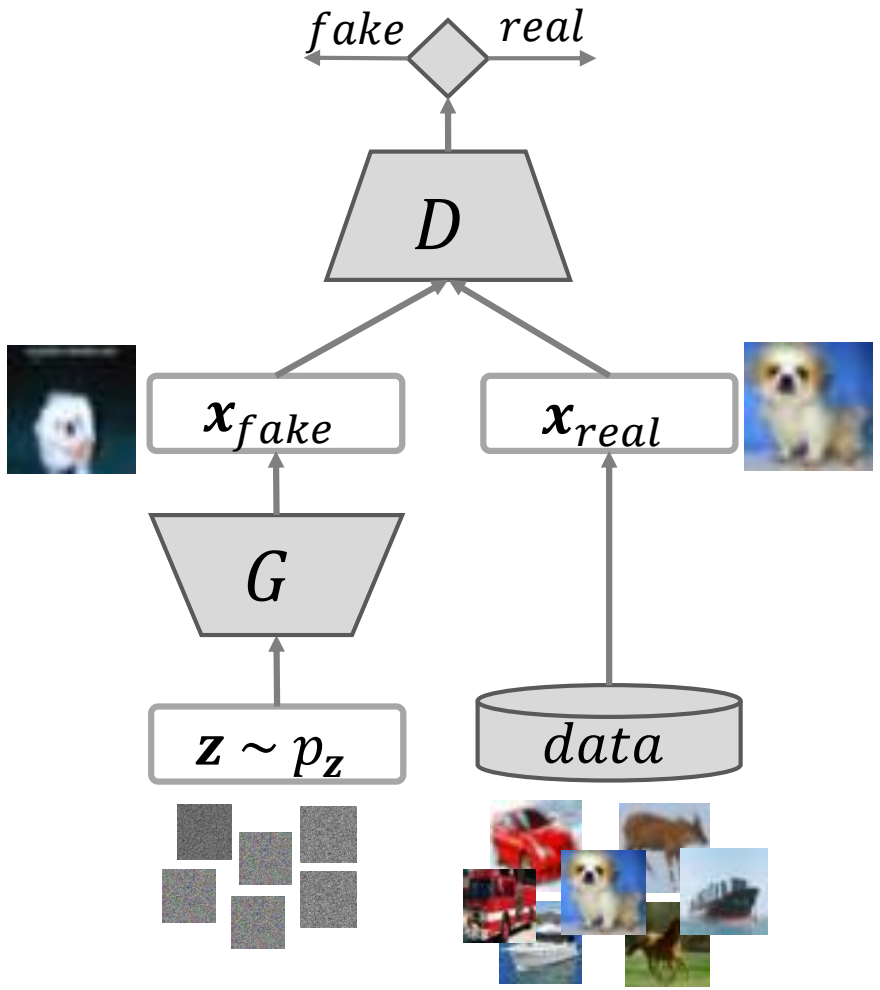
Mô hình chung của GAN

- Noise variable $\boldsymbol{z} \sim p_{\boldsymbol{z}}$ - the prior distribution
- A generator G maps from noise $\boldsymbol{z}$ to data $\tilde{\boldsymbol{x}}$ space:

$$\tilde{\boldsymbol{x}} = G(\boldsymbol{z}; \boldsymbol{\theta}_g)$$

constructed by a deep neural network with parameters $\boldsymbol{\theta}_g$.

- The discriminator $D(\boldsymbol{x}; \boldsymbol{\theta}_d)$, another deep neural network with parameters $\boldsymbol{\theta}_d$, maps a sample $\boldsymbol{x}$ to a single scalar in $[0, 1]$. This represents the probability that $\boldsymbol{x}$ comes from the real data rather than G.



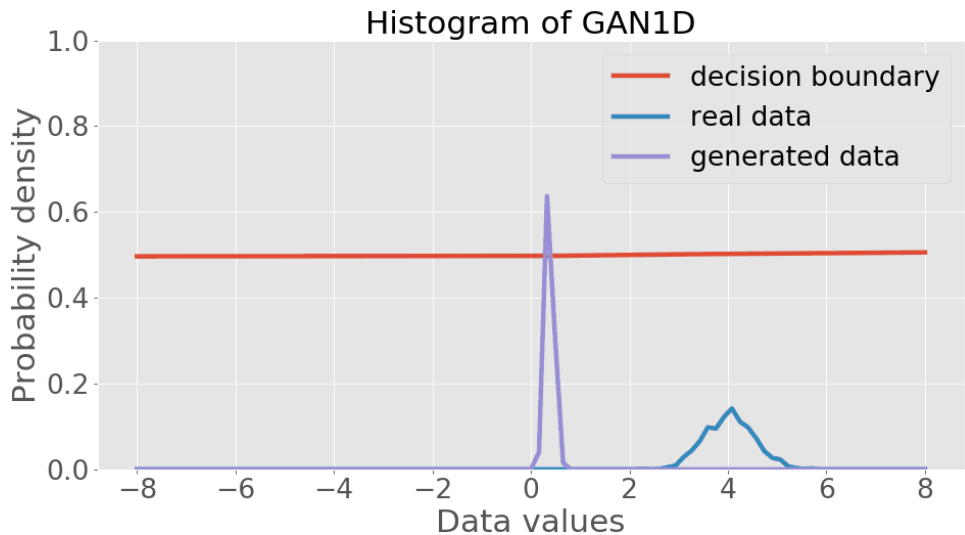
- To summarize:
  - $\boldsymbol{x} \sim p_{data} \rightarrow D(\boldsymbol{x})$ tries to be near 1,
  - $\boldsymbol{z} \sim p_{\boldsymbol{z}} \rightarrow D$ tries to push $D(G(\boldsymbol{z}))$ near 0,
  - $\boldsymbol{z} \sim p_{\boldsymbol{z}} \rightarrow G$ tries to push $D(G(\boldsymbol{z}))$ near 1.

Huấn luyện GAN

- D and G can be considered two competitors in a two-player minimax game with the value function $V(G, D)$:
$$\min_G \max_D V(G, D) = \mathbb{E}_{\boldsymbol{x} \sim p_{data}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z}))]$$

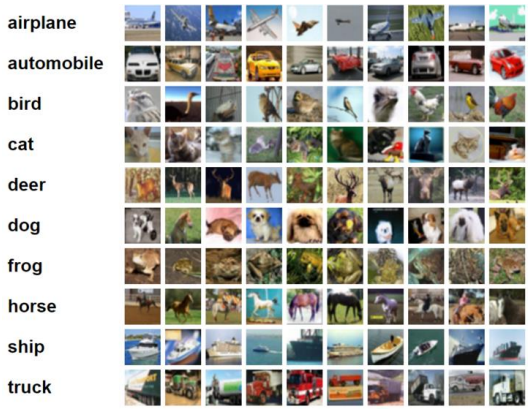
- Alternatively train D to maximize the probability of assigning the correct label to both training data and samples from G:
  - $\{\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, \dots, \boldsymbol{z}^{(M)}\} \sim p_{\boldsymbol{z}}$
  - $\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots, \boldsymbol{x}^{(M)}\} \sim p_{data}$
  - Update D:
$$\nabla_{\boldsymbol{\theta}_\mathrm{d}} \frac{1}{\mathrm{M}} \sum_{i=1}^{M} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right]$$

- and train G to minimize $1 - D\left(G(\boldsymbol{z})\right)$ to fool the discriminator:
  - $\{\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, \dots, \boldsymbol{z}^{(M)}\} \sim p_{\boldsymbol{z}}$
  - Update G:
$$\nabla_{\boldsymbol{\theta}_\mathrm{d}} \frac{1}{\mathrm{M}} \sum_{i=1}^{M} \left[\log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right]$$

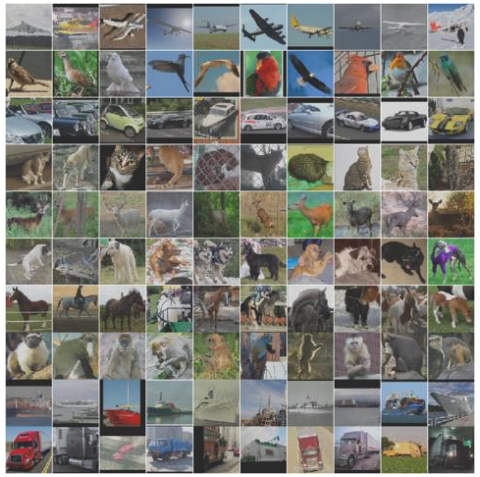


More examples

Real images



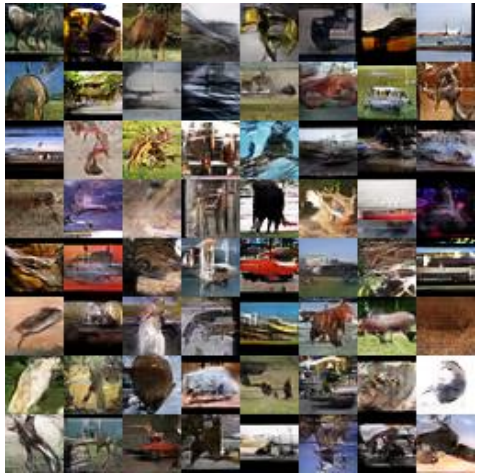
CIFAR-10



STL-10

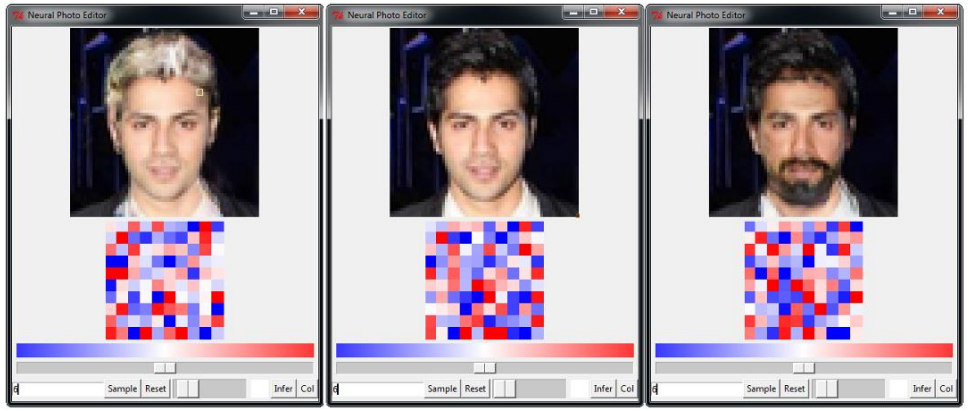
ImageNet

Generated images

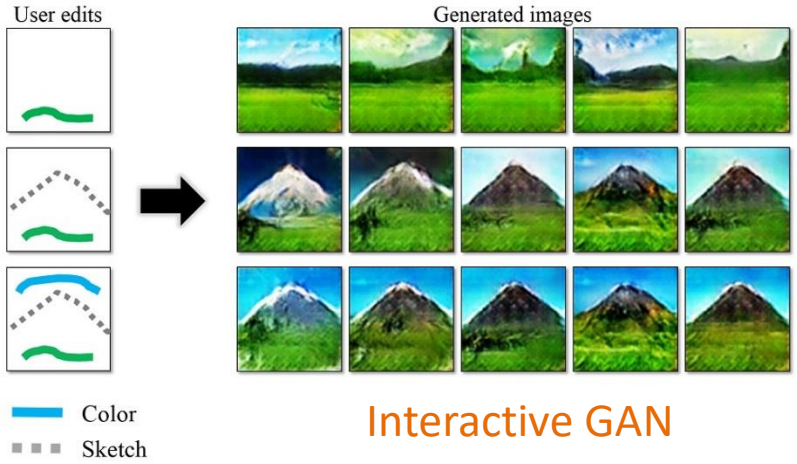


GAN Zoo

Super-resolution GAN



Introspective adversarial networks



Interactive GAN

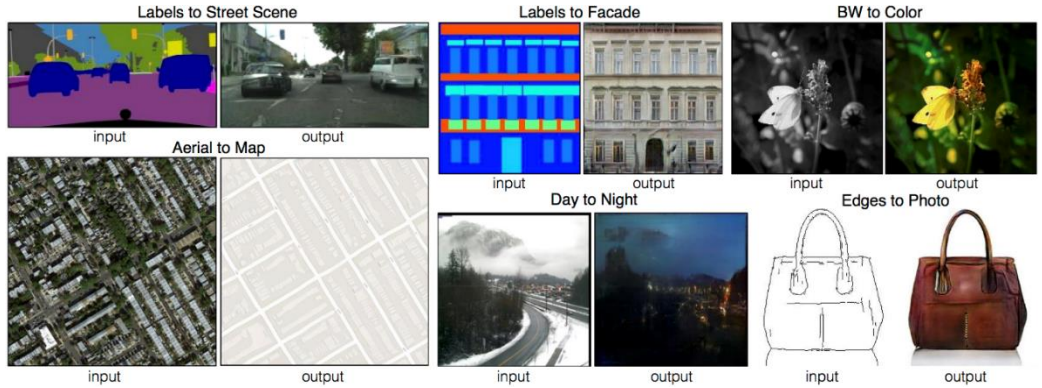


Image-to-Image

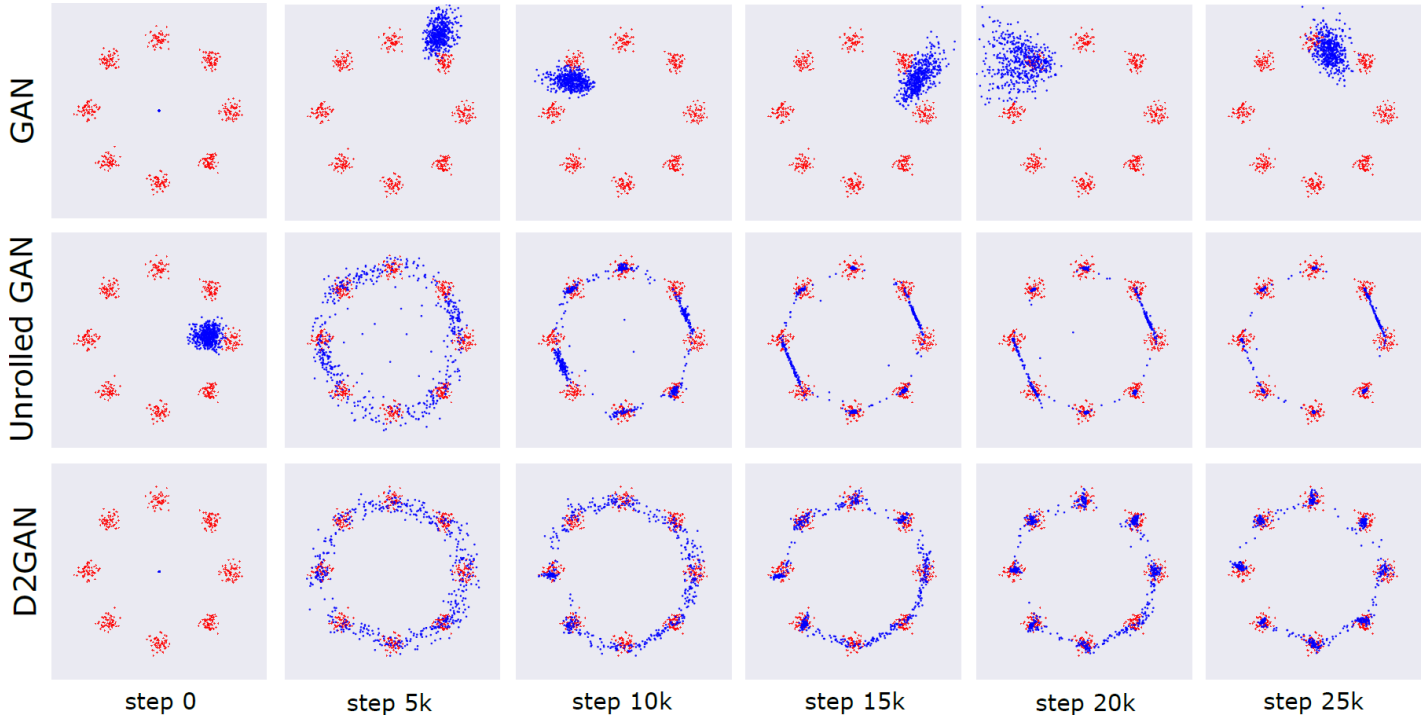
Hundreds of GAN's variants can be found here: https://github.com/nightrome/really-awesome-gan

Nhược điểm của GAN?

- Sharp images but unrecognizable when train on large-scale and diverse dataset like ImageNet.

Nhược điểm của GAN?

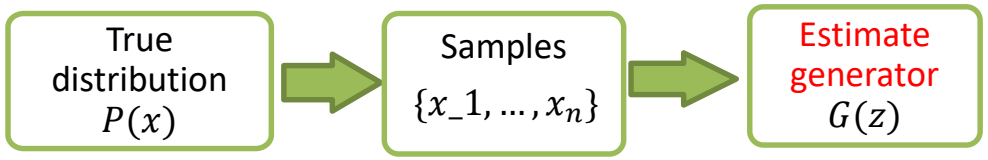
- Mode collapse



(our on-going work)

Deep Generative Models

Implicit density estimation



VAE
GAN

- Remarks
  - we can also use $\tilde{P}(x)$ as a generator to generate new data (although it is not always easy to do so, e.g., think of drawing samples from a complicated graphical model)
  - The generator $G(z)$ doesn't give an explicit form of the density, hence can't evaluate the likelihood of an unseen data point (but it's possible to get around this – new research problem!)
  - Estimate $\tilde{P}(x)$ is generally a harder problem than $G(z)$ (at least computationally in high-dimensional data); but why bother with $\tilde{P}(x)$ if what we want is to just generate new samples?

Outline

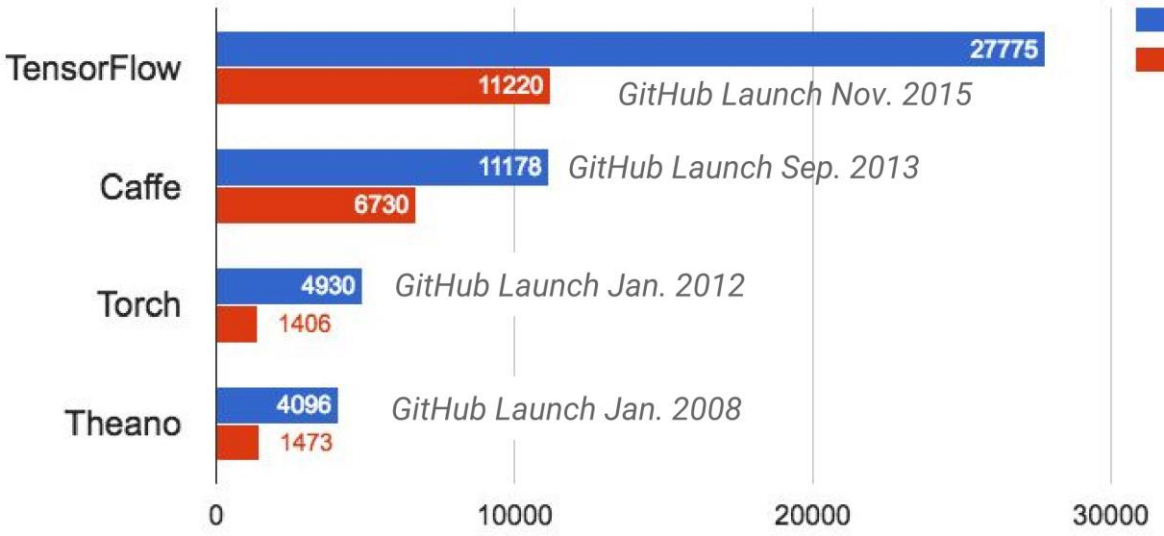
- What is deep learning and few application examples.
- Deep neural networks
  - Key intuitions and techniques
  - Deep Autoencoders
  - Convolutional networks (CNN)
- Neural embedding
  - Word2vec, recent embedding methods
- Deep Generative Models
  - RBM and DBM
  - VAE and GAN
- TensorFlow
- Few practical tips

Deep Learning Packages

- TensorFlow
- Theano
- Caffe
- Torch
- MxNet
- cuda-convnet
- Chainer
- MatConvNet
- CxxNet
- EbLearn
- deeplearning4j
- convnet.js
- cudamat
- H20
- deepdist
- mshadow
- neon
- BigDL
- SparkDL
- ….

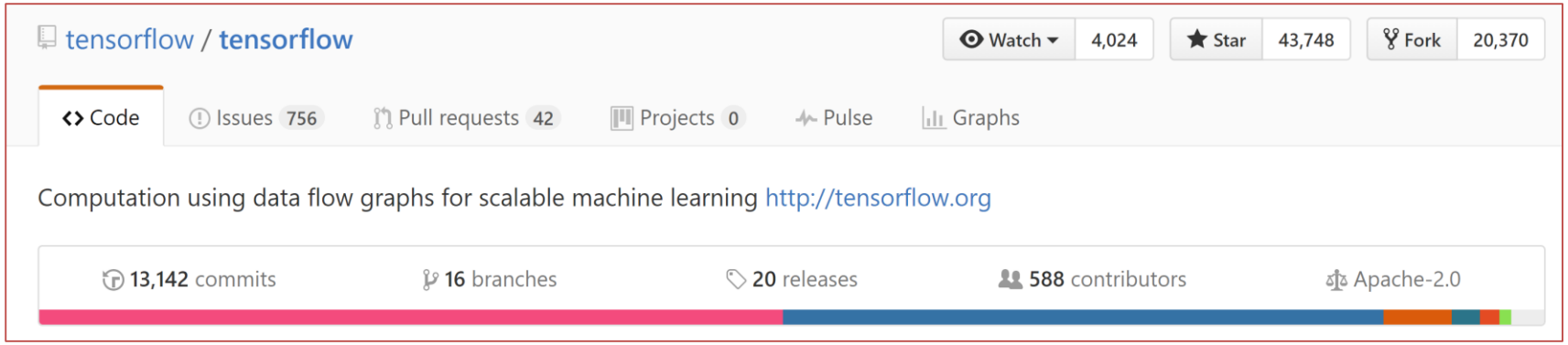
TensorFlow - Popularity

Strong External Adoption



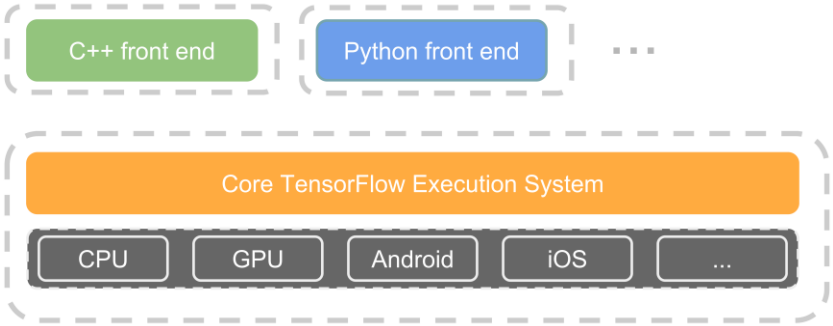
https://www.tensorflow.org/
https://github.com/tensorflow/tensorflow

[Nguồn: Jeff Dean}



TensorFlow

- Back-end in C++: very low overhead
- Front-end in Python or C++: friendly programming language

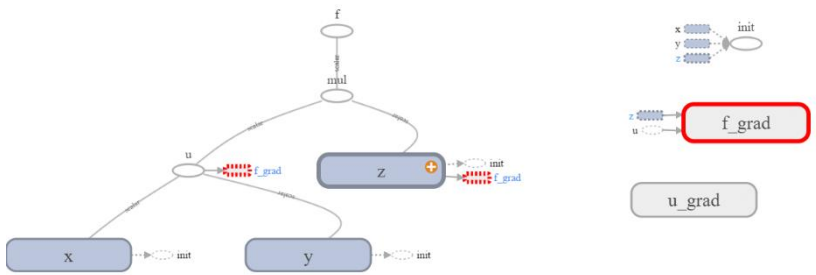


- … and even in more platforms

Linux CPU	Linux GPU	Mac OS CPU	Windows CPU	Android
build passing	build passing	build passing	build passing	build passing

- Computational graph as dataflow graph
  - Construction phase: assemble a graph to represent the design, and to train the model
  - Execution phase: repeatedly execute a set of training ops in the graph

- Flexible, intuitive construction
- Automatic differentiation
- Switchable between CPUs and GPUs
- Multiple GPUs in one machine or distributed over multiple machines
- Automatically add ops to calculate symbolic gradients of variables w.r.t loss function
- Apply these gradients with an optimization algorithm
- Launch the graph and run the training ops in a loop



A very simple graph

Automatic Differentiation

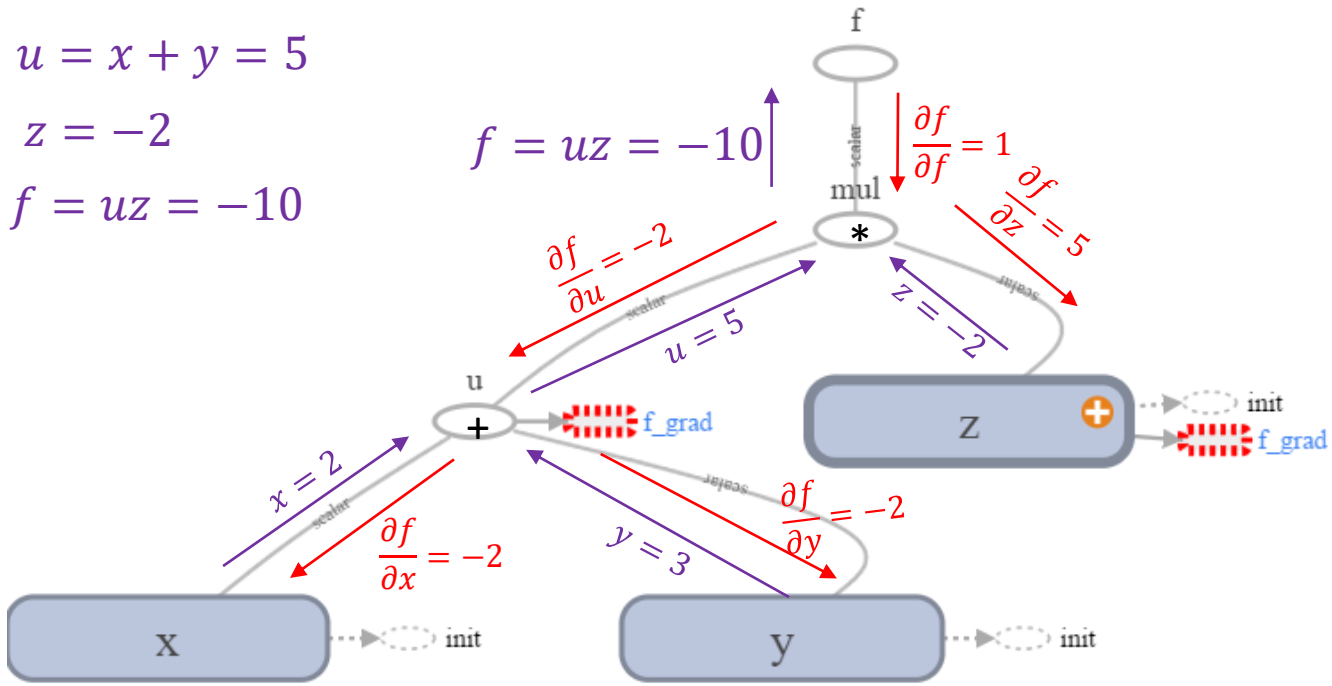
$x = 2 \quad y = 3$

$u = x + y = 5$

$z = -2$

$f = uz = -10$

$$f(x, y, z) = (x + y)z$$



$$\frac{\partial f}{\partial f} = 1$$

$$\frac{\partial f}{\partial u} = \frac{\partial f}{\partial f}\frac{\partial f}{\partial u} = 1 * z = -2$$

$$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial f}\frac{\partial f}{\partial z} = 1 * u = 5$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial u}\frac{\partial u}{\partial x} = -2 * 1 = -2$$

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial u}\frac{\partial u}{\partial y} = -2 * 1 = -2$$

Outline

- What is deep learning and few application examples.
- Deep neural networks
  - Key intuitions and techniques
  - Deep Autoencoders
  - Convolutional networks (CNN)
- Neural embedding
  - Word2vec, recent embedding methods
- Deep Generative Models
  - RBM and DBM
  - VAE and GAN
- TensorFlow
- Few practical tips

Vài kinh nghiệm …

Summary for practical use – **Design**

- Design and train
  - activation function: ReLU
  - data preprocessing: whitening
  - weight initialization: Xavier
  - batch normalization: should use
  - Babysitting the learning progress
  - hyperparameters optimization: random sample hyperparams.
- Pay attention to sensitivity of learning rate η
- Minibatch size
  - Larger batches are more accurate, but less scalable.
  - Too small batches under-utilize multicore architectures.
  - Larger batches cost more memory
  - Power of 2, e.g., 16, 32, 64, 128, 256, …

- During the learning process
  - Start with one hidden layer and small # units
  - Start with small learning rate and regularization
  - Monitor the loss
  - Early stopping
  - Decay learning rate
  - Hyperparam Opt: coarse -> fine
    - first stage: few epochs to get rough idea of what params work
    - second stage: longer running time, finer search
  - Ratio of weights updates
    - Update_scale / param_scale = 0.001 (1e-3)
- Ensembles
  - different initializations, average results of multiple independent models, average multiple model checkpoints of a single model

Vài kinh nghiệm …

Summary for practical use - **Debugging**

- Visualize the model and what it learns
- Visualize and analyze the worst mistakes
- Criticize the software using training and testing errors
  - Training error is often low, and testing error high
  - If both are high, the model is underfitting or software defect
- Fit a tiny data, e.g., only few sample
- Compare back-propagated gradient to numerical gradient.
  - E.g, use centered difference for better accuracy

$$f'(x) \approx \frac{f\left(x + \frac{1}{2}\epsilon\right) - f\left(x - \frac{1}{2}\epsilon\right)}{\epsilon}$$

- Monitoring histograms of activations and gradient.
  - Statistics of activations and gradients collected over a large amount of training iterations (perhaps one epoch).
  - The pre-activation value of hidden units can indicate if the units saturate, or how often they do.
    - ReLU: How often are they off?
    - tanh: average of absolute value of the pre-activation.
  - How quickly gradients grow or vanish.
  - Observe the magnitude of parameter gradients to the magnitude of parameters themselves. Parameter updates over a minibatch = 1% parameter values (Bottou, 2015

Vài kinh nghiệm …

Summary for practical use - **Debugging**

- Make sure our model is able to (over) fit on a small dataset.
- If not, investigate the following situations:
  - Are some of the units saturated, even before the first update?
    - Scale down the initialization of your parameters for these units
    - Properly normalize the inputs
  - Is the training error fluctuating significantly / bouncing up and down?
    - Decrease the learning rate
- Parameter initialization
  - The initialization of parameters affects the convergence and numerical stability.
  - biases: initialize all biases to 0
  - weights:
    - cannot initialize weights to 0 with tanh activation, since the gradients would be 0 (saddle points)
    - The initial parameters need to break symmetry between different units, otherwise all hidden units would act the same.
- Useful references:
  - *Practical Recommendations for Gradient-Based Training of Deep Architectures* – Yoshua Bengio, 2012
  - *Stochastic Gradient Descent Tricks* – Leon Bottou

Tóm tắt

- What is deep learning and few application examples.
- Deep neural networks
  - Key intuitions and techniques
  - Deep Autoencoders
  - Convolutional networks (CNN)
- Neural embedding
  - Word2vec, node2vec, recent embedding methods
- Deep Generative Models
  - RBM and Deep DMs
  - Directed GM: VAE, GAN
- TensorFlow
- Few practical tips

Vài suy nghĩ thêm

- Why deep learning works?
  - Can learn and represent complex, nonlinear relationships
  - Strong and flexible priors (layered representation)
  - Train DNN is not too hard and fast
- When does it work?
  - Data, data and data
  - Good quality supervision, labels
  - Compositional
  - Hierarchical structures
- When it doesn’t work?
  - Data, data and data
  - Black box, theory ?

Xin cảm ơn các anh, chị đã lắng nghe!

Appendix

Node2vec

[Grover and Leskovec, KDD'16]

- **Step 1: Apply Word2vec (Skip-gram) to the network**
  - Let $G = (V, E)$ be a network and $N_S(u)$ be the neighbourhood of a node u, $N_S(u) \subset V$
  - Let $f: V \rightarrow \mathbb{R}^d$ be the mapping function from nodes to feature representations
  - **Objective function**:

$$\max_f \sum_{u \in V} \log \Pr(N_S(u) \mid f(u))$$

  - Conditional independence: $\Pr(N_S(u) \mid f(u)) = \prod_{n_i \in N_S(u)} \Pr(n_i \mid f(u))$
  - Symmetry in feature space: $\Pr(n_i \mid f(u)) = \frac{\exp(f(n_i) f(u))}{\sum_{v \in V} \exp(f(v) f(u))}$
  - **Objective function simplifies to**:

$$\max_f \sum_{u \in V} [-\log Z_u + \sum_{n_i \in N_S(u)} f(n_i) f(u)]$$

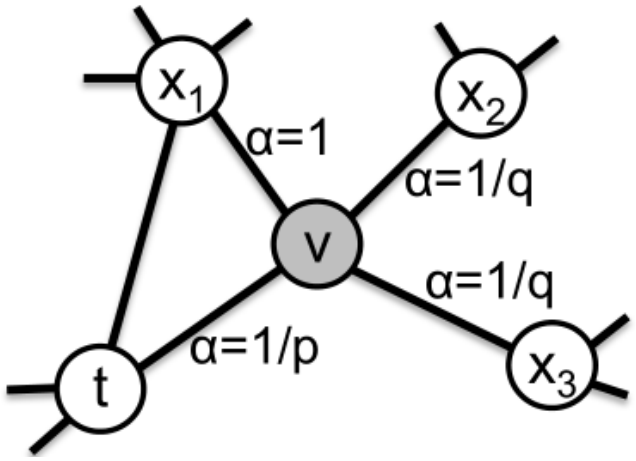
where $Z_u = \sum_{v \in V} \exp(f(u) f(v))$

Node2vec

[Grover and Leskovec, KDD'16]

- **Step 2: Develop biased random walks to capture node neighbourhood**
  - Breath-first search (BFS): search for immediate neighbours → capture structural equivalence
  - Depth-first search (DFS): search for far away neighbours → capture homophily
  - Node2vec smoothly interpolates between BFS and DFS
  - **Random walks**:

$$\Pr(c_i = x \mid c_{i-1} = v) = \begin{cases} \frac{\pi_{vx}}{Z} & \text{if } (v,x) \in E \\ 0 & \text{otherwise} \end{cases}$$



  - **Biased random walks**:

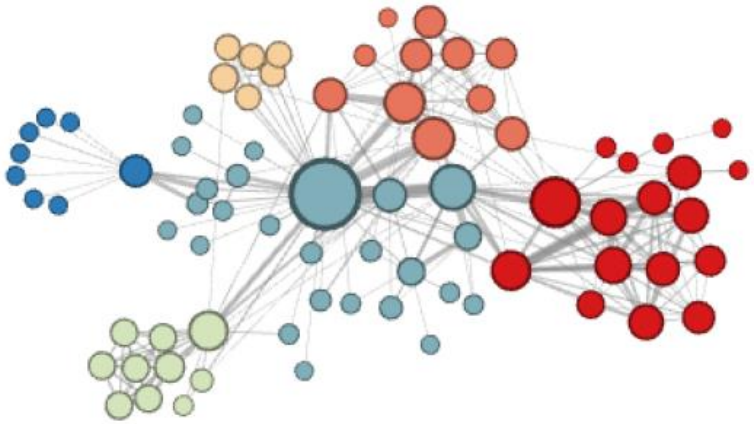
$$\Pr(c_i = x \mid c_{i-1} = v) = \begin{cases} \frac{\alpha_{qp}(t,x) w_{vx}}{Z} & \text{if } (v,x) \in E \\ 0 & \text{otherwise} \end{cases}$$

  - where w_{vx} is a weight and α_{qp} is a search bias, $\alpha_{qp}(t,x) = \begin{cases} 1/p & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ 1/q & \text{if } d_{tx} = 2 \end{cases}$

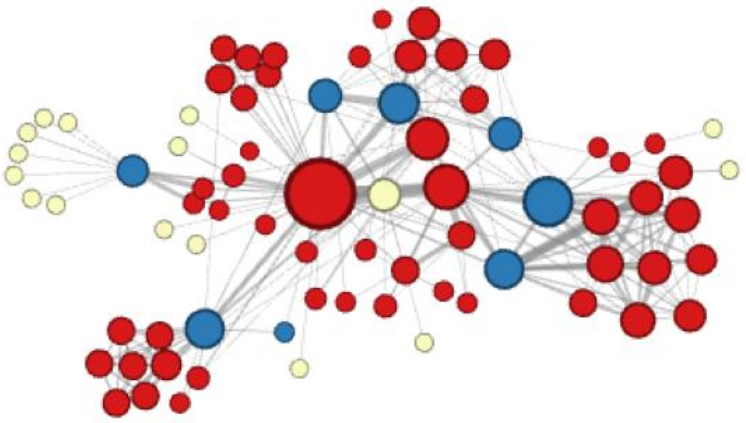
Node2vec

[Grover and Leskovec, KDD'16]

- **Demonstration**: Les Miserables network



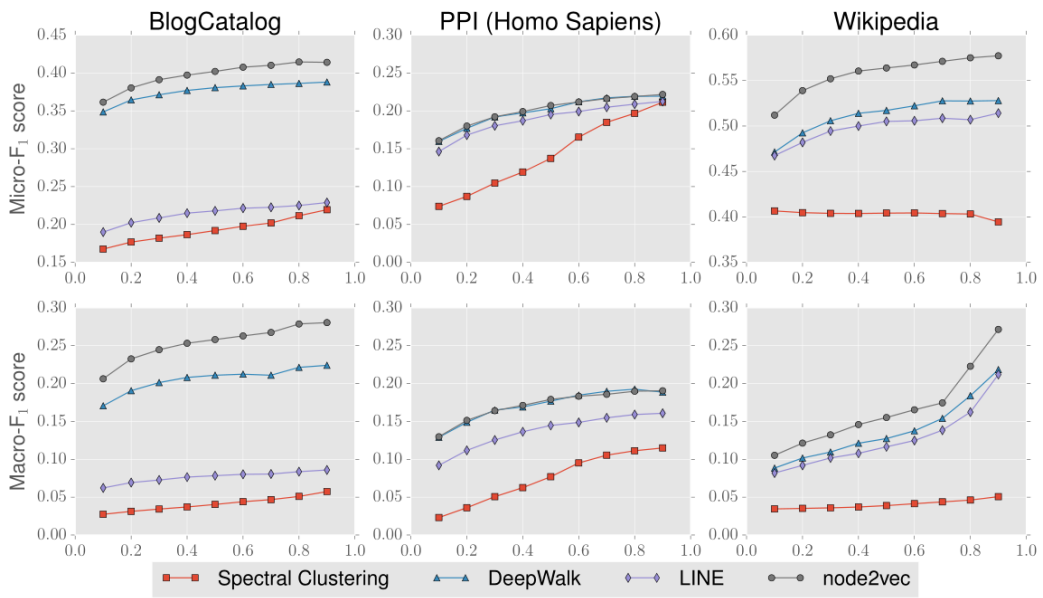
capture h*omophily*



capture *structural equivalence*

- **Experimental results**:

Algorithm	Dataset		
	BlogCatalog	PPI	Wikipedia
Spectral Clustering	0.0405	0.0681	0.0395
DeepWalk	0.2110	0.1768	0.1274
LINE	0.0784	0.1447	0.1164
node2vec	**0.2581**	**0.1791**	**0.1552**
node2vec settings (p,q)	0.25, 0.25	4, 1	4, 0.5
Gain of *node2vec* [%]	**22.3**	**1.3**	**21.8**

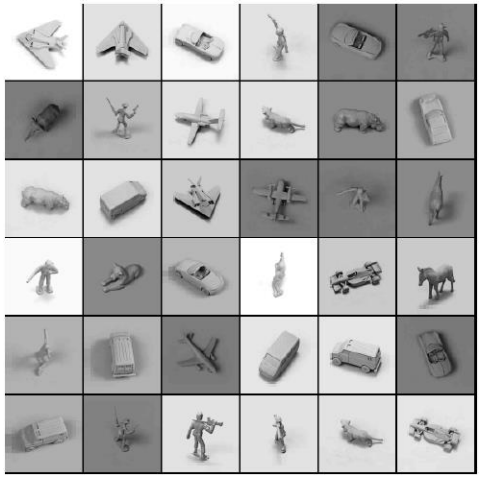


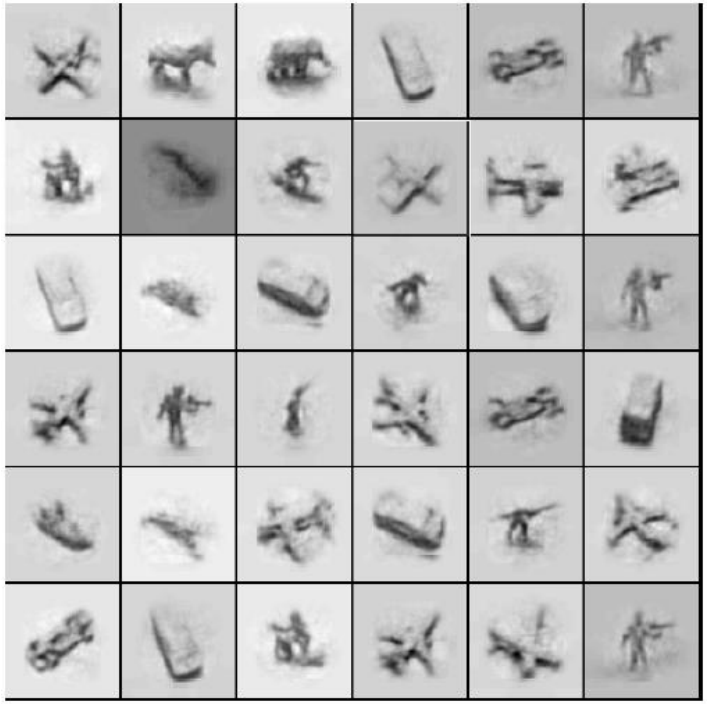
DBM Applications

[Ruslan Salakhudinov, Deep Learning Summer School, 2015]

- 3D object generation

real





generated

- Pattern completion

Damaged images

Completed images

True images