CAP5415-Computer Vision
Lecture 11-Neural Nets for Computer Vision II (Deep Learning)

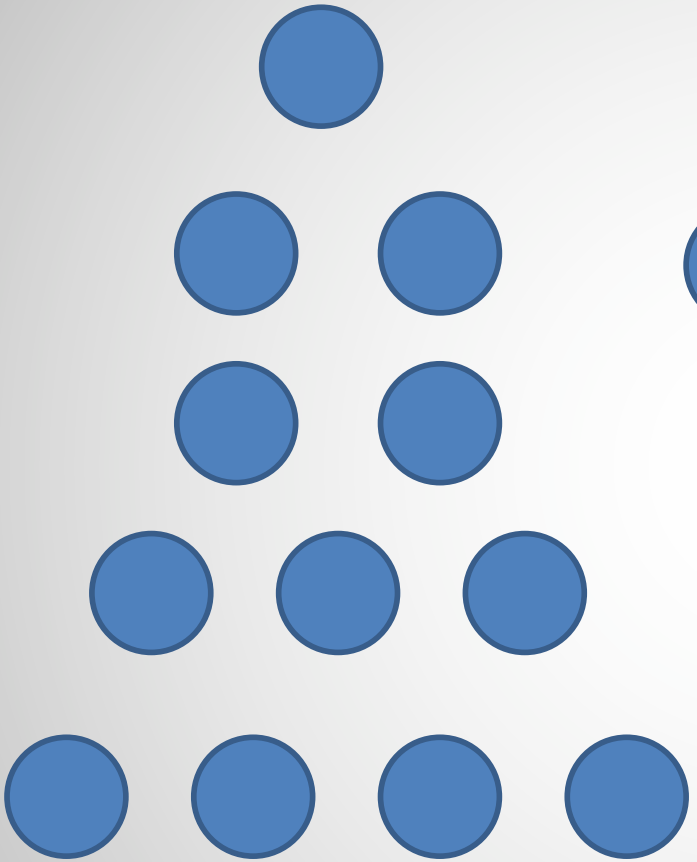
Dr. Ulas Bagci

bagci@ucf.edu

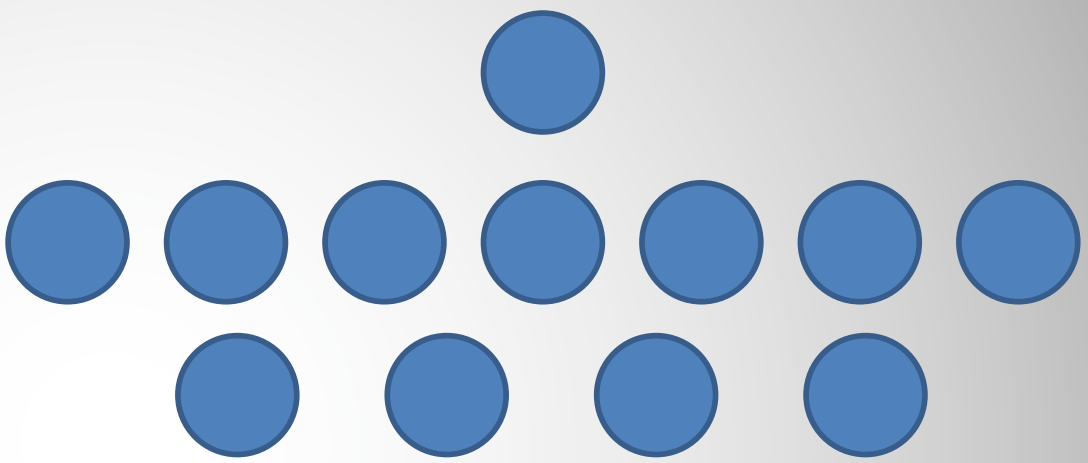
Knowledge Check

Deep or Shallow ?

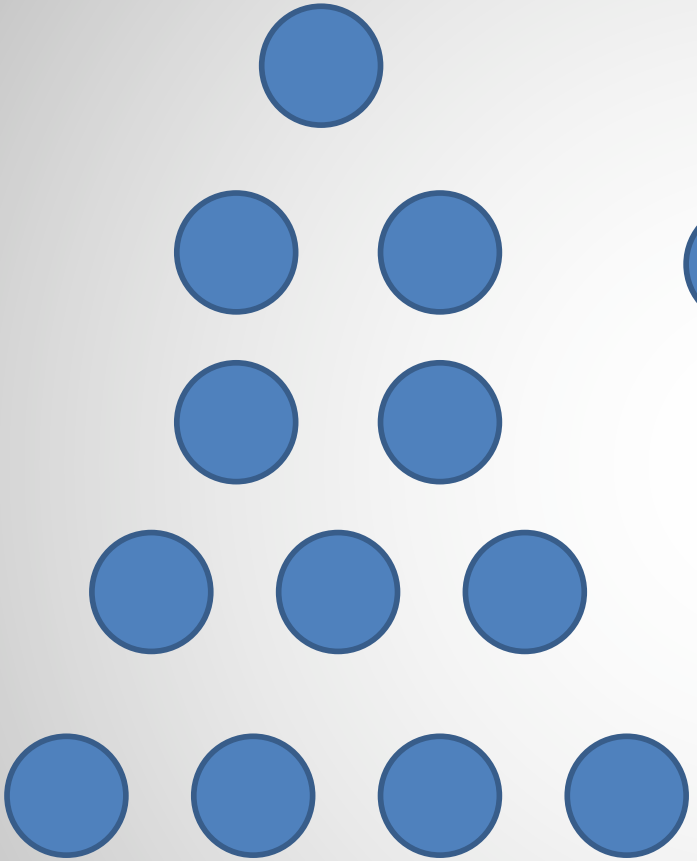
?



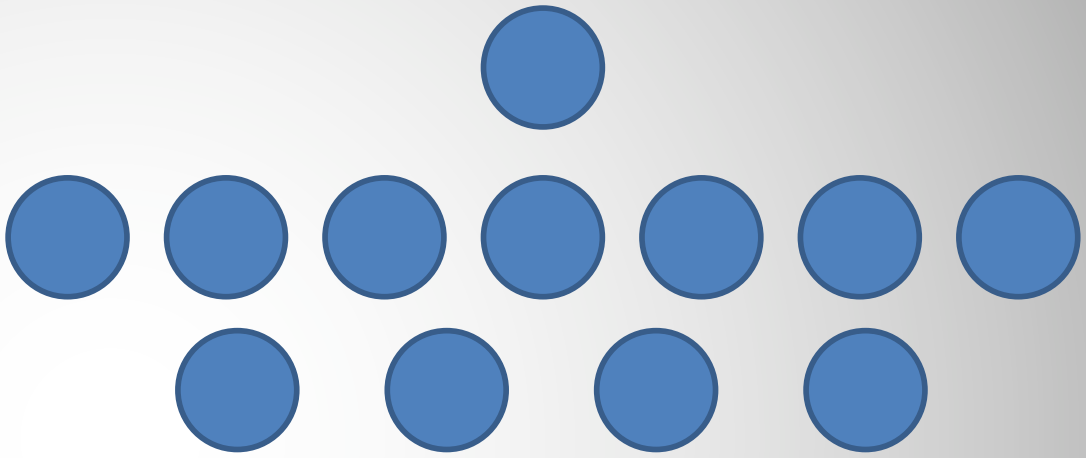
?



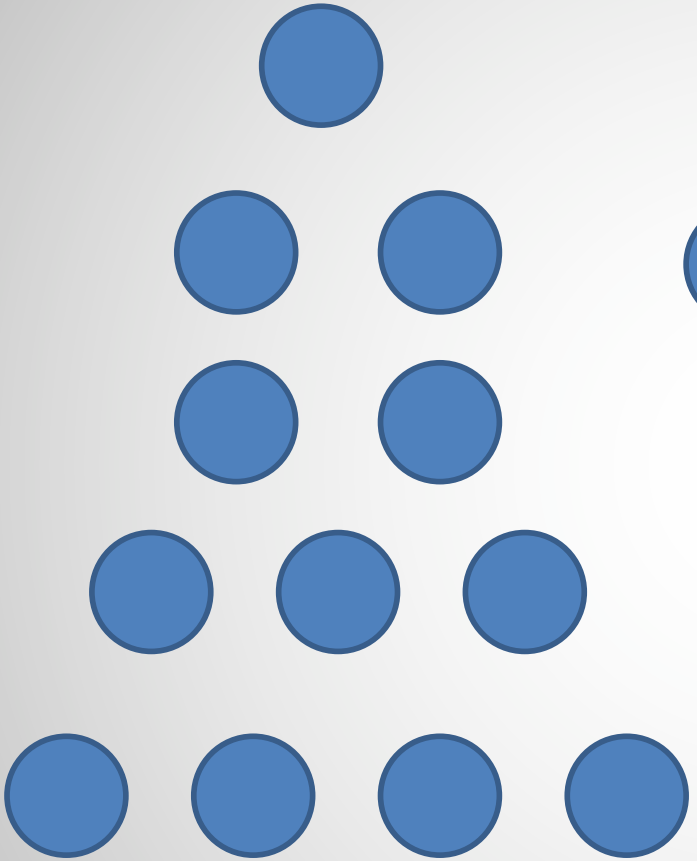
Deep Network



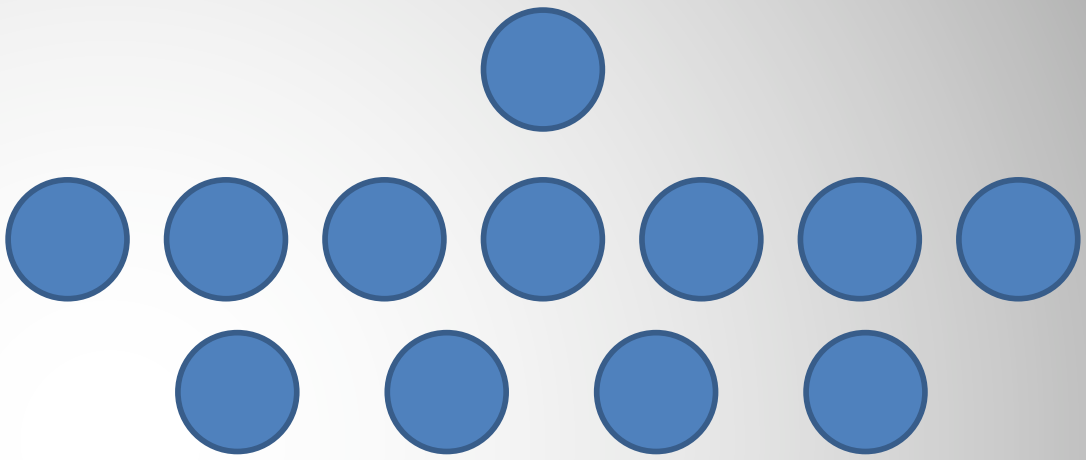
Shallow Network



Deep Network



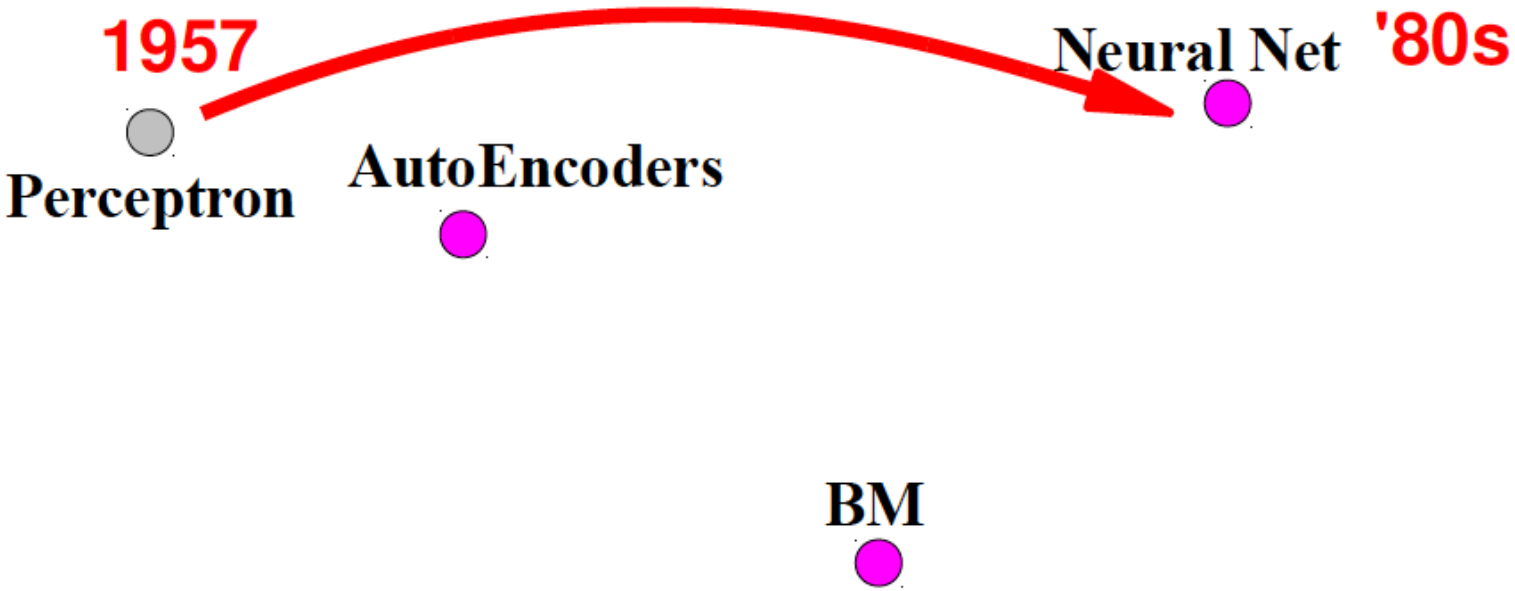
Shallow Network



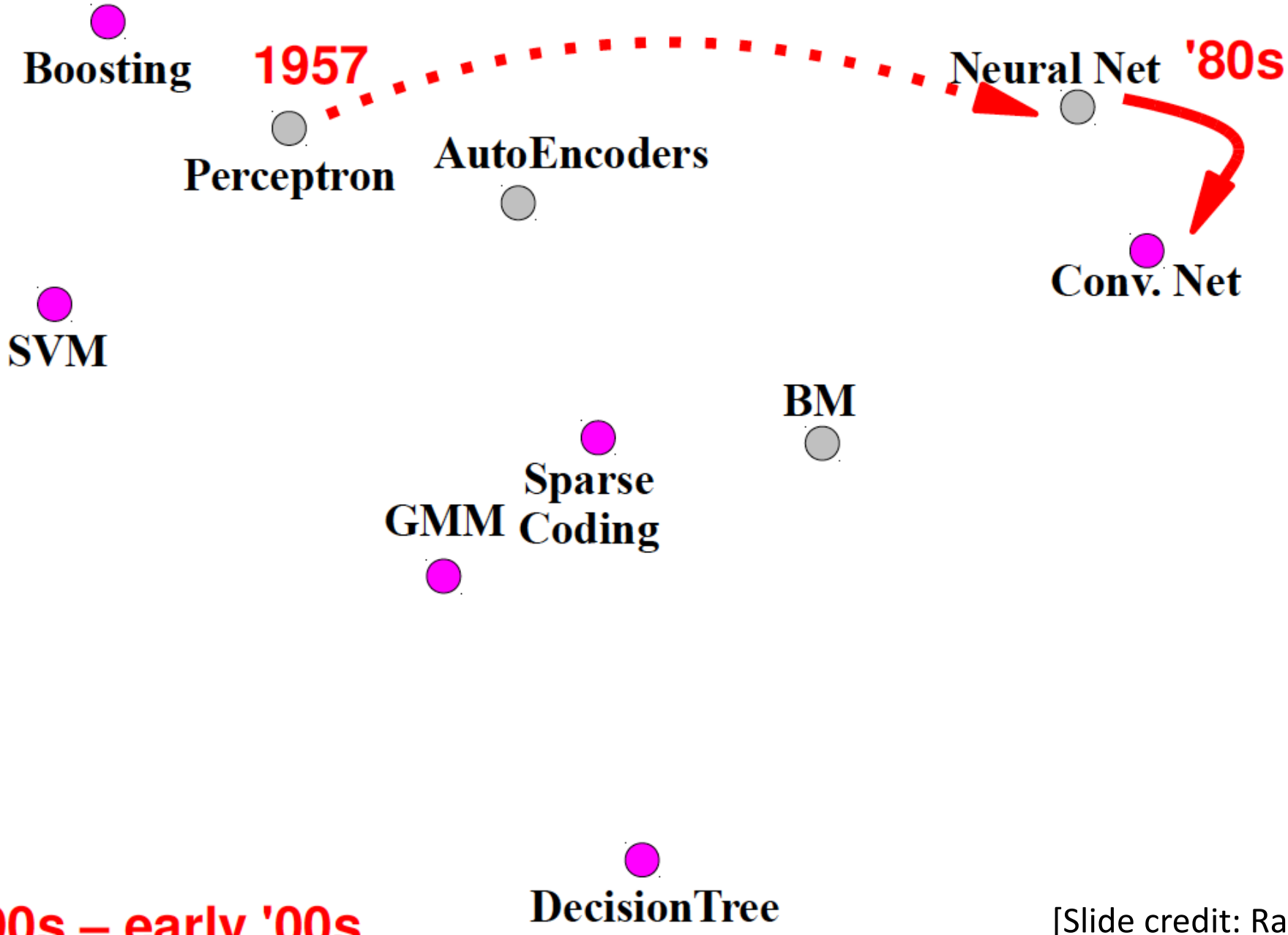
Shallow (2 layer) networks need a lot more hidden layer nodes to compensate for lack of expressivity

In a deep network, high levels can express combinations between features learned at lower levels

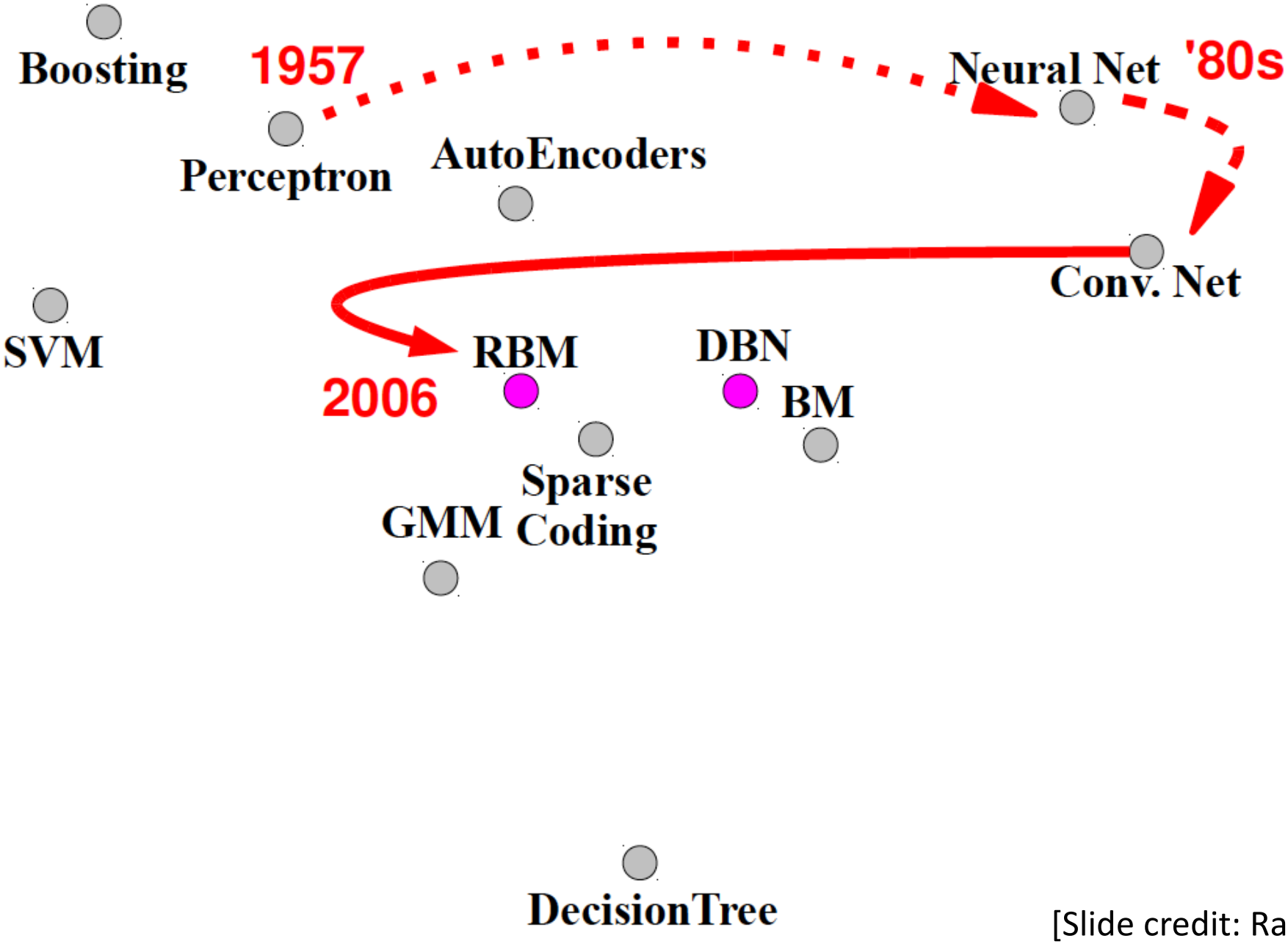
Shallow and Deep Methods



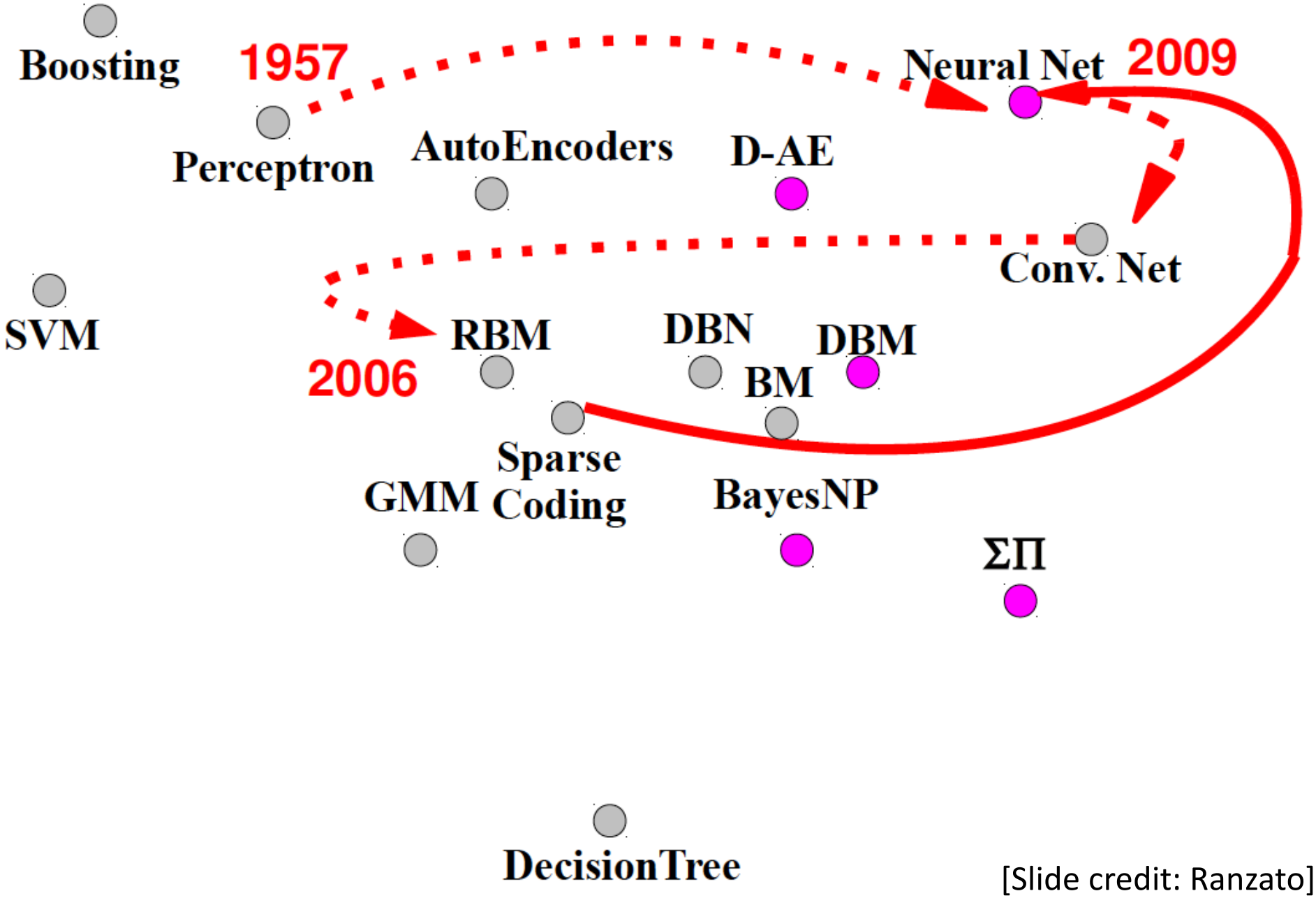
[Slide credit: Ranzato]



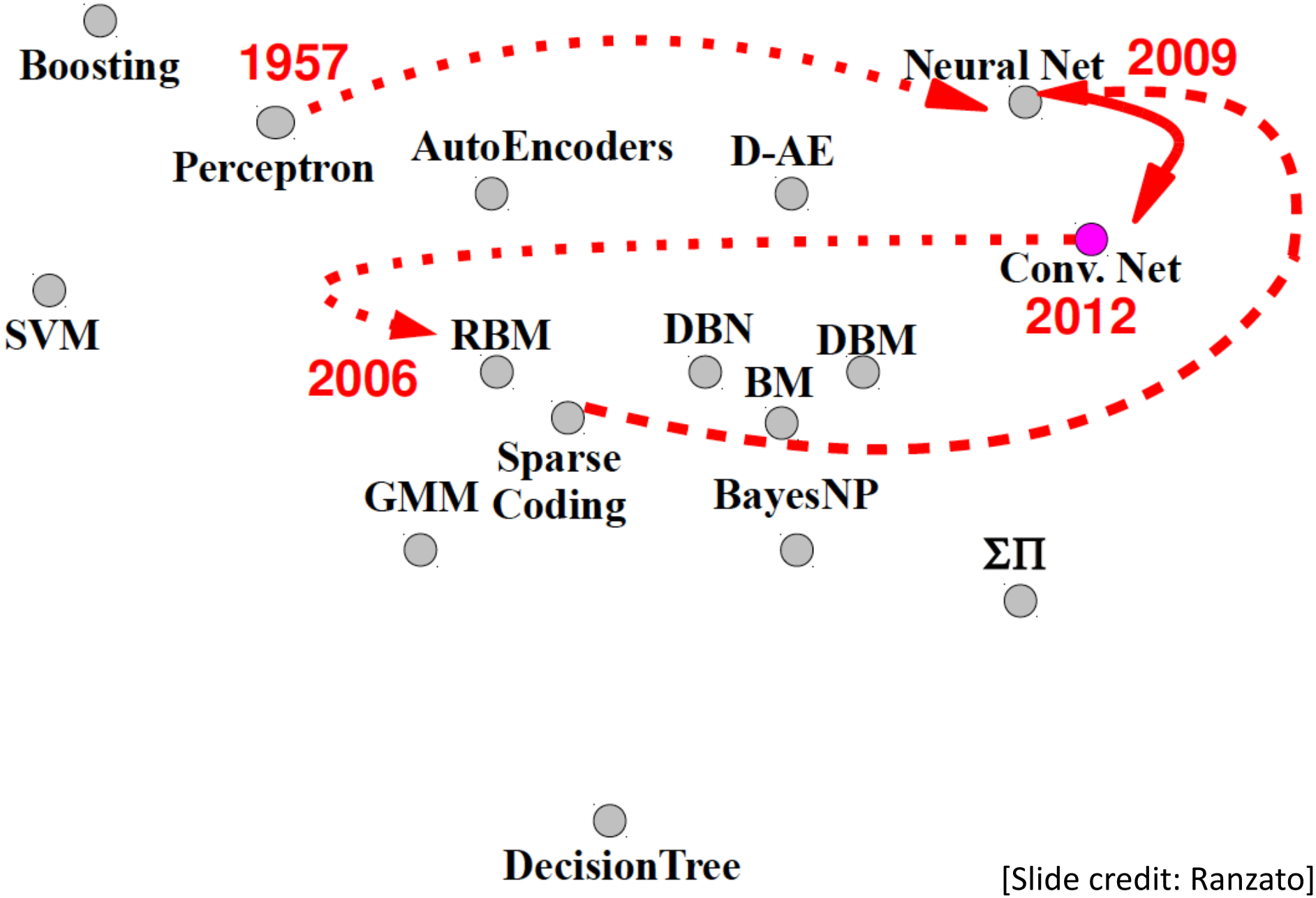
[Slide credit: Ranzato]



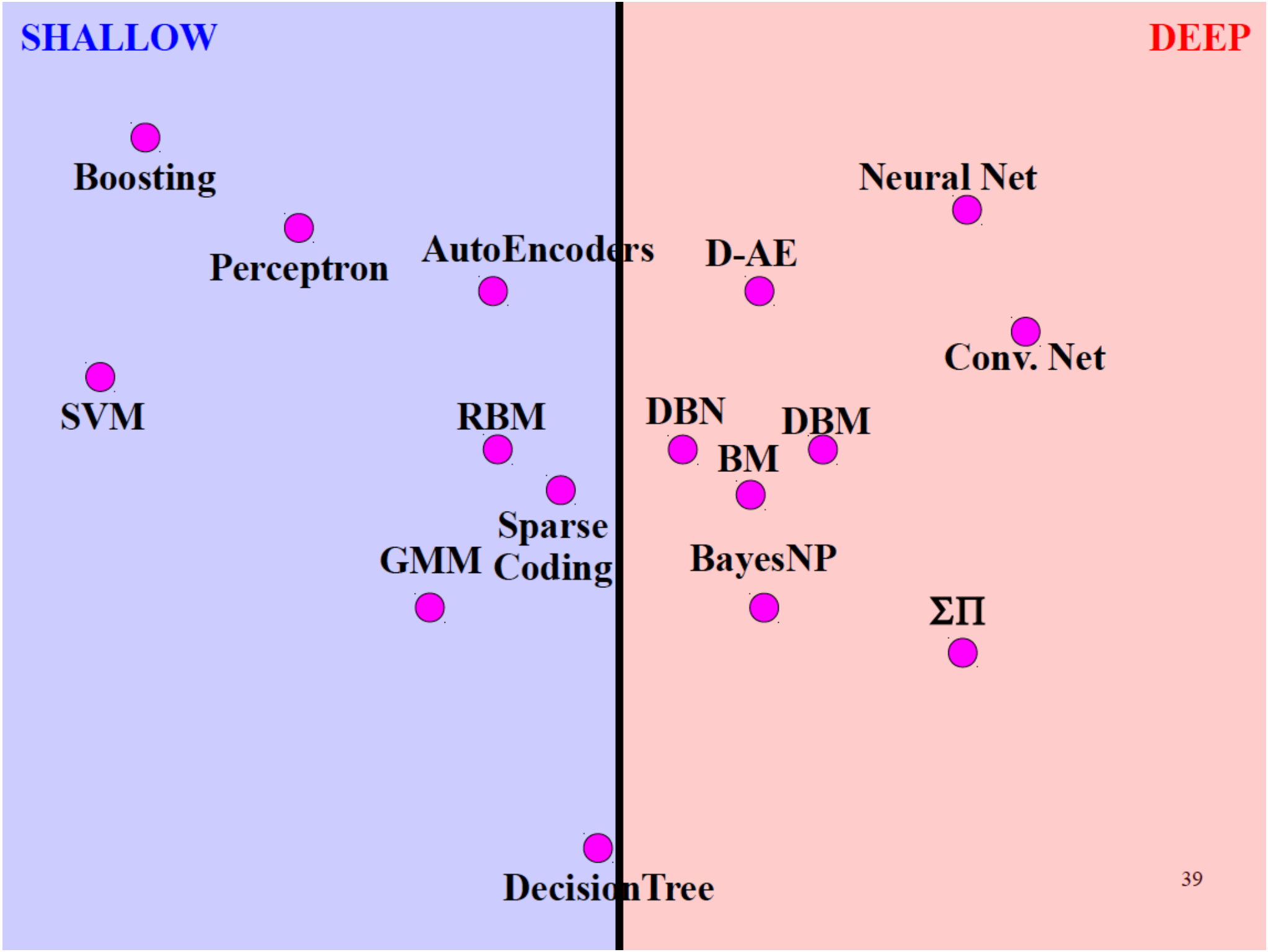
[Slide credit: Ranzato]

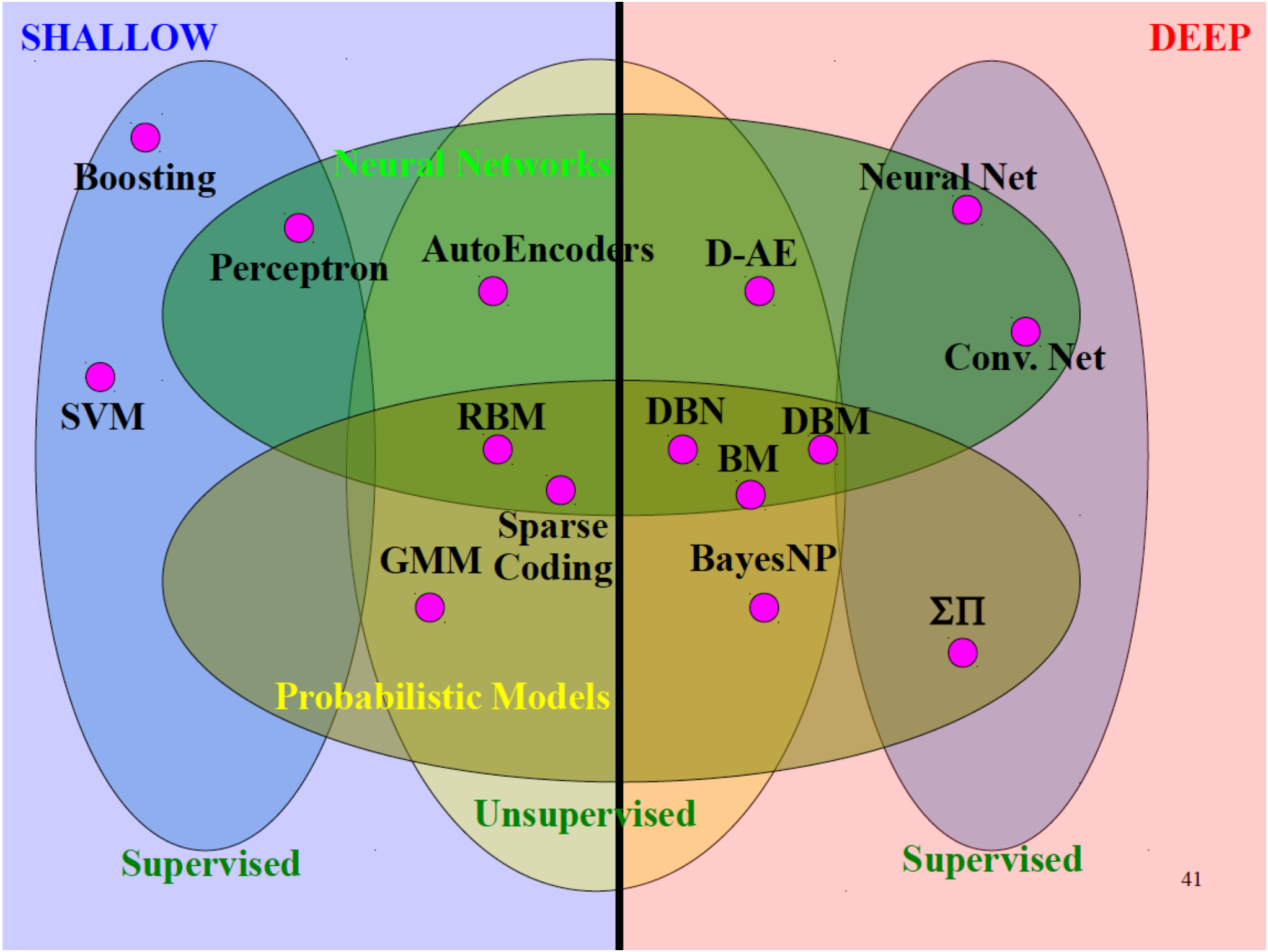


[Slide credit: Ranzato]



[Slide credit: Ranzato]

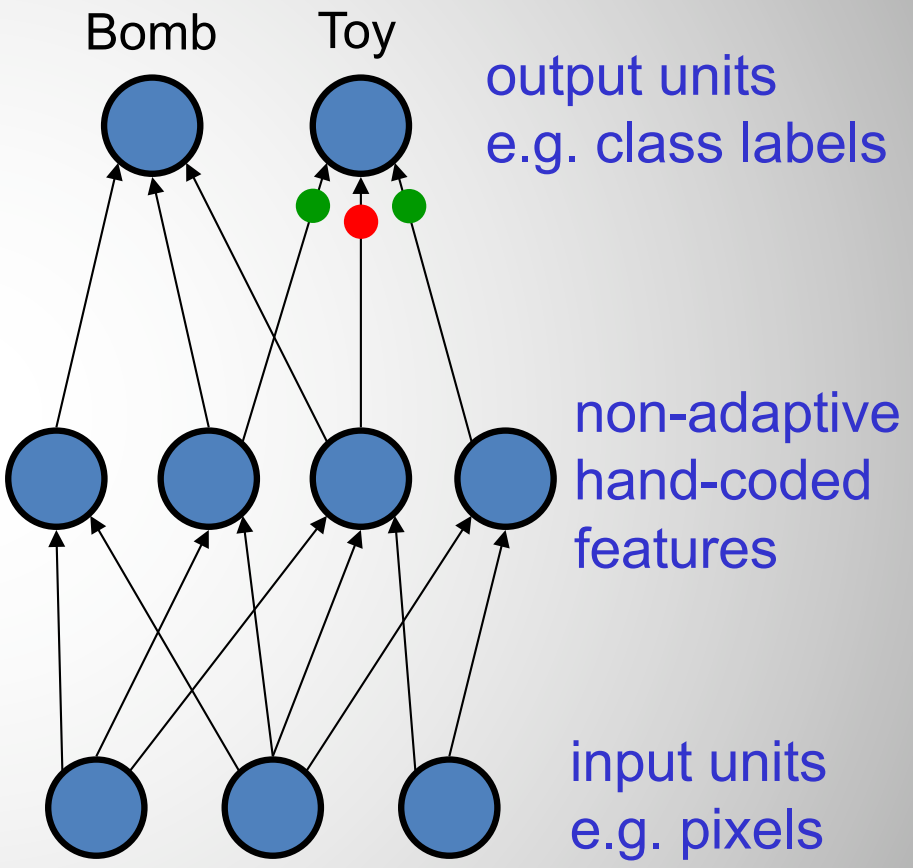

SHALLOW
DEEP
Boosting
Perceptron
AutoEncoders
D-AE
Neural Net
Conv. Net
SVM
RBM
DBN
DBM
BM
Sparse Coding
GMM
BayesNP
ΣΠ
DecisionTree


SHALLOW
DEEP
Boosting
Neural Networks
Neural Net
Perceptron
AutoEncoders
D-AE
Conv. Net
SVM
RBM
DBN
DBM
BM
Sparse Coding
GMM
BayesNP
ΣΠ
Probabilistic Models
Unsupervised
Supervised
Supervised

Earlier Studies

First Generation Neural Networks

- Perceptrons (~1960) used a layer of hand-coded features and tried to recognize objects by learning how to weight these features
  - Fundamentally limited in what they can learn to do

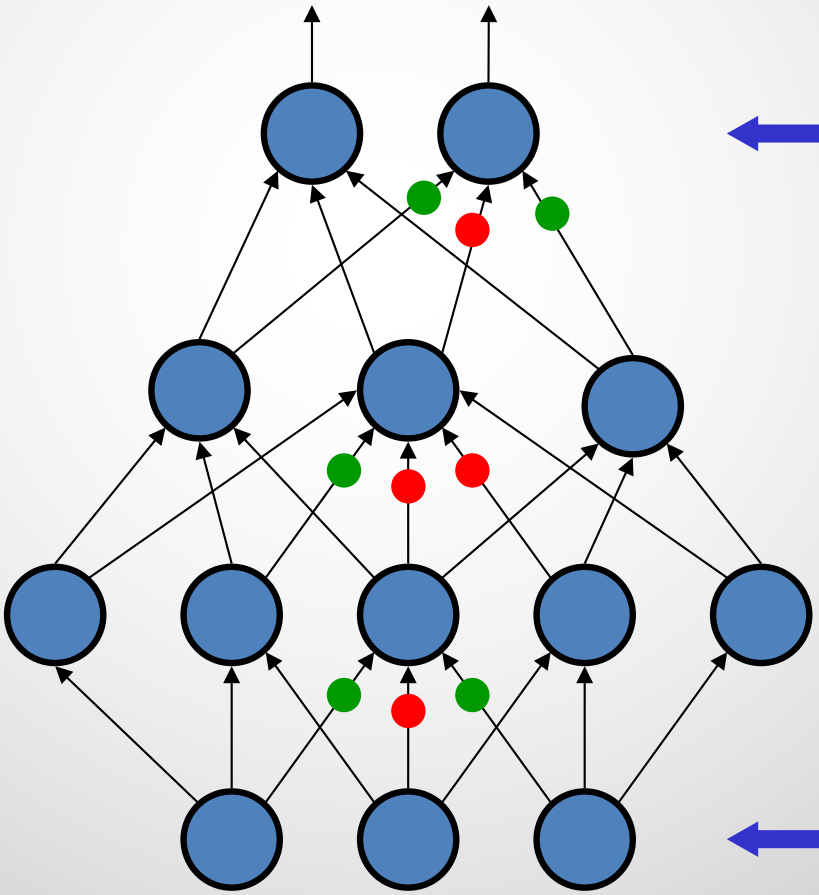


Sketch of a typical perceptron from the 1960's

Second Generation Neural Networks (~1985)

Back-propagate error signal to get derivatives for learning

Compare outputs with correct answer to get error signal



outputs

hidden layers

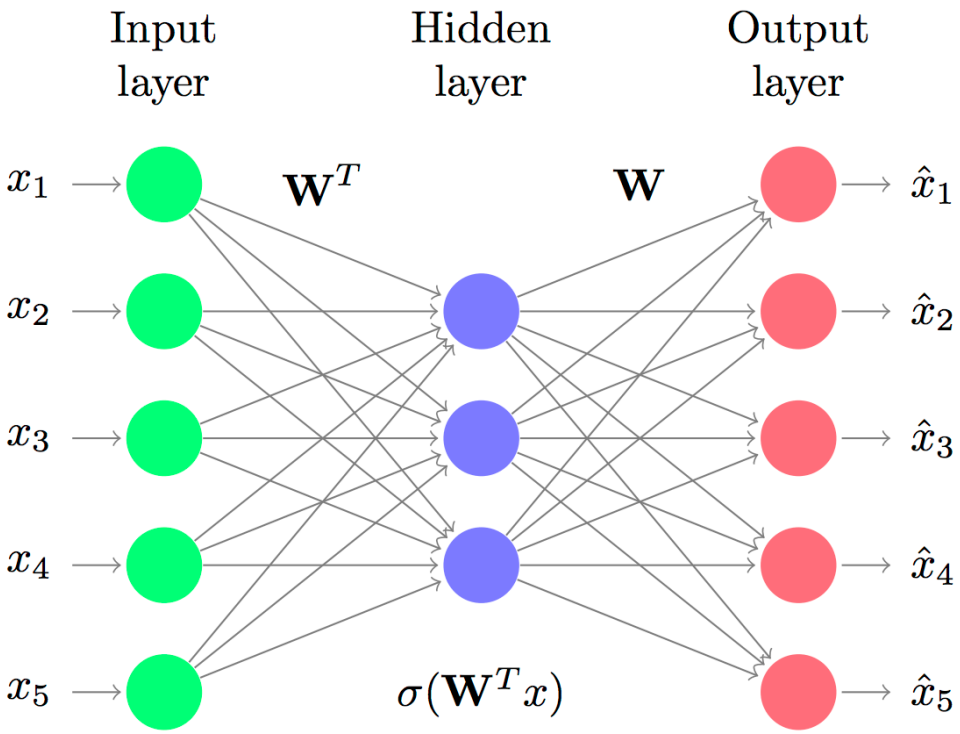
input vector

AutoEncoders

- Generalize the idea of principal components.
- Remember that with PCA, with limited number of samples (components), we try to reconstruct the input signal itself.
- A type of unsupervised learning which tries to discover generic features of the data

AutoEncoders

- Generalize the idea of principal components.



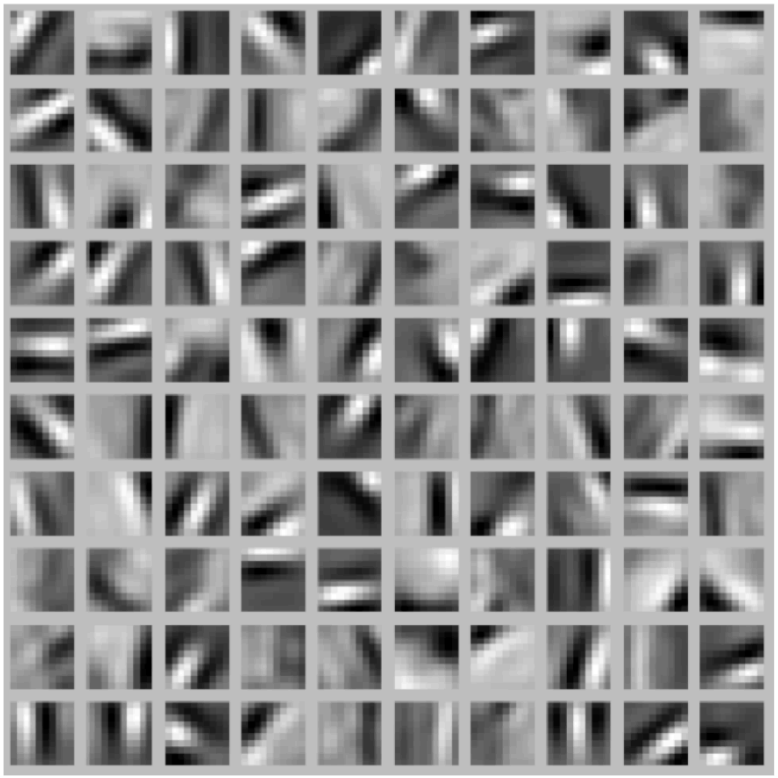
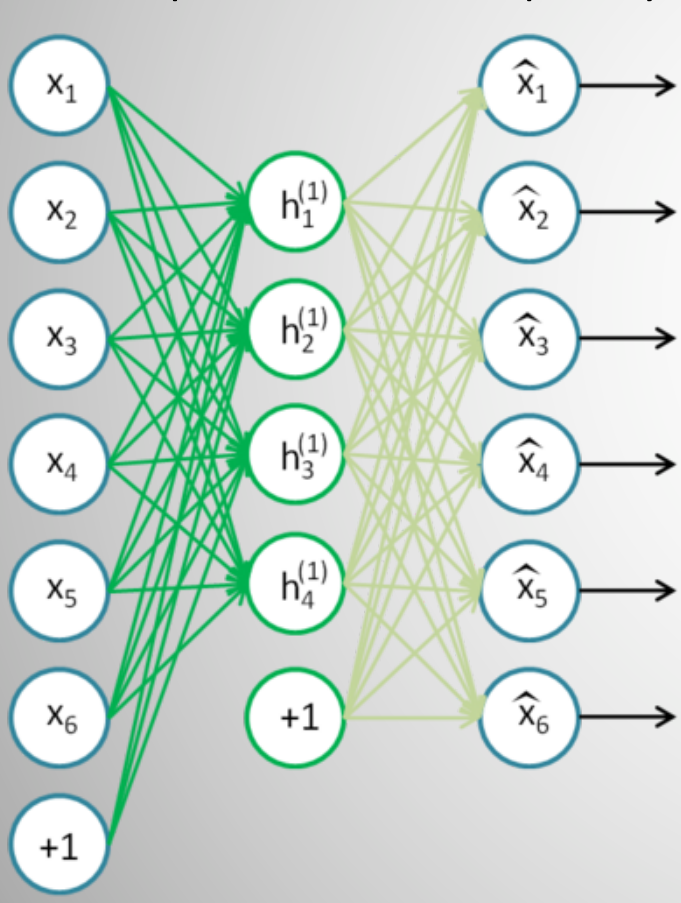
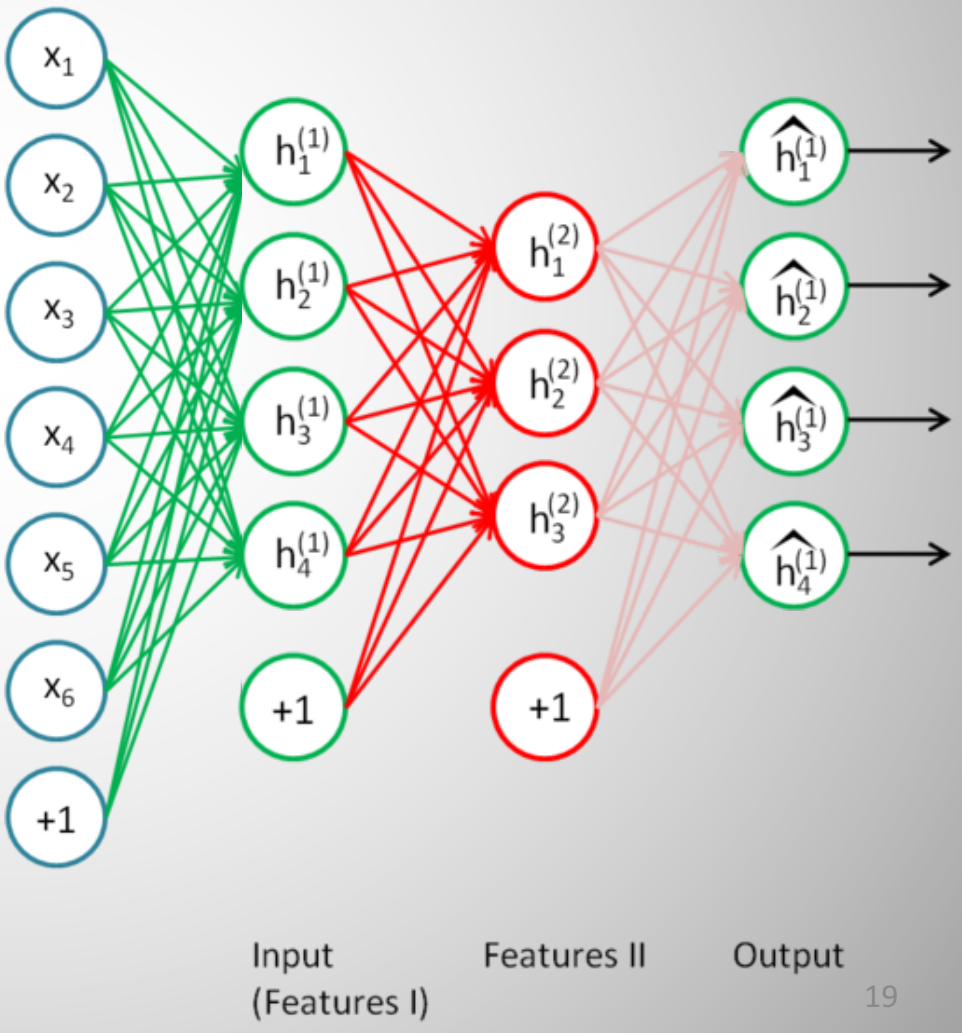


Figure : Left: Network representation of an autoencoder used for unsupervised learning of nonlinear principal components. The middle layer of hidden units creates a bottleneck, and learns nonlinear representations of the inputs. The output layer is the transpose of the input layer, and so the network tries to reproduce the input data using this restrictive representation. Right: Images representing the estimated columns of $\mathbf{W}$ in an image modeling task.

Stacked AutoEncoders

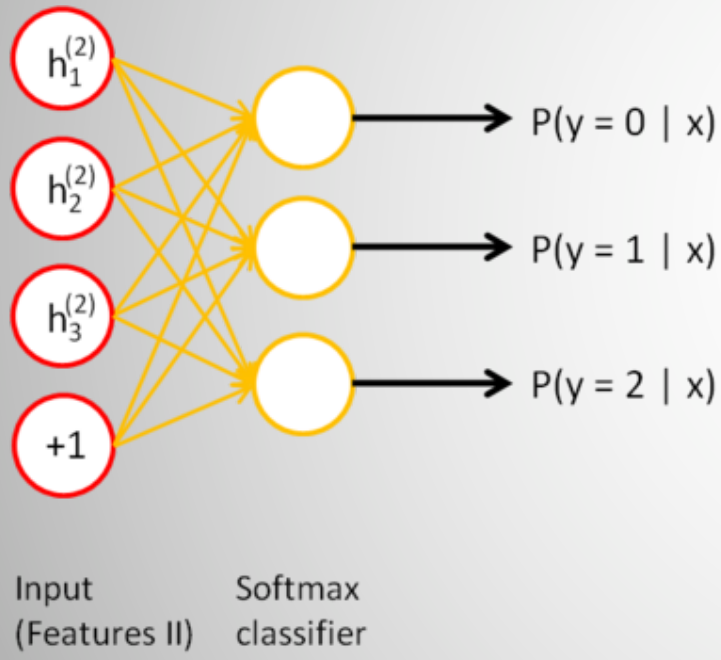
- Bengio (2007) – After Deep Belief Networks (2006)
- Stack many (sparse) auto-encoders in succession and train them using greedy layer-wise training
- Drop the decode output layer each time

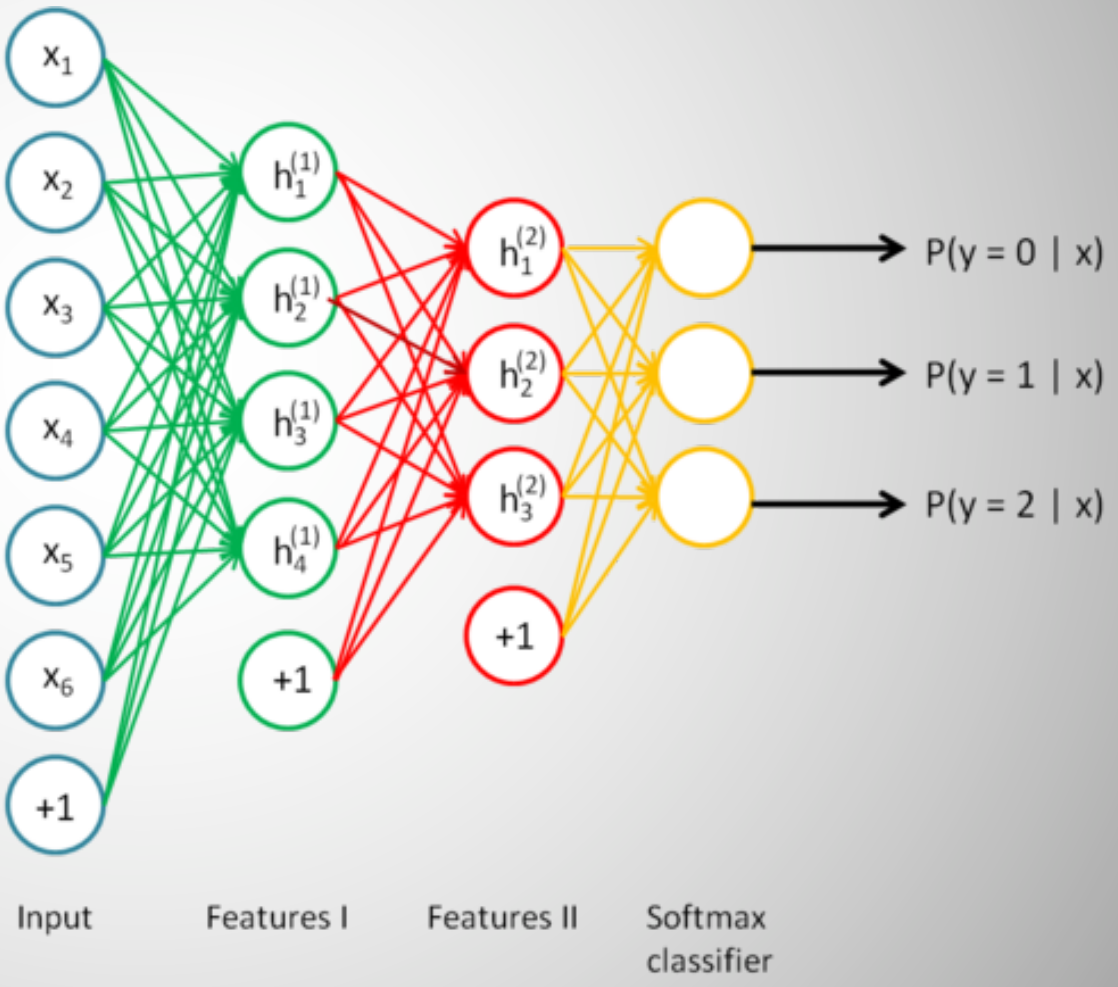




Stacked AutoEncoders

- Do supervised training on the last layer using final features
- Then do supervised training on the entire network to fine- tune all weights





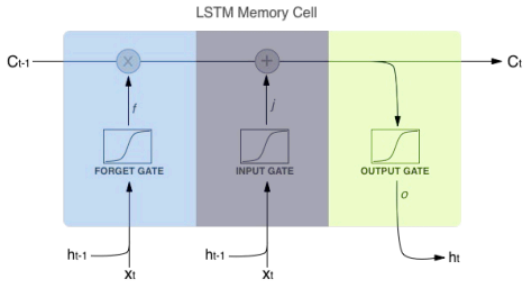
Deep Learning

Why Now?

1. Large Datasets
2. GPU Hardware Advances + Price Decreases
3. Improved Techniques

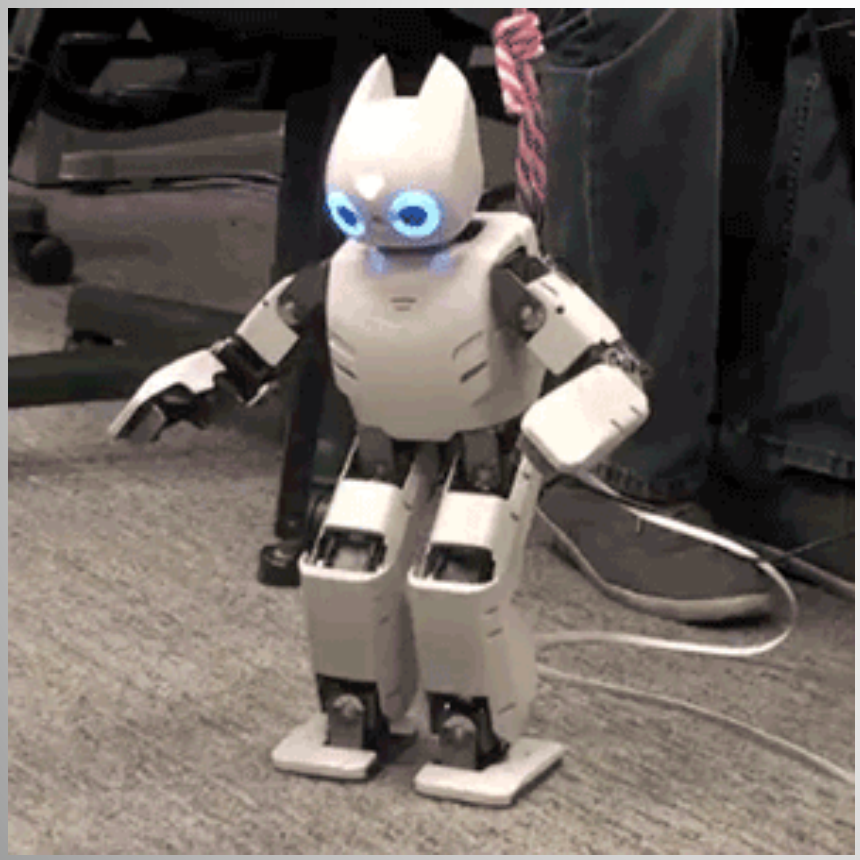
Inception 7a

[1]Going Deeper with Convolutions, [C. Szegedy et al, CVPR 2015]



Robot Toddler Learns to Stand by "Imaging" How to do it (MIT TechReview)

- Instead of being programmed, a robot uses brain-inspired algorithms to "imagine" doing tasks before trying them in the real world.



The new approach is important because it may not always be possible for a robot to indulge in an extensive period of testing. And simulations lack the complexities found in the real world, conditions that with robots can cascade into a catastrophic failure.

Deep Learning yields superior results!

CIFAR image classification	acc.
Prior art [Ciresan et al. 2011]	80.5%
Stanford feature learning	82.0%

NORB image classification	acc.
Prior art [Scherer et al. 2010]	94.4%
Stanford feature learning	95.0%

Hollywood2 video classification	acc.
Prior art [Laptev et al. 2004]	48%
Stanford feature learning	53%

YouTube video classification	acc.
Prior art [Liu et al. 2009]	71.2%
Stanford feature learning	75.8%

KTH video classification	acc.
Prior art [Wang et al. 2010]	92.1%
Stanford feature learning	93.9%

UCF video classification	acc.
Prior art [Wang et al. 2010]	85.6%
Stanford feature learning	86.5%

Paraphrase detection	acc.
Prior art [Das & Smith 2009]	76.1%
Stanford feature learning	76.4%

MPQA sentiment detection	acc.
Prior art [Nakagawa et al. 2010]	77.3%
Stanford feature learning	77.7%

AVLetters lip reading	acc.
Prior art [Zhao et al. 2009]	58.9%
Stanford feature learning	65.8%

(even these results are now beaten by some “other” deep learning based systems)

Successful Deep NN Applications

- Microsoft uses DL for speech rec. service (audio video indexing), based on Hinton/Toronto's DBNs (Mohamed et al 2012)
- Google uses DL in its Google Goggles service, using Ng/Stanford DL systems, and in its Google+ photo search service, using deep convolutional nets
- NYT talks about these: http://www.nytimes.com/2012/06/26/technology/in-a-big-network-of-computers-evidence-of-machine-learning.html?_r=1
- Substantially beating SOTA in language modeling (perplexity from 140 to 102 on Broadcast News) for speech recognition (WSJ WER from 16.9% to 14.4%) (Mikolov et al 2011) and translation (+1.8 BLEU) (Schwenk 2012)
- SENNA: Unsup. pre-training + multi-task DL reaches SOTA on POS, NER, SRL, chunking, parsing, with >10x better speed & memory (Collobert et al 2011)
- Recursive nets surpass SOTA in paraphrasing (Socher et al 2011)
- Denoising AEs substantially beat SOTA in sentiment analysis (Glorot et al 2011)
- Contractive AEs SOTA in knowledge-free MNIST (.8% err) (Rifai et al NIPS 2011)
- Le Cun/NYU's stacked PSDs most accurate & fastest in pedestrian detection and DL in top 2 winning entries of German road sign recognition competition

Deep Learning Machine Teaches Itself Chess in 72 Hours, Plays at International Master Level

Imperial College London

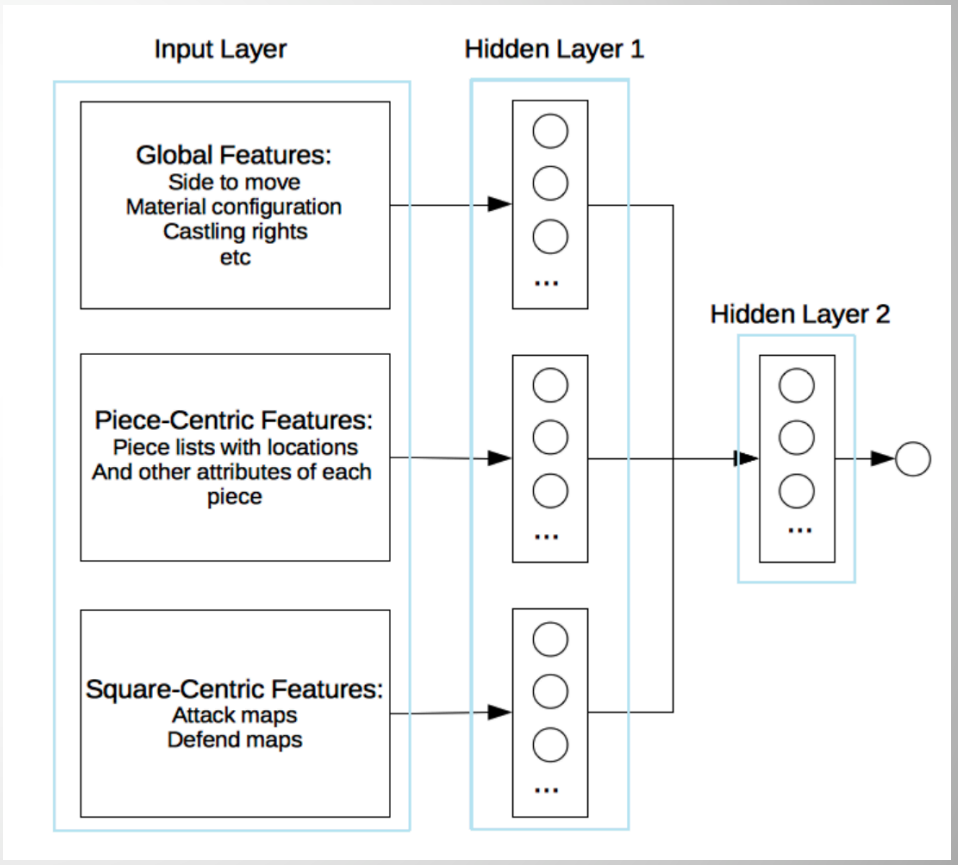
Department of Computing

Giraffe: Using Deep Reinforcement Learning to Play Chess

by

Matthew Lai

Submitted in partial fulfilment of the requirements for the MSc Degree in Advanced Computing of Imperial College London



GIGAOM EVENTS RESEARCH

Apple Cleantech Cloud Data Europe Media Mobile Podcas

artificial intelligence / machine-learning / open source

A startup called Skymind launches, pushing open source deep learning

by Derrick Harris JUN. 2, 2014 - 10:03 AM PDT

No Comments +1

A▼ A▲

SUMMARY: *Skymind is providing commercial support and services for an open source project called deeplearning4j. It's a collection of of approaches to deep learning that mimic those developed by leading researchers, but tuned for enterprise adoption.*

Microsoft

The Fire Hose

Covering the news of the day at Microsoft

TechNet Blogs » The Fire Hose » Bing helps you find better images in your searches with 'deep learning'

Bing helps you find better images in your searches with 'deep learning'

22 Nov 2013 9:21 AM

China's Baidu Bets on Deep Learning

MIT Technology Review - 3 days ago

Deep learning makes it possible for machines to process large amounts of date using simulated networks of simple neurons, crudely modeled ...

Baidu snatches Google's deep-learning visionary, Andrew ...

VentureBeat - 3 days ago

Microsoft's Skype "Star Trek" Language Translator Takes on Tower of Babel

By Ina Fried

ETHICS BIO ARTICLES

May 27, 2014, 5:48 PM PDT

Remember the universal translator on Star Trek? The gadget that let Kirk and Spock talk to aliens?

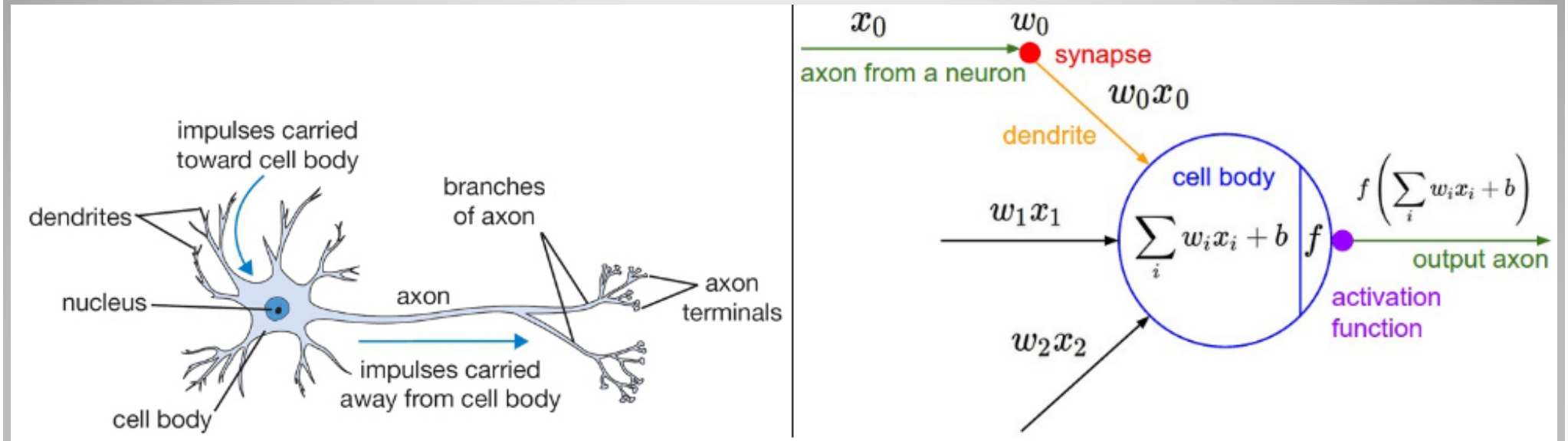
A reasonable definition for DL

- Deep learning is a method that makes predictions using a sequence of non-linear processing stages

- The resulting intermediate representations can be interpreted as feature hierarchies and the whole system is jointly learned from data

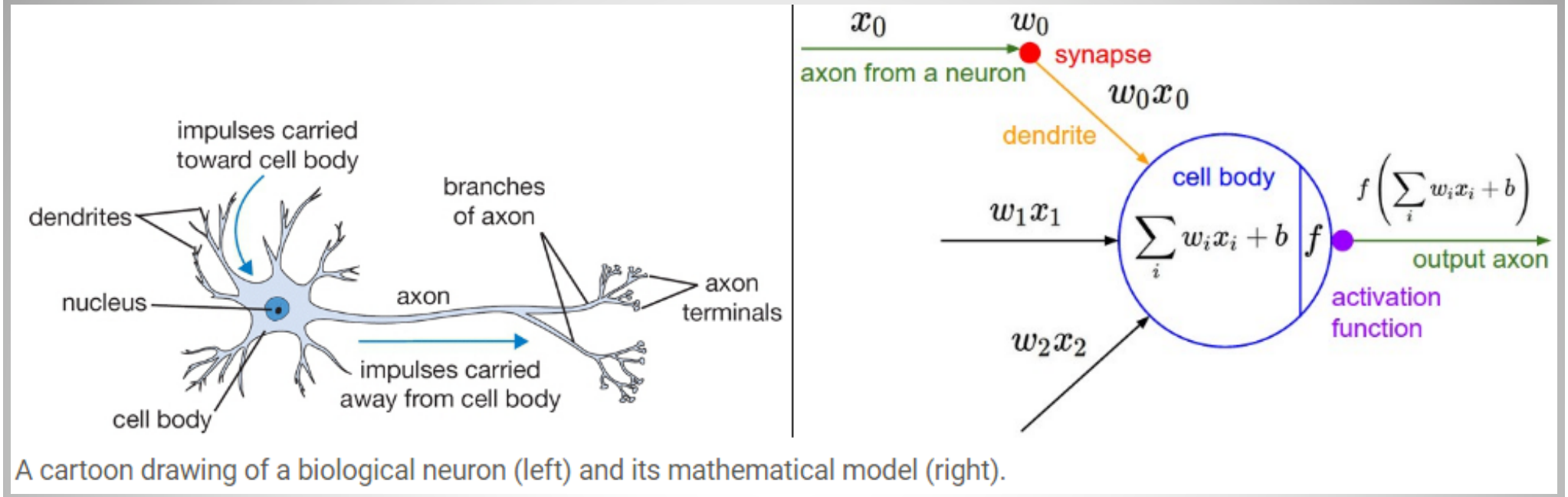
Fundamentals

Inspiration Behind Neural Networks

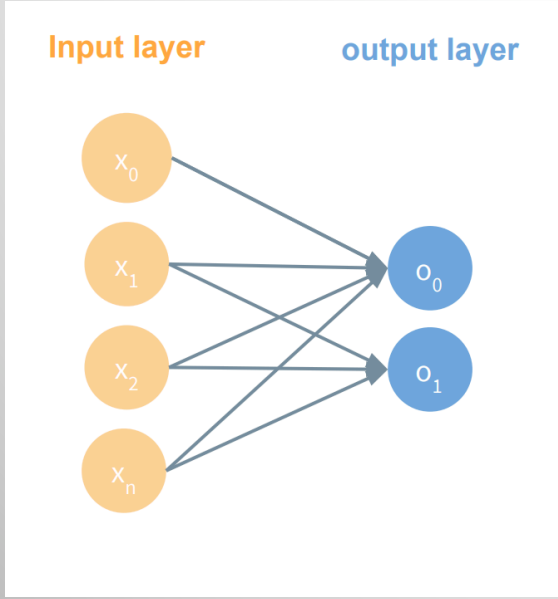


A cartoon drawing of a biological neuron (left) and its mathematical model (right).

Inspiration Behind Neural Networks

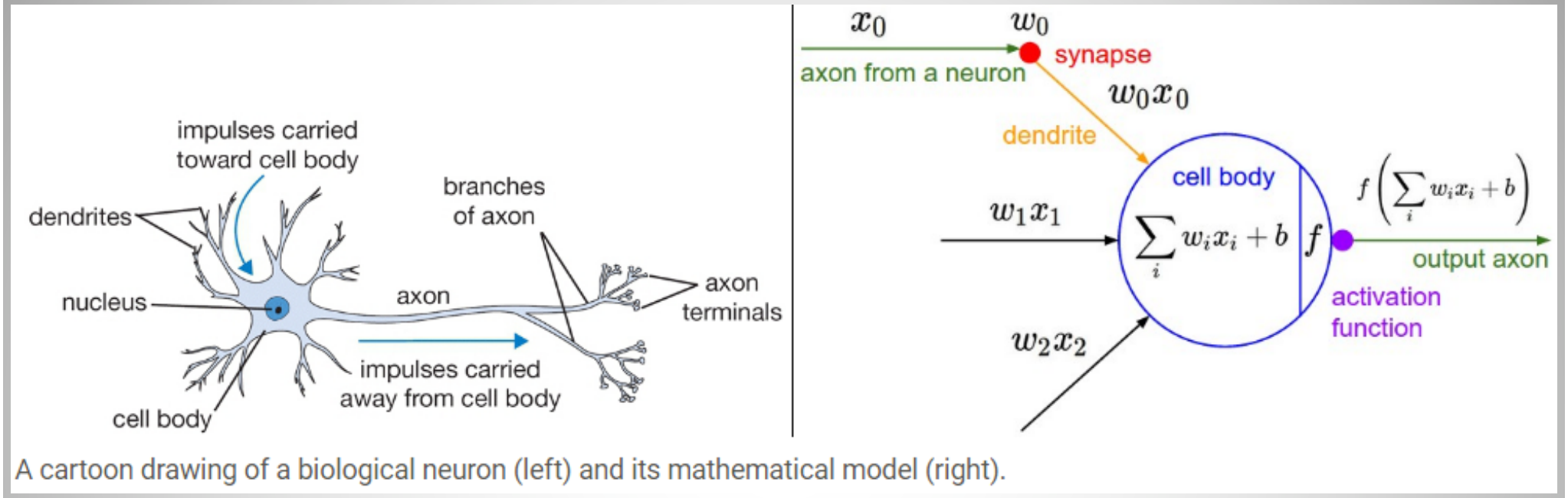


A cartoon drawing of a biological neuron (left) and its mathematical model (right).

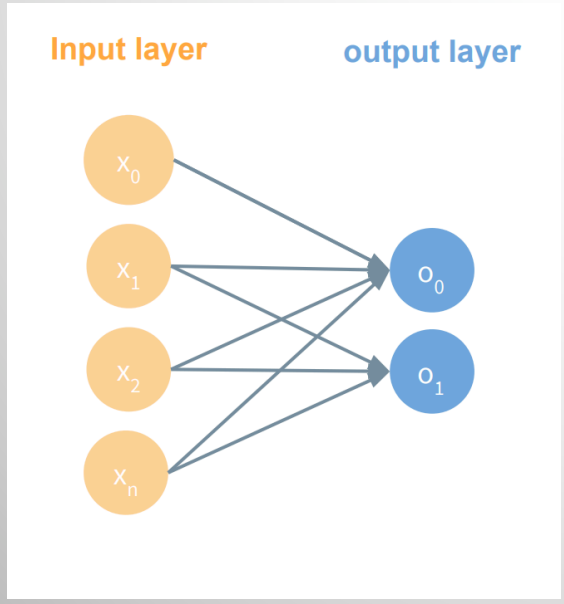


Multi-output perceptron

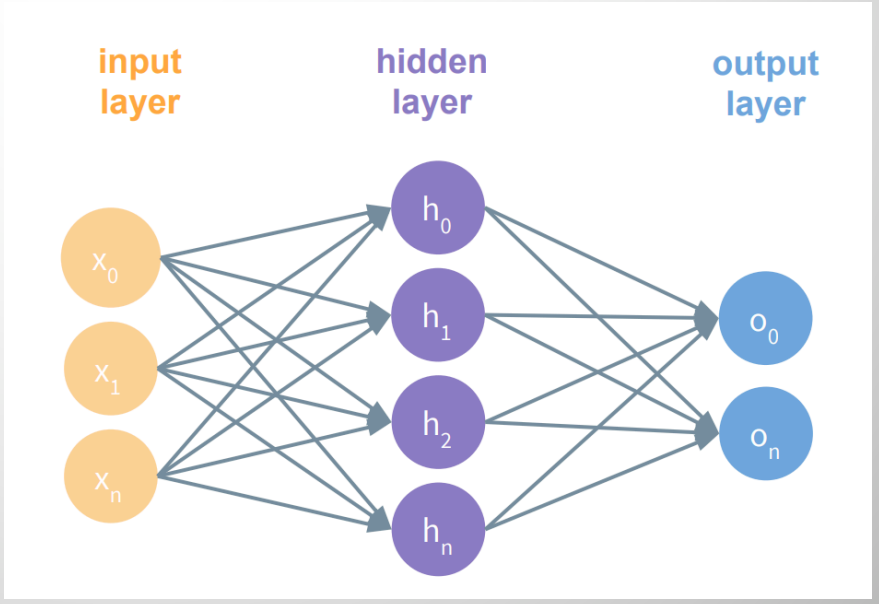
Inspiration Behind Neural Networks



A cartoon drawing of a biological neuron (left) and its mathematical model (right).

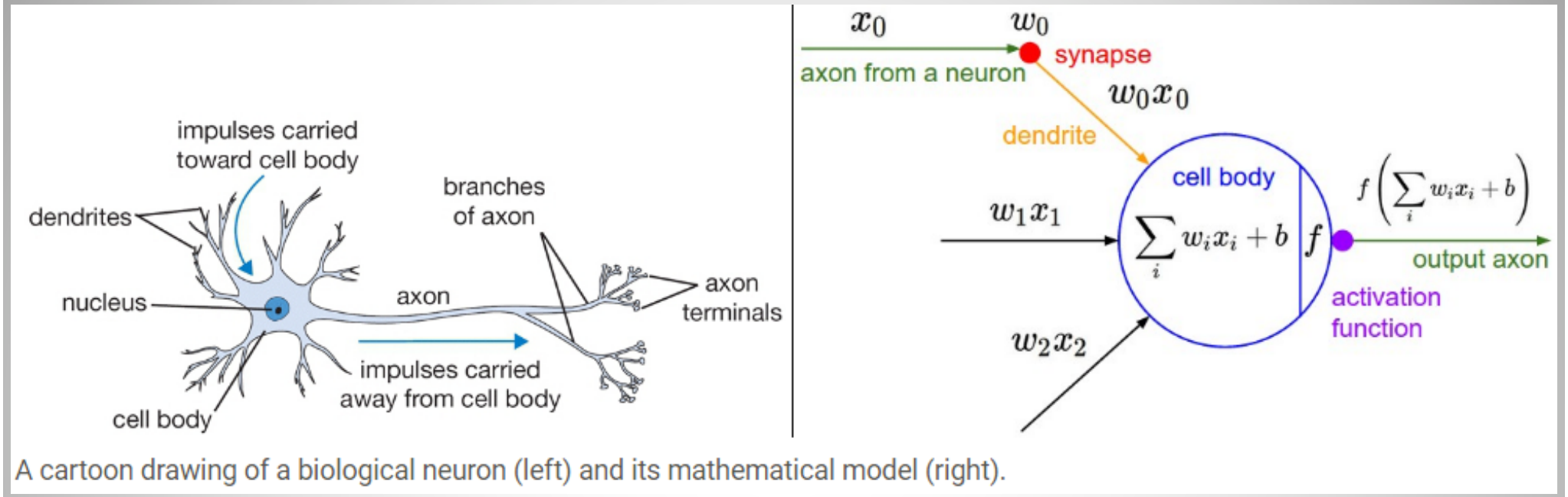


Multi-output perceptron

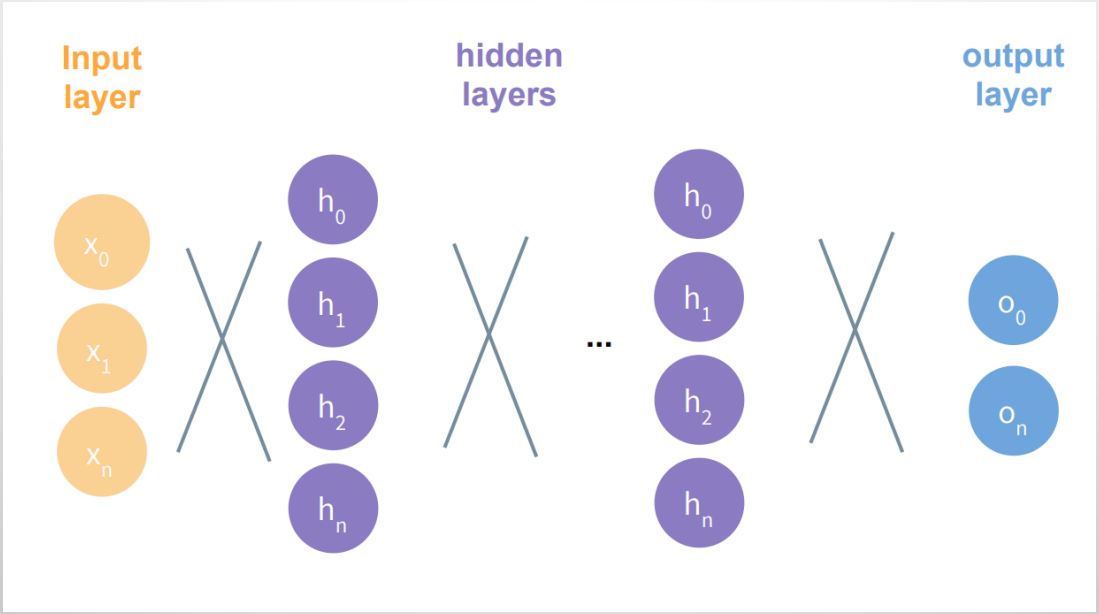


Multi-layer perceptron (MLP)

Inspiration Behind Neural Networks



A cartoon drawing of a biological neuron (left) and its mathematical model (right).



Deep Neural Networks

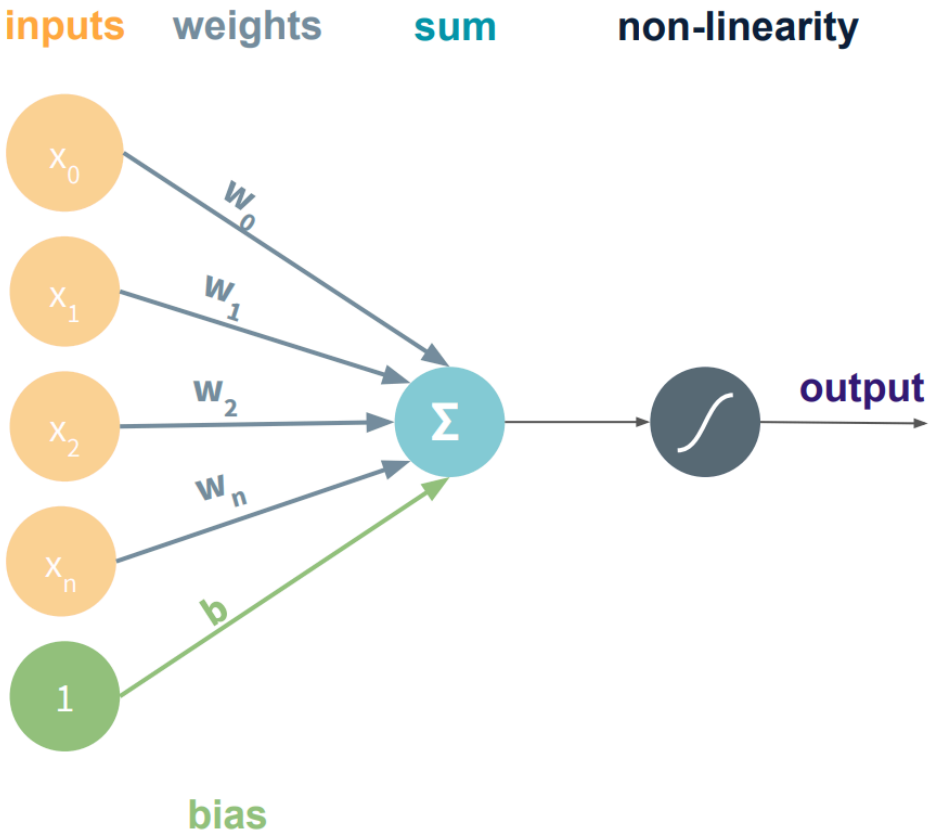
Perceptron Forward Pass

Activation Function

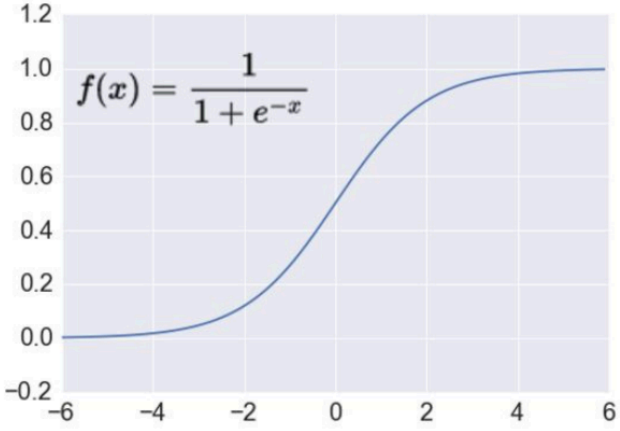
$$output = g(XW + b)$$

$$X = x_0, x_1, \ldots x_n$$

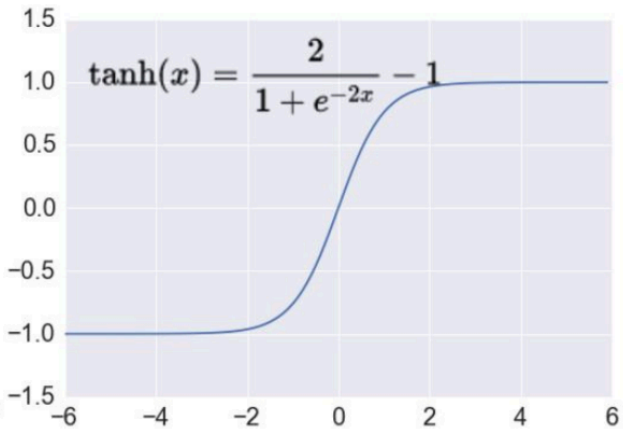
$$W = w_0, w_1, \ldots w_n$$



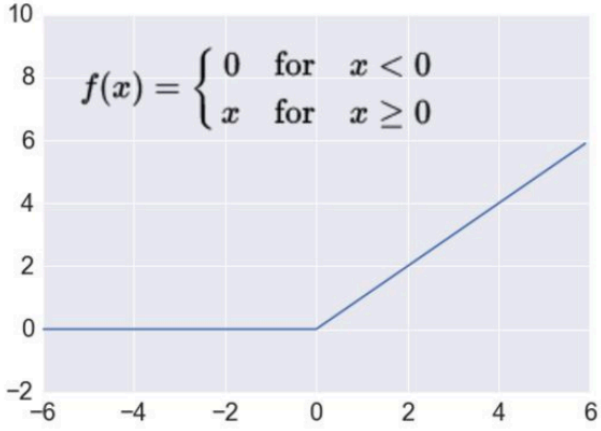
Sigmoid



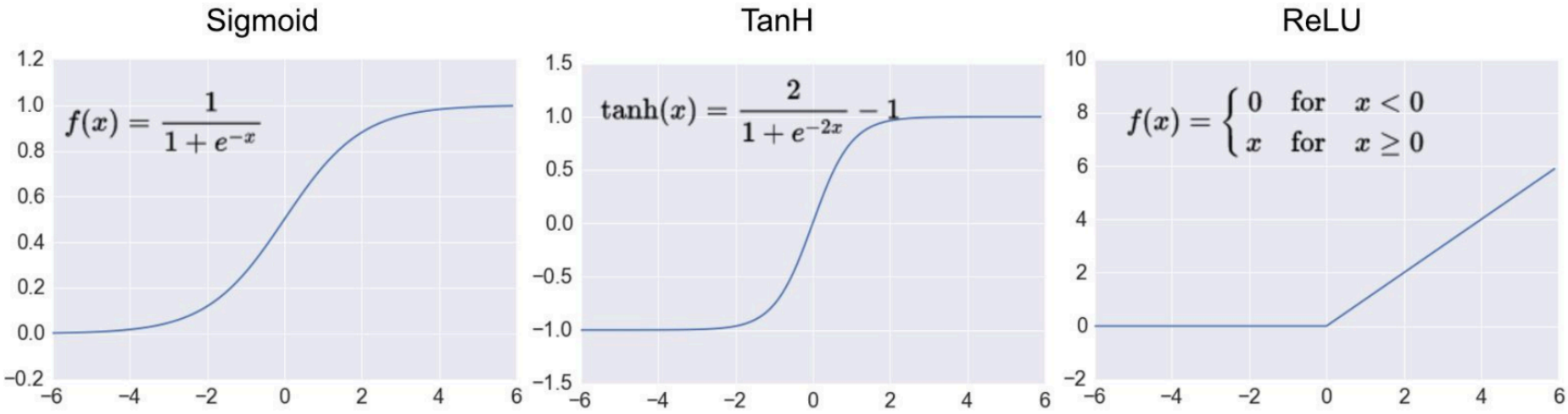
TanH



ReLU



Comparison of Activation Functions



+ providing non-linearity

- Sigmoid/tanh saturate and kill gradients

- Sigmoid/tanh neurons involve expensive operations (exponentials, etc.

+ Inexpensive

+ greatly accelerate the convergence of gradient descent compared to sigmoid/tanh

-ReLUs are piecewise linear

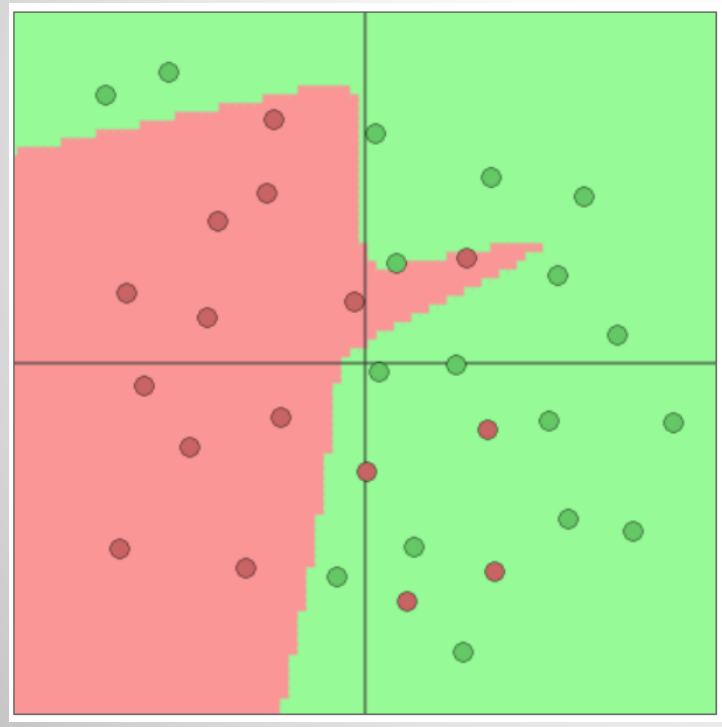
-somewhat temperamental and easily “die”.

+Deeper networks can solve piece-wise linearity

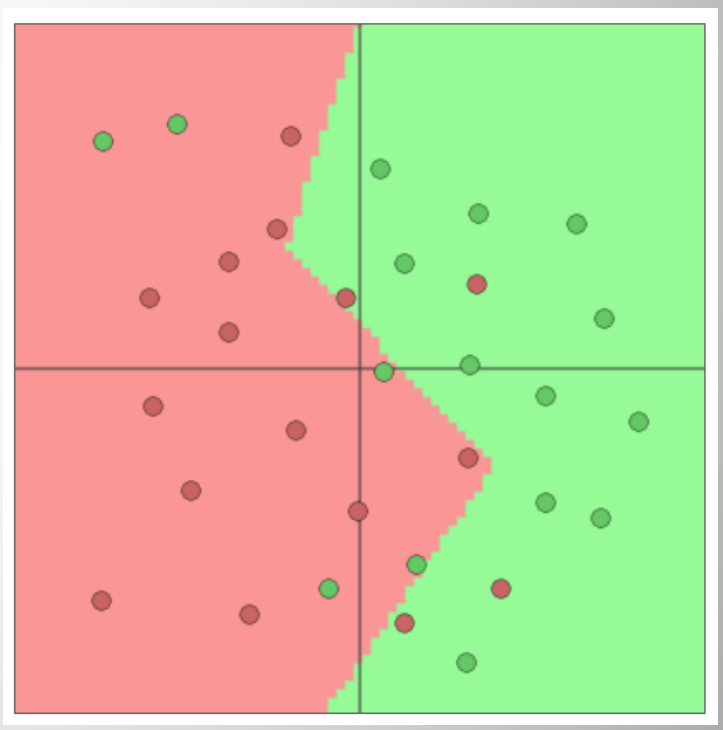
Why Non-linearity Matters ?

- Neural networks are *universal approximators*.
- Given a single hidden fully-connected layer and a non-linear function, a network can approximate **any** possible continuous function to within any given accuracy, given enough neurons.

3 hidden neurons with tanh



3 hidden neurons with ReLU

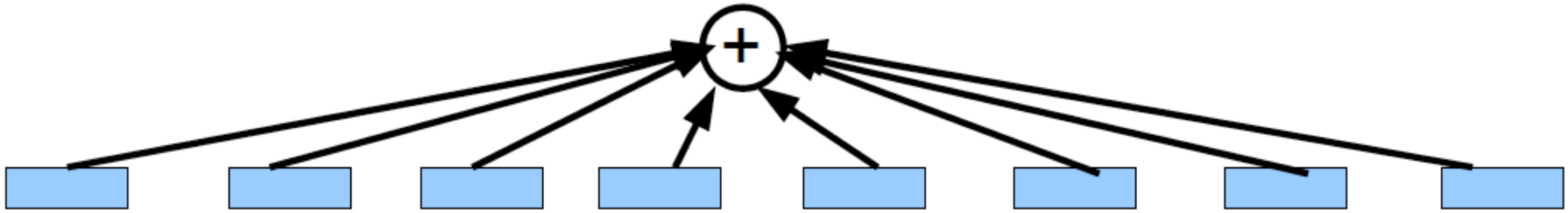


Breadth vs. Depth
&
Summation vs. Convolution

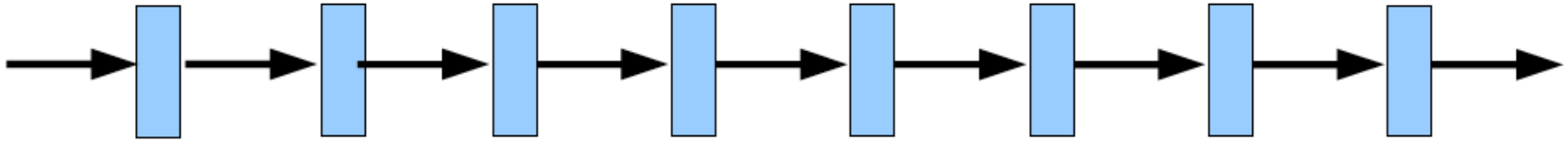
Learning Non-Linear Features

Given a dictionary of simple non-linear functions: $g_1, \ldots, g_n$

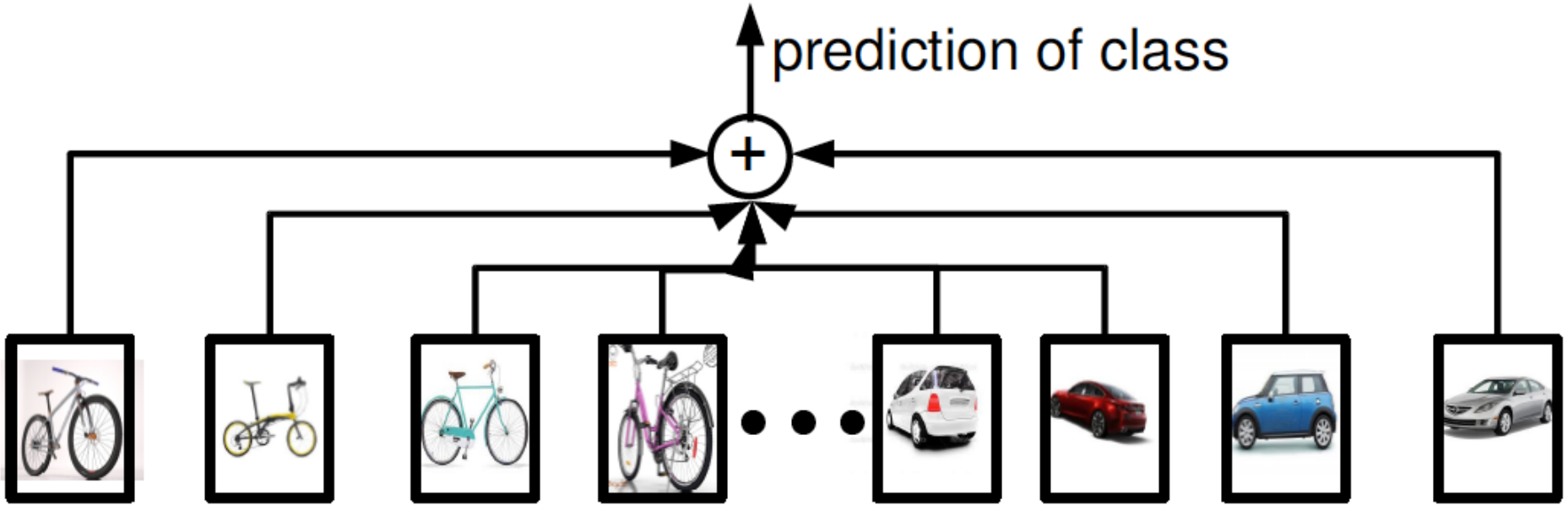
Proposal #1: linear combination $f(x) \approx \sum_j g_j$



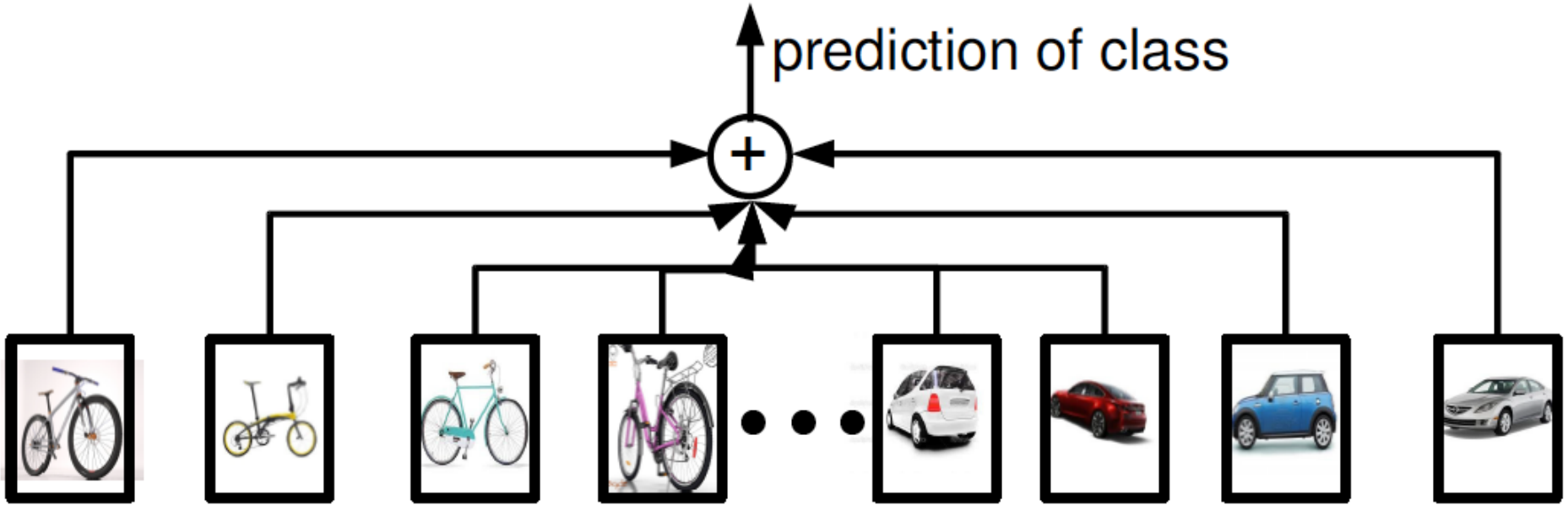
Proposal #2: composition $f(x) \approx g_1(g_2(\ldots g_n(x)\ldots))$



Proposal 1. Combination: Broad & Shallow

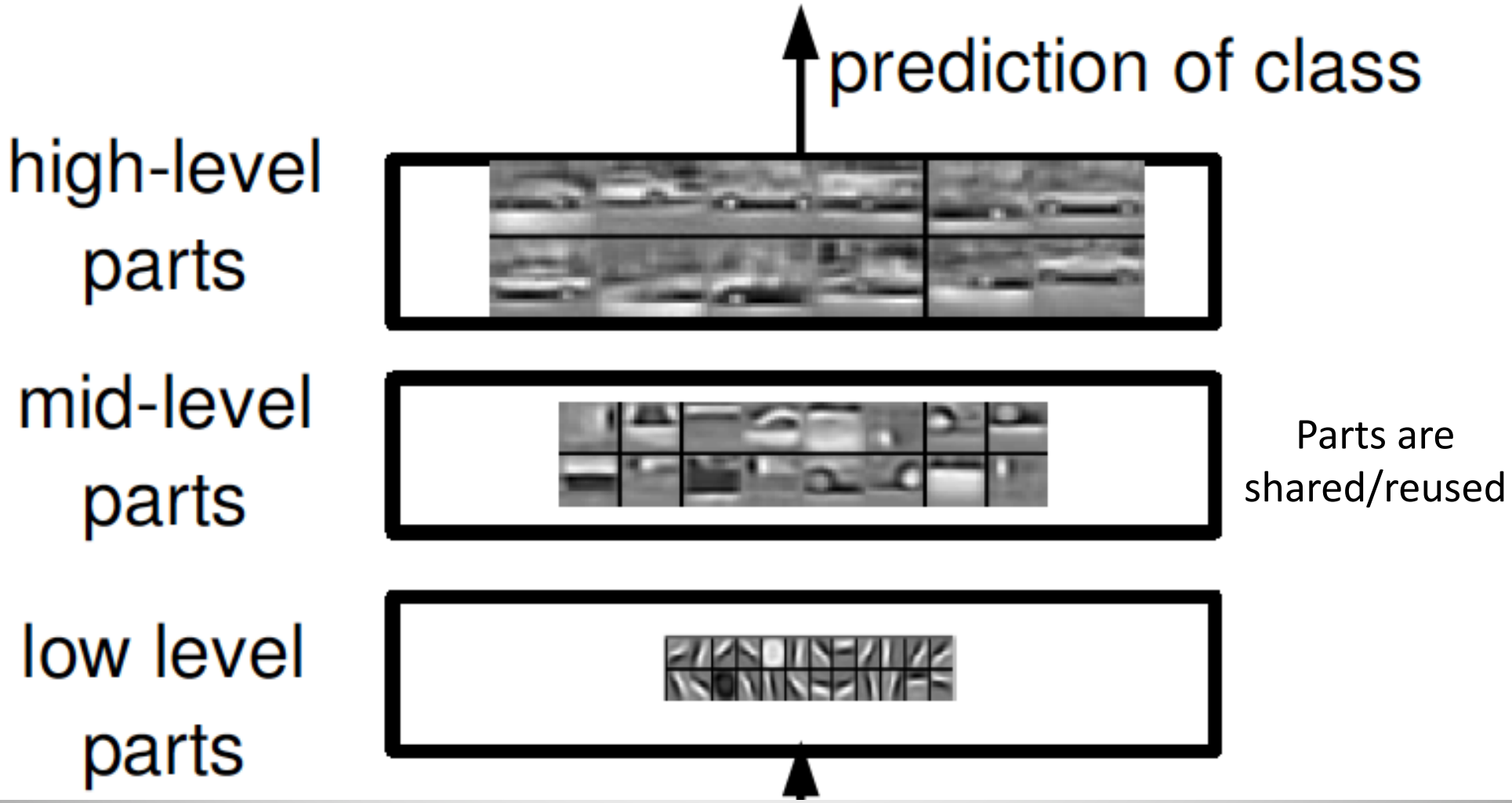


Proposal 1. Combination: Broad & Shallow



Linear combination of classifiers (e.g., templates)
Architecture is like **boosting** or **kernel learning**

Proposal 2. Composition: Narrow & Deep



Can be *exponentially* more efficient!

Why Convolution ?

- Convolution is simply a **mathematical operation** just as addition, subtraction, multiplication, and division are mathematical operations.

- Where the above operations acts on two input numbers to produce a single output number, the **convolution** operation acts on **two input functions** to produce a **single output function**.

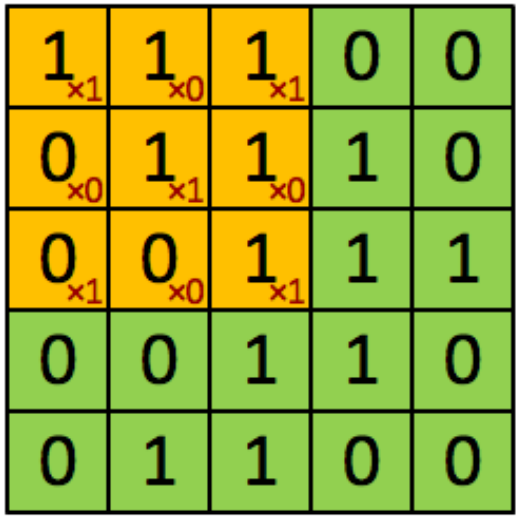
Why Convolution ?

Natural images have the property of being "'stationary"', meaning that the statistics of one part of the image are the same as any other part.

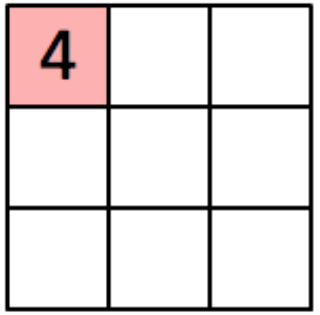
This suggests that the features that we learn at one part of the image can also be applied to other parts of the image, and we can use the same features at all locations.

$$(f * \mathrm{g})(\mathrm{t}) = \sum_{m=0}^{t} f(m) \times g(t-m)$$

where f and g are both functions of t



Image



Convolved Feature

Why Deep?

At least three reasons

- Larger Receptive Field
- Larger Family of Functions
- More Easily Computed Gradients

Network Types

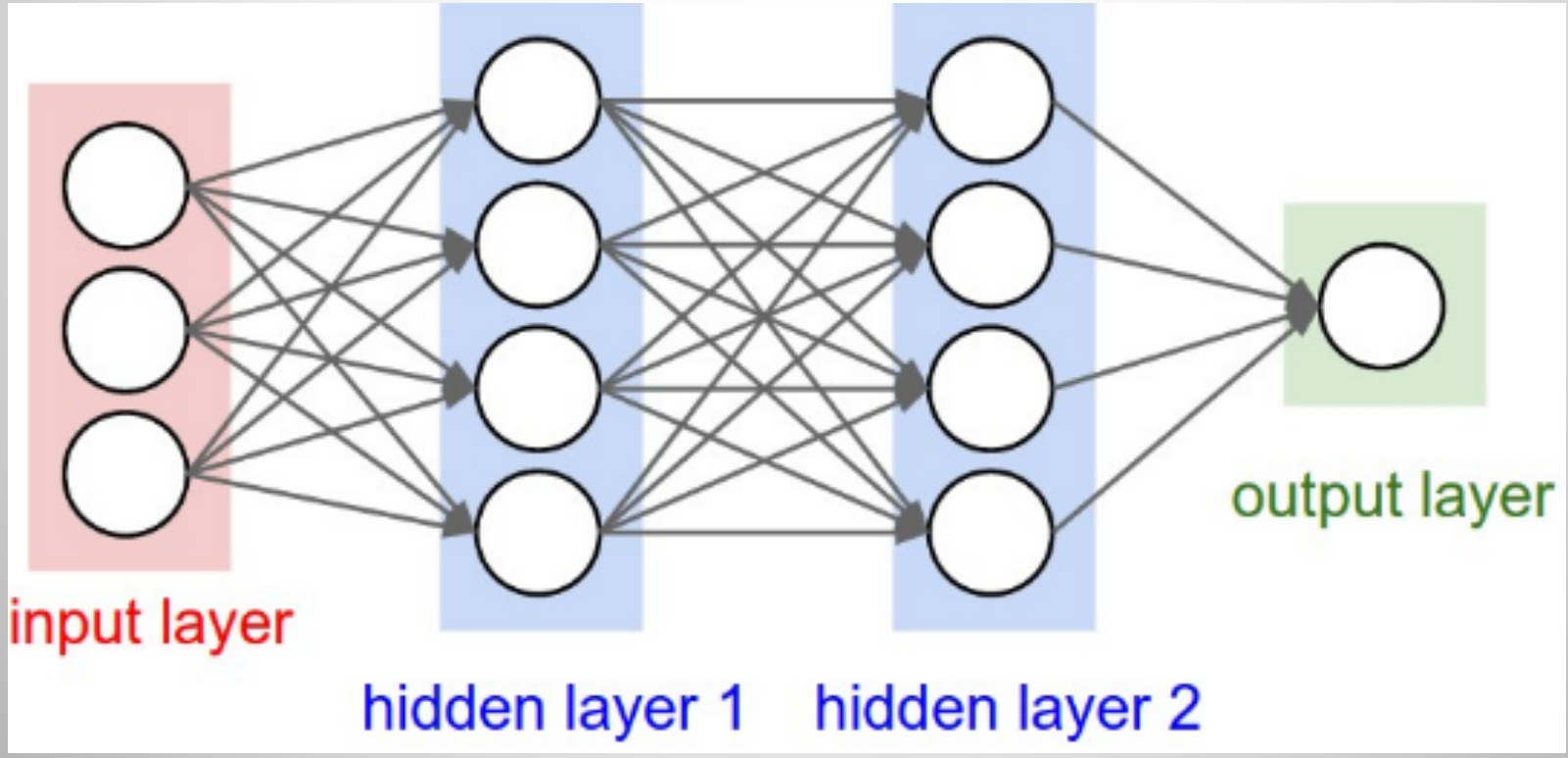
1-Fully Connected Networks (FCN)

2-Locally Connected Networks (LCN)

3-Convolutional Neural Networks (CNN)

1. Fully Connected Network (FCN)

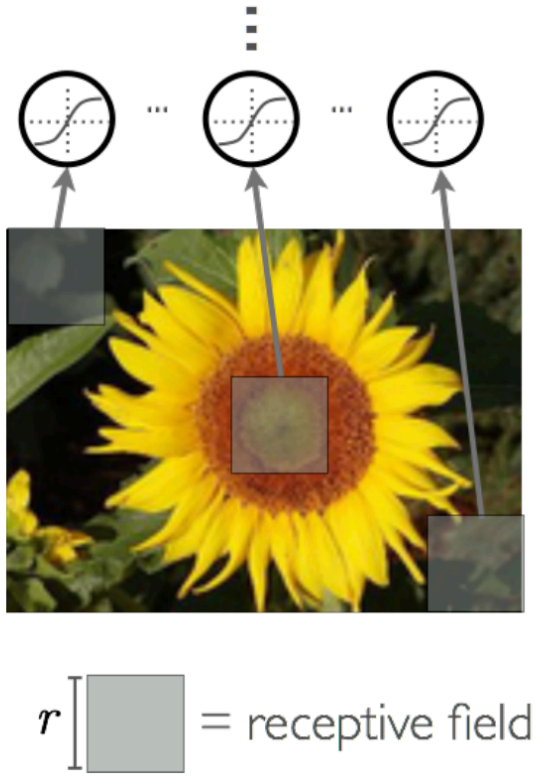
- "Traditional" Neural Networks
- Given m inputs and n outputs, matrix multiplication requires m×n parameters and O(m×n) runtime.



2. Locally Connected Network (LCN)

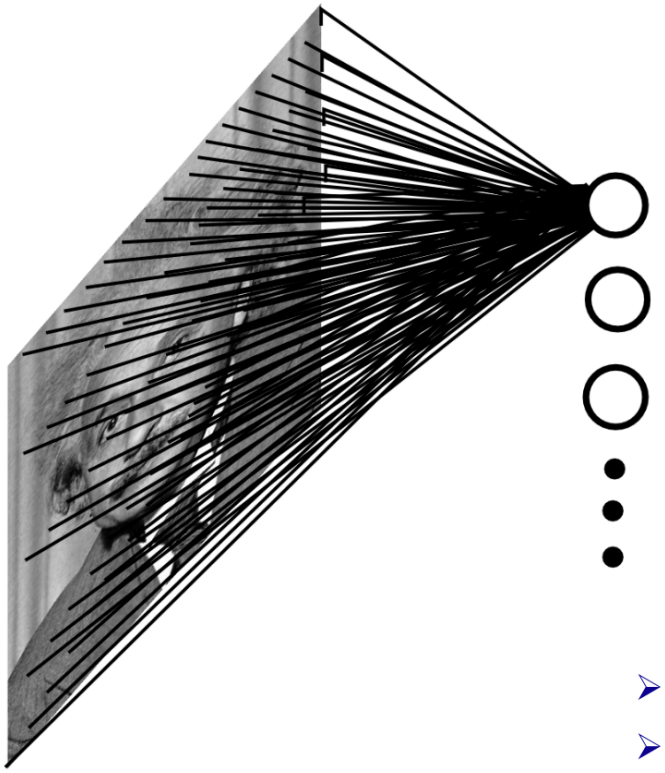
- **Problem in FCN:** very computationally expensive
- **One simple solution:** to restrict the connections between the hidden units and the input units, allowing each hidden unit to connect to only a small subset of the input units.

- Use a local connectivity of hidden units
  - Each hidden unit is connected only to a sub-region (patch) of the input image
  - It is connected to all channels: 1 if grayscale, 3 (R, G, B) if color image
- Why local connectivity?
  - Fully connected layer has a lot of parameters to fit, requires a lot of data
  - Spatial correlation is local



Local Connectivity

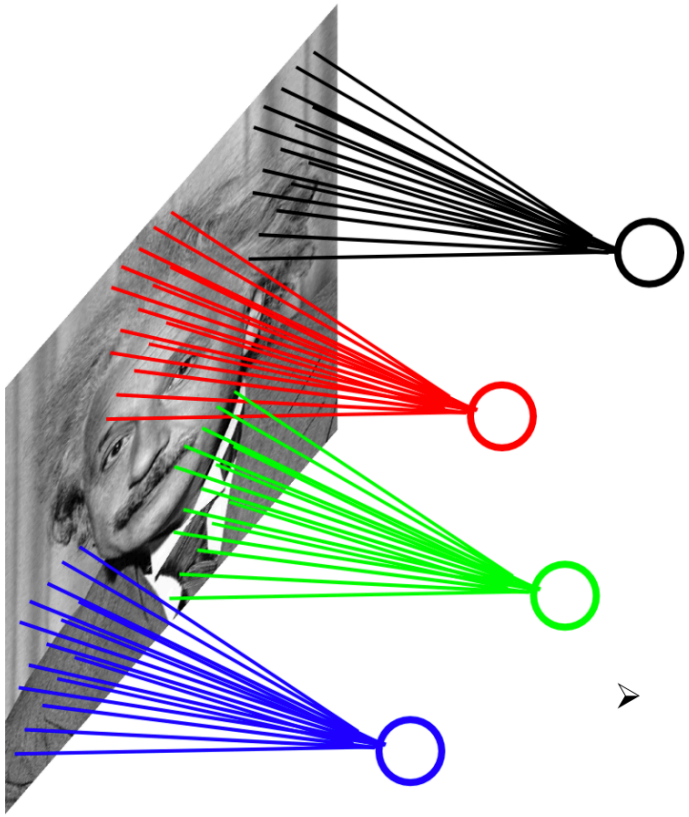
- Example: 200x200 image, 40K hidden units, ~2B parameters!



- Spatial correlation is local
- Too many parameters, will require a lot of training data!

Local Connectivity

• Example: 200x200 image, 40K hidden units, filter size 10x10, 4M parameters!



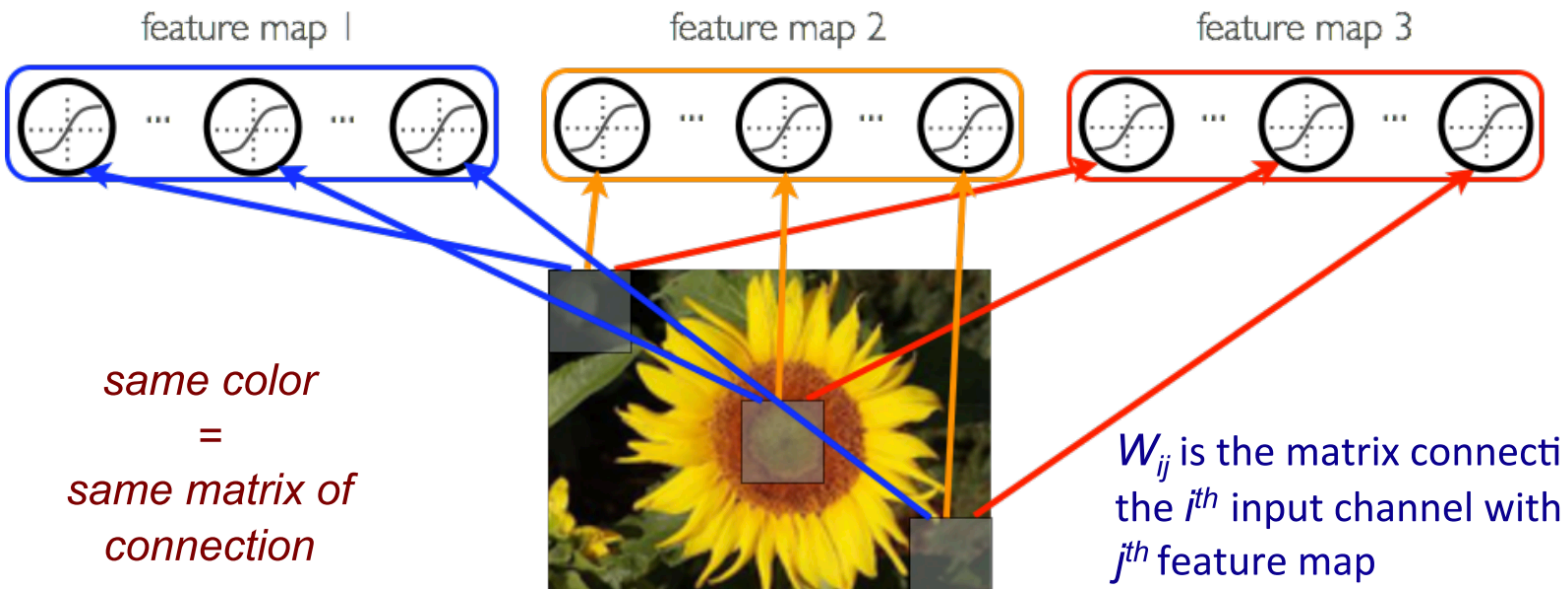
- This parameterization is good when input image is registered

3. Convolutional Neural Networks (CNN)

- Local connectivity
- Parameter sharing
- Convolution
- Pooling / subsampling hidden units

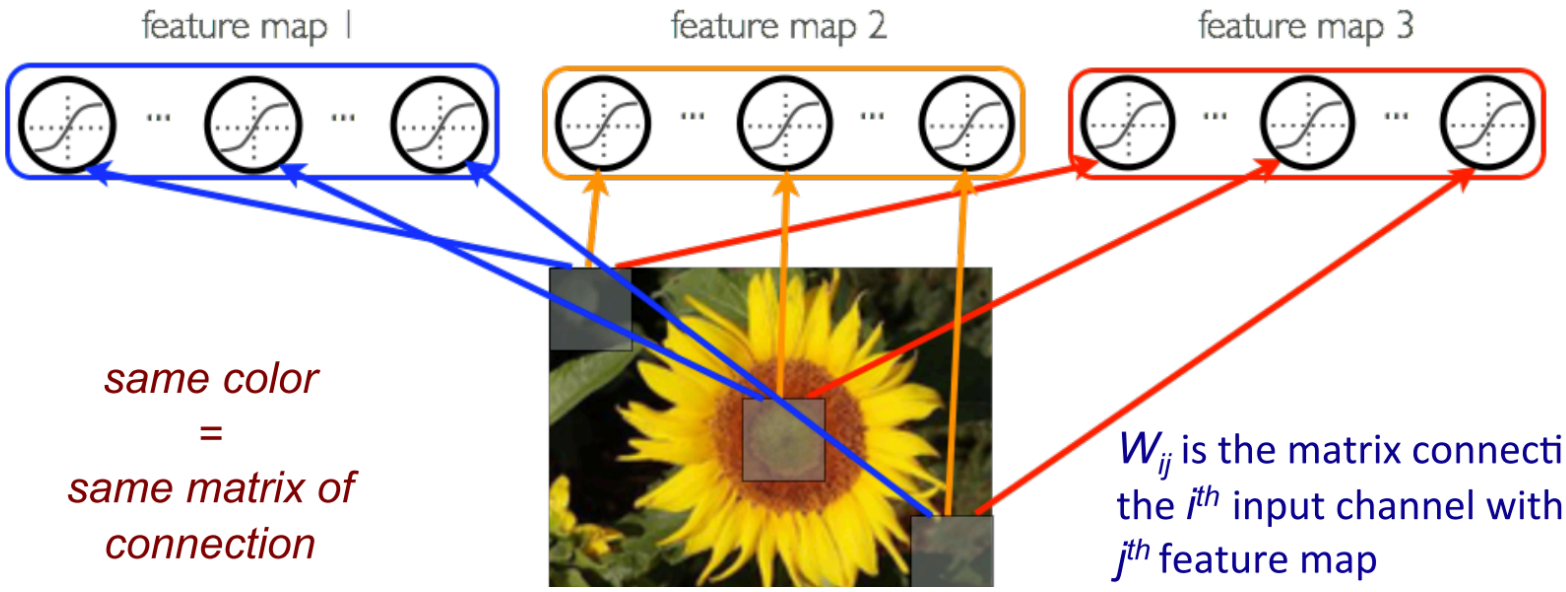
Parameter Sharing

- Share matrix of parameters across some units
  - Units that are organized into the 'feature map" share parameters
  - Hidden units within a feature map cover different positions in the image



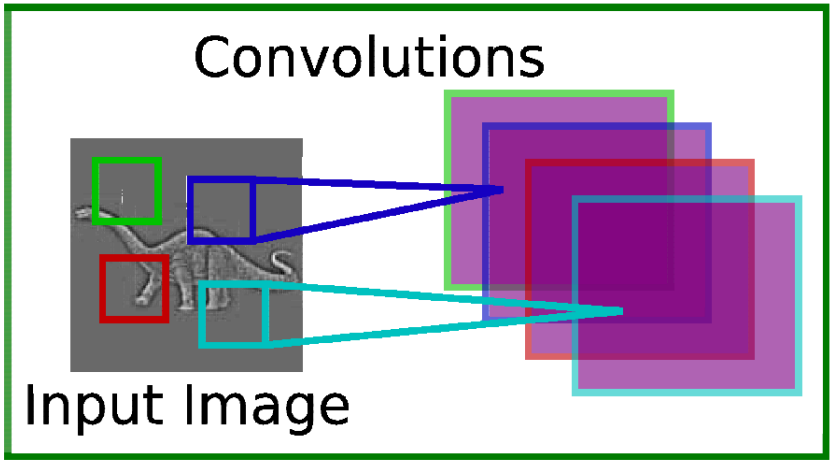
Parameter Sharing

- Why parameter sharing?
  - Reduces even more the number of parameters
  - Will extract the same features at every position (features are "equivariant")



Parameter Sharing

- Each feature map forms a 2D grid of features
  - can be computed with a discrete convolution ($*$) of a kernel matrix k_{ij} which is the hidden weights matrix W_{ij} with its rows and columns flipped



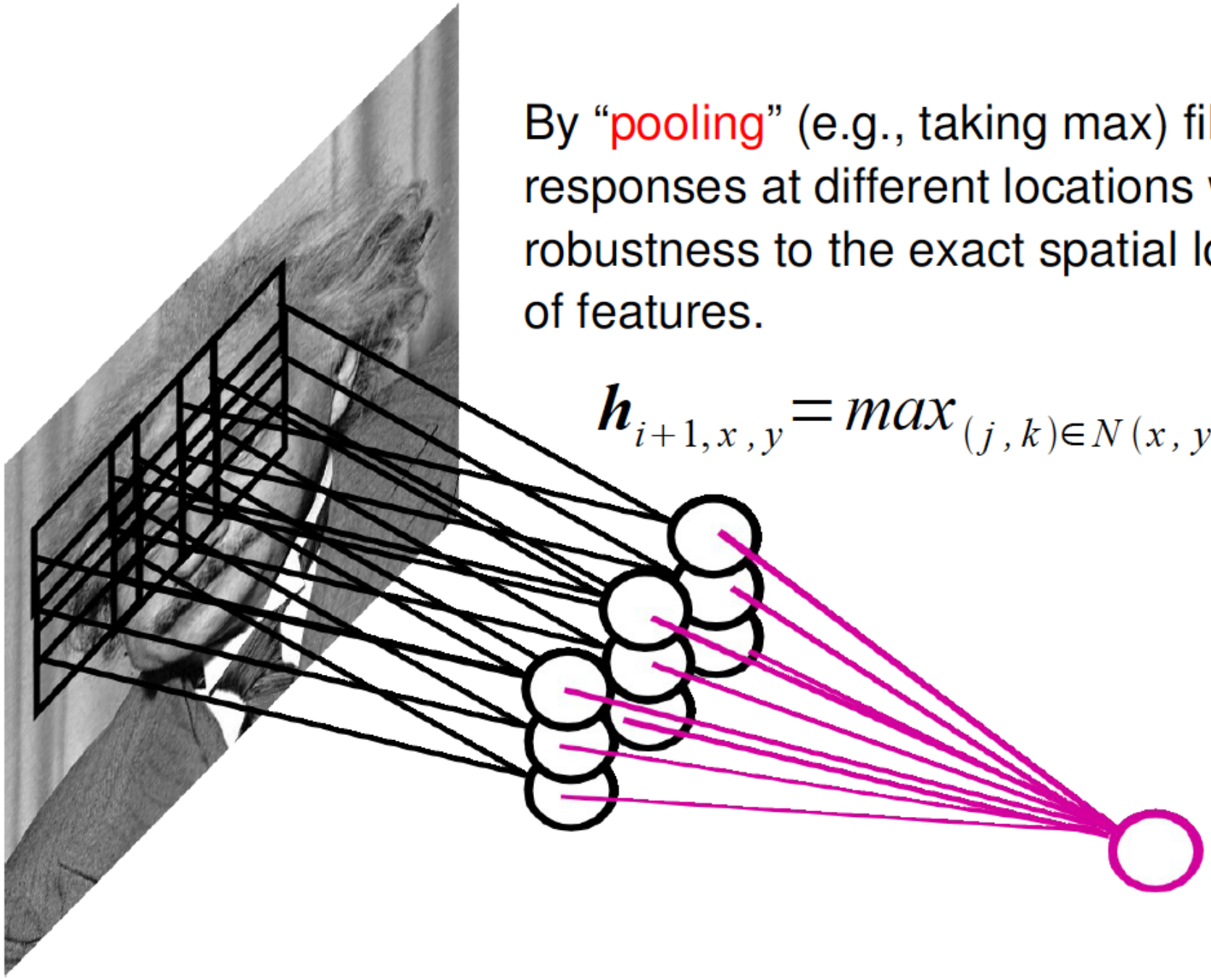
Jarret et al. 2009

$$y_j = g_j \tanh(\sum_i k_{ij} * x_i)$$

- x_i is the i[th] channel of input
- k_{ij} is the convolution kernel
- g_j is a learned scaling factor
- g_j is the hidden layer

can add bias

POOLING



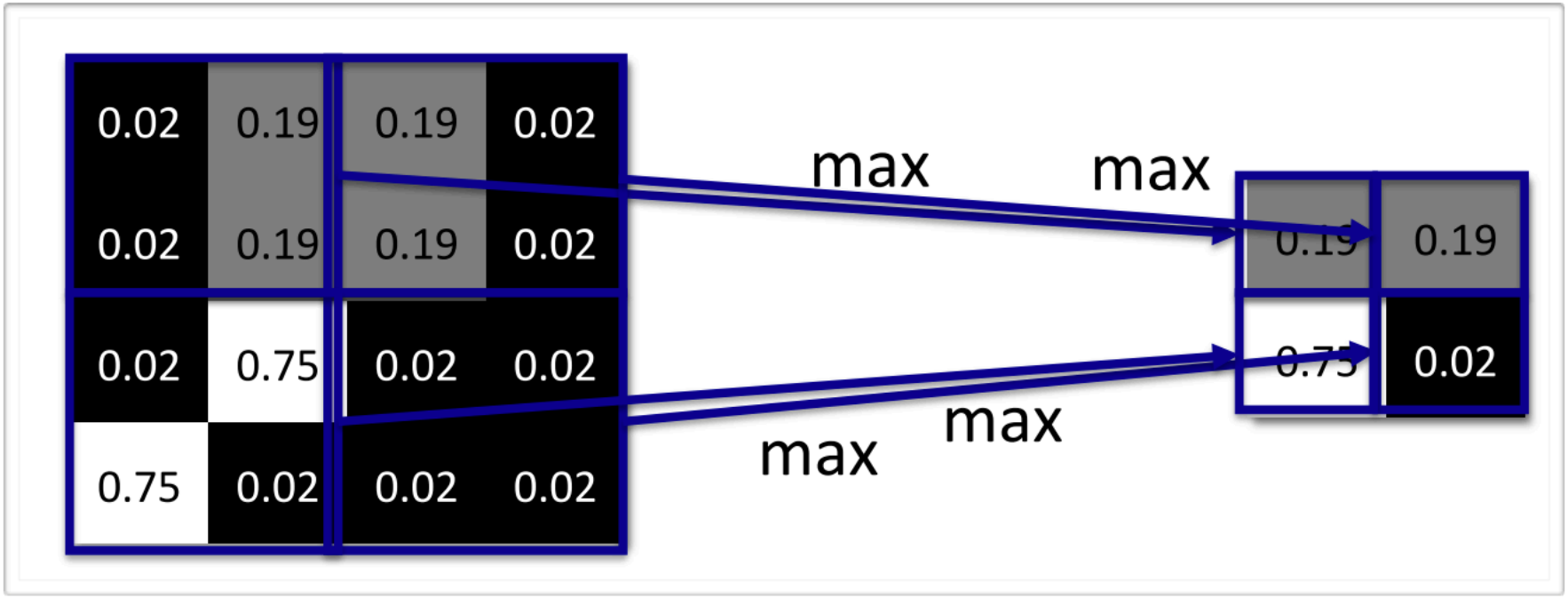
By "pooling" (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.

$$\boldsymbol{h}_{i+1,x,y} = max_{(j,k)\in N(x,y)} \boldsymbol{h}_{i,j,k}$$

Ranzato

Example: Pooling

- Illustration of pooling/subsampling operation

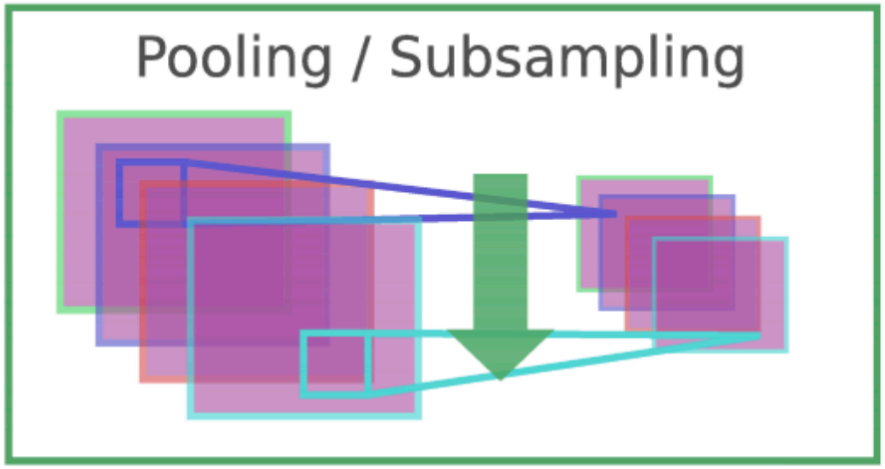


- Why pooling?
  - Introduces invariance to local translations
  - Reduces the number of hidden units in hidden layer

Pooling

- Pool hidden units in same neighborhood
  - an alternative to "max" pooling is "average" pooling

$$y_{ijk} = \max_{p,q} x_{i,j+p,k+q}$$



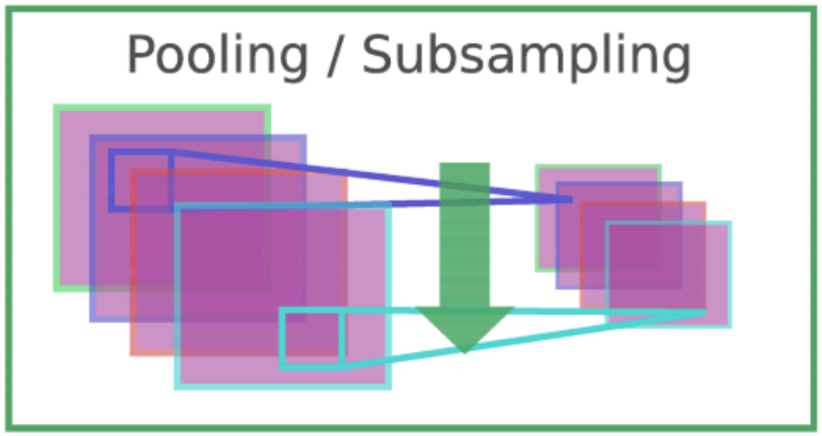
Jarret et al. 2009

- x_i is the i^{th} channel of input
- $x_{i,j,k}$ is value of the i^{th} feature map at position j,k
- p is vertical index in local neighborhood
- q is horizontal index in local neighborhood
- y_{ijk} is pooled / subsampled layer
- m is the neighborhood height/width

Pooling

- Pool hidden units in same neighborhood
  - an alternative to "max" pooling is "average" pooling

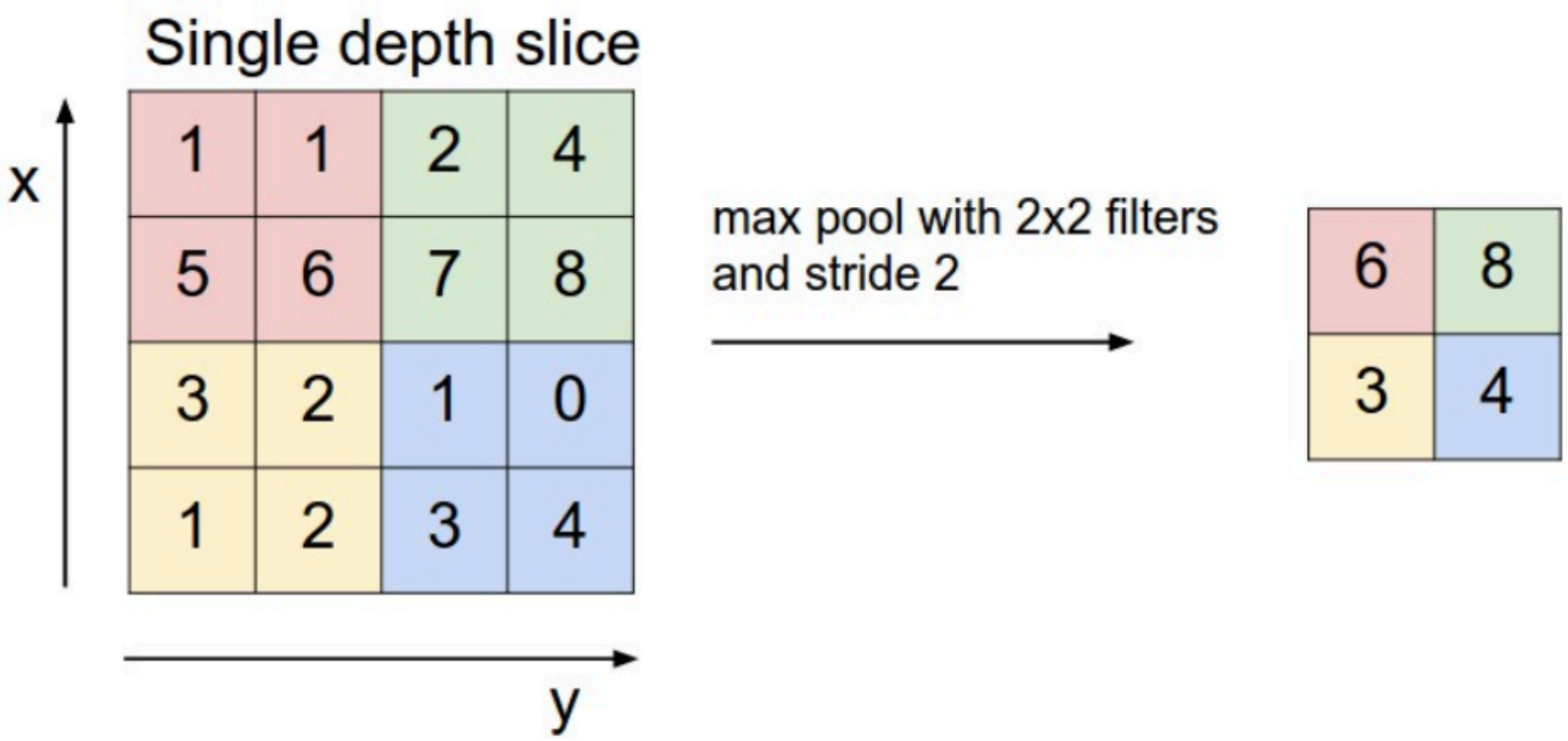
$$y_{ijk} = \frac{1}{m^2} \sum_{p,q} x_{i,j+p,k+q}$$



Jarret et al. 2009

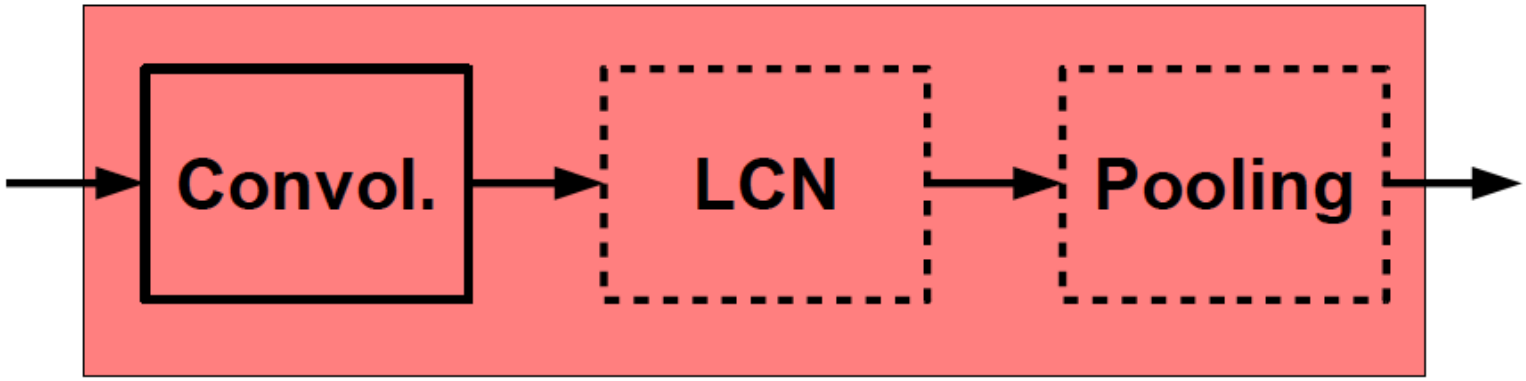
- x_i is the i^{th} channel of input
- $x_{i,j,k}$ is value of the i^{th} feature map at position j,k
- p is vertical index in local neighborhood
- q is horizontal index in local neighborhood
- y_{ijk} is pooled / subsampled layer
- m is the neighborhood height/width

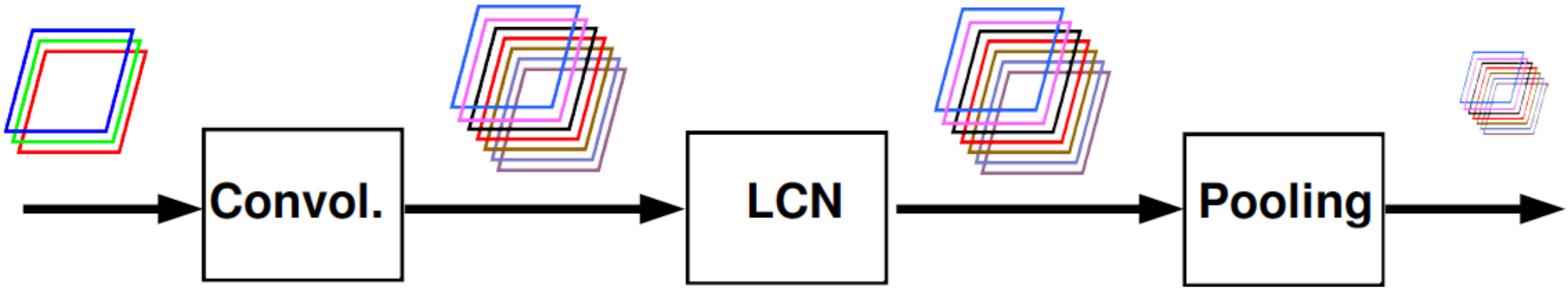
Example: Pooling



CONV NETS: TYPICAL ARCHITECTURE

One stage (zoom)



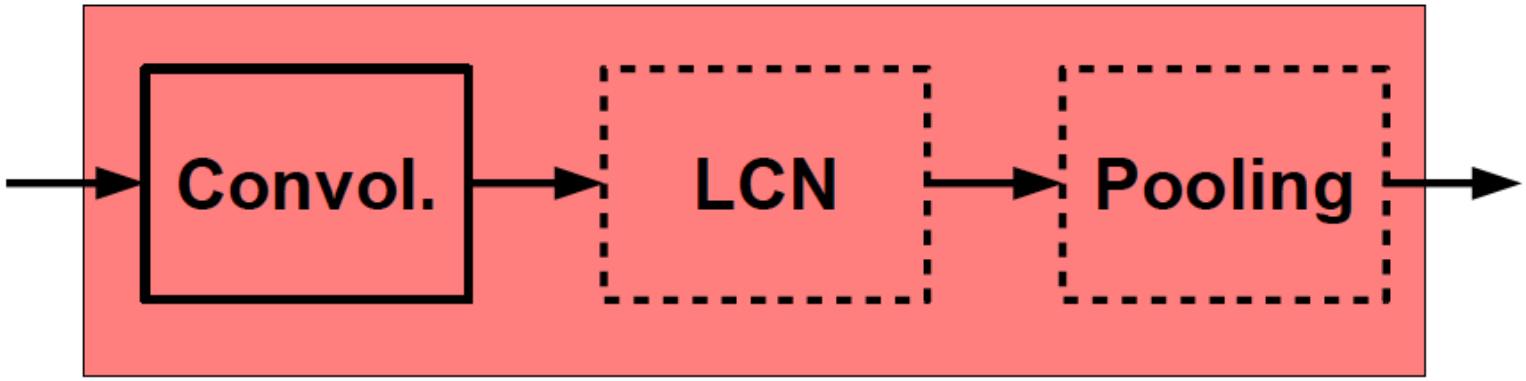


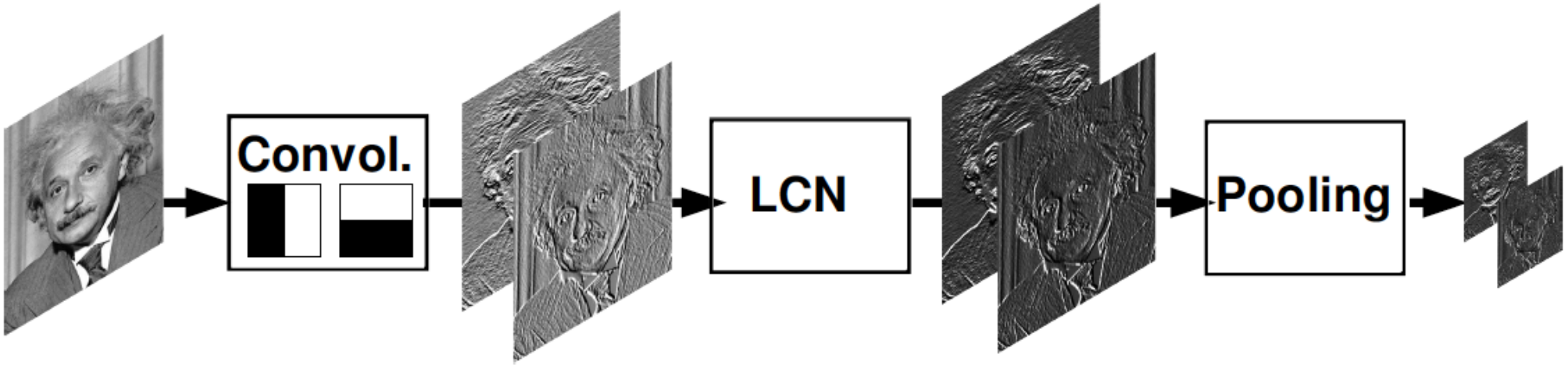
Convolutional layer increases nr. feature maps.
Pooling layer decreases spatial resolution.

Ranzato

CONV NETS: TYPICAL ARCHITECTURE

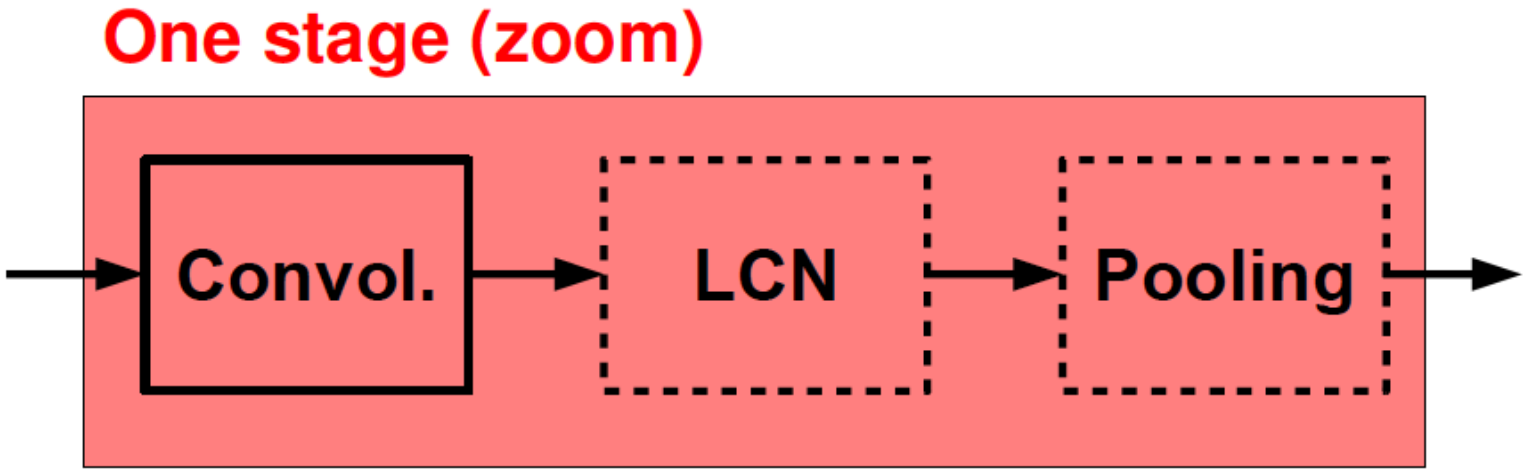
One stage (zoom)





Example with only two filters.

CONV NETS: TYPICAL ARCHITECTURE



Conceptually similar to:

SIFT → K-Means → Pyramid Pooling → SVM

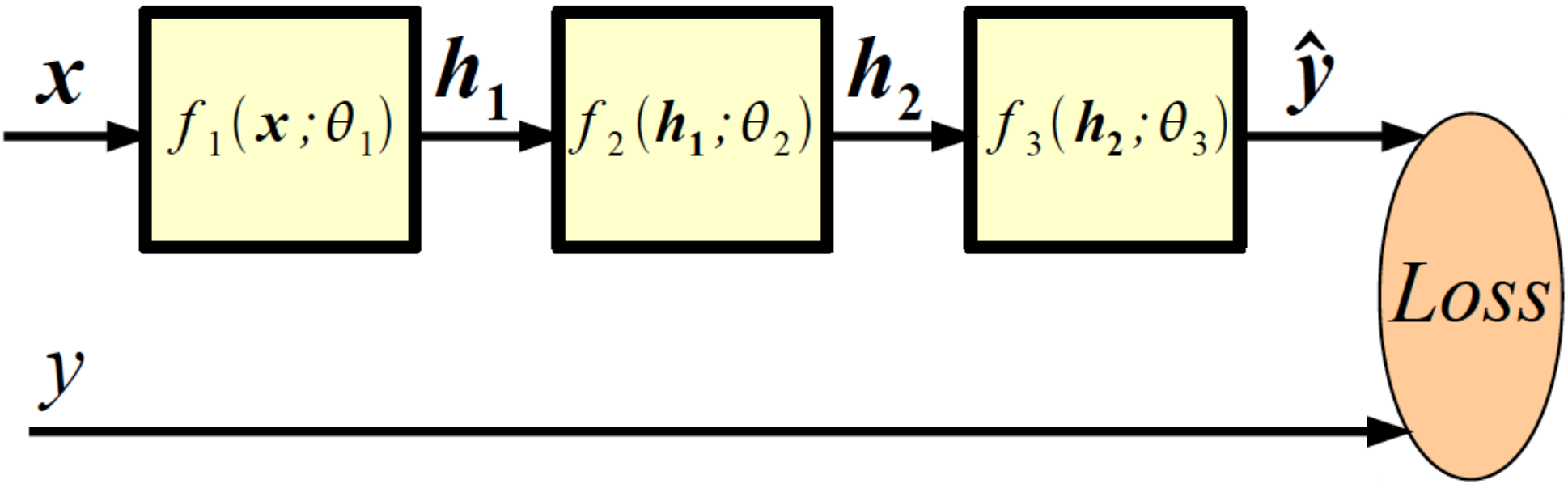
Lazebnik et al. "...Spatial Pyramid Matching..." CVPR 2006

SIFT → Fisher Vect. → Pooling → SVM

Sanchez et al. "Image classifcation with F.V.: Theory and practice" IJCV 2012

How to Train Neural Networks?

Loss

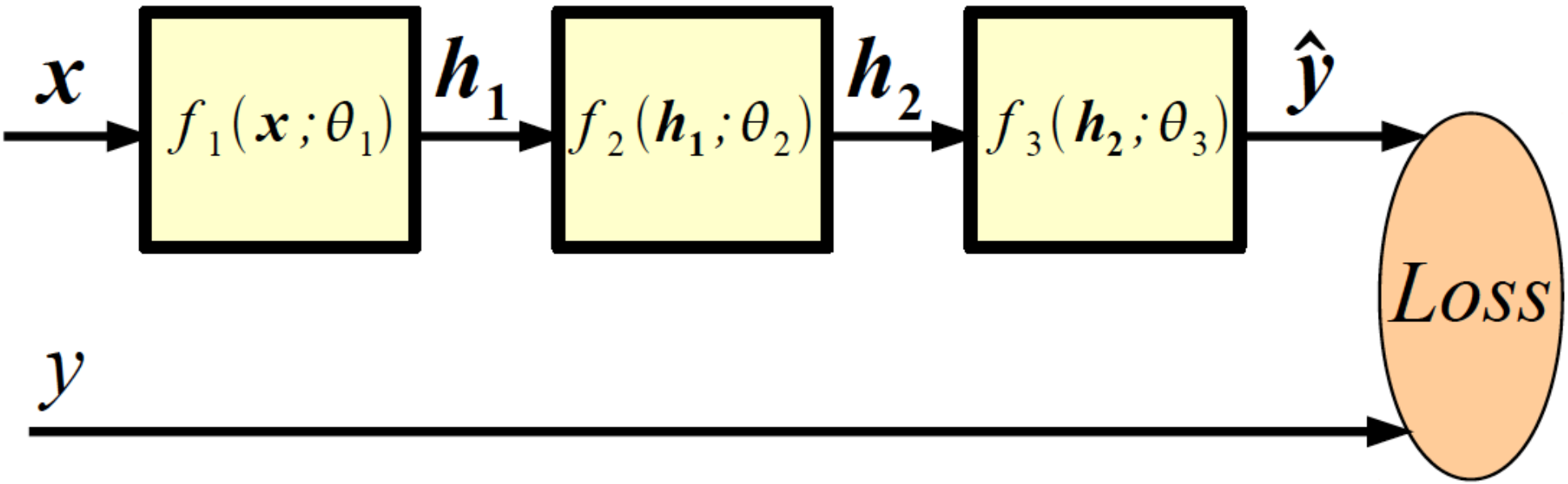


The measure of how well the model fits the training set is given by a suitable loss function: $L(\boldsymbol{x}, y; \boldsymbol{\theta})$

The loss depends on the input $\boldsymbol{x}$, the target label y, and the parameters $\boldsymbol{\theta}$.

[Slide credit: Ranzato]

Loss



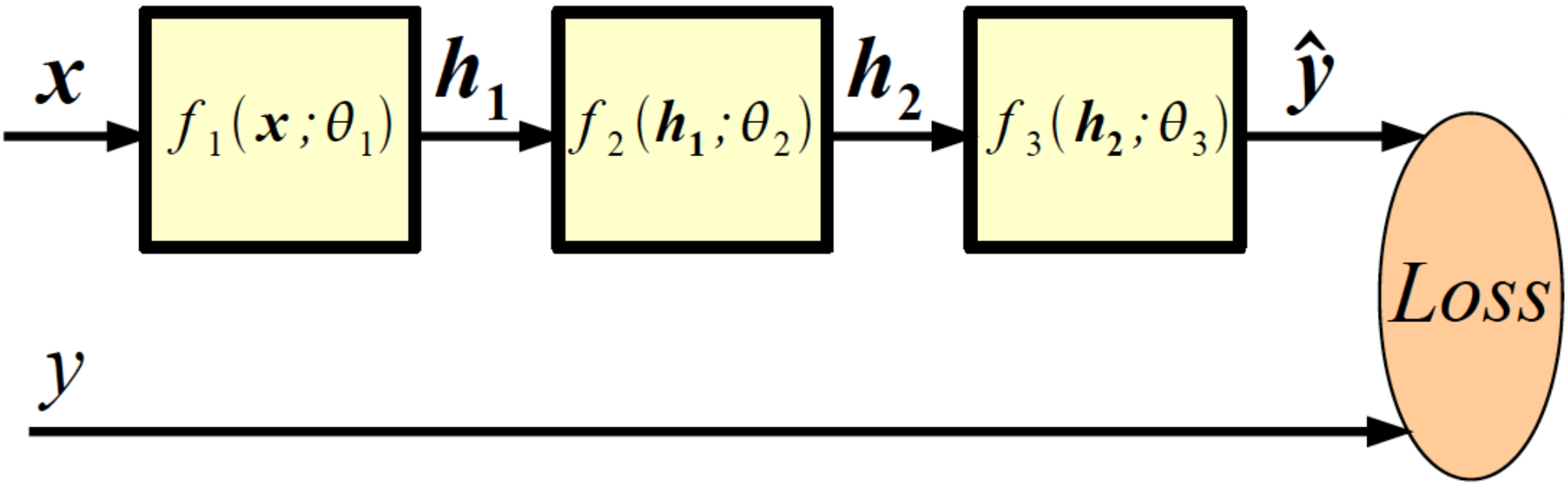
The measure of how well the model fits the training set is given by a suitable loss function: $L(\boldsymbol{x}, y; \boldsymbol{\theta})$

For instance,

$$L(\boldsymbol{x}, y=k; \boldsymbol{\theta}) = -\log(p(class=k|\boldsymbol{x}))$$

[Slide credit: Ranzato]

Loss



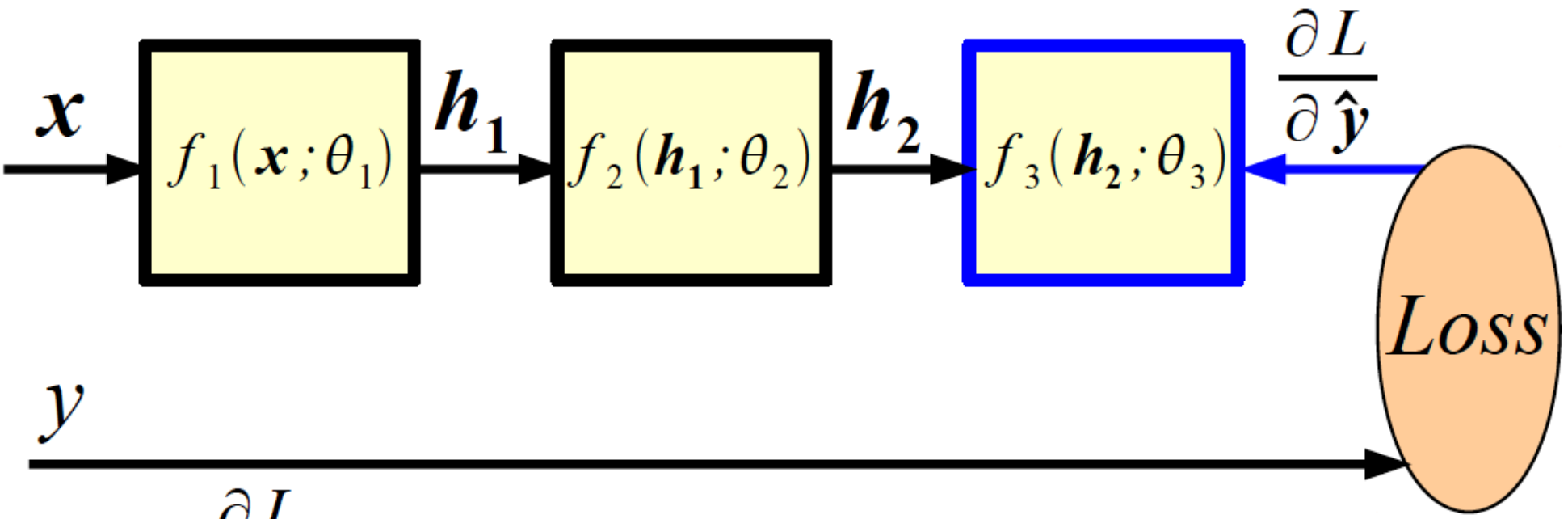
Q.: how to tune the parameters to decrease the loss?

If loss is (a.e.) differentiable we can compute gradients.

We can use chain-rule, a.k.a. **back-propagation**, to compute the gradients w.r.t. parameters at the lower layers.

Rumelhart et al. "Learning internal representations by back-propagating.." Nature 1986

Backward Propagation (BPROP)

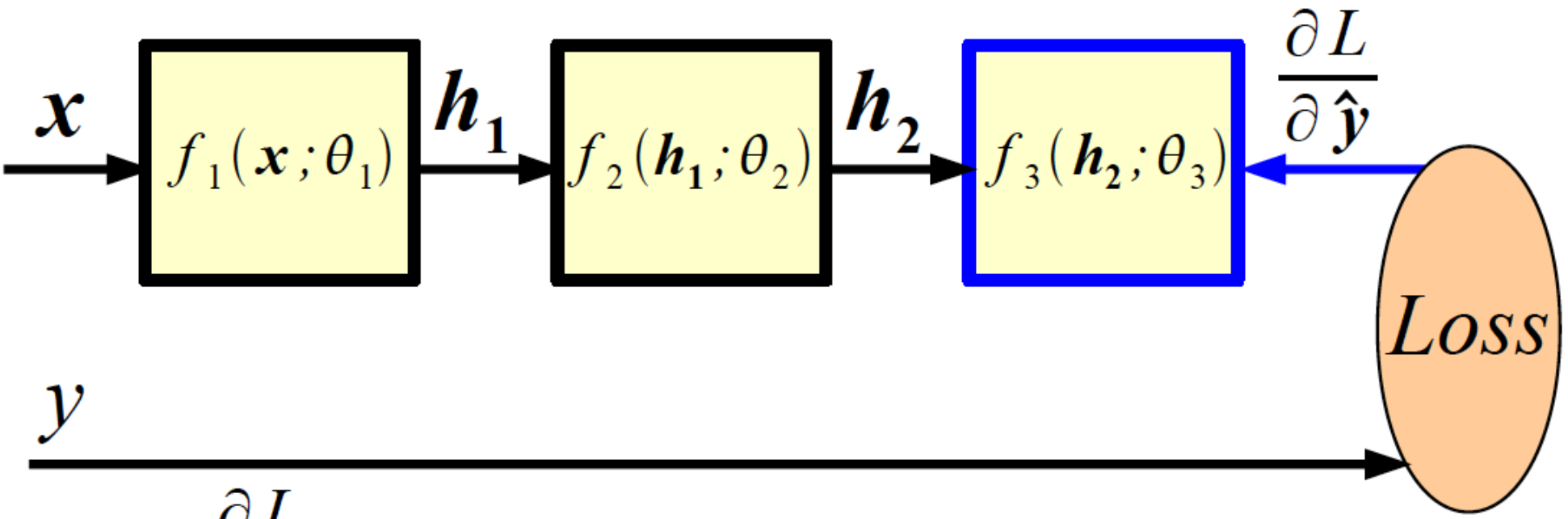


Given $\frac{\partial L}{\partial \hat{\mathbf{y}}}$ and assumiing the Jacobian of each module is easy to compute, then we have:

$$\frac{\partial L}{\partial \theta_3} = \frac{\partial L}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \theta_3} \qquad \frac{\partial L}{\partial \mathbf{h}_2} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{h}_2}$$

[Slide credit: Ranzato]

Backward Propagation (BPROP)

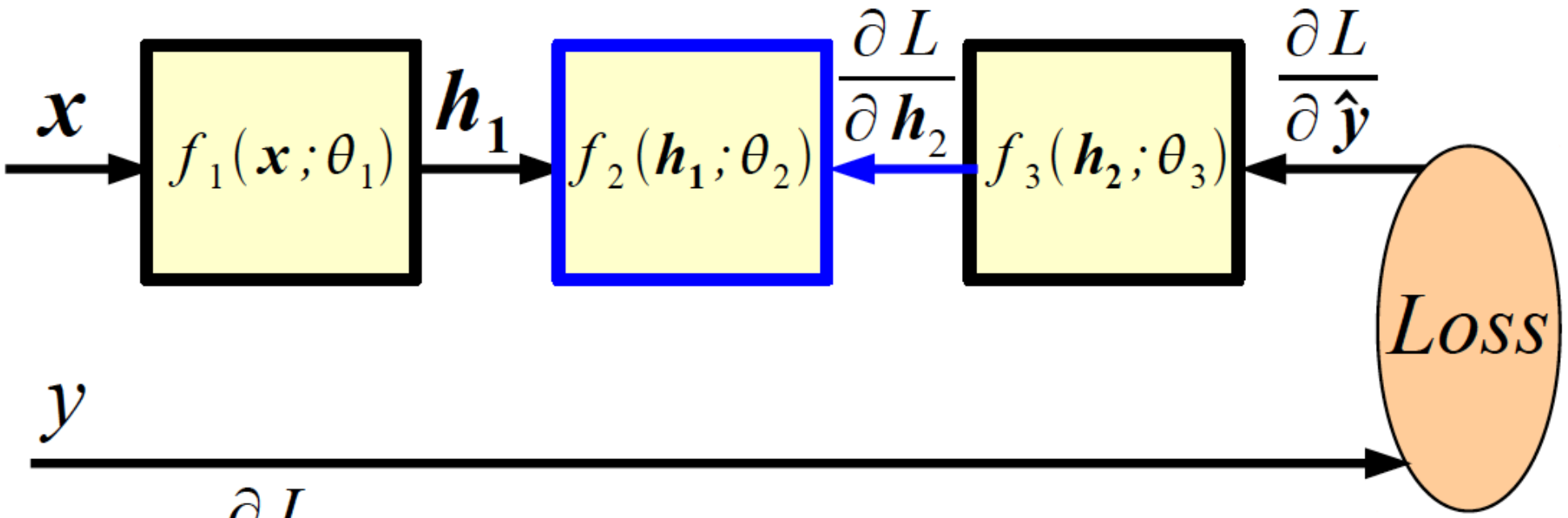


Given $\frac{\partial L}{\partial \hat{y}}$ and assumiing the Jacobian of each module is easy to compute, then we have:

$$\frac{\partial L}{\partial \theta_3} = (\hat{y} - y)\ h_2' \qquad \frac{\partial L}{\partial h_2} = (\hat{y} - y)\ \theta_3'$$

[Slide credit: Ranzato]

Backward Propagation (BPROP)

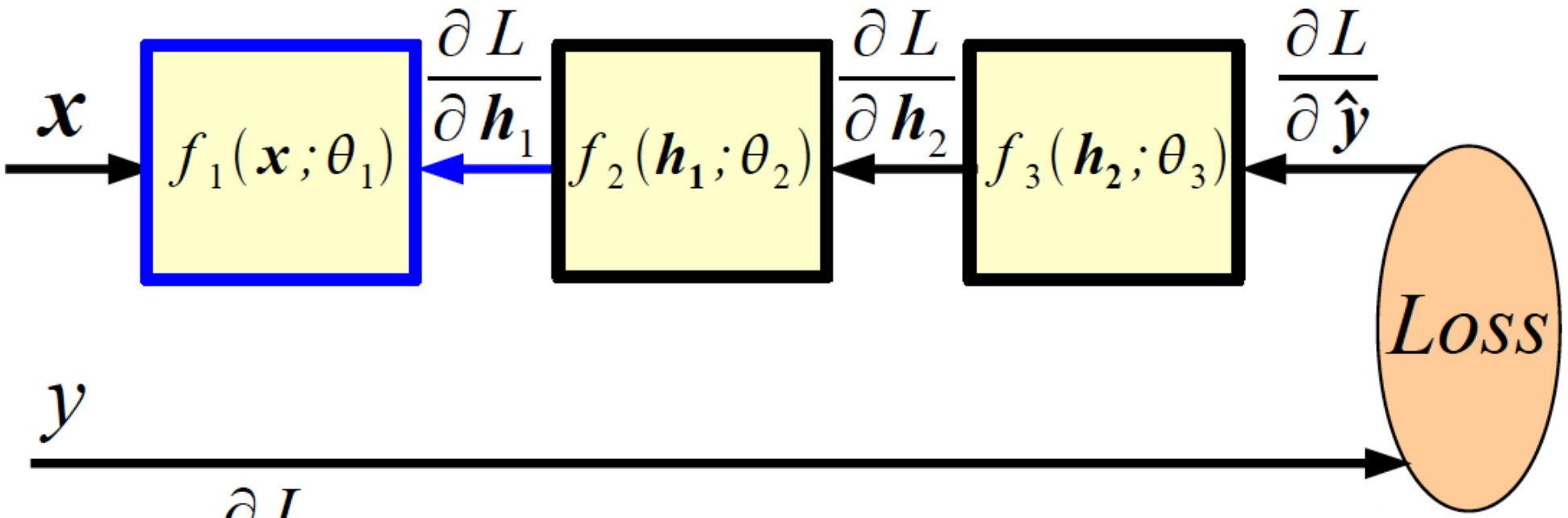


Given $\frac{\partial L}{\partial h_2}$ we can compute now:

$$\frac{\partial L}{\partial \theta_2} = \frac{\partial L}{\partial h_2}\frac{\partial h_2}{\partial \theta_2} \qquad \frac{\partial L}{\partial h_1} = \frac{\partial L}{\partial h_2}\frac{\partial h_2}{\partial h_1}$$

[Slide credit: Ranzato]

Backward Propagation (BPROP)



Given $\frac{\partial L}{\partial h_1}$ we can compute now:

$$\frac{\partial L}{\partial \theta_1} = \frac{\partial L}{\partial h_1} \frac{\partial h_1}{\partial \theta_1}$$

[Slide credit: Ranzato]

Learning Rules

- Once we have $\frac{\partial L}{\partial \theta}$ we can update θ

- Stochastic gradient descent (SGD):

$$\theta \leftarrow \theta - \eta \frac{\partial L}{\partial \theta}, \eta \in R$$

- SGD with momentum

$$\theta \leftarrow \theta - \eta \Delta$$

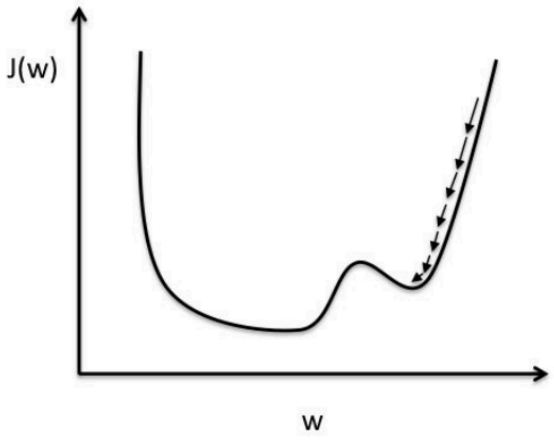
$$\Delta \leftarrow 0.9 \Delta + \frac{\partial L}{\partial \theta}$$

Learning Rate

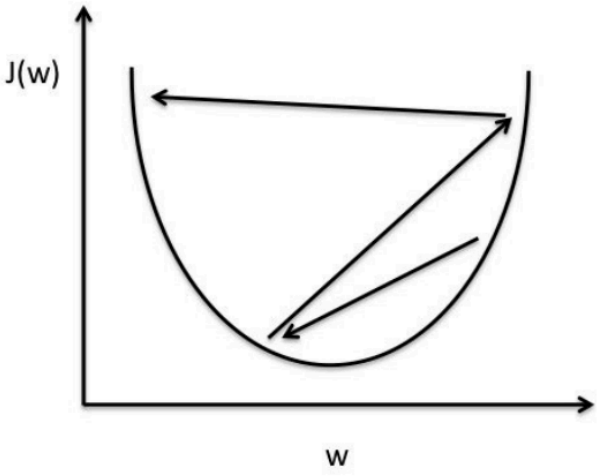
How to Choose Learning Rate?

Update Rule:

$$\theta := \theta - \eta \frac{\partial J(\theta)}{\partial \theta}$$



Small learning rate: Many iterations until convergence and trapping in local minima.

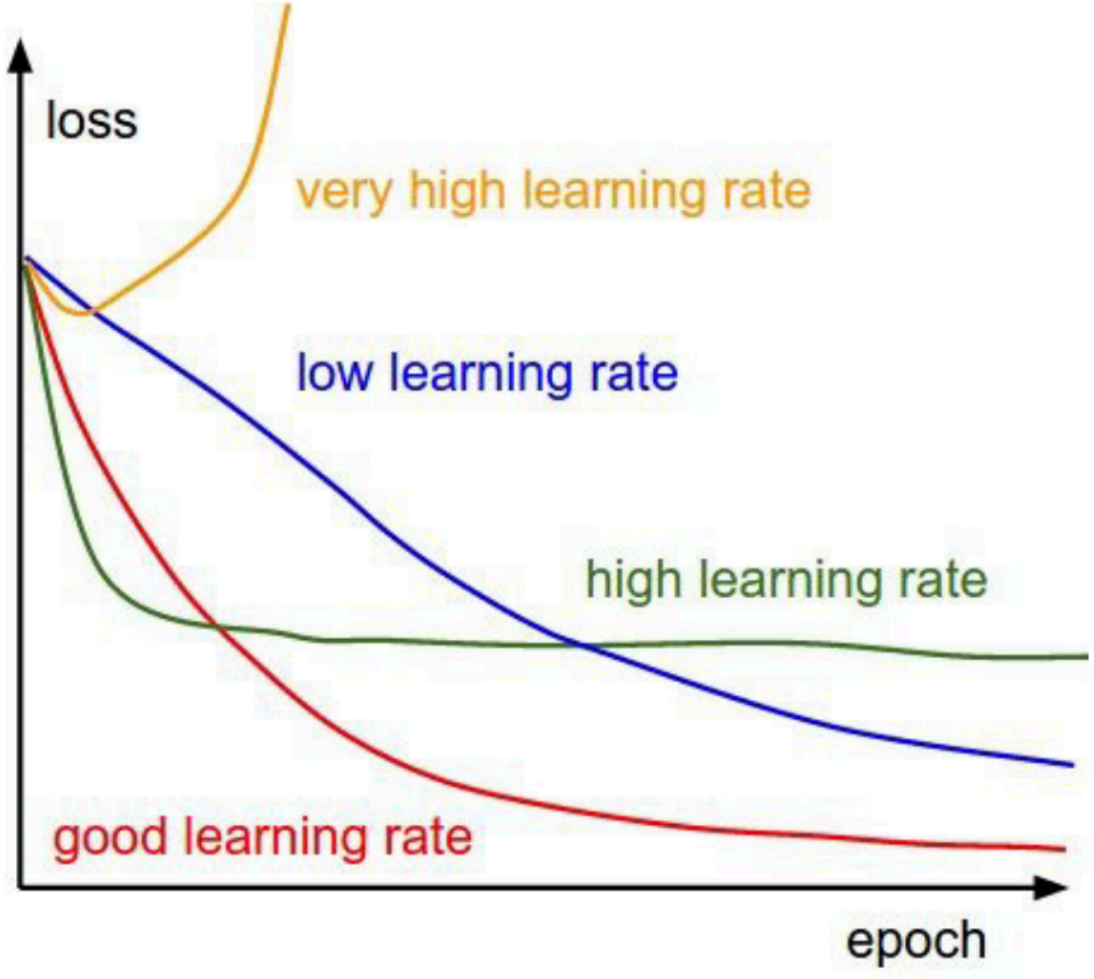


Large learning rate: Overshooting.

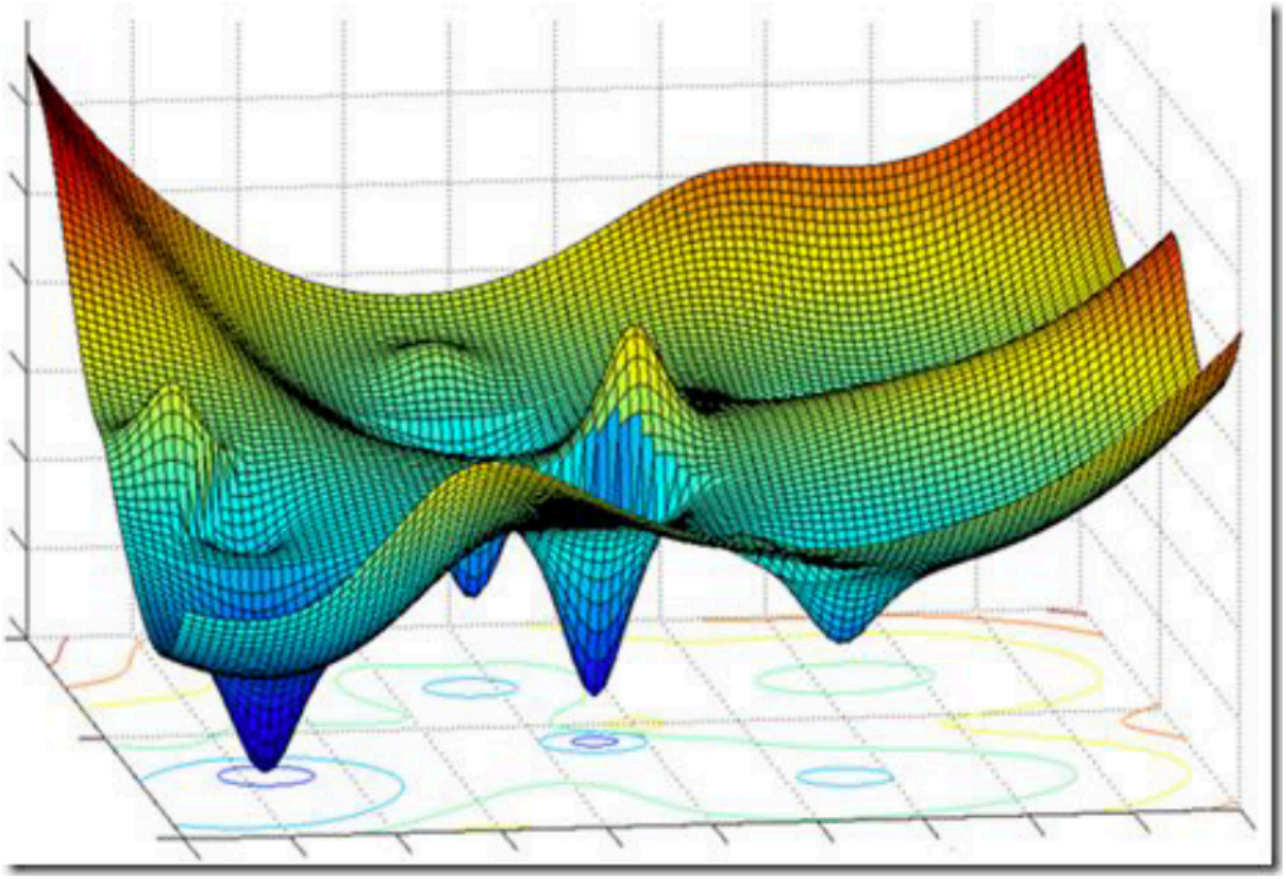
Small learning rate

Large learning rate

Learning Rate

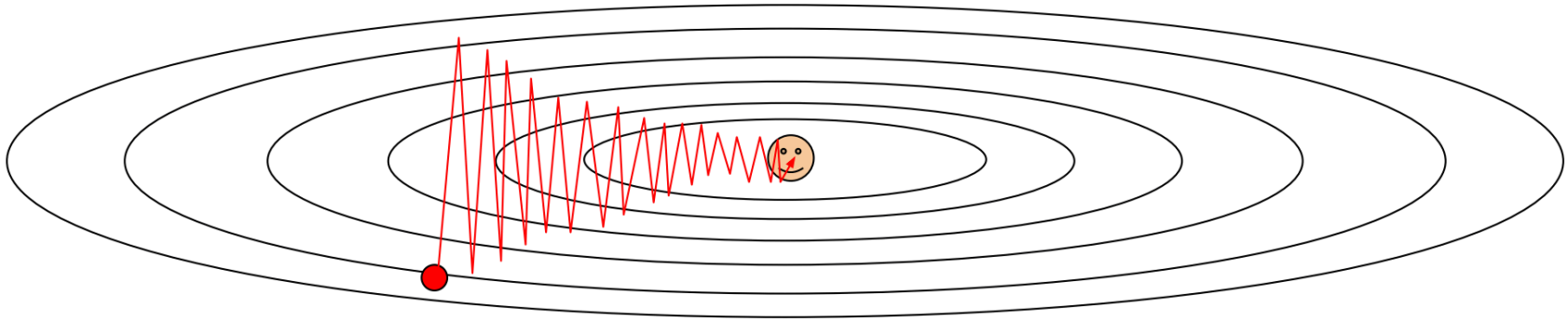


Loss Function can be difficult to optimize



Optimization: Problems with SGD

What if loss changes quickly in one direction and slowly in another?
What does gradient descent do?
Very slow progress along shallow dimension, jitter along steep direction



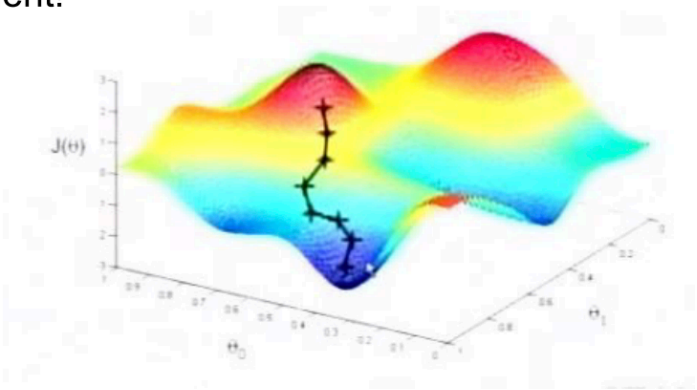
Loss function has high **condition number**: ratio of largest to smallest singular value of the Hessian matrix is large

Why is it **stochastic** gradient descent?

- Initialize θ randomly
- For N Epochs
  - For each training example (x, y):
    - Compute Loss Gradient: $\frac{\partial J(\theta)}{\partial \theta}$
    - Update θ with update rule:

$$\theta := \theta - \eta \frac{\partial J(\theta)}{\partial \theta}$$

Only an estimate of true gradient!

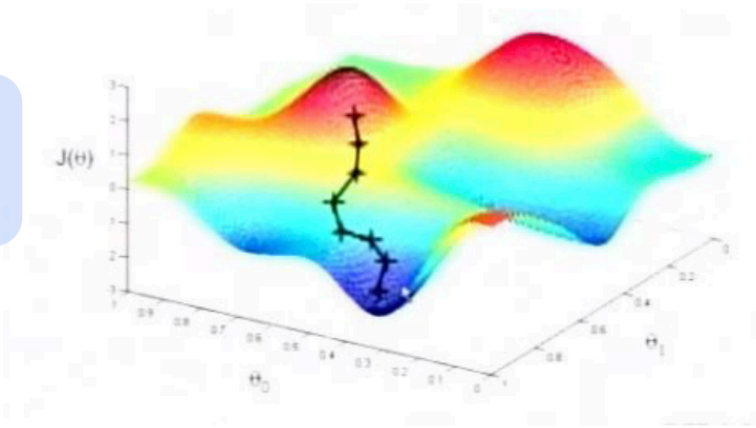


Minibatches reduces gradient variation

- Initialize θ randomly
- For N Epochs
  - For each training **batch {(x0, y0), … , (x_B, y_B)}:**
    - Compute Loss Gradient: $\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{B}\sum_{i}^{B}\frac{\partial J_i(\theta)}{\partial \theta}$
    - Update θ with update rule:

$$\theta := \theta - \eta\frac{\partial J(\theta)}{\partial \theta}$$

More accurate estimate!



- More accurate estimation of gradient
  - Smoother convergence
  - Allows for larger learning rates
- Minibatches lead to fast training!
  - Can parallelize computation + achieve significant speed increases on GPU's

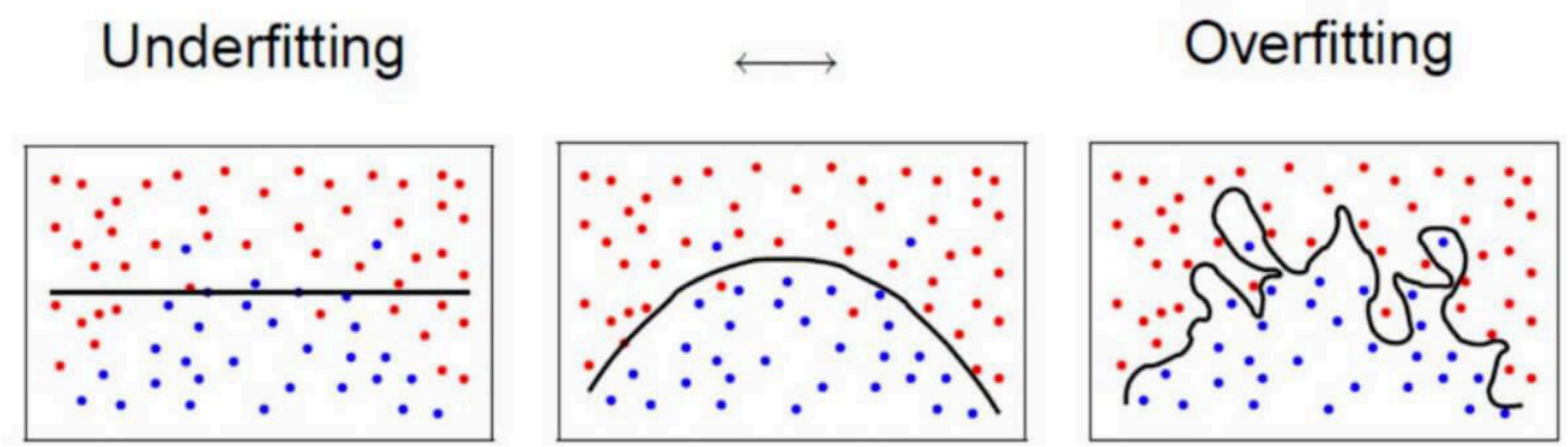
Other Optimization Algorithms

Adaptive Learning Rate Algorithms

- ADAM
- Momentum
- NAG
- Adagrad
- Adadelta
- RMSProp

For details: check out http://sebastianruder.com/optimizing-gradient-descent/

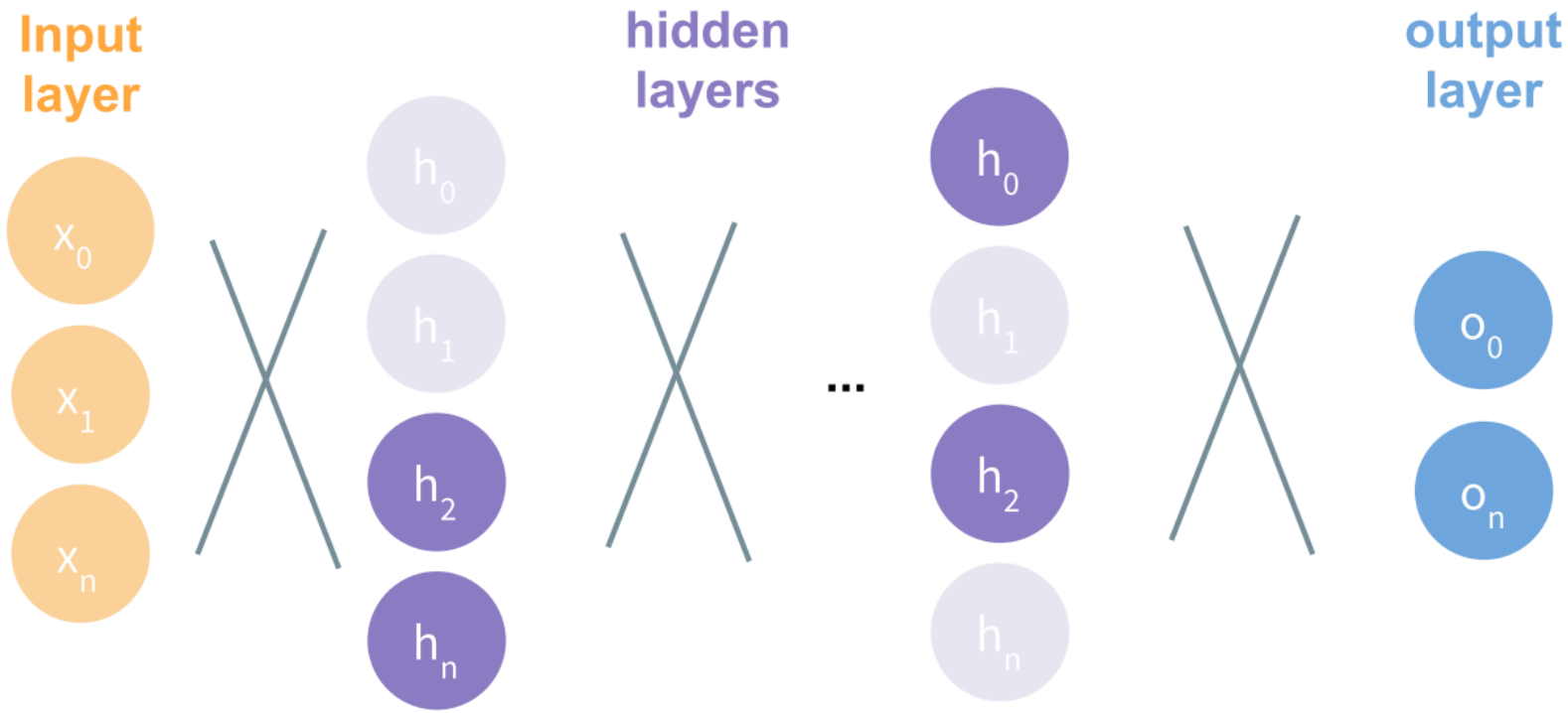
Fighting Overfitting

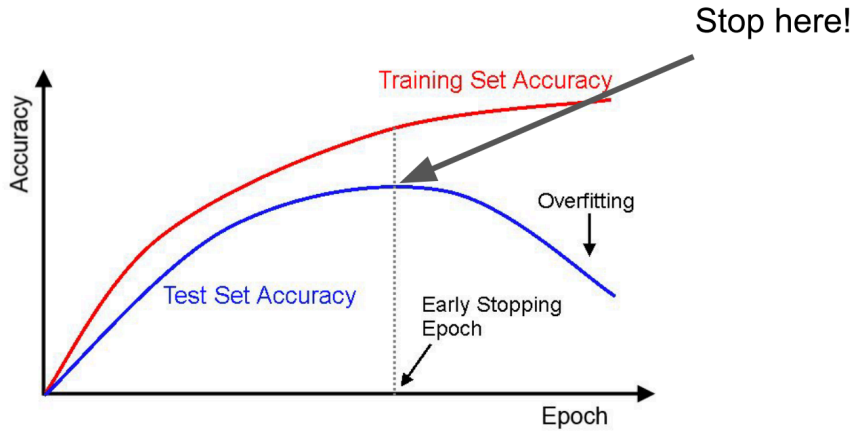


1. Dropout
2. Early Stopping
3. Weight Regularization
4. ...many more

1. Dropout

- During training, randomly set some activations to 0
  - Typically ‘drop’ 50% of activations in layer
  - Forces network to not rely on any 1 node





- Don't give the network time to overfit
- ...
- **Epoch 15:** Train: 85% Validation: 80%
- **Epoch 16:** Train: 87% Validation: 82%
- **Epoch 17: Train: 90% Validation: 85%**
- **Epoch 18:** Train: 95% Validation: 83%
- **Epoch 19:** Train: 97% Validation: 78%
- **Epoch 20:** Train: 98% Validation: 75%

Stop here!

2. Early Stopping

3. Weight Regularization

- Large weights typically mean model is overfitting
- Add the size of the weights to our loss function
- Perform well on task + keep weights small

$$\arg_\theta \min \frac{1}{T} \sum_t \text{loss}(f(x^{(t)}; \theta), y^{(t)})) \; + \lambda \sum_i (\theta_i)^2$$

$$J(\theta)$$

DL Software

Caffe
(UC Berkeley)

Caffe2
(Facebook)

Torch
(NYU / Facebook)

PyTorch
(Facebook)

Theano
(U Montreal)

TensorFlow
(Google)

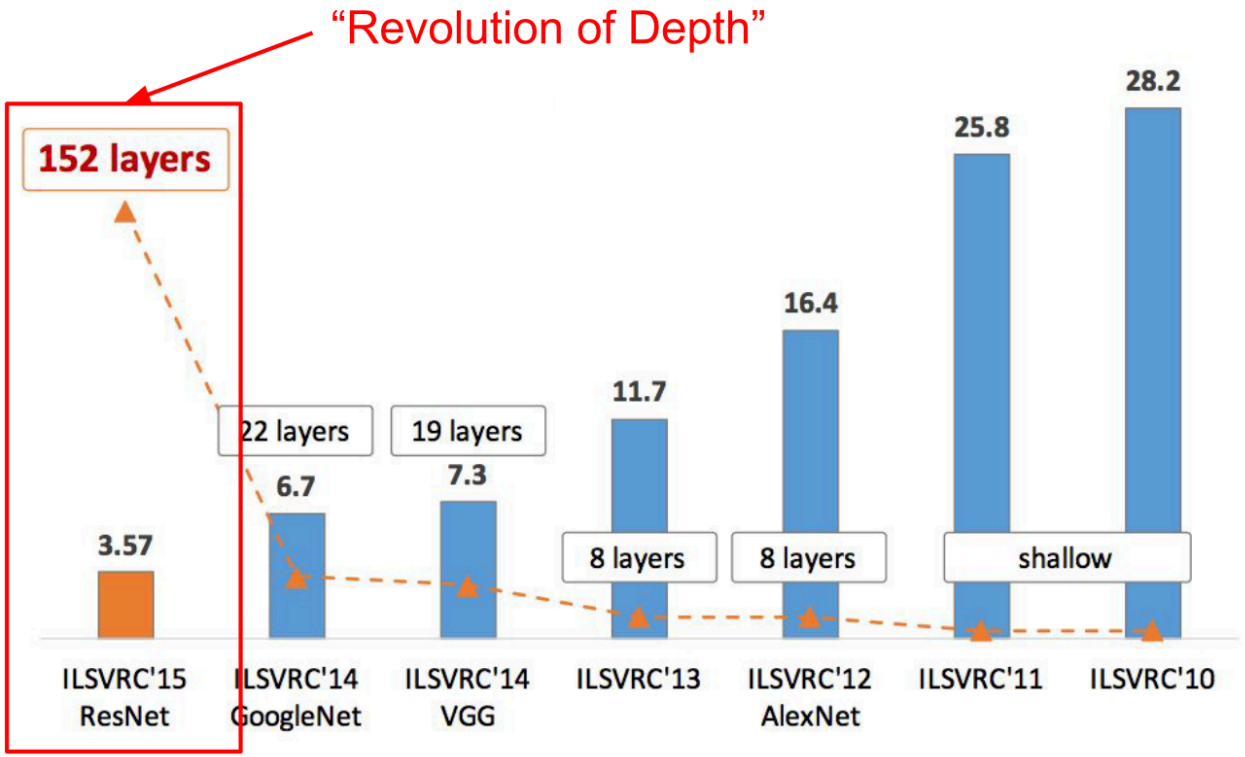
CNTK
(Microsoft)

MXNet
(Amazon)
Developed by U Washington, CMU, MIT, Hong Kong U, etc but main framework of choice at AWS

And others...

DL Architectures

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Slide Credits and References

- Stanford CS class CS231n: Convolutional Neural Networks for Visual Recognition for the image.
- MIT 6.S191-Intro. to Deep Learning Lec Notes
- 2007 NIPS Tutorial on Deep Belief Nets, G. Hinton.
- Ranzato, LeCun, Dean, Krishevsky, Sutskever.
- Rahul Sukthankar (Google Research).
- A.Ng, ECCV 2010 Tutorial.
- http://deeplearning.net/tutorial/deeplearning.pdf
- http://cs.nyu.edu/~fergus/tutorials/deep_learning_cvpr12/talk_CVPR.pdf
- Graham Taylor (CVPR 2012 tutorial)
- Y. LeCun