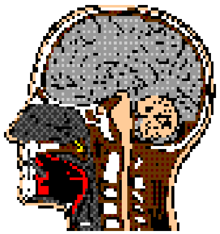
An Introduction to Neural Nets & Deep Learning

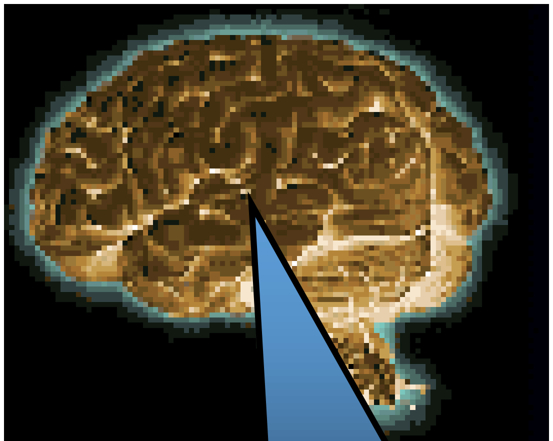
Slides by Rajesh Rao, Hung-yi Lee,
Ismini Lourentzou, Noah Smith

The human brain is extremely good at classifying images

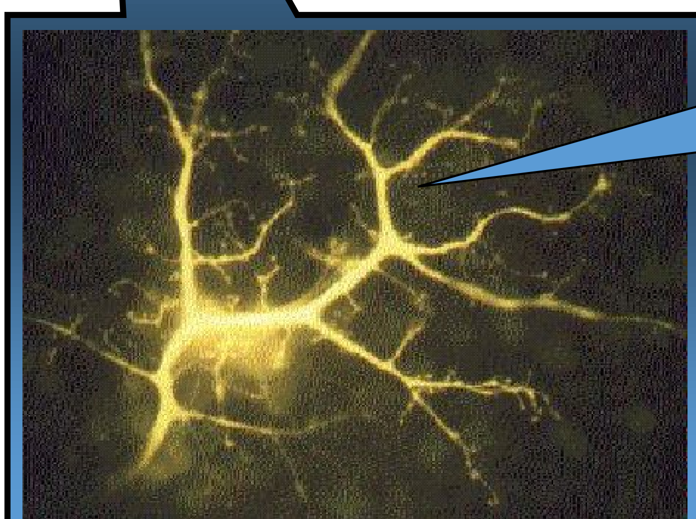
Can we develop classification methods by emulating the brain?

Brain Computer: What is it?





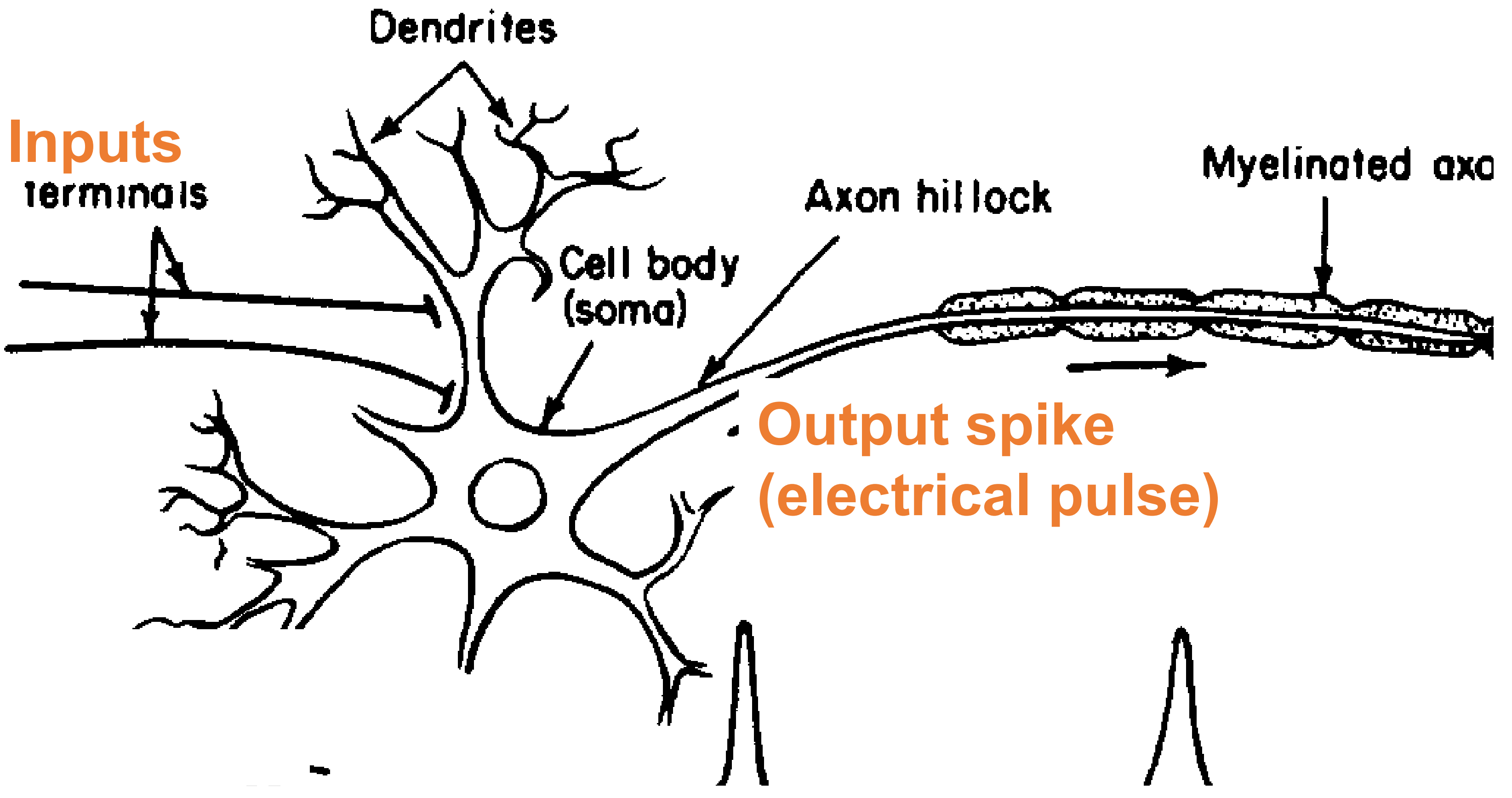
Human brain contains a massively interconnected net of 10^{10}-10^{11} (10 billion) neurons (cortical cells)



Biological Neuron

- The simple "arithmetic computing" element

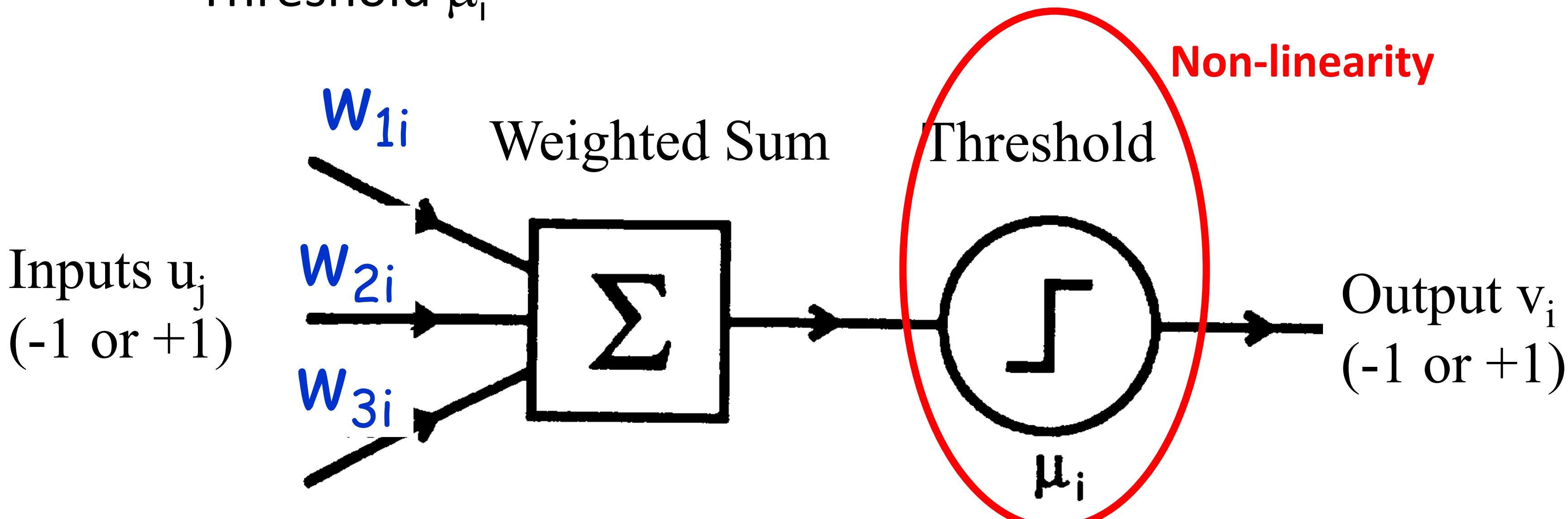
Neurons communicate via spikes



Output spike roughly dependent on whether sum of all inputs reaches a threshold

Neurons as “Threshold Units”

- Artificial neuron:
  - m binary inputs (-1 or 1), 1 output (-1 or 1)
  - Synaptic weights w_{ji}
  - Threshold μ_i



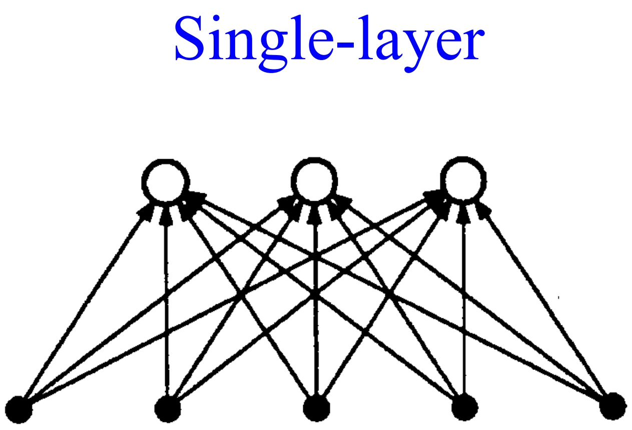
$$v_i = \Theta\left(\sum_j w_{ji} u_j - \mu_i\right)$$

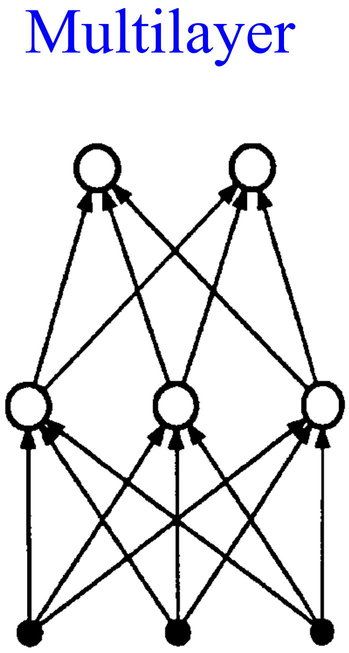
$\Theta(x) = 1$ if $x > 0$ and -1 if $x \leq 0$

“Perceptrons” for Classification

- Fancy name for a type of layered “feed-forward” networks (no loops)

- Uses artificial neurons (“units”) with binary inputs and outputs





Perceptrons and Classification

- Consider a single-layer perceptron
  - Weighted sum forms a *linear hyperplane*

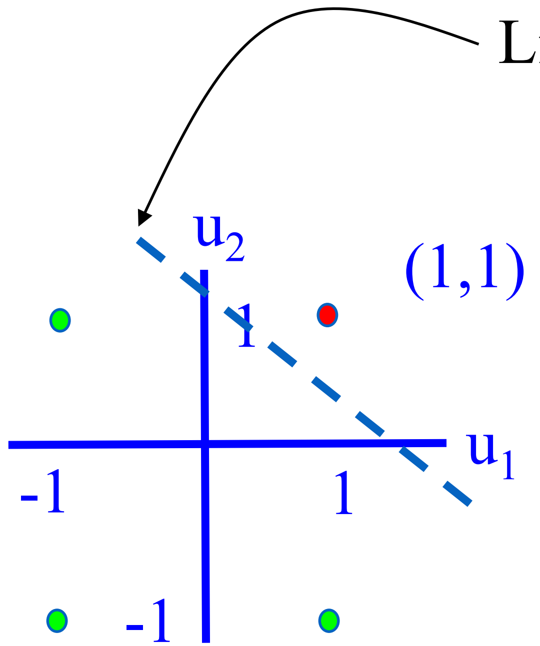
$$\sum_j w_{ji} u_j - \mu_i = 0$$

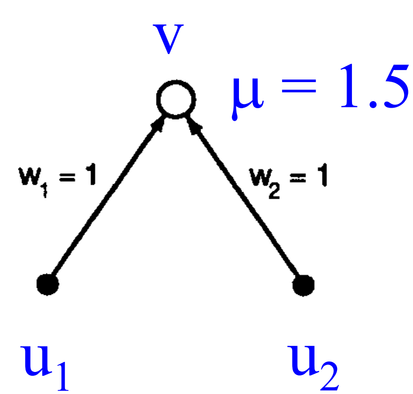
  - Everything *on one side* of this hyperplane is in class 1 (output = +1) and everything *on other side* is class 2 (output = -1)
- Any function that is linearly separable can be computed by a perceptron

Linear Separability

- Example: AND is linearly separable

u_1	u_2	AND
-1	-1	-1
1	-1	-1
-1	1	-1
1	1	1



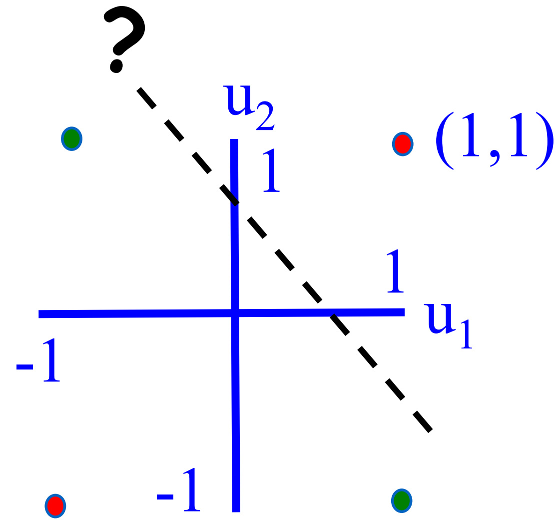


$v = 1$ iff $u_1 + u_2 - 1.5 > 0$

Similarly for OR and NOT

What about the XOR function?

u_1	u_2	XOR
-1	-1	1
1	-1	-1
-1	1	-1
1	1	1

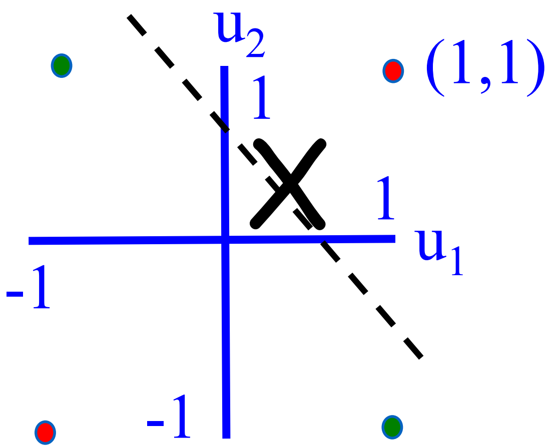


Can a perceptron separate the +1 outputs from the -1 outputs?

Linear Inseparability

- Perceptron with threshold units fails if classification task is not linearly separable
  - Example: XOR
  - No single line can separate the "yes" (+1) outputs from the "no" (−1) outputs!

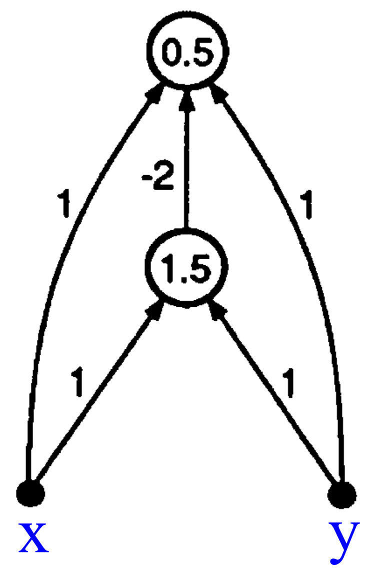
Minsky and Papert's book showing such negative results put a damper on neural networks research for over a decade!



How do we deal with linear inseparability?

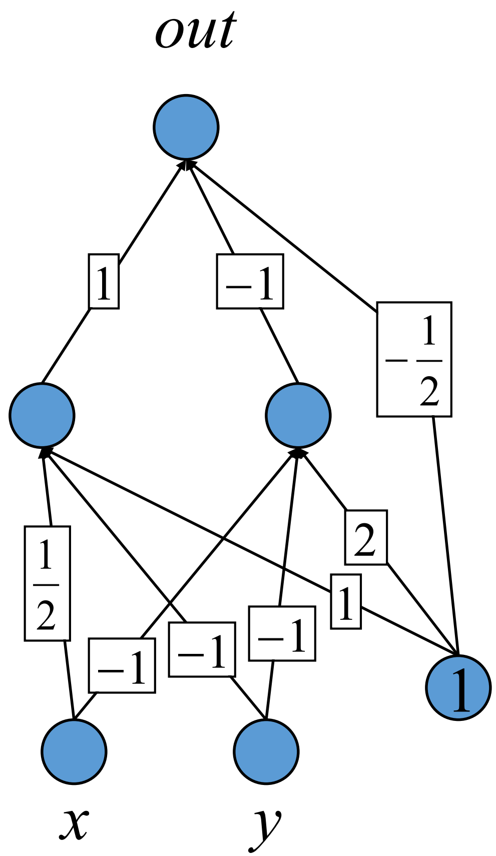
Idea 1: Multilayer Perceptrons

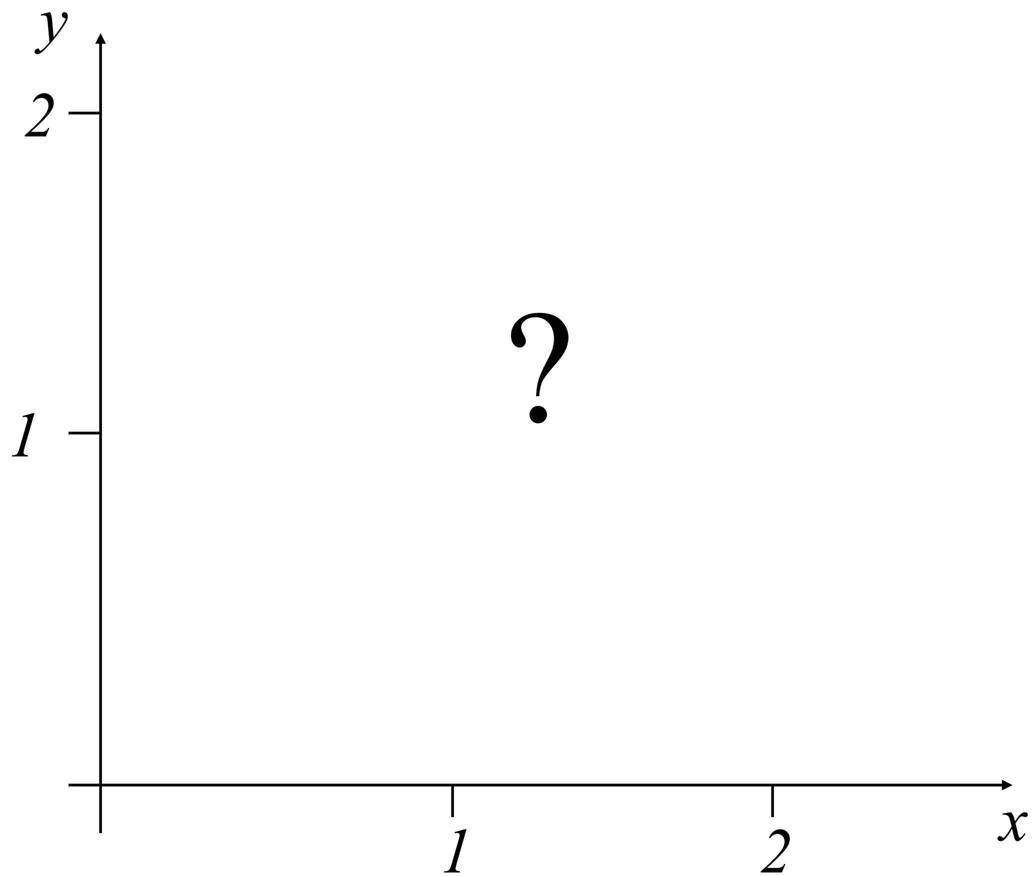
- Removes limitations of single-layer networks
  - Can solve XOR
- Example: Two-layer perceptron that computes XOR



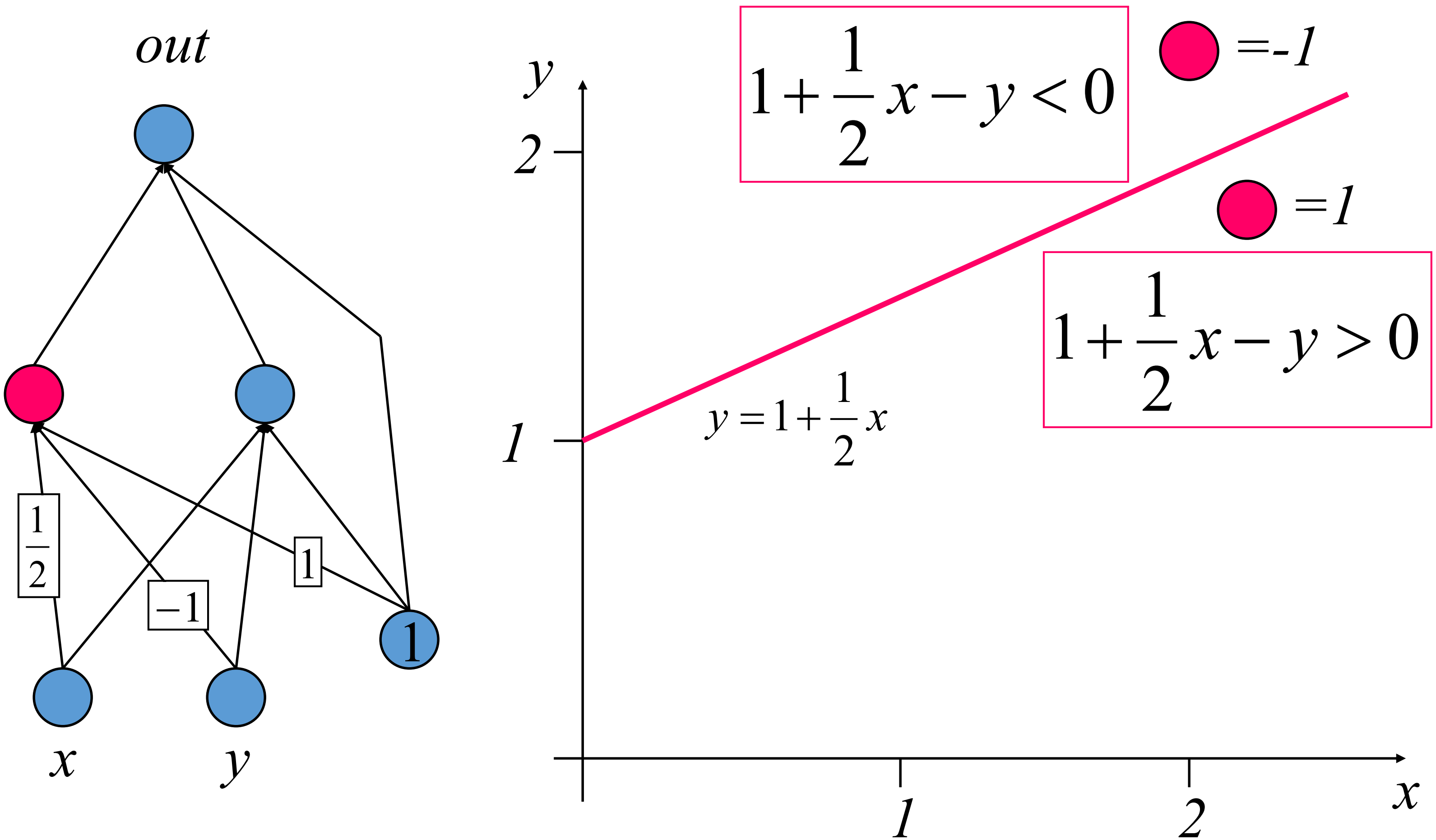
- Output is +1 if and only if $x + y - 2\Theta(x + y - 1.5) - 0.5 > 0$

Multilayer Perceptron: What does it do?

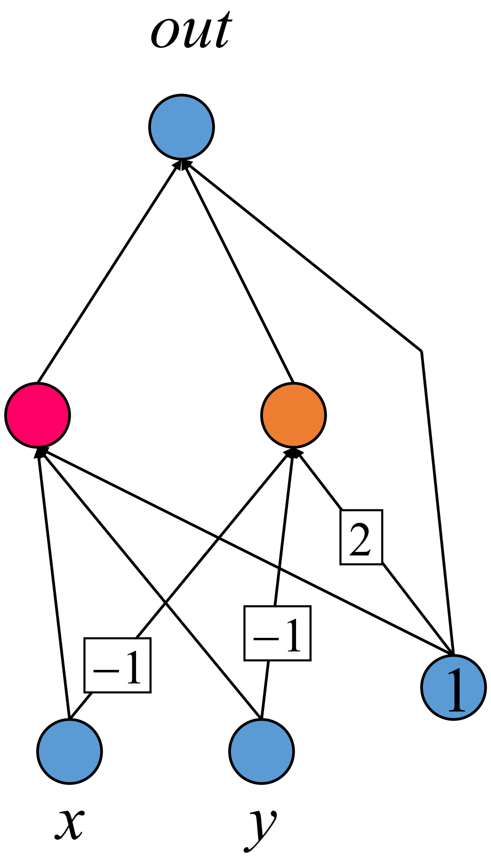


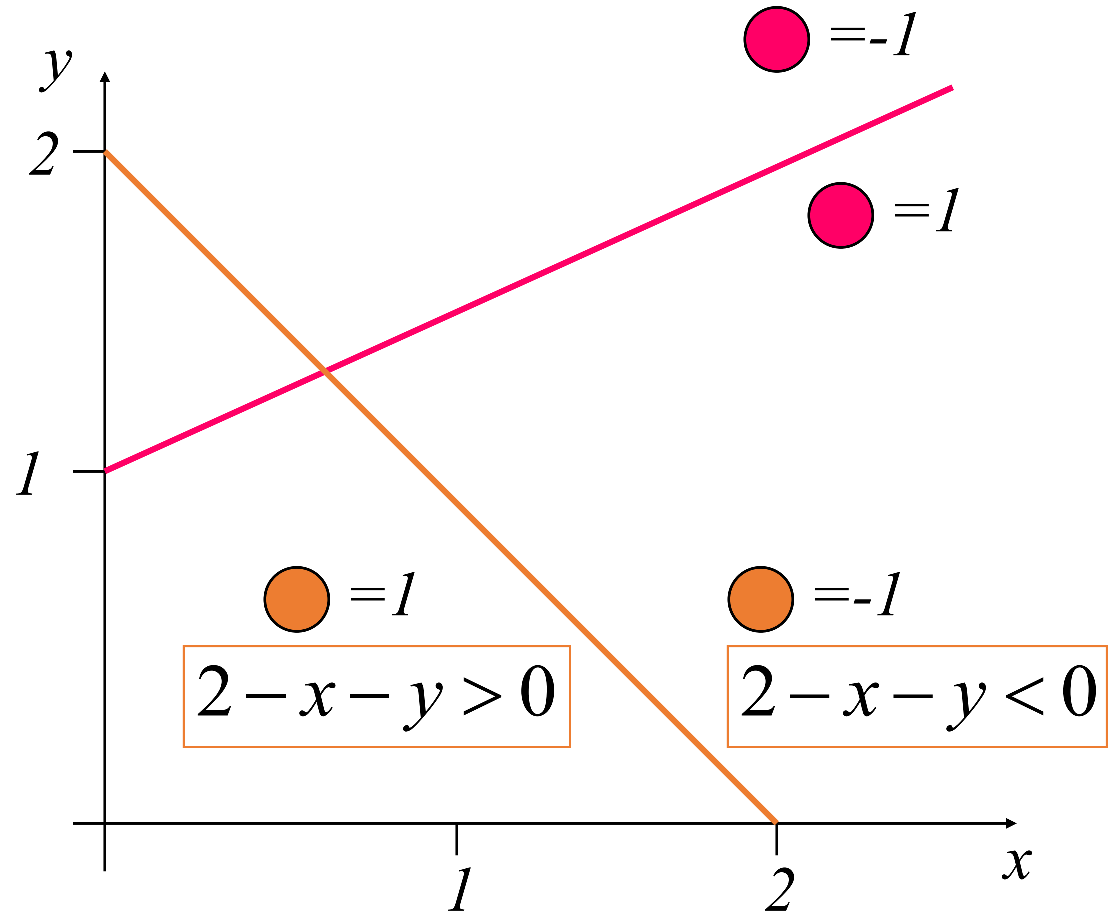


Multilayer Perceptron: What does it do?

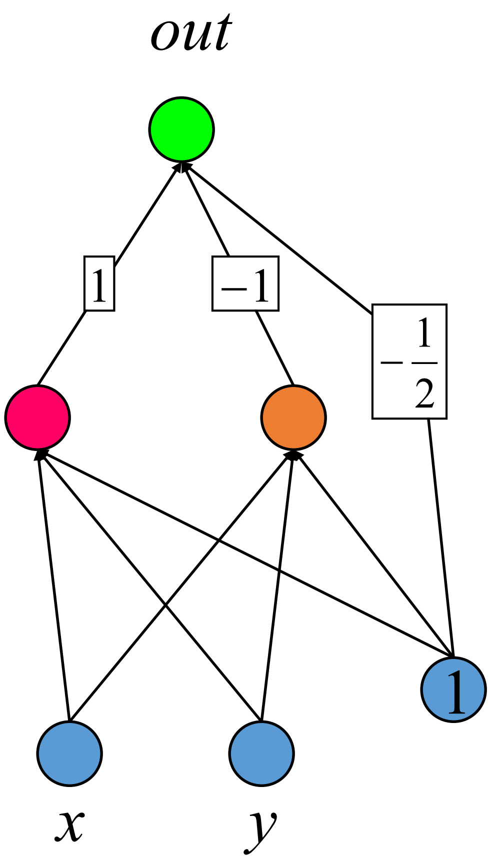


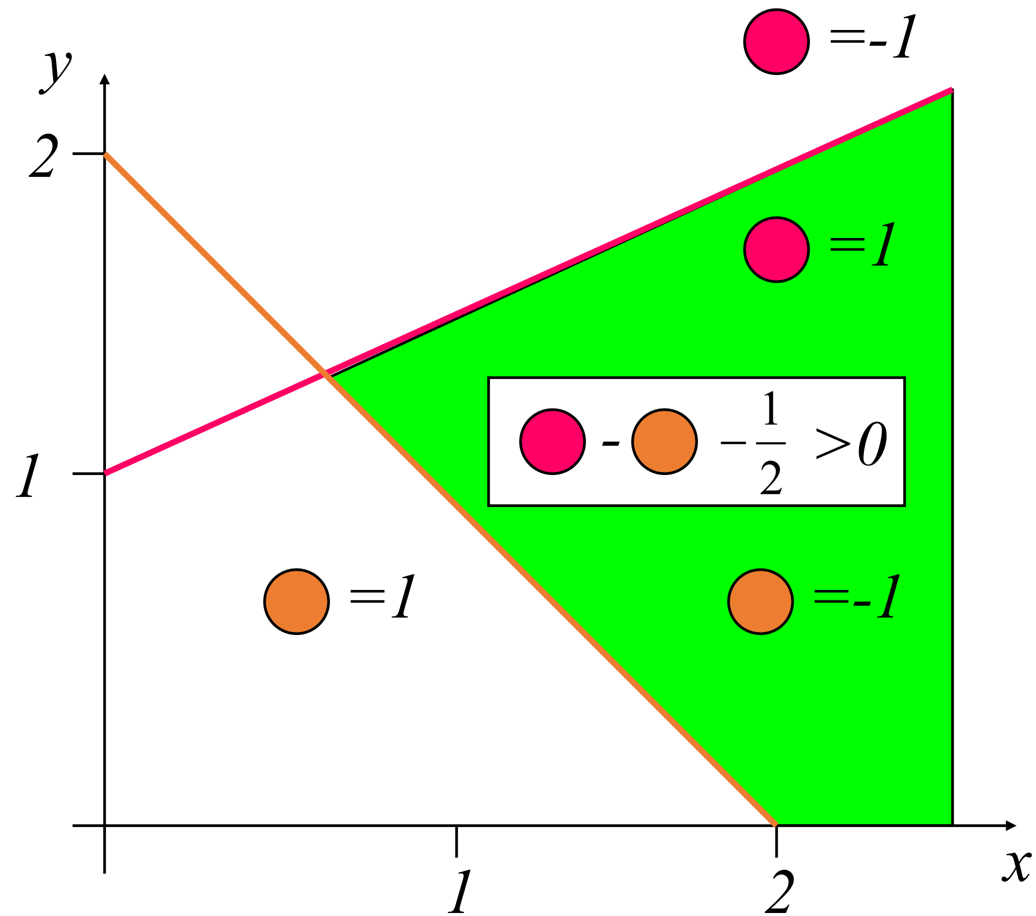
Multilayer Perceptron: What does it do?





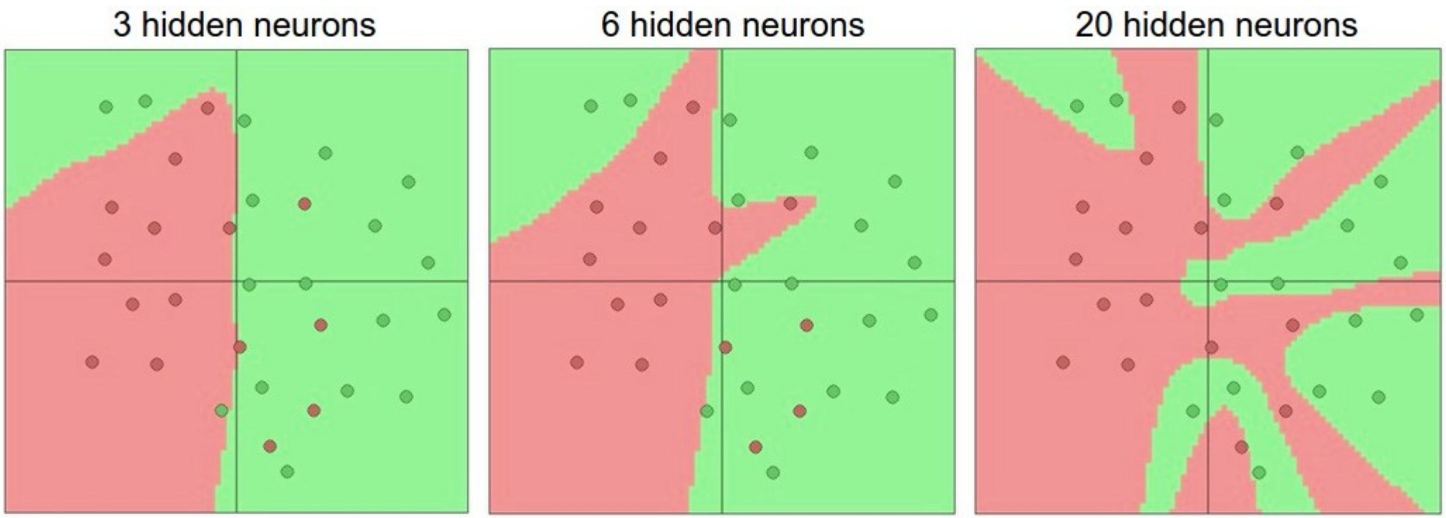
Multilayer Perceptron: What does it do?





Idea 2: Activation functions

Non-linearities needed to learn complex (non-linear) representations of data, otherwise the NN would be just a linear function $\mathrm{W}_1\mathrm{W}_2 x = Wx$

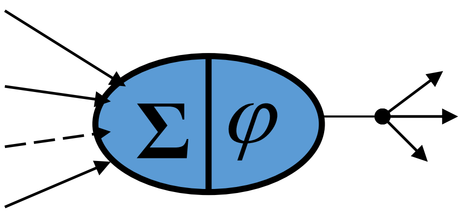


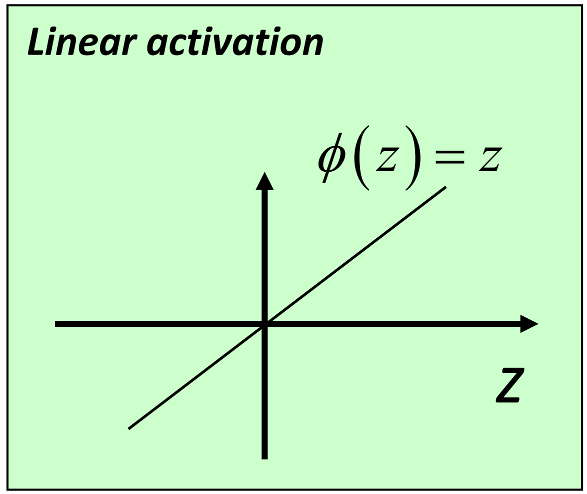
http://cs231n.github.io/assets/nn1/layer_sizes.jpeg

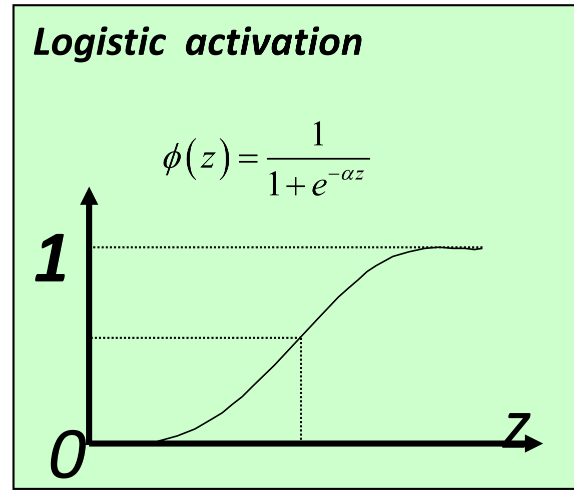
More layers and neurons can approximate more complex functions

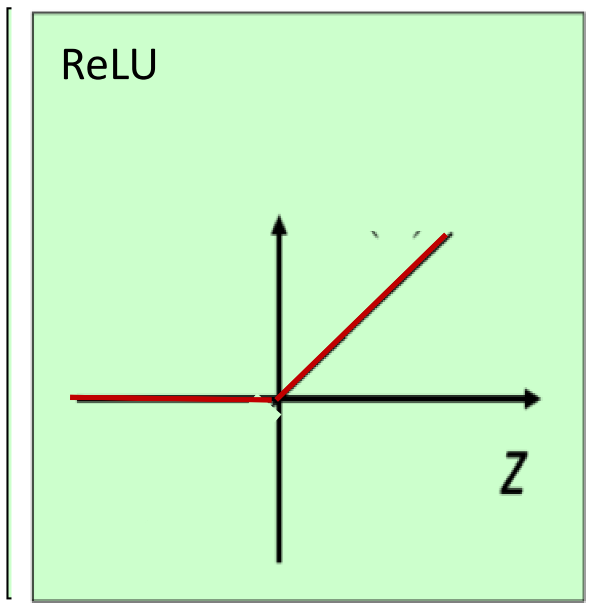
Full list: https://en.wikipedia.org/wiki/Activation_function

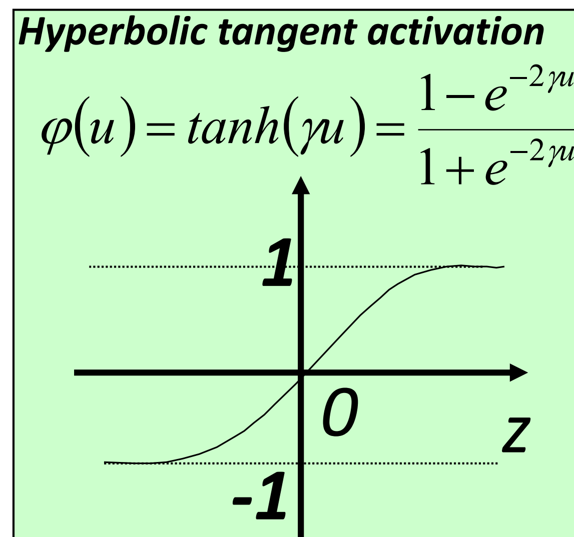
Activation Functions



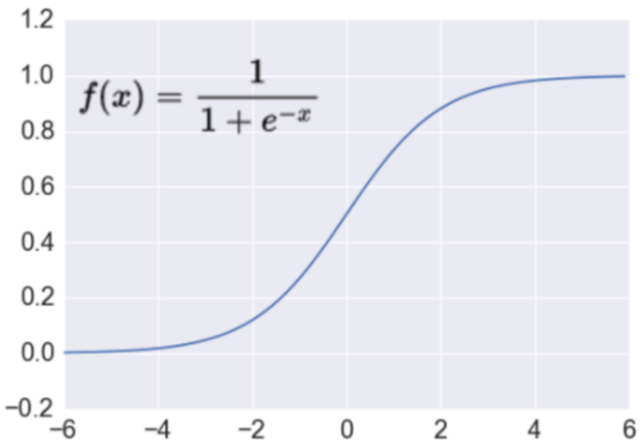








Activation: Sigmoid



http://adilmoujahid.com/images/activation.png

Takes a real-valued number and "squashes" it into range between 0 and 1.

$$R^n \rightarrow [0,1]$$

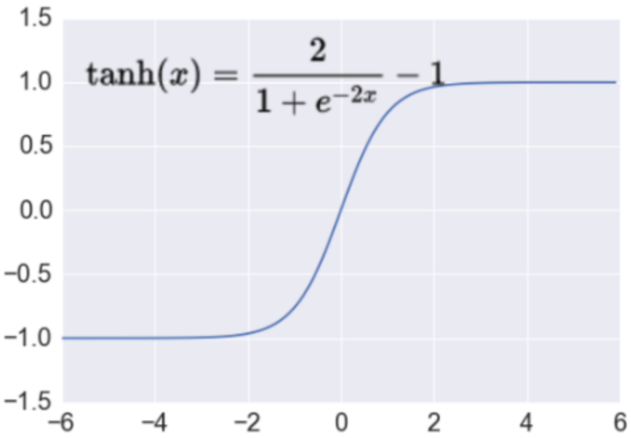
\+ Nice interpretation as the **firing rate** of a neuron

- 0 = not firing at all
- 1 = fully firing

\- Sigmoid neurons **saturate** and **kill gradients**, thus NN will barely learn

- when the neuron's activation are 0 or 1 (saturate)
  - ☐ gradient at these regions almost zero
  - ☐ almost no signal will flow to its weights
  - ☐ if initial weights are too large then most neurons would saturate

Activation: Tanh



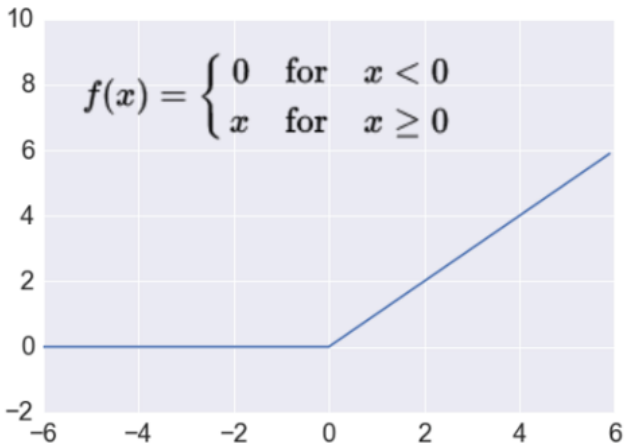
http://adilmoujahid.com/images/activation.png

Takes a real-valued number and "squashes" it into range between -1 and 1.

$$R^n \rightarrow [-1,1]$$

- Like sigmoid, tanh neurons **saturate**
- Unlike sigmoid, output is **zero-centered**
- Tanh is a **scaled sigmoid**: $\tanh(x) = 2sigm(2x) - 1$

Activation: ReLU



http://adilmoujahid.com/images/activation.png

Takes a real-valued number and thresholds it at zero $\mathrm{f}(x) = \max(0, x)$

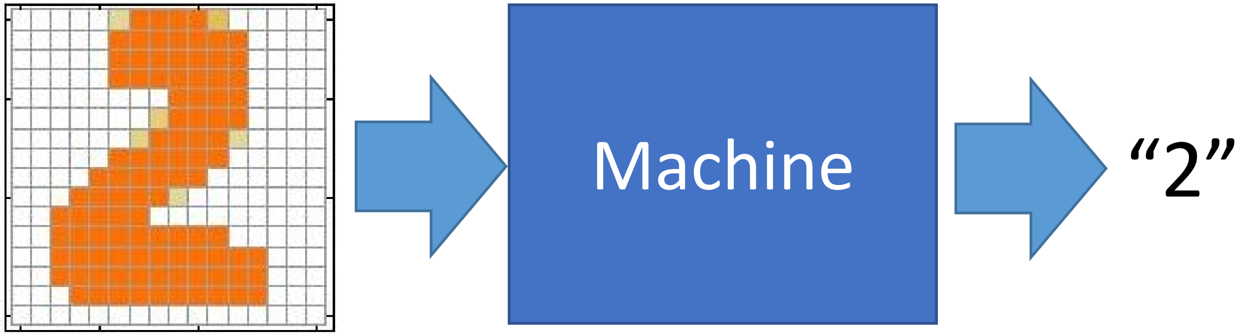
$$R^n \rightarrow R^n_+$$

Most Deep Networks use ReLU nowadays

- ☐ Trains much **faster**
  - accelerates the convergence of SGD
  - due to linear, non-saturating form
- ☐ Less expensive operations
  - compared to sigmoid/tanh (exponentials etc.)
  - implemented by simply thresholding a matrix at zero
- ☐ More **expressive**
- ☐ Reduces the **gradient vanishing problem**

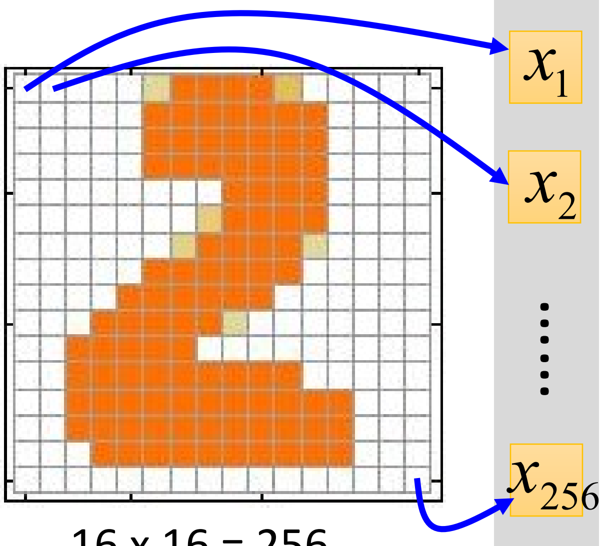
Example Application

- Handwriting Digit Recognition



Handwriting Digit Recognition

Input

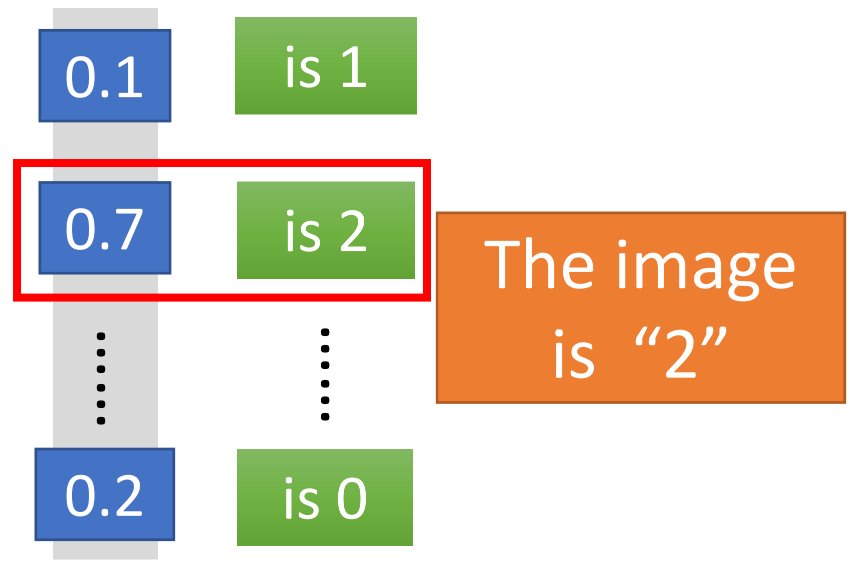


16 x 16 = 256

Ink → 1

No ink → 0

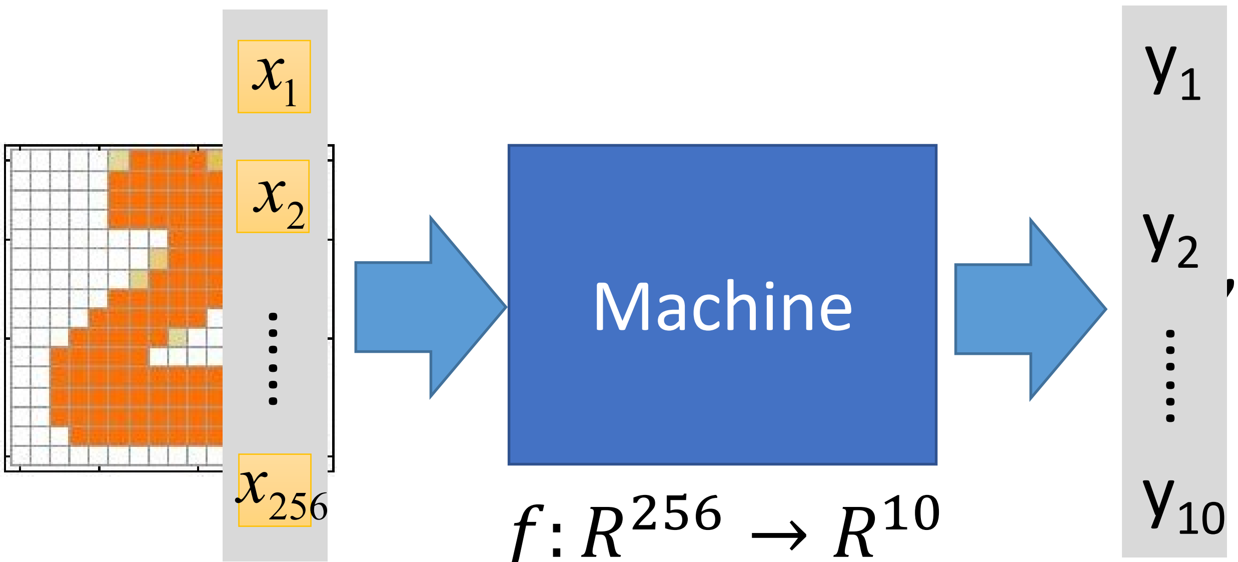
Output



Each dimension represents the confidence of a digit.

Example Application

- Handwriting Digit Recognition



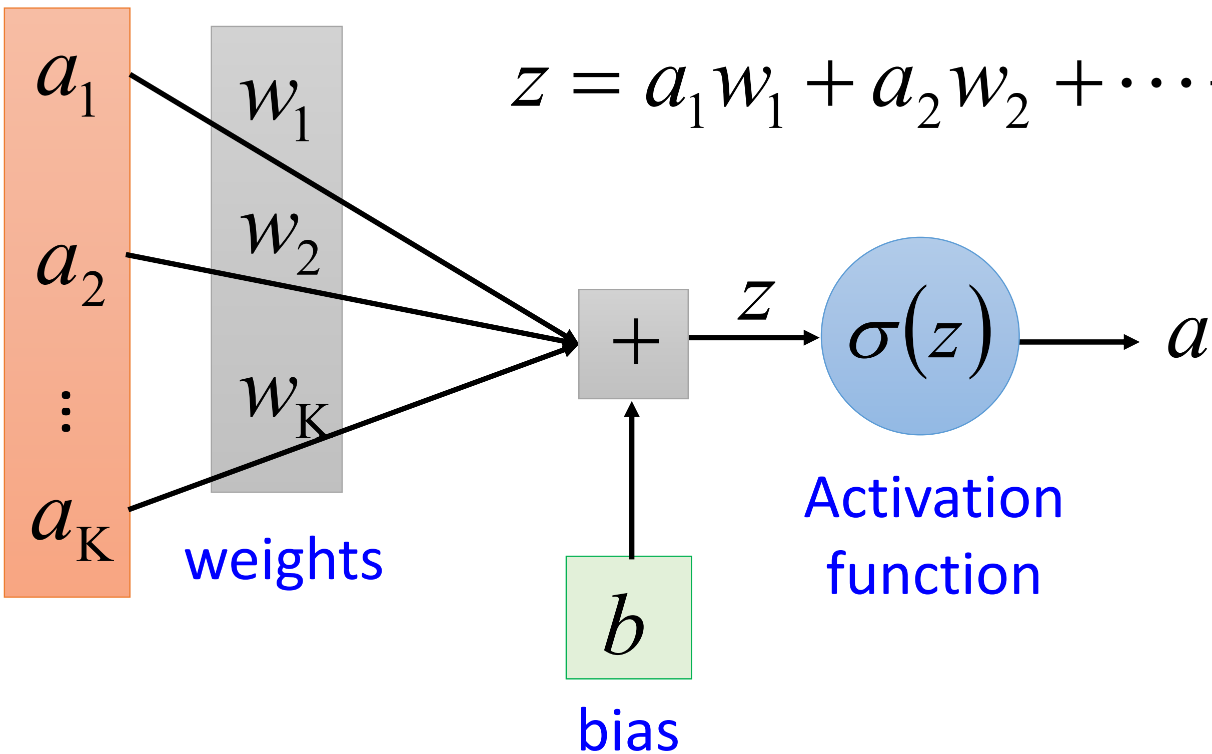
$f: R^{256} \rightarrow R^{10}$

In deep learning, the function f is represented by neural network

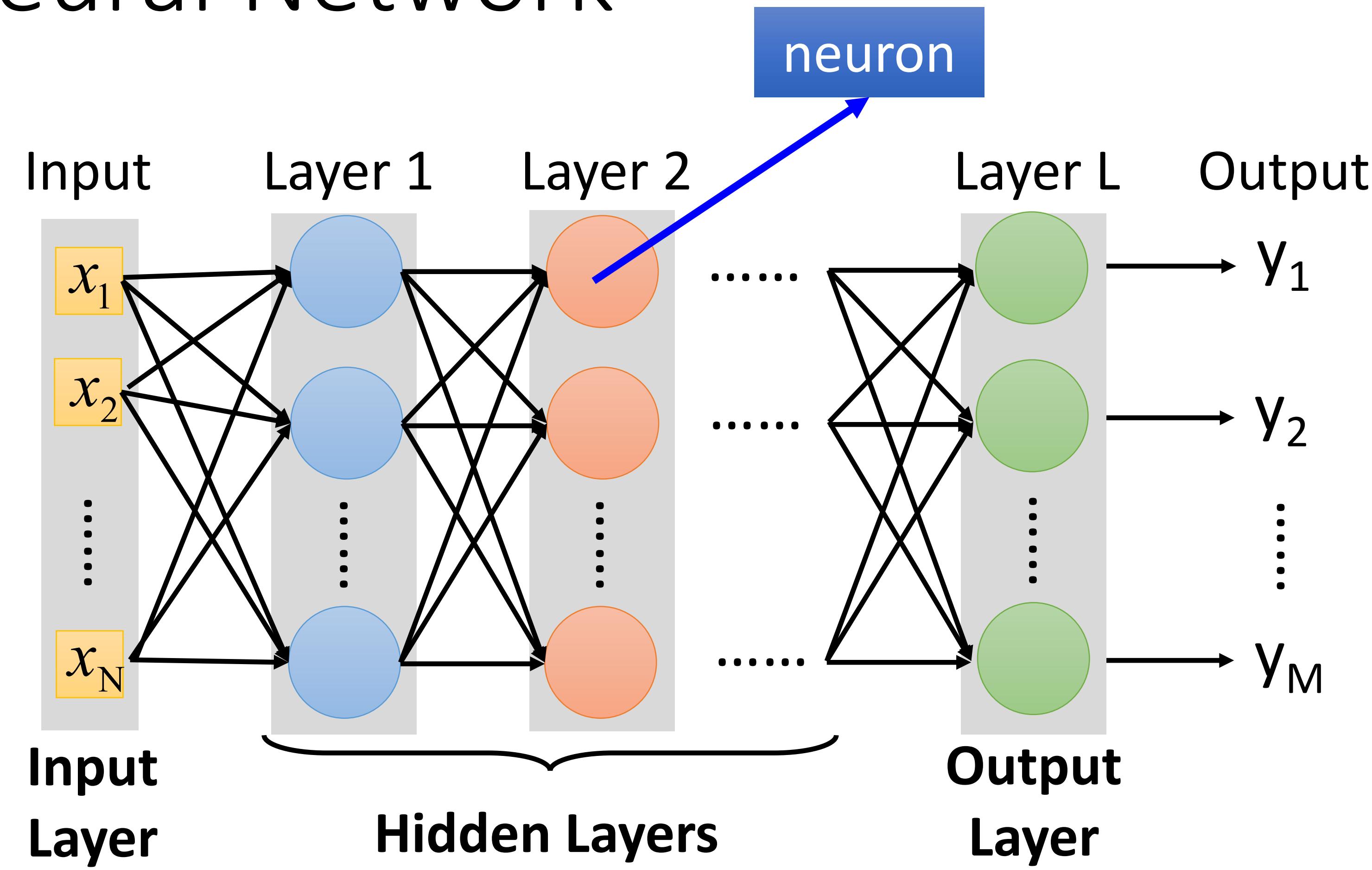
Element of Neural Network

Neuron $f: R^K \rightarrow R$

$$z = a_1 w_1 + a_2 w_2 + \cdots + a_K w_K + b$$

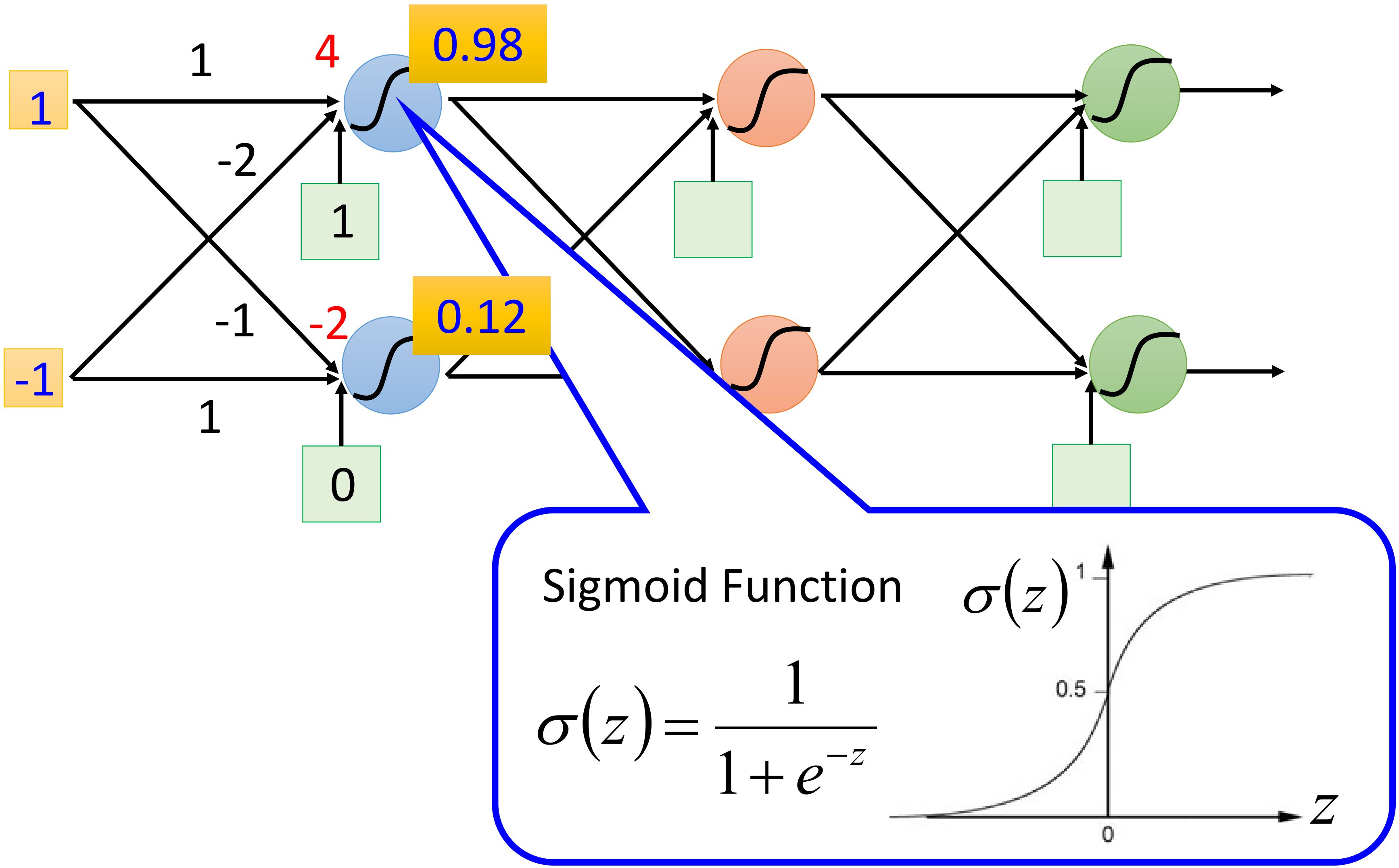


Neural Network

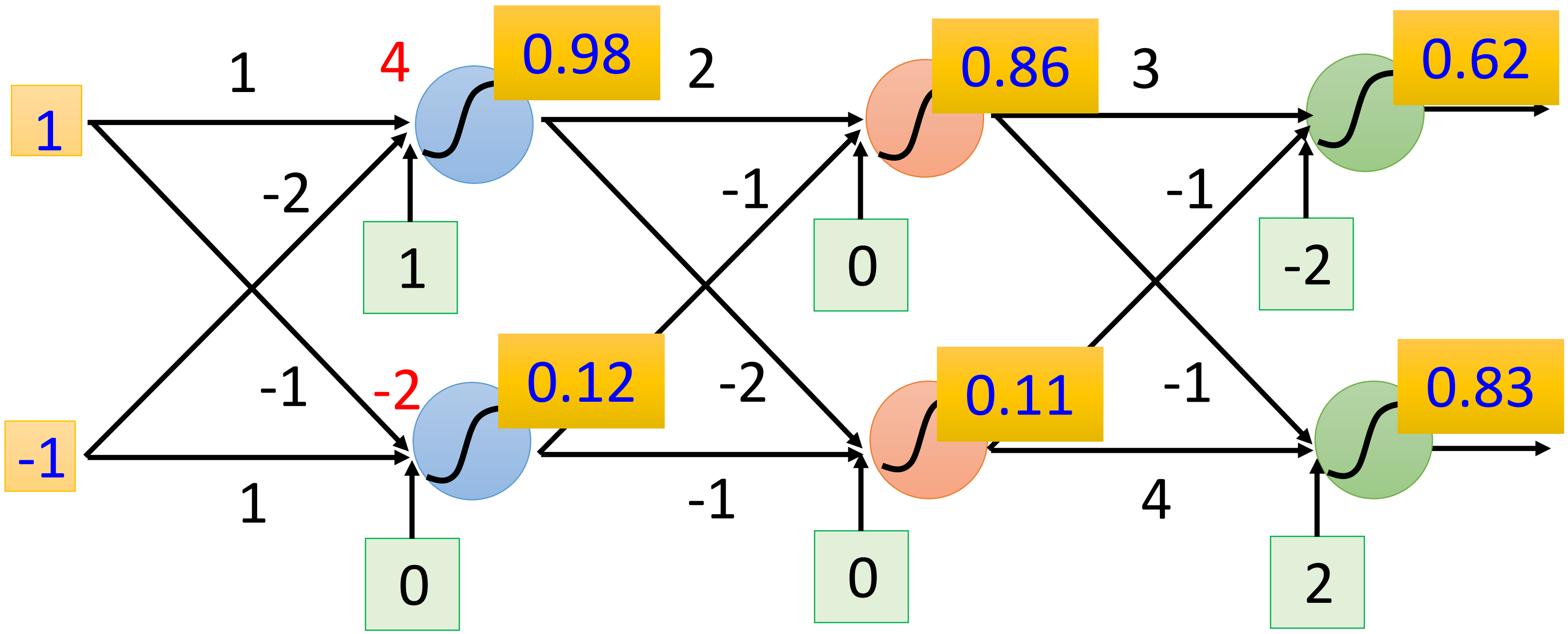


Deep means many hidden layers

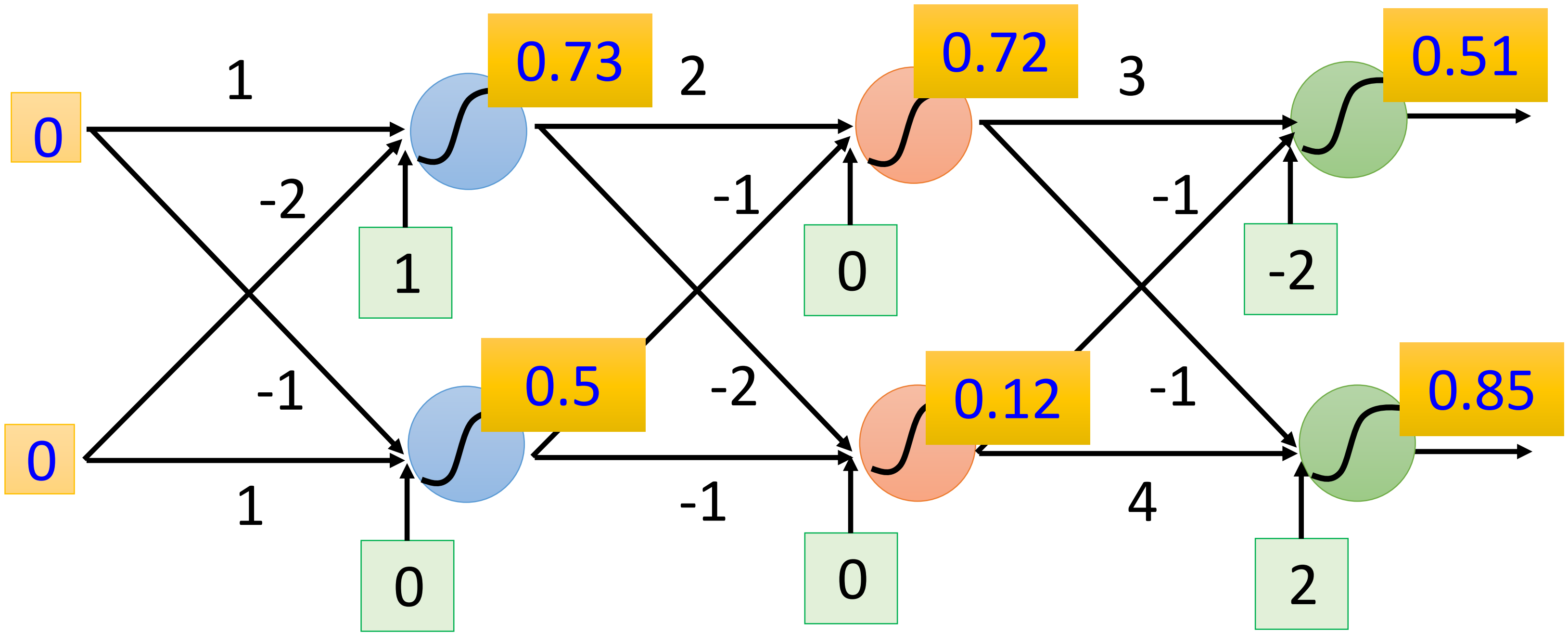
Example of Neural Network



Example of Neural Network



Example of Neural Network

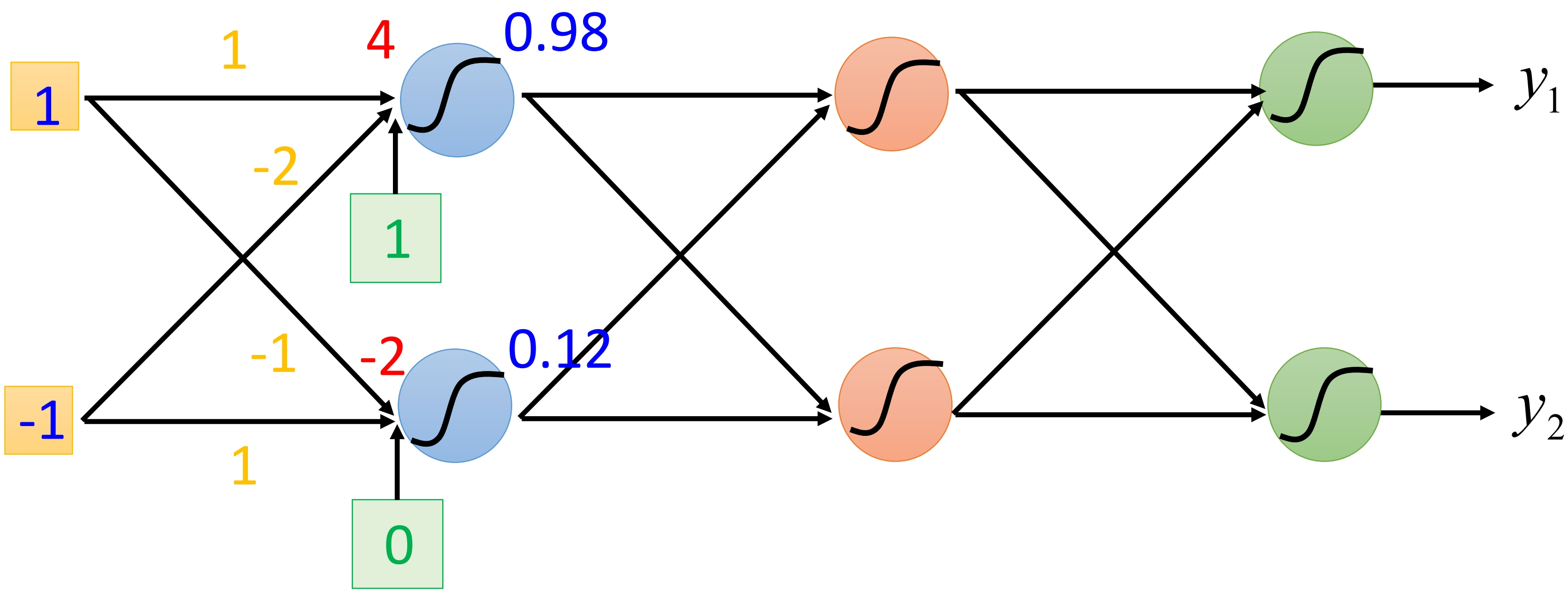


$$f: R^2 \rightarrow R^2$$

$$f\left(\begin{bmatrix} 1 \\ -1 \end{bmatrix}\right) = \begin{bmatrix} 0.62 \\ 0.83 \end{bmatrix} \quad f\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}\right) = \begin{bmatrix} 0.51 \\ 0.85 \end{bmatrix}$$

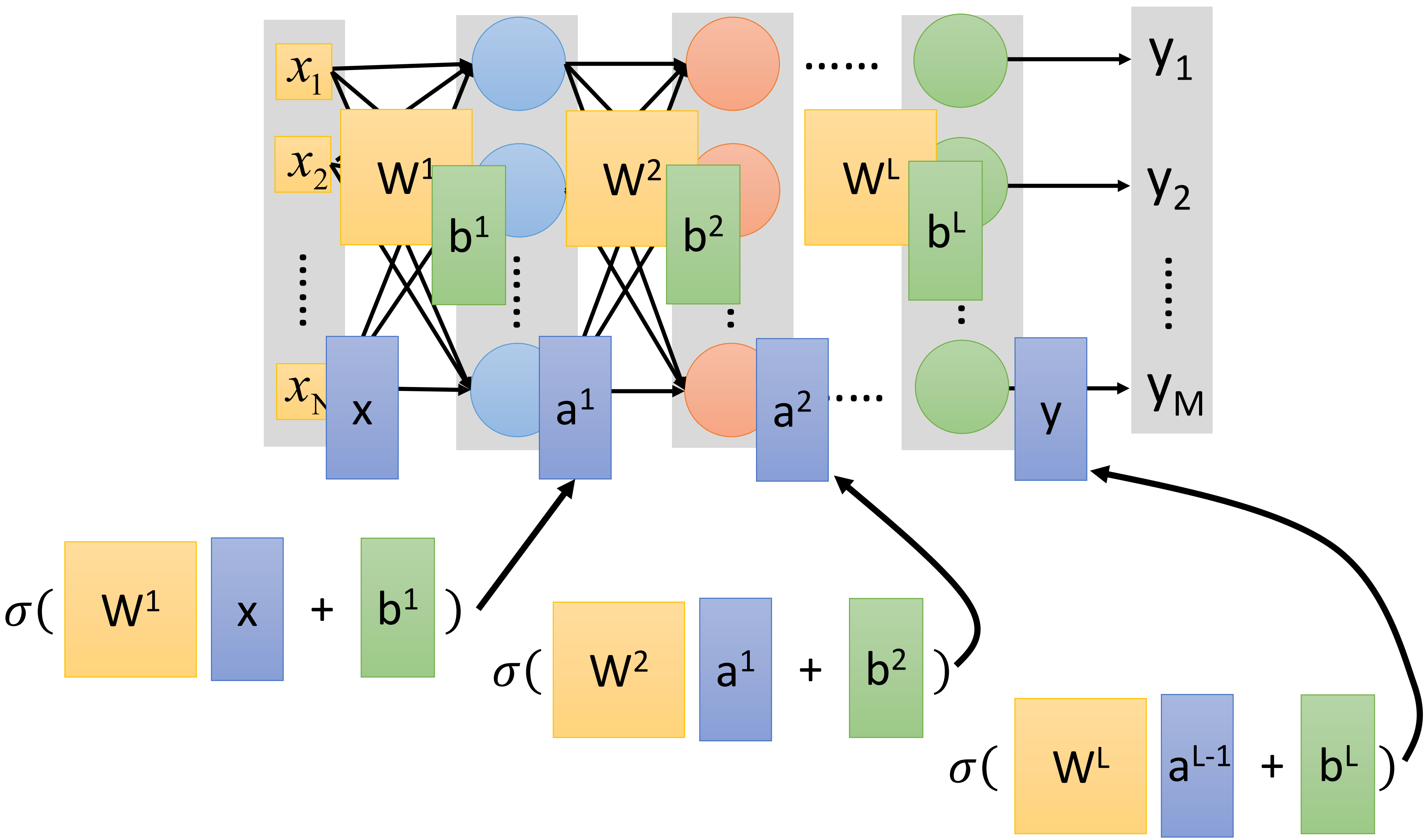
Different parameters define different function

Matrix Operation

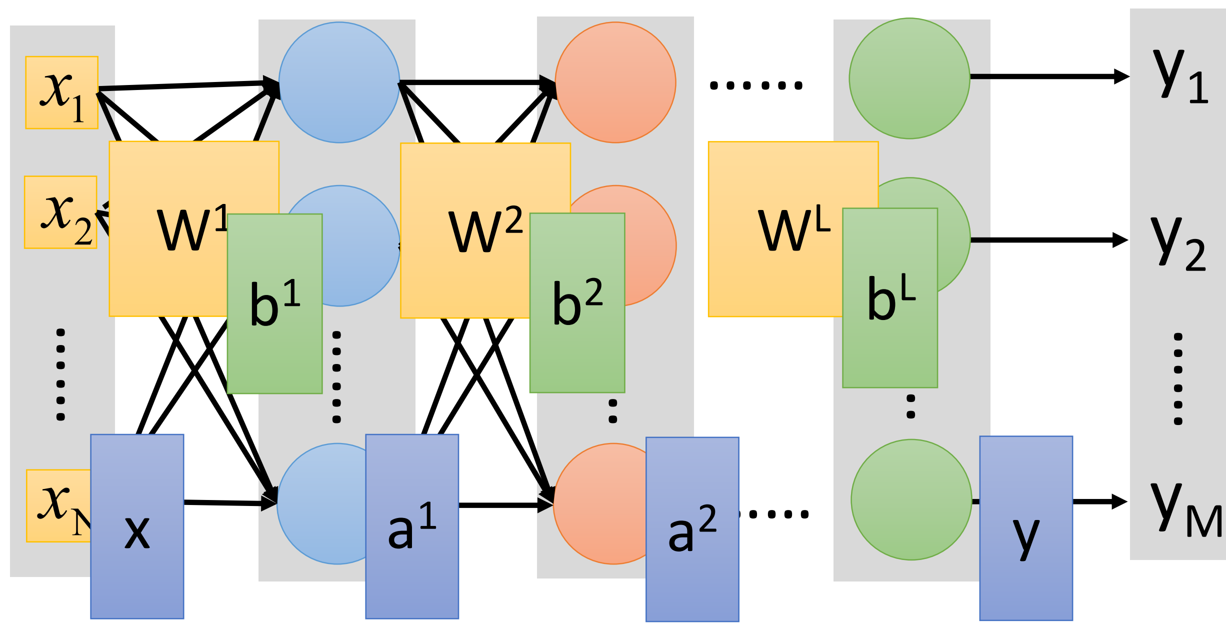


$$\sigma\Big(\underbrace{\begin{bmatrix} 1 & -2 \\ -1 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ -1 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix}}_{\begin{bmatrix} 4 \\ -2 \end{bmatrix}}\Big) = \begin{bmatrix} 0.98 \\ 0.12 \end{bmatrix}$$

Neural Network



Neural Network



$$y = f(x)$$

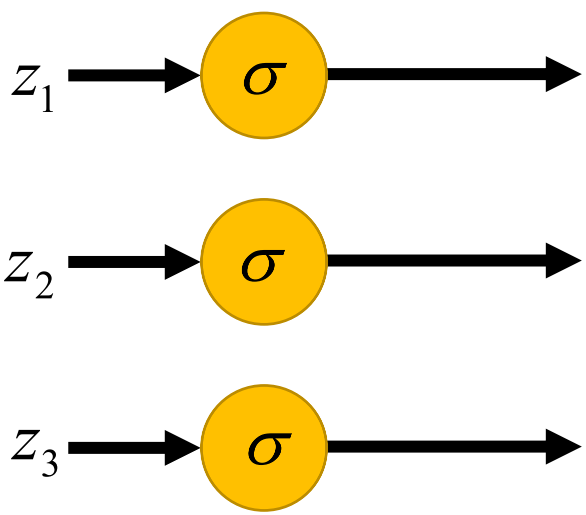
Using parallel computing techniques to speed up matrix operation

$$= \sigma(W^L \cdots \sigma(W^2 \, \sigma(W^1 x + b^1) + b^2) \cdots + b^L)$$

Softmax

- Softmax layer as the output layer

<u>Ordinary Layer</u>



$$y_1 = \sigma(z_1)$$

$$y_2 = \sigma(z_2)$$

$$y_3 = \sigma(z_3)$$

In general, the output of network can be any value.

May not be easy to interpret

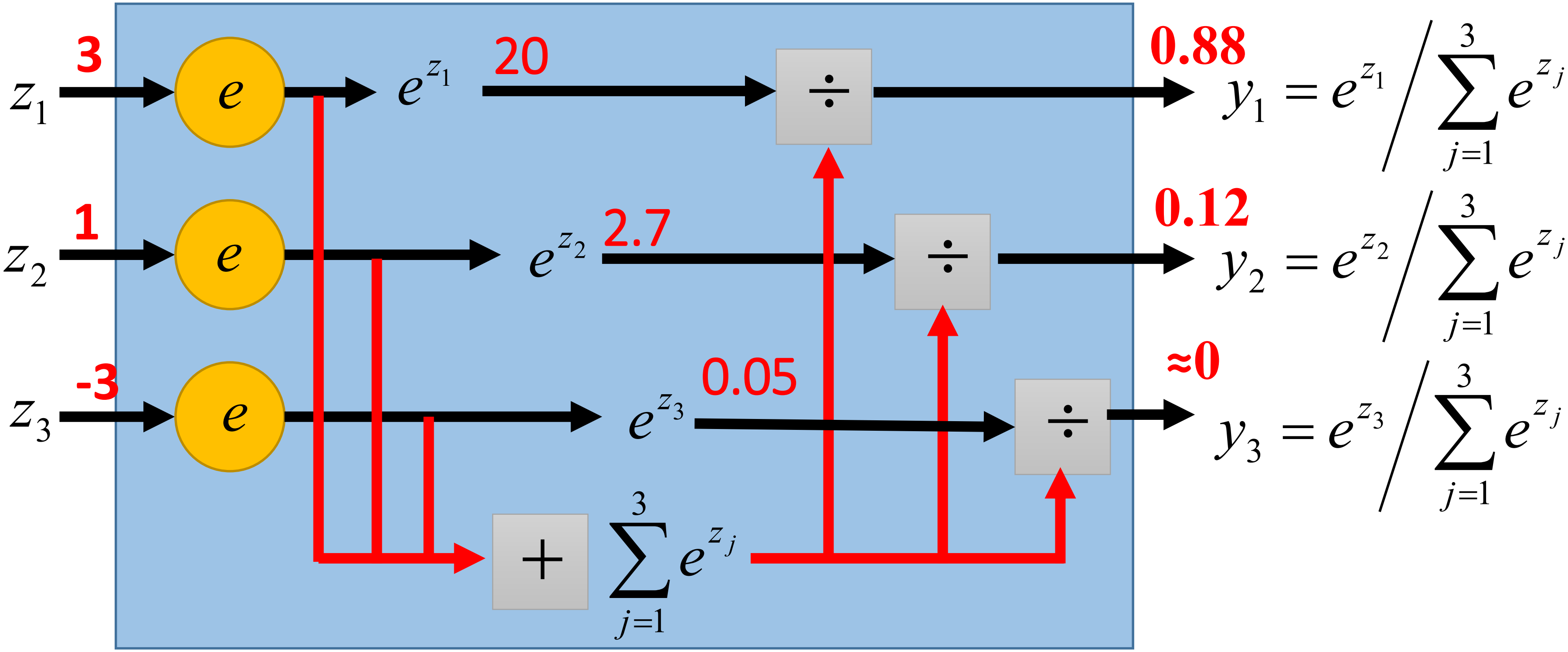
Softmax

- Softmax layer as the output layer

Probability:

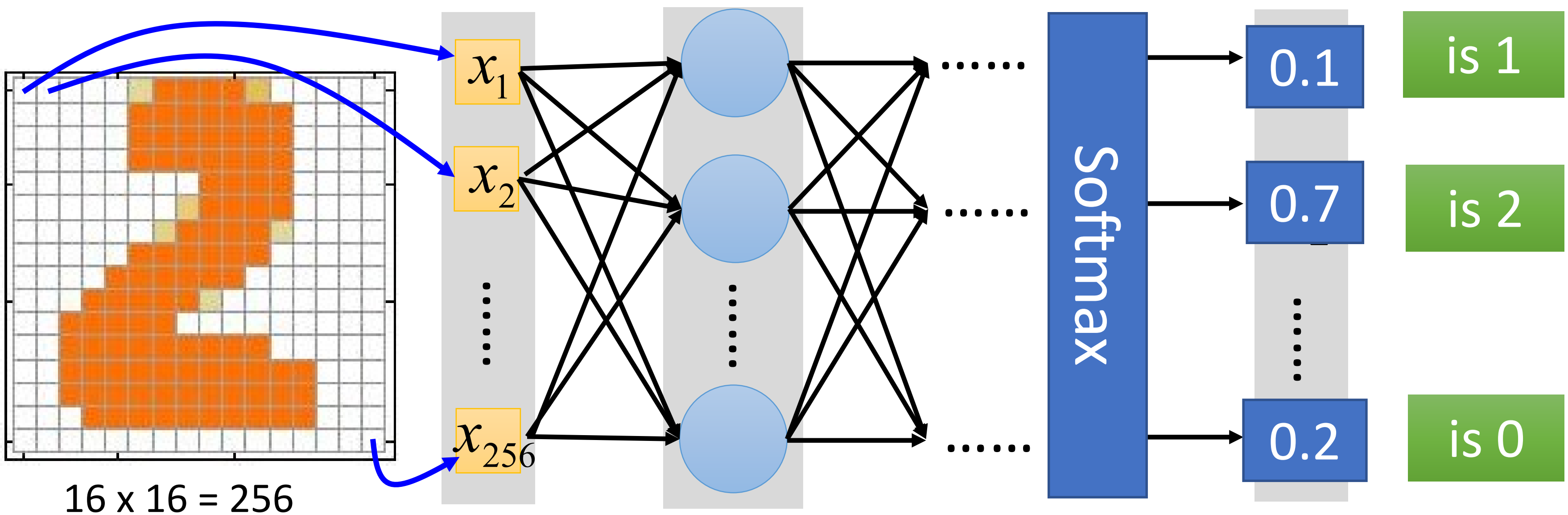
- $1 > y_i > 0$
- $\sum_i y_i = 1$

Softmax Layer



How to set network parameters

$$\theta = \{W^1, b^1, W^2, b^2, \cdots W^L, b^L\}$$



16 x 16 = 256

Ink → 1

No ink → 0

Set the network parameters θ such that ......

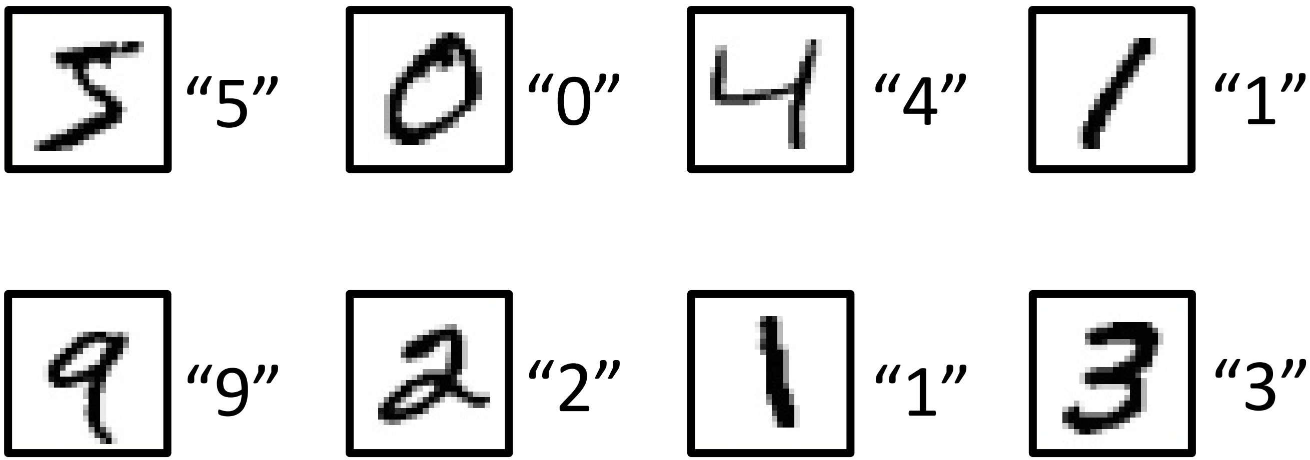
Input ... m value

How to let the neural network achieve this

Input:  y_2 has the maximum value

Training Data

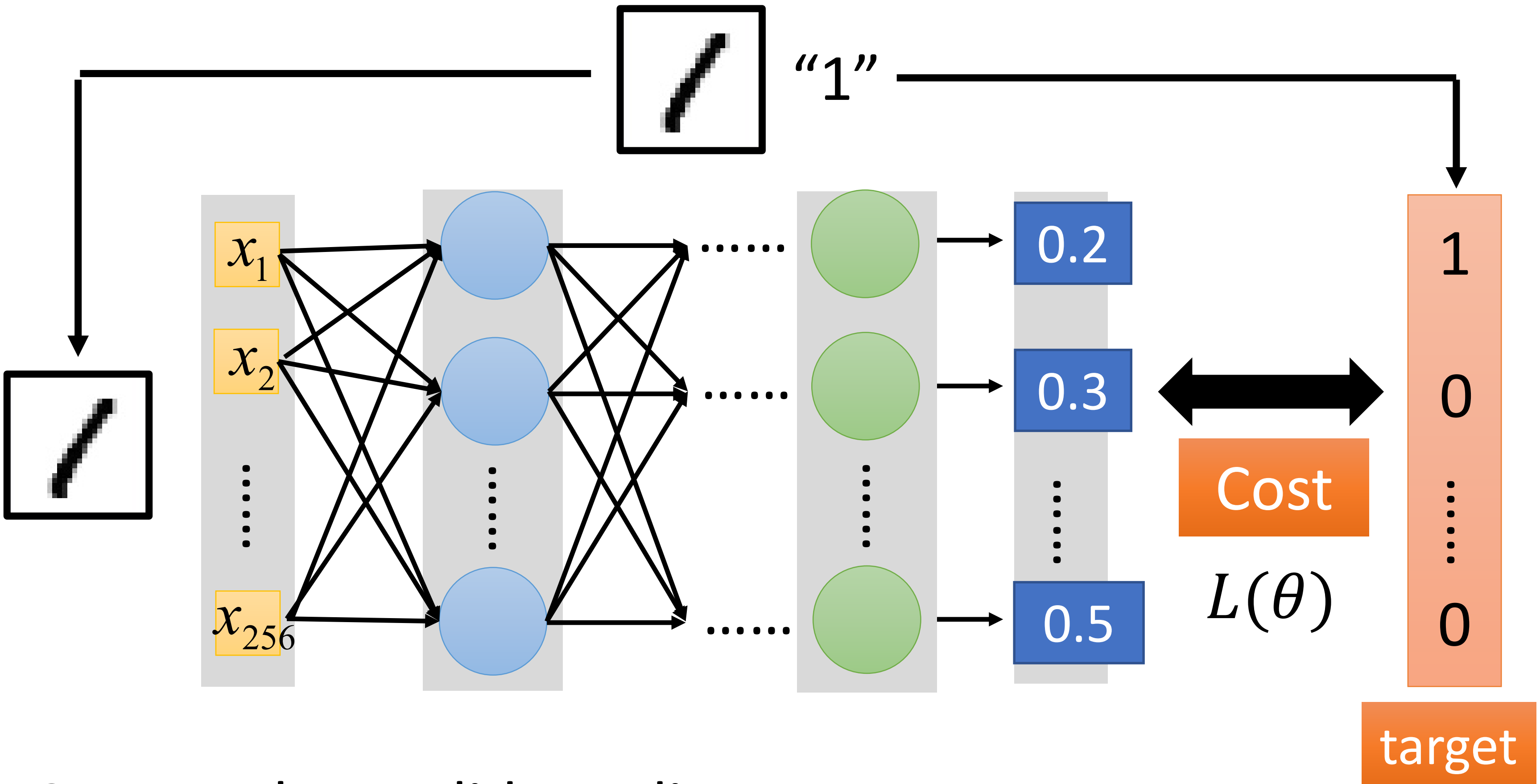
- Preparing training data: images and their labels



Using the training data to find the network parameters.

Cost

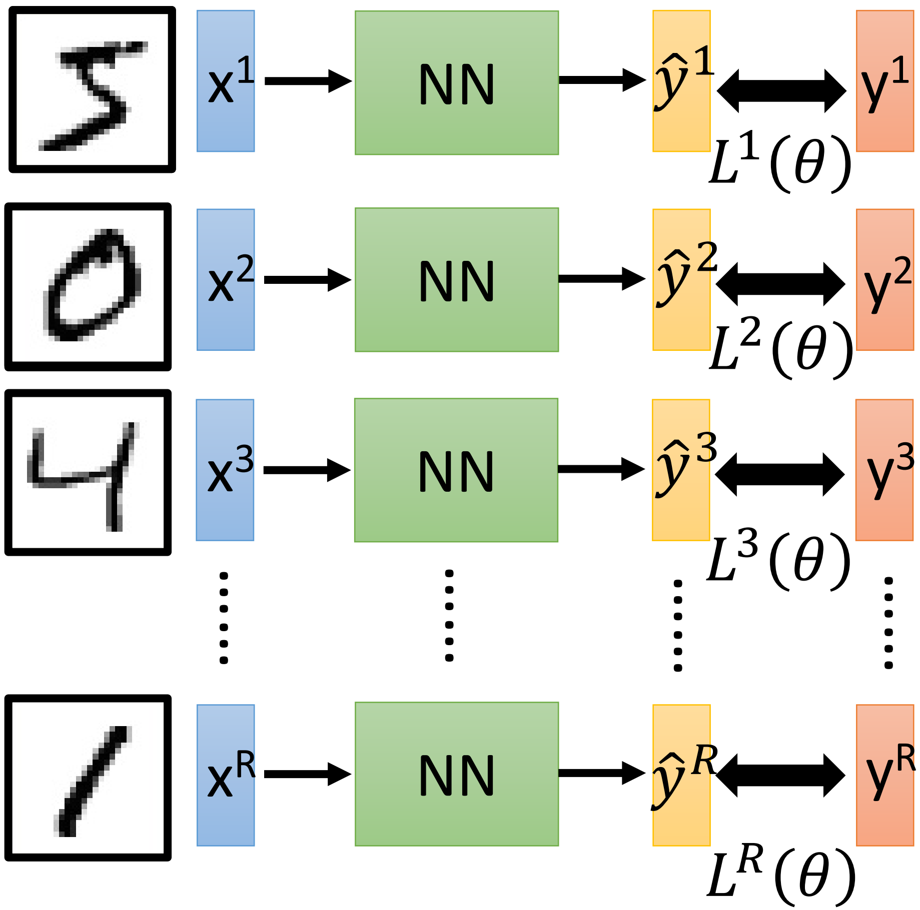
Given a set of network parameters θ, each example has a cost value.



Cost can be Euclidean distance or cross entropy of the network output and target

Total Cost

For all training data …



Total Cost:

$$C(\theta) = \sum_{r=1}^{R} L^r(\theta)$$

How bad the network parameters θ is on this task

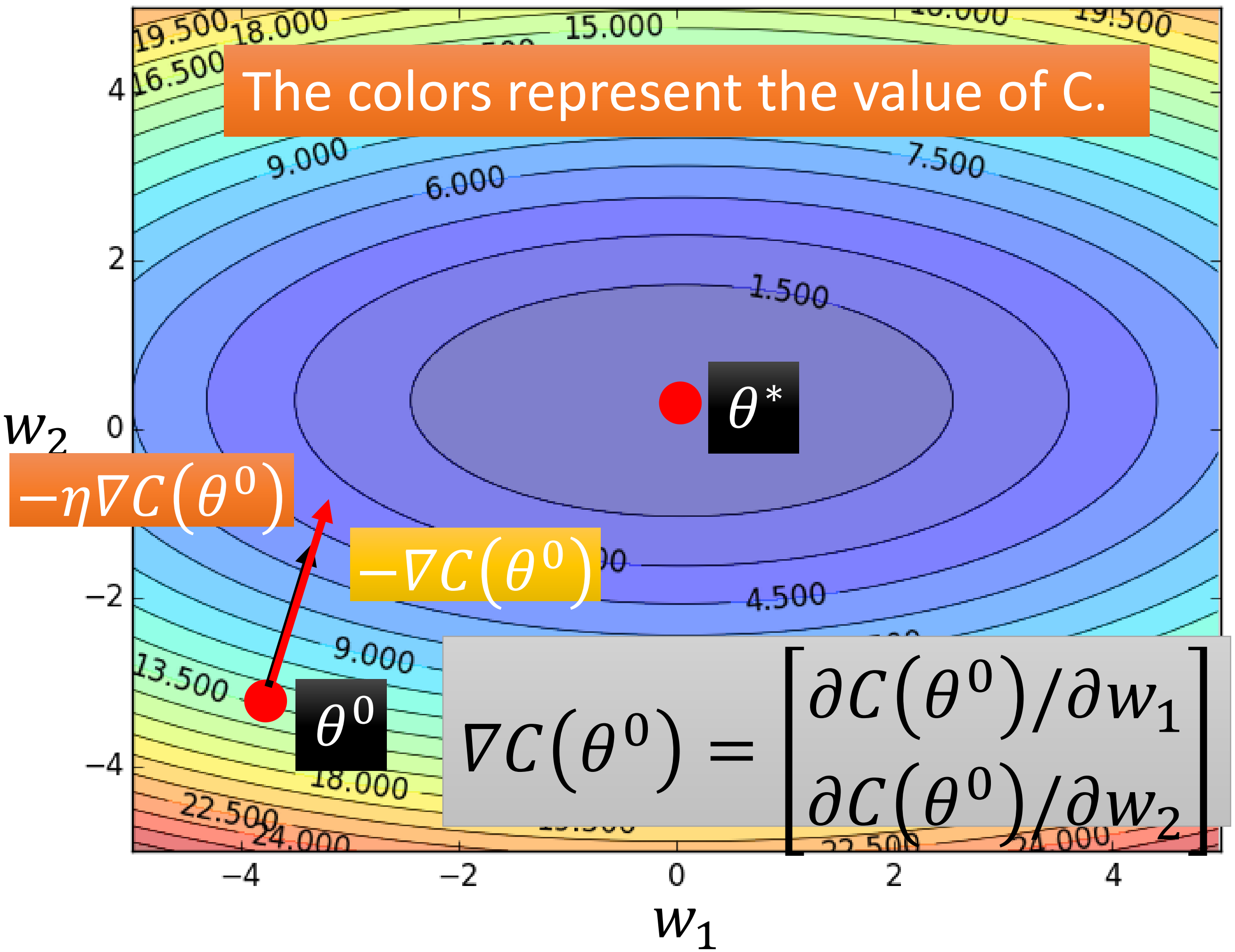
Find the network parameters θ^* that minimize this value

Gradient Descent

Assume there are only two parameters w_1 and w_2 in a network.

$$\theta = \{w_1, w_2\}$$

Error Surface



$$\nabla C(\theta^0) = \begin{bmatrix} \partial C(\theta^0)/\partial w_1 \\ \partial C(\theta^0)/\partial w_2 \end{bmatrix}$$

Randomly pick a starting point θ^0

Compute the negative gradient at θ^0

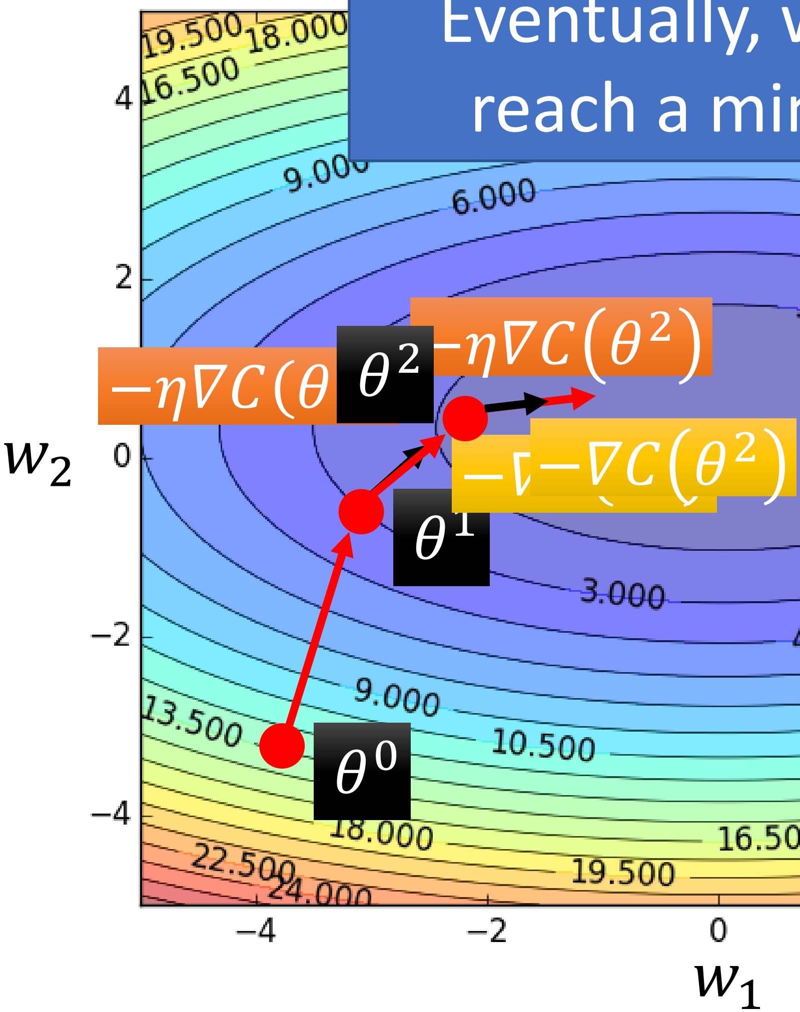
$\rightarrow -\nabla C(\theta^0)$

Times the learning rate η

$\rightarrow -\eta \nabla C(\theta^0)$

Gradient Descent

Eventually, we would reach a minima .....



Randomly pick a starting point θ^0

Compute the negative gradient at θ^0

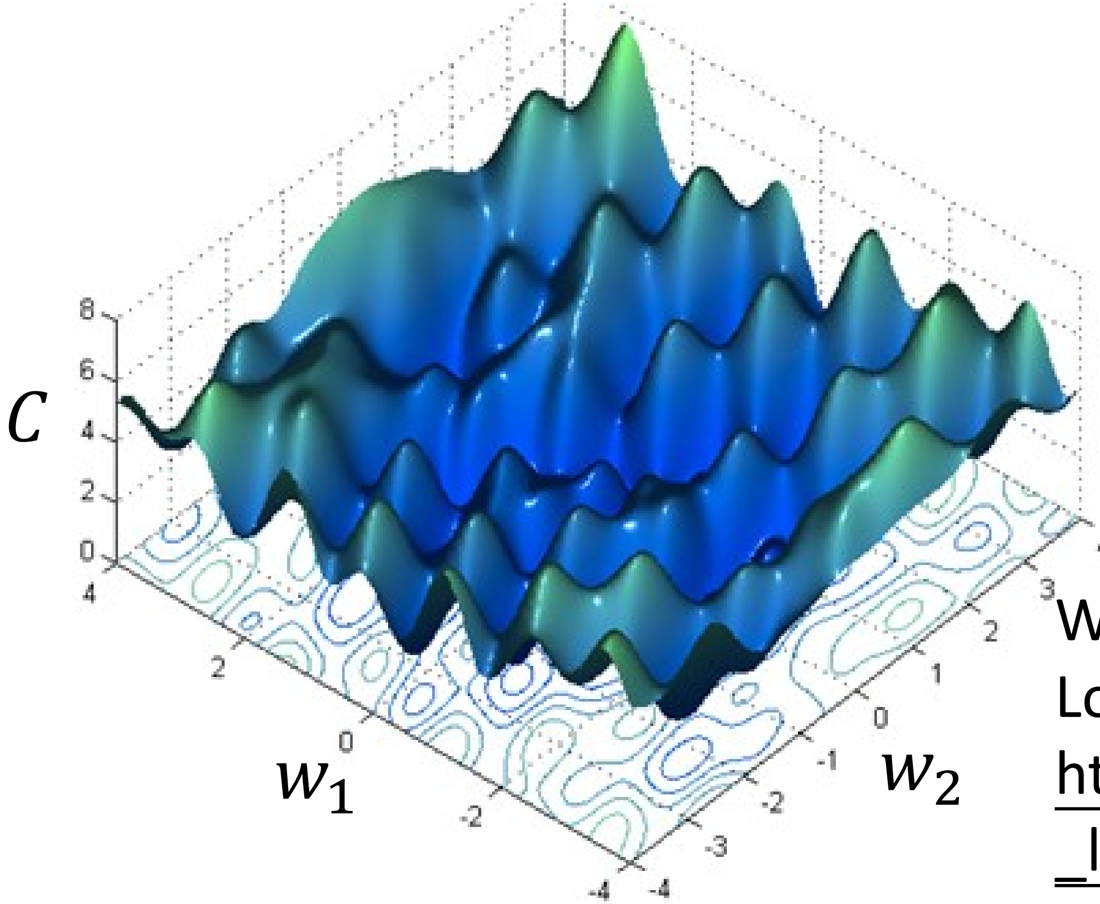
$\Rightarrow -\nabla C(\theta^0)$

Times the learning rate η

$\Rightarrow -\eta \nabla C(\theta^0)$

Local Minima

- Gradient descent never guarantee global minima

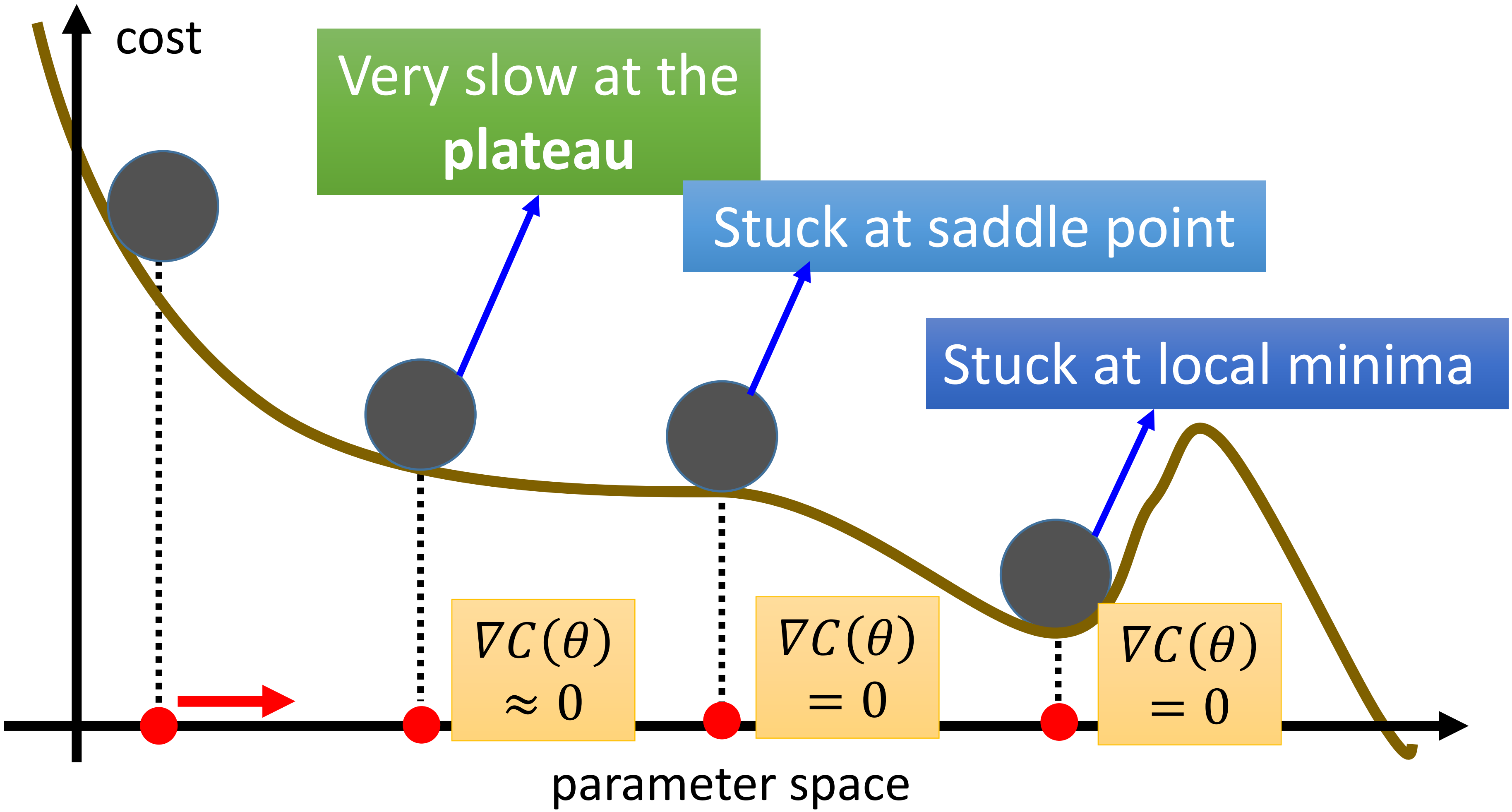


Different initial point θ^0

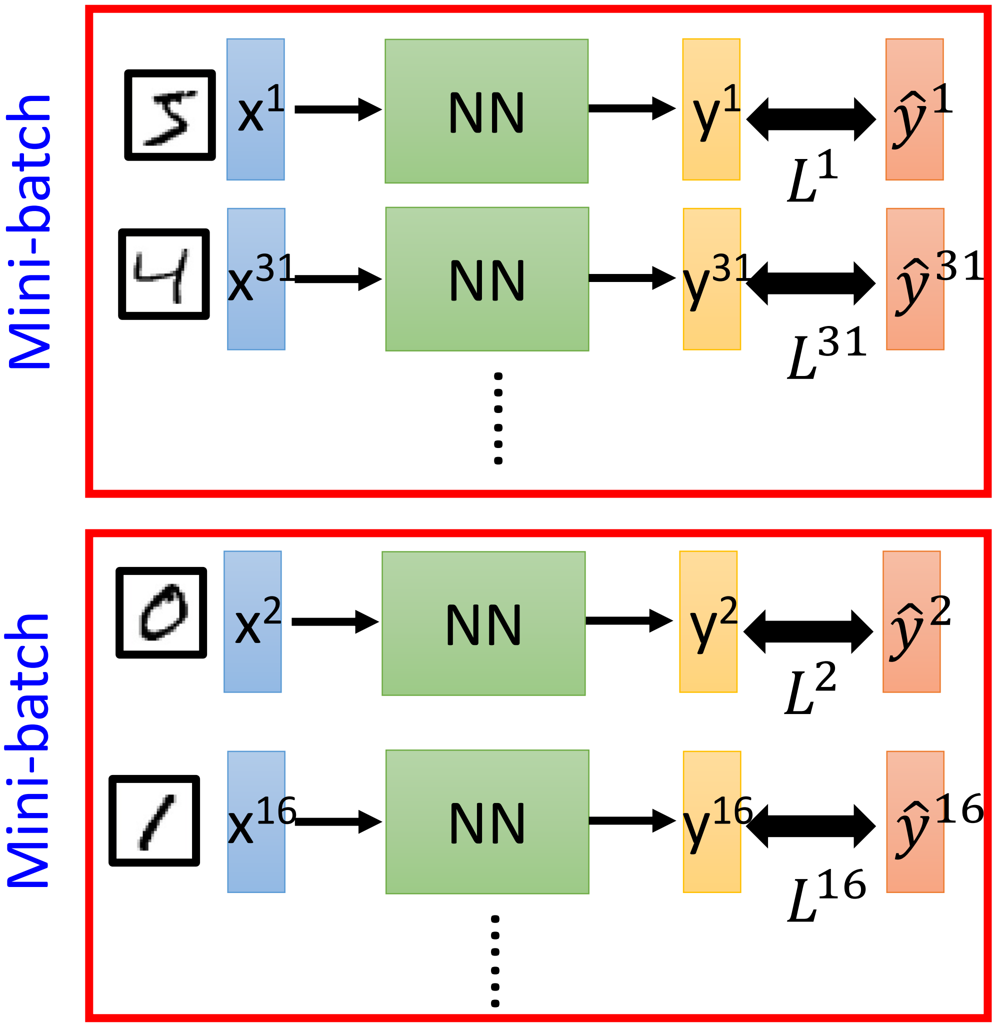
Reach different minima, so different results

Who is Afraid of Non-Convex Loss Functions? http://videolectures.net/eml07_lecun_wia/

Besides local minima ……



Mini-batch



- Randomly initialize θ^0
- Pick the 1st batch

$$C = L^1 + L^{31} + \cdots$$

$$\theta^1 \leftarrow \theta^0 - \eta \nabla C(\theta^0)$$

- Pick the 2nd batch

$$C = L^2 + L^{16} + \cdots$$

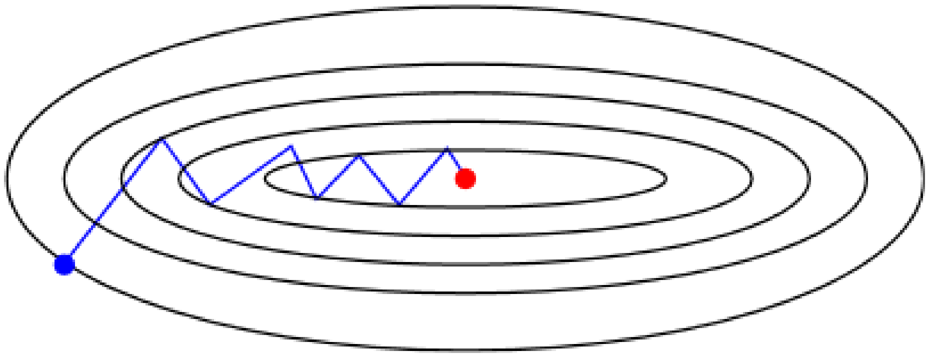
$$\theta^2 \leftarrow \theta^1 - \eta \nabla C(\theta^1)$$

$$\vdots$$

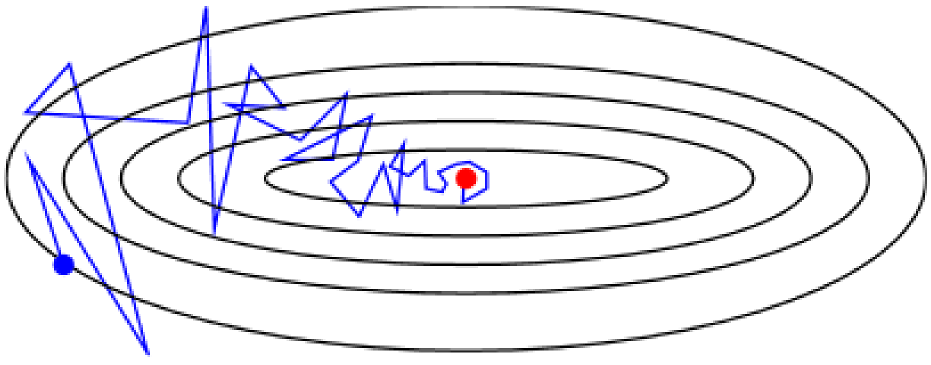
C is different each time when we update parameters!

SGD vs. GD

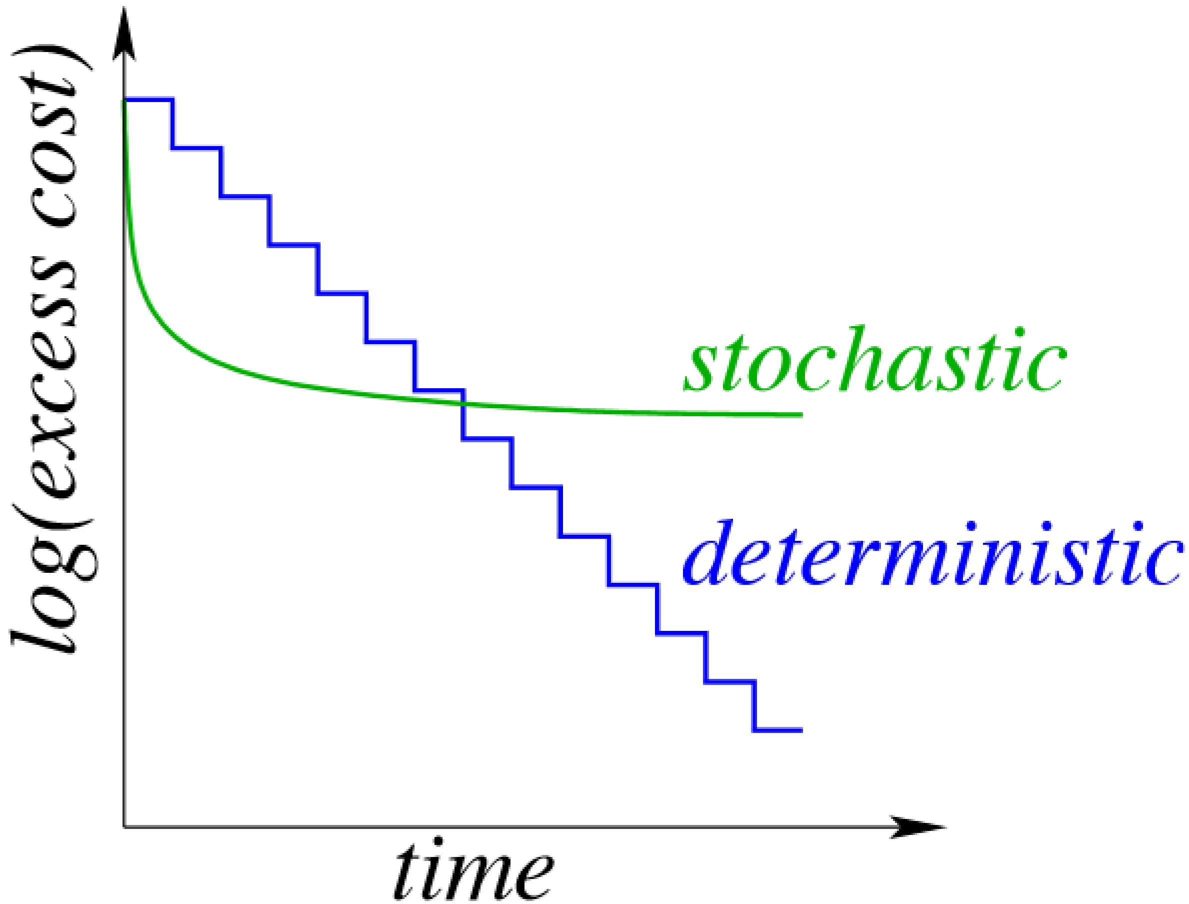
- Deterministic gradient method [Cauchy, 1847]:



- Stochastic gradient method [Robbins & Monro, 1951]:



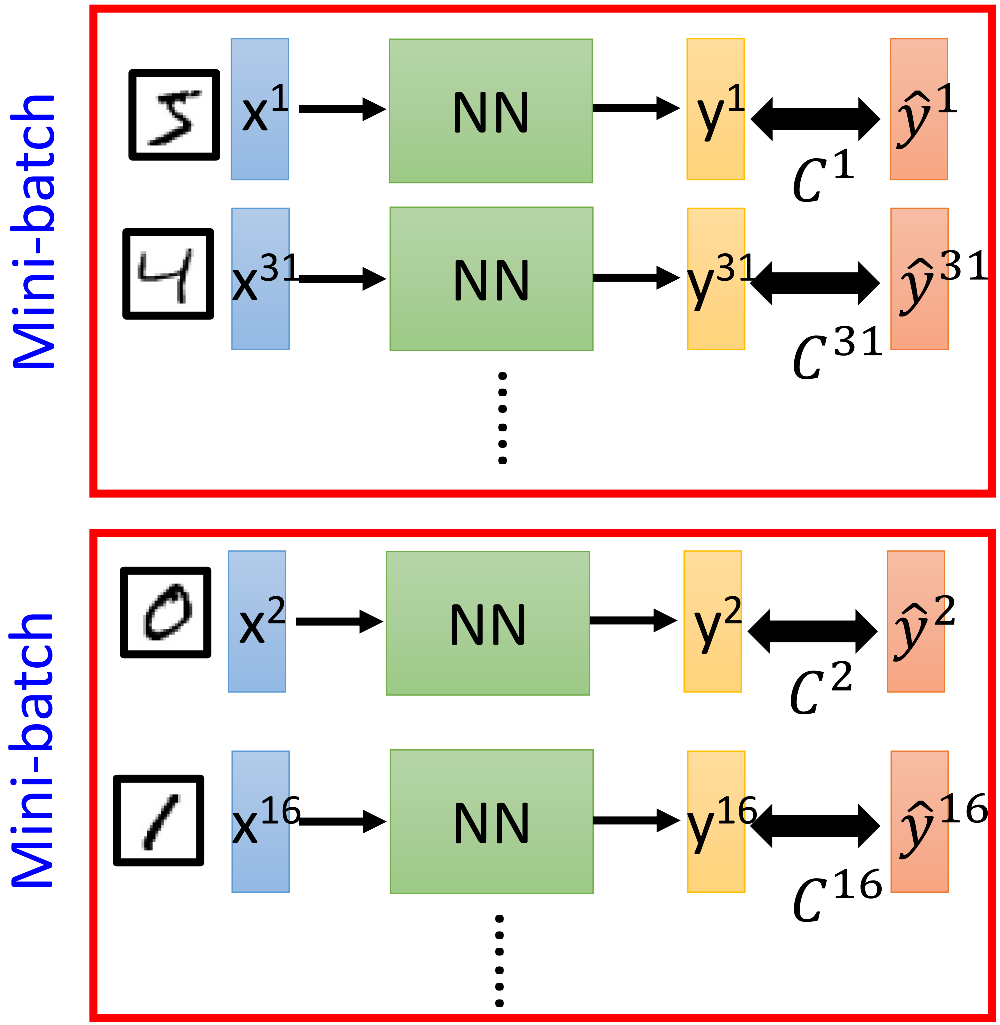
Convergence curves



Stochastic will be superior for low-accuracy/time situations.

Mini-batch

Faster | Better!



- Randomly initialize θ^0
- Pick the 1st batch

  $C = C^1 + C^{31} + \cdots$

  $\theta^1 \leftarrow \theta^0 - \eta \nabla C(\theta^0)$
- Pick the 2nd batch

  $C = C^2 + C^{16} + \cdots$

  $\theta^2 \leftarrow \theta^1 - \eta \nabla C(\theta^1)$

  ⋮
- Until all mini-batches have been picked

one epoch

Repeat the above process

Backpropagation: Computing Gradients

- If we choose a differentiable loss, then the the whole function will be differentiable with respect to all parameters.
- Because of non-linear activations whose combination is not convex, the overall learning problem is not convex.
- What does (stochastic) (sub)gradient descent do with non-convex functions? It finds a local minimum.
- To calculate gradients, we need to use the chain rule from calculus.
- Special name for (S)GD with chain rule invocations: backpropagation.

Backpropagation

For every node in the computation graph, we wish to calculate the first derivative of L_n with respect to that node. For any node a, let:

$$\bar{a} = \frac{\partial L_n}{\partial a}$$

Base case:

$$\bar{L_n} = \frac{\partial L_n}{\partial L_n} = 1$$

Backpropagation

For every node in the computation graph, we wish to calculate the first derivative of L_n with respect to that node. For any node a, let:

$$\bar{a} = \frac{\partial L_n}{\partial a}$$

After working forwards through the computation graph to obtain the loss L_n, we work *backwards* through the computation graph, using the chain rule to calculate $\bar{a}$ for every node a, making use of the work already done for nodes that depend on a.

$$\begin{aligned}
\frac{\partial L_n}{\partial a} &= \sum_{b:a\to b} \frac{\partial L_n}{\partial b} \cdot \frac{\partial b}{\partial a} \\
\bar{a} &= \sum_{b:a\to b} \bar{b} \cdot \frac{\partial b}{\partial a} \\
&= \sum_{b:a\to b} \bar{b} \cdot \begin{cases} 1 & \text{if } b = a + c \text{ for some } c \\ c & \text{if } b = a \cdot c \text{ for some } c \\ 1 - b^2 & \text{if } b = \tanh(a) \end{cases}
\end{aligned}$$

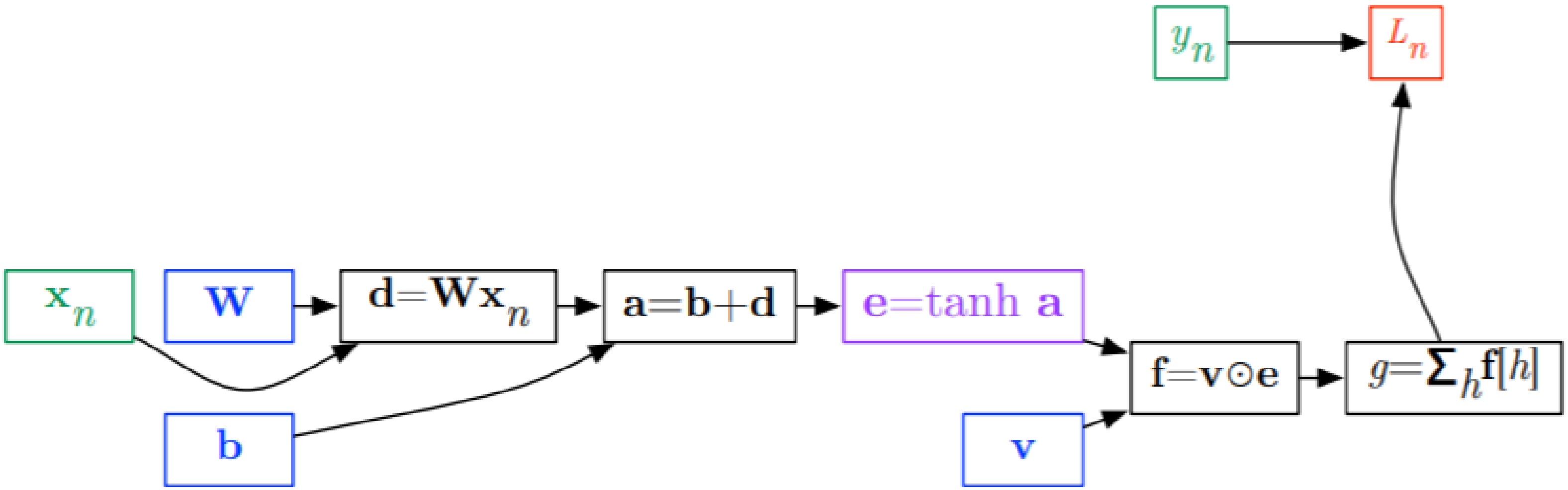
Backpropagation

Pointwise ("Hadamard") product for vectors in $\mathbb{R}^n$:

$$\mathbf{a} \odot \mathbf{b} = \begin{bmatrix} \mathbf{a}[1] \cdot \mathbf{b}[1] \\ \mathbf{a}[2] \cdot \mathbf{b}[2] \\ \vdots \\ \mathbf{a}[n] \cdot \mathbf{b}[n] \end{bmatrix}$$

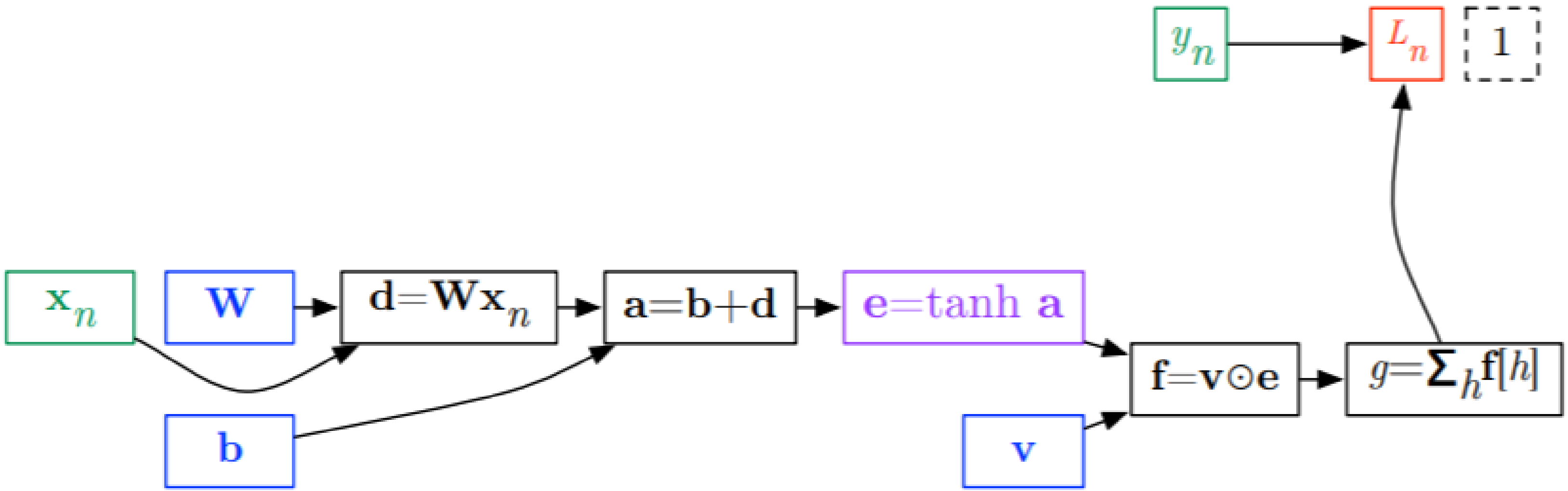
$$\begin{aligned} \bar{\mathbf{a}} &= \sum_{\mathbf{b}:\mathbf{a}\to\mathbf{b}} \sum_{i=1}^{|\mathbf{b}|} \bar{\mathbf{b}}[i] \cdot \frac{\partial \mathbf{b}[i]}{\partial \mathbf{a}} \\ &= \sum_{\mathbf{b}:\mathbf{a}\to\mathbf{b}} \begin{cases} \bar{\mathbf{b}} & \text{if } \mathbf{b} = \mathbf{a} + \mathbf{c} \text{ for some } \mathbf{c} \\ \bar{\mathbf{b}} \odot \mathbf{c} & \text{if } \mathbf{b} = \mathbf{a} \odot \mathbf{c} \text{ for some } \mathbf{c} \\ \bar{\mathbf{b}} \odot (\mathbf{1} - \mathbf{b} \odot \mathbf{b}) & \text{if } \mathbf{b} = \tanh(\mathbf{a}) \end{cases} \end{aligned}$$

Backpropagation



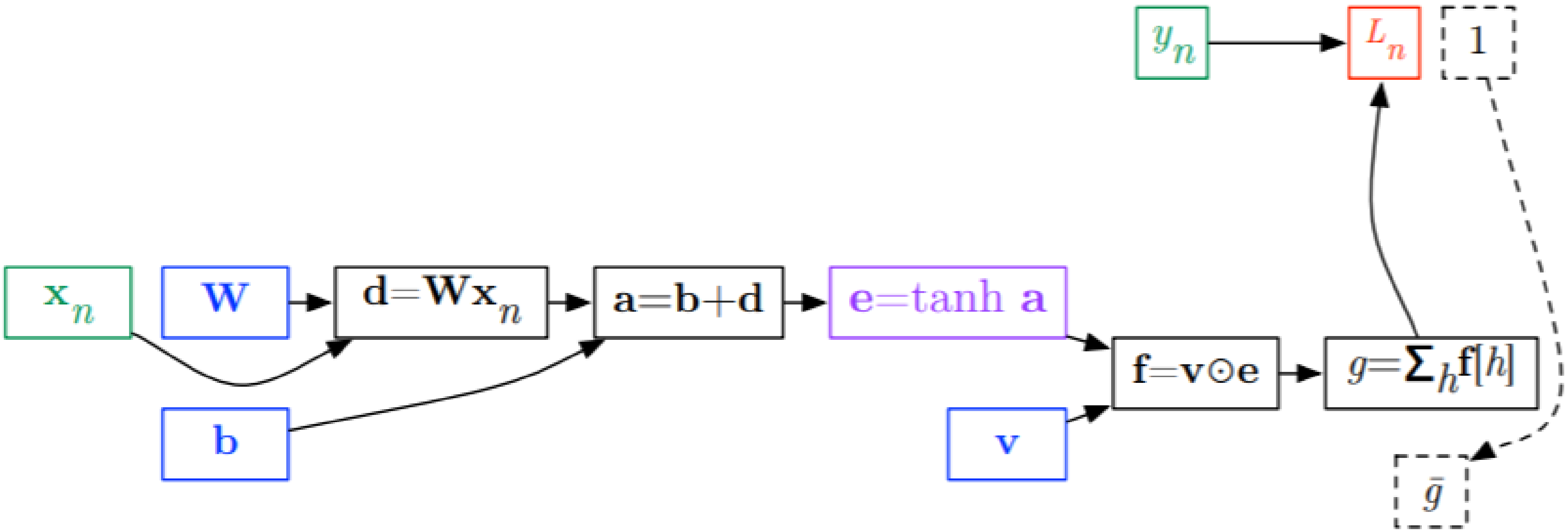
Intermediate nodes are de-anonymized, to make notation easier.

Backpropagation



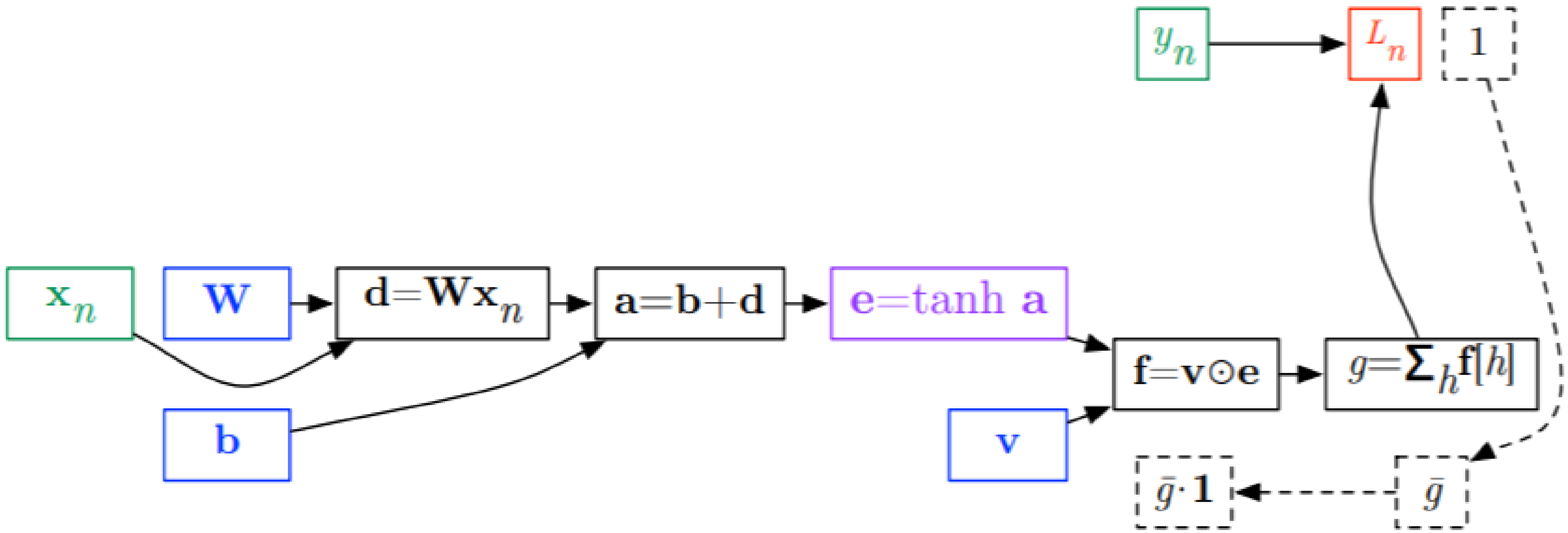
$$\frac{\partial L_n}{\partial L_n} = 1$$

Backpropagation



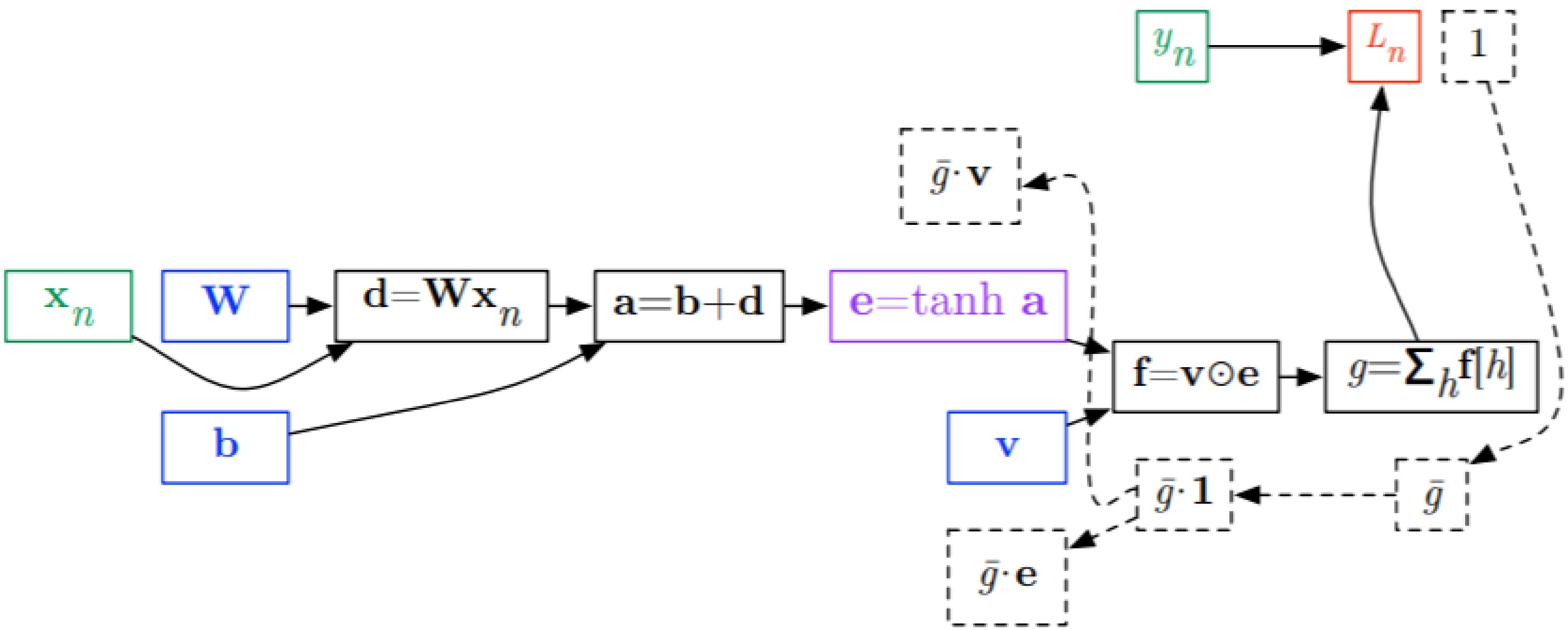
The form of $\bar{g}$ will be loss-function specific (e.g., $-2(y_n - g)$ for squared loss).

Backpropagation



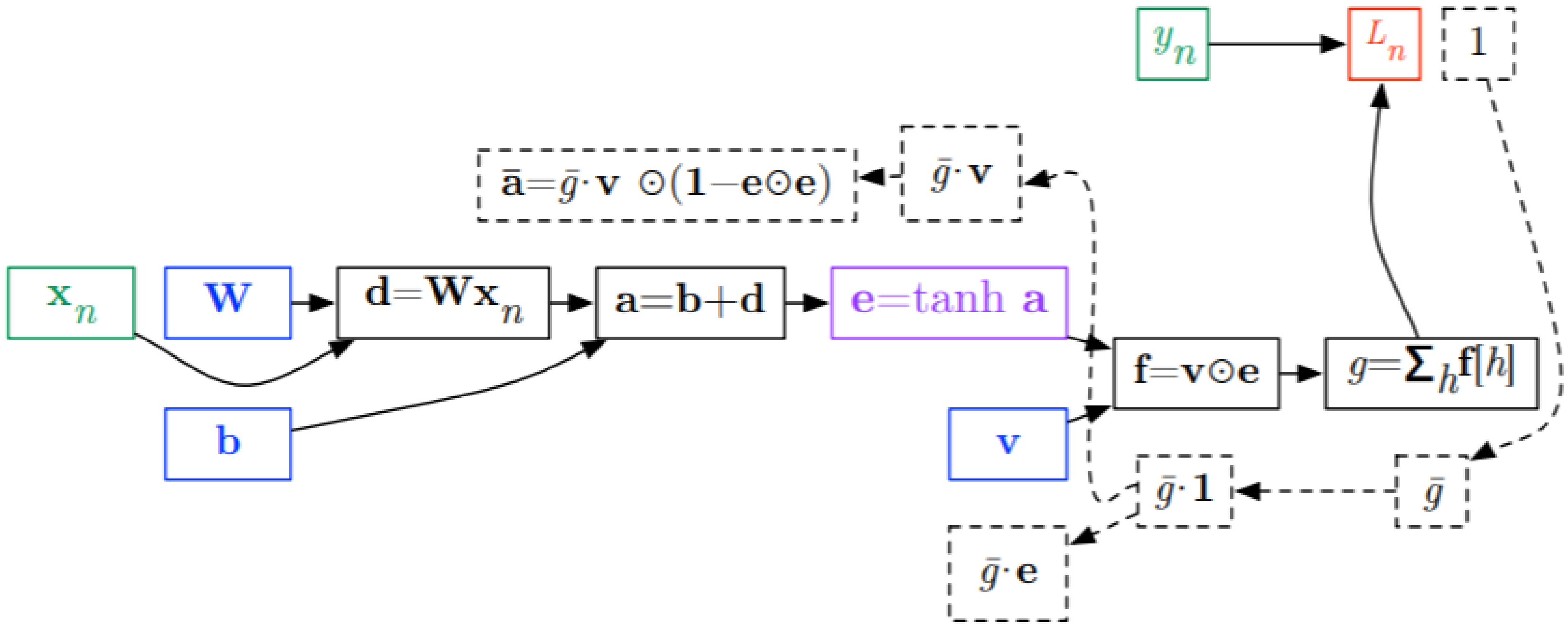
Sum.

Backpropagation



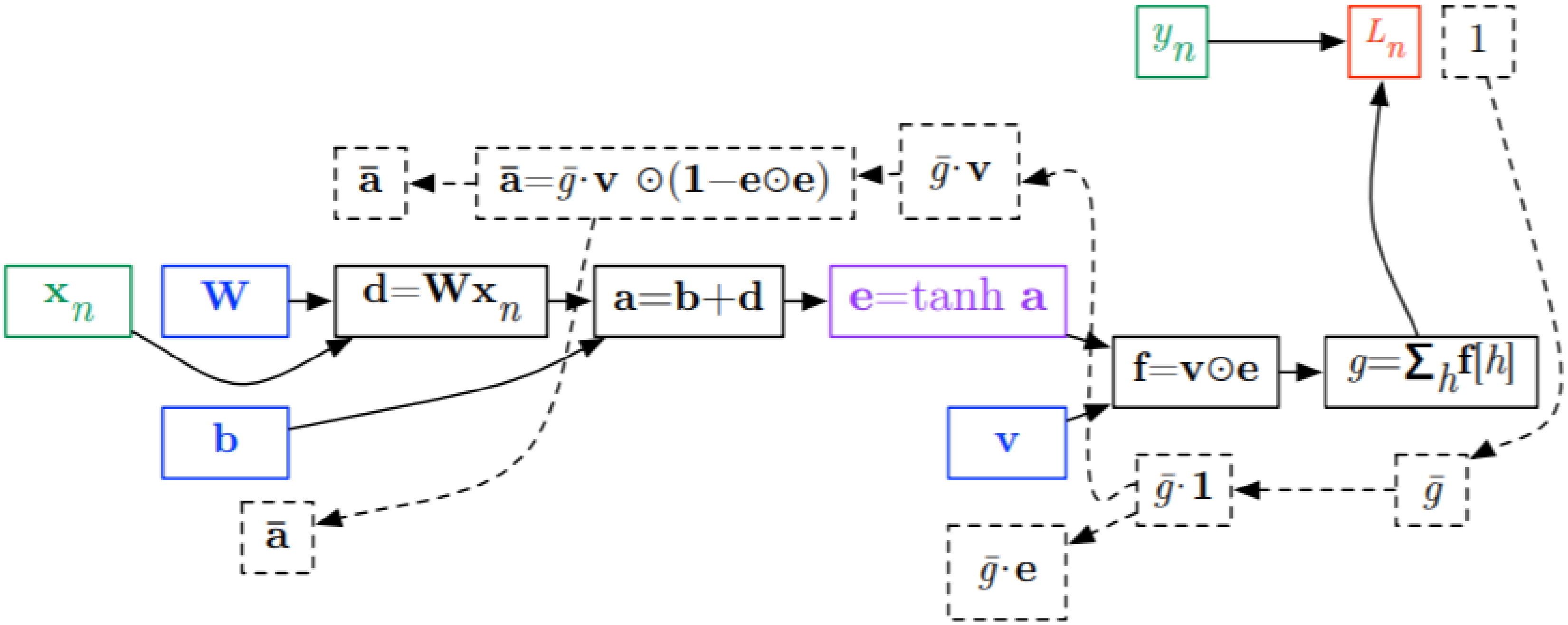
Product.

Backpropagation



Hyperbolic tangent.

Backpropagation



Sum.

Derivative w.r.t. Matrix Multiplication

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} w_{11}x_1 + w_{12}x_2 + w_{13}x_3 \\ w_{21}x_1 + w_{22}x_2 + w_{23}x_3 \\ w_{31}x_1 + w_{32}x_2 + w_{33}x_3 \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix}$$

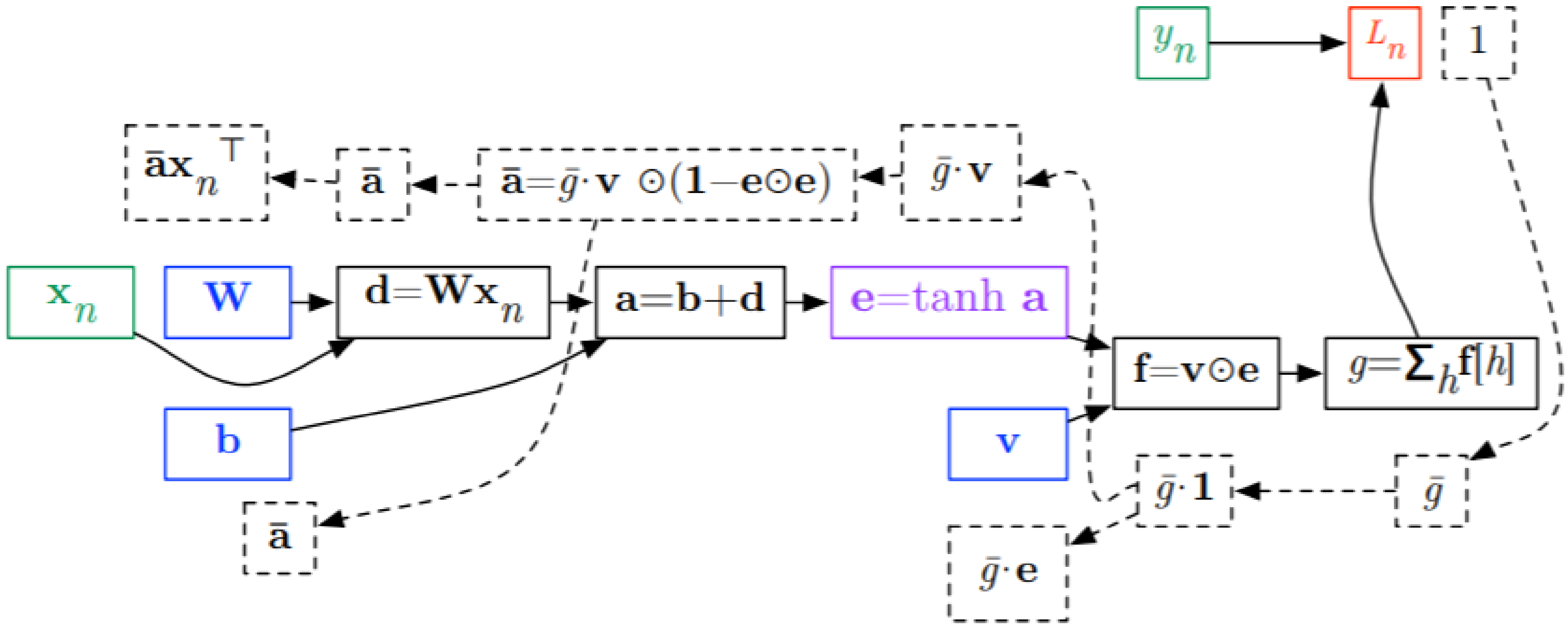
w_{ij} only influences d_i

$$\frac{\partial d_i}{\partial w_{ij}} = x_j$$

If we are given $\bar{d}$

$$\frac{\partial L}{\partial W} =$$

Backpropagation



Product.

Part II:
Why Deep?

Deeper is Better?

Layer X Size	Word Error Rate (%)
1 X 2k	24.2
2 X 2k	20.4
3 X 2k	18.4
4 X 2k	17.8
5 X 2k	17.2
7 X 2k	17.1

Not surprised, more parameters, better performance

Seide, Frank, Gang Li, and Dong Yu. "Conversational Speech Transcription Using Context-Dependent Deep Neural Networks." *Interspeech*. 2011.

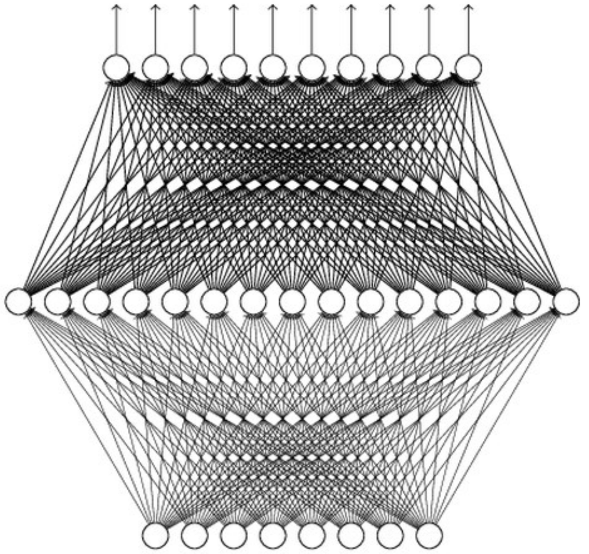
Universality Theorem

Any continuous function f

$$f : R^N \rightarrow R^{\mathrm{M}}$$

Can be realized by a network with one hidden layer

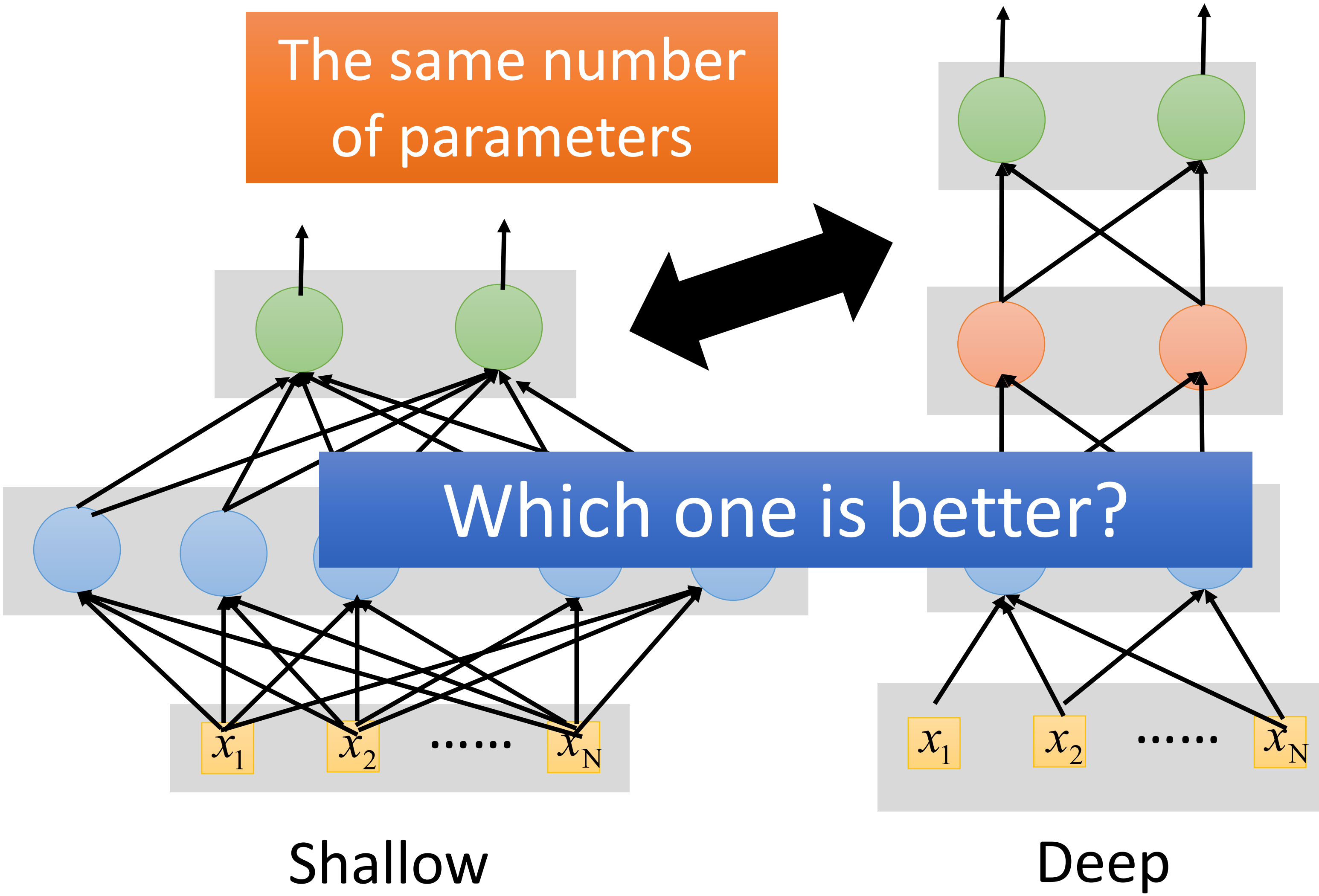
(given **enough** hidden neurons)



Reference for the reason: http://neuralnetworksanddeeplearning.com/chap4.html

Why "Deep" neural network not "Fat" neural network?

Fat + Short v.s. Thin + Tall



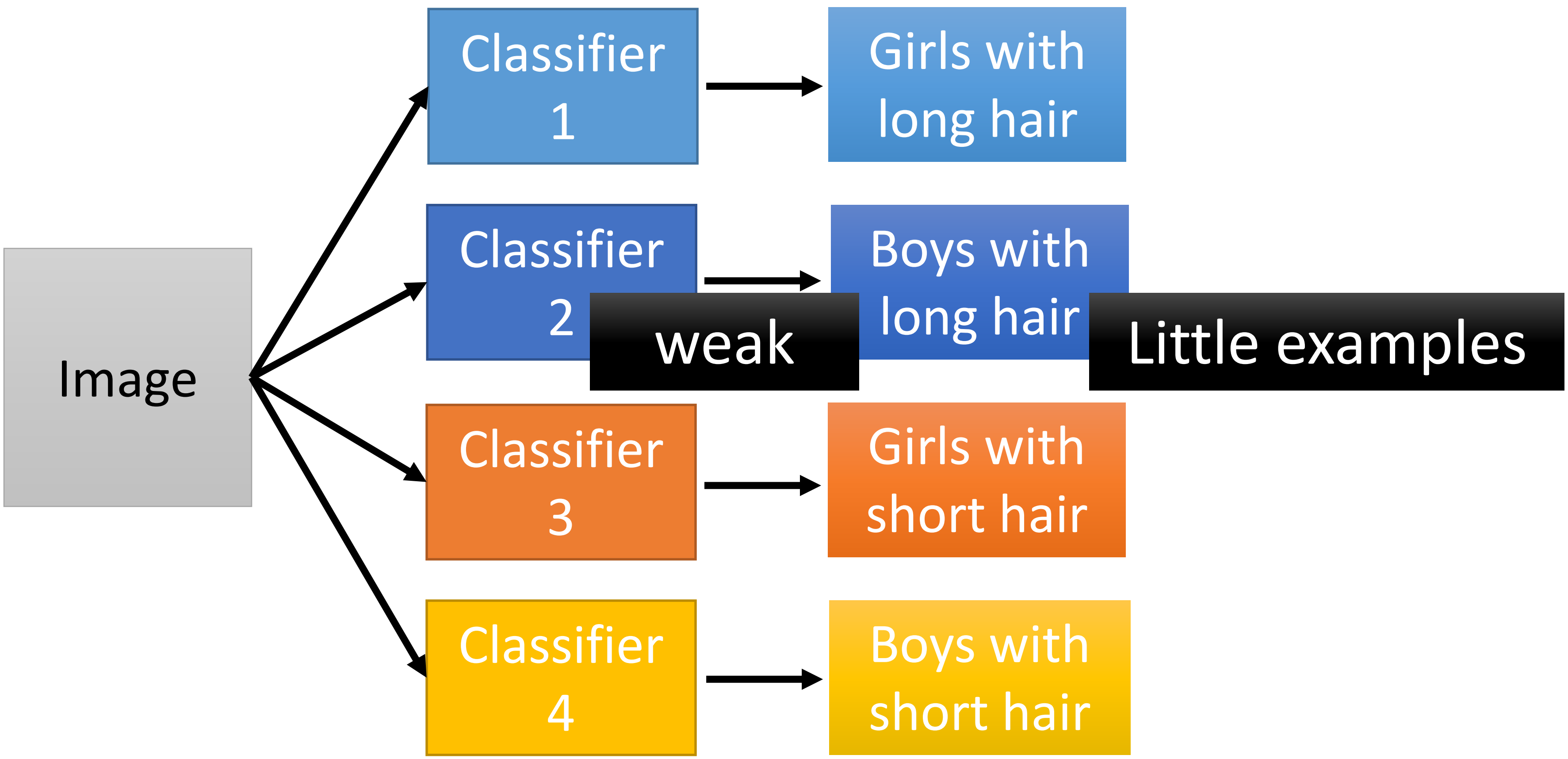
Fat + Short v.s. Thin + Tall

Layer X Size	Word Error Rate (%)	Layer X Size	Word Error Rate (%)
1 X 2k	24.2		
2 X 2k	20.4		
3 X 2k	18.4		
4 X 2k	17.8		
5 X 2k	17.2	1 X 3772	22.5
7 X 2k	17.1	1 X 4634	22.6
		1 X 16k	22.1

Seide, Frank, Gang Li, and Dong Yu. "Conversational Speech Transcription Using Context-Dependent Deep Neural Networks." *Interspeech*. 2011.

Why Deep?

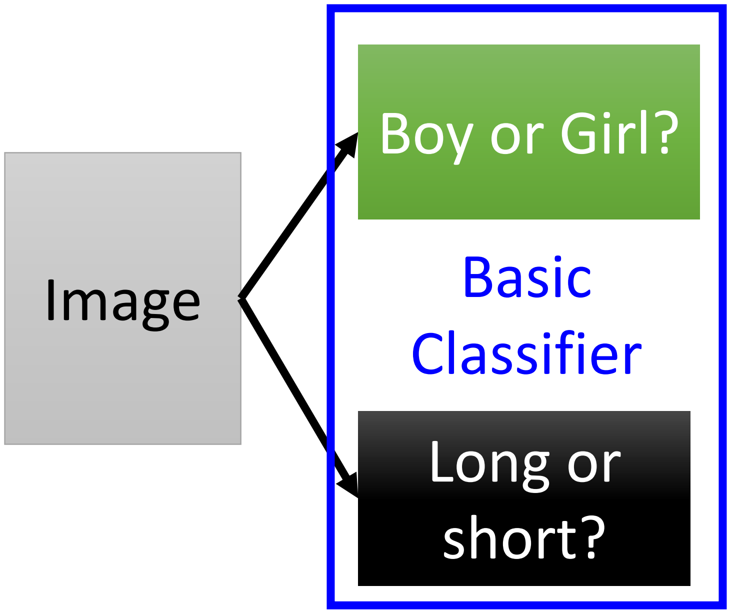
- Deep → Modularization



Why Deep?

Each basic classifier can have sufficient training examples.

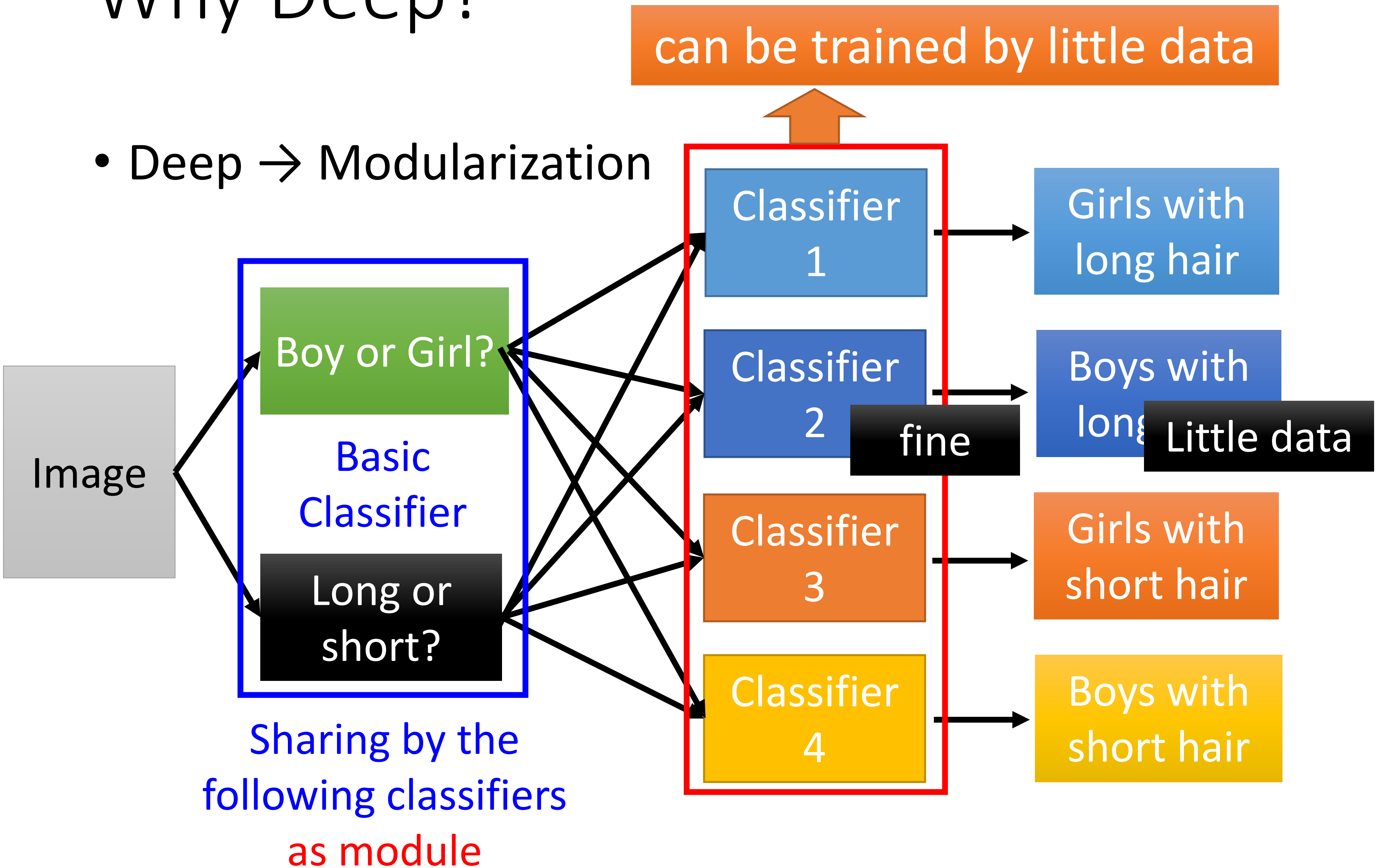
- Deep → Modularization



Classifiers for the attributes

Why Deep?

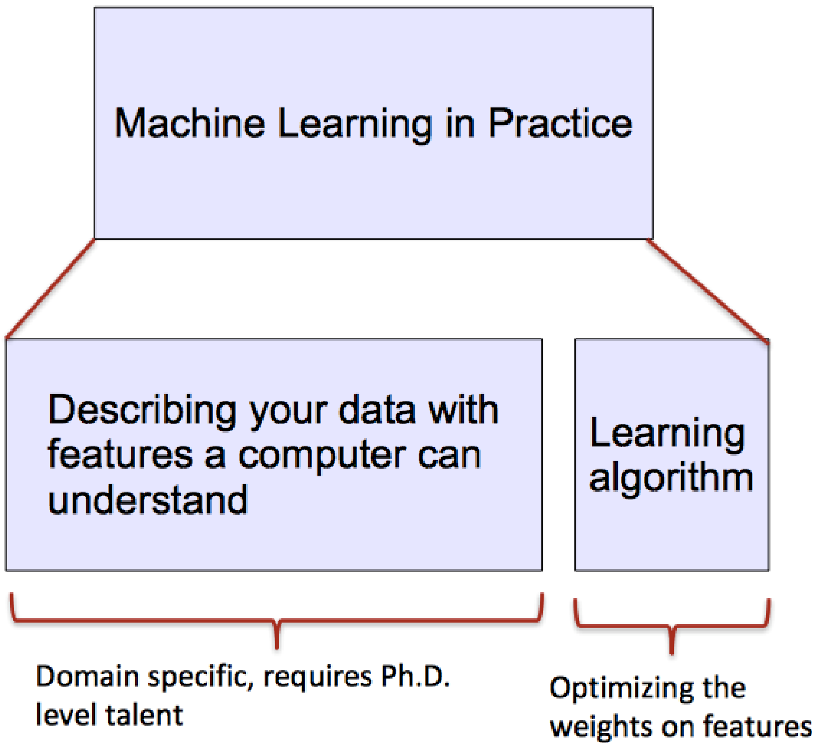
- Deep → Modularization



Traditional ML vs. Deep Learning

Most machine learning methods work well because of **human-designed representations** and **input features**

ML becomes just **optimizing weights** to best make a final prediction



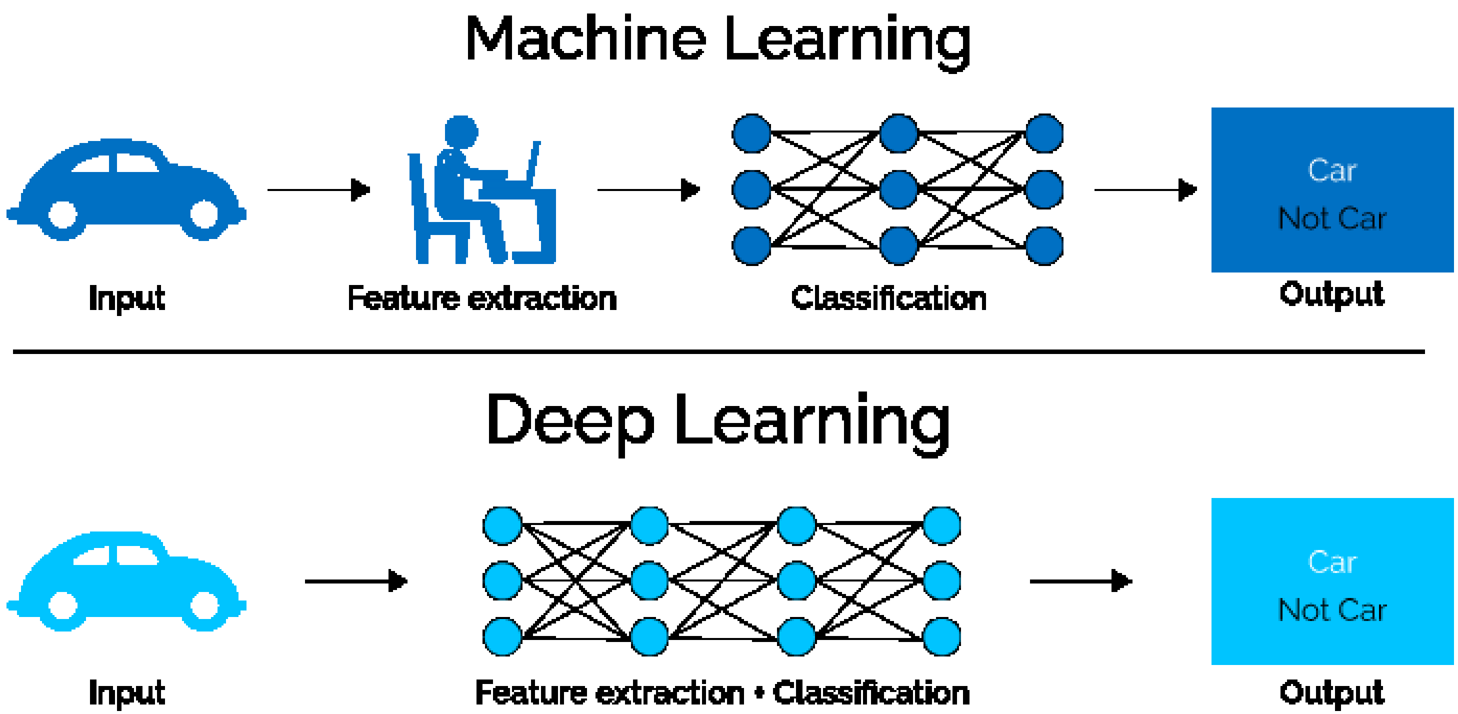
Feature	NER
Current Word	✓
Previous Word	✓
Next Word	✓
Current Word Character n-gram	all
Current POS Tag	✓
Surrounding POS Tag Sequence	✓
Current Word Shape	✓
Surrounding Word Shape Sequence	✓
Presence of Word in Left Window	size 4
Presence of Word in Right Window	size 4

What is Deep Learning (DL) ?

A machine learning subfield of learning **representations** of data. Exceptional effective at **learning patterns**.

Deep learning algorithms attempt to learn (multiple levels of) representation by using a **hierarchy of multiple layers**

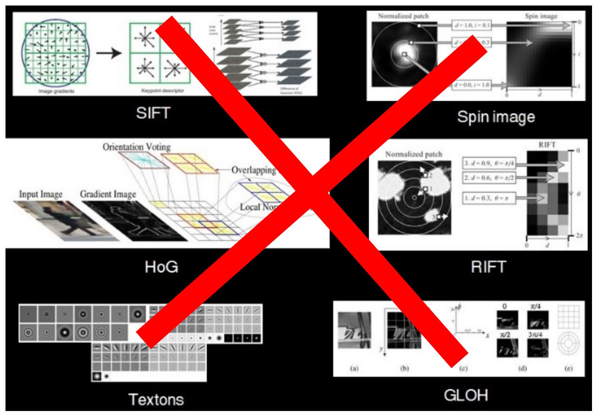
If you provide the system **tons of information**, it begins to understand it and respond in useful ways.

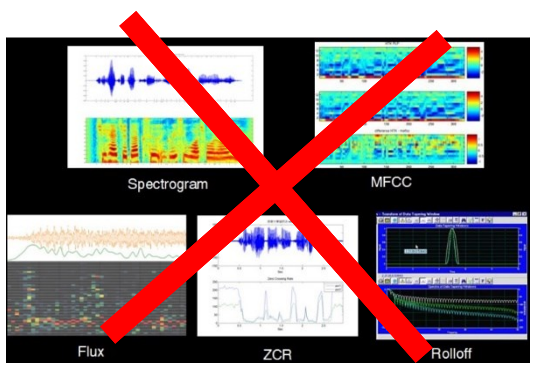


https://www.xenonstack.com/blog/static/public/uploads/media/machine-learning-vs-deep-learning.png

Part III:
Convolutional Neural Nets

Feature Learning

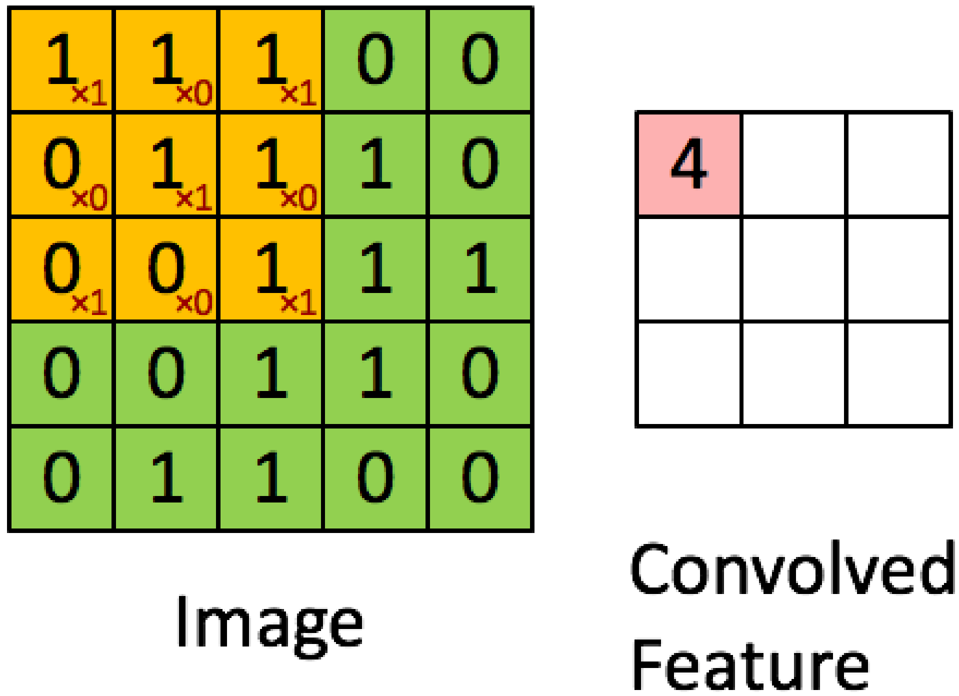




Learning algorithm

Feature representation

Convolution



1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

1	0	1
0	1	0
1	0	1

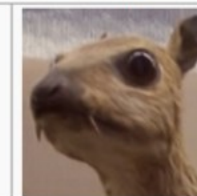
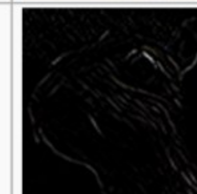
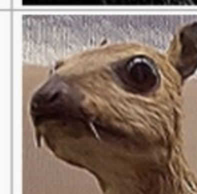
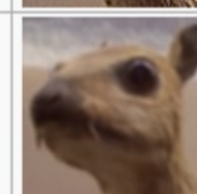
4	3	4
2	4	3
2	3	4

5x5 input. 3x3 filter/kernel/feature detector. 3x3

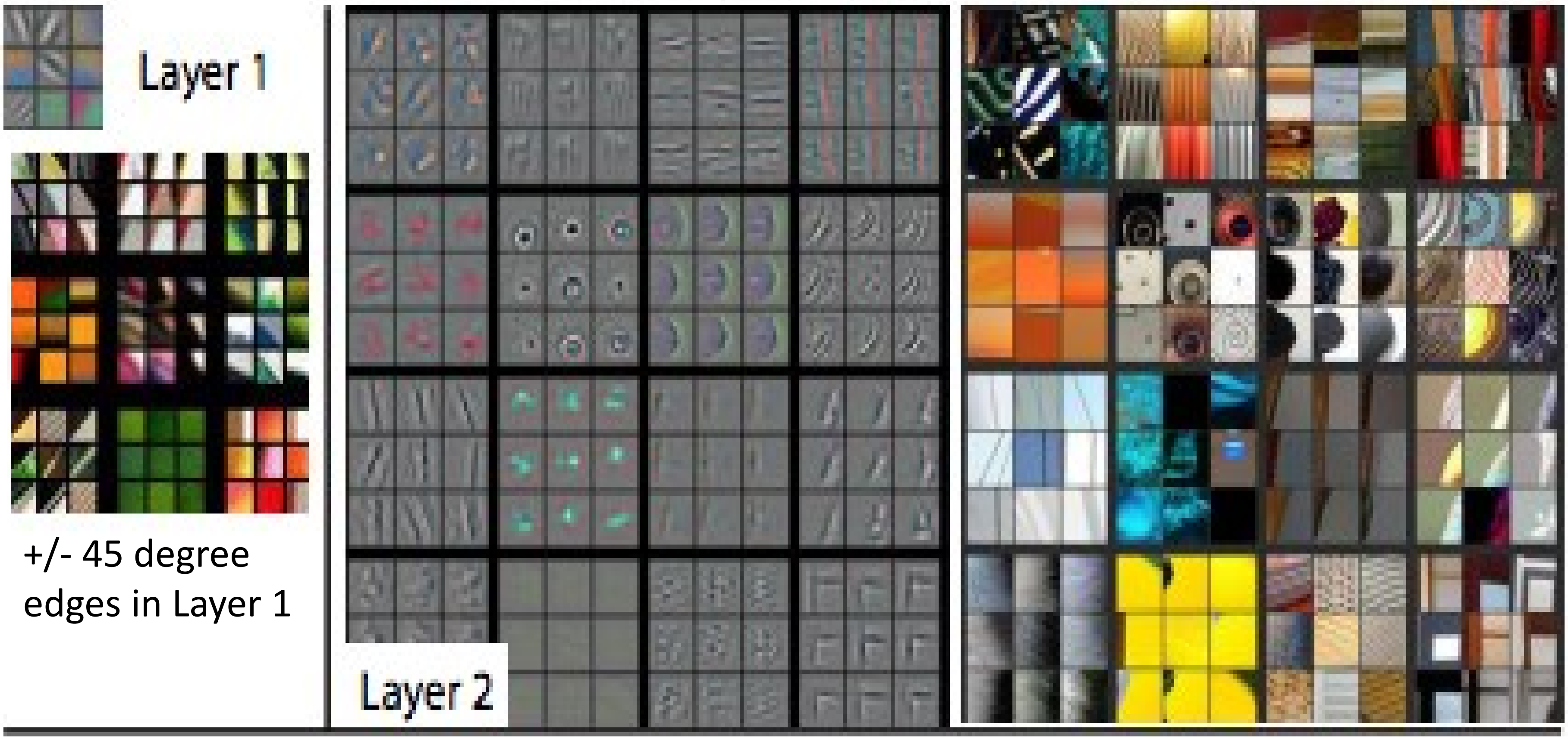
convolved feature/

Multiple filters

Original image

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	

Features at successive convolutional layers

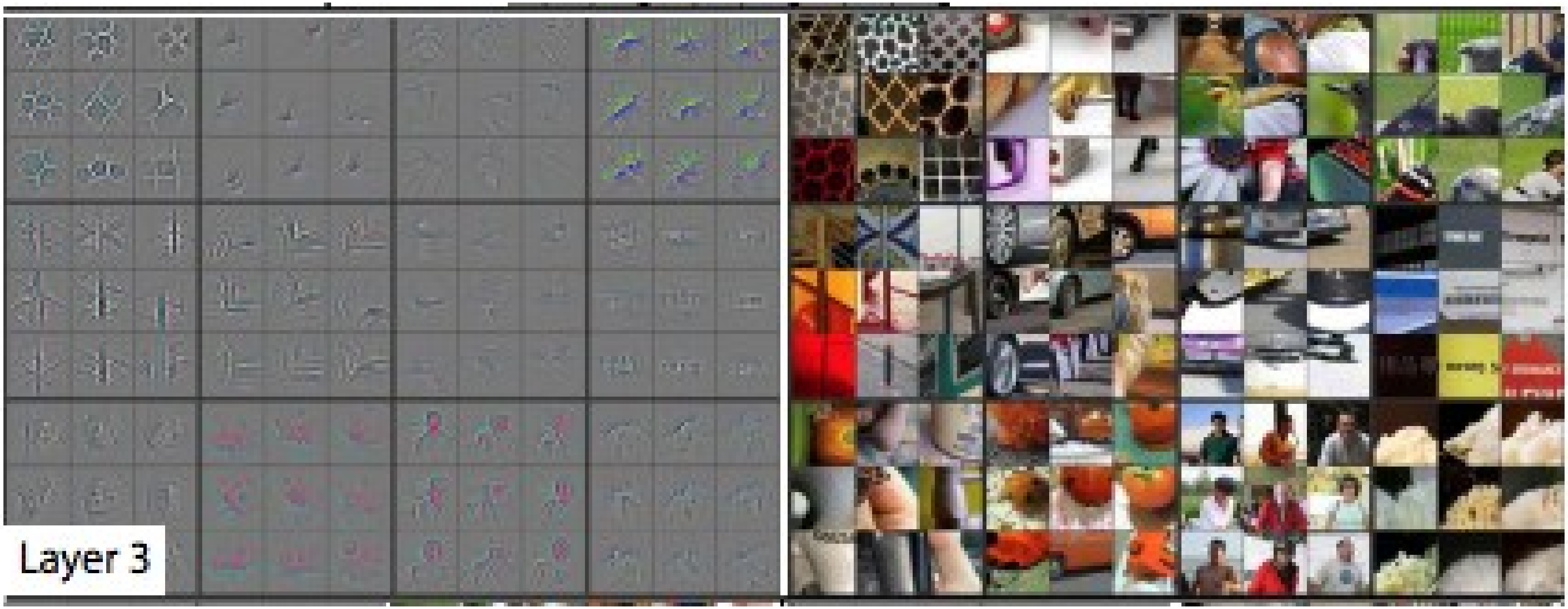


+/- 45 degree edges in Layer 1

Corners and other edge color conjunctions in Layer 2

Visualizing and Understanding Convolutional Networks,
Matthew D. Zeiler and Rob Fergus, ECCV 2014

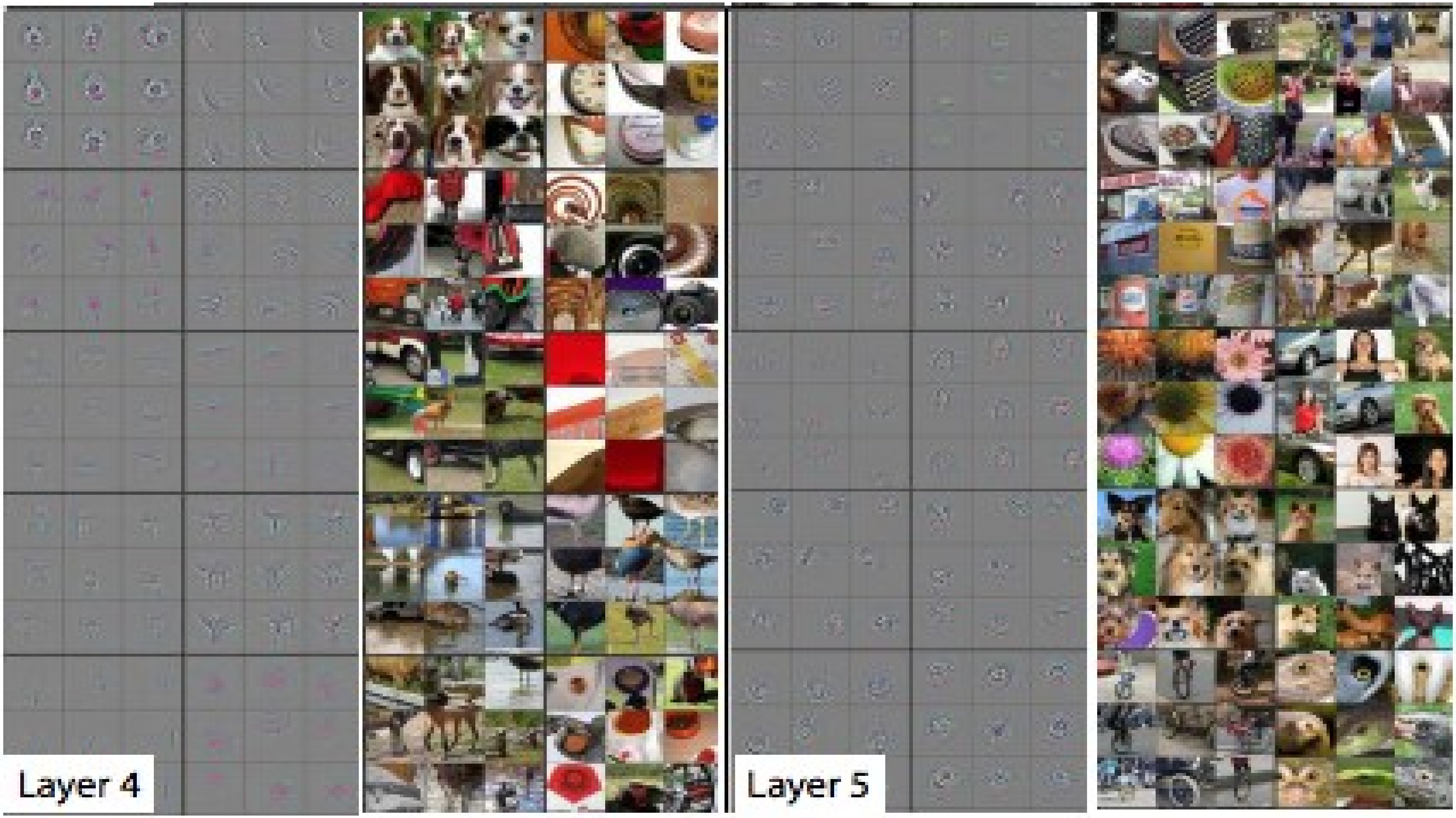
Features at successive convolutional layers



More complex invariances than Layer 2. Similar textures e.g. mesh patterns (R1C1); Text (R2C4).

Visualizing and Understanding Convolutional Networks,
Matthew D. Zeiler and Rob Fergus, ECCV 2014

Features at successive convolutional layers

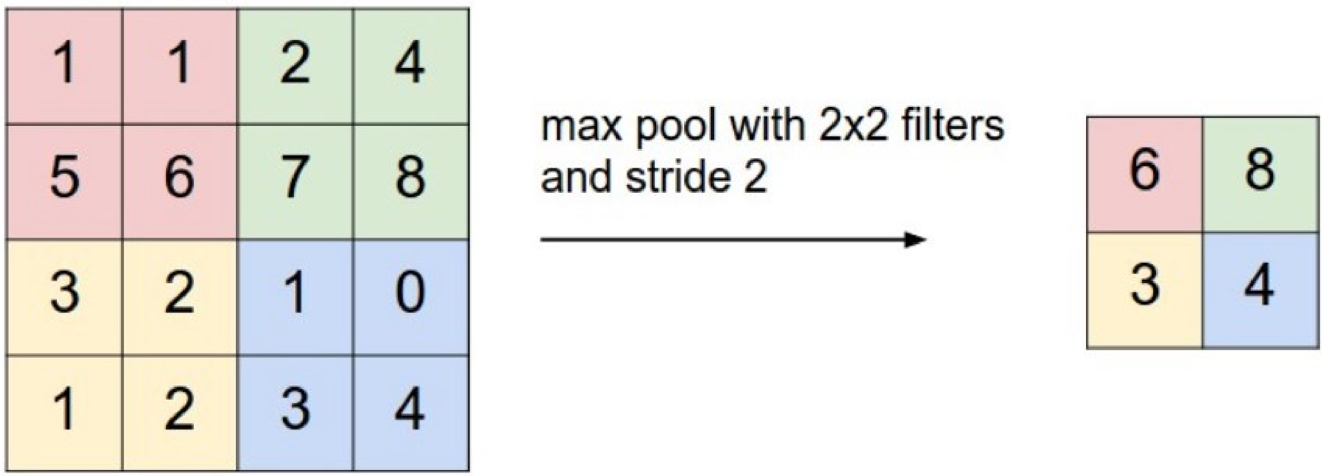


Significant variation, more class specific.
Dog faces (R1C1); Bird legs (R4C2).

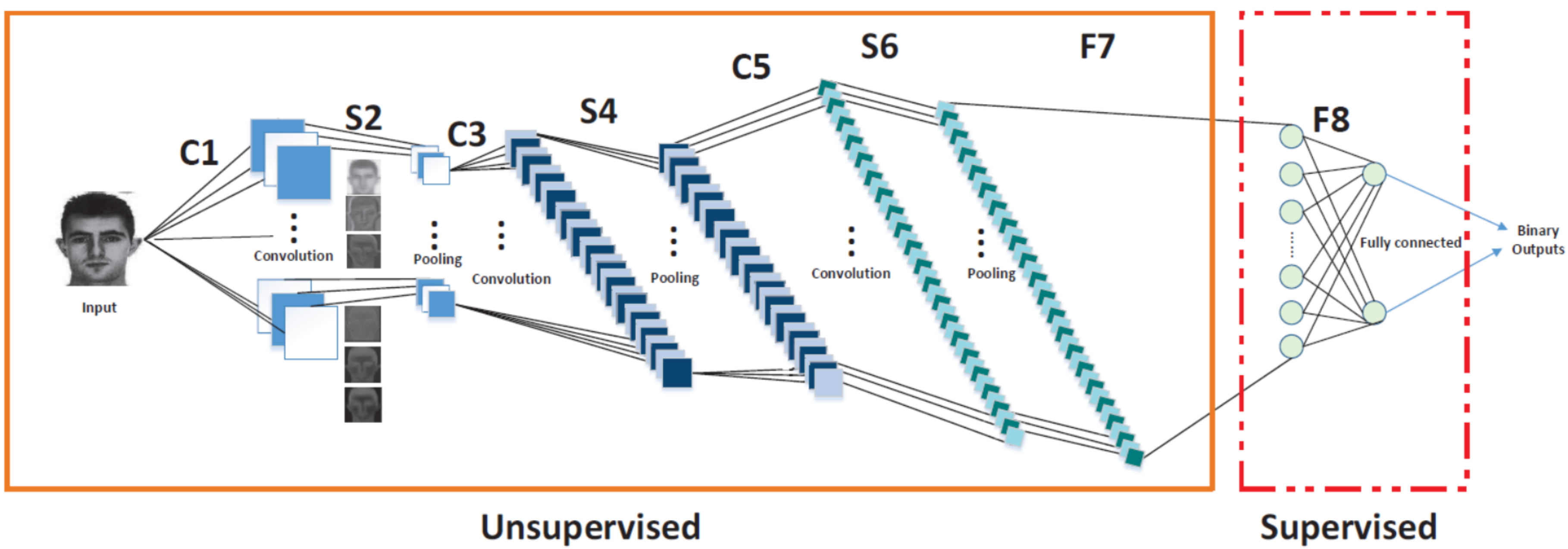
Entire objects with significant pose variation.
Keyboards (R1C1); dogs (R4).

Visualizing and Understanding Convolutional Networks,
Matthew D. Zeiler and Rob Fergus, ECCV 2014

Max pooling


1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

CNN architecture



Object Recognition

CIFAR
CANADIAN INSTITUTE for ADVANCED RESEARCH

IMAGENET

Network	Error	Layers
AlexNet	16.0%	8
ZFNet	11.2%	8
VGGNet	7.3%	19
GoogLeNet	6.7%	22
MS ResNet	3.6%	152!!