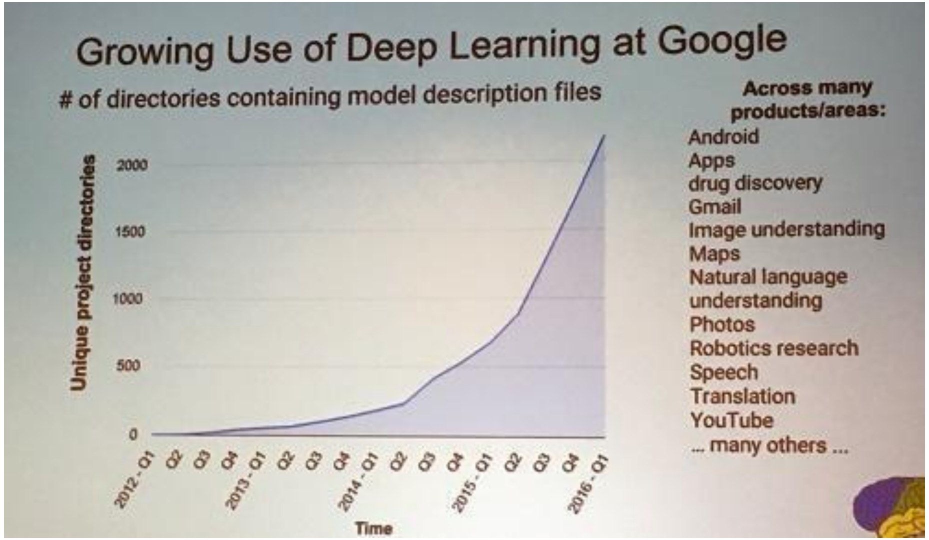
Neural Networks

Some of the contents are adapted from A. Ng, S. Kim, F. Li, and H. Lee

Deep learning attracts lots of attention.

- I believe you have seen lots of exciting results before.

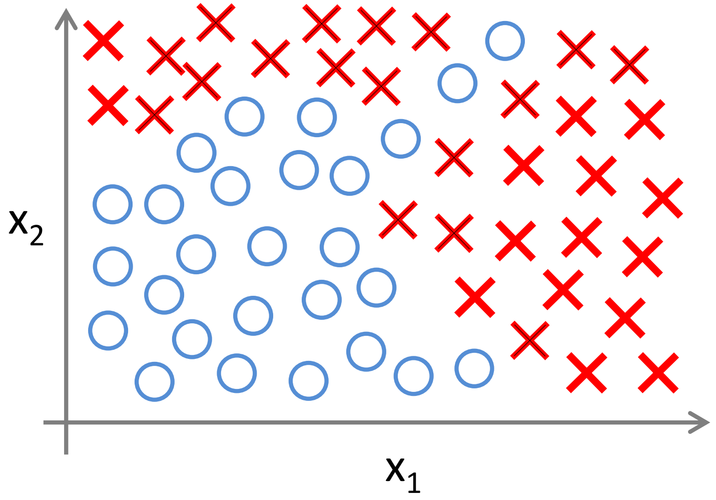


Deep learning trends at Google. Source: SIGMOD/Jeff Dean

History of Deep Learning

- 1958: Perceptron (linear model)
- 1969: Perceptron has limitation
- 1980s: Multi-layer perceptron
  - Do not have significant difference from DNN today
- 1986: Backpropagation
  - Usually more than 3 hidden layers is not helpful
- 1989: 1 hidden layer is “good enough”, why deep?
- 2006: RBM initialization (breakthrough)
- 2009: GPU
- 2011: Start to be popular in speech recognition
- 2012: win ILSVRC image competition

Non-linear Classification

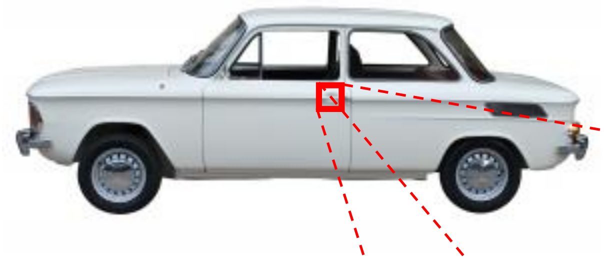


$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^3 x_2 + \theta_6 x_1 x_2^2 + \ldots)$$

x_1 = size
x_2 = # bedrooms
x_3 = # floors
x_4 = age
$\ldots$
x_{100}

What is this?

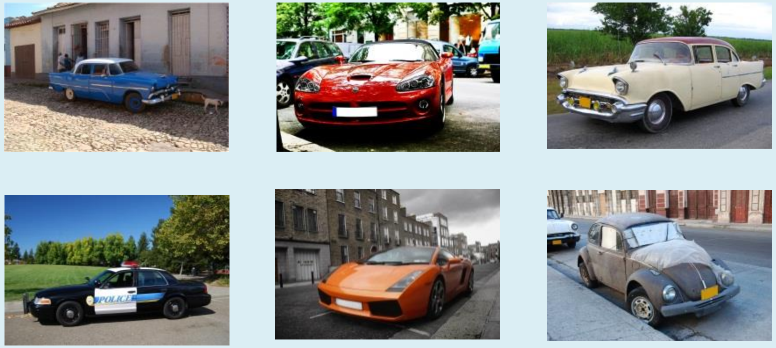
You see this:



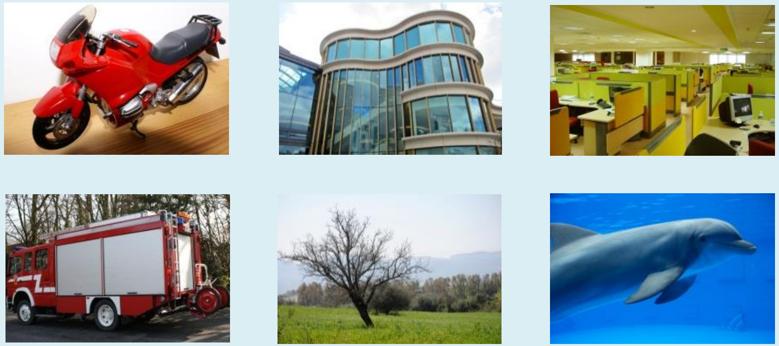
But the camera sees this:

194	210	201	212	199	213	215	195	178	158	182	209
180	189	190	221	209	205	191	167	147	115	129	163
114	126	140	188	176	165	152	140	170	106	78	88
87	103	115	154	143	142	149	153	173	101	57	57
102	112	106	131	122	138	152	147	128	84	58	66
94	95	79	104	105	124	129	113	107	87	69	67
68	71	69	98	89	92	98	95	89	88	76	67
41	56	68	99	63	45	60	82	58	76	75	65
20	43	69	75	56	41	51	73	55	70	63	44
50	50	57	69	75	75	73	74	53	68	59	37
72	59	53	66	84	92	84	74	57	72	63	42
67	61	58	65	75	78	76	73	59	75	69	50

Computer Vision: Car detection



Cars



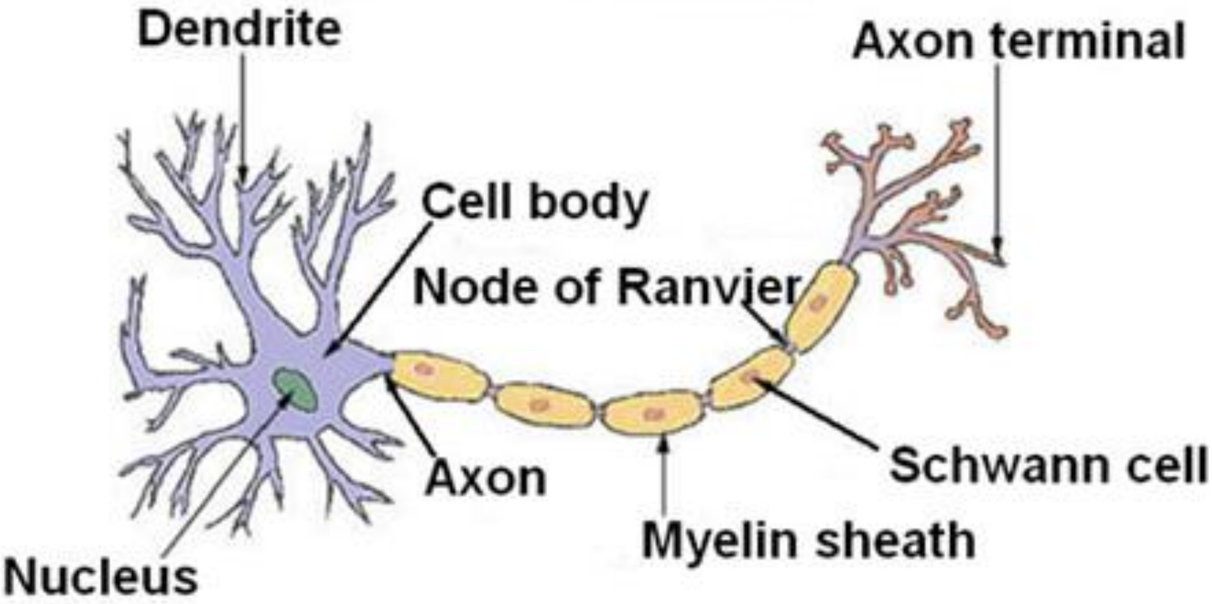
Not a car

Testing: 

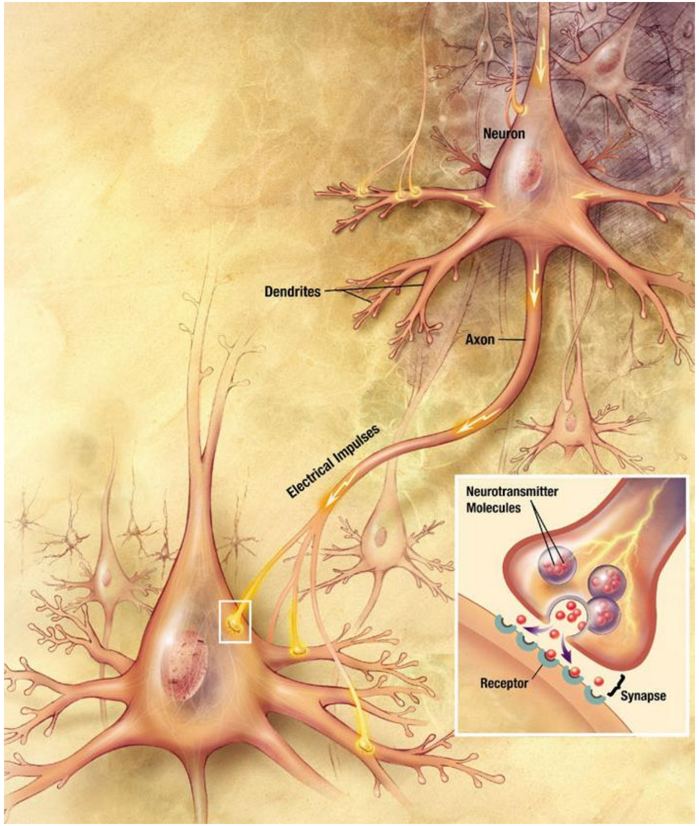
Neural Networks

Origins: Algorithms that try to mimic the brain.
Was very widely used in 80s and early 90s; popularity diminished in late 90s.
Recent resurgence: State-of-the-art technique for many applications

Neuron in the brain

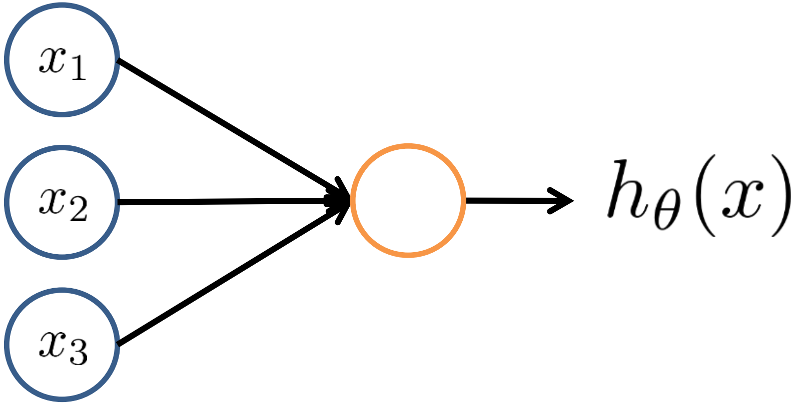


Neurons in the brain



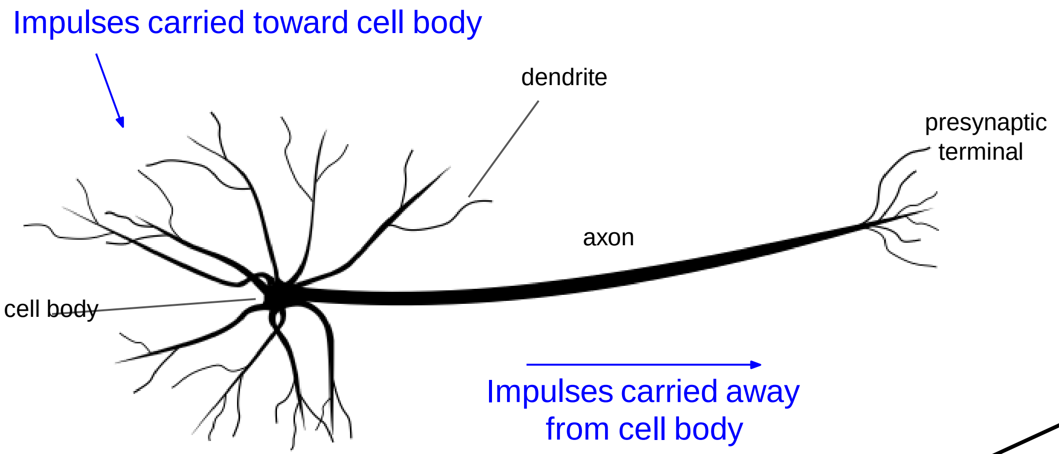
[Credit: US National Institutes of Health, National Institute on Aging]

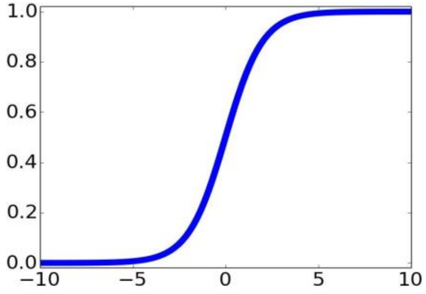
Neuron model: Logistic unit



$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

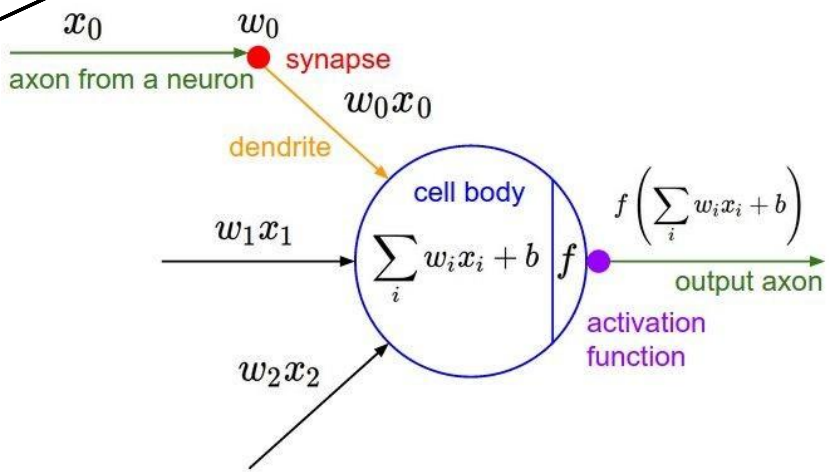
Sigmoid (logistic) activation function.



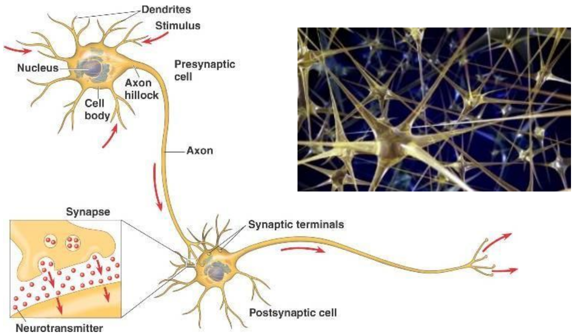


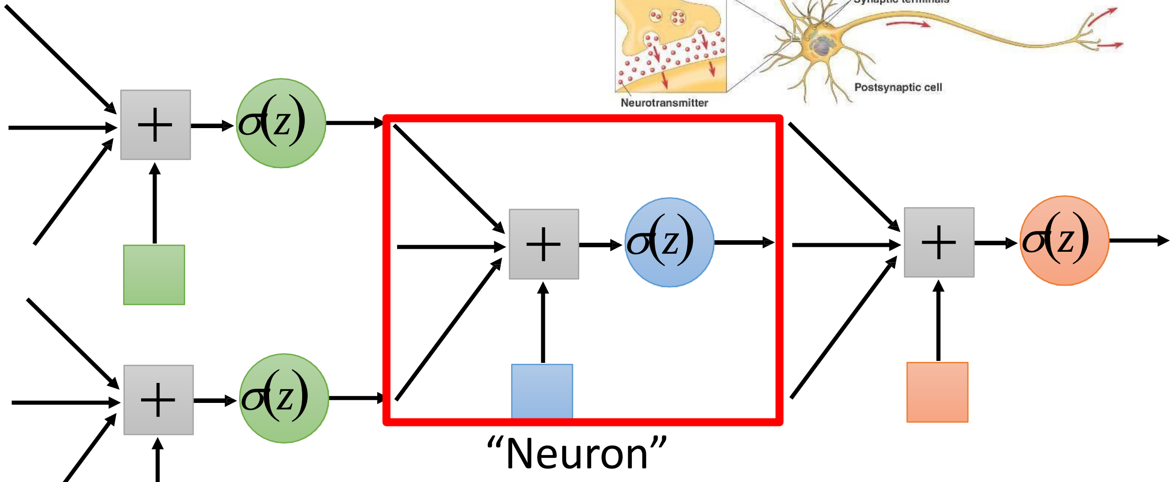
sigmoid activation function

$$\frac{1}{1+e^{-x}}$$



Neural Network



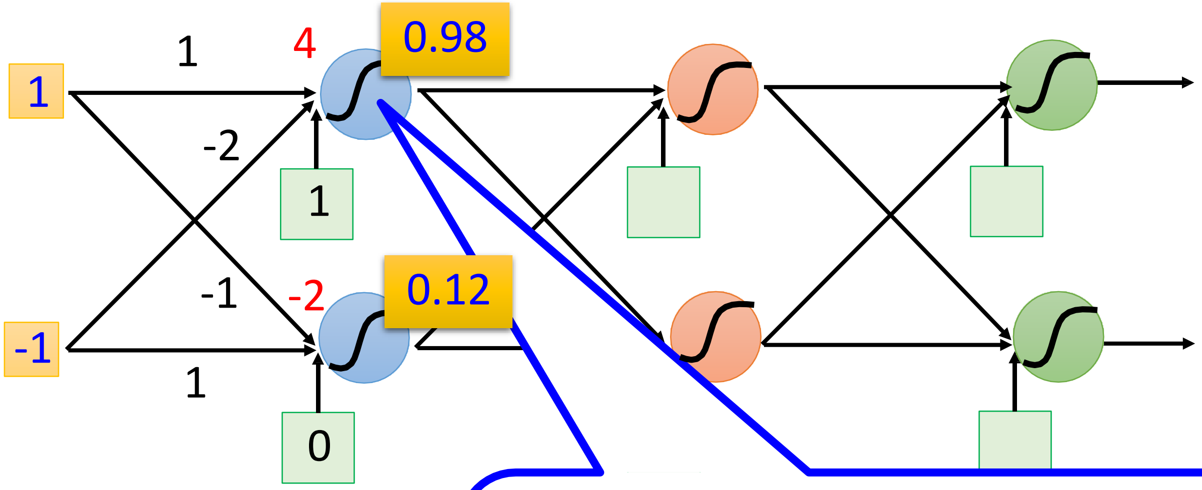


<u>Neural Network</u>

Different connection leads to different network structures

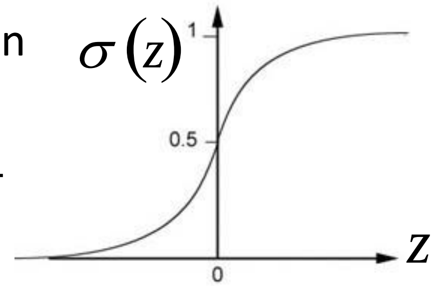
Network parameter θ: all the weights and biases in the "neurons"

Fully Connect Feedforward Network

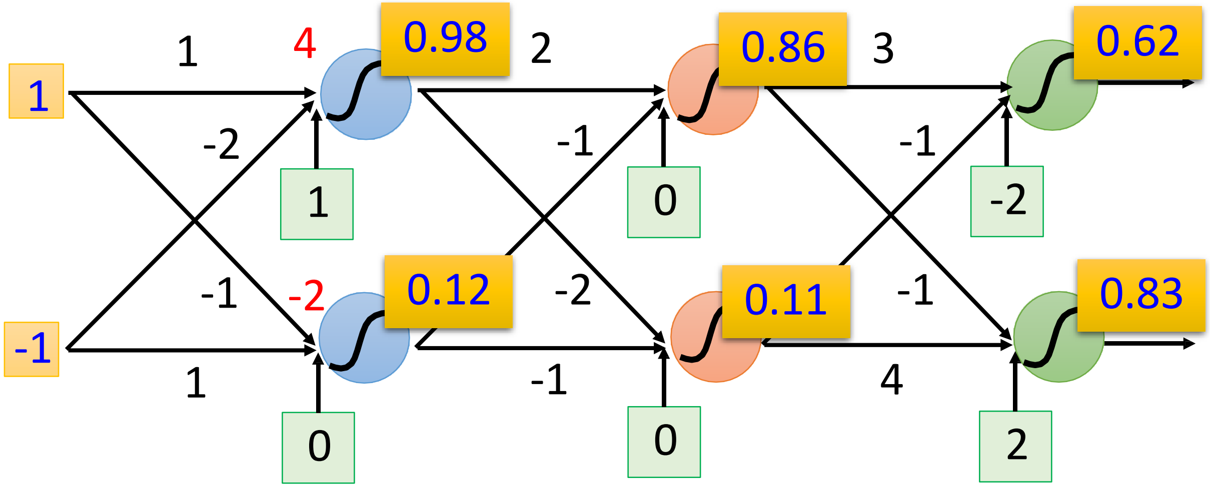


Sigmoid Function

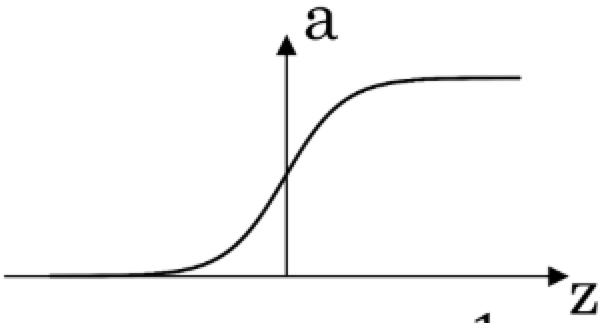
$$\sigma(z)=\frac{1}{1+e^{-z}}$$



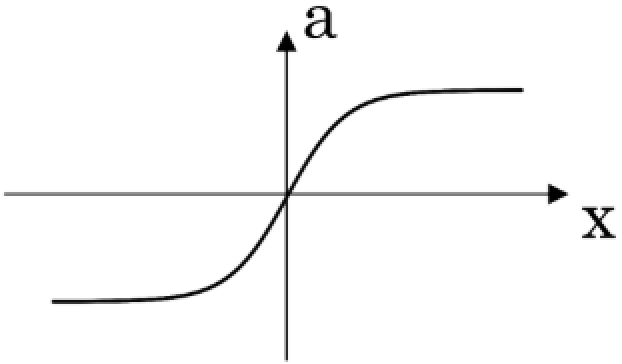
Fully Connect Feedforward Network



Activation Functions

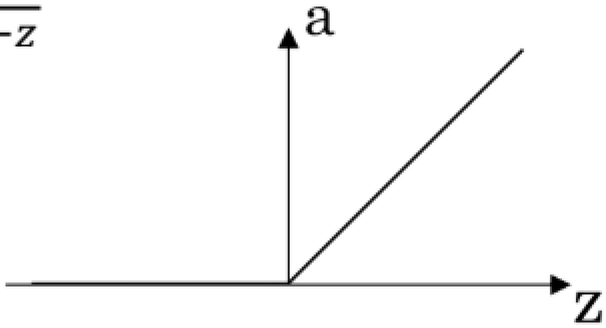


sigmoid: $a = \frac{1}{1 + e^{-z}}$



$g(z) = \tanh(z)$

$$y = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$

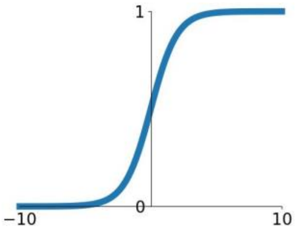


ReLU

Activation functions

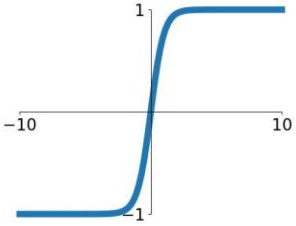
Sigmoid

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



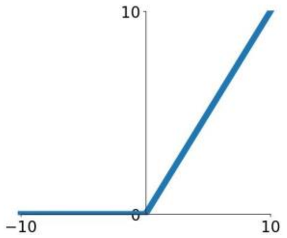
tanh

$$\tanh(x)$$



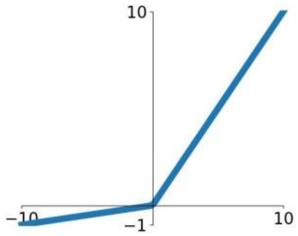
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

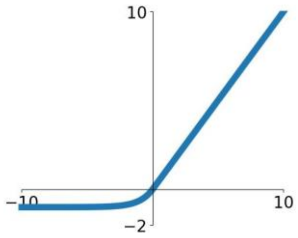


Maxout

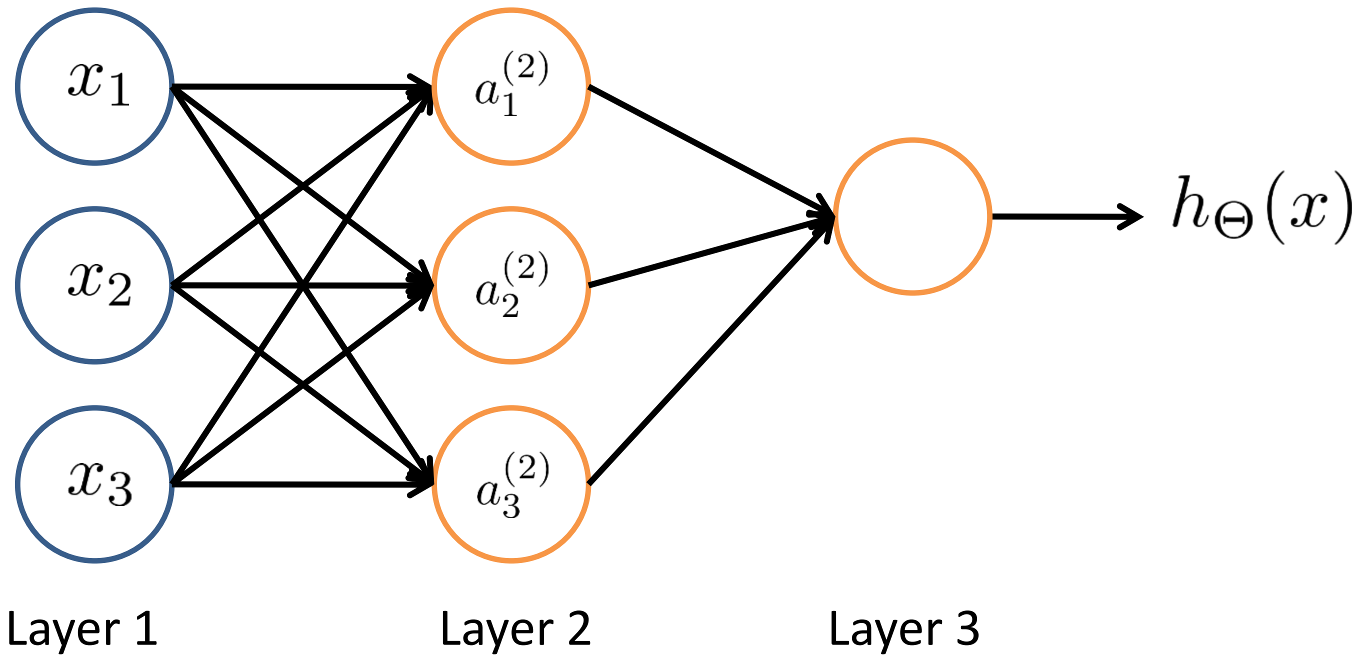
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

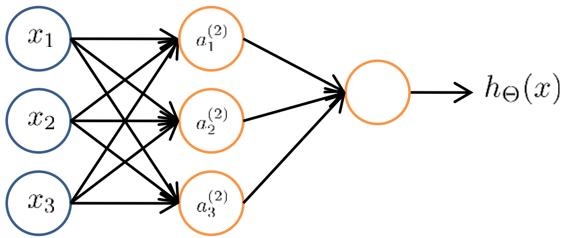
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Neural Network



Neural Network



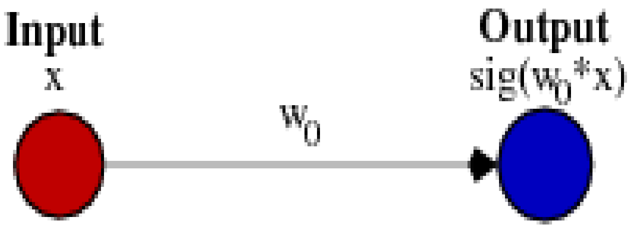
$a_i^{(j)}$ = "activation" of unit i in layer j

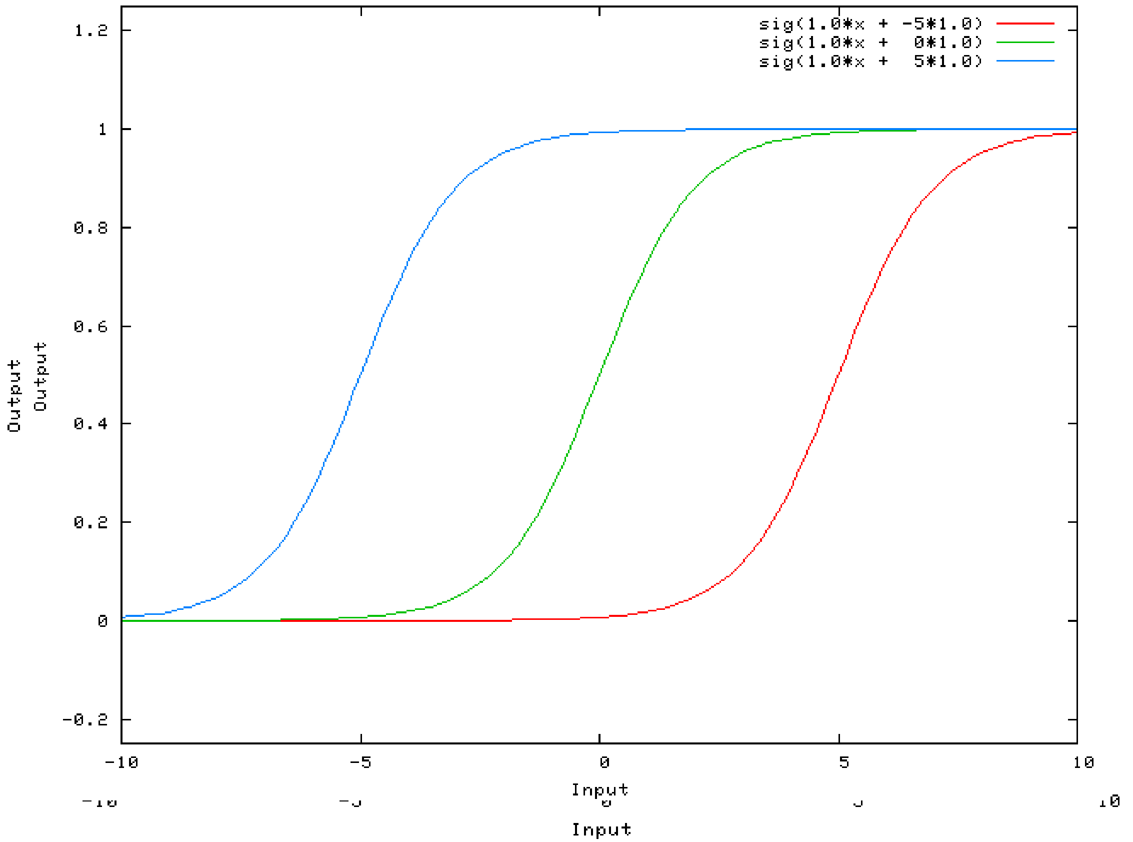
$\Theta^{(j)}$ = matrix of weights controlling function mapping from layer j to layer $j+1$

$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

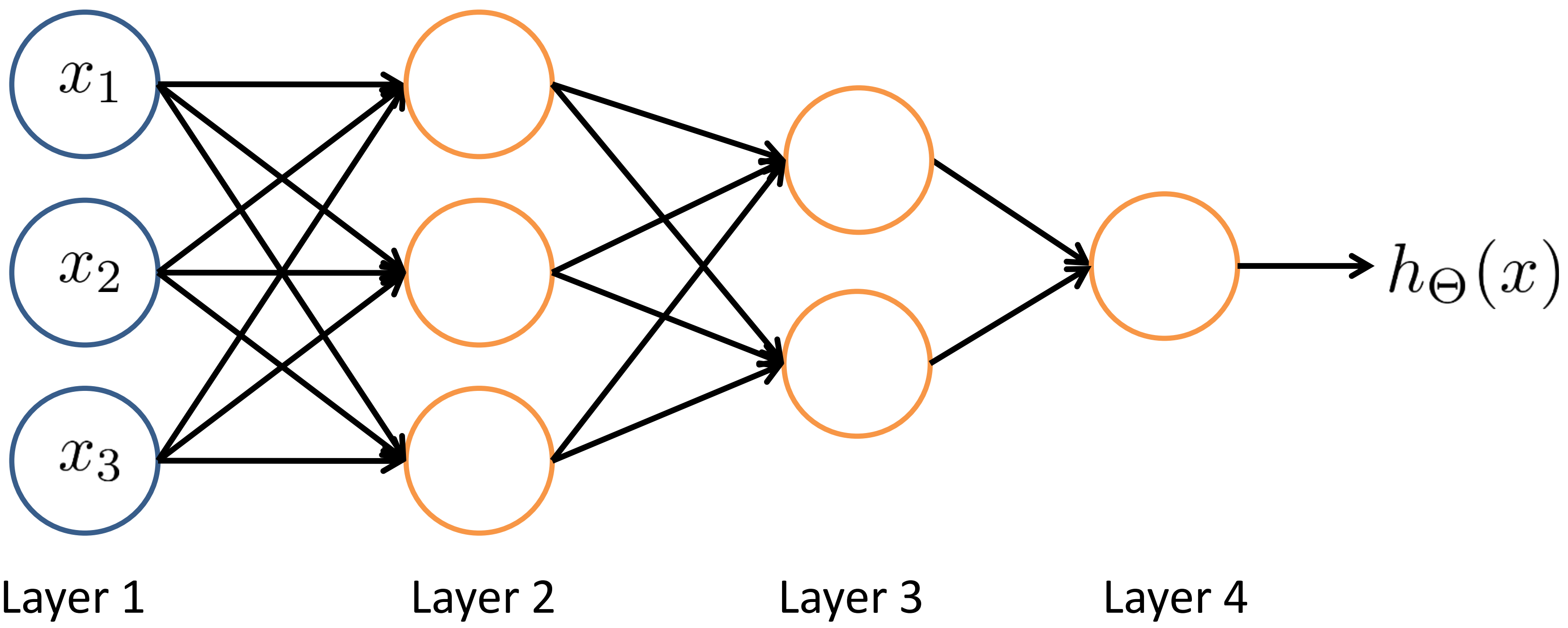
If network has s_j units in layer j, s_{j+1} units in layer $j+1$, then $\Theta^{(j)}$ will be of dimension $s_{j+1} \times (s_j + 1)$.

Weights and Bias





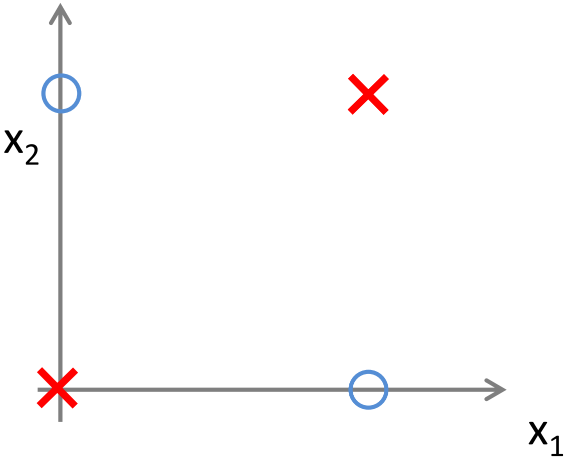
network architectures

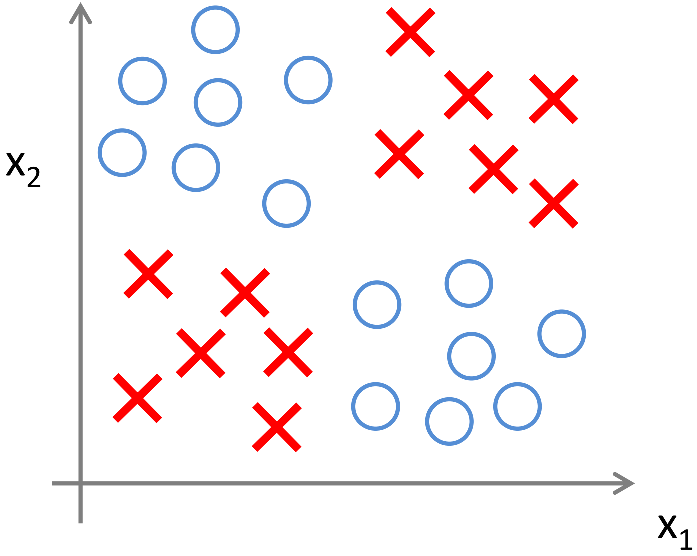


Non-linear classification example: XOR/XNOR

x1, x2 Features

x_1, x_2 are binary (0 or 1).



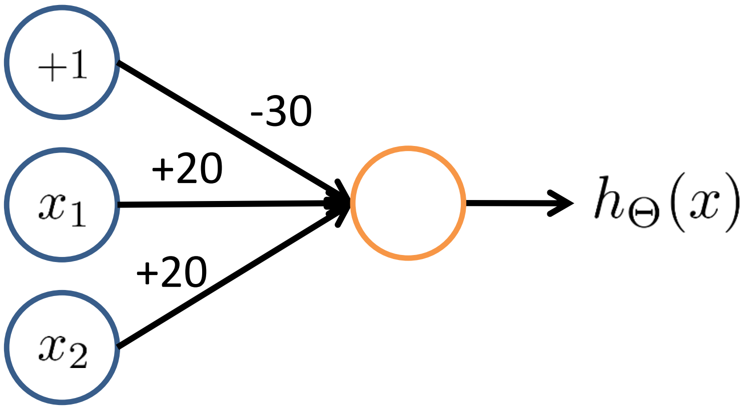


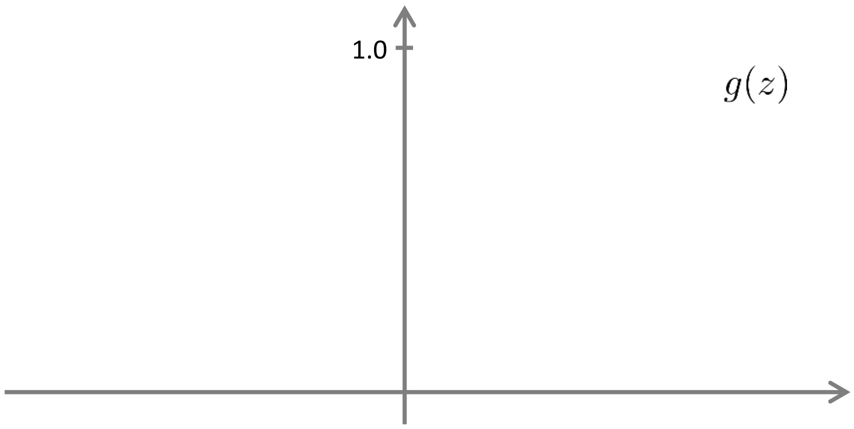
$$
\begin{aligned}
y = {} & x_1 \text{ XOR } x_2 \\
& x_1 \text{ XNOR } x_2 \\
& \text{NOT } (x_1 \text{ XOR } x_2)
\end{aligned}
$$

Simple example: AND

$x_1, x_2 \in \{0, 1\}$

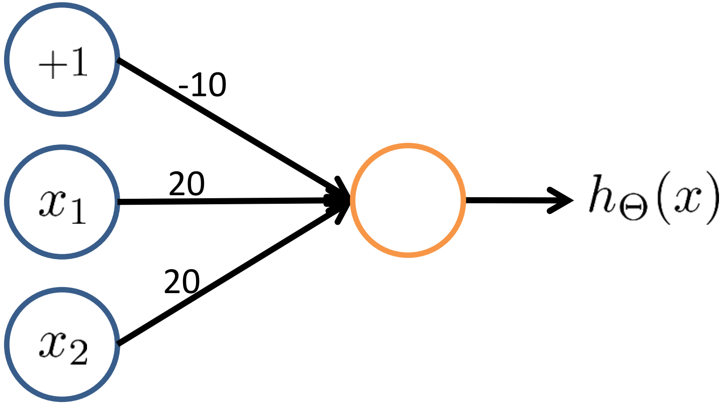
$y = x_1 \text{ AND } x_2$





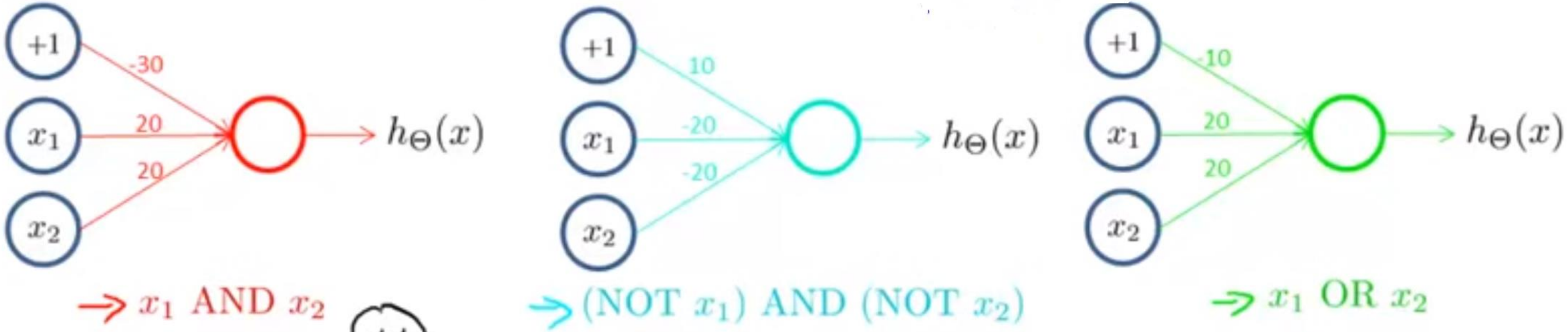
x_1	x_2	$h_\Theta(x)$
0	0	
0	1	
1	0	
1	1	

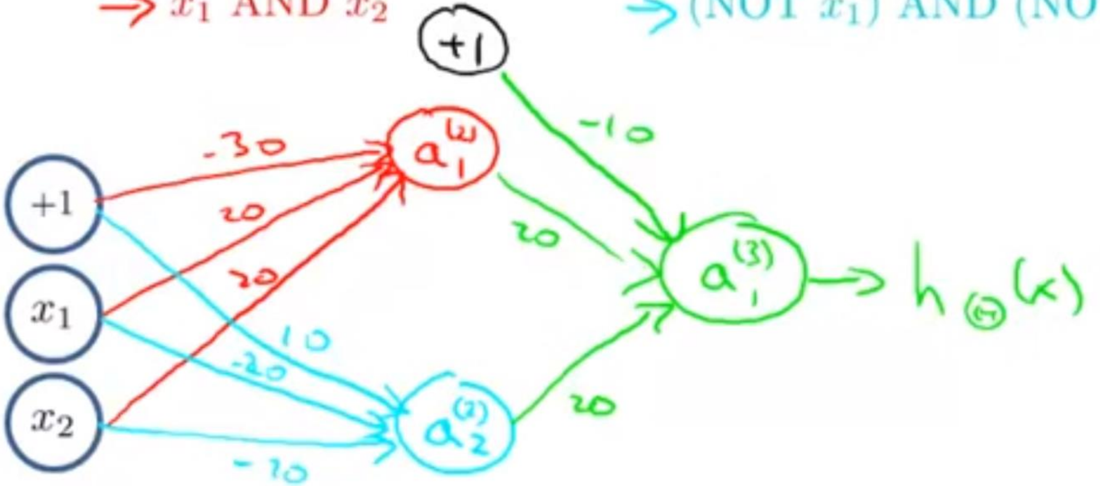
Example: OR function



x_1	x_2	$h_\Theta(x)$
0	0	
0	1	
1	0	
1	1	

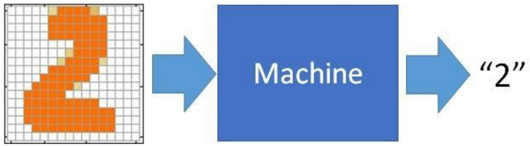
Putting it together: x_1 XNOR x_2



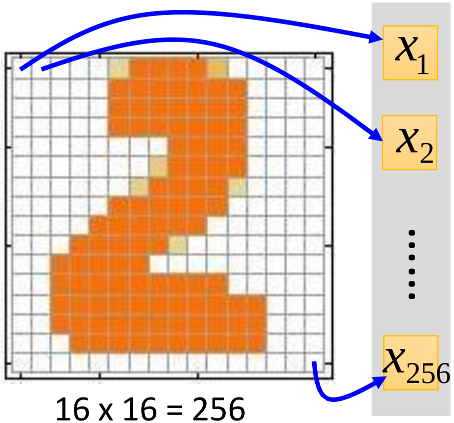


x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\Theta(x)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

Example Application



Input

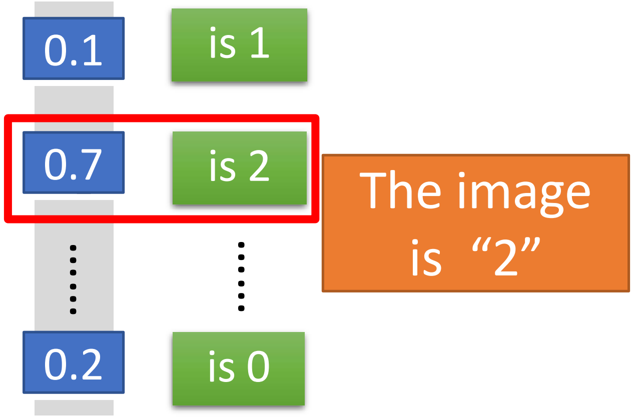


16 x 16 = 256

Ink → 1

No ink → 0

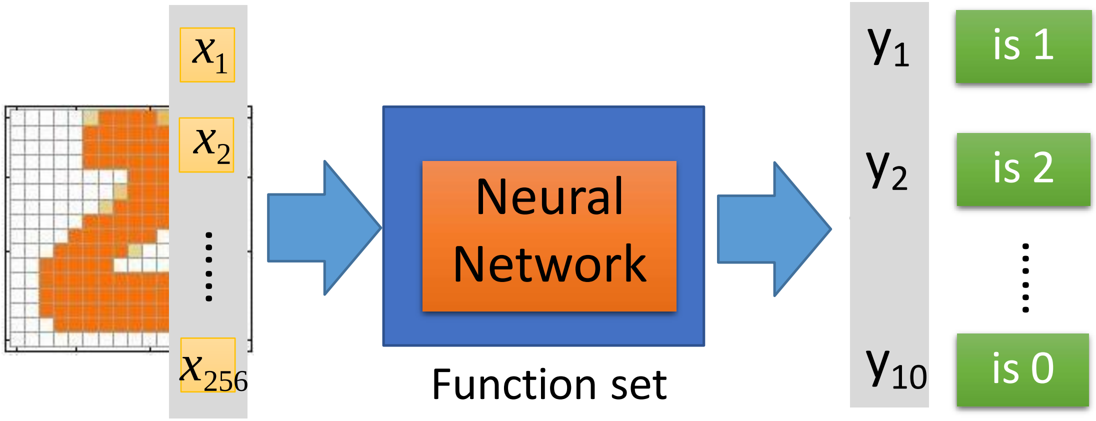
Output



Each dimension represents the confidence of a digit.

Example Application

- Handwriting Digit Recognition

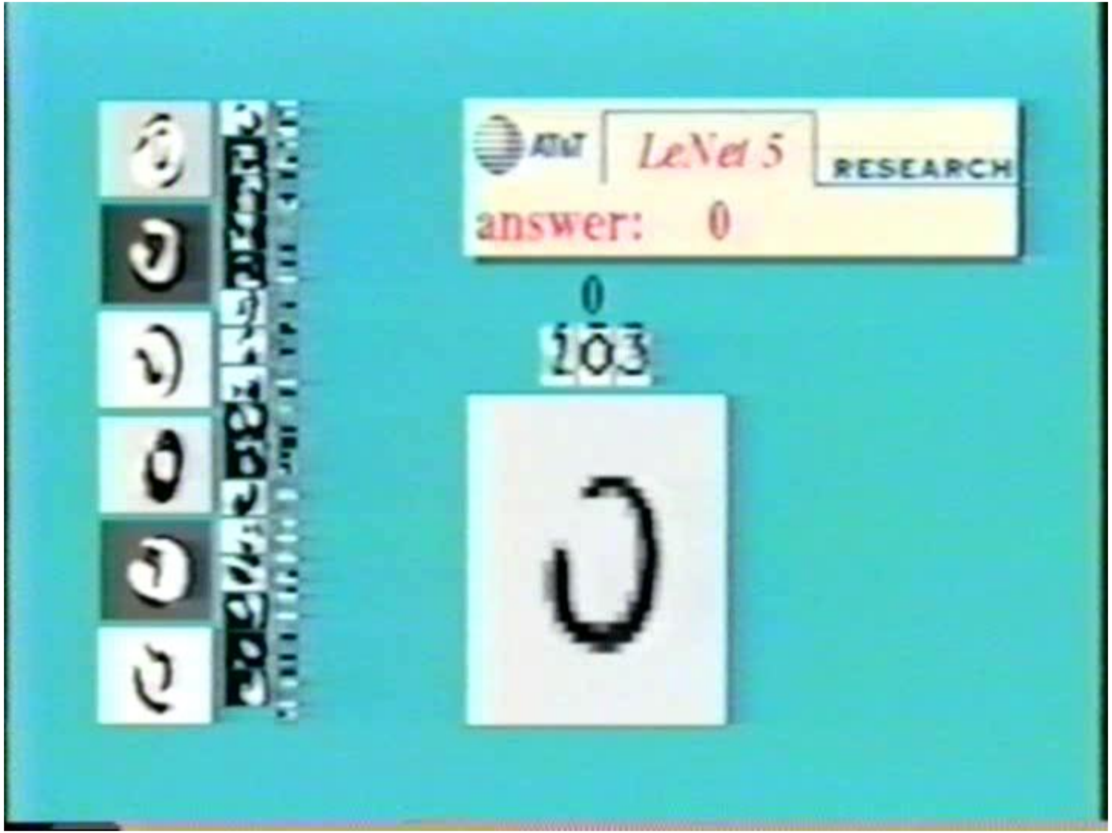


Input:
256-dim vector

output:
10-dim vector

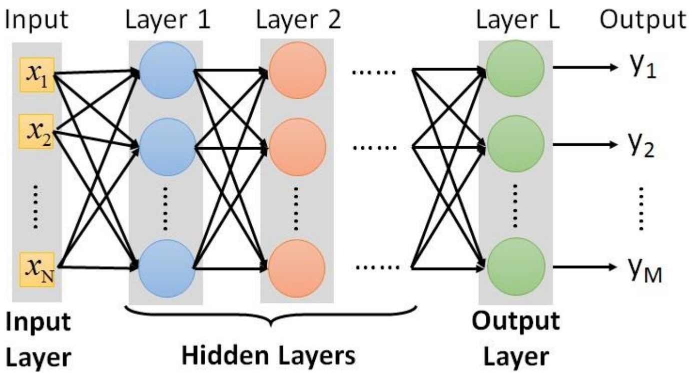
Handwritten digit classification

https://www.youtube.com/watch?v=yxuRnBEczUU



[Yann LeCun]

FAQ



- Q: How many layers? How many neurons for each layer?

Trial and Error + Intuition

- Q: Can the structure be automatically determined?
  - E.g. Evolutionary Artificial Neural Networks
- Q: Can we design the network structure?

Convolutional Neural Network (CNN)

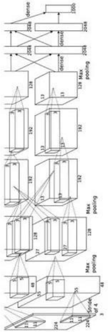
Neural Networks – Part 2

Deep = Many hidden layers

http://cs231n.stanford.edu/slides/winter1516_lecture8.pdf

8 layers

16.4%



AlexNet (2012)

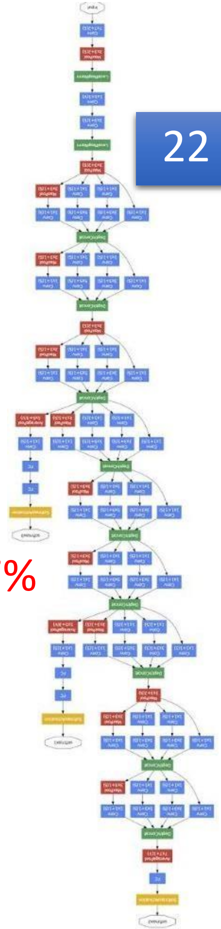
19 layers

7.3%

VGG (2014)

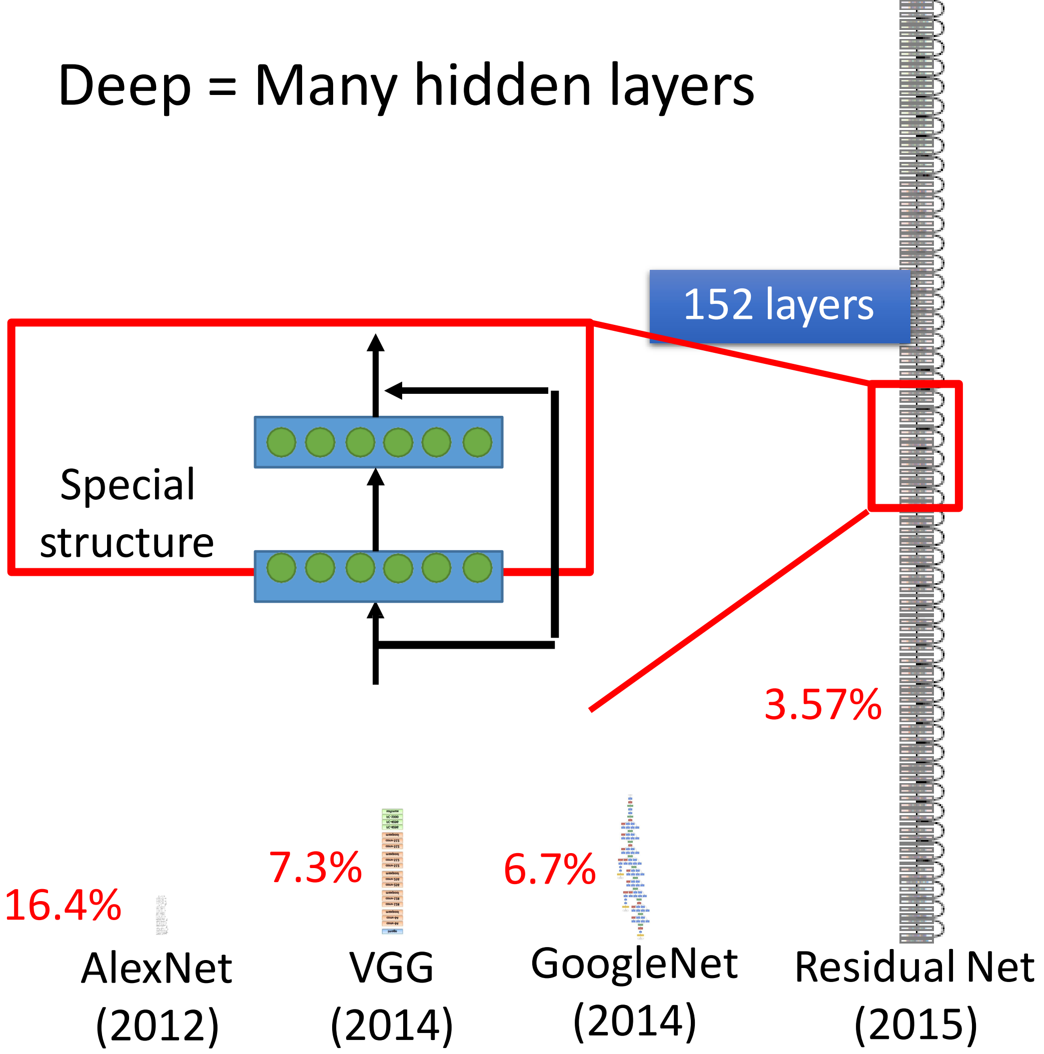
22 layers

6.7%

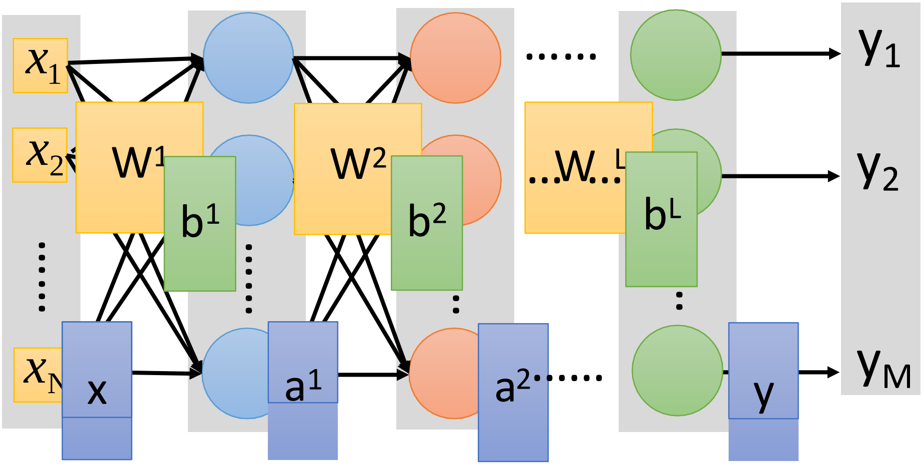


GoogleNet (2014)

Deep = Many hidden layers



Neural Network

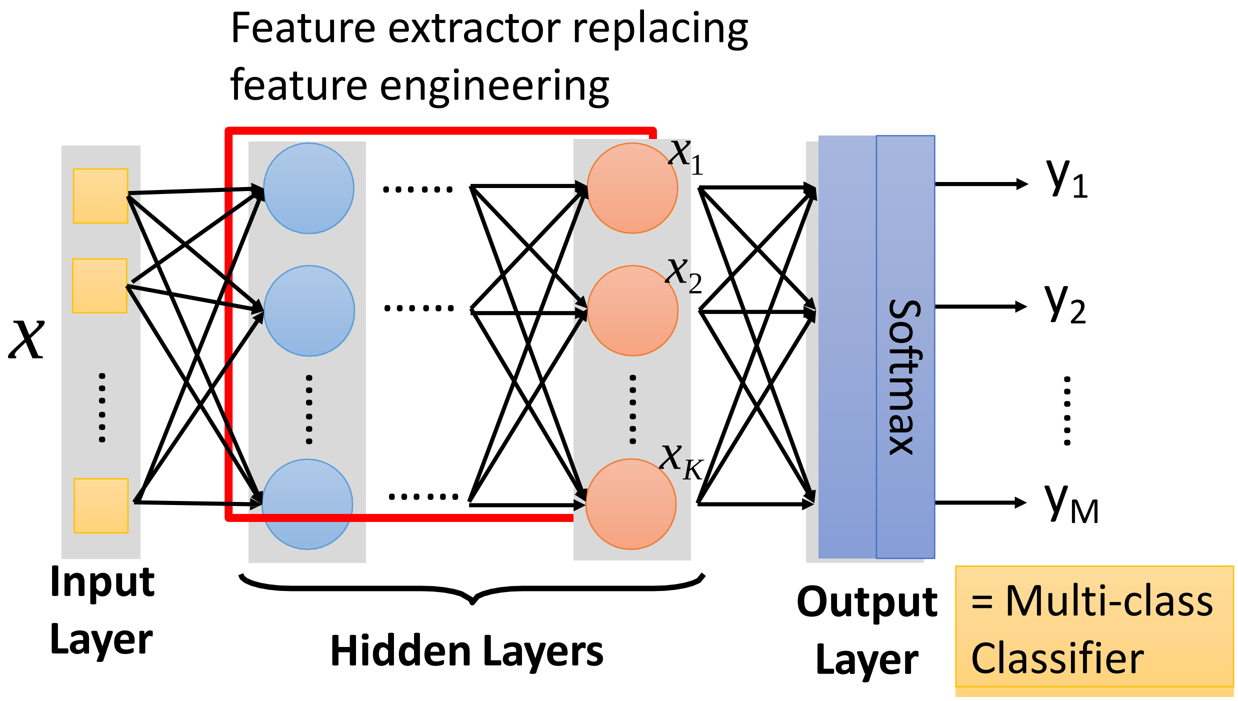


$$y = f(x)$$

Using parallel computing techniques to speed up matrix operation

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

Output Layer



Gradient Descent

This is the “learning” of machines in deep learning ……

➡ Even alpha go using this approach.

Backpropagation

- Backpropagation: an efficient way to compute $\partial L / \partial w$ in neural network

TensorFlow

torch

theano

Caffe

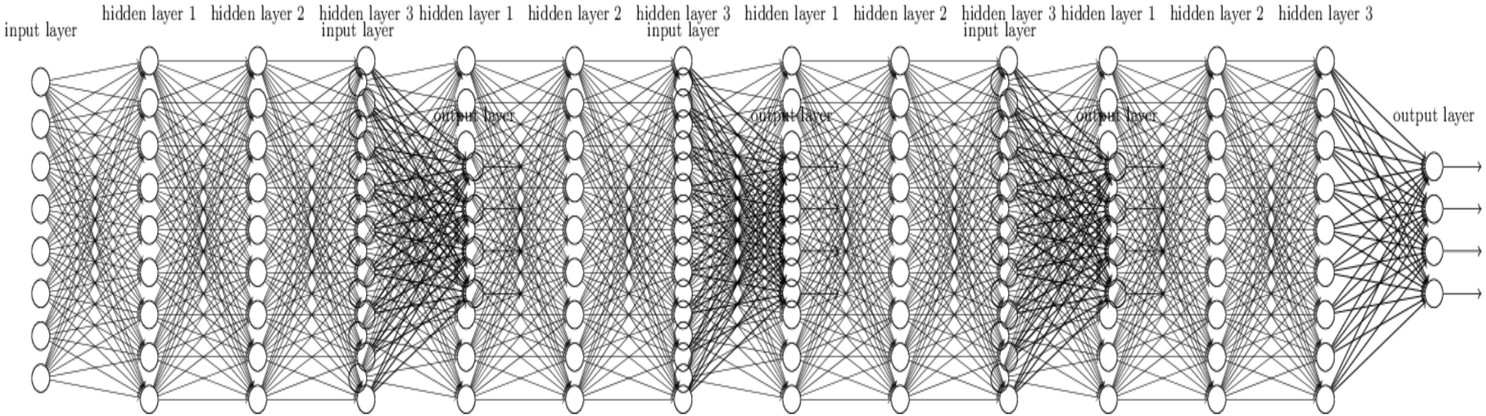
Microsoft CNTK

Chainer

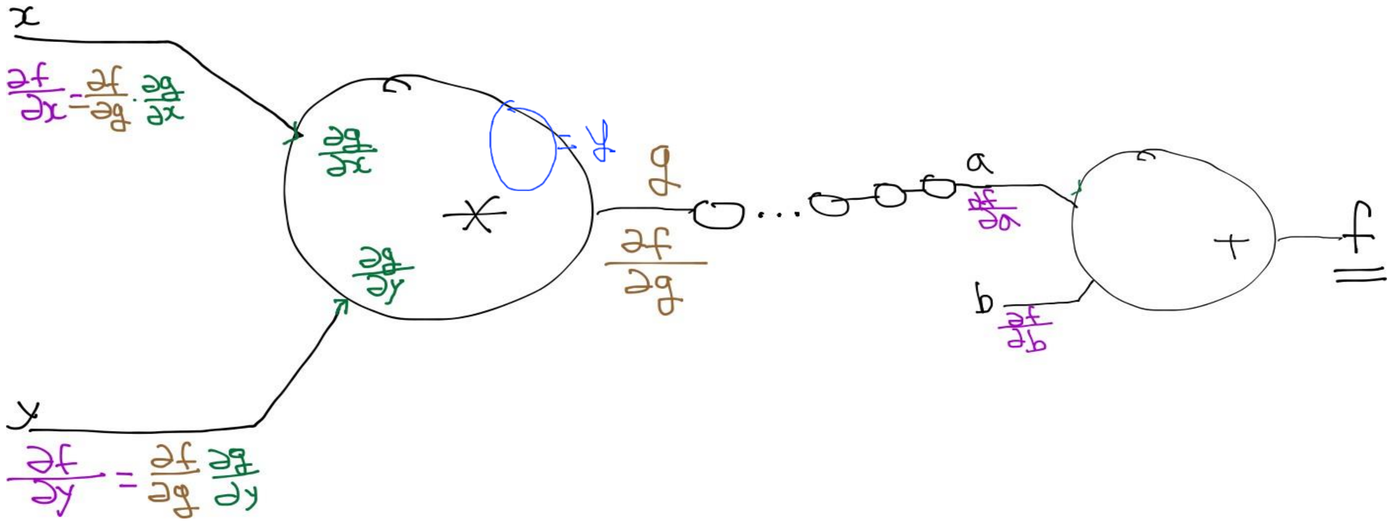
Deep Learning library produced by Amazon DSSTNE

mxnet

Back propagation



Backpropagation (chain rule)



http://cs231n.stanford.edu/

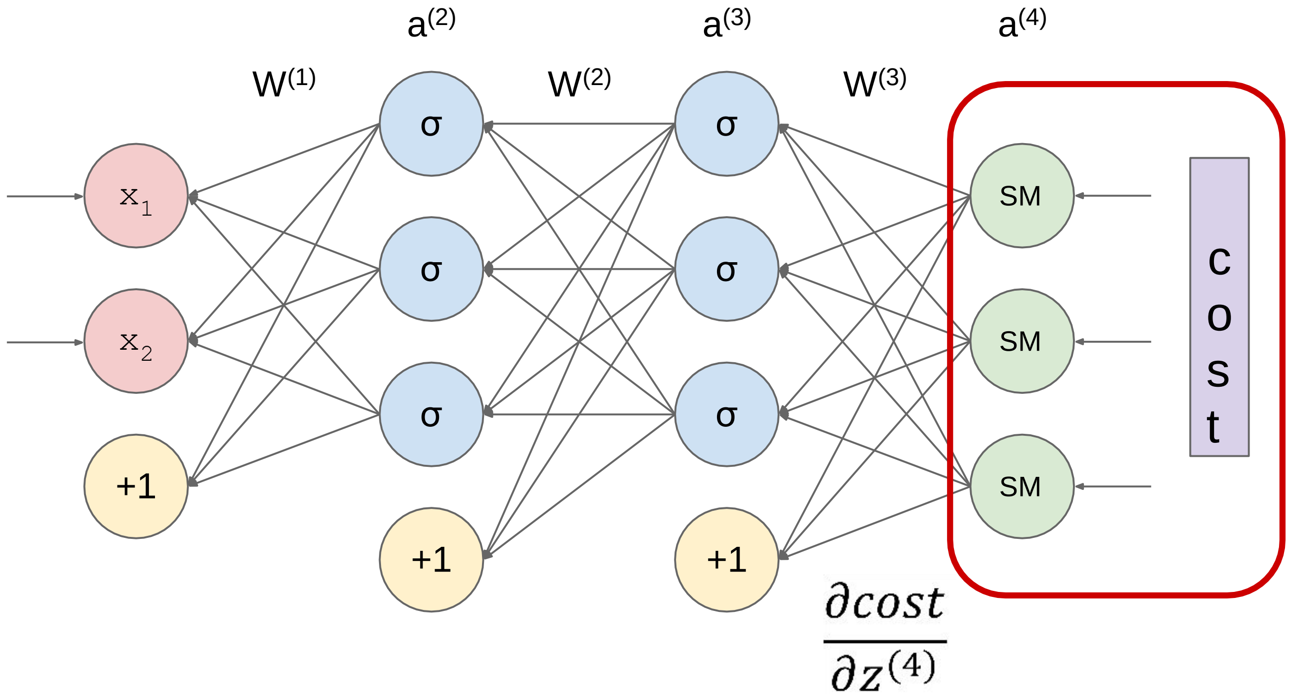
chain rule

$$f_1\left(f_2\left(f_3(f_4(x))\right)\right)$$

$$\frac{\partial f_1}{\partial x} = \frac{\partial f_1}{\partial f_2} \cdot \frac{\partial f_2}{\partial f_3} \cdot \frac{\partial f_3}{\partial f_4} \cdot \frac{\partial f_4}{\partial x}$$

Back Propagation

Want: $\frac{\partial cost}{\partial W^{(1)}}$ $\frac{\partial cost}{\partial W^{(2)}}$ $\frac{\partial cost}{\partial W^{(3)}}$



$$\frac{\partial cost}{\partial W^{(3)}} = \frac{\partial cost}{\partial z^{(4)}} \cdot \frac{\partial z^{(4)}}{\partial W^{(3)}} = \delta^{(4)} a^{(3)}$$

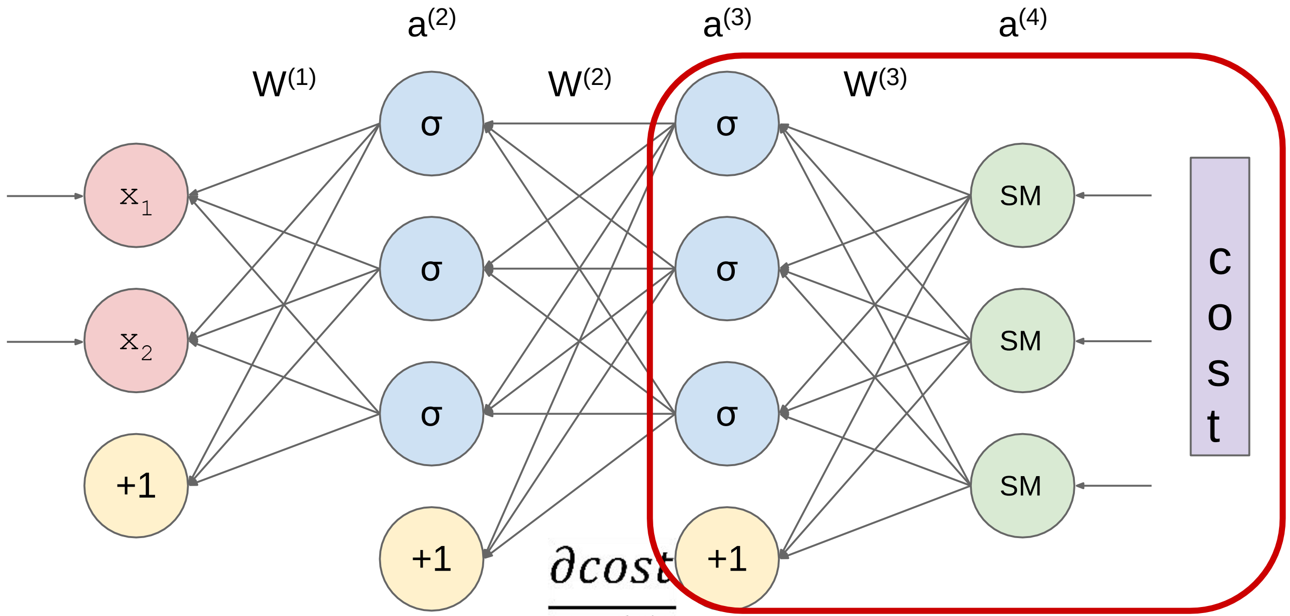
for layer l

er l

Sam Abrahams

Back Propagation

Want: $\frac{\partial cost}{\partial W^{(1)}}$ $\frac{\partial cost}{\partial W^{(2)}}$ $\frac{\partial cost}{\partial W^{(3)}}$



$$\frac{\partial cost}{\partial z^{(3)}}$$

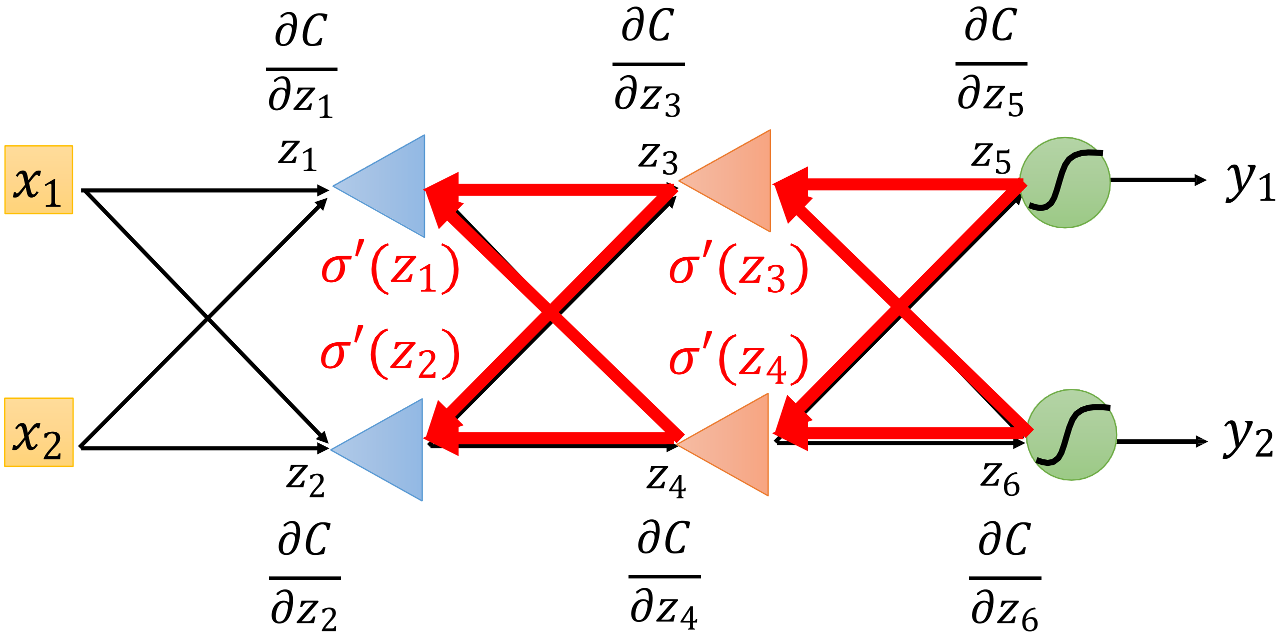
$$\frac{\partial cost}{\partial z^{(3)}} = \frac{\partial cost}{\partial z^{(4)}} \cdot \frac{\partial z^{(4)}}{\partial a^{(3)}} \cdot \frac{\partial a^{(3)}}{\partial z^{(3)}} = \delta^{(3)}$$

Sam Abrahams

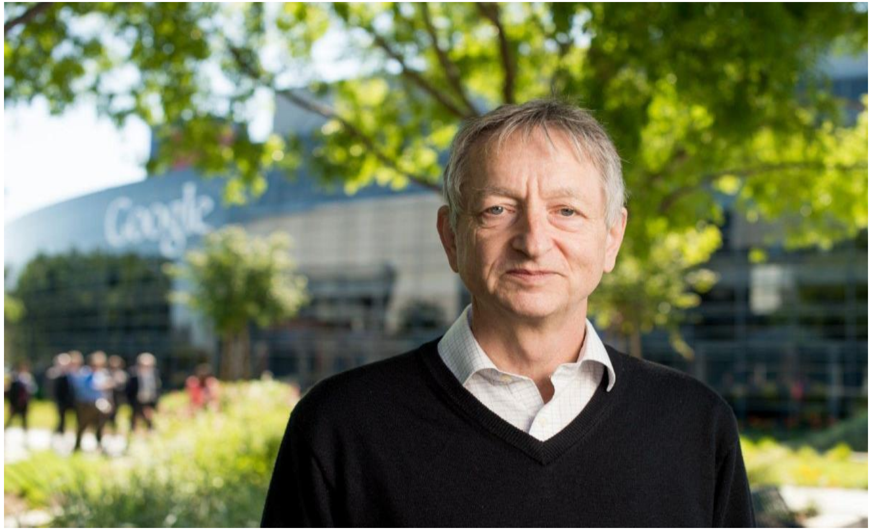
Backpropagation – Backward Pass

Compute $\partial C/\partial z$ for all activation function inputs z

Compute $\partial C/\partial z$ from the output layer



- Pre-training for deep neural networks



LETTER — Communicated by Yann Le Cun

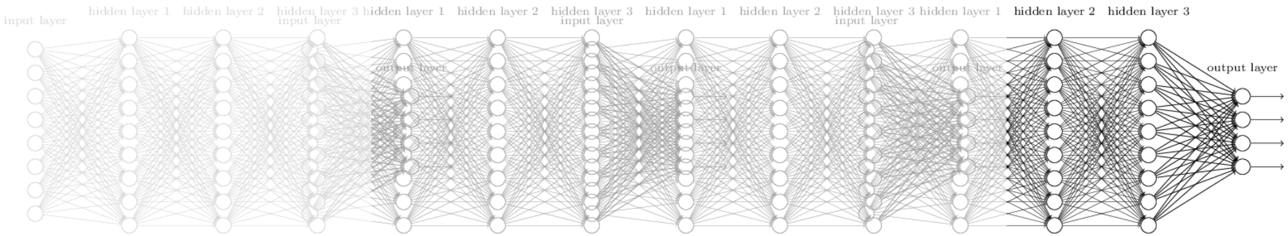
A Fast Learning Algorithm for Deep Belief Nets

Geoffrey E. Hinton
hinton@cs.toronto.edu
Simon Osindero
osindero@cs.toronto.edu
Department of Computer Science, University of Toronto, Toronto, Canada M5S 3G4

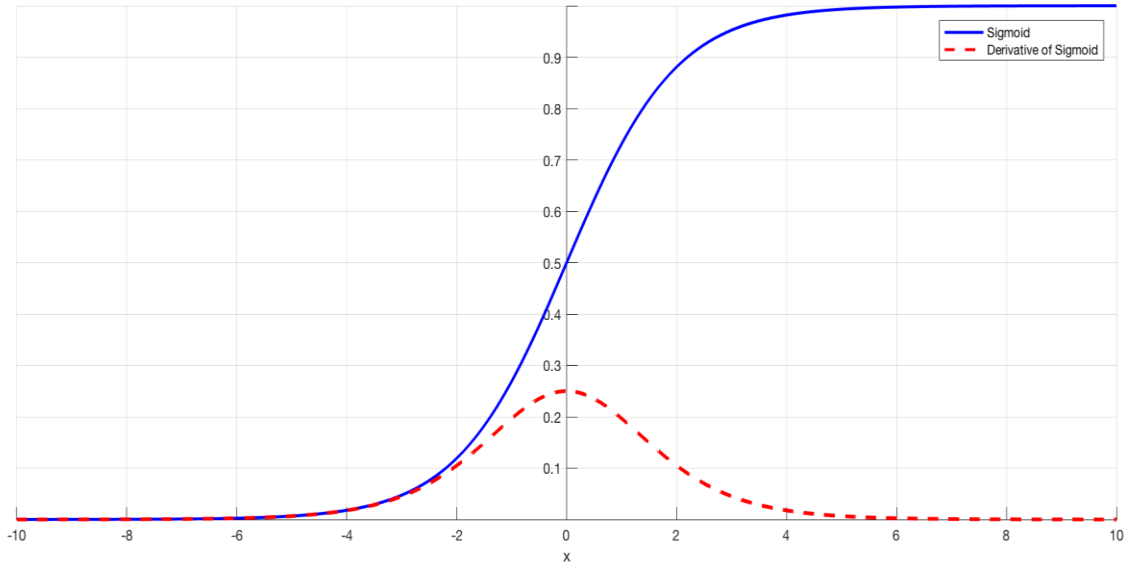
Yee-Whye Teh
tehyw@comp.nus.edu.sg
Department of Computer Science, National University of Singapore, Singapore 117543

We show how to use "complementary priors" to eliminate the explaining-away effects that make inference difficult in densely connected belief nets that have many hidden layers. Using complementary priors, we derive a fast, greedy algorithm that can learn deep, directed belief networks one layer at a time, provided the top two layers form an undirected associative memory. The fast, greedy algorithm is used to initialize a slower learning procedure that fine-tunes the weights using a contrastive version of the wake-sleep algorithm. After fine-tuning, a network with three hidden layers forms a very good generative model of the joint distribution of handwritten digit images and their labels. This generative model gives better digit classification than the best discriminative learning algorithms. The low-dimensional manifolds on which the digits lie are modeled by long ravines in the free-energy landscape of the top-level associative memory, and it is easy to explore these ravines by using the directed connections to display what the associative memory has in mind.

Vanishing gradient (NN winter2: 1986-2006)



Sigmoid!



https://isaacchanghau.github.io/img/deeplearning/activationfunction/sigmoid.png

NN Training

1. Do **forwards propagation**, calculate the input sum and activation of each neuron by iteratively do matrix-vector multiplication and taking component-wise transfer function of all neurons in every layer. Save the results for later.
2. Calculate the **error signal of the final layer *L***, by obtaining the gradient of the cost function with respect to the outputs of the network.
3. Do **backwards propagation**, calculate the error signals of the neurons in each layer. The input sum of each neuron is required to do this, and it is also done by iteratively computing matrix-vector multiplications

NN training cont’d

4.Calculate the **derivative of the cost function with respect to the weights**, the activation of each neuron is required to do this. This will be a matrix with the same shape as the weight matrix.

5.Calculate the **derivative of the cost function with respect to the biases** (this can be skipped). This will only be a column vector.

6. Update the weights according to the a gradient descent learning rule.

Xavier/He initialization

- Makes sure the weights are 'just right', not too small, not too big
- Using number of input (fan_in) and output (fan_out)

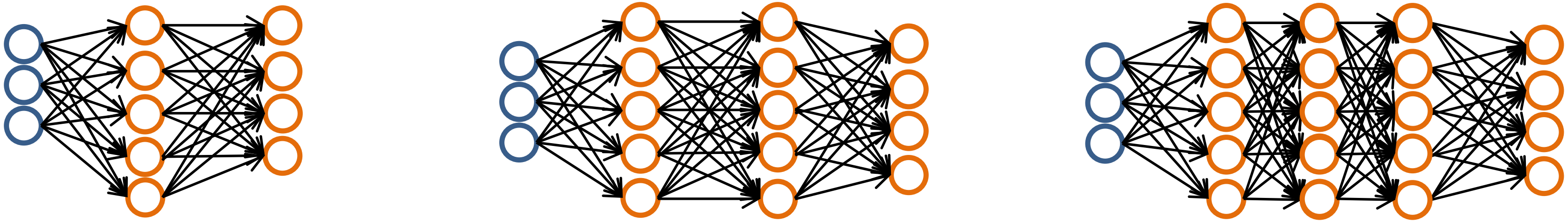
```
# Xavier initialization
# Glorot et al. 2010
W = np.random.randn(fan_in, fan_out)/np.sqrt(fan_in)

# He et al. 2015
W = np.random.randn(fan_in, fan_out)/np.sqrt(fan_in/2)
```

http://cs231n.stanford.edu/

Training a neural network

Pick a network architecture (connectivity pattern between neurons)



No. of input units: Dimension of features $x^{(i)}$

No. output units: Number of classes

Reasonable default: 1 hidden layer, or if >1 hidden layer, have same no. of hidden units in every layer (usually the more the better)

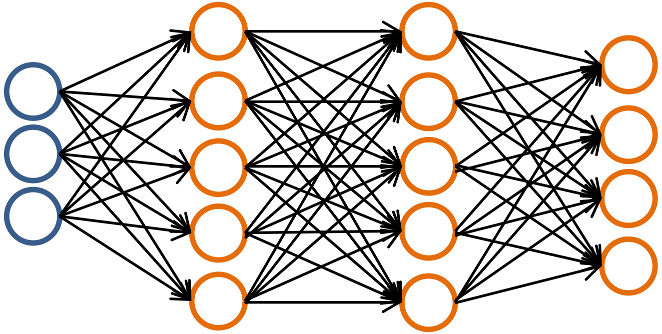
Multiple output units: One-vs-all.

Pedestrian

Car

Motorcycle

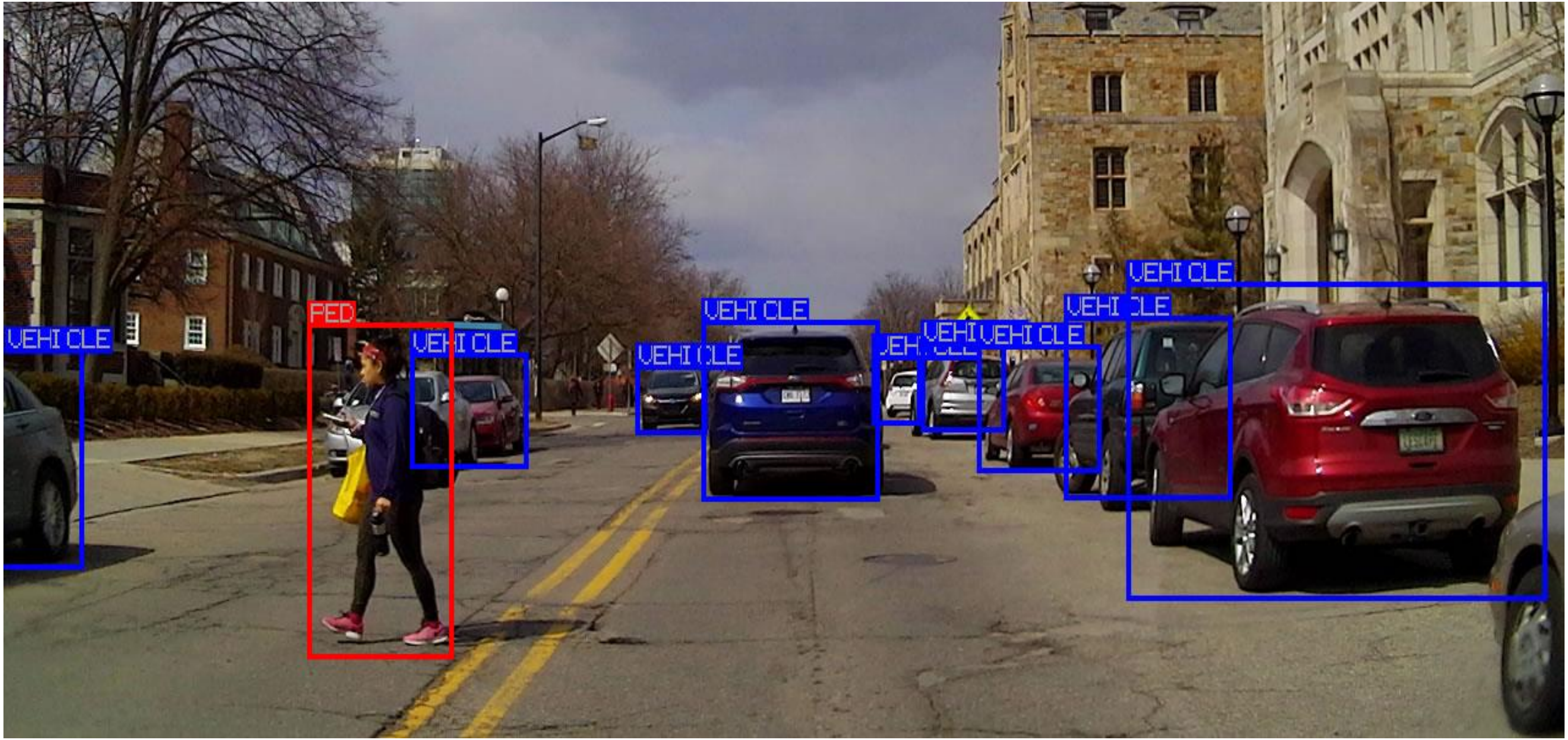
Truck



$$h_\Theta(x) \in \mathbb{R}^4$$

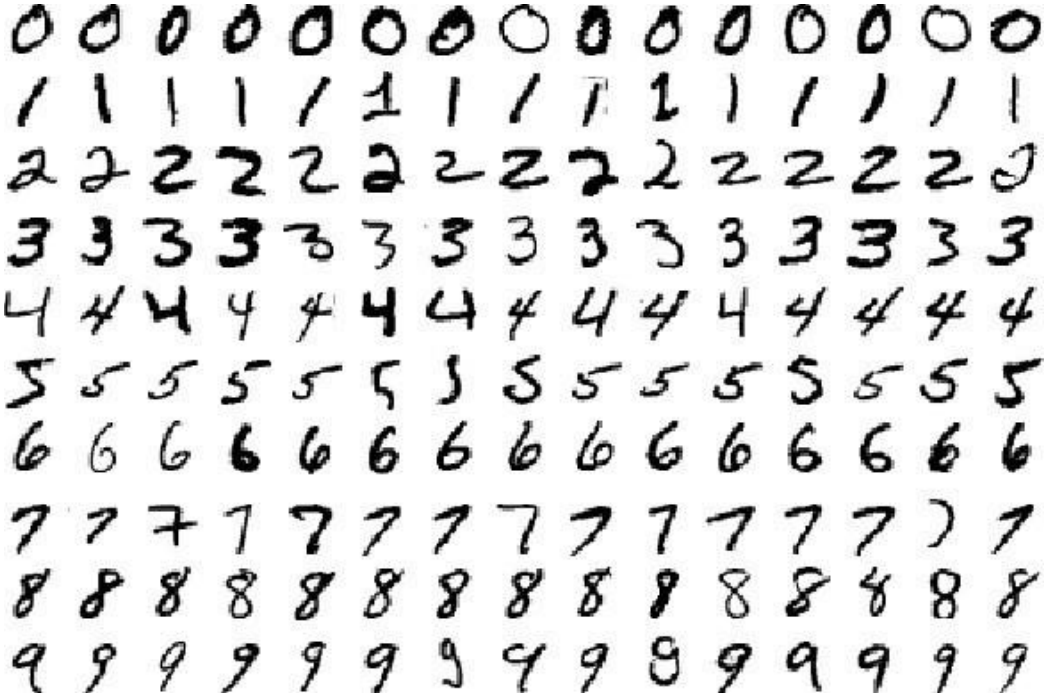
Want $h_\Theta(x) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$, etc.

when pedestrian when car when motorcycle


VEHICLE
PED
VEHICLE
VEHICLE
VEHICLE
VEHICLE
VEHICLE
VEHICLE

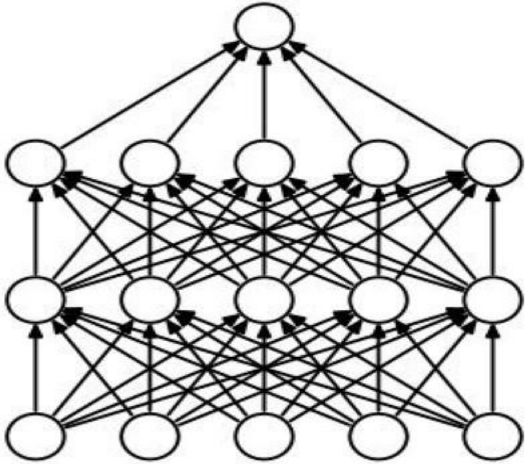
MNIST dataset (28x28 gray scale image)

The MNIST database contains 60,000 training images and 10,000 testing images

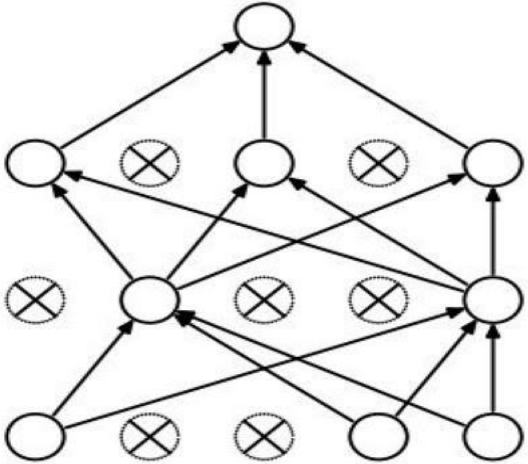


Regularization: **Dropout**

"randomly set some neurons to zero in the forward pass"



(a) Standard Neural Net



(b) After applying dropout.

[Srivastava et al., 2014]

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

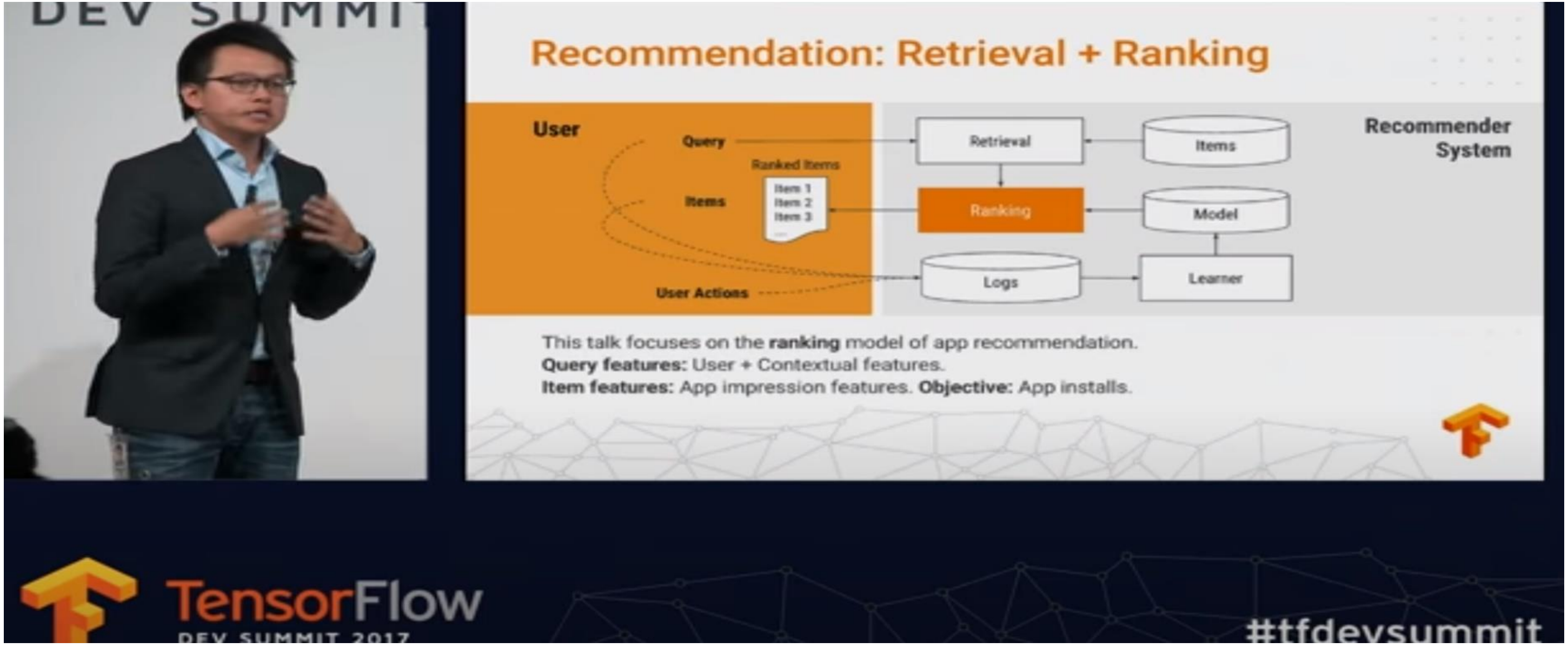
more parameters,
better performance

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

Wide & Deep Learning

https://ai.googleblog.com/2016/06/wide-deep-learning-better-together-with.html

https://www.youtube.com/watch?v=NV1tkZ9Lq48&feature=emb_rel_end



Wide & Deep Learning for Recommender Systems

https://arxiv.org/pdf/1606.07792.pdf

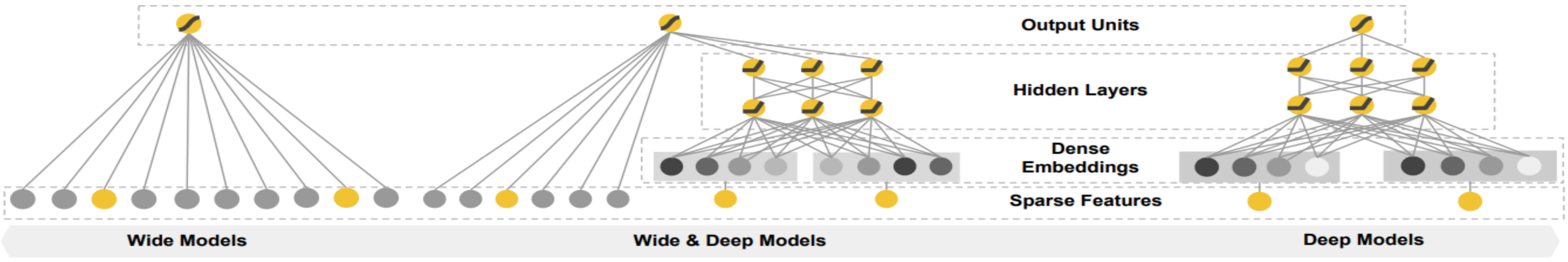


Figure 1: The spectrum of Wide & Deep models.

linear model with feature transformations for generic recommender systems with sparse inputs.

- The implementation and evaluation of the Wide & Deep recommender system productionized on Google Play, a mobile app store with over one billion active users and over one million apps.
- We have open-sourced our implementation along with a high-level API in TensorFlow[1].

While the idea is simple, we show that the Wide & Deep framework significantly improves the app acquisition rate

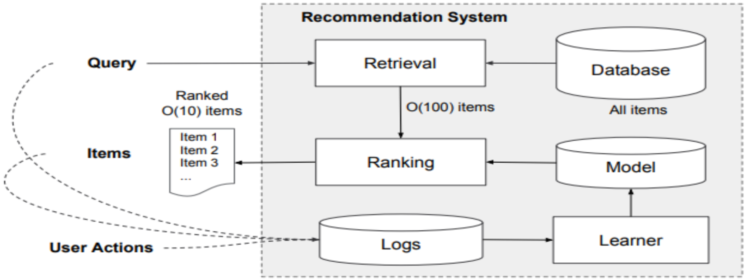


Figure 2: Overview of the recommender system.