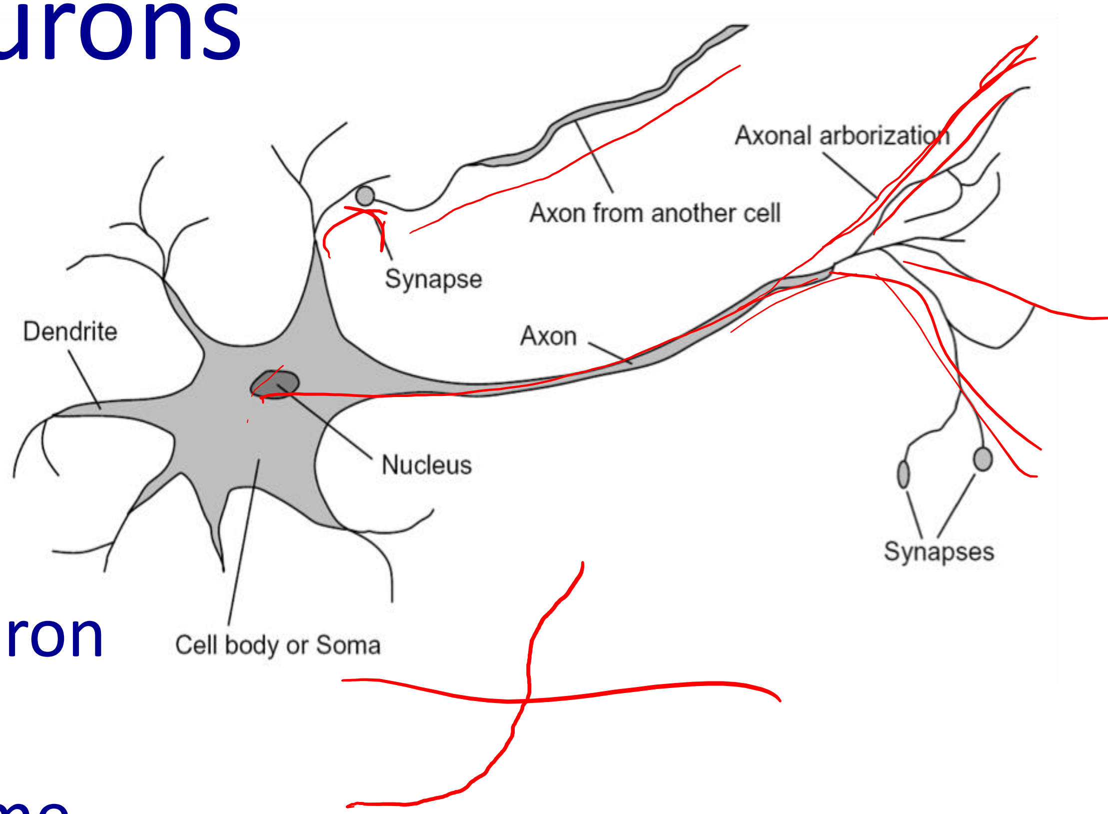
CSE446: Neural Networks

Spring 2017

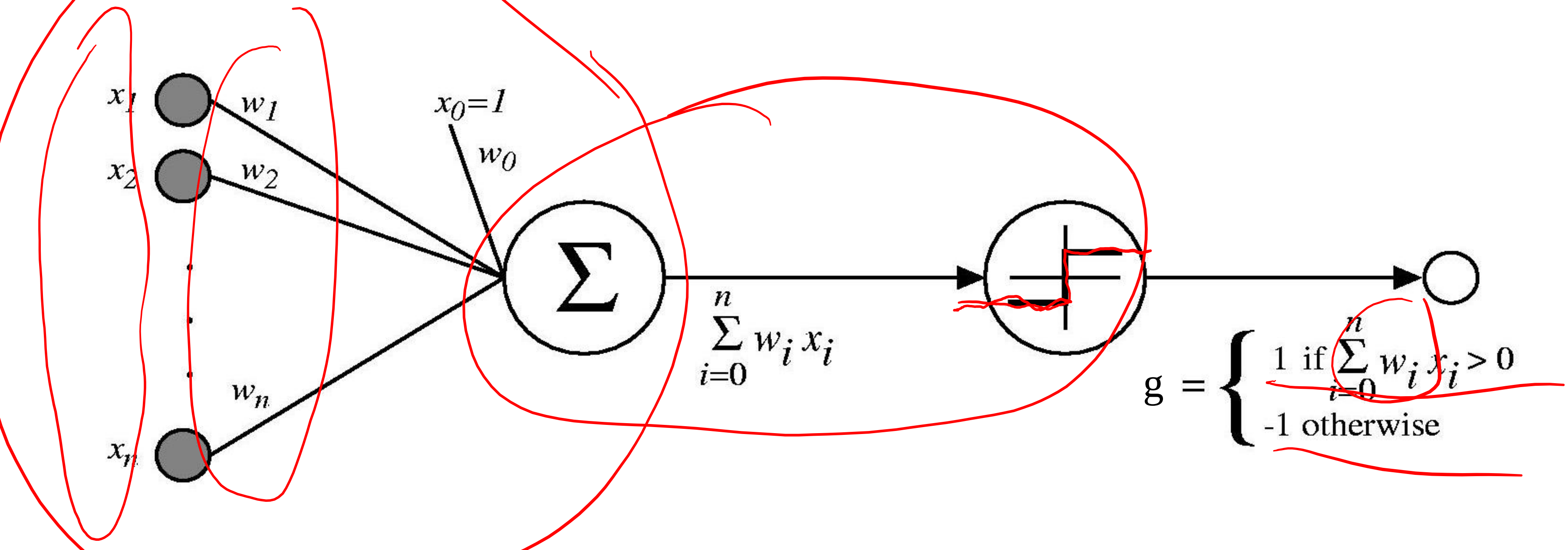
Many slides are adapted from Carlos Guestrin and Luke Zettlemoyer

Human Neurons



- Switching time
  - ~ 0.001 second
- Number of neurons
  - 10^{10}
- Connections per neuron
  - $10^{4\text{-}5}$
- Scene recognition time
  - 0.1 seconds
- Number of cycles per scene recognition?
  - 100 → much parallel computation!

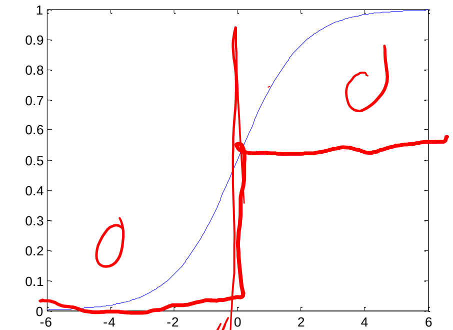
Perceptron as a Neural Network



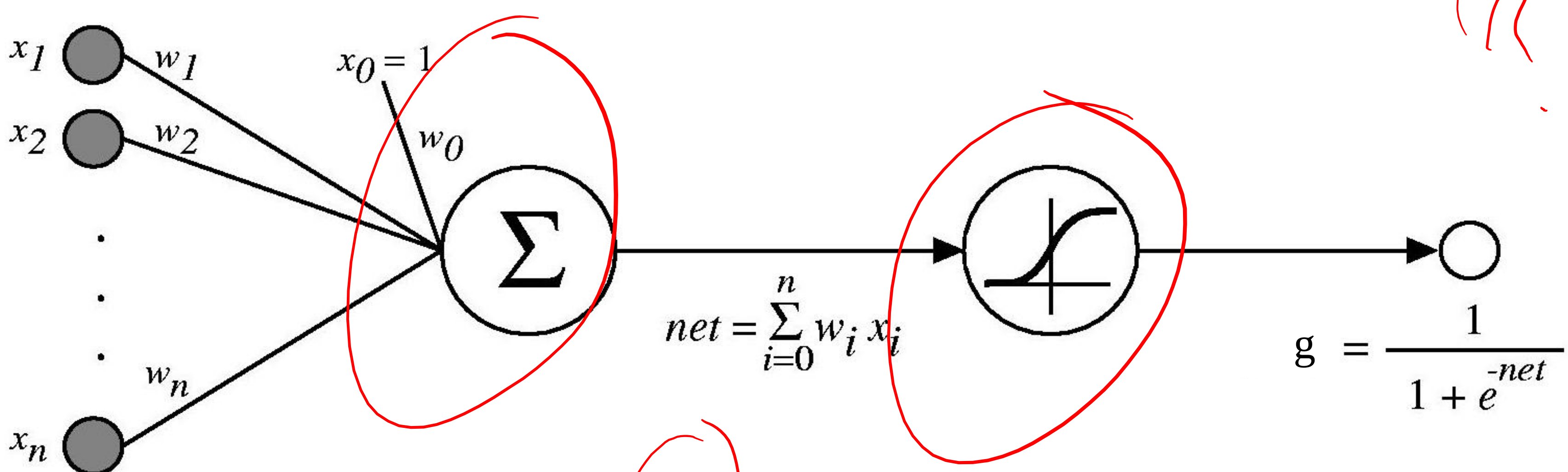
This is one neuron:

- Input edges x_1 ... x_n, along with basis
- The sum is represented graphically
- Sum passed through an activation function g

Sigmoid Neuron



$$g(w_0 + \sum_i w_i x_i) = \frac{1}{1 + e^{-(w_0 + \sum_i w_i x_i)}}$$



Just change g!

- Why would be want to do this?
- Notice new output range [0,1]. What was it before?
- Look familiar?

Optimizing a neuron

$$\frac{\partial}{\partial x} f(g(x)) = f'(g(x))g'(x)$$

We train to minimize sum-squared error

$$\ell(W) = \frac{1}{2}\sum_j [y^j - g(w_0 + \sum_i w_i x_i^j)]^2$$

$$\frac{\partial l}{\partial w_i} = -\sum_j [y_j - g(w_0 + \sum_i w_i x_i^j)]\frac{\partial}{\partial w_i} g(w_0 + \sum_i w_i x_i^j)$$

$f(g(x))$

$$\frac{\partial}{\partial w_i} g(w_0 + \sum_i w_i x_i^j) = x_i^j \frac{\partial}{\partial w_i} g(w_0 + \sum_i w_i x_i^j) = x_i^j g'(w_0 + \sum_i w_i x_i^j)$$

$$\boxed{\frac{\partial \ell(W)}{\partial w_i} = -\sum_j [y^j - g(w_0 + \sum_i w_i x_i^j)]\ x_i^j\ g'(w_0 + \sum_i w_i x_i^j)}$$

Solution just depends on g': derivative of activation function!

Re-deriving the perceptron update

$$\frac{\partial \ell(W)}{\partial w_i} = -\sum_j [y^j - g(w_0 + \sum_i w_i x_i^j)]\ x_i^j\ g'(w_0 + \sum_i w_i x_i^j)$$

$g' = 1$??

$$g = \begin{cases} 1 \text{ if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 \text{ otherwise} \end{cases}$$

$$\frac{\partial \ell(W)}{\partial w_i} = -\sum_j [y^j - g(w_0 + \sum_i w_i x_i^j)]\ x_i^j$$

For a specific, incorrect example:

- $w = w + y^*x$ (our familiar update!)

Sigmoid units: have to differentiate g

$$\frac{\partial \ell(W)}{\partial w_i} = -\sum_j [y^j - g(w_0 + \sum_i w_i x_i^j)] \; x_i^j \; g'(w_0 + \sum_i w_i x_i^j)$$

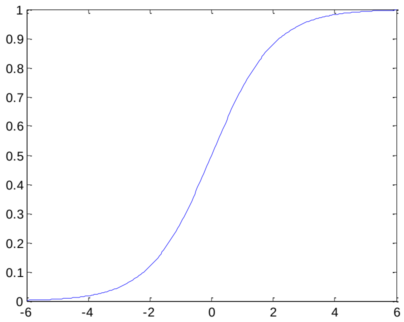
$$g(x) = \frac{1}{1+e^{-x}} \qquad g'(x) = g(x)(1-g(x))$$

$$\boxed{w_i \leftarrow w_i + \eta \sum_j x_i^j \delta^j}$$

$$\delta^j = [y^j - g(w_0 + \sum_i w_i x_i^j)] g^j (1 - g^j)$$

$$g^j = g(w_0 + \sum_i w_i x_i^j)$$

Aside: Comparison to logistic regression



- P(Y|X) represented by:

$$P(Y=1 \mid x, W) = \frac{1}{1+e^{-(w_0+\sum_i w_i x_i)}}$$
$$= g(w_0 + \sum_i w_i x_i)$$

- Learning rule – MLE:

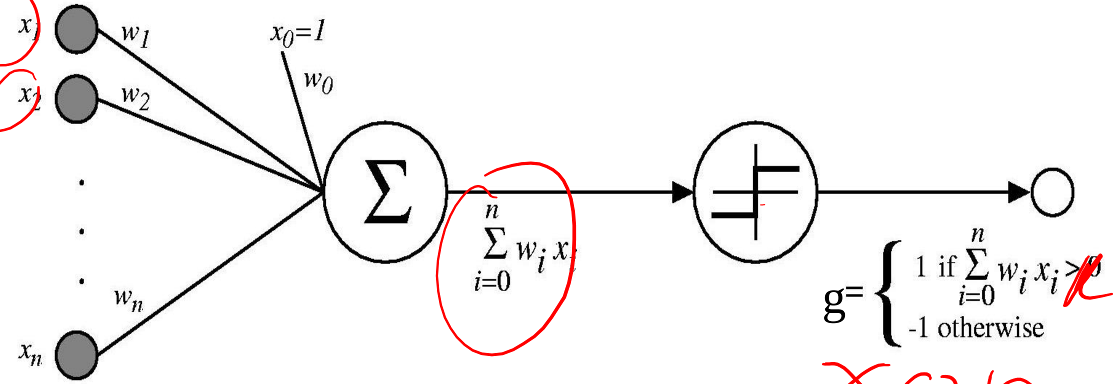
$$\frac{\partial \ell(W)}{\partial w_i} = \sum_j x_i^j [y^j - P(Y^j = 1 \mid x^j, W)]$$
$$= \sum_j x_i^j [y^j - g(w_0 + \sum_i w_i x_i^j)]$$

$$w_i \leftarrow w_i + \eta \sum_j x_i^j \delta^j$$

$$\delta^j = y^j - g(w_0 + \sum_i w_i x_i^j)$$

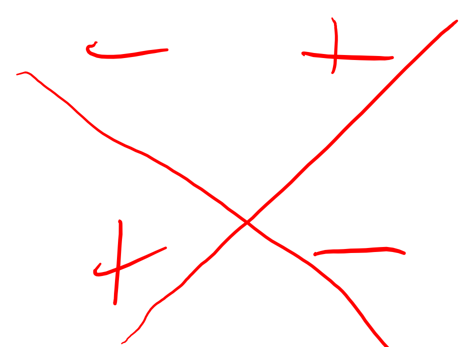
Perceptron, linear classification, Boolean functions: $x_i \in \{0,1\}$

- Can learn $x_1 \lor x_2$?
  - $-0.5 + x_1 + x_2$
- Can learn $x_1 \land x_2$?
  - $-1.5 + x_1 + x_2$
- Can learn any conjunction or disjunction?
  - $0.5 + x_1 + \ldots + x_n$
  - $(-n+0.5) + x_1 + \ldots + x_n$
- Can learn majority?
  - $(-0.5*n) + x_1 + \ldots + x_n$
- What are we missing? The dreaded XOR!, etc.



XOR

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



Going beyond linear classification

hidden units

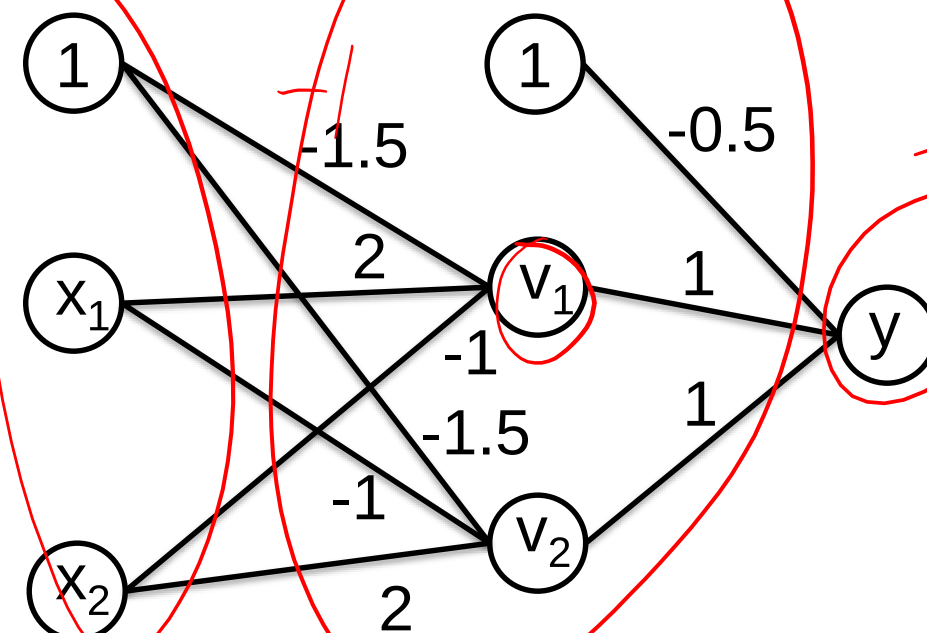
Solving the XOR problem

$$y = x_1 \text{ XOR } x_2 = (x_1 \wedge \neg x_2) \vee (x_2 \wedge \neg x_1)$$

$$v_1 = (x_1 \wedge \neg x_2)$$
$$= -1.5 + 2x_1 - x_2$$

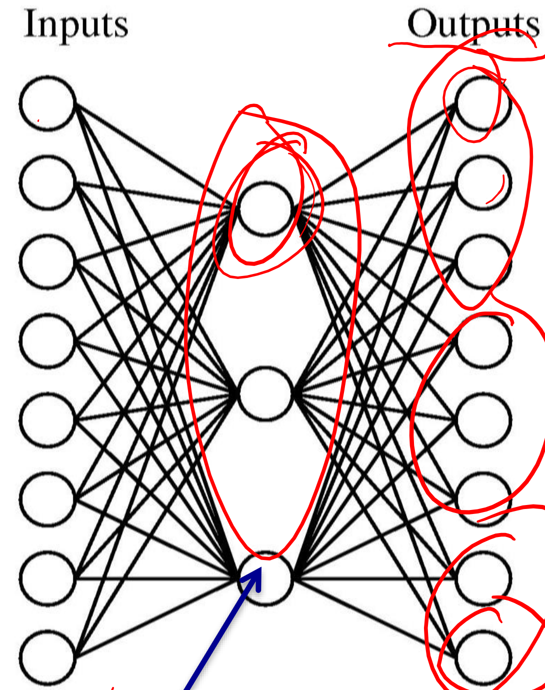
$$v_2 = (x_2 \wedge \neg x_1)$$
$$= -1.5 + 2x_2 - x_1$$

$$y = v_1 \vee v_2$$
$$= -0.5 + v_1 + v_2$$



f(x) = x

Hidden layer



class

bbox

- Single unit: g = 1

$$out(\mathbf{x}) = g(w_0 + \sum_i w_i x_i)$$

- 1-hidden layer:

output

$$out(\mathbf{x}) = g\left(w_0 + \sum_k w_k g(w_0^k + \sum_i w_i^k x_i)\right)$$

hidden layer

- No longer convex function!

logistic

RELU

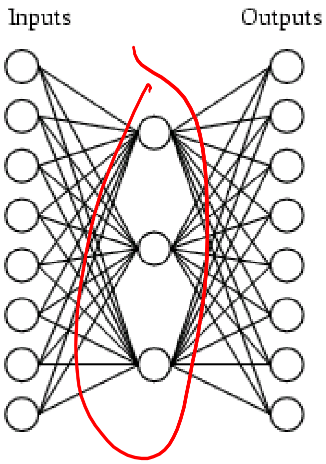
0

f(x) = x, x > 0

f(x) = x

f(x) = x

Example data for NN with hidden layer



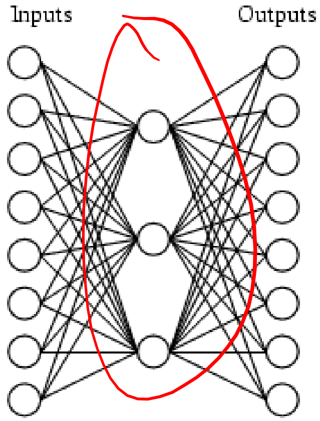
A target function:

Input		Output
10000000	$\rightarrow$	10000000
01000000	$\rightarrow$	01000000
00100000	$\rightarrow$	00100000
00010000	$\rightarrow$	00010000
00001000	$\rightarrow$	00001000
00000100	$\rightarrow$	00000100
00000010	$\rightarrow$	00000010
00000001	$\rightarrow$	00000001

Can this be learned??

Learned weights for hidden layer

A network:



Learned hidden layer representation:

Input		Hidden Values				Output
10000000	→	.89	.04	.08	→	10000000
01000000	→	.01	.11	.88	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.22	.99	.99	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

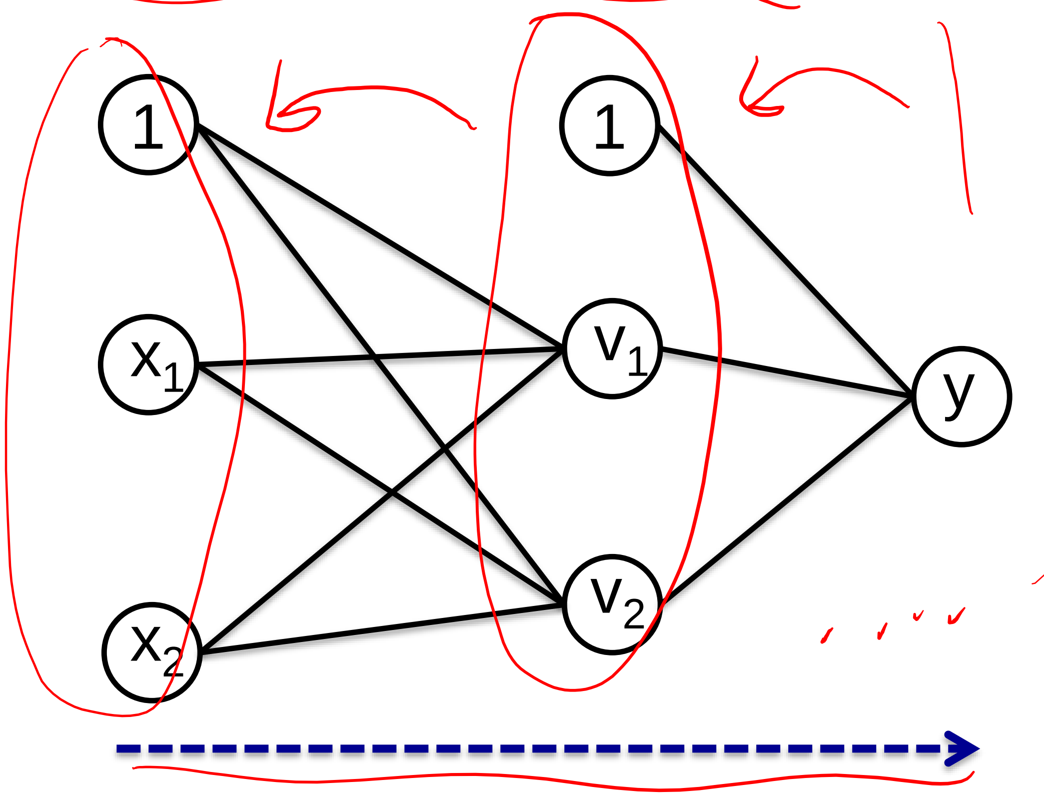
Forward propagation

1-hidden layer:

$$out(\mathbf{x}) = g\left(w_0 + \sum_k w_k g(w_0^k + \sum_i w_i^k x_i)\right)$$

Compute values left to right

1. Inputs: $x_1, \ldots, x_n$
2. Hidden: $v_1, \ldots, v_n$
3. Output: y



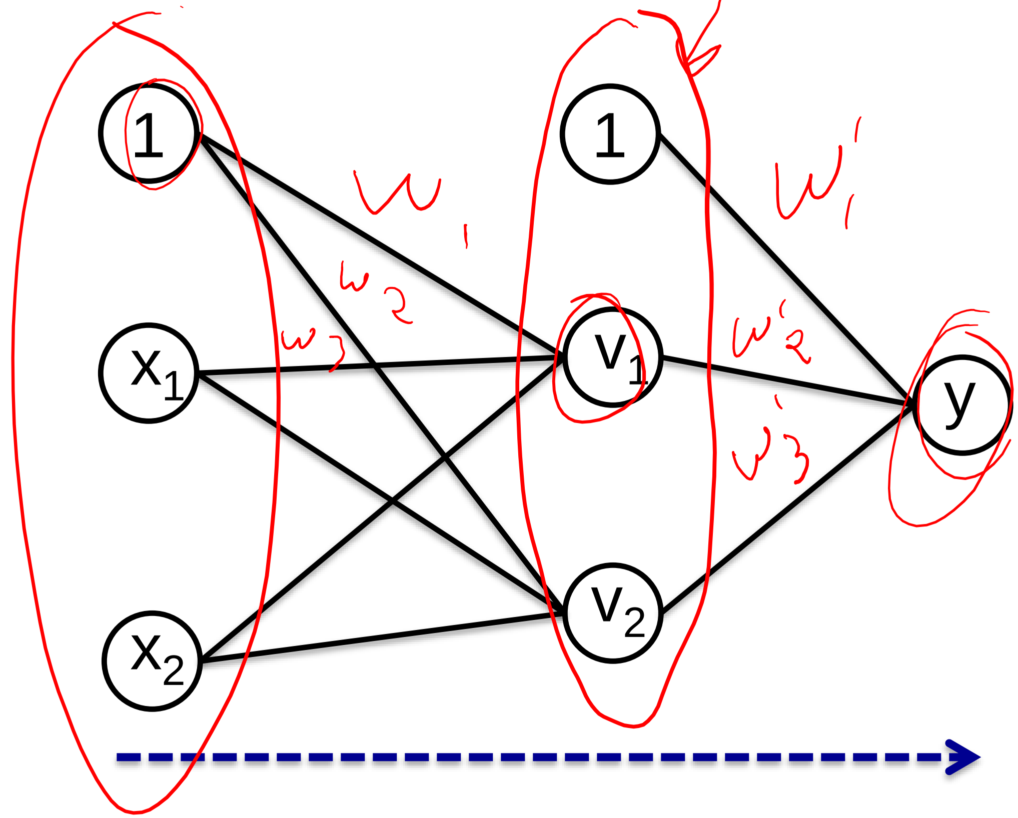
Forward propagation

1-hidden layer:

$$out(\mathbf{x}) = g\left(w_0 + \sum_k w_k g(w_0^k + \sum_i w_i^k x_i)\right)$$

Compute values left to right

1. Inputs: $x_1, \ldots, x_n$
2. Hidden: $v_1, \ldots, v_n$
3. Output: y



Forward propagation

$g'(x)$

$g(x)\cdot(1-g(x))$

1-hidden layer:

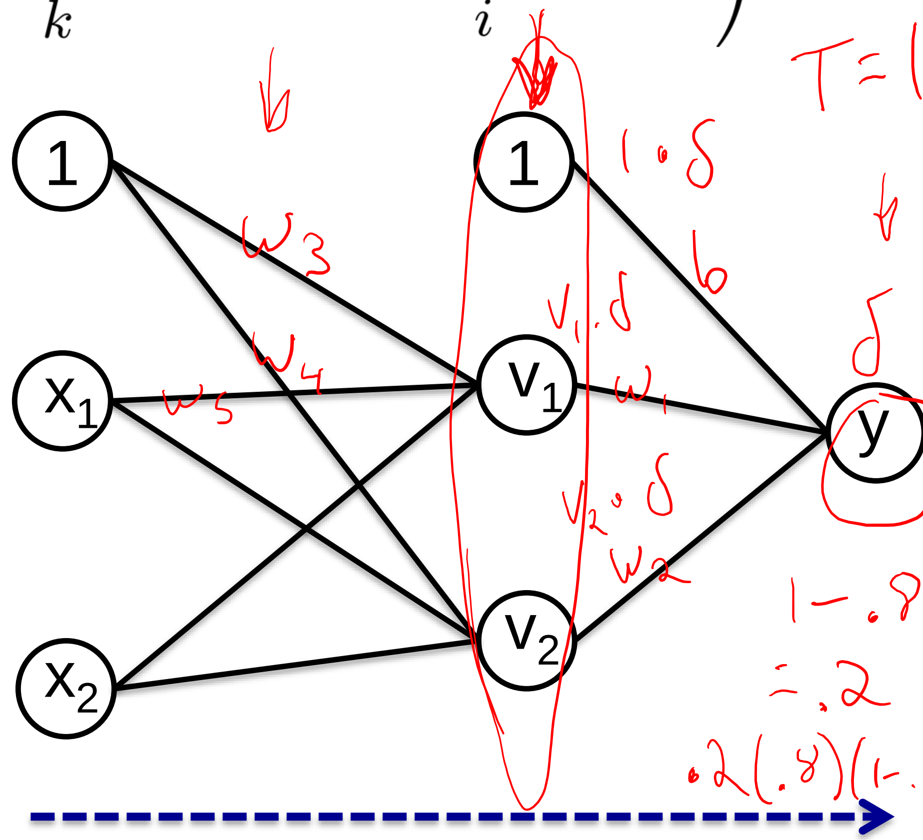
$$out(\mathbf{x}) \;=\; g\left(w_0 + \sum_k w_k g(w_0^k + \sum_i w_i^k x_i)\right)$$

Compute values left to right

1. Inputs: $x_1, \ldots, x_n$
2. Hidden: $v_1, \ldots, v_n$
3. Output: y

$b = b + \eta \cdot 1 \cdot \delta$

$w_1 = w_1 + \eta \cdot v_1 \cdot \delta$



$T = 1$

$1 - .8$

$= .2$

$.2(.8)(1-.8)$

$= \delta$

Forward propagation

1-hidden layer:

$$out(\mathbf{x}) = g\left(w_0 + \sum_k w_k g(w_0^k + \sum_i w_i^k x_i)\right)$$

$g'(x)$

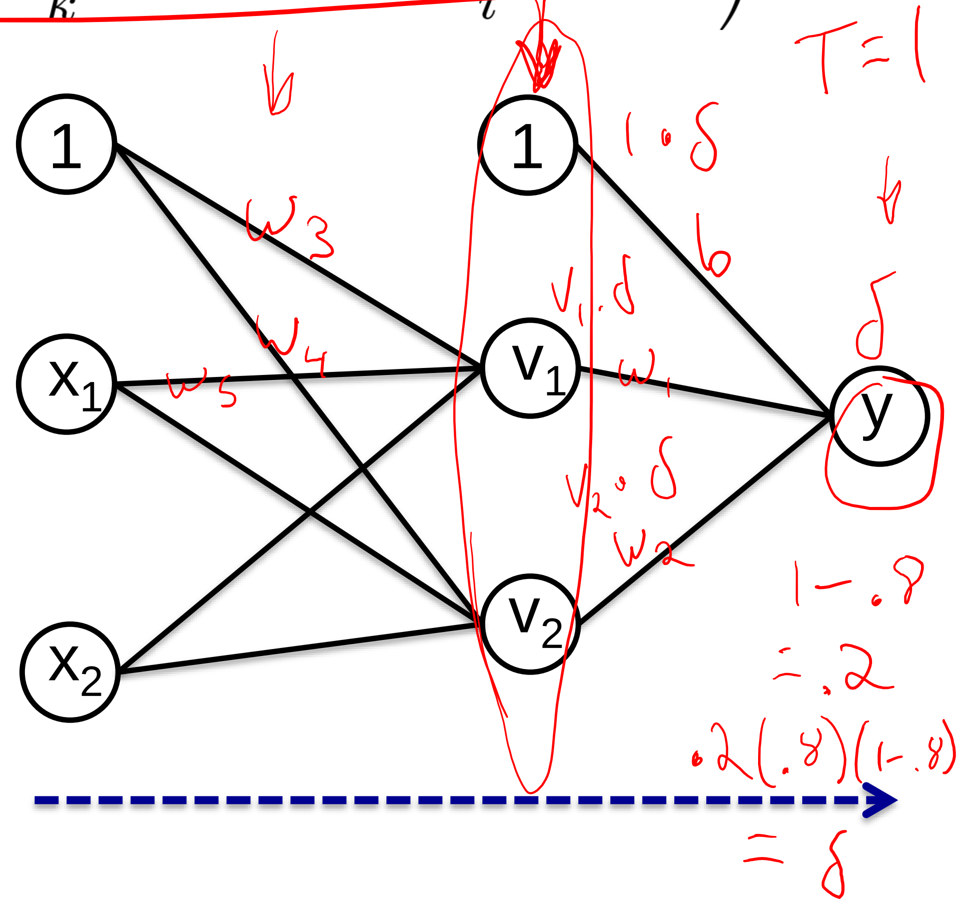
$g(x) \cdot (1 - g(x))$

Compute values left to right

1. Inputs: $x_1, \ldots, x_n$
2. Hidden: $v_1, \ldots, v_n$
3. Output: y

$b = b + \eta \cdot 1 \cdot \delta$

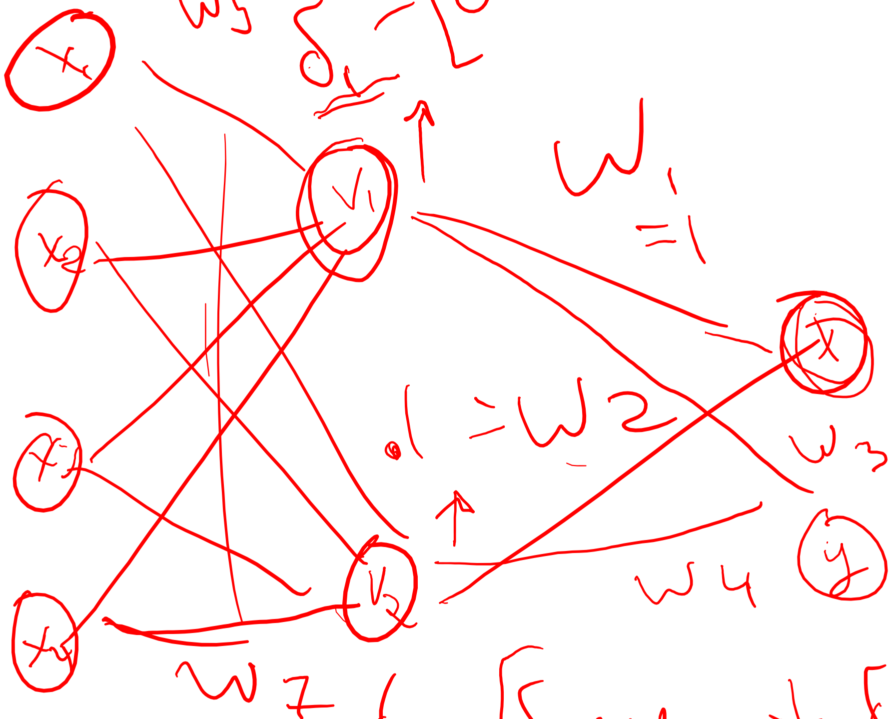
$w_1 = w_1 + \eta \cdot v_1 \cdot \delta$



θ

$$\delta_1 = [\delta_0 \cdot w_1 + \delta_\pi \cdot w_3] \cdot v_1 \cdot (1 - v_1)$$

$g'(x) = g(x) \cdot (1 - g(x))$



δ_0 want ↑

$(\hat{y} - y)$

δ_π

$$\delta_2 = [\delta_0 \cdot w_2 + \delta_\pi \cdot w_4] \cdot v_2 \cdot (1 - v_2)$$

$$w_7 = w_7 + \eta \cdot \delta_2 \cdot x_4$$

Gradient descent for 1-hidden layer

$$\boxed{\frac{\partial \ell(W)}{\partial w_k}}$$

Dropped w_0 to make derivation simpler

$$\ell(W) = \frac{1}{2}\sum_j [y^j - out(\mathbf{x}^j)]^2$$

$$out(\mathbf{x}) = g\left(\sum_{k'} w_{k'} g(\sum_{i'} w_{i'}^{k'} x_{i'})\right)$$

$$\boxed{v_k^j = g\left(\sum_{i'} w_{i'}^{k'} x_{i'}\right)}$$

$$\frac{\partial \ell(W)}{\partial w_k} = \sum_{j=1}^{m} -[y^j - out(\mathbf{x}^j)]\frac{\partial out(\mathbf{x}^j)}{\partial w_k}$$

$$out(x) = g\left(\sum_{k'} w_{k'} v_k^j\right)$$

$$\frac{\partial out(\mathbf{x})}{\partial w_k} = v_k^j g'\left(\sum_{k'} w_{k'} v_k^j\right)$$

Gradient for last layer same as the single node case, but with hidden nodes v as input!

Gradient descent for 1-hidden layer

$$\frac{\partial \ell(W)}{\partial w_i^k}$$

Dropped w_0 to make derivation simpler

$$\frac{\partial}{\partial x} f(g(x)) = f'(g(x))g'(x)$$

$$\ell(W) = \frac{1}{2}\sum_j [y^j - out(\mathbf{x}^j)]^2$$

$$out(\mathbf{x}) = g\left(\sum_{k'} w_{k'} g(\sum_{i'} w_{i'}^{k'} x_{i'})\right)$$

$$\frac{\partial \ell(W)}{\partial w_i^k} = \sum_{j=1}^{m} -[y - out(\mathbf{x}^j)] \frac{\partial out(\mathbf{x}^j)}{\partial w_i^k}$$

$$\frac{\partial out(\mathbf{x})}{\partial w_i^k} = g'\left(\sum_{k'} w_{k'} g(\sum_{i'} w_{i'}^{k'} x_{i'})\right) \frac{\partial}{\partial w_i^k} g\left(\sum_{i'} w_{i'}^{k'} x_{i'}\right)$$

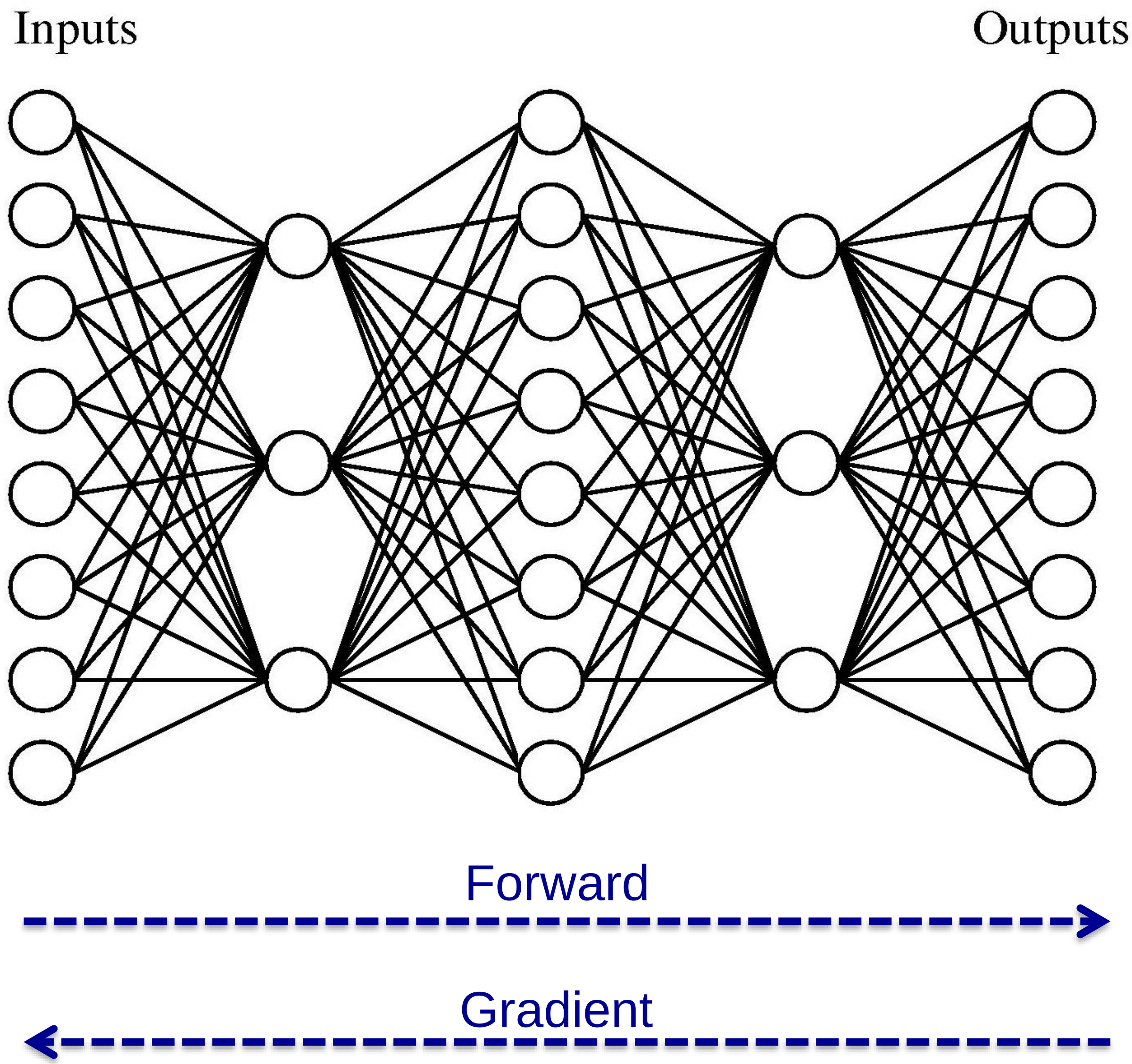
For hidden layer, two parts:

- Normal update for single neuron
- Recursive computation of gradient on output layer

Multilayer neural networks

Inference and Learning:

- Forward pass: left to right, each hidden layer in turn
- Gradient computation: right to left, propagating gradient for each node



Forward propagation – prediction

- Recursive algorithm
- Start from input layer
- Output of node V_k with parents U_1,U_2,…:

$$\boxed{V_k \; = \; g\left(\sum_i w_i^k U_i\right)}$$

Back-propagation – learning

- Just gradient descent!!!
- Recursive algorithm for computing gradient
- For each example
  - Perform forward propagation
  - Start from output layer
    - Compute gradient of node V_k with parents U_1,U_2,...
    - Update weight w_i^k
    - Repeat (move to preceding layer)

Back-propagation – pseudocode

Initialize all weights to small random numbers

- Until convergence, do:
  - For each training example x,y:
    1. Forward propagation, compute node values V_k
    2. For each output unit o (with labeled output y):

       $$\delta_o = V_o(1-V_o)(y-V_o)$$

    3. For each hidden unit h:

       $$\delta_h = V_h(1-V_h) \sum_{k \text{ in output}(h)} w_{h,k}\delta_k$$

    4. Update each network weight $w_{i,j}$ from node i to node j

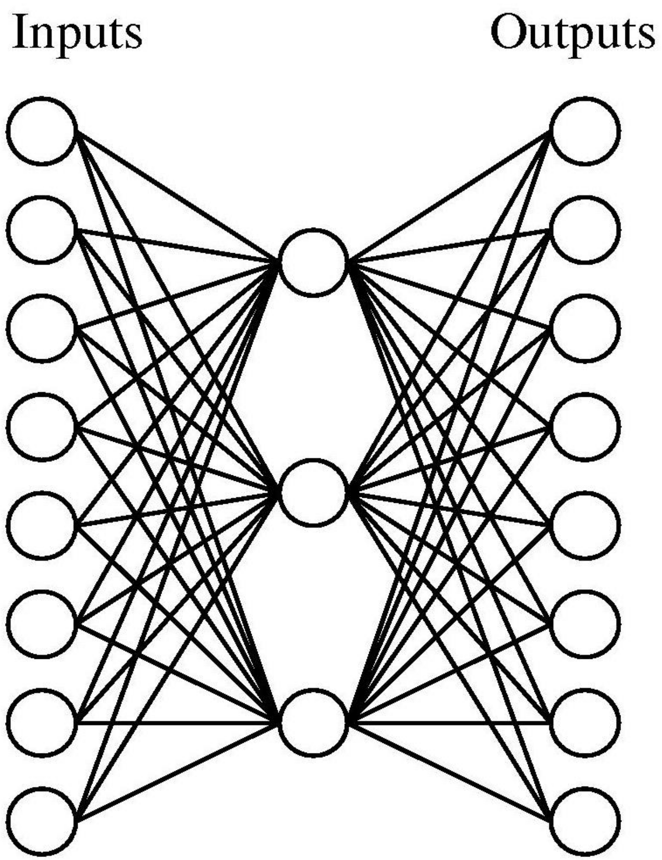
       $$w_{i,j} = w_{i,j} + \eta\delta_j x_{i,j}$$

Convergence of backprop

- Perceptron leads to convex optimization
  - Gradient descent reaches **global minima**
- Multilayer neural nets **not convex**
  - Gradient descent gets stuck in local minima
  - Selecting number of hidden units and layers = fuzzy process
  - NNs have made a HUGE comeback in the last few years!!!
    - Neural nets are back with a new name!!!!
      - Deep belief networks
      - Huge error reduction when trained with lots of data on GPUs

Overfitting in NNs

- Are NNs likely to overfit?
  - Yes, they can represent arbitrary functions!!!
- Avoiding overfitting?
  - More training data
  - Fewer hidden nodes / better topology
  - Regularization
  - Early stopping



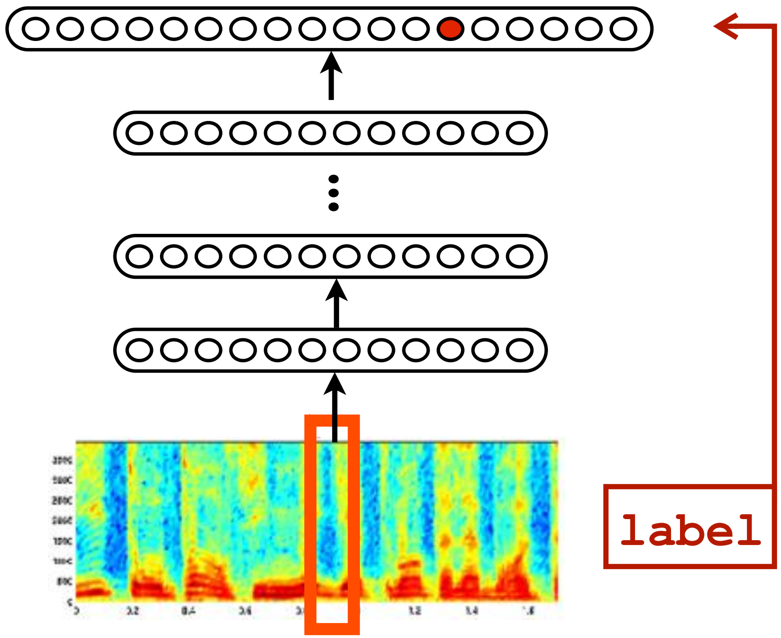
Object Recognition

Slides from Jeff Dean at Google

Number Detection

Slides from Jeff Dean at Google

Acoustic Modeling for Speech Recognition



Close collaboration with Google Speech team

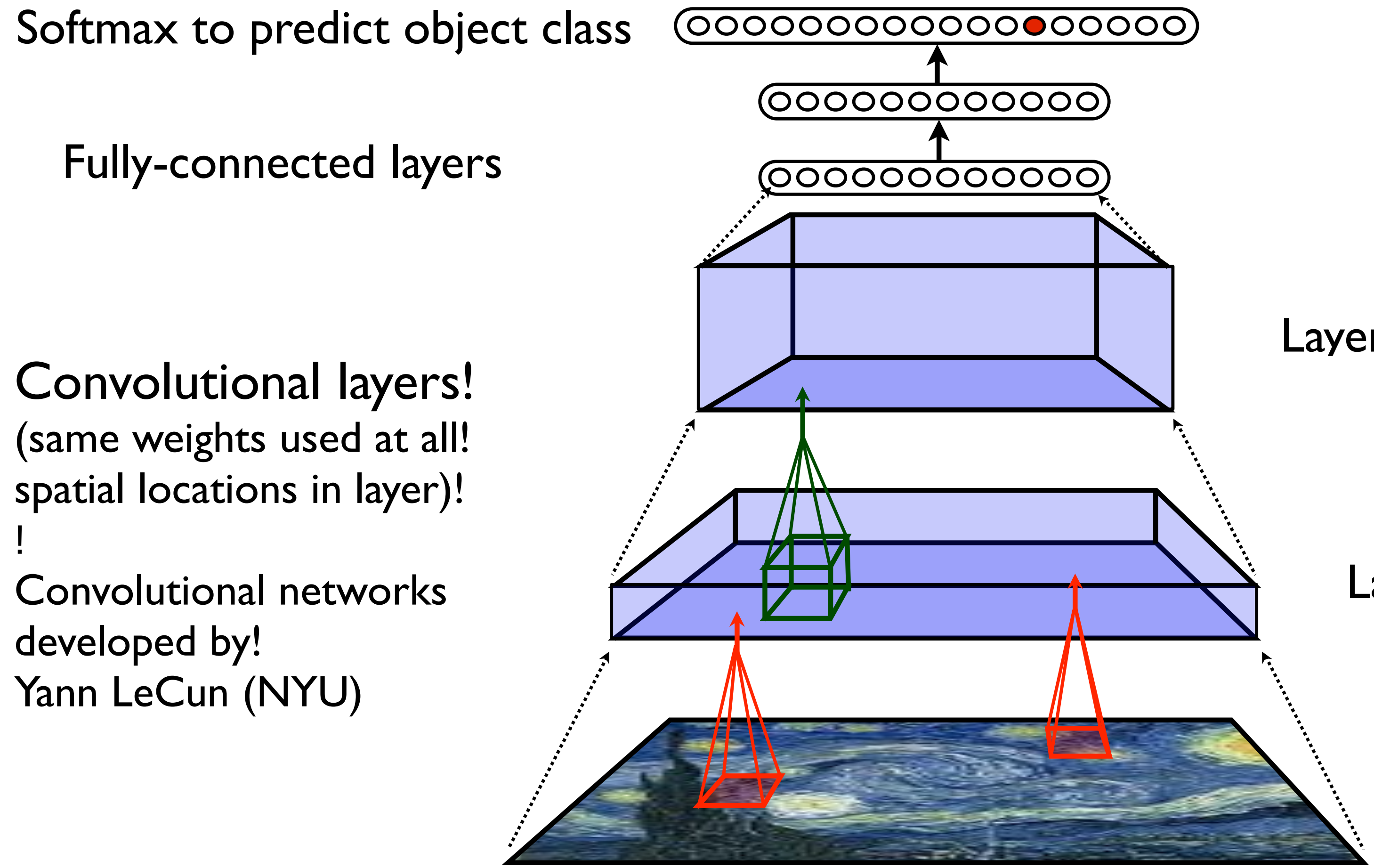
Trained in <5 days on cluster of 800 machines

30% reduction in Word Error Rate for English!
("biggest single improvement in 20 years of speech research")

Launched in 2012 at time of Jellybean release of Android

Slides from Jeff Dean at Google

2012-era Convolutional Model for Object Recognition



Convolutional layers!
(same weights used at all!
spatial locations in layer)!
!
Convolutional networks
developed by!
Yann LeCun (NYU)

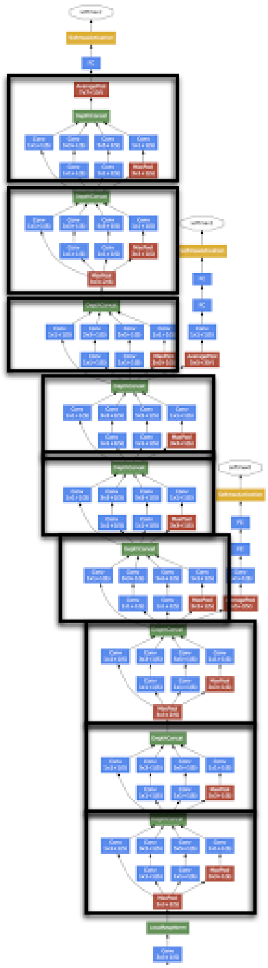
Basic architecture developed by Krizhevsky, Sutskever & Hinton
(all now at Google).!

Won 2012 ImageNet challenge with 16.4% top-5 error rate

Slides from Jeff Dean at Google

2014-era Model for Object Recognition

Module with 6 separate!
convolutional layers



24 layers deep!

Developed by team of Google Researchers:!

Won 2014 ImageNet challenge with 6.66% top-5 error rate

Slides from Jeff Dean at Google

Good Fine-grained Classification

"hibiscus"

"dahlia"

Slides from Jeff Dean at Google

Good Generalization

Both recognized as a "meal"

Slides from Jeff Dean at Google

Sensible Errors

“snake”

“dog”

Slides from Jeff Dean at Google

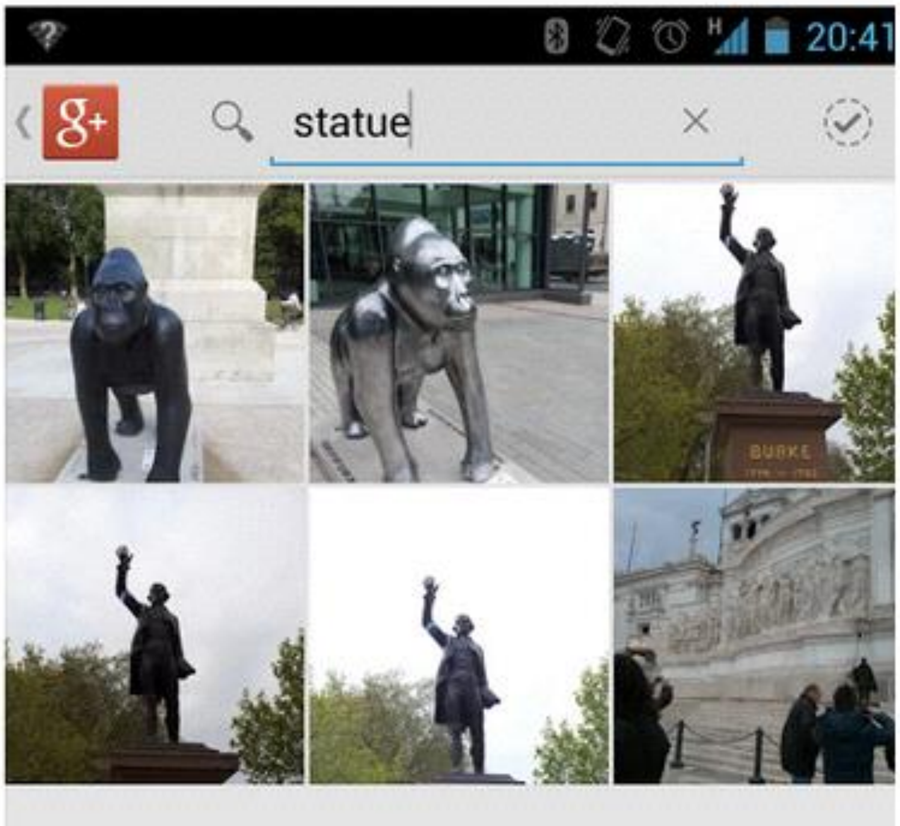
Works in practice

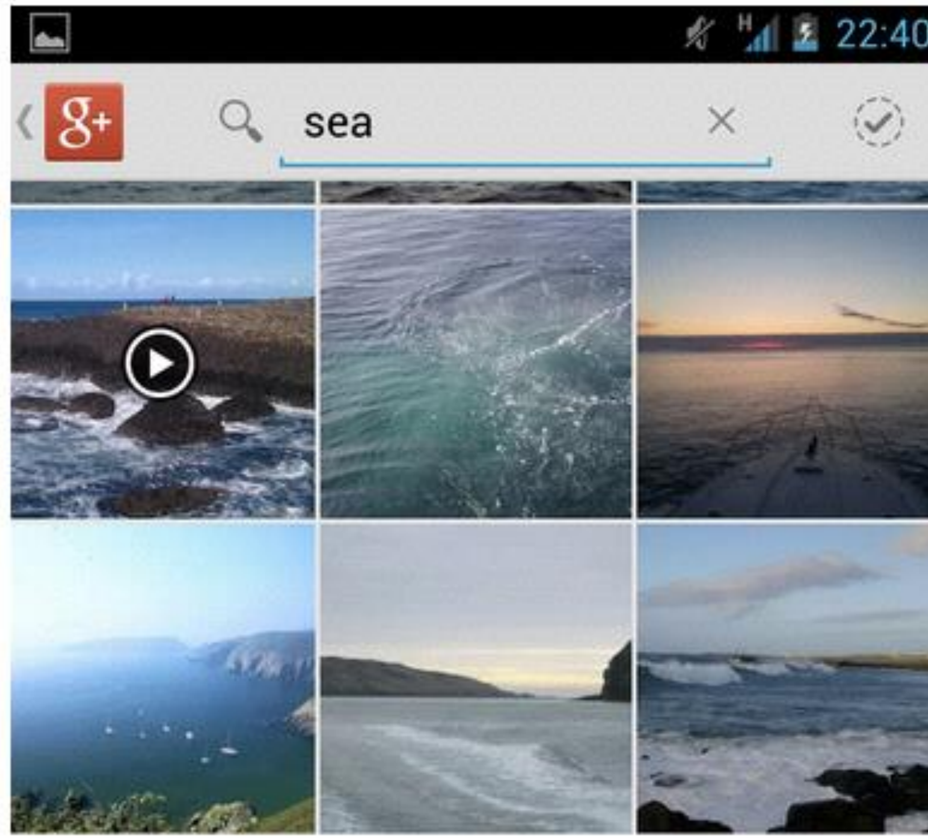
for real users.

Wow.

The new Google plus photo search is a bit insane.

I didn't tag those... :)

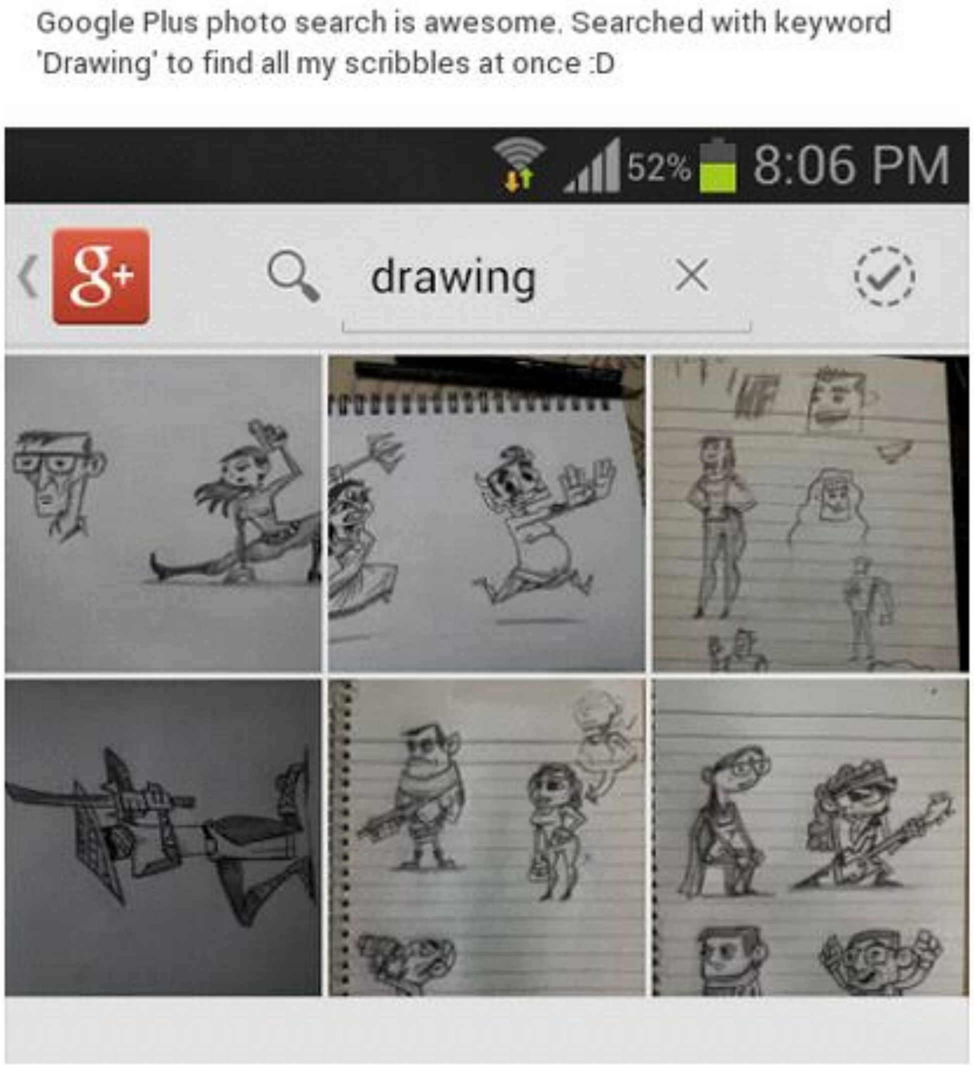


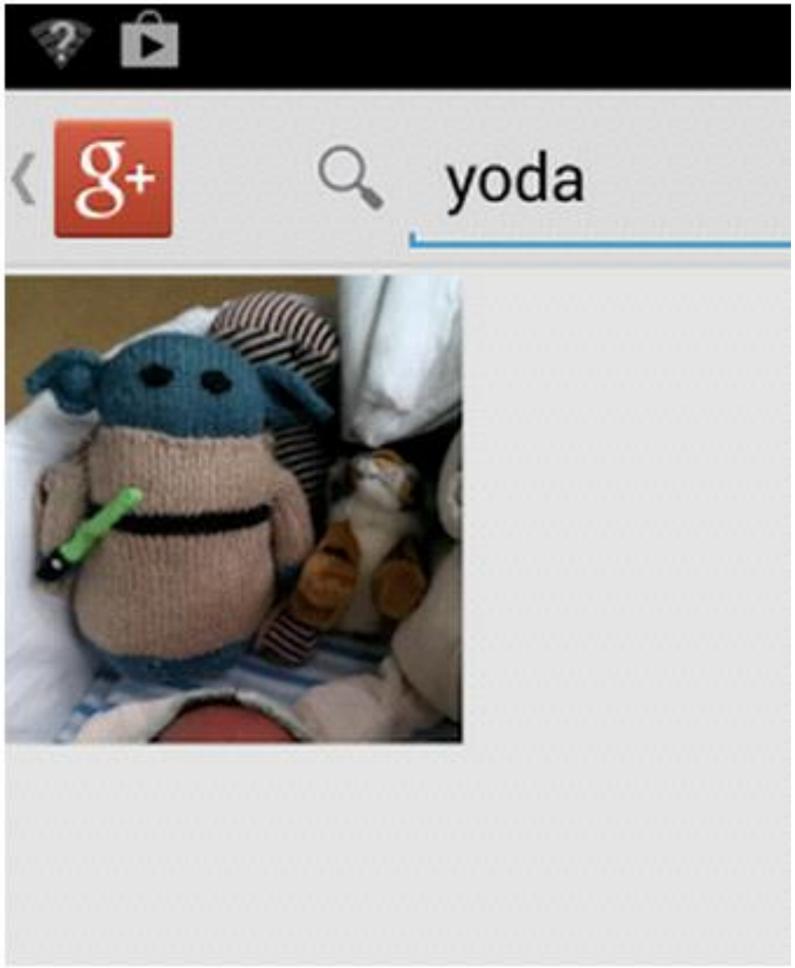


Slides from Jeff Dean at Google

Works in practice

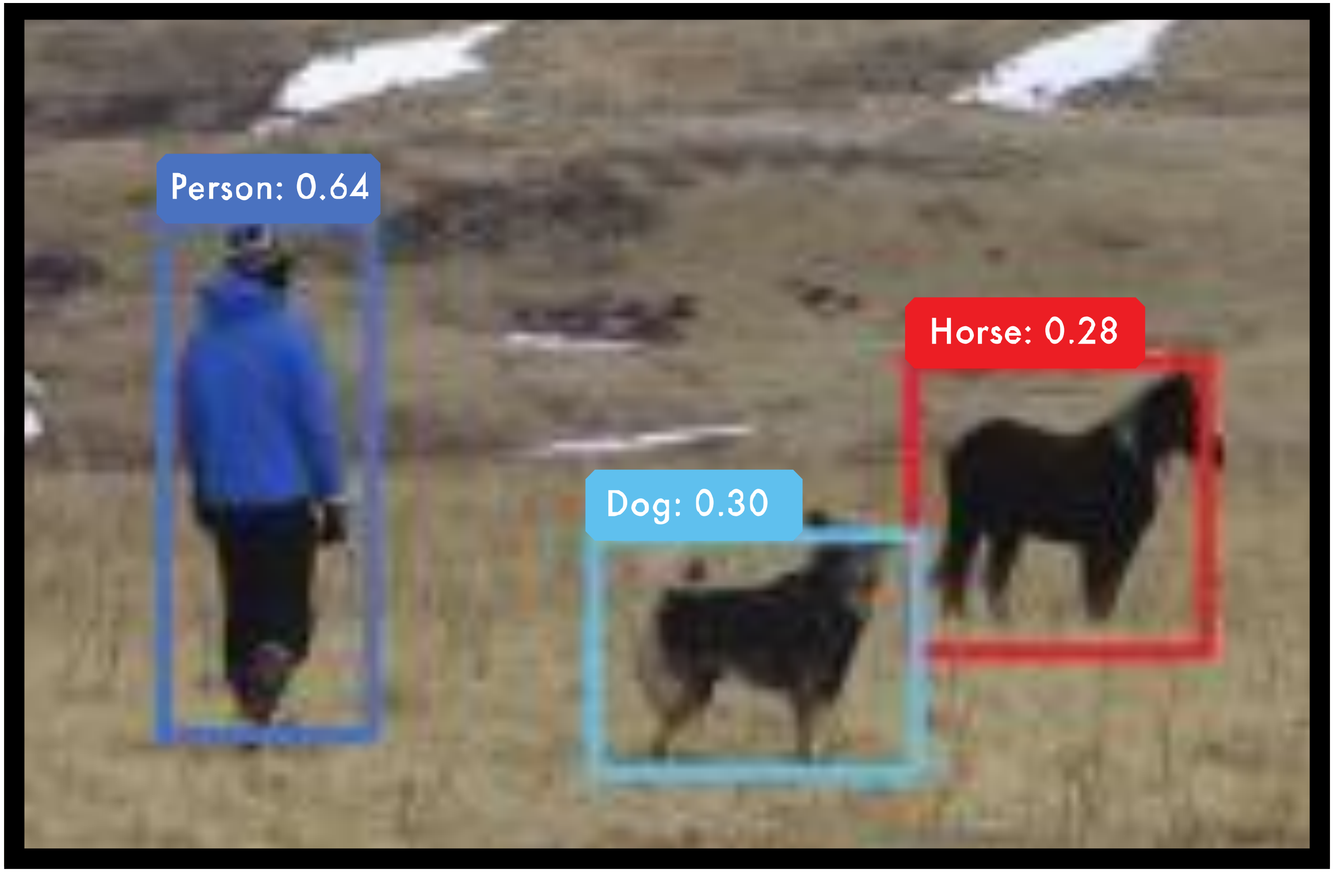
for real users.



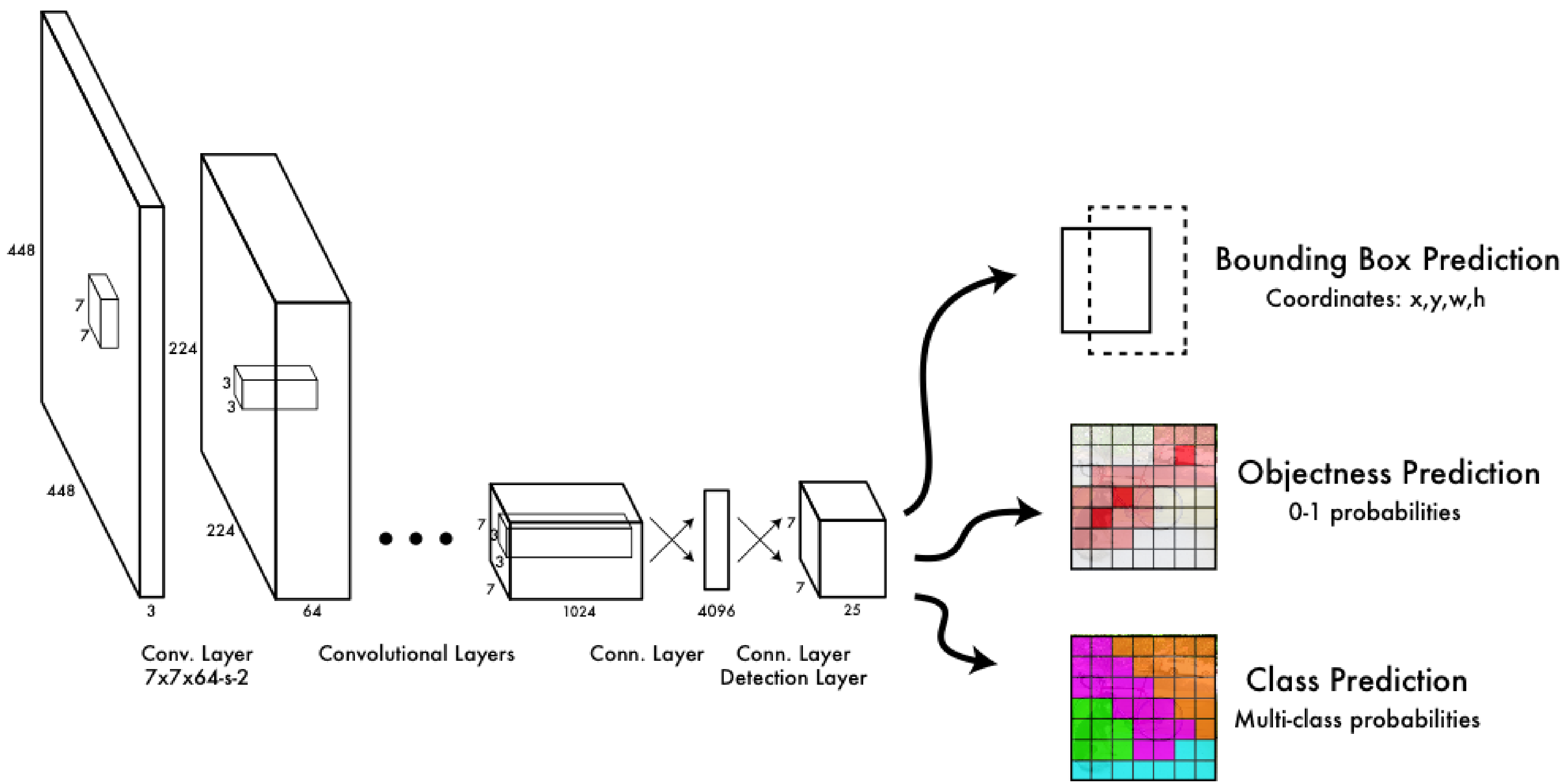


Slides from Jeff Dean at Google

Object Detection



YOLO


448
7
7
224
3
3
448
224
7
3
3
7
7
7
3
64
1024
4096
25
Conv. Layer
7x7x64-s-2
Convolutional Layers
Conn. Layer
Conn. Layer
Detection Layer
Bounding Box Prediction
Coordinates: x,y,w,h
Objectness Prediction
0-1 probabilities
Class Prediction
Multi-class probabilities

DEMO

What you need to know about neural networks

- Perceptron:
  - Relationship to general neurons
- Multilayer neural nets
  - Representation
  - Derivation of backprop
  - Learning rule
- Overfitting