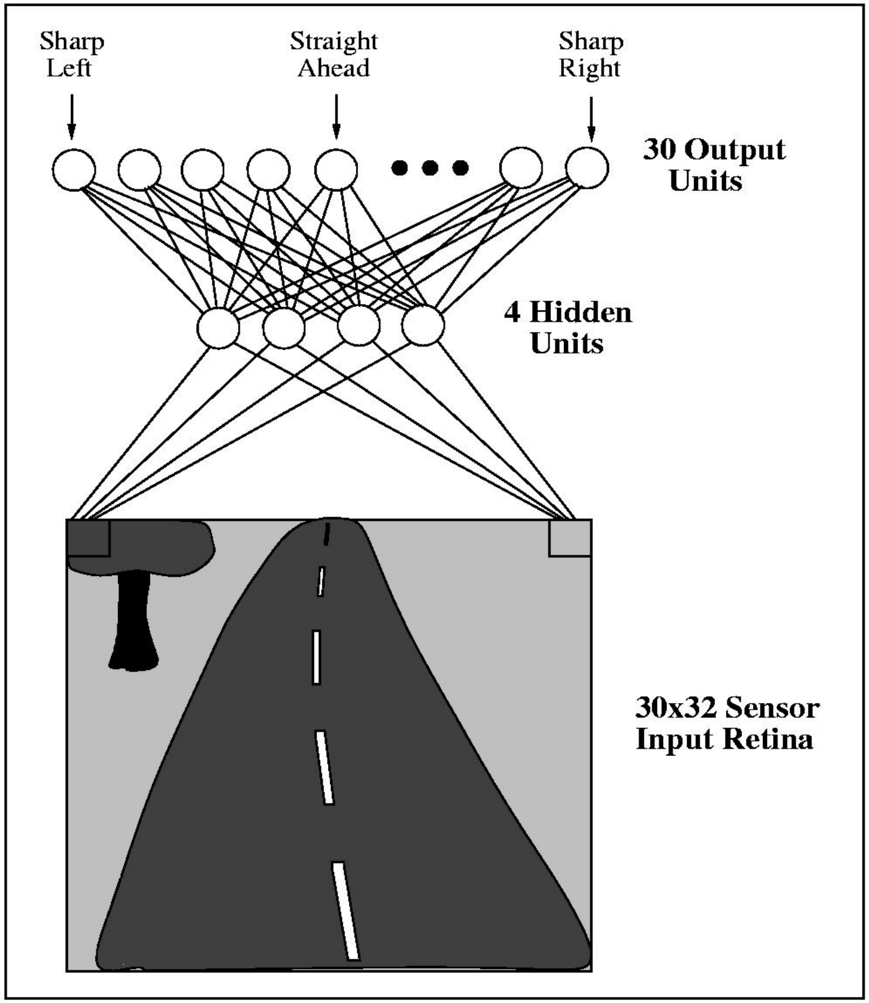
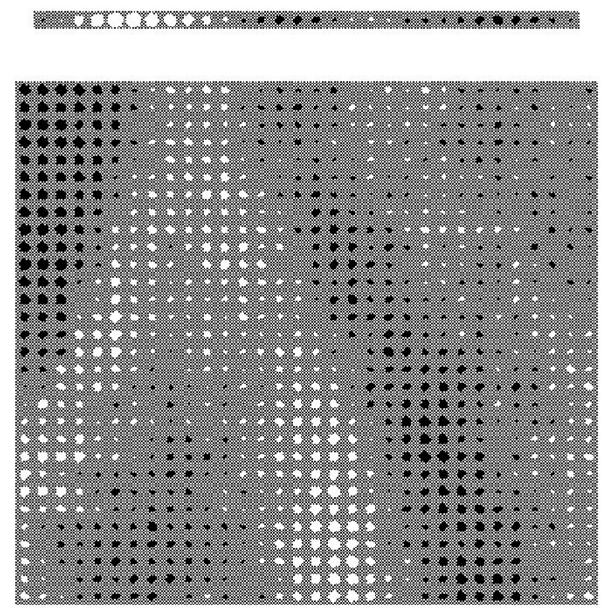
CSE546: Neural Networks
Winter 2012

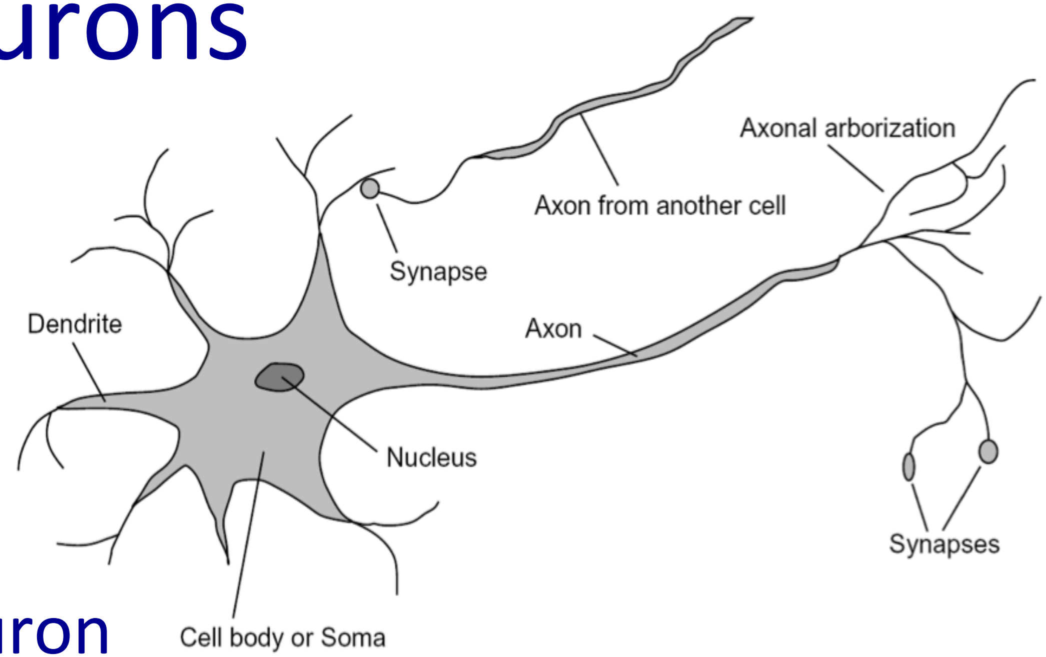
Luke Zettlemoyer

Slides adapted from Carlos Guestrin


Sharp Left
Straight Ahead
Sharp Right
30 Output Units
4 Hidden Units
30x32 Sensor Input Retina



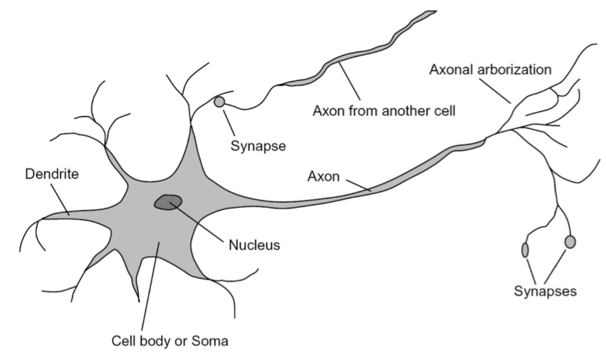
Human Neurons



- 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!

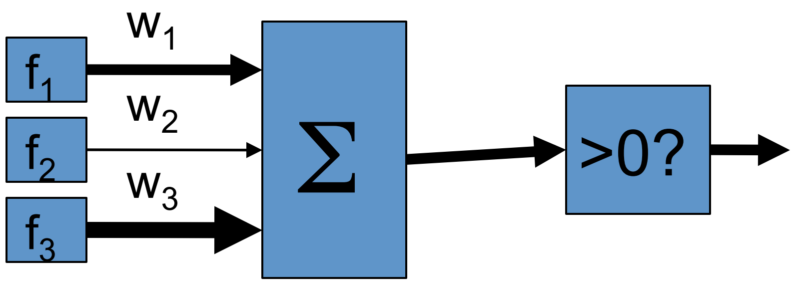
Review: Linear Classifiers as Activation

- Inputs are feature values
- Each feature has a weight
- Sum is the activation



$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$

- If the activation is:
  - Positive, output *class 1*
  - Negative, output *class 2*



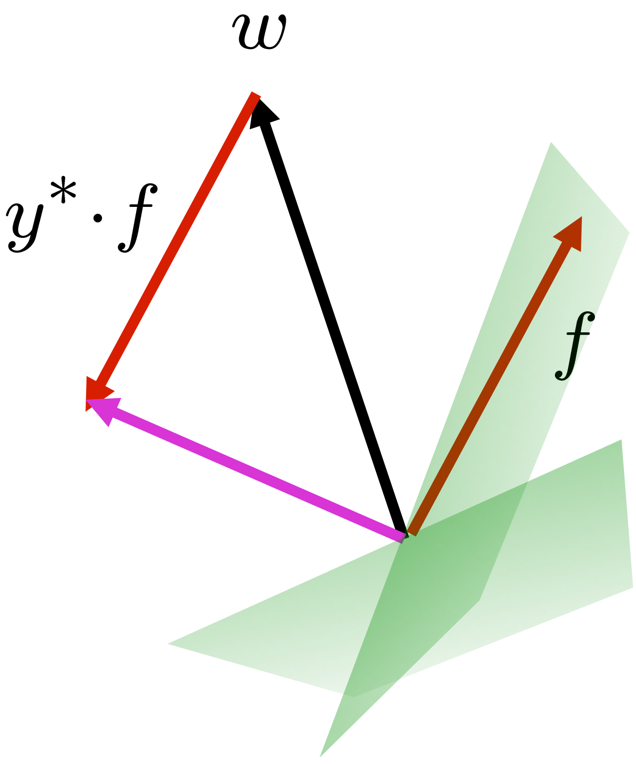
Review: Binary Perceptron Algorithm

- Start with zero weights
- For each training instance (x,y*):
  - Classify with current weights

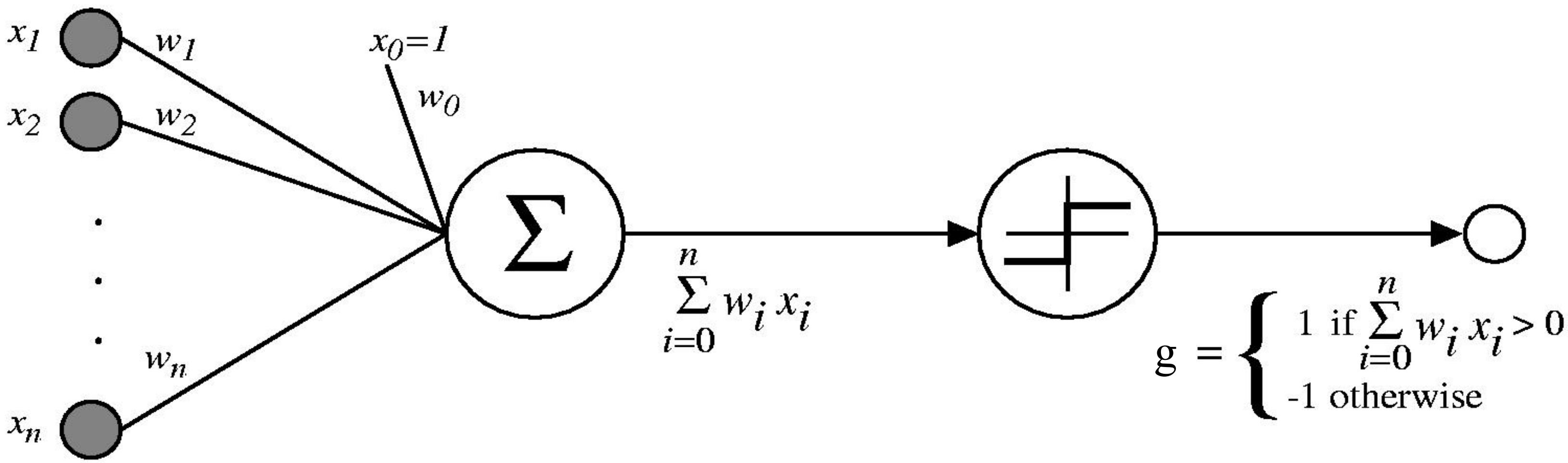
$$y = \begin{cases} +1 & \text{if } \; w \cdot f(x) \geq 0 \\ -1 & \text{if } \; w \cdot f(x) < 0 \end{cases}$$

  - If correct (i.e., y=y*), no change!
  - If wrong: update

$$w = w + y^* f(x)$$



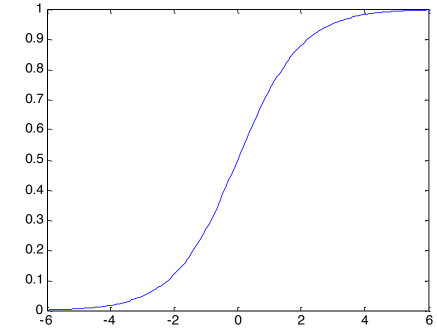
Perceptron as a Neural Network



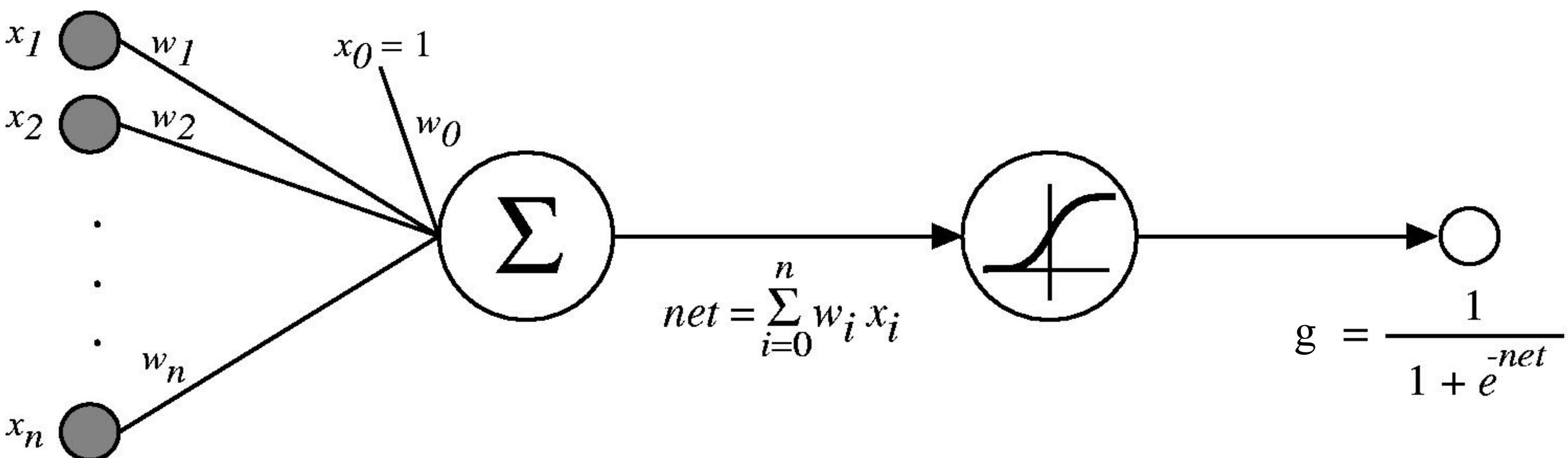
This is one neuron:

- 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

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

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

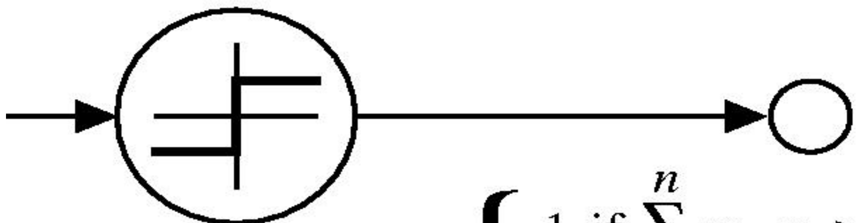
$$\frac{\partial}{\partial w_i} g(w_0 + \sum_i w_i x_i^j) = x_i^j \frac{\partial}{\partial w_i} g(w_0 + \sum_i w_i x_i^j) = x_i^j g'(w_0 + \sum_i w_i x_i^j)$$

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

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

Re-deriving the perceptron update

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



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

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

For a specific, incorrect example:

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

Sigmoid units: have to differentiate g

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

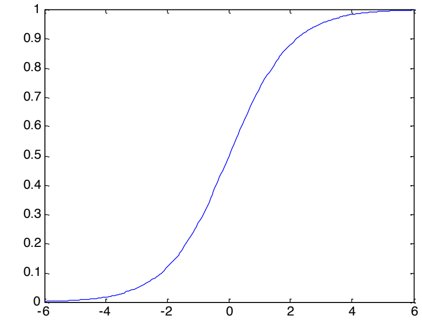
$$g(x) = \frac{1}{1 + e^{-x}} \qquad g'(x) = g(x)(1 - g(x))$$

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

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

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

Aside: Comparison to logistic regression



- P(Y|X) represented by:

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

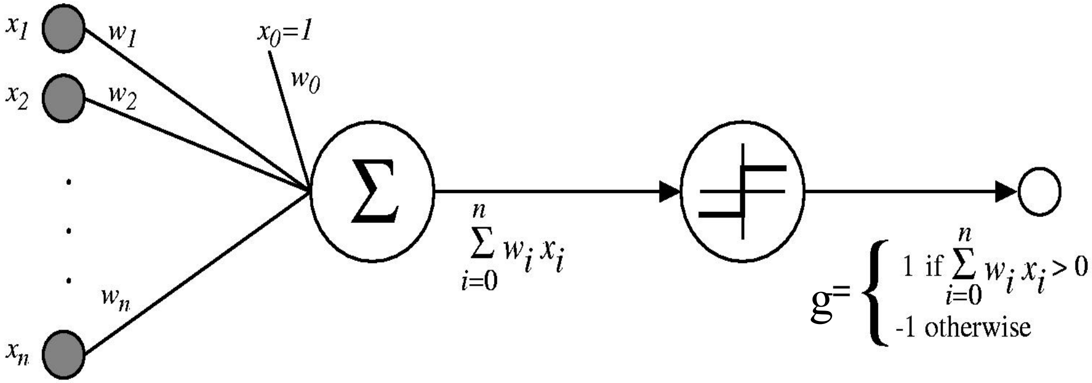
- Learning rule – MLE:

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

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

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

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



- Can learn $x_1 \vee x_2$?
  - $0.5 + x_1 + x_2$
- Can learn $x_1 \wedge x_2$?
  - $-1.5 + x_1 + x_2$
- 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.

Going beyond linear classification

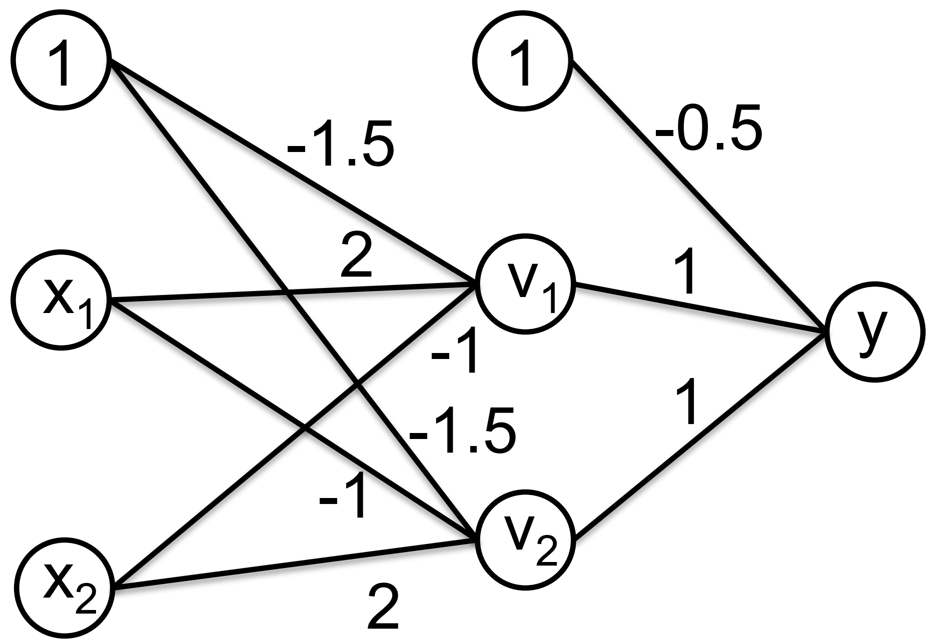
Solving the XOR problem

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

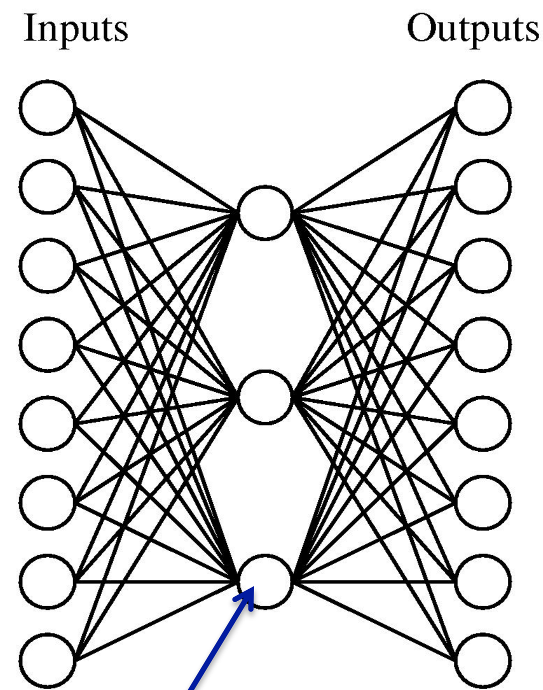
$v_1 = (x_1 \wedge \neg x_2)$
$= -1.5 + 2x_1 - x_2$

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

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



Hidden layer



- Single unit:

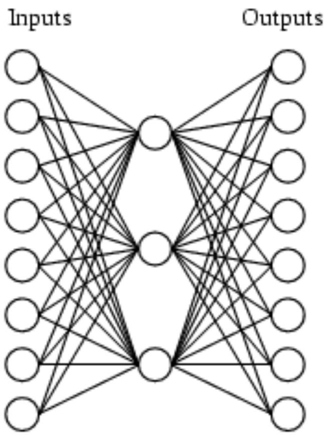
$$out(\mathbf{x}) = g(w_0 + \sum_i w_i x_i)$$

- 1-hidden layer:

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

- No longer convex function!

Example data for NN with hidden layer



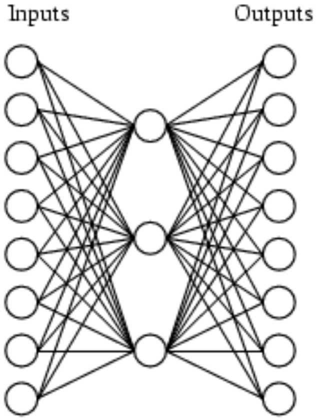
A target function:

Input		Output
10000000	→	10000000
01000000	→	01000000
00100000	→	00100000
00010000	→	00010000
00001000	→	00001000
00000100	→	00000100
00000010	→	00000010
00000001	→	00000001

Can this be learned??

Learned weights for hidden layer

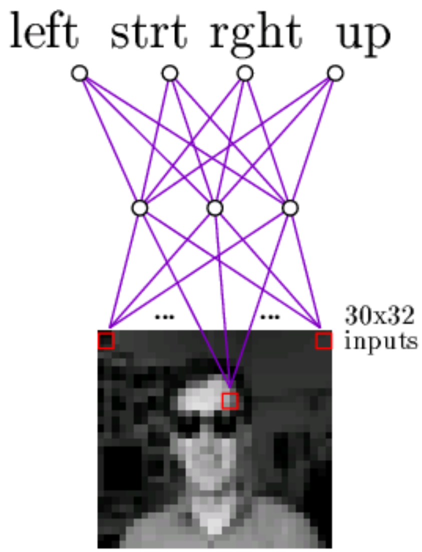
A network:



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

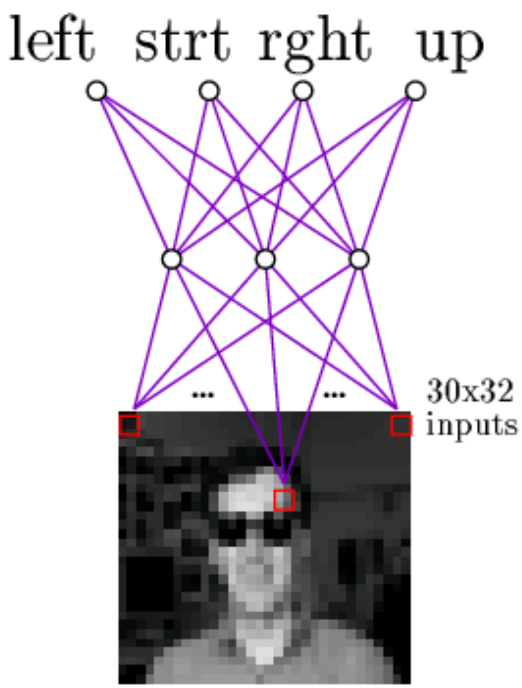
NN for images

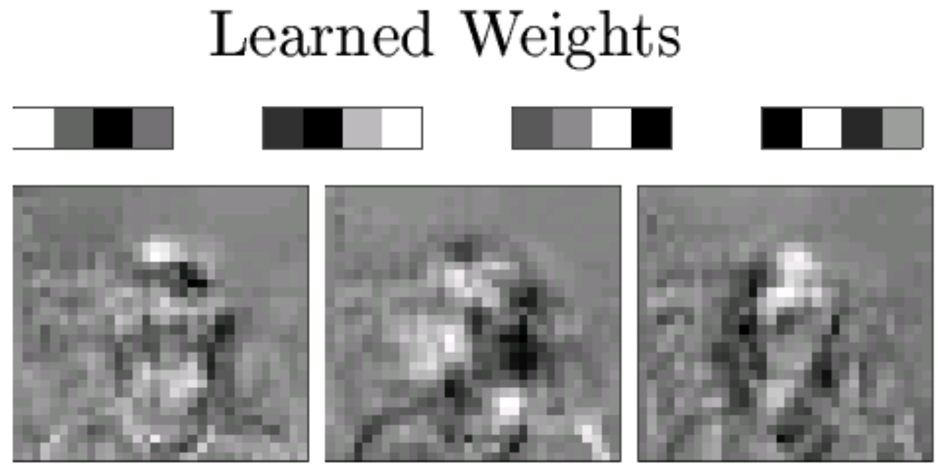


Typical input images

90% accurate learning head pose, and recognizing 1-of-20 faces

Weights in NN for images





Typical input images

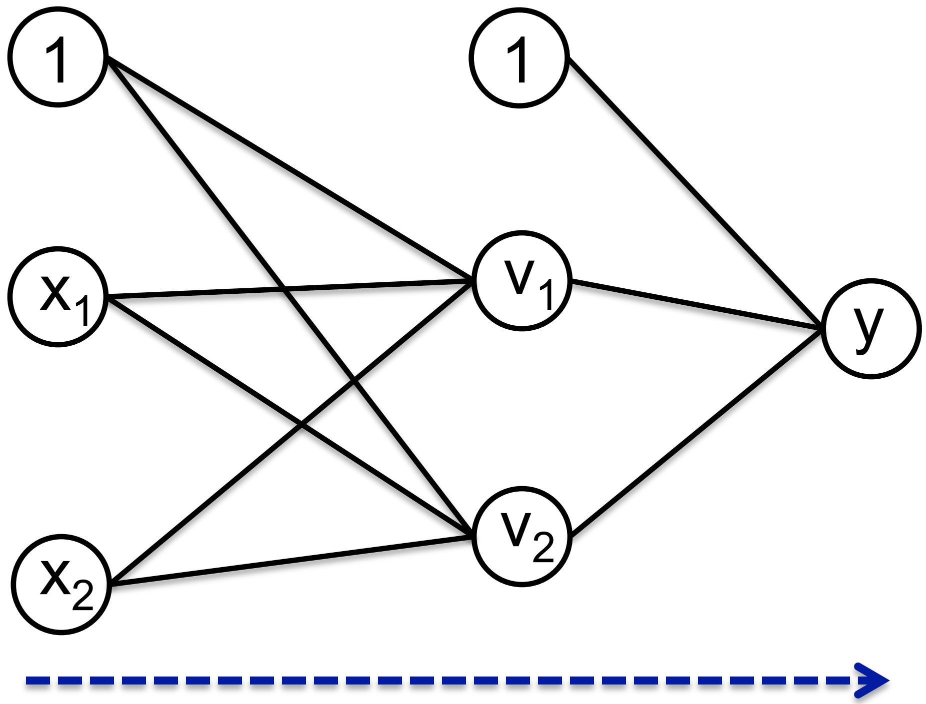
Forward propagation

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



Gradient descent for 1-hidden layer – Back-propagation: Computing $\frac{\partial \ell(W)}{\partial w_k}$

Dropped w_0 to make derivation simpler

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

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

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

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

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

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

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

Gradient descent for 1-hidden layer – Back-propagation: Computing $\frac{\partial \ell(W)}{\partial w_i^k}$

Dropped w_0 to make derivation simpler

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

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

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

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

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

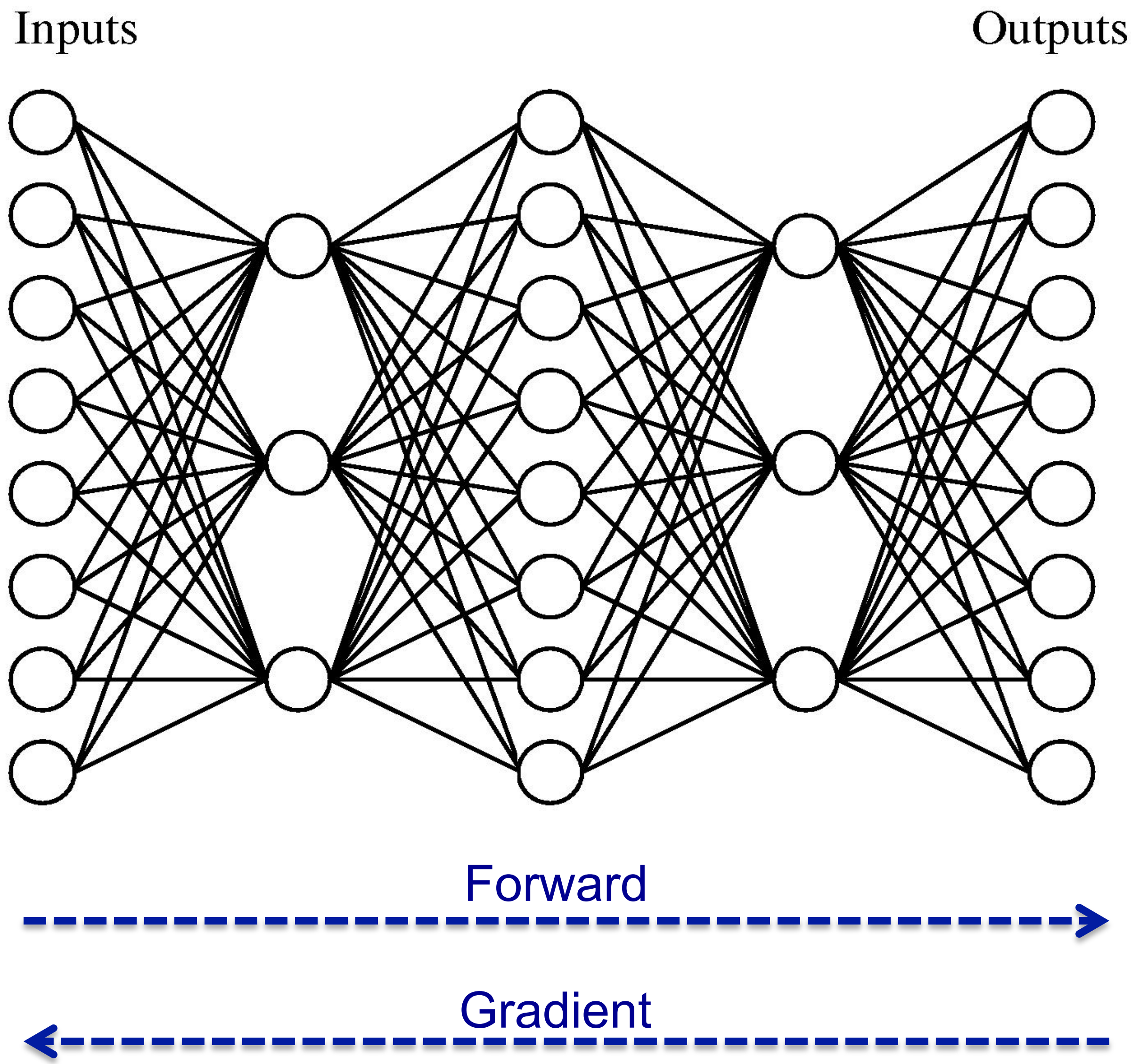
For hidden layer, two parts:

- 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,…:

$$V_k = g\left(\sum_i w_i^k U_i\right)$$

Back-propagation – learning

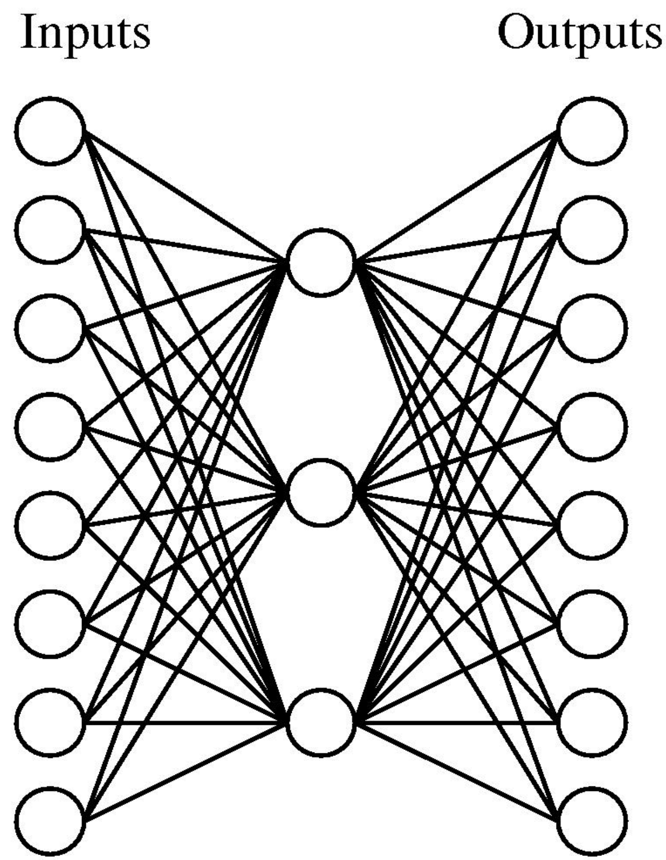
- Just gradient descent!!!
- Recursive algorithm for computing gradient
- For each example
  - Perform forward propagation
  - Start from output layer
    - Compute gradient of node V_k with parents U_1,U_2,…
    - Update weight w_i^k
    - Repeat (move to preceeding layer)

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 falling in disfavor in last few years
  - *Kernel trick* is considered a good alternative
  - Nonetheless, neural nets are one of the most used ML approaches
    - Plus, neural nets are back with a new name!!!!
      - Deep belief networks
        - » (and a probabilistic interpretation & different learning procedure)

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



What you need to know about neural networks

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