

# 15-494/694: Cognitive Robotics

Dave Touretzky

Lecture 12:  
Backpropagation Learning

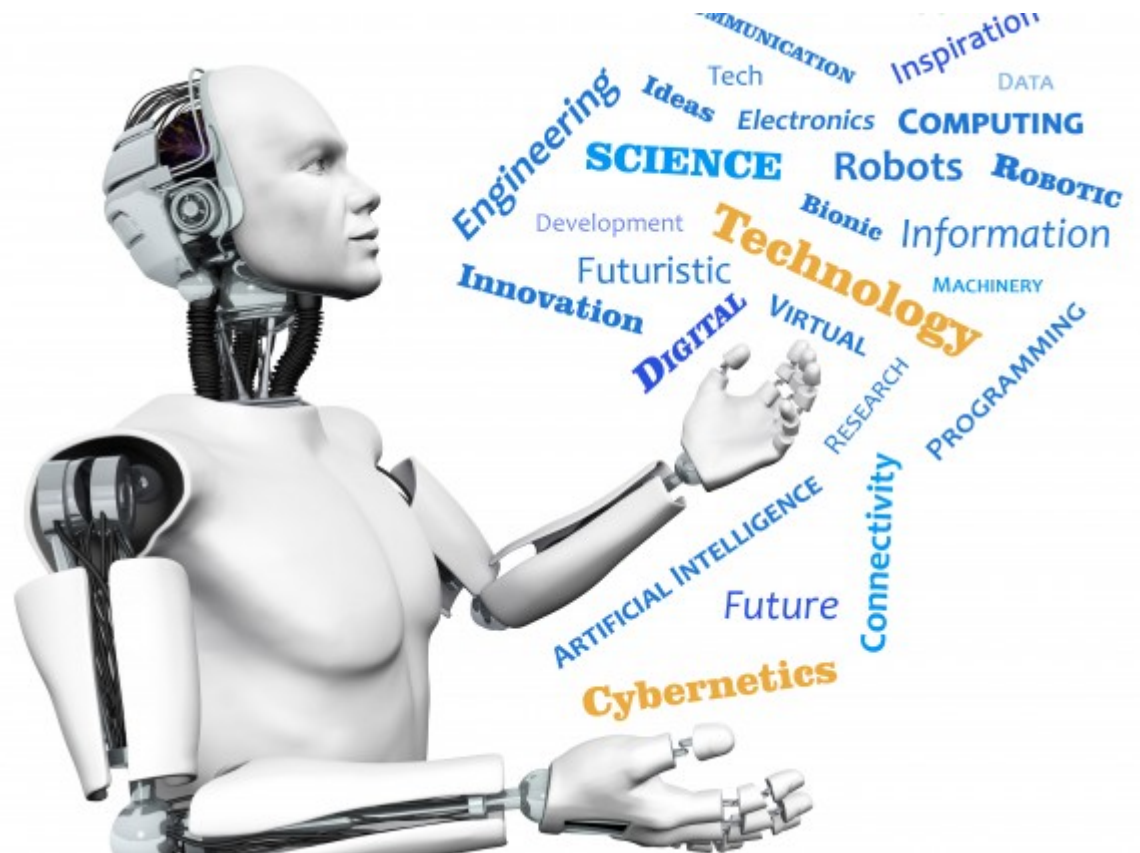
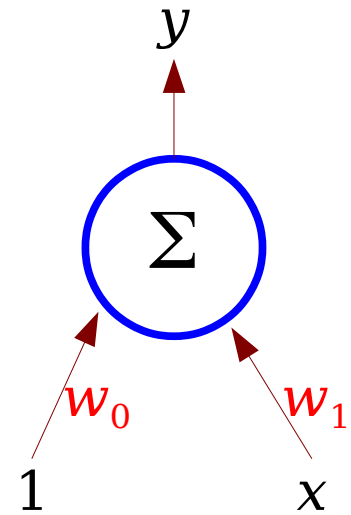
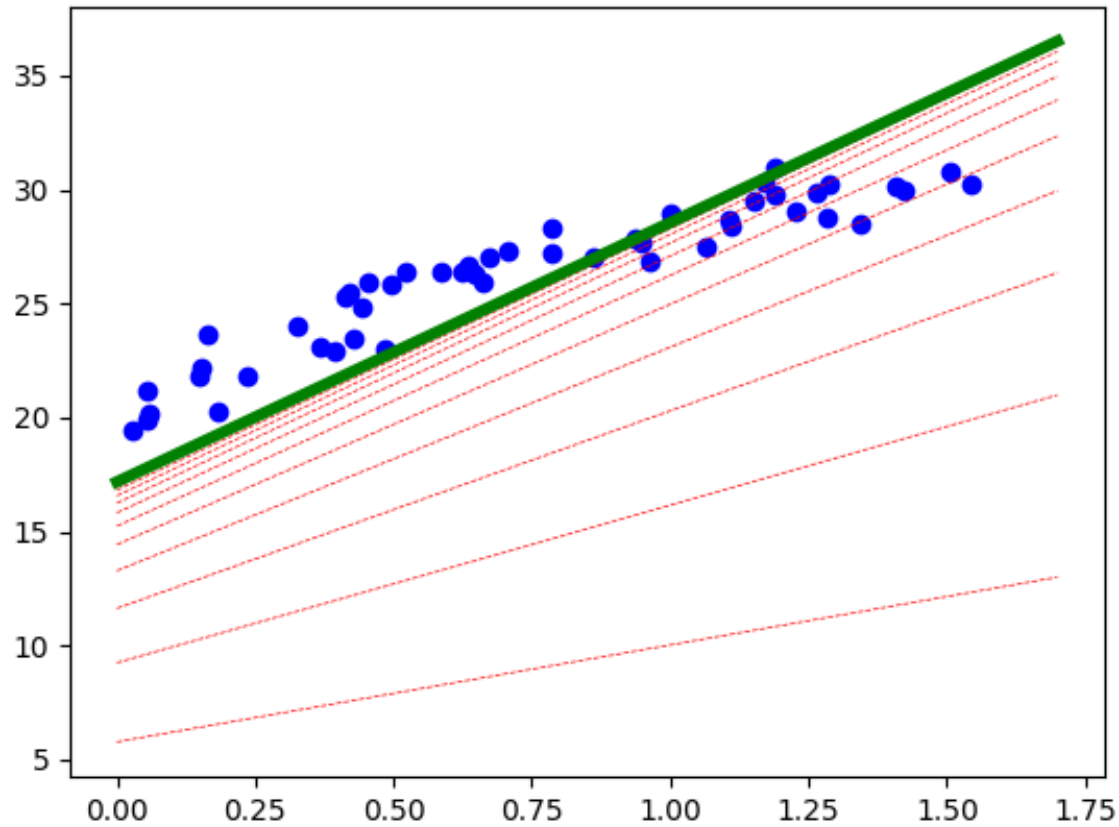


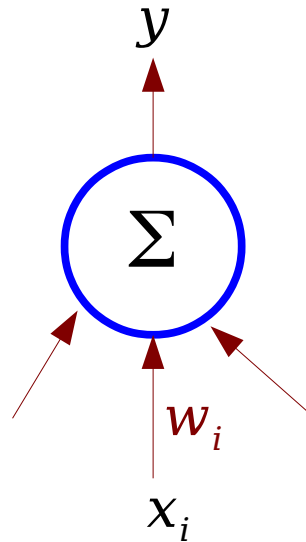
Image from <http://www.futuristgerd.com/2015/09/10>

# Training A Linear Unit

$$y = w_0 + w_1 \cdot x$$



# LMS / Widrow-Hoff Rule

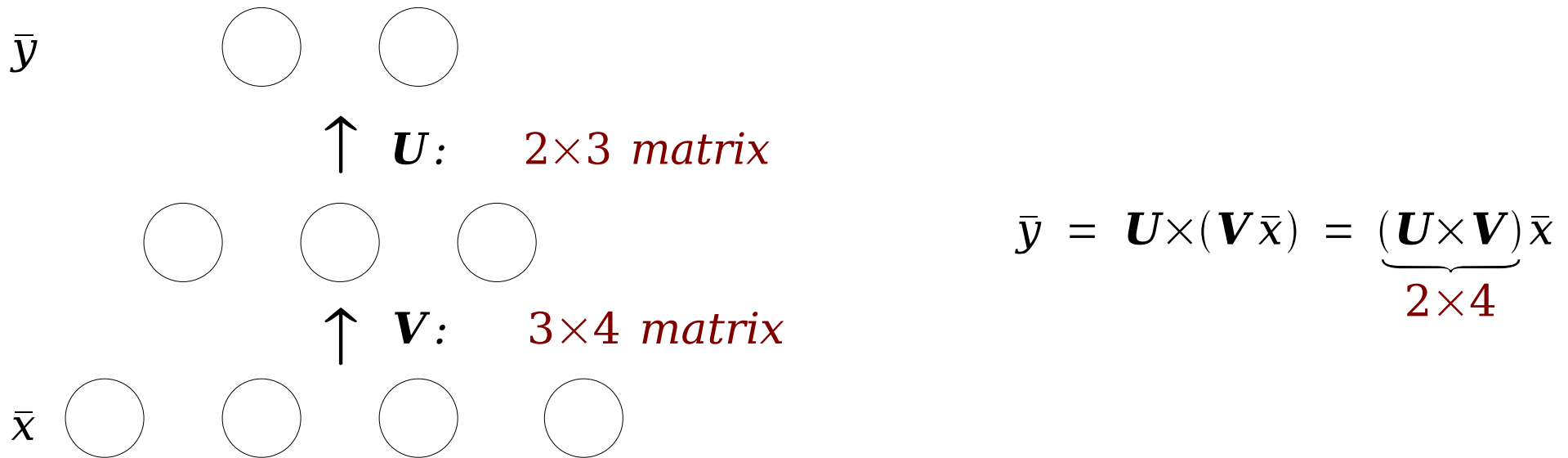


$$\Delta w_i = -\eta(y-d)x_i$$

$\eta$  is a learning rate

Works fine for a single layer of trainable weights.  
What about multi-layer networks?

# With Linear Units, Multiple Layers Don't Add Anything



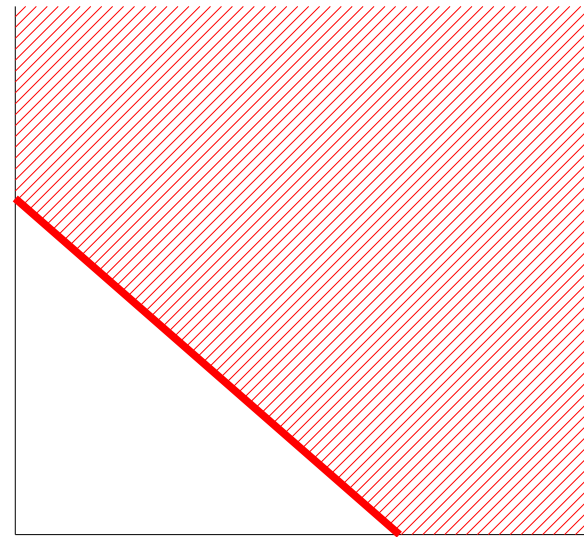
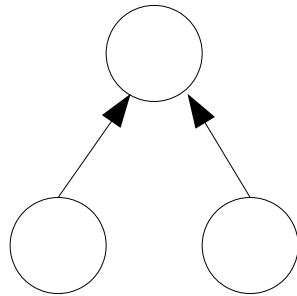
*Linear operators are closed under composition.  
Equivalent to a single layer of weights  $\mathbf{W} = \mathbf{U} \times \mathbf{V}$*

*But with non-linear units, extra layers add  
computational power.*

# What Can be Done with Non-Linear (e.g., Threshold) Units?

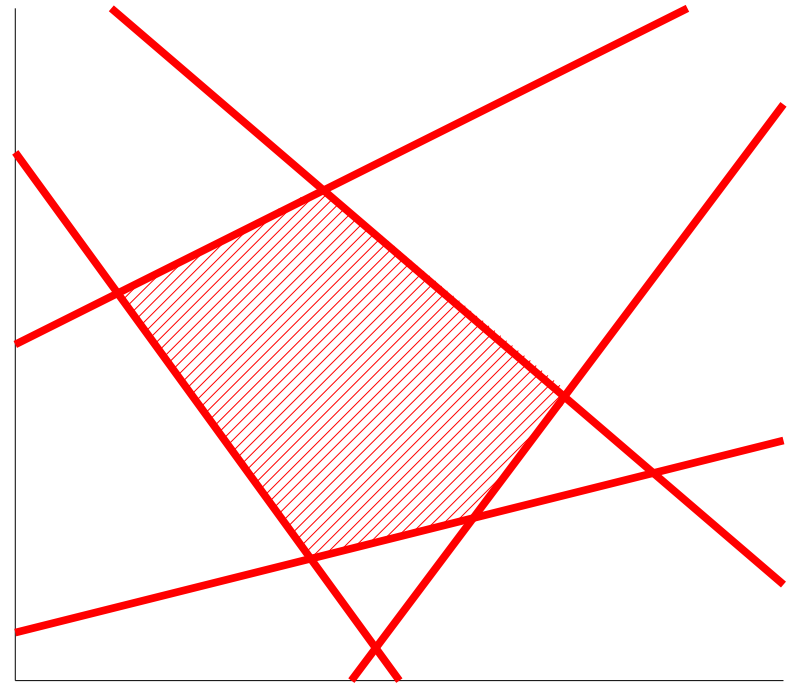
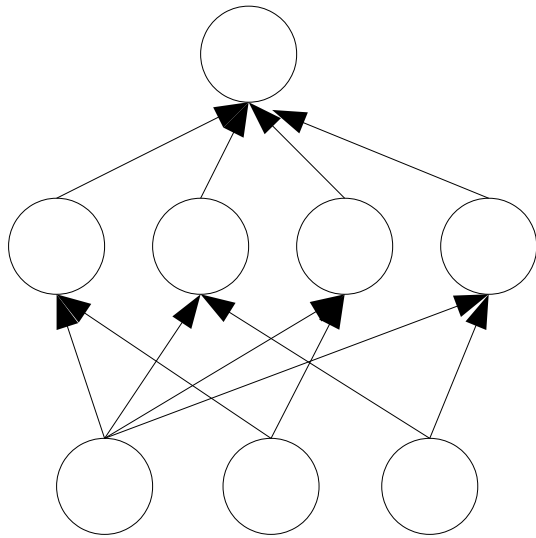
$$y = w_0 + w_1 \cdot x_1 + w_2 \cdot x_2$$

1 layer of trainable weights



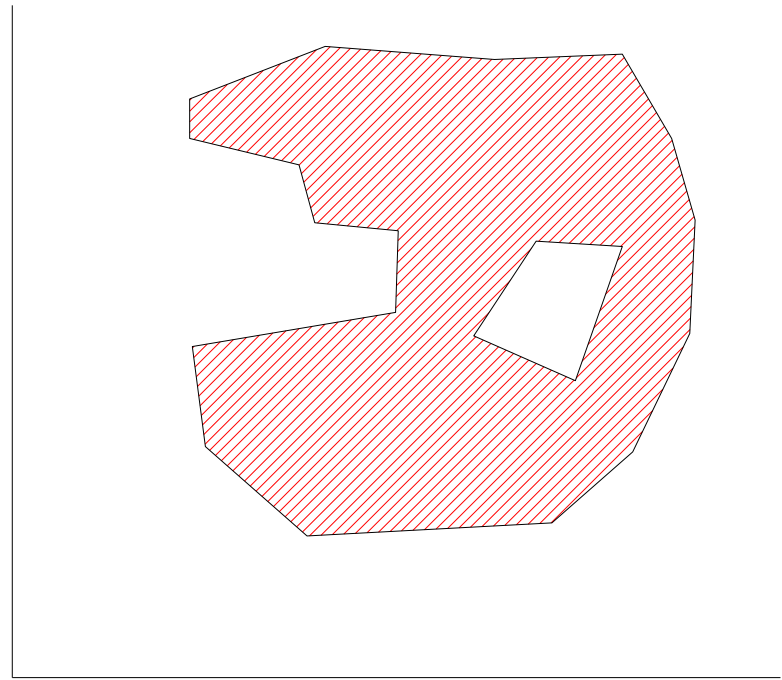
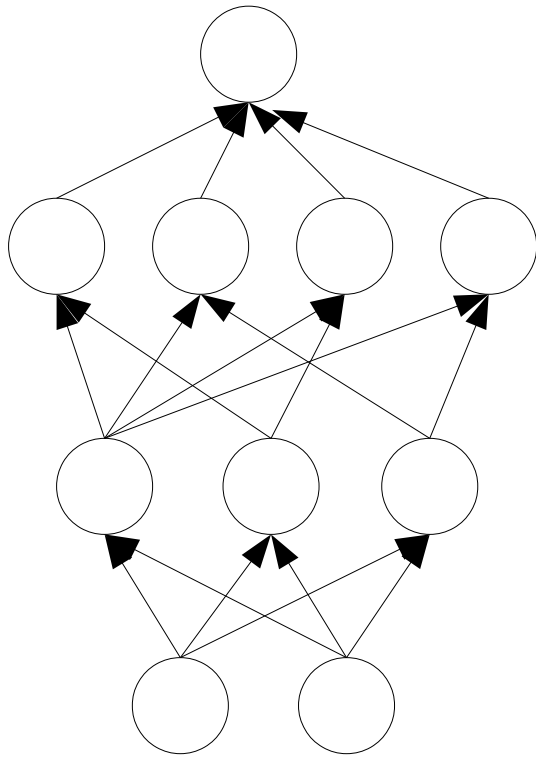
separating hyperplane

2 layers of trainable weights



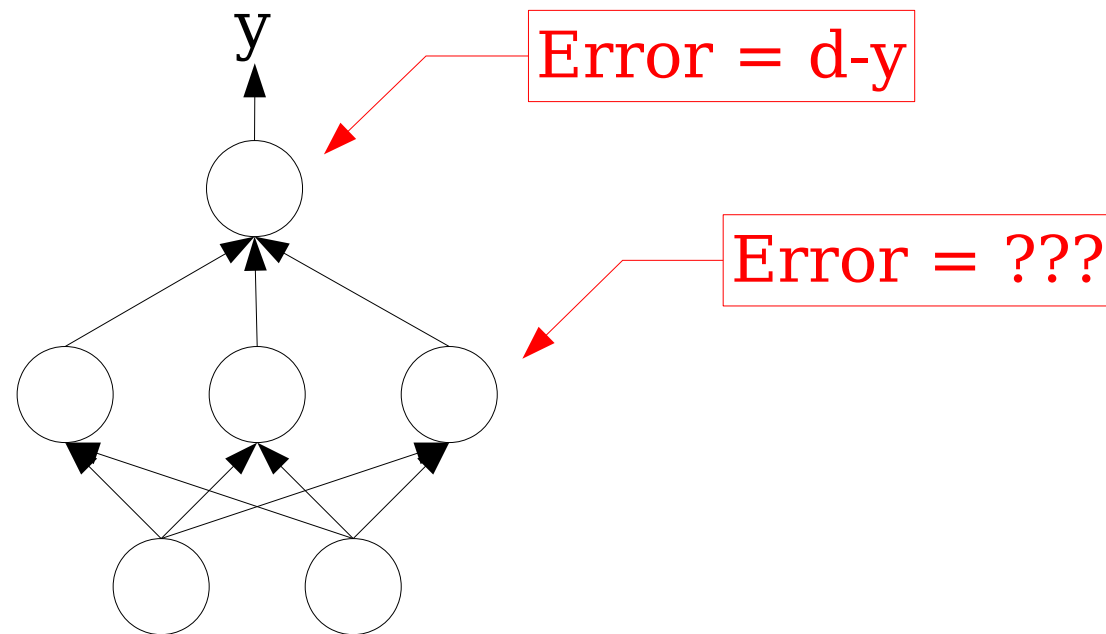
convex polygon region

3 layers of trainable weights



composition of polygons:  
convex regions

# How Do We Train A Multi-Layer Network?



Can't use perceptron training algorithm because we don't know the 'correct' outputs for hidden units.



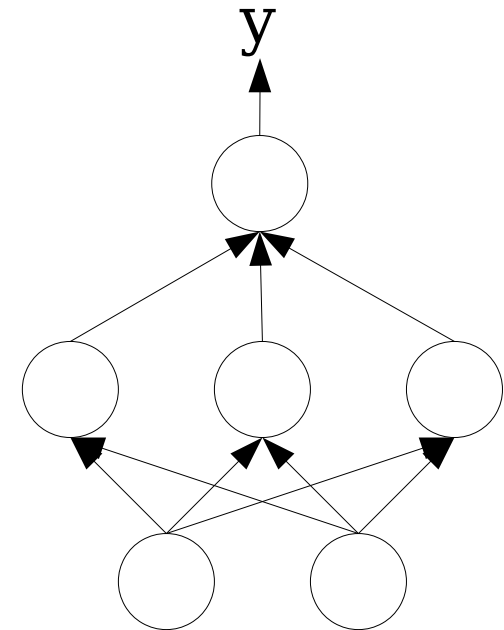
# How Do We Train A Multi-Layer Network?

*Define sum-squared error:*

$$E = \frac{1}{2} \sum_p (d^p - y^p)^2$$

*Use gradient descent error minimization:*

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$



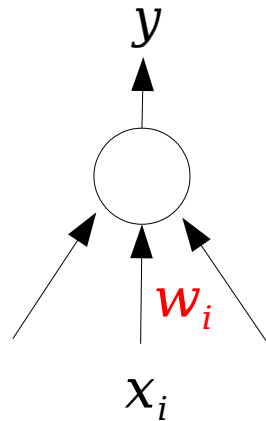
*Works if the nonlinear transfer function is differentiable.*

# Deriving the LMS or “Delta” Rule As Gradient Descent Learning

$$y = \sum_i w_i x_i$$

$$E = \frac{1}{2} \sum_p (d^p - y^p)^2$$

$$\frac{dE}{dy} = y - d$$



$$\frac{\partial E}{\partial w_i} = \frac{dE}{dy} \cdot \frac{\partial y}{\partial w_i} = (y - d) x_i$$

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} = -\eta (y - d) x_i$$

*How do we extend this to two layers?*

# Switch to Smooth Nonlinear Units

$$\text{net}_j = \sum_i w_{ij} y_i$$

$$y_j = g(\text{net}_j) \quad g \text{ must be differentiable}$$

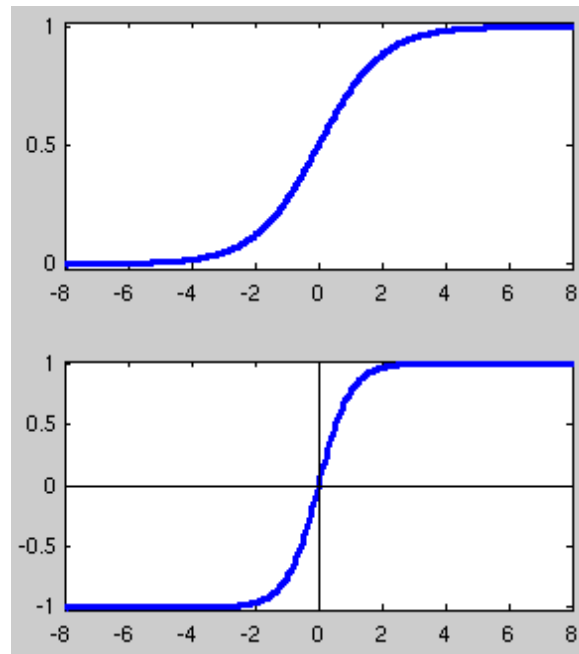
Common choices for  $g$ :

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

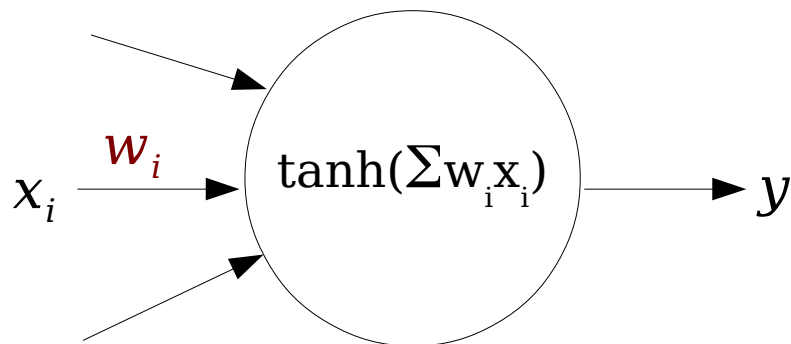
$$g'(x) = g(x) \cdot (1 - g(x))$$

$$g(x) = \tanh(x)$$

$$g'(x) = 1 / \cosh^2(x)$$



# Gradient Descent with Nonlinear Units

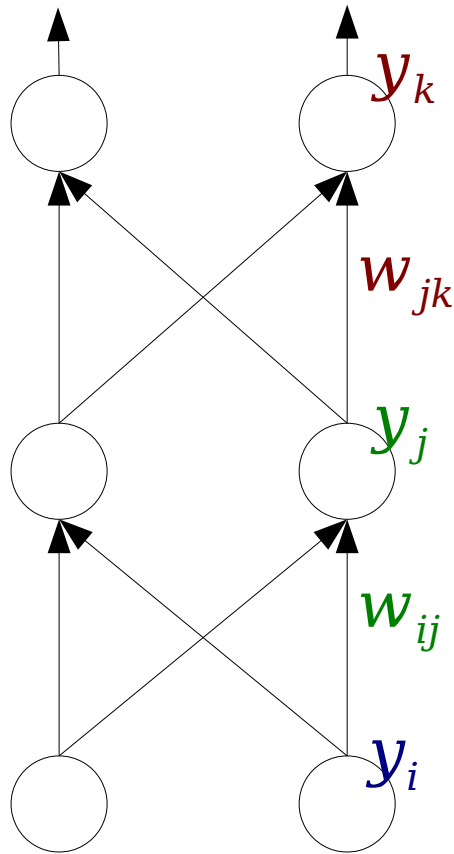


$$y = g(\text{net}) = \tanh\left(\sum_i w_i x_i\right)$$

$$\frac{dE}{dy} = (y - d), \quad \frac{dy}{d\text{net}} = 1/\cosh^2(\text{net}), \quad \frac{\partial \text{net}}{\partial w_i} = x_i$$

$$\begin{aligned} \frac{\partial E}{\partial w_i} &= \frac{dE}{dy} \cdot \frac{dy}{d\text{net}} \cdot \frac{\partial \text{net}}{\partial w_i} \\ &= (y - d) / \cosh^2\left(\sum_i w_i x_i\right) \cdot x_i \end{aligned}$$

# Now We Can Use The Chain Rule



$$\frac{\partial E}{\partial y_k} = (y_k - d_k)$$

$$\delta_k = \frac{\partial E}{\partial net_k} = (y_k - d_k) \cdot g'(net_k)$$

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial net_k} \cdot \frac{\partial net_k}{\partial w_{jk}} = \frac{\partial E}{\partial net_k} \cdot y_j$$

$$\frac{\partial E}{\partial y_j} = \sum_k \left( \frac{\partial E}{\partial net_k} \cdot \frac{\partial net_k}{\partial y_j} \right)$$

$$\delta_j = \frac{\partial E}{\partial net_j} = \frac{\partial E}{\partial y_j} \cdot g'(net_j)$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial net_j} \cdot y_i$$

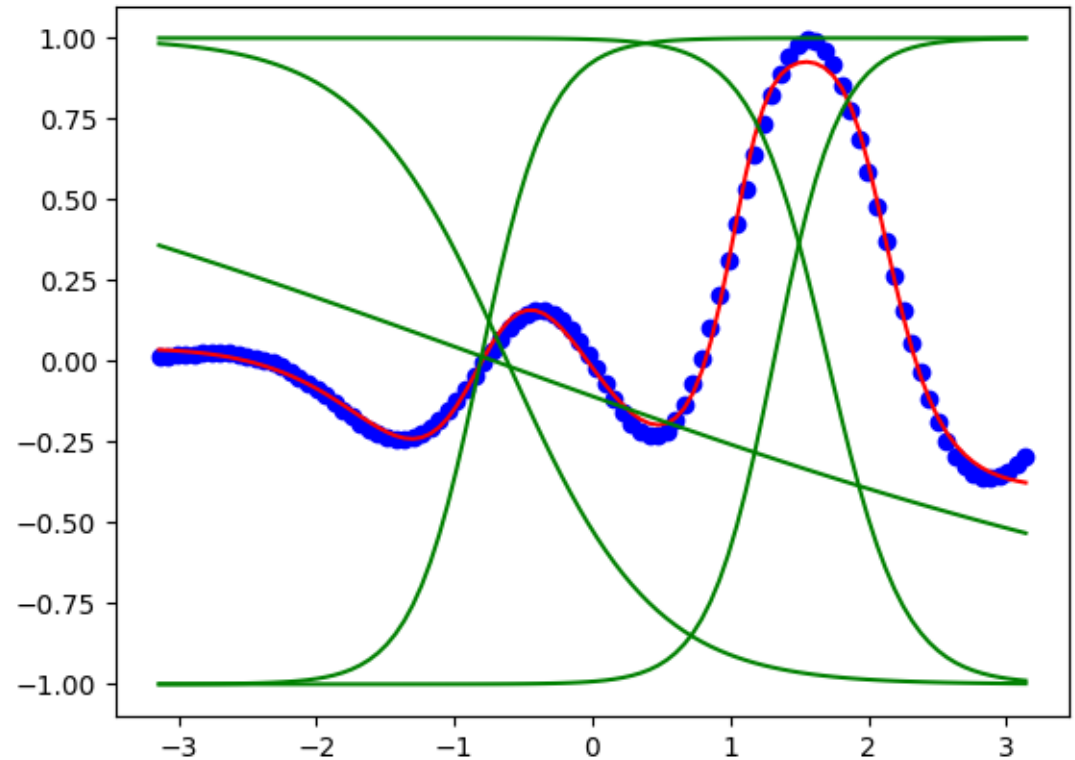
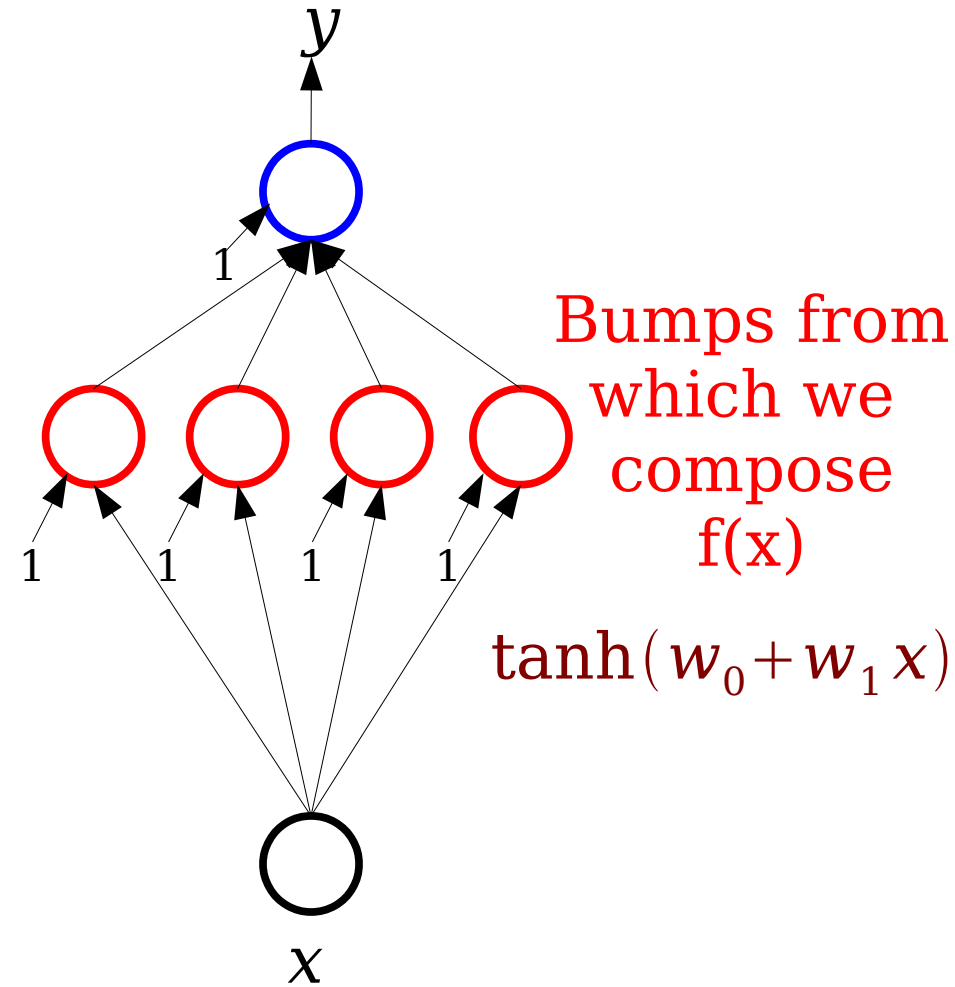
# Weight Updates

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial net_k} \cdot \frac{\partial net_k}{\partial w_{jk}} = \delta_k \cdot y_j$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ij}} = \delta_j \cdot y_i$$

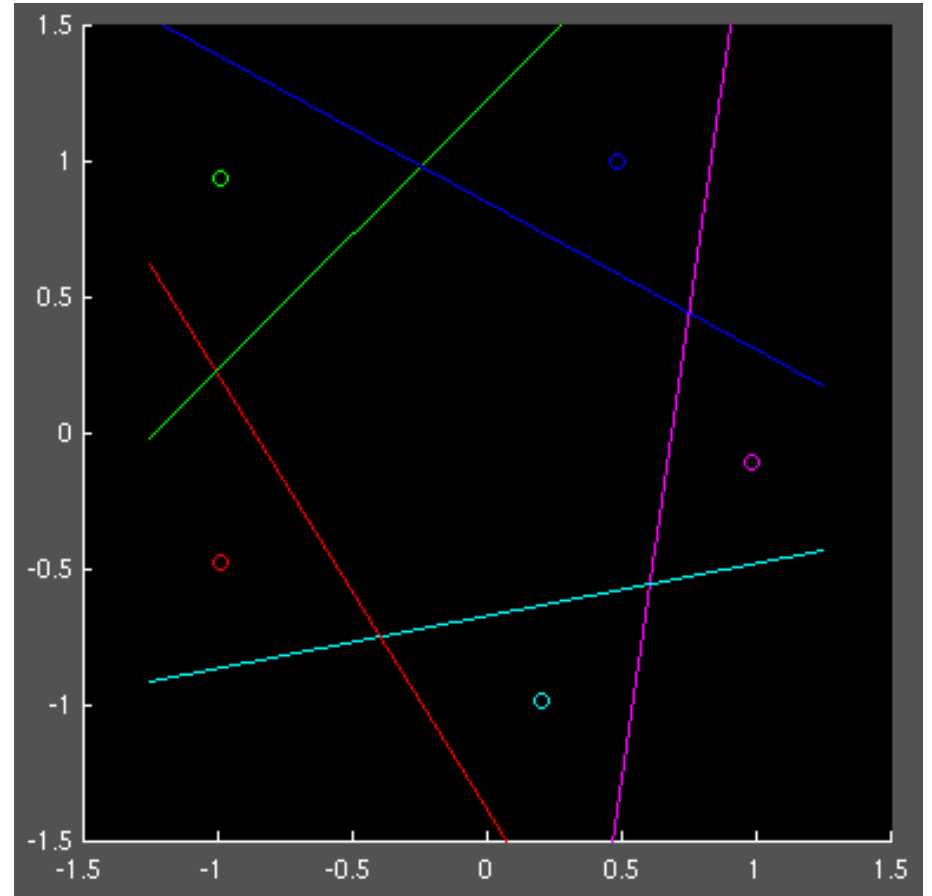
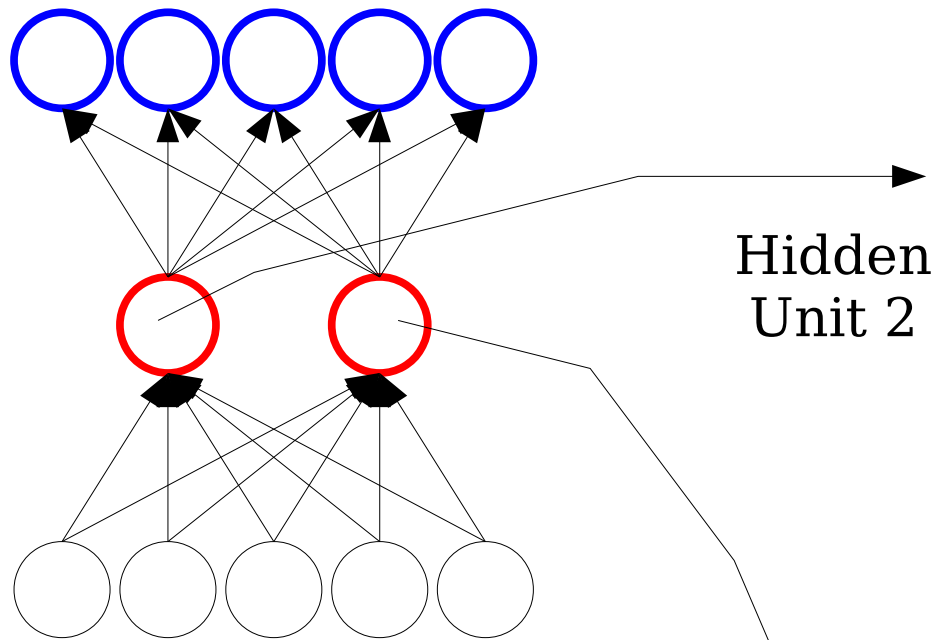
$$\Delta w_{jk} = -\eta \cdot \frac{\partial E}{\partial w_{jk}} \qquad \Delta w_{ij} = -\eta \cdot \frac{\partial E}{\partial w_{ij}}$$

# Function Approximation



$3n+1$  free parameters for  $n$  hidden units

# Encoder Problem



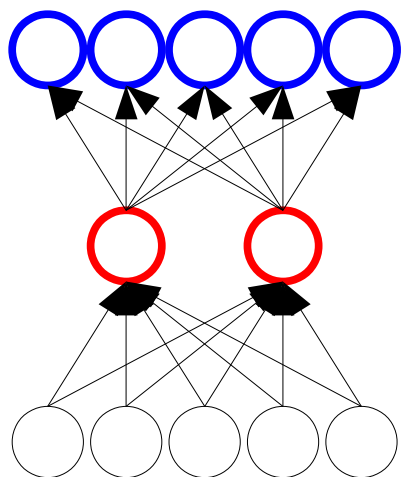
Input patterns: 1 bit on out of N.  
Output pattern: same as input.

Only 2 hidden units: bottleneck!

Hidden  
Unit 1



# 5-2-5 Encoder Problem

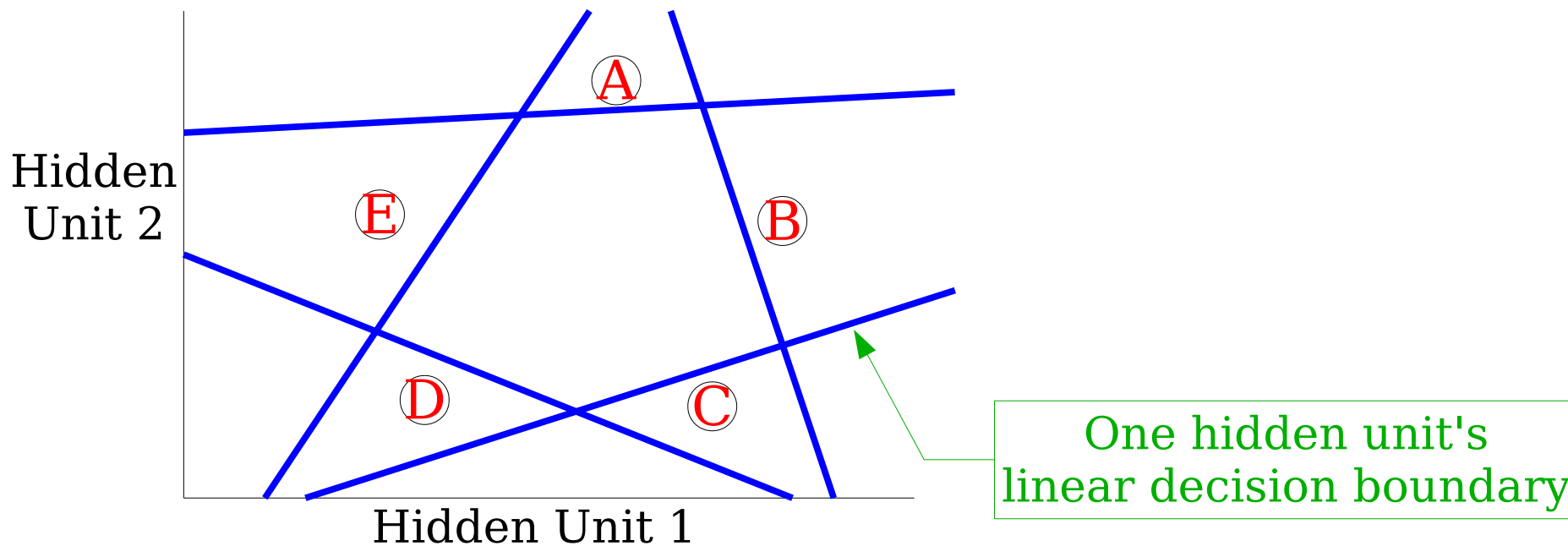


*Training patterns:*

<i>A</i> :	0	0	0	0	1
<i>B</i> :	0	0	0	1	0
<i>C</i> :	0	0	1	0	0
<i>D</i> :	0	1	0	0	0
<i>E</i> :	1	0	0	0	0

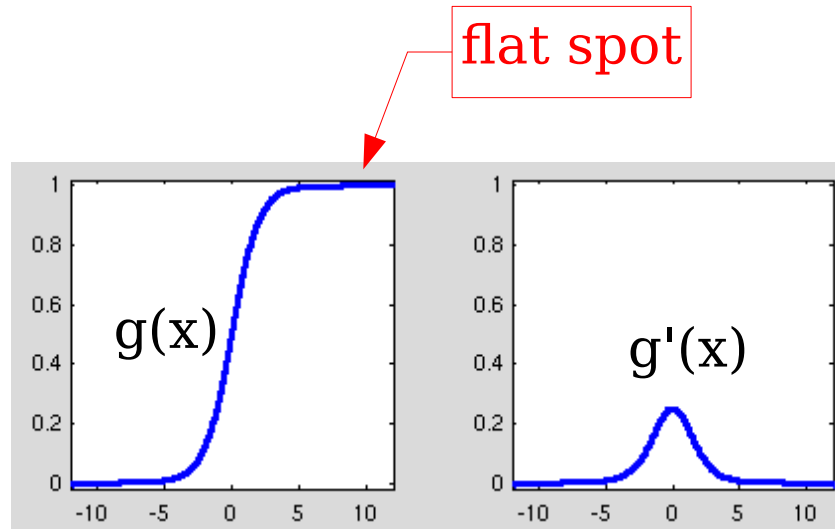
*Hidden code:*

<i>2,0</i>
<i>0,2</i>
<i>1,-1</i>
<i>-1,1</i>
<i>-1,0</i>



# Flat Spots

If weights become large,  $net_j$  becomes large, derivative of  $g()$  goes to zero.



Fahlman's trick: add a small constant to  $g'(x)$  to keep the derivative from going to zero. Typical value is 0.1.

# Momentum

Learning is slow if the learning rate is set too low.

Gradient may be steep in some directions but shallow in others.

Solution: add a momentum term  $\alpha$ .

$$\Delta w_{ij}(t) = -\eta \frac{\partial E}{\partial w_{ij}(t)} + \alpha \cdot \Delta w_{ij}(t-1)$$

Typical value for  $\alpha$  is 0.5.

If the direction of the gradient remains constant, the algorithm will take increasingly large steps.

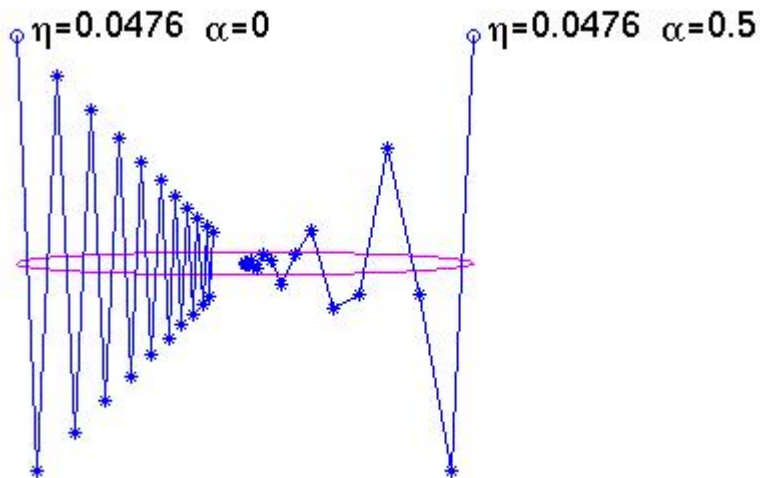
# Momentum Illustration

Hertz, Krogh & Palmer figs. 5.10 and 6.3: gradient descent on a quadratic error surface  $E$  (no neural net) involved:

$$E = x^2 + 20y^2$$

$$\frac{\partial E}{\partial x} = 2x, \quad \frac{\partial E}{\partial y} = 40y$$

Initial  $[x, y] = [-1, 1]$  or  $[1, 1]$

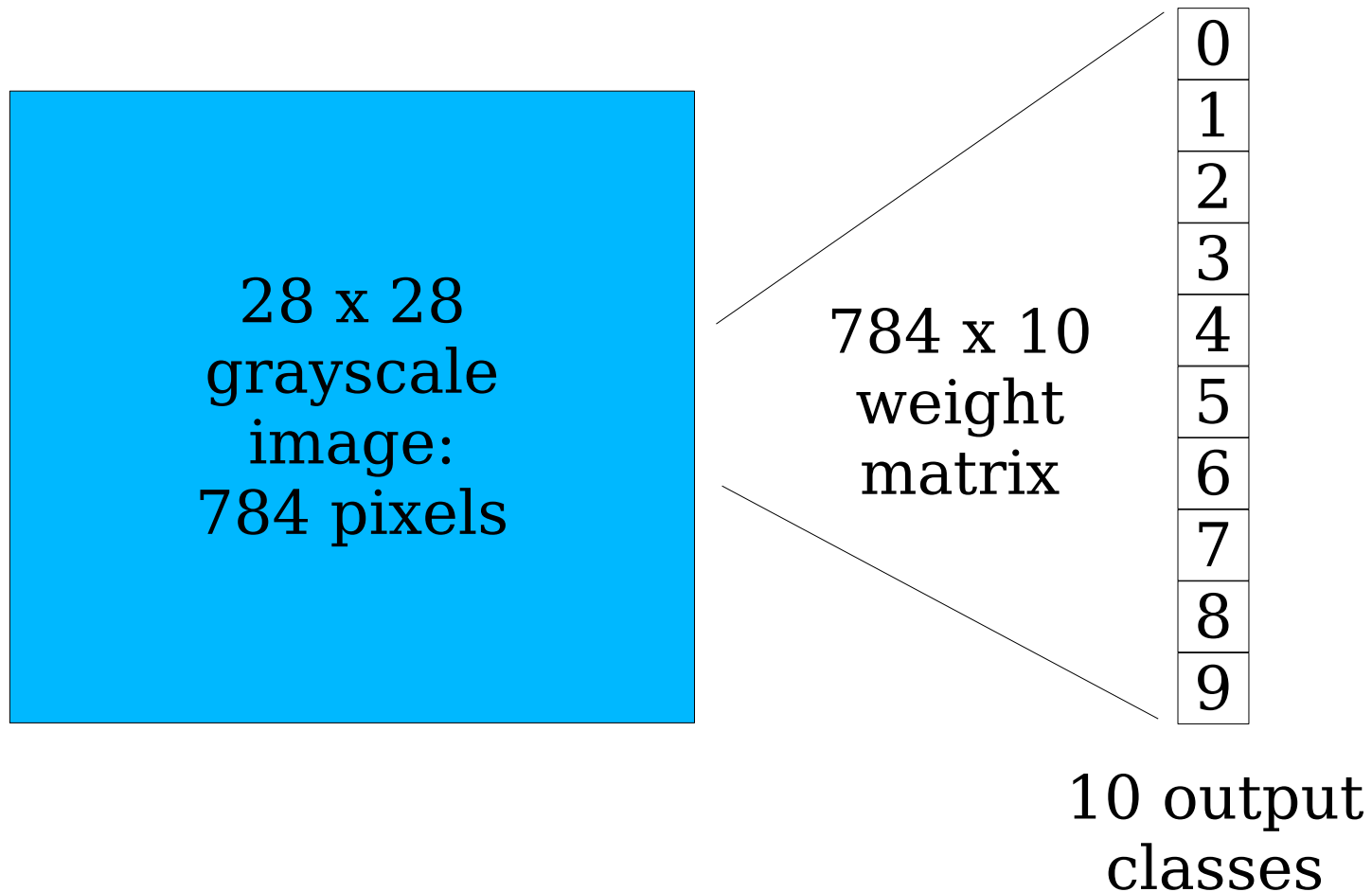


# MNIST Dataset

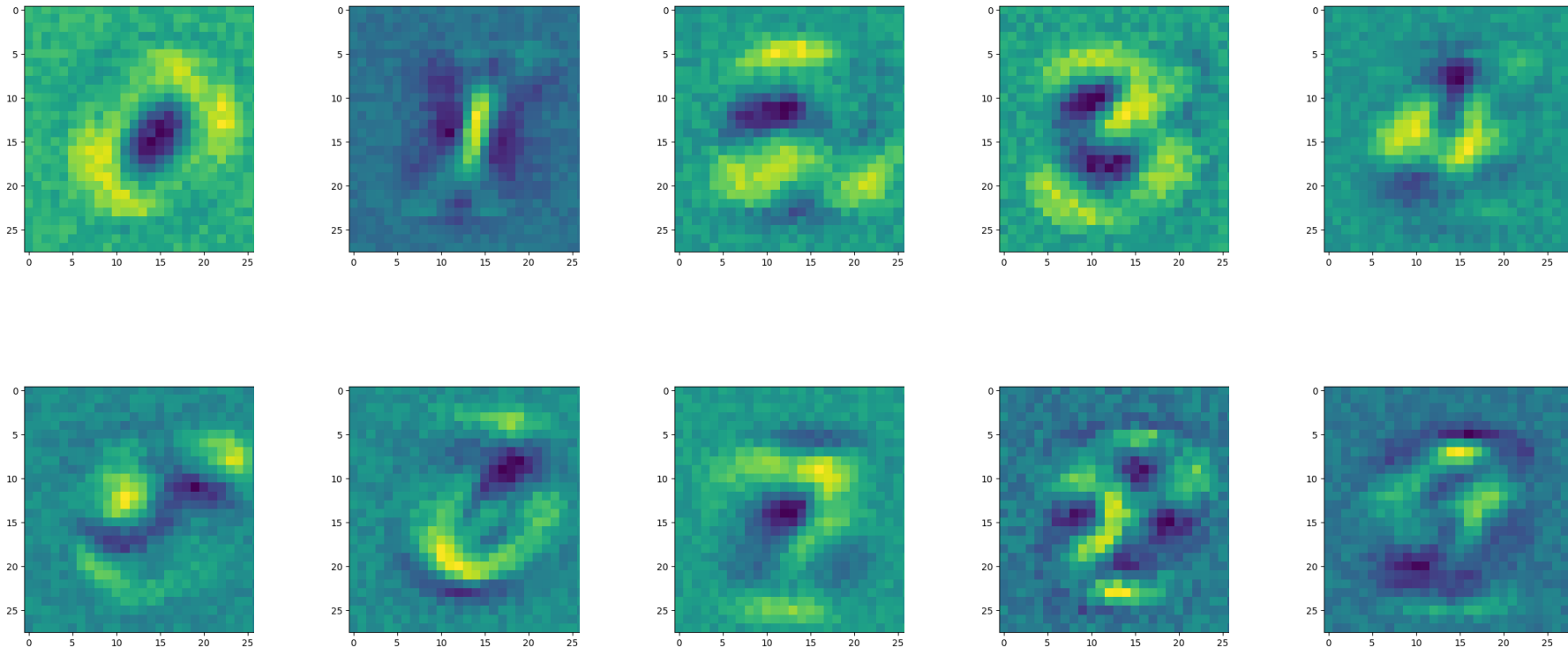
- 60,000 labeled handwritten digits
- 28 x 28 pixel grayscale images



# Recognition With a Linear Network



# Learned Weights to Output Units



Training set performance: 89% correct.

# TensorFlow Playground

Google's interactive backprop simulator.  
<https://playground.tensorflow.org>

