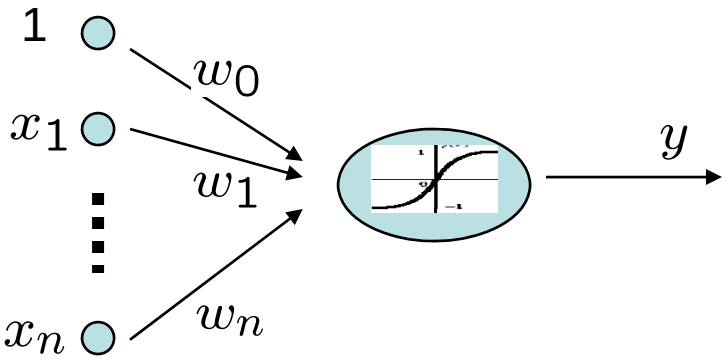
Artificial Neural Networks

Motivations

- Analogy to biological systems, which are the best examples of robust learning systems
- Consider human brain:
  - Neuron “switching time” ~ 10^{-3} S
  - Scene recognition can be done in 0.1 S
  - There is only time for about a hundred serial steps for performing such tasks
- Current AI system typically require a lot more serial steps than this!
- We need to exploit massive parallelism!

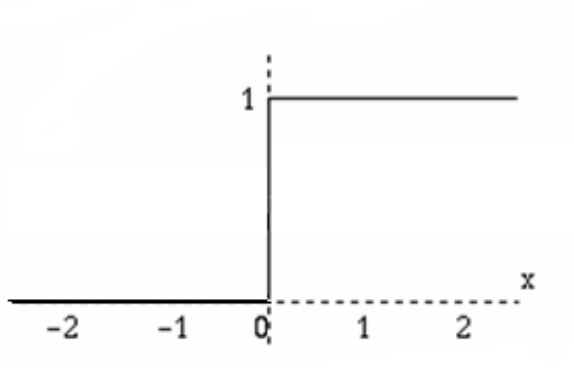
Neural Network Neurons



- Receives n inputs (plus a bias term)
- Multiplies each input by its weight
- Applies an activation function to the sum of results
- Outputs result

Activation Functions

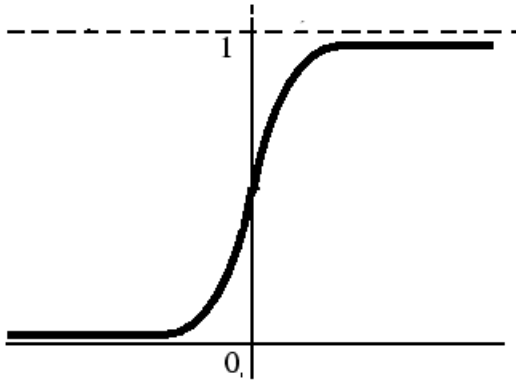
- Controls whether a neuron is “active” or “inactive”
- Threshold function: outputs 1 when input is positive and 0 otherwise – perceptron



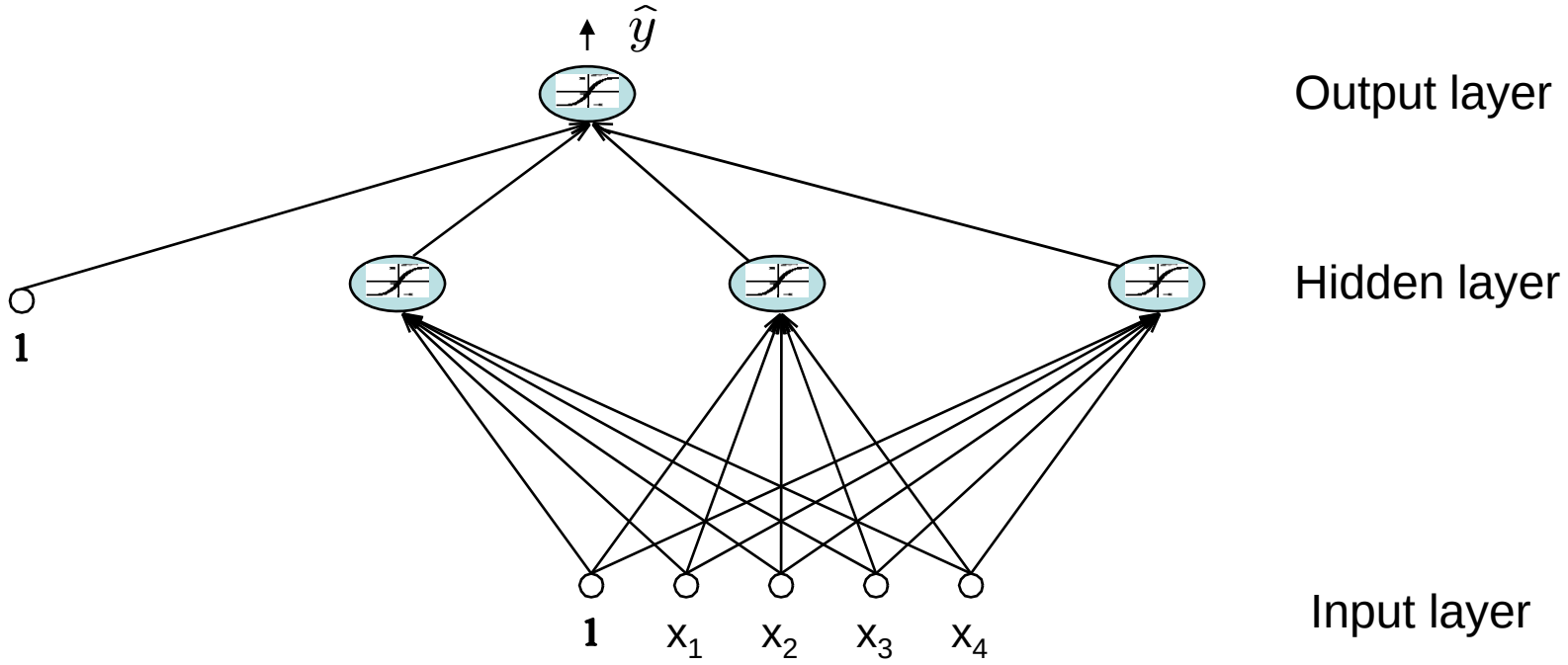
- Logistic sigmoid function

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

Differentiable – a good property for learning



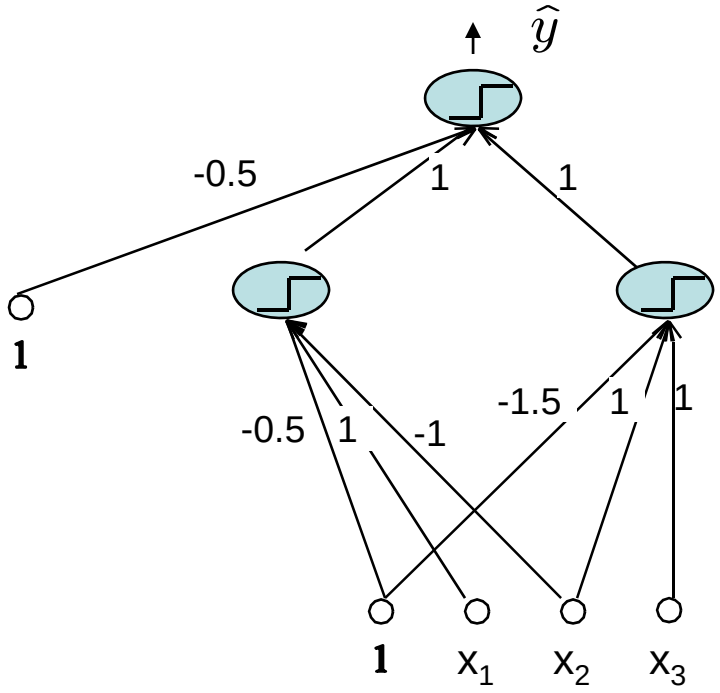
Multilayer Neural Network



- Each layer receives its inputs from the previous layer and forwards its outputs to the next – feed forward structure
- Output layer: sigmoid activation ftn for classification, linear activation ftn for regression problem
- Referred to as a two-layer network (two layer of weights)

Representational Power

- Any Boolean Formula
  - Consider the following boolean formula $(x_1 \wedge \neg x_2) \vee (x_2 \wedge x_3)$



OR units

AND units

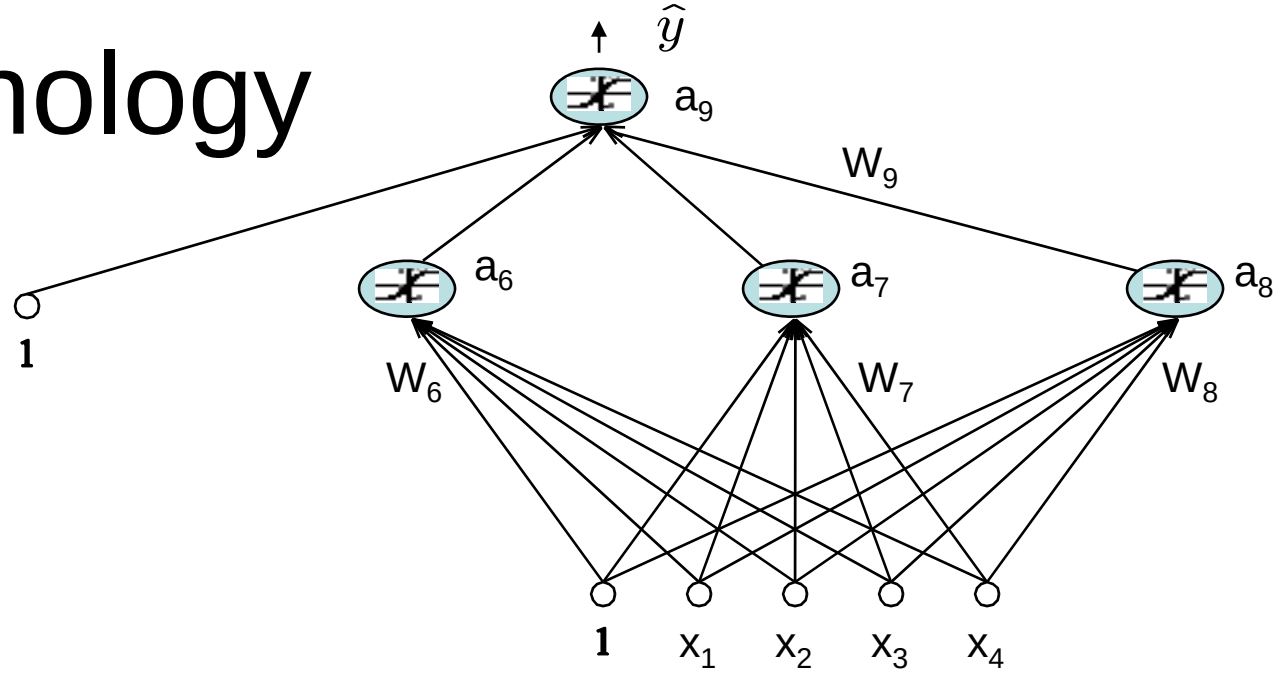
Representational Power (cont.)

- Universal approximator
  - A two layer (linear output) network can approximate any continuous function on a compact input domain to arbitrary accuracy given a sufficiently large number of hidden units

Neural Network Learning

- Start from a untrained network
- Present a training example X^i to the input layer, pass the signals through the network and determine the output at the output layer
- Compare output to the target values y^i, any difference corresponds to an error.
- We will adjust the weights to reduce this error on the training data

Terminology



- $X=[1, x_1, x_2, x_3, x_4]^T$ – the input vector with the bias term
- $A=[1, a_6, a_7, a_8]^T$ – the output of the hidden layer with the bias term
- W_i represents the weight vector leading to node *i*
- $w_{i,j}$ represents the weight connecting from the *j*th node to the *i*th node
  - $w_{9,6}$ is the weight connecting from a_6 to a_9
- We will use σ to represent the activation function (generally a sigmoid function), so

$$\hat{y} = \sigma(W_9 \cdot [1, a_6, a_7, a_8]^T) = \sigma(W_9 \cdot [1, \sigma(W_6 \cdot X), \sigma(W_7 \cdot X), \sigma(W_8 \cdot X)]^T)$$

Mean Squared Error

- We adjust the weights of the neural network to minimize the mean squared error (MSE) on training set.

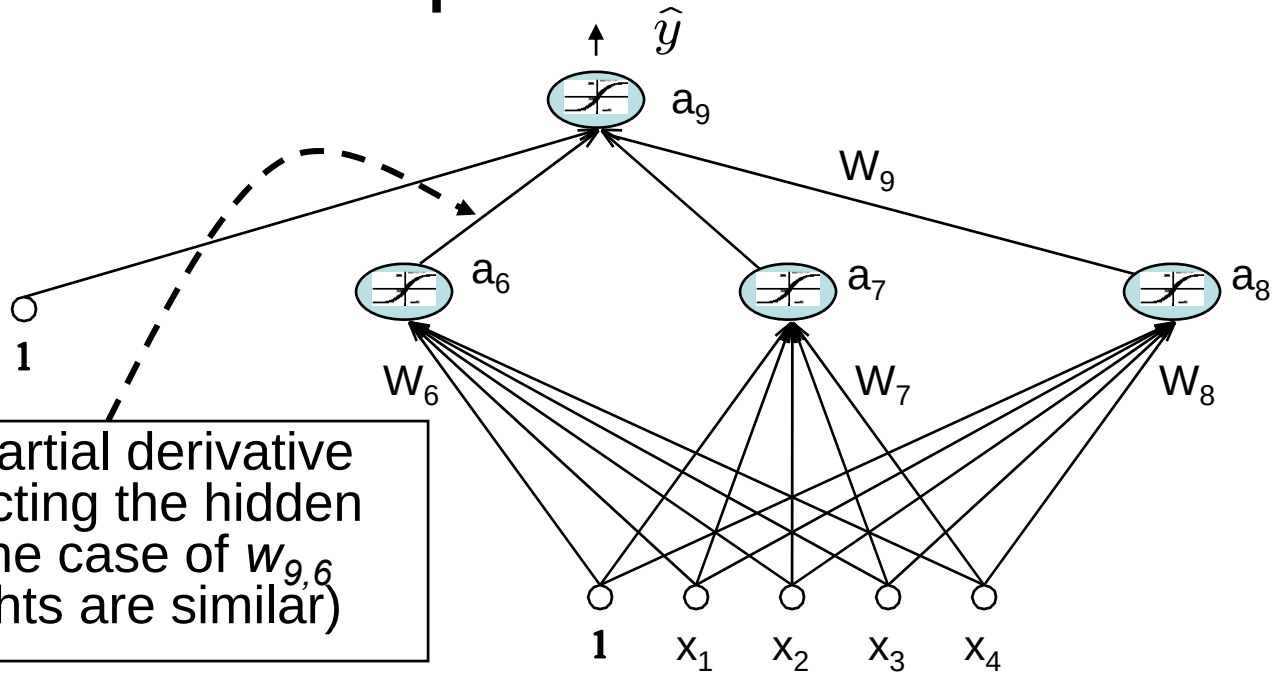
$$J(W) = \frac{1}{2}\sum_{i=1}^{N}(\widehat{y}^i - y^i)^2$$

$$J_i(W) = \frac{1}{2}(\widehat{y}^i - y^i)^2$$

- **Useful Fact**: the derivative of the sigmoid activation function is

$$\frac{\partial}{\partial x}\sigma(x) = \sigma(x)(1 - \sigma(x))$$

Gradient Descent: Output Unit



Lets compute the partial derivative wrt a weight connecting the hidden to output layer. In the case of $w_{9,6}$ we get: (other weights are similar)

$$\frac{\partial J_i(W)}{\partial w_{9,6}} = \frac{\partial}{\partial w_{9,6}} \frac{1}{2}(\widehat{y}^i - y^i)^2$$

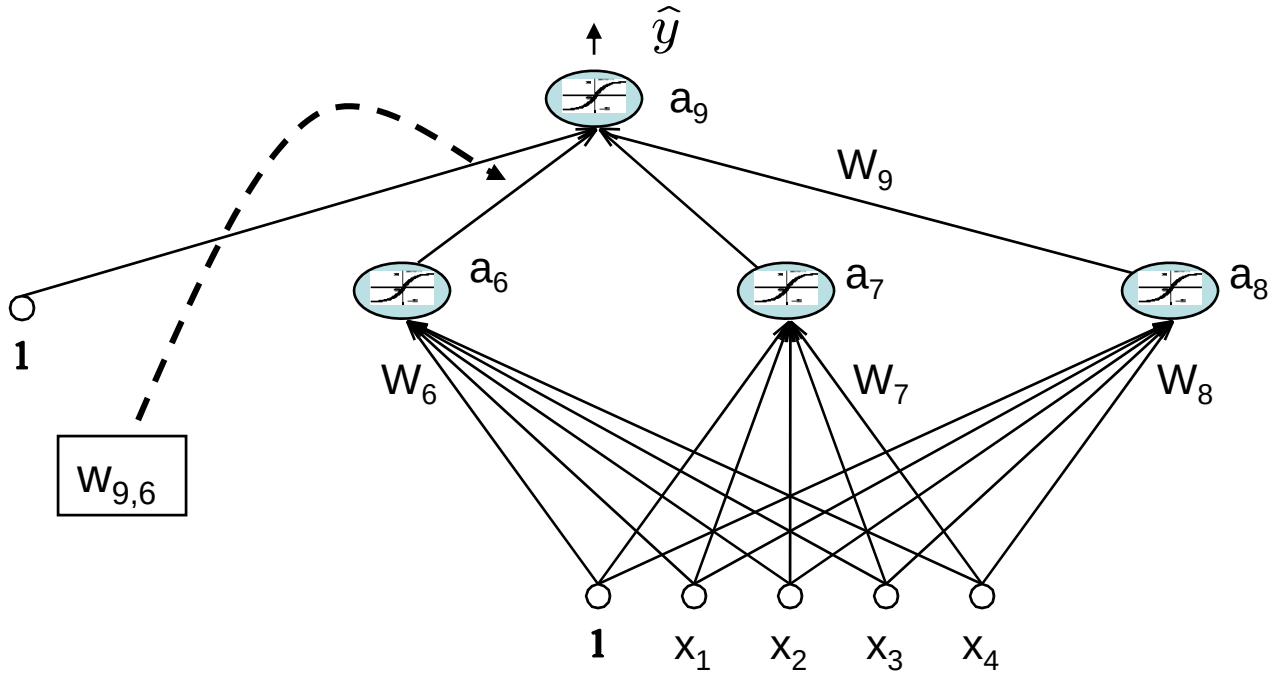
$$= \frac{1}{2} \cdot 2 \cdot (\widehat{y}^i - y^i) \cdot \frac{\partial}{\partial w_{9,6}} (\sigma(W_9 \cdot A^i) - y^i)$$

$$= (\widehat{y}^i - y^i) \cdot \sigma(W_9 \cdot A^i)(1 - \sigma(W_9 \cdot A^i)) \cdot \frac{\partial}{\partial w_{9,6}} W_9 \cdot A^i$$

$$= (\widehat{y}^i - y^i)\widehat{y}^i(1 - \widehat{y}^i) \cdot a_6^i$$

hidden layer activation for i'th example

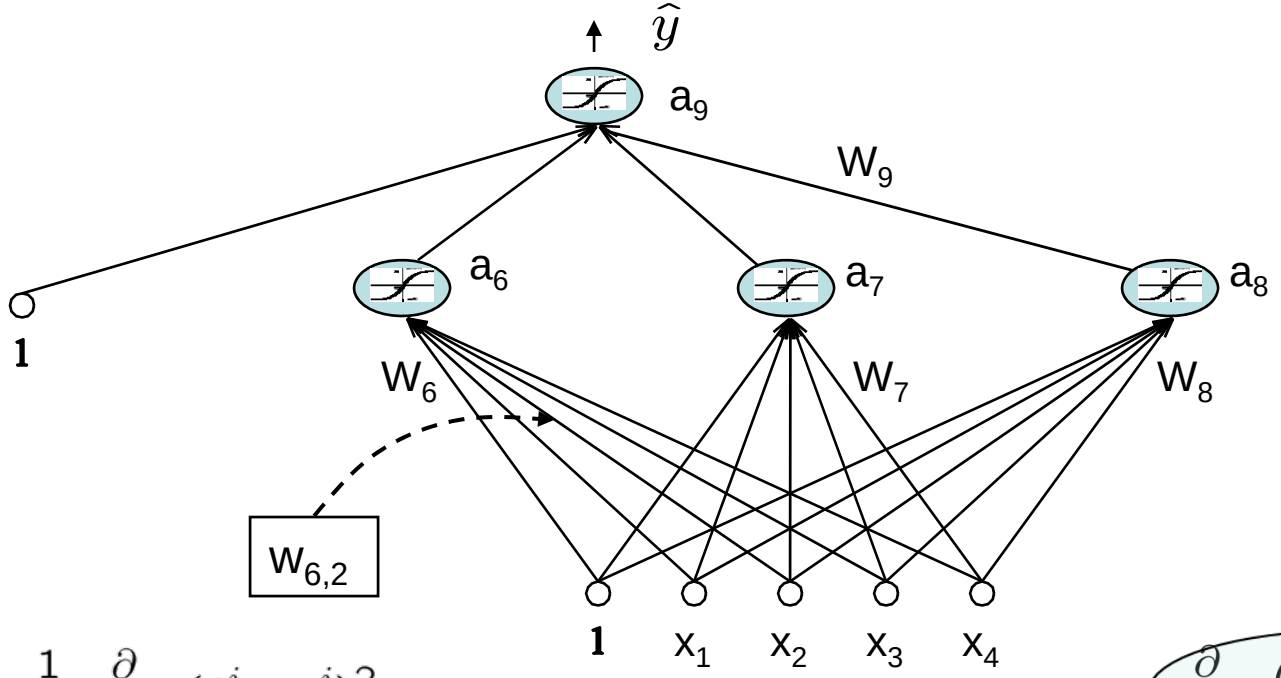
The Delta Rule



- Define $\delta_9^i = (\widehat{y}^i - y^i)\widehat{y}^i(1 - \widehat{y}^i)$

  then $$\frac{\partial J_i(W)}{\partial w_{9,6}} = (\widehat{y}^i - y^i)\widehat{y}^i(1 - \widehat{y}^i) \cdot a_6^i$$
  $$= \delta_9^i \cdot a_6^i$$

Derivation: Hidden Units



$$\frac{\partial J_i(W)}{\partial w_{6,2}} = \frac{1}{2}\frac{\partial}{\partial w_{6,2}}(\hat{y}^i - y^i)^2$$

$$= (\hat{y}^i - y^i) \cdot \sigma(W_9 \cdot A^i)(1 - \sigma(W_9 \cdot A^i)) \cdot \frac{\partial}{\partial w_{6,2}}(W_9 \cdot A^i)$$

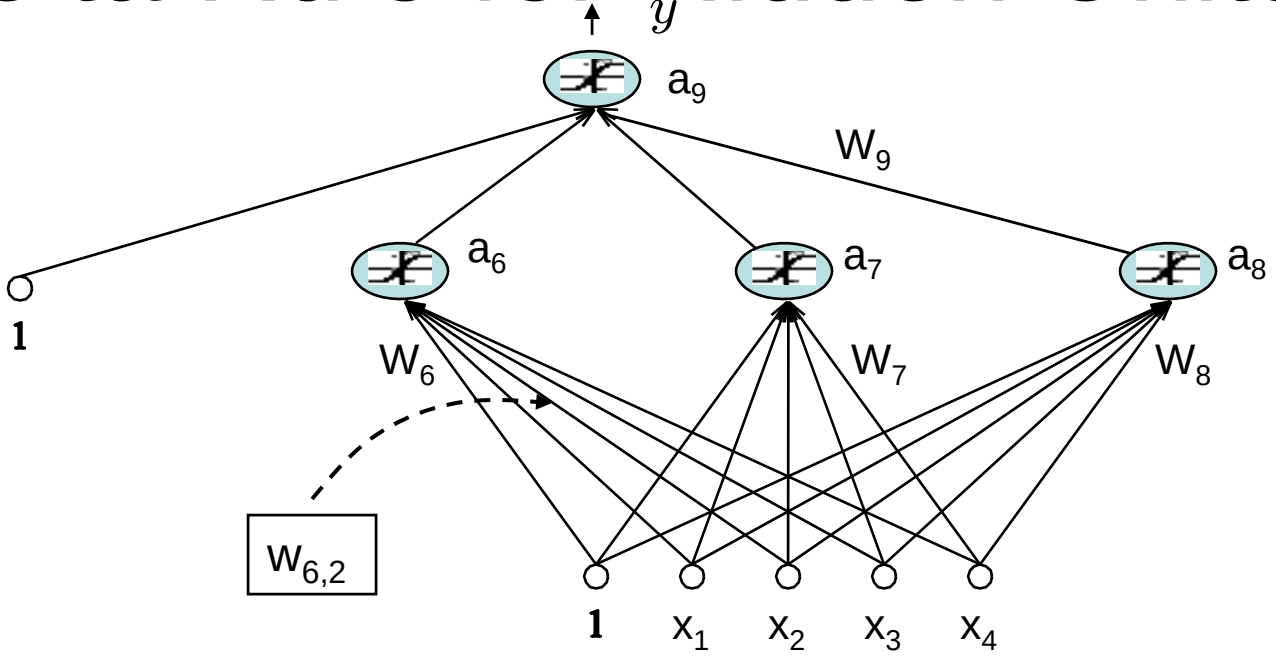
$$= \delta_9^i \cdot w_{9,6} \cdot \frac{\partial}{\partial w_{6,2}}\sigma(W_6 \cdot X^i)$$

$$= \delta_9^i \cdot w_{9,6} \cdot \sigma(W_6 \cdot X^i)(1 - \sigma(W_6 \cdot X^i)) \cdot \frac{\partial}{\partial w_{6,2}}(W_6 \cdot X^i)$$

$$= \delta_9^i \cdot w_{9,6} \cdot a_6(1 - a_6) \cdot x_2^i$$

$$= \frac{\partial}{\partial w_{6,2}}(w_{9,6} \cdot a_6^i)$$

Delta Rule for Hidden Units



Define $\delta_6^i = \delta_9^i \cdot w_{9,6} \cdot a_6^i(1 - a_6^i)$

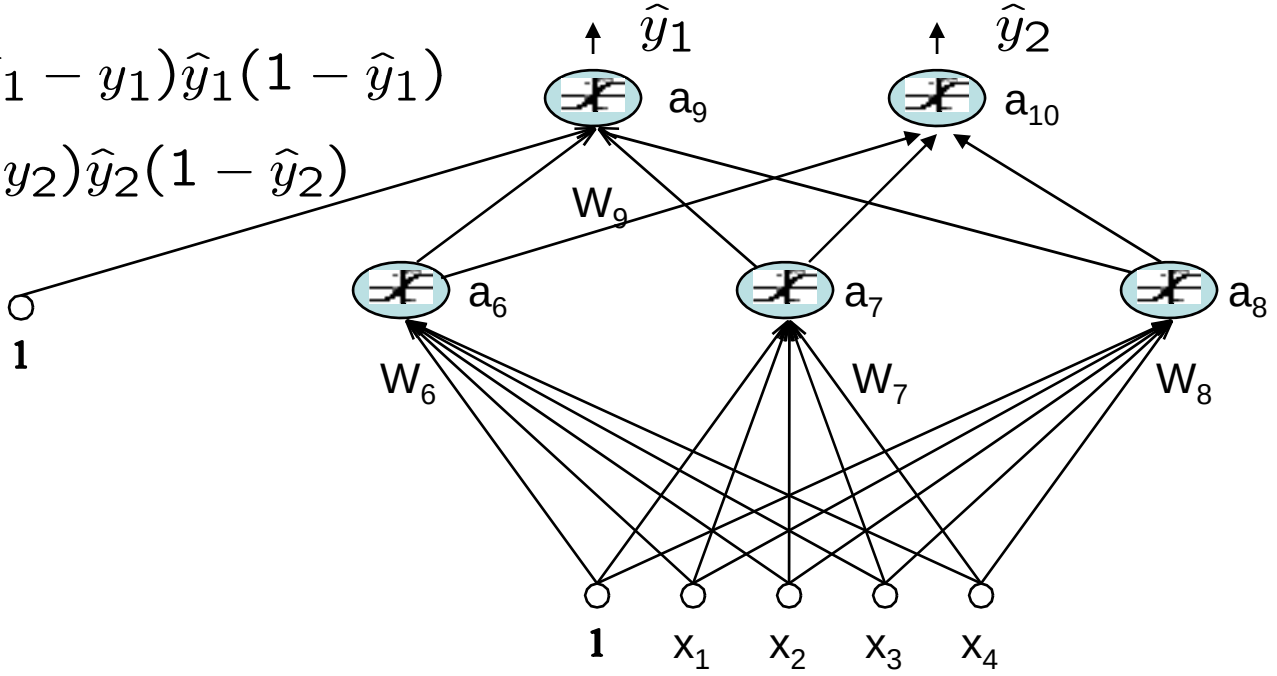
and rewrite as

$$\frac{\partial J_i(W)}{\partial w_{6,2}} = \delta_6^i \cdot x_2^i.$$

Networks with Multiple Output Units

$$\delta_9 = (\hat{y}_1 - y_1)\hat{y}_1(1 - \hat{y}_1)$$

$$\delta_{10} = (\hat{y}_2 - y_2)\hat{y}_2(1 - \hat{y}_2)$$



- We get a separate contribution to the gradient from each output unit.
- Hence, for input-to-hidden weights, we must sum up the contributions:

$$\delta_6 = a_6(1 - a_6)(w_{9,6}\delta_9 + w_{10,6}\delta_{10})$$

The Backpropagation Algorithm

- **Forward Pass** Given X, compute a_u and $\hat{y}_v$ for hidden units u and output units v.
- **Compute Errors** Compute $\varepsilon_v = (\hat{y}_v - y_v)$ for each output unit v
- **Compute Output Deltas** $\delta_v = (\hat{y}_v - y_v)\, \hat{y}_v(1 - \hat{y}_v)$
- **Compute Hidden Deltas** $\delta_u = a_u(1 - a_u) \sum_v w_{v,u}\, \delta_v$
- **Compute Gradient**
  - Compute $\frac{\partial J_i}{\partial w_{v,u}} = \delta_v a_u^i$ for hidden-to-output weights.
  - Compute $\frac{\partial J_i}{\partial w_{u,j}} = \delta_u x_j^i$ for input-to-hidden weights.

Backpropagate error from layer to layer

- **For each weight take a gradient step**

$$w_{u,v} := w_{u,v} - \eta \frac{\partial J_i}{\partial w_{u,v}}$$

Backpropagation Training

- Create a three layer network with N hidden units, fully connect the network and assign small random weights
- Until all training examples produce correct output or MSE ceases to decrease
  - For all training examples, do

    Begin Epoch

      For each training example do
        - Compute the network output
        - Compute the error
        - Backpropagate this error from layer to layer and adjust weights to decrease this error

    End Epoch

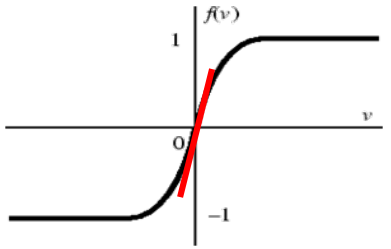
Remarks on Training

- No guarantee of convergence, may oscillate or reach a local minima.
- However, in practice many large networks can be adequately trained on large amounts of data for realistic problems, e.g.
  - Driving a car
  - Recognizing handwritten zip codes
  - Play world championship level Backgammon
- Many epochs (thousands) may be needed for adequate training, large data sets may require hours or days of CPU time.
- Termination criteria can be:
  - Fixed number of epochs
  - Threshold on training set error
  - Increased error on a validation set
- To avoid local minima problems, can run several trials starting from different initial random weights and:
  - Take the result with the best training or validation performance.
  - Build a committee of networks that vote during testing, possibly weighting vote by training or validation accuracy

Notes on Proper Initialization

- Start in the “linear” regions

  keep all weights near zero, so that all sigmoid units are in their linear regions. Otherwise nodes can be initialized into flat regions of the sigmoid causing for very small gradients



- Break symmetry
  - If we start with the weights all equal, what will happen?
  - Ensure that each hidden unit has different input weights so that the hidden units move in different directions.
- Set each weight to a random number in the range

$$[-1, +1] \times \frac{1}{\sqrt{\text{fan-in}}}.$$

  Where “fan-in” of weight $w_{v,u}$ is the number of inputs to unit v.

Batch, Online, and Online with Momentum

- Batch. Sum the $\nabla_W J_i(W)$ for each example *i*. Then take a gradient descent step.
- Online. Take a gradient descent step with each $\nabla_W J_i(W)$ as it is computed (this is the algorithm we described)
- Momentum factor. Make the *t+1*-th update dependent on the *t*-th update

$$\Delta W^{(t+1)} = \nabla_W J(W^t)$$

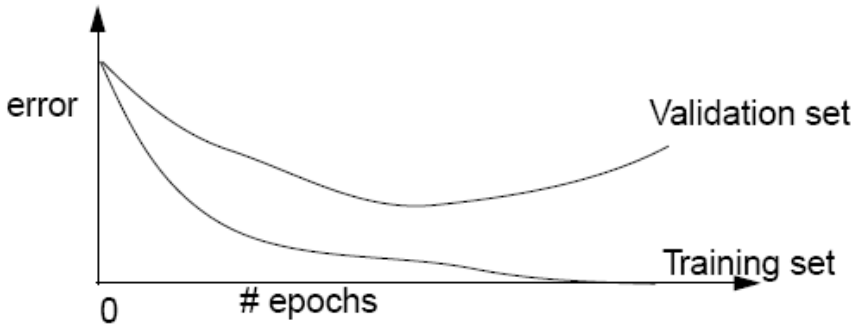
$$\Delta W^{(t+1)} = \alpha \Delta W^{(t)} + \nabla_W J(W^t)$$

α is called the momentum factor, and typically take values in the range [0.7, 0.95]

This tends to keep weight moving in the same direction and improves convergence.

Overtraining Prevention

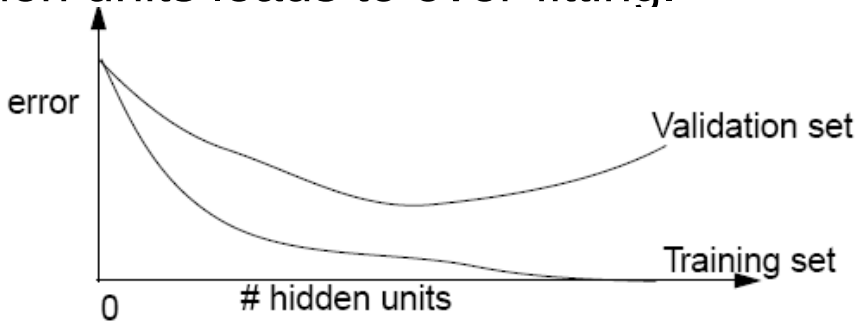
- Running too many epochs may overtrain the network and result in overfitting. Tries too hard to exactly match the training data.



- Keep a validation set and test accuracy after every epoch. Maintain weights for best performing network on the validation set and return it when performance decreases significantly beyond this.
- To avoid losing training data to validation:
  - Use 10-fold cross-validation to determine the average number of epochs that optimizes validation performance.
    - We will discuss cross-validation later in the course
  - Train on the full data set using this many epochs to produce the final result.

Over-fitting Prevention

- Too few hidden units prevent the system from adequately fitting the data and learning the concept.
- Too many hidden units leads to over-fitting.



- Can also use a validation set or cross-validation to decide an appropriate number of hidden units.
- Another approach to preventing over-fitting is ***weight decay***, in which we multiply all weights by some fraction between 0 and 1 after each epoch.
  - Encourages smaller weights and less complex hypotheses.
  - Equivalent to including an additive penalty in the error function proportional to the sum of the squares of the weights of the network.

Input/Output Coding

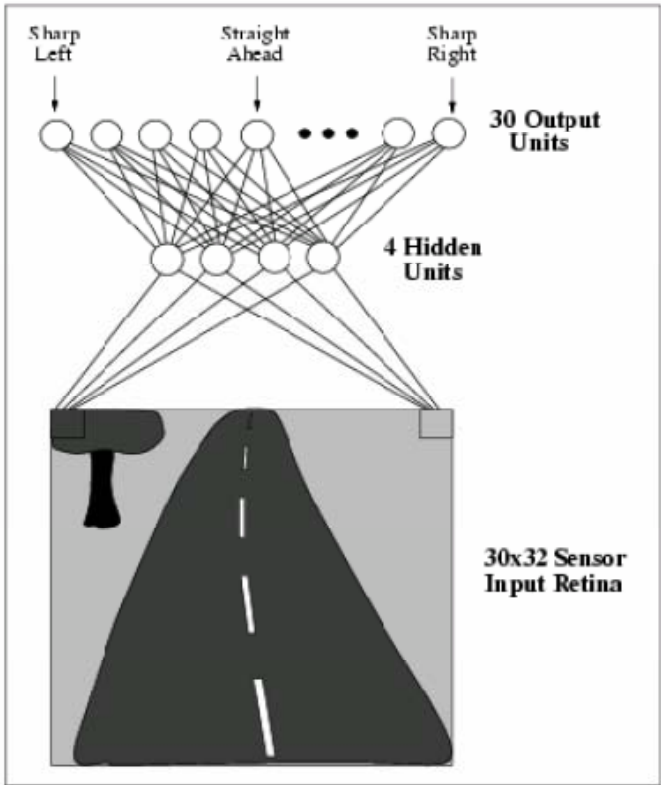
- Appropriate coding of inputs/outputs can make learning easier and improve generalization.
- Best to encode multinomial features using multiple input units and include one binary unit per value
  - A multinomial feature takes values from a finite unordered domain.
- Continuous inputs can be handled by a single input unit by scaling them between 0 and 1.
- For classification problems, best to have one output unit per class.
  - Continuous output values then represent certainty in various classes.
  - Assign test instances to the class with the highest output.
- Use target values of 0.9 and 0.1 for binary problems rather than forcing weights to grow large enough to closely approximate 0/1 outputs.
- Continuous outputs (regression) can also be handled by using identity activation function at the output layer

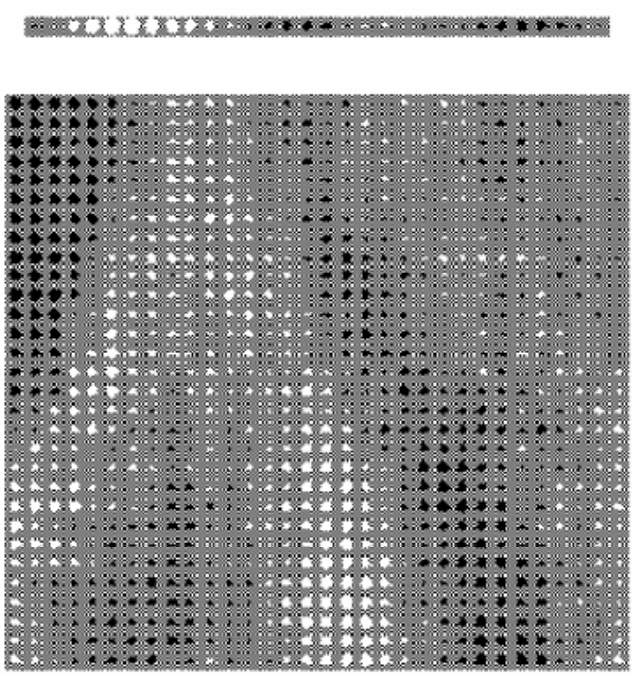
Neural Networks: Summary

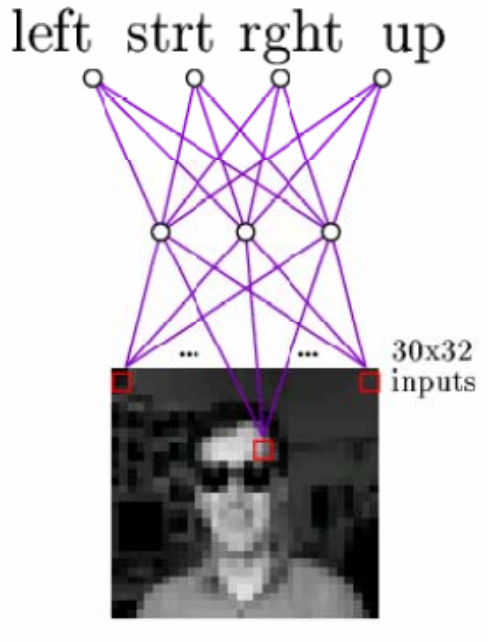
- Neural Networks can represent complex decision boundaries
  - Variable size. Any boolean function can be represented. Hidden units can be interpreted as new features
- Learning Algorithms for neural networks
  - Local Search.
  - Batch or Online
- Because of the large number of parameter
  - Learning can be slow
  - Prone to overfitting
  - Training is somewhat of an art

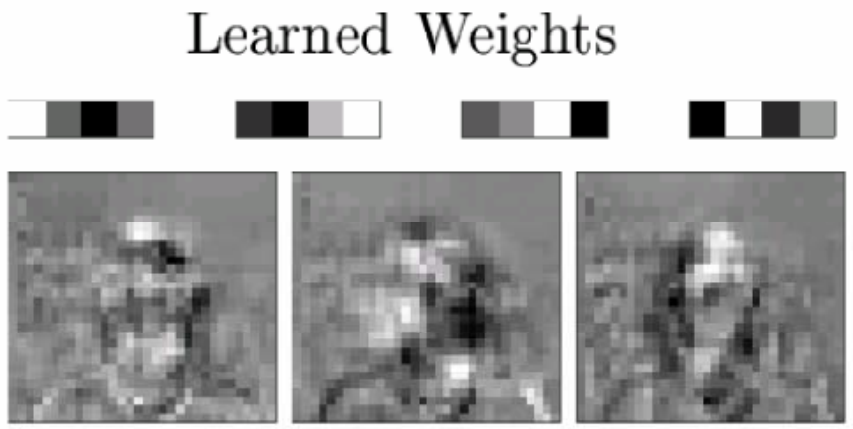
Example

Neural net is one of the most effective methods when the data include complex sensory inputs such as images.









Typical input images