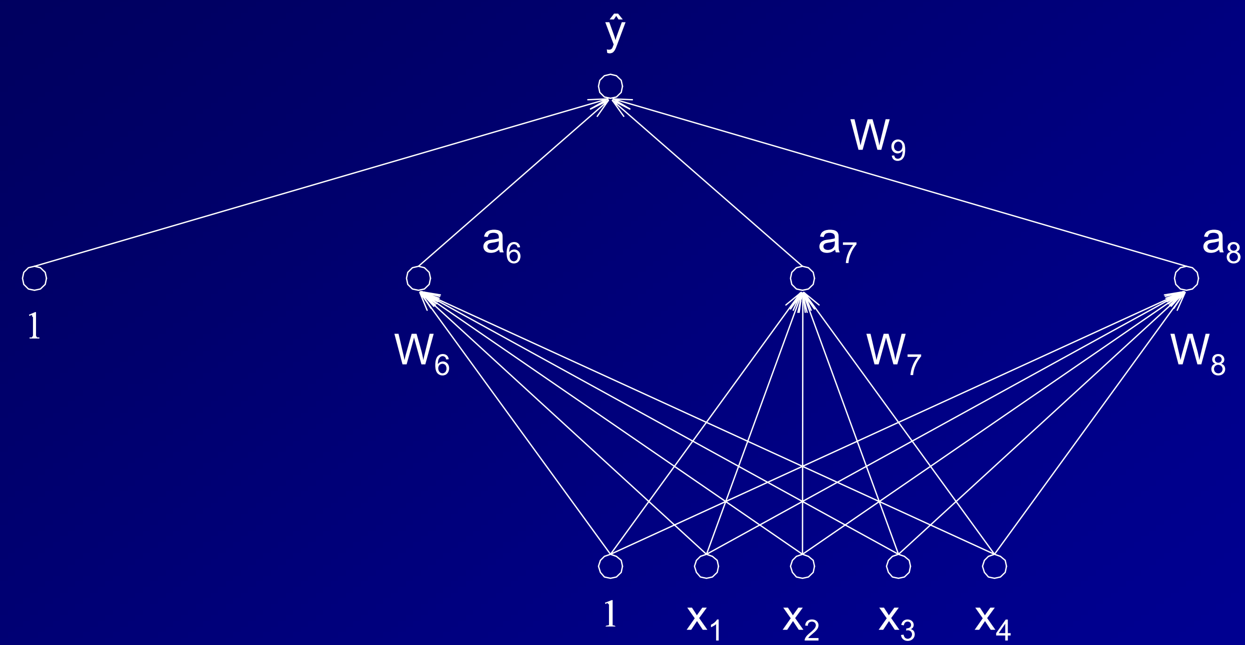
Learning Neural Networks

- Neural Networks can represent complex decision boundaries
  - Variable size. Any boolean function can be represented. Hidden units can be interpreted as new features
  - Deterministic
  - Continuous Parameters
- Learning Algorithms for neural networks
  - Local Search. The same algorithm as for sigmoid threshold units
  - Eager
  - Batch or Online

Neural Network Hypothesis Space



- Each unit a_6, a_7, a_8, and $\hat{y}$ computes a sigmoid function of its inputs:
  $a_6 = \sigma(W_6 \cdot X)$ $a_7 = \sigma(W_7 \cdot X)$ $a_8 = \sigma(W_8 \cdot X)$ $\hat{y} = \sigma(W_9 \cdot A)$
  where $A = [1, a_6, a_7, a_8]$ is called the vector of *hidden unit activitations*
- Original motivation: Differentiable approximation to multi-layer LTUs

Representational Power

- Any Boolean Formula
  - Consider a formula in disjunctive normal form:

    $(x_1 \wedge \neg x_2) \vee (x_2 \wedge x_4) \vee (\neg x_3 \wedge x_5)$

    Each AND can be represented by a hidden unit and the OR can be represented by the output unit. Arbitrary boolean functions require exponentially-many hidden units, however.
- Bounded functions
  - Suppose we make the output linear: $\hat{y} = W_9 \cdot A$ of hidden units. It can be proved that any bounded continuous function can be approximated to arbitrary accuracy with enough hidden units.
- Arbitrary Functions
  - Any function can be approximated to arbitrary accuracy with two hidden layers of sigmoid units and a linear output unit.

Fixed versus Variable Size

- In principle, a network has a fixed number of parameters and therefore can only represent a fixed hypothesis space (if the number of hidden units is fixed).
- However, we will initialize the weights to values near zero and use gradient descent. The more steps of gradient descent we take, the more functions can be "reached" from the starting weights.
- So it turns out to be more accurate to treat networks as having a variable hypothesis space that depends on the number of steps of gradient descent

Backpropagation: Gradient Descent for Multi-Layer Networks

- It is traditional to train neural networks to minimize the squared error. This is really a mistake—they should be trained to maximize the log likelihood instead. But we will study the MSE first.

$$\hat{y} = \sigma(W_9 \cdot [1, \sigma(W_6 \cdot X), \sigma(W_7 \cdot X), \sigma(W_9 \cdot X)])$$

$$J_i(W) = \frac{1}{2}(\hat{y}_i - y_i)^2$$

- We must apply the chain rule many times to compute the gradient
- We will number the units from 0 to U and index them by *u* and *v*.
- $w_{v,u}$ will be the weight connecting unit *u* to unit *v*. (Note: This seems backwards. It is the *u*th input to node *v*.)

Derivation: Output Unit

- Suppose $w_{9,6}$ is a component of W_9, the output weight vector, connecting it from a_6.

$$
\begin{aligned}
\frac{\partial J_i(W)}{\partial w_{9,6}} &= \frac{\partial}{\partial w_{9,6}} \frac{1}{2}(\widehat{y}_i - y_i)^2 \\
&= \frac{1}{2} \cdot 2 \cdot (\widehat{y}_i - y_i) \cdot \frac{\partial}{\partial w_{9,6}} (\sigma(W_9 \cdot A_i) - y_i) \\
&= (\widehat{y}_i - y_i) \cdot \sigma(W_9 \cdot A_i)(1 - \sigma(W_9 \cdot A_i)) \cdot \frac{\partial}{\partial w_{9,6}} W_9 \cdot A_i \\
&= (\widehat{y}_i - y_i)\widehat{y}_i(1 - \widehat{y}_i) \cdot a_6
\end{aligned}
$$

The Delta Rule

- Define $\delta_9 = (\widehat{y}_i - y_i)\widehat{y}_i(1 - \widehat{y}_i)$

  then

$$\begin{aligned} \frac{\partial J_i(W)}{\partial w_{9,6}} &= (\widehat{y}_i - y_i)\widehat{y}_i(1 - \widehat{y}_i) \cdot a_6 \\ &= \delta_9 \cdot a_6 \end{aligned}$$

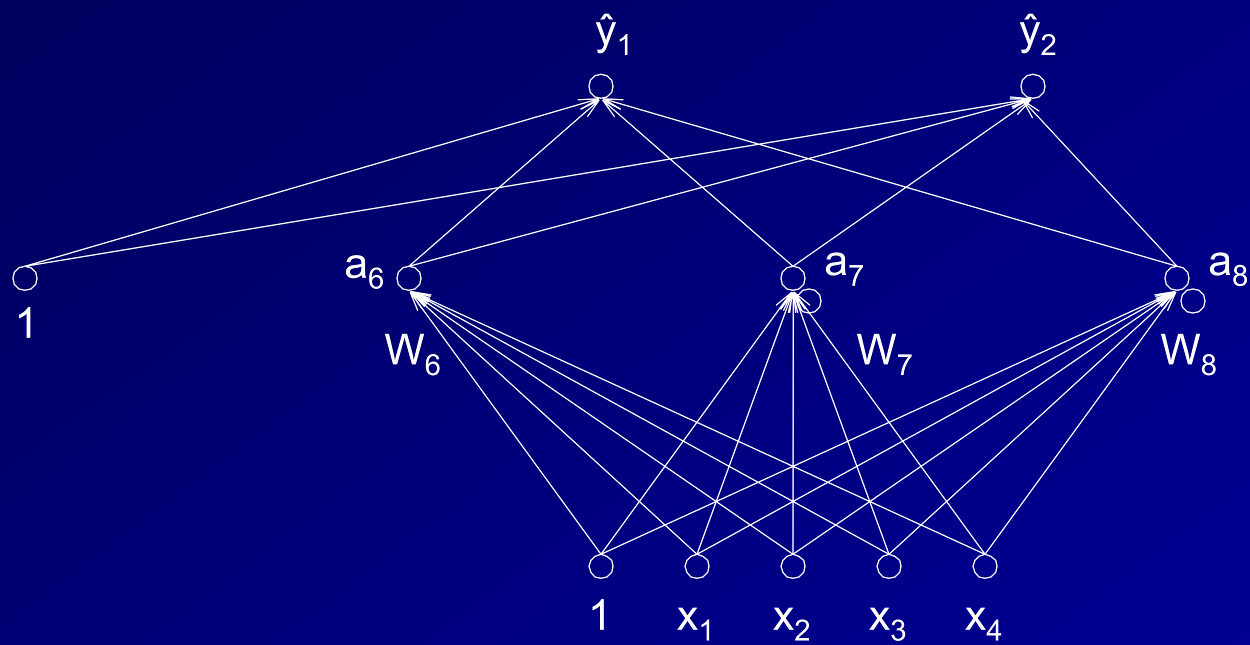
Derivation: Hidden Units

$$
\begin{aligned}
\frac{\partial J_i(W)}{\partial w_{6,2}} &= (\hat{y}_i - y_i) \cdot \sigma(W_9 \cdot A_i)(1 - \sigma(W_9 \cdot A_i)) \cdot \frac{\partial}{\partial w_{6,2}} W_9 \cdot A_i \\
&= \delta_9 \cdot w_{9,6} \cdot \frac{\partial}{\partial w_{6,2}} \sigma(W_6 \cdot X) \\
&= \delta_9 \cdot w_{9,6} \cdot \sigma(W_6 \cdot X)(1 - \sigma(W_6 \cdot X)) \cdot \frac{\partial}{\partial w_{6,2}} (W_6 \cdot X) \\
&= \delta_9 \cdot w_{9,6} a_6 (1 - a_6) \cdot x_2
\end{aligned}
$$

Define $\delta_6 = \delta_9 \cdot w_{9,6} a_6 (1 - a_6)$
and rewrite as

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

Networks with Multiple Output Units



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

$$\delta_6 = a_6(1 - a_6) \sum_{u=9}^{10} w_{u,6}\delta_u$$

The Backpropagation Algorithm

- Forward Pass. 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. Compute $\delta_u = a_u(1 - a_u) \sum_v w_{v,u} \delta_v$
- Compute Gradient.
  - Compute $\frac{\partial J_i}{\partial w_{u,j}} = \delta_u x_{ij}$ for input-to-hidden weights.
  - Compute $\frac{\partial J_i}{\partial w_{v,u}} = \delta_v a_{iu}$ for hidden-to-output weights.
- Take Gradient Step.

$$W := W - \eta \nabla_W J(\mathbf{x}_i)$$

Proper Initialization

- Start in the "linear" regions
  - keep all weights near zero, so that all sigmoid units are in their linear regions. This makes the whole net the equivalent of one linear threshold unit—a very simple function.
- Break symmetry.
  - 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 the "fan-in" of weight $w_{v,u}$ is the number of inputs to unit v.

Batch, Online, and Online with Momentum

- Batch. Sum the $\nabla_W J(\mathbf{x}_i)$ for each example *i*. Then take a gradient descent step.
- Online. Take a gradient descent step with each $\nabla_W J(\mathbf{x}_i)$ as it is computed.
- Momentum. Maintain an exponentially-weighted moved sum of recent

$$\Delta W^{(t+1)} := \mu \Delta W^{(t)} + \nabla_W J(\mathbf{x}_i)$$

$$W^{(t+1)} := W^{(t)} - \eta \Delta W^{(t+1)}$$

Typical values of μ are in the range [0.7, 0.95]

Softmax Output Layer

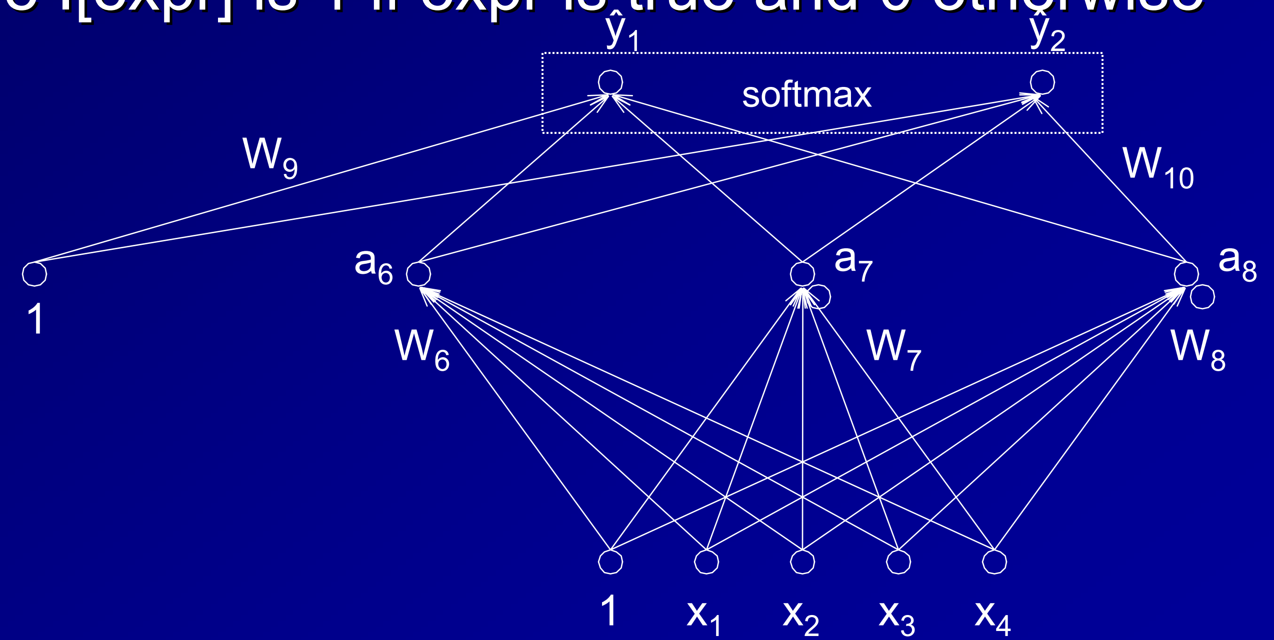
- Let a_9 and a_{10} be the output activations: $a_9 = W_9 \cdot A$, $a_{10} = W_{10} \cdot A$. Then define

$$\hat{y}_1 = \frac{\exp a_9}{\exp a_9 + \exp a_{10}} \quad \hat{y}_2 = \frac{\exp a_{10}}{\exp a_9 + \exp a_{10}}$$

- The objective function is the negative log likelihood:

$$J(W) = \sum_i \sum_k -I[y_i = k] \log \hat{y}_k$$

where I[expr] is 1 if expr is true and 0 otherwise



Neural Network Evaluation

Criterion	Perc	Logistic	LDA	Trees	Nets
Mixed data	no	no	no	yes	no
Missing values	no	no	yes	yes	no
Outliers	no	yes	no	yes	yes
Monotone transformations	no	no	no	yes	somewhat
Scalability	yes	yes	yes	yes	yes
Irrelevant inputs	no	no	no	somewhat	no
Linear combinations	yes	yes	yes	no	yes
Interpretable	yes	yes	yes	yes	no
Accurate	yes	yes	yes	no	yes