Neural Network Learning

- **Single Neuron: Linear Threshold Unit**

  Trained with LMS algorithm.

- **Multilayer Networks of Neurons**

  Trained with Backpropagation algorithm.

Learning for Classification Problems: Linear Threshold Units

$$f(x) = \begin{cases} Setosa & \text{if } w_0 + w_1 \cdot x.\textsc{SepalWidth} + w_2 \cdot x.\textsc{SepalLength} > 0 \\ Virginica & \text{otherwise} \end{cases}$$

Using vector notation the input feature vector is represented as

$$\mathbf{x} = (1, \textsc{SepalWidth}, \textsc{SepalLength})$$

and the weights are represented as

$$\mathbf{w} = (w_0, w_1, w_2)$$

The output values can be represented by $+1$ (Setosa) and -1 (Virginica).

The linear threshold unit can be written as

$$h(x) = \begin{cases} +1 & \text{if } \mathbf{w} \cdot \mathbf{x} > 0 \\ -1 & \text{otherwise} \end{cases}$$

Alternatively, define $sign(a) = 1$ if $a > 0$ and -1 otherwise. Then

$$h(x) = sign(\mathbf{w} \cdot \mathbf{x}).$$

LMS: An online, local search algorithm for LTUs

Every local-search learning algorithm requires three choices:

- **Hypothesis Space:** the search space.
- **Error Function:** often called the *loss function*.
- **Search Procedure**

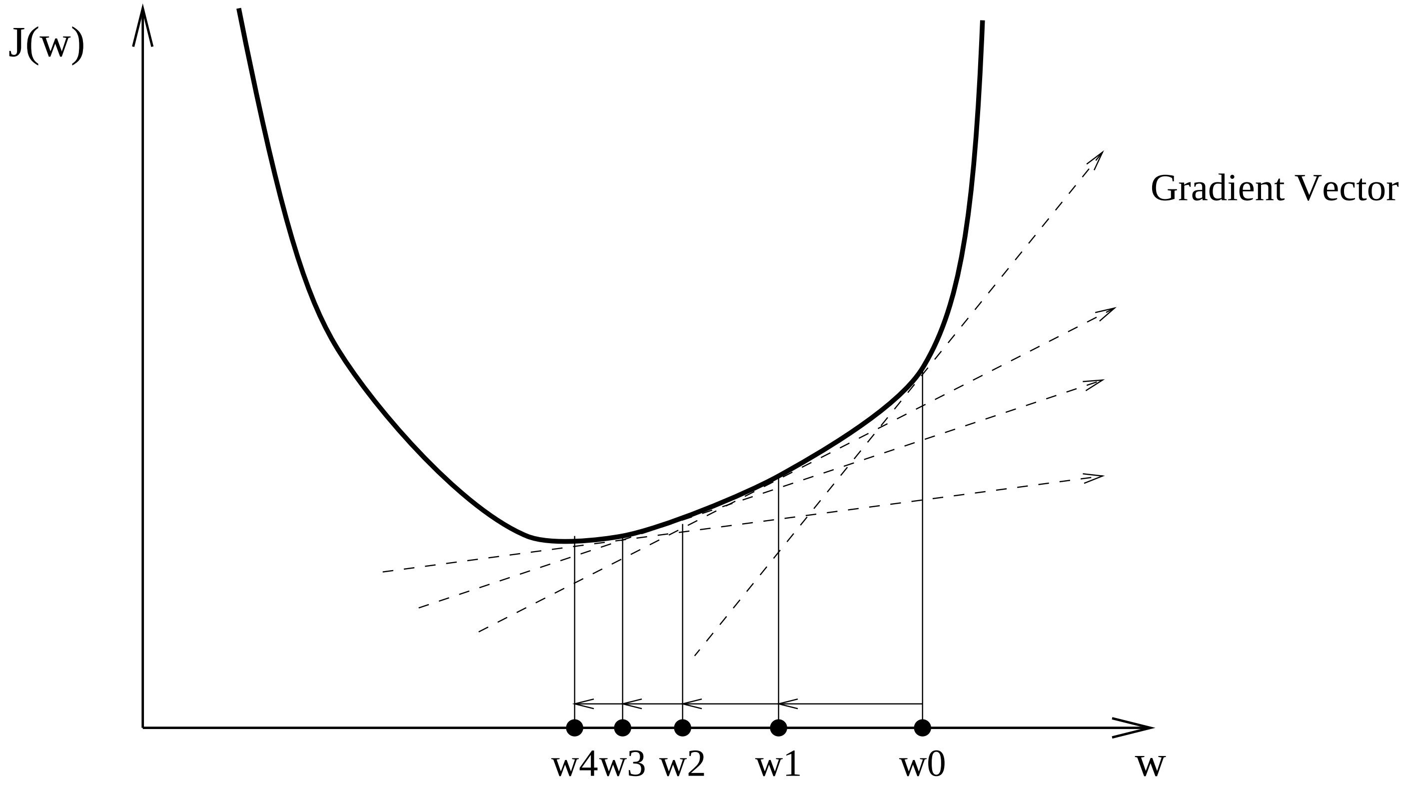
For the LMS algorithm these choices are:

- **Hypothesis Space:** Linear threshold units
- **Error Function:** Mean-squared error

$$
\begin{aligned}
J_i(\mathbf{w}) &= \frac{1}{2}\left[\mathbf{w} \cdot \mathbf{x}_i - y_i\right]^2 \\
J(\mathbf{w}) &= \frac{1}{m} \sum_{i=1}^{m} J_i(\mathbf{w})
\end{aligned}
$$

- **Search Procedure:** Gradient descent

Local Search via Gradient Descent



Start with weight vector $\mathbf{w}_0$.

Compute $\nabla J(\mathbf{w}_0) = \left(\frac{\partial J(\mathbf{w}_0)}{\partial w_0}, \frac{\partial J(\mathbf{w}_0)}{\partial w_1}, \ldots, \frac{\partial J(\mathbf{w}_0)}{\partial w_n}\right)$

Compute $\mathbf{w}_1 = w_0 - \eta \nabla J(\mathbf{w}_0)$

Repeat until convergence.

Computing the Gradient

$$
\begin{aligned}
\frac{\partial J(\mathbf{w})}{\partial w_j} &= \frac{\partial}{\partial w_j}\left(\frac{1}{m}\sum_{i=1}^{m} J_i(\mathbf{w})\right) \\
&= \frac{1}{m}\sum_{i=1}^{m}\left(\frac{\partial}{\partial w_j} J_i(\mathbf{w})\right)
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial J_i(\mathbf{w})}{\partial w_j} &= \frac{\partial}{\partial w_j}\frac{1}{2}\left[\left(\sum_j w_j x_{ij}\right) - y_i\right]^2 \\
&= \frac{1}{2}\cdot 2\cdot\left[\left(\sum_j w_j x_{ij}\right) - y_i\right]\cdot\frac{\partial}{\partial w_j}\left[\left(\sum_j w_j x_{ij}\right) - y_i\right] \\
&= \left[\left(\sum_j w_j x_{ij}\right) - y_i\right]\cdot x_{ij}
\end{aligned}
$$

Batch LMS Algorithm

Given: training examples $\langle \mathbf{x}_i, y_i \rangle$, $y = 1 \ldots m$

Let $\mathbf{w} = (0, 1, 1, 1, \ldots, 1)$ be the initial weight vector.

Repeat until convergence

  Let $\mathbf{g} = (0, 0, \ldots, 0)$ be the gradient vector.

  For $i = 1$ **to** m **do**

    $\hat{y}_i = \mathbf{w} \cdot \mathbf{x}_i$

    $\mathrm{error}_i = \hat{y}_i - y_i$

    For $j = 1$ **to** n **do**

      $g_j = g_j + \mathrm{error}_i \cdot x_{ij}$

  $\mathbf{w} := \mathbf{w} - \eta \mathbf{g}$

Online LMS Algorithm

When example $\langle \mathbf{x}_i, y_i \rangle$ is presented, compute $\nabla J_i(\mathbf{w})$, and update $\mathbf{w}$.

Let $\mathbf{w} = (0, 1, 1, 1, \ldots, 1)$ be the initial weight vector.
Repeat forever

  Accept training example i: $\langle \mathbf{x}_i, y_i \rangle$

  $\hat{y}_i = \mathbf{w} \cdot \mathbf{x}_i$

  $\text{error}_i = \hat{y}_i - y_i$

  For $j = 1$ **to** n **do**

    $g_j := \text{error}_i \cdot x_{ij}$

  $\mathbf{w} := \mathbf{w} - \eta \mathbf{g}$

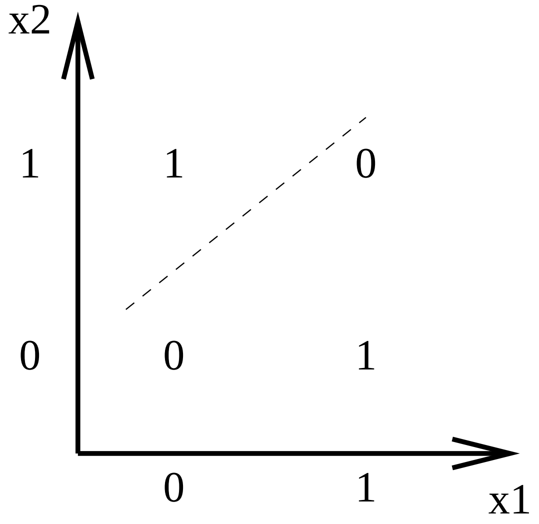
This is called **Stochastic Gradient Descent**.
It is a temporal averaging approach to computing the full gradient.
Sometimes we store the training examples and make repeated passes through them until convergence.

How LMS Manages the Fundamental Tradeoff

Two Tricks

- **Trick 1: Stop Early.** Each time we take a gradient descent step, we are expanding the hypothesis space by considering a new hypothesis. The sooner we find a consistent hypothesis, the smaller the *effective size of* H.
- **Trick 2: Restricted Hypothesis Space** A linear threshold unit cannot represent all boolean functions. Indeed, there are 2^{2^n} possible boolean functions over n features, but only $O(2^{n^2})$ of these can be represented by LTU's.

Example: An LTU cannot represent the Exclusive-OR function

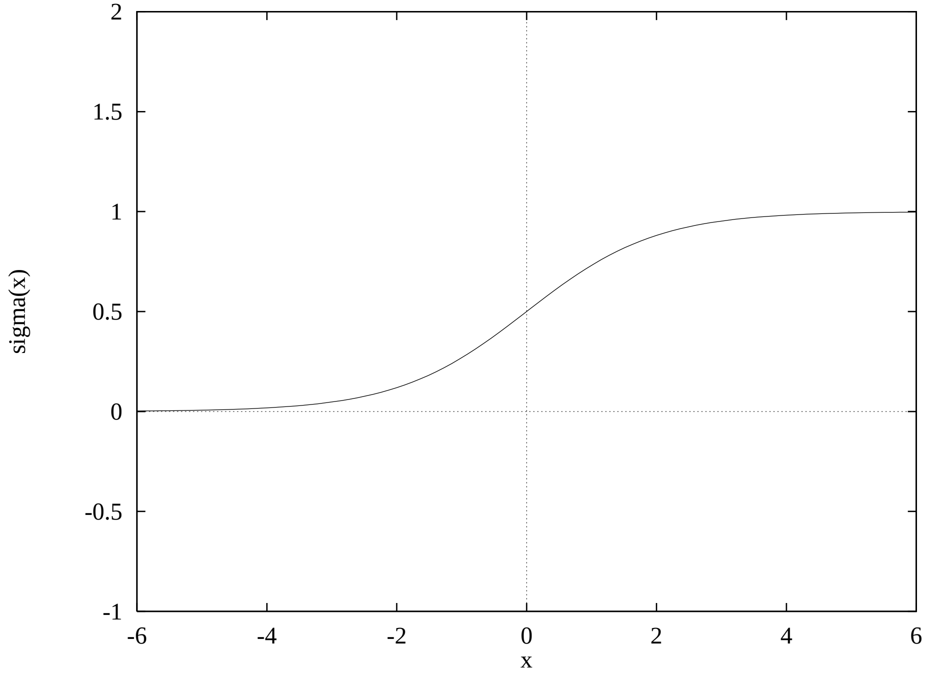


The classes are not *linearly separable*.

Approximating the LTU by the Sigmoid Function

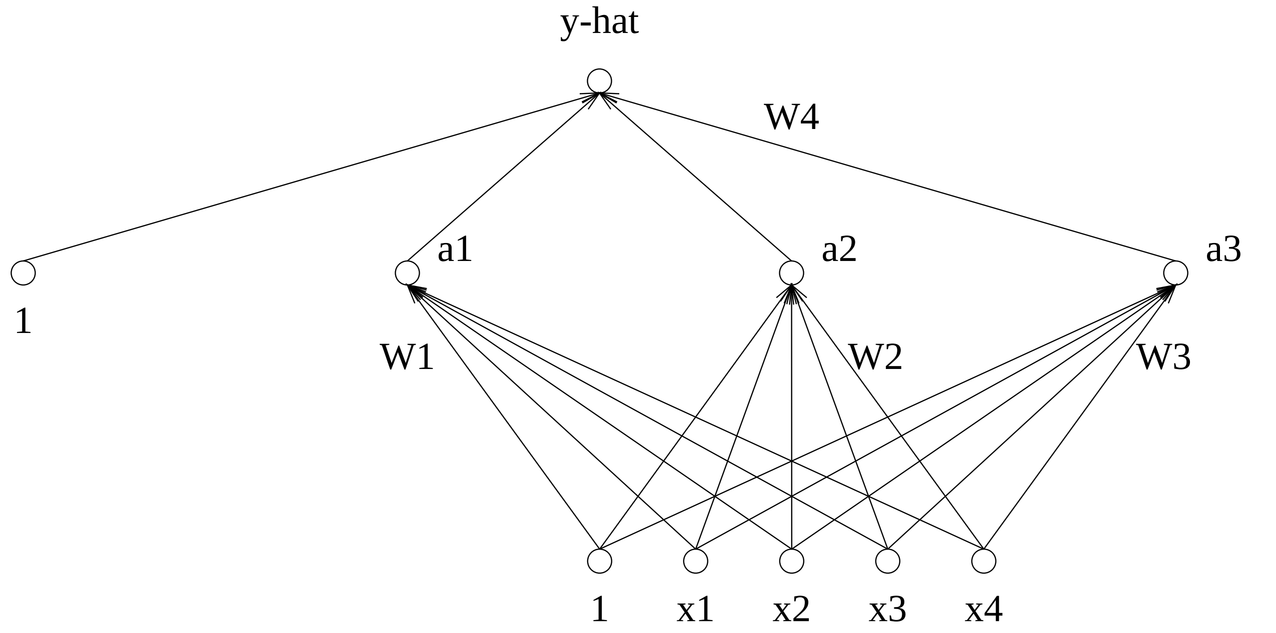
Rather than using the $sign(x)$ function, we can use the $\sigma(x)$ function defined as

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



This is a continuously differentiable approximation to the sign function.

Multilayer Neural Networks



Each unit a_1, a_2, a_3, and $\hat{y}$ computes a sigmoid function of its inputs:

$$a_1 = \sigma(W_1 \cdot X) \qquad a_2 = \sigma(W_2 \cdot X) \qquad a_3 = \sigma(W_3 \cdot X) \qquad \hat{y} = \sigma(W_4 \cdot A)$$

where $A = [1, a_1, a_2, a_3]$ is called the vector of *hidden unit activations*.

Backpropagation: Gradient Descent for Multi-layer Networks

Gradient descent search, exactly like the LMS algorithm. However, computing the gradient is more complex.

$$\hat{y} = \sigma(W_4 \cdot [1, \sigma(W_1 \cdot X), \ldots, \sigma(W_2 \cdot X), \sigma(W_3 \cdot X), \sigma(W_4 \cdot X)])$$

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

We must apply the chain rule many times to compute the gradient.
We will number the units from 1 to U and index them by u and v.
$w_{u,v}$ will be the weight connecting unit u to unit v.
$net(u)$ will be the network input (e.g., $W_3 \cdot X$) of unit u.

Derivation: Output Unit

Suppose $w_{4,2}$ is a component of W_4, the output weight vector, connecting to it from a_2. This is just the LMS case:

$$
\begin{aligned}
\frac{\partial J_i(W)}{\partial w_{4,2}} &= \frac{\partial}{\partial w_{4,2}} \frac{1}{2}(\hat{y} - y_i)^2 \\
&= \frac{1}{2} \cdot 2 \cdot (\hat{y} - y_i) \cdot \frac{\partial}{\partial w_{4,2}} (\sigma(W_4 \cdot A_i) - y_i) \\
&= (\hat{y} - y_i) \cdot \sigma(W_4 \cdot A_i)(1 - \sigma(W_4 \cdot A_i)) \cdot \frac{\partial}{\partial w_{4,2}} W_4 \cdot A_i \\
&= (\hat{y} - y_i)\hat{y}(1 - \hat{y}) \cdot a_2
\end{aligned}
$$

Define $\delta_4 = (\hat{y} - y_i)\hat{y}(1 - \hat{y})$. Then this can be written as

$$
\frac{\partial J_i(W)}{\partial w_{4,2}} = \delta_4 \cdot a_2.
$$

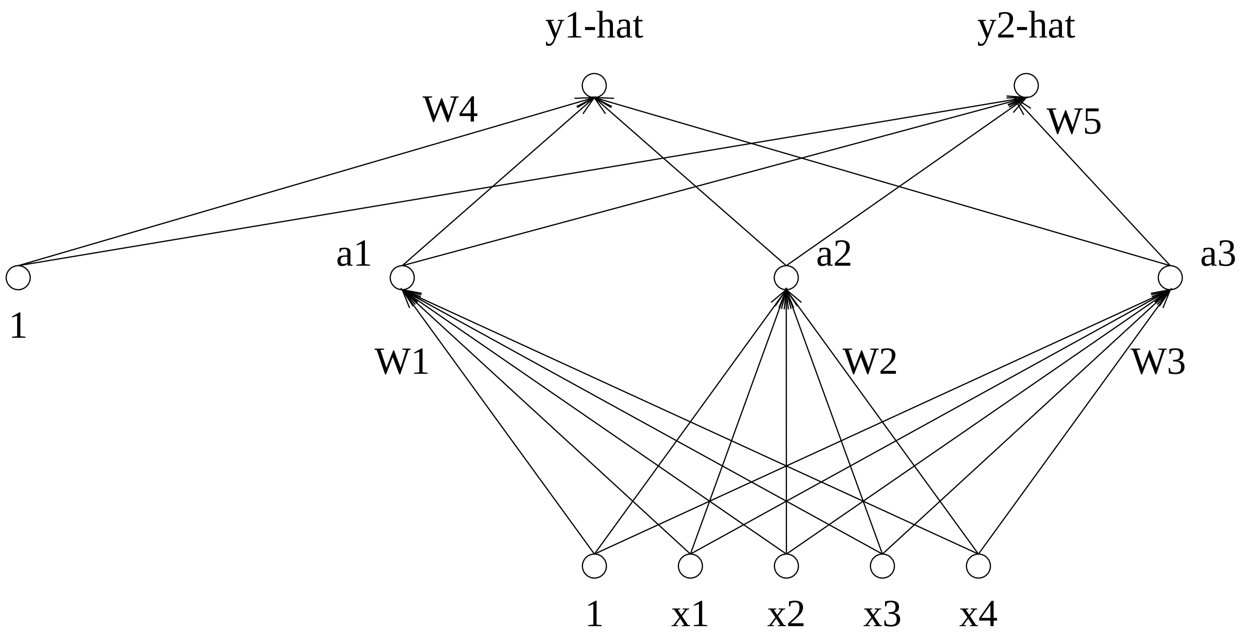
Derivation: Hidden Units

$$\begin{aligned}
\frac{\partial J_i(W)}{\partial w_{2,3}} &= (\hat{y} - y_i) \cdot \sigma(W_4 \cdot A_i)(1 - \sigma(W_4 \cdot A_i)) \cdot \frac{\partial}{\partial w_{2,3}} W_4 \cdot A_i \\
&= \delta_4 \cdot w_{4,2} \cdot \frac{\partial}{\partial w_{2,3}} \sigma(W_2 \cdot X) \\
&= \delta_4 \cdot w_{4,2} \cdot \sigma(W_2 \cdot X)(1 - \sigma(W_2 \cdot X)) \cdot \frac{\partial}{\partial w_{2,3}} (W_2 \cdot X). \\
&= \delta_4 \cdot w_{4,2} a_2 (1 - a_2) \cdot x_3.
\end{aligned}$$

Define $\delta_2 = \delta_4 \cdot w_{4,2} a_2 (1 - a_2)$. Then this can be written as

$$\frac{\partial J_i(W)}{\partial w_{2,3}} = \delta_2 x_2.$$

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:

$$\delta_2 = a_2(1 - a_2) \sum_{u=4}^{5} w_{u,2}\delta_u$$

The Backpropagation Algorithm

- **Forward pass**. Compute a_u and $\hat{y}_v$ for hidden units u and output units v.
- **Compute Errors**. Compute $\epsilon_v = (\hat{y}_v - y_v)$ for each output unit.
- **Compute Output Deltas**. Compute $\delta_v = \epsilon_v \hat{y}_v (1 - \hat{y}_v)$.
- **Compute Hidden Unit 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 Descent 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 the 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 be a random number in the range

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

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 moving sum of recent $\nabla_W J(\mathbf{x}_i)$:

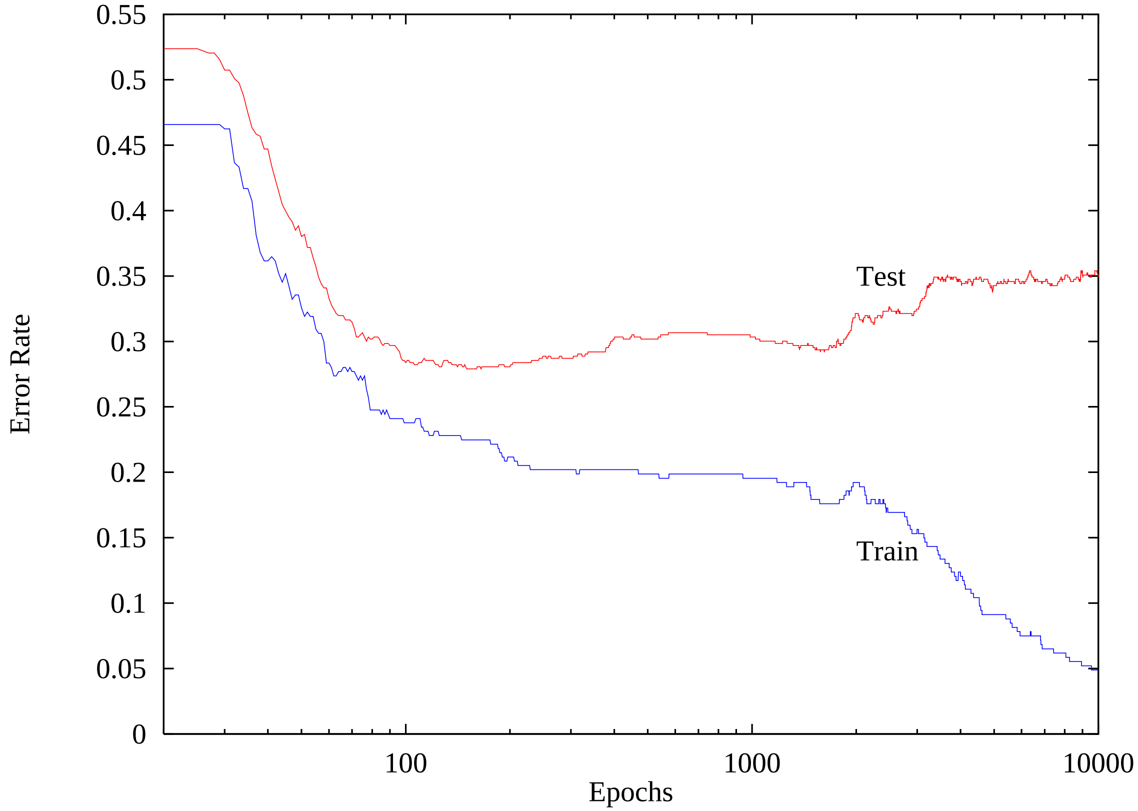
$$\Delta W^{(t+1)} := \mu \Delta W^{(t)} + \nabla_W J(\mathbf{x}_i)$$

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

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

How Neural Networks Manage the Fundamental Tradeoff

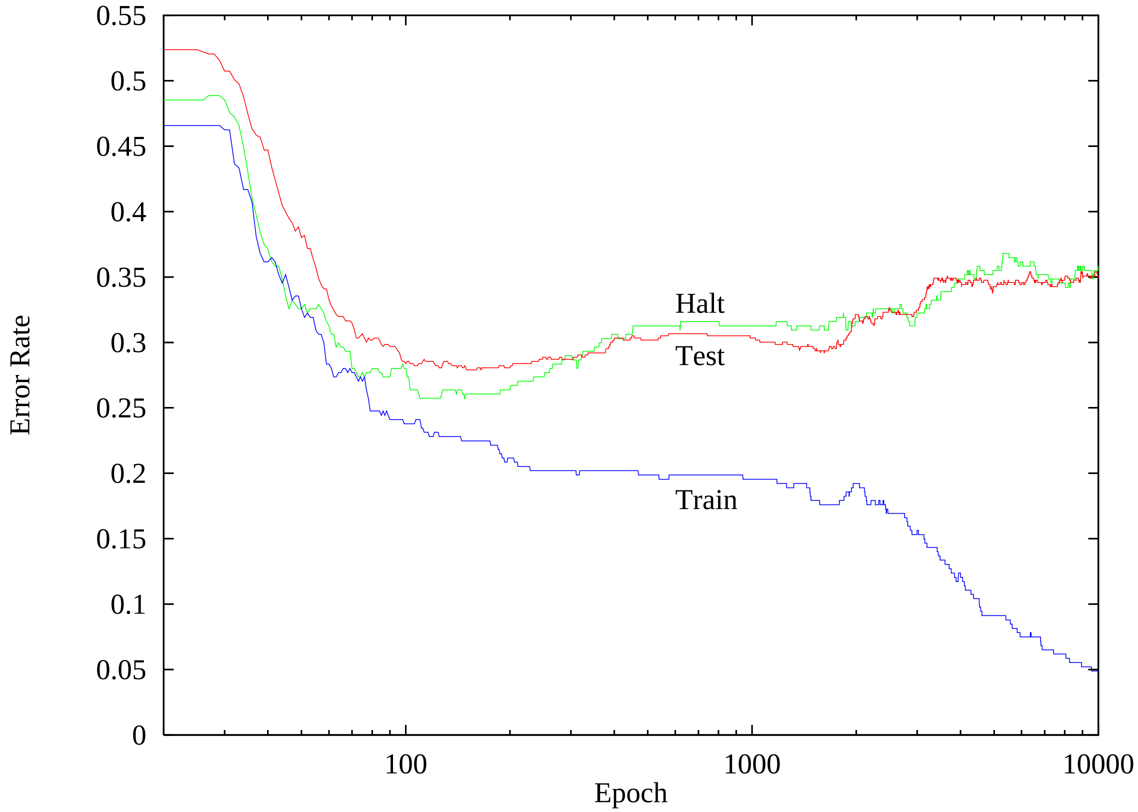
The neural network hypothesis space H is very large. This can lead to **overfitting**.



Overfitting: $h \in H$ *overfits* training set S if there exists $h' \in H$ that has higher training set error but lower test error on new data points. (More specifically, if learning algorithm A explicitly considers and rejects h' in favor of h, we say that A has overfit the data.)

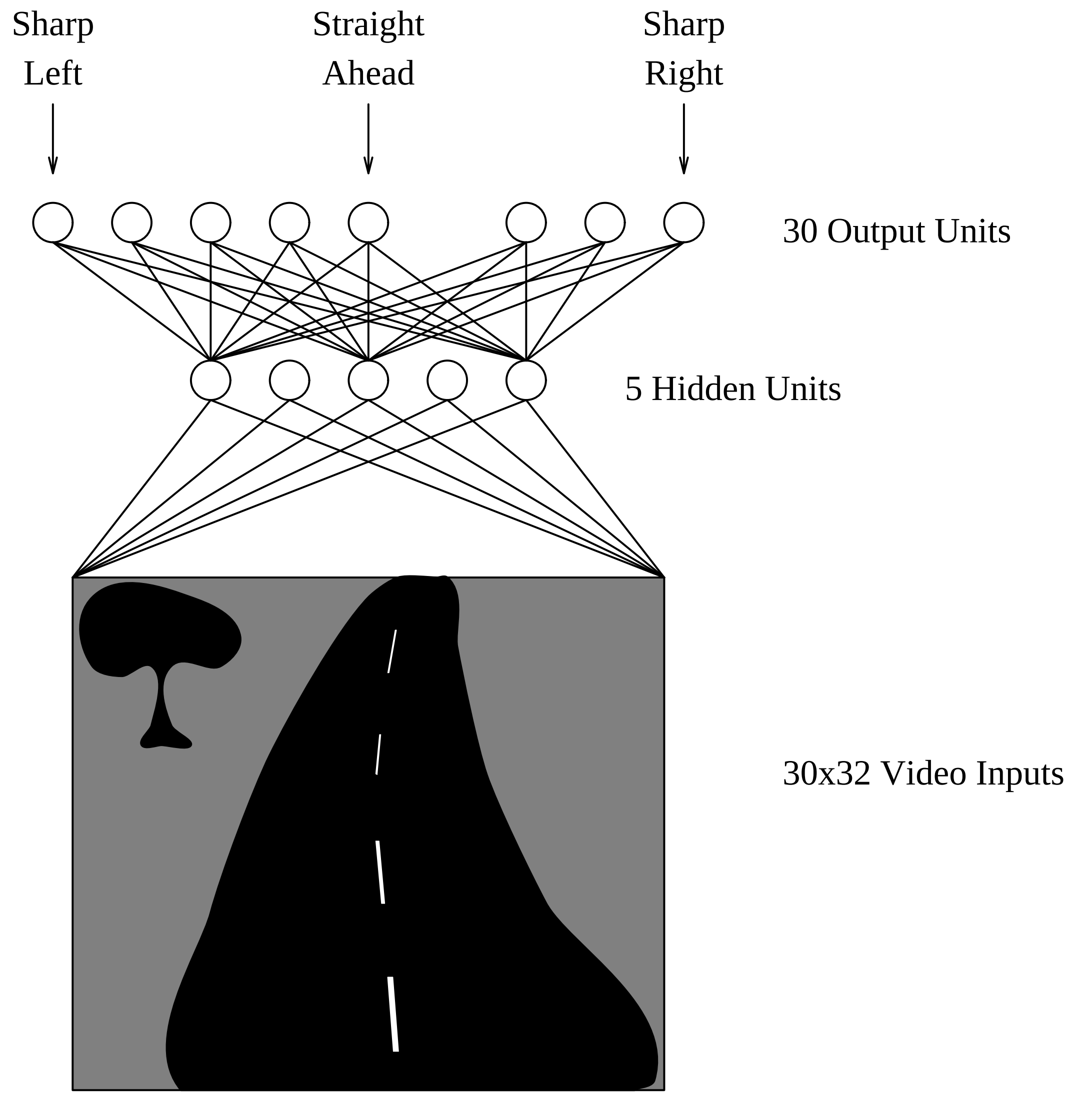
Simple Holdout Method: Early Stopping

- Subdivide S into S_{train} and S_{halt}.
- For each H_i, find h_i that best fits S_{train}.
- Measure the error rate of each h_i on S_{halt}.
- Choose h_i with the best error rate.



ALVINN (Pomerleau)

Task: Learn to steer the CMU NavLab van

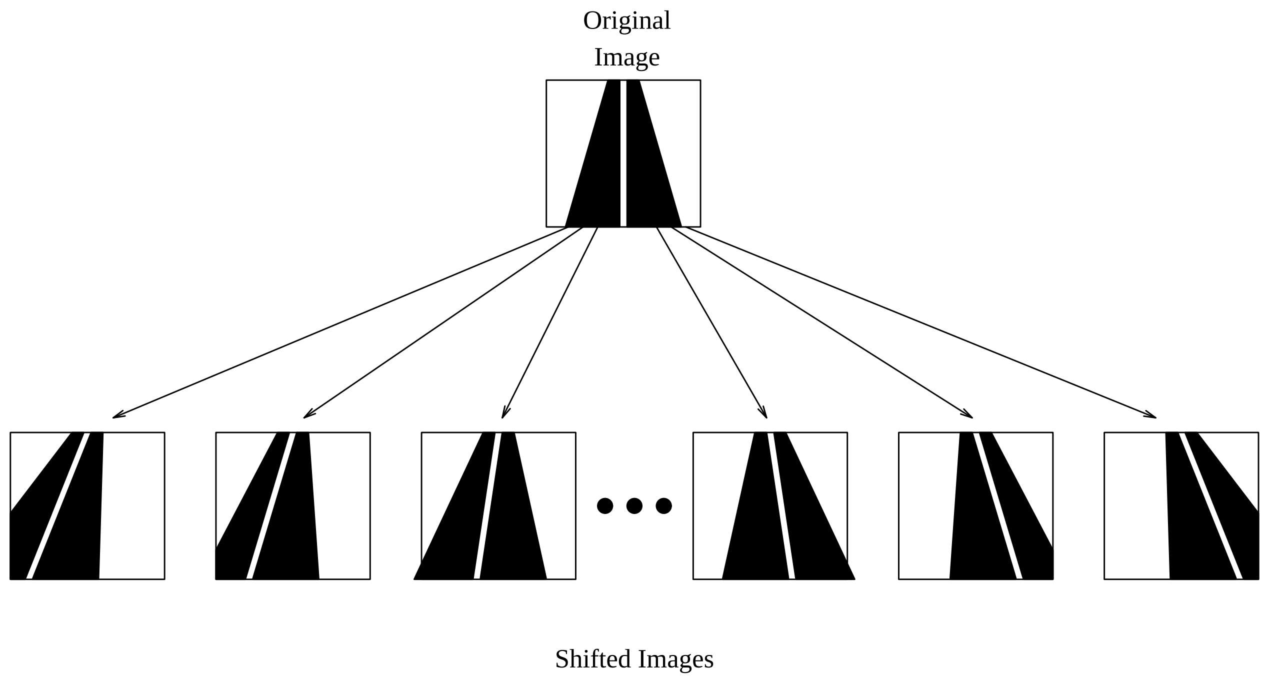


ALVINN (2)

- **Outputs encoded as overlapping range units.**
- **Supervised training from human drivers.**

  Does not provide "off road" experiences.
- **Training from perturbed images.**

  Apply camera model to shift image and driving signal.



Neural Network Summary

- **"Neural networks are the second best way of doing anything" (Denker)**

  Very robust and broadly applicable

  Usually a domain-specific algorithm will do better.

- **Large networks with early stopping gives best results**

- **Slow training**

- **Results are not easy to understand**