



DRONACHARYA
College of Engineering

Section D

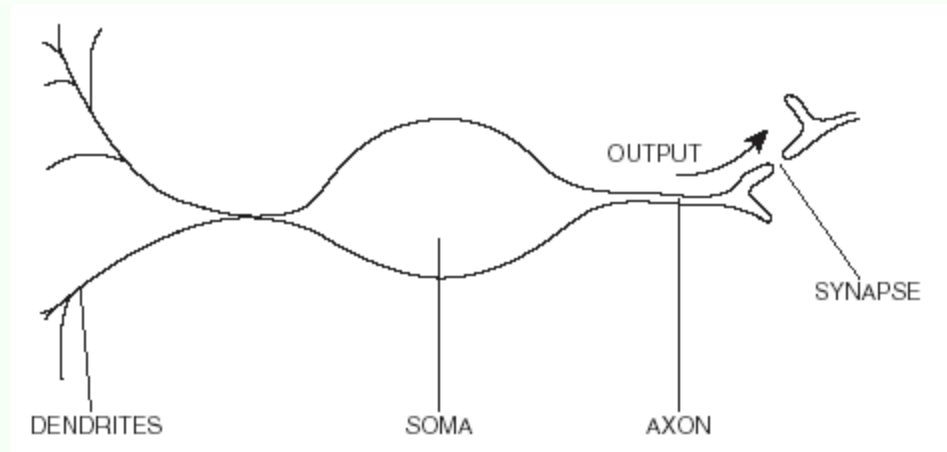
Neural Networks

Artificial Neural Networks

- Artificial neural networks (ANNs) provide a practical method for learning
 - real-valued functions
 - discrete-valued functions
 - vector-valued functions
- Robust to errors in training data
- Successfully applied to such problems as
 - interpreting visual scenes
 - speech recognition
 - learning robot control strategies

Biological Neurons

- The human brain is made up of billions of simple processing units – neurons.



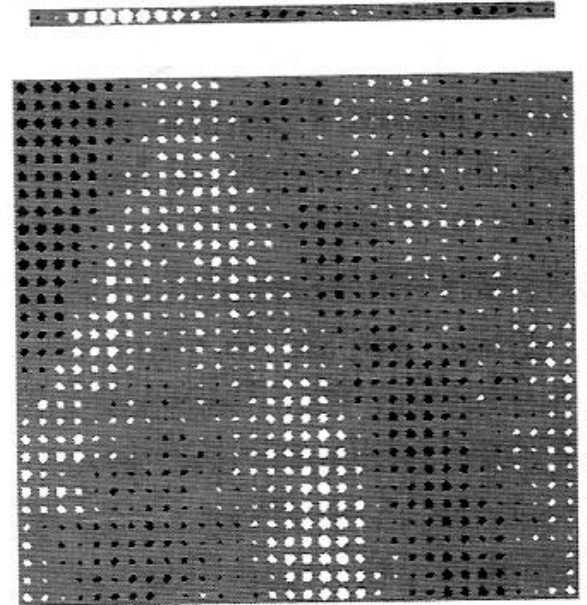
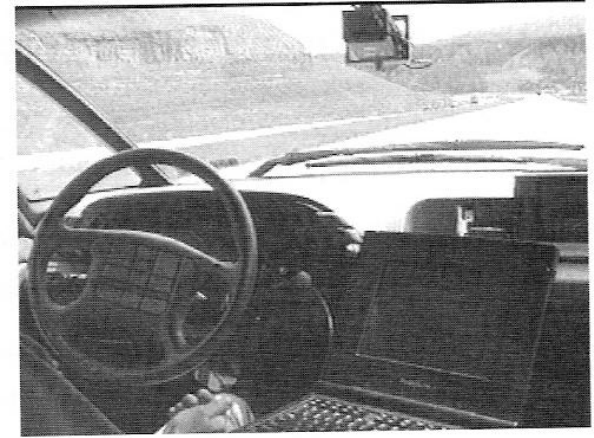
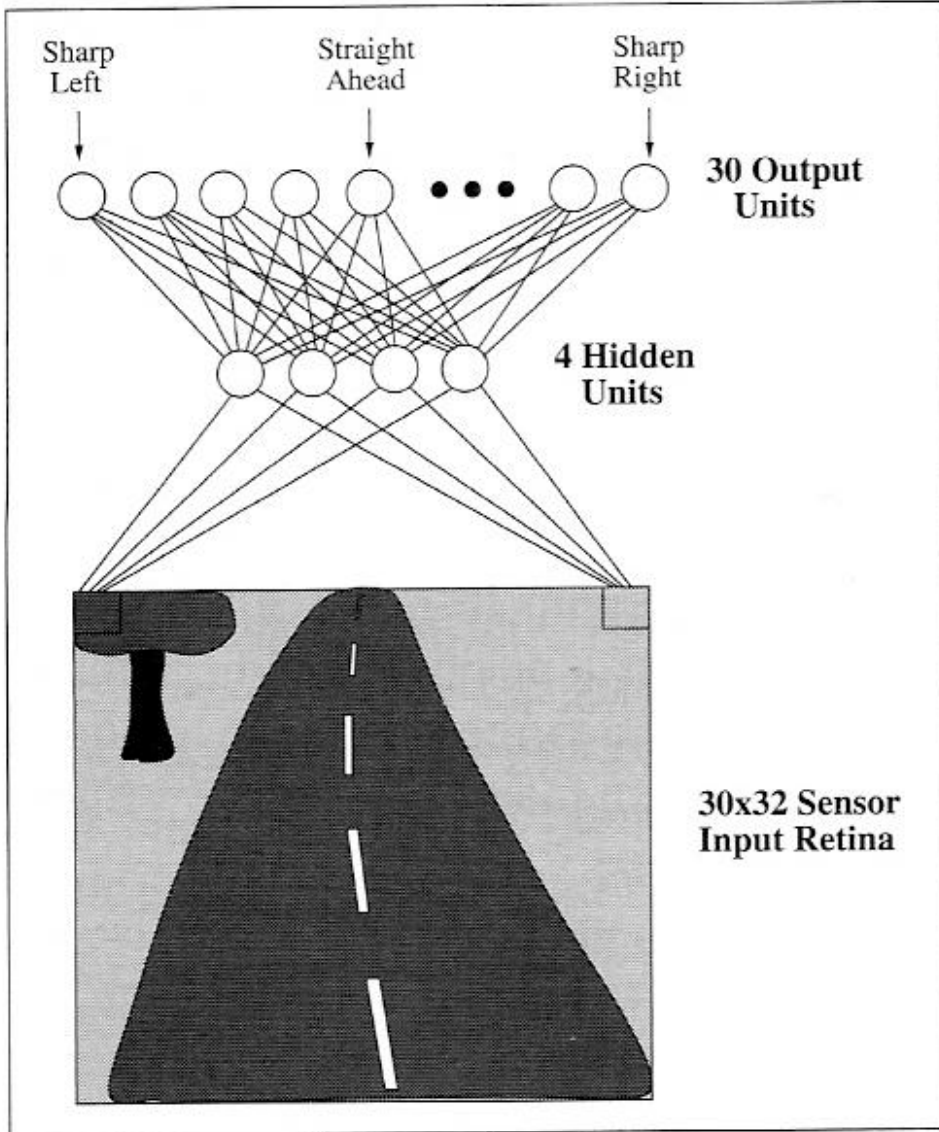
- Inputs are received on dendrites, and if the input levels are over a threshold, the neuron fires, passing a signal through the axon to the synapse which then connects to another neuron.



Neural Network Representation

- ALVINN uses a learned ANN to steer an autonomous vehicle driving at normal speeds on public highways
 - Input to network: 30x32 grid of pixel intensities obtained from a forward-pointed camera mounted on the vehicle
 - Output: direction in which the vehicle is steered
 - Trained to mimic observed steering commands of a human driving the vehicle for approximately 5 minutes

ALVINN

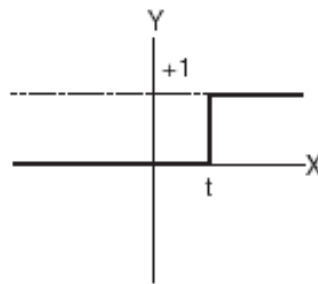


Appropriate problems

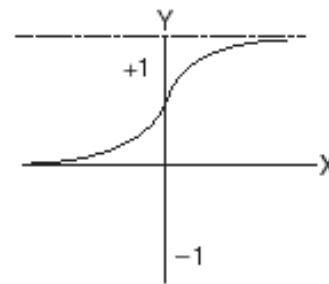
- ANN learning well-suited to problems which the training data corresponds to noisy, complex data (inputs from cameras or microphones)
- Can also be used for problems with symbolic representations
- Most appropriate for problems where
 - Instances have many attribute-value pairs
 - Target function output may be discrete-valued, real-valued, or a vector of several real- or discrete-valued attributes
 - Training examples may contain errors
 - Long training times are acceptable
 - Fast evaluation of the learned target function may be required
 - The ability for humans to understand the learned target function is not important

Artificial Neurons (1)

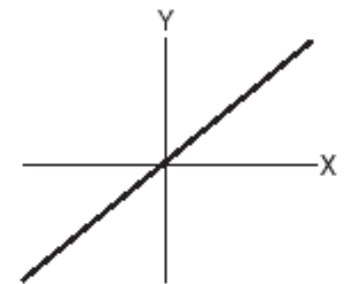
- Artificial neurons are based on biological neurons.
- Each neuron in the network receives one or more inputs.
- An activation function is applied to the inputs, which determines the output of the neuron – the activation level.
- The charts on the right show three typical activation functions.



(a) Step function



(b) Sigmoid function



(c) Linear function

Artificial Neurons (2)

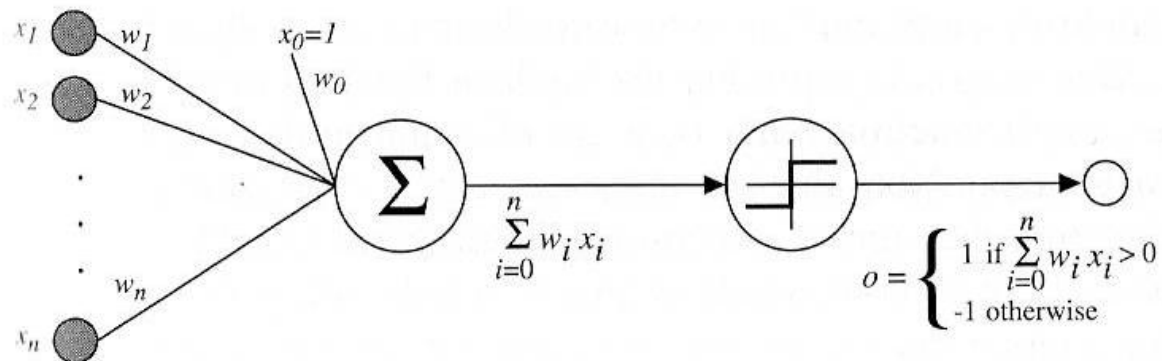
- A typical activation function works as follows:

$$X = \sum_{i=1}^n w_i x_i \quad Y = \begin{cases} +1 & \text{for } X > t \\ 0 & \text{for } X \leq t \end{cases}$$

- Each node i has a weight, w_i associated with it. The input to node i is x_i .
- t is the threshold.
- So if the weighted sum of the inputs to the neuron is above the threshold, then the neuron fires.

Perceptrons

- A perceptron is a single neuron that classifies a set of inputs into one of two categories (usually 1 or -1).
- If the inputs are in the form of a grid, a perceptron can be used to recognize visual images of shapes.
- The perceptron usually uses a step function, which returns 1 if the weighted sum of inputs exceeds a threshold, and 0 otherwise.

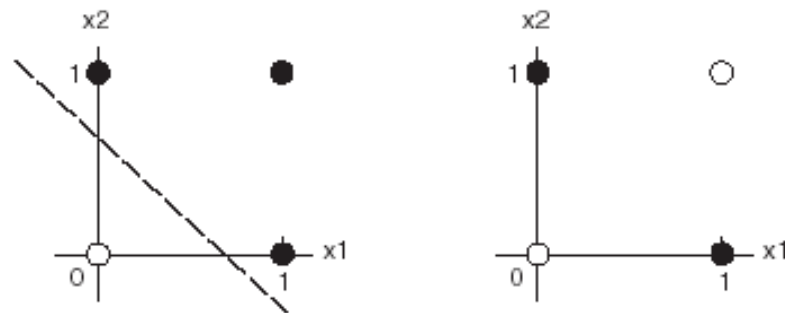


Training Perceptrons

- Learning involves choosing values for the weights
- The perceptron is trained as follows:
 - First, inputs are given random weights (usually between -0.5 and 0.5).
 - An item of training data is presented. If the perceptron mis-classifies it, the weights are modified according to the following:
$$w_i \leftarrow w_i + (a \times x_i \times (t - o))$$
 - where t is the target output for the training example, o is the output generated by the preceptron and a is the learning rate, between 0 and 1 (usually small such as 0.1)
- Cycle through training examples until successfully classify all examples
 - Each cycle known as an **epoch**

Bias of Perceptrons

- Perceptrons can only classify linearly separable functions.
- The first of the following graphs shows a linearly separable function (OR).
- The second is not linearly separable (Exclusive-OR).

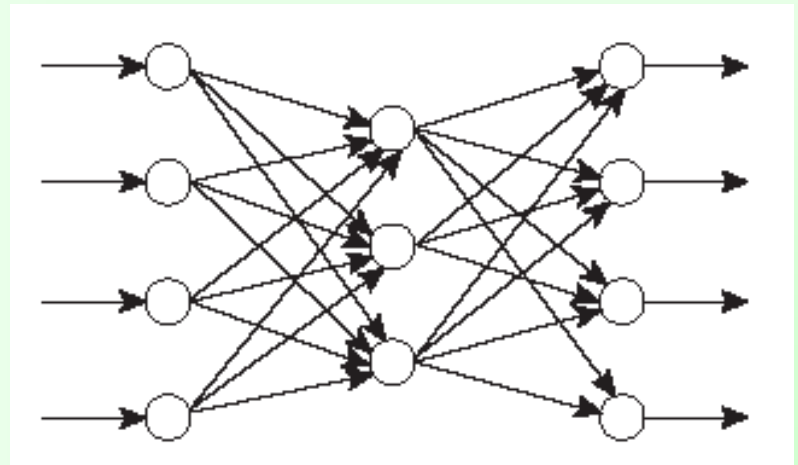


Convergence

- Perceptron training rule only converges when training examples are linearly separable and it has a small learning constant
- Another approach uses the *delta rule* and gradient descent
 - Same basic rule for finding update value
 - Changes
 - Do not incorporate the threshold in the output value (unthresholded perceptron)
 - Wait to update weight until cycle is complete
 - Converges asymptotically toward the minimum error hypothesis, possibly requiring unbounded time, but converges regardless of whether the training data are linearly separable

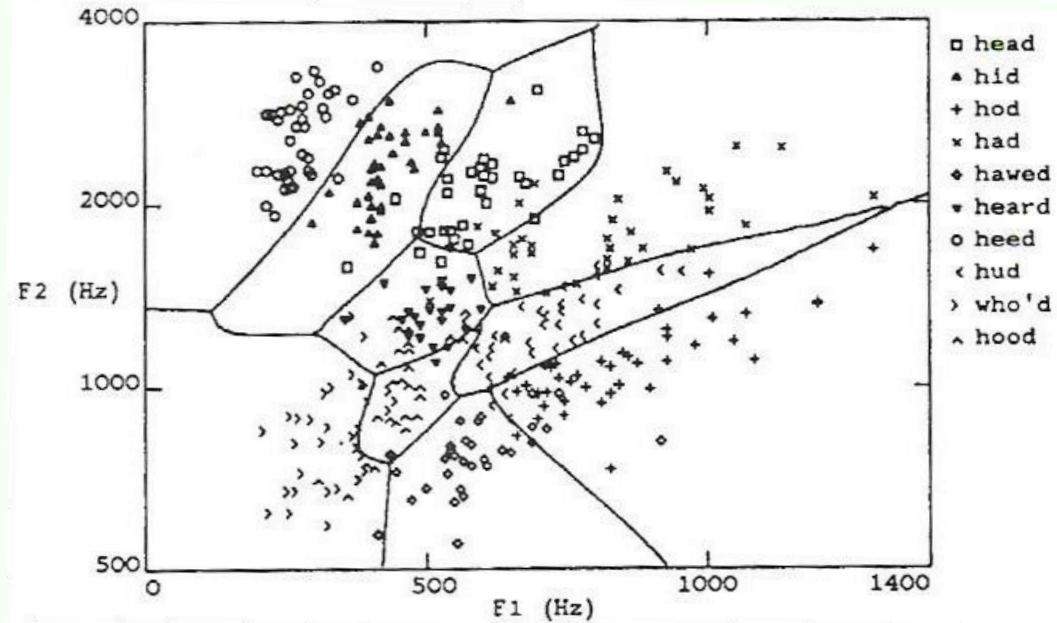
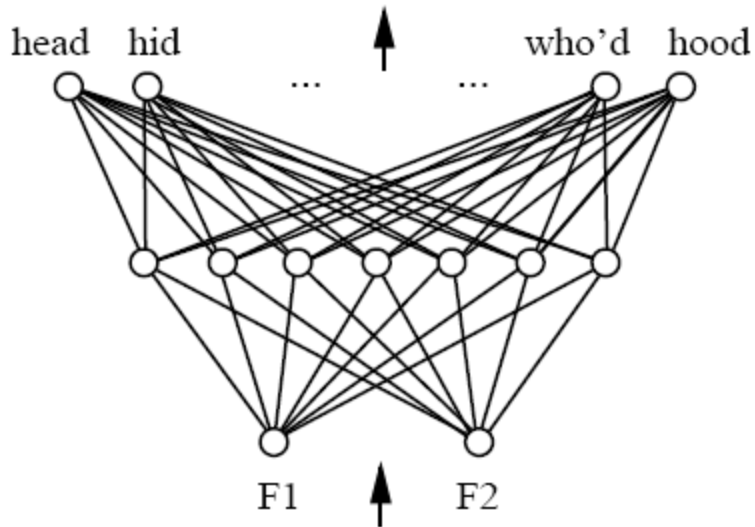
Multilayer Neural Networks

- Multilayer neural networks can classify a range of functions, including non linearly separable ones.
- Each input layer neuron connects to all neurons in the hidden layer.
- The neurons in the hidden layer connect to all neurons in the output layer.

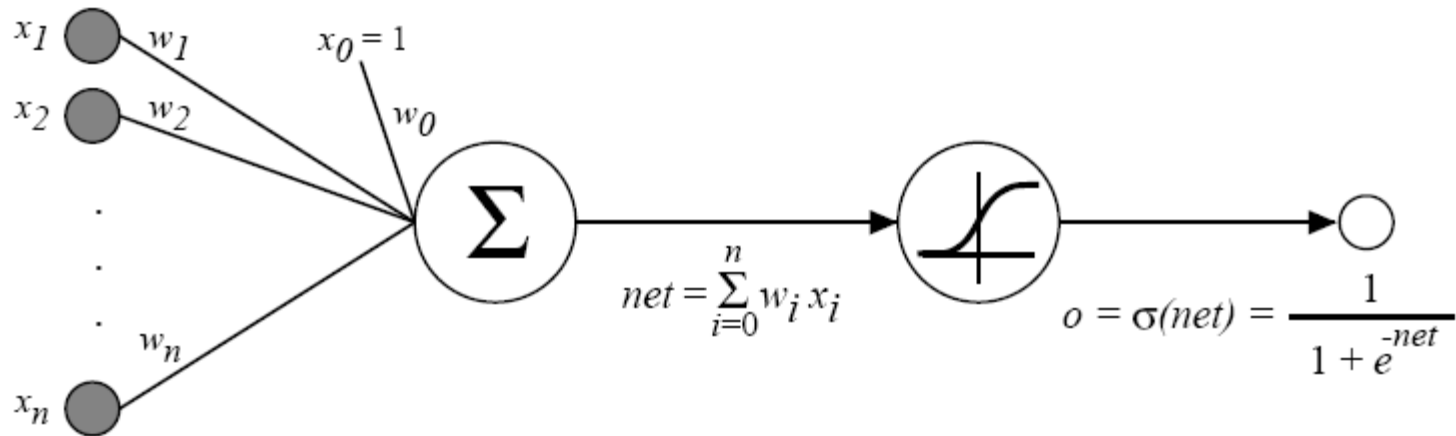


A feed-forward network

Speech Recognition ANN



Sigmoid Unit



- $\sigma(x)$ is the sigmoid function

$$\frac{1}{1 + e^{-x}}$$

- Nice property: differentiable

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

- Derive gradient descent rules to train

- One sigmoid unit - node
- Multilayer networks of sigmoid units

Backpropagation

- Multilayer neural networks learn in the same way as perceptrons.
- However, there are many more weights, and it is important to assign credit (or blame) correctly when changing weights.
- E sums the errors over all of the network output units

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in \text{outputs}} (t_{kd} - o_{kd})^2$$

Backpropagation Algorithm

- Create a feed-forward network with n_{in} inputs, n_{hidden} hidden units, and n_{out} output units.
- Initialize all network weights to small random numbers
- Until termination condition is met, Do

- For each $\langle x, t \rangle$ in training examples, Do

Propagate the input forward through the network:

1. Input the instance x to the network and compute the output o_u of every unit u in the network

Propagate the errors backward through the network:

2. For each network output unit k , calculate its error term δ_k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

3. For each hidden unit h , calculate its error term δ_h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in \text{outputs}} w_{kh} \delta_k$$

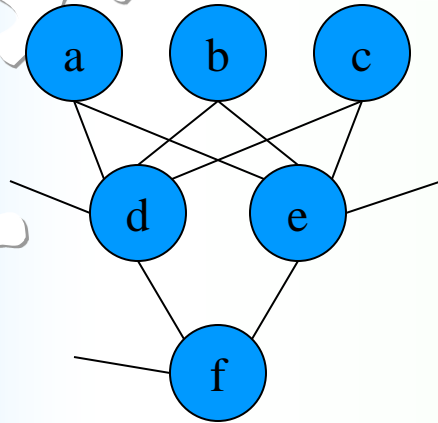
4. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

where

$$\Delta w_{ji} = \alpha \delta_j x_{ji}$$

Example: Learning AND



Training Data:

$$\text{AND}(1,0,1) = 0$$

$$\text{AND}(1,1,1) = 1$$

Alpha = 0.1

Initial Weights:

$$w_{da} = .2$$

$$w_{db} = .1$$

$$w_{dc} = -.1$$

$$w_{d0} = .1$$

$$w_{ea} = -.5$$

$$w_{eb} = .3$$

$$w_{ec} = -.2$$

$$w_{e0} = 0$$

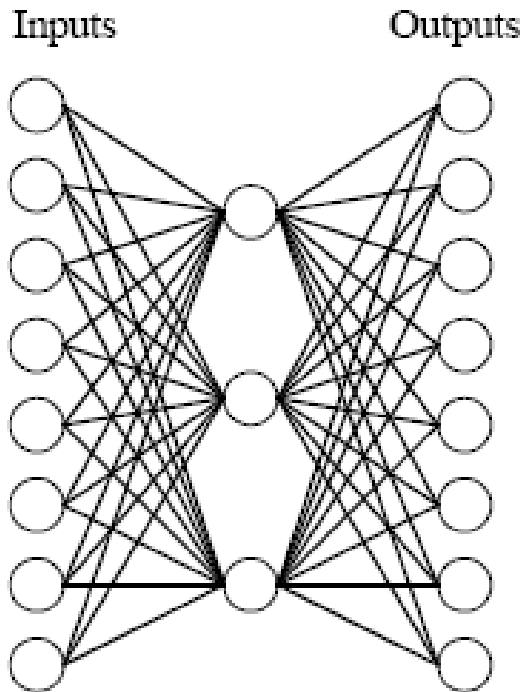
$$w_{fd} = .4$$

$$w_{fe} = -.2$$

$$w_{f0} = -.1$$

Hidden Layer representation

Target Function:



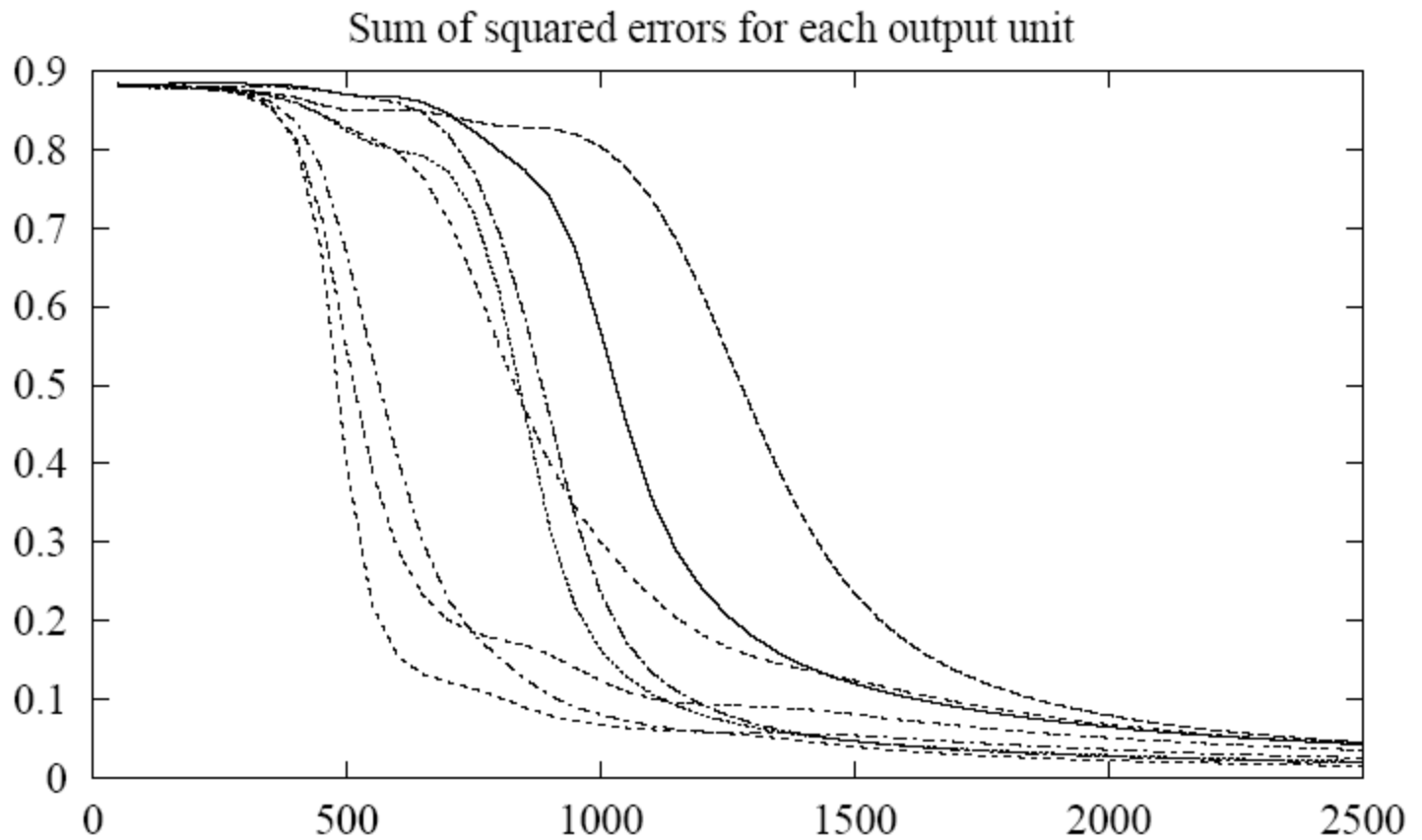
| Input | Output |
|----------|------------|
| 10000000 | → 10000000 |
| 01000000 | → 01000000 |
| 00100000 | → 00100000 |
| 00010000 | → 00010000 |
| 00001000 | → 00001000 |
| 00000100 | → 00000100 |
| 00000010 | → 00000010 |
| 00000001 | → 00000001 |

Can this be learned?

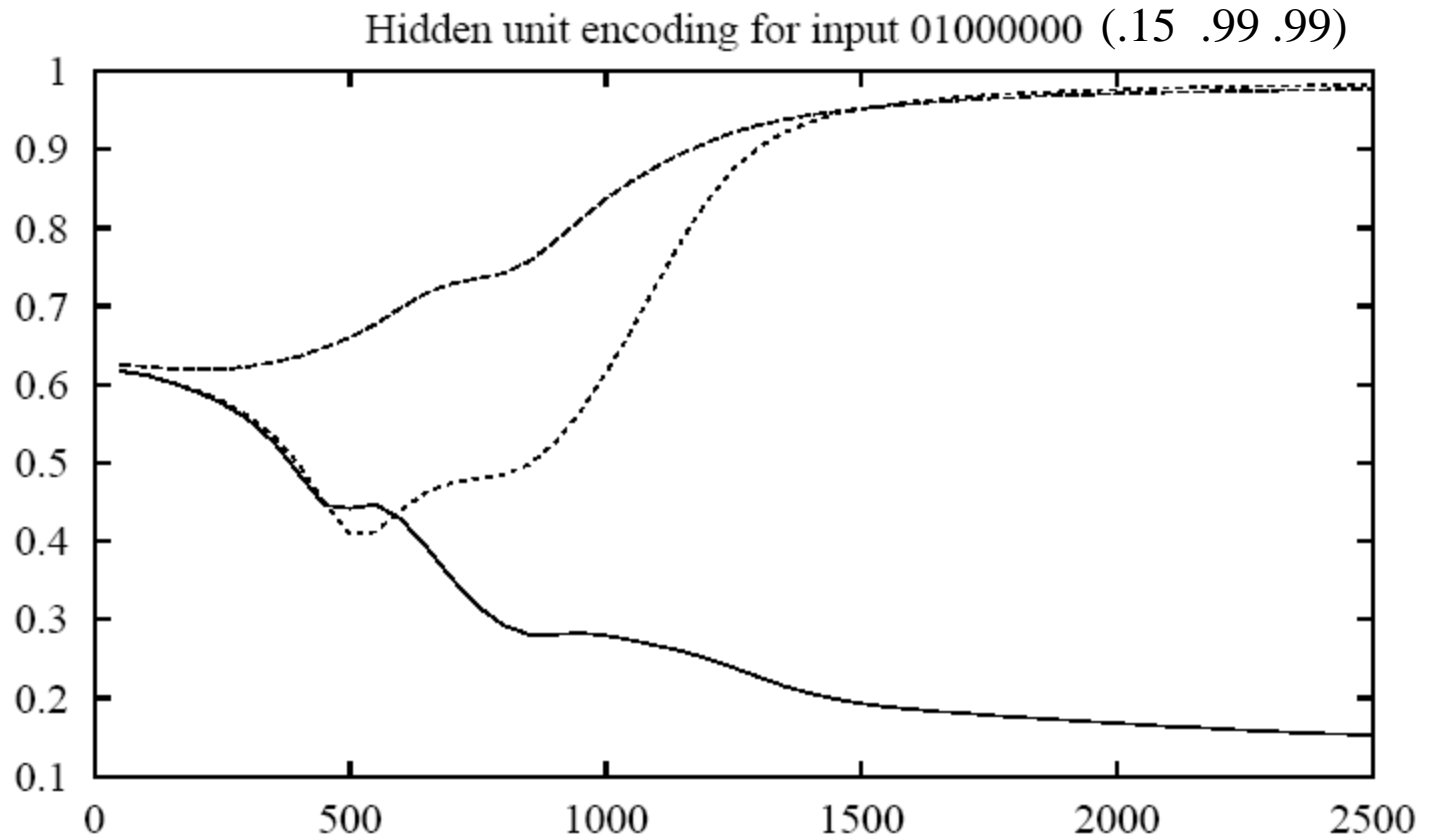
Yes

| Input | | Hidden Values | | Output |
|----------|---|---------------|---|----------|
| 10000000 | → | .89 .04 .08 | → | 10000000 |
| 01000000 | → | .15 .99 .99 | → | 01000000 |
| 00100000 | → | .01 .97 .27 | → | 00100000 |
| 00010000 | → | .99 .97 .71 | → | 00010000 |
| 00001000 | → | .03 .05 .02 | → | 00001000 |
| 00000100 | → | .01 .11 .88 | → | 00000100 |
| 00000010 | → | .80 .01 .98 | → | 00000010 |
| 00000001 | → | .60 .94 .01 | → | 00000001 |

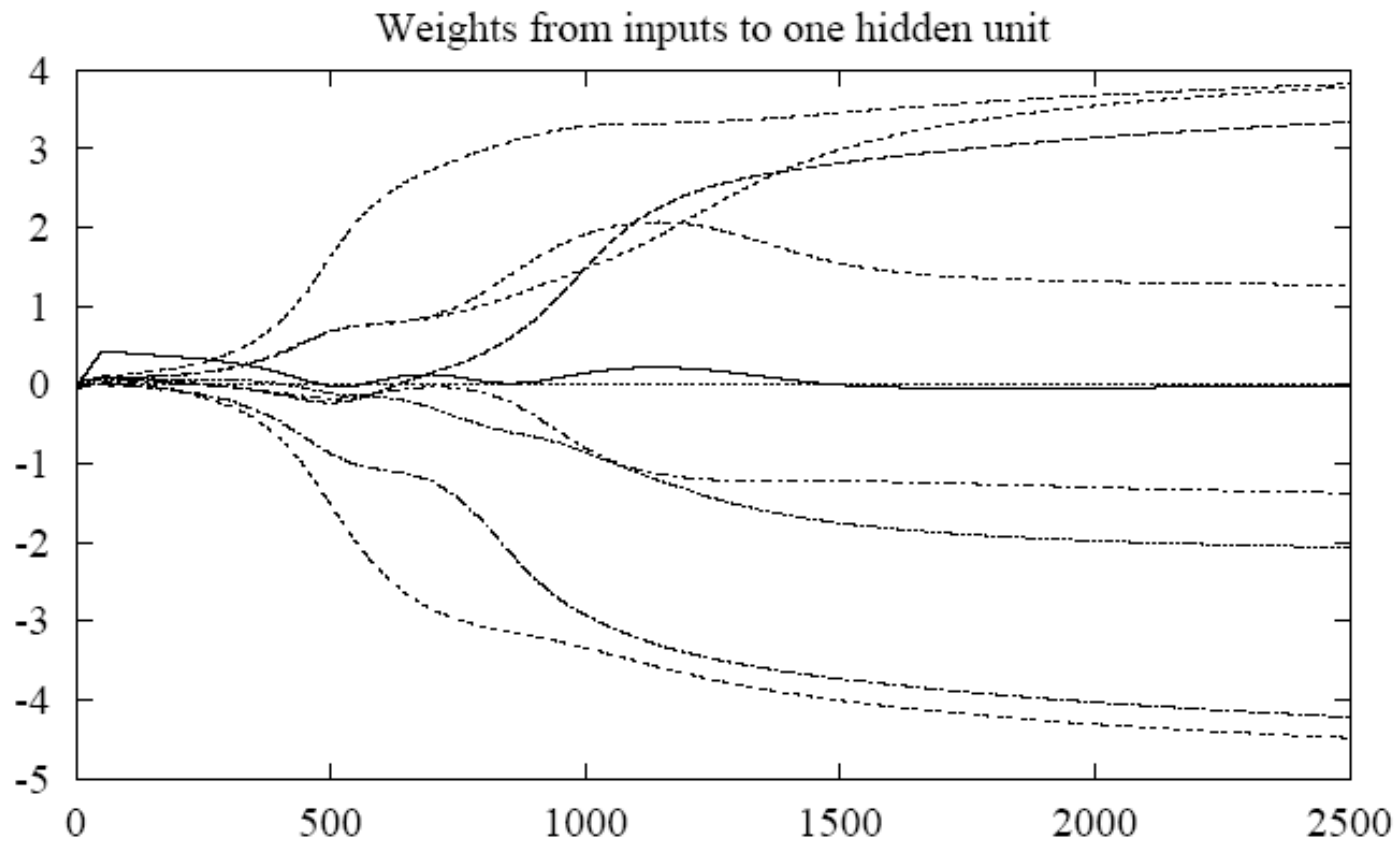
Plots of Squared Error



Hidden Unit



Evolving weights



Momentum

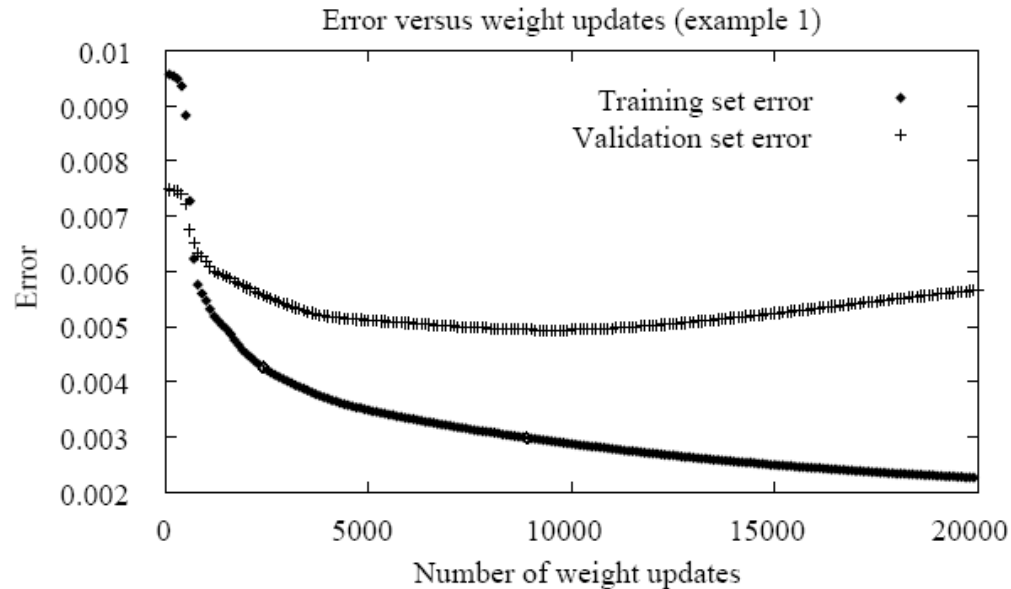
- One of many variations
- Modify the update rule by making the weight update on the n th iteration depend partially on the update that occurred in the $(n-1)$ th iteration

$$\Delta w_{ji}(n) = \alpha \delta_j x_{ji} + \beta \Delta w_{ji}(n-1)$$

- Minimizes error over training examples
- Speeds up training since it can take 1000s of iterations

When to stop training

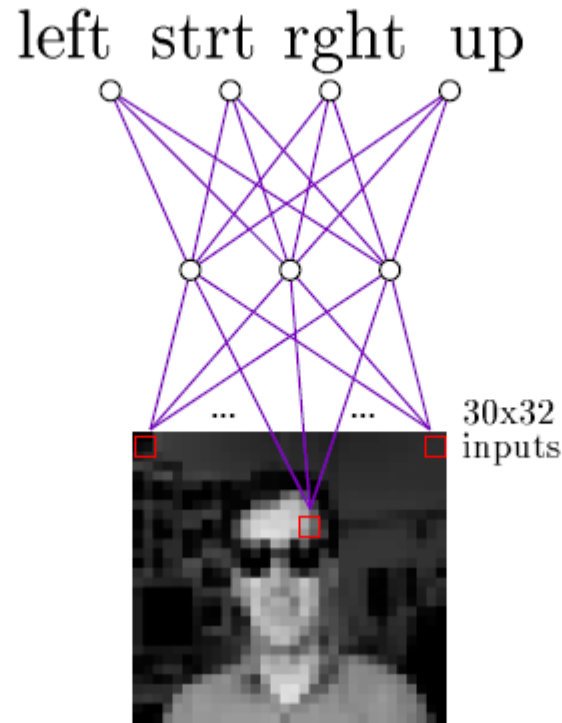
- Continue until error falls below some predefined threshold
 - Bad choice because Backpropagation is susceptible to overfitting
 - Won't be able to generalize as well over unseen data



Cross Validation

- Common approach to avoid overfitting
- Reserve part of the training data for testing
- m examples are partitioned into k disjoint subsets
- Run the procedure k times
 - Each time a different one of these subsets is used as validation
- Determine the number of iterations that yield the best performance
- Mean of the number of iterations is used to train all n examples

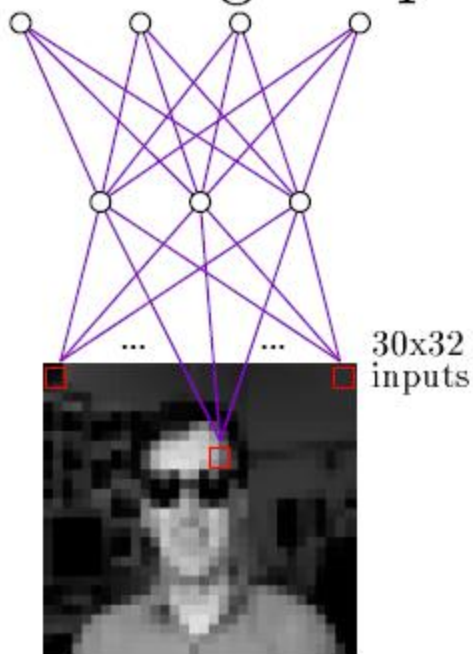
Neural Nets for Face Recognition



Typical input images

Hidden Unit Weights

left strt right up



Learned Weights

