

# Intro. to machine learning (CSI 5325)

## Lecture 7: neural networks

Greg Hamerly

Some content from Tom Mitchell.

1 Connectionist models

2 Threshold units

3 Gradient descent

# Connectionist Models

Consider humans:

- Neuron switching time  $\approx .001$  second
  - Number of neurons  $\approx 10^{10}$
  - Connections per neuron  $\approx 10^{4-5}$
  - Scene recognition time  $\approx .1$  second
  - 100 inference steps doesn't seem like enough
- lots of parallel computation

Properties of artificial neural nets (ANN's):

- Many neuron-like threshold switching units
- Many weighted interconnections among units
- Highly parallel, distributed process
- Emphasis on tuning weights automatically

# When to Consider Neural Networks

- Input is high-dimensional discrete or real-valued (e.g. raw sensor input)
- Output is discrete or real valued
- Output is a vector of values
- Possibly noisy data
- Form of target function is unknown
- Human readability of result is unimportant

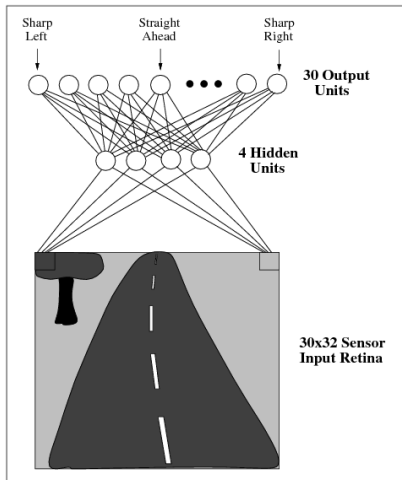
## Examples:

- Speech phoneme recognition [Waibel]
- Image classification [Kanade, Baluja, Rowley]
- Emotion identification [Cottrell, Dailey]
- Detecting fraudulent credit card transactions [HNC, Fair Isaac]

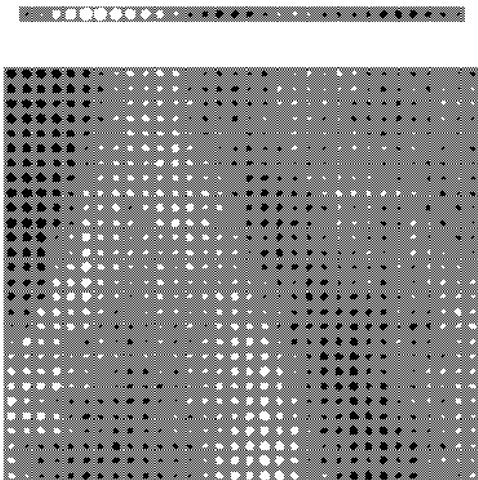
## CMU's ALVINN drives 70 mph on highways



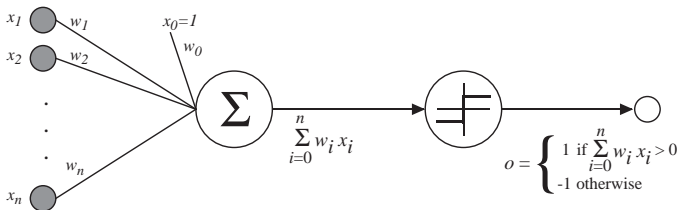
# ALVINN's basic network configuration



# ALVINN's input/output weights for a hidden unit



# Perceptron

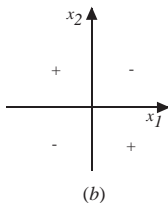
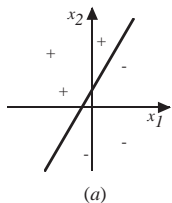


$$o(x_1, \dots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \dots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Sometimes we'll use simpler vector notation:

$$o(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w} \cdot \vec{x} > 0 \\ -1 & \text{otherwise.} \end{cases}$$

# Decision Surface of a Perceptron



Represents some useful functions

- What weights represent  $g(x_1, x_2) = \text{AND}(x_1, x_2)$ ?

But some functions are not representable...

- e.g., not linearly separable
- Therefore, we'll want networks of perceptrons...

## Visualizing the weight vector and decision surface

Consider the weight vector  $\vec{w} = [w_1, w_2]$

What does it mean? What does it look like? (example on the board)

The vector  $\vec{w}$  is perpendicular (aka normal) to the decision surface.

$\vec{w}$  points toward the positive half-space.

The bias term  $w_0$  moves the decision surface away from the origin.

# Perceptron training rule

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = \eta(t - o(\vec{x}))x_i$$

- $t = c(\vec{x})$  is the target value
- $o(\vec{x})$  is the perceptron output
- $\eta$  is a small constant (e.g., .1) called the *learning rate*

# Perceptron training rule

We can prove the training rule will converge...

- ...if the training data are linearly separable
- ...and  $\eta$  is sufficiently small

(If these conditions aren't true, then the rule may never converge...)

# The delta rule

The *delta rule* approximates the perceptron training rule. It...

- converges on a best-fit approximation if the training examples aren't linearly separable
- uses *gradient descent* to search the hypothesis space of possible weights

We also eliminate the thresholding on the output (so output can be any real number).

# Gradient Descent

Consider the simple, non-thresholded *linear unit*, where

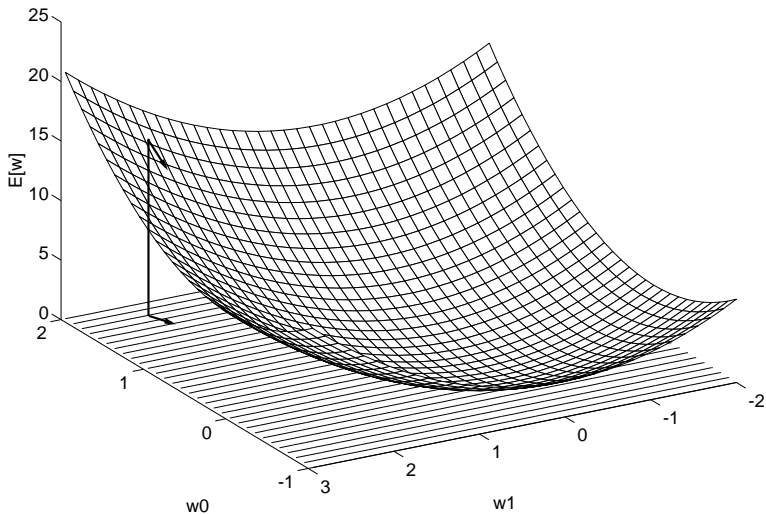
$$o(\vec{x}) = w_0 + w_1x_1 + \cdots + w_nx_n$$

Let's choose an error function over training examples  $D$ :

$$E[\vec{w}] \equiv \frac{1}{2} \sum_{\vec{x} \in D} (t(\vec{x}) - o(\vec{x}))^2$$

Now we want to learn  $w_i$ 's that minimize the squared error  $E[\vec{w}]$ .

# Visualizing $E[\vec{w}]$ (for two weights)



# Gradient Descent

Gradient

$$\nabla E[\vec{w}] \equiv \left[ \frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right]$$

Training rule:

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

i.e.,

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

# Gradient Descent

$$\begin{aligned}\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{\vec{x}} (t(\vec{x}) - o(\vec{x}))^2 \\ &= \frac{1}{2} \sum_{\vec{x}} \frac{\partial}{\partial w_i} (t(\vec{x}) - o(\vec{x}))^2 \\ &= \frac{1}{2} \sum_{\vec{x}} 2(t(\vec{x}) - o(\vec{x})) \frac{\partial}{\partial w_i} (t(\vec{x}) - o(\vec{x})) \\ &= \sum_{\vec{x}} (t(\vec{x}) - o(\vec{x})) \frac{\partial}{\partial w_i} (t(\vec{x}) - \vec{w} \cdot \vec{x}) \\ &= \sum_{\vec{x}} (t(\vec{x}) - o(\vec{x})) (-x_i)\end{aligned}$$

# Gradient Descent

## Gradient-Descent(*training\_examples*, $\eta$ )

Each training example is a pair of the form  $\langle \vec{x}, t \rangle$ , where  $\vec{x}$  is the vector of input values, and  $t$  is the target output value.  $\eta$  is the learning rate (e.g., .05).

- Initialize each  $w_i$  to some small random value
- Until the termination condition is met, Do
  - Initialize each  $\Delta w_i$  to zero.
  - For each  $\langle \vec{x}, t \rangle$  in *training\_examples*, Do
    - Input the instance  $\vec{x}$  to the unit and compute the output  $o$
    - For each linear unit weight  $w_i$ , Do

$$\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i$$

- For each linear unit weight  $w_i$ , Do

$$w_i \leftarrow w_i + \Delta w_i$$

# Summary

Perceptron training rule guaranteed to succeed if

- Training examples are linearly separable
- Sufficiently small learning rate  $\eta$

Linear unit training rule uses gradient descent

- Guaranteed to converge to hypothesis with minimum squared error
- Given sufficiently small learning rate  $\eta$
- Even when training data contains noise
- Even when training data not separable by  $H$

# Incremental (Stochastic) Gradient Descent

## **Batch mode** Gradient Descent:

Do until satisfied

- 1 Compute the gradient  $\nabla E_D[\vec{w}]$
- 2  $\vec{w} \leftarrow \vec{w} - \eta \nabla E_D[\vec{w}]$

## **Incremental mode** Gradient Descent:

Do until satisfied

- For each training example  $\vec{x}$  in  $D$ 
  - 1 Compute the gradient  $\nabla E_{\vec{x}}[\vec{w}]$
  - 2  $\vec{w} \leftarrow \vec{w} - \eta \nabla E_{\vec{x}}[\vec{w}]$

# Incremental (Stochastic) Gradient Descent

$$E_D[\vec{w}] \equiv \frac{1}{2} \sum_{\vec{x} \in D} (t(\vec{x}) - o(\vec{x}))^2$$

$$E_{\vec{x}}[\vec{w}] \equiv \frac{1}{2} (t(\vec{x}) - o(\vec{x}))^2$$

*Incremental Gradient Descent* can approximate *Batch Gradient Descent* arbitrarily closely if  $\eta$  made small enough