# Neural Networks

TM Chapter 4

# Outline

- Threshold units
- Gradient descent
- Multilayer networks
- Backpropagation
- Hidden layer representations
- Alternative loss functions

# Properties of Neural Nets

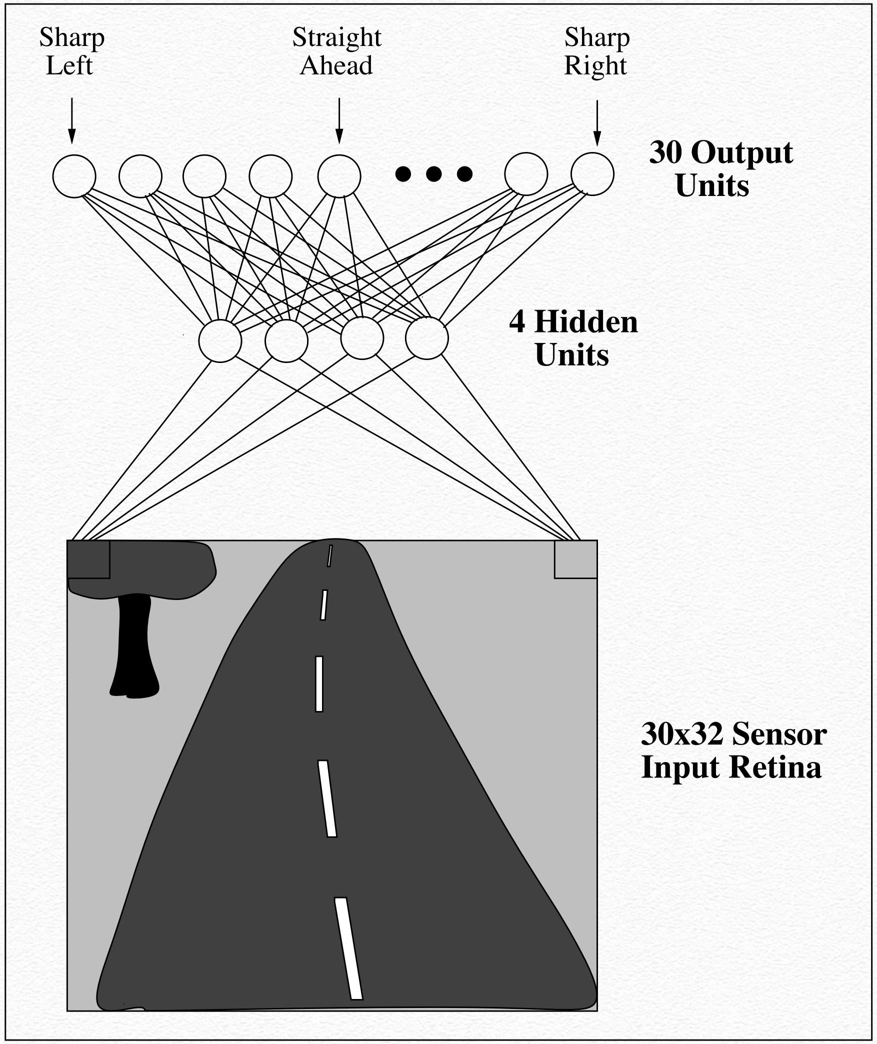
- Many neuron-like threshold switching units
- Many weighted interconnections among units
- Highly parallel, distributed process
- Emphasis on tuning weights automatically

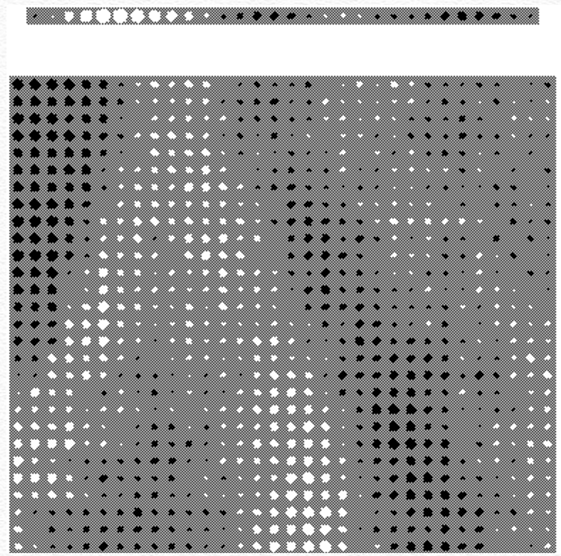
# Why Study Neural Nets?

| | DT Learning | Neural Nets |
|---|---|---|
| Target fn/output | Discrete | Discrete/real vector |
| Input instance | Discrete | Discrete/real, high-dim |
| Training data | Robust to noise | Robust to noise |
| Hypothesis space | Complete, expressive | Restricted: #hidden units (hard bias), expressive |
| Search strategy | Incomplete: prefer shorter tree (soft bias)<br><br>Refine search using all examples<br><br>No backtracking | Incomplete: prefer smaller weights (soft bias)<br><br>Gradient ascent/descent<br>batch mode: all examples<br>stochastic: mini-batches |
| Training time | Short | Long |
| Prediction time | Fast | Fast |
| Interpretability | White-box | Black-box |

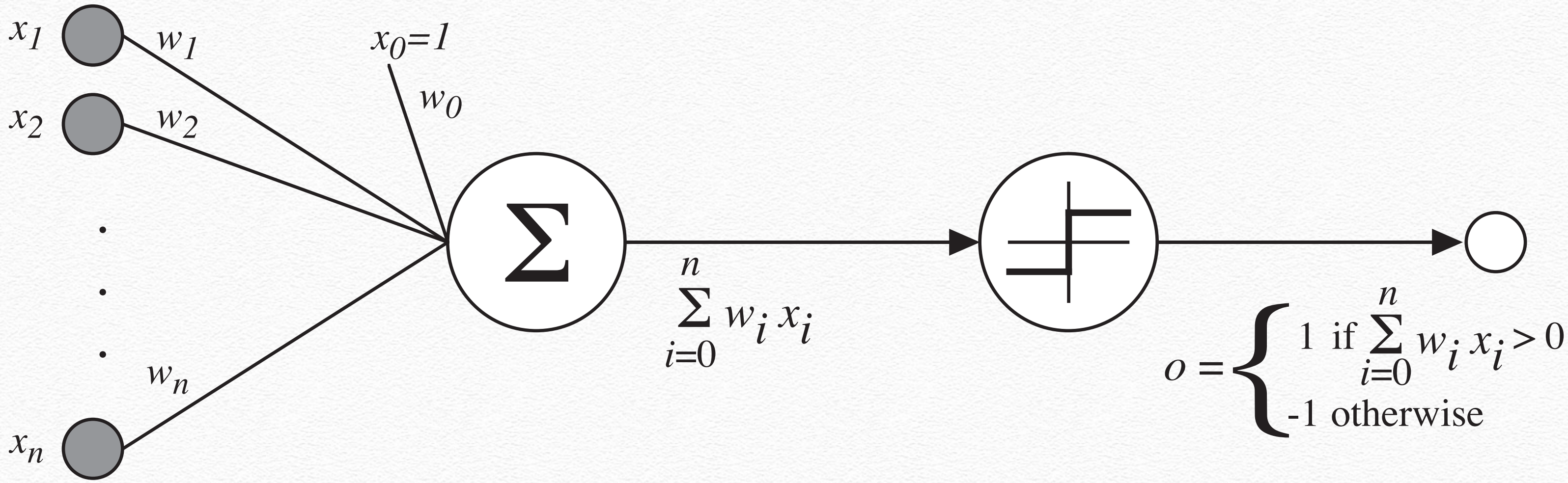
# ALVINN drives 70 mph on highways

https://papers.nips.cc/paper/95-alvinn-an-autonomous-land-vehicle-in-a-neural-network.pdf



# Perceptron Unit



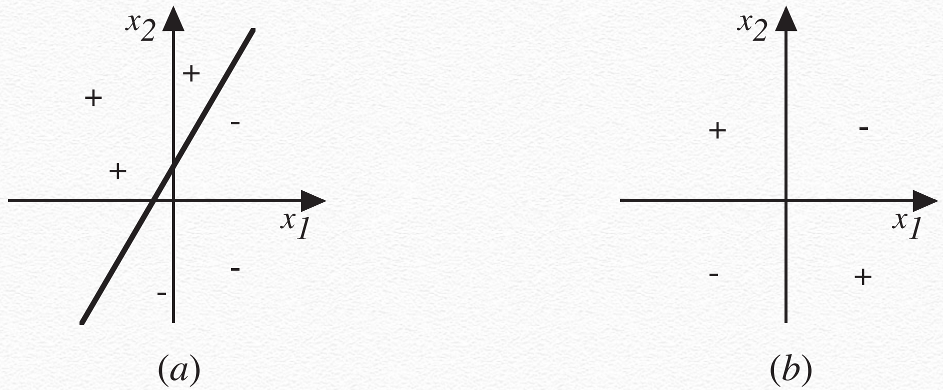

$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0, \\ -1 & \text{otherwise.} \end{cases}$$

Vector notation.

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

where $\mathbf{w} = (w_0, w_1, \ldots, w_n)^\top \in H = \mathbb{R}^{n+1}$
and $\mathbf{x} = (1, x_1, \ldots, x_n)^\top \in X$
s.t. $X$ is an $n$-dimensional subspace of $\mathbb{R}^{n+1}$.

# Decision Surface of Perceptron




Expressive power. Some useful functions can be represented:

- What weights can represent AND($x_1, x_2$)?

Some functions (e.g., not linearly separable) cannot be represented:

- We want to design neural nets to represent them!

# Perceptron Training Rule

Idea. Initialize **w** randomly, apply perceptron training rule to every training example, and iterate thru all training examples till **w** is consistent

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

for $i = 0, 1, \ldots, n$ where

- $t = c(\mathbf{x})$ is target output for training example $\langle \mathbf{x}, c(\mathbf{x}) \rangle$,
- $o = o(\mathbf{x})$ is perceptron output, and
- $\eta$ is small +ve constant (e.g., .1) called learning rate.

Guaranteed to converge if training examples are linearly separable and $\eta$ is sufficiently small

# Gradient Descent

Idea. Search $H$ to find weight vector that "best fits" the (possibly linearly non-separable) training examples

To ease understanding, consider a simpler linear unit:

$$o = \mathbf{w} \cdot \mathbf{x}$$

Learn $\mathbf{w}$ that minimizes loss function (i.e., sum of squared errors):

$$L_D(\mathbf{w}) = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

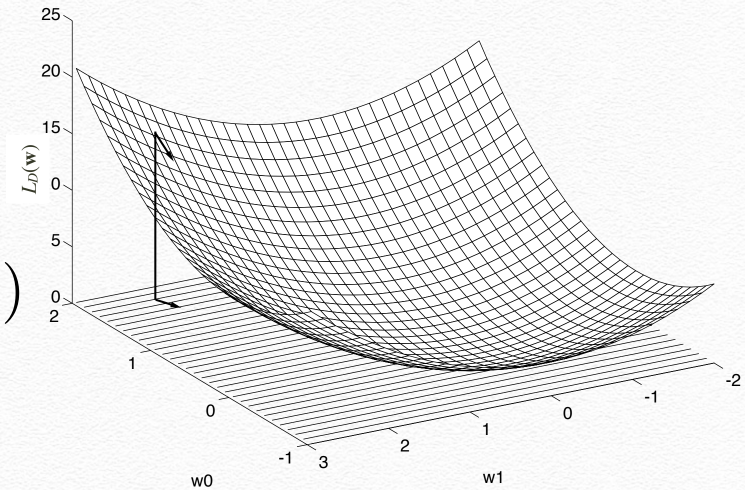
where $D$ is the set of training examples, $t_d$ and $o_d$ are, respectively, target output and output of linear unit for training example $d$

# Gradient Descent

Idea. Finds **w** that minimizes $L$ by first initializing it randomly and then repeatedly updating it in the direction of steepest descent

Gradient.

$$\nabla L_D(\mathbf{w}) = \left[\frac{\partial L_D}{\partial w_0}, \frac{\partial L_D}{\partial w_1}, \ldots, \frac{\partial L_D}{\partial w_n}\right]$$




Training rule.

$$\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{w}, \quad \Delta\mathbf{w} = -\eta\nabla L_D(\mathbf{w})$$

that is,

$$w_i \leftarrow w_i + \Delta w_i, \quad \Delta w_i = -\eta\frac{\partial L_D}{\partial w_i}$$

# Gradient Descent

$$
\begin{aligned}
\frac{\partial L_D}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d\in D}(t_d - o_d)^2 \\
&= \frac{1}{2}\sum_{d\in D}\frac{\partial}{\partial w_i}(t_d - o_d)^2 \\
&= \frac{1}{2}\sum_{d\in D}2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d) \\
&= \sum_{d\in D}(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \mathbf{w}\cdot\mathbf{x}_d) \\
\frac{\partial L_D}{\partial w_i} &= \sum_{d\in D}(t_d - o_d)(-x_{id}) \\
\Delta w_i &= \eta\sum_{d\in D}(t_d - o_d)x_{id}
\end{aligned}
$$

# Gradient-Descent

Idea. Initialize **w** randomly, apply linear unit training rule to all training examples, and repeat

Gradient-Descent$(D, \eta)$

- Initialize each $w_i$ to some small random value
- Until termination condition is met, do
  - Initialize each $\Delta w_i$ to zero.
  - For each $d \in D$, do
    - Input instance $\mathbf{x}_d$ to linear unit and compute output $o$
    - For each linear unit weight $w_i$, do

$$\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_{id}$$

  - For each linear unit weight $w_i$, do

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

# Summary so far…

Perceptron training rule is guaranteed to converge if

- Training examples are linearly separable
- Learning rate $\eta$ is sufficiently small

Linear unit training rule utilizing gradient descent is guaranteed to converge to hypothesis with min. squared error/loss

- If learning rate $\eta$ is sufficiently small
- Even when training examples are noisy and/or linearly non-separable by $H$