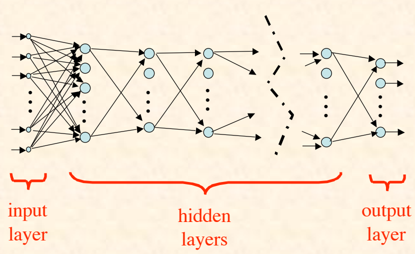
VII. Neural Networks and Learning

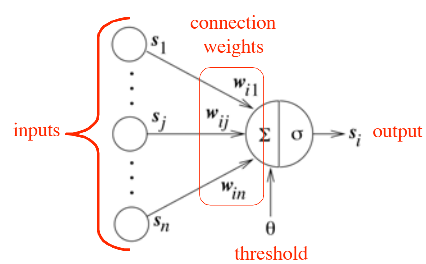
Supervised Learning

- Produce desired outputs for training inputs
- Generalize reasonably & appropriately to other inputs
- Good example: pattern recognition
- Feedforward multilayer networks

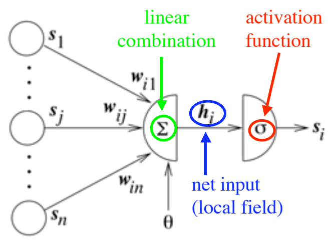
Feedforward Network



Typical Artificial Neuron



Typical Artificial Neuron



Equations

Net input:

$$h_i = \left(\sum_{j=1}^{n} w_{ij} s_j \right) - \theta$$

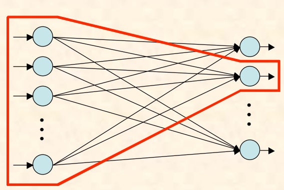
$$\mathbf{h} = \mathbf{W}\mathbf{s} - \theta$$

Neuron output:

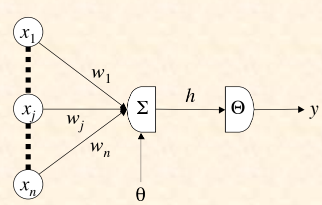
$$s_i' = \sigma(h_i)$$

$$\mathbf{s}' = \sigma(\mathbf{h})$$

Single-Layer Perceptron



Variables



Single Layer Perceptron Equations

Binary threshold activation function:

$$\sigma(h) = \Theta(h) = \begin{cases} 1, & \text{if } h > 0 \\ 0, & \text{if } h \le 0 \end{cases}$$

$$\text{Hence, } y = \begin{cases} 1, & \text{if } \sum_j w_j x_j > \theta \\ 0, & \text{otherwise} \end{cases}$$

$$= \begin{cases} 1, & \text{if } \mathbf{w} \cdot \mathbf{x} > \theta \\ 0, & \text{if } \mathbf{w} \cdot \mathbf{x} \le \theta \end{cases}$$

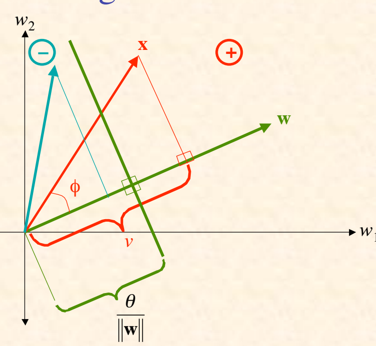
11/24/04 9

2D Weight Vector

$$\mathbf{w} \cdot \mathbf{x} = \|\mathbf{w}\| \|\mathbf{x}\| \cos\phi$$

$$\cos\phi = \frac{v}{\|\mathbf{x}\|}$$

$$\mathbf{w} \cdot \mathbf{x} = \|\mathbf{w}\| v$$



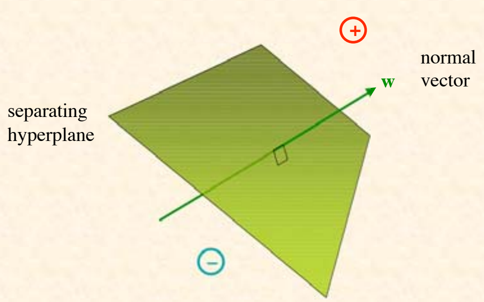
$$\mathbf{w} \cdot \mathbf{x} > \theta$$

$$\Leftrightarrow \|\mathbf{w}\| v > \theta$$

$$\Leftrightarrow v > \theta / \|\mathbf{w}\|$$

11/24/04 10

N-Dimensional Weight Vector



11/24/04 11

Goal of Perceptron Learning

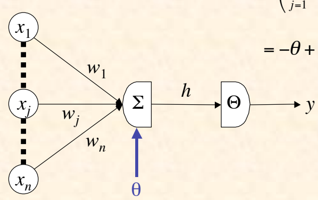
- Suppose we have training patterns $\mathbf{x}^1$, $\mathbf{x}^2$, …, $\mathbf{x}^P$ with corresponding desired outputs y^1, y^2, …, y^P
- where $\mathbf{x}^p \in \{0, 1\}^n$, $y^p \in \{0, 1\}$
- We want to find $\mathbf{w}$, θ such that $y^p = \Theta(\mathbf{w} \cdot \mathbf{x}^p - \theta)$ for $p = 1, \ldots, P$

11/24/04 12

Treating Threshold as Weight

$$h = \left(\sum_{j=1}^{n} w_j x_j \right) - \theta$$

$$= -\theta + \sum_{j=1}^{n} w_j x_j$$

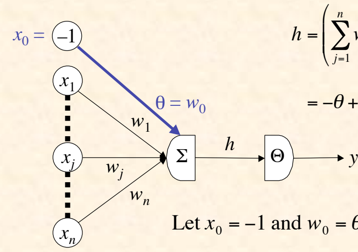


11/24/04 13

Treating Threshold as Weight

$$h = \left(\sum_{j=1}^{n} w_j x_j \right) - \theta$$

$$= -\theta + \sum_{j=1}^{n} w_j x_j$$



Let $x_0 = -1$ and $w_0 = \theta$

$$h = w_0 x_0 + \sum_{j=1}^{n} w_j x_j = \sum_{j=0}^{n} w_j x_j = \tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}}$$

11/24/04 14

Augmented Vectors

$$\tilde{\mathbf{w}} = \begin{pmatrix} \theta \\ w_1 \\ \vdots \\ w_n \end{pmatrix} \qquad \tilde{\mathbf{x}}^p = \begin{pmatrix} -1 \\ x_1^p \\ \vdots \\ x_n^p \end{pmatrix}$$

We want $y^p = \Theta\left(\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}}^p\right), \; p = 1, \ldots, P$

11/24/04 15

Reformulation as Positive Examples

We have positive ($y^p = 1$) and negative ($y^p = 0$) examples

Want $\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}}^p > 0$ for positive, $\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}}^p \le 0$ for negative

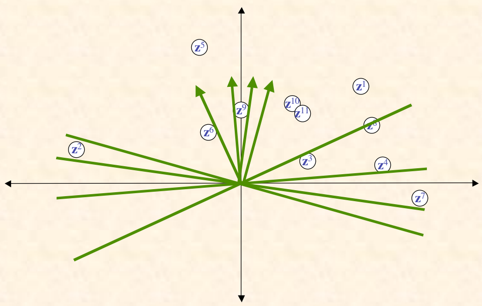
Let $\mathbf{z}^p = \tilde{\mathbf{x}}^p$ for positive, $\mathbf{z}^p = -\tilde{\mathbf{x}}^p$ for negative

Want $\tilde{\mathbf{w}} \cdot \mathbf{z}^p \ge 0$, for $p = 1, \ldots, P$

Hyperplane through origin with all $\mathbf{z}^p$ on one side

11/24/04 16

Adjustment of Weight Vector



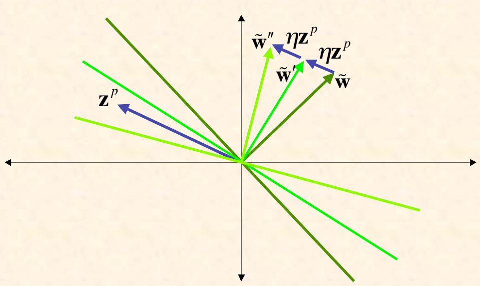
11/24/04 17

Outline of Perceptron Learning Algorithm

1. initialize weight vector randomly
2. until all patterns classified correctly, do:
   a) for $p = 1, \ldots, P$ do:
      1) if $\mathbf{z}^p$ classified correctly, do nothing
      2) else adjust weight vector to be closer to correct classification

11/24/04 18

Weight Adjustment



11/24/04 19

Improvement in Performance

If $\tilde{\mathbf{w}} \cdot \mathbf{z}^p < 0$,

$$\begin{aligned}\tilde{\mathbf{w}}' \cdot \mathbf{z}^p &= \left(\tilde{\mathbf{w}} + \eta\mathbf{z}^p\right) \cdot \mathbf{z}^p \\ &= \tilde{\mathbf{w}} \cdot \mathbf{z}^p + \eta\mathbf{z}^p \cdot \mathbf{z}^p \\ &= \tilde{\mathbf{w}} \cdot \mathbf{z}^p + \eta\left\|\mathbf{z}^p\right\|^2 \\ &> \tilde{\mathbf{w}} \cdot \mathbf{z}^p\end{aligned}$$

11/24/04 20

Perceptron Learning Theorem

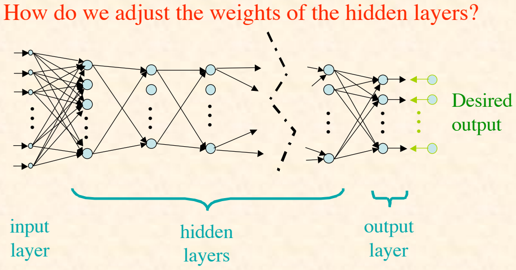
- If there is a set of weights that will solve the problem,
- then the PLA will eventually find it
- (for a sufficiently small learning rate)
- Note: only applies if positive & negative examples are linearly separable

Classification Power of Multilayer Perceptrons

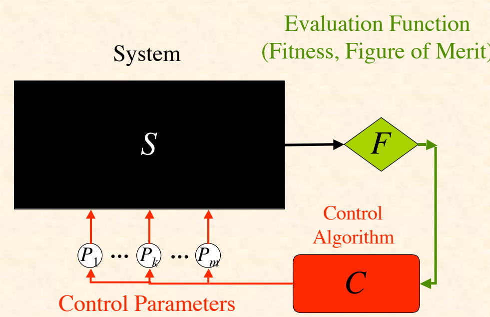
- Perceptrons can function as logic gates
- Therefore MLP can form intersections, unions, differences of linearly-separable regions
- Classes can be arbitrary *hyperpolyhedra*
- Minsky & Papert criticism of perceptrons
- No one succeeded in developing a MLP learning algorithm

Credit Assignment Problem

How do we adjust the weights of the hidden layers?



Adaptive System



Gradient

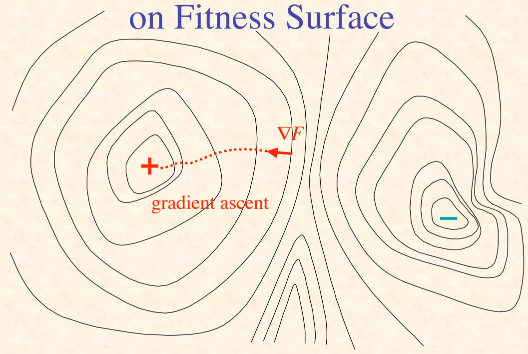
$\frac{\partial F}{\partial P_k}$ measures how F is altered by variation of P_k

$$\nabla F = \begin{pmatrix} \partial F / \partial P_1 \\ \vdots \\ \partial F / \partial P_k \\ \vdots \\ \partial F / \partial P_m \end{pmatrix}$$

∇F points in direction of maximum increase in F

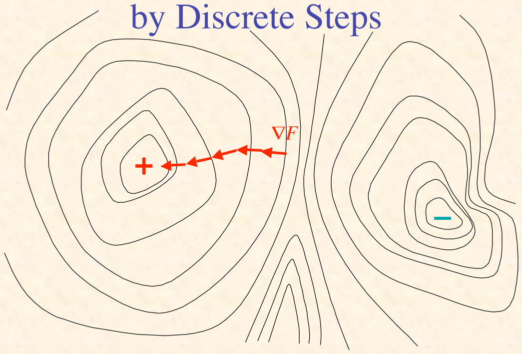
11/24/04 25

Gradient Ascent on Fitness Surface



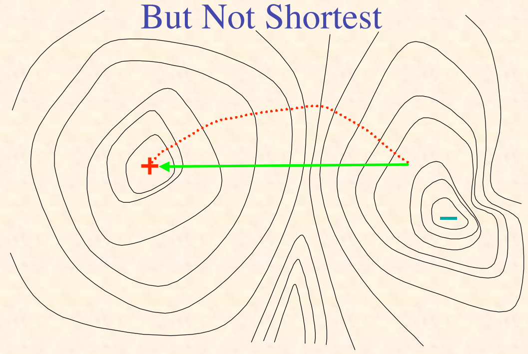
11/24/04 26

Gradient Ascent by Discrete Steps



11/24/04 27

Gradient Ascent is Local But Not Shortest



11/24/04 28

Gradient Ascent Process

$$\dot{\mathbf{P}} = \eta \nabla F(\mathbf{P})$$

Change in fitness:

$$\dot{F} = \frac{\mathrm{d}F}{\mathrm{d}t} = \sum_{k=1}^{m} \frac{\partial F}{\partial P_k} \frac{\mathrm{d}P_k}{\mathrm{d}t} = \sum_{k=1}^{m} (\nabla F)_k \dot{P}_k$$

$$\dot{F} = \nabla F \cdot \dot{\mathbf{P}}$$

$$\dot{F} = \nabla F \cdot \eta \nabla F = \eta \|\nabla F\|^2 \geq 0$$

Therefore gradient ascent increases fitness
(until reaches 0 gradient)

General Ascent in Fitness

Note that any parameter adjustment process $\mathbf{P}(t)$ will increase fitness provided:

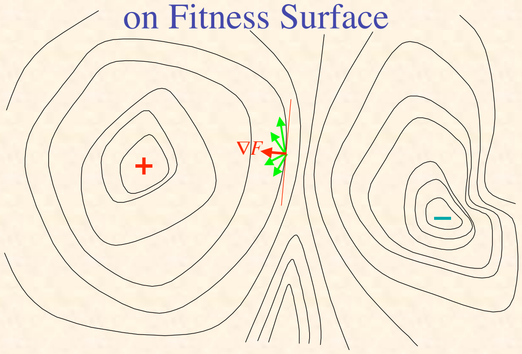
$$0 < \dot{F} = \nabla F \cdot \dot{\mathbf{P}} = \|\nabla F\| \|\dot{\mathbf{P}}\| \cos\varphi$$

where φ is angle between ∇F and $\dot{\mathbf{P}}$

Hence we need $\cos\varphi > 0$

or $|\varphi| < 90^\circ$

General Ascent on Fitness Surface



Fitness as Minimum Error

Suppose for Q different inputs we have target outputs $\mathbf{t}^1, \ldots, \mathbf{t}^Q$

Suppose for parameters $\mathbf{P}$ the corresponding actual outputs are $\mathbf{y}^1, \ldots, \mathbf{y}^Q$

Suppose $D(\mathbf{t}, \mathbf{y}) \in [0, \infty)$ measures difference between target & actual outputs

Let $E^q = D(\mathbf{t}^q, \mathbf{y}^q)$ be error on qth sample

Let $F(\mathbf{P}) = -\sum_{q=1}^{Q} E^q(\mathbf{P}) = -\sum_{q=1}^{Q} D[\mathbf{t}^q, \mathbf{y}^q(\mathbf{P})]$

Gradient of Fitness

$$\nabla F = \nabla\left(-\sum_q E^q\right) = -\sum_q \nabla E^q$$

$$\frac{\partial E^q}{\partial P_k} = \frac{\partial}{\partial P_k} D(\mathbf{t}^q, \mathbf{y}^q) = \sum_j \frac{\partial D(\mathbf{t}^q, \mathbf{y}^q)}{\partial y_j^q} \frac{\partial y_j^q}{\partial P_k}$$

$$= \frac{\mathrm{d}\, D(\mathbf{t}^q, \mathbf{y}^q)}{\mathrm{d}\, \mathbf{y}^q} \cdot \frac{\partial \mathbf{y}^q}{\partial P_k}$$

$$= \nabla_{\mathbf{y}^q} D(\mathbf{t}^q, \mathbf{y}^q) \cdot \partial \mathbf{y}^q / \partial P_k$$

11/24/04 33

Jacobian Matrix

Define Jacobian matrix $\mathbf{J}^q = \begin{pmatrix} \partial y_1^q / \partial P_1 & \cdots & \partial y_1^q / \partial P_m \\ \vdots & \ddots & \vdots \\ \partial y_n^q / \partial P_1 & \cdots & \partial y_n^q / \partial P_m \end{pmatrix}$

Note $\mathbf{J}^q \in \Re^{n \times m}$ and $\nabla D(\mathbf{t}^q, \mathbf{y}^q) \in \Re^{n \times 1}$

Since $\left(\nabla E^q\right)_k = \frac{\partial E^q}{\partial P_k} = \sum_j \frac{\partial y_j^q}{\partial P_k} \frac{\partial D(\mathbf{t}^q, \mathbf{y}^q)}{\partial y_j^q}$,

$\therefore \nabla E^q = \left(\mathbf{J}^q\right)^{\mathrm{T}} \nabla D(\mathbf{t}^q, \mathbf{y}^q)$

11/24/04 34

Derivative of Squared Euclidean Distance

Suppose $D(\mathbf{t}, \mathbf{y}) = \|\mathbf{t} - \mathbf{y}\|^2 = \sum_i (t_i - y_i)^2$

$$\frac{\partial D(\mathbf{t} - \mathbf{y})}{\partial y_j} = \frac{\partial}{\partial y_j} \sum_i (t_i - y_i)^2 = \sum_i \frac{\partial (t_i - y_i)^2}{\partial y_j}$$

$$= \frac{\mathrm{d}(t_i - y_i)^2}{\mathrm{d}\, y_i} = -2(t_i - y_i)$$

$$\therefore \frac{\mathrm{d}\, D(\mathbf{t}, \mathbf{y})}{\mathrm{d}\, \mathbf{y}} = 2(\mathbf{y} - \mathbf{t})$$

11/24/04 35

Gradient of Error on q^{th} Input

$$\frac{\partial E^q}{\partial P_k} = \frac{\mathrm{d}\, D(\mathbf{t}^q, \mathbf{y}^q)}{\mathrm{d}\, \mathbf{y}^q} \cdot \frac{\partial \mathbf{y}^q}{\partial P_k}$$

$$= 2(\mathbf{y}^q - \mathbf{t}^q) \cdot \frac{\partial \mathbf{y}^q}{\partial P_k}$$

$$= 2 \sum_j (y_j^q - t_j^q) \frac{\partial y_j^q}{\partial P_k}$$

11/24/04 36

Recap

$$\dot{\mathbf{P}} = \eta \sum_q \left(\mathbf{J}^q\right)^{\mathrm{T}} \left(\mathbf{t}^q - \mathbf{y}^q\right)$$

To know how to decrease the differences between actual & desired outputs,

we need to know elements of Jacobian, $\partial y_j^q / \partial P_k$,

which says how *j*th output varies with *k*th parameter (given the *q*th input)

The Jacobian depends on the specific form of the system, in this case, a feedforward neural network

11/24/04 37