IV. Neural Networks and Learning

3/23/16   1

A.
Artificial Neural Net Learning

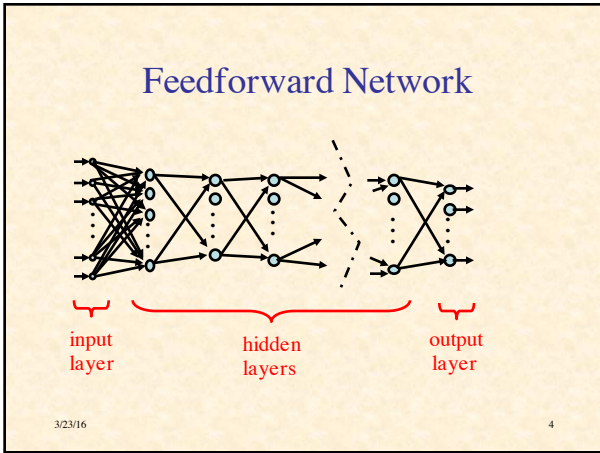3/23/16   2

Supervised Learning

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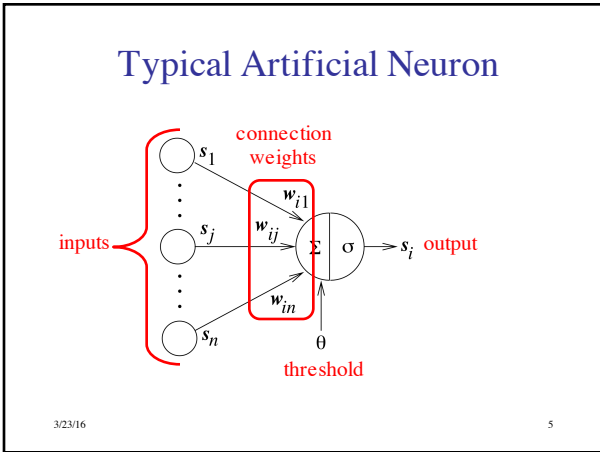- Feedforward multilayer networks

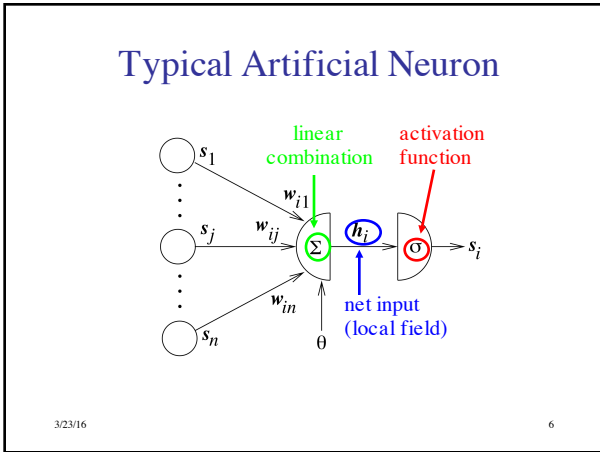3/23/16   3

Feedforward Network

input layer   hidden layers   output layer

3/23/16   4

Typical Artificial Neuron

connection weights

inputs

s_1, s_j, s_n

w_{i1}, w_{ij}, w_{in}

Σ σ s_i output

θ

threshold

3/23/16   5

Typical Artificial Neuron

linear combination

activation function

s_1, s_j, s_n

w_{i1}, w_{ij}, w_{in}

Σ h_i σ s_i

net input (local field)

θ

3/23/16   6
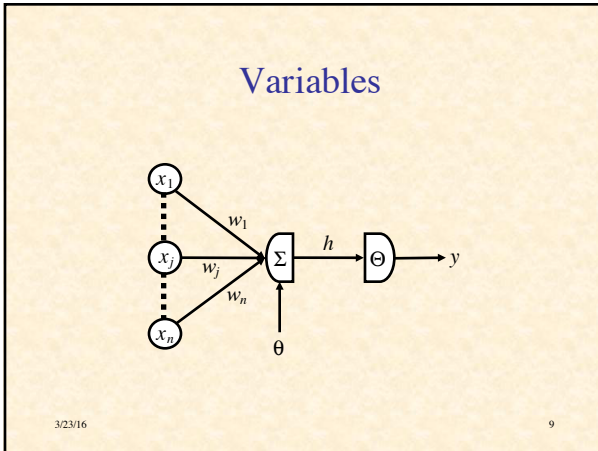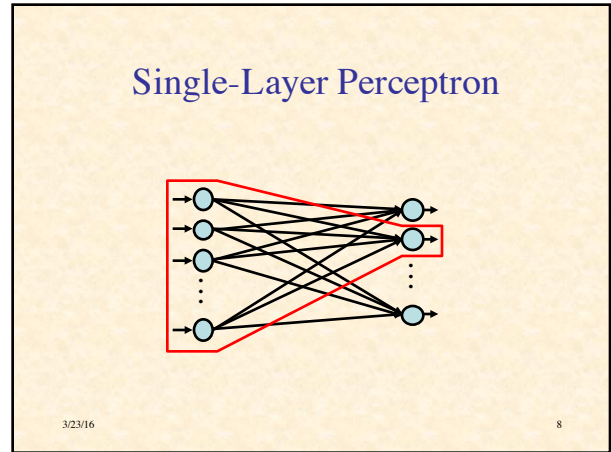
1

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

3/23/16   7

Single-Layer Perceptron

3/23/16   8

Variables

3/23/16   9
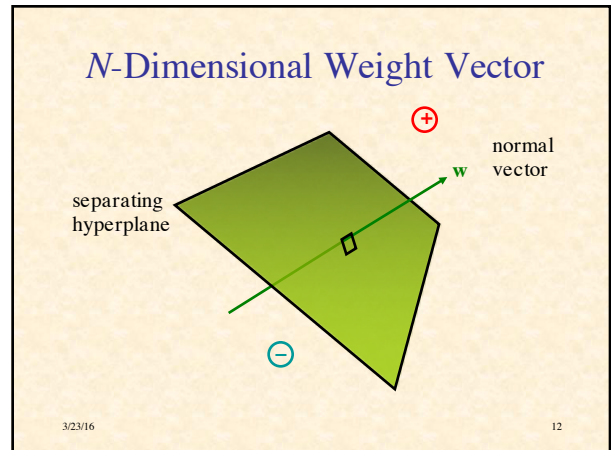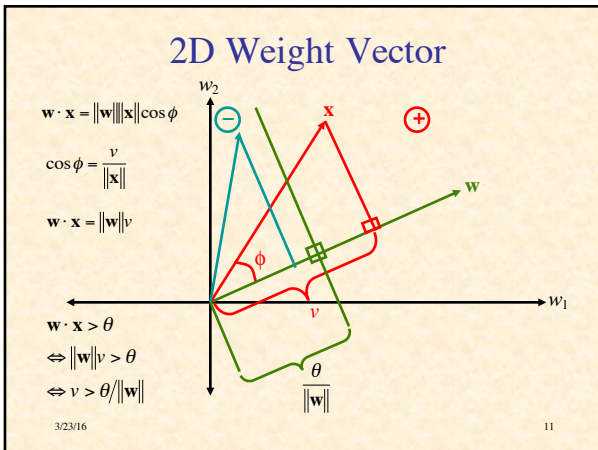
Single Layer Perceptron Equations

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

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

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

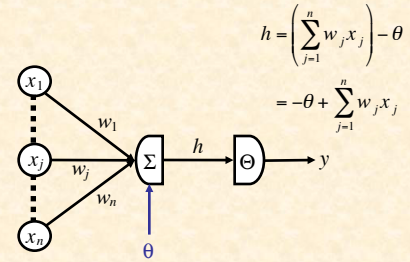3/23/16   10

2D Weight Vector

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

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

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

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

3/23/16   11

N-Dimensional Weight Vector

3/23/16   12

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 $\mathbf{w}, \theta$ such that
  $y^p = \Theta(\mathbf{w} \cdot \mathbf{x}^p - \theta)$ for $p = 1, …, P$
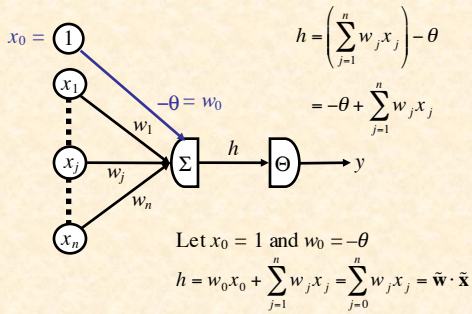
3/23/16　　　　13

Treating Threshold as Weight

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

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

3/23/16　　　　14

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 $w_0 = -\theta$

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

3/23/16　　　　15

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 $y^p = \Theta\left(\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}}^p\right),\ p = 1,…,P$
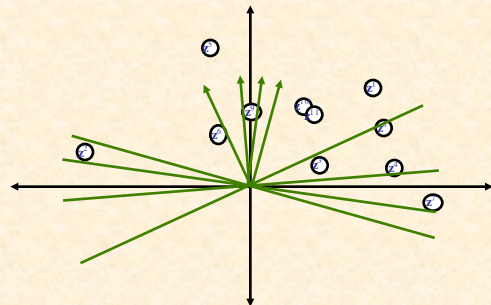
3/23/16　　　　16

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, $\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}}^p \leq 0$ for negative

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

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

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

3/23/16　　　　17
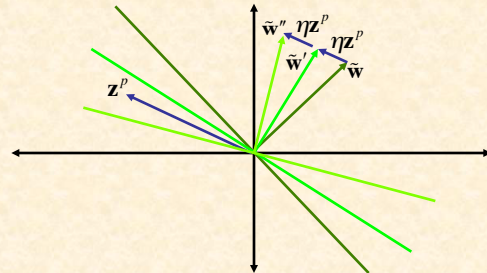
Adjustment of Weight Vector

3/23/16　　　　18

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

3/23/16   19

Weight Adjustment

3/23/16   20

Improvement in Performance

$$\tilde{\mathbf{w}}' \cdot \mathbf{z}^p = \left(\tilde{\mathbf{w}} + \eta \mathbf{z}^p \right) \cdot \mathbf{z}^p$$

$$= \tilde{\mathbf{w}} \cdot \mathbf{z}^p + \eta \mathbf{z}^p \cdot \mathbf{z}^p$$

$$= \tilde{\mathbf{w}} \cdot \mathbf{z}^p + \eta \left\| \mathbf{z}^p \right\|^2$$

$$> \tilde{\mathbf{w}} \cdot \mathbf{z}^p$$

3/23/16   21

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

3/23/16   22

NetLogo Simulation of Perceptron Learning

Run Perceptron-Geometry.nlogo

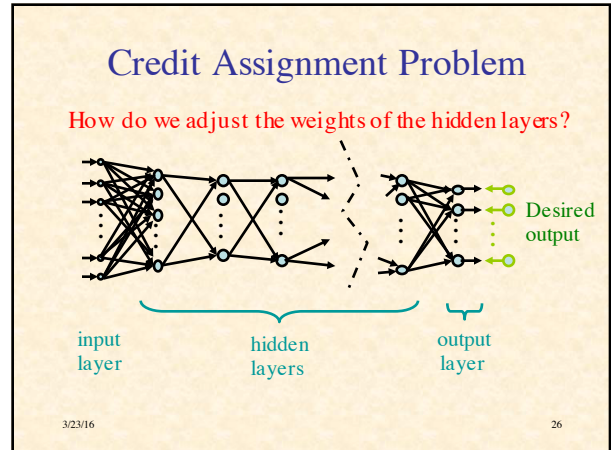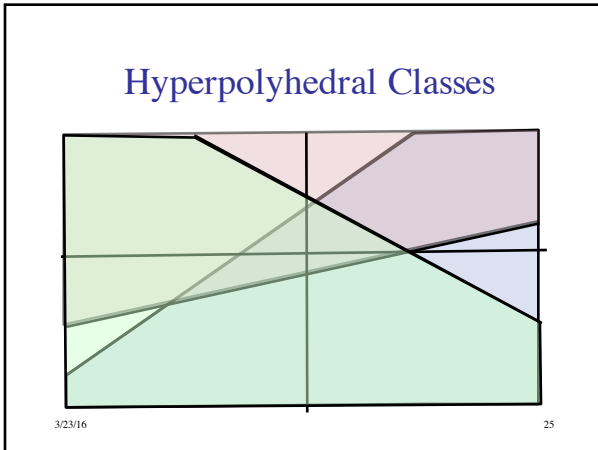3/23/16   23

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
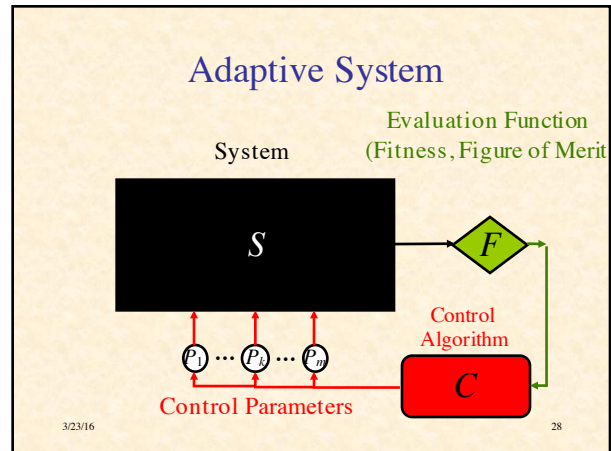
3/23/16   24

Hyperpolyhedral Classes

3/23/16   25

Credit Assignment Problem

How do we adjust the weights of the hidden layers?

Desired output

input layer   hidden layers   output layer

3/23/16   26

NetLogo Demonstration of Back-Propagation Learning

Run Artificial Neural Net.nlogo

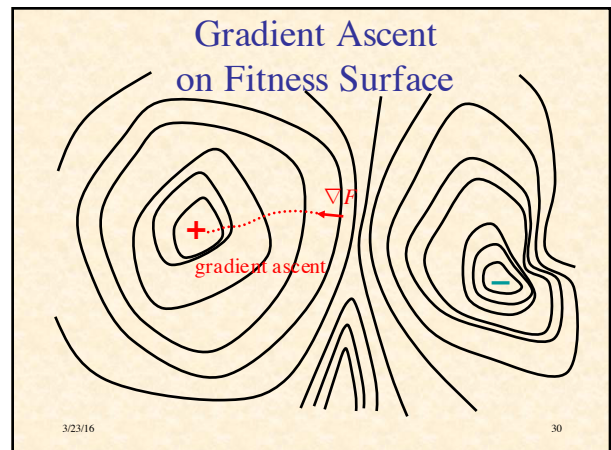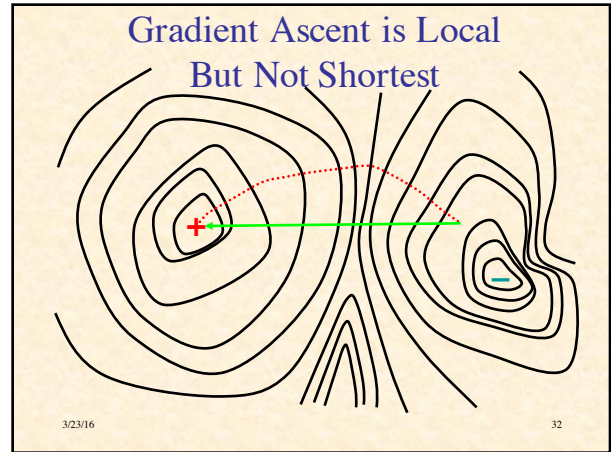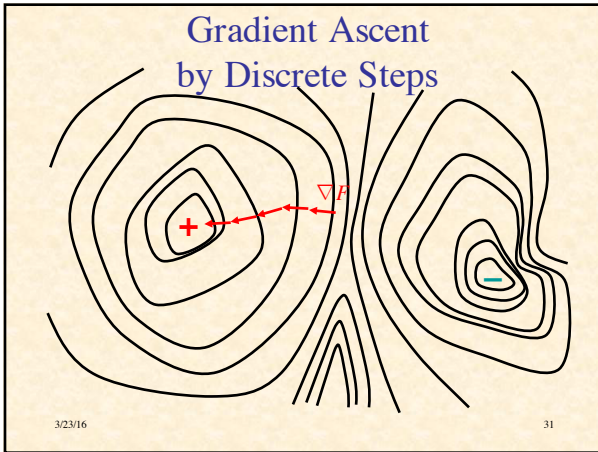3/23/16   27

Adaptive System

Evaluation Function (Fitness, Figure of Merit)

System

S

F

Control Algorithm

P_1 ⋯ P_k ⋯ P_m

C

Control Parameters

3/23/16   28

Gradient

$\dfrac{\partial F}{\partial P_k}$ 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 local increase in F

3/23/16   29

Gradient Ascent on Fitness Surface

∇F

+

gradient ascent

−

3/23/16   30

5

Gradient Ascent
by Discrete Steps

Gradient Ascent is Local
But Not Shortest

Gradient Ascent Process

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

Change in fitness :

$$\dot{F} = \frac{dF}{dt} = \sum_{k=1}^{m} \frac{\partial F}{\partial P_k} \frac{dP_k}{dt} = \sum_{k=1}^{m} (\nabla F)_k \dot{P}_k$$

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

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

Therefore gradient ascent increases fitness
(until reaches 0 gradient)
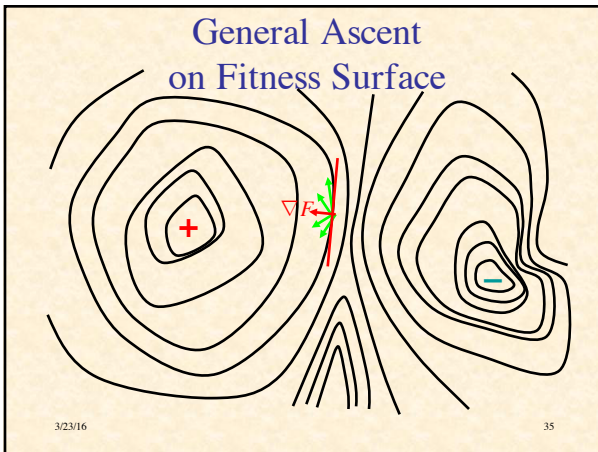
General Ascent in Fitness

Note that any adaptive 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 $|\varphi| < 90°$

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

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

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

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

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

3/23/16   37

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 $\nabla D\left(\mathbf{t}^q, \mathbf{y}^q\right) \in \Re^{n \times 1}$

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

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

3/23/16   38

Derivative of Squared Euclidean Distance

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

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

$$= \frac{d\left(t_j - y_j\right)^2}{d\, y_j} = -2\left(t_j - y_j\right)$$

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

3/23/16   39

Gradient of Error on q^{th} Input

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

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

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

$$\nabla E^q = 2\left(\mathbf{J}^q\right)^{\mathrm{T}} \left(\mathbf{y}^q - \mathbf{t}^q\right)$$

3/23/16   40

Recap

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

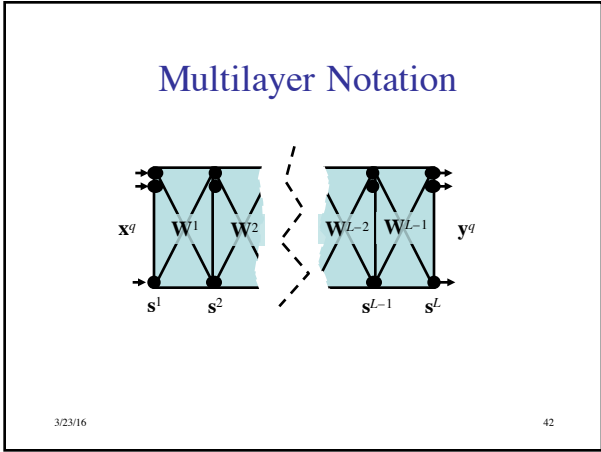To know how to decrease the differences between actual & desired outputs,

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

which says how jth output varies with kth parameter (given the qth input)

The Jacobian depends on the specific form of the system, in this case, a feedforward neural network
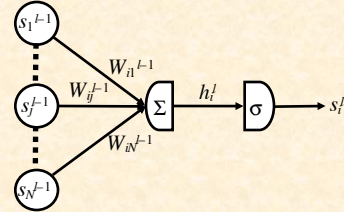
3/23/16   41

Multilayer Notation

3/23/16   42

Notation

- L layers of neurons labeled $1, \ldots, L$
- N_l neurons in layer l
- $\mathbf{s}^l$ = vector of outputs from neurons in layer l
- input layer $\mathbf{s}^1 = \mathbf{x}^q$ (the input pattern)
- output layer $\mathbf{s}^L = \mathbf{y}^q$ (the actual output)
- $\mathbf{W}^l$ = weights between layers l and $l+1$
- Problem: find out how outputs y_i^q vary with weights W_{jk}^l ($l = 1, \ldots, L-1$)

3/23/16   43

Typical Neuron

3/23/16   44

Error Back-Propagation

We will compute $\dfrac{\partial E^q}{\partial W_{ij}^l}$ starting with last layer ($l = L-1$)

and working back to earlier layers ($l = L-2, \ldots, 1$)

3/23/16   45

Delta Values

Convenient to break derivatives by chain rule :

$$\frac{\partial E^q}{\partial W_{ij}^{l-1}} = \frac{\partial E^q}{\partial h_i^l} \frac{\partial h_i^l}{\partial W_{ij}^{l-1}}$$
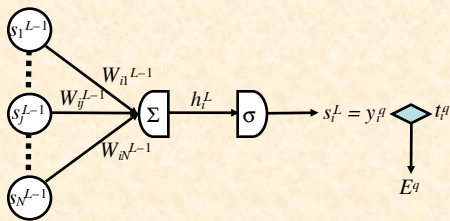
Let $\delta_i^l = \dfrac{\partial E^q}{\partial h_i^l}$

So $\dfrac{\partial E^q}{\partial W_{ij}^{l-1}} = \delta_i^l \dfrac{\partial h_i^l}{\partial W_{ij}^{l-1}}$

3/23/16   46

Output-Layer Neuron

3/23/16   47

Output-Layer Derivatives (1)

$$\delta_i^L = \frac{\partial E^q}{\partial h_i^L} = \frac{\partial}{\partial h_i^L} \sum_k \left(s_k^L - t_k^q \right)^2$$

$$= \frac{d\left(s_i^L - t_i^q \right)^2}{d h_i^L} = 2\left(s_i^L - t_i^q \right) \frac{d s_i^L}{d h_i^L}$$

$$= 2\left(s_i^L - t_i^q \right) \sigma'\left(h_i^L \right)$$

3/23/16   48

8

Output-Layer Derivatives (2)

$$\frac{\partial h_i^L}{\partial W_{ij}^{L-1}} = \frac{\partial}{\partial W_{ij}^{L-1}} \sum_k W_{ik}^{L-1} s_k^{L-1} = s_j^{L-1}$$
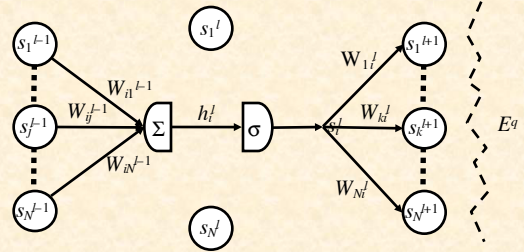
$$\therefore \frac{\partial E^q}{\partial W_{ij}^{L-1}} = \delta_i^L s_j^{L-1}$$

$$\text{where } \delta_i^L = 2\left(s_i^L - t_i^q\right)\sigma'\!\left(h_i^L\right)$$

Hidden-Layer Neuron

Hidden-Layer Derivatives (1)

$$\text{Recall } \frac{\partial E^q}{\partial W_{ij}^{l-1}} = \delta_i^l \frac{\partial h_i^l}{\partial W_{ij}^{l-1}}$$

$$\delta_i^l = \frac{\partial E^q}{\partial h_i^l} = \sum_k \frac{\partial E^q}{\partial h_k^{l+1}} \frac{\partial h_k^{l+1}}{\partial h_i^l} = \sum_k \delta_k^{l+1} \frac{\partial h_k^{l+1}}{\partial h_i^l}$$

$$\frac{\partial h_k^{l+1}}{\partial h_i^l} = \frac{\partial \sum_m W_{km}^l s_m^l}{\partial h_i^l} = \frac{\partial W_{ki}^l s_i^l}{\partial h_i^l} = W_{ki}^l \frac{d\sigma\!\left(h_i^l\right)}{d h_i^l} = W_{ki}^l \sigma'\!\left(h_i^l\right)$$

$$\therefore \delta_i^l = \sum_k \delta_k^{l+1} W_{ki}^l \sigma'\!\left(h_i^l\right) = \sigma'\!\left(h_i^l\right) \sum_k \delta_k^{l+1} W_{ki}^l$$

Hidden-Layer Derivatives (2)

$$\frac{\partial h_i^l}{\partial W_{ij}^{l-1}} = \frac{\partial}{\partial W_{ij}^{l-1}} \sum_k W_{ik}^{l-1} s_k^{l-1} = \frac{d W_{ij}^{l-1} s_j^{l-1}}{d W_{ij}^{l-1}} = s_j^{l-1}$$

$$\therefore \frac{\partial E^q}{\partial W_{ij}^{l-1}} = \delta_i^l s_j^{l-1}$$

$$\text{where } \delta_i^l = \sigma'\!\left(h_i^l\right) \sum_k \delta_k^{l+1} W_{ki}^l$$

Derivative of Sigmoid

$$\text{Suppose } s = \sigma(h) = \frac{1}{1+\exp(-\alpha h)} \text{ (logistic sigmoid)}$$

$$D_h\, s = D_h\left[1+\exp(-\alpha h)\right]^{-1} = -\left[1+\exp(-\alpha h)\right]^{-2} D_h\left(1+e^{-\alpha h}\right)$$

$$= -\left(1+e^{-\alpha h}\right)^{-2}\left(-\alpha e^{-\alpha h}\right) = \alpha \frac{e^{-\alpha h}}{\left(1+e^{-\alpha h}\right)^2}$$

$$= \alpha \frac{1}{1+e^{-\alpha h}} \frac{e^{-\alpha h}}{1+e^{-\alpha h}} = \alpha s\left(\frac{1+e^{-\alpha h}}{1+e^{-\alpha h}} - \frac{1}{1+e^{-\alpha h}}\right)$$

$$= \alpha s(1-s)$$

Summary of Back-Propagation Algorithm

$$\text{Output layer}: \delta_i^L = 2\alpha s_i^L\left(1-s_i^L\right)\left(s_i^L - t_i^q\right)$$
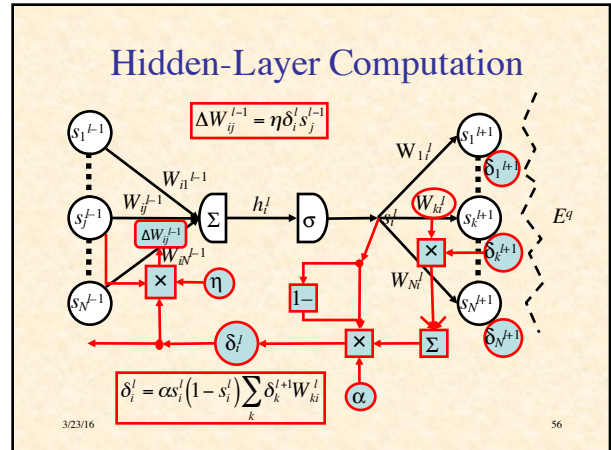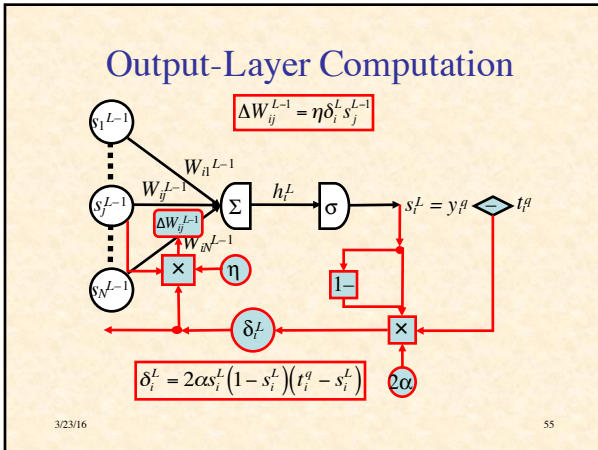
$$\frac{\partial E^q}{\partial W_{ij}^{L-1}} = \delta_i^L s_j^{L-1}$$

$$\text{Hidden layers}: \delta_i^l = \alpha s_i^l\left(1-s_i^l\right)\sum_k \delta_k^{l+1} W_{ki}^l$$
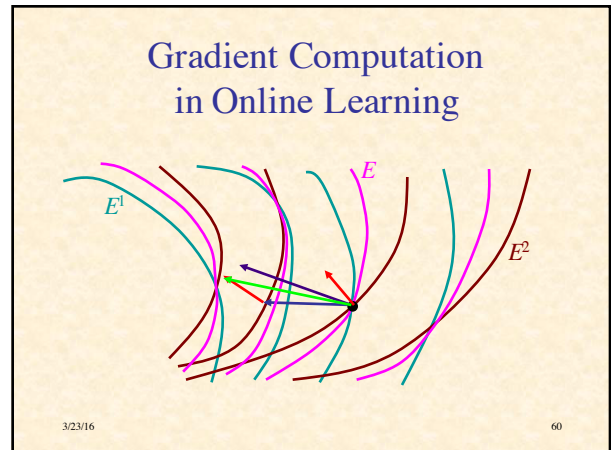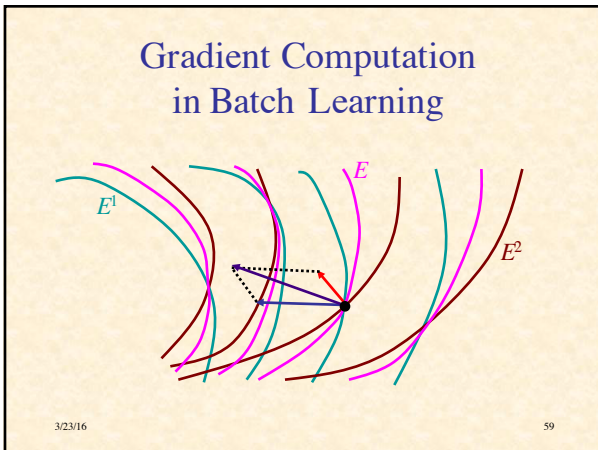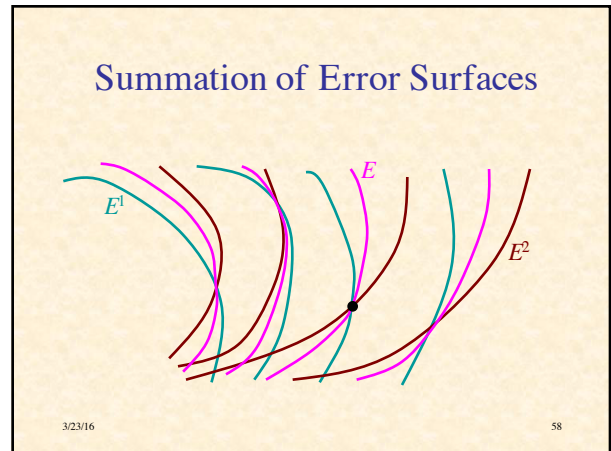
$$\frac{\partial E^q}{\partial W_{ij}^{l-1}} = \delta_i^l s_j^{l-1}$$
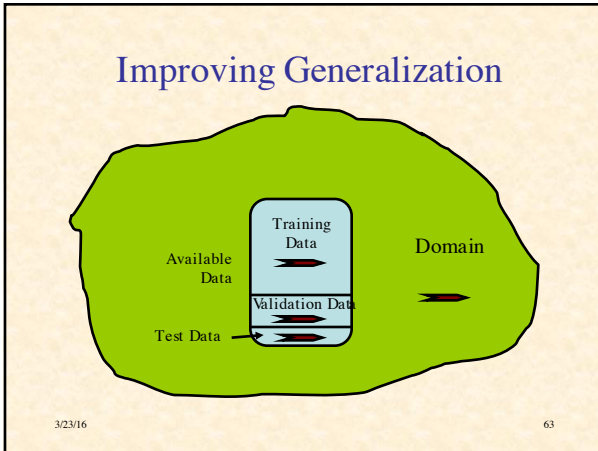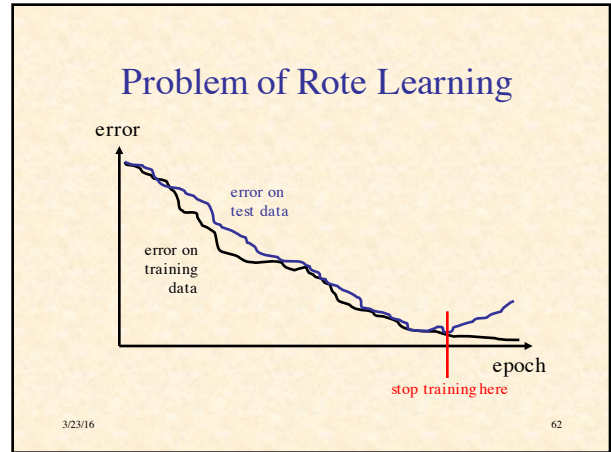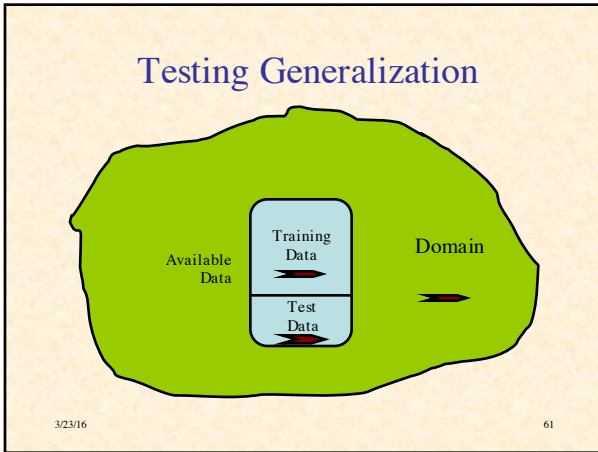
Output-Layer Computation

$$\Delta W_{ij}^{L-1} = \eta \delta_i^L s_j^{L-1}$$

$$\delta_i^L = 2\alpha s_i^L \left(1 - s_i^L\right)\left(t_i^q - s_i^L\right)$$

3/23/16   55

Hidden-Layer Computation

$$\Delta W_{ij}^{l-1} = \eta \delta_i^l s_j^{l-1}$$

$$\delta_i^l = \alpha s_i^l \left(1 - s_i^l\right) \sum_k \delta_k^{l+1} W_{ki}^l$$

3/23/16   56

Training Procedures

- Batch Learning
  - on each *epoch* (pass through all the training pairs),
  - weight changes for all patterns accumulated
  - weight matrices updated at end of epoch
  - accurate computation of gradient
- Online Learning
  - weight are updated after back-prop of each training pair
  - usually randomize order for each epoch
  - approximation of gradient
- Doesn't make much difference

3/23/16   57

Summation of Error Surfaces

3/23/16   58

Gradient Computation in Batch Learning

3/23/16   59

Gradient Computation in Online Learning

3/23/16   60

10

Testing Generalization

Available Data

Training Data

Test Data

Domain

Problem of Rote Learning

error

error on test data

error on training data

epoch

stop training here

Improving Generalization

Available Data

Training Data

Validation Data

Test Data

Domain

A Few Random Tips

- Too few neurons and the ANN may not be able to decrease the error enough
- Too many neurons can lead to rote learning
- Preprocess data to:
  - standardize
  - eliminate irrelevant information
  - capture invariances
  - keep relevant information
- If stuck in local min., restart with different random weights

Run Example BP Learning

Beyond Back-Propagation

- Adaptive Learning Rate
- Adaptive Architecture
  - Add/delete hidden neurons
  - Add/delete hidden layers
- Radial Basis Function Networks
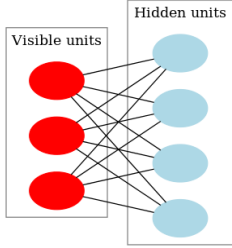- Recurrent BP
- Etc., etc., etc.…

Deep Belief Networks

- Inspired by hierarchical representations in mammalian sensory systems
- Use "deep" (multilayer) feed-forward nets
- Layers self-organize to represent input at progressively more abstract, task-relevant levels
- Supervised training (e.g., BP) can be used to tune network performance.
- Each layer is a *Restricted Boltzmann Machine*

3/23/16   67

Restricted Boltzmann Machine

- Goal: hidden units become model of input domain
- Should capture statistics of input
- Evaluate by testing its ability to reproduce input statistics
- Change weights to decrease difference

Visible units   Hidden units

(fig. from wikipedia)   68

3/23/16

Unsupervised RBM Learning

- Stochastic binary units
- Assume bias units
  $$x_0 = y_0 = 1$$
- Set y_i with probability
  $$\sigma\left(\sum_j W_{ij} x_j\right)$$
- Set x_j' with probability
  $$\sigma\left(\sum_i W_{ij} y_i\right)$$

- Set y_i' with probability
  $$\sigma\left(\sum_j W_{ij} x_j'\right)$$
- After several cycles of sampling, update weights based on statistics:
  $$\Delta W_{ij} = \eta\left(\langle y_i x_j \rangle - \langle y_i' x_j' \rangle\right)$$

3/23/16   69

Training a DBN Network

- Present inputs and do RBM learning with first hidden layer to develop model
- When converged, do RBM learning between first and second hidden layers to develop higher-level model
- Continue until all weight layers trained
- May further train with BP or other supervised learning algorithms

3/23/16   70

What is the Power of Artificial Neural Networks?

- With respect to Turing machines?
- As function approximators?

3/23/16   71

Can ANNs Exceed the "Turing Limit"?

- There are many results, which depend sensitively on assumptions; for example:
- Finite NNs with real-valued weights have super-Turing power (Siegelmann & Sontag '94)
- Recurrent nets with Gaussian noise have sub-Turing power (Maass & Sontag '99)
- Finite recurrent nets with real weights can recognize all languages, and thus are super-Turing (Siegelmann '99)
- Stochastic nets with rational weights have super-Turing power (but only P/POLY, BPP/log*) (Siegelmann '99)
- But computing classes of functions is not a very relevant way to evaluate the capabilities of neural computation

3/23/16   72

A Universal Approximation Theorem

Suppose f is a continuous function on $[0,1]^n$

Suppose σ is a nonconstant, bounded, monotone increasing real function on $\Re$.

For any $\varepsilon > 0$, there is an m such that

$\exists \mathbf{a} \in \Re^m, \mathbf{b} \in \Re^n, \mathbf{W} \in \Re^{m \times n}$ such that if

$$F(x_1, \ldots, x_n) = \sum_{i=1}^{m} a_i \sigma \left(\sum_{j=1}^{n} W_{ij} x_j + b_j \right)$$

$$\left[\text{i.e., } F(\mathbf{x}) = \mathbf{a} \cdot \sigma(\mathbf{Wx} + \mathbf{b}) \right]$$

then $\left| F(\mathbf{x}) - f(\mathbf{x}) \right| < \varepsilon$ for all $\mathbf{x} \in [0,1]^n$
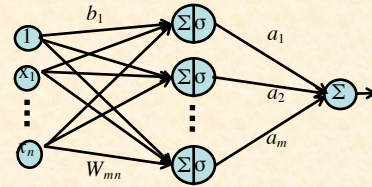
3/23/16

(see, e.g., Haykin, *N.Nets* 2/e, 208–9)   73

One Hidden Layer is Sufficient

- <u>Conclusion</u>: One hidden layer is sufficient to approximate any continuous function arbitrarily closely

3/23/16   74

The Golden Rule of Neural Nets

Neural Networks are the
second-best way
to do *everything*!

3/23/16   75