CSE 5526: Introduction to Neural Networks

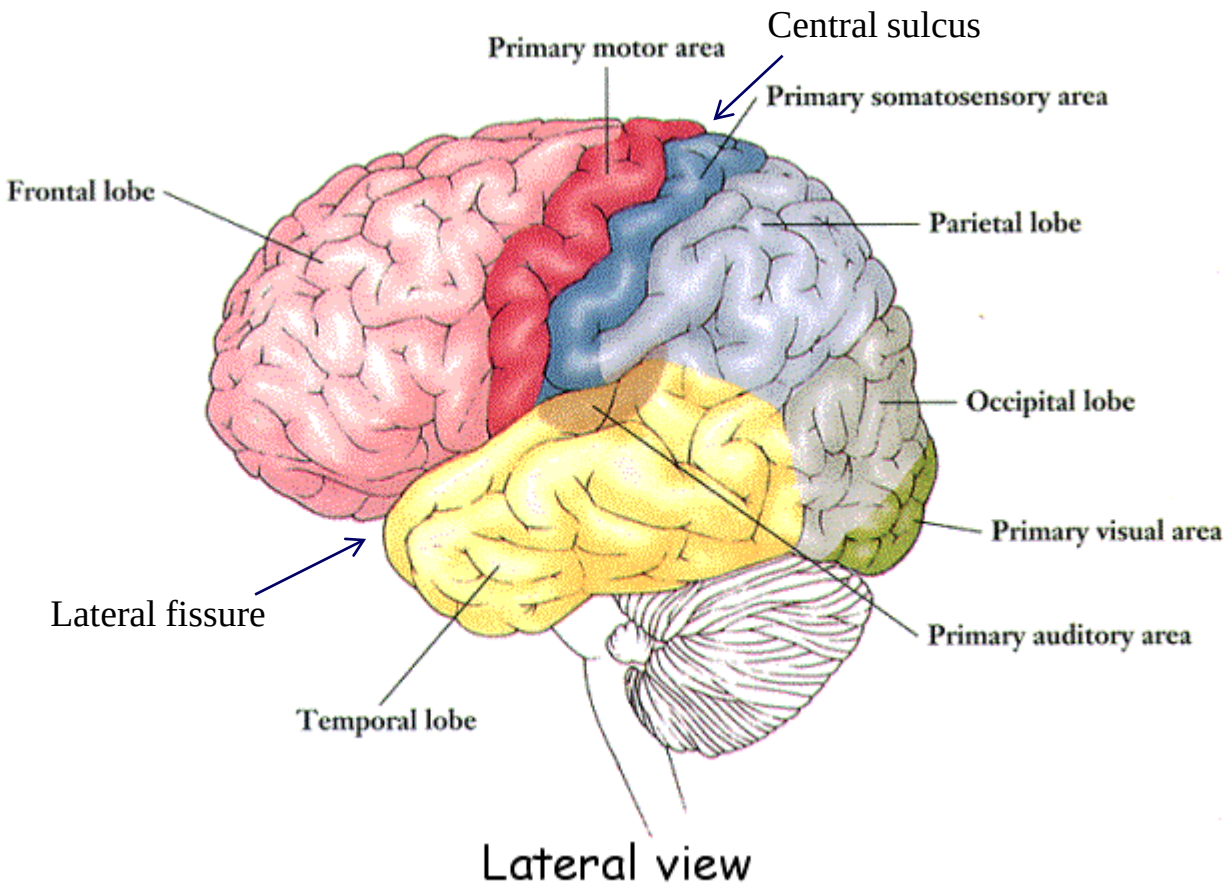
Instructor: Jeremy Morris

(course designed by Dr. DeLiang Wang)

What is this course about?

- AI (artificial intelligence) in the broad sense, in particular learning
- The human brain and its amazing ability, e.g. vision

Human brain


Central sulcus
Primary motor area
Primary somatosensory area
Frontal lobe
Parietal lobe
Occipital lobe
Primary visual area
Lateral fissure
Primary auditory area
Temporal lobe
Lateral view

Brain versus computer

Brain	Computer
•	•
•	•
•	•

Brain-like computation – Neural networks (NN or ANN) – Neural computation

- Discuss syllabus

A single neuron

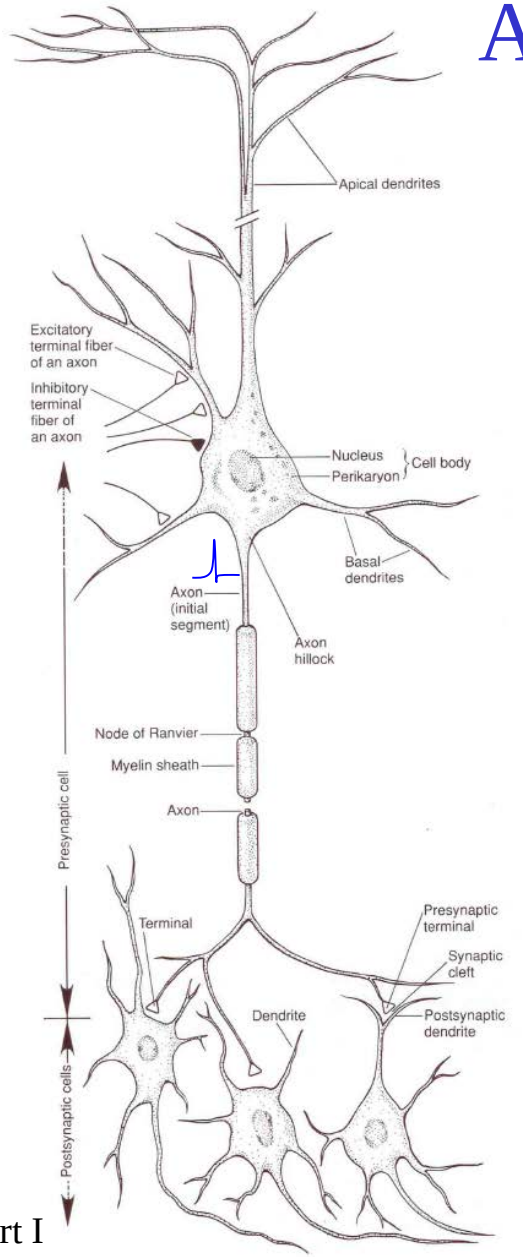


FIGURE 2–1
The main features of a typical vertebrate neuron. This neuron is drawn to illustrate its various regions and its points of contact with other nerve cells. The cell body contains the nucleus and perikaryon. The cell body gives rise to two types of processes—dendrites (both apical and basal) and axons. The axon is the transmitting element of the neuron. Axons vary greatly in length, with some extending more than 1 meter. Most axons in the central nervous system are very thin (between 0.2 and 20 μm) compared with the diameter of the cell body (up to 50 μm or more in diameter). The axon hillock, the region of the cell body where the axon emerges, is where the action potential is initiated. Many axons are insulated by a fatty myelin sheath, which is interrupted at regular intervals by regions known as the nodes of Ranvier. Branches of the axon of one neuron (the presynaptic neuron) form synaptic connections with the dendrites or cell body of another neuron (the postsynaptic cell). The branches of the axon may form synapses with as many as 1000 other neurons.

A single neuron

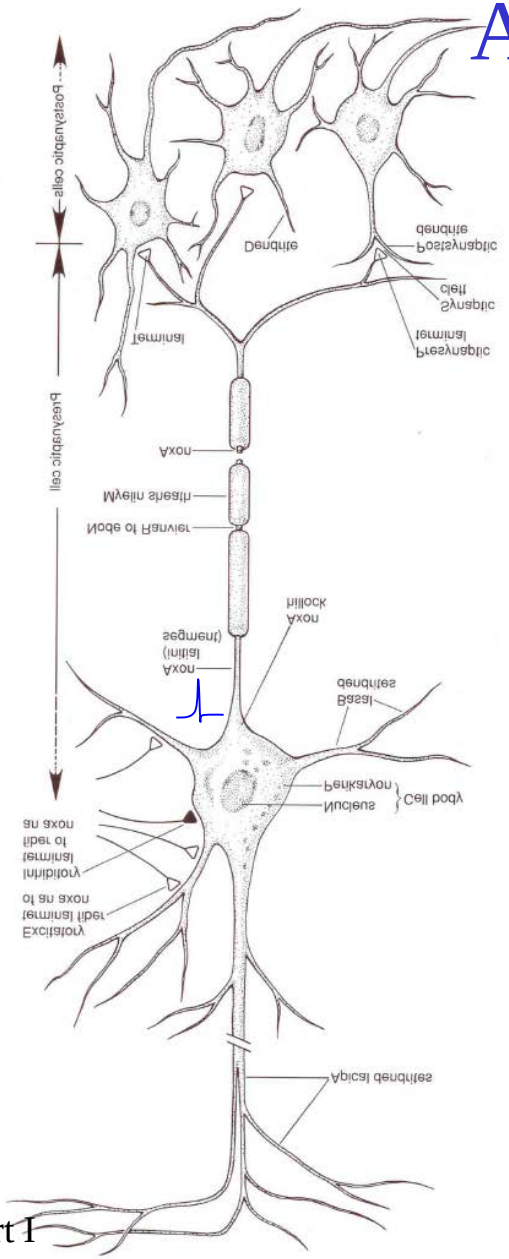
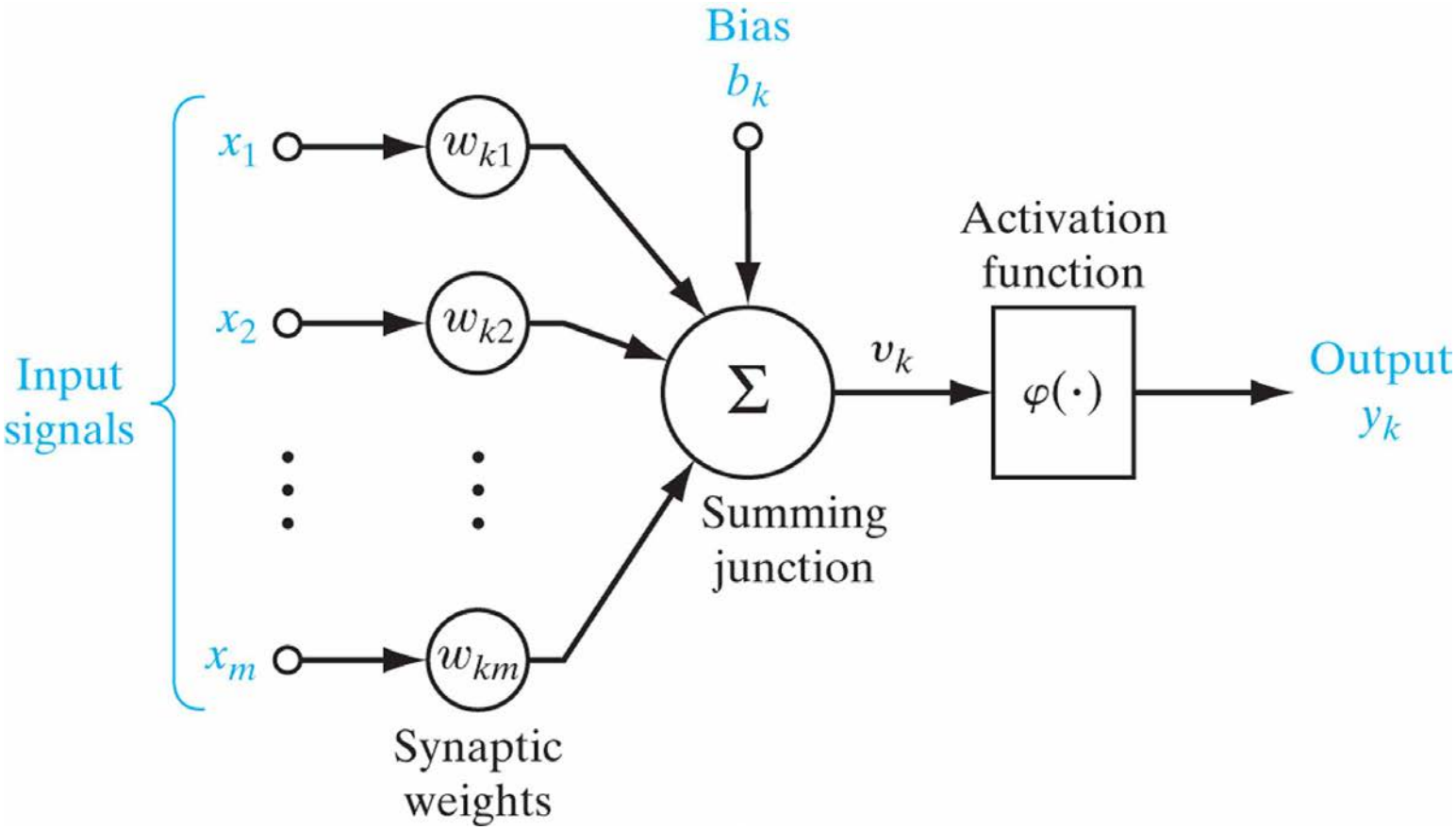


FIGURE 2–1
The main features of a typical vertebrate neuron. This neuron is drawn to illustrate its various regions and its points of contact with other nerve cells. The cell body contains the nucleus and perikaryon. The cell body gives rise to two types of processes—dendrites (both apical and basal) and axons. The axon is the transmitting element of the neuron. Axons vary greatly in length, with some extending more than 1 meter. Most axons in the central nervous system are very thin (between 0.2 and 20 μm) compared with the diameter of the cell body (up to 50 μm or more in diameter). The axon hillock, the region of the cell body where the axon emerges, is where the action potential is initiated. Many axons are insulated by a fatty myelin sheath, which is interrupted at regular intervals by regions known as the nodes of Ranvier. Branches of the axon of one neuron (the presynaptic neuron) form synaptic connections with the dendrites or cell body of another neuron (the postsynaptic cell). The branches of the axon may form synapses with as many as 1000 other neurons.

Real neurons, real synapses

- **Properties**
  - Action potential (impulse) generation
  - Impulse propagation
  - Synaptic transmission & plasticity
  - Spatial summation
- **Terminology**
  - Neurons – units – nodes
  - Synapses – connections – architecture
  - Synaptic weight – connection strength (either positive or negative)

Model of a single neuron



Neuronal model

$$u_k = \sum_{j=1}^{m} w_{kj} x_j$$

Adder, weighted sum, linear combiner

$$v_k = u_k + b_k$$

Activation potential; b_k: bias

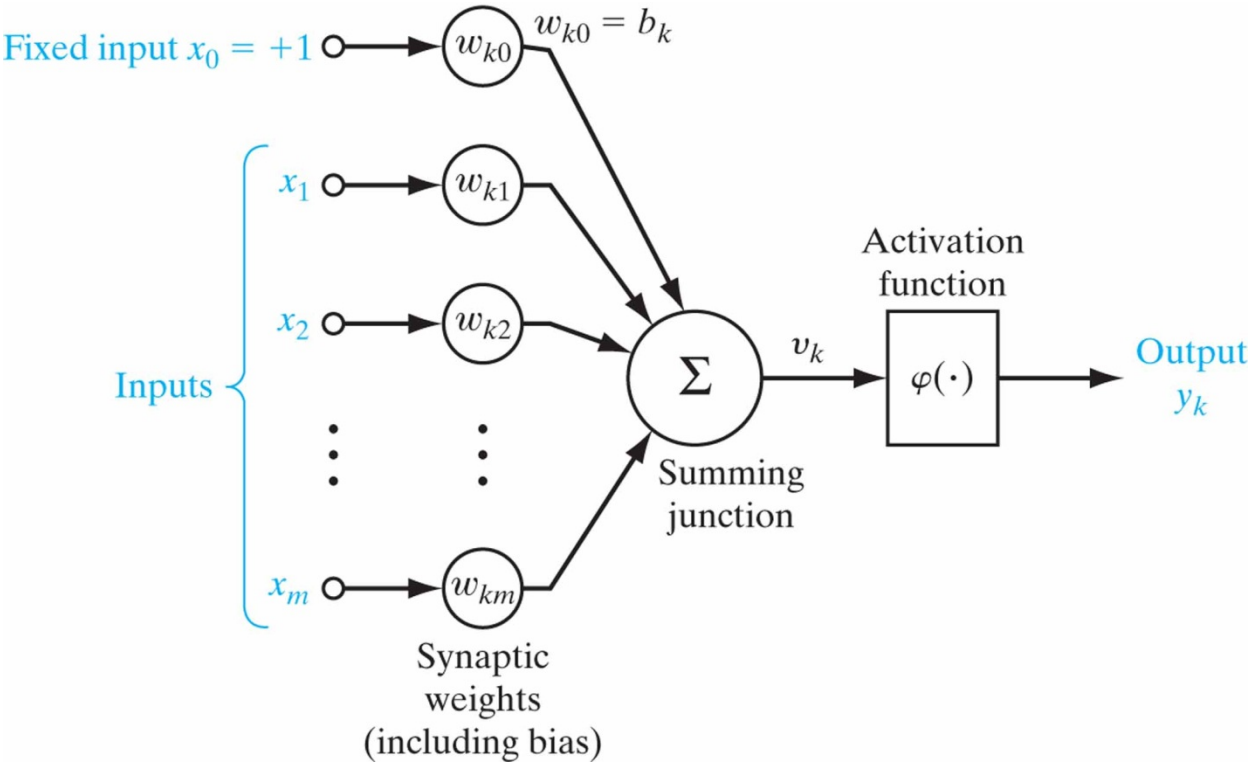
$$y_k = \varphi(v_k)$$

Output; φ: activation function

Another way of including bias

Set $x_0 = +1$ and $w_{k0} = b_k$

So we have $v_k = \sum_{j=0}^{m} w_{kj} x_j$



McCulloch-Pitts model

$x_i \in \{-1, 1\}$ Bipolar input

$$y = \varphi\left(\sum_{i=1}^{m} w_i x_i + b\right)$$

$$\varphi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ -1 & \text{if } v < 0 \end{cases}$$

A form of signum (sign) function

- Note difference from textbook; also number of neurons in the brain

McCulloch-Pitts model (cont.)

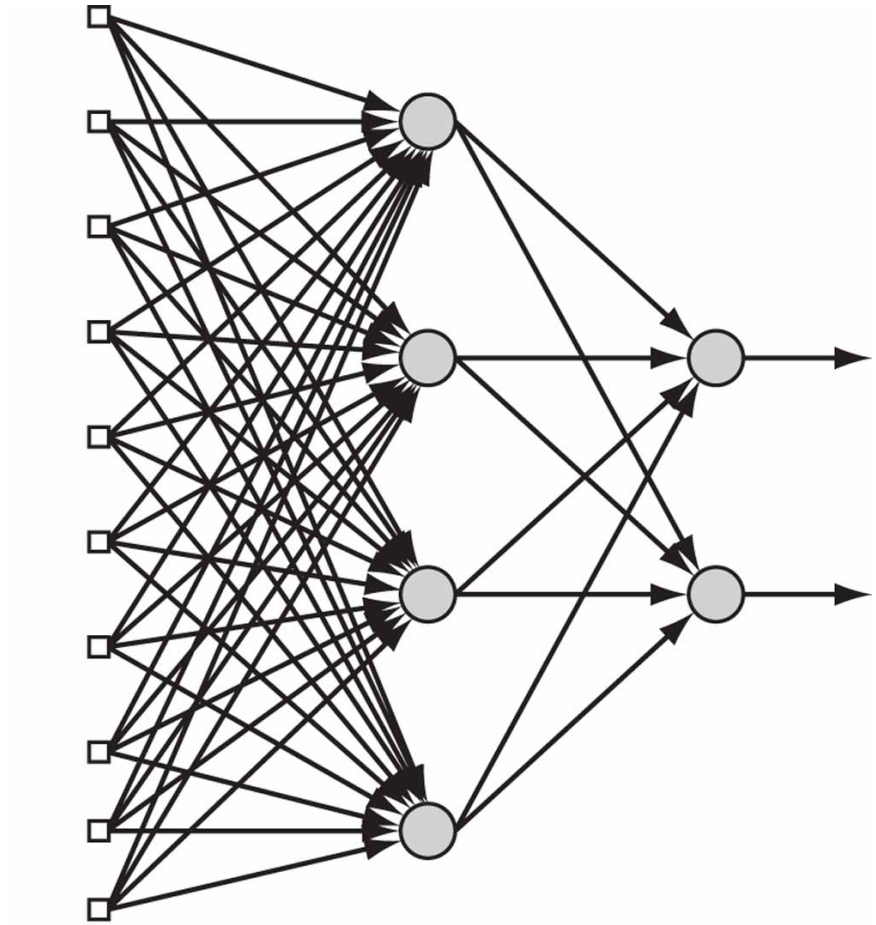
- Example logic gates (see blackboard)

- McCulloch-Pitts networks (introduced in 1943) are the first class of abstract computing machines: finite-state automata
  - Finite-state automata can compute any logic (Boolean) function

Network architecture

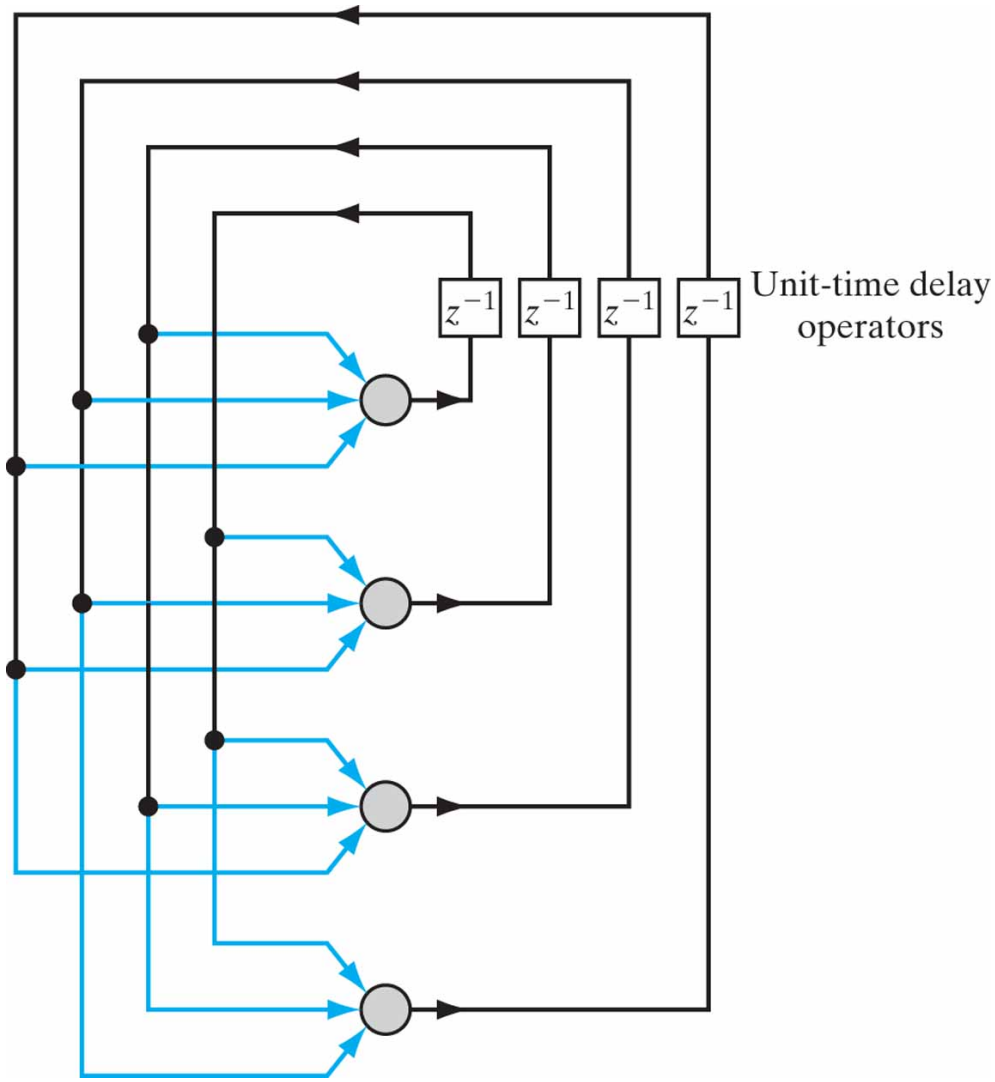
- View an NN as a connected, directed graph, which defines its architecture
  - Feedforward nets: loop-free graph
  - Recurrent nets: with loops

Feedforward net



- Since the input layer consists of source nodes, it is typically not counted when we talk about the number of layers in a feedforward net
- For example, the architecture of 10-4-2 counts as a two-layer net

A one-layer recurrent net



In this net, the input typically sets the initial condition of the output layer

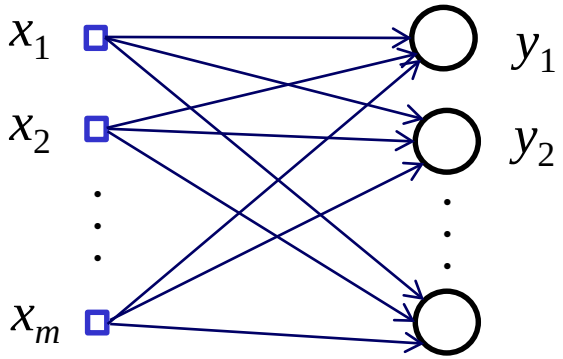
Network components

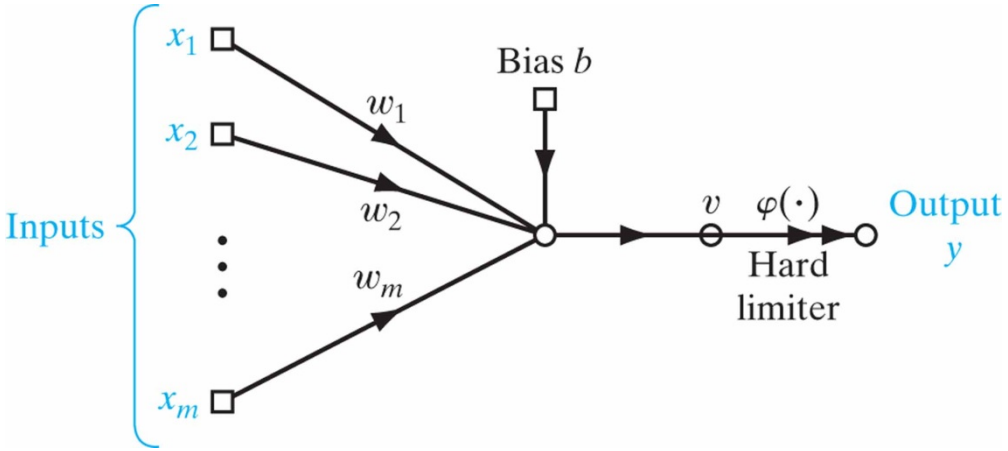
- **Three components characterize a neural net**
  - Architecture
  - Activation function
  - Learning rule (algorithm)

Perceptrons

Perceptrons

- Architecture: **one-layer feedforward net**
  - Without loss of generality, consider a single-neuron perceptron





Definition

$$y = \varphi(v)$$

$$v = \sum_{i=1}^{m} w_i x_i + b$$

$$\varphi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

Hence a McCulloch-Pitts neuron, but with real-valued inputs

Pattern recognition

- With a bipolar output, the perceptron performs a 2-class classification problem
  - Apples vs. oranges
- How do we learn to perform a classification problem?
- Task: The perceptron is given pairs of input $\mathbf{x}_p$ and desired output d_p. How to find $\mathbf{w}$ (with b incorporated) so that

$$y_p = d_p, \quad \text{for all } p?$$

Decision boundary

- The decision boundary for a given **w**:

$$g(\mathbf{x}) = \sum_{i=1}^{m} w_i x_i + b = 0$$

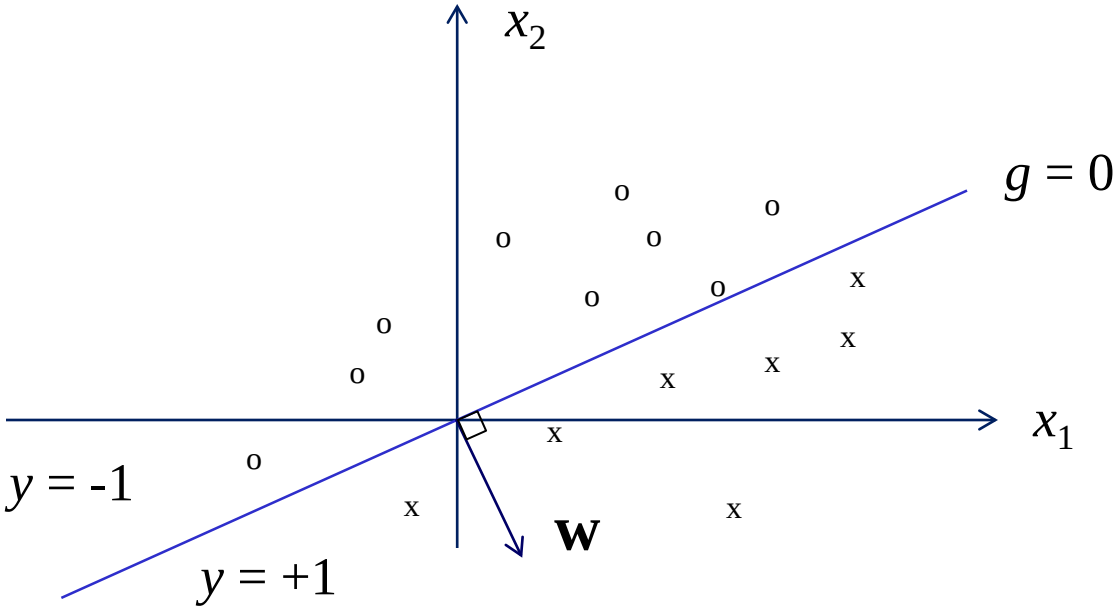
  - *g* is also called the discriminant function for the perceptron, and it is a linear function of **x**. Hence it is a linear discriminant function

Example

- See blackboard

Decision boundary (cont.)

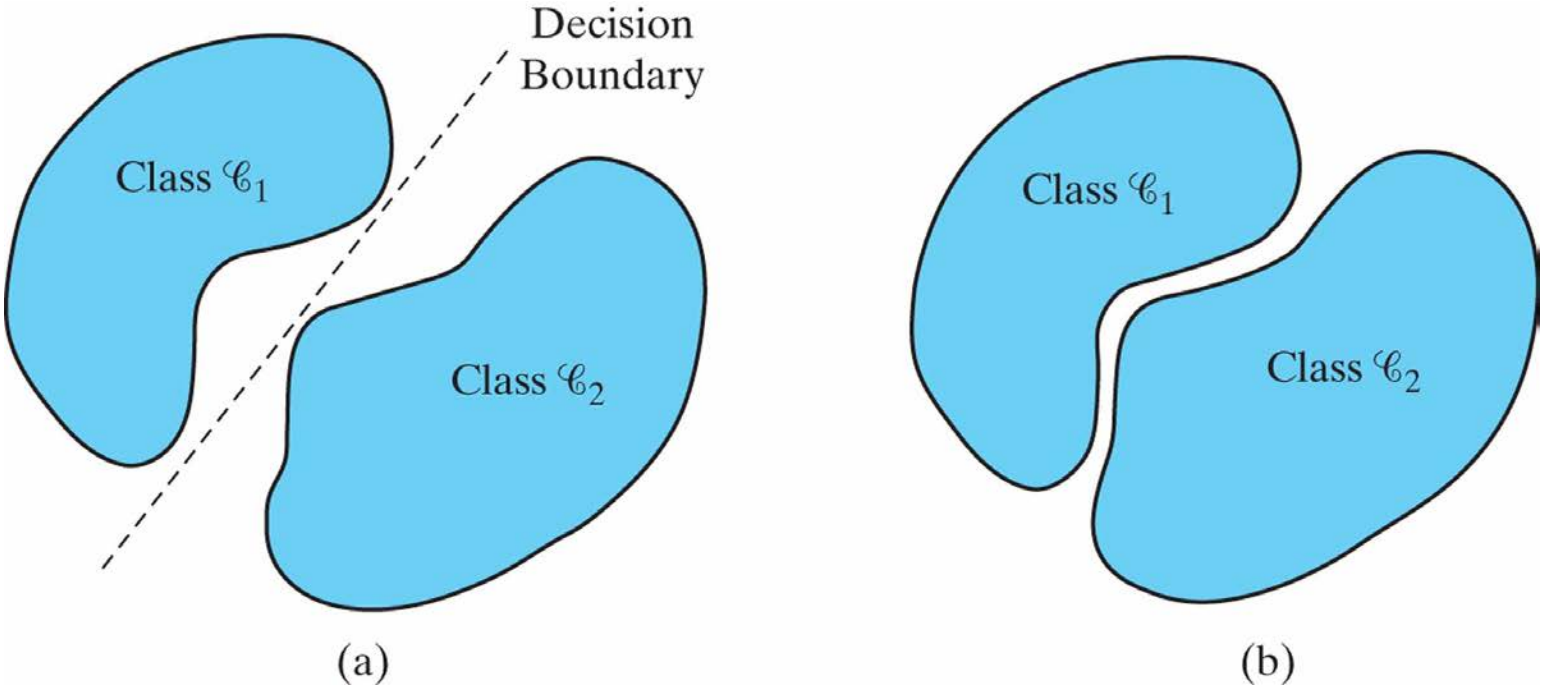
- For an *m*-dimensional input space, the decision boundary is an (*m* – 1)-dimensional hyperplane perpendicular to **w**. The hyperplane separates the input space into two halves, with one half having $y = 1$, and the other half having $y = -1$
  - When $b = 0$, the hyperplane goes through the origin



Linear separability

- For a set of input patterns $\mathbf{x}_p$, if there exists one $\mathbf{w}$ that separates $d = 1$ patterns from $d = -1$ patterns, then the classification problem is linearly separable
  - In other words, there exists a linear discriminant function that produces no classification error
  - Examples: AND, OR, XOR (see blackboard)
- A very important concept

Linear separability: a more general illustration



Perceptron learning rule

- Strengthen an active synapse if the postsynaptic neuron fails to fire when it should have fired; weaken an active synapse if the neuron fires when it should not have fired
  - Formulated by Rosenblatt based on biological intuition

Quantitatively

$$w(n+1) = w(n) + \Delta w(n)$$

$$= w(n) + \eta[d(n) - y(n)]x(n)$$

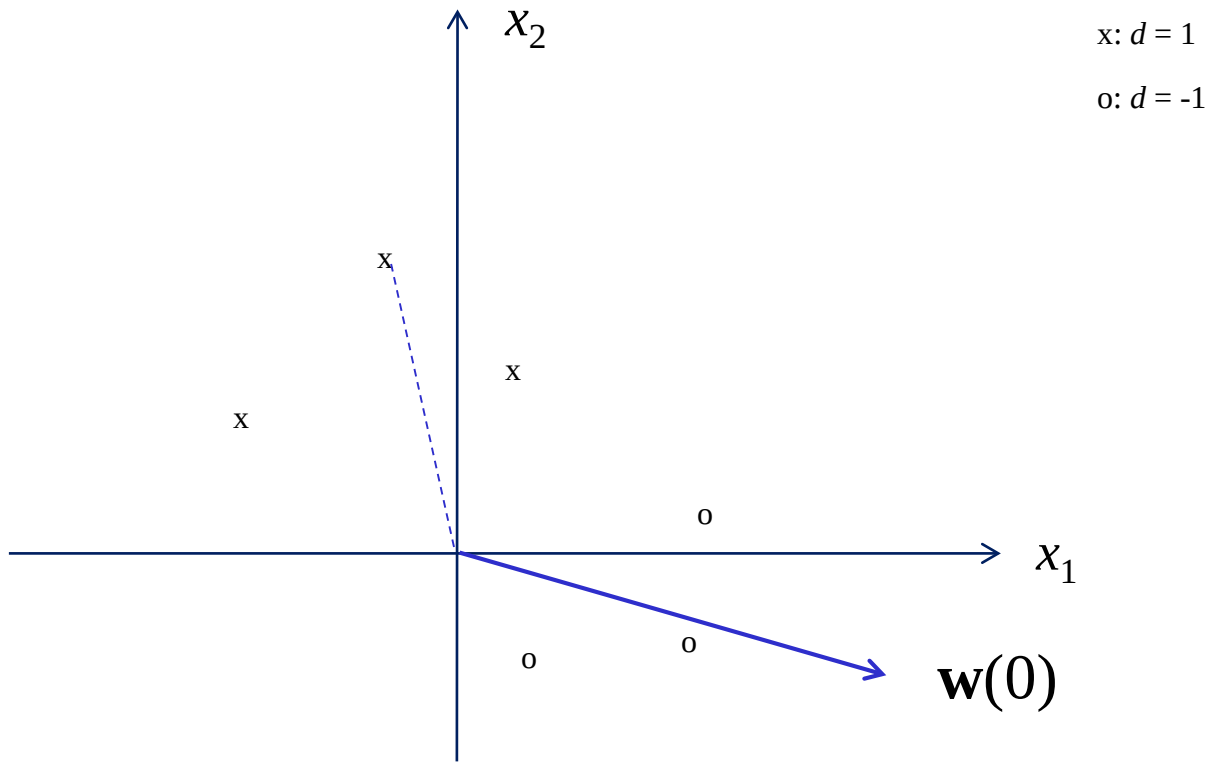
- n: iteration number
- η: step size or learning rate

In vector form

$$\Delta \mathbf{w} = \eta[d - y]\mathbf{x}$$

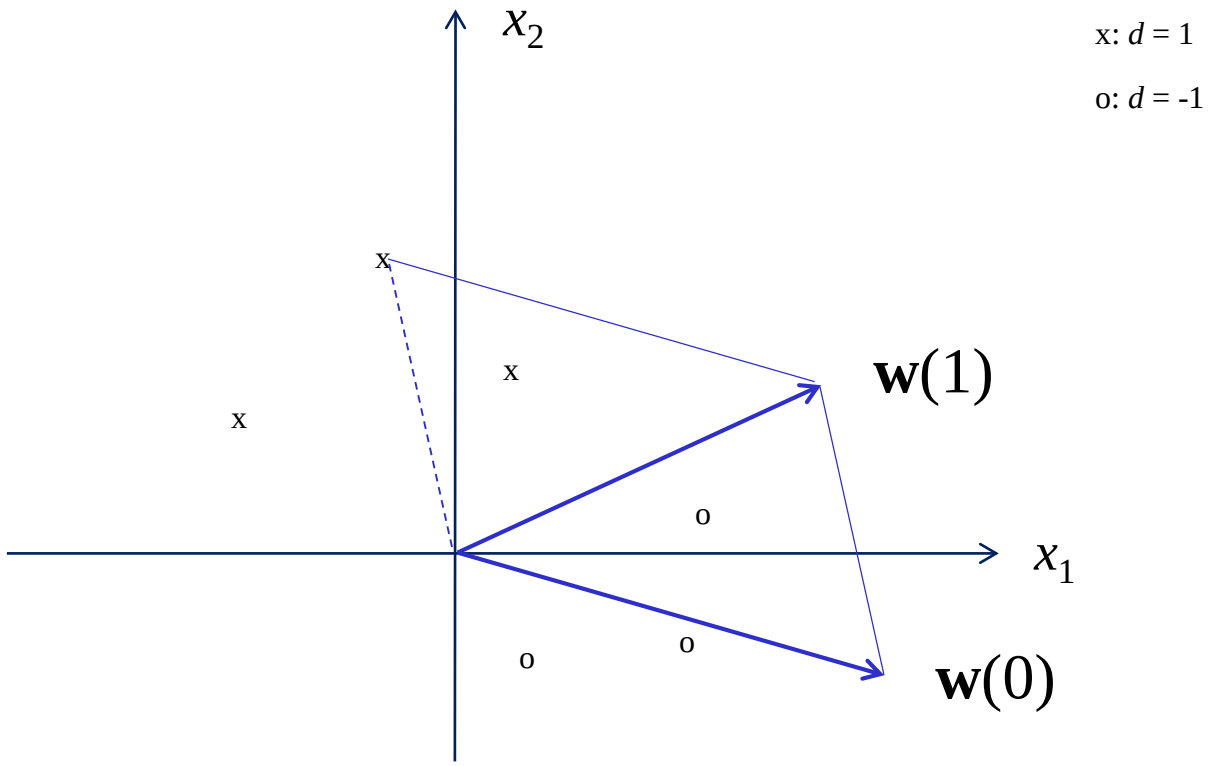
Geometric interpretation

- Assume $\eta = 1/2$



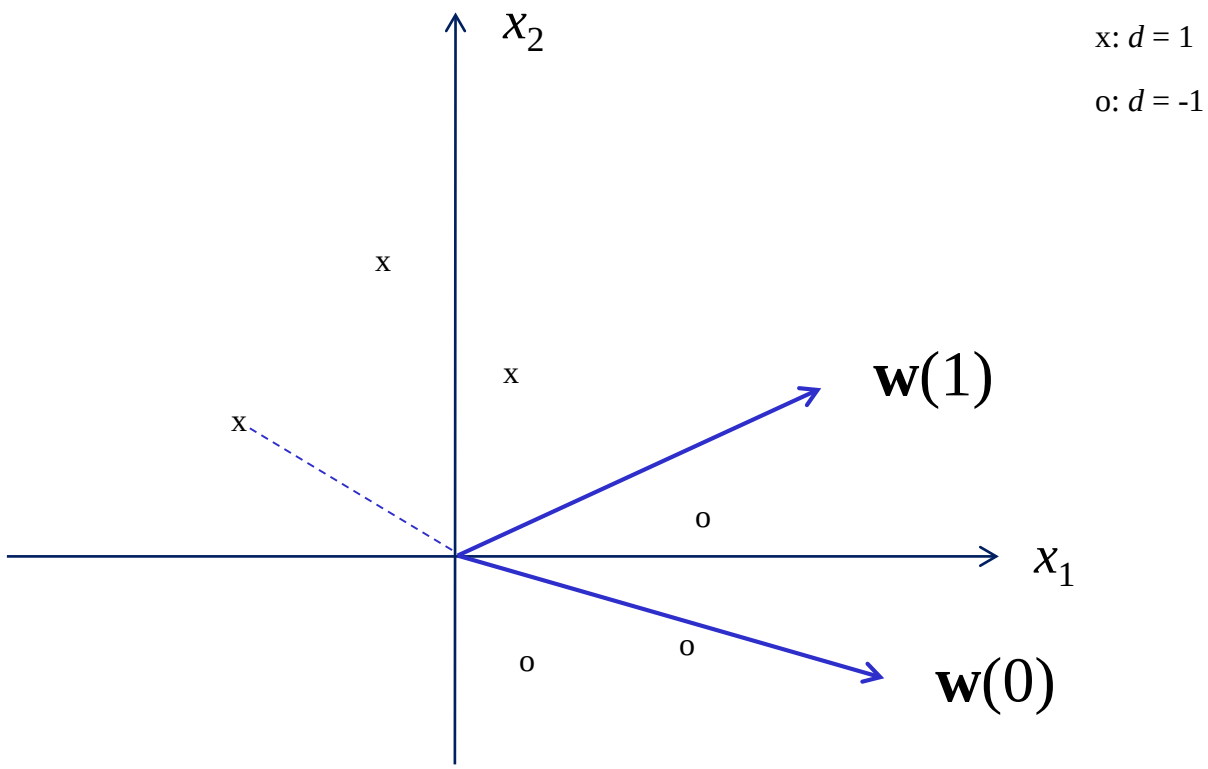
Geometric interpretation

- Assume $\eta = 1/2$



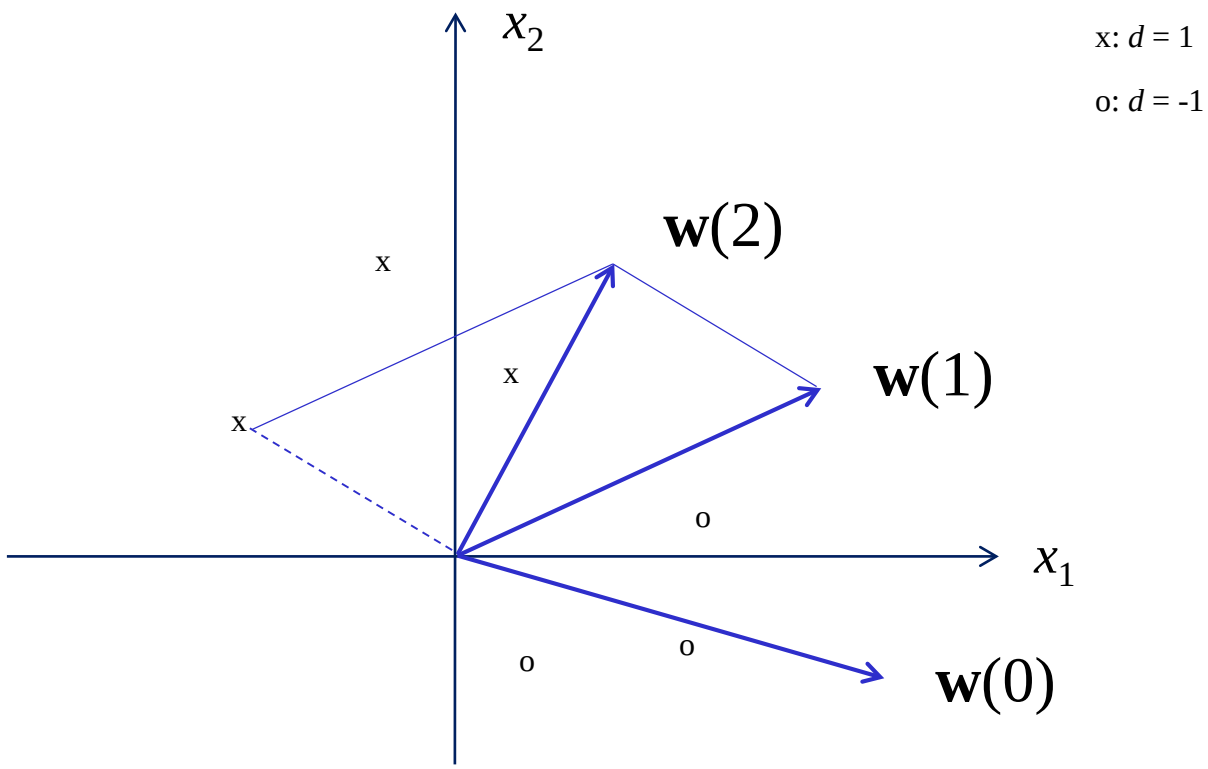
Geometric interpretation

- Assume $\eta = 1/2$



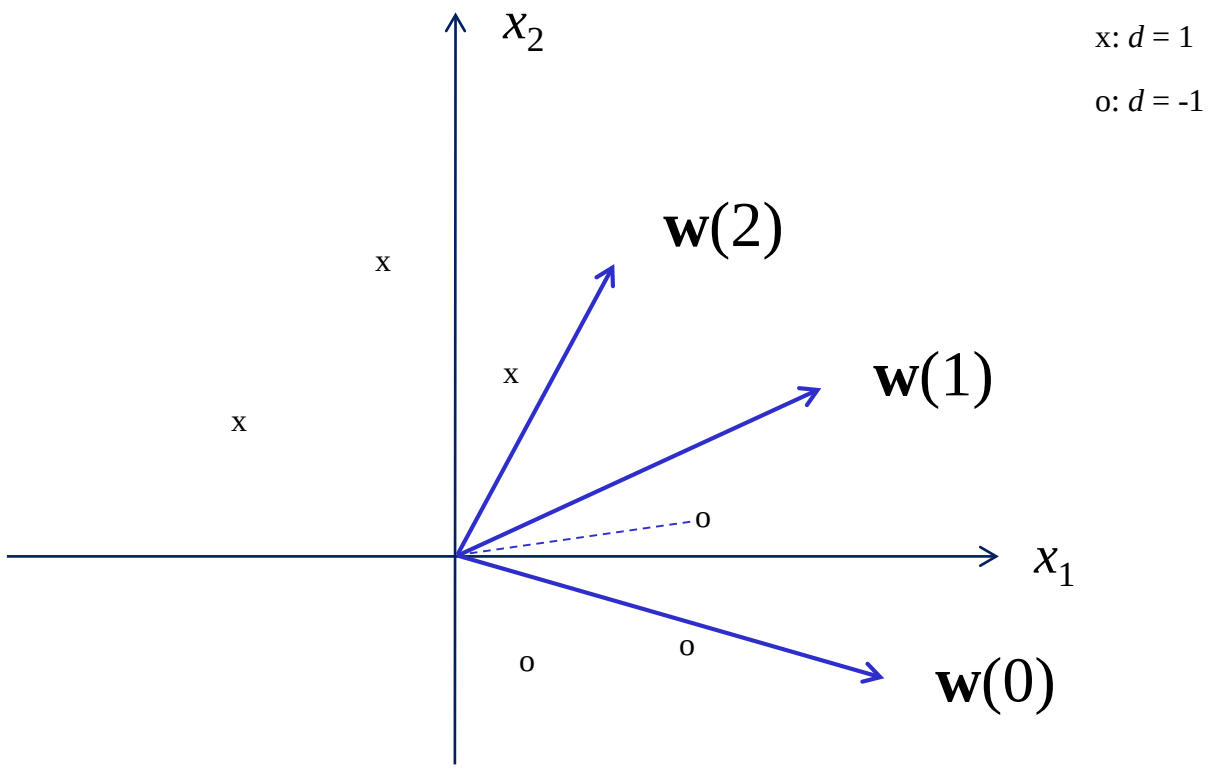
Geometric interpretation

- Assume $\eta = 1/2$



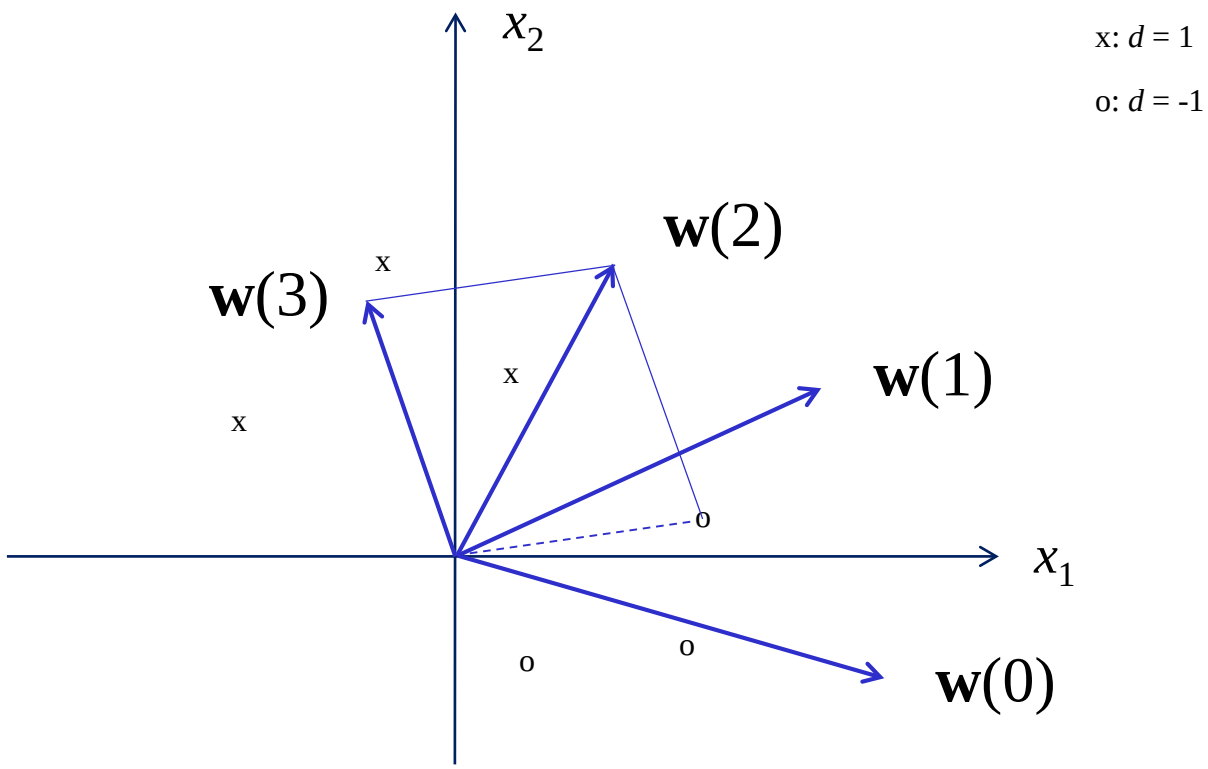
Geometric interpretation

- Assume $\eta = 1/2$

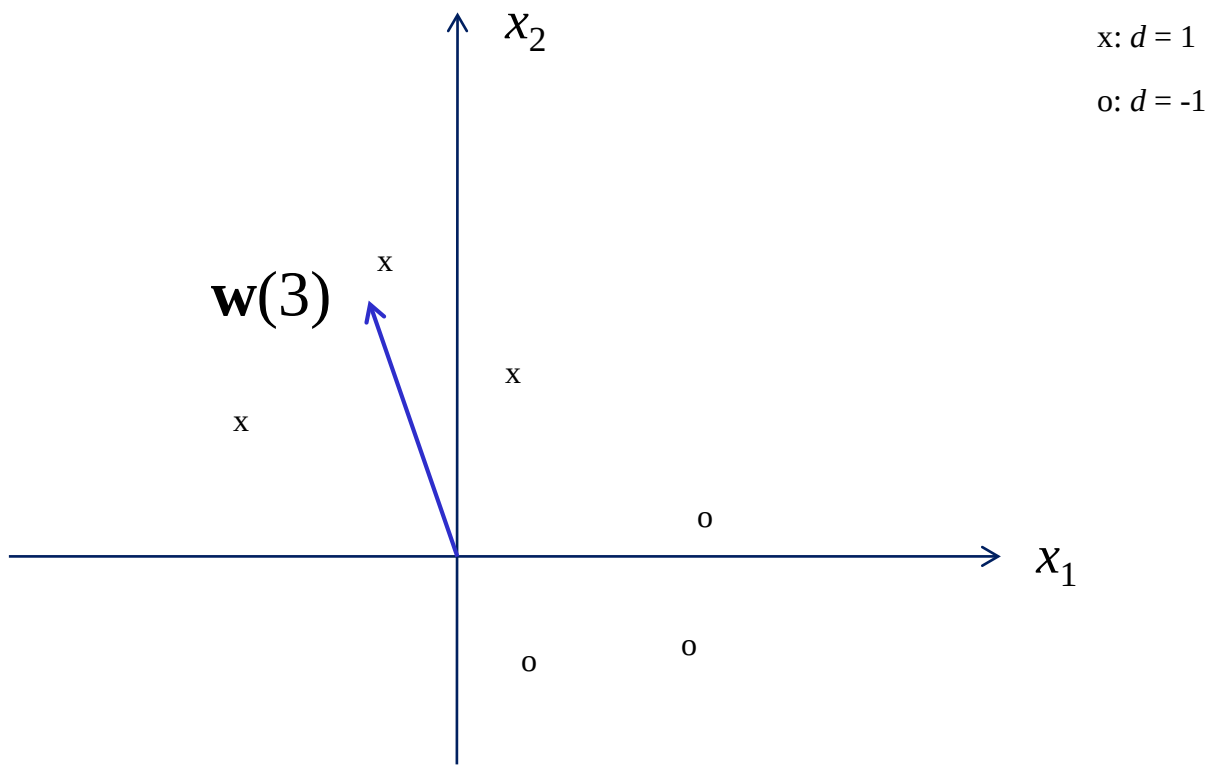


Geometric interpretation

- Assume $\eta = 1/2$



Geometric interpretation



Each weight update moves **w** closer to d = 1 patterns, or away from d = -1 patterns. **w**(3) solves the classification problem

Perceptron convergence theorem

- **Theorem**: *If a classification problem is linearly separable, a perceptron will reach a solution in a finite number of iterations*
- **[Proof]**

  Given a finite number of training patterns, because of linear separability, there exists a weight vector $\mathbf{w}_o$ so that

$$d_p \mathbf{w}_o^T \mathbf{x}_p \geq \alpha > 0 \qquad (1)$$

where $\alpha = \min_p (d_p \mathbf{w}_o^T \mathbf{x}_p)$

Proof (cont.)

- We assume that the initial weights are all zero. Let N_p denote the number of times $\mathbf{x}_p$ has been used for *actually* updating the weight vector at some point in learning

At that time:

$$\mathbf{w} = \sum_p N_p[\eta(d_p - y_p)\mathbf{x}_p]$$

$$= 2\eta \sum_p N_p d_p \mathbf{x}_p$$

Proof (cont.)

- Consider $\mathbf{w}_o^T \mathbf{w}$ first

$$\mathbf{w}_o^T \mathbf{w} = 2\eta \sum_p N_p d_p \mathbf{w}_o^T \mathbf{x}_p$$

due to (1)

$$\geq 2\eta\alpha \sum_p N_p$$

$$= 2\eta\alpha P \qquad (2)$$

where $P = \sum_p N_p$

Proof (cont.)

- Now consider the change in square length $\|\mathbf{w}\|^2$ after a single update by $\mathbf{x}$:

$$\Delta\|\mathbf{w}\|^2 = \|\mathbf{w}+\Delta\mathbf{w}\|^2 - \|\mathbf{w}\|^2 = \|\mathbf{w}+2\eta d\mathbf{x}\|^2 - \|\mathbf{w}\|^2$$

$$= 4\eta^2\|\mathbf{x}\|^2 + 4\eta d\mathbf{w}^T\mathbf{x}$$

$$\leq 4\eta^2\beta$$

where $\beta = \max_p \|\mathbf{x}_p\|^2$

Since upon an update, $d(\mathbf{w}^T\mathbf{x}) \leq 0$

Proof (cont.)

- By summing $\|\mathbf{w}\|^2$ for P steps we have the bound:

$$\|\mathbf{w}\|^2 \leq 4\eta^2 \beta P \qquad (3)$$

- Now square the cosine of the angle between $\mathbf{w}_o$ and $\mathbf{w}$, we have by (2) and (3)

$$1 \geq \frac{(\mathbf{w}_o^T \mathbf{w})^2}{\|\mathbf{w}_o\|^2 \|\mathbf{w}\|^2} \geq \frac{4\eta^2 \alpha^2 P^2}{4\eta^2 \beta P \|\mathbf{w}_o\|^2} = \frac{\alpha^2 P}{\beta \|\mathbf{w}_o\|^2}$$

Cauchy-Schwarz inequality

Proof (cont.)

- Thus, P must be finite to satisfy the above inequality. This complete the proof

- **Remarks**
  - In the case of $\mathbf{w}(0) = \mathbf{0}$, the learning rate has no effect on the proof. That is, the theorem holds no matter what η ($\eta > 0$) is
  - The solution weight vector is not unique

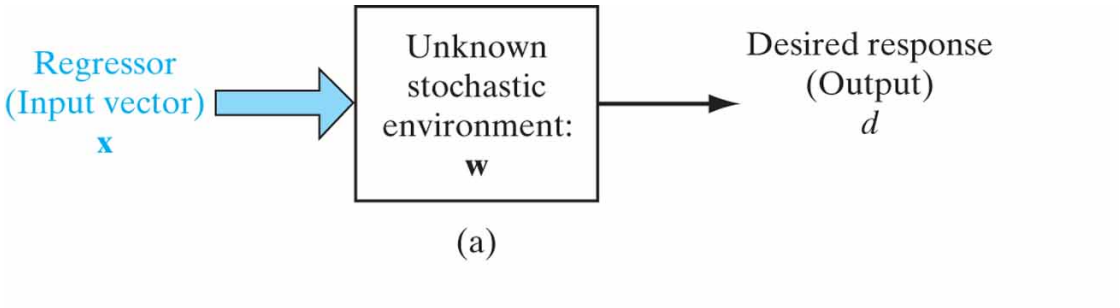
Generalization

- Performance of a learning machine on test patterns not used during training
- Example: Class 1: handwritten "*m*": class 2: handwritten "*n*"
  - See blackboard
- Perceptrons generalize by deriving a decision boundary in the input space. Selection of training patterns is thus important for generalization

CSE 5526: Introduction to Neural Networks

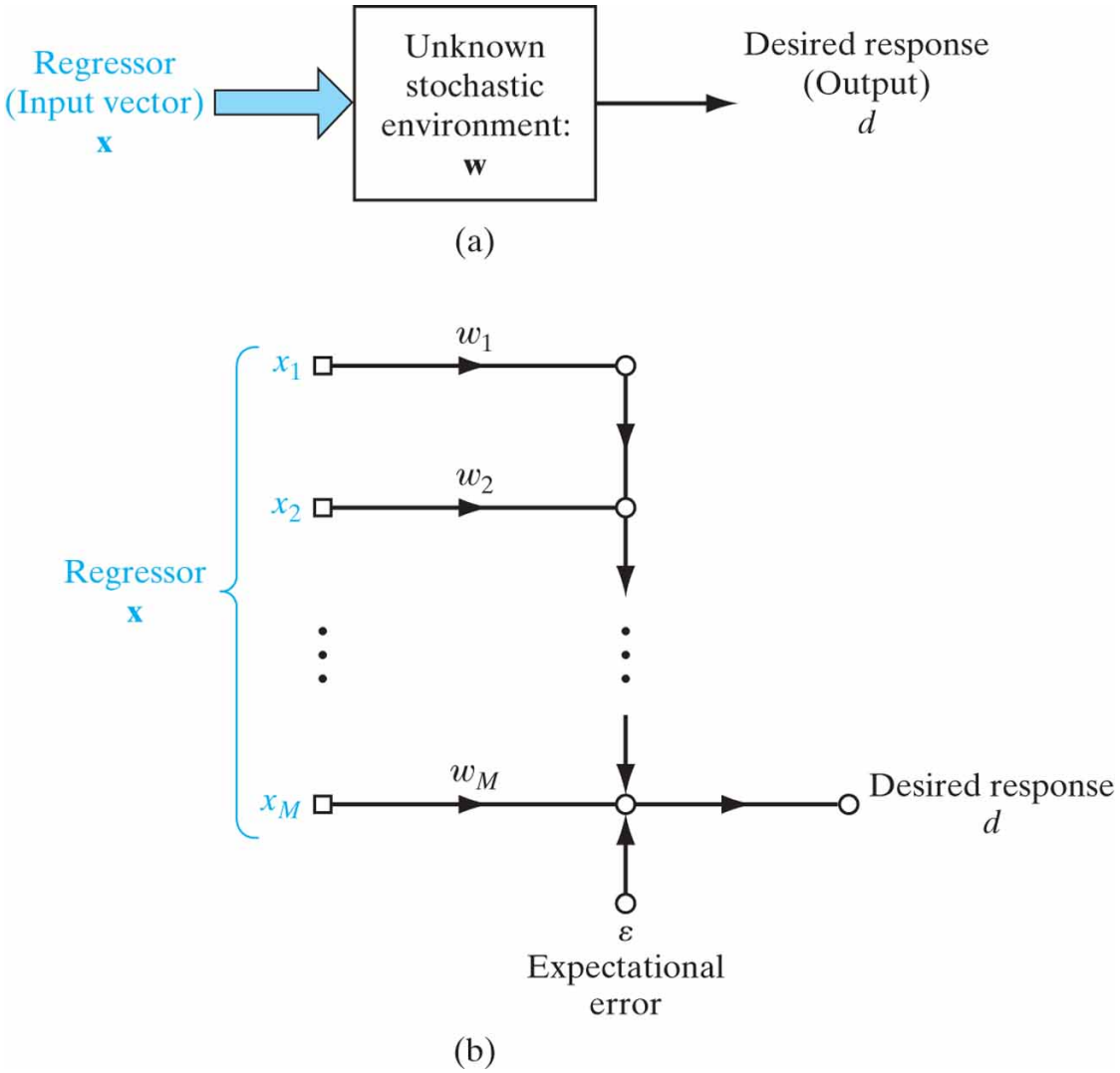
Linear Regression

Problem statement



(a)

Problem statement



Linear regression with one variable

- Given a set of N pairs of data $< x_i, d_i >$, approximate d by a linear function of x (regressor)

  i.e.

$$d \approx wx + b$$

  or

$$d_i = y_i + \varepsilon_i = \varphi(wx_i + b) + \varepsilon_i$$

$$= wx_i + b + \varepsilon_i$$

  where the activation function $\varphi(x) = x$ is a linear function, and it corresponds to a linear neuron. y is the output of the neuron, and

$$\varepsilon_i = d_i - y_i$$

  is called the regression (expectational) error

Linear regression (cont.)

- The problem of regression with one variable is how to choose *w* and *b* to minimize the regression error

- The least squares method aims to minimize the square error *E*:

$$E = \frac{1}{2}\sum_{i=1}^{N}\varepsilon_i^2 = \frac{1}{2}\sum_{i=1}^{N}(d_i - y_i)^2$$

Linear regression (cont.)

- To minimize the two-variable square function, set

$$\begin{cases} \dfrac{\partial E}{\partial b} = 0 \\ \dfrac{\partial E}{\partial w} = 0 \end{cases}$$

Linear regression (cont.)

$$\frac{\partial E}{\partial b} = \frac{1}{2}\sum_i \frac{\partial (d_i - wx_i - b)^2}{\partial b}$$

$$= -\sum_i (d_i - wx_i - b) = 0$$

$$\frac{\partial E}{\partial w} = \frac{1}{2}\sum_i \frac{\partial (d_i - wx_i - b)^2}{\partial w}$$

$$= -\sum_i (d_i - wx_i - b)x_i = 0$$

Linear regression (cont.)

- Hence

$$b = \frac{\sum_i x_i^2 \sum_i d_i - \sum_i x_i \sum_i x_i d_i}{N[\sum_i (x_i - \bar{x})^2]}$$

Derive yourself!

$$w = \frac{\sum_i (x_i - \bar{x})(d_i - \bar{d})}{\sum_i (x_i - \bar{x})^2}$$

where an overbar (i.e. $\bar{x}$) indicates the mean

Linear regression (cont.)

- This method gives an optimal solution, but it can be time- and memory-consuming as a batch solution

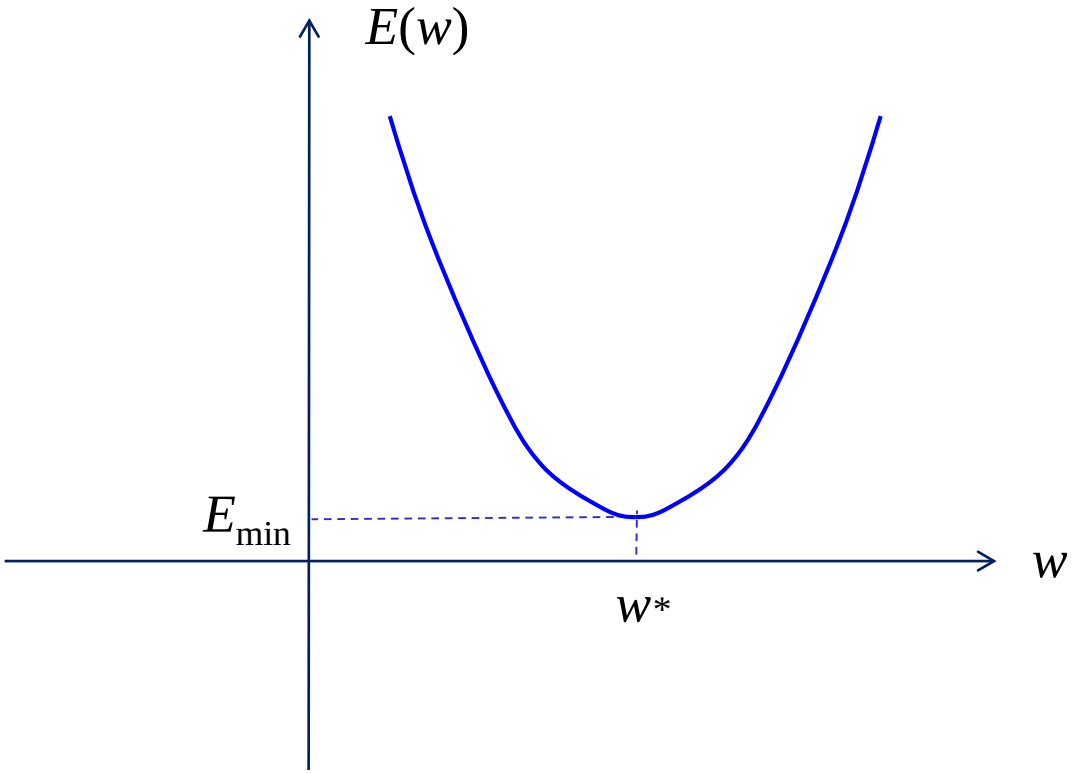
Finding optimal parameters via search

- Without loss of generality, set $b = 0$

$$E(w) = \frac{1}{2}\sum_{i=1}^{N}(d_i - wx_i)^2$$

$E(w)$ is called a cost function

Cost function



- Question: how can we update w to minimize E?

Gradient and directional derivatives

- Without loss of generality, consider a two-variable function $f(x, y)$. The gradient of $f(x, y)$ at a given point $(x_0, y_0)^T$ is

$$\nabla f = \left(\frac{\partial f(x,y)}{\partial x}, \frac{\partial f(x,y)}{\partial y}\right)^T \Bigg|_{\substack{x=x_0 \\ y=y_0}}$$

$$= f_x(x_0, y_0)\mathbf{u}_x + f_y(x_0, y_0)\mathbf{u}_y$$

where $\mathbf{u}_x$ and $\mathbf{u}_y$ are unit vectors in the x and y directions, and $f_x = \partial f / \partial x$ and $f_y = \partial f / \partial y$

Gradient and directional derivatives (cont.)

- At any given direction, $\mathbf{u} = a\mathbf{u}_x + b\mathbf{u}_y$, with $\sqrt{a^2 + b^2} = 1$, the directional derivative at $(x_0, y_0)^T$ along the unit vector $\mathbf{u}$ is

$$D_{\mathbf{u}} f(x_0, y_0) = \lim_{h \to 0} \frac{f(x_0 + ha, y_0 + hb) - f(x_0, y_0)}{h}$$

$$= \lim_{h \to 0} \frac{[f(x_0 + ha, y_0 + hb) - f(x_0, y_0 + hb)] + [f(x_0, y_0 + hb) - f(x_0, y_0)]}{h}$$

$$= af_x(x_0, y_0) + bf_y(x_0, y_0)$$

$$= \nabla f^T(x_0, y_0)\mathbf{u}$$

  - Which direction has the greatest slope?
    - **The gradient** because of the dot product!

Gradient and directional derivatives (cont.)

- Example: see blackboard

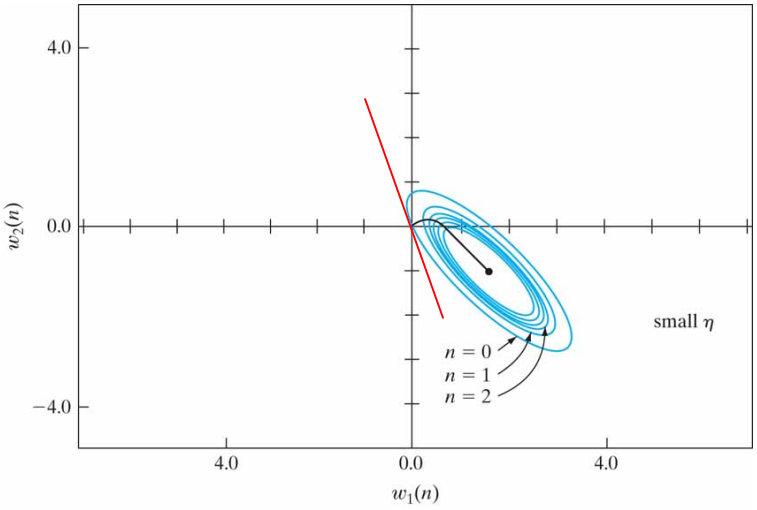
Gradient and directional derivatives (cont.)

- To find the gradient at a particular point $(x_0, y_0)^T$, first find the level curve or contour of $f(x, y)$ at that point, $C(x_0, y_0)$. A tangent vector $\mathbf{u}$ to C satisfies

$$D_{\mathbf{u}} = \nabla f^T(x_0, y_0)\mathbf{u} = 0$$

because $f(x, y)$ is constant on a level curve. Hence the gradient vector is perpendicular to the tangent vector

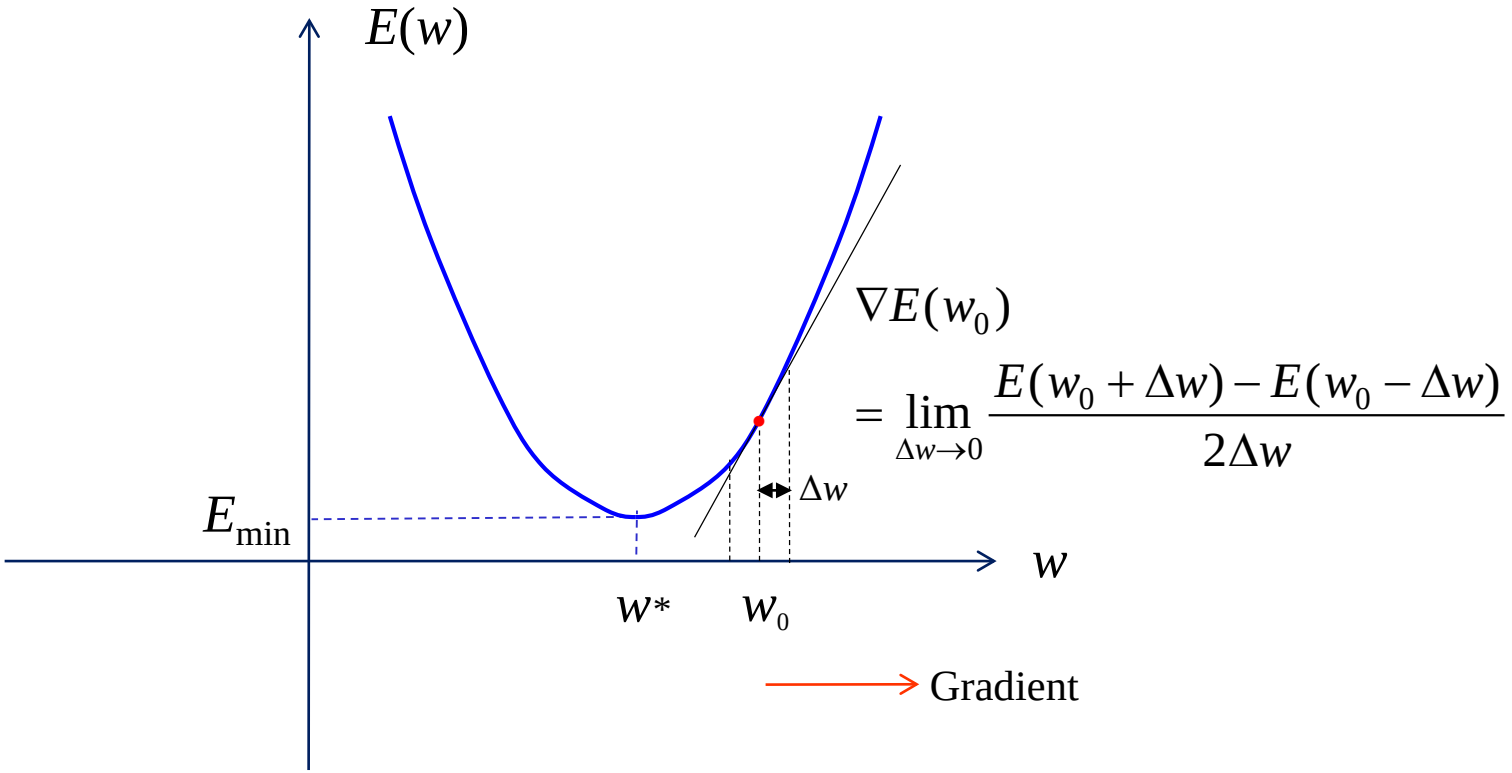
An illustration of level curves



Gradient and directional derivatives (cont.)

- The gradient of a cost function is a vector with the dimension of **w** that points to the direction of maximum E increase and with a magnitude equal to the slope of the tangent of the cost function along that direction
  - Can the slope be negative?

Gradient illustration



Gradient descent

- Minimize the cost function via gradient (steepest) descent – a case of hill-climbing

$$w(n+1) = w(n) - \eta \nabla E(n)$$

n: iteration number

η: learning rate

- See previous figure

Gradient descent (cont.)

- For the mean-square-error cost function:

$$E(n) = \frac{1}{2}e^2(n) = \frac{1}{2}[d(n) - y(n)]^2$$

$$= \frac{1}{2}[d(n) - w(n)x(n)]^2$$

linear neurons

$$\nabla E(n) = \frac{\partial E}{\partial w(n)} = \frac{1}{2}\frac{\partial e^2(n)}{\partial w(n)}$$

$$= -e(n)x(n)$$

Gradient descent (cont.)

- Hence

$$w(n+1) = w(n) + \eta e(n) x(n)$$

$$= w(n) + \eta [d(n) - y(n)] x(n)$$

  - This is the least-mean-square (LMS) algorithm, or the Widrow-Hoff rule

Multi-variable case

- The analysis for the one-variable case extends to the multi-variable case

$$E(n) = \frac{1}{2}[d(n) - \mathbf{w}^T(n)\mathbf{x}(n)]^2$$

$$\nabla E(\mathbf{w}) = \left(\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_m}\right)^T$$

where $w_0 = b$ (bias) and $x_0 = 1$, as done for perceptron learning

Multi-variable case (cont.)

- The LMS algorithm

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \eta \nabla E(n)$$

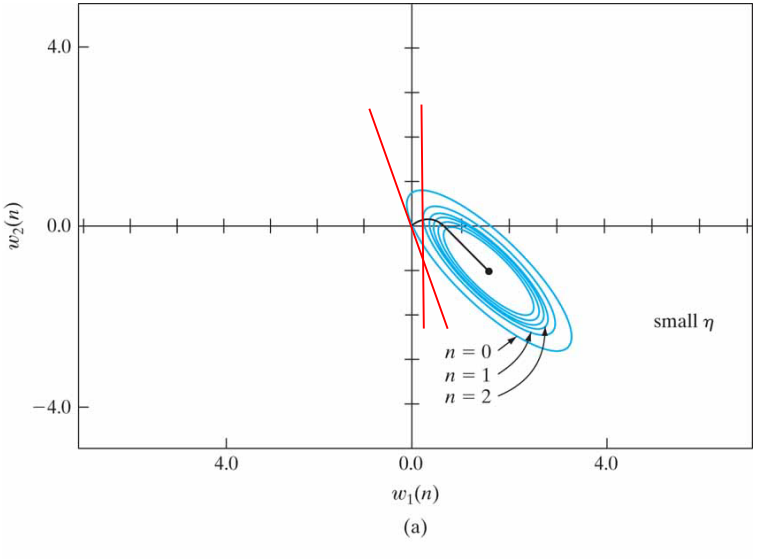
$$= \mathbf{w}(n) + \eta e(n)\mathbf{x}(n)$$

$$= \mathbf{w}(n) + \eta [d(n) - y(n)]\mathbf{x}(n)$$

LMS algorithm

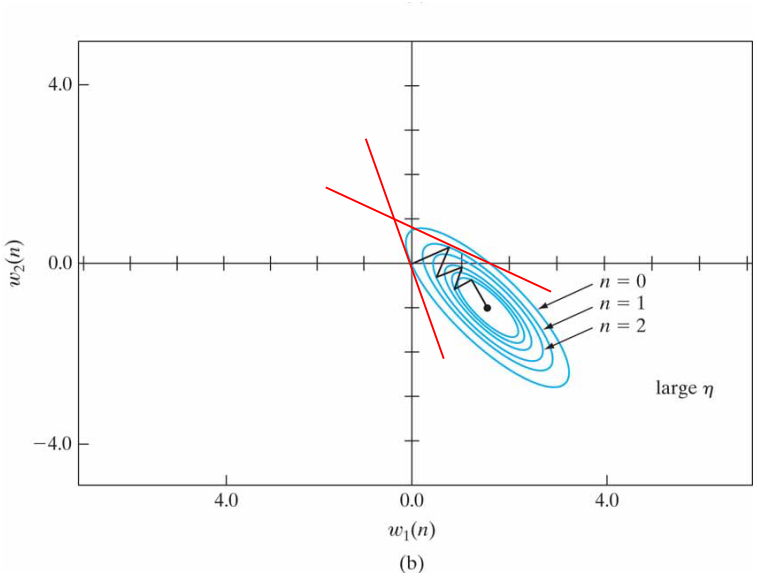
- **Remarks**
  - The LMS rule is exactly the same in math form as the perceptron learning rule
  - Perceptron learning is for McCulloch-Pitts neurons, which are nonlinear, whereas LMS learning is for linear neurons. In other words, perceptron learning is for classification and LMS is for function approximation
  - LMS should be less sensitive to noise in the input data than perceptrons. On the other hand, LMS learning converges slowly
  - Newton's method changes weights in the direction of the minimum $E(\mathbf{w})$ and leads to fast convergence. But it is not an online version and computationally extensive

Stability of adaptation



- When η is too small, learning converges slowly

Stability of adaptation (cont.)



- When η is too large, learning doesn't converge

Learning rate annealing

- Basic idea: start with a large rate but gradually decrease it
- **Stochastic approximation**

$$\eta(n) = \frac{c}{n}$$

c is a positive parameter

Learning rate annealing (cont.)

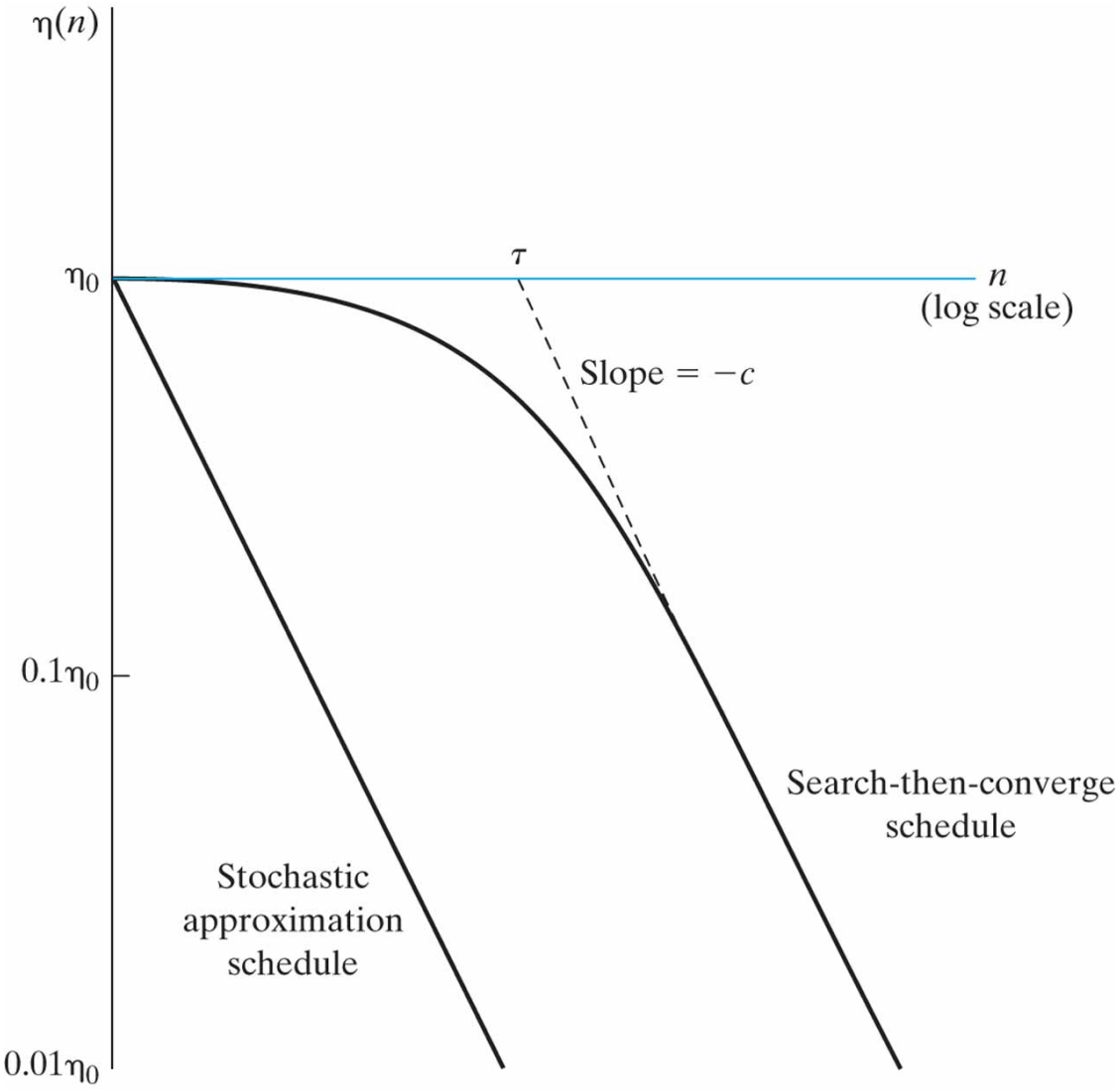
- **Search-then-converge**

$$\eta(n) = \frac{\eta_0}{1 + (n/\tau)}$$

η_0 and τ are positive parameters

- When n is small compared to τ, learning rate is approximately constant
- When n is large compared to τ, learning rate schedule roughly follows stochastic approximation

Rate annealing illustration



Nonlinear neurons

- To extend the LMS algorithm to nonlinear neurons, consider differentiable activation function φ at iteration n

$$E(n) = \frac{1}{2}[d(n) - y(n)]^2$$

$$= \frac{1}{2}[d(n) - \varphi(\sum_j w_j x_j(n))]^2$$

Nonlinear neurons (cont.)

- By chain rule of differentiation

$$\frac{\partial E}{\partial w_j} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial v}\frac{\partial v}{\partial w_j}$$

$$= -[d(n) - y(n)]\varphi'(v(n))x_j(n)$$

$$= -e(n)\varphi'(v(n))x_j(n)$$

Nonlinear neurons (cont.)

- The gradient descent gives

$$w_j(n+1) = w_j(n) + \eta \underline{e(n)\varphi'(v(n))} x_j(n)$$

$$= w_j(n) + \eta \delta(n) x_j(n)$$

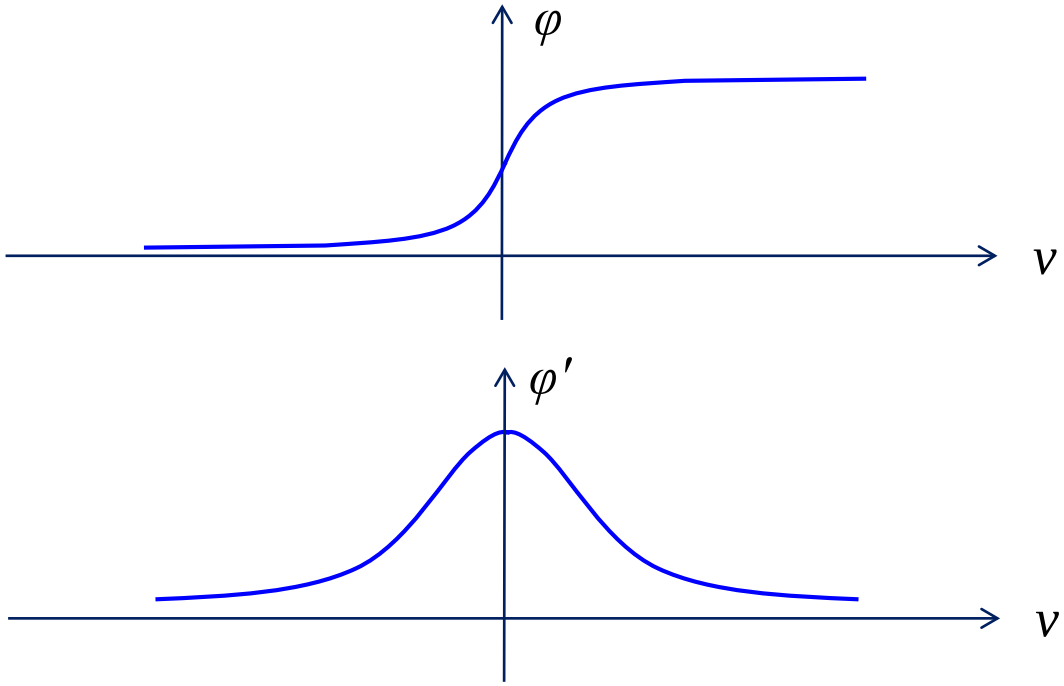
  - The above is called the delta (δ) rule
- If we choose a logistic sigmoid for φ

$$\varphi(v) = \frac{1}{1+\exp(-av)}$$

then

$$\varphi'(v) = a\varphi(v)[1-\varphi(v)]$$ (see textbook)

Role of activation function



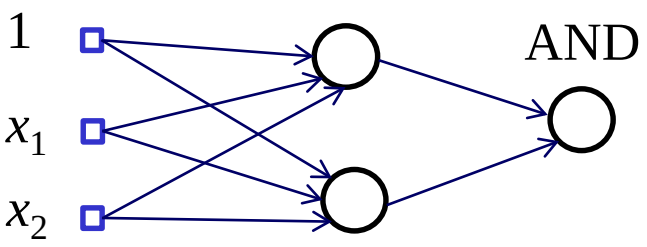
- The role of φ': weight update is most sensitive when v is near zero

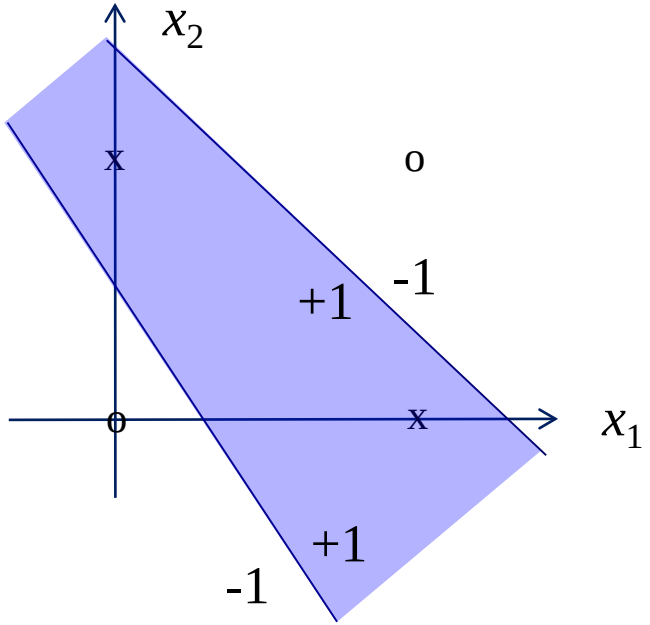
CSE 5526: Introduction to Neural Networks

Multilayer Perceptrons (MLPs)

Motivation

- Multilayer networks are more powerful than single-layer nets
  - Example: XOR problem cannot be solved by a perceptron but is solvable with a 2-layer perceptron

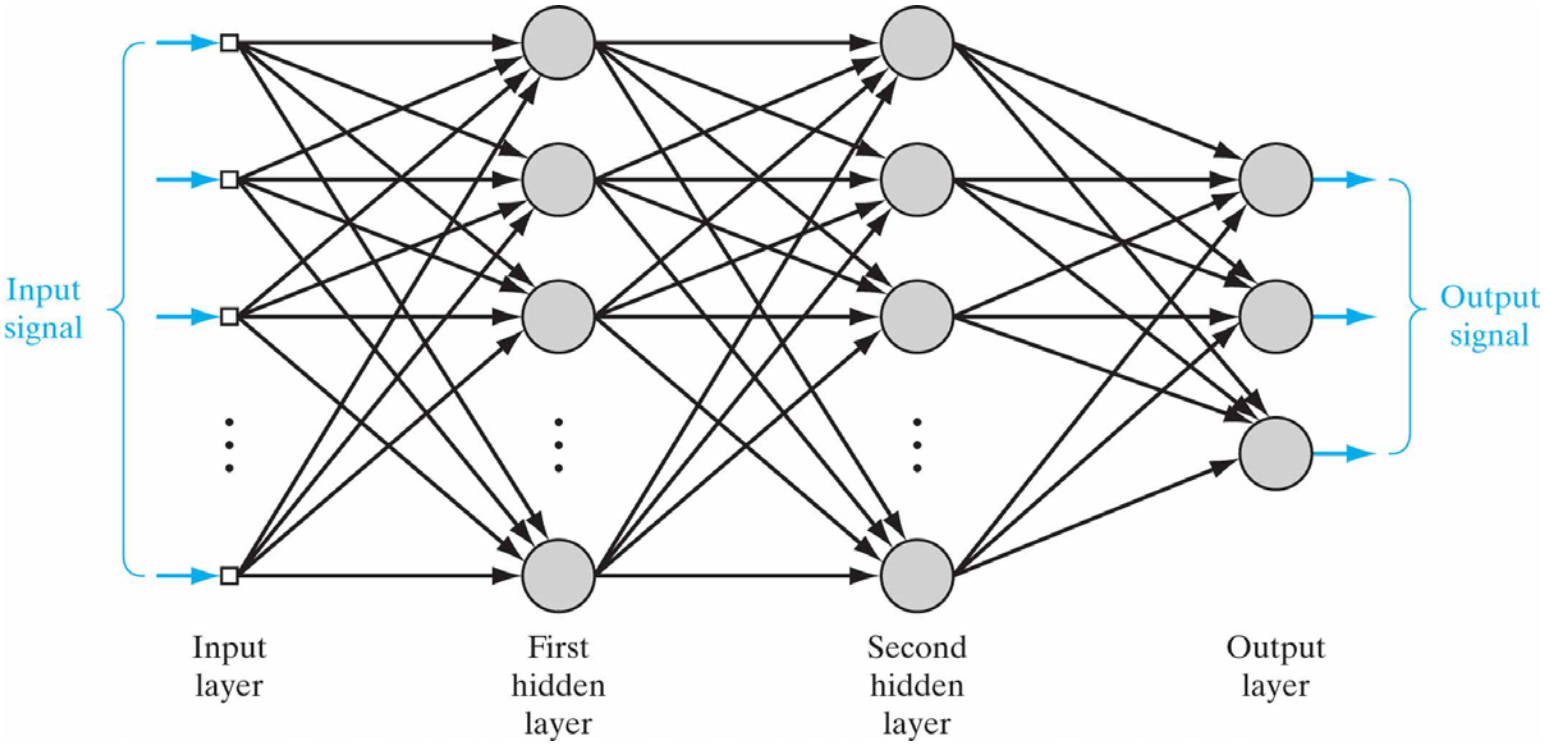




Power of nonlinearity

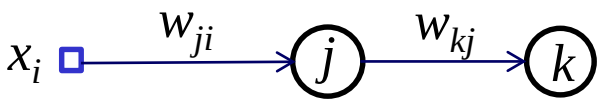
- For linear neurons, a multilayer net is equivalent to a single-layer net. This is not the case for nonlinear neurons
  - Why?

MLP architecture



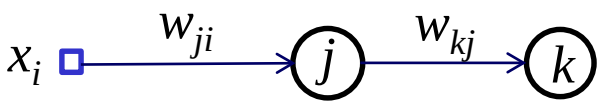
Notations

- Notation for one hidden layer



- The format of presentation to an MLP is the same as to a perceptron

Backpropagation



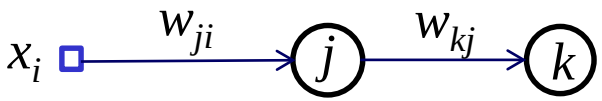
- Derive the backpropagation (backprop) algorithm by cost minimization via gradient descent

  For the output layer at iteration *n*, we have

$$E(n) = \frac{1}{2}\sum_k [d_k(n) - y_k(n)]^2$$

$$= \frac{1}{2}\sum_k [d_k(n) - \varphi(\sum_j w_{kj} y_j)]^2$$

Backprop (cont.)



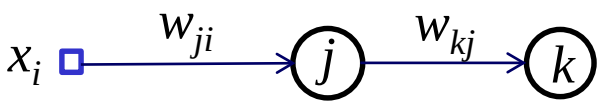
- Then

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial v_k}\frac{\partial v_k}{\partial w_{kj}}$$

$$= -e_k \varphi'(v_k)\frac{\partial v_k}{\partial w_{kj}}$$

$$= -e_k \varphi'(v_k) y_j$$

Backprop (cont.)



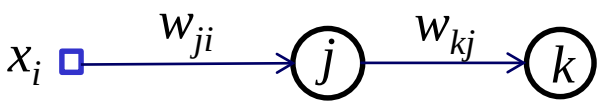
- Hence

$$w_{kj}(n+1) = w_{kj}(n) - \eta \frac{\partial E}{\partial w_{kj}}$$

$$= w_{kj}(n) + \eta \underline{e_k(n)\varphi'(v_k)} y_j(n)$$

$$= w_{kj}(n) + \eta \delta_k(n) y_j(n) \qquad \text{(δ rule)}$$

Backprop (cont.)



- For the hidden layer at iteration *n*,

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial v_j}\frac{\partial v_j}{\partial w_{ji}}$$

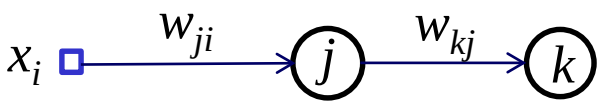
$$= \frac{\partial}{\partial y_j}\{\frac{1}{2}\sum_k [d_k - \varphi(\sum_j w_{kj} y_j)]^2\}\varphi'(v_j)x_i$$

$$= -\sum_k [d_k - \varphi(\sum_j w_{kj} y_j)]\frac{\partial \varphi(\sum_j w_{kj} y_j)}{\partial y_j}\varphi'(v_j)x_i$$

$$= -\sum_k (d_k - y_k)\varphi'(v_k)w_{kj}\varphi'(v_j)x_i$$

$$= -\varphi'(v_j)(\sum_k \delta_k w_{kj})x_i$$

Backprop (cont.)



- Therefore,

$$w_{ji}(n+1) = w_{ji}(n) + \eta\varphi'(v_j(n))[\sum_k \delta_k(n) w_{kj}(n)]x_i(n)$$

$$= w_{ji}(n) + \eta\delta_j(n)x_i(n)$$

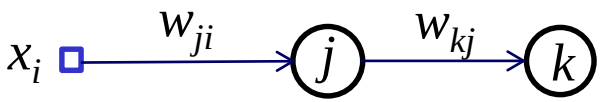
where

$$\delta_j(n) = \varphi'(v_j(n))\sum_k w_{kj}(n)\delta_k(n)$$

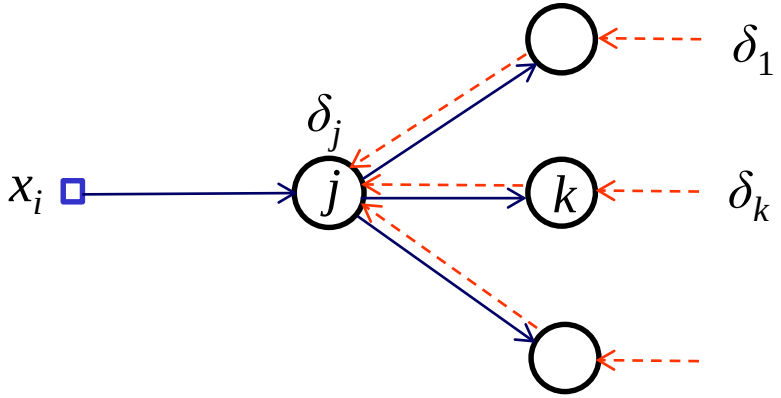
The above is called the generalized δ rule

- textbook correction

Backprop (cont.)

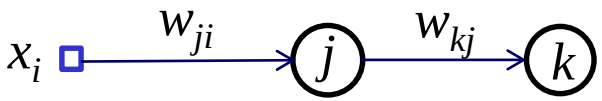


- Illustration of the generalized δ rule,



  - The generalized δ rule gives a solution to the credit (blame) assignment problem

Backprop (cont.)



- For the logistic sigmoid activation, we have

$$\varphi'(v) = a\varphi(v)[1-\varphi(v)]$$

hence

$$\delta_k(n) = e_k(n)[ay_k(n)(1-y_k(n))]$$

$$= ay_k(n)[1-y_k(n)][d_k(n)-y_k(n)]$$

$$\delta_j(n) = ay_j(n)[1-y_j(n)]\sum_k w_{kj}(n)\delta_k(n)$$

Backprop (cont.)

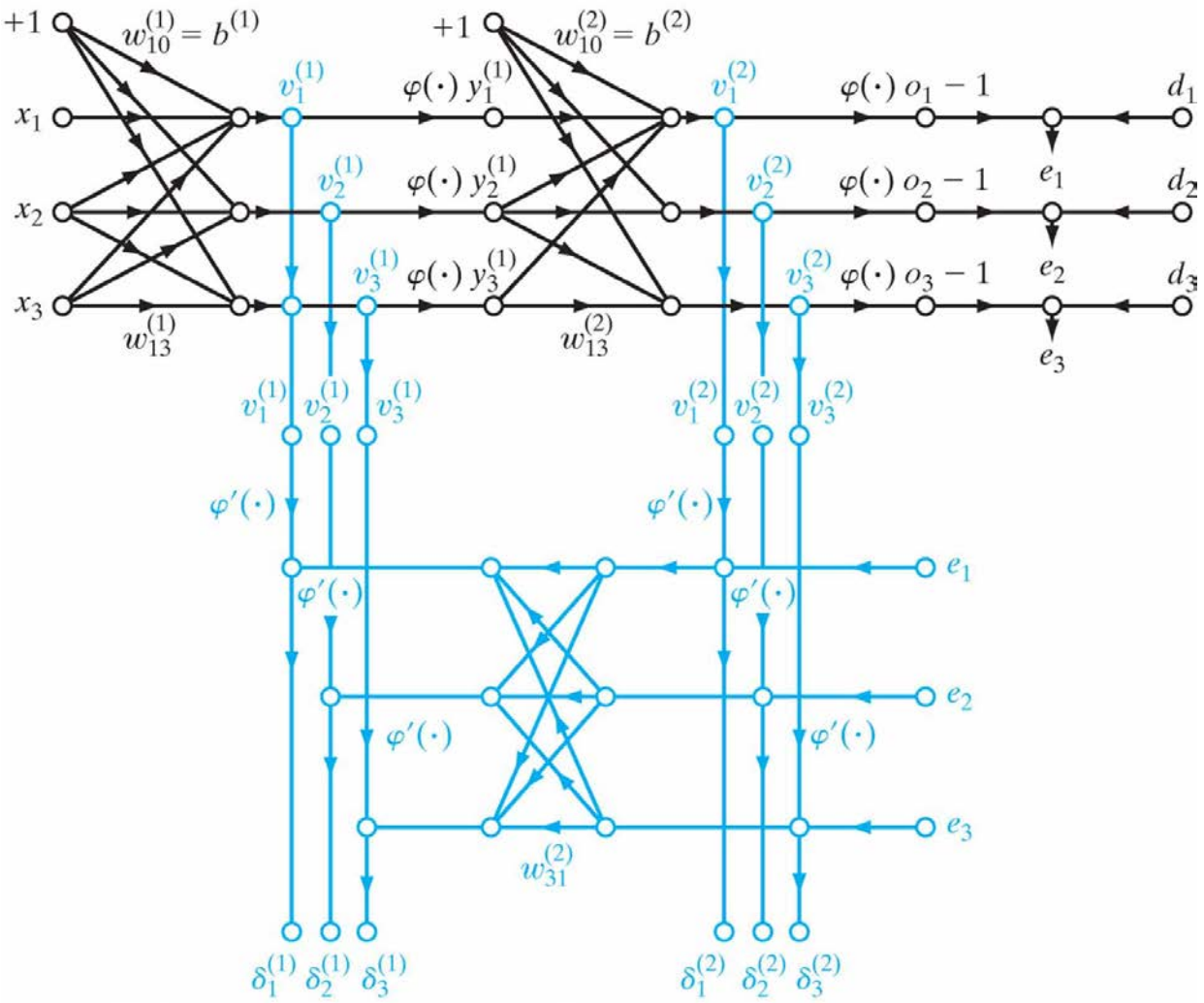
- Extension to more hidden layers is straightforward. In general we have

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n)$$

The δ rule applies to the output layer and the generalized δ rule applies to hidden layers, layer by layer from the output end.

The entire procedure is called backprop (error is back propagated)

Backprop illustration



Learning rate control: momentum

- To ease oscillating weights due to large η, some inertia (momentum) of weight update is added

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n) + \alpha \Delta w_{ji}(n-1), \qquad 0 < \alpha < 1$$

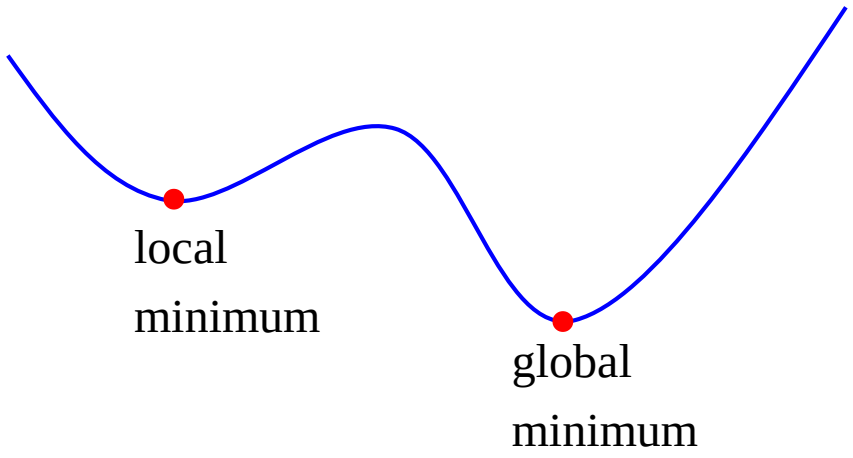
  - In the downhill situation, $\Delta w_{ji}(n) \approx \frac{\eta}{1-\alpha} \delta_j(n) y_i(n)$

    thus accelerating learning by a factor of $1/(1-\alpha)$

  - In the oscillating situation, it smoothes weight change, thus stabilizing oscillations

Remarks on backprop

- Backprop learning is local, concerning “presynaptic” and “postsynaptic” neurons only
- Local minima are possible due to nonlinearity, but infrequent due to online update and the practice of randomizing the order of presentations in different epochs



Remarks (cont.)

- MLP can learn to approximate any function, given sufficient layers and neurons (an existence proof)
- At most two hidden layers are sufficient to approximate any function. One hidden layer is sufficient for any continuous function
- Gradient descent is a simple method for cost optimization. More advanced methods exist, such as conjugate gradient and quasi-Newton methods (see textbook)
- Much design is still needed, such as the number of hidden layers, number of neurons for each layer, initial weights, learning rate, stopping criteria, etc.

MLP Design

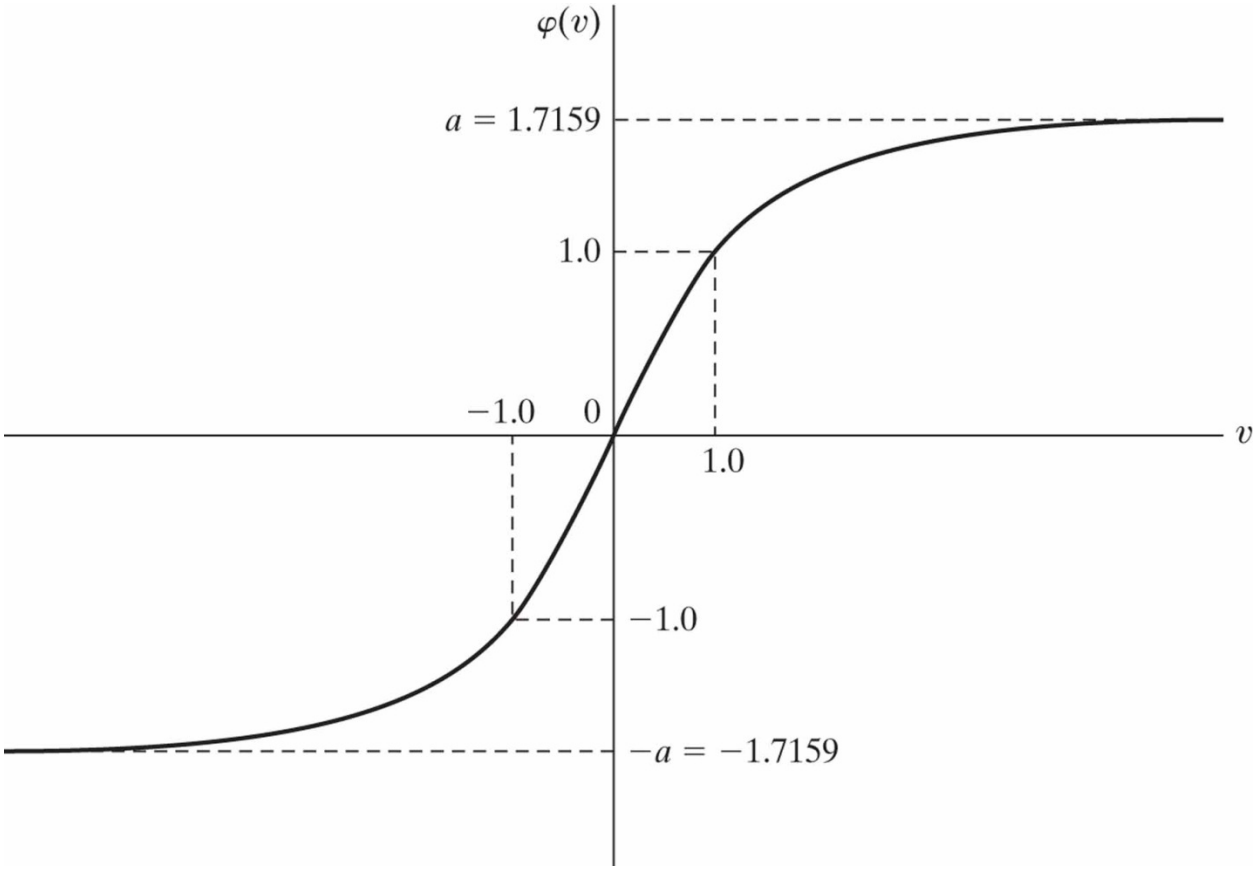
- Besides the use of momentum, learning rate annealing can be applied as for the LMS algorithm

- Batch update versus online update
  - For batch update, weight adjustment occurs after one presentation of all training patterns (one epoch)
  - For online update, weight adjustment occurs after the presentation of each pattern
  - For backprop, online learning is commonly used

MLP Design (cont.)

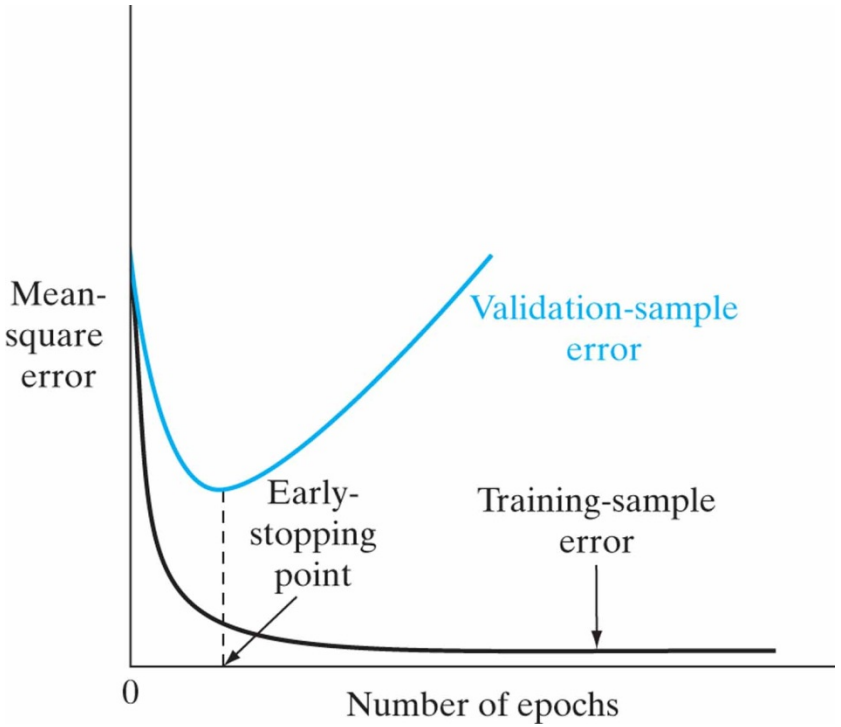
- Weight initialization: To prevent saturating neurons and break symmetry that can stall learning, initial weights (including biases) are typically randomized to produce zero mean and activation potentials away from saturation parts of the activation function
  - For the hyperbolic tangent activation function, avoiding saturation can be achieved by initializing weights so that the variance equals the reciprocal of the number of weights of a neuron

Hyperbolic tangent function



When to stop learning

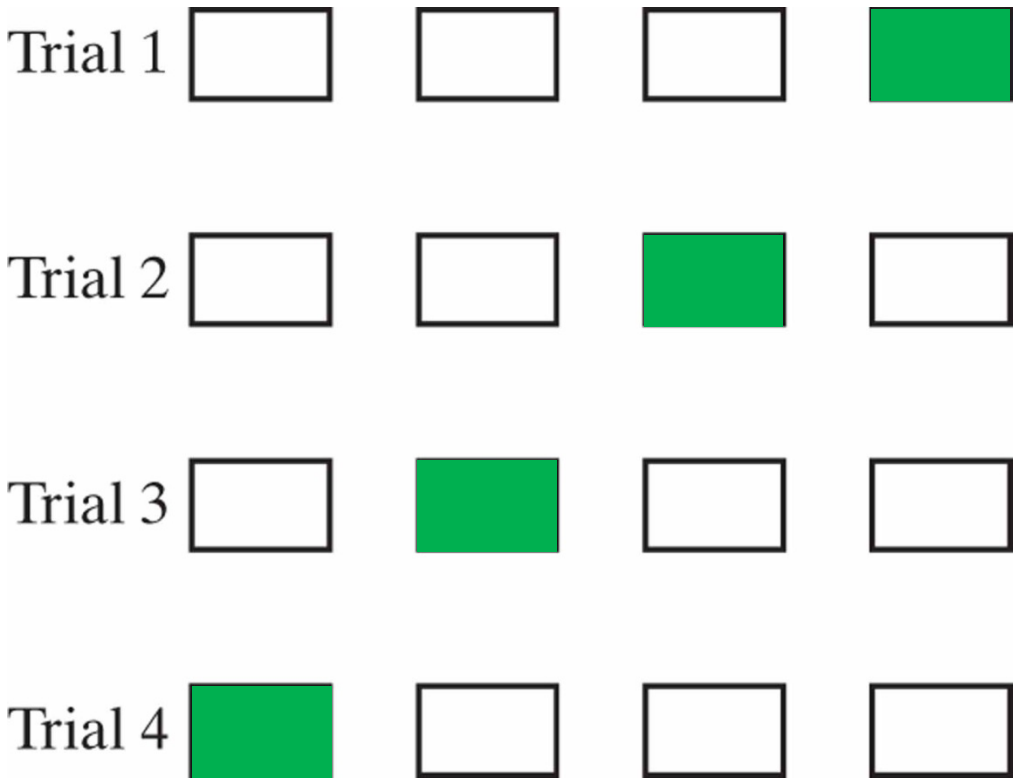
- One could stop after a predetermined number of epochs or when the MSE decrease is below a given criterion
- Early stopping with cross validation: keep part of the training set, called validation subset, as a test for generalization performance



Cross validation

- To do it systematically
  - Divide the entire training sample into an estimation subset and a validation subset (e.g. 80-20 split)
  - Every few epochs (e.g. 5), check validation error on the validation subset
  - Stop training when validation error starts to increase
- To avoid systematic bias, validation error can be measured repeatedly for different validation subsets, taken as the average
  - See next slide

Cross validation illustration



Cross validation (cont.)

- Cross validation is widely used as a general approach for model selection. Hence it can also be used to decide on the number of hidden layers and the number of neurons for a given hidden layer

MLP applications

- Task: Handwritten zipcode recognition (1989)
- Network description
  - Input: binary pixels for each digit
  - Output: 10 digits
  - Architecture: 4 layers (16x16–12x8x8–12x4x4–30–10)
- Each feature detector encodes only one feature within a local input region. Different detectors in the same module respond to the same feature at different locations through weight sharing. Such a layout is called a convolutional net

Zipcode recognizer architecture

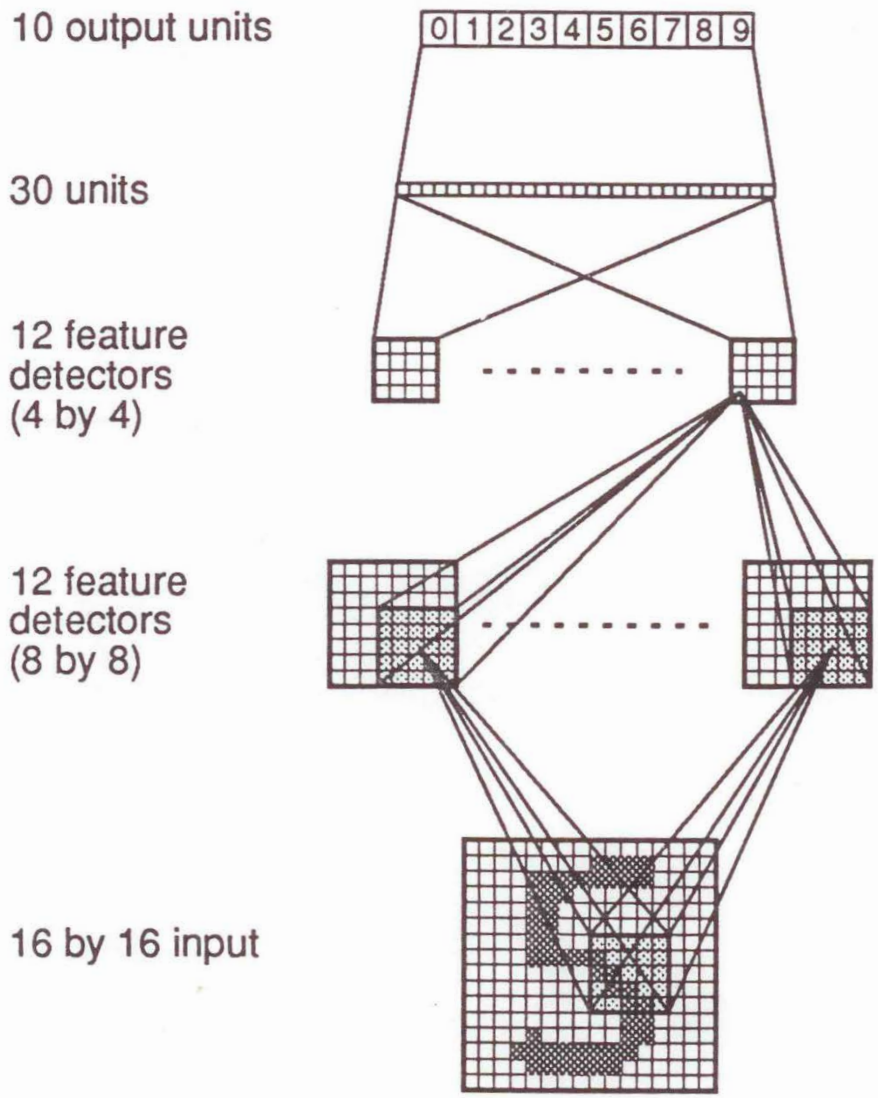


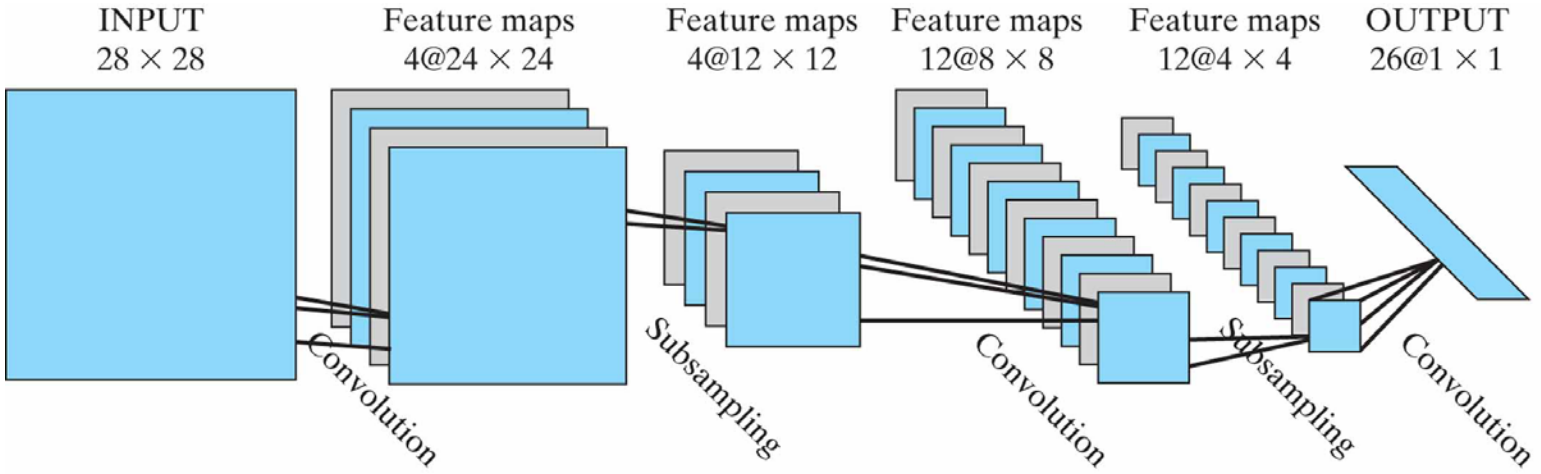
FIGURE 6.10 Archi
of the ZIP-code re
network.

Zipcode recognition (cont.)

- Performance: trained on 7300 digits and tested on 2000 new ones
  - Achieved 1% error on the training set and 5% error on the test set
  - If allowing rejection (no decision), 1% error on the test set
  - The task is not easy (see a handwriting example)
- **Remark**: constraining network design is a way of incorporating prior knowledge about a specific problem
  - Backprop applies whether or not the network is constrained

Letter recognition example

- The convolutional net has been subsequently applied to a number of pattern recognition tasks with state-of-the-art results
  - Handwritten letter recognition



Automatic driving

- ALVINN (automatic land vehicle in a neural network)

  - One hidden layer, one output layer
  - Five hidden nodes, 32 output nodes (steer left – steer right)
  - 960 inputs (30 x 32 image intensity array)
  - 5000 trainable weights
- Later success of Stanley (won $2M Darpa Grand Chellenge in 2005)

Other MLP applications

- NETtalk, a speech synthesizer
- GloveTalk, which converts hand gestures to speech

CSE 5526: Introduction to Neural Networks

Radial Basis Function (RBF) Networks

Function approximation

- MLP is both a pattern classifier and a function approximator
- As a function approximator, MLP is nonlinear, semiparametric, and universal

Function approximation background

- Weierstrass theorem: any continuous real function in an interval can be approximated arbitrarily well by a set of polynomials
- Taylor expansion approximates any differentiable function by polynomials in a neighborhood of a point
- Fourier series gives a way of approximating any periodic function by a sum of sine's

Linear projection

- Approximate function $f(\mathbf{x})$ by a linear combination of simpler functions

$$F(\mathbf{x}) = \sum_j w_j \varphi_j(\mathbf{x})$$

- If w_j's can be chosen so that approximation error is arbitrarily small for any function $f(\mathbf{x})$ over the domain of interest, $\{\varphi_j\}$ has the property of universal approximation, or $\{\varphi_j\}$ is complete

Example bases

- $\operatorname{sinc} x = \dfrac{\sin(x)}{x}$

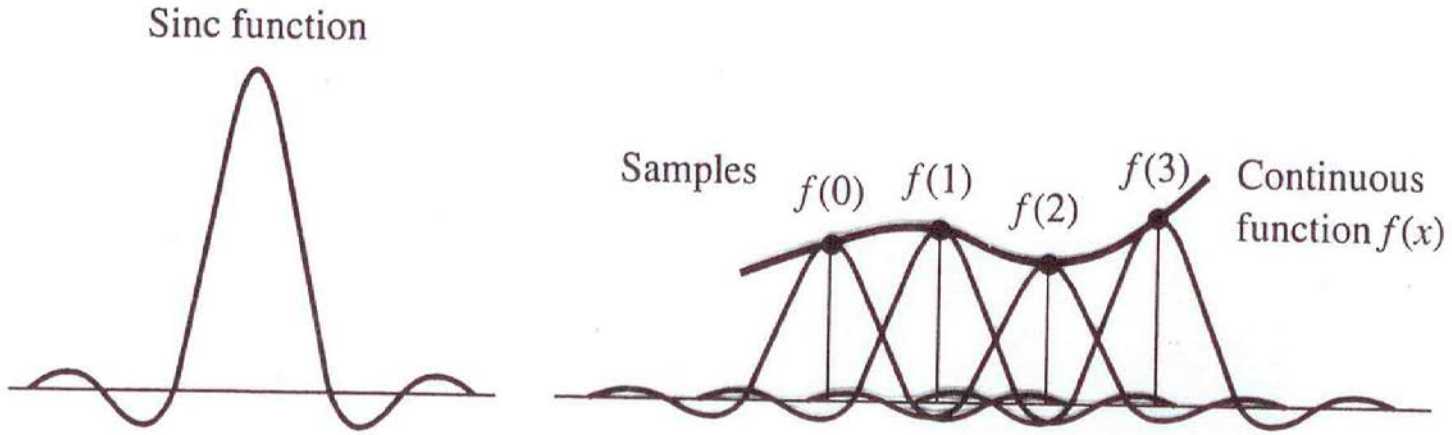


FIGURE 5-4 Decomposition by sinc functions

Example bases (cont.)

- sine function

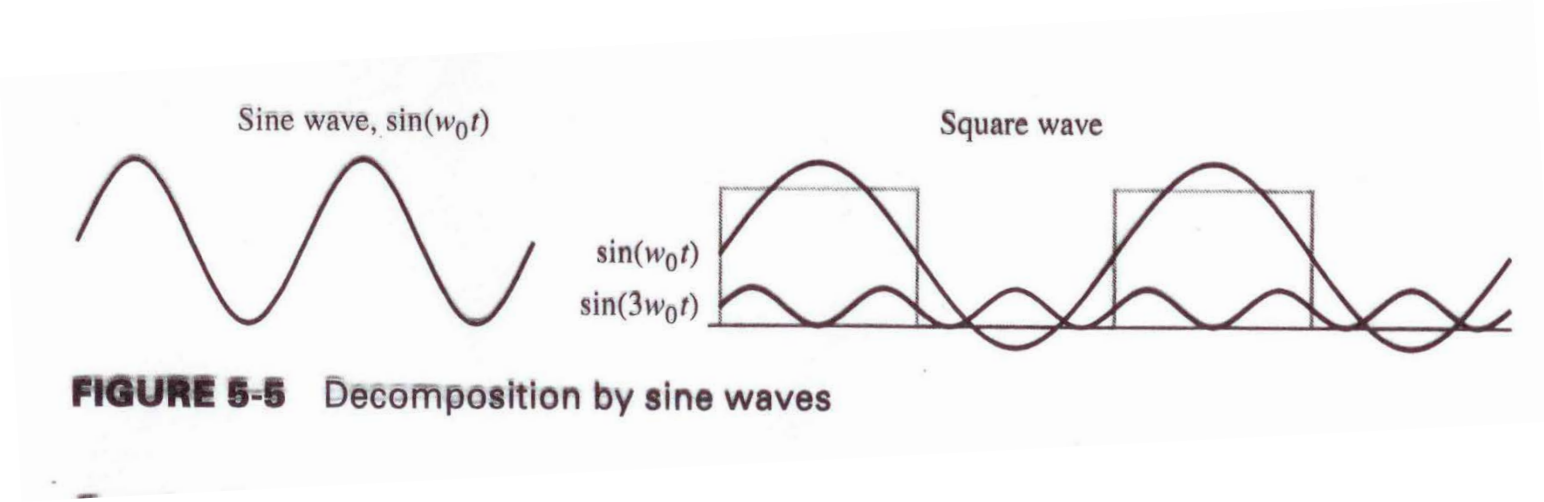


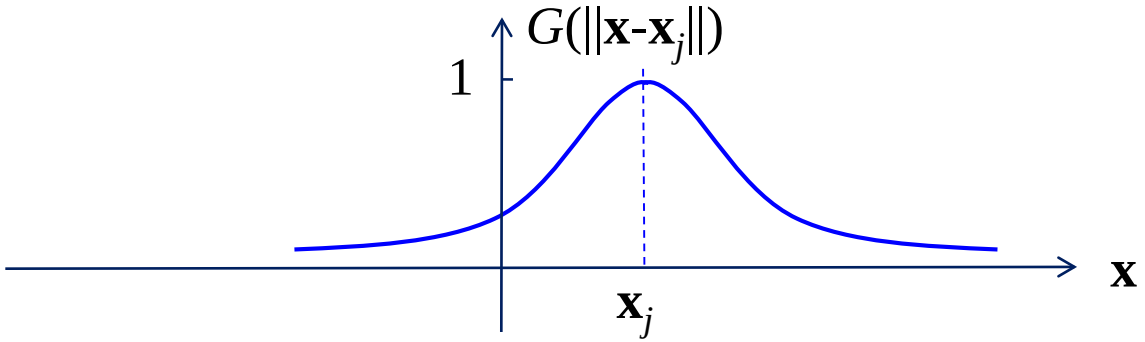
FIGURE 5-5 Decomposition by sine waves

Radial basis functions

- Consider

$$\varphi_j(\mathbf{x}) = \exp\left(-\frac{1}{2\sigma^2}||\mathbf{x}-\mathbf{x}_j||^2\right)$$
$$= G(||\mathbf{x}-\mathbf{x}_j||)$$

- A Gaussian is a local basis function, falling off exponentially from the center

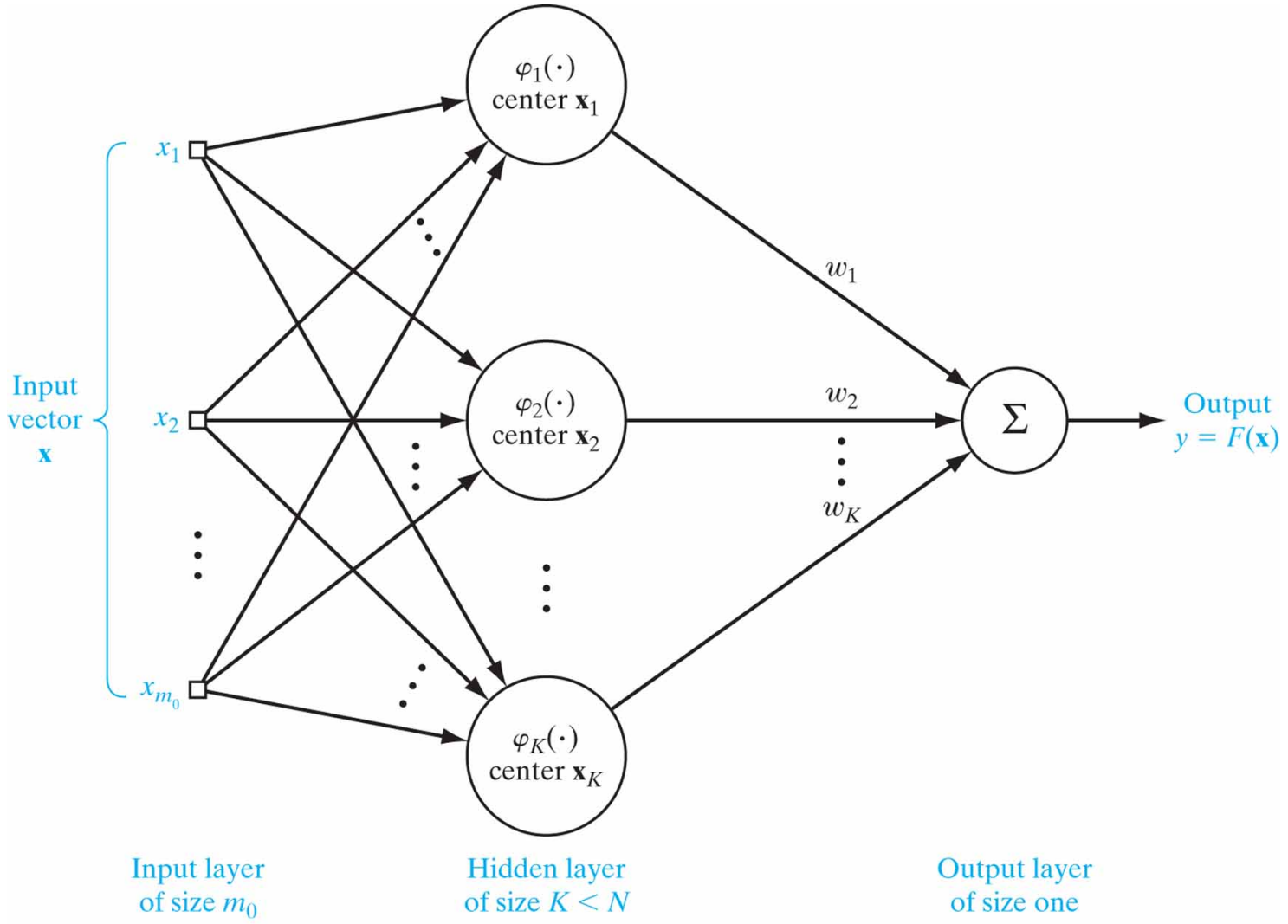


Radial basis functions (cont.)

- Thus approximation by RBF becomes

$$F(\mathbf{x}) = \sum_j w_j \, G(||\mathbf{x} - \mathbf{x}_j||)$$

RBF approximation illustration



Remarks

- Gaussians are universal approximators

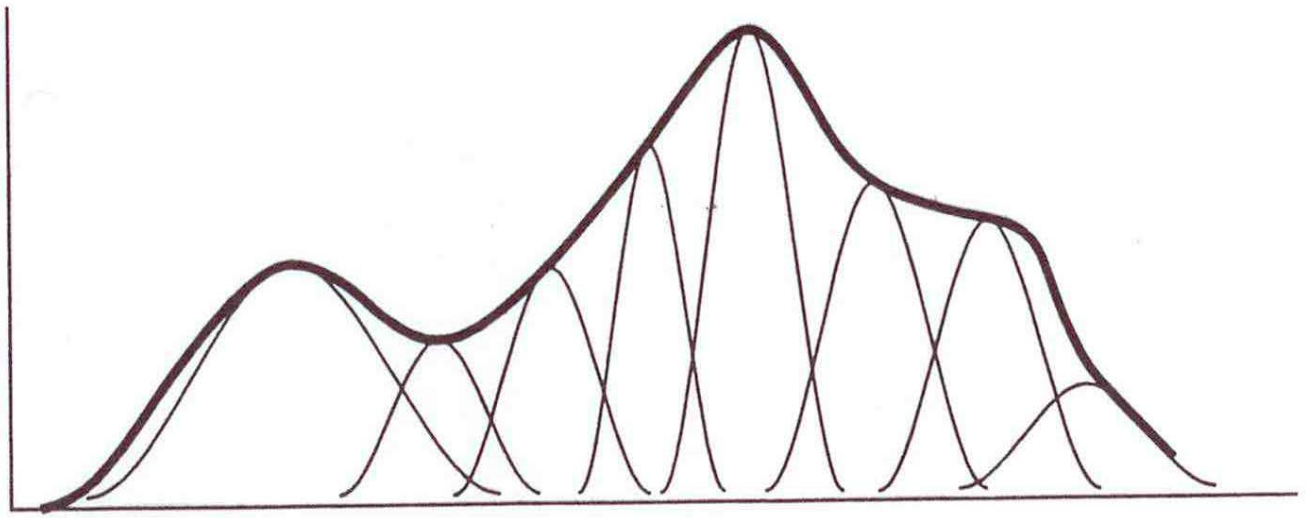


FIGURE 5-8 Approximation by RBFs in one dimension

Remarks (cont.)

- Such a radial basis function is called a kernel, a term from statistics
  - As a result, RBF nets are a kind of kernel methods
- Other RBFs exist, such as multiquadrics (see textbook)

Four questions to answer for RBF nets

- How to identify Gaussian centers?
- How to determine Gaussian widths?
- How to choose weights w_j's?
- How to select the number of bases?

Gaussian centers

- Identify Gaussian centers via unsupervised clustering: *K*-means algorithm
- Goal of the *K*-means algorithm: Divide *N* input patterns into *K* clusters so as to minimize the final variance. In other words, partition patterns into *K* clusters C_j's to minimize the following cost function

$$J=\sum_{j=1}^{K}\sum_{i\in C_j}||\mathbf{x}_i-\mathbf{u}_j||^2$$

where $\mathbf{u}_j=\frac{1}{||C_j||}\sum_{i\in C_j}\mathbf{x}_i$ is the mean (center) of cluster *j*

K-means algorithm

1. Choose a set of *K* cluster centers randomly from the input patterns
2. Assign the *N* input patterns to the *K* clusters using the squared Euclidean distance rule:

$\mathbf{x}$ is assigned to C_j if $||\mathbf{x}-\mathbf{u}_j||^2 \leq ||\mathbf{x}-\mathbf{u}_i||^2$ for $i \neq j$

K-means algorithm (cont.)

3. Update cluster centers

$$\mathbf{u}_j = \frac{1}{||C_j||} \sum_{i \in C_j} \mathbf{x}_i$$

4. If any cluster center changes, go to step 2; otherwise stop

- **Remark:** The *K*-means algorithm always converges, but the global minimum is not assured

Calculating Gaussian widths

- Once cluster centers are determined, the variance within each cluster can be set to

$$\sigma_j^2 = \frac{1}{||C_j||} \sum_{i \in C_j} ||\mathbf{u}_j - \mathbf{x}_i||^2$$

  - **Remark**: to simplify the RBF net design, different clusters often assume the same Gaussian width:

$$\sigma = \frac{d_{\max}}{\sqrt{2K}}$$

where $d_{\max}$ is the maximum distance between cluster centers

Weight update

- With the hidden layer decided, weight training can be treated as a linear regression problem, and the LMS algorithm is an efficient way for weight update
  - Note that a bias term needs to be included

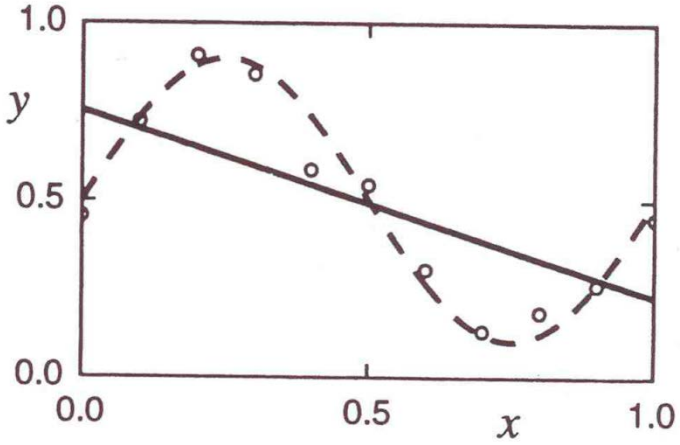
Selection of the number of bases: bias-variance dilemma

- The same problem as that of selecting the size of an MLP for classification
- The problem of overfitting
  - Example: Consider polynomial curve fitting

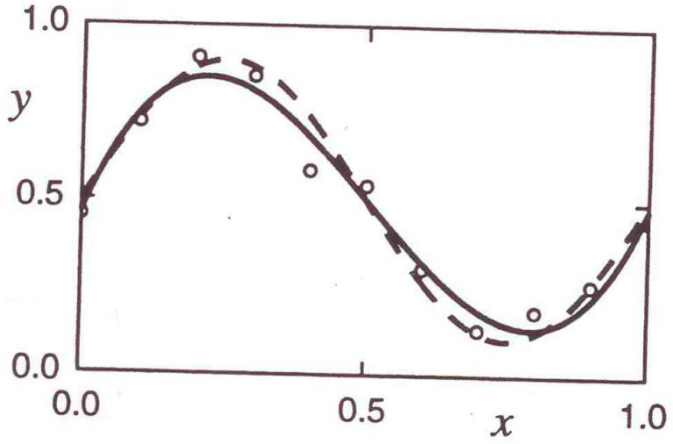
$$F(x)=\sum_{j=0}^{M} w_j x^j$$

for $f(x) = 0.5 + 0.4\sin(2\pi x)$ by an M-order F

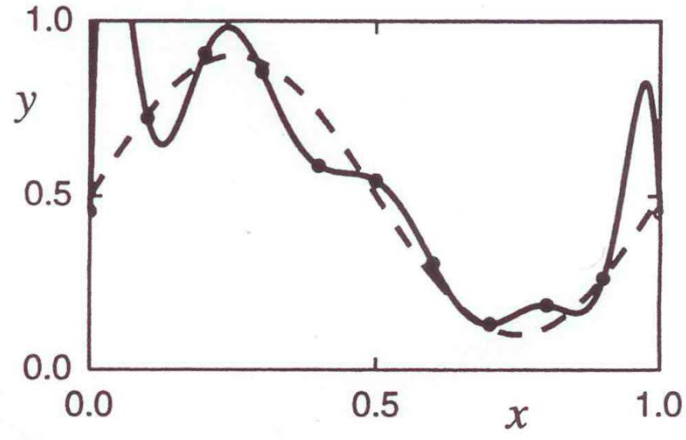
Overfitting: $F(x) = \sum_{j=0}^{M} w_j x^j$



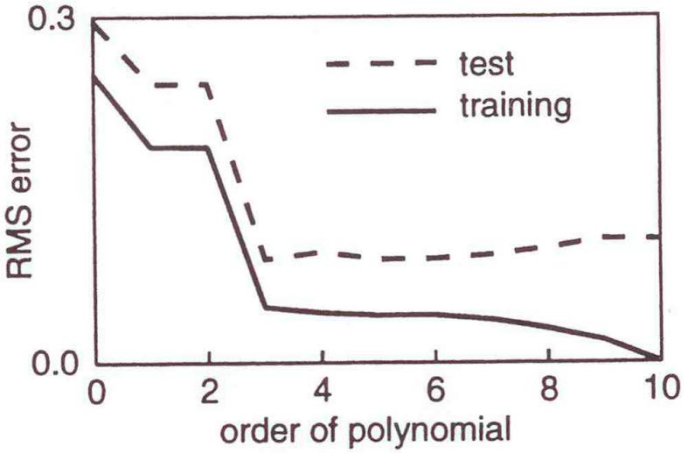
$M = 1$



$M = 3$



$M = 10$



Occam's razor

- The best scientific model is the simplest that is consistent with the data
  - In our case, it translates to the principle that a learning machine should be large enough to approximate the data well, but not larger
- Occam's razor is a general principle governing supervised learning and generalization

Bias and variance

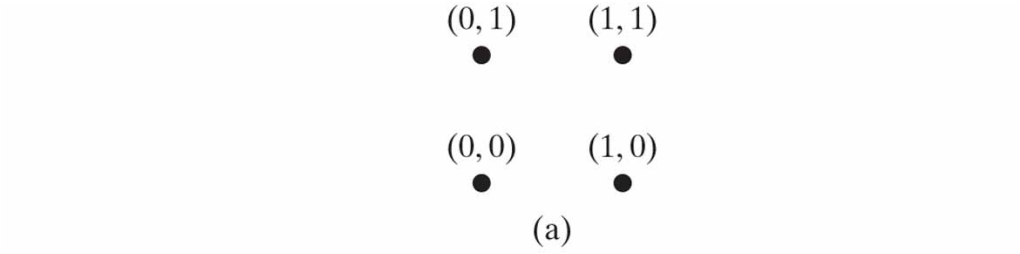
- Bias: training error – difference between desired output and actual output for a particular training sample
- Variance: generalization error – difference between the learned function from a particular training sample and the function derived from all training samples
  - Example: two extreme cases: zero bias and zero variance

Bias and variance (cont.)

- The optimal size of a learning machine is thus a compromise between the bias and the variance of a model
  - In other words, a good-sized model is the one where both bias and variance are low
- For RBF nets, in practice, cross validation can be applied to select the number of bases, where validation error is measured for a series of numbers of bases

XOR problem, again

- RBF nets can also be applied to pattern classification problems
  - XOR problem revisited



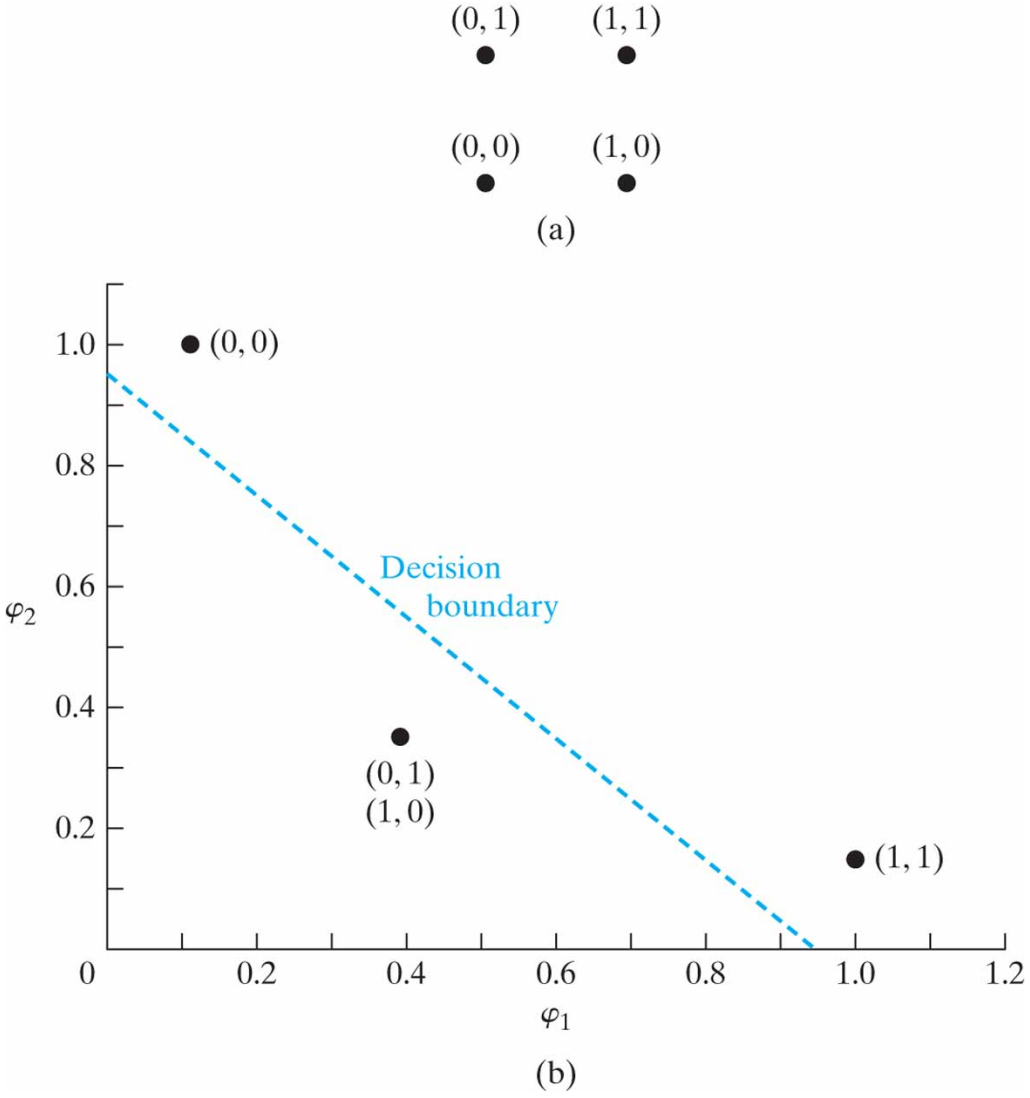
XOR problem (cont.)

TABLE 5.1 Specification of the Hidden Functions for the XOR Problem of Example 1

Input Pattern $\mathbf{x}$	First Hidden Function $\varphi_1(\mathbf{x})$	Second Hidden Function $\varphi_2(\mathbf{x})$
(1,1)	1	0.1353
(0,1)	0.3678	0.3678
(0,0)	0.1353	1
(1,0)	0.3678	0.3678

XOR problem, again

- RBF nets can also be applied to pattern classification problems
  - XOR problem revisited



Comparison between RBF and MLP

- For RBF nets, bases are local, while for MLP, “bases” are global
- Generally, more bases are needed than hidden units in MLP
- Training is more efficient for RBF nets

CSE 5526: Introduction to Neural Networks

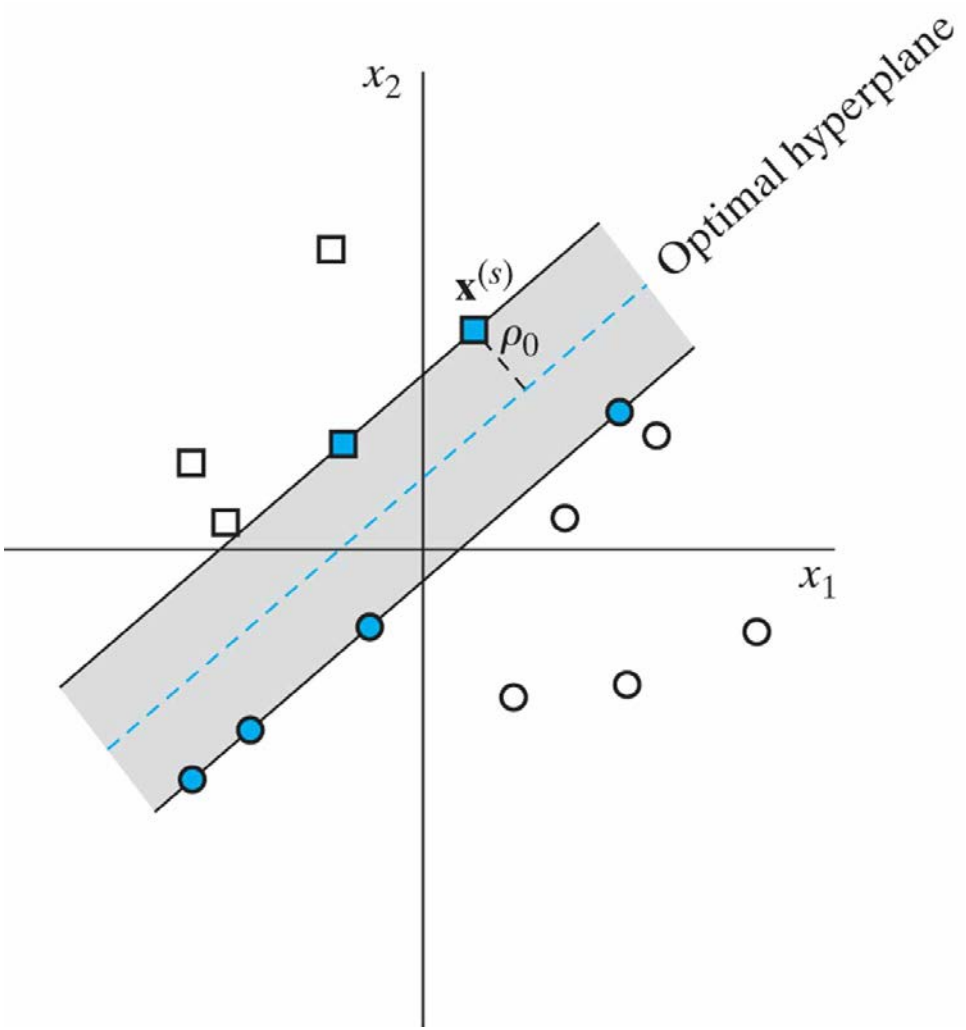
Support Vector Machines (SVM)

Motivation

- For a linearly separable classification task, there are generally infinitely many separating hyperplanes. Perceptron learning, however, stops as soon as one of them is reached

- To improve generalization, we want to place a decision boundary as far away from training classes as possible. In other words, place the boundary at equal distances from class boundaries

Optimal hyperplane

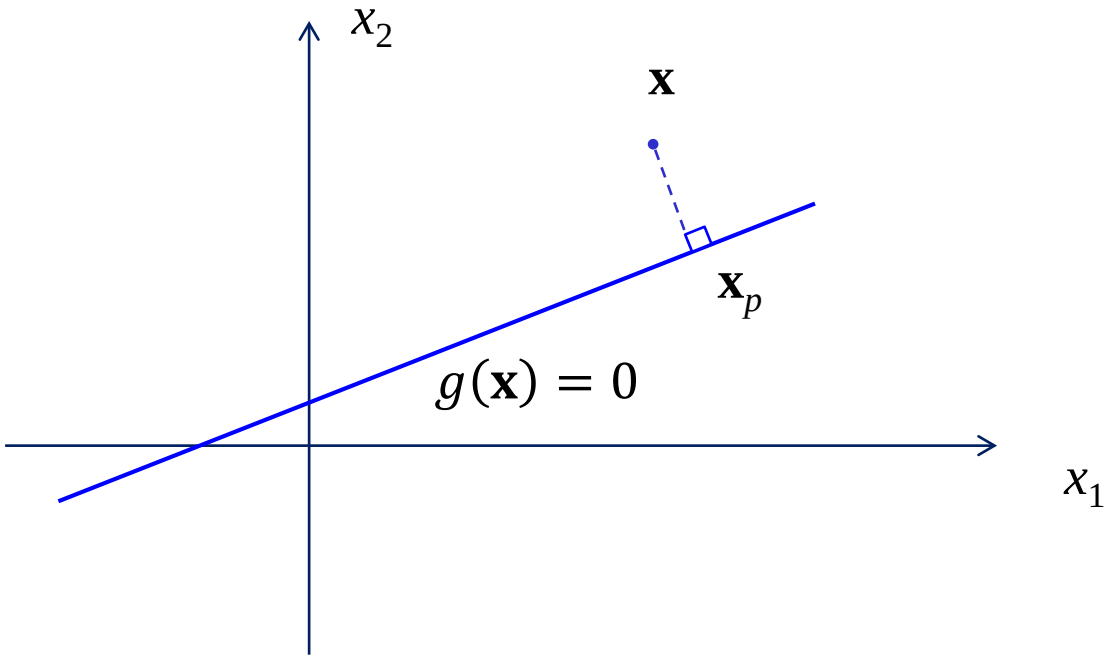


Decision boundary

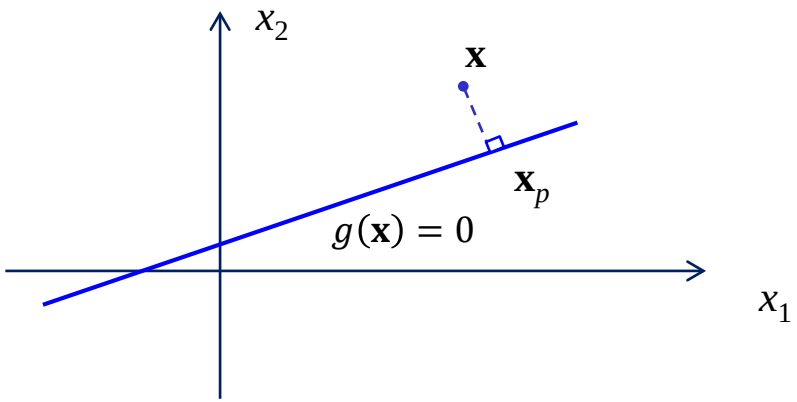
- Given a linear discriminant function

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = 0$$

- To find its distance to a given pattern **x**, project **x** onto the decision boundary:



Decision boundary (cont.)



$$\mathbf{x} = \mathbf{x}_p + r\frac{\mathbf{w}}{||\mathbf{w}||}$$

where $\mathbf{x}_p$ is $\mathbf{x}$'s projection and the second term arises from the fact that the weight vector is perpendicular to the decision boundary. The algebraic distance r is positive if $\mathbf{x}$ is on the positive side of the boundary and negative if $\mathbf{x}$ is on the negative side

Decision boundary (cont.)

Then

$$g(\mathbf{x}) = g(\mathbf{x}_p + r\frac{\mathbf{w}}{||\mathbf{w}||})$$
$$= \mathbf{w}^T(\mathbf{x}_p + r\frac{\mathbf{w}}{||\mathbf{w}||}) + b$$
$$= \mathbf{w}^T\mathbf{x}_p + b + r||\mathbf{w}||$$
$$= r||\mathbf{w}||$$

Decision boundary (cont.)

Thus

$$r = \frac{g(\mathbf{x})}{||\mathbf{w}||}$$

- As a special case, for the origin, $r = \frac{b}{||\mathbf{w}||}$, as discussed before

Margin of separation

- The margin of separation is the smallest distance of the hyperplane to a data set. Equivalently, the margin is the distance to the nearest data points
- The training patterns closest to the optimal hyperplane are called *support vectors*

Finding optimal hyperplane for linearly separable problems

- Question: Given N pairs of input and desired output $< \mathbf{x}_i, d_i >$, how to find $\mathbf{w}_o$ and b_o for the optimal hyperplane?
- Without loss of generality, $\mathbf{w}_o$ and b_o must satisfy

$$\mathbf{w}_o{}^T \mathbf{x}_i + b_o \geq 1 \quad \text{for } d_i = 1$$
$$\mathbf{w}_o{}^T \mathbf{x}_i + b_o \leq -1 \quad \text{for } d_i = -1$$

or $$d_i(\mathbf{w}_o{}^T \mathbf{x}_i + b_o) \geq 1$$

where the equality holds for support vectors only

Optimal hyperplane

- For a support vector $\mathbf{x}^{(s)}$, its algebraic distance to the optimal hyperplane:

$$r = \frac{g(\mathbf{x}^{(s)})}{||\mathbf{w}_o||} = \begin{cases} \dfrac{1}{||\mathbf{w}_o||} & \text{if } d^{(s)} = 1 \\ \dfrac{-1}{||\mathbf{w}_o||} & \text{if } d^{(s)} = -1 \end{cases}$$

- Thus the margin of separation:

$$|r| = \frac{1}{||\mathbf{w}_o||}$$

- In other words, maximizing the margin of separation is equivalent to minimizing $||\mathbf{w}_o||$

Primal problem

- Therefore $\mathbf{w}_o$ and b_o satisfy

$$d_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \quad \text{for } i = 1, \ldots, N$$

$$\text{and } \Phi(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} \text{ is minimized}$$

  - The above constrained minimization problem is called the *primal* problem

Lagrangian formulation

- Using Lagrangian formulation, we construct the Lagrangian function:

$$J(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{N} \alpha_i [d_i(\mathbf{w}^T\mathbf{x}_i + b) - 1]$$

where nonnegative variables, α_i's, are called Lagrange multipliers

Lagrangian formulation (cont.)

- The solution is a saddle point, minimized with respect to $\mathbf{w}$ and b, but maximized with respect to α

Condition 1:

$$\frac{\partial J(\mathbf{w}, b, \alpha)}{\partial \mathbf{w}} = \mathbf{0}$$

Condition 2:

$$\frac{\partial J(\mathbf{w}, b, \alpha)}{\partial b} = 0$$

Lagrangian formulation (cont.)

$$J(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{N} \alpha_i [d_i(\mathbf{w}^T\mathbf{x}_i + b) - 1]$$

- From condition 1:

$$\mathbf{w} - \sum_i \alpha_i d_i \mathbf{x}_i = \mathbf{0}$$

or $\mathbf{w} = \sum_i \alpha_i d_i \mathbf{x}_i$

- From condition 2:

$$\sum_i \alpha_i d_i = 0$$

Karush-Kuhn-Tucker conditions

$$J(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{N} \alpha_i [d_i(\mathbf{w}^T\mathbf{x}_i + b) - 1]$$

- **Remark**: The above constrained optimization problem satisfies:

$$\alpha_i [d_i(\mathbf{w}^T\mathbf{x}_i + b) - 1] = 0$$

called Karush-Kuhn-Tucker conditions

- In other words, $\alpha_i = 0$ when $d_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 > 0$
- α_i can be greater than 0 only when $d_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 = 0$, that is, for support vectors

How to find α?

$$J(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{N} \alpha_i [d_i(\mathbf{w}^T\mathbf{x}_i + b) - 1]$$

- The primal problem has a corresponding dual problem in terms of α. From the Lagrangian

$$J(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_i \alpha_i d_i \mathbf{w}^T\mathbf{x}_i - b\sum_i \alpha_i d_i + \sum_i \alpha_i$$

How to find α (cont.)

$$J(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_i \alpha_i d_i \mathbf{w}^T\mathbf{x}_i - b\sum_i \alpha_i d_i + \sum_i \alpha_i$$

- Because of the two conditions, the third term is zero and

$$\mathbf{w}^T\mathbf{w} = \sum_i \alpha_i d_i \mathbf{w}^T\mathbf{x}_i$$

Hence

$$Q(\alpha) = \sum_i \alpha_i - \frac{1}{2}\sum_i\sum_j \alpha_i\alpha_j d_i d_j \mathbf{x}_i{}^T\mathbf{x}_j$$

Dual problem

- The *dual* problem is stated as follows:

  The Lagrange multipliers maximize

$$Q(\alpha) = \sum_i \alpha_i - \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j d_i d_j \mathbf{x}_i{}^T \mathbf{x}_j$$

  subject to

$$(1)\ \textstyle\sum_i \alpha_i d_i = 0$$

$$(2)\ \alpha_i \geq 0$$

  - The dual problem can be solved as a quadratic optimization problem. Note that $\alpha_i > 0$ only for support vectors

Solution

- Having found optimal multipliers, $\alpha_{o,i}$

$$\mathbf{w}_o = \sum_{i=1}^{N_s} \alpha_{o,i} d_i \mathbf{x}_i$$

where N_s is the number of support vectors

Solution (cont.)

- For any support vector $\mathbf{x}^{(s)}$, we have

$$d^{(s)}\left(\mathbf{w}_o{}^T\mathbf{x}^{(s)} + b_o\right) = 1$$

$$b_o = \frac{1}{d^{(s)}} - \mathbf{w}_o{}^T\mathbf{x}^{(s)} = \frac{1}{d^{(s)}} - \sum_{i=1}^{N_s} \alpha_{o,i} d_i \mathbf{x}_i{}^T\mathbf{x}^{(s)}$$

  - Note that for robustness, one should average over all support vectors to compute b_o

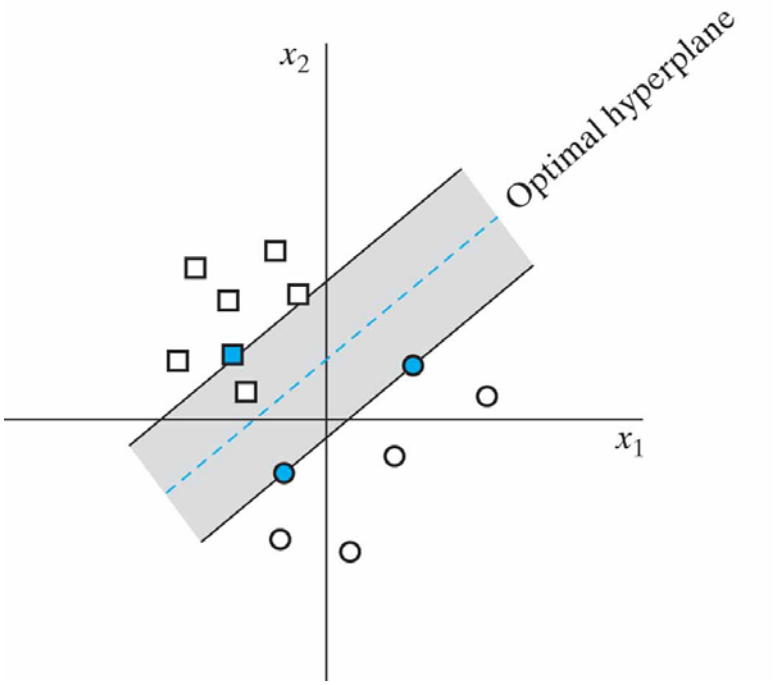
Linearly inseparable problems

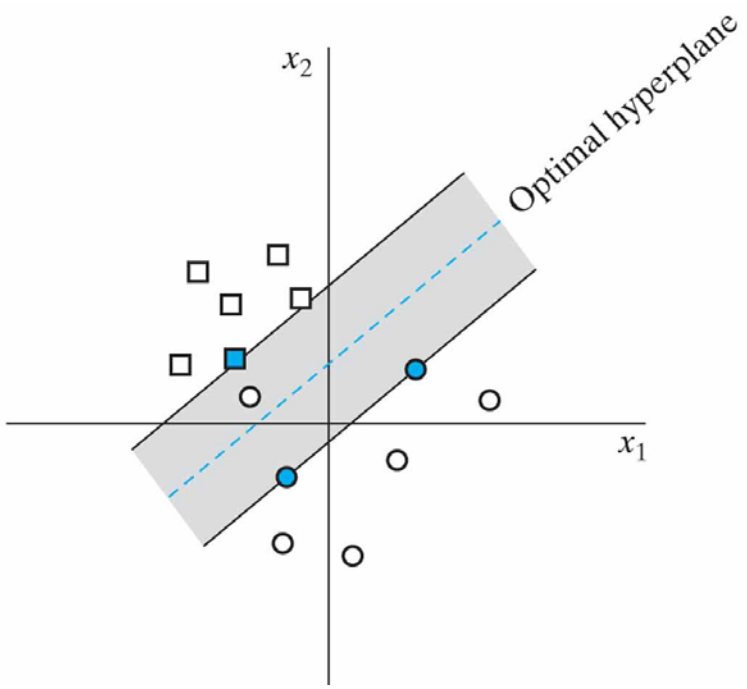
- How to find optimal hyperplanes for linearly inseparable cases?
- For such problems, the condition

$$d_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \quad \text{for } i = 1, \ldots, N$$

is violated. In such cases, the margin of separation is called soft.

Linearly inseparable cases





Slack variables

- To extend the constrained optimization framework, we introduce a set of nonnegative variables, $\xi_i (i = 1, \ldots, N)$, called slack variables, into the condition:

$$d_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i \quad \text{for } i = 1, \ldots, N$$

  - For $0 < \xi_i \leq 1$, $\mathbf{x}_i$ falls into the region of separation, but on the correct side of the decision boundary
  - For $\xi_i > 1$, $\mathbf{x}_i$ falls on the wrong side
  - The equality holds for support vectors, no matter whether $\xi_i > 0$ or $\xi_i = 0$. Thus, linearly separable problems can be treated as a special case

Classification error

- In general, the goal is to minimize the classification error:

$$\Phi(\xi) = \sum_i I(\xi_i - 1)$$

$$\text{where } I(\xi) = \begin{cases} 0 & \text{if } \xi \leq 0 \\ 1 & \text{if } \xi > 0 \end{cases}$$

Classification error

- To turn the above problem into a convex optimization problem with respect to **w** and b, we minimize instead

$$\Phi(\xi) = \sum_i \xi_i$$

Classification error (cont.)

- Adding the term to the minimization of $\|\mathbf{w}\|$, we have

$$\Phi(\mathbf{w}, \xi) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \xi_i$$

  - The parameter C controls the tradeoff between minimizing the classification error and maximizing the margin of separation. C has to be chosen by the user, reflecting the confidence on the training sample

Primal problem for linearly inseparable case

- The primal problem becomes:

  Find optimal $\mathbf{w}_o$ and b_o so that

$$d_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \quad \text{for } i = 1, \ldots, N$$
$$\xi_i \geq 0$$

  and

$$\Phi(\mathbf{w}, \xi) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \xi_i$$

  is minimized

Lagrangian formulation

- Again using Lagrangian formulation,

$$J(\mathbf{w}, b, \xi, \alpha, \mu)$$
$$= \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \xi_i - \sum_i \alpha_i\left[d_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 + \xi_i\right]$$
$$- \sum_i \mu_i\xi_i$$

with nonnegative Lagrange multipliers α_i's and μ_i's

Dual problem for linearly inseparable case

- By a similar derivation, the dual problem is to find α_i's to maximize

$$Q(\alpha) = \sum_i \alpha_i - \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j d_i d_j \mathbf{x}_i{}^T \mathbf{x}_j$$

subject to

$$(1)\ \textstyle\sum_i \alpha_i d_i = 0$$

$$(2)\ 0 \le \alpha_i \le C$$

Solution

- The dual problem contains neither ξ_i nor μ_i, and is the same as for the linearly separable case, except for the more stringent constraint $0 \leq \alpha_i \leq C$. So it can be solved as a quadratic optimization problem

Solution (cont.)

- With optimal multipliers found,

$$\mathbf{w}_o = \sum_{i=1}^{N_s} \alpha_{o,i} d_i \mathbf{x}_i$$

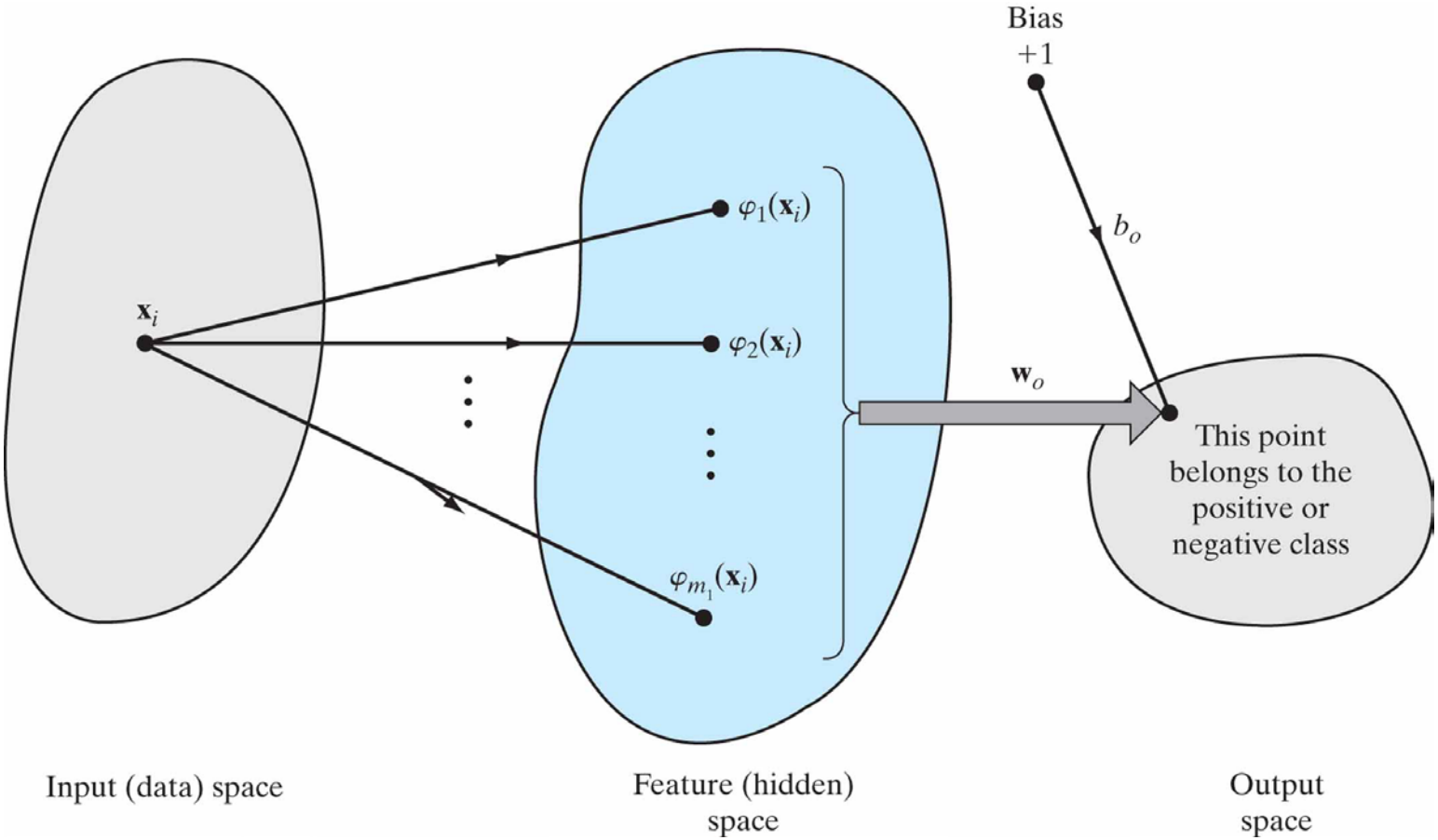
- Due to the Karush-Kuhn-Tucker conditions, for any $0 < \alpha_i < C$, ξ_i must be zero, corresponding to a support vector. Hence b_o can be computed in the same way as for linearly separable cases for such α_i

SVM as a kernel machine

- **Cover's theorem**: A complex classification problem, cast in a high-dimensional space nonlinearly, is more likely to be linearly separable than in the low-dimensional input space

- SVM for pattern classification
  1. Nonlinear mapping of the input space onto a high-dimensional feature space
  2. Constructing the optimal hyperplane for the feature space

Kernel machine illustration



Inner product kernel

- Let $\varphi_j(\mathbf{x}), j = 1, \ldots, \infty$, denote a set of nonlinear mapping functions onto the feature space
- Without loss of generality, set $b = 0$. For a given weight vector $\mathbf{w}^T$, the discriminant function is

$$\sum_{j=1}^{\infty} w_j \varphi_j(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}) = 0$$

Inner product kernel (cont.)

- Treating the feature space as input to an SVM, we have

$$\mathbf{w} = \sum_{i=1}^{N_s} \alpha_i d_i \boldsymbol{\phi}(\mathbf{x}_i)$$

- Then the optimal hyperplane becomes

$$\sum_{i=1}^{N_s} \alpha_i d_i \boldsymbol{\phi}^T(\mathbf{x}_i)\boldsymbol{\phi}(\mathbf{x}) = 0$$

Optimal hyperplane

- Denote the inner product of the mapping functions as

$$k(\mathbf{x}, \mathbf{x}_i) = \boldsymbol{\phi}^T(\mathbf{x}_i)\boldsymbol{\phi}(\mathbf{x})$$

$$= \sum_{j=1}^{\infty} \varphi_j(\mathbf{x}_i)\varphi_j(\mathbf{x}), \quad i = 1, \ldots, N$$

- Then we have

$$\sum_{i=1}^{N_s} \alpha_i d_i k(\mathbf{x}, \mathbf{x}_i) = 0$$

Kernel trick

- Function $k(\mathbf{x}, \mathbf{x}_i)$ is called an inner-product kernel, satisfying the condition $k(\mathbf{x}, \mathbf{x}_i) = k(\mathbf{x}_i, \mathbf{x})$

- Kernel trick: For pattern classification in the output space, specifying the kernel $k(\mathbf{x}, \mathbf{x}_i)$ is sufficient. That is, no need to train $\mathbf{w}$

Kernel matrix

- The matrix

$$\mathbf{K} = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \cdots & k(\mathbf{x}_1, \mathbf{x}_N) \\ \vdots & & \\ \ldots & k(\mathbf{x}_i, \mathbf{x}_j) & \ldots \\ \vdots & & \\ k(\mathbf{x}_N, \mathbf{x}_1) & \ldots & k(\mathbf{x}_N, \mathbf{x}_N) \end{bmatrix}$$

is called the kernel matrix, or the Gram metrix. **K** is positive, semidefinite

Remarks

- Even though the feature space could be of infinite dimensionality, the optimal hyperplane for classification has a finite number of terms that is equal to the number of support vectors in the feature space
- Mercer's theorem (see textbook) specifies the conditions for a candidate kernel to be an inner-product kernel, admissible for SVM

SVM design

- Given N pairs of input and desired output $< \mathbf{x}_i, d_i >$, find the Lagrange multipliers, $\alpha_1, \alpha_2, \ldots, \alpha_N$, by maximizing the objective function:

$$Q(\alpha) = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j d_i d_j k(\mathbf{x}_i, \mathbf{x}_j)$$

subject to

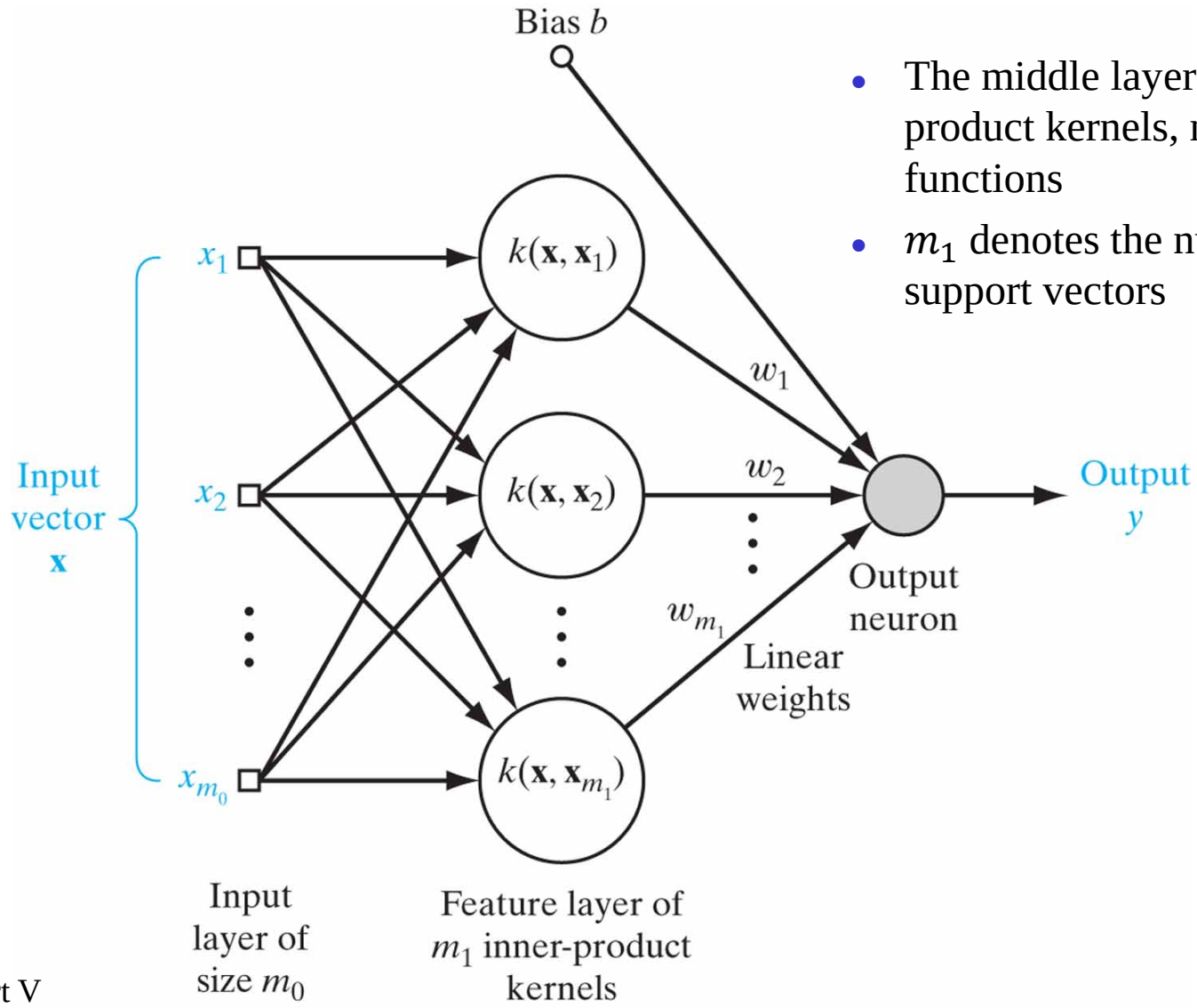
$$(1)\ \textstyle\sum_i \alpha_i d_i = 0$$

$$(2)\ 0 \leq \alpha_i \leq C$$

SVM design (cont.)

- The above dual problem has the same form as for linearly inseparable problems except for the substitution of $k(\mathbf{x}_i, \mathbf{x}_j)$ for $\mathbf{x}_i{}^T \mathbf{x}_j$

Optimal hyperplane



- The middle layer depicts inner-product kernels, not mapping functions
- m_1 denotes the number of support vectors

Typical kernels

1. Polynomial kernel:

$$(\mathbf{x}^T\mathbf{x}_i + 1)^p$$

- p is a parameter

2. RBF kernels:

$$\exp(-\frac{1}{2\sigma^2}||\mathbf{x} - \mathbf{x}_i||^2)$$

- σ is a parameter common to all kernels

Typical kernels (cont.)

3. Hyperbolic tangent kernel (two-layer perceptron):

$$\tanh(\beta_0 \mathbf{x}^T \mathbf{x}_i + \beta_1)$$

- only certain values of β_0 and β_1 satisfy Mercer's theorem

- Note that SVM theory avoids the need for heuristics in RBF and MLP design, and guarantees a measure of optimality

Example: XOR problem again

- See blackboard

TABLE 6.2 XOR Problem

Input vector $\mathbf{x}$	Desired response d
$(-1, -1)$	-1
$(-1, +1)$	$+1$
$(+1, -1)$	$+1$
$(+1, +1)$	-1

SVM summary

- SVM builds on strong theoretical foundations, eliminating the need for much user design
- SVM produces very good results for classification, and is the kernel method of choice
- Computational complexity (both time & memory) increases quickly with the size of the training sample

CSE 5526: Introduction to Neural Networks

Unsupervised Learning

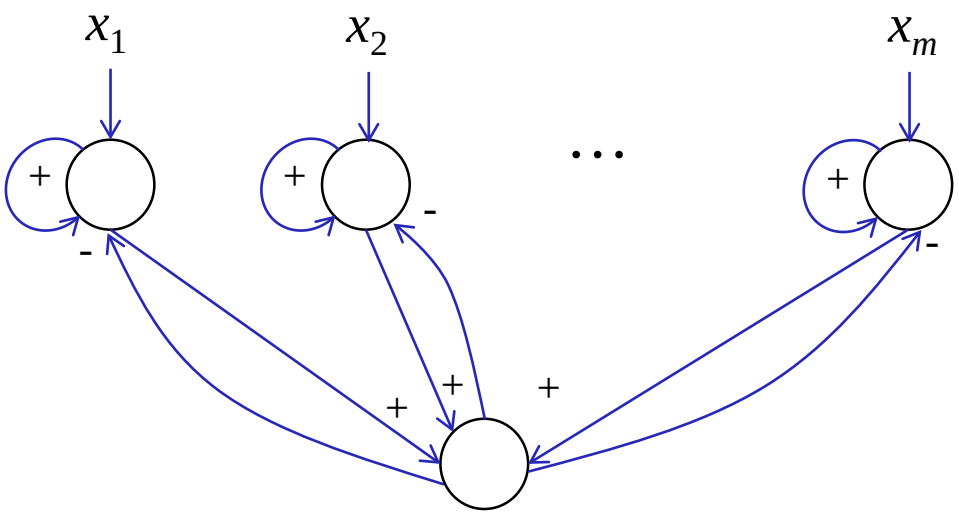
Types of learning

- Supervised learning: Detailed desired output is provided externally
- Reinforcement learning: Evaluative output is provided externally
  - It is sometimes considered a form of supervised learning (reward/penalty)
- Unsupervised learning, comprising competitive learning and self organization

Competitive dynamics

- Winner-take-all (WTA) networks implement competitive dynamics
- Two different architectures of WTA

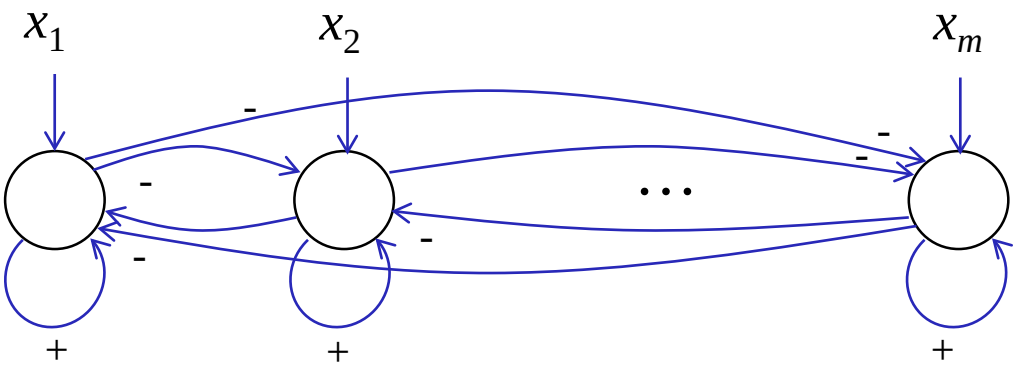
  Global inhibition:



WTA networks

- Two different architectures of WTA

  Mutual inhibition

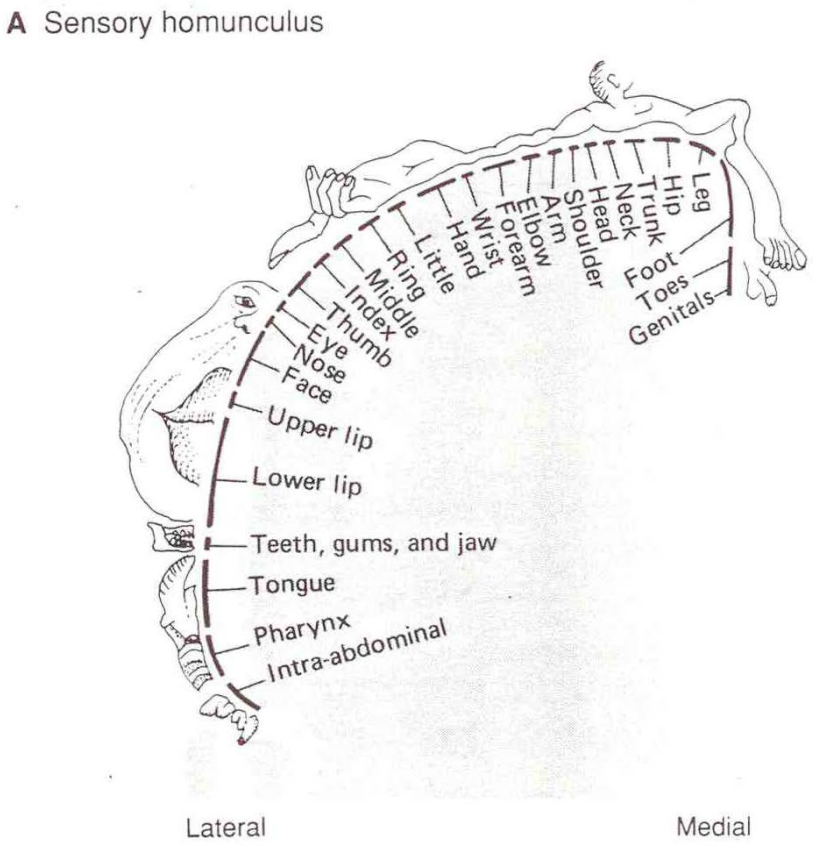


WTA networks (cont.)

- External input sets the initial conditions of neurons. Under certain conditions, only the neuron with the largest input reaches 1 and all the other neurons reach 0
  - Thus WTA is a maximum selector, a parallel implementation of MAX operation

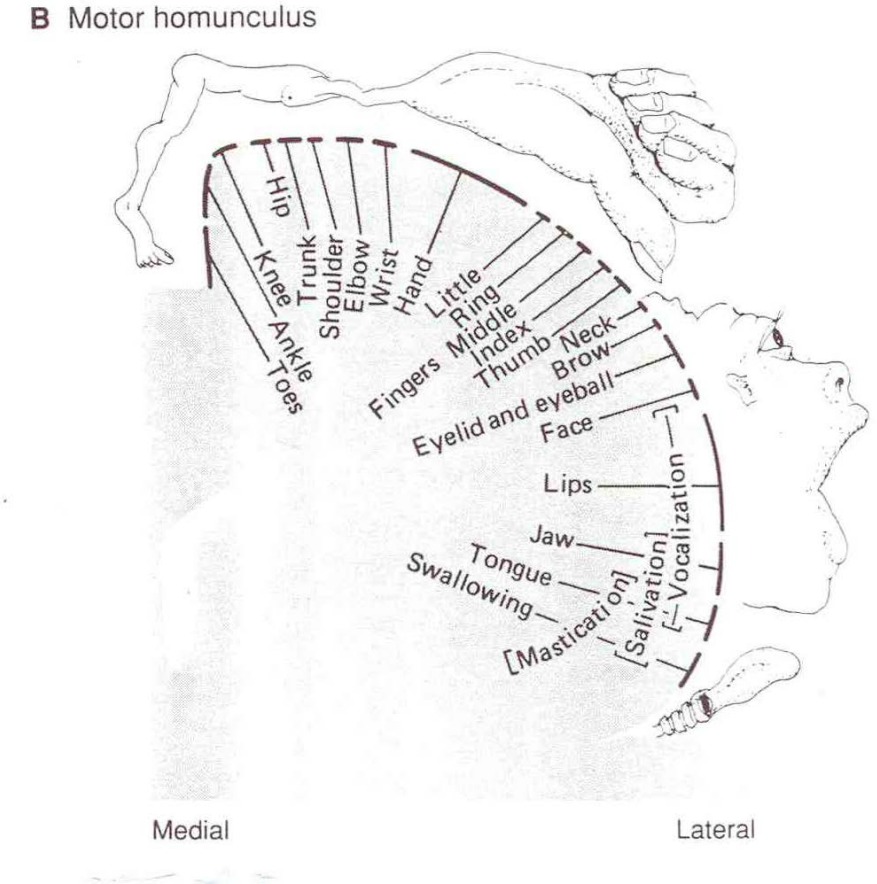
Self-organizing maps (SOM)

- Maps are commonly found in the brain: retinotopic map, tonotopic map, somatosensory (tactile) map, etc.



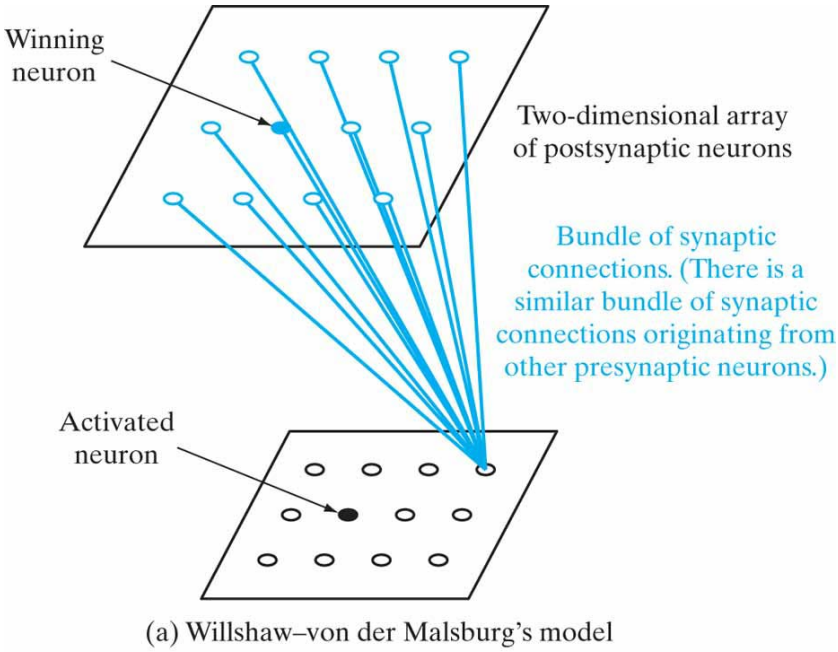
Self-organizing maps (SOM)

- Maps are commonly found in the brain: retinotopic map, tonotopic map, somatosensory (tactile) map, etc.

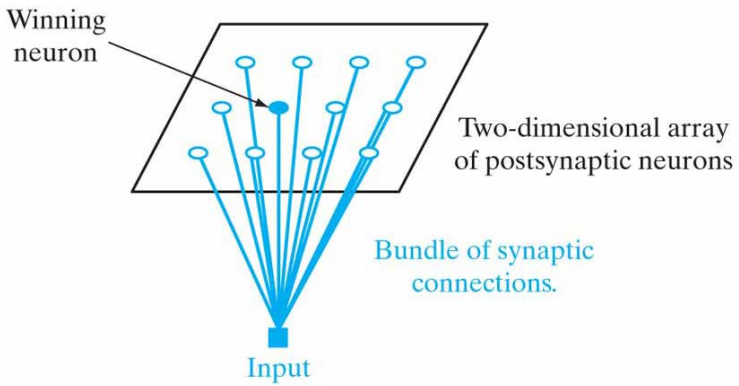


Self-organizing feature maps

- Competition and local cooperation along with synaptic plasticity can produce such maps



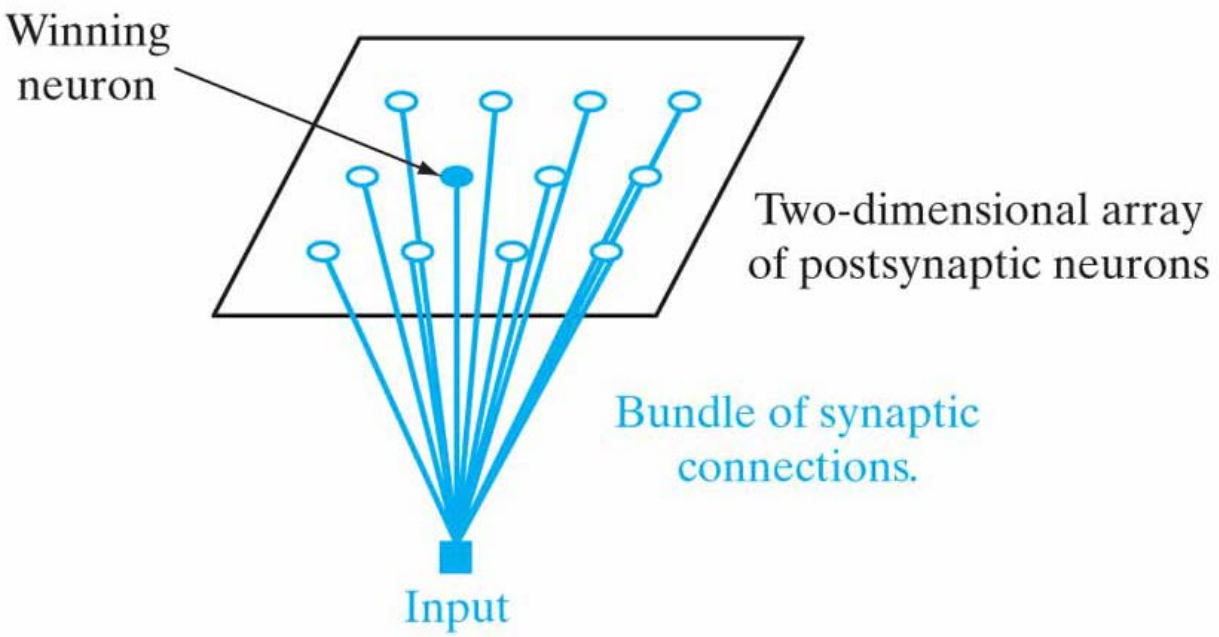
(a) Willshaw–von der Malsburg's model



(b) Kohonen model

SOM architecture

- Architecture: One layer with recurrent connections



(b) Kohonen model

SOM (cont.)

- Question: how to represent the input space by output neurons through training?
- The idea is to adjust the weight vectors of the winning neuron (via WTA competition) and its neighboring neurons, to make them closer to the input vector

Learning rule

- Weight update

$$\mathbf{w}_j(n+1) = \mathbf{w}_j(n) + \eta(n) h_{j,i(\mathbf{x})}(n)\left[\mathbf{x}(n) - \mathbf{w}_j(n)\right]$$

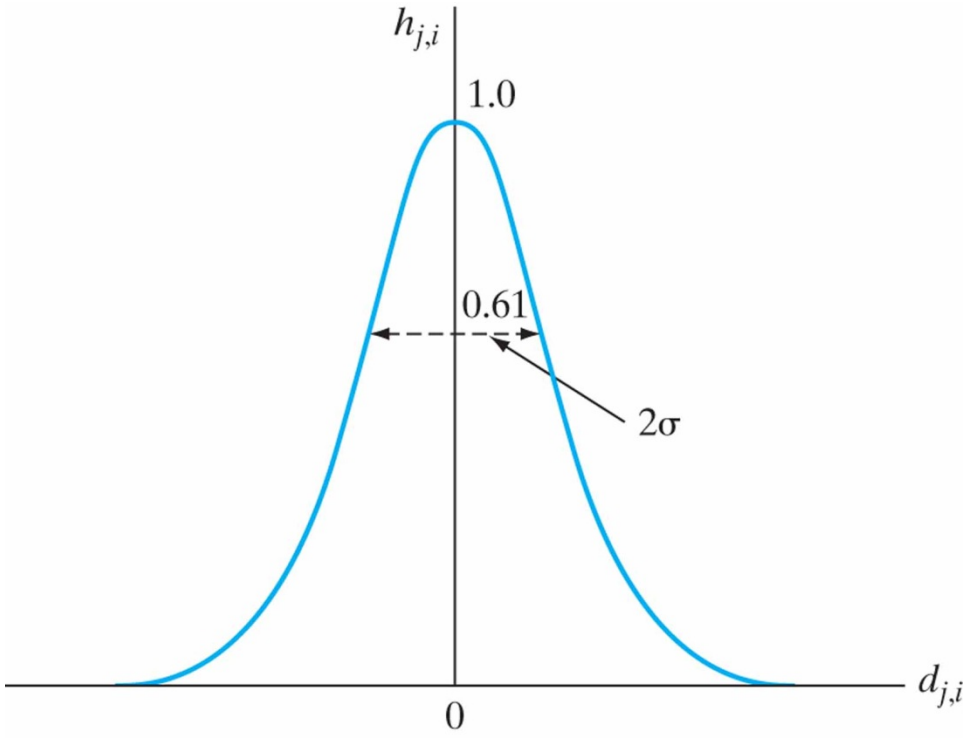
where $i(\mathbf{x})$ indicates the winning neuron, and $h_{j,i}$ denotes a neighborhood function centered at neuron i

- A typical choice for $h_{j,i}$ is a Gaussian function

$$h_{j,i}(n) = \exp\left[-\frac{{d_{j,i}}^2}{2\sigma^2(n)}\right]$$

where $d_{j,i}$ denotes the Euclidean distance between neuron j and i on the output layer

Neighborhood function



- To ensure convergence, both η and σ need to decrease gradually

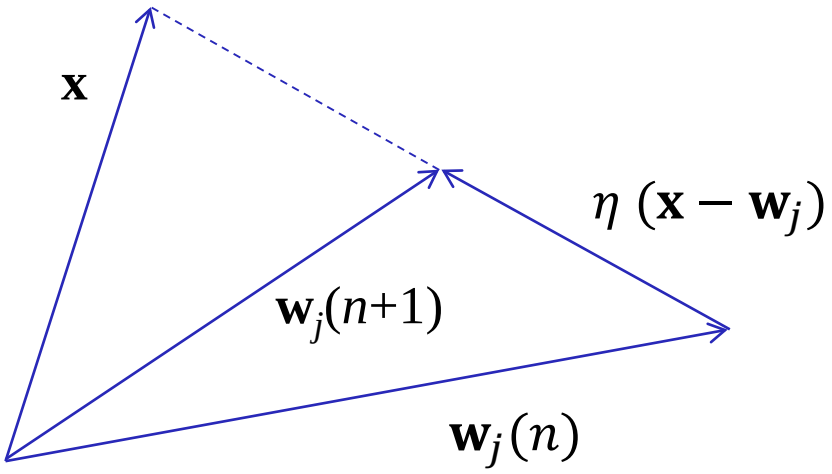
Competitive learning

- For the special case of a neighborhood function that includes just the winning neuron, SOM reduces to competitive learning:

$$\Delta \mathbf{w}_j = \eta y_j (\mathbf{x} - \mathbf{w}_j)$$

Here y_j is the (binary) response of neuron j

Competitive learning illustration



- Competitive learning implements an online version of *K*-means clustering

Two phases of SOM training

- **Ordering phase**: This phase is to achieve topological ordering of weight vectors by adapting $\sigma(n)$ and $\eta(n)$
- One approach is to set

$$\sigma(n) = \sigma_0(1 - \frac{n}{N_0})$$

where σ_0 is the initial (large) Gaussian width and N_0 is the number of iterations for the phase

$$\eta(n) = \eta_0(1 - \frac{n}{N_0 + K})$$

Here η_0 is the initial learning rate and K is another parameter

Two phases of training (cont.)

- Alternatively, we can set $\sigma(n)$ and $\eta(n)$ as given in textbook

$$\sigma(n) = \sigma_0 \exp(-\frac{n}{\tau_1})$$

$$\eta(n) = \eta_0 \exp(-\frac{n}{\tau_2})$$

where τ_1 and τ_2 are called time constants

Two phases of training (cont.)

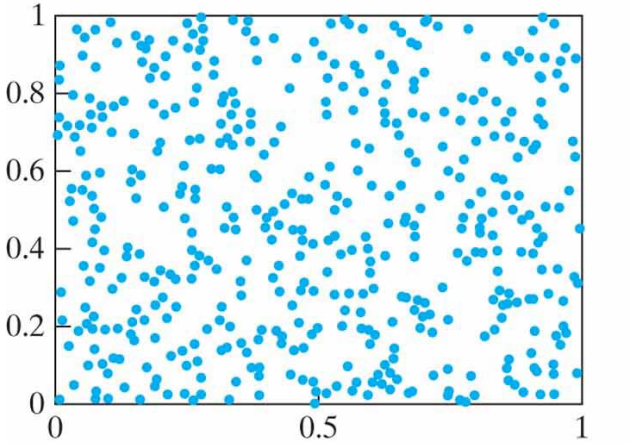
- **Convergence phase**. This phase fine-tunes the output neurons to match the input distribution
- For the convergence phase, $h_{j,i}(n)$ should contain just the nearest neighbors, which may reduce to one neuron. η should be small.

Properties of SOM

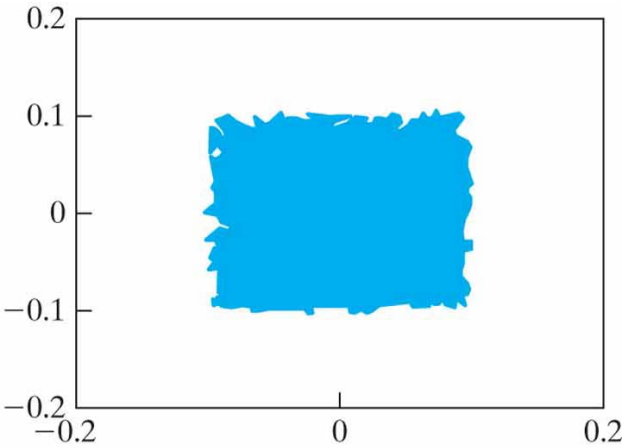
- Input space approximation
- Topological ordering

- **Remark**: SOM gives an online version of vector quantization

SOM illustrations

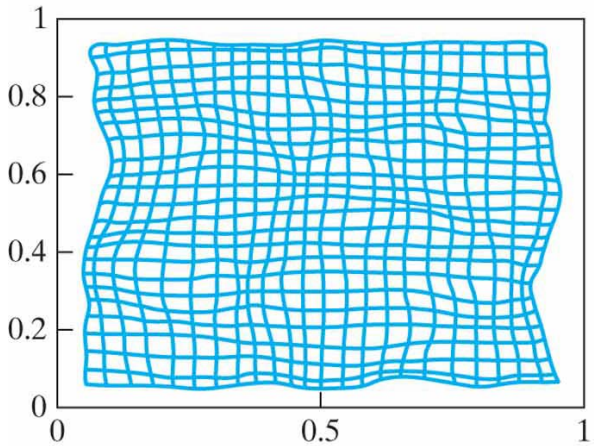


(a) Input distribution



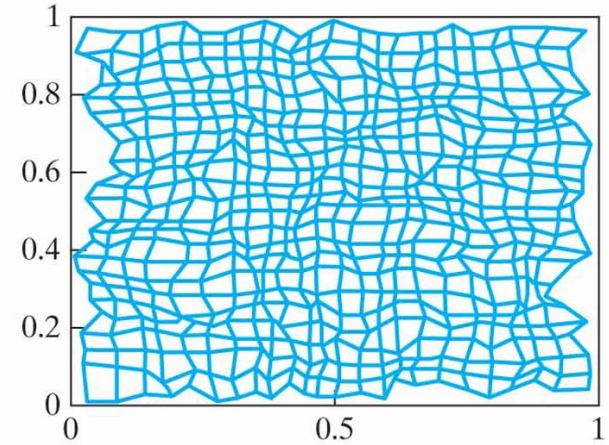
Time = 0

(b) Initial weights



Time = 160 K

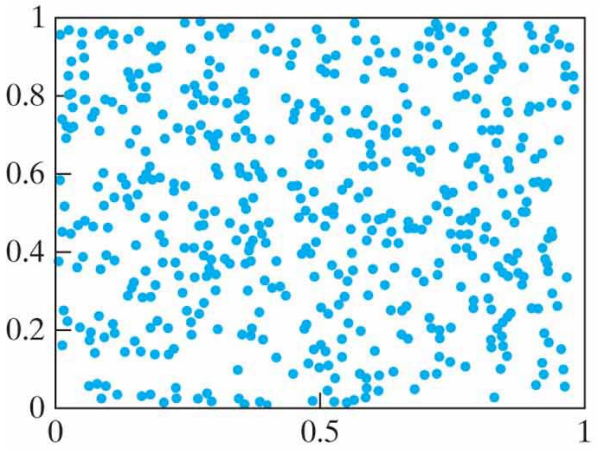
(c) Ordering phase



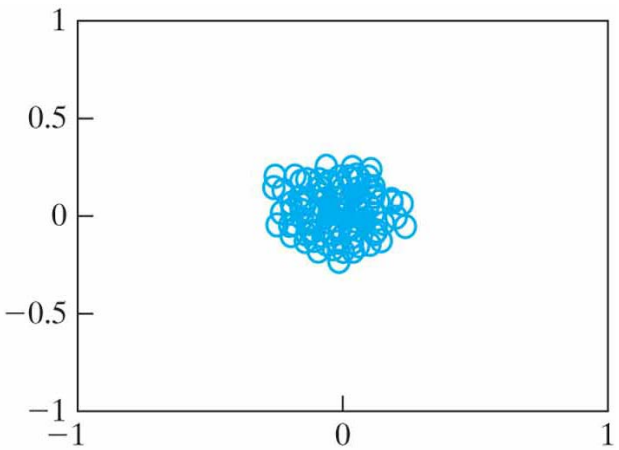
Time = 800 K

(d) Convergence phase

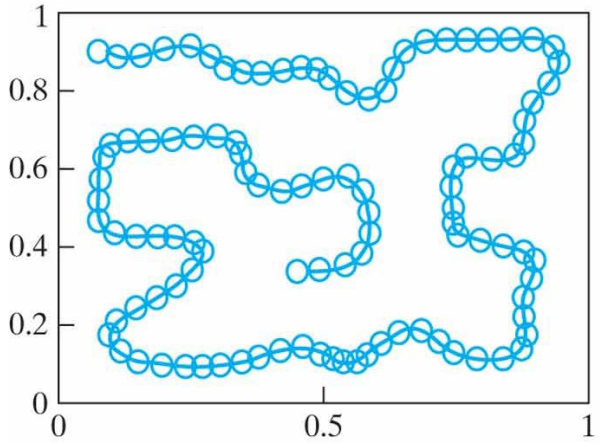
SOM illustrations (cont.)



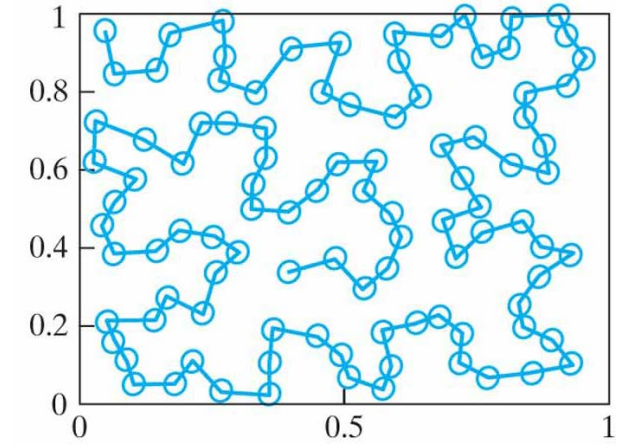
(a) Input distribution



Time = 0
(b) Initial weights

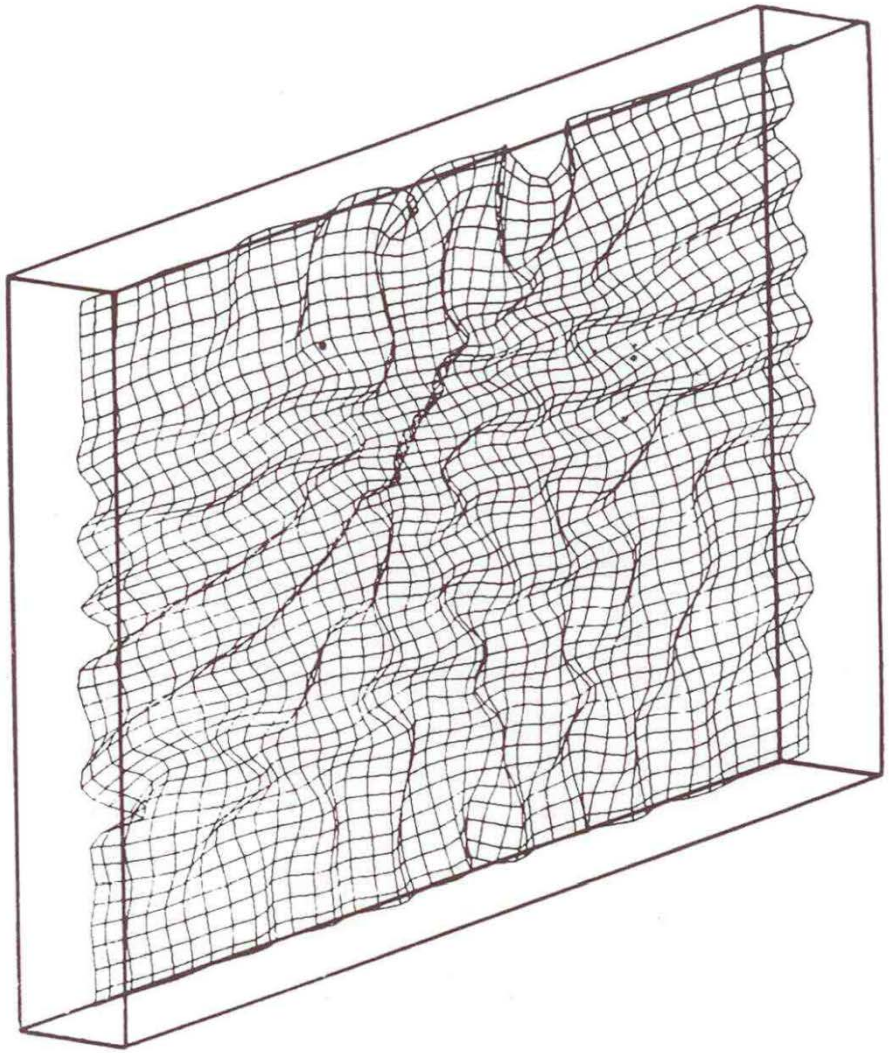


Time = 50 K
(c) Ordering phase

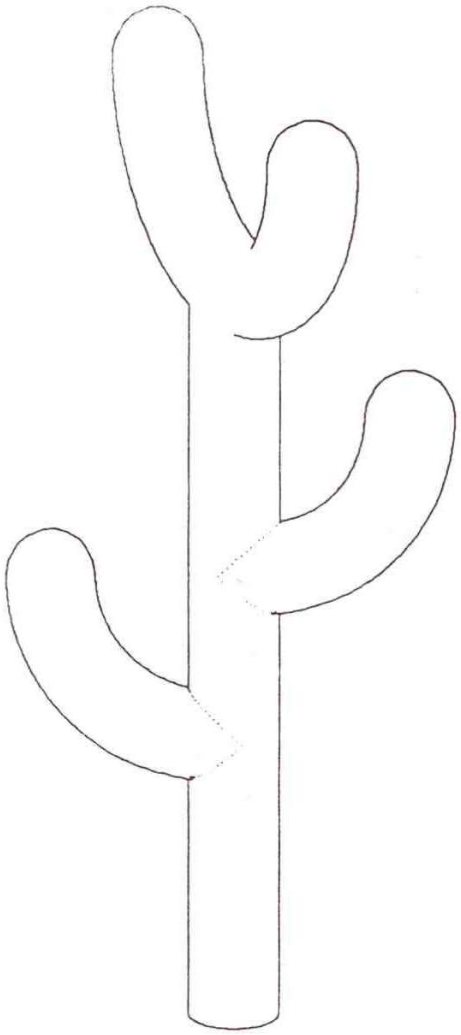


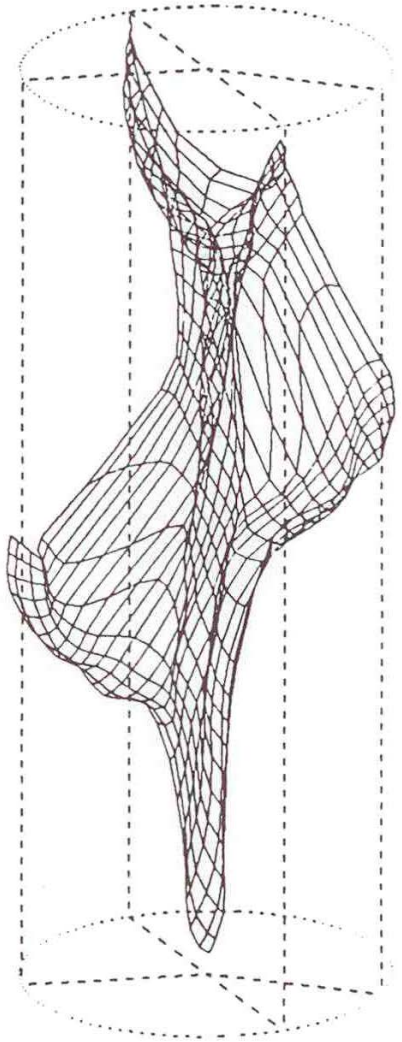
Time = 100 K
(d) Converging phase

SOM illustrations (cont.)

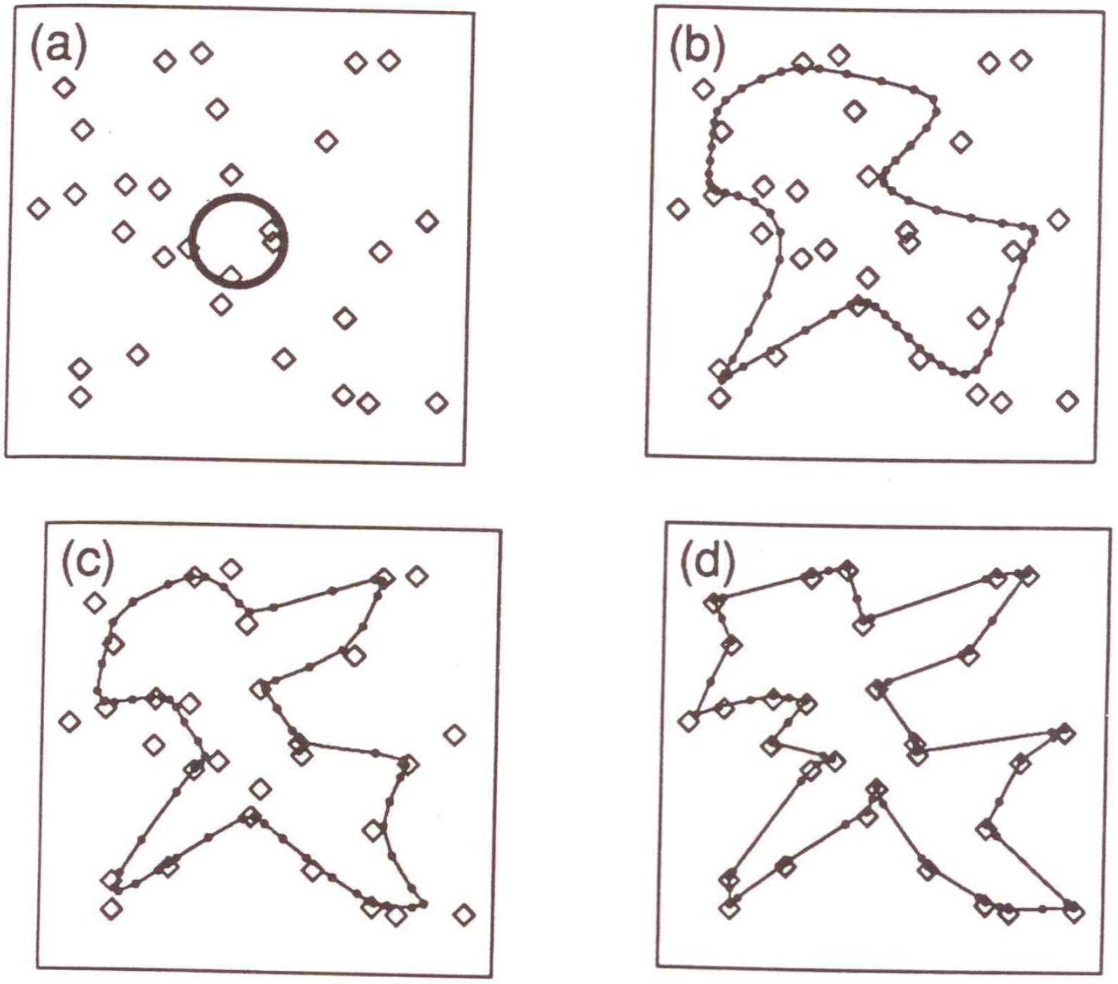


SOM illustrations (cont.)





Elastic net for traveling salesman problem



CSE 5526: Introduction to Neural Networks

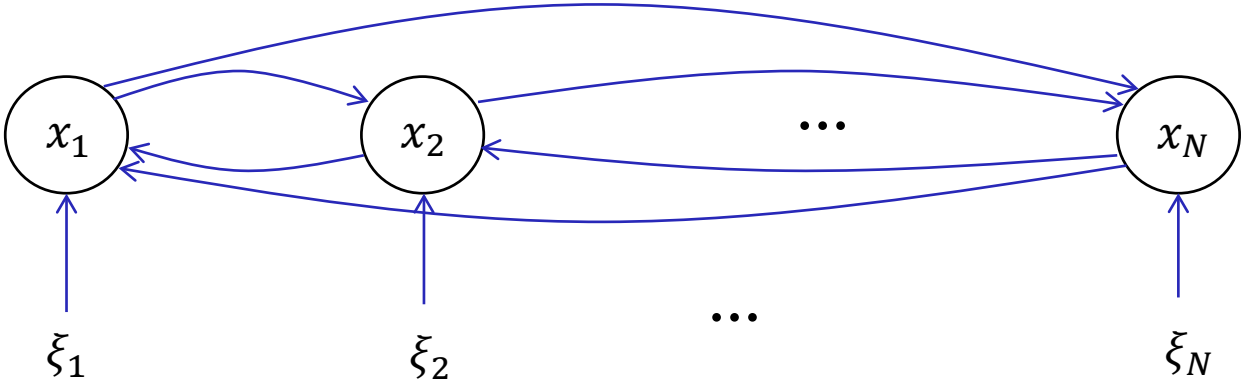
Hopfield Network for Associative Memory

The basic task

- Store a set of fundamental memories $\{\boldsymbol{\xi}_1, \boldsymbol{\xi}_2, \ldots, \boldsymbol{\xi}_M\}$ so that, when presented a new pattern $\mathbf{x}$, the system outputs one of the stored memories that is most similar to $\mathbf{x}$
  - Such a system is called content-addressable memory

Architecture

- The Hopfield net consists of *N* McCulloch-Pitts neurons, recurrently connected among themselves



Definition

- Each neuron is defined as

$$x_j = \varphi(v_j)$$

where $v_j = \sum_{i=1}^{N} w_{ji} x_i + b_j$

and $\varphi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ -1 & \text{otherwise} \end{cases}$

  - Without loss of generality, let $b_j = 0$

Storage phase

- To store fundamental memories, the Hopfield model uses the outer-product rule, a form of Hebbian learning:

$$w_{ji} = \frac{1}{N} \sum_{\mu=1}^{M} \xi_{\mu,j} \, \xi_{\mu,i}$$

- Hence $w_{ji} = w_{ij}$, i.e., $\mathbf{w} = \mathbf{w}^T$, so the weight matrix is symmetric

Retrieval phase

- The Hopfield model sets the initial state of the net to the input pattern. It adopts asynchronous (serial) update, which updates one neuron at a time

Hamming distance

- Hamming distance between two binary/bipolar patterns is the number of differing bits
  - Example (see whiteboard)

One memory case

- Let the input **x** be the same as the single memory **ξ**

$$
\begin{aligned}
x_j &= \varphi\left(\sum_i w_{ji}\, x_i\right) \\
&= \varphi\left(\frac{1}{N}\sum_i \xi_j \xi_i \xi_i\right) \\
&= \varphi(\xi_j) \\
&= \xi_j
\end{aligned}
$$

Therefore the memory is stable

One memory case (cont.)

- Actually for any input pattern, as long as the Hamming distance between $\mathbf{x}$ and $\boldsymbol{\xi}$ is less than $N/2$, the net converges to $\boldsymbol{\xi}$. Otherwise it converges to $-\boldsymbol{\xi}$
  - Think about it

Multiple memories

- The stability (alignment) condition for any memory $\boldsymbol{\xi}_{\vartheta}$ is

$$\varphi(v_j) = \xi_{\vartheta,j}$$

where

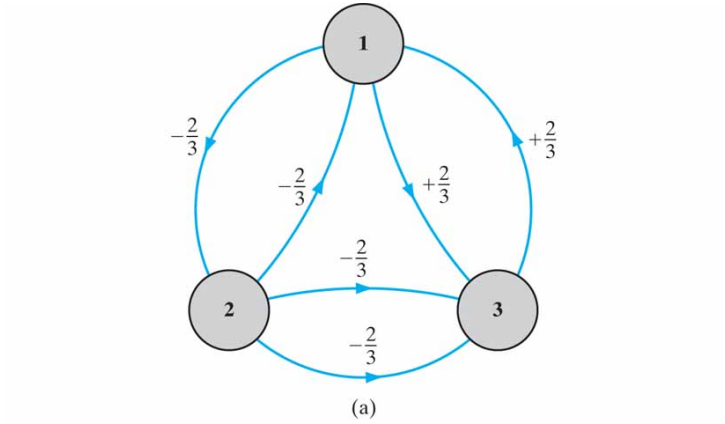
$$v_j = \sum_i w_{ji}\xi_{\vartheta,i} = \frac{1}{N}\sum_i\sum_\mu \xi_{\mu,j}\xi_{\mu,i}\xi_{\vartheta,i}$$

$$= \xi_{\vartheta,j} + \underbrace{\frac{1}{N}\sum_i\sum_{\mu\neq\vartheta} \xi_{\mu,j}\xi_{\mu,i}\xi_{\vartheta,i}}_{\text{crosstalk}}$$

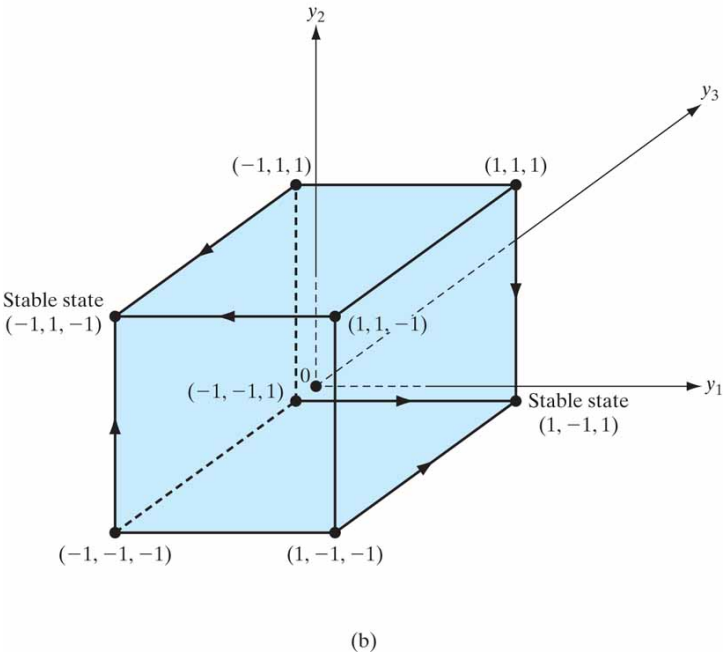
Multiple memories (cont.)

- If |crosstalk| < N, the memory system is stable. In general, fewer memories are more likely stable
- Example 2 in book (see whiteboard)

Example (cont.)



(a)



(b)

Memory capacity

- Define

$$C_j^{\vartheta} = -\xi_{\vartheta,j} \sum_i \sum_{\mu \neq \vartheta} \xi_{\mu,j} \xi_{\mu,i} \xi_{\vartheta,i}$$

$$C_j^{\vartheta} < 0 \quad \Rightarrow \quad \text{stable}$$
$$0 \leq C_j^{\vartheta} < N \quad \Rightarrow \quad \text{stable}$$
$$C_j^{\vartheta} > N \quad \Rightarrow \quad \text{unstable}$$

  - What if $C_j^{\vartheta} = N$?

Memory capacity (cont.)

- Consider random memories where each element takes +1 or -1 with equal probability (prob.). We measure

$$P_{\text{error}} = \text{Prob}({C_j}^{\vartheta} > N)$$

- To compute capacity M_{max}, decide on an error criterion
- For random patterns, ${C_j}^{\vartheta}$ is proportional to a sum of $N(M-1)$ random numbers of +1 or -1. For large NM, it can be approximated by a Gaussian distribution (central limit theorem) with zero mean and variance $\sigma^2 = NM$

Memory capacity (cont.)

- So

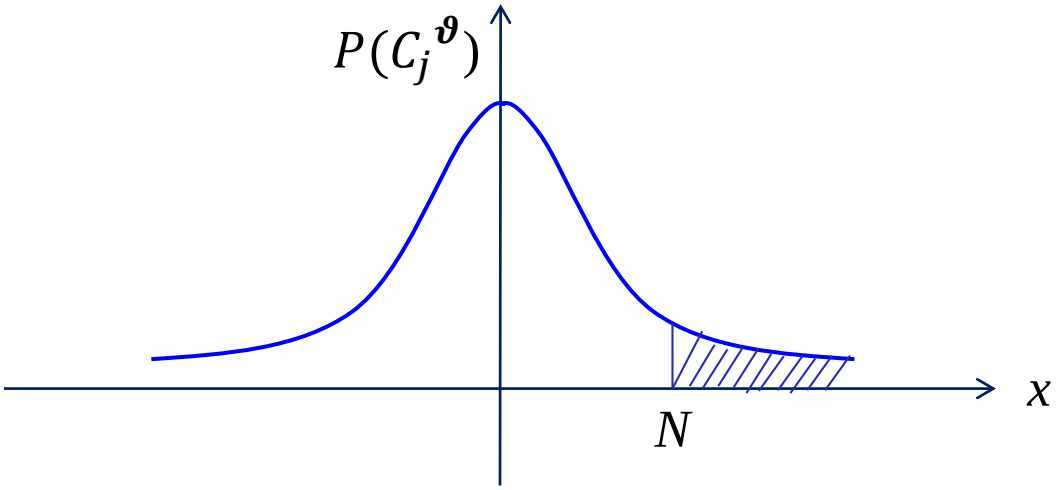
$$P_{\text{error}} = \frac{1}{\sqrt{2\pi}\sigma}\int_{N}^{\infty} \exp(-\frac{x^2}{2\sigma^2})\,\mathrm{d}x$$

$$= \frac{1}{2} - \frac{1}{\sqrt{2\pi}\sigma}\int_{0}^{N} \exp\left(-\frac{x^2}{2\sigma^2}\right)\mathrm{d}x$$

$$(x = \sqrt{2}\sigma\mu) \quad = \frac{1}{2}\left(1 - \underbrace{\frac{2}{\sqrt{\pi}}\int_{0}^{\sqrt{N/(2M)}} \exp(-\mu^2)\,\mathrm{d}\mu}_{\text{error function}}\right)$$

Memory capacity (cont.)

- Error probability plot



Memory capacity (cont.)

P_{error}	M_{max}/N
0.001	0.105
0.0036	0.138
0.01	0.185
0.05	0.37
0.1	0.61

- So $P_{\text{error}} < 0.01 \Rightarrow M_{\text{max}} = 0.185N$, an upper bound

Memory capacity (cont.)

- What if 1% flips occur? Avalanche effect?
- Real upper bound: $0.138N$ to prevent the avalanche effect

Memory capacity (cont.)

- The above analysis is for one bit. If we want perfect retrieval for $\boldsymbol{\xi}^{\vartheta}$ with probability of 0.99:

$$(1 - P_{\text{error}})^{N} > 0.99$$

  - Approximately $P_{\text{error}} < \frac{0.01}{N}$

- For this case

$$M_{\max} = \frac{N}{2\log N}$$

Memory capacity (cont.)

- But real patterns are not random (they could be encoded) and the capacity is worse for correlated patterns

- At one favorable extreme, if memories are orthogonal

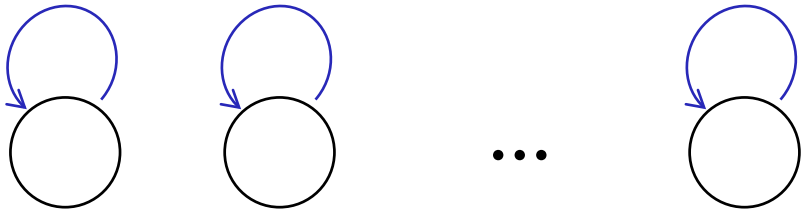
$$\sum_i \xi_{\mu,i}\xi_{\vartheta,i} = 0 \quad \text{for } \vartheta \neq \mu$$

then $C_j{}^{\vartheta} = 0$ and $M_{\max} = N$

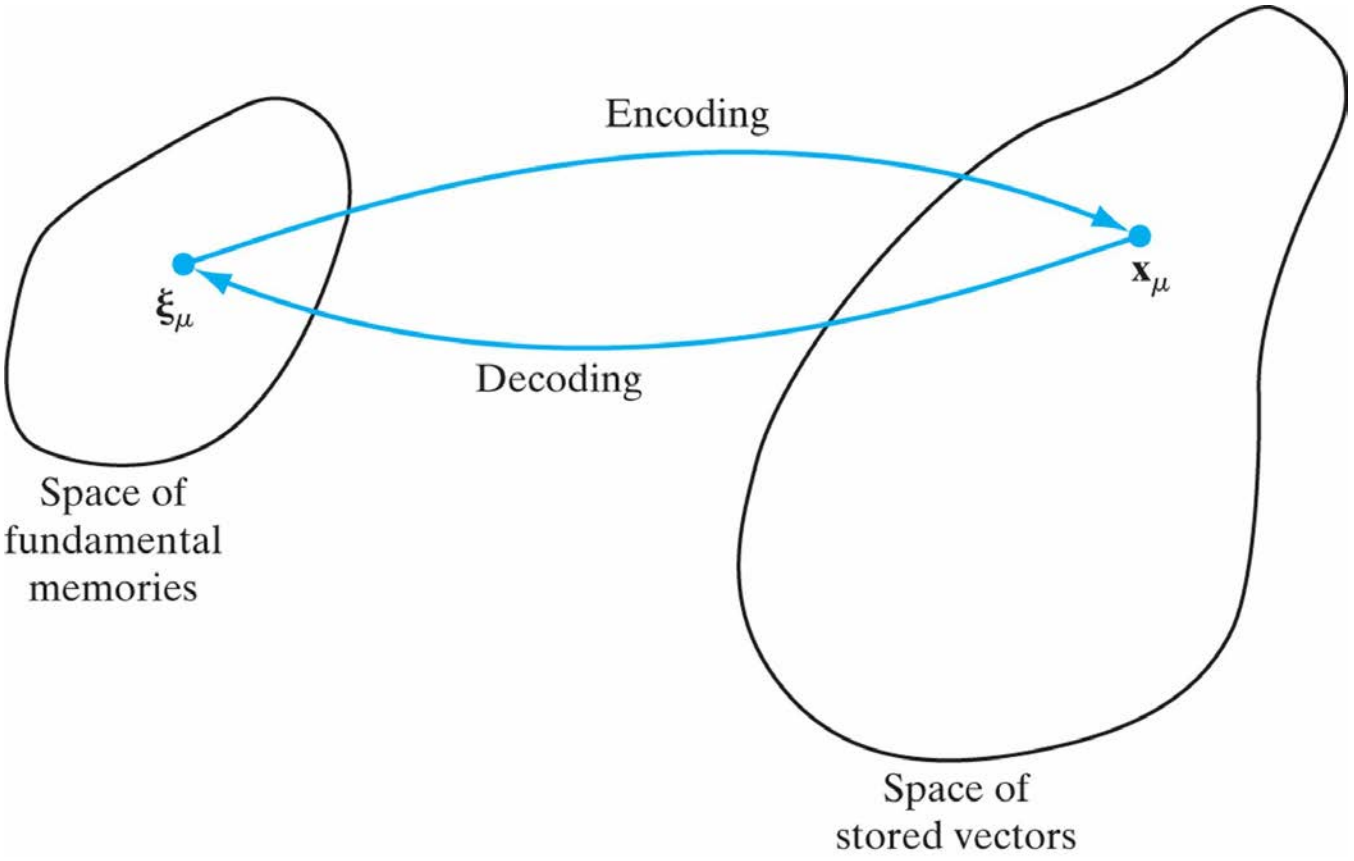
  - This is the maximum number of orthogonal patterns

Memory capacity (cont.)

- But in reality a useful system stores a little less; otherwise the memory is useless as it does not evolve



Coding illustration



Energy function (Lyapunov function)

- The existence of an energy (Lyapunov) function for a dynamical system ensures its stability
- The energy function for the Hopfield net is

$$E(\mathbf{x}) = -\frac{1}{2}\sum_i \sum_j w_{ji} x_i x_j$$

- **Theorem**: Given symmetric weights, $w_{ji} = w_{ij}$, the energy function does not increase as the Hopfield net evolves

Energy function (cont.)

- Let $x_j{}'$ be the new value of x_j after an update

$$x_j{}' = \varphi\left(\sum_i w_{ji} x_i\right)$$

- If $x_j' = x_j$, E remains the same

Energy function (cont.)

- In the other case, $x_j' = -x_j$:

$$E(x_j') - E(x_j) = -\frac{1}{2}\sum_i\sum_j w_{ji}x_i x_j' + \frac{1}{2}\sum_i\sum_j w_{ji}x_i x_j$$

since $w_{jj} = {}^{M}/_{N}$

$$= -\frac{1}{2}\sum_{i\neq j}\sum_j w_{ji}x_i x_j' + \frac{1}{2}\sum_{i\neq j}\sum_j w_{ji}x_i x_j$$

since $w_{ij} = w_{ji}$

$$= -x_j'\sum_{i\neq j} w_{ji}x_i + x_j\sum_{i\neq j} w_{ji}x_i$$

$$= 2x_j\sum_{i\neq j} w_{ji}x_i$$

$$= 2x_j\sum_i w_{ji}x_i - 2w_{jj} < 0$$

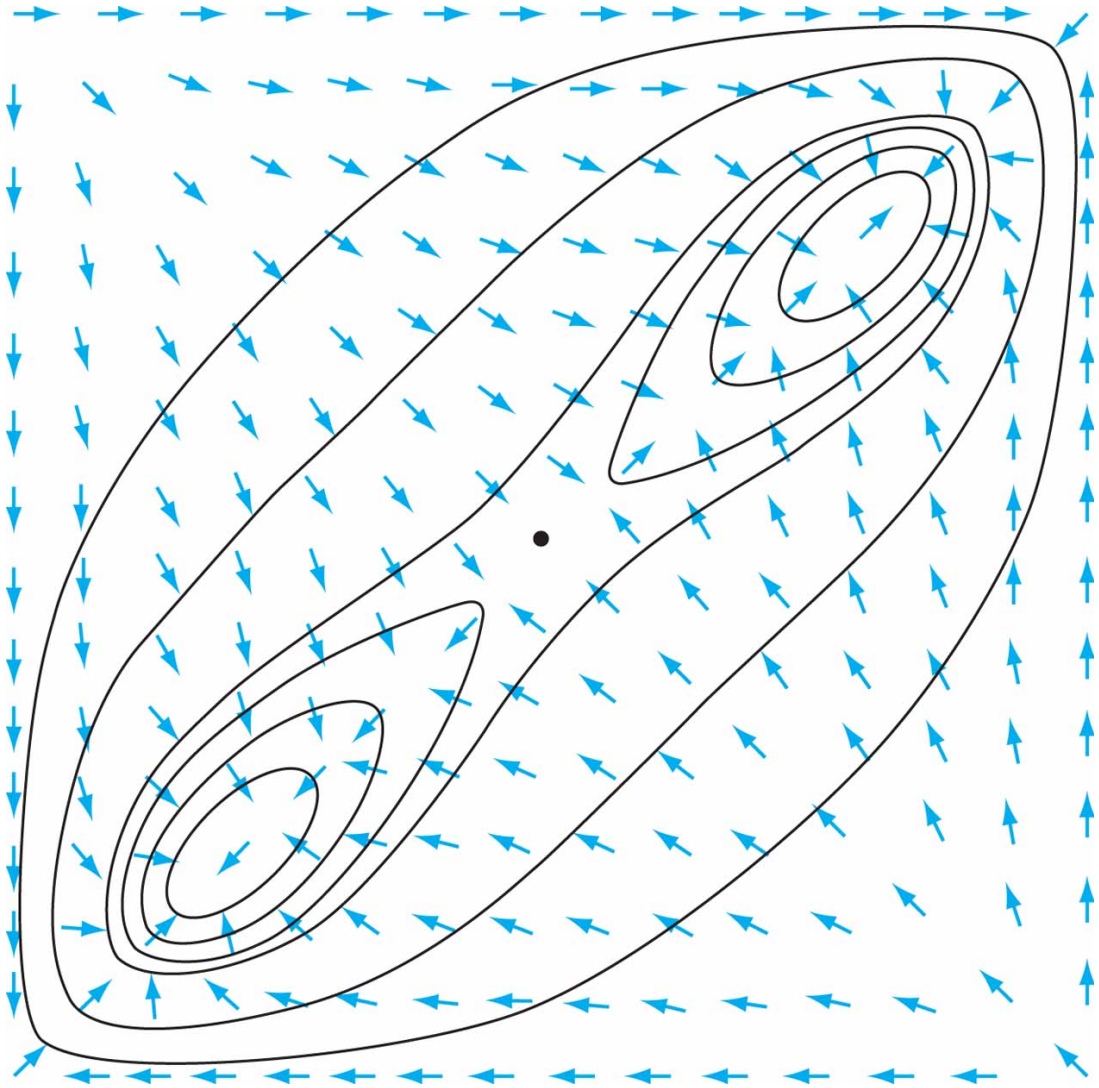
different signs

Energy function (cont.)

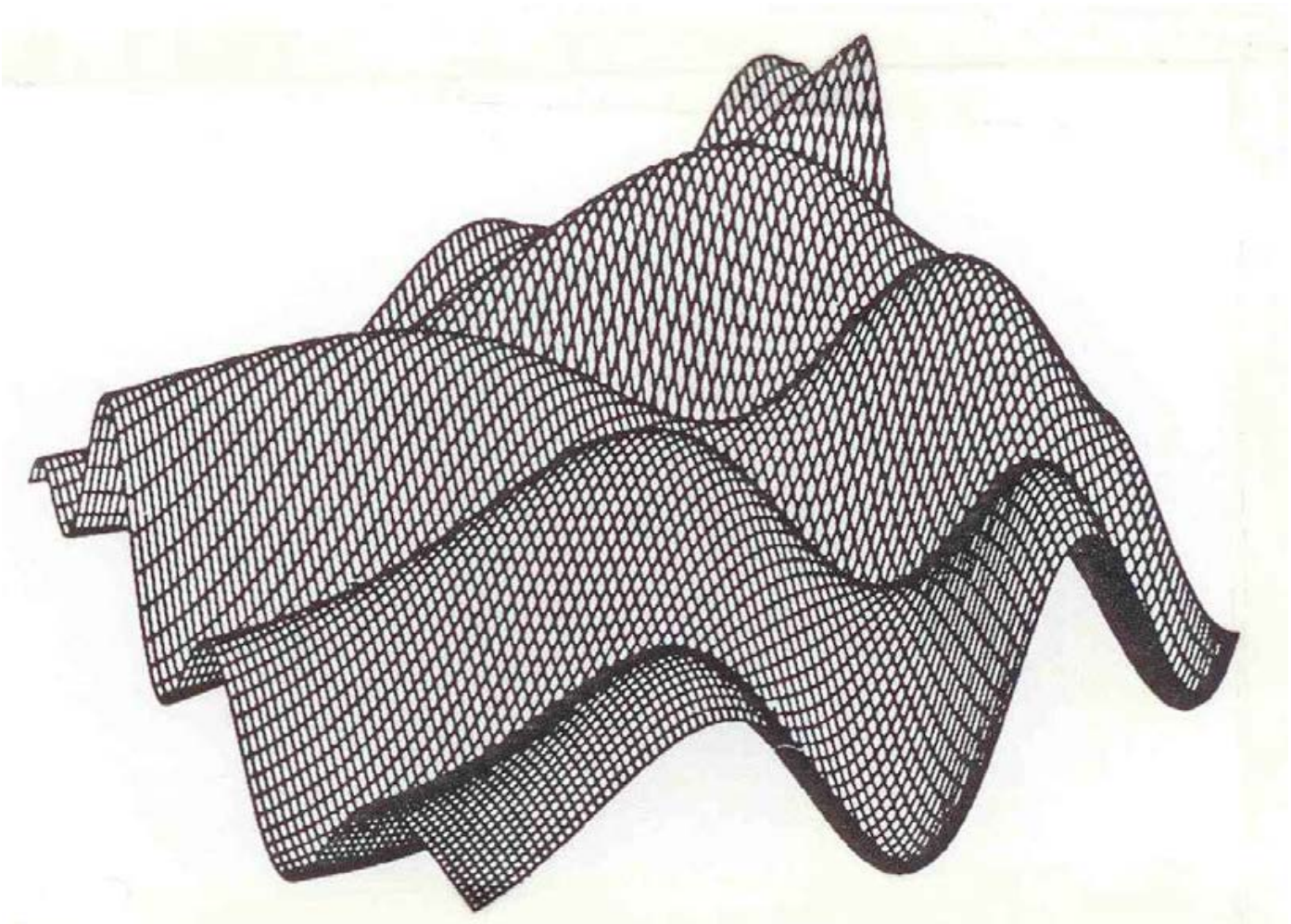
- Thus, $E(\mathbf{x})$ decreases every time x_j flips. Since E is bounded, the Hopfield net is always stable

- **Remarks**:
  - Useful concepts from dynamical systems: attractors, basins of attraction, energy (Lyapunov) surface or landscape

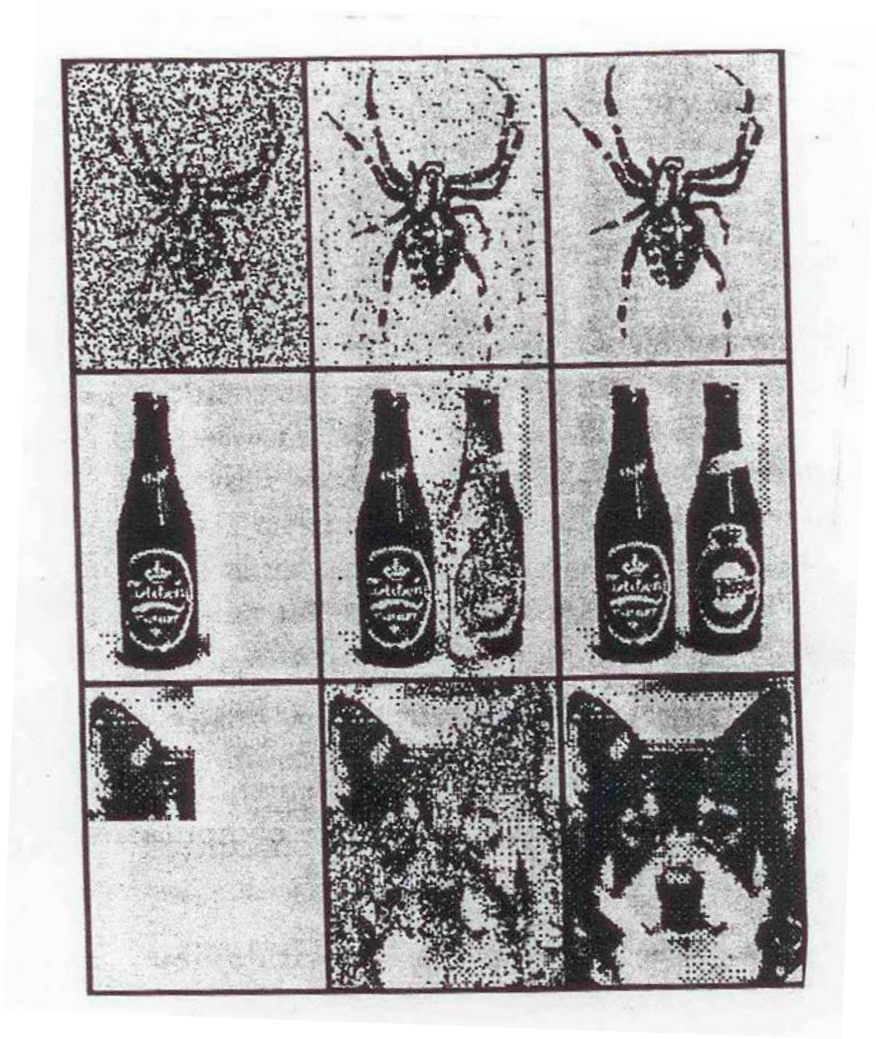
Energy contour map



2-D energy landscape



Memory recall illustration



Remarks (cont.)

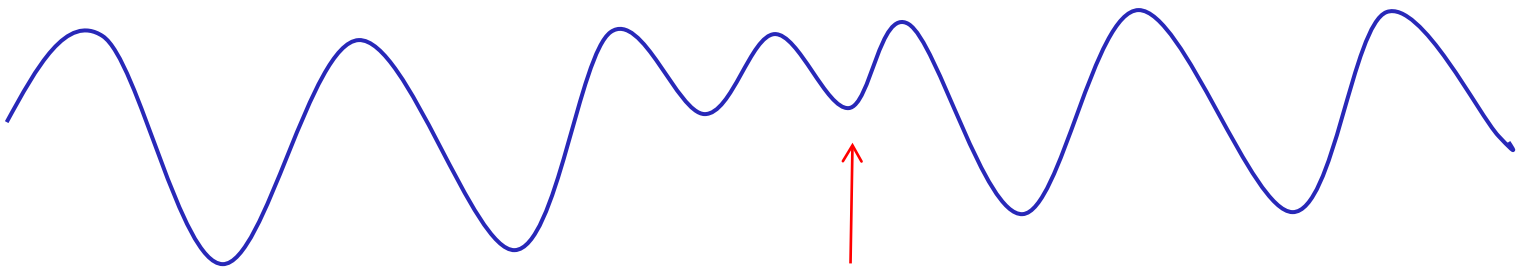
- Bipolar neurons can be extended to continuous-valued neurons by using hyperbolic tangent activation function, and discrete update can be extended to continuous-time dynamics (good for analog VLSI implementation)
- The concept of energy minimization has been applied to optimization problems (neural optimization)

Spurious states

- Not all local minima (stable states) correspond to fundamental memories. Typically, $-\xi_\mu$, linear combination of odd number of memories, or other uncorrelated patterns, can be attractors
  - Such attractors are called spurious states

Spurious states (cont.)

- Spurious states tend to have smaller basins and occur higher on the energy surface

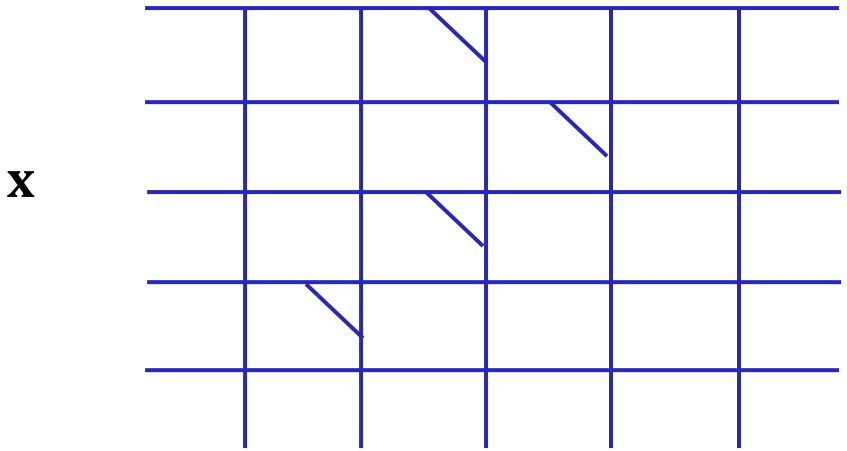


local minima
for spurious states

Kinds of associative memory

- Autoassociative (e.g. Hopfiled net)

  Heteroassociative: store pairs of $< \mathbf{x}_\mu, \mathbf{y}_\mu >$ explicitly



matrix memory

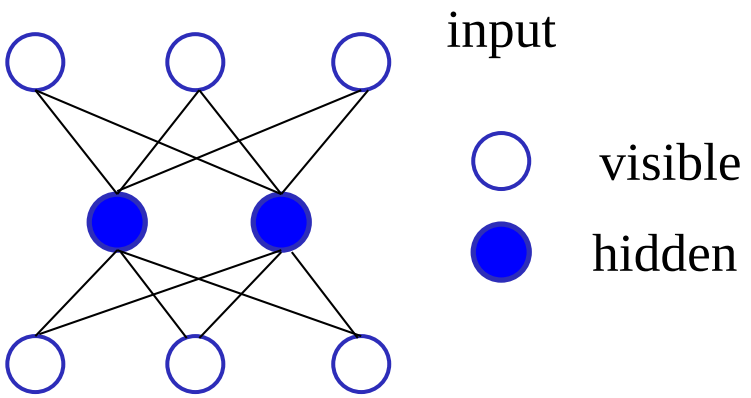
holographic memory

CSE 5526: Introduction to Neural Networks

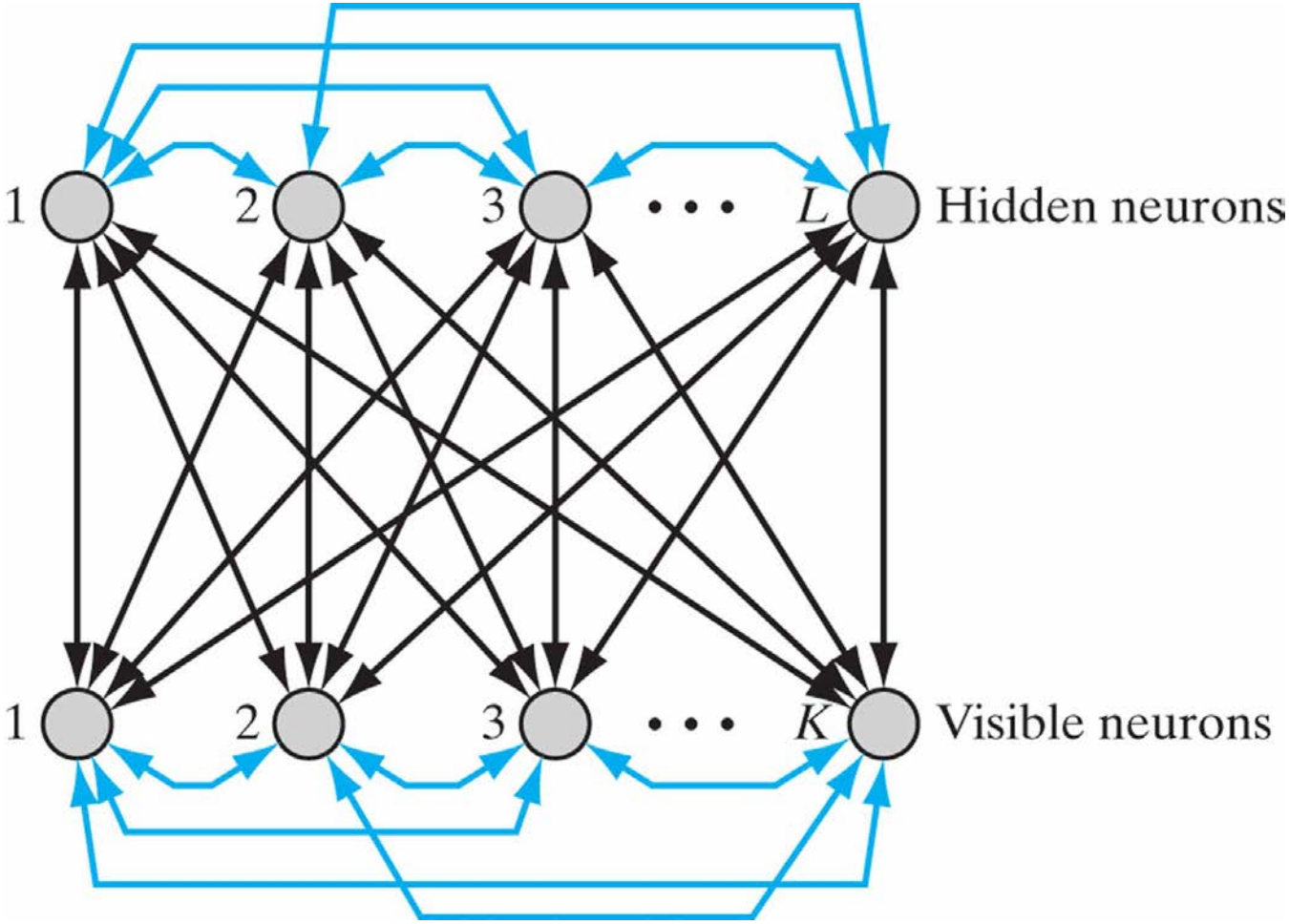
Boltzmann Machines

Introduction

- Boltzmann machine is a stochastic learning machine that consists of visible and hidden units and symmetric connections
  - It may be viewed as a stochastic extension of the Hopfield net
- The network can be layered and visible units can be either input or output



Another architecture



Stochastic neurons

- Definition:

$$x_i = \begin{cases} 1 & \text{with prob. } \varphi(v_i) \\ -1 & \text{with prob. } 1 - \varphi(v_i) \end{cases}$$

- For symmetric connections, there is an energy function:

$$E(\mathbf{x}) = -\frac{1}{2}\sum_i \sum_j w_{ji} x_i x_j$$

  - The same as in the Hopfield net

Boltzmann-Gibbs distribution

- Consider a physical system with a large number of states. Let p_i denote the prob. of occurrence of state i of the stochastic system. Let E_i denote the energy of state i
- From statistical mechanics, when the system is in thermal equilibrium, it satisfies the Boltzmann-Gibbs distribution

$$p_i = \frac{1}{Z}\exp(-\frac{E_i}{T})$$

and

$$Z = \sum_i \exp(-\frac{E_i}{T})$$

- Z is called the partition function, and T is called the temperature

Remarks

- Lower energy states have higher prob. of occurrences
- As T decreases, the prob. is concentrated on a small subset of low energy states

Boltzmann machines

- Boltzmann machines use stochastic neurons
- For neuron i:

$$v_i = \sum_j w_{ij} x_j$$

  - A bias term can be included

$$\varphi(v) = \frac{1}{1 + \exp(-\frac{2v}{T})}$$

Objective of Boltzmann machines

- The primary goal of Boltzmann learning is to produce a network that correctly models the probability distribution of visible neurons
  - Such a net can be used for pattern completion, part of associative memory, among other tasks

Positive and negative phases

- Divide the entire net into the subset $\mathbf{x}_\alpha$ of visible units and $\mathbf{x}_\beta$ of hidden units. There are two phases to the learning process:
  1. **Positive phase**: the net operates in the "clamped" condition, where visible units take on training patterns with the desired prob. distribution
  2. **Negative phase**: the net operates freely without the influence of external input

Positive and negative phases (cont.)

- In the negative phase, the prob. of having visible units in state α is

$$P(\mathbf{x}_\alpha) = \frac{1}{Z} \sum_{\mathbf{x}_\beta} \exp\left(-\frac{E(\mathbf{x})}{T}\right)$$

which is the marginal distribution

Learning

- By adjusting the weight vector **w**, the objective of Boltzmann learning is to maximize the likelihood of the visible units taking on training patterns during the negative phase
- Assuming that each pattern of the training sample is statistically independent. The log prob. of the training sample is:

$$L(\mathbf{w}) = \log \prod_{\mathbf{x}_\alpha} P(\mathbf{x}_\alpha) = \sum_{\mathbf{x}_\alpha} \log P(\mathbf{x}_\alpha)$$

$$= \sum_{\mathbf{x}_\alpha} \left[\log \sum_{\mathbf{x}_\beta} \exp\left(-\frac{E(\mathbf{x})}{T}\right) - \log \sum_{\mathbf{x}} \exp\left(-\frac{E(\mathbf{x})}{T}\right) \right]$$

Learning (cont.)

$$\frac{\partial L(\mathbf{w})}{\partial w_{ji}}$$

$$= \sum_{\mathbf{x}_\alpha} \left[\frac{\partial}{\partial w_{ji}} \log \sum_{\mathbf{x}_\beta} \exp\left(-\frac{E(\mathbf{x})}{T}\right) - \frac{\partial}{\partial w_{ji}} \log \sum_{\mathbf{x}} \exp\left(-\frac{E(\mathbf{x})}{T}\right) \right]$$

$$= \sum_{\mathbf{x}_\alpha} \left[\frac{-\frac{1}{T}\sum_{\mathbf{x}_\beta} \exp\left(-\frac{E(\mathbf{x})}{T}\right) \frac{\partial E(\mathbf{x})}{\partial w_{ji}}}{\sum_{\mathbf{x}_\beta} \exp\left(-\frac{E(\mathbf{x})}{T}\right)} + \frac{\frac{1}{T}\sum_{\mathbf{x}} \exp\left(-\frac{E(\mathbf{x})}{T}\right) \frac{\partial E(\mathbf{x})}{\partial w_{ji}}}{\sum_{\mathbf{x}} \exp\left(-\frac{E(\mathbf{x})}{T}\right)} \right]$$

since $\frac{\partial E(\mathbf{x})}{\partial w_{ji}} = -x_j x_i$

Gradient of log probability

- We have

$$\frac{\partial L(\mathbf{w})}{\partial w_{ji}} = \frac{1}{T}\sum_{\mathbf{x}_\alpha}\left[\sum_{\mathbf{x}_\beta}\frac{\exp\left(-\frac{E(\mathbf{x})}{T}\right)x_j x_i}{\sum_{\mathbf{x}_\beta}\exp\left(-\frac{E(\mathbf{x})}{T}\right)} - \frac{\sum_{\mathbf{x}}\exp\left(-\frac{E(\mathbf{x})}{T}\right)x_j x_i}{Z}\right]$$

$$= \frac{1}{T}\left(\sum_{\mathbf{x}_\alpha}\sum_{\mathbf{x}_\beta}\frac{\exp\left(-\frac{E(\mathbf{x})}{T}\right)x_j x_i}{\sum_{\mathbf{x}_\beta}\exp\left(-\frac{E(\mathbf{x})}{T}\right)} - \sum_{\mathbf{x}_\alpha}\sum_{\mathbf{x}} P(\mathbf{x})x_j x_i\right)$$

training patterns

constant w.r.t. $\Sigma_{\mathbf{x}_\alpha}$.

Gradient of log probability (cont.)

$$= \frac{1}{T}\left(\sum_{\mathbf{x}_\alpha}\sum_{\mathbf{x}_\beta} P(\mathbf{x}_\beta|\mathbf{x}_\alpha) x_j x_i - < x_j x_i >\right)$$

$$= \frac{1}{T}\left(\rho_{ji}^{+} - \rho_{ji}^{-}\right)$$

where $\rho_{ji}^{+} = \sum_{\mathbf{x}_\alpha}\sum_{\mathbf{x}_\beta} P(\mathbf{x}_\beta|\mathbf{x}_\alpha) x_j x_i$

is the mean correlation between neurons *i* and *j* when the machine operates in the positive phase

$\rho_{ji}^{-} =< x_j x_i >$ is the mean correlation between *i* and *j* when the machine operates in the negative phase

Maximization of $L(\mathbf{w})$

- To maximize $L(\mathbf{w})$, we use gradient ascent:

$$\Delta w_{ji} = \varepsilon \frac{\partial L(\mathbf{w})}{\partial w_{ji}} = \frac{\varepsilon}{T}\left(\rho_{ji}{}^{+} - \rho_{ji}{}^{-}\right)$$
$$= \eta\left(\rho_{ji}{}^{+} - \rho_{ji}{}^{-}\right)$$

where the learning rate η incorporates the temperature T

- **Remarks**: The Boltzmann learning rule is a local rule, concerning only “presynaptic” and “ postsynaptic” neurons

Gibbs sampling and simulated annealing

- Consider a *K*-dimensional random vector $\mathbf{x} = (x_1, \ldots, x_K)^T$. Suppose we know the conditional distribution x_k given the values of the remaining random variables. Gibbs sampling operates in iterations

Gibbs sampling

- For iteration n:

  $x_1(n)$ is drawn from the conditional distribution of x_1 given $x_2(n-1), x_3(n-1),\ldots, x_K(n-1)$

  …

  $x_k(n)$ is drawn from the conditional distribution of x_k given $x_1(n), x_2(n),..., x_{k-1}(n), x_{k+1}(n-1),..., x_K(n-1)$

  …

  $x_K(n)$ is drawn from the conditional distribution of x_K given $x_1(n),..., x_{K-1}(n)$

Gibbs sampling (cont.)

- In other words, each iteration samples a random variable once in the natural order, and newly sampled values are used immediately (i.e., asynchronous sampling)

Prob. of flipping a single neuron

- For Boltzmann machines, each step of Gibbs sampling corresponds to updating a single stochastic neuron
- Equivalently, we can consider the prob. of flipping a single neuron i:

$$P(x_i \to -x_i) = \frac{1}{1 + \exp\left(\frac{\Delta E_i}{T}\right)}$$

where ΔE_i is the energy change due to the flip (proof is a homework problem)

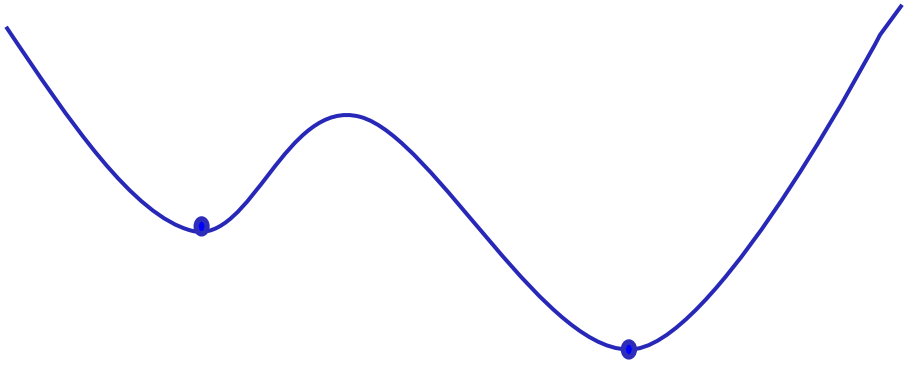
- So a change that decreases the energy is more likely than that increasing the energy

Simulated annealing

- As the temperature T decreases, the average energy of a stochastic system tends to decrease. It reaches the global minimum as $T \rightarrow 0$
- So for optimization problems, we should favor very low temperatures. On the other hand, convergence to thermal equilibrium is very slow at low temperature due to trapping at local minima
- Simulated annealing is a stochastic optimization technique that gradually decreases T. In this case, the energy is interpreted as the cost function and the temperature as a control parameter

Simulated annealing (cont.)

- No guarantee for the global minimum, but higher chances for lower local minima



- Boltzmann machines use simulated annealing to gradually lower T

Simulated annealing algorithm

- The entire algorithm consists of the following nested loops:
  1. Many epochs of adjusting weights
  2. For each epoch, compute $< x_i x_j >$ for each clamped state for a single training pattern, and a free-running, unclamped state
  3. For each state, using simulated annealing by gradually decreasing T
  4. For each T, update the entire net for a number of times with Gibbs sampling
- Boltzmann machines are extremely slow, but potentially effective. Because of its computational complexity, the algorithm has only been applied to toy problems

An example

- The encoder problem (see blackboard)

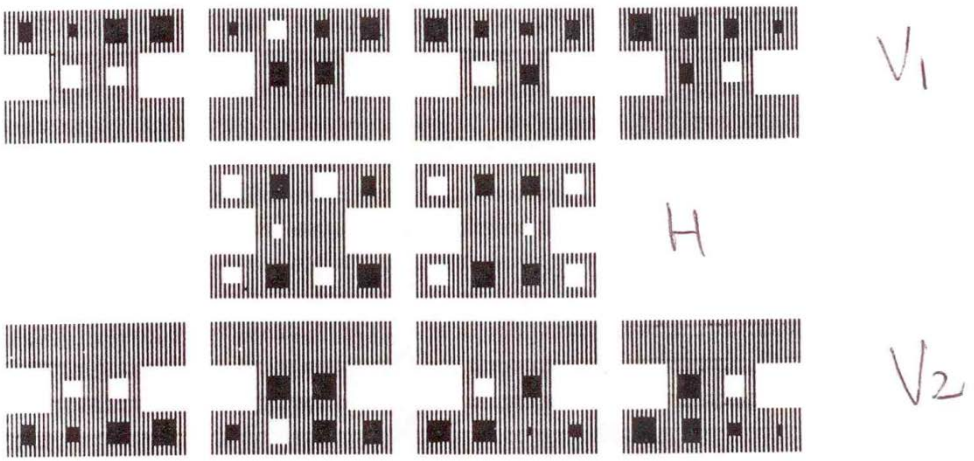


Figure 2 A solution to an encoder problem. The link weights are displayed using a recursive notation. Each unit is represented by a shaded 1-shaped box; from top to bottom the rows of boxes represent groups V_1, H, and V_2. Each shaded box is a map of the entire network, showing the strengths of that unit's connections to other units. At each position in a box, the size of the white (positive) or black (negative) rectangle indicates the magnitude of the weight. In the position that would correspond to a unit connecting to itself (the second position in the top row of the second unit in the top row, for example), the bias is displayed. All connections between units appear twice in the diagram, once in the box for each of the two units being connected. For example, the black square in the top right corner of the left-most unit of V_1 represents the same connection as the black square in the top left corner of the rightmost unit of V_1. This connection has a weight of -30.

CSE 5526: Introduction to Neural Networks

Deep Networks

Motivation

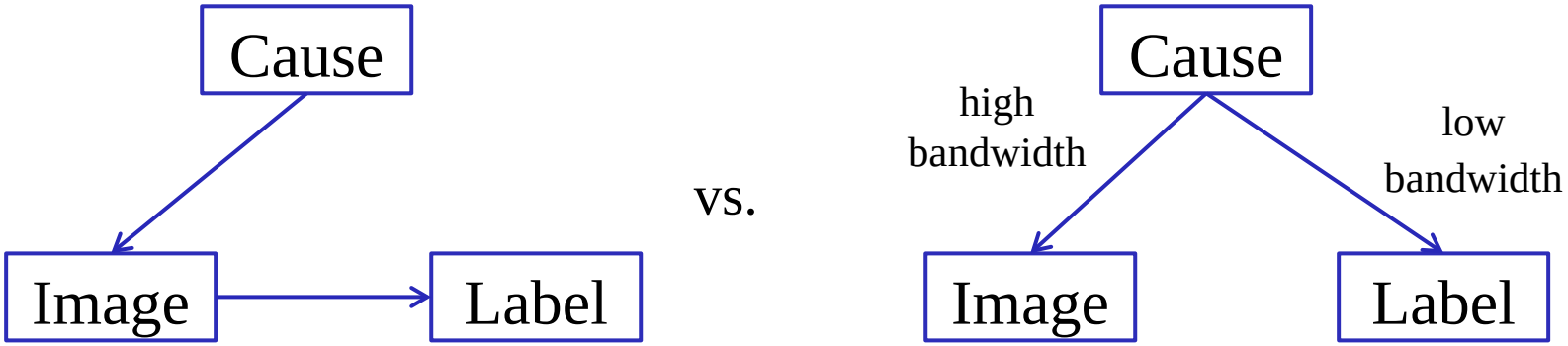
- Shallow nets involve one (hidden) layer of feature detectors followed by an output layer, e.g. MLP with one hidden layer, RBF, and SVM
- Deep nets use more than one hidden layer, e.g. convolutional nets
  - Ex: The addition of two 3-bit binary numbers. The most natural solution would use a small net of two hidden layers. A shallow net can do the job, but with a large net

Motivation (cont.)

- Backprop is applicable to an arbitrary number of hidden layers. But in practice, it exhibits vanishing gradients in shallower (near the input layer) layers. As a result, training is slow and tends to overfit the data
- To overcome this problem, we train a deep net by dividing training into two phases:
  - Use unsupervised, generative pre-training to initialize a deep net
  - Use discriminative fine-tuning to finalize the net

Motivation (cont.)

- Why two phases?



Traditional learning

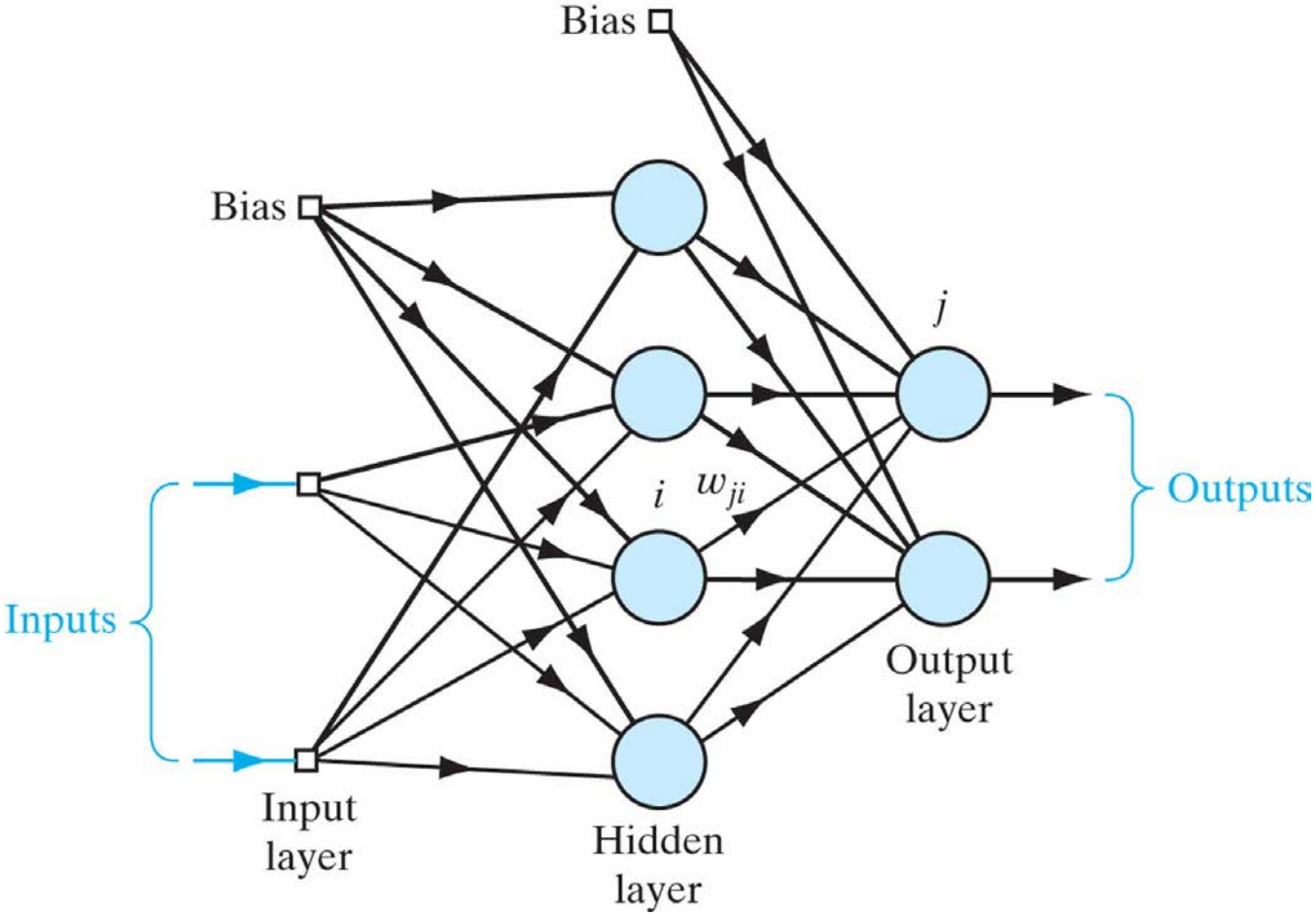
Deep learning

Motivation (cont.)

- Because overfitting is not as problematic for unsupervised learning, we expect deep learning to generalize better
- In addition, Phase 1 allows us to validate what the net has learned by generating patterns from the learned, generative model

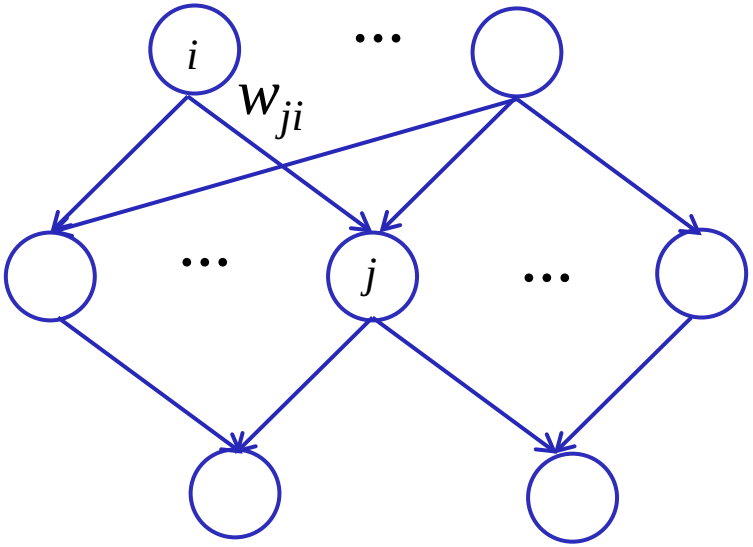
Logistic belief net

- An example logistic belief net



Logistic belief net (cont.)

- Each unit is bipolar (binary) and stochastic. Generating data from the belief net is easy



State of each unit

- Given the bipolar states of the units in layer k, we generate the state of each unit in layer $k-1$:

$$P\left(h_j{}^{(k-1)}=1\right)=\varphi\left(\sum_i w_{ji}{}^{(k)} h_i{}^{(k)}\right)$$

where superscript indicates layer number and

$$\varphi(x)=\frac{1}{1+\exp(-x)}$$

a logistic activation function

Learning rule

- The bottom layer $\mathbf{h}^{(0)}$ is the same as the visible layer $\mathbf{v}$
- Learning in a belief net is to maximize the likelihood of generating the input patterns applied to $\mathbf{v}$. Similar to Boltzmann learning, we have

$$\Delta w_{ji} =< h_i^{(k)} \left\{ h_j^{(k-1)} - 2[P\left(h_j^{(k-1)} = 1\right) - \frac{1}{2}] \right\} >$$

(or for sigmoid units that output between 0 and 1)

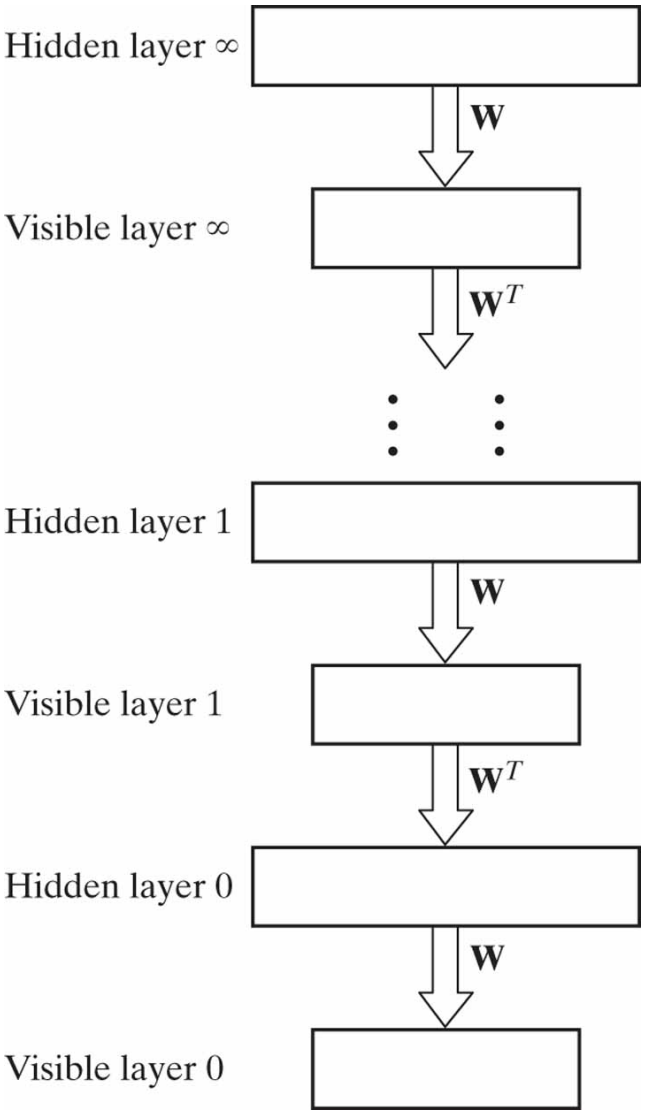
$$\Delta w_{ji} =< h_i^{(k)} \left(h_j^{(k-1)} - \widehat{h_j}^{(k-1)} \right) >$$

- The difference term in the above equation includes an evaluation of the posterior prob. given the training data
- Computing posteriors is, unfortunately, very difficult

A special belief net

- However, for a special kind of belief net, computing posteriors is easy
- Consider a logistic belief net with an infinite no. of layers and tied weights
  - That is, a deep belief net (DBN)

Infinite logistic net



- In such a net, sampling the posterior prob. is the same as generating data in belief nets

Learning in DBN

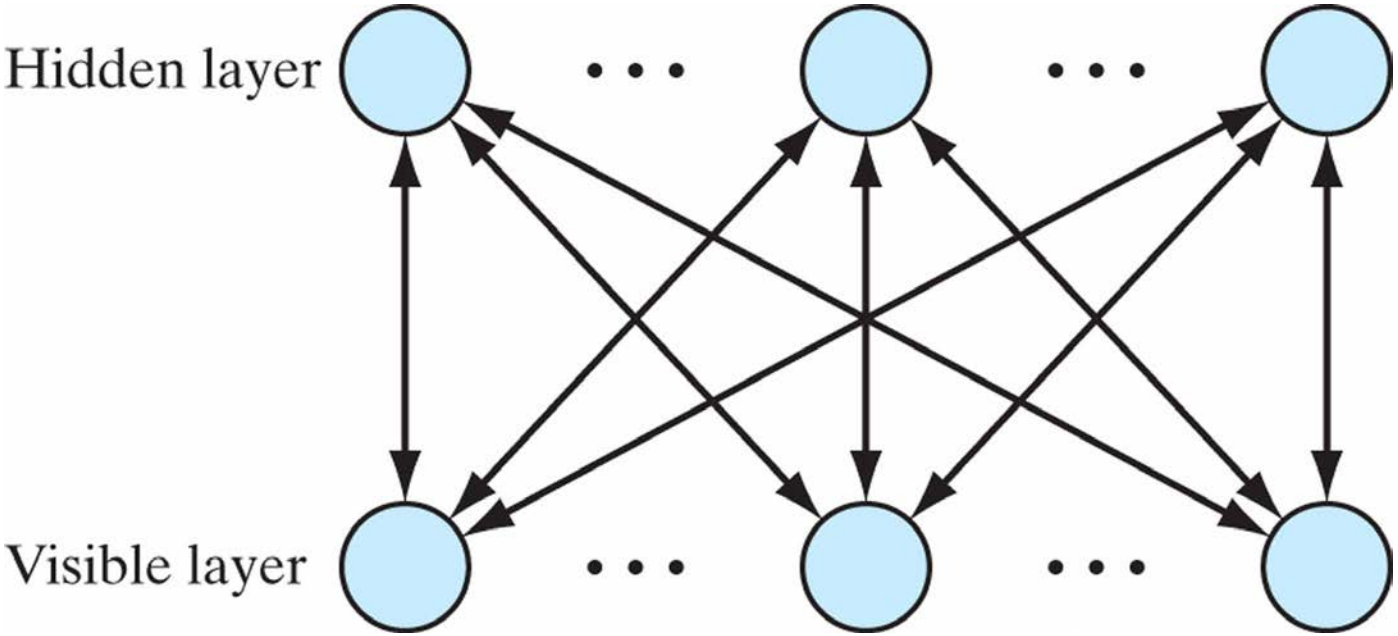
- Because of the tied weights,

$$\frac{\partial L(\mathbf{w})}{\partial w_{ji}} = < h_i^{(0)} \left(v_j^{(0)} - v_j^{(1)} \right) > + < v_j^{(1)} \left(h_i^{(0)} - h_i^{(1)} \right) > +$$
$$< h_i^{(1)} \left(v_j^{(1)} - v_j^{(2)} \right) > \cdots$$

$$= < h_i^{(0)} v_j^{(0)} > - < h_i^{(\infty)} v_j^{(\infty)} >$$

Restricted Boltzmann machines

- A restricted Boltzmann machine (RBM) is a Boltzmann machine with one visible layer and one hidden layer, and no connection within each layer



Energy function of RBM

- The energy function is:

$$E(\mathbf{v}, \mathbf{h}) = -\sum_i \sum_j w_{ji} v_j h_i$$

- Setting $T = 1$, we have

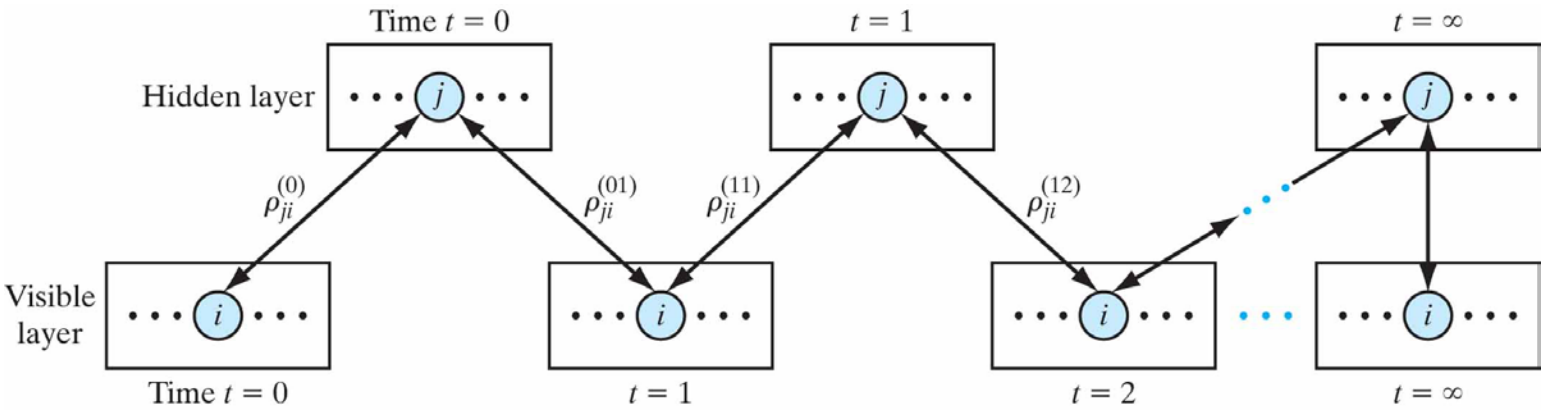
$$\frac{\partial L(\mathbf{w})}{\partial w_{ji}} = \rho_{ji}{}^{+} - \rho_{ji}{}^{-}$$

$$= < h_i{}^{(0)} v_j{}^{(0)} > - < h_i{}^{(\infty)} v_j{}^{(\infty)} >$$

  - The second correlation is computed using alternating Gibbs sampling until thermal equilibrium

Learning in RBM

- This rule is exactly the same as the one for the infinite logistic belief net
  - Hence the equivalence between learning a DBN and an RBM



Contrastive divergence

- In practice, a quick way to learn RBM:

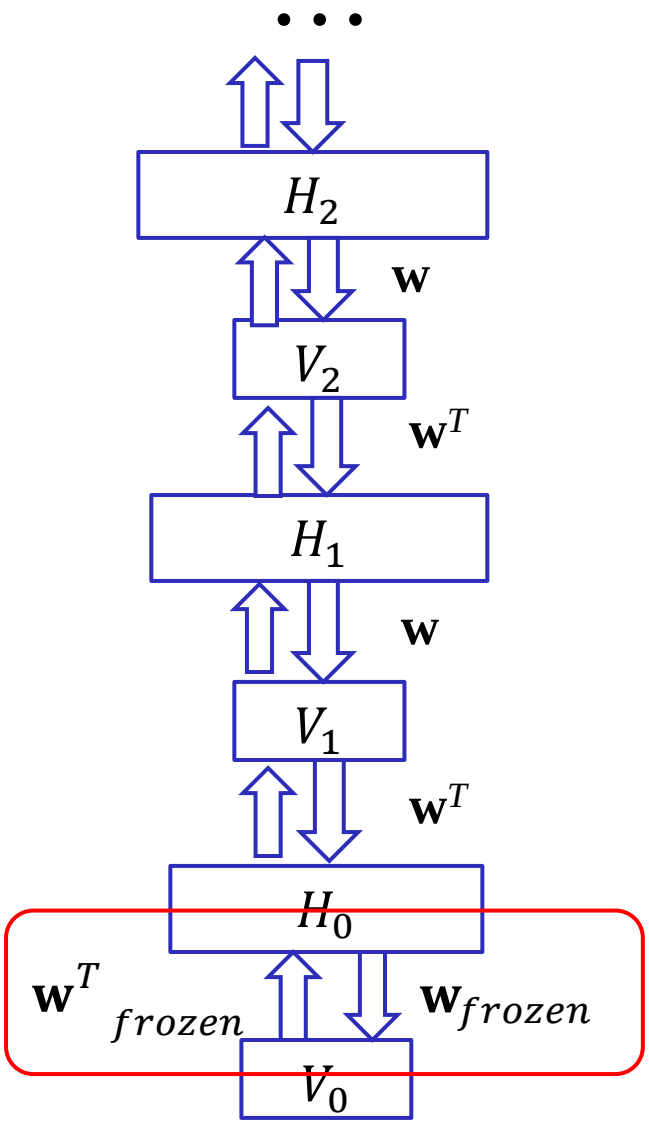
$$\Delta w_{ji} = \eta\left(< h_i^{(0)} v_j^{(0)} > - < h_i^{(1)} v_j^{(1)} >\right)$$

- The above rule is called contrastive divergence

Training a general deep net layer-by-layer

1. First learn **w** with all weights tied
2. Freeze (fix) **w**, which represents the learned weights for the first hidden layer
3. Learn the weights for the second hidden layer by treating responses of the first hidden layer to the training data as "input data"
4. Freeze the weights for the second hidden layer
5. Repeat steps 3-4 as many times as the prescribed number of hidden layers

Illustration



- This training process is the same as repeatedly training RBMs layer-by-layer, which is adopted due to its simplicity

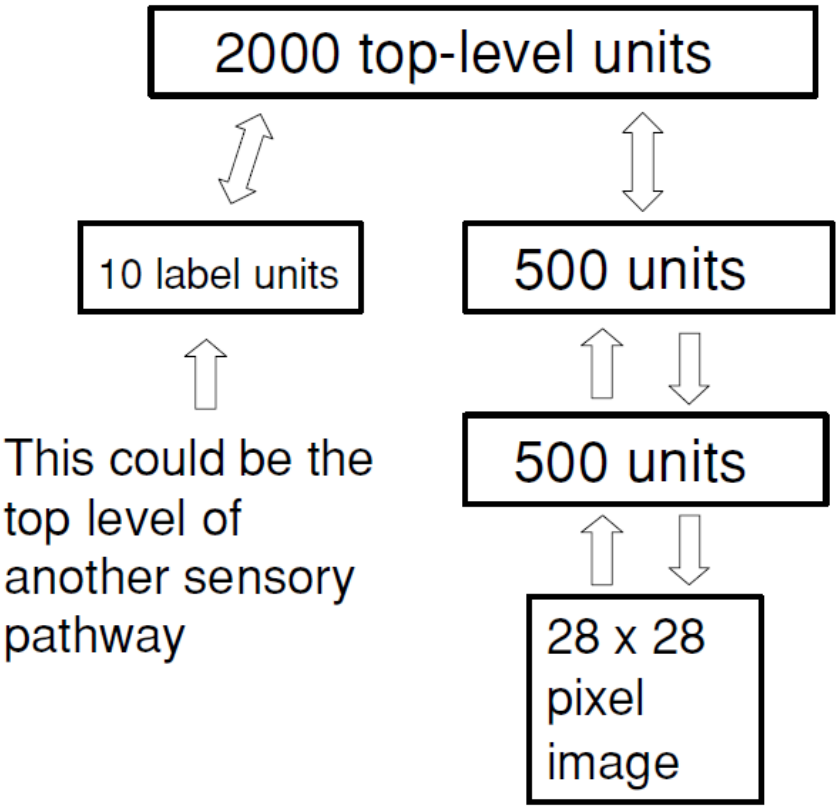
Remarks

- As the number of layers increases, the maximum likelihood approximation of the training data improves
- For discriminative training (e.g. for classification) we add an output layer on top of the learned generative model, and train the entire net by a discriminative algorithm
  - e.g. backprop or SVM
- Although much faster than Boltzmann machines (e.g. no simulated annealing), pretraining is still quite slow, and involves a lot of design as for MLP

Applications

- DNNs have been successfully applied to an increasing number of tasks
- Ex: MNIST handwritten digit recognition
  - A DNN with two hidden layers achieves 1.25% error rate, vs. 1.4% for SVM and 1.5% for MLP

Digit recognition DNN



- The network used to model the joint distribution of digit images and digit labels

Digit recognition illustration

- Samples from the learned generative model

  - Each row shows 10 samples from the generative model with a particular label clamped on. The top-level associative memory is run for 1000 iterations of alternating Gibbs sampling between samples

More samples

- Each row shows 10 samples from the generative model with a particular label clamped on. The top-level associative memory is initialized by an up-pass from a random binary image in which each pixel is on with a probability of 0.5. The first column shows the results of a down-pass from this initial high level state. Subsequent columns are produced by 20 iterations of alternating Gibbs sampling in the associative memory