

## Lecture 6 of 42

# Perceptrons and Winnow

Monday, 11 February 2008

William H. Hsu

Department of Computing and Information Sciences, KSU

<http://www.kddresearch.org/Courses/Spring-2008/CIS732/>

Readings:

Section 6.6, Han & Kamber 2e



CIS 732: Machine Learning and Pattern Recognition

Kansas State University  
Department of Computing and Information Sciences

## Lecture Outline

- Textbook Reading: Sections 4.1-4.4, Mitchell
- Read “The Perceptron”, F. Rosenblatt; “Learning”, M. Minsky and S. Papert
- Next Lecture: 4.5-4.9, Mitchell; “The MLP”, Bishop; Chapter 8, RHW
- This Week’s Paper Review: “Learning by Experimentation”, Mitchell *et al*
- This Month: Numerical Learning Models (e.g., Neural/Bayesian Networks)
- The Perceptron
  - Today: as a linear threshold gate/unit (LTG/LTU)
    - Expressive power and limitations; ramifications
    - Convergence theorem
    - Derivation of a gradient learning algorithm and training (Delta aka LMS) rule
  - Next lecture: as a neural network element (especially in multiple layers)
- The Winnow
  - Another linear threshold model
  - Learning algorithm and training rule



CIS 732: Machine Learning and Pattern Recognition

Kansas State University  
Department of Computing and Information Sciences

## Connectionist (Neural Network) Models

- **Human Brains**
  - Neuron switching time: ~ 0.001 ( $10^{-3}$ ) second
  - Number of neurons: ~10-100 billion ( $10^{10} - 10^{11}$ )
  - Connections per neuron: ~10-100 thousand ( $10^4 - 10^5$ )
  - Scene recognition time: ~0.1 second
  - 100 inference steps doesn't seem sufficient! → highly parallel computation
- **Definitions of Artificial Neural Networks (ANNs)**
  - "... a system composed of many simple processing elements operating in parallel whose function is determined by network structure, connection strengths, and the processing performed at computing elements or nodes." - DARPA (1988)
  - NN FAQ List: <http://www.ci.tuwien.ac.at/docs/services/nnfaq/FAQ.html>
- **Properties of ANNs**
  - Many neuron-like threshold switching units
  - Many weighted interconnections among units
  - Highly parallel, distributed process
  - Emphasis on tuning weights automatically



CIS 732: Machine Learning and Pattern Recognition

Kansas State University  
Department of Computing and Information Sciences

## When to Consider Neural Networks

- **Input: High-Dimensional and Discrete or Real-Valued**
  - e.g., raw sensor input
  - Conversion of symbolic data to quantitative (numerical) representations possible
- **Output: Discrete or Real Vector-Valued**
  - e.g., low-level control policy for a robot actuator
  - Similar qualitative/quantitative (symbolic/numerical) conversions may apply
- **Data: Possibly Noisy**
- **Target Function: Unknown Form**
- **Result: Human Readability Less Important Than Performance**
  - Performance measured purely in terms of accuracy and efficiency
  - Readability: ability to explain inferences made using model; similar criteria
- **Examples**
  - Speech phoneme recognition [Waibel, Lee]
  - Image classification [Kanade, Baluja, Rowley, Frey]
  - Financial prediction



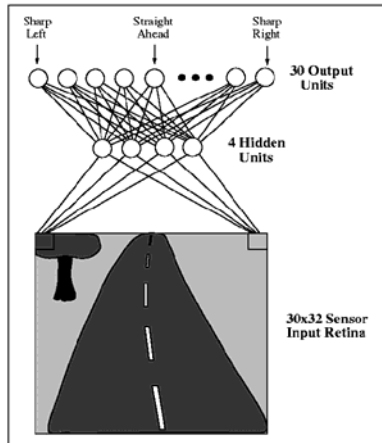
CIS 732: Machine Learning and Pattern Recognition

Kansas State University  
Department of Computing and Information Sciences

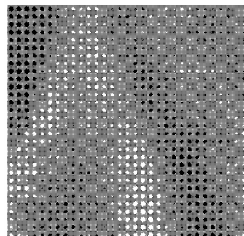
# Autonomous Learning Vehicle in a Neural Net (ALVINN)

- Pomerleau et al

- <http://www.cs.cmu.edu/afs/cs/project/av/member/www/projects/ALVINN.html>
- Drives 70mph on highways



Hidden-to-Output Unit Weight Map (for one hidden unit)

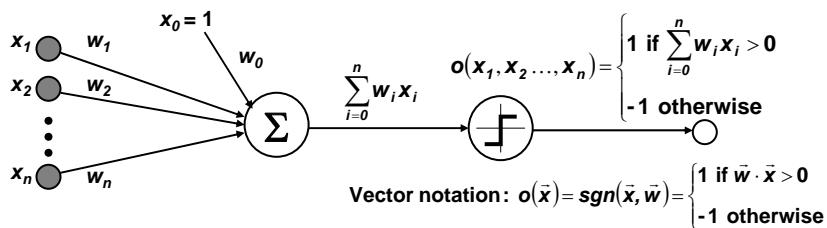


Input-to-Hidden Unit Weight Map (for one hidden unit)



Kansas State University  
Department of Computing and Information Sciences

## The Perceptron



- Perceptron: Single Neuron Model

- aka Linear Threshold Unit (LTU) or Linear Threshold Gate (LTG)
- Net input to unit: defined as linear combination  $net = \sum_{i=0}^n w_i x_i$
- Output of unit: threshold (activation) function on net input (threshold  $\theta = w_0$ )

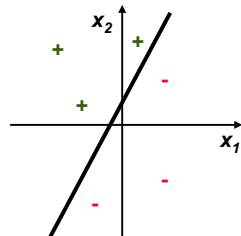
- Perceptron Networks

- Neuron is modeled using a unit connected by weighted links  $w_i$  to other units
- Multi-Layer Perceptron (MLP): next lecture

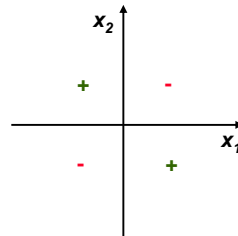


Kansas State University  
Department of Computing and Information Sciences

## Decision Surface of a Perceptron



Example A



Example B

- **Perceptron: Can Represent *Some* Useful Functions**
  - LTU emulation of logic gates (McCulloch and Pitts, 1943)
  - e.g., What weights represent  $g(x_1, x_2) = \text{AND}(x_1, x_2)$ ?  $\text{OR}(x_1, x_2)$ ?  $\text{NOT}(x)$ ?
- **Some Functions *Not* Representable**
  - e.g., not linearly separable
  - Solution: use networks of perceptrons (LTUs)



## Learning Rules for Perceptrons

- **Learning Rule  $\equiv$  Training Rule**
  - Not specific to supervised learning
  - Context: updating a model
- **Hebbian Learning Rule (Hebb, 1949)**
  - Idea: if two units are both active (“firing”), weights between them should increase
  - $w_{ij} = w_{ij} + r o_i o_j$  where  $r$  is a **learning rate** constant
  - Supported by neuropsychological evidence
- **Perceptron Learning Rule (Rosenblatt, 1959)**
  - Idea: when a target output value is provided for a single neuron with fixed input, it can incrementally update weights to learn to produce the output
  - Assume binary (boolean-valued) input/output units; single LTU
  - $w_i \leftarrow w_i + \Delta w_i$
  - $\Delta w_i = r(t - o)x_i$
  - where  $t = c(x)$  is target output value,  $o$  is perceptron output,  $r$  is small learning rate constant (e.g., 0.1)
  - Can prove convergence if  $D$  linearly separable and  $r$  small enough



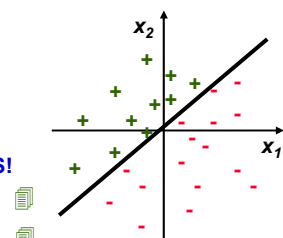
## Perceptron Learning Algorithm

- **Simple Gradient Descent Algorithm**
  - Applicable to concept learning, symbolic learning (with proper representation)
- **Algorithm *Train-Perceptron* ( $D \equiv \{ \langle x, t(x) \equiv c(x) \rangle \}$ )**
  - Initialize all weights  $w_i$  to random values
  - WHILE not all examples correctly predicted DO
    - FOR each training example  $x \in D$ 
      - Compute current output  $o(x)$
      - FOR  $i = 1$  to  $n$ 
        - $w_i \leftarrow w_i + r(t - o)x_i$  // perceptron learning rule
- **Perceptron Learnability**
  - Recall: can only learn  $h \in H$  - i.e., linearly separable (LS) functions
  - Minsky and Papert, 1969: demonstrated representational limitations
    - e.g., parity ( $n$ -attribute XOR:  $x_1 \oplus x_2 \oplus \dots \oplus x_n$ )
    - e.g., symmetry, connectedness in visual pattern recognition
    - Influential book *Perceptrons* discouraged ANN research for ~10 years
  - NB: \$64K question - "Can we transform learning problems into LS ones?"



## Linear Separators

- **Functional Definition**
  - $f(x) = 1$  if  $w_1x_1 + w_2x_2 + \dots + w_nx_n \geq \theta$ , 0 otherwise
  - $\theta$ : threshold value
- **Linearly Separable Functions**
  - NB:  $D$  is LS does not necessarily imply  $c(x) = f(x)$  is LS!
  - Disjunctions:  $c(x) = x_1' \vee x_2' \vee \dots \vee x_m'$
  - $m$  of  $n$ :  $c(x) =$  at least 3 of  $(x_1', x_2', \dots, x_m')$
  - Exclusive OR (XOR):  $c(x) = x_1 \oplus x_2$
  - General DNF:  $c(x) = T_1 \vee T_2 \vee \dots \vee T_m$ ;  $T_i = I_1 \wedge I_1 \wedge \dots \wedge I_k$
- **Change of Representation Problem**
  - Can we transform non-LS problems into LS ones?
  - Is this meaningful? Practical?
  - Does it represent a significant fraction of real-world problems?



Linearly Separable (LS) Data Set



## Perceptron Convergence

- **Perceptron Convergence Theorem**
  - **Claim:** If there exist a set of weights that are consistent with the data (i.e., the data is linearly separable), the perceptron learning algorithm will converge
  - **Proof:** well-founded ordering on search region (“wedge width” is strictly decreasing) - see Minsky and Papert, 11.2-11.3
  - **Caveat 1:** How long will this take?
  - **Caveat 2:** What happens if the data is *not* LS?
- **Perceptron Cycling Theorem**
  - **Claim:** If the training data is not LS the perceptron learning algorithm will eventually repeat the same set of weights and thereby enter an infinite loop
  - **Proof:** bound on number of weight changes until repetition; induction on  $n$ , the dimension of the training example vector - MP, 11.10
- **How to Provide More Robustness, Expressivity?**
  - **Objective 1:** develop algorithm that will find closest approximation (today)
  - **Objective 2:** develop architecture to overcome representational limitation (next lecture)

CIS 732: Machine Learning and Pattern Recognition

Kansas State University  
Department of Computing and Information Sciences



## Gradient Descent: Principle

- **Understanding Gradient Descent for Linear Units**
  - Consider simpler, unthresholded linear unit:
$$o(\vec{x}) = \text{net}(\vec{x}) = \sum_{i=0}^n w_i x_i$$
  - **Objective:** find “best fit” to  $D$
- **Approximation Algorithm**
  - **Quantitative objective:** minimize error over training data set  $D$
  - **Error function:** sum squared error (SSE)
$$E[\vec{w}] = \text{error}_D[\vec{w}] = \frac{1}{2} \sum_{\mathbf{x} \in D} (t(\mathbf{x}) - o(\mathbf{x}))^2$$
- **How to Minimize?**
  - Simple optimization
  - Move in direction of steepest gradient in weight-error space
    - Computed by finding tangent
    - i.e. partial derivatives (of  $E$ ) with respect to weights ( $w_i$ )

CIS 732: Machine Learning and Pattern Recognition

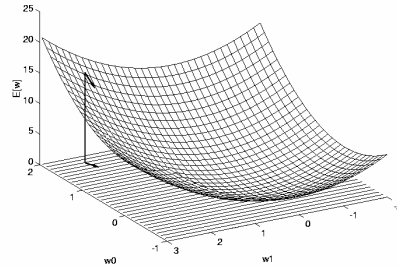
Kansas State University  
Department of Computing and Information Sciences



## Gradient Descent: Derivation of Delta/LMS (Widrow-Hoff) Rule

- **Definition: Gradient**

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



- **Modified Gradient Descent Training Rule**

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

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

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \left[ \frac{1}{2} \sum_{x \in D} (t(x) - o(x))^2 \right] = \frac{1}{2} \sum_{x \in D} \left[ \frac{\partial}{\partial w_i} (t(x) - o(x))^2 \right]$$

$$= \frac{1}{2} \sum_{x \in D} \left[ 2(t(x) - o(x)) \frac{\partial}{\partial w_i} (t(x) - o(x)) \right] = \sum_{x \in D} \left[ (t(x) - o(x)) \frac{\partial}{\partial w_i} (t(x) - \vec{w} \cdot \vec{x}) \right]$$

$$\frac{\partial E}{\partial w_i} = \sum_{x \in D} [(t(x) - o(x))(-x_i)]$$



## Gradient Descent: Algorithm using Delta/LMS Rule

- **Algorithm Gradient-Descent ( $D, r$ )**

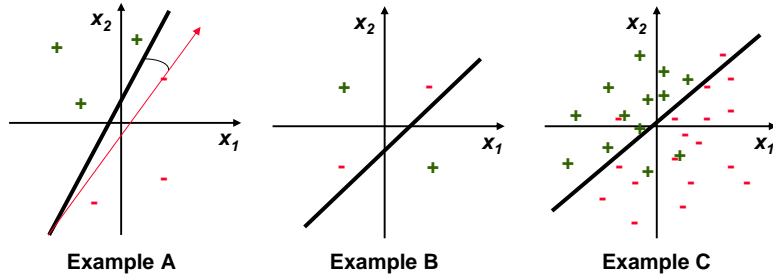
- Each training example is a pair of the form  $\langle x, t(x) \rangle$ , where  $x$  is the vector of input values and  $t(x)$  is the output value.  $r$  is the learning rate (e.g., 0.05)
- Initialize all weights  $w_i$  to (small) random values
- UNTIL the termination condition is met, DO
  - Initialize each  $\Delta w_i$  to zero
  - FOR each  $\langle x, t(x) \rangle$  in  $D$ , DO
    - Input the instance  $x$  to the unit and compute the output  $o$
    - FOR each linear unit weight  $w_i$ , DO
      - $\Delta w_i \leftarrow \Delta w_i + r(t - o)x_i$
      - $w_i \leftarrow w_i + \Delta w_i$
- RETURN final  $w$

- **Mechanics of Delta Rule**

- Gradient is based on a derivative
- Significance: later, will use nonlinear activation functions (aka transfer functions, squashing functions)



## Gradient Descent: Perceptron Rule versus Delta/LMS Rule



- **LS Concepts: Can Achieve Perfect Classification**
  - Example A: perceptron training rule converges
- **Non-LS Concepts: Can Only Approximate**
  - Example B: not LS; delta rule converges, but can't do better than 3 correct
  - Example C: not LS; better results from delta rule
- **Weight Vector  $w = \text{Sum of Misclassified } x \in D$** 
  - Perceptron: minimize  $w$
  - Delta Rule: minimize error  $\equiv$  distance from separator (i.e., maximize  $\frac{\partial E}{\partial \bar{w}}$ )

CIS 732: Machine Learning and Pattern Recognition

Kansas State University  
 Department of Computing and Information Sciences

## Incremental (Stochastic) Gradient Descent

- **Batch Mode Gradient Descent**
  - UNTIL the termination condition is met, DO
    1. Compute the gradient  $\nabla E_D[\bar{w}]$
    2.  $\bar{w} \leftarrow \bar{w} - r \nabla E_D[\bar{w}]$
  - RETURN final  $w$
- **Incremental (Online) Mode Gradient Descent**
  - UNTIL the termination condition is met, DO
    - FOR each  $\langle x, t(x) \rangle$  in  $D$ , DO
      1. Compute the gradient  $\nabla E_d[\bar{w}]$
      2.  $\bar{w} \leftarrow \bar{w} - r \nabla E_d[\bar{w}]$
  - RETURN final  $w$
- **Emulating Batch Mode**
  - $E_D[\bar{w}] \equiv \frac{1}{2} \left[ \sum_{x \in D} (t(x) - o(x))^2 \right]$ ,  $E_d[\bar{w}] \equiv \frac{1}{2} (t(x) - o(x))^2$
  - Incremental gradient descent can approximate batch gradient descent arbitrarily closely if  $r$  made small enough

CIS 732: Machine Learning and Pattern Recognition

Kansas State University  
 Department of Computing and Information Sciences

## Learning Disjunctions

- **Hidden Disjunction to Be Learned**
  - $c(x) = x_1' \vee x_2' \vee \dots \vee x_m'$  (e.g.,  $x_2 \vee x_4 \vee x_5 \dots \vee x_{100}$ )
  - Number of disjunctions:  $3^n$  (each  $x_i$ : included, negation included, or excluded)
  - *Change of representation*: can turn into a monotone disjunctive formula?
    - How?
    - How many disjunctions then?
  - Recall from COLT: mistake bounds
    - $\log(|C|) = O(n)$
    - Elimination algorithm makes  $O(n)$  mistakes
- **Many Irrelevant Attributes**
  - Suppose only  $k \ll n$  attributes occur in disjunction  $c$  - i.e.,  $\log(|C|) = O(k \log n)$
  - Example: learning natural language (e.g., learning over text)
  - Idea: use a **Winnow** - perceptron-type LTU model (Littlestone, 1988)
    - Strengthen weights for false positives
    - Learn from negative examples too: weaken weights for false negatives

CIS 732: Machine Learning and Pattern Recognition

Kansas State University  
Department of Computing and Information Sciences



## Winnow Algorithm

- **Algorithm *Train-Winnow* ( $D$ )**
  - Initialize:  $\theta = n, w_i = 1$
  - UNTIL the termination condition is met, DO
    - FOR each  $\langle x, t(x) \rangle$  in  $D$ , DO
      1. CASE 1: no mistake - do nothing
      2. CASE 2:  $t(x) = 1$  but  $w \cdot x < \theta - w_i \leftarrow 2w_i$  if  $x_i = 1$  (promotion/strengthening)
      3. CASE 3:  $t(x) = 0$  but  $w \cdot x \geq \theta - w_i \leftarrow w_i / 2$  if  $x_i = 1$  (demotion/weakening)
  - RETURN final  $w$
- **Winnow Algorithm Learns Linear Threshold (LT) Functions**
- **Converting to Disjunction Learning**
  - Replace demotion with elimination
  - Change weight values to 0 instead of halving
  - Why does this work?

CIS 732: Machine Learning and Pattern Recognition

Kansas State University  
Department of Computing and Information Sciences



## Winnow : An Example

- $f(x) \equiv c(x) = x_1 \vee x_2 \vee x_{1023} \vee x_{1024}$ 
  - Initialize:  $\theta = n = 1024, w = (1, 1, 1, \dots, 1)$
  - $\langle (1, 1, 1, \dots, 1), + \rangle$        $w \bullet x \geq \theta, w = (1, 1, 1, \dots, 1)$       **OK**
  - $\langle (0, 0, 0, \dots, 0), - \rangle$        $w \bullet x < \theta, w = (1, 1, 1, \dots, 1)$       **OK**
  - $\langle (0, 0, 1, 1, 1, \dots, 0), - \rangle$        $w \bullet x < \theta, w = (1, 1, 1, \dots, 1)$       **OK**
  - $\langle (1, 0, 0, \dots, 0), + \rangle$        $w \bullet x < \theta, w = (2, 1, 1, \dots, 1)$       **mistake**
  - $\langle (1, 0, 1, 1, 0, \dots, 0), + \rangle$        $w \bullet x < \theta, w = (4, 1, 2, 2, \dots, 1)$       **mistake**
  - $\langle (1, 0, 1, 0, 0, \dots, 1), + \rangle$        $w \bullet x < \theta, w = (8, 1, 4, 2, \dots, 2)$       **mistake**
  - ...       $w = (512, 1, 256, 256, \dots, 256)$
- **Promotions for each good variable:**  $\lceil \lg(n) \rceil < \lg(n) + 1 = \lg(2n)$ 
  - $\langle (1, 0, 1, 0, 0, \dots, 1), + \rangle$        $w \bullet x \geq \theta, w = (512, 1, 256, 256, \dots, 256)$       **OK**
  - $\langle (0, 0, 1, 0, \underline{1}, \underline{1}, \underline{1}, \dots, 0), - \rangle$        $w \bullet x \geq \theta, w = (512, 1, 0, 256, \underline{0}, \underline{0}, \underline{0}, \dots, 256)$       **mistake**
  - Last example: elimination rule (bit mask)
- **Final Hypothesis:**  $w = (1024, 1024, 0, 0, 0, 1, 32, \dots, 1024, 1024)$



## Winnow: Mistake Bound

- **Claim:** *Train-Winnow* makes  $O(k \log n)$  mistakes on  $k$ -disjunctions ( $\leq k$  of  $n$ )
- **Proof**
  - $u \equiv$  number of mistakes on positive examples (**promotions**)
  - $v \equiv$  number of mistakes on negative examples (**demotions/eliminations**)
  - **Lemma 1:**  $u < k \lg(2n) = k(\lg n + 1) = k \lg n + k = O(k \log n)$
  - **Proof**
    - A weight that corresponds to a good variable is only promoted
    - When these weights reach  $n$  there will be **no more false positives**
  - **Lemma 2:**  $v < 2(u + 1)$
  - **Proof**
    - Total weight  $W = n$  initially
    - False positive:  $W(t+1) < W(t) + n$  - in worst case, every variable promoted
    - False negative:  $W(t+1) < W(t) - n/2$  - elimination of a bad variable
    - $0 < W < n + un - vn/2 \Rightarrow v < 2(u + 1)$
  - Number of mistakes:  $u + v < 3u + 2 = O(k \log n)$ , Q.E.D.



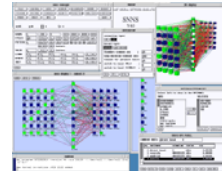
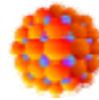
## Extensions to Winnow

- **Train-Winnow Learns Monotone Disjunctions**
  - *Change of representation*: can convert a general disjunctive formula
    - Duplicate each variable:  $x \rightarrow \{y_+, y_-\}$
    - $y_+$  denotes  $x$ ;  $y_-$  denotes  $\neg x$
  - $2n$  variables - but *can now learn general disjunctions!*
  - NB: we're not finished
    - $\{y_+, y_-\}$  are coupled
    - Need to keep two weights for each (original) variable and update both (how?)
- **Robust Winnow**
  - Adversarial game: may change  $c$  by adding (at cost 1) or deleting a variable  $x$
  - Learner: makes prediction, then is told correct answer
  - *Train-Winnow-R*: same as *Train-Winnow*, but with lower weight bound of  $1/2$
  - Claim: *Train-Winnow-R* makes  $O(k \log n)$  mistakes ( $k$  = total cost of adversary)
  - Proof: generalization of previous claim



## NeuroSolutions and SNNS

- **NeuroSolutions 3.0 Specifications**
  - Commercial ANN simulation environment (<http://www.nd.com>) for Windows NT
  - Supports multiple ANN architectures and training algorithms (temporal, modular)
  - Produces embedded systems
    - Extensive data handling and visualization capabilities
    - Fully modular (object-oriented) design
    - Code generation and dynamic link library (DLL) facilities
  - Benefits
    - Portability, parallelism: code tuning; fast offline learning
    - Dynamic linking: extensibility for research and development
- **Stuttgart Neural Network Simulator (SNNS) Specifications**
  - Open source ANN simulation environment for Linux
  - <http://www.informatik.uni-stuttgart.de/ipvr/bv/projekte/snns/>
  - Supports multiple ANN architectures and training algorithms
  - Very extensive visualization facilities
  - Similar portability and parallelization benefits



## Terminology

- **Neural Networks (NNs): Parallel, Distributed Processing Systems**
  - Biological NNs and artificial NNs (ANNs)
  - Perceptron aka Linear Threshold Gate (LTG), Linear Threshold Unit (LTU)
    - Model neuron
    - Combination and activation (transfer, squashing) functions
- **Single-Layer Networks**
  - Learning rules
    - Hebbian: strengthening connection weights when both endpoints activated
    - Perceptron: minimizing total weight contributing to errors
    - Delta Rule (LMS Rule, Widrow-Hoff): minimizing sum squared error
    - Winnow: minimizing classification mistakes on LTU with multiplicative rule
  - Weight update regime
    - Batch mode: cumulative update (all examples at once)
    - Incremental mode: non-cumulative update (one example at a time)
- **Perceptron Convergence Theorem and Perceptron Cycling Theorem**



CIS 732: Machine Learning and Pattern Recognition

Kansas State University  
Department of Computing and Information Sciences

## Summary Points

- **Neural Networks: Parallel, Distributed Processing Systems**
  - Biological and artificial (ANN) types
  - Perceptron (LTU, LTG): model neuron
- **Single-Layer Networks**
  - Variety of update rules
    - Multiplicative (Hebbian, Winnow), additive (gradient: Perceptron, Delta Rule)
    - Batch versus incremental mode
  - Various convergence and efficiency conditions
  - Other ways to learn linear functions
    - Linear programming (general-purpose)
    - Probabilistic classifiers (some assumptions)
- **Advantages and Disadvantages**
  - “Disadvantage” (tradeoff): simple and restrictive
  - “Advantage”: perform well on many realistic problems (e.g., some text learning)
- **Next: Multi-Layer Perceptrons, Backpropagation, ANN Applications**



CIS 732: Machine Learning and Pattern Recognition

Kansas State University  
Department of Computing and Information Sciences