Lecture 35 of 42

Machine Learning: Artificial Neural Networks Discussion: Feedforward ANNs & Backprop

William H. Hsu

Department of Computing and Information Sciences, KSU

KSOL course page: http://snipurl.com/v9v3
Course web site: http://www.kddresearch.org/Courses/CIS730
Instructor home page: http://www.cis.ksu.edu/~bhsu

Reading for Next Class:

Chapter 20, Russell and Norvig

Illustrative Example

- **Training Examples for Concept *PlayTennis***

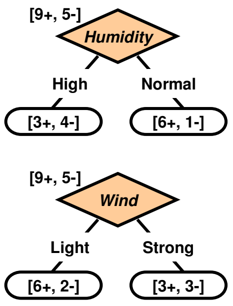
Day	Outlook	Temperature	Humidity	Wind	*PlayTennis*?
1	Sunny	Hot	High	Light	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Light	Yes
4	Rain	Mild	High	Light	Yes
5	Rain	Cool	Normal	Light	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Light	No
9	Sunny	Cool	Normal	Light	Yes
10	Rain	Mild	Normal	Light	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Light	Yes
14	Rain	Mild	High	Strong	No

- ***ID3* ≡ *Build-DT* using *Gain*(•)**
- **How Will *ID3* Construct A Decision Tree?**

Constructing Decision Tree For *PlayTennis* using ID3 [1]

- **Selecting The Root Attribute**

Day	Outlook	Temperature	Humidity	Wind	*PlayTennis?*
1	Sunny	Hot	High	Light	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Light	Yes
4	Rain	Mild	High	Light	Yes
5	Rain	Cool	Normal	Light	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Light	No
9	Sunny	Cool	Normal	Light	Yes
10	Rain	Mild	Normal	Light	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Light	Yes
14	Rain	Mild	High	Strong	No



- **Prior (unconditioned) distribution: 9+, 5-**
  - H(D) = -(9/14) lg (9/14) - (5/14) lg (5/14) bits = 0.94 bits
  - H(D, Humidity = High) = -(3/7) lg (3/7) - (4/7) lg (4/7) = 0.985 bits
  - H(D, Humidity = Normal) = -(6/7) lg (6/7) - (1/7) lg (1/7) = 0.592 bits
  - Gain(D, Humidity) = 0.94 - (7/14) * 0.985 + (7/14) * 0.592 = 0.151 bits
  - Similarly, Gain (D, Wind) = 0.94 - (8/14) * 0.811 + (6/14) * 1.0 = 0.048 bits

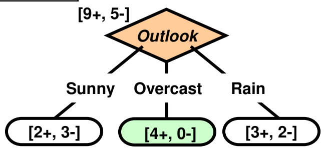
$$Gain(D, A) \equiv -H(D) - \sum_{v \in values(A)} \left[\frac{|D_v|}{|D|} \bullet H(D_v) \right]$$

Constructing Decision Tree For *PlayTennis* using ID3 [2]

- **Selecting The Root Attribute**

Day	Outlook	Temperature	Humidity	Wind	*PlayTennis?*
1	Sunny	Hot	High	Light	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Light	Yes
4	Rain	Mild	High	Light	Yes
5	Rain	Cool	Normal	Light	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Light	No
9	Sunny	Cool	Normal	Light	Yes
10	Rain	Mild	Normal	Light	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Light	Yes
14	Rain	Mild	High	Strong	No

  - *Gain*(*D*, *Humidity*) = 0.151 bits
  - *Gain*(*D*, *Wind*) = 0.048 bits
  - *Gain*(*D*, *Temperature*) = 0.029 bits
  - *Gain*(*D*, *Outlook*) = 0.246 bits



- **Selecting The Next Attribute (Root of Subtree)**
  - Continue until every example is included in path or purity = 100%
  - What does purity = 100% mean?
  - Can *Gain*(*D*, *A*) < 0?

Constructing Decision Tree For *PlayTennis* using ID3 [3]

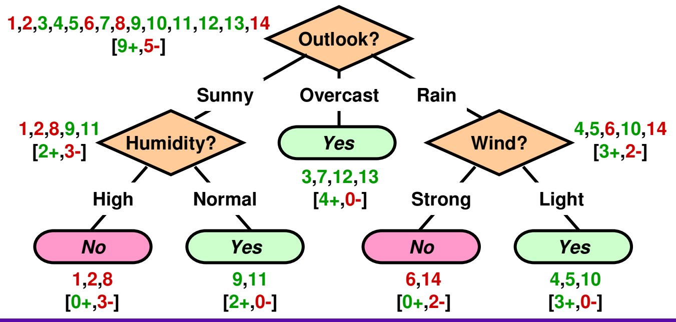
- **Selecting The Next Attribute (Root of Subtree)**

Day	Outlook	Temperature	Humidity	Wind	*PlayTennis*?
1	Sunny	Hot	High	Light	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Light	Yes
4	Rain	Mild	High	Light	Yes
5	Rain	Cool	Normal	Light	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Light	No
9	Sunny	Cool	Normal	Light	Yes
10	Rain	Mild	Normal	Light	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Light	Yes
14	Rain	Mild	High	Strong	No

  - **Convention: lg (0/a) = 0**
  - ***Gain*(D_{Sunny}, *Humidity*) = 0.97 - (3/5) * 0 - (2/5) * 0 = 0.97 bits**
  - ***Gain*(D_{Sunny}, *Wind*) = 0.97 - (2/5) * 1 - (3/5) * 0.92 = 0.02 bits**
  - ***Gain*(D_{Sunny}, *Temperature*) = 0.57 bits**
- **Top-Down Induction**
  - **For discrete-valued attributes, terminates in O(*n*) splits**
  - **Makes at most one pass through data set at each level (why?)**

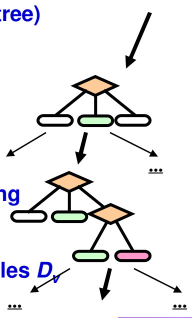
Constructing Decision Tree For *PlayTennis* using ID3 [4]

Day	Outlook	Temperature	Humidity	Wind	*PlayTennis*?
1	Sunny	Hot	High	Light	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Light	Yes
4	Rain	Mild	High	Light	Yes
5	Rain	Cool	Normal	Light	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Light	No
9	Sunny	Cool	Normal	Light	Yes
10	Rain	Mild	Normal	Light	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Light	Yes
14	Rain	Mild	High	Strong	No



Hypothesis Space Search In ID3

- **Search Problem**
  - Conduct a search of the ***space of decision trees***, which can represent all possible discrete functions
    - Pros: expressiveness; flexibility
    - Cons: computational complexity; large, incomprehensible trees (next time)
  - Objective: to find the best decision tree (minimal consistent tree)
  - Obstacle: finding this tree is NP-hard
  - Tradeoff
    - Use heuristic (figure of merit that guides search)
    - Use greedy algorithm
    - *Aka* hill-climbing (gradient "descent") without backtracking
- **Statistical Learning**
  - Decisions based on statistical descriptors p_+, p_- for subsamples D_v
  - In ID3, *all data used*
  - Robust to noisy data



Inductive Bias in ID3 (& C4.5 / J48)

- **Heuristic : Search :: Inductive Bias : Inductive Generalization**
  - *H* is the power set of instances in *X*
  - ⇒ Unbiased? Not really…
    - Preference for short trees (termination condition)
    - Preference for trees with high information gain attributes near the root
    - *Gain*(•): a heuristic function that *captures the inductive bias of ID3*
  - Bias in *ID3*
    - Preference for some hypotheses is encoded in heuristic function
    - Compare: a restriction of hypothesis space *H* (previous discussion of propositional normal forms: *k*-CNF, etc.)
- **Preference for Shortest Tree**
  - Prefer shortest tree that fits the data
  - An Occam's Razor bias: shortest hypothesis that explains the observations

Terminology

- Decision Trees (DTs)
  - Boolean DTs: target concept is binary-valued (i.e., Boolean-valued)
  - Building DTs
    - Histogramming: method of vector quantization (encoding input using bins)
    - Discretization: continuous input into discrete (*e.g.*, histogramming)
- Entropy and Information Gain
  - Entropy *H*(*D*) for data set *D* relative to implicit concept *c*
  - Information gain *Gain* (*D*, *A*) for data set partitioned by attribute *A*
  - Impurity, uncertainty, irregularity, surprise *vs.* purity, certainty, regularity, redundancy
- Heuristic Search
  - Algorithm *Build-DT*: greedy search (hill-climbing without backtracking)
  - *ID3* as *Build-DT* using the heuristic *Gain*(•)
  - Heuristic : Search :: Inductive Bias : Inductive Generalization
- *MLC++* (Machine Learning Library in C++)
  - Data mining libraries (e.g., *MLC++*) and packages (e.g., *MineSet*)
  - Irvine Database: the Machine Learning Database Repository at UCI

Summary Points

- Decision Trees (DTs)
  - Can be boolean ($c(x) \in \{+, -\}$) or range over multiple classes
  - When to use DT-based models
- Generic Algorithm *Build-DT*: Top Down Induction
  - Calculating best attribute upon which to split
  - Recursive partitioning
- Entropy and Information Gain
  - Goal: to measure *uncertainty removed* by splitting on a candidate attribute *A*
    - Calculating information gain (change in entropy)
    - Using information gain in construction of tree
  - *ID3* ≡ *Build-DT* using *Gain*(•)
- ID3 as Hypothesis Space Search (in State Space of Decision Trees)
- Heuristic Search and Inductive Bias
- Data Mining using *MLC++* (Machine Learning Library in C++)
- Next: More Biases (Occam’s Razor); Managing DT Induction

Connectionist (Neural Network) Models

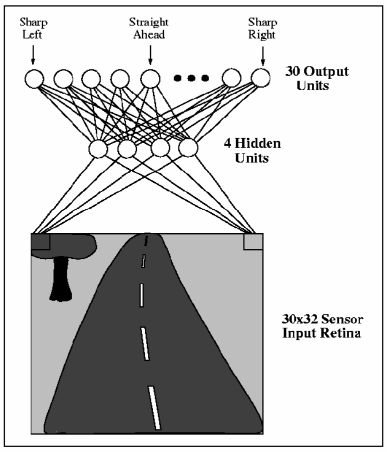
- **Human Brains**
  - Neuron switching time: ~ 0.001 (10^{-3}) second
  - Number of neurons: ~10-100 billion ($10^{10} – 10^{11}$)
  - Connections per neuron: ~10-100 thousand ($10^4 – 10^5$)
  - Scene recognition time: ~0.1 second
  - 100 inference steps doesn't seem sufficient! $\rightarrow$ highly 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

When to Consider Neural Networks

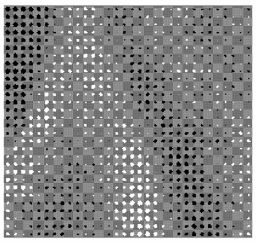
- **Input: High-Dimensional and Discrete or Real-Valued**
  - e.g., raw sensor input
  - Conversion of symbolic data to quantitative (numerical) representations possible
- **Output: Discrete or Real Vector-Valued**
  - e.g., low-level control policy for a robot actuator
  - Similar qualitative/quantitative (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

Autonomous Learning Vehicle In a Neural Net (ALVINN)

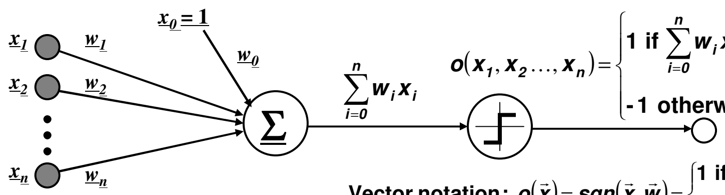
- Pomerleau *et al*
  - http://www.cs.cmu.edu/afs/cs/project/alv/member/www/projects/ALVINN.html
  - Drives 70mph on highways



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


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

The Perceptron

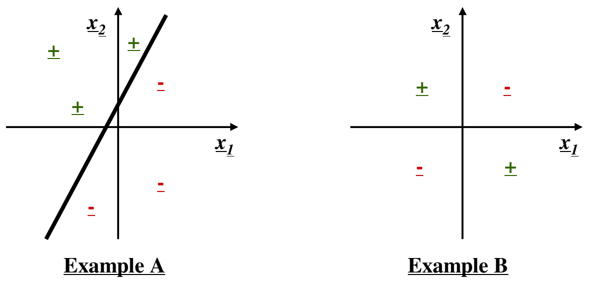


$$o(x_1, x_2 \ldots, x_n) = \begin{cases} 1 \text{ if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 \text{ otherwise} \end{cases}$$

Vector notation: $o(\vec{x}) = sgn(\vec{x}, \vec{w}) = \begin{cases} 1 \text{ if } \vec{w} \cdot \vec{x} > 0 \\ -1 \text{ otherwise} \end{cases}$

- 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

Decision Surface of a Perceptron



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

Learning Rules for Perceptrons

- Learning Rule ≡ 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

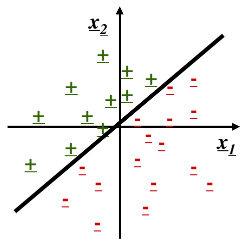
- **Simple Gradient Descent Algorithm**
  - Applicable to concept learning, symbolic learning (with proper representation)
- **Algorithm *Train-Perceptron* ($D \equiv \{<x, t(x) \equiv c(x)>\}$)**
  - Initialize all weights w_i to random values
  - WHILE not all examples correctly predicted DO

```
    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: $x_1 \oplus x_2 \oplus \ldots \oplus x_n$)
    - 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 $w_1x_1 + w_2x_2 + \ldots + w_nx_n \geq \theta$, 0 otherwise
  - θ: threshold value
- **Linearly Separable Functions**
  - NB: D is LS does not necessarily imply $c(x) = f(x)$ is LS!
  - Disjunctions: $c(x) = x_1' \vee x_2' \vee \ldots \vee x_m'$
  - m of n: $c(x)$ = at least 3 of (x_1', x_2', …, x_m')
  - Exclusive *OR* (*XOR*): $c(x) = x_1 \oplus x_2$
  - General DNF: $c(x) = T_1 \vee T_2 \vee \ldots \vee T_m$; $T_i = l_1 \wedge l_1 \wedge \ldots \wedge l_k$
- **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)**

Gradient Descent: Principle

- **Understanding Gradient Descent for Linear Units**
  - **Consider simpler, unthresholded linear unit:**

$$o(\vec{x}) = net(\vec{x}) = \sum_{i=0}^{n} w_i x_i$$

  - **Objective: find "best fit" to *D***
- **Approximation Algorithm**
  - **Quantitative objective: minimize error over training data set *D***
  - **Error function: sum squared error (SSE)**

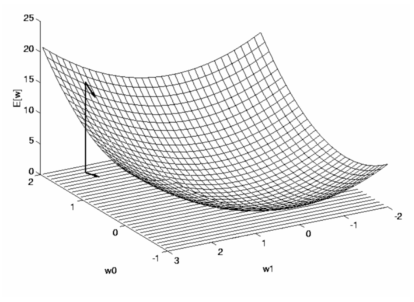
$$E[\vec{w}] = error_D[\vec{w}] = \frac{1}{2}\sum_{x \in D}(t(x) - o(x))^2$$

- **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)**

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

- **Definition: Gradient**

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



- **Modified Gradient Descent Training Rule**

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

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

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

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

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

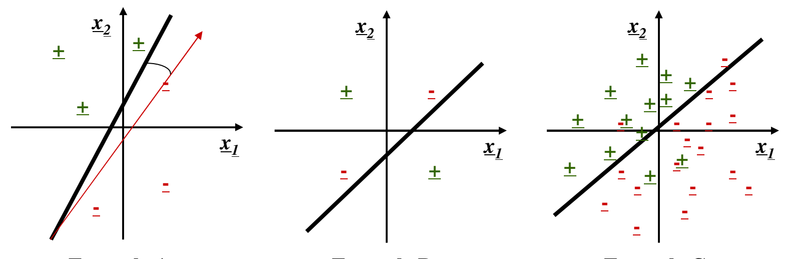
Gradient Descent: Algorithm using Delta/LMS Rule

- **Algorithm *Gradient-Descent* (*D*, *r*)**
  - Each training example is a pair of the form <*x*, *t*(*x*)>, 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 <$x$, $t(x)$> 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



Example A Example B Example C

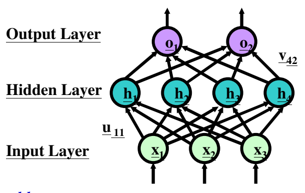
- LS Concepts: Can Achieve Perfect Classification
  - Example A: perceptron training rule converges
- Non-LS Concepts: Can Only Approximate
  - Example B: not LS; delta rule converges, but can't do better than 3 correct
  - Example C: not LS; better results from delta rule
- Weight Vector w = Sum of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

Review: Backprop, Feedforward

- Intuitive Idea: Distribute *Blame* for Error to Previous Layers
- Algorithm *Train-by-Backprop* (*D*, *r*)
  - Each training example is a pair of the form <*x*, *t*(*x*)>, 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

```
FOR each <x, t(x)> in D, DO
    Input the instance x to the unit and compute the output $o(x) = \sigma(net(x))$
    FOR each output unit k, DO
        $\delta_k = o_k(x)(1 - o_k(x))(t_k(x) - o_k(x))$
    FOR each hidden unit j, DO
        $\delta_j = h_j(x)(1 - h_j(x)) \sum_{k \in outputs} v_{j,k}\delta_j$
    Update each $w = u_{i,j}$ $(a = h_j)$ or $w = v_{j,k}$ $(a = o_k)$
        $w_{start\text{-}layer,\ end\text{-}layer} \leftarrow w_{start\text{-}layer,\ end\text{-}layer} + \Delta w_{start\text{-}layer,\ end\text{-}layer}$
        $\Delta w_{start\text{-}layer,\ end\text{-}layer} \leftarrow r\,\delta_{end\text{-}layer}\,a_{end\text{-}layer}$
```

  - RETURN final *u*, *v*



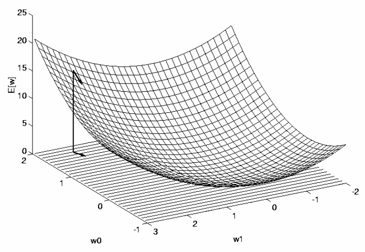
Review: Derivation of Backprop

- **Recall: Gradient of Error Function** $\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n}\right]$
- **Gradient of Sigmoid Activation Function**

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

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

$$= -\sum_{\langle \vec{x}, t(\vec{x})\rangle \in D}\left[\left(t(\vec{x}) - o(\vec{x})\right)\frac{\partial o(\vec{x})}{\partial net(\vec{x})}\frac{\partial net(\vec{x})}{\partial w_i}\right]$$



- **But We Know:**

$$\frac{\partial o(\vec{x})}{\partial net(\vec{x})} = \frac{\partial \sigma(net(\vec{x}))}{\partial net(\vec{x})} = o(\vec{x})\left(1 - o(\vec{x})\right)$$

$$\frac{\partial net(\vec{x})}{\partial w_i} = \frac{\partial(\vec{w} \cdot \vec{x})}{\partial w_i} = x_i$$

- **So:** $\frac{\partial E}{\partial w_i} = -\sum_{\langle \vec{x}, t(\vec{x})\rangle \in D}\left[\left(t(\vec{x}) - o(\vec{x})\right) \cdot \left(o(\vec{x})\left(1 - o(\vec{x})\right)\right) \cdot x_i\right]$