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

<table>
<thead>
<tr>
<th>Day</th>
<th>Outlook</th>
<th>Temperature</th>
<th>Humidity</th>
<th>Wind</th>
<th>PlayTennis?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Light</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>Overcast</td>
<td>Hot</td>
<td>High</td>
<td>Light</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>Rain</td>
<td>Mild</td>
<td>High</td>
<td>Light</td>
<td>Yes</td>
</tr>
<tr>
<td>5</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Light</td>
<td>Yes</td>
</tr>
<tr>
<td>6</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>7</td>
<td>Overcast</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>8</td>
<td>Sunny</td>
<td>Mild</td>
<td>High</td>
<td>Light</td>
<td>No</td>
</tr>
<tr>
<td>9</td>
<td>Sunny</td>
<td>Cool</td>
<td>Normal</td>
<td>Light</td>
<td>Yes</td>
</tr>
<tr>
<td>10</td>
<td>Rain</td>
<td>Mild</td>
<td>Normal</td>
<td>Light</td>
<td>Yes</td>
</tr>
<tr>
<td>11</td>
<td>Sunny</td>
<td>Mild</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>12</td>
<td>Overcast</td>
<td>Mild</td>
<td>High</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>13</td>
<td>Overcast</td>
<td>Hot</td>
<td>Normal</td>
<td>Light</td>
<td>Yes</td>
</tr>
<tr>
<td>14</td>
<td>Rain</td>
<td>Mild</td>
<td>High</td>
<td>Strong</td>
<td>No</td>
</tr>
</tbody>
</table>

• ID3 = Build-DT using Gain(·)
• How Will ID3 Construct A Decision Tree?
• Selecting The Root Attribute

Prior (unconditioned) distribution: 9+, 5−

- $H(D) = -(9/14 \lg (9/14) - (5/14) \lg (5/14) bits = 0.94$ bits
- $H(D, \text{Humidity} = \text{High}) = -(3/7 \lg (3/7) - (4/7) \lg (4/7) = 0.985$ bits
- $H(D, \text{Humidity} = \text{Normal}) = -(6/7 \lg (6/7) - (1/7) \lg (1/7) = 0.592$ bits
- $\text{Gain}(D, \text{Humidity}) = 0.94 - (7/14) * 0.985 + (7/14) * 0.592 = 0.151$ bits
- Similarly, $\text{Gain}(D, \text{Wind}) = 0.94 - (8/14) * 0.811 + (6/14) * 1.0 = 0.048$ 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 $\text{Gain}(D, A) < 0$?
Selecting The Next Attribute (Root of Subtree)

- Convention: \( \log \frac{0}{a} = 0 \)
- \( \text{Gain}(D_{\text{Sunny, Humidity}}) = 0.97 - \frac{3}{5} \times 0 - \frac{2}{5} \times 0 = 0.97 \) bits
- \( \text{Gain}(D_{\text{Sunny, Wind}}) = 0.97 - \frac{2}{5} \times 1 - \frac{3}{5} \times 0.92 = 0.02 \) bits
- \( \text{Gain}(D_{\text{Sunny, Temperature}}) = 0.57 \) bits

Top-Down Induction

- For discrete-valued attributes, terminates in \( \Theta(n) \) splits
- Makes at most one pass through data set at each level (why?)

Constructing Decision Tree

### For PlayTennis using ID3 [3]

<table>
<thead>
<tr>
<th>Day</th>
<th>Outlook</th>
<th>Temperature</th>
<th>Humidity</th>
<th>Wind</th>
<th>PlayTennis?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Light</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>Overcast</td>
<td>Hot</td>
<td>High</td>
<td>Light</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Light</td>
<td>Yes</td>
</tr>
<tr>
<td>5</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>6</td>
<td>Overcast</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>7</td>
<td>Sunny</td>
<td>Cool</td>
<td>Normal</td>
<td>Light</td>
<td>Yes</td>
</tr>
<tr>
<td>8</td>
<td>Rain</td>
<td>Mild</td>
<td>Normal</td>
<td>Light</td>
<td>Yes</td>
</tr>
<tr>
<td>9</td>
<td>Rain</td>
<td>Mild</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>10</td>
<td>Overcast</td>
<td>Mild</td>
<td>High</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>11</td>
<td>Overcast</td>
<td>Cool</td>
<td>Normal</td>
<td>Light</td>
<td>Yes</td>
</tr>
<tr>
<td>12</td>
<td>Sunny</td>
<td>Mild</td>
<td>High</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>13</td>
<td>Overcast</td>
<td>Mild</td>
<td>High</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>14</td>
<td>Rain</td>
<td>Mild</td>
<td>High</td>
<td>Strong</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Constructing Decision Tree

### For PlayTennis using ID3 [4]

<table>
<thead>
<tr>
<th>Day</th>
<th>Outlook</th>
<th>Temperature</th>
<th>Humidity</th>
<th>Wind</th>
<th>PlayTennis?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Light</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>Overcast</td>
<td>Hot</td>
<td>High</td>
<td>Light</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Light</td>
<td>Yes</td>
</tr>
<tr>
<td>5</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>6</td>
<td>Overcast</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>7</td>
<td>Sunny</td>
<td>Cool</td>
<td>Normal</td>
<td>Light</td>
<td>Yes</td>
</tr>
<tr>
<td>8</td>
<td>Rain</td>
<td>Mild</td>
<td>Normal</td>
<td>Light</td>
<td>Yes</td>
</tr>
<tr>
<td>9</td>
<td>Rain</td>
<td>Mild</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>10</td>
<td>Overcast</td>
<td>Mild</td>
<td>High</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>11</td>
<td>Overcast</td>
<td>Cool</td>
<td>Normal</td>
<td>Light</td>
<td>Yes</td>
</tr>
<tr>
<td>12</td>
<td>Sunny</td>
<td>Mild</td>
<td>High</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>13</td>
<td>Overcast</td>
<td>Mild</td>
<td>High</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>14</td>
<td>Rain</td>
<td>Mild</td>
<td>High</td>
<td>Strong</td>
<td>Yes</td>
</tr>
</tbody>
</table>
**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 $Gain(\cdot)$
  - 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 as Build-DT using $Gain(\cdot)$
- 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
Human Brains
- Neuron switching time: \( \sim 0.001 \times 10^{-3} \) second
- Number of neurons: \( \sim 10^{-100} \) billion \( \left( 10^{10} - 10^{11} \right) \)
- Connections per neuron: \( \sim 10^{-100} \) thousand \( \left( 10^4 - 10^5 \right) \)
- Scene recognition time: \( \sim 0.1 \) 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

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*
- Drives 70mph on highways

**Perceptron**: Single Neuron Model
- *aka* Linear Threshold Unit (LTU) or Linear Threshold Gate (LTG)
- Net input to unit: defined as linear combination \( \text{net} = \sum w_i x_i \)
- 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
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 = 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
  - \( w_i = w_i + \Delta w_i \)
  - \( \Delta w_i = r (t - o) x_i \)
  - 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
Simple Gradient Descent Algorithm

- Applicable to concept learning, symbolic learning (with proper representation)

Algorithm Train-Perceptron \((D \equiv \{<x, t(x) \equiv c(x)\rangle\})\)

- 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: \(x_1 \oplus x_2 \oplus \ldots \oplus x_n\))
  - 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
- \(\theta\): threshold value

Linearly Separable Functions

- NB: \(D\) is LS does not necessarily imply \(c(x) = f(x)\) is LS!
- Disjunctions: \(c(x) = x_1 \lor x_2 \lor \ldots \lor x_m\)
- \(m\) of \(n\): \(c(x) = \) at least 3 of \((x_1', x_2', \ldots, x_m')\)
- Exclusive OR (XOR): \(c(x) = x_1 \oplus x_2\)
- General DNF: \(c(x) = T_1 \lor T_2 \lor \ldots \lor T_m = l_1 \land l_2 \land \ldots \land l_k\)

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?
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: Understanding Gradient Descent for Linear Units**

- **Consider simpler, unthresholded linear unit:**
  \[ o(\tilde{x}) = \text{net}(\tilde{x}) = \sum_{i} w_i x_i \]

- **Objective:** find “best fit” to \( D \)

**Approximation Algorithm**

- **Quantitative objective:** minimize error over training data set \( D \)
- **Error function:** sum squared error (SSE)
  \[ E[\tilde{w}] = \text{error}_D[\tilde{w}] = \frac{1}{2} \sum_{x \in D} (t(x) - o(\tilde{x}))^2 \]

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

Definition: Gradient

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

Modified Gradient Descent Training Rule

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

\[ \Delta w_i = -r \frac{\partial E}{\partial w_i} \]

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

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

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

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 \(w\) to (small) random values
- Until the termination condition is met, do
  - Initialize each \(\Delta w\) 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 \(w\), do
      - \(\Delta w \leftarrow \Delta w + r(t - o)x\)\)
      - \(w \leftarrow w + \Delta w\)
  - 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)
**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 \( \mathbf{w} = \text{Sum of Misclassified } \mathbf{x} \in D \)**
- Perceptron: minimize \( \mathbf{w} \)
- Delta Rule: minimize \( \text{error} \equiv \text{distance from separator (I.e., maximize } \frac{\partial E}{\partial \mathbf{w}} \text{)} \)

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

**Example A**

**Example B**

**Example C**

**Intuitive Idea: Distribute Blame for Error to Previous Layers**

**Algorithm Train-by-Backprop \( (D, r) \)**
- Each training example is a pair of the form \( \langle \mathbf{x}, t(\mathbf{x}) \rangle \), where \( \mathbf{x} \) is the vector of input values and \( t(\mathbf{x}) \) is the output value, \( r \) is the learning rate (e.g., 0.05)
- Initialize all weights \( w \) to (small) random values
- UNTIL the termination condition is met, DO

  FOR each \( \langle \mathbf{x}, t(\mathbf{x}) \rangle \) in \( D \), DO

  **Input the instance \( \mathbf{x} \) to the unit and compute the output \( o(\mathbf{x}) = \sigma(\text{net}(\mathbf{x})) \)**

  FOR each output unit \( k \), DO

  \[ \delta_k = o_k(\mathbf{x})(1 - o_k(\mathbf{x}))(t_k(\mathbf{x}) - o_k(\mathbf{x})) \]

  **FOR each hidden unit \( j \), DO**

  \[ \delta_j = h_j(\mathbf{x})(1 - h_j(\mathbf{x}))(\sum_{\text{outputs} k} o_{k}(\mathbf{x}) \delta_k) \]

  **Update each \( w = u_{i,j} (a = h) \) or \( w = v_{j,k} (a = o) \)**

  \[ w_{\text{start-layer}, \text{end-layer}} \leftarrow w_{\text{start-layer}, \text{end-layer}} + \Delta w_{\text{start-layer}, \text{end-layer}} \]

  **RETURN final \( u, v \)**
Review: Derivation of Backprop

- Recall: Gradient of Error Function
  \[ \nabla E(\theta) = \left[ \frac{\partial E}{\partial \theta_1}, \frac{\partial E}{\partial \theta_2}, \ldots, \frac{\partial E}{\partial \theta_n} \right] \]

- Gradient of Sigmoid Activation Function
  \[ \frac{\partial E}{\partial w_i} = \frac{1}{2} \sum_{(x, y) \in D} (t(x) - o(x))^2 \]
  \[ = \frac{1}{2} \sum_{(x, y) \in D} \left[ 2(t(x) - o(x)) \frac{\partial}{\partial w_i} (t(x) - o(x)) \right] \]
  \[ = \sum_{(x, y) \in D} \left[ (t(x) - o(x)) \cdot \frac{\partial o(x)}{\partial w_i} \cdot \frac{\partial o(x)}{\partial o(x)} \right] \]

- But We Know:
  \[ \frac{\partial o(x)}{\partial \text{net}(x)} = o(x)(1 - o(x)) \]
  \[ \frac{\partial \text{net}(x)}{\partial w_j} = x_j \]

- So:
  \[ \frac{\partial E}{\partial w_j} = -\sum_{(x, y) \in D} (t(x) - o(x)) \cdot o(x)(1 - o(x)) \cdot x_j \]