Artificial Neural Networks (ANNs):
More Perceptrons and Winnow

Wednesday, 08 February 2007

William H. Hsu
Department of Computing and Information Sciences, KSU

Readings:
Sections 4.1-4.4, Mitchell
Section 2.2.6, Shavlik and Dietterich (Rosenblatt)
Section 2.4.5, Shavlik and Dietterich (Minsky and Papert)

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: “Discriminative Models for IR”, Nallapati
• 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 \(\text{Delta aka LMS}\) rule
– Next lecture: as a neural network element (especially in multiple layers)

The Winnow
– Another linear threshold model
– Learning algorithm and training rule
Review:
ALVINN and Feedforward ANN Topology

- Pomerleau et al
  - Drives 70mph on highways
Review: The Perceptron

Perceptron: Single Neuron Model
- *aka* Linear Threshold Unit (LTU) or Linear Threshold Gate (LTG)
- Net input to unit: defined as linear combination
  \[ \sum_{i=0}^{n} w_i x_i \]
- 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

Review: Linear Separators

Functional Definition
- \( f(x) = 1 \) if \( w_1 x_1 + w_2 x_2 + \ldots + w_n x_n \geq \theta, 0 \) 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 \( n \): \( c(x) = \text{at least } 3 \) of \( (x_1', x_2', \ldots, x_m') \)
- Exclusive OR (XOR): \( c(x) = x_1 \oplus x_2 \)
- General DNF: \( c(x) = T_1 \lor T_2 \lor \ldots \lor T_m; T_i = l_1 \land l_2 \land \ldots \land l_k \)

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?
Review:
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:
    \[
    o(x) = \text{net}(x) = \sum_{i=0}^{n} w_i x_i
    \]
  – Objective: find “best fit” to \( D \)

• Approximation Algorithm
  – Quantitative objective: minimize error over training data set \( D \)
  – Error function: sum squared error (SSE)
    \[
    E[w] = \text{error}_o[w] = \frac{1}{2} \sum_{i=0}^{n} (t(x_i) - o(x_i))^2
    \]

• 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

\[ \nabla E[w] = \begin{bmatrix} \frac{\partial E}{\partial w_1} \\ \frac{\partial E}{\partial w_2} \\ \vdots \\ \frac{\partial E}{\partial w_n} \end{bmatrix} \]

- Modified Gradient Descent Training Rule

\[ \Delta w = -\nabla E[w] \]

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

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

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

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

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)
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 $\frac{\partial E}{\partial w}$)

Incremental (Stochastic) Gradient Descent

- Batch Mode Gradient Descent
  - UNTIL the termination condition is met, DO
    1. Compute the gradient $\nabla E_D[w]$
    2. $w \leftarrow w - r \nabla E_D[w]$
  - RETURN final $w$
- Incremental (Online) Mode Gradient Descent
  - UNTIL the termination condition is met, DO
    FOR each $<x, t(x)>$ in $D$, DO
    1. Compute the gradient $\nabla E_x[w]$
    2. $w \leftarrow w - r \nabla E_x[w]$
  - RETURN final $w$
- Emulating Batch Mode
  \[ E_D[w] = \frac{1}{2} \sum_{x \in D} (t(x) - o(x))^2 \]
  \[ E_x[w] = \frac{1}{2} (t(x) - o(x))^2 \]
  - Incremental gradient descent can approximate batch gradient descent arbitrarily closely if $r$ made small enough
**Multi-Layer Networks of Nonlinear Units**

- **Nonlinear Units**
  - Recall: activation function $\text{sgn}(w \cdot x)$
  - Nonlinear activation function: generalization of $\text{sgn}$

- **Multi-Layer Networks**
  - A specific type: Multi-Layer Perceptrons (MLPs)
  - Definition: a multi-layer feedforward network is composed of an input layer, one or more hidden layers, and an output layer
  - “Layers”: counted in weight layers (e.g., 1 hidden layer = 2-layer network)
  - Only hidden and output layers contain perceptrons (threshold or nonlinear units)

- **MLPs in Theory**
  - Network (of 2 or more layers) can represent any function (arbitrarily small error)
  - Training even 3-unit multi-layer ANNs is \(\text{NP}\)-hard (Blum and Rivest, 1992)

- **MLPs in Practice**
  - Finding or designing effective networks for arbitrary functions is difficult
  - Training is very computation-intensive even when structure is “known”

**Nonlinear Activation Functions**

- **Sigmoid Activation Function**
  - Linear threshold gate activation function: $\text{sgn}(w \cdot x)$
  - Nonlinear activation (aka transfer, squashing) function: generalization of $\text{sgn}$
  - $\sigma$ is the sigmoid function
  - Can derive gradient rules to train
    - One sigmoid unit
    - Multi-layer, feedforward networks of sigmoid units (using backpropagation)

- **Hyperbolic Tangent Activation Function**
  - $\text{tanh}(\text{net}) = \frac{e^{\text{net}} - e^{-\text{net}}}{e^{\text{net}} + e^{-\text{net}}}$
Error Gradient for a Sigmoid Unit

- **Recall: Gradient of Error Function**
  \[ \nabla E(\theta) = \left[ \frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_n} \right] \]

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

- **But We Know:**
  \[
  \frac{\partial o(x)}{\partial \sigma(x)} = o(x) \sigma(x) \\
  \frac{\partial \sigma(x)}{\partial w_i} = x_i \\
  \]

- **So:**
  \[
  \frac{\partial E}{\partial w_i} = -\sum_{(x, \sigma(x) \in \Theta)} \left[ \frac{\partial (t(x) - o(x))}{\partial \sigma(x)} o(x)(1-o(x)) x_i \right] \\
  \]

Learning Disjunctions

- **Hidden Disjunction to Be Learned**
  - \( c(x) = x_1' \lor x_2' \lor \ldots \lor x_{m'} \) (e.g., \( x_2 \lor x_4 \lor x_5 \ldots \lor x_{100} \))
  - 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 \( k << n \) 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
Winnow Algorithm

- **Algorithm Train-Winnow \((D)\)**
  - Initialize: \( \theta = n, w_i = 1 \)
  - UNTIL the termination condition is met, DO
    - FOR each \(<x, t(x)>\) 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?

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**
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