Lecture 7

Multi-Layer Perceptrons and Backpropagation of Error

Thursday, September 16, 1999

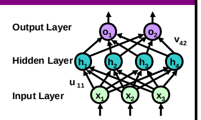
William H. Hsu
Department of Computing and Information Sciences, KSU
http://www.cis.ksu.edu/~bhsu

Readings:
Sections 4.5-4.9, Mitchell
Chapter 4 ("The Multi-Layer Perceptron"), Bishop
Section 2.2.5, Shavlik and Dietterich (Rumelhart *et al*)

Lecture Outline

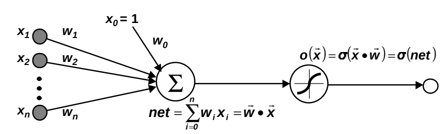
- Read Sections 4.5-4.9, Mitchell; Chapter 4, Bishop; Rumelhart *et al*
- Multi-Layer Networks
  - Nonlinear transfer functions
  - Multi-layer networks of nonlinear units (sigmoid, hyperbolic tangent)
- Backpropagation of Error
  - The backpropagation algorithm
    - Relation to error gradient function for nonlinear units
    - Derivation of training rule for feedfoward multi-layer networks
  - Training issues
    - Local optima
    - Overfitting in ANNs
- Hidden-Layer Representations
- Examples: Face Recognition and Text-to-Speech
- Advanced Topics (Brief Survey)
- Next Week: Chapter 5 and Sections 6.1-6.5, Mitchell; Quinlan paper

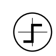
Multi-Layer Networks of Nonlinear Units



- Nonlinear Units
  - Recall: activation function *sgn* (***w*** • ***x***)
  - Nonlinear activation function: generalization of *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 $\mathcal{NP}$-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: *sgn* (***w*** • ***x***)
  - Nonlinear activation (*aka* transfer, squashing) function: generalization of *sgn*
  - σ is the sigmoid function $\sigma(net) = \frac{1}{1+e^{-net}}$
  - Can derive gradient rules to train
    - One sigmoid unit
    - Multi-layer, feedforward networks of sigmoid units (using backpropagation)
- Hyperbolic Tangent Activation Function $\sigma(net) = \frac{\sinh(net)}{\cosh(net)} = \frac{e^{net} - e^{-net}}{e^{net} + e^{-net}}$

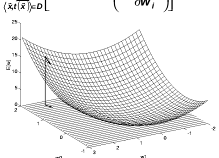
Error Gradient for a Sigmoid Unit

- 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

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

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

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



- But We Know:

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

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

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

Backpropagation Algorithm

- 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*) = σ(*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

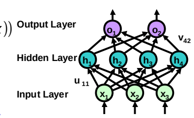
    $$\delta_j = h_j(x)(1 - h_j(x)) \sum_{k \in outputs} v_{j,k}\delta_j$$

    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*



Backpropagation and Local Optima

- Gradient Descent in Backprop
  - Performed over entire *network* weight vector
  - Easily generalized to arbitrary directed graphs
  - Property: Backprop on feedforward ANNs will find a *local* (not necessarily global) error minimum
- Backprop in Practice
  - Local optimization often works well (can *run multiple times*)
  - Often include weight momentum α

$$\Delta w_{start\text{-}layer,\, end\text{-}layer}(n) = r\,\delta_{end\text{-}layer}\; a_{end\text{-}layer} + \alpha \Delta w_{start\text{-}layer,\, end\text{-}layer}(n-1)$$

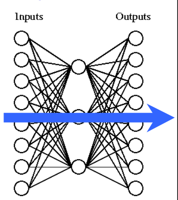
  - Minimizes error over training examples - generalization to subsequent instances?
  - Training often *very* slow: thousands of iterations over *D* (epochs)
  - Inference (applying network after training) typically very fast
    - Classification
    - Control

Feedforward ANNs: Representational Power and Bias

- Representational (i.e., Expressive) Power
  - Backprop presented for feedforward ANNs with single hidden layer (2-layer)
  - 2-layer feedforward ANN
    - Any Boolean function (simulate a 2-layer AND-OR network)
    - Any bounded continuous function (*approximate with arbitrarily small error*) [Cybenko, 1989; Hornik *et al*, 1989]
  - Sigmoid functions: set of basis functions; used to compose arbitrary functions
  - 3-layer feedforward ANN: any function (*approximate with arbitrarily small error*) [Cybenko, 1988]
  - Functions that ANNs are good at acquiring: Network Efficiently Representable Functions (NERFs) - how to characterize? [Russell and Norvig, 1995]
- Inductive Bias of ANNs
  - *n*-dimensional Euclidean space (weight space)
  - Continuous (error function smooth with respect to weight parameters)
  - Preference bias: "smooth interpolation" among positive examples
  - Not well understood yet (known to be computationally hard)

Learning Hidden Layer Representations

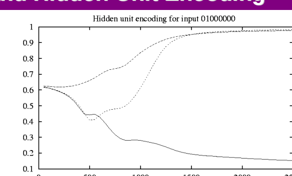
- Hidden Units and Feature Extraction
  - Training procedure: hidden unit representations that minimize error E
  - Sometimes backprop will define new hidden features that are not explicit in the input representation *x*, but which capture properties of the input instances that are most relevant to learning the target function *t*(*x*)
  - Hidden units express *newly constructed features*
  - *Change of representation* to linearly separable *D'*
- A Target Function (Sparse *aka* 1-of-C, Coding)



Input									Hidden Values				Output							
1	0	0	0	0	0	0	0	→	0.89	0.04	0.08	→	1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	→	0.01	0.11	0.88	→	0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0	→	0.01	0.97	0.27	→	0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0	→	0.99	0.97	0.71	→	0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0	→	0.03	0.05	0.02	→	0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0	→	0.22	0.99	0.99	→	0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0	→	0.80	0.01	0.98	→	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1	→	0.60	0.94	0.01	→	0	0	0	0	0	0	0	1

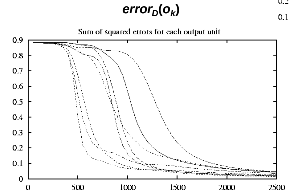
  - Can this be learned? (Why or why not?)

Training: Evolution of Error and Hidden Unit Encoding

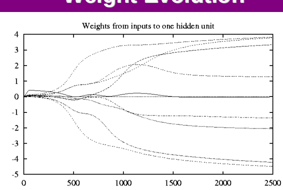


$h_j(01000000)$, $1 \le j \le 3$

$error_D(o_k)$



Training: Weight Evolution



u_{i1}, $1 \le i \le 8$

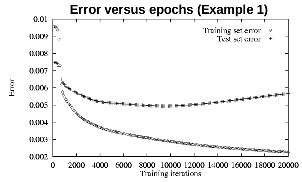
- Input-to-Hidden Unit Weights and Feature Extraction
  - Changes in first weight layer values correspond to changes in hidden layer encoding and consequent output squared errors
  - w_0 (bias weight, analogue of threshold in LTU) converges to a value near 0
  - Several changes in first 1000 epochs (different encodings)

Convergence of Backpropagation

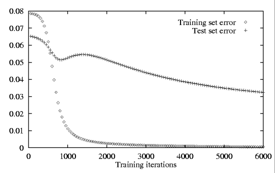
- No Guarantee of Convergence to Global Optimum Solution
  - Compare: perceptron convergence (to best $h \in H$, provided $h \in H$; i.e., LS)
  - Gradient descent to some local error minimum (perhaps not global minimum…)
  - Possible improvements on backprop (BP)
    - Momentum term (BP variant with slightly different weight update rule)
    - Stochastic gradient descent (BP algorithm variant)
    - Train multiple nets with different initial weights; find a good mixture
  - Improvements on feedforward networks
    - Bayesian learning for ANNs (e.g., simulated annealing) - later
    - Other global optimization methods that integrate over multiple networks
- Nature of Convergence
  - Initialize weights near zero
  - Therefore, initial network near-linear
  - Increasingly non-linear functions possible as training progresses

Overtraining in ANNs

- Recall: Definition of Overfitting
  - *h'* worse than *h* on D_{train}, better on D_{test}
- Overtraining: A Type of Overfitting
  - Due to excessive iterations
  - Avoidance: stopping criterion (cross-validation: holdout, *k*-fold)
  - Avoidance: weight decay



Error versus epochs (Example 1)

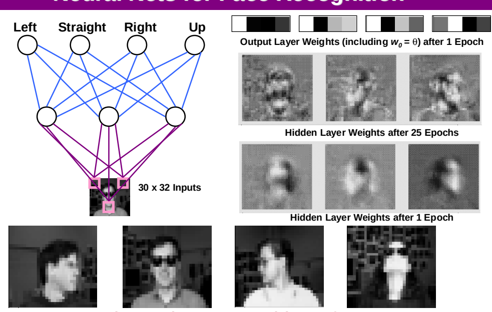


Error versus epochs (Example 2)

Overfitting in ANNs

- Other Causes of Overfitting Possible
  - Number of hidden units sometimes set in advance
  - Too few hidden units ("underfitting")
    - ANNs with no growth
    - Analogy: underdetermined linear system of equations (more unknowns than equations)
  - Too many hidden units
    - ANNs with no pruning
    - Analogy: fitting a quadratic polynomial with an approximator of degree >> 2
- Solution Approaches
  - Prevention: attribute subset selection (using pre-filter or wrapper)
  - Avoidance
    - Hold out cross-validation (CV) set or split *k* ways (when to stop?)
    - Weight decay: decrease each weight by some factor on each epoch
  - Detection/recovery: random restarts, addition and deletion of weights, units

Example: Neural Nets for Face Recognition



- 90% Accurate Learning Head Pose, Recognizing 1-of-20 Faces
- http://www.cs.cmu.edu/~tom/faces.html

Example: *NetTalk*

- Sejnowski and Rosenberg, 1987
- Early Large-Scale Application of Backprop
  - Learning to convert text to speech
    - Acquired model: *a mapping from letters to phonemes and stress marks*
    - Output passed to a speech synthesizer
  - Good performance after training on a vocabulary of ~1000 words
- Very Sophisticated Input-Output Encoding
  - Input: 7-letter window; determines the phoneme for the center letter and context on each side; distributed (i.e., sparse) representation: 200 bits
  - Output: units for articulatory modifiers (e.g., "voiced"), stress, closest phoneme; distributed representation
  - 40 hidden units; 10000 weights total
- Experimental Results
  - Vocabulary: trained on 1024 of 1463 (informal) and 1000 of 20000 (dictionary)
  - 78% on informal, ~60% on dictionary
- http://www.boltz.cs.cmu.edu/benchmarks/nettalk.html

Alternative Error Functions

- Penalize Large Weights (with Penalty Factor w_p)

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{\langle \vec{x}, t(\vec{x}) \rangle \in D} \sum_{k \in outputs} \left[\left(t_k(\vec{x}) - o_k(\vec{x}) \right)^2 + w_p \sum_{start-layer, end-layer} w^2_{start-layer, end-layer} \right]$$

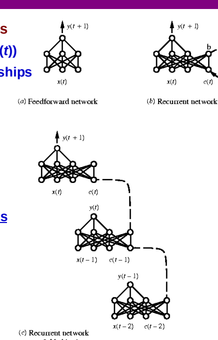
- Train on Both Target Slopes and Values

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{\langle \vec{x}, t(\vec{x}) \rangle \in D} \sum_{k \in outputs} \left[\left(t_k(\vec{x}) - o_k(\vec{x}) \right)^2 + w_s \sum_{i \in inputs} \left(\frac{\partial t_k(\vec{x})}{\partial x_i} - \frac{\partial o_k(\vec{x})}{\partial x_i} \right)^2 \right]$$

- Tie Together Weights
  - e.g., in phoneme recognition network
  - See: *Connectionist Speech Recognition* [Bourlard and Morgan, 1994]

Recurrent Networks

- Representing Time Series with ANNs
  - Feedforward ANN: *y*(*t* + 1) = *net* (*x*(*t*))
  - Need to capture temporal relationships
- Solution Approaches
  - Directed cycles
  - Feedback
    - Output-to-input [Jordan]
    - Hidden-to-input [Elman]
    - Input-to-input
  - Captures time-lagged relationships
    - Among $x(t' \le t)$ and $y(t + 1)$
    - Among $y(t' \le t)$ and $y(t + 1)$
  - Learning with recurrent ANNs
    - Elman, 1990; Jordan, 1987
    - Principe and deVries, 1992
    - Mozer, 1994; Hsu and Ray, 1998



New Neuronal Models

- Neurons with State
  - Neuroids [Valiant, 1994]
  - Each basic unit may have a state
  - Each may use a different update rule (or compute differently based on state)
  - Adaptive model of network
    - Random graph structure
    - Basic elements receive meaning as part of learning process
- Pulse Coding
  - Spiking neurons [Maass and Schmitt, 1997]
  - Output represents more than activation level
  - Phase shift between firing sequences counts and adds expressivity
- New Update Rules
  - Non-additive update [Stein and Meredith, 1993; Seguin, 1998]
  - Spiking neuron model
- Other Temporal Codings: (Firing) Rate Coding

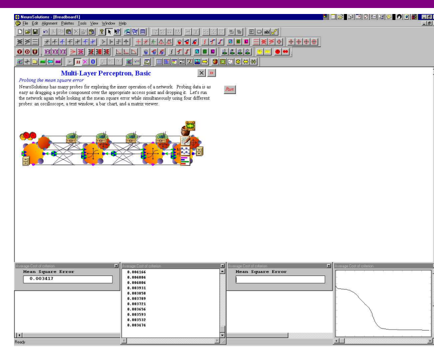
Some Current Issues and Open Problems in ANN Research

- Hybrid Approaches
  - Incorporating knowledge and analytical learning into ANNs
    - Knowledge-based neural networks [Flann and Dietterich, 1989]
    - Explanation-based neural networks [Towell *et al*, 1990; Thrun, 1996]
  - Combining uncertain reasoning and ANN learning and inference
    - Probabilistic ANNs
    - Bayesian networks [Pearl, 1988; Heckerman, 1996; Hinton *et al*, 1997] - later
- Global Optimization with ANNs
  - Markov chain Monte Carlo (MCMC) [Neal, 1996] - e.g., simulated annealing
  - Relationship to genetic algorithms - later
- Understanding ANN Output
  - Knowledge extraction from ANNs
    - Rule extraction
    - Other decision surfaces
  - Decision support and KDD applications [Fayyad *et al*, 1996]
- Many, Many More Issues (Robust Reasoning, Representations, etc.)

Some ANN Applications

- Diagnosis
  - Closest to pure concept learning and classification
  - Some ANNs can be post-processed to produce probabilistic diagnoses
- Prediction and Monitoring
  - *aka* prognosis (sometimes forecasting)
  - Predict a continuation of (typically numerical) data
- Decision Support Systems
  - *aka* recommender systems
  - Provide assistance to human "subject matter" experts in making decisions
    - Design (manufacturing, engineering)
    - Therapy (medicine)
    - Crisis management (medical, economic, military, computer security)
- Control Automation
  - Mobile robots
  - Autonomic sensors and actuators
- Many, Many More (ANNs for Automated Reasoning, etc.)

NeuroSolutions Demo



Terminology

- Multi-Layer ANNs
  - Focused on one species: (feedforward) multi-layer perceptrons (MLPs)
  - Input layer: an implicit layer containing x_i
  - Hidden layer: a layer containing input-to-hidden unit weights and producing h_j
  - Output layer: a layer containing hidden-to-output unit weights and producing o_k
  - *n*-layer ANN: an ANN containing $n - 1$ hidden layers
  - Epoch: one training iteration
  - Basis function: set of functions that span H
- Overfitting
  - Overfitting: h does better than h' on training data and worse on test data
  - Overtraining: overfitting due to training for too many epochs
  - Prevention, avoidance, and recovery techniques
    - Prevention: attribute subset selection
    - Avoidance: stopping (termination) criteria (CV-based), weight decay
- Recurrent ANNs: Temporal ANNs with Directed Cycles

Summary Points

- Multi-Layer ANNs
  - Focused on feedforward MLPs
  - Backpropagation of error: distributes penalty (loss) function throughout network
  - Gradient learning: takes derivative of error surface with respect to weights
    - Error is based on difference between desired output (t) and actual output (o)
    - Actual output (o) is based on activation function
    - Must take partial derivative of $\sigma \Rightarrow$ choose one that is easy to differentiate
    - Two σ definitions: sigmoid (*aka* logistic) and hyperbolic tangent (*tanh*)
- Overfitting in ANNs
  - Prevention: attribute subset selection
  - Avoidance: cross-validation, weight decay
- ANN Applications: Face Recognition, Text-to-Speech
- Open Problems
- Recurrent ANNs: Can Express Temporal Depth (Non-Markovity)
- Next: Statistical Foundations and Evaluation, Bayesian Learning Intro