

Lecture 14 of 42

Support Vector Machines

Thursday, 15 February 2007

William H. Hsu

Department of Computing and Information Sciences, KSU

<http://www.cis.ksu.edu/~bhsu>

<http://www.kddresearch.org>

Readings:

Lecture Notes on SVM by Carlos Guestrin, CMU: <http://snipurl.com/g4i7>

“Bagging, Boosting, and C4.5”, Quinlan

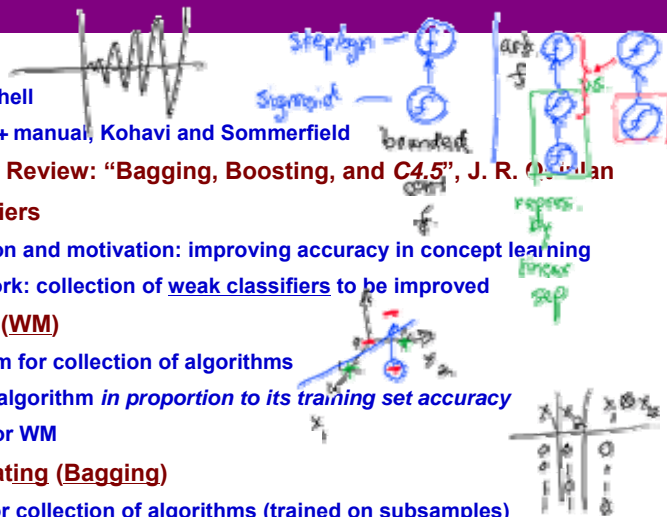


CIS 732: Machine Learning and Pattern Recognition

Department of Computing and Information Sciences

Lecture Outline

- **Readings**
 - Section 7.5, Mitchell
 - Section 5, *MLC++ manual*, Kohavi and Sommerfield
- **This Week's Paper Review: “Bagging, Boosting, and C4.5”, J. R. Quinlan**
- **Combining Classifiers**
 - Problem definition and motivation: improving accuracy in concept learning
 - General framework: collection of weak classifiers to be improved
- **Weighted Majority (WM)**
 - Weighting system for collection of algorithms
 - “Trusting” each algorithm *in proportion to its training set accuracy*
 - Mistake bound for WM
- **Bootstrap Aggregating (Bagging)**
 - Voting system for collection of algorithms (trained on subsamples)
 - When to expect bagging to work (unstable learners)
- **Next Lecture: Boosting the Margin, Hierarchical Mixtures of Experts**

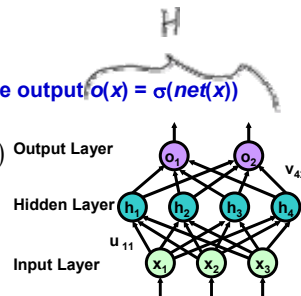


CIS 732: Machine Learning and Pattern Recognition

Department of Computing and Information Sciences

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

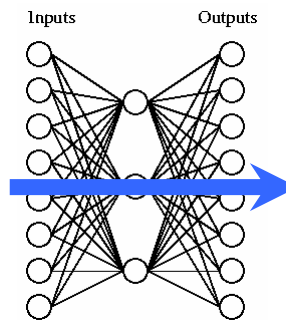
- **Representational (i.e., Expressive) Power**
 - Backprop presented for feedforward ANNs with single hidden layer (2-layer)
 - 2-layer feedforward ANN
 - Any **Boolean function** (simulate a 2-layer AND-OR network)
 - Any **bounded continuous function** (*approximate with arbitrarily small error*) [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)



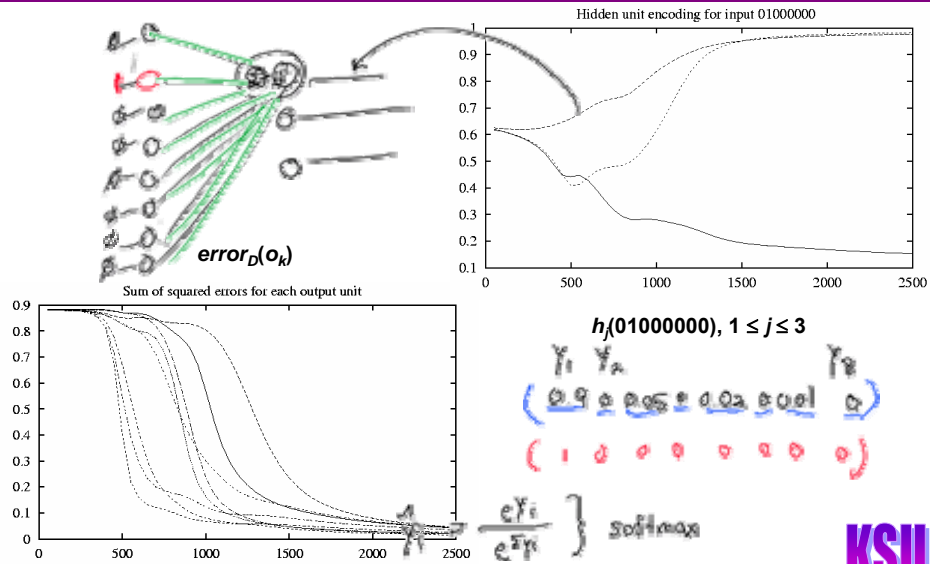
Learning Hidden Layer Representations

- **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 \vec{x} , but which capture properties of the input instances that are most relevant to learning the target function $f(x)$
 - 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



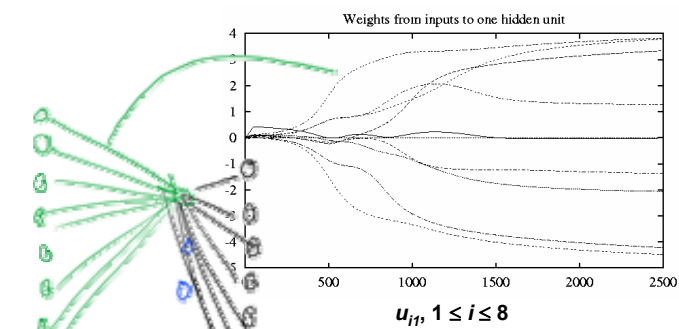
Training: Evolution of Error and Hidden Unit Encoding



CIS 732: Machine Learning and Pattern Recognition

Kansas State University
Department of Computing and Information Sciences

Training: Weight Evolution



- **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)

CIS 732: Machine Learning and Pattern Recognition

Kansas State University
Department of Computing and Information Sciences

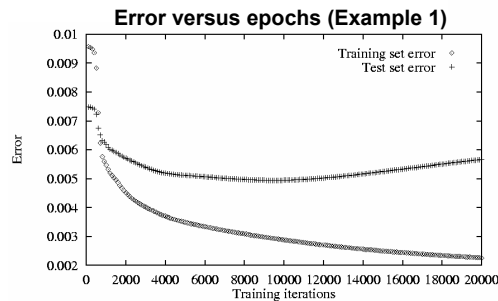
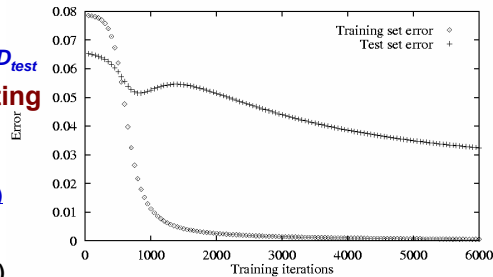
Convergence of Backpropagation

- **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

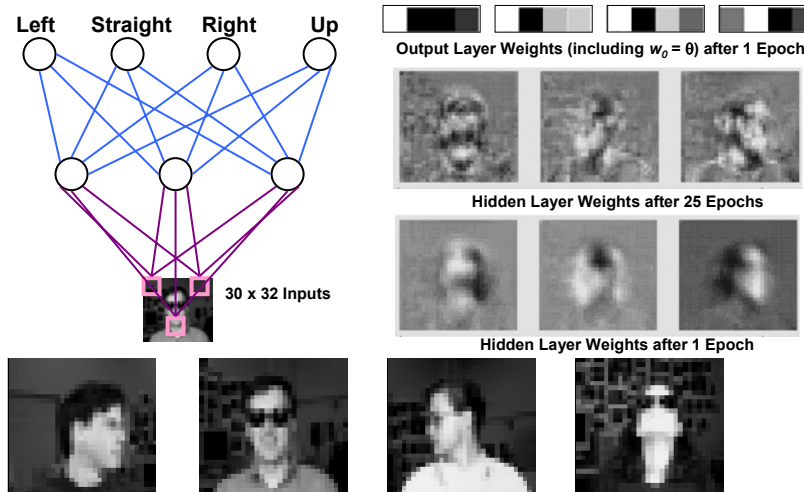


Overfitting in ANNs

- **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 $\gg 2$
- **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)**

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

- **Train on Both Target Slopes and Values**

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

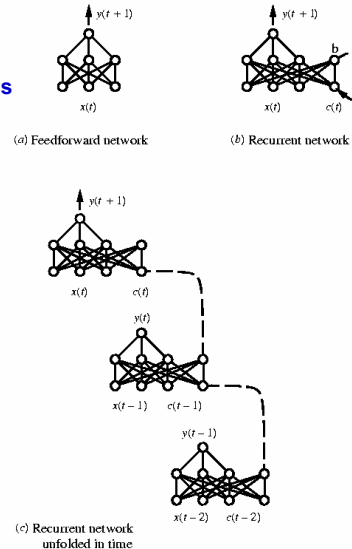
- **Tie Together Weights**

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



Recurrent Networks

- **Representing Time Series with ANNs**
 - Feedforward ANN: $y(t+1) = \text{net}(x(t))$
 - 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 $x(t' \leq t)$ and $y(t+1)$
 - Among $y(t' \leq t)$ 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**



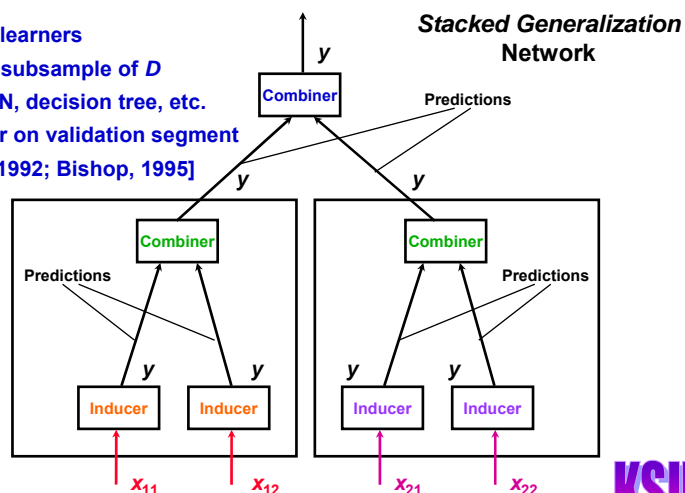
Some Current Issues and Open Problems in ANN Research

- **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.)**



Stacked Generalization: Idea

- **Stacked Generalization aka Stacking**
- **Intuitive Idea**
 - Train multiple learners
 - Each uses subsample of D
 - May be ANN, decision tree, etc.
 - Train combiner on validation segment
 - See [Wolpert, 1992; Bishop, 1995]



Stacked Generalization: Procedure

- **Algorithm *Combiner-Stacked-Gen* ($D, L, k, n, m', Levels$)**
 - Divide D into k segments, $S[1], S[2], \dots, S[k]$ *// Assert $D.size = m$*
 - FOR $i \leftarrow 1$ TO k DO
 - $Validation-Set \leftarrow S[i]$ *// m/k examples*
 - FOR $j \leftarrow 1$ TO n DO
 - $Train-Set[j] \leftarrow Sample-With-Replacement(D \sim S[i], m')$ *// $m - m/k$ examples*
 - IF $Levels > 1$ THEN
 - $P[j] \leftarrow Combiner-Stacked-Gen(Train-Set[j], L, k, n, m', Levels - 1)$
 - ELSE *// Base case: 1 level*
 - $P[j] \leftarrow L[j].Train-Inducer(Train-Set[j])$
 - $Combiner \leftarrow L[0].Train-Inducer(Validation-Set.targets, Apply-Each(P, Validation-Set.inputs))$
 - $Predictor \leftarrow Make-Predictor(Combiner, P)$
 - RETURN $Predictor$
- **Function *Sample-With-Replacement*: Same as for Bagging**



Stacked Generalization: Properties

- **Similar to Cross-Validation**
 - k -fold: rotate validation set
 - Combiner mechanism based on validation set as well as training set
 - Compare: committee-based combiners [Perrone and Cooper, 1993; Bishop, 1995] aka consensus under uncertainty / fuzziness, consensus models
 - Common application with cross-validation: treat as overfitting control method
 - *Usually improves generalization performance*
- **Can Apply Recursively (Hierarchical Combiner)**
 - Adapt to inducers on different subsets of input
 - Can apply $s(Train-Set[j])$ to transform each input data set
 - e.g., attribute partitioning [Hsu, 1998; Hsu, Ray, and Wilkins, 2000]
 - Compare: Hierarchical Mixtures of Experts (HME) [Jordan *et al*, 1991]
 - Many differences (validation-based vs. mixture estimation; online vs. offline)
 - Some similarities (hierarchical combiner)



Other Combiners

- **So Far: Single-Pass Combiners**
 - First, train each inducer
 - Then, train combiner on their output and evaluate based on criterion
 - **Weighted majority**: training set accuracy
 - **Bagging**: training set accuracy
 - **Stacking**: validation set accuracy
 - Finally, apply combiner function to get new prediction algorithm (classifier)
 - **Weighted majority**: weight coefficients (penalized based on mistakes)
 - **Bagging**: voting committee of classifiers
 - **Stacking**: validated hierarchy of classifiers with trained combiner inducer
- **Next: Multi-Pass Combiners**
 - Train inducers and combiner function(s) *concurrently*
 - Learn how to *divide* and *balance* learning problem across multiple inducers
 - Framework: mixture estimation



Terminology

- **Combining Classifiers**
 - Weak classifiers: not guaranteed to do better than random guessing
 - Combiners: functions $f: \text{prediction vector} \times \text{instance} \rightarrow \text{prediction}$
- **Single-Pass Combiners**
 - Weighted Majority (WM)
 - Weights prediction of each inducer according to its training-set accuracy
 - Mistake bound: maximum number of mistakes before converging to correct h
 - Incrementality: ability to update parameters without complete retraining
 - Bootstrap Aggregating (aka Bagging)
 - Takes vote among multiple inducers trained on different samples of D
 - Subsampling: drawing one sample from another ($D \sim D$)
 - Unstable inducer: small change to D causes large change in h
 - Stacked Generalization (aka Stacking)
 - Hierarchical combiner: can apply recursively to re-stack
 - Trains combiner inducer using validation set



Summary Points

- **Combining Classifiers**
 - Problem definition and motivation: improving accuracy in concept learning
 - General framework: collection of weak classifiers to be improved (data fusion)
- **Weighted Majority (WM)**
 - Weighting system for collection of algorithms
 - Weights each algorithm *in proportion to its training set accuracy*
 - Use this weight in performance element (and on test set predictions)
 - Mistake bound for WM
- **Bootstrap Aggregating (Bagging)**
 - Voting system for collection of algorithms
 - Training set for each member: sampled with replacement
 - Works for unstable inducers
- **Stacked Generalization (aka Stacking)**
 - Hierarchical system for combining inducers (ANNs or other inducers)
 - Training sets for “leaves”: sampled with replacement; combiner: validation set
- **Next Lecture: Boosting the Margin, Hierarchical Mixtures of Experts**

