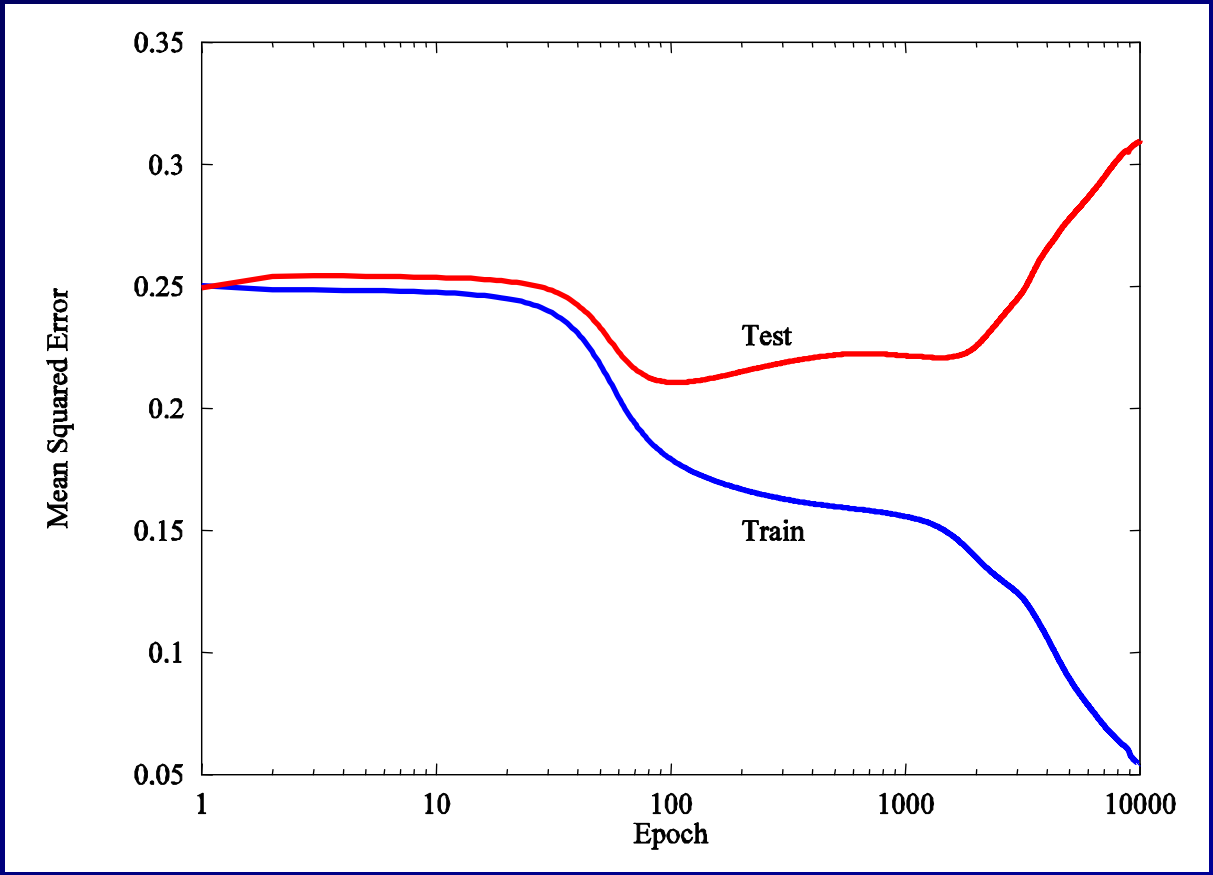
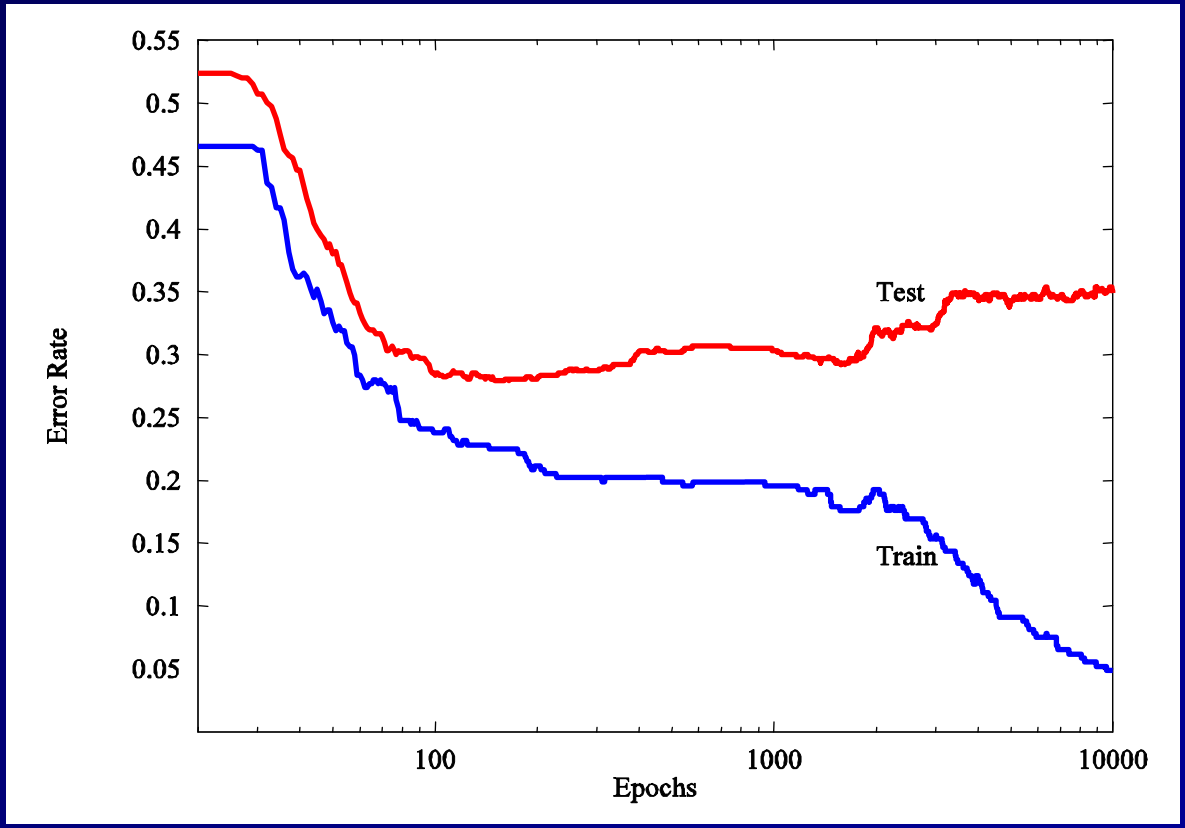
The Problem of Overfitting



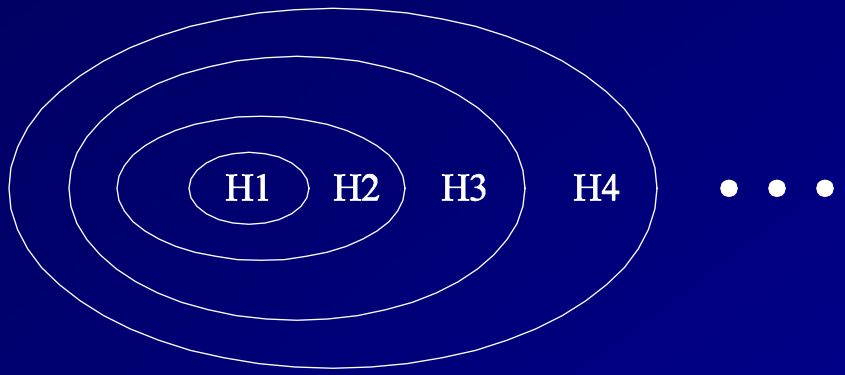
BR data: neural network with 20% classification noise, 307 training examples

Overfitting on BR (2)



- Overfitting: $h \in H$ overfits training set S if there exists $h' \in H$ that has higher training set error but lower test error on new data points. (More specifically, if learning algorithm A explicitly considers and rejects h' in favor of h, we say that A has overfit the data.)

Overfitting



$$H_1 \subset H_2 \subset H_3 \subset \cdots$$

- If we use an hypothesis space H_i that is too large, eventually we can trivially fit the training data. In other words, the VC dimension will eventually be equal to the size of our training sample *m*.
- This is sometimes called “model selection”, because we can think of each H_i as an alternative “model” of the data

Approaches to Preventing Overfitting

- Penalty methods
  - MAP provides a penalty based on P(H)
  - Structural Risk Minimization
  - Generalized Cross-validation
  - Akaike's Information Criterion (AIC)
- Holdout and Cross-validation methods
  - Experimentally determine when overfitting occurs
- Ensembles
  - Full Bayesian methods vote many hypotheses
    $\sum_h P(y|\mathbf{x},h)\, P(h|S)$
  - Many practical ways of generating ensembles

Penalty methods

- Let ε_{train} be our training set error and ε_{test} be our test error. Our real goal is to find the *h* that minimizes ε_{test}. The problem is that we can't directly evaluate ε_{test}. We can measure ε_{train}, but it is optimistic
- Penalty methods attempt to find some penalty such that

  $\varepsilon_{test} = \varepsilon_{train} + \text{penalty}$

- The penalty term is also called a regularizer or regularization term.
- During training, we set our objective function J to be

  $J(\mathbf{w}) = \varepsilon_{train}(\mathbf{w}) + \text{penalty}(\mathbf{w})$

  and find the **w** to minimize this function

MAP penalties

$$h_{map} = \text{argmax}_h \, P(S|h) \, P(h)$$

- As *h* becomes more complex, we can assign it a lower prior probability. A typical approach is to assign equal probability to each of the nested hypothesis spaces so that

  $P(h \in H_1) = P(h \in H_2) = \cdots = \alpha$

- Because H_2 contains more hypotheses than H_1, each individual $h \in H_2$ will have lower prior probability:

  $P(h) = \sum_i P(h \in H_i) = \sum_i \alpha/|H_i|$ for each i where $h \in H_i$

- If there are infinitely many H_i, this will not work, because the probabilities must sum to 1. In this case, a common approach is

  $P(h) = \sum_i 2^{-i}/|H_i|$ for each i where $h \in H_i$

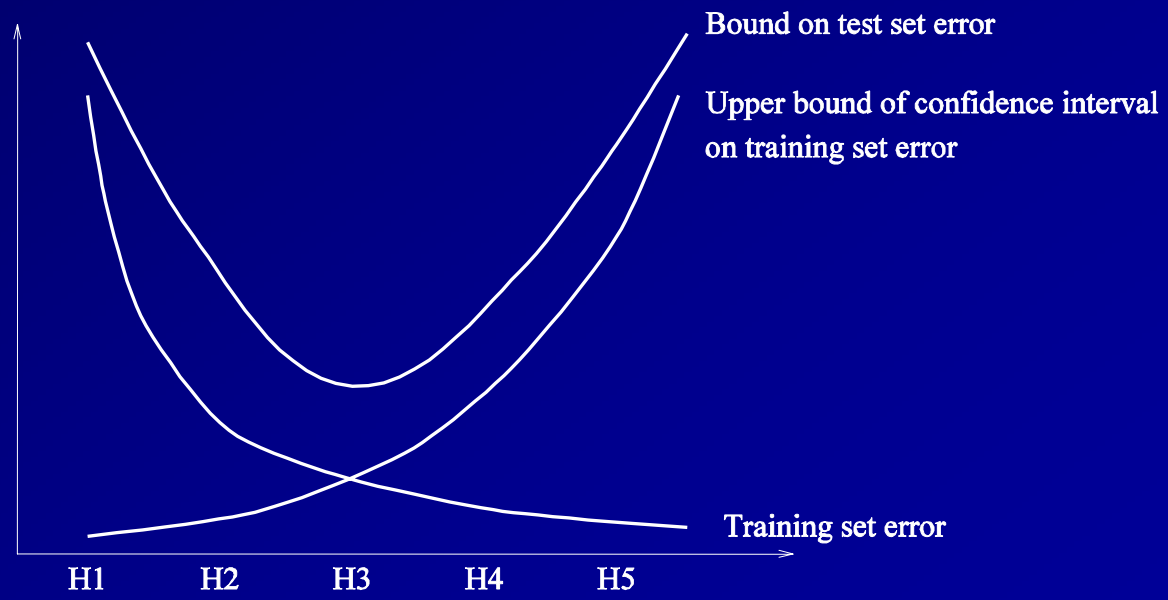
  This is not usually a big enough penalty to prevent overfitting, however

Structural Risk Minimization

- Define regularization penalty using PAC theory

$$\epsilon <= 2\epsilon_T^k + \frac{4}{m}\left[d_k \log\frac{2em}{d_k} + \log\frac{4}{\delta}\right]$$

$$\epsilon <= \frac{C}{m}\left[\frac{R^2 + \|\xi\|^2}{\gamma^2}\log^2 m + \log\frac{1}{\delta}\right]$$



Other Penalty Methods

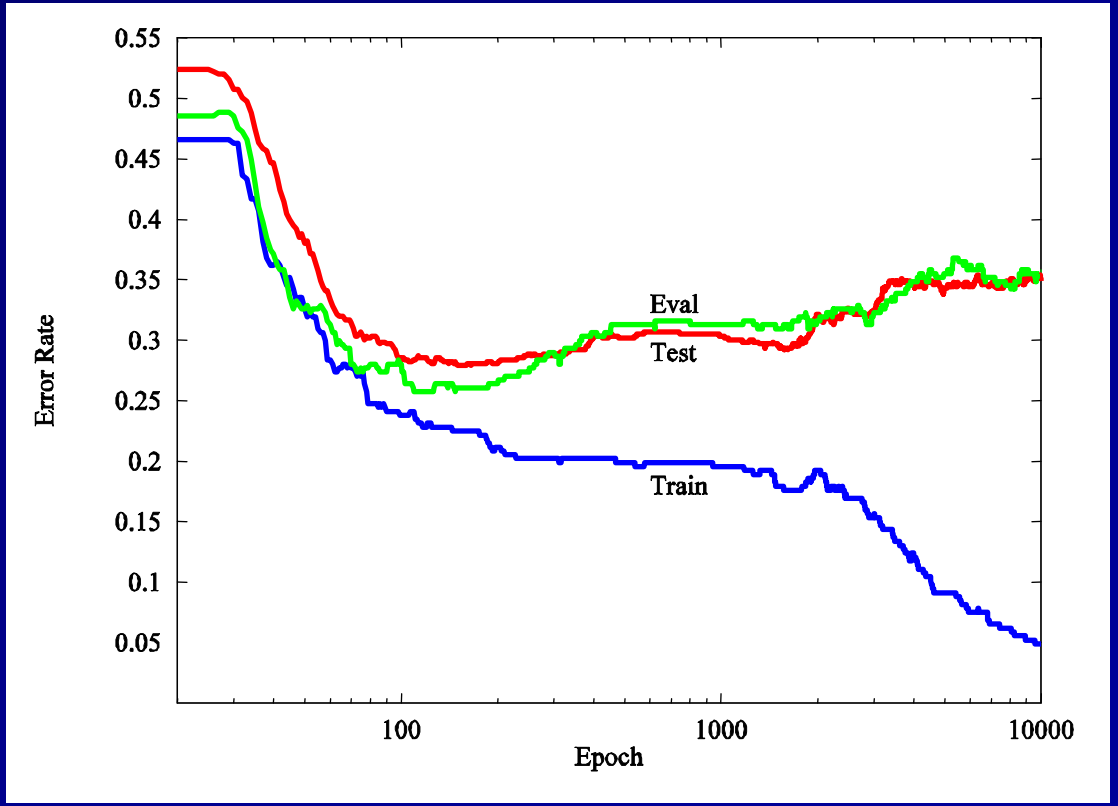
- Generalized Cross Validation
- Akaike's Information Criterion
- Mallow's P
- …

Simple Holdout Method

- Subdivide S into S_{train} and S_{eval}
- For each H_i, find $h_i \in H_i$ that best fits S_{train}
- Measure the error rate of each h_i on S_{eval}
- Choose h_i with the best error rate

Example: let H_i be the set of neural network weights after *i* epochs of training on S_{train}

Our goal is to choose *i*

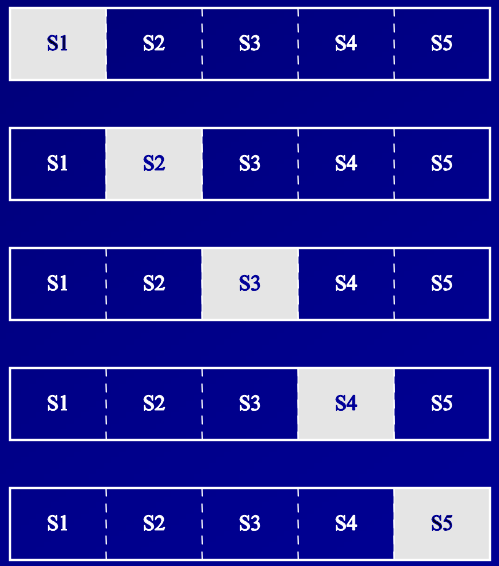


Simple Holdout Assessment

- Advantages
  - Guaranteed to perform within a constant factor of any penalty method (Kearns, et al., 1995)
  - Does not rely on theoretical approximations
- Disadvantages
  - S_{train} is smaller than S, so *h* is likely to be less accurate
  - If S_{eval} is too small, the error rate estimates will be very noisy
- Simple Holdout is widely applied to make other decisions such as learning rates, number of hidden units, SVM kernel parameters, relative size of penalty, which features to include, feature encoding methods, etc.

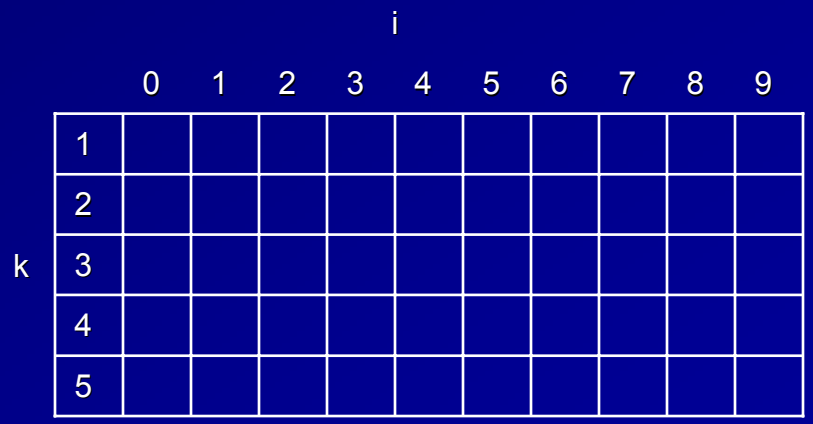
k-fold Cross-Validation to determine H_i

- Randomly divide S into k equal-sized subsets
- Run learning algorithm k times, each time use one subset for S_{eval} and the rest for S_{train}
- Average the results



K-fold Cross-Validation to determine H_i

- Partition S into K disjoint subsets $S_1, S_2, \ldots, S_k$
- Repeat simple holdout assessment K times
  - In the *k*-th assessment, $S_{train} = S - S_k$ and $S_{eval} = S_k$
  - Let h_i^k be the best hypothesis from H_i from iteration k.
  - Let ε_i be the average S_{eval} of h_i^k over the K iterations
  - Let $i^* = \text{argmin}_i \varepsilon_i$
- Train on S using H_{i^*} and output the resulting hypothesis



Ensembles

- Bayesian Model Averaging. Sample hypotheses h_i according to their posterior probability P(h|S). Vote them. A method called Markov chain Monte Carlo (MCMC) can do this (but it is quite expensive)
- Bagging. Overfitting is caused by high variance. Variance reduction methods such as bagging can help. Indeed, best results are often obtained by bagging overfitted classifiers (e.g., unpruned decision trees, over-trained neural networks) than by bagging well-fitted classifiers (e.g., pruned trees).
- Randomized Committees. We can train several hypotheses h_i using different random starting weights for backpropagation
- Random Forests. Grow many decision trees and vote them. When growing each tree, randomly (at each node) choose a subset of the available features (e.g., $\sqrt{n}$ out of n features). Compute the best split using only those features.

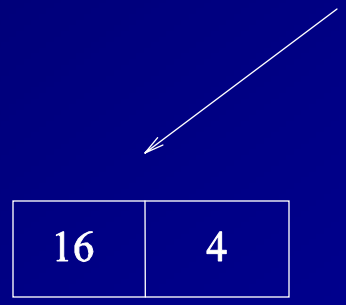
Overfitting Summary

- Minimizing training set error (ε_{train}) does not necessarily minimize test set error (ε_{test}).
  - This is true when the hypothesis space is too large (too expressive)
- Penalty methods add a penalty to ε_{train} to approximate ε_{test}
  - Bayesian, MDL, and Structural Risk Minimization
- Holdout and Cross-Validation methods without a subset of the training data, S_{eval}, to determine the proper hypothesis space H_i and its complexity
- Ensemble Methods take a combination of several hypotheses, which tends to cancel out overfitting errors

Penalty Methods for decision trees, neural networks, and SVMs

- Decision Trees
  - pessimistic pruning
  - MDL pruning
- Neural Networks
  - weight decay
  - weight elimination
  - pruning methods
- Support Vector Machines
  - maximizing the margin

Pessimistic Pruning of Decision Trees



- Error rate on training data is 4/20 = 0.20 = p.
- Binomial confidence interval (using the normal approximation to the binomial distribution) is

$$p - z_{\alpha/2} \cdot \sqrt{\frac{p(1-p)}{n}} <= p <= p + z_{\alpha/2} \cdot \sqrt{\frac{p(1-p)}{n}}$$

- If we use α = 0.25, then $z_{\alpha/2}$ = 1.150 so we obtain

  $0.097141 \leq p \leq 0.302859$

- We use the upper bound of this as our error rate estimate. Hence, we estimate $0.302859 \times 20 = 6.06$ errors

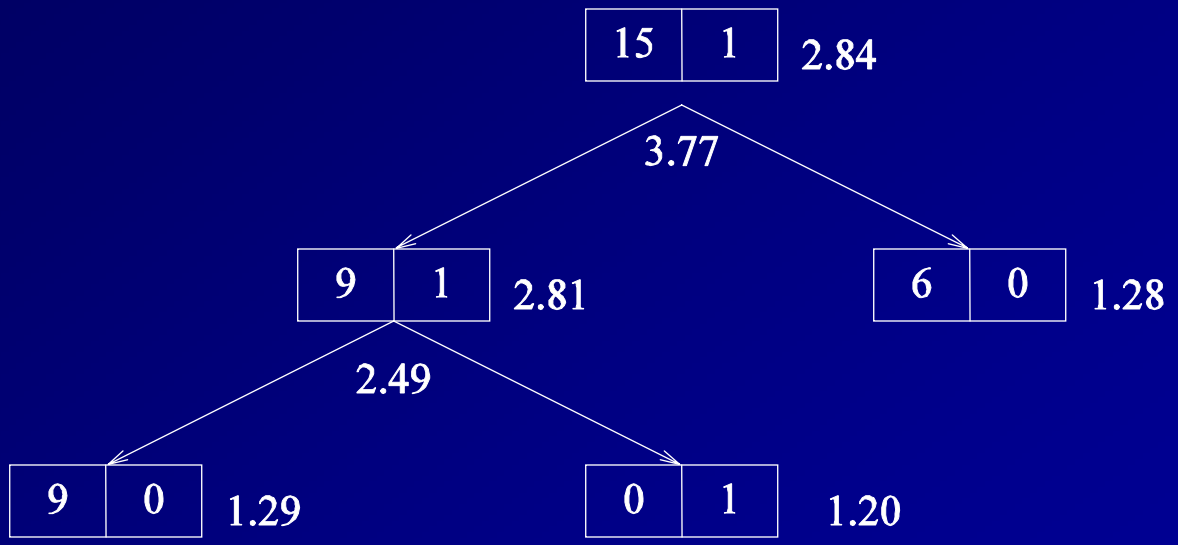
Pruning Algorithm (1): Traversing the Tree

```
float Prune(Node & node)
{
  if (node.leaf) return PessError(node);
  float childError = Prune(node.left) + Prune(node.right);
  float prunedError = PessError(node);
  if (prunedError < childError) {  // prune
    node.leaf = true;
    node.left = node.right = NULL;
    return prunedError}
  else  // don't prune
    return childError;
}
```

Pruning Algorithm (2): Computing the Pessimistic Error

```
const float zalpha2 = 1.150; // p = 0.25 two-sided
float PessError(Node & node)
{
  float n = node.class[0] + node.class[1];
  float nl = n + 2.0;
  float wrong = min(node.class[0], node.class[1]) + 1.0;
  // Laplace estimate of error rate
  float p =  wrong / nl;
  return n * (p + zalpha2 * sqrt( p * (1.0 - p) / n));
}
```

Pessimistic Pruning Example



Penalty methods for Neural Networks

- Weight Decay

$$J_i(W) = \frac{1}{2}(\hat{y}_i - y_i)^2 + \lambda \sum_j w_j^2$$

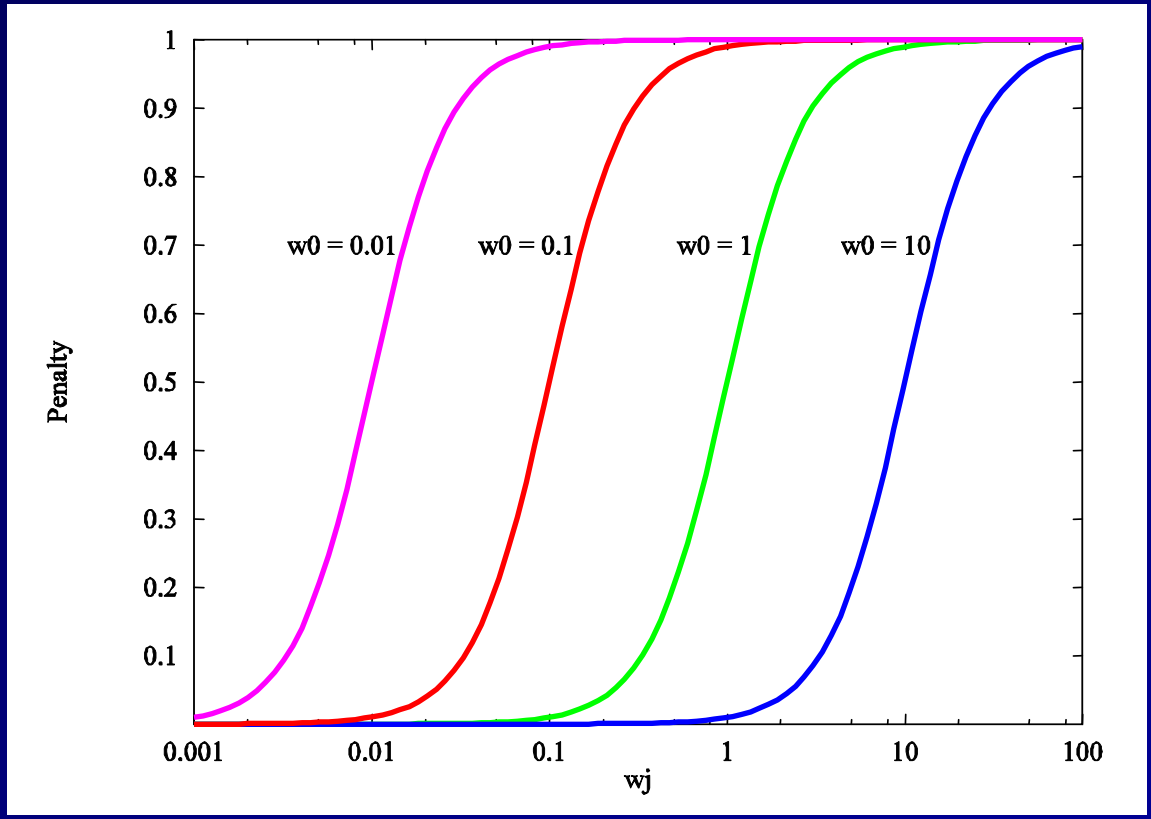
- Weight Elimination

$$J_i(W) = \frac{1}{2}(\hat{y}_i - y_i)^2 + \lambda \sum_j \frac{w_j^2/w_0^2}{1 + w_j^2/w_0^2}$$

w_0 large encourages many small weights

w_0 small encourages a few large weights

Weight Elimination



- This essentially counts the number of large weights. Once they are large enough, their penalty does not change

Neural Network Pruning Methods: Optimal Brain Damage

(LeCun, Denker, Solla, 1990)

Taylor's Series Expansion of the Squared Error:

$$\Delta J(W) = \sum_j g_j \Delta w_j + \frac{1}{2} \sum_j h_{jj} (\Delta w_j)^2 + \frac{1}{2} \sum_{j \neq k} h_{jk} \Delta w_j \Delta w_k + O(||\Delta w_j||^3)$$

where

$$g_j = \frac{\partial J(W)}{\partial w_j} \qquad \text{and} \qquad h_{jk} = \frac{\partial^2 J(W)}{\partial w_j \partial w_k}$$

At a local minimum, $g_j = 0$.

Assume off-diagonal terms $h_{jk} = 0$

$$\Delta J(W) = \frac{1}{2} \sum_j h_{jj} (\Delta w_j)^2$$

If we set $w_j = 0$, the error will change by $h_{jj} w_j^2 / 2$

Optimal Brain Damage Procedure

1. Choose a reasonable network architecture
2. Train the network until a reasonable solution is obtained
3. Compute the second derivatives h_{jj} for each weight w_j
4. Compute the saliencies for each weight $h_{jj}w_j\}^2/2$
5. Sort the weights by saliency and delete some low-saliency weights
6. Repeat from step 2

OBD Results

- On an OCR problem, they started with a highly-constrained and sparsely connected network with 2,578 weights, trained on 9300 training examples. They were able to delete more than 1000 weights without hurting training or test error.
- Optimal Brain Surgeon attempts to do this before reaching a local minimum
- Experimental evidence is mixed about whether this reduced overfitting, but it does reduce the computational cost of using the neural network

Penalty Methods for Support Vector Machines

- Our basic SVM tried to fit the training data perfectly (possibly by using kernels). However, this will quickly lead to overfitting.
- Recall margin-based bound. With probability 1 – δ, a linear separator with unit weight vector and margin γ on training data lying in a ball of radius R will have an error rate on new data points bounded by

$$\epsilon <= \frac{C}{m}\left[\frac{R^2 + \|\xi\|^2}{\gamma^2}\log^2 m + \log\frac{1}{\delta}\right]$$

For some constant C. ξ is the margin slack vector such that

$$\xi_i = \max\{0, \gamma - y_i g(\mathbf{x}_i)\}$$

Preventing SVM Overfitting

- Maximize margin γ
- Minimize slack vector $||\xi||$
- Minimize R

- The (reciprocal) of the margin acts as a penalty to prevent overfitting

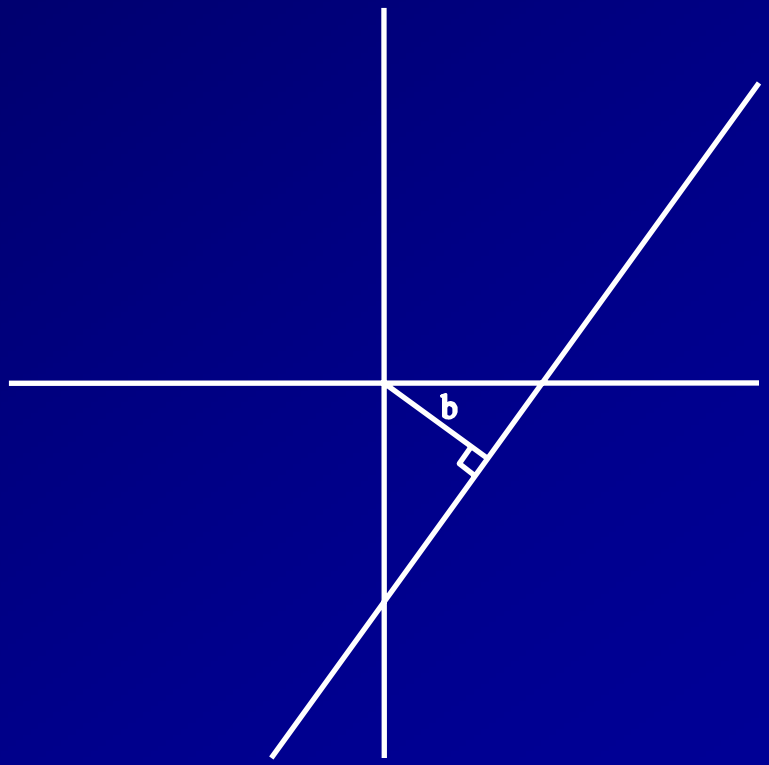
Functional Margin versus Geometric Margin

- Functional margin: $\gamma_f = y_i \cdot \mathbf{w} \cdot \mathbf{x}_i$
- Geometric margin: $\gamma_g = \gamma_f / \|\mathbf{w}\|$

  The margin bound applies only to the geometric margin γ_g
- The functional margin can be made arbitrarily large by rescaling the weight vector, but the geometric margin is invariant to scaling

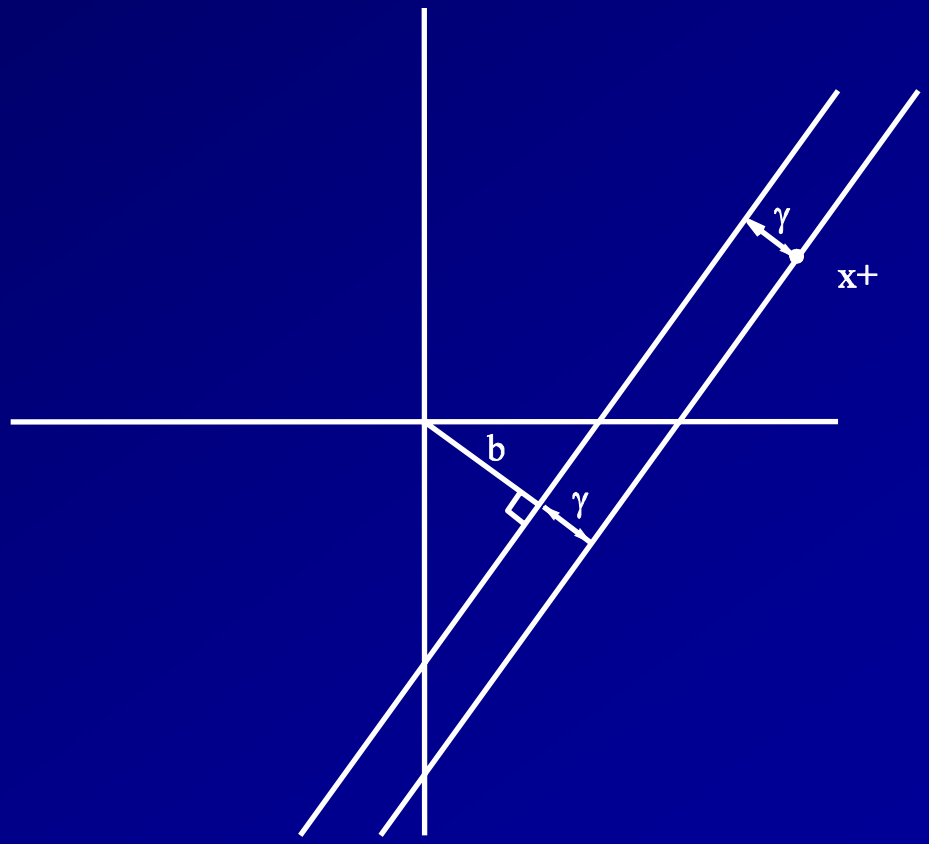
Intermission: Geometry of Lines

- Consider the line $\mathbf{w} \cdot \mathbf{x} + b = 0$, where $||\mathbf{w}|| = 1$ is a vector of unit length. Then the minimum distance to the origin is b.



Geometry of a Margin

- If a point $\mathbf{x}^+$ is a distance γ away from the line, then it lies on the line $\mathbf{w} \cdot \mathbf{x} + b = \gamma$



The Geometric Margin is the Inverse of ||w||

- Lemma: $\gamma_g = 1/||\mathbf{w}||$
- Proof:
  - Let **w** be an arbitrary weight vector such that the positive point $\mathbf{x}^+$ has a functional margin of 1. Then

    $\mathbf{w} \cdot \mathbf{x}^+ + b = 1$
  - Now normalize this equation by dividing by ||**w**||.

$$\frac{\mathbf{w}}{\|\mathbf{w}\|} \cdot \mathbf{x}^+ + \frac{b}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|} = \gamma_g$$

  - Implication: We can hold the functional margin at 1 and minimize the norm of the weight vector

Support Vector Machine Quadratic Program

- Find $\mathbf{w}$
- Minimize $||\mathbf{w}||^2$
- Subject to

  $y_i \cdot (\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$
- This requires every training example to have a functional margin of at least 1 and then maximizes the geometric margin. However it still requires perfectly fitting the data

Handling Non-Separable Data: Introduce Margin Slack Variables

- Find: $\mathbf{w}, \xi$
- Minimize: $||\mathbf{w}||^2 + C||\xi||^2$
- Subject to:

$$y_i \cdot (\mathbf{w} \cdot \mathbf{x}_i + b) + \xi_i \geq 1$$

  - ξ_i is positive only if example $\mathbf{x}_i$ does not have a functional margin of at least 1
  - $||\xi||^2$ measures how well the SVM fits the training data
  - $||\mathbf{w}||^2$ is the penalty term
  - C is the tradeoff parameter that determines the relative weight of the penalty compared to the fit to the data

Kernel Trick Form of SVM

- To apply the Kernel Trick, we need to reformulate the SVM quadratic program so that it only involves dot products between training examples. This can be done by an operation called the Lagrange Dual

Lagrange Dual Problem

- Find α_i
- Minimize

  $\sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j y_i y_j \alpha_i \alpha_j [\mathbf{x}_i \cdot \mathbf{x}_j + \delta_{ij}/C]$
- Subject to

  $\sum_i y_i \alpha_i = 0$

  $\alpha_i \geq 0$
- where $\delta_{ij} = 1$ if i = j and 0 otherwise.

Kernel Trick Form

- Find α_i
- Minimize

  $\sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j y_i y_j \alpha_i \alpha_j [K(\mathbf{x}_i,\mathbf{x}_j) + \delta_{ij}/C]$
- Subject to

  $\sum_i y_i \alpha_i = 0$

  $\alpha_i \geq 0$
- where $\delta_{ij} = 1$ if i = j and 0 otherwise.

Resulting Classifier

- The resulting classifier is

  $f(\mathbf{x}) = \sum_j y_j \alpha_j K(\mathbf{x}_j, \mathbf{x}) + b$

  where b is chosen by finding an *i* with $\alpha_i > 0$ and solving

  $y_i f(\mathbf{x}_i) = 1 - \alpha_i / C$

  for $f(\mathbf{x}_i)$

Variations on the SVM Problem: Variation 1: Use L_1 norm of ξ

- This is the "official" SVM, which was originally published by Vapnik and Cortes
- Find: $\mathbf{w}$, ξ
- Minimize: $||\mathbf{w}||^2 + C||\xi||$
- Subject to:

$$y_i \cdot (\mathbf{w} \cdot \mathbf{x}_i + b) + \xi_i \geq 1$$

Dual Form of L_1 SVM

- Find α_i
- Minimize

$$\sum_i \alpha_i - \tfrac{1}{2} \sum_i \sum_j y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j)$$

- Subject to

$$\sum_i y_i \alpha_i = 0$$

$$C \geq \alpha_i \geq 0$$

Variation 2: Linear Programming SVMs Use L_1 norm for **w** too

- Find **u**, **v**, ξ
- Minimize $\sum_j u_j + \sum_j v_j + C \sum_i \xi_i$
- Subject to

$$y_i \cdot ((\mathbf{u} - \mathbf{v}) \cdot \mathbf{x}_i + b) + \xi_i \geq 1$$

- The kernel form of this is
- Find α_i, ξ
- Minimize $\sum_i \alpha_i + C \sum_i \xi_i$
- Subject to

$$\sum_j \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) + \xi_i \geq 1$$
$$\alpha_j \geq 0$$

Setting the Value of C

- We see that the full SVM algorithm requires choosing the value of C, which controls the tradeoff between fitting the data and obtaining a large margin.
- To set C, we could train an SVM with different values of C and plug the resulting C, γ, and ξ into the margin bounds theorem to choose the C that minimizes the bound on ε.
- In practice, this does not work well, and we must rely on holdout methods (next lecture).