Overfitting and Early Stopping

15-496/782: Artificial Neural Networks
David S. Touretzky

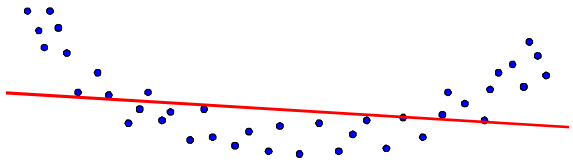
Spring 2004

(Based on lecture notes by Rich Caruana.)

Bias vs. Variance

- Bias: inability to match the training data.
  - The learner can only represent a certain class of functions: n-th order polynomials, sigmoid curves, etc.
  - The best it can do is pick the closest match to the data from within that class, e.g., fit a sine wave with a poly.
  - High bias = strong a priori constraints on the learner.
- Variance: how sensitive is the model to the choice of training set?
- All learners exhibit a bias-variance tradeoff.

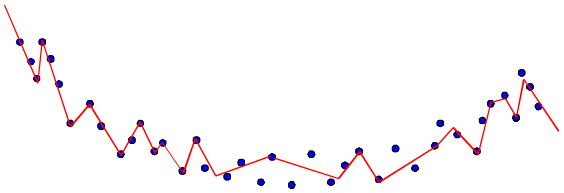
Linear Model: High Bias, Low Variance



High bias: "the world is linear".

Low variance: insensitive to slight changes in the training data.

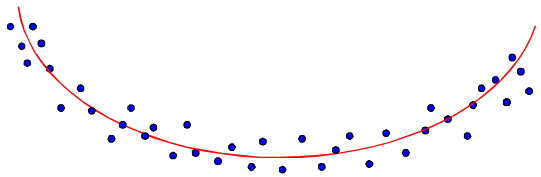
Overly Complex Model: Low Bias, High Variance



Low bias: can fit anything.

High variance: very sensitive to training data.

Example of a Good Bias-Variance Tradeoff



Moderate bias: elliptical arcs more complex than lines.

Low variance: not overly sensitive to training data.

Generalization Performance

Generalization: how "correct" is the network's behavior on novel inputs?

Can we predict generalization performance?

Low training set error is no guarantee of good generalization, due to **overfitting** (fitting noise.)

Example: train a polynomial approximator on data from:

$$y = \sin(x/3)+\nu \qquad x \in \{0,\ldots,20\}$$

ν is random noise in $[-0.25,+0.25]$

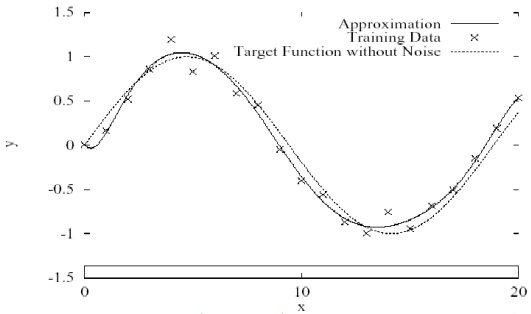
A 20th order poly will fit the training points exactly. But what order gives the best generalization?

Polynomial Fitting a Sine Wave

$y = \sin(x)+\nu$

10th order polynomial

moderately good fit



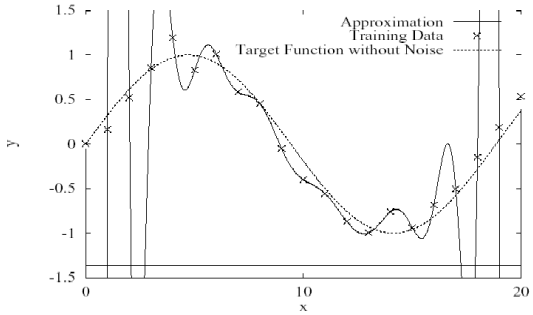
Train on points $x \in \{0,\ldots,20\}$. Test on points $x \in [0,20]$.

Polynomial Fitting a Sine Wave

$y = \sin(x)+\nu$

20th order polynomial

severe overfitting



Training a Neural Net on y = sin(x/3) + ν

Train on same 21 integer data points: {0, ..., 20}

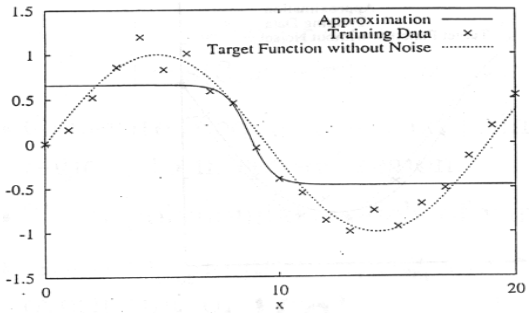
100,000 stochastic updates

Learning rate reduced linearly from 0.5 down to zero.

Try different numbers of hidden units...

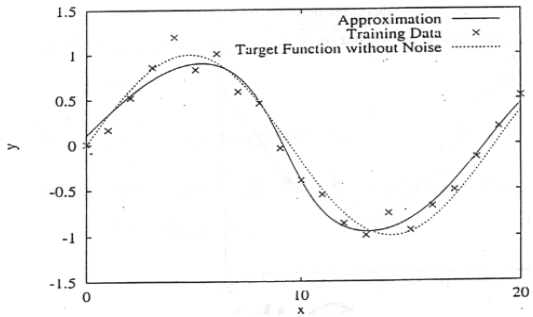
Neural Net Fitting a Sine Wave

y = sin(x)+ν
1 hidden unit: 4 parameters
poor fit



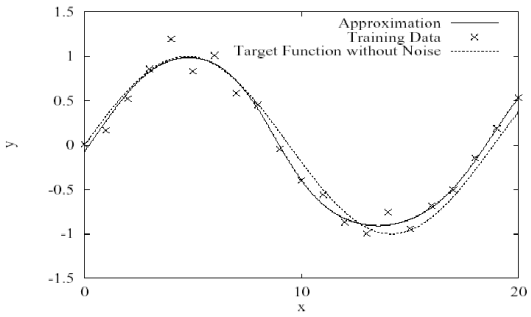
Neural Net Fitting a Sine Wave

y = sin(x)+ν
2 hidden units: 7 parameters
respectable fit



Neural Net Fitting a Sine Wave

y = sin(x)+ν
10 hidden units: 31 parameters
moderately good fit

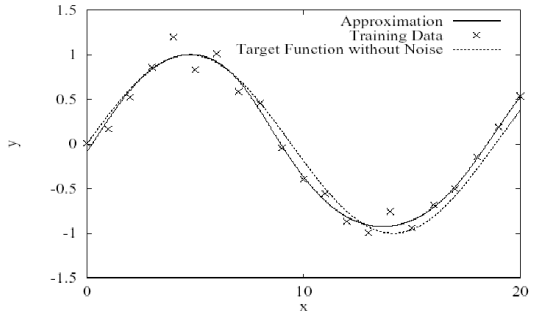


Neural Net Fitting a Sine Wave

$y = \sin(x) + \nu$

50 hidden units: 151 parameters

still moderately good fit



Neural Nets Don't Overfit?

- This is a myth.
- Even small nets can overfit in some situations.
- But big nets, with many parameters, may not overfit significantly more than small nets.

What Causes Overfitting?

- Too many free parameters?
  - High VC dimension: models are too complex:
  - Too many terms in the polynomial?
  - Too many weights in the neural net? No!
- Not enough training data to smooth things out?
  - Overfitting cannot occur with infinite training data.
- Training for too long?
- Something else?

Error Measures

Sum-squared error (SSE) is most commonly used.

$$E = \sum_p \left(d^p - y^p\right)^2$$

Cross-entropy is also popular:

$$E = -\sum_p \left[d^p \log y^p + (1-d^p)\log(1-y^p)\right]$$

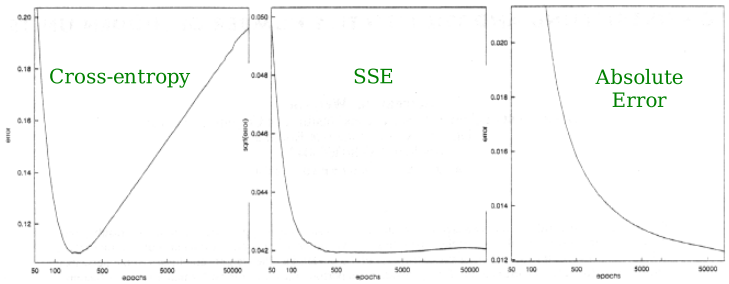
Cross-entropy strongly penalizes outputs that are far from their targets.

Log term diverges if $y^p=0$ and $d^p=1$ or vice versa.

Some binary decision problems not learnable using SSE can be learned using cross-entropy.

Onset of overfitting depends on the error measure.

Measuring Overfitting



Error rises for misclassified points as network's outputs become more certain.

Measurements by A. Weigend on a six-way phoneme classification task.

Overfitting Can Vary by Region

$$y = \begin{cases} -\cos(x)+\nu & 0 \leq x < \pi \\ \cos(3(x-\pi))+\nu & \pi \leq x \leq 2\pi \end{cases}$$

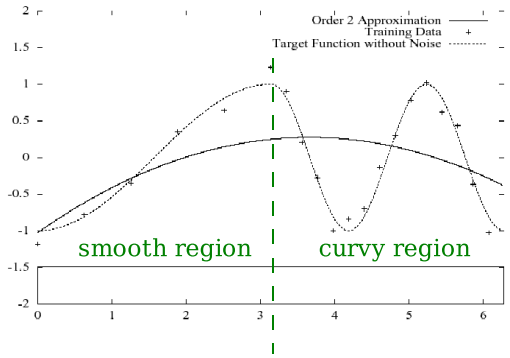
Function gets more curvy for $x > \pi$.

Generate 5 equally spaced points in the smooth region, and 15 points in the curvy region.

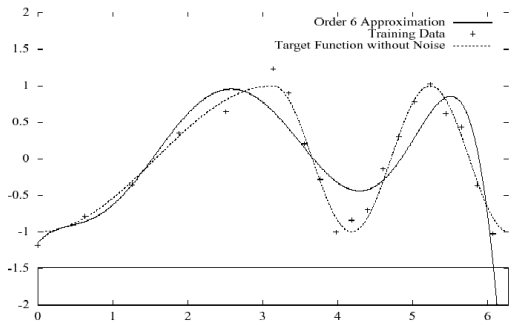
Train polynomial models of varying order.

Good fit on curvy region but overfitting on smooth one.

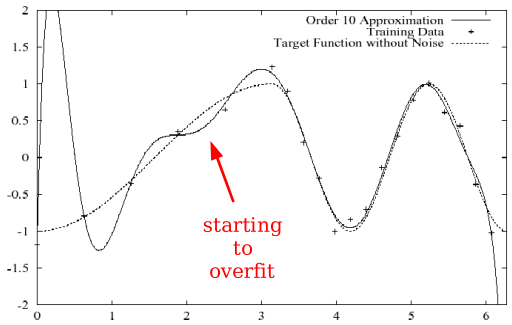
2nd Order Polynomial



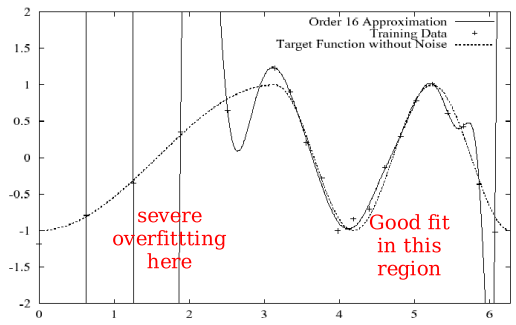
6th Order Polynomial



10th Order Polynomial



16th Order Polynomial



Training A Neural Net On the Same Data

Trained MLP on same 20 point dataset.

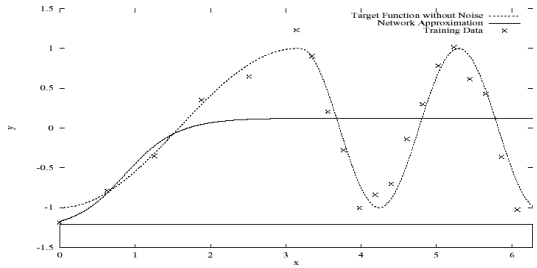
20,000 batch updates (not stochastic).

Learning rate decreased linearly from 0.5 to zero.

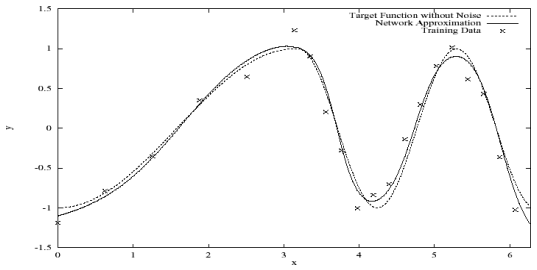
Small nets underfit.

Large nets fit both regions well, even with 100 hiddens.

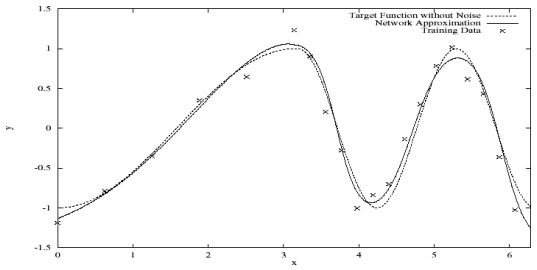
1 Hidden Unit



10 Hidden Units

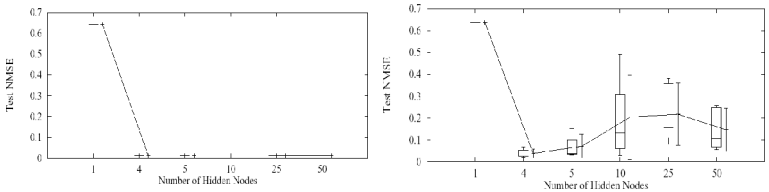


100 Hidden Units



BP Does Better than CG

Conjugate gradient gives lower error on training set but produces significant overfitting compared to BP.



Training with Backprop

Training with Conj. Gradient

Measuring Generalization Performance in a Neural Network

- Split the data into a **training set** and a separate "**validation set**".
- Error on the training set goes down monotonically as training progresses.
- But error on the validation set will go down and then back up: overfitting.
- A good bias/variance tradeoff produces good performance on the validation set.
  - So how do we find this tradeoff?

"Early Stopping"

- Halt training when error on **validation set** rises.
- Problems:
  - Final net not trained on all the data, since we held some back to form the validation set.
  - Early stopping point is based on just a small sample.
  - Measure is biased: to accurately predict generalization, we should measure performance on a separate "test set", distinct from the validation set, after early stopping.

Cross-Validation

- Goal: predict generalization performance as a function of some parameter, e.g.:
  - Number of hidden units.
  - Training time.
  - Amount of noise added to training set.

Cross-Validation Procedure

1) Partition the dataset into M sections.

2) For each section, train a separate network using the M-1 other sections as training data. Measure performance on the section that was left out.

3) Estimate generalization as the average peformance of the M networks on their respective validation sets.

Leave-one-out method: most extreme case of cross-validation. Choose M = training set size.

Practical Early Stopping

- Use 5-fold cross-validation (M=5).
- Train each network on 80% of the data.
- Early stopping based on perf. on remaining 20%.
- Note: each network may be trained for a different number of epochs.
- Final model is the averaged output of all 5 networks.

Study of Network Generalization

Looked at generalization performance on 7 problems:

- NETtalk
- 7 and 12 bit parity (nasty problems)
- Inverse kinematics for four-joint arm
- Robot modeling (Base1) and robot sonar (Base 2)
- ALVINN

Problems differed in a number of key dimensions:

- Binary vs. continuous inputs/outputs
- Large vs. small input space
- Presence/absence of noise

Generalization Study (cont.)

Used small training sets: 100-1000 points.

Varied hidden units from 2 to 800.

All nets trained with stochastic updates, learning rate 0.1, momentum 0.9, early stopping.

Plotted generalization as a function of training epochs.

Test Problems

NETtalk

- Words randomly selected from corpus.
- Boolean inputs and outputs: 0 = 0.1; 1 = 0.9
- 250 words (1800 patterns) in each of train, test sets
- Update every 10 patterns

7 bit parity

- 64 patterns in training set; 64 in test set
- Boolean inputs and outputs
- Update on every pattern

Test Problems

Inverse kinematics for a robot manipulator

- Map goal (x,y,z,θ) to four joint angles.
- No excess degrees of freedom (unique solution).
- Four continuous inputs; four continuous outputs.
- Linear in most regions; very nonliear in some.
- Code based on a real arm.
- Update on every pattern.
- Runs with 25, 250, 2500 patterns in training set.

Test Problems

Base 1 (robotic platform):

- Map (distance, angle, action) to (distance, angle)
- Three continuous inputs, two continuous outputs.
- Real world data; multiple sources of noise.
- 486 training patterns; 156 test patterns

Base 2 (robotic sonar)

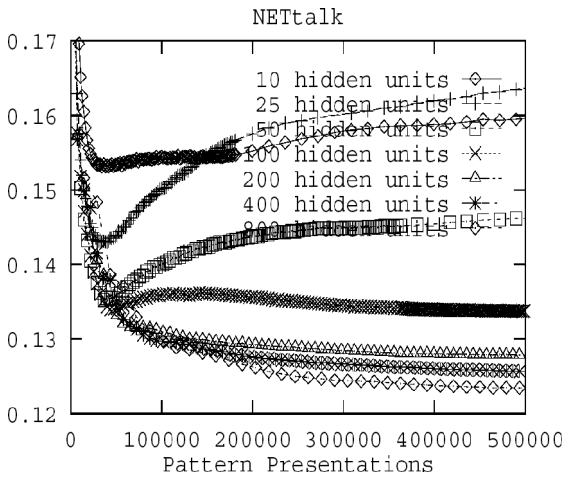
- Map (sonar sweep, base action) to new sonar sweep
- 32 continuous inputs, 31 continuous outputs
- Update every epoch.
- 118 training patterns; 118 test patterns.

Test Problems

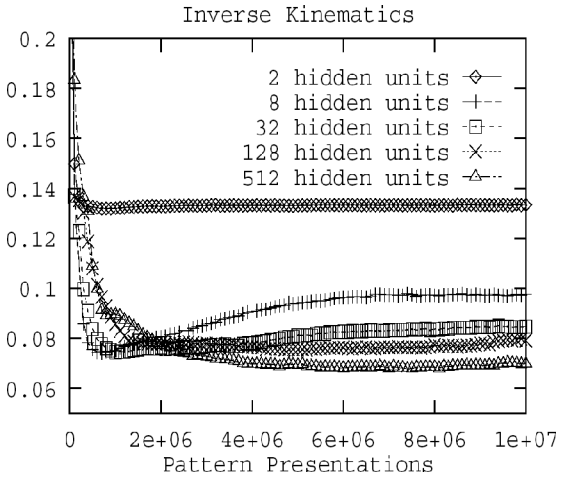
Simulated ALVINN

- Learn steering direction for simulated road images.
- 960 boolean inputs, 50 continuous outputs.
- Distributed gaussian output representation.
- Update every epoch.
- 100 training patterns; 100 test patterns.

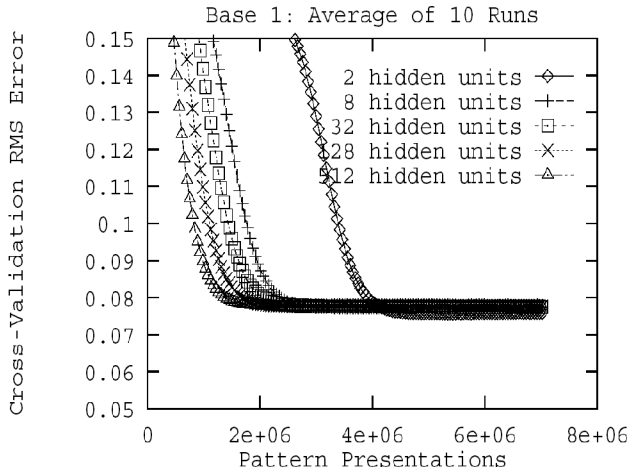
Generalization Curves



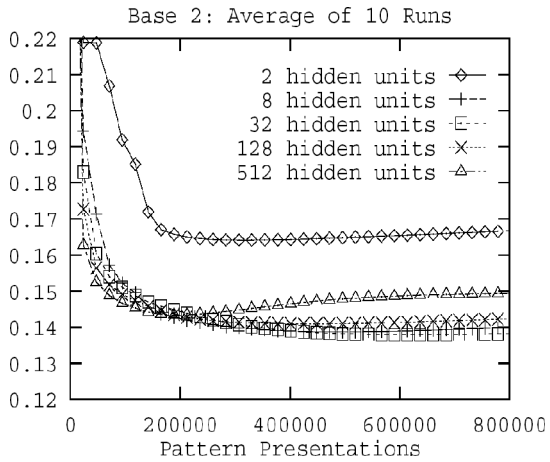
Generalization Curves



Generalization Curves



Generalization Curves



Results

- Large nets perform very well.
- Only 3 of the 7 problems showed any drop in performance for large nets.
- The drop was very slight.
- NETtalk showed dramatic overfitting for small nets, not for large nets.
- Results show that early stopping is needed even for small nets.

Why Excess Capacity is Okay

- Network is initialized with all weights small.
- This limits its representational capacity.
- Nets with large weights have more representational power: can represent more complex functions.
- But weights can grow only after many updates.
- So hypotheses with small weights are considered before those with large weights.
- Simple hypotheses are explored first!

Comparing Small vs. Large Nets

Train small nets with 2-400 hiddens using early stopping to get the best network of each size.

Compare i/o behavior of large net (800 hiddens) against optimally trained smaller nets:

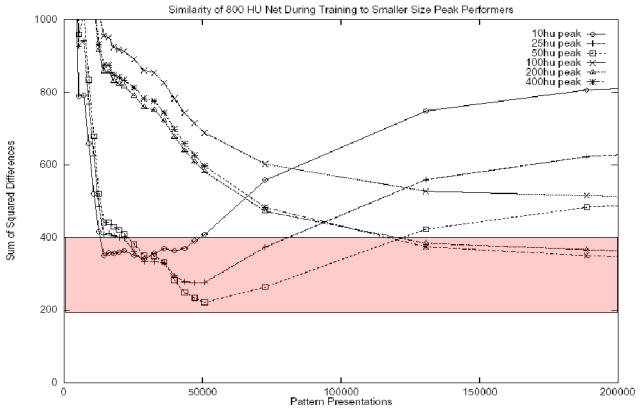
- Compute SSE of large net's output vs. small net.
- Small values imply the nets are computing similar things.

Minimum SSE between large net and smaller nets was 200-400.

By comparison, SSE on training set was around 1600.

So the small and large models are performing similarly.

Comparing Small vs. Large (cont.)



What Do Large Nets Learn?

Large net's performance was initially closest to the net with 10 hidden units.

Then 25 hiddens, then 50, etc.

Large net learns a sequence of models corresponding to each of the smaller nets in turn.

If early stopping is used, large net learns a model that is similar to the best model that could be learned by a smaller net.

Hidden Unit Variance, Covariance

$$\langle x_i \rangle = \frac{1}{N} \sum_p x_i^{(p)}$$

$$VAR_i = \frac{1}{N-1} \sum_p \left(x_i^{(p)} - \langle x_i \rangle \right)^2$$

$$STD_i = \sqrt{VAR_i}$$

$$COV_{ij} = \frac{1}{N-1} \sum_p \left(x_i^{(p)} - \langle x_i \rangle \right) \left(x_j^{(p)} - \langle x_j \rangle \right)$$

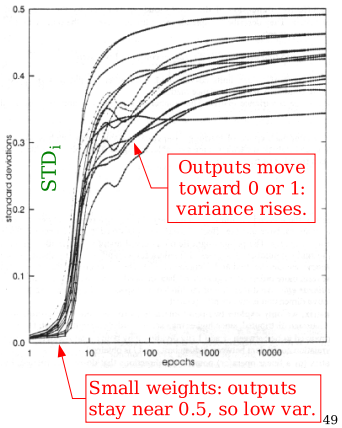
When Do the Hidden Units Learn?

Weigend analyzed hidden unit learning trajectories on a phoneme classification task.

160 inputs, 15 hiddens units, 6 outputs. Cross-entropy error measure.

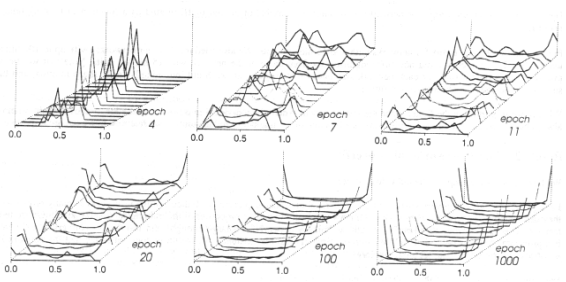
Plot standard deviation of hidden unit activity over the training set as learning progresses.

Result: all the hidden units "wake up" at about the same time.



What Do Hidden Units Learn?

Units developed into binary-valued feature detectors:



Histogram of hidden unit activation values.

"Effective Number" of Hidden Units

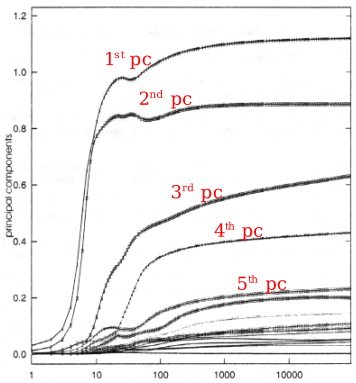
Looked at covariance matrix of hidden unit activation patterns.

How many eigenvectors does the matrix have at each time point?

Estimates the "effective" number of hidden units; the rest are doing redundant things.

Plot sqrts of the eigenvalues.

Result: the network is extracting the principal components sequentially!



Conclusions

- Backprop nets' inductive bias favors models with smooth solutions.
- Backprop nets with excess capacity explore simpler, smoother models first.
- Overfitting is possible in MLPs of all sizes.
- Conjugate gradient learning is more prone to overfitting than backprop.

Conclusions

• Early stopping can prevent overfitting and allow a net to take advantage of excess capacity.

• Backprop uses excess capacity to fit more closely in regions of high nonlinearity without overfitting in the linear regions.

• It doesn't pay to try to keep your networks small. Use the largest network you can afford to train.

Conclusions

• Weigend: all hidden units “wake up” at the same time.

• Initially they all do the same thing.

• Hidden units differentiate over time: the number of “effective” hidden units gradually increases.

• The principal components are learned in sequence.

• Backprop nets progress through a series of models with increasing numbers of effective hidden units. So early stopping favors simple models.