Chapter 11: Neural Networks

DD3364

December 16, 2012

Projection Pursuit Regression

Projection Pursuit Regression

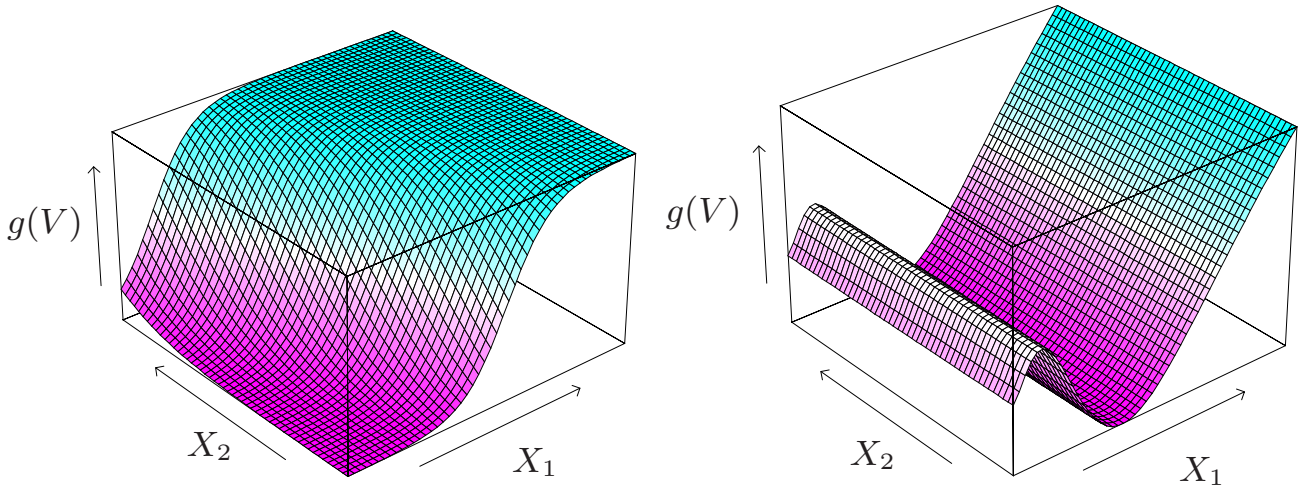
Projection **P**ursuit **R**egression model:

$$f(X) = \sum_{i=1}^{M} g_m(w_m^t X)$$

where $X \in \mathbb{R}^p$ and have targets $Y \in \mathbb{R}$.

- Additive model in the derived features $V_m = w_m^t X$.
- $g_m(w_m^t X)$ the ridge function in $\mathbb{R}^p$ - only varies in direction of w_m.
- PPR model can approximate any continuous function in $\mathbb{R}^p$ if M arbitrarily large and appropriate choice of g_m's.
- $\implies$ PPR model is a universal approximator.

Example Ridge Functions



- **Left graph**

$$g(V) = \frac{1}{1 + \exp\{-5(V - 0.5)\}}, \quad V = \frac{(X_1 + X_2)}{\sqrt{2}}$$

- **Right graph**

$$g(V) = (V + 0.1)\sin\left(\frac{1}{V/3 + .1}\right), \quad V = X_1$$

How to fit a PPR model ?

- Have training data $\{(x_i, y_i)\}_{i=1}^n$.
- Seek to minimize

$$\sum_{i=1}^{n} \left[y_i - \sum_{m=1}^{M} g_m(w_m^t x_i) \right]^2$$

  over functions g_m and directions w_m, $m = 1, \ldots, M$.

- **How**??
- General approach
  - Build model in a forward stage-wise manner. Add a pair (w_m, g_m) at each stage.
  - At each stage iterate
    * Fix w_m and update g_m
    * Fix g_m and update w_m

How to fit a PPR model ?

- Have training data $\{(x_i, y_i)\}_{i=1}^n$.
- Seek to minimize

$$\sum_{i=1}^{n} \left[y_i - \sum_{m=1}^{M} g_m(w_m^t x_i) \right]^2$$

  over functions g_m and directions w_m, $m = 1, \ldots, M$.

- **How**??
- General approach
  - Build model in a forward stage-wise manner. Add a pair (w_m, g_m) at each stage.
  - At each stage iterate
    * Fix w_m and update g_m
    * Fix g_m and update w_m

How to fit a PPR model ?

- **Fix w and update g**
- Must impose complexity constraints on g_m to avoid overfitting.
- **Fix g and update w**

$$g(w^t x_i) \approx g(w_{\text{old}}^t x_i) + g'(w_{\text{old}}^t x_i)(w - w_{\text{old}})^t x_i$$

to give

$$\sum_{i=1}^{n} \left[y_i - g(w^t x_i)\right]^2 \approx \sum_{i=1}^{n} g'(w_{\text{old}}^t x_i)^2 \left[\left(w_{\text{old}}^t x_i + \frac{y_i - g(w_{\text{old}}^t x_i)}{g'(w_{\text{old}}^t x_i)}\right) - w^t x_i\right]^2$$

To minimize the rhs:

- Perform least squares regression (no intercept (bias) term).
- Input $w^t x_i$ has target $w_{\text{old}}^t x_i + \dfrac{y_i - g(w_{\text{old}}^t x_i)}{g'(w_{\text{old}}^t x_i)}$
- Weight errors with $g'(w_{\text{old}}^t x_i)^2$
- This produces the updated coefficient vector w_{new}.

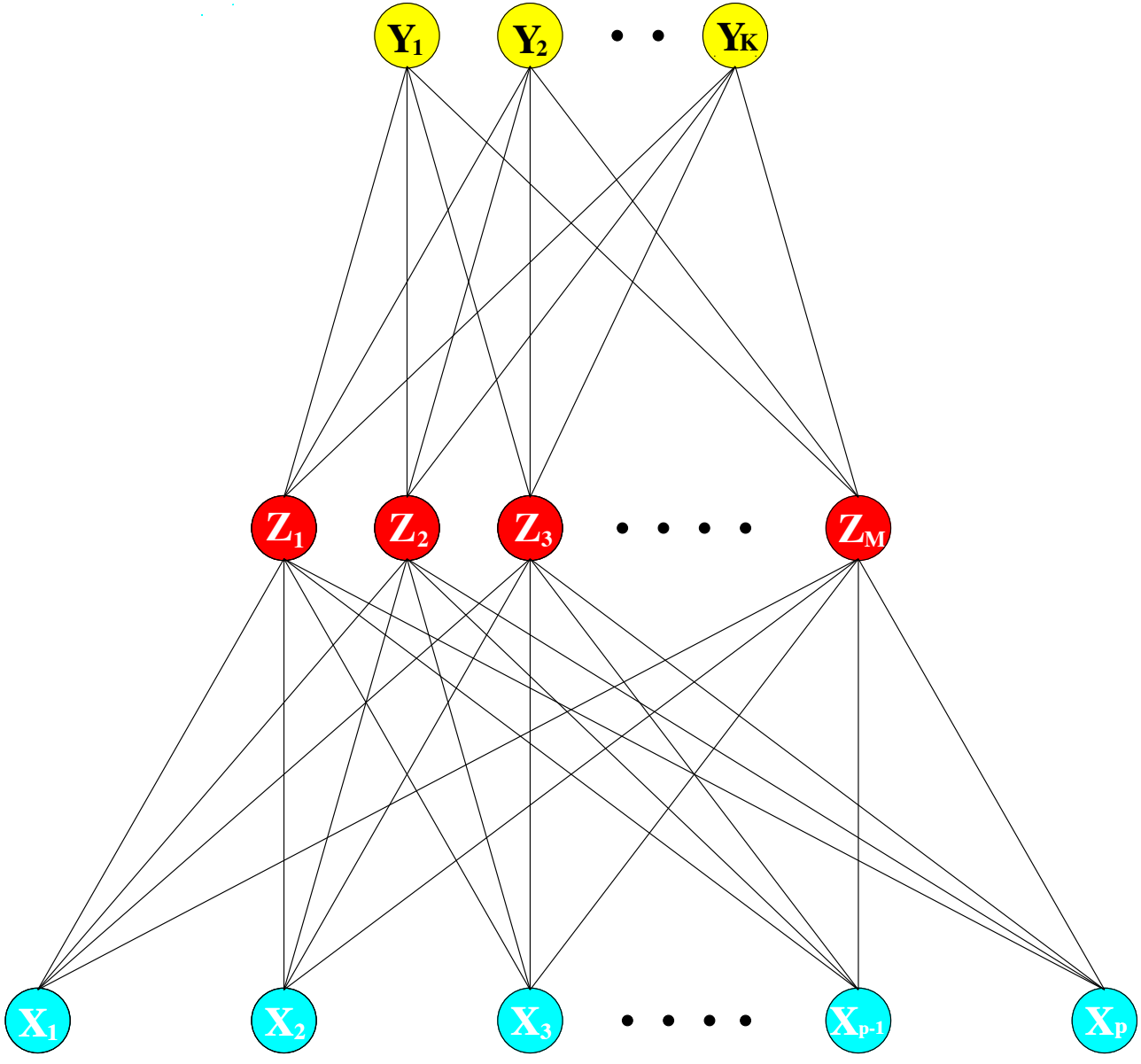
How to fit a PPR model ?

Iterate these two steps until convergence

- **Fix w and update g**
- **Fix g and update w**

Neural Networks

Single hidden layer, feed-forward network



K-classification

- **Input**: $X = (X_1, X_2, \ldots, X_p)$ and say it belongs to class k
- **Ideal output**: $Y_1, \ldots, Y_K$ where

$$Y_i = \begin{cases} 0 & \text{if } i \neq k \\ 1 & \text{if } i = k \end{cases}$$

- The 2 layer neural network estimates the outputs by
  - deriving features $Z_1, \ldots, Z_M$ - *hidden units* - from linear combinations of X
  - the target Y_k is modeled as a function of linear combinations of $Z_1, \ldots, Z_M$.
- In maths...

- Computation of the kth output

$$Y_k = f_k(X) = g_k(T_1, \ldots, T_K)$$

where

$$T_k = \beta_{k0} + \sum_{m=1}^{M} \beta_{km} Z_m,$$

$$Z_m = \sigma\left(\alpha_{m0} + \sum_{l=1}^{p} \alpha_{ml} X_l\right)$$

the activation function σ can be defined

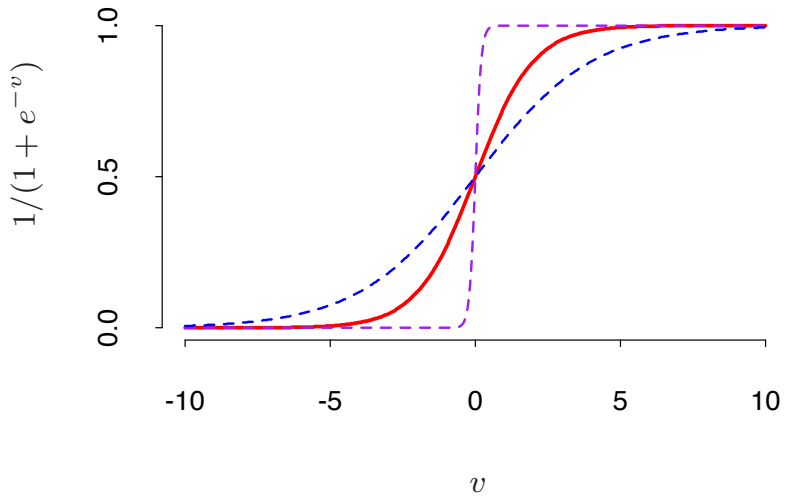
$$\sigma(v) = \frac{1}{1 + \exp\{-v\}} \quad \longleftarrow \textit{ sigmoid function}$$

and the output function g_k

$$g_k(T_1, \ldots, T_K) = \frac{\exp\{T_k\}}{\sum_{l=1}^{K} \exp\{T_l\}} \quad \longleftarrow \textit{ softmax function}$$

The Activation Function

Shown is $\sigma(sv)$ for $s = .5, 1, 10$



- If σ is the identity $\implies$ each $T_k = w_{k0} + \sum_{i=1}^{p} w_{kl} X_l$
- Can think of **neural networks** as a non-linear generalization of the linear model.
- Rate of activation of the *sigmoid* depends on the norm of α_m where $Z_m = \sigma\left(\alpha_{m0} + \sum_{l=1}^{p} \alpha_{ml} X_l\right)$
- When $\|\sigma_m\|$ small $\implies$ unit operates in the *linear part* of its activation function.

Neural Network is a universal approximator

A NN with one hidden units, can approximate arbitrarily well any functional continuous mapping from one finite dimensional space to another, provided number of hidden units is sufficiently large.

Fitting Neural Networks

Error measure

- This 2-layer neural network has unknown parameters, θ,

$$\{\alpha_{m0}, \alpha_{m1}, \ldots, \alpha_{mp};\ m = 1, \ldots, M\} \qquad M(p+1) \text{ weights}$$
$$\{\beta_{k0}, \beta_{k1}, \ldots, \beta_{kM};\ k = 1, \ldots, K\} \qquad K(M+1) \text{ weights}$$

- **Aim**: Estimate parameters, θ, from labeled training data:

$$\{x_i, g_i\}_{i=1}^{n} \quad \text{with each } x_i \in \mathbb{R}^p,\ g_i \in \{1, \ldots, K\}$$

- Do this by minimizing a measure-of-fit such as

$$R(\theta) = \sum_{i=1}^{n} \sum_{k=1}^{K} (y_{ik} - f_k(x_i))^2 \quad \longleftarrow \textbf{sum-of-squared} \text{ error}$$

or

$$R(\theta) = -\sum_{i=1}^{n} \sum_{k=1}^{K} y_{ik} \log f_k(x_i) \quad \longleftarrow \textbf{cross-entropy} \text{ error}$$

Minimizing $R(\theta)$

- Typically don't want

$$\hat{\theta} = \arg\min_{\theta} \; R(\theta)$$

  $\Longrightarrow$ an overfit solution.

- Some form of regularization is required - will come back to this.

- Generic approach to minimizing $R(\theta)$ is by gradient descent a.k.a. **back-propagation**.

- This amounts to implementation of the chain rule for differentiation.

Back-propagation for squared-error loss

- Let $z_i = (z_{1i}, \ldots, z_{Mi})$ and

$$z_{im} = \sigma(\alpha_{m0} + \alpha_m^t x_i) \quad \text{where } \alpha_m = (\alpha_{m1}, \ldots, \alpha_{mp})$$

- Have

$$R(\theta) = \sum_{i=1}^{n} R_i = \sum_{i=1}^{n} \sum_{k=1}^{K} (y_{ik} - f_k(x_i))^2$$

with derivatives

$$\frac{\partial R_i(\theta)}{\partial \beta_{km}} = -2\,(y_{ik} - f_k(x_i))\, g_k'(\beta_{10} + \beta_1^t z_i, \ldots, \beta_{K0} + \beta_K^t z_i)\, z_{im} = \delta_{ki}\, z_{im}$$

$$\frac{\partial R_i(\theta)}{\partial \alpha_{ml}} = \sum_{k=1}^{K} \delta_{ki}\, \beta_{km}\, \sigma'(\alpha_{m0} + \alpha_m^t x_i)\, x_{il}$$

$$= x_{il}\, \sigma'(\alpha_{m0} + \alpha_m^t x_i) \sum_{k=1}^{K} \delta_{ki}\, \beta_{km} = x_{il}\, s_{mi}$$

Back-propagation for squared-error loss

- Given these derivatives update at the $(r+1)$st iteration

$$\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{n} \frac{\partial R_i(\theta)}{\partial \beta_{km}}\bigg|_{\beta_{km}=\beta_{km}^{(r)}}$$

$$\alpha_{ml}^{(r+1)} = \alpha_{ml}^{(r)} - \gamma_r \sum_{i=1}^{n} \frac{\partial R_i(\theta)}{\partial \alpha_{ml}}\bigg|_{\alpha_{ml}=\alpha_{ml}^{(r)}}$$

  where γ_r is the learning rate.

- The quantities δ_{ki} and s_{mi} are "*errors*" from the current model at the output and hidden layer units respective

$$\frac{\partial R_i(\theta)}{\partial \beta_{km}} = \delta_{ki}\, z_{im}, \qquad \frac{\partial R_i(\theta)}{\partial \alpha_{ml}} = x_{il}\, s_{mi}$$

- Remember the errors satisfy

$$s_{mi} = \sigma'(\alpha_{m0} + \alpha_m^t x_i) \sum_{k=1}^{K} \delta_{ki}\, \beta_{km}$$

Back-propagation update equations

The updates can be implemented in a two-pass algorithm:

- **Forward pass**: current weights are fixed and compute $\hat{f}_k(x_i)$
- **Backward pass**: Compute errors δ_{ki} and then back-propagated with

$$s_{mi} = \sigma'(\alpha_{m0} + \alpha_m^t x_i) \sum_{k=1}^{K} \delta_{ki}\, \beta_{km}$$

  to give the errors s_{mi}.

- Use both sets of errors to compute the gradients for the updates.

Details of back propagation

- Can do updates with batch learning.

  Parameters updated by summing over all training examples.

- Can do updates with online learning.

  Parameters updated after each training example.

  $\Longrightarrow$ can train network with very large trained datasets.

- *Training epoch* $\equiv$ one sweep through the entire training set.

- **Learning rate**: γ_r
  - Batch learning - usually taken to be constant and can be optimized by a line search.
  - Online learning - $\gamma_r \to 0$ as $r \to \infty$

- **Note**: Back-prop is very slow.

Some Issues in Training Neural Networks

Training Neural Networks

- Training a neural networks is non-trivial!

  Why?

  - Model is overparametrized
  - Optimization problem is nonconvex and unstable

- Book summarizes some of the important issues...

Starting Values

- If weights are near zero
  $\implies$ $\sigma(\cdot)$ is roughly linear
  $\implies$ neural network collapses into an approx linear model.

- Usually start with random values close to zero.
  $\implies$ model starts out linear and becomes non-linear as weights increase.

- Use of exact zero weights gives zero derivatives, perfect symmetry and the algorithm never moves.

- Starting with large weights often leads to poor solutions.

Combating Overfitting

Neural networks will overfit at the global minimum of R.

Therefore different approaches to *regularization* have been adopted:

- **Early stopping**
  - Only train the model for a while.
  - Stop before converging to a minimum of $R(\theta)$.
  - As initial weights are close to 0
    $\Longrightarrow$ initially have a highly regularized linear solution
    $\Longrightarrow$ early stopping shrinks the model towards a linear model.
  - Can use a validation dataset to determine when to stop.
- **Weight decay**
  - Add a penalty to the error function $R(\theta) + \lambda J(\theta)$, where
    $$J(\theta) = \sum_{km} \beta_{km}^2 + \sum_{ml} \alpha_{ml}^2$$
    and $\lambda \geq 0$ is a tuning parameter.
  - Larger values of λ tend to shrink weights towards zero.

Combating Overfitting

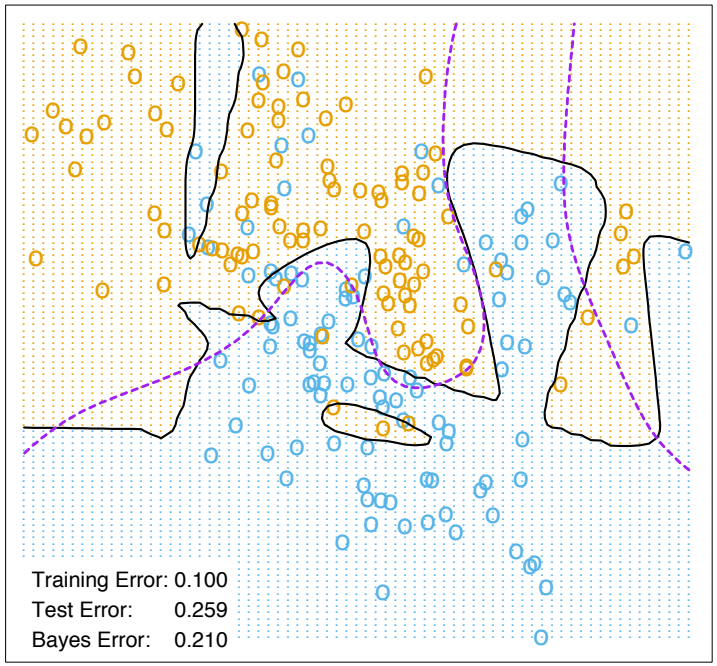
Neural networks will overfit at the global minimum of R.

Therefore different approaches to *regularization* have been adopted:

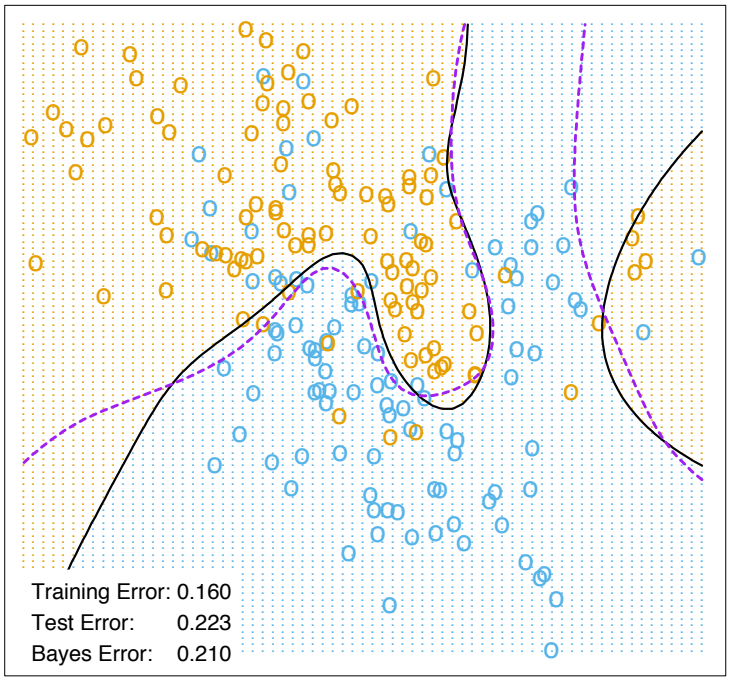
- **Early stopping**
  - Only train the model for a while.
  - Stop before converging to a minimum of $R(\theta)$.
  - As initial weights are close to 0
    $\Longrightarrow$ initially have a highly regularized linear solution
    $\Longrightarrow$ early stopping shrinks the model towards a linear model.
  - Can use a validation dataset to determine when to stop.
- **Weight decay**
  - Add a penalty to the error function $R(\theta) + \lambda J(\theta)$, where
    $$J(\theta) = \sum_{km} \beta_{km}^2 + \sum_{ml} \alpha_{ml}^2$$
    and $\lambda \geq 0$ is a tuning parameter.
  - Larger values of λ tend to shrink weights towards zero.

Effect of Weight Decay

Neural Network - 10 Units, No Weight Decay

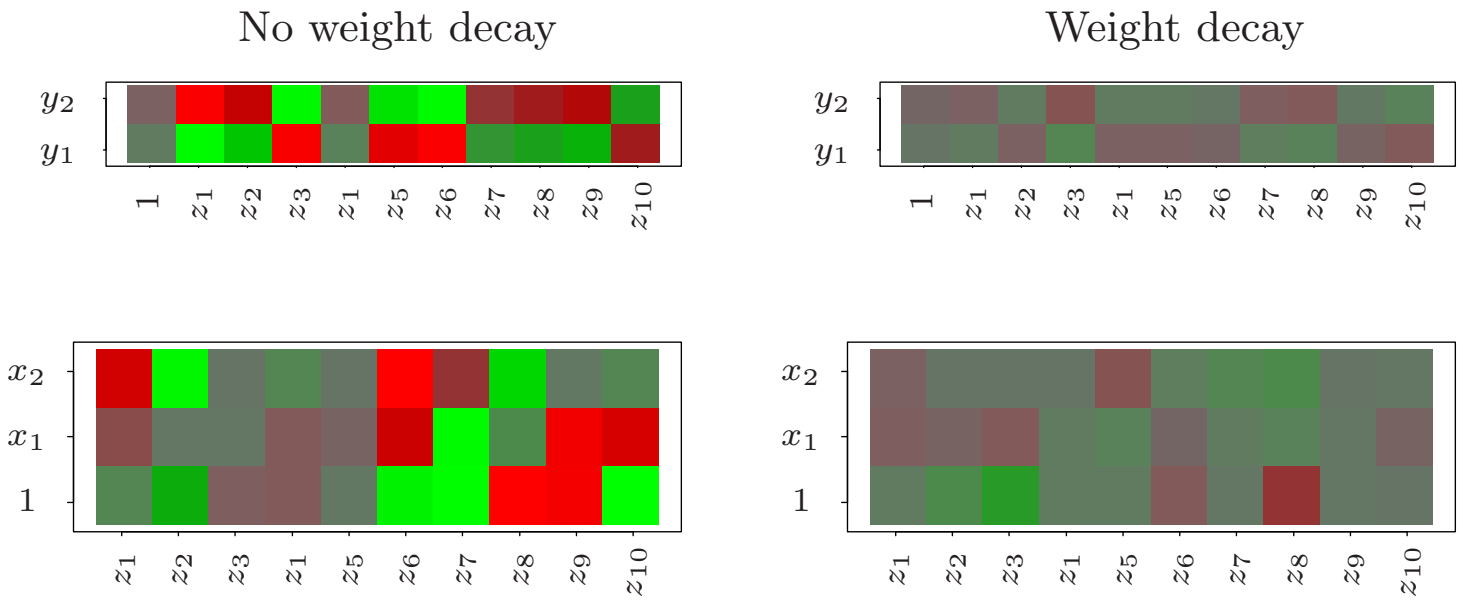


Neural Network - 10 Units, Weight Decay=0.02



- Both use softmax g_k and cross-entropy error.
- Bayes optimal decision boundary is the purple curve

Weights learnt



- Both use softmax g_k and cross-entropy error.
- The display ranges from bright green (negative) to bright red (positive).

Combating Overfitting

- **Scaling the Inputs**
  - Scale of inputs determines scale of bottom layer weights.
  - At beginning best to standardize all inputs to have mean 0 and standard deviation 1
  - Ensures all inputs are treated equally in the regularization process.
- **Number of Hidden Units and Layers**
  - Generally better to have too many than too few hidden units.
  - Fewer hidden units $\Longrightarrow$ less flexibility in the model
  - Proper regularization should shrink unnecessary hidden unit weights to zero.
  - Multiple hidden layers allows construction of hierarchical features at different resolutions.

- **Multiple Minima**
  - $R(\theta)$ non-convex $\Longrightarrow$ final solution depends on initial weights.
  - Option 1:
    Learn different networks for different random initial weights.
    Choose the network with lowest penalized error.
  - Option 2:
    Learn different networks for different random initial weights.
    For a test example average the prediction of each network.
  - Option 3: (bagging)
    Learn different networks from random subsets of the training data.
    For a test example average the prediction of each network.

Example: Simulated Data

Example 1: Underlying model

- Generated data from this additive model

$$Y = \sigma(a_1^t X) + \sigma(a_2^t X) + \epsilon$$

where

$$\begin{aligned} X &= (X_1, X_2)^t \quad \text{with } X_i \sim \mathcal{N}(0,1) \text{ for } i = 1,2 \\ a_1^t &= (3,3), \\ a_2^t &= (3,-3), \\ \epsilon &\sim \mathcal{N}(0,\sigma^2) \end{aligned}$$

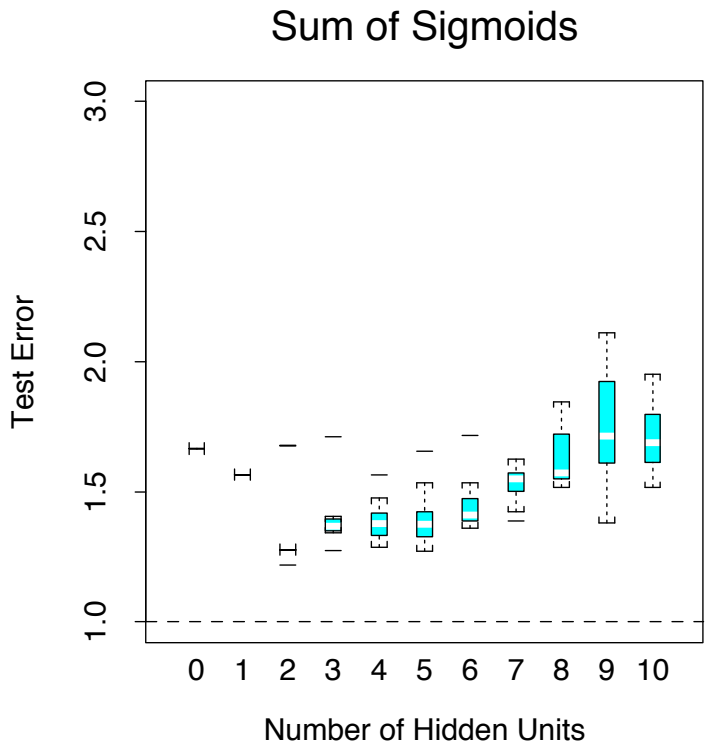
and σ^2 is chosen so the s-n-r is 4 that is

$$\text{Var}\{f(X)\} = 4\sigma^2$$

- $n_{\text{train}} = 100$ and $n_{\text{test}} = 10000$

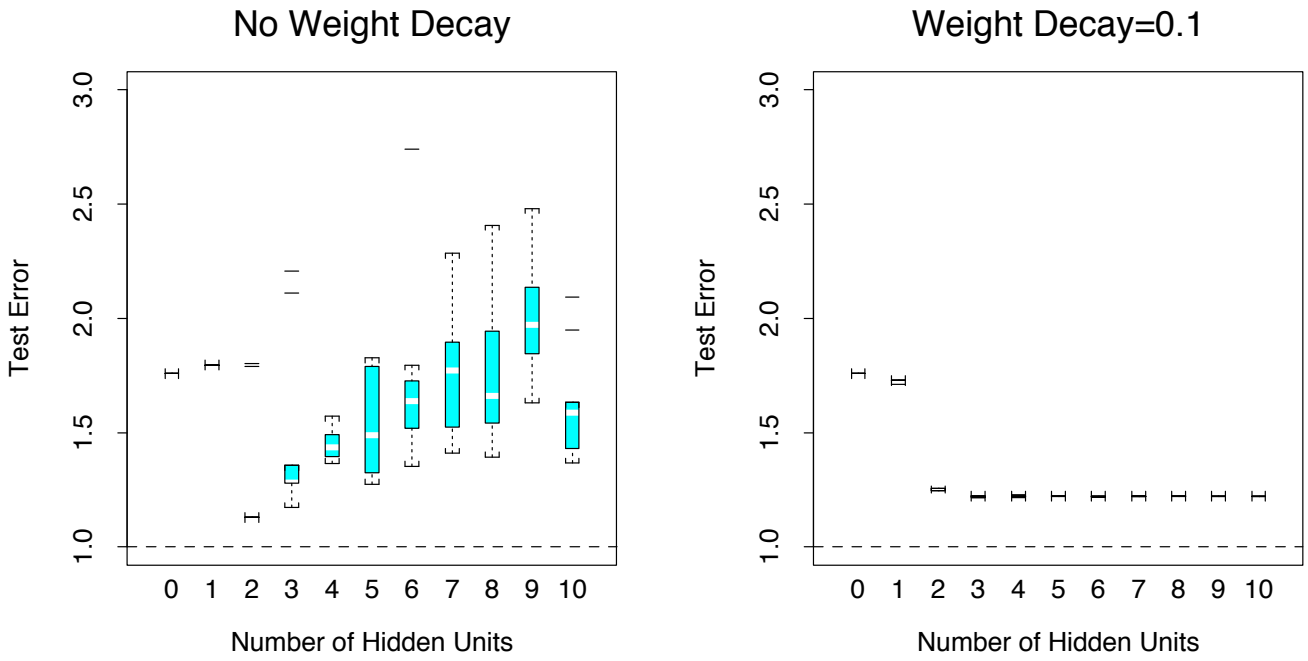
Example 1: Neural network fit

- Fit neural network with weight decay and various number of hidden units.
- Recorded the average test error for 10 random starting weights.
- Zero hidden unit model refers to linear least squares regression.

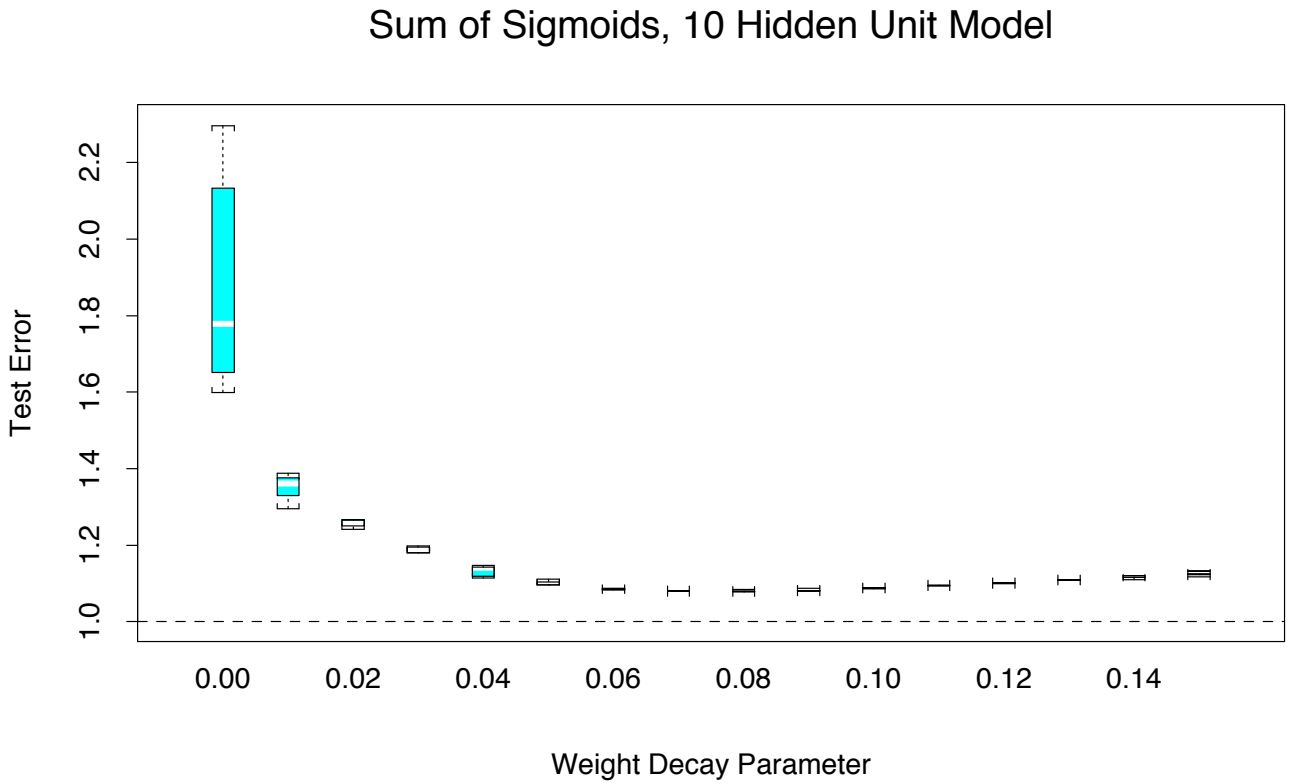


Test error quoted relative to the Bayes error, $\lambda = 0.0005$

Example 1: Effect of weight decay on test error



Example 1: Fixed number of hidden units, vary λ



Example 2: Underlying model

- Generated data from this additive model

$$Y = \prod_{j=1}^{10} \phi(X_j) + \epsilon$$

where

$$X = (X_1, \ldots, X_{10})^t \quad \text{with } X_i \sim \mathcal{N}(0,1) \text{ for } i = 1, 2$$
$$\phi(v) = \exp\{-v^2/2\}/\sqrt{2\pi},$$
$$\epsilon \sim \mathcal{N}(0, \sigma^2)$$

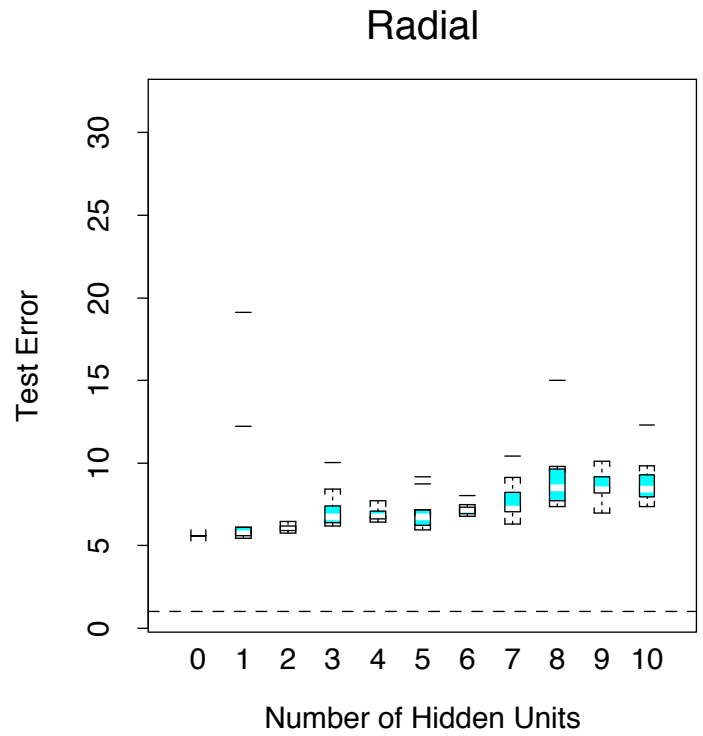
and σ^2 is chosen so the s-n-r is 4 that is

$$\text{Var}\{f(X)\} = 4\sigma^2$$

- $n_{\text{train}} = 100$ and $n_{\text{test}} = 10000$

Example 2: Neural network does not produce a good fit

- Fit neural network with weight decay and various number of hidden units.
- Recorded the average test error for 10 random starting weights.
- Zero hidden unit model refers to linear least squares regression.



Test error quoted relative to the Bayes error, $\lambda = 0.0005$

Convolutional Neural Networks

Neural Nets and image recognition tasks

- A **black box neural network** applied to pixel intensity data does not perform well for image pattern recognition task.
- **Why**?
  - Because the pixel representation of the images lack certain invariances (such as small rotations of the image).
  - Huge number of parameters
- **Solution**
  Introduce constraints on the network to allow for more complex connectivity but fewer parameters.
- **Prime Example**: Convolutional Neural Networks

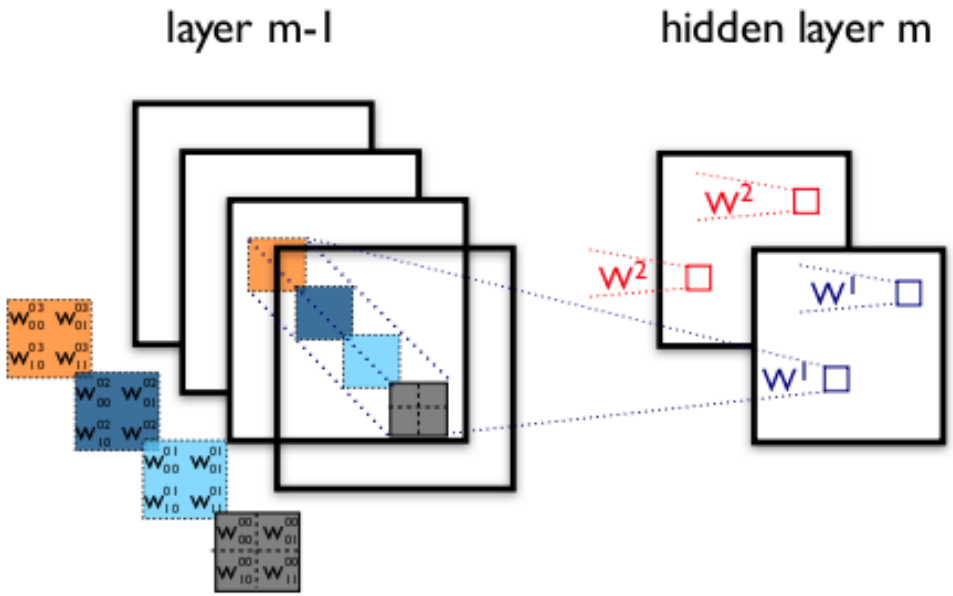
Neural Nets and image recognition tasks

- A **black box neural network** applied to pixel intensity data does not perform well for image pattern recognition task.
- **Why**?
  - Because the pixel representation of the images lack certain invariances (such as small rotations of the image).
  - Huge number of parameters
- **Solution**
  Introduce constraints on the network to allow for more complex connectivity but fewer parameters.
- **Prime Example**: Convolutional Neural Networks

Property 1: Convolutional Neural Networks

Sparse Connectivity

- CNNs exploit spatially local correlation by enforcing a local connectivity pattern between units of adjacent layers.
- The hidden units in the m-th layer are connected to a local subset of units in the $(m-1)$-th layer, which have spatially contiguous receptive fields.

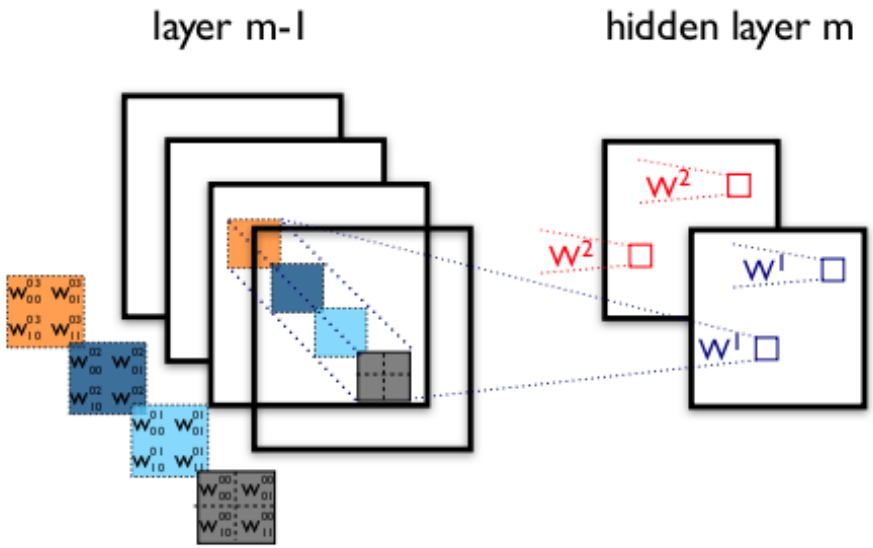


Example of a convolutional layer

Property 2: Convolutional Neural Networks

Shared Weights

- In CNNs, each sparse filter is replicated across the image.
- These "*replicated*" units form a feature map.



Example of a convolutional layer

The k-th feature map h^k, whose filters are defined by the weights W^k and bias b_k, is (with $\tanh$ used for non-linearities):

$$h^k_{ij} = \tanh\{(W^k * x)_{ij} + b_k\}$$

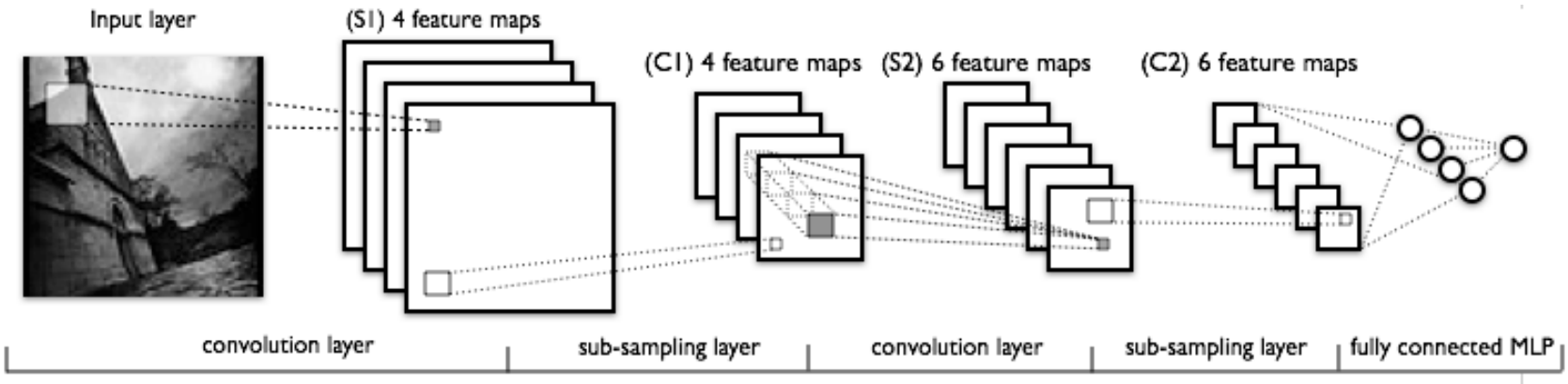
Property 3: Convolutional Neural Networks

Max Pooling

- Max-pooling partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum value.
- Reduces the computational complexity for upper layers.
- Provides a form of translation invariance.

The Full Model

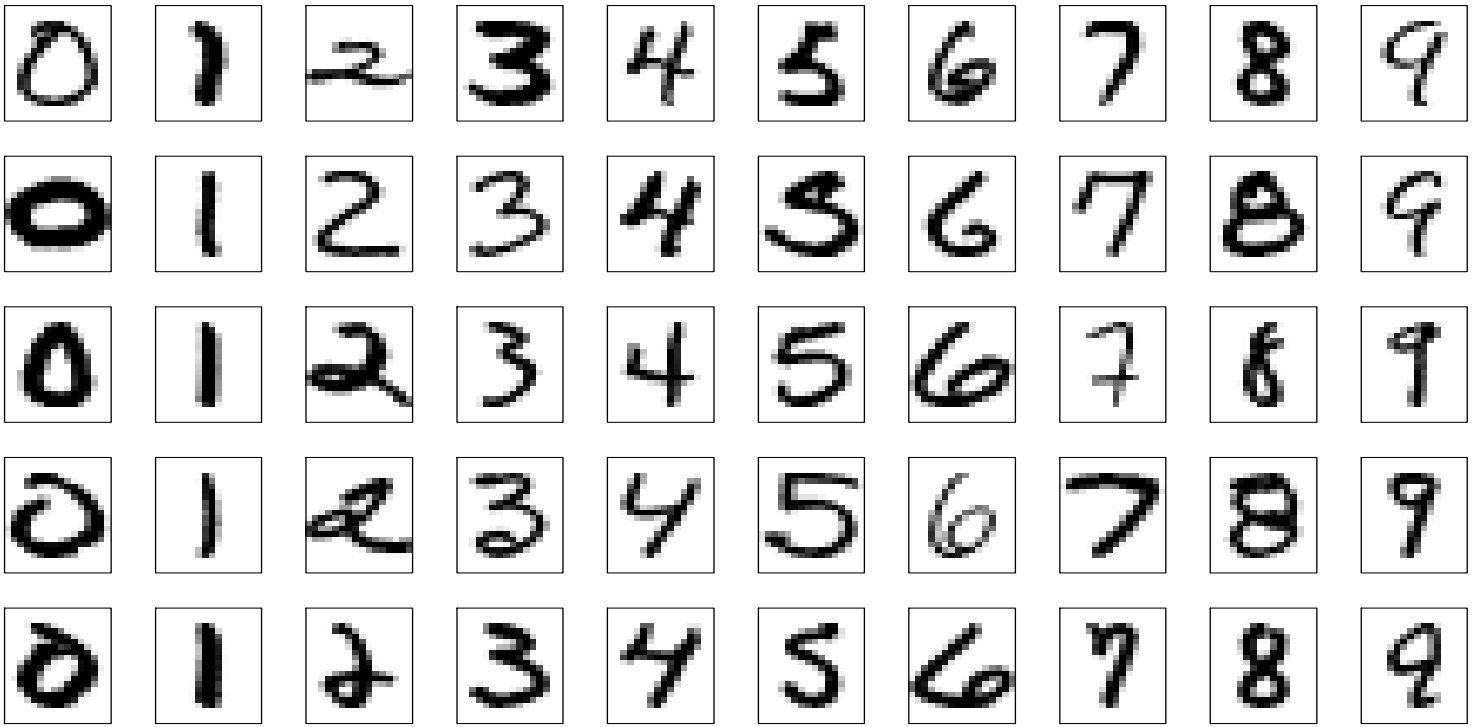
Sparse, convolutional layers and max-pooling are at the heart of the LeNet family of models.



Graphical depiction of a LeNet model.

Example: ZIP Code Data

The data



Each image is a 16×16 8-bit grayscale representation of a handwritten digit.

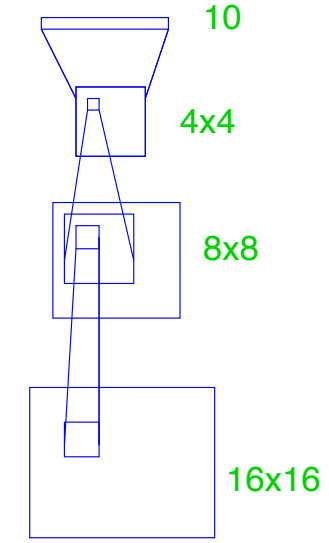
For the experiments in the book: $n_{\text{train}} = 320$ and $n_{\text{test}} = 160$.

Five networks fit to the data

- **Net-1**
  No hidden layer, equivalent to multinomial logistic regression.

- **Net-2**
  One hidden layer, 12 hidden units fully connected.

- **Net-3**
  Two hidden layers locally connected.
  - 1st hidden layer (8×8 array), each unit takes inputs from a 3×3 patch of the input layer after subsampling by 2.
  - 2nd hidden layer, inputs are from a 5×5 patch of the input layer after subsampling by 2.
  - Local connectivity makes each unit responsible for extracting local features from the layer below.



Net-3
Local Connectivity

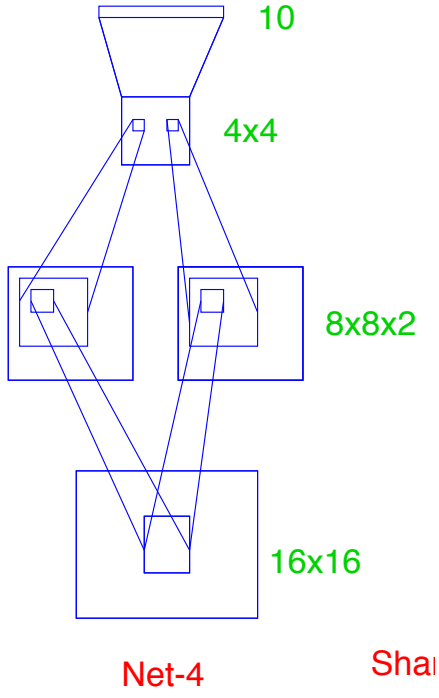
Five networks fit to the data

- **Net-4** (convolutional neural network)
  Two hidden layers, locally connected with weight sharing.

  - 1st hidden layer has two 8×8 arrays. Each unit takes inputs from a 3×3 patch of the input layer after subsampling by 2. The units in the feature map share the same set of nine weights (but have their own bias parameter).

  - 2nd hidden layer, inputs are from a $5 \times 5 \times 2$ volume of the two input layers after subsampling by 2. It has no weight sharing.

  - Local connectivity makes each unit responsible for extracting local features from the layer below.



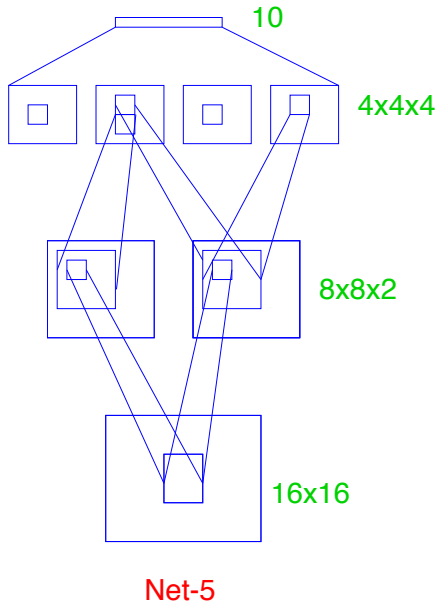
Five networks fit to the data

- **Net-5** (convolutional neural network)
  Two hidden layers, locally connected, two levels of weight sharing.

  - 1st hidden layer has two 8×8 arrays. Each unit takes inputs from a 3×3 patch of the input layer after subsampling by 2. The units in the feature map share the same set of nine weights (but have their own bias parameter).

  - 2nd hidden layer has four 4×4 feature maps. Inputs are from a $5 \times 5 \times 2$ volume of the two input layers after subsampling by 2. The units in the feature map share the same set of 50 weights (but have their own bias parameter).

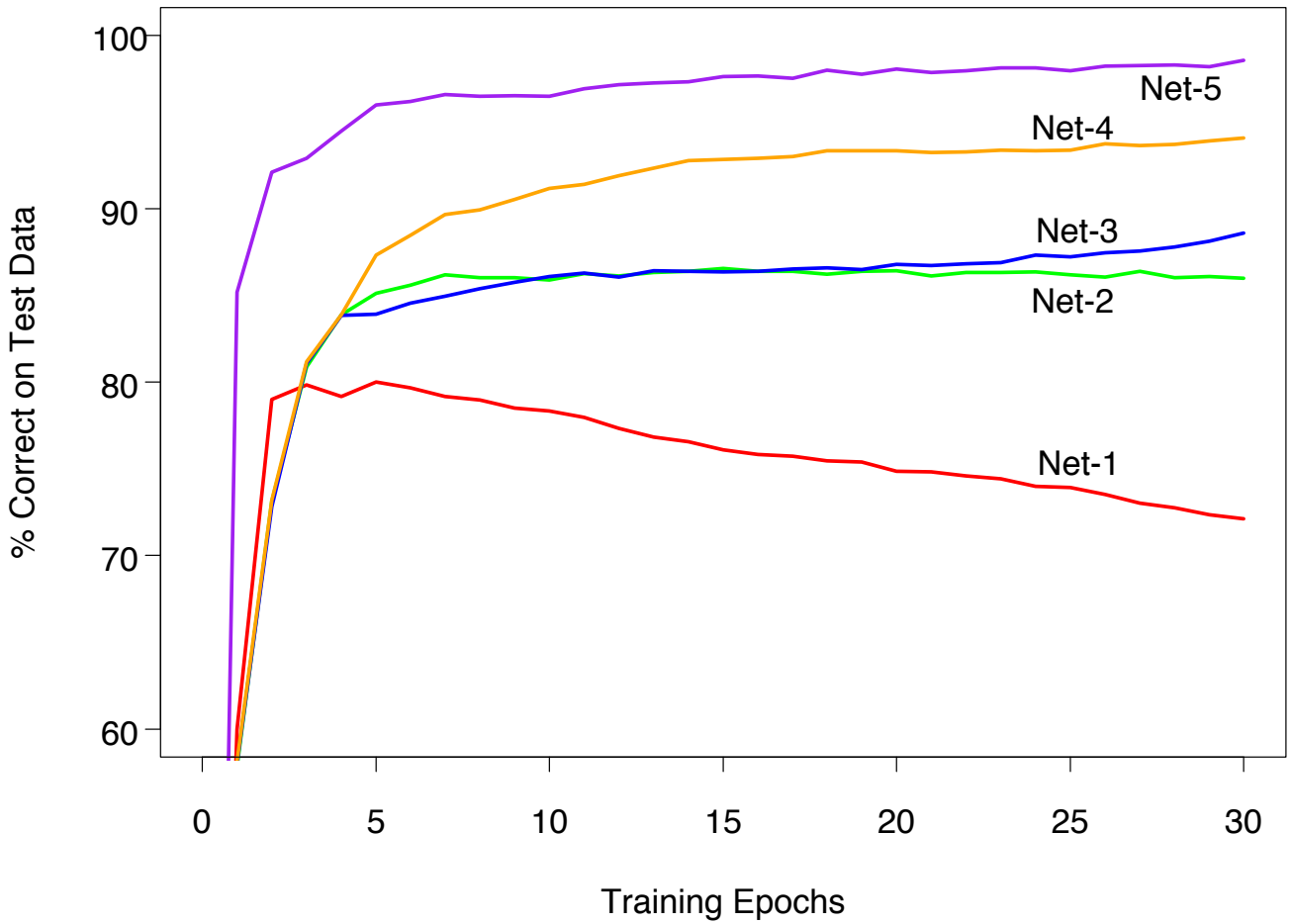
  - Local connectivity makes each unit responsible for extracting local features from the layer below.



Number of parameters

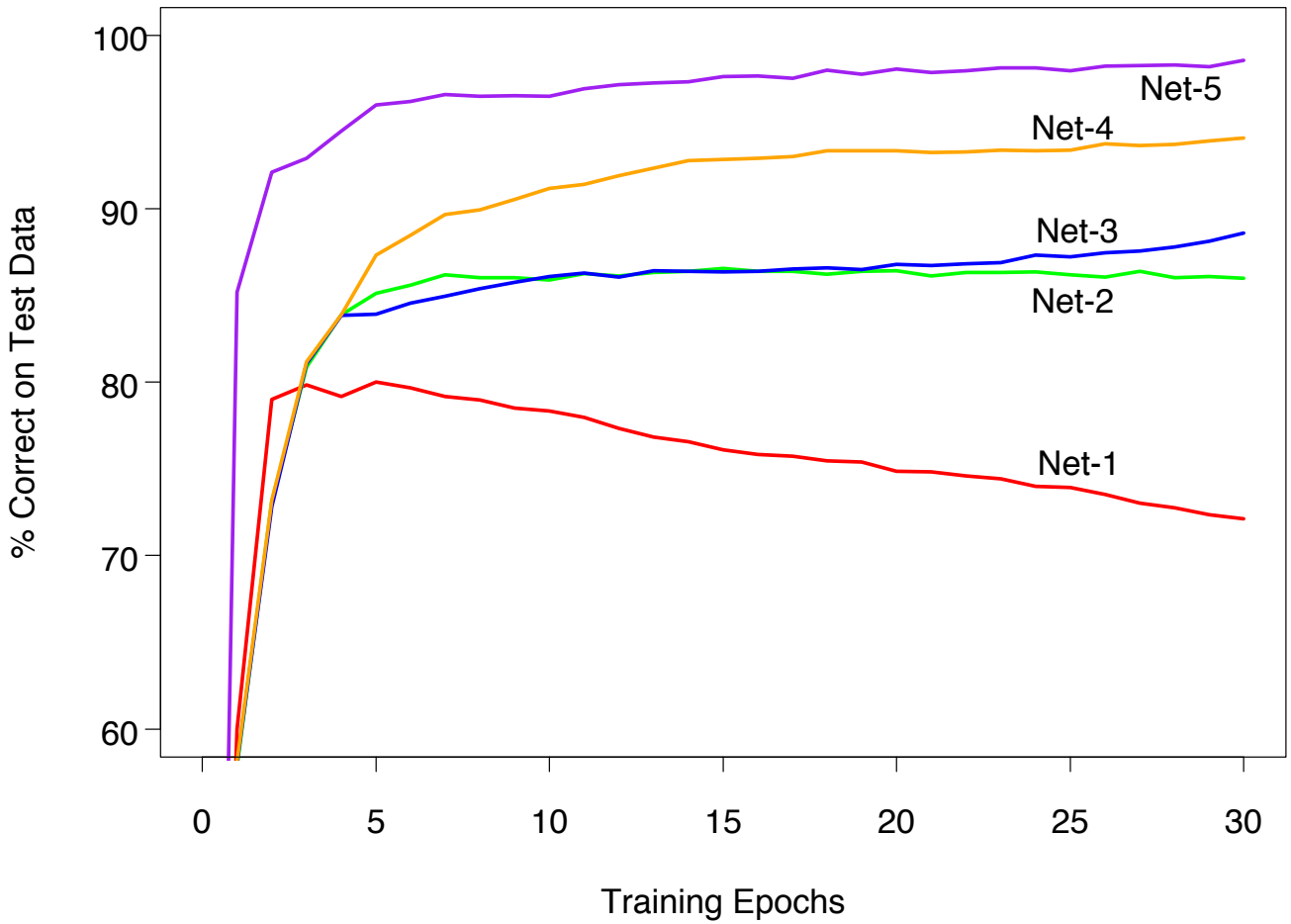
	Network Architecture	Links	Weights	% Correct
Net-1:	Single layer network	2570	2570	80.0%
Net-2:	Two layer network	3214	3214	87.0%
Net-3:	Locally connected	1226	1226	88.5%
Net-4:	Constrained network 1	2266	1132	94.0%
Net-5:	Constrained network 2	5194	1060	98.4%

Results



The networks all have sigmoidal output units, and were all fit with the sum-of-squares error function.

Results



The networks all have sigmoidal output units, and were all fit with the sum-of-squares error function.

Overview of results on full MNIST dataset

$n_{\text{train}} = 60,000$ and $n_{\text{test}} = 10,000$.

CLASSIFIER	PREPROCESSING	TEST ERROR RATE (%)	Reference
Linear Classifiers			
linear classifier (1-layer NN)	none	12.0	LeCun et al. 1998
linear classifier (1-layer NN)	deskewing	8.4	LeCun et al. 1998
pairwise linear classifier	deskewing	7.6	LeCun et al. 1998
K-Nearest Neighbors			
K-nearest-neighbors, Euclidean (L2)	none	5.0	LeCun et al. 1998
K-nearest-neighbors, Euclidean (L2)	none	3.09	Kenneth Wilder, U. Chicago
K-nearest-neighbors, L3	none	2.83	Kenneth Wilder, U. Chicago
K-nearest-neighbors, Euclidean (L2)	deskewing	2.4	LeCun et al. 1998
K-nearest-neighbors, Euclidean (L2)	deskewing, noise removal, blurring	1.80	Kenneth Wilder, U. Chicago
K-nearest-neighbors, L3	deskewing, noise removal, blurring	1.73	Kenneth Wilder, U. Chicago
K-nearest-neighbors, L3	deskewing, noise removal, blurring, 1 pixel shift	1.33	Kenneth Wilder, U. Chicago
K-nearest-neighbors, L3	deskewing, noise removal, blurring, 2 pixel shift	1.22	Kenneth Wilder, U. Chicago
K-NN with non-linear deformation (IDM)	shiftable edges	0.54	Keysers et al. IEEE PAMI 2007
K-NN with non-linear deformation (P2DHMDM)	shiftable edges	0.52	Keysers et al. IEEE PAMI 2007
K-NN, Tangent Distance	subsampling to 16x16 pixels	1.1	LeCun et al. 1998
K-NN, shape context matching	shape context feature extraction	0.63	Belongie et al. IEEE PAMI 2002

Overview of results on full MNIST dataset

$n_{\text{train}} = 60,000$ and $n_{\text{test}} = 10,000$.

Boosted Stumps			
boosted stumps	none	7.7	Kegl et al., ICML 2009
products of boosted stumps (3 terms)	none	1.26	Kegl et al., ICML 2009
boosted trees (17 leaves)	none	1.53	Kegl et al., ICML 2009
stumps on Haar features	Haar features	1.02	Kegl et al., ICML 2009
product of stumps on Haar f.	Haar features	0.87	Kegl et al., ICML 2009

Non-Linear Classifiers			
40 PCA + quadratic classifier	none	3.3	LeCun et al. 1998
1000 RBF + linear classifier	none	3.6	LeCun et al. 1998

SVMs			
SVM, Gaussian Kernel	none	1.4	
SVM deg 4 polynomial	deskewing	1.1	LeCun et al. 1998
Reduced Set SVM deg 5 polynomial	deskewing	1.0	LeCun et al. 1998
Virtual SVM deg-9 poly [distortions]	none	0.8	LeCun et al. 1998
Virtual SVM, deg-9 poly, 1-pixel jittered	none	0.68	DeCoste and Scholkopf, MLJ 2002
Virtual SVM, deg-9 poly, 1-pixel jittered	deskewing	0.68	DeCoste and Scholkopf, MLJ 2002
Virtual SVM, deg-9 poly, 2-pixel jittered	deskewing	0.56	DeCoste and Scholkopf, MLJ 2002

Overview of results on full MNIST dataset

$n_{\text{train}} = 60,000$ and $n_{\text{test}} = 10,000$.

Neural Nets			
2-layer NN, 300 hidden units, mean square error	none	4.7	LeCun et al. 1998
2-layer NN, 300 HU, MSE, [distortions]	none	3.6	LeCun et al. 1998
2-layer NN, 300 HU	deskewing	1.6	LeCun et al. 1998
2-layer NN, 1000 hidden units	none	4.5	LeCun et al. 1998
2-layer NN, 1000 HU, [distortions]	none	3.8	LeCun et al. 1998
3-layer NN, 300+100 hidden units	none	3.05	LeCun et al. 1998
3-layer NN, 300+100 HU [distortions]	none	2.5	LeCun et al. 1998
3-layer NN, 500+150 hidden units	none	2.95	LeCun et al. 1998
3-layer NN, 500+150 HU [distortions]	none	2.45	LeCun et al. 1998
3-layer NN, 500+300 HU, softmax, cross entropy, weight decay	none	1.53	Hinton, unpublished, 2005
2-layer NN, 800 HU, Cross-Entropy Loss	none	1.6	Simard et al., ICDAR 2003
2-layer NN, 800 HU, cross-entropy [affine distortions]	none	1.1	Simard et al., ICDAR 2003
2-layer NN, 800 HU, MSE [elastic distortions]	none	0.9	Simard et al., ICDAR 2003
2-layer NN, 800 HU, cross-entropy [elastic distortions]	none	0.7	Simard et al., ICDAR 2003
NN, 784-500-500-2000-30 + nearest neighbor, RBM + NCA training [no distortions]	none	1.0	Salakhutdinov and Hinton, AI-Stats 2007
6-layer NN 784-2500-2000-1500-1000-500-10 (on GPU) [elastic distortions]	none	0.35	Ciresan et al. Neural Computation 10, 2010 and arXiv 1003.0358, 2010
committee of 25 NN 784-800-10 [elastic distortions]	width normalization, deslanting	0.39	Meier et al. ICDAR 2011
deep convex net, unsup pre-training [no distortions]	none	0.83	Deng et al. Interspeech 2010

Overview of results on full MNIST dataset

$n_{\text{train}} = 60,000$ and $n_{\text{test}} = 10,000$.

Convolutional nets			
Convolutional net LeNet-1	subsampling to 16x16 pixels	1.7	LeCun et al. 1998
Convolutional net LeNet-4	none	1.1	LeCun et al. 1998
Convolutional net LeNet-4 with K-NN instead of last layer	none	1.1	LeCun et al. 1998
Convolutional net LeNet-4 with local learning instead of last layer	none	1.1	LeCun et al. 1998
Convolutional net LeNet-5, [no distortions]	none	0.95	LeCun et al. 1998
Convolutional net LeNet-5, [huge distortions]	none	0.85	LeCun et al. 1998
Convolutional net LeNet-5, [distortions]	none	0.8	LeCun et al. 1998
Convolutional net Boosted LeNet-4, [distortions]	none	0.7	LeCun et al. 1998
Trainable feature extractor + SVMs [no distortions]	none	0.83	Lauer et al., Pattern Recognition 40-6, 2007
Trainable feature extractor + SVMs [elastic distortions]	none	0.56	Lauer et al., Pattern Recognition 40-6, 2007
Trainable feature extractor + SVMs [affine distortions]	none	0.54	Lauer et al., Pattern Recognition 40-6, 2007
unsupervised sparse features + SVM, [no distortions]	none	0.59	Labusch et al., IEEE TNN 2008
Convolutional net, cross-entropy [affine distortions]	none	0.6	Simard et al., ICDAR 2003
Convolutional net, cross-entropy [elastic distortions]	none	0.4	Simard et al., ICDAR 2003
large conv. net, random features [no distortions]	none	0.89	Ranzato et al., CVPR 2007
large conv. net, unsup features [no distortions]	none	0.62	Ranzato et al., CVPR 2007
large conv. net, unsup pretraining [no distortions]	none	0.60	Ranzato et al., NIPS 2006
large conv. net, unsup pretraining [elastic distortions]	none	0.39	Ranzato et al., NIPS 2006
large conv. net, unsup pretraining [no distortions]	none	0.53	Jarrett et al., ICCV 2009
large/deep conv. net, 1-20-40-60-80-100-120-120-10 [elastic distortions]	none	0.35	Ciresan et al. IJCAI 2011
committee of 7 conv. net, 1-20-P-40-P-150-10 [elastic distortions]	width normalization	0.27 +-0.02	Ciresan et al. ICDAR 2011
committee of 35 conv. net, 1-20-P-40-P-150-10 [elastic distortions]	width normalization	0.23	Ciresan et al. CVPR 2012

Bayesian Neural Nets & the NIPS 2003 Challenge

NIPS 2003 Classification Challenge

Dataset	Domain	Feature Type	p	Percent Probes	N_{tr}	N_{val}	N_{te}
Arcene	Mass spectrometry	Dense	10,000	30	100	100	700
Dexter	Text classification	Sparse	20,000	50	300	300	2000
Dorothea	Drug discovery	Sparse	100,000	50	800	350	800
Gisette	Digit recognition	Dense	5000	30	6000	1000	6500
Madelon	Artificial	Dense	500	96	2000	600	1800

- Each dataset represents a two-class classification problems
- Emphasis on feature extraction
- Artificial "*probes*" (noise features) added to the data.

Winning method: Neal and Zhang (2006) used a series of

- preprocessing feature-selection steps,
- followed by Bayesian neural networks,
- Dirichlet diffusion trees, and
- combinations of these methods.

NIPS 2003 Classification Challenge

Dataset	Domain	Feature Type	p	Percent Probes	N_{tr}	N_{val}	N_{te}
Arcene	Mass spectrometry	Dense	10,000	30	100	100	700
Dexter	Text classification	Sparse	20,000	50	300	300	2000
Dorothea	Drug discovery	Sparse	100,000	50	800	350	800
Gisette	Digit recognition	Dense	5000	30	6000	1000	6500
Madelon	Artificial	Dense	500	96	2000	600	1800

- Each dataset represents a two-class classification problems
- Emphasis on feature extraction
- Artificial "*probes*" (noise features) added to the data.

Winning method: Neal and Zhang (2006) used a series of

- preprocessing feature-selection steps,
- followed by Bayesian neural networks,
- Dirichlet diffusion trees, and
- combinations of these methods.

Overview of Neal & Zang's approach

- Have training data $\mathbf{X}_{\text{tr}}, y_{\text{tr}}$
- Build a two-hidden layer network with parameters θ.
- Output nodes model $P(Y = -1 \mid X, \theta)$ and $P(Y = +1 \mid X, \theta)$.
- Given a prior distribution $p(\theta)$ then

$$p(\theta \mid \mathbf{X}_{\text{tr}}, y_{\text{tr}}) = \frac{p(\theta)\, P(y_{\text{tr}} \mid \mathbf{X}_{\text{tr}}, \theta)}{\int p(\theta)\, P(y_{\text{tr}} \mid \mathbf{X}_{\text{tr}}, \theta)\, d\theta}$$

- For a test case X_{new} the predictive distribution for Y_{new} is

$$P(Y_{\text{new}} \mid X_{\text{new}}, \mathbf{X}_{\text{tr}}, y_{\text{tr}}) = \int P(Y_{\text{new}} \mid X_{\text{new}}, \theta)\, p(\theta \mid \mathbf{X}_{\text{tr}}, y_{\text{tr}})\, d\theta$$

- MCMC methods are used to sample from $p(\theta \mid \mathbf{X}_{\text{tr}}, y_{\text{tr}})$ and hence to approximate the integral.

Overview of Neal & Zang's approach

Also tried different forms of pre-processing the features

- univariate screening using t-tests,
- automatic relevance determination

Potentially 3 main features important for its success:

- the feature selection and pre-processing,
- the neural network model, and
- the Bayesian inference for the model using MCMC.

Authors of book wanted to understand the reasons for the success of the Bayesian method.....

Authors' opinion

"power of modern Bayesian methods does not lie in their use as a formal inference procedure; most people would not believe that the priors in a high-dimensional, complex neural network model are actually correct. Rather the Bayesian/MCMC approach gives an efficient way of sampling the relevant parts of model space, and then averaging the predictions for the high-probability models."

Bagging & Boosting also average models

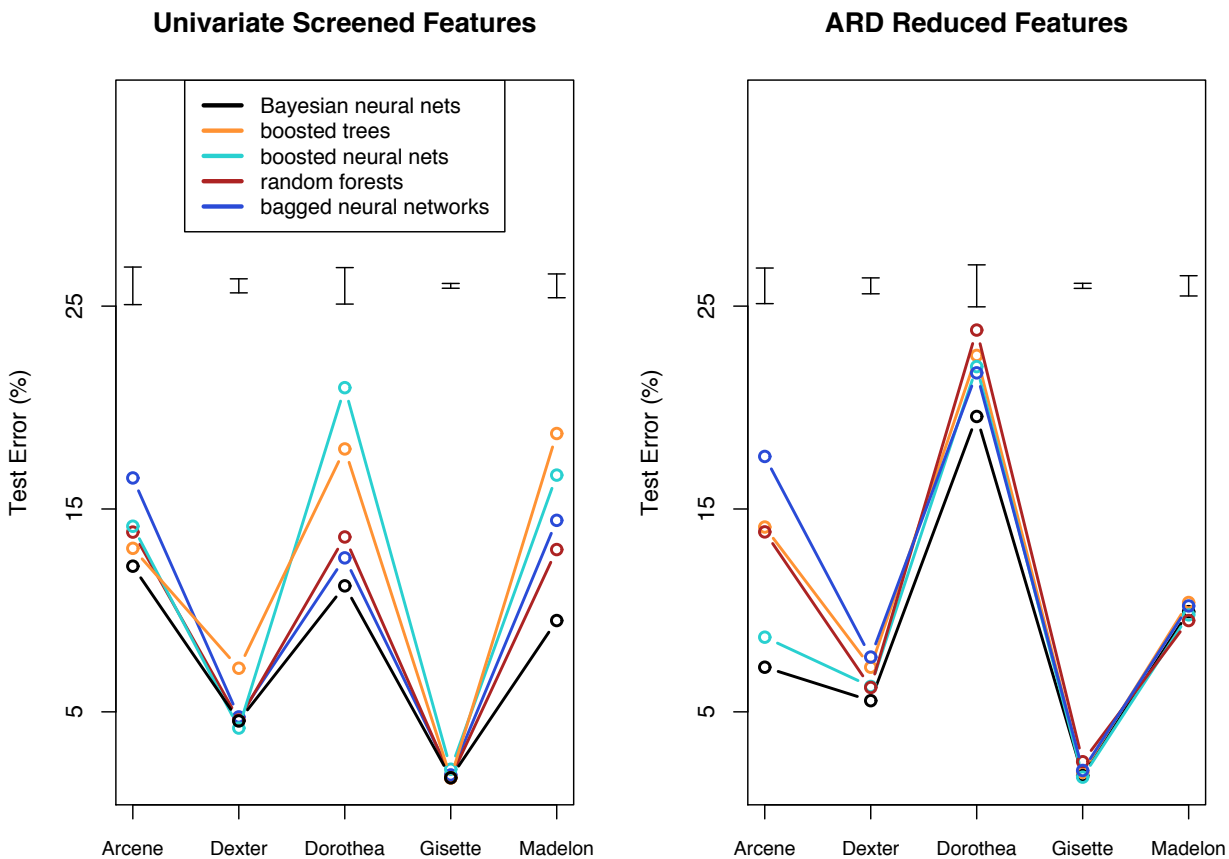
- **Bagging**
  - Perturbs the data in an i.i.d fashion and then
  - re-estimates the model to give a new set of model parameters.
  - Output is a simple average of the model predictions from different bagged samples
- **Boosting**
  - Fits a model that is additive in the models of each individual base learner
  - Base learners fit using non i.i.d. samples
- **Bayesian Approach**
  - Fixes the data and
  - perturbs the parameters according to current estimate of the poste- rior distribution

Methods compared

- **Bayesian Neural Nets** (2 hidden layers of 20 and 8 units)
- **Boosted trees**
- **Boosted Neural Nets**
- **Random forests**
- **Bagged Neural Networks**

Results

Dataset	Domain	Feature Type	p	Percent Probes	N_{tr}	N_{val}	N_{te}
Arcene	Mass spectrometry	Dense	10,000	30	100	100	700
Dexter	Text classification	Sparse	20,000	50	300	300	2000
Dorothea	Drug discovery	Sparse	100,000	50	800	350	800
Gisette	Digit recognition	Dense	5000	30	6000	1000	6500
Madelon	Artificial	Dense	500	96	2000	600	1800



Results

Dataset	Domain	Feature Type	p	Percent Probes	N_{tr}	N_{val}	N_{te}
Arcene	Mass spectrometry	Dense	10,000	30	100	100	700
Dexter	Text classification	Sparse	20,000	50	300	300	2000
Dorothea	Drug discovery	Sparse	100,000	50	800	350	800
Gisette	Digit recognition	Dense	5000	30	6000	1000	6500
Madelon	Artificial	Dense	500	96	2000	600	1800

Method	Screened Features		ARD Reduced Features	
	Average Rank	Average Time	Average Rank	Average Time
Bayesian neural networks	1.5	384(138)	1.6	600(186)
Boosted trees	3.4	3.03(2.5)	4.0	34.1(32.4)
Boosted neural networks	3.8	9.4(8.6)	2.2	35.6(33.5)
Random forests	2.7	1.9(1.7)	3.2	11.2(9.3)
Bagged neural networks	3.6	3.5(1.1)	4.0	6.4(4.4)

Why Bayesian Neural Networks works best?

Authors conjecture that reasons are perhaps

- the neural network model is well suited to these five problems
- the MCMC approach provides an efficient way of exploring the important part of the parameter space, and then averaging the resulting models according to their quality.