Chapter 11: Neural Networks

DD3364

December 16, 2012
Projection Pursuit Regression
Projection Pursuit Regression

Projection Pursuit Regression 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

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

- **Left graph**

- **Right graph**

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

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

to give

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

To minimize the rhs:

- Perform least squares regression \(\text{no intercept (bias) term}\).

- Input $w^t x_i$ has target $w_{old}^t x_i + \frac{y_i - g(w_{old}^t x_i)}{g'(w_{old}^t x_i)}$

- Weight errors with $g'(w_{old}^t x_i)^2$

- This produces the updated coefficient vector $w_{new}$.
Iterate these two steps until convergence

- **Fix** \( w \) and **update** \( g \)
- **Fix** \( g \) and **update** \( w \)
Neural Networks
FIGURE 11.2. Schematic of a single hidden layer, feed-forward neural network.

Thinking of the constant "1" as an additional input feature, this bias unit captures the intercepts $\alpha_0$ and $\beta_0$ in model (11.5).

The output function $g_k(T)$ allows a final transformation of the vector of outputs $T$. For regression we typically choose the identity function $g_k(T) = T_k$. Early work in -class classification also used the identity function, but this was later abandoned in favor of the softmax function $g_k(T) = \frac{e^{T_k}}{\sum_{\ell=1}^{K} e^{T_\ell}}$. (11.6)

This is of course exactly the transformation used in the multilogit model (Section 4.4), and produces positive estimates that sum to one. In Section 4.2 we discuss other problems with linear activation functions, in particular potentially severe masking effects.

The units in the middle of the network, computing the derived features $Z_m$, are called hidden units because the values $Z_m$ are not directly observed. In general there can be more than one hidden layer, as illustrated in the example at the end of this chapter. We can think of the $Z_m$ as a basis expansion of the original inputs $X$; the neural network is then a standard linear model, or linear multilogit model, using these transformations as inputs. There is, however, an important enhancement over the basis-expansion techniques discussed in Chapter 5; here the parameters of the basis functions are learned from the data.
• **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...
\textbf{\textit{K-classification}}

- Computation of the \(k\)th 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 \(\sigma\) can be defined

\[\sigma(v) = \frac{1}{1 + \exp\{-v\}} \quad \text{\textit{sigmoid function}}\]

and the output function \(g_k\)

\[g_k(T_1, \ldots, T_K) = \frac{\exp\{T_k\}}{\sum_{l=1}^{K} \exp\{T_l\}} \quad \text{\textit{softmax function}}\]
Shown is $\sigma(sv)$ for $s = .5, 1, 10$

- If $\sigma$ is the identity $\Rightarrow$ 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 $\alpha_m$ where $Z_m = \sigma(\alpha_{m0} + \sum_{i=1}^{p} \alpha_{mi}X_l)$

- When $\|\sigma_m\|$ small $\Rightarrow$ unit operates in the linear part of its activation function.
A NN with one hidden units, can approximate arbitrarily well any functional continuous mapping from one finite dimensional space to another, provided \textit{number of hidden units is sufficiently large}. 
Fitting Neural Networks
• This 2-layer neural network has unknown parameters, $\theta$,

\[
\{\alpha_{m0}, \alpha_{m1}, \ldots, \alpha_{mp}; m = 1, \ldots, M\} \quad M(p + 1) \text{ weights}
\]

\[
\{\beta_{k0}, \beta_{k1}, \ldots, \beta_{kM}; k = 1, \ldots, K\} \quad K(M + 1) \text{ weights}
\]

• **Aim**: Estimate parameters, $\theta$, 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

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

or

\[
R(\theta) = -\sum_{i=1}^{n} \sum_{k=1}^{K} y_{ik} \log f_k(x_i) \quad \leftarrow \text{cross-entropy error}
\]
• Typically don’t want

\[ \hat{\theta} = \arg \min_{\theta} R(\theta) \]

\[ \implies \] 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

\[
z_{im} = \sigma(\alpha_{m0} + \alpha_m^t x_i)
\]

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 \(\gamma_r\) is the learning rate.

- The quantities \(\delta_{ki}\) and \(s_{mi}\) are “errors” from the current model at the output and hidden layer units respective

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

- Remember the errors satisfy

\[
s_{mi} = \sigma'(\alpha_{m0} + \alpha_{m}^t x_i) \sum_{k=1}^{K} \delta_{ki} \ \beta_{km}
\]
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 $\delta_{ki}$ and then back-propagated with

$$s_{mi} = \sigma'(\alpha_{m0} + \alpha^t_m x_i) \sum_{k=1}^{K} \delta_{ki} \beta_{km}$$


to give the errors $s_{mi}$.

- Use both sets of errors to compute the gradients for the updates.
• 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.
  \[\rightarrow\] can train network with very large trained datasets.

• Training epoch \(\equiv\) one sweep through the entire training set.

• Learning rate: \(\gamma_r\)
  - Batch learning - usually taken to be constant and can be optimized by a line search.
  - Online learning - \(\gamma_r \rightarrow 0\) as \(r \rightarrow \infty\)

• Note: Back-prop is very slow.
Some Issues in 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...
• If weights are near zero
  \[ \Rightarrow \sigma(\cdot) \text{ is roughly linear} \]
  \[ \Rightarrow \text{neural network collapses into an approx linear model.} \]

• Usually start with random values close to zero.
  \[ \Rightarrow \text{model starts out linear and becomes non-linear as weights increase.} \]

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

• Starting with large weights often leads to poor solutions.
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
    $\Rightarrow$ initially have a highly regularized linear solution
    $\Rightarrow$ 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 $\lambda$ tend to shrink weights towards zero.
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
    $\Rightarrow$ initially have a highly regularized linear solution
    $\Rightarrow$ 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 $\lambda$ tend to shrink weights towards zero.
11.5 Some Issues in Training Neural Networks

Both use softmax $g_k$ and cross-entropy error.

Bayes optimal decision boundary is the purple curve.
Weights learnt

No weight decay

Weight decay

- Both use softmax $g_k$ and cross-entropy error.
- The display ranges from bright green (negative) to bright red (positive).
• **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 \(\Rightarrow\) 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.
Combating Overfitting

- **Multiple Minima**
  
  - $R(\theta)$ non-convex $\implies$ 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

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

and \( \sigma^2 \) 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

**Figure 11.7.** Boxplots of test error, for simulated data example, relative to Bayes error. True function is a sum of two sigmoids. The test error is displayed for ten different starting weights, for a single hidden layer neural network with the number units as indicated. The two panels represent no weight decay (left) and strong weight decay $\lambda = 0.1$ (right).

**Figure 11.8.** Boxplots of test error, for simulated data example. True function is a sum of two sigmoids. The test error is displayed for ten different starting weights, for a single hidden layer neural network with ten hidden units and weight decay parameter value as indicated.
Example 1: Fixed number of hidden units, vary $\lambda$

FIGURE 11.7. Boxplots of test error, for simulated data example, relative to the Bayes error. True function is a sum of two sigmoids. The test error is displayed for ten different starting weights, for a single hidden layer neural network with the number units as indicated. The two panels represent no weight decay (left) and strong weight decay $\lambda = 0.1$ (right).

FIGURE 11.8. Boxplots of test error, for simulated data example. True function is a sum of two sigmoids. The test error is displayed for ten different starting weights, for a single hidden layer neural network with ten hidden units and weight decay parameter value as indicated.
• Generated data from this additive model

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

where

\[ X = (X_1, \ldots, X_{10})^t \quad \text{with} \quad X_i \sim \mathcal{N}(0, 1) \quad \text{for} \quad i = 1, 2 \]

\[ \phi(v) = \exp\{-v^2/2\}/\sqrt{2\pi}, \]

\[ \epsilon \sim \mathcal{N}(0, \sigma^2) \]

and \( \sigma^2 \) 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
• 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
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
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):

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

FIGURE 11.9. Examples of training cases from ZIP code data. Each image is a $16 \times 16$ 8-bit grayscale representation of a handwritten digit.

Each image is a $16 \times 16$ 8-bit grayscale representation of a handwritten digit.

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

The figures and tables in this example were recreated from Le Cun (1989).
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 ($8 \times 8$ array), each unit takes inputs from a $3 \times 3$ patch of the input layer after subsampling by 2.
  - 2nd hidden layer, inputs are from a $5 \times 5$ patch of the input layer after subsampling by 2.
  - Local connectivity makes each unit responsible for extracting local features from the layer below.

**FIGURE 11.10.** Architecture of the five networks used in the ZIP code example.

The examples were obtained by scanning some actual hand-drawn digits, and then generating additional images by random horizontal shifts. Details may be found in Le Cun (1989). There are 320 digits in the training set, and 160 in the test set.
Five networks fit to the data

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

  - 1st hidden layer has two $8 \times 8$ arrays. Each unit takes inputs from a $3 \times 3$ 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 $8 \times 8$ arrays. Each unit takes inputs from a $3 \times 3$ 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 $4 \times 4$ 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.
Table 11.1. Test set performance of five different neural networks on a handwritten digit classification example (Le Cun, 1989).

<table>
<thead>
<tr>
<th>Network Architecture</th>
<th>Links</th>
<th>Weights</th>
<th>% Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Net-1: Single layer network</td>
<td>2570</td>
<td>2570</td>
<td>80.0%</td>
</tr>
<tr>
<td>Net-2: Two layer network</td>
<td>3214</td>
<td>3214</td>
<td>87.0%</td>
</tr>
<tr>
<td>Net-3: Locally connected</td>
<td>1226</td>
<td>1226</td>
<td>88.5%</td>
</tr>
<tr>
<td>Net-4: Constrained network 1</td>
<td>2266</td>
<td>1132</td>
<td>94.0%</td>
</tr>
<tr>
<td>Net-5: Constrained network 2</td>
<td>5194</td>
<td>1060</td>
<td>98.4%</td>
</tr>
</tbody>
</table>

Net-4 and Net-5 have local connectivity with shared weights. All units in a local feature map perform the same operation on different parts of the image, achieved by sharing the same weights. The first hidden layer of Net-4 has two $8 \times 8$ arrays, and each unit takes input from a $3 \times 3$ patch just like in Net-3. However, each of the units in a single $8 \times 8$ feature map share the same set of nine weights (but have their own bias parameter). This forces the extracted features in different parts of the image to be computed by the same linear functional, and consequently these networks are sometimes known as convolutional networks. The second hidden layer of Net-4 has no weight sharing, and is the same as in Net-3. The gradient of the error function $R$ with respect to a shared weight is the sum of the gradients of $R$ with respect to each connection controlled by the weights in question.

Table 11.1 gives the number of links, the number of weights and the optimal test performance for each of the networks. We see that Net-4 has more links but fewer weights than Net-3, and superior test performance. Net-5 has four $4 \times 4$ feature maps in the second hidden layer, each unit connected to a $5 \times 5$ local patch in the layer below. Weights are shared in each of these feature maps. We see that Net-5 does the best, having errors of only 1.6%, compared to 13% for the “vanilla” network Net-2. The clever design of network Net-5, motivated by the fact that features of handwriting style should appear in more than one part of a digit, was the result of many person years of experimentation. This and similar networks gave better performance on ZIP code problems than any other learning method at that time (early 1990s). This example also shows that neural networks are not a fully automatic tool, as they are sometimes advertised. As with all statistical models, subject matter knowledge can and should be used to improve their performance.

This network was later outperformed by the tangent distance approach (Simard et al., 1993) described in Section 13.3.3, which explicitly incorporates natural affinities. At this point the digit recognition datasets become test beds for every new learning procedure, and researchers worked...
The networks all have sigmoidal output units, and were all fit with the sum-of-squares error function.
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

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

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

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

<table>
<thead>
<tr>
<th>Boosted Stumps</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>boosted stumps</td>
<td>none</td>
<td>7.7</td>
</tr>
<tr>
<td>products of boosted stumps (3 terms)</td>
<td>none</td>
<td>1.26</td>
</tr>
<tr>
<td>boosted trees (17 leaves)</td>
<td>none</td>
<td>1.53</td>
</tr>
<tr>
<td>stumps on Haar features</td>
<td>Haar features</td>
<td>1.02</td>
</tr>
<tr>
<td>product of stumps on Haar f.</td>
<td>Haar features</td>
<td>0.87</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Non-Linear Classifiers</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>40 PCA + quadratic classifier</td>
<td>none</td>
<td>3.3</td>
</tr>
<tr>
<td>1000 RBF + linear classifier</td>
<td>none</td>
<td>3.6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SVMs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM, Gaussian Kernel</td>
<td>none</td>
<td>1.4</td>
</tr>
<tr>
<td>SVM deg 4 polynomial</td>
<td>deskewing</td>
<td>1.1</td>
</tr>
<tr>
<td>Reduced Set SVM deg 5 polynomial</td>
<td>deskewing</td>
<td>1.0</td>
</tr>
<tr>
<td>Virtual SVM deg-9 poly [distortions]</td>
<td>none</td>
<td>0.8</td>
</tr>
<tr>
<td>Virtual SVM, deg-9 poly, 1-pixel jittered</td>
<td>none</td>
<td>0.68</td>
</tr>
<tr>
<td>Virtual SVM, deg-9 poly, 1-pixel jittered</td>
<td>deskewing</td>
<td>0.68</td>
</tr>
<tr>
<td>Virtual SVM, deg-9 poly, 2-pixel jittered</td>
<td>deskewing</td>
<td>0.56</td>
</tr>
</tbody>
</table>
Overview of results on full MNIST dataset

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

<table>
<thead>
<tr>
<th>Neural Nets</th>
<th>Loss Function</th>
<th>Error Rate</th>
<th>Ref.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-layer NN, 300 hidden units, mean square error</td>
<td>none</td>
<td>4.7</td>
<td>LeCun et al., 1998</td>
</tr>
<tr>
<td>2-layer NN, 300 HU, MSE, [distortions]</td>
<td>none</td>
<td>3.6</td>
<td>LeCun et al., 1998</td>
</tr>
<tr>
<td>2-layer NN, 300 HU</td>
<td>deskewing</td>
<td>1.6</td>
<td>LeCun et al., 1998</td>
</tr>
<tr>
<td>2-layer NN, 1000 hidden units</td>
<td>none</td>
<td>4.5</td>
<td>LeCun et al., 1998</td>
</tr>
<tr>
<td>2-layer NN, 1000 HU, [distortions]</td>
<td>none</td>
<td>3.8</td>
<td>LeCun et al., 1998</td>
</tr>
<tr>
<td>3-layer NN, 300+100 hidden units</td>
<td>none</td>
<td>3.05</td>
<td>LeCun et al., 1998</td>
</tr>
<tr>
<td>3-layer NN, 300+100 HU [distortions]</td>
<td>none</td>
<td>2.5</td>
<td>LeCun et al., 1998</td>
</tr>
<tr>
<td>3-layer NN, 500+150 hidden units</td>
<td>none</td>
<td>2.95</td>
<td>LeCun et al., 1998</td>
</tr>
<tr>
<td>3-layer NN, 500+150 HU [distortions]</td>
<td>none</td>
<td>2.45</td>
<td>LeCun et al., 1998</td>
</tr>
<tr>
<td>3-layer NN, 500+300 HU, softmax, cross entropy, weight decay</td>
<td>none</td>
<td>1.53</td>
<td>Hinton, unpublished, 2005</td>
</tr>
<tr>
<td>2-layer NN, 800 HU, Cross-Entropy Loss</td>
<td>none</td>
<td>1.6</td>
<td>Simard et al., ICDAR 2003</td>
</tr>
<tr>
<td>2-layer NN, 800 HU, cross-entropy [affine distortions]</td>
<td>none</td>
<td>1.1</td>
<td>Simard et al., ICDAR 2003</td>
</tr>
<tr>
<td>2-layer NN, 800 HU, MSE [elastic distortions]</td>
<td>none</td>
<td>0.9</td>
<td>Simard et al., ICDAR 2003</td>
</tr>
<tr>
<td>2-layer NN, 800 HU, cross-entropy [elastic distortions]</td>
<td>none</td>
<td>0.7</td>
<td>Simard et al., ICDAR 2003</td>
</tr>
<tr>
<td>6-layer NN 784-2500-2000-1500-1000-500-10 (on GPU) [elastic distortions]</td>
<td>none</td>
<td>0.35</td>
<td>Creesan et al., Neural Computation 10, 2010 and arXiv 1003.0358, 2010</td>
</tr>
<tr>
<td>committee of 25 NN 784-800-10 [elastic distortions]</td>
<td>width normalization, deslanting</td>
<td>0.39</td>
<td>Meier et al., ICDAR 2011</td>
</tr>
<tr>
<td>deep convex net, unsup pre-training [no distortions]</td>
<td>none</td>
<td>0.83</td>
<td>Deng et al., Interspeech 2010</td>
</tr>
</tbody>
</table>
Overview of results on full MNIST dataset

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

<table>
<thead>
<tr>
<th>Convolutional nets</th>
<th></th>
<th>[ \text{LeCun et al., 1998} ]</th>
<th>[ \text{LeCun et al., 1998} ]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convolutional net LeNet-1</td>
<td>subsampling to 16x16 pixels</td>
<td>1.7</td>
<td></td>
</tr>
<tr>
<td>Convolutional net LeNet-4</td>
<td>none</td>
<td>1.1</td>
<td></td>
</tr>
<tr>
<td>Convolutional net LeNet-4 with K-NN instead of last layer</td>
<td>none</td>
<td>1.1</td>
<td>[ \text{LeCun et al., 1998} ]</td>
</tr>
<tr>
<td>Convolutional net LeNet-4 with local learning instead of last layer</td>
<td>none</td>
<td>1.1</td>
<td>[ \text{LeCun et al., 1998} ]</td>
</tr>
<tr>
<td>Convolutional net LeNet-5, [no distortions]</td>
<td>none</td>
<td>0.95</td>
<td>[ \text{LeCun et al., 1998} ]</td>
</tr>
<tr>
<td>Convolutional net LeNet-5, [huge distortions]</td>
<td>none</td>
<td>0.85</td>
<td>[ \text{LeCun et al., 1998} ]</td>
</tr>
<tr>
<td>Convolutional net LeNet-5, [distortions]</td>
<td>none</td>
<td>0.8</td>
<td>[ \text{LeCun et al., 1998} ]</td>
</tr>
<tr>
<td>Convolutional net Boosted LeNet-4, [distortions]</td>
<td>none</td>
<td>0.7</td>
<td>[ \text{LeCun et al., 1998} ]</td>
</tr>
<tr>
<td>Trainable feature extractor + SVMs [no distortions]</td>
<td>none</td>
<td>0.83</td>
<td>[ \text{Larner et al., Pattern Recognition 40-6, 2007} ]</td>
</tr>
<tr>
<td>Trainable feature extractor + SVMs [elastic distortions]</td>
<td>none</td>
<td>0.56</td>
<td>[ \text{Larner et al., Pattern Recognition 40-6, 2007} ]</td>
</tr>
<tr>
<td>Trainable feature extractor + SVMs [affine distortions]</td>
<td>none</td>
<td>0.54</td>
<td>[ \text{Larner et al., Pattern Recognition 40-6, 2007} ]</td>
</tr>
<tr>
<td>unsupervised sparse features + SVM, [no distortions]</td>
<td>none</td>
<td>0.59</td>
<td>[ \text{Lubusch et al., IEEE TNN 2008} ]</td>
</tr>
<tr>
<td>Convolutional net, cross-entropy [affine distortions]</td>
<td>none</td>
<td>0.6</td>
<td>[ \text{Simard et al., ICDAR 2003} ]</td>
</tr>
<tr>
<td>Convolutional net, cross-entropy [elastic distortions]</td>
<td>none</td>
<td>0.4</td>
<td>[ \text{Simard et al., ICDAR 2003} ]</td>
</tr>
<tr>
<td>large conv. net, random features [no distortions]</td>
<td>none</td>
<td>0.89</td>
<td>[ \text{Ranzato et al., CVPR 2007} ]</td>
</tr>
<tr>
<td>large conv. net, unsup features [no distortions]</td>
<td>none</td>
<td>0.62</td>
<td>[ \text{Ranzato et al., CVPR 2007} ]</td>
</tr>
<tr>
<td>large conv. net, unsup pretraining [no distortions]</td>
<td>none</td>
<td>0.60</td>
<td>[ \text{Ranzato et al., NIPS 2006} ]</td>
</tr>
<tr>
<td>large conv. net, unsup pretraining [elastic distortions]</td>
<td>none</td>
<td>0.39</td>
<td>[ \text{Ranzato et al., NIPS 2006} ]</td>
</tr>
<tr>
<td>large conv. net, unsup pretraining [affine distortions]</td>
<td>none</td>
<td>0.53</td>
<td>[ \text{Jarrett et al., ICCV 2009} ]</td>
</tr>
<tr>
<td>large/deep conv. net, 1-20-40-60-80-100-120-120-10 [elastic distortions]</td>
<td>none</td>
<td>0.35</td>
<td>[ \text{Ciresan et al., IJCAI 2011} ]</td>
</tr>
<tr>
<td>committee of 7 conv. net, 1-20-P-40-P-150-10 [elastic distortions]</td>
<td>width normalization</td>
<td>0.27 + 0.02</td>
<td>[ \text{Ciresan et al., ICDAR 2011} ]</td>
</tr>
<tr>
<td>committee of 35 conv. net, 1-20-P-40-P-150-10 [elastic distortions]</td>
<td>width normalization</td>
<td>0.23</td>
<td>[ \text{Ciresan et al., CVPR 2012} ]</td>
</tr>
</tbody>
</table>
Bayesian Neural Nets & the NIPS 2003 Challenge
Each dataset represents a two-class classification problem

- 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.
TABLE 11.2. NIPS 2003 challenge data sets. The column labeled \( p \) is the number of features. For the Dorothea dataset the features are binary. \( N_{tr} \), \( N_{val} \) and \( N_{te} \) are the number of training, validation and test cases, respectively.

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

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

$$p(\theta \mid X_{tr}, y_{tr}) = \frac{p(\theta) P(y_{tr} \mid X_{tr}, \theta)}{\int p(\theta) P(y_{tr} \mid X_{tr}, \theta) d\theta}$$

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

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

- MCMC methods are used to sample from $p(\theta \mid X_{tr}, y_{tr})$ 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.....
“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 posterior distribution
Methods compared

- **Bayesian Neural Nets** (2 hidden layers of 20 and 8 units)
- Boosted trees
- Boosted Neural Nets
- Random forests
- Bagged Neural Networks
The distribution for the parameters is

Given a prior distribution \( \Pr(\theta) \)

Markov Chain Monte Carlo (MCMC) methods are used to sample from the

Since the integral in (11.20) is intractable, sophis

cation to neural networks. Given training data

Let us first review briefly the Bayesian approach to inference and its appli-

11.9.1 Bayes, Boosting and Bagging

Aspects of their approach were important for its success. We rerun their

only on the Bayesian neural network approach, and try to discern which

performs the best among the competitors using the selected feature set,

while the boosted neural networks perform best with the reduced feature

error of the different learning methods on five problems,

The superiority of boosted neural networks over boosted trees suggest


\[ \Pr(Y_{\text{new}} | X_{\text{tr}}, \theta) \Pr(\theta) \]

the predictive distribution for the

\[ \Pr(Y_{\text{new}} | X_{\text{tr}}, \theta) \]

\[ \Pr(Y_{\text{tr}} | X_{\text{tr}}, \theta) \]

\[ \int \Pr(Y_{\text{tr}} | X_{\text{tr}}, \theta) \Pr(\theta) \, d\theta \]

\[ \frac{1}{N} \sum_{i=1}^{N} \Pr(Y_{\text{new}} | X_{\text{tr}}, \theta_{i}) \]

\[ \frac{1}{N} \sum_{i=1}^{N} \Pr(Y_{\text{tr}} | X_{\text{tr}}, \theta_{i}) \]

\[ \theta_{i} \sim \text{Gaussian prior} \]

\[ \text{Hybrid Monte Carlo} \]

\[ \text{Neal and Zhang (2006)} \]

\[ \text{ARD Reduced Features} \]

\[ \text{Univariate Screened Features} \]

\[ \text{Bayesian neural nets} \]

\[ \text{boosted trees} \]

\[ \text{boosted neural nets} \]

\[ \text{random forests} \]

\[ \text{bagged neural networks} \]

\[ \text{Table 11.2.} \]

\[ \text{Dataset} \quad \text{Domain} \quad \text{Feature Type} \quad p \quad \text{Percent Probes} \quad N_{tr} \quad N_{val} \quad N_{te} \]

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

\[ \text{FIGURE 11.12.} \]

Performance of different learning methods on five problems,

The error bars at the top of each plot indicate one standard

\[ \text{Results} \]

\[ \text{Table 11.3.} \]

\[ \text{Dataset} \quad \text{Domain} \quad \text{Feature Probes} \quad N_{tr} \quad N_{val} \quad N_{te} \]

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Domain</th>
<th>Feature Probes</th>
<th>N_{tr}</th>
<th>N_{val}</th>
<th>N_{te}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arcene</td>
<td>Mass spectrometry</td>
<td>5 15 25</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dexter</td>
<td>Text classification</td>
<td>30</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dorothea</td>
<td>Drug discovery</td>
<td>50</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gisette</td>
<td>Digit recognition</td>
<td>50</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Madelon</td>
<td>Artificial</td>
<td>96</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ \text{We also thank Isabelle Guyon for help in preparing the result so ft h i ss e c t i o n .} \]
Let us first review briefly the Bayesian approach to inference and its application. The Bayesian approach works well for smoothly parametrized models like neural networks; it is not yet clear that it works as well for non-smooth models. For a test case with features \( x \), the predictive distribution for the new observation \( y \mid x, \theta \) is

\[
\text{Pr}(y \mid x, \theta) = \int \text{Pr}(y \mid x, \theta, \phi) \text{Pr}(\phi \mid \theta) d\phi.
\]

(a) The neural network model is well suited to these five problems, and may be important for the success of the method. It includes important part of the parameter space, and then averaging the resulting posterior distribution over the parameters (c.f. equation 8.24).

(b) The MCMC approach provides an efficient way of exploring the important region of the posterior distribution. Here the non-Bayesian methods show a clear advantage.

Overall, the superior performance of Bayesian neural networks here may be due to the fact that the reduced feature sets come from the Bayesian neural network, with output nodes the class probabilities \( \text{Pr}(Y = 1 | X) \). For internal feature selection might help the overall performance of boosted trees, and other related methods.

For the binary outcomes. Given a prior distribution \( \text{Pr}(\theta) \), the posterior distribution \( \text{Pr}(\theta \mid y) \) is

\[
\text{Pr}(\theta \mid y) \propto \text{Pr}(y \mid \theta) \text{Pr}(\theta).
\]

For the parameters, the posterior distribution is

\[
\text{Pr}(\theta) \propto \text{Pr}(y \mid \theta) \text{Pr}(\theta).
\]

With an auxiliary momentum vector and implements Hamiltonian dynamics in which the potential function is the target density. This is done to avoid a fixed point of the dynamics. Neal and Zhang (2006) use different methods. Values are average rank of test cases.

Results

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Screened Features</th>
<th>ARD Reduced Features</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Average Rank</td>
<td>Average Time</td>
</tr>
<tr>
<td>Bayesian neural networks</td>
<td>1.5</td>
<td>384(138)</td>
</tr>
<tr>
<td>Boosted trees</td>
<td>3.4</td>
<td>3.03(2.5)</td>
</tr>
<tr>
<td>Boosted neural networks</td>
<td>3.8</td>
<td>9.4(8.6)</td>
</tr>
<tr>
<td>Random forests</td>
<td>2.7</td>
<td>1.9(1.7)</td>
</tr>
<tr>
<td>Bagged neural networks</td>
<td>3.6</td>
<td>3.5(1.1)</td>
</tr>
</tbody>
</table>
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.