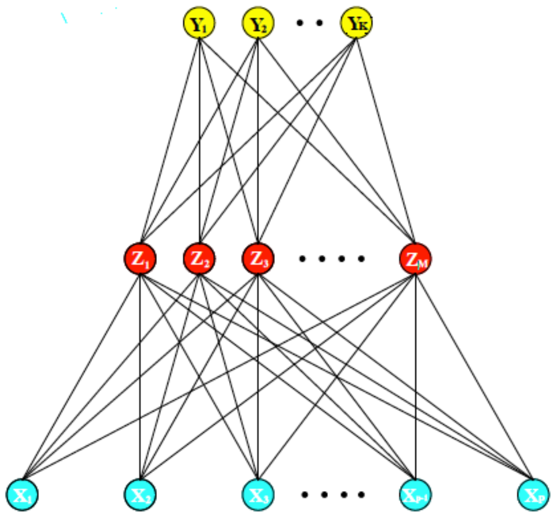
Neural Networks



Statistical Data Mining I

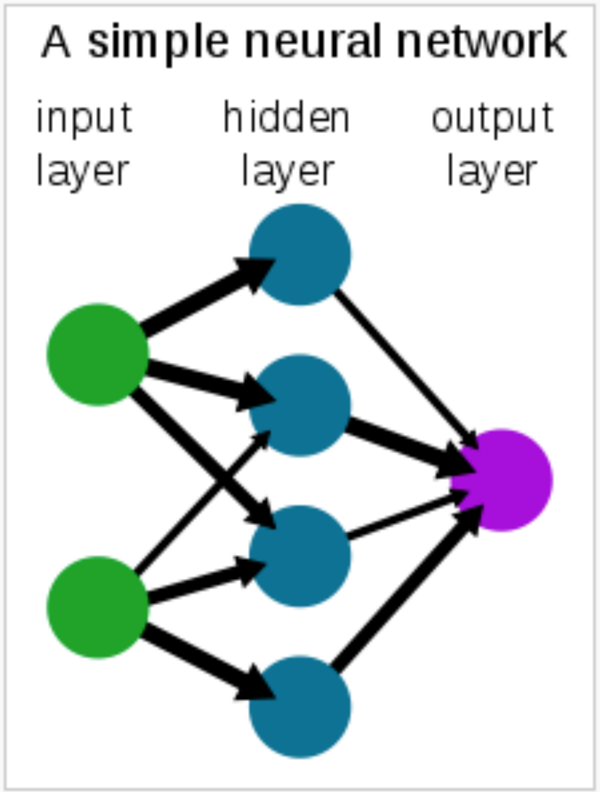
Rachael Hageman Blair

Central Idea

Extract linear combinations of inputs as derived features, and then model the target as a nonlinear function of these features.

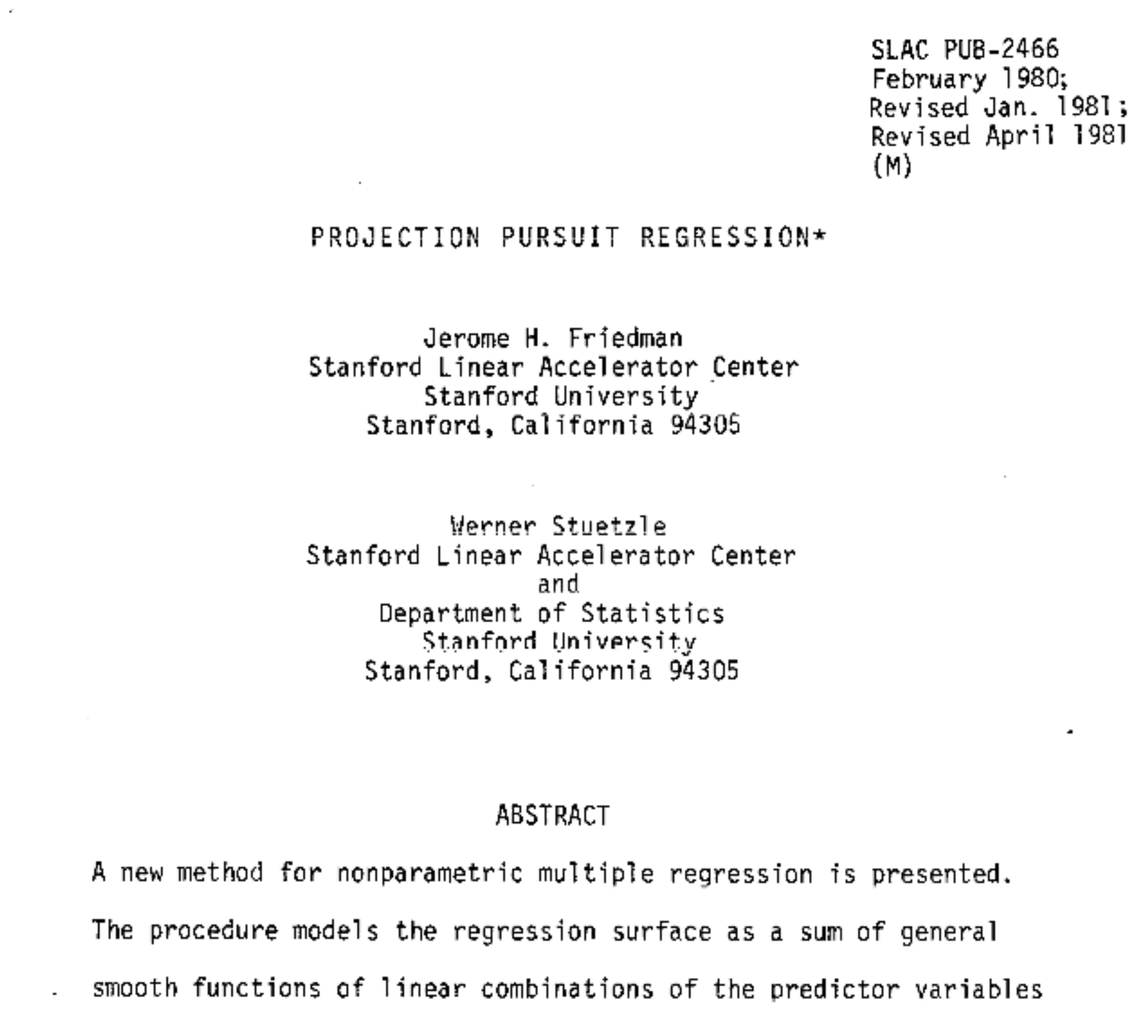
Outline

- Projection Pursuit Regression (PPR)
- Model Fitting
- DAG motivation
- Neural Networks
- Fitting Neural Networks
- Model fitting obstacles (PGM as well)
- Simulation Example
- Conclusions



Simplified view of a feedforward artificial neural network

Projection Pursuit Regression



SLAC PUB-2466
February 1980;
Revised Jan. 1981;
Revised April 1981
(M)

PROJECTION PURSUIT REGRESSION*

Jerome H. Friedman
Stanford Linear Accelerator Center
Stanford University
Stanford, California 94305

Werner Stuetzle
Stanford Linear Accelerator Center
and
Department of Statistics
Stanford University
Stanford, California 94305

ABSTRACT

A new method for nonparametric multiple regression is presented. The procedure models the regression surface as a sum of general smooth functions of linear combinations of the predictor variables

Projection Pursuit Regression

- Model the regression surface as the sum of general smooth functions of a linear combination of the predictive variables in an iterative manner.

More general than standard stepwise and stage-wise regression, does not require the definition of a metric in the predictor space.

Projection Pursuit Regression

- Let *X* denote the input with *p* components, with response (target) *Y*.

- The projection pursuit regression model (PPR) has the form:

$$f(X) = \sum_{m=1}^{M} g_m \underline{\left(\omega_m^T X\right)},$$

where $\omega_m,\ m = 1,2,\ldots,M$, be unit p- vectors of unknown parameters.

An additive model in the derived features

Functions are not specified, but estimated
Along with the directions ω_m.

Projection Pursuit Regression

$$f(X) = \sum_{m=1}^{M} \underline{g_m\left(\omega_m^T X\right)},$$

Called the "ridge function" in

$g_m\left(\omega_m^T X\right) \in \Re^p$

Which varies only in the direction ω_m

The scalar variable $V_m = \omega_m^T X$ is the projection of X onto the unit vector ω_m.

We are in "pursuit" of a ω_m which fits the model well.

Projection Pursuit Regression

- PPR model is very general, can use a large class of models, and is fairly flexible.
- M can be taken arbitrarily large, giving rise to a “universal approximator”. As before, there is a price for complexity.
- PPR model is most useful for prediction purposes. There is less of an interpretive value.

Projection Pursuit Regression

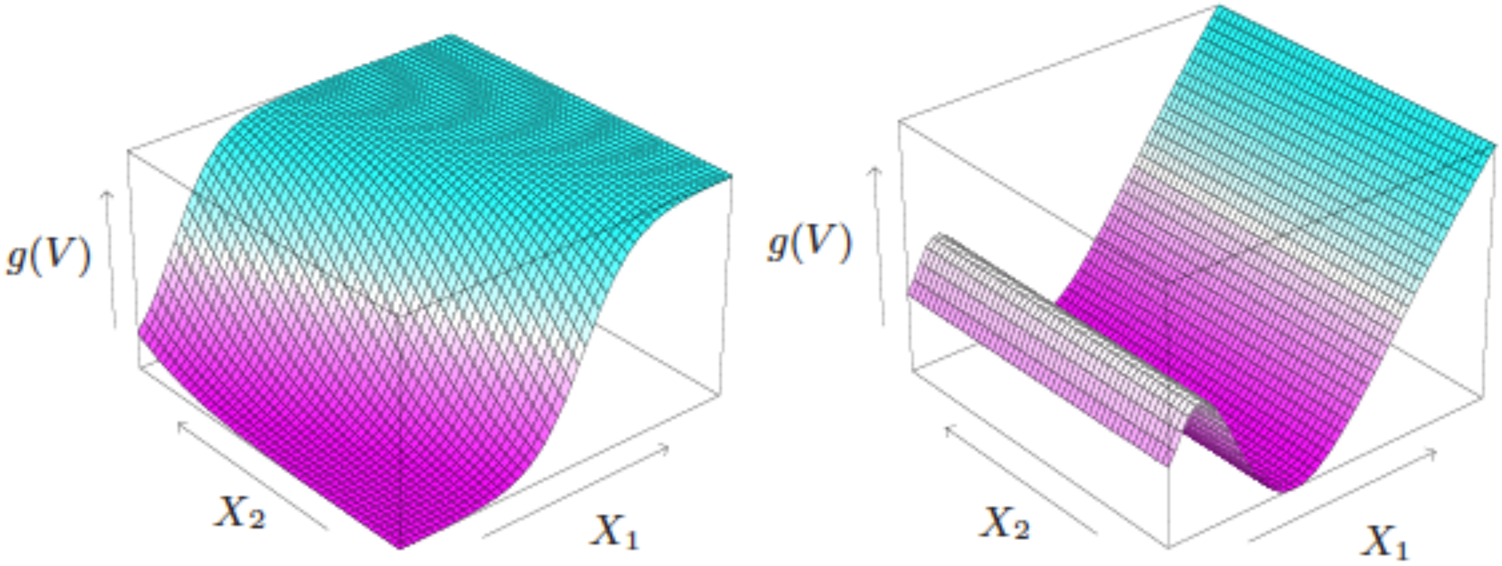


FIGURE 11.1. *Perspective plots of two ridge functions.*
(Left:) $g(V) = 1/[1 + \exp(-5(V - 0.5))]$, *where* $V = (X_1 + X_2)/\sqrt{2}$.
(Right:) $g(V) = (V + 0.1)\sin(1/(V/3 + 0.1))$, *where* $V = X_1$.

Projection Pursuit Regression

$\omega = (1/\sqrt{2})(1,1)^T$

The function varies only in the Direction X1+X2

$\omega = (1,0)^T$

The function varies only in the Direction X1

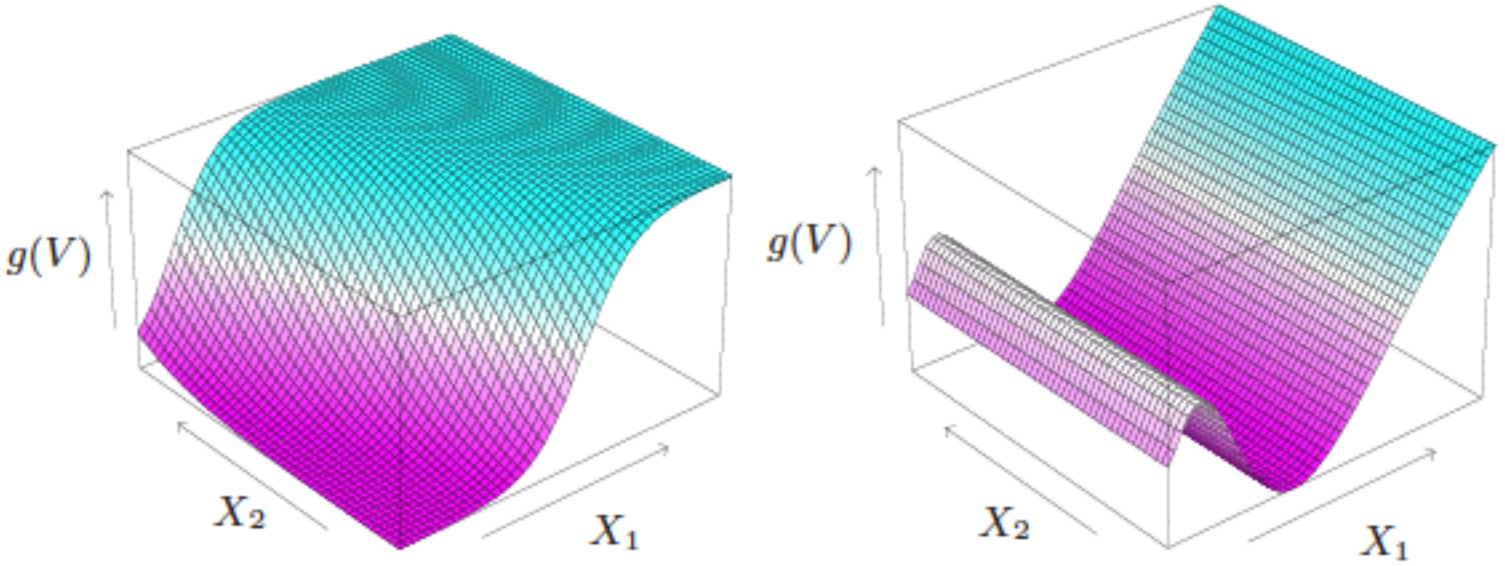


FIGURE 11.1. *Perspective plots of two ridge functions.*
(Left:) $g(V) = 1/[1+\exp(-5(V-0.5))]$, *where* $V = (X_1+X_2)/\sqrt{2}$.
(Right:) $g(V) = (V+0.1)\sin(1/(V/3+0.1))$, *where* $V = X_1$.

Model Fitting

- Suppose we have training data, $\{(x_1, y_1)\ldots(x_N, y_N)\}$.

  and we want to fit a PPR model.
- We have a "response" now, so we just have to minimize the loss:

$$\sum_{i=1}^{N}\left[y_i - \sum_{m=1}^{M} g_m\left(\omega_m^T x_i\right)\right]^2,$$

over the unknowns ω_m and g_m, for $m = 1,\ldots,M$.

Complexity constraints have to be imposed implicitly or explicitly on g_m, to avoid overfitting.

Model Fitting

<u>Strategy:</u> Gauss-Newton Method (NLS - Alternating)

A quasi-Newton method, in which the part of the Hessian involving the second derivative of g_m is discarded.

Model Fitting

- Estimation of g and ω are carried out until convergence .
- When M>1, the model is built in a stage-wise manner, adding pair (ω_m, g_m) at each stage.
- There are a number of implementation details regarding the model fitting, most importantly:

  - Although any smoothing method can be used, it is convenient to use local regression or smoothing splines.

  - M is typically estimated as a complexity parameter, stopping when adding to the model does not improve the fit substantially or enough to justify the complexity cost.

Motivation: A Neuron

Recall:

- Neural Networks, rooted in cognitive science.
- Connectivism:

  uses analogies of neurons and their connections together with the concept of neuron firing, activation functions, and their ability to modify these connections.

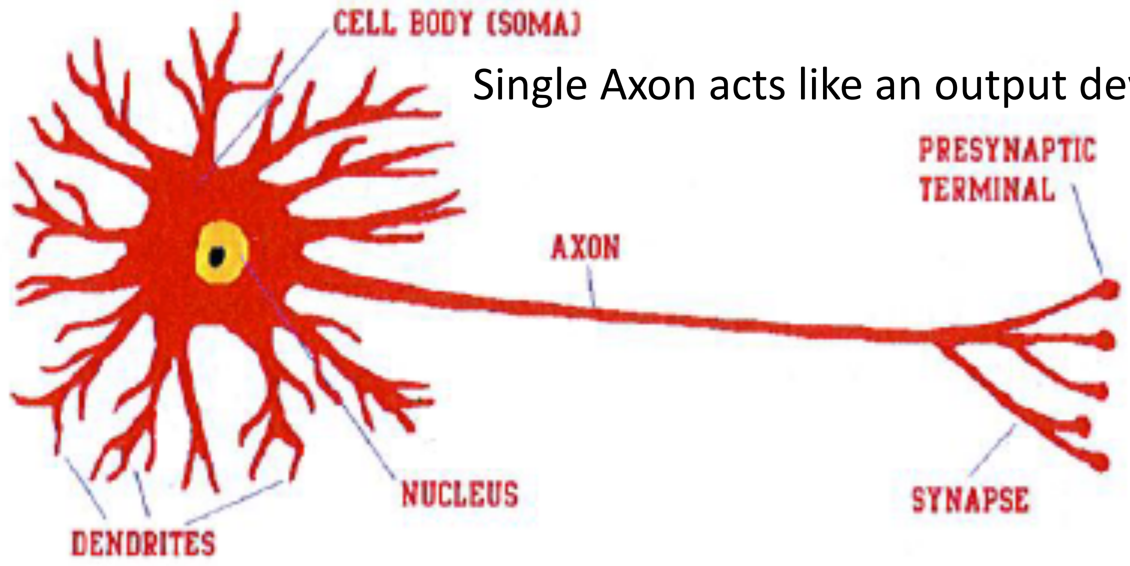
  Highlights the importance of "networks" in learning.

- Artificial Neural Networks were originally designed to mimic brain activity.

Motivation: A Neuron

Cell Body (Soma) of a Neuron has a nucleus
And two types of “processes” or “projections”
Dendrites and Axons.

Single Axon acts like an output device



Single Axon branches
Into strands, and each
Strand terminates in a
Synapse.
The synapse connect
To another neuron, or
Terminate in muscle.

FIGURE 10.1. *Schematic view of a biological neuron.*

Neuron receives its signals from other
Neurons through its dendrites, which
Act as an input device.

Neural Networks

- Neural Networks: describes what is now a large class of models and learning methods.

- A lot of “hype” around neural networks.

  Nothing more than a nonlinear statistical model similar in spirit to the PPR models.

- Two-stages regression or classification model represented by a layered network of “neurons”.

- **Focus**: Single hidden layer back-propagation network aka single layer perceptron (simplest).

Motivation: A Neuron

The McCulloch-Pitts Neuron (1943):

- Concept of “artificial neuron” introduced as a crude model/function.
- Artificial Neurons join together to form Artificial neural networks.
- Inputs (dendrites) and output (axon)

  The piece in the middle represents the “activation function”.

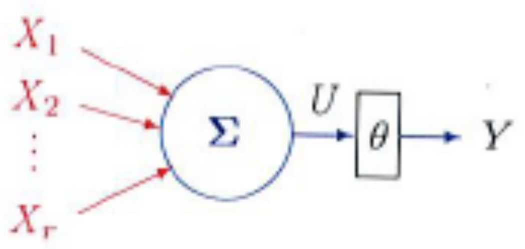


FIGURE 10.2. *McCulloch–Pitts neuron with r binary inputs, $X_1, X_2, \ldots, X_r$, one binary output, Y, and threshold θ.*

Motivation: A Neuron

A little too simple of a model:

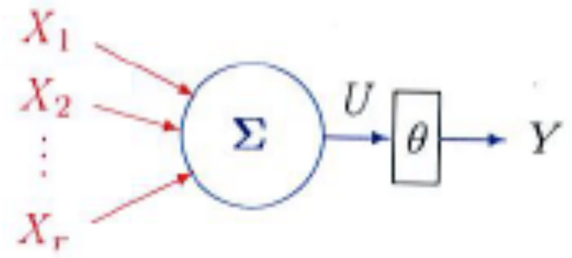


FIGURE 10.2. *McCulloch–Pitts neuron with r binary inputs, $X_1, X_2, \ldots, X_r$, one binary output, Y, and threshold θ.*

Not a good approximation of the system. There are no adjustable parameters or weights in the network, which means that different problems can only be solved by changing the input structure or the threshold values.

Motivation: A Neuron

Donald Hebb in 1949: learned a little bit more in his study of behavior.

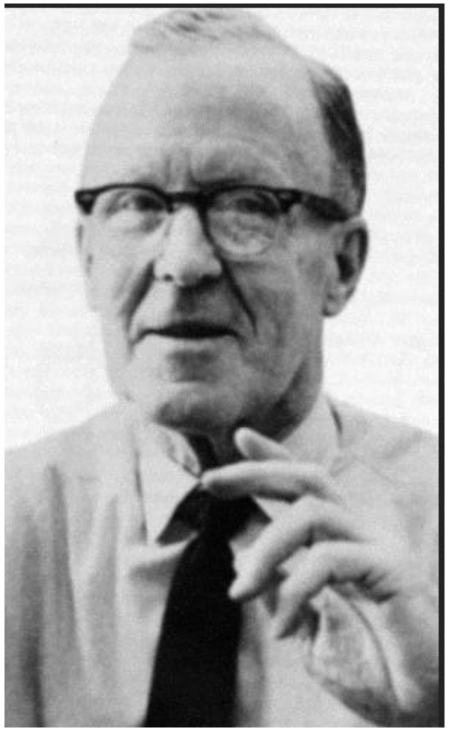
- Assumed all neurons we needed in life are present at birth, the initial neural connections are randomly distributed and as we get older the connections get stronger and increase.

- One’s perceptions, thoughts, emotions, memory, and sensations are strongly influenced by life experiences, which then in turn, leave behind a memory trace through “interconnected neurons” which influence future behavior.

Motivation: A Neuron

Hebb Learning Rule

When an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells so that A's efficiency, as one of the cells firing B, is increased.



The strength of synaptic connections between neurons, depends on their associated firing history!

Motivation: A Neuron

<u>Frank Rosenblatt (a psychologist)</u>

- Did not agree that most neural connections were at random and had other views on cell assemblies.
- Developed the perceptron.
- Exactly the “McCulloch-Pitts neuron, but now input comes with real-valued “connection weights”.

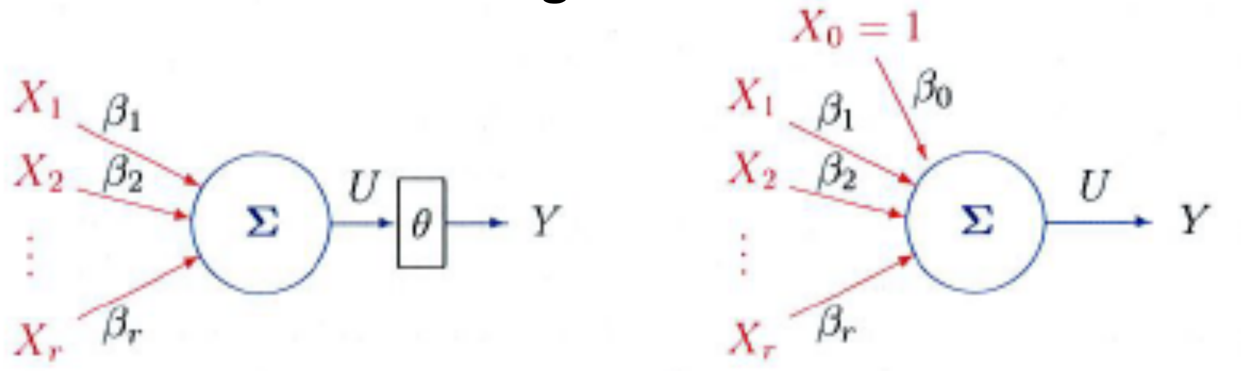


FIGURE 10.4. *Rosenblatt's single-layer perceptron with r inputs, connection weights $\{\beta_j\}$, and binary output Y. The left panel shows the perceptron with threshold θ, and the right panel shows the equivalent perceptron with bias element $\beta_0 = -\theta$ and $X_0 = 1$.*

Motivation: A Neuron

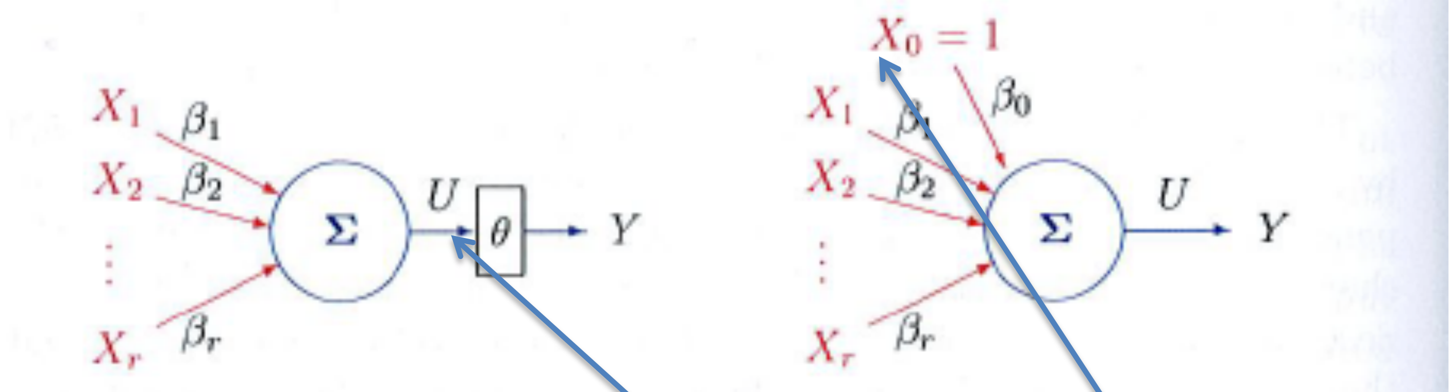


FIGURE 10.4. *Rosenblatt's single-layer perceptron with r inputs, connection weights $\{\beta_j\}$, and binary output Y. The left panel shows the perceptron with threshold θ, and the right panel shows the equivalent perceptron with bias element $\beta_0 = -\theta$ and $X_0 = 1$.*

Intercept term
Can shift threshold

- Positive Weights – excitatory synapses
- Negative Weights – inhibitory synapses
- Magnitude – strength of the connection

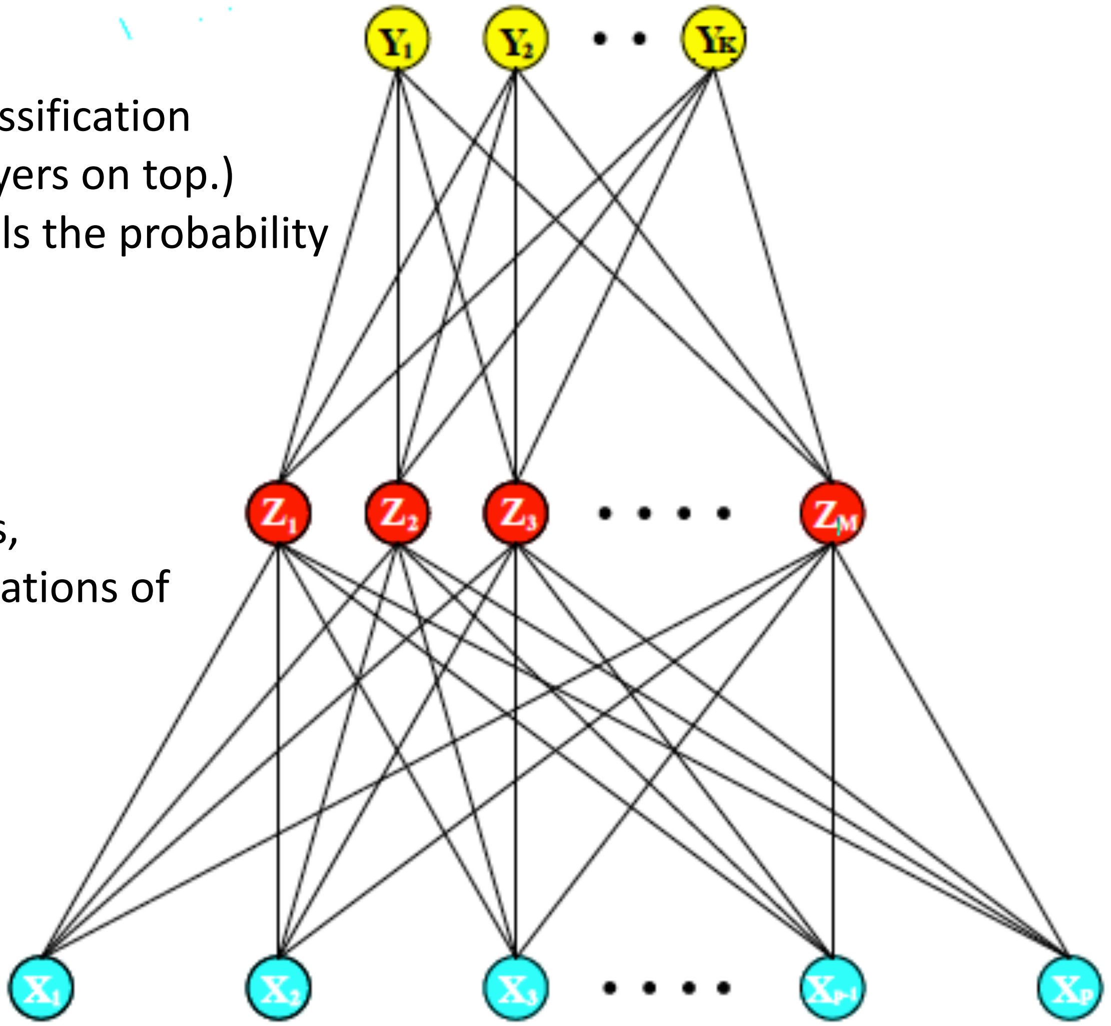
A weighted sum

Neural Networks

Target layer
(for K class classification
There are K layers on top.)
Kth unit models the probability
Of class K.

Hidden Layer
Derived inputs,
Linear combinations of
the inputs

Input layer



Neural Networks

Lets consider the K-class classification problem, and the multiple layers of the network.

- Derived features Z_m are created from linear combinations of the inputs.
- The target $Y_k, k = 1, 2, \ldots, K$, is modeled as a function of linear combinations of the Z_m ,

$$Z_m = \underline{\sigma}\left(\alpha_{0m} + \alpha_m^T X\right), \quad m = 1, \ldots, M,$$

Activation function

$$T_k = \beta_{0k} + \beta_k^T Z, \quad k = 1, \ldots, K,$$

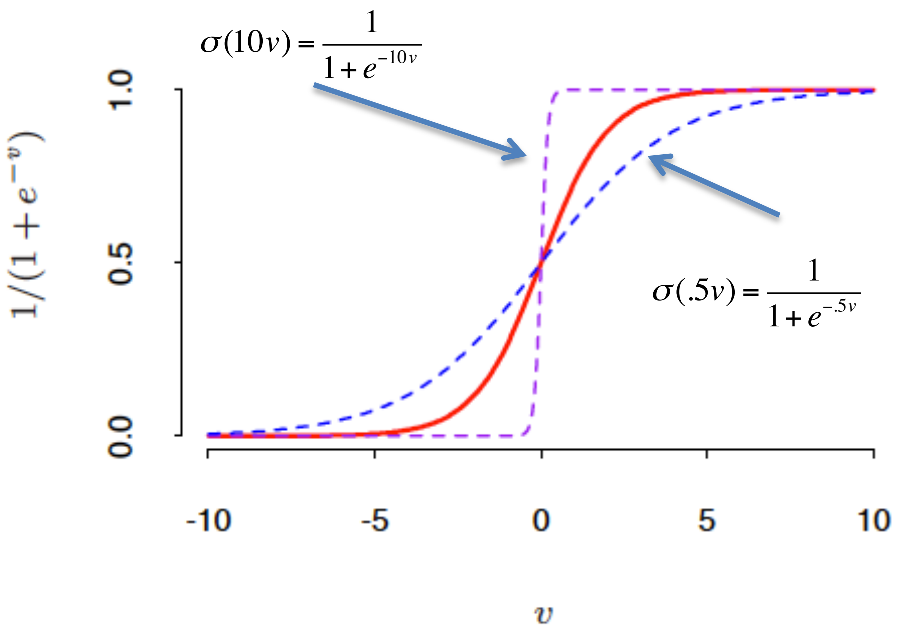
$$f_k(X) = g_k(T), \quad k = 1, \ldots, K,$$

where $Z = \left(Z_1, Z_2, \ldots, Z_M\right)$ and $T = \left(T_1, T_2, \ldots, T_K\right).$

Neural Networks

The activation function $\sigma(v)$, is typically chosen to be:

$$\sigma(sv) = \frac{1}{1+e^{-sv}}$$



*s controls the "activation rate" $\sigma(s(v-v_0))$ shifts the activation threshold from v to v_0 .

Neural Networks

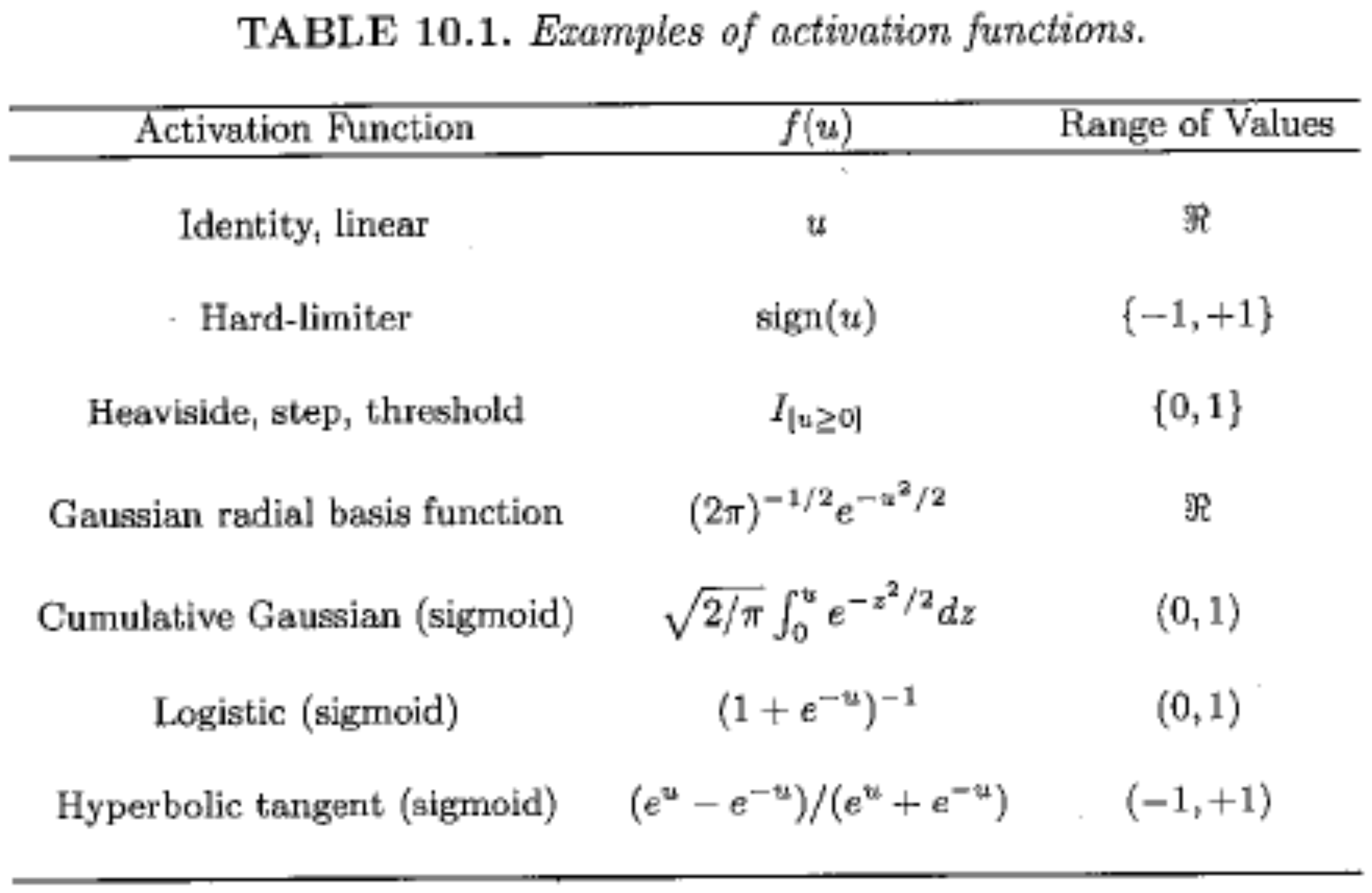


TABLE 10.1. *Examples of activation functions.*

Activation Function	$f(u)$	Range of Values
Identity, linear	u	$\Re$
Hard-limiter	$\text{sign}(u)$	$\{-1,+1\}$
Heaviside, step, threshold	$I_{[u \geq 0]}$	$\{0,1\}$
Gaussian radial basis function	$(2\pi)^{-1/2}e^{-u^2/2}$	$\Re$
Cumulative Gaussian (sigmoid)	$\sqrt{2/\pi}\int_0^u e^{-z^2/2}dz$	$(0,1)$
Logistic (sigmoid)	$(1+e^{-u})^{-1}$	$(0,1)$
Hyperbolic tangent (sigmoid)	$(e^u - e^{-u})/(e^u + e^{-u})$	$(-1,+1)$

Neural Networks

Output function: $g_k(T)$ allows for the final transformation of the vector of outputs T.

Typical Choices:

- Regression: $g_k(T) = T_k$.
- Classification (the same, or softmax):

$$g_k(T) = \frac{e^{T_k}}{\sum_{l=1}^{K} e^{T_l}}.$$

Neural Networks

Notice:

$$Z_m = \underline{\sigma}\left(\alpha_{0m} + \alpha_m^T X\right), \quad m = 1, \ldots, M,$$

Activation function

$$T_k = \beta_{0k} + \beta_k^T Z, \quad k = 1, \ldots, K,$$

$$f_k(X) = g_k(T), \quad k = 1, \ldots, K,$$

The activation function makes the model "non-linear".

Special case:

$\sigma = 1$ the hidden layer is a linear function of inputs.

* Neural Network thought of as a nonlinear generalization of the linear regression/classification problem.

Fitting Neural Networks

We have to estimate the weights, such that the training data is fit well.

- Let the complete set of weights be defined by Θ, which consists of:

$$\{\alpha_{0m}, \alpha_m; m = 1,2,\dots,M\} \quad M(p+1) \text{ weights}$$

$$\{\beta_{0m}, \beta_m; m = 1,2,\dots,K\} \quad K(M+1) \text{ weights.}$$

- We specify a loss function, for regression, typically:

$$R(\Theta) = \sum_{k=1}^{K} \sum_{i=1}^{N} \left(y_{ik} - f_k(x_i) \right)^2$$

- We specify a loss function, for classification, typically cross-entropy:

$$R(\Theta) = \sum_{k=1}^{K} \sum_{i=1}^{N} y_{ik} \log f_k(x_i).$$

and the corresponding classifier: $G(x) = \arg\max_k f_k(x)$.

Fitting Neural Networks

- As we know, going for the maximum in the training data will lead to over-fitting. Achieved by early stopping or penalty terms.

- Solution by “gradient-descent”, in this setting, “back-propagation algorithm”.

Due to the functional forms, and composition.

The gradients can be calculated fairly easily

Fitting Neural Networks

WLOG Consider “back-propagation” for squared error loss:

- Let $z_{mi} = \sigma\left(\alpha_{0m} + \alpha_m^T x_i\right)$ and $z_i = \left(z_{1i}, z_{2i}, \ldots, z_{Mi}\right)$.
- We have:

$$R(\Theta) = \sum_{i=1}^{N} R_i$$

$$= \sum_{i=1}^{N} \sum_{k=1}^{K} \left(y_{ik} - f_k\left(x_i\right)\right)^2$$

with derivatives

$$\frac{\partial R_i}{\partial \beta_{km}} = -2\left(y_{ik} - f_k(x_i)\right) g_k^{'}\left(\beta_k^T z_i\right) z_{mi}$$

$$\frac{\partial R_i}{\partial \alpha_{ml}} = -\sum_{k=1}^{K} 2\left(y_{ik} - f_k(x_i)\right) g_k^{'}\left(\beta_k^T z_i\right) \beta_{km} \sigma'\left(\alpha_m^T x_i\right) x_{il}.$$

Fitting Neural Networks

Given the derivatives, the gradient descent update at the (r+1) iteration is of the form:

$$\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{r}}$$

$$\alpha_{km}^{(r+1)} = \alpha_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{ml}^{r}}.$$

- Learning Rate

Fitting Neural Networks

Take a closer look at the derivatives:

$$\frac{\partial R_i}{\partial \beta_{km}} = -2\left(y_{ik} - f_k(x_i)\right) g_k^{'}\left(\beta_k^T z_i\right) z_{mi}$$

$$\frac{\partial R_i}{\partial \alpha_{ml}} = -\sum_{k=1}^{K} 2\left(y_{ik} - f_k(x_i)\right) g_k^{'}\left(\beta_k^T z_i\right) \beta_{km} \sigma'\left(\alpha_m^T x_i\right) x_{il}.$$

We can think of them in terms of errors.

Fitting Neural Networks

Error at the output layer δ_{ki}

Take a closer look at the derivatives:

$$\frac{\partial R_i}{\partial \beta_{km}} = -\underbrace{2\left(y_{ik} - f_k(x_i)\right) g_k^{'}\left(\beta_k^T z_i\right)}_{\delta_{ki}} z_{mi}$$

$$\frac{\partial R_i}{\partial \alpha_{ml}} = -\sum_{k=1}^{K} \underbrace{2\left(y_{ik} - f_k(x_i)\right) g_k^{'}\left(\beta_k^T z_i\right) \beta_{km} \sigma'\left(\alpha_m^T x_i\right) x_{il}}_{s_{mi}}.$$

Error at the hidden layer s_{mi}

The errors satisfy the following "back-propagation" equations:

$$s_{mi} = \sigma^{'}\left(\sigma_m^T x_i\right) \sum_{k=1}^{K} \beta_{km} \delta_{ki}.$$

Fitting Neural Networks

In practice two passes (forward and backwards):

1. **Forward pass**: the current weights are fixed and the predicted values $\hat{f}_k(x_i)$ are computed:

$$Z_m = \sigma\left(\alpha_{0m} + \alpha_m^T X\right), \quad m = 1, \ldots, M,$$

$$T_k = \beta_{0k} + \beta_k^T Z, \quad k = 1, \ldots, K,$$

$$f_k(X) = g_k(T), \quad k = 1, \ldots, K,$$

1. **Backward pass:** the errors δ_{ki} are computed, and them back-propagated through:

$$s_{mi} = \sigma'\left(\sigma_m^T x_i\right) \sum_{k=1}^{K} \beta_{km} \delta_{ki}.$$

to give the errors s_{mi}, which are used to update the gradient.

Fitting Neural Networks

Back Propagation

- Each hidden unit passes and receives information only to and from units that share a connection.

  Can be implemented in parallel.

- The updates:

$$\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{r}}$$

$$\alpha_{km}^{(r+1)} = \alpha_{km}^{(r)} - \gamma_r \underbrace{\sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{ml}^{r}}}.$$

Constant that can be Selected via line search.

- Can be regarded as a form of "batch learning", with the parameter Updates being a sum over all training observations.
- Learning can also be carried out one observation at a time, updating the gradient after each training case, and cycling through the training cases many times.

- Can be slow and often not the method of choice.

Training Obstacles

Overfitting - too many weights will overfit the data.

Strategies

Stopping early, using a validation set. Has the effect of shrinking towards a linear model.

Penalty terms: Add a penalty to regularize the error function:

$$R(\Theta) + \lambda J(\Theta)$$

Tuning parameter

where $J(\Theta) = \sum_{km} \beta_{km}^2 + \sum_{ml} \alpha_{ml}^2$

or, $J(\Theta) = \sum_{km} \frac{\beta_{km}^2}{1+\beta_{km}^2} + \sum_{ml} \frac{\alpha_{ml}^2}{1+\alpha_{ml}^2}.$

Stiffer penalty on Smaller weights

Training Obstacles

Rule of Thumb:

- Better to have too many hidden units than not enough.
- Can always shrink the weights toward zero.
- Typical number of hidden units is between 5-100, more with more inputs and training cases.
- The optimal number of hidden units can be estimated using a cross-validation.
- The use of multiple hidden layers allows for the construction of hierarchical features at different levels of resolution.

Training Obstacles

Multiple Minima:

- The optimization is non-convex.
- No unique solution, and solution is sensitive to starting values.

<u>Strategies:</u>

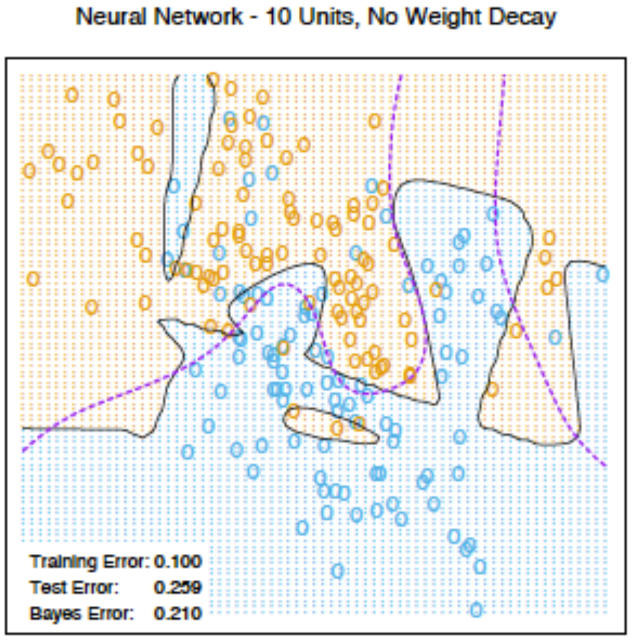
Multiple starting points

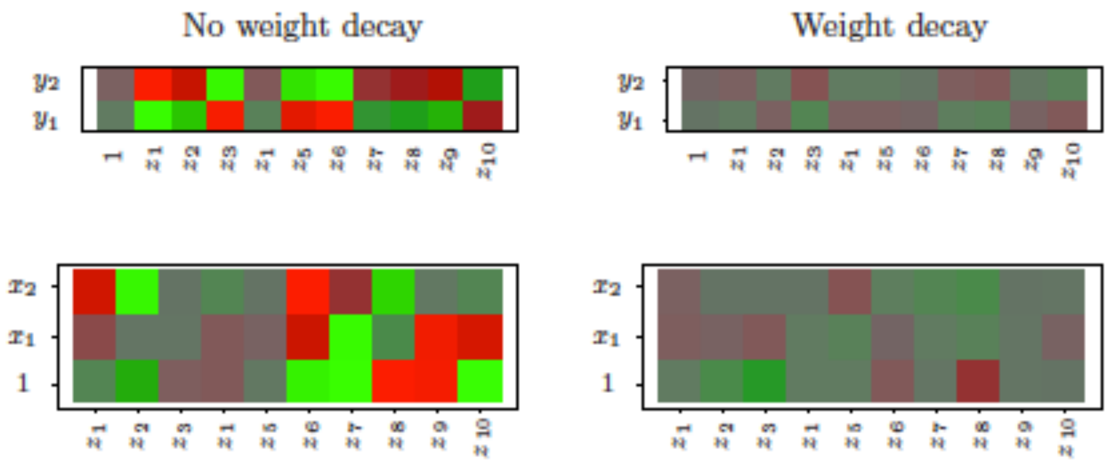
Averaging of solutions

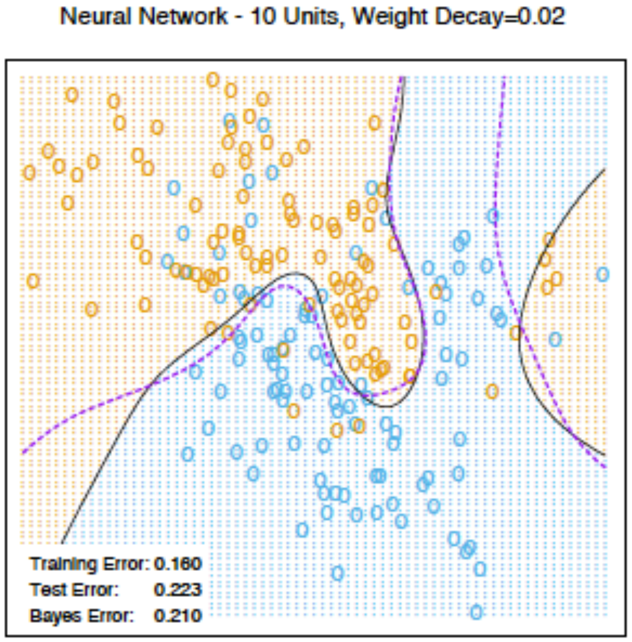
Bagging – works on randomly perturbed sets of the training data.

Simulation I

No penalty
Overfits the training data







Heat Maps of the estimated weights from the Training data.

Simulation II

Data Generated from the additive error models of the form:

$$Y = f(X) + \varepsilon$$

Sum of sigmoids: $Y = \sigma\left(a_1^T X\right) + \sigma\left(a_2^T X\right) + \varepsilon_1$

Radial: $Y = \prod_{m=1}^{10} \phi\left(X_m\right) + \varepsilon_2$

Where the input is: $X^T = \left(X_1, X_2, \ldots, X_p\right)$ where each X_j is a standard Gaussian, p=2 for the sigmoid model, and p=10 for the radial model. Both have Gaussian errors with equal signal to noise ratios.

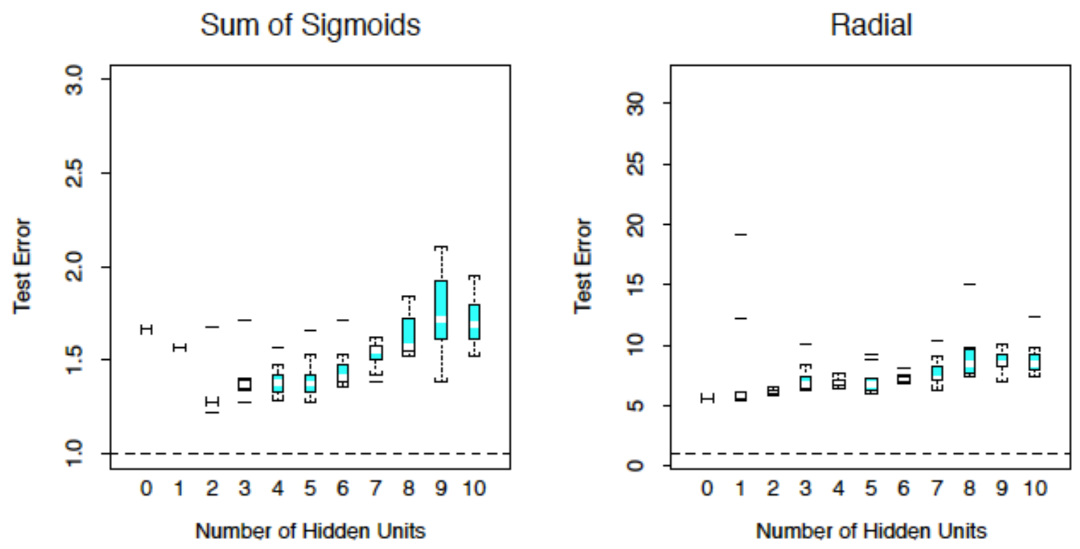
In the sigmoid model: a1 = (3,3), a2 = (3,-3).

In the radial model: $\phi\left(t\right) = \left(1/2\pi\right)^{1/2} \exp\left(-t^2/2\right)$.

Simulation II

Data fitting (mild regularization 1e-5):

- With weight decay and various numbers of hidden units, recorded the average test error for 10 random starting weights:
  $E_{Test}\left(Y-\hat{f}(X)\right)^2$ for each of 10 random starting weights.



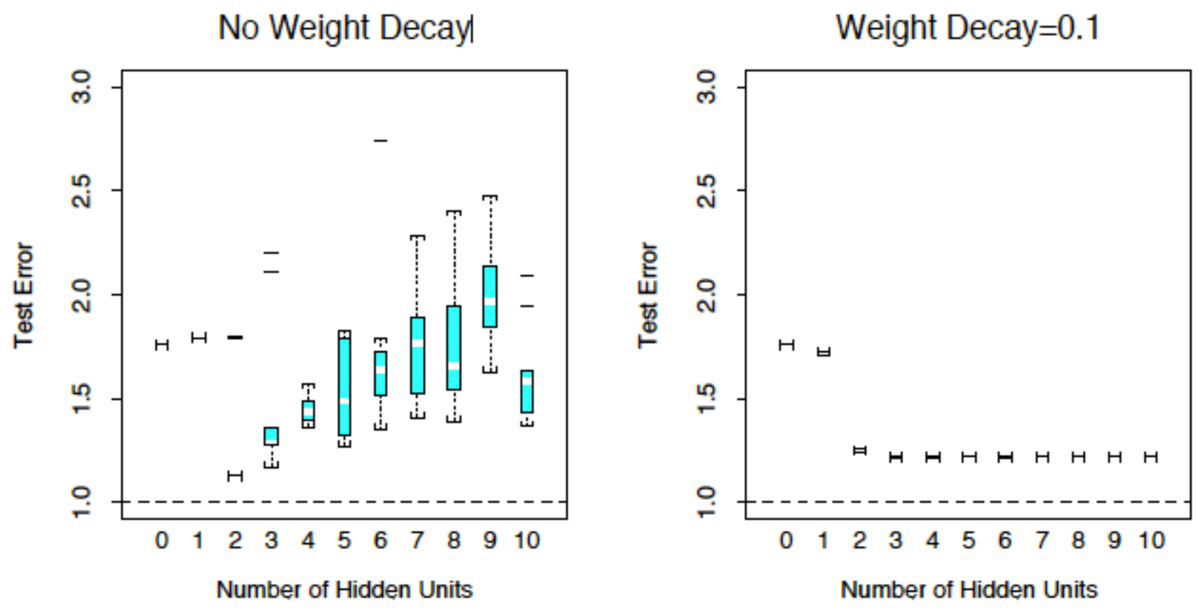
More hidden units, more overfitting, more sensitivity to starting weights

FIGURE 11.6. *Boxplots of test error, for simulated data example, relative to the Bayes error (broken horizontal line). True function is a sum of two sigmoids on the left, and a radial function is on the right. The test error is displayed for 10 different starting weights, for a single hidden layer neural network with the*

Simulation II

Data fitting sum of sigmoids (no regularization vs 1e-1):

- Same scenario, but now a fixed weight decay parameter of 1e-1 a stronger regularization and no regularization.



1e-1 seems to be Fitting all of the Models well without Overfitting

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*

Simulation II

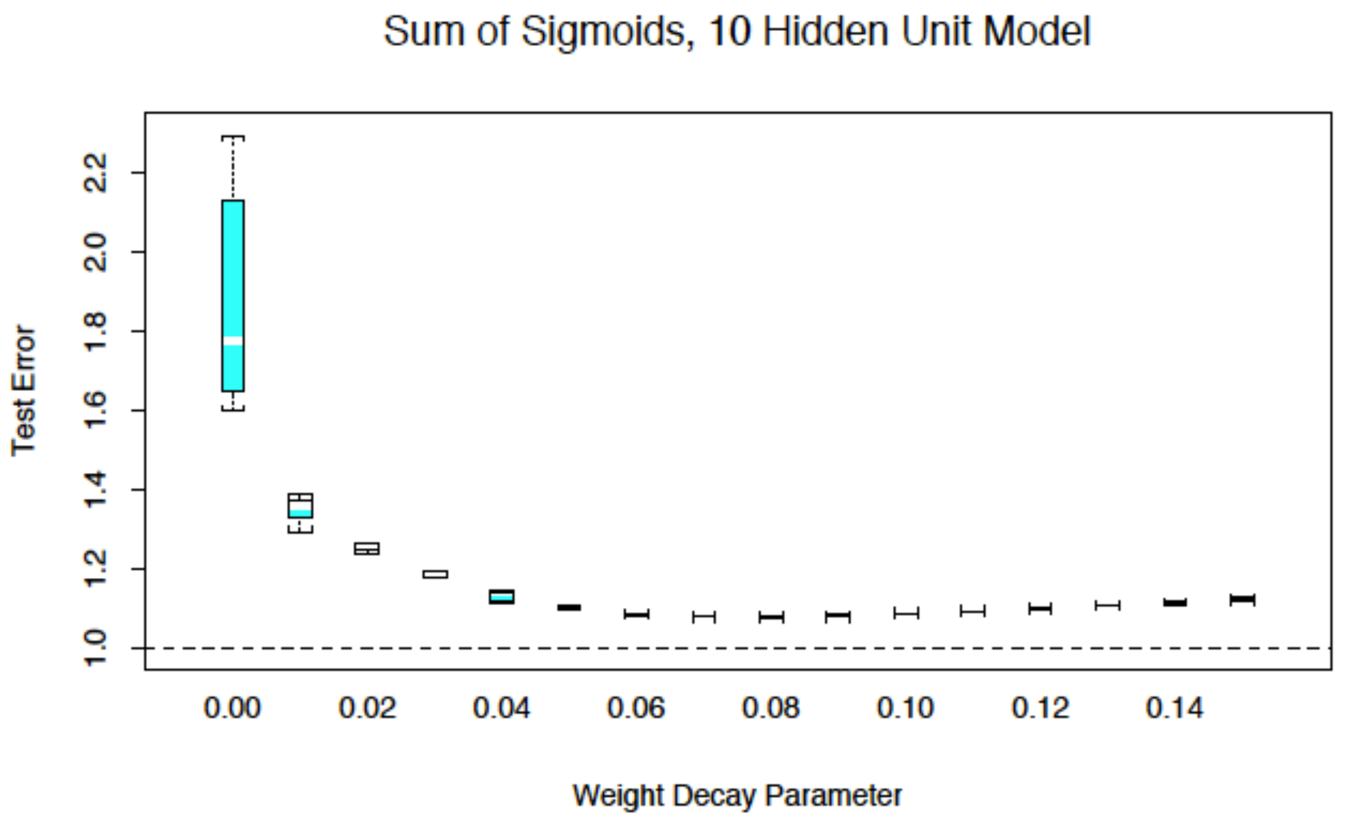


FIGURE 11.8. *Boxplots of test error, for simulated data example. True function is a sum of two sigmoids. The test error is displayed for ten different starting weights, for a single hidden layer neural network with ten hidden units and weight*

Simulation II

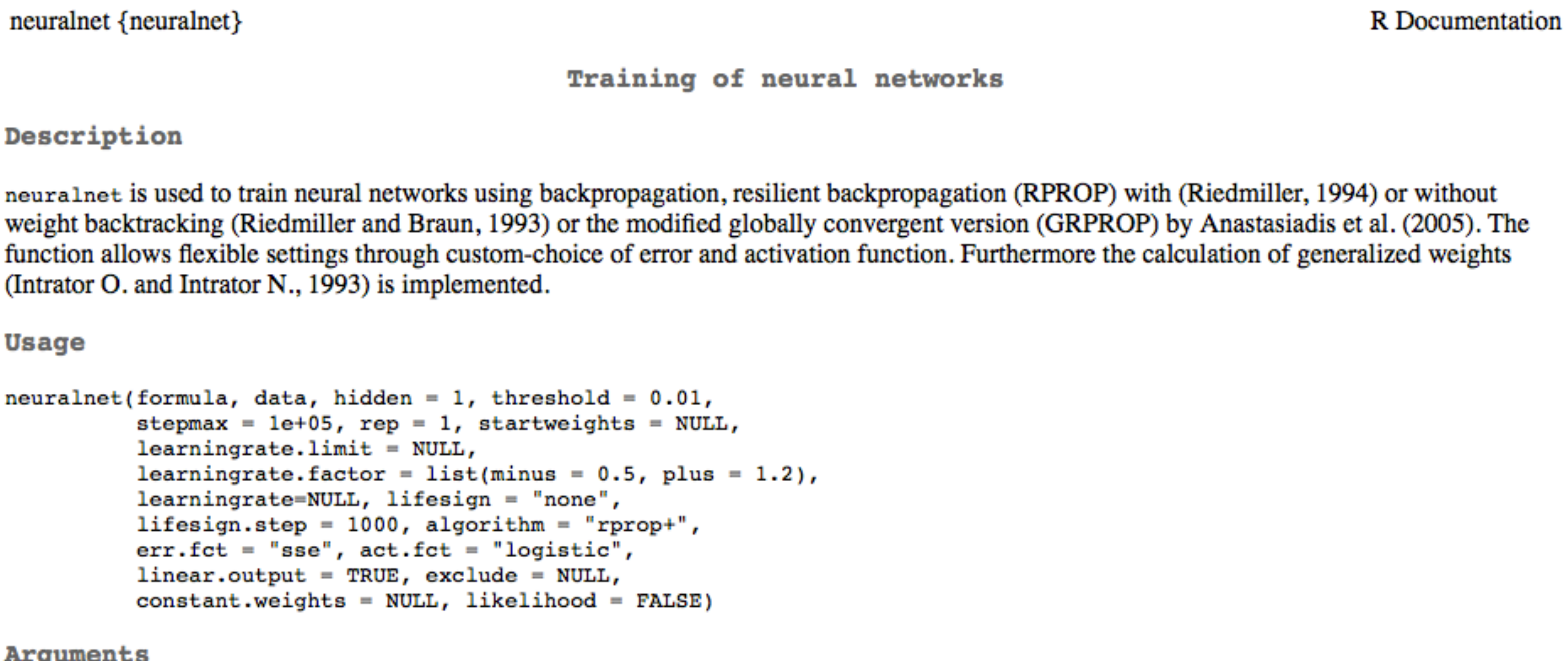
<u>In practice:</u>

There are two free parameters to select in the algorithm:

- The weight decay λ .
- The number of hidden units M.
- The number of layers (not discussed).

<u>One strategy:</u>

Fix either parameter to the least constrained model, ensuring it is rich enough, and use cross-validation to choose the other.



neuralnet {neuralnet} R Documentation

Training of neural networks

Description

`neuralnet` is used to train neural networks using backpropagation, resilient backpropagation (RPROP) with (Riedmiller, 1994) or without weight backtracking (Riedmiller and Braun, 1993) or the modified globally convergent version (GRPROP) by Anastasiadis et al. (2005). The function allows flexible settings through custom-choice of error and activation function. Furthermore the calculation of generalized weights (Intrator O. and Intrator N., 1993) is implemented.

Usage

```
neuralnet(formula, data, hidden = 1, threshold = 0.01,
          stepmax = 1e+05, rep = 1, startweights = NULL,
          learningrate.limit = NULL,
          learningrate.factor = list(minus = 0.5, plus = 1.2),
          learningrate=NULL, lifesign = "none",
          lifesign.step = 1000, algorithm = "rprop+",
          err.fct = "sse", act.fct = "logistic",
          linear.output = TRUE, exclude = NULL,
          constant.weights = NULL, likelihood = FALSE)
```

Arguments

Conclusions

- Both PPR and Neural Networks rely on “derived inputs”, which are nonlinear functions of the inputs.
- Powerful approach competitive with other learning methods.
- Useful when prediction, rather than interpretation is the goal.
- Has been extended to the Bayesian Domain.
- NIPS 2003: 2-class classification competition, which dimensional data, mass spec, text, drug discovery, digit recognition, etc.

  - Bayesian Neural Networks *best performance, longest runtime

  - Boosted Trees

  - Boosted Neural Networks

  - Random Forests

  - Bagged Neural Networks