# Predictive Modeling

## Chapter 7: Nonlinear Regression Models

## STA 6543

## The University of Texas at San Antonio

# Overview

- Part I: General Strategies
- Part II: Regression Models
  - Chapter 6: Linear Regression and Its Cousins
  - Chapter 7: Nonlinear Regression Models
  - Chapter 8: Regression Trees and Rule-Based Models
- Part III: Classification Models
  - Chapter 12: Discriminant Analysis and Other Linear Classification Models
  - Chapter 13: Nonlinear Classification Models
  - Chapter 14: Classification Trees and Rule-Based Models

# Nonlinear regression models

- A motivating example about solubility data
- Four types of nonlinear models
  - Neural Networks
  - Multivariate Adaptive Regression Splines
  - Support Vector Machines
  - K-Nearest Neighbors
- R demonstrations

# Motivations

- In Chapter 6, we learned a number of approaches for modeling the relationship between a response Y and a set of predictors X.
- Many of these models can be adapted to nonlinear trends in the data by manually adding terms (e.g., squared terms) with knowing the specific nature of the nonlinearity of the data.
- In this chapter, we consider nonlinear regression models without knowing the exact form of the nonlinearity
  - Neural networks
  - Multivariate adaptive regression splines (MARS)
  - Support vector machines (SVMs)
  - *K*-nearest neighbors (*K*NNs)

# Neural networks

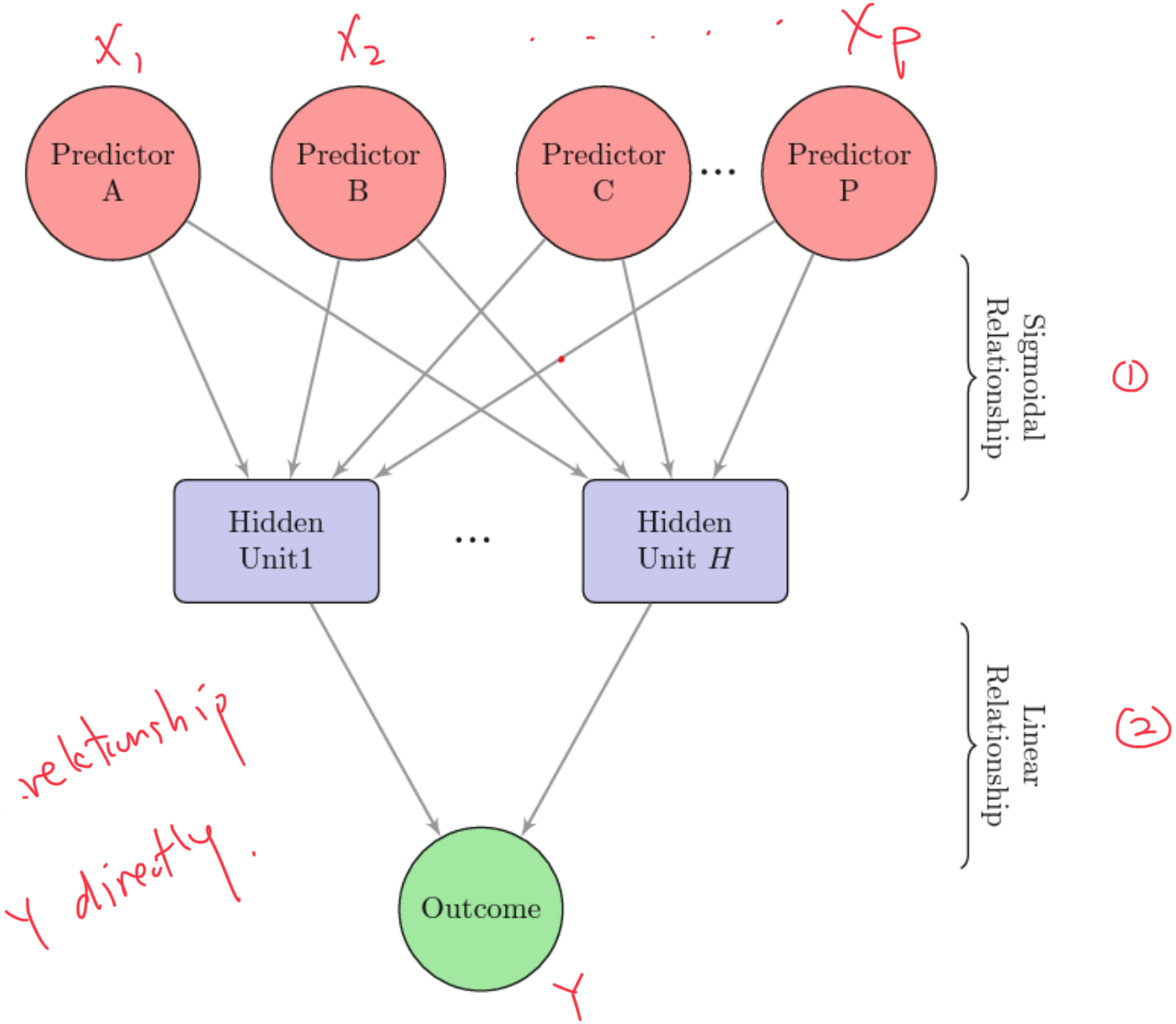
Chapter 7-Part II Regression Models

# Neural networks

- The term *neural network* has evolved to encompass a large class of models and learning methods.
- Neural network can be viewed as a multi-stage regression or classification model, typically represented by a *network diagram.*
- We focus the most widely used *"vanilla" neural net*, sometimes called *the single hidden layer back-propagation network, or single layer perceptron*.
- The central idea is to extract linear combinations of the inputs as derived features, and then model the target as a nonlinear function of these features.

↓ method for finding unknown parameters in the model

A diagram of a neural network with a single hidden layer.




We don't build the relationship between X and Y directly.

# Neural networks

- Each hidden unit is a linear combination of some or all of the predictor variables.
- The linear combination is typically transformed by a nonlinear (*activation*) function *g(·),* such as the logistic (i.e., sigmoidal) function

$$h_k(x) = g\left(\beta_{0k} + \sum_{j=1}^{P} x_j \beta_{jk}\right)$$

$$g(u) = \frac{1}{1 + e^{-u}}$$

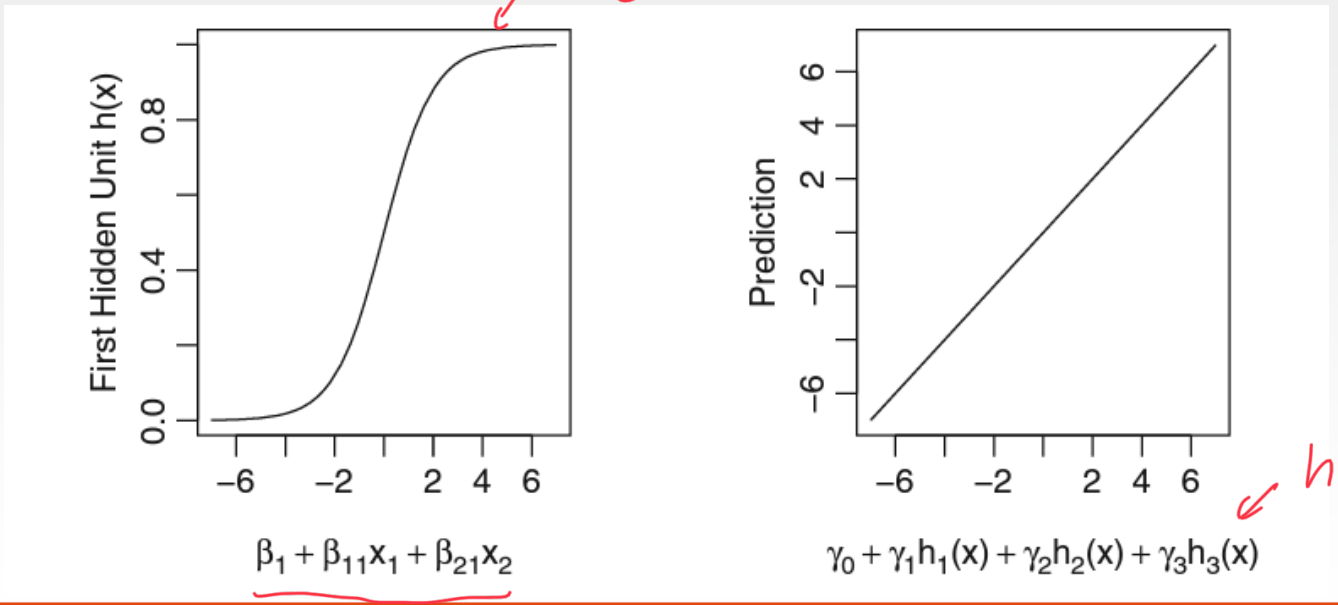
- The output is modeled by a linear combination of the hidden units.

# Neural networks

$h_k(x) = \beta_{0k} + \sum_{j=1}^{p} x_j \beta_{jk}$

- Notice that if *g()* is the identity function, then the entire model collapses to a linear model in the inputs.
- By introducing the nonlinear transformation *g()*, it greatly enlarges the class of linear models.

stage 1: predictors & hidden units

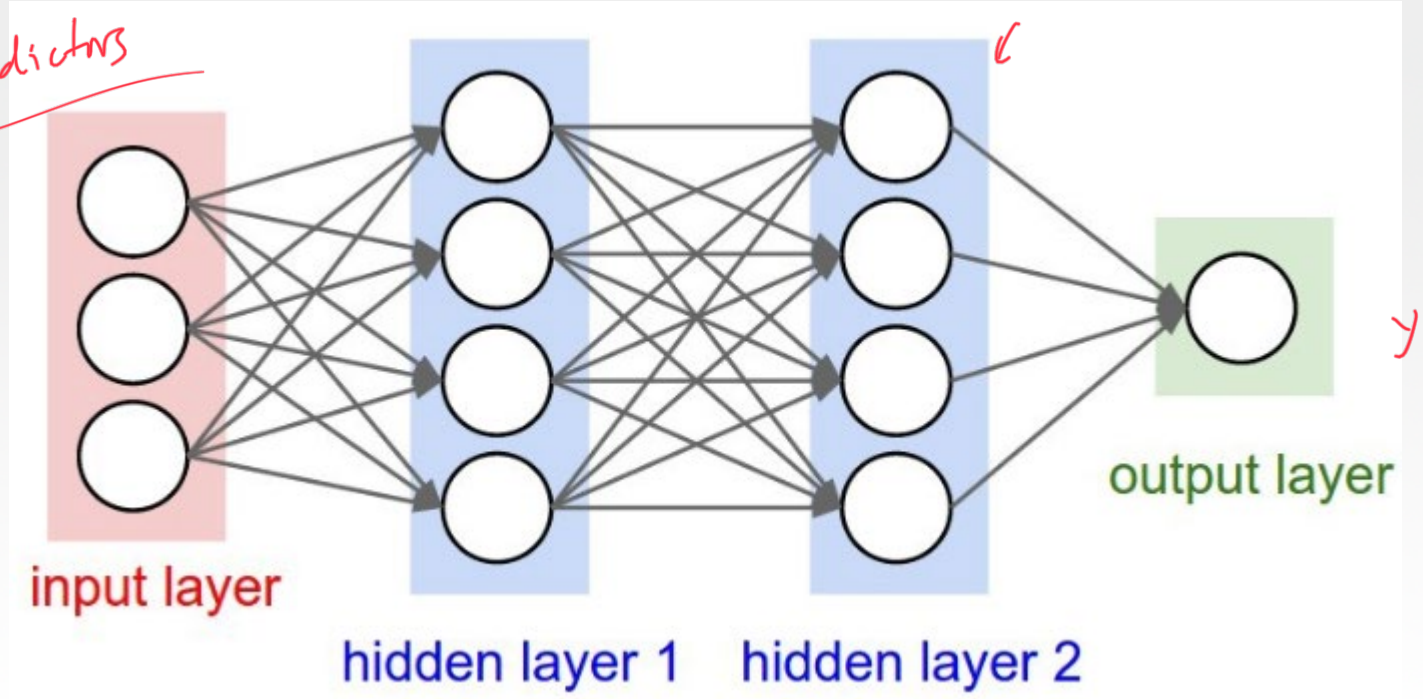



stage 2: response & hidden units

hidden units

# Remarks

- The β coefficients are similar to regression coefficients; coefficient $\beta_{jk}$ is the effect of the *j*th predictor on the *k*th hidden unit.
- In general, there can be more than one hidden layers.



# Neural networks

- Once the number of hidden units is defined, each unit must be related to the outcome. Another linear combination connects the hidden units to the outcome:

$$y = f(x) = \gamma_0 + \sum_{k=1}^{H} \gamma_k h_k$$

$h_k$: hidden unit. We have $H$ hidden units

- The neural network model has unknown parameters, often called *weights*. Denote the complete set of weights by $\theta$, which consists of

$$\{\beta_{0k}, \beta_{11}, \ldots, \beta_{Pk}, k = 1,2,\ldots,H\}$$

$k=1$: $p+1$; $k=2$: $p+1$; $k=H$: $p+1$ $\Rightarrow (p+1)H$ unknown parameters

$$\{\gamma_0, \ldots, \gamma_H\} \Rightarrow (H+1) \text{ unknown parameters}$$

$$\theta = \{\beta_{0k}, \beta_{11}, \ldots \beta_{pk}, k=1,2,\ldots H, \gamma_0, \ldots \gamma_H\}$$

# Illustration

- For this type of network model and P predictors, there are a total of $H(P + 1)+H+1$ total parameters being estimated, which quickly becomes large as the number of predictors $P$ increases.
- For the solubility data, recall that there are $P = 228$ predictors. A neural network model with three hidden units ($H = 3$) would estimate 691 parameters while a model with five hidden units ($H = 5$) would have 1,151 coefficients, which is larger than the sample size $n = 951$.

and H hidden units

and n = 951

When P = 228 and H = 5

5(228 + 1) + 5 + 1 = 1151 unknown parameters > n = 951

# Fitting neural networks

- For regression, we can use sum-of-squared errors as our measure of fit (K = 1)

$$\min_{\theta} R(\theta) = \sum_{i=1}^{n} (y_i - f_i(x))^2$$

OLS method. when n > # of parameters

- Neural networks usually tend to *over-fit* the relationship between the predictors and the response due to the large number of regression coefficients. $\theta$: = $H(p+1) + H+1$
- A common approach to moderating over-fitting is to use *weight decay*, a penalization method to regularize the model similar to ridge regression discussed in Chapter 6.

OLS. $\hat{\beta} = (X'X)^{-1}X'y$ ridge $\hat{\beta} = (X'X + \lambda I)^{-1}X'y$

# Fitting neural networks

- Formally, the optimization produced would try to minimize an alternative version of the sum of the squared errors:

$$R(\theta) = \sum_{i=1}^{n} (y_i - f_i(x))^2 + \lambda \sum_{k=1}^{H} \sum_{j=0}^{P} \beta_{jk}^2 + \lambda \sum_{j=0}^{P} \gamma_k^2$$

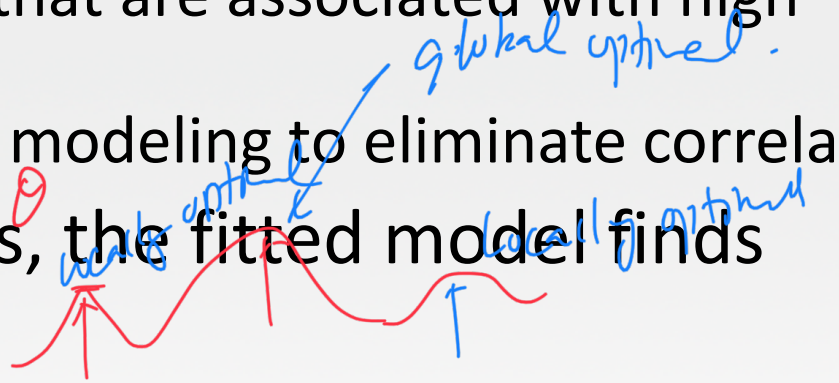
for a given λ value.

penalty term to control the over-fitting issue.

- As the regularization value λ increases, the fitted model becomes more smooth and less likely to over-fit the training set.
- In practice, reasonable values of λ range between 0 and 0.1.

# Remarks

- The predictors should be *centered and scaled* prior to modeling.
- These models are often *adversely* affected by *high correlation* among the predictor variables
  - Pre-filter the predictors to remove the predictors that are associated with high correlations.
  - Principal component analysis can be used prior to modeling to eliminate correlations
- Due to a large number of unknown parameters, the fitted model finds parameter estimates that are locally optimal.




- That is, different locally optimal solutions can produce models that are very different but have nearly equivalent performance.
- To get stable prediction, *model averaging* based on several models from different starting values are suggested.

# Load the data

```
#Required packages
library(AppliedPredictiveModeling)
library(caret)
library(earth)

### Load the data
data(solubility)

### Create a control function that will be used across models. We
### create the fold assignments explicitly instead of relying on the
### random number seed being set to identical values.

set.seed(100)
indx <- createFolds(solTrainY, returnTrain = TRUE)
ctrl <- trainControl(method = "cv", index = indx)
```

10-folder C.V.

# R codes for neural networks

```
### Neural Networks

#Create a grid for tuning parameters
nnetGrid <- expand.grid(decay = c(0, 0.01, .1),
                        size = c(1, 3, 5, 7),
                        bag = FALSE)

#It takes time run
##The following codes takes about 6,000 seconds to run in my computer.
## Your running time may be different depending
## on your cpu.
```

# R codes for neural networks

```
ptm <- proc.time() #takes more than 6,000 seconds to run in my computer
set.seed(100)
nnetTune <- train(x = solTrainXtrans, y = solTrainY,
                  method = "avNNet",
                  tuneGrid = nnetGrid,
                  trControl = ctrl,
                  preProc = c("center", "scale"),
                  linout = TRUE,
                  trace = FALSE,
                  MaxNWts = 13 * (ncol(solTrainXtrans) + 1) + 13 + 1,
                  maxit = 1000,
                  allowParallel = FALSE)
nnetTune
proc.time() - ptm
### Stop the clock

plot(nnetTune)
```

# R codes for neural networks

```
#save the predicted values into testResults
testResults <- data.frame(obs = solTestY,
                          NNet = predict(nnetTune, solTestXtrans))
```

test data for response variable

function of neural network.

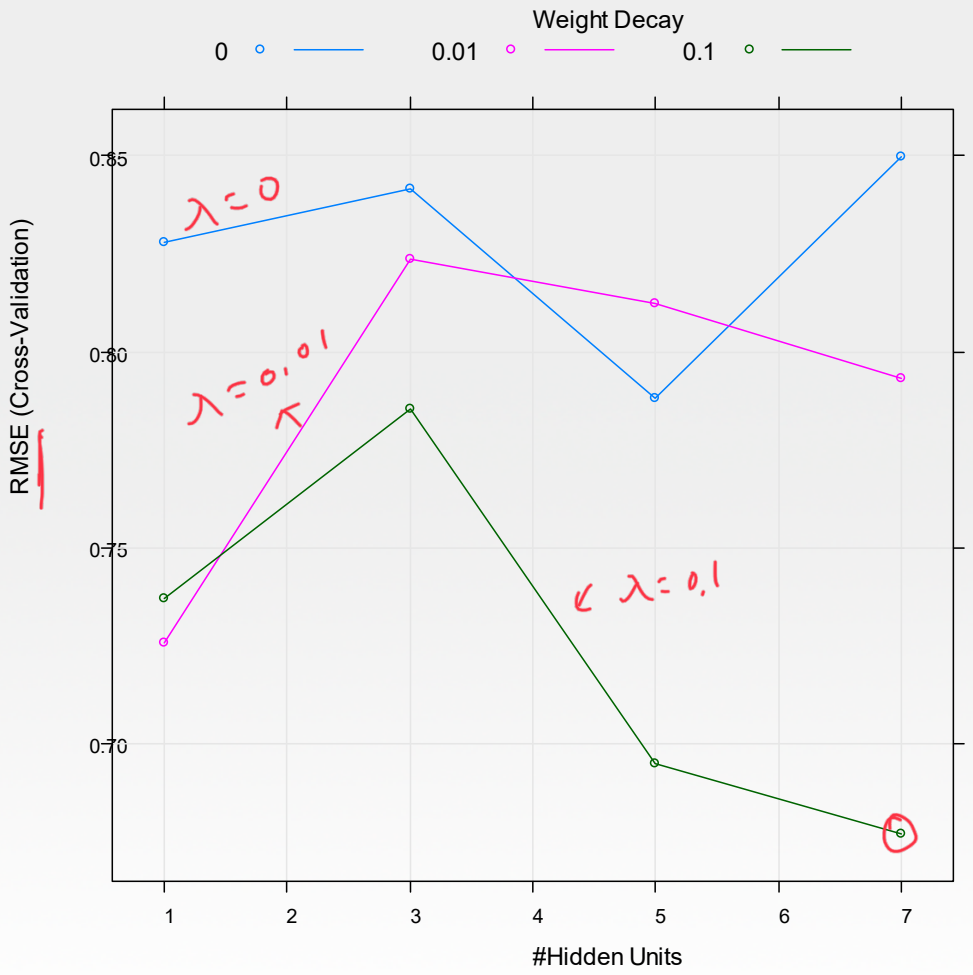
# Neural Networks

```
> nnetTune
Model Averaged Neural Network

951 samples
228 predictors

Pre-processing: centered (228), scaled (228)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 856, 855, 857, 856, 856, 855, ...
Resampling results across tuning parameters:

  decay  size  RMSE       Rsquared   MAE
  0.00   1     0.8279833  0.8411337  0.6351877
  0.00   3     0.8415406  0.8320163  0.6362154
  0.00   5     0.7879532  0.8518015  0.5818827
  0.00   7     0.8497575  0.8325912  0.6314441
  0.01   1     0.7255300  0.8766697  0.5318100
  0.01   3     0.8237462  0.8456118  0.6105706
  0.01   5     0.8124533  0.8517685  0.5963451
  0.01   7     0.7930880  0.8551410  0.5678490
  0.10   1     0.7369941  0.8724582  0.5393379
  0.10   3     0.7854472  0.8560870  0.5544581
  0.10   5     0.6950386  0.8879509  0.5015236
  0.10   7     0.6769793  0.8928230  0.4863267

Tuning parameter 'bag' was held constant at a value of FALSE
RMSE was used to select the optimal model using the smallest value.
The final values used for the model were size = 7, decay = 0.1 and bag = FALSE.
```

smallest

λ = 0.1

# Neural Networks




Tuning parameters

# of hidden units is 7

$\lambda = 0.1$