Discovering Knowledge in Data

Daniel T. Larose, Ph.D.

Chapter 9
Neural Networks

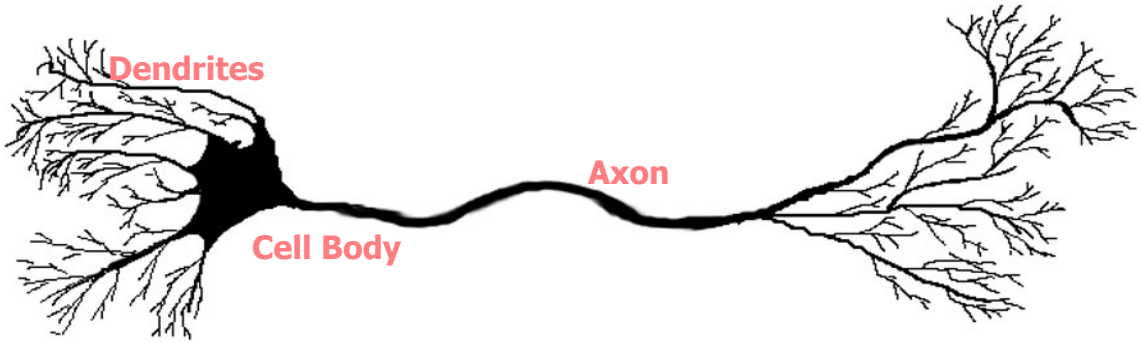
Prepared by James Steck and Eric Flores

Neural Networks

- Neural Networks
  - Complex learning systems recognized in animal brains
  - Single neuron has simple structure
  - Interconnected sets of neurons perform complex learning tasks
  - Human brain has 10^{15} synaptic connections
  - Artificial Neural Networks attempt to replicate non-linear learning found in nature

Neural Networks (*cont'd*)

- Dendrites gather inputs from other neurons and combine information
- Then generate non-linear response when some threshold is reached
- Signal sent to other neurons via axon



Neural Networks (*cont'd*)

- Artificial neuron model is similar
- Data inputs (x_i) are collected from upstream neurons and combined through a combination function (e.g., sigma)
- Combined data is then input into a nonlinear activation function to produce an output response (i.e., y)
- Output response y is then channeled downstream to other neurons.

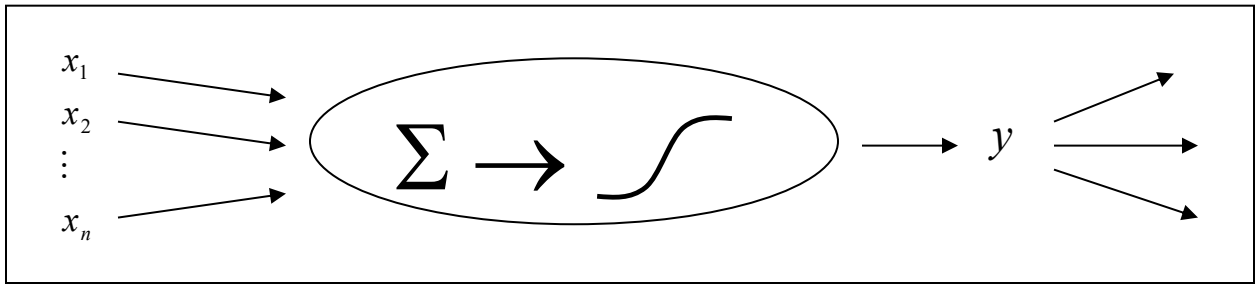


Figure 9.1

Neural Networks (*cont'd*)

- What problems are applicable for Neural Networks (NN)?
  - NN are quite robust with respect to noisy data (advantage)
    - Contain many nodes (neurons) with weights assigned to each connection
    - Can learn to work around noisy or erroneous data
  - Results are not transparent or intuitive to human interpretation (unlike Decision Trees) (disadvantage)
  - Often require longer training times compared to Decision Trees (several hours) (disadvantage)

Input and Output Encoding

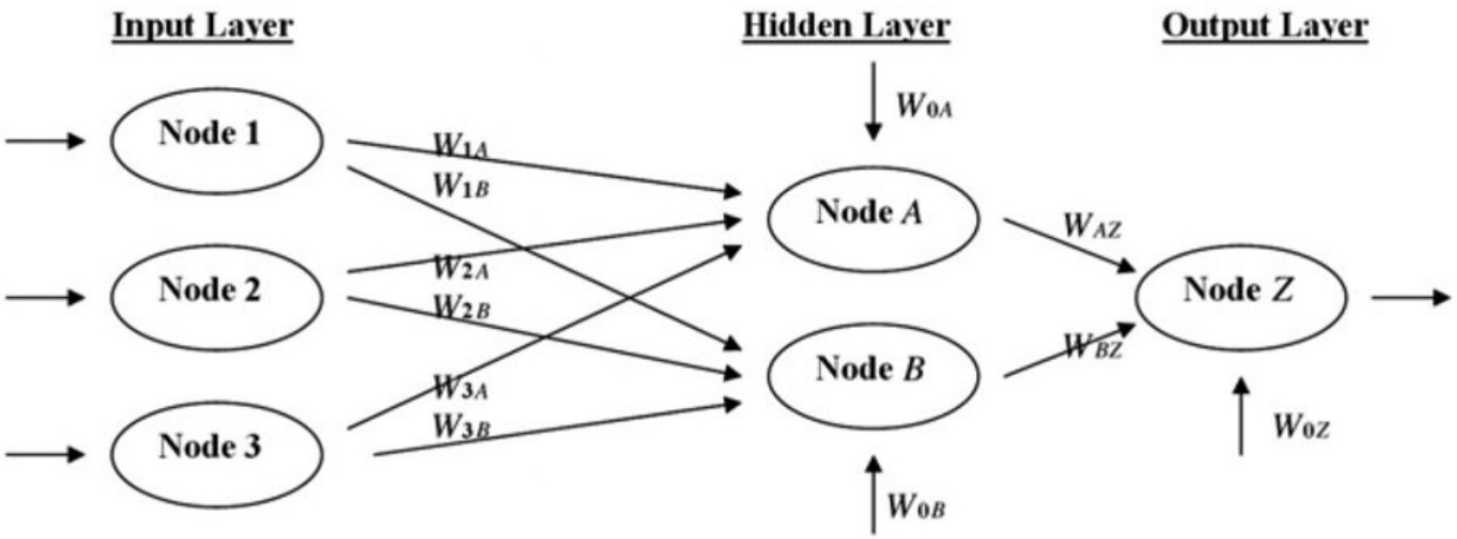


Figure 9.2 Simple neural network.

Input and Output Encoding

Neural Networks input and output nodes receive and return values encoded to [0, 1]

Input:

- Numeric
  - Apply Min-max Normalization to continuous variables

$$X^* = \frac{X - \min(X)}{\text{range}(X)} = \frac{X - \min(X)}{\max(X) - \min(X)}$$

  - Works well when Min and Max known
  - Also assumes new data values occur within Min-Max range
  - Values outside range may be rejected or mapped to Min or Max

Input and Output Encoding (*cont'd*)

- Categorical
  - Indicator (flag) Variables are used when the number of possible categories is small
  - Categorical variable with *k* classes translated to $k - 1$ indicator variables
    - For example, *Gender* attribute values are "Male", "Female", and "Unknown"
    - Classes $k = 3$
    - Create $k - 1 = 2$ indicator variables named $Male\_I$ and $Female\_I$
    - *Male* records have values $Male\_I = 1, Female\_I = 0$
    - *Female* records have values $Male\_I = 0, Female\_I = 1$
    - *Unknown* records have values $Male\_I = 0, Female\_I = 0$

Input and Output Encoding (*cont'd*)

- Be wary of reordering unordered categorical values to [0, 1] range
  - For example, attribute *Marital_Status* has values "Divorced", "Married", "Separated", "Single", "Widowed", and "Unknown"
  - Suppose values coded as 0.0, 0.2, 0.4, 0.6, 0.8, and 1.0, respectively
  - Coding implies "Divorced" is closer to "Married", and farther from "Separated"
  - Neural Network is only aware of the numeric values
  - It is naive to pre-encoded meaning of these categorical values
  - Results of network model may be meaningless

Input and Output Encoding (*cont'd*)

Output:

- Neural Network output nodes always return continuous values between [0, 1]
- Many classification problems have two outcomes
- One option is to use threshold established *a priori* in single output node to separate classes
  - For example, target variable is "leave" or "stay" for a customer
  - Threshold value is "leave if output >= 0.67"
  - An output of 0.72 from the output node classifies the record as "leave"

Input and Output Encoding (*cont'd*)

- Single output nodes may also be used when target classes are clearly ordered
  - For example, classify elementary-level reading ability
  - Define thresholds:

"if 0.00 <= output < 0.25"	"first-grade reading level"
"if 0.25 <= output < 0.50"	"second-grade reading level"
"if 0.50 <= output < 0.75"	"third-grade reading level"
"if output >= 0.75"	"fourth-grade reading level"

- Fine-tuning of thresholds may be required

Input and Output Encoding (*cont'd*)

- Single output node not applicable to all classification problems
  - For example, target variable is Marital_Status
  - Contains unordered categories: divorced, married, separated, single, widowed, and unknown
  - Use 1-of-n Output Encoding, where one output node is used for each target class
  - Network has six output nodes in the output layer
  - Output node with highest value is chosen as the classification for a record
- 1-of-n Output Encoding provides a measure of confidence in classification (Benefit)
  - Confidence is in the form of the difference between the highest and the second-highest value in output nodes

Neural Networks for Estimation and Prediction

- Continuous output in NN is useful for estimation and prediction problems
- For example, predict a stock price 3 months from now
- Input values normalized using Min-max Normalization
- Output values [0, 1] need to be denormalized to represent the scale of the stock prices

Prediction = output(data range) + minimum

- For example, suppose stock price ranged $20 to $30 dollars
- Network output prediction value = 0.69

0.69(30.0 – 20.0) + 20.0 = $26.90

Simple Example of a Neural Network

- Neural Network consists of a layered, feedforward, completely connected network of artificial neurons, or nodes.
- The Feedforward nature restricts network flow to a single direction (no looping or cycling)
- Neural network is composed of two or more layers. Most networks have three layers (input, hidden, output)

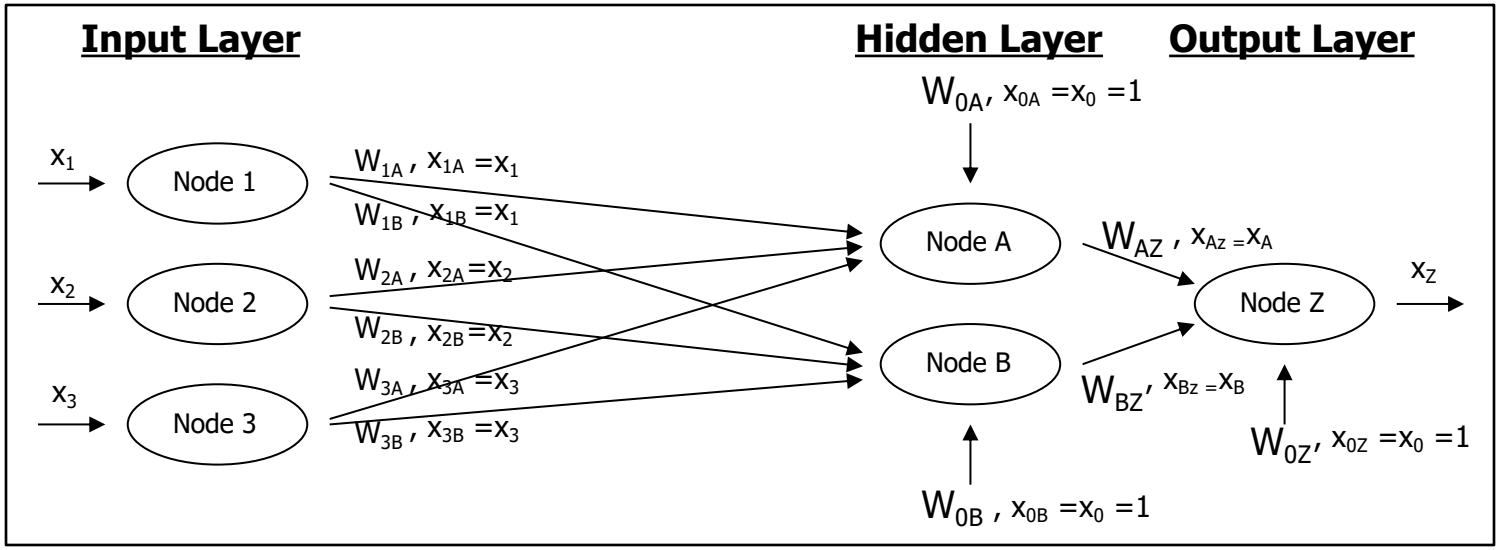


Figure 9.2

Simple Example of a Neural Network (*cont'd*)

- Network may contain more than one hidden layer
- Network is completely connected, meaning each node in a given layer is connected to every node in the next layer, although not to other nodes in the same layer.
- Every connection has a weight (W_{ij}) associated with it.
- At initialization, these weights are randomly assigned to values between zero and one.
- Number of input nodes depends on the number and type of attributes in the data set
- The number of hidden layers and the number of nodes in each hidden layer are both configurable by the user.
- The number of nodes in the output layer depends on the particular classification task at hand.

Simple Example of a Neural Network (*cont'd*)

- Hidden Layer
  - How many nodes in hidden layer?
  - More nodes in the hidden layer increases the power and flexibility of the network for identifying complex patterns
  - Large number of hidden nodes leads to overfitting, memorizing the training set, loss of generalizability.
  - Reduce the number of hidden nodes when overfitting occurs
  - Increase the number of hidden nodes when training accuracy is unacceptably low
- Input Layer
  - Input layer accepts inputs from the data set
  - Passes these values to the hidden layer without further processing.
  - Input layer nodes lack detailed structure compared to hidden and output layer nodes

Simple Example of a Neural Network (*cont'd*)

- In a hidden layer node or an output layer node, first, a combination function (usually summation, Σ) produces a linear combination of node inputs and connection weights into a single scalar value called "net".
- For a given hidden or output node j:

$$\text{net}_j = \sum_i W_{ij} x_{ij} = W_{0j} x_{0j} + W_{1j} x_{1j} + \ldots + W_{Ij} x_{Ij}$$

where

- We assume there are $I + 1$ inputs to node j
- x_{ij} is i^{th} input to node j
- W_{ij} is the weight associated with the i^{th} input to node j
- $x_1, x_2, \ldots, x_I$ are inputs from upstream nodes. Note $x_{ij} = x_i$ for all i
- x_0 is a constant input value $= x_{0j} = 1.0$
- Each hidden layer or output layer node j has extra input $W_{0j}x_{0j} = W_{0j}$

Simple Example of a Neural Network *(cont'd)*

- Example – Using values from table 9.1
  - The scalar value computed for hidden layer Node A equals

$$\text{net}_A = \sum_i W_{iA} x_{iA} = W_{0A}(1.0) + W_{1A} x_{1A} + W_{2A} x_{2A} + W_{3A} x_{3A} =$$

$$0.5 + 0.6(0.4) + 0.8(0.2) + 0.6(0.7) = 1.32$$

  - Within Node A, $\text{net}_A = 1.32$ is the input to an activation function

Table 9.1

$x_0 = 1.0$	$W_{0A} = 0.5$	$W_{0B} = 0.7$	$W_{0Z} = 0.5$
$x_1 = 0.4$	$W_{1A} = 0.6$	$W_{1B} = 0.9$	$W_{AZ} = 0.9$
$x_2 = 0.2$	$W_{2A} = 0.8$	$W_{2B} = 0.8$	$W_{BZ} = 0.9$
$x_3 = 0.7$	$W_{3A} = 0.6$	$W_{3B} = 0.4$	

Simple Example of a Neural Network (*cont'd*)

- This activation is analogous to how neurons "fire" nonlinearly in biological organisms. Signals are sent between neurons when the combination of inputs to a neuron passes a threshold.
- Firing response not necessarily linearly related to increase in input stimulation
- Artificial Neural Networks model this behavior using a non-linear activation function
- Sigmoid function most commonly used

$$y = \frac{1}{1+e^{-x}}$$, where e=2.718281828

- In Node A, sigmoid function takes net_A = 1.32 as input and produces output

$$y = \frac{1}{1+e^{-1.32}} = 0.7892 = X_A$$

Simple Example of a Neural Network (*cont'd*)

- Node A outputs 0.7892 along connection to Node Z, and becomes component of net_Z
- Before net_Z is computed, contribution from Node B required

$$\text{net}_B = \sum_i W_{iB} x_{iB} = W_{0B}(1.0) + W_{1B} x_{1B} + W_{2B} x_{2B} + W_{3B} x_{3B} = 0.7 + 0.9(0.4) + 0.8(0.2) + 0.4(0.7) = 1.5$$

- then

$$f(\text{net}_B) = \frac{1}{1 + e^{-1.5}} = 0.8176 = X_B$$

- Node Z combines outputs from Node A and Node B, through net_Z, a weighted sum, using weight associated to the connections between nodes

Simple Example of a Neural Network (*cont'd*)

- Inputs X_i to Node Z are not data attribute values, but the outputs from the sigmoid functions from upstream nodes

$$\text{net}_Z = \sum_i W_{iZ} x_{iZ} = W_{0Z}(1.0) + W_{AZ} x_{AZ} + W_{BZ} x_{BZ} = 0.5 + 0.9(0.7892) + 0.9(0.8176) = 1.9461$$

- then

$$f(\text{net}_z) = \frac{1}{1 + e^{-1.9461}} = 0.8750 = X_Z$$

- Value 0.8750 is the output from Neural Network on first pass
- Represents the predicted value for the target variable for the first given observation

Sigmoid Activation Function

- Sigmoid function f(x) combines nearly linear, curvilinear, and nearly constant behavior depending on input value
  - See graph for Sigmoid function is Figure 9.3 below
- Function nearly linear for domain values $-1 < x < 1$
- Becomes curvilinear as values move away from center
- At extreme values, f(*x*) is nearly constant
- Moderate increments in *x* produce variable increase in f(*x*), depending on the location of *x (moderate increase near the center and tiny increase near the extremes).*
- Sigmoid function is sometimes called "Squashing Function", since it takes any real-valued input and returns values [0, 1]

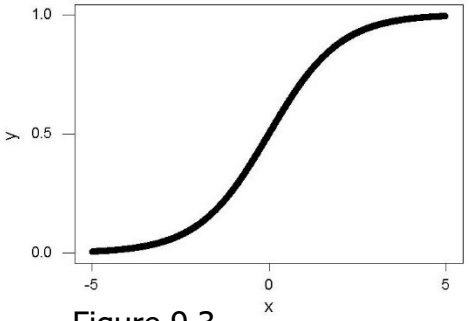


Figure 9.3

Back-Propagation

- Neural Networks are supervised learning method
- Require a large training set of complete records, including the target variable
- As each observation from the training set is passed through the network, an output value is produced from the output node (assuming we have only one output node)
- The output value is compared to the actual value of target variable
- (Actual – Output) = Error
- Prediction error analogous to residuals in regression models
- Most networks use Sum of Squared Errors (SSE) to measure how well predictions fit target values

$$SSE = \sum_{\text{Re}cords} \sum_{OutputNodes} (actual - output)^2$$

Back-Propagation (*cont'd*)

- SSE is calculated as the squared prediction errors summed over all output nodes, and all records in the training data set
- The problem is to construct model weights that minimize SSE
- Weights are analogous to the parameters of a regression model.
- Actual values that minimize SSE are unknown
- We should estimate these weights using the given the data set
- Unlike least-squares regression, no closed-form solution exists for minimizing SSE due to the nonlinear nature of the sigmoid functions spread through the network.

Gradient Descent Method

- Gradient Descent Method is an optimization method used to determine the set of weights that minimize SSE
- Given a set of *m* weights **w** = $w_1, w_2, ..., w_m$ in network model, find values for weights that, together, minimize SSE
- Gradient Descent method determines the direction to adjust weights in order to decreases SSE
- Gradient of SSE with respect to the vector of weights **w**, is vector derivative:

$$\nabla SSE(w) = \left[\frac{\partial SSE}{\partial w_0}, \frac{\partial SSE}{\partial w_1}, \ldots, \frac{\partial SSE}{\partial w_m} \right]$$

Gradient Descent Method (*cont'd*)

- Gradient Descent is illustrated using single weight w_1
- Figure plots SSE error against range of values for w_1
- We prefer values of w_1 that would minimize SSE
- Optimal value for w_1 is w^*_1

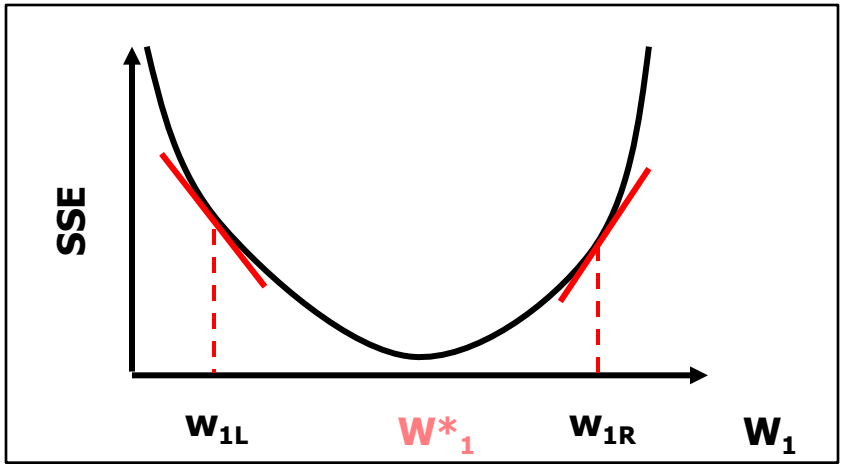


Figure 9.4

Gradient Descent Method (*cont'd*)

- We like to develop a rule defining movement from current w_1 to optimal value w^*_1

$$w_{NEW} = w_{CURRENT} + \Delta w_{CURRENT}$$

where

$\Delta w_{CURRENT}$ is change in current location w

- If current weight near w_{1L}, increasing w approaches w^*_1
- If current weight near w_{1R}, decreasing w approaches w^*_1
- Gradient of SSE with respect to weight $w_{CURRENT}$, is the slope of SSE curve at $w_{CURRENT}$
- If value $w_{CURRENT}$ is close to w_{1L}, the slope is negative
- If value $w_{CURRENT}$ is close to w_{1R}, the slope is positive

Gradient Descent Method (*cont'd*)

- Hence, the direction for adjusting $w_{CURRENT}$ is the negative of the derivative of SSE at $w_{CURRENT}$

$$- \left(\frac{\partial SSE}{\partial w_{CURRENT}}\right)$$

- In figure 9.4, to adjust, suppose we use the magnitude of the derivative of SSE at $w_{CURRENT}$
  - When the curve is steep, the adjustment is large (large slope)
  - When the curve is nearly flat, the adjustment is small (less slope)
- Finally, in general,

$$\Delta w_{CURRENT} = -\eta\left(\frac{\partial SSE}{\partial w_{CURRENT}}\right)$$

where Learning Rate (Greek "eta") has values in [0, 1]

Back-Propagation Rules

- Back-propagation percolates (spreads) prediction error for a record back through the network
- Partitioned responsibility for prediction error is assigned to various connections
- Weights of connections are adjusted to decrease the error using Gradient Descent Method
- Using sigmoid activation function and gradient decent, Back-propagation rules are defined as (Mitchell)

$$w_{ij,NEW} = w_{ij,CURRENT} + \Delta w_{ij}$$

where

$$\Delta w_{ij} = \eta \delta_j x_{ij}$$

η = learning rate

x_{ij} = signifies ith input to node j

δ_j = represents responsibility for a particular error belonging to node j

Back-Propagation Rules (*cont'd*)

- Error responsibility computed using partial derivative of the sigmoid function with respect to net$_j$
- Values take one of two forms

$$\delta_j = \begin{cases} \text{output}_j(1-\text{output}_j)(\text{actual}_j - \text{output}_j) & \text{for output layer nodes} \\ \text{output}_j(1-\text{output}_j)\sum_{DOWNSTREAM} W_{jk}\delta_k & \text{for hidden layer nodes} \end{cases}$$

where $\sum_{DOWNSTREAM} W_{jk}\delta_k$

refers to the weighted sum of error responsibilities for the nodes downstream from (after) the particular hidden layer node.

- Rules above show why input values require normalization in [0,1]
  - Large input values xij would dominate weight adjustment
  - Error propagation would be overwhelmed, and learning stifled (held back)

Example of Back-Propagation

- Recall that first pass through network yielded *output* = 0.8750
- Assume actual target value = 0.8, and learning rate, η = 0.1
- Prediction error = 0.8 - 0.8750 = -0.075
- Neural Networks use stochastic (or *online*) back-propagation, in which the weights are updated after each record processed by network
- Adjusting the weights using back-propagation shown next

- Error responsibility for Node Z, an output node, found first

$$\delta_Z = \text{output}_Z(1-\text{output}_Z)(\text{actual}_Z - \text{output}_Z) =$$
$$0.875(1-0.875)(0.8-0.875) = -0.0082$$

Example of Back-Propagation (*cont'd*)

- Now adjust "constant" weight w_{0Z} using rules

$$\Delta W_{0Z} = \eta \delta_Z (1) = 0.1(-0.0082)(1) = -.00082$$
$$w_{0Z,NEW} = w_{0Z,CURRENT} + \Delta w_{0Z} = 0.5 - 0.00082 = 0.49918$$

- Move upstream (backward) to Node A, a hidden layer node
- Only node downstream (forward) from Node A is Node Z

$$\delta_A = \text{output}_A (1 - \text{output}_A) \sum_{DOWNSTREAM} W_{jk} \delta_j$$
$$= 0.7892(1 - 0.7892)(0.9)(-0.0082) = -0.00123$$

Example of Back-Propagation (*cont'd*)

- Adjust weight w_{AZ} using back-propagation rules

$$\Delta W_{AZ} = \eta \delta_Z (OUTPUT_A) = 0.1(-0.0082)(0.7892) = -0.000647$$
$$w_{AZ,NEW} = w_{AZ,CURRENT} + \Delta w_{AZ} = 0.9 - 0.000647 = 0.899353$$

- Connection weight between Node A and Node Z adjusted from 0.9 to 0.899353

- Next, Node B is hidden layer node
- Only node downstream from Node B is Node Z

$$\delta_B = \text{output}_B (1 - \text{output}_B) \sum_{DOWNSTREAM} W_{jk} \delta_j$$
$$= 0.8176(1 - 0.8176)(0.9)(-0.0082) = -0.0011$$

Example of Back-Propagation (*cont'd*)

- Adjust weight w_{BZ} using back-propagation rules

$$\Delta W_{BZ} = \eta \delta_Z (OUTPUT_B) = 0.1(-0.0082)(0.8176) = -0.00067$$

$$w_{BZ,NEW} = w_{BZ,CURRENT} + \Delta w_{BZ} = 0.9 - 0.0.00067 = 0.89933$$

- Connection weight between Node B and Node Z adjusted from 0.9 to 0.89933

- Similarly, application of back-propagation rules continues to input layer nodes
- Weights $\{w_{1A}, w_{2A}, w_{3A}, w_{0A}\}$ and $\{w_{1B}, w_{2B}, w_{3B}, w_{0B}\}$ updated by process (left as an exercise)
- The adjusted weights result in a smaller prediction error (left as an exercise).

Example of Back-Propagation (*cont'd*)

- Now, all network weights in model are updated
- Each iteration based on a single record from data set

- Summary
  - Network calculated predicted value for target variable
  - Prediction error derived
  - Prediction error percolated back through network
  - Weights adjusted to generate smaller prediction error
  - Process repeats record by record

Termination Criteria

- Many passes of back propagation through data set are performed before termination criterion is met
- Constantly adjusting weights to reduce prediction error
- When to terminate?

- Stopping criterion may be computational "clock" time or number of passes through the data
  - Short training times likely result in poor model

- Terminate when SSE is reduced to a threshold level
  - Neural Networks are prone to overfitting
  - Overfitting: Memorizing patterns rather than generalizing

Termination Criteria (*cont'd*)

- Cross-Validation Termination Procedure (Popular)
  - Retain portion of training set as "holdout" validation set
  - Train network on remaining data record by record
  - After processing each record of the training set, apply weights learned to all records of validation set
  - Monitor two sets of weights at all times:
    - "Current" weights produced by the training data so far, and "Best" weights (among the ones produced during the training so far) with minimum SSE on validation set
  - Terminate algorithm when current weights has significantly greater SSE than best weights on validation set

Termination Criteria (*cont'd*)

- Regardless of stopping criterion used, Neural Network is not guaranteed to arrive at global minimum for SSE
- Algorithm may become stuck in local minimum
- Results in good, but not optimal solution
- Not necessarily an insuperable problem
  - (Solution 1) Multiple networks trained using different starting weights, and the best model from group chosen as "final"
  - (Solution 2) Stochastic back-propagation method acts to prevent getting stuck in local minimum, e.g., random element introduced to gradient descent
  - (Solution 3) Momentum term may be added to back-propagation algorithm as discussed next.

Learning Rate

- Recall Learning Rate (Greek "eta") is a constant

$$0 < \eta < 1, \text{ where}$$
$$\eta = \text{learning rate}$$

- It helps adjust the weights toward a global minimum for SSE

- Small Learning Rate
  - With small learning rate, weight adjustments are small
  - Network takes unacceptably long time to converge to a solution
- Large Learning Rate
  - Suppose algorithm is close to the optimal solution, with large learning rate, network likely to "overshoot" the optimal solution

Learning Rate *(cont'd)*

- In figure below, w* is the optimum weight for w, which has current value $w_{CURRENT}$
- According to Gradient Descent Rule, $w_{CURRENT}$ will be adjusted in the direction of w*
- Learning rate acts as a multiplier to formula $\Delta w_{CURRENT}$
- Large learning rate may cause w_{NEW} to jump past w*
- w_{NEW} may be farther away from w* than $w_{CURRENT}$

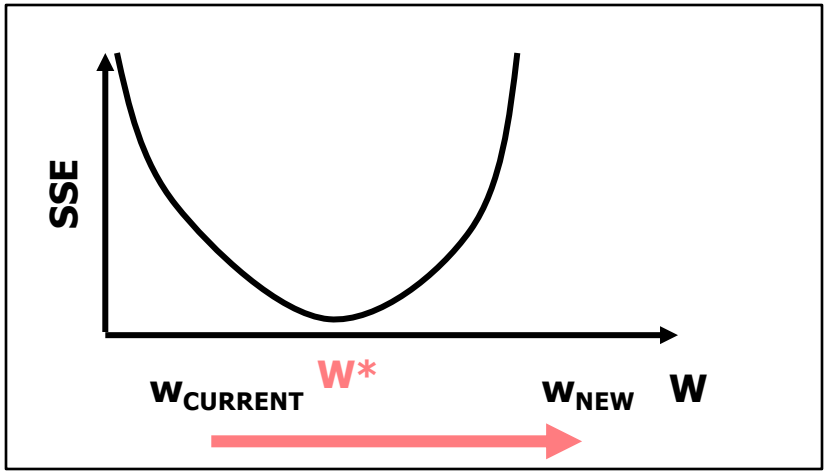


Figure 9.5

Learning Rate *(cont'd)*

- Next adjusted weight value will be on the opposite side of w*
- Next adjustment will again overshoot w*,
- Leads to oscillation (back and forth) between two "slopes"
- Network never settles down to minimum between them

- Solution: Adjust Learning Rate as Training Progresses
  - Learning rate initialized with a large value so that the network quickly approaches general vicinity of the optimal solution.
  - As network begins to converge, learning rate will be gradually reduced to avoids overshooting the minimum.

Application of Neural Network Modeling

- Insightful Miner models *adult* data set using a neural network
- Model uses a training set of 25,000 cases, and one hidden layer with 8 nodes
- Algorithm iterated 47 epochs (runs through the data set) before terminating
- Resulting network model shown

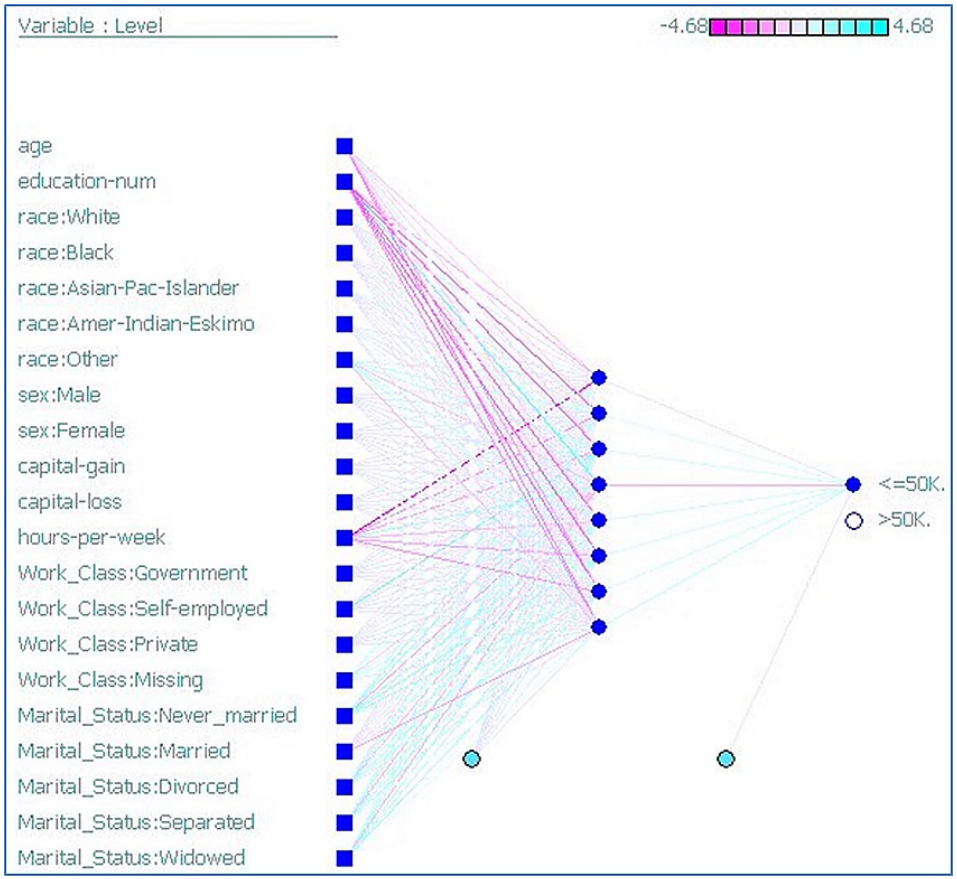


Figure 9.8

Application of Neural Network Modeling (*cont'd*)

- Input nodes appear on left shown as squares
- For the categorical variables, there is one input node per class.
- Eight hidden nodes in center shown as circles
- The light gray circles represent the constant inputs.
- Includes single output node, whether *income* <= $50,000
- Figure 9.9 in next slide shows network weights from model
- Input nodes numbered 1 – 9, hidden nodes 22 – 29
- Weight from input node *Age* (1) to topmost hidden node (22) is -0.97
- The lower section of Figure 9.9 displays the weights from the hidden nodes to the output node.

Application of Neural Network Modeling (*cont'd*)

Weights

To/From	1	2	3	4	5	6	7	8	9
22	-0.97	-1.32	-0.18	-0.51	0.69	0.13	-0.25	-0.33	0.30
23	-0.70	-2.97	-0.12	0.34	0.43	0.50	1.03	-0.29	-0.10
24	-0.70	-2.96	-0.24	0.05	0.16	0.46	1.15	-0.16	-0.07
25	0.74	2.86	0.22	0.41	-0.03	-0.59	-1.05	0.18	0.14
26	-0.84	-2.82	-0.23	0.02	-0.16	0.62	1.06	-0.22	-0.20
27	-0.68	-2.89	-0.18	-0.03	-0.03	0.50	1.07	-0.24	-0.12
28	-1.68	-2.54	-0.43	-0.09	0.04	0.54	0.88	-0.18	-0.26
29	-2.11	-1.95	0.01	0.34	0.04	-0.75	-1.16	-0.03	0.38

Weights

To/From	22	23	24	25	26	27	28	29	0
30	0.18	0.59	0.69	-1.40	0.77	0.76	0.74	1.06	-0.08

Figure 9.9

Application of Neural Network Modeling (*cont'd*)

- Estimated prediction accuracy of model is 82%
- Since over 75% of records have *income* <= $50,000, simple prediction for a person will have baseline accuracy of 75%.

Figure 9.10