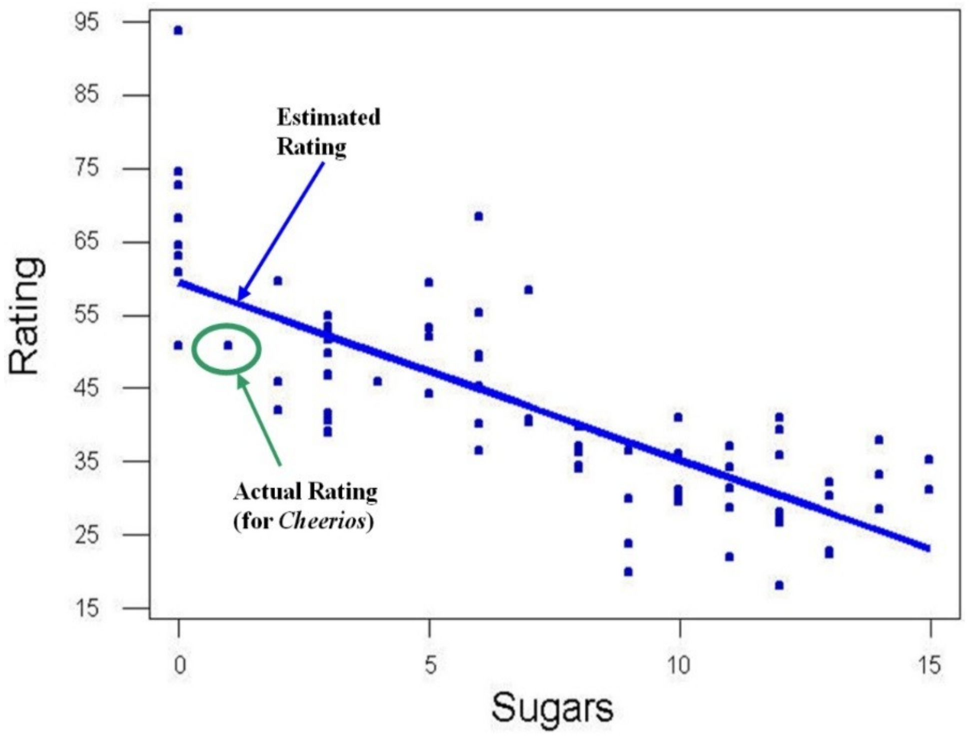
Logistic Regression and Neural Networks

Linear Regression Analysis

- Scatter plot of the nutritional rating vs sugar content, along with the least-squares regression line
- The regression equation is , where:
  - is the estimated value of the response variable
  - is the y-intercept of the regression line
  - is the slope of the regression line
  - and , together, are called the regression coefficie

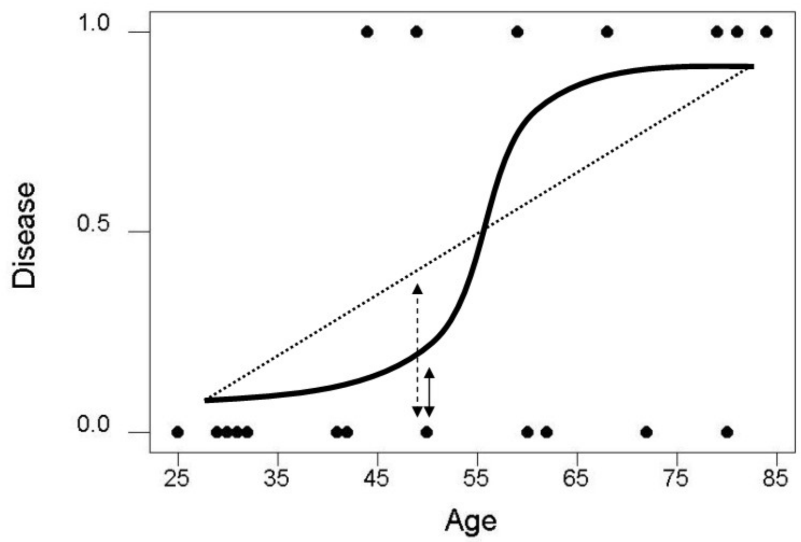


Logistic Regression

- Linear regression is not appropriate where response variable is ***categorical***
- Alternatively, **Logistic Regression** method describes relationship between **categorical** response and set of predictors
  - Specifically, we explore applications with dichotomous response
  - **Example:** Suppose researchers interested in **potential relationship between patient age and presence/absence of disease**
  - Data set includes 20 patients

Non-Linear Relationships

- Plot shows *least squares regression* line (straight) and *logistic regression* line (curved) for *disease* on *age*
- Least squares, assume linear relationship between variables
- In contrast, logistic regression line assumes non-linear relationship between predictor and response
- Patient 11 estimation errors (vertical lines) shown
- Patient 11's estimation error greater for linear regression versus logistic regression
- For this point, and many others, logistic regression does a better job of estimating *disease*



Sigmoid Function

- How is logistic regression line derived?
- *E(Y|x)* is expected value of response, for given predictor value
  - Equals the conditional mean of Y, given x
- Recall linear regression, where response random variable defined as:

$$Y = \beta_0 + \beta_1 x + \varepsilon$$

- Since *ε* (error term) has a mean = 0, the conditional mean of Y, given x *E(Y|x)* equals:

$$E(Y \mid x) = \beta_0 + \beta_1 x$$

Sigmoid Function (continued)

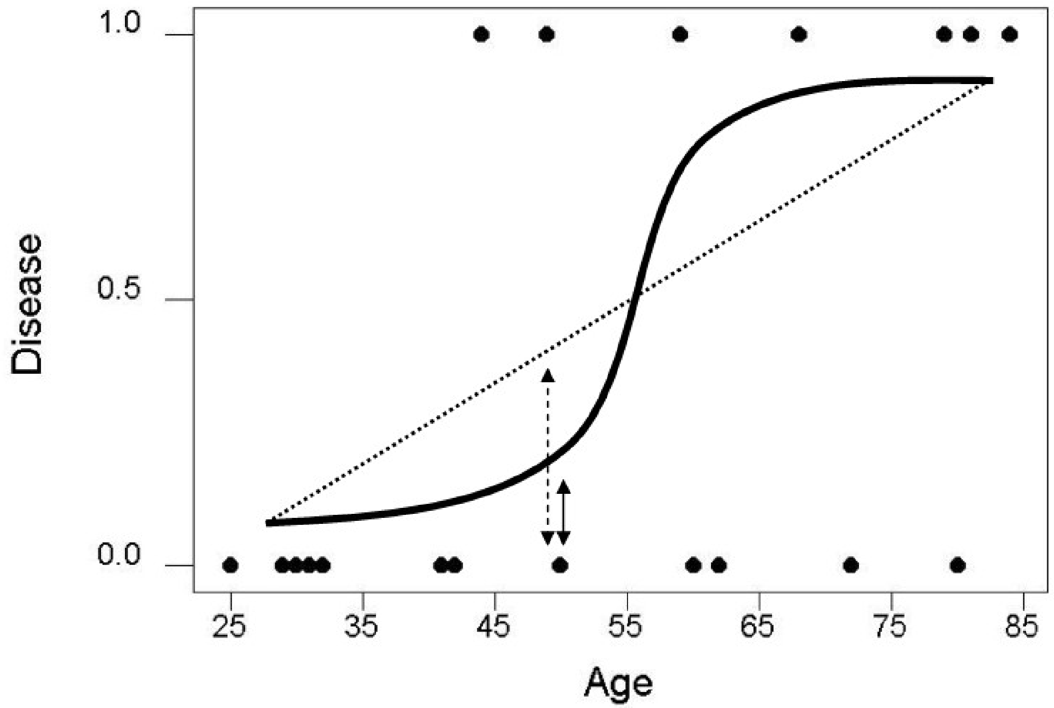
- Denote *E(Y|x)* as *π(x)*, where conditional mean for logistic regression takes the following form:

$$\pi(x) = \frac{e^{\beta_0 + \beta_1 x}}{1 + e^{\beta_0 + \beta_1 x}}$$

- Function forms s-shaped (sigmoidal) curves, which are non-linear
- Logistic function models binary data well, because of simplicity and interpretability

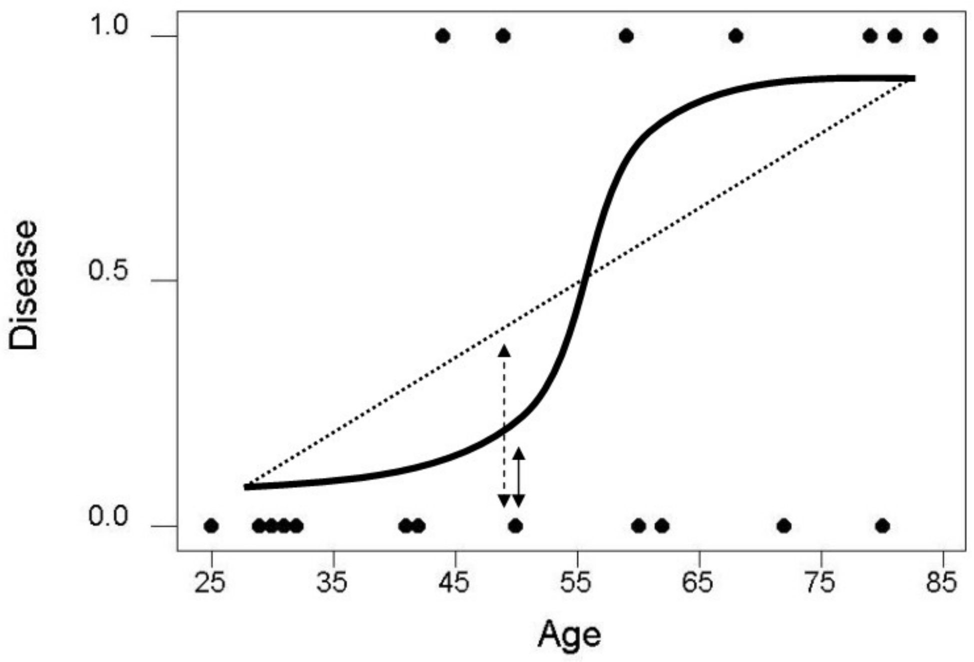
Logistic Regression and Error

- $\pi(x)$ interpreted as probability *disease* (positive outcome) present for records $X = x$
- $1 - \pi(x)$ interpreted as probability *disease* (positive outcome) not present for records $X = x$
- Recall linear regression error term ε normally distributed, with mean = 0 and constant variance
- However, assumptions regarding error term is different for logistic regression



Logistic Regression and Error

- Because response dichotomous, errors take one of two forms:
- $Y = 1$ (disease present)
  - Occurs with probability π(x), probability response positive
  - $\varepsilon = 1 - \pi(x)$ represents vertical distance between point $Y = 1$ and curve $\pi(x)$ below, for $X = x$
- Y = 0 (disease not present)
  - Occurs with probability 1 – π(x), probability response negative
  - $\varepsilon = 0 - \pi(x) = -\pi(x)$, which represents vertical distance between point $Y = 0$ and curve $\pi(x)$ above, for $X = x$



Logit Transformation

- Variance of *ε = π(x)·(1 – π(x))*, variance of binomial distribution
- Therefore, logistic regression response *Y = π(x) + ε* assumed to follow binomial distribution with probability success = *π(x)*
- Transformation for logistic regression, *logit transformation*, defined as:

$$g(x) = \ln\left[\frac{\pi(x)}{1 - \pi(x)}\right] = \beta_0 + \beta_1 x \qquad \pi(x) = \frac{e^{g(x)}}{1 + e^{g(x)}}$$

- Includes useful properties for linearity, continuity, and ranges from positive to negative infinity

Maximum Likelihood Estimation

- Linear regression has closed form solution
- However, closed form solution does not exist for estimating logistic regression coefficients
- Therefore, *maximum likelihood estimation* finds parameter estimates
- Likelihood function is function of β_i parameters, which expresses probability of observed data, *x*

$$l(\boldsymbol{\beta} \mid x), where\ \boldsymbol{\beta} = \beta_0, \beta_1, \ldots, \beta_m$$

- Maximum likelihood estimators determined by finding values for β_i, which maximize likelihood function
- These are parameters most likely favored by data

Maximum Likelihood Estimation

- Probability of <u>positive</u> response, given data:

$$\pi(x) = P(Y=1|x)$$

- Probability of <u>negative</u> response, given data:

$$1 - \pi(x) = P(Y=0|x)$$

- Now, observations where response are positive ($X_i=x_i$, $Y_i=1$) contribute $\pi(x)$ to likelihood, while those with negative response ($X_i=x_i$, $Y_i=0$) contribute $1 - \pi(x)$
- Since observations assumed **independent** and take on values $Y_i = 0$ or 1, likelihood function expressed as product of terms:

$$l(\boldsymbol{\beta}|x) = \prod_{i=1}^{n} \left[\pi(x_i)\right]^{y_i} \left[1 - \pi(x_i)\right]^{1-y_i}$$

Log-likelihood Form

- Log-likelihood form $L(\boldsymbol{\beta}|x)$ more computationally tractable:

$$L(\boldsymbol{\beta}|x) = \ln\left[l(\boldsymbol{\beta}|x)\right] = \sum_{i=1}^{n}\left\{y_i \ln\left[\pi(x_i)\right] + (1 - y_i)\ln\left[1 - \pi(x_i)\right]\right\}$$

- Finally, maximum likelihood estimates found by *differentiating $L(\beta \mid x)$ with respect to each parameter β_i, and setting result = 0*
- Iterative weighted least squares method applied, since closed form solutions for differentiations are non-existent

Logistic Regression Example

Logistic Regression Table

Predictor	Coef	StDev	Z	P	Odds Ratio	95% CI Lower	95% CI Upper
Constant	-4.372	1.966	-2.22	0.026			
Age	0.06696	0.03223	2.08	0.038	1.07	1.00	1.14

Log-Likelihood = -10.101

Test that all slopes are zero: G = 5.696, DF = 1, P-Value = 0.017

- Coefficients (<u>maximum likelihood estimates</u>) of unknown parameters β_0 *and* β_1, given as
  - $b_0 = -4.372$ and
  - $b_1 = 0.06696$, respectively

Logistic Regression Example

Logistic Regression Table

Predictor	Coef	StDev	Z	P	Odds Ratio	95% CI Lower	95% CI Upper
Constant	-4.372	1.966	-2.22	0.026			
Age	0.06696	0.03223	2.08	0.038	1.07	1.00	1.14

Log-Likelihood = -10.101

Test that all slopes are zero: G = 5.696, DF = 1, P-Value = 0.017

- Here, *π(x)* estimated as:

$$\hat{\pi}(x) = \frac{e^{\hat{g}(x)}}{1 + e^{\hat{g}(x)}} = \frac{e^{-4.372 + 0.06696\,(age)}}{1 + e^{-4.372 + 0.06696\,(age)}}$$

- With estimated logit:

$$\hat{g}(x) = -4.372 + 0.06696(age)$$

Logistic Regression Example

Logistic Regression Table

Predictor	Coef	StDev	Z	P	Odds Ratio	95% CI Lower	95% CI Upper
Constant	-4.372	1.966	-2.22	0.026			
Age	0.06696	0.03223	2.08	0.038	1.07	1.00	1.14

Log-Likelihood = -10.101

Test that all slopes are zero: G = 5.696, DF = 1, P-Value = 0.017

- Using these equations, estimated probability disease present in patient, given their age derived
- Example: Estimate probability disease present in particular patient, age = 50

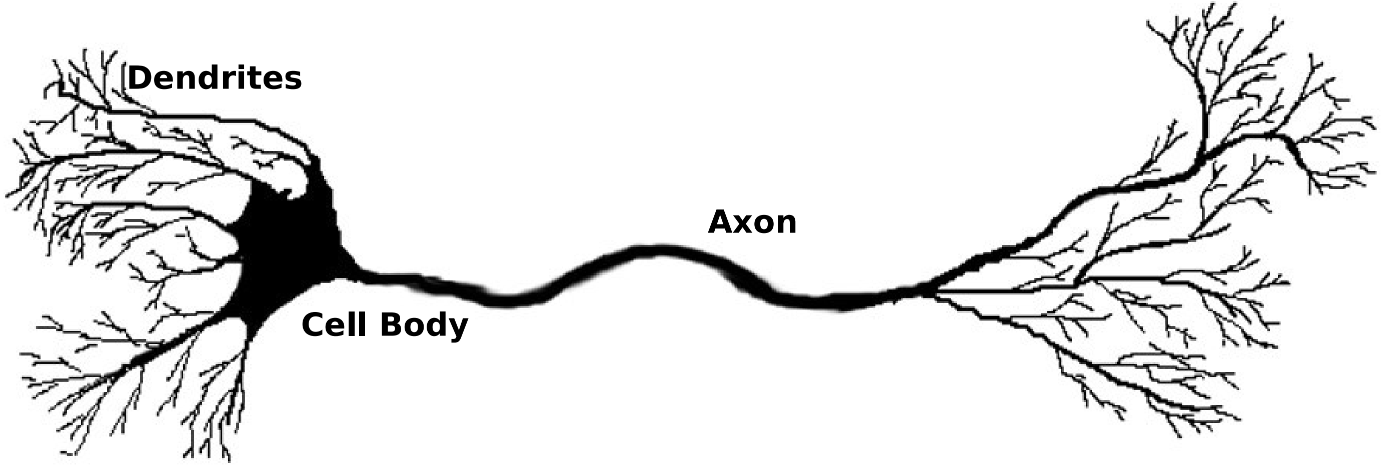
$$\hat{g}(x) = -4.372 + 0.06696(50) = -1.024 \qquad \hat{\pi}(x) = \frac{e^{\hat{g}(x)}}{1+e^{\hat{g}(x)}} = \frac{e^{-1.024}}{1+e^{-1.024}} = 0.26$$

- Estimated probability 50-year old patient has disease = 26%, with probability disease not present = 74%

Neural Networks

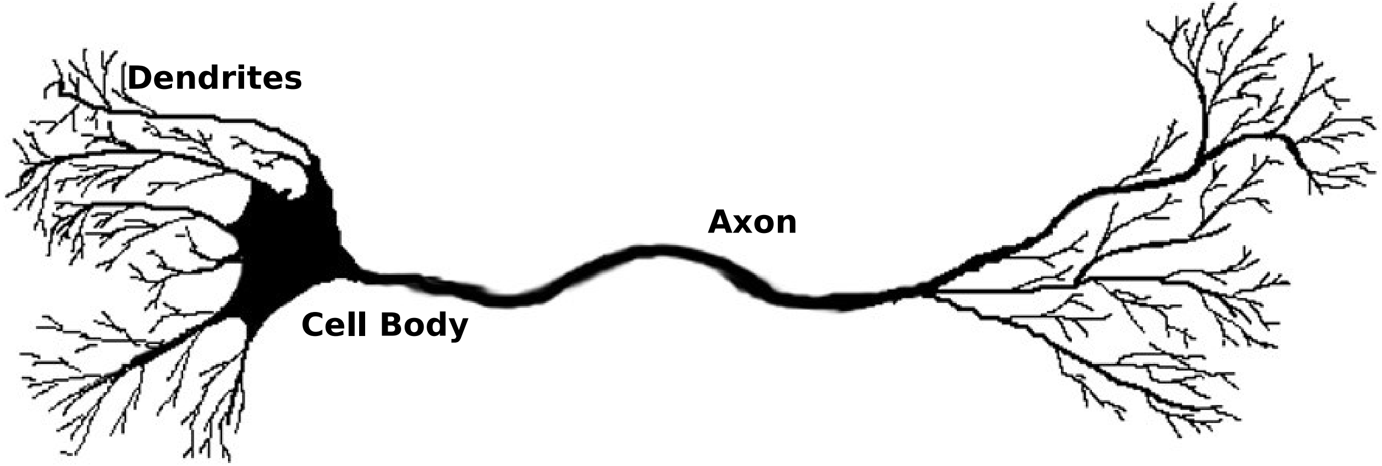
Neural Networks

- Complex learning systems recognized in animal brains
- Single neuron has a simple structure
- Interconnected sets of neurons perform complex learning tasks
- Artificial Neural Networks attempt to *replicate* non-linear learning found in nature



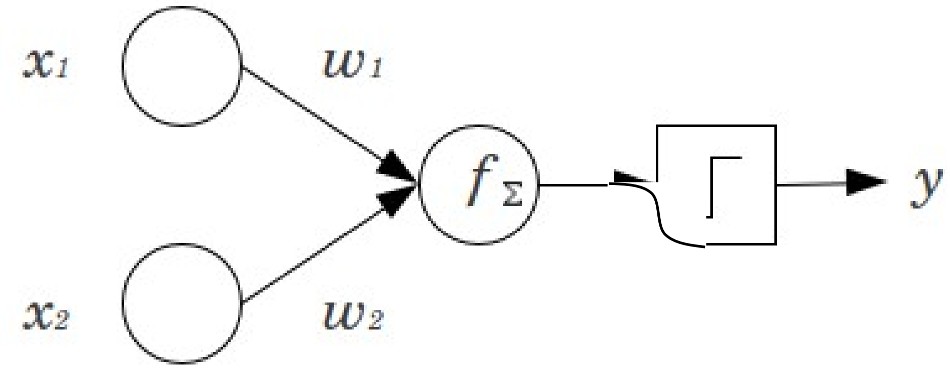
Neural Networks

- Quite **robust** with respect to noisy data
- Can learn and work around erroneous data
- Results opaque to human interpretation
- Often require <u>long</u> training times

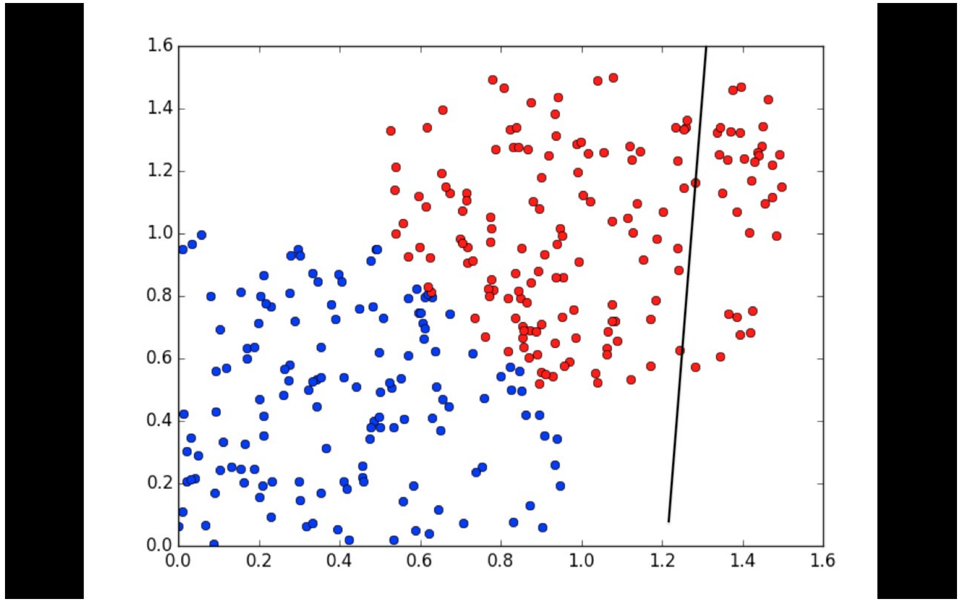


Before Neural Networks: The Perceptron

- Linear classifier
- Find decision boundary between classes
- Steps:
  - Initialize *w = 0*
  - For each *x* predict *positive* *iff* $w_t \bullet x > 0$
  - On mistake update w
    - Positive: $w_{t+1} \leftarrow w_t + x$
    - Negative: $w_{t+1} \leftarrow w_t - x$
  - $t \leftarrow t + 1$



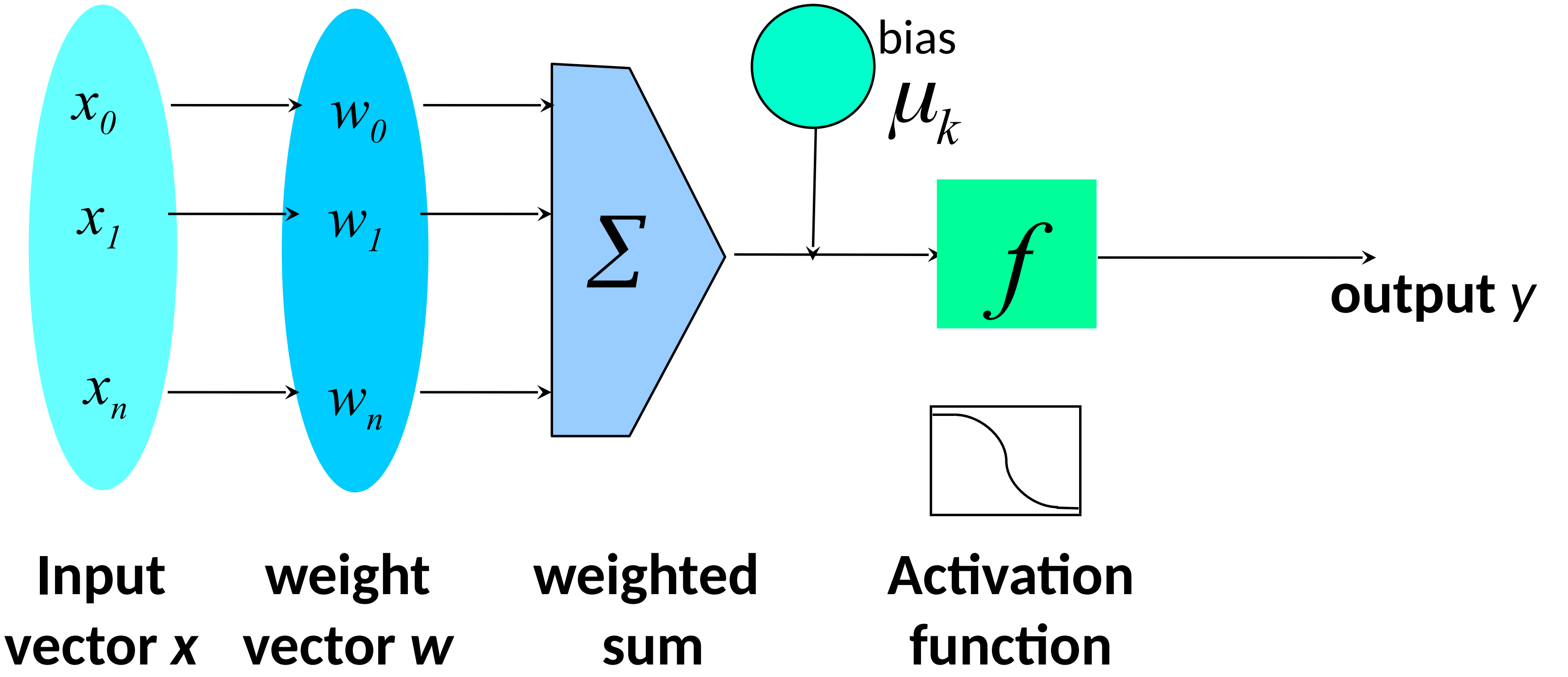
$$F(x) = \begin{cases} 1, if\ \boldsymbol{wx} + b > 0 \\ 0, otherwise \end{cases}$$



Classification by Back-propagation

- Back-propagation: A **neural network** learning algorithm
- Started by psychologists and neurobiologists to develop and test computational analogues of neurons
- A neural network: A set of connected input/output units where each connection has a ***weight*** associated with it
- During the learning phase, the **network learns by** ***adjusting the weights*** so as to be able to predict the correct class label of the input tuples
- Also referred to as **connectionist learning** due to the connections between units

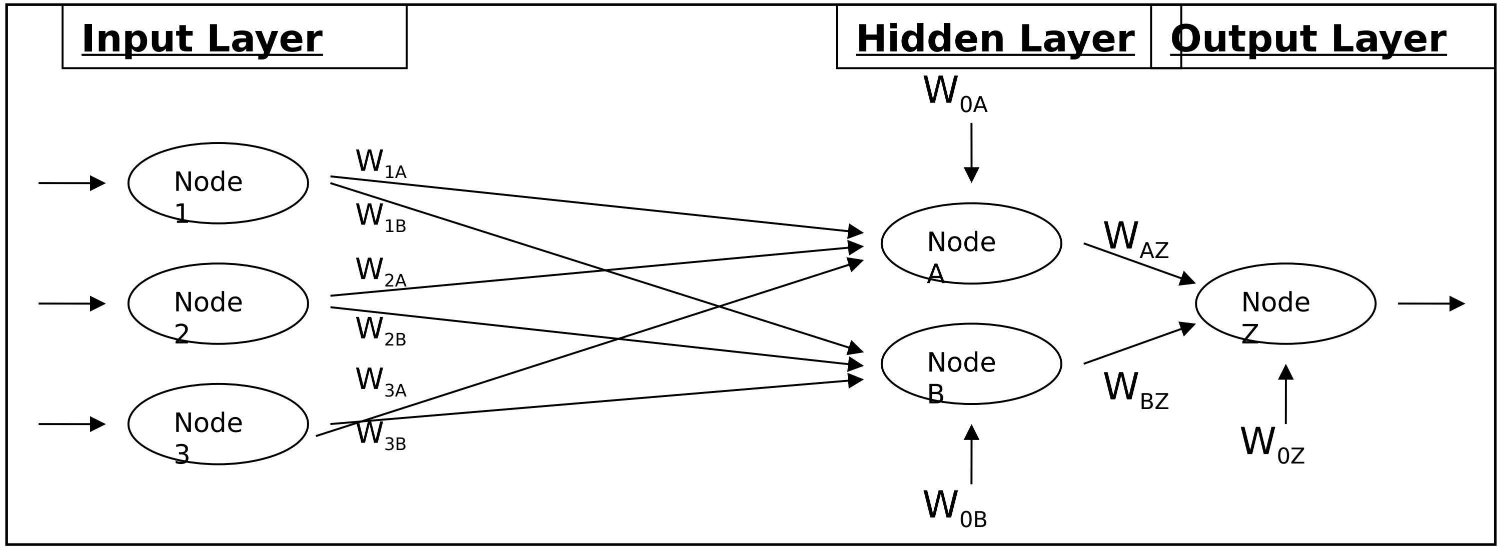
Neuron: A Hidden/Output Layer Unit



- An *n*-dimensional input vector ***x*** is mapped into variable *y* by means of the scalar product and *a nonlinear function mapping*
- The inputs to unit are outputs from the previous layer. They are multiplied by their corresponding weights to form a weighted sum, which is added to the bias associated with unit. Then *a nonlinear activation function* is applied to it.

Neural Network Components

- Neural Network consists of layered, feedforward, completely connected network of nodes
- Feedforward restricts network flow to single direction
- Flow does not loop or cycle
- Network composed of three or more layers



Neural Network Components

- Most networks have *Input, Hidden, Output* layers
- Network may contain more than one hidden layer
- Network is completely connected
- Each node in given layer, connected to every node in next layer
- Every connection has weight (W_{ij}) associated with it
- Weight values randomly assigned 0 to 1 by algorithm initially
- Number of input nodes dependent on number of predictors
- Number of hidden and output nodes configurable

Defining a Network Topology

- Decide the **network topology:** Specify *# of units in the input layer, # of hidden layers (if > 1), # of units in each hidden layer, and # of units in the output layer*
- If input values on different scales, **normalize** the input values for each attribute measured in the training tuples to [0.0, 1.0]: ***Min-max***
- One **input** unit per domain value, each initialized to 0
- **Output**, if for classification and more than two classes, one output unit per class is used
- Once a network has been trained and its accuracy is **unacceptable**, repeat the training process with *a different network topology* or a *different set of initial weights*

Input and Output Encoding

- Neural Networks require attribute values encoded to [0, 1]
- **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 are 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 Variables used when number of category values 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 <u>unordered</u> categorical values to [0, 1] range
- For example, attribute *Marital_Status* has values “Divorced”, “Married”, “Separated”, “Single”, “Widowed”, and “Unknown”
- 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 only aware of numeric values
- Naive to pre-encoded meaning of categorical values
- Results of network model may be meaningless

Input and Output Encoding (*cont'd*)

- **Output**
  - Neural Networks always return continuous values [0, 1]
  - Many classification problems have two outcomes
  - Solution uses **threshold** established *a priori* in single output node to separate classes
  - For example, target variable is "leave" or "stay"
  - Threshold value is "leave if output >= 0.67"
  - Single output node value = 0.72 classifies record as "leave"

Input and Output Encoding (*cont'd*)

- Single output nodes applicable when target classes <u>ordered</u>
- For example, classify elementary-level reading ability

- Define thresholds Classify
- "if 0.00 <= output < 0.25" "first-grade"
- "if 0.25 <= output < 0.50" "second-grade"
- "if 0.50 <= output < 0.75" "third-grade"
- "if output >= 0.75" "fourth-grade"

- 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
- Use ***1-of-n Encoding***, with single output node for each target class
- Network has six output nodes for values "Divorced", "Married", "Separated", "Single", "Widowed", and "Unknown"
- Output node with highest value chosen for classification
- Approach provides measure of confidence in classification
- Confidence is difference between highest and second-highest value in output nodes

Neural Network Layers

- Hidden Layer
  - How many nodes in hidden layer?
  - Large number of nodes increases complexity of model
  - Detailed patterns uncovered in data
  - Leads to overfitting, at expense of generalizability
  - Reduce number of hidden nodes when overfitting occurs
  - Increase number of hidden nodes when training accuracy unacceptably low
- Input Layer
  - Input layer accepts values from input variables
  - Values passed to hidden layer nodes
  - Input layer nodes lack detailed structure compared to hidden and output layer nodes

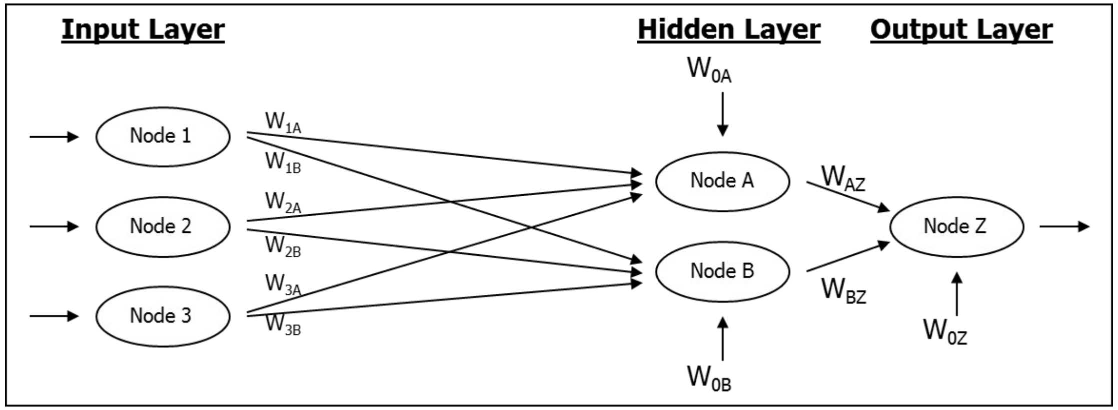
Nodes and Weights

- Combination function produces *linear combination* of node inputs and connection weights to single scalar value
- For a given node *j*:

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

- where
  - x_{ij} is *i*th input to node *j*
  - W_{ij} is weight associated with *i*th input node
  - and there are I + 1 inputs to node *j*
  - $x_1, x_2, \ldots, x_I$ are inputs from upstream nodes
  - x_0 is constant input value = *1.0*
  - Each input node has extra input $W_{0j}x_{0j} = W_{0j}$

Nodes and Weights



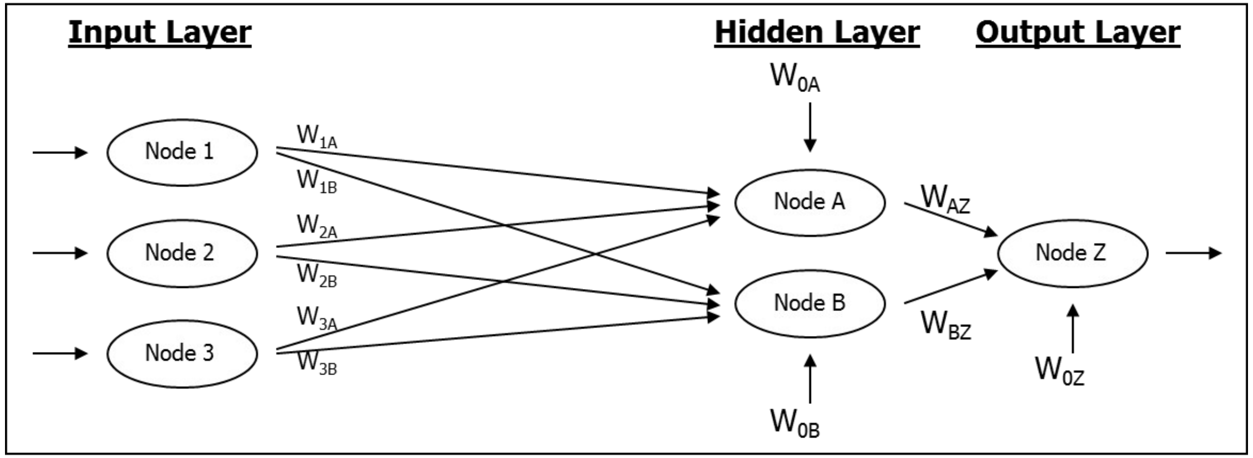
$x_0 = 1.0$	$W_{0A} = 0.5$	$W_{0B} = 0.7$	$W_{0Z} = 0.5$
$x_1 = 0.4$	$W_{1A} = 0.6$	$W_{1B} = 0.9$	$W_{AZ} = 0.9$
$x_2 = 0.2$	$W_{2A} = 0.8$	$W_{2B} = 0.8$	$W_{BZ} = 0.9$
$x_3 = 0.7$	$W_{3A} = 0.6$	$W_{3B} = 0.4$	

- The scalar value computed for hidden layer Node A equals

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

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

- For Node A, *netA* = 1.32 is the input to activation function
- This activation is analogous to <u>how neurons “fire” nonlinearly in biological organisms</u>



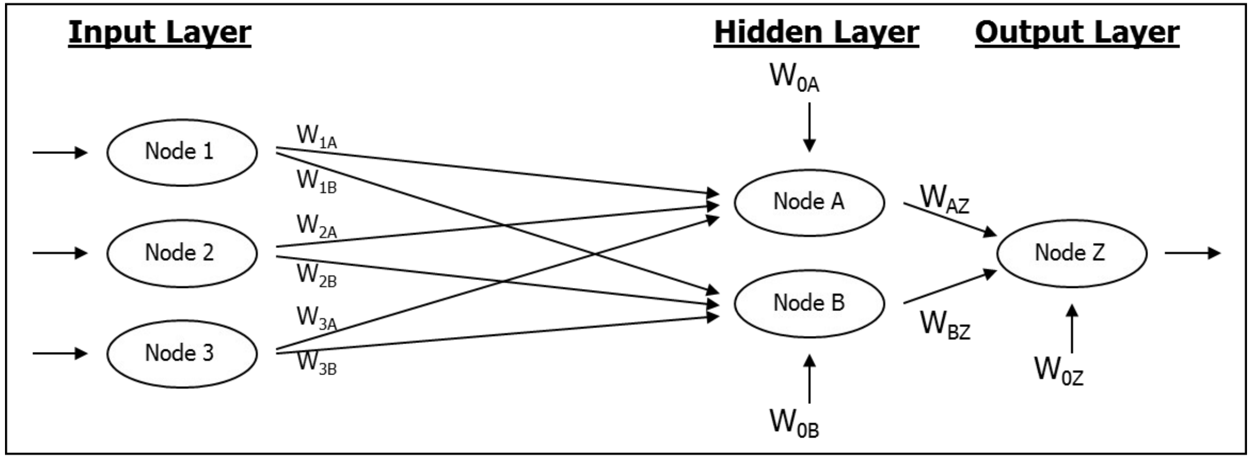
$x_0 = 1.0$	$W_{0A} = 0.5$	$W_{0B} = 0.7$	$W_{0Z} = 0.5$
$x_1 = 0.4$	$W_{1A} = 0.6$	$W_{1B} = 0.9$	$W_{AZ} = 0.9$
$x_2 = 0.2$	$W_{2A} = 0.8$	$W_{2B} = 0.8$	$W_{BZ} = 0.9$
$x_3 = 0.7$	$W_{3A} = 0.6$	$W_{3B} = 0.4$	

- Firing response not necessarily linearly related to increase in input stimulation
- Artificial Neural Networks model behavior using non-linear activation function
- Example: Sigmoid function

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

- In Node A, activation function takes *netA = 1.32* as input and produces output

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



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

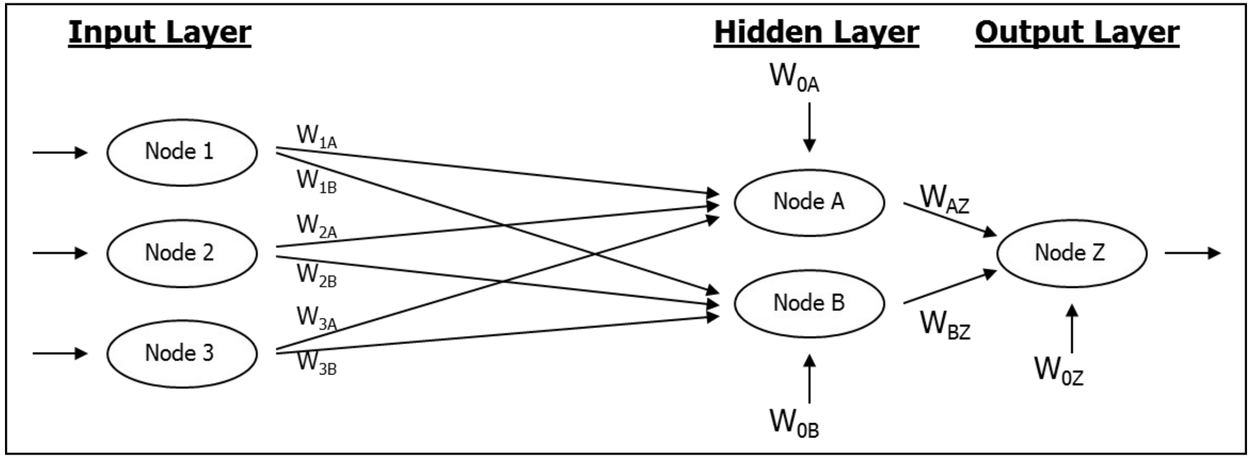
- Node A outputs 0.7892 along connection to Node B, and becomes component of *netZ*
- Before *netZ* is computed, contribution from Node B required

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

- then

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

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



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

- Inputs to Node *Z* not data attribute values
- Rather, outputs are from sigmoid function in 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

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

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

Back Propagation

- Neural Networks are supervised learning method
- Require target variable
- Each observation passed through network results in output value
- **Output value compared to actual value of target variable**
- *(Actual – Output) = Error*
- Prediction error analogous to residuals in regression models
- Most networks use *Sum of Squares (SSE)* to measure how well predictions fit target values

$$SSE = \sum_{Records} \sum_{OutputNodes} (actual - output)^2$$

Back Propagation

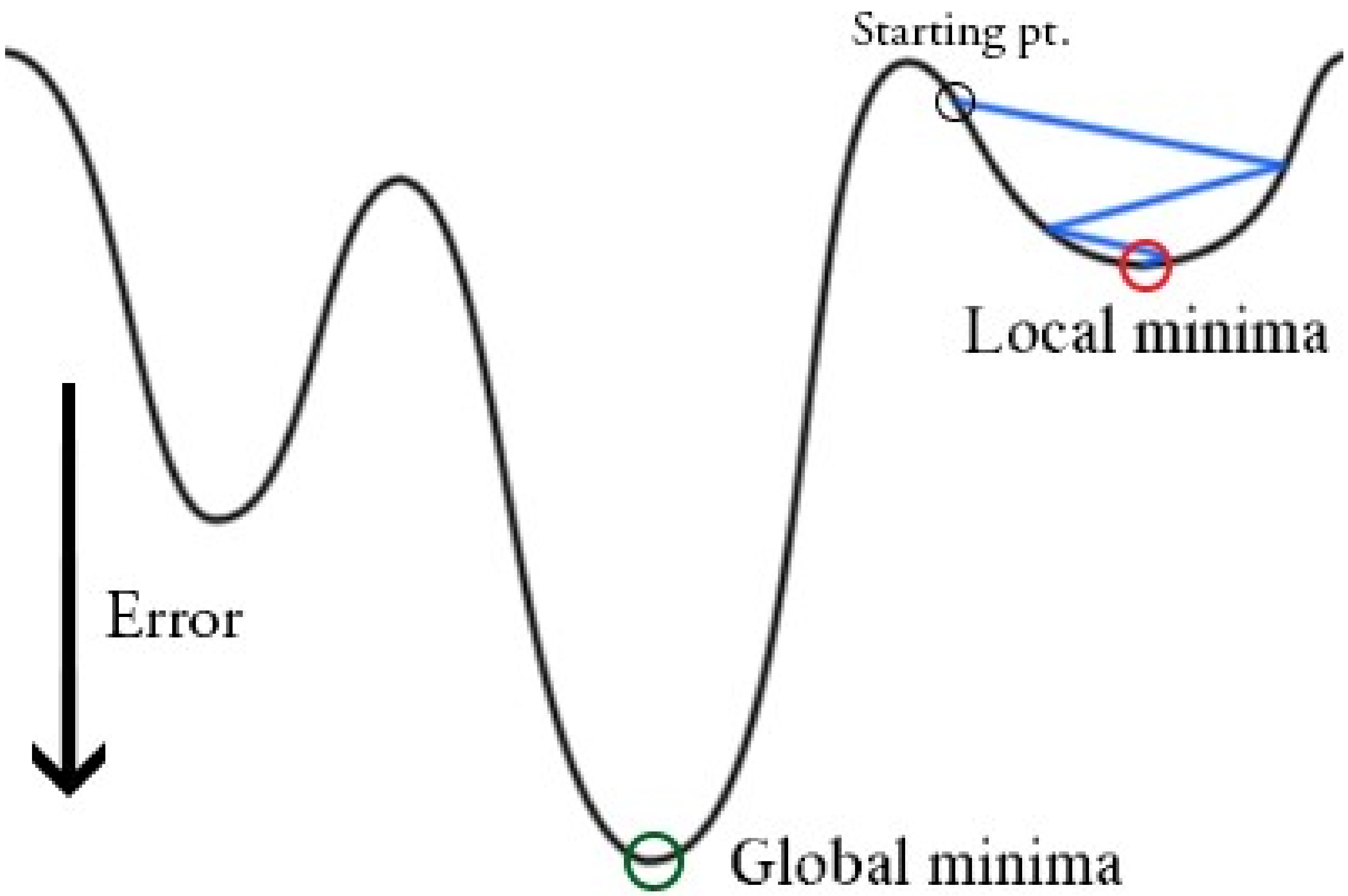
- Squared prediction errors summed over all output nodes, and all records in data set
- Model weights constructed that ***minimize SSE***
- Actual values that minimize SSE are unknown
- Weights estimated, given the data set
- Unlike least-squares regression, no closed-form solution exists for minimizing SSE

Gradient Descent

- Gradient Descent Method determines set of weights that minimize SSE
- Given a set of *m* weights $w = w_1, w_2, \ldots, w_m$ in network model
- Find values for weights that, together, ***minimize SSE***
- Gradient Descent determines direction to adjust weights, that decreases SSE
- Gradient of SSE, with respect to vector of weights ***w***, is vector derivative:

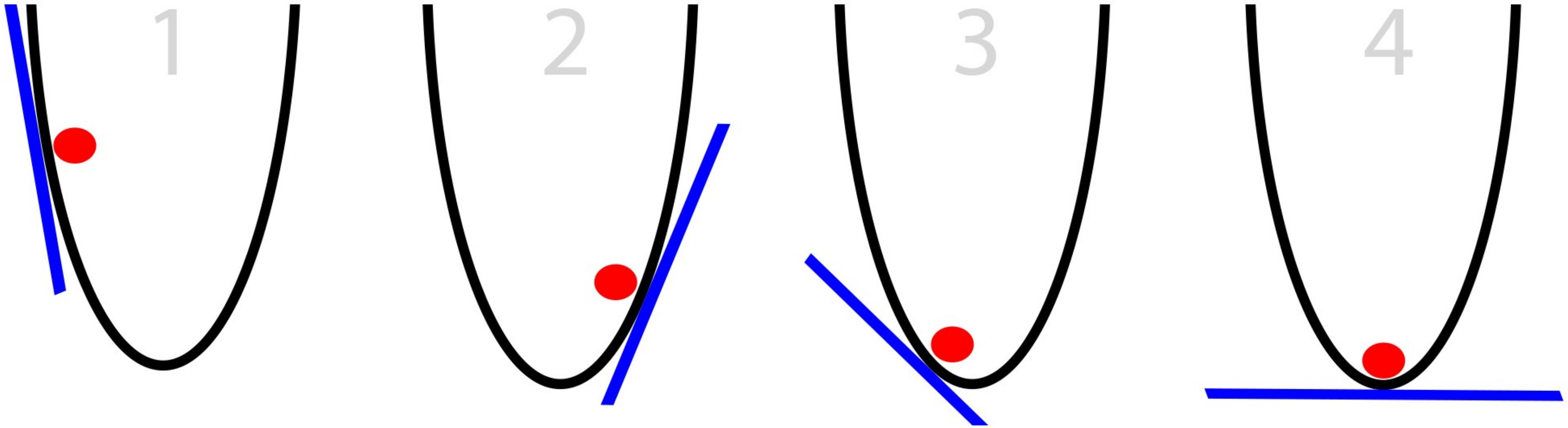
$$\nabla SSE(w) = \left[\frac{\partial SSE}{\partial w_0}, \frac{\partial SSE}{\partial w_1}, \ldots, \frac{\partial SSE}{\partial w_m} \right]$$

Gradient Descent



Gradient Descent

- Basic Intuition:
  - Calculate slope at current position
    - If slope is negative, move right
    - If slope is positive, move left
  - (Repeat until slope == 0)



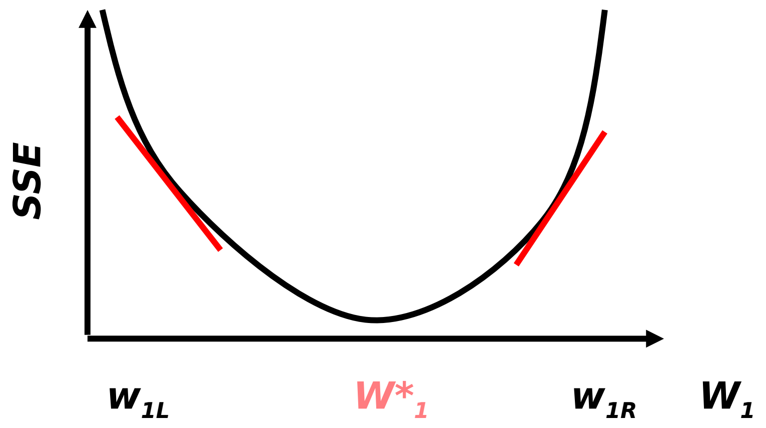
Gradient Descent

- Develop 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 slope of *SSE* curve at $w_{CURRENT}$
- Value $w_{CURRENT}$ close to w_{1L}, slope is negative
- Value $w_{CURRENT}$ close to w_{1R}, slope is positive

Gradient Descent

- **Direction** for adjusting $w_{CURRENT}$ is negative sign of derivative at SSE at $w_{CURRENT}$

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

- To adjust, use **magnitude** of the derivative of SSE at $w_{CURRENT}$
- **When curve steep, adjustment is large**
- **When curve nearly flat, adjustment is small**

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

- *Learning Rate* (Greek “eta”) has values [0, 1]

Back Propagation Rules

- Back-propagation percolates prediction error for record back through network
- Partitioned responsibility for prediction error assigned to various connections
- Weights of connections adjusted to decrease error, using Gradient Descent Method

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

where

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

η = learning rate

x_{ij} = signifies ith input to node j

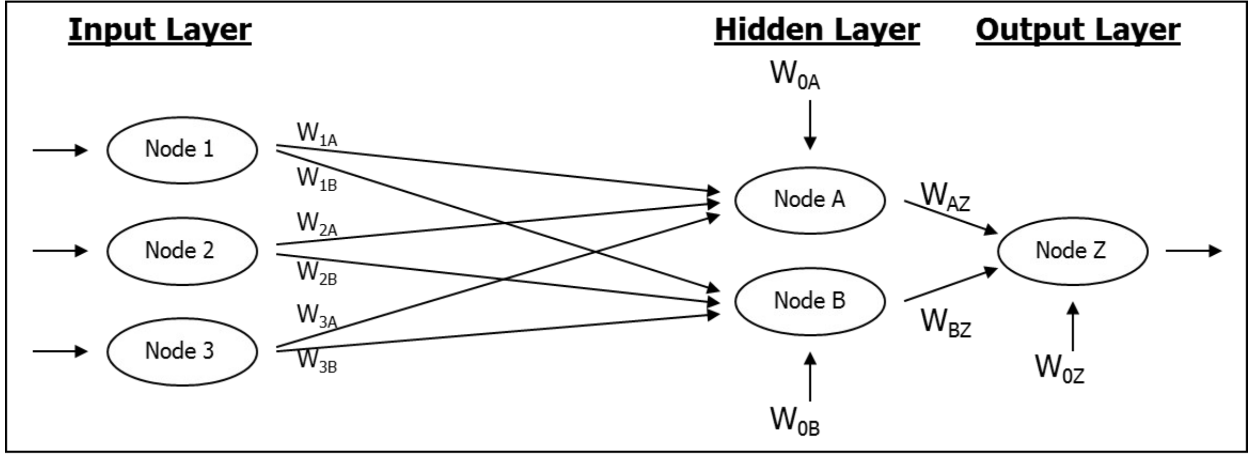
δ_j = represents responsibility for a particular error belonging to node j

Back Propagation Rules

- Error responsibility, , computed using partial derivative of the sigmoid function with respect to net_j
  - Values take one of two forms

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

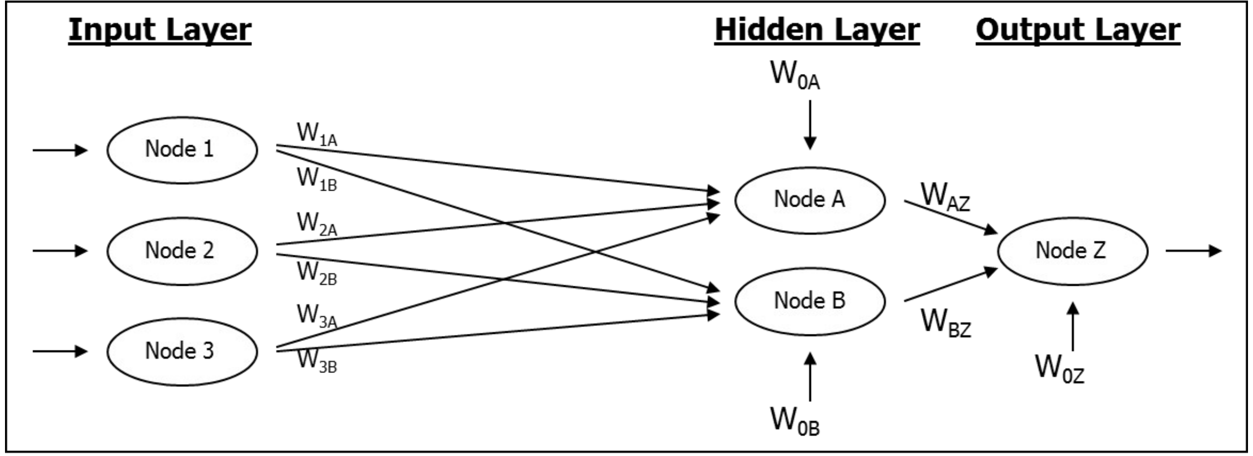
  - Where $\sum_{DOWNSTREAM} W_{jk}\delta_j$ refers to the *<u>weighted sum of error responsibilities</u>* for nodes downstream
- Rules show why input values require normalization
  - Large input values x_{ij} would dominate weight adjustment
  - Error propagation would be overwhelmed, and learning stifled



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

- Recall that first pass through network yielded *output = 0.8750*
- Assume actual target value = 0.8, and learning rate, η = 0.01
- Prediction error = 0.8 - 0.8750 = -0.075
- Neural Networks use stochastic (or online) back-propagation
- Weights updated after each record processed by network
- Error responsibility for Node *Z*, an output node, found first

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



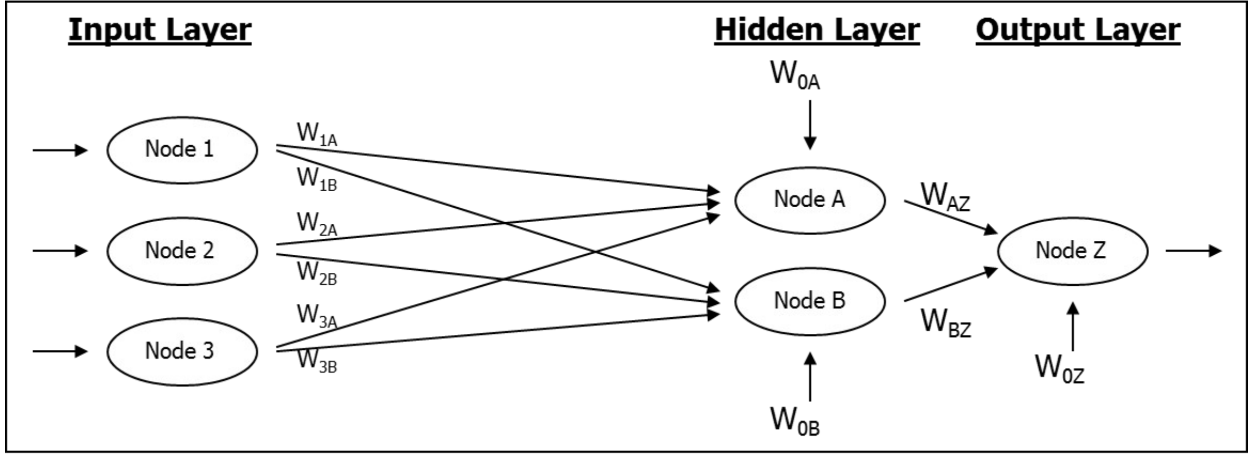
$x_0 = 1.0$	$W_{0A} = 0.5$	$W_{0B} = 0.7$	$W_{0Z} = 0.499$
$x_1 = 0.4$	$W_{1A} = 0.6$	$W_{1B} = 0.9$	$W_{AZ} = 0.9$
$x_2 = 0.2$	$W_{2A} = 0.8$	$W_{2B} = 0.8$	$W_{BZ} = 0.9$
$x_3 = 0.7$	$W_{3A} = 0.6$	$W_{3B} = 0.4$	

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

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

- Move upstream to Node *A*, a hidden layer node

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



$x_0 = 1.0$	$W_{0A} = 0.5$	$W_{0B} = 0.7$	$W_{0Z} = 0.499$
$x_1 = 0.4$	$W_{1A} = 0.6$	$W_{1B} = 0.9$	$W_{AZ} = 0.899$
$x_2 = 0.2$	$W_{2A} = 0.8$	$W_{2B} = 0.8$	$W_{BZ} = 0.9$
$x_3 = 0.7$	$W_{3A} = 0.6$	$W_{3B} = 0.4$	

- Adjust weight using back-propagation rules

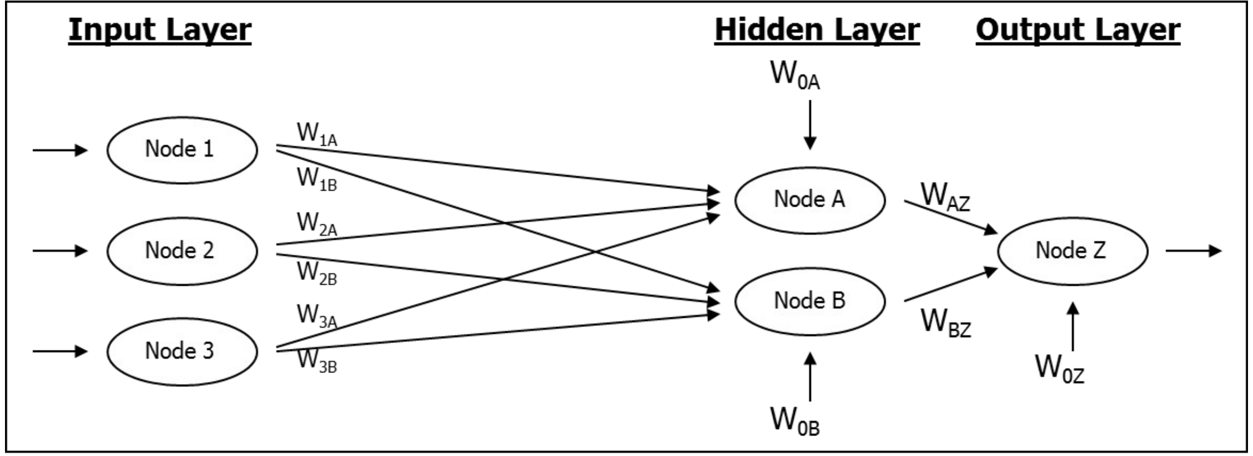
$$\Delta W_{AZ} = \eta\delta_Z(OUTPUT_A) = 0.1(-0.0082)(0.7892) = -0.000647$$

$$w_{AZ,NEW} = w_{AZ,CURRENT} + \Delta w_{AZ} = 0.9 - 0.000647 = 0.899353$$

- Connection weight between Node *A* and Node *Z* adjusted from 0.9 to 0.899353
- Calculate error at Node *B* (hidden layer node)

$$\delta_B = \mathrm{output_B}(1 - \mathrm{output_B}) \sum_{DOWNSTREAM} W_{jk}\delta_j$$

$$= 0.8176(1 - 0.8176)(0.9)(-0.0082) = -0.0011$$



$x_0 = 1.0$	$W_{0A} = 0.5$	$W_{0B} = 0.7$	$W_{0Z} = 0.499$
$x_1 = 0.4$	$W_{1A} = 0.6$	$W_{1B} = 0.9$	$W_{AZ} = 0.899$
$x_2 = 0.2$	$W_{2A} = 0.8$	$W_{2B} = 0.8$	$W_{BZ} = 0.899$
$x_3 = 0.7$	$W_{3A} = 0.6$	$W_{3B} = 0.4$	

- Adjust weight 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

Example Summary

- Now, all network weights in model are updated
- Each iteration based on single record from data set
- 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 through data set performed before termination criterion is met
- Constantly adjusting weights to reduce prediction error
- When to terminate?
    - Stopping criterion may be computational “clock” time?

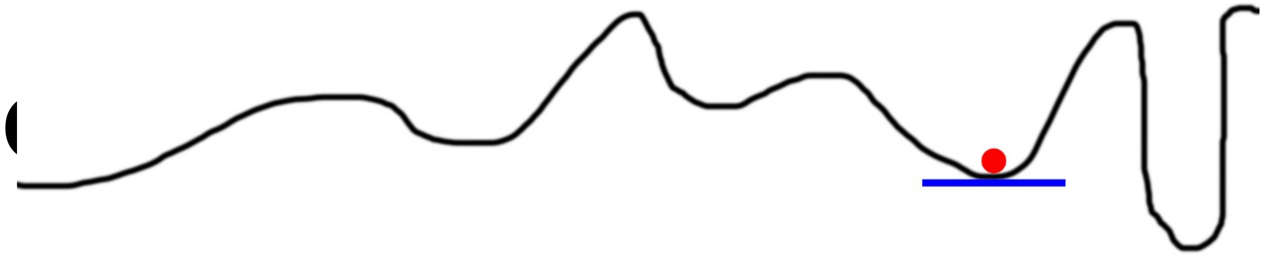
      Short training times likely result in poor model

    - Terminate when SSE reaches threshold level?
    - Neural Networks are prone to overfitting
    - Memorizing patterns rather than generalizing

Termination Criteria

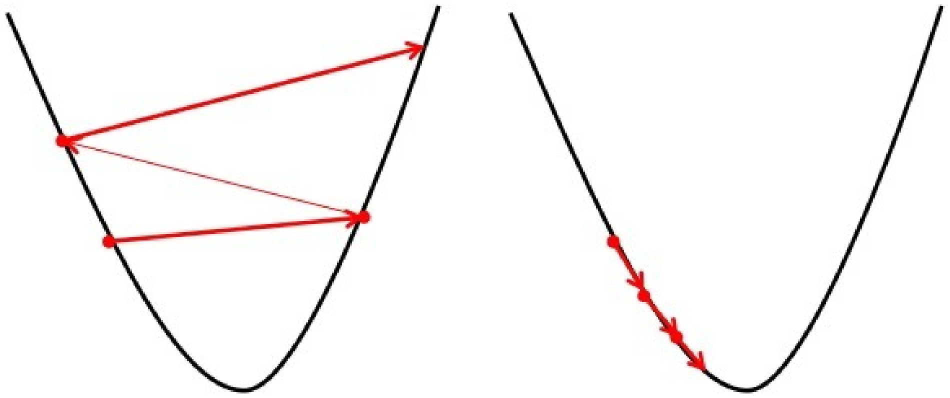
- Cross-Validation Termination Procedure
  - Retain portion of training set as "hold out" data set
  - Train network on remaining data
  - Apply weights learned from training set to validation set
  - Measure two sets of weights
  - "Current" weights for training set, "Best" weights with minimum SSE on validation set
  - Terminate algorithm when current weights have significantly greater SSE than best weights
  - However, Neural Networks not guaranteed to arrive at global minimum for SSE

Termination Crit



- Algorithm may become stuck in local minimum
- Results in good, but not optimal solution
- Not necessarily an insuperable problem
- Multiple networks trained using different starting weights
- Best model from group chosen as “final”
- Stochastic back-propagation method acts to prevent getting stuck in local minimum
- Random element introduced to gradient descent
- Momentum term may be added to back-propagation algorithm

Learning Rate



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

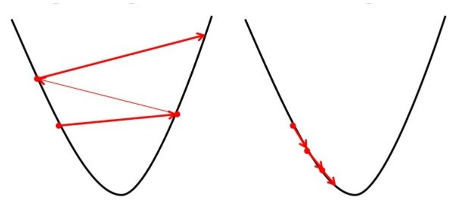
$$0 < \eta < 1, \text{ where}$$

$$\eta = \text{learning rate}$$

- Helps adjust weights toward global minimum for SSE
- Small Learning Rate
  - With small learning rate, weight adjustments small
  - Network <u>*takes unacceptable time*</u> converging to solution
- Large Learning Rate
  - Suppose algorithm close to optimal solution
  - With large learning rate, network likely to "<u>*overshoot*</u>" optimal solution

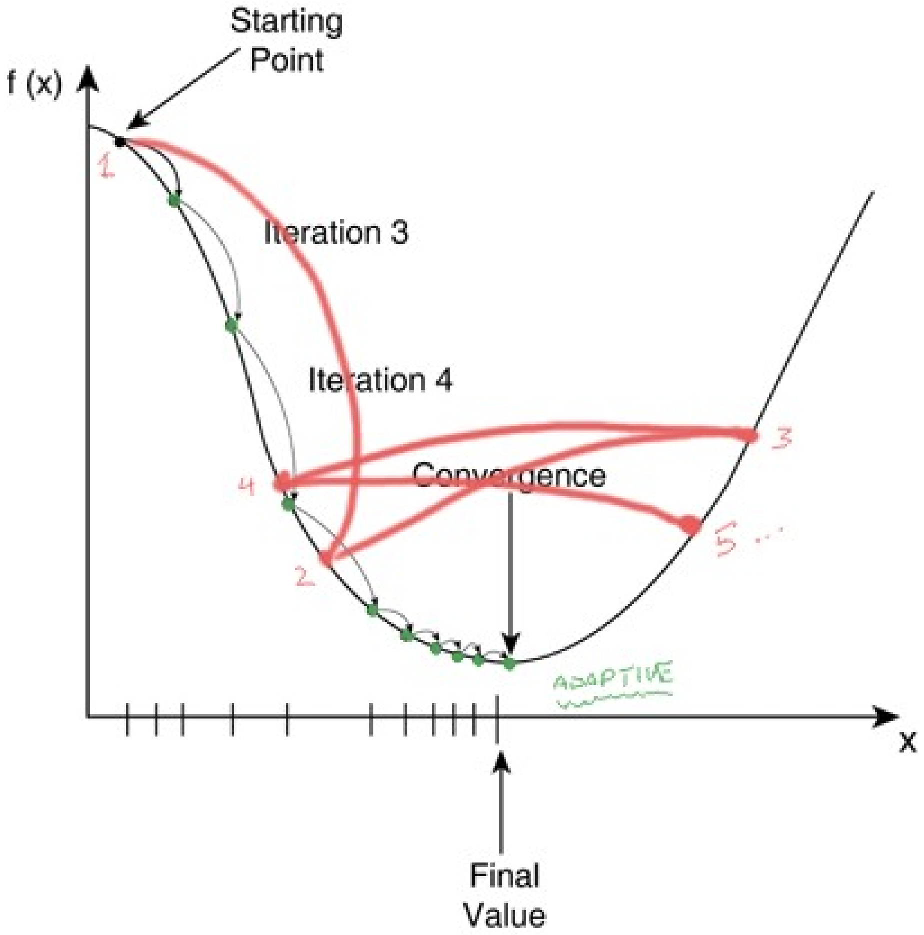
Learning Rate

- w^*: optimum weight for w, which has current value $w_{CURRENT}$
- According to Gradient Descent Rule, $w_{CURRENT}$ adjusted in direction of w^*
- Learning rate acts as **multiplier** to formula $\Delta w_{CURRENT}$
- Large learning may cause w_{NEW} to jump past w^*
- w_{NEW} may be farther away from w^* than $w_{CURRENT}$
- Next adjusted weight value on opposite side of w^*
- Leads to oscillation between two "slopes"
- Network never settles down to minimum between them



Adjust Learning Rate as Training Process

- Learning rate initialized with large value
- Network quickly approaches general vicinity of optimal solution
- As network begins to converge, learning rate gradually reduced
- Avoids overshooting minimum



Momentum Term

- Momentum term (“alpha”) makes back-propagation more powerful, and represents inertia

$$\Delta w_{CURRENT} = -\eta \frac{\partial SSE}{\partial w_{CURRENT}} + \alpha \Delta w_{PREVIOUS}$$

where $\Delta w_{PREVIOUS}$ = previous weight adjustment, $0 \le \alpha < 1$

$\alpha \Delta w_{PREVIOUS}$ = fraction of previous weight adjustment for a given weight

- Large momentum values influence $\Delta w_{CURRENT}$ to move ***same*** direction as previous adjustments
- Including momentum in back-propagation results in adjustments becoming ***exponential average*** of all previous adjustments (Reed and Marks)

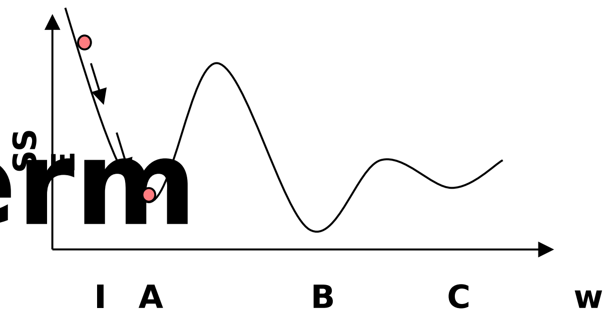
$$\Delta w_{CURRENT} = -\eta \sum_{k=0}^{\infty} \alpha^k \frac{\partial SSE}{\partial w_{CURRENT-k}}, \text{ where}$$

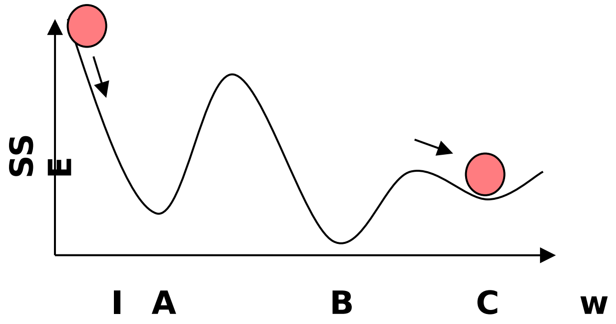
α^k = indicates more recent adjustments exert larger influence

Momentum Term

- Large *alpha* values enable algorithm to remember more terms in adjustment history
- Small alpha values reduce inertial effects, and influence of previous adjustments
- With *alpha* = 0, all previous components disappear
- Momentum term encourages adjustments in **same** direction
- Increases rate which algorithm approaches neighborhood of optimality
- Early adjustments in same direction
- Exponential average of adjustments, also in same direction

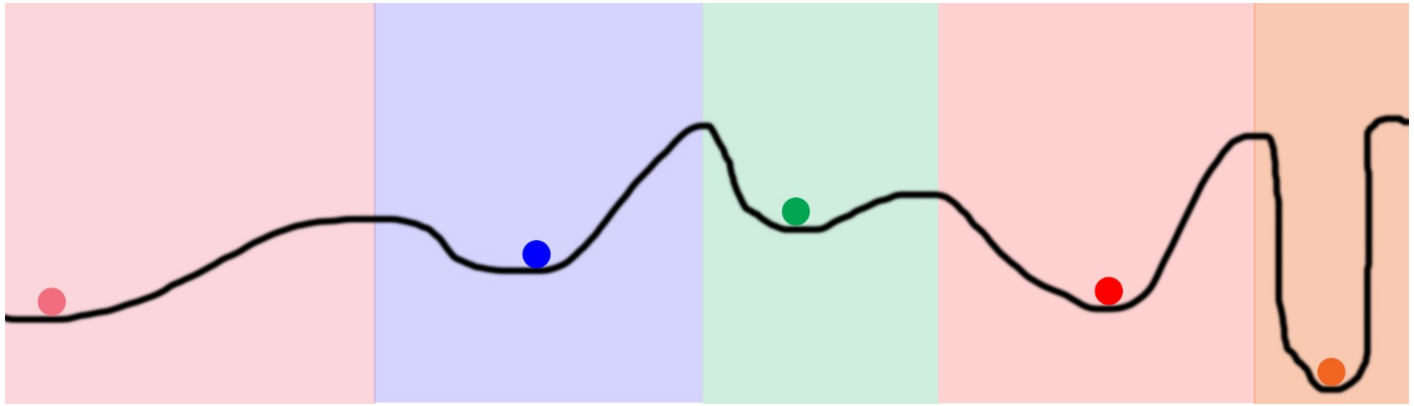
Momentum Term





- Weight initialized to I, with local minima at A, C
- Global minimum at B
- Small Ball
  - Small “ball”, symbolized as momentum, rolls down hill
  - <u>Gets stuck in first trough A</u>
  - Momentum helps find A (local minimum), but not global minimum B
- Large Ball
  - Now, large “ball” symbolizes momentum term
  - Ball rolls down curve and passes over first and second hills
  - Overshoots global minimum B because of too much momentum
  - Settles to local minimum at C
- Values of learning rate and momentum require ***careful consideration***
- Experimentation with different values necessary to obtain best results

Why Neural Networks Work



- Neural networks have **many** hidden layers
- Each node is initialized in a random starting state
- This increases the ability of the neural network to cover a large <u>search space</u>
- Results in finding many local minima and arrive at the optimal solution.

Sensitivity Analysis

- Opacity is drawback of Neural Networks
- Flexibility enables modeling of non-linear behavior
- However, limits ability to interpret results
- No procedure exists for <u>translating weights to decision rules</u>
- Sensitivity Analysis measures relative influence attributes have on solution

Sensitivity Analysis Procedure

- Generate new observation x_{MEAN}
- Each attribute value of x_{MEAN} equals mean value for attributes in data set
- Find network output, for input x_{MEAN}, called $output_{MEAN}$
- Attribute by attribute, vary x_{MEAN} to reflect attribute Min and Max
- Find network output for each variation, compare to $output_{MEAN}$
- Determines attributes, varying from their Min to Max, having greater effect on network results, compared to other attributes

If you really want to delve into neural networks

- Here is a two part blog post with examples in Python!
- http://iamtrask.github.io/2015/07/12/basic-python-network/
- http://iamtrask.github.io/2015/07/27/python-network-part2/
- Talk with me!