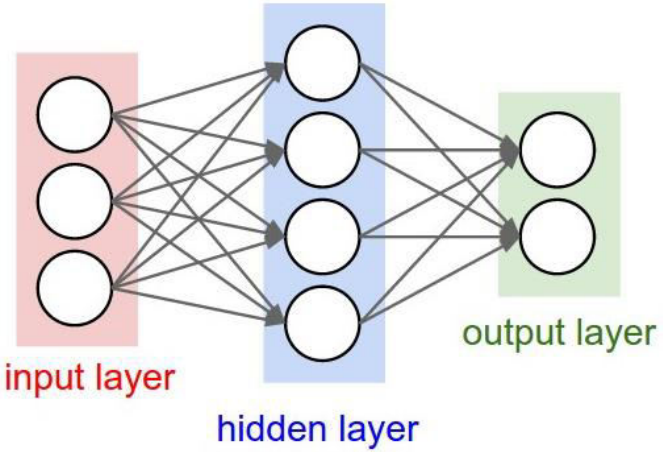
Neural Network Classification

Neural Network : The **neural network** is based on a model of **biological activity in the brain**, where **neurons are interconnected** and **learn from experience**.

Multilayer feed forward networks (Multilayer perceptron of ANN)
Input Layer – consists the nodes that simply accept the input values.

Hidden Layer – Hidden layer take the inputs and then apply **transfer function** included with weights and bias for each connections.

Output Layer - It obtains input values from hidden layer and take weighted average of these inputs and then apply the transfer function to get the prediction.

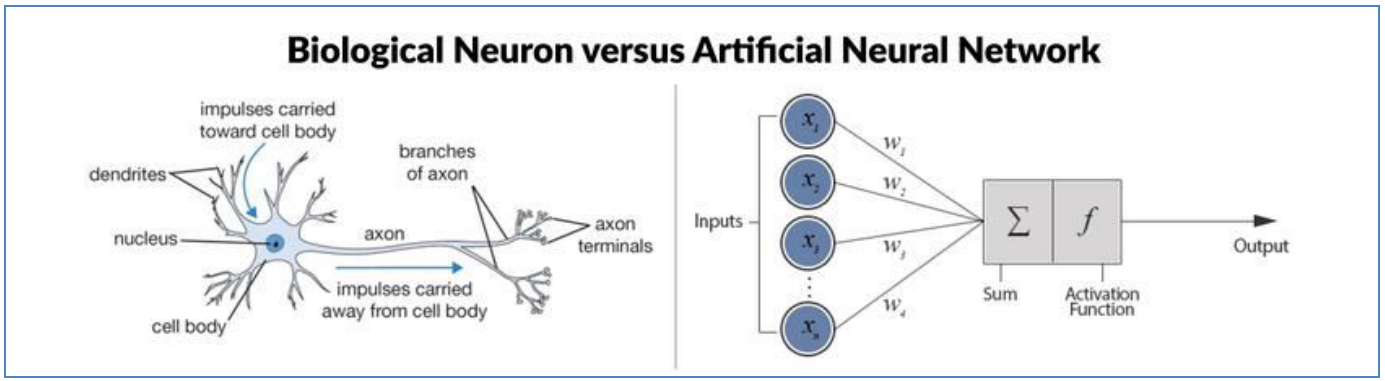


Transfer Functions : linear function , exponential function, Logistic Function

Logistic Function $output_j = g(\theta_j + \sum_{i=1}^{p} w_{ij}x_i) = \frac{1}{1+e^{-(\theta_j+\sum_{i=1}^{p} w_{ij}x_i)}}$.

Wij = Wieghts

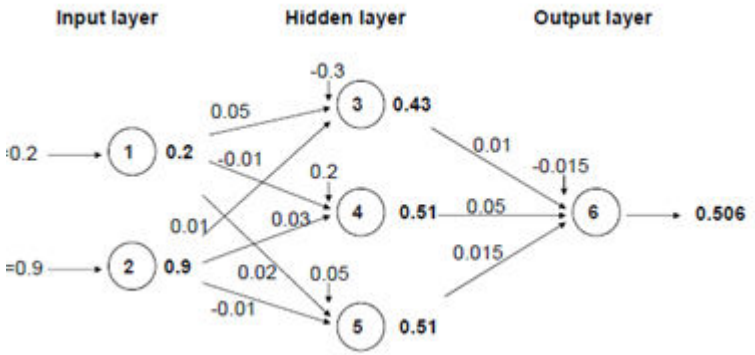
θ_j = Bias for each node



Biological Neuron

Single Perceptron

Neural Network Classification



Txn_Amt	Txn_Time	Fraud
0.9	0.2	1
0.1	0.1	0
0.2	0.2	0
0.2	0.3	0
0.7	0.2	1
0.8	0.7	1

(i) Initialize the weights and bias values with random numbers in the range of 0.00 +-0.05.

(ii) Using logistic function we can compute the output of node-3.

$$output_j = g(\theta_j + \sum_{i=1}^{p} w_{ij}x_i) = \frac{1}{1+e^{-(\theta_j+\sum_{i=1}^{p} w_{ij}x_i)}}. \qquad Output_3 = \frac{1}{1+e^{-\{-0.3+(0.05)(0.2)+(0.01)(0.9)\}}} = 0.43$$

(iii) Similar way outputs for other nodes will get computed and computed for final output node also.

$$Output_6 = \frac{1}{1+e^{-\{-0.015+(0.01)(0.430)+(0.05)(0.507)+(0.015)(0.511)\}}} = 0.506.$$

(iv) Classify this record to 1 based on cutoff 0.5 because 0.506 > 0.5

(v) Training the Model using **back propagation** of error to updates the weights and biases for nodes.

(vi) Error associated with node K is computed by $Err_k = \hat{y}_k(1-\hat{y}_k)(y_k-\hat{y}_k)$

(vii) weights are then updated $\theta_j^{new} = \theta_j^{old} + lErr_j \quad w_{i,j}^{new} = w_{i,j}^{old} + lErr_j$

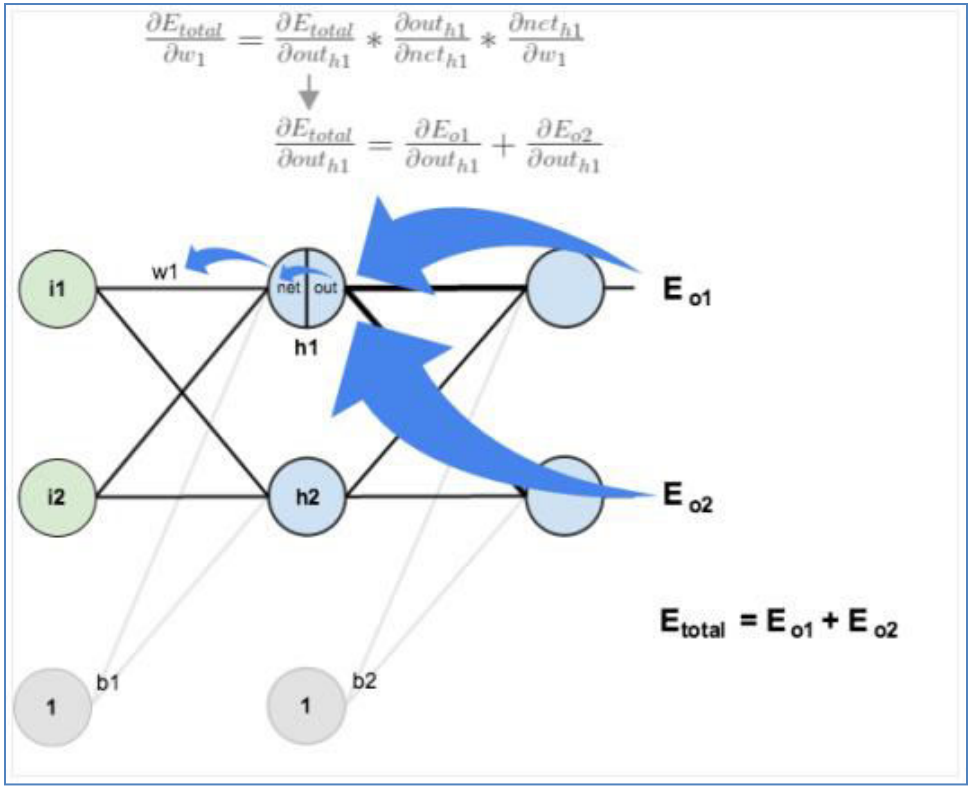
(viii) Error associated with output = (0.506) (1-.506)(1-.506) = 0.123, l is learning rate

(ix) weights got updated with error in back propagation manner for node-6.

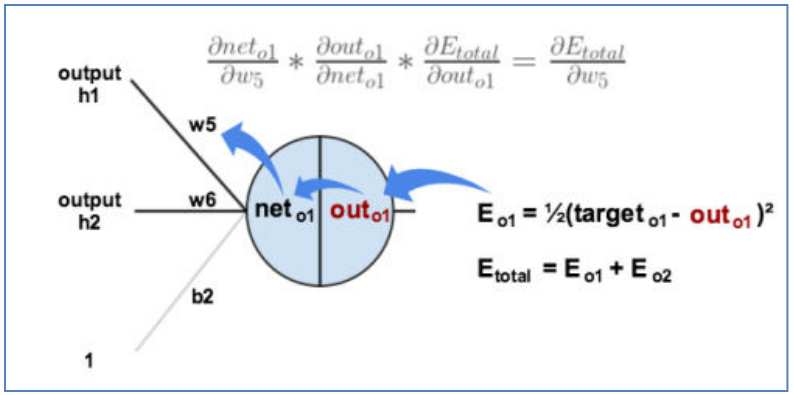
$$\theta_6 = -0.015 + (0.5)(0.123) = 0.047 \qquad w_{3,6} = 0.01 + (0.5)(0.123) = 0.072 \qquad w_{4,6} = 0.05 + (0.5)(0.123) = 0.112$$

(x) these weights keep get updated until all the observations are used, this is called one sweep or epoch.

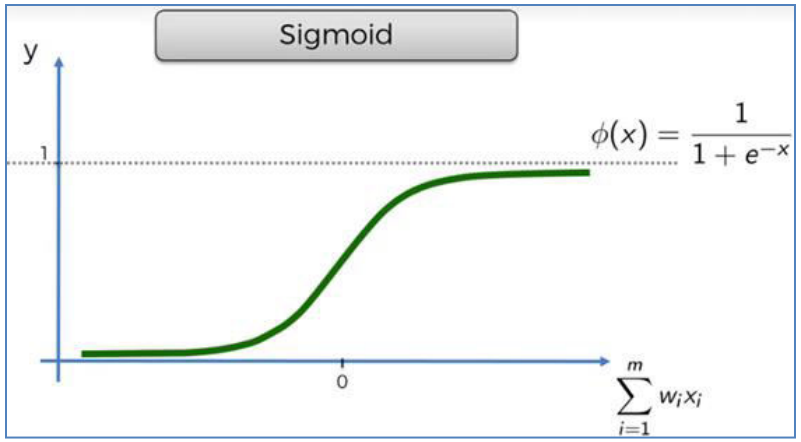
Chain Rule Back Propagation

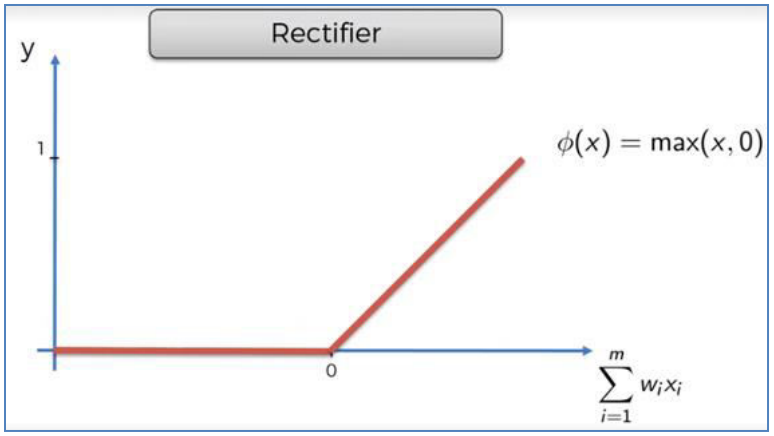


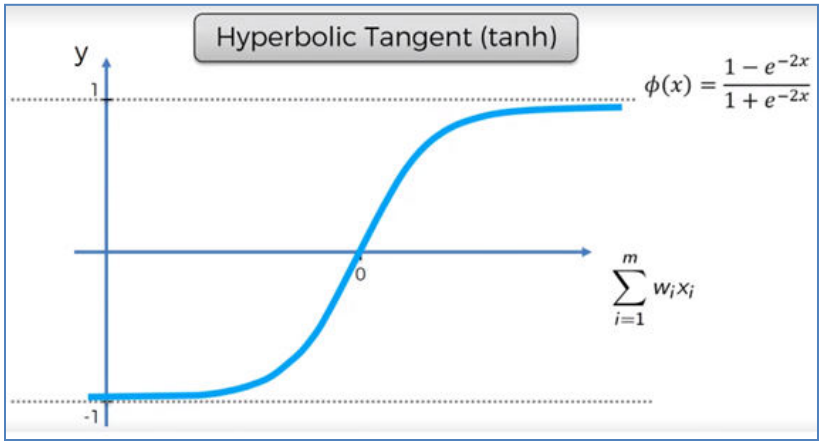
$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx} = f'(y)g'(x) = f'(g(x))g'(x)$$

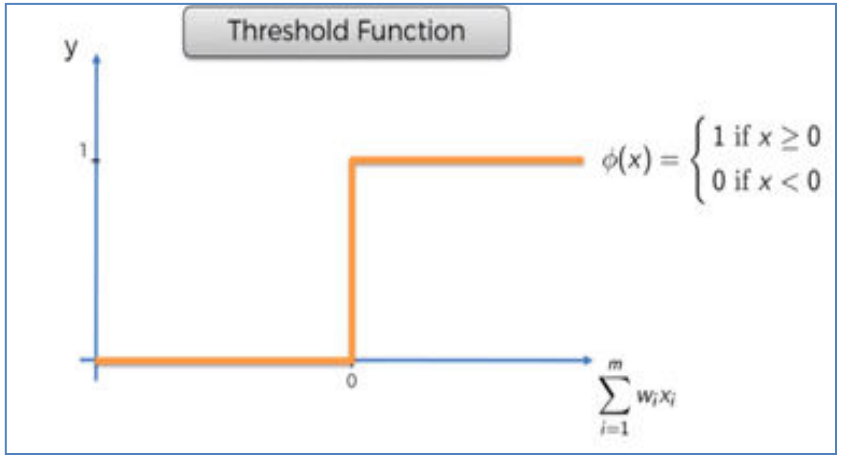


Activation Function









Neural Network Classification

- **Two key architecture** decisions that you need to make to make your model:
  (i) how many **layers** you're going to use
  (ii) how many "**hidden units**" you will chose for each layer

- Pre processing of predictors to standardize them before applying to neural network for best performance.

- Categorical variables need to transformed into dummy variables before applying to neural network.

- Neural Network back propagation weights updating gets stop when
  (i) when misclassification error reached to threshold.
  (ii) when the limit on the number of training epoch is reached.

- To avoid the over fitting, limit the number of epochs and not to over train the data and keep measuring the validation error after each epoch and select optimum epoch based on minimum validation error.

Neural Network Classification

- **Two key architecture** decisions that you need to make to make your model:
  (i) how many **layers** you're going to use
  (ii) how many "**hidden units**" you will chose for each layer
- Pre processing of predictors to standardize them before applying to neural network for best performance.
- Categorical variables need to transformed into dummy variables before applying to neural network.

MLP(Multiple Perceptron NN) Tuning Parameters:

(i) hidden_layer_sizes : Number of hidden layer in multiple NN

(ii) activation : Activation function for the hidden layer. {'identity', 'logistic', 'tanh', 'relu'}, default 'relu'
'**identity**', no-op activation, useful to implement linear bottleneck, returns f(x) = x
'**logistic**', the logistic sigmoid function, returns f(x) = 1 / (1 + exp(-x)).
'**tanh**', the hyperbolic tan function, returns f(x) = tanh(x).
'**relu',** the rectified linear unit function, returns f(x) = max(0, x), It gives an output x if x is positive and 0 otherwise.

(iii) Optimizers (Solver) : Stochastic Gradient Descent (SGD), ADAM and RMSprop.
solver : {'lbfgs', 'sgd', 'adam'}, default 'adam'-stochastic gradient descent.
learning rate : Learning rate to update weights {'constant', 'invscaling', 'adaptive'}, default 'constant'
(invscaling – gradually decrease scaling exponent of 'power_t', adaptive – decrease in case of no tol improvement
Loss Function: For regression - Mean Squared Error (MSE), For classification cross-entropy function.
Cross Entropy - $$E = -\sum_{i=1}^{nout} (t_i \log(y_i) + (1 - t_i) \log(1 - y_i))$$

(v) Tolerance(tol) : Tolerance for the optimization (default = .0001)

(vi) Maximum iteration (max_iter): The solver iterates until convergence (determined by 'tol') or this number of iterations.(default = 200)

(vii) validation_fraction : Proportion of training data set for validation for early stopping.
float, optional, default 0.1.

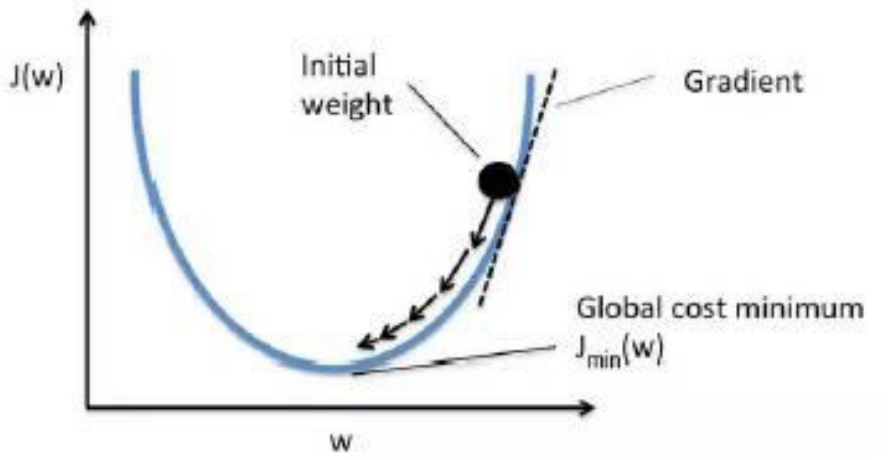
Neural Network Classification

Gradient Descent (Batch) : Updates the weights after each epoch.

- for one or more epochs:
  - for each weight j
    - $w_j := w + \Delta w_j$, where: $\Delta w_j = \eta \sum_i (\text{target}^{(i)} - \text{output}^{(i)}) x_j^{(i)}$

Stochastic Gradient Descent (Iterative) : Updates the weights for each training samples.

- for one or more epochs, or until approx. cost minimum is reached:
  - for training sample i:
    - for each weight j
      - $w_j := w + \Delta w_j$, where: $\Delta w_j = \eta (\text{target}^{(i)} - \text{output}^{(i)}) x_j^{(i)}$



$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^{n} (y^{(i)} - \hat{y}^{(i)})^2$$

Mean Squared Error

$$\mathcal{L} = -\frac{1}{n} \sum_{i=1}^{n} \left[y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right]$$

Cross Entropy

Multinomial Classification

The **Sigmoid** function is used for the **two-class logistic** regression, whereas the **Softmax** function is used for the **multiclass logistic** regression.

In Softmax Regression, we replace the **Sigmoid** logistic function by **Softmax** function.

Binary-class Sigmoid Logistic Regression

Predicted Probabilities for Binary Class

$$\Pr(Y_i = 0) = \frac{e^{-\beta \cdot \mathbf{X}_i}}{1 + e^{-\beta_0 \cdot \mathbf{X}_i}}$$

$$\Pr(Y_i = 1) = 1 - \Pr(Y_i = 0) = \frac{1}{1 + e^{-\beta \cdot \mathbf{X}_i}}$$

Multi-class Softmax Logistic Regression

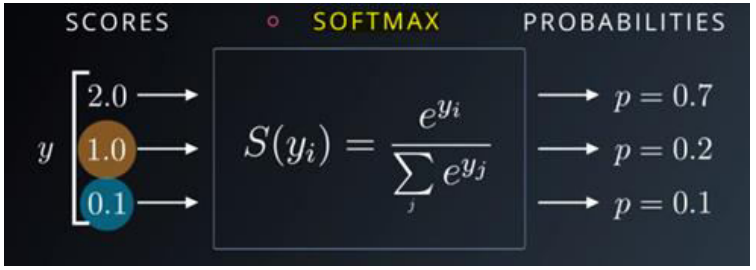
Predicted Probabilities for Multiple Classes K

$$\Pr(Y_i = k) = \frac{e^{\beta_k \cdot \mathbf{X}_i}}{\sum_{0 \le c \le K} e^{\beta_c \cdot \mathbf{X}_i}}$$

$$P(y = j \mid z^{(i)}) = \phi_{softmax}(z^{(i)}) = \frac{e^{z^{(i)}}}{\sum_{j=0}^{k} e^{z_k^{(i)}}}.$$

$$\Pr(Y_i = 0) = \frac{e^{\beta_0 \cdot \mathbf{X}_i}}{\sum_{0 \le c \le K} e^{\beta_c \cdot \mathbf{X}_i}} = \frac{e^{\beta_0 \cdot \mathbf{X}_i}}{e^{\beta_0 \cdot \mathbf{X}_i} + e^{\beta_1 \cdot \mathbf{X}_i}} = \frac{e^{(\beta_0 - \beta_1) \cdot \mathbf{X}_i}}{e^{(\beta_0 - \beta_1) \cdot \mathbf{X}_i} + 1} = \frac{e^{-\beta \cdot \mathbf{X}_i}}{1 + e^{-\beta \cdot \mathbf{X}_i}}$$

$$\Pr(Y_i = 1) = \frac{e^{\beta_1 \cdot \mathbf{X}_i}}{\sum_{0 \le c \le K} e^{\beta_c \cdot \mathbf{X}_i}} = \frac{e^{\beta_1 \cdot \mathbf{X}_i}}{e^{\beta_0 \cdot \mathbf{X}_i} + e^{\beta_1 \cdot \mathbf{X}_i}} = \frac{1}{e^{(\beta_0 - \beta_1) \cdot \mathbf{X}_i} + 1} = \frac{1}{1 + e^{-\beta \cdot \mathbf{X}_i}}$$



For Multi Classification, Soft Max can be used that is generalized of Sigmoid (Binary Classification)
http://ufldl.stanford.edu/wiki/index.php/Softmax_Regression