Perceptron Network classes: documentation – Neuron Class

CLASS NAME	**Cneuron**	
Data Members	Access	Description
struct dendrite	private	Contains to double variables 'x' (the dendrite's input) and 'w' (the weight modifier).
dendrite* dendrites	private	Dendrites are the input elements that feed into the neuron
unsigned int dendriteTotal	private	Total # of dendrites attached to the neuron
double activation	private	The sum of the weighted inputs (or dendrites)
double y γ	private	The actual output (after the activation has been passed through the 'sigmoid function')
double d δ	private	delta (or ERROR) between the desired and actual outputs
double dy	Private	The Desired Output - only applies to output layer neurons (to calculating their delta)

Return Value	Function Name	Parameters	Description	Notes
	default constructor			Called in *layer* object in creating an array of *neurons*
	custom constructor	unsigned int newDendriteTotal	Total # of dendrites (including bias input)	
	copy constructor			
	destructor			
void	**CalcOutput**			Calculates and sets the net activation of the neuron, then passes this through a 'Sigmoid Function' to get a non-linear output.
void	**CreateDendrites**	unsigned int newDendriteTotal	The total # of dendrites to be created for the neuron.	Sets up and DMAs the dendrites of a neuron (if the default constructor was used this needs to be called before neuron is used).
double returns the value of *activation*	**GetActivation**			Get the current *activation* level of the neuron.
double returns the value of *d*	**GetDelta**			Retrieve the Delta (or Error) in the neuron's output
double returns the value of *dendrites[*].w*	**GetDendrite_w**			Returns the current weight modifier applied to the dendrite
double returns the value of *dendrites[*].x*	**GetDendrite_x**			Returns the input value currently applied to the dendrite
double returns the value of *dy*	**GetDesiredOutput**			Get the Desired Output of an output layer neuron.
double returns the value of *y*	**GetOutput**			Get the actual output of the neuron after it has passed through the 'sigmoid function'.
void	**info**			Display info about the neuron
void	**SetDelta**	double new_d		Sets the Delta (or Error) in the neuron's output used to adjust the dendrite weights of the neuron.
bool 0 = ERROR 1 = SUCCESS	**SetDendrite_w**	unsigned int dendriteElem	Index in the *dedrites* array	Sets the weight modifier applied to a dendrite (usually random values are assigned before training begins).
		double new_w	weight modifier to be applied to dendrite	

bool 0 = ERROR 1 = SUCCESS	**SetDendrite_x**	unsigned int dendriteElem	Index in the *dendrites* array	Set the input value applied to a dendrite. Any attempt to set the bias input(x) will be rejected (dendrites[0]) as this is ALWAYS +1.
		double new_x	new input value to be applied to dendrite	
void	**SetDesiredOutput**	double new_dy	The new value of *dy*	Set the Desired Output of an output layer neuron.
void	**SetOutput**	double new_y	The new output value of the neuron	Explicitly set the output value: This is used to set up the input layer neurons. No processing takes place at input layer neurons. They are used simply to supply input values to be processed by subsequent layers in the net.

Perceptron Network classes: documentation – Layer Class

CLASS NAME	**Clayer**	
Data Members	Access	Description
Cneuron* neurons	private	Array of neurons belonging to the layer
unsigned int neuronTotal	private	
unsigned int dendriteTotal	private	The total # of dendrites belonging to each neuron in the layer
Unsigned int type	private	The type of layer (set during initialisation). 0 = input layer, 1 = hidden layer, 2 = output layer There is only ever one input & output layer. There can be any number of hidden layers in-between.
	private	

Return Value	Function Name	Parameters	Description	Notes
	default constructor			default values: type = 1, neuronTotal = 0, dendriteTotal = 2
	custom constructor	unsigned int newNeuronTotal	# of neurons within the layer	
		unsigned int newDendriteTotal	The # of dendrites required by each neuron in a processing layer = # of neurons in preceding layer	
		unsigned int newType	The type of layer (input, hidden or output)	
	copy constructor			
	destructor			deletes dynamically allocated array to which 'neurons' is a pointer
void	**CalcOutput**			Calculates the outputs of all the neurons in the layer (will have no effect on 'input layer' neurons)
void	**CreateNeurons**	unsigned int newNeuronTotal	Sets *neuronTotal*	Creates the neuron elements for this layer (each with dendrites determined by the number of neurons in the previous layer) – also sets layer *type.*
		unsigned int newDendriteTotal	Sets *dendriteTotal*	
		unsigned int newType	Sets the layer *type*	
double returns the value of a neuron's *activation*	**GetActivation**	unsigned int neuronElem	Index of a neuron within the *neurons* array	Get the current *activation* level of a neuron.
double returns the value of *d* (of a particular neuron element in the layer)	**GetDelta**	unsigned int neuronElem		Retrieve the Delta (or Error) in the neuron's output
double returns the value of *dendrites[*].w* (of a neuron element)	**GetDendrite_w**	unsigned int neuronElem		Returns the current weight modifier applied to a neuron's dendrite.
		unsigned int dendriteElem	Index of a particular dendrite belonging to the neuron	
double returns the value of *dendrites[*].x* (of a neuron element)	**GetDendrite_x**	unsigned int neuronElem		Returns the input value currently applied to a neuron's dendrite
		unsigned int dendriteElem		

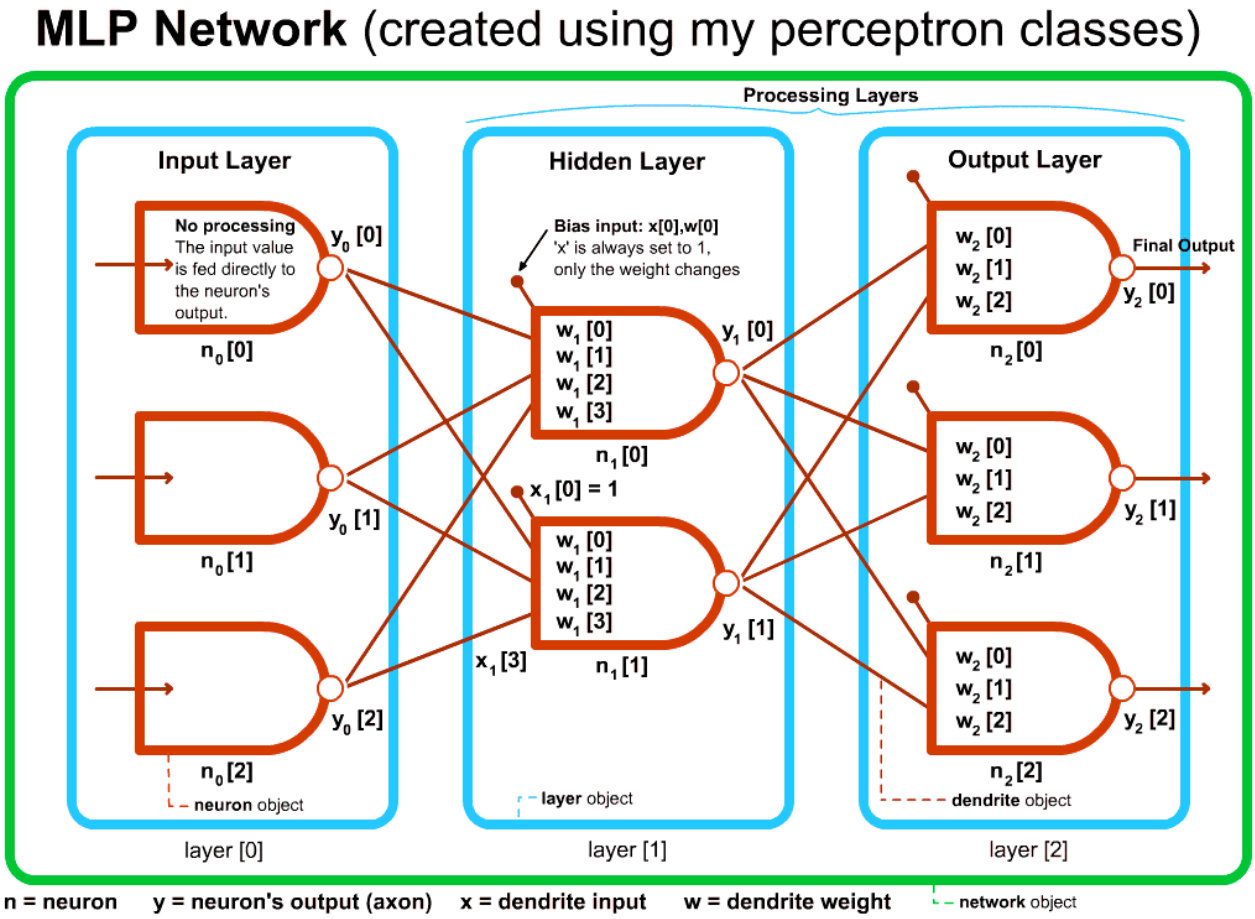
double returns the value of *dy* (of a neuron element)	**GetDesiredOutput**	unsigned int neuronElem		Get the Desired Output of an output layer neuron
unsigned int returns the value of *neuronTotal*	**GetNeuronTotal**			Returns the total number of neurons in the layer
double returns the value of *y* (of a neuron element)	**GetOutput**	unsigned int neuronElem		Get the actual output of the neuron after it has passed through the 'sigmoid function'.
void	**Info**			Display into on the layer (DOS only)
bool 0 = ERROR 1 = SUCCESS	**SetDelta**	unsigned int neuronElem		Set the delta (or Error) in the neuron's output.
		double new_d	The new delta (ERROR) value.	
bool 0 = ERROR 1 = SUCCESS	**SetDendrite_w**	unsigned int neuronElem		Set the weight modifier of a neuron's dendrite.
		unsigned int dendriteElem		
		double new_w		
bool 0 = ERROR 1 = SUCCESS	**SetDendrite_x**	unsigned int neuronElem		Set the input value of a neuron's dendrite.
		unsigned int dendriteElem		
		double new_x		
bool 0 = ERROR 1 = SUCCESS	**SetDesiredOutput**	unsigned int neuronElem		Set the Desired Output of an output layer neuron.
		double new_dy		
void	**SetType**	unsigned int newType		Set the layer's *type* ()

Perceptron Network classes: documentation – Network Class

CLASS NAME	**Cnetwork**	
Data Members	Access	Description
Clayer* layers	private	0 (first element) => input layer 1 (inbetween elements) => hidden layers 2 (last element) => output layer
unsigned int layerTotal	private	The number of layers in the entire network
unsigned int pass	private	The number of passes through the training algorithm
unsigned int epoch	private	The number of times the training algorithm has been applied to the ENTIRE training/pattern set. Incremented by 'IsTrained()'.
double eta η	private	(n) also know as the 'learning co-efficient' (eta x delta x input(i))
double epsilon ϵ	private	The netError must be be below this value to terminate training
double alpha α	private	[not yet implemented] modulates 'momentum' – which is proportional to the change in weight from the previous training pass?
double rmsDelta	private	The net 'Root-Mean-Square' of all the delta errors in the net.

Return Value	Function Name	Parameters	Description	Notes
	default constructor			
	custom constructor	unsigned int newlayerTotal	Number of layers in the network	The network object encapsulates / co-ordinates all neural net objects.
		double newEta		
		double newEpsilon	Value used to decide whether to the net is trained.	
	copy constructor			
	destructor			
void	**createLayers**	unsigned int newlayerTotal		DMAs layer objects to the *layers* array (only required if the default constructor is called).
void	**defineLayers**	unsigned int layerElem	The index of the layer element within the *layers* array. The first element is defined as a *input layer*, the last element becomes the *output layer*. All elements in between are *hidden layers.*	After the net's *layers* have been created they need to be defined before they are useable within the network. (The # of dendrites required by each neuron in a processing layer = # of neurons in the preceding layer +1).
		unsigned int newNeuronTotal	Total # of neurons within the layer.	
double	**GetDendrite_w**	unsigned int layerElem	Index of a layer element	Returns the current weight modifier applied to a neuron's dendrite.
		unsigned int neuronElem	Index of a neuron within the layer	
		unsigned int dendriteElem	Index of a particular dendrite belonging to the neuron	
double	**GetDendrite_x**	unsigned int layerElem		Returns the input value currently applied to a neuron's dendrite.
		unsigned int neuronElem		
		unsigned int dendriteElem		

double	**GetDesiredOutput**	unsigned int layerElem		Get the Desired Output of an output layer neuron
		unsigned int neuronElem		
unsigned int	**GetPass**		*pass* is incremented by the *train()* function	Get the current # of passes through the network training algorithm.
double	**GetRMS**			Get the net *Root-Mean-Square* of the all the neuron deltas (errors) in the network.
void	**info**			Display info on the current state of the network. (DOS-only)
bool 0 = FAIL 1 = SUCCESS (network trained)	**IsTrained**			If the Root-Mean-Square of all deltas is below *epsilon* then the network IS TRAINED, return success to stop training.
bool 0 = ERROR 1 = SUCCESS	**SetDelta**	unsigned int layerElem		Set the delta (or Error) in a neuron's output.
		unsigned int neuronElem		
		double new_d	Neuron's delta	
bool 0 = ERROR 1 = SUCCESS	**SetDendrite_w**	unsigned int layerElem		Set the weight modifier of a neuron's dendrite.
		unsigned int neuronElem		
		unsigned int dendriteElem		
		double new_w	Weight modifier	
bool 0 = ERROR 1 = SUCCESS	**SetDendrite_x**	unsigned int layerElem		Set the input value of a neuron's dendrite.
		unsigned int neuronElem		
		unsigned int dendriteElem		
		double new_x	Input value	
bool 0 = ERROR 1 = SUCCESS	**SetDesiredOutput**	unsigned int neuronElem		Set the Desired Output of an output layer neuron (used by the training algorithm).
		double new_dy	Desired Output	
void	**test**			Gets the inputs from the *input layer* and forward-propagates through the net to produce the corresponding output values at the *output layer.*
void	**train**			Given a particular *pattern* of input values at the *input layer,* apply the training algorithm and adjust the weight modifiers of all dendrites accordingly. Finally, calculate the *Root-Mean-Square* error of the current training pass.



Equation 1:

To find the total ***activation*** of a neuron:

a = activation
n = total number of dendrites
w = dendrite's *weight* modifier
x = dendrite's *input*

$$a = \sum_{i=0}^{n} w_i x_i$$

Equation 2:

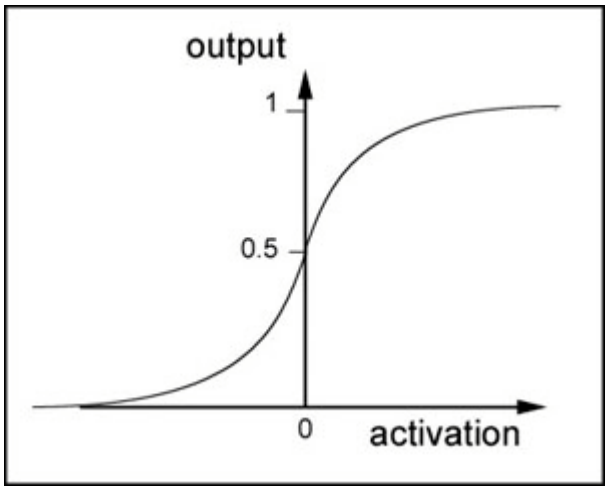
To find the change in a dendrites *weight* (after finding all neuron deltas):

Δw = change in *weight* of the dendrite
η = (*eta*) learning co-efficient
x = *input* applied to the dendrite
δ = *delta* error of the neuron

$$\Delta w = \eta x \delta$$

Equation 3:

To find the neuron's actual output (squashing / sigmoid function):



y = the actual output
a = neuron's *activation*

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

Equation 4:

To find an output layer neuron's *delta* (or error) value:

δ = *delta* error of the neuron
y = the actual output
dy = the desired output

$$\delta = y(1-y)(dy-y)$$

Equation 5:

To find an hidden layer neuron's *delta* (or error) value:

δ = *delta* error of the neuron
y = the actual output
dy = the desired output
λ = the current *layer* index
ρ = the current neuron element
τ = total number of neurons in the next layer up

$$\delta_{\lambda}(\rho) = x_{\lambda}(\rho)[1 - x_{\lambda}(\rho)] \sum_{i=0}^{\tau} w_{\lambda+1}(\rho, i+1)\delta_{\lambda+1}(i)$$