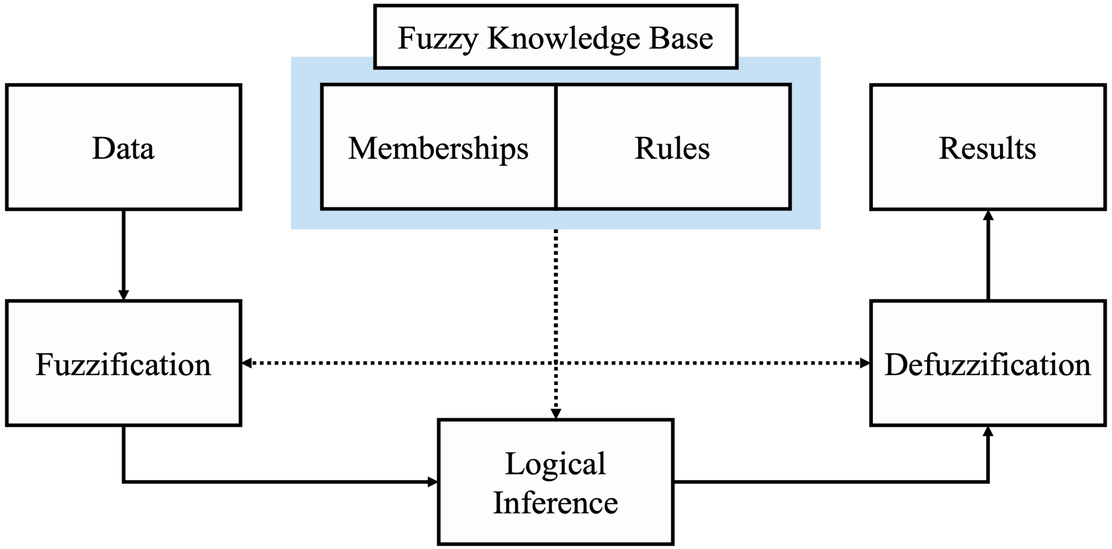
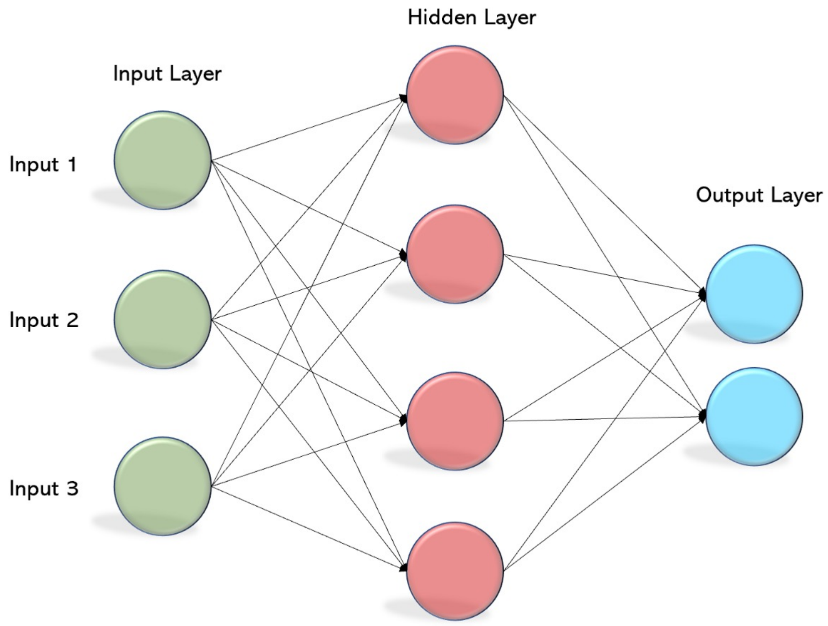
C.I. Neural Networks

Charlie Veal

Electrical Engineering
& Computer Science
University of Missouri

New Transition: Neural Networks





Predictions require human expert knowledge

Explainable by design (i.e., membership functions, rule propositions, implication operations, etc.)

Produces competitive results, but often overfit (i.e., not generalized for similar types of problem)

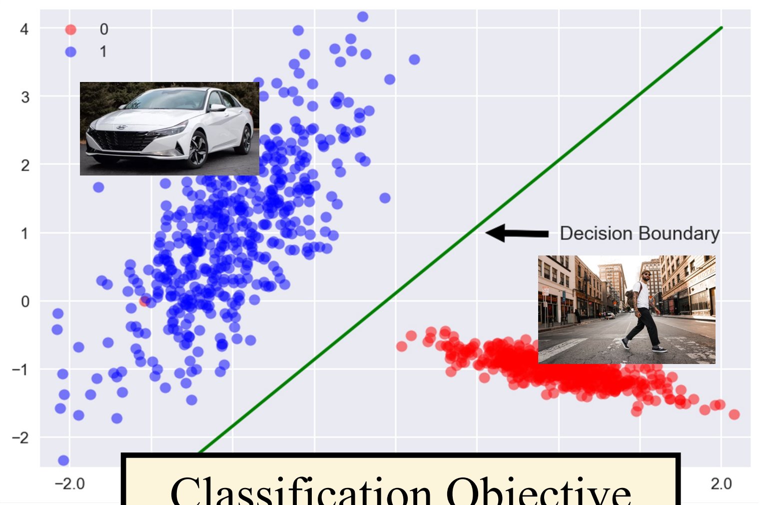
Predictions require data driven learning

Not very explainable (i.e., black box system),

Produces state-of-the-art results for a variety of problems across numerous domains. These results often generalize better than other systems

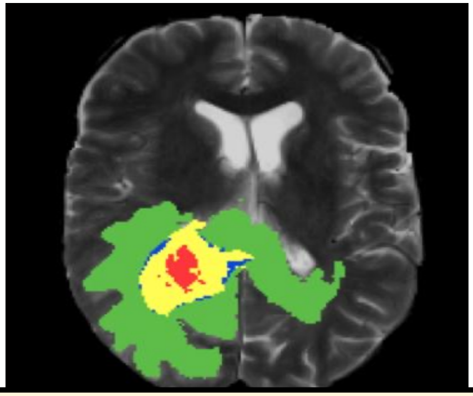
Neural Network Capabilities

NN learning for discrete outputs



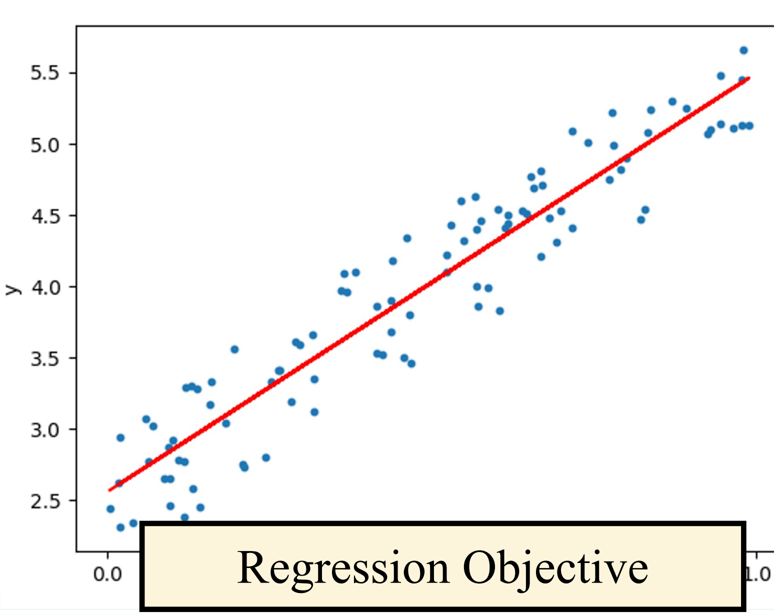
Classification Objective

Classify all pixels, RGB image

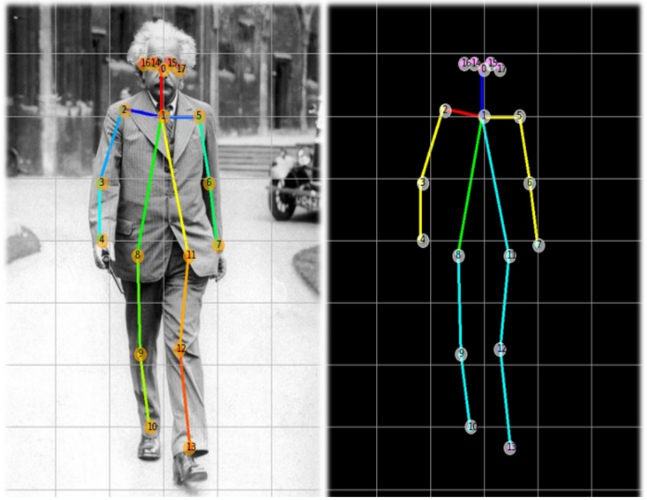


Tumor Detection, MRIs

NN learning for continuous outputs



Regression Objective

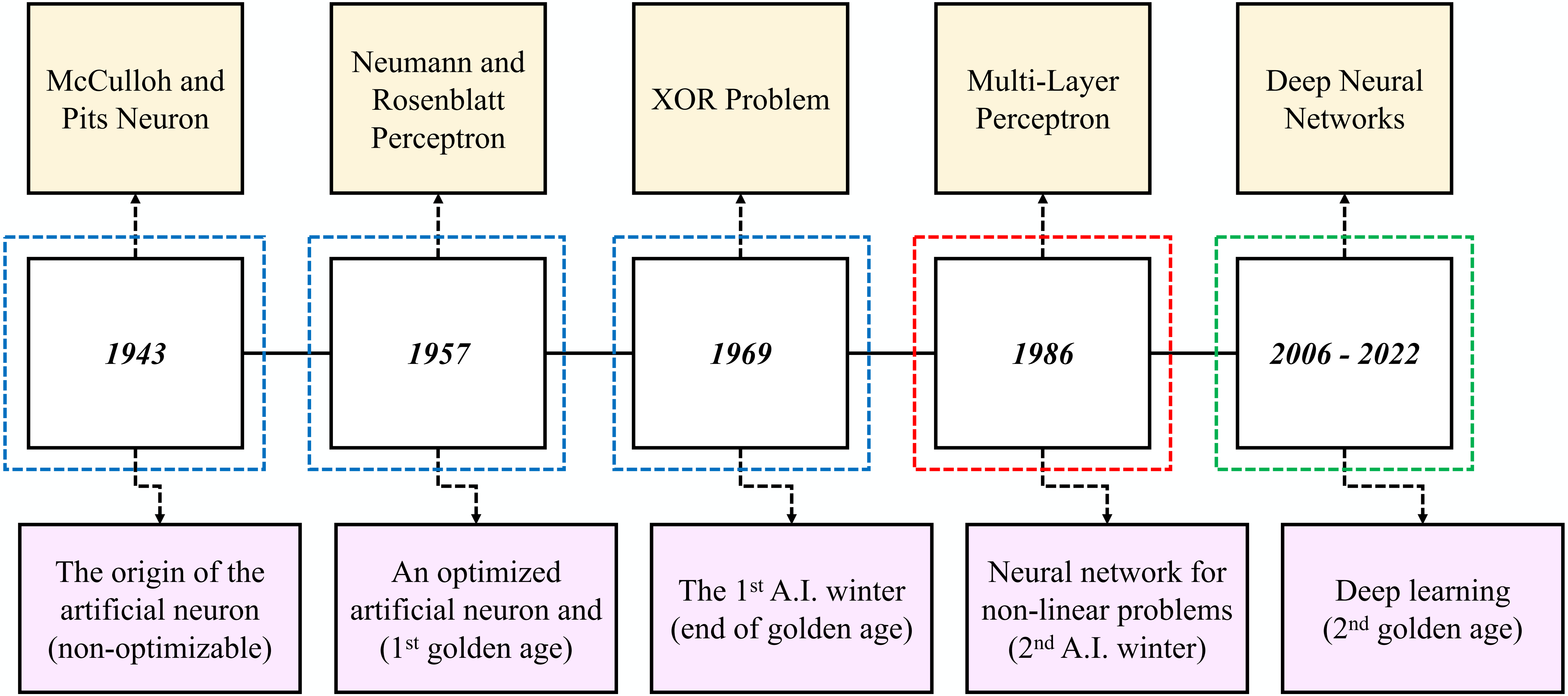


Pose Prediction

Virtual Assistants

Short History of Neural Networks

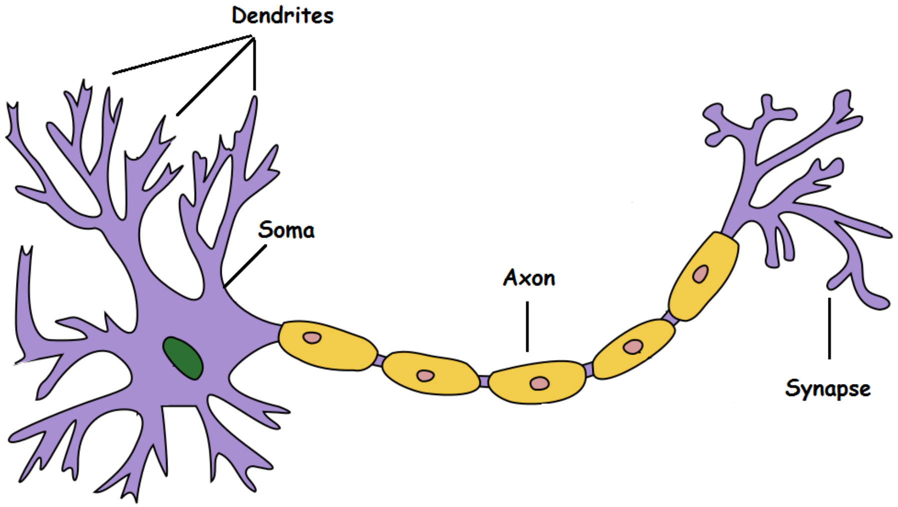
Today | Next Time | N.N. Class



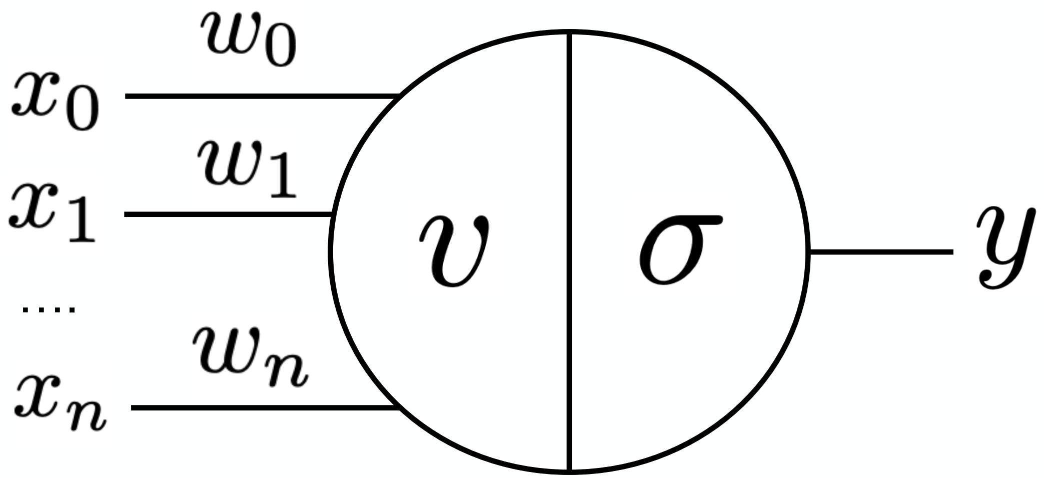
McCulloh and Pits Neuron

What was the artificial neuron that started it all?

Super Simplified Biological Neuron



Computational Equivalent Neuron

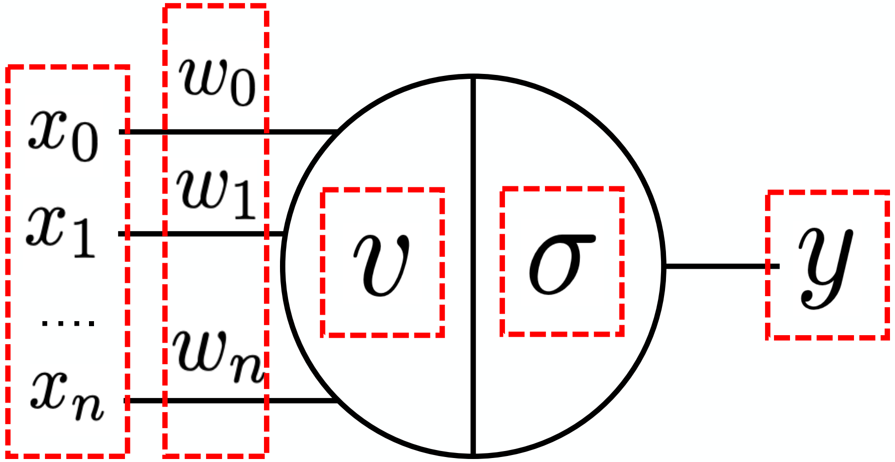


Dendrites	Receiver signals from other neurons
Soma	Processor of info from dendrites
Axon	Transmits outputs of soma to synapse
Synapse	Point of connection to other neurons

x, w	Observation, weights
v	Combination function
σ	Activation function
y	Output / activation / predictions

McCulloh and Pits Neuron

What was the artificial neuron that started it all?



$$x^k = \{x_0^k, x_1^k, \ldots, x_n^k\} | x^k \in X$$

x^k represents the k-th observation from dataset X

$$w = \{w_0, w_1, \ldots, w_n\}$$

w represents strength of connections mapping inputs to the neuron body

$$v = \sum_{i=0}^{n} x_i^k w_i = x^k w^T$$

$$y = \sigma(v)$$

$$\sigma(v,t) = \begin{cases} 1 & \text{if } v > t \\ 0 & \text{otherwise} \end{cases}$$

v represents the linear combination or dot product aggregation between observations and weights

y or σ(v) represents the neuron activation or prediction from linear combination

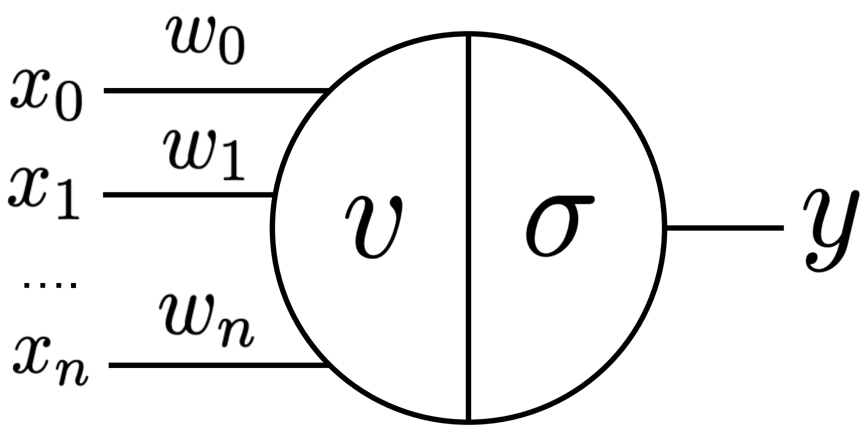
$$x_i^k \in \{0,1\} \quad w_i \in \{0,1\}$$

Constraints: each feature of x and neuron weight be binary : {0, 1}

McCulloh and Pits Neuron

What are the applications of M & P neuron?

M & P Neuron is a non-optimized model that can be used as a logical operator (AND, OR, etc.)



$$x_i^k \in \{0,1\} \quad w_i = 1$$

$$y = \sigma(x^k w^T) \quad \sigma(v,t) = \begin{cases} 1 & \text{if } v > t \\ 0 & \text{otherwise} \end{cases}$$

Example: M & P Neuron, 2 Inputs

$$y = \sigma(x_0^k * w_0 + x_1^k * w_1)$$

$$y = \sigma(x_0^k + x_1^k) \;\Big|\; w_i = 1$$

AND : t = n - 1

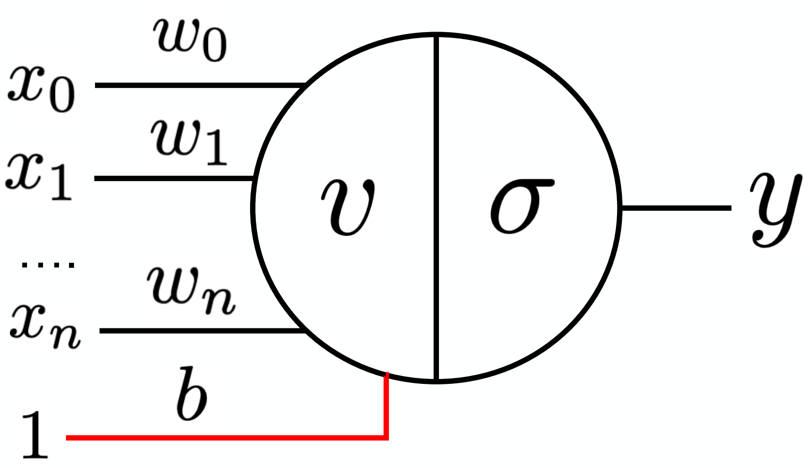
x_0	x_1	y
0	0	0
0	1	0
1	0	0
1	1	1

OR : t = 0

x_0	x_1	y
0	0	0
0	1	1
1	0	1
1	1	1

Neumann and Rosenblatt Perceptron

How does Perceptron differ from M & P Neuron?



$$x_i^k \in \mathbb{R}$$
$$w_i \in \mathbb{R}$$

Relax constraints from M & P Neuron

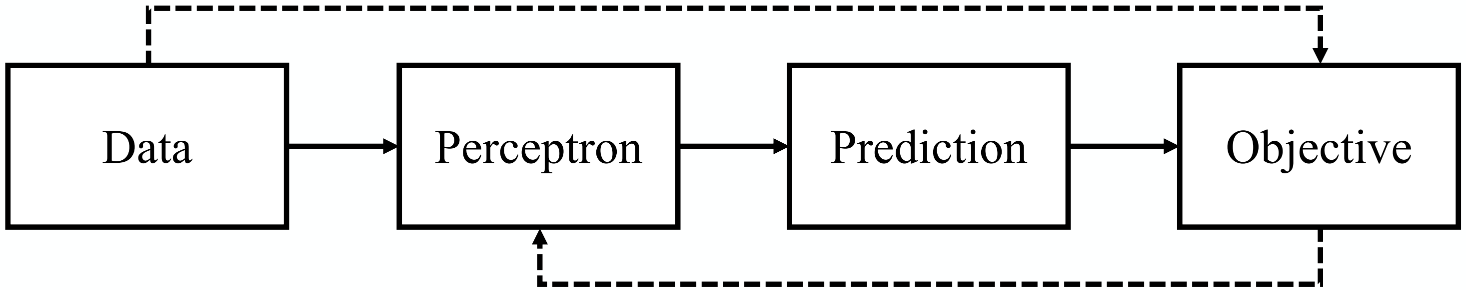
$$v = b + \sum_{i=0}^{n} x_i^k w_i$$

$$x^k = \{x_0^k, x_1^k, ..., x_n^k, 1\}$$
$$w = \{w_0, w_1, ..., w_n, b\}$$

Add in a bias to give more computational and geometrical expression

$$y = \sigma(b + x^k w^T)$$

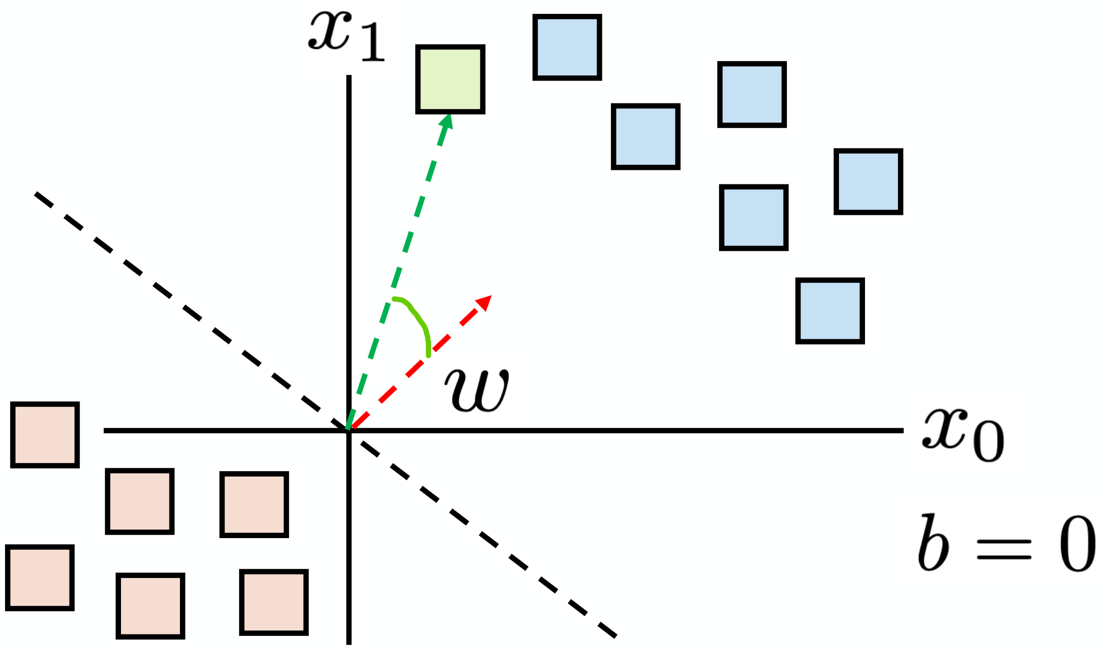
$$y = \sigma(x^k w^T)$$

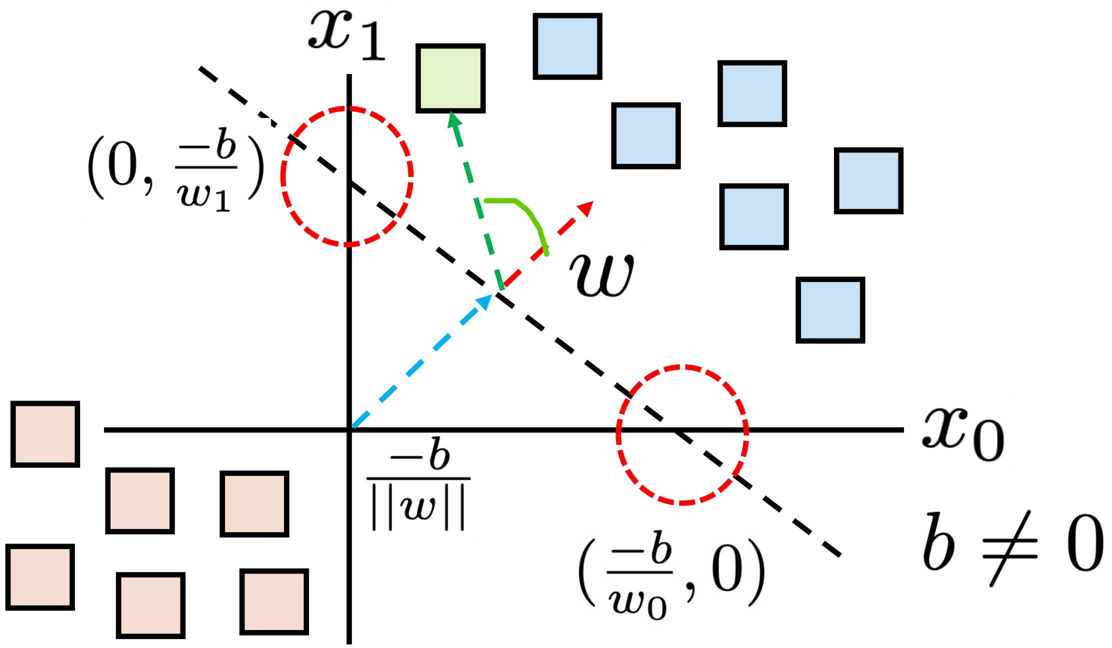


Perceptron is an optimizable model

Neumann and Rosenblatt Perceptron

What does the perceptron look like geometrically?





Perceptron creates a decision boundary in feature space. Weight vector is perpendicular to boundary and points towards positive class.

Perceptron classification can be expressed through geometrical dot product

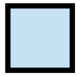
Solve for axis intercepts and plot boundary

$$x_0 w_0 + x_1 w_1 + b = 0$$

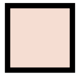
$$x_0 = \frac{-b}{w_0} \qquad x_1 = \frac{-b}{w_1}$$

$$v = x^k w^T = ||x||||w||cos(\theta)$$

For new each observation:



$\theta < 90$



$\theta > 90$

Neumann and Rosenblatt Perceptron

How to learn perceptron parameters?

Learning requires an explicit definition of error ***e*** defined by an objective function ***E***

Perceptron Learning Rule	Hand-crafted methodology to learn a perceptron's parameters from data

$$E = e = \boxed{d_i} - \boxed{y_i}$$

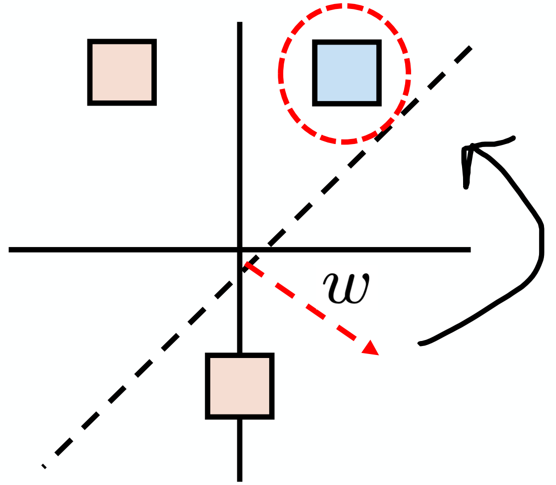
Truth label	Pred label

$$e = 0,\ e = 1,\ e = -1$$

$$w^n = w^o + e x_i^k \qquad b^n = b^o + e$$

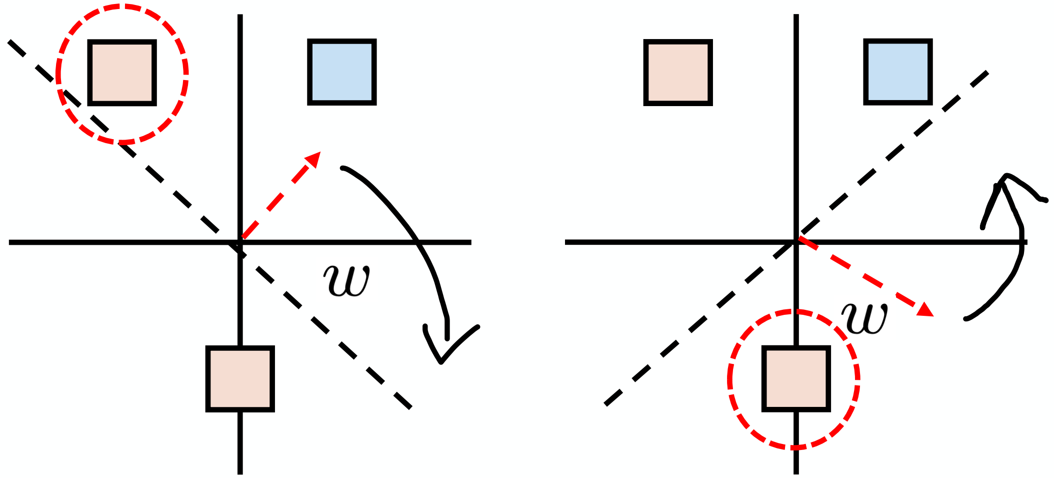
$$d = 1,\ y = 0$$

$$w^n = w^o + x_i^k$$



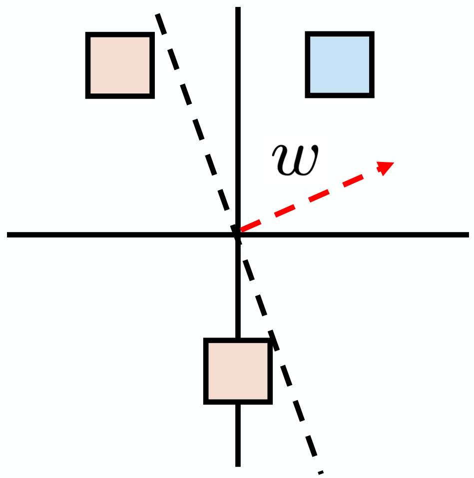
$$d = 0,\ y = 1$$

$$w^n = w^o - x_i^k$$



$$d = y$$

$$w^n = w^o$$



Neumann and Rosenblatt Perceptron

What is a perceptron example?

AND

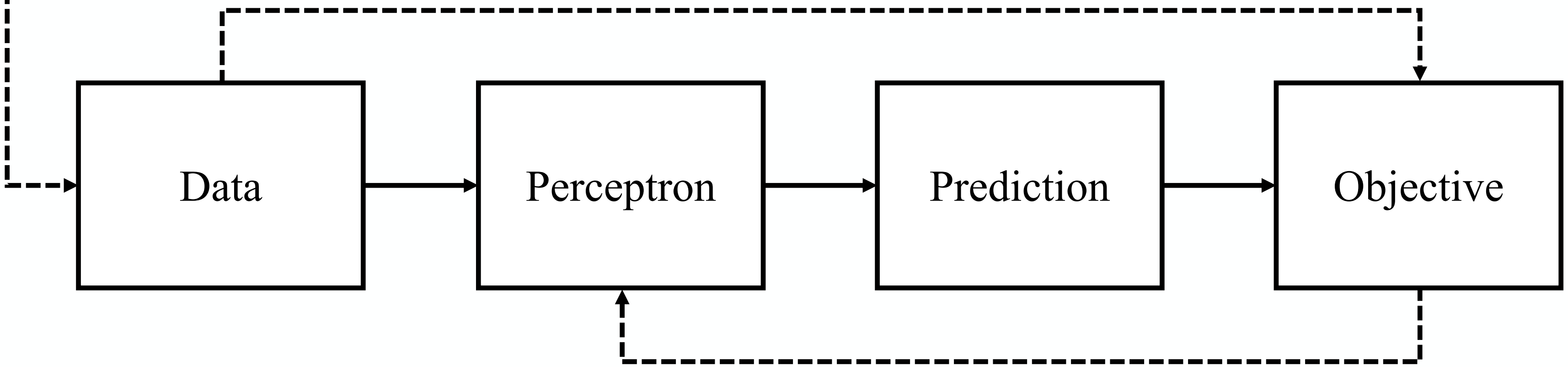
x_0	x_1	y
0	0	0
0	1	0
1	0	0
1	1	1

OR

x_0	x_1	y
0	0	0
0	1	1
1	0	1
1	1	1

XOR

x_0	x_1	y
0	0	0
0	1	1
1	0	1
1	1	0



Examples of perceptron performance on linear and non-linear problems

Neumann and Rosenblatt Perceptron

What are some perceptron example?

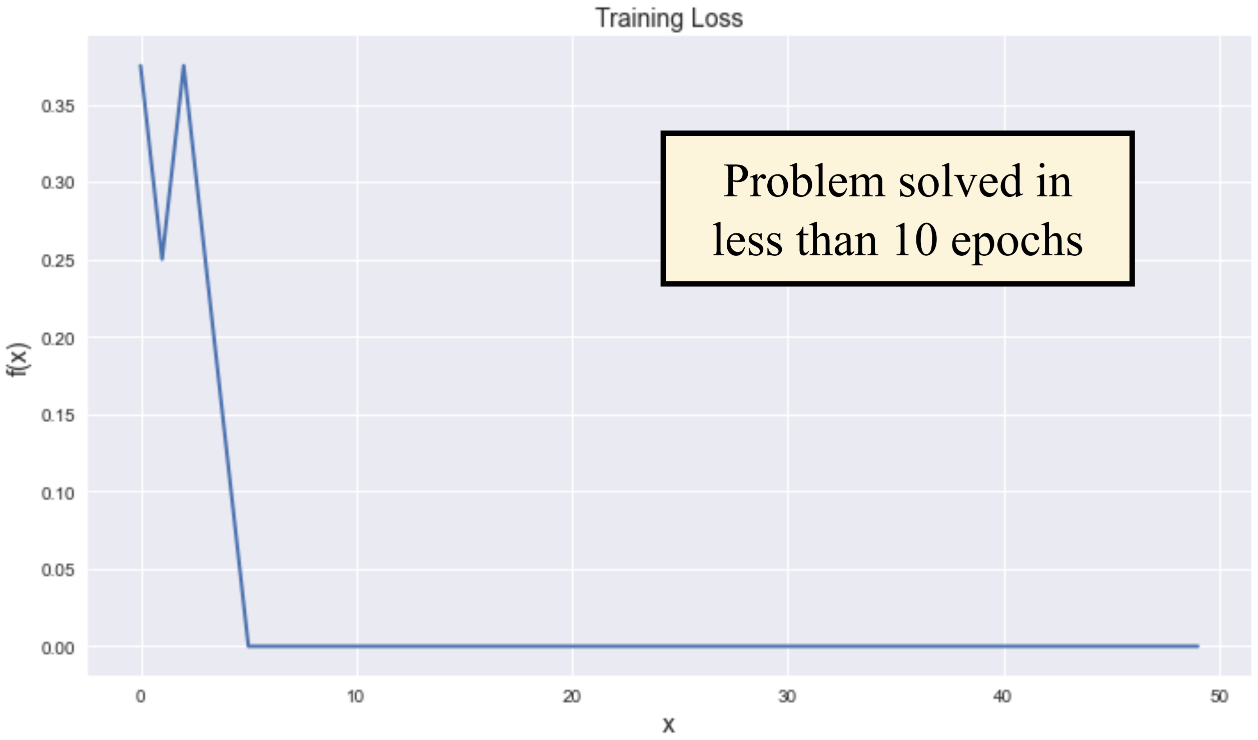
x_0	x_1	y
0	0	0
0	1	0
1	0	0
1	1	1

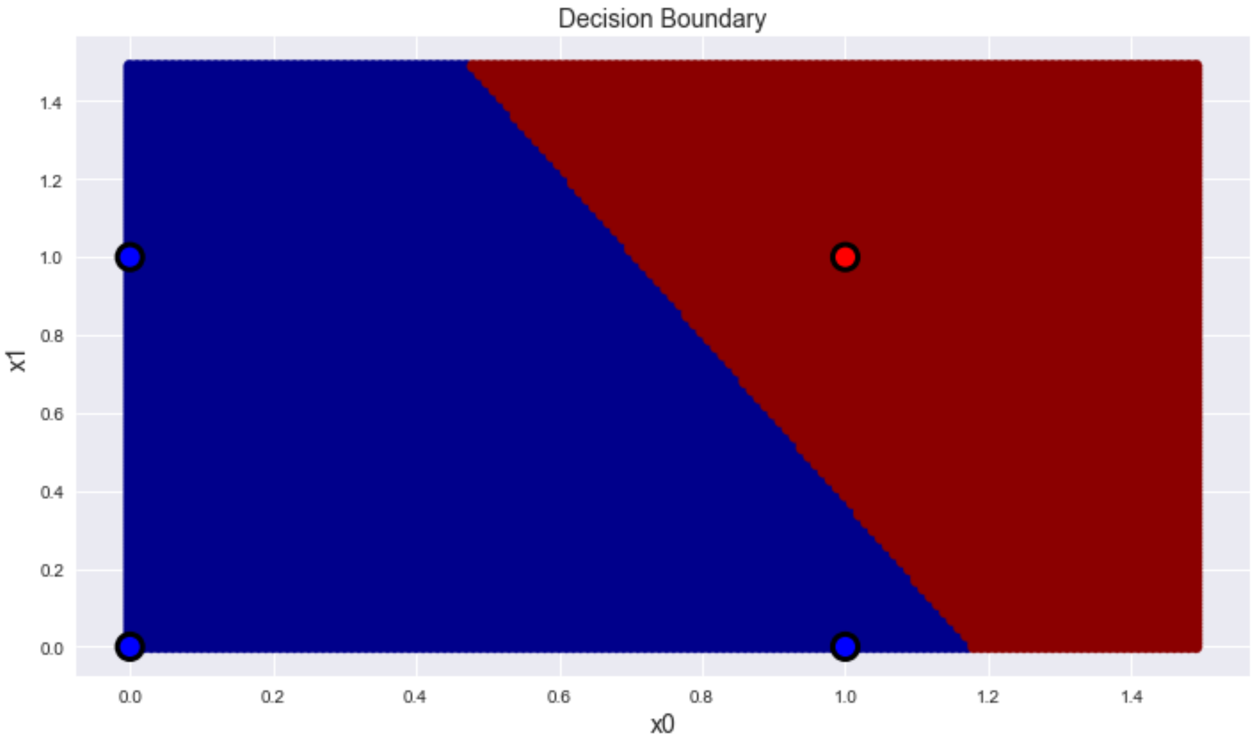
Dataset: AND problem

Epochs: 50

Parameter initialization: Random uniform [0, 1]

Training Method: Perceptron Learning Rule





Neumann and Rosenblatt Perceptron

What are some perceptron example?

x_0	x_1	y
0	0	0
0	1	1
1	0	1
1	1	1

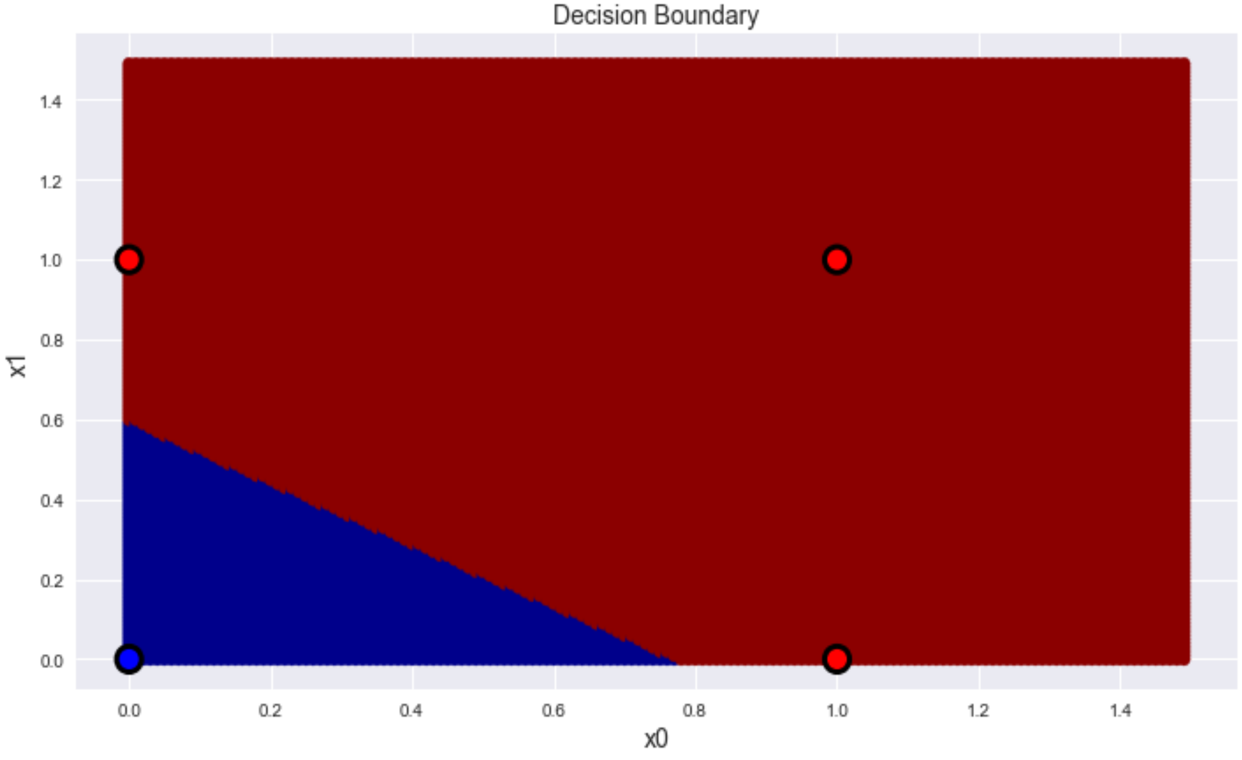
Dataset: OR problem

Epochs: 50

Parameter initialization: Random uniform [0, 1]

Training Method: Perceptron Learning Rule

Problem solved in less than 5 epochs



Neumann and Rosenblatt Perceptron

What are some perceptron example?

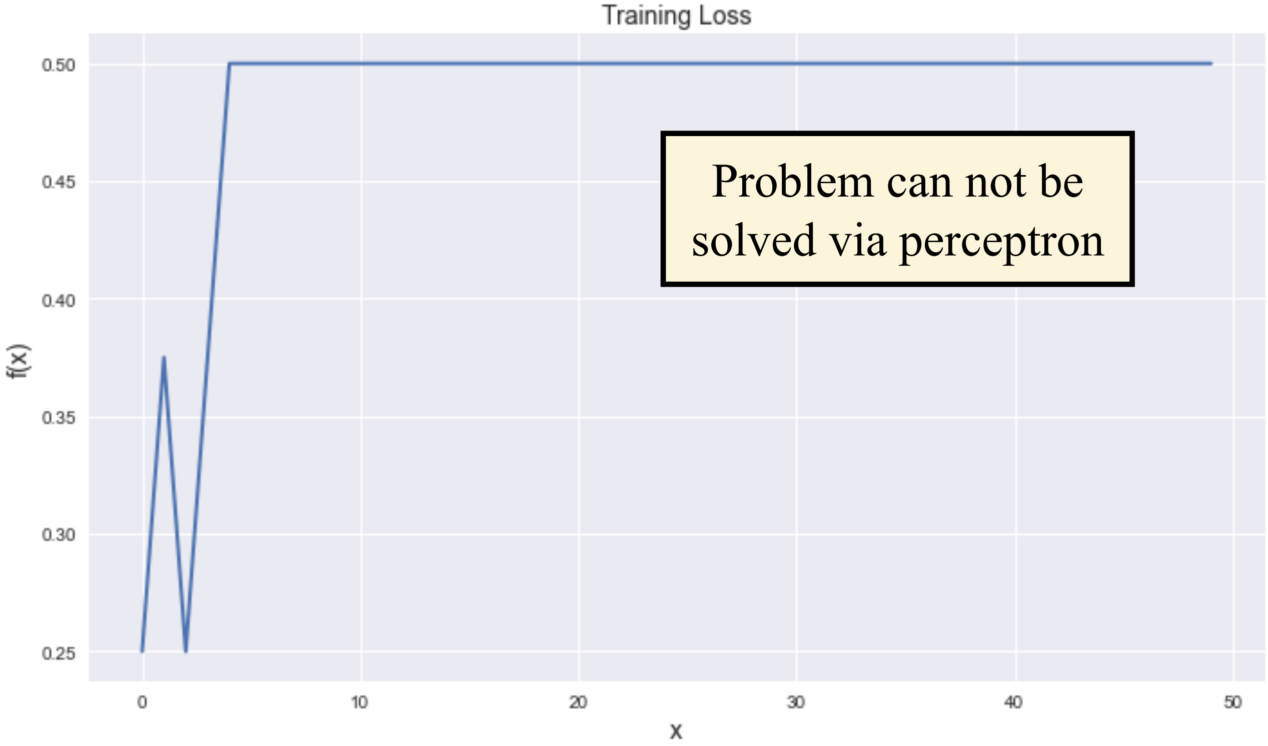
x_0	x_1	y
0	0	0
0	1	1
1	0	1
1	1	1

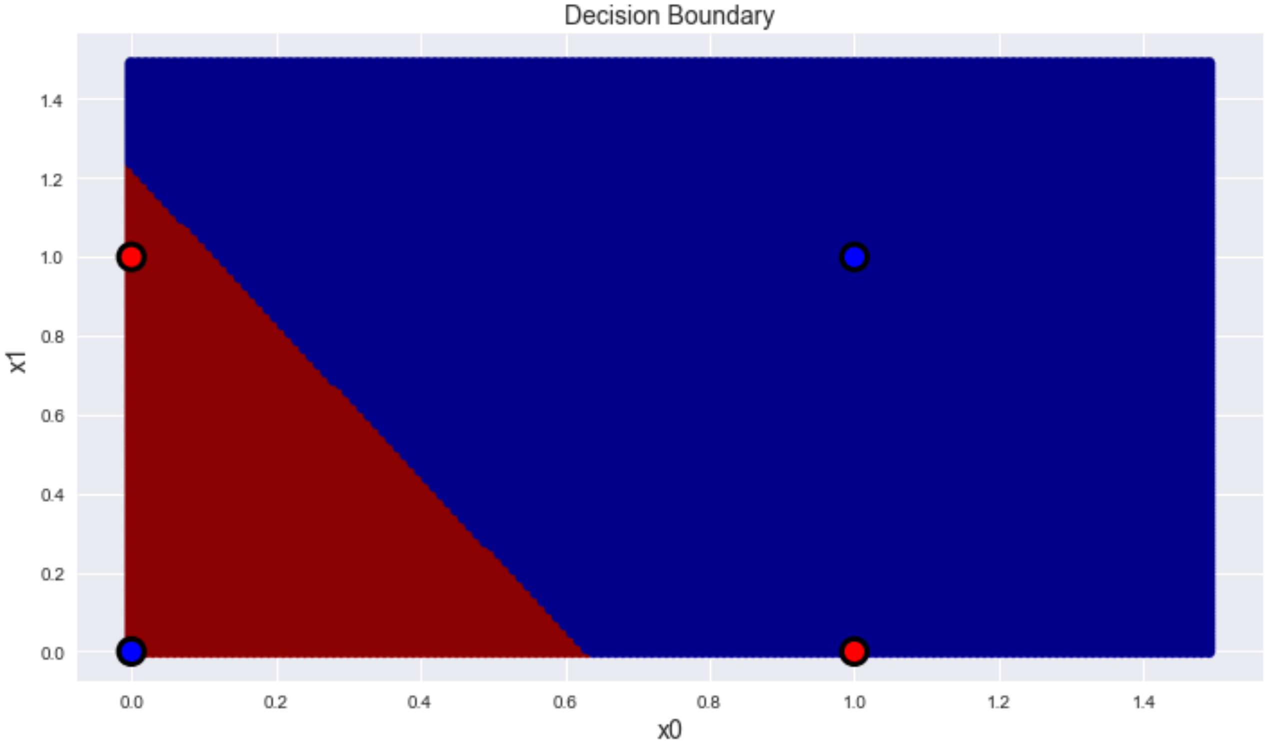
Dataset: XOR problem

Epochs: 50

Parameter initialization: Random uniform [0, 1]

Training Method: Perceptron Learning Rule



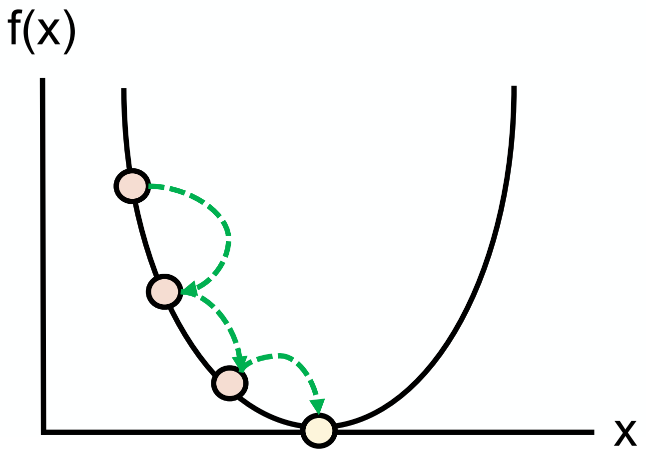


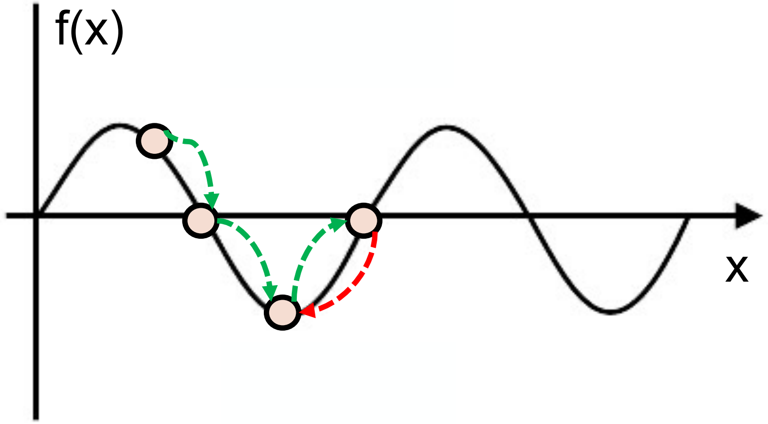
Neumann and Rosenblatt Perceptron

What is gradient descent for learning parameters?

Gradient Descent	Calculus based methodology to optimize a systems parameters given a differentiable function

Gradient	Vector representation of "slope" that points to direction of steepest ascent





Assume I wanted to find x that minimizes f(x)

Gradient descent can be used as an exploitation technique to discover ***local*** minima

However, unlike GAs, gradient descent has no exploration. This makes it vulnerable to problems with numerous minima

$$w^n = w^o - \boxed{\alpha}\boxed{\nabla w}$$

$$b^n = b^o - \boxed{\alpha}\boxed{\nabla b}$$

Subtracting the gradient can be used to discover function minimum

Parameter gradient : How a parameter changes regarding a differentiable function

Learning rate: determines the influence of the gradient to updating the parameter

$$\nabla w = \frac{\partial E}{\partial w} = \frac{\partial E}{\partial e}\frac{\partial e}{\partial y}\frac{\partial y}{\partial v}\frac{\partial v}{\partial w}$$

$$\nabla b = \frac{\partial E}{\partial b} = \frac{\partial E}{\partial e}\frac{\partial e}{\partial y}\frac{\partial y}{\partial v}\frac{\partial v}{\partial b}$$

Just partial derivatives!

Neumann and Rosenblatt Perceptron

What is gradient descent for learning parameters?

$$y = \sigma(b + x^k w^T)$$

Using the equation of a perceptron, we calculate a prediction

$$E = \frac{1}{2}(d - y)^2 = \frac{1}{2}e^2$$

Gradient based learning requires an explicit differentiable objective function

$$w^n = w^o - \alpha \nabla w$$

$$\nabla w = \frac{\partial E}{\partial w} = \boxed{\frac{\partial E}{\partial e}}\boxed{\frac{\partial e}{\partial y}}\boxed{\frac{\partial y}{\partial v}}\boxed{\frac{\partial v}{\partial w}}$$

$$b^n = b^o - \alpha \nabla b$$

$$\nabla b = \frac{\partial E}{\partial b} = \boxed{\frac{\partial E}{\partial e}}\boxed{\frac{\partial e}{\partial y}}\boxed{\frac{\partial y}{\partial v}}\boxed{\frac{\partial v}{\partial b}}$$

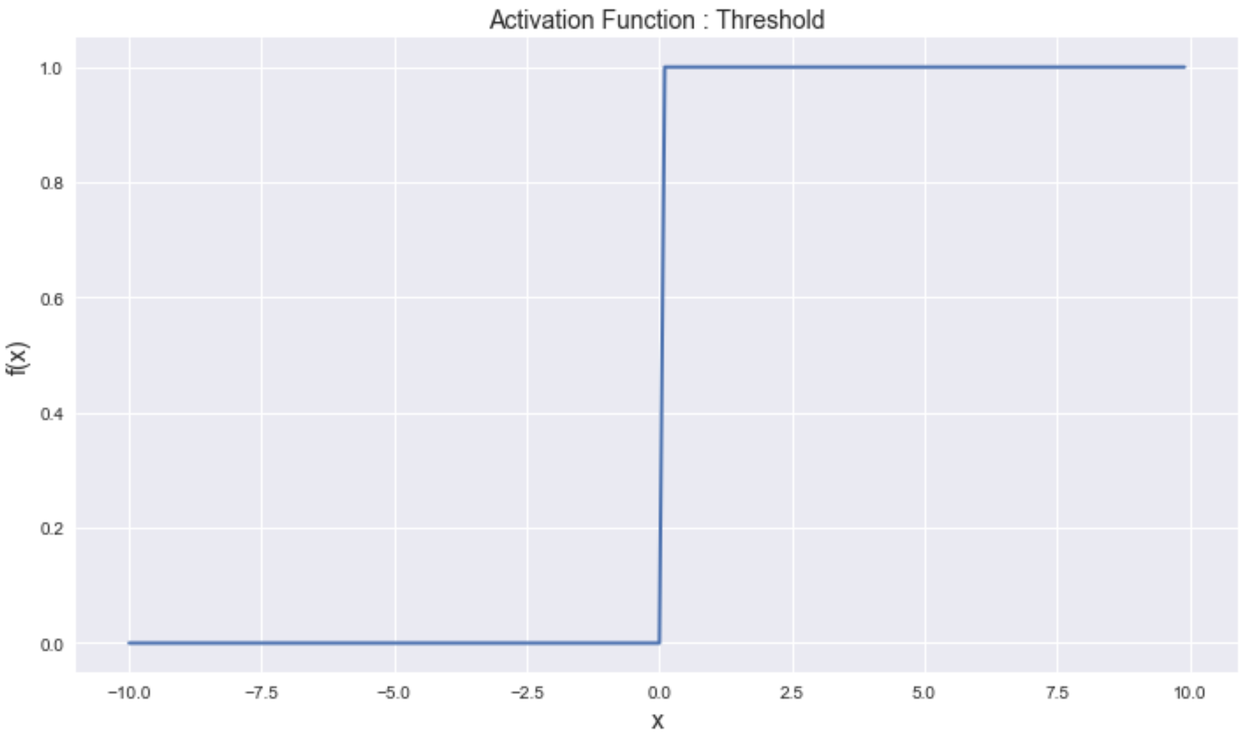
Update parameters w.r.t. how values minimize the objective function

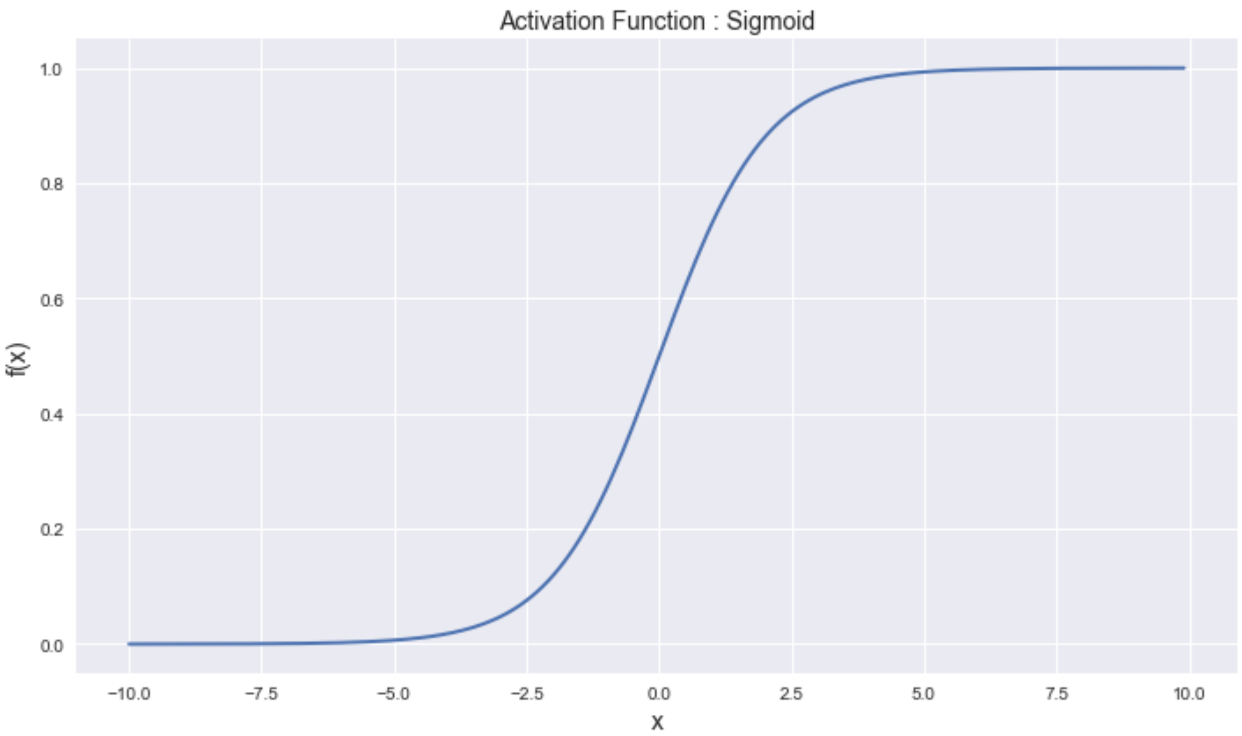
$$\frac{\partial E}{\partial e} = e \qquad \frac{\partial e}{\partial y} = -1 \qquad \frac{\partial y}{\partial v} = \boxed{\sigma'(v)} \qquad \frac{\partial v}{\partial w} = x \qquad \frac{\partial v}{\partial b} = 1$$

What activation function should be used?

Neumann and Rosenblatt Perceptron

What is gradient descent for learning parameters?





$$\sigma(v,t) = \begin{cases} 1 & \text{if } v > t \\ 0 & \text{otherwise} \end{cases}$$

$$\sigma(v) = \frac{1}{(1+e^{-v})} \quad \sigma'(v) = 1 - \sigma(v)$$

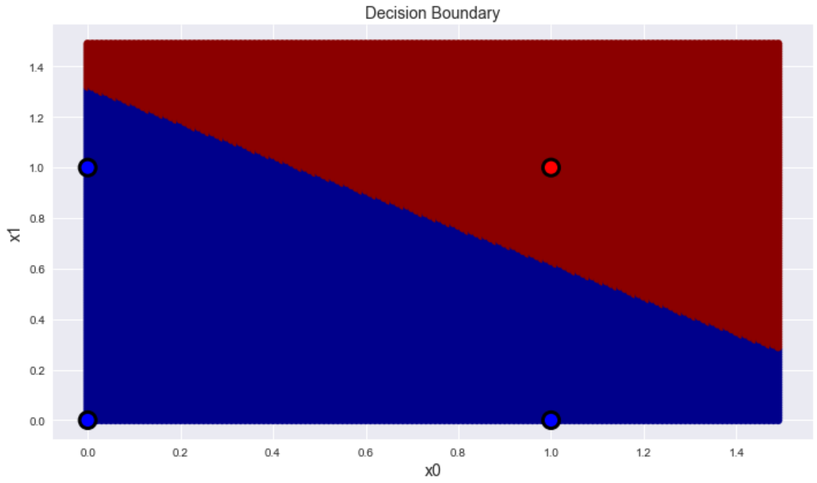
Step function is not completely differentiable

Sigmoid function is differentiable, continuous, and monotonically increasing

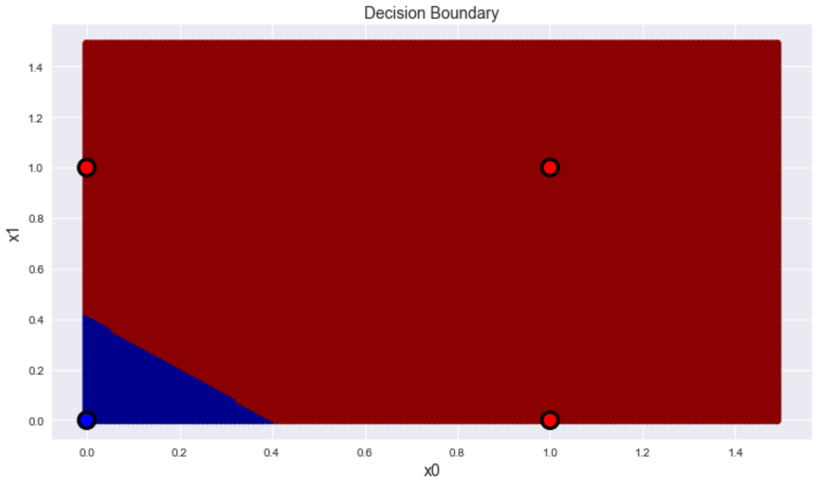
Neumann and Rosenblatt Perceptron

What are some perceptron example?

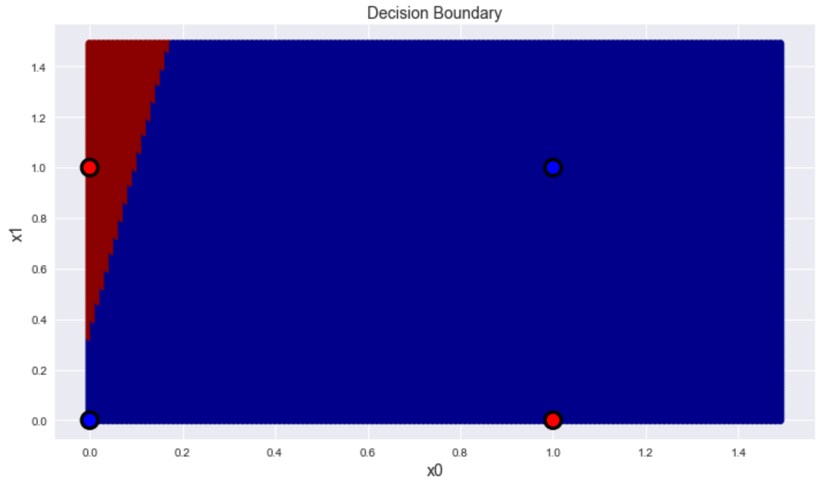
Dataset: AND problem



Dataset: OR problem

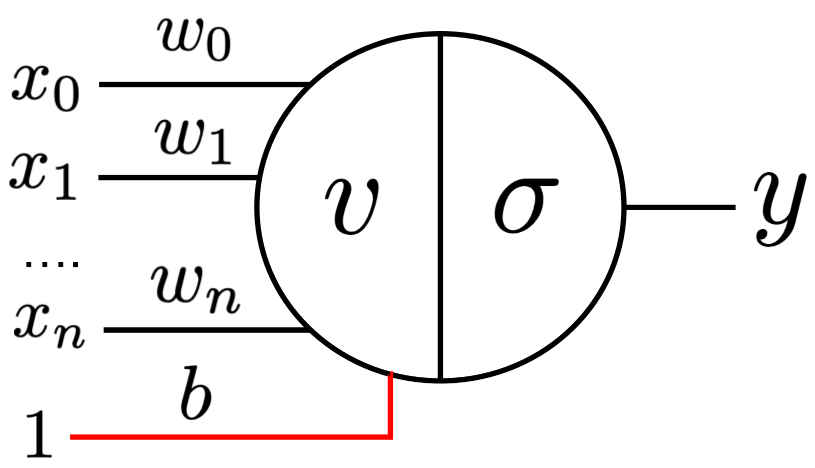


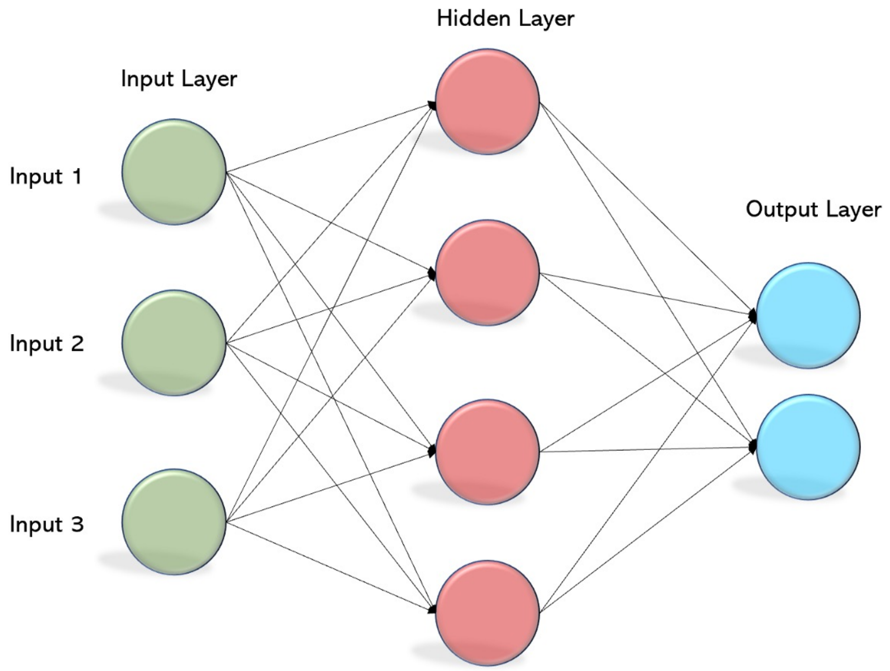
Dataset: XOR problem



The Multi Layer Perceptron (MLP)

What is the Multi-Layer-Perceptron?





Perceptron is a classification system that can solve any linear separable problem

Buuuuuut….. How many real-world problems are linearly separable? Not a lot of them ☺

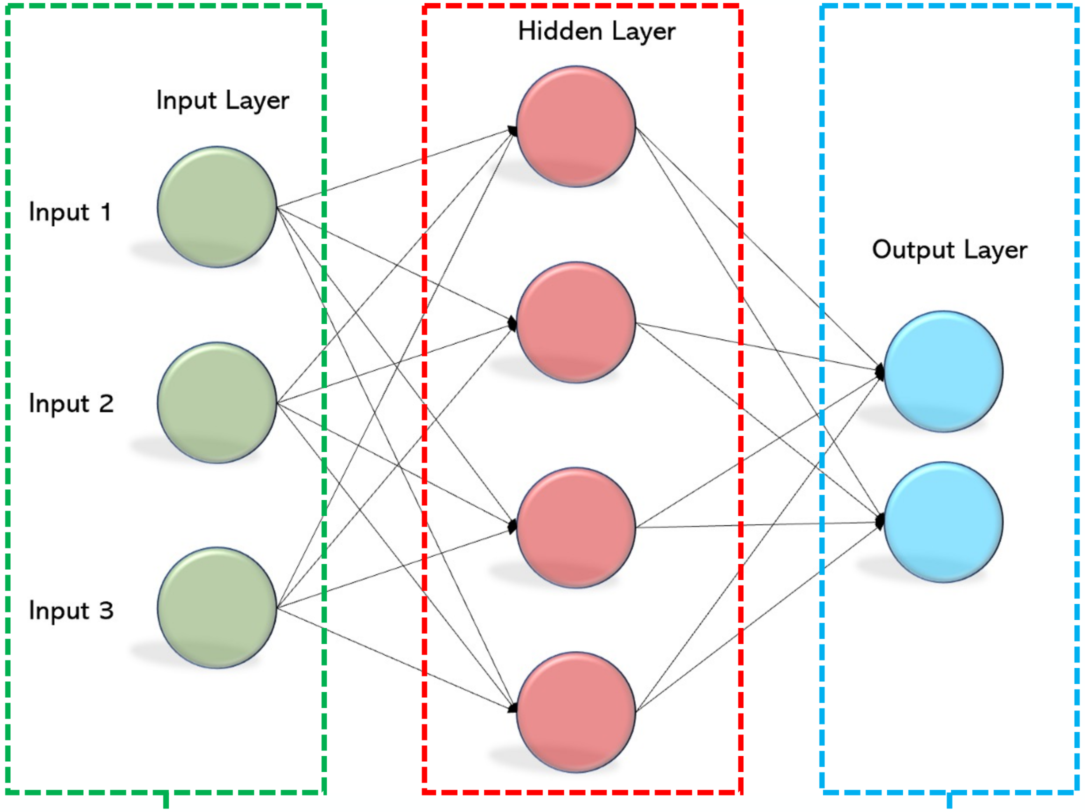
What happens if we create a system of neurons (perceptrons), each with differentiable non-linear activation functions?

MLP is a system of **fully-connected** neurons, each with their own learnable parameters (i.e., weights, bias)

MLP is commonly defined by its number of layers and number of neurons in each layer.

The Multi Layer Perceptron (MLP)

What is the Multi-Layer-Perceptron?



Input Layer	**This layer represents data** flowing into the MLP. Specifically, each node is a feature of an observation (just like the perceptron!)
Hidden Layer(s)	The primary computational layers of the MLP. Each hidden layer has a set of neurons. Each neuron has its own optimizable parameters.
Output Layer	**This layer represents predictions** of the MLP. Each neuron of this layer has its own optimizable parameters.

Sensor Readings

….

Simulation

Classification

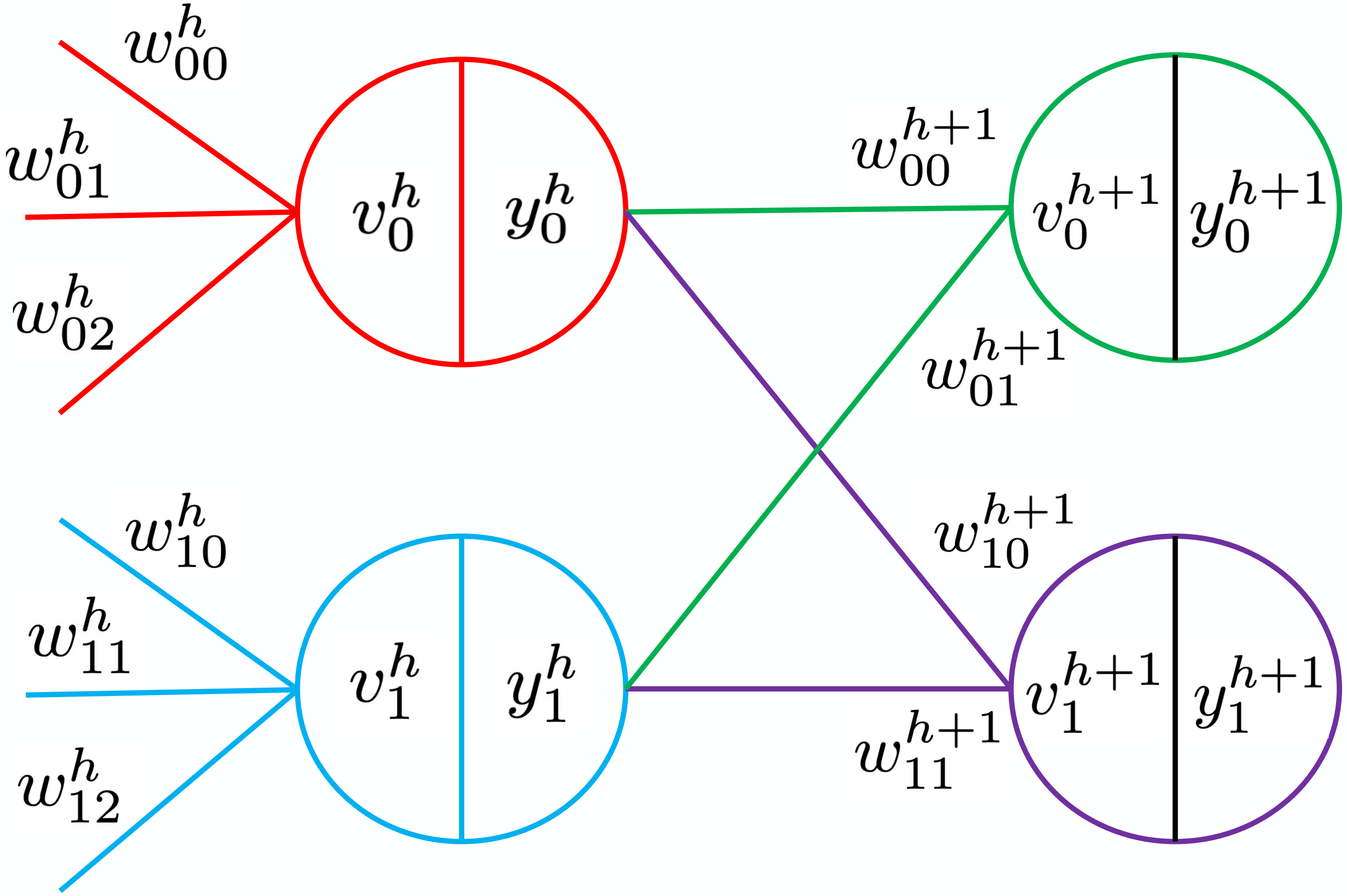
Regression

The Multi Layer Perceptron (MLP)

What is the Multi-Layer-Perceptron?

HL: ***h***

OL: ***h + 1***

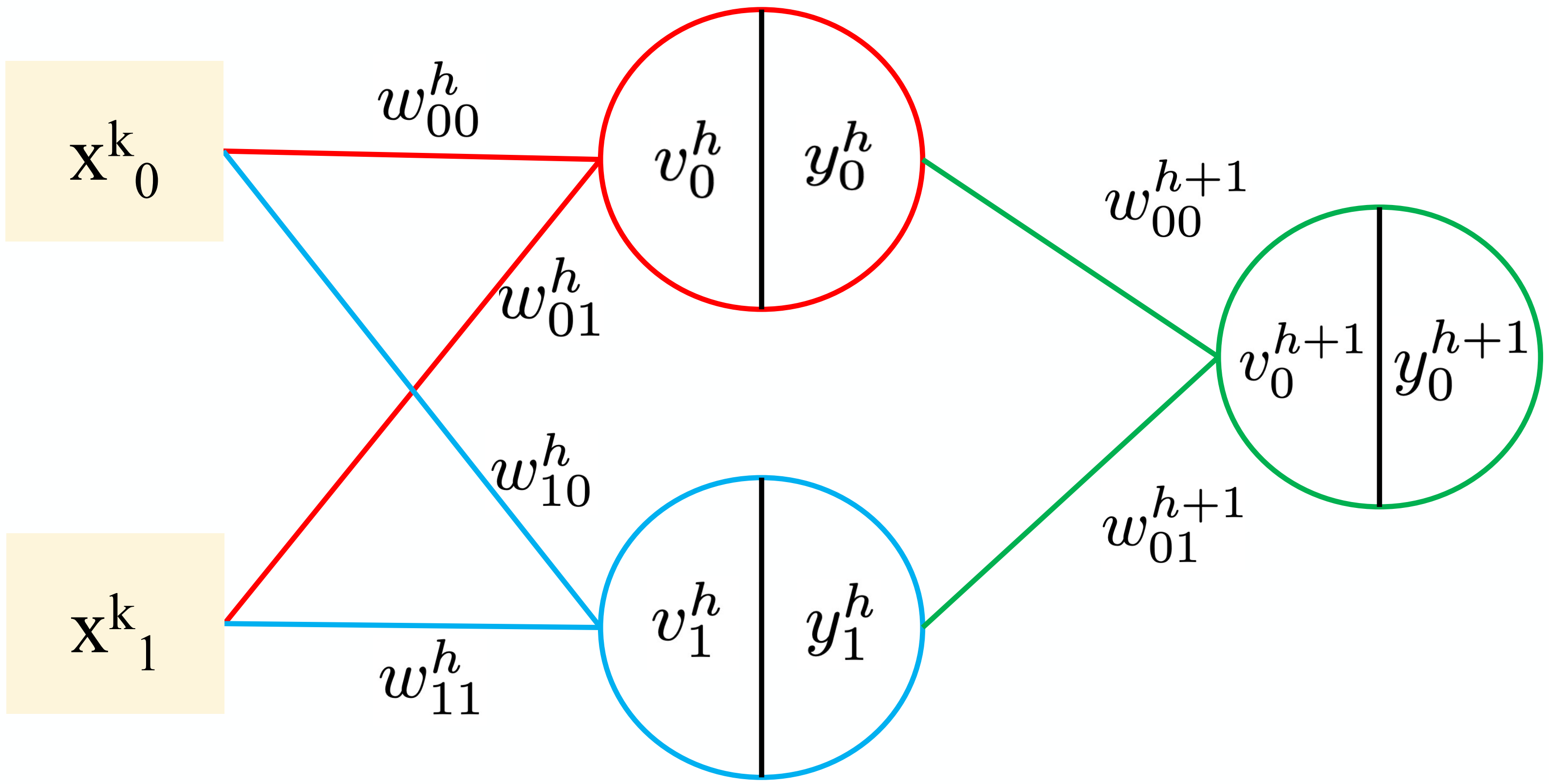


w_{ji}^{h}	Represents the ***i-th*** weight of neuron ***j*** inside layer ***h***
v_j^h	Represents the linear combination of neuron ***j*** inside layer ***h***
y_j^h	Represents the prediction of neuron ***j*** inside layer ***h***

Fully Connected Layers	For any layer (***h***), each neurons output is connected to all neurons of the next later (***h + 1***)

The Multi Layer Perceptron (MLP)

How does MLP make a prediction?



The input layer contains observations $\mathbf{x}^k$ from some dataset $\mathbf{X}$. Assume one single observation with 2 features $(\mathbf{x}^k{}_0, \mathbf{x}^k{}_1)$

Starting at first hidden layer, predictions propagate forward through each layer

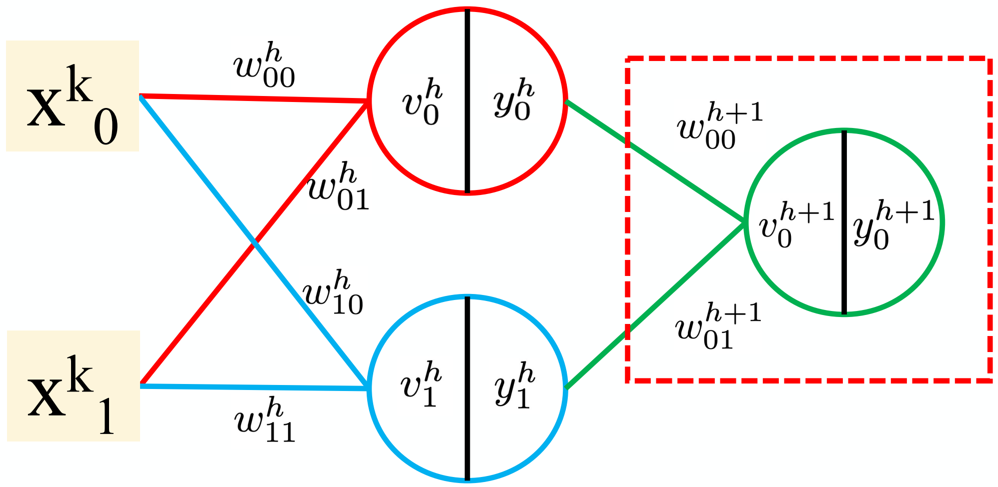
The output layer represents the final predictions of the MLP

Forward Pass	For each layer, all neurons make predictions from observing the outputs of neurons from previous layers.

Electrical Engineering & Computer Science
University of Missouri

The Multi Layer Perceptron (MLP)

How does MLP evaluate its prediction?



Each neuron of the output layer contributes to MLP prediction

Perceptron is limited to the classification predictions {0, 1}. The MLP can make **z** predictions for classification or regression

$$E = \frac{1}{2}\sum_{j=0}^{z}(\boxed{d_j} - \boxed{y_j})^2 = \frac{1}{2}\sum_{j=0}^{z}\boxed{e_j^2}$$

Truth label element j	Pred of neuron j	Error of neuron j

Truth labels can be non-scalars for both classification and regression problems

For classification, these are One-Hot-Labels (OHL)

Classification Labels

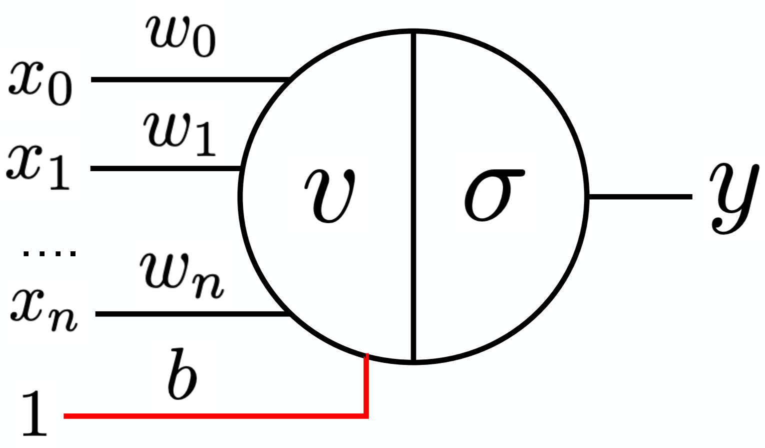
Class	Label	OHL
Cat	0	[1, 0, …, 0]
Dog	1	[0, 1, …, 0]
….	…	….
Person	c	[0, 0, …, 1]

Regression Labels

Variable	Label
Temperature	78.3
Time (Hours)	13
….	…
Humidity	51

The Multi Layer Perceptron (MLP)

How do we update MLP parameters?

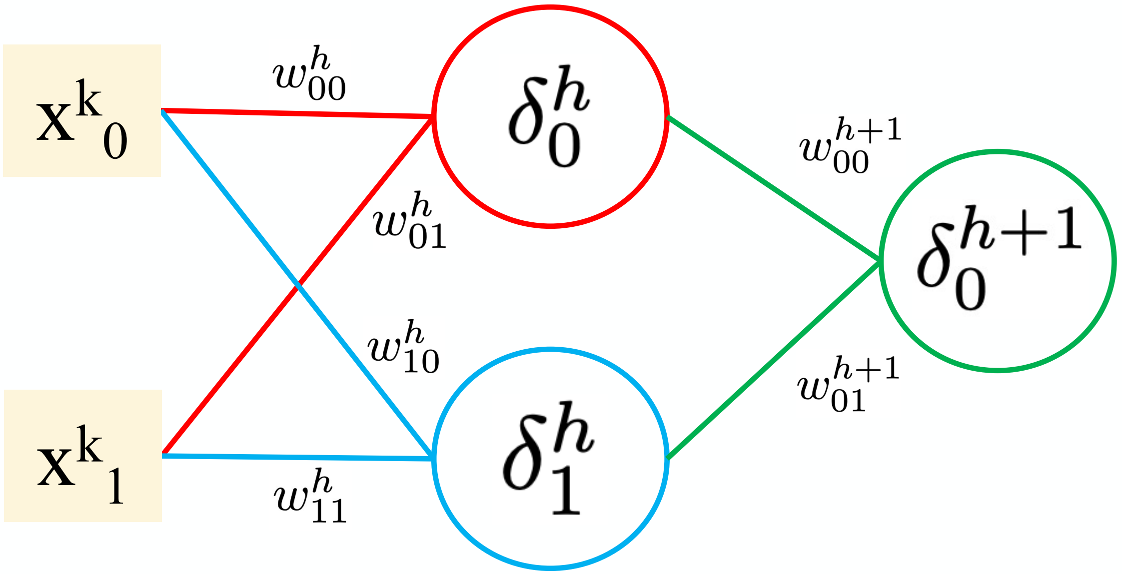


$$\nabla w = \frac{\partial E}{\partial w} = \boxed{\frac{\partial E}{\partial e}\frac{\partial e}{\partial y}\frac{\partial y}{\partial v}}\frac{\partial v}{\partial w}$$

$$\boxed{\delta = \frac{\partial E}{\partial e}\frac{\partial e}{\partial y}\frac{\partial y}{\partial v}}$$

$$\nabla w = \delta\frac{\partial v}{\partial w} \qquad \nabla b = \delta\frac{\partial v}{\partial b}$$

δ represents the local gradient at the neuron

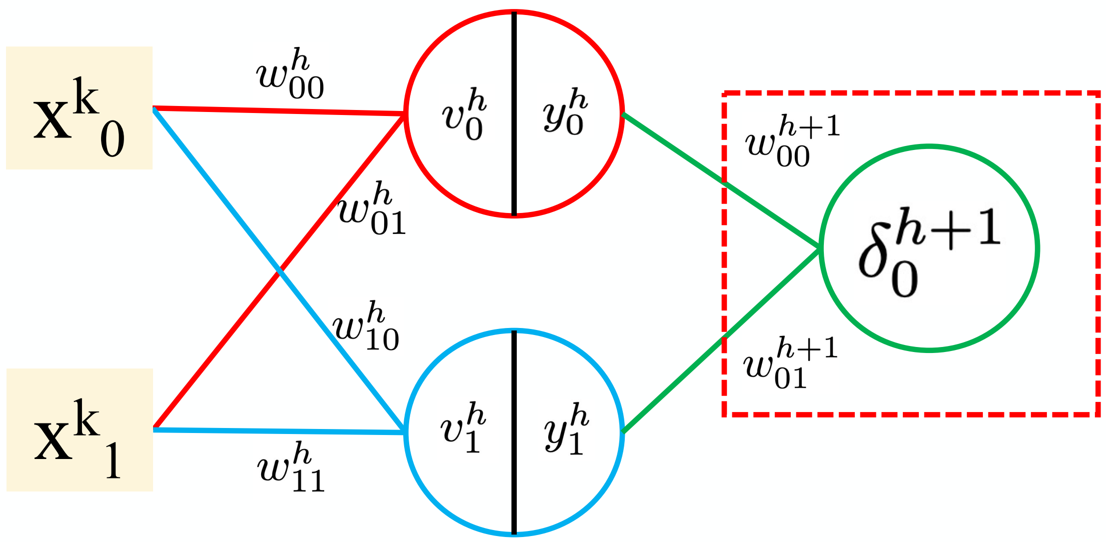


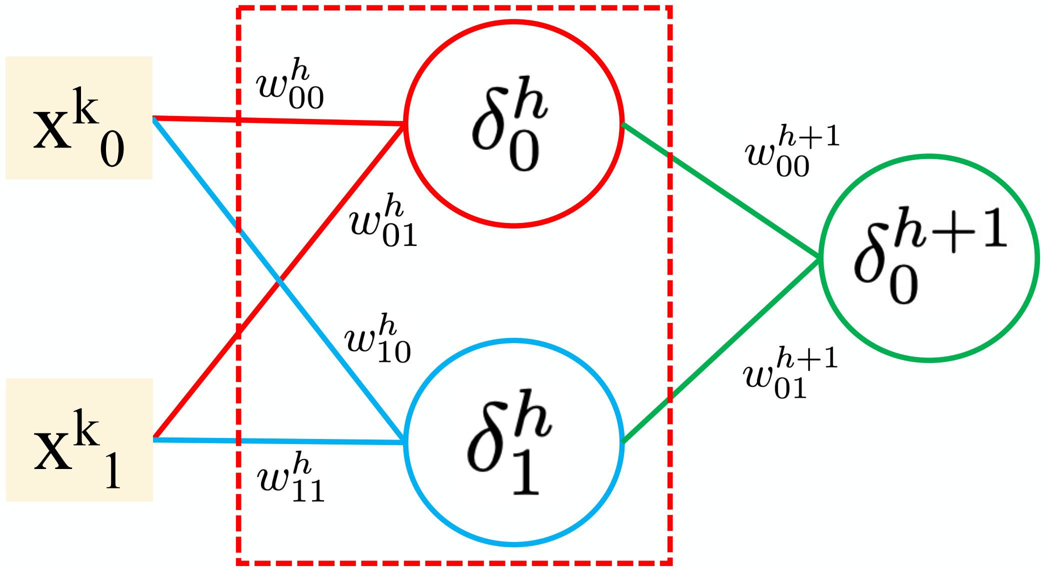
δ^h_j	Represents the local gradient of neuron ***j*** inside layer ***h***
Backpropagation	Calculating gradients of each MLP neuron going **backwards** through the network
Gradient Descent	For each neuron, use local gradient information to update its parameters

The Multi Layer Perceptron (MLP)

How do we update MLP parameters?

How does backpopagation calculate gradient information at each neuron?



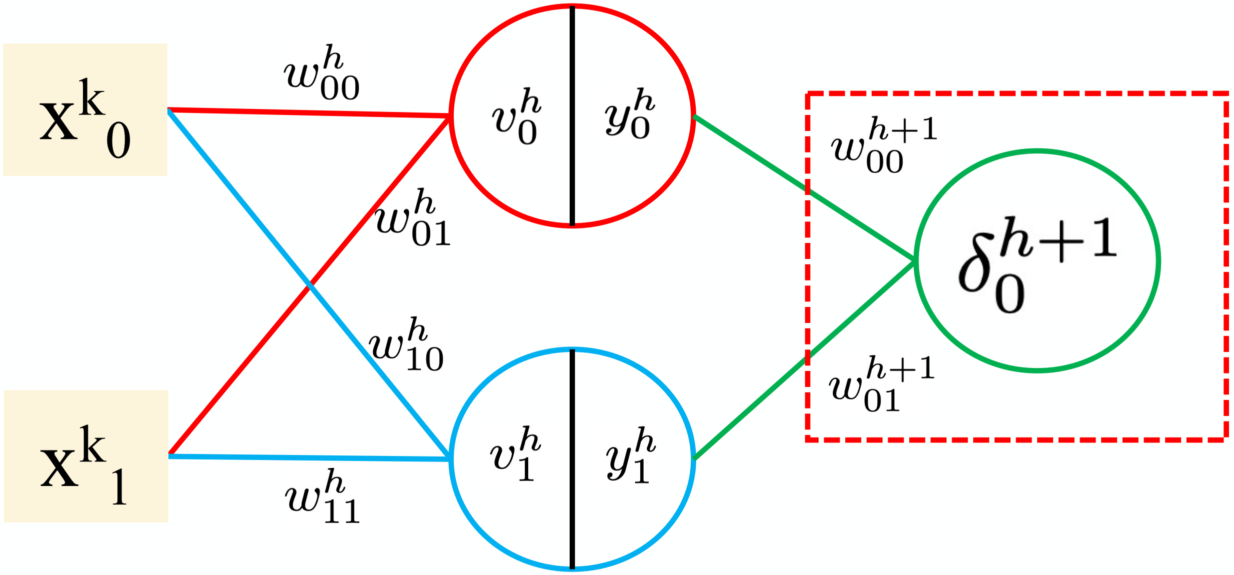


Gradients for Output Layer Neurons	Calculating gradients of output layer neurons is identical to the perceptron!

Gradients for Hidden Layer Neurons	Calculating gradients of hidden layer neurons requires gradient info from next layer (h + 1)

The Multi Layer Perceptron (MLP)

How do we update MLP parameters?



$$E = \frac{1}{2}\sum_{j=0}^{z}(d_j - y_j)^2 = \frac{1}{2}\sum_{j=0}^{z} e_j^2$$

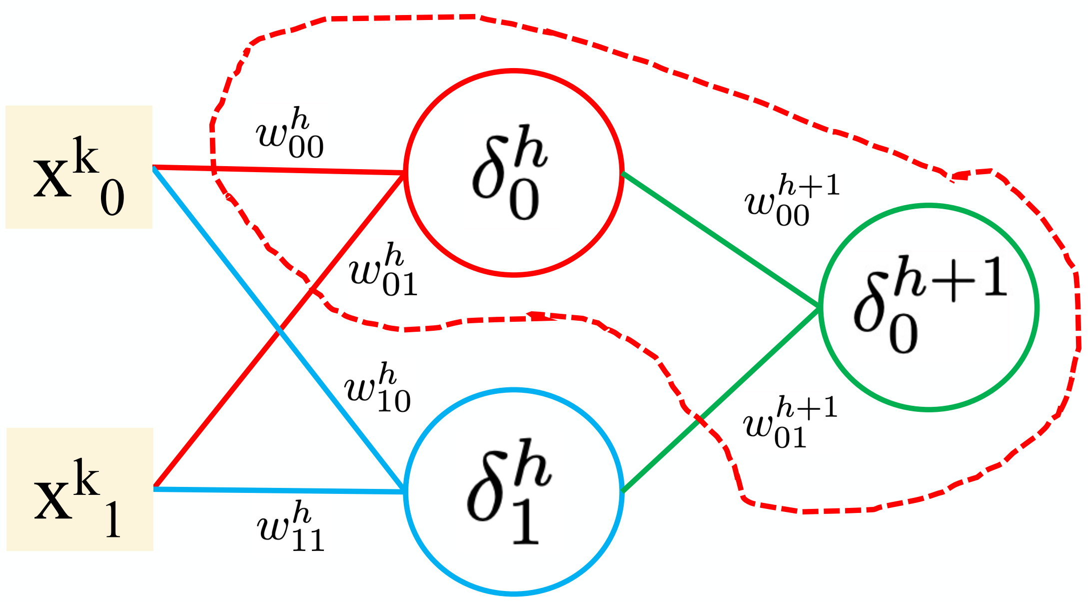
Objective Function

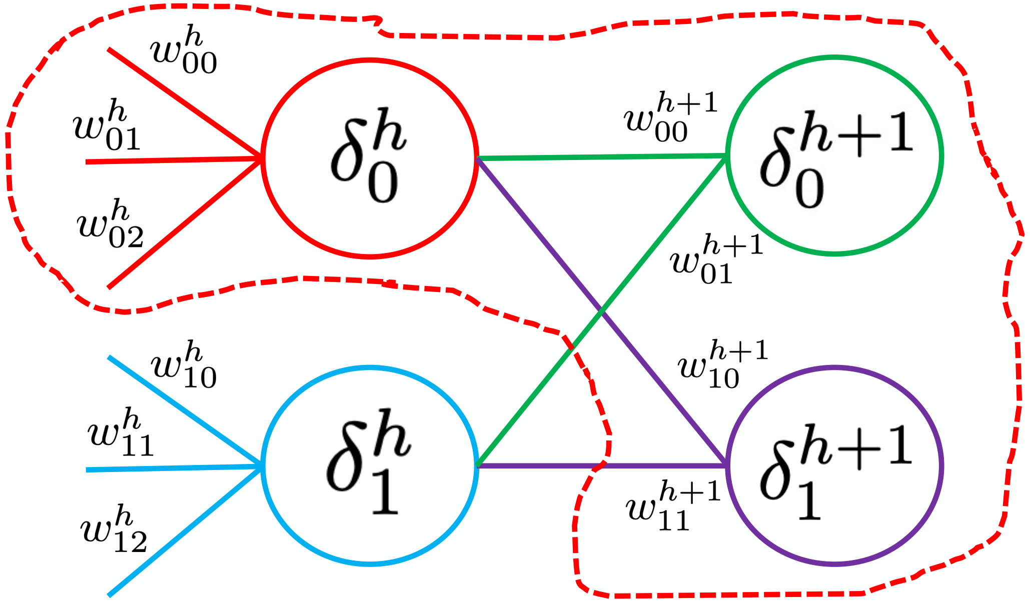
$$\delta_0^{h+1} = \frac{\partial E}{\partial e_0}\frac{\partial e_0}{\partial y_0^{h+1}}\frac{\partial y_0^{h+1}}{\partial v_0^{h+1}}$$

Similar to the perceptron, the local gradient of a output neuron is calculated by directly using the neuron's error associated with the system prediction

The Multi Layer Perceptron (MLP)

How do we update MLP parameters?





$$\delta^h_j = \sum_{i=0}^{z^{h+1}} (w^{h+1}_{ij} \delta^{h+1}_{ij}) \frac{\partial y^h_j}{\partial v^h_j}$$

$$\delta^h_0 = \sum_{i=0}^{z^{h+1}} (w^{h+1}_{i0} \delta^{h+1}_{i0}) \frac{\partial y^h_0}{\partial v^h_0}$$

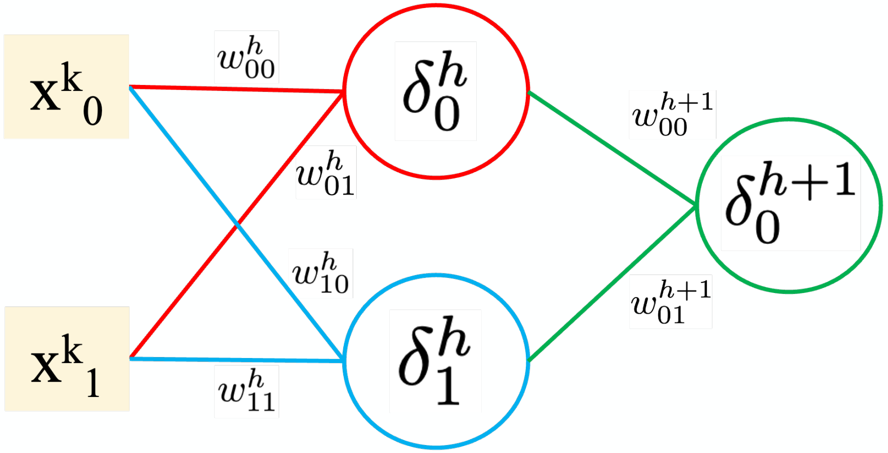
The local gradient of a hidden neuron is calculated by using the gradient information of the neuron in the next layer

Even though the MLP architecture is different, the update algorithm is the same!

The Multi Layer Perceptron (MLP)

How do we update MLP parameters?

Backpropagation	Calculate gradients of all neurons going **backwards** through the MLP



$$\delta_j^{h+1} = \frac{\partial E}{\partial e_j} \frac{\partial e_j}{\partial y_j^{h+1}} \frac{\partial y_j^{h+1}}{\partial v_j^{h+1}}$$

Gradients for Output Layer Neurons

$$\delta_j^h = \sum_{i=0}^{z^{h+1}} (w_{ij}^{h+1} \delta_{ij}^{h+1}) \frac{\partial y_j^h}{\partial v_j^h}$$

Gradients for Hidden Layer Neurons

Gradient Descent	For each neuron, use local gradient information to update its parameters

$$w_j^h = w_j^h - \alpha \nabla w_j^h \qquad b_j^h = b_j^h - \alpha \nabla b_j^h$$

$$\nabla w_j^{h+1} = \nabla w_j^h = \delta_j^h \frac{\partial v_j^h}{\partial w_j^h}$$

$$\nabla b_j^{h+1} = \nabla b_j^h = \delta_j^h \frac{\partial v_j^h}{\partial b_j^h}$$