Welcome to SEAS Online at George Washington University

Class will begin shortly

Audio: To eliminate background noise, please be sure your audio is muted. To speak, please click the hand icon at the bottom of your screen (**Raise Hand**). When instructor calls on you, click microphone icon to unmute. When you've finished speaking, ***be sure to mute yourself again***.

Chat: Please type your questions in Chat.

Recordings: As part of the educational support for students, we provide downloadable recordings of each class session to be used exclusively by registered students in that particular class. Releasing these recordings is strictly prohibited.

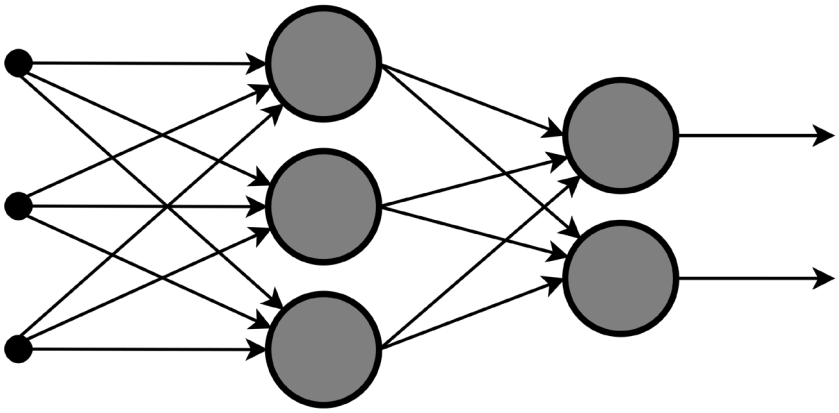
THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

SEAS 8520 – Lecture 3: Neural Networks

Walid Hassan, M.B.A, D.Eng.

Agenda

- ANN Intuition
- Gradient Descent
- Derivative of a Sigmoid Function
- Calculating the Derivatives
- Classification with a neural network
- Activation Functions
- Non-Linear Examples
- Vanishing, Exploding Gradient Descents and Dying ReLU Problem
- Gradient Descent Variants



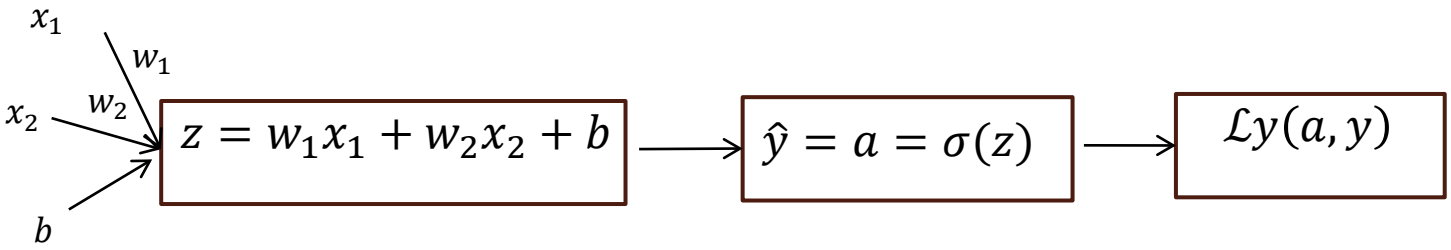
Artificial Neural Networks

ANN Intuition

Computational Graph
Forward Propagation on one training example

- $z = wx + b$
- $\hat{y} = a = \frac{1}{1+e^{-z}}$
- $\mathcal{L}(a, y) = -y * \log(a) - (1 - y) * \log(1 - a)$

Forward propagation:



Perceptron

- A type of artificial neuron that uses a binary step function as its activation function.

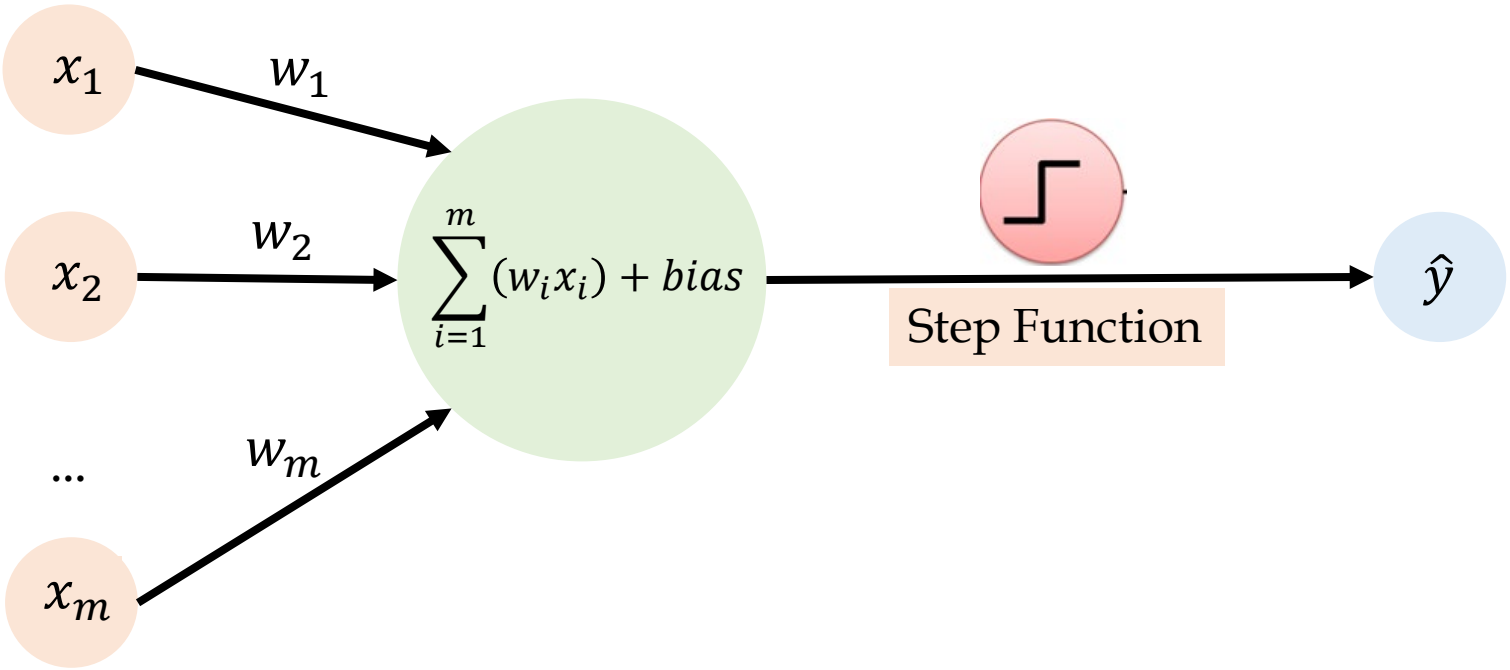
- Original algorithm developed by Frank Rosenblatt in 1957, was designed to classify inputs into one of two categories (e.g., 0 or 1) .

- If the weighted sum of the inputs exceeds a certain threshold, perceptron outputs a 1; otherwise, 0.

A simple perceptron can be formulated as

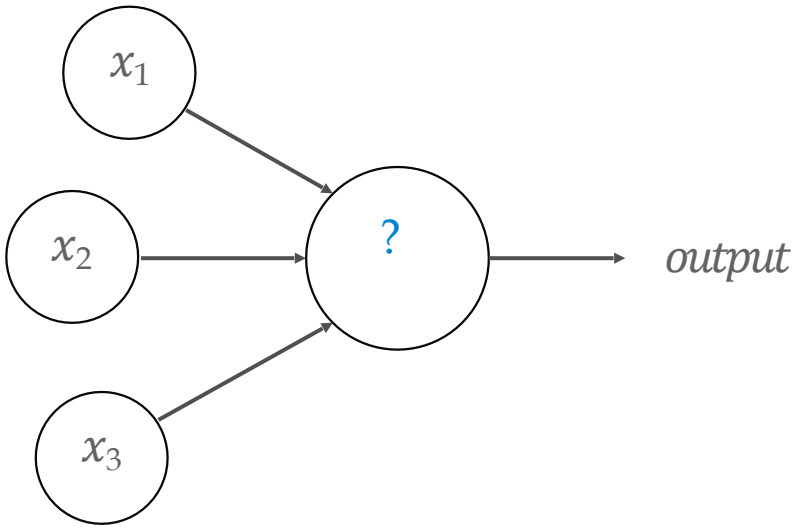
$$f(x) = \begin{cases} 1 & wx + b > 0 \\ 0 & otherwise \end{cases}$$

Perceptron



Perceptron

Simplified (binary) artificial perceptron



Do I go hiking this weekend?

$x_1 \rightarrow$ Is the weather suitable for hiking?

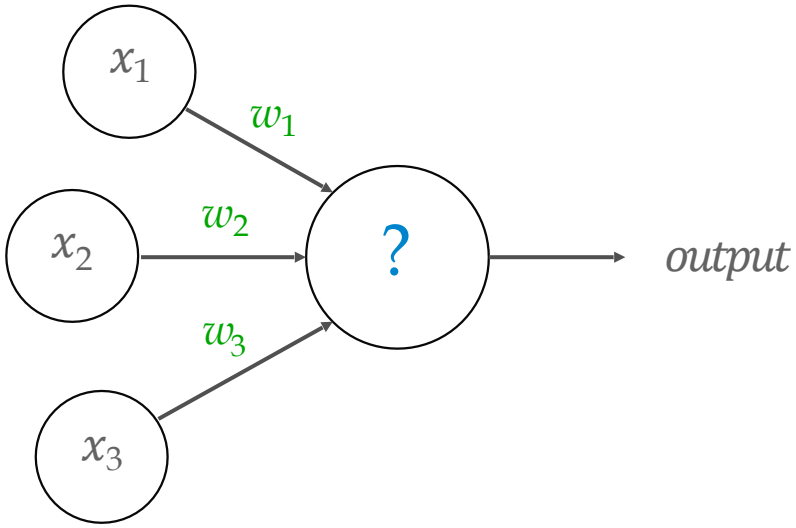
$x_2 \rightarrow$ Are the trail conditions good?

$x_3 \rightarrow$ Am I physically feeling up to the challenge?

Source; databricks - Deep Learning Fundementals

Perceptron

Simplified (binary) artificial perceptron with weights

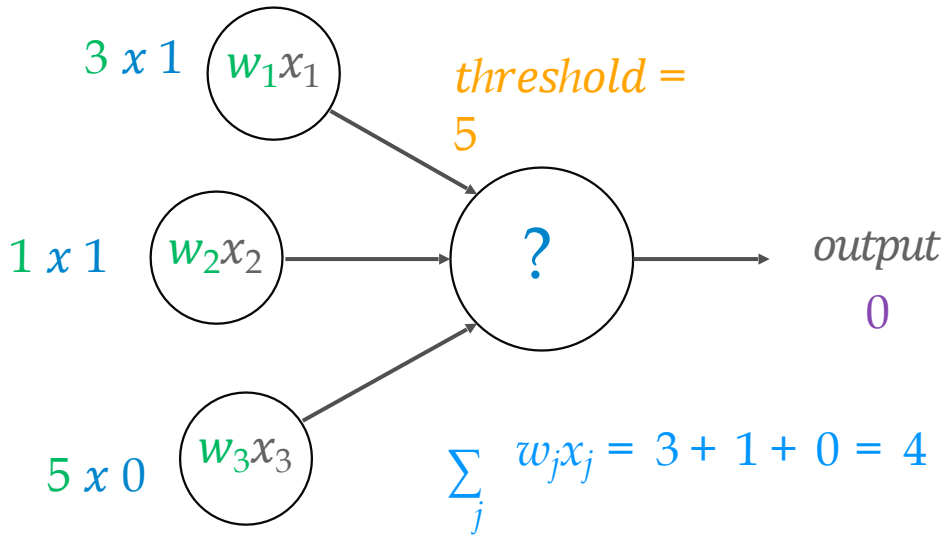


$$output = \begin{cases} 0, & \sum_{j=0}^{n} w_j x_j \leq threshold \\ 1, & \sum_{j=0}^{n} w_j x_j > threshold \end{cases}$$

Source; databricks - Deep Learning Fundementals

Perceptron

Simplified (binary) artificial perceptron;
Person: Pete Simpers



Do I go hiking this weekend?

$x_1 = 1$ (*good weather*)
$w_1 = 3$

$x_2 = 1$ (*trail conditions are good*)
$w_2 = 1$

$x_3 = 0$ (*Physically tired*)
$w_3 = 5$

Source; databricks - Deep Learning Fundementals

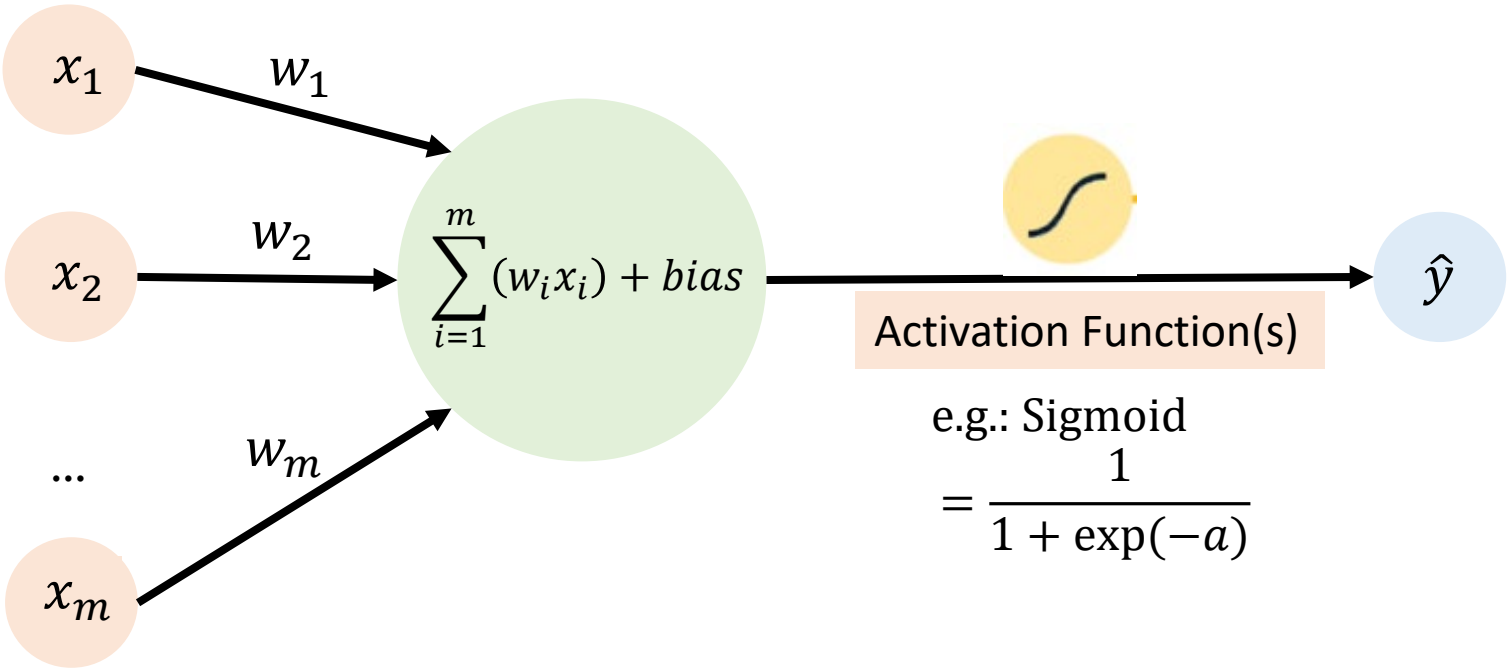
Artificial Neuron

- Artificial neuron used in contemporary neural networks, generalizes the perceptron idea by allowing for a variety of activation functions other than the step function.

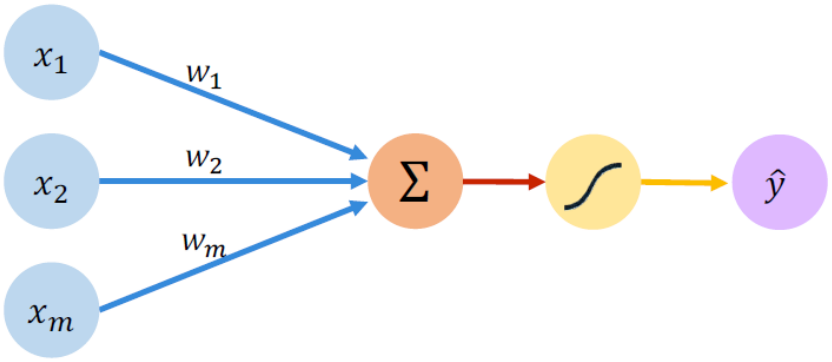
- Common examples include the sigmoid, hyperbolic tangent (tanh), ReLU (Rectified Linear Unit), and softmax functions.

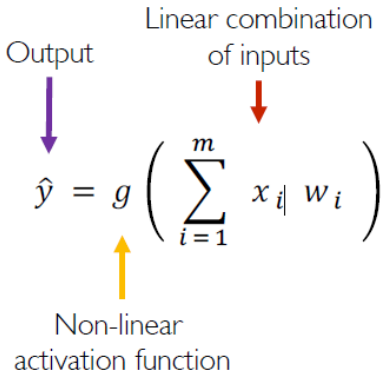
- These activation functions allow for the creation of neural networks that can handle complex, non-linearly separable problems, which a single-layer perceptron with a step function cannot.

Artificial Neuron



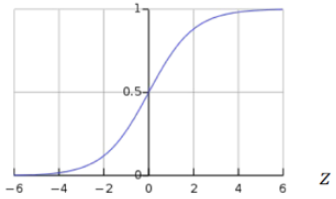
Artificial Neuron





- Example: sigmoid function

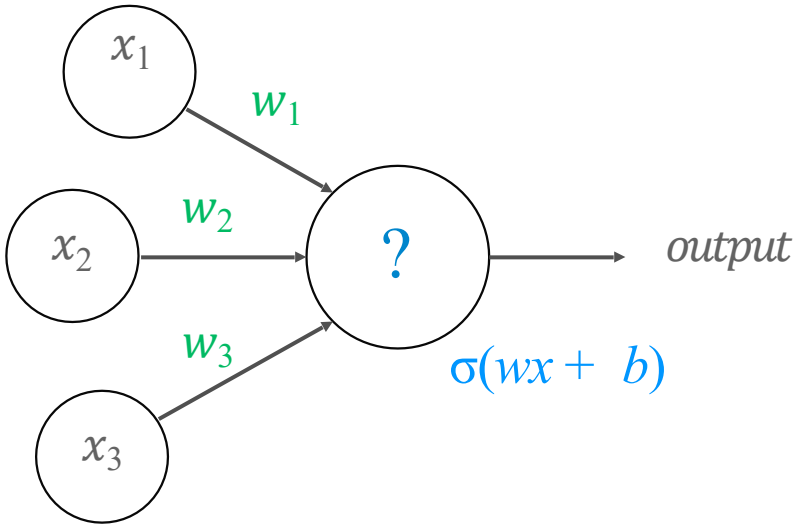
$$g(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$

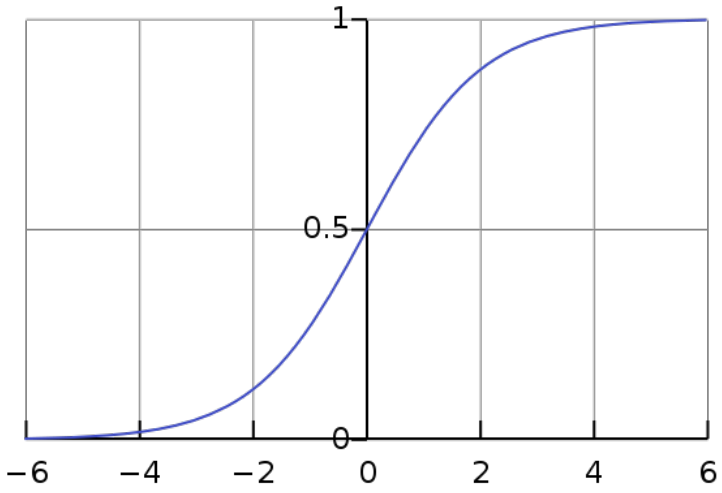


Source: MIT

Artificial Neuron

Sigmoid Activation





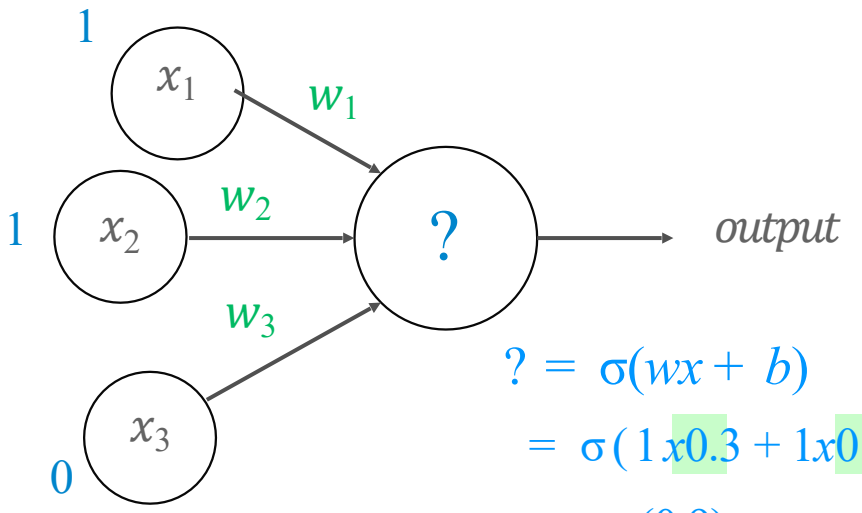
Instead of [0, 1], now (0…1)

Where output is defined by $\sigma(wx + b)$

Source; databricks - Deep Learning Fundementals

Artificial Neuron

Person: Shredder



$$? = \sigma(wx + b)$$
$$= \sigma(1x0.3 + 1x0.6 + 0x0.1)$$
$$= \sigma(0.9)$$
$$= 0.7109$$

Do I go hiking this weekend?

$x_1 = 1$ (*good weather*)
$w_1 = 0.3$

$x_2 = 1$ (*trail conditions are good*)
$w_2 = 0.6$

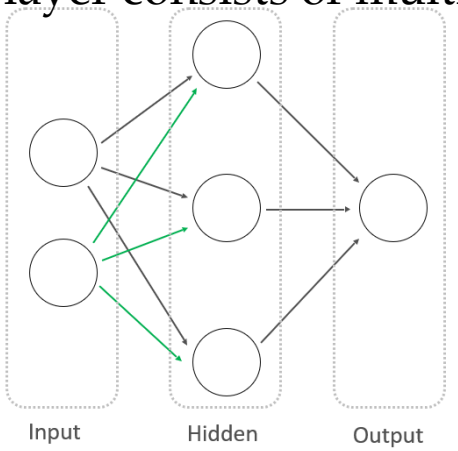
$x_3 = 0$ (*physically tired*)
$w_3 = 0.1$

Source; databricks - Deep Learning Fundementals

Artificial Neural Network

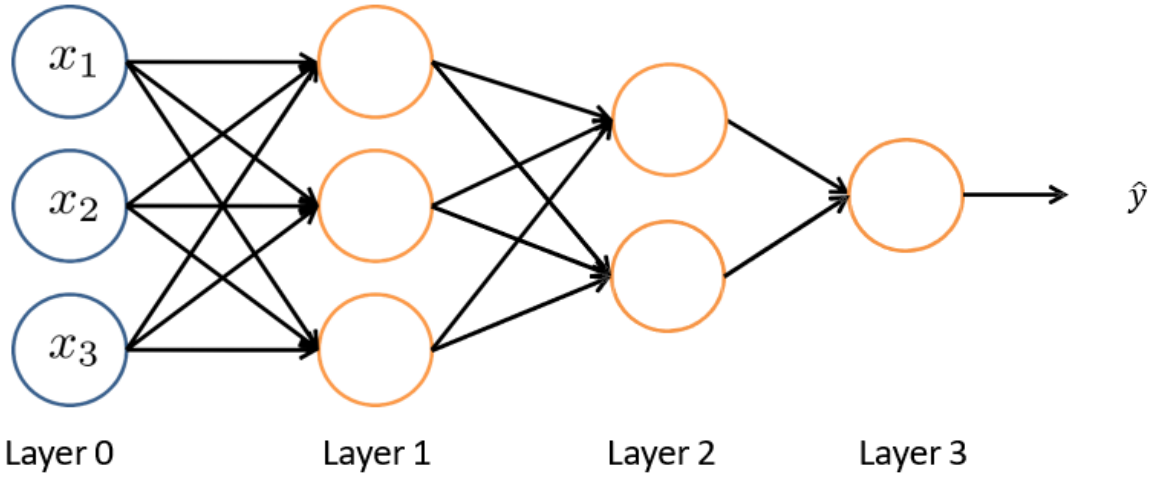
An ANN is a network of interconnected artificial neurons. It's a system that collectively processes complex data inputs to perform tasks like classification, regression, etc.

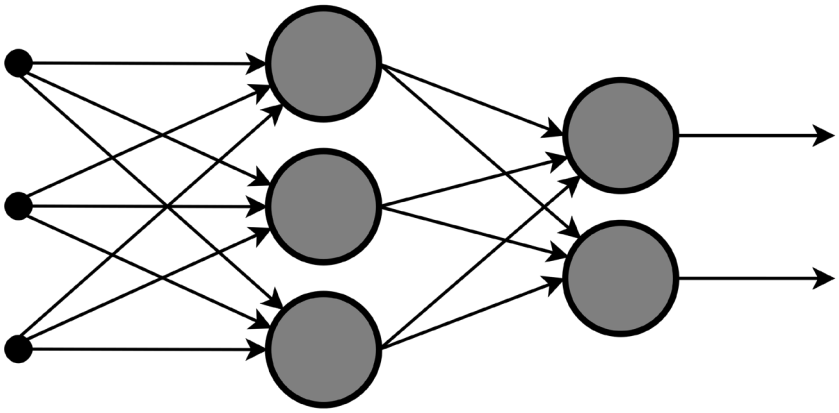
An ANN can have multiple layers: an input layer, one or more hidden layers, and an output layer. Each layer consists of multiple neurons.



Multi-Layer Perceptron

An MLP is a specific type of ANN that is fully connected, meaning each neuron in one layer is connected to every neuron in the subsequent layer.





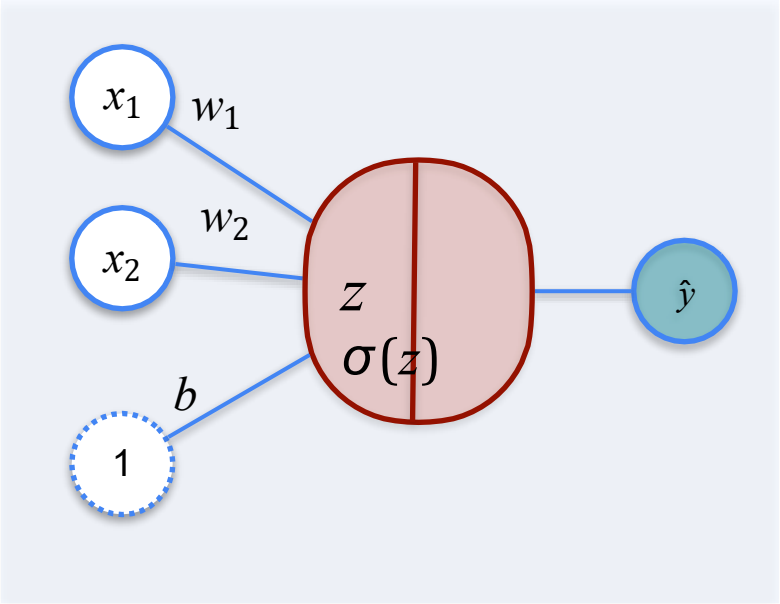
Classification with a Neuron

Gradient descent

Goal of Gradient Descent

- The prime objective is to minimize the cost function.
- By doing so, we are optimizing our model to make the most accurate predictions possible.
- The 'gradient' in Gradient Descent signifies the partial derivatives of the cost function with respect to the model parameters.
- We continually adjust the parameters in the direction of the steepest decrease in the cost function until convergence.

Classification With a Neuron

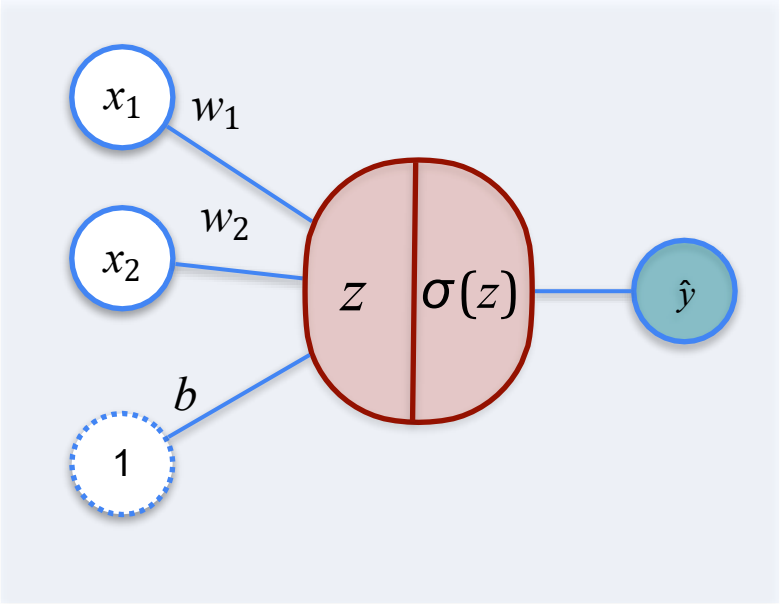


1

$L(y, \hat{y})$

0

Classification With a Neuron



1

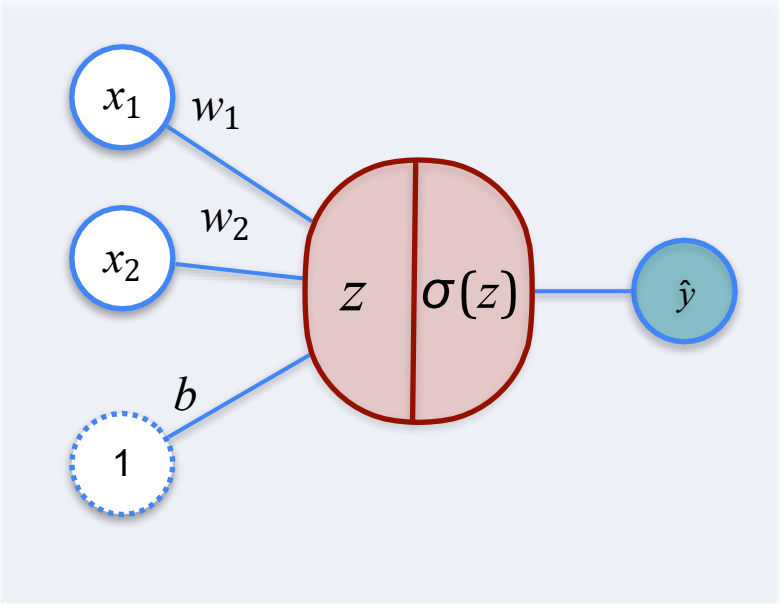
$L(y, \hat{y})$

0

Prediction Function:

$$\hat{y} = \sigma(w_1 x_1 + w_2 x_2 + b)$$

Classification With a Neuron



1

$L(y, \hat{y})$

0

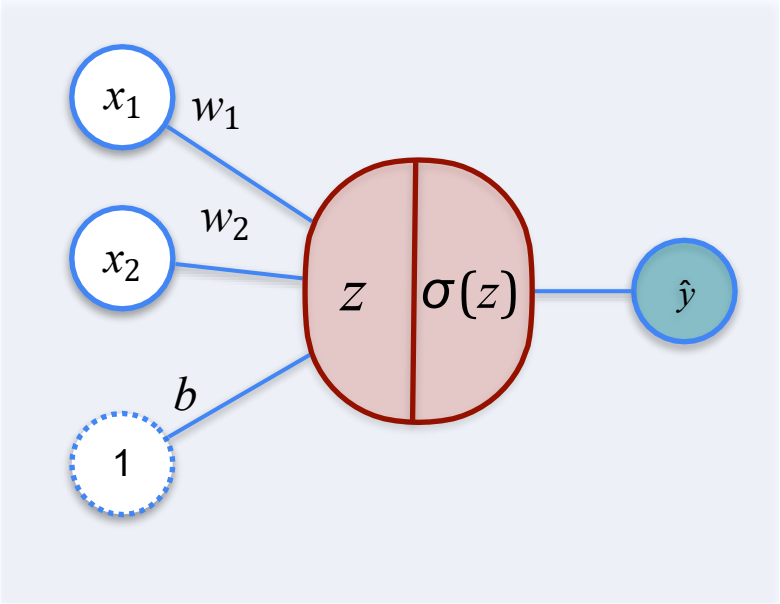
Prediction Function:

$$\hat{y} = \sigma(w_1 x_1 + w_2 x_2 + b)$$

Loss Function:

$$L(y, \hat{y}) = y \ln(\hat{y}) - (1 - y)\ln(1 - \hat{y})$$

Classification With a Neuron



1

$L(y, \hat{y})$

0

Prediction Function:

$$\hat{y} = \sigma(w_1 x_1 + w_2 x_2 + b)$$

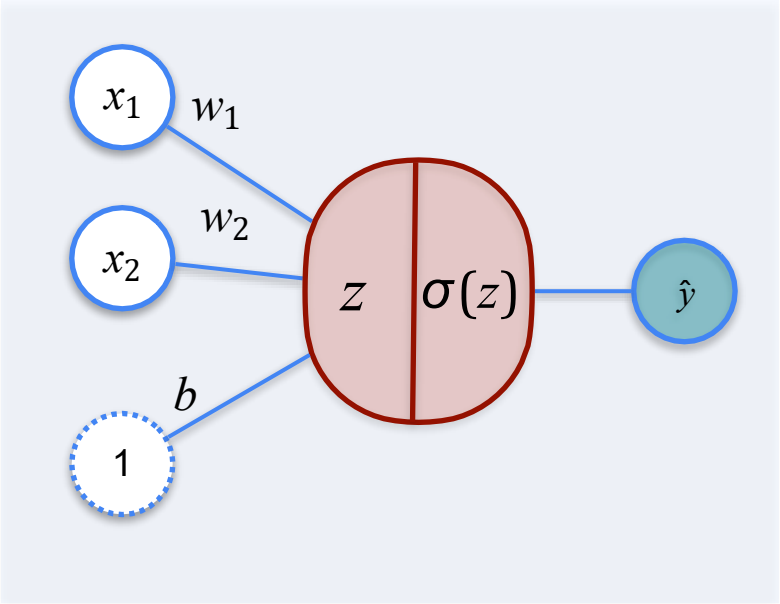
Loss Function:

$$L(y, \hat{y}) = y \ln(\hat{y}) - (1 - y)\ln(1 - \hat{y})$$

Main Goal:

Find w_1, w_2, b that yields $\hat{y}$ with the least error

Classification With a Neuron



1

$L(y, \hat{y})$

0

To find optimal values for:

w_1, w_2, b

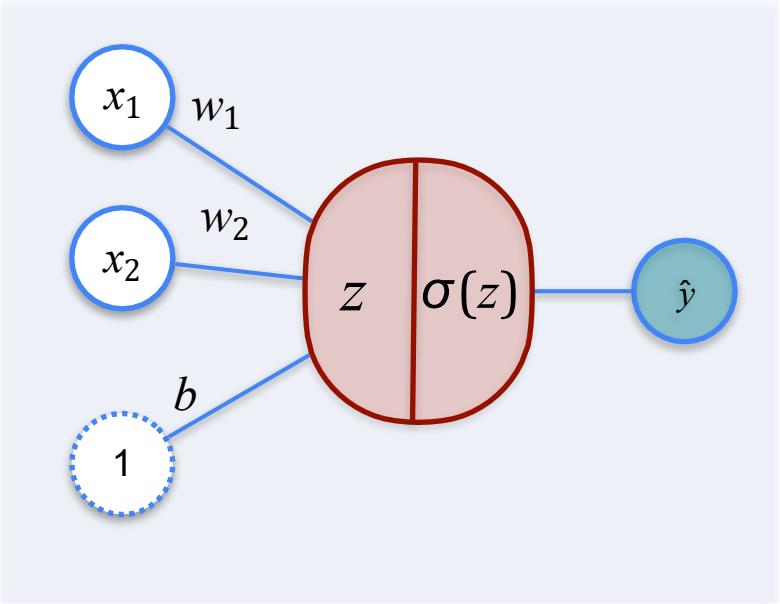
You need gradient descent

$$w_1 \rightarrow w_1 - a \frac{\partial L}{\partial w_1}$$

$$w_2 \rightarrow w_2 - a \frac{\partial L}{\partial w_2}$$

$$b \rightarrow b - a \frac{\partial L}{\partial b}$$

Classification With a Neuron



1

$L(y, \hat{y})$

0

To find optimal values for: w_1, w_2, b

You need gradient descent

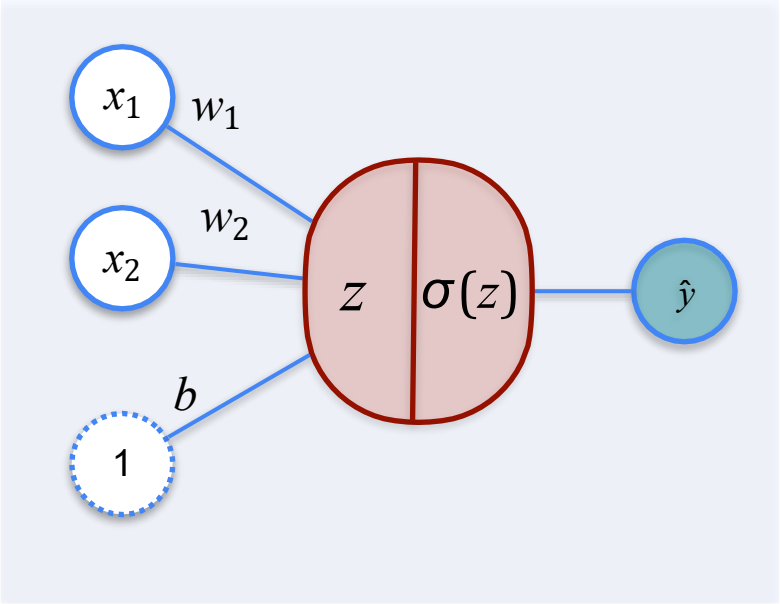
$$w_1 \rightarrow w_1 - \alpha \frac{\partial L}{\partial w_1}$$

$$w_2 \rightarrow w_2 - \alpha \frac{\partial L}{\partial w_2}$$

$$b \rightarrow b - \alpha \frac{\partial L}{\partial b}$$

Some initial starting values

Classification With a Neuron



1

$L(y, \hat{y})$

0

To find optimal values for: w_1, w_2, b

You need gradient descent

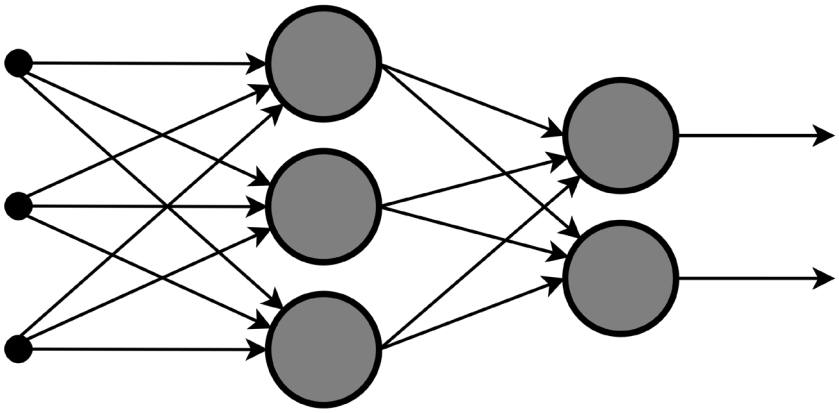
$$w_1 \rightarrow w_1 - \alpha \frac{\partial L}{\partial w_1}$$

$$w \rightarrow w_2 - \alpha \frac{\partial L}{\partial w_2}$$

$$b \rightarrow b - \alpha \frac{\partial L}{\partial b}$$

SUB-TASK

Find the following partial derivatives



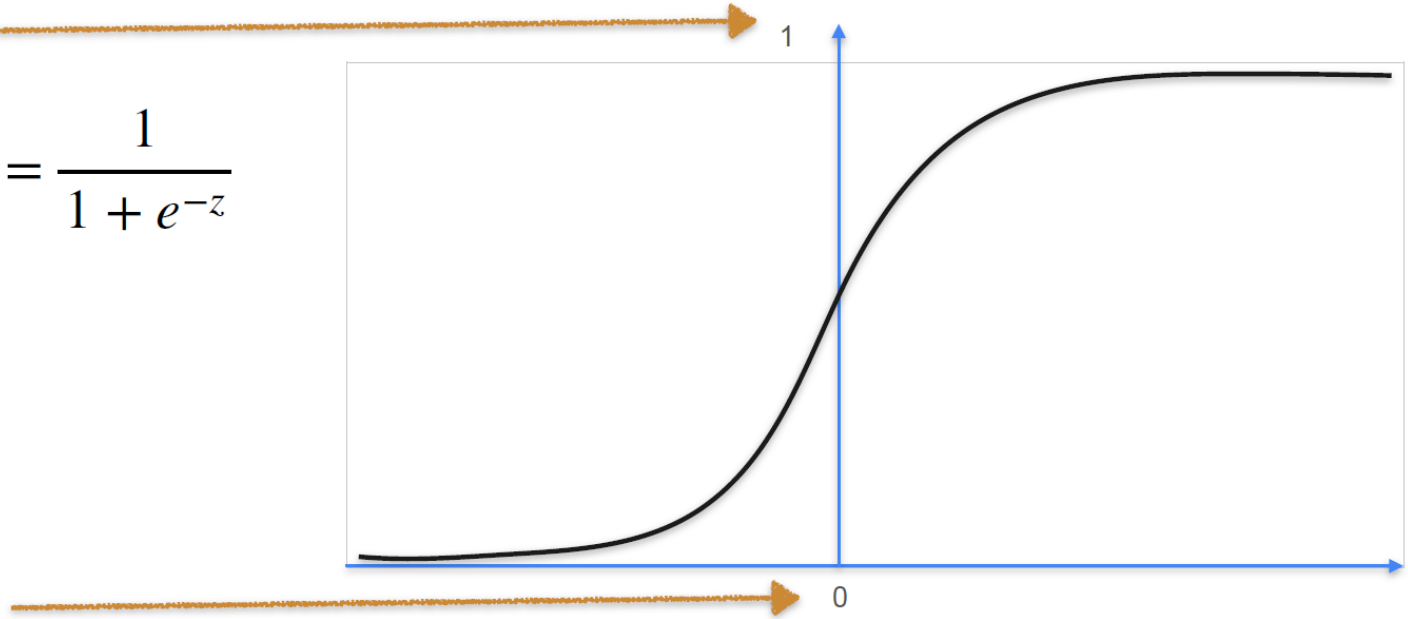
Classification with a Neuron

Derivative of a Sigmoid Function

Sigmoid Function

Sigmoid Function

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}$$



Derivative of a Sigmoid Function

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Derivative of a Sigmoid Function

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

$$\sigma(z) = (1 + e^{-z})^{-1}$$

Derivative of a Sigmoid Function

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

$$\sigma(z) = (1+e^{-z})^{-1}$$

$$\frac{d}{dz}\sigma(z) = \frac{d}{dz}(1+e^{-z})^{-1}$$

Derivative of a Sigmoid Function

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

$$\sigma(z) = (1+e^{-z})^{-1}$$

$$\frac{d}{dz}\sigma(z) = \frac{d}{dz}(1+e^{-z})^{-1}$$

Derivative of a Sigmoid Function

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

$$\sigma(z) = (1+e^{-z})^{-1}$$

$$\frac{d}{dz}\sigma(z) = \frac{d}{dz}(1+e^{-z})^{-1}$$

$$\frac{d}{dz}\sigma(z)$$

Derivative of a Sigmoid Function

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

$$\sigma(z) = (1+e^{-z})^{-1}$$

$$\frac{d}{dz}\sigma(z) = \frac{d}{dz}(1+e^{-z})^{-1}$$

$$\frac{d}{dz}\sigma(z) = -1$$

Derivative of a Sigmoid Function

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

$$\sigma(z) = (1+e^{-z})^{-1}$$

$$\frac{d}{dz}\sigma(z) = \frac{d}{dz}(1+e^{-z})^{-1}$$

$$\frac{d}{dz}\sigma(z) = -1\,(1+e^{-z})^{-1-1}$$

Derivative of a Sigmoid Function

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

$$\sigma(z) = (1+e^{-z})^{-1}$$

$$\frac{d}{dz}\sigma(z) = \frac{d}{dz}(1+e^{-z})^{-1}$$

$$\frac{d}{dz}\sigma(z) = -1\,(1+e^{-z})^{-1-1}\,(\frac{d}{dz}(1+e^{-z}))$$

Derivative of a Sigmoid Function

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

$$\sigma(z) = (1+e^{-z})^{-1}$$

$$\frac{d}{dz}\sigma(z) = \frac{d}{dz}(1+e^{-z})^{-1}$$

$$\frac{d}{dz}\sigma(z) = -1\,(1+e^{-z})^{-1-1}\,(\frac{d}{dz}(1+e^{-z}))$$

$$= -1\,(1+e^{-z})^{-2}\,(\frac{d}{dz}(1) + \frac{d}{dz}(e^{-z}))$$

Derivative of a Sigmoid Function

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

$$\sigma(z) = (1+e^{-z})^{-1}$$

$$\frac{d}{dz}\sigma(z) = \frac{d}{dz}(1+e^{-z})^{-1}$$

$$\frac{d}{dz}\sigma(z) = -1\,(1+e^{-z})^{-1-1}\,(\frac{d}{dz}(1+e^{-z}))$$

$$= -1\,(1+e^{-z})^{-2}\,(\frac{d}{dz}(1) + \frac{d}{dz}(e^{-z}))$$

$$= -1\,(1+e^{-z})^{-2}\,(0 + e^{-z}(\frac{d}{dz}(-z)))$$

Derivative of a Sigmoid Function

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

$$\sigma(z) = (1+e^{-z})^{-1}$$

$$\frac{d}{dz}\sigma(z) = \frac{d}{dz}(1+e^{-z})^{-1}$$

$$\frac{d}{dz}\sigma(z) = -1\,(1+e^{-z})^{-1-1}\,(\frac{d}{dz}(1+e^{-z}))$$

$$= -1\,(1+e^{-z})^{-2}\,(\frac{d}{dz}(1) + \frac{d}{dz}(e^{-z}))$$

$$= -1\,(1+e^{-z})^{-2}\,(0 + e^{-z}(\frac{d}{dz}(-z)))$$

$$= -1\,(1+e^{-z})^{-2}\,(e^{-z})\,(-1)$$

Derivative of a Sigmoid Function

$$\frac{d}{dz}\sigma(z) = 1\ (1+e^{-z})^{-2}\ (e^{-z})\ (-1)$$

Derivative of a Sigmoid Function

$$\frac{d}{dz}\sigma(z) = -1\ (1+e^{-z})^{-2}\ \ (e^{-z})\ (-1)$$

$$= (1+e^{-z})^{-2}\ \ (e^{-z})$$

Derivative of a Sigmoid Function

$$\frac{d}{dz}\sigma(z) = -1\ (1+e^{-z})^{-2}\ \ (e^{-z})\ (-1)$$

$$= (1+e^{-z})^{-2}\ \ (e^{-z})$$

$$= \frac{1}{(1+e^{-z})^2}\ \ (e^{-z})$$

Derivative of a Sigmoid Function

$$\frac{d}{dz}\sigma(z) = -1\ (1+e^{-z})^{-2}\ \ (e^{-z})\ (-1)$$

$$= (1+e^{-z})^{-2}\ \ (e^{-z})$$

$$= \frac{1}{(1+e^{-z})^2}\ \ (e^{-z})$$

$$= \frac{e^{-z}}{(1+e^{-z})^2}$$

Derivative of a Sigmoid Function

$$\frac{d}{dz}\sigma(z) = \frac{e^{-z}}{(1+e^{-z})^2}$$

Derivative of a Sigmoid Function

$$\frac{d}{dz}\sigma(z) = \frac{e^{-z}}{(1+e^{-z})^2}$$

$$= \frac{1+e^{-z}-1}{(1+e^{-z})^2}$$

Derivative of a Sigmoid Function

$$\frac{d}{dz}\sigma(z) = \frac{e^{-z}}{(1+e^{-z})^2}$$

$$= \frac{1+e^{-z}-1}{(1+e^{-z})^2}$$

$$= \frac{1+e^{-z}}{(1+e^{-z})^2} - \frac{1}{(1+e^{-z})^2}$$

Derivative of a Sigmoid Function

$$\frac{d}{dz}\sigma(z) = \frac{e^{-z}}{(1+e^{-z})^2}$$

$$= \frac{1+e^{-z}-1}{(1+e^{-z})^2}$$

$$= \frac{1+e^{-z}}{(1+e^{-z})^2} - \frac{1}{(1+e^{-z})^2}$$

$$= \frac{1}{(1+e^{-z})} - \frac{1}{(1+e^{-z})^2}$$

Derivative of a Sigmoid Function

$$\frac{d}{dz}\sigma(z) = \frac{e^{-z}}{(1+e^{-z})^2}$$

$$= \frac{1+e^{-z}-1}{(1+e^{-z})^2}$$

$$= \frac{1+e^{-z}}{(1+e^{-z})^2} - \frac{1}{(1+e^{-z})^2}$$

$$= \frac{1}{(1+e^{-z})} - \frac{1}{(1+e^{-z})^2}$$

$$\frac{d}{dz}\sigma(z) = \frac{1}{(1+e^{-z})} - \left(\frac{1}{(1+e^{-z})}\right)\left(\frac{1}{(1+e^{-z})}\right)$$

Derivative of a Sigmoid Function

$$\frac{d}{dz}\sigma(z) = \frac{e^{-z}}{(1+e^{-z})^2}$$

$$= \frac{1+e^{-z}-1}{(1+e^{-z})^2}$$

$$= \frac{1+e^{-z}}{(1+e^{-z})^2} - \frac{1}{(1+e^{-z})^2}$$

$$= \frac{1}{(1+e^{-z})} - \frac{1}{(1+e^{-z})^2}$$

$$\frac{d}{dz}\sigma(z) = \frac{1}{(1+e^{-z})} - \left(\frac{1}{(1+e^{-z})}\right)\left(\frac{1}{(1+e^{-z})}\right)$$

$$= \frac{1}{(1+e^{-z})}\left(1 - \frac{1}{(1+e^{-z})}\right)$$

Derivative of a Sigmoid Function

$$\frac{d}{dz}\sigma(z) = \frac{e^{-z}}{(1+e^{-z})^2}$$

$$= \frac{1+e^{-z}-1}{(1+e^{-z})^2}$$

$$= \frac{1+e^{-z}}{(1+e^{-z})^2} - \frac{1}{(1+e^{-z})^2}$$

$$= \frac{1}{(1+e^{-z})} - \frac{1}{(1+e^{-z})^2}$$

$$\frac{d}{dz}\sigma(z) = \frac{1}{(1+e^{-z})} - \left(\frac{1}{(1+e^{-z})}\right)\left(\frac{1}{(1+e^{-z})}\right)$$

$$= \frac{1}{(1+e^{-z})}\left(1 - \frac{1}{(1+e^{-z})}\right)$$

Recall that: $\sigma(z) = \frac{1}{1+e^{-z}}$

Derivative of a Sigmoid Function

$$\frac{d}{dz}\sigma(z) = \frac{e^{-z}}{(1+e^{-z})^2}$$

$$= \frac{1+e^{-z}-1}{(1+e^{-z})^2}$$

$$= \frac{1+e^{-z}}{(1+e^{-z})^2} - \frac{1}{(1+e^{-z})^2}$$

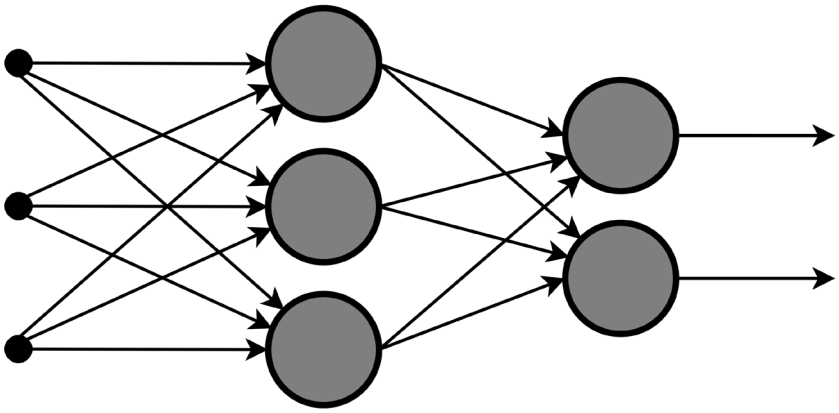
$$= \frac{1}{(1+e^{-z})} - \frac{1}{(1+e^{-z})^2}$$

$$\frac{d}{dz}\sigma(z) = \frac{1}{(1+e^{-z})} - \left(\frac{1}{(1+e^{-z})}\right)\left(\frac{1}{(1+e^{-z})}\right)$$

$$= \frac{1}{(1+e^{-z})}\left(1 - \frac{1}{(1+e^{-z})}\right)$$

Recall that: $\sigma(z) = \frac{1}{1+e^{-z}}$

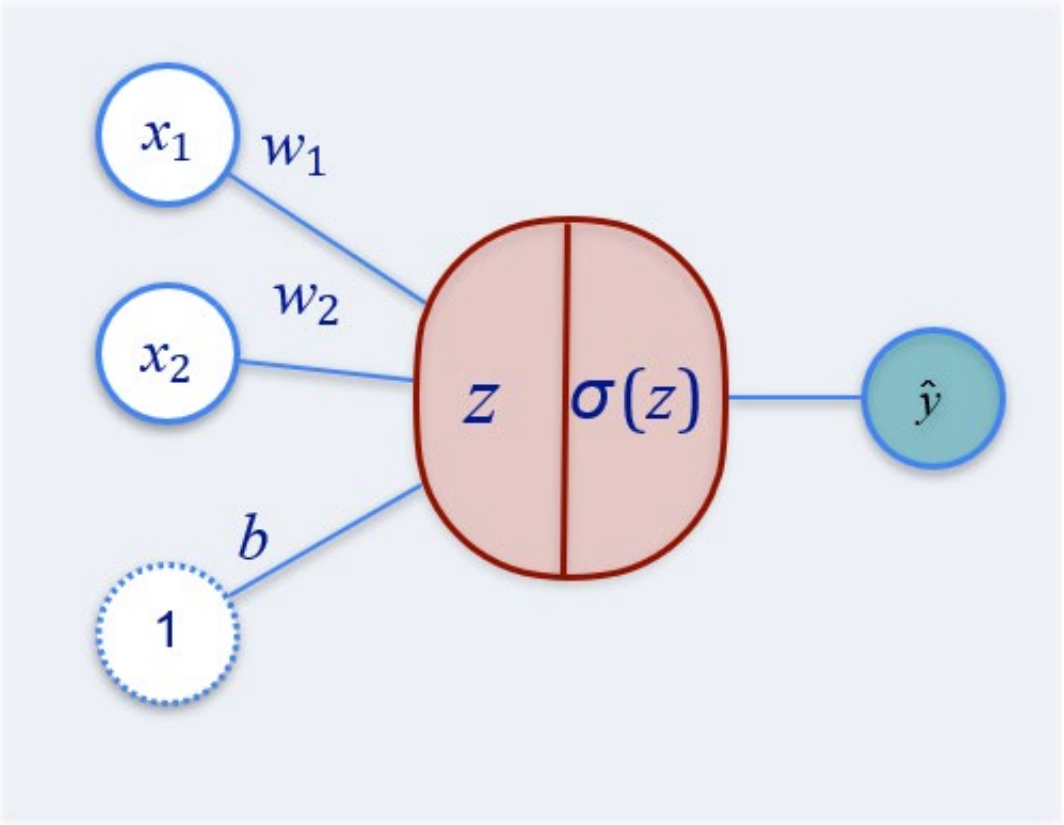
$$\frac{d}{dz}\sigma(z) = \sigma(z)\,(1-\sigma(z))$$



Classification with a Neuron

Calculating the derivatives

Classification With a Neuron



$L(y, \hat{y})$

Calculating the Derivatives

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial b}$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_1}$$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_2}$$

Calculating the Derivatives

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial b}$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_1}$$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_2}$$

$\frac{\partial L}{\partial \hat{y}} =$

$\frac{\partial \hat{y}}{\partial b} =$

$\frac{\partial \hat{y}}{\partial w_1} =$

$\frac{\partial \hat{y}}{\partial w_2} =$

?

Calculating the Derivatives

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial b}$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_1}$$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_2}$$

$\frac{\partial L}{\partial \hat{y}} =$

$\frac{\partial \hat{y}}{\partial b} =$

$\frac{\partial \hat{y}}{\partial w_1} =$

$\frac{\partial \hat{y}}{\partial w_2} =$

?

$$\hat{y} = \sigma(w_1 x_1 + w_2 x_2 + b)$$

$$L(y, \hat{y}) = -y \ln(\hat{y}) - (1 - y)\ln(1 - \hat{y})$$

Calculating the Derivatives

$$\frac{\partial L}{\partial \hat{y}}$$

$$L(y, \hat{y}) = -y \ln(\hat{y}) - (1 - y)\ln 1 - \hat{y})$$

Calculating the Derivatives

$$\frac{\partial L}{\partial \hat{y}}$$

$$L(y, \hat{y}) = -y \ln(\hat{y}) - (1 - y)\ln 1 - \hat{y})$$

$$\frac{\partial L}{\partial \hat{y}} = \frac{-y}{\hat{y}} + \frac{1 - y}{1 - \hat{y}}$$

Calculating the Derivatives

$$\frac{\partial L}{\partial \hat{y}}$$

$$L(y, \hat{y}) = -y \ln(\hat{y}) - (1 - y)\ln 1 - \hat{y})$$

$$\frac{\partial L}{\partial \hat{y}} = \frac{-y}{\hat{y}} + \frac{1 - y}{1 - \hat{y}}$$

$$= \frac{-y + y\hat{y} + \hat{y} - y\hat{y}}{\hat{y}(1 - \hat{y})}$$

Calculating the Derivatives

$$\frac{\partial L}{\partial \hat{y}}$$

$$L(y, \hat{y}) = -y \ln(\hat{y}) - (1 - y)\ln 1 - \hat{y})$$

$$\frac{\partial L}{\partial \hat{y}} = \frac{-y}{\hat{y}} + \frac{1 - y}{1 - \hat{y}}$$

$$= \frac{-y + y\hat{y} + \hat{y} - y\hat{y}}{\hat{y}(1 - \hat{y})}$$

Calculating the Derivatives

$$\frac{\partial L}{\partial \hat{y}} = \frac{-(y - \hat{y})}{\hat{y}(1 - \hat{y})}$$

$$L(y, \hat{y}) = -y \ln(\hat{y}) - (1 - y)\ln 1 - \hat{y})$$

$$\frac{\partial L}{\partial \hat{y}} = \frac{-y}{\hat{y}} + \frac{1 - y}{1 - \hat{y}}$$

$$= \frac{-y + y\hat{y} + \hat{y} - y\hat{y}}{\hat{y}(1 - \hat{y})}$$

$$= \frac{-(y - \hat{y})}{\hat{y}(1 - \hat{y})}$$

Calculating the Derivatives

$$\hat{y} = \sigma(w_1 x_1 + w_2 x_2 + b)$$

Calculating the Derivatives

$$\hat{y} = \sigma(w_1 x_1 + w_2 x_2 + b)$$

$\frac{\partial \hat{y}}{\partial w_1}$	$= \hat{y}(1 - \hat{y}) x_1$

Calculating the Derivatives

$$\hat{y} = \sigma(w_1 x_1 + w_2 x_2 + b)$$

$\frac{\partial \hat{y}}{\partial w_1}$	$= \hat{y}(1 - \hat{y})x_1$
$\frac{\partial \hat{y}}{\partial w_2}$	$= \hat{y}(1 - \hat{y})x_2$
$\frac{\partial \hat{y}}{\partial b}$	$= \hat{y}(1 - \hat{y})$

Calculating the Derivatives

$$\frac{\partial L}{\partial \hat{y}} = \frac{-(y - \hat{y})}{\hat{y}(1-\hat{y})}$$

$$\frac{\partial \hat{y}}{\partial b} = \hat{y}(1-\hat{y})$$

$$\frac{\partial \hat{y}}{\partial w_1} = \hat{y}(1-\hat{y})x_1$$

$$\frac{\partial \hat{y}}{\partial w_2} = \hat{y}(1-\hat{y})x_2$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial b} = \frac{-(y - \hat{y})}{\hat{y}(1-\hat{y})}\,\hat{y}(1-\hat{y})$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_1} = \frac{-(y - \hat{y})}{\hat{y}(1-\hat{y})}\,\hat{y}(1-\hat{y})x_1$$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_2} = \frac{-(y - \hat{y})}{\hat{y}(1-\hat{y})}\,\hat{y}(1-\hat{y})x_2$$

Calculating the Derivatives

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial b} = \frac{-(y - \hat{y})}{\hat{y}(1 - \hat{y})} \hat{y}(1 - \hat{y})$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_1} = \frac{-(y - \hat{y})}{\hat{y}(1 - \hat{y})} \hat{y}(1 - \hat{y}) x_1$$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_2} = \frac{-(y - \hat{y})}{\hat{y}(1 - \hat{y})} \hat{y}(1 - \hat{y}) x_2$$

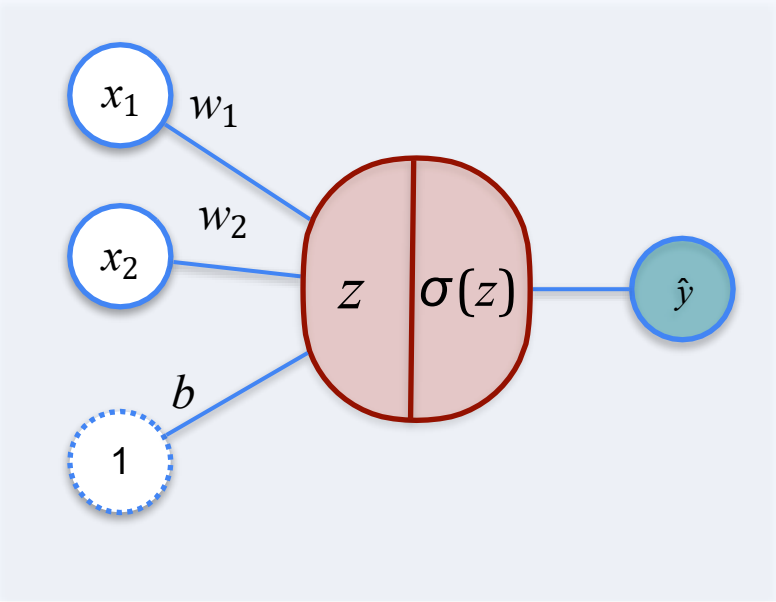
Calculating the Derivatives

$$\frac{\partial L}{\partial b} = -(y - \hat{y})$$

$$\frac{\partial L}{\partial w_1} = -(y - \hat{y})x_1$$

$$\frac{\partial L}{\partial w_2} = -(y - \hat{y})x_2$$

Classification With a Neuron



1

$L(y, \hat{y})$

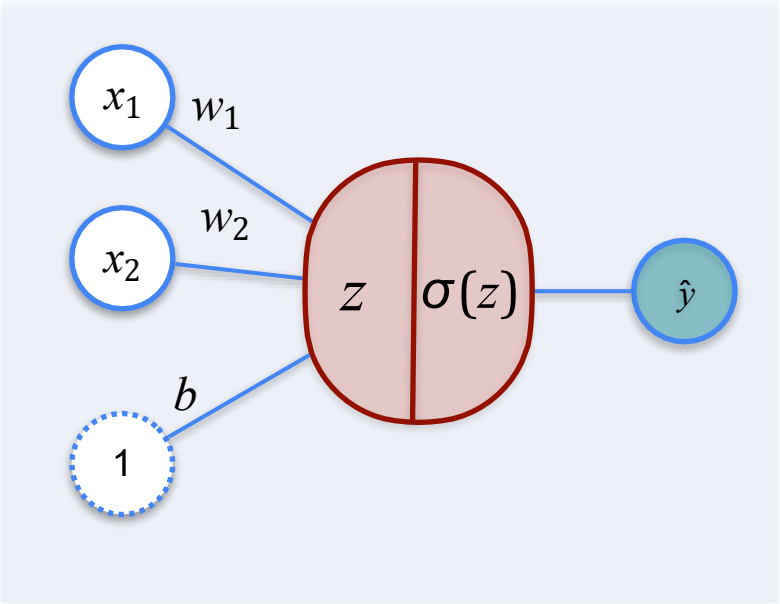
0

To find optimal values for:
w_1, w_2, b

You need gradient descent

$$w_1 \rightarrow w_1 - \alpha \frac{\partial L}{\partial w_1}$$

Classification With a Neuron



1

$L(y, \hat{y})$

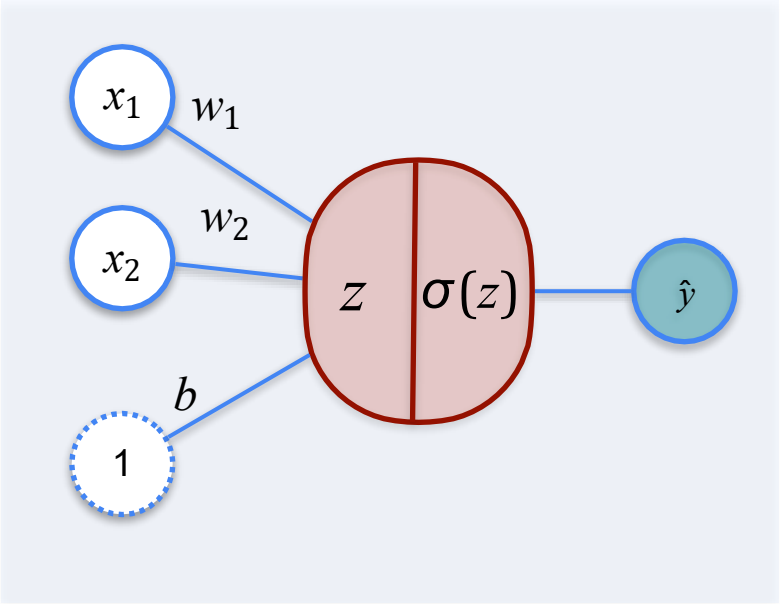
0

To find optimal values for:
w_1 , w_2 , b

You need gradient descent

$$w_1 \rightarrow w_1 - \alpha(-x_1(y - \hat{y}))$$

Classification With a Neuron



1

$L(y, \hat{y})$

0

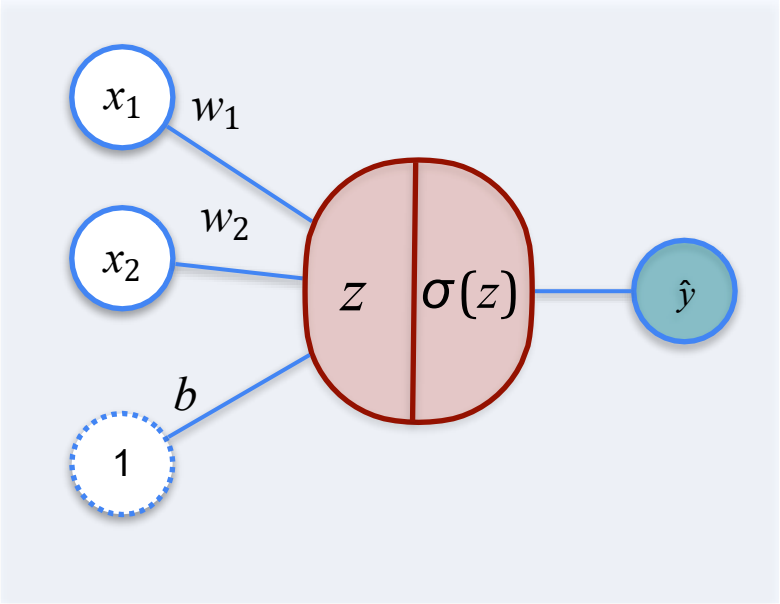
To find optimal values for:
w_1, w_2, b

You need gradient descent

$$w_1 \rightarrow w_1 - \alpha(-x_1(y - \hat{y}))$$

$$w_2 \rightarrow w_2 - \alpha \frac{\partial L}{\partial w_2}$$

Classification With a Neuron



1

$L(y, \hat{y})$

0

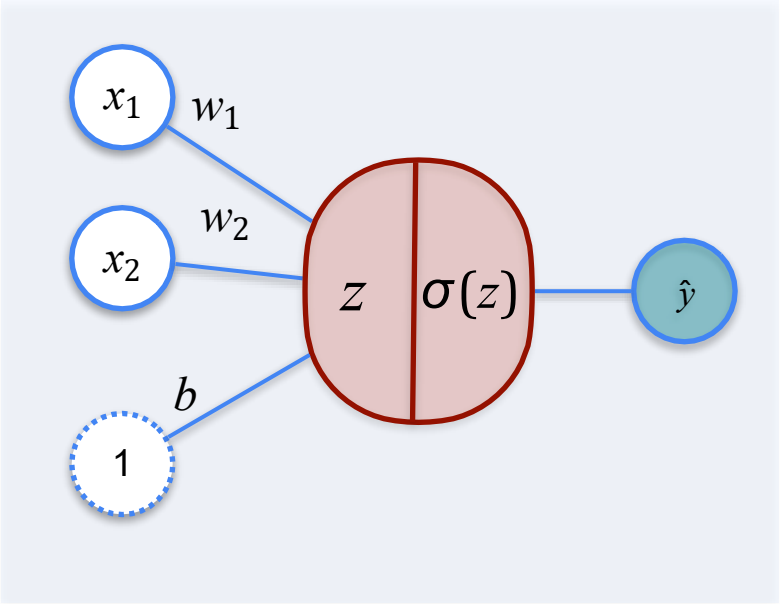
To find optimal values for:
w_1, w_2, b

You need gradient descent

$$w_1 \rightarrow w_1 - \alpha(-x_1(y - \hat{y}))$$

$$w_2 \rightarrow w_2 - \alpha(-x_2(y - \hat{y}))$$

Classification With a Neuron



1

$L(y, \hat{y})$

0

To find optimal values for:
w_1, w_2, b

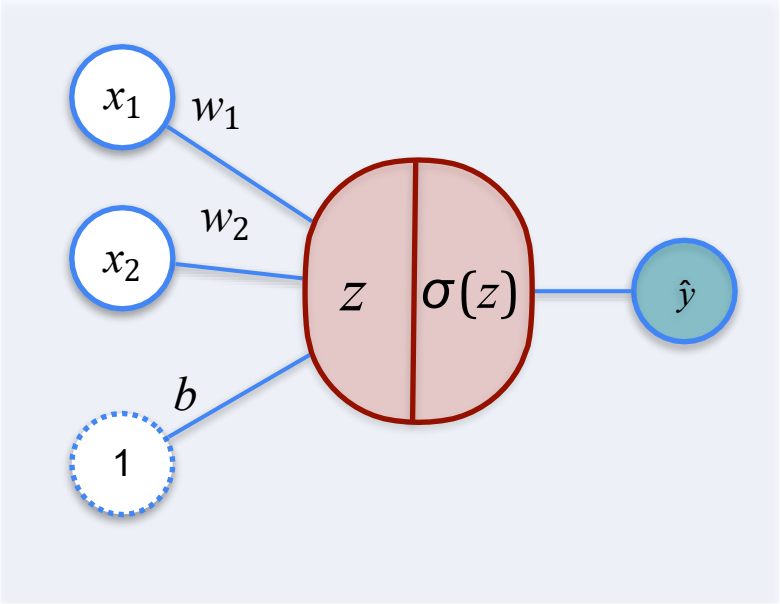
You need gradient descent

$$w_1 \rightarrow w_1 - \alpha(-x_1(y - \hat{y}))$$

$$w_2 \rightarrow w_2 - \alpha(-x_2(y - \hat{y}))$$

$$b \rightarrow b - \alpha \frac{\partial L}{\partial b}$$

Classification With a Neuron



1

$L(y, \hat{y})$

0

To find optimal values for:
w_1, w_2, b

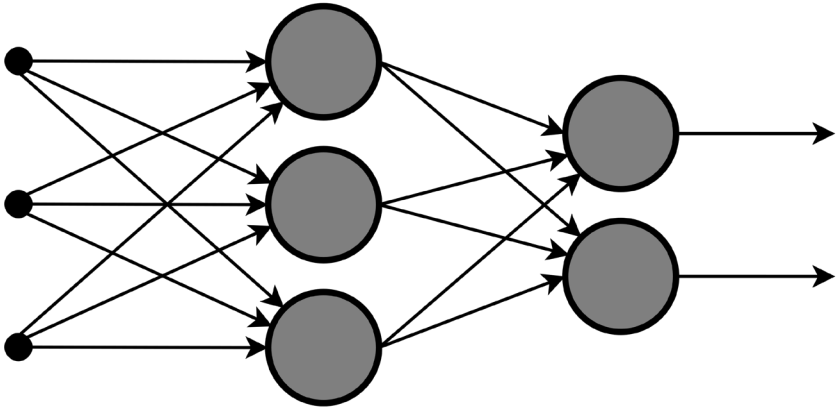
You need gradient descent

$$w_1 \rightarrow w_1 - \alpha(-x_1(y - \hat{y}))$$

$$w_2 \rightarrow w_2 - \alpha(-x_2(y - \hat{y}))$$

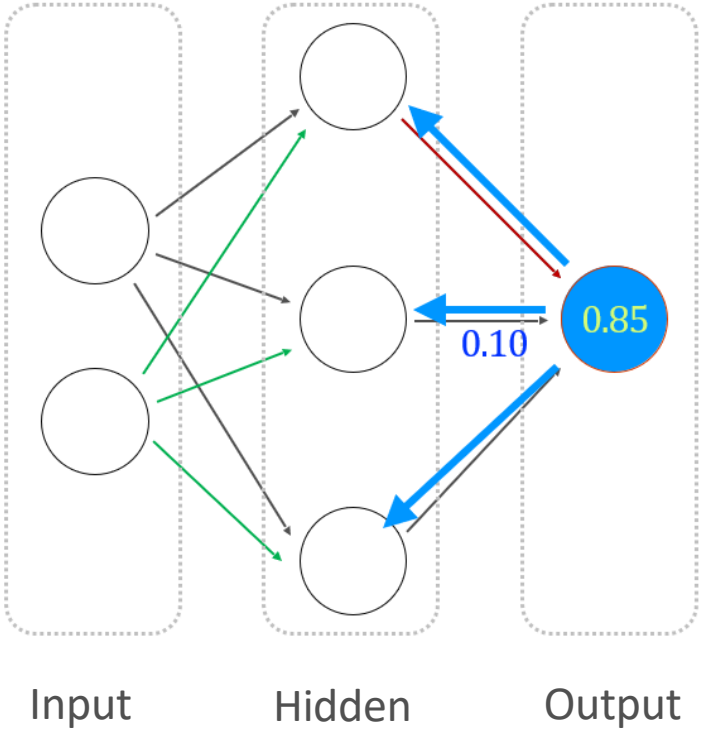
$$b \rightarrow b - \alpha(-(y - \hat{y}))$$

Neural Networks



Classification with a Neural Network

Classification With a Neuron - Backpropagation



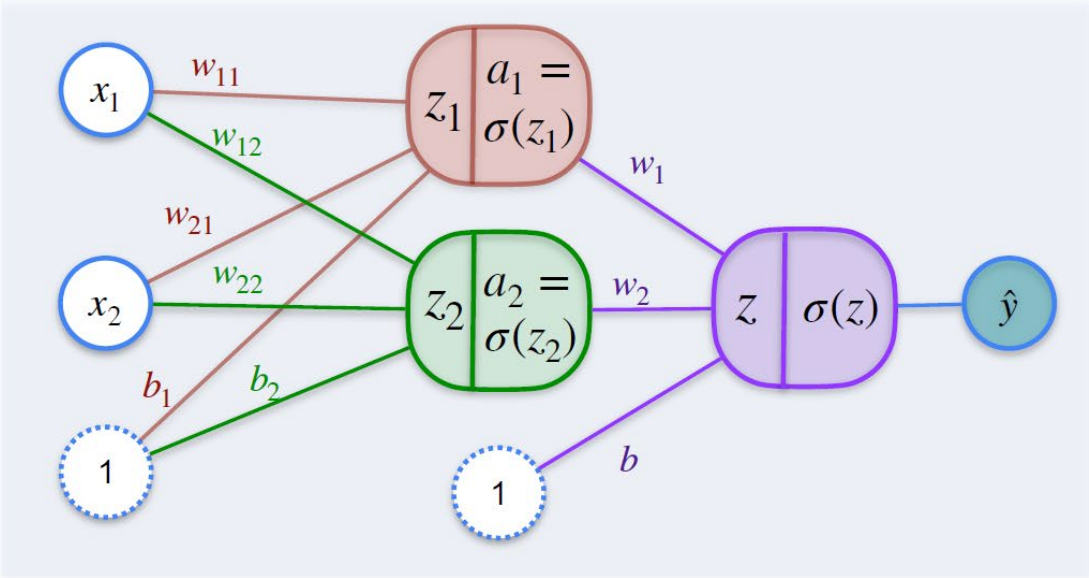
Source; databricks - Deep Learning Fundementals

- Backpropagation: Calculate the gradient of the cost function with respect to each model parameter by the chain rule, propagating the error backward through the network.
- Used by gradient descent algorithm to adjust weight of neurons
- Also known as backward propagation of errors *as the error is calculated and distributed back through the network of layers*

Neural Network

Neural network of depth 2

- one input layer
- one hidden layer
- one output layer



Neural Network

$$a_1 = \sigma(z_1)$$

$$z_1 = x_1 w_{11} + x_2 w_{21} + b_1$$

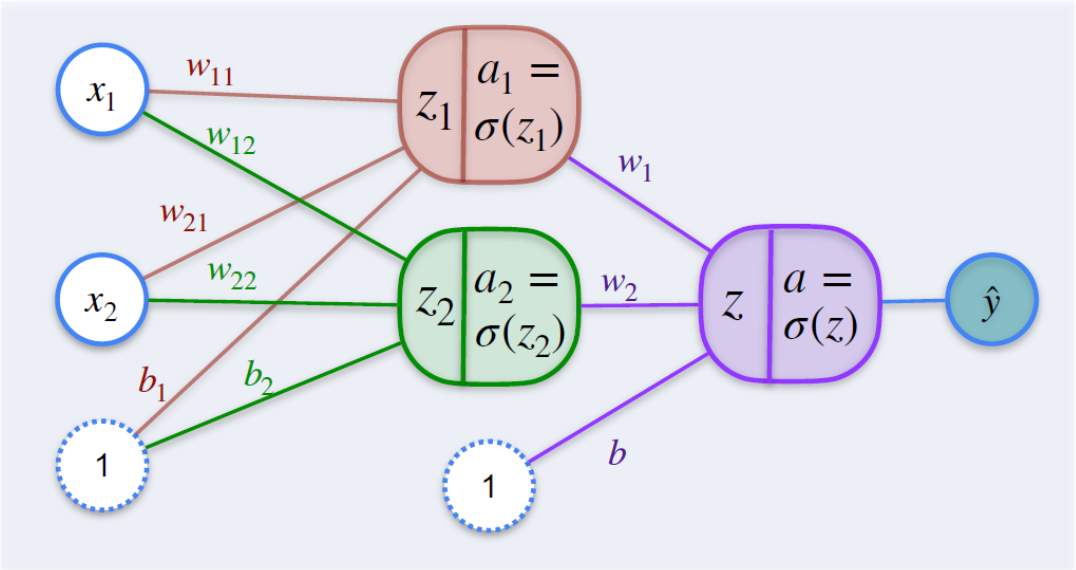
$$a_2 = \sigma(z_2)$$

$$z_2 = x_1 w_{12} + x_2 w_{22} + b_2$$

$$\hat{y} = \sigma(z)$$

$$z = a_1 w_1 + a_2 w_2 + b$$

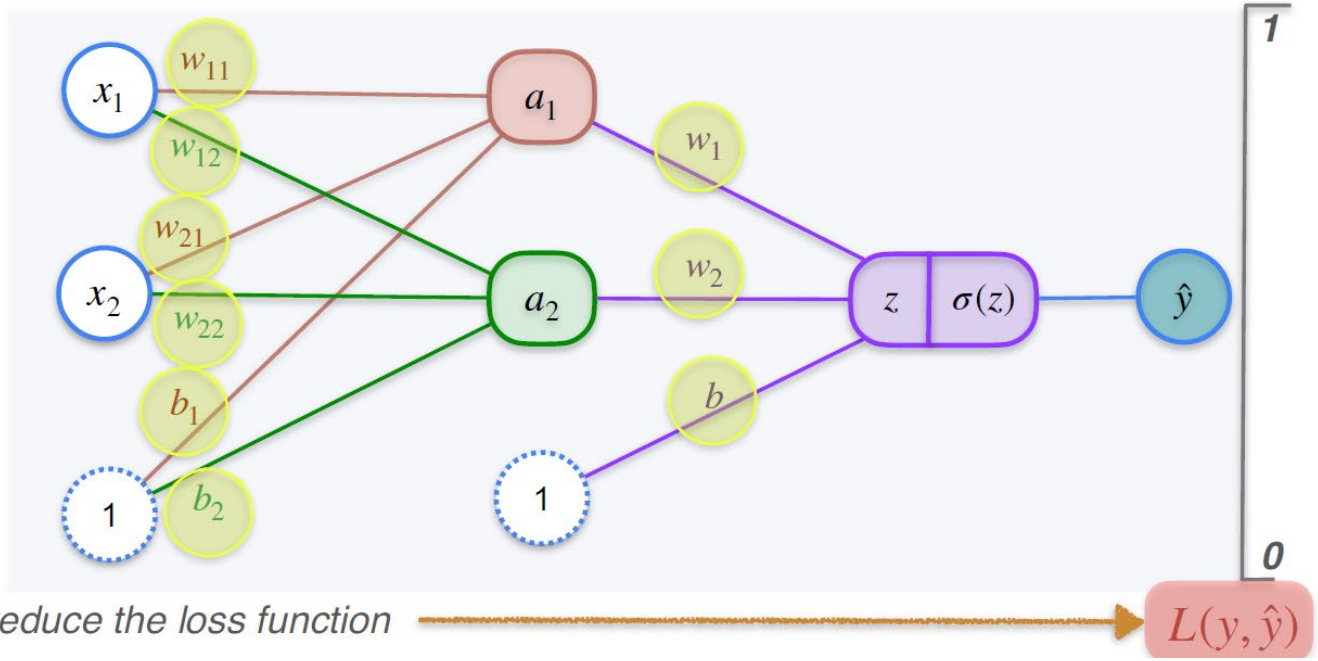
$$L(y, \hat{y}) = -y log(\hat{y}) - (1 - y) log(1 - \hat{y})$$



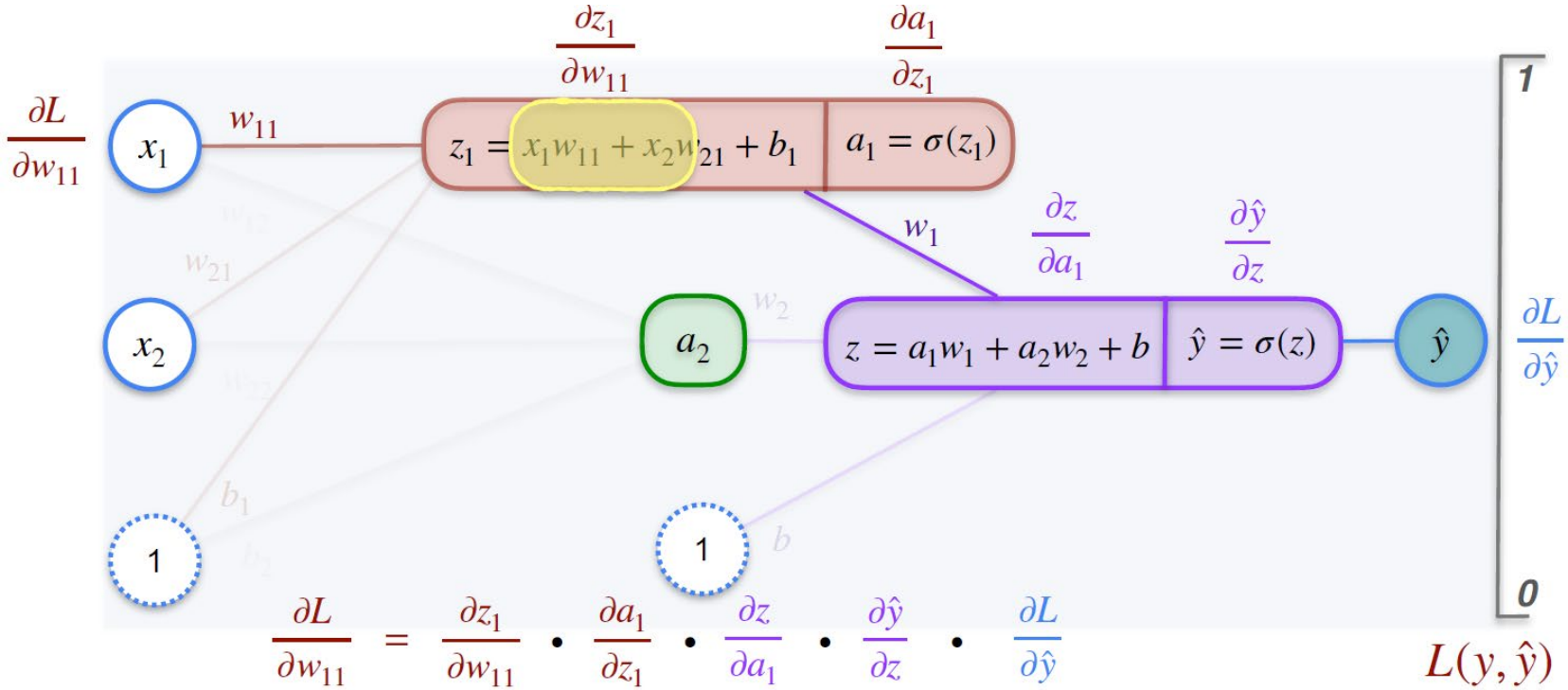
Neural Network

Goal

Adjust each of the highlighted weights and biases



Neural Network



Neural Network

$$L(y, \hat{y}) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$$

$$\hat{y} = \sigma(z)$$

$$z = a_1 w_1 + a_2 w_2 + b$$

$$a_1 = \sigma(z_1)$$

$$z_1 = x_1 w_{11} + x_2 w_{21} + b_1$$

$$\frac{\partial L}{\partial w_{11}} = \frac{\partial z_1}{\partial w_{11}} \cdot \frac{\partial a_1}{\partial z_1} \cdot \frac{\partial z}{\partial a_1} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial L}{\partial \hat{y}}$$

$$\frac{\partial L}{\partial w_{11}} = x_1 \cdot a_1(1-a_1) \cdot w_1 \cdot \cancel{\hat{y}(1-\hat{y})} \cdot \frac{-(y-\hat{y})}{\cancel{\hat{y}(1-\hat{y})}}$$

$$= -x_1 w_1 a_1 (1-a_1)(y-\hat{y})$$

Perform gradient descent with

$$w_{11} \rightarrow w_{11} - \alpha\text{-}x_1 w_1 a_1 (1-a_1)(y-\hat{y})$$

to find optimal value of w_{11} that gives the least error

Neural Network

$$L(y, \hat{y}) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$$

$$\hat{y} = \sigma(z)$$

$$z = a_1 w_1 + a_2 w_2 + b$$

$$a_1 = \sigma(z_1)$$

$$z_1 = x_1 w_{11} + x_2 w_{21} + b_1$$

$$\frac{\partial L}{\partial w_{21}} = \frac{\partial z_1}{\partial w_{21}} \cdot \frac{\partial a_1}{\partial z_1} \cdot \frac{\partial z}{\partial a_1} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial L}{\partial \hat{y}}$$

$$\frac{\partial L}{\partial w_{21}} = x_2 \cdot a_1(1-a_1) \cdot w_1 \cdot \cancel{\hat{y}(1-\hat{y})} \cdot \frac{-(y-\hat{y})}{\cancel{\hat{y}(1-\hat{y})}}$$

$$= -x_2 w_1 a_1 (1-a_1)(y-\hat{y})$$

Perform gradient descent with

$$w_{21} \rightarrow w_{21} - \alpha \frac{\partial L}{\partial w_{21}}$$

to find optimal value of w_{21} that gives the least error

Neural Network

$$L(y, \hat{y}) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$$

$$\hat{y} = \sigma(z)$$

$$z = a_1 w_1 + a_2 w_2 + b$$

$$a_1 = \sigma(z_1)$$

$$z_1 = x_1 w_{11} + x_2 w_{21} + b_1$$

$$\frac{\partial L}{\partial w_{21}} = \frac{\partial z_1}{\partial w_{21}} \cdot \frac{\partial a_1}{\partial z_1} \cdot \frac{\partial z}{\partial a_1} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial L}{\partial \hat{y}}$$

$$\frac{\partial L}{\partial w_{21}} = x_2 \cdot a_1(1-a_1) \cdot w_1 \cdot \cancel{\hat{y}(1-\hat{y})} \cdot \frac{-(y - \hat{y})}{\cancel{\hat{y}(1-\hat{y})}}$$

$$= -x_2 w_1 a_1 (1-a_1)(y - \hat{y})$$

Perform gradient descent with

$$w_{21} \rightarrow w_{21} - \alpha \text{-} x_2 w_1 a_1 (1-a_1)(y - \hat{y})$$

to find optimal value of w_{21} that gives the least error

Neural Network

$$L(y, \hat{y}) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$$

$$\hat{y} = \sigma(z)$$

$$z = a_1 w_1 + a_2 w_2 + b$$

$$a_1 = \sigma(z_1)$$

$$z_1 = x_1 w_{11} + x_2 w_{21} + b_1$$

$$\frac{\partial L}{\partial b_1} = \frac{\partial z_1}{\partial b_1} \cdot \frac{\partial a_1}{\partial z_1} \cdot \frac{\partial z}{\partial a_1} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial L}{\partial \hat{y}}$$

$$\frac{\partial L}{\partial b_1} = 1 \cdot a_1(1-a_1) \cdot w_1 \cdot \cancel{\hat{y}(1-\hat{y})} \cdot \frac{-(y-\hat{y})}{\cancel{\hat{y}(1-\hat{y})}}$$

$$= -w_1 a_1 (1-a_1)(y-\hat{y})$$

Perform gradient descent with

$$b_1 \rightarrow b_1 - \alpha \frac{\partial L}{\partial b_1}$$

to find optimal value of b_1 that gives the least error

Neural Network

$$L(y, \hat{y}) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$$

$$\hat{y} = \sigma(z)$$

$$z = a_1 w_1 + a_2 w_2 + b$$

$$a_1 = \sigma(z_1)$$

$$z_1 = x_1 w_{11} + x_2 w_{21} + b_1$$

$$\frac{\partial L}{\partial b_1} = \frac{\partial z_1}{\partial b_1} \cdot \frac{\partial a_1}{\partial z_1} \cdot \frac{\partial z}{\partial a_1} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial L}{\partial \hat{y}}$$

$$\frac{\partial L}{\partial b_1} = 1 \cdot a_1(1-a_1) \cdot w_1 \cdot \cancel{\hat{y}(1-\hat{y})} \cdot \frac{-(y - \hat{y})}{\cancel{\hat{y}(1-\hat{y})}}$$

$$= -w_1 a_1 (1-a_1)(y - \hat{y})$$

Perform gradient descent with

$$b_1 \rightarrow b_1 - \alpha(-w_1 a_1 (1-a_1)(y - \hat{y}))$$

to find optimal value of b_1 that gives the least error

Neural Network

$$L(y, \hat{y}) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$$

$$\hat{y} = \sigma(z)$$

$$z = a_1 w_1 + a_2 w_2 + b$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial z}{\partial w_1} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial L}{\partial \hat{y}}$$

$$\frac{\partial L}{\partial w_1} = a_1 \cdot \cancel{\hat{y}(1-\hat{y})} \cdot \frac{-(y - \hat{y})}{\cancel{\hat{y}(1-\hat{y})}}$$

$$= -a_1(y - \hat{y})$$

Perform gradient descent with

$$w_1 \rightarrow w_1 - \alpha \frac{\partial L}{\partial w_1}$$

to find optimal value of w_1 that gives the least error

Neural Network

$$L(y, \hat{y}) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$$

$$\hat{y} = \sigma(z)$$

$$z = a_1 w_1 + a_2 w_2 + b$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial z}{\partial w_1} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial L}{\partial \hat{y}}$$

$$\frac{\partial L}{\partial w_1} = a_1 \cdot \cancel{\hat{y}(1-\hat{y})} \cdot \frac{-(y - \hat{y})}{\cancel{\hat{y}(1-\hat{y})}}$$

$$= -a_1(y - \hat{y})$$

Perform gradient descent with

$$w_1 \rightarrow w_1 - \alpha(-a_1(y - \hat{y}))$$

to find optimal value of w_1 that gives the least error

Neural Network

$$L(y, \hat{y}) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$$

$$\hat{y} = \sigma(z)$$

$$z = a_1 w_1 + a_2 w_2 + b$$

$$\frac{\partial L}{\partial b} = \frac{\partial z}{\partial b} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial L}{\partial \hat{y}}$$

$$\frac{\partial L}{\partial b} = 1 \cdot \cancel{\hat{y}(1-\hat{y})} \cdot \frac{-(y - \hat{y})}{\cancel{\hat{y}(1-\hat{y})}}$$

$$= -(y - \hat{y})$$

Perform gradient descent with

$$b \rightarrow b - \alpha \frac{\partial L}{\partial b}$$

to find optimal value of b that gives the least error

Neural Network

$$L(y, \hat{y}) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$$

$$\hat{y} = \sigma(z)$$

$$z = a_1 w_1 + a_2 w_2 + b$$

$$\frac{\partial L}{\partial b} = \frac{\partial z}{\partial b} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial L}{\partial \hat{y}}$$

$$\frac{\partial L}{\partial b} = 1 \cdot \cancel{\hat{y}(1-\hat{y})} \cdot \frac{-(y - \hat{y})}{\cancel{\hat{y}(1-\hat{y})}}$$

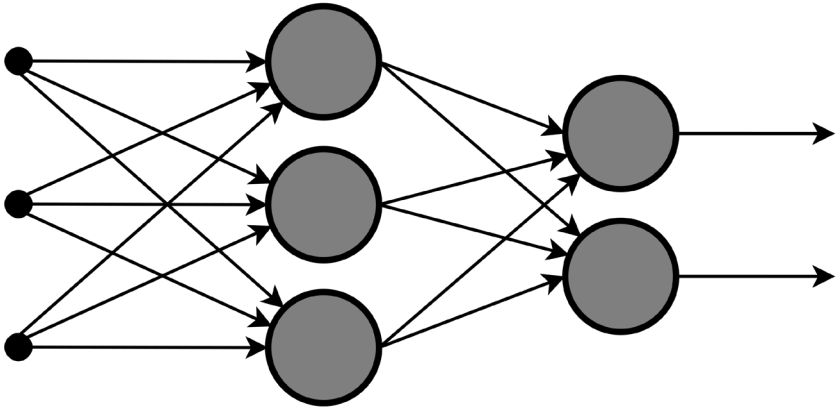
$$= -(y - \hat{y})$$

Perform gradient descent with

$$b \rightarrow b - \alpha(-(y - \hat{y}))$$

to find optimal value of b that gives the least error

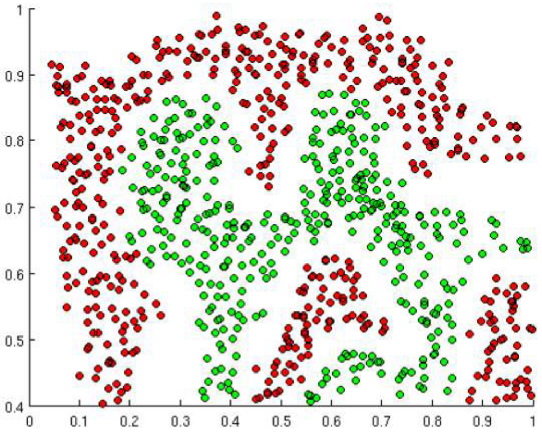
Neural Networks



Activation Functions

Importance of Activation Functions

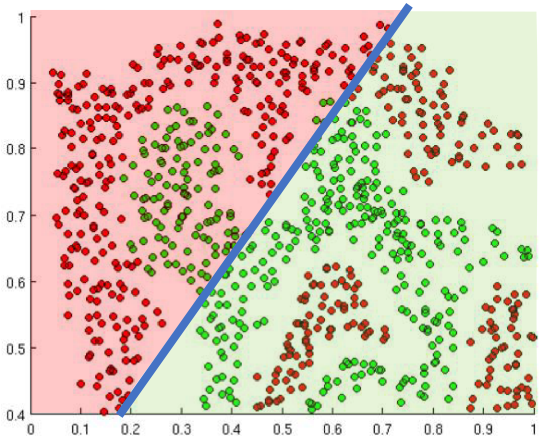
The purpose of activation functions is to introduce non-linearities into the network



What if we wanted to build a Neural Network to distinguish green from red points?

Importance of Activation Functions

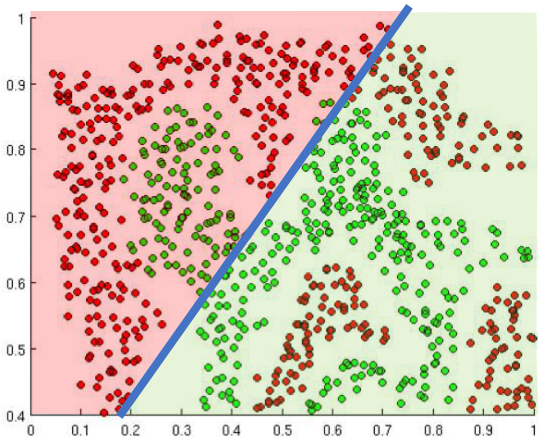
The purpose of activation functions is to introduce non-linearities into the network



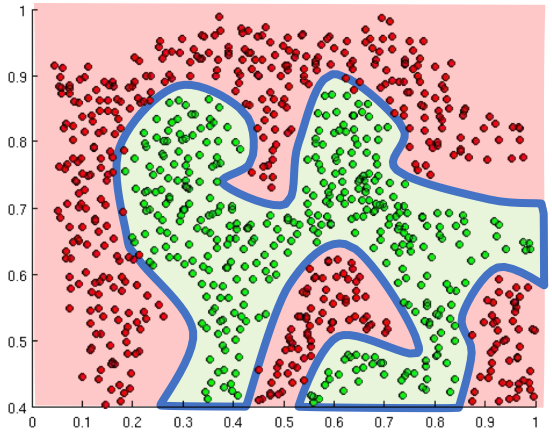
Linear Activation functions produce linear decisions no matter the network size

Importance of Activation Functions

The purpose of activation functions is to introduce non-linearities into the network



Linear Activation functions produce linear decisions no matter the network size



Non-linearities allow us to approximate arbitrarily complex functions

- Activation functions introduce non-linearity. This is critical because it allows the neural network to model highly complex and non-linear relationships between input features and output.

- Without an activation function, each neuron in the network simply performs a linear transformation of the input. Specifically, the output of a neuron without an activation function is just the weighted sum of its inputs, plus a bias term. No matter how many layers you have, if all are linear, you can always collapse them into a single layer that is also linear.

- Consider a simple network with two layers without activation functions and a single input.

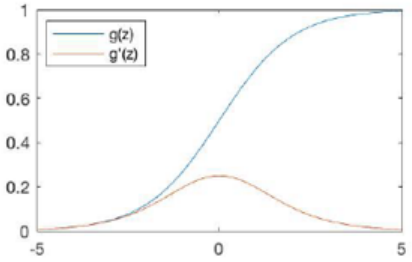
- Let the weights and biases for the first layer(a_1) be $W1=[3]$ and $b1=[1]$, producing an output of $a_1 = W_1x+b_1=3x+1$.

- For the second layer, let the weights and biases be $W2=[2]$ and $b2=[-1]$, producing an output of $a_2 = W_2(a_1)+b_2= 2*a_1-1$.

- The operation of both layers combined is equivalent to $2*(3x+1) - 1 = 6x+1$, which is still a linear transformation of the input x.

Common Activation Functions

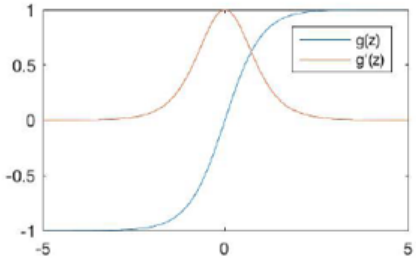
Sigmoid Function



$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

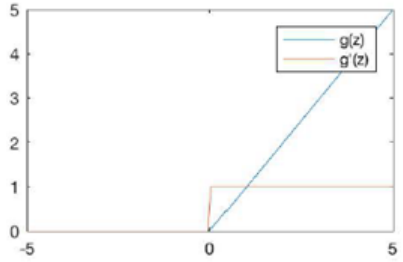
Hyperbolic Tangent



$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

Rectified Linear Unit (ReLU)



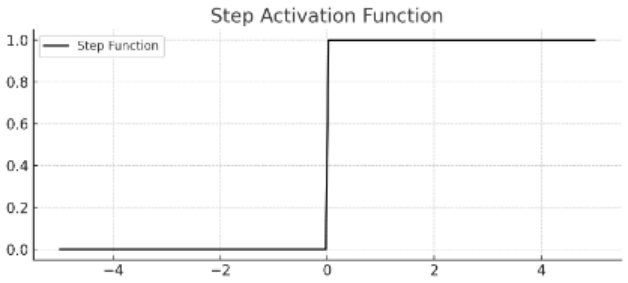
$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

Source: MIT

- This function is one of the simplest activation functions and is mainly used in simple binary classification problems.
- Use it when you need a simple decision-making model and do not require the gradients for learning. It's seldom used in modern deep learning.
- Not Differentiable at the point of transition (jump).

$$f(x) = \begin{cases} 0 \text{ if } 0 > x \\ 1 \text{ if } x \geq 0 \end{cases}$$



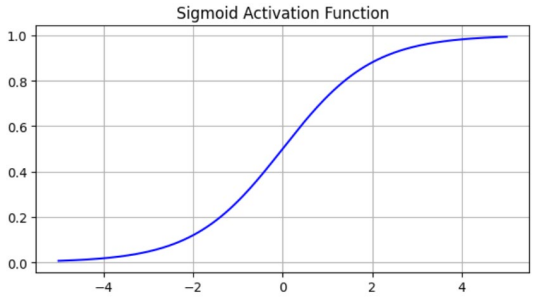
Source: https://www.saedsayad.com/artificial_neural_network.htm

- Sigmoid function, $\sigma(x) = \frac{1}{1+e^{-x}}$, maps input to a value between 0 and 1.
- Commonly used in binary classification as the final layer for binary output.
- **Pros:**

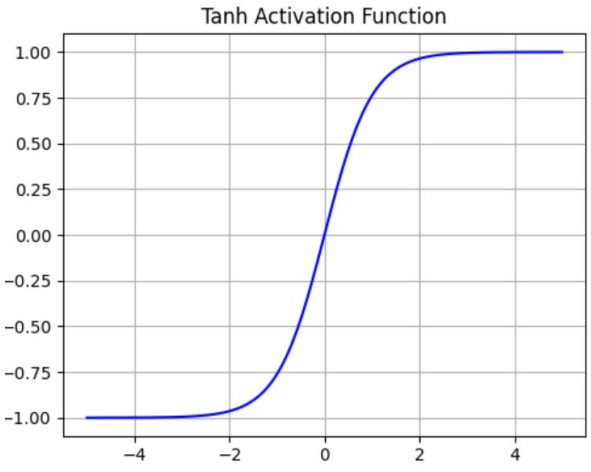
  Smooth gradient, preventing sudden jumps in output values.

  Useful for probabilities since output is in range [0,1].
- **Cons:**

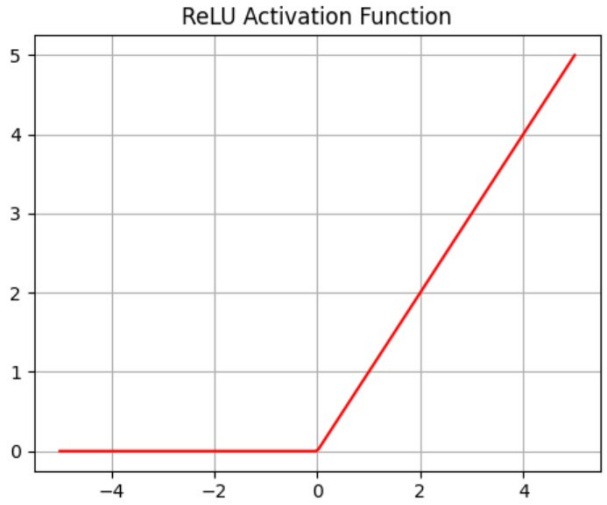
  Vanishing gradient problem, leading to slow or ineffective training in deep networks.



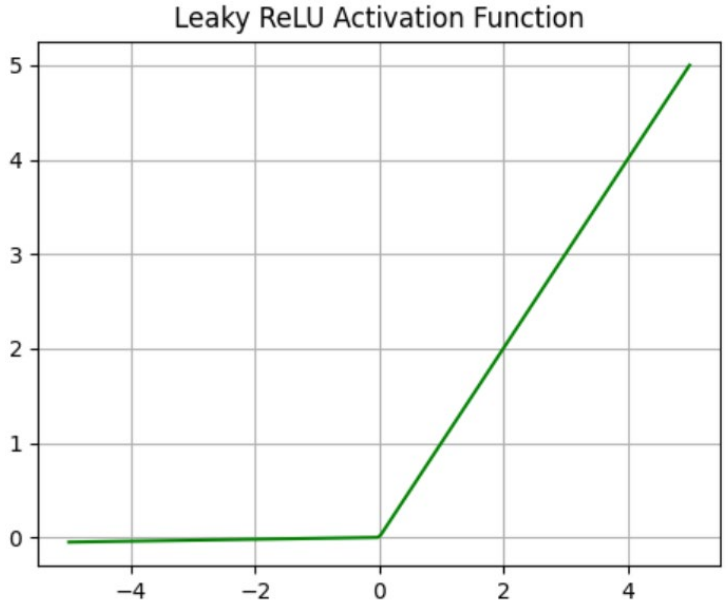
- Equation: $\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$
- Maps input to a value between -1 and 1.
- Commonly used in hidden layers of neural networks.
- **Pros:**
  - Smooth gradient, preventing sudden jumps in output values.
- **Cons:**
  - Vanishing gradient problem can still occur, though less severe than with the sigmoid function.



- Equation: $\text{ReLU}(x) = \max(0, x)$
- Maps input to a value between 0 and x (for $x > 0$)
- Commonly used in hidden layers of neural networks.
- **Pros:**
  - Reduces the likelihood of the vanishing gradient problem.
  - Computationally efficient, allowing for faster training.
- **Cons:**
  - Dying ReLU problem, where neurons can become inactive and only output 0.



- Equation: Leaky ReLU(x) = max(αx,x)
- Similar to ReLU, but allows for a small, non-zero gradient when $x < 0$.
- **Pros:**
  - Attempts to fix the dying ReLU problem by allowing a small gradient when $x < 0$.
  - Retains advantages of ReLU for $x > 0$.
- **Cons:**
  - Leaky ReLU is slightly more complex than ReLU because it involves a multiplication operation for negative inputs

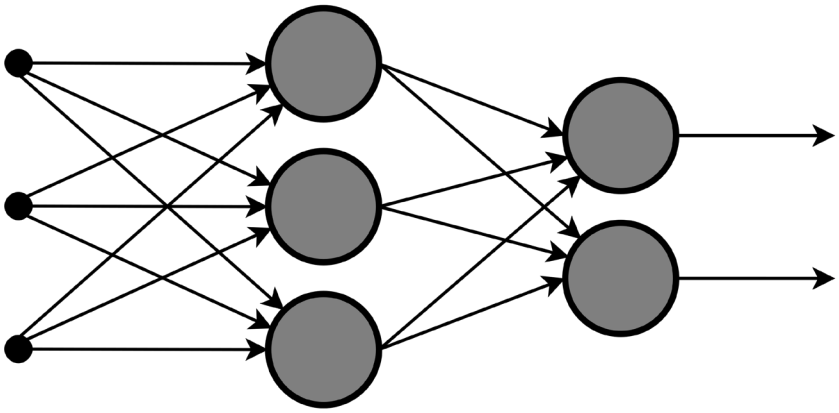


- Equatio$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$
- Used for multi-class classification in the final layer of neural networks.
- **Pros:**
  - Outputs a probability distribution that sums to 1, which is useful for outputting probabilities for mutually exclusive classes.
  - Can handle multiple classes unlike sigmoid.
- **Cons:**
  - Exponentiation can cause issues with numerical stability.

Summary of Activation Functions

Function	Equation	Pros	Cons	Typical Use-Cases
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$	Smooth, outputs between 0 and 1	Vanishing gradient problem	Output layer for Binary classification
Tanh	$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$	Smooth, outputs between -1 and 1	Vanishing gradient problem	Hidden layers
ReLU	$f(x) = \max(0,x)$	Fast to compute, helps with vanishing gradient	Dying ReLU problem	Hidden layers
Leaky ReLU	$f(x) = \max(\alpha x,x)$	Addresses dying ReLU problem	Hyperparameter tuning needed	Hidden layers
Softmax	$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$	Converts scores to probabilities	Computationally expensive	Output layer for classification

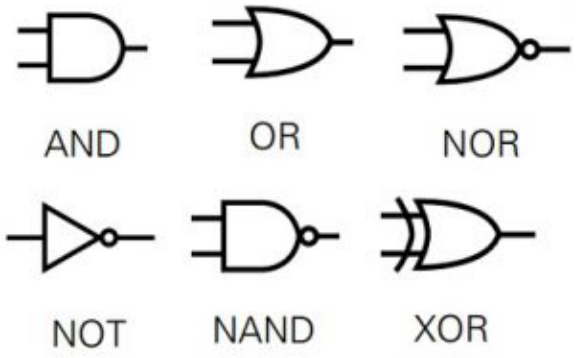
- In general, ReLU is often the default choice (in hidden layers) for many deep learning applications due to its simplicity and efficiency. However, depending on the nature of the problem and the architecture of the neural network, other activation functions might be more appropriate. When in doubt, it's often a good practice to experiment with a few different activation functions and see which one performs best on the validation data.



Neural Networks

Non-linear Examples

- Basic building blocks of digital circuits and computers.
- Implement Boolean operations: AND, OR, NOT, NAND, NOR, XOR, XNOR.
- Process binary inputs to produce a binary output according to a logical function.



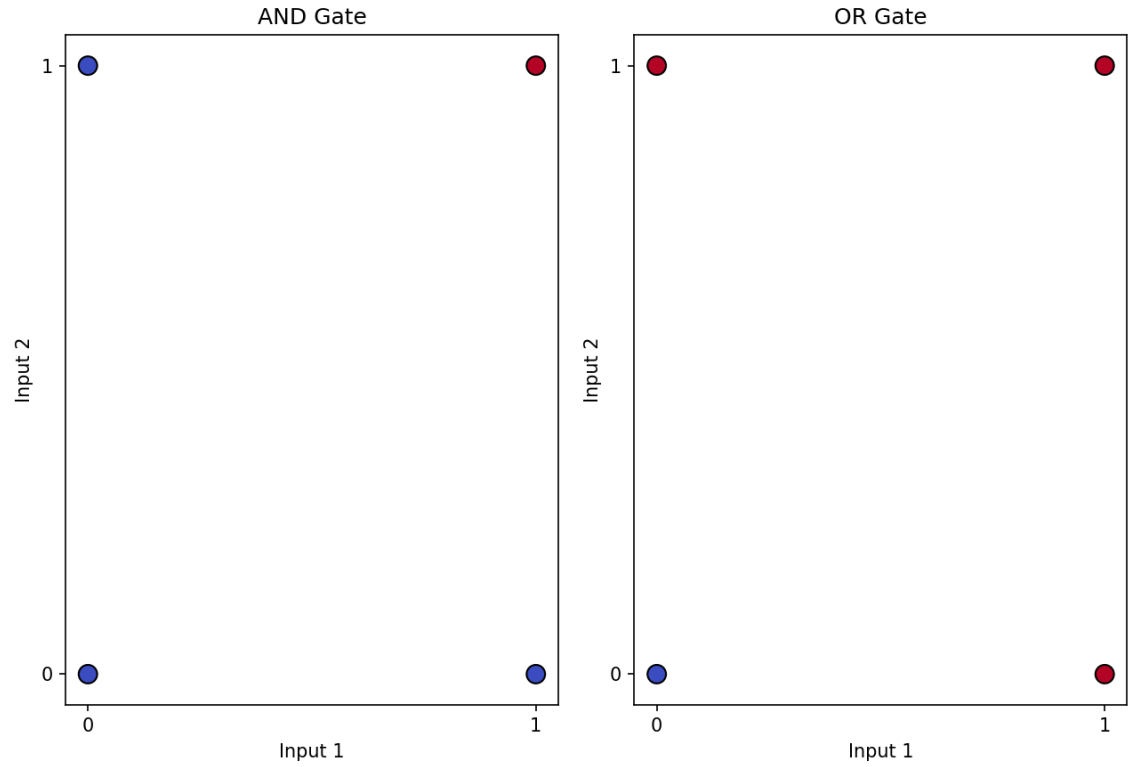
Source: https://www.elprocus.com/basic-logic-gates-with-truth-tables/

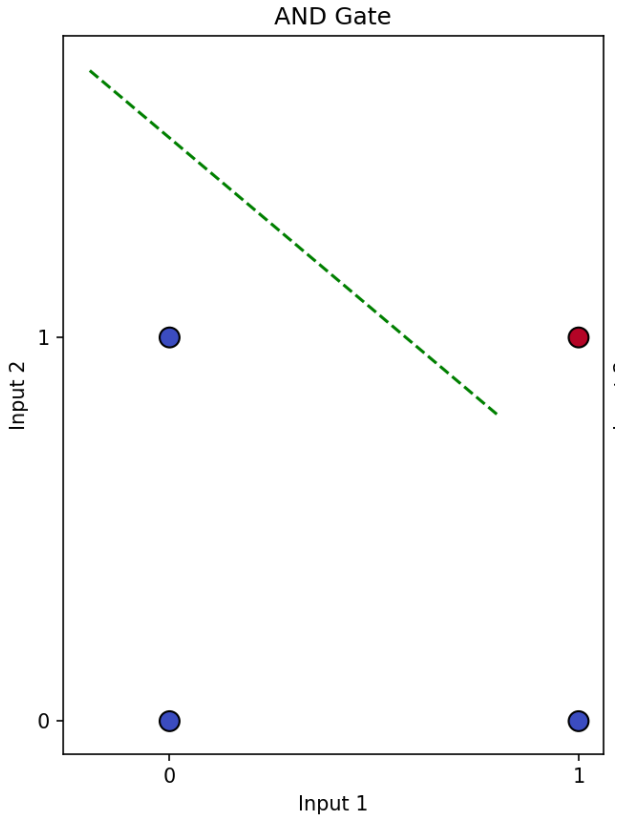
- **And Truth Table:**

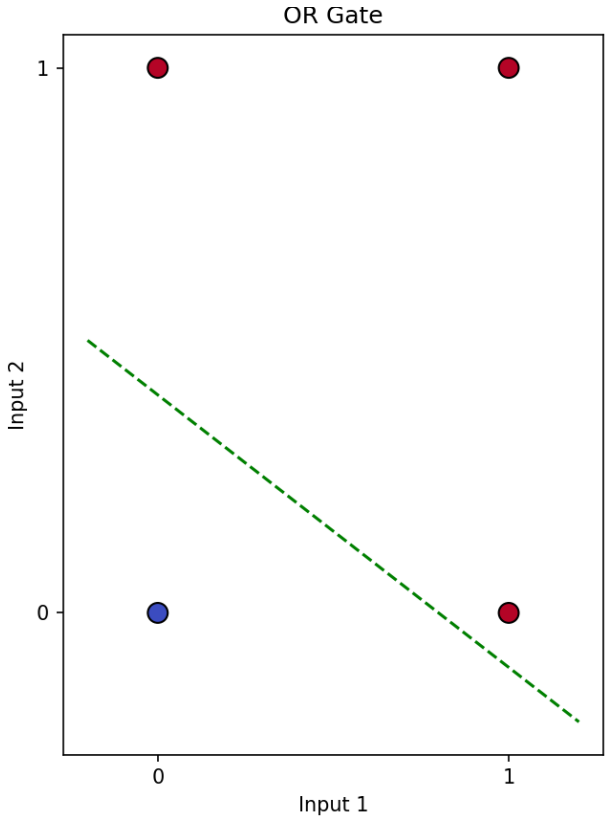
A	B	A And B
0	0	0
0	1	0
1	0	0
1	1	1

Or Truth Table:

A	B	A Or B
0	0	0
0	1	1
1	0	1
1	1	1


AND Gate
1
0
Input 2
0
1
Input 1
OR Gate
1
0
Input 2
0
1
Input 1

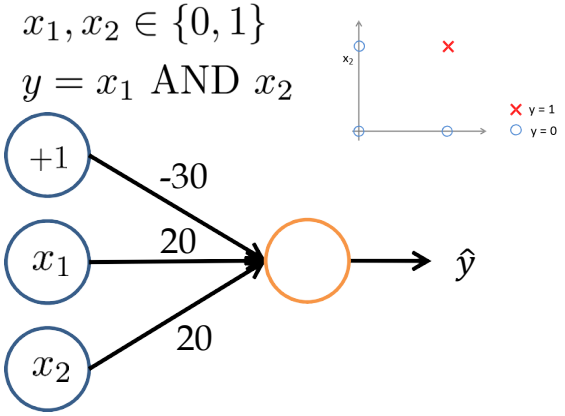

AND Gate
Input 2
1
0
0
1
Input 1

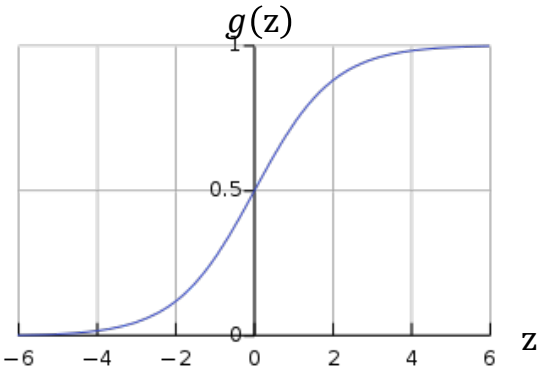

OR Gate
Input 2
1
0
0
1
Input 1

Simple example: AND gate

$x_1, x_2 \in \{0, 1\}$

$y = x_1$ AND x_2



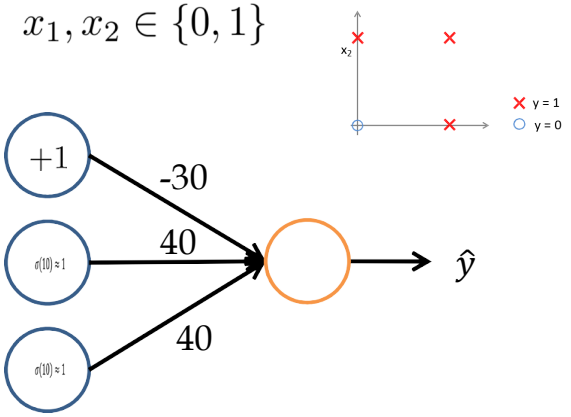


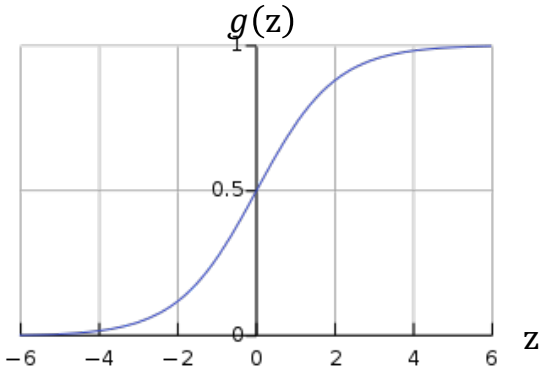
x_1	x_2	$\hat{y}$
0	0	$\sigma(-30) \approx 0$
0	1	$\sigma(-10) \approx 0$
1	0	$\sigma(-10) \approx 0$
1	1	$\sigma(10) \approx 1$

$$\hat{y} = \sigma(20x_1 + 20x_2 - 30)$$

Simple example: OR gate

$x_1, x_2 \in \{0, 1\}$

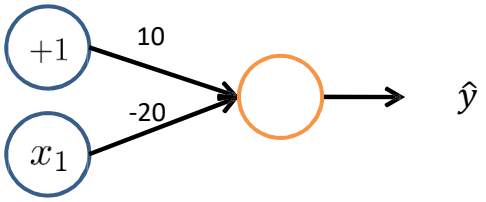




x_1	x_2	$\hat{y}$
0	0	$\sigma(-30) \approx 0$
0	1	$\sigma(10) \approx 1$
1	0	$\sigma(10) \approx 1$
1	1	$\sigma(50) \approx 1$

$$\hat{y} = \sigma(40x_1 + 40x_2 - 30)$$

Negation: NOT gate



x_1	$\hat{y}$
0	$\sigma(10) \approx 1$
1	$\sigma(-10) \approx 0$

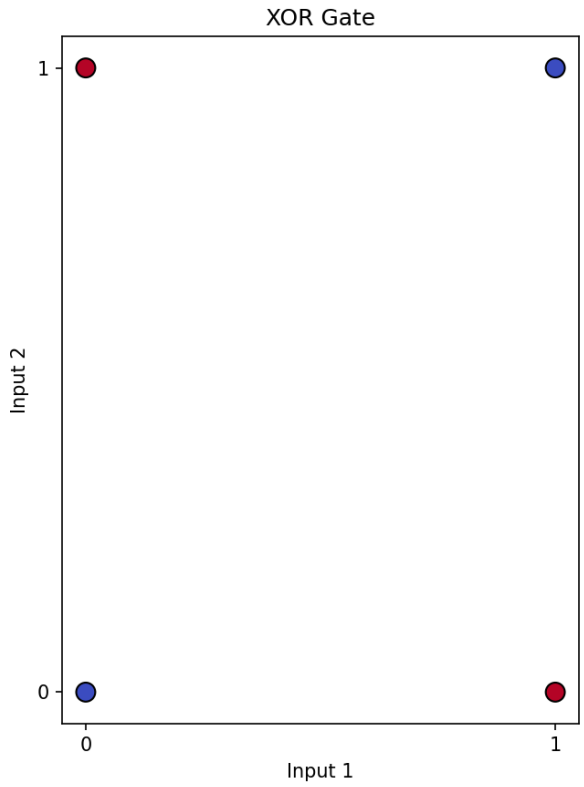
$$\hat{y} = g(-20x_1 + 10)$$

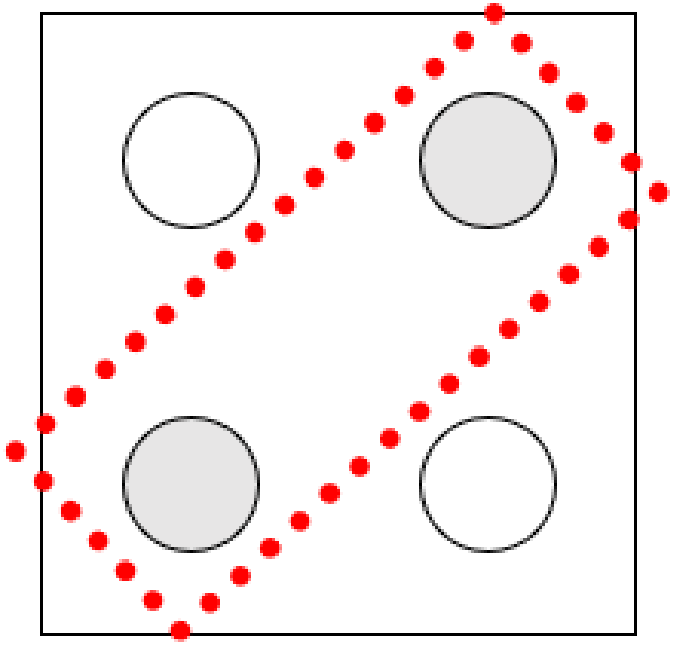
The XOR Problem and Neural Networks

- What is the XOR Problem?
  - XOR is a logical operation that outputs true only when inputs differ.
  - The XOR function maps $(0,0),(1,1) \rightarrow 0$ and $(0,1),(1,0) \rightarrow 1$.
  - It's not linearly separable, making it a challenge for simple classifiers.

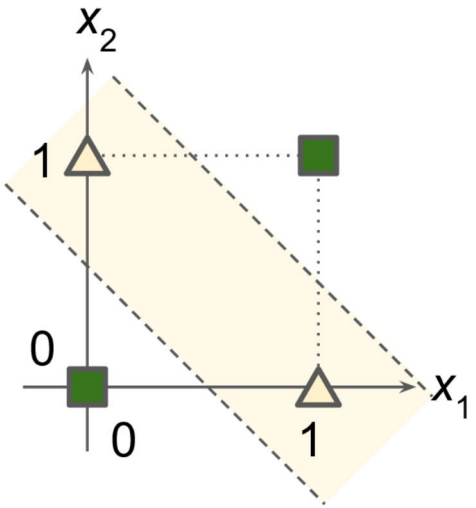
- Why is XOR Important in Neural Networks?
  - Historical Context: Early neural networks struggled with XOR, highlighting limitations.
  - Benchmark for Non-Linearity: Solving XOR proves a model can solve non-linear problems.

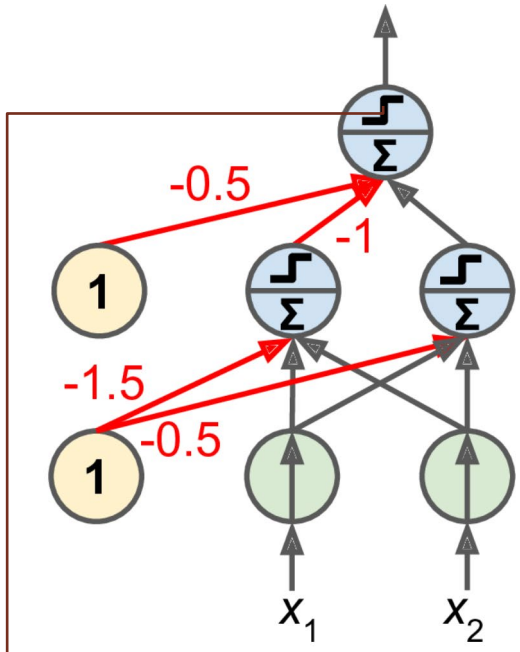
Input 1	Input 2	Input 1 XOR Input 2
0	0	0
0	1	1
1	0	1
1	1	0





MLP that solves XOR problem





- with inputs (0, 0) or (1, 1), the network outputs 0.
- and with inputs (0, 1) or (1, 0) it outputs 1.
- All connections have a weight equal to 1, except the four connections where the weight is shown.

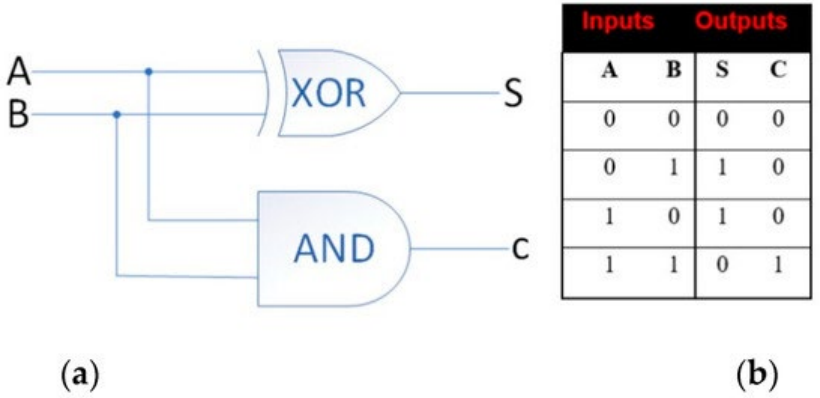
$$f(x) = \begin{cases} 1 & wx + b > 0 \\ 0 & otherwise \end{cases}$$

The Half-Adder: A Practical Application of Logic Gates

- A half adder is a simple circuit that performs binary addition.

- **Truth Table for Half-Adder**:

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



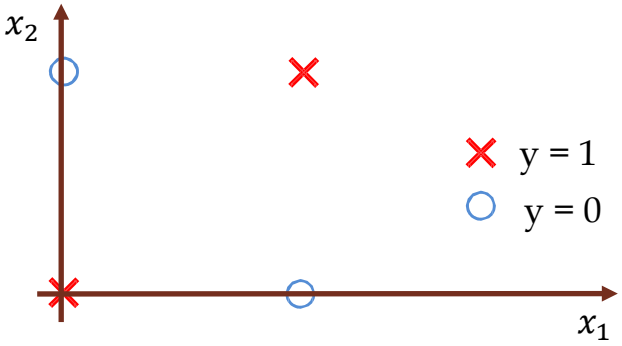
Inputs		Outputs	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

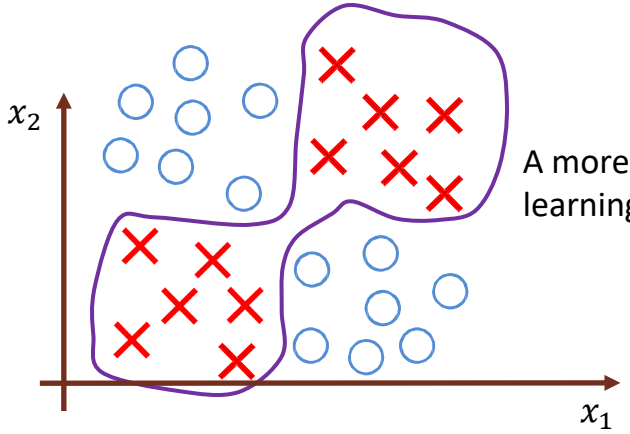
- Logic for Sum: A XOR B
- Logic for Carry: A AND B

- Reference: Mdpi: https://www.mdpi.com/2073-8994/15/5/1063

In-Class Exercise: XNOR gate

x_1, x_2 are binary (0 or 1)



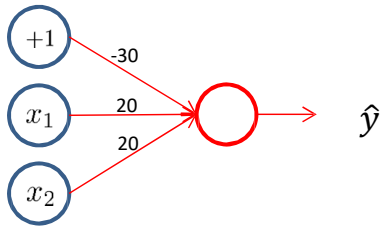


A more complex learning problem

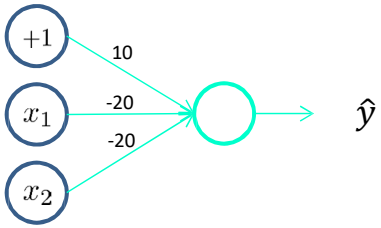
$$y = x_1\ XNOR\ x_2 = NOT(x_1\ XOR\ x_2)$$

Input		Output
A	B	A XNOR B
0	0	1
0	1	0
1	0	0
1	1	1

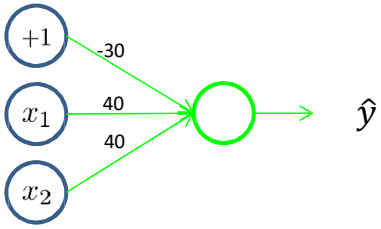
In-Class Exercise: XNOR gate



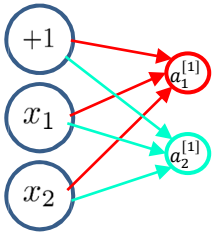
x_1 AND x_2



(NOT x_1) AND (NOT x_2)

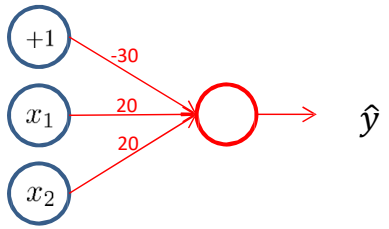


x_1 OR x_2

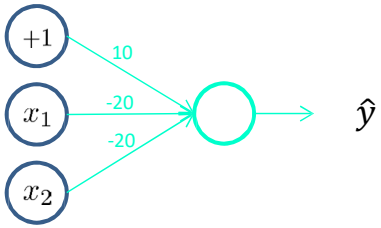


x_1	x_2	$a_1^{[1]}$	$a_2^{[1]}$	$\hat{y}$
0	0	0	1	
0	1	0	0	
1	0	0	0	
1	1	1	0	

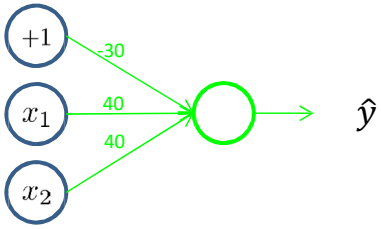
In-Class Exercise: XNOR gate



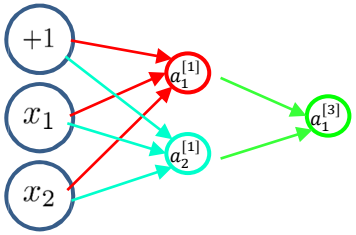
x_1 AND x_2



(NOT x_1) AND (NOT x_2)

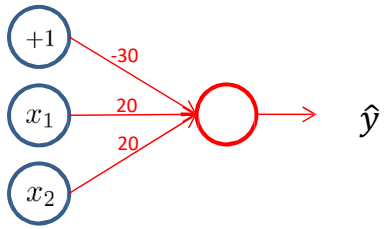


x_1 OR x_2

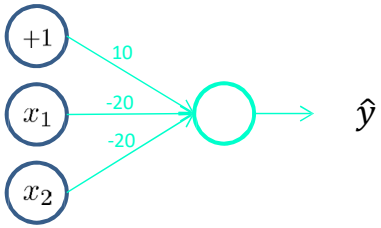


x_1	x_2	$a_1^{[1]}$	$a_2^{[1]}$	$\hat{y}$
0	0	0	1	
0	1	0	0	
1	0	0	0	
1	1	1	0	

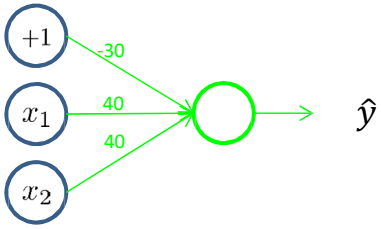
In-Class Exercise: XNOR gate



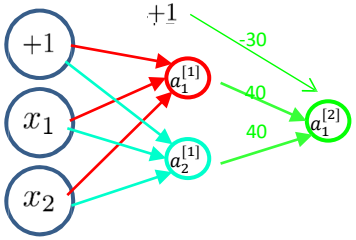
x_1 AND x_2



(NOT x_1) AND (NOT x_2)

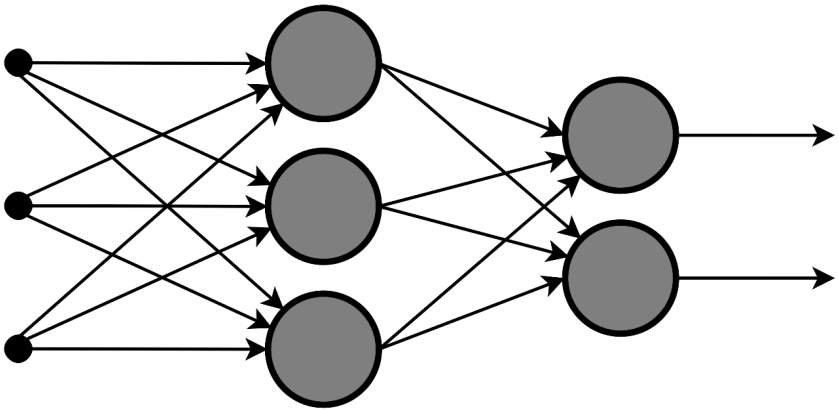


x_1 OR x_2



x_1	x_2	$a_1^{[1]}$	$a_2^{[1]}$	$\hat{y}$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

Neural Networks



Vanishing, Exploding Gradient Descents and Dying ReLU Problem

Vanishing Gradient Problem

- The Vanishing Gradient Problem refers to the diminishing value of gradients during backpropagation.
- Occurs predominantly in deep neural networks, particularly with sigmoid or tanh activation functions.
- Affects the weight updates during training, making them negligible.
- Leads to slow or stalled learning

Vanishing Gradient Problem: Illustration

Consider a deep network using a sigmoid activation function. For small input values to the sigmoid function, the gradients become very small.

Given the sigmoid function: $\sigma(z)=1/(1 + e^{-z})$

- The range of the sigmoid function varies between 0 and 1.

Its derivative is: $\sigma'(z)=\sigma(z)(1-\sigma(z))$

For values of z that are very large in magnitude (either positively or negatively), $\sigma(z)$ approaches 0 or 1. This means the derivative approaches 0.

Exploding Gradient Problem

Opposite of the vanishing gradient problem.

Gradients become too large for the model to handle during backpropagation.

Typically happens in deep networks with long sequences.

Large gradients lead to very large weight updates, destabilizing the learning process.

Dying ReLU Problem

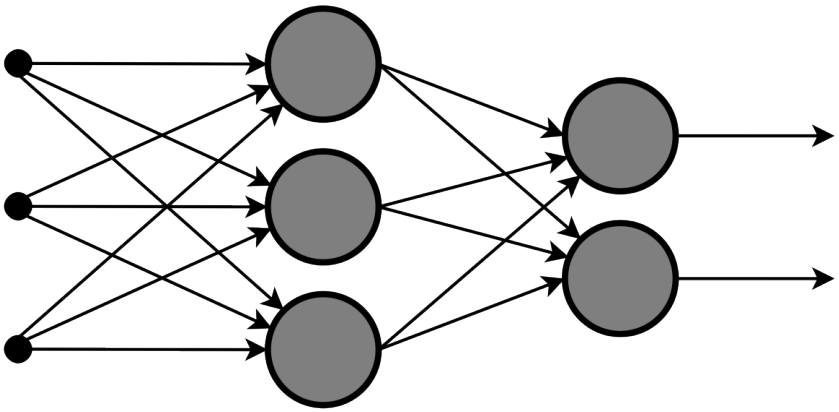
- The Dying ReLU problem refers to a situation where a neuron's output becomes zero for all inputs during training, causing gradients to perpetually be zero.

- This issue often occurs in ANNs that use ReLU activation function when the parameters of a neuron are updated in such a way that the neuron only receives negative inputs, resulting in a zero gradient that prevents further learning.

- To mitigate this problem, variants of ReLU such as Leaky ReLU can be used, which allows for a small, non-zero gradient when the input is negative.

Solutions to Gradient Problems

- **Solutions for the Vanishing Gradient Problem:**
  - Changing Activation Functions: Use ReLU or Leaky ReLU to ensure a constant gradient for positive inputs, preventing gradients from vanishing.
- **Solutions for the Exploding Gradient Problem:**
  - Gradient Clipping: Set a threshold value to scale down gradients when they are too large, preventing excessively large updates to weights.
  - Batch Normalization: Standardize the outputs of each layer to have a mean of 0 and a standard deviation of 1, which can help control the scale of gradients.



Neural Networks

Gradient Descent Variants

Variants of Gradient Descent

- Batch Gradient Descent: The whole dataset is used to compute the gradient of the cost function with respect to the parameters.
- Stochastic Gradient Descent (SGD): Only one example from the dataset is used at each iteration to update the parameters.
- Mini-batch Gradient Descent: A random subset of the dataset is used at each iteration to update the parameters.

Batch Gradient Descent

- Batch Gradient Descent computes the gradient using the entire dataset.
- It provides a stable and less erratic journey down to the minimum.
- However, it requires more computation power and memory, making it inefficient for large datasets.

Batch Gradient Descent - Steps

Initialization: Initialize the weights (and biases), usually with small random numbers.

Epoch 1:

You process the entire dataset as one large batch.

You compute the gradient of the average loss function with respect to the weights (and biases) over the entire dataset.

You then update the weights (and biases) of the network once, based on the calculated gradient and the chosen learning rate.

Epoch 2 and Onwards:

For each subsequent epoch, you use the updated weights (and biases) from the end of the previous epoch.

This continues for the specified number of epochs, or until convergence criteria are met.

Batch Pseudocode

Here's a simplistic representation of Batch Gradient Descent in Pseudocode.

It highlights the cycle through epochs and adjustment of weights using the entire dataset to calculate the gradient.

```
for i in range(epochs):
    grad = compute_gradient(X_train,
    y_train) weights = weights - learning_rate
    * grad
```

Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) utilizes a single, randomly selected instance from the dataset per iteration.

SGD can escape local minima more effectively than Batch GD due to its stochastic nature.

However, its journey to the minimum is noisier, often oscillating and potentially overshooting.

The frequent updates with high variance often lead to a detailed exploration of the parameter space.

Stochastic Gradient Descent (SGD) - Steps

Epoch 1:

You iterate over each individual data point in the dataset.

For each data point, you compute the gradient of the loss function with respect to the weights, and then update the weights immediately.

This process is repeated for every single data point in the dataset, causing many updates to the weights in a single epoch.

Epoch 2 and Onwards:

For each subsequent epoch, you continue using the updated weights from the previous data point and go through the entire dataset again, updating the weights after each data point.

Stochastic Gradient Descent Pseudocode

Here's the stripped-down representation of Stochastic Gradient Descent in Pseudocode.

This illustrates the individual processing of each data point, leading to more frequent and varied updates.

```
for i in range(epochs):
    for x, y in zip(X_train, y_train):
        grad = compute_gradient(x,
        y)
        weights = weights - learning_rate * grad
```

Mini-Batch Gradient Descent

Mini-Batch Gradient Descent strikes a balance, utilizing a subset of data points at each step.

It combines the computational efficiency of Stochastic with the stability of Batch Gradient Descent.

It's less erratic than Stochastic GD and faster than Batch GD, especially in large datasets.

The balance between speed and stability makes it the preferred choice in many applications.

Mini-batch Gradient Descent - Steps

Initialization: You initialize the weights (usually randomly) before the start of the first epoch.

Epoch 1:

You divide the dataset into small subsets (mini-batches) of data points.

For each mini-batch, you compute the gradient of the average loss function with respect to the weights and then update the weights immediately.

This process is repeated for every mini-batch in the dataset, causing multiple updates to the weights in a single epoch.

Epoch 2 and Onwards:

For each subsequent epoch, you continue using the updated weights from the previous mini-batch and go through the entire dataset again, updating the weights after each mini-batch.

Mini-Batch Gradient Descent: Pseudocode

Here's the streamlined representation of Mini-Batch Gradient Descent in Pseudocode.

It illustrates the process of handling data in batches, providing a middle ground between the speed of Stochastic and the stability of Batch Gradient Descent.

```
for i in range(epochs):
    for batch in get_batches(X_train, y_train, batch_size):
        grad = compute_gradient(batch)
        weights = weights - learning_rate * grad
```

When to Use Which?

The choice among the variants is influenced by practical constraints like dataset size, memory availability, and the need for precision.

For large datasets or limited computational resources, Mini-Batch or Stochastic GD might be preferable.

For applications where computational resources are abundant, Batch GD might be suitable.

References

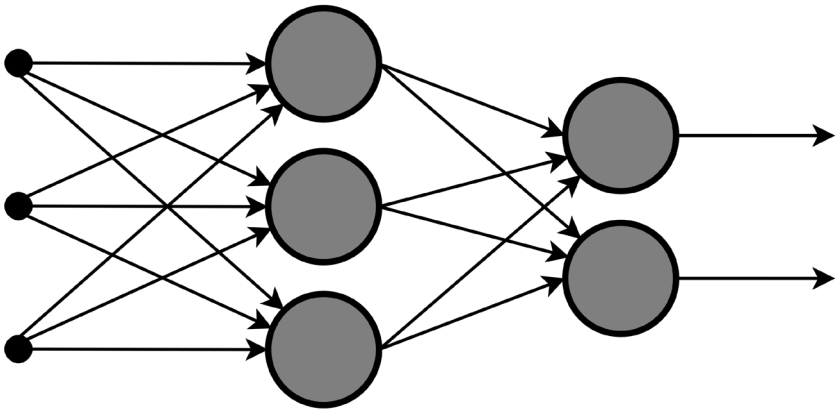
In addition to the references in each slide, the following are leveraged throughout this lecture:

DeepLearning.AI(https://www.deeplearning.ai/)

Backup

Derivatives Cheat Sheet

Derivative Rule	Example
Power Rule: $d/dx\ x^n = nx^{(n-1)}$	$d/dx\ x^3 = 3x^2$
Chain Rule: $d/dx\ f(g(x)) = f'(g(x)) * g'(x)$	$f(x) = \ln(x^2 + 1)$, $d/dx = 2x / (x^2 + 1)$
Natural Logarithm: $d/dx\ \ln(x) = 1/x$	Example: $d/dx\ \ln(x^2) = 2/x$
Exponential Function: $d/dx\ e^x = e^x$	$d/dx\ e^{(2x)} = 2e^{(2x)}$
Derivative of a Constant: $d/dx\ c = 0$	$d/dx\ 7 = 0$
Constant Multiplied by a Variable: $d/dx\ (c*f(x)) = c * d/dx\ f(x)$	$d/dx\ (5x^2) = 10x$



Neural Networks

Importance of Differentiability in Activation Functions

1. **Definition**: Function has a derivative at each point in its domain.

2. **Derivative**: Slope of the tangent line at each point.

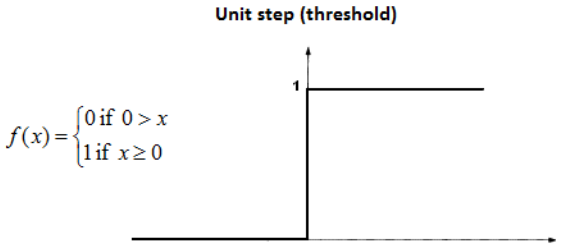
3. **Why Important?**: Facilitates Backpropagation: Differentiability is essential for backpropagation, the algorithm used to train neural networks, where gradients of the loss function are propagated backwards through the network to update weights.

4. **Example**: $f(x)=x^2$ is differentiable; $f'(x)=2x$.

1. **Equation**: $f(x)=\{0, x< 0;1, x >= 0\}$

2. **Issue with Step Function**:
   - Derivative undefined at $x=0$.
   - Derivative zero almost everywhere else.

3. **Implication**: Cannot be used with Gradient Descent-based learning algorithms.



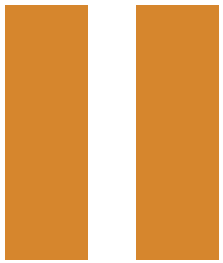
1. **Definition**: Neurons output zero constantly.

2. **Cause**: Weights won't update due to zero gradients for negative inputs.

3. **Example**: For weights $w1=-0.5, w2=-0.5$, ReLU output is zero for positive $x1, x2$.

4. **Effect**: Neuron becomes "dead."

5. **Why it Happens**: Poor initialization, high learning rate.



BREAK

Please come back @

2:25 PM EST

1:25 PM CST

Hands On…..

References

In addition to the references in each slide the below are leveraged throughout this course.

Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.

VanderPlas, J. (2017). Python Data Science Handbook. O'Reilly.
Wolff, S. G. (2018). Less is more: optimizing classification performance through feature selection in a very-high-resolution remote sensing object-based urban application. GIScience & Remote Sensing. doi:10.1080/15481603.2017.1408892

Prince, S. J. (January 28, 2024). Understanding Deep Learning. MIT Press.