Neural Nets and Back Propagation

Anis Yazidi

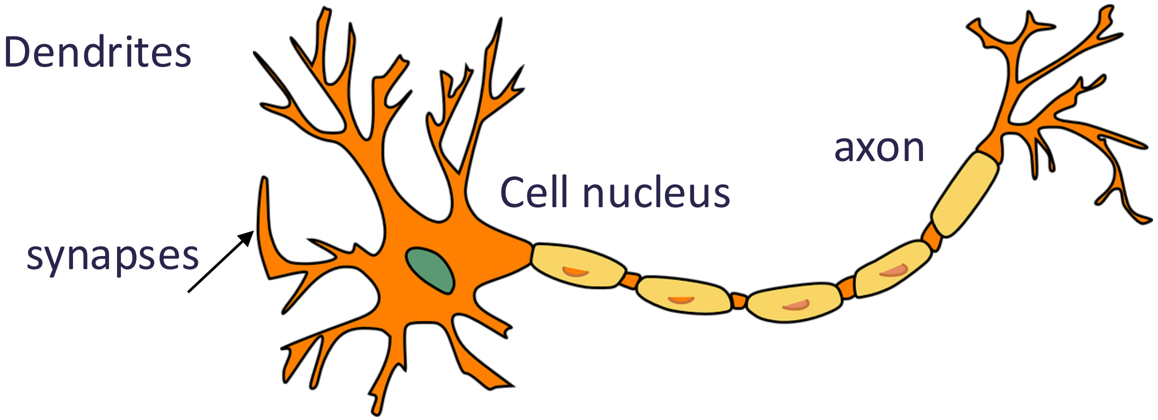
Assoc. Prof. UiO

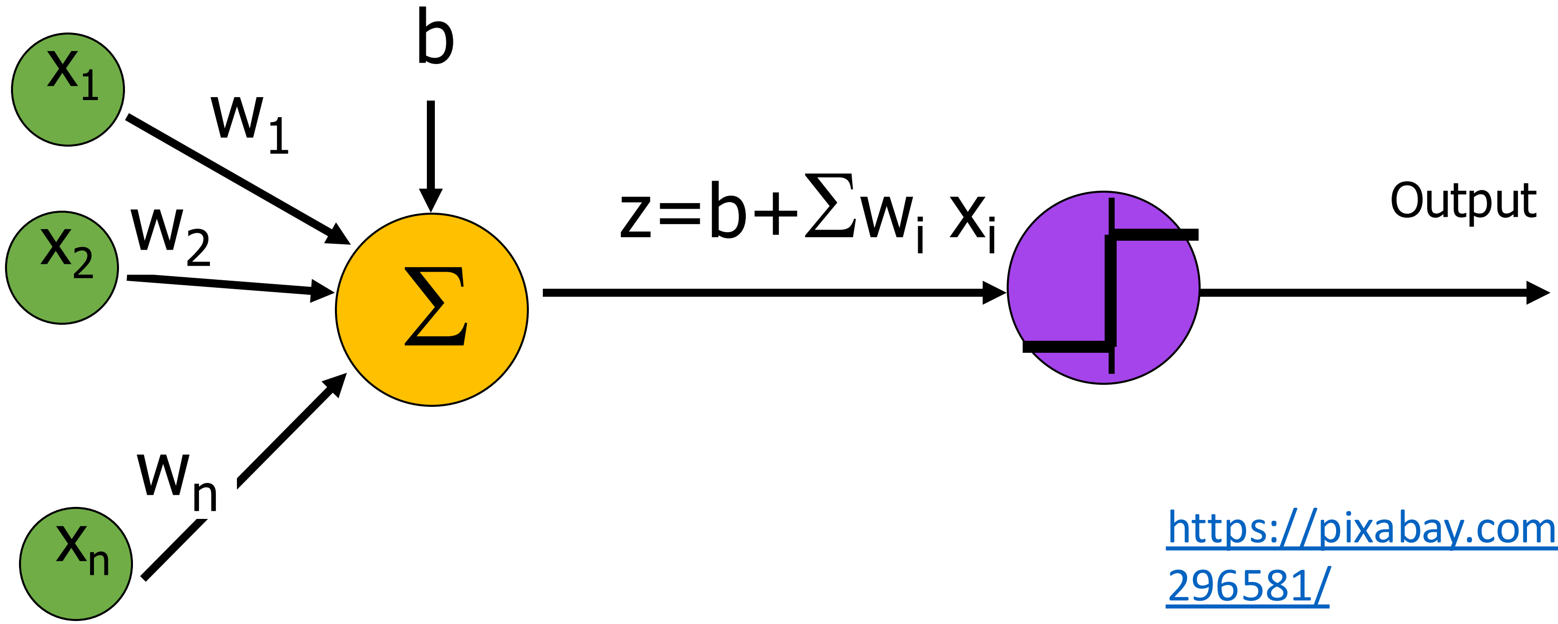
IFI

Plan

- Introduction to Neural Networks
- Backpropagation algorithm
- Learned representation/properties in deep learning
- Training techniques: "making it work"

A perceptron: model of a neuron

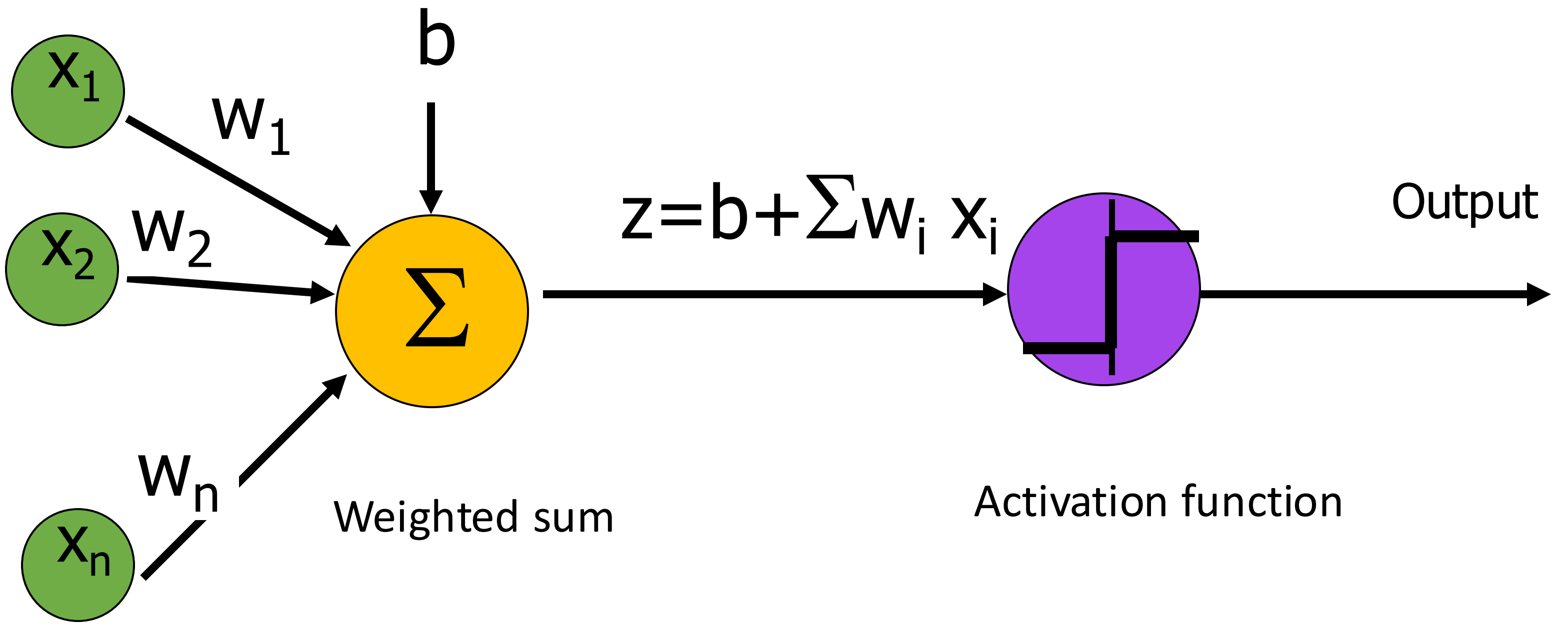


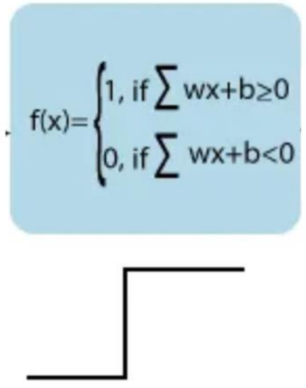


Biology	Artificial NN
Dendrites	Inputs
Synapses	Weighted
Cell nucleus	Weight sum
Output	Axon

https://pixabay.com/vectors/neuron-nerve-cell-axon-dendrite-296581/

Mathematical model: firing with binary actuation function (threshold-based)

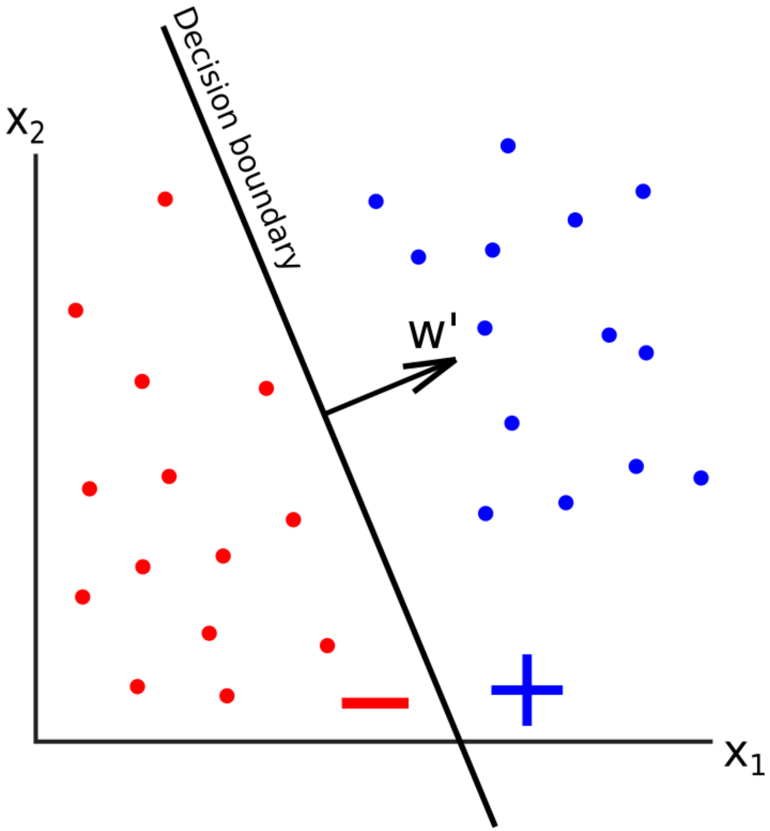




Activation

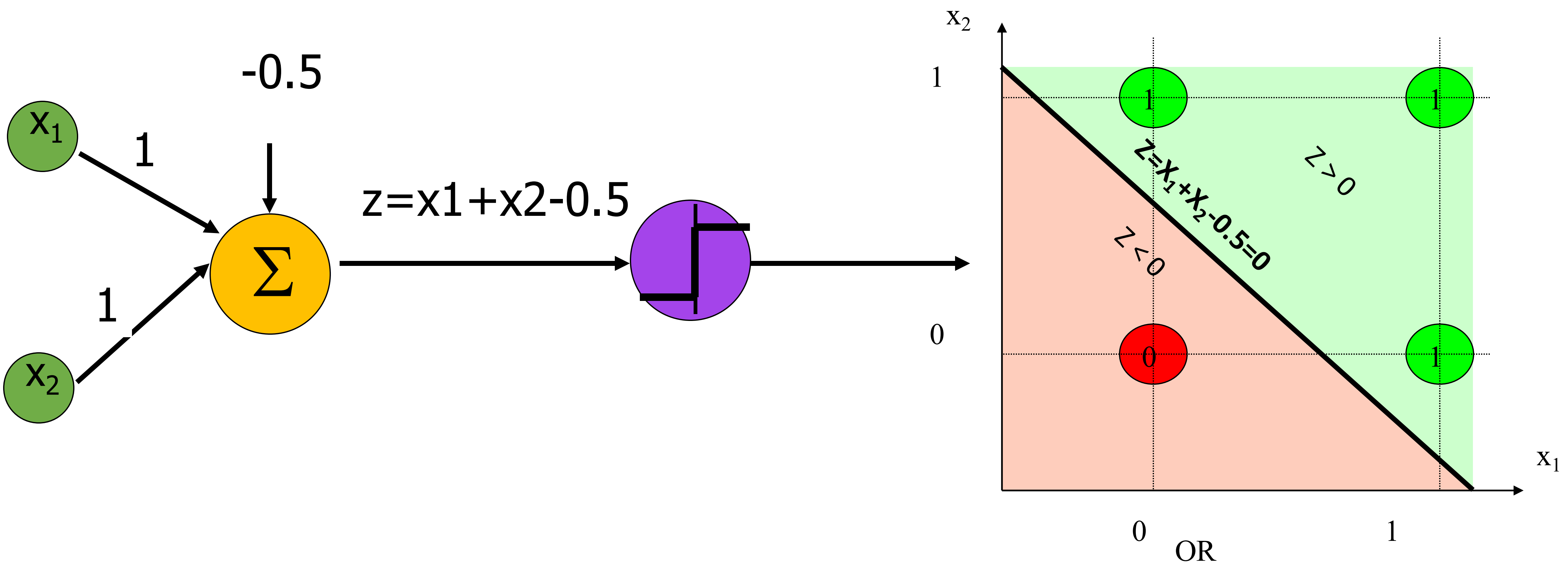
How biological neurons are fired:
They can either trigger an action potential (activate) or not, depending on whether the overall signal strength from the synaptic inputs reaches a certain threshold.

A perceptron: classifier with decision limit z=0

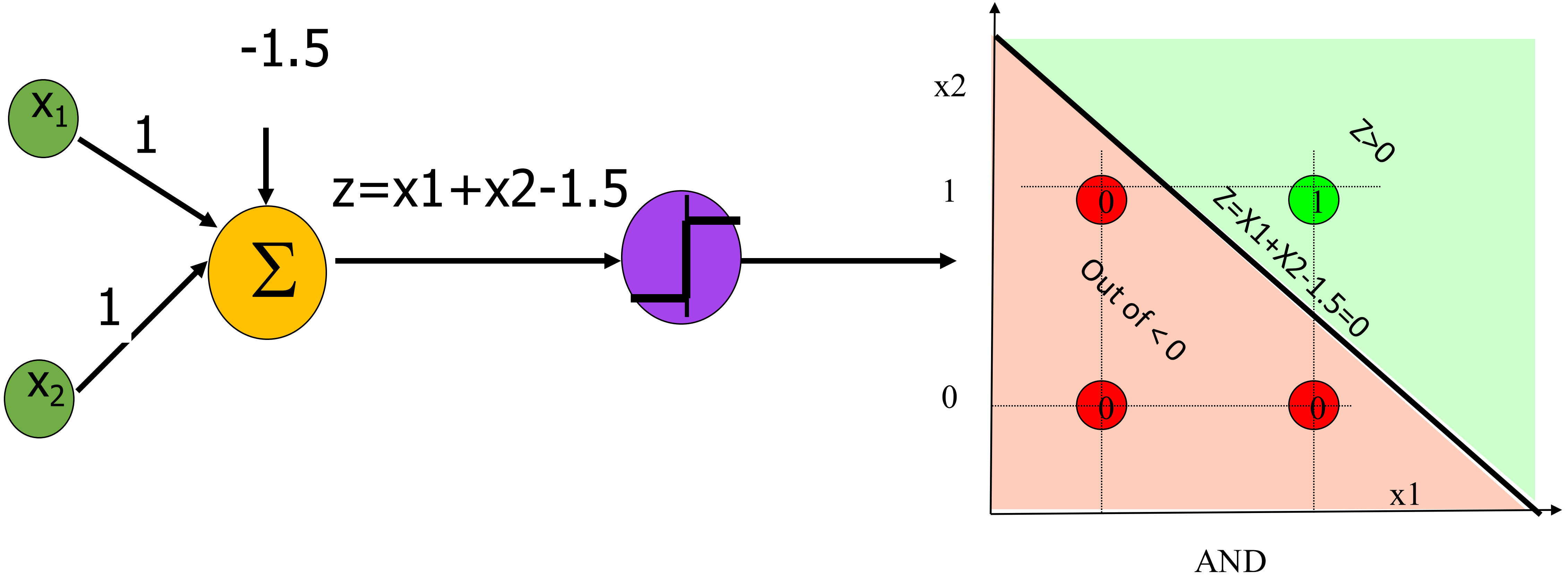


- Researchers in the field of artificial intelligence between the 1960s and 1980s were focused on logic and symbol manipulation.
- They attempted to use perceptron to represent and learn logical expressions.

OR with a single threshold-based perceptron

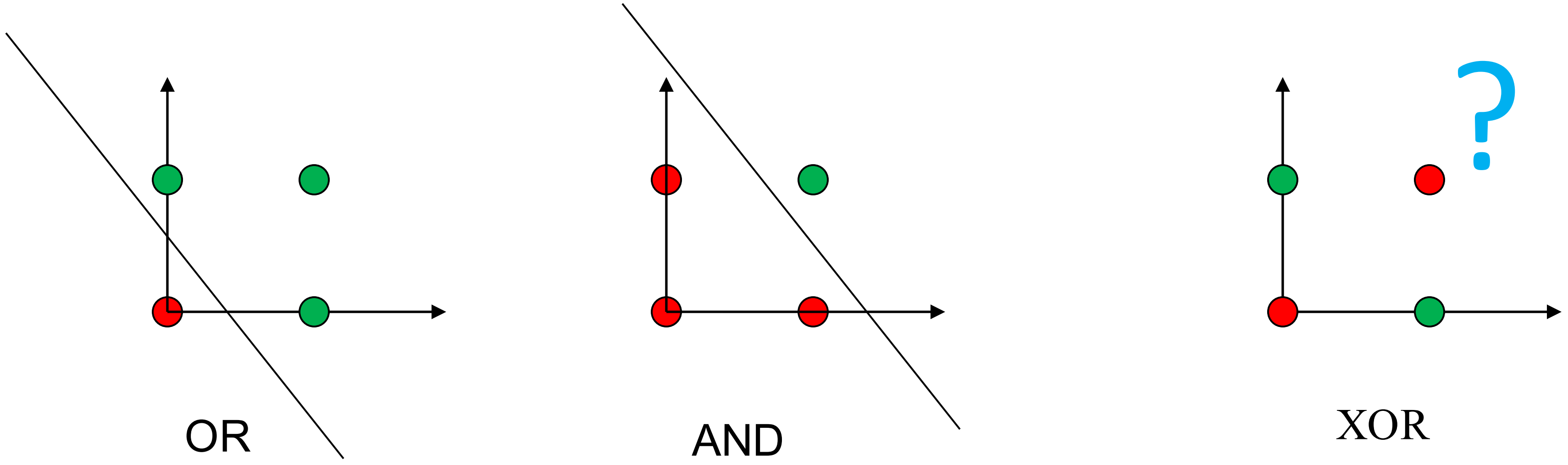


AND with a single threshold-based perceptron



Limitation of a single perceptron

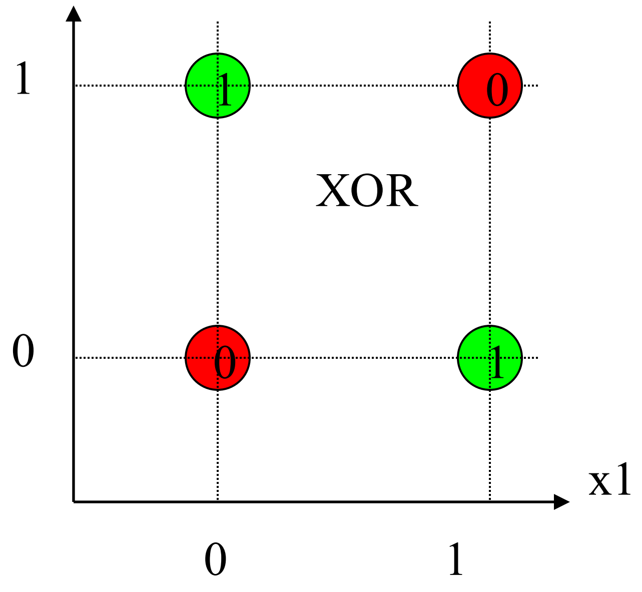
- The problem with a single perceptron is that it can only handle problems where the data is linearly separable.
- XOR is not linearly separable and cannot be represented by a single perceptron



Making XOR with help of AND and OR

XOR(x1,x2) ⇔ OR(x1,x2)) AND NOT(AND(x1,x2))

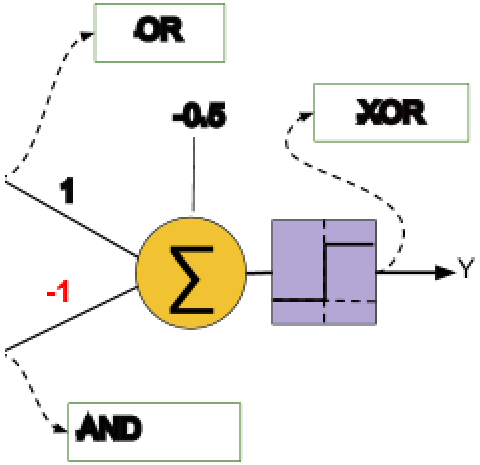
$X1 \oplus X2 \Leftrightarrow (X1 \vee X2) \wedge \neg(X1 \wedge X2)$

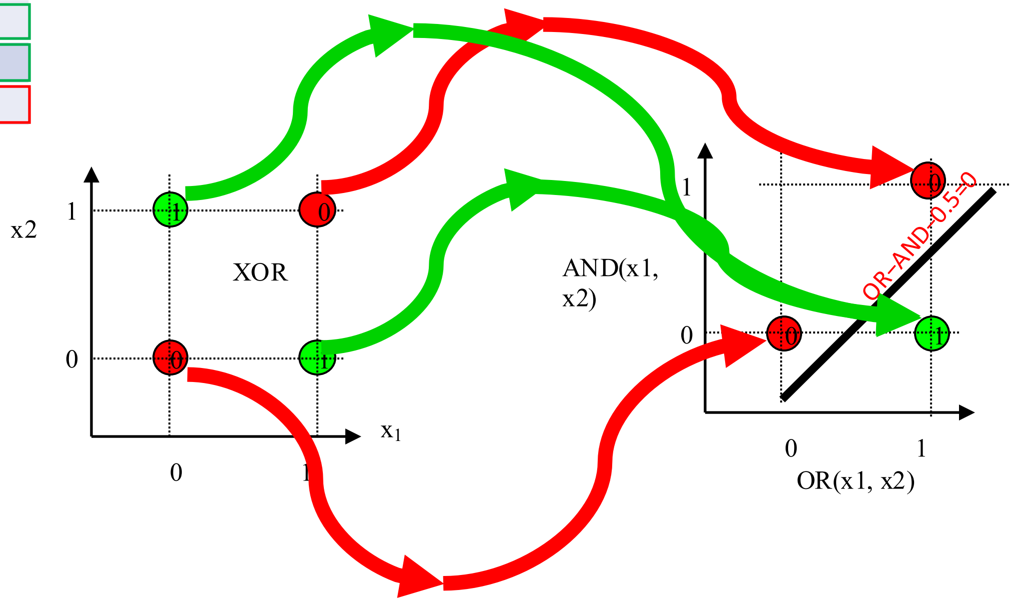


Multi-Layer Perceptron (MLP)

- We can use new axes to represent XOR: with OR and AND as new axes.

Original (x_1, x_2)	OR (x_1, x_2)	AND (x_1, x_2)	New coordinates (OR, AND)	XOR-value
(0, 0)	0	0	(0, 0)	0
(0, 1)	1	0	(1, 0)	1
(1, 0)	1	0	(1, 0)	1
(1, 1)	1	1	(1, 1)	0

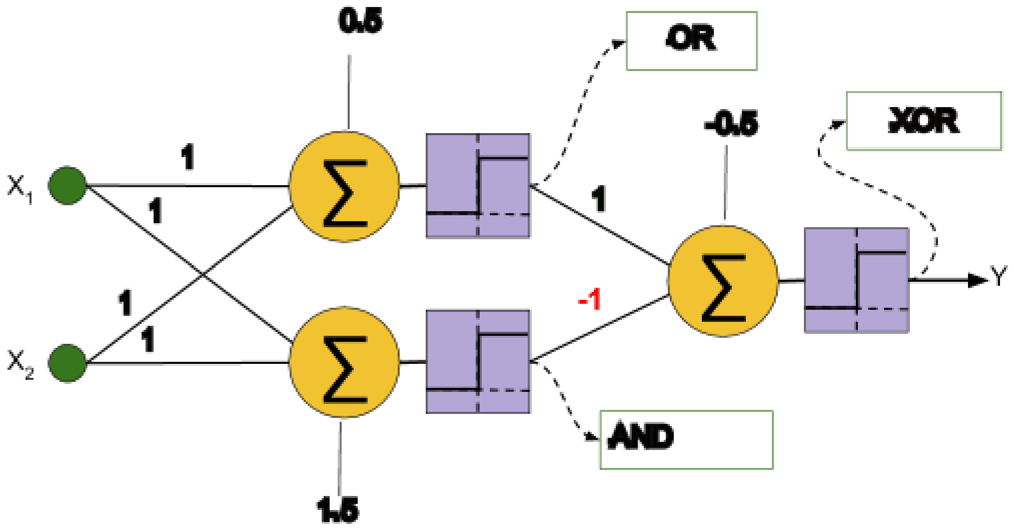


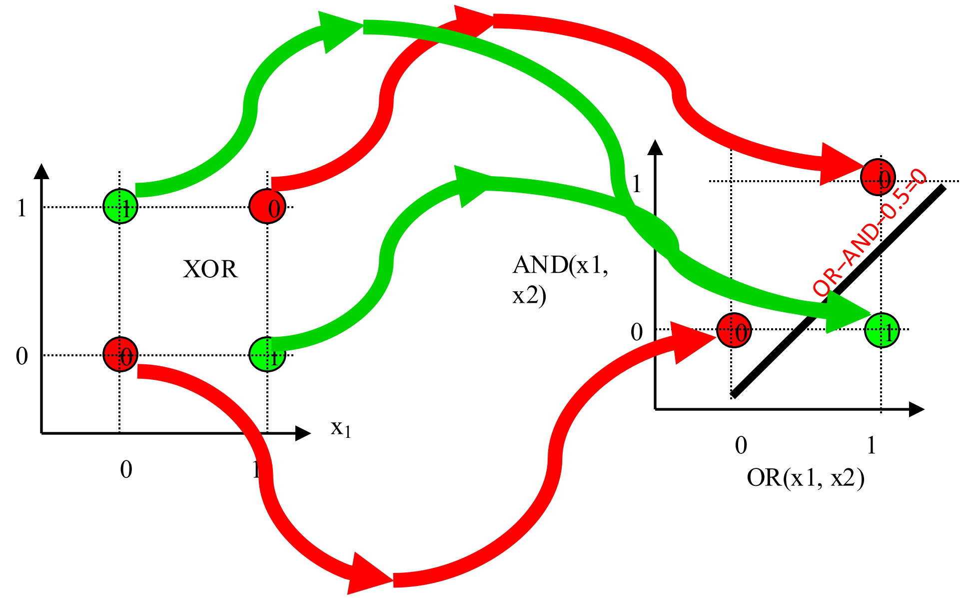


Multi-Layer Perceptron (MLP)

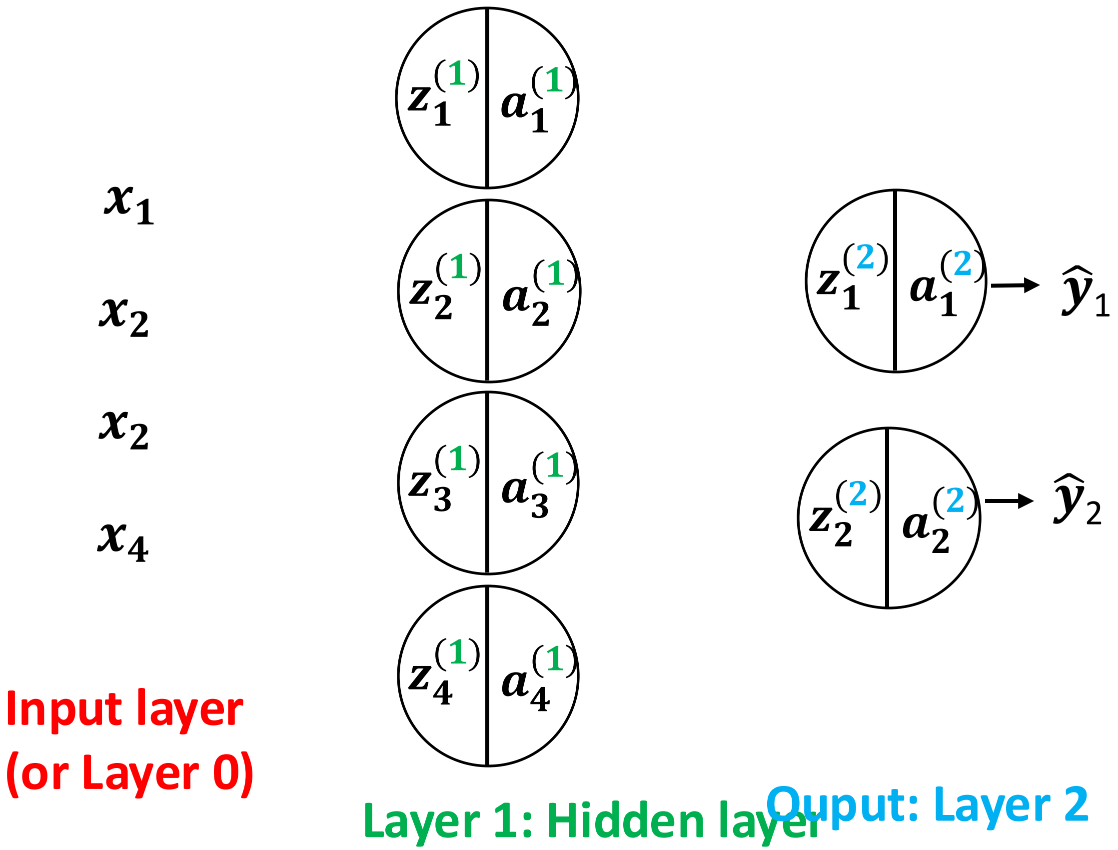
- XOR can be represented by a two-layer neural network.
- We can stack layers: In the first layer, we calculate OR and AND, and in the second layer, we use the output of OR and AND as input to calculate the final output.

Original (x_1, x_2)	OR (x_1, x_2)	AND (x_1, x_2)	New coordinates (OR, AND)	XOR-value
(0, 0)	0	0	(0, 0)	0
(0, 1)	1	0	(1, 0)	1
(1, 0)	1	0	(1, 0)	1
(1, 1)	1	1	(1, 1)	0





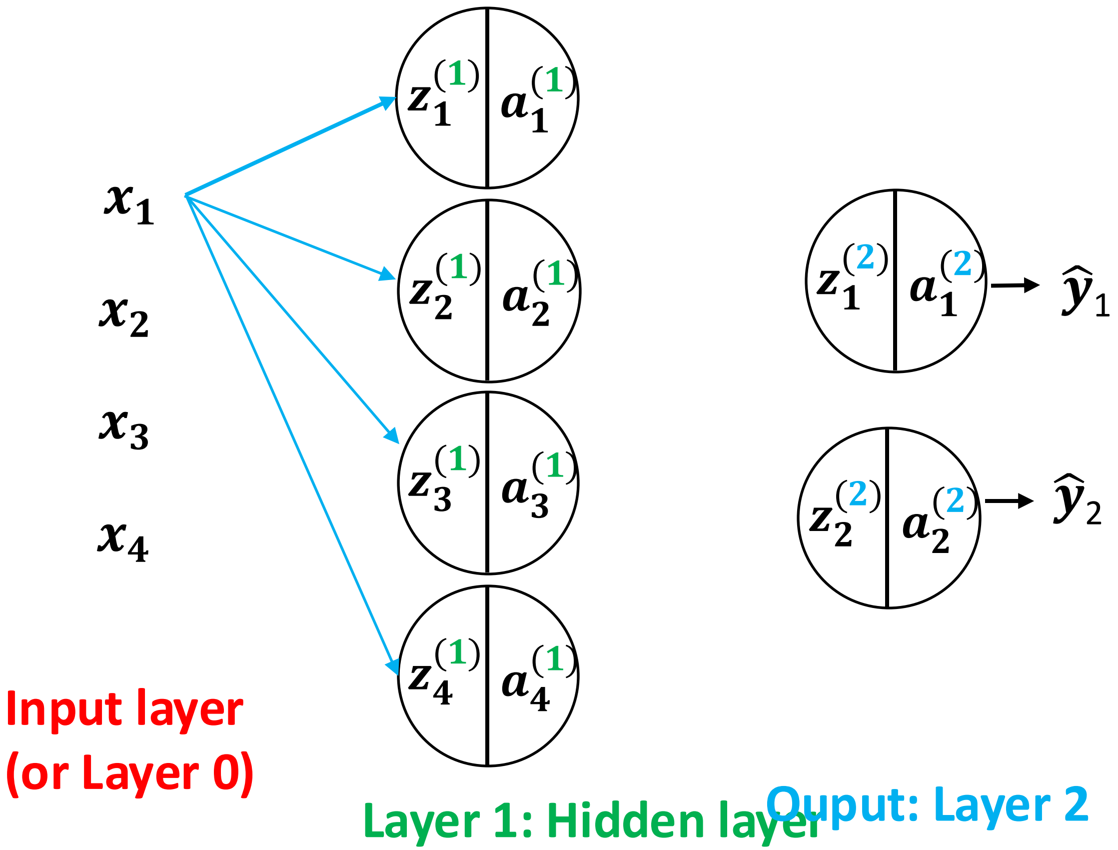
Architecture of a Neural Network



A neural network consists of:

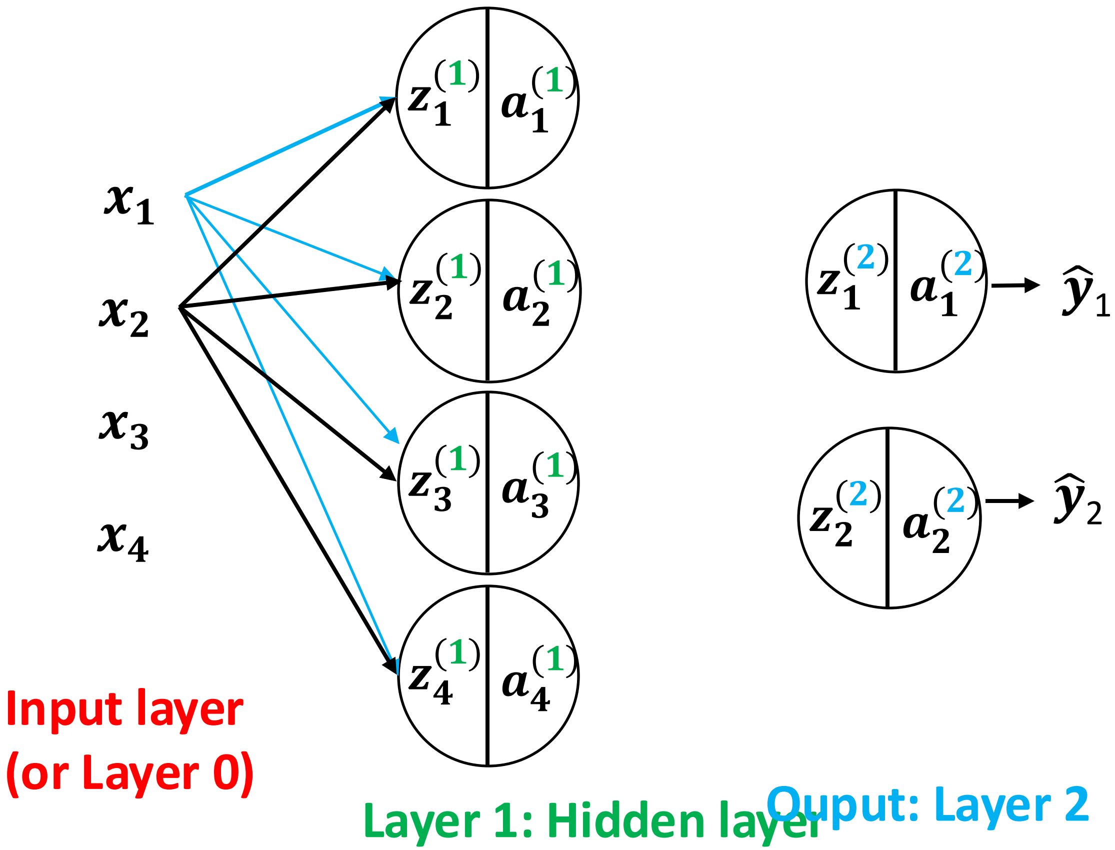
- Input layer: receives raw data
- Hidden layers: Performs calculations.
- Output layer: Returns the result.

Architecture of a Neural Network



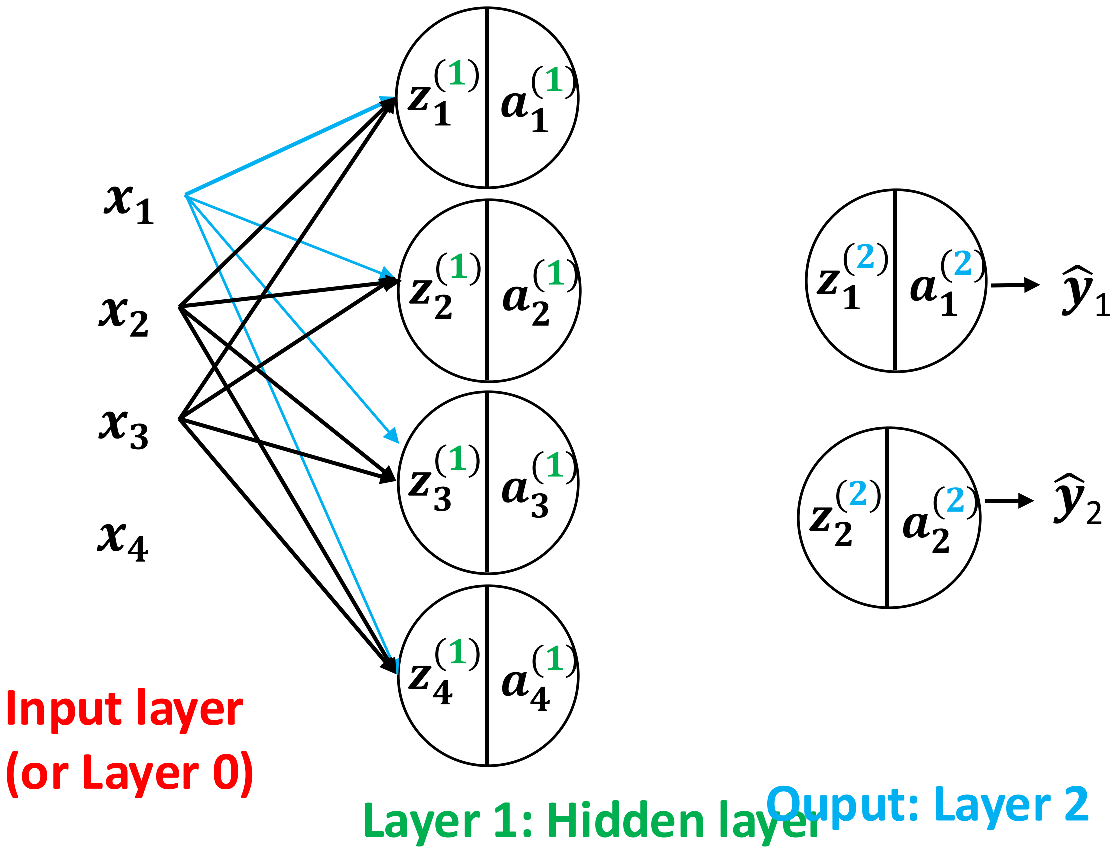
Each layer of a neural network is connected to the next layer.

Architecture of a Neural Network



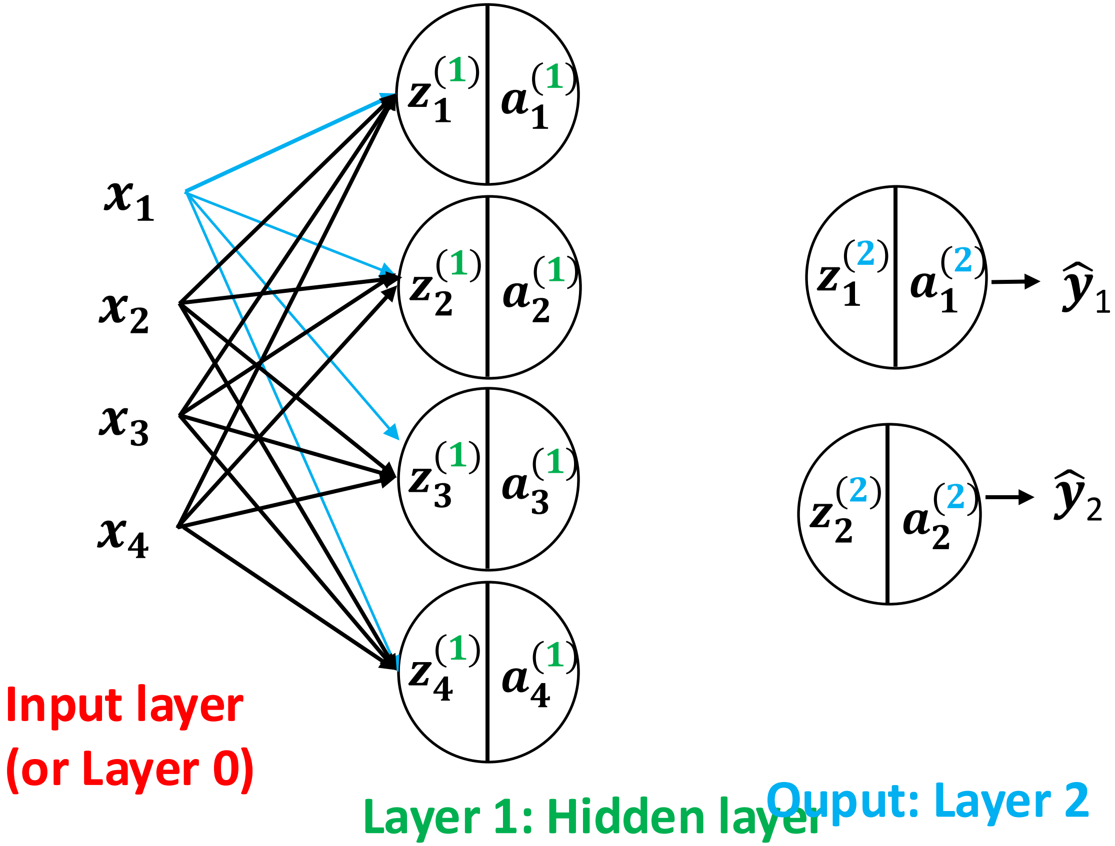
Each layer of a neural network is connected to the next layer.

Architecture of a Neural Network



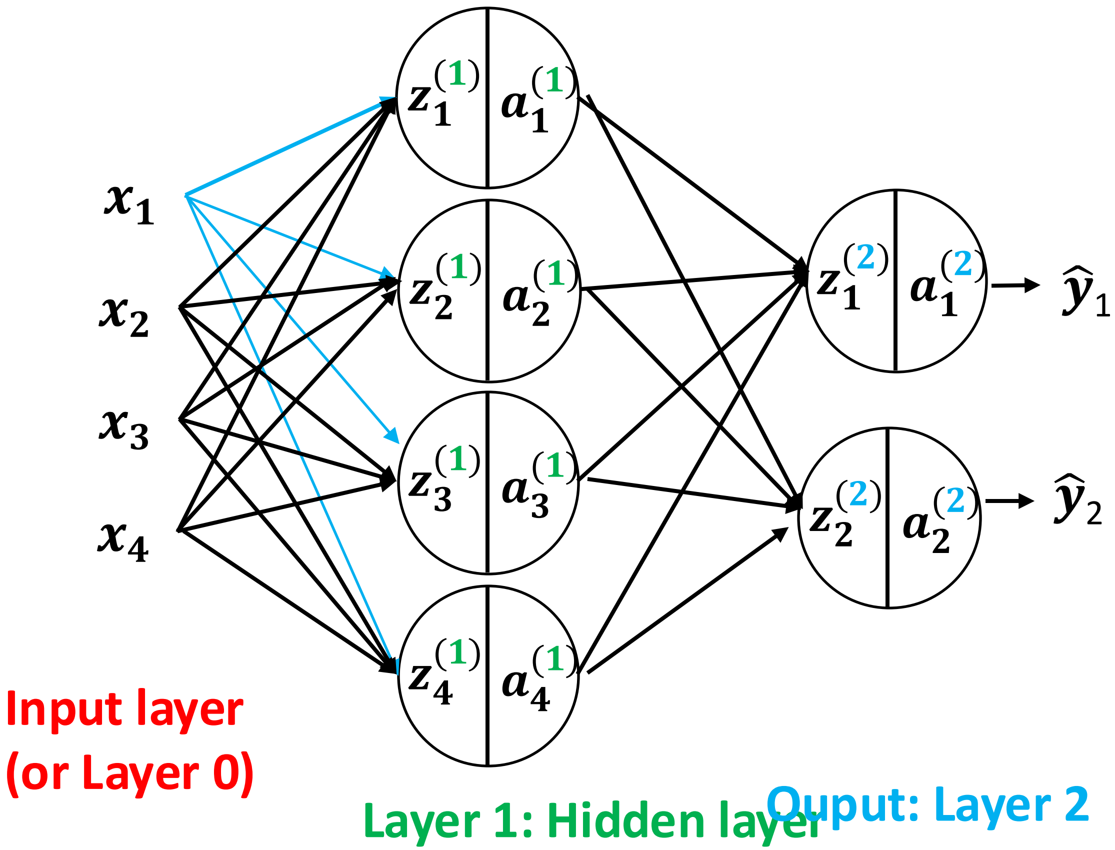
Each layer of a neural network is connected to the next layer.

Architecture of a Neural Network



Each layer of a neural network is connected to the next layer.

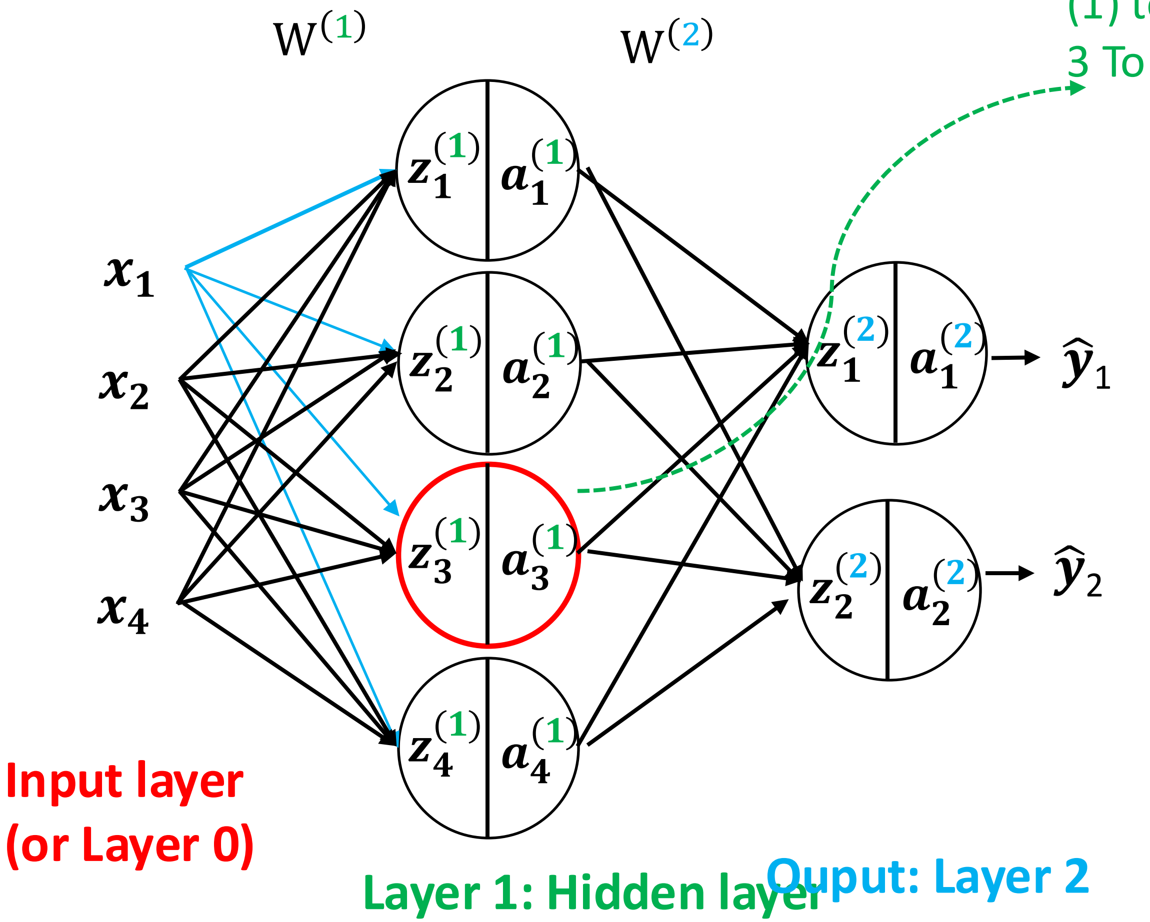
Architecture of a Neural Network



The weights between the neurons in different layers represent the strength of the connection between them.

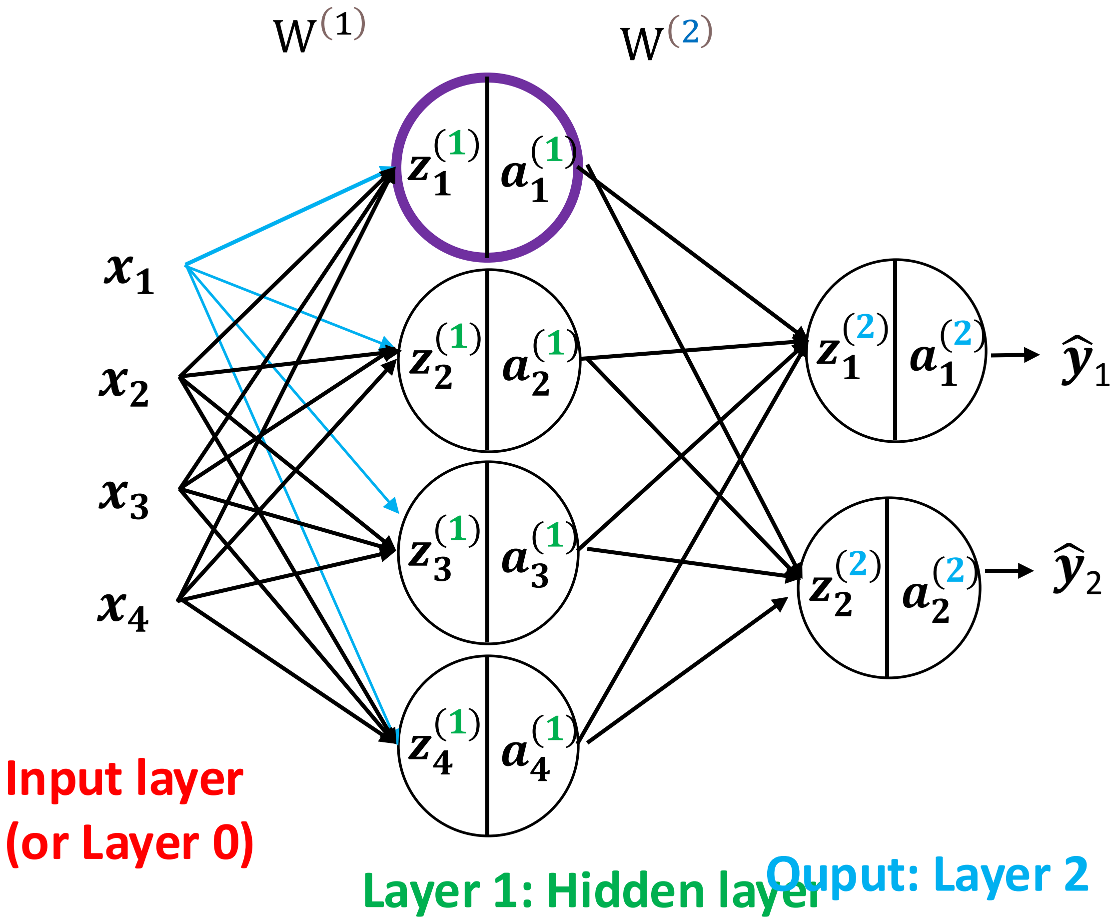
Architecture of a Neural Network

(1) to indicate that this activation, a, is in layer 1.
3 To indicate node No. 3 in hidden layer



$W^{(1)}$ Represents the weights between the input layer and the first hidden layer.
$W^{(2)}$ The weights between the first hidden layer and the output layer.

Architecture of a Neural Network

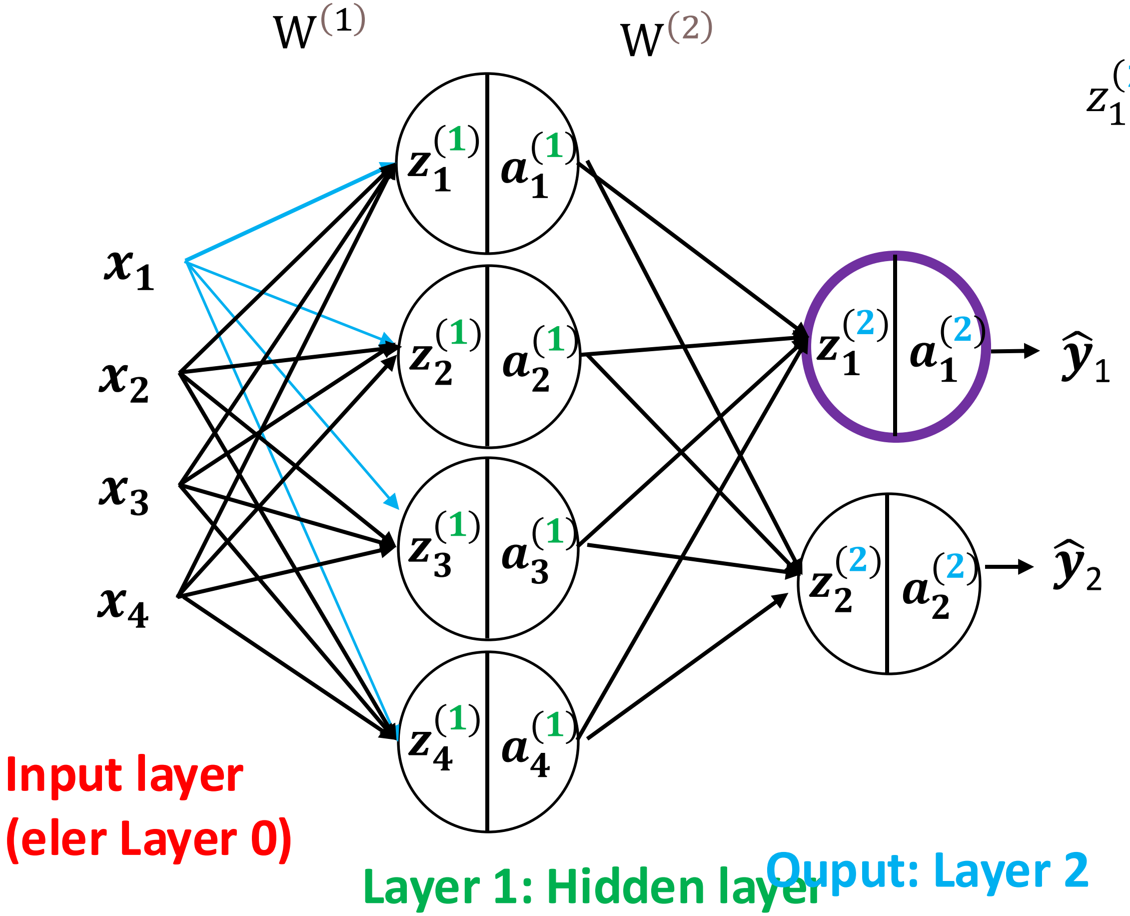


$$z_1^{(1)} = w_{1,1}^{(1)} \cdot x_1 + w_{2,1}^{(1)} \cdot x_2 + w_{3,1}^{(1)} \cdot x_3 + w_{4,1}^{(1)} \cdot x_4 + b_1^{(1)}$$

$$a_1^{(1)} = f\left(z_1^{(1)}\right)$$

f( ) is an activation function
For example, sigmoid

Architecture of a Neural Network

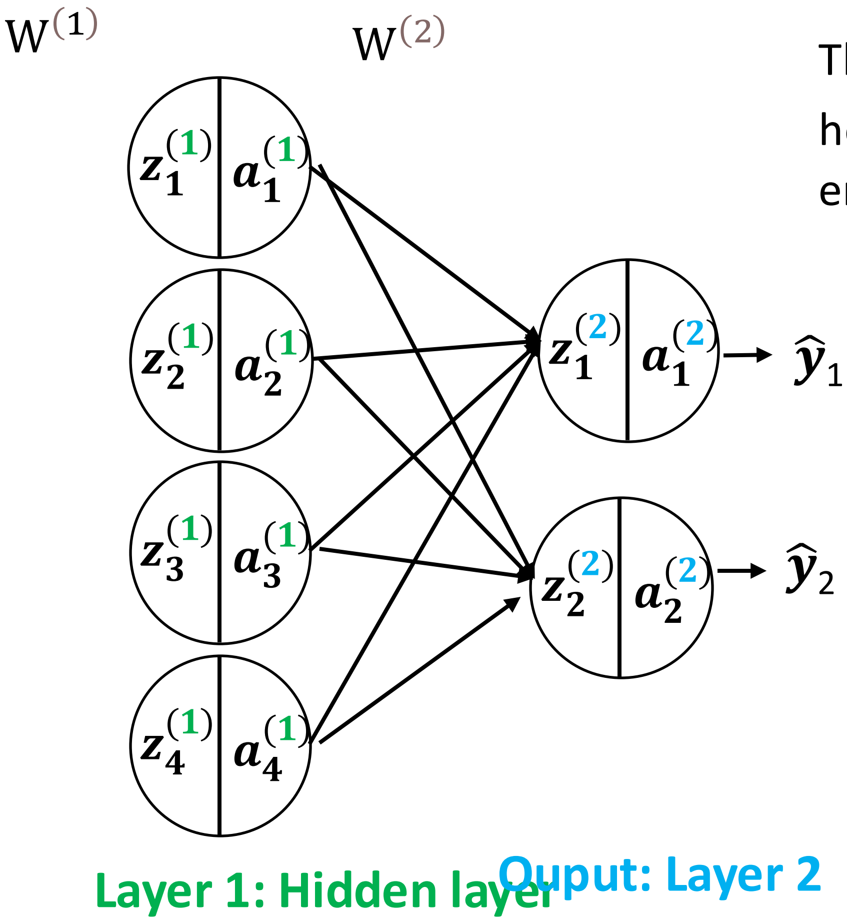


$$z_1^{(2)} = w_{1,1}^{(2)} \cdot a_1^{(1)} + w_{2,1}^{(2)} \cdot a_2^{(1)} + w_{3,1}^{(2)} \cdot a_3^{(1)} + w_{4,1}^{(2)} \cdot a_4^{(1)} + b_1^{(2)}$$

$$\hat{y}_1 = a_1^{(2)} = f\left(z_1^{(2)}\right)$$

$z_1^{(2)}$ Represents the weighted sum of activations from the first hidden layer (layer 1) to the first output node in the output layer.
(layer 2).

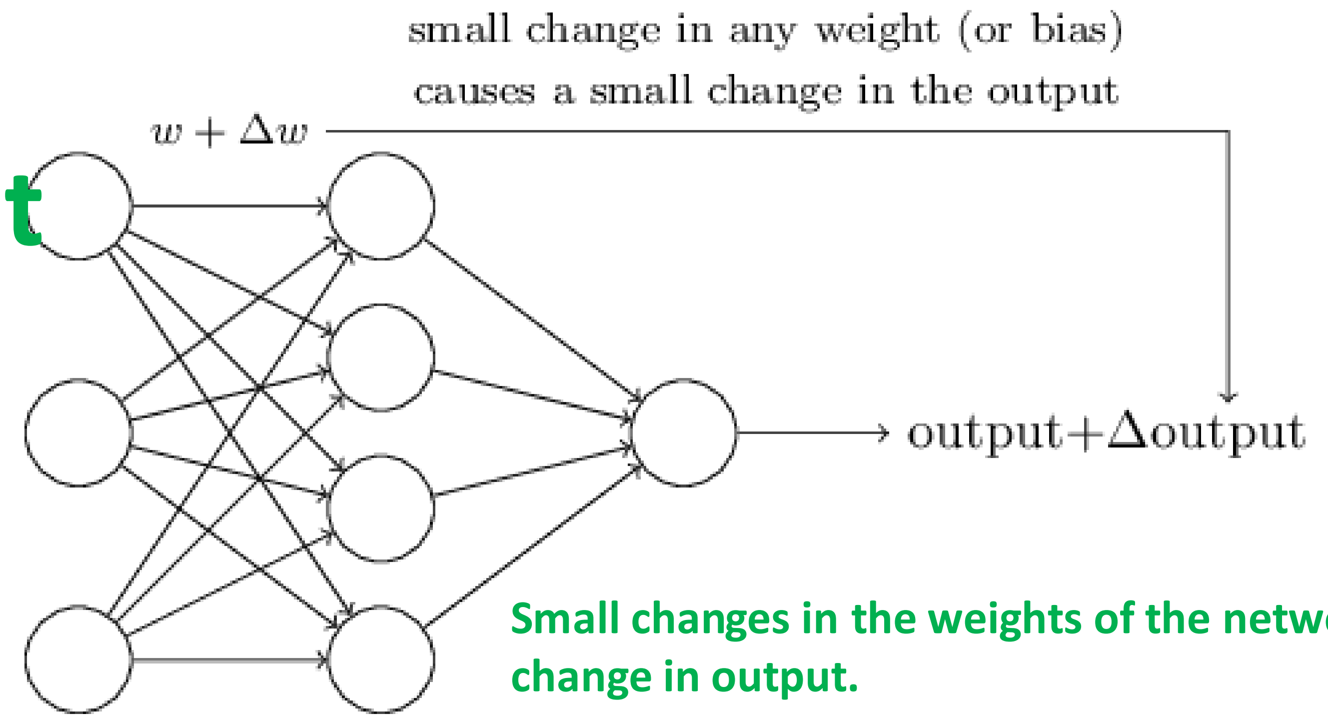
Architecture of a Neural Network



This looks like logistic regression, but with 'features' that we hopefully will learn (i.e., $(a_1^{(1)}, a_2^{(1)}, a_3^{(1)}, a_4^{(1)})$) and NOT engineered by us (i.e., (x_1, x_2, x_3, x_4)).

Disadvantage No. 1 of perceptron with binary activation : Difficult to train

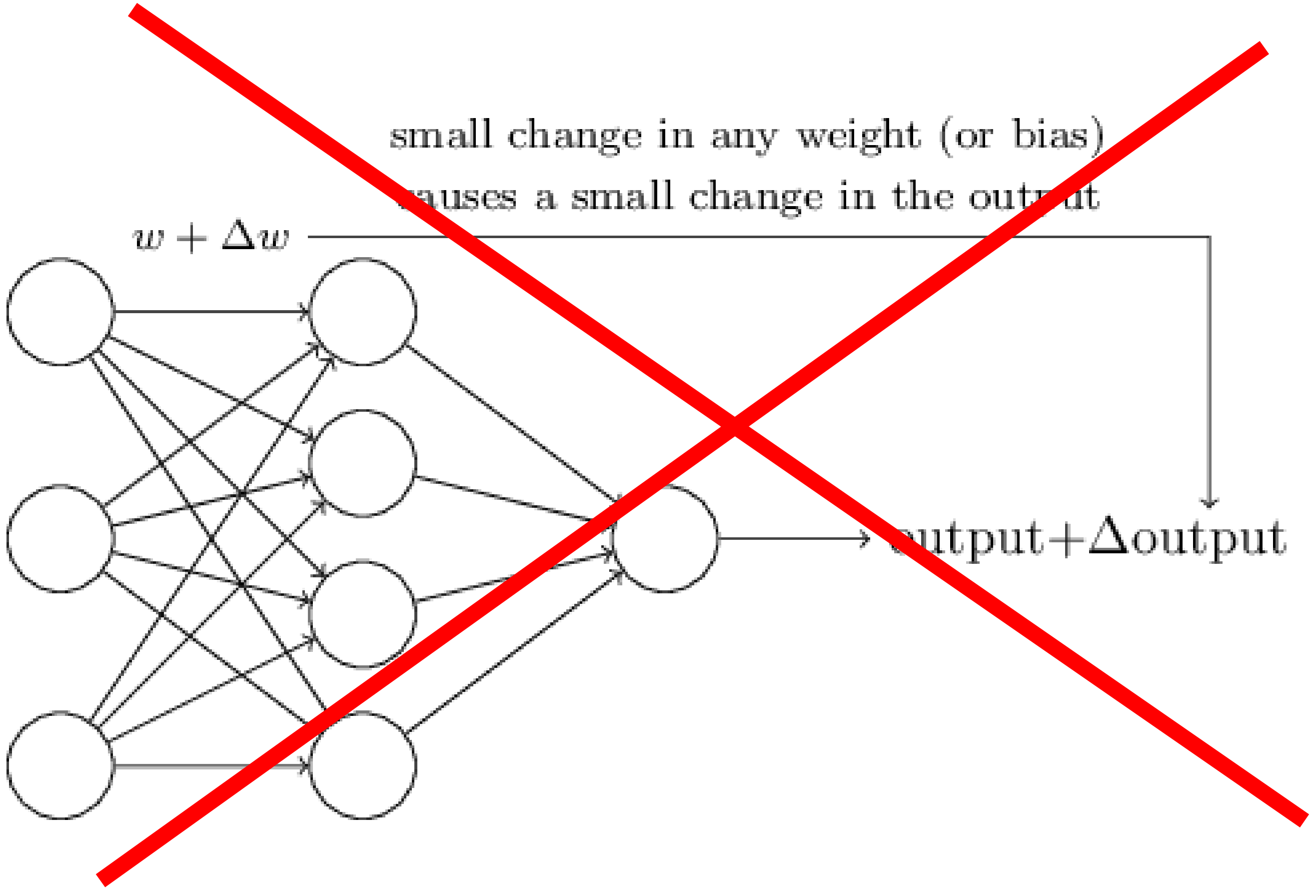
What we want



Small changes in the weights of the network=> lead to little change in output.

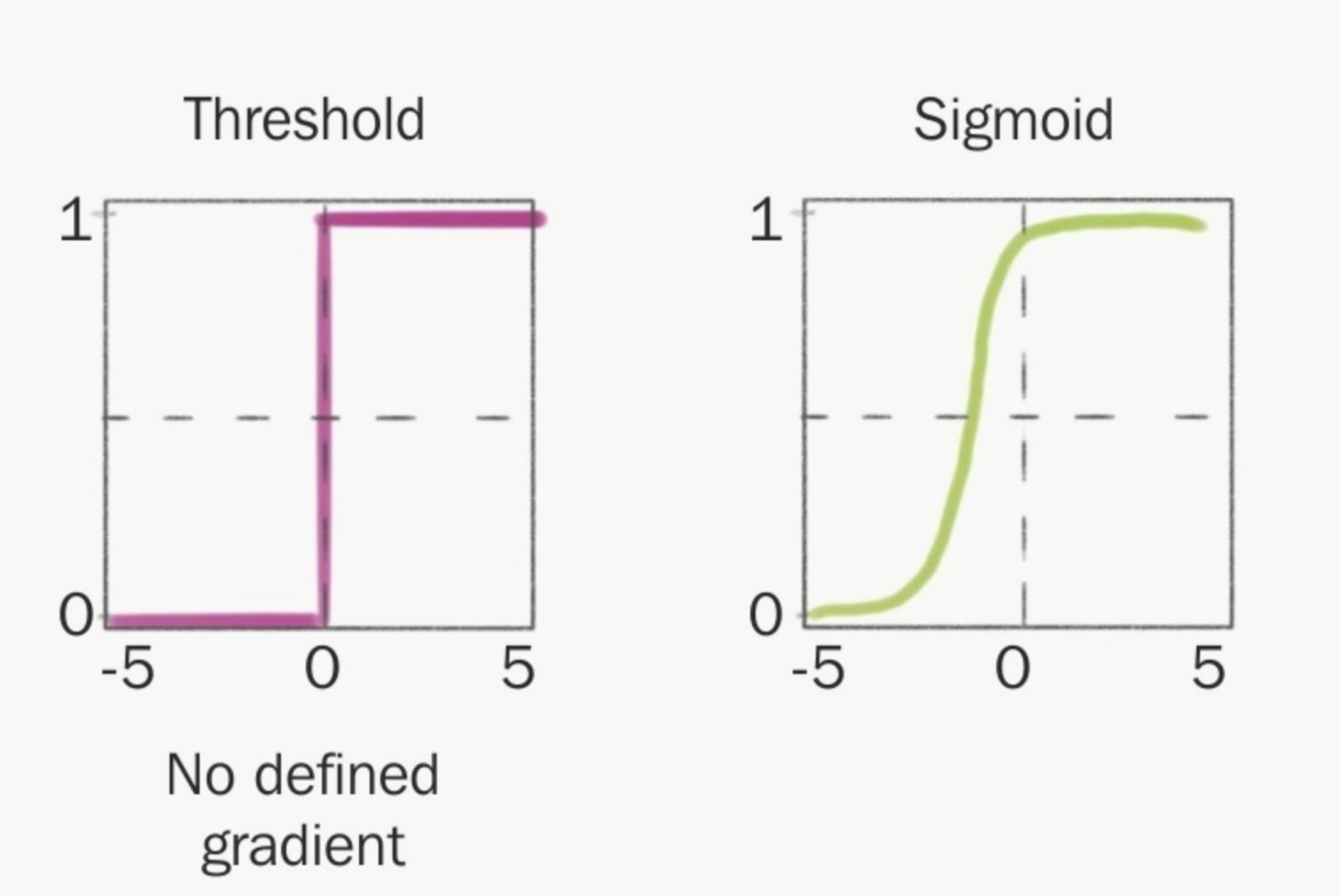
Disadvantage No. 1 of perceptron with binary activation : Difficult to train

What we have



Small changes in the weights of the network=> lead to a big change in output !!

Sigmoid: is a smooth feature



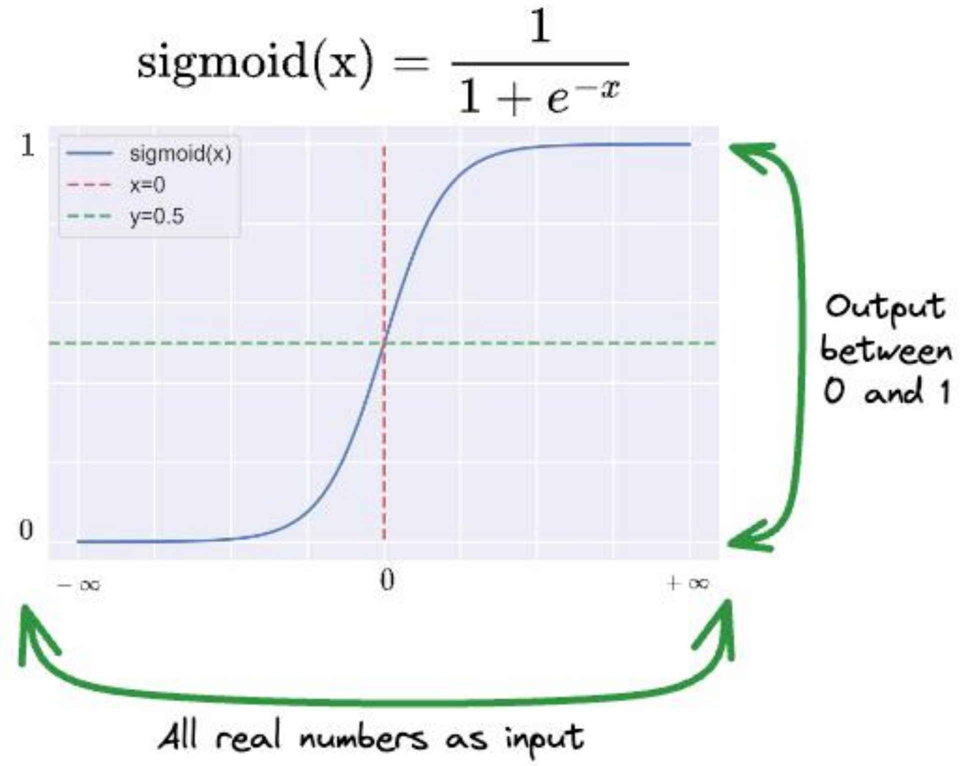
The sigmoid function is differentiable, which makes it possible to see the effect of changes in output

https://files.codingninjas.in/article_images/activation-functions-0-1651309244.webp

Sigmoid

- Intuitive for classification: Can be "translated" into a probability of belonging to a class.

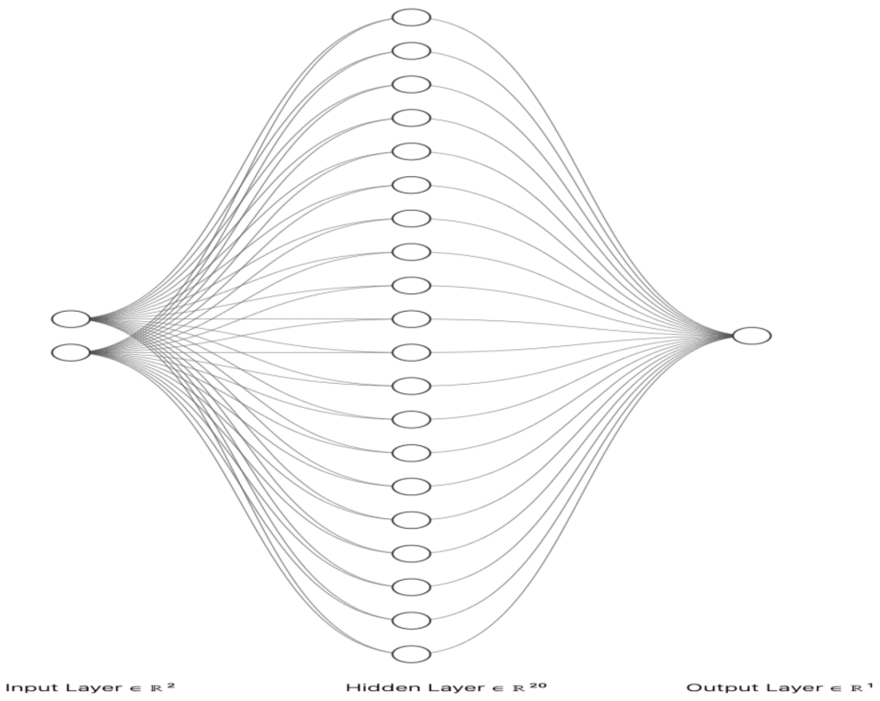
- **Regression: For tasks such as predicting continuous values (e.g., house prices), a binary output is not suitable.**



Disadvantage 2: Composition of Linear Classifiers is still linear

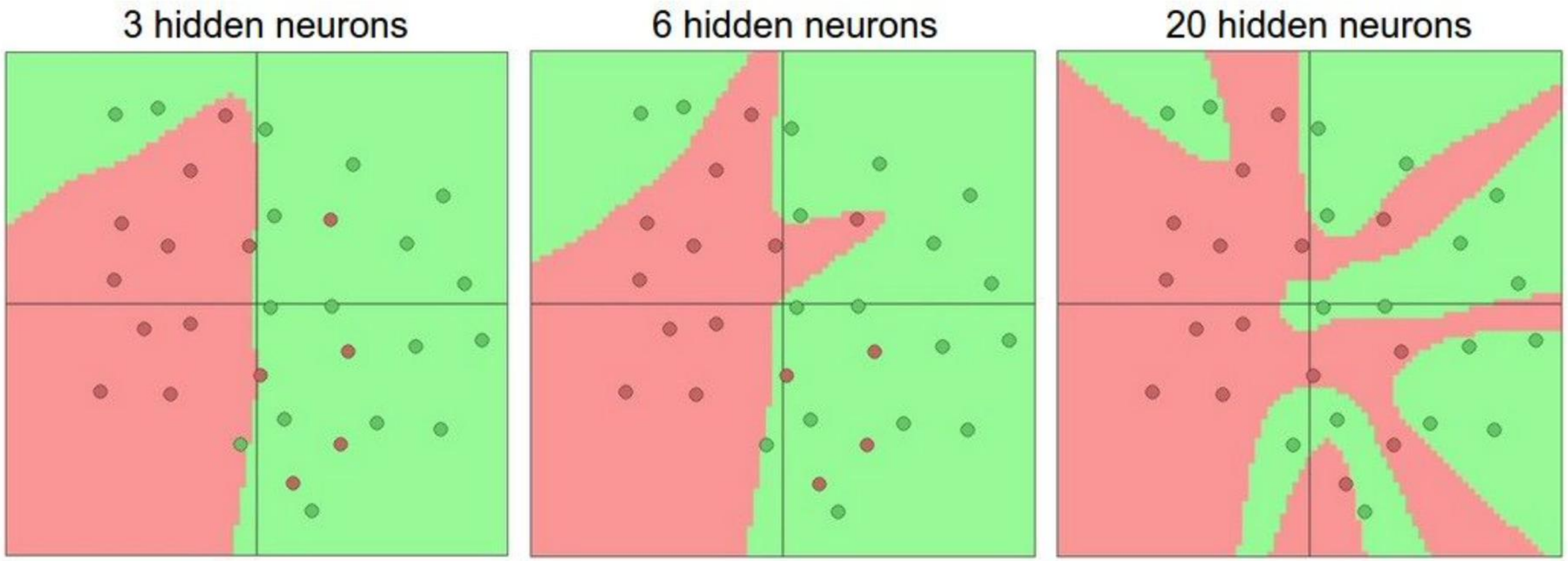
- **Composition of Linear Classifiers :**
  - Although stacking linear Classifiers can create a more complex decision-making function,
  - the decision limit will still be linear with respect to the input functions.
- It cannot find a complex decision-making boundary that is not linear

Universal approximation theorem



Theorem: "We can approximate any continuous function by using a neural network with one hidden layer, a nonlinear activation function, and an infinite number of nodes"

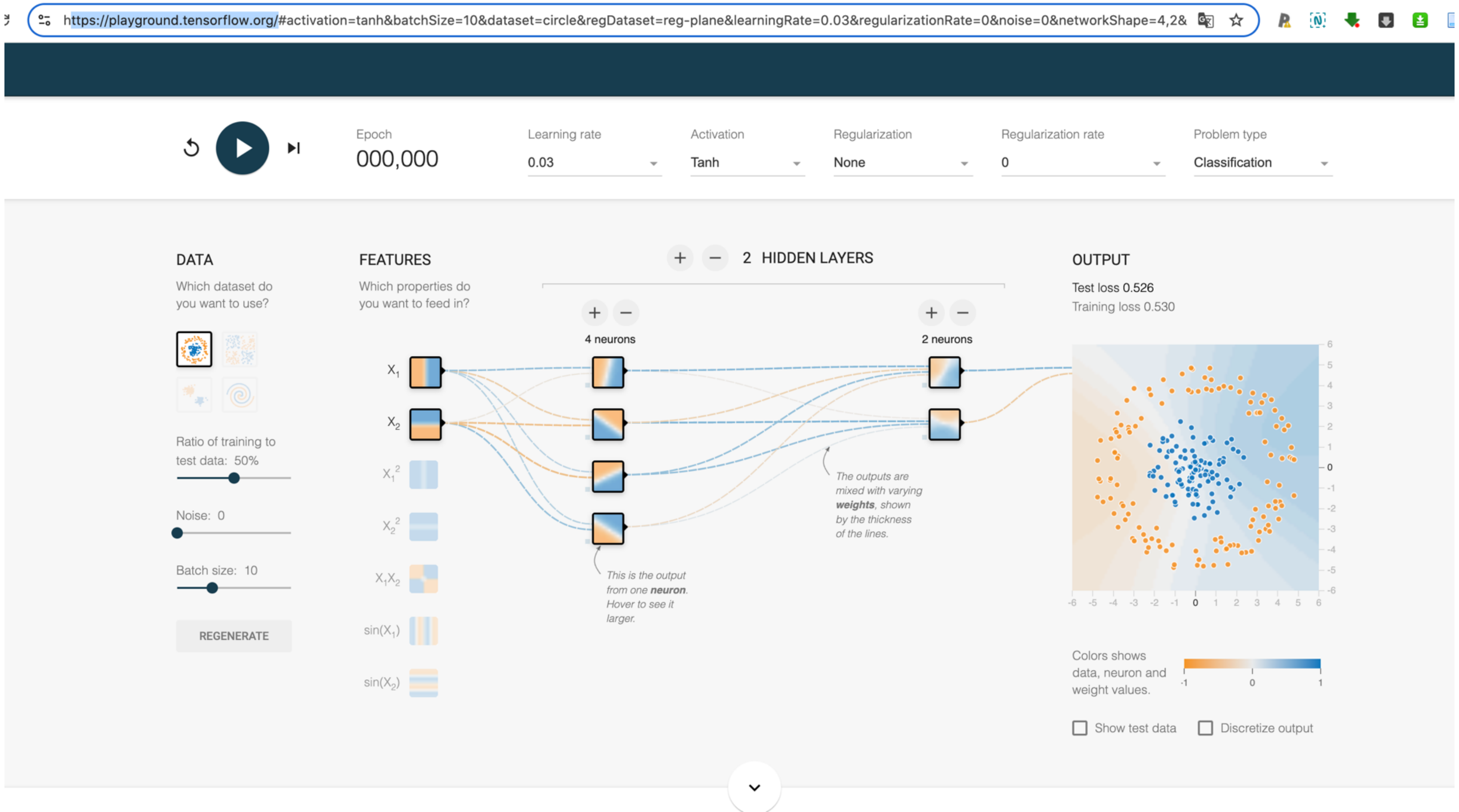
Universal approximation theorem: Decision limits for neural networks



https://storage1.ucsd.edu/slides/CSE152/L6_NeuralNetwork.html#/--DecisionBoundariesofNeuralNetworks_2

Demo: Testing the effect of the number of nodes in the hidden layer

- https://playground.tensorflow.org/



Loss function for regression problems: Mean Squared Error (MSE)

- $L = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \widehat{y_i} \right)^2$

L is avg mean square error(MSE).

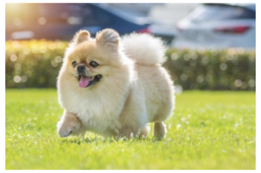
n is number of examples in the data

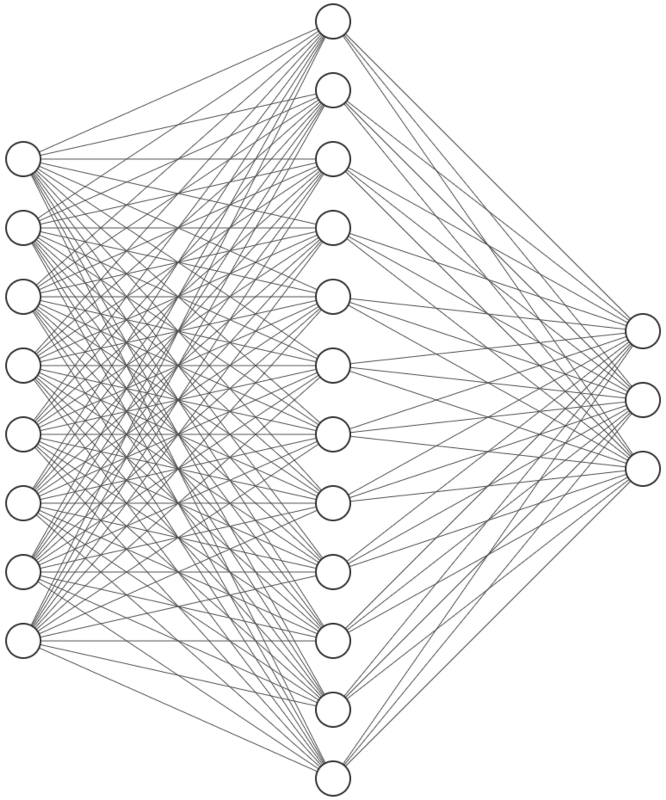
y_i is the ground truth for the **i –th** example.

$\widehat{y_i}$ is the predicted value for the **i –th example**.

$(y_i - \widehat{y_i})^2$ is the squared difference between the true value and the predicted value.

The loss of multiclass classification: cross-entropy loss





$\hat{y_1} = 2$

$\hat{y_2} = 1$

$\hat{y_3} = 0.1$

Probability   Label

Softmax	Sannsynlighet	Label	
$p_j = \frac{e^{\hat{y_j}}}{\sum_{k=1}^{C} e^{\hat{y_k}}}$	p_1=0.7	$y_1 = 1$	Hund
	$p_2 = 0.24$	$y_2 = 0$	Katt
	$p_3 = 0.01$	$y_3 = 0$	Hest

$$\text{kryssentropi} = -\sum_{j=1}^{C} y_j \log(p_j)$$

cross entropy

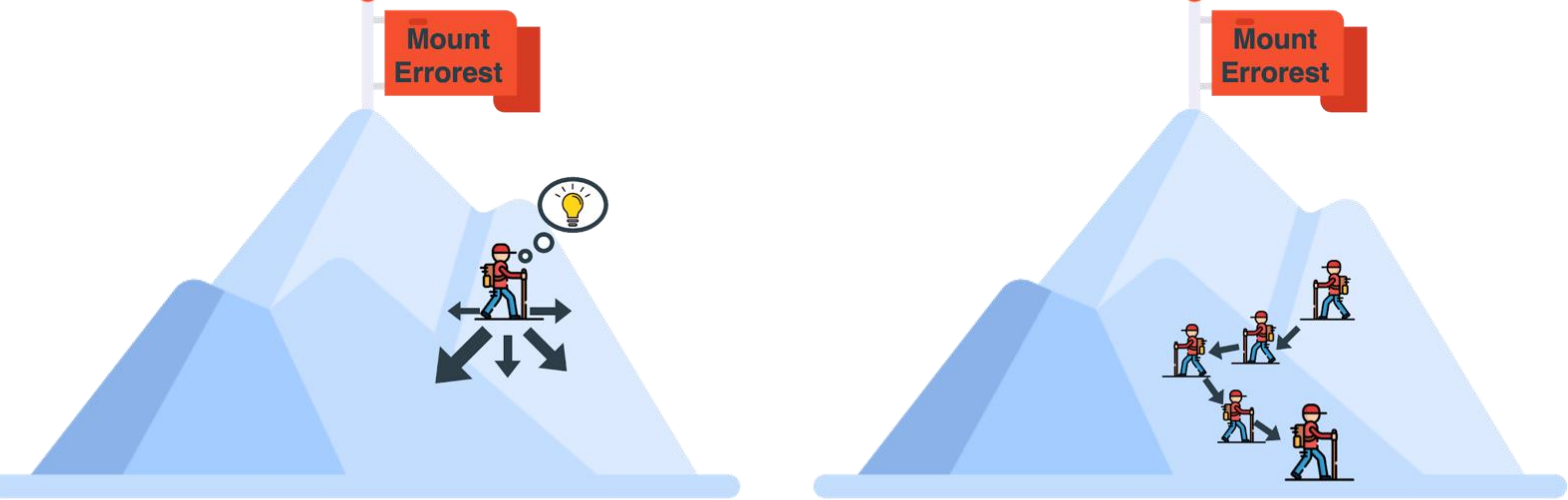
Cross-entropy loss for an example

Cross-entropy loss for multiple samples (entire dataset)

- $\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_j^{(i)} \log\left(p_j^{(i)}\right)$

- N er number of examples in the dataset,
- $y_j^{(i)}$ $t$he true label for the i-th sample for class j.
- $p_j^{(i)}$ is prob for example “i“ to belong to class “j” using softmax

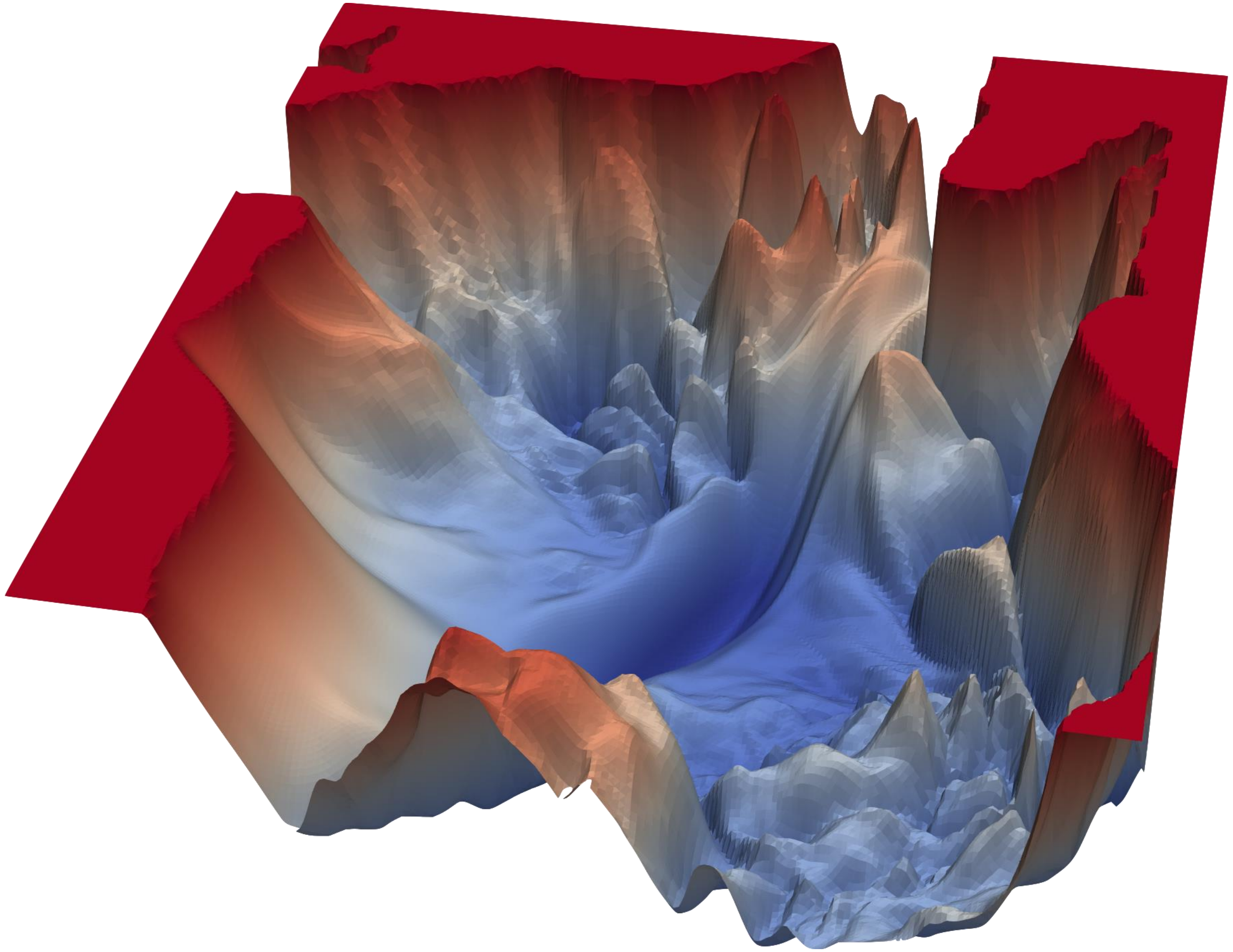
How can we reduce the loss function in NN? The answer: backpropagation



- Imagine you're standing on top of a mountain on a foggy day, and your goal is to get to the bottom.
- But because of the fog, you can only see a little around you, so you can't see all the way down.
- To get down, you start walking in the direction where the hill slopes the most downwards.
- You take small steps in that direction, constantly checking where it is steeper downhill, and adjust your course accordingly.
- As you take more steps, you'll get closer to the bottom, even if you can't see the entire route right away.

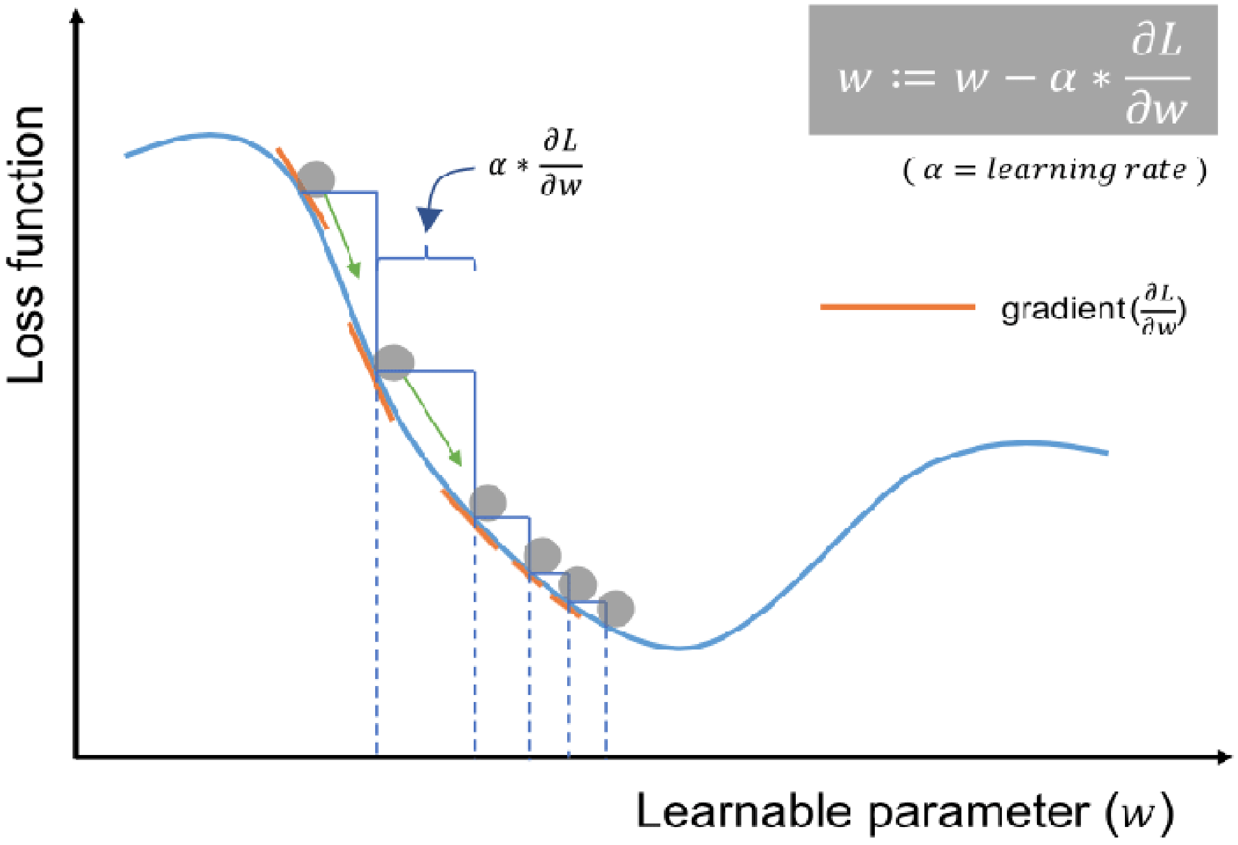
https://livebook.manning.com/book/grokking-machine-learning/appendix-b/v-14/13

Example of a true loss function



Li, Hao, et al. "Visualizing the loss landscape of neural nets." Advances in neural information processing systems 31 (2018).

Gradient descent



Gradient: Derived from Loss with respect to weight

Backpropagation with an example

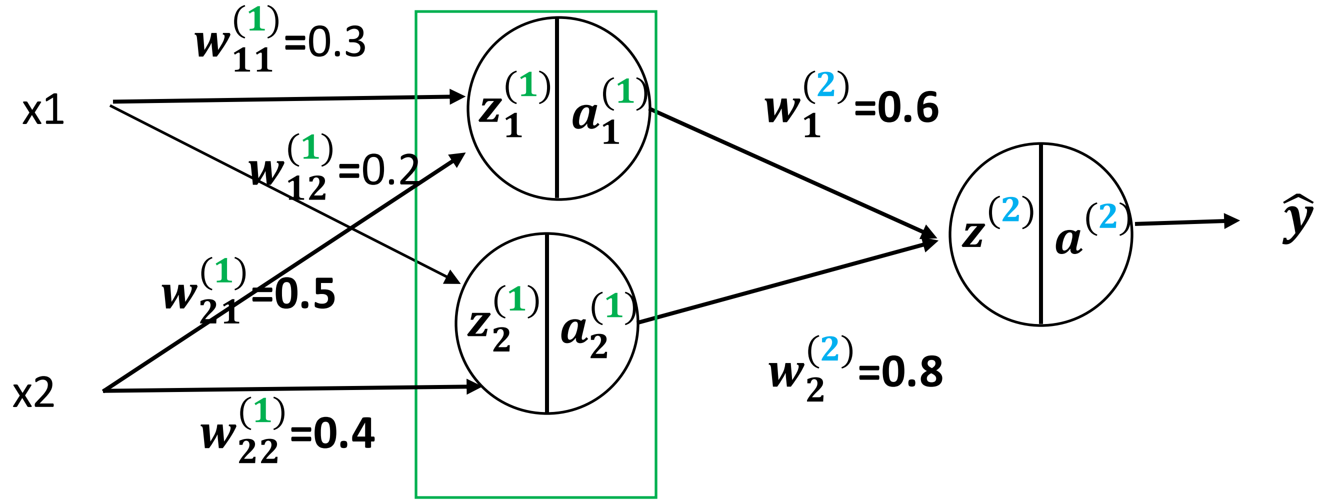
Out temperature(X1)	Scaled Temp. (X1_scaled)	Number of people (X2)	Scaled number. (X2_scaled)	Energy usage (y)	Scaled Energy Usage (y_scaled)
10.0	0.4	50.0	0.5	2700.0	0.27
5.0	0.3	30.0	0.3	1700.0	0.17
20.0	0.6	70.0	0.7	3700.0	0.37
-5.0	0.1	20.0	0.2	1200.0	0.12
30.0	0.8	90.0	0.9	4700.0	0.47

- You are responsible for analysing the energy consumption of a building to ensure that it becomes more energy-efficient and sustainable.
- The goal is to predict the daily energy consumption in the building based on two important factors:

1.Outside temperature (X1) - in degrees Celsius.

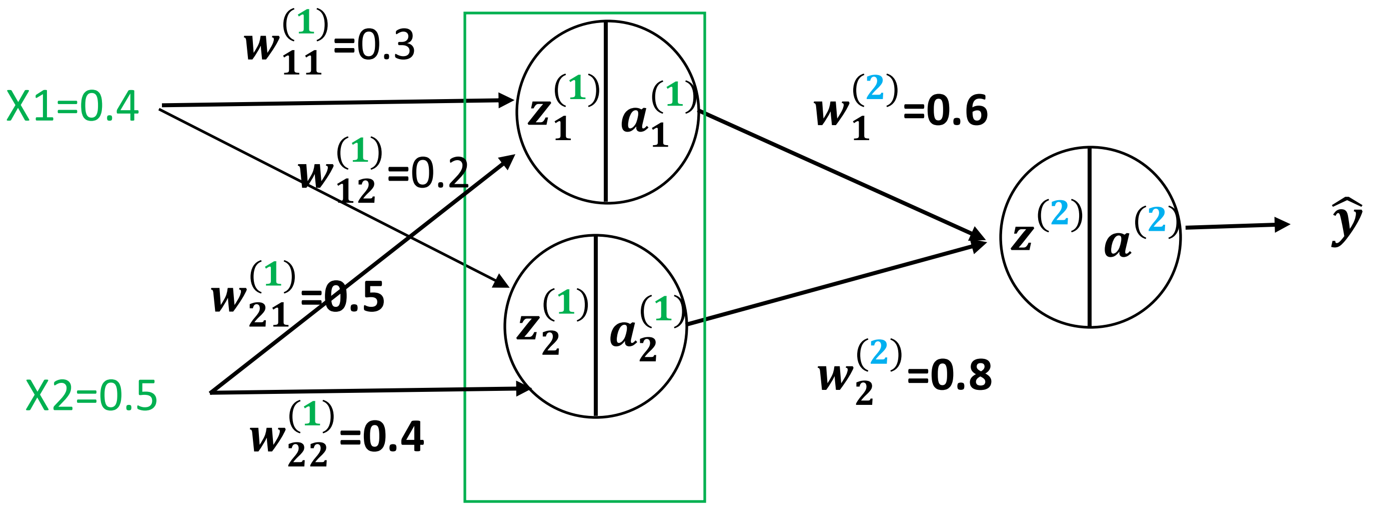
2.Number of people in the building (X2) - the number of people in the building per day.

Initialization of scales



Forward propagation from input: weighted sum

- We will use the following input variables:
  - X1 = Scaled Outdoor Temperature = 0.40 (i.e. 10°C)
  - X2 = Scaled number of people = 0.50 (i.e. 50 people)

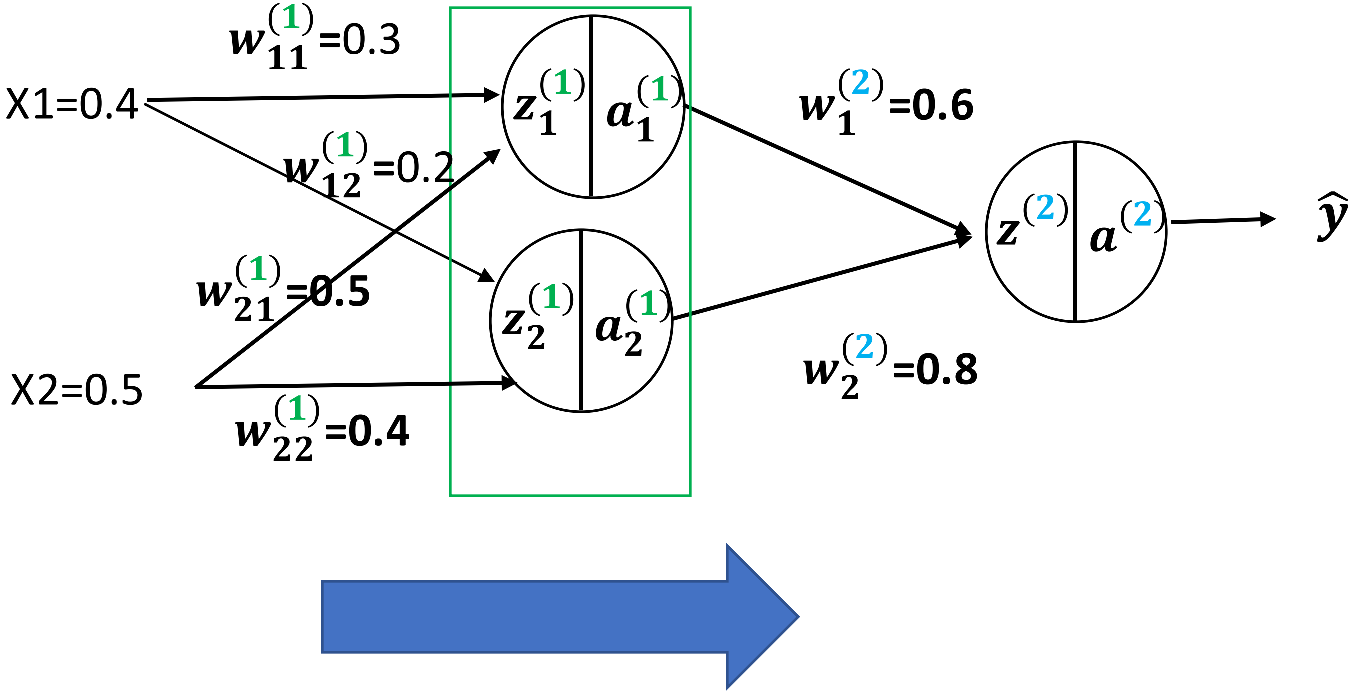


$z_1^{(1)}$=(0.3×0.40)+(0.5×0.50)=0.37

$z_2^{(1)}$=(0.2×0.40)+(0.4×0.50)=0.28

Forward propagation from input: activation

- We will use the following input variables:
  - X1 = Scaled Outdoor Temperature = 0.40 (i.e. 10°C)
  - X2 = Scaled number of people = 0.50 (i.e. 50 people)

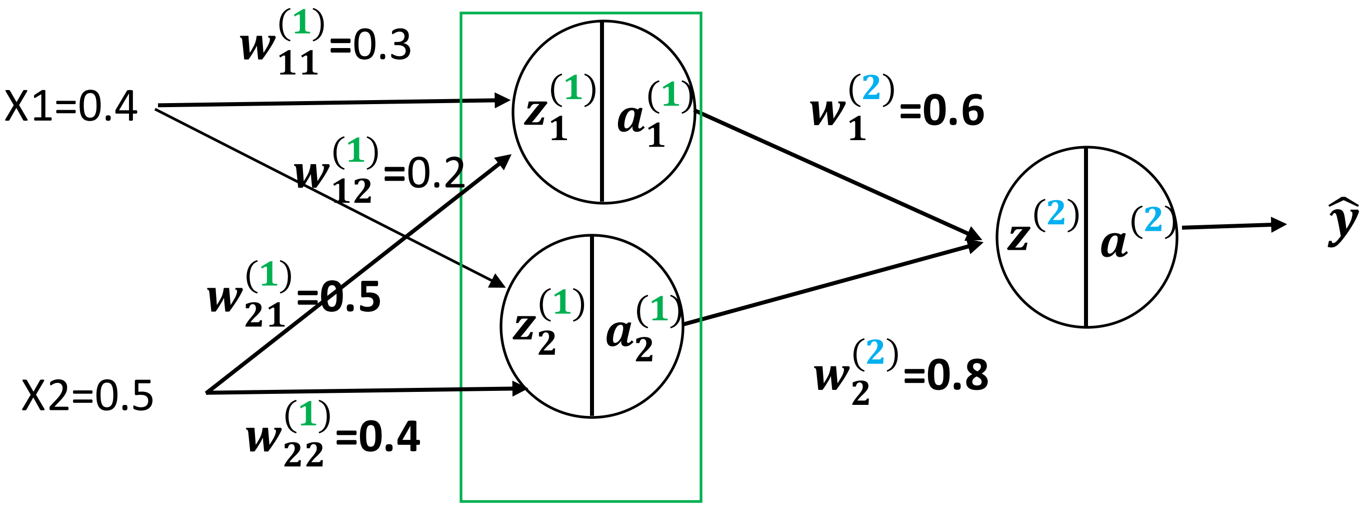


$$a_1^{(1)} = \left(z_1^{(1)}\right) = \frac{1}{1+e^{-0.37}} \approx 0.5915$$

$$a_2^{(1)} = \sigma\left(z_2^{(1)}\right) = \frac{1}{1+e^{-0.28}} \approx 0.5695$$

Forward propagation from input: weighted sum

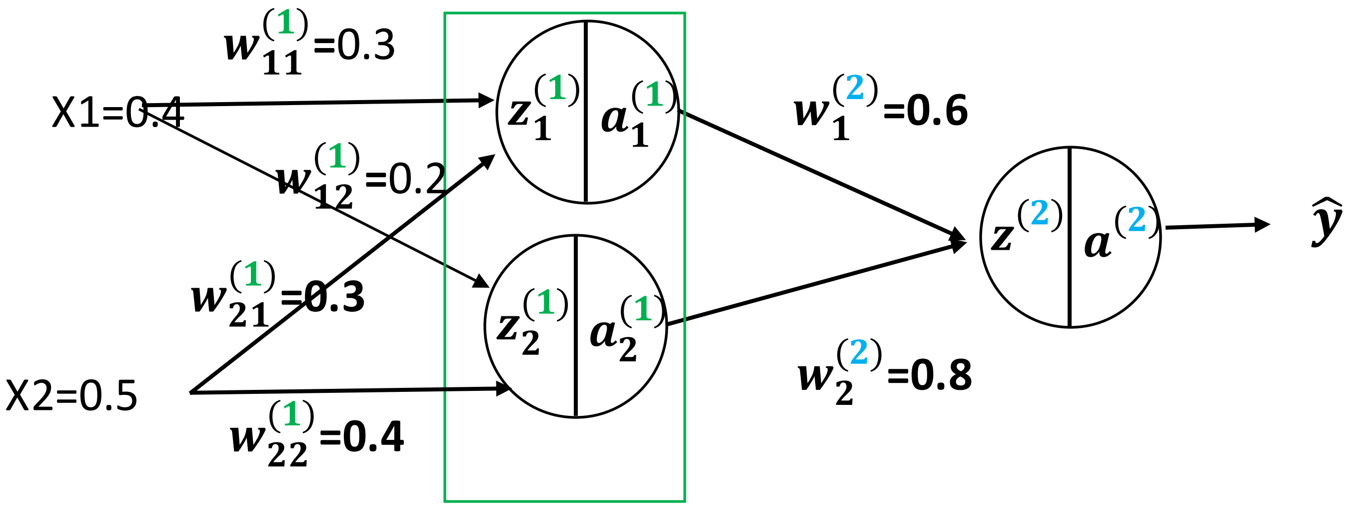
- We will use the following input variables:
  - X1 = Scaled Outdoor Temperature = 0.40 (i.e. 10°C)
  - X2 = Scaled number of people = 0.50 (i.e. 50 people)



$z^{(2)}$= (0.6×0.5915)+(0.8×0.5695)= 0.8105

Forward propagation from input: activation

- We will use the following input variables:
  - X1 = Scaled Outdoor Temperature = 0.40 (i.e. 10°C)
  - X2 = Scaled number of people = 0.50 (i.e. 50 people)



$$\hat{y} = a^{(2)} = \sigma(z^{(2)}) = \frac{1}{1+e^{-0.8105}} \approx 0.6922$$

File

Out temperature(X1)	Scaled Temp. (X1_scaled)	Number of people (X2)	Scaled number. (X2_scaled)	Energy usage (y)	Scaled Energy Usage (y_scaled)
10.0	0.4	50.0	**0.5**	2700.0	0.27
5.0	0.3	30.0	0.3	1700.0	0.17
20.0	0.6	70.0	0.7	3700.0	0.37
-5.0	0.1	20.0	0.2	1200.0	0.12
30.0	0.8	90.0	0.9	4700.0	0.47

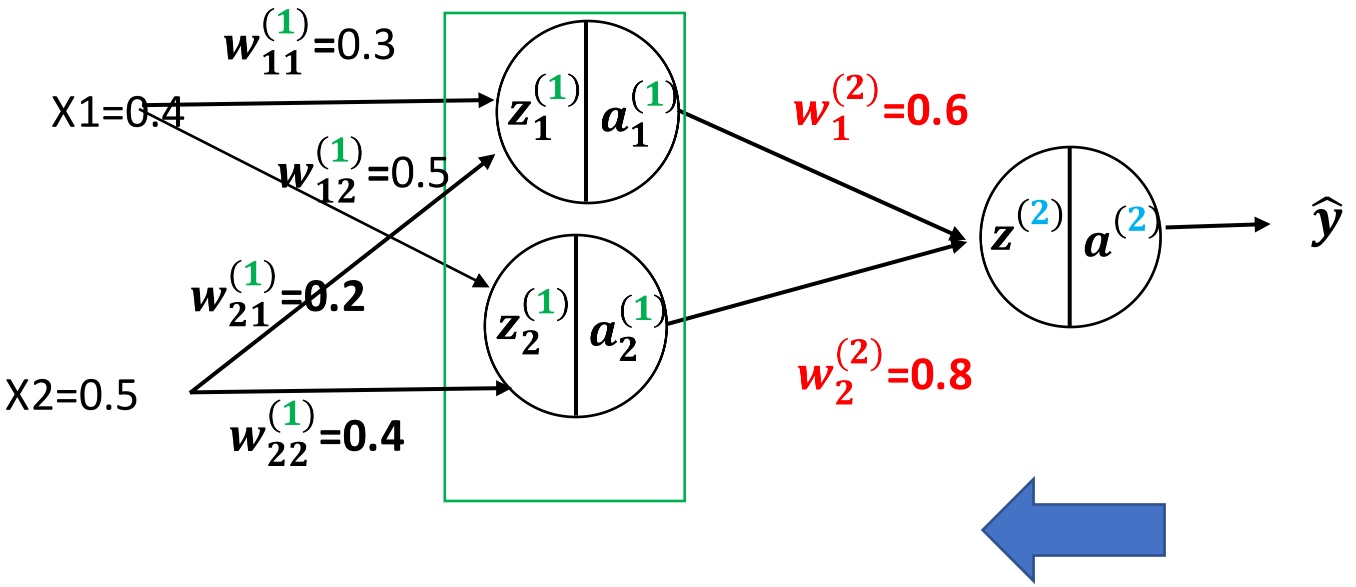
MSE= $\frac{1}{2}(0.27 - 0.6922)^2 = \frac{1}{2}(-0.4222)^2 = \frac{1}{2} \times 0.1783 = 0.08915$

We need to change the weights to reduce the error

We will find a direction for updating the weights so that the "error" is reduced.

How change in weight affects Loss➔Answer: Derivative of Loss with respect to weight: Called backpropgation

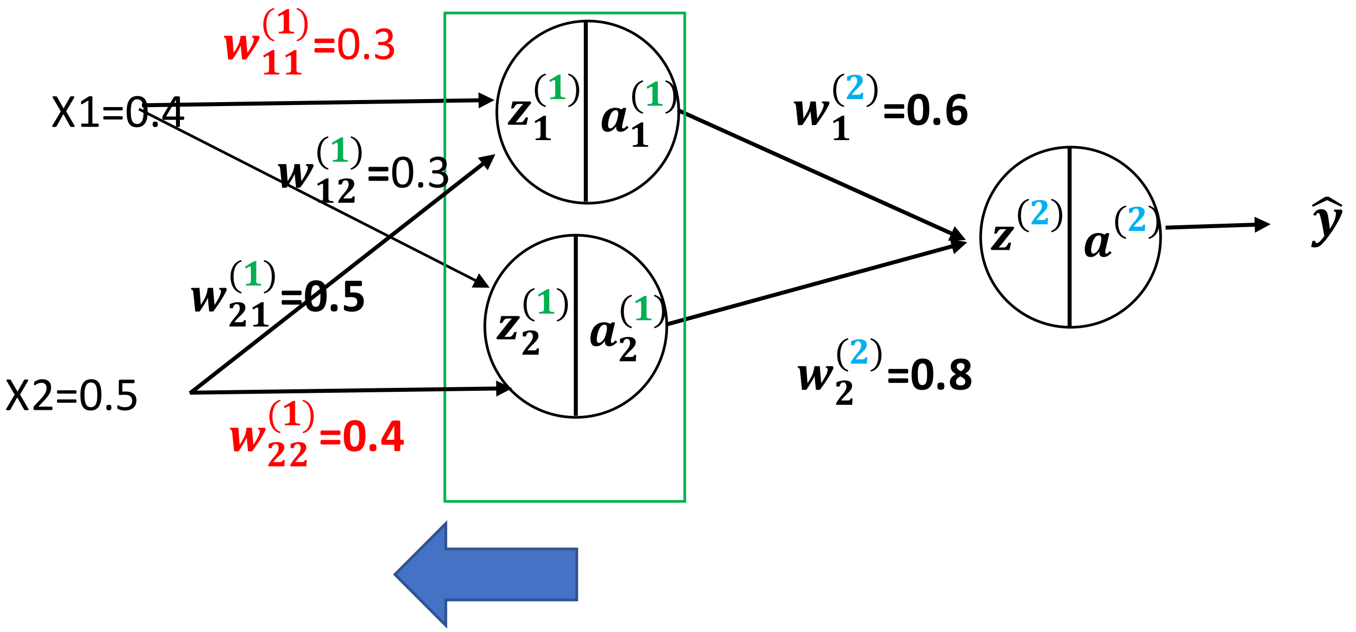
We have to "send" the error backwards from the output to the previous teams



We need to adjust the weights so that the output best mimics the "ground truth" value.

The error' or more precisely, the derivative of the error" (0.27 – 0.6922)

We have to "send" the error backwards from the output to the previous teams



We need to adjust the weights so that the output best mimics the "ground truth" value.

The error' or more precisely, the derivative of the error" (0.27 – 0.6922)

Chain Rule

- To derive a composite function f(g(x))

$$\frac{df(g(x))}{dx} = \frac{df(g(x))}{dg(x)} \cdot \frac{dg(x)}{dx}$$

outer function f with respect to deriving a composite function f(g(x)) to g(x)

- The derivative of the intrinsic function g(x)

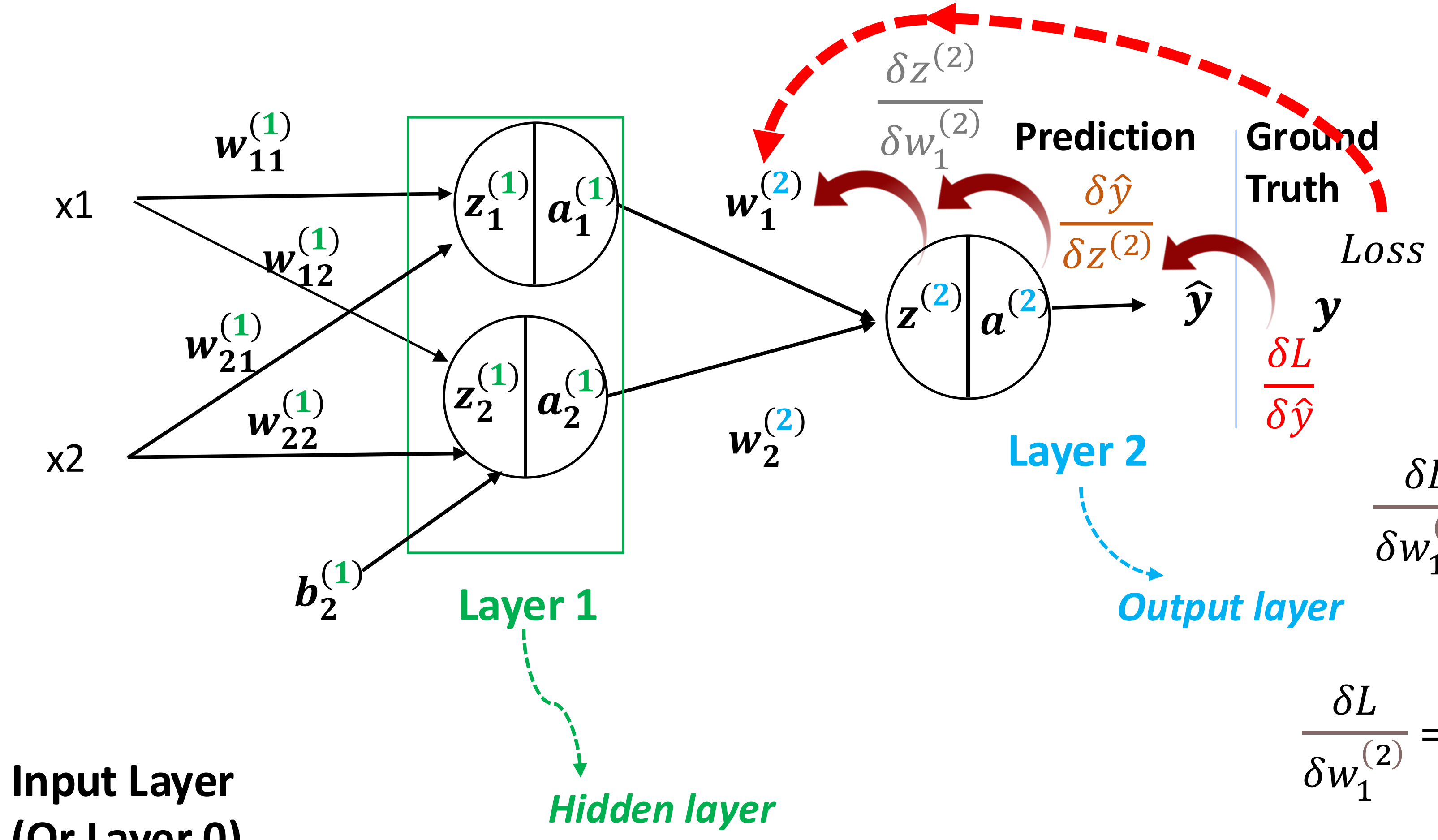
Example of calculating derivatives

$$\frac{\delta L}{\delta . w_1^{(2)}}$$

$$L(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$$

$$\hat{y} = \sigma\left(z^{(2)}\right)$$

$$z^{(2)} = w_1^{(2)} \cdot a_1^{(1)} + w_2^{(2)} a_2^{(1)} + b_1^{(1)}$$



$$\frac{\delta L}{\delta w_1^{(2)}} = \frac{\delta L}{\delta \hat{y}} \frac{\delta \hat{y}}{\delta z^{(2)}} \frac{\delta z^{(2)}}{\delta w_1^{(2)}}$$

$$\frac{\delta L}{\delta w_1^{(2)}} = -(y - \hat{y})\sigma\left(z^{(2)}\right)\left(1 - \sigma\left(z^{(2)}\right)\right) a_1^{(1)}$$

$$\frac{\delta L}{\delta w_1^{(2)}} = -(y - \hat{y})a^{(2)}(1 - a^{(2)})a_1^{(1)}$$

Example of calculating derivatives

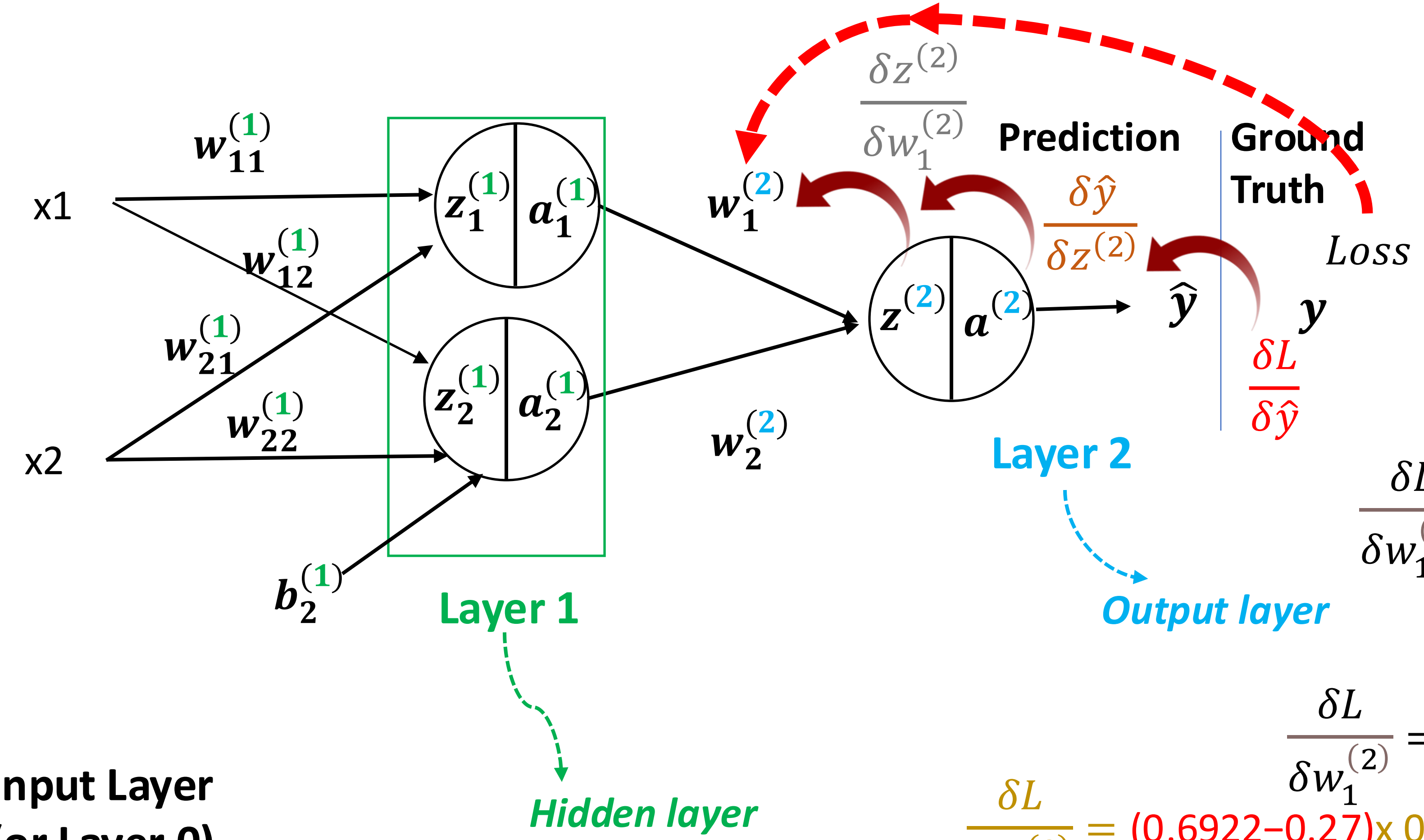
$$\frac{\delta L}{\delta . w_1^{(2)}}$$

$$L(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$$

$$\hat{y} = \sigma\left(z^{(2)}\right)$$

$$z^{(2)} = w_1^{(2)} \cdot a_1^{(1)} + w_2^{(2)} a_2^{(1)} + b_1^{(1)}$$

$$\frac{\delta L}{\delta w_1^{(2)}} = \frac{\delta L}{\delta \hat{y}} \frac{\delta \hat{y}}{\delta z^{(2)}} \frac{\delta z^{(2)}}{\delta w_1^{(2)}}$$



Input Layer
(or Layer 0)

$$\frac{\delta L}{\delta w_1^{(2)}} = -(y - \hat{y})\sigma\left(z^{(2)}\right)\left(1 - \sigma\left(z^{(2)}\right)\right) a_1^{(1)}$$

$$\frac{\delta L}{\delta w_1^{(2)}} = -(y - \hat{y})a^{(2)}(1 - a^{(2)})a_1^{(1)}$$

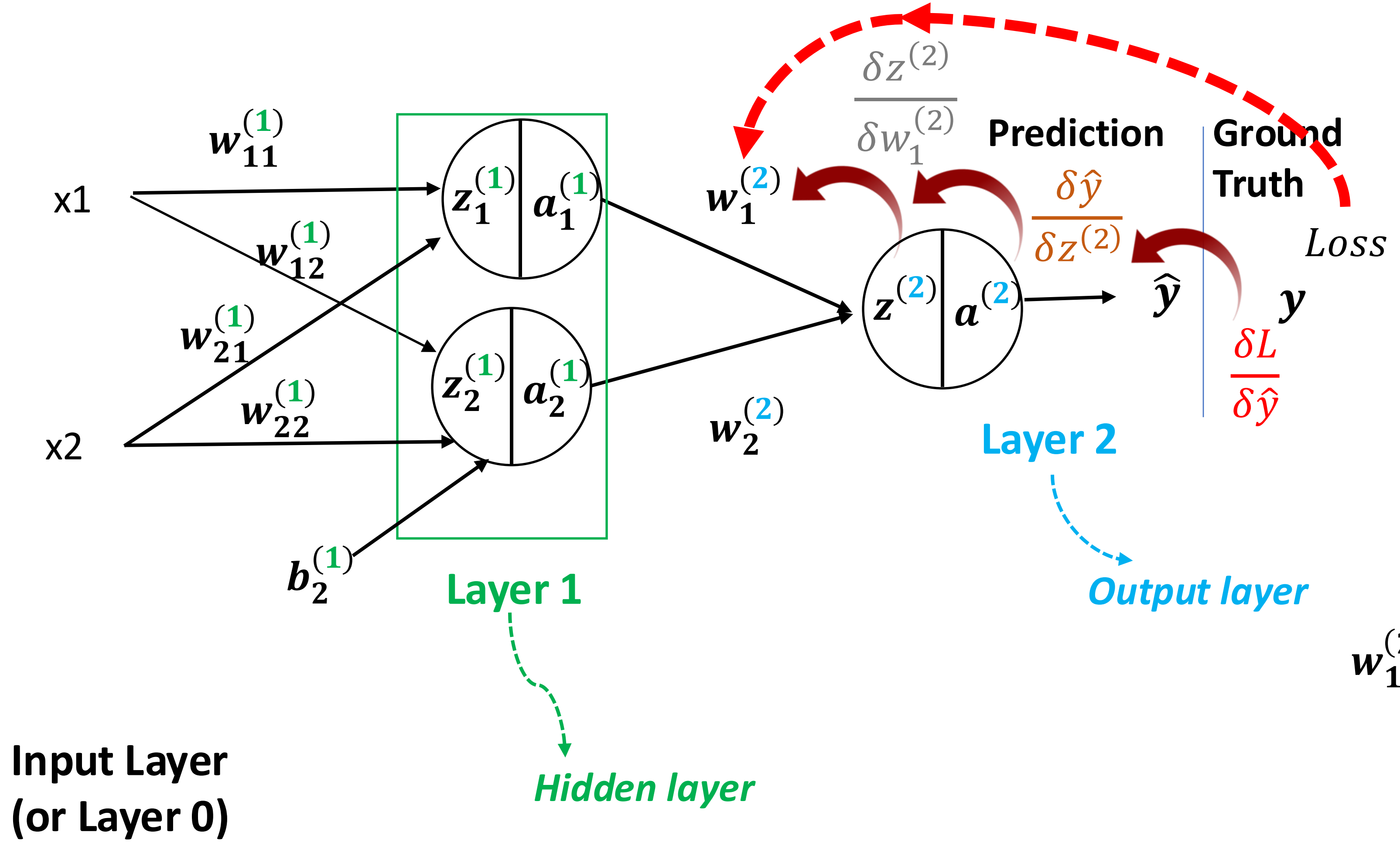
$$\frac{\delta L}{\delta w_1^{(2)}} = (0.6922-0.27) \text{x } 0.6922 \text{x } (1-0.6922) \text{x} 0.5915 = 0.0532$$

backpropagation
: new weight

$$L(y,\hat{y}) = \frac{1}{2}(y-\hat{y})^2$$

$$\hat{y} = \sigma(z^{(2)})$$

$$z^{(2)} = w_1^{(2)} \cdot a_1^{(1)} + w_2^{(2)} a_2^{(1)} + b_1^{(1)}$$

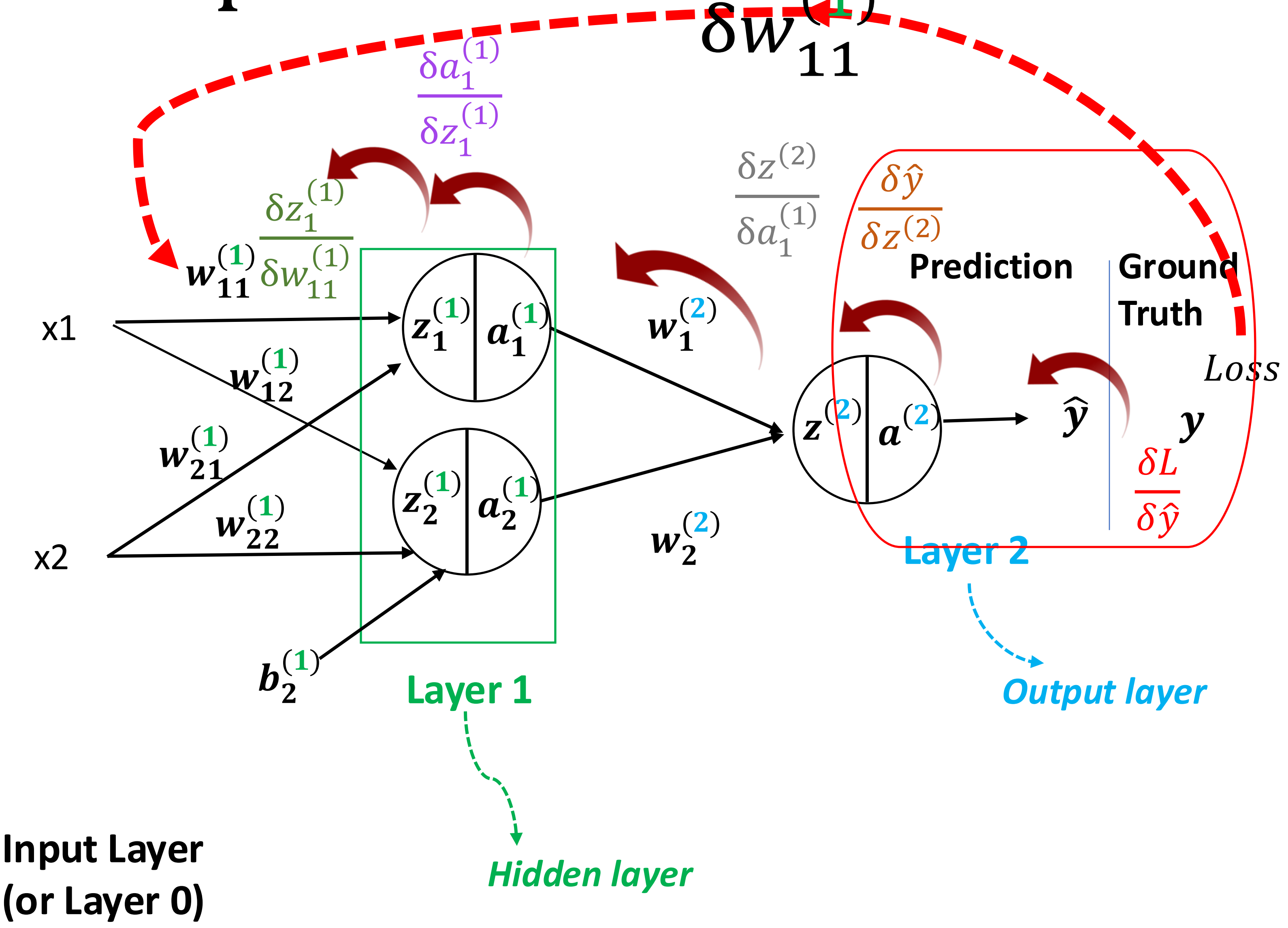


$$\frac{\delta L}{\delta w_1^{(2)}} = \frac{\delta L}{\delta \hat{y}} \frac{\delta \hat{y}}{\delta z^{(2)}} \frac{\delta z^{(2)}}{\delta w_1^{(2)}}$$

$$\boldsymbol{w_{1,new}^{(2)} = w_{1,old}^{(2)} - \eta \frac{\delta L}{\delta w_1^{(2)}}}$$

$$w_{1,new}^{(2)} = 0.6 - 0.01 \text{ x } 0.0532 = 0.599468$$

Computation of $\frac{\delta L}{\delta w_{11}^{(1)}}$



$$L(y, \hat{y}) = (y - \hat{y})^2$$

$$\hat{y} = \sigma(z^{(2)})$$

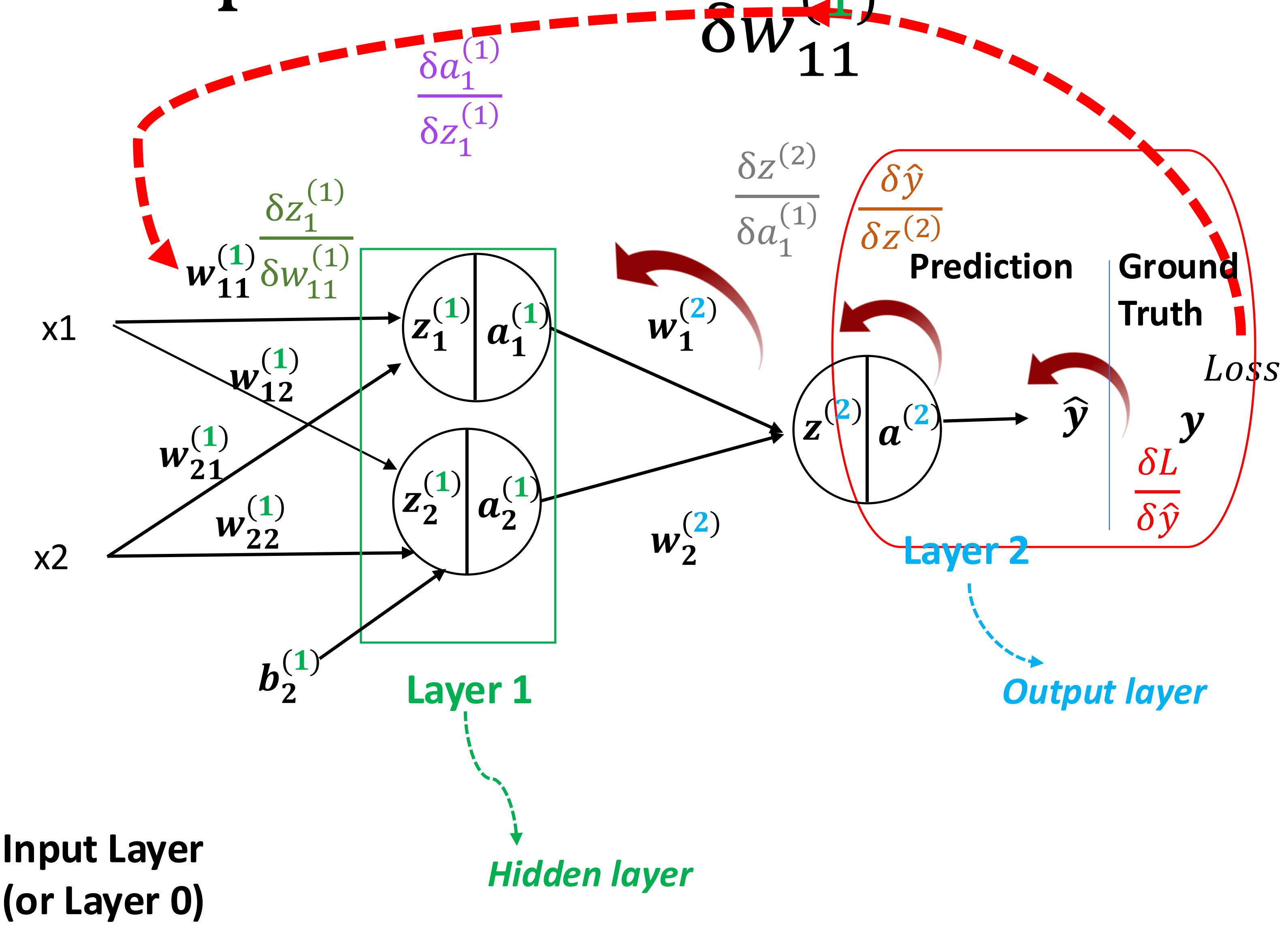
$$z^{(2)} = w_1^{(2)} \cdot a_1^{(1)} + b_1^{(1)}$$

$$a_1^{(1)} = \sigma(z^{(1)})$$

$$z^{(1)} = w_{11}^{(1)} \cdot x_1 + w_{21}^{(1)} \cdot x_2 + b_1^1$$

$$\frac{\delta L}{\delta w_{11}^{(1)}} = \boxed{\frac{\delta L}{\delta \hat{y}} \frac{\delta \hat{y}}{\delta z^{(2)}}} \frac{\delta z^{(2)}}{\delta a_1^{(1)}} \frac{\delta a_1^{(1)}}{\delta z_1^{(1)}} \frac{\delta z_1^{(1)}}{\delta w_{11}^{(1)}}$$

Computation of $\frac{\delta L}{\delta w_{11}^{(1)}}$



$$L(y, \hat{y}) = (y - \hat{y})^2$$

$$\hat{y} = \sigma(z^{(2)})$$

$$z^{(2)} = w_1^{(2)} \cdot a_1^{(1)} + b_1^{(1)}$$

$$a_1^{(1)} = \sigma(z^{(1)})$$

$$z^{(1)} = w_{11}^{(1)} \cdot x_1 + w_{21}^{(1)} \cdot x_2 + b_1^1$$

$$\frac{\delta L}{\delta w_{11}^{(1)}} = \boxed{\frac{\delta L}{\delta \hat{y}} \frac{\delta \hat{y}}{\delta z^{(2)}}} \frac{\delta z^{(2)}}{\delta a_1^{(1)}} \frac{\delta a_1^{(1)}}{\delta z_1^{(1)}} \frac{\delta z_1^{(1)}}{\delta w_{11}^{(1)}}$$

Computation of $\frac{\delta L}{\delta w_{11}^{(1)}}$

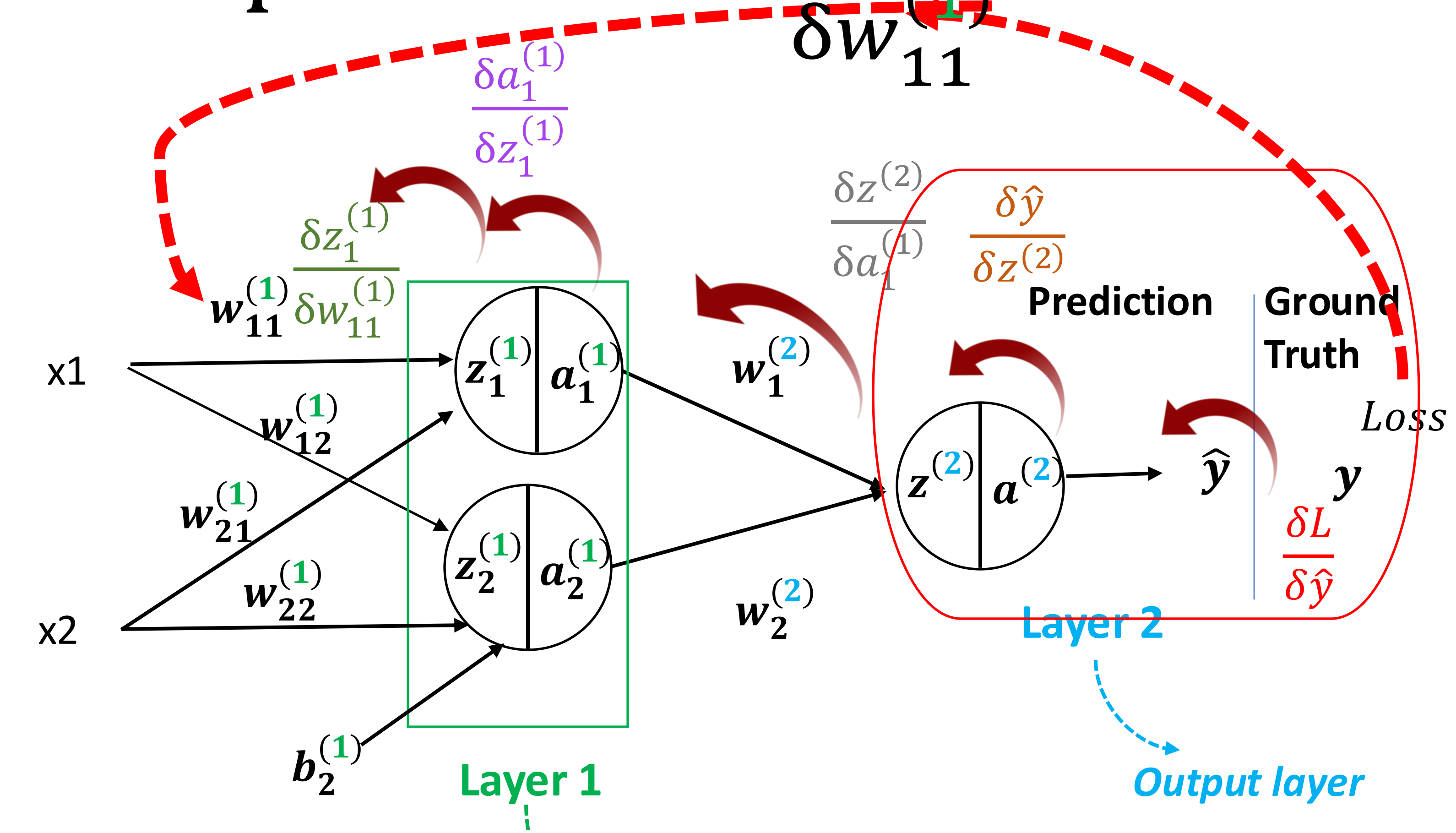
$$L(y, \hat{y}) = (y - \hat{y})^2$$

$$\hat{y} = \sigma\left(z^{(2)}\right)$$

$$z^{(2)} = w_1^{(2)} \cdot a_1^{(1)} + b_1^{(1)}$$

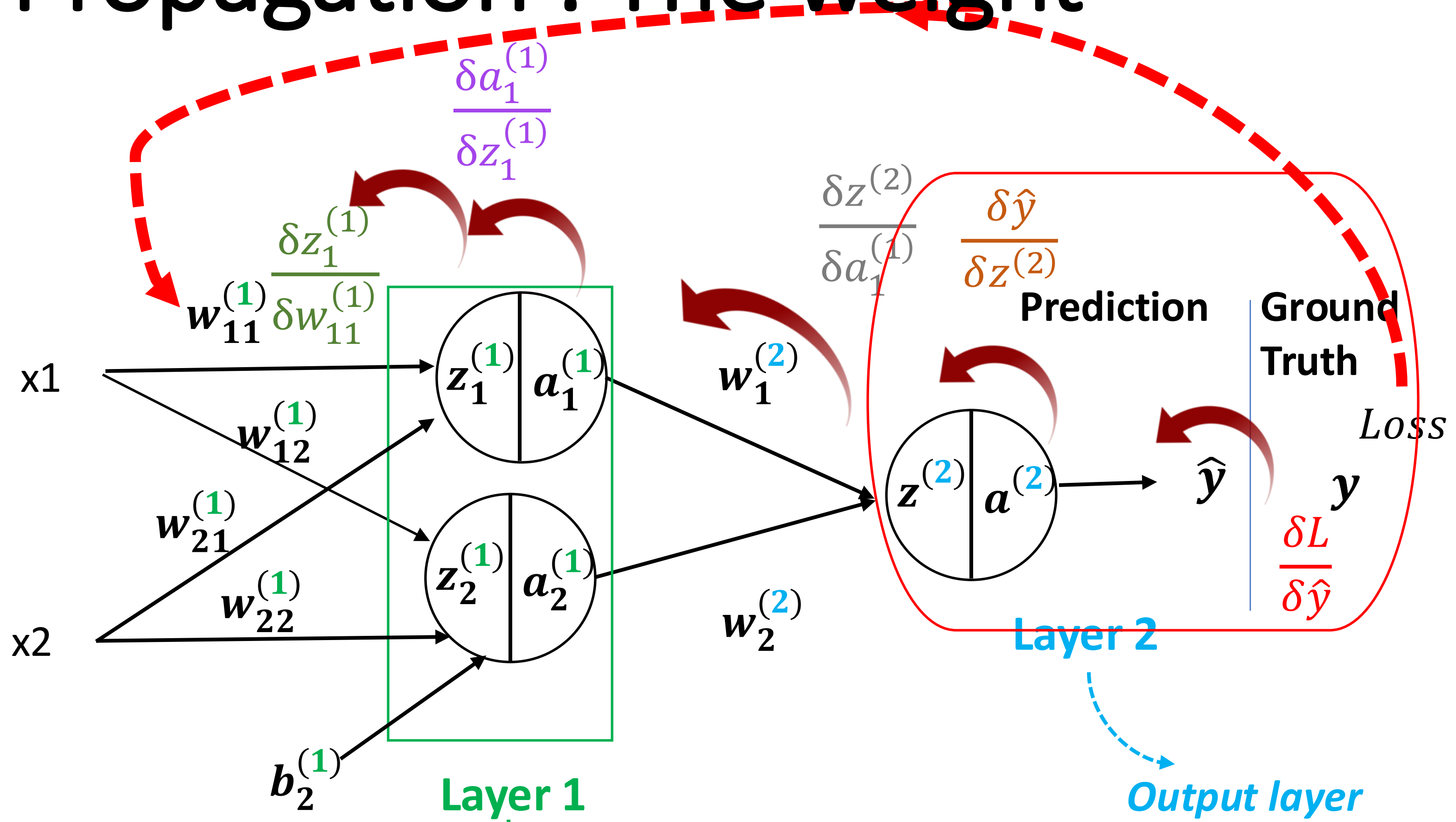
$$a_1^{(1)} = \sigma\left(z^{(1)}\right)$$

$$z^{(1)} = w_{11}^{(1)} \cdot x_1 + w_{21}^{(1)} \cdot x_2 + b_1^1$$



$$\frac{\delta L}{\delta w_{11}^{(1)}} = -(y - \hat{y})\sigma\left(z^{(2)}\right)\left(1 - \sigma\left(z^{(2)}\right)\right) w_1^{(2)} \sigma\left(z^{(1)}\right)\left(1 - \sigma\left(z^{(1)}\right)\right) x_1$$

Propagation : The weight



Input Layer
(Or Layer 0)

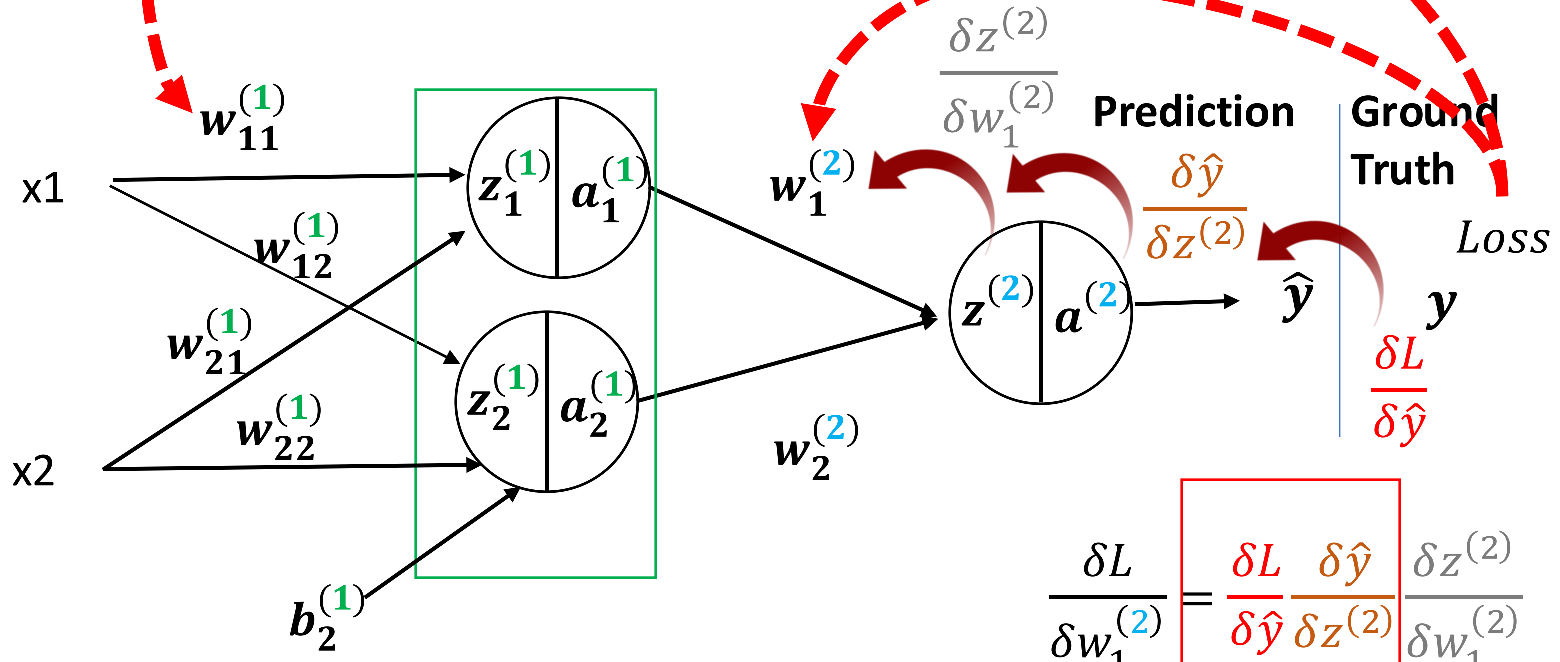
Hidden layer

$$\frac{\delta L}{\delta w_{11}^{(1)}} = -0.0899 \times 0.6 \times 0.5915 \times (1-0.5915) \times 0.4 = 0.0052$$

$$w_{1,new}^{(1)} = w_{1,old}^{(1)} - \eta \frac{\delta L}{\delta w_1^{(1)}}$$

$$w_{1,new}^{(1)} = 0.3 - 0.01 \times 0.0052 = 0.299948$$

Common link between "layer 2" and "layer 1"



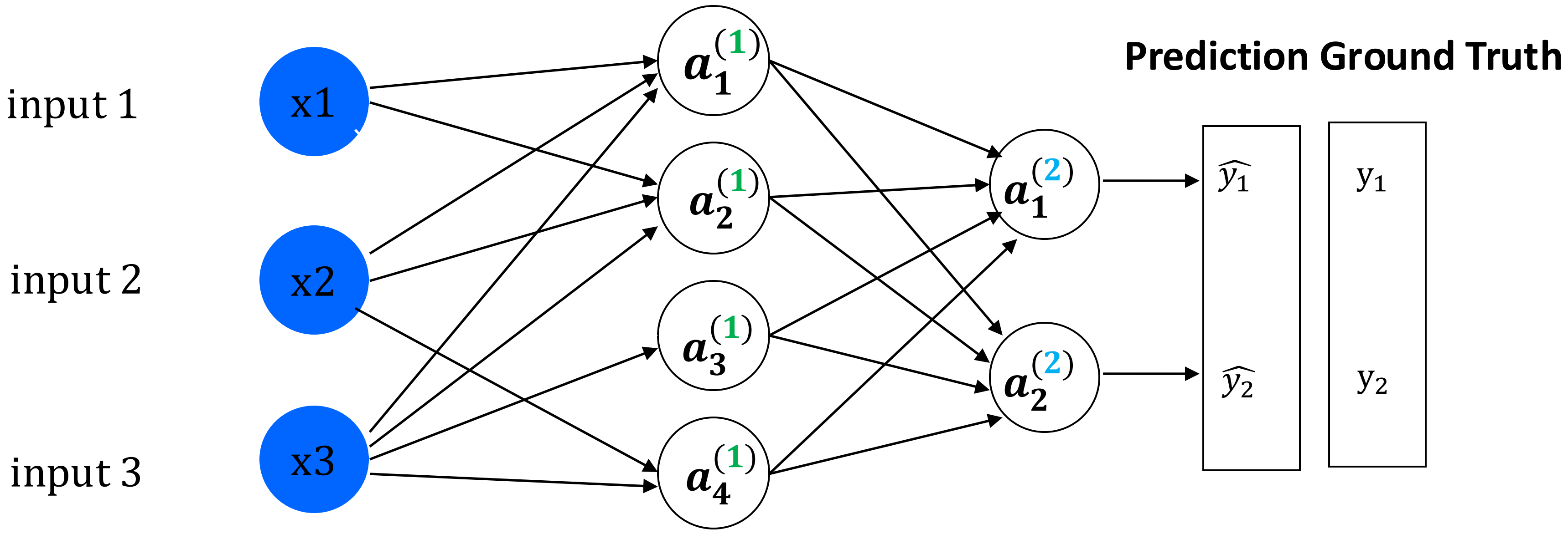
$$\frac{\delta L}{\delta w_1^{(2)}} = \boxed{\frac{\delta L}{\delta \hat{y}} \frac{\delta \hat{y}}{\delta z^{(2)}}} \frac{\delta z^{(2)}}{\delta w_1^{(2)}}$$

$$\frac{\delta L}{\delta z^{(2)}} = \delta^{(2)}$$

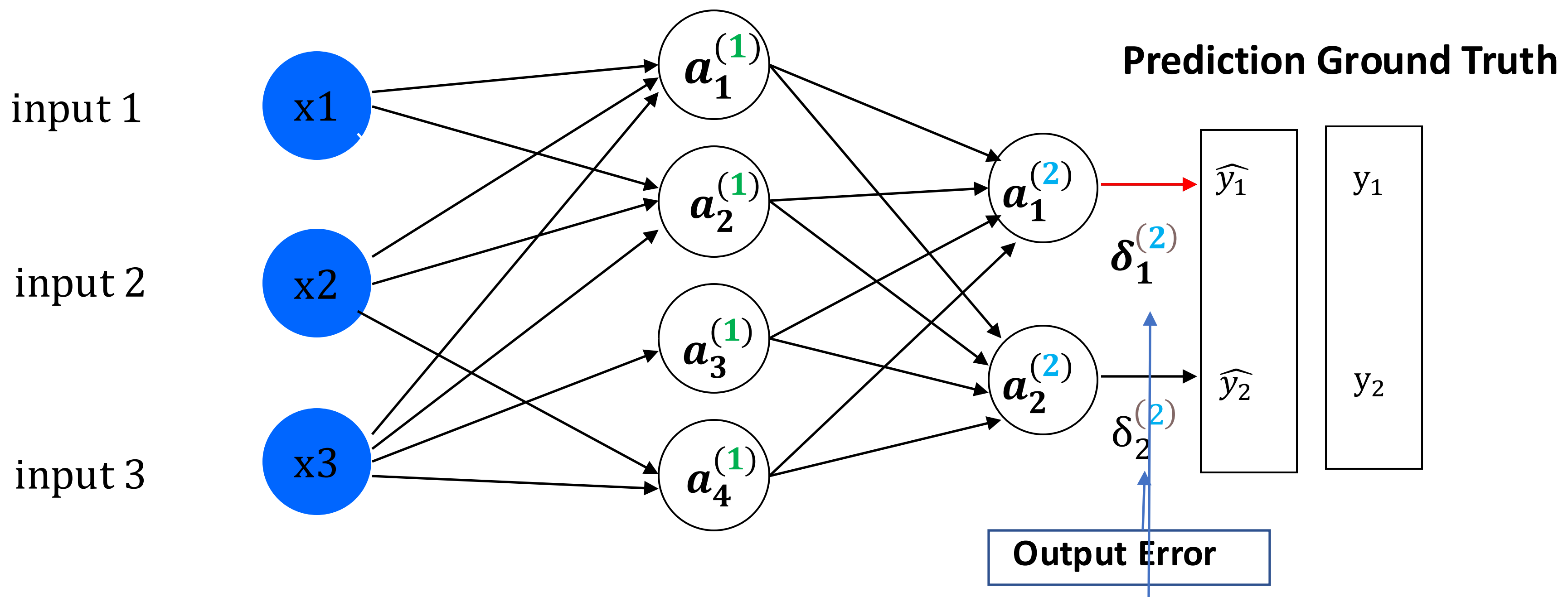
$$\frac{\delta L}{\delta w_{11}^{(1)}} = \boxed{\frac{\delta L}{\delta \hat{y}} \frac{\delta \hat{y}}{\delta z^{(2)}}} \frac{\delta z^{(2)}}{\delta a_1^{(1)}} \frac{\delta a_1^{(1)}}{\delta z_1^{(1)}} \frac{\delta z_1^{(1)}}{\delta w_{11}^{(1)}}$$

We can create a recursive algorithm that does this.

Delta rule for recursive update of the gradient



Delta in layer 2

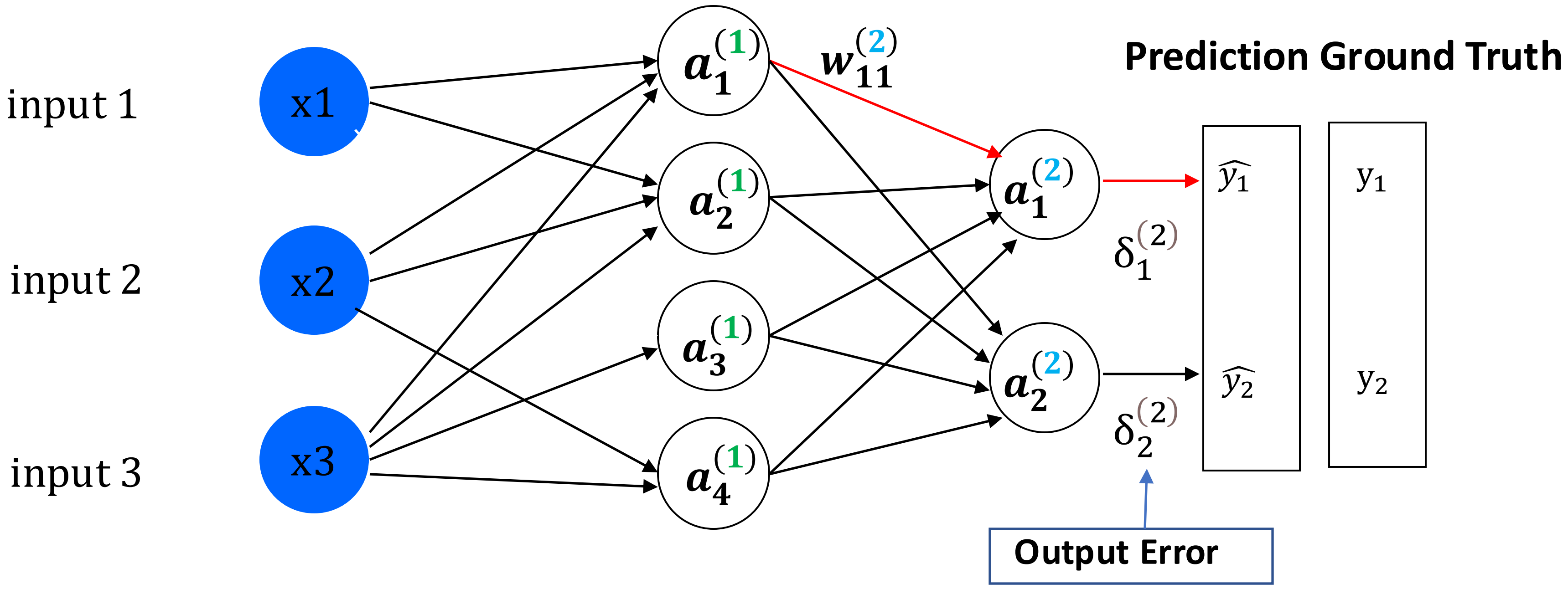


Calculate delta for each output layer

$$\delta_k^{(2)} = \widehat{y_k}(1 - \widehat{y_k})(\widehat{y_k} - y_k)$$

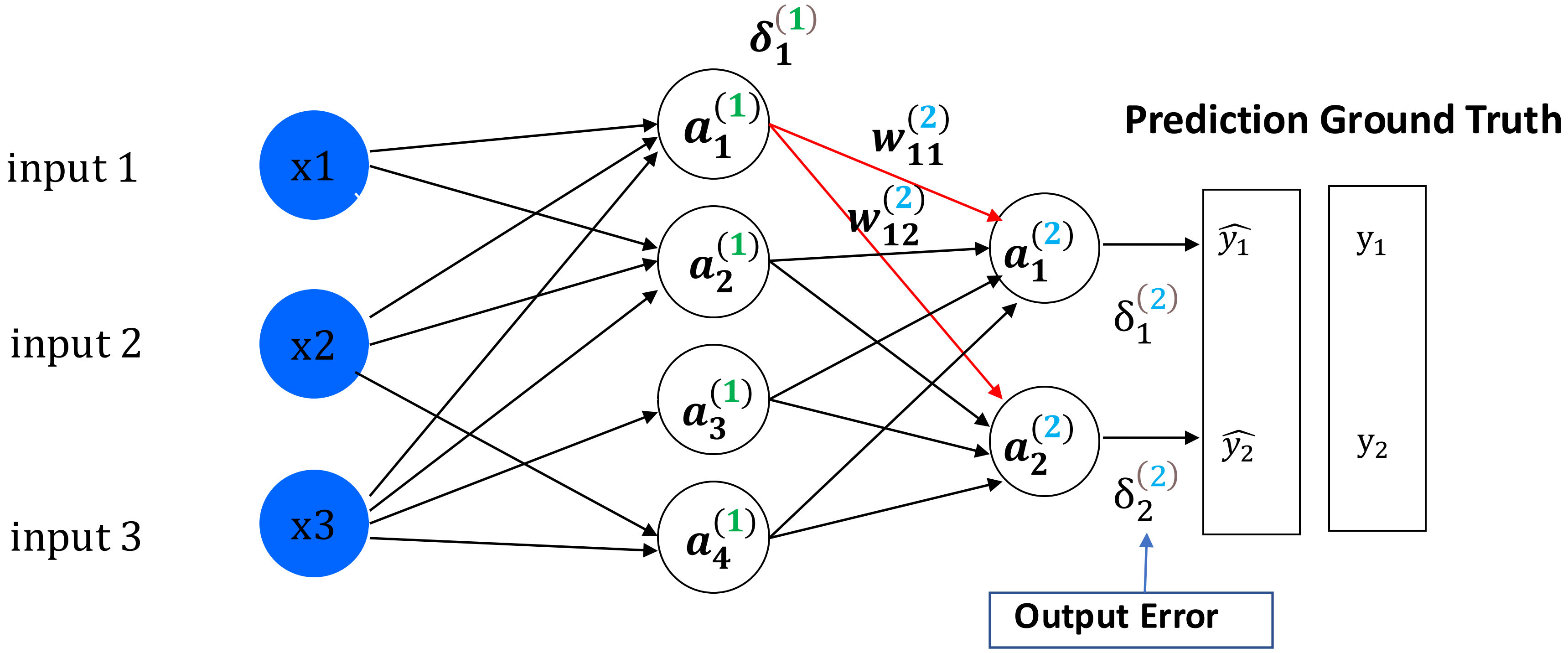
for example $\boldsymbol{\delta}_1^{(2)} = \underbrace{\widehat{y_1}(1 - \widehat{y_1})}_{\text{The derivative of The activation function applied to Node}} \underbrace{(\widehat{y_1} - y_1)}_{\text{The derivative of the error component 1}}$

Gradient using delta



$$w_{11,new}^{(2)} = w_{11,\text{old}}^{(2)} - \eta \cdot \delta_1^{(2)} \cdot a_1^{(1)}$$

Delta in layer 1



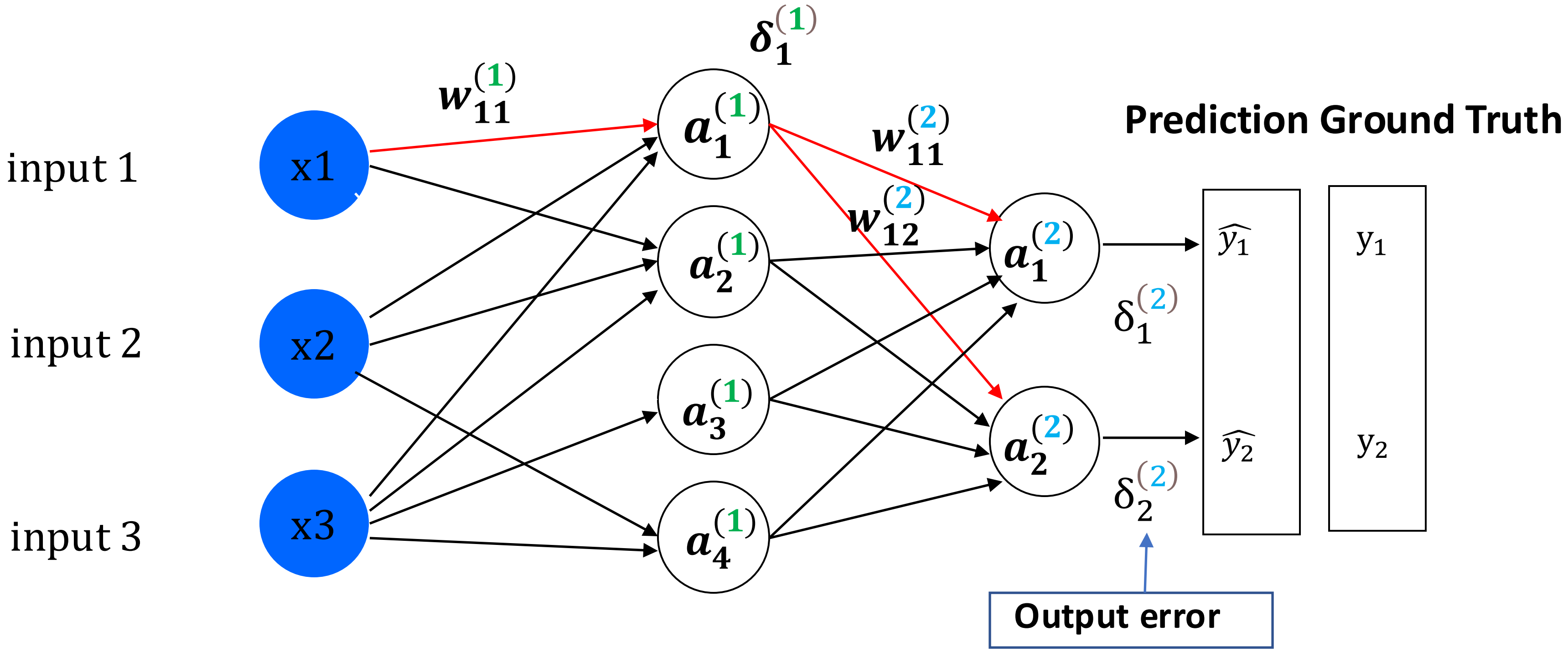
for example

Calculate delta error for each output layer (output error)

$$\delta_j^{(l)} = a_j^{(l)} \cdot \left(1 - a_j^{(l)}\right) \cdot \left(\sum_k \delta_k^{(l+1)} w_{jk}^{(l+1)}\right)$$

$$\boldsymbol{\delta}_1^{(1)} = \underbrace{a_1^{(1)}\left(1 - a_1^{(1)}\right)}_{\text{The derivative of the activation function applied to the node}} \underbrace{\left(w_{11}^{(2)}\delta_1^{(2)} + w_{12}^{(2)}\delta_2^{(2)}\right)}_{\text{weighted sum of delta from the next layer}}$$

Gradient using delta



for example

$$w_{11,new}^{(1)} = w_{11,old}^{(1)} - \eta \cdot \delta_1^{(1)} \cdot x_1$$

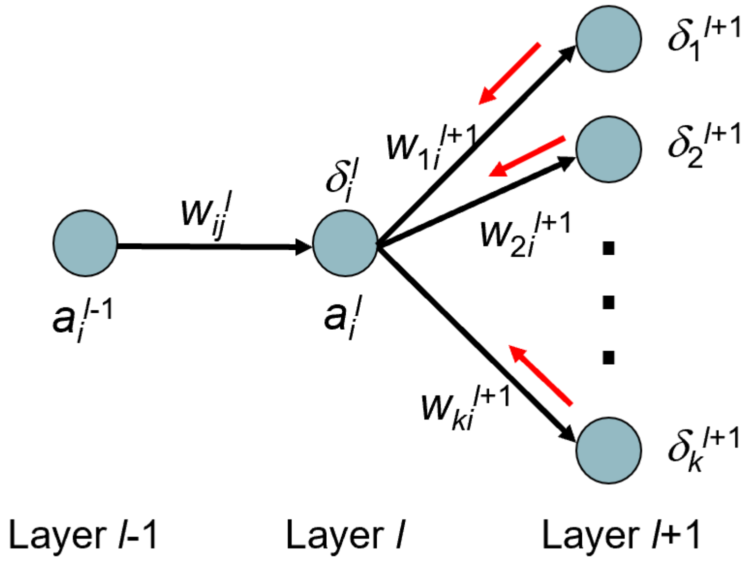
A recursive algorithm for delta refresh

- Error calculation: (Last layer L), where we make predictions $\hat{y}_i$

$$\delta_i^{(L)} = (\hat{y}_i - y_i) \cdot \hat{y}_i \cdot (1 - \hat{y}_i)$$

Error calculation: For any hidden layer $l < L$

$$\delta_j^{(l)} = \underbrace{a_j^{(l)} \cdot \left(1 - a_j^{(l)}\right)}_{\text{the derivative of the activation function (for a sigmoid activation function)}} \cdot \underbrace{\left(\sum_k \delta_k^{(l+1)} w_{jk}^{(l+1)}\right)}_{\textbf{weighted total delta from the next team}}$$



A recursive algorithm for delta refresh

- Error calculation: (Last layer L), where we make predictions $\widehat{y}_i$

$$\delta_i^{(L)} = (\widehat{y}_i - y_i) \cdot \widehat{y}_i \cdot (1 - \widehat{y}_i)$$

Error calculation: For any hidden layer l where $l < L$

$$\delta_j^{(l)} = \underbrace{a_j^{(l)} \cdot \left(1 - a_j^{(l)}\right)}_{\text{the derivative of the activation function (for a sigmoid activation function)}} \cdot \left(\underbrace{\sum_k \delta_k^{(l+1)} w_{jk}^{(l+1)}}_{\textbf{weighted total delta from the next team}}\right)$$

- Weight update formulas
- $w_{ij,new}^{(l)} = w_{ij,\mathrm{old}}^{(l)} - \eta \cdot \delta_j^{(l)} \cdot a_i^{(l-1)}$

- η is learning rate
- $\delta_j^{(l)}$ is delta error from noden j (i layer l)
- $a_i^{(l-1)}$ is activation from noden i (i layer $l - 1$)

- Specific Case for the Input Layer($a_i^{(0)} = x_i$)

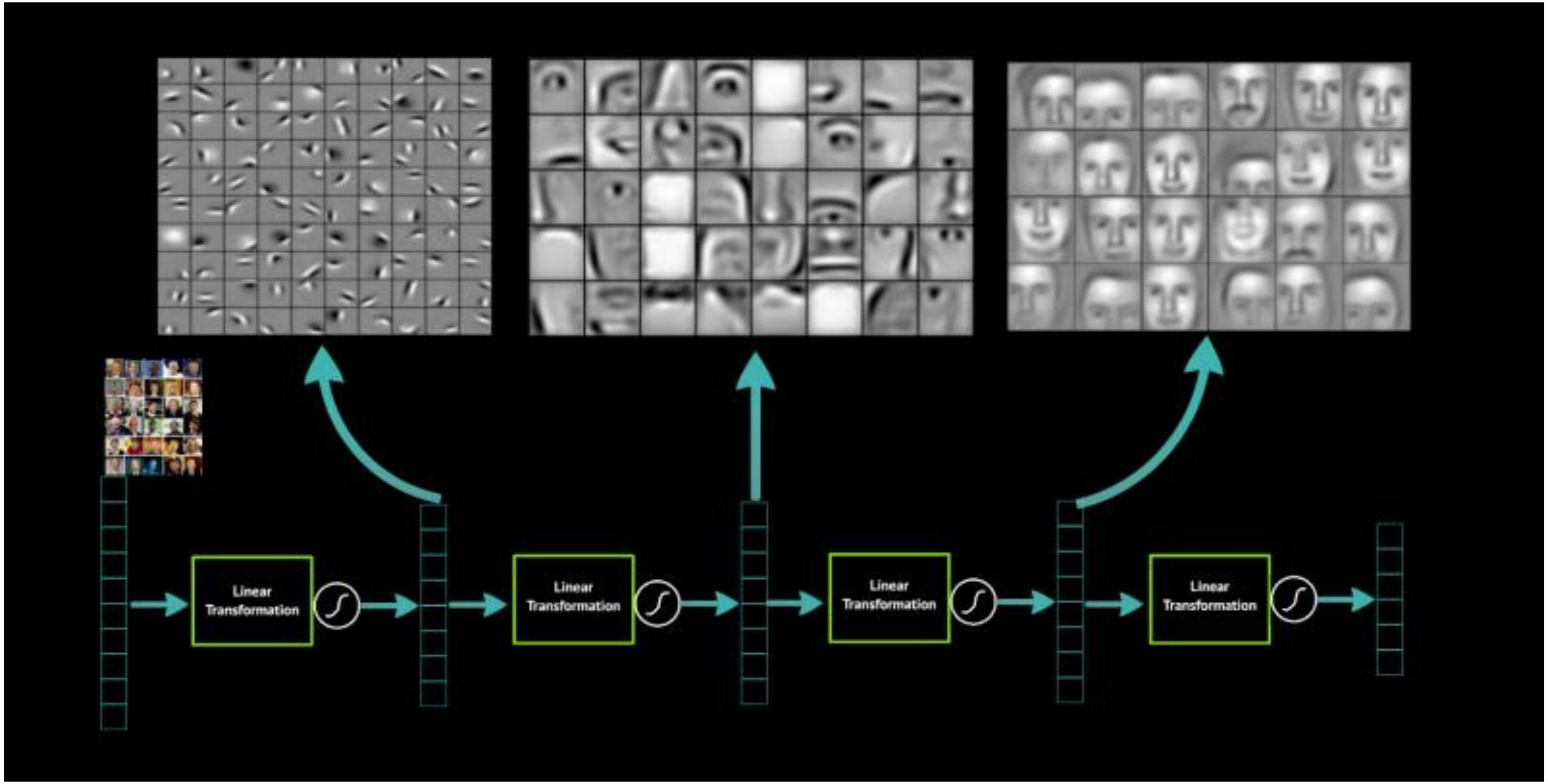
$$w_{ij,new}^{(1)} = w_{ij,\mathrm{old}}^{(1)} - \eta \cdot \delta_j^{(1)} \cdot x_i$$

Deep Neural Networks: Learned Representations

What is meant by learned representations, and how do we learn useful representations?

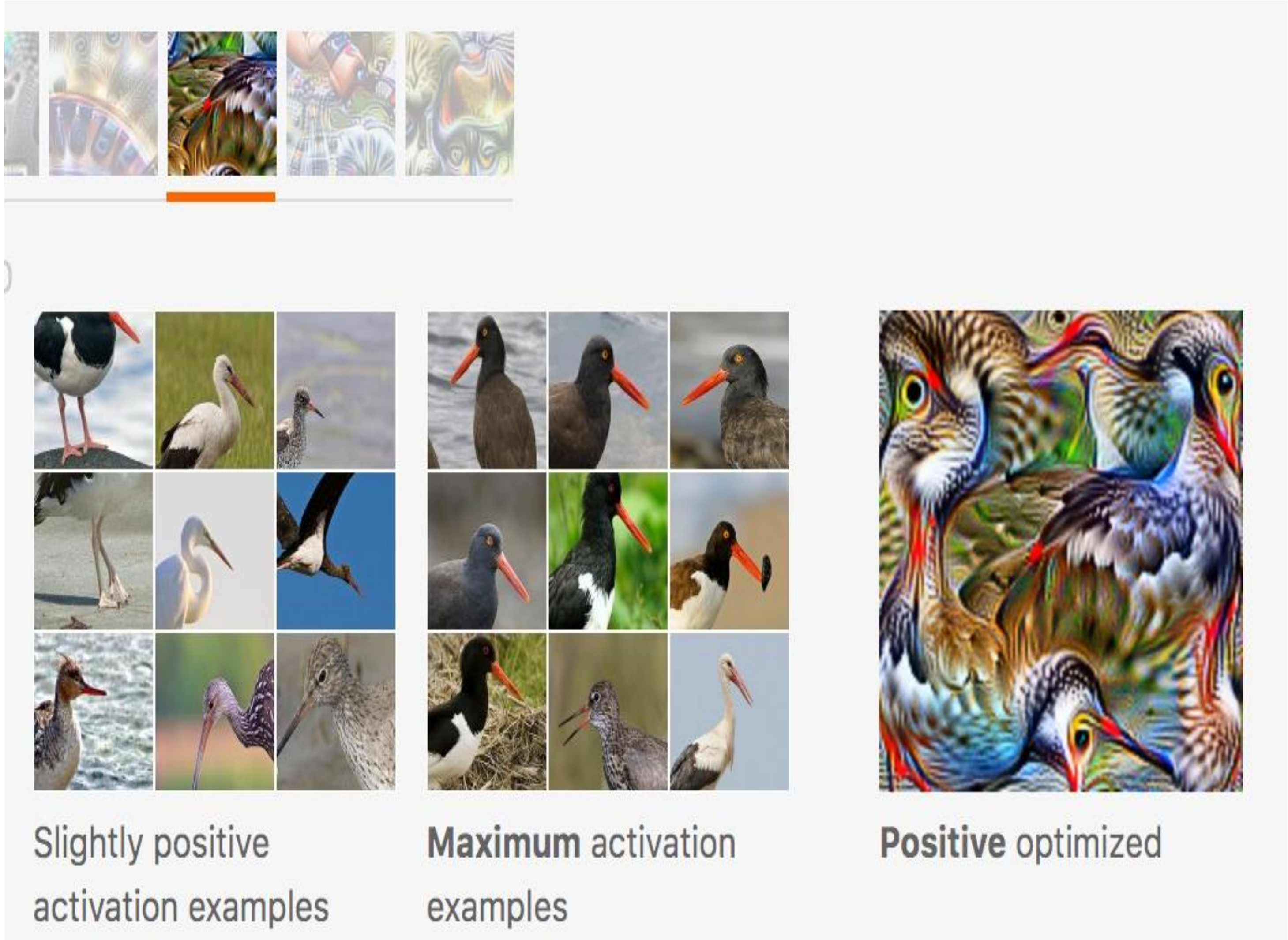
What is a Learned Representation?

- In deep learning, each layer of a neural network learns to transform input data.
- Early layers teach simple patterns like edges, while deeper layers capture more abstract concepts.
- Multiple layers allow the learning of hierarchical representations, from simple to complex.



https://devblogs.nvidia.com/parallelforall/deep-learning-nutshell-core-concepts/

Multiple layers allow the learning of hierarchical representations, from simple to complex.



https://distill.pub/2017/feature-visualization/

- To find the image on the right, the authors took a node high up in the network, they optimised the input to maximise the activation of that node.
- They also searched the dataset for natural images that caused a high activation in that particular node.

Thank You