Neural Networks

Dr. Amani RAAD

amani.raad@ul.edu.lb

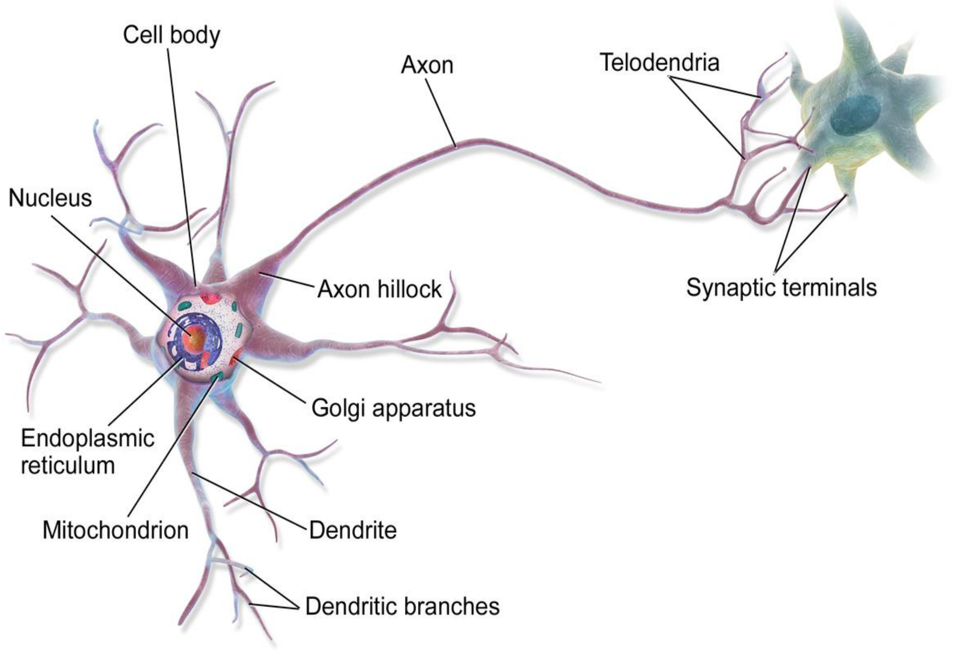
2022-2023

Neural Network Definition

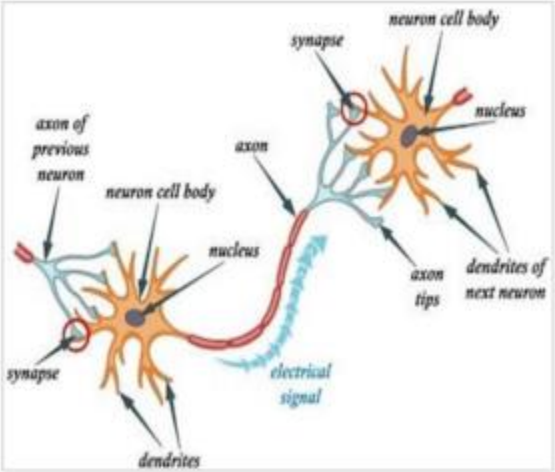
- Currently, the definition of the neural network is not determined yet. Hecht Nielsen, a neural network researcher in the US, defines a neural network as "a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs."
- Based on the origin, features, and interpretations of the neural network, it can be simply defined as **an information processing system designed to simulate human brain's structure and functions**.
- **Artificial neural network (neural network for short):** refers to a network composed of artificial neurons. It abstracts and simplifies a human brain based on its microscopic structure and functions. It is an important way to simulate human intelligence and reflects some basic features of human brain functions, such as parallel information processing, learning, association, pattern classification, and memory.

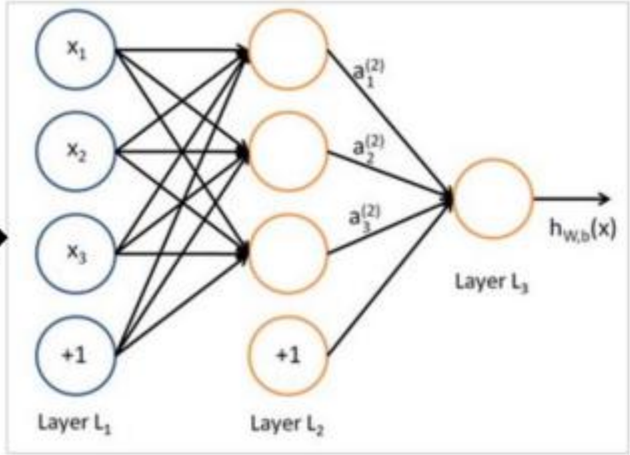
Biologic Neuron

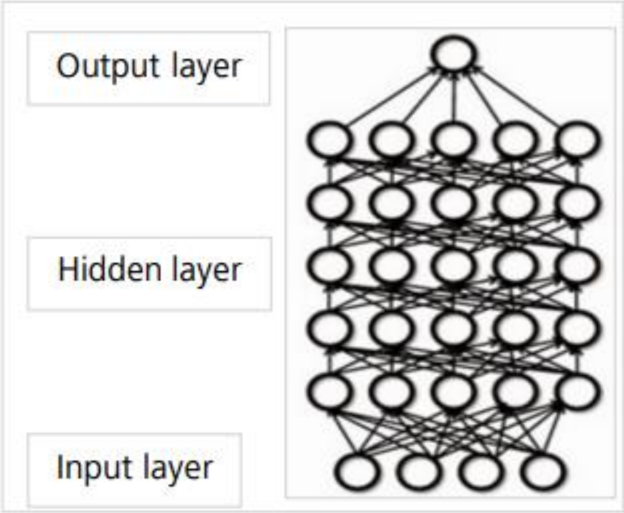
- The basic computational unit of the brain is a neuron
- Neurons receive input signal from dendrites and produce output signal along axon, which interact with the dendrites of other neurons via synaptic weights
- The neuron collects info through dendrites; if this is large enough the neuron fires.



From the brain to the perceptron to the Multi Layer Perceptrons

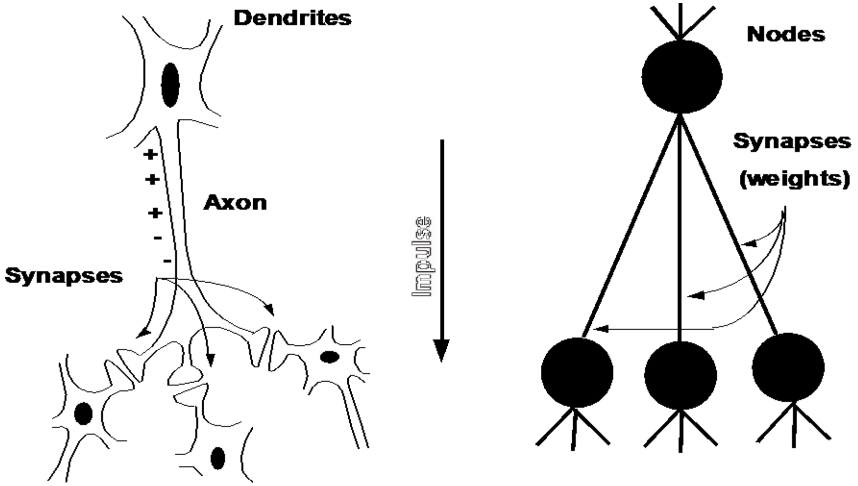




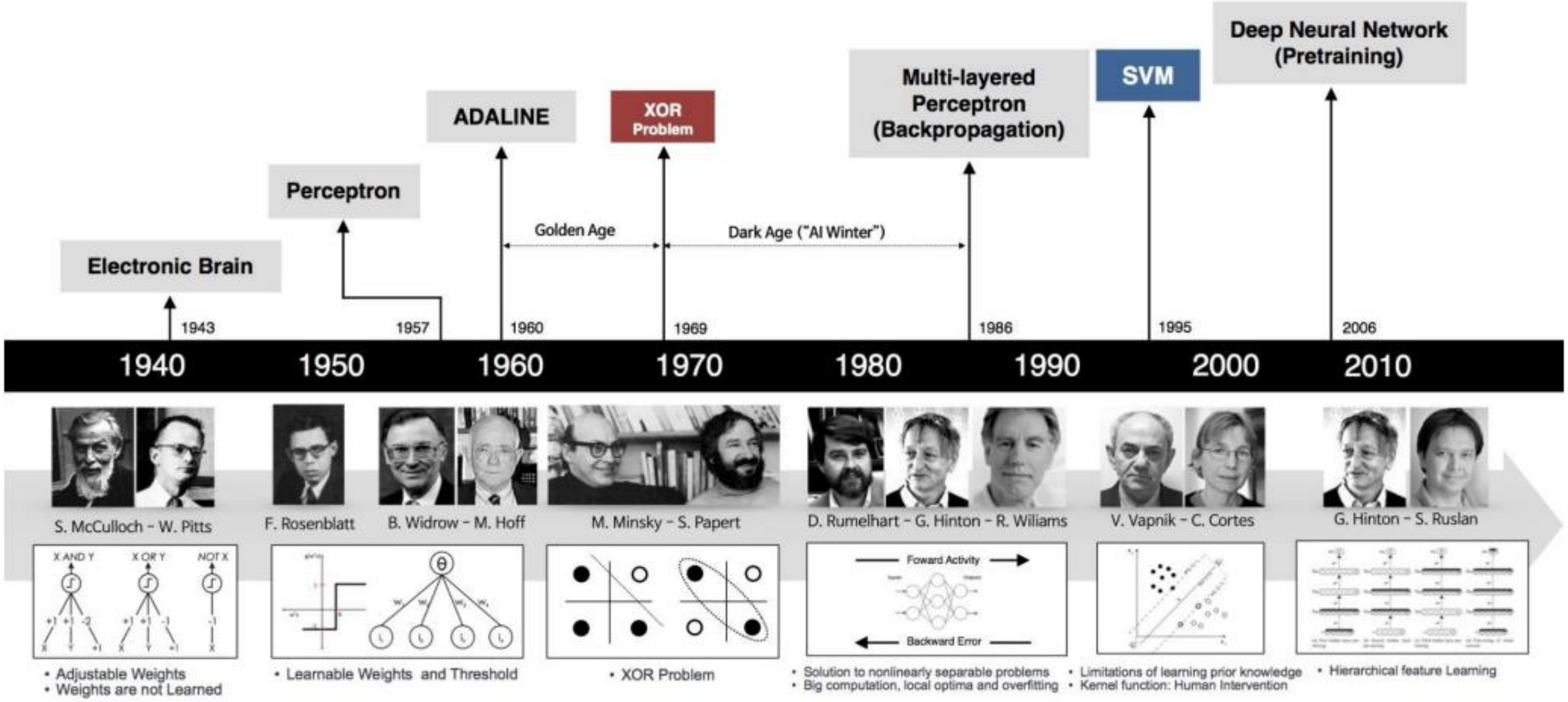


Connectionist Models

- Consider humans:
    - Neuron switching time
      ~ 0.001 second
    - Number of neurons
      ~ 10^{10}
    - Connections per neuron
      ~ 10^{4-5}
    - Scene recognition time
      ~ 0.1 second
    - 100 inference steps doesn't seem like enough
      → much parallel computation
- Properties of artificial neural nets (ANN)
    - Many neuron-like threshold switching units
    - Many weighted interconnections among units
    - Highly parallel, distributed processes

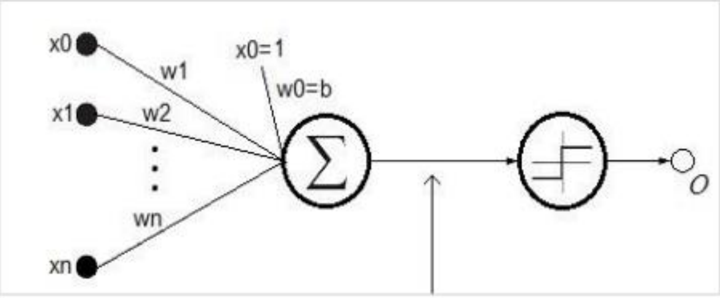


Neural Networks Milestones (Deep Learning)


Electronic Brain
Perceptron
ADALINE
XOR Problem
Multi-layered Perceptron (Backpropagation)
SVM
Deep Neural Network (Pretraining)
Golden Age
Dark Age ("AI Winter")
1943
1957
1960
1969
1986
1995
2006
1940
1950
1960
1970
1980
1990
2000
2010
S. McCulloch - W. Pitts
F. Rosenblatt
B. Widrow - M. Hoff
M. Minsky - S. Papert
D. Rumelhart - G. Hinton - R. Wiliams
V. Vapnik - C. Cortes
G. Hinton - S. Ruslan
X AND Y
X OR Y
NOT X
θ
Foward Activity
Backward Error
• Adjustable Weights
• Weights are not Learned
• Learnable Weights and Threshold
• XOR Problem
• Solution to nonlinearly separable problems
• Big computation, local optima and overfitting
• Limitations of learning prior knowledge
• Kernel function: Human Intervention
• Hierarchical feature Learning

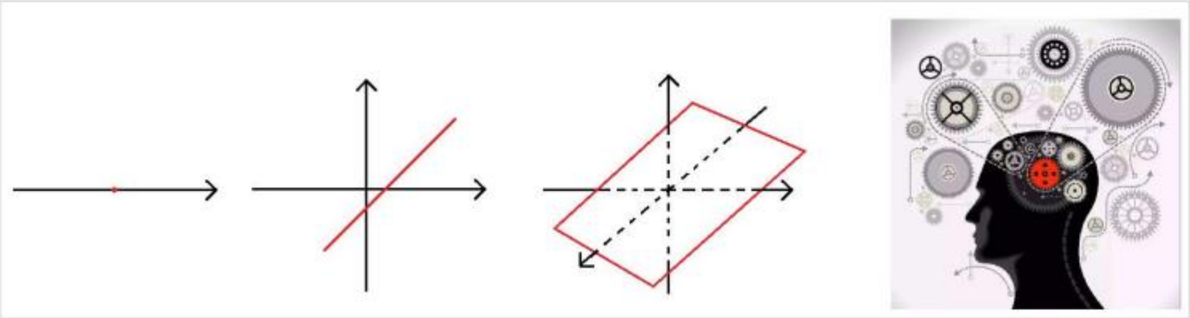
Perceptron

- **Input vector:** $X = [x_0, x_1, \dots, x_n]^T$.
- **Weight:** $W = [\omega_0, \omega_1, \dots, \omega_n]^T$, where ω_0 is the bias.
- **Activation function:** $O = sign(net) = \begin{cases} 1, net > 0, \\ -1, otherwise. \end{cases}$



$$net = \sum_{i=0}^{n} \omega_i x_i = \boldsymbol{W^T X}$$

- The perceptron is equivalent to a classifier. Its input is the high-dimensional vector X and it performs binary classification on input samples in the high-dimensional space. If $\boldsymbol{W^T X} > 0$, $o = 1$ and the sample is classified into one class. Otherwise, $o = -1$ and the sample is classified into the other class. What is the boundary? $\boldsymbol{W^T X} = 0$. This is a high-dimensional hyperplane.



Segmentation point $Ax + B = 0$

Segmentation line $Ax + By + C = 0$

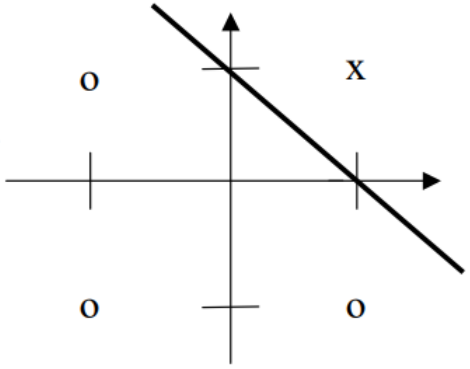
Segmentation surface $Ax + By + Cz + D = 0$

Segmentationhyperplane $W^T X + b = 0$

- Examples of linearly separable classes

- Logical **AND** function

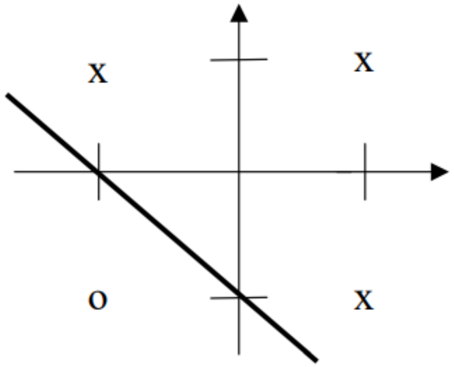
patterns (bipolar)			decision boundary
x1	x2	y	$w1 = 1$
-1	-1	-1	$w2 = 1$
-1	1	-1	$b = -1$
1	-1	-1	$\theta = 0$
1	1	1	$-1 + x1 + x2 = 0$



x: class I (y = 1)
o: class II (y = -1)

- Logical **OR** function

patterns (bipolar)			decision boundary
x1	x2	y	$w1 = 1$
-1	-1	-1	$w2 = 1$
-1	1	1	$b = 1$
1	-1	1	$\theta = 0$
1	1	1	$1 + x1 + x2 = 0$

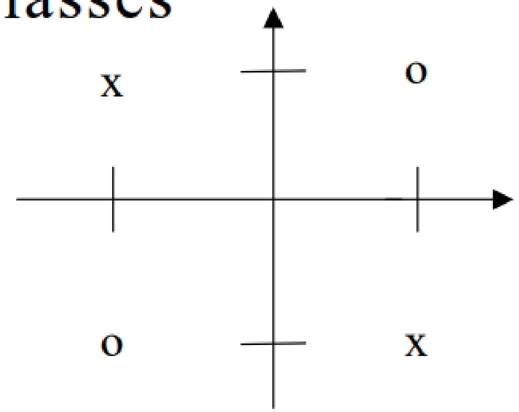


x: class I (y = 1)
o: class II (y = -1)

- Examples of linearly inseparable classes
  - Logical **XOR** (exclusive OR) function

patterns (bipolar) decision boundary

x1	x2	y
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1

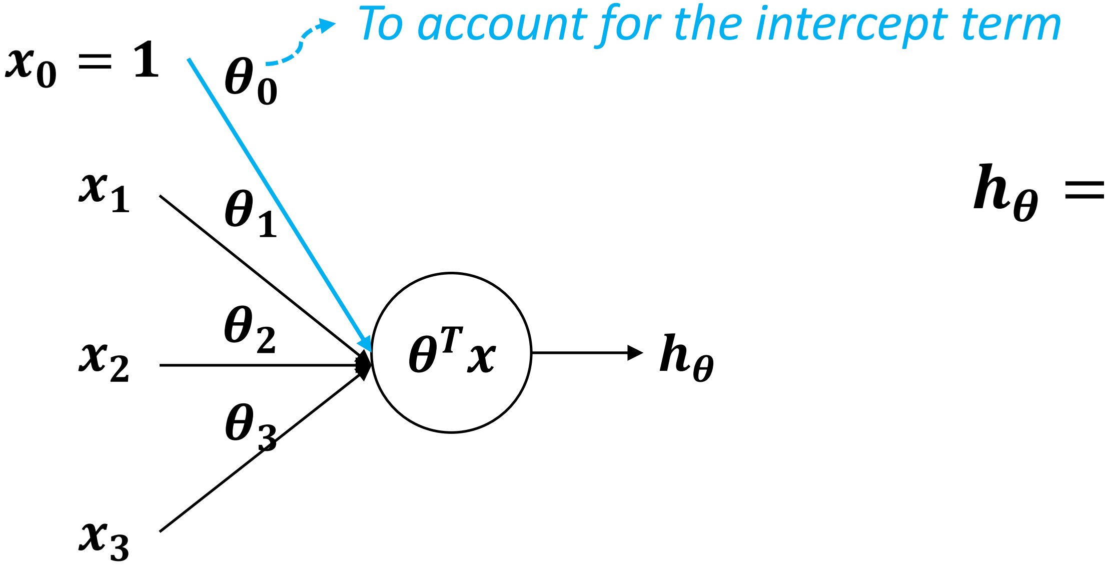


x: class I (y = 1)
o: class II (y = -1)

No line can separate these two classes, as can be seen from

Linear Regression

- What is the hypothesis function of linear regression?
  - $\boldsymbol{h_\theta = \theta^T x}$, where $\boldsymbol{\theta} = [\boldsymbol{\theta_0, \theta_1, \ldots, \theta_m}]$, $\boldsymbol{x} = [\boldsymbol{x_0, x_1, \ldots, x_m}]$, and $\boldsymbol{x_0} = 1$



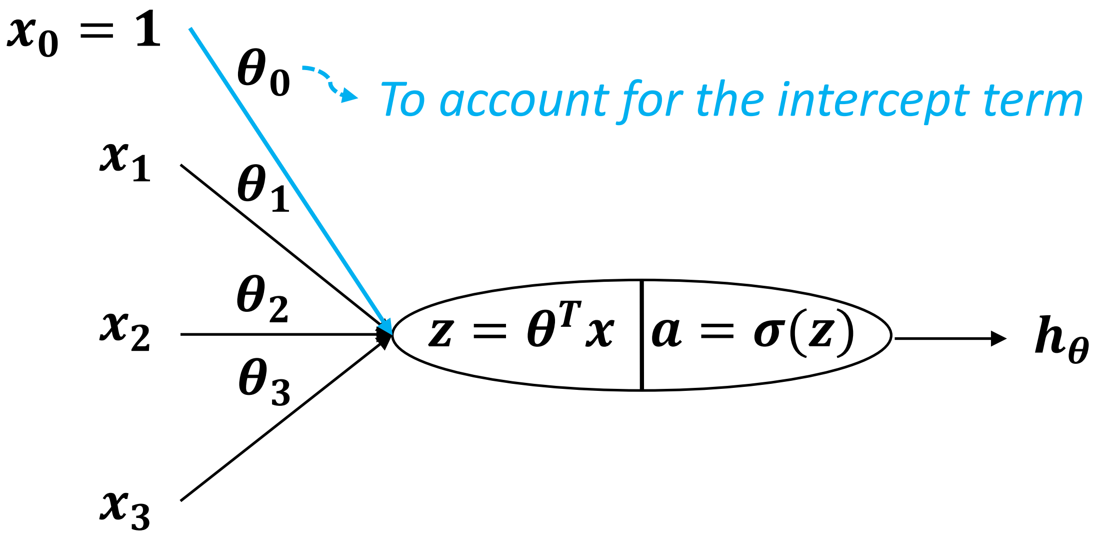
$$\boldsymbol{h_\theta = \theta^T x} = \begin{bmatrix} \boldsymbol{\theta_0} \\ \boldsymbol{\theta_1} \\ \boldsymbol{\theta_2} \\ \boldsymbol{\theta_3} \end{bmatrix} [\boldsymbol{x_0\ x_1\ x_2\ x_3}]$$

$$= \boldsymbol{\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3}$$

$$= \boldsymbol{\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3}$$

Logistic Regression

- What is the hypothesis function of **logistic regression**?
  - $\boldsymbol{h_\theta = \sigma(\theta^T x)}$, where $\boldsymbol{\theta} = [\boldsymbol{\theta_0}, \boldsymbol{\theta_1}, \ldots, \boldsymbol{\theta_m}]$, $\boldsymbol{x} = [\boldsymbol{x_0}, \boldsymbol{x_1}, \ldots, \boldsymbol{x_m}]$, $\boldsymbol{x_0} = 1$, and $\boldsymbol{\sigma(z)} = \frac{1}{1+e^{-z}}$



To account for the intercept term

$$\boldsymbol{h_\theta = a = \sigma(z) = \sigma\left(\theta^T x\right) = \sigma\left(\begin{bmatrix}\theta_0\\ \theta_1\\ \theta_2\\ \theta_3\end{bmatrix}[x_0\ x_1\ x_2\ x_3]\right)}$$

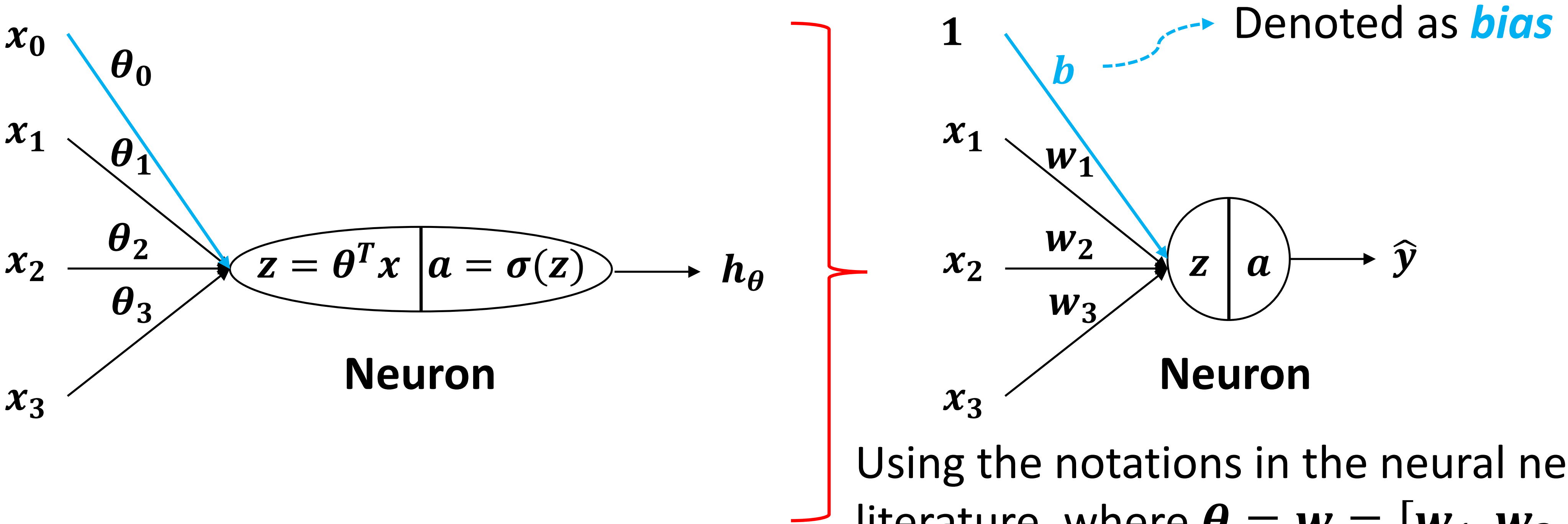
$$\boldsymbol{= \sigma(\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3)}$$

$$\boldsymbol{= \sigma(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3)}$$

$$= \frac{1}{1 + e^{-(\boldsymbol{\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3})}}$$

Towards Neural Networks

- Technically, logistic regression is a **neural network** with only 1 neuron



Neuron

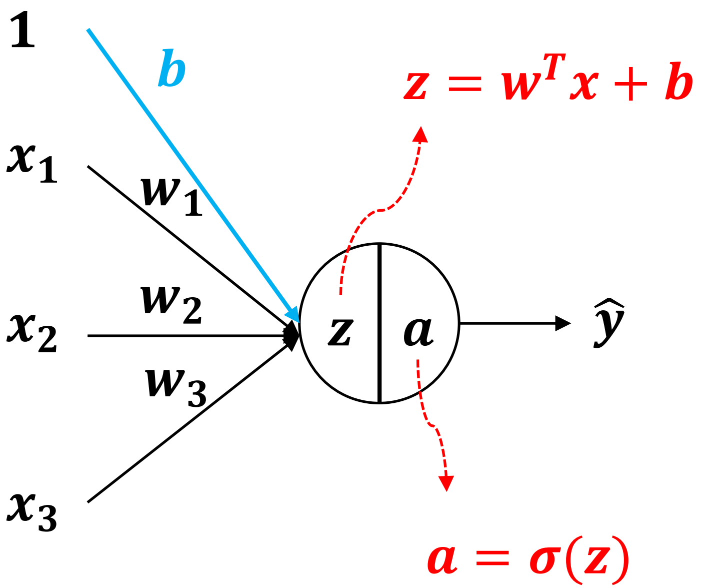
Neuron

Denoted as ***bias***

Using the notations in the neural network literature, where $\boldsymbol{\theta} = \boldsymbol{w} = [\boldsymbol{w}_1, \boldsymbol{w}_2, \boldsymbol{w}_3]$ ($\boldsymbol{w}_0$ is not part of this vector here), $\boldsymbol{h}_{\boldsymbol{\theta}} = \hat{\boldsymbol{y}}$, and $\boldsymbol{\theta}_0 = \boldsymbol{w}_0 = \boldsymbol{b}$

Towards Neural Networks

- Technically, logistic regression is a **neural network** with only 1 neuron



$\boldsymbol{z = w^T x + b}$

$\boldsymbol{a = \sigma(z)}$

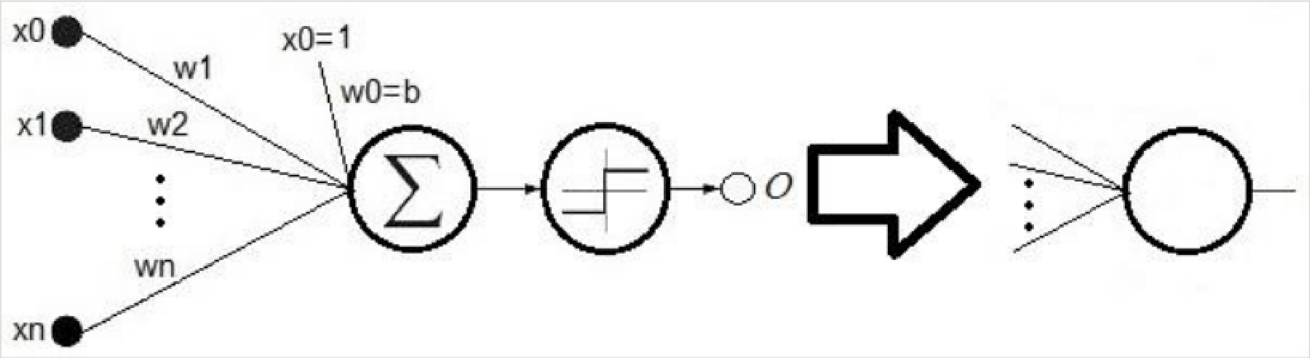
$$\boldsymbol{\hat{y} = a = \sigma(z) = \sigma\left(w^T x + b\right) = \sigma\left(\begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} [x_1 \; x_2 \; x_3] + b\right)}$$

$$\boldsymbol{= \sigma(w_1 x_1 + w_2 x_2 + w_3 x_3 + b)}$$

$$= \frac{1}{1 + e^{-(\boldsymbol{w_1 x_1 + w_2 x_2 + w_3 x_3 + b})}}$$

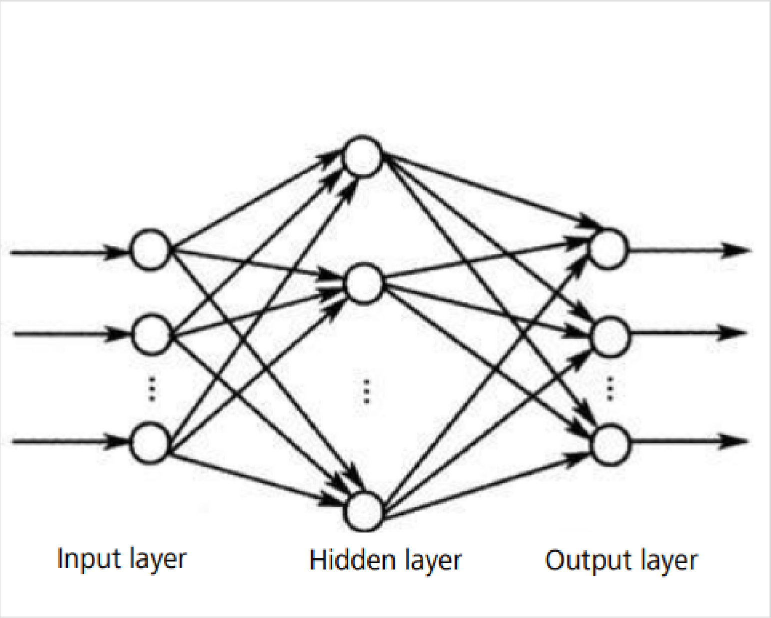
Multi-layers perceptrons MLP
ML Fully Connected

- A single perceptron has limited representation capability and can only represent the linear decision surface (hyperplane). If we connect many perceptrons like a human brain does and then replace the activation function with a non-linear function, we can express a wide range of non-linear surfaces.



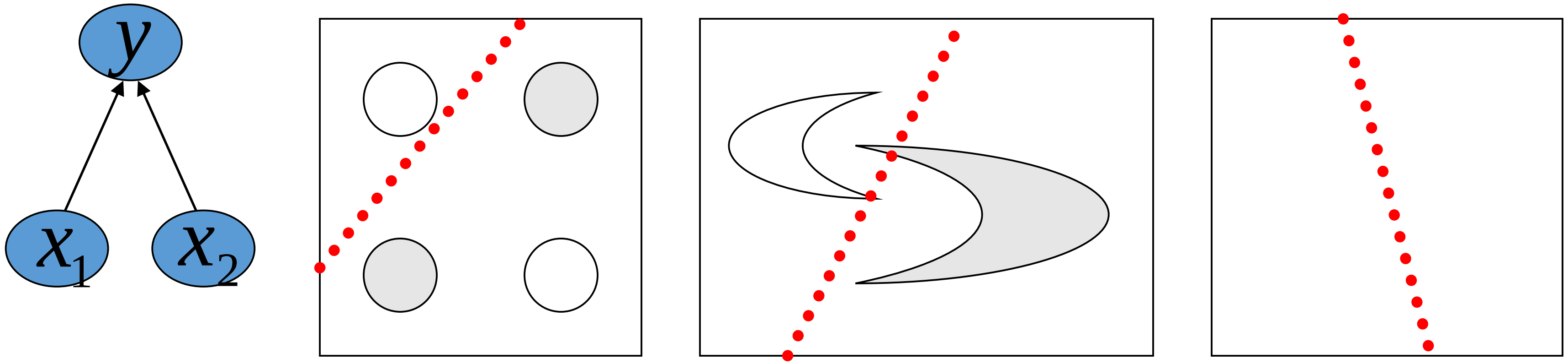
Feed Forward Neural Network

- **The feedforward neural network** is one of the simplest neural networks in which neurons are arranged hierarchically. It is the most widely neural network with the fastest development.
- The input node does not support computation, but is merely used to represent each element value of the input vector.
- Each node represents a neuron that supports computation, which is called a computational unit. Each neuron is connected only to the neurons at the previous layer.
- A hidden layer receives the output from the previous layer and sends the results to the next layer. A unidirectional multi-layer structure is adopted. Each layer contains several neurons. The neurons at the same layer are not connected to each other. Inter-layer information is transmitted only in one direction.



Decision Boundary

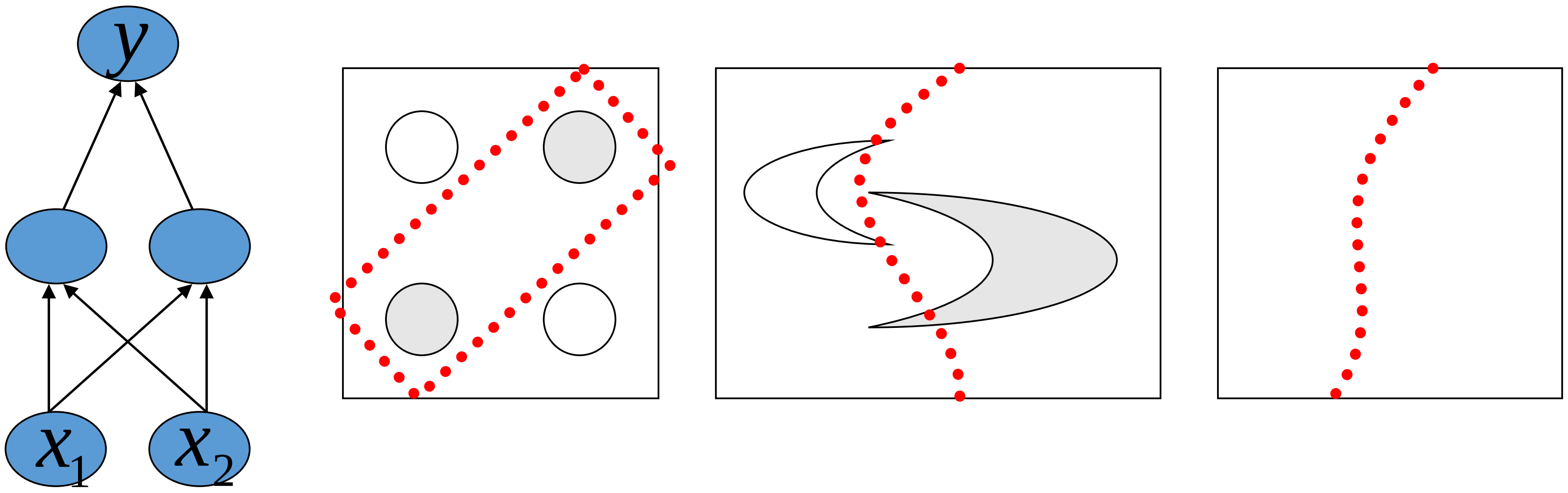
- 0 hidden layers: linear classifier
  - Hyperplanes



Example from to Eric Postma via Jason Eisner

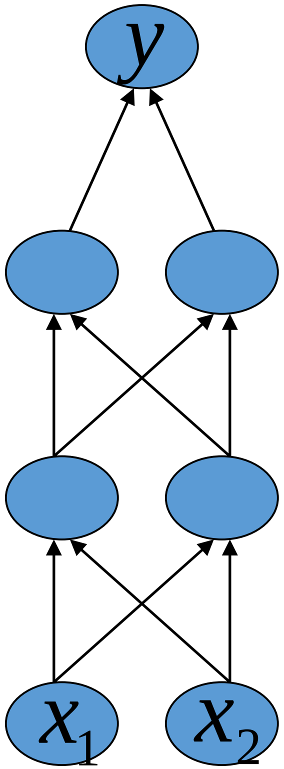
Decision Boundary

- 1 hidden layer
  - Boundary of convex region (open or closed)

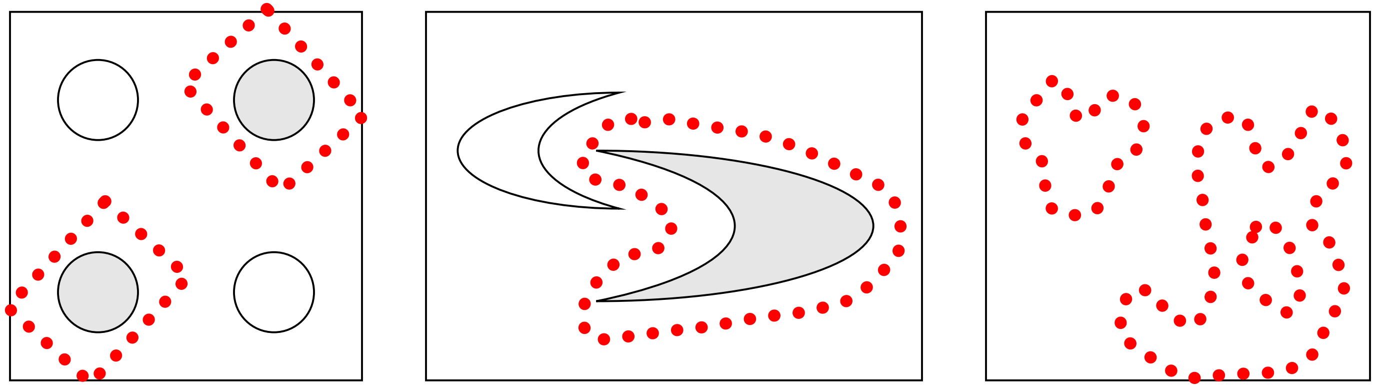


Example from to Eric Postma via Jason Eisner

Decision Boundary

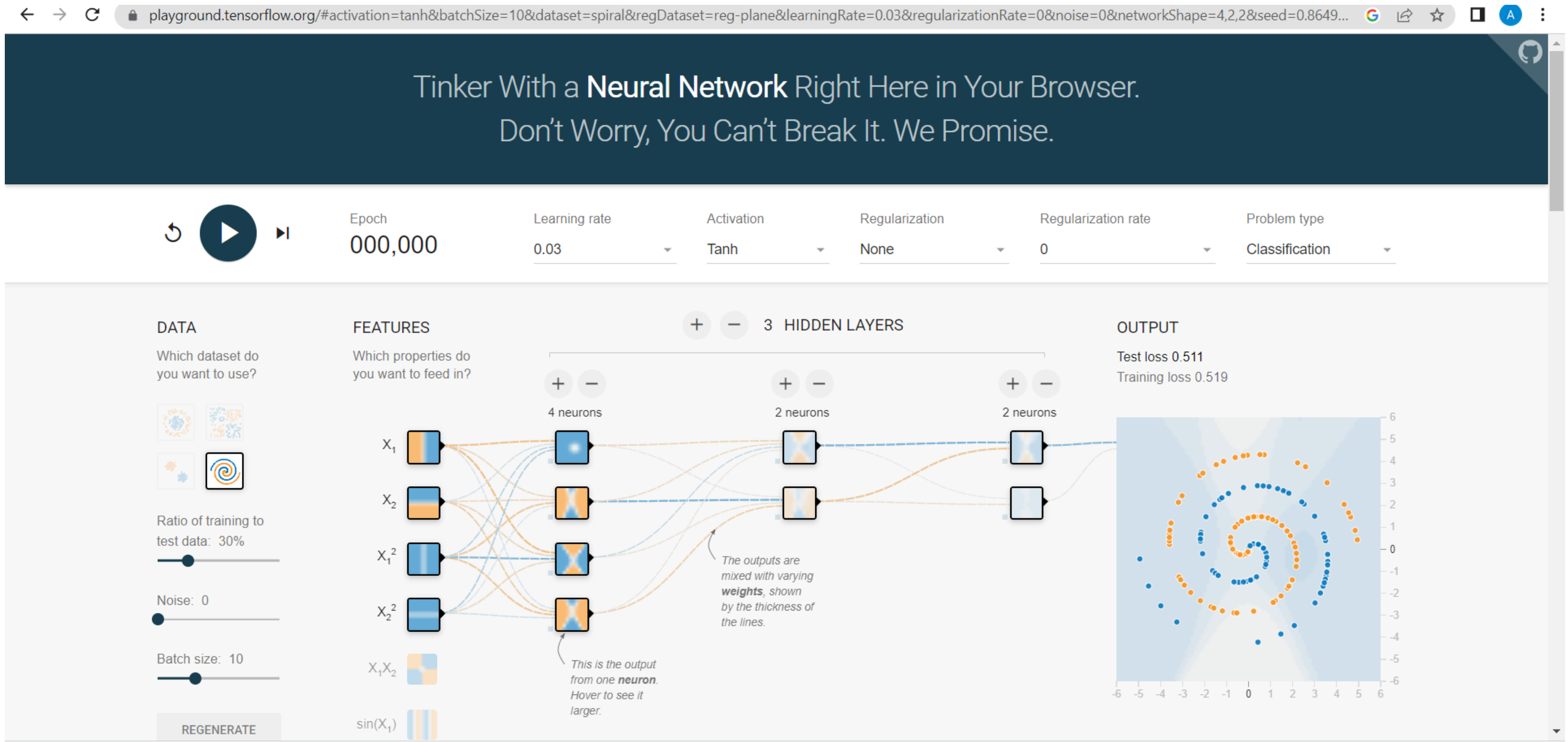


- 2 hidden layers
  - Combinations of convex regions



Example from to Eric Postma via Jason Eisner

playground.tensorflow.org



playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=spiral&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=4,2,2&seed=0.8649...

Tinker With a **Neural Network** Right Here in Your Browser.
Don't Worry, You Can't Break It. We Promise.

Epoch 000,000

Learning rate 0.03

Activation Tanh

Regularization None

Regularization rate 0

Problem type Classification

DATA

Which dataset do you want to use?

Ratio of training to test data: 30%

Noise: 0

Batch size: 10

REGENERATE

FEATURES

Which properties do you want to feed in?

3 HIDDEN LAYERS

OUTPUT

Test loss 0.511

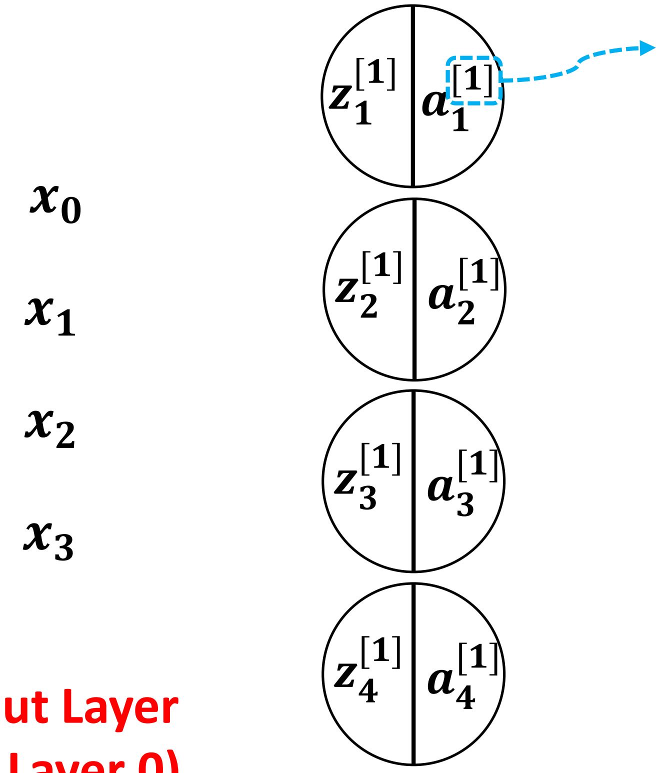
Training loss 0.519

The outputs are mixed with varying ***weights****, shown by the thickness of the lines.*

This is the output from one ***neuron****. Hover to see it larger.*

Neural Networks

- We can construct a network of neurons (i.e., a neural network) with as many *layers,* and neurons in any layer, as needed



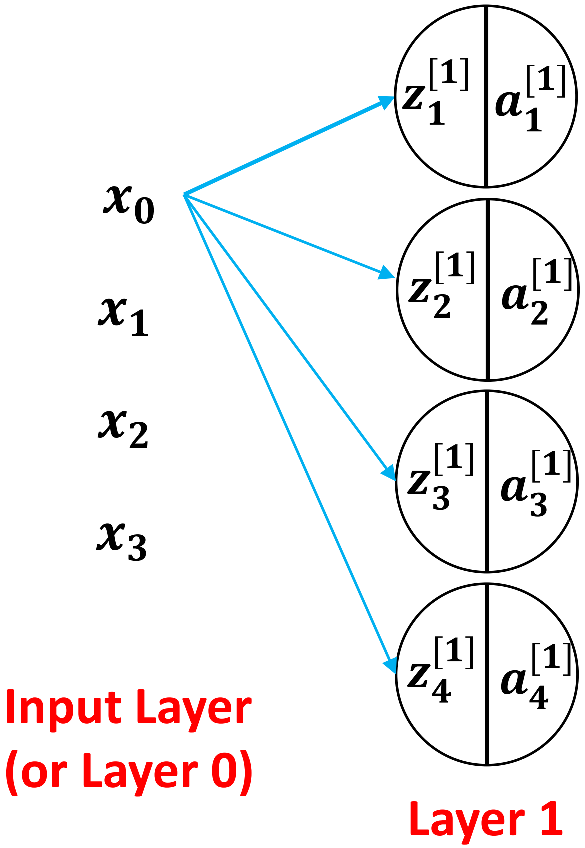
To indicate that this activation, ***a***, is in layer 1

Input Layer (or Layer 0)

Layer 1

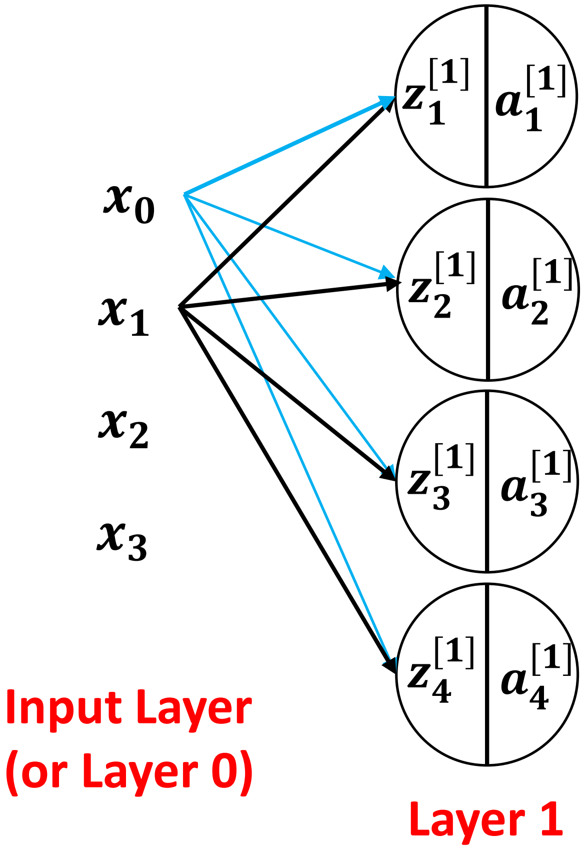
Neural Networks

- We can construct a network of neurons (i.e., a neural network) with as many *layers,* and neurons in any layer, as needed



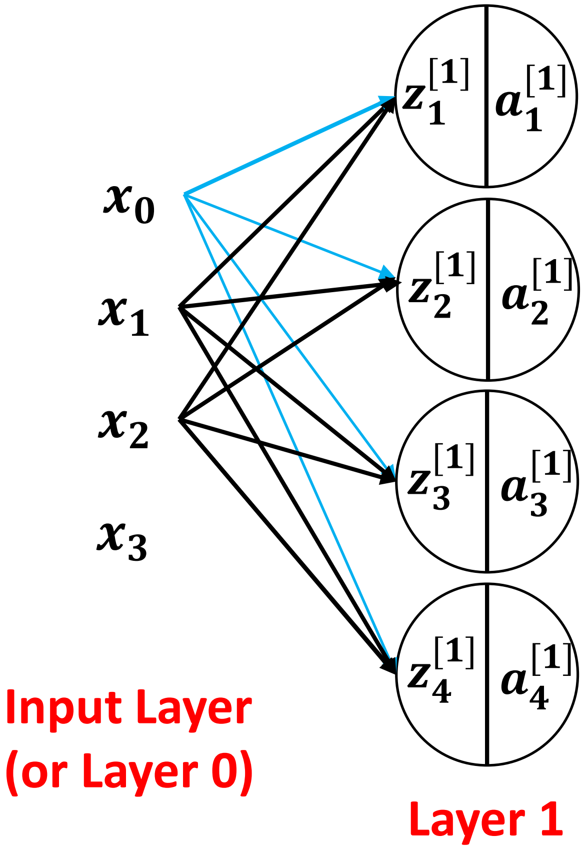
Neural Networks

- We can construct a network of neurons (i.e., a neural network) with as many *layers,* and neurons in any layer, as needed



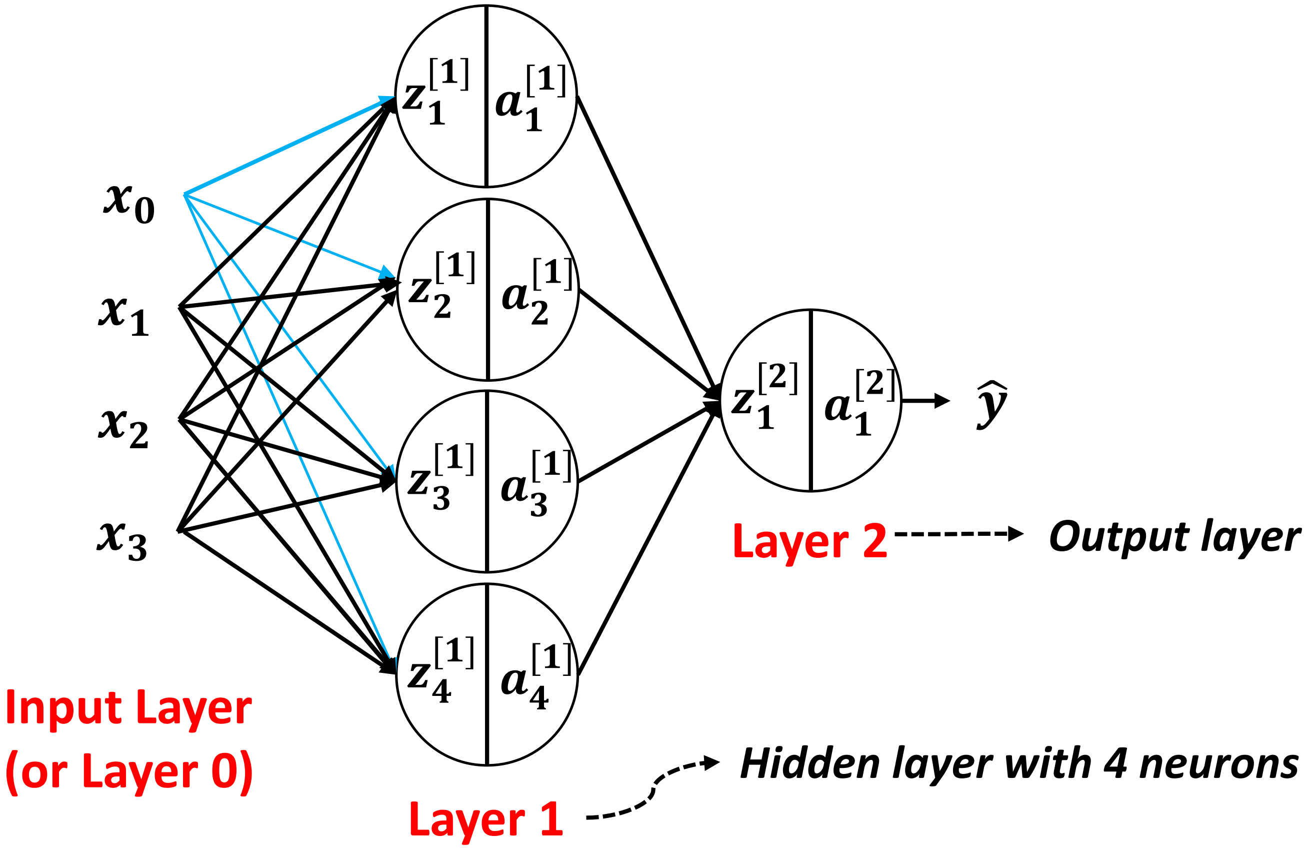
Neural Networks

- We can construct a network of neurons (i.e., a neural network) with as many *layers,* and neurons in any layer, as needed



Neural Networks

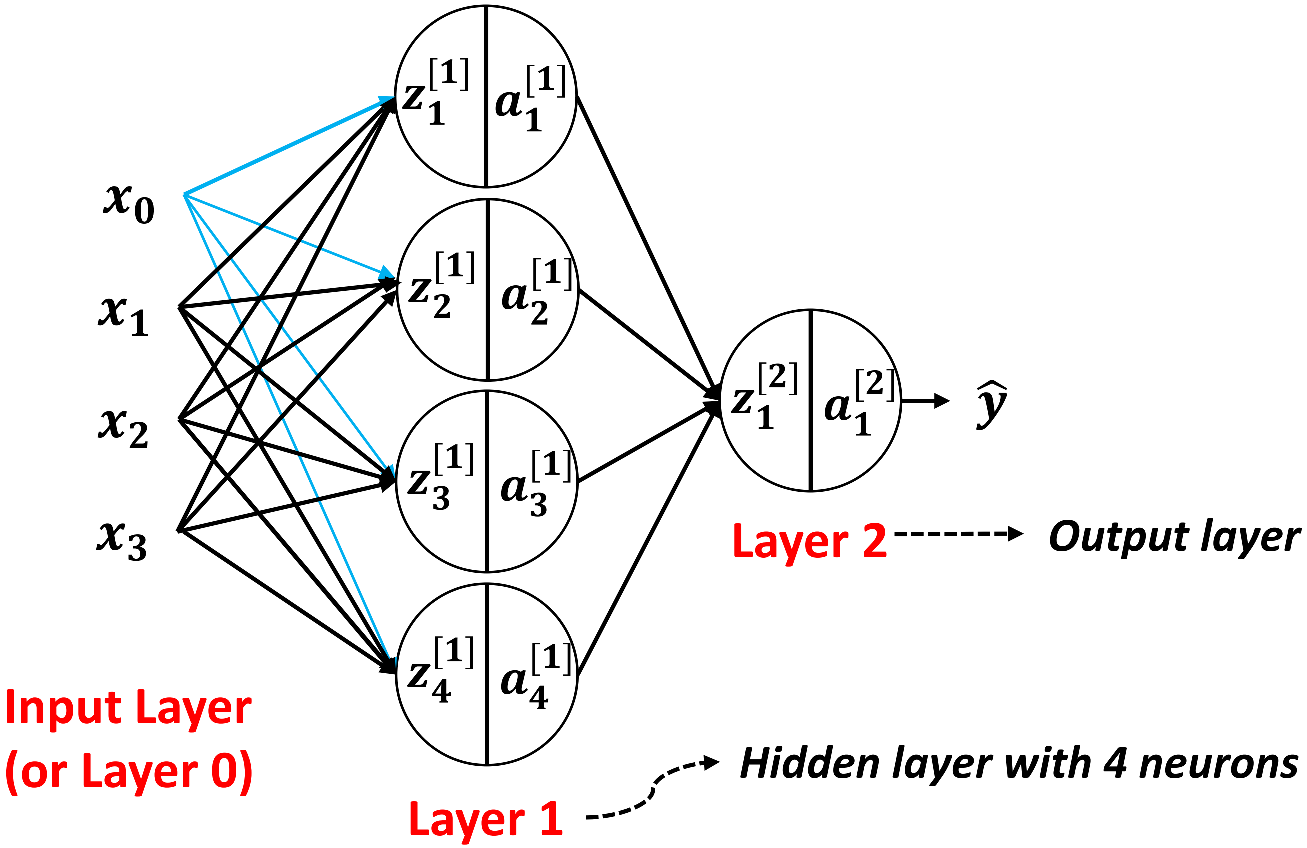
- We can construct a network of neurons (i.e., a neural network) with as many *layers,* and neurons in any layer, as needed



By convention, this neural network is said to have 2 layers (and not 3) since the input layer is typically not counted!

Neural Networks

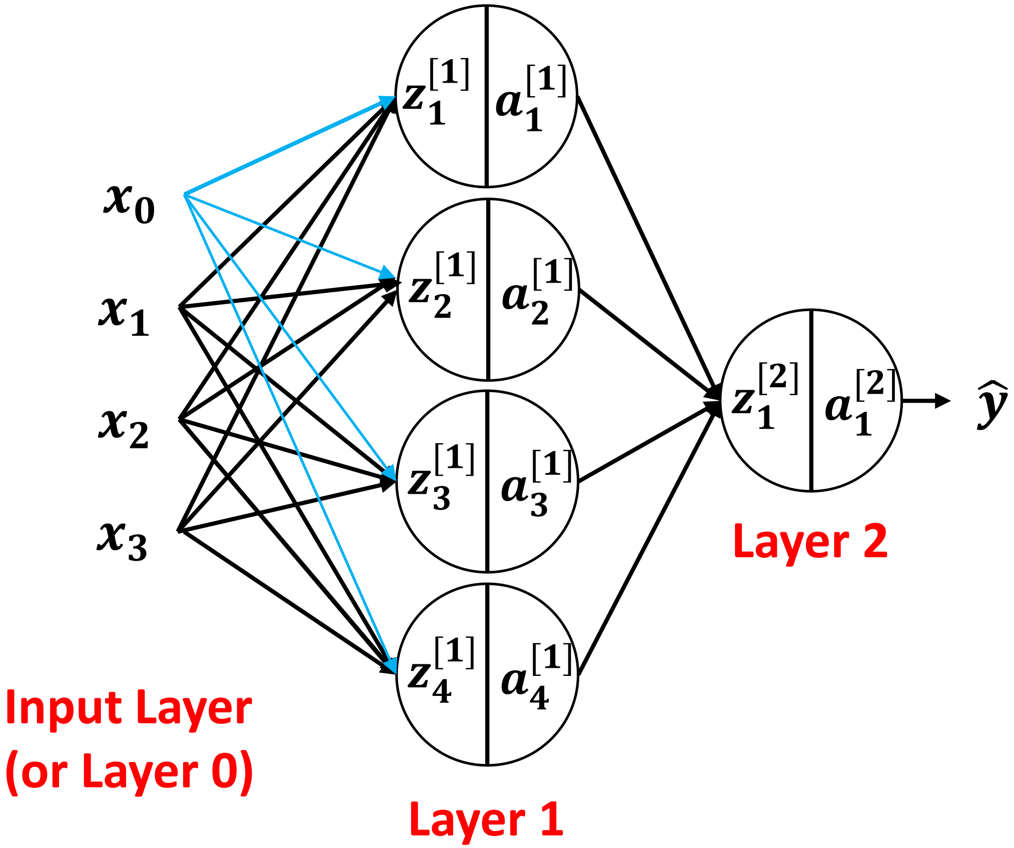
- We can construct a network of neurons (i.e., a neural network) with as many *layers,* and neurons in any layer, as needed



Also, the more layers we add, the ***deeper*** the neural network becomes, giving rise to the concept of ***deep learning***!

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved

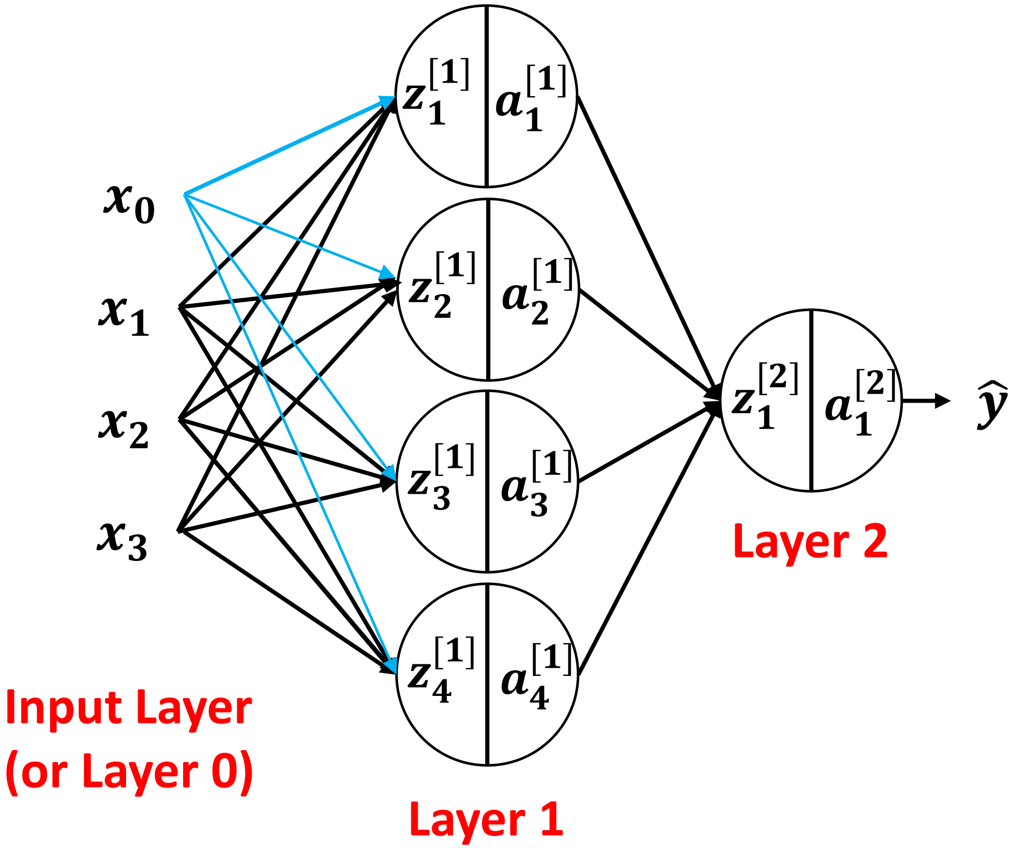


$$\boldsymbol{z}^{[1]} = \boldsymbol{w}^{[1]^T}\boldsymbol{x} + \boldsymbol{b}^{[1]}$$

$$= \begin{bmatrix} w_{11}^{[1]} & w_{21}^{[1]} & w_{31}^{[1]} \\ w_{12}^{[1]} & w_{22}^{[1]} & w_{32}^{[1]} \\ w_{13}^{[1]} & w_{23}^{[1]} & w_{33}^{[1]} \\ w_{14}^{[1]} & w_{24}^{[1]} & w_{34}^{[1]} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} \boldsymbol{b}_1^{[1]} \\ \boldsymbol{b}_2^{[1]} \\ \boldsymbol{b}_3^{[1]} \\ \boldsymbol{b}_4^{[1]} \end{bmatrix}$$

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



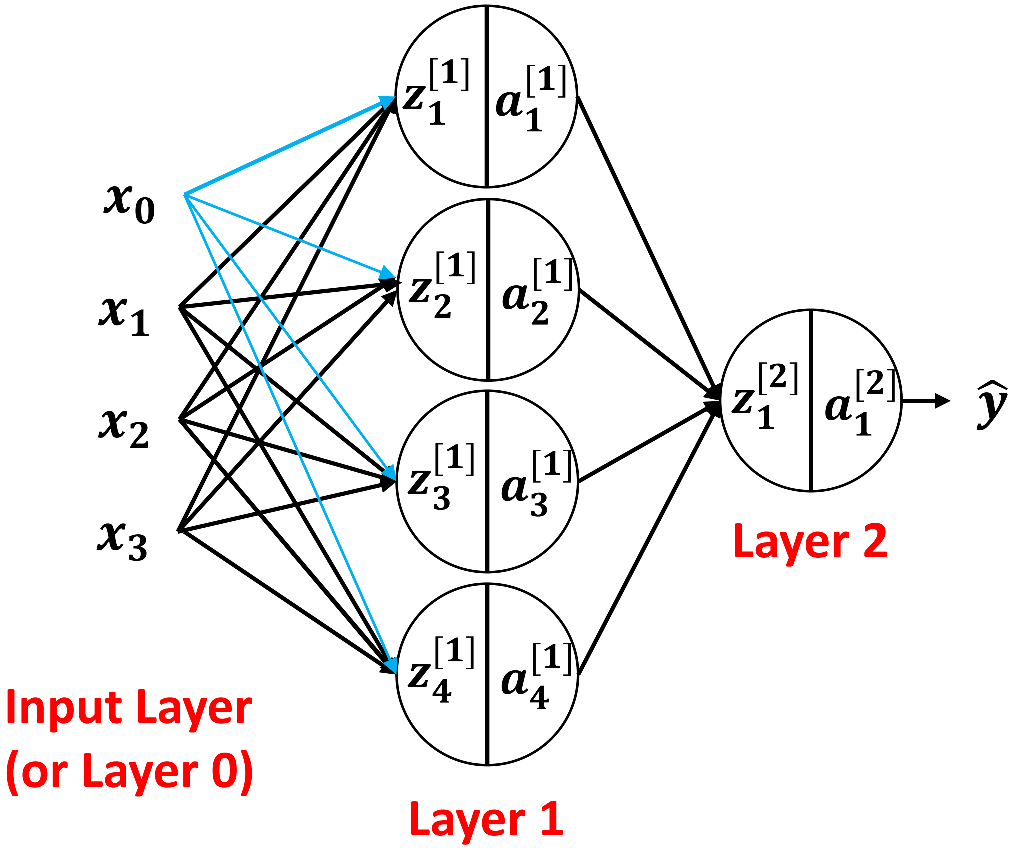
$$\boldsymbol{z}^{[1]} = \boldsymbol{w}^{[1]^T}\boldsymbol{x} + \boldsymbol{b}^{[1]}$$

$$= \begin{bmatrix} w_{11}^{[1]}x_1 + w_{21}^{[1]}x_2 + w_{31}^{[1]}x_3 + b_1^{[1]} \\ w_{12}^{[1]}x_1 + w_{22}^{[1]}x_2 + w_{32}^{[1]}x_3 + b_2^{[1]} \\ w_{13}^{[1]}x_1 + w_{23}^{[1]}x_2 + w_{33}^{[1]}x_3 + b_3^{[1]} \\ w_{14}^{[1]}x_1 + w_{24}^{[1]}x_2 + w_{34}^{[1]}x_3 + b_4^{[1]} \end{bmatrix}$$

$$\boldsymbol{a}^{[1]} = \boldsymbol{\sigma}(\boldsymbol{z}^{[1]})$$

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$\boldsymbol{z}^{[1]} = \boldsymbol{w}^{[1]^T}\boldsymbol{x} + \boldsymbol{b}^{[1]}$$

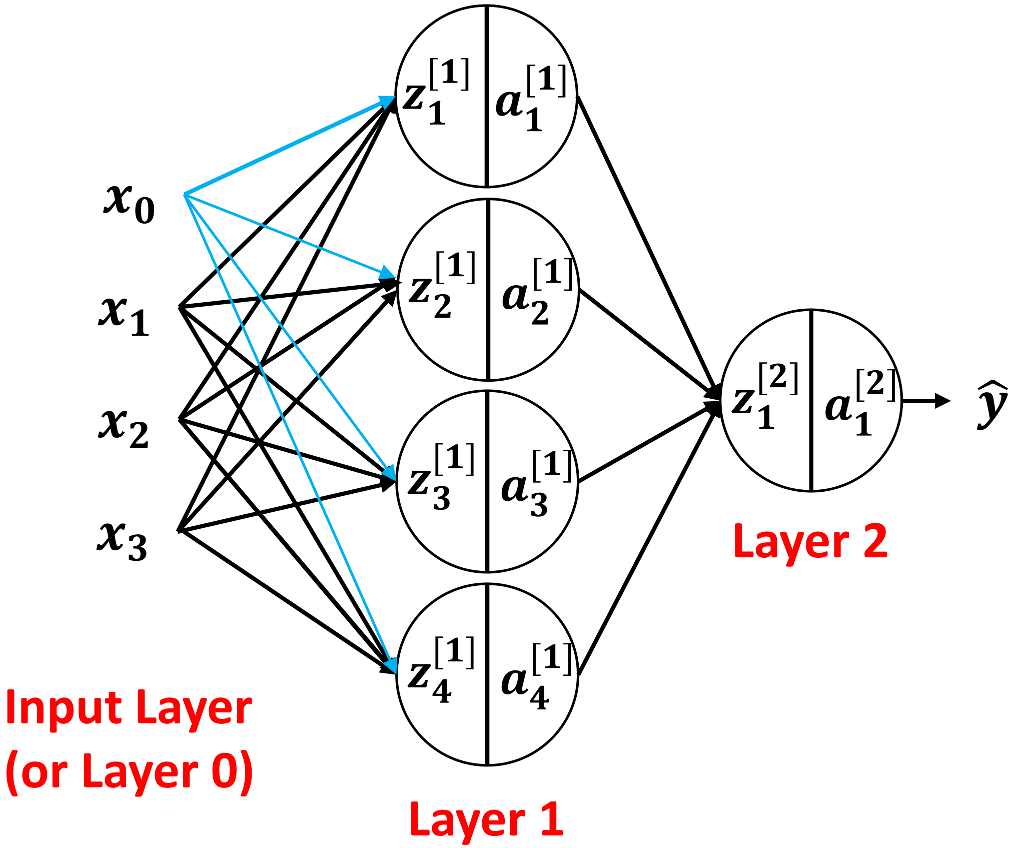
$$= \begin{bmatrix} \boxed{w_{11}^{[1]}x_1 + w_{21}^{[1]}x_2 + w_{31}^{[1]}x_3 + b_1^{[1]}} \\ w_{12}^{[1]}x_1 + w_{22}^{[1]}x_2 + w_{32}^{[1]}x_3 + b_2^{[1]} \\ w_{13}^{[1]}x_1 + w_{23}^{[1]}x_2 + w_{33}^{[1]}x_3 + b_3^{[1]} \\ w_{14}^{[1]}x_1 + w_{24}^{[1]}x_2 + w_{34}^{[1]}x_3 + b_4^{[1]} \end{bmatrix}$$

$$\boldsymbol{a}^{[1]} = \boldsymbol{\sigma}(\boldsymbol{z}^{[1]})$$

$\boldsymbol{z}_1^{[1]}$

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved


$\boldsymbol{x_0}$
$\boldsymbol{x_1}$
$\boldsymbol{x_2}$
$\boldsymbol{x_3}$
$\boldsymbol{z_1^{[1]}}$ $\boldsymbol{a_1^{[1]}}$
$\boldsymbol{z_2^{[1]}}$ $\boldsymbol{a_2^{[1]}}$
$\boldsymbol{z_3^{[1]}}$ $\boldsymbol{a_3^{[1]}}$
$\boldsymbol{z_4^{[1]}}$ $\boldsymbol{a_4^{[1]}}$
$\boldsymbol{z_1^{[2]}}$ $\boldsymbol{a_1^{[2]}}$
$\boldsymbol{\hat{y}}$
Layer 2
Input Layer
(or Layer 0)
Layer 1

$$\boldsymbol{z}^{[1]} = \boldsymbol{w}^{[1]^T}\boldsymbol{x} + \boldsymbol{b}^{[1]}$$

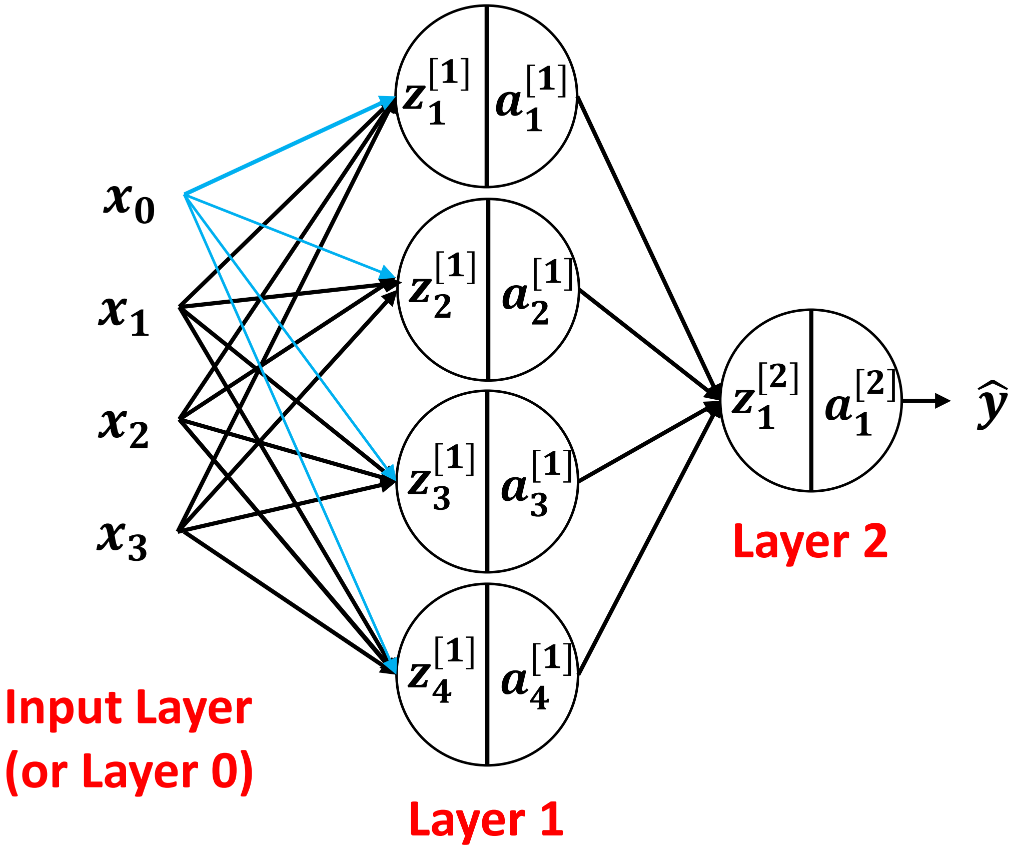
$$= \begin{bmatrix} \boxed{w_{11}^{[1]}x_1 + w_{21}^{[1]}x_2 + w_{31}^{[1]}x_3 + b_1^{[1]}} \\ w_{12}^{[1]}x_1 + w_{22}^{[1]}x_2 + w_{32}^{[1]}x_3 + b_2^{[1]} \\ w_{13}^{[1]}x_1 + w_{23}^{[1]}x_2 + w_{33}^{[1]}x_3 + b_3^{[1]} \\ w_{14}^{[1]}x_1 + w_{24}^{[1]}x_2 + w_{34}^{[1]}x_3 + b_4^{[1]} \end{bmatrix}$$

$$\boldsymbol{a}^{[1]} = \boldsymbol{\sigma}(\boldsymbol{z}^{[1]})$$

$$\boldsymbol{a_1^{[1]}} = \boldsymbol{\sigma}(\boldsymbol{z_1^{[1]}})$$

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$\boldsymbol{z}^{[1]} = \boldsymbol{w}^{[1]^T}\boldsymbol{x} + \boldsymbol{b}^{[1]}$$

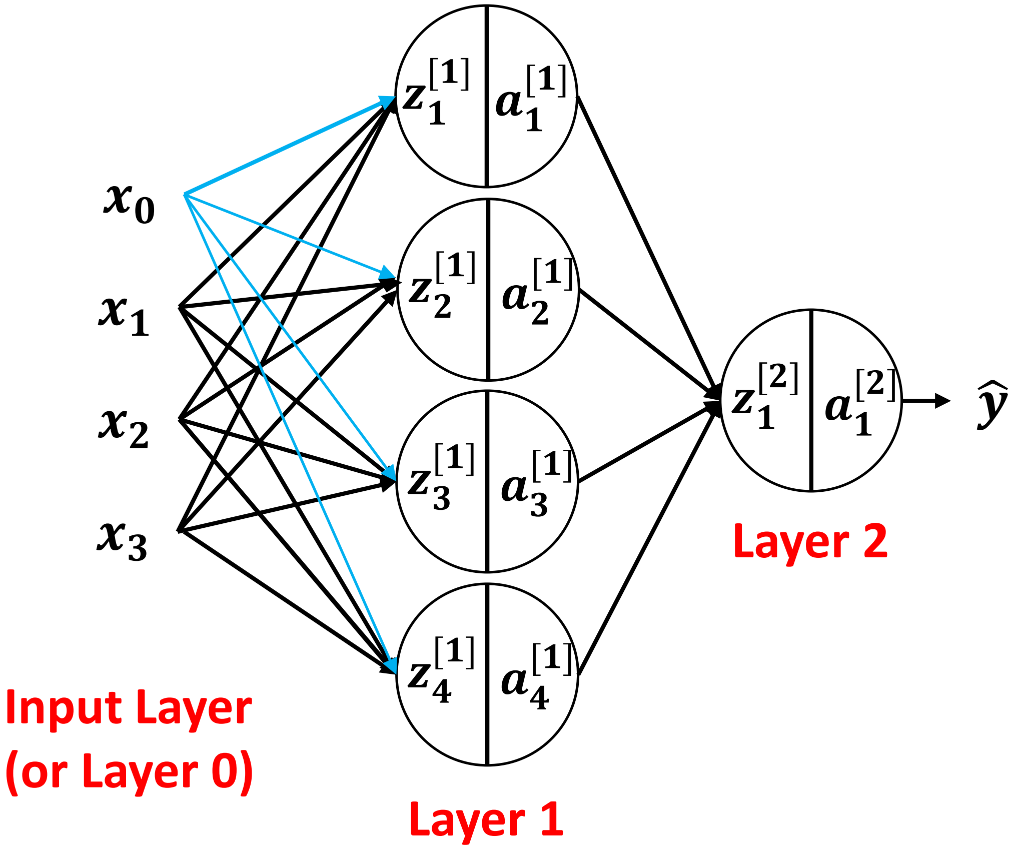
$$= \begin{bmatrix} w_{11}^{[1]}x_1 + w_{21}^{[1]}x_2 + w_{31}^{[1]}x_3 + b_1^{[1]} \\ \boxed{w_{12}^{[1]}x_1 + w_{22}^{[1]}x_2 + w_{32}^{[1]}x_3 + b_2^{[1]}} \\ w_{13}^{[1]}x_1 + w_{23}^{[1]}x_2 + w_{33}^{[1]}x_3 + b_3^{[1]} \\ w_{14}^{[1]}x_1 + w_{24}^{[1]}x_2 + w_{34}^{[1]}x_3 + b_4^{[1]} \end{bmatrix}$$

$$\boldsymbol{a}^{[1]} = \boldsymbol{\sigma}(\boldsymbol{z}^{[1]})$$

$\boldsymbol{z}_2^{[1]}$

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$\boldsymbol{z}^{[1]} = \boldsymbol{w}^{[1]^T}\boldsymbol{x} + \boldsymbol{b}^{[1]}$$

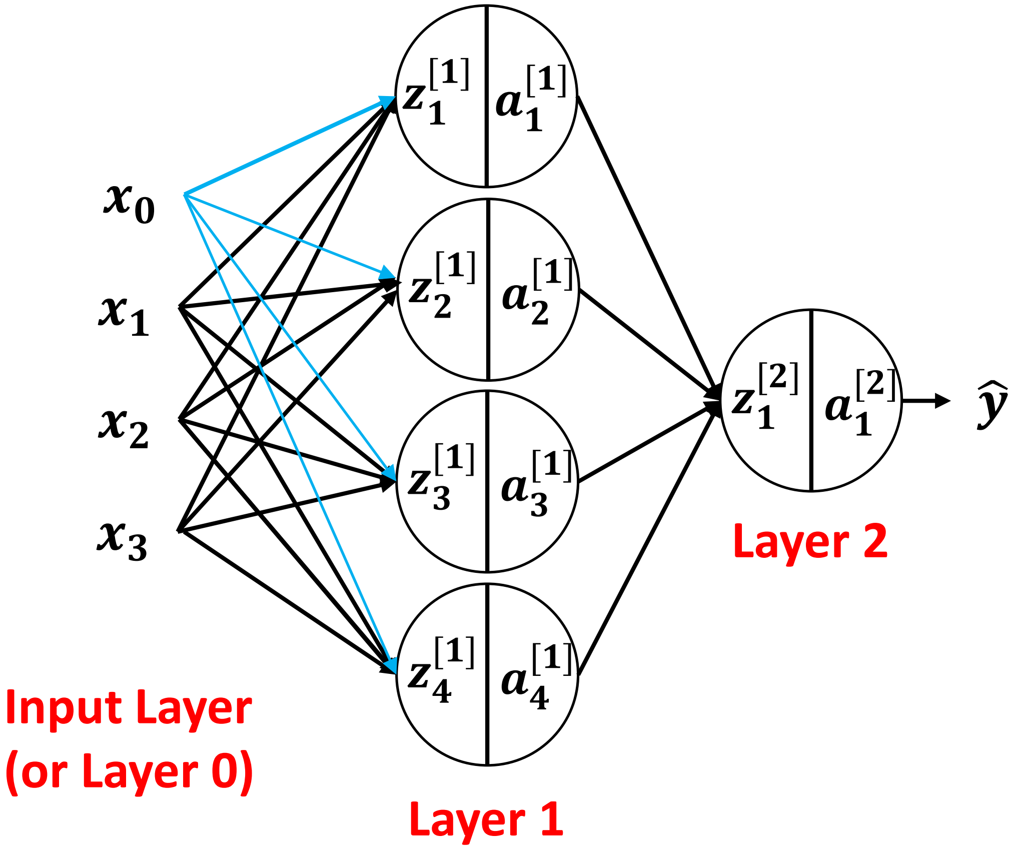
$$= \begin{bmatrix} w_{11}^{[1]}x_1 + w_{21}^{[1]}x_2 + w_{31}^{[1]}x_3 + b_1^{[1]} \\ \boxed{w_{12}^{[1]}x_1 + w_{22}^{[1]}x_2 + w_{32}^{[1]}x_3 + b_2^{[1]}} \\ w_{13}^{[1]}x_1 + w_{23}^{[1]}x_2 + w_{33}^{[1]}x_3 + b_3^{[1]} \\ w_{14}^{[1]}x_1 + w_{24}^{[1]}x_2 + w_{34}^{[1]}x_3 + b_4^{[1]} \end{bmatrix}$$

$$\boldsymbol{a}^{[1]} = \boldsymbol{\sigma}(\boldsymbol{z}^{[1]})$$

$$\boldsymbol{a}_2^{[1]} = \boldsymbol{\sigma}(\boldsymbol{z}_2^{[1]})$$

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$\boldsymbol{x}_0$
$\boldsymbol{x}_1$
$\boldsymbol{x}_2$
$\boldsymbol{x}_3$
$\boldsymbol{z}_1^{[1]}$ $\boldsymbol{a}_1^{[1]}$
$\boldsymbol{z}_2^{[1]}$ $\boldsymbol{a}_2^{[1]}$
$\boldsymbol{z}_3^{[1]}$ $\boldsymbol{a}_3^{[1]}$
$\boldsymbol{z}_4^{[1]}$ $\boldsymbol{a}_4^{[1]}$
$\boldsymbol{z}_1^{[2]}$ $\boldsymbol{a}_1^{[2]}$
$\hat{\boldsymbol{y}}$
Layer 2
Input Layer
(or Layer 0)
Layer 1

$$\boldsymbol{z}^{[1]} = \boldsymbol{w}^{[1]^{T}}\boldsymbol{x} + \boldsymbol{b}^{[1]}$$

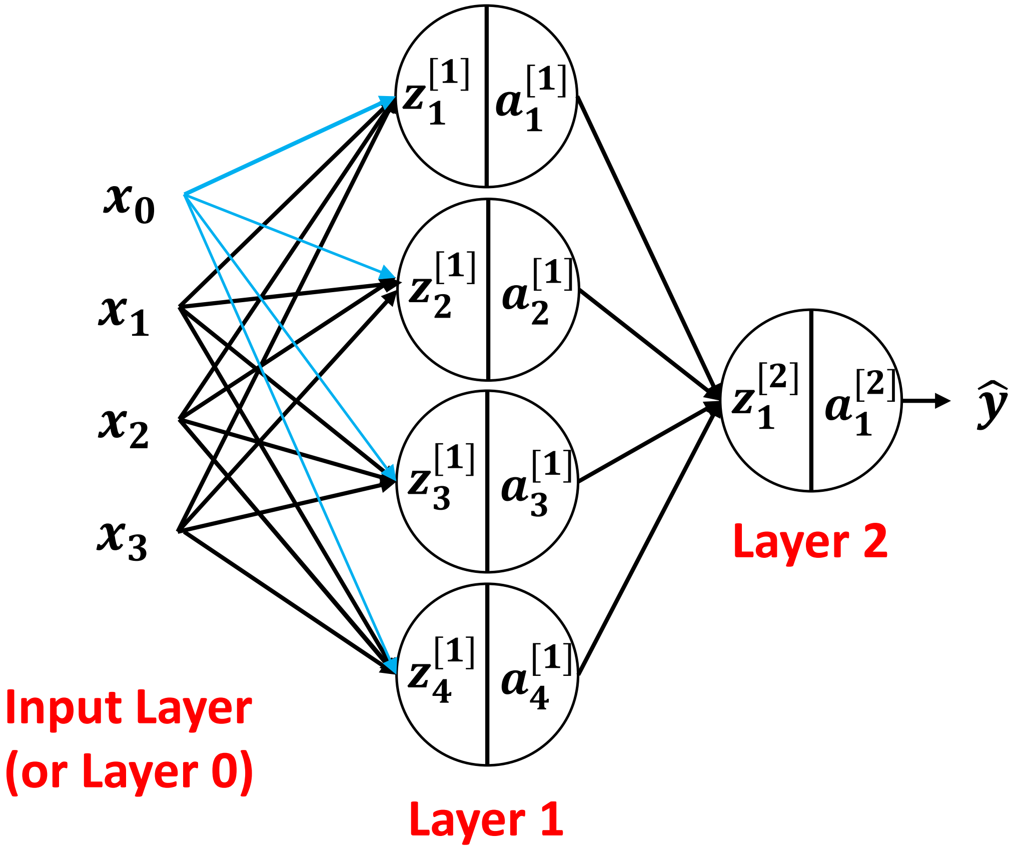
$$= \begin{bmatrix} w_{11}^{[1]}x_1 + w_{21}^{[1]}x_2 + w_{31}^{[1]}x_3 + b_1^{[1]} \\ w_{12}^{[1]}x_1 + w_{22}^{[1]}x_2 + w_{32}^{[1]}x_3 + b_2^{[1]} \\ \boxed{w_{13}^{[1]}x_1 + w_{23}^{[1]}x_2 + w_{33}^{[1]}x_3 + b_3^{[1]}} \\ w_{14}^{[1]}x_1 + w_{24}^{[1]}x_2 + w_{34}^{[1]}x_3 + b_4^{[1]} \end{bmatrix}$$

$$\boldsymbol{a}^{[1]} = \boldsymbol{\sigma}(\boldsymbol{z}^{[1]})$$

$\boldsymbol{z}_3^{[1]}$

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$\boldsymbol{z}^{[1]} = \boldsymbol{w}^{[1]^{T}}\boldsymbol{x} + \boldsymbol{b}^{[1]}$$

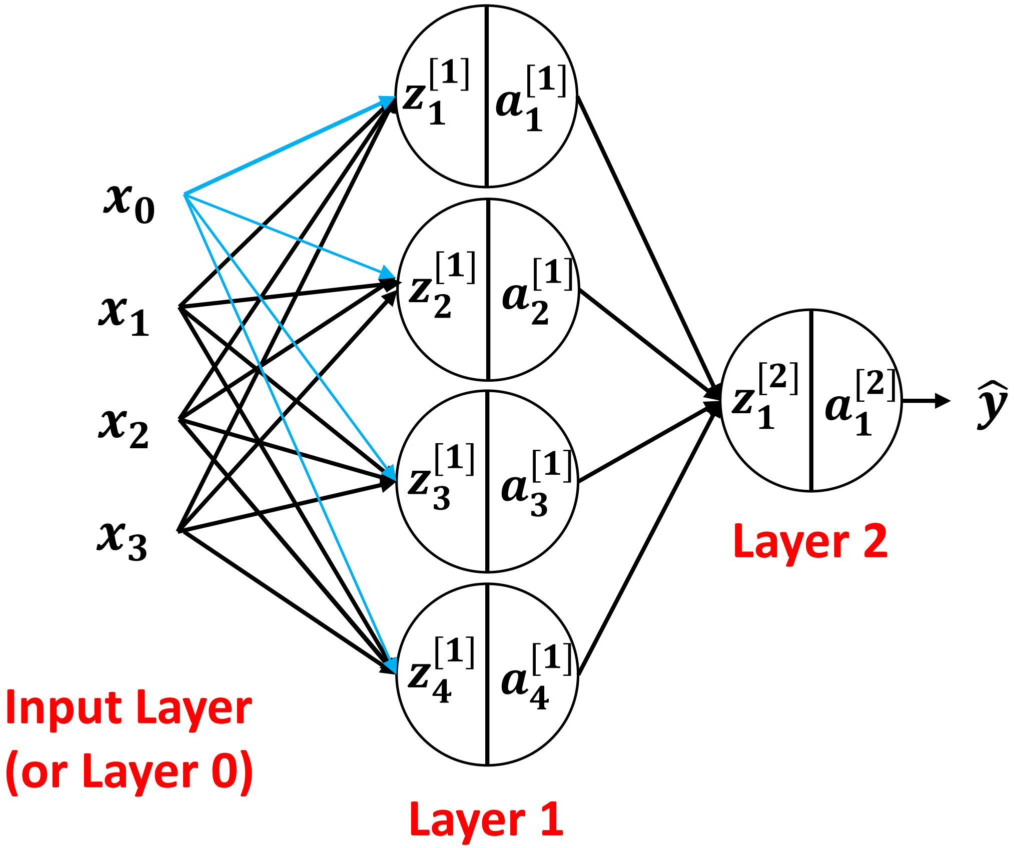
$$= \begin{bmatrix} w_{11}^{[1]}x_1 + w_{21}^{[1]}x_2 + w_{31}^{[1]}x_3 + b_1^{[1]} \\ w_{12}^{[1]}x_1 + w_{22}^{[1]}x_2 + w_{32}^{[1]}x_3 + b_2^{[1]} \\ \boxed{w_{13}^{[1]}x_1 + w_{23}^{[1]}x_2 + w_{33}^{[1]}x_3 + b_3^{[1]}} \\ w_{14}^{[1]}x_1 + w_{24}^{[1]}x_2 + w_{34}^{[1]}x_3 + b_4^{[1]} \end{bmatrix}$$

$$\boldsymbol{a}^{[1]} = \boldsymbol{\sigma}(\boldsymbol{z}^{[1]})$$

$$\boldsymbol{a}_3^{[1]} = \boldsymbol{\sigma}(\boldsymbol{z}_3^{[1]})$$

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$\boldsymbol{z}^{[1]} = \boldsymbol{w}^{[1]^T}\boldsymbol{x} + \boldsymbol{b}^{[1]}$$

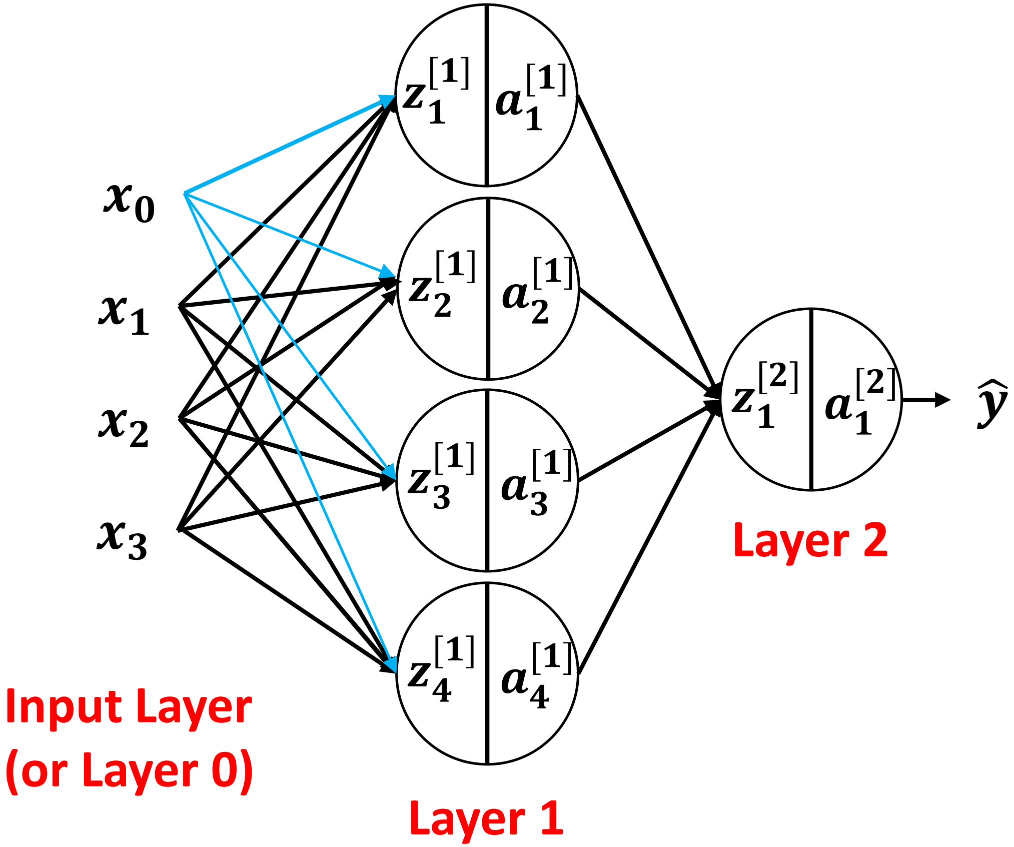
$$= \begin{bmatrix} w_{11}^{[1]}x_1 + w_{21}^{[1]}x_2 + w_{31}^{[1]}x_3 + b_1^{[1]} \\ w_{12}^{[1]}x_1 + w_{22}^{[1]}x_2 + w_{32}^{[1]}x_3 + b_2^{[1]} \\ w_{13}^{[1]}x_1 + w_{23}^{[1]}x_2 + w_{33}^{[1]}x_3 + b_3^{[1]} \\ \boxed{w_{14}^{[1]}x_1 + w_{24}^{[1]}x_2 + w_{34}^{[1]}x_3 + b_4^{[1]}} \end{bmatrix}$$

$$\boldsymbol{a}^{[1]} = \boldsymbol{\sigma}(\boldsymbol{z}^{[1]})$$

$$\boldsymbol{z}_4^{[1]}$$

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$\boldsymbol{z}^{[1]} = \boldsymbol{w}^{[1]^T}\boldsymbol{x} + \boldsymbol{b}^{[1]}$$

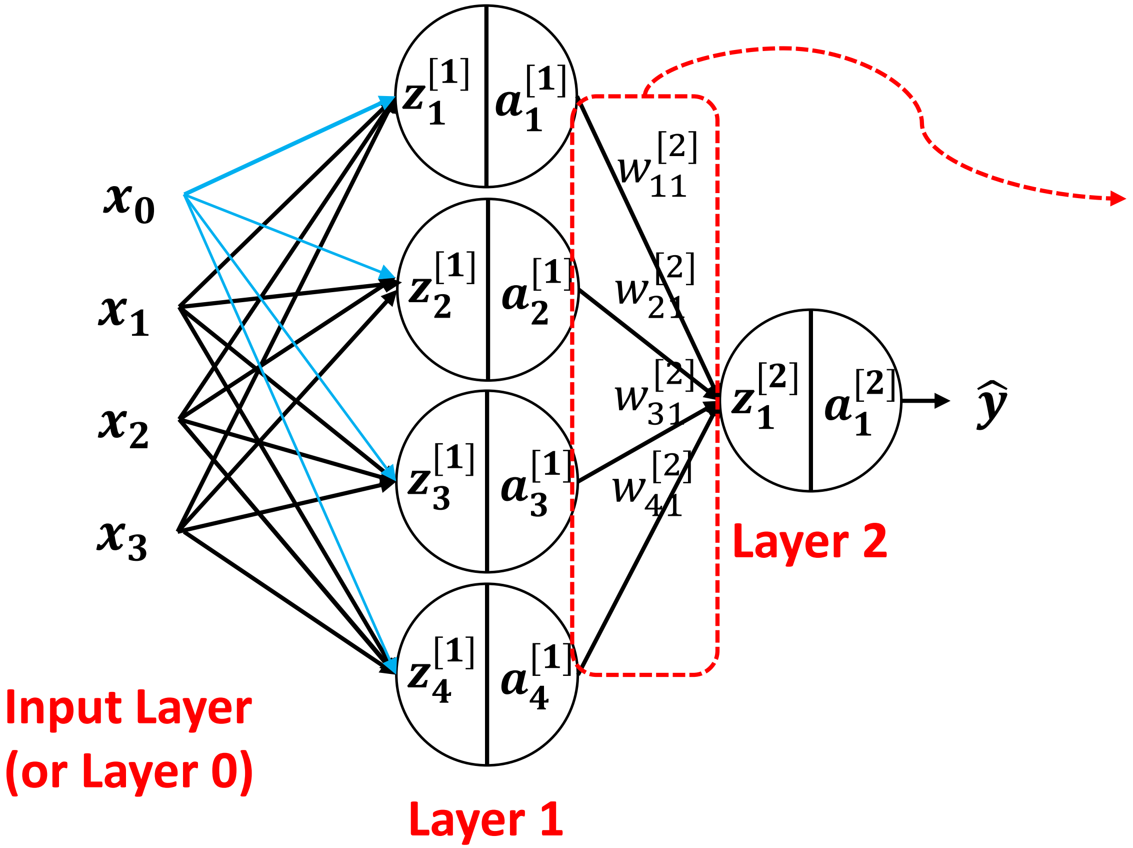
$$= \begin{bmatrix} w_{11}^{[1]}x_1 + w_{21}^{[1]}x_2 + w_{31}^{[1]}x_3 + b_1^{[1]} \\ w_{12}^{[1]}x_1 + w_{22}^{[1]}x_2 + w_{32}^{[1]}x_3 + b_2^{[1]} \\ w_{13}^{[1]}x_1 + w_{23}^{[1]}x_2 + w_{33}^{[1]}x_3 + b_3^{[1]} \\ \boxed{w_{14}^{[1]}x_1 + w_{24}^{[1]}x_2 + w_{34}^{[1]}x_3 + b_4^{[1]}} \end{bmatrix}$$

$$\boldsymbol{a}^{[1]} = \boldsymbol{\sigma}(\boldsymbol{z}^{[1]})$$

$$\boldsymbol{a}_4^{[1]} = \boldsymbol{\sigma}(\boldsymbol{z}_4^{[1]})$$

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved

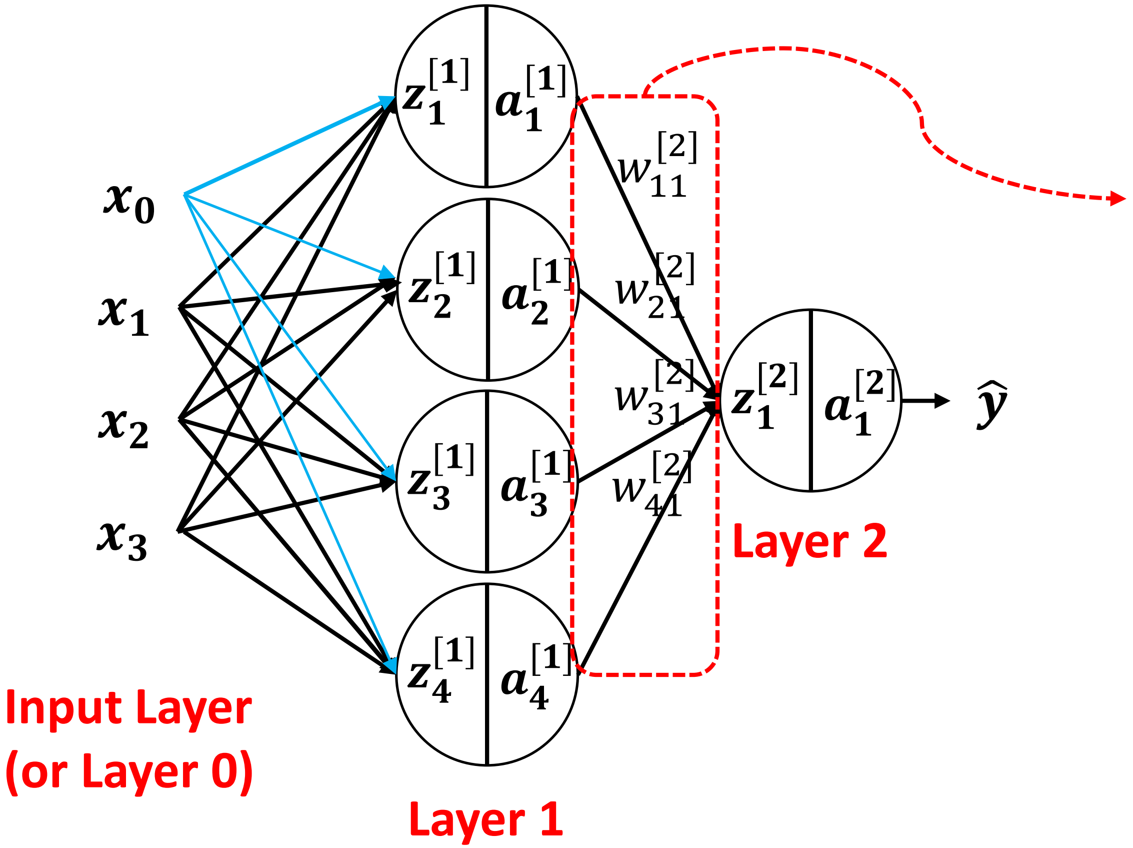


$$\boldsymbol{w}^{[2]} = \begin{bmatrix} w_{11}^{[2]} \\ w_{21}^{[2]} \\ w_{31}^{[2]} \\ w_{41}^{[2]} \end{bmatrix}$$

Dimension of $\boldsymbol{w}^{[2]}$= (4, 1)

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved

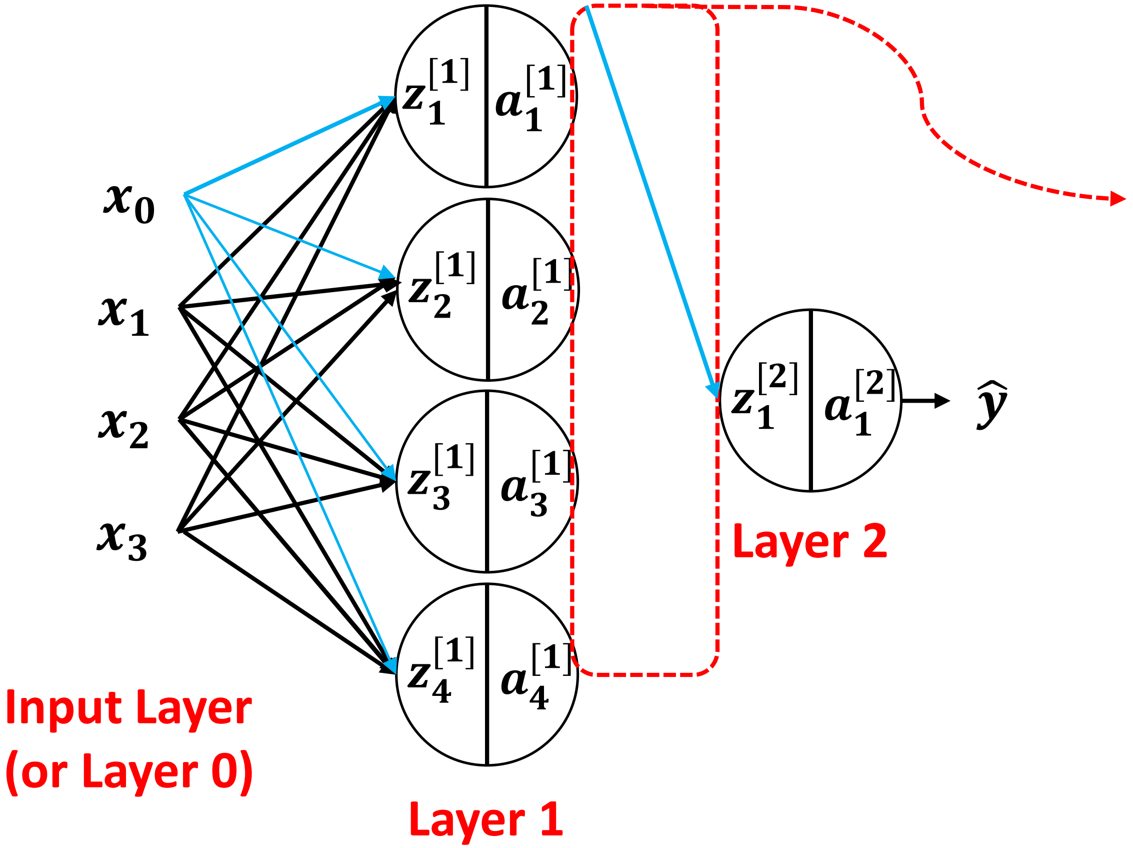


$$\boldsymbol{w}^{[2]^{T}} = \left[w_{11}^{[2]}\ w_{21}^{[2]}\ w_{31}^{[2]}\ w_{41}^{[2]} \right]$$

Dimension of $\boldsymbol{w}^{[2]^{T}}$ = (1, 4)

Vectorizing Input and All Variables

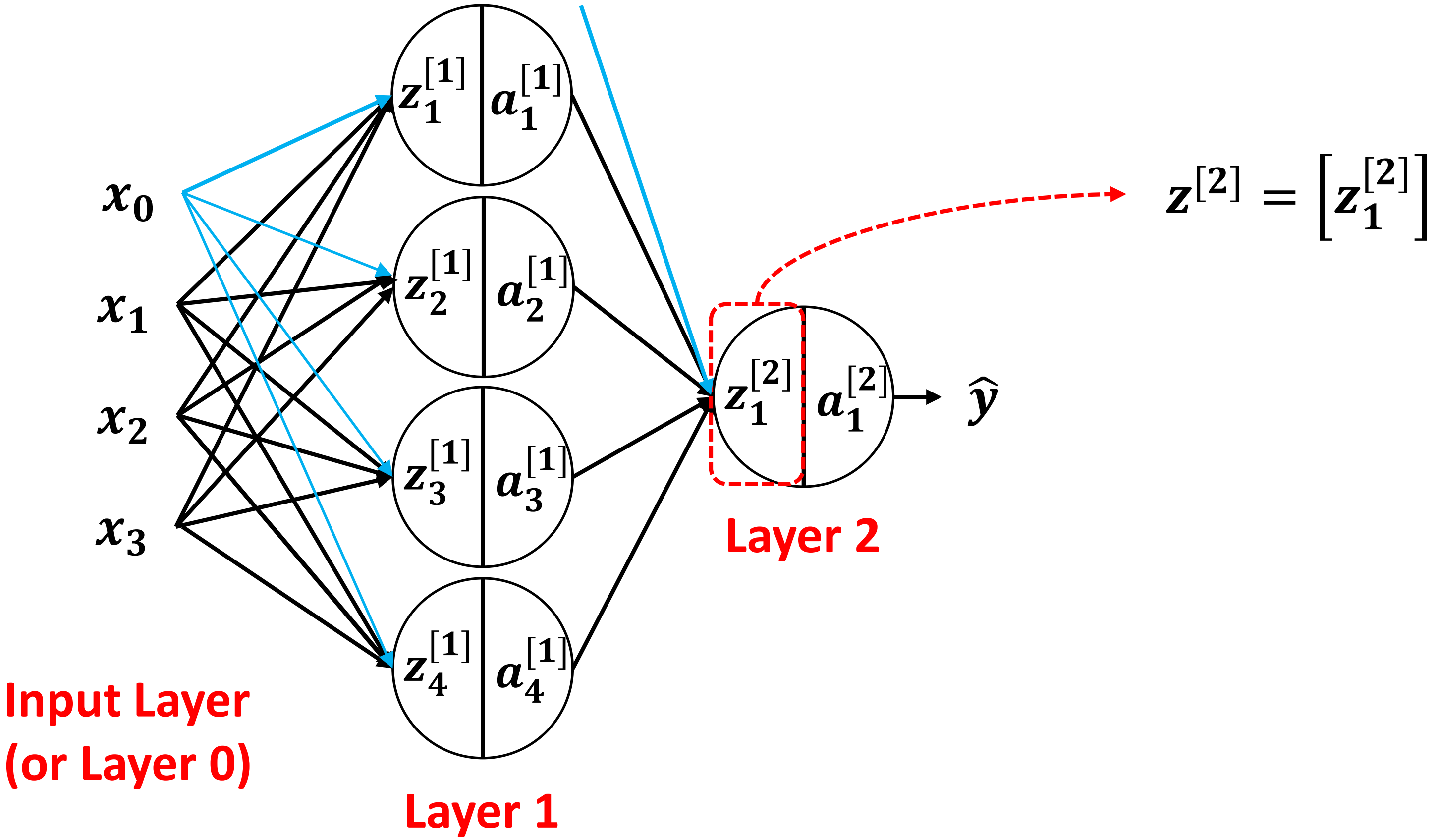
- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$\boldsymbol{b}^{[2]} = \left[\boldsymbol{b}_1^{[2]}\right]$$

Vectorizing Input and All Variables

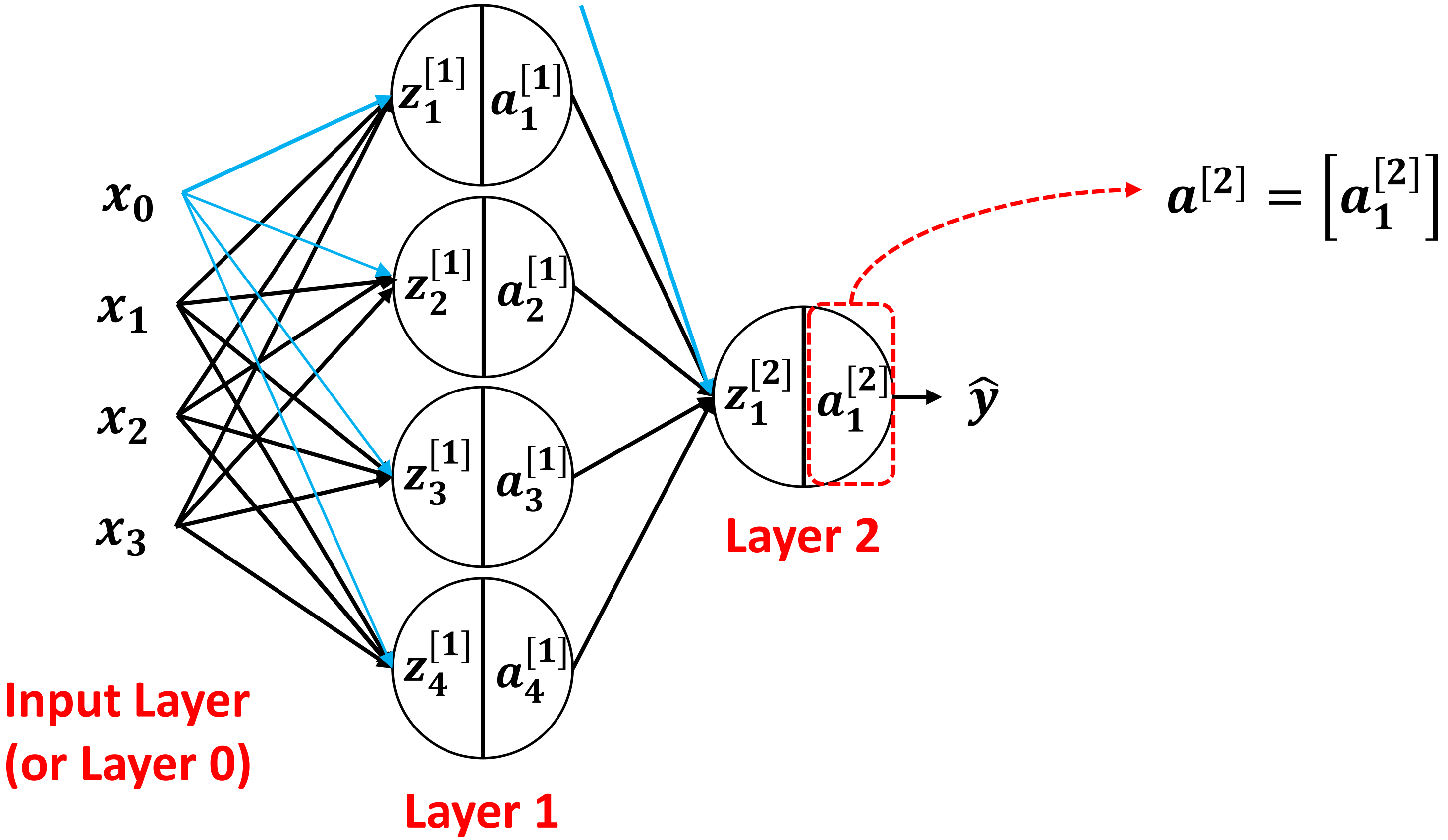
- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$\boldsymbol{z}^{[2]} = \left[z_1^{[2]} \right]$$

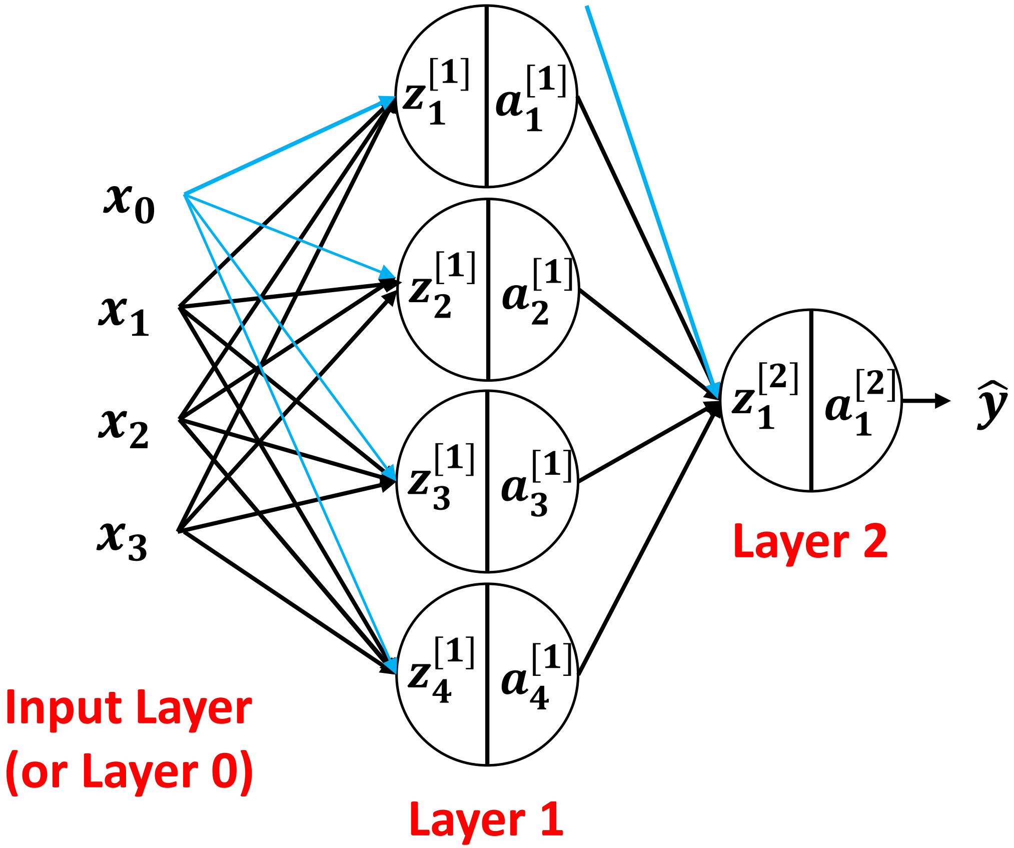
Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



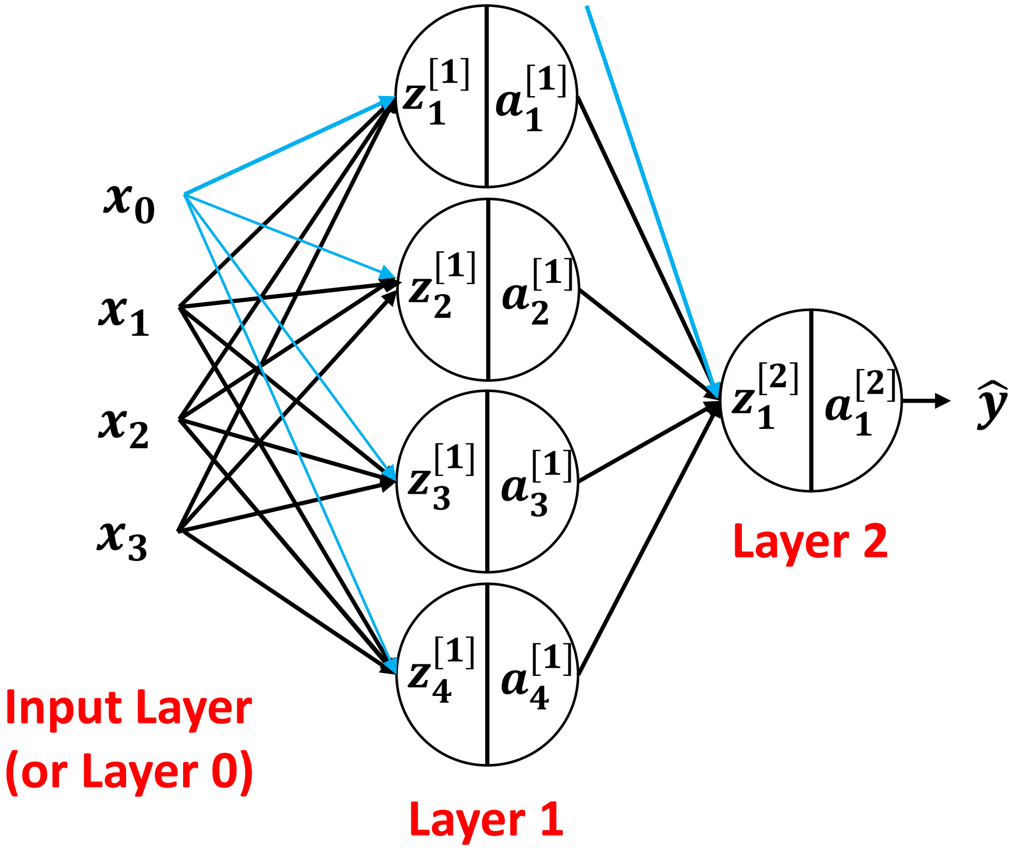
$$\boldsymbol{z}^{[2]} = \boldsymbol{w}^{[2]^T}\boldsymbol{x} + \boldsymbol{b}^{[2]}$$

$$= \boldsymbol{w}^{[2]^T}\boldsymbol{a}^{[1]} + \boldsymbol{b}^{[2]}$$

$$= \begin{bmatrix} w_{11}^{[2]} & w_{21}^{[2]} & w_{31}^{[2]} & w_{41}^{[2]} \end{bmatrix} \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix} + \begin{bmatrix} b_1^{[2]} \end{bmatrix}$$

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



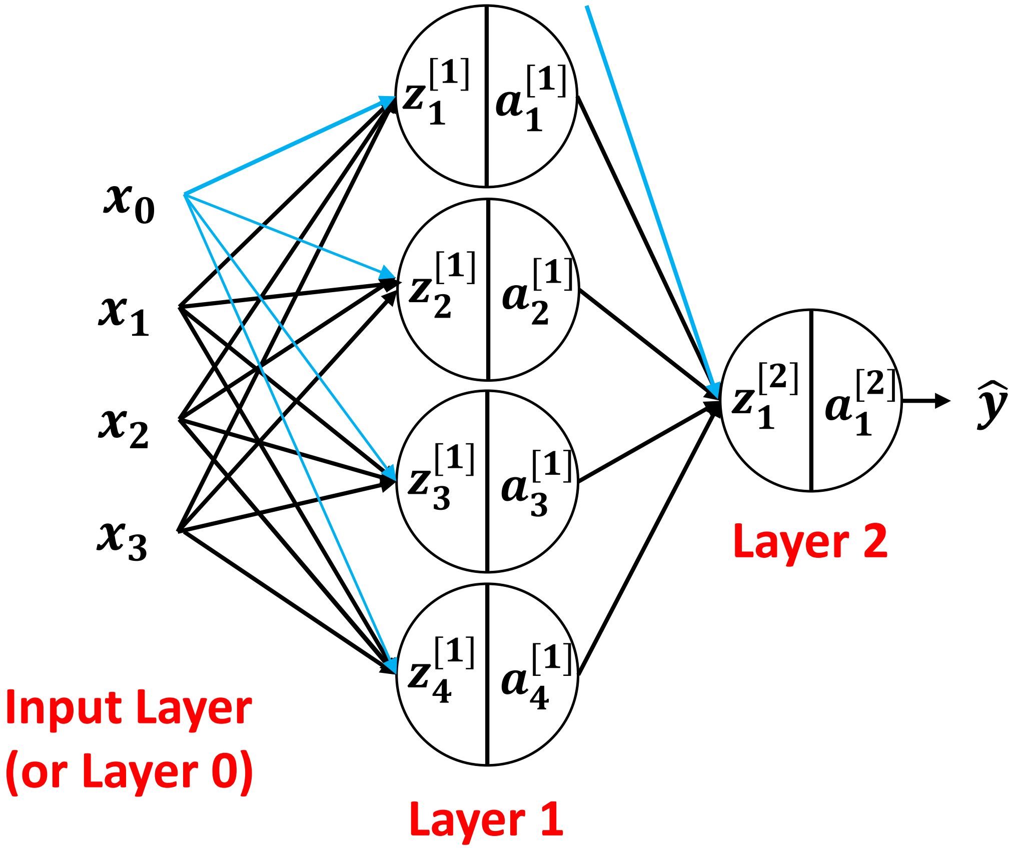
$$\boldsymbol{z}^{[2]} = \boldsymbol{w}^{[2]^T}\boldsymbol{x} + \boldsymbol{b}^{[2]}$$

$$= \boldsymbol{w}^{[2]^T}\boldsymbol{a}^{[1]} + \boldsymbol{b}^{[2]}$$

$$= \left[w_{11}^{[2]}a_1^{[1]} + w_{21}^{[2]}a_2^{[1]} + w_{31}^{[2]}a_3^{[1]} + w_{41}^{[2]}a_4^{[1]}\right] + \left[b_1^{[2]}\right]$$

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$\boldsymbol{z}^{[2]} = \boldsymbol{w}^{[2]^T}\boldsymbol{x} + \boldsymbol{b}^{[2]}$$

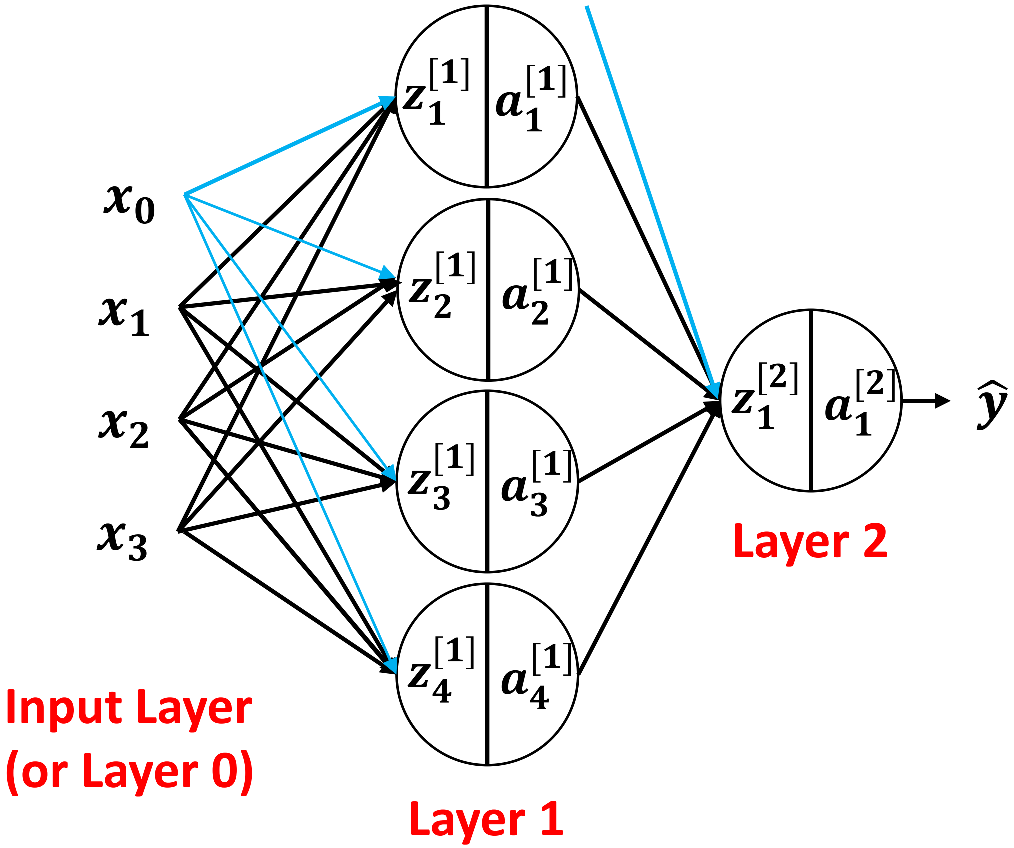
$$= \boldsymbol{w}^{[2]^T}\boldsymbol{a}^{[1]} + \boldsymbol{b}^{[2]}$$

$$= \left[w_{11}^{[2]}a_1^{[1]} + w_{21}^{[2]}a_2^{[1]} + w_{31}^{[2]}a_3^{[1]} + w_{41}^{[2]}a_4^{[1]} + b_1^{[2]}\right]$$

$$\boldsymbol{a}^{[2]} = \boldsymbol{\sigma}(\boldsymbol{z}^{[2]})$$

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$\boldsymbol{z}^{[2]} = \boldsymbol{w}^{[2]^T}\boldsymbol{x} + \boldsymbol{b}^{[2]}$$

$$= \boldsymbol{w}^{[2]^T}\boldsymbol{a}^{[1]} + \boldsymbol{b}^{[2]}$$

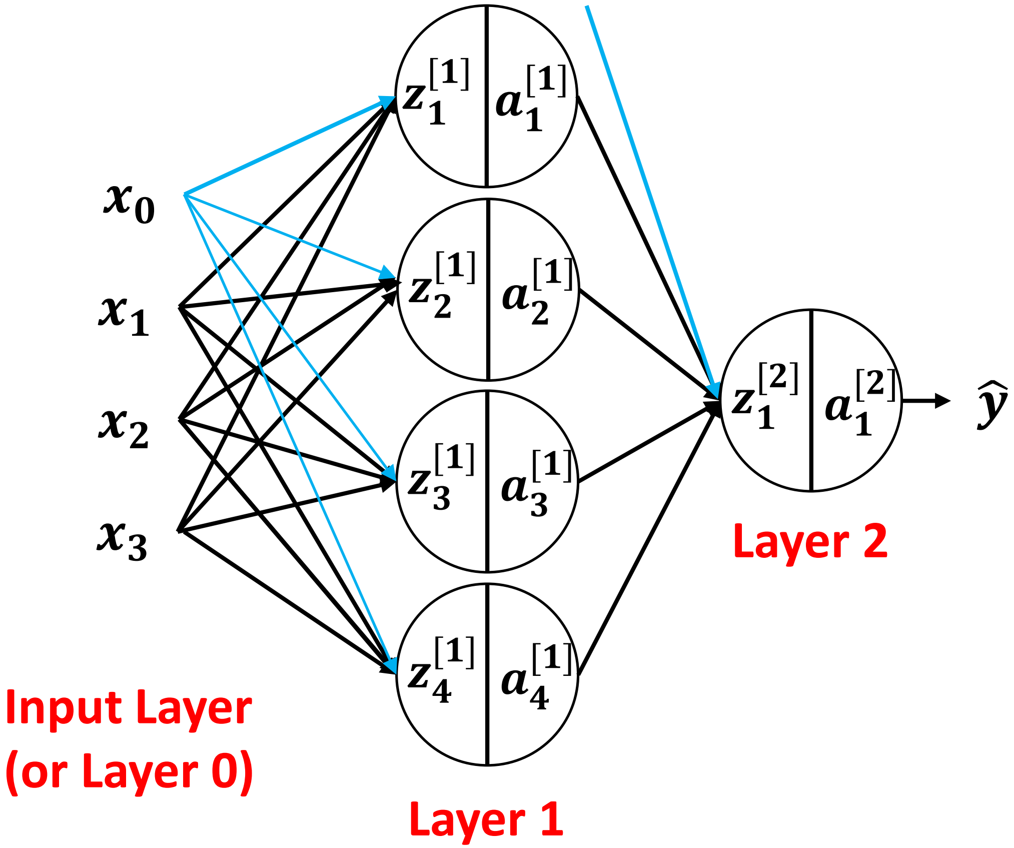
$$= \left[\boxed{w_{11}^{[2]} a_1^{[1]} + w_{21}^{[2]} a_2^{[1]} + w_{31}^{[2]} a_3^{[1]} + w_{41}^{[2]} a_4^{[1]} + b_1^{[2]}}\right]$$

$$\boldsymbol{z}_1^{[2]}$$

$$\boldsymbol{a}^{[2]} = \boldsymbol{\sigma}(\boldsymbol{z}^{[2]})$$

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$\boldsymbol{z}^{[2]} = \boldsymbol{w}^{[2]^T}\boldsymbol{x} + \boldsymbol{b}^{[2]}$$

$$= \boldsymbol{w}^{[2]^T}\boldsymbol{a}^{[1]} + \boldsymbol{b}^{[2]}$$

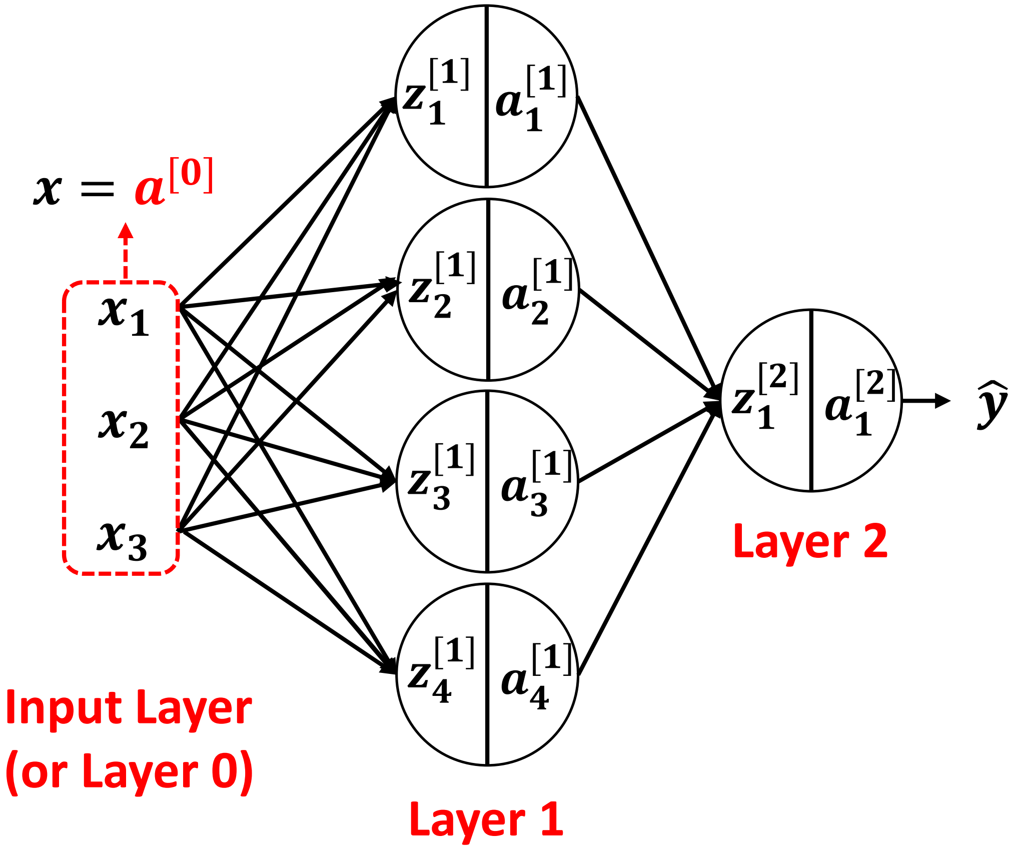
$$= \left[\boxed{w_{11}^{[2]}a_1^{[1]} + w_{21}^{[2]}a_2^{[1]} + w_{31}^{[2]}a_3^{[1]} + w_{41}^{[2]}a_4^{[1]} + b_1^{[2]}}\right]$$

$$\boldsymbol{a}^{[2]} = \boldsymbol{\sigma}(\boldsymbol{z}^{[2]})$$

$$\boldsymbol{a}_1^{[2]} = \boldsymbol{\sigma}(\boldsymbol{z}_1^{[2]})$$

Vectorizing Input and All Variables

- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$z^{[1]} = w^{[1]^T} a^{[0]} + b^{[1]}$$

$$a^{[1]} = \sigma(z^{[1]})$$

$$z^{[2]} = w^{[2]^T} a^{[1]} + b^{[2]}$$

$$a^{[2]} = \sigma(z^{[2]})$$

Neural Network Architectures

Even for a basic Neural Network, there are many design decisions to make:

1. # of hidden layers (depth)
2. # of units per hidden layer (width)
3. Type of activation function (nonlinearity)
4. Form of objective function

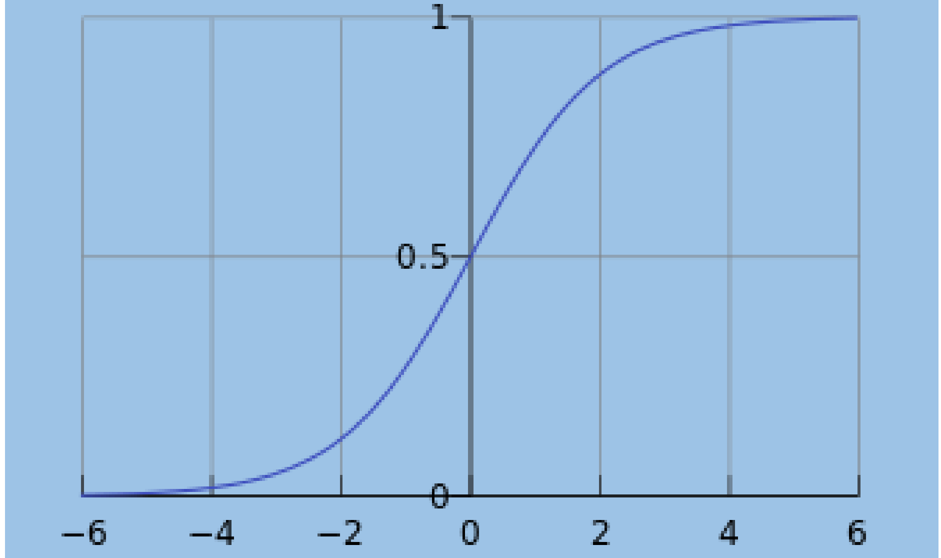
Number of neurons

- Many neurons:
  - Higher accuracy
  - Slower
  - Risk of over-fitting
    - Memorizing, rather than understanding
    - The network will be useless with new problems.
- Few neurons:
  - Lower accuracy
  - Inability to learn at all
- Optimal number.

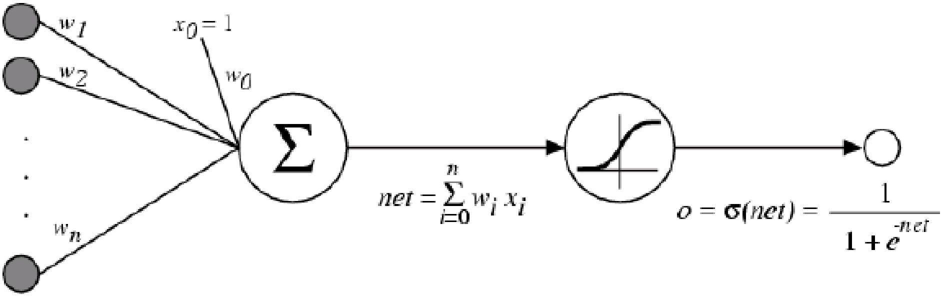
Activation Functions

Sigmoid / Logistic Function

$$\text{logistic}(u) \circ \frac{1}{1+e^{-u}}$$



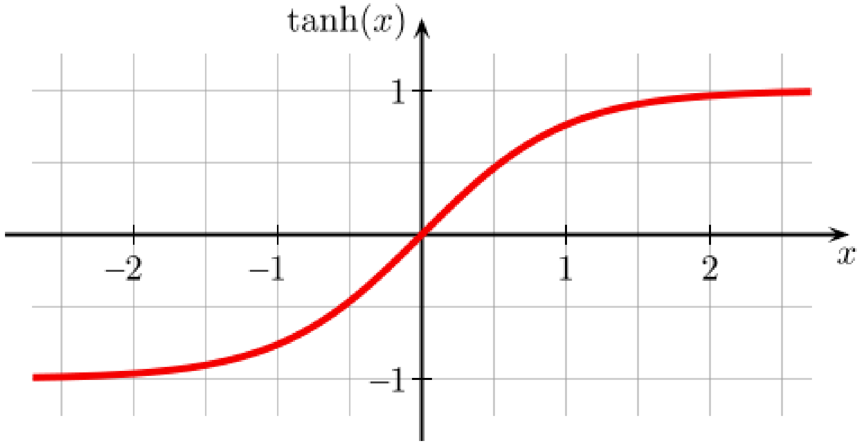
So far, we've assumed that the activation function (nonlinearity) is always the sigmoid function…



w_1
w_2
w_n
$x_0 = 1$
w_0
Σ
$net = \sum_{i=0}^{n} w_i x_i$
$o = \sigma(net) = \frac{1}{1+e^{-net}}$

Activation Functions

- A new change: modifying the nonlinearity
  - The logistic is not widely used in modern ANNs



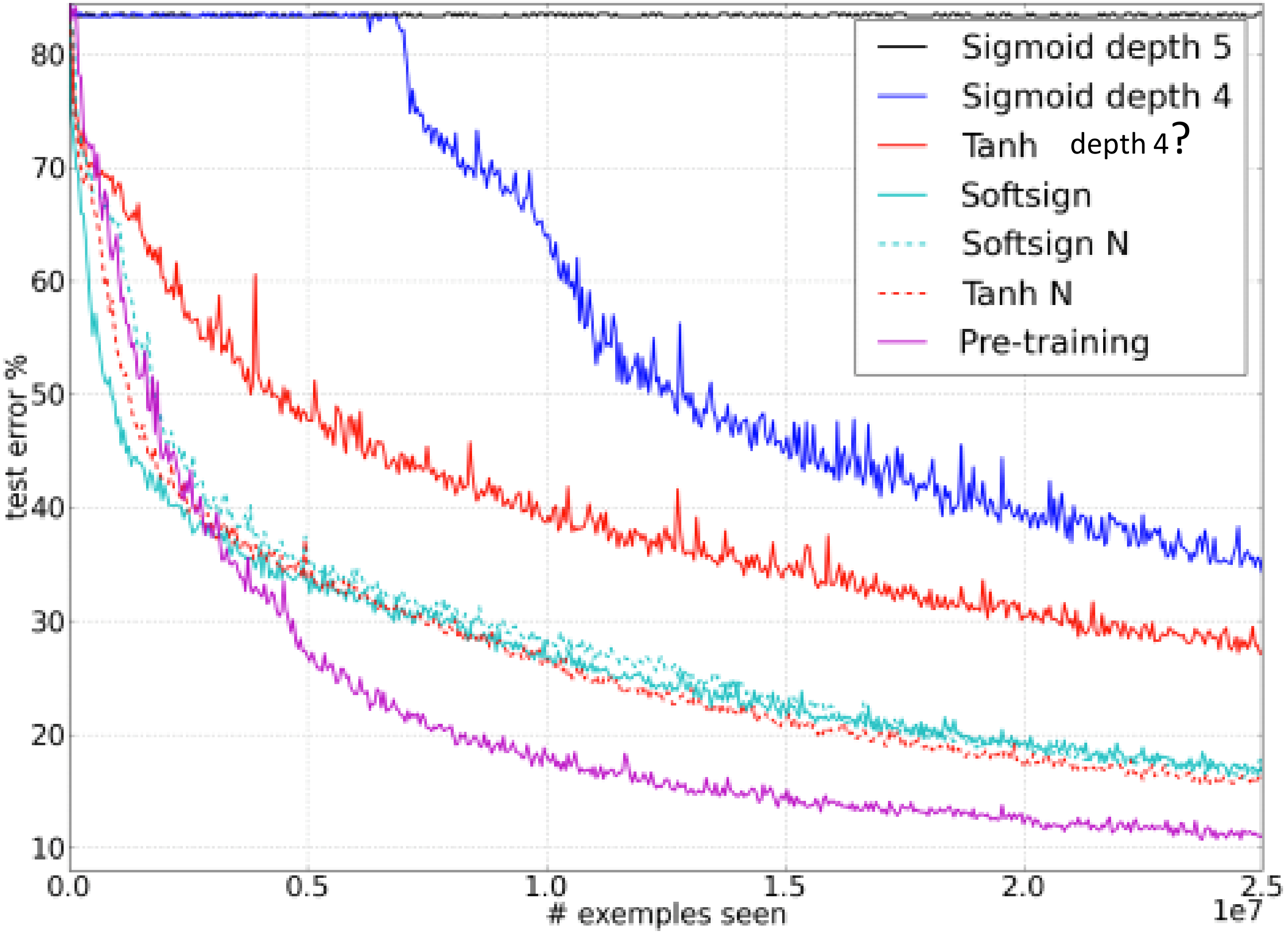
Alternate 1: tanh

Like logistic function but shifted to range [-1, +1]

Slide from William Cohen

Understanding the difficulty of training deep feedforward neural networks

AI Stats 2010

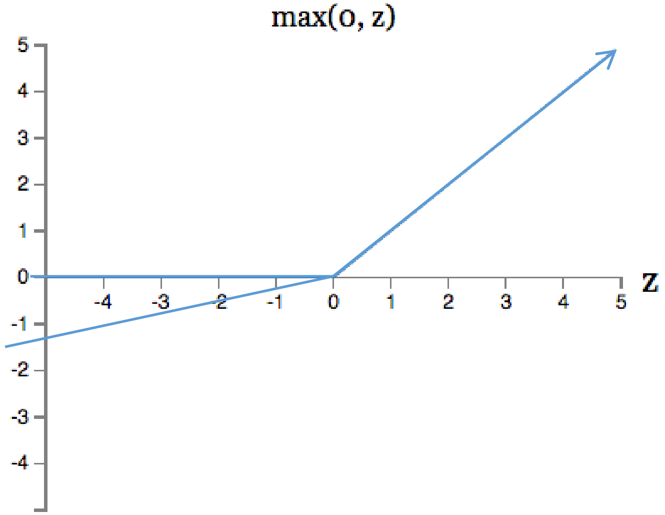


sigmoid
vs.
tanh

Figure from Glorot & Bentio (2010)

Activation Functions

- A new change: modifying the nonlinearity
  - reLU often used in vision tasks

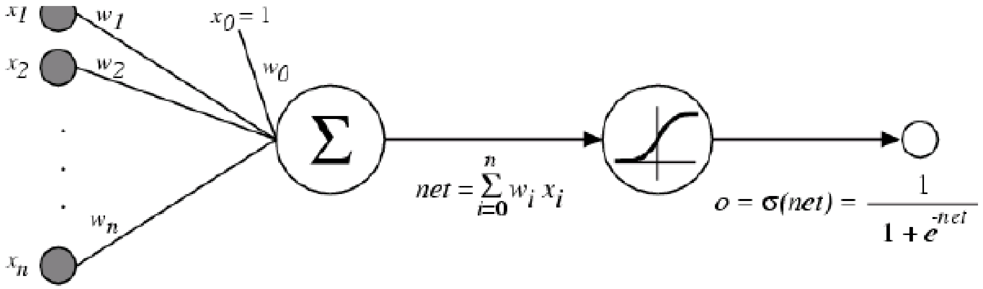


$$\max(0, w \cdot x + b).$$

Alternate 2: rectified linear unit

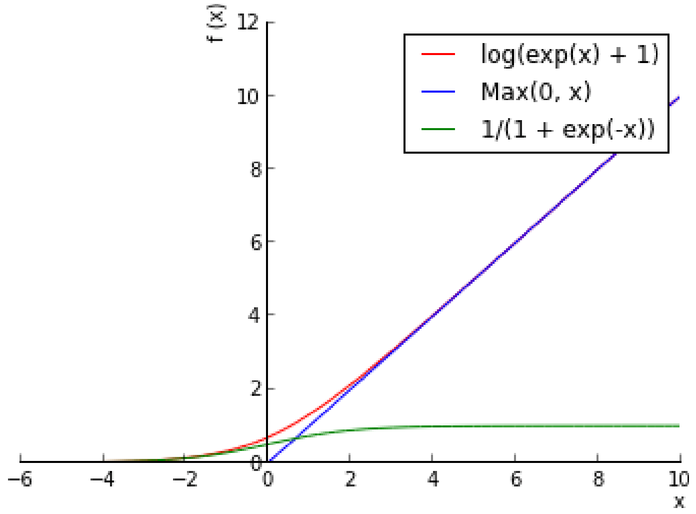
Linear with a cutoff at zero

(Implementation: clip the gradient when you pass zero)


x_1
w_1
$x_0 = 1$
w_0
x_2
w_2
w_n
x_n
Σ
$net = \sum_{i=0}^{n} w_i x_i$
$o = \sigma(net) = \frac{1}{1 + e^{-net}}$

Activation Functions

- A new change: modifying the nonlinearity
  - reLU often used in vision tasks



Alternate 2: rectified linear unit

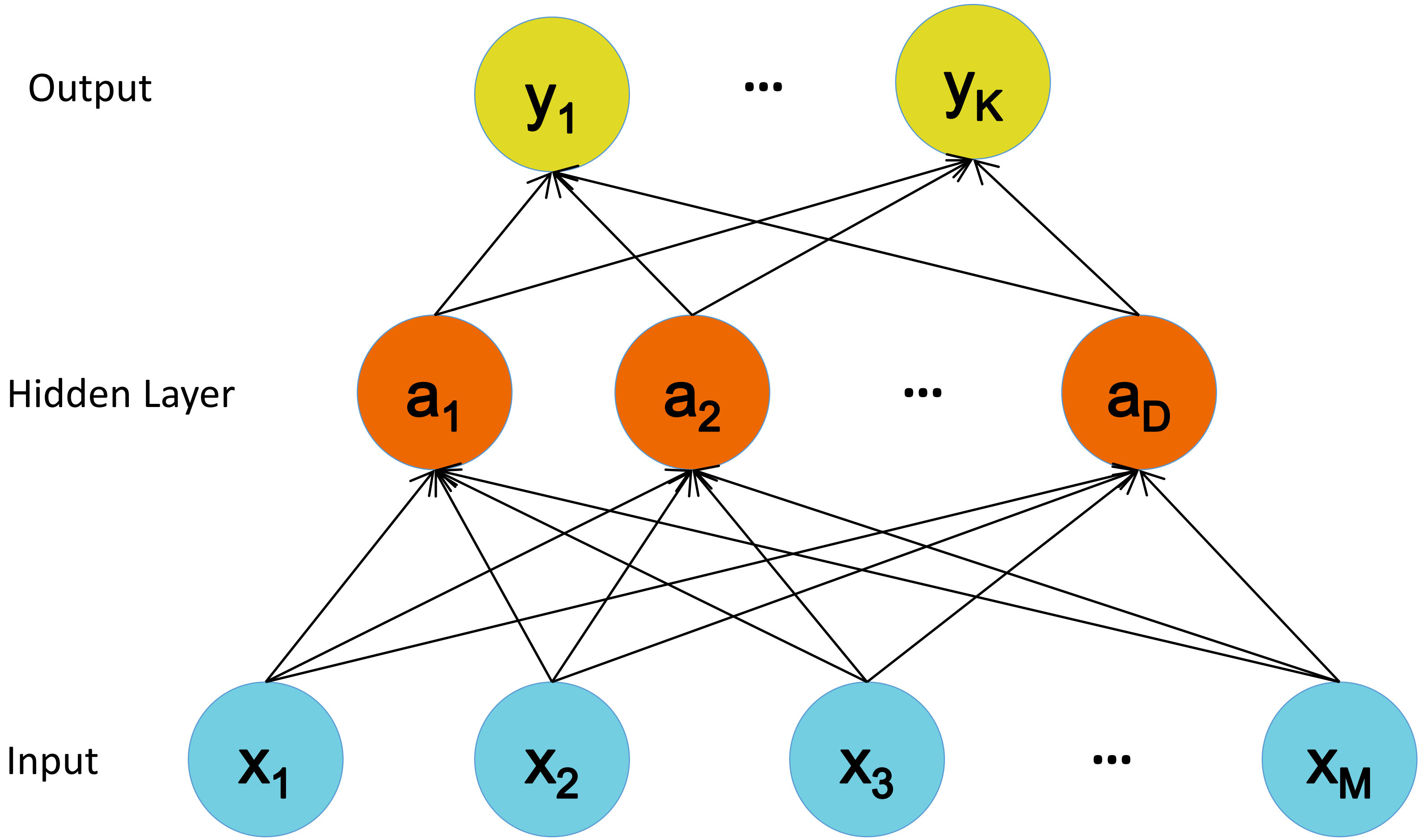
Soft version: log(exp(x)+1)

Doesn’t saturate (at one end)
Sparsifies outputs
Helps with vanishing gradient

Objective Functions for NNs

- Regression:
  - Use the same objective as Linear Regression
  - Quadratic loss (i.e. mean squared error)
- Classification:
  - Use the same objective as Logistic Regression
  - Cross-entropy (i.e. negative log likelihood)
  - This requires probabilities, so we add an additional "softmax" layer at the end of our network

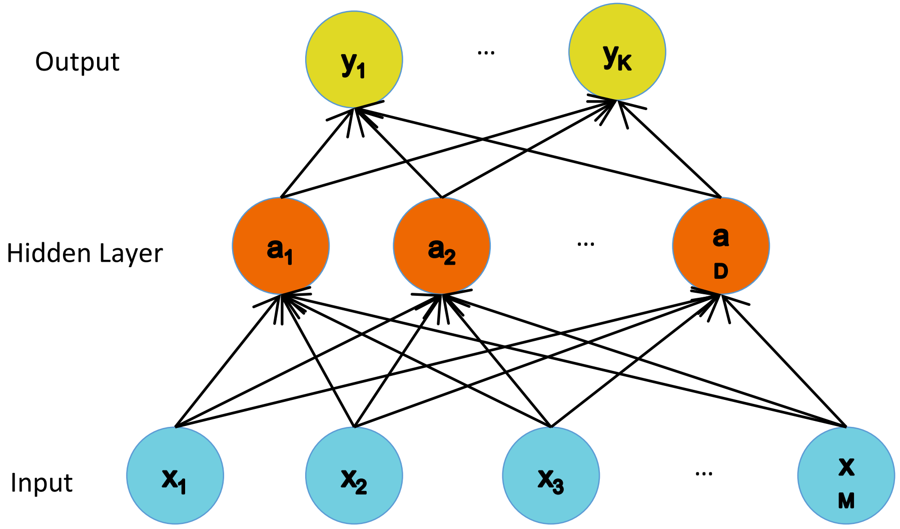
Multi-Class Output



Multi-Class Output

$$y_k = \frac{\exp(b_k)}{\sum_{l=1}^{K} \exp(b_l)}$$

Softmax

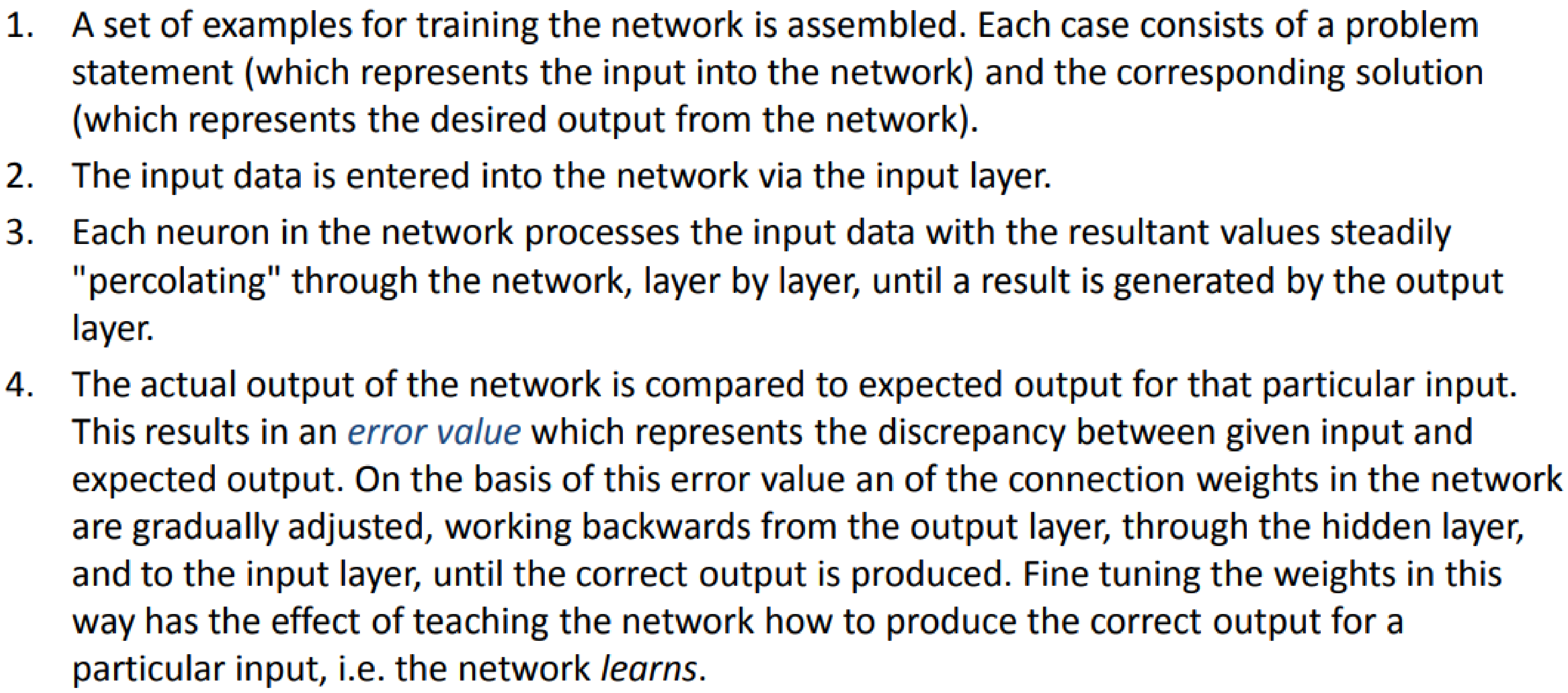


The Flow of Computations in Neural Networks

- The flow of computations in a neural network goes in two ways:
    1. Left-to-right: This is referred to as *forward propagation*, which results in computing the output of the network
    2. Right-to-left: This is referred to as *back propagation*, which results in computing the gradients (or derivatives) of the parameters in the network

- The intuition behind this 2-way flow of computations can be explained through the concept of “computation graphs”

Backpropagation

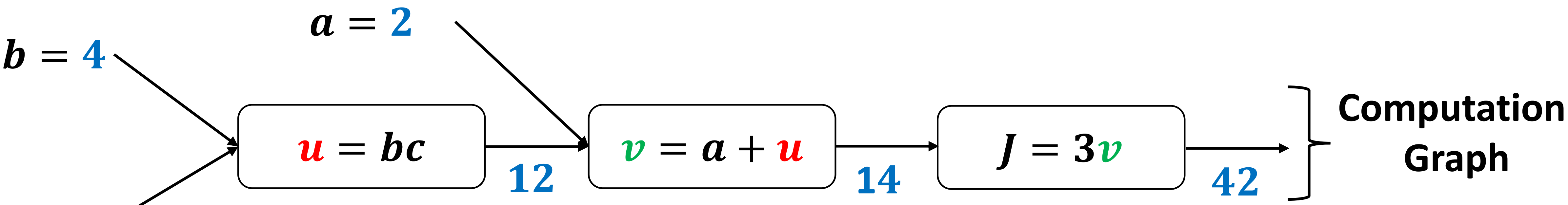


1. A set of examples for training the network is assembled. Each case consists of a problem statement (which represents the input into the network) and the corresponding solution (which represents the desired output from the network).
2. The input data is entered into the network via the input layer.
3. Each neuron in the network processes the input data with the resultant values steadily "percolating" through the network, layer by layer, until a result is generated by the output layer.
4. The actual output of the network is compared to expected output for that particular input. This results in an *error value* which represents the discrepancy between given input and expected output. On the basis of this error value an of the connection weights in the network are gradually adjusted, working backwards from the output layer, through the hidden layer, and to the input layer, until the correct output is produced. Fine tuning the weights in this way has the effect of teaching the network how to produce the correct output for a particular input, i.e. the network *learns*.

Forward Propagation

- Let us assume we want to compute the following function J:

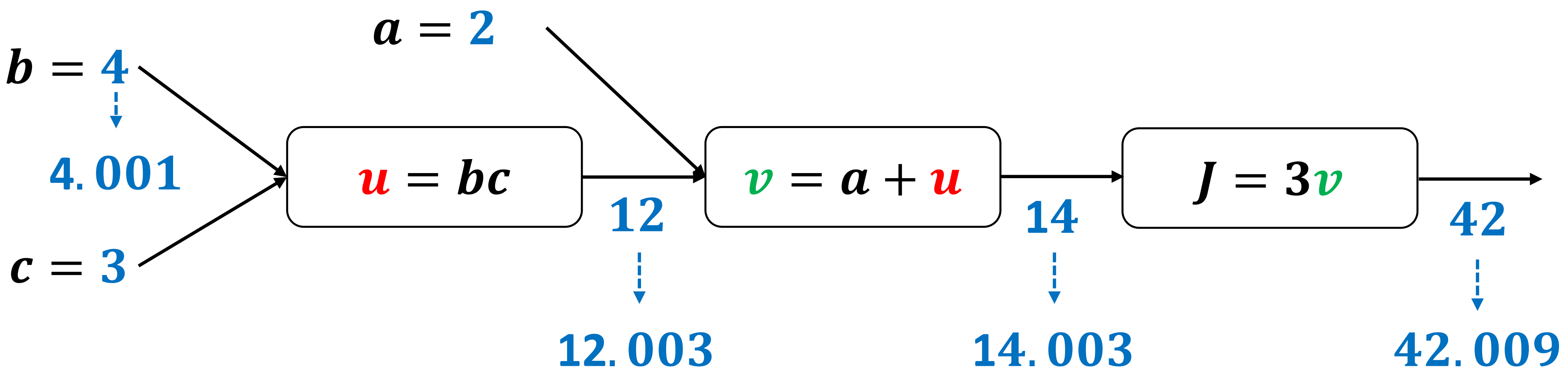
$$\left. J(a,b,c) = 3(\underbrace{\underbrace{a+\underbrace{bc}_{u}}_{v}}_{J}) \quad \right\} \quad \begin{aligned} u &= bc \\ v &= a + u \\ J &= 3v \end{aligned}$$



Forward propagation allows computing J

Backward Propagation

- Let us now compute the derivatives of the variables through the computation graph as follows:

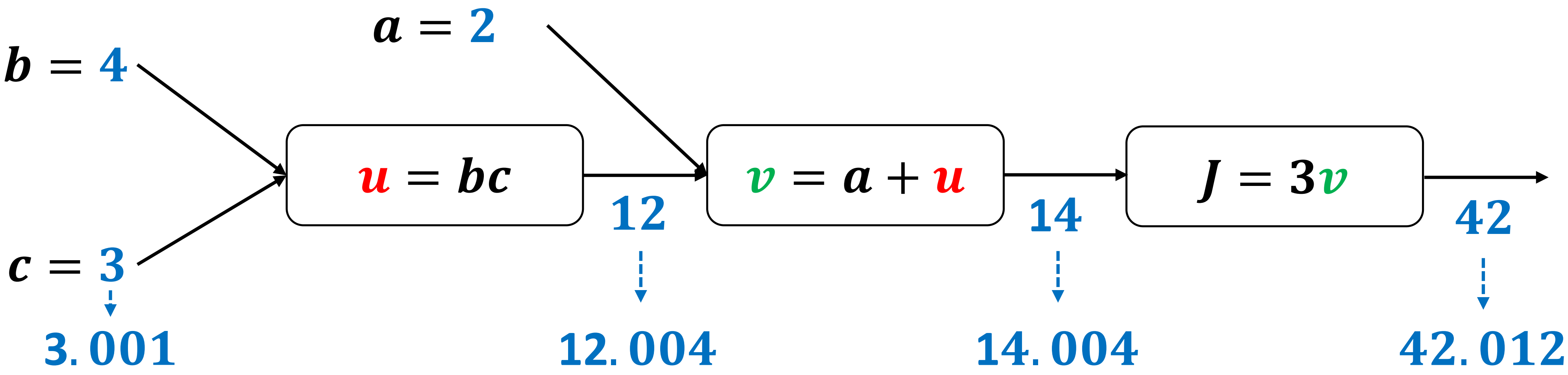


$$\frac{dJ}{db} = 9 = \underset{\overset{\downarrow}{3}}{\frac{dJ}{dv}} \times \underset{\overset{\downarrow}{1}}{\frac{dv}{du}} \times \underset{\overset{\downarrow}{3}}{\frac{du}{db}}$$

This is denoted as ***the chain rule*** in calculus

Backward Propagation

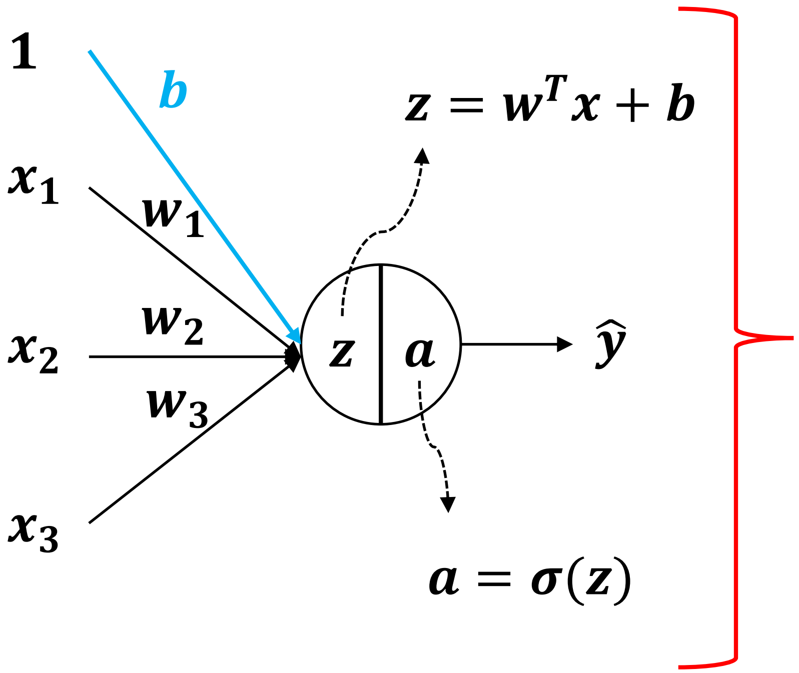
- Let us now compute the derivatives of the variables through the computation graph as follows:

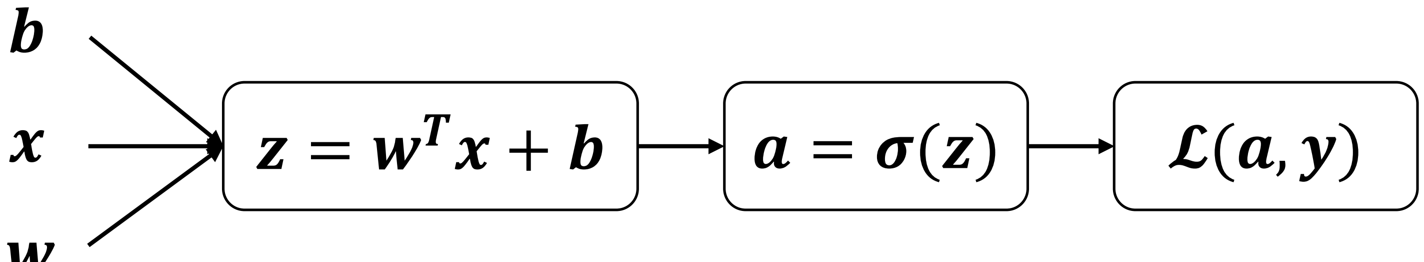


$$\frac{dJ}{dc} = 12 = \underset{\substack{\downarrow \\ 3}}{\frac{dJ}{dv}} \times \underset{\substack{\downarrow \\ 1}}{\frac{dv}{du}} \times \underset{\substack{\downarrow \\ 4}}{\frac{du}{dc}}$$

The Computation Graph of Logistic Regression

- Let us translate logistic regression (which is a **neural network** with only 1 neuron) into a computation graph



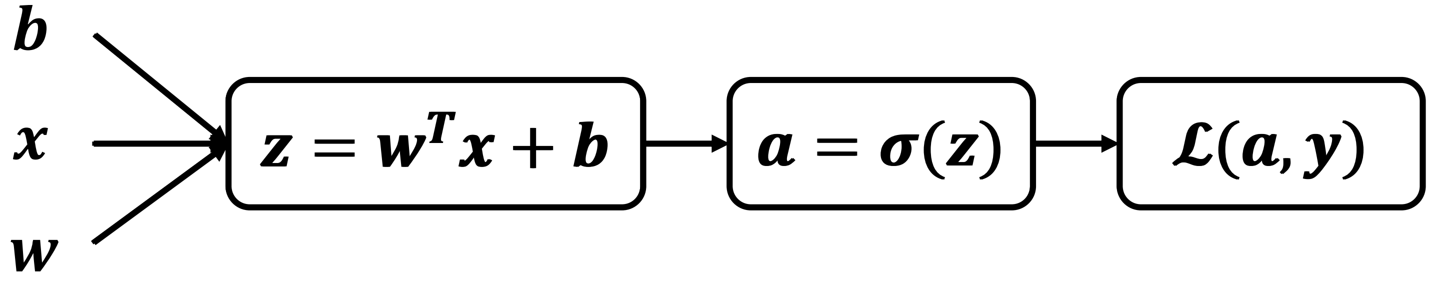


$\boldsymbol{b}$
$\boldsymbol{x}$
$\boldsymbol{w}$
$\boldsymbol{z = w^T x + b}$
$\boldsymbol{a = \sigma(z)}$
$\boldsymbol{\mathcal{L}(a, y)}$

Where $\boldsymbol{b = 1}$, $\boldsymbol{w = [w_1, w_2, w_3]}$, $\boldsymbol{x = [x_1, x_2, x_3]}$, and $\boldsymbol{\mathcal{L}(a, y)}$ is the cost (or ***loss***) function

Forward Propagation

- The loss function can be computed by moving from left to right



$\boldsymbol{b}$

$\boldsymbol{x}$

$\boldsymbol{w}$

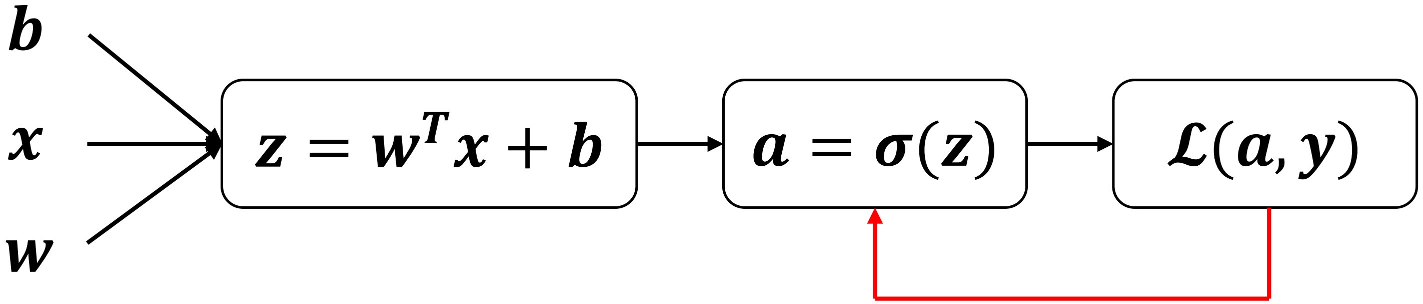
$\boldsymbol{z = w^T x + b}$

$\boldsymbol{a = \sigma(z)}$

$\boldsymbol{\mathcal{L}(a, y)}$

Backward Propagation

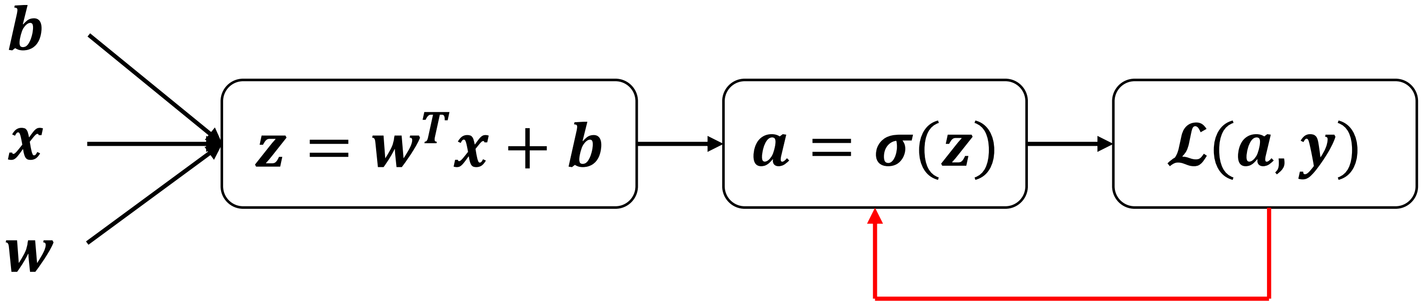
- The derivatives can be computed by moving from right to left



$\frac{\partial \mathcal{L}}{\partial a}$ = Partial derivative of $\mathcal{L}$ with respect to a

Backward Propagation

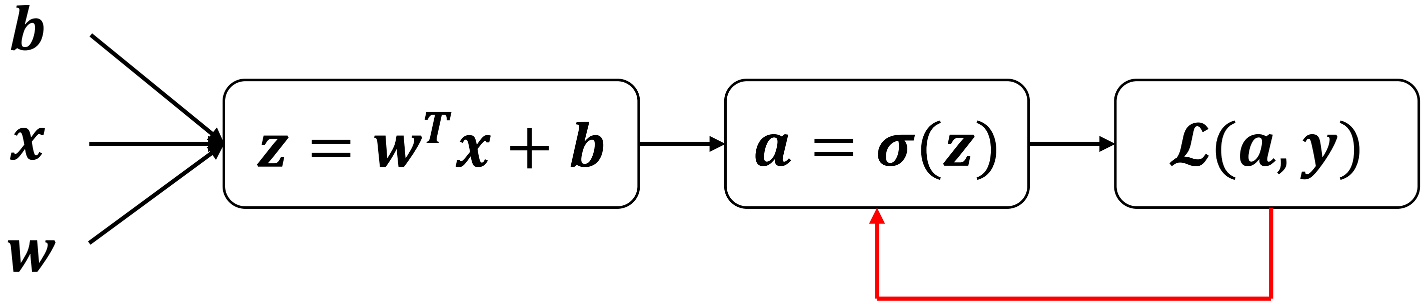
- The derivatives can be computed by moving from right to left



$$\frac{\partial \mathcal{L}}{\partial a} = \frac{\partial}{\partial a}\left(-ylog(a) - (1 - y)log(1 - a)\right)$$

Backward Propagation

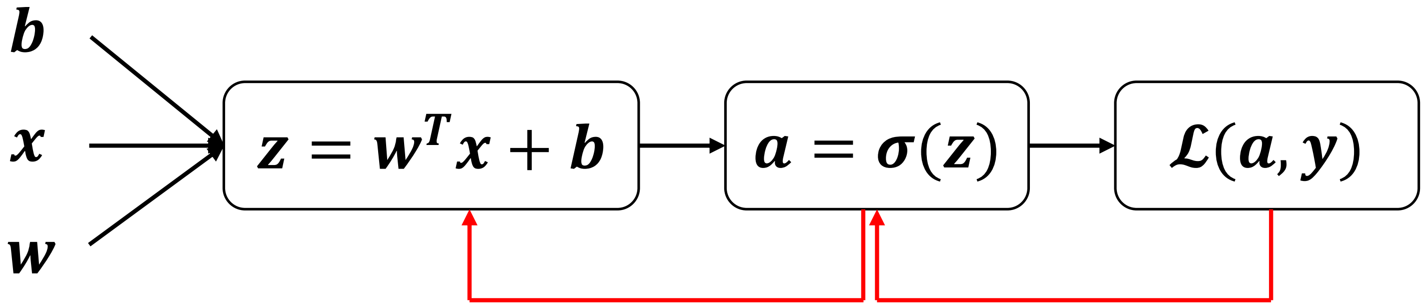
- The derivatives can be computed by moving from right to left



$$\frac{\partial \mathcal{L}}{\partial a} = \frac{-y}{a} + \frac{(1-y)}{(1-a)}$$

Backward Propagation

- The derivatives can be computed by moving from right to left

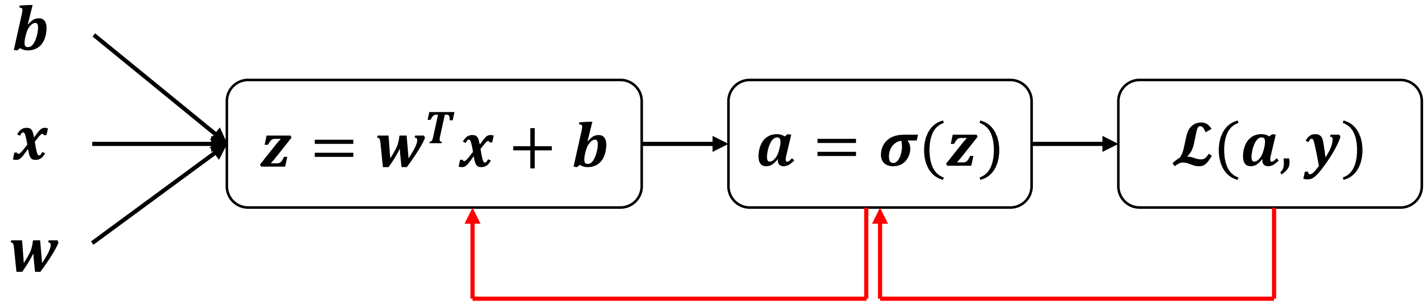


$\boldsymbol{b}$
$\boldsymbol{x}$
$\boldsymbol{w}$
$\boldsymbol{z = w^T x + b}$
$\boldsymbol{a = \sigma(z)}$
$\boldsymbol{\mathcal{L}(a, y)}$

$\dfrac{\partial \mathcal{L}}{\partial z}$ = Partial derivative of $\mathcal{L}$ with respect to z

Backward Propagation

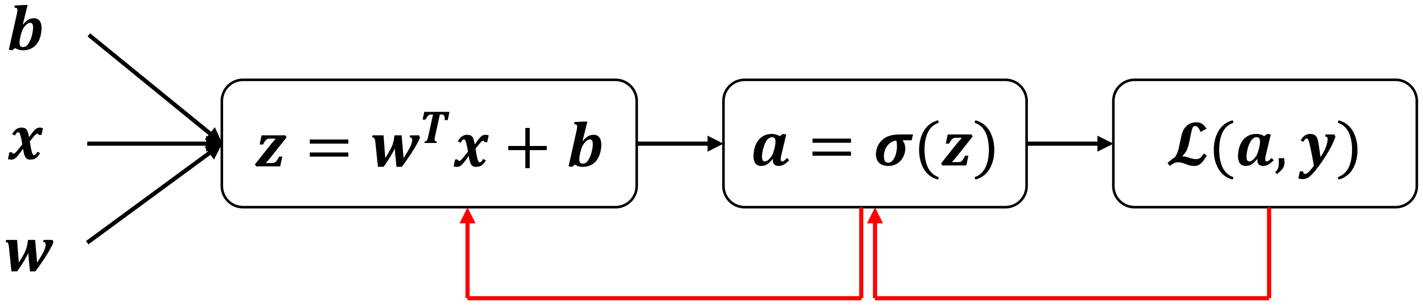
- The derivatives can be computed by moving from right to left



$$\frac{\partial \mathcal{L}}{\partial z} = \frac{\partial \mathcal{L}}{\partial a} \times \frac{\partial a}{\partial z} = \left(\frac{-y}{a} + \frac{(1-y)}{(1-a)} \right) \times \frac{\partial a}{\partial z} = \left(\frac{-y}{a} + \frac{(1-y)}{(1-a)} \right) \times \ a(1-a)$$

Backward Propagation

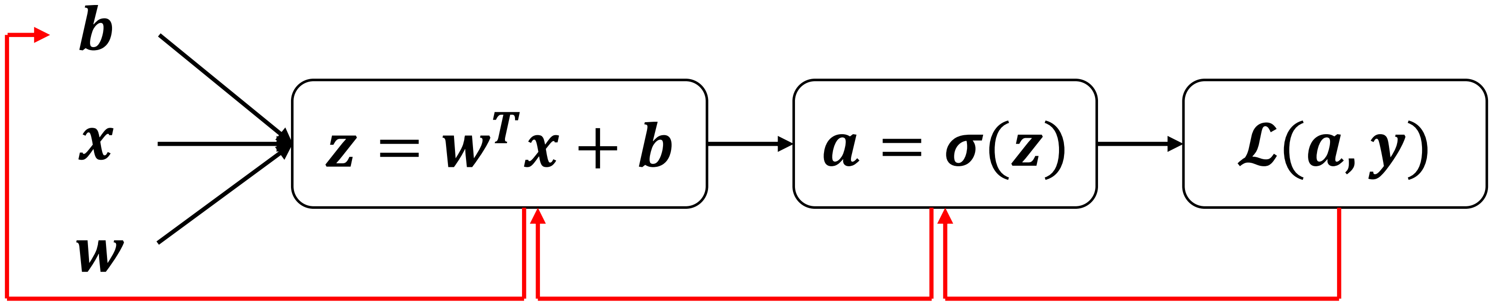
- The derivatives can be computed by moving from right to left



$$\frac{\partial \mathcal{L}}{\partial z} = a - y$$

Backward Propagation

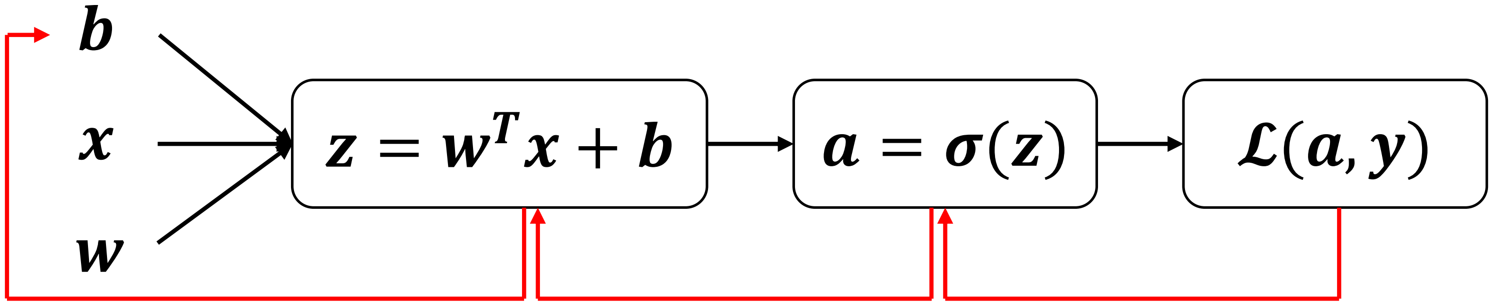
- The derivatives can be computed by moving from right to left


$\boldsymbol{b}$
$\boldsymbol{x}$
$\boldsymbol{w}$
$\boldsymbol{z = w^T x + b}$
$\boldsymbol{a = \sigma(z)}$
$\boldsymbol{\mathcal{L}(a, y)}$

$\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}} =$ Partial derivative of $\mathcal{L}$ with respect to $\boldsymbol{b}$

Backward Propagation

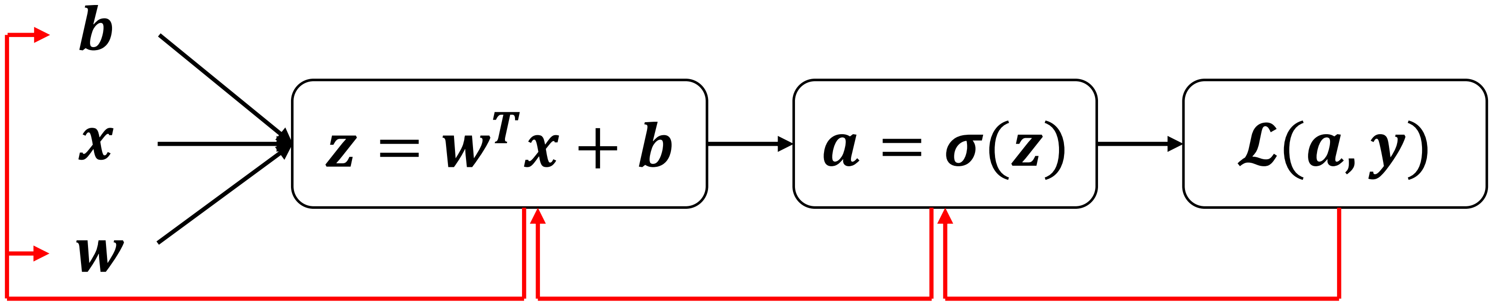
- The derivatives can be computed by moving from right to left



$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial a} \times \frac{\partial a}{\partial z} \times \frac{\partial z}{\partial b} = (a - y) \times \frac{\partial z}{\partial b} = (a - y) \times 1 = (a - y)$$

Backward Propagation

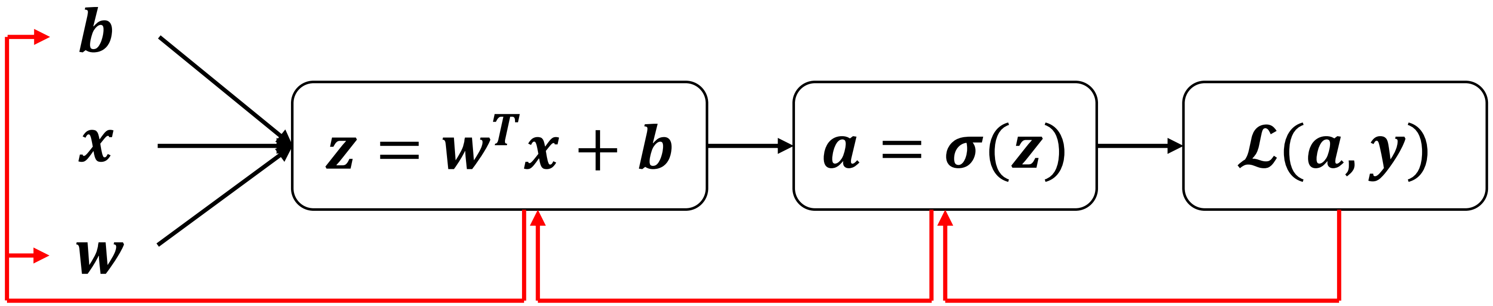
- The derivatives can be computed by moving from right to left



$\frac{\partial \mathcal{L}}{\partial w}$ = Partial derivative of $\mathcal{L}$ with respect to w

Backward Propagation

- The derivatives can be computed by moving from right to left



$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}}{\partial a} \times \frac{\partial a}{\partial z} \times \frac{\partial z}{\partial w} = (a - y) \times \frac{\partial z}{\partial w} = (a - y)x$$

Backward Propagation: Summary

- Here is the summary of the gradients in logistic regression:

$$\boldsymbol{dz} = \underbrace{\frac{\boldsymbol{\partial \mathcal{L}}}{\boldsymbol{\partial z}}} = \boldsymbol{a} - \boldsymbol{y}$$

We will denote this as $\boldsymbol{dz}$ for simplicity

Backward Propagation: Summary

- Here is the summary of the gradients in logistic regression:

$$\boldsymbol{dz} = \frac{\boldsymbol{\partial \mathcal{L}}}{\boldsymbol{\partial z}} = \boldsymbol{a - y}$$

$$\boldsymbol{db} = \frac{\boldsymbol{\partial \mathcal{L}}}{\boldsymbol{\partial b}} = \boldsymbol{a - y}$$

$$\boldsymbol{dw} = \frac{\boldsymbol{\partial \mathcal{L}}}{\boldsymbol{\partial w}} = \boldsymbol{(a - y)x}$$

Gradient Descent For Logistic Regression

- **Outline**:
  - Have a loss function $\boldsymbol{\mathcal{L}(w,b)}$, where $\boldsymbol{w}$ = $[\boldsymbol{w_1}, \dots, \boldsymbol{w_m}]$ and $\boldsymbol{b} = \boldsymbol{w_0}$
  - Start off with some guesses for $\boldsymbol{w_1}, \dots, \boldsymbol{w_m}$
    - It does not really matter what values you start off with for $\boldsymbol{w_1}, \dots, \boldsymbol{w_m}$, but a common choice is to set them all initially to zero
  - Repeat until convergence{

$$w_j = w_j - \alpha \frac{\partial \boldsymbol{\mathcal{L}(w,b)}}{\partial w_j}$$

$$b = b - \alpha \frac{\partial \boldsymbol{\mathcal{L}(w,b)}}{\partial b}$$

}

Let us focus on this part

Gradient Descent For Logistic Regression

- **Outline**:
  - Repeat until convergence{

Assuming n examples

```
for i = 1 to n:
    $z^{(i)} = w^T x^{(i)} + b$
    $a^{(i)} = \sigma(z^{(i)})$
    $dz^{(i)} = a^{(i)} - y^{(i)}$
    $dw = dw + dz^{(i)} x^{(i)}$
    $db = db + dz^{(i)}$
$dw = dw/n$
$db = db/n$
$w = w - \alpha dw$
$b = b - \alpha db$
}
```

Forward propagation

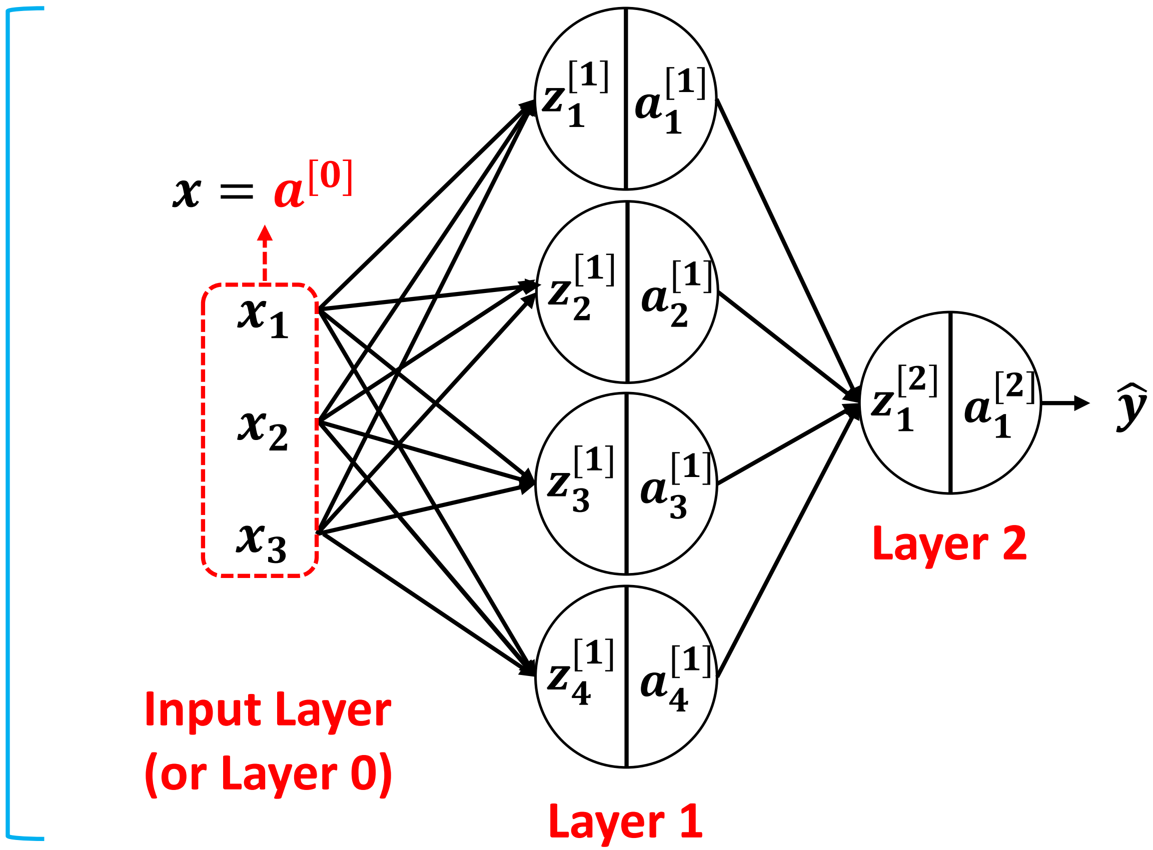
Backward propagation

Outside the loop

The Computation Graph of A Neural Network

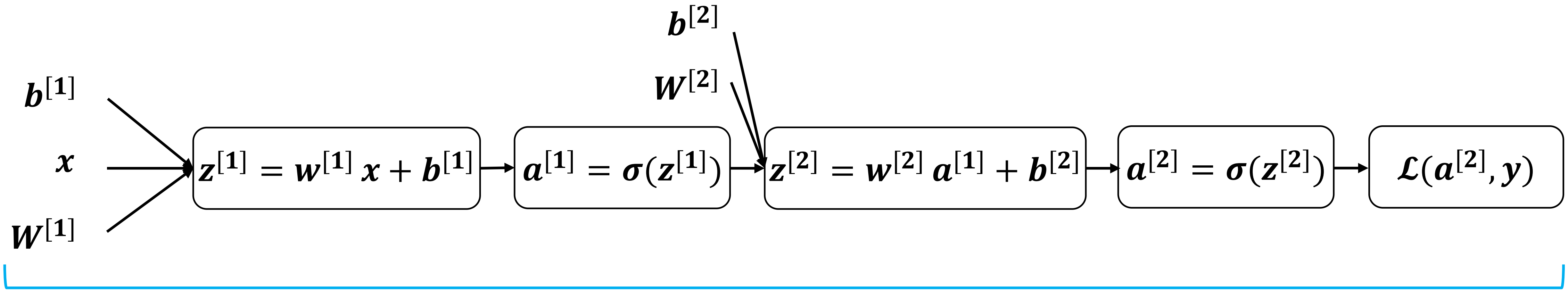
- we can represent any neural network in terms of a computation graph

A neural network with 2 layers



The Computation Graph of A Neural Network

- Akin to logistic regression, we can represent any neural network in terms of a computation graph



$\boldsymbol{b}^{[2]}$

$\boldsymbol{W}^{[2]}$

$\boldsymbol{b}^{[1]}$

$\boldsymbol{x}$

$\boldsymbol{W}^{[1]}$

$\boldsymbol{z}^{[1]} = \boldsymbol{w}^{[1]}\,\boldsymbol{x} + \boldsymbol{b}^{[1]}$

$\boldsymbol{a}^{[1]} = \boldsymbol{\sigma}(\boldsymbol{z}^{[1]})$

$\boldsymbol{z}^{[2]} = \boldsymbol{w}^{[2]}\,\boldsymbol{a}^{[1]} + \boldsymbol{b}^{[2]}$

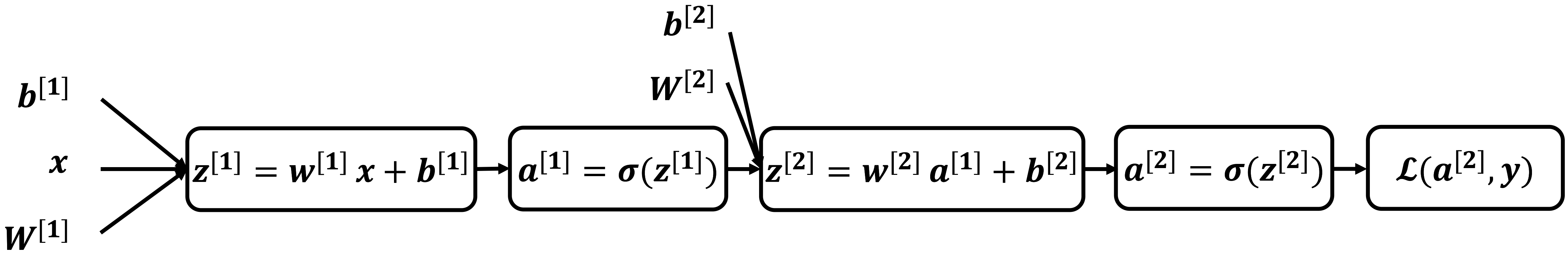
$\boldsymbol{a}^{[2]} = \boldsymbol{\sigma}(\boldsymbol{z}^{[2]})$

$\mathcal{L}(\boldsymbol{a}^{[2]}, \boldsymbol{y})$

The corresponding computation graph

Forward Propagation

- The loss function can be computed by moving from left to right



$\boldsymbol{b}^{[1]}$

$\boldsymbol{x}$

$\boldsymbol{W}^{[1]}$

$\boldsymbol{z}^{[1]} = \boldsymbol{w}^{[1]}\,\boldsymbol{x} + \boldsymbol{b}^{[1]}$

$\boldsymbol{a}^{[1]} = \boldsymbol{\sigma}(\boldsymbol{z}^{[1]})$

$\boldsymbol{b}^{[2]}$

$\boldsymbol{W}^{[2]}$

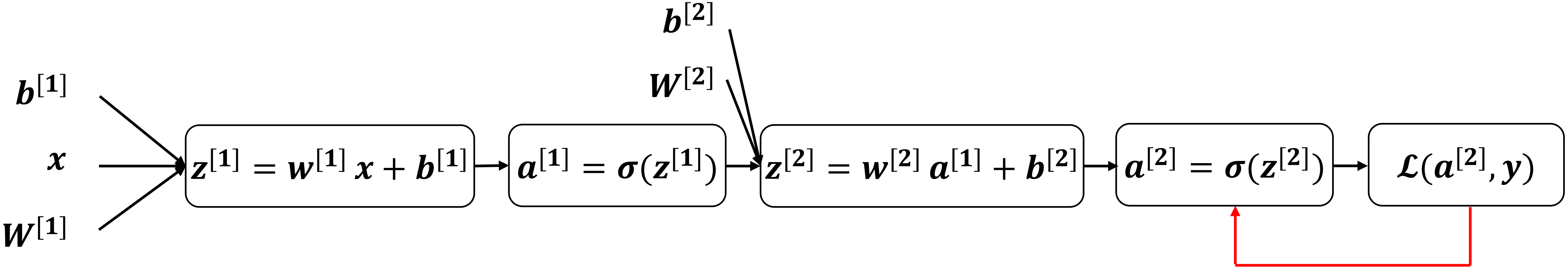
$\boldsymbol{z}^{[2]} = \boldsymbol{w}^{[2]}\,\boldsymbol{a}^{[1]} + \boldsymbol{b}^{[2]}$

$\boldsymbol{a}^{[2]} = \boldsymbol{\sigma}(\boldsymbol{z}^{[2]})$

$\boldsymbol{\mathcal{L}}(\boldsymbol{a}^{[2]}, \boldsymbol{y})$

Backward Propagation

- The derivatives can be computed by moving from right to left



$\boldsymbol{b}^{[1]}$

$\boldsymbol{x}$

$\boldsymbol{W}^{[1]}$

$\boldsymbol{z}^{[1]} = \boldsymbol{w}^{[1]}\,\boldsymbol{x} + \boldsymbol{b}^{[1]}$

$\boldsymbol{a}^{[1]} = \boldsymbol{\sigma}(\boldsymbol{z}^{[1]})$

$\boldsymbol{b}^{[2]}$

$\boldsymbol{W}^{[2]}$

$\boldsymbol{z}^{[2]} = \boldsymbol{w}^{[2]}\,\boldsymbol{a}^{[1]} + \boldsymbol{b}^{[2]}$

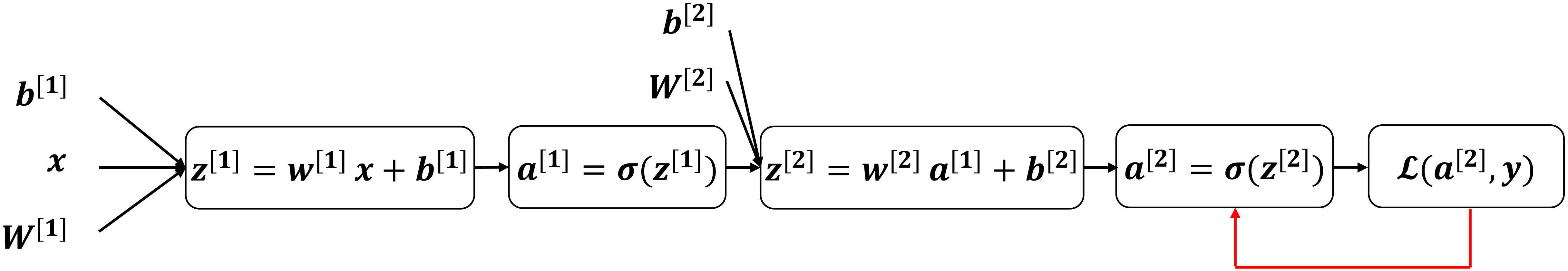
$\boldsymbol{a}^{[2]} = \boldsymbol{\sigma}(\boldsymbol{z}^{[2]})$

$\boldsymbol{\mathcal{L}}(\boldsymbol{a}^{[2]}, \boldsymbol{y})$

$\frac{\partial \boldsymbol{\mathcal{L}}}{\partial \boldsymbol{a}^{[2]}}$ = Partial derivative of $\boldsymbol{\mathcal{L}}$ with respect to $\boldsymbol{a}^{[2]}$

Backward Propagation

- The derivatives can be computed by moving from right to left



$\boldsymbol{b}^{[1]}$

$\boldsymbol{x}$

$\boldsymbol{W}^{[1]}$

$\boldsymbol{z}^{[1]} = \boldsymbol{w}^{[1]}\,\boldsymbol{x} + \boldsymbol{b}^{[1]}$

$\boldsymbol{a}^{[1]} = \boldsymbol{\sigma}(\boldsymbol{z}^{[1]})$

$\boldsymbol{b}^{[2]}$

$\boldsymbol{W}^{[2]}$

$\boldsymbol{z}^{[2]} = \boldsymbol{w}^{[2]}\,\boldsymbol{a}^{[1]} + \boldsymbol{b}^{[2]}$

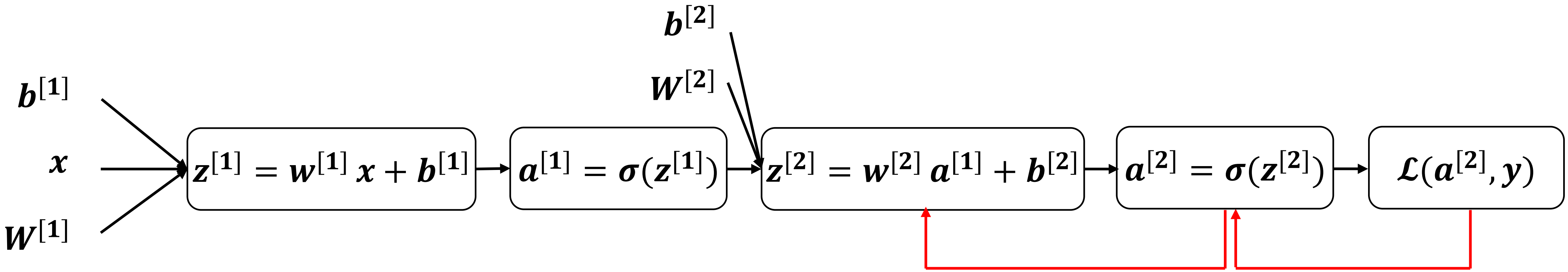
$\boldsymbol{a}^{[2]} = \boldsymbol{\sigma}(\boldsymbol{z}^{[2]})$

$\mathcal{L}(\boldsymbol{a}^{[2]}, \boldsymbol{y})$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{a}^{[2]}} = \frac{-\boldsymbol{y}}{\boldsymbol{a}^{[2]}} + \frac{(\mathbf{1} - \boldsymbol{y})}{(\mathbf{1} - \boldsymbol{a}^{[2]})}$$

Backward Propagation

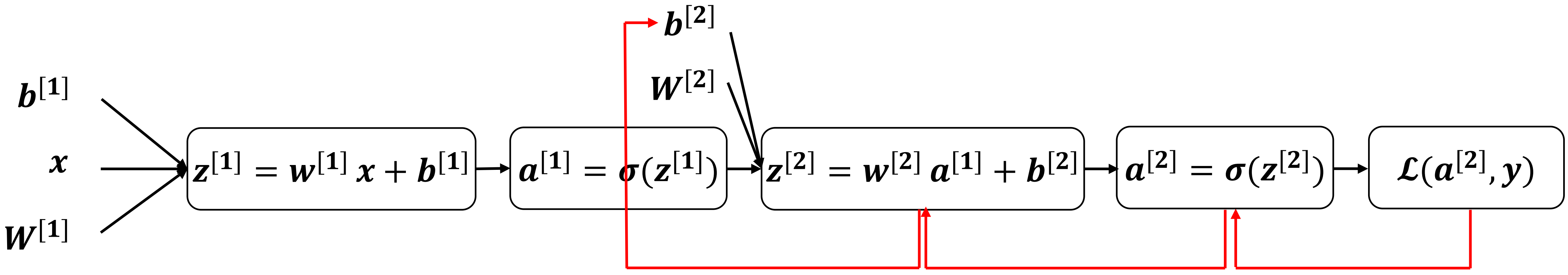
- The derivatives can be computed by moving from right to left


$b^{[2]}$
$W^{[2]}$
$b^{[1]}$
x
$W^{[1]}$
$z^{[1]} = w^{[1]} x + b^{[1]}$
$a^{[1]} = \sigma(z^{[1]})$
$z^{[2]} = w^{[2]} a^{[1]} + b^{[2]}$
$a^{[2]} = \sigma(z^{[2]})$
$\mathcal{L}(a^{[2]}, y)$

$$\frac{\partial \mathcal{L}}{\partial z^{[2]}} = \frac{\partial \mathcal{L}}{\partial a^{[2]}} \times \frac{\partial a^{[2]}}{\partial z^{[2]}} = a^{[2]} - y$$

Backward Propagation

- The derivatives can be computed by moving from right to left

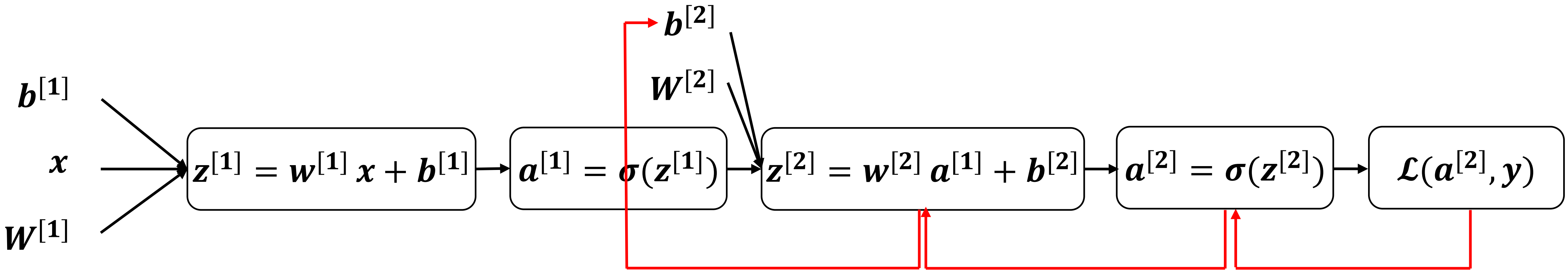


$b^{[1]}$
x
$W^{[1]}$
$z^{[1]} = w^{[1]}x + b^{[1]}$
$a^{[1]} = \sigma(z^{[1]})$
$b^{[2]}$
$W^{[2]}$
$z^{[2]} = w^{[2]}a^{[1]} + b^{[2]}$
$a^{[2]} = \sigma(z^{[2]})$
$\mathcal{L}(a^{[2]}, y)$

$\frac{\partial \mathcal{L}}{\partial b^{[2]}}$ = Partial derivative of $\mathcal{L}$ with respect to $b^{[2]}$

Backward Propagation

- The derivatives can be computed by moving from right to left

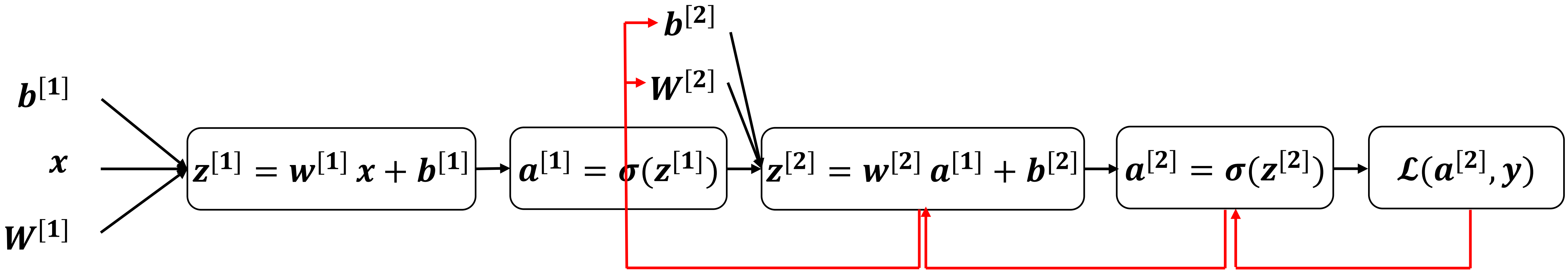


$b^{[1]}$
x
$W^{[1]}$
$z^{[1]} = w^{[1]} x + b^{[1]}$
$a^{[1]} = \sigma(z^{[1]})$
$b^{[2]}$
$W^{[2]}$
$z^{[2]} = w^{[2]} a^{[1]} + b^{[2]}$
$a^{[2]} = \sigma(z^{[2]})$
$\mathcal{L}(a^{[2]}, y)$

$$\frac{\partial \mathcal{L}}{\partial b^{[2]}} = \frac{\partial \mathcal{L}}{\partial a^{[2]}} \times \frac{\partial a^{[2]}}{\partial z^{[2]}} \times \frac{\partial z^{[2]}}{\partial b^{[2]}} = a^{[2]} - y$$

Backward Propagation

- The derivatives can be computed by moving from right to left

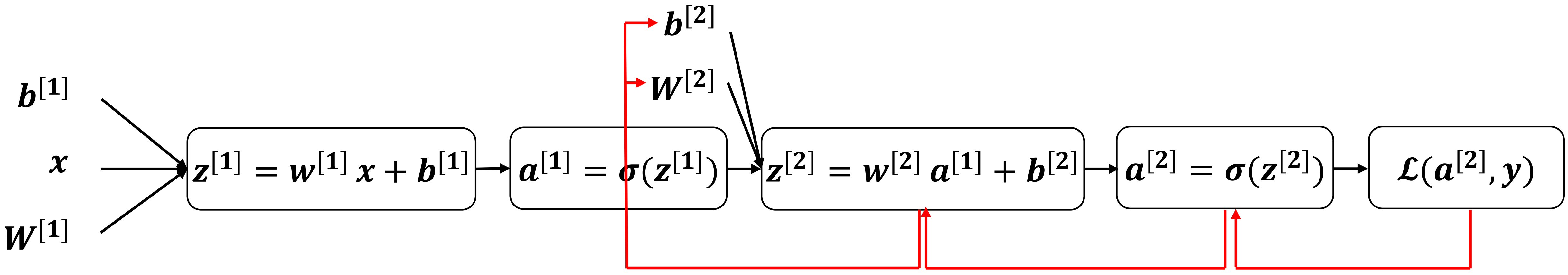


$b^{[1]}$
x
$W^{[1]}$
$z^{[1]} = w^{[1]}x + b^{[1]}$
$a^{[1]} = \sigma(z^{[1]})$
$b^{[2]}$
$W^{[2]}$
$z^{[2]} = w^{[2]}a^{[1]} + b^{[2]}$
$a^{[2]} = \sigma(z^{[2]})$
$\mathcal{L}(a^{[2]}, y)$

$\frac{\partial \mathcal{L}}{\partial W^{[2]}}$ = Partial derivative of $\mathcal{L}$ with respect to $W^{[2]}$

Backward Propagation

- The derivatives can be computed by moving from right to left

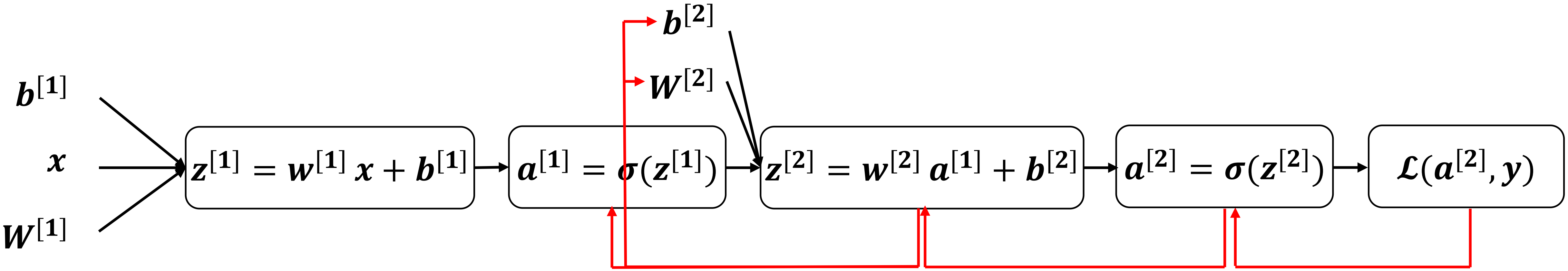


$b^{[2]}$
$W^{[2]}$
$b^{[1]}$
x
$W^{[1]}$
$z^{[1]} = w^{[1]}x + b^{[1]}$
$a^{[1]} = \sigma(z^{[1]})$
$z^{[2]} = w^{[2]}a^{[1]} + b^{[2]}$
$a^{[2]} = \sigma(z^{[2]})$
$\mathcal{L}(a^{[2]}, y)$

$$\frac{\partial \mathcal{L}}{\partial W^{[2]}} = \frac{\partial \mathcal{L}}{\partial a^{[2]}} \times \frac{\partial a^{[2]}}{\partial z^{[2]}} \times \frac{\partial z^{[2]}}{\partial W^{[2]}} = (a^{[2]} - y)a^{[1]T}$$

Backward Propagation

- The derivatives can be computed by moving from right to left

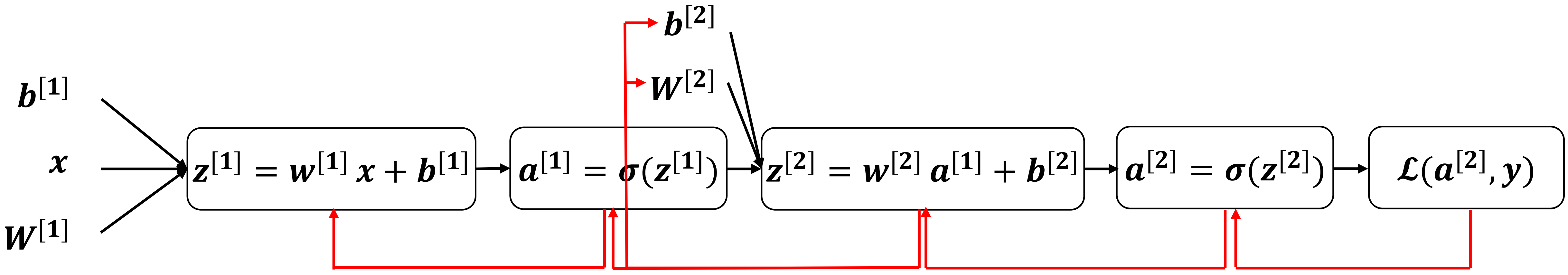


$b^{[1]}$
x
$W^{[1]}$
$z^{[1]} = w^{[1]}x + b^{[1]}$
$a^{[1]} = \sigma(z^{[1]})$
$b^{[2]}$
$W^{[2]}$
$z^{[2]} = w^{[2]}a^{[1]} + b^{[2]}$
$a^{[2]} = \sigma(z^{[2]})$
$\mathcal{L}(a^{[2]}, y)$

$$\frac{\partial \mathcal{L}}{\partial a^{[1]}} = \frac{\partial \mathcal{L}}{\partial a^{[2]}} \times \frac{\partial a^{[2]}}{\partial z^{[2]}} \times \frac{\partial z^{[2]}}{\partial a^{[1]}} = (a^{[2]} - y)w^{[2]T}$$

Backward Propagation

- The derivatives can be computed by moving from right to left

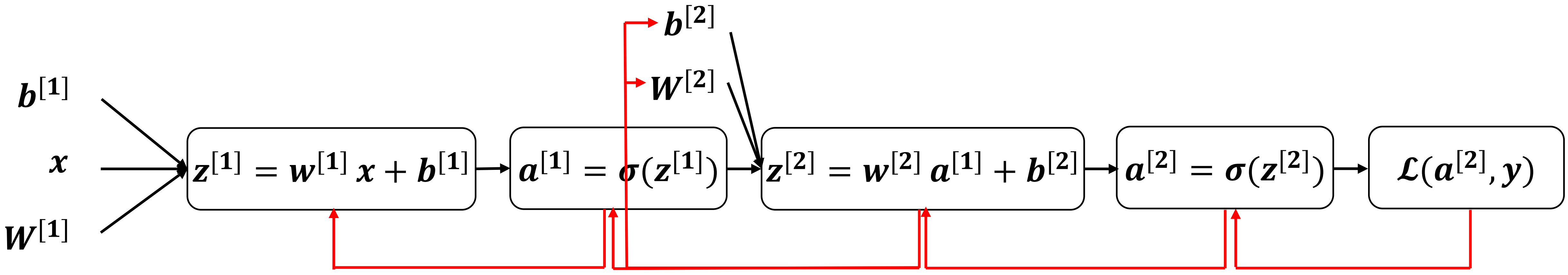


$b^{[1]}$
x
$W^{[1]}$
$z^{[1]} = w^{[1]}x + b^{[1]}$
$a^{[1]} = \sigma(z^{[1]})$
$b^{[2]}$
$W^{[2]}$
$z^{[2]} = w^{[2]}a^{[1]} + b^{[2]}$
$a^{[2]} = \sigma(z^{[2]})$
$\mathcal{L}(a^{[2]}, y)$

$\dfrac{\partial \mathcal{L}}{\partial z^{[1]}} =$ Partial derivative of $\mathcal{L}$ with respect to $z^{[1]}$

Backward Propagation

- The derivatives can be computed by moving from right to left



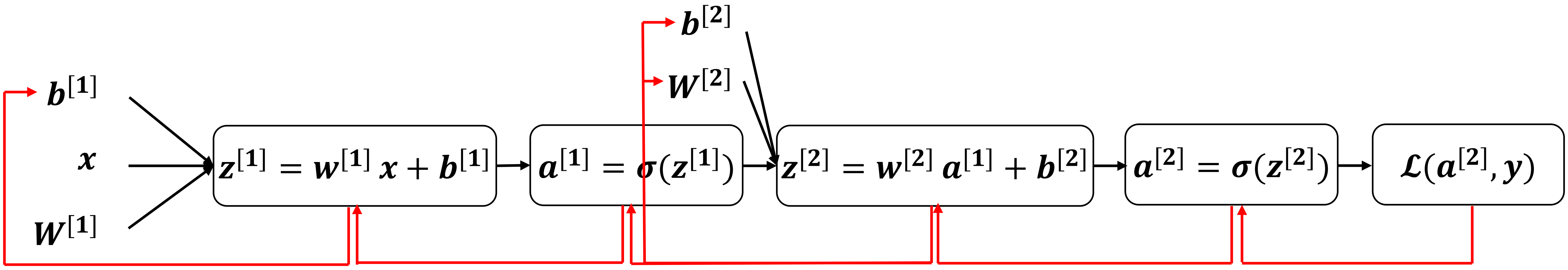
$b^{[1]}$
x
$W^{[1]}$
$z^{[1]} = w^{[1]}x + b^{[1]}$
$a^{[1]} = \sigma(z^{[1]})$
$b^{[2]}$
$W^{[2]}$
$z^{[2]} = w^{[2]}a^{[1]} + b^{[2]}$
$a^{[2]} = \sigma(z^{[2]})$
$\mathcal{L}(a^{[2]}, y)$

$$\frac{\partial \mathcal{L}}{\partial z^{[1]}} = \frac{\partial \mathcal{L}}{\partial a^{[2]}} \times \frac{\partial a^{[2]}}{\partial z^{[2]}} \times \frac{\partial z^{[2]}}{\partial a^{[1]}} \times \frac{\partial a^{[1]}}{\partial z^{[1]}} = (a^{[2]} - y)w^{[2]T} * a^{[1]}(1 - a^{[1]})$$

Element-wise product

Backward Propagation

- The derivatives can be computed by moving from right to left

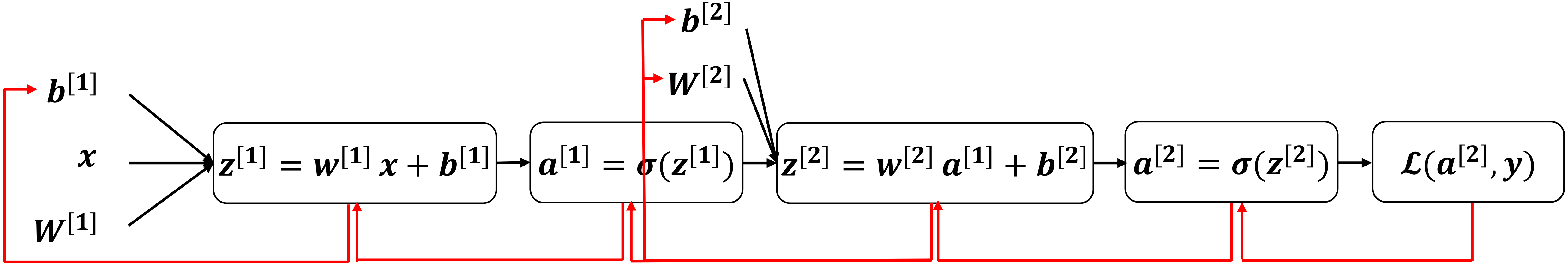


$b^{[1]}$
x
$W^{[1]}$
$z^{[1]} = w^{[1]} x + b^{[1]}$
$a^{[1]} = \sigma(z^{[1]})$
$b^{[2]}$
$W^{[2]}$
$z^{[2]} = w^{[2]} a^{[1]} + b^{[2]}$
$a^{[2]} = \sigma(z^{[2]})$
$\mathcal{L}(a^{[2]}, y)$

$$\frac{\partial \mathcal{L}}{\partial b^{[1]}} = \frac{\partial \mathcal{L}}{\partial a^{[2]}} \times \frac{\partial a^{[2]}}{\partial z^{[2]}} \times \frac{\partial z^{[2]}}{\partial a^{[1]}} \times \frac{\partial a^{[1]}}{\partial z^{[1]}} \times \frac{\partial z^{[1]}}{\partial b^{[1]}}$$

Backward Propagation

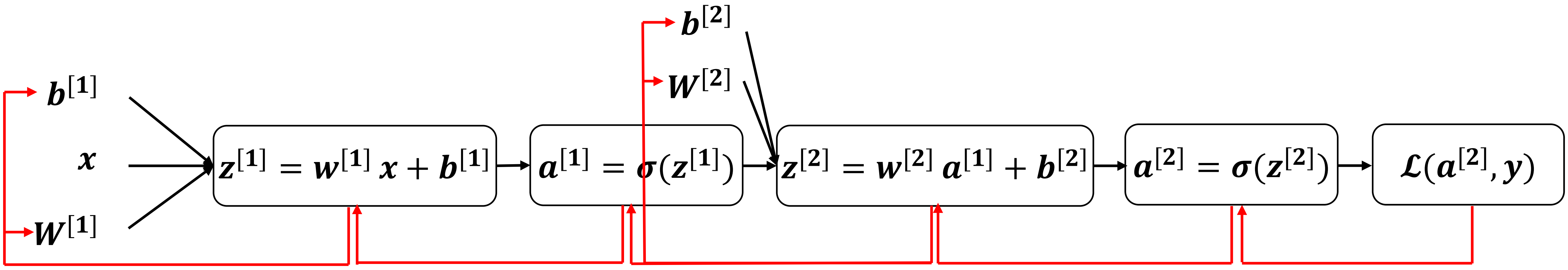
- The derivatives can be computed by moving from right to left


$b^{[2]}$
$W^{[2]}$
$b^{[1]}$
x
$W^{[1]}$
$z^{[1]} = w^{[1]}x + b^{[1]}$
$a^{[1]} = \sigma(z^{[1]})$
$z^{[2]} = w^{[2]}a^{[1]} + b^{[2]}$
$a^{[2]} = \sigma(z^{[2]})$
$\mathcal{L}(a^{[2]}, y)$

$$\frac{\partial \mathcal{L}}{\partial b^{[1]}} = (a^{[2]} - y)w^{[2]T} * a^{[1]}(1 - a^{[1]})$$

Backward Propagation

- The derivatives can be computed by moving from right to left

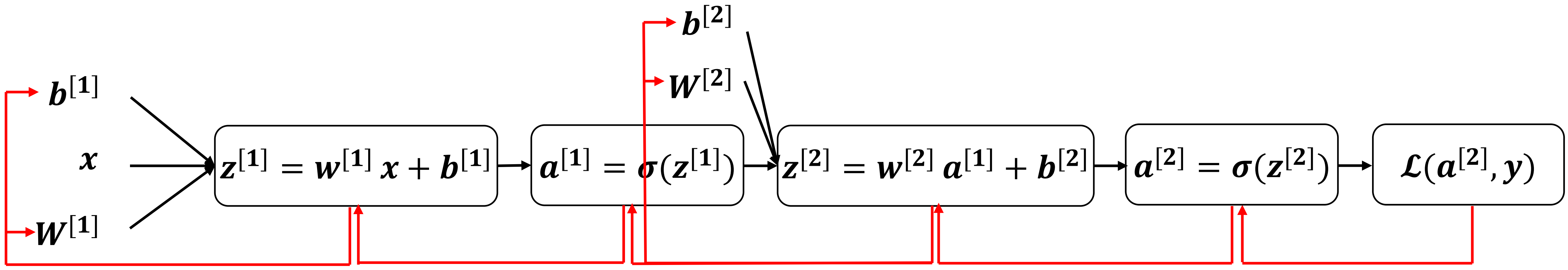


$b^{[1]}$
x
$W^{[1]}$
$z^{[1]} = w^{[1]}x + b^{[1]}$
$a^{[1]} = \sigma(z^{[1]})$
$b^{[2]}$
$W^{[2]}$
$z^{[2]} = w^{[2]}a^{[1]} + b^{[2]}$
$a^{[2]} = \sigma(z^{[2]})$
$\mathcal{L}(a^{[2]}, y)$

$$\frac{\partial \mathcal{L}}{\partial W^{[1]}} = \frac{\partial \mathcal{L}}{\partial a^{[2]}} \times \frac{\partial a^{[2]}}{\partial z^{[2]}} \times \frac{\partial z^{[2]}}{\partial a^{[1]}} \times \frac{\partial a^{[1]}}{\partial z^{[1]}} \times \frac{\partial z^{[1]}}{\partial W^{[1]}}$$

Backward Propagation

- The derivatives can be computed by moving from right to left



$b^{[2]}$
$W^{[2]}$
$b^{[1]}$
x
$W^{[1]}$
$z^{[1]} = w^{[1]} x + b^{[1]}$
$a^{[1]} = \sigma(z^{[1]})$
$z^{[2]} = w^{[2]} a^{[1]} + b^{[2]}$
$a^{[2]} = \sigma(z^{[2]})$
$\mathcal{L}(a^{[2]}, y)$

$$\frac{\partial \mathcal{L}}{\partial W^{[1]}} = \left((a^{[2]} - y)w^{[2]T} * a^{[1]}(1 - a^{[1]})\right) x^T$$

Backward Propagation: Summary

- Here is the summary of the gradients in our given neural network:

$$\boldsymbol{dz^{[2]} = \frac{\partial \mathcal{L}}{\partial z^{[2]}} = a^{[2]} - y}$$

$$\boldsymbol{db^{[2]} = \frac{\partial \mathcal{L}}{\partial b^{[2]}} = a^{[2]} - y}$$

$$\boldsymbol{dW^{[2]} = \frac{\partial \mathcal{L}}{\partial W^{[2]}} = (a^{[2]} - y)a^{[1]T}}$$

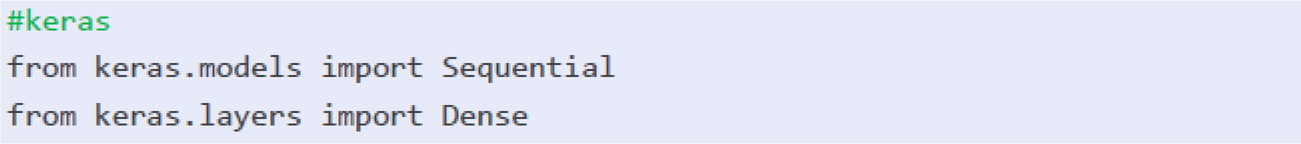
$$\boldsymbol{dz^{[1]} = \frac{\partial \mathcal{L}}{\partial z^{[1]}} = dz^{[2]}w^{[2]T} * a^{[1]}(1 - a^{[1]})}$$

$$\boldsymbol{db^{[1]} = \frac{\partial \mathcal{L}}{\partial b^{[1]}} = dz^{[2]}w^{[2]T} * a^{[1]}(1 - a^{[1]})}$$

$$\boldsymbol{dW^{[1]} = \frac{\partial \mathcal{L}}{\partial W^{[1]}} = \left(dz^{[2]}w^{[2]T} * a^{[1]}(1 - a^{[1]})\right)x^{T}}$$

Perceptron

- **Architecture du réseau**
- Nous importons les classes Sequential et Dense pour définir notre modèle et son architecture.



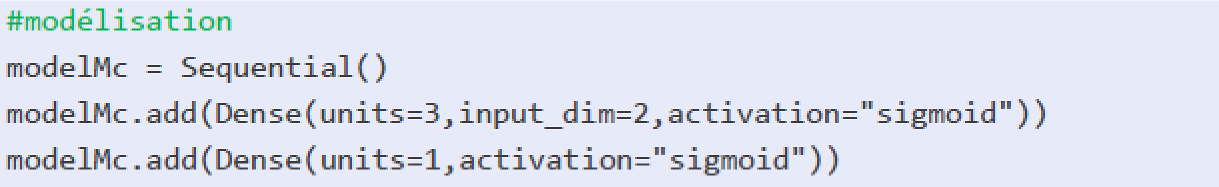
```
#keras
from keras.models import Sequential
from keras.layers import Dense
```

- La classe Sequential est une structure, initialement vide, qui permet de définir un empilement de couches de neurones (https://keras.io/getting-started/sequential-model-guide/)

Perceptron multicouche

- Dans cette section, nous avons un perceptron multicouche. Nous créons toujours une structure Sequential, dans lequel nous ajoutons successivement deux objets Dense : le premier fait la jonction entre la couche d’entrée (d’où l’option input_dim indiquant le nombre de variables prédictives) et la couche cachée qui comporte (units = 3) neurones ; le second entre cette couche cachée et la sortie à un seul neurone (units = 1). Nous avons une fonction d’activation sigmoïde dans les deux cas.

MLP en utilisant Keras



```
#modélisation
modelMc = Sequential()
modelMc.add(Dense(units=3,input_dim=2,activation="sigmoid"))
modelMc.add(Dense(units=1,activation="sigmoid"))
```

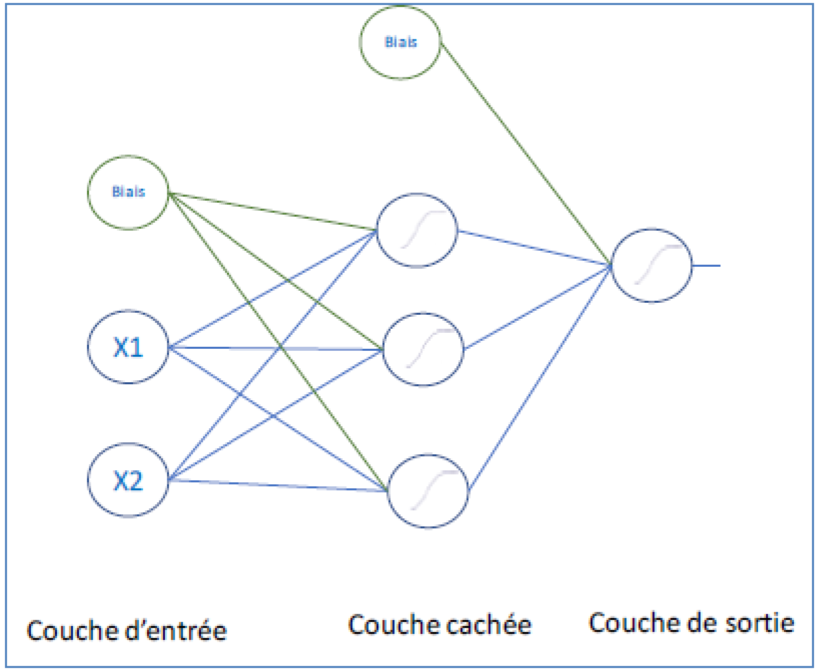
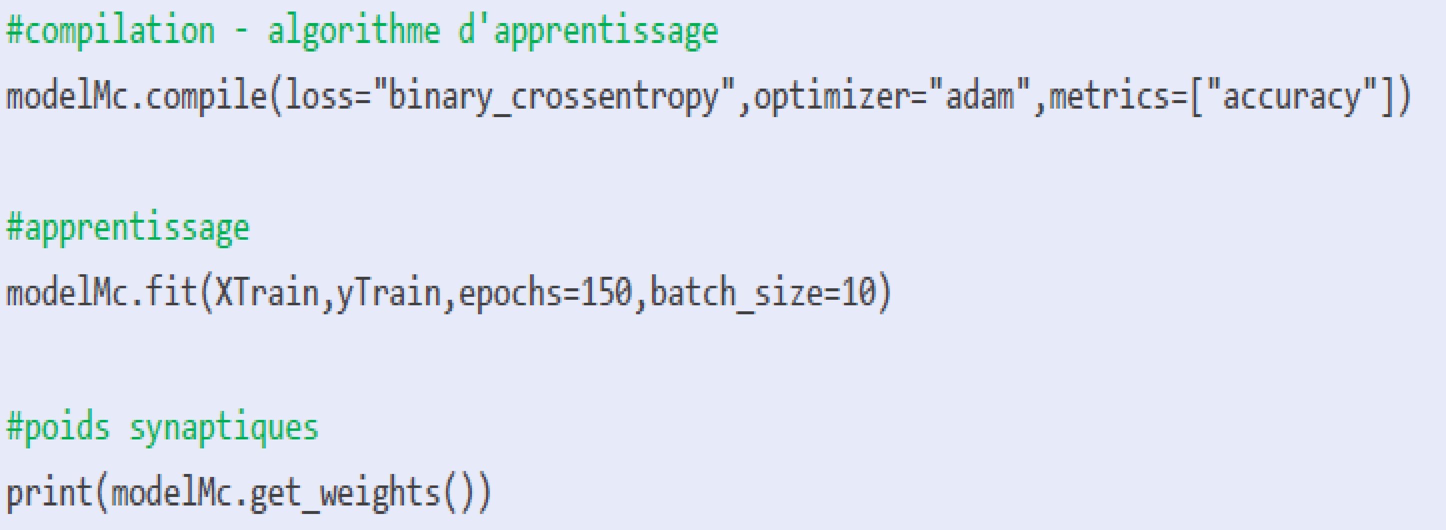
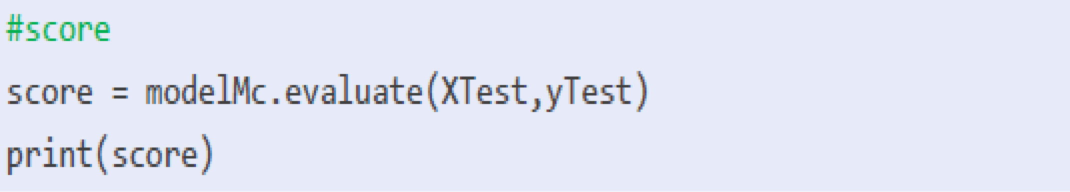


Figure 4 - Perceptron multicouche - Structure



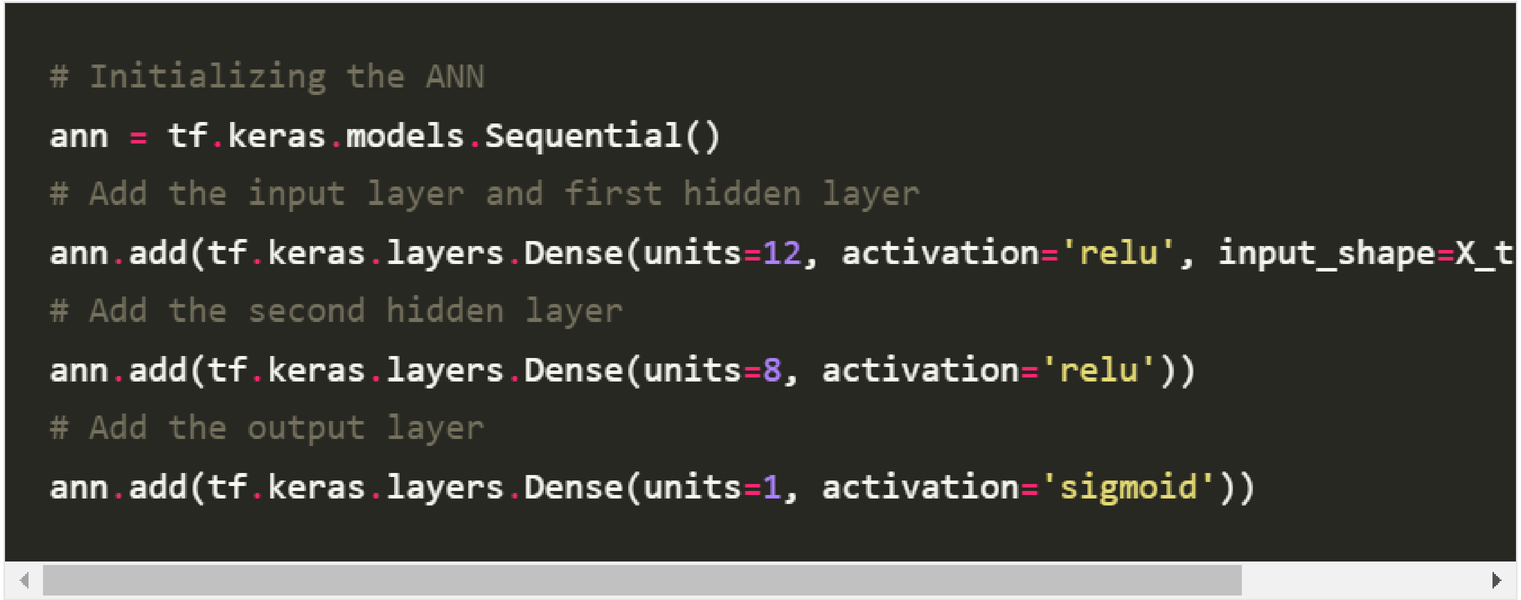
```
#compilation - algorithme d'apprentissage
modelMc.compile(loss="binary_crossentropy",optimizer="adam",metrics=["accuracy"])

#apprentissage
modelMc.fit(XTrain,yTrain,epochs=150,batch_size=10)

#poids synaptiques
print(modelMc.get_weights())
```



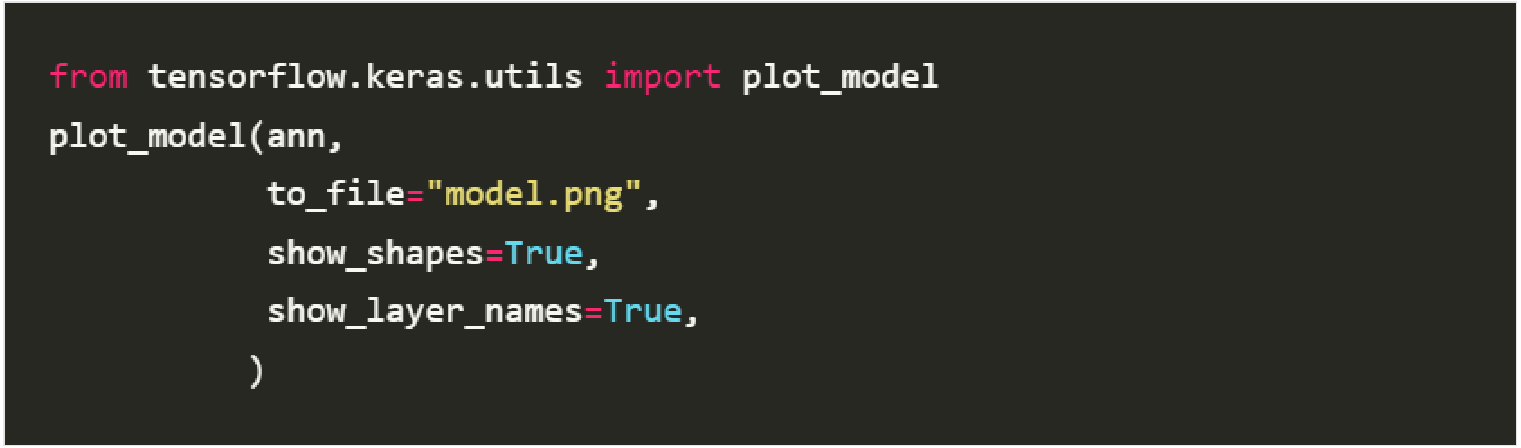
```
#score
score = modelMc.evaluate(XTest,yTest)
print(score)
```

Here two hidden layers..

g

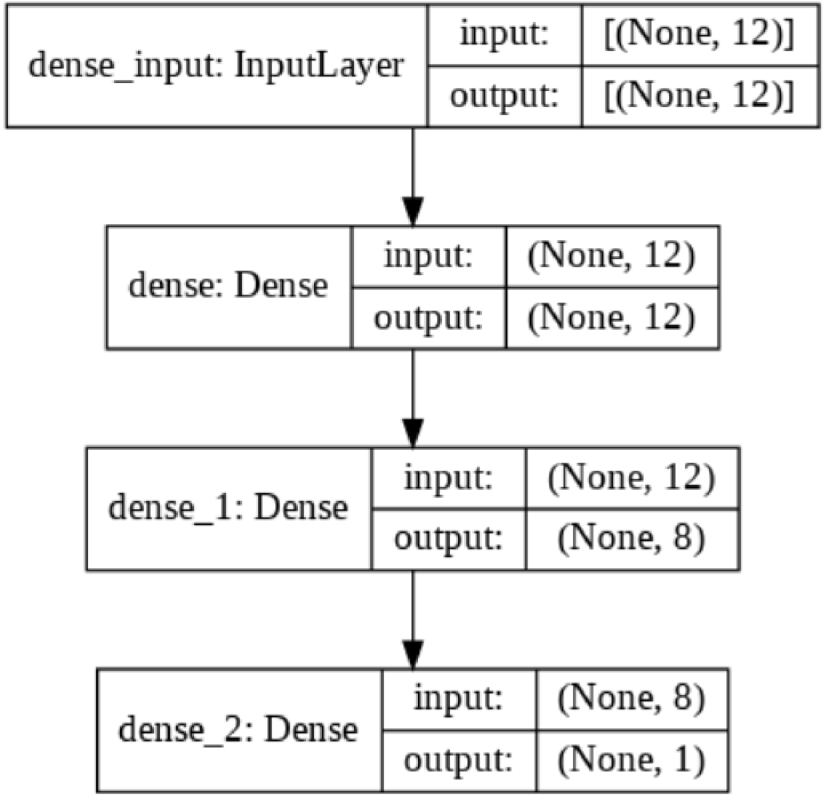


```
# Initializing the ANN
ann = tf.keras.models.Sequential()
# Add the input layer and first hidden layer
ann.add(tf.keras.layers.Dense(units=12, activation='relu', input_shape=X_t
# Add the second hidden layer
ann.add(tf.keras.layers.Dense(units=8, activation='relu'))
# Add the output layer
ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```



```python
from tensorflow.keras.utils import plot_model
plot_model(ann,
           to_file="model.png",
           show_shapes=True,
           show_layer_names=True,
          )
```

Output



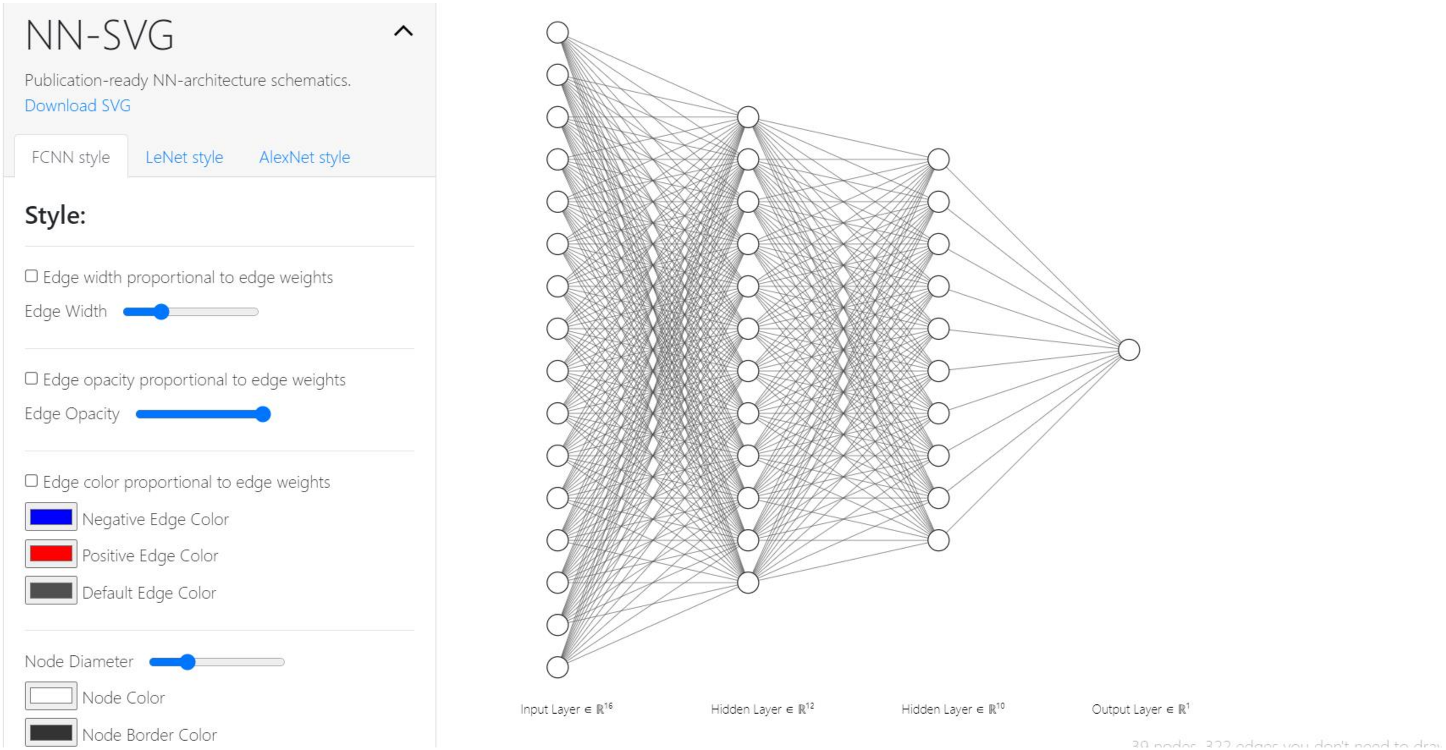
dense_input: InputLayer | input: | [(None, 12)]
output: | [(None, 12)]

dense: Dense | input: | (None, 12)
output: | (None, 12)

dense_1: Dense | input: | (None, 12)
output: | (None, 8)

dense_2: Dense | input: | (None, 8)
output: | (None, 1)

https://alexlenail.me/NN-SVG/



NN-SVG

Publication-ready NN-architecture schematics.

Download SVG

FCNN style | LeNet style | AlexNet style

Style:

□ Edge width proportional to edge weights

Edge Width

□ Edge opacity proportional to edge weights

Edge Opacity

□ Edge color proportional to edge weights

🟦 Negative Edge Color

🟥 Positive Edge Color

⬛ Default Edge Color

Node Diameter

⬜ Node Color

⬛ Node Border Color

39 nodes, 322 edges you don't need to dra