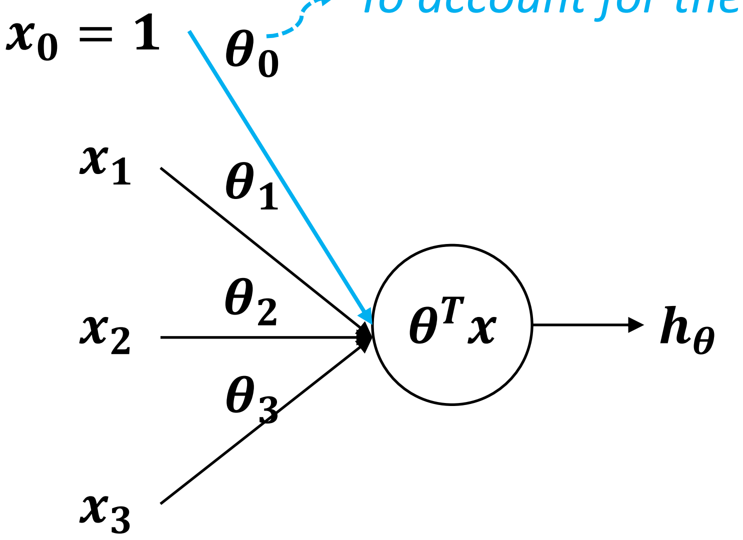
Basics of Neural Networks

Linear Regression

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

To account for the intercept term



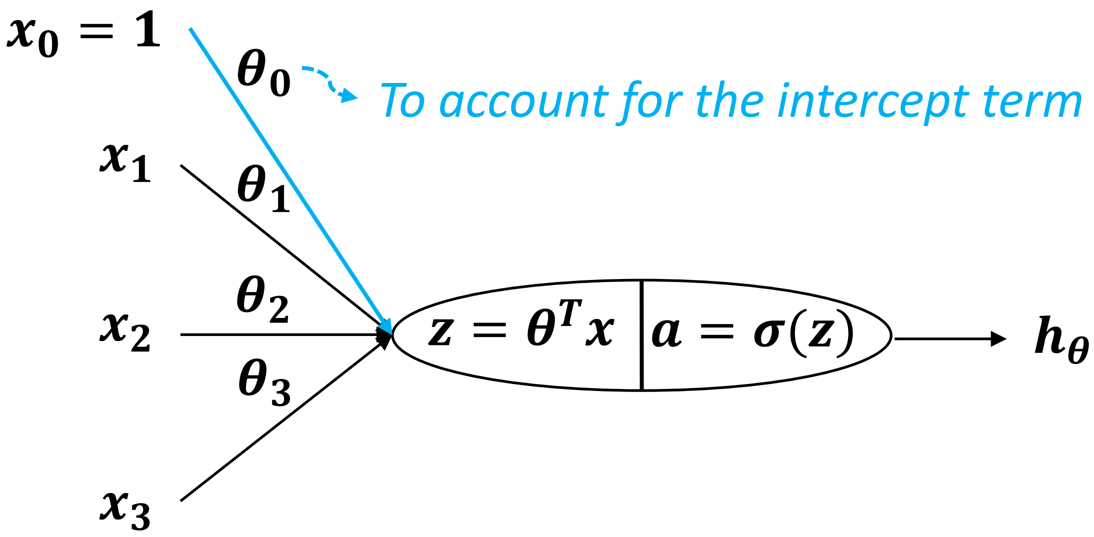
$$\boldsymbol{h_\theta = \theta^T x = [\theta_0\ \theta_1\ \theta_2 \theta_3] \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}}$$

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

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

Logistic Regression

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



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

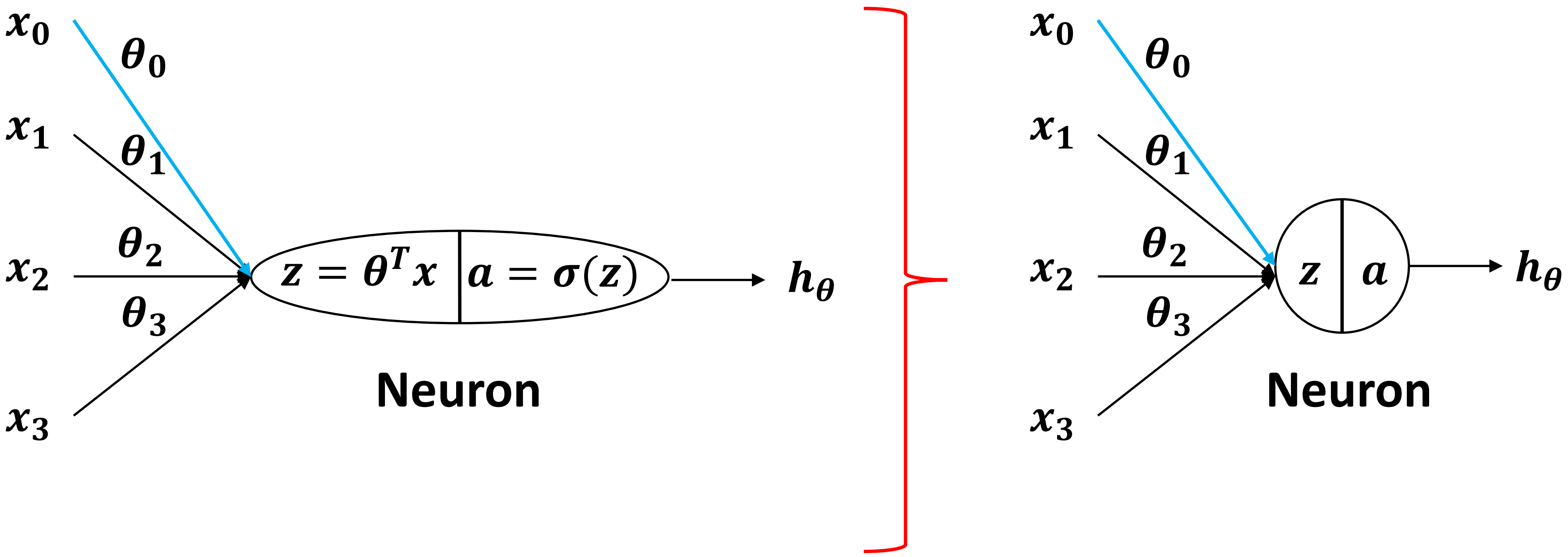
$$= \sigma(\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3)$$

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

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

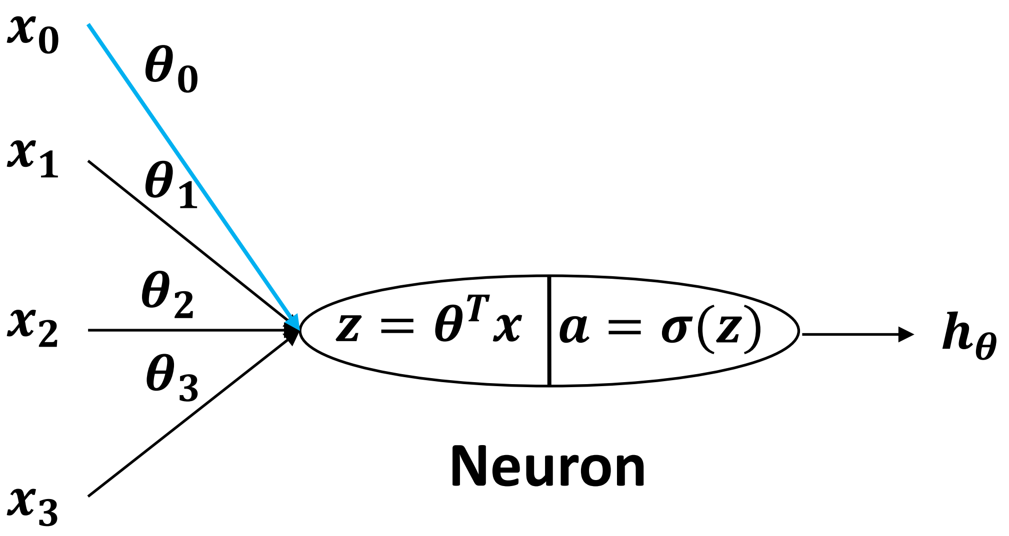
Towards Neural Networks

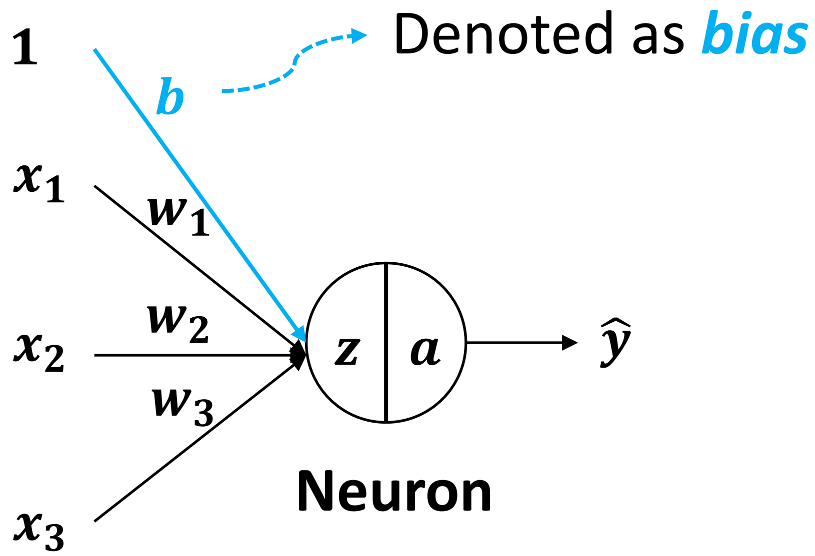
Logistic regression can be considered as **neural network** with only 1 neuron



Towards Neural Networks

Logistic regression can be considered as **neural network** with only 1 neuron

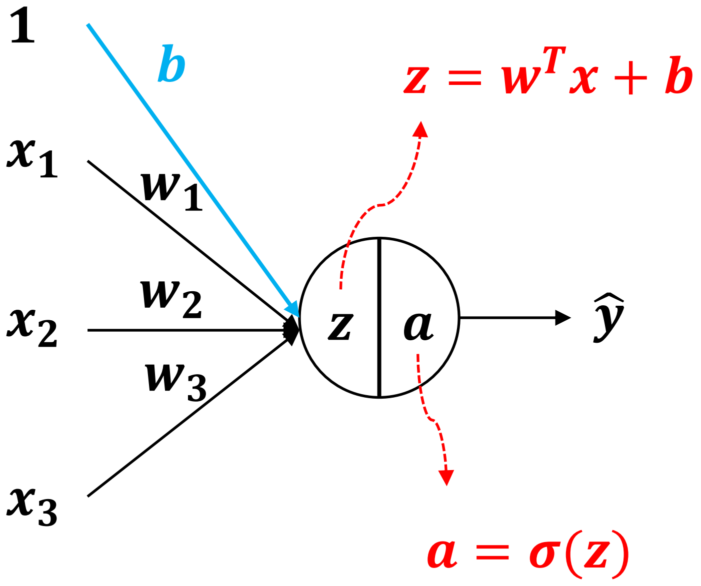




Using the notations in the neural network literature, where $\boldsymbol{\theta} = \boldsymbol{w} = [\boldsymbol{w_1}, \boldsymbol{w_2}, \boldsymbol{w_3}]$ ($\boldsymbol{w_0}$ is not part of this vector here), $\boldsymbol{h_\theta} = \hat{\boldsymbol{y}}$, and $\boldsymbol{\theta_0} = \boldsymbol{w_0} = \boldsymbol{b}$

Towards Neural Networks

Logistic regression can be considered as **neural network** with only 1 neuron



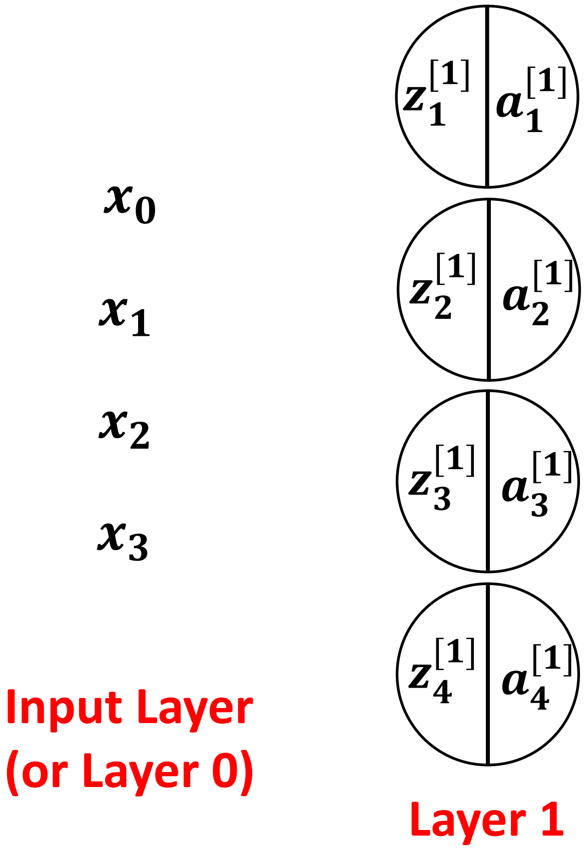
$$\hat{y} = a = \sigma(z) = \sigma\left(w^T x + b\right) = \sigma\left([w_1\ w_2 w_3]\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + b\right)$$

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

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

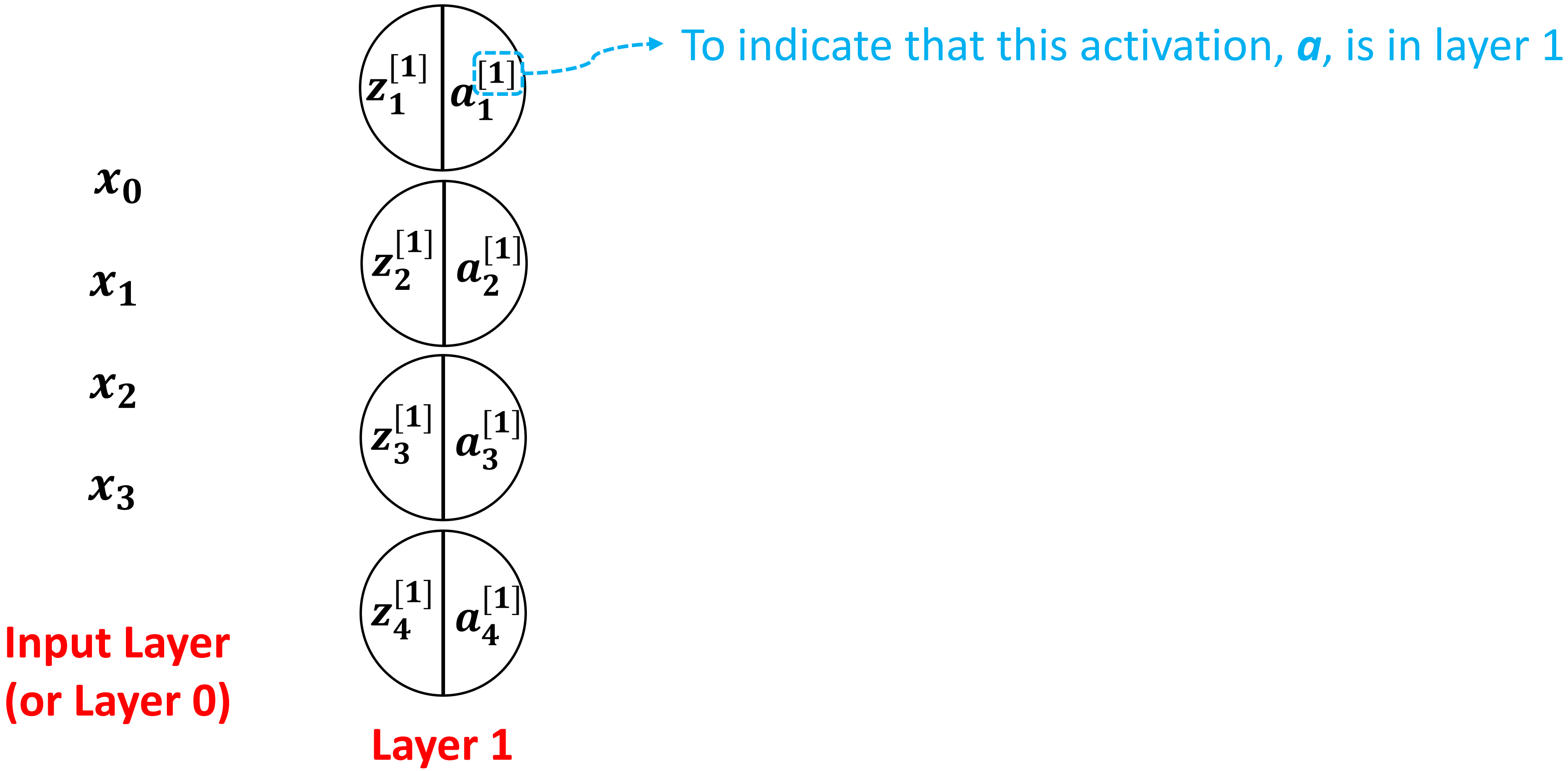
Neural Networks

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



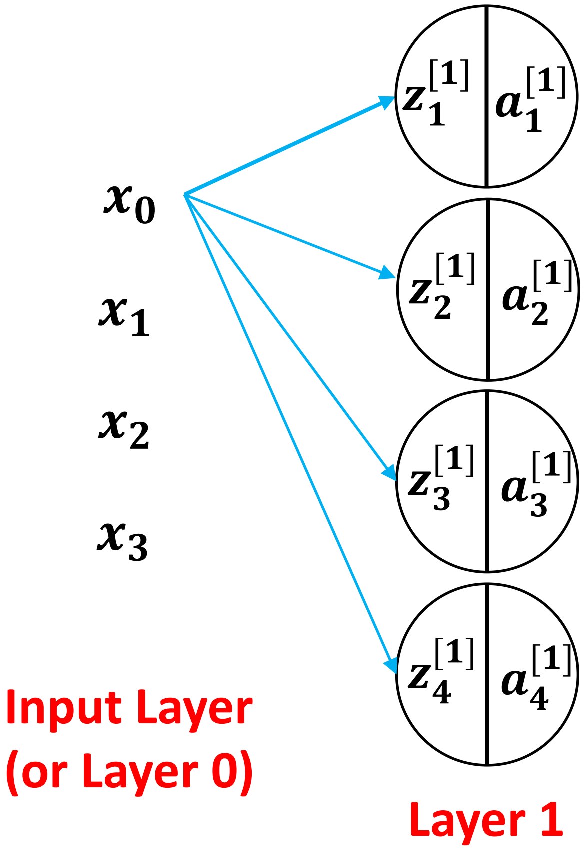
Neural Networks

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



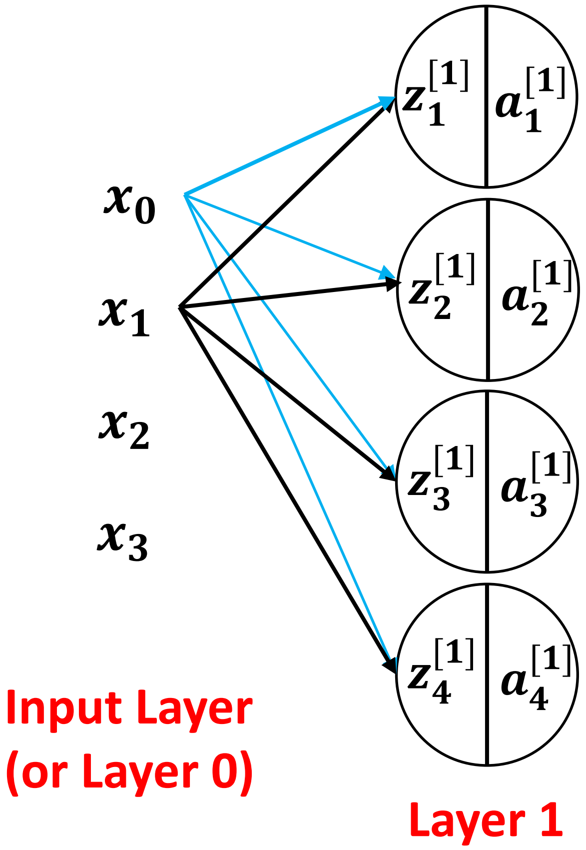
Neural Networks

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



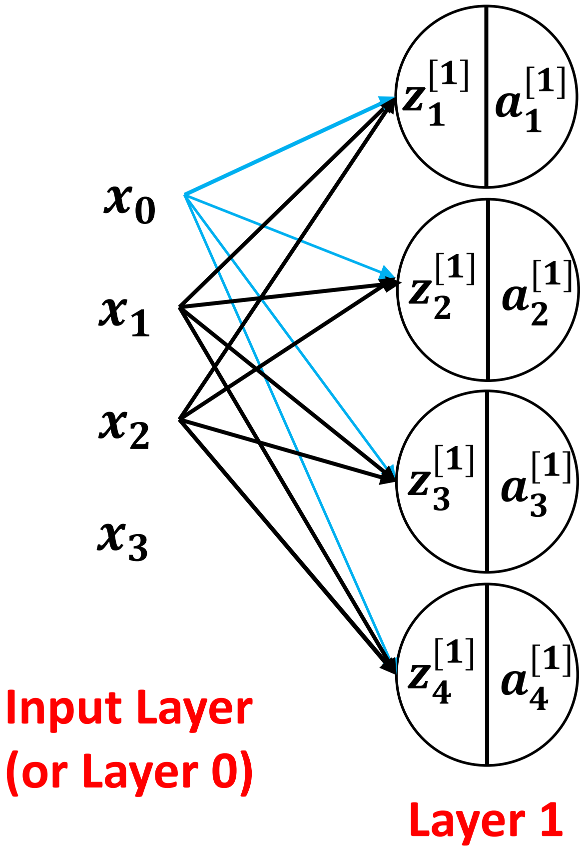
Neural Networks

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



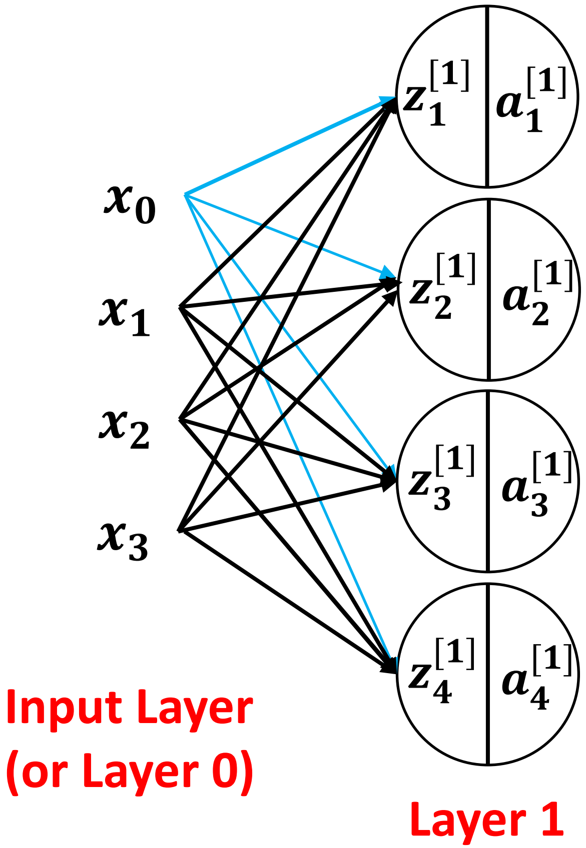
Neural Networks

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



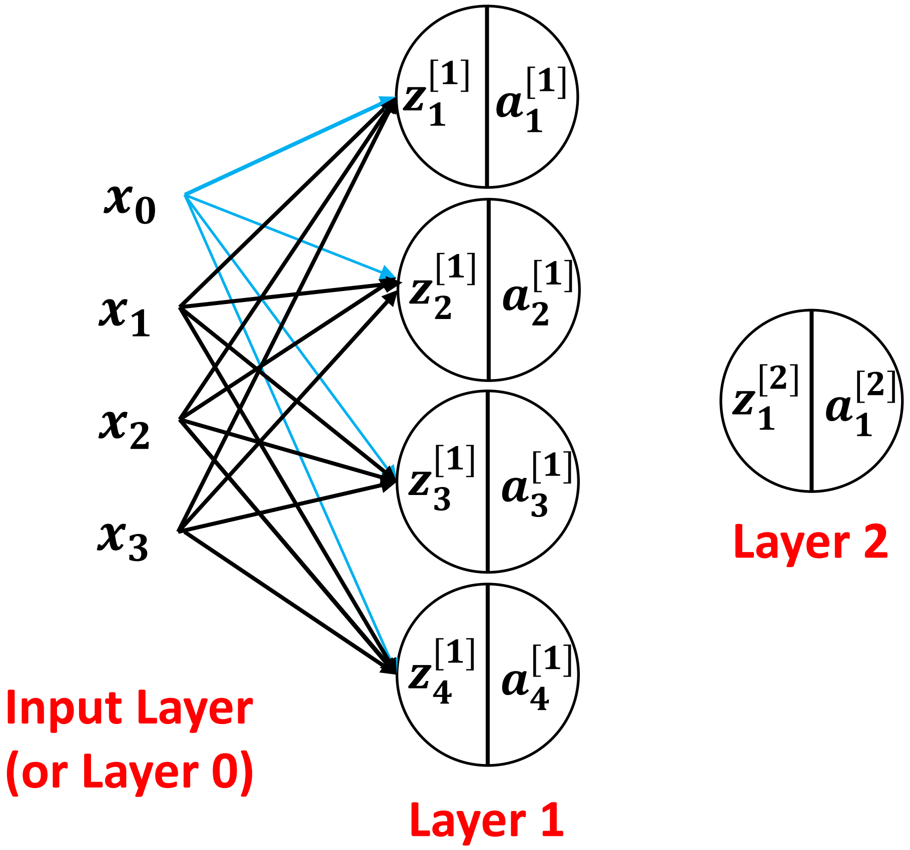
Neural Networks

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



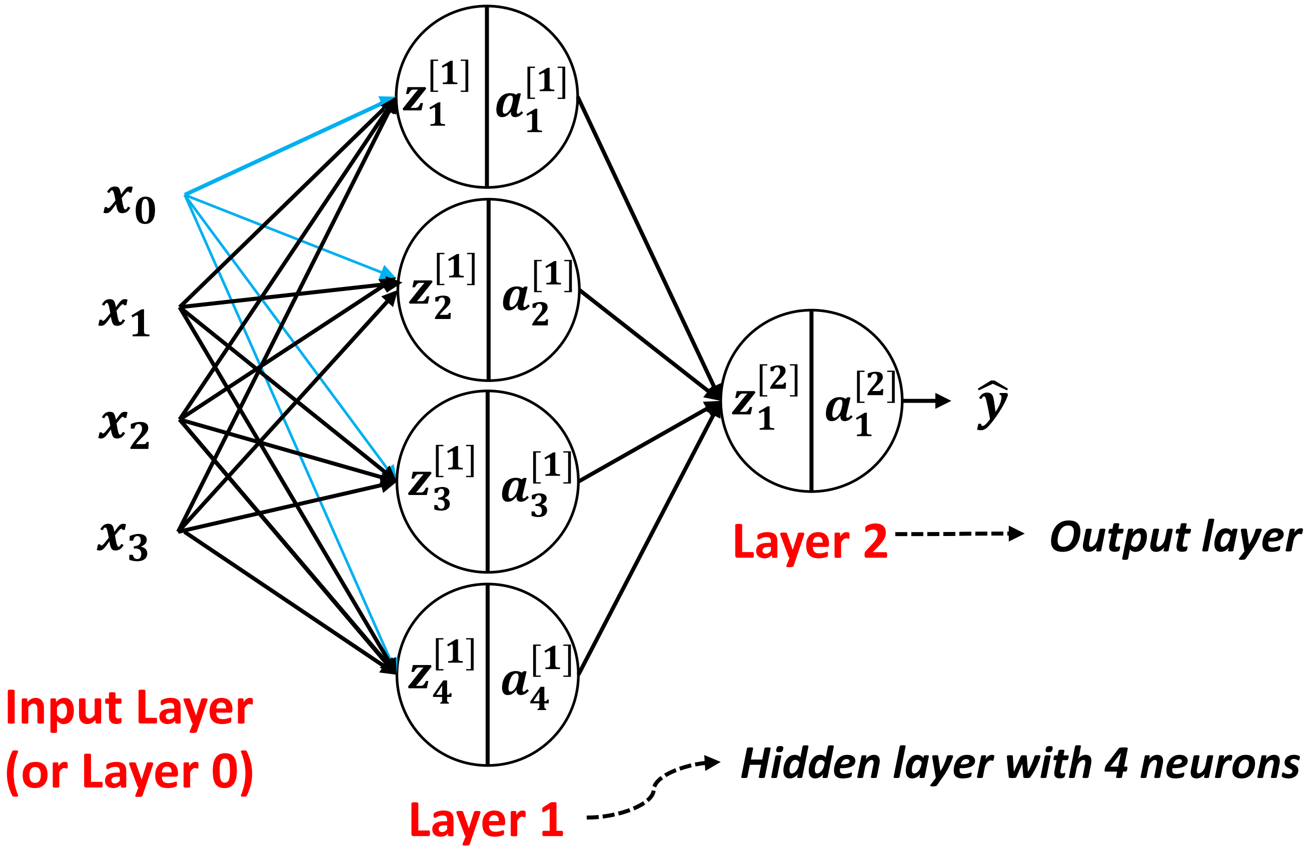
Neural Networks

- 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

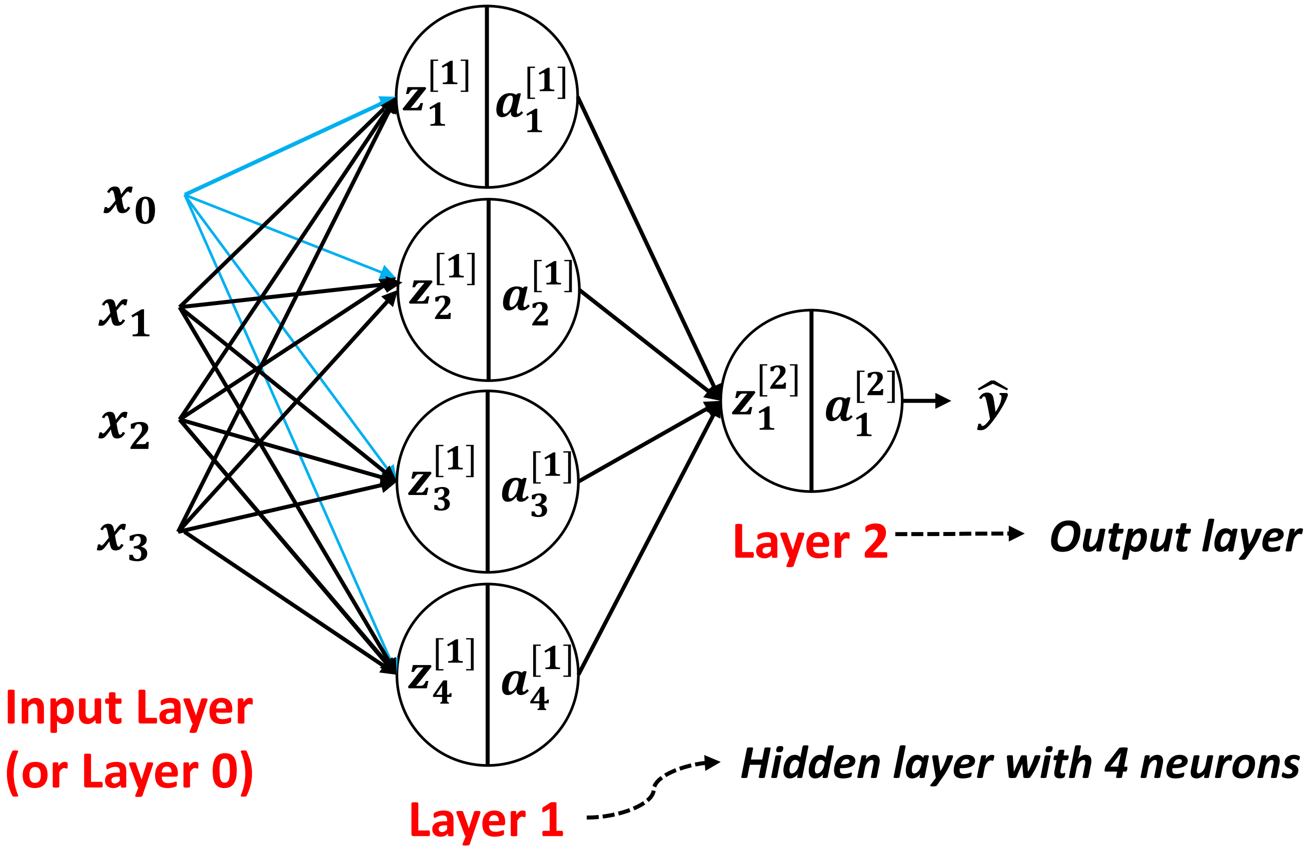
- We can construct a network of neurons (i.e., a neural network) with as many *layers,* and neurons in any layer, as needed



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

Neural Networks

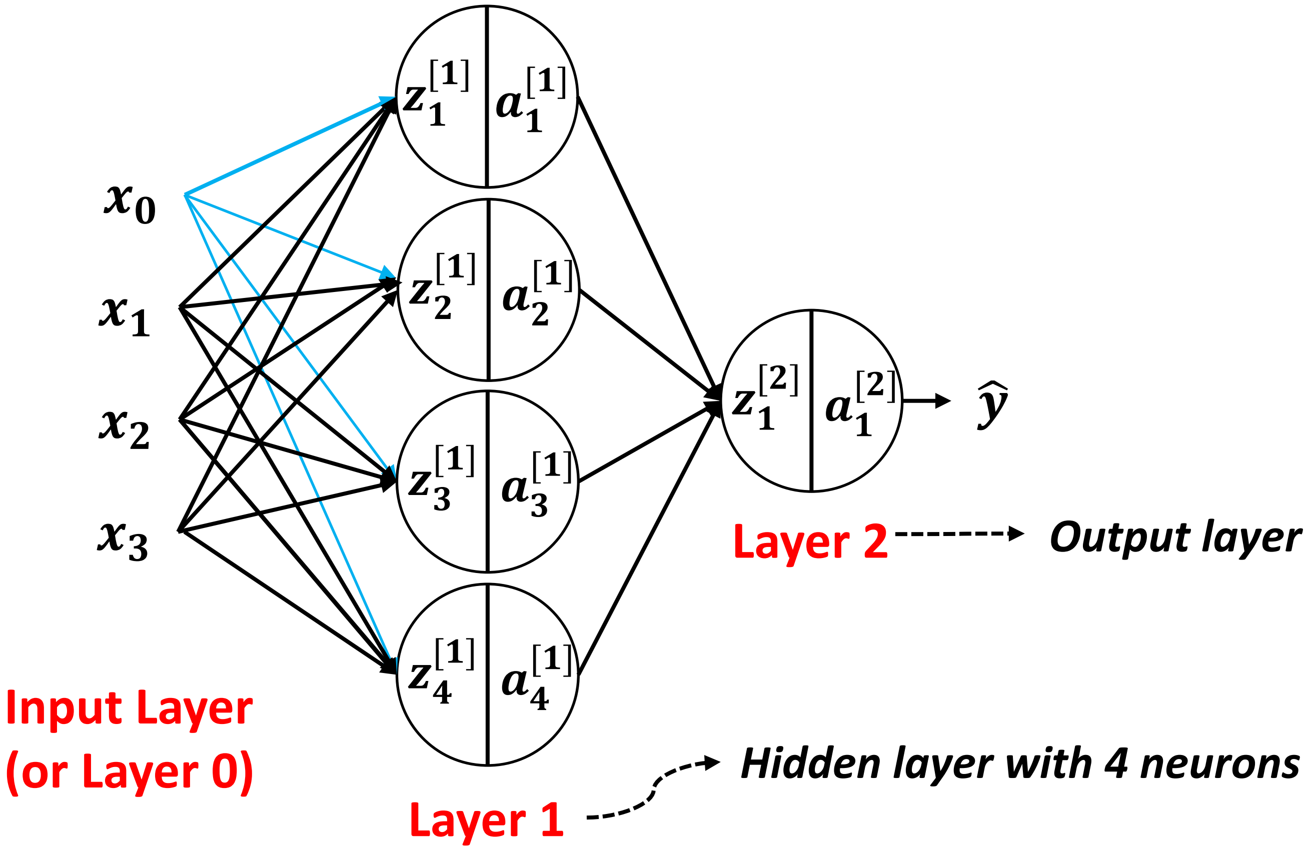
- We can construct a network of neurons (i.e., a neural network) with as many *layers,* and neurons in any layer, as needed



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

Neural Networks

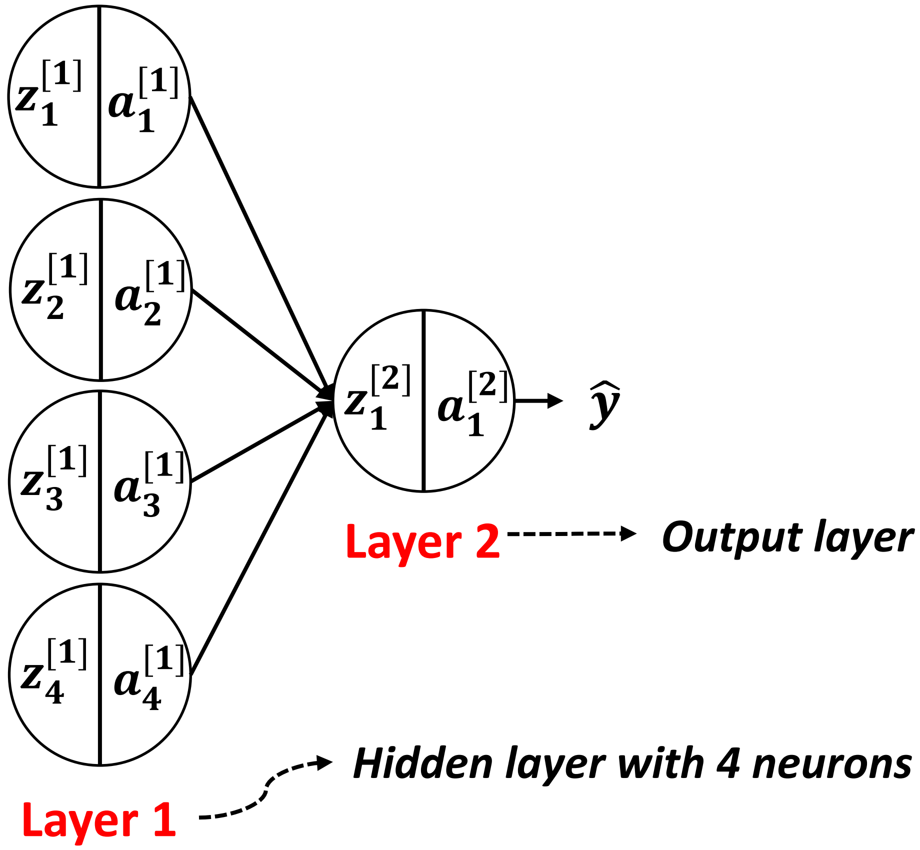
- We can construct a network of neurons (i.e., a neural network) with as many *layers,* and neurons in any layer, as needed



Interestingly, neural networks *learn* their own features!

Neural Networks

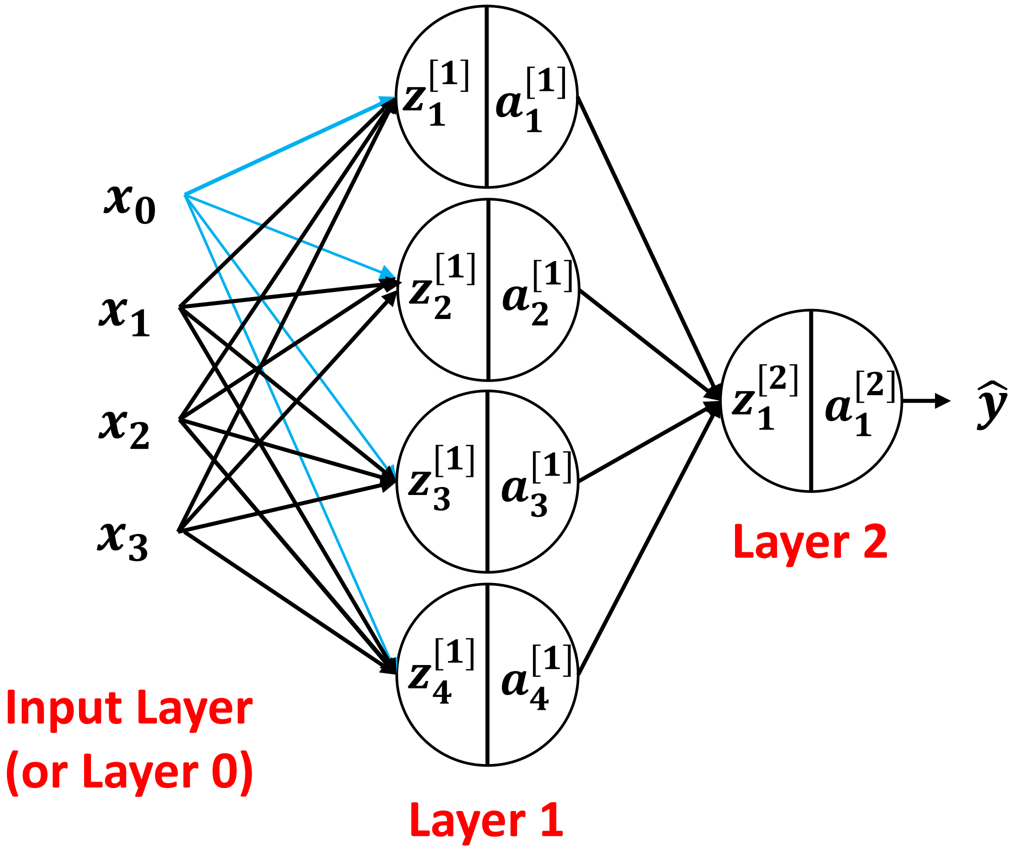
- We can construct a network of neurons (i.e., a neural network) with as many *layers,* and neurons in any layer, as needed



This looks like logistic regression, but with ***features that were learnt*** (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, and x_3)

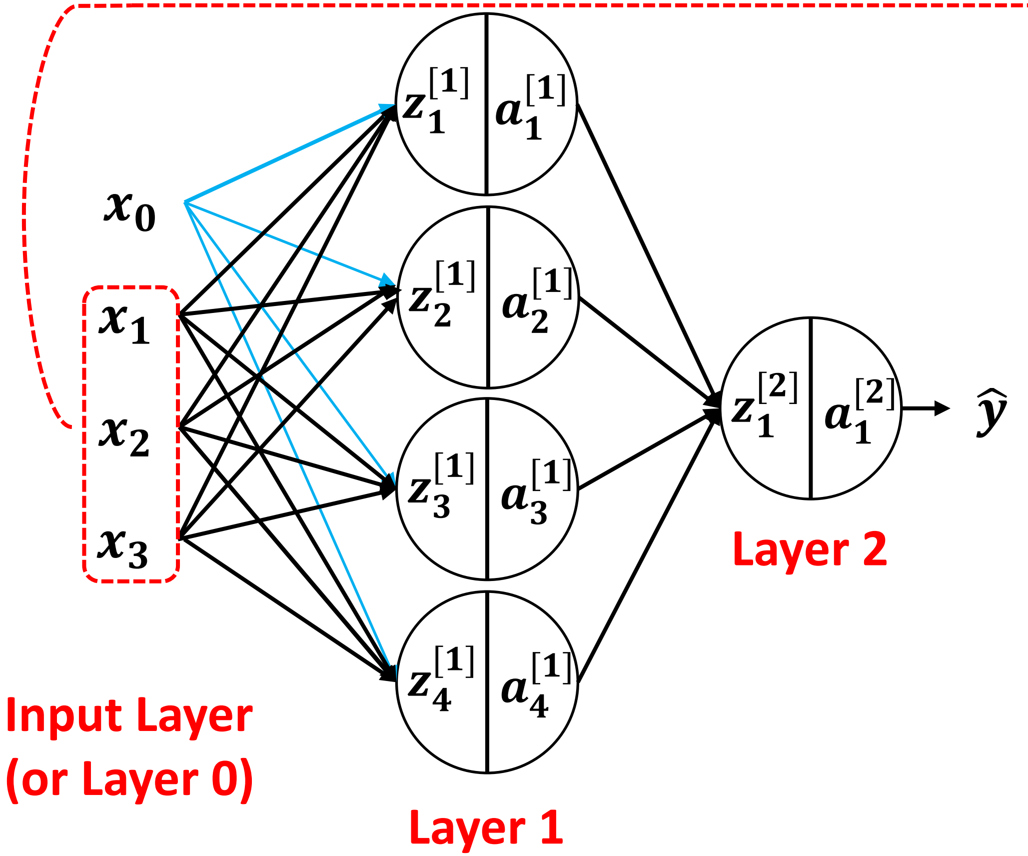
Vectorizing Input and All Variables

- 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

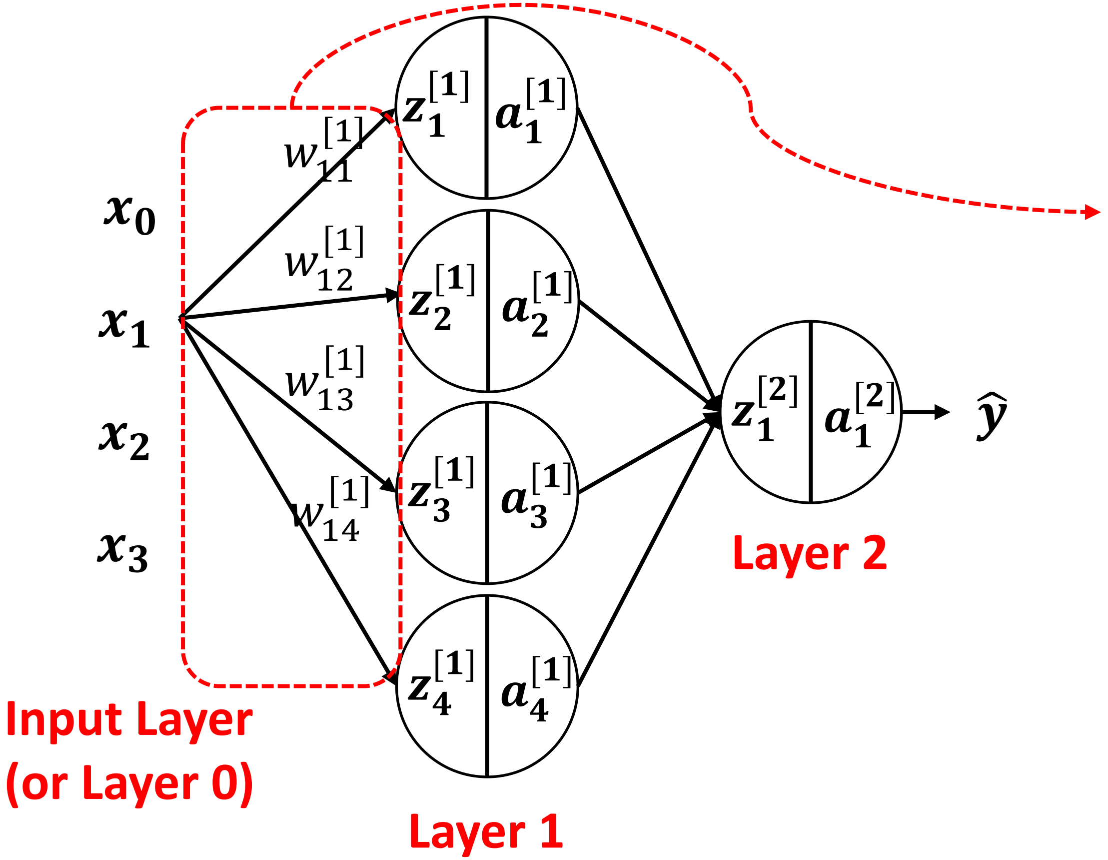
- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Vectorizing Input and All Variables

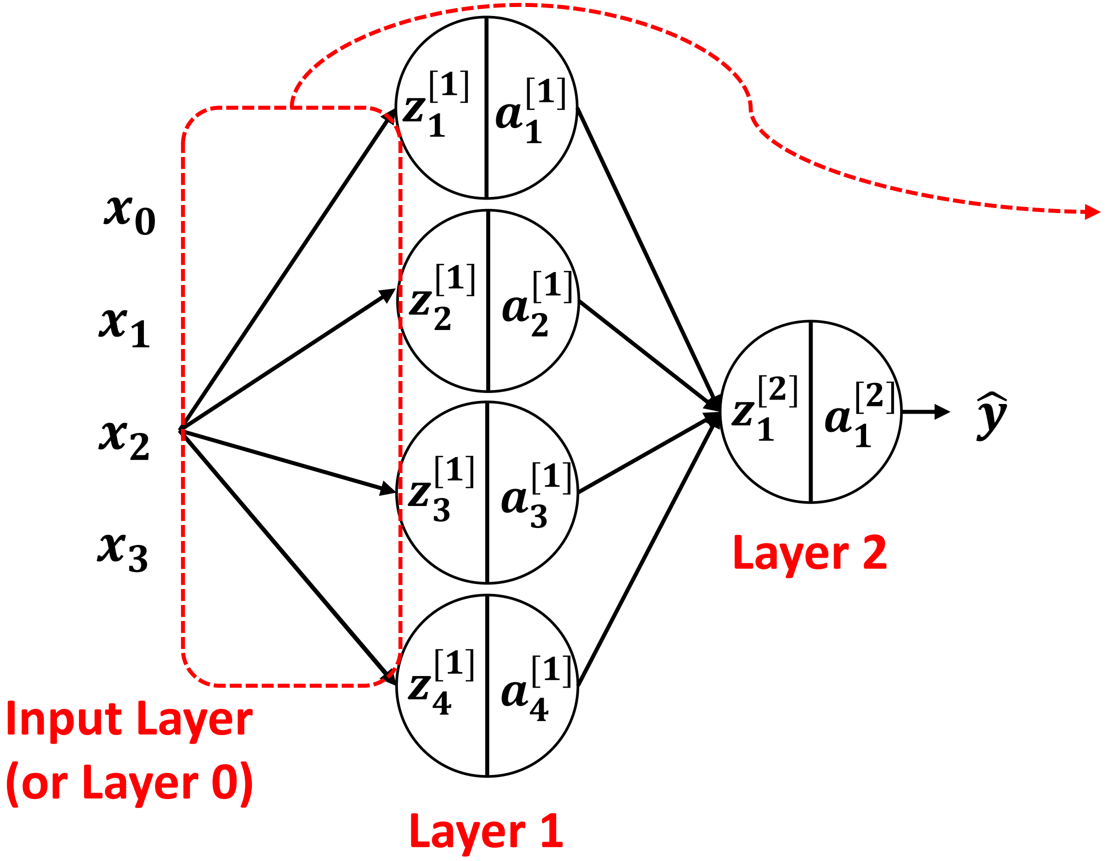
- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$\boldsymbol{w}^{[1]} = \begin{bmatrix} w_{11}^{[1]} & w_{12}^{[1]} & w_{13}^{[1]} & w_{14}^{[1]} \\ & & & \\ & & & \end{bmatrix}$$

Vectorizing Input and All Variables

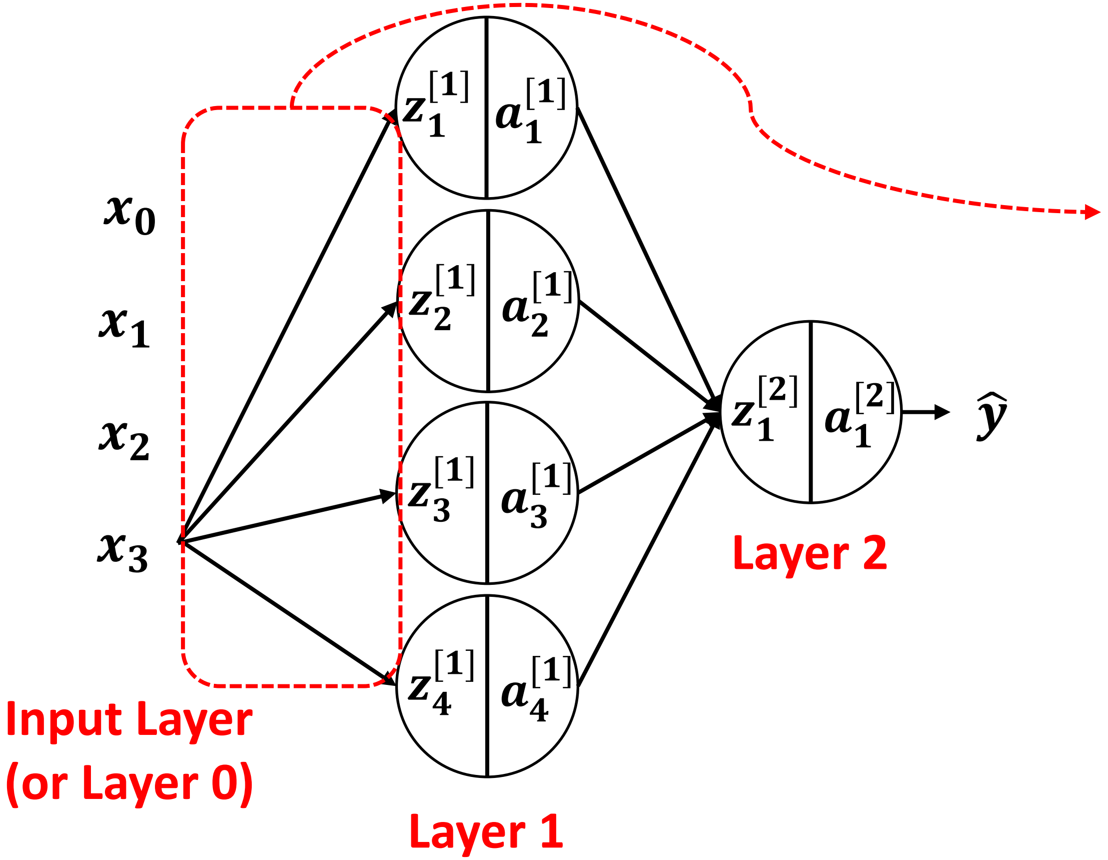
- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$\boldsymbol{w}^{[1]} = \begin{bmatrix} w_{21}^{[1]} & w_{22}^{[1]} & w_{23}^{[1]} & w_{24}^{[1]} \end{bmatrix}$$

Vectorizing Input and All Variables

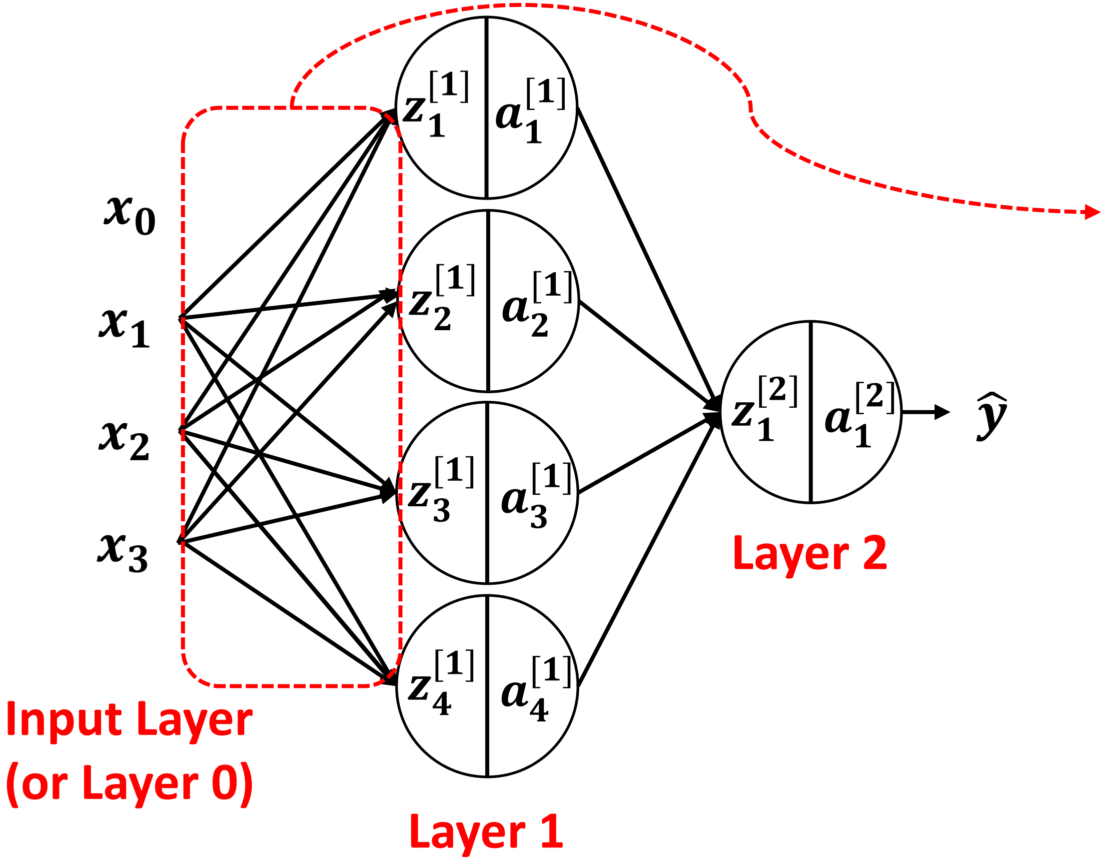
- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$\boldsymbol{w}^{[1]} = \begin{bmatrix} & & & \\ & & & \\ w_{31}^{[1]} & w_{32}^{[1]} & w_{33}^{[1]} & w_{34}^{[1]} \end{bmatrix}$$

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

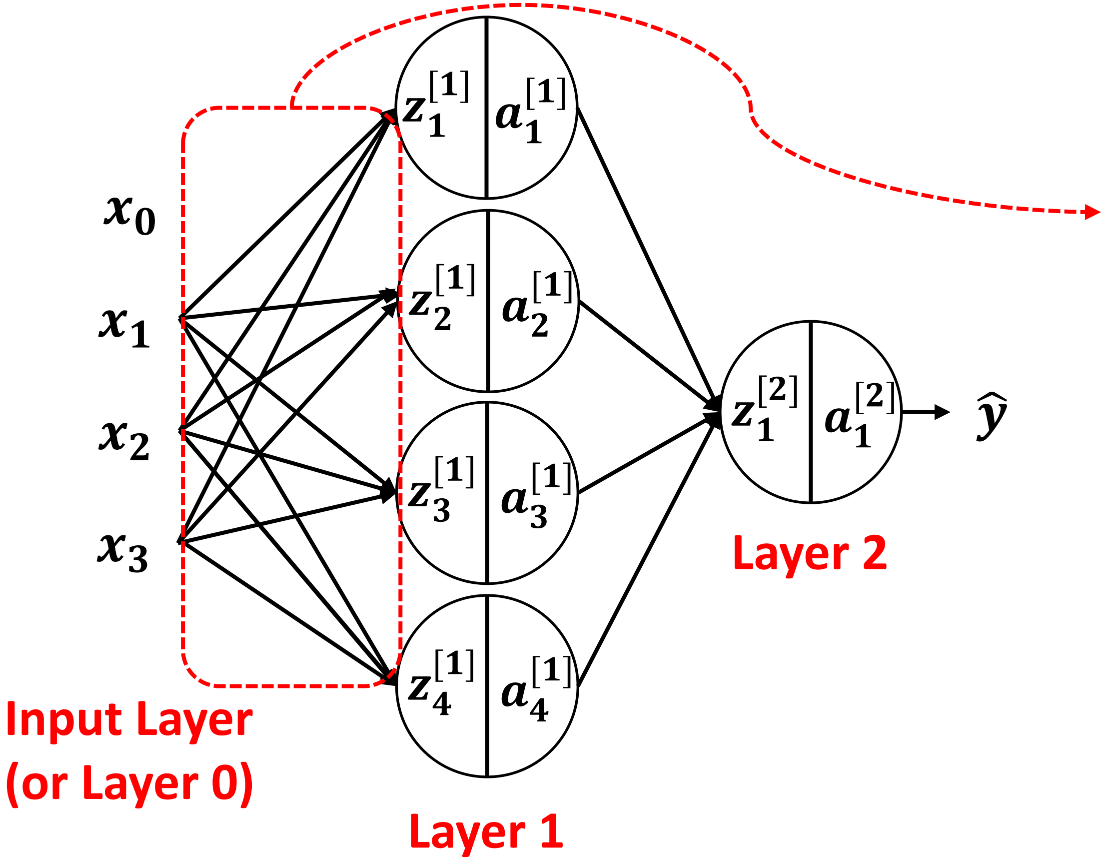


$$\boldsymbol{w}^{[1]} = \begin{bmatrix} w_{11}^{[1]} & w_{12}^{[1]} & w_{13}^{[1]} & w_{14}^{[1]} \\ w_{21}^{[1]} & w_{22}^{[1]} & w_{23}^{[1]} & w_{24}^{[1]} \\ w_{31}^{[1]} & w_{32}^{[1]} & w_{33}^{[1]} & w_{34}^{[1]} \end{bmatrix}$$

Dimension of $\boldsymbol{w}^{[1]}$ = (3, 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

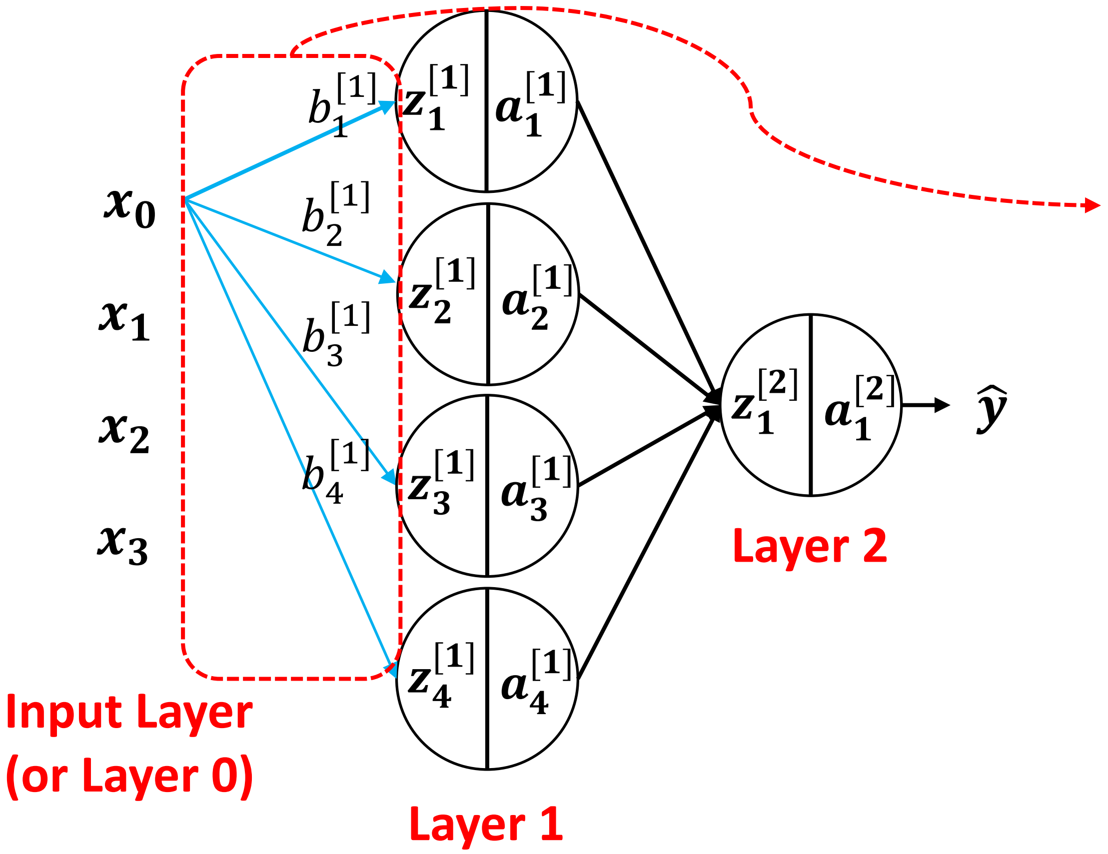


$$w^{[1]^T} = \begin{bmatrix} w_{11}^{[1]} & w_{21}^{[1]} & w_{31}^{[1]} \\ w_{12}^{[1]} & w_{22}^{[1]} & w_{32}^{[1]} \\ w_{13}^{[1]} & w_{23}^{[1]} & w_{33}^{[1]} \\ w_{14}^{[1]} & w_{24}^{[1]} & w_{34}^{[1]} \end{bmatrix}$$

Dimension of $w^{[1]^T}$ = (4, 3)

Vectorizing Input and All Variables

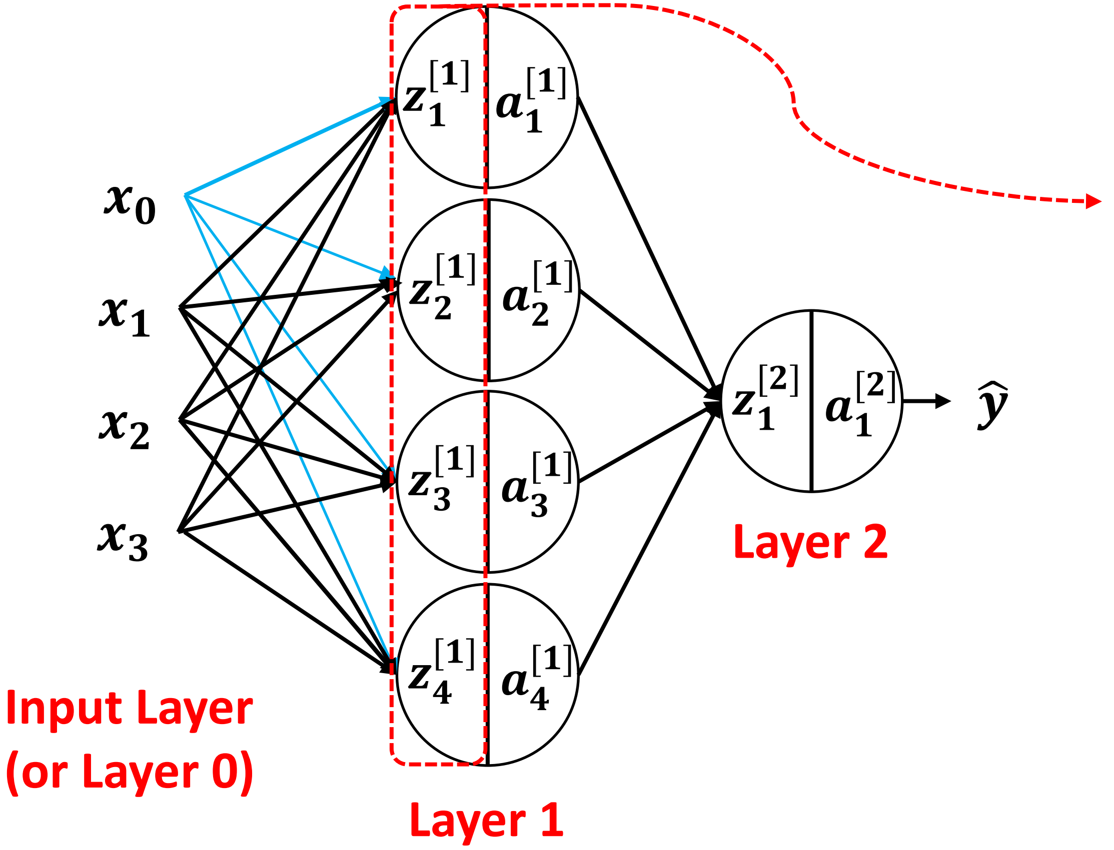
- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$\boldsymbol{b}^{[1]} = \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}$$

Vectorizing Input and All Variables

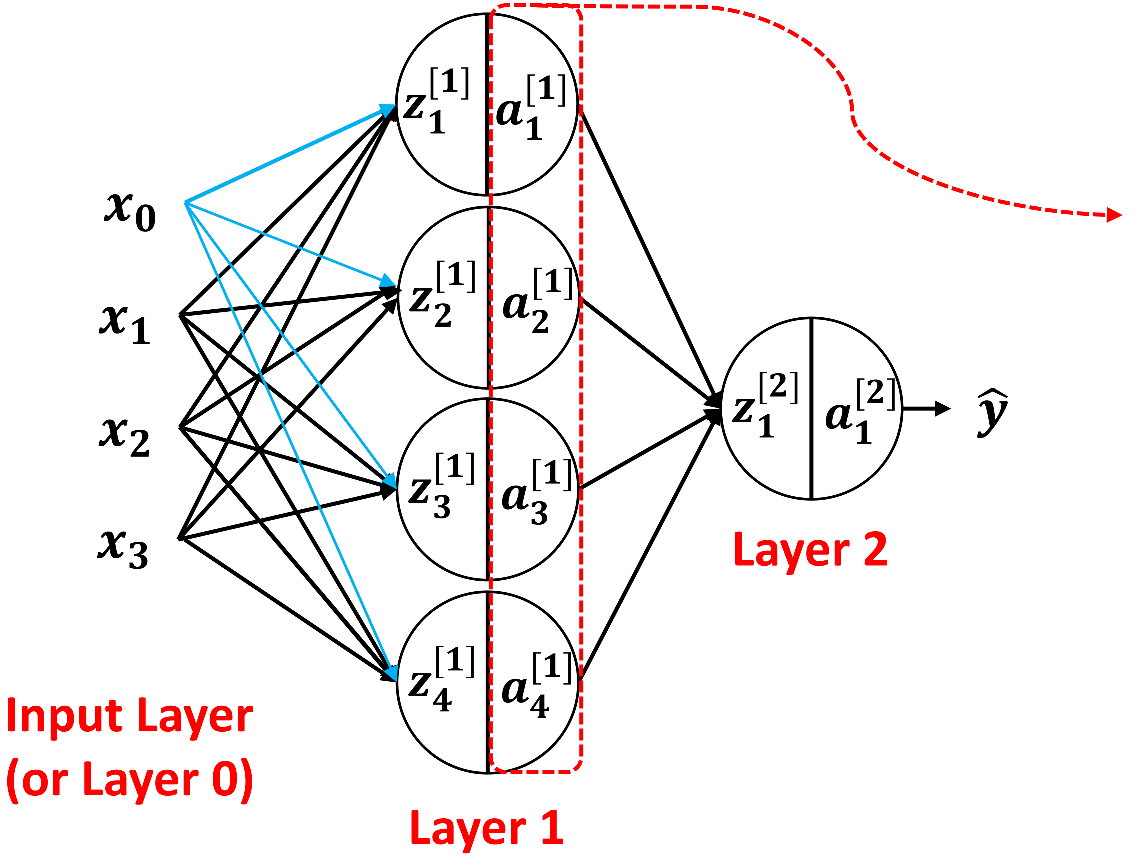
- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$\boldsymbol{z}^{[1]} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}$$

Vectorizing Input and All Variables

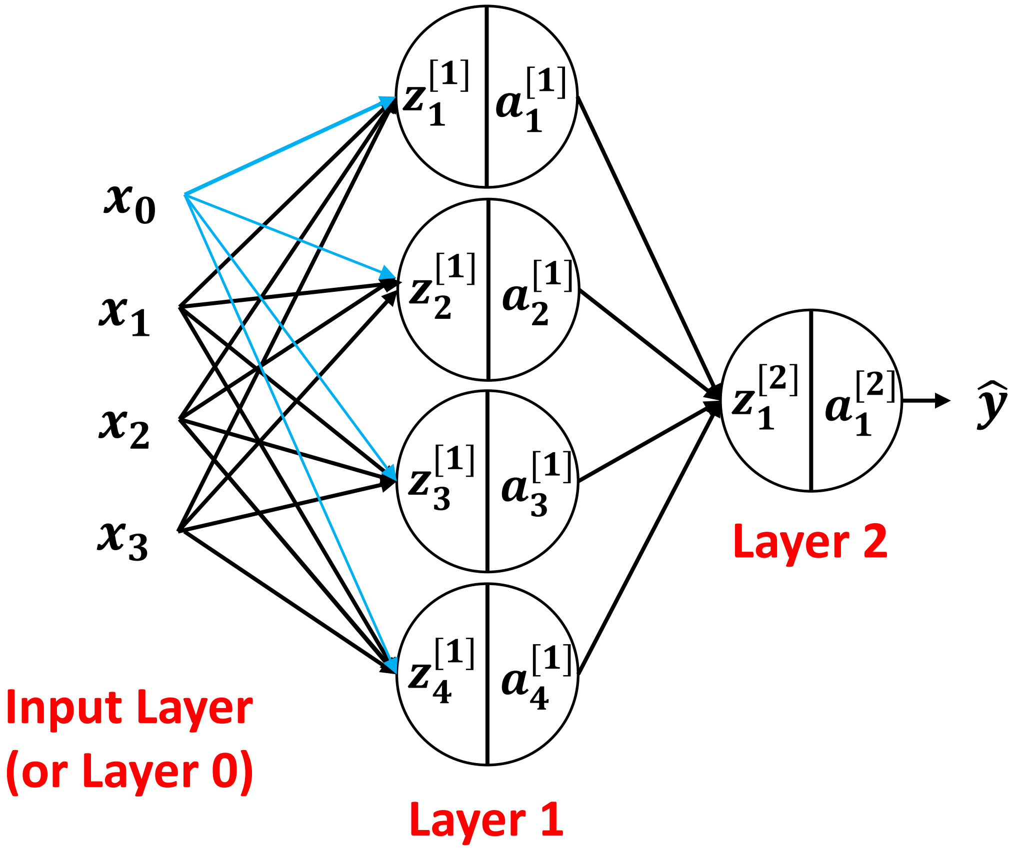
- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$\boldsymbol{a}^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix}$$

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

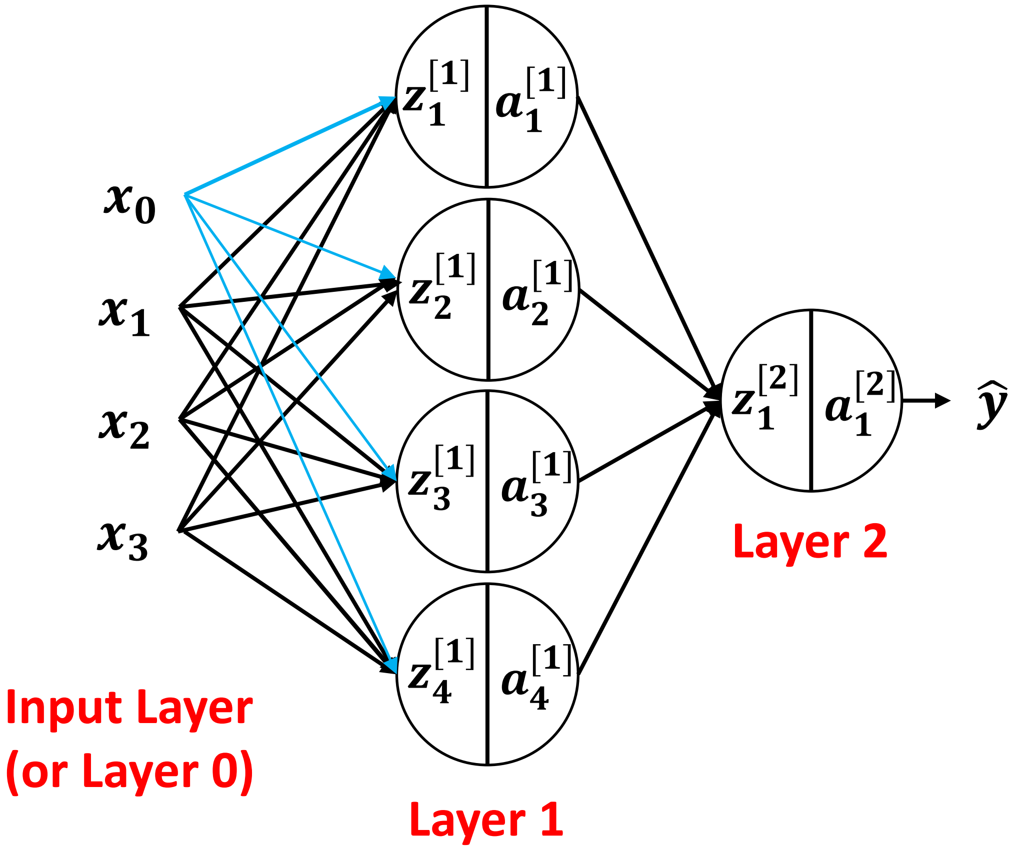


$$\boldsymbol{z}^{[1]} = \boldsymbol{w}^{[1]^T}\boldsymbol{x} + \boldsymbol{b}^{[1]}$$

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

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

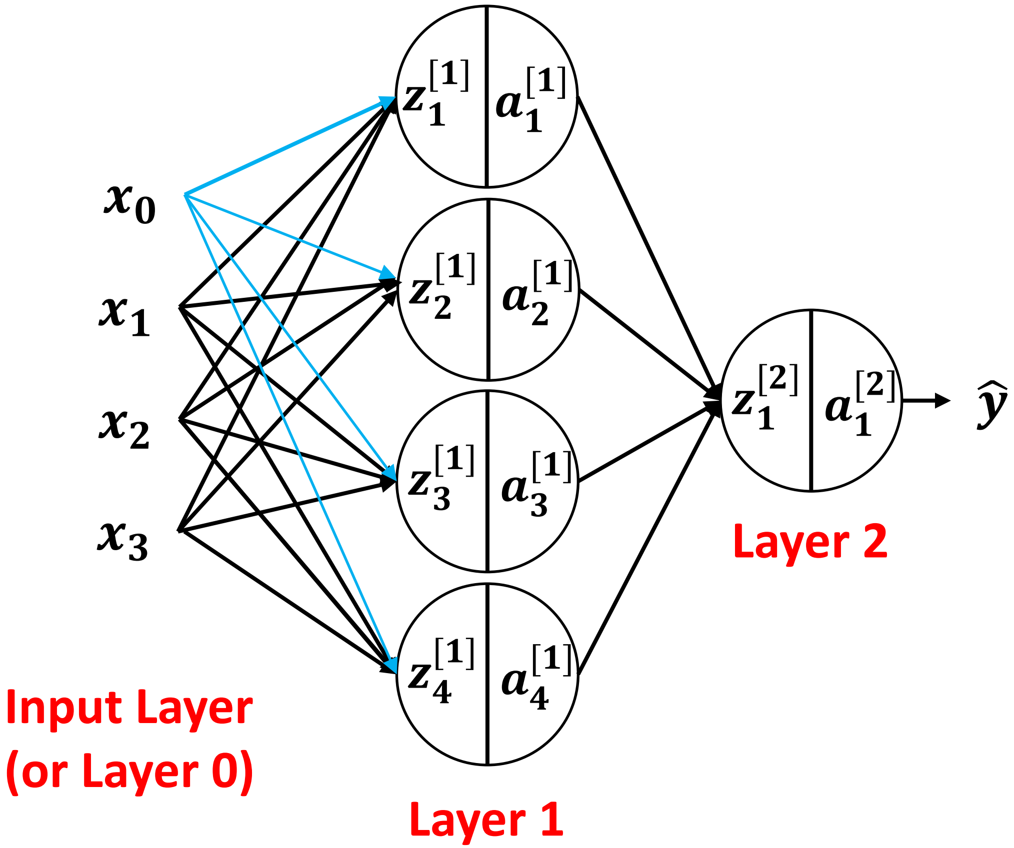


$$\boldsymbol{z}^{[1]} = \boldsymbol{w}^{[1]^T}\boldsymbol{x} + \boldsymbol{b}^{[1]}$$

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

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



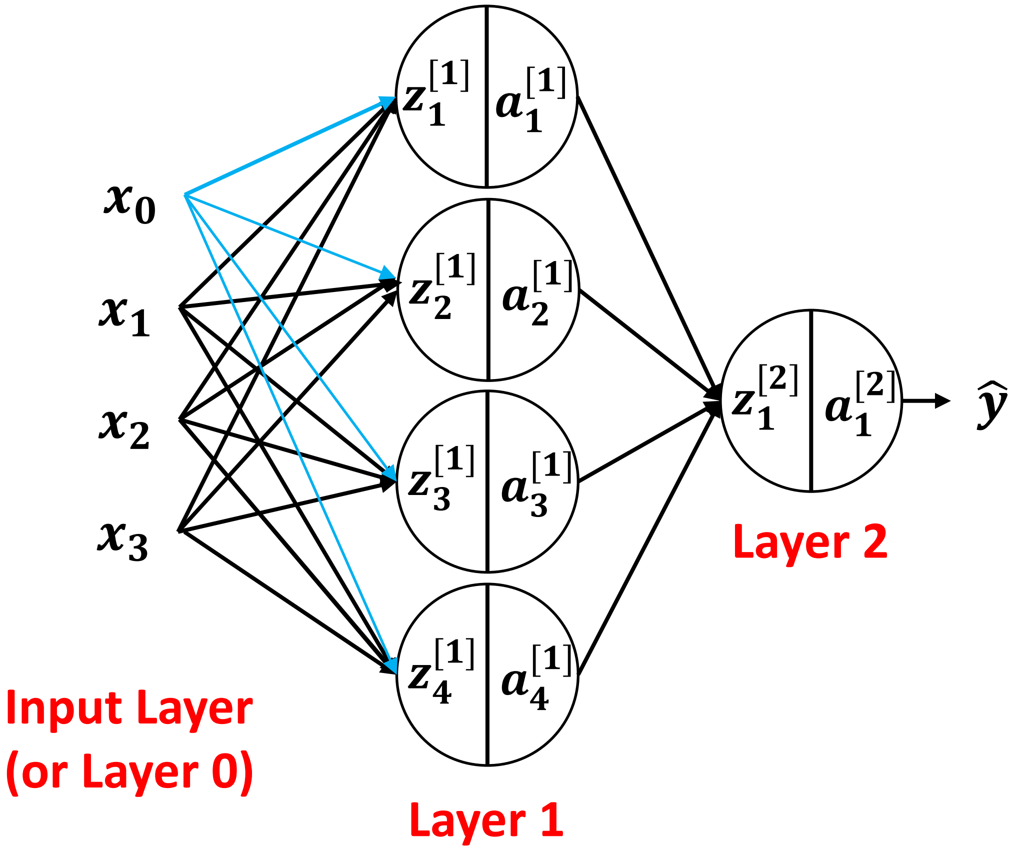
$$\boldsymbol{z}^{[1]} = \boldsymbol{w}^{[1]^T}\boldsymbol{x} + \boldsymbol{b}^{[1]}$$

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

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

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



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

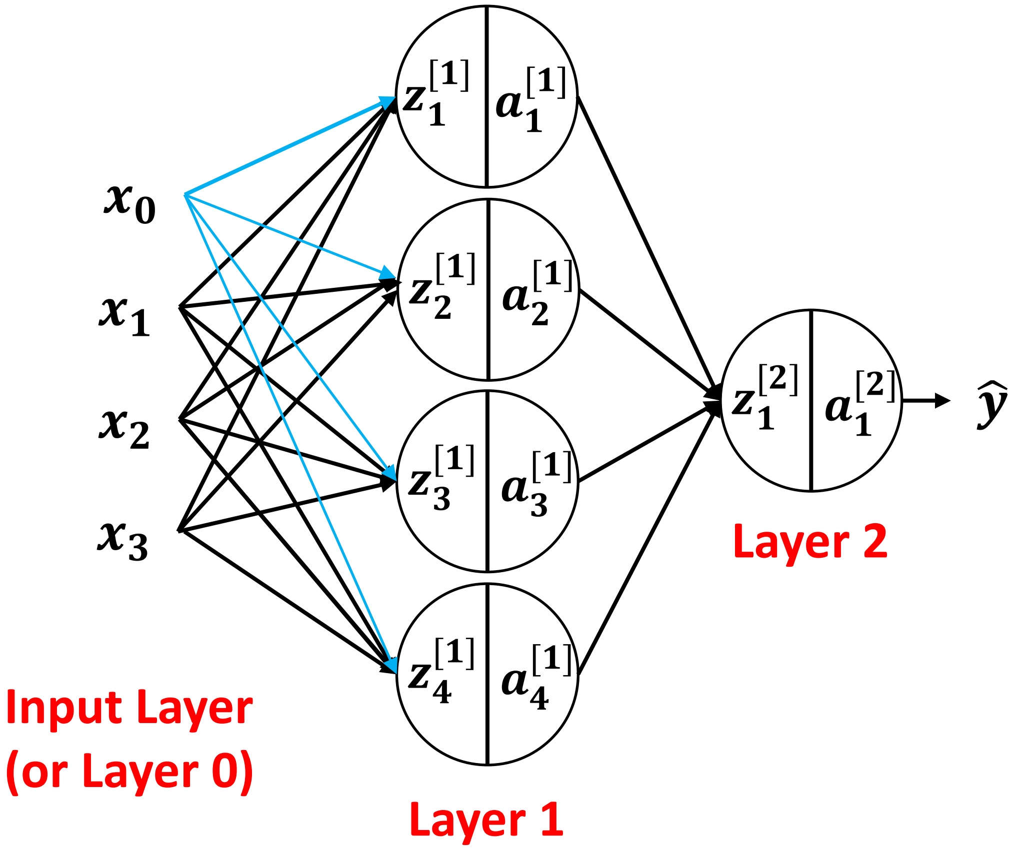
$$= \begin{bmatrix} \boxed{w_{11}^{[1]}x_1 + w_{21}^{[1]}x_2 + w_{31}^{[1]}x_3 + b_1^{[1]}} \\ w_{12}^{[1]}x_1 + w_{22}^{[1]}x_2 + w_{32}^{[1]}x_3 + b_2^{[1]} \\ w_{13}^{[1]}x_1 + w_{23}^{[1]}x_2 + w_{33}^{[1]}x_3 + b_3^{[1]} \\ w_{14}^{[1]}x_1 + w_{24}^{[1]}x_2 + w_{34}^{[1]}x_3 + b_4^{[1]} \end{bmatrix}$$

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

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

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



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

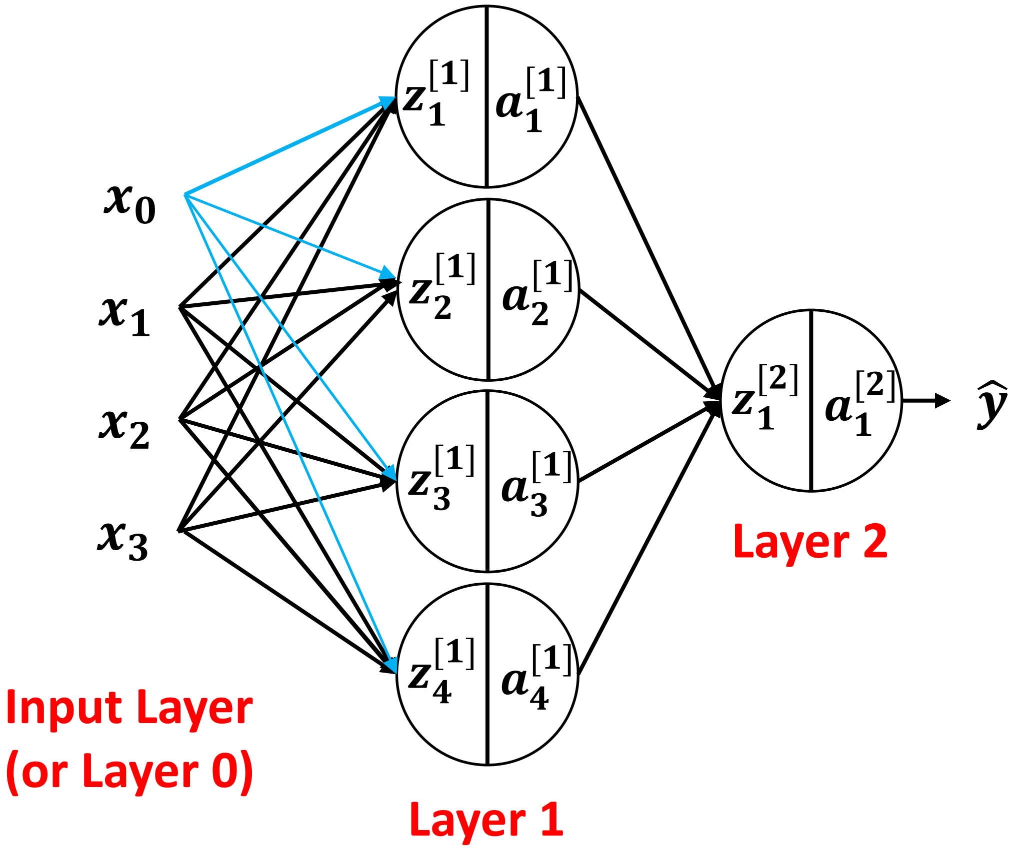
$$= \begin{bmatrix} \boxed{w_{11}^{[1]}x_1 + w_{21}^{[1]}x_2 + w_{31}^{[1]}x_3 + b_1^{[1]}} \\ w_{12}^{[1]}x_1 + w_{22}^{[1]}x_2 + w_{32}^{[1]}x_3 + b_2^{[1]} \\ w_{13}^{[1]}x_1 + w_{23}^{[1]}x_2 + w_{33}^{[1]}x_3 + b_3^{[1]} \\ w_{14}^{[1]}x_1 + w_{24}^{[1]}x_2 + w_{34}^{[1]}x_3 + b_4^{[1]} \end{bmatrix}$$

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

$$\boldsymbol{a}_1^{[1]} = \boldsymbol{\sigma}(\boldsymbol{z}_1^{[1]})$$

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



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

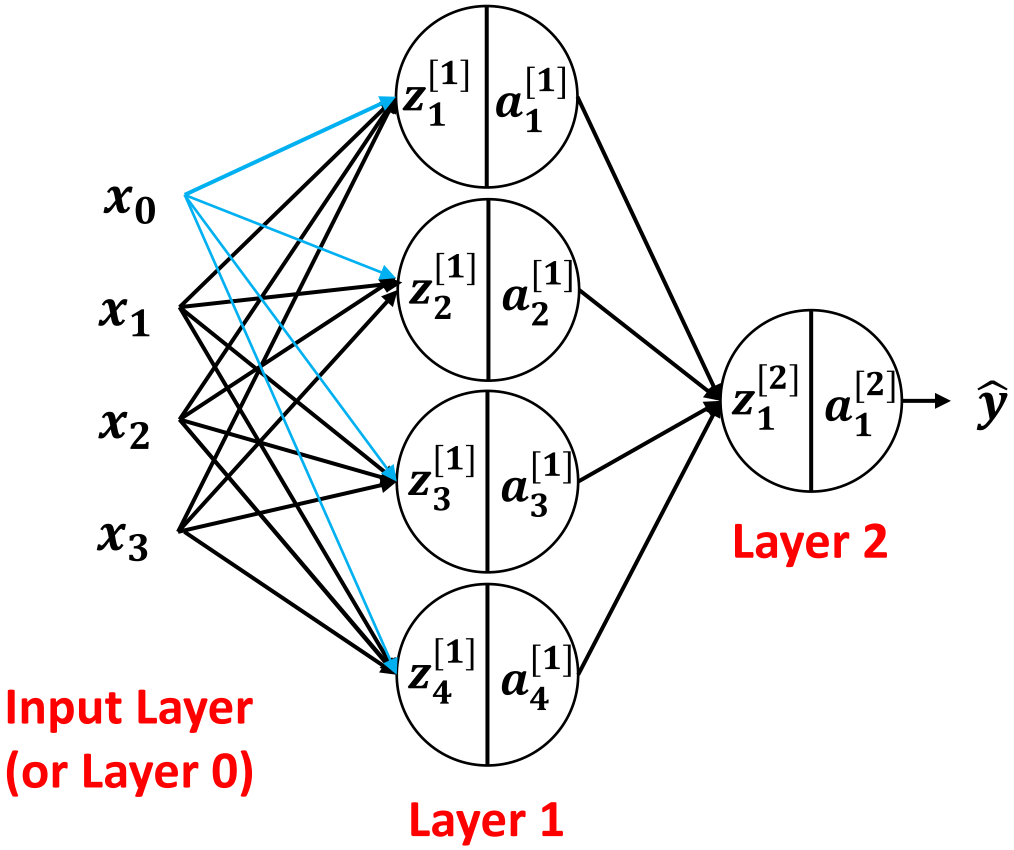
$$= \begin{bmatrix} w_{11}^{[1]}x_1 + w_{21}^{[1]}x_2 + w_{31}^{[1]}x_3 + b_1^{[1]} \\ \boxed{w_{12}^{[1]}x_1 + w_{22}^{[1]}x_2 + w_{32}^{[1]}x_3 + b_2^{[1]}} \\ w_{13}^{[1]}x_1 + w_{23}^{[1]}x_2 + w_{33}^{[1]}x_3 + b_3^{[1]} \\ w_{14}^{[1]}x_1 + w_{24}^{[1]}x_2 + w_{34}^{[1]}x_3 + b_4^{[1]} \end{bmatrix}$$

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

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

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



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

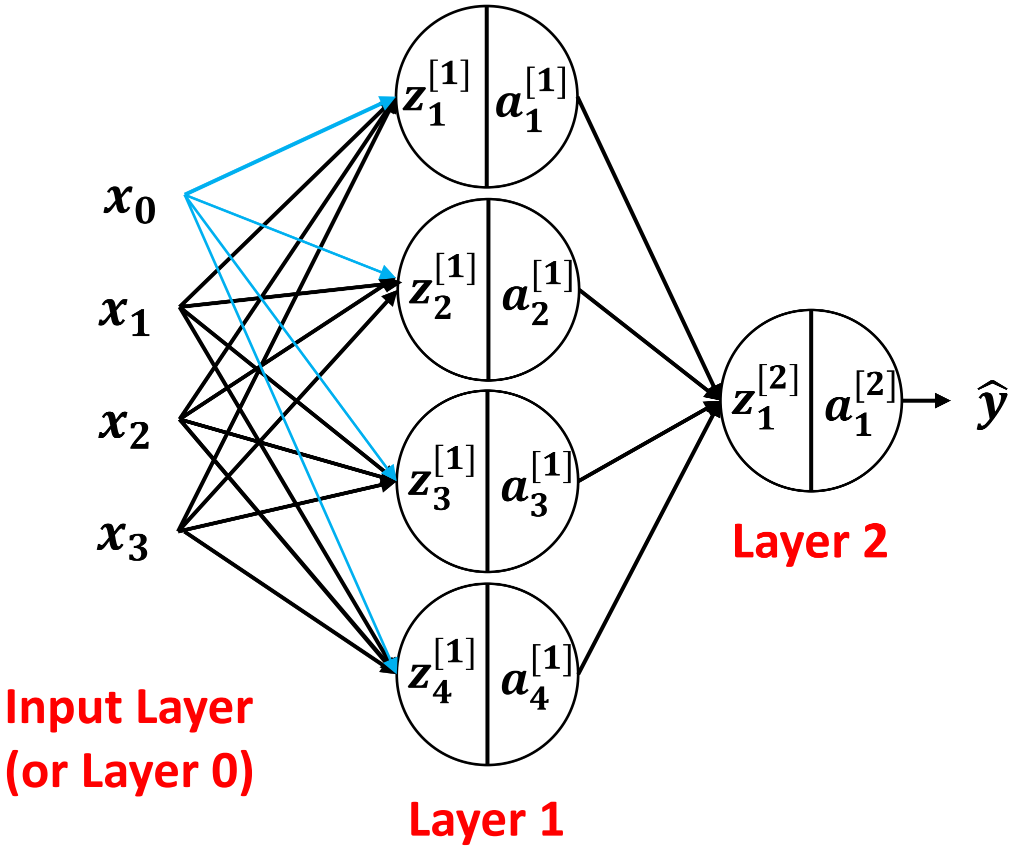
$$= \begin{bmatrix} w_{11}^{[1]}x_1 + w_{21}^{[1]}x_2 + w_{31}^{[1]}x_3 + b_1^{[1]} \\ \boxed{w_{12}^{[1]}x_1 + w_{22}^{[1]}x_2 + w_{32}^{[1]}x_3 + b_2^{[1]}} \\ w_{13}^{[1]}x_1 + w_{23}^{[1]}x_2 + w_{33}^{[1]}x_3 + b_3^{[1]} \\ w_{14}^{[1]}x_1 + w_{24}^{[1]}x_2 + w_{34}^{[1]}x_3 + b_4^{[1]} \end{bmatrix}$$

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

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

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



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

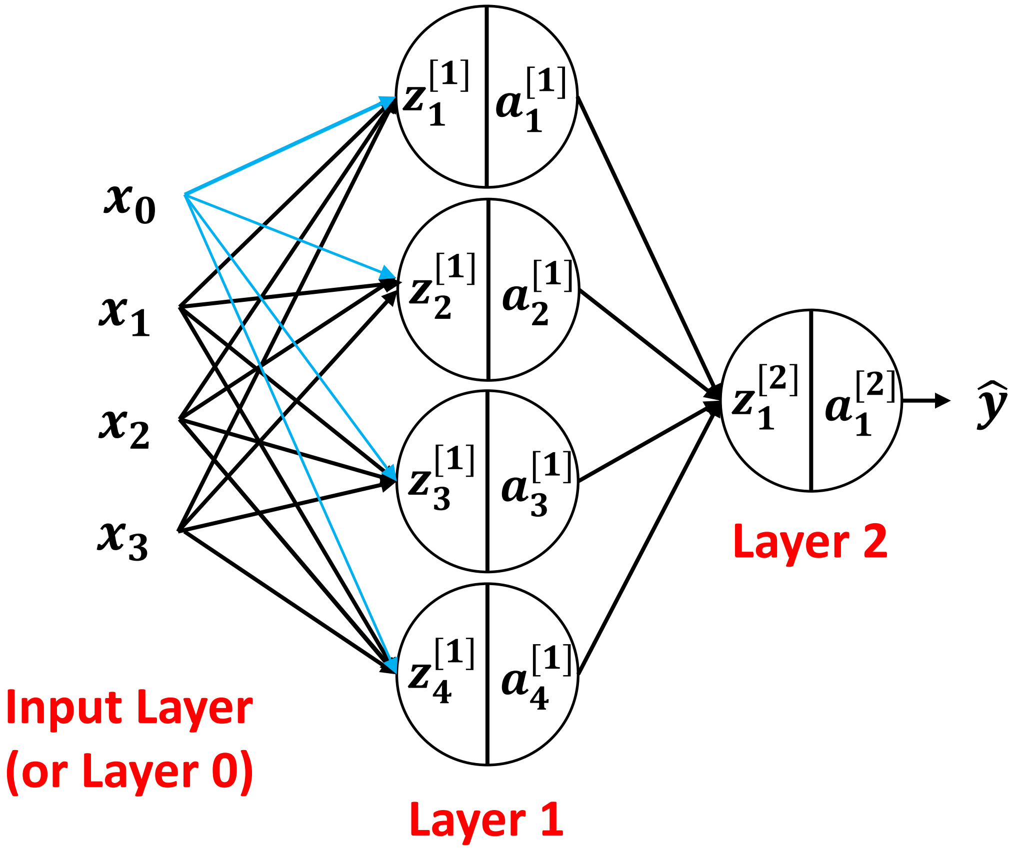
$$= \begin{bmatrix} w_{11}^{[1]}x_1 + w_{21}^{[1]}x_2 + w_{31}^{[1]}x_3 + b_1^{[1]} \\ w_{12}^{[1]}x_1 + w_{22}^{[1]}x_2 + w_{32}^{[1]}x_3 + b_2^{[1]} \\ \boxed{w_{13}^{[1]}x_1 + w_{23}^{[1]}x_2 + w_{33}^{[1]}x_3 + b_3^{[1]}} \\ w_{14}^{[1]}x_1 + w_{24}^{[1]}x_2 + w_{34}^{[1]}x_3 + b_4^{[1]} \end{bmatrix}$$

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

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

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



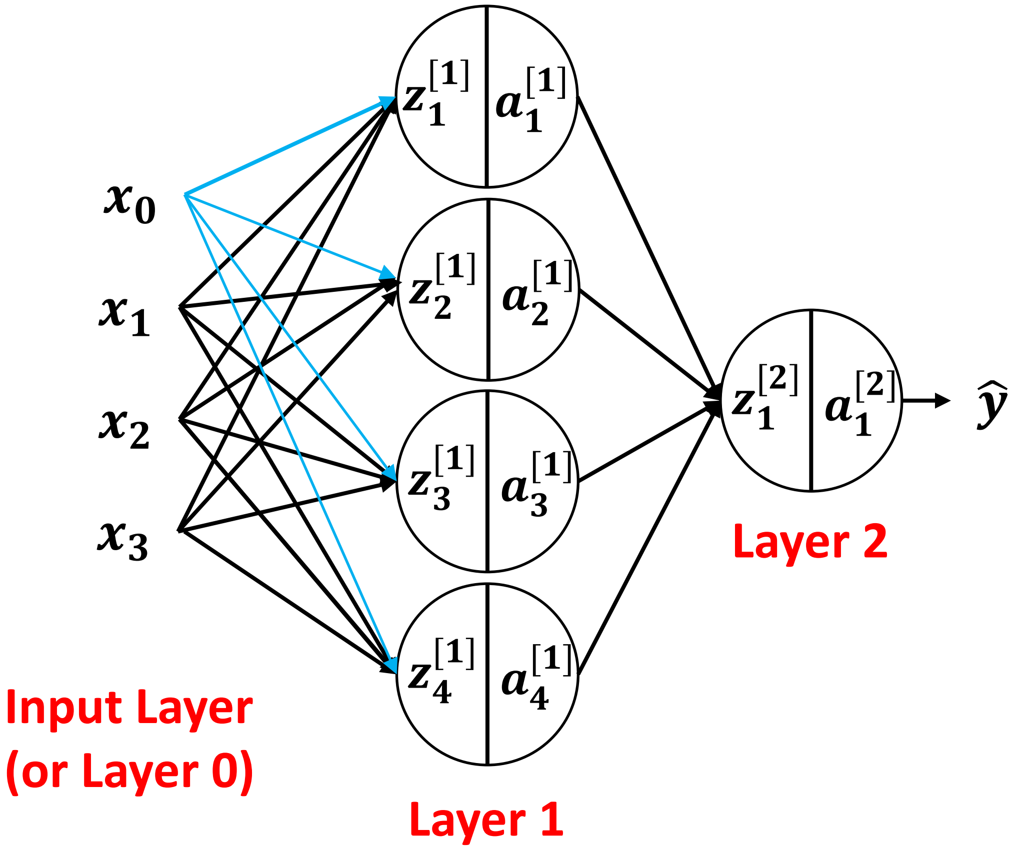
$$\boldsymbol{z}^{[1]} = \boldsymbol{w}^{[1]^T}\boldsymbol{x} + \boldsymbol{b}^{[1]}$$

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

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

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



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

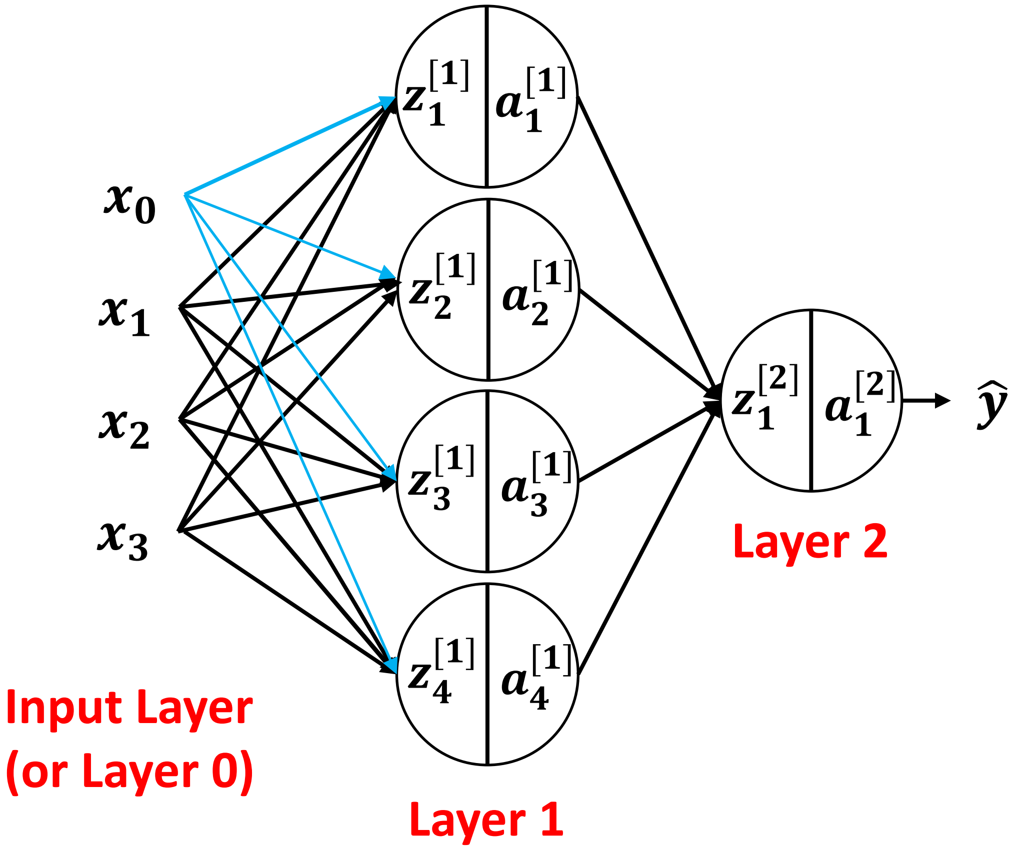
$$= \begin{bmatrix} w_{11}^{[1]}x_1 + w_{21}^{[1]}x_2 + w_{31}^{[1]}x_3 + b_1^{[1]} \\ w_{12}^{[1]}x_1 + w_{22}^{[1]}x_2 + w_{32}^{[1]}x_3 + b_2^{[1]} \\ w_{13}^{[1]}x_1 + w_{23}^{[1]}x_2 + w_{33}^{[1]}x_3 + b_3^{[1]} \\ \boxed{w_{14}^{[1]}x_1 + w_{24}^{[1]}x_2 + w_{34}^{[1]}x_3 + b_4^{[1]}} \end{bmatrix}$$

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

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

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



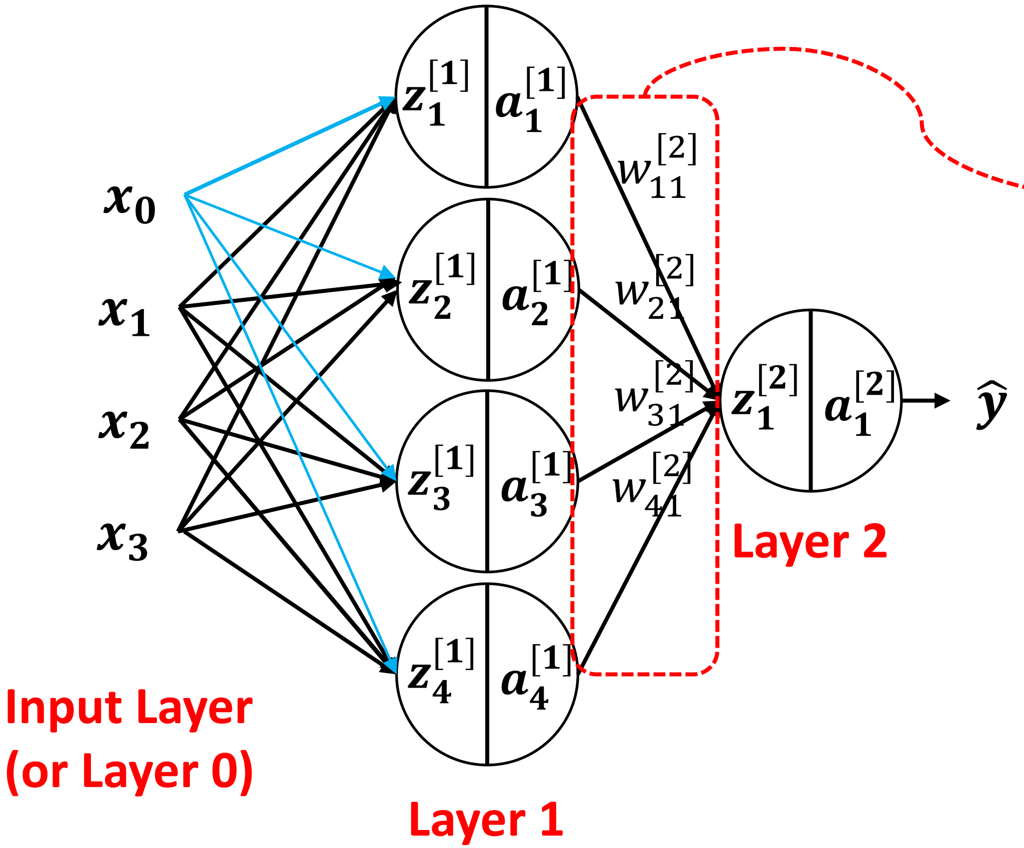
$$\boldsymbol{z}^{[1]} = \boldsymbol{w}^{[1]^T}\boldsymbol{x} + \boldsymbol{b}^{[1]}$$

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

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

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

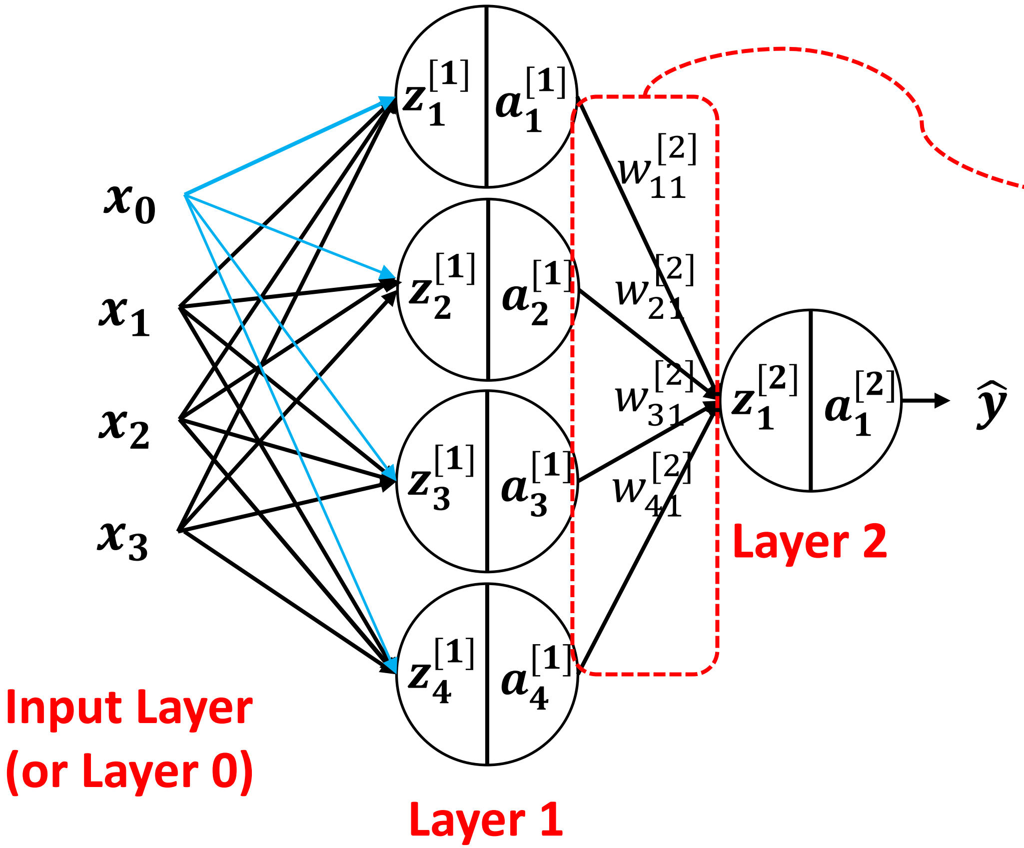


$$\boldsymbol{w}^{[2]} = \begin{bmatrix} w_{11}^{[2]} \\ w_{21}^{[2]} \\ w_{31}^{[2]} \\ w_{41}^{[2]} \end{bmatrix}$$

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

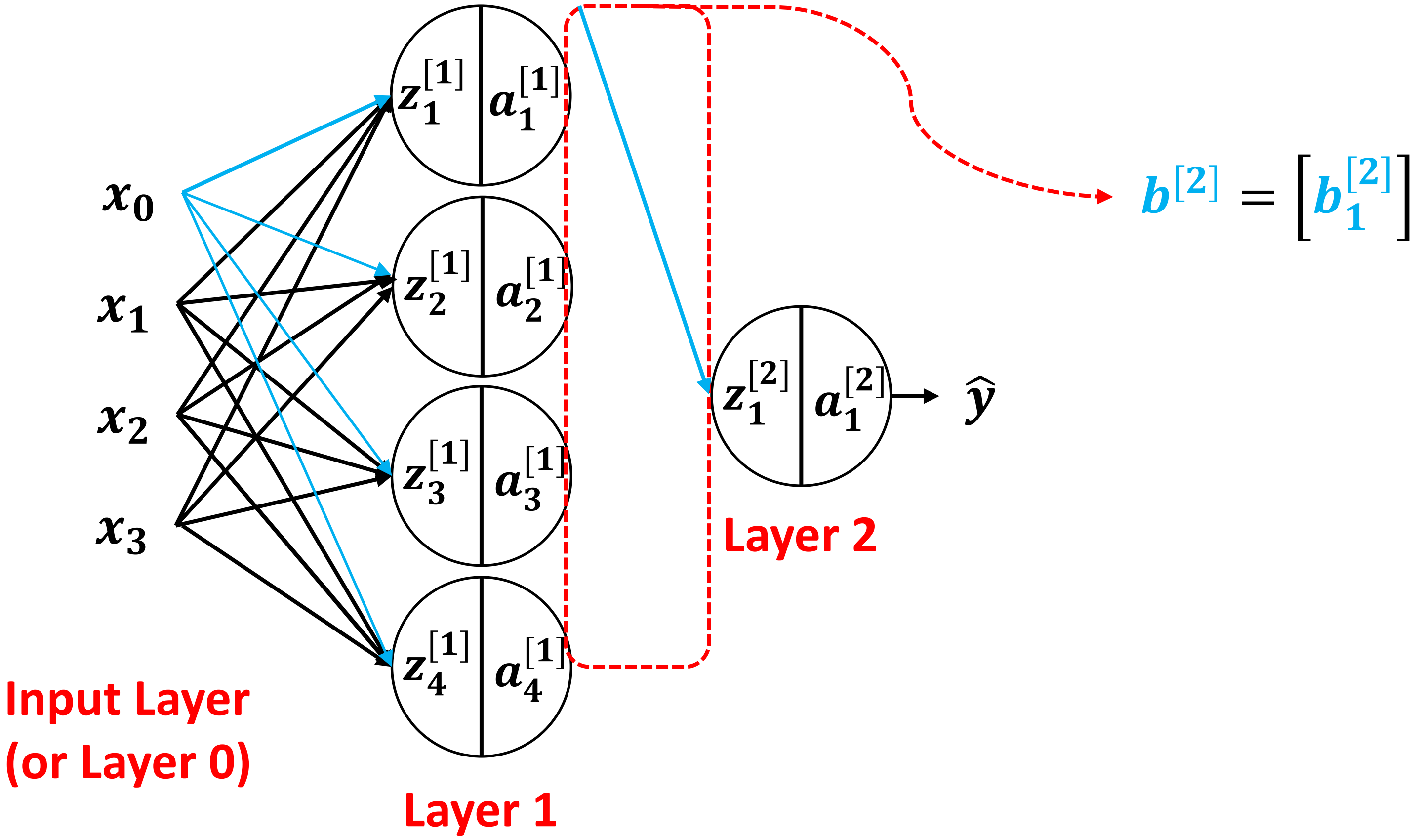


$$\boldsymbol{w}^{[2]^T} = \left[w_{11}^{[2]}\ w_{21}^{[2]}\ w_{31}^{[2]}\ w_{41}^{[2]}\right]$$

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

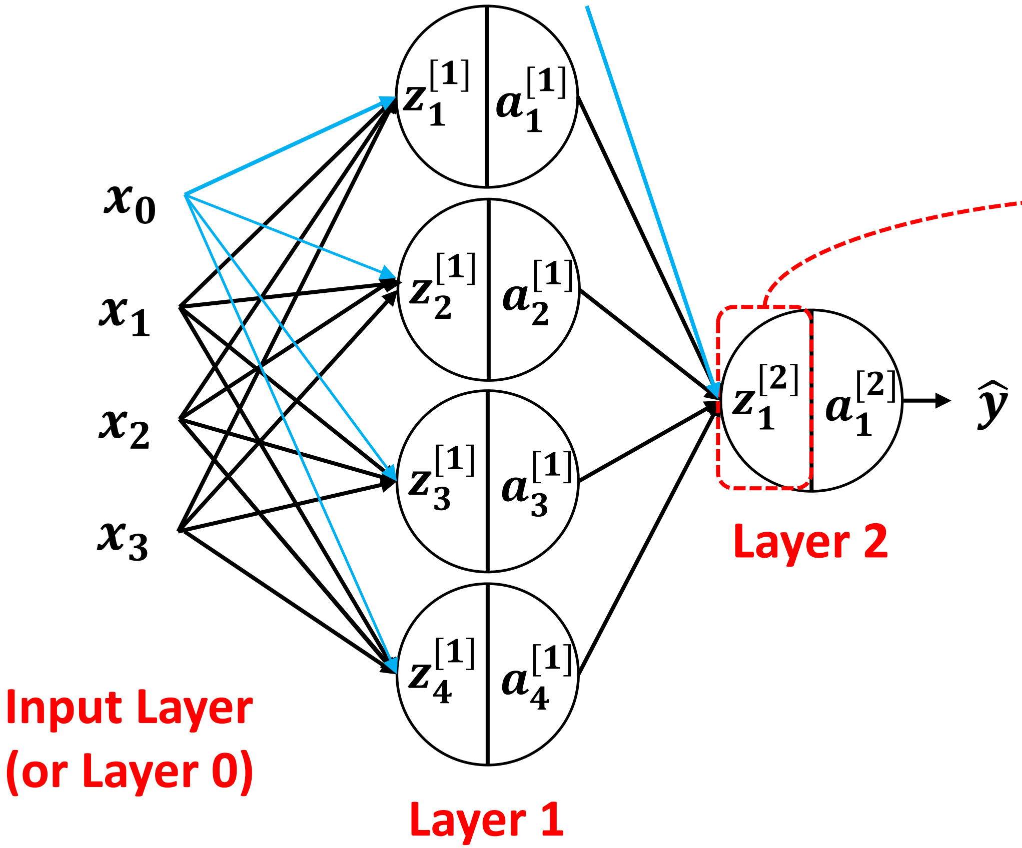
Vectorizing Input and All Variables

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



Vectorizing Input and All Variables

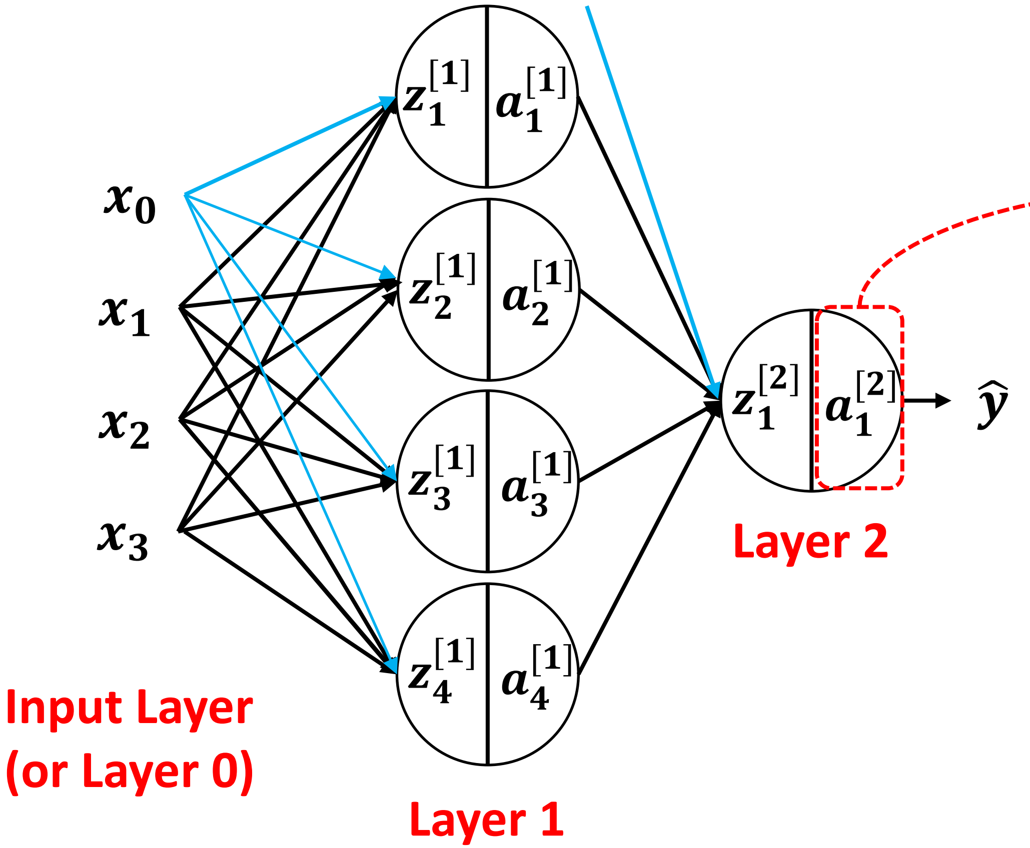
- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



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

Vectorizing Input and All Variables

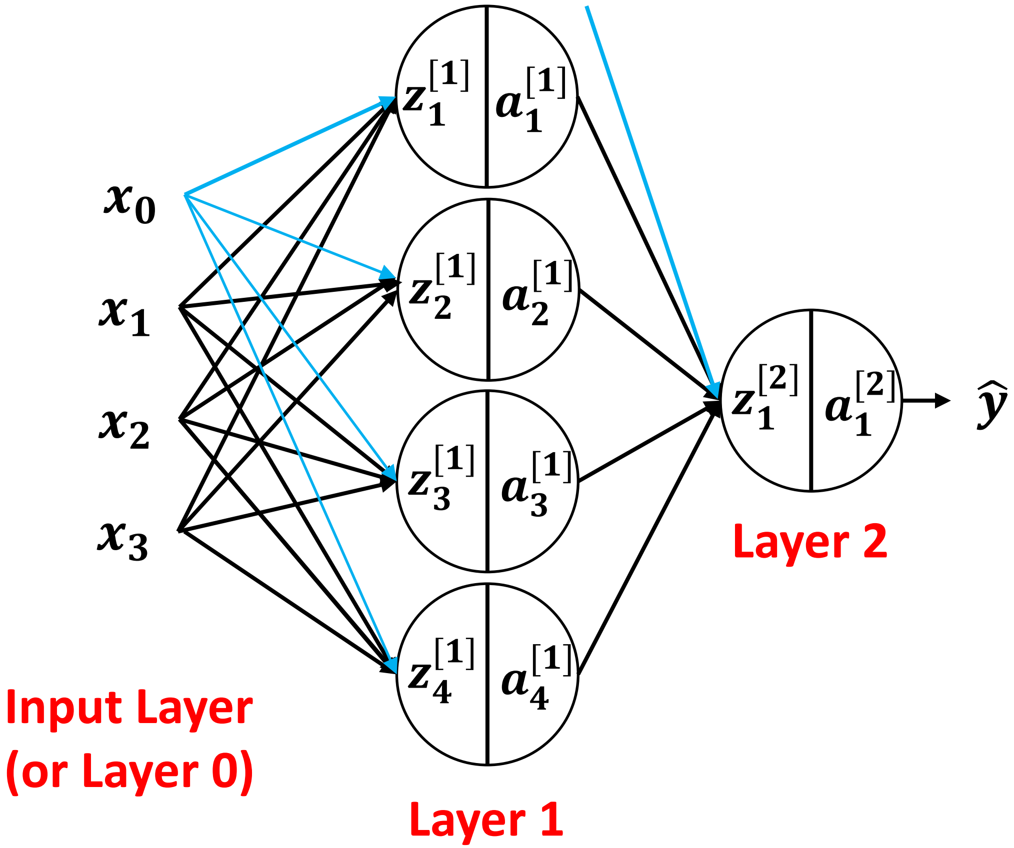
- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$a^{[2]} = \left[a_1^{[2]}\right]$$

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



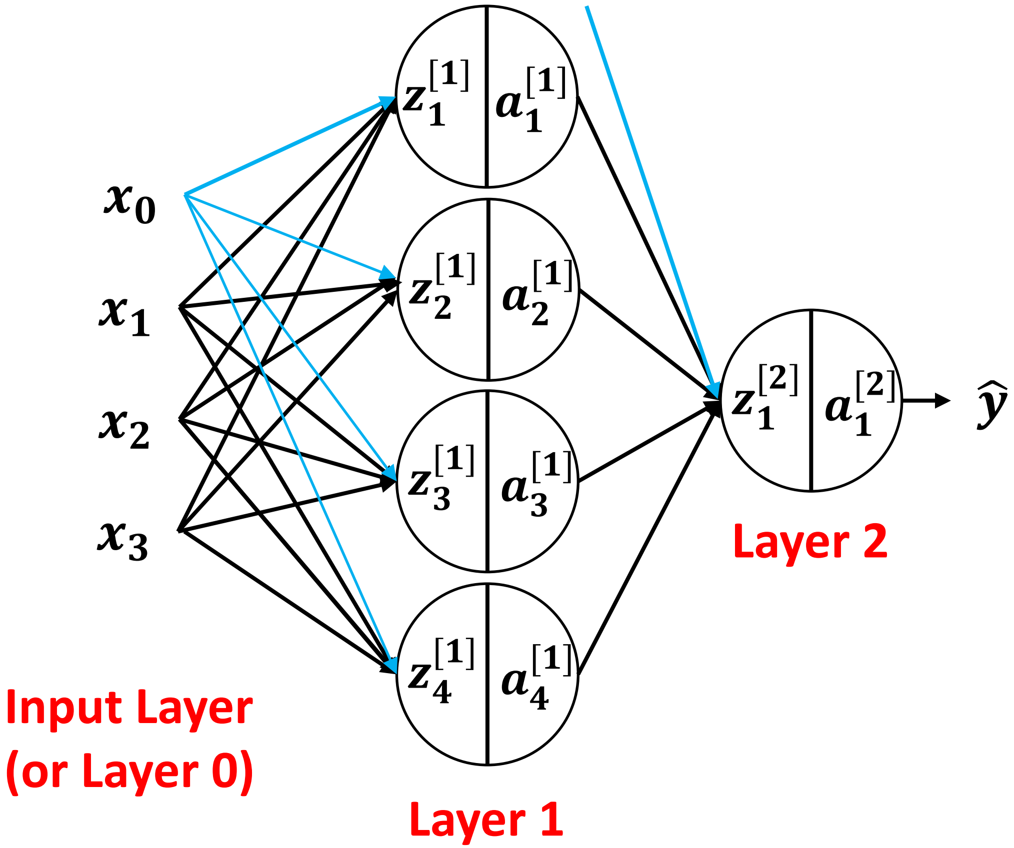
$$\boldsymbol{z}^{[2]} = \boldsymbol{w}^{[2]^T}\boldsymbol{x} + \boldsymbol{b}^{[2]}$$

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

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

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



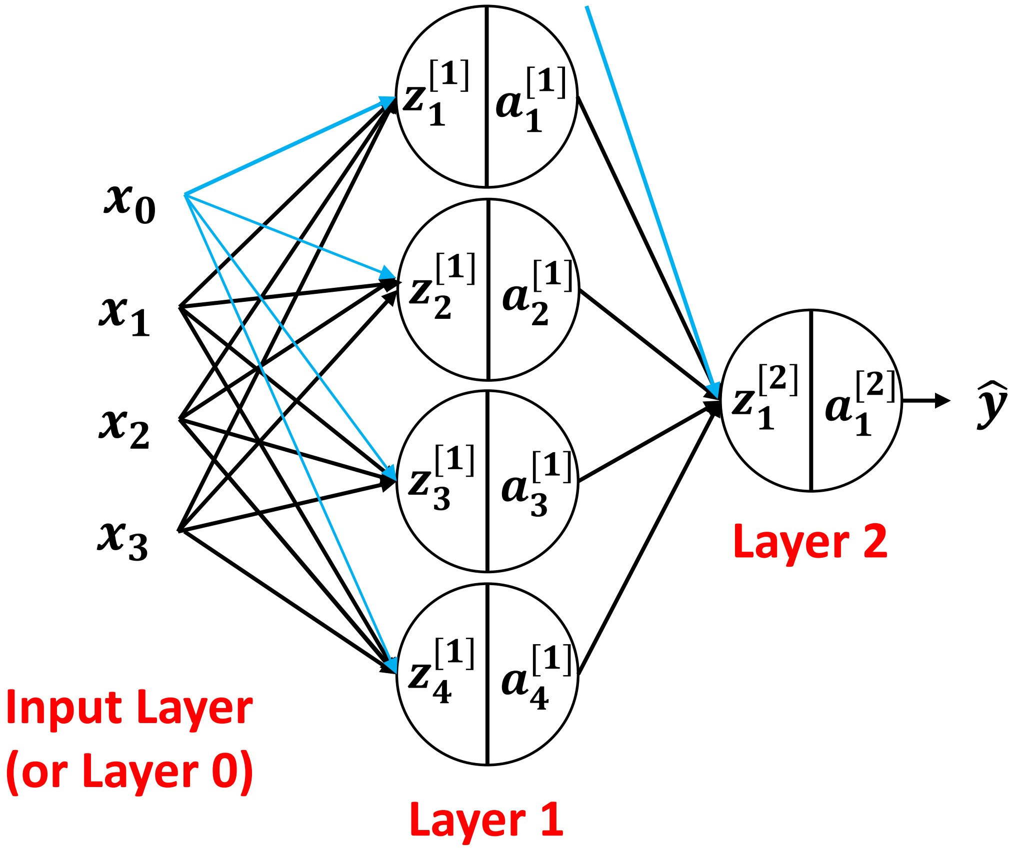
$$\boldsymbol{z}^{[2]} = \boldsymbol{w}^{[2]^T}\boldsymbol{x} + \boldsymbol{b}^{[2]}$$

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

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

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



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

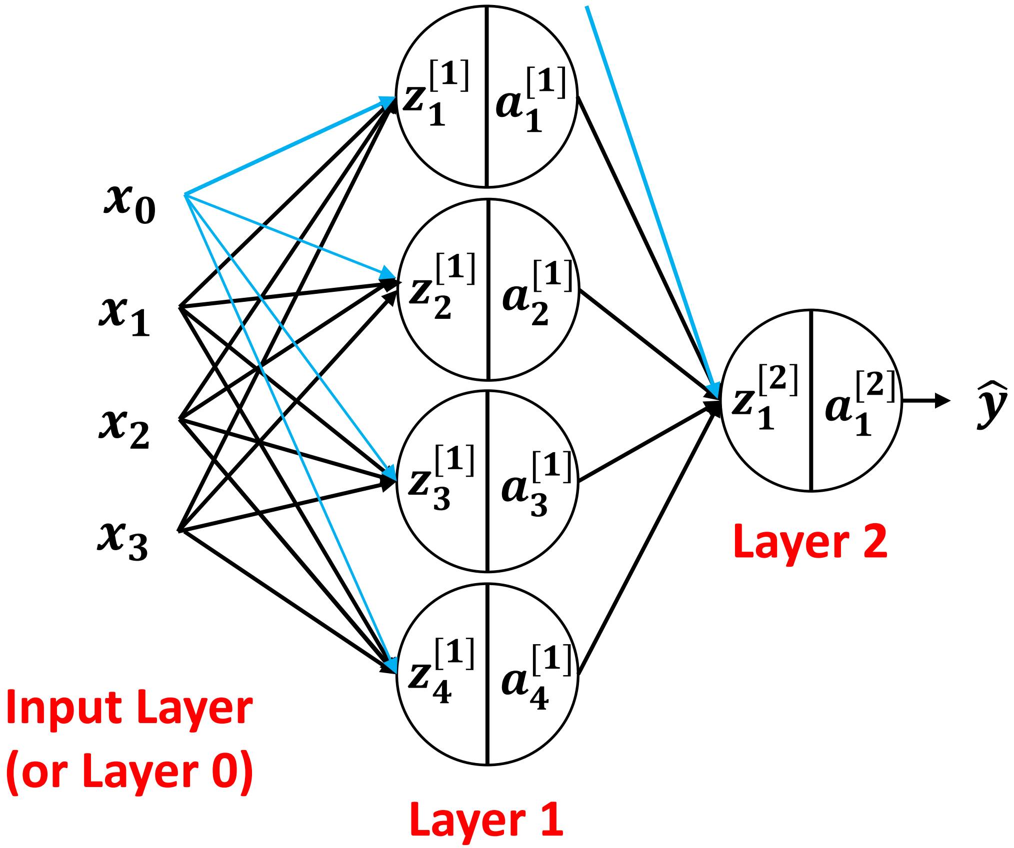
$$= \boldsymbol{w}^{[2]^T}\boldsymbol{a}^{[1]} + \boldsymbol{b}^{[2]}$$

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

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

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



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

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

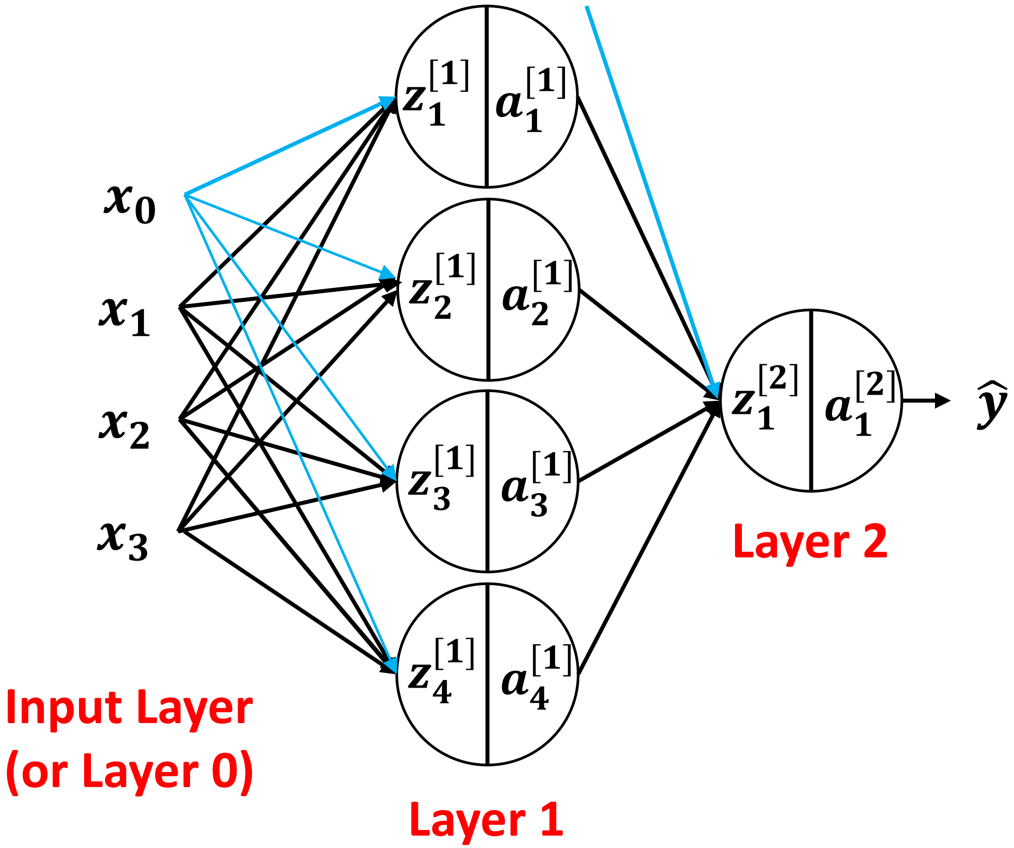
$$= \left[\boxed{w_{11}^{[2]}a_1^{[1]} + w_{21}^{[2]}a_2^{[1]} + w_{31}^{[2]}a_3^{[1]} + w_{41}^{[2]}a_4^{[1]} + b_1^{[2]}}\right]$$

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

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

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



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

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

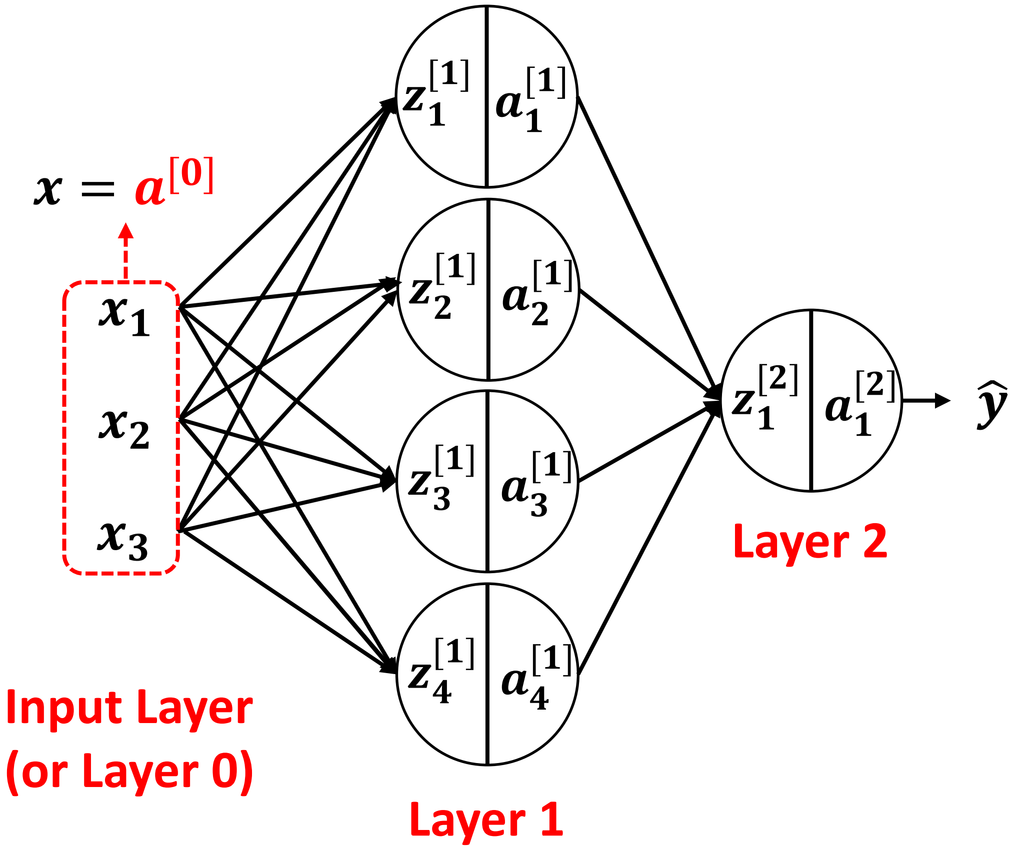
$$= \left[\boxed{w_{11}^{[2]}a_1^{[1]} + w_{21}^{[2]}a_2^{[1]} + w_{31}^{[2]}a_3^{[1]} + w_{41}^{[2]}a_4^{[1]} + b_1^{[2]}}\right]$$

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

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

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



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

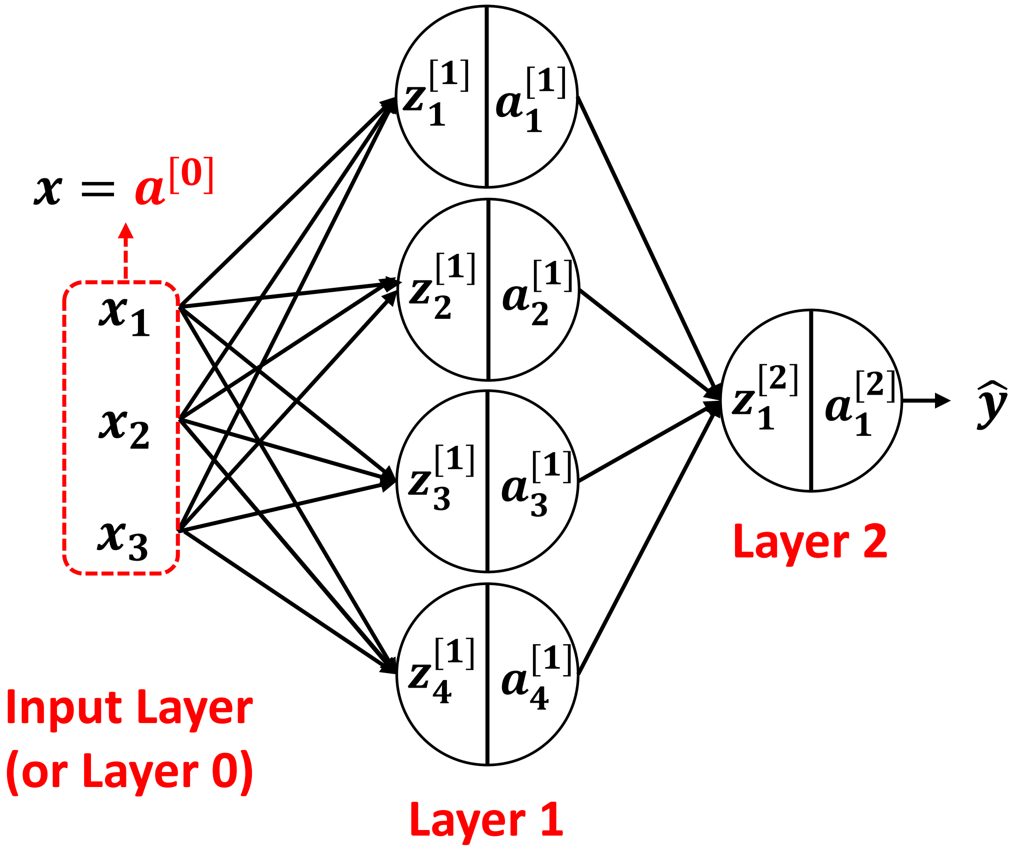
$$a^{[1]} = \sigma(z^{[1]})$$

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

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

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



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

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

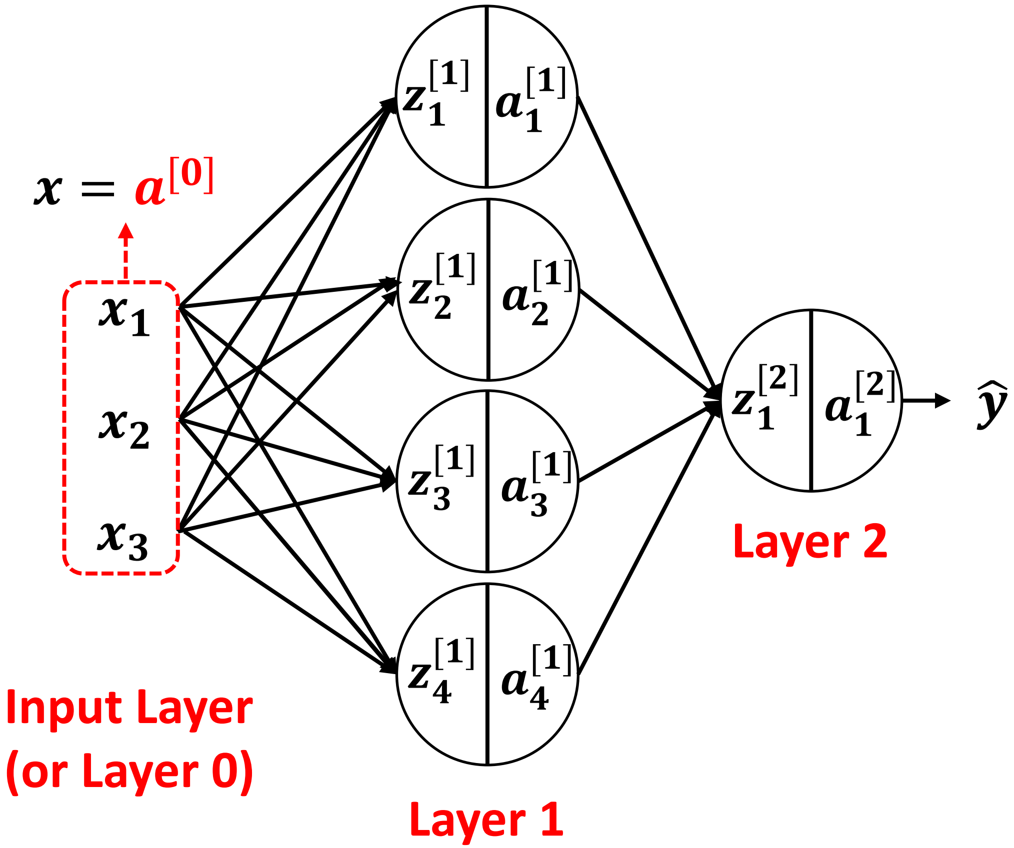
$$z^{[2]} = w^{[2]^T} a^{[1]} + b^{[2]}$$

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

But, this assumes only 1 training example!

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



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

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

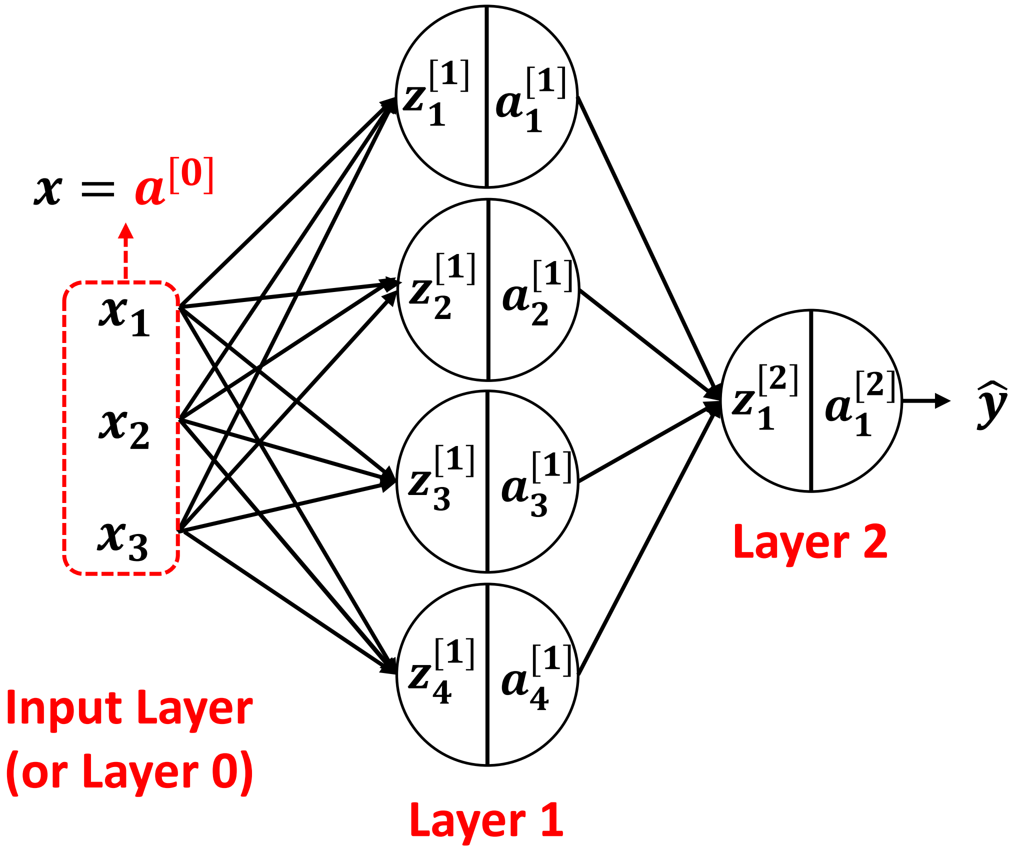
$$z^{[2]} = w^{[2]^T} a^{[1]} + b^{[2]}$$

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

How can we account for all the training examples?

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



$\boldsymbol{for\ i = 1\ to\ n}$: Assuming $\boldsymbol{n}$ examples

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

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

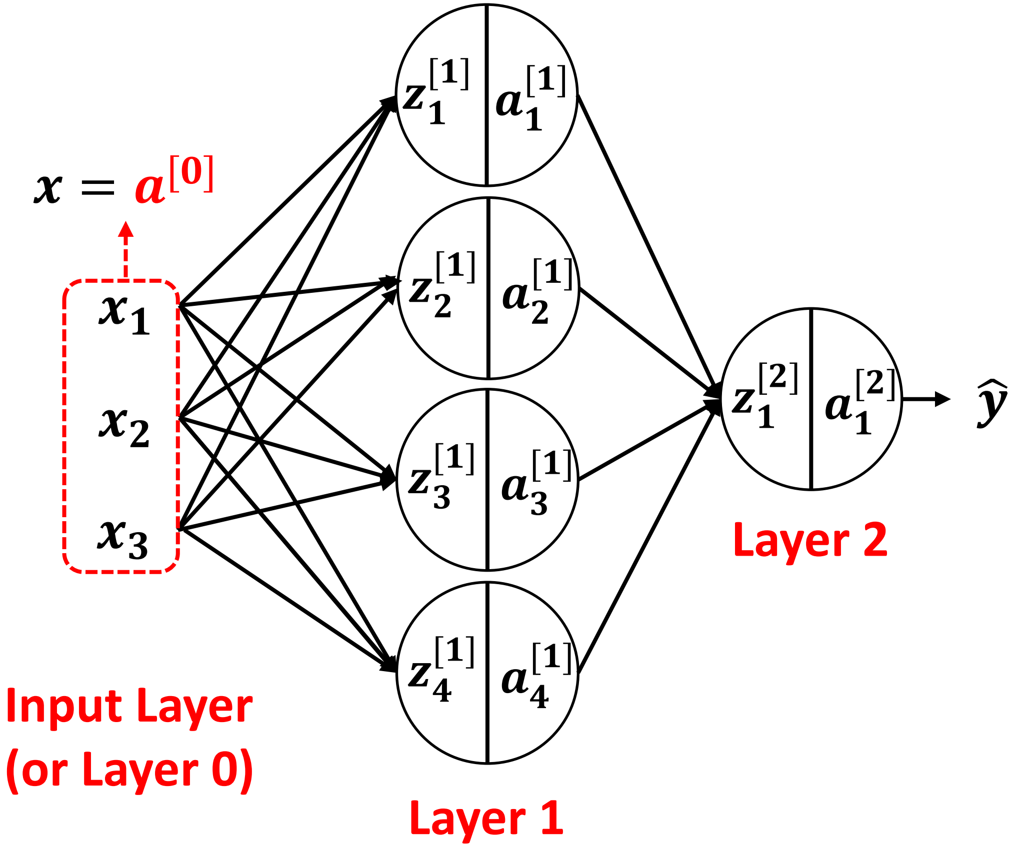
$$z^{[2](i)} = w^{[2]^{T(i)}} a^{[1](i)} + b^{[2]}$$

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

Refers to the i^{th} example in the training dataset

Vectorizing Input and All Variables

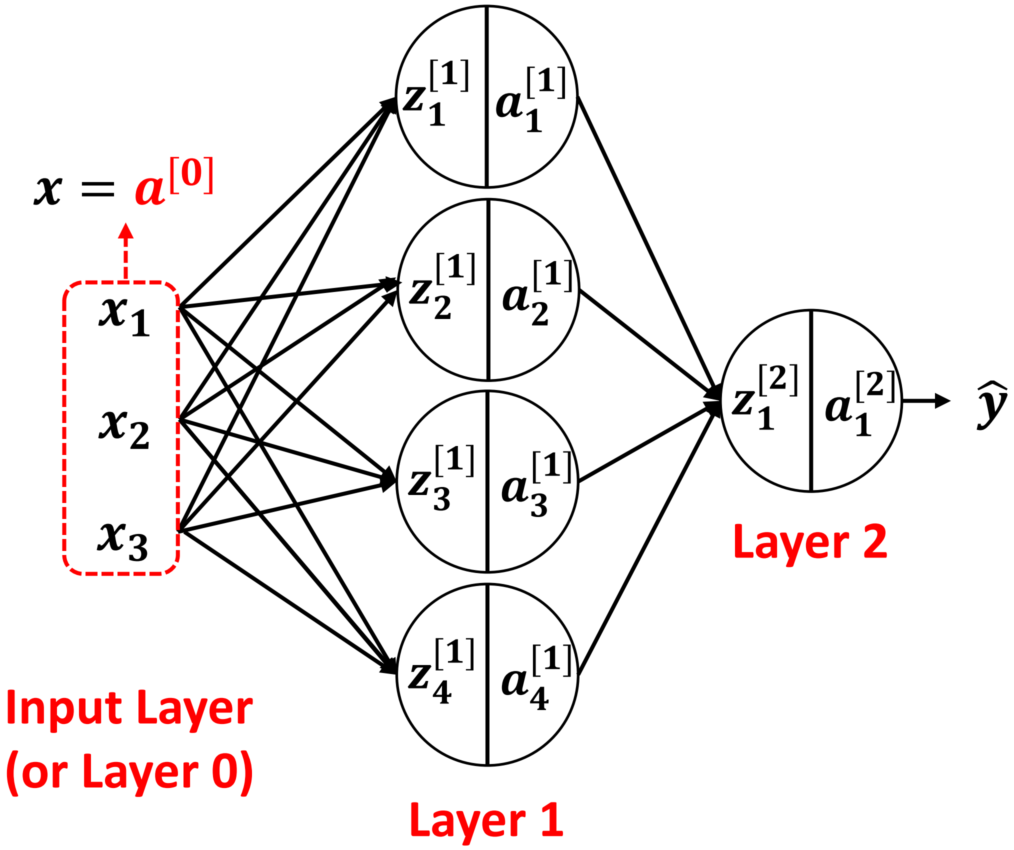
- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



But, loops in general slow down programs; hence, it is better to further ***vectorize*** the implementation in order to avoid any loop, whenever possible

Vectorizing Input and All Variables

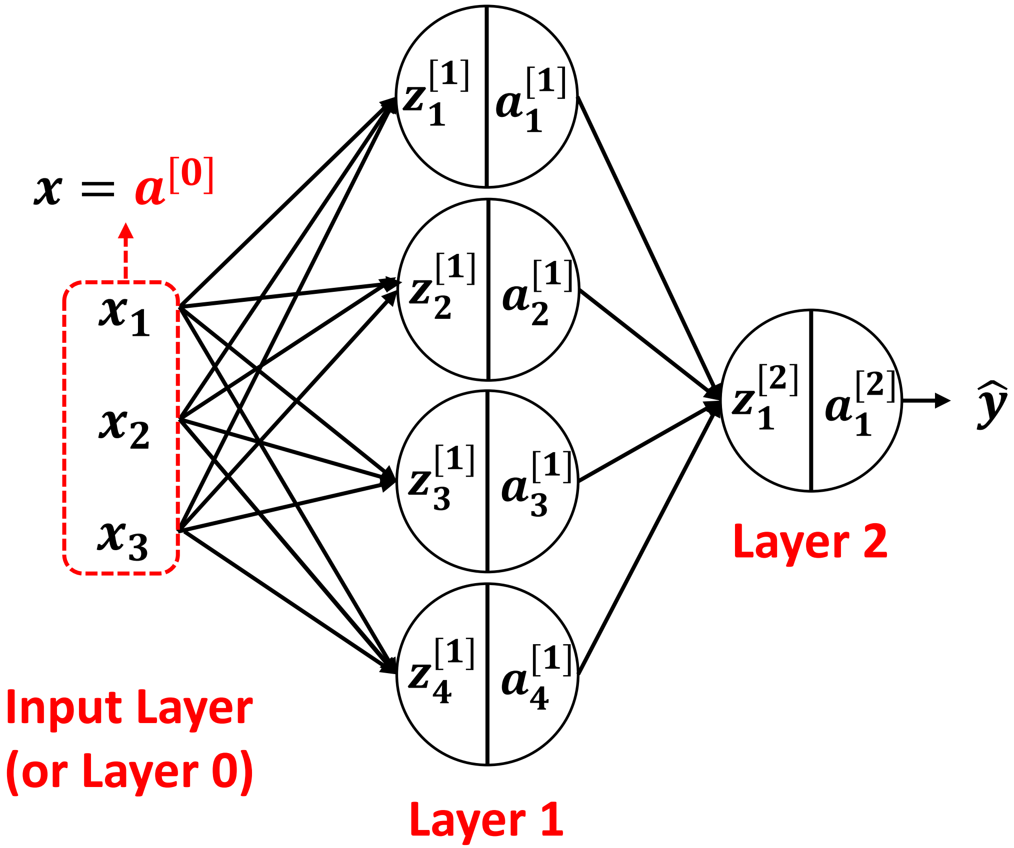
- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



To this end, we can simply stack all $\boldsymbol{x}$ vectors (or $\boldsymbol{a}^{[0]}$ vectors), $\boldsymbol{z}$ vectors, and $\boldsymbol{a}$ vectors in different matrices of every layer!

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



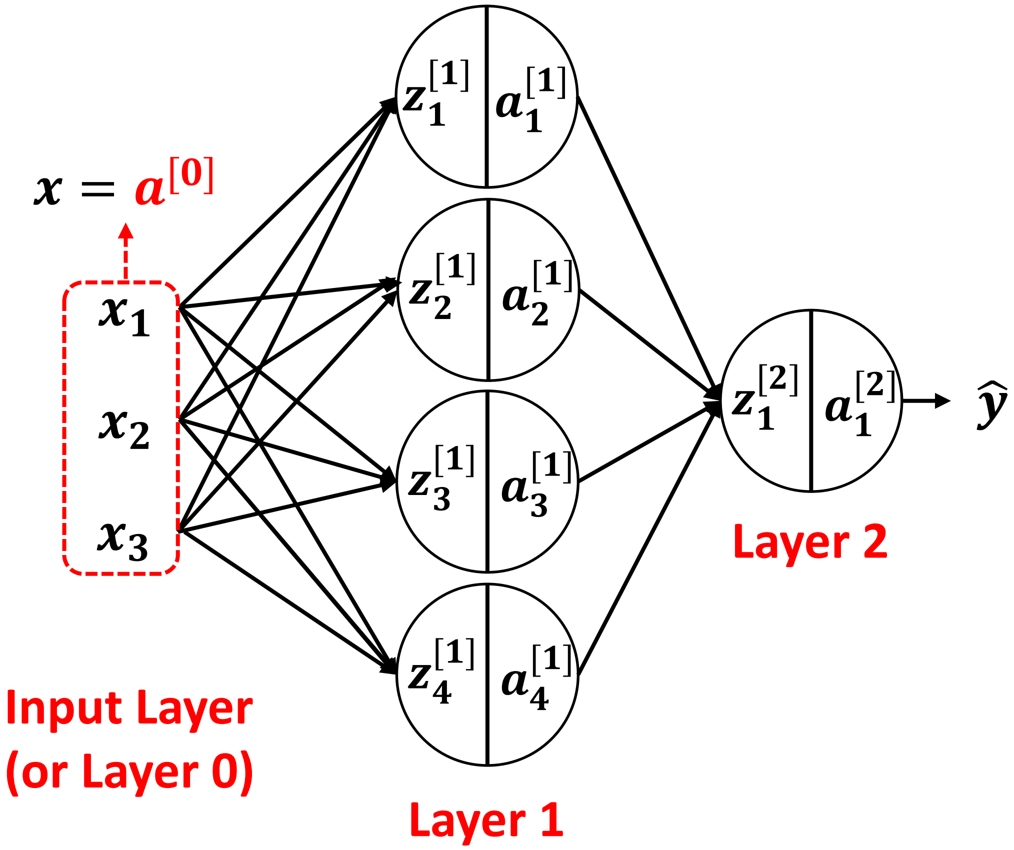
$$X = \begin{bmatrix} \boldsymbol{x}_1^{(1)} & \boldsymbol{x}_1^{(2)} & & \boldsymbol{x}_1^{(n)} \\ \boldsymbol{x}_2^{(1)} & \boldsymbol{x}_2^{(2)} & \dots & \boldsymbol{x}_2^{(n)} \\ \boldsymbol{x}_3^{(1)} & \boldsymbol{x}_3^{(2)} & & \boldsymbol{x}_3^{(n)} \end{bmatrix}$$

Assuming $\boldsymbol{n}$ examples

This vector represents the *first* example in the training dataset

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



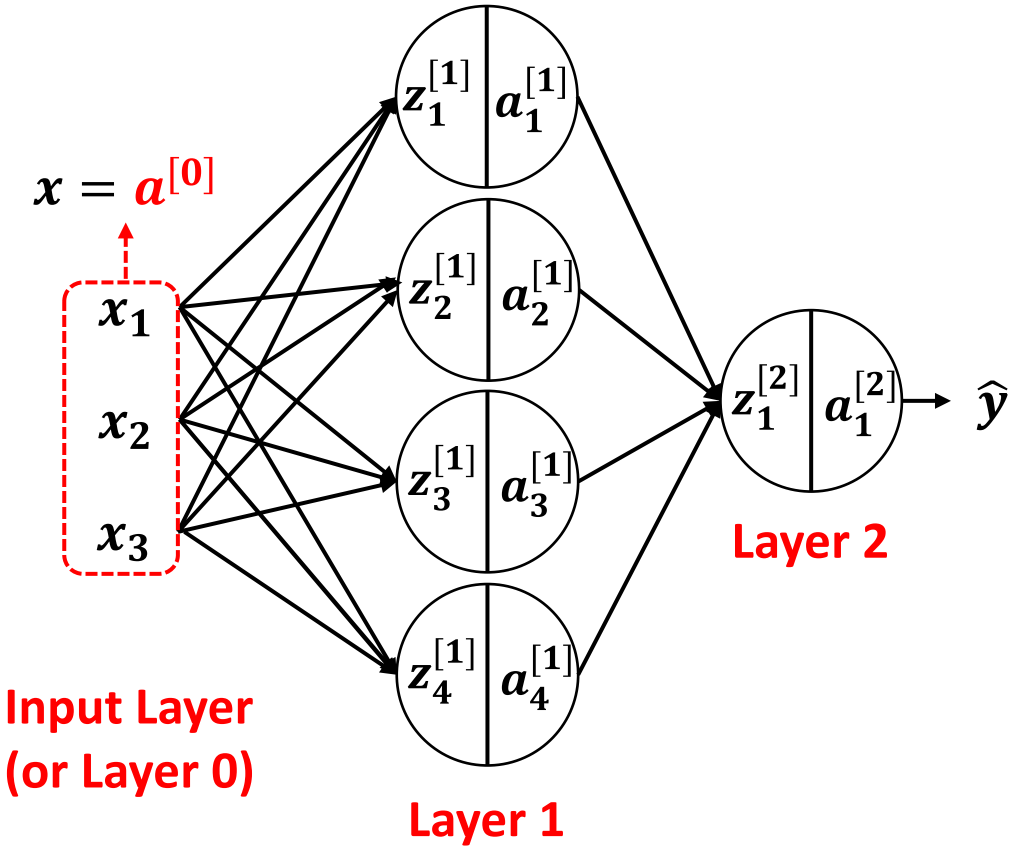
$$X = \begin{bmatrix} x_1^{(1)} & \boxed{\begin{matrix} x_1^{(2)} \\ x_2^{(2)} \\ x_3^{(2)} \end{matrix}} & & x_1^{(n)} \\ x_2^{(1)} & & \dots & x_2^{(n)} \\ x_3^{(1)} & & & x_3^{(n)} \end{bmatrix}$$

Assuming
n examples

This vector represents the *second* example in the training dataset

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



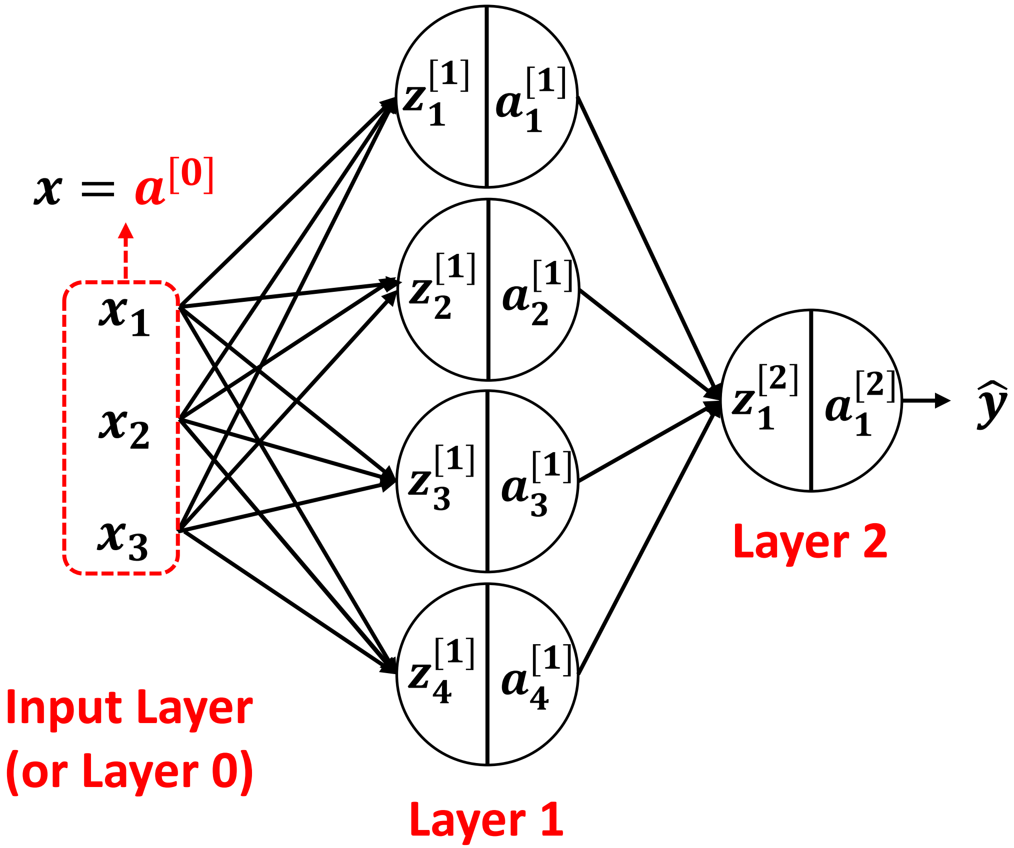
$$X = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & & \boxed{x_1^{(n)}} \\ x_2^{(1)} & x_2^{(2)} & \dots & \boxed{x_2^{(n)}} \\ x_3^{(1)} & x_3^{(2)} & & \boxed{x_3^{(n)}} \end{bmatrix}$$

Assuming
n examples

This vector represents the n^{th}
example in the training dataset

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

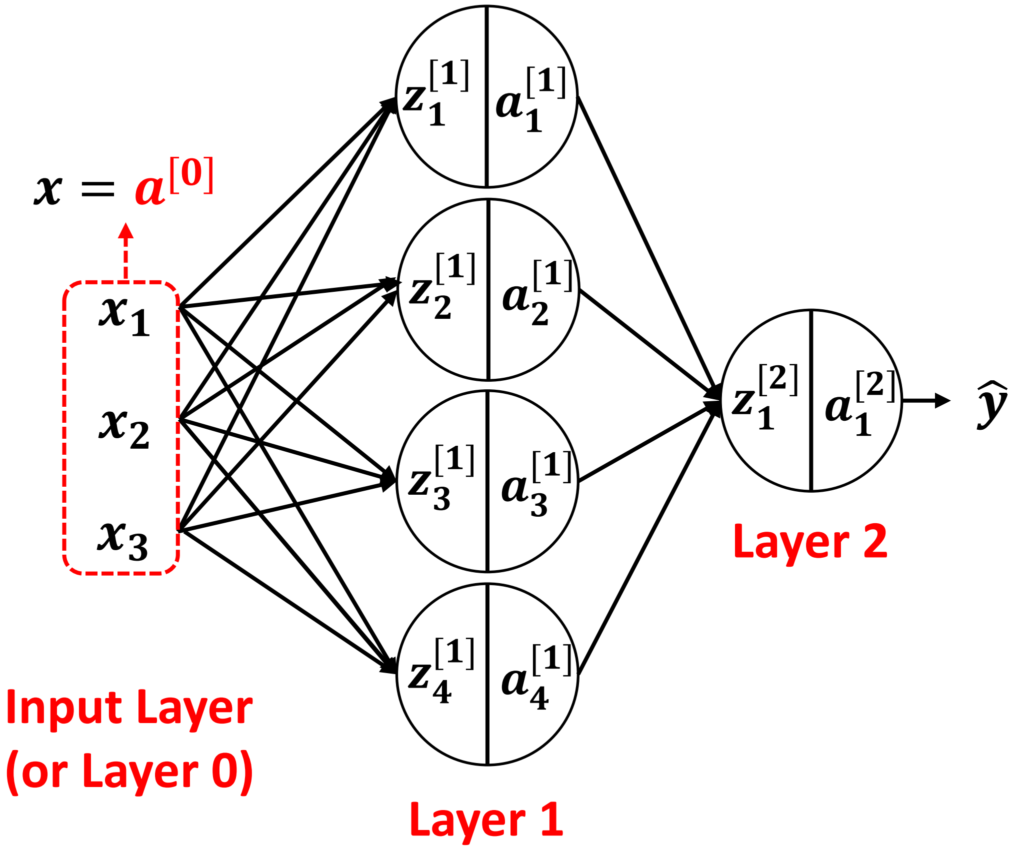


$$X = \underbrace{\begin{bmatrix} \boldsymbol{a}_1^{[0](1)} & \boldsymbol{a}_1^{[0](2)} & & \boldsymbol{a}_1^{[0](n)} \\ \boldsymbol{a}_2^{[0](1)} & \boldsymbol{a}_2^{[0](2)} & \cdots & \boldsymbol{a}_2^{[0](n)} \\ \boldsymbol{a}_3^{[0](1)} & \boldsymbol{a}_3^{[0](2)} & & \boldsymbol{a}_3^{[0](n)} \end{bmatrix}}$$

If we denote $\boldsymbol{x}$ as $\boldsymbol{a}^{[0]}$

Vectorizing Input and All Variables

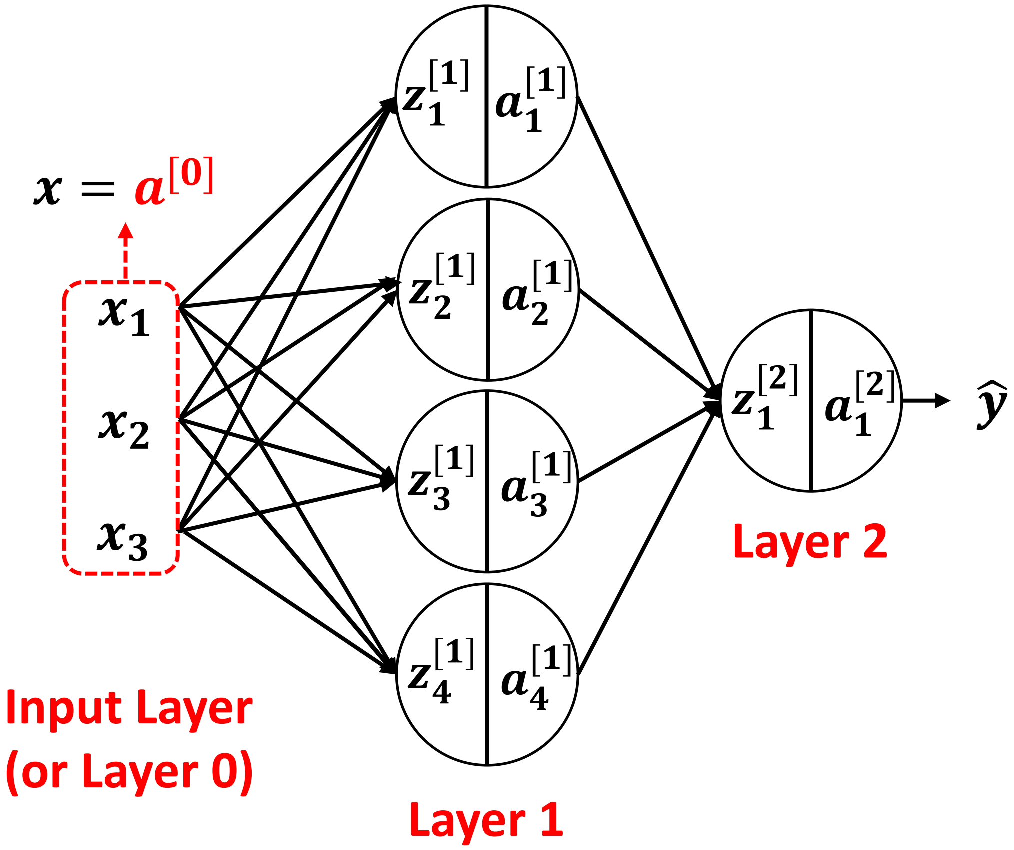
- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$X = A^{[0]} = \underbrace{\begin{bmatrix} a_1^{[0](1)} & a_1^{[0](2)} & & a_1^{[0](n)} \\ a_2^{[0](1)} & a_2^{[0](2)} & \cdots & a_2^{[0](n)} \\ a_3^{[0](1)} & a_3^{[0](2)} & & a_3^{[0](n)} \end{bmatrix}}_{\text{If we denote } x \text{ as } a^{[0]}}$$

Vectorizing Input and All Variables

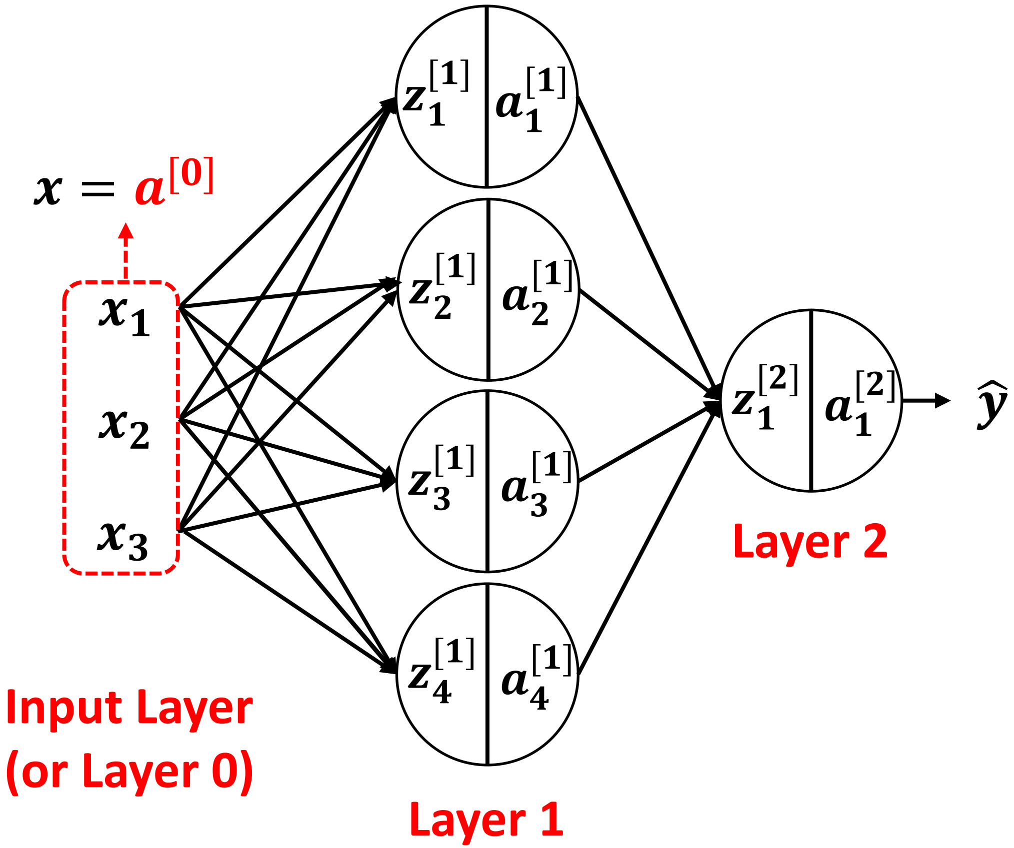
- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$Z^{[1]} = \begin{bmatrix} z_1^{[1](1)} & z_1^{[1](2)} & & z_1^{[1](n)} \\ z_2^{[1](1)} & z_2^{[1](2)} & & z_2^{[1](n)} \\ z_3^{[1](1)} & z_3^{[1](2)} & \cdots & z_3^{[1](n)} \\ z_4^{[1](1)} & z_4^{[1](2)} & & z_4^{[1](n)} \end{bmatrix}$$

Vectorizing Input and All Variables

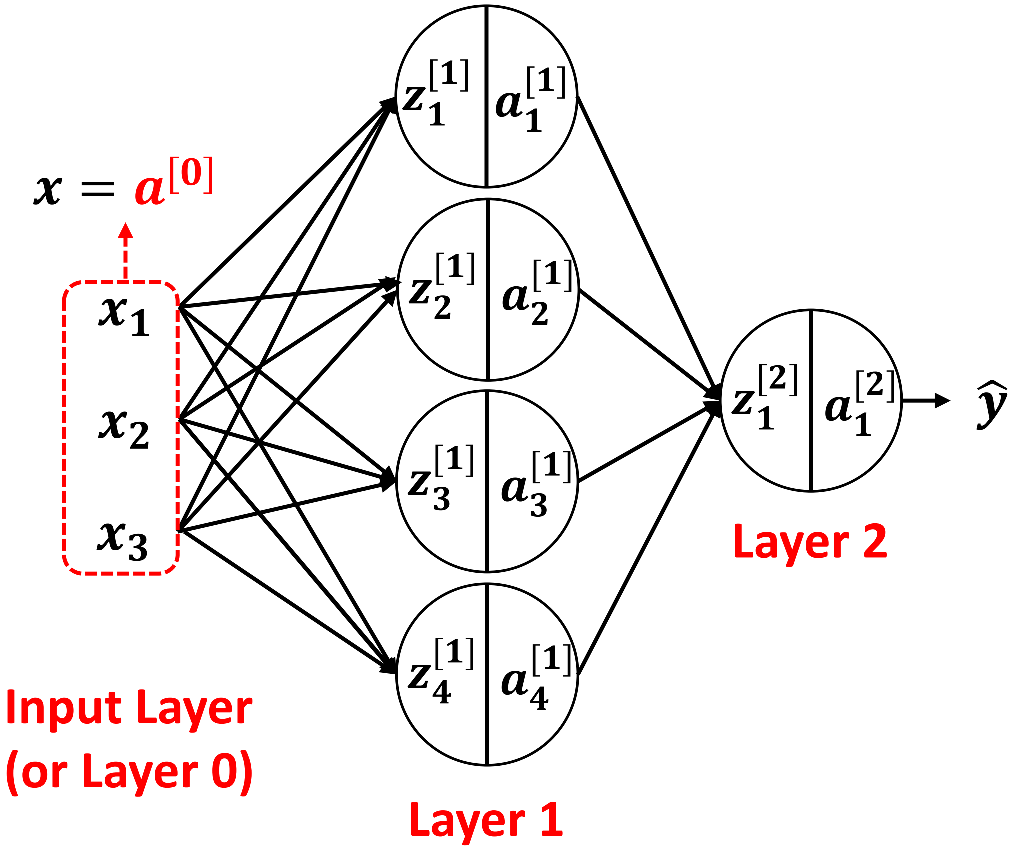
- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$A^{[1]} = \begin{bmatrix} a_1^{[1](1)} & a_1^{[1](2)} & & a_1^{[1](n)} \\ a_2^{[1](1)} & a_2^{[1](2)} & & a_2^{[1](n)} \\ a_3^{[1](1)} & a_3^{[1](2)} & \cdots & a_3^{[1](n)} \\ a_4^{[1](1)} & a_4^{[1](2)} & & a_4^{[1](n)} \end{bmatrix}$$

Vectorizing Input and All Variables

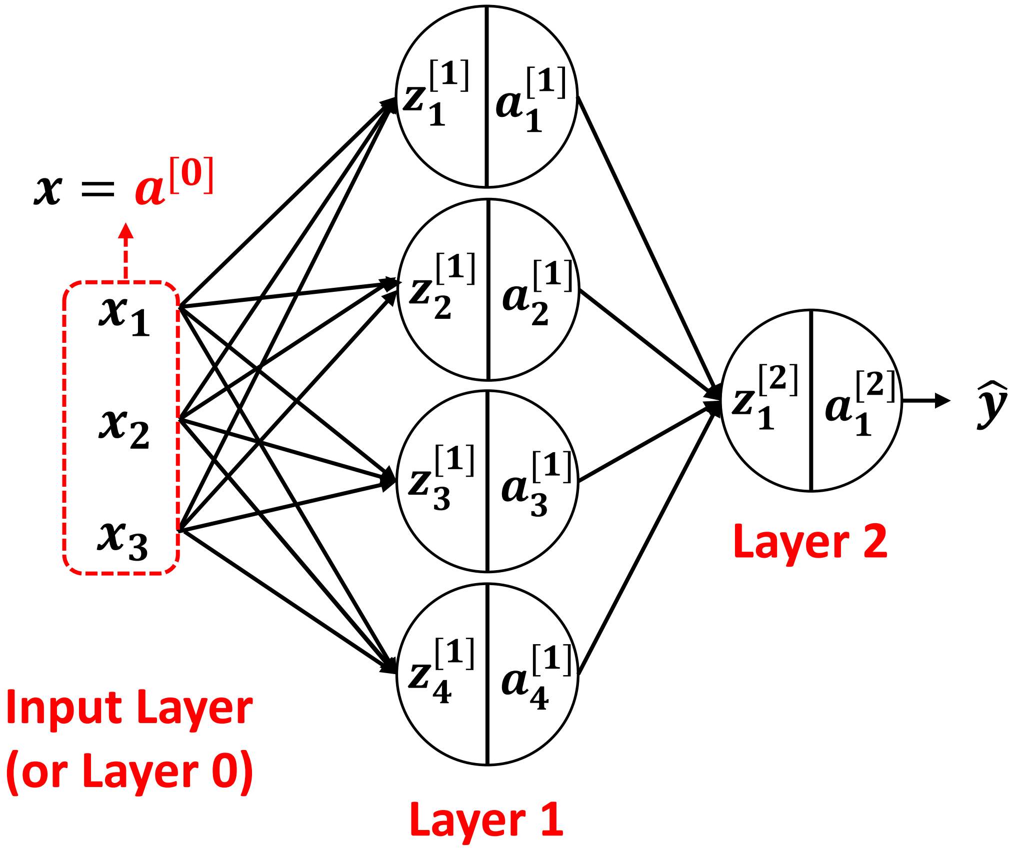
- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$Z^{[2]} = \left[z_1^{[2](1)} \quad z_1^{[2](2)} \dots \quad z_1^{[2](n)} \right]$$

Vectorizing Input and All Variables

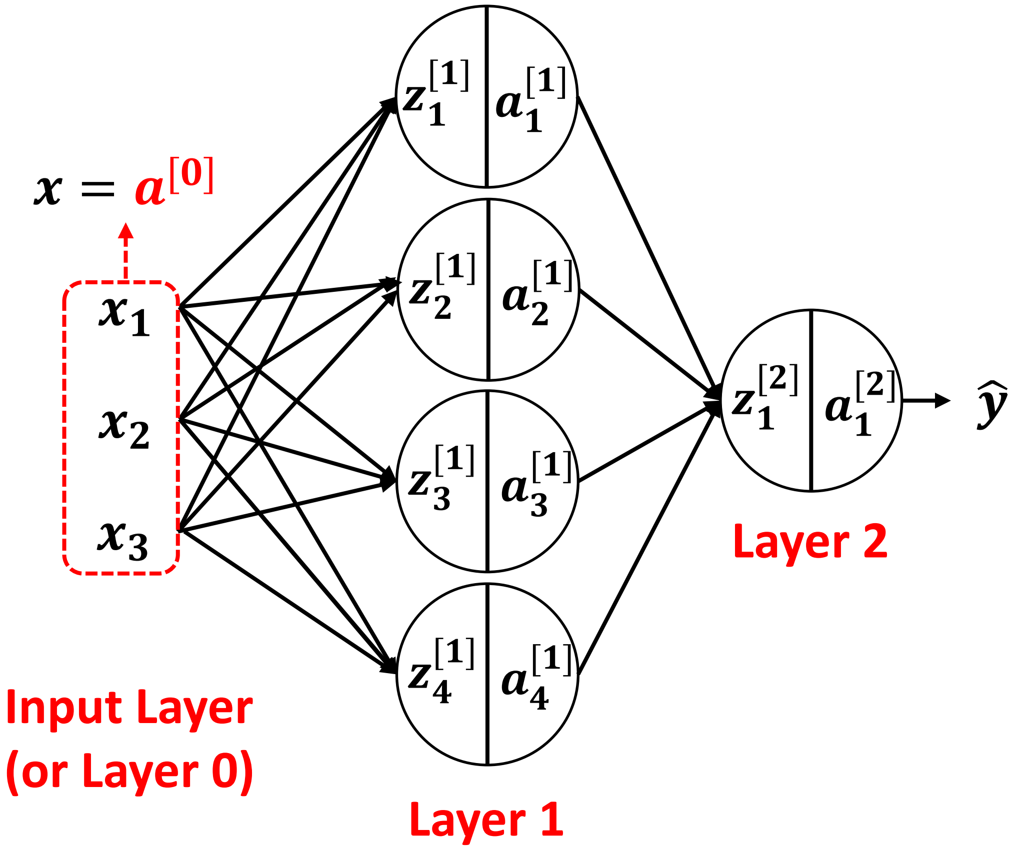
- To help develop an efficient learning algorithm, let us ***vectorize*** (represent in vectors & matrices) the input and the variables involved



$$A^{[2]} = \left[a_1^{[2](1)} \quad a_1^{[2](2)} \dots \quad a_1^{[2](n)}\right]$$

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

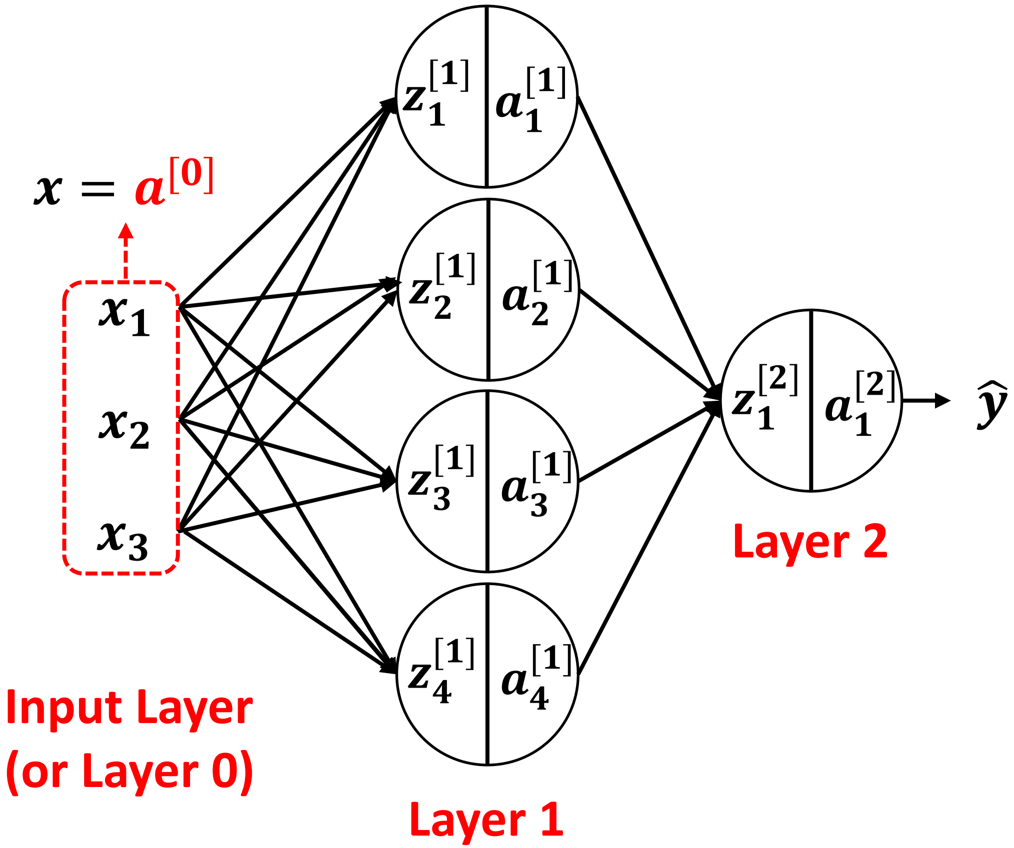


$$\boldsymbol{for\ i = 1\ to\ n:}$$

$$\underbrace{\begin{aligned} z^{[1](i)} &= w^{[1]^T(i)} a^{[0](i)} + b^{[1]} \\ a^{[1](i)} &= \sigma(z^{[1](i)}) \\ z^{[2](i)} &= w^{[2]^{T(i)}} a^{[1](i)} + b^{[2]} \\ a^{[2](i)} &= \sigma(z^{[2](i)}) \end{aligned}}_{\textit{Before}\ \text{Vectorization}}$$

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



No Explicit Loop!

$$Z^{[1]} = w^{[1]^T} A^{[0]} + b^{[1]}$$

$$A^{[1]} = \sigma(Z^{[1]})$$

$$Z^{[2]} = w^{[2]^T} A^{[1]} + b^{[2]}$$

$$A^{[2]} = \sigma(Z^{[2]})$$

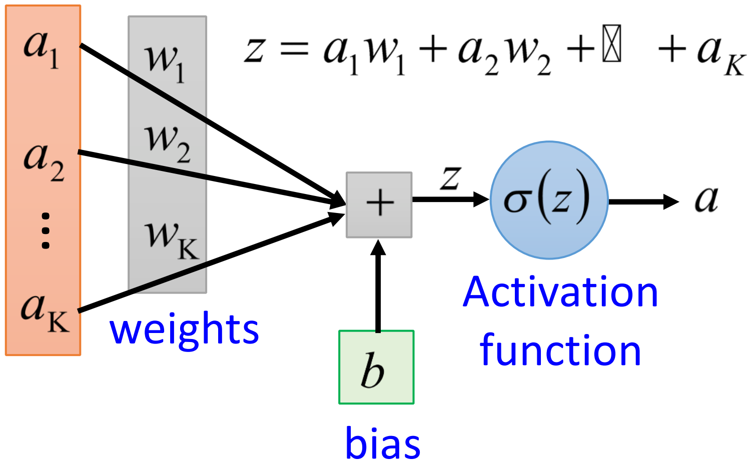
***After* Vectorization**

Epitome: A Neuron

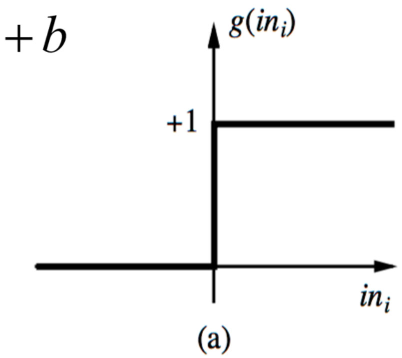
A neuron is a computational unit in the neural network that exchanges messages with each other.

<u>Neuron</u> $f: R^K \to R$

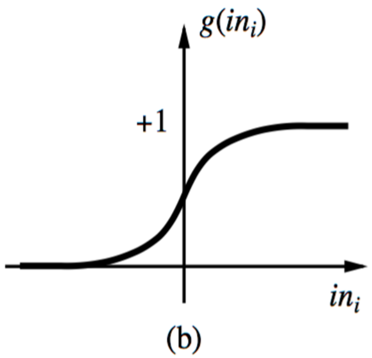
$$z = a_1 w_1 + a_2 w_2 + \cdots + a_K w_K + b$$



Possible activation functions:



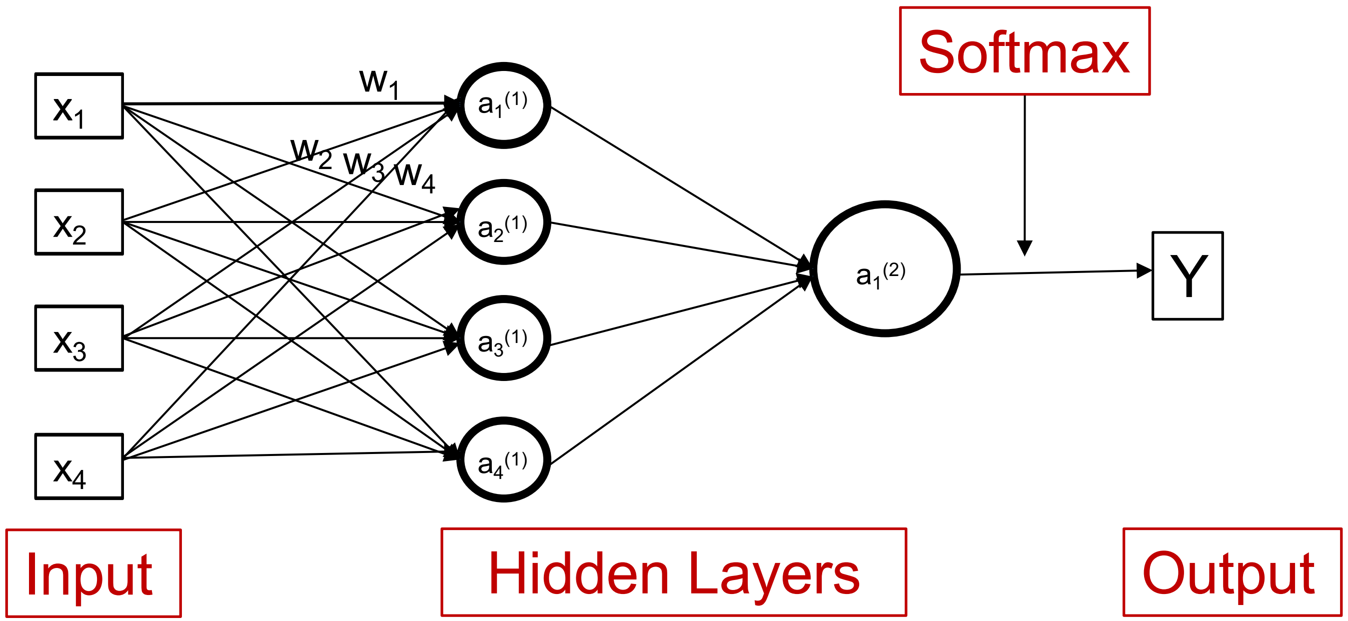
Step function/threshold function



Sigmoid function (a.k.a, logistic function)

Epitome: A simple neural network

Ignoring bias…



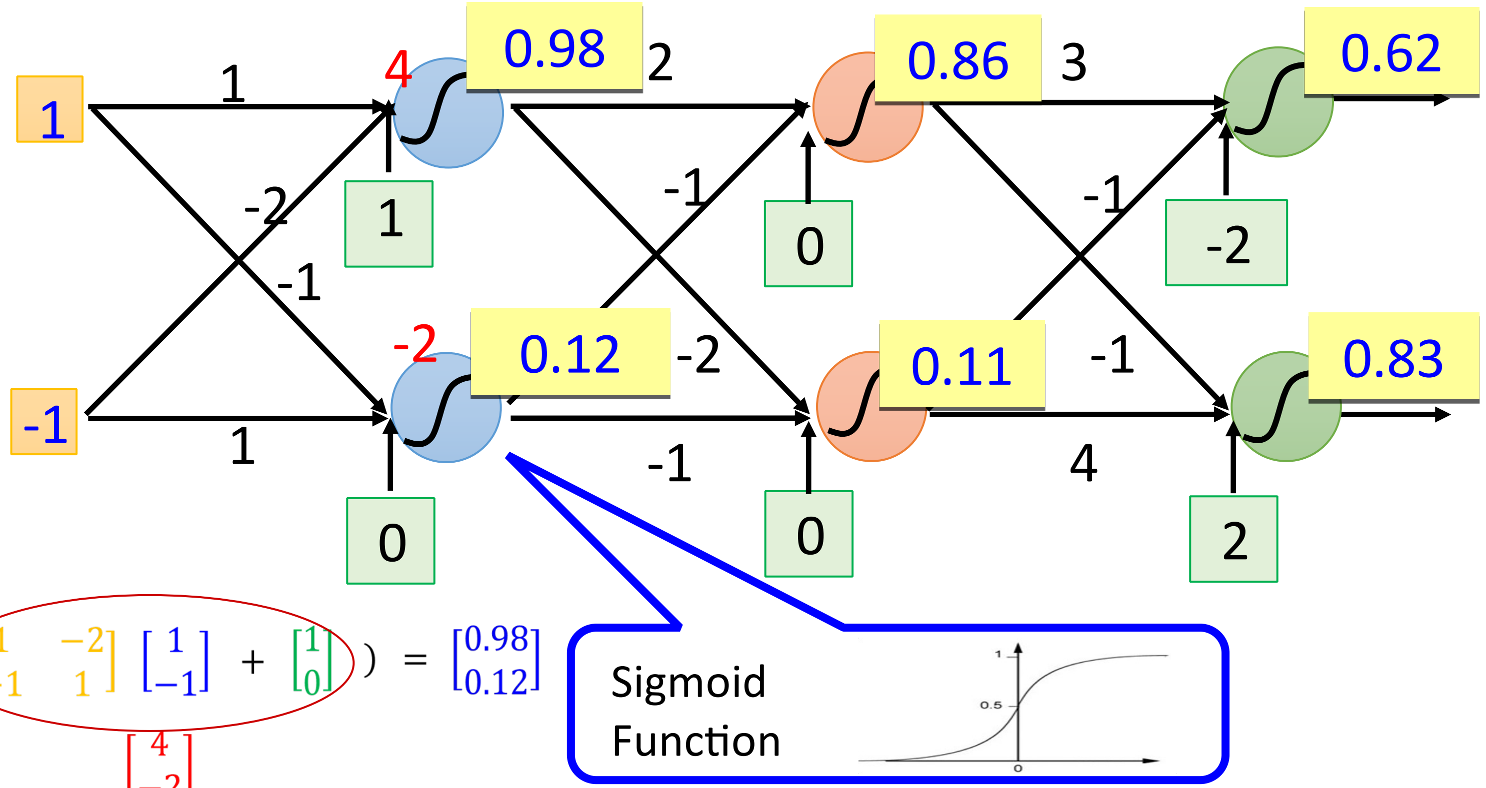
$$a1^{(1)} = f(w1 * x1 + w2 * x2 + w3 * x3 + w4 * x4)$$

Number of parameters:
4*4 + 4 +1

$f(\,)$ is activation function: Relu or sigmoid

$$Relu\!: \max(0, x)$$
$$a1^{(1)} = max(0, w1 * x1 + w2 * x2 + w3 * x3 + w4 * x4)$$

Epitome: Example of Neural Network

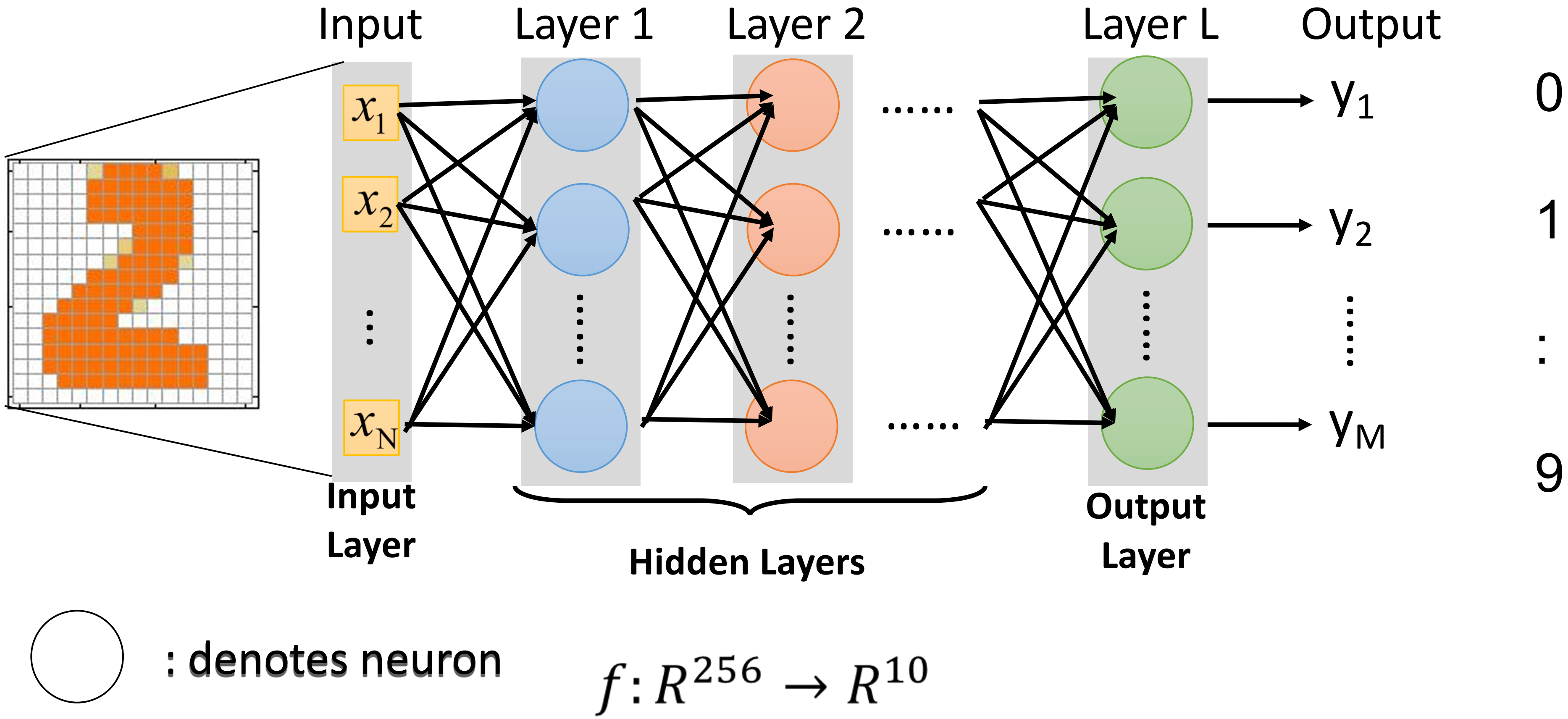


$$\sigma\left(\begin{bmatrix} 1 & -2 \\ -1 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ -1 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix}\right) = \begin{bmatrix} 0.98 \\ 0.12 \end{bmatrix}$$

$$\begin{bmatrix} 4 \\ -2 \end{bmatrix}$$

$$f: R^2 \rightarrow R^2 \qquad f\left(\begin{bmatrix} 1 \\ -1 \end{bmatrix}\right) = \begin{bmatrix} 0.62 \\ 0.83 \end{bmatrix} \quad f\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}\right) = \begin{bmatrix} 0.51 \\ 0.85 \end{bmatrix}$$

Epitome: A case study

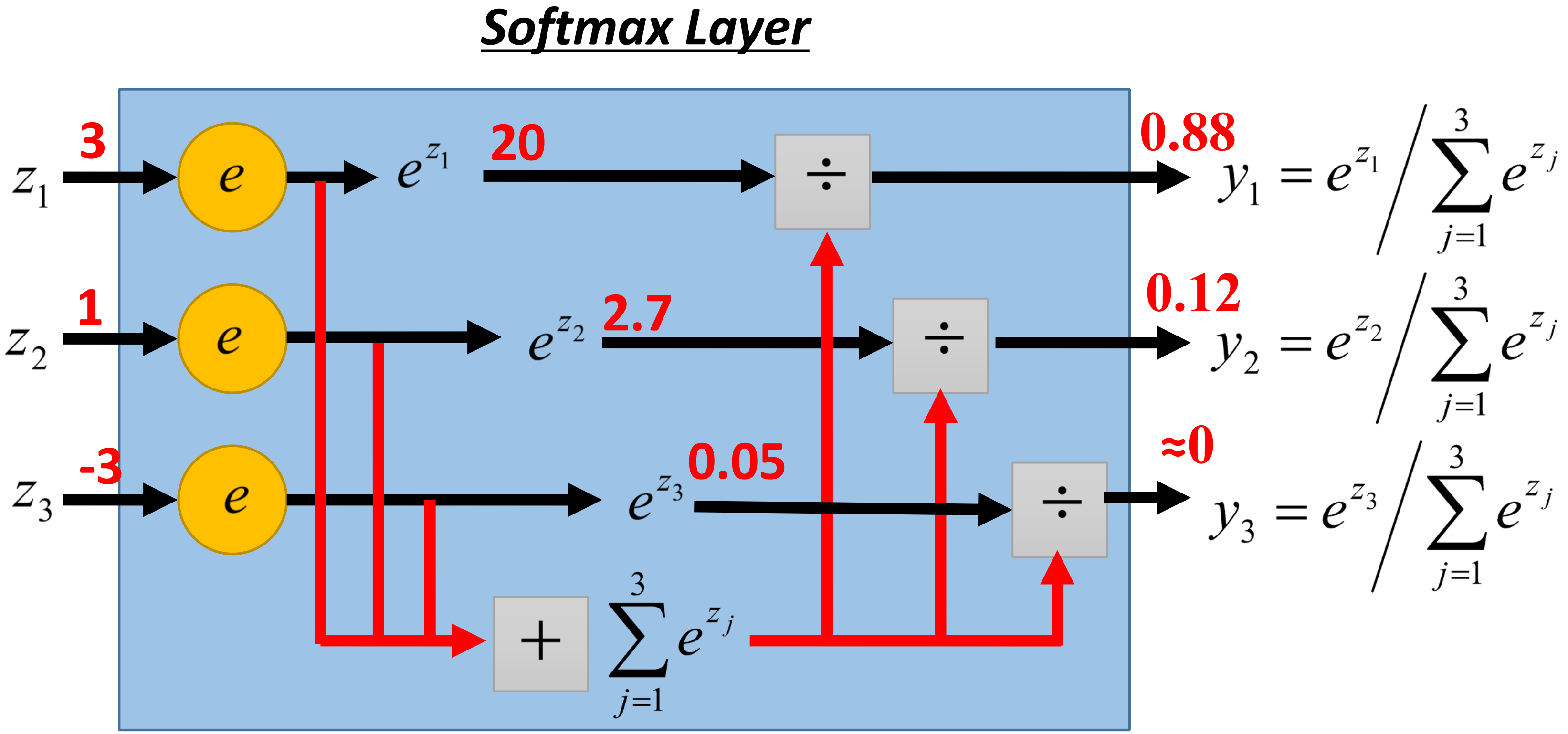


$$f: R^{256} \to R^{10}$$

Deep learning (DL) means many hidden layers.
DL represents the function f by neural network

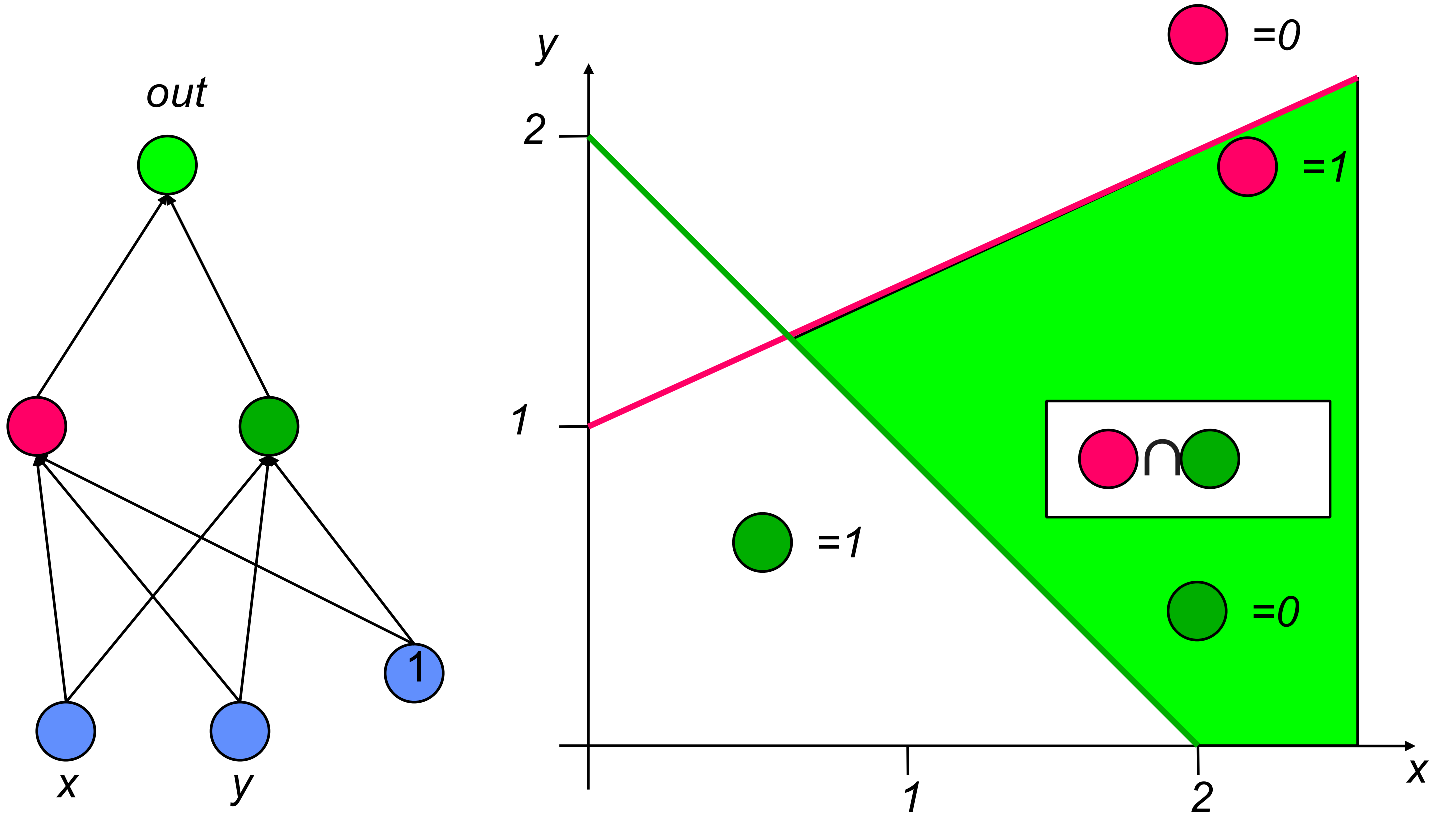
Softmax

- Softmax layer as the output layer, to provide probabilistic interpretation.
- Output of network can be any value, which is difficult to interpret



WHY Neural Network works?

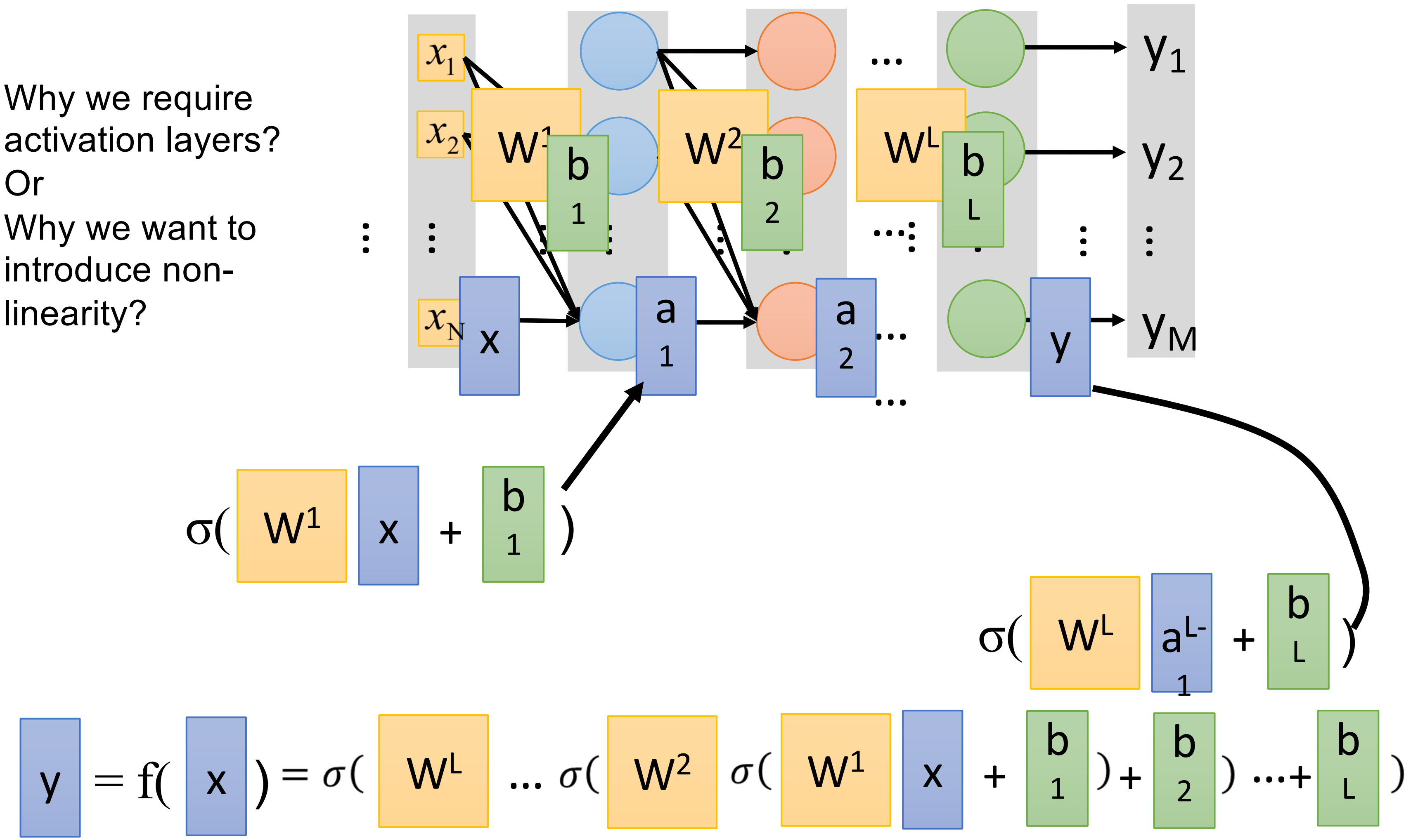
Visualizing as Constraint Satisfaction Networks without activation function



Each neuron in the hidden layer computes a nonlinear transform of inputs it receives.
Hidden layer neurons act as “features” for final layer (a linear model) to produce output.
The overall effect is a nonlinear mapping from inputs to outputs.

WHY Neural Network works?

Why we require activation layers?
Or
Why we want to introduce non-linearity?



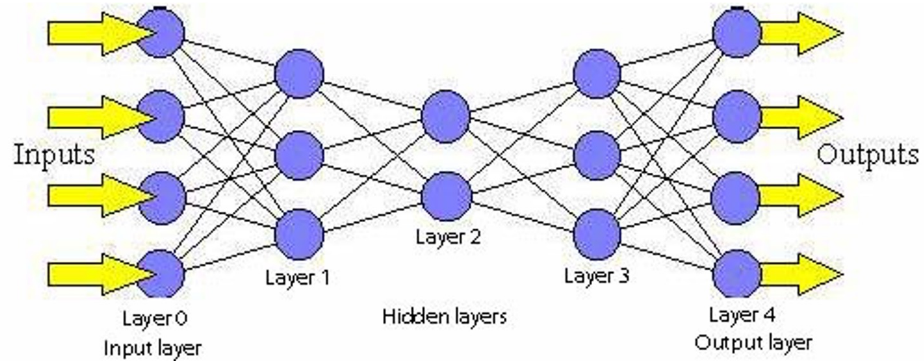
It is related to kernel learning where kernels are learnt implicitly.
What happen if we use linear mapping as an activation function?
MLP or NN with a single, sufficiently wide hidden layer can approximate any function.

Preliminaries

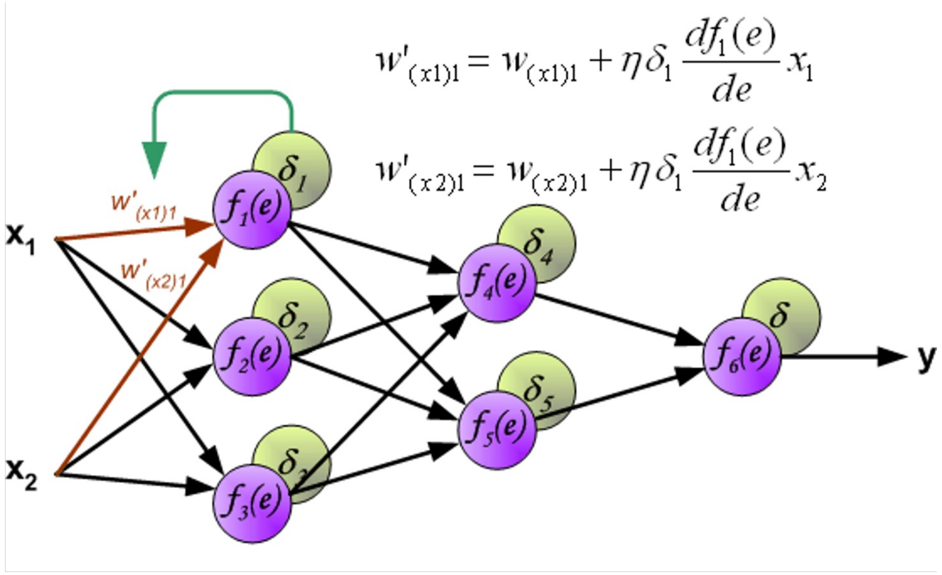
A visual proof that neural nets can approximate any function
http://neuralnetworksanddeeplearning.com/chap4.html

Why non-linearity is required?
1) Example of fitting non-linear data without non-linearities
2) Example of fiiting non-linear data with non-linearities

Feed forward/Backpropagation Neural Network



Feed forward algorithm

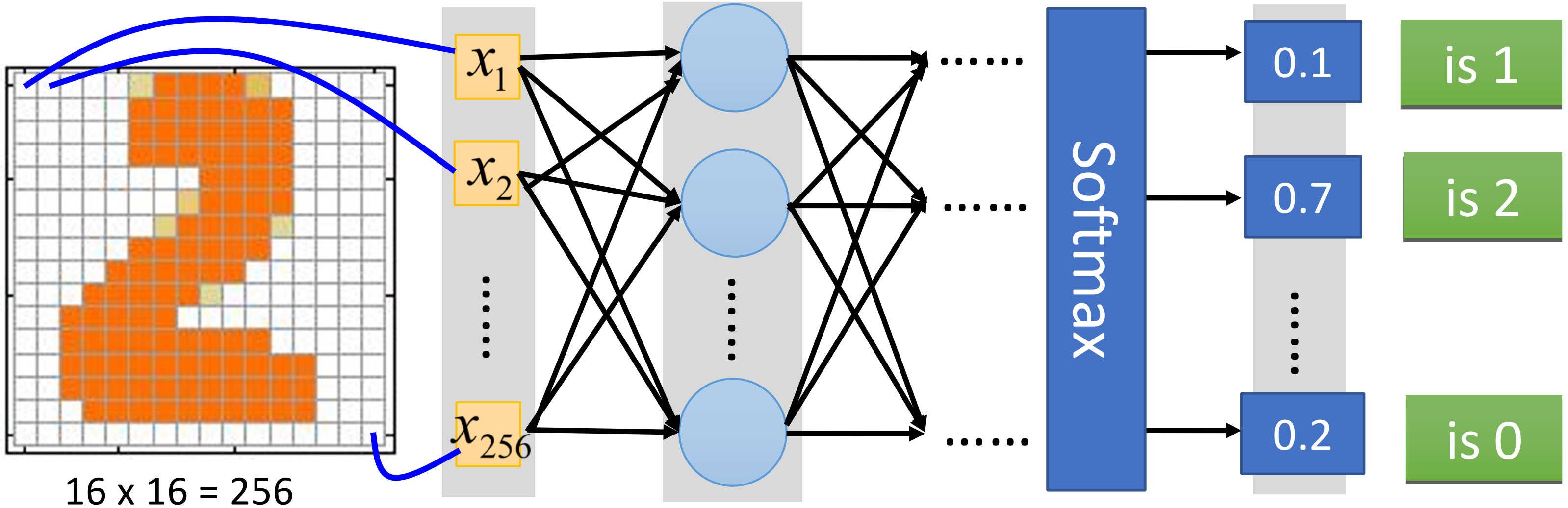


Backpropagation:

- Initialize the parameters
- Compute total error
- Then calculate gradient w.r.t each weight and eventually modify the weights to get better results.

Learning network parameters

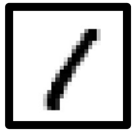
$$\theta = \{W^1, b^1, W^2, b^2, \cdots W^L, b^L\}$$



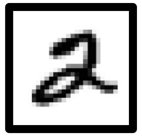
16 x 16 = 256

Orange→ 1
Black → 0

Aim: Find network parameters such that ......

Input  y_1 has the maximum value

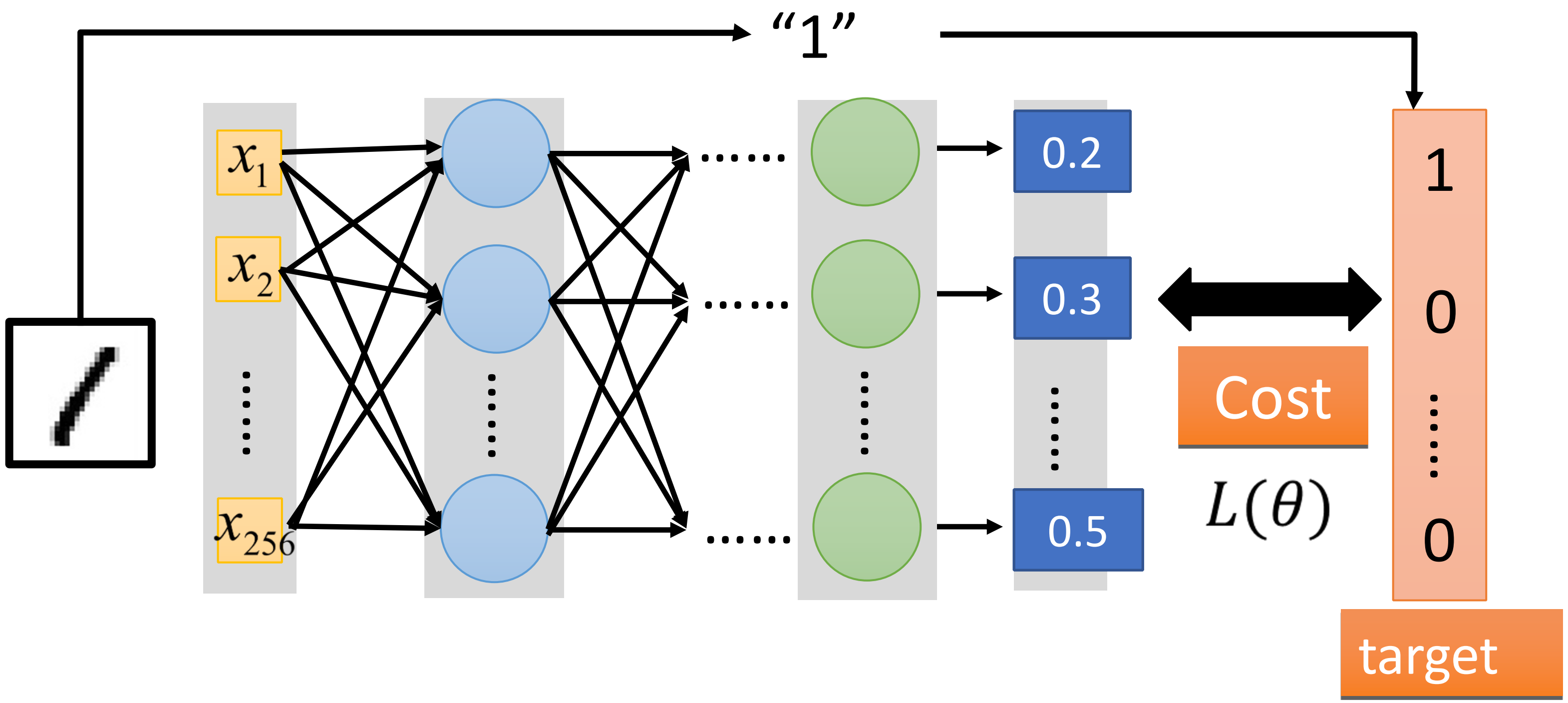
:

Input  y_2 has the maximum value

:

How can the neural network achieve this ?

Computing cost of a sample



Cost can be Euclidean distance or cross entropy between network output and target
Given a set of network parameters, each example has a cost value.

How to evaluate that how bad the network parameters performs on this task?

Total Cost: $$C(\theta) = \sum_{r=1}^{R} L^r(\theta)$$

Find the network parameters that minimize total cost

The Flow of Computations in Neural Networks

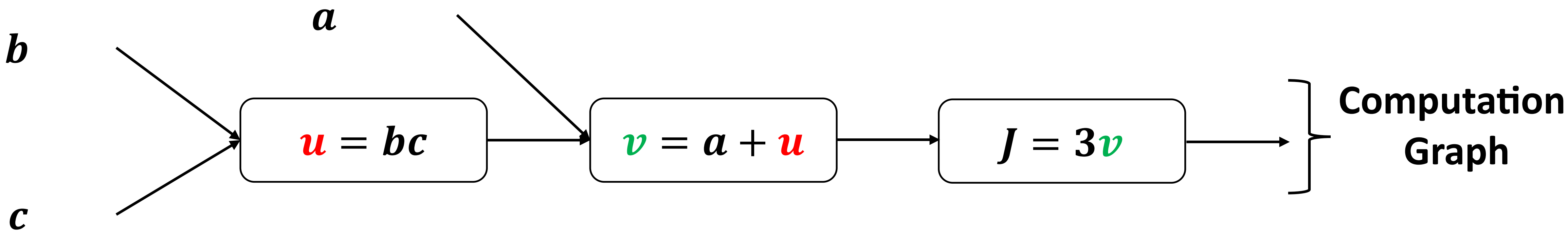
- The flow of computations in a neural network goes in two ways:
  1. Left-to-right: This is referred to as *forward propagation*, which results in computing the output of the network
  2. Right-to-left: This is referred to as *back propagation*, which results in computing the gradients (or derivatives)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

- The intuition behind this 2-way flow of computations can be explained through the concept of "computation graphs"
  - What is a computation graph?

What is a Computation Graph?

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

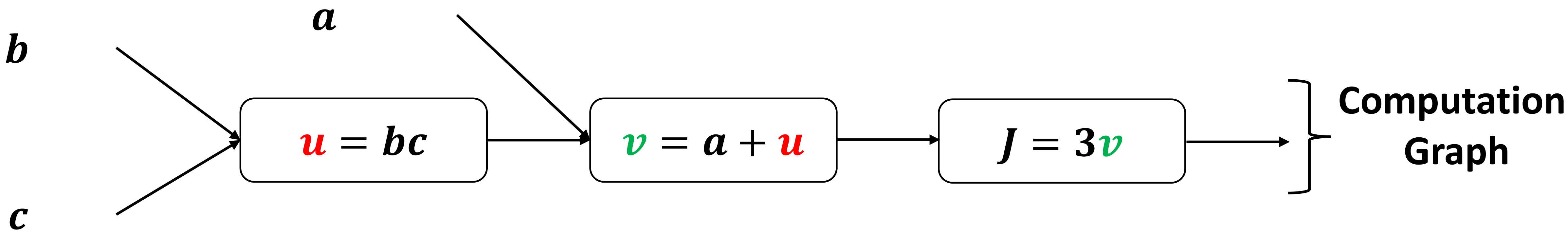
$$\left.J(a,b,c) = 3(\underbrace{\underbrace{a + \underbrace{bc}_{u}}_{v}}_{J})\right\} \quad \begin{aligned} u &= bc \\ v &= a + u \\ J &= 3v \end{aligned}$$



Forward Propagation

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

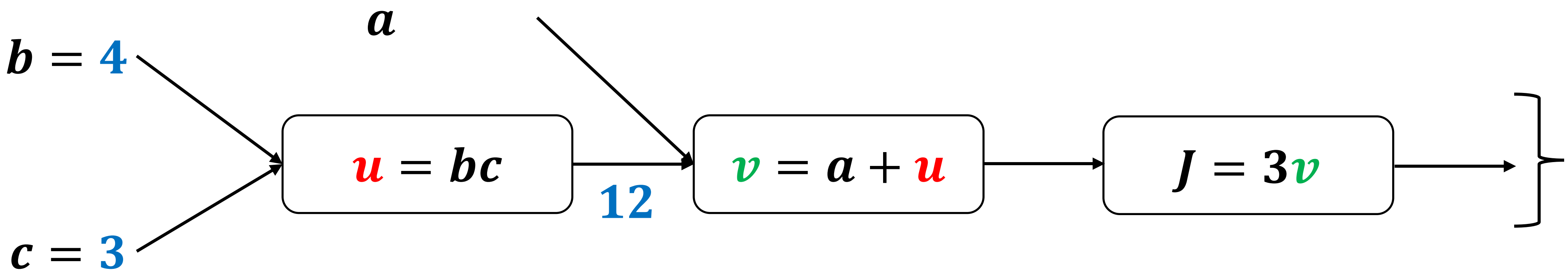
$$J(a,b,c) = \underbrace{3(\underbrace{a+\underbrace{bc}_{u}}_{v})}_{J} \quad \left.\vphantom{\begin{matrix}1\\1\\1\end{matrix}}\right\} \quad \begin{aligned} u &= bc \\ v &= a+u \\ J &= 3v \end{aligned}$$



Forward Propagation

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

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

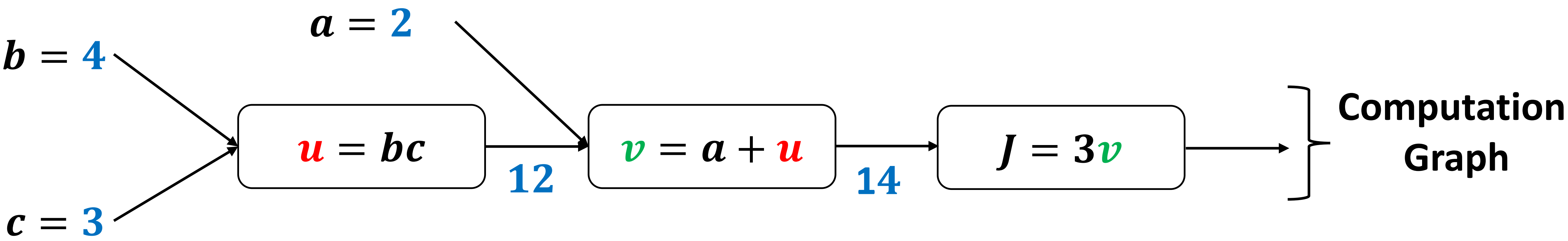


Computation Graph

Forward Propagation

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

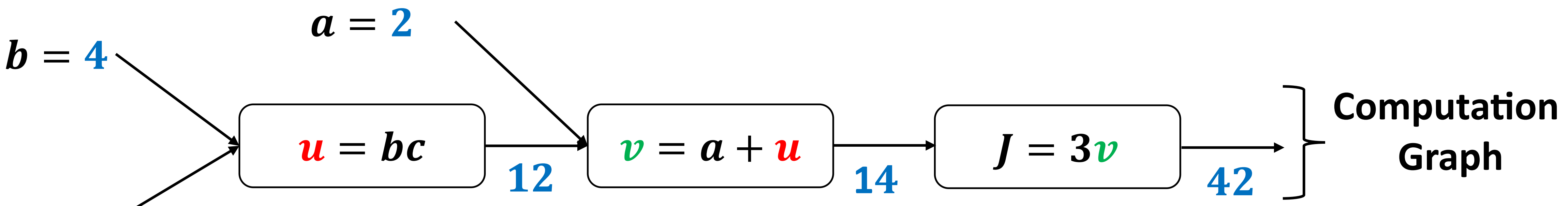
$$J(a,b,c) = 3\underbrace{(\underbrace{a+\underbrace{bc}_{u}}_{v})}_{J} \quad \left.\right\} \quad \begin{aligned} u &= bc \\ v &= a + u \\ J &= 3v \end{aligned}$$



Forward Propagation

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

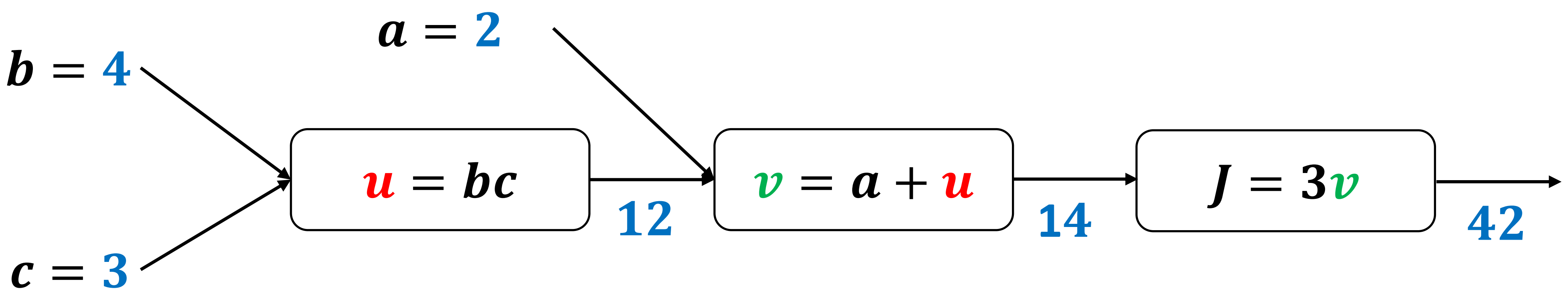
$$\left.\underbrace{3(\underbrace{a+\underbrace{bc}_{u}}_{v})}_{J} \quad J(a,b,c) = 3(a+bc)\right\} \qquad \begin{aligned} u &= bc \\ v &= a+u \\ J &= 3v \end{aligned}$$



Forward propagation allows computing J

Backward Propagation

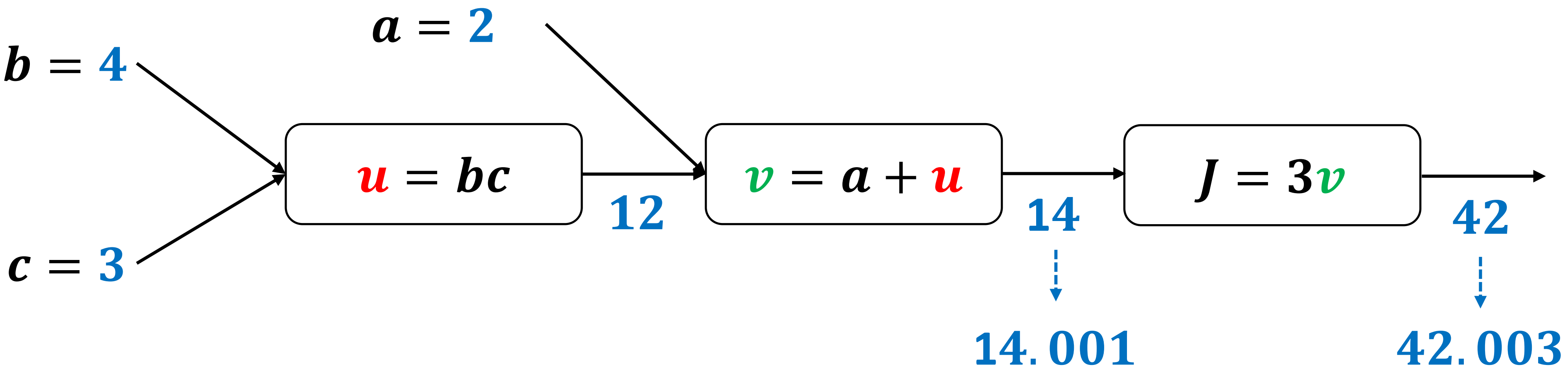
- Let us now compute the derivatives of the variables through the computation graph as follows:



$\frac{dJ}{dv}$ = Derivative of J with respect to v

Backward Propagation

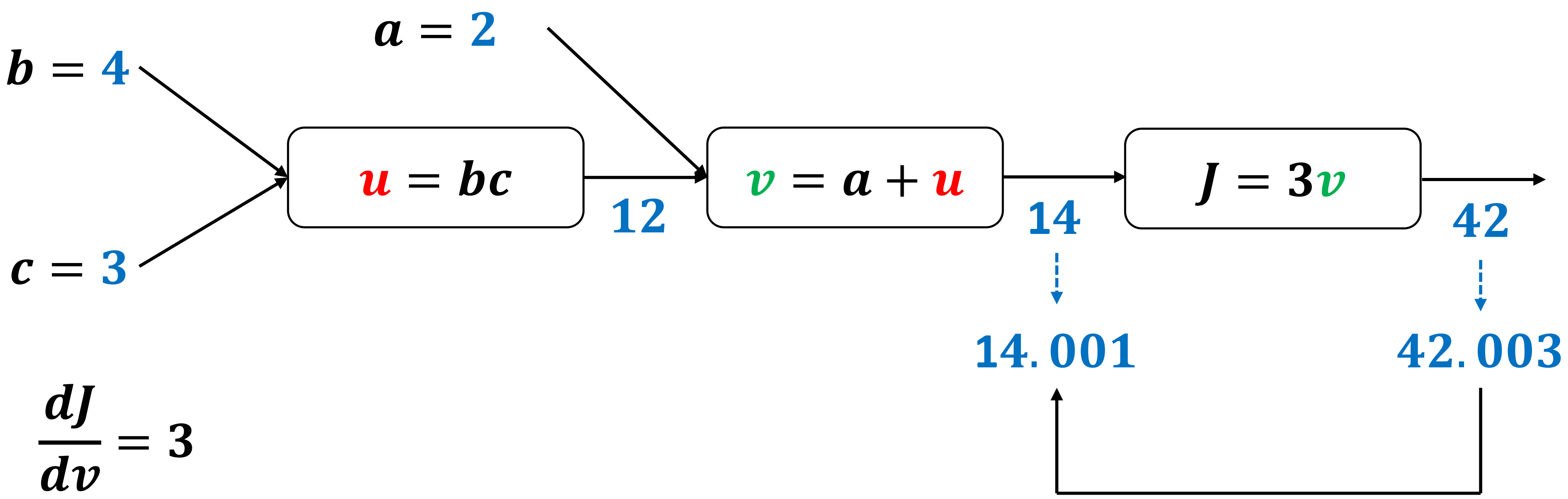
- Let us now compute the derivatives of the variables through the computation graph as follows:



$$\frac{dJ}{dv} = \text{If we change } \boldsymbol{v} \text{ a little bit, how would } \boldsymbol{J} \text{ change?}$$

Backward Propagation

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

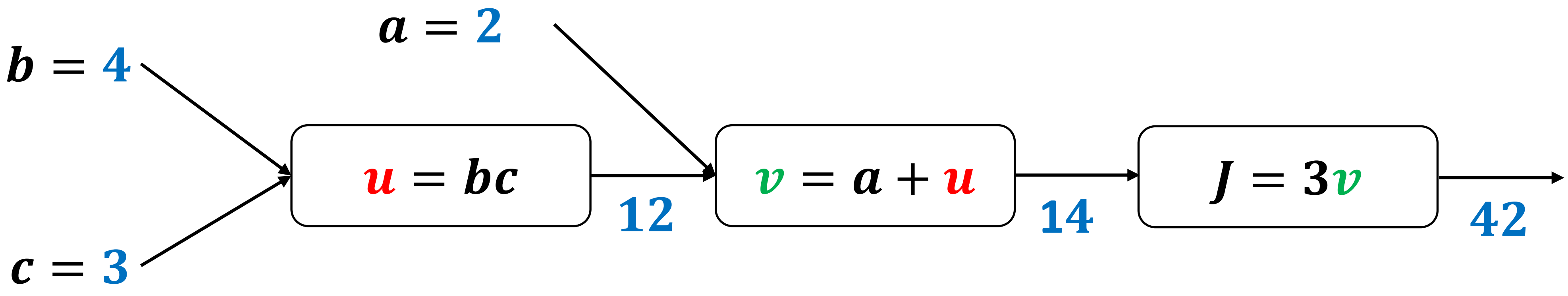


$$\frac{dJ}{dv} = 3$$

To compute the derivative of J with respect to v, we went *back* to v, nudged it, and measured the corresponding resultant increase on J

Backward Propagation

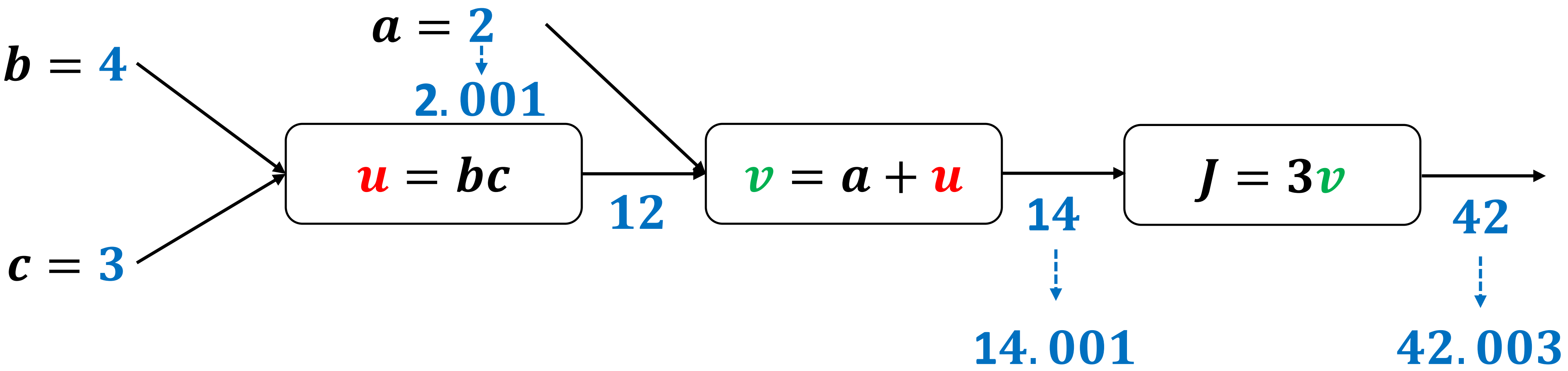
- Let us now compute the derivatives of the variables through the computation graph as follows:



$$\frac{dJ}{da} = \text{Derivative of } J \text{ with respect to } a$$

Backward Propagation

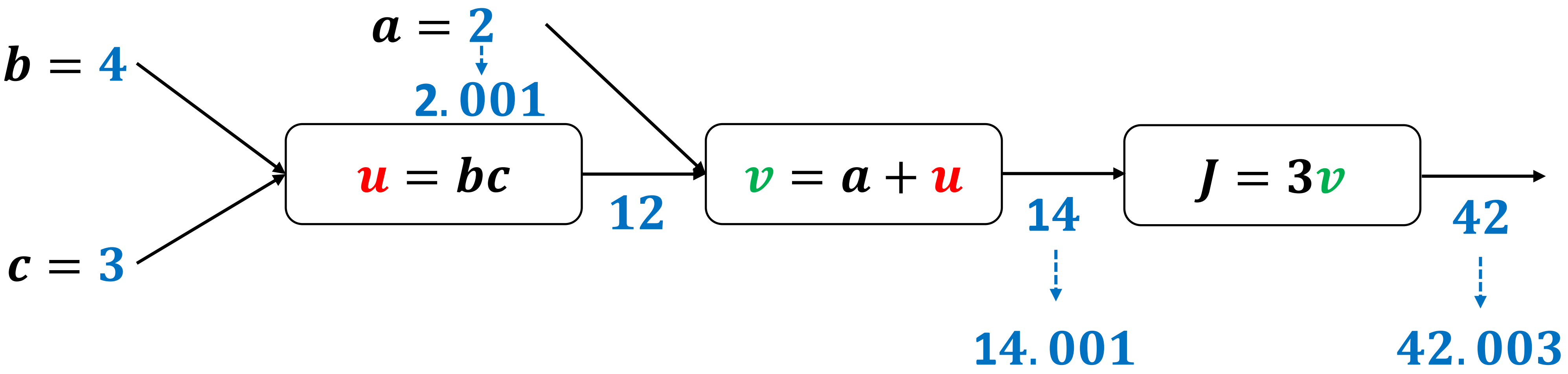
- Let us now compute the derivatives of the variables through the computation graph as follows:



$\frac{dJ}{da}$ = If we change a a little bit, how would J change?

Backward Propagation

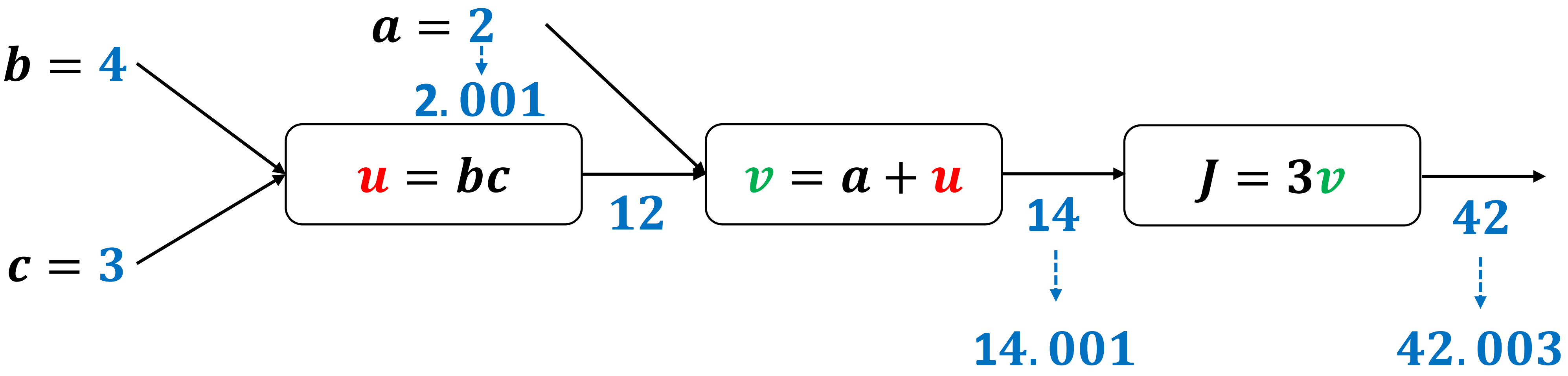
- Let us now compute the derivatives of the variables through the computation graph as follows:



$$\frac{dJ}{da} = 3$$

Backward Propagation

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

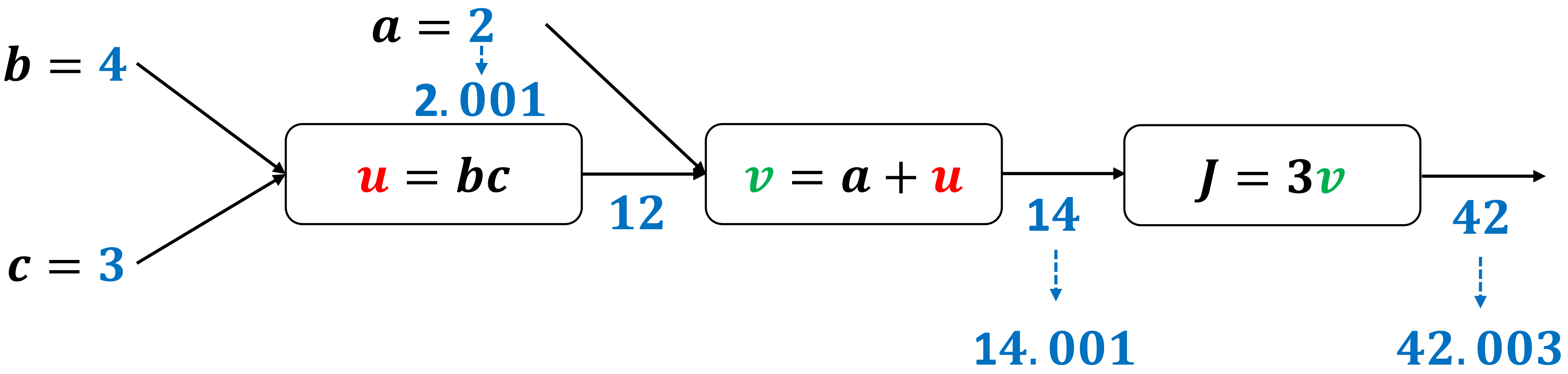


$$\frac{dJ}{da} = 3 = \qquad \frac{dv}{da}$$

The change in a caused a change in v

Backward Propagation

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

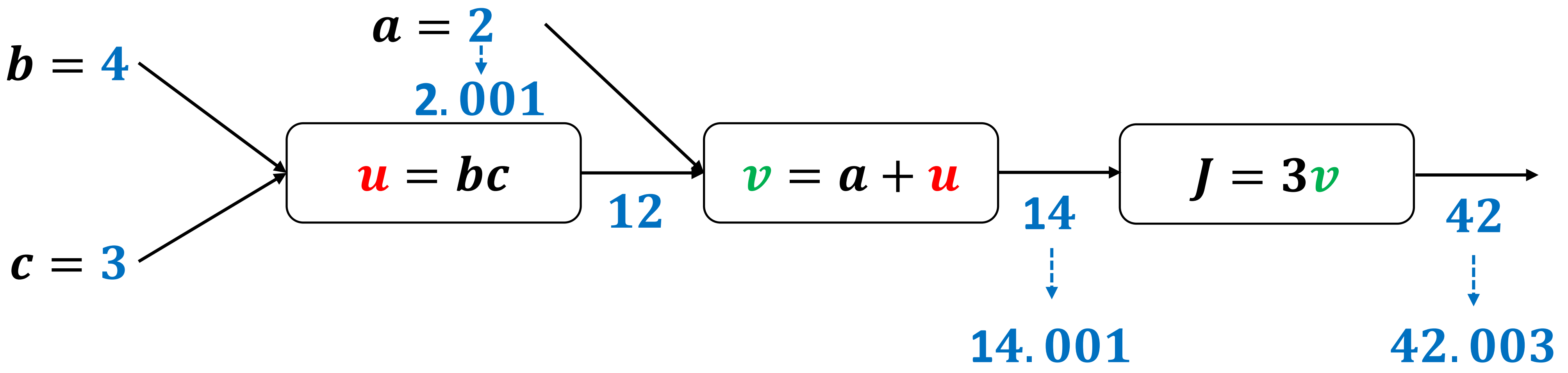


$$\frac{dJ}{da} = 3 = \frac{dJ}{dv} \times \frac{dv}{da}$$

And the change in v caused a change in J

Backward Propagation

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

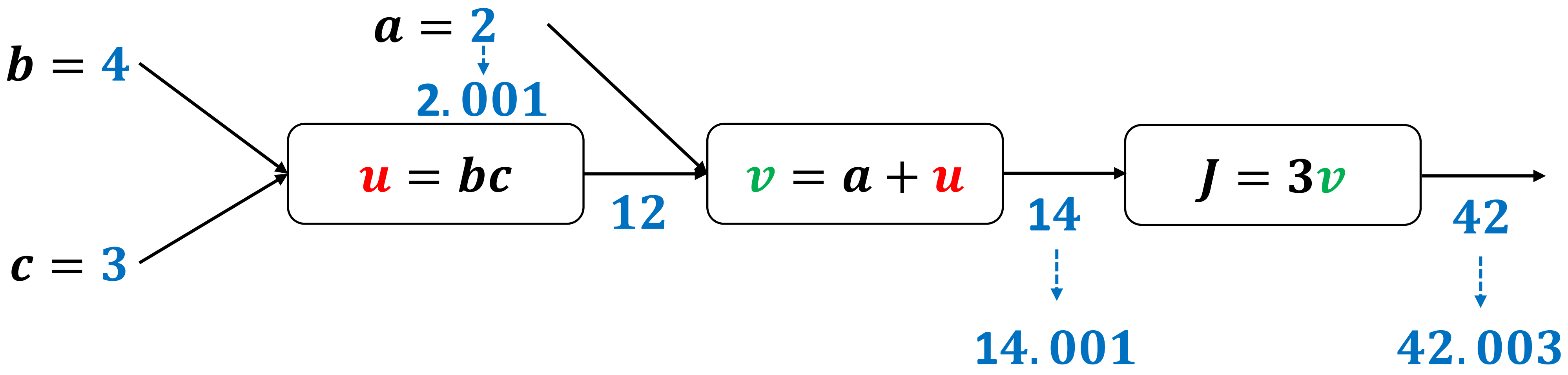


$$\frac{dJ}{da} = 3 = \frac{dJ}{dv} \times \frac{dv}{da}$$

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

Backward Propagation

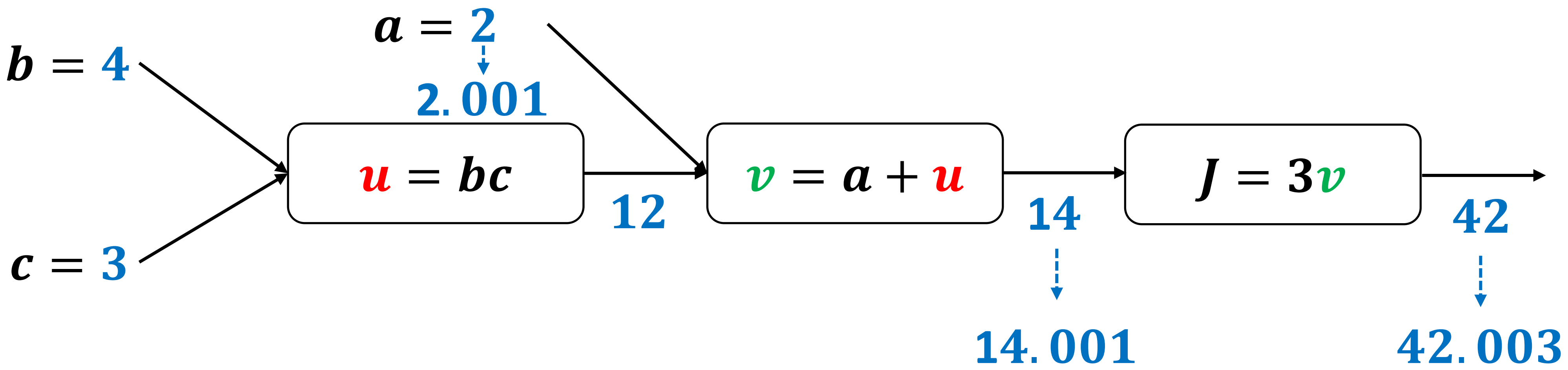
- Let us now compute the derivatives of the variables through the computation graph as follows:



$$\frac{dJ}{da} = 3 = \frac{dJ}{dv} \times 1$$

Backward Propagation

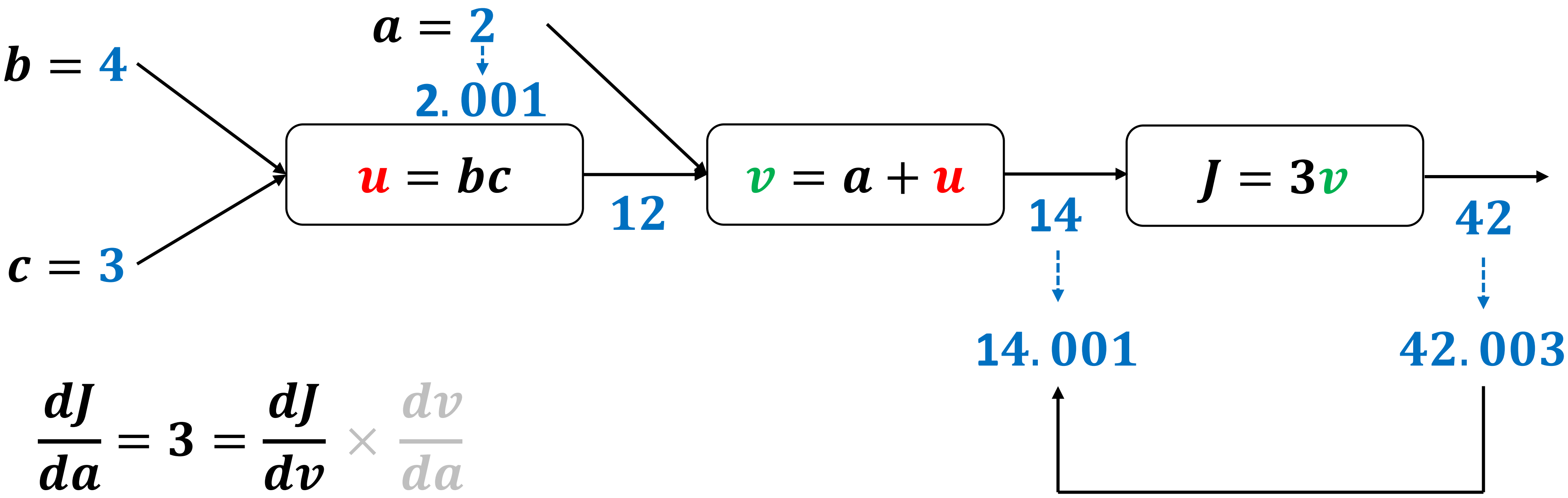
- Let us now compute the derivatives of the variables through the computation graph as follows:



$$\frac{dJ}{da} = 3 = 3 \times 1$$

Backward Propagation

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

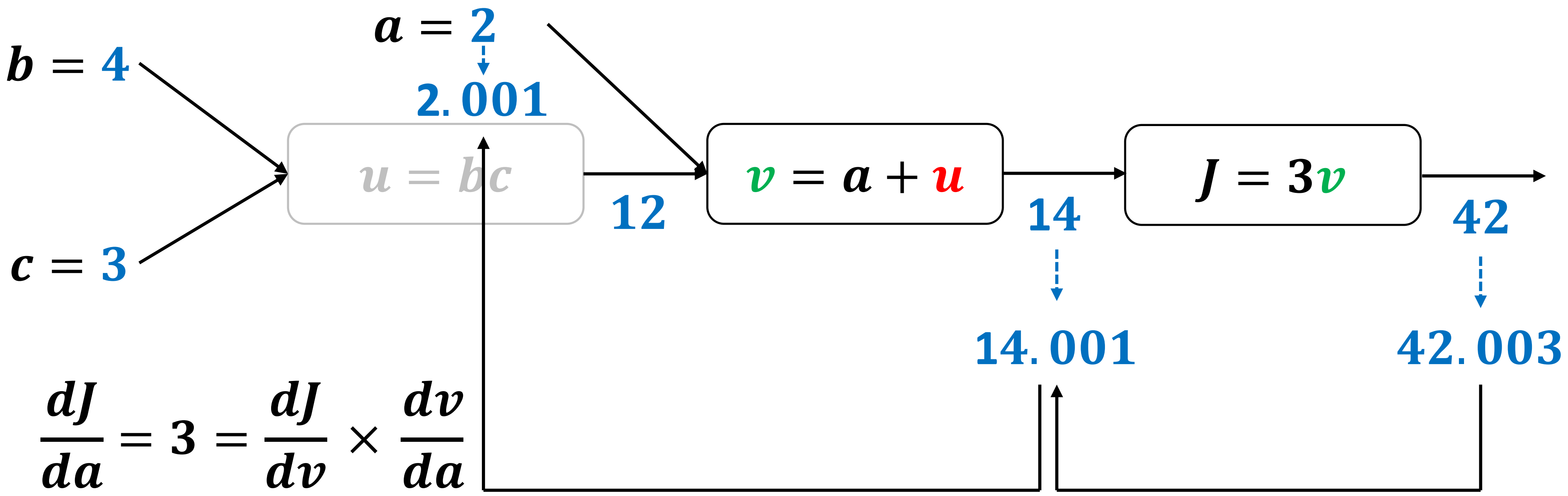


$$\frac{dJ}{da} = 3 = \frac{dJ}{dv} \times \frac{dv}{da}$$

In essence, to compute the derivative of J with respect to a, we had to go *back* to v, nudge it a little bit, and measure the corresponding resultant increase on J

Backward Propagation

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

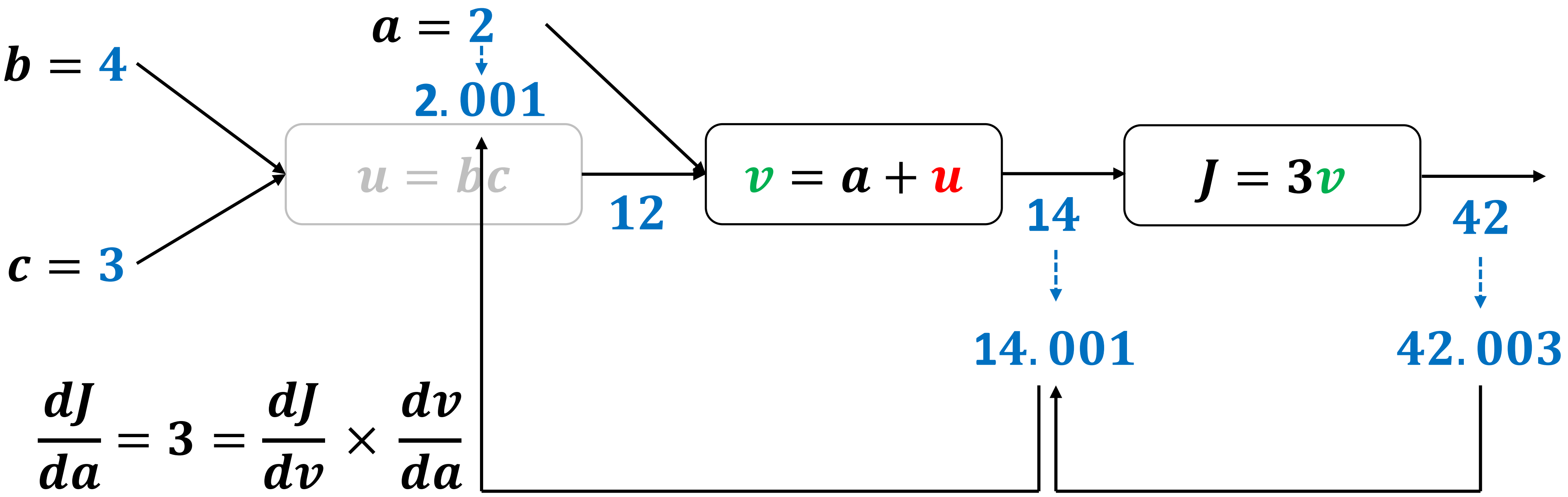


$$\frac{dJ}{da} = 3 = \frac{dJ}{dv} \times \frac{dv}{da}$$

Then, we had to go *back* to $\boldsymbol{a}$, nudge it a little bit, and measure the corresponding resultant increase on $\boldsymbol{v}$

Backward Propagation

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

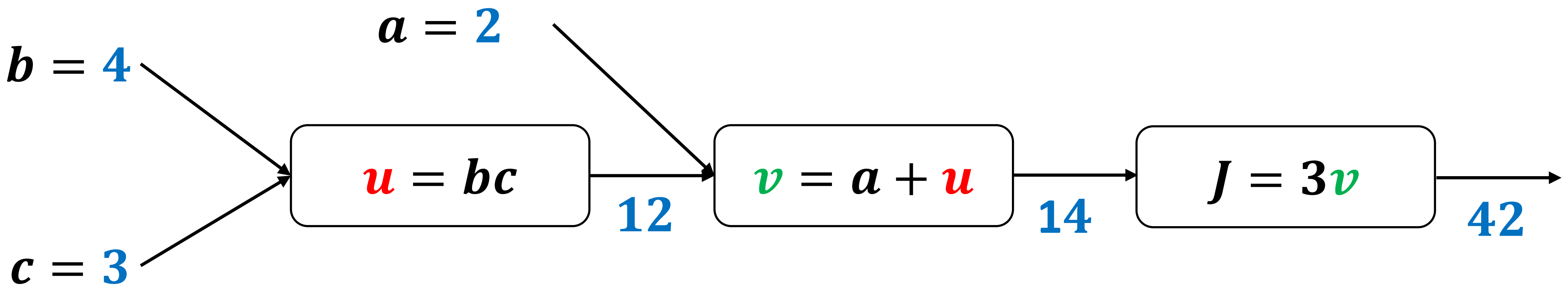


$$\frac{dJ}{da} = 3 = \frac{dJ}{dv} \times \frac{dv}{da}$$

Then, we multiplied the changes together (i.e., we applied the chain rule!)

Backward Propagation

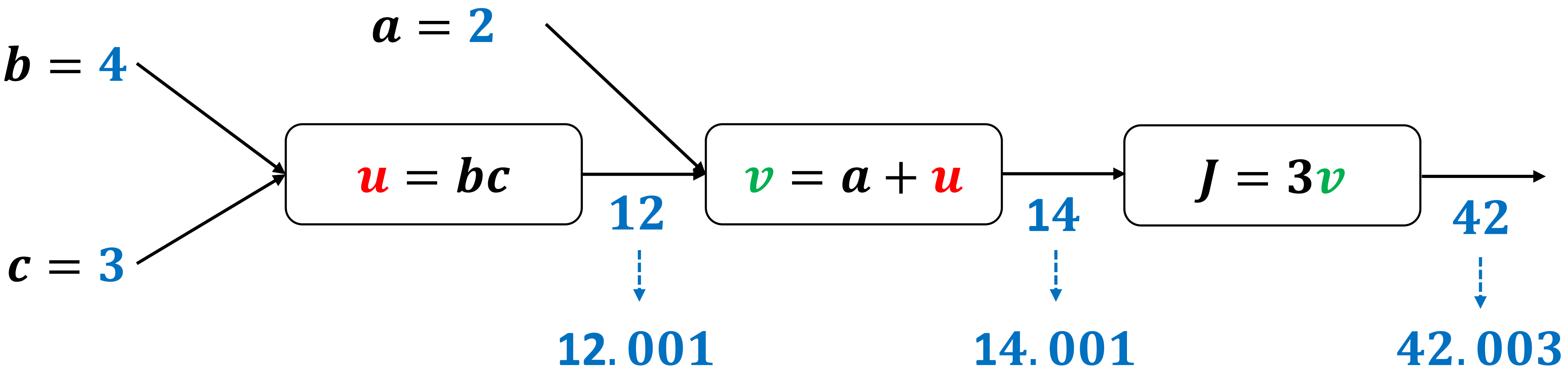
- Let us now compute the derivatives of the variables through the computation graph as follows:



$$\frac{dJ}{du} = \text{Derivative of } J \text{ with respect to } u$$

Backward Propagation

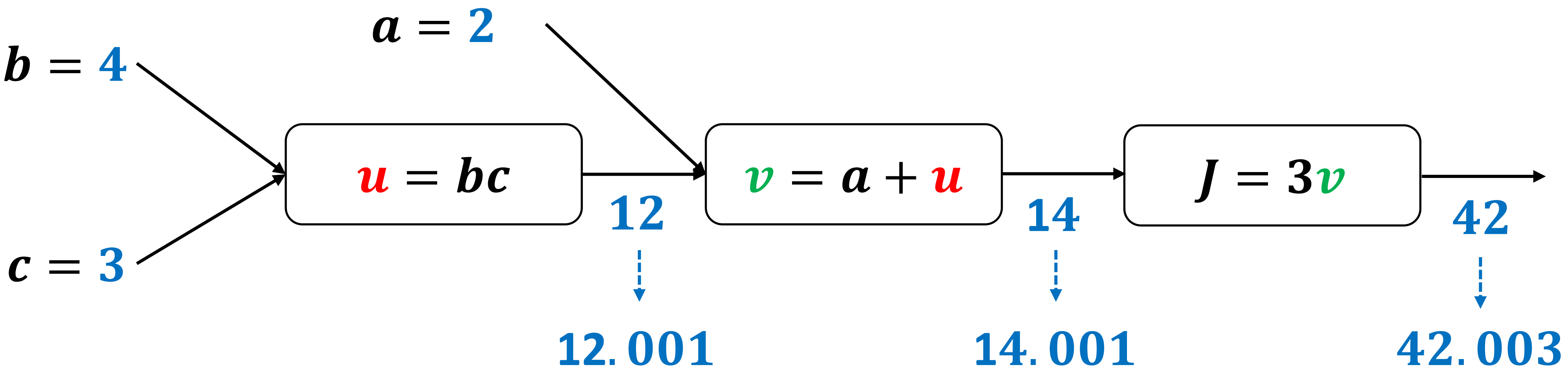
- Let us now compute the derivatives of the variables through the computation graph as follows:



$$\frac{dJ}{du}$$ = If we change u a little bit, how would J change?

Backward Propagation

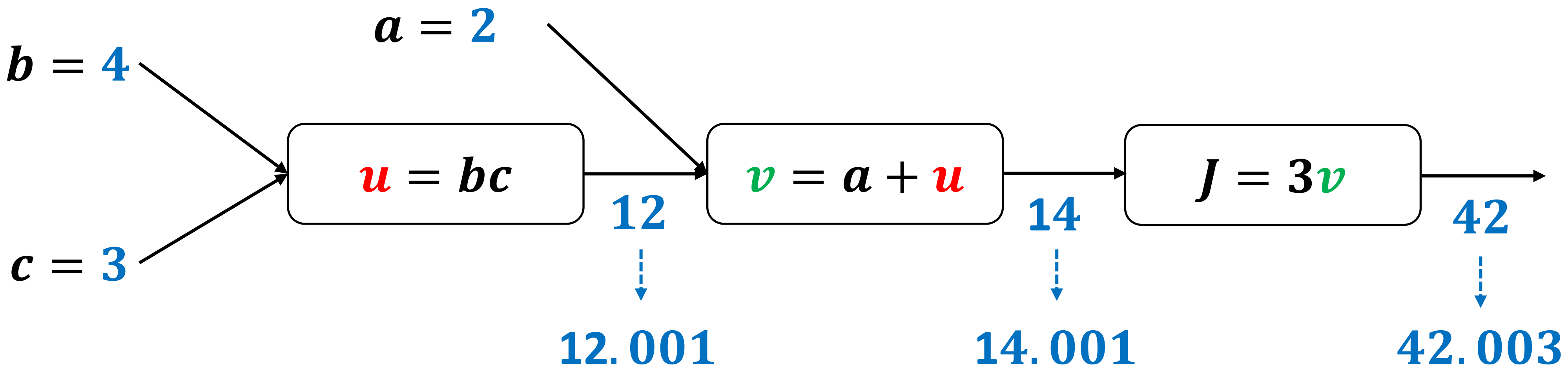
- Let us now compute the derivatives of the variables through the computation graph as follows:



$$\frac{dJ}{du} = 3$$

Backward Propagation

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

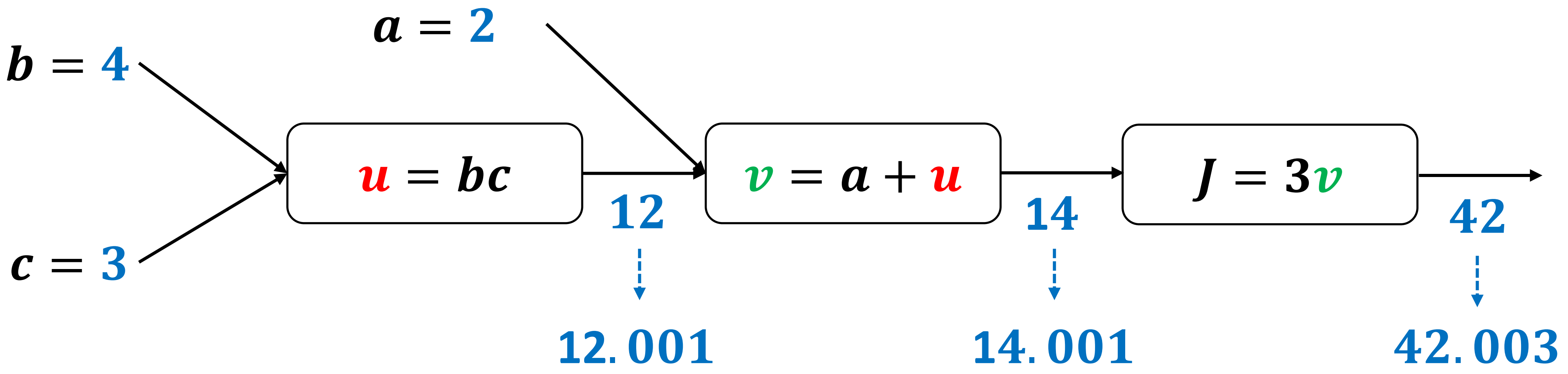


$$\frac{dJ}{du} = 3 = \quad \frac{dv}{du}$$

The change in u caused a change in v

Backward Propagation

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

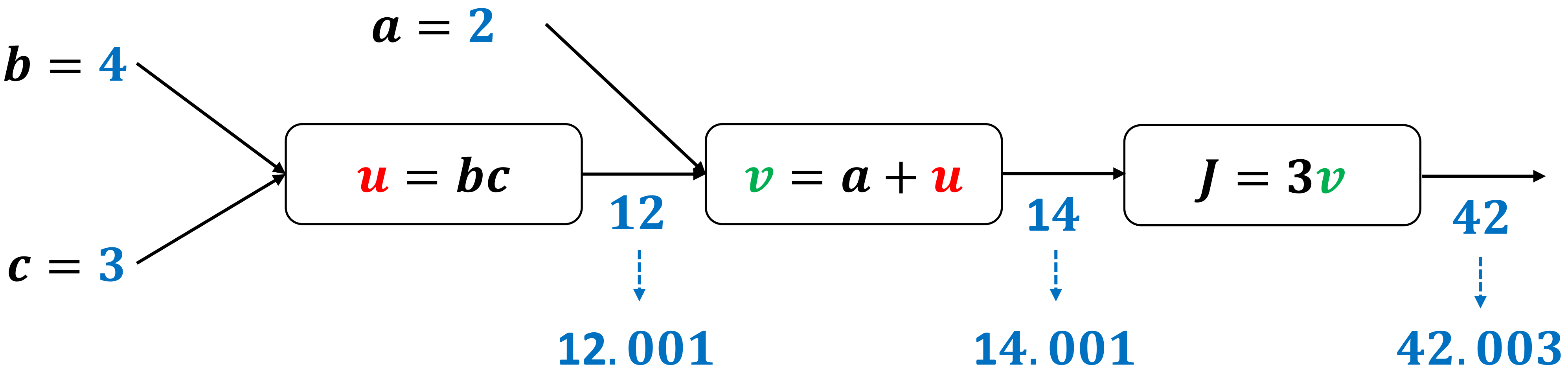


$$\frac{dJ}{du} = 3 = \frac{dJ}{dv} \times \frac{dv}{du}$$

And the change in v caused a change in J

Backward Propagation

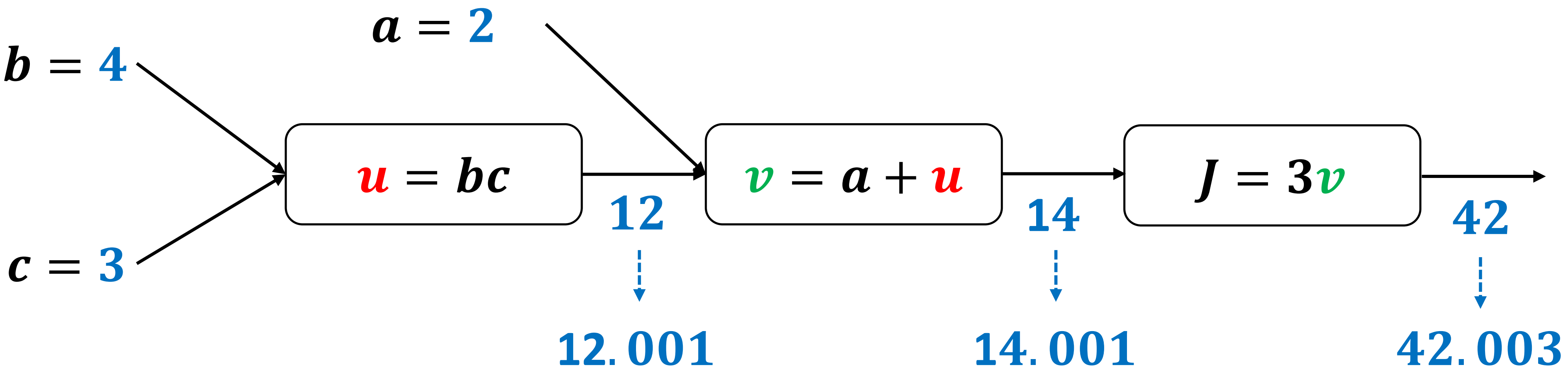
- Let us now compute the derivatives of the variables through the computation graph as follows:



$$\frac{dJ}{du} = 3 = \frac{dJ}{dv} \times 1$$

Backward Propagation

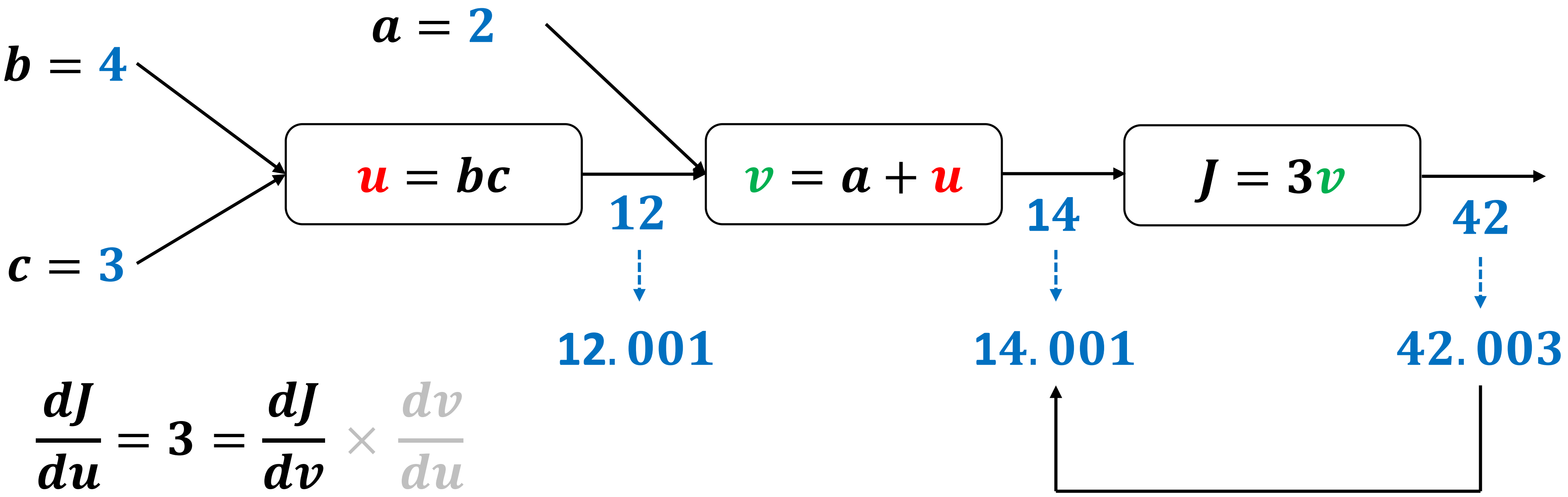
- Let us now compute the derivatives of the variables through the computation graph as follows:



$$\frac{dJ}{du} = 3 = 3 \times 1$$

Backward Propagation

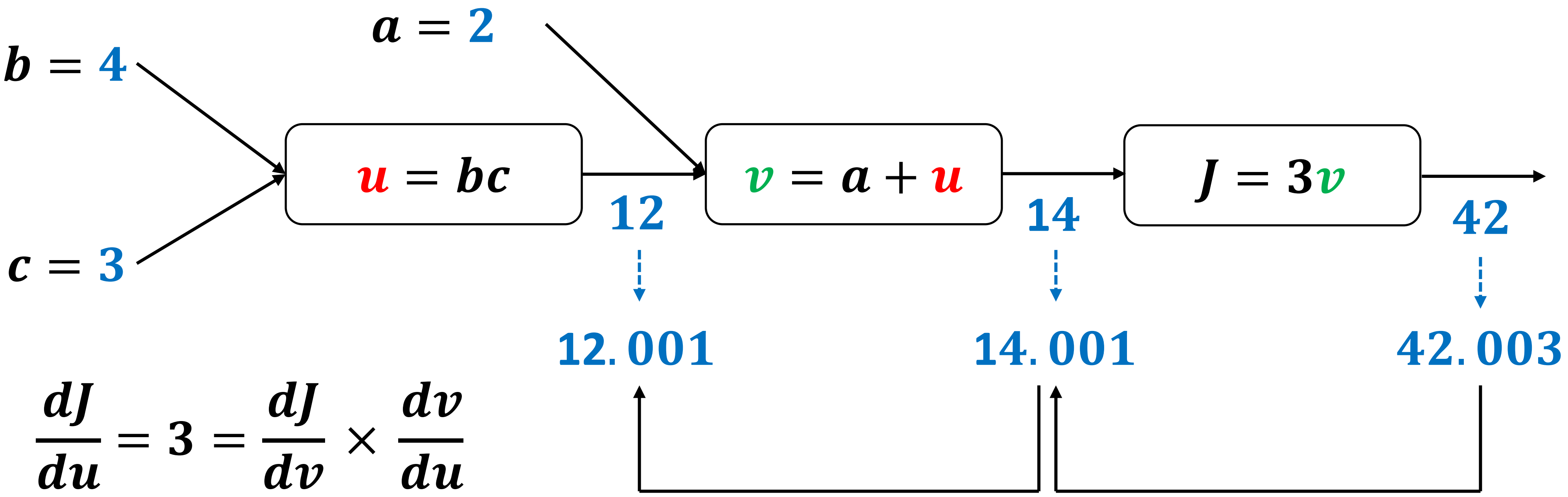
- Let us now compute the derivatives of the variables through the computation graph as follows:



$$\frac{dJ}{du} = 3 = \frac{dJ}{dv} \times \frac{dv}{du}$$

Backward Propagation

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

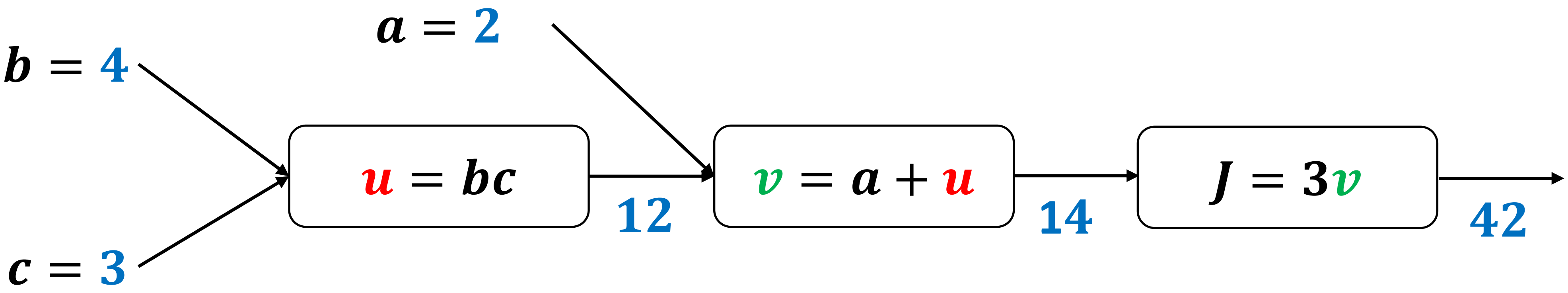


$$\frac{dJ}{du} = 3 = \frac{dJ}{dv} \times \frac{dv}{du}$$

Same as before, we had to go back to $\boldsymbol{v}$ then to $\boldsymbol{u}$ in order to compute the derivative of $\boldsymbol{J}$

Backward Propagation

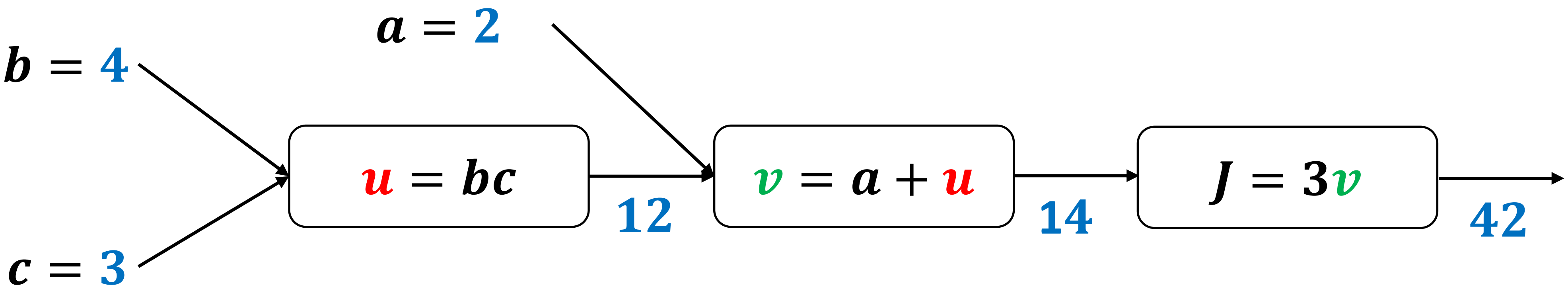
- Let us now compute the derivatives of the variables through the computation graph as follows:



$$\frac{dJ}{db} = \text{Derivative of } J \text{ with respect to } b$$

Backward Propagation

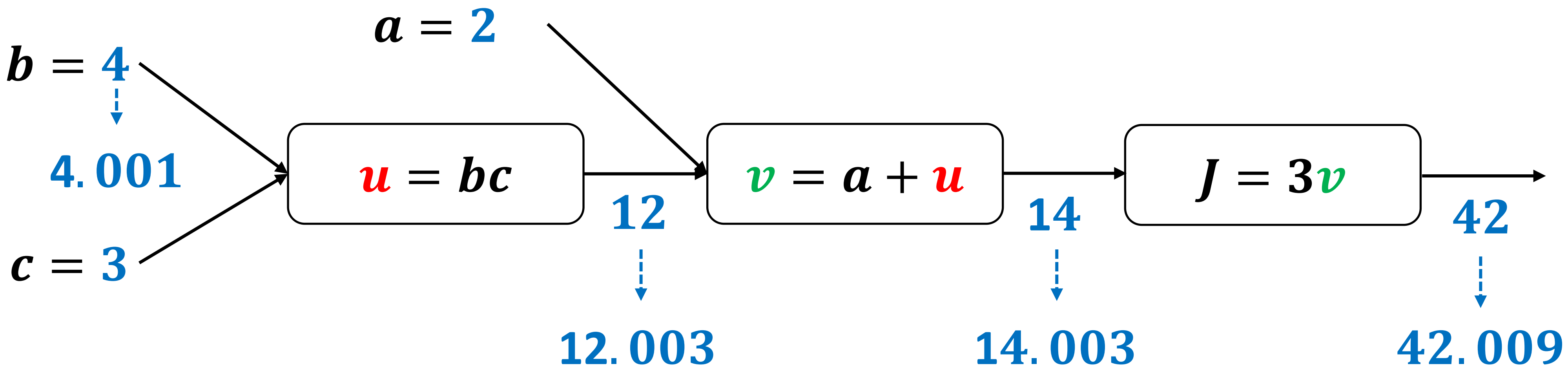
- Let us now compute the derivatives of the variables through the computation graph as follows:



$$\frac{dJ}{db}$$ = If we change b a little bit, how would J change?

Backward Propagation

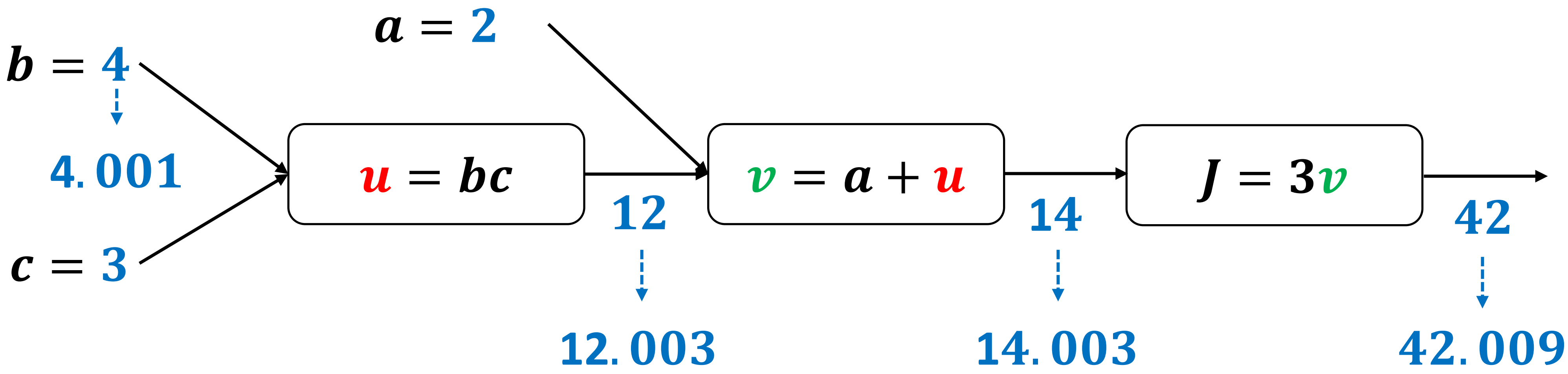
- Let us now compute the derivatives of the variables through the computation graph as follows:



$$\frac{dJ}{db} = ? = \underset{3}{\frac{dJ}{dv}} \times \underset{1}{\frac{dv}{du}} \times \underset{3}{\frac{du}{db}}$$

Backward Propagation

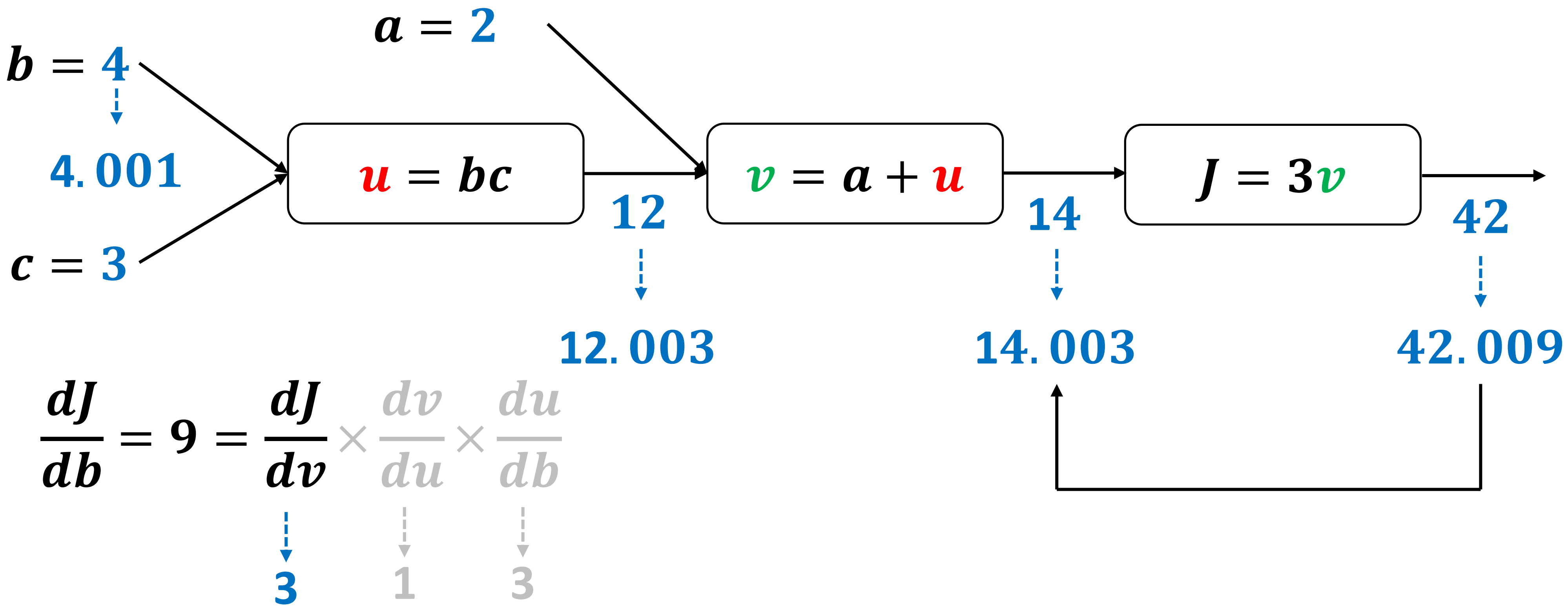
- Let us now compute the derivatives of the variables through the computation graph as follows:



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

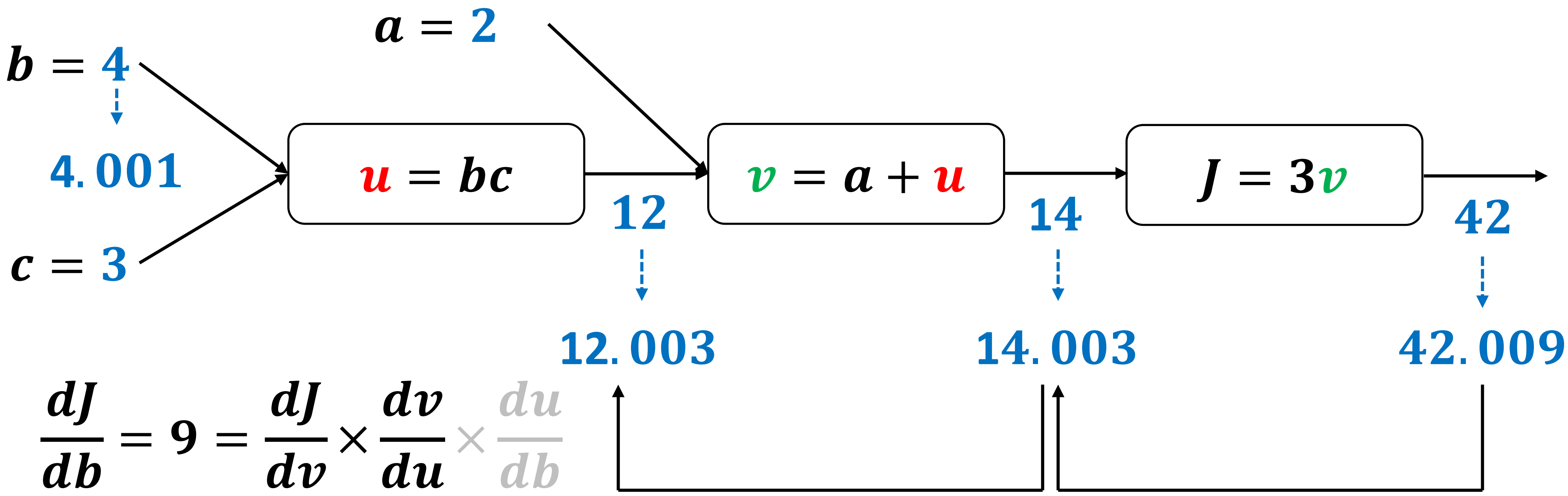
Backward Propagation

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



Backward Propagation

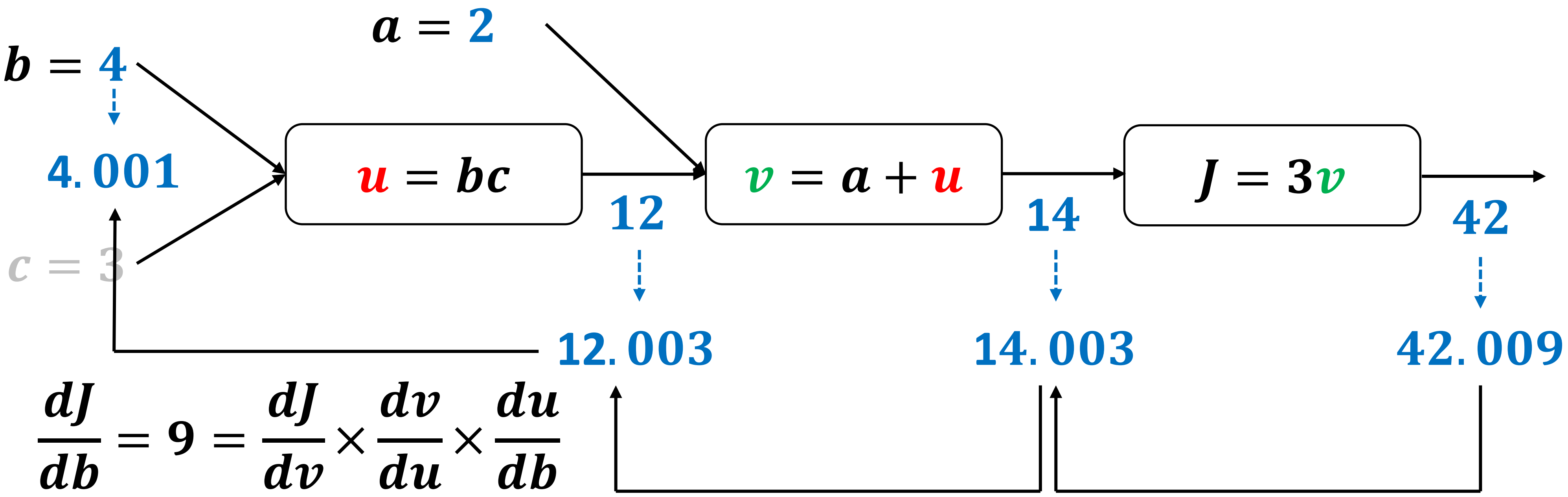
- Let us now compute the derivatives of the variables through the computation graph as follows:



$$\frac{dJ}{db} = 9 = \underbrace{\frac{dJ}{dv}}_{3} \times \underbrace{\frac{dv}{du}}_{1} \times \underbrace{\frac{du}{db}}_{3}$$

Backward Propagation

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

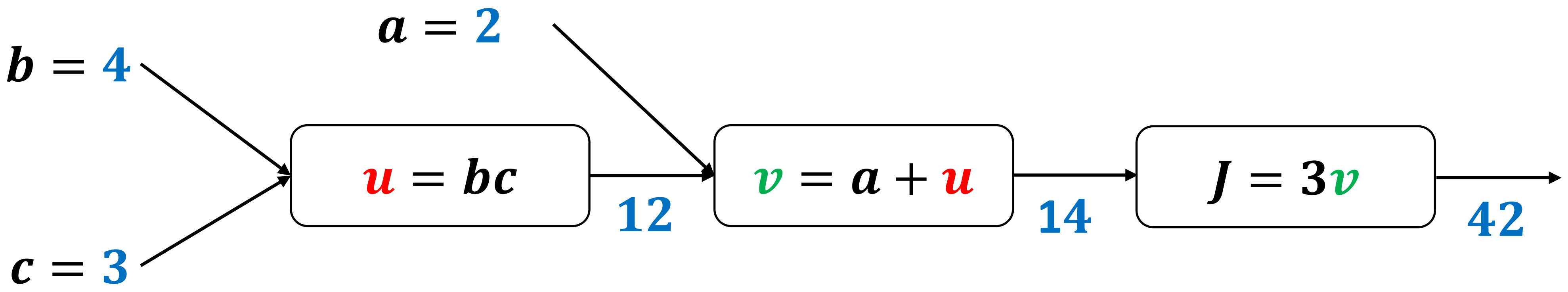


$$\frac{dJ}{db} = 9 = \frac{dJ}{dv} \times \frac{dv}{du} \times \frac{du}{db}$$

3 1 3

Backward Propagation

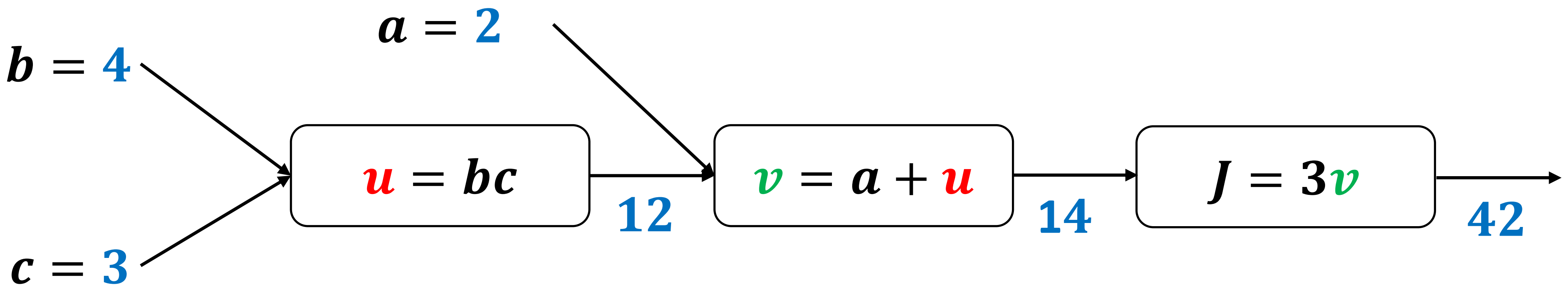
- Let us now compute the derivatives of the variables through the computation graph as follows:



$$\frac{dJ}{dc} = \text{Derivative of } J \text{ with respect to } c$$

Backward Propagation

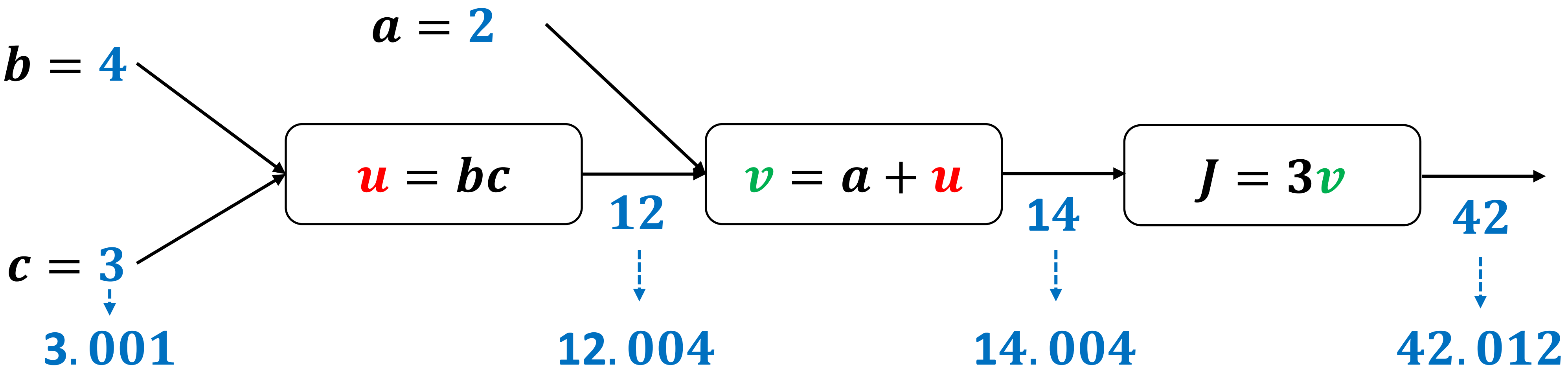
- Let us now compute the derivatives of the variables through the computation graph as follows:



$\frac{dJ}{dc} =$ If we change c a little bit, how would J change?

Backward Propagation

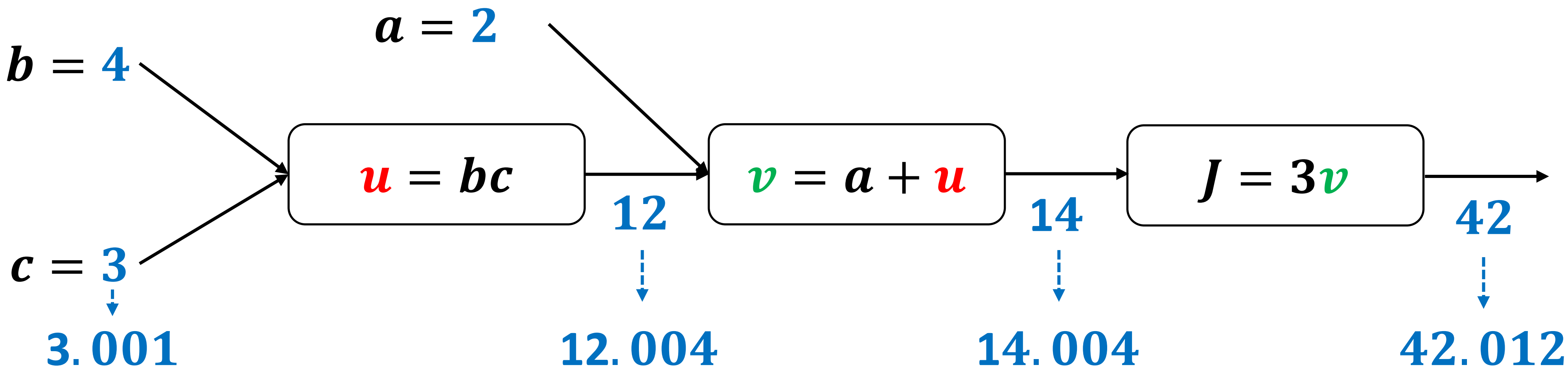
- Let us now compute the derivatives of the variables through the computation graph as follows:



$$\frac{dJ}{dc} = ? = \underbrace{\frac{dJ}{dv}}_{3} \times \underbrace{\frac{dv}{du}}_{1} \times \underbrace{\frac{du}{dc}}_{4}$$

Backward Propagation

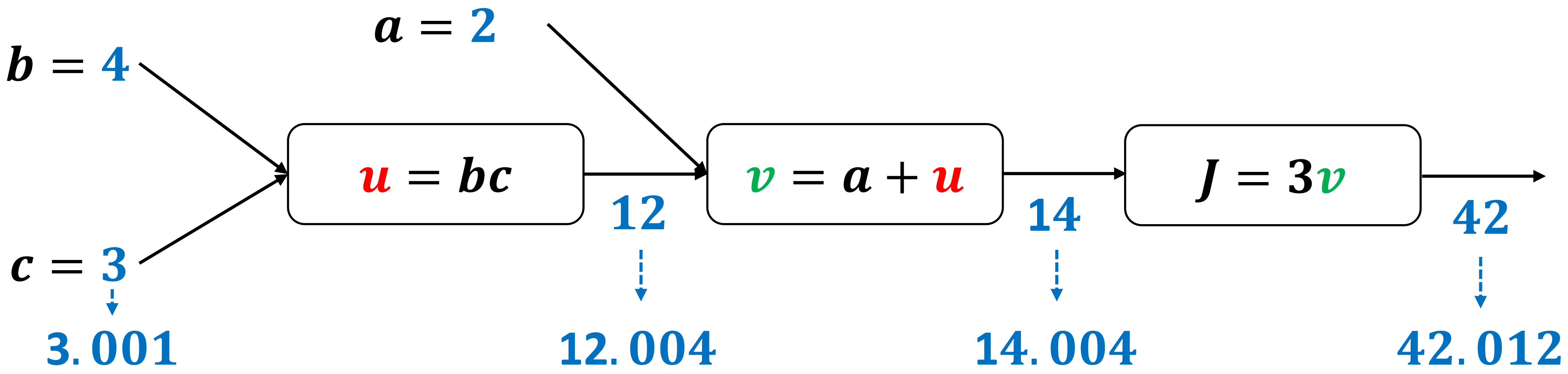
- Let us now compute the derivatives of the variables through the computation graph as follows:



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

Backward Propagation

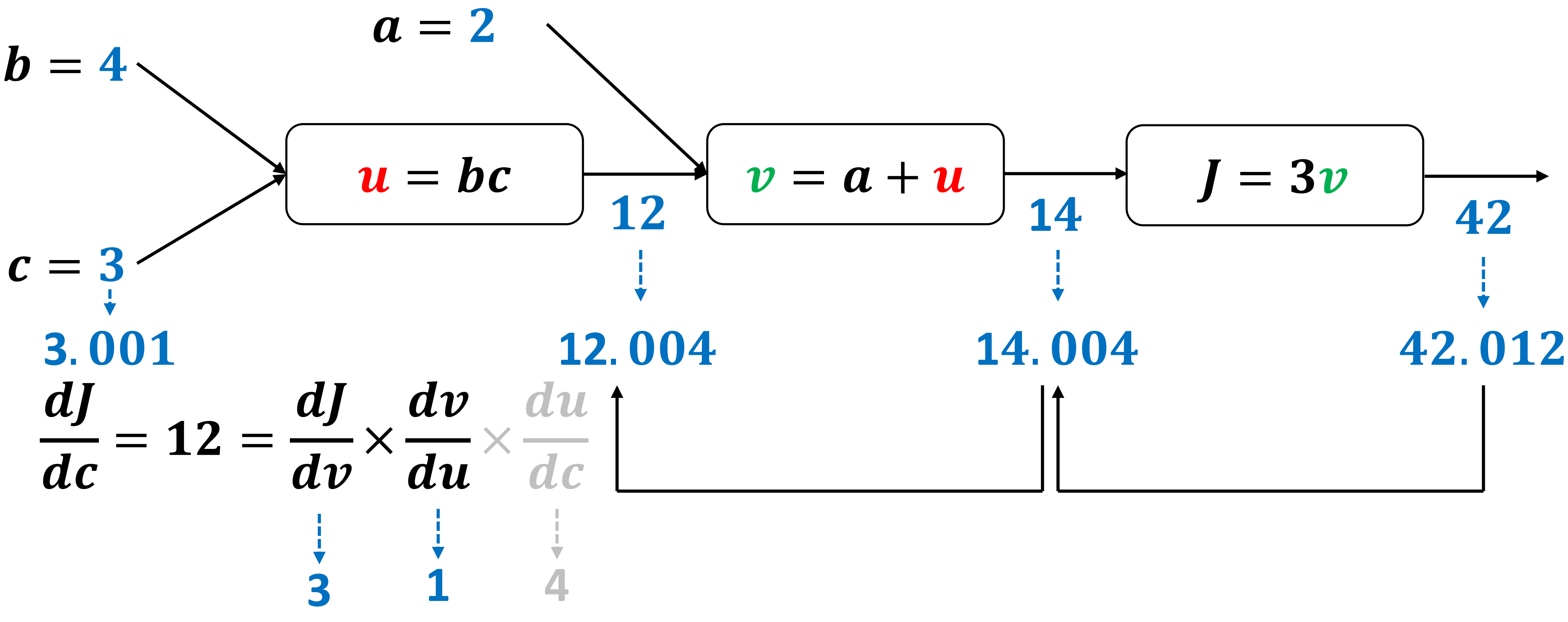
- Let us now compute the derivatives of the variables through the computation graph as follows:



$$\frac{dJ}{dc} = 12 = \underbrace{\frac{dJ}{dv}}_{3} \times \underbrace{\frac{dv}{du}}_{1} \times \underbrace{\frac{du}{dc}}_{4}$$

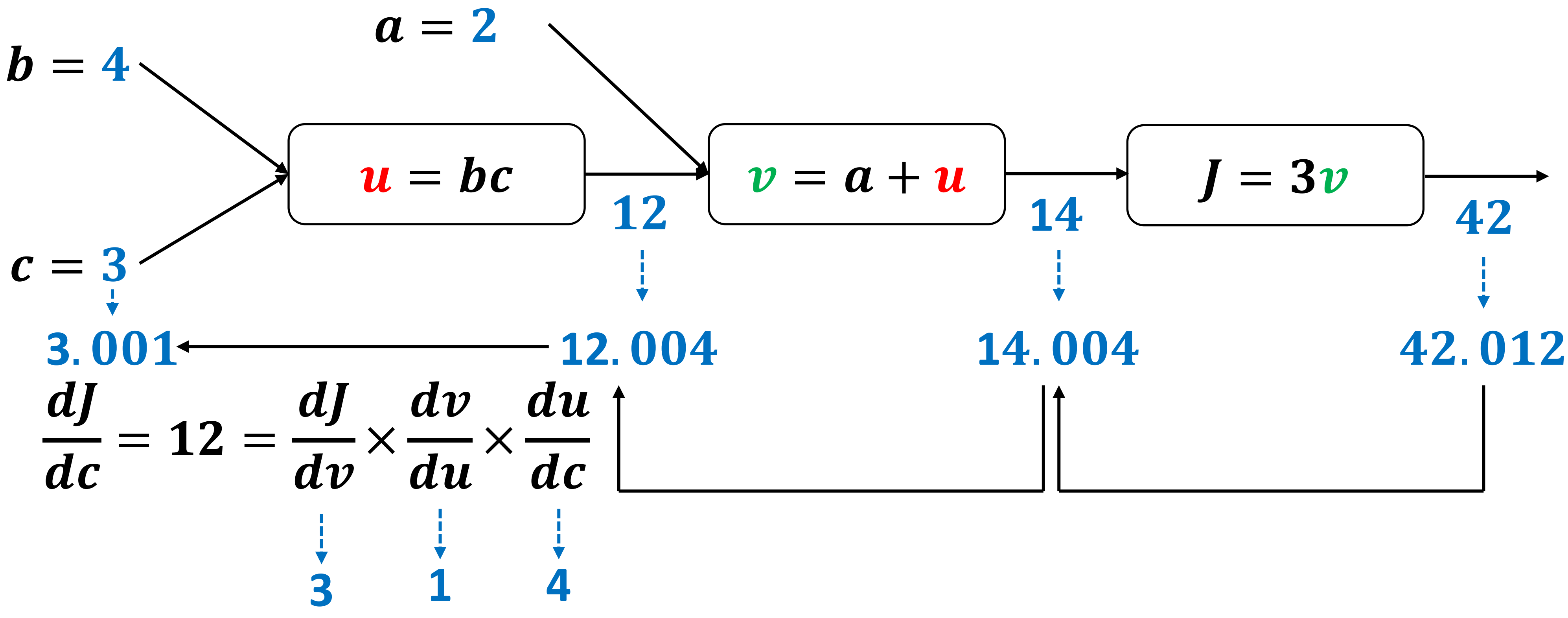
Backward Propagation

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



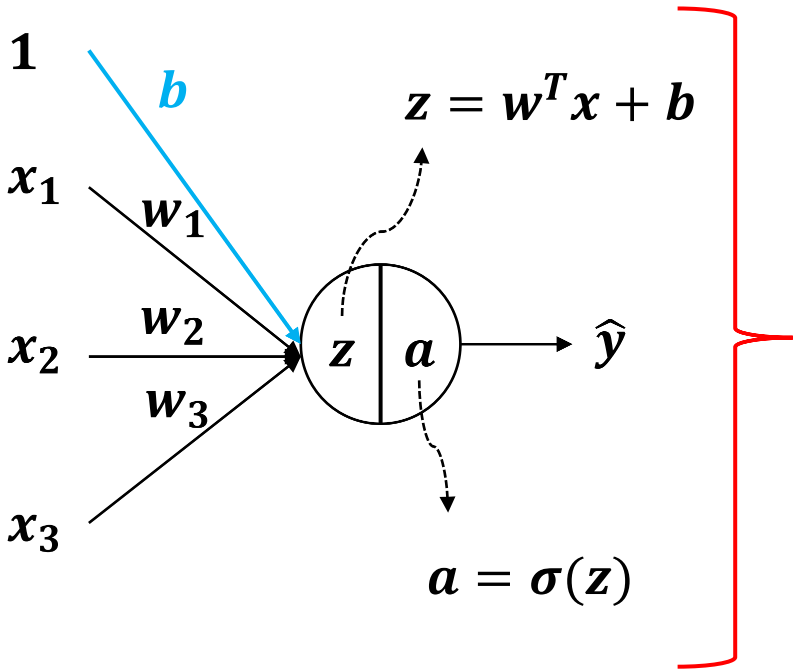
Backward Propagation

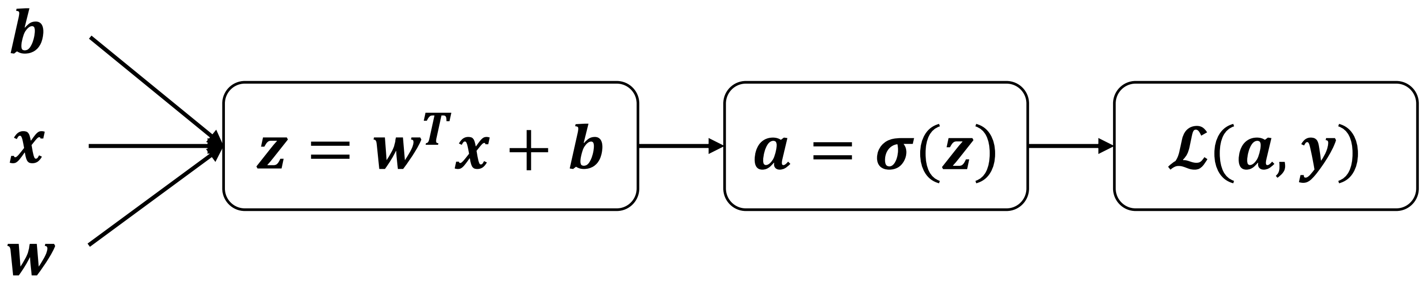
- Let us now compute the derivatives of the variables through the computation graph as follows:



The Computation Graph of Logistic Regression

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

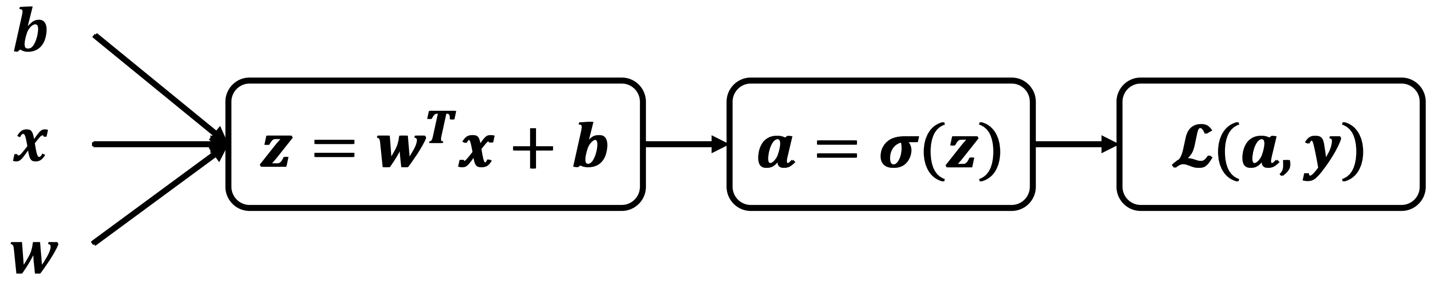




Where $b = 1$, $w = [w_1, w_2, w_3]$, $x = [x_1, x_2, x_3]$, and $\mathcal{L}(a, y)$ is the cost (or ***loss***) function

Forward Propagation

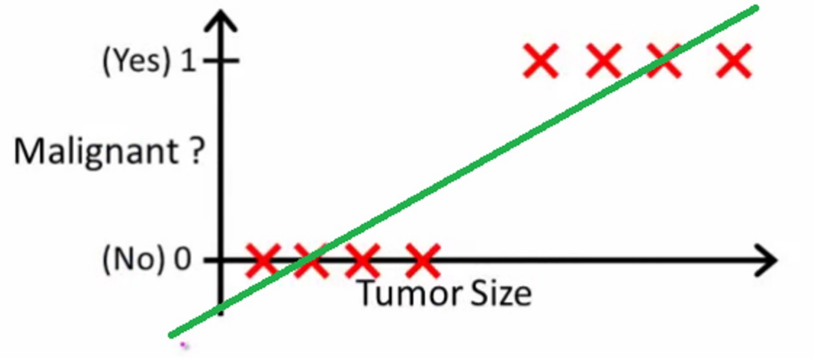
- The loss function can be computed by moving from left to right



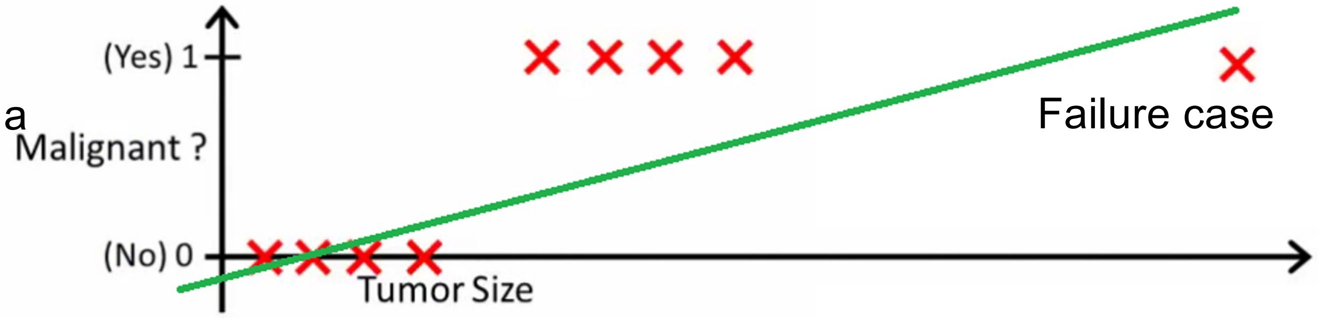
Recap: Using Linear Regression for Binary Classification

Example

- Using linear regression, fit a polynomial (line in this case) through the training data of type {tumor size, tumor type}.
- Malignant tumors means 1 and non-malignant means 0.
- Intuition: All tumors larger certain threshold are malignant
- Green line is the model $h(x)$.
- Prediction Rule: If $h(x)$ > 0.5, for any given tumor size x, predict malignant tumor and benign otherwise.



Linear regression gives a raw number but logistic regression tells probability that x belongs to the "positive" class. Hence, it is a regression algorithm but, by setting a rule on the probability, we can perform classification.

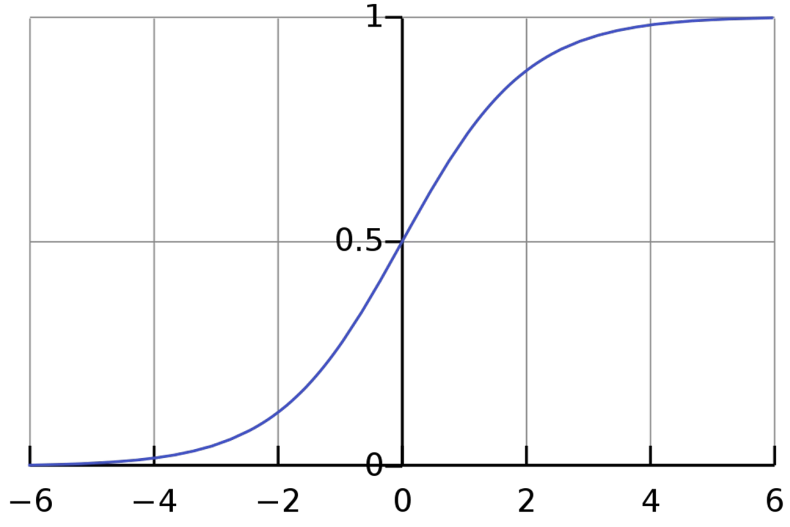


Using Linear Regression for Binary Classification

$$Turning\ scores\ of\ input\ \mathbf{x},\ \mathbf{w}^T\mathbf{x}\ into\ probabilities\ using\ sigmoid\ function$$

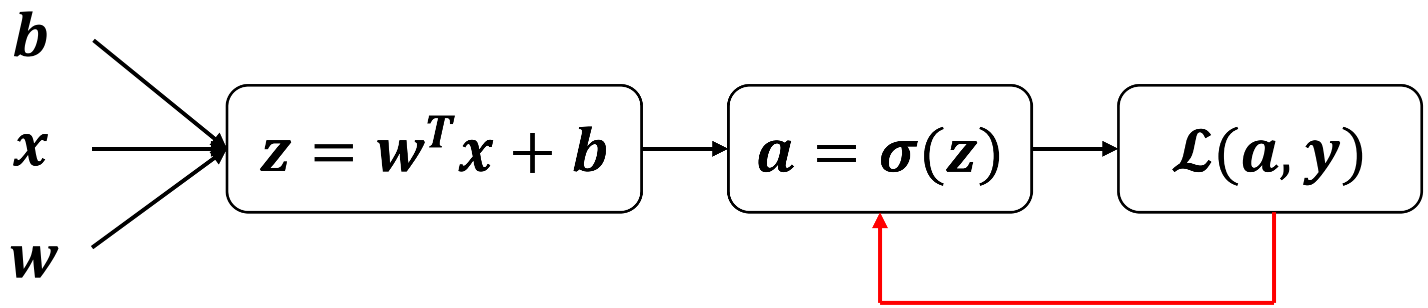
$$p\left(y=1|\mathbf{x},\mathbf{w}\right)=a=\sigma\left(\mathbf{w}^T\mathbf{x}\right)=\frac{e^{\mathbf{w}^T\mathbf{x}}}{1+e^{\mathbf{w}^T\mathbf{x}}}=\frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$$

$$p\left(y=0|\mathbf{x},\mathbf{w}\right)=1-a=1-\sigma\left(\mathbf{w}^T\mathbf{x}\right)=\frac{1}{1+e^{\mathbf{w}^T\mathbf{x}}}$$



Backward Propagation

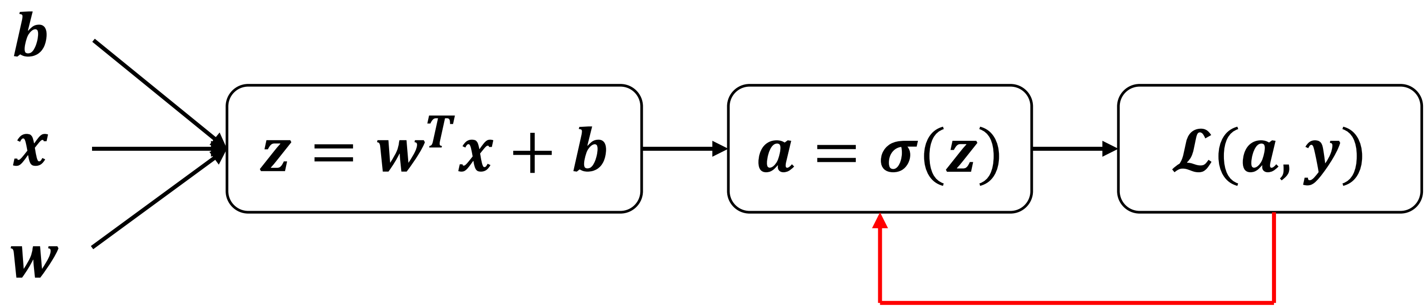
- The derivatives can be computed by moving from right to left



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

Backward Propagation

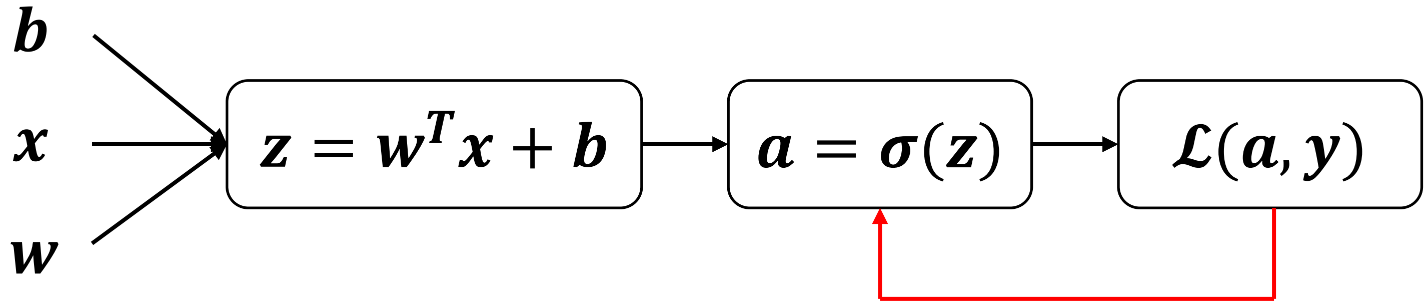
- The derivatives can be computed by moving from right to left



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

Backward Propagation

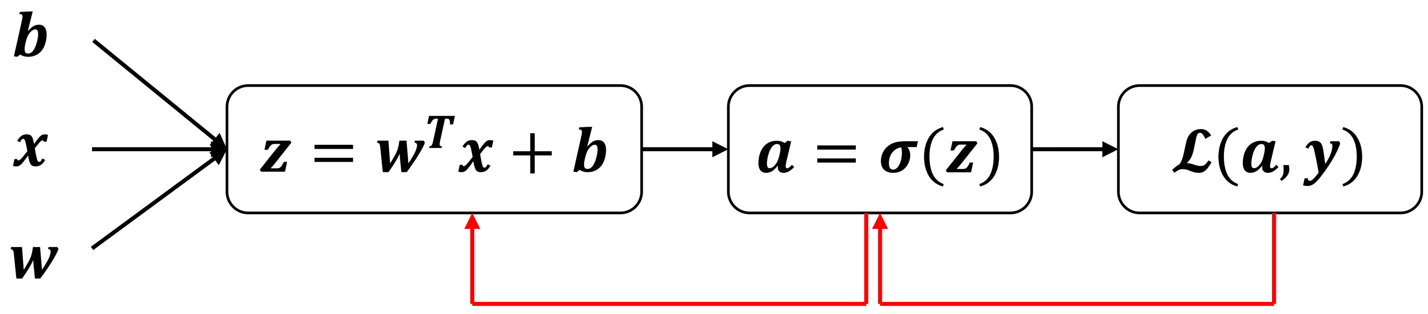
- The derivatives can be computed by moving from right to left



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

Backward Propagation

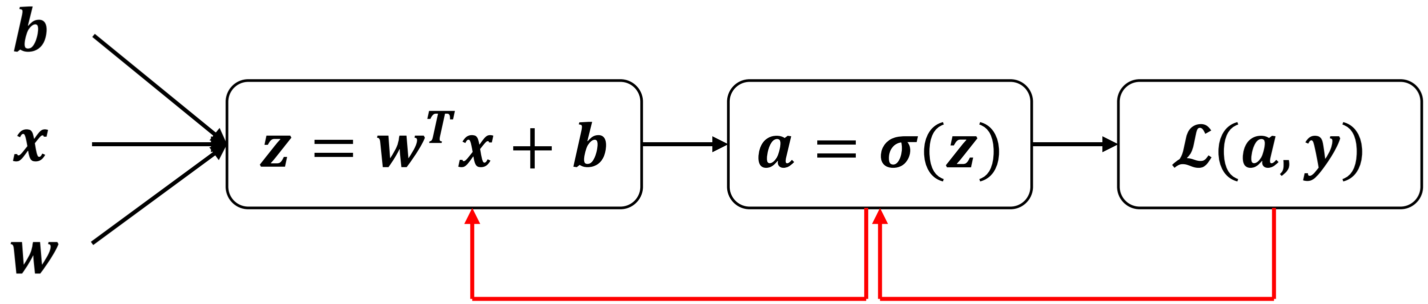
- The derivatives can be computed by moving from right to left



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

Backward Propagation

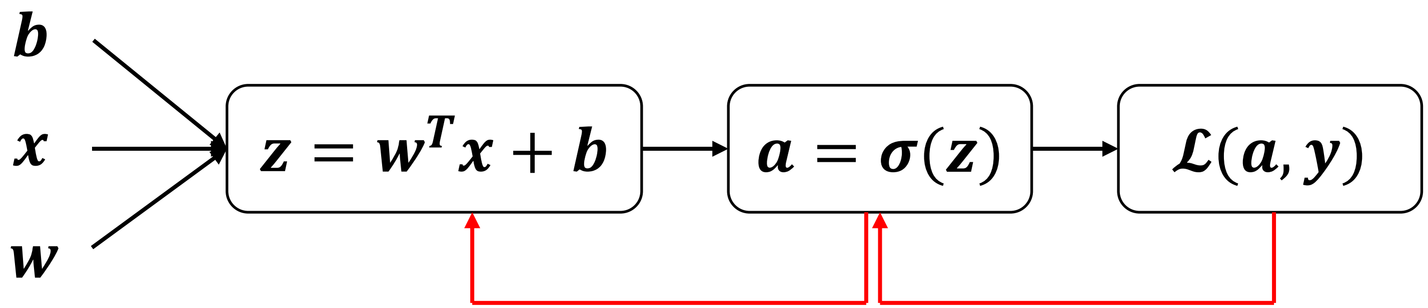
- The derivatives can be computed by moving from right to left



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

Backward Propagation

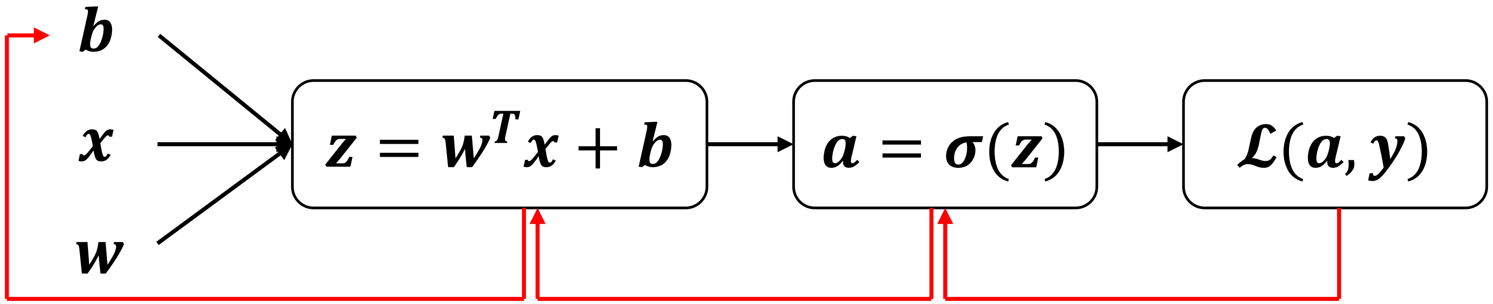
- The derivatives can be computed by moving from right to left



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

Backward Propagation

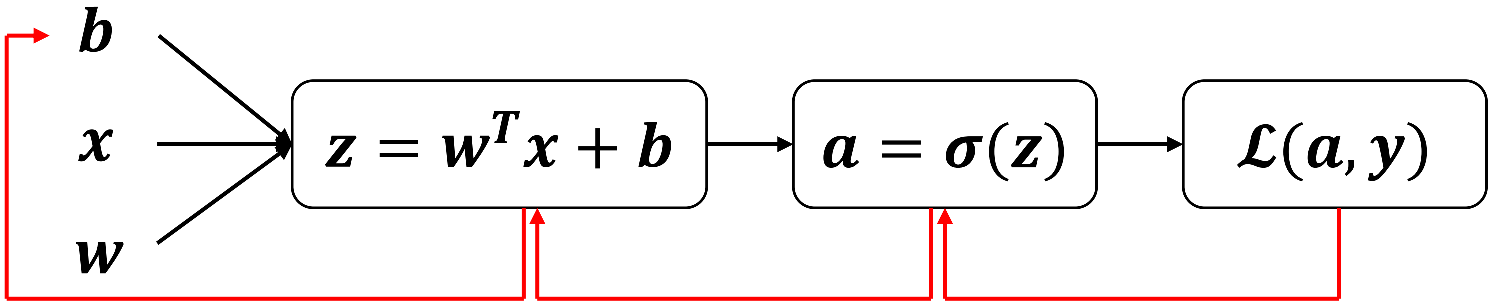
- The derivatives can be computed by moving from right to left



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

Backward Propagation

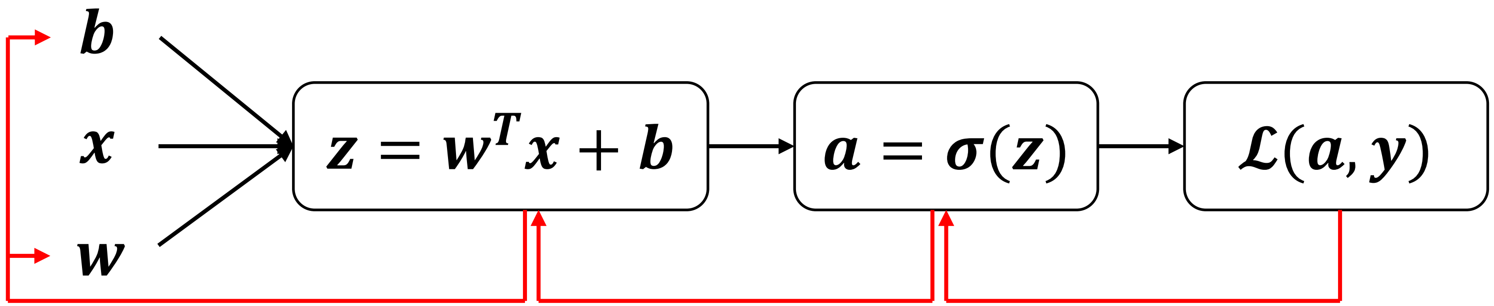
- The derivatives can be computed by moving from right to left



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

Backward Propagation

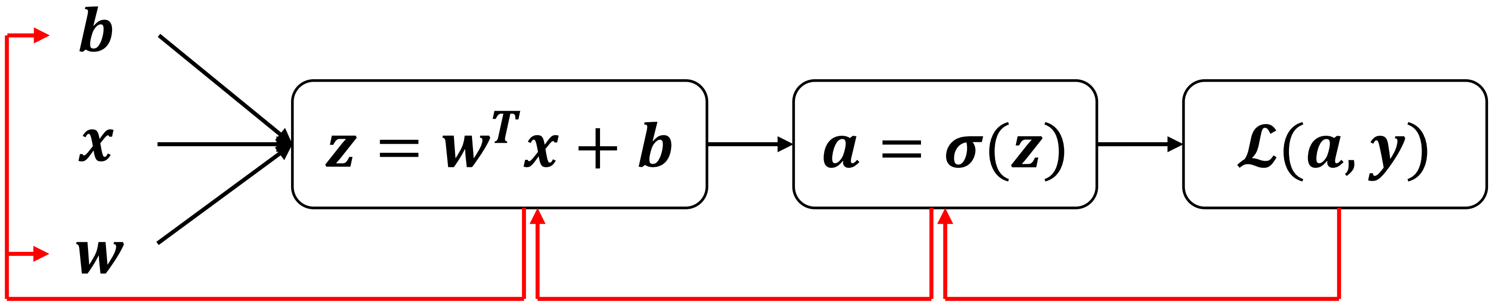
- The derivatives can be computed by moving from right to left



$\frac{\partial \mathcal{L}}{\partial w}$ = 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:

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

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

Backward Propagation: Summary

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

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

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

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

Gradient Descent For Logistic Regression

- **Outline**:
  - Have a loss function $\mathcal{L}(\boldsymbol{w}, \boldsymbol{b})$, 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{

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

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

}

Let us focus on this part

Gradient Descent For Logistic Regression

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

Assuming n examples

$$\boldsymbol{for\ i = 1\ to\ n:}$$

Forward propagation

$$z^{(i)} = w^T x^{(i)} + b$$
$$a^{(i)} = \sigma(z^{(i)})$$

Backward propagation

$$dz^{(i)} = a^{(i)} - y^{(i)}$$
$$dw = dw + dz^{(i)} x^{(i)}$$
$$db = db + dz^{(i)}$$

Outside the loop

$$dw = dw/n$$
$$db = db/n$$
$$w = w - \alpha dw$$
$$b = b - \alpha db$$

}

Vectorized version

$$Z = w^T X + b$$
$$A = \sigma(Z)$$
$$dZ = A - Y$$
$$dw = \frac{1}{n} X dZ^T$$
$$db = \frac{1}{n} \sum_{i=1}^{n} dz^{(i)}$$
$$w = w - \alpha dw$$
$$b = b - \alpha db$$

Gradient Descent For Logistic Regression

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

$$\boldsymbol{Z = w^T X + b}$$

$$\boldsymbol{A = \sigma(Z)}$$

$$\boldsymbol{dZ = A - Y}$$

$$\boldsymbol{dw = \frac{1}{n} X dZ^T}$$

$$\boldsymbol{db = \frac{1}{n} \sum_{i=1}^{n} dz^{(i)}}$$

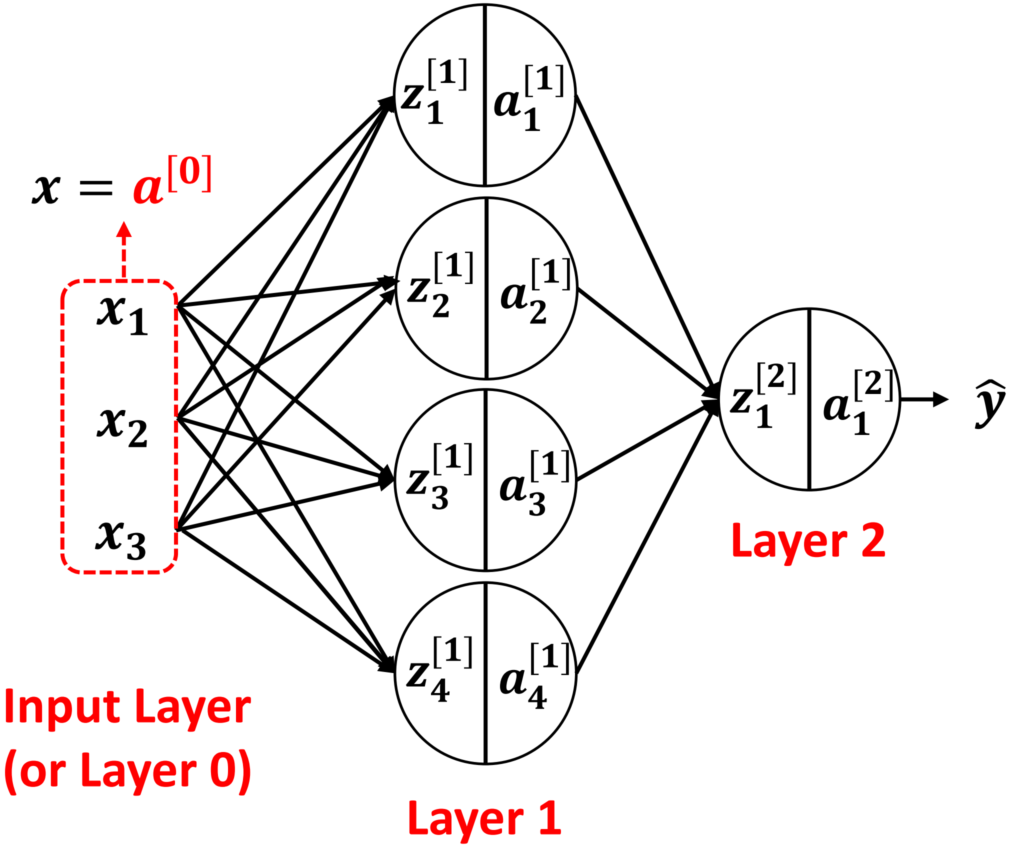
$$\boldsymbol{w = w - \alpha dw}$$
$$\boldsymbol{b = b - \alpha db}$$

}

The Computation Graph of A Neural Network

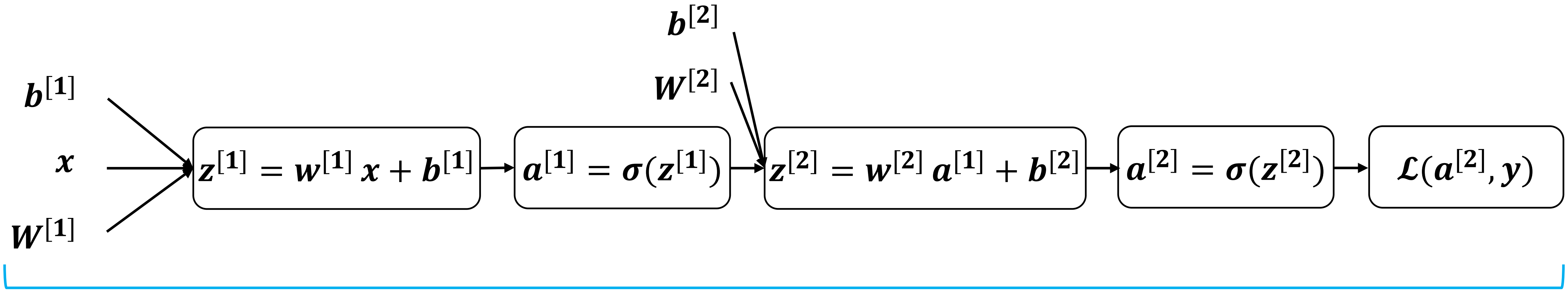
- Akin to logistic regression, we can represent any neural network in terms of a computation graph

A neural network with 2 layers



The Computation Graph of A Neural Network

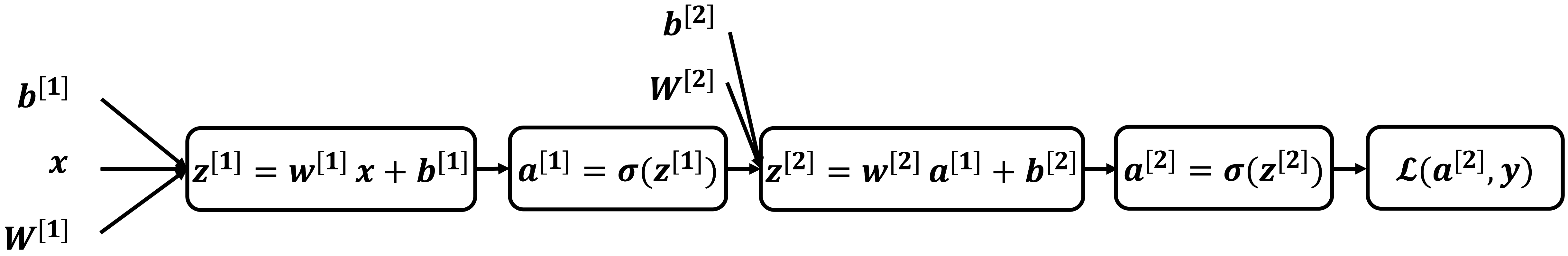
- Akin to logistic regression, we can represent any neural network in terms of a computation graph



The corresponding computation graph

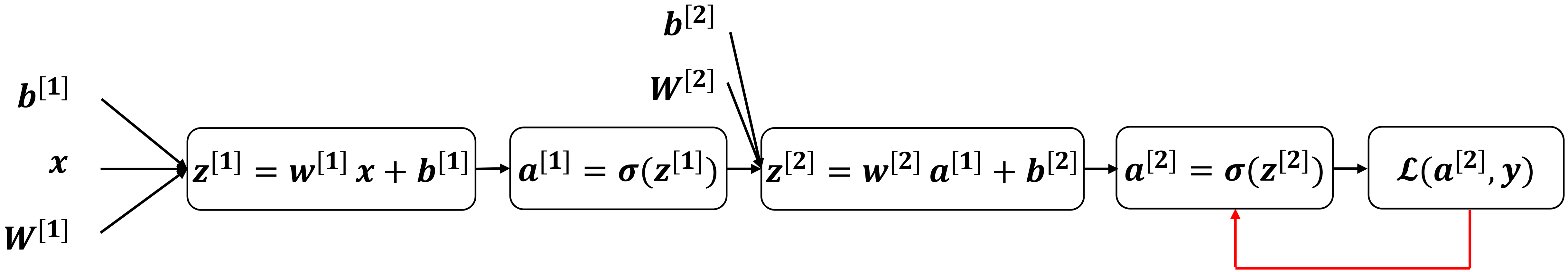
Forward Propagation

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



Backward Propagation

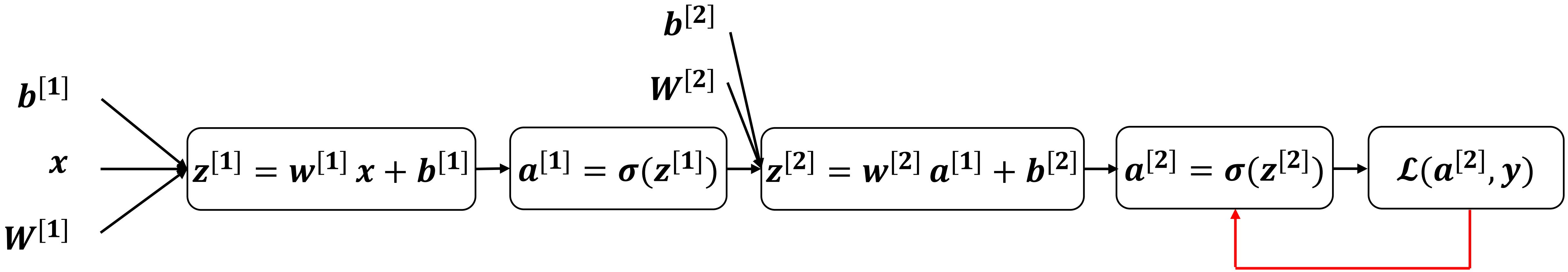
- The derivatives can be computed by moving from right to left



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

Backward Propagation

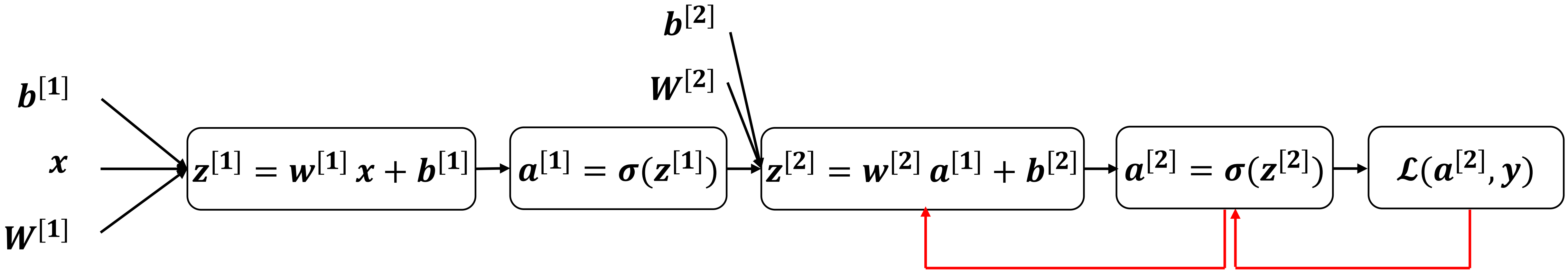
- The derivatives can be computed by moving from right to left



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

Backward Propagation

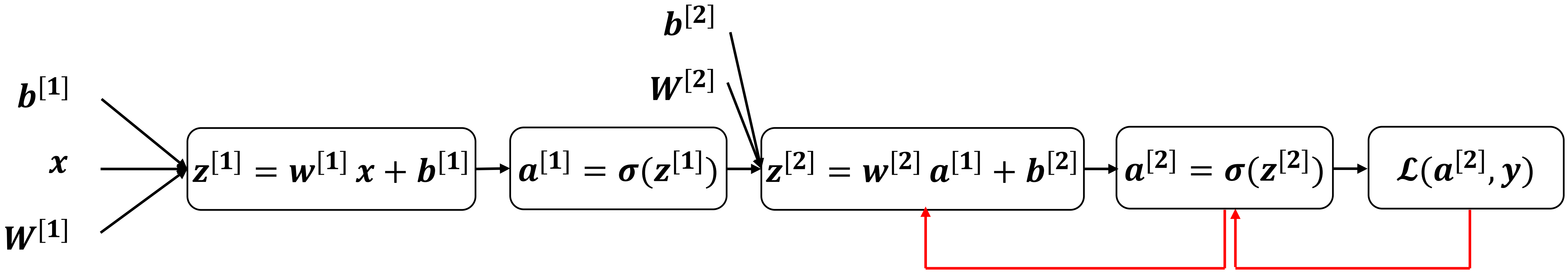
- The derivatives can be computed by moving from right to left



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

Backward Propagation

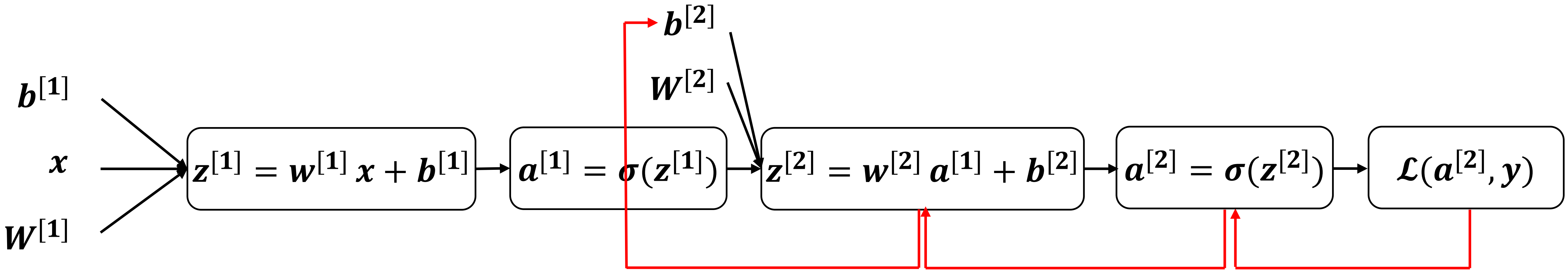
- The derivatives can be computed by moving from right to left



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

Backward Propagation

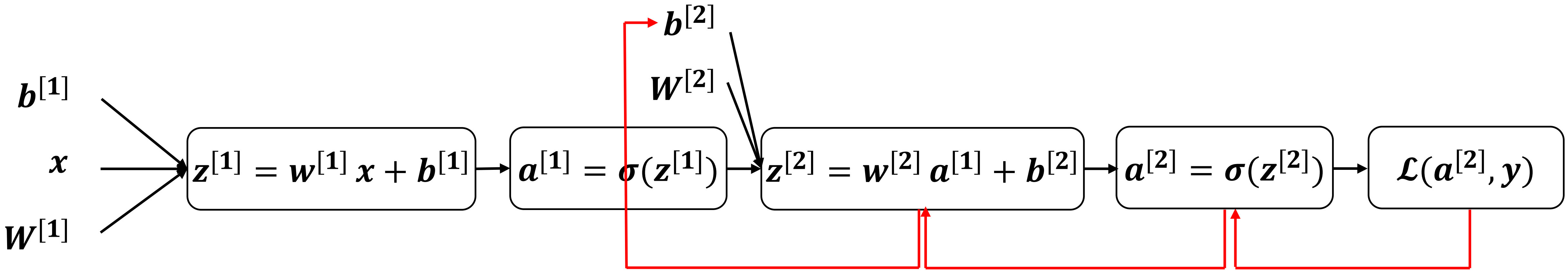
- The derivatives can be computed by moving from right to left



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

Backward Propagation

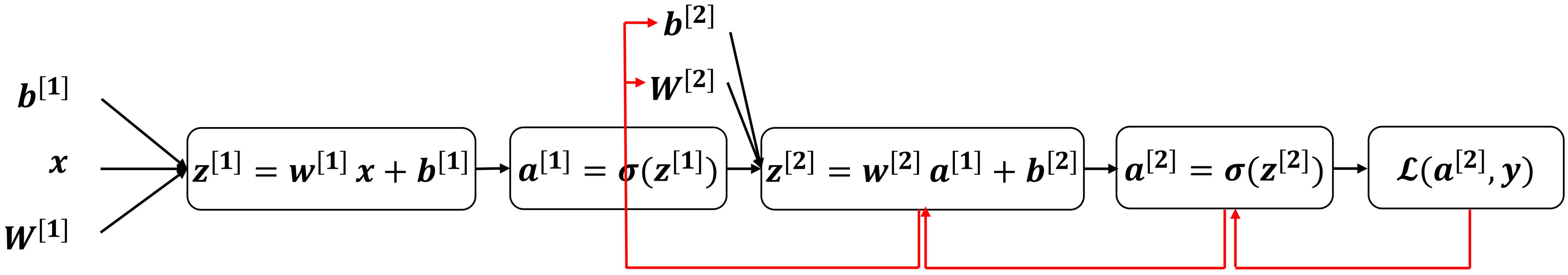
- The derivatives can be computed by moving from right to left



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

Backward Propagation

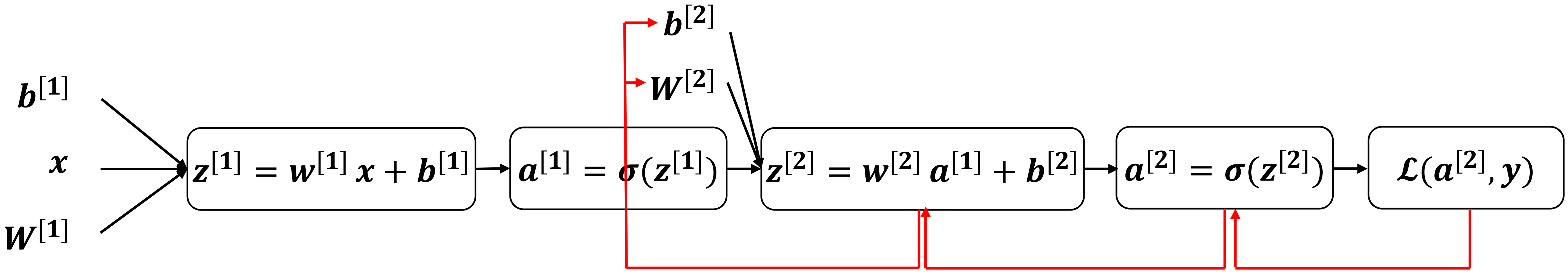
- The derivatives can be computed by moving from right to left



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

Backward Propagation

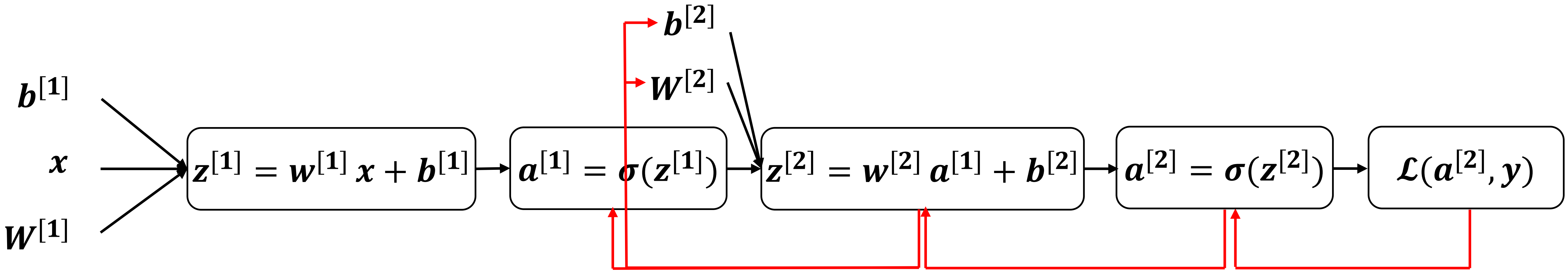
- The derivatives can be computed by moving from right to left



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

Backward Propagation

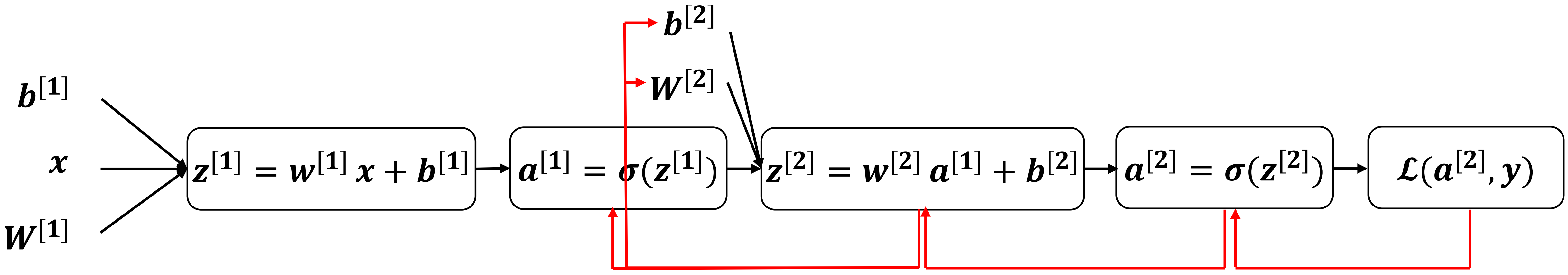
- The derivatives can be computed by moving from right to left



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

Backward Propagation

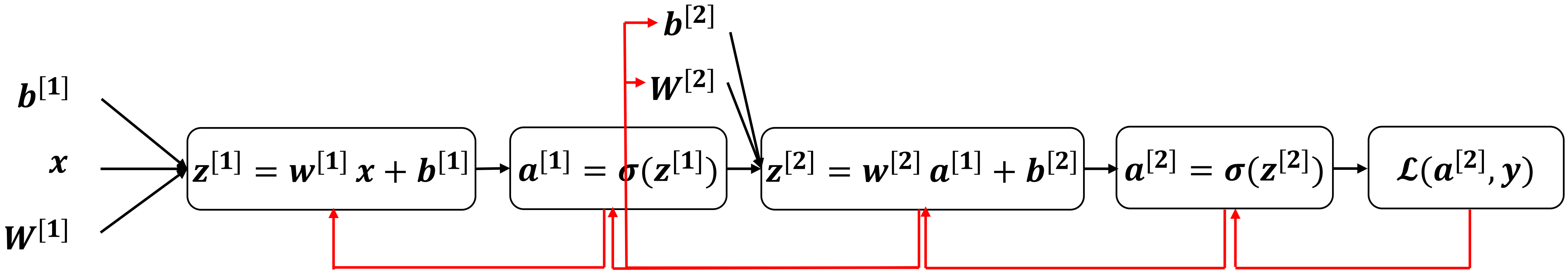
- The derivatives can be computed by moving from right to left



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

Backward Propagation

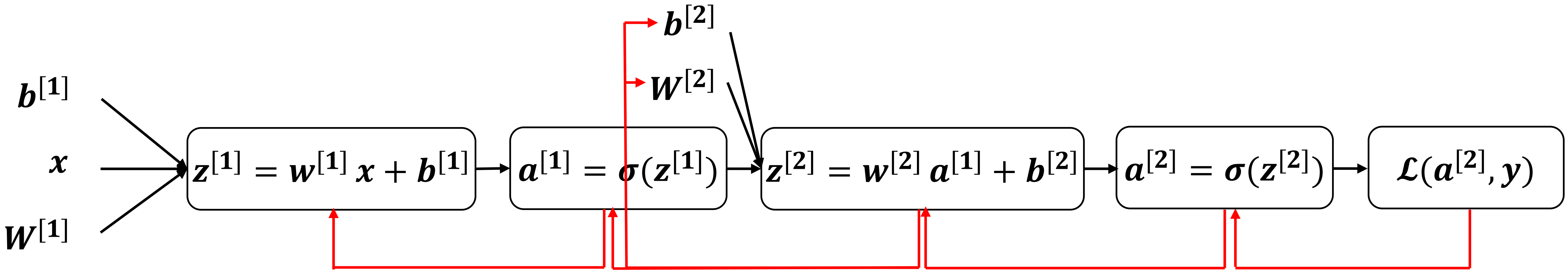
- The derivatives can be computed by moving from right to left



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

Backward Propagation

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

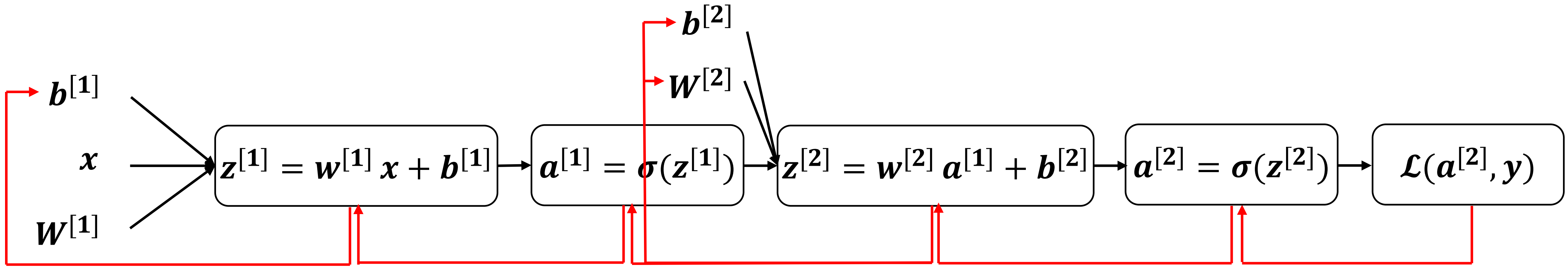


$$\frac{\partial \mathcal{L}}{\partial z^{[1]}} = \frac{\partial \mathcal{L}}{\partial a^{[2]}} \times \frac{\partial a^{[2]}}{\partial z^{[2]}} \times \frac{\partial z^{[2]}}{\partial a^{[1]}} \times \frac{\partial a^{[1]}}{\partial z^{[1]}} = (a^{[2]} - y) w^{[2]T} * a^{[1]}(1 - a^{[1]})$$

Element-wise product

Backward Propagation

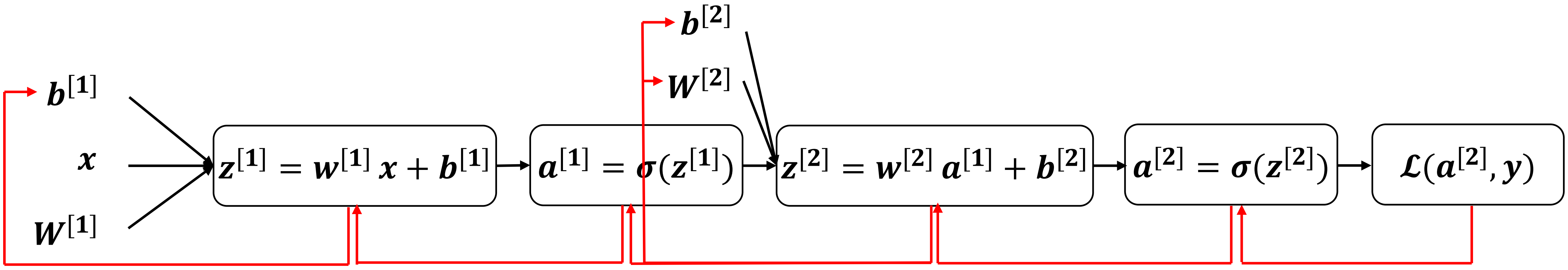
- The derivatives can be computed by moving from right to left



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

Backward Propagation

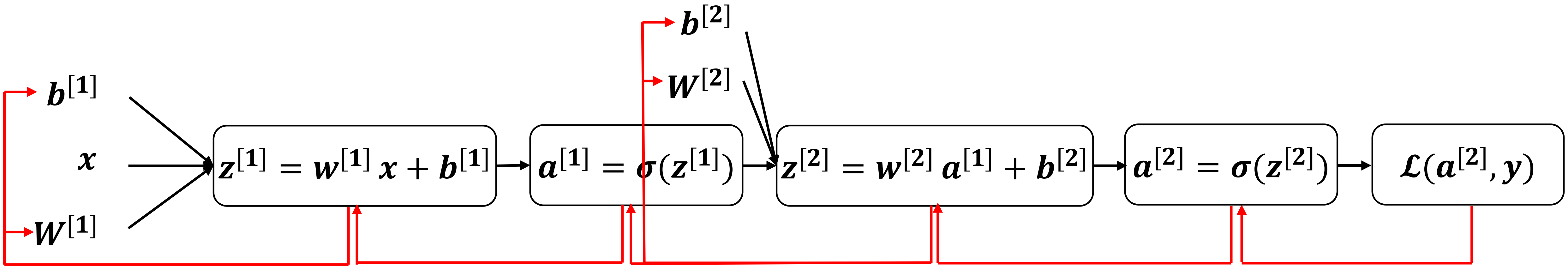
- The derivatives can be computed by moving from right to left



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

Backward Propagation

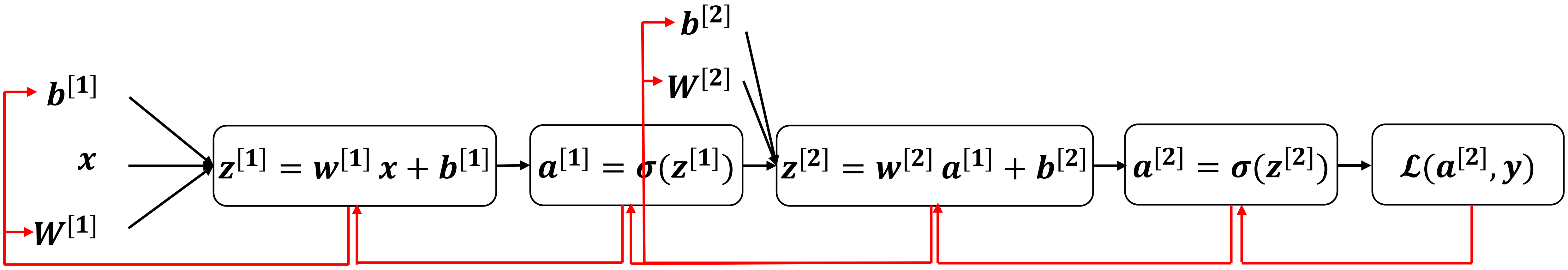
- The derivatives can be computed by moving from right to left



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

Backward Propagation

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



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

Backward Propagation: Summary

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

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

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

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

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

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

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