Introduction to Computer Vision

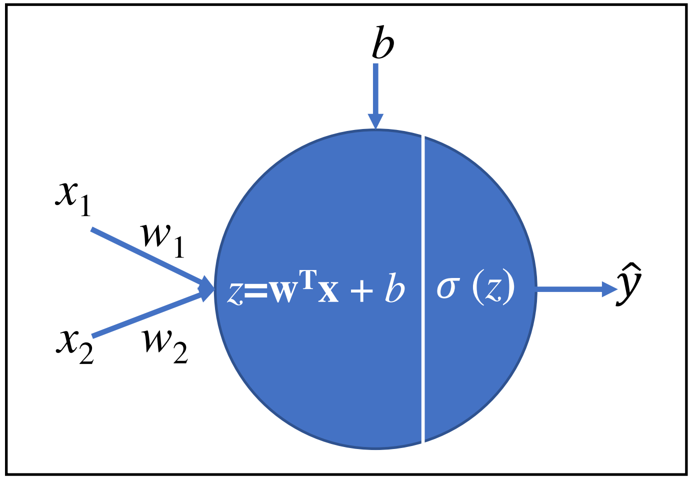
Introduction to Neural Networks: Part2

Dr. Sedat Ozer

Use of Python

- Both GPU and CPU has parallelization instructions and many Python built-in functions supports that.
- Use built-in functions for matrix (or vector) operations and avoid using for loops in your code whenever you can!

Cost function and GD implementation



Remember the loss for single sample:

$$\frac{\partial \mathcal{L}(a,y)}{\partial w_1} = x_1(a - y)$$

$$\frac{\partial \mathcal{L}(a,y)}{\partial w_2} = x_2(a - y)$$

$$\frac{\partial \mathcal{L}(a,y)}{\partial b} = a - y$$

Cost Function:

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

Final derivatives to be used:

$$\frac{\partial J(w,b)}{\partial w_1} = \frac{1}{m}\sum_{i=1}^{m} \frac{\partial \mathcal{L}(a^{(i)}, y^{(i)})}{\partial w_1}$$

$$\frac{\partial J(w,b)}{\partial w_2} = \frac{1}{m}\sum_{i=1}^{m} \frac{\partial \mathcal{L}(a^{(i)}, y^{(i)})}{\partial w_2}$$

$$\frac{\partial J(w,b)}{\partial b} = \frac{1}{m}\sum_{i=1}^{m} \frac{\partial \mathcal{L}(a^{(i)}, y^{(i)})}{\partial b}$$

Implement all that:

J =0; $\mathrm{d}w_1$=0; $\mathrm{d}w_2$=0; $\mathrm{d}b$ =0; $\propto = 0.00001$

For i=1 to m

$\quad z^{(i)} = \mathbf{w}^T\mathbf{x}^{(i)} + b$

$\quad a^{(i)} = \sigma(z^{(i)})$

$\quad J \mathrel{+}= -[y^{(i)}\log a^{(i)} + (1 - y^{(i)})\log(1 - a^{(i)})]$

$\quad \mathrm{d}z^{(i)} = a^{(i)} - y^{(i)}$

$\quad \mathrm{d}w_1 \mathrel{+}= x_1^{(i)}\,\mathrm{d}z^{(i)}$

$\quad \mathrm{d}w_2 \mathrel{+}= x_2^{(i)}\,\mathrm{d}z^{(i)}$

$\quad \mathrm{d}b \mathrel{+}= \mathrm{d}z^{(i)}$

$J = J/\mathrm{m}$; $\mathrm{d}w_1 = \mathrm{d}w_1/\mathrm{m}$;

$\mathrm{d}w_2 = \mathrm{d}w_2/\mathrm{m}$; $\mathrm{d}b = \mathrm{d}b/\mathrm{m}$

$w_1 := w_1 - \propto \mathrm{d}w_1$

$w_2 := w_2 - \propto \mathrm{d}w_2$

$b := \mathrm{b} - \propto \mathrm{db}$

Lets Re-implement the Logistic Regression

Remember: $\mathbf{X}_{nxm} = \begin{array}{c} \text{samples} \rightarrow \\ \begin{bmatrix} \mathbf{x}^1 \end{bmatrix} \begin{bmatrix} \cdots \end{bmatrix} \begin{bmatrix} \mathbf{x}^m \end{bmatrix} \end{array} \downarrow \text{features}$ $\quad \mathbf{Y} = \left\{ y^1, y^2, \ldots, y^m \right\}$

(Training) Data

$$\mathbf{Z} = \begin{bmatrix} \mathbf{z}^1 \end{bmatrix} \begin{bmatrix} \cdots \end{bmatrix} \begin{bmatrix} \mathbf{z}^m \end{bmatrix} = \mathbf{w}^T \mathbf{X} + [b, b, \ldots, b]_{1xm} \Rightarrow \mathbf{A} = \begin{bmatrix} \mathbf{a}^1 \end{bmatrix} \begin{bmatrix} \cdots \end{bmatrix} \begin{bmatrix} \mathbf{a}^m \end{bmatrix} = \boldsymbol{\sigma}(\mathbf{Z})$$

Implementation for **Z**: $\mathbf{Z} = \text{np.dot}(\text{w.T}, \text{X}) + b$

Note that in this particular example, the output is a vector. Therefore, the matrix Z (thus, the matrix A) becomes a vector.

A Better Implementation in Python

One iteration of gradient descent

J =0; $\mathrm{d}w_1$=0; $\mathrm{d}w_2$=0; $\mathrm{d}b$ =0
For i=1 to m
$\quad z^{(i)} = \mathbf{w}^T\mathbf{x}^{(i)} + b$
$\quad a^{(i)} = \sigma(z^{(i)})$
$\quad J \mathrel{+}= -[y^{(i)}\log a^{(i)} + (1-y^{(i)})\log(1-a^{(i)})]$
$\quad \mathrm{d}z^{(i)} = a^{(i)} - y^{(i)}$
$\quad \mathrm{d}w_1 \mathrel{+}= x_1^{(i)}\, \mathrm{d}z^{(i)}$
$\quad \mathrm{d}w_2 \mathrel{+}= x_2^{(i)}\, \mathrm{d}z^{(i)}$
$\quad \mathrm{d}b \mathrel{+}= \mathrm{d}z^{(i)}$
$J = J/\mathrm{m}$; $\mathrm{d}w_1 = \mathrm{d}w_1/\mathrm{m}$;
$\mathrm{d}w_2 = \mathrm{d}w_2/\mathrm{m}$; $\mathrm{d}b = \mathrm{d}b/\mathrm{m}$
$w_1 := w_1 - \propto \mathrm{d}w_1$
$w_2 := w_2 - \propto \mathrm{d}w_2$
$b := \mathrm{b} - \propto \mathrm{db}$

One iteration of gradient descent

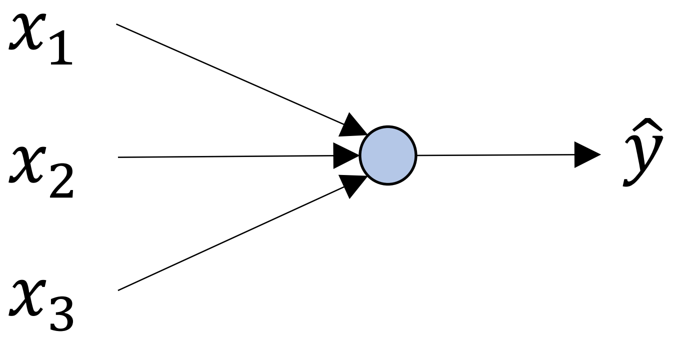
$\mathbf{Z} = \mathbf{w}^T \boldsymbol{X} + b$
$\mathbf{A} = \sigma(\mathbf{Z})$
$\mathrm{d}\mathbf{Z} = \mathbf{A} - \mathbf{Y}$
$\mathrm{d}\,\mathbf{w} = (1/\mathrm{m})\mathbf{X}\,\mathrm{d}\mathbf{Z}^{\mathbf{T}}$
$\mathrm{d}\mathbf{b} = (1/\mathrm{m})\mathrm{np.sum}(\mathrm{d}\mathbf{Z})$
$\mathbf{w} := \mathbf{w} - \propto \mathrm{d}\mathbf{w}$
$\mathbf{b} := \mathbf{b} - \propto \mathrm{d}\mathbf{b}$

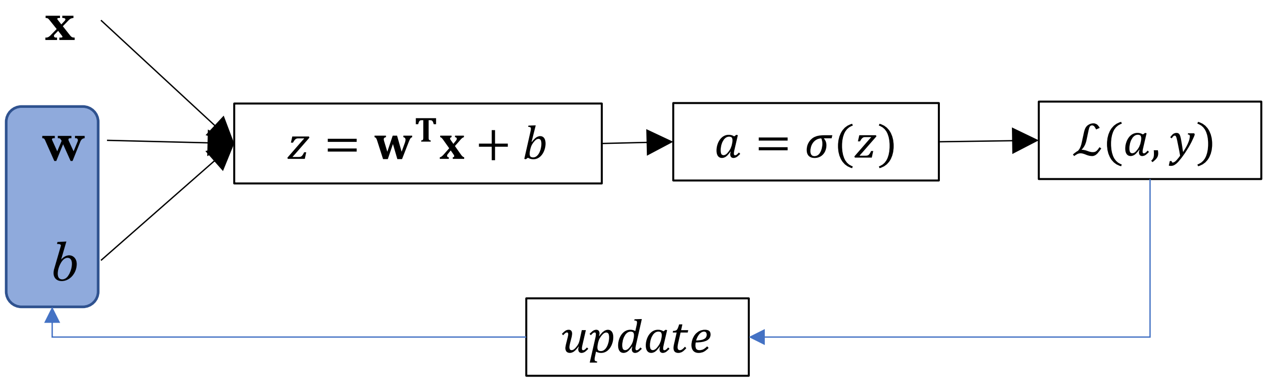
Neural “**Networks**”

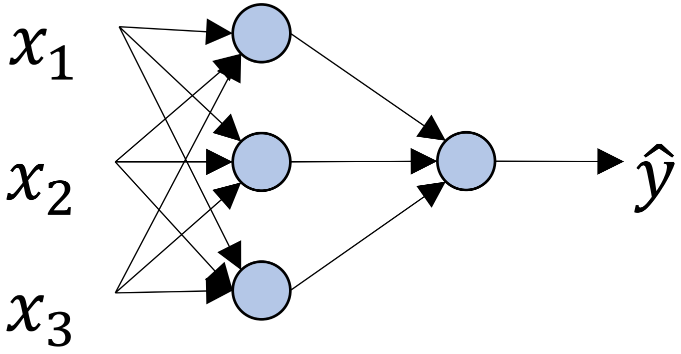
Neural Networks

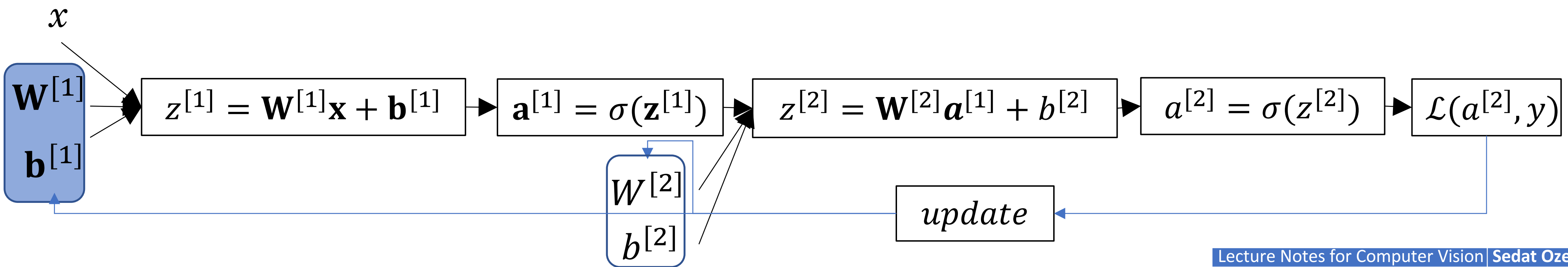
- So far, we have seen only a “single neuron” (with two models)!
  - Linear model
  - Logistic regression model
- The performance of a single unit (single neuron) is limited!
- Higher performance can be achieved by forming a network of multiple neurons.

A Neuron vs. A Neural Network



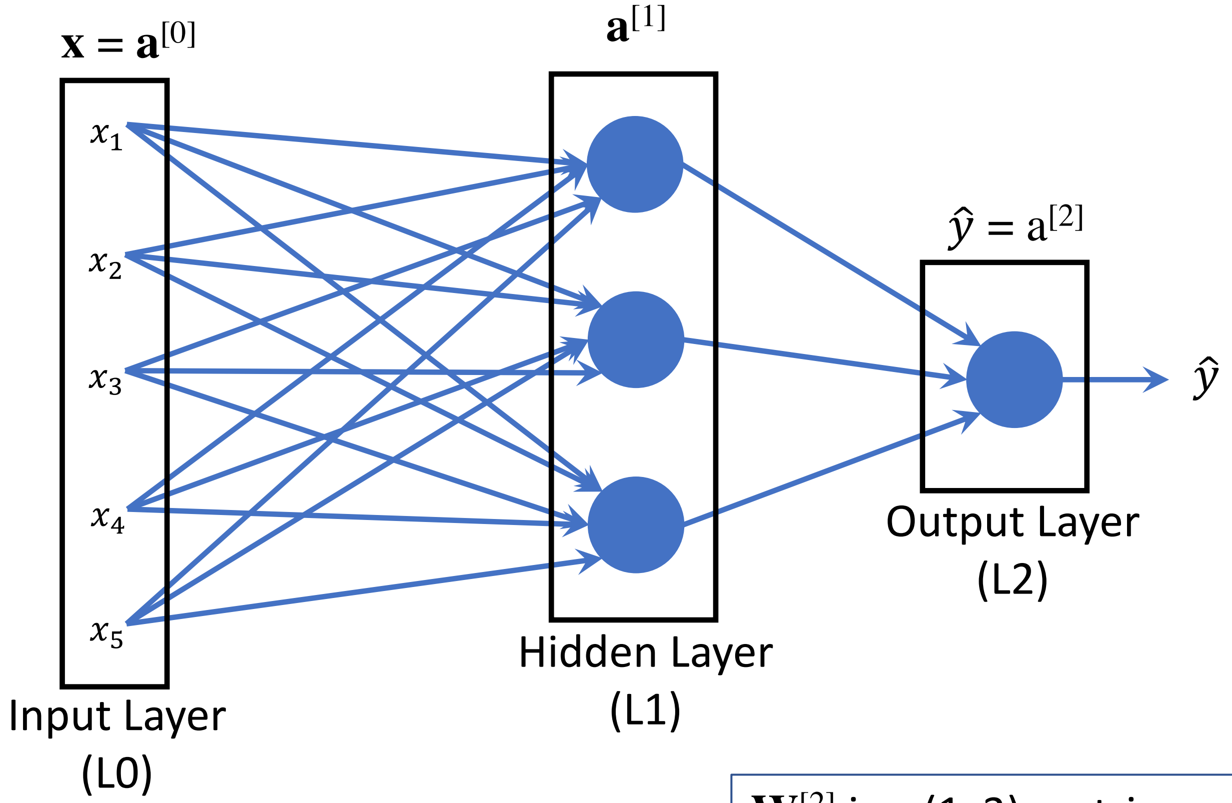






A 2-layer FC-NN Example:

Each layer has its own weights and bias values. So… the k[th] layer would have: $\mathbf{W}^{[k]}$ and $\mathbf{b}^{[k]}$



$$\mathbf{a}^{[1]} = \begin{pmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \end{pmatrix}_{3\text{x}1}$$

$$\hat{y} = \mathrm{a}^{[2]} = \begin{pmatrix} a_1^{[2]} \end{pmatrix}_{1\text{x}1} = \begin{pmatrix} \hat{y}_1 \end{pmatrix}_{1\text{x}1}$$

The Weight **Matrix** for the entire layer 1:

$$\mathbf{W}^{[1]} = \begin{array}{cl} \begin{pmatrix} w_1^1 & w_2^1 & w_3^1 & w_4^1 & w_5^1 \\ w_1^2 & w_2^2 & w_3^2 & w_4^2 & w_5^2 \\ w_1^3 & w_2^3 & w_3^3 & w_4^3 & w_5^3 \end{pmatrix}^{[1]}_{3\text{x}5} & \begin{array}{l} \text{1}^{\text{st}} \text{ unit in L1 } (\mathbf{w}_1^{[1]T}) \\ \text{2}^{\text{nd}} \text{ unit in L1 } (\mathbf{w}_2^{[1]T}) \\ \text{3}^{\text{rd}} \text{ unit in L1 } (\mathbf{w}_3^{[1]T}) \end{array} \end{array}$$

Layer

$$\mathbf{w}_1^{[1]} = \begin{pmatrix} w_1^1 \\ w_2^1 \\ w_3^1 \\ w_4^1 \\ w_5^1 \end{pmatrix}_{5\text{x}1}$$

a **vector**

Unit

(Reads: the weight vector of the 1st unit at the first layer)

$\mathbf{W}^{[1]}$ is a (3x5) matrix,
$\mathbf{b}^{[1]}$ is a (3x1) vector

$\mathbf{W}^{[2]}$ is a (1x3) matrix
(i.e., a row vector),
$\mathbf{b}^{[2]}$ is a (1x1) vector

What if I have more than 2 classes?

- In logistic regression we assumed that we had only two classes: Class 0 & Class 1.
- What if I have **more than two classes**?
- One typical approach is: using **one vs. all approach**.
  - (where, for example, you first consider Class 0 as one class and the combination of all the other classes as the “other” class. Then you consider Class 1 as one class and the combination of all the other classes as the “other class”,...)
- Another approach might be using **Softmax Regression** instead of logistic regression.
  - Define a new cost function and derive all the weight update rules according to that cost function.

Softmax Regression

- Remember Logistic Regression: We had only two classes (Class 0 and Class 1, i.e., $y^{(i)} \in \{0,1\}$).
- Softmax Regression is the case where we have *K* classes ($K > 2$) such that: $y^{(i)} \in \{1, \ldots, K\}$.
- Sigmoid function is no longer being used.
- Now the output can take *K* different values rather than just two.

$$a_i = \hat{y}_i = g(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Softmax Activation Function

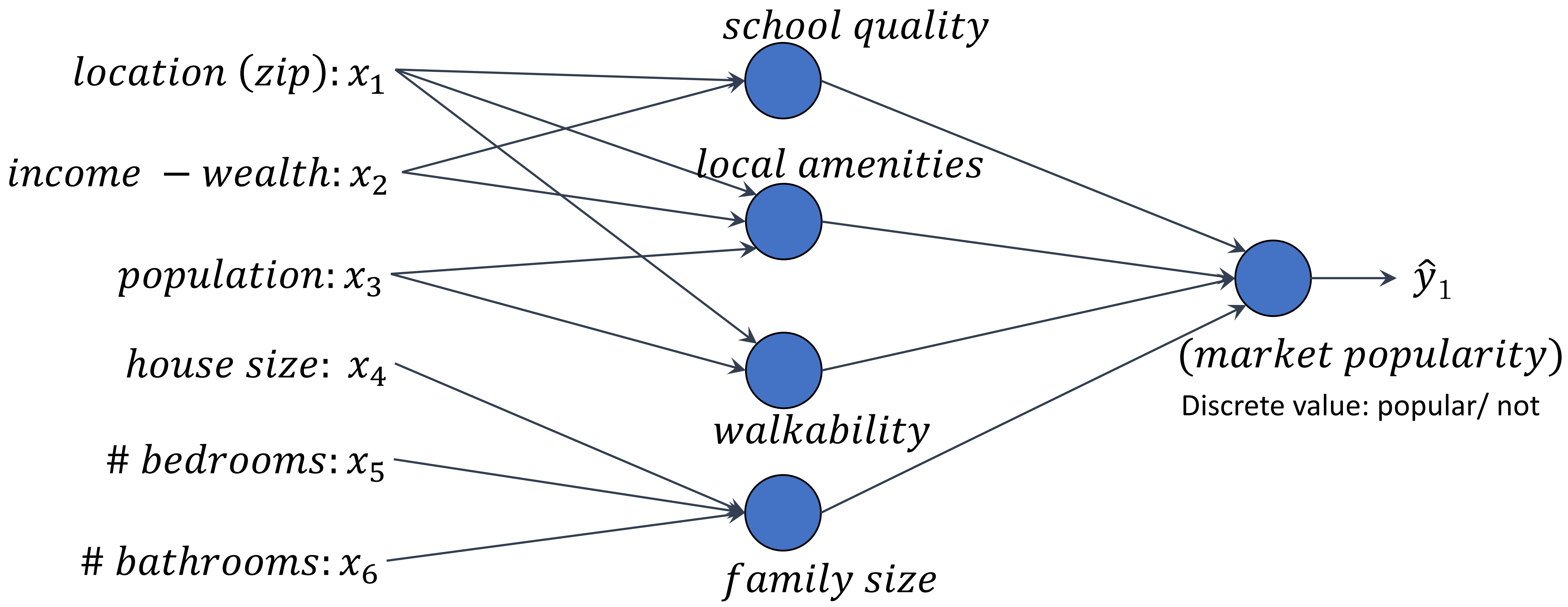
$$\mathcal{L}(\mathrm{a}, y) = -\sum_{j=1}^{K} y_j log(a_j)$$

Softmax Loss Function

$$J(w, b) = \frac{1}{m} \sum_{j=1}^{m} \mathcal{L}(\mathrm{a}^{(j)}, y^{(j)})$$

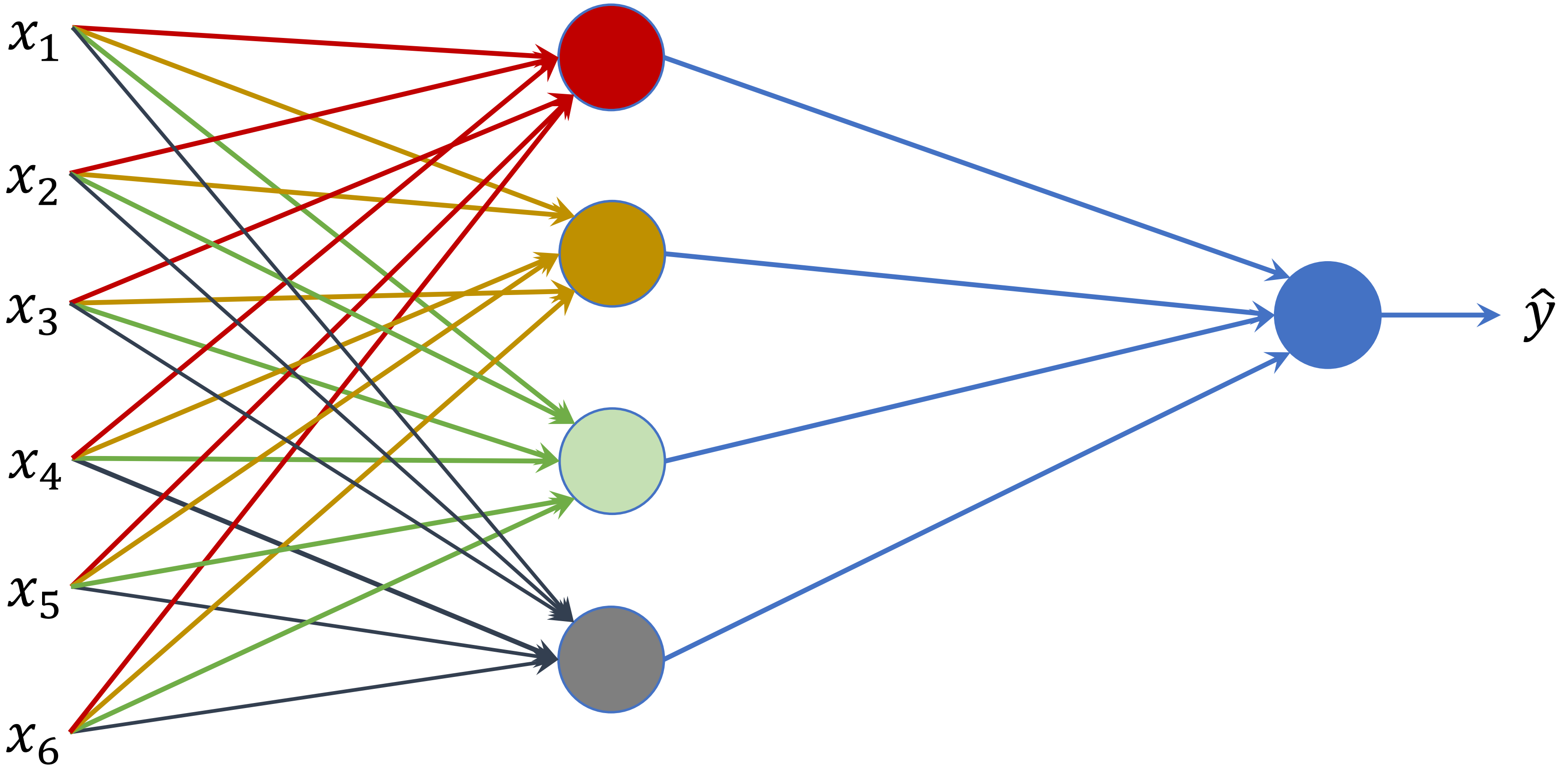
Softmax Cost Function

Lets have a look at an example:



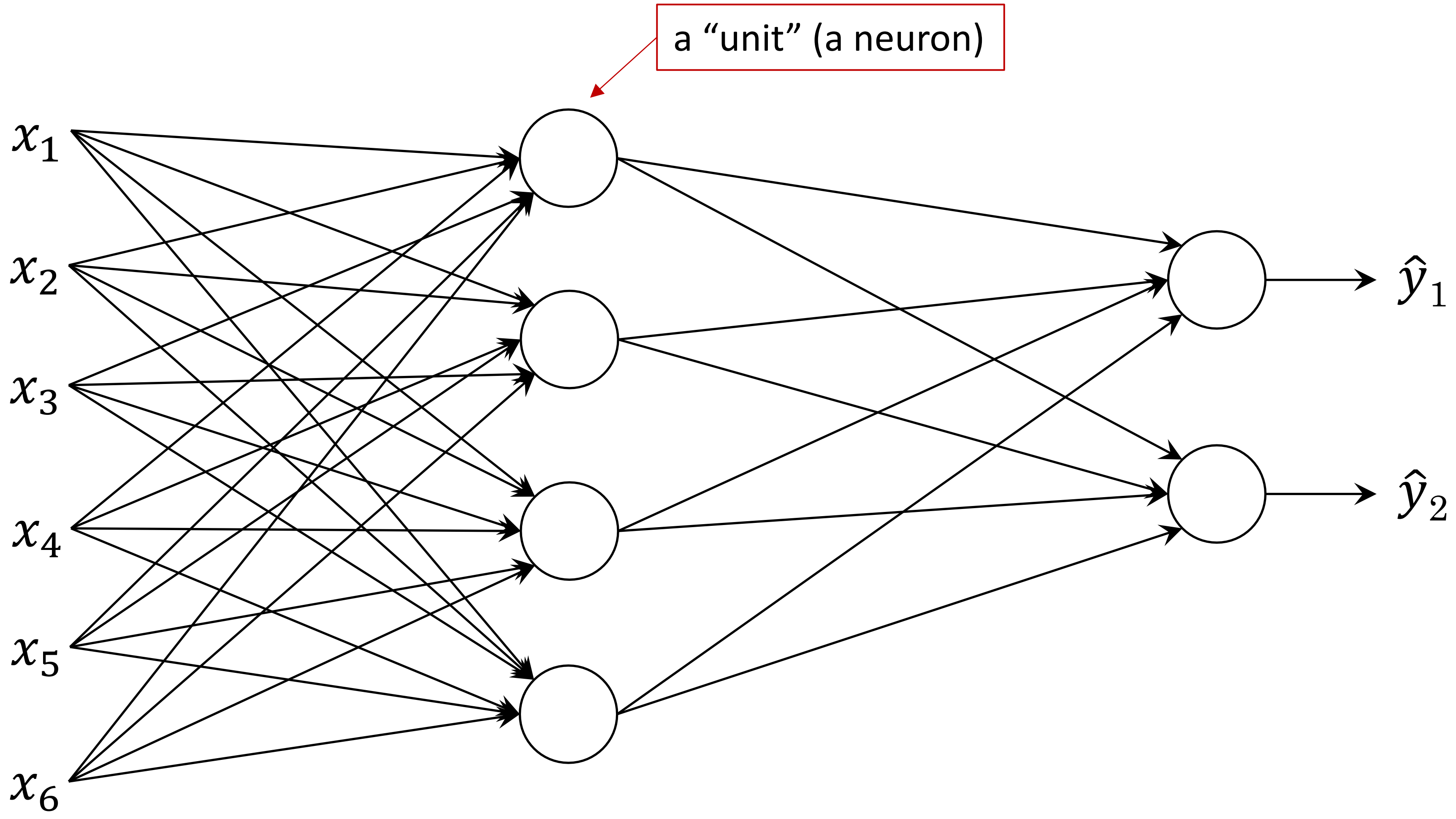
n: total number of features = 6

A Fully Connected (FC) Neural Network (6 inputs, 1 output)



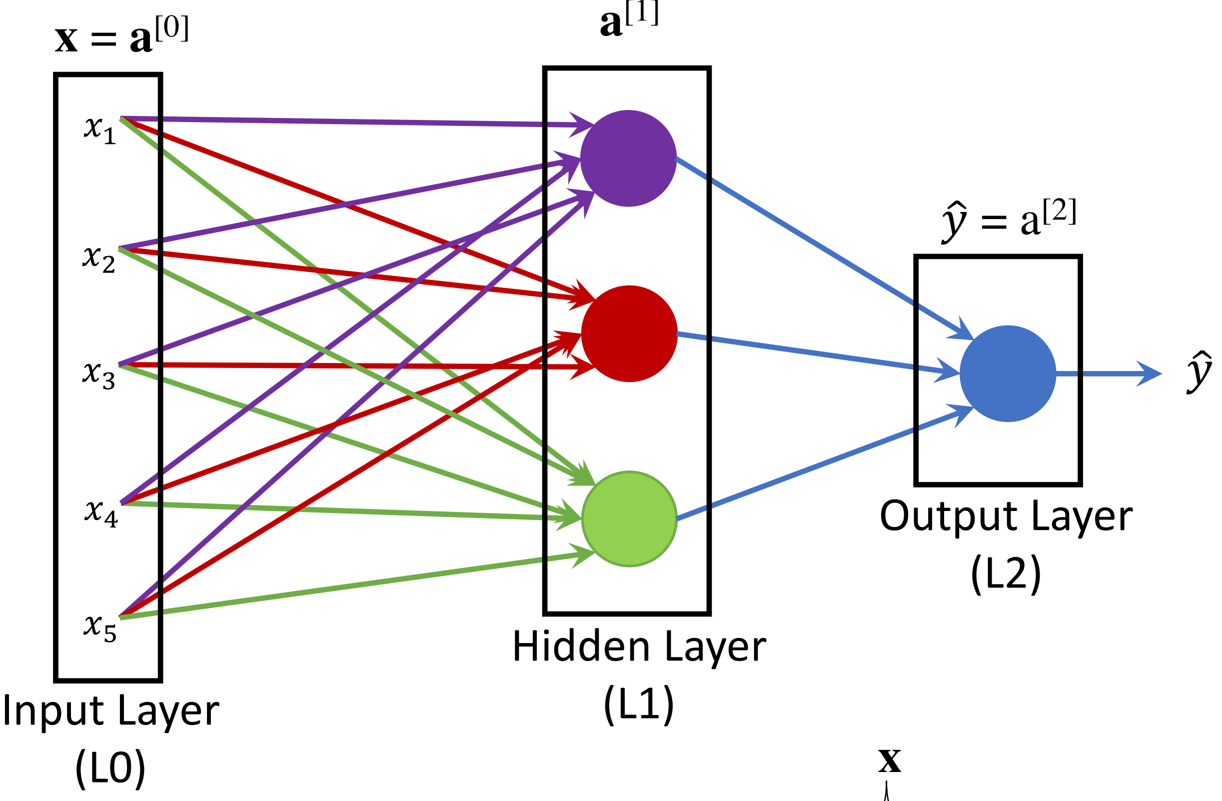
n: total number of features = 6

A Fully Connected (FC) Neural Network (6 inputs, 2 outputs)



n: total number of features = 6

A 2-layer FC-NN Example:



$$\hat{y} = a^{[2]} = \begin{pmatrix} a_1^{[2]} \end{pmatrix}_{1x1} = \begin{pmatrix} \hat{y}_1 \end{pmatrix}_{1x1}$$

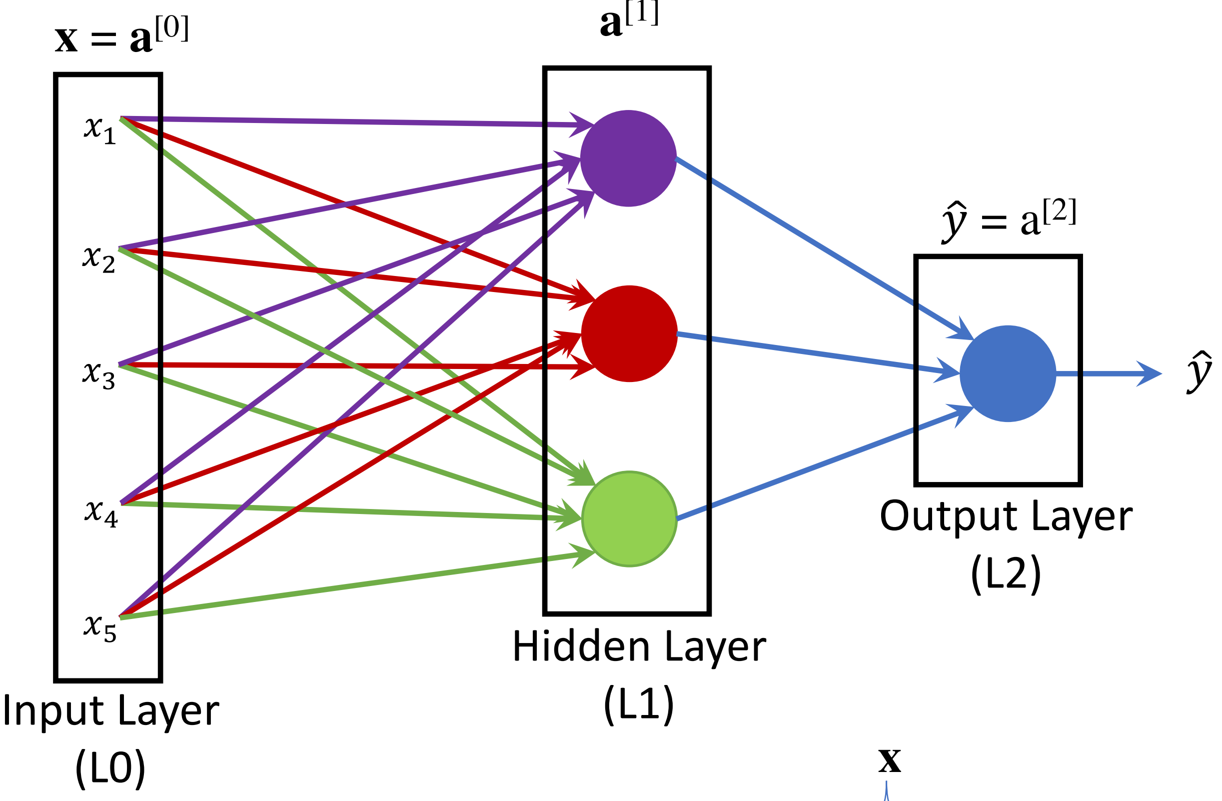
The Weight **Matrix** for the entire layer 1:

$$\mathbf{W}^{[1]} = \begin{pmatrix} w_1^1 & w_2^1 & w_3^1 & w_4^1 & w_5^1 \\ w_1^2 & w_2^2 & w_3^2 & w_4^2 & w_5^2 \\ w_1^3 & w_2^3 & w_3^3 & w_4^3 & w_5^3 \end{pmatrix}_{3x5}$$

$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]}\,\mathbf{x} + \mathbf{b}^{[1]} = \begin{pmatrix} w_1^1 & w_2^1 & w_3^1 & w_4^1 & w_5^1 \\ w_1^2 & w_2^2 & w_3^2 & w_4^2 & w_5^2 \\ w_1^3 & w_2^3 & w_3^3 & w_4^3 & w_5^3 \end{pmatrix}_{3x5} \overbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix}_{5x1}}^{\mathbf{x}} + \begin{pmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \end{pmatrix}_{3x1} = \begin{pmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \end{pmatrix}_{3x1}$$

$$\Rightarrow \quad \mathbf{a}^{[1]} = \begin{pmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \end{pmatrix}_{3x1} = \begin{pmatrix} \sigma(z_1^{[1]}) \\ \sigma(z_2^{[1]}) \\ \sigma(z_3^{[1]}) \end{pmatrix}_{3x1} = \sigma(\mathbf{z}^{[1]})$$

A 2-layer FC-NN Example:



Steps to compute the output for logistic regression for **one input sample**:

$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]}\,\mathbf{x} + \mathbf{b}^{[1]}$$

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

$$\mathbf{z}^{[2]} = \mathbf{W}^{[2]}\,\mathbf{a}^{[1]} + \mathbf{b}^{[2]}$$

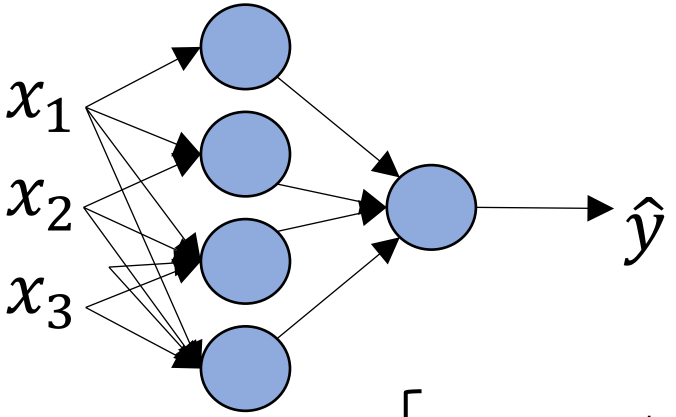
$$\mathbf{a}^{[2]} = \sigma(\mathbf{z}^{[2]})$$

$$\hat{y} = \mathbf{a}^{[2]} = \underset{1\text{x}1}{\begin{pmatrix} a_1^{[2]} \end{pmatrix}} = \underset{1\text{x}1}{\begin{pmatrix} \hat{y}_1 \end{pmatrix}}$$

$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]}\,\mathbf{x} + \mathbf{b}^{[1]} = \underset{3\text{x}5}{\begin{pmatrix} w_1^1 & w_2^1 & w_3^1 & w_4^1 & w_5^1 \\ w_1^2 & w_2^2 & w_3^2 & w_4^2 & w_5^2 \\ w_1^3 & w_2^3 & w_3^3 & w_4^3 & w_5^3 \end{pmatrix}} \overbrace{\underset{5\text{x}1}{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix}}}^{\mathbf{x}} + \underset{3\text{x}1}{\begin{pmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \end{pmatrix}} = \underset{3\text{x}1}{\begin{pmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \end{pmatrix}}$$

$$\Rightarrow \quad \mathbf{a}^{[1]} = \underset{3\text{x}1}{\begin{pmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \end{pmatrix}} = \underset{3\text{x}1}{\begin{pmatrix} \sigma(z_1^{[1]}) \\ \sigma(z_2^{[1]}) \\ \sigma(z_3^{[1]}) \end{pmatrix}} = \sigma(\mathbf{z}^{[1]})$$

Computation with less for-loop



$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

$$\mathrm{A}^{[1]} = \begin{bmatrix} | & | & & | \\ a^{[1](1)} & a^{[1](2)} & \dots & a^{[1](m)} \\ | & | & & | \end{bmatrix}$$

($Z^{[1]}$ has the same shape as $\mathrm{A}^{[1]}$)

Algorithm 1:

```
for i = 1 to m
```

$$\mathbf{z}^{[1](i)} = W^{[1]}\mathbf{x}^{(i)} + \mathbf{b}^{[1]}$$

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

$$z^{[2](i)} = W^{[2]}\mathbf{a}^{[1](i)} + b^{[2]}$$

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

Algorithm 2:

$$\mathbf{Z}^{[1]} = \mathbf{W}^{[1]}\mathbf{X} + \mathbf{b}^{[1]}$$

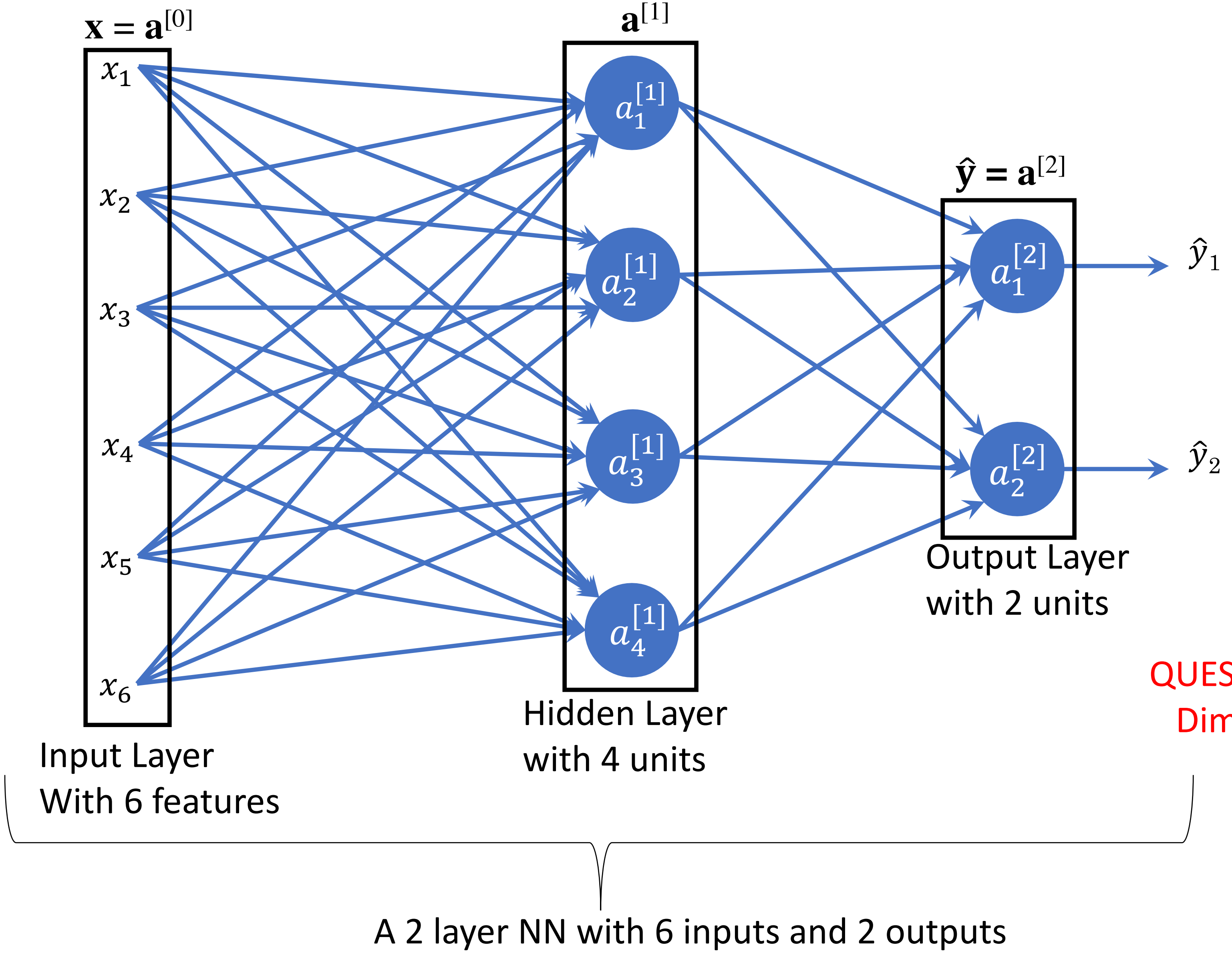
$$\mathbf{A}^{[1]} = \sigma(\mathbf{Z}^{[1]})$$

$$\mathbf{Z}^{[2]} = \mathbf{W}^{[2]}\mathbf{A}^{[1]} + \mathbf{b}^{[2]}$$

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

m = number of training samples

Another 2 layer FC NN Example



$$\mathbf{a}^{[1]} = \begin{pmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{pmatrix}_{4x1}$$

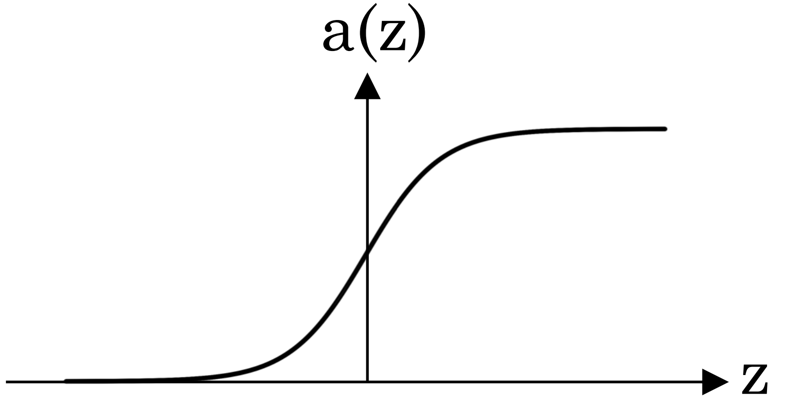
$$\hat{\mathbf{y}} = \mathbf{a}^{[2]} = \begin{pmatrix} a_1^{[2]} \\ a_2^{[2]} \end{pmatrix}_{2x1} = \begin{pmatrix} \hat{y}_1 \\ \hat{y}_2 \end{pmatrix}_{2x1}$$

QUESTION: What are the dims of $W^{[1]}$ and $W^{[2]}$?

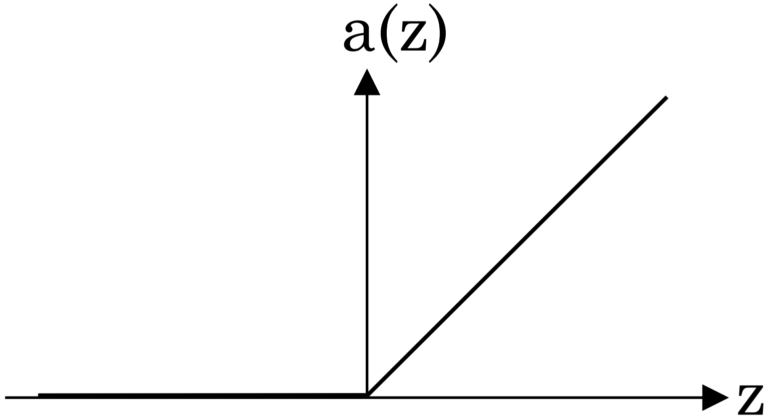
Dims = (a x b); a=? b=?

a=4 and b=6 for $W^{[1]}$

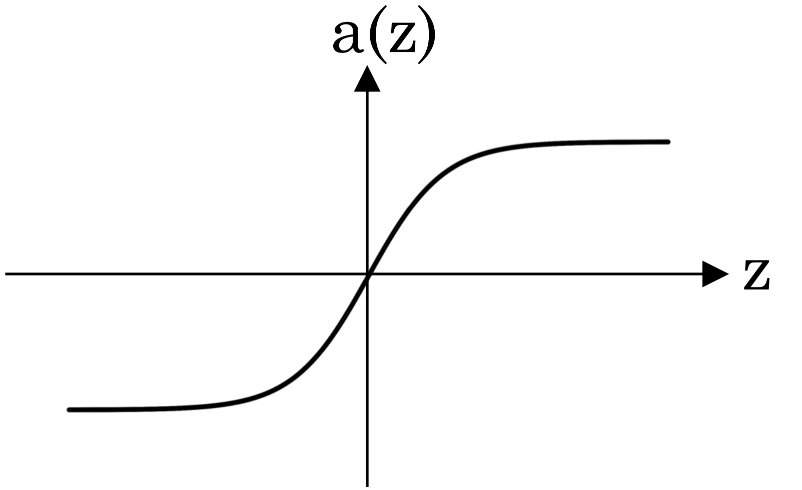
Common Activation Functions



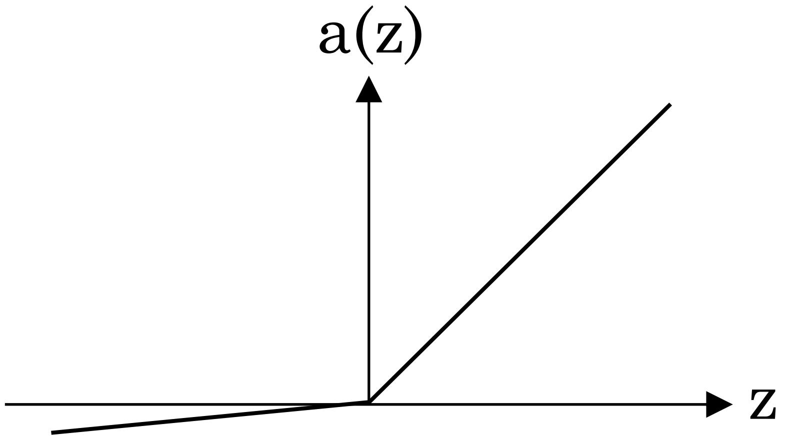
Sigmoid: $a = g(z) = \dfrac{1}{1 + e^{-z}}$



ReLU: $a = g(z) = \max(0, z)$



tanh: $a = g(z) = \dfrac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$



Leaky ReLU: $a = g(z) = \max(0.01z, z)$

Activation Function as: g(**z**)

Algorithm 2:

$$\mathbf{Z}^{[1]} = \mathbf{W}^{[1]}\mathbf{X} + \mathbf{b}^{[1]}$$
$$\mathbf{A}^{[1]} = \sigma(\mathbf{Z}^{[1]})$$
$$\mathbf{Z}^{[2]} = \mathbf{W}^{[2]}\mathbf{A}^{[1]} + \mathbf{b}^{[2]}$$
$$\mathbf{A}^{[2]} = \sigma(\mathbf{Z}^{[2]})$$

Algorithm 2:

$$\mathbf{Z}^{[1]} = \mathbf{W}^{[1]}\mathbf{X} + \mathbf{b}^{[1]}$$
$$\mathbf{A}^{[1]} = g(\mathbf{Z}^{[1]})$$
$$\mathbf{Z}^{[2]} = \mathbf{W}^{[2]}\mathbf{A}^{[1]} + \mathbf{b}^{[2]}$$
$$\mathbf{A}^{[2]} = g(\mathbf{Z}^{[2]})$$

With or Without the Activation Function

Lets have a look at the case where we do not use any activation function. (That is also equivalent to setting $g(\mathbf{z}^{[1]}) = \mathbf{z}^{[1]}$)

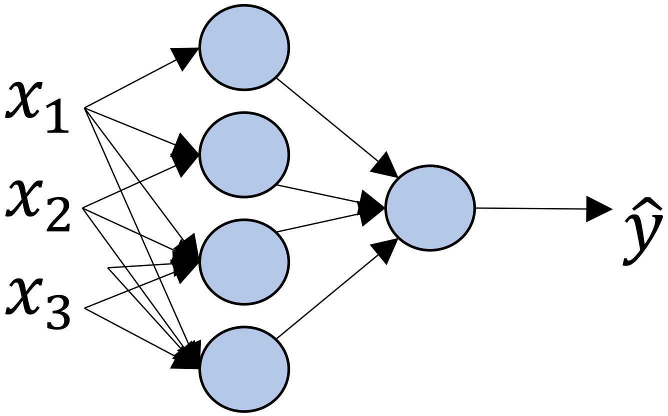
$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]}$$

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

$$\mathbf{z}^{[2]} = \mathbf{W}^{[2]}\mathbf{a}^{[1]} + \mathbf{b}^{[2]}$$

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

$$\hat{y} = \mathbf{a}^{[2]} = \underset{1\text{x}1}{\left(a_1^{[2]}\right)} = \underset{1\text{x}1}{\left(\hat{y}_1\right)}$$



$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]}$$

$$\boxed{\mathbf{a}^{[1]} = \mathbf{z}^{[1]}}$$

$$\mathbf{z}^{[2]} = \mathbf{W}^{[2]}\mathbf{z}^{[1]} + \mathbf{b}^{[2]}$$

$$\boxed{\begin{aligned}\mathbf{a}^{[2]} &= \mathbf{z}^{[2]} = \mathbf{W}^{[2]}\mathbf{z}^{[1]} + \mathbf{b}^{[2]} \\ &= \mathbf{W}^{[2]}[\mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]}] + \mathbf{b}^{[2]} \\ &= \underbrace{\mathbf{W}^{[2]}\mathbf{W}^{[1]}}_{\mathbf{W}}\mathbf{x} + \underbrace{\mathbf{W}^{[2]}\mathbf{b}^{[1]} + \mathbf{b}^{[2]}}_{\mathbf{b}} \\ &= \mathbf{W}\mathbf{x} + \mathbf{b}\end{aligned}}$$

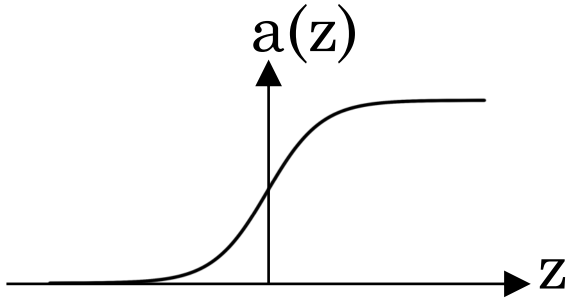
$$\hat{y} = \mathbf{z}^{[2]} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

The output is always a linear function of the input!

Derivatives for the Activation Functions

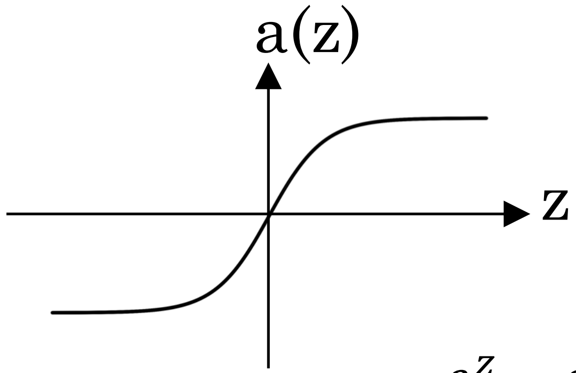
- Remember that the updating process of the parameters depends on the derivatives!
- That also depends on the derivative of the chosen activation function!
  - (we used sigmoid function previously in our logistic regression implementation).

Derivatives for the Activation Functions



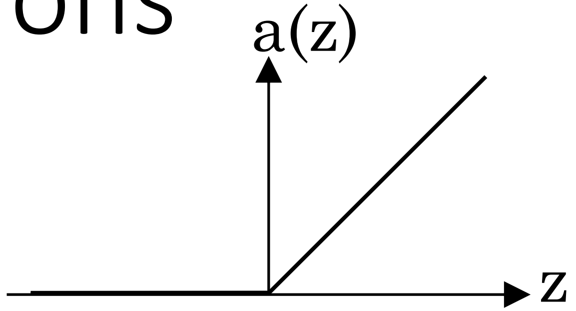
Sigmoid: $a = g(z) = \dfrac{1}{1+e^{-z}}$

$$g'(z) = \frac{dg(z)}{dz} = g(z)(1-g(z)) = a(1-a)$$



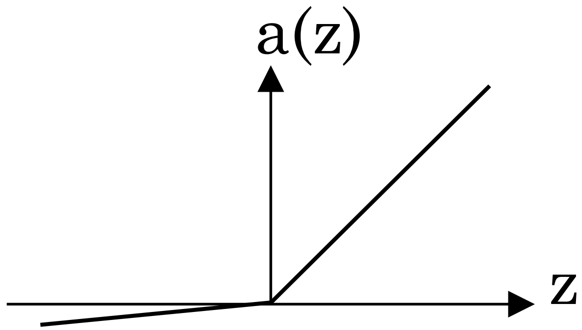
tanh: $a = g(z) = \dfrac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$

$$g'(z) = \frac{dg(z)}{dz} == (1-(\tanh(z))^2) = (1-a^2)$$



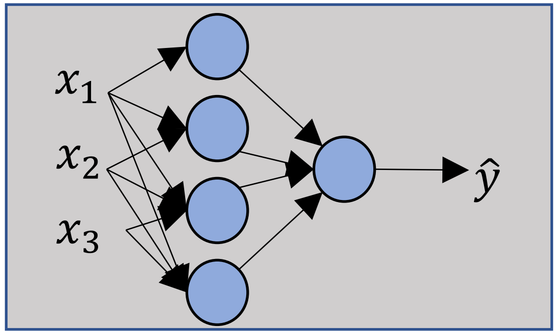
ReLU: $a = g(z) = \max(0, z)$

$$g'(z) = \frac{dg(z)}{dz} = \begin{cases} 0, if\ z < 0 \\ 1, if\ z > 0 \\ undefined, if\ z = 0 \end{cases}$$



Leaky ReLU: $a = g(z) = \max(0.01z, z)$

$$g'(z) = \frac{dg(z)}{dz} = \begin{cases} 0.01, & if\ z < 0 \\ 1, & if\ z \geq 0 \end{cases}$$



$$x,\ W^{[1]},\ b^{[1]} \rightarrow \boxed{W^{[1]}x + b^{[1]} = z^{[1]}} \rightarrow \boxed{\sigma\left(z^{[1]}\right) = a^{[1]}} \rightarrow \boxed{W^{[2]}a^{[1]} + b^{[2]} = z^{[2]}} \rightarrow \boxed{\sigma\left(z^{[2]}\right) = a^{[2]}} \rightarrow \boxed{\mathcal{L}(a^{[2]}, \mathrm{y})}$$

$$W^{[2]},\ b^{[2]} \rightarrow \boxed{W^{[2]}a^{[1]} + b^{[2]} = z^{[2]}}$$

$$dz^{[2]} = a^{[2]} - y$$

$$dW^{[2]} = dz^{[2]}{a^{[1]}}^T$$

$$db^{[2]} = dz^{[2]}$$

$$dz^{[1]} = W^{[2]T}dz^{[2]} * g^{[1]\prime}(\mathrm{z}^{[1]})$$

$$dW^{[1]} = dz^{[1]}x^T$$

$$db^{[1]} = dz^{[1]}$$

$$dZ^{[2]} = A^{[2]} - Y$$

$$dW^{[2]} = \frac{1}{m}dZ^{[2]}{A^{[1]}}^T$$

$$db^{[2]} = \frac{1}{m}np.sum(dZ^{[2]}, axis = 1, keepdims = True)$$

$$dZ^{[1]} = W^{[2]T}dZ^{[2]} * g^{[1]\prime}(\mathrm{Z}^{[1]})$$

$$dW^{[1]} = \frac{1}{m}dZ^{[1]}X^T$$

$$db^{[1]} = \frac{1}{m}np.sum(dZ^{[1]}, axis = 1, keepdims = True)$$