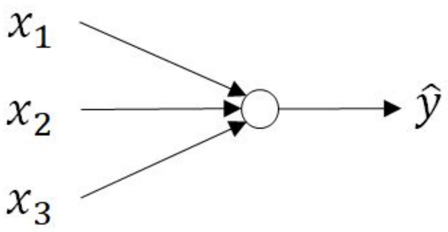
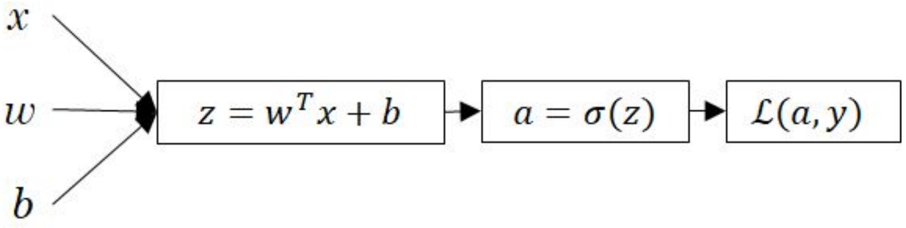
Shallow Neural Networks

CSE 4237 - Soft Computing

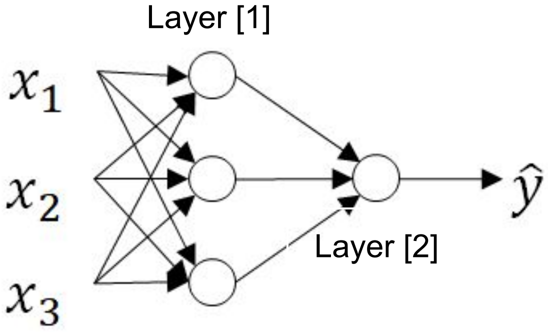
Mir Tafseer Nayeem
Faculty Member, CSE AUST
tafseer.nayeem@gmail.com

What is a Neural Network?

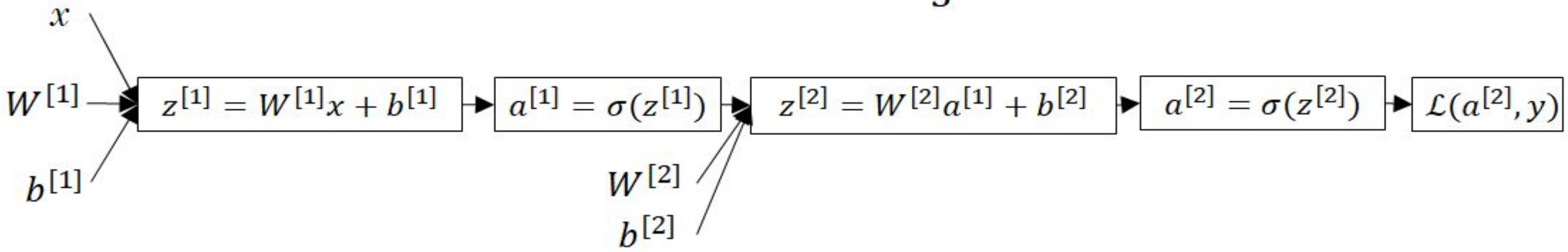




Logistic Regression

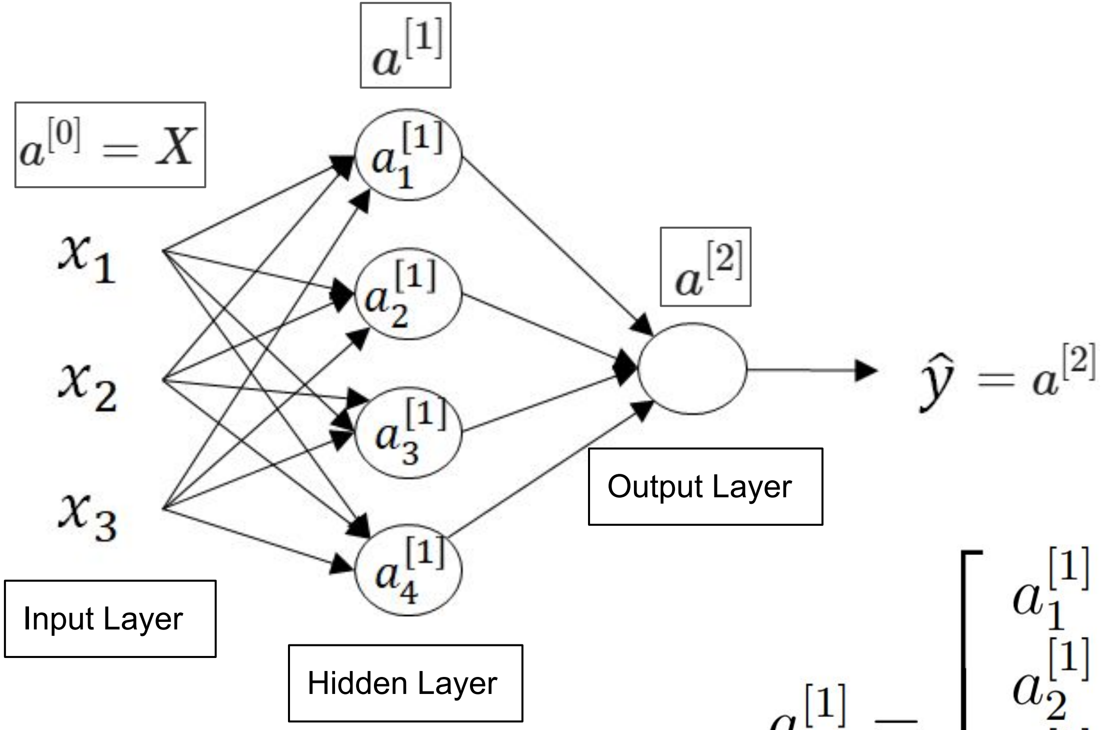


Neural Network



Content Credit: Andrew Ng

Neural Network Representation

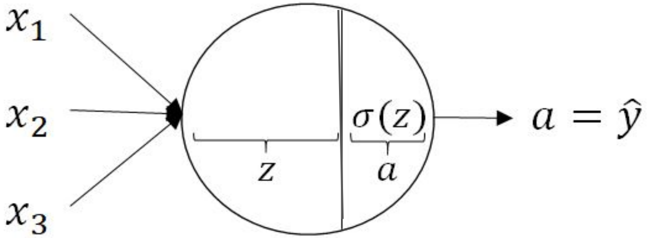


2 Layer Neural Network

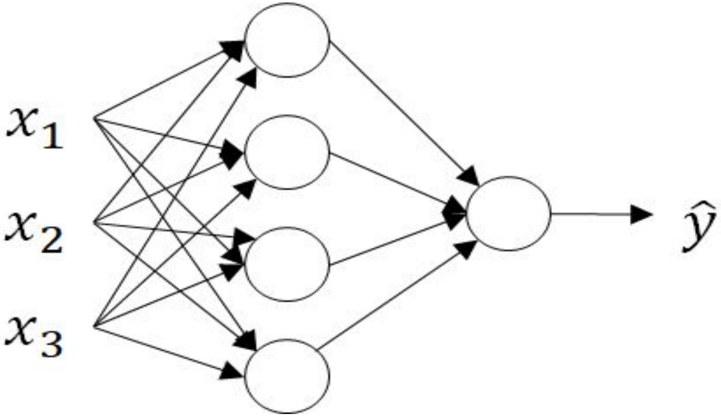
$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix}$$

- In supervised learning, the training set contains the input as well as the target output.
- Hidden layer means the true values in the middle are not observed means and we cannot see that in the training set.
- Activations are the values different layers of the neural network are passing on to the subsequent layers.
- When we count layers in Neural Network we don’t count input layer which is layer 0.

Neural Network Representation



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



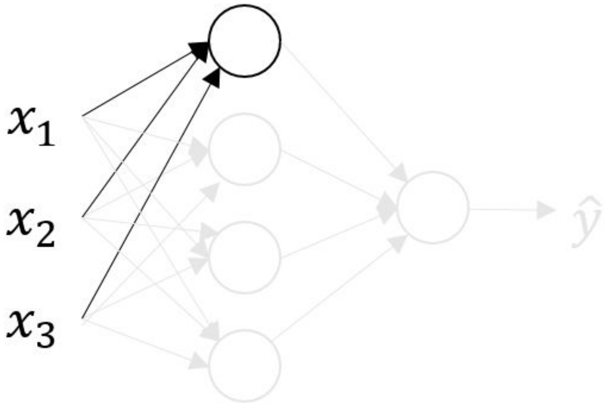
Like Logistic Regression but repeated a lot of time.

Neural Network Representation

$$a_i^{[l]}$$

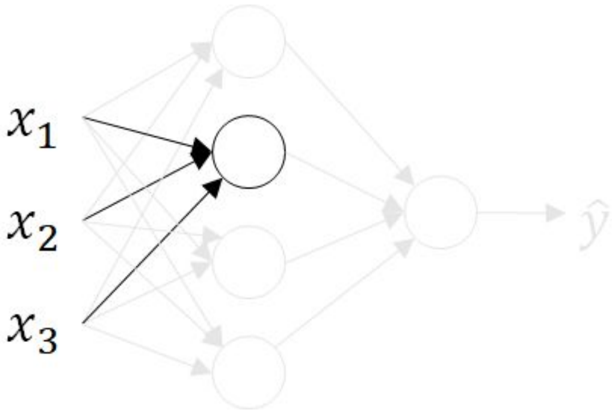
Layer

Node in the Layer



$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}$$

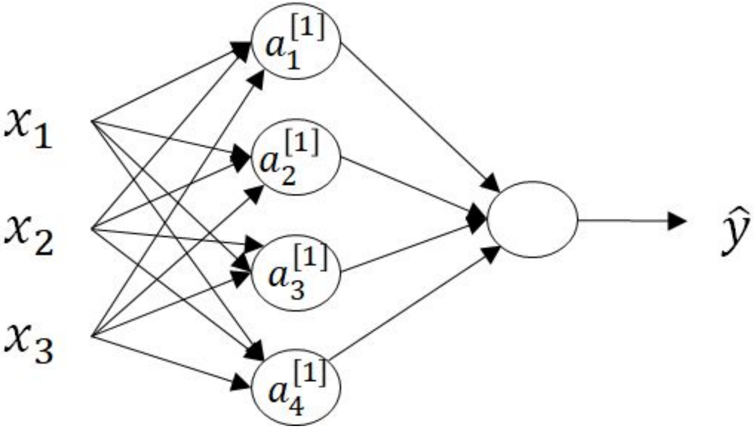
$$a_1^{[1]} = \sigma(z_1^{[1]})$$



$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}$$

$$a_2^{[1]} = \sigma(z_2^{[1]})$$

Neural Network Representation



Vector

$$z_1^{[1]} = \boxed{w_1^{[1]T}} x + b_1^{[1]}, a_1^{[1]} = \sigma(z_1^{[1]})$$

$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}, a_2^{[1]} = \sigma(z_2^{[1]})$$

$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]}, a_3^{[1]} = \sigma(z_3^{[1]})$$

$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}, a_4^{[1]} = \sigma(z_4^{[1]})$$

$$z^{[1]} = \underbrace{\begin{bmatrix} z_1^{[1]} \\ \vdots \\ \vdots \\ z_4^{[1]} \end{bmatrix}}_{z^{[1]} \in \mathbb{R}^{4\times 1}} = \underbrace{\begin{bmatrix} - W_1^{[1]T} - \\ - W_2^{[1]T} - \\ \vdots \\ - W_4^{[1]T} - \end{bmatrix}}_{\substack{W^{[1]} \in \mathbb{R}^{4\times 3} \\ W^{[1]}}} \text{x} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{\substack{x \in \mathbb{R}^{3\times 1} \\ x}} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ \vdots \\ b_4^{[1]} \end{bmatrix}}_{\substack{b^{[1]} \in \mathbb{R}^{4\times 1} \\ b^{[1]}}} = \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T} x + b_2^{[1]} \\ \vdots \\ w_4^{[1]T} x + b_4^{[1]} \end{bmatrix}$$

Neural Network Representation

$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}, a_1^{[1]} = \sigma(z_1^{[1]})$$

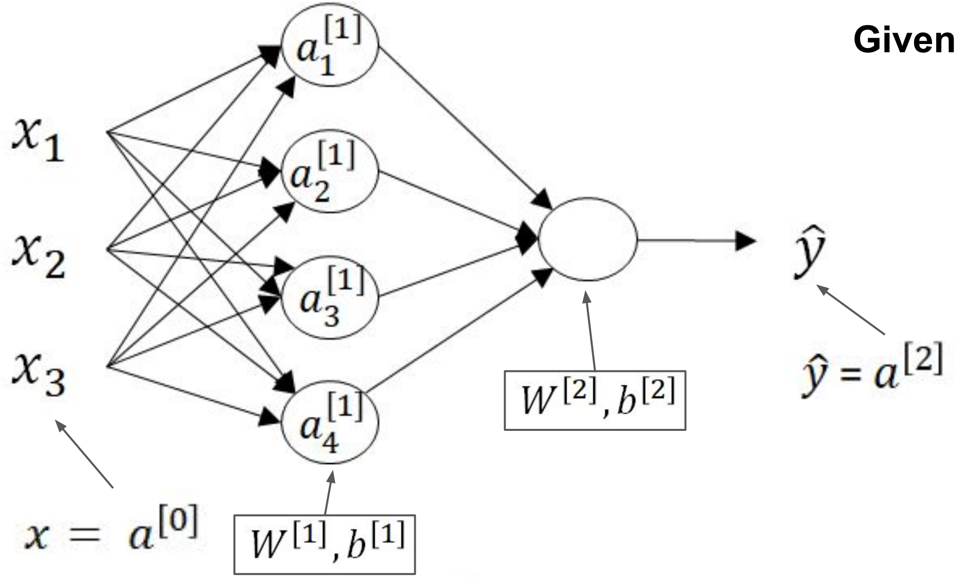
$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}, a_2^{[1]} = \sigma(z_2^{[1]})$$

$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]}, a_3^{[1]} = \sigma(z_3^{[1]})$$

$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}, a_4^{[1]} = \sigma(z_4^{[1]})$$

$$z^{[1]} = \underbrace{\begin{bmatrix} z_1^{[1]} \\ \vdots \\ \vdots \\ z_4^{[1]} \end{bmatrix}}_{z^{[1]} \in \mathbb{R}^{4\times 1}} \quad a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix} = \sigma(z^{[1]})$$

Neural Network Representation learning



You can replace x with $a^{[0]}$

Given Input x

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

(4,1) (4,3) (3,1) (4, 1)

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

(4, 1) (4, 1)

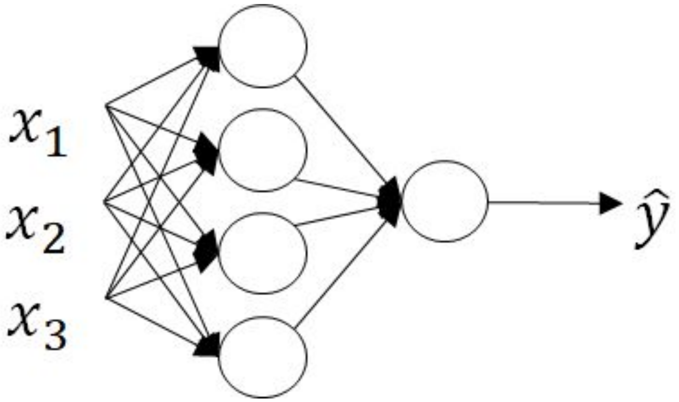
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

(1, 1) (1, 4) (4, 1) (1, 1)

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

(1, 1) (1, 1)

Vectorizing across multiple examples



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

X . . . . $\hat{y} = a^{[2]}$

$x^{(1)}$ . . . . $a^{[2](1)} = \hat{y}^{(1)}$

$x^{(2)}$ . . . . $a^{[2](2)} = \hat{y}^{(2)}$

$\vdots$ $\vdots$

$x^{(m)}$ . . . . $a^{[2](m)} = \hat{y}^{(m)}$

Layer No 2

$a^{[2](i)}$ → Example (i)

Vectorizing across multiple examples

m Training Example

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

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

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

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

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

Output prediction of the neural network. We need to **vectorize** this in order to get rid of this **for loop**.

One Training Example

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

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

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

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

Vectorizing across multiple examples

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

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

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

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

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

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

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

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

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

Stacking them Horizontally

Hidden Units

$$\mathbf{X} = \begin{bmatrix} \vdots & \vdots & & \vdots \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ \vdots & \vdots & & \vdots \end{bmatrix} \text{ while } \mathbb{R}^{n_x \times m} \quad Z^{[1]} = \begin{pmatrix} \vdots & \vdots & & \vdots \\ z^{[1](1)} & z^{[1](2)} & \dots & z^{[1](m)} \\ \vdots & \vdots & & \vdots \end{pmatrix} \quad A^{[1]} = \begin{pmatrix} \vdots & \vdots & & \vdots \\ a^{[1](1)} & a^{[1](2)} & \dots & a^{[1](m)} \\ \vdots & \vdots & & \vdots \end{pmatrix}$$

Training Examples

Justification for vectorized implementation

First Example

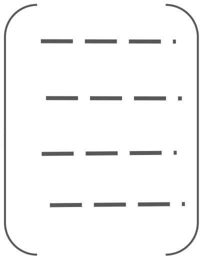
$$z^{[1](1)} = w^{[1]}x^{(1)} + b^{[1]}$$

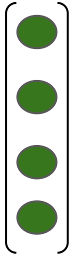
Second Example

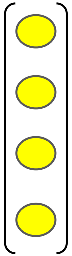
$$z^{[1](2)} = w^{[1]}x^{(2)} + b^{[1]}$$

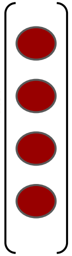
Third Example

$$z^{[1](3)} = w^{[1]}x^{(3)} + b^{[1]}$$

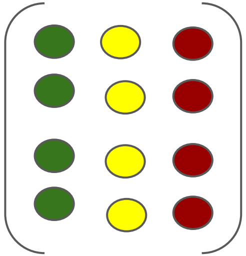
$W^{[1]} =$ 

$W^{[1]}x^{(1)} =$ 

$W^{[1]}x^{(2)} =$ 

$W^{[1]}x^{(3)} =$ 

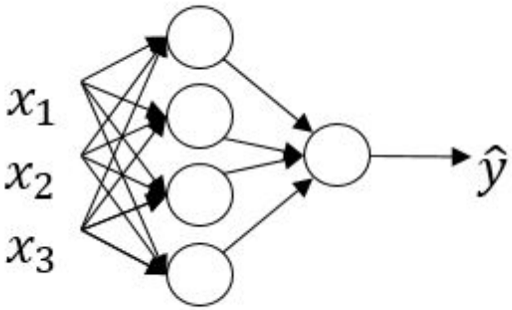
$$W^{[1]} \times \underbrace{\begin{pmatrix} \boxed{x^{(1)}} & \boxed{x^{(2)}} & \boxed{x^{(3)}} \end{pmatrix}}_{X} =$$



$$= \begin{pmatrix} \boxed{z^{[1](1)}} & \boxed{z^{[1](2)}} & \boxed{z^{[1](3)}} \end{pmatrix} = Z^{[1]}$$

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

Justification for vectorized implementation



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

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

Previous implementation of Forward Propagation

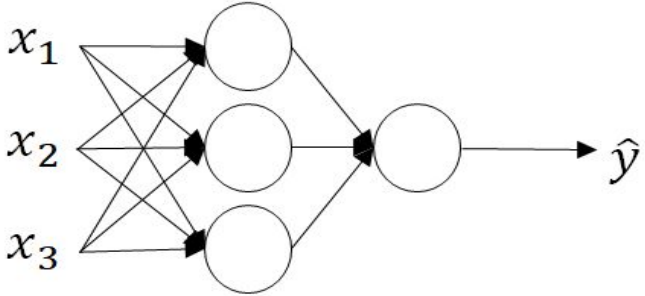
for i = 1 to m

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

$$Z^{[1]} = W^{[1]}X + b^{[1]} \longleftrightarrow Z^{[1]} = W^{[1]}A^{[0]} + b^{[1]}$$
$$A^{[1]} = \sigma(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$A^{[2]} = \sigma(Z^{[2]})$$

Vectorized implementation of Forward Propagation

Activation functions



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

Forward Propagation steps of Neural Network.

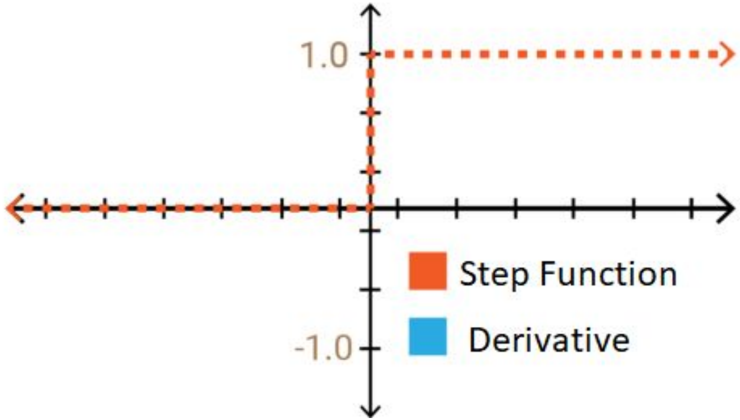
$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = g\,(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = g\,(z^{[2]})$$

g can be any non linear activation function.

Activation function can be used in the **hidden layers** and the **output layers**. Until now we are using Sigmoid activation function but other choices might work better!

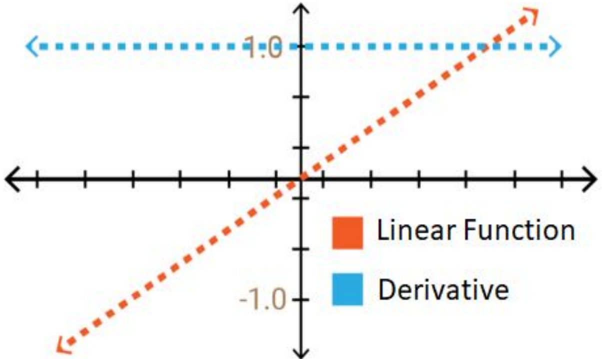
Types of Activation Functions

- Binary Step Function
  - A binary step function is a **threshold-based activation function**.
  - If the input value is **above or below a certain threshold**, the neuron is activated and sends exactly the same signal to the next layer.
  - The problem with a step function is that it does not allow **multi-value outputs**—for example, it cannot support classifying the inputs into one of several categories.
  - It is **not recommended to use it in hidden layers** because it does not represent derivative learning value.



Types of Activation Functions

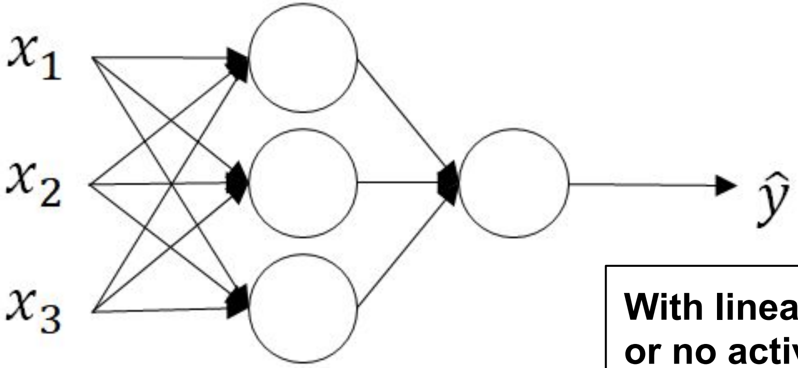
- Linear Activation Function
  - A linear activation function takes the form: **A = cx.**
  - It takes the inputs, multiplied by the weights for each neuron, and creates an output signal proportional to the input. **In one sense, a linear function is better than a step function because it allows multiple outputs, not just yes and no.**
- Linear activation function has two major problems:
  - **Not possible to use backpropagation (gradient descent) to train the model** — the **derivative of the function is a constant**, and has no relation to the input, X. So it's not possible to go back and understand which weights in the input neurons can provide a better prediction.



Linear Activation Function

- When **A = c.x** is derived from **x**, we reach **c**. This means that there is no relationship with **x**. If the derivative is **always a constant value**, can we say that the learning process is taking place? **Unfortunately no!**
- **All layers of the neural network collapse into one**—with **linear activation functions, no matter how many layers in the neural network**, the last layer will be a linear function of the first layer (***because a linear combination of linear functions is still a linear function***). **So a linear activation function turns the neural network or even a deep neural network into just one layer.**
- **A neural network with a linear activation function or without any activation function is simply a linear regression model.** It has limited power and ability to handle complexity varying parameters of input data.

Why do you need nonlinear activation functions?



Given x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = g^{[1]}(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = g^{[2]}(z^{[2]})$$

With linear activation function or no activation function

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

Why do you need nonlinear activation functions?

- If the activation function is **not applied or we apply linear activation function**, the output signal becomes a **simple linear function**.
- Linear functions are only **single-grade polynomials**.
- A **non-activated neural network will act as a linear regression** with limited learning power.
- But we also want our neural network to learn non-linear states. **Because we will give you complex real-world information such as image, video, text, and sound which are non-linear or have high dimensionality to learn to our neural network.**
- Multilayered deep neural networks can learn meaningful featıures from data.

Why do you need nonlinear activation functions?

- They **allow backpropagation because they have a derivative function which is related to the inputs**.
- They **allow “stacking” of multiple layers of neurons to create a deep neural network**. Multiple hidden layers of neurons are **needed to learn complex data sets** with high levels of accuracy.

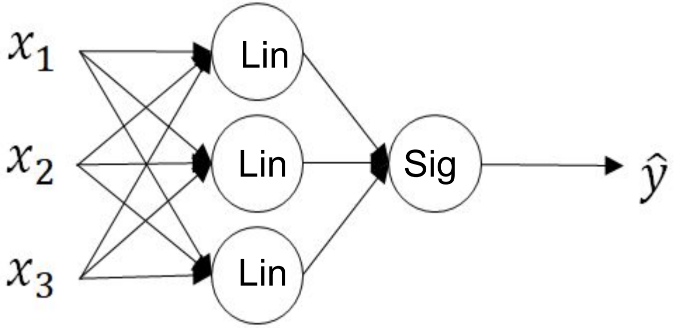
Why do you need nonlinear activation functions?

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

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

$$a^{[2]} = W^{[2]}(W^{[1]}x + b^{[1]}) + b^{[2]}$$

$$a^{[2]} = \underbrace{(W^{[2]}\, W^{[1]})}_{\mathbf{m}}\, x + \underbrace{(W^{[2]}\, b^{[1]} + b^{[2]})}_{\mathbf{c}}$$



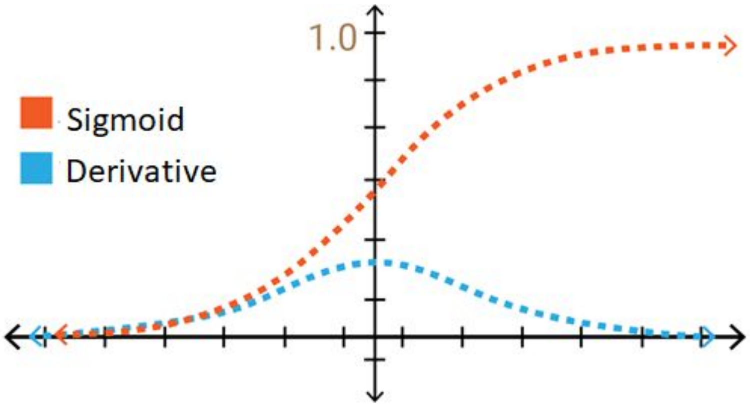
It is just a logistic regression.

Neural Network is outputting a linear function of inputs!
Composition of two or more linear function is also a linear function.

One place you can use a **linear activation function when you are solving a regression problem** means the output is a real number. But only in the last hidden layer not in the intermediate layers.

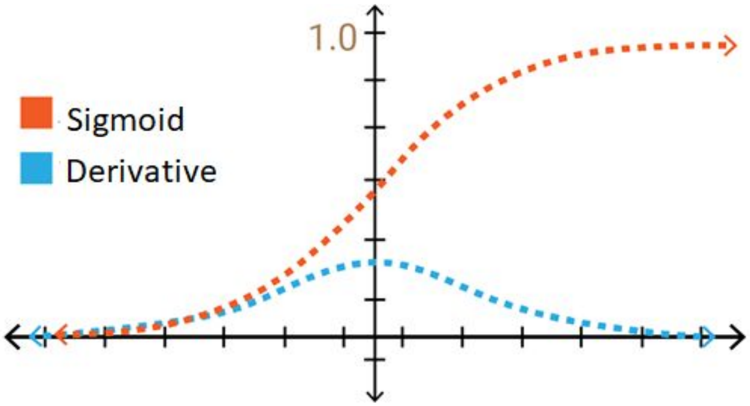
Sigmoid / Logistic Function

- It is also derived because it is different from the step function. **This means that learning can happen.**
- **Smooth gradient**, preventing **“jumps”** in output values.
- **Output values bound** between 0 and 1, normalizing the output of each neuron.
- **Clear predictions** — For **X** above 2 or below -2, tends to bring the Y value (the prediction) to the edge of the curve, very close to 1 or 0. This enables clear predictions.
- **The sigmoid function is the most frequently used activation function, but there are many other and more efficient alternatives.**



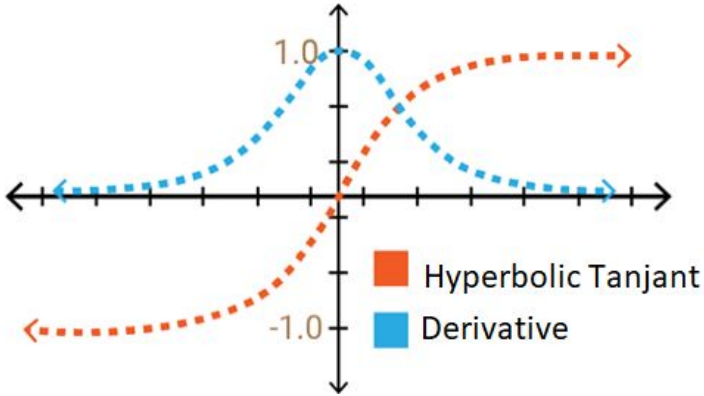
What’s the problem with sigmoid function?

- **Vanishing gradient**—for very high or very low values of X, there is almost no change to the prediction. The derivative values in these regions are very small and converge to 0. This is called the vanishing gradient and the learning is minimal.
- The network can refuse to learn further, or being too slow to reach an accurate prediction.
- When slow learning occurs, the optimization algorithm that minimizes error **can be attached to local minimum** values and cannot get maximum performance from the artificial neural network model.
- **Outputs not zero centered.** So output of all the neurons will be of the same sign.
- **Computationally expensive.**



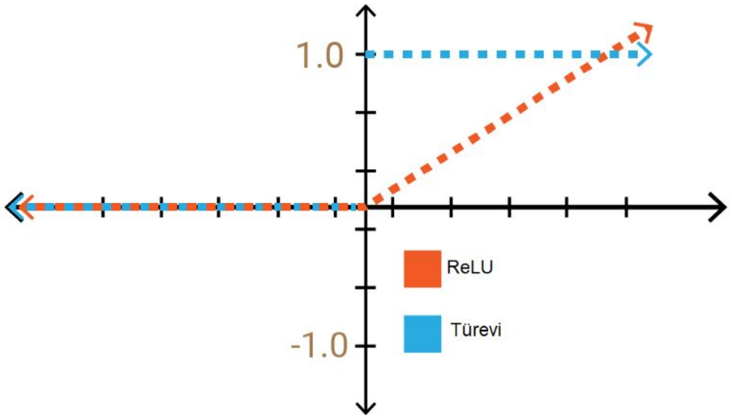
Hyperbolic Tangent Function

- It has a structure very similar to Sigmoid function.
- **Zero centered**—making it easier to model inputs that have strongly negative, neutral, and strongly positive values. **The range of values in this case is from -1 to 1**.
- The advantage over the sigmoid function is that **its derivative is more steep**, which means it can get more value.
- **This means that it will be more efficient** because it has a wider range for faster learning and grading.
- But again, the problem of gradients at the ends of the function continues.



ReLU (Rectified Linear Unit) Function

- **Computationally efficient**—allows the network to converge very quickly.
- **Non-linear**—although it looks like a linear function, ReLU has a derivative function and allows for backpropagation
- The Dying ReLU problem—when inputs approach zero, or are negative, the gradient of the function becomes zero, the network cannot perform backpropagation and cannot learn.

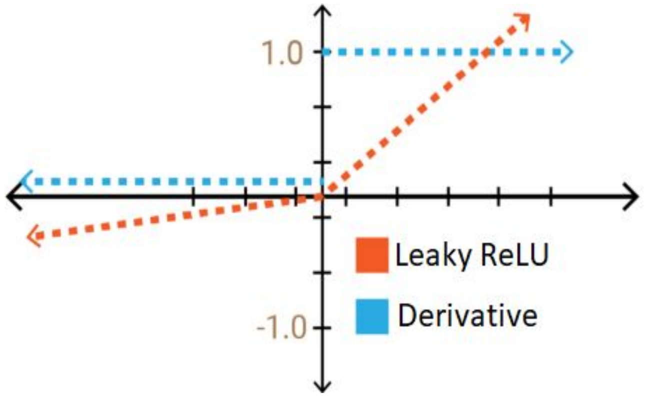


ReLU - What are the returns and their benefits?

- A large neural network with too many neurons. **Sigmoid and hyperbolic tangent** caused almost all neurons to be **activated in the same way**.
- **This means that the activation is very intensive.** Some of the neurons in the network are active, and activation is infrequent, so we want an efficient computational load.
- **We get it with ReLU.** Having a value of 0 on the negative axis means that the network will run faster as it does not activate all the neurons at the same time.
- The fact that the c**alculation load is less than the sigmoid and hyperbolic tangent** functions has led ReLu to a higher preference for multi-layer networks.

Leaky-ReLU Function

- **Prevents dying ReLU problem** — this variation of ReLU has a **small positive slope** in the negative area, so it does **enable backpropagation**, even for negative input values. **This leaky value is given as a value of 0.01 if not +ve.**
- **Results not consistent** — leaky ReLU does not provide consistent predictions for negative input values.



Other Activation Functions

- **Variants of Leaky-ReLU**
  - Parameterised ReLU
  - Exponential ReLU
- **Swish**
  - Discovered by researchers at **Google in the year 2017**. According to their paper, it performs **better than ReLU with a similar level of computational efficiency.**

Summary of Activation Function Definitions

ACTIVATION FUNCTION	EQUATION	RANGE
Linear Function	$f(x) = x$	$(-\infty, \infty)$
Step Function	$f(x) = \begin{cases} 0 \; for \; x < 0 \\ 1 \; for \; x \geq 0 \end{cases}$	$\{0, 1\}$
Sigmoid Function	$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$	$(0, 1)$
Hyperbolic Tanjant Function	$f(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$	$(-1, 1)$
ReLU	$f(x) = \begin{cases} 0 \; for \; x < 0 \\ x \; for \; x \geq 0 \end{cases}$	$[0, \infty)$
Leaky ReLU	$f(x) = \begin{cases} 0.01 \; for \; x < 0 \\ x \; for \; x \geq 0 \end{cases}$	$(-\infty, \infty)$

Choosing the Right Activation Function

- **Sigmoid functions and their combinations** generally work better in the case of classifiers, specially binary classifiers at the output layer.
- **Sigmoids and tanh functions** are sometimes avoided due to the **vanishing gradient problem**.
- **ReLU function is a general activation function** and is used in most cases these days.
- If we encounter a **case of dead neurons in our networks** the leaky ReLU function is the best choice.
- Always keep in mind that **ReLU function should only be used in the hidden layers.**
- As a rule of thumb, **you can begin with using ReLU function** and then move over to other activation functions in case ReLU doesn’t provide with optimum results.

Softmax

- Softmax function is often described as **a combination of multiple sigmoids**. We know that sigmoid returns values **between 0 and 1**, which can be treated as **probabilities of a data point belonging to a particular class**. Thus sigmoid is widely used for **binary classification problems**.
- The softmax function can be used for **multiclass classification problems**. This function returns the probability for a datapoint belonging to each individual class. Here is the mathematical expression-

The Softmax function can be defined as below, where **c** is equal to the **number of classes**.

$$a_i = \frac{e^{z_i}}{\sum_{k=1}^{c} e^{z_k}}$$

$$\text{where} \sum_{i=1}^{c} a_i = 1$$

Softmax

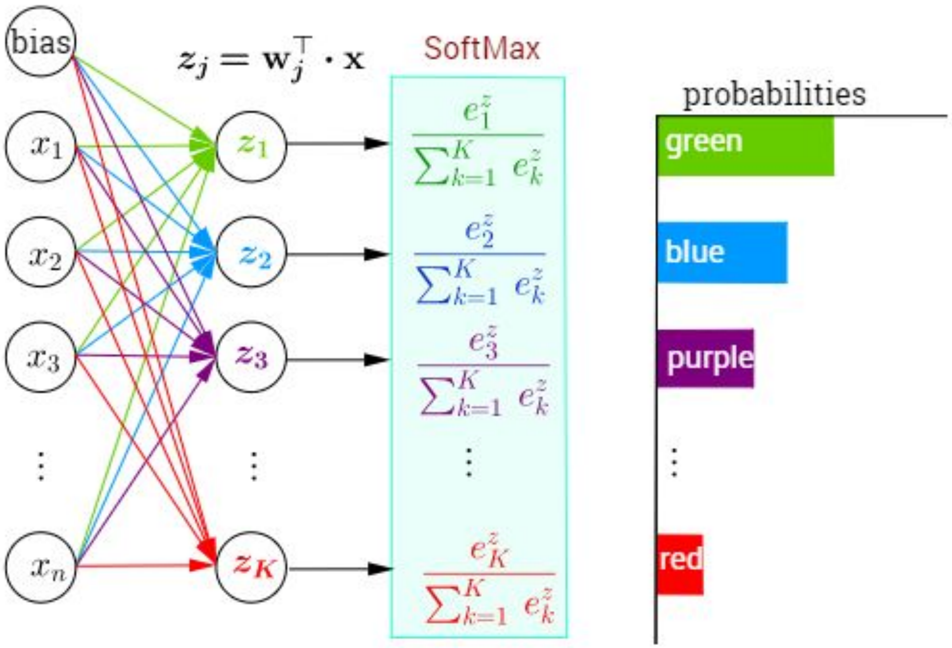
- While building a network for a multiclass problem, the **output layer would have as many neurons** as the number of classes in the target.
- For instance, if you have **three classes**, there would be three neurons in the output layer. Suppose you got the output from the neurons as **[1.2 , 0.9 , 0.75]**.
- Applying the softmax function over these values, you will get the following result **[0.42 , 0.31, 0.27]**. These represent the probability for the data point belonging to each class. Note that the sum of all the values is 1.

$$a_i = \frac{e^{z_i}}{\sum_{k=1}^{c} e^{z_k}}$$

$$\text{where} \sum_{i=1}^{c} a_i = 1$$

Multi-Class Classification with NN and SoftMax Function

$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_K \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \\ \mathbf{w}_3^\top \\ \vdots \\ \mathbf{w}_K^\top \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$



Which Activation Function Should Be Preferred?

- **Easy and fast convergence** of the network can be the first criterion.
- If your network is **too deep and the computational load is a major problem**, ReLU can be preferred than hyperbolic tangent or sigmoid.
- ReLU will be advantageous in terms of speed. You have to **let the gradients die/vanish**. It is usually **used in intermediate layers rather than an output**.
- **Leaky ReLU** can be the first solution to the problem of the **gradients' vanish**.
- For deep learning models, it is advisable to **start experiments with ReLU**.
- **Softmax** is usually used in **output layers**.

Derivative Formulas

In the following, u and v are functions of x, and n, e, a, and k are constants.

1. $f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$ — The Definition of the Derivative.

2. $\frac{d}{dx}(k) = 0$ — The derivative of a constant is zero.

3. $\frac{d}{dx}\left(k\left(u(x)\right)\right) = k\frac{du}{dx}$ — The derivative of a constant times a function.

4. $\frac{d}{dx}\left(u^n\right) = nu^{n-1}\frac{du}{dx}$ — The Power Rule (Variable raised to a constant).

5. $\frac{d}{dx}(u+v) = \frac{du}{dx} + \frac{dv}{dx}$ — The Sum Rule.

6. $\frac{d}{dx}(u-v) = \frac{du}{dx} - \frac{dv}{dx}$ — The Difference Rule.

7. $\frac{d}{dx}(uv) = uv' + vu'$ — The Product Rule.

8. $\frac{d}{dx}\left(\frac{u}{v}\right) = \frac{vu' - uv'}{v^2}$ — The Quotient Rule.

9. $\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx}$ — The Chain Rule.

Derivatives of Activation Functions

- Derivative of sigmoid function

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

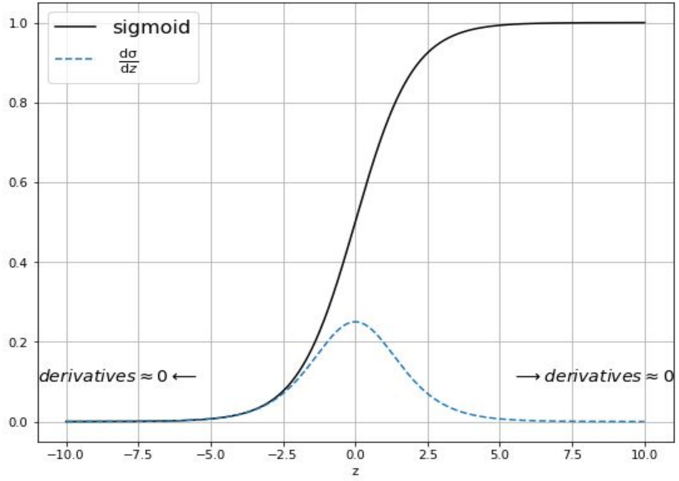
$$\frac{d}{dz}g(z) = slope\ of\ g(z)\ at\ z$$

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

$$z = 10 \quad g(z) \approx 1 \Rightarrow g'(z) \approx 1(1-1) \approx 0$$

$$z = -10 \quad g(z) \approx 0 \Rightarrow g'(z) \approx 0(1-0) \approx 0$$

$$z = 0 \quad g(z) = \frac{1}{2} \Rightarrow g'(z) = \frac{1}{2} = \left(1-\frac{1}{2}\right) = \frac{1}{4}$$



Derivatives of Activation Functions

- Derivative of a tanh function

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

$$\frac{d}{dz}g(z) = slope\ of\ g(z)\ at\ z$$

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

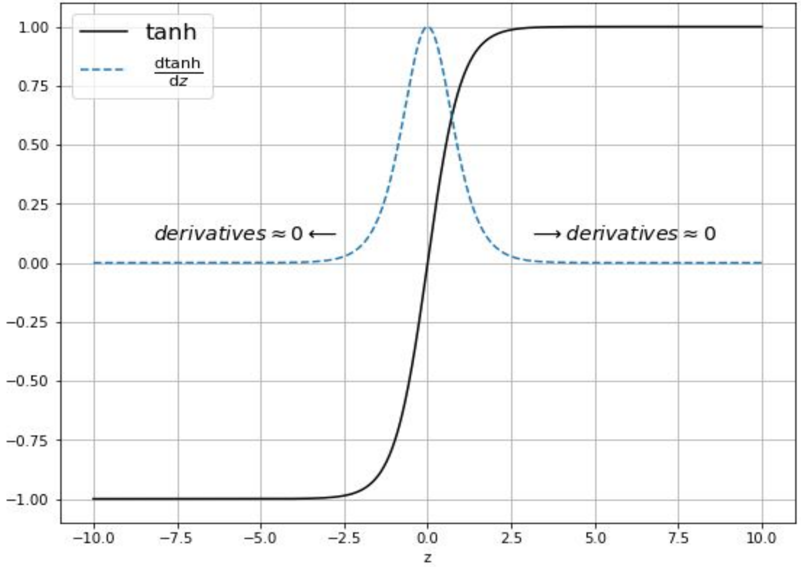
$$\frac{d}{dz}g(z) = \frac{\frac{(e^z+e^{-z})^2 - (e^z-e^{-z})^2}{(e^z+e^{-z})^2}}{\frac{(e^z+e^{-z})^2}{(e^z+e^{-z})^2}} = \frac{\frac{1 - tanh(z)^2}{1}}{1} = 1 - tanh(z)^2$$

Derivatives of Activation Functions

$$z = 10 \quad tanh(z) \approx 1 \Rightarrow \frac{d}{dz}g(z) \approx 1 - 1^2 \approx 0$$

$$z = -10 \quad tanh(z) \approx -1 \Rightarrow \frac{d}{dz}g(z) \approx 1 - (-1)^2 \approx 0$$

$$z = 0 \quad tanh(z) = 0 \Rightarrow \frac{d}{dz}g(z)(z) = 1 - 0^2 = 1$$

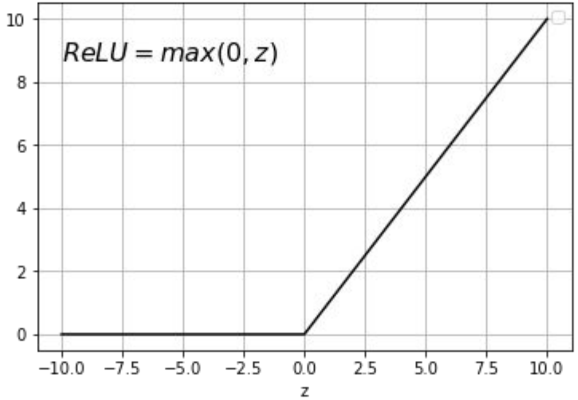


Derivatives of Activation Functions

- Derivative of a ReLU function
  - This function is commonly used activation function nowadays.
  - A derivative of a ReLU function is:

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

$$g'(z) = \begin{cases} 1 \ if \ \ z > 0 \\ 0 \ if \ \ z < 0 \\ \boxed{undefined \ if \ \ z = 0} \end{cases}$$

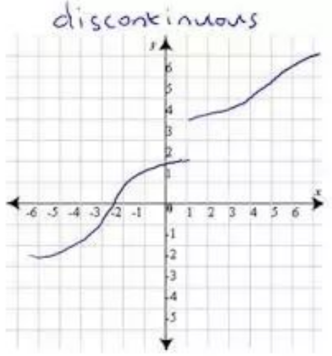


The chance of z = 0.00000…..0000 is very small.
So gradient descent still works just fine.

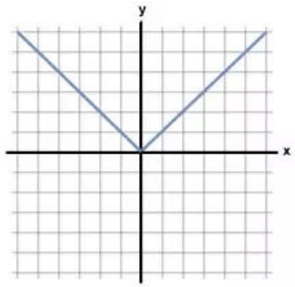
Derivatives of Activation Functions

- You can define a derivative only if the function is continuous at some point and there is one and only one tangent to the curve / function at that point . There are two cases when the derivative doesn't exist.

If the function is discontinuous



If there can be more than one tangents to the curve at that point.

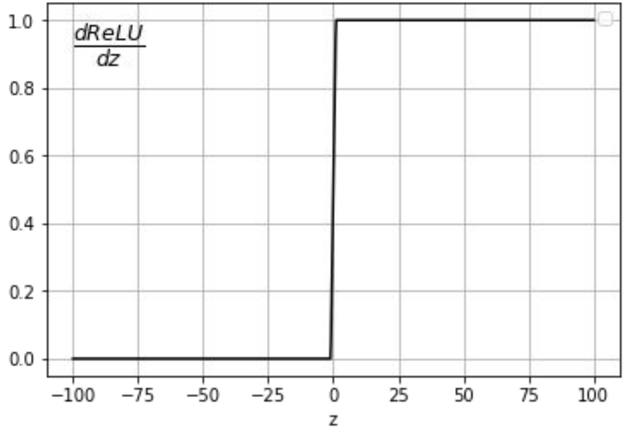


- At **x=0** there can be infinite number of lines that touch the curve at 0. So we can't assign a well defined tangent.
- The right hand derivative and the left hand derivative are not **equal at 0**. So the derivative doesn't exist.

Derivatives of Activation Functions

- Derivative of a ReLU function
  - The derivative of a ReLU function is undefined at 0, but we can say that derivative of this function at zero is either 0 or 1. Both solution would work when they are implemented in software. The same solution works for LeakyReLU function.

$$g'(z) = \begin{cases} 0 & if \;\; z < 0 \\ 1 & if \;\; z \geq 0 \end{cases}$$

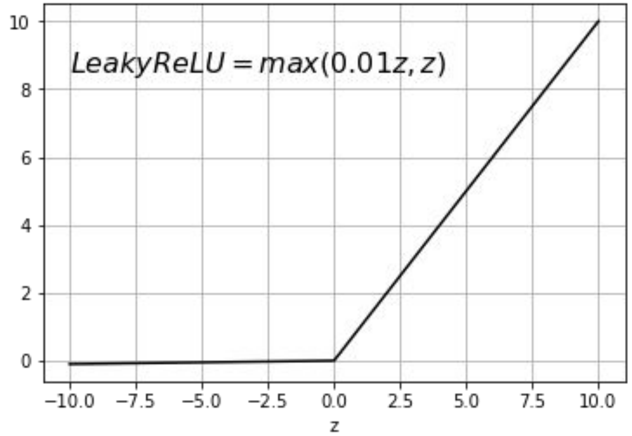


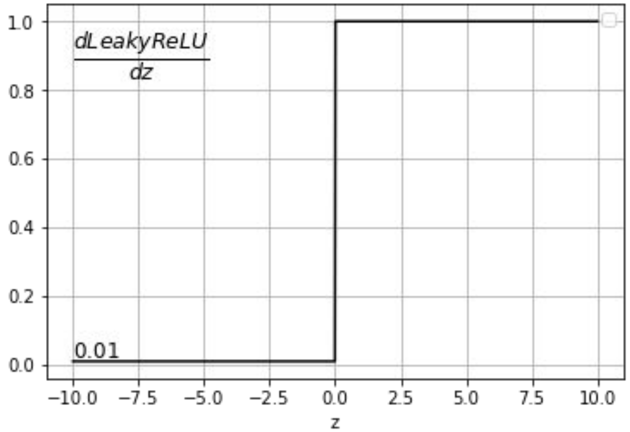
Derivatives of Activation Functions

- Derivative of a LeakyReLU function
    - LeakyReLU usually works better then ReLU function.

$$g(z) = max(0.01z, z)$$

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





END