Intro to Deep Learning

Hello!

I am Eslam Ahmed

I am a software engineer.

You can find me at jeksogsa@gmail.com

Hello!

I am Eman Ehab

I am a ML research engineer.

You can find me at
emanehab.ieee@gmail.com

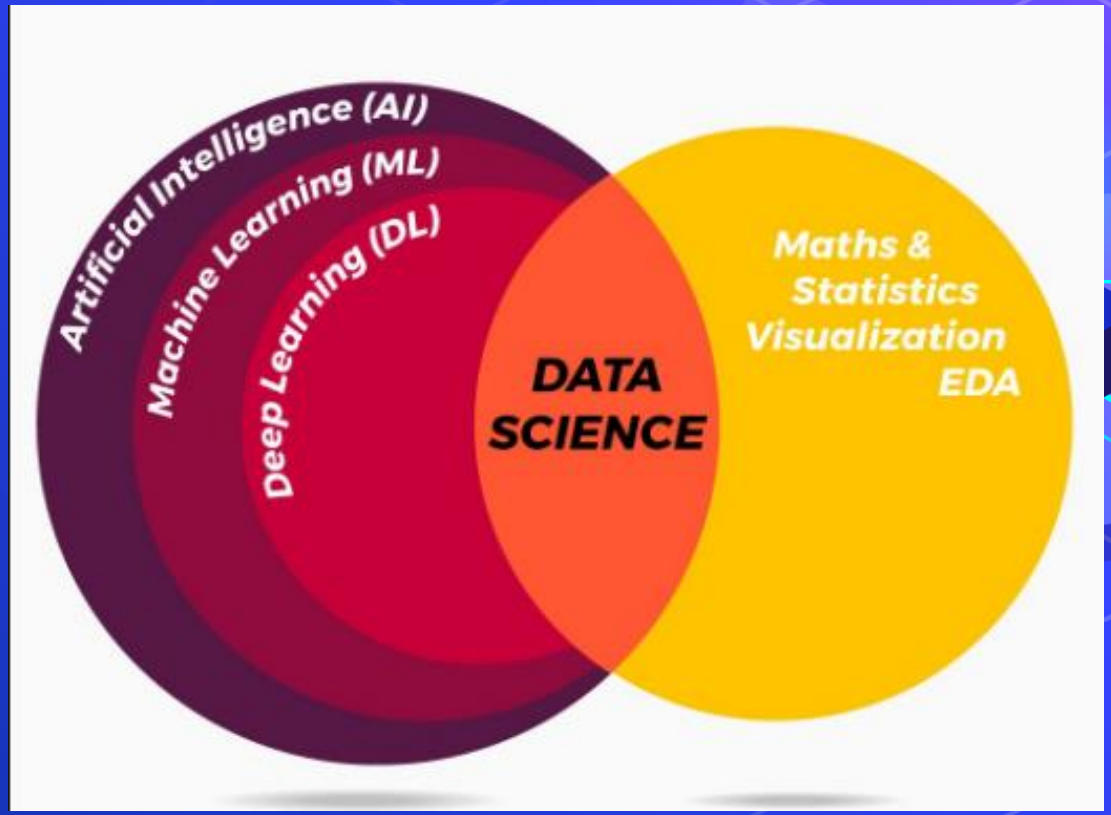
Agenda

- What is Deep Learning
- Deep Learning Applications
- Google Colab
- Perceptron
- Artificial Neural Networks (ANN)
- Error or Loss or Cost functions
- Optimization algorithms
- Backpropagation
- Activation functions
- Improve neural networks training
- Deep Learning frameworks

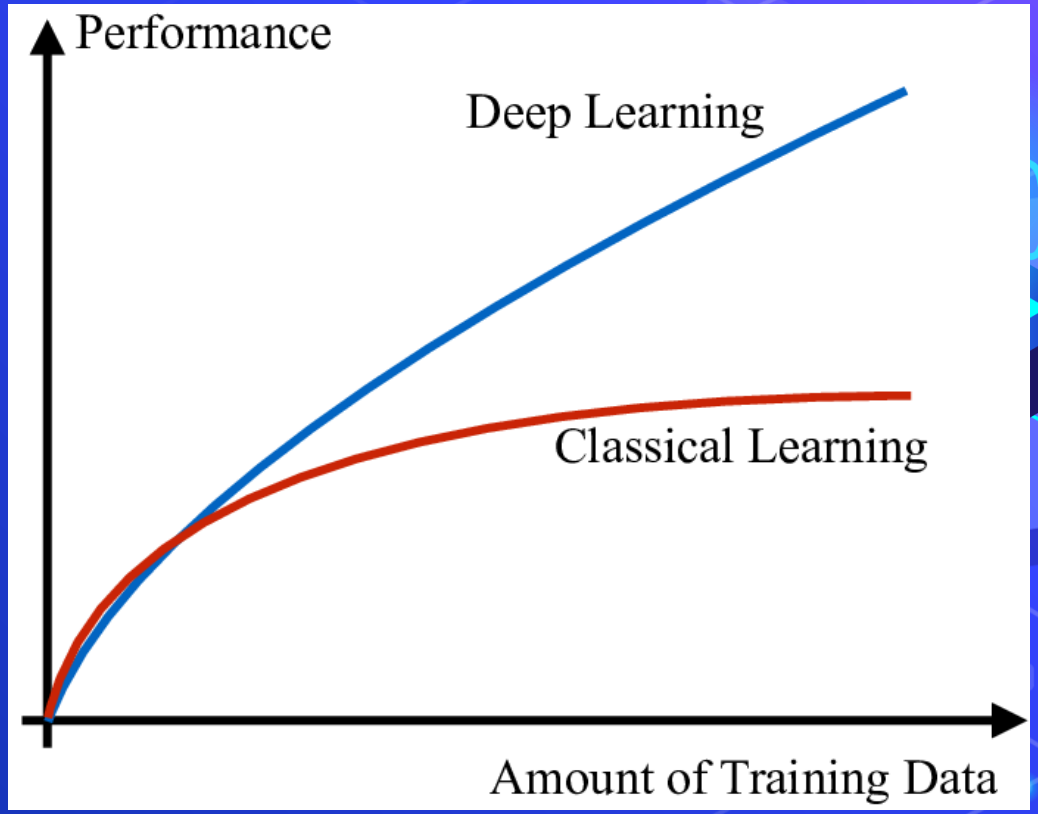
Agenda

- What is Deep Learning
- Deep Learning Applications
- Google Colab
- Perceptron
- Artificial Neural Networks (ANN)
- Error or Loss or Cost functions
- Optimization algorithms
- Backpropagation
- Activation functions
- Improve neural networks training
- Deep Learning frameworks

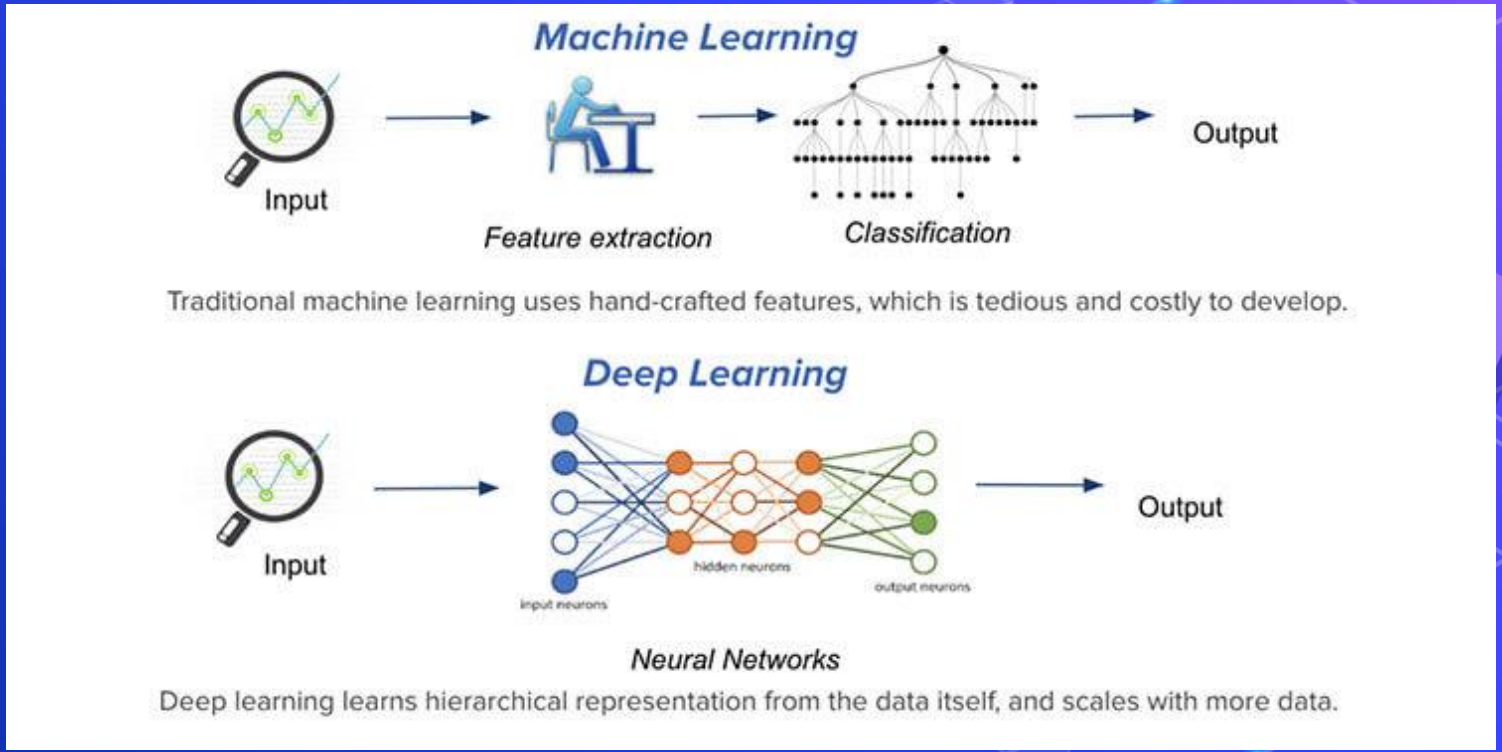
What is Deep Learning



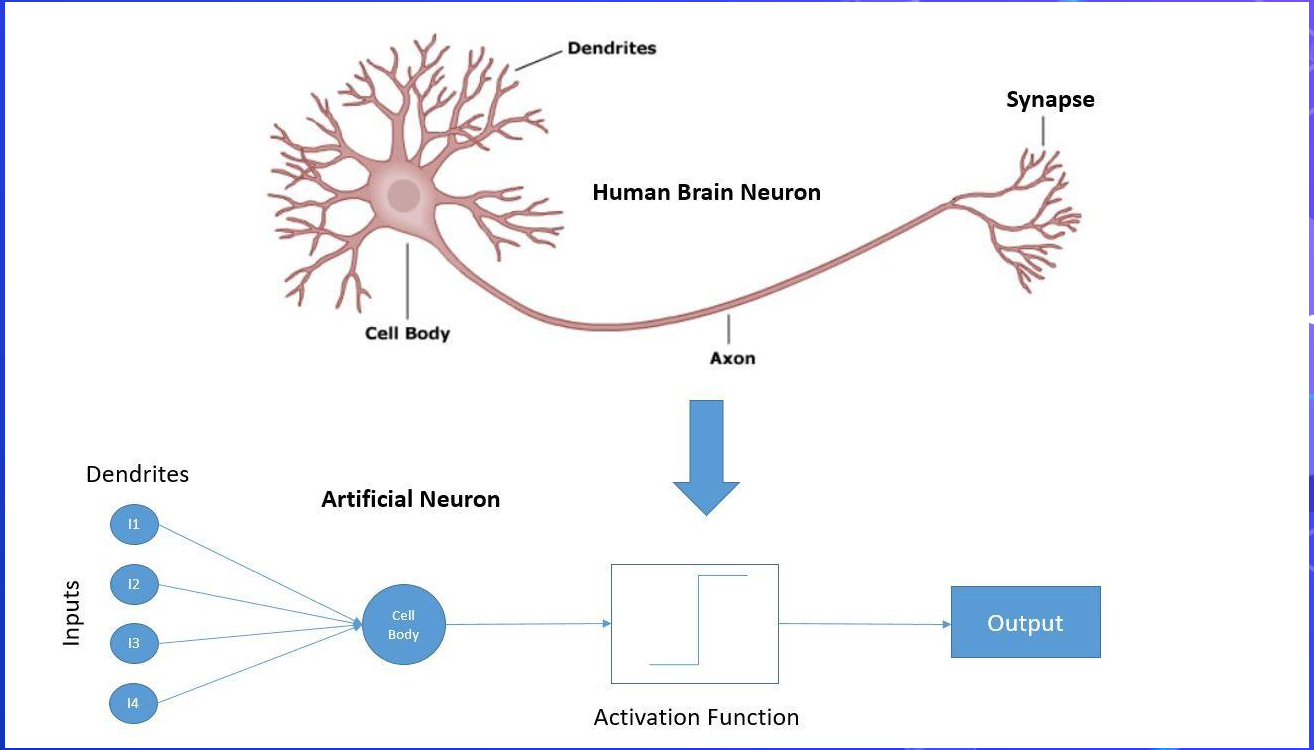
What is Deep Learning



What is Deep Learning



What is Deep Learning



What is Deep Learning

Why is Deep Learning Hot Now?

Big Data Availability

facebook 350 millions images uploaded per day

Walmart 2.5 Petabytes of customer data hourly

YouTube 300 hours of video uploaded every minute

New ML Techniques

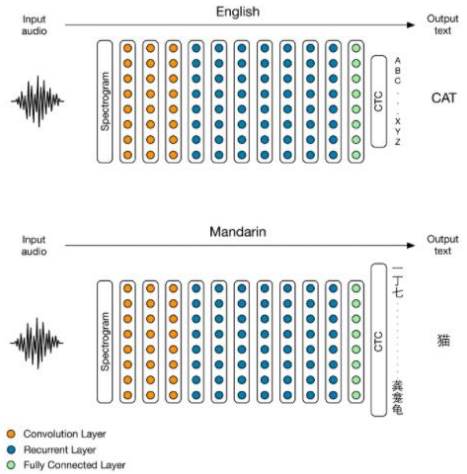
GPU Acceleration

Agenda

- What is Deep Learning
- **Deep Learning Applications**
- Google Colab
- Perceptron
- Artificial Neural Networks (ANN)
- Error or Loss or Cost functions
- Optimization algorithms
- Backpropagation
- Activation functions
- Improve neural networks training
- Deep Learning frameworks

Deep Learning Applications

DL Today: Speech-to-Text



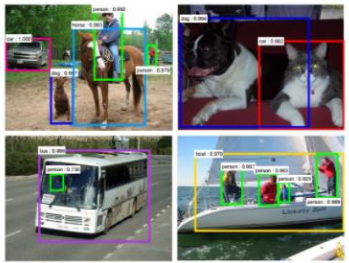
[Baidu 2014]

Deep Learning Applications

DL Today: Vision

[Krizhevsky 2012]

[Ciresan et al. 2013]



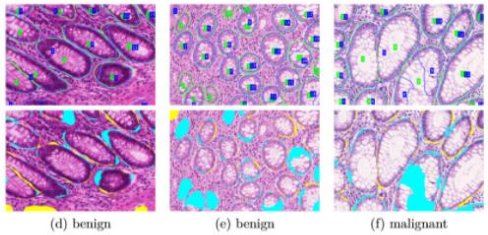
[Faster R-CNN - Ren 2015]

[NVIDIA dev blog]

Deep Learning Applications

DL Today: Vision

[Stanford 2017]



[Nvidia Dev Blog 2017]

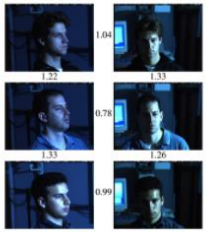


Figure 1. **Illumination and Pose invariance.**

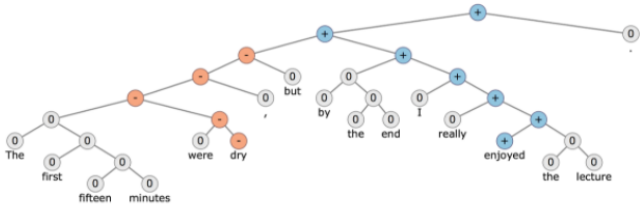
[FaceNet - Google 2015]

[Facial landmark detection CUHK 2014]

Deep Learning Applications

DL Today: NLP

[Google Translate System - 2016]



[Socher 2015]

Deep Learning Applications

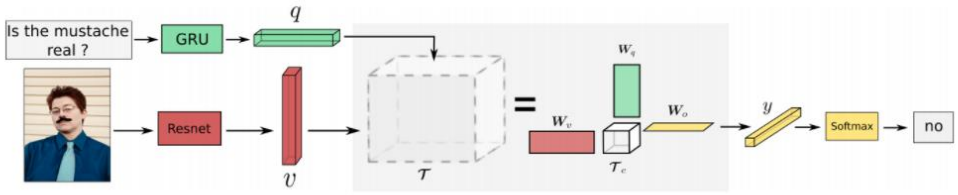
DL Today: NLP

[Google Inbox Smart Reply]

[Amazon Echo / Alexa]

Deep Learning Applications

DL Today: Vision + NLP



[VQA - Mutan 2017]

"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

"boy is doing backflip on wakeboard."

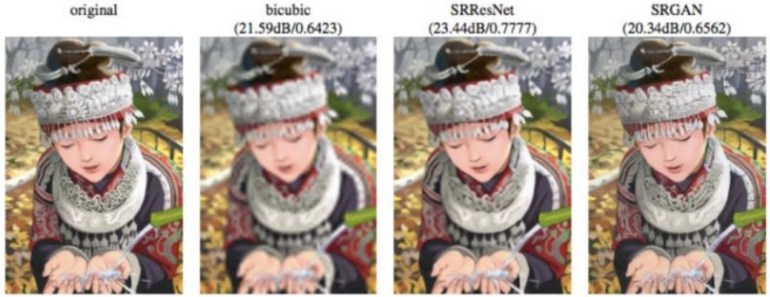
[Karpathy 2015]

Deep Learning Applications

DL Today: Image translation

[DeepDream 2015]

[Gatys 2015]



[Ledig 2016]

Deep Learning Applications

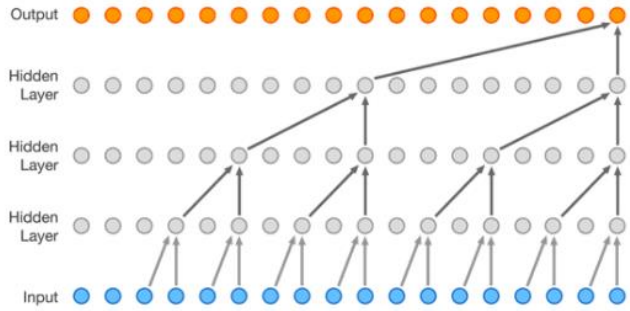
DL Today: Generative models

Sampled celebrities [Nvidia 2017]

StackGAN v2 [Zhang 2017]

Deep Learning Applications

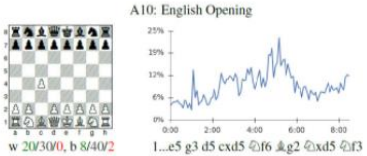
DL Today: Generative models



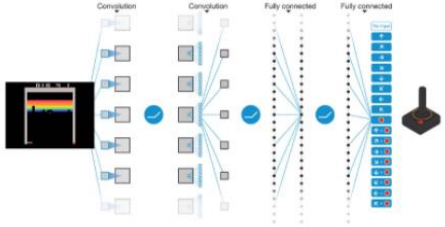
Sound generation with WaveNet [DeepMind 2017]

Deep Learning Applications

DL for AI in games



[Deepmind AlphaGo / Zero 2017]



[Atari Games - DeepMind 2016]

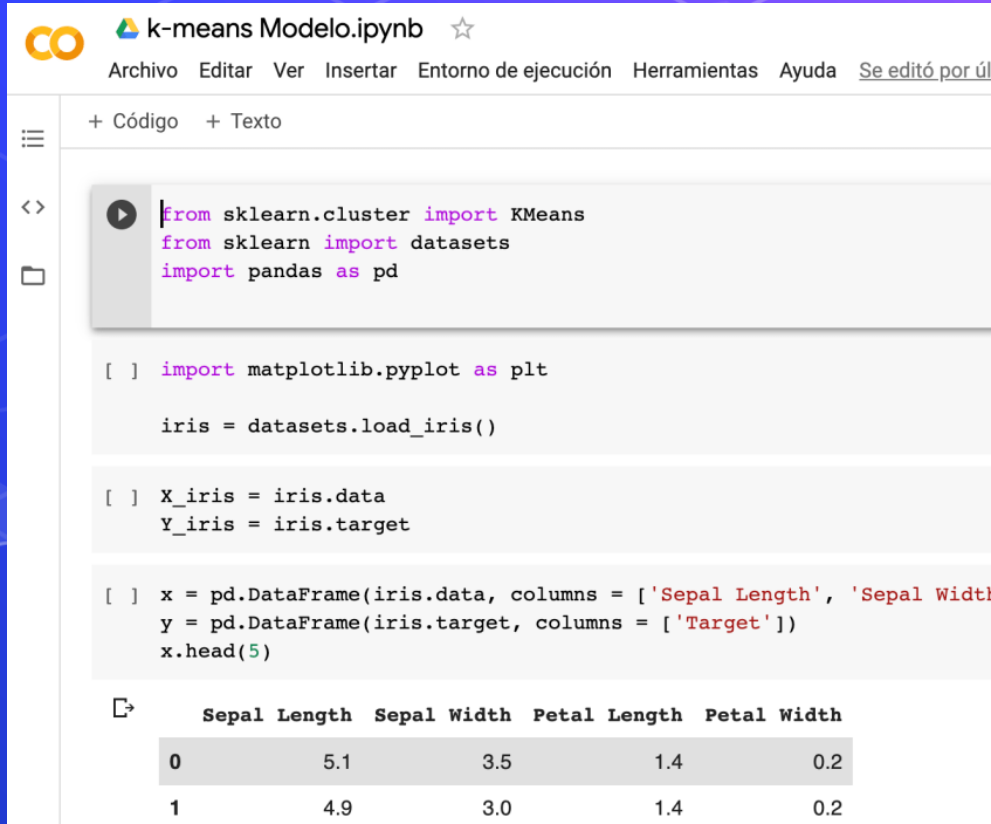
[Starcraft 2 for AI research]

Agenda

- What is Deep Learning
- Deep Learning Applications
- **Google Colab**
- Perceptron
- Artificial Neural Networks (ANN)
- Error or Loss or Cost functions
- Optimization algorithms
- Backpropagation
- Activation functions
- Improve neural networks training
- Deep Learning frameworks

Google Colab

- High speed download.
- No Environment Setup.
- Make use of **GPU** or **TPU**.
- Make use of Google Drive.
- Free to use.



Google Colab

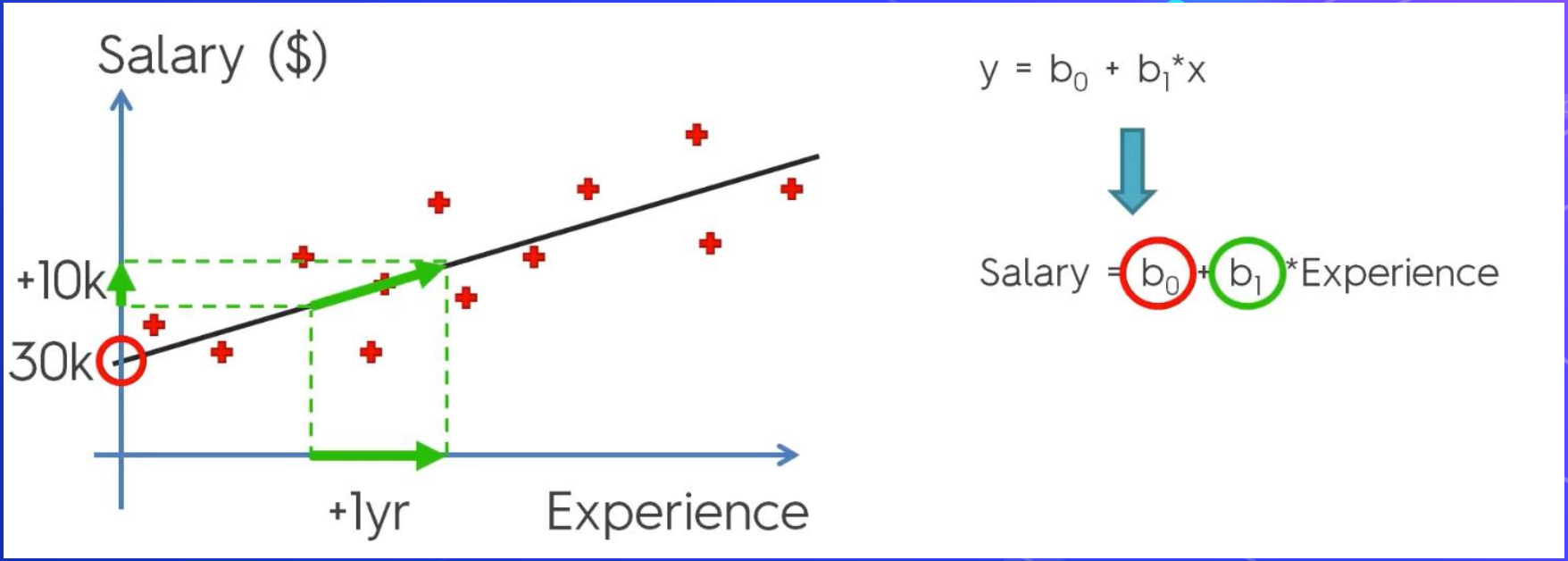
- Integrate Colab to google account
- Create new .ipynb file
- Work with UI
- Use cell for code & text
- Use cell for environment cmd
- Use custom uploaded files
- Use Google Drive
- WGET files
- Use GPU & TPU

Agenda

- What is Deep Learning
- Deep Learning Applications
- Google Colab
- **Perceptron**
- Artificial Neural Networks (ANN)
- Error or Loss or Cost functions
- Optimization algorithms
- Backpropagation
- Activation functions
- Improve neural networks training
- Deep Learning frameworks

Perceptron - Linear Regression Reminder

It also called ordinary least squares (OLS)

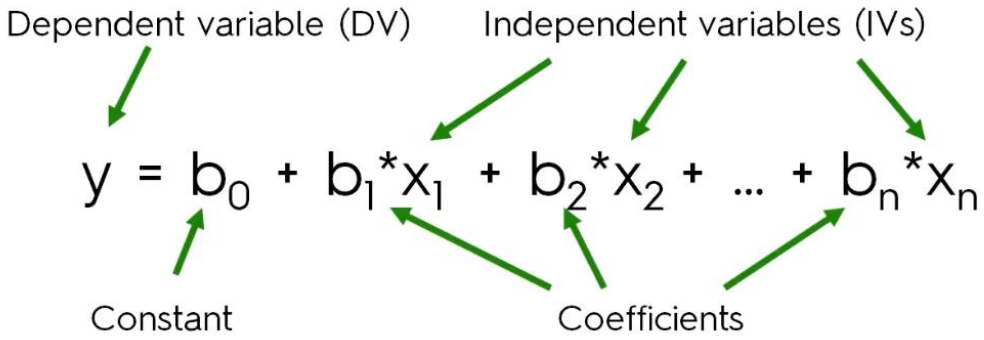


Perceptron - Linear Regression Reminder

Simple Linear Regression

$$y = b_0 + b_1 * x_1$$

Multiple Linear Regression



Perceptron - Logistic Regression Reminder

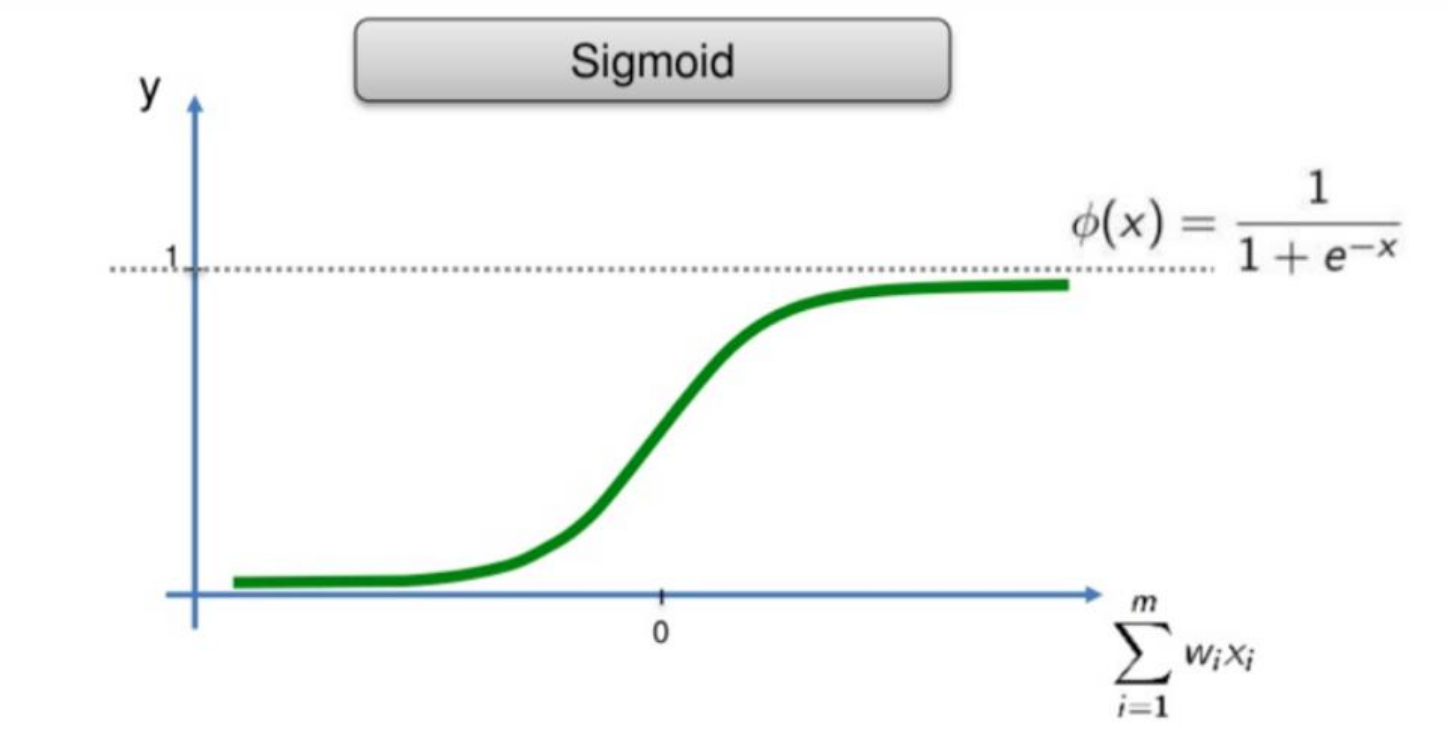
The Logistic Function

$$y = \frac{1}{1 + e^{-f(x_1, x_2, \ldots, x_n)}} \in (0, 1)$$

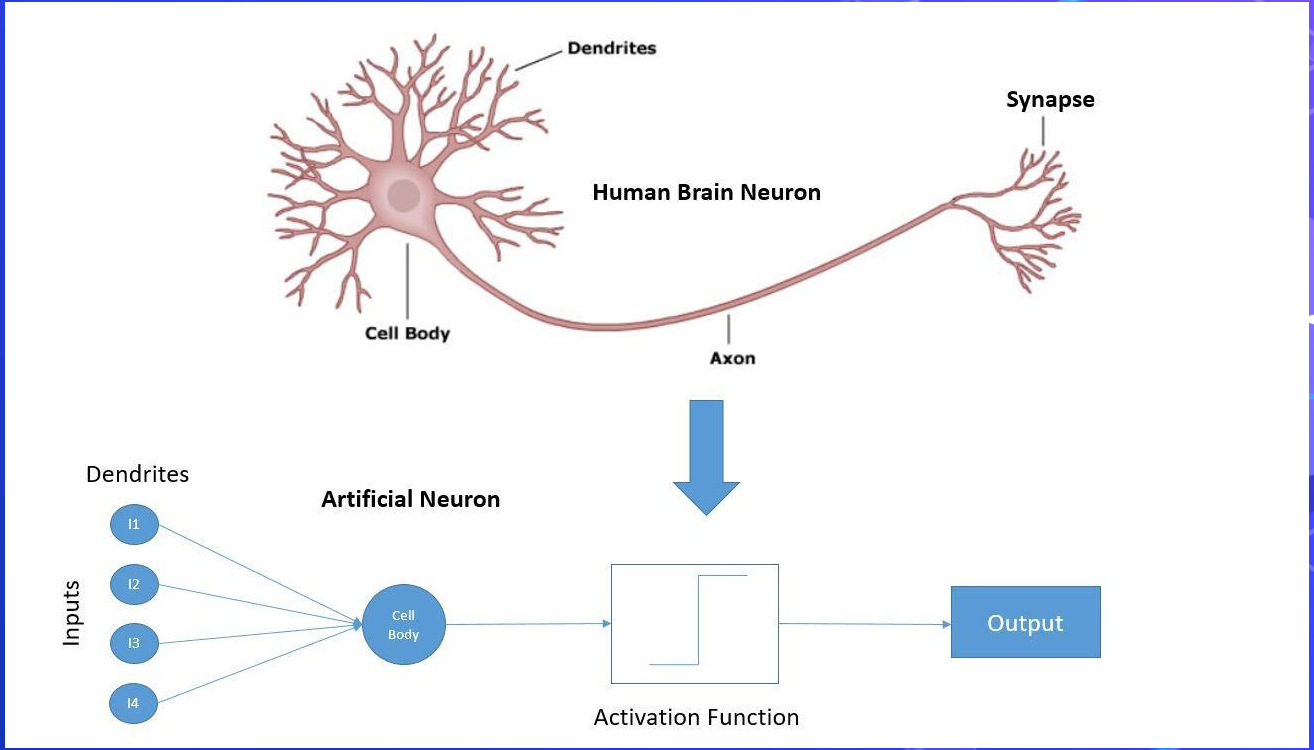
where

$$f(x_1, x_2, \ldots, x_n) = a_0 + a_1 x_1 + \ldots + a_n x_n \in (-\infty, +\infty)$$

Perceptron - Logistic Regression Reminder

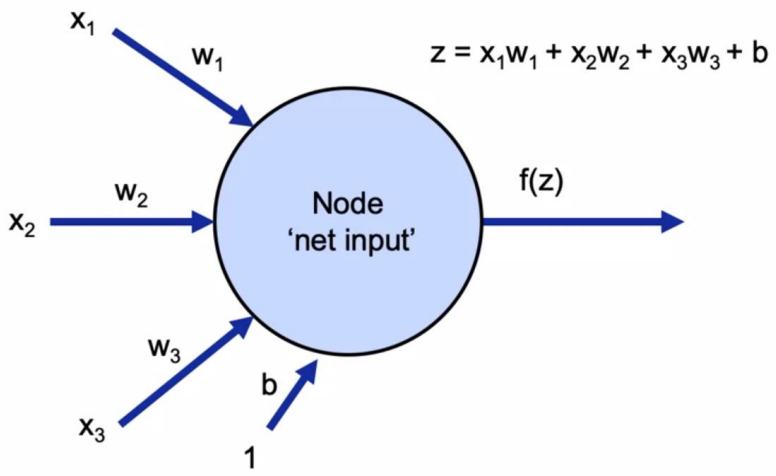


Perceptron (Neuron)



Perceptron (Neuron)

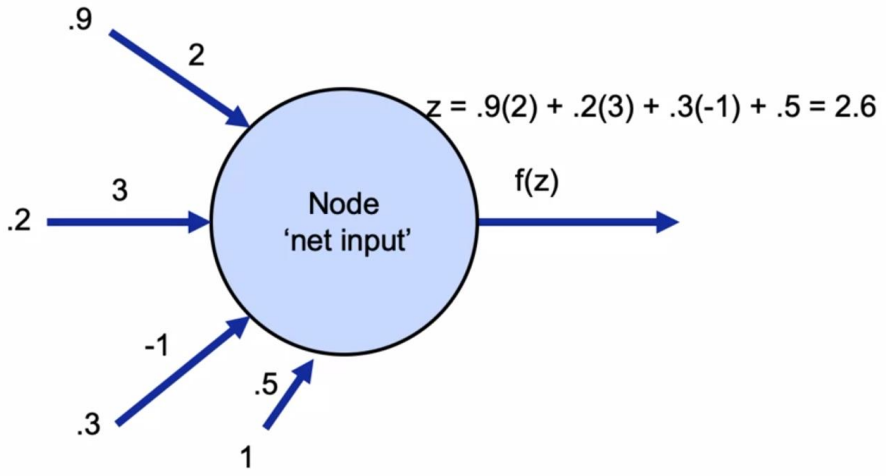
Example Neuron Computation



IBM

Perceptron (Neuron)

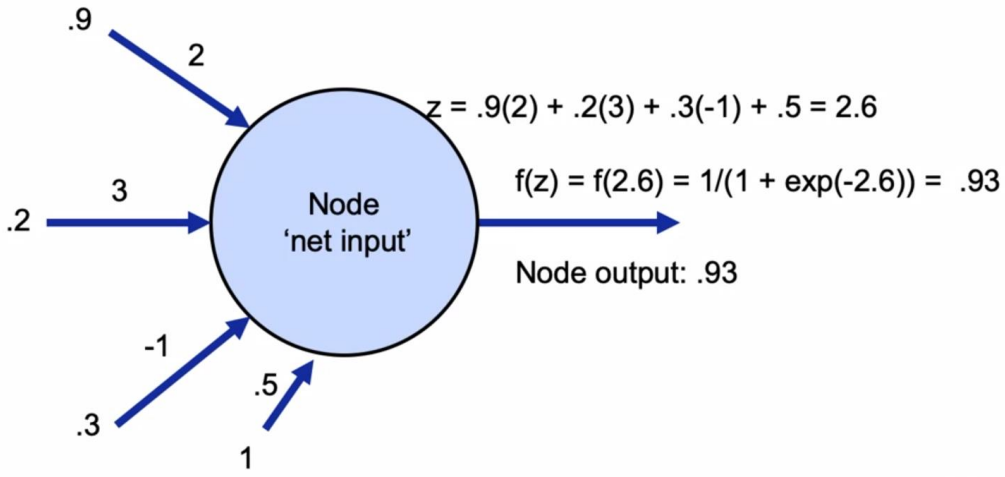
Example Neuron Computation



IBM

Perceptron (Neuron)

Example Neuron Computation

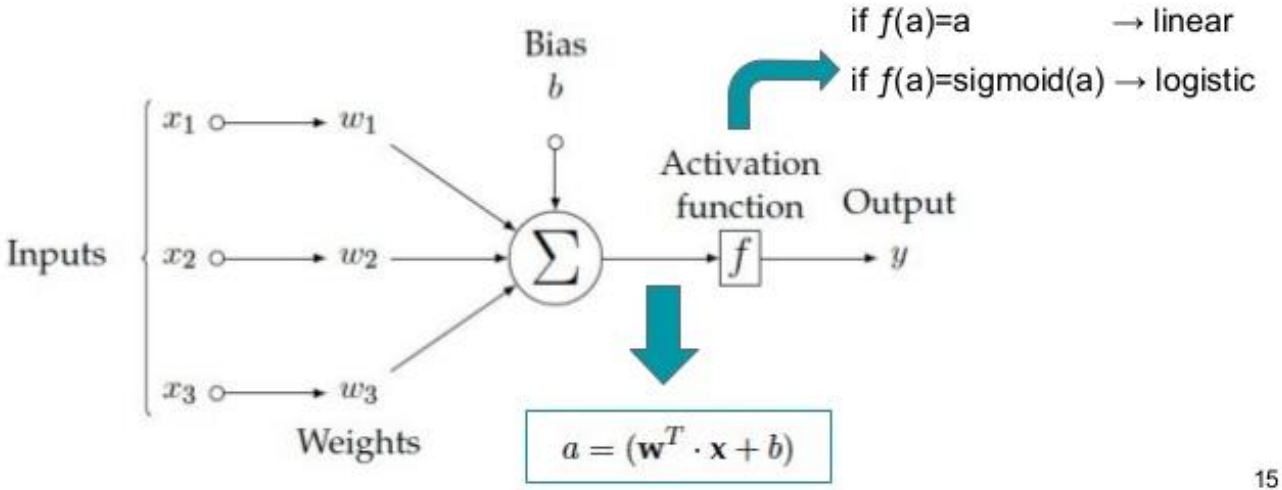


IBM

Perceptron (Neuron)

The Perceptron (Neuron)

The Perceptron can represent both linear & logistic regression:

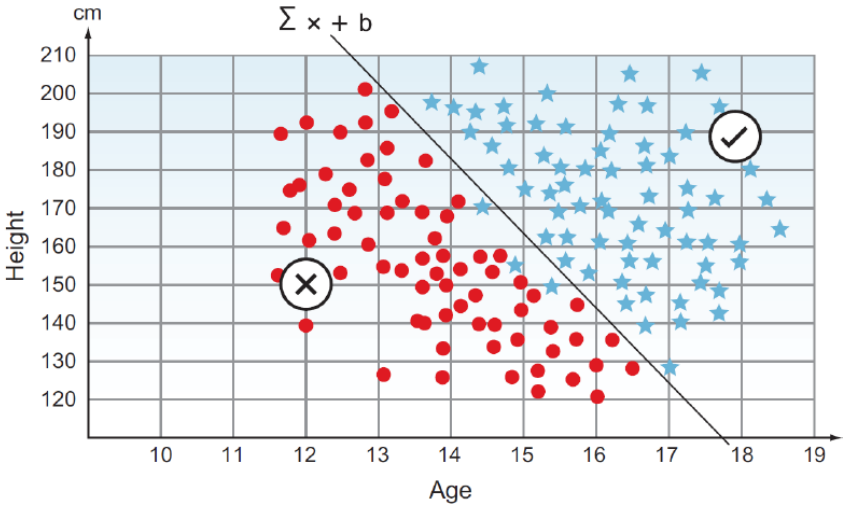


Agenda

- What is Deep Learning
- Deep Learning Applications
- Google Colab
- Perceptron
- **Artificial Neural Networks (ANN)**
- Error or Loss or Cost functions
- Optimization algorithms
- Backpropagation
- Activation functions
- Improve neural networks training
- Deep Learning frameworks

Artificial Neural Networks (ANN)

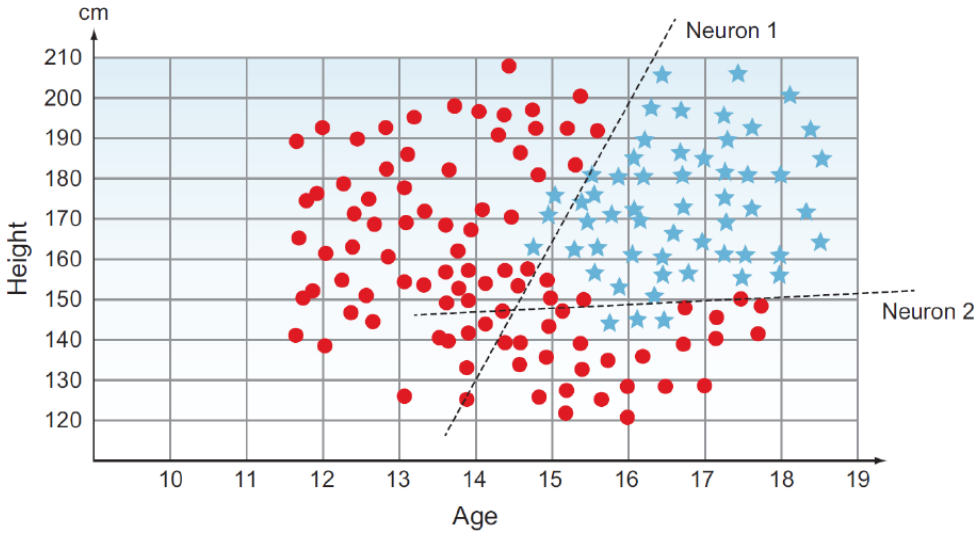
Suppose we want to train a perceptron to predict whether a player will be accepted into the college squad. We collect all the data from the previous years and train the perceptron to predict whether players will be accepted or not based on only two features (height and weight). The trained perceptron will find the best weights and bias values to produce the *straight line* that best separates the accepted from non-accepted (best fit). This line has this equation: $z = height \cdot w_1 + age \cdot w_2 + b$. After the training is complete on the training data, we can start using the this perceptron to predict with new players. When we get a player who is 150 cm height and 12 years old, we compute the above equation with the values (150, 12). When plotted in the graph, you can see that it falls below the line, then the neuron is predicting that this player will not be accepted. If it falls above the line, then the player will be accepted.



Artificial Neural Networks (ANN)

In the above example, the single perceptron works fine because our data was ***linearly separable***. Which means that the training data can be separated by a straight line. But life isn't always that simple. What happens when we have a more complex dataset that cannot be separated by a straight line (*non-linear dataset*)?

As you can see in the figure below, a single straight line will not separate our training data. We say that: "it does not fit our data". We need a more complex network for more complex data like this. What if we built a network with two perceptrons? This will produce two lines. Would that help us separate the data better?



Artificial Neural Networks (ANN)

Sometimes it's called
Multi Layer Perceptron (MLP)

https://playground.tensorflow.org

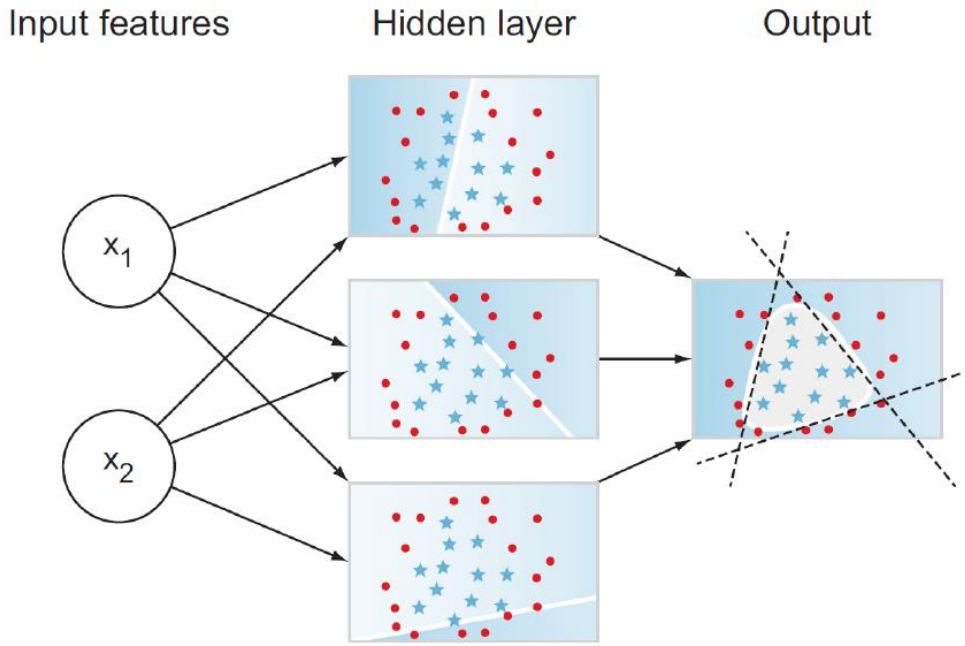
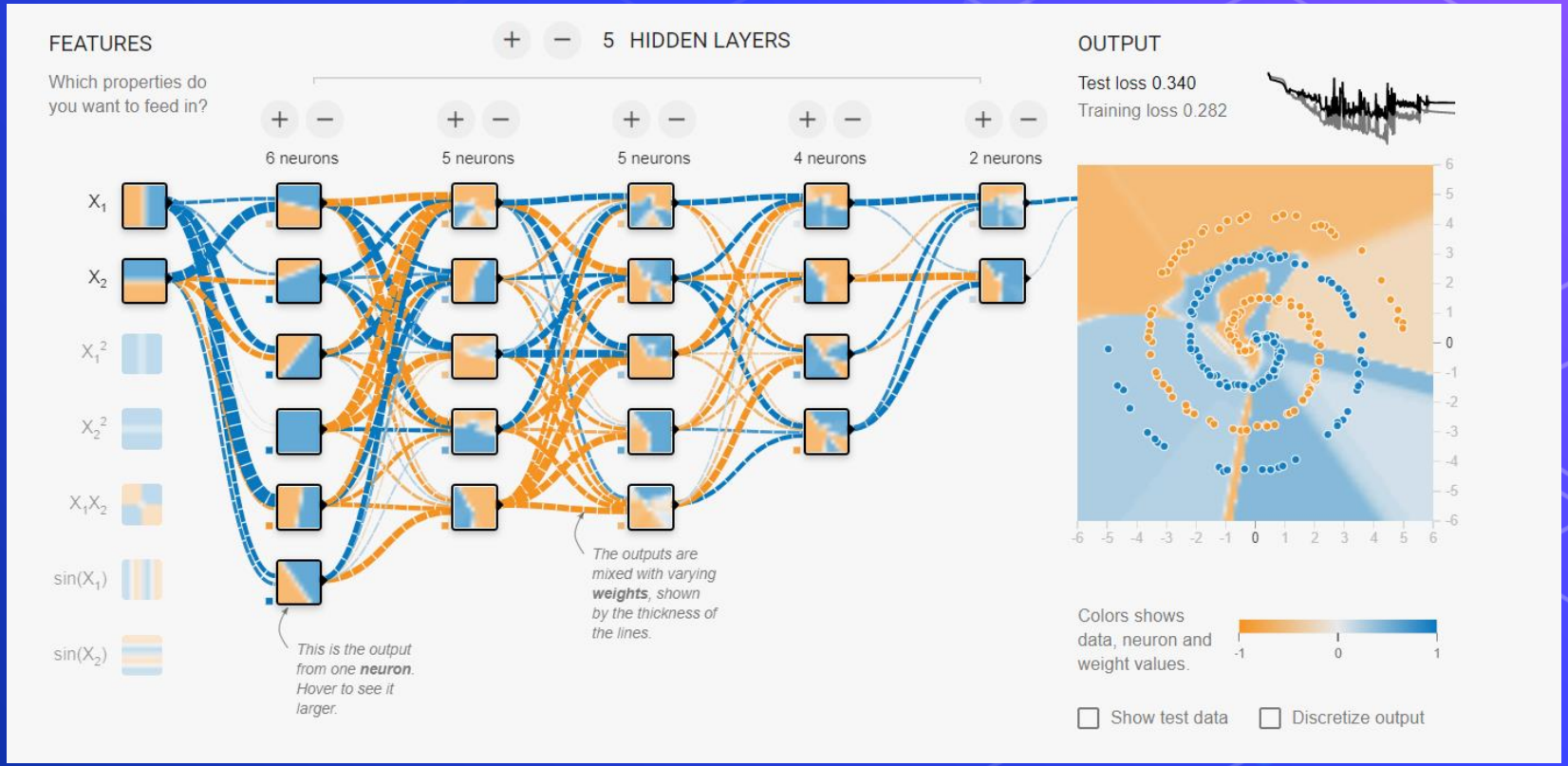
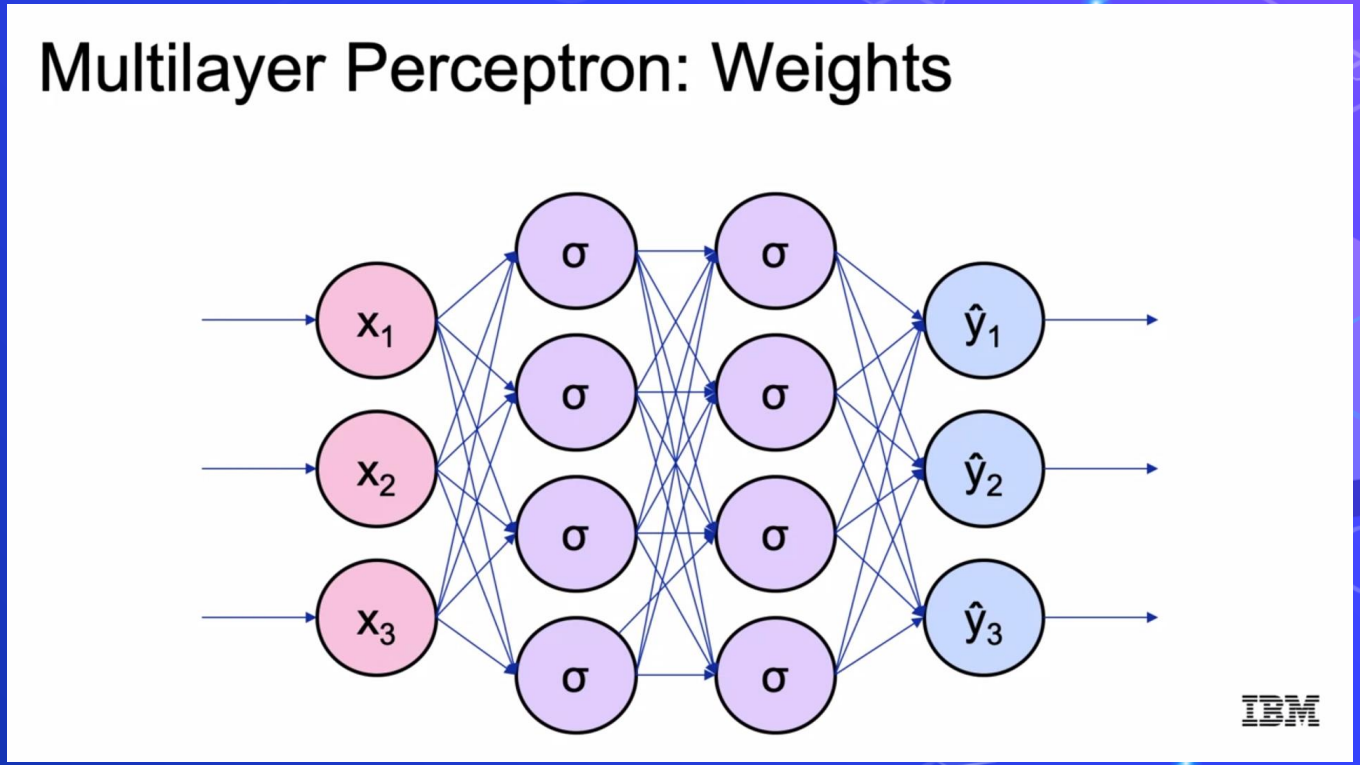


Figure 2.10

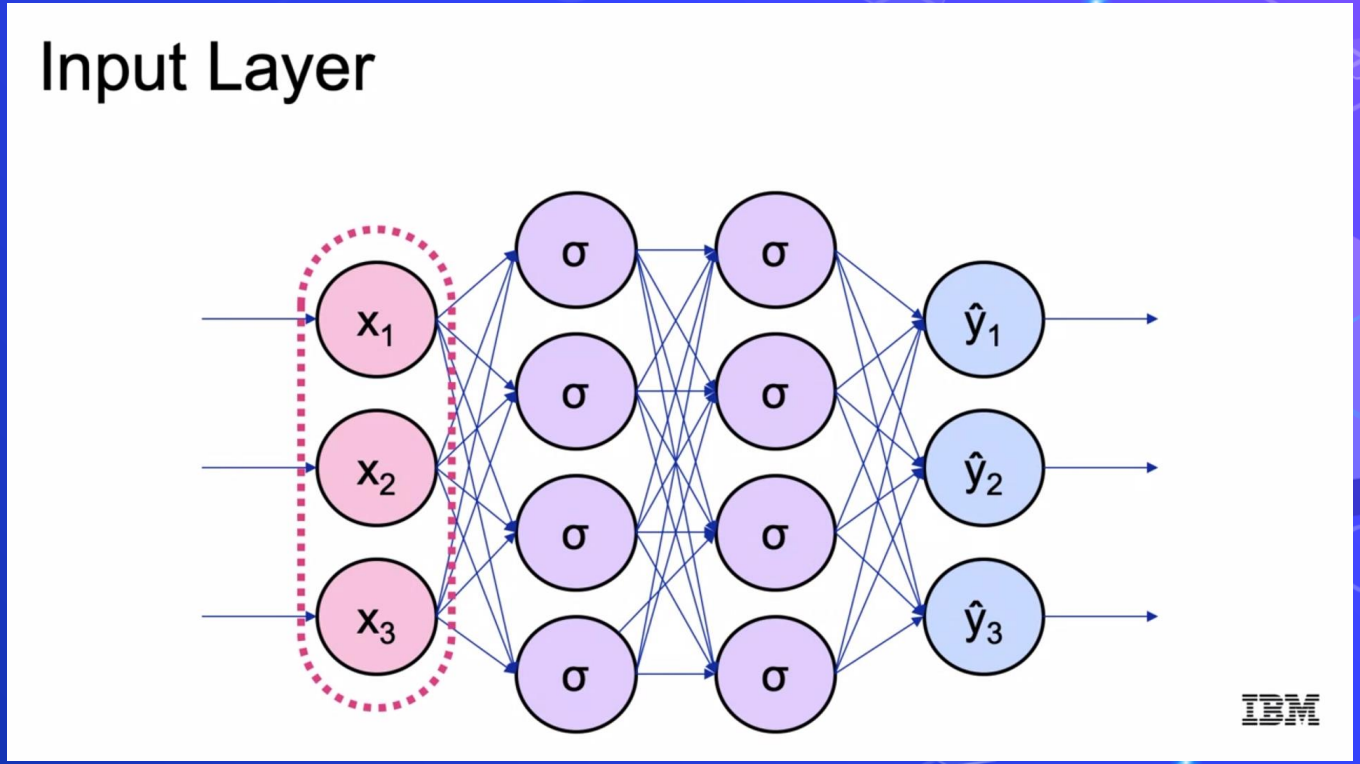
Artificial Neural Networks (ANN)



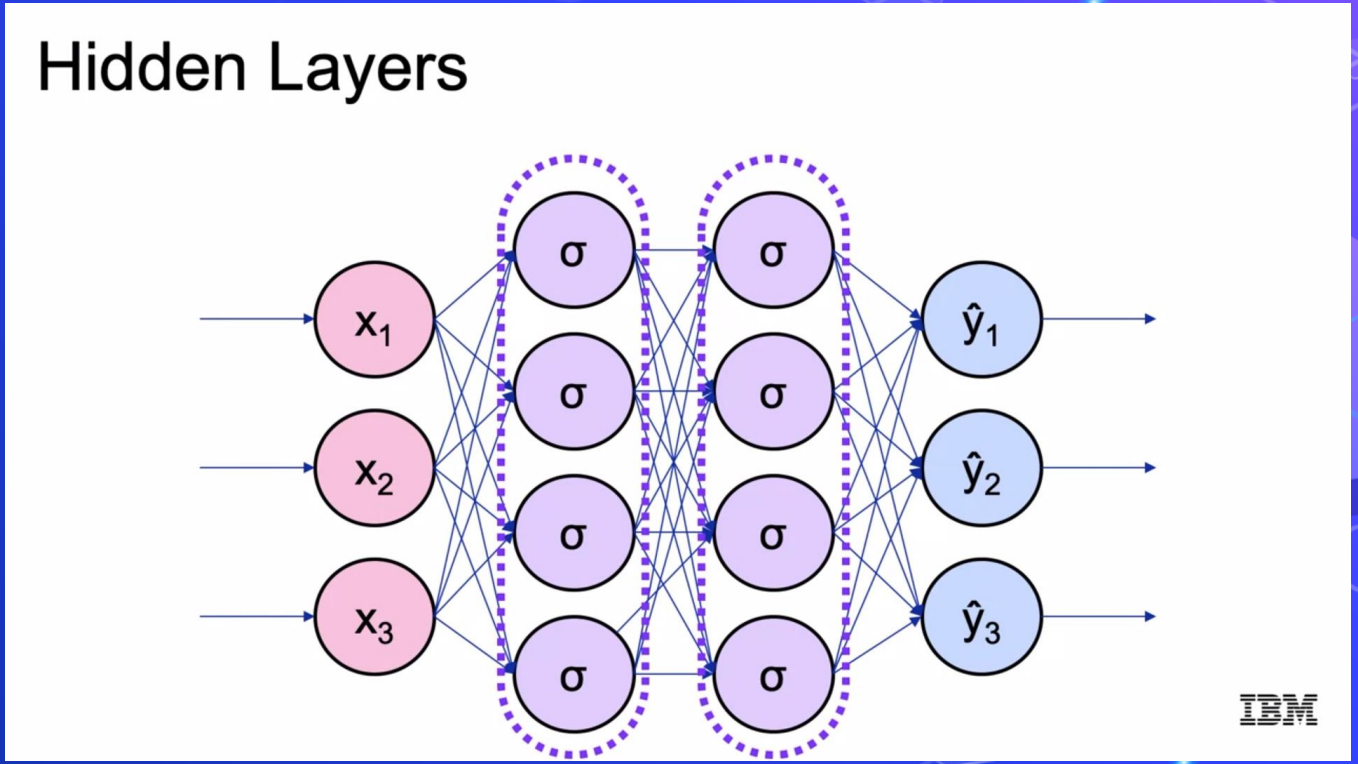
Artificial Neural Networks (ANN)



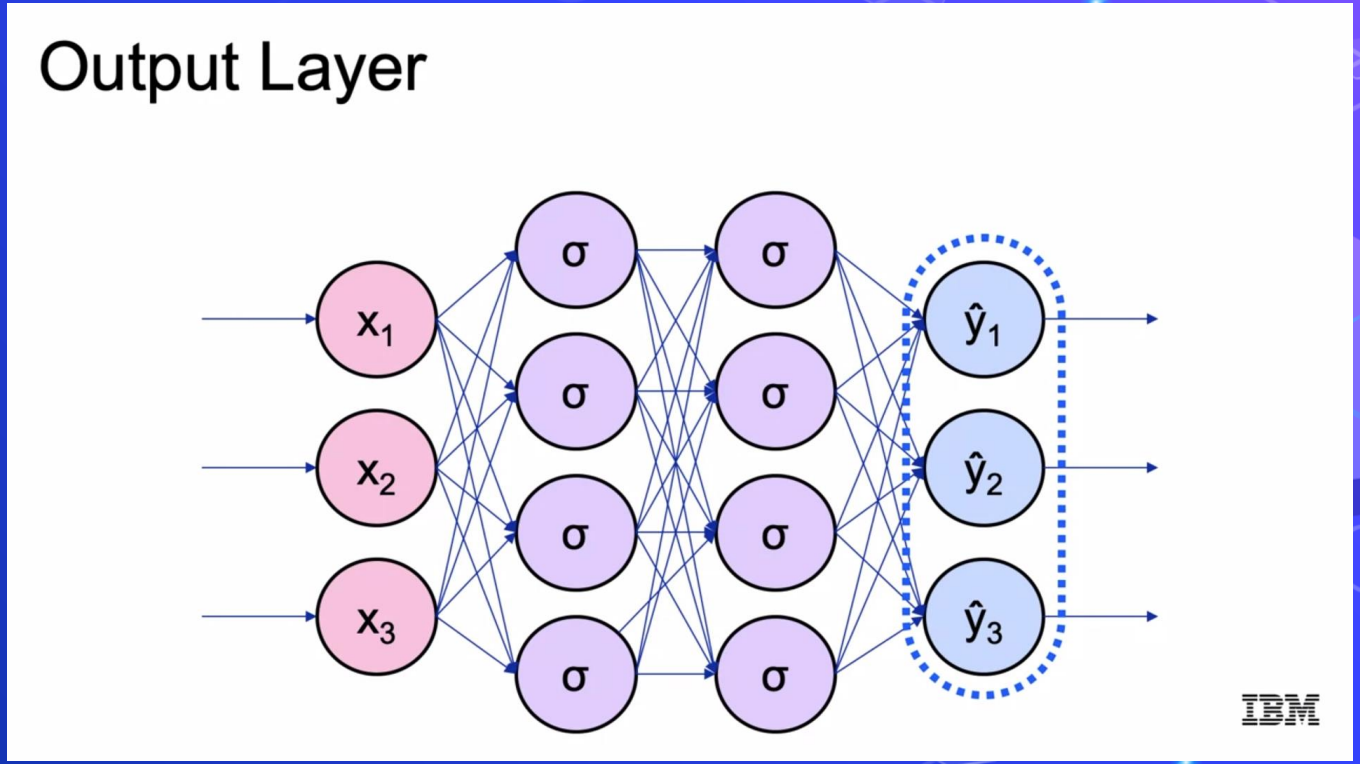
Artificial Neural Networks (ANN)



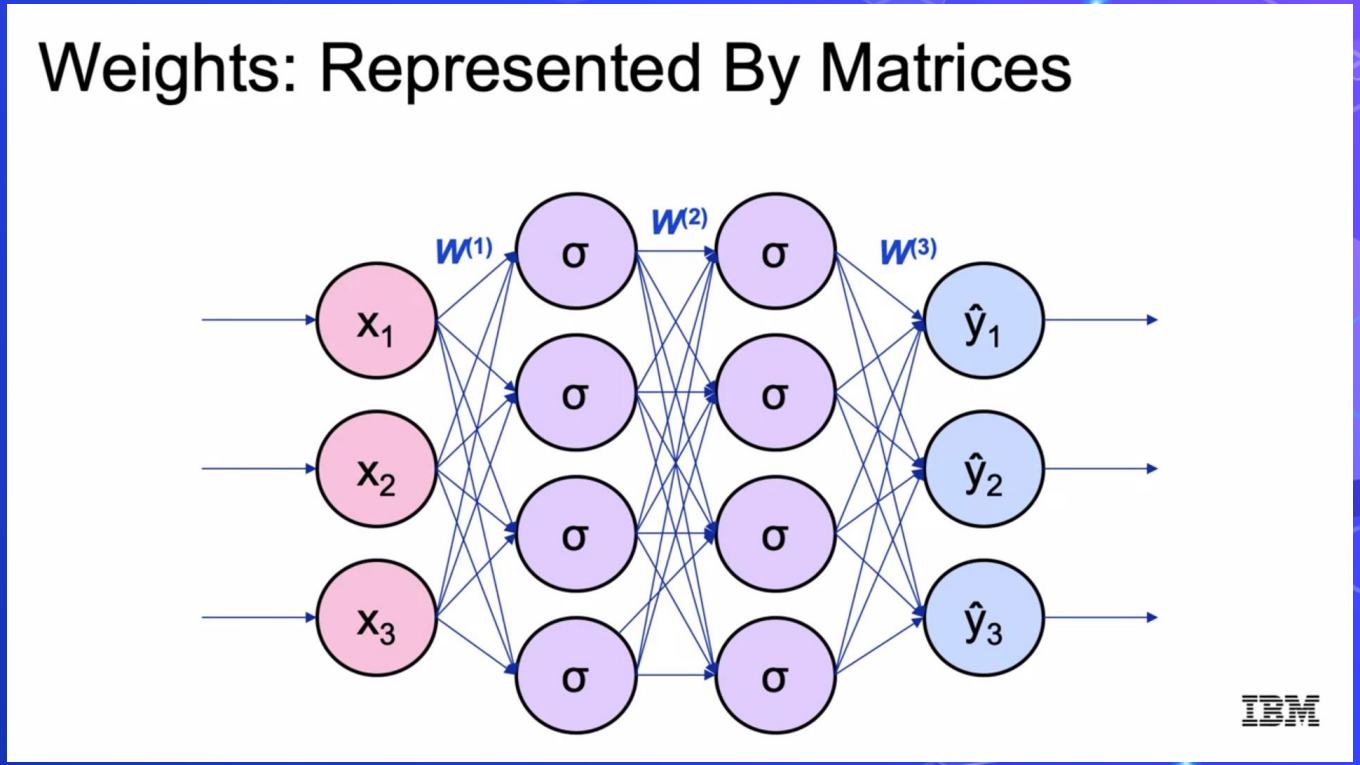
Artificial Neural Networks (ANN)



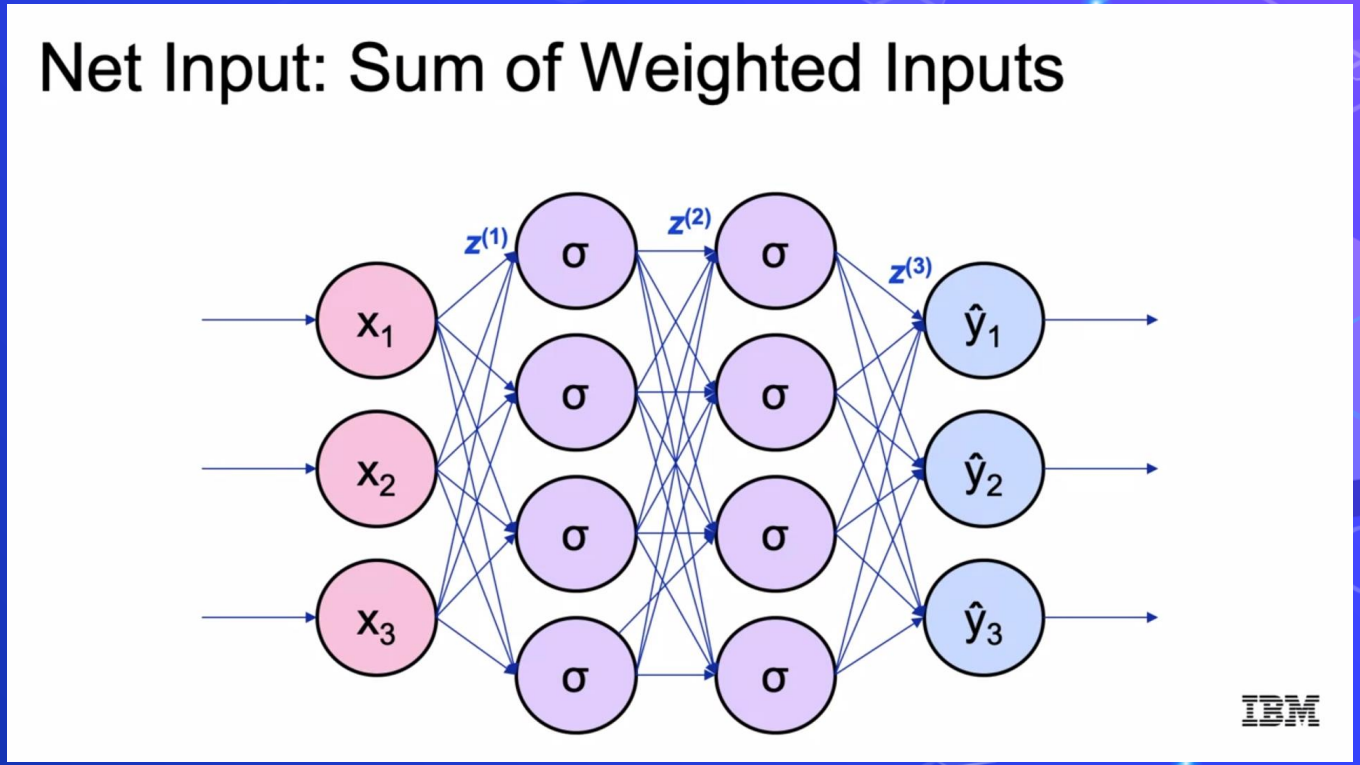
Artificial Neural Networks (ANN)



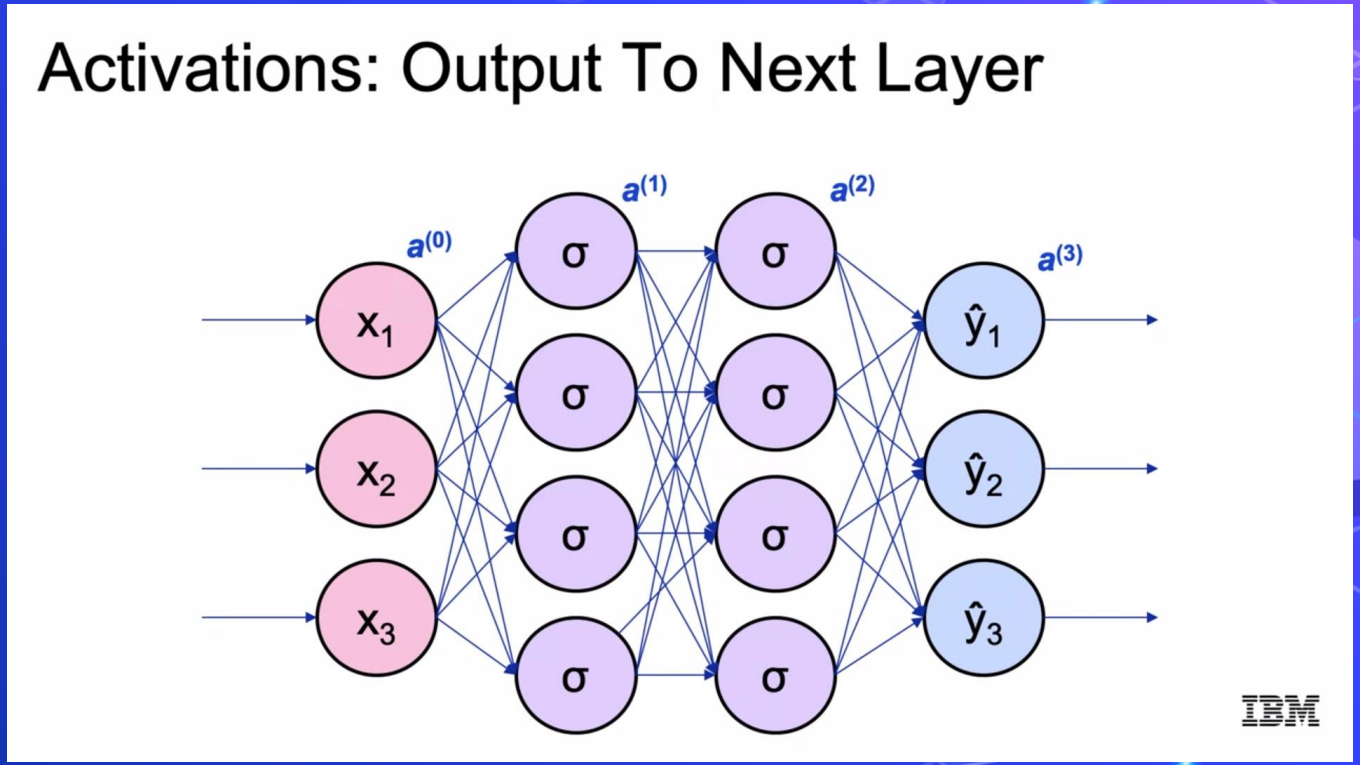
Artificial Neural Networks (ANN)



Artificial Neural Networks (ANN)



Artificial Neural Networks (ANN)



Artificial Neural Networks (ANN)

Matrix Representation of Computation

For a single data point (instance):

$W^{(1)}$ is a 3x4 matrix

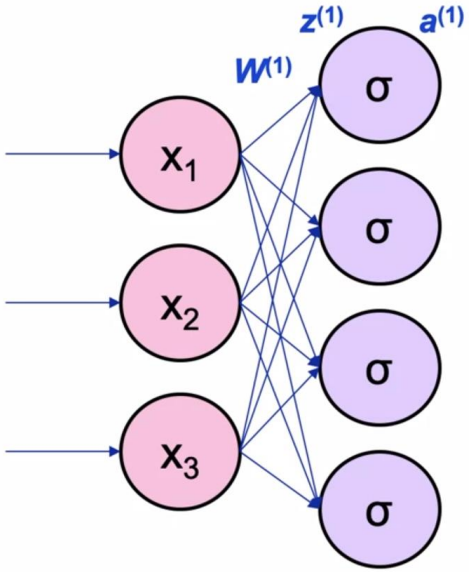
$z^{(1)}$ is a 4-vector

$a^{(1)}$ is a 4-vector

x $(x = a^{(0)})$

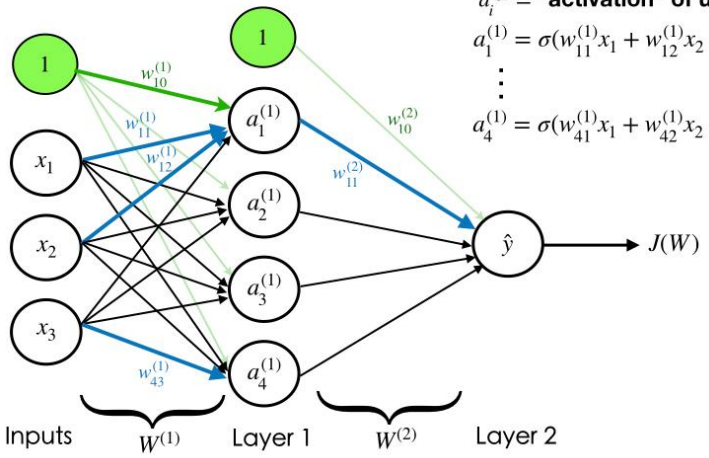
$z^{(1)} = xW^{(1)}$

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



IBM

Artificial Neural Networks (ANN) – forward propagation



$a_i^{(j)}$ = "activation" of unit i in layer j

$$a_1^{(1)} = \sigma(w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 + w_{13}^{(1)}x_3 + w_{10}^{(1)})$$

$$\vdots$$

$$a_4^{(1)} = \sigma(w_{41}^{(1)}x_1 + w_{42}^{(1)}x_2 + w_{43}^{(1)}x_3 + w_{40}^{(1)})$$

$$\hat{y} = a_1^{(2)} = \sigma(w_{11}^{(2)}a_1^{(1)} + w_{12}^{(2)}a_2^{(1)} + w_{13}^{(2)}a_3^{(1)} + w_{14}^{(2)}a_4^{(1)} + w_{10}^{(2)})$$

Artificial Neural Networks (ANN) – forward propagation

Continuing the Computation

For a single training instance (data point):

Input: vector x (a row vector of length 3)

Output: vector ŷ (a row vector of length 3)

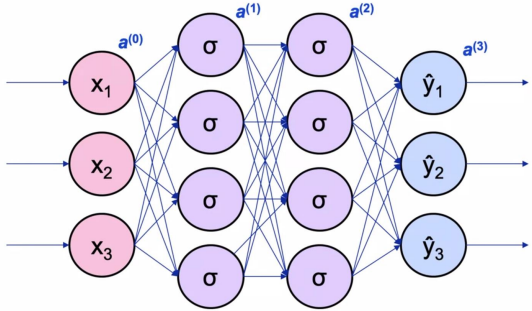
$$z^{(1)} = xW^{(1)} \longrightarrow a^{(1)} = \sigma(z^{(1)})$$

$$z^{(2)} = a^{(1)}W^{(2)} \longrightarrow a^{(2)} = \sigma(z^{(2)})$$

$$z^{(3)} = a^{(2)}W^{(3)} \longrightarrow \hat{y} = softmax(z^{(3)})$$

IBM

Activations: Output To Next Layer



IBM

Agenda

- What is Deep Learning
- Deep Learning Applications
- Google Colab
- Perceptron
- Artificial Neural Networks (ANN)
- **Error or Loss or Cost functions**
- Optimization algorithms
- Backpropagation
- Activation functions
- Improve neural networks training
- Deep Learning frameworks

Error or Loss or Cost functions

- Mean Square Error (MSE)
- Binary Cross Entropy
- Categorical Cross Entropy

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

- Used for **Regression.**
- Also Called L2-Norm.
- Output layer should use no activation function or linear activation function.

Error or Loss or Cost functions

- Mean Square Error (MSE)
- Binary Cross Entropy
- Categorical Cross Entropy

$$Loss = -\frac{1}{N}\sum_{i=1}^{N}\{y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)\}$$

- Used for **Binary & Multi Label Classification.**
- For Binary Classification output y must be a single vector has value 1 or 0.
- For Multi Label Classification output y must be One-Hot-Encoded vector.
- Output layer should use **Sigmoid** activation function.

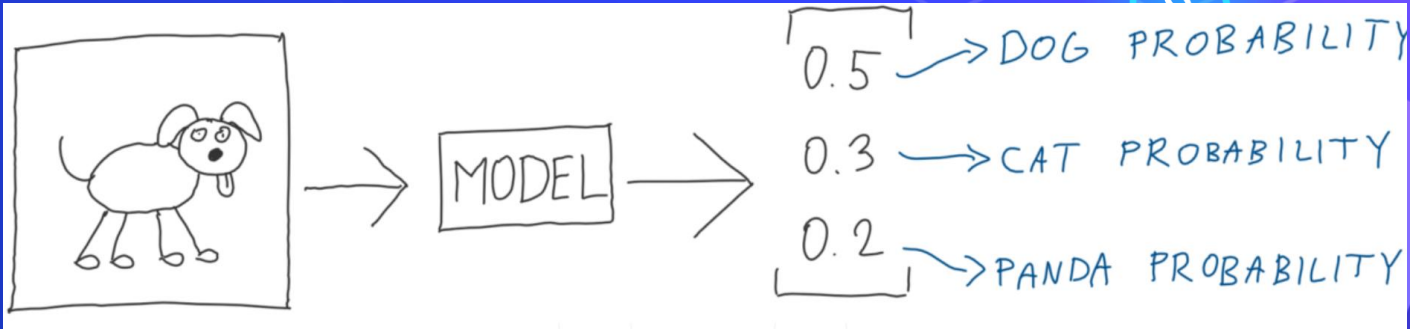
Error or Loss or Cost functions

- Mean Square Error (MSE)
- Binary Cross Entropy
- Categorical Cross Entropy

$$CCE = -\frac{1}{N}\sum_{i=0}^{N}\sum_{j=0}^{J} y_j \cdot log(\hat{y}_j) + (1 - y_j) \cdot log(1 - \hat{y}_j)$$

- Used for **Multi Class Classification**
- Output y must be One-Hot-Encoded vector.
- Output layer should use **Softmax** activation function.

Error or Loss or Cost functions



Example:
https://towardsdatascience.com/cross-entropy-for-classification-d98e7f974451

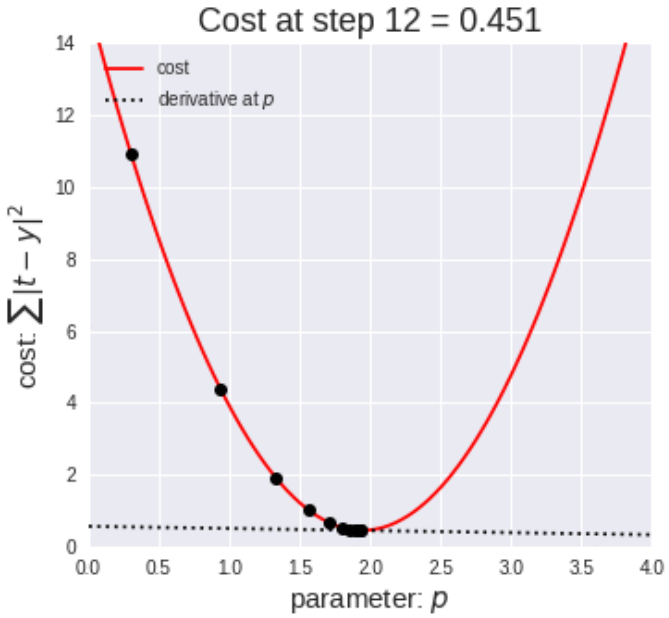
Agenda

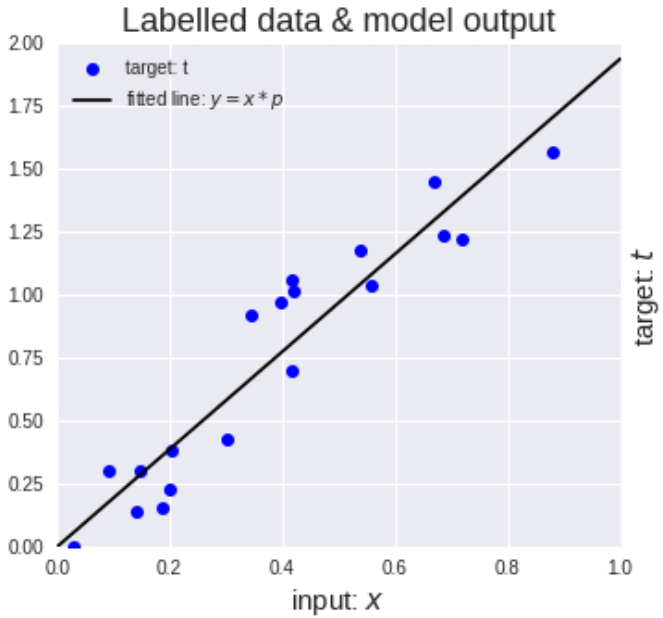
- What is Deep Learning
- Deep Learning Applications
- Google Colab
- Perceptron
- Artificial Neural Networks (ANN)
- Error or Loss or Cost functions
- **Optimization algorithms**
- Backpropagation
- Activation functions
- Improve neural networks training
- Deep Learning frameworks

Optimization algorithms

- **Gradient Descent**
- SGD with Momentum
- RMSProp
- Adam

Optimization algorithms (Gradient Descent)





Optimization algorithms (Gradient Descent)

Full Batch Gradient Descent

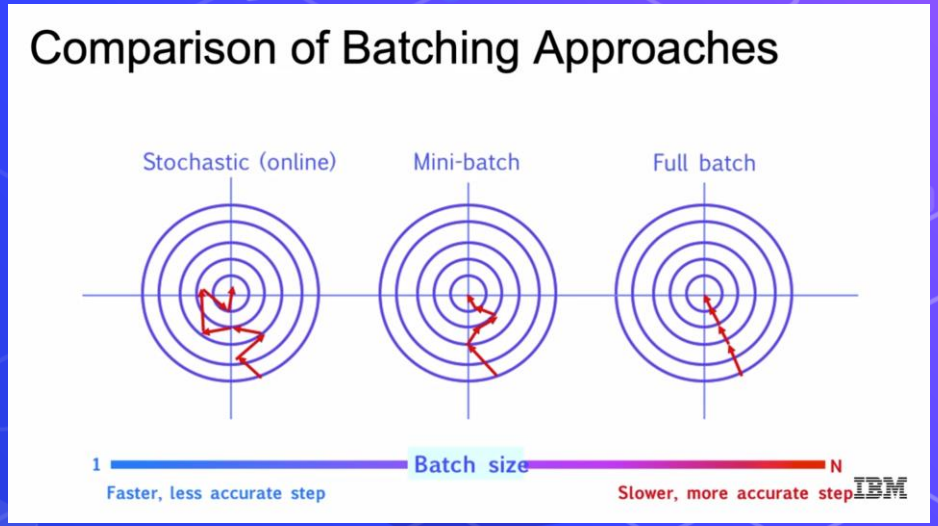
For an **Epoch** take the whole data and calculate the average loss then update weights, it's memory intensive but gives stable accurate accuracy. Takes 1 iteration.

Stochastic Gradient Descent

For an **Epoch** take a random data point from dataset and calculate its loss then update the weights and repeat over the rest of the data until the epoch is done, it's causes a stochastic conversion but uses less memory. Takes N iterations.

Mini-Batch Gradient Descent

For an **Epoch** take a batch of data points could be 32 or 64 etc from dataset and calculate average loss on the batch then update the weights and repeat over the rest of the batches until the epoch is done, it's very good compared to the previous methods. Takes Number of Batches iterations.



Role of thumb

If you have small data < 2K use Full Batch.
If you have more data use Mini-Batch.

Optimization algorithms (Gradient Descent)

Full Batch Gradient Descent

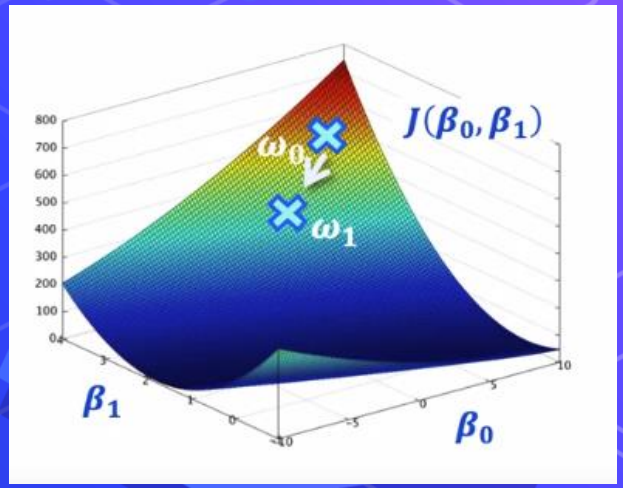
$$\omega_1 = \omega_0 - \alpha\nabla\frac{1}{2}\sum_{i=1}^{m}\left(\left(\beta_0 + \beta_1 x_{obs}^{(i)}\right) - y_{obs}^{(i)}\right)^2$$

Stochastic Gradient Descent

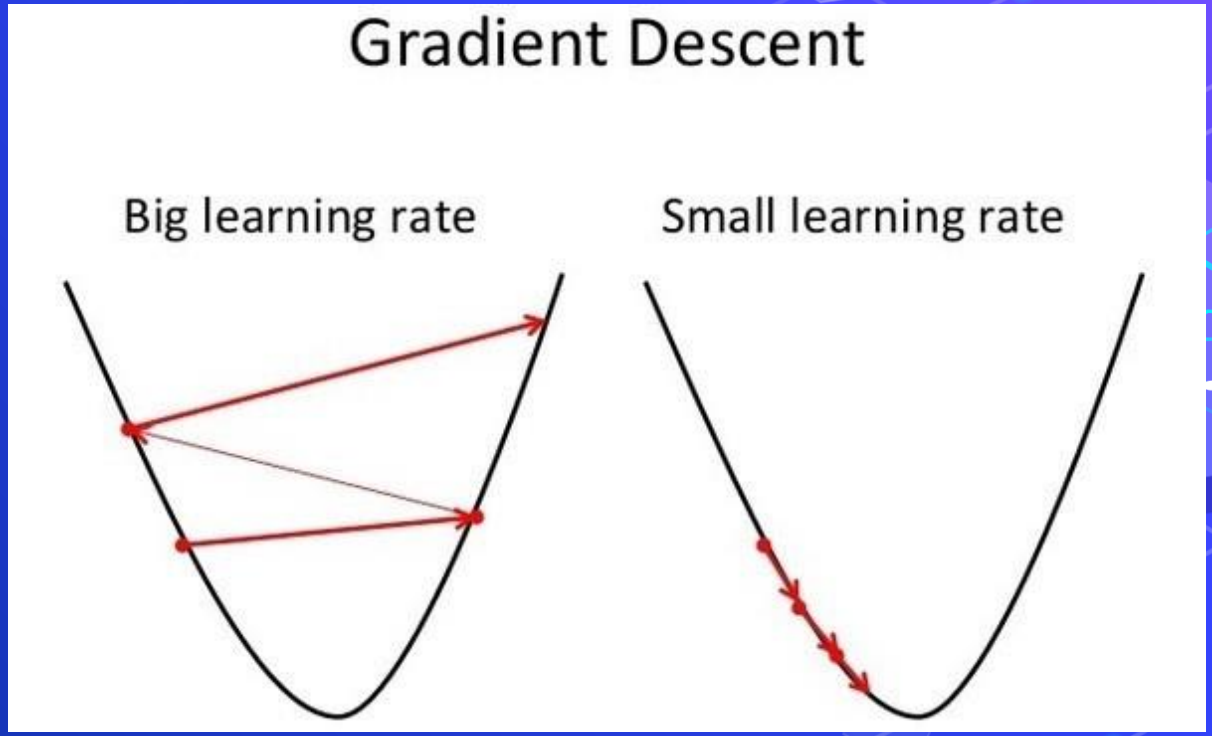
$$\omega_1 = \omega_0 - \alpha\nabla\frac{1}{2}\left(\left(\beta_0 + \beta_1 x_{obs}^{(0)}\right) - y_{obs}^{(0)}\right)^2$$

Mini-Batch Gradient Descent

$$\omega_1 = \omega_0 - \alpha\nabla\frac{1}{2}\sum_{i=1}^{n}\left(\left(\beta_0 + \beta_1 x_{obs}^{(i)}\right) - y_{obs}^{(i)}\right)^2$$



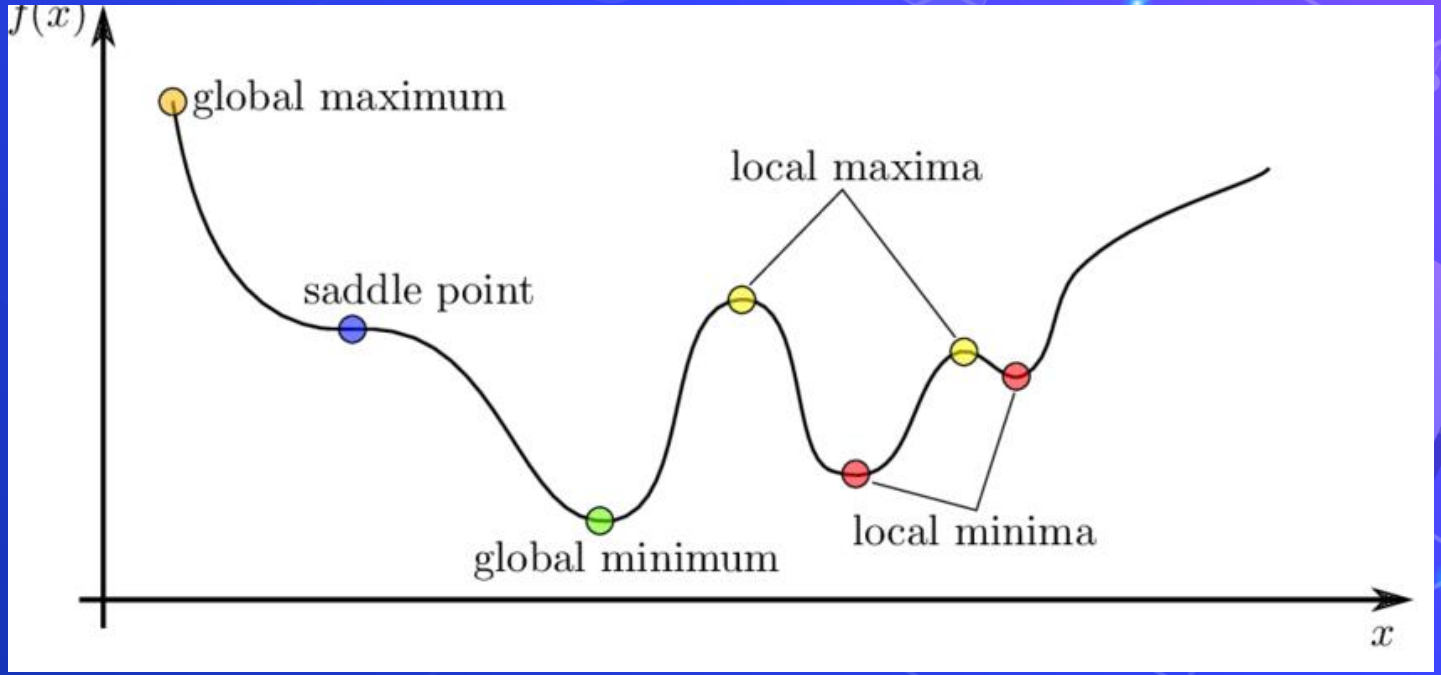
Optimization algorithms (Gradient Descent)



Optimization algorithms

- Gradient Descent
- **SGD with Momentum**
- RMSProp
- Adam

SGD with Momentum



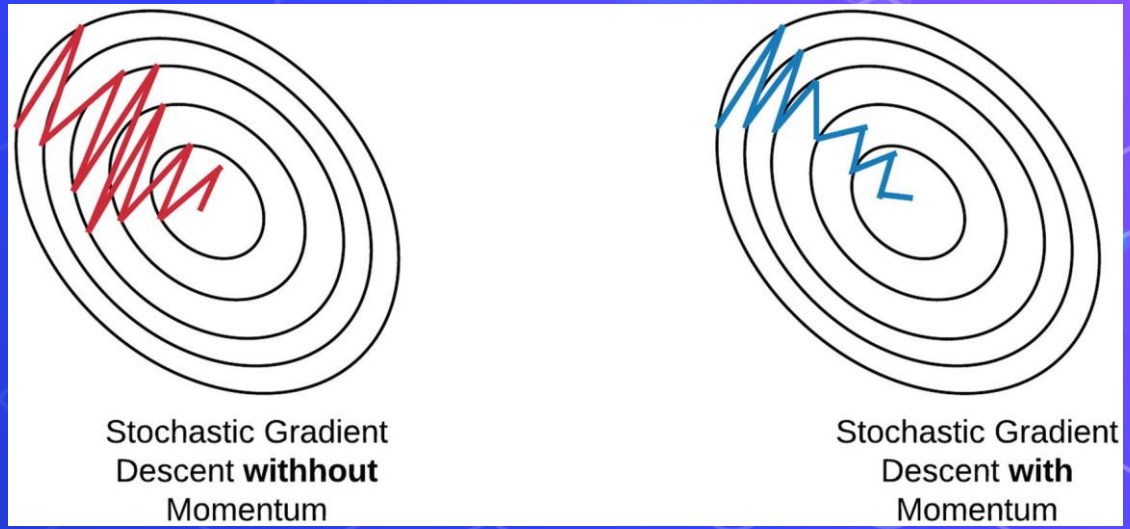
SGD with Momentum

Momentum was invented for reducing high variance in SGD and softens the convergence (reaching the global minima). It accelerates the convergence towards the relevant direction and reduces the fluctuation to the irrelevant direction by adding the speed parameter, one disadvantage it may overshoot the local minima if Learning rate was too high.

We can choose Learning rate = 0.01, beta = 0.9
And v starts with 0.

$$v_{dw} = \beta \cdot v_{dw} + (1 - \beta) \cdot dw$$

$$W = W - \alpha \cdot v_{dw}$$



Optimization algorithms

- Gradient Descent
- SGD with Momentum
- **RMSProp**
- Adam

RMSProp (Root Mean Square Propagation)

RMSProp solves the overshooting problem and modify its path to the global minima.

Learning rate = 0.01, beta = 0.9, v starts with 0 and epsilon = 0.000001

$$v_{dw} = \beta \cdot v_{dw} + (1 - \beta) \cdot dw^2$$

$$W = W - \alpha \cdot \frac{dw}{\sqrt{v_{dw}} + \epsilon}$$

Optimization algorithms

- Gradient Descent
- SGD with Momentum
- RMSProp
- **Adam**

Adam

Adam combines the Momentum and RMSProp, and it considered the best choice to use.
Beta_1 = 0.9, beta_2 = 0.999 & LR = 0.0001

- $m_t = \beta_1 m_t + (1 - \beta_1) g_t$
- $v_t = \beta_1 m_t + (1 - \beta_1) g_t^2$
- $\widehat{m_t} = \frac{m_t}{1 - \beta_1^t}$

To correct for bias of averaging in a 0 at the beginning.

- $\widehat{v_t} = \frac{v_t}{1 - \beta_1^t}$

This is like a combination of RMSprop and momentum.

- $w_{t+1} = w_t - \frac{\eta}{\sqrt{\widehat{v_t}}} \widehat{m_t}$

Agenda

- What is Deep Learning
- Deep Learning Applications
- Google Colab
- Perceptron
- Artificial Neural Networks (ANN)
- Error or Loss or Cost functions
- Optimization algorithms
- **Backpropagation**
- Activation functions
- Improve neural networks training
- Deep Learning frameworks

Backpropagation (Chain Rule Formula)

In calculus, the chain rule is a formula to compute the derivative of a composite function.

$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx}$$

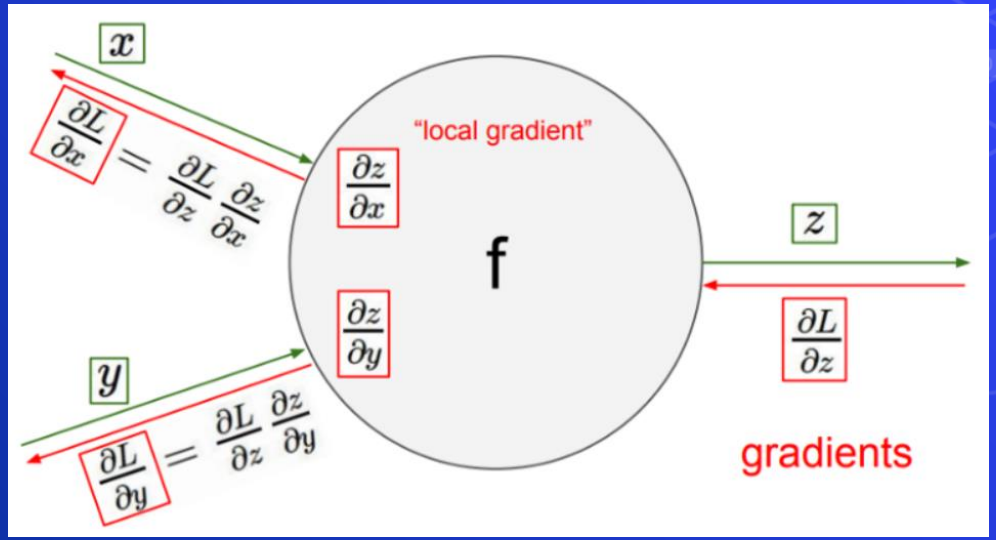
$\frac{dy}{dx}$ = derivative of y with respect to x

$\frac{dy}{du}$ = derivative of y with respect to u

$\frac{du}{dx}$ = derivative of u with respect to x

Backpropagation (Chain Rule Formula)

So using chain rule we can calculate gradients of the error
With respect to the weights that is a value of another function.



Backpropagation (Algorithm)

Backpropagation Algorithm

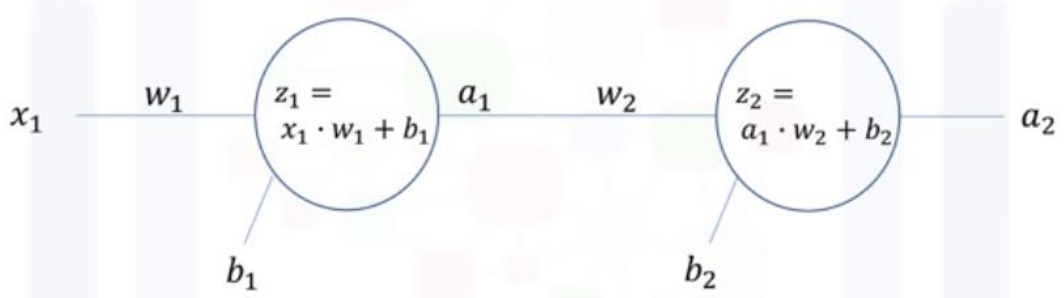
1. Calculate the error between the ground truth and the estimated output. Let's denote the error by E.
2. Propagate the error back into the network and update each weight and bias as per the following equations:

$$w_i \rightarrow w_i - \eta \cdot \frac{\partial E}{\partial w_i}$$

$$b_i \rightarrow b_i - \eta \cdot \frac{\partial E}{\partial b_i}$$

Backpropagation (Example of one epoch)

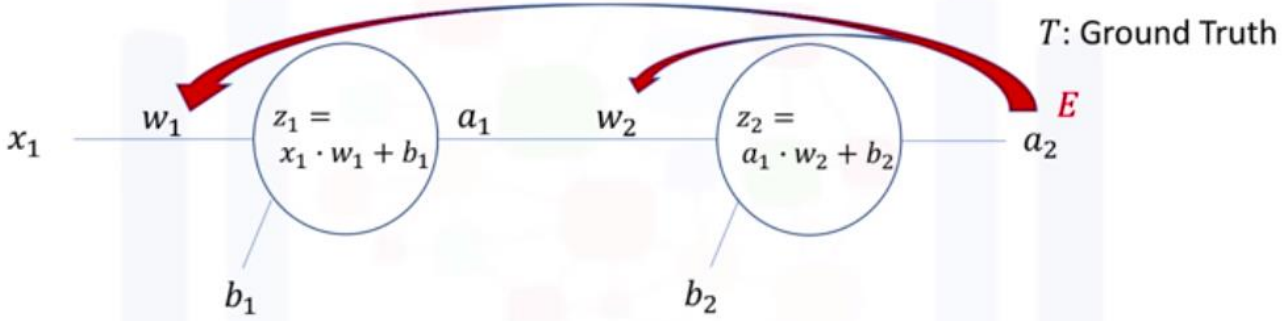
Backpropagation



T: Ground Truth

Backpropagation (Example of one epoch)

Backpropagation Algorithm



1. Calculate the error: $E = \frac{1}{2}(T - a_2)^2$

$$E = \frac{1}{2m}\sum_{i=1}^{m}(T_i - a_{2,i})^2$$

2. Update w_2, b_2, w_1, and b_1

Backpropagation (Example of one epoch)

Updating w_2

$$w_2 \rightarrow w_2 - \eta \cdot \frac{\partial E}{\partial w_2}$$

$$E = \frac{1}{2}(T - a_2)^2 \longrightarrow \frac{\partial E}{\partial a_2}$$

$$a_2 = f(z_2) = \frac{1}{1 + e^{-z_2}} \longrightarrow \frac{\partial a_2}{\partial z_2}$$

$$z_2 = a_1 \cdot w_2 + b_2 \longrightarrow \frac{\partial z_2}{\partial w_2}$$

Backpropagation (Example of one epoch)

Updating w_2

$$w_2 \rightarrow w_2 - \eta \cdot \frac{\partial E}{\partial w_2}$$

$$\frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_2}$$

$$= \left(-(T - a_2)\right) \cdot \left(a_2(1 - a_2)\right) \cdot (a_1)$$

$$w_2 \rightarrow w_2 - \eta \cdot \left(-(T - a_2)\right) \cdot \left(a_2(1 - a_2)\right) \cdot (a_1)$$

Updating b_2

$$b_2 \rightarrow b_2 - \eta \cdot \frac{\partial E}{\partial b_2}$$

$$\frac{\partial E}{\partial b_2} = \frac{\partial E}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial b_2}$$

$$= \left(-(T - a_2)\right) \cdot \left(a_2(1 - a_2)\right) \cdot 1$$

Backpropagation (Example of one epoch)

Updating w_1

$$w_1 \rightarrow w_1 - \eta \cdot \frac{\partial E}{\partial w_1}$$

$$E = \frac{1}{2}(T - a_2)^2 \longrightarrow \frac{\partial E}{\partial a_2}$$

$$a_1 = f(z_1) = \frac{1}{1+e^{-z_1}} \longrightarrow \frac{\partial a_1}{\partial z_1}$$

$$a_2 = f(z_2) = \frac{1}{1+e^{-z_2}} \longrightarrow \frac{\partial a_2}{\partial z_2}$$

$$z_1 = x_1 \cdot w_1 + b_1 \longrightarrow \frac{\partial z_1}{\partial w_1}$$

$$z_2 = a_1 \cdot w_2 + b_2 \longrightarrow \frac{\partial z_2}{\partial a_1}$$

Backpropagation (Example of one epoch)

Updating w_1

$$w_1 \rightarrow w_1 - \eta \cdot \frac{\partial E}{\partial w_1}$$

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial a_1} \cdot \frac{\partial a_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1}$$

$$= \big(-(T - a_2)\big) \cdot \big(a_2(1 - a_2)\big) \cdot (w_2) \cdot a_1(1 - a_1) \cdot x_1$$

$$w_1 \rightarrow w_1 - \eta \cdot \big(-(T - a_2)\big) \cdot \big(a_2(1 - a_2)\big) \cdot (w_2) \cdot a_1(1 - a_1) \cdot x_1$$

Updating b_1

$$b_1 \rightarrow b_1 - \eta \cdot \frac{\partial E}{\partial b_1}$$

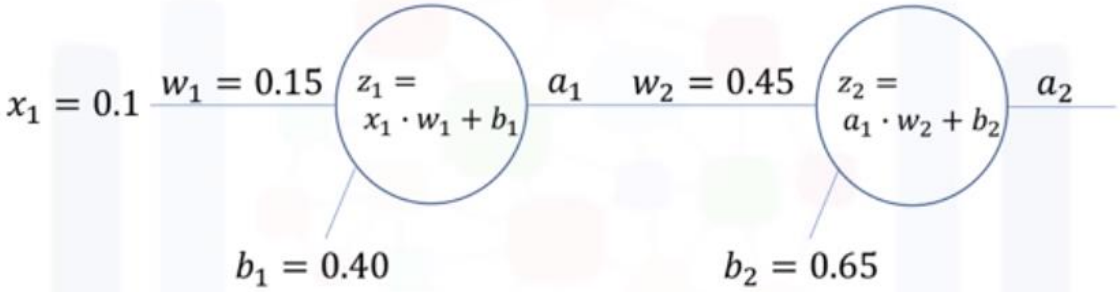
$$\frac{\partial E}{\partial b_1} = \frac{\partial E}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial a_1} \cdot \frac{\partial a_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial b_1}$$

$$= \big(-(T - a_2)\big) \cdot \big(a_2(1 - a_2)\big) \cdot (w_2) \cdot a_1(1 - a_1) \cdot 1$$

$$b_1 \rightarrow b_1 - \eta \cdot \big(-(T - a_2)\big) \cdot \big(a_2(1 - a_2)\big) \cdot (w_2) \cdot a_1(1 - a_1) \cdot 1$$

Backpropagation (Example of one epoch)

Backpropagation Example



$$z_1 = w_1 \cdot x_1 + b_1$$
$$= 0.15 \cdot 0.1 + 0.4$$
$$= 0.415$$

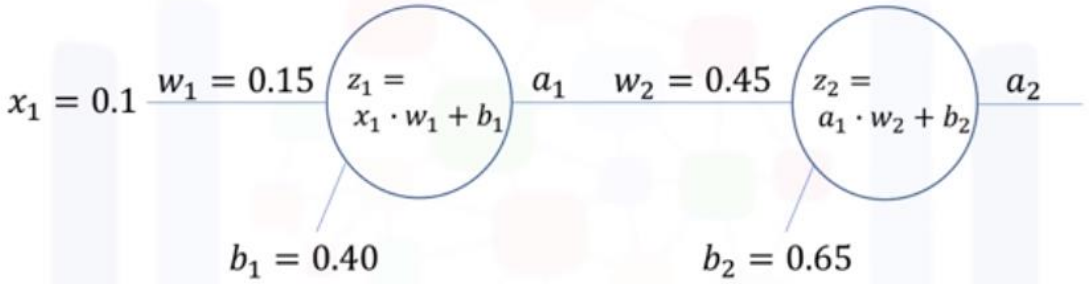
$$z_2 = w_2 \cdot a_1 + b_2$$
$$= 0.45 \cdot 0.6023 + 0.65$$
$$= 0.9210$$

$$a_1 = f(z_1) = \frac{1}{1+e^{-(0.415)}} = 0.6023$$

$$a_2 = f(z_2) = \frac{1}{1+e^{-(0.9210)}} = 0.7153$$

Backpropagation (Example of one epoch)

Backpropagation Example



$z_1 = 0.415$

$a_1 = 0.6023$

$z_2 = 0.9210$

$a_2 = 0.7153$

$T = 0.25$

$$E = \frac{1}{2}(T - a_2)^2$$

$$= \frac{1}{2}(T - a_2)^2 = 0.1083$$

$epochs = 1000$

$\epsilon = 0.001$

$\eta = 0.4$

Backpropagation (Example of one epoch)

Update w_2

$$w_2 \rightarrow w_2 - \eta \cdot \left(-(T - a_2)\right) \cdot \left(a_2(1 - a_2)\right) \cdot (a_1)$$

$$w_2 \rightarrow 0.45 - 0.4 \cdot \left(-(0.25 - 0.7153)\right) \cdot \left(0.7153(1 - 0.7153)\right) \cdot (0.6023)$$

$$w_2 \rightarrow 0.45 - 0.4 \cdot 0.05706$$

$$w_2 \rightarrow 0.427$$

Update b_2

$$b_2 \rightarrow b_2 - \eta \cdot \left(-(T - a_2)\right) \cdot \left(a_2(1 - a_2)\right) \cdot 1$$

$$b_2 \rightarrow 0.65 - 0.4 \cdot \left(-(0.25 - 0.7153)\right) \cdot \left(0.7153(1 - 0.7153)\right) \cdot 1$$

$$b_2 \rightarrow 0.65 - 0.4 \cdot 0.0948$$

$$b_2 \rightarrow 0.612$$

Backpropagation (Example of one epoch)

Update w_1

$$w_1 \rightarrow w_1 - \eta \cdot \left(-(T - a_2)\right) \cdot \left(a_2(1 - a_2)\right) \cdot (w_2) \cdot a_1(1 - a_1) \cdot x_1$$

$$w_1 \rightarrow 0.15 - 0.4 \cdot \left(-(0.25 - 0.7153)\right) \cdot \left(0.7153(1 - 0.7153)\right) \cdot 0.45 \cdot 0.6023(1 - 0.6023) \cdot 0.1$$

$$w_1 \rightarrow 0.15 - 0.4 \cdot 0.001021$$

$$w_1 \rightarrow 0.1496$$

Update b_1

$$b_1 \rightarrow b_1 - \eta \cdot \left(-(T - a_2)\right) \cdot \left(a_2(1 - a_2)\right) \cdot (w_2) \cdot a_1(1 - a_1) \cdot 1$$

$$b_1 \rightarrow 0.40 - 0.4 \cdot \left(-(0.25 - 0.7153)\right) \cdot \left(0.7153(1 - 0.7153)\right) \cdot 0.45 \cdot 0.6023(1 - 0.6023) \cdot 1$$

$$b_1 \rightarrow 0.40 - 0.4 \cdot 0.01021$$

$$b_1 \rightarrow 0.3959$$

Backpropagation (Conclusion)

Complete Training Algorithm

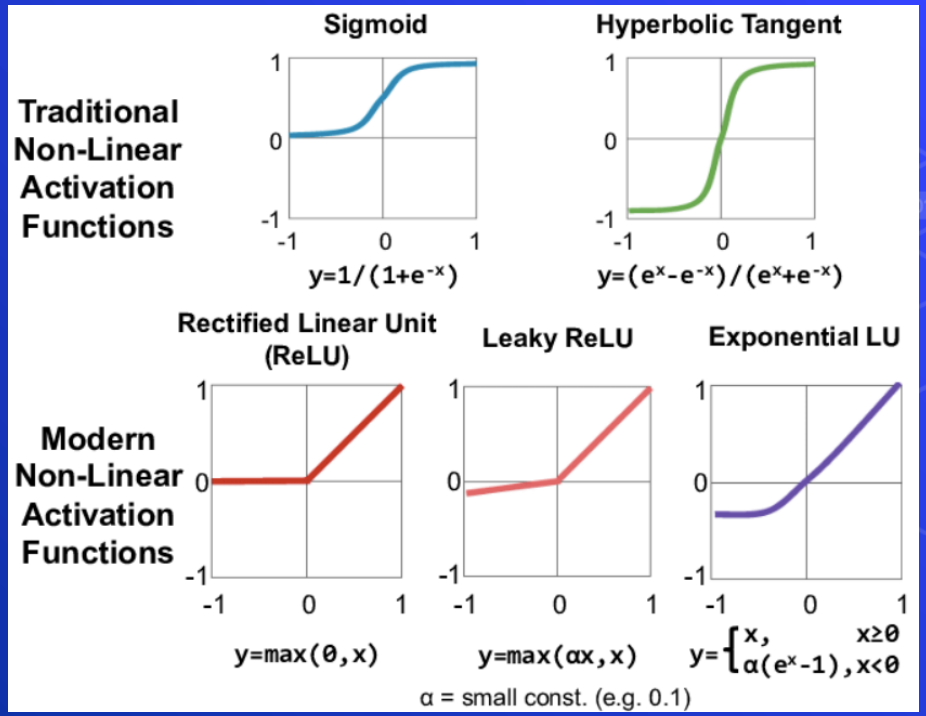
1. Initialize the weights and the biases
2. Iteratively repeat the following steps:
    1. Calculate network output using forward propagation
    2. Calculate error between ground truth and estimated or predicted output
    3. Update weights and biases through backpropagation
    4. Repeat the above three steps until number of iterations/epochs is reached or error between ground truth and predicted output is below a predefined threshold

Agenda

- What is Deep Learning
- Deep Learning Applications
- Google Colab
- Perceptron
- Artificial Neural Networks (ANN)
- Error or Loss or Cost functions
- Optimization algorithms
- Backpropagation
- **Activation functions**
- Improve neural networks training
- Deep Learning frameworks

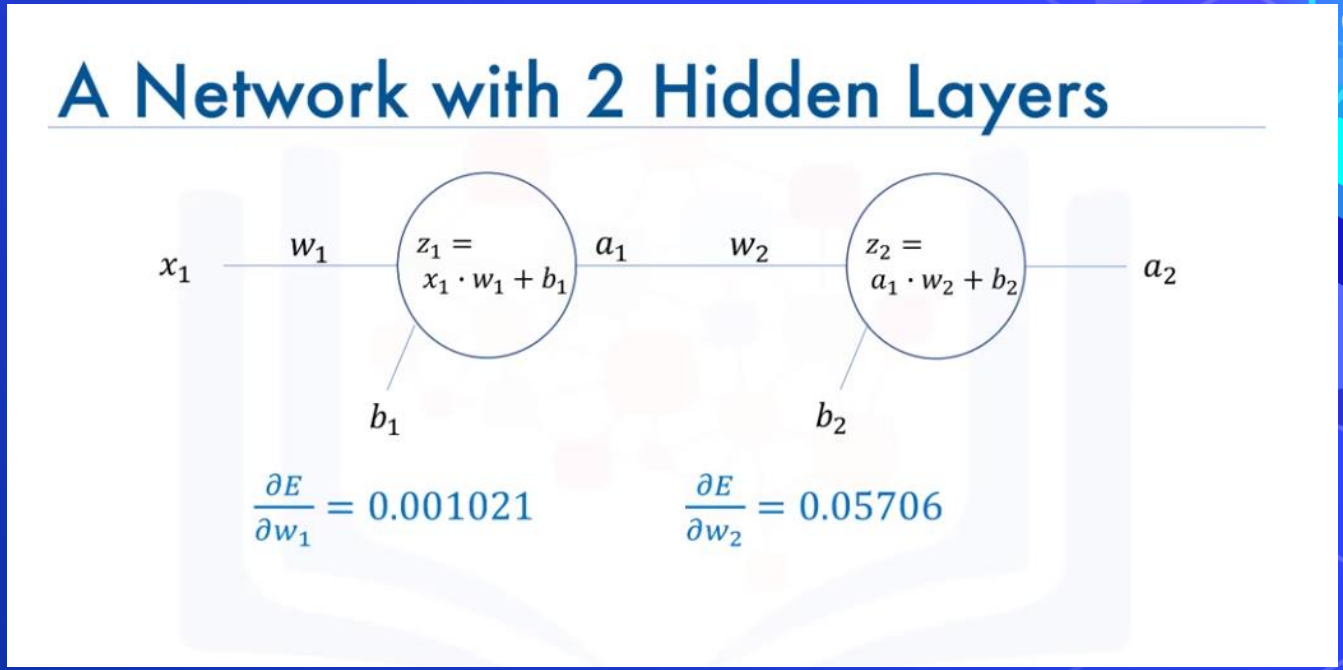
Activation functions (Purpose)

Activation functions is crucial to achieve the desired non-linearity behavior so the network be able to learn hard non linear functions and the model will be smart enough to learn complex patterns in the data.



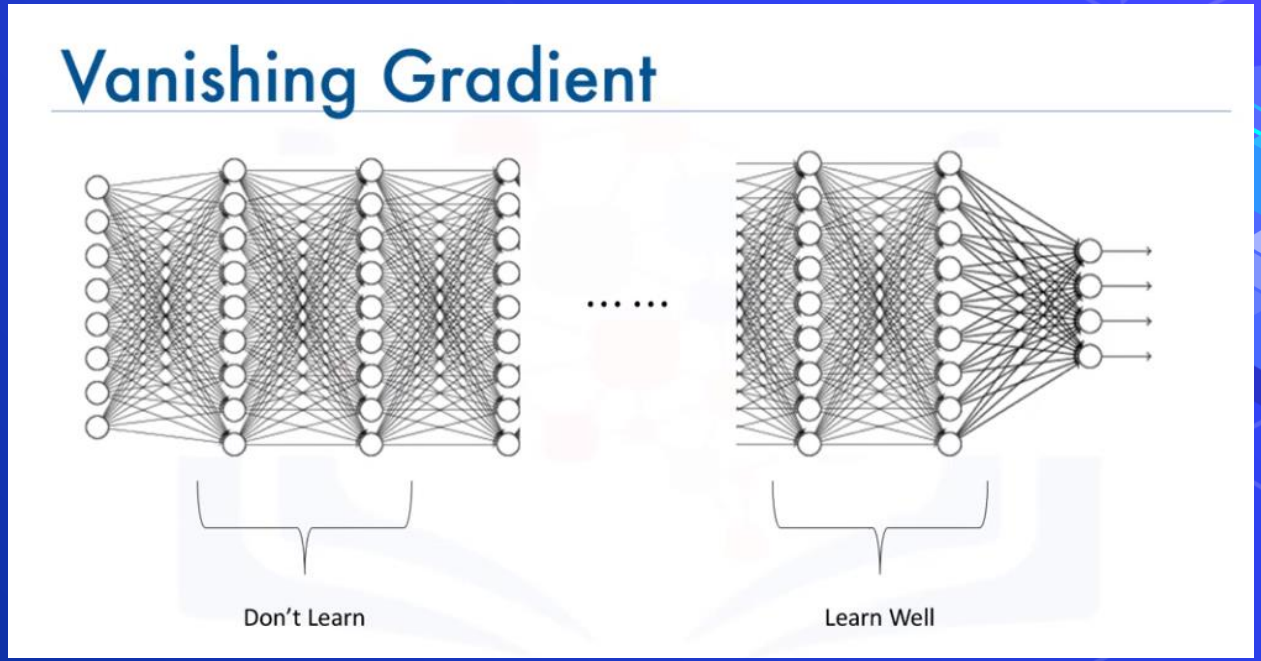
Activation functions (Vanishing Gradient)

Till now we use **Sigmoid** function as an activation function, but it makes a problem because it gives values between 0 and 1 so the gradient is small and with chain rule multiplication operations It gets smaller and smaller ~= 0 causing the neuron being useless, this is called vanishing gradient problem.



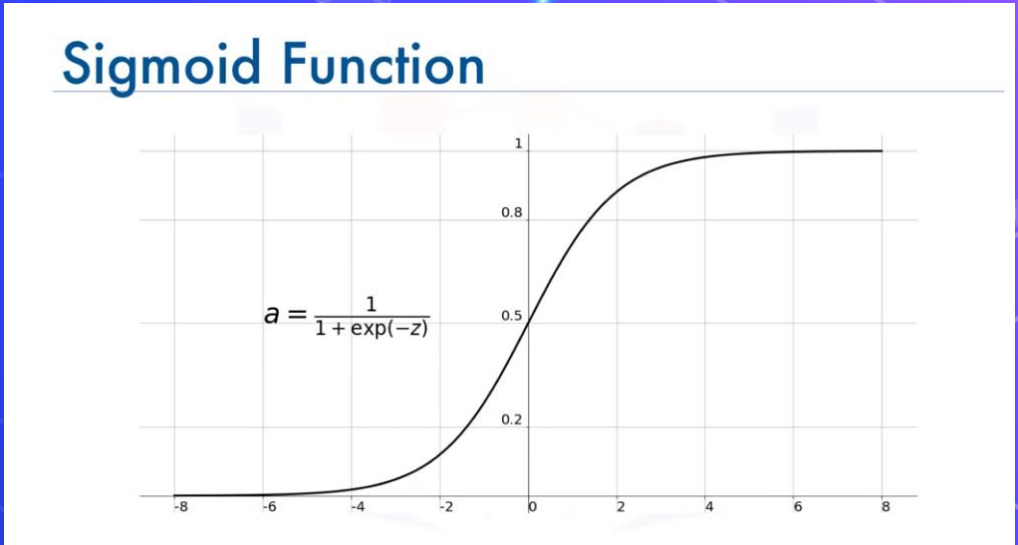
Activation functions (Vanishing Gradient)

So for the weights of earlier layers is almost 0 making the next layers also 0 and so on to the end of neural network so the network doesn't learn. we could use simpler network but in very hard problems that is bad **so what we do ???** The solution is to use another activation functions.



Activation functions

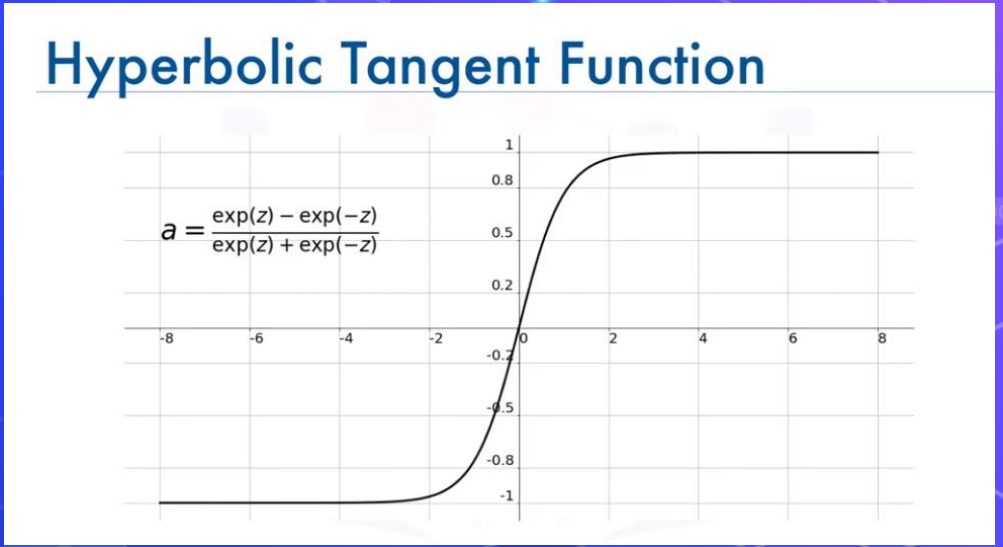
- Sigmoid (Logistic) Function
- Hyperbolic Tangent (tanh) Function
- ReLU (Rectified Linear Unit) Function
- Leaky ReLU Function
- Softmax Function



- Bad because it causes vanishing gradient problem.
- Bad because exponential function takes longer to calculate.
- Bad because values between 0 and 1, so no negative gradients.

Activation functions

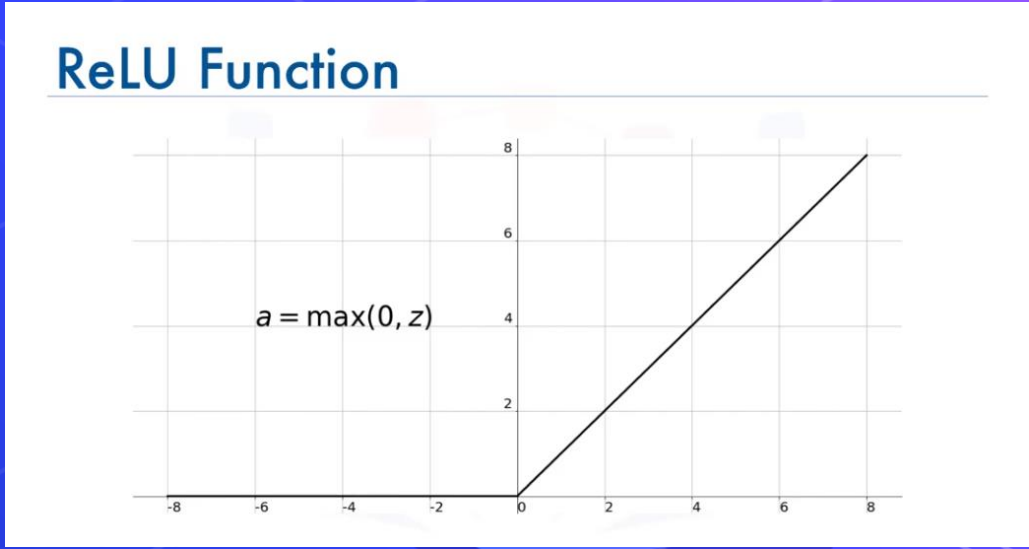
- Sigmoid (Logistic) Function
- **Hyperbolic Tangent (tanh) Function**
- ReLU (Rectified Linear Unit) Function
- Leaky ReLU Function
- Softmax Function



- Much better than sigmoid, values are between -1 and 1.
- Bad because it causes vanishing gradient problem.
- Bad because exponential function takes longer to calculate.

Activation functions

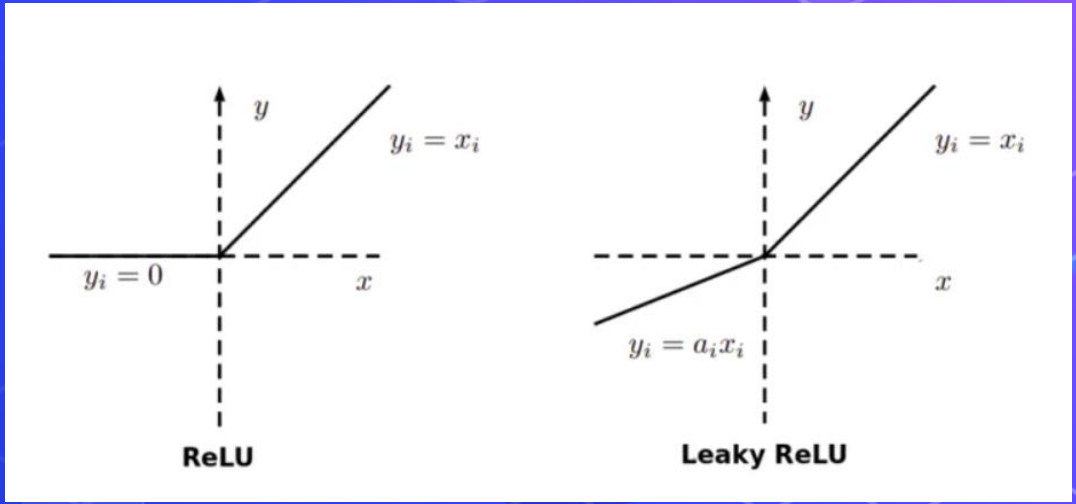
- Sigmoid (Logistic) Function
- Hyperbolic Tangent (tanh) Function
- **ReLU (Rectified Linear Unit) Function**
- Leaky ReLU Function
- Softmax Function



- Most used function when designing deep neural networks.
- Solved the vanishing gradient problem.
- Easy to calculate it just a simple comparison.
- Bad in activating neurons in the negative region.

Activation functions

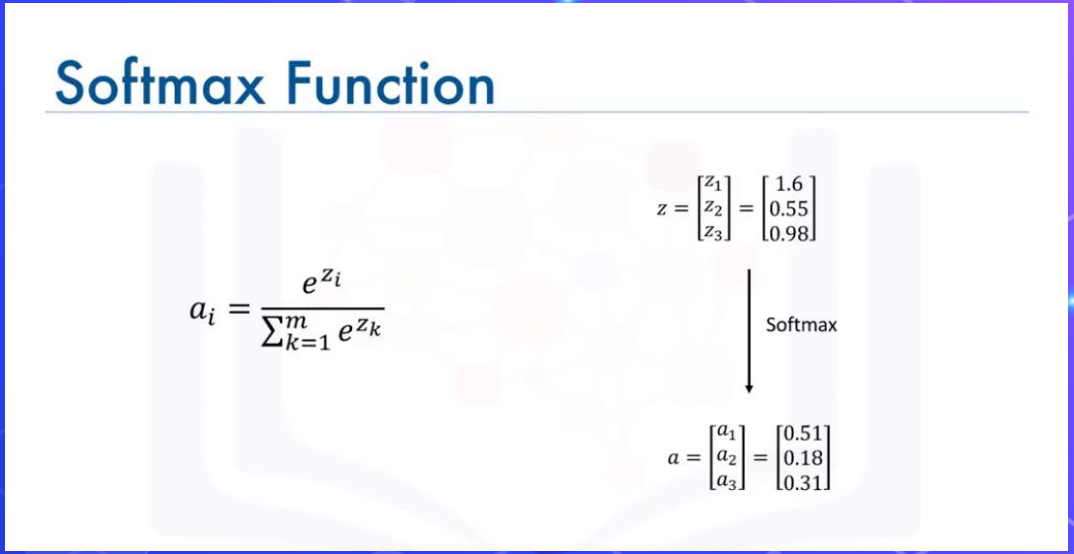
- Sigmoid (Logistic) Function
- Hyperbolic Tangent (tanh) Function
- ReLU (Rectified Linear Unit) Function
- **Leaky ReLU Function**
- Softmax Function



- Very good like ReLU.
- Easy to calculate it just a simple comparison.
- Solves the activating neurons in the negative region problem.
- Use alpha = 0.1 as a good start but not 1 it became linear.

Activation functions

- Sigmoid (Logistic) Function
- Hyperbolic Tangent (tanh) Function
- ReLU (Rectified Linear Unit) Function
- Leaky ReLU Function
- **Softmax Function**



- Used only for output layer for multi class classification problems.

Activation functions (Very Important Notes)

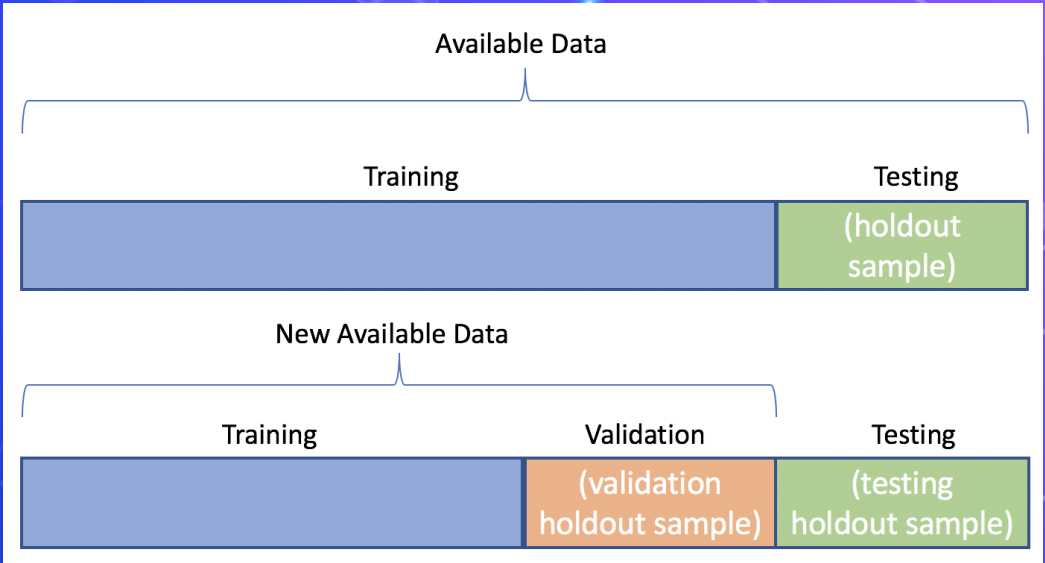
- Don't use **Sigmoid** or **Tanh** for hidden layers neurons because of vanishing gradient problem, but instead use **ReLU** or **Leaky ReLU**.
- Use **Sigmoid** only for output layer neurons for **Binary Classification** and **Multi Label classification** problems, note that you use **Binary Cross Entropy** loss function.
- Use **Softmax** only for output layer neurons for **Multi Class classification** problems, note that you use **Categorical Cross Entropy** loss function.
- For Regression problems you don't need to use any activation function at output layer, as if you use a **linear** activation function.

Agenda

- What is Deep Learning
- Deep Learning Applications
- Google Colab
- Perceptron
- Artificial Neural Networks (ANN)
- Error or Loss or Cost functions
- Optimization algorithms
- Backpropagation
- Activation functions
- **Improve neural networks training**
- Deep Learning frameworks

Improve neural networks training

- Validation Sets
- Hyperparameter tuning
- Dropout
- Early Stopping
- Learning Rate Decay



- Used for tuning the model while training.
- You can use for example 0.1 split from training data.

Improve neural networks training

- Validation Sets
- Hyperparameter tuning
- Dropout
- Early Stopping
- Learning Rate Decay

- **Number of layers.**

  Keep increasing layers till validation loss stop decreasing.

- **Number of neurons or units for each layer.**

  Keep increasing neurons till validation loss stop decreasing.

- **Batch Size.**

  Can be 32, 64, 128, etc.

- **Learning Rate.**

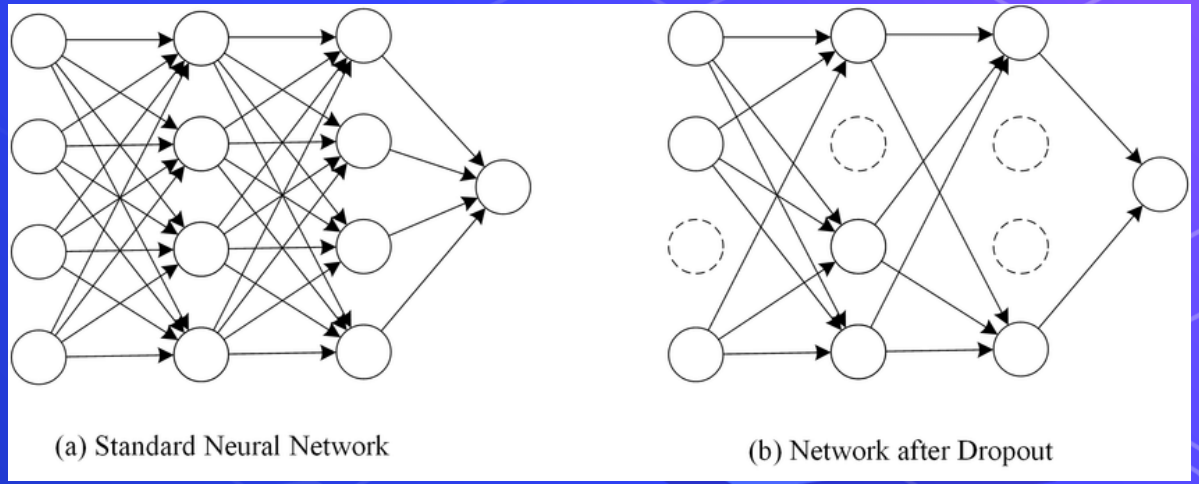
  Start with 0.001 and decrease gradually.

- **Number of Epochs.**

  Keep increasing epochs while train loss & validation loss decrease.

Improve neural networks training

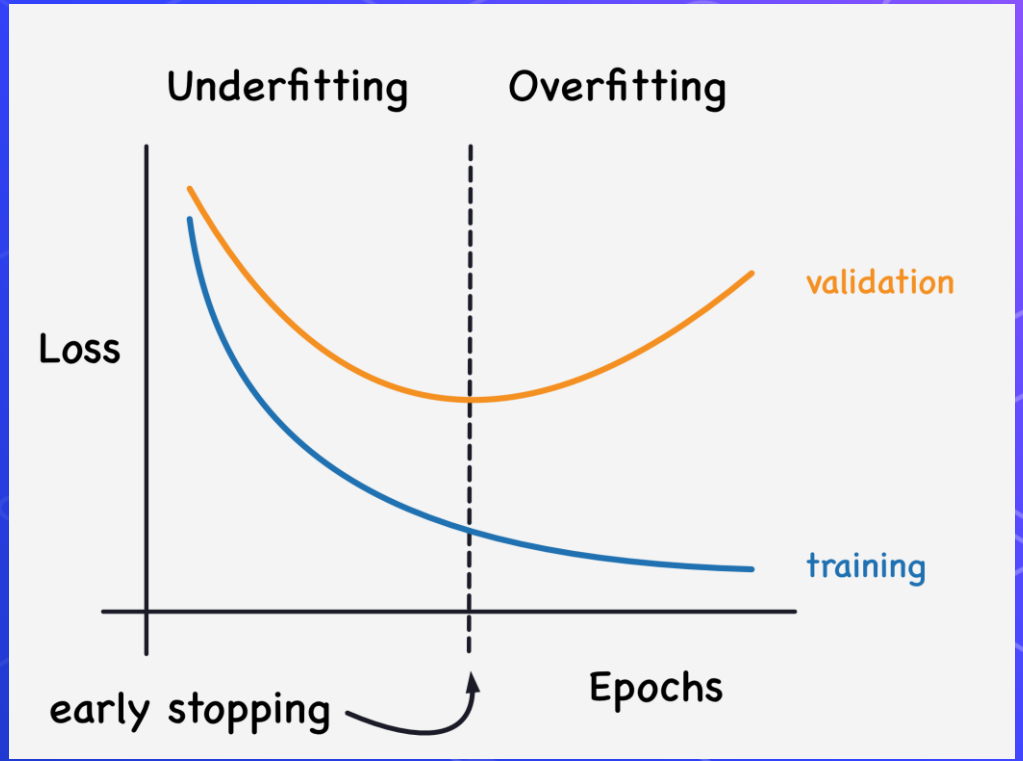
- Validation Sets
- Hyperparameter tuning
- **Dropout**
- Early Stopping
- Learning Rate Decay



- Used to reduce overfitting by making simpler different models hence generalize better.
- Choose P(drop) the probability of dropping a neuron to a number between 0.2 and 0.5.
- There are another methods called **regularization** to add a penalty to high weights minimizing it's value =~ 0 they are L1 & L2 methods they also called **weight decay.**

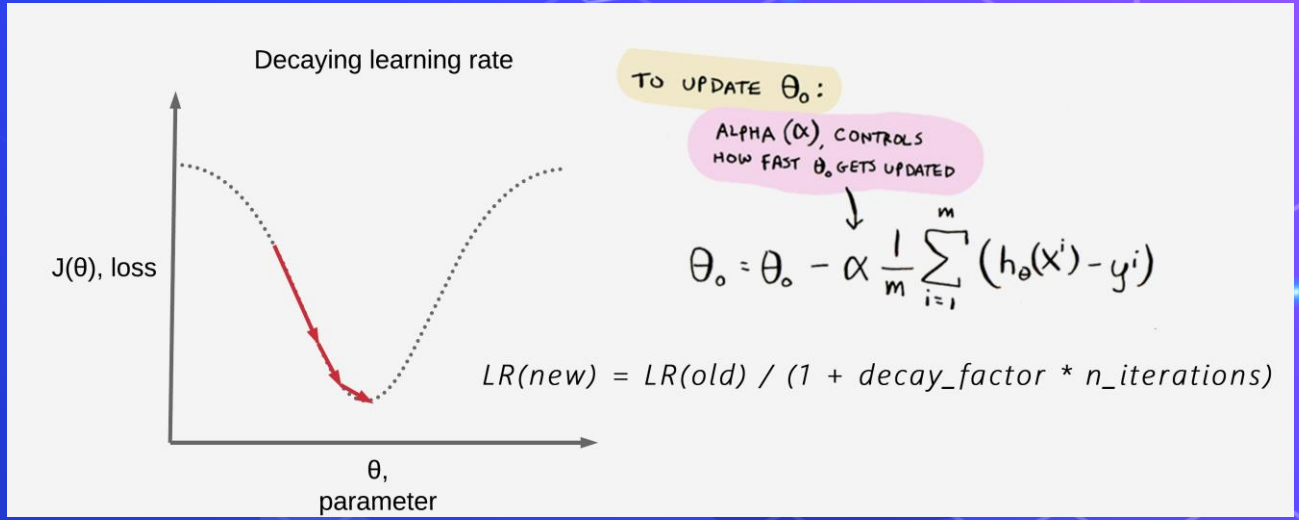
Improve neural networks training

- Validation Sets
- Hyperparameter tuning
- Dropout
- **Early Stopping**
- Learning Rate Decay



Improve neural networks training

- Validation Sets
- Hyperparameter tuning
- Dropout
- Early Stopping
- Learning Rate Decay



- Too small a learning rate and your neural network will learn very slow hence consuming time and may not learn at all.
- Too large a learning rate and you may overshoot areas of low loss.
- Start decay the LR when you detect the validation loss stay the same or increased.

Agenda

- What is Deep Learning
- Deep Learning Applications
- Google Colab
- Perceptron
- Artificial Neural Networks (ANN)
- Error or Loss or Cost functions
- Optimization algorithms
- Backpropagation
- Activation functions
- Improve neural networks training
- **Deep Learning frameworks**

Deep Learning frameworks

TensorFlow

Keras

PyTorch

Deep Learning frameworks

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau, ModelCheckpoint, EarlyStopping


# Define the structure of model
model = Sequential()
model.add(Dense(1024, input_shape=[6], activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(2048, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(512, activation='relu'))
model.add(Dense(5, activation='softmax'))
model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])


# Define the callbacks functions
lrd = ReduceLROnPlateau(monitor='val_loss', patience=200, verbose=1,
                        factor = 0.75,
                        min_lr = 1e-10)
mcp = ModelCheckpoint('model.h5')
es = EarlyStopping(verbose=1, patience=600)


# Train the model
model.fit(x=x_train, y=y_train, epochs=20, batch_size=64,
          validation_split=0.1,
          callbacks=[lrd, mcp, es])


# Evaluate the model
model.evaluate(x_test, y_test)


# Predict test dataset
model.predict(x_test)
```

Questions ?!

Thanks!

>_ Live long and prosper