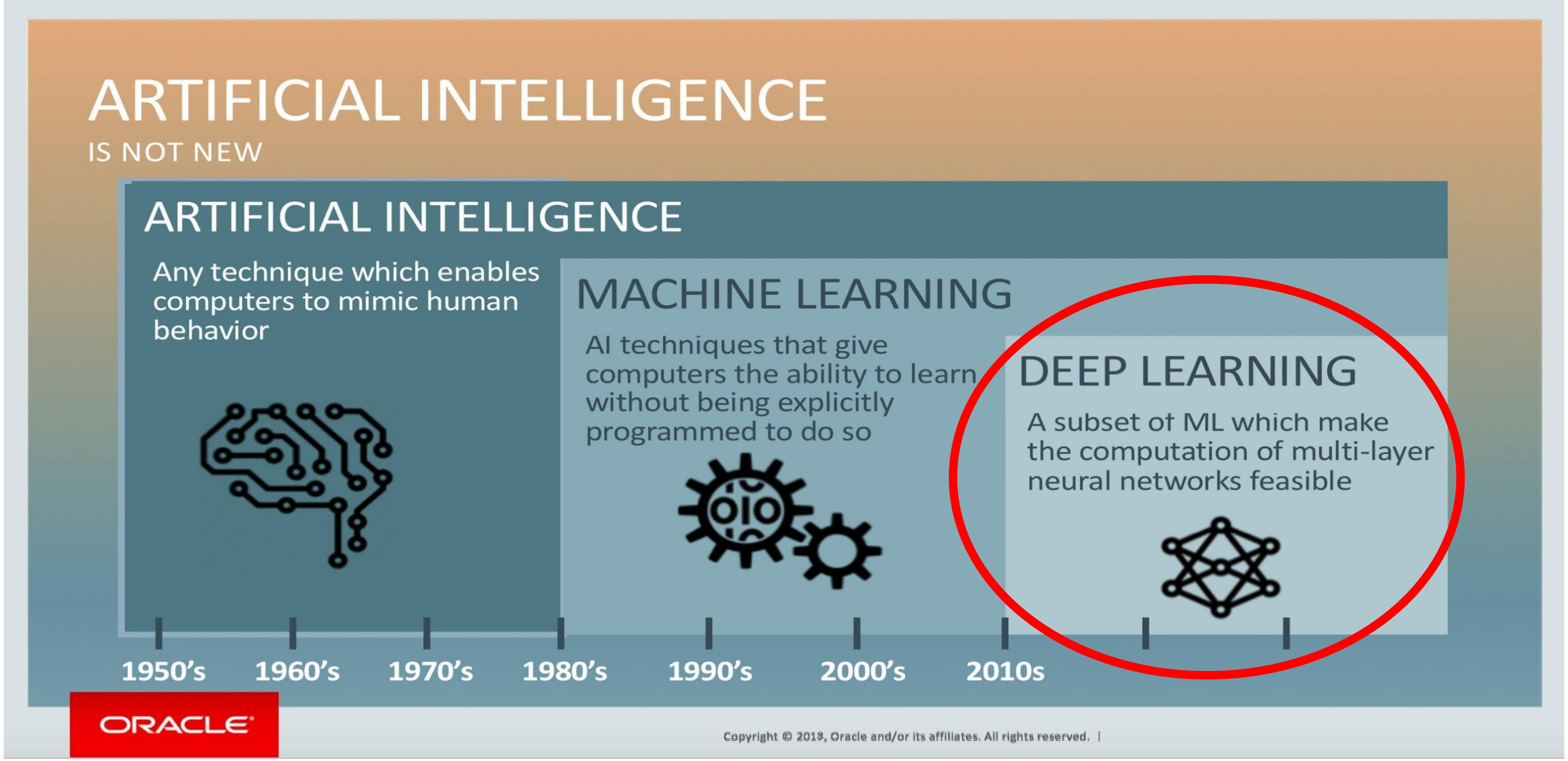
Deep Learning

Big Data & Machine Learning Bootcamp - Keep Coding

Outline

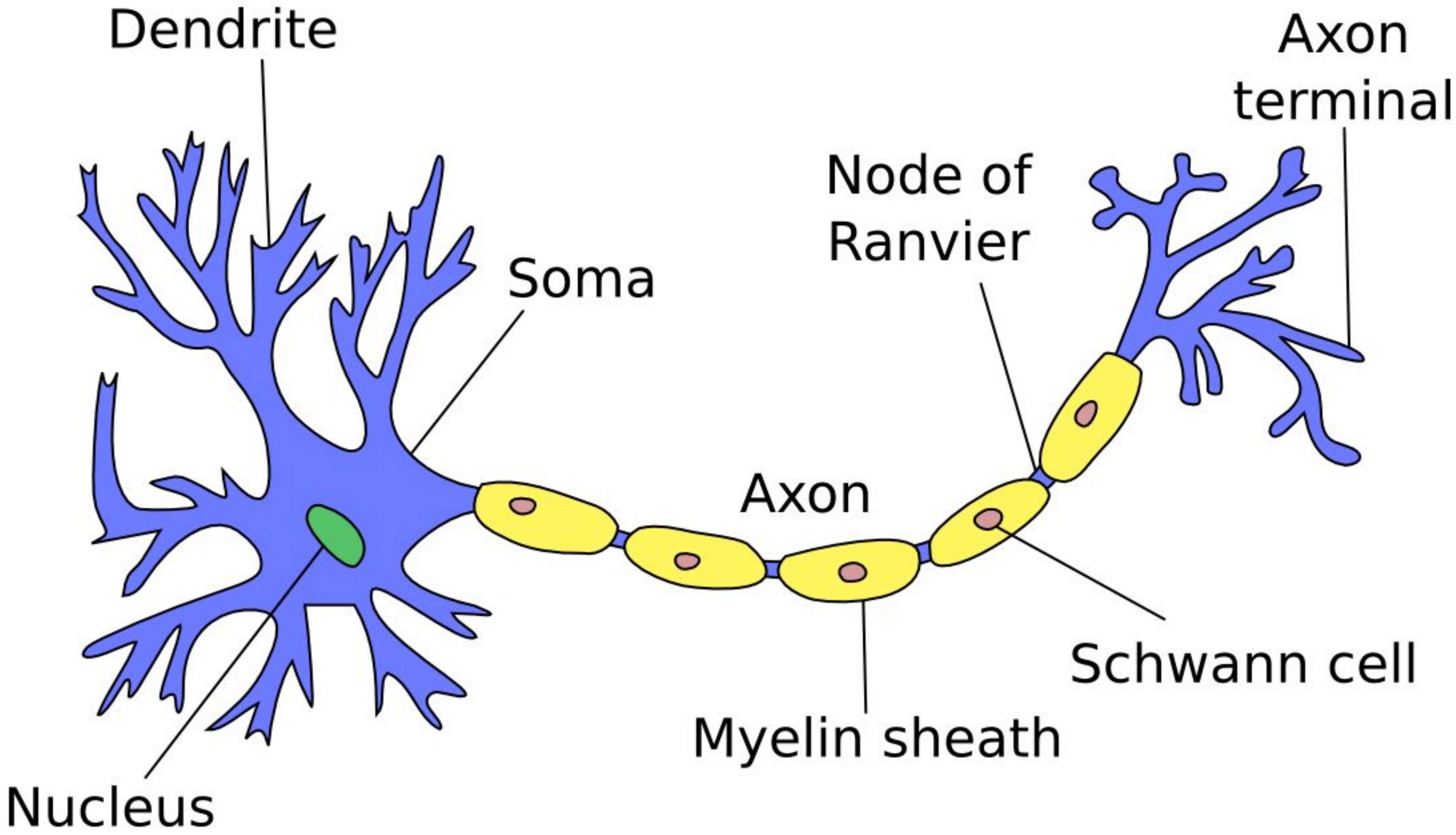
1. Intro to Neural Networks
2. Binary Classification and Logistic Regression
3. Gradient Descent
4. Vectorization
5. Activation functions
6. Backpropagation
7. Random initialization



Fuente: Oracle

1. Intro to Neural Networks

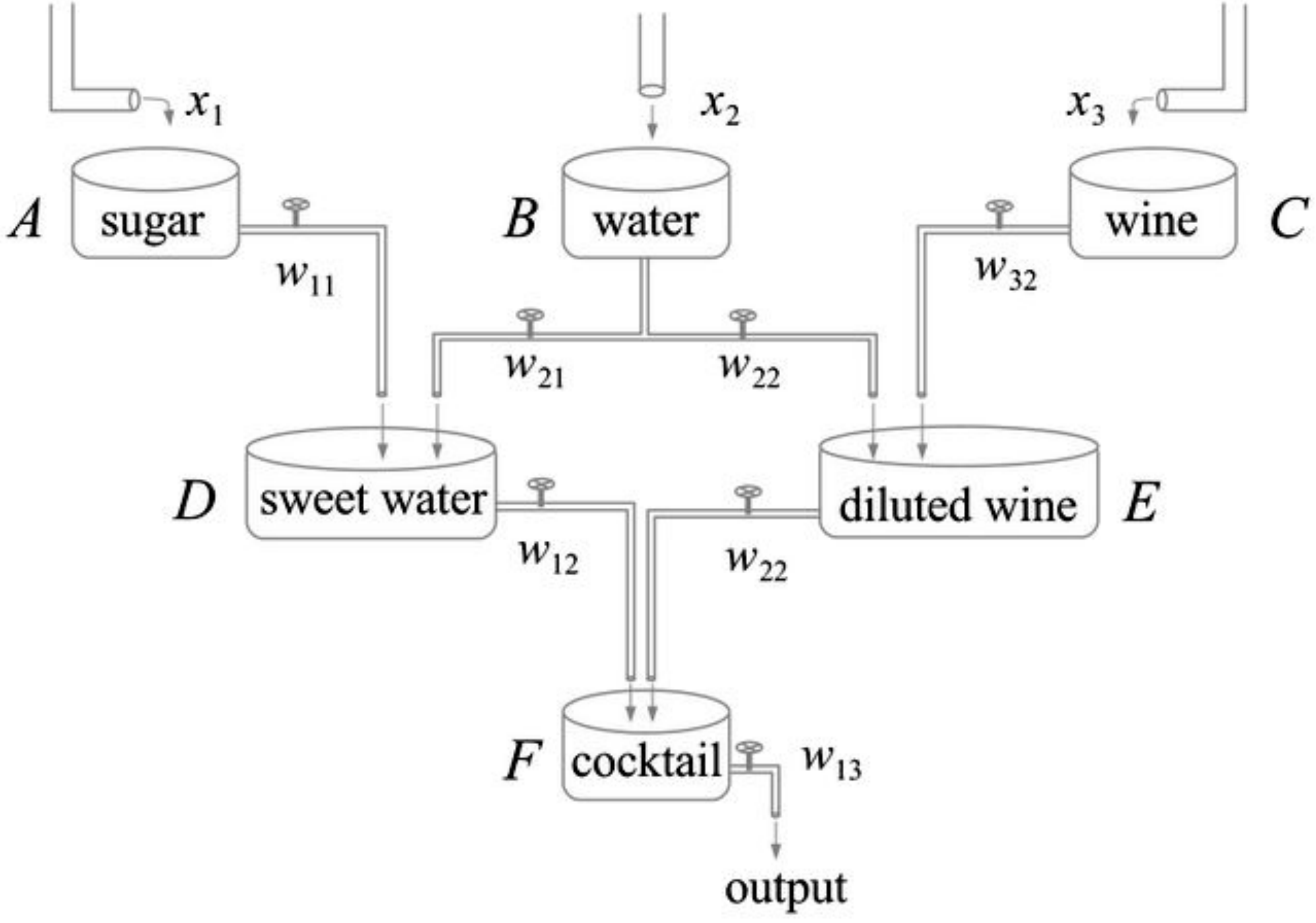
Source of inspiration



Source: https://www.marekrei.com/blog/neural-networks-part-2-the-neuron/

1. Intro to Neural Networks

Examples of neural networks

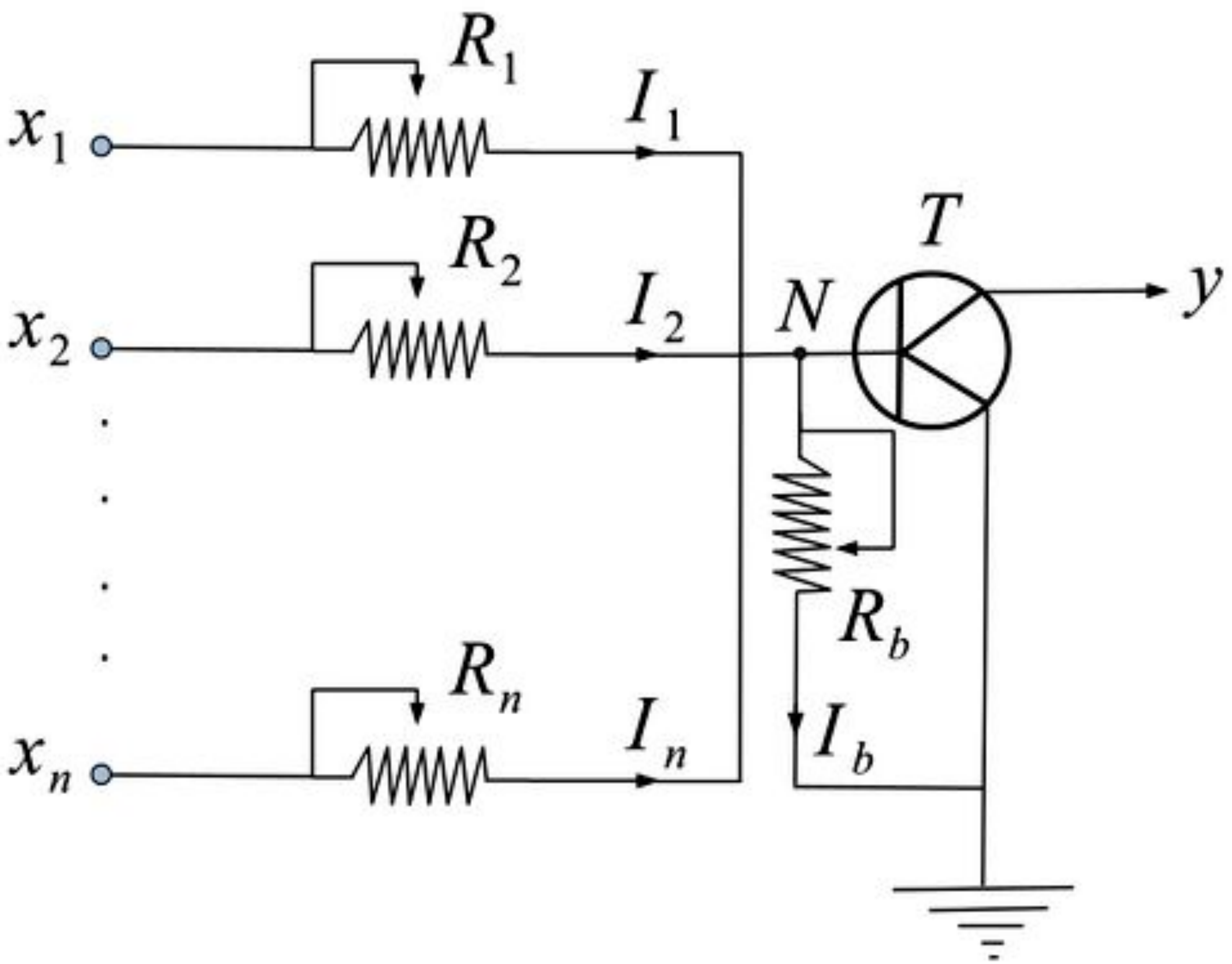


A cocktail factory as a neural network with two hidden layers

Source: Ovidiu Calin. Deep Learning Architectures: A Mathematical Approach

1. Intro to Neural Networks

Examples of neural networks

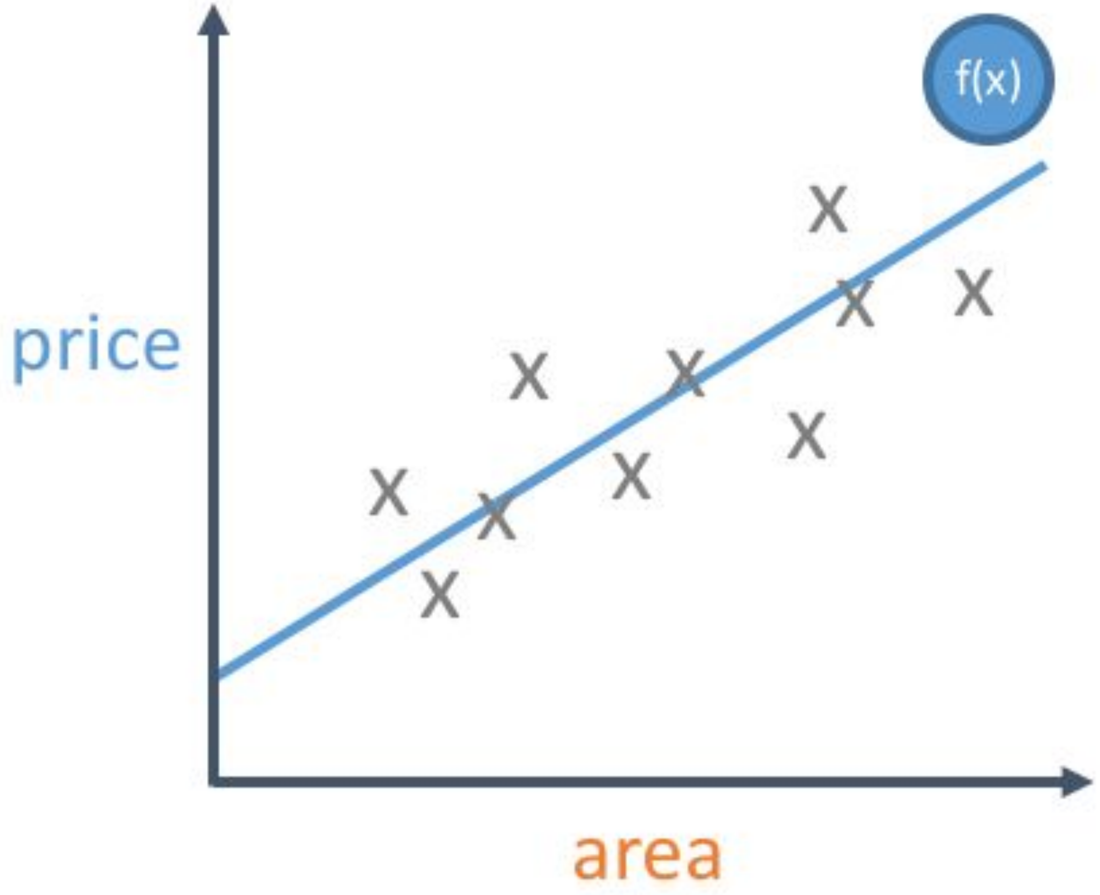


The neuron implemented as an electronic circuit

Source: Ovidiu Calin. Deep Learning Architectures: A Mathematical Approach

1. Intro to Neural Networks

Let's continue by thinking on a simple linear model. We want to predict the house price based on the area.



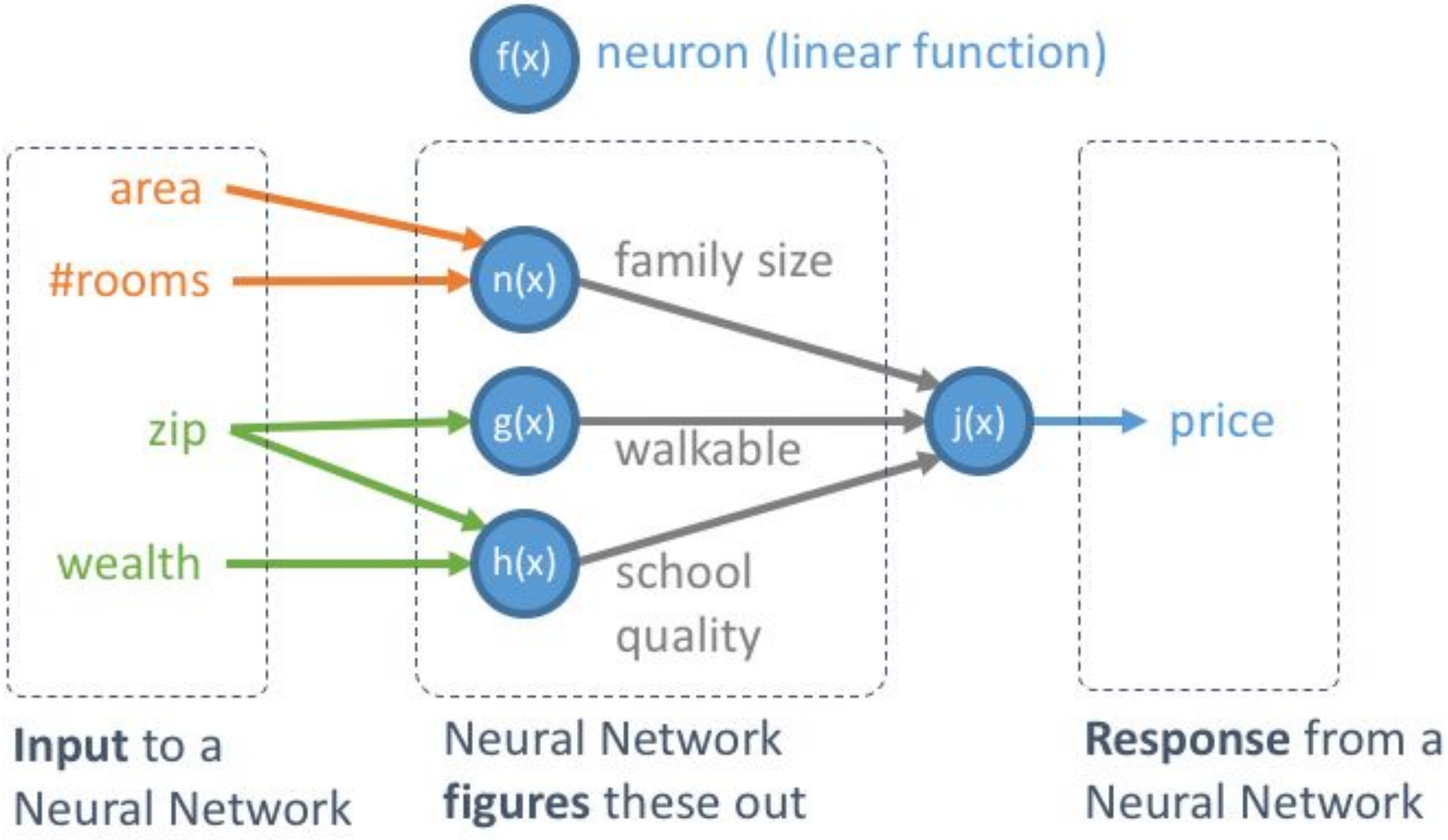
This can be solve by drawing a line using linear regression.

Prices are non-negative!

Sources:
- https://manavsehgal.com/lecture-notes-from-artificial-intelligence-is-the-new-electricity-by-andrew-ng-4712dcbf26e5

1. Intro to Neural Networks

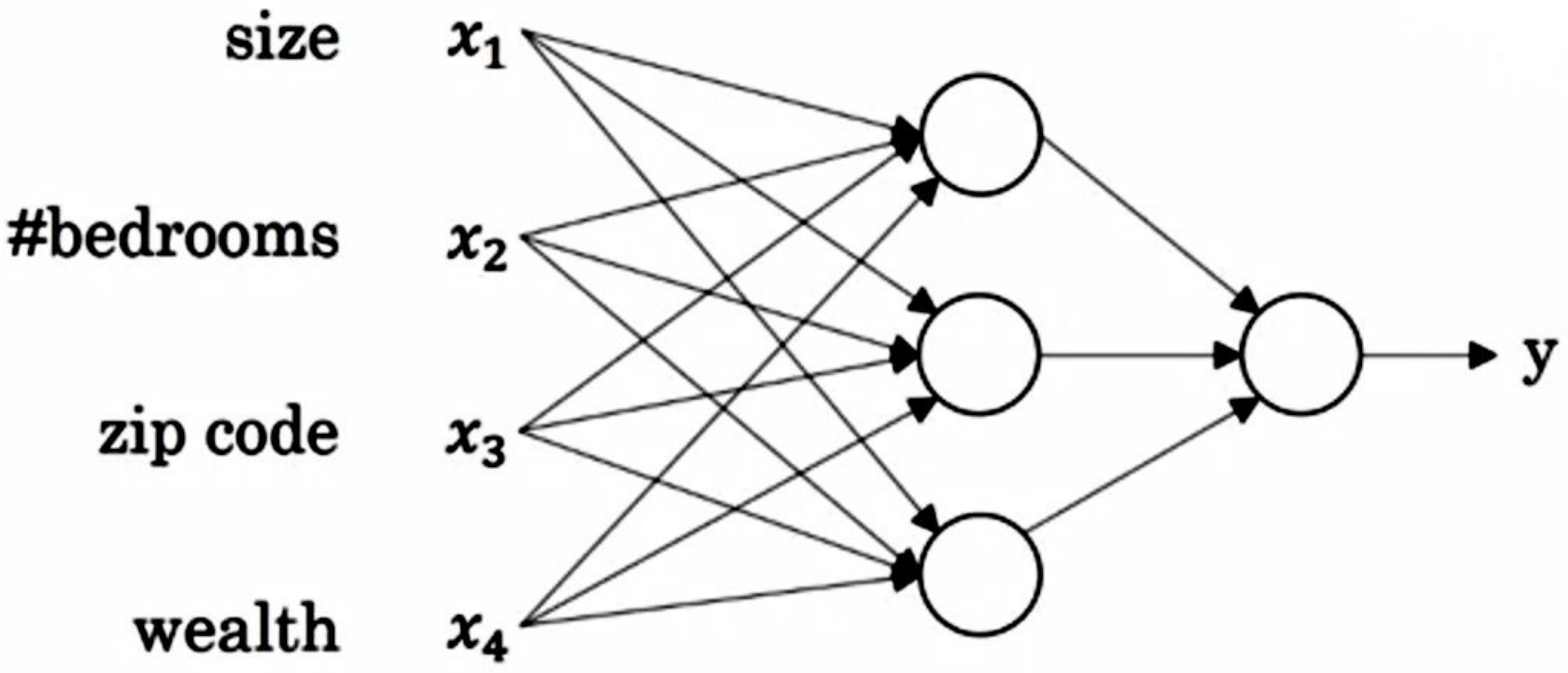
Now the problem is to predict house price for a given house with a number of input features like area, number of rooms, zip code, and wealth of the neighbourhood.



Sources:
- https://manavsehgal.com/lecture-notes-from-artificial-intelligence-is-the-new-electricity-by-andrew-ng-4712dcbf26e5

1. Intro to Neural Networks

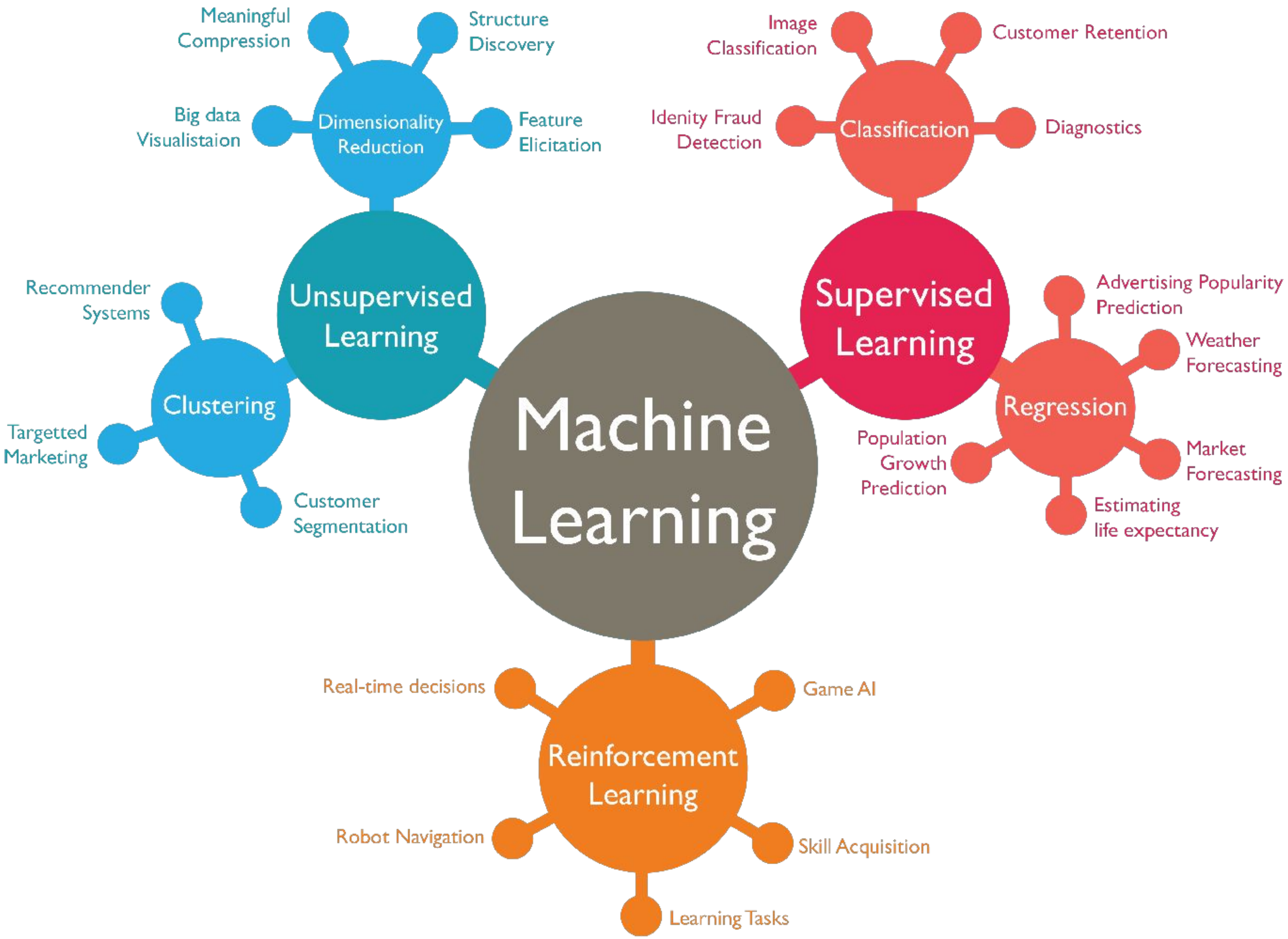
This is what we actually implement when using a neural network. We let the NN to learn whatever features/characteristics to predict the house price



Sources:
- Coursera

1. Intro to Neural Networks

Let's put this application in context. **Supervised learning!**



Sources:
- http://www.favouriteblog.com/wp-content/uploads/2017/07/Types-of-Learning.png

1. Intro to Neural Networks

Most of the hype in deep learning is given by their power on supervised learning applications

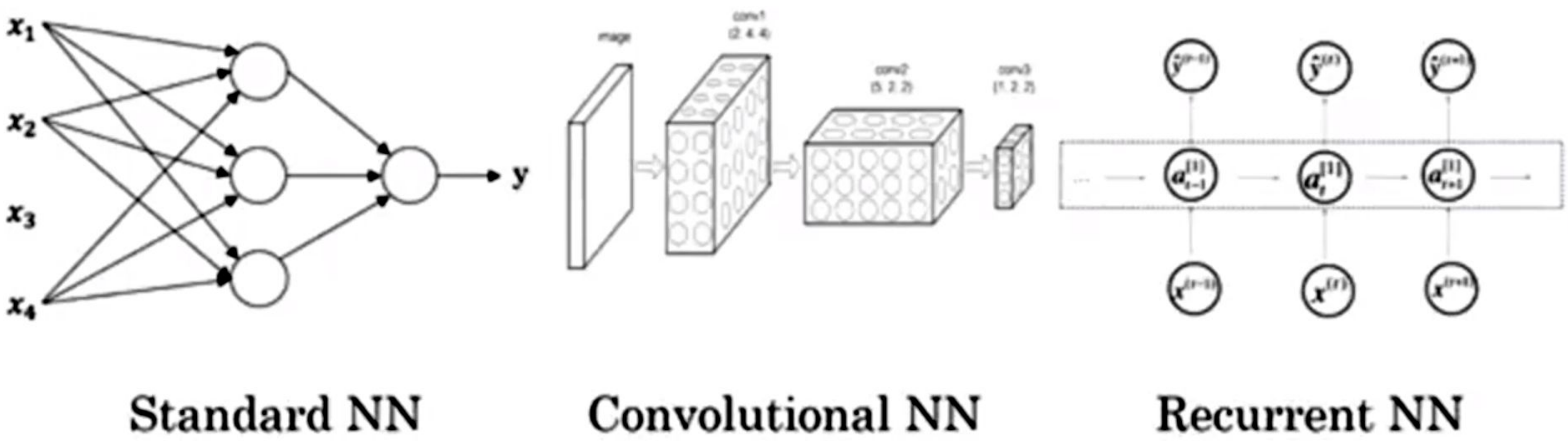
Supervised Learning

Input(x)	Output (y)	Application
Home features	Price	Real Estate
Ad, user info	Click on ad? (0/1)	Online Advertising
Image	Object (1,...,1000)	Photo tagging
Audio	Text transcript	Speech recognition
English	Chinese	Machine translation
Image, Radar info	Position of other cars	Autonomous driving

Sources:
- Coursera

1. Intro to Neural Networks

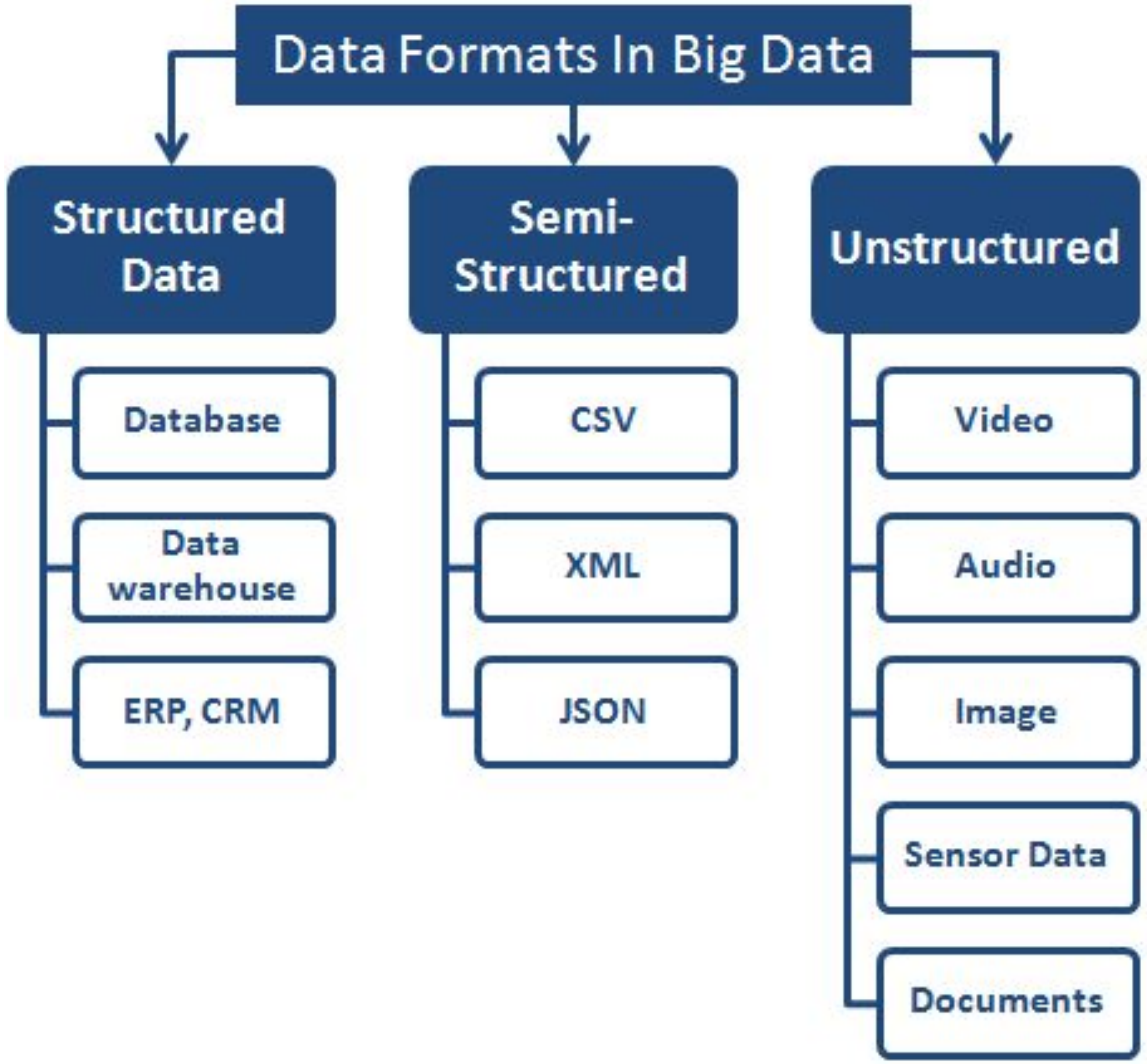
Depending on the application we use a standard NN, convolutional NN or a recurrent NN



Sources:
- Coursera

1. Intro to Neural Networks

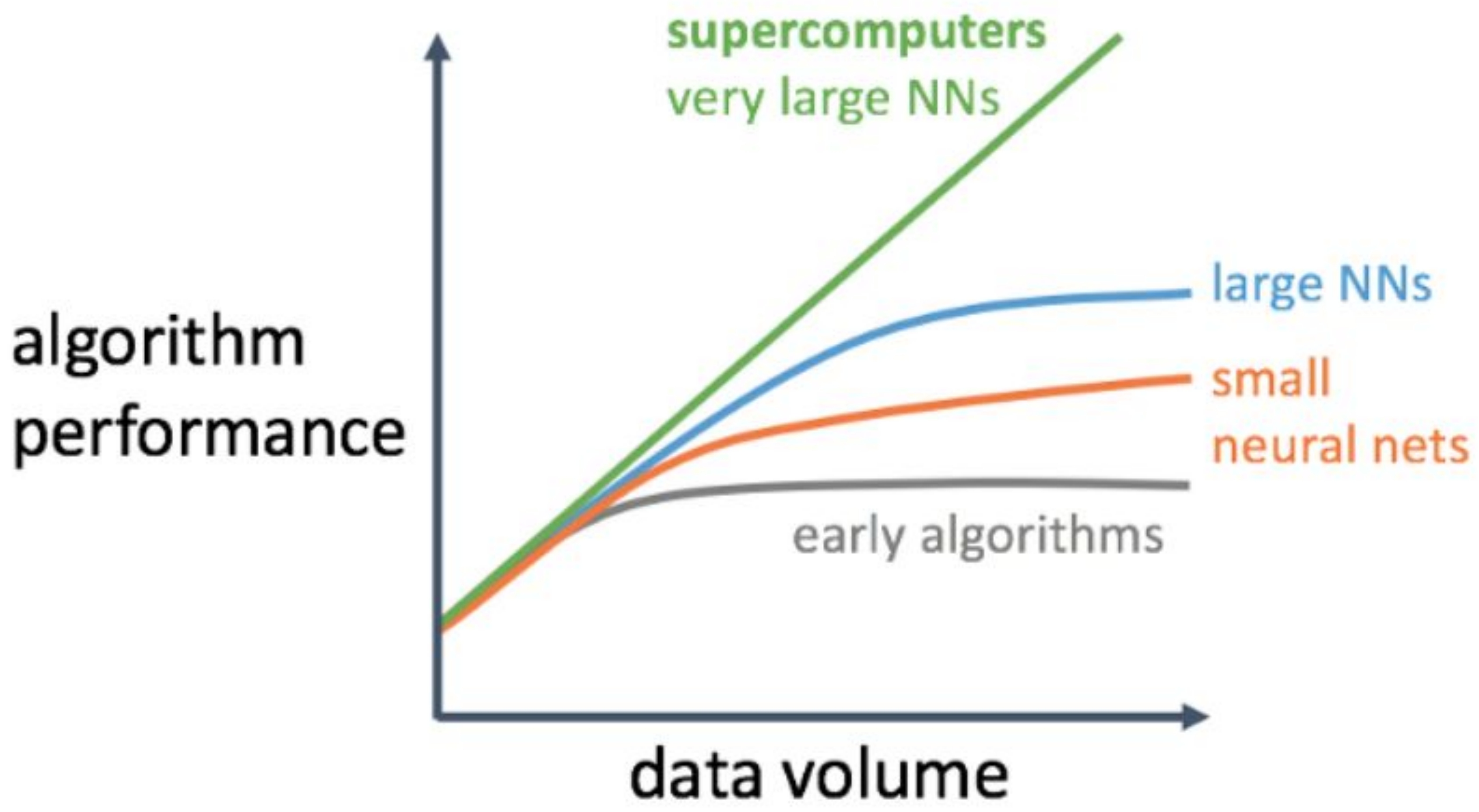
There also different data formats used to train/test NN depending on the application:



Sources:
- http://tryqa.com/wp-content/uploads/2017/07/Big_Data_Testing-Data_formats_tutorial.png

1. Intro to Neural Networks

The main concepts of deep learning have been around for many decades. But why now they that powerful and useful?



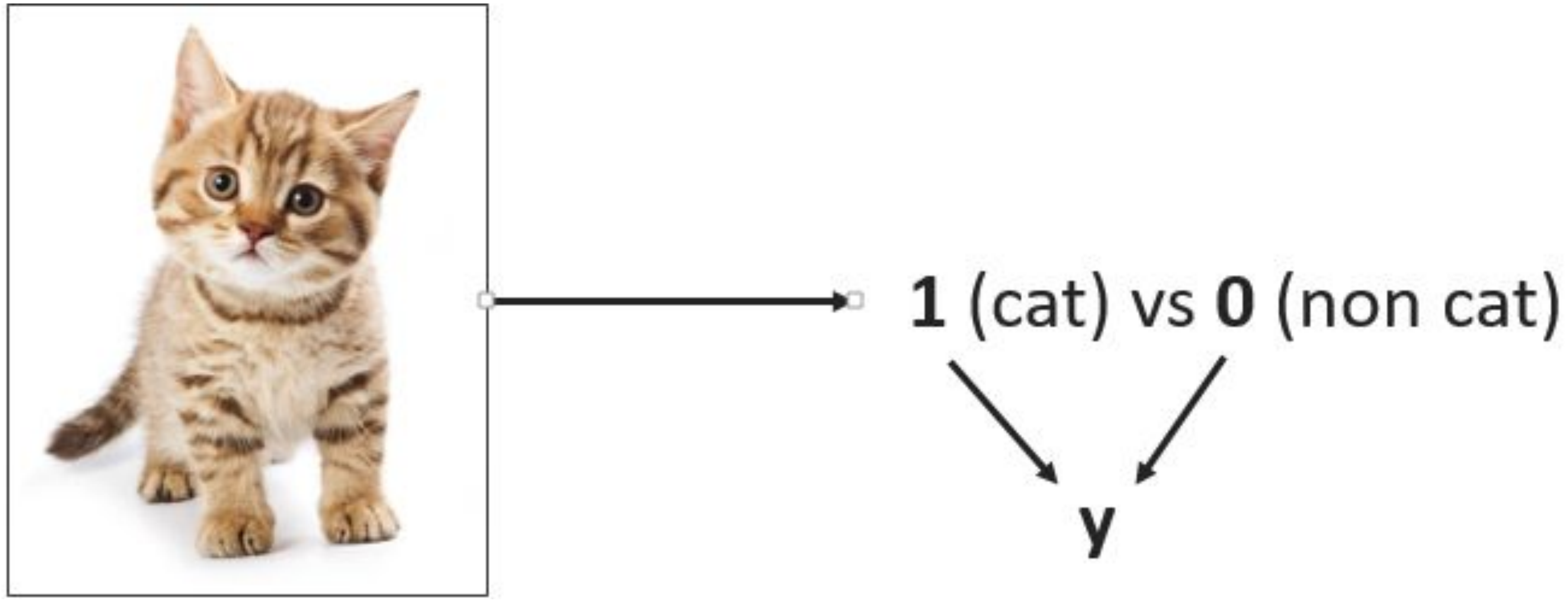
When having small amount of data algorithm performance is similar.

Sources:
- https://manavsehgal.com/lecture-notes-from-artificial-intelligence-is-the-new-electricity-by-andrew-ng-4712dcbf26e5

2. Binary Classification and Logistic Regression

To clarify: *Logistic regression is an algorithm for binary classification!*

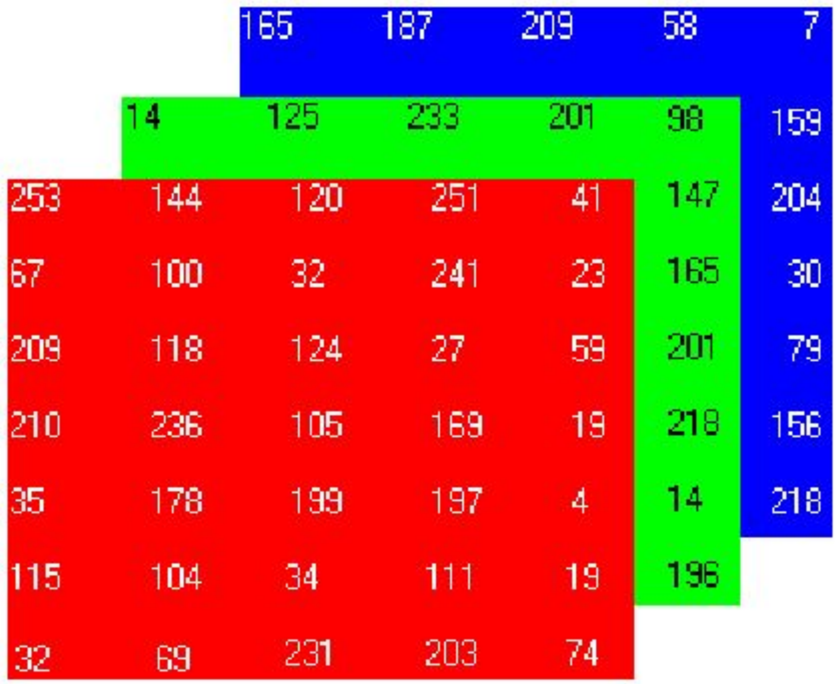
It is a simple way of explaining what a neural network can do.



Sources:
- http://media5.datahacker.rs/2018/06/word-image-58.png

2. Binary Classification and Logistic Regression

What is actually an image? Three 2D matrices that represent the Red, Green, and Blue colors



In this case (for logistic regression), the labels/classes/Categories/Ground-Truth are represented by a vector of zeros or ones

Sources:
- http://media5.datahacker.rs/2018/06/word-image-58.png
- https://yangwangresearch.com/2019/06/11/identifying-scene-changes-in-videos/

2. Binary Classification and Logistic Regression

We use all the pixel values (RGB) to classify the images into a cat or non-cat class.
To do that the equation for logistic regression:

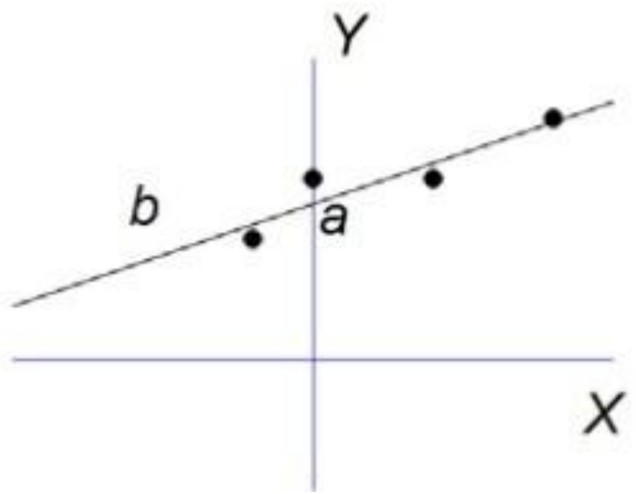
$$\hat{Y} = bX + \boxed{a}$$

predicted values of *Y*

b = slope = rate of predicted ↑/↓ for Y scores for each unit increase in *X*

Y-intercept = level of *Y* when X is 0

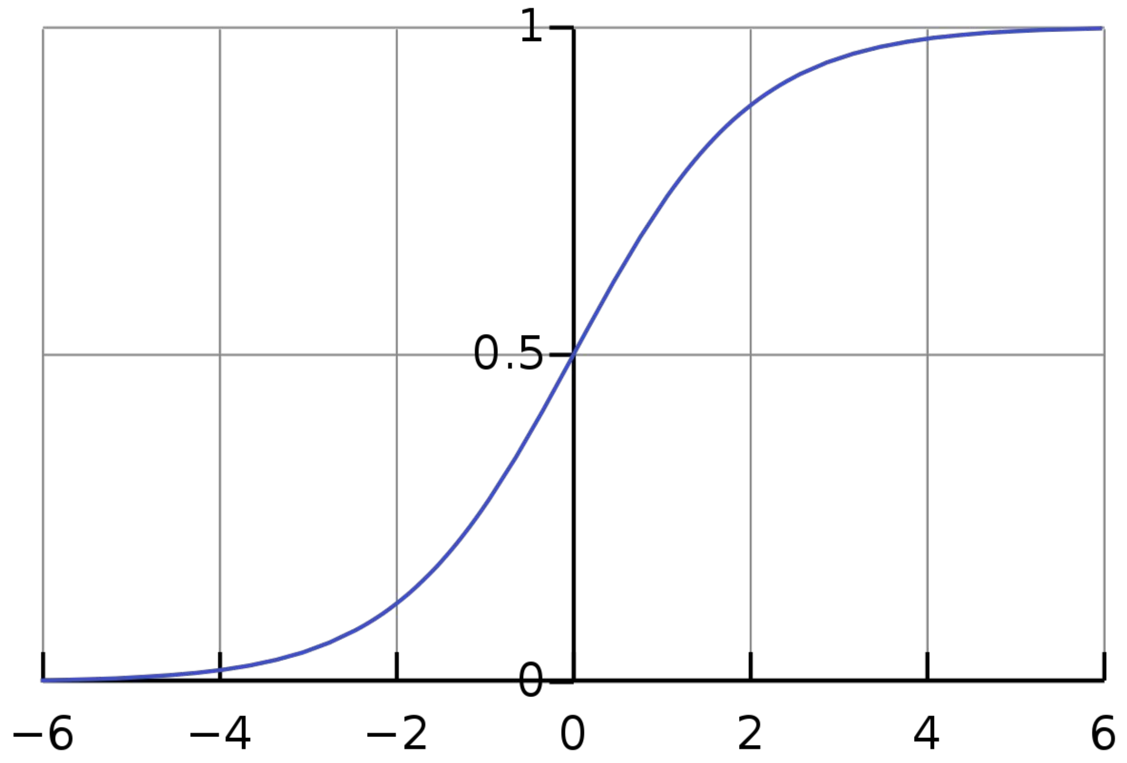
Also called bias (b)



Sources:
- https://oldwomanatschool.blog/2017/10/14/age-is-just-a-perception/

2. Binary Classification and Logistic Regression

What we need from the logistic regression equation are values between 0 and 1 representing the **probability of being a cat or a non-cat**. However, the output of the logistic regression could be bigger than 1 and smaller than 0. A sigmoid function will help us to deal with this problem.



Values in the X-axis that are smaller than zero will output values close to zero in the Y-axis.

Similar to values bigger than 1 in the X-axis. The output will be close to 1

2. Binary Classification and Logistic Regression

So, what we have after using the sigmoid function and the logistic regression equation is the following:

$$\hat{y} = \sigma(w^T x + b), \text{ where } \sigma(z) = \frac{1}{1+e^{-z}}$$

How can we evaluate that the model is doing well? We use the so-called cost function:

$$\mathcal{L}(\hat{y}, y) = \frac{1}{2}(\hat{y}-y)^2$$

The gradient descent doesn't work well for this equation.

Sources:
- Coursera

2. Binary Classification and Logistic Regression

So, we defined a function that works for the gradient descent:

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

To clarify: A loss function is the error of a single training sample and the cost function is the average error of all the training samples.

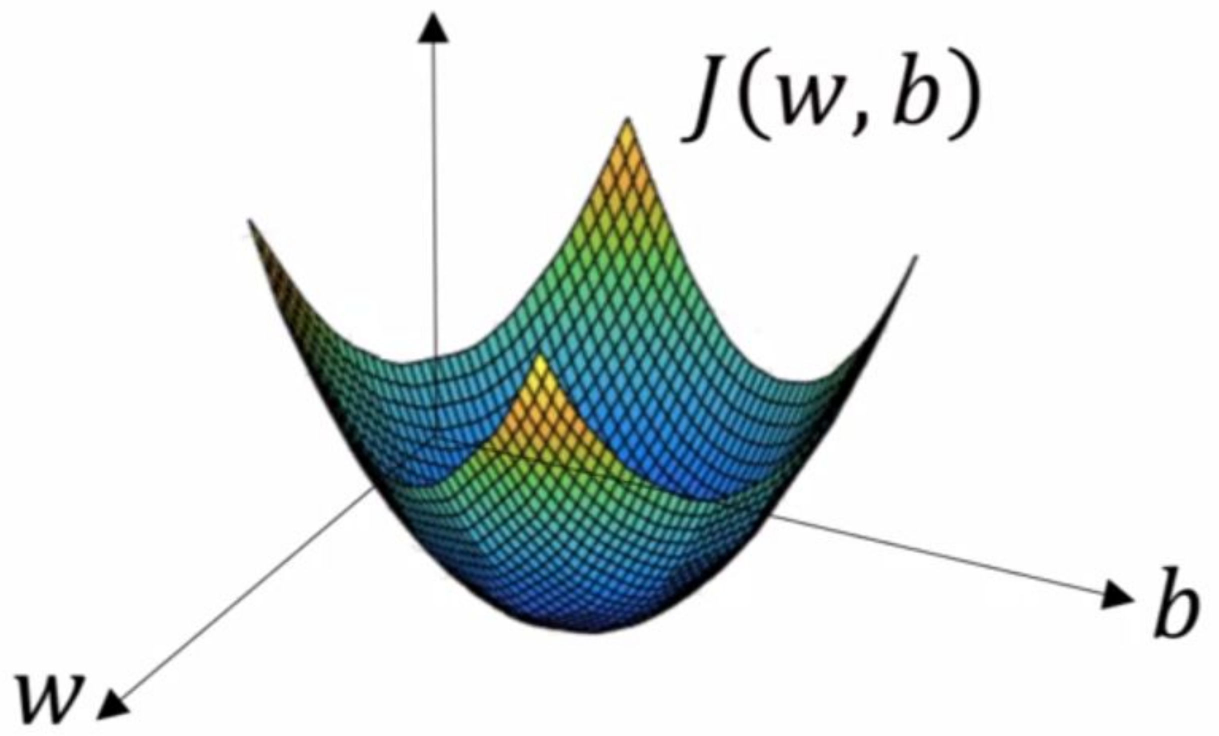
Sources:
- Coursera

3. Gradient Descent

So, we defined a function that works for the gradient descent:

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

*For intuition purposes, **W** is unidimensional*



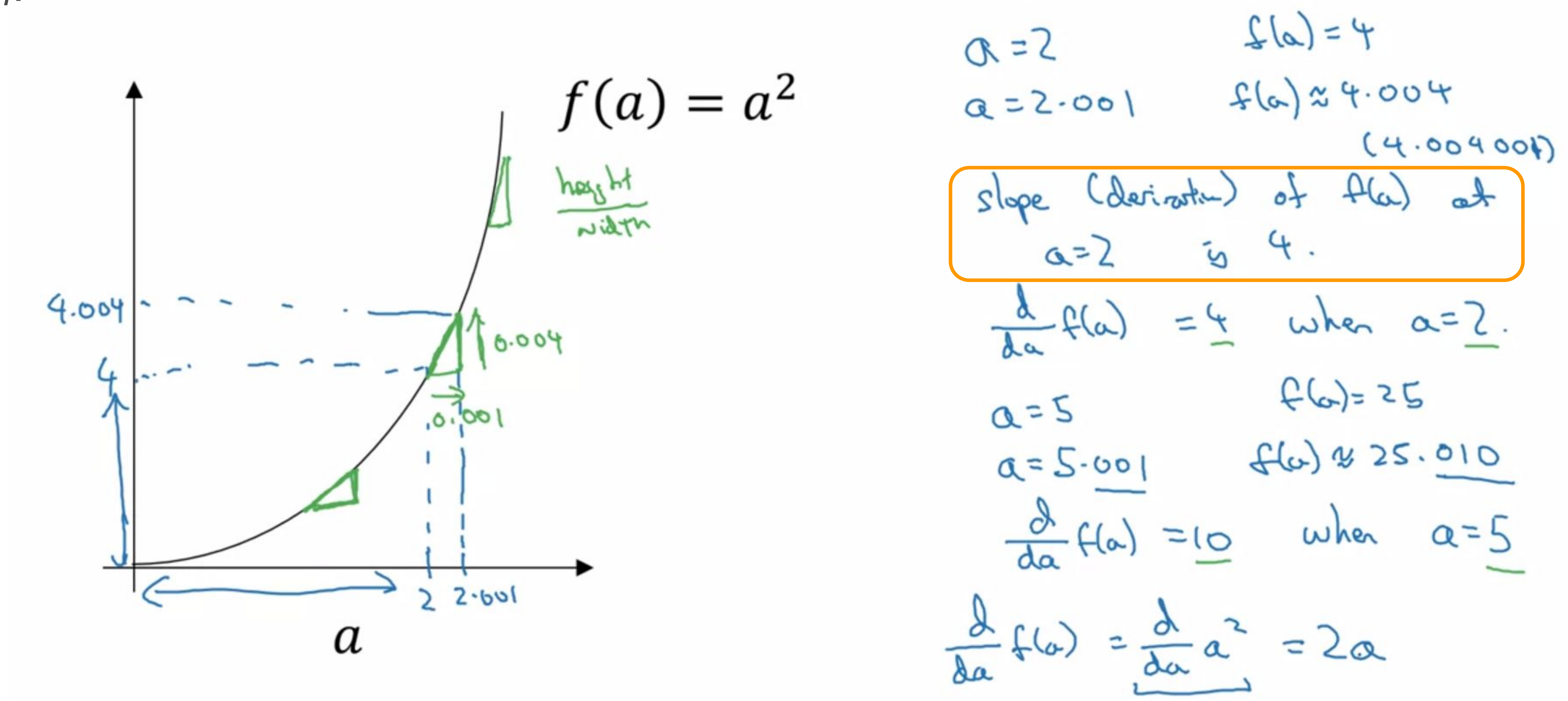
Gradient descent to the rescue! Derivatives! We compute the derivatives to find out the minimum of a function.

Think the derivative as the slope of a function

Sources:
- Coursera

3. Gradient Descent

What is the main intuition about derivatives? *That the derivative is the slope of a function!*



Sources:
- Coursera

3. Gradient Descent

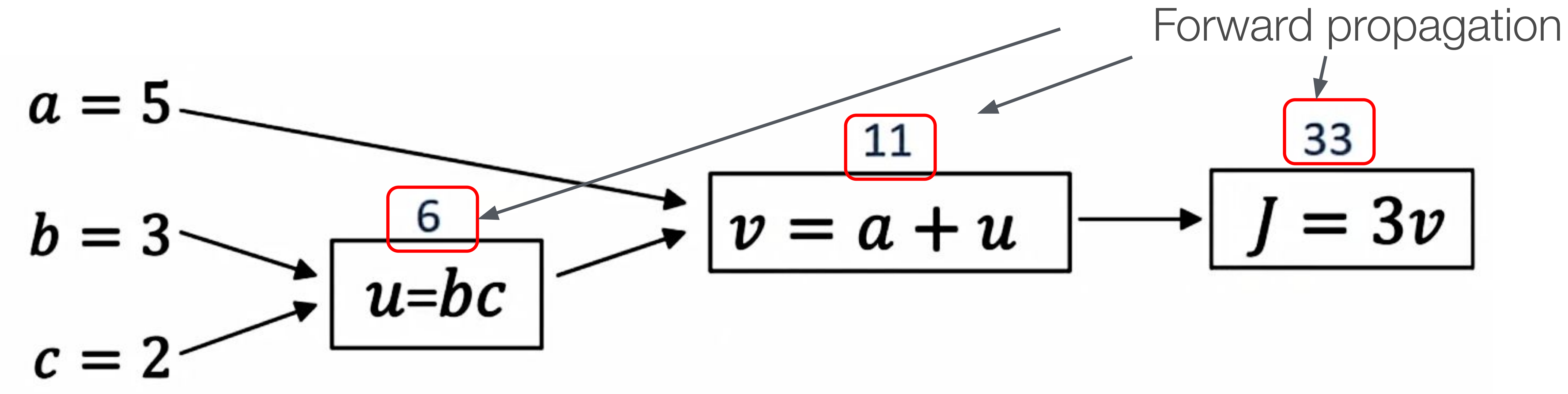
Take messages from the computation of derivatives:

- The derivative of a function is the slope of a function
- The slope of a function can be different at different points of a function
- The computation of a derivative of a function can be done analytically or using graphs

Sources:
- Coursera

3. Gradient Descent

How do we apply this concept to the neural networks?

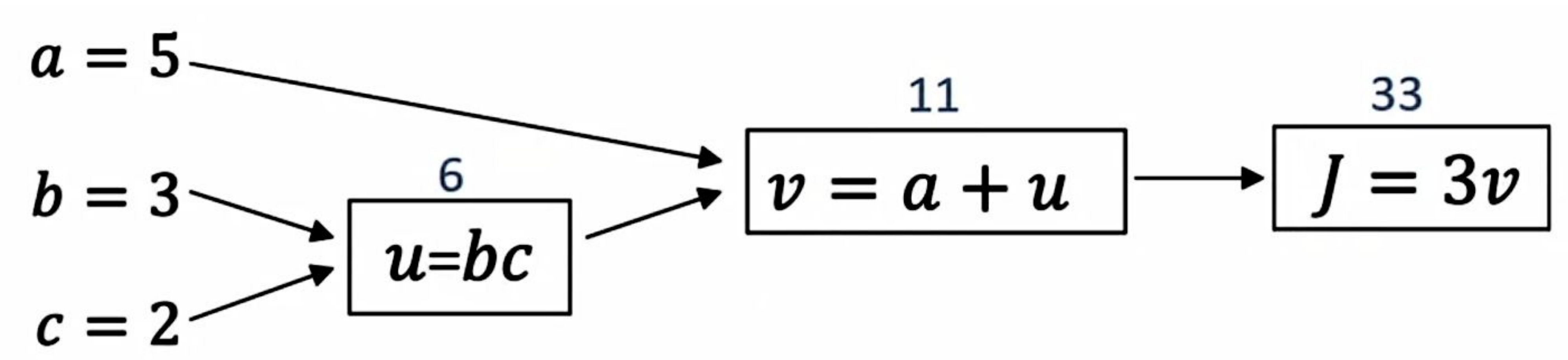


Applying the **chain rule** we can know the derivative w.r.t. a specific variable in the graph

Sources:
- Coursera

3. Gradient Descent

Applying the chain rule we obtained that:



$$\frac{dJ}{du} = 3 = \underbrace{\frac{dJ}{dv}}_{3} \cdot \underbrace{\frac{dv}{du}}_{1}$$

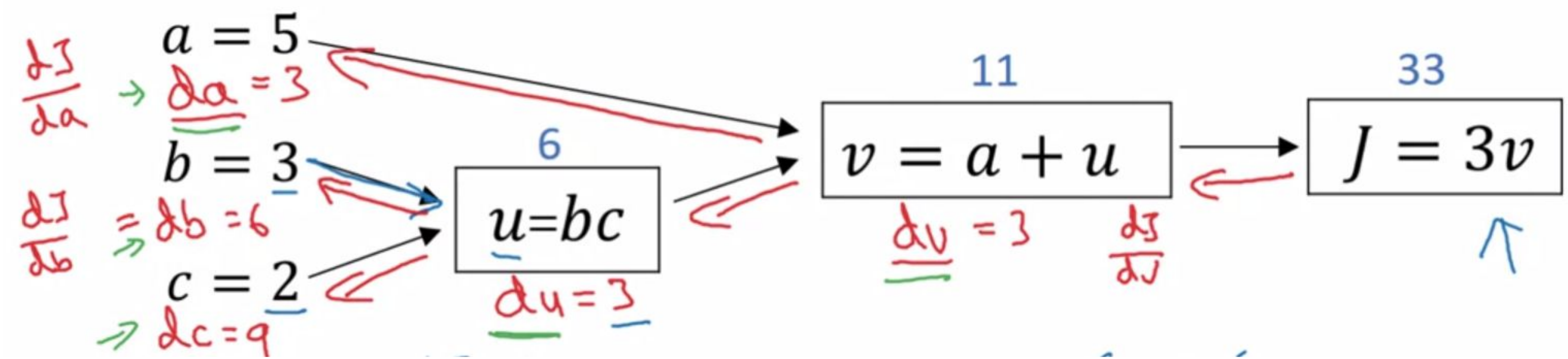
$$\frac{dJ}{db} = \underbrace{\frac{dJ}{du}}_{\rightarrow 3} \cdot \underbrace{\frac{du}{db}}_{=2} = 6$$

$$\frac{dJ}{dc} = \underbrace{\frac{dJ}{du}}_{3} \cdot \underbrace{\frac{du}{dc}}_{\times\, 3} = 9$$

Sources:
- Coursera

3. Gradient Descent

The key message from this is that if we want to compute derivatives, the most efficient way to do this is from right to left

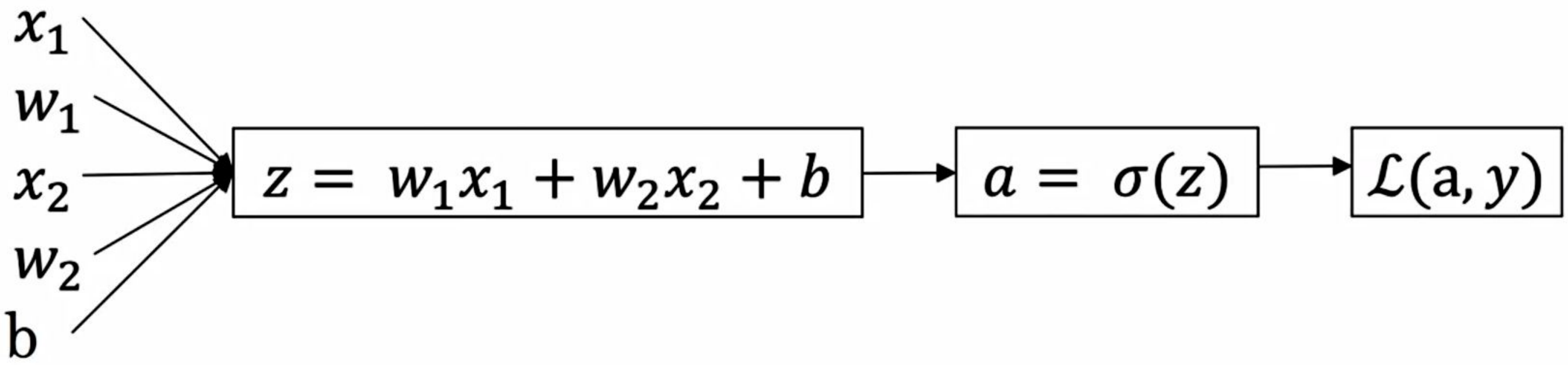


Sources:
- Coursera

3. Gradient Descent

Graph for gradient descent for logistic regression on one sample

$$\left.\begin{aligned} z &= w^T x + b \\ \hat{y} &= a = \sigma(z) \\ \mathcal{L}(a, y) &= -(y \log(a) + (1 - y) \log(1 - a)) \end{aligned}\right\} \textbf{\textit{Forward pass}}$$



Sources:
- Coursera

3. Gradient Descent

Gradient descent for logistic regression on all training samples m:

$$
\begin{aligned}
&\mathtt{J = 0,\ dw_1 = 0,\ dw_2 = 0,\ db = 0} \\
&\mathtt{for\ i = 1\ to\ m:} \\
&\qquad z^{(i)} = w^T x^{(i)} + b \\
&\qquad a^{(i)} = \sigma(z^{(i)}) \\
&\qquad J \mathrel{+}= -[y^{(i)} \log a^{(i)} + (1 - y^{(i)}) \log(1 - a^{(i)})] \\
&\qquad dz^{(i)} = a^{(i)} - y^{(i)} \\
&\qquad dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)} \\
&\qquad dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)} \\
&\qquad db \mathrel{+}= dz^{(i)} \\
&\mathtt{J = J/m,\ dw_1 = dw_1/m,\ dw_2 = dw_2/m} \\
&\mathtt{db = db/m}
\end{aligned}
$$

Forward pass

Backward pass

Sources:
- Coursera

4. Vectorization

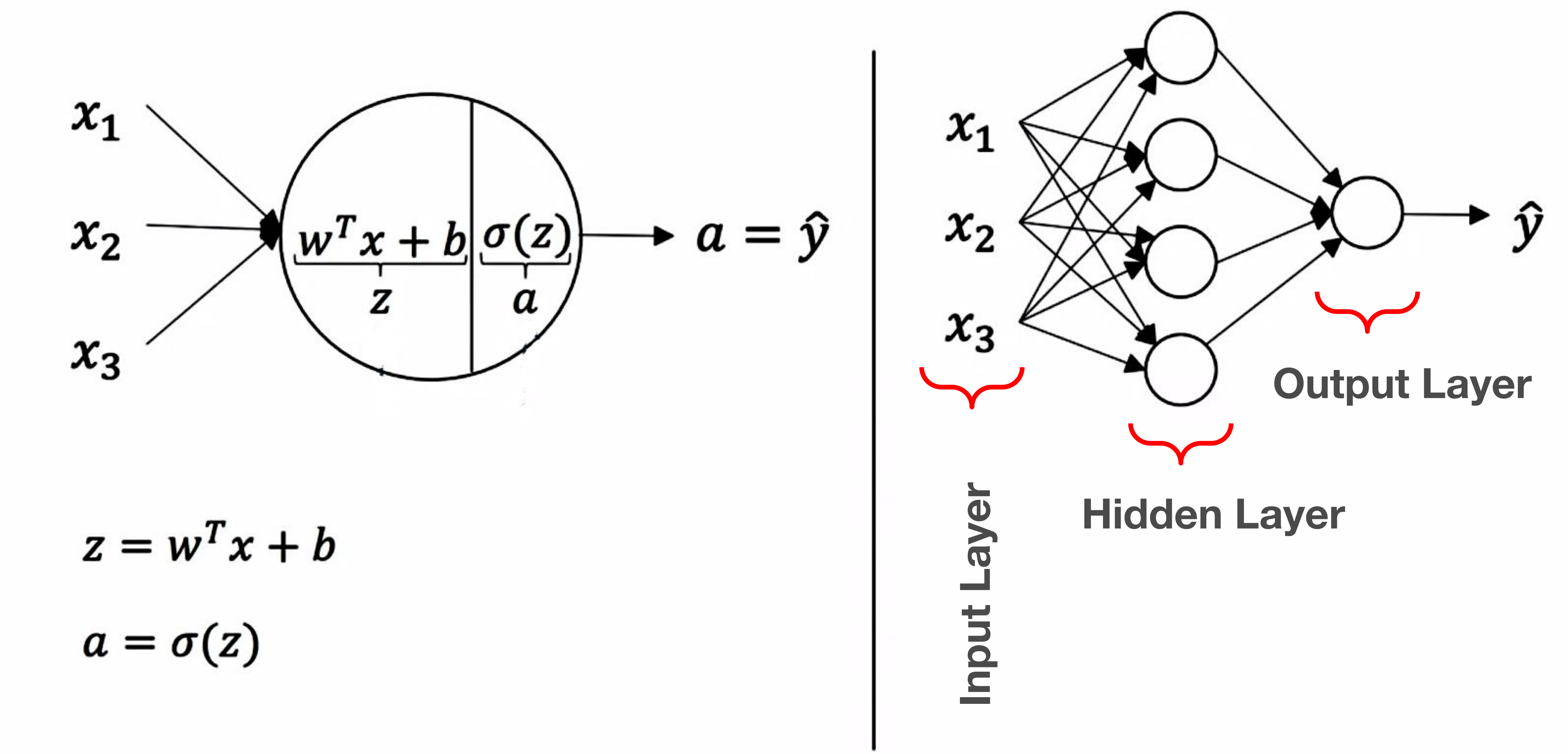
How we can do to take advantage of GPUs to compute gradient descent?

A neural network (NN) with two layers is basically one logistic regression follow by another logistic regression! A NN with three layers is a logistic regression repeated 3 times ...

Sources:
- Coursera

4. Vectorization

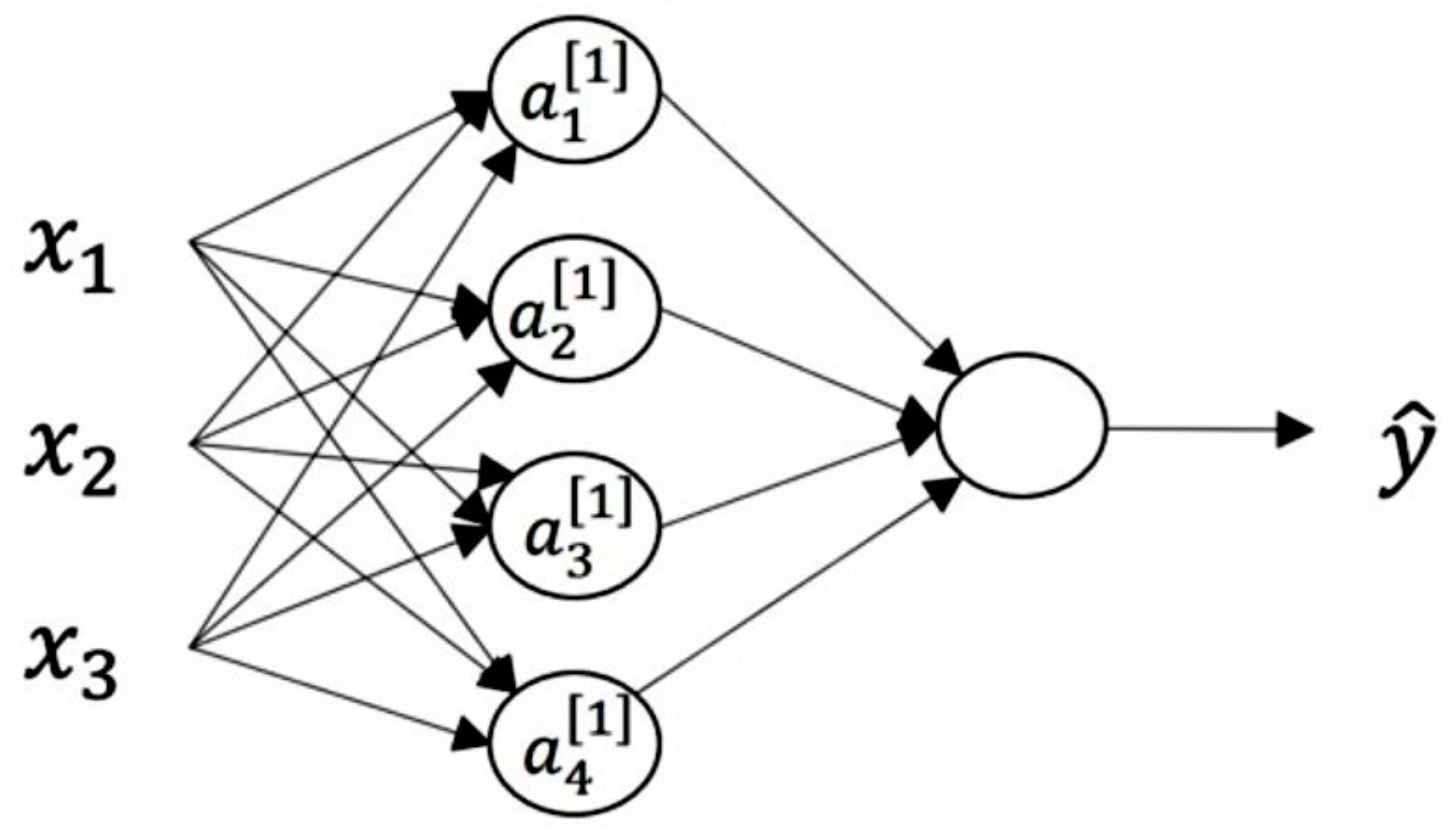
From the logistic regression to a neural network (NN)



4. Vectorization

From the logistic regression to a neural network (NN).

In these equations, the index in square parenthesis represents the layer



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

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

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

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

Sources:
- Coursera

4. Vectorization

From the logistic regression to a neural network (NN).

In these equations, the index in square parenthesis represents the layer

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

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

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

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

Computing these equations using a for loop is really inefficient!

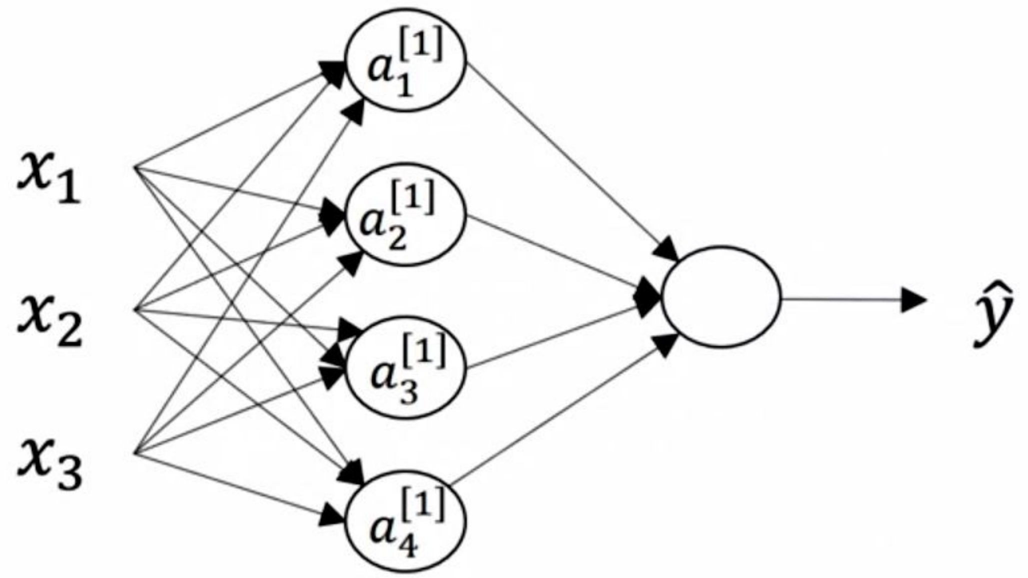
The more logistic units (neurons) we have the bigger is the for loop

For that reason vectorizing the weights, biases and Z values is important to increase performance

Sources:
- Coursera

4. Vectorization

Vectorizing previous equations resulted on similar logistic regression equations. The only difference is that the ***weights of the hidden layer are represented by a matrix of size (4,3)***, the biases by a matrix of size (4,1) and the zs by a matrix of (4,1)



Given input x:

Hidden Layer

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

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

Output Layer

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

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

Hidden layer

Weights: 4 neurons, 3 inputs each

Biases: 4 values. One for each neuron

Output layer

Weights: 1 neurons, 4 inputs each

Biases: 1 value

Sources:
- Coursera

4. Vectorization

So how can we compute **z** and **a** values for m samples in the training set?
Vectorizing across multiple samples!

for i = 1 to m:

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

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

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

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

$z^{[1](i)}$ *-> Columns represent training samples and rows represent hidden units*

Sources:
- Coursera

5. Activation functions

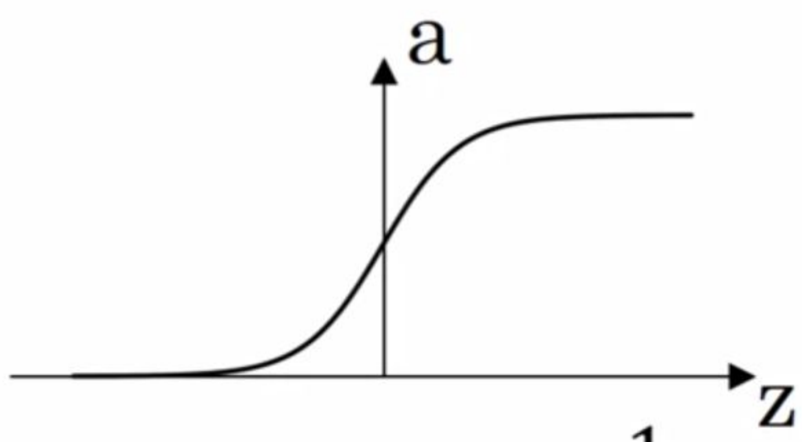
When building a neural network, one of the important parts that you need to decide if the activation function.

So far we have been using the sigmoid activation function, but it’s been demonstrated that it is not the best option for hidden layers. **Tanh or ReLU (rectifier linear unit)** are more used and have better performance than the sigmoid in the hidden layers.

Sigmoid function is used in the output layer when classifying two classes (Binary classification)

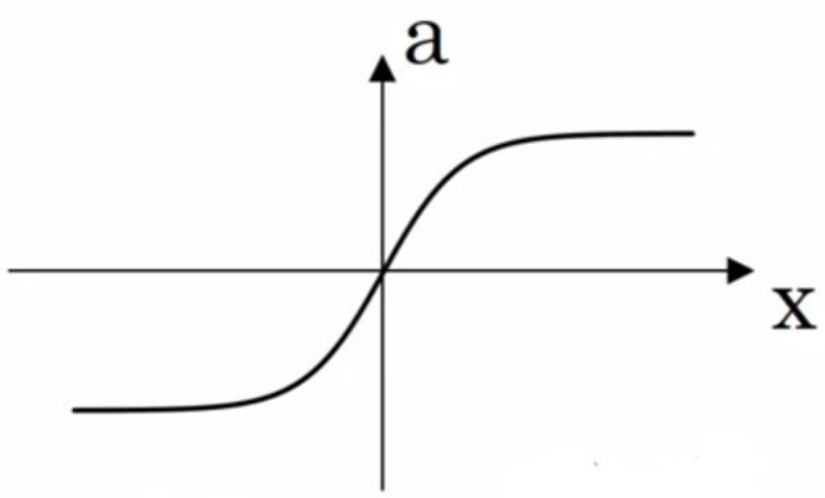
Sources:
- Coursera

5. Activation functions

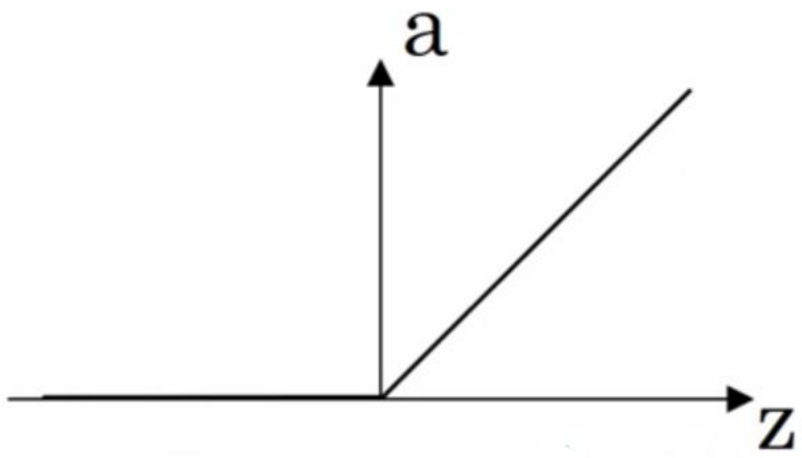


sigmoid: $a = \dfrac{1}{1 + e^{-z}}$

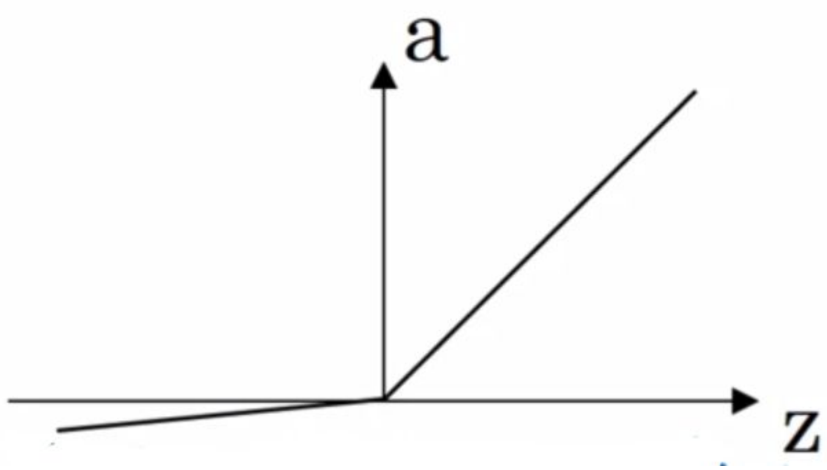
Tanh



ReLU



Leaky ReLU



Sources:
- Coursera

5. Activation functions

Why neural networks need non-linear activation functions?

The most important reason for using non-linear activation functions is that they allow the NN to learn **non-linear mapping/functions between the input and the output!**

Otherwise, using linear activation function will only make the NN to learn **linear functions between input and output**

Sources:
- Coursera

5. Activation functions

Remember that in order to implement gradient descent, we need **the derivatives of the activation functions**. Here are some of them:

Name	Equation	Derivative
Sigmoid	$f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Tanh	$f(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$	$f'(x) = 1 - f(x)^2$
Rectified Linear Unit (relu)	$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Leaky Rectified Linear Unit (Leaky relu)	$f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0.01 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$

Sources:
- Coursera
- https://engmrk.com/activation-function-for-dnn/

6. Backpropagation

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

$$W^{[2]},\ b^{[2]} \rightarrow z^{[2]}$$

Output Layer

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

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

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

Hidden Layer

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

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

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

In these equations, dz actually means:

$$dz = \frac{\delta L}{\delta z} = \frac{\delta L}{\delta a}\frac{\delta a}{\delta z}$$

Sources:
- Coursera

7. Random Initialization

Randomly initializing the weights in a neural network is highly beneficial for the gradient descent computation.

There are different methods developed in the literature presenting different ways of randomly initialized the weights.

Easy to do and highly beneficial!

Sources:
- Coursera

Let’s move to Google Colab!

Notebooks:

- *0_Numpy_Basic_Functions.ipynb*
- *1_First_Neural_Network.ipynb*
- *2_Neural_Network_with_One_Hidden_Layer.ipynb*

Sources:
- Coursera