Embedded Machine Learning for Edge Computing

Introduction to Machine Learning with Hands on Experience using Google Colab

Pamuditha Somarathne

June/2024

Your Instructor

Pamuditha Somarathne

Fields: Machine Learning, Human-Computer Interaction

2018 - 2023: BSc. Engineering (Electronics and Telecommunication Engineering), University of Moratuwa, Sri Lanka

2022 - 2023: Research Affiliate, School of Computer Science, The University of Sydney, Australia

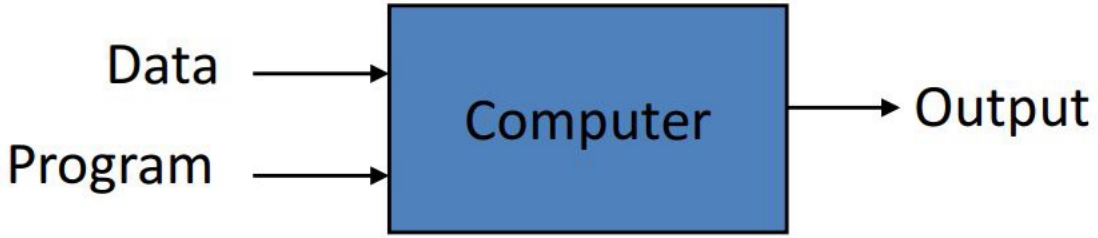
2024 – Present: PhD Candidate, School of Computer Science, The University of Sydney, Australia

Content

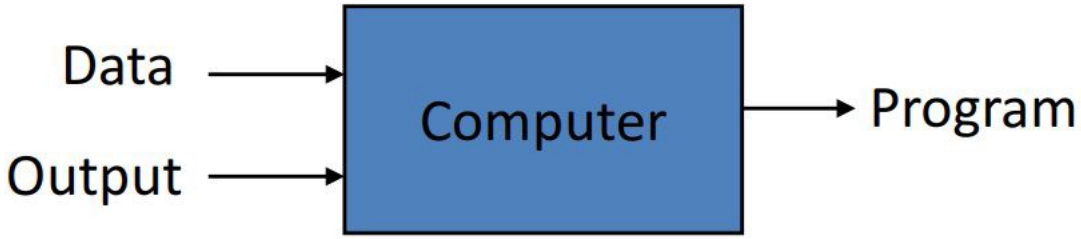
1. Recap
    1. Introduction to Machine Learning
    2. Traditional Programming Vs. Machine Learning
2. Introduction to Deep Learning
    1. Colab Hands-on Sessions
        1. Part 1: Notebook
        2. Part 2: Notebook
        3. Part 3: Notebook
3. Assignment: Detect Hand Gestures

Traditional vs ML

Traditional Programming



Machine Learning



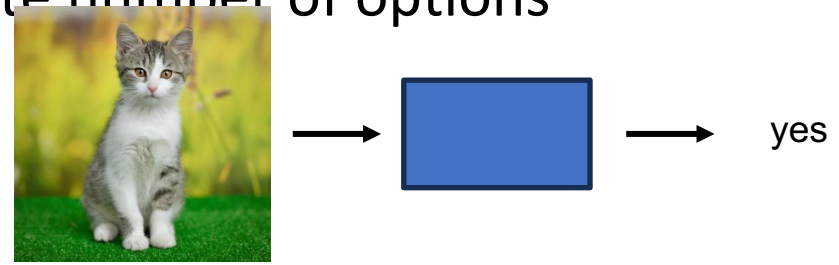
Supervised Learning

In supervised learning, we provide the neural network with both inputs and targets (correct results). The model learns to predict outputs that are closer to the target.

Two types of supervised learning

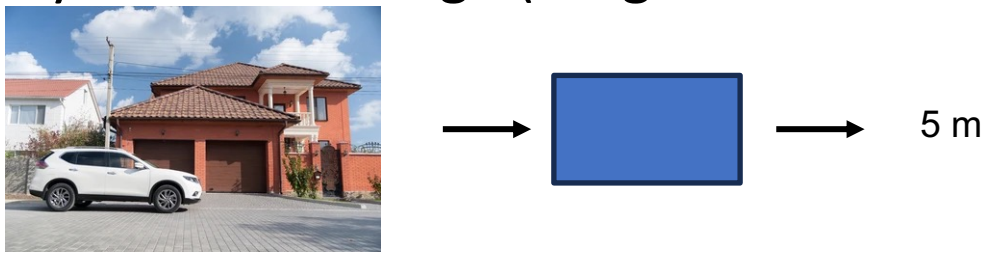
- Classifications: picking from a finite number of options

  “Does this figure contain a cat?”

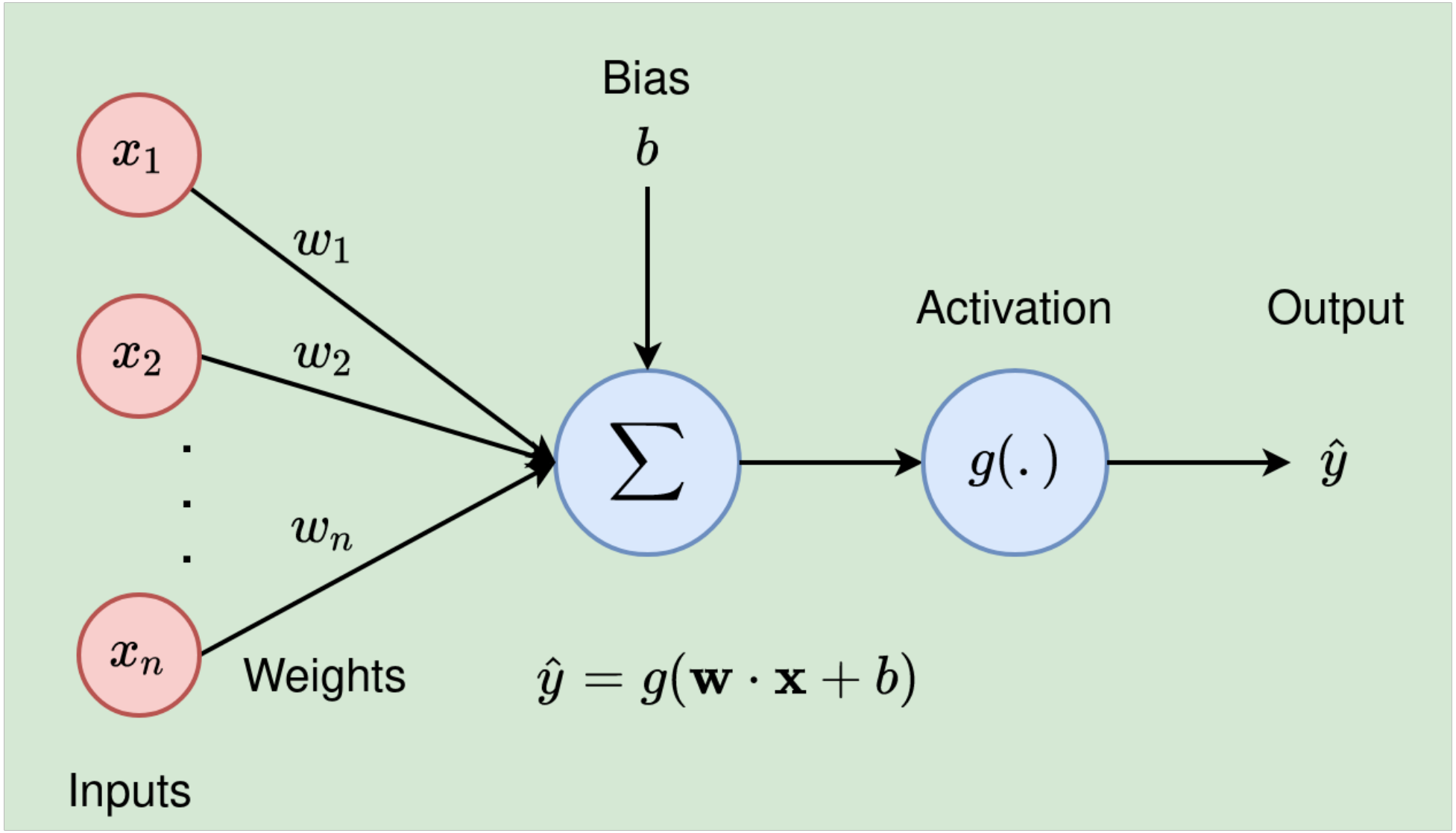


- Regression: predictions can take any value in a range (range could be infinite as well)

  “How far is the house from the car?”



Inside a Neuron



Why ML?

Non-intuitive problems & solutions: we don't know how to instruct on solving these problems

- Vision
- Speech

Highly complex/expensive/specific systems

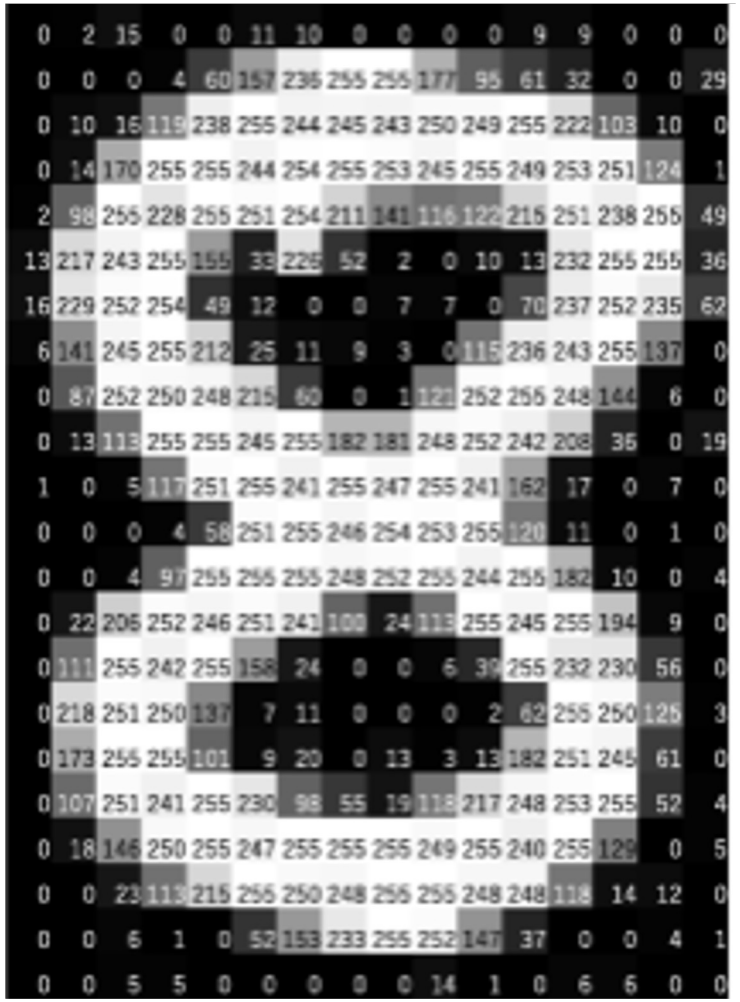
Adaptability, Customizability, Personalization

Mimic/replace humans in repetitive tasks

Perform better than human counterparts and human programmers

What an image is?

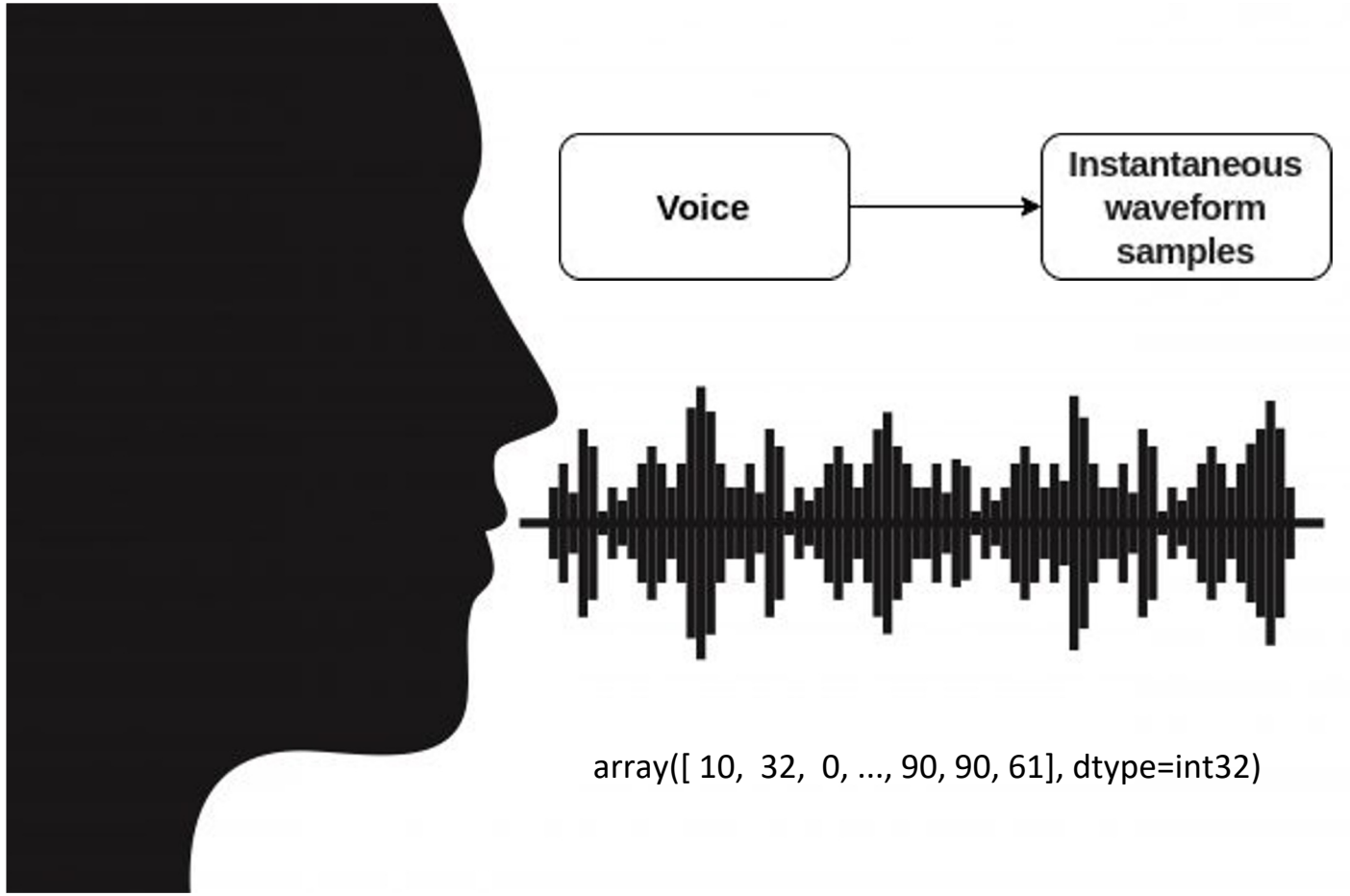
What number is this?



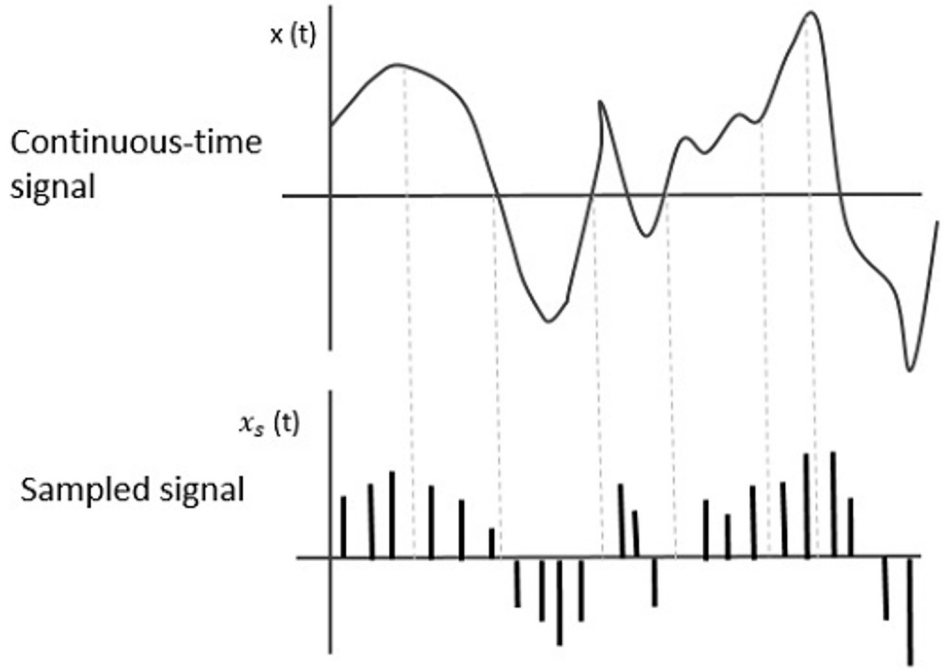
0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4
0	18	146	250	255	247	255	255	255	249	255	240	255	129	0	5
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0

What an audio is?

Trigger Apple's Personal Assistant with 'Hey Siri'



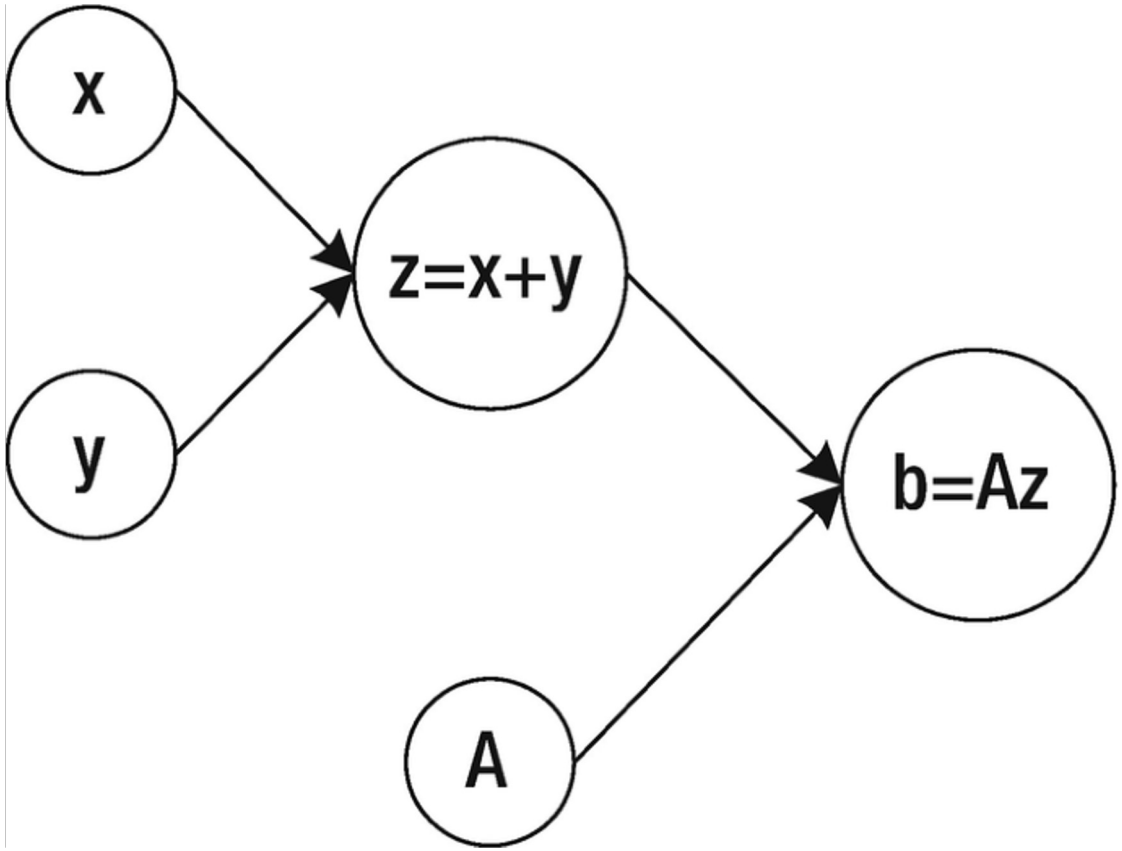
array([10, 32, 0, ..., 90, 90, 61], dtype=int32)



Example 1 - Hands on Session

Link to the Notebook:
https://colab.research.google.com/drive/1stc_ozMdd2MubQzV61ThZNyF9l3LPZBI

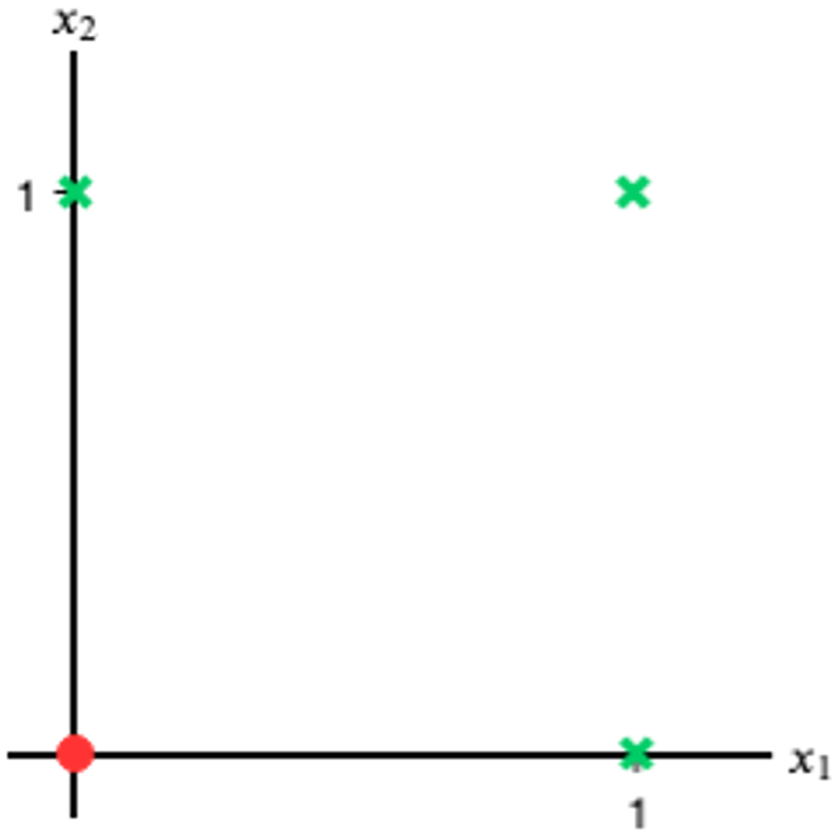
Computational Graph



- They are a form of directed graphs that represent a mathematical expression.
- Thus, a Neural Network can be defined as a Computational Graph.

Understanding a DL Training Pipeline

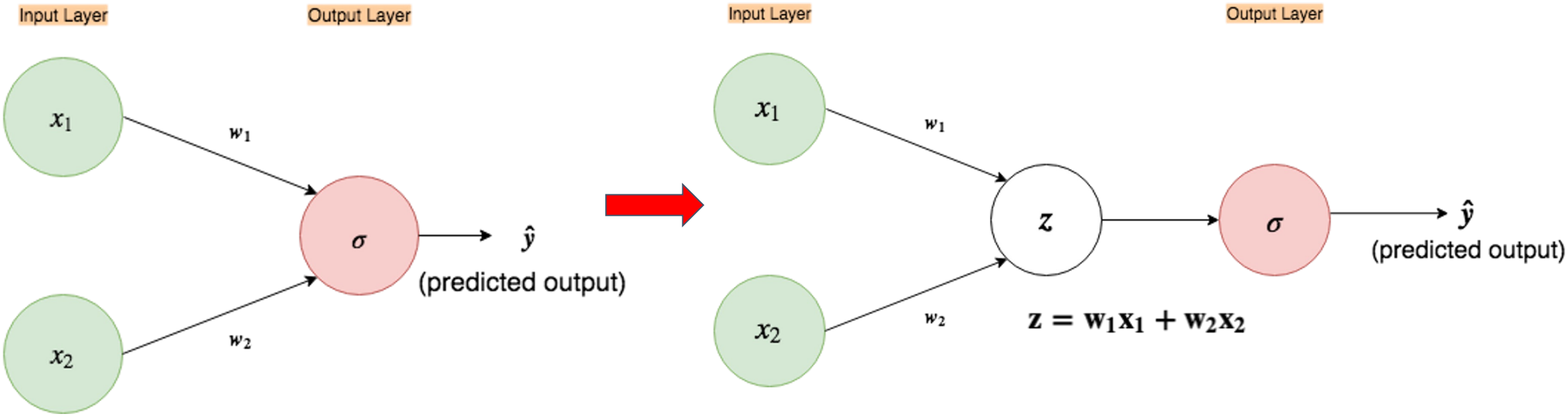
Dataset



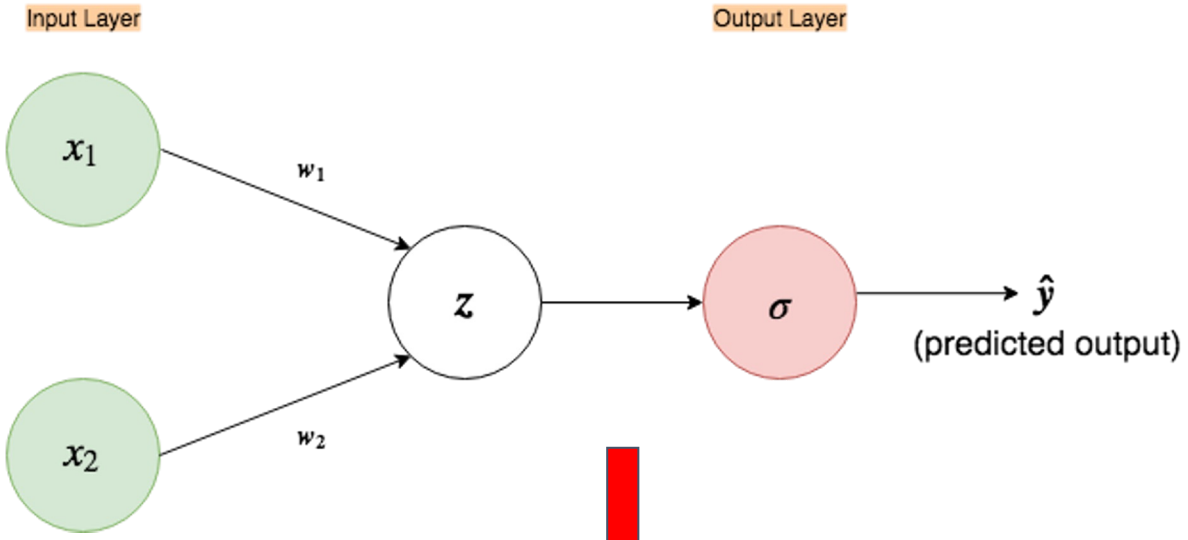
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

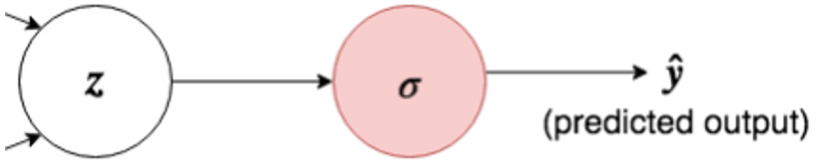
Data represents OR Gate

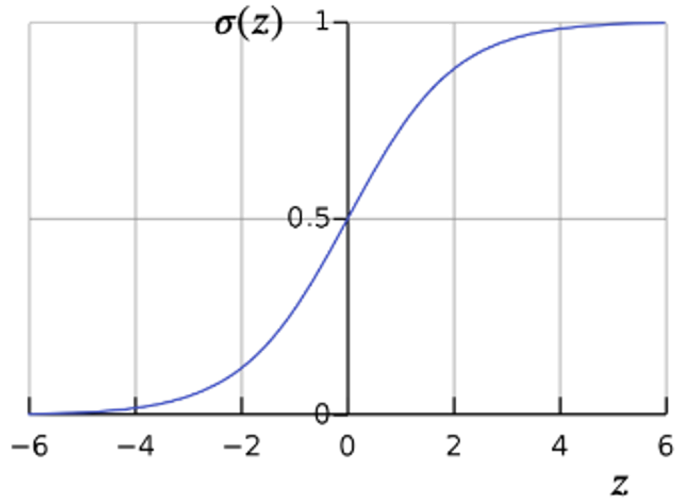
Single Neuron NN

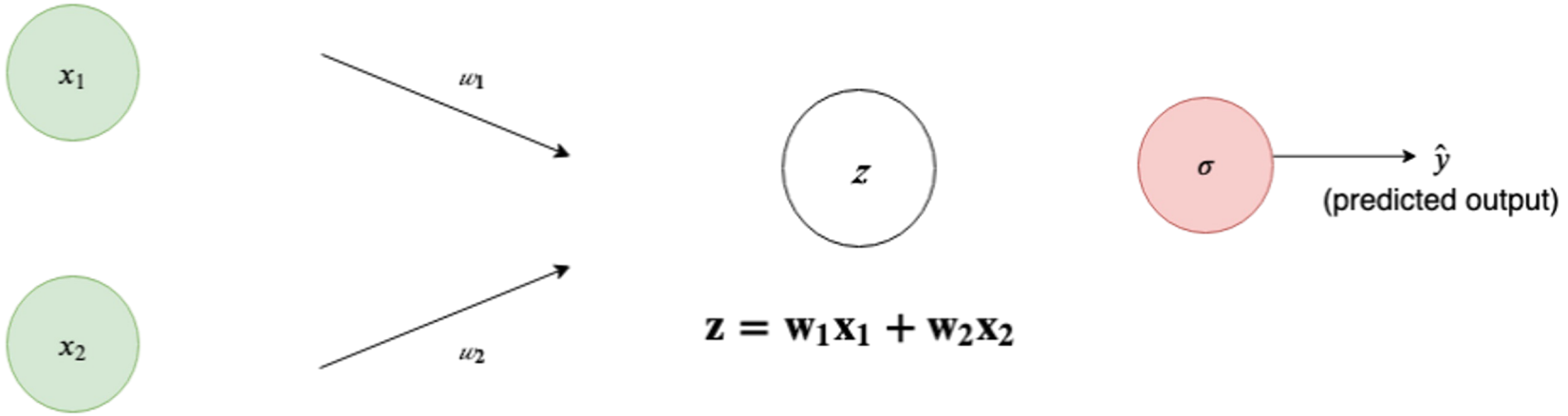


Understanding a DL Training Pipeline







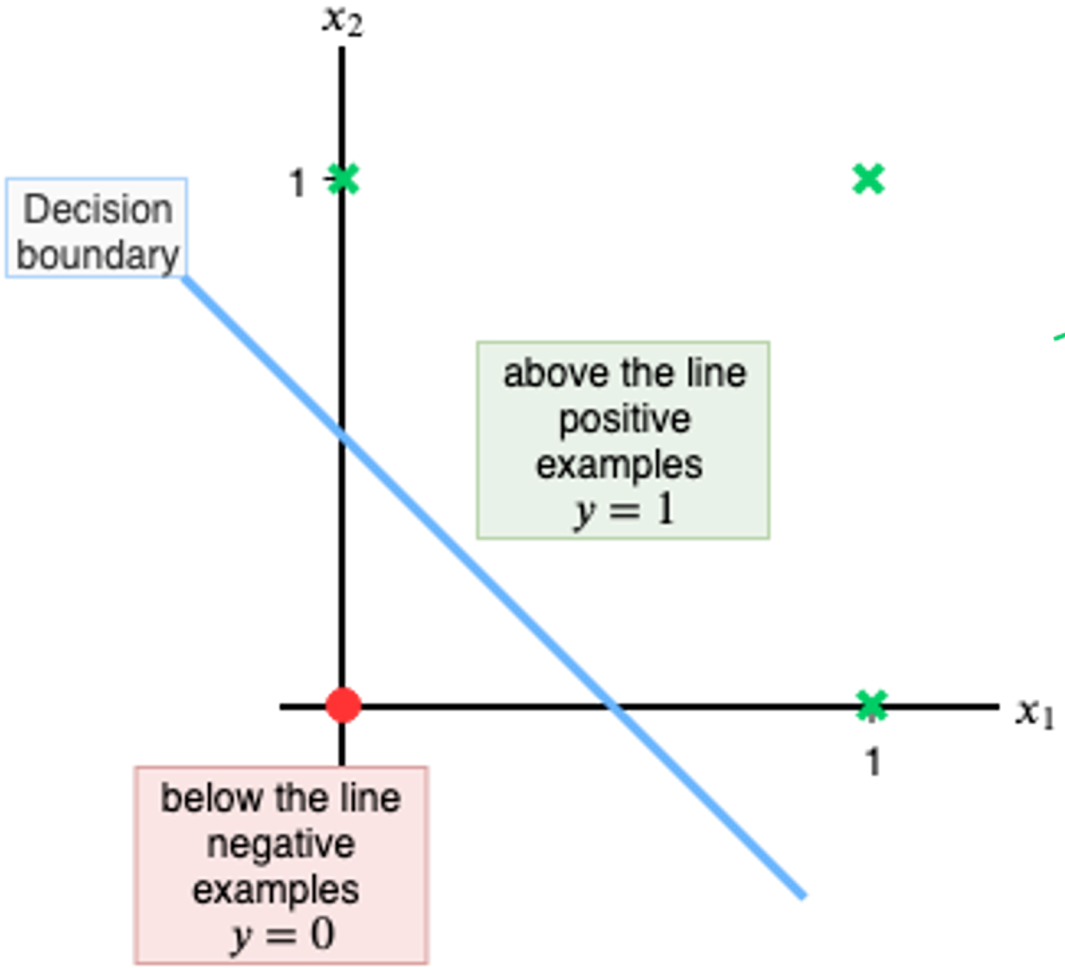


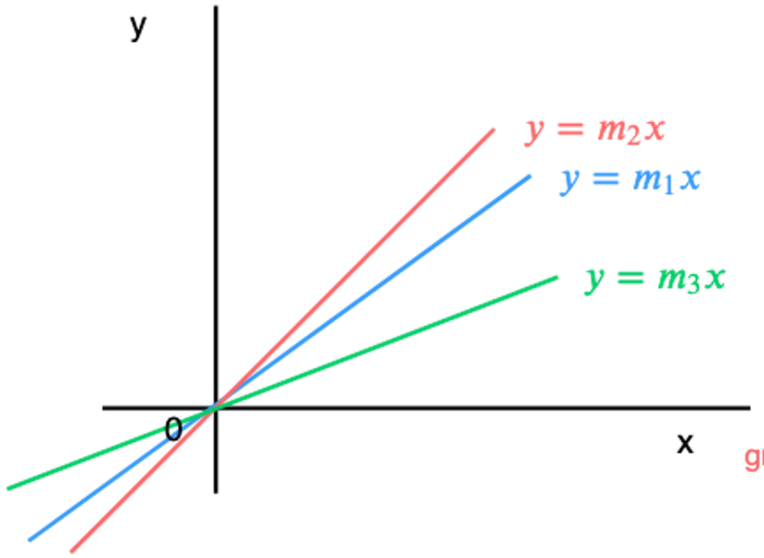
$$z = w_1x_1 + w_2x_2$$

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

Why we need a bias ?

Expected Output

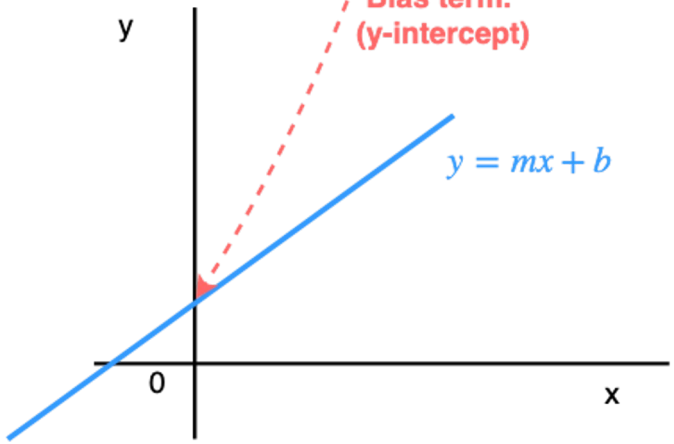




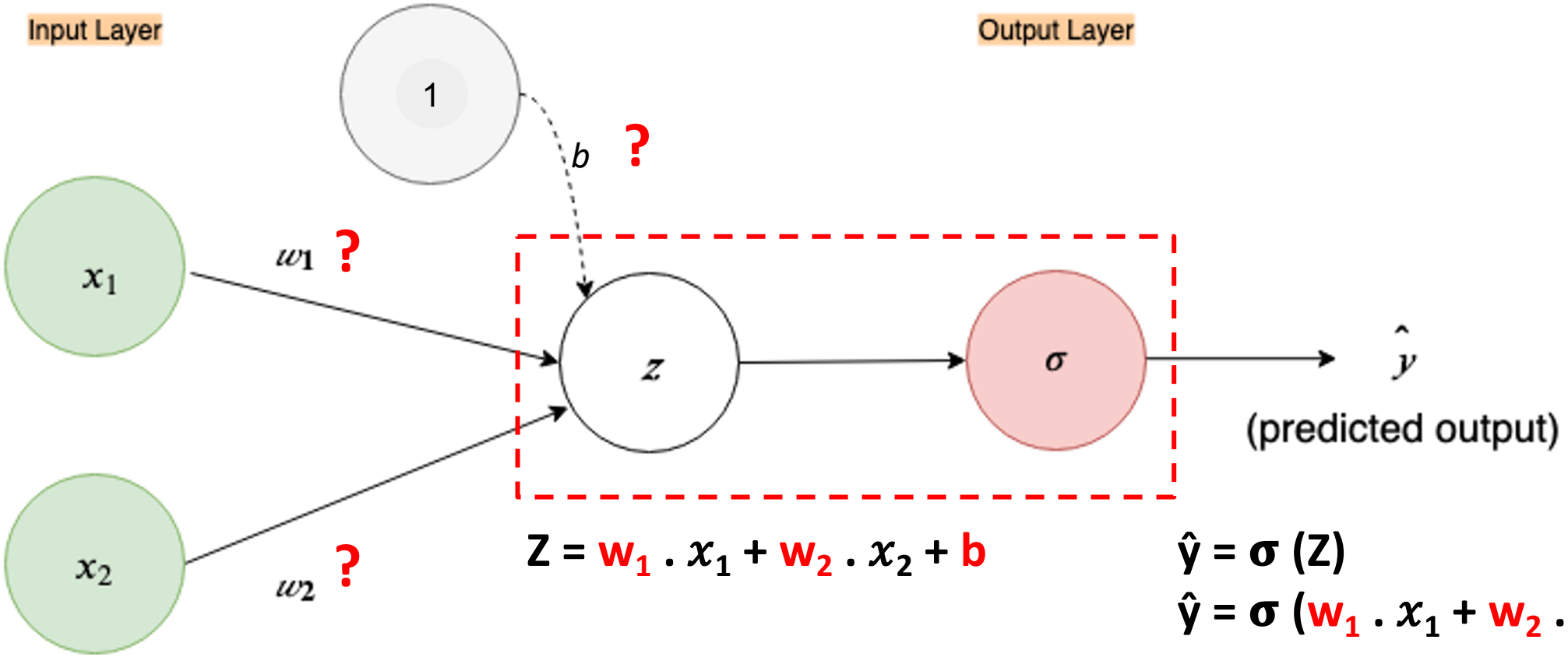
gradient(slope)

$y = mx + b$

Bias term. (y-intercept)



Understanding a DL Training Pipeline

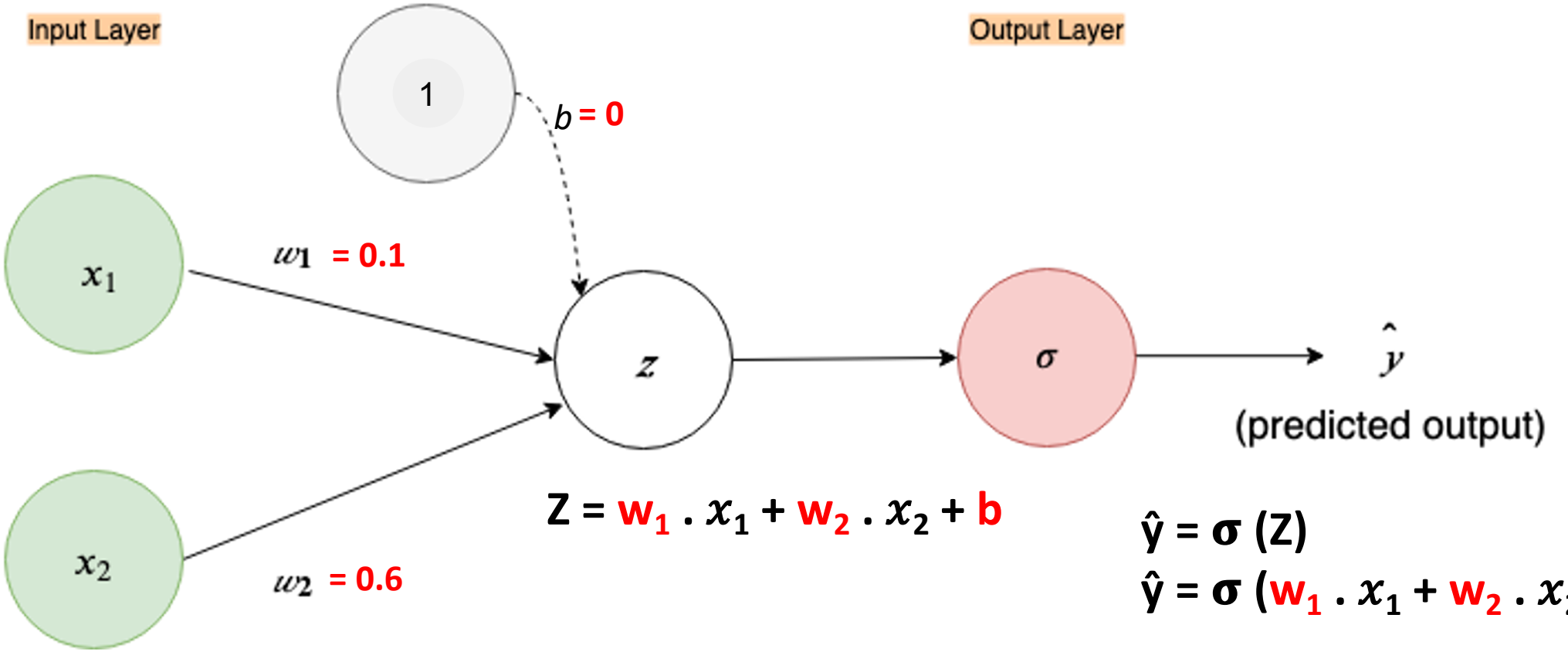


$Z = w_1 . x_1 + w_2 . x_2 + b$

$\hat{y} = \sigma(Z)$

$\hat{y} = \sigma(w_1 . x_1 + w_2 . x_2 + b)$

STEP1: Initialization of Weights & Biases

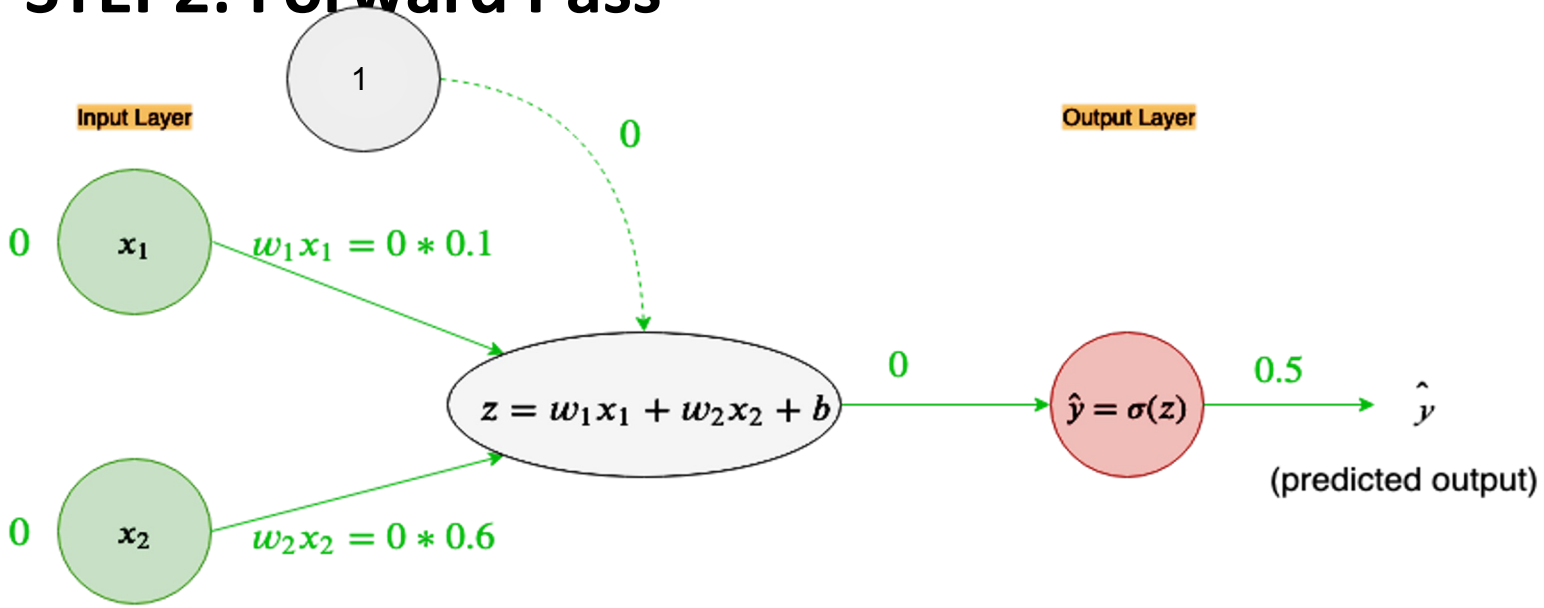


$$Z = w_1 \cdot x_1 + w_2 \cdot x_2 + b$$

$$\hat{y} = \sigma(Z)$$

$$\hat{y} = \sigma(w_1 \cdot x_1 + w_2 \cdot x_2 + b)$$

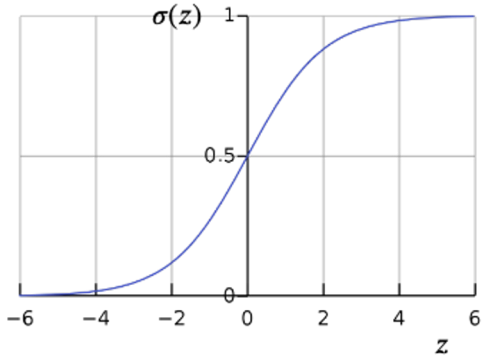
STEP2: Forward Pass



The Neural Network is going through the following computations.

Forward Computations are marked in Green

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1



- Our input for first example $x_1 = 0, x_2 = 0$
- Recall our randomly initialized weights $w_1 = 0.1, w_2 = 0.6$.
- We'll initialize our bias to be zero, $b = 0$
- $z = w_1 x_1 + w_2 x_2 + b = 0 * 0.1 + 0 * 0.6 + 0 = 0$
- $\hat{y} = \sigma(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{0}} = 0.5$

STEP3: Calculating the Loss

Tells us,
how far our neural network's predicted output($\hat{y}$) is from our desired output(y),

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

Where $\mathbf{y}$ is the actual desired output, and
$\hat{\mathbf{y}}$ is the predicted output

squared-error Loss function

For current example $\hat{y} = 0.5$ and $y = 0$,

$$Loss = L(\hat{y}, y) = \frac{1}{2}(y - \hat{y})^2 = \frac{1}{2}(0 - 0.5)^2 = \frac{1}{2}(-0.5)^2 = \frac{1}{8} = 0.125$$

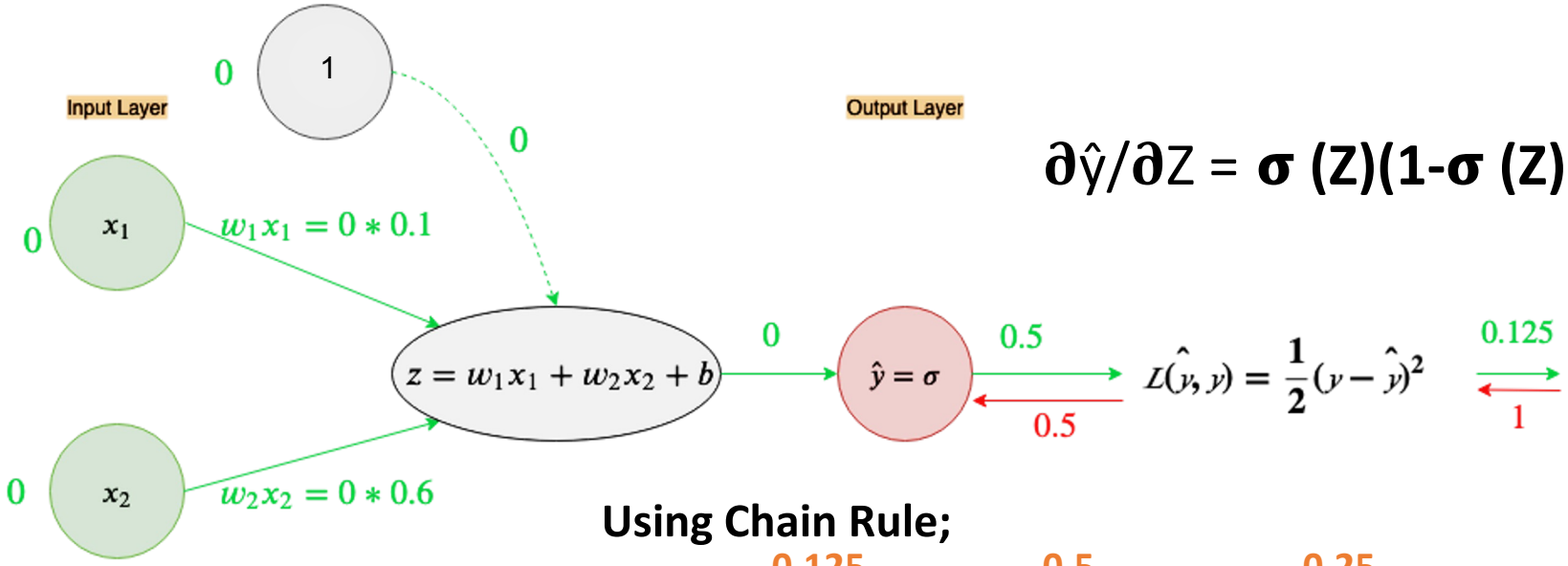
$$\begin{aligned} Cost &= C(L(y^{(i)}, \hat{y}^{(i)})) \\ &= \frac{1}{m}\sum_{i=1}^{m} L(y^{(i)}, \hat{y}^{(i)}) \\ &= \frac{1}{m}\sum_{i=1}^{m} \frac{1}{2}(y^{(i)} - \hat{y}^{(i)})^2 \\ &= \frac{1}{2m}\sum_{i=1}^{m} (y^{(i)} - \hat{y}^{(i)})^2 \end{aligned}$$

$i \rightarrow$ is the i^{th} training example
$m \rightarrow$ is the total number of training examples
$L(y^{(i)}, \hat{y}^{(i)}) \rightarrow$ is the loss in the i^{th} training example

STEP4: Backward Pass

The Neural Network is going through the following computations.

Backward Computations are marked in Red



$$\partial\hat{y}/\partial Z = \sigma(Z)(1-\sigma(Z))$$

The *Local gradient* at $L(\hat{y}, y)) = \frac{1}{2}(y - \hat{y})^2$ still remains the same:

$$\frac{\partial L}{\partial \hat{y}} = -(y - \hat{y})$$

Recall, $\hat{y}$ for current example is $\hat{y} = 0.5$ and $y = 0$. So, numercal vale of local gradient also remains same:

$$\frac{\partial L}{\partial \hat{y}} = -(0 - 0.5) = 0.5$$

Using Chain Rule;

0.125 0.5 0.25

$$\partial L/\partial Z = \partial L/\partial\hat{y} * \partial\hat{y}/\partial Z$$

0.0 0.125 0.0

$$\partial L/\partial W_1 = \partial L/\partial Z * \partial Z/\partial W_1$$

0.0 0.125 0.0

$$\partial L/\partial W_2 = \partial L/\partial Z * \partial Z/\partial W_2$$

0.125 0.125 1.0

$$\partial L/\partial b = \partial L/\partial Z * \partial Z/\partial b$$

STEP5: Updating the Weights

General Equation for Gradient Descent

Optimization Algorithm

$$w_{new} = w_{old} - \alpha \frac{\partial L}{\partial w}$$

Learning Rate

For $\alpha = 1$,

$$b = b - 1 \frac{\partial L}{\partial b}$$

1 step

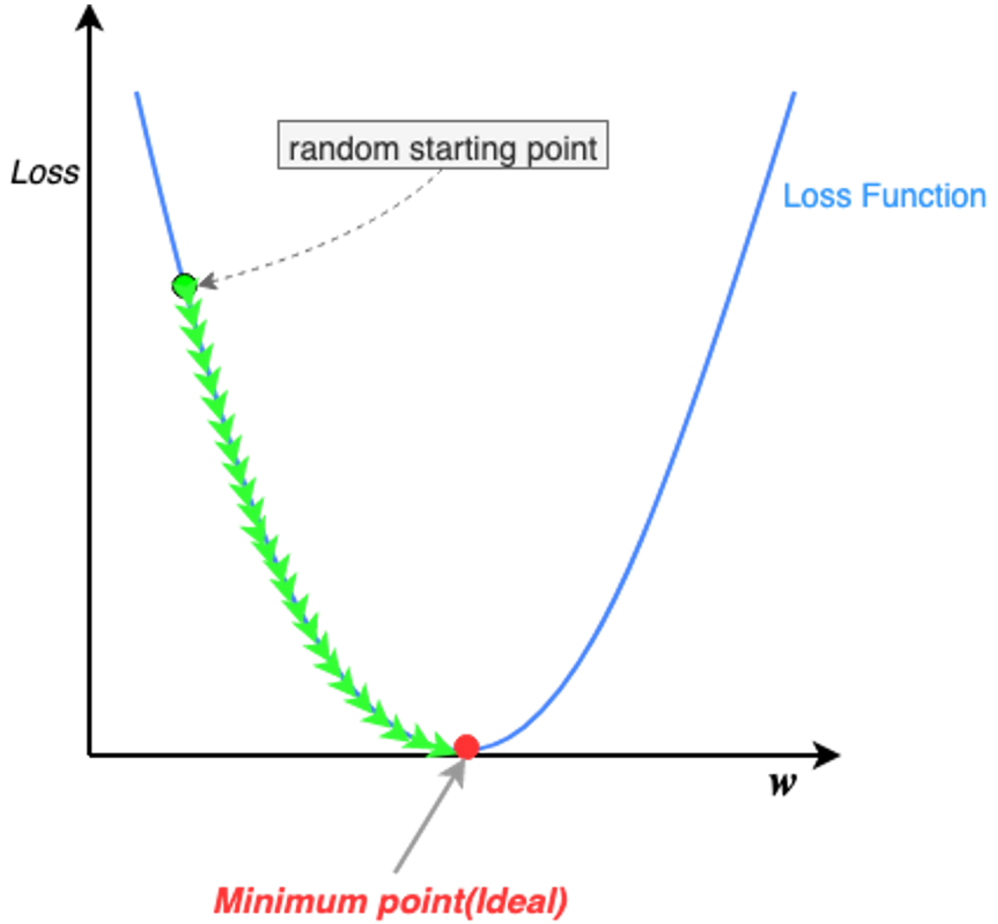
To calculate new bias we do the following:

Recall, current bias, $b = 0$ and $\frac{\partial L}{\partial b} = 0.125$

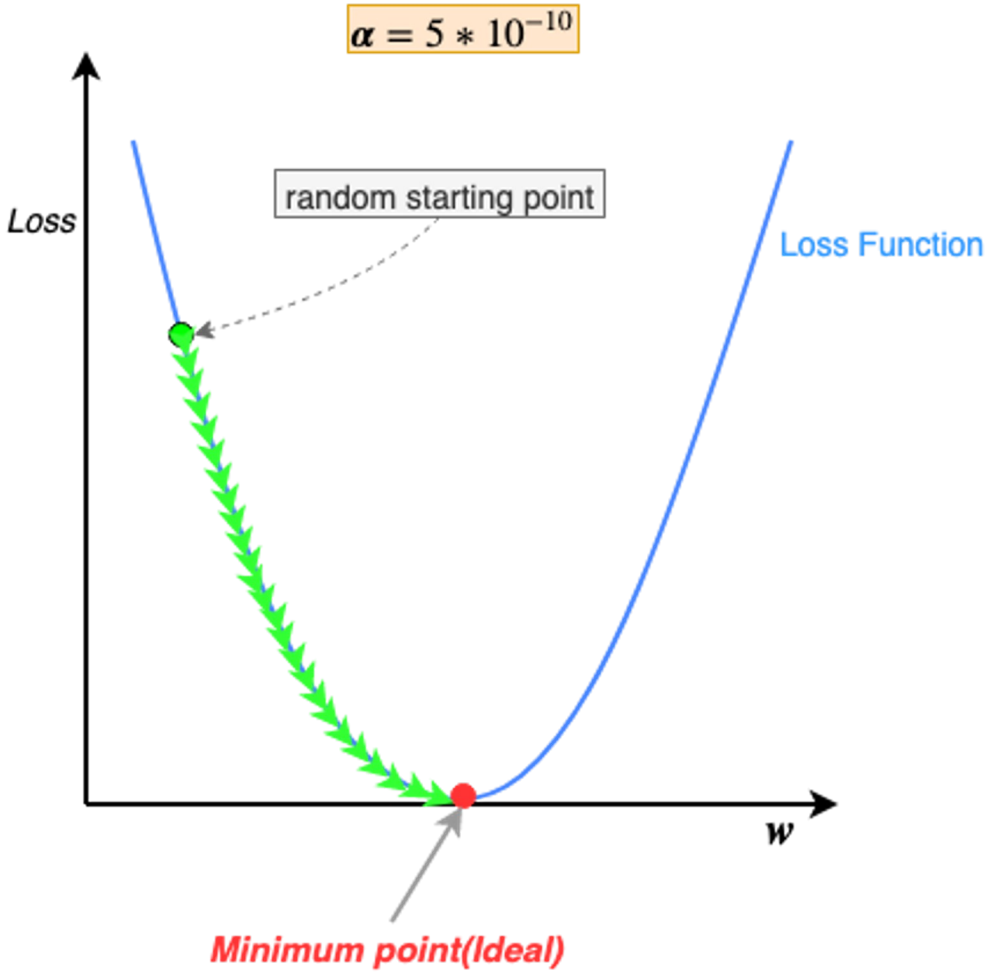
The new bias is:

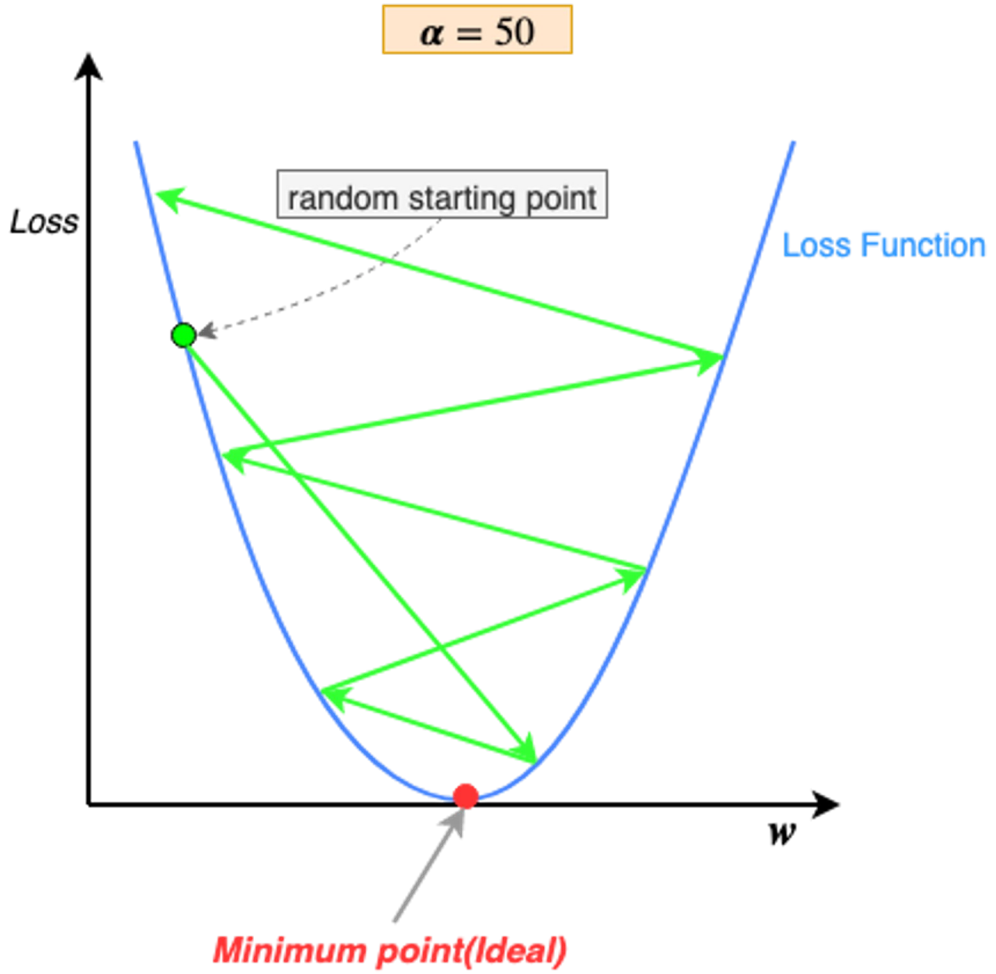
$$b = b - \frac{\partial L}{\partial b} = 0 - 0.125 = \mathbf{-0.125}$$

STEP5: Updating the Weights



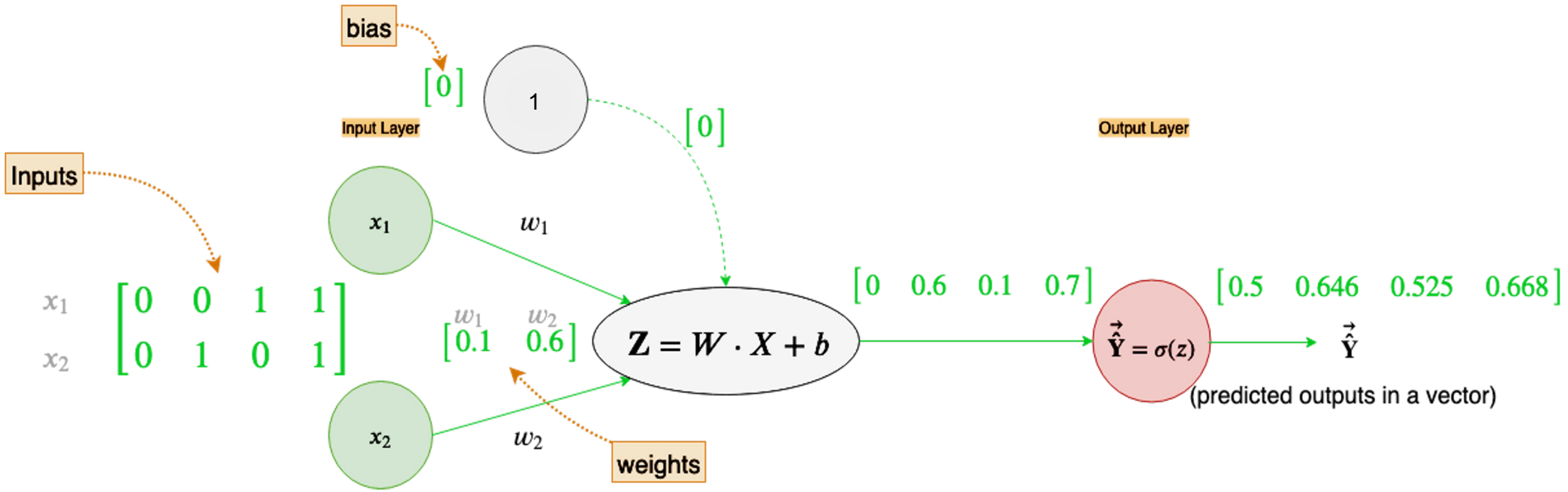
STEP5: Updating the Weights





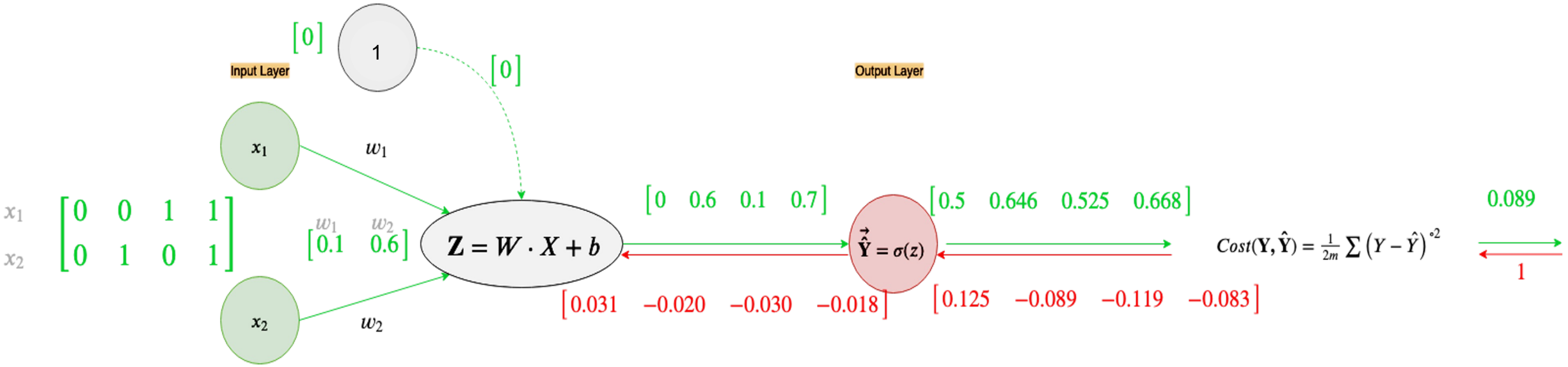
Actual Training Cycle

Forward Pass



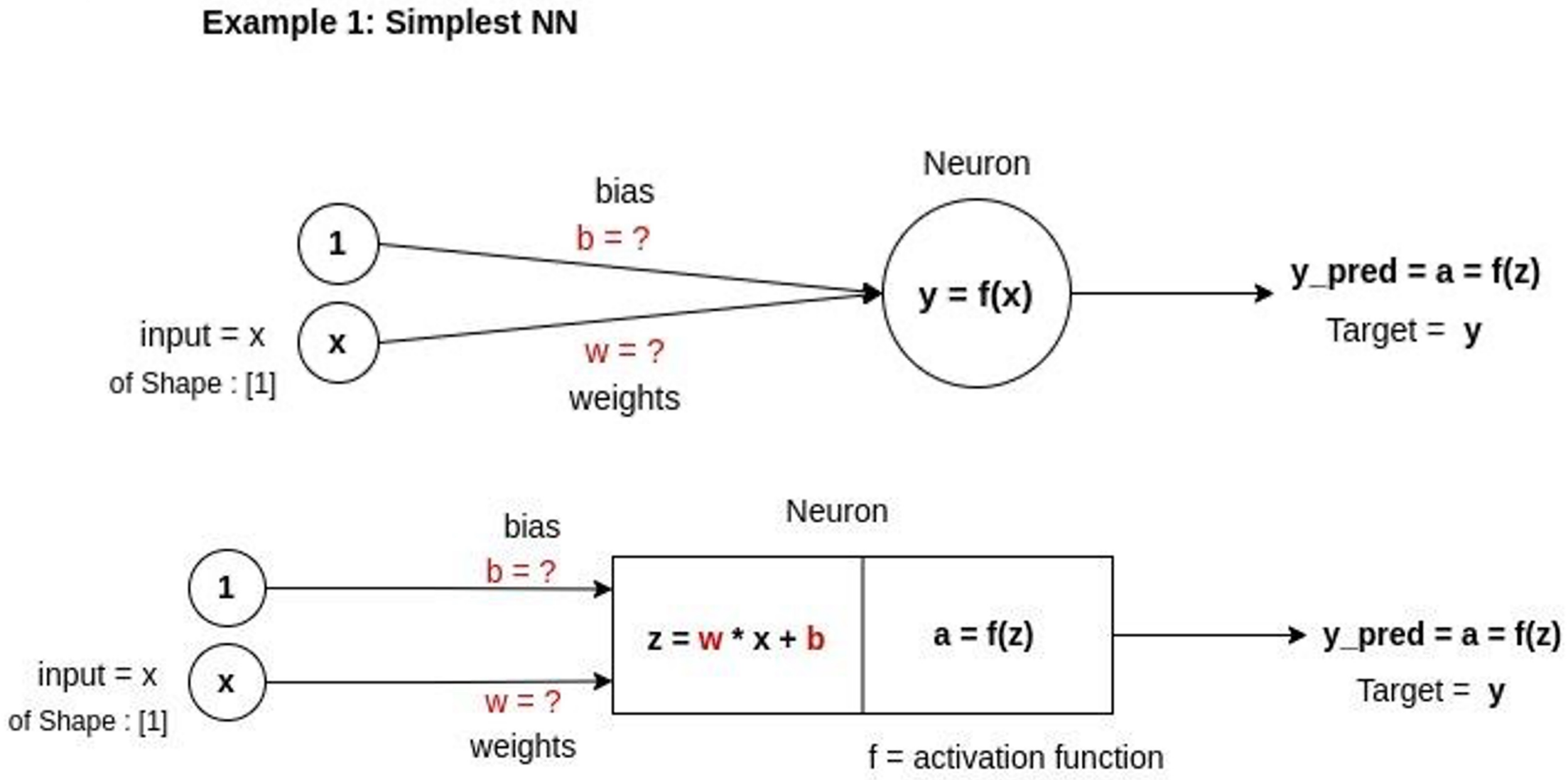
Actual Training Cycle

Backward Pass

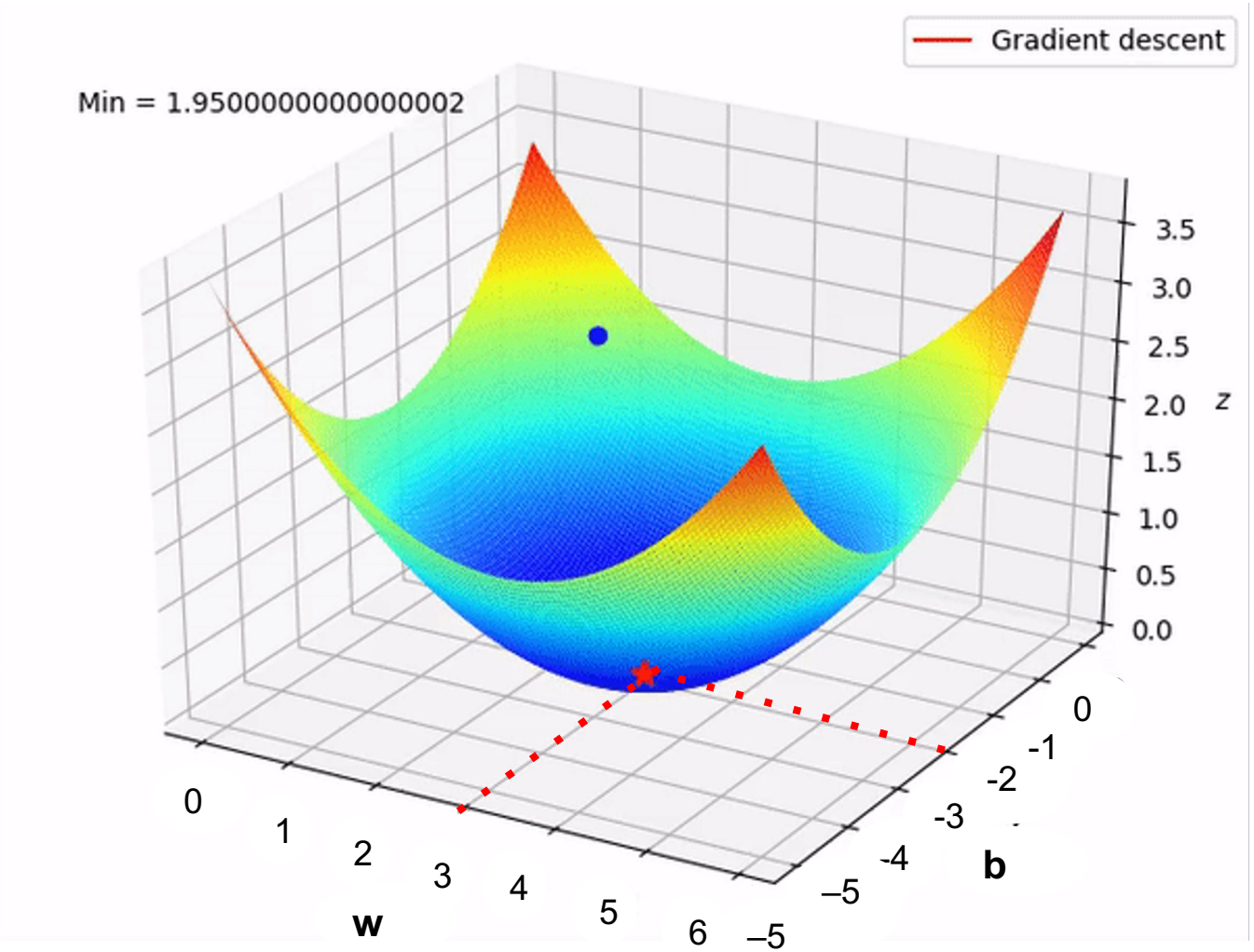


Reference: https://towardsai.net/p/machine-learning/nothing-but-numpy-understanding-creating-neural-networks-with-computational-graphs-from-scratch-6299901091b0

Example 1 - Hands on Session Revisit



Stochastic Gradient Descent



* Error can be defined as,

$$J = (y - y\_pred)^2$$

$$J(w, b) = (y - f(w * x + b))^2$$

* How to learn parameters **w** and **b**?

Repeat {

$$w : w - lr * \partial J(w, b)/\partial w$$

$$b : b - lr * \partial J(w, b)/\partial b$$

}

After Training Converged,

y ~ 3.X - 2

w ~= 3

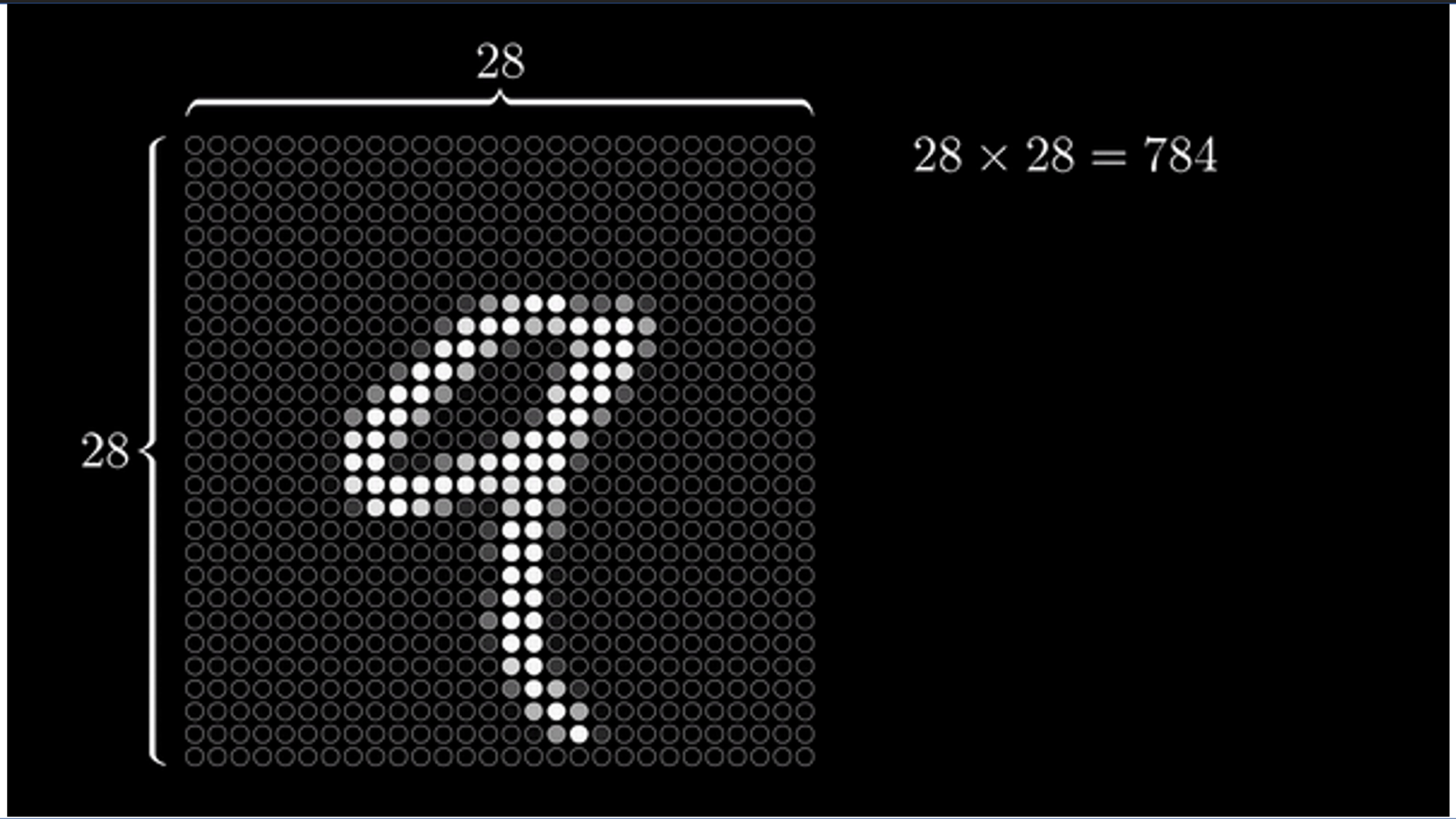
b ~= -2

```
model.compile(optimizer = keras.optimizers.SGD(lr=0.01), loss = 'mean_squared_error')
```

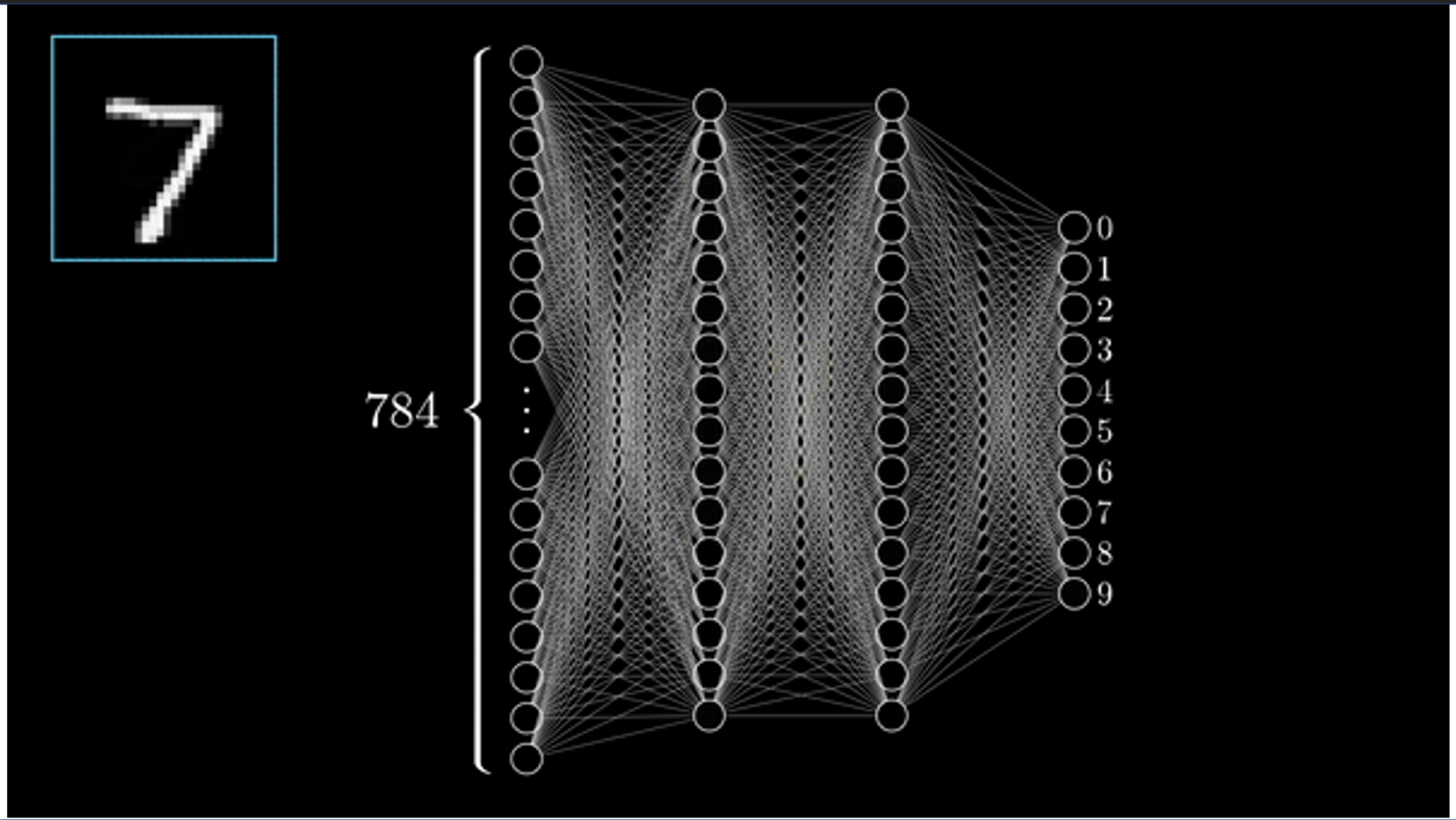
Example 2 - Hands on Session

Link to the Notebook:
https://colab.research.google.com/drive/1G-tBElfFzj7FgEWv3OUEpJq4ftFMvj4R

Pixels to Neurons



Network Propagation

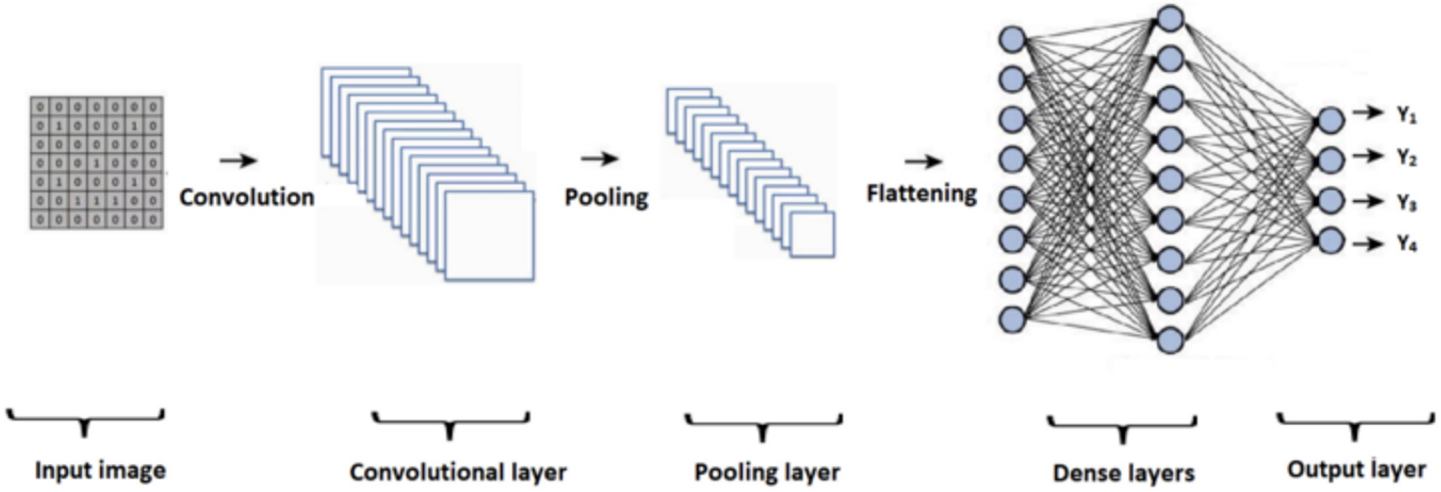


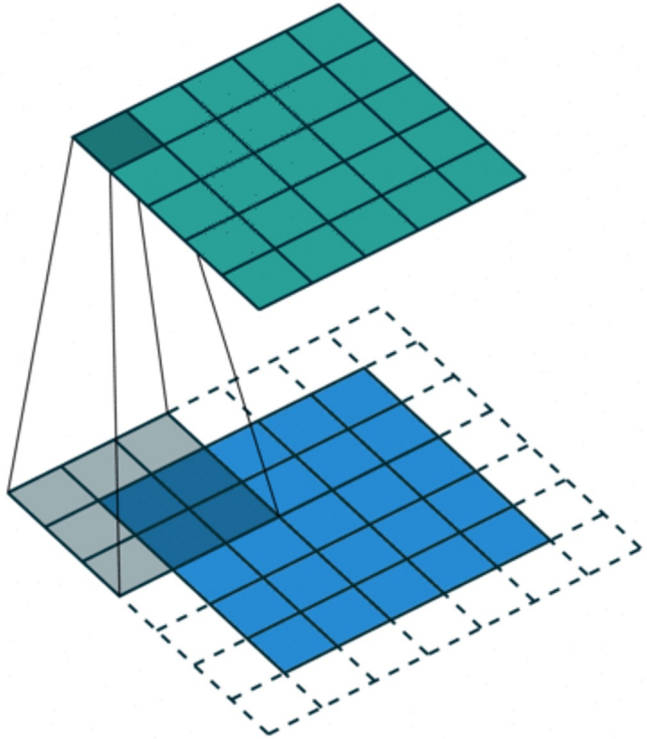
Convolutional Neural Networks

1. What if there are multiple objects in an image?
2. What if the object is not centered?
3. How to use deep learning in a complex environment?

Reference:
https://github.com/tinyMLx/colabs/blob/master/2-3-3-ExploringConvolutions.ipynb

Solution: Convolution Layers.



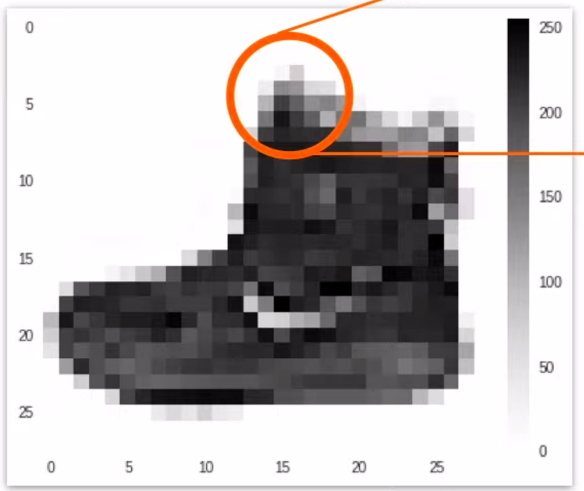


Example 3 - Hands on Session

Link to the Notebook:
https://colab.research.google.com/drive/1fSDZCXrViHVpwDgAajceAEzOBF2qVf02

Convolution Operation

https://github.com/tinyMLx/colabs/blob/master/2-3-3-ExploringConvolutions.ipynb



0	64	128
48	192	144
142	226	168

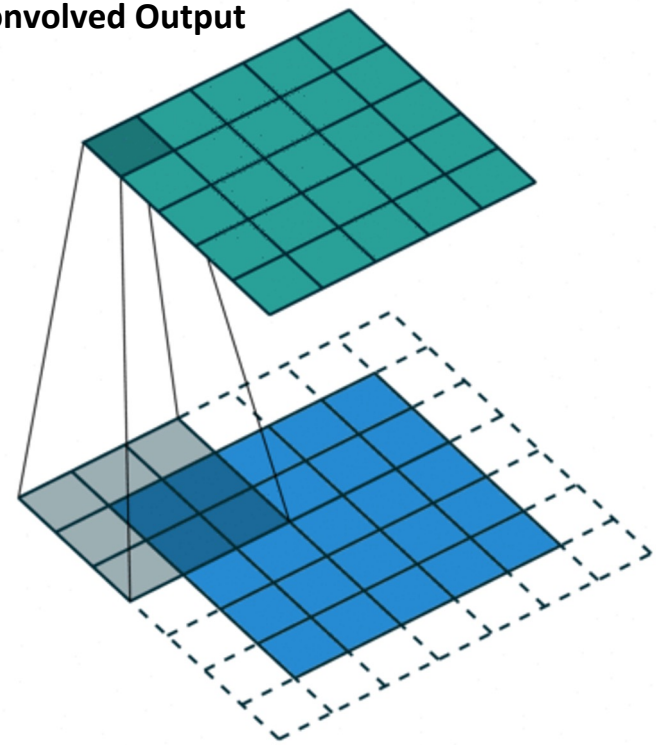
Current pixel value is 192

Consider neighbor values

-1	0	-2
.5	4.5	-1.5
1.5	2	-3

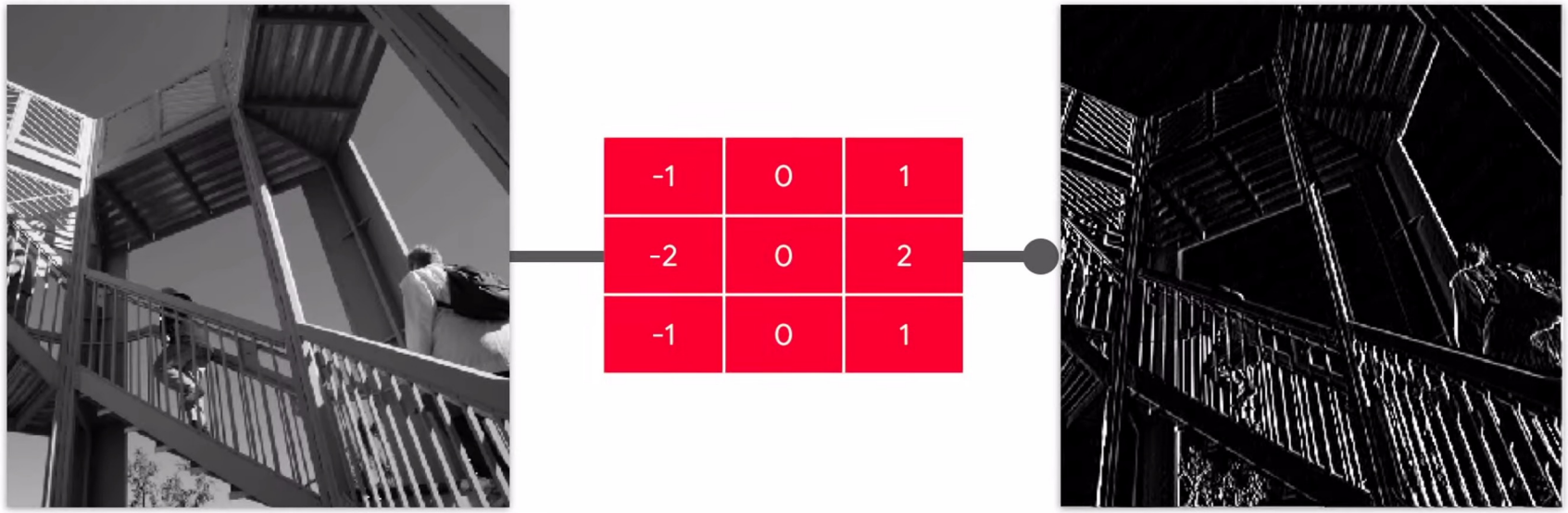
Filter definition

Convolved Output

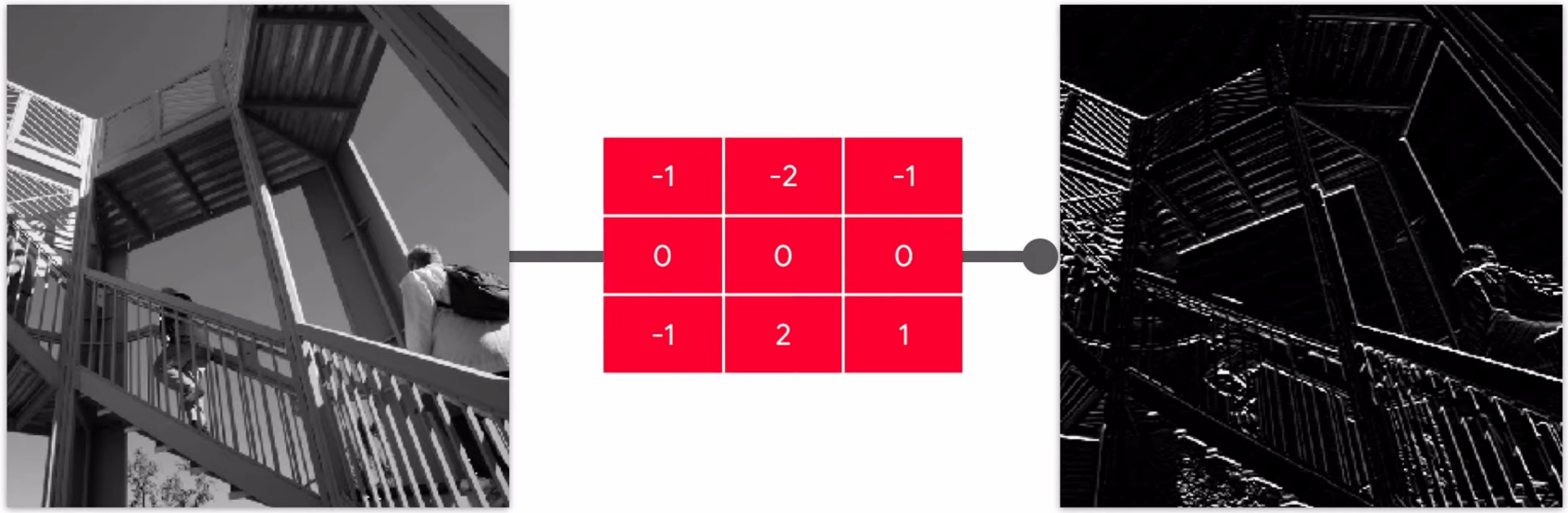


```
CURRENT_PIXEL_VALUE = 192
NEW_PIXEL_VALUE = (-1 * 0) + (0 * 64) + (-2 * 128) +
                  (.5 * 48) + (4.5 * 192) + (-1.5 * 144) +
                  (1.5 * 42) + (2 * 226) + (-3 * 168)
```

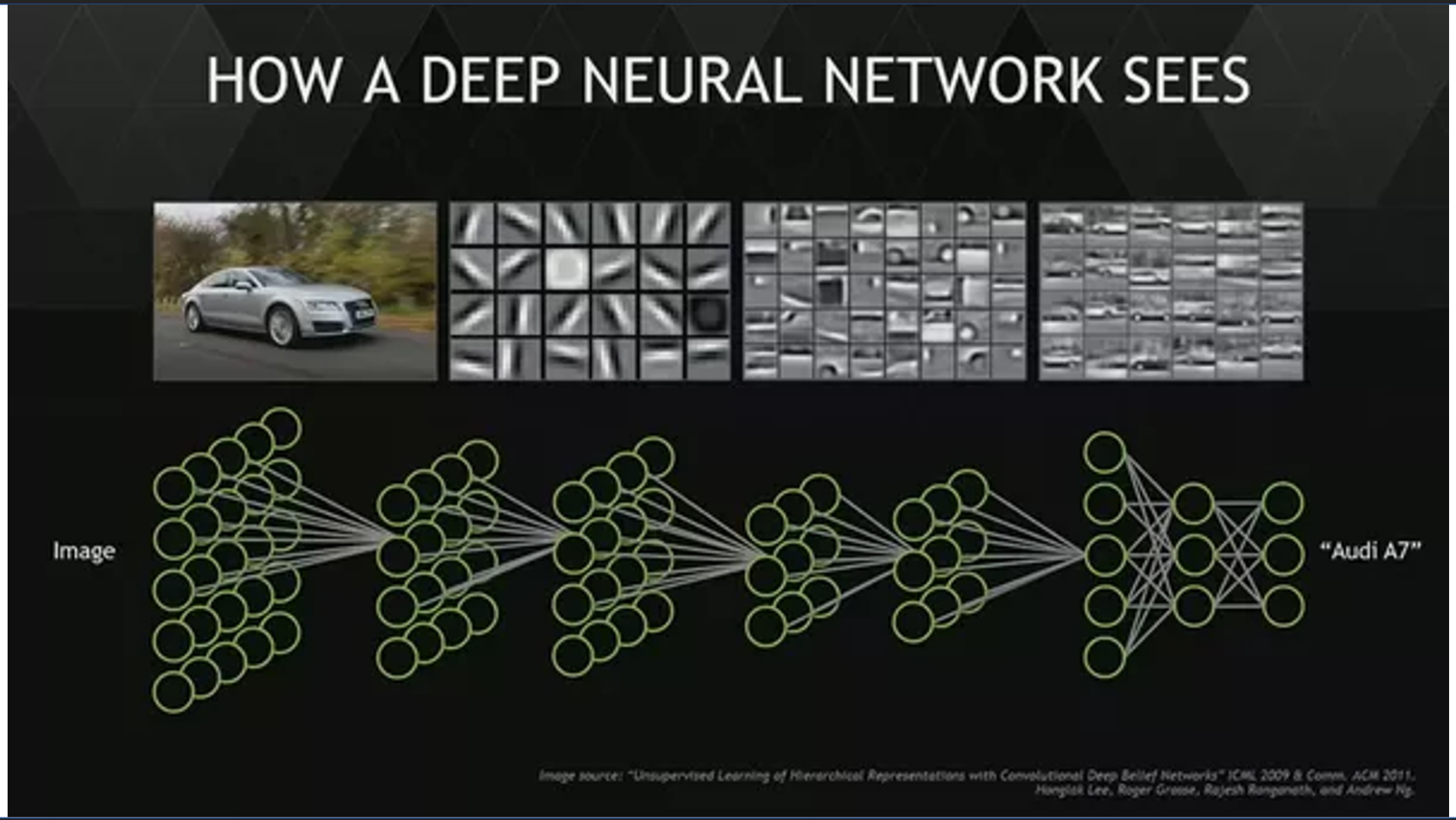
Convolution Operation



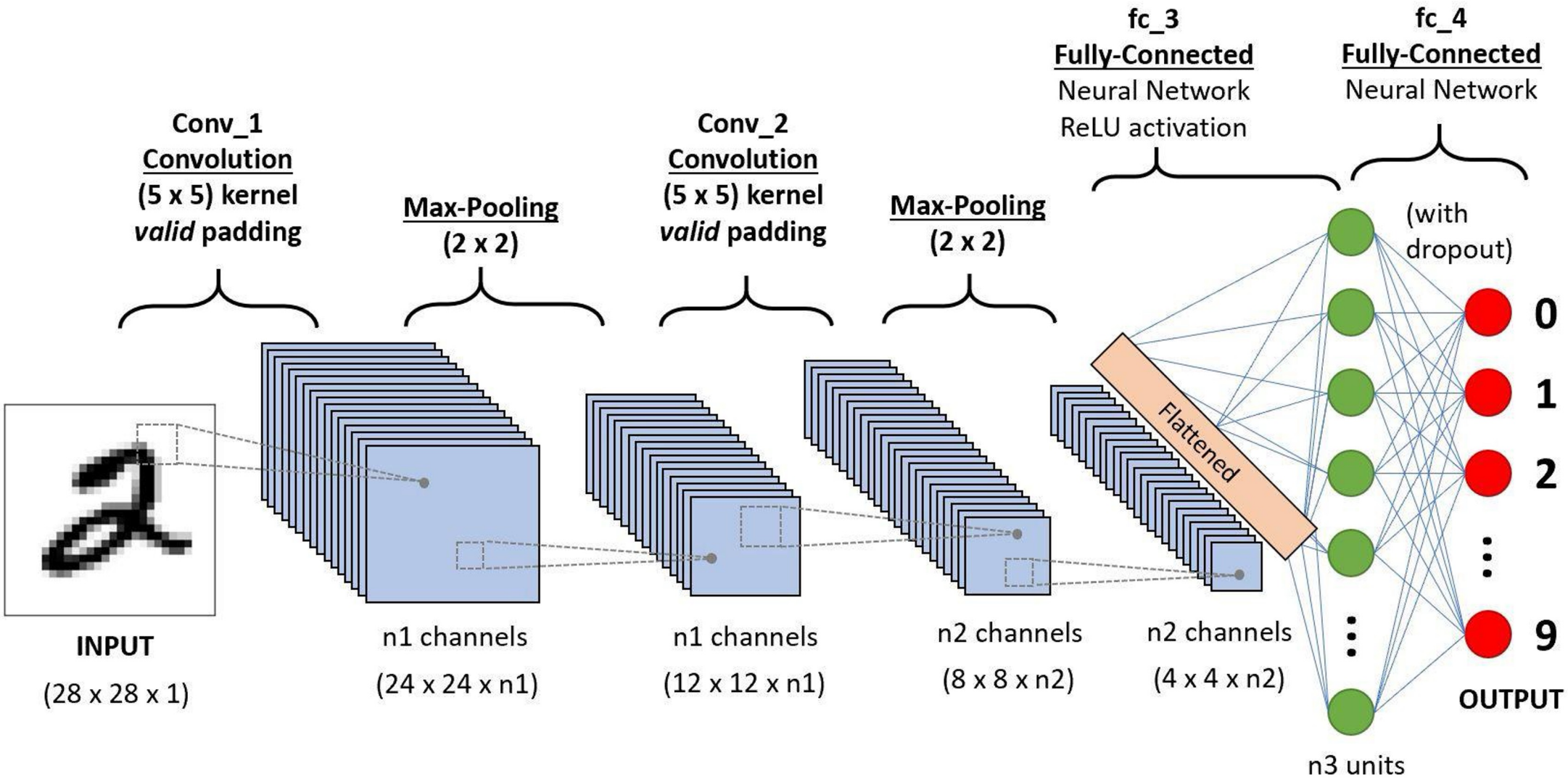
Convolution Operation



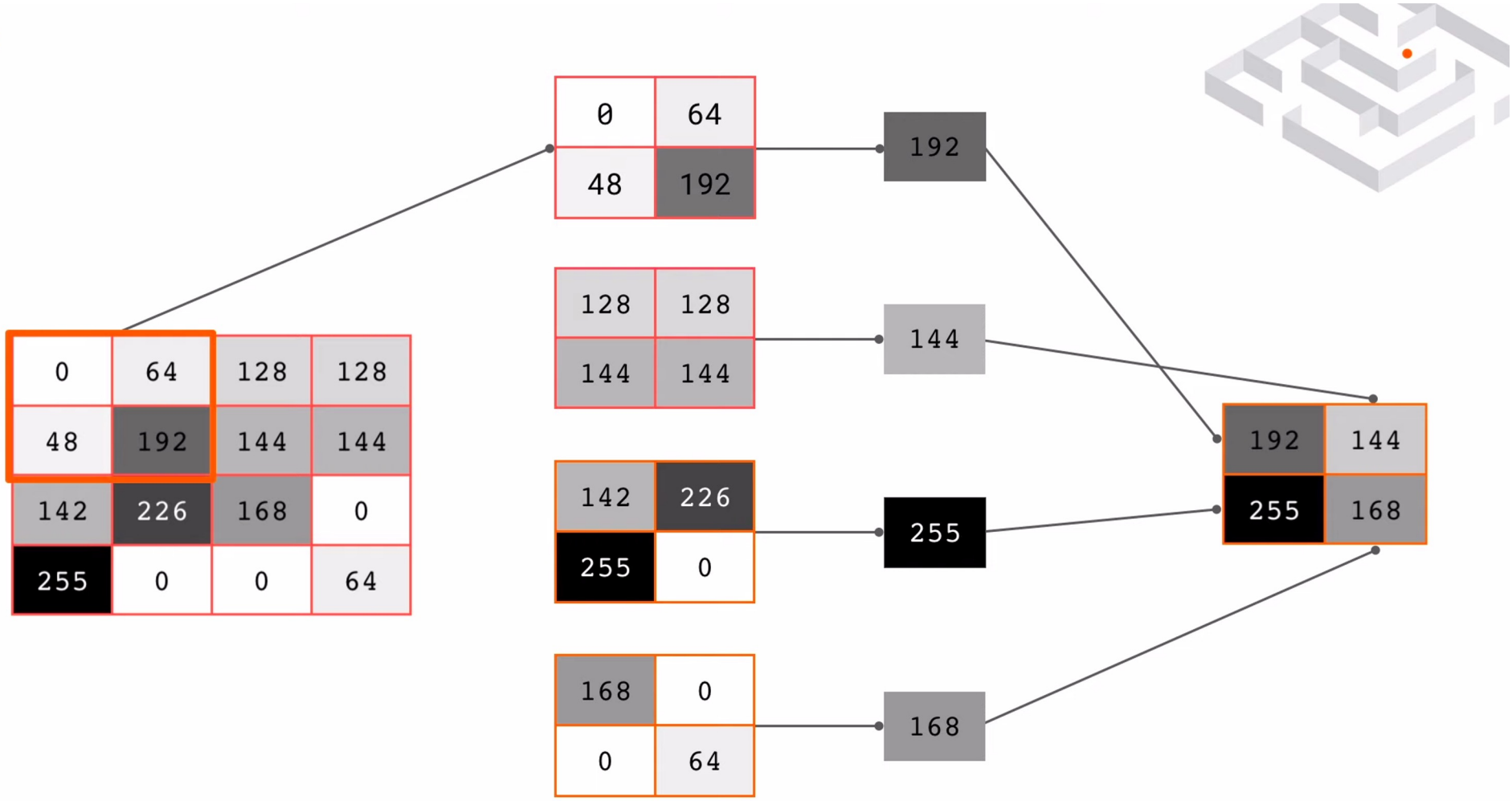
Convolution NN



Convolution NN



Max pool Operation



Max pool Operation

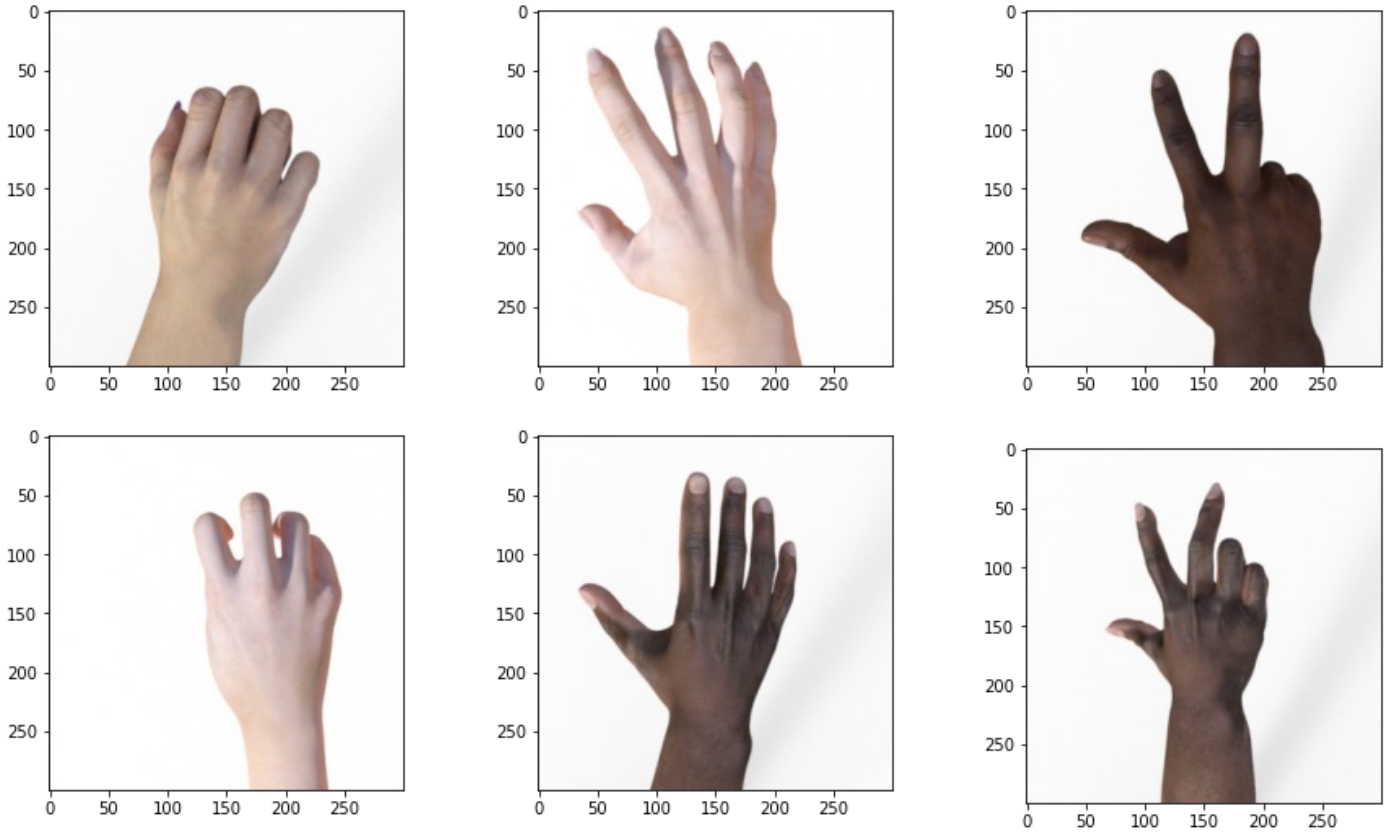
Refer: tinyMLx/colabs

https://github.com/tinyMLx/colabs

Assignment 1

Image Classification
Detect Hand Gestures
Classify Rock-Paper-Scissor game

25% of marks



Link to the Notebook:
https://colab.research.google.com/drive/10-ZvmCxksr5ujoguDyLn_suS0U5HaG91