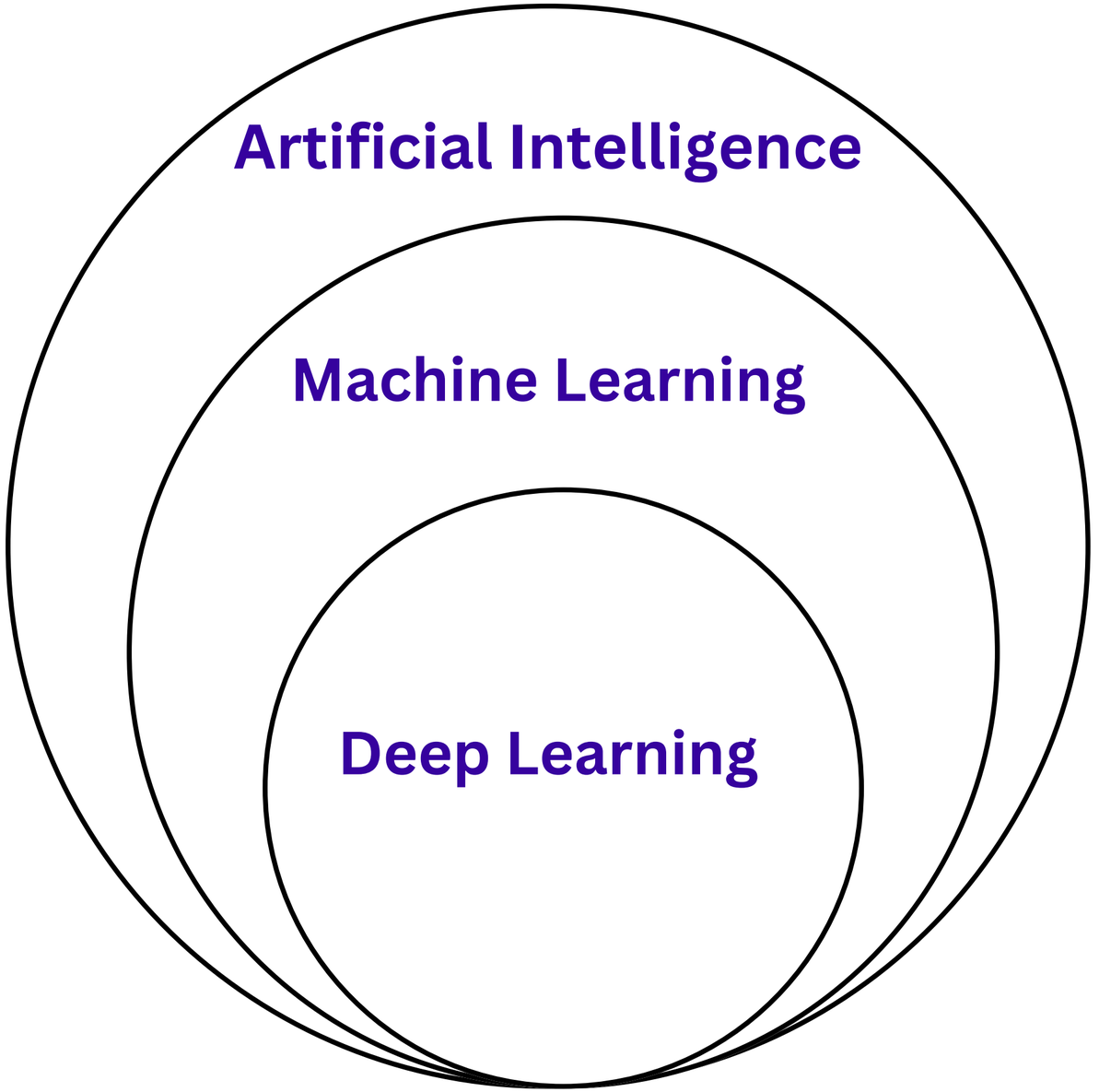
Deep Learning And Neural Networks

Bindu Paudel
Krishant Timilsina



PDSC

How machine learning works?

Image a small baby, trying to learn what apple is, how can we make sure he learns?

Step 1: Data Collection
Step 2: Error Calculation based on metric.
Step 3: Error Correction

PDSC

Cost intuition

PDSC

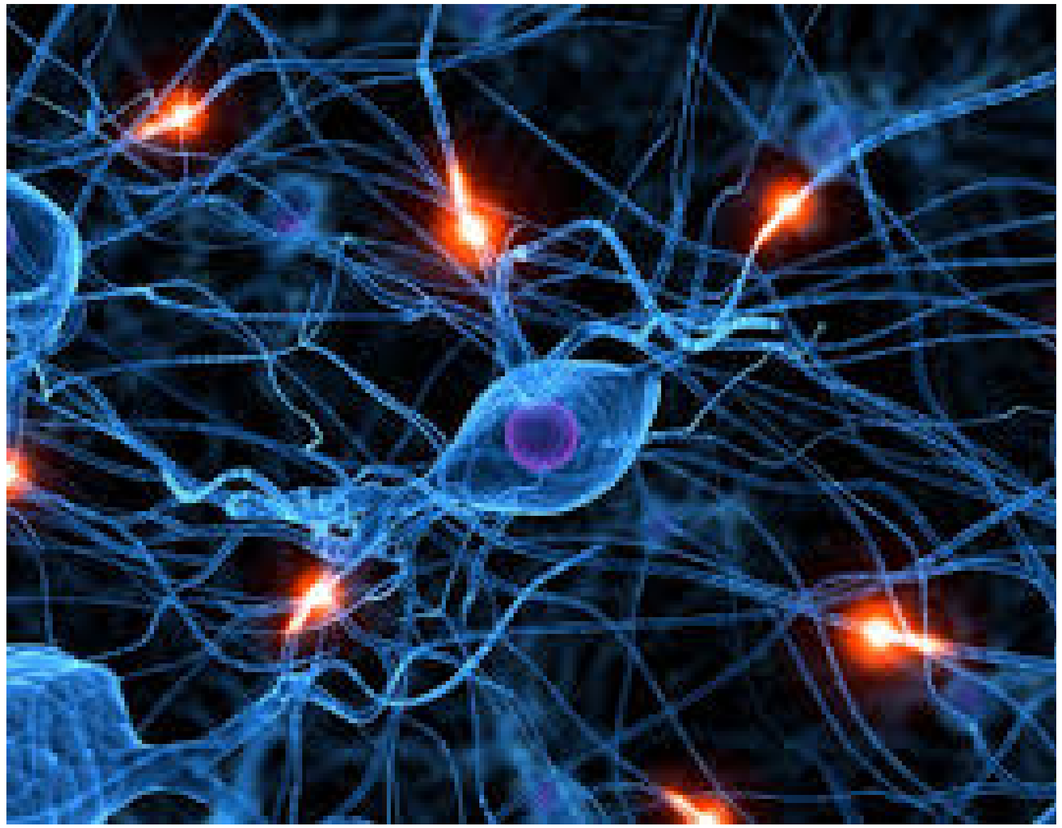
PDSC

Cost = No. of Errors/ No of total prediction

 Cost -> 👍 Model

PDSC

Neural Networks



PDSC

100 Billion Parameters

PDSC

Can we minimize this ??

PDSC

How many of you have a understanding of

- matrices?
- dot product?
- partial derivatives?
- python?
- gradient descent?

Matrices

We know matrix multiplication right, but when can't we multiply 2 matrices?

$$\begin{bmatrix} 2 & 3 \\ 1 & 4 \end{bmatrix} \begin{bmatrix} 5 & 6 \\ 7 & 8 \\ 9 & 10 \end{bmatrix} = \text{you just can't do it}$$

PDSC

Calculus

Calcute slope of parameters to know where its headed

PDSC

$$Z = 2X + 3Y$$

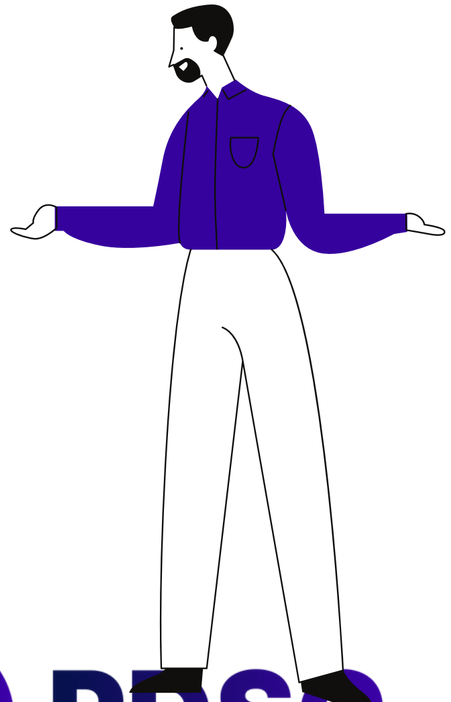
Derivative = ??

PDSC

Gradient Descent

$\partial f/\partial x$ is positive, say $\partial f/\partial x = 2$,

WHAT DOES IT MEAN ???



PDSC

Cost function has say, **1000** parameters

The gradient of C is denoted as ∇C, which is just the collection of all the partial derivatives of C with respect to each one of its 1000 variables:

$$\nabla C = \left[\frac{\partial C}{\partial w_1} \quad \frac{\partial C}{\partial w_2} \quad \cdots \quad \frac{\partial C}{\partial w_{1000}} \right]$$

PDSC

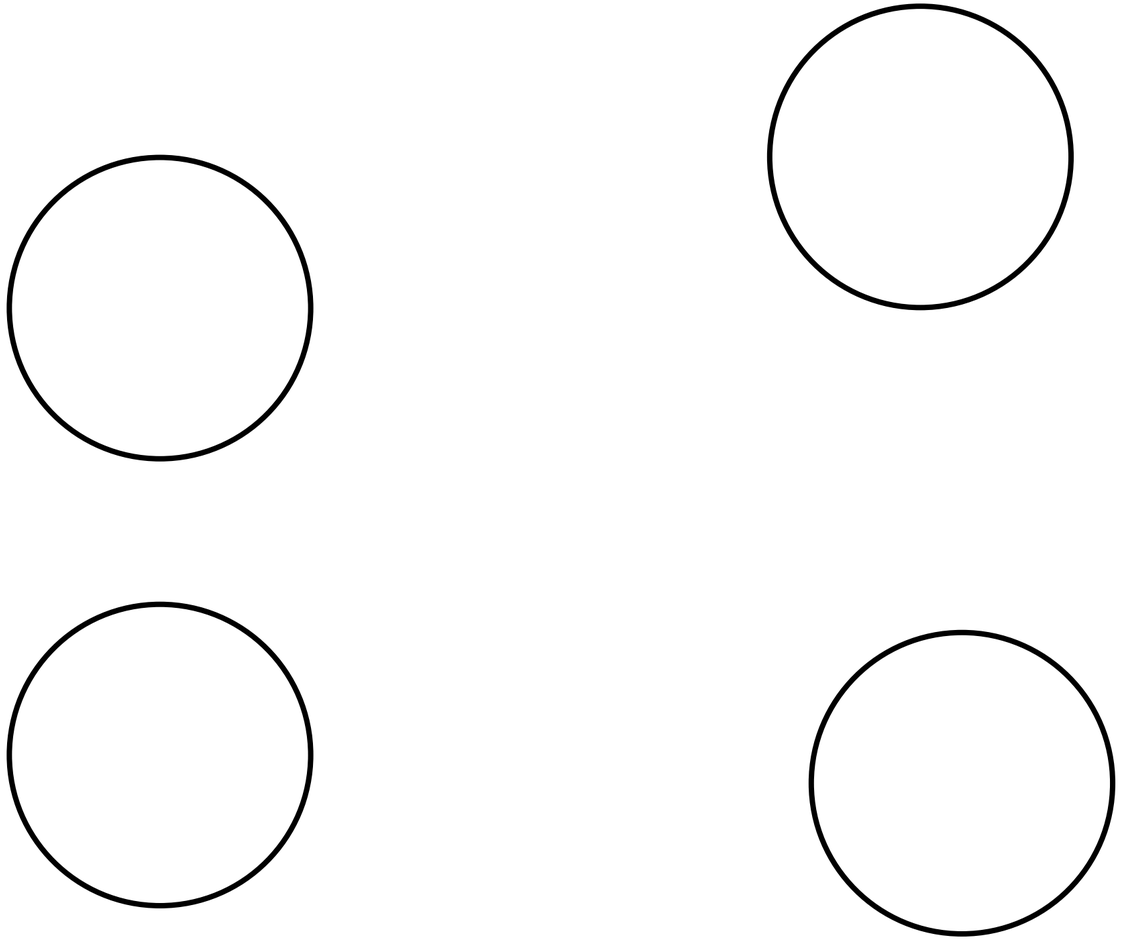
$$\nabla C = \left[\frac{\partial C}{\partial w_1} \quad \frac{\partial C}{\partial w_2} \quad \cdots \quad \frac{\partial C}{\partial w_{1000}} \right]$$

PDSC

PERCEPTON MODEL

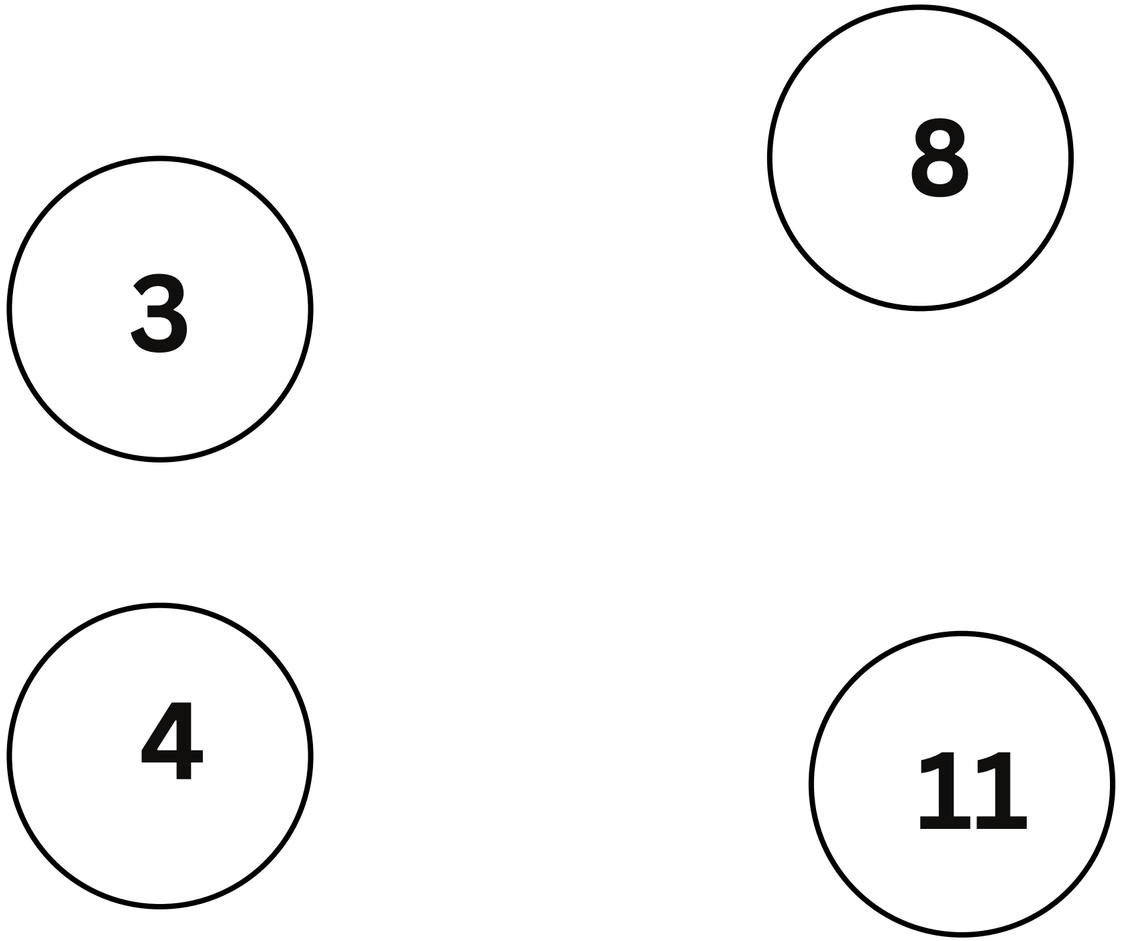
PDSC

NODES



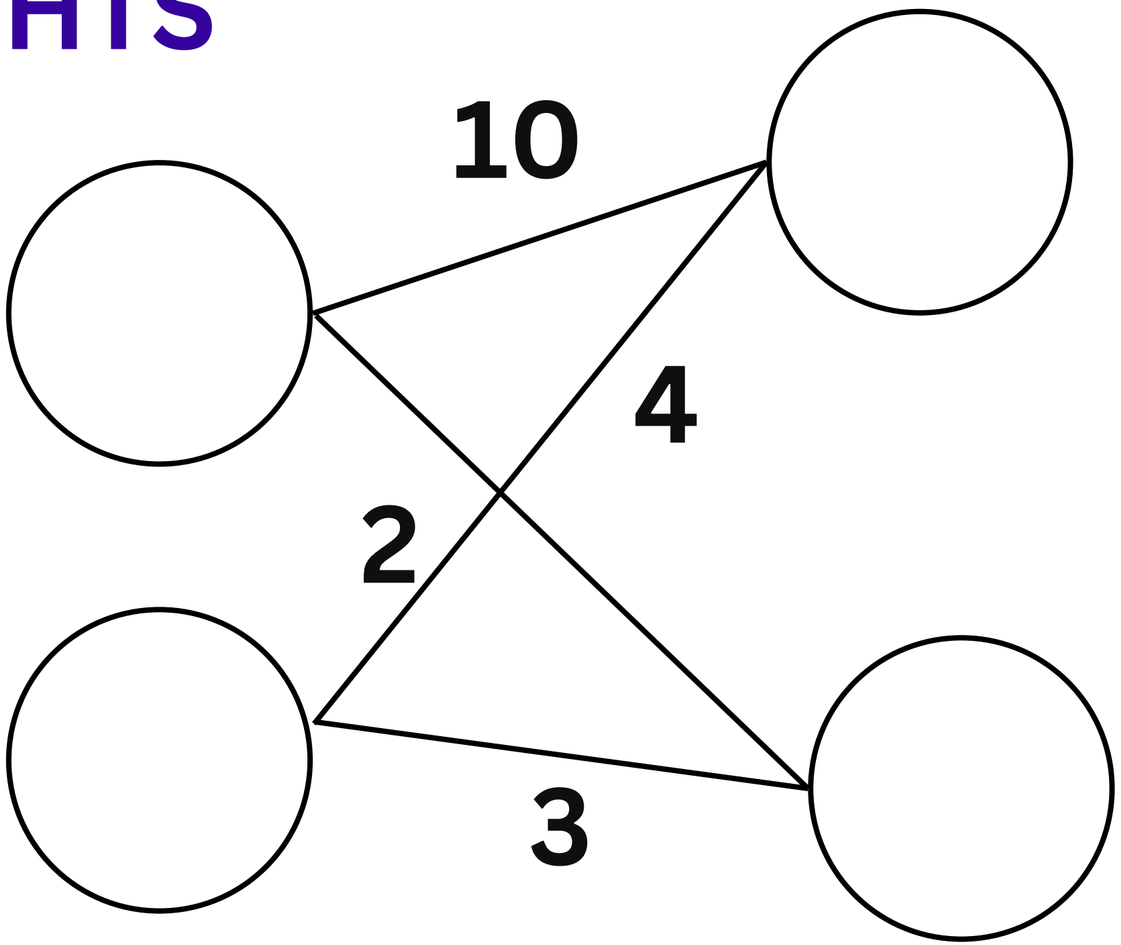
PDSC

NODES

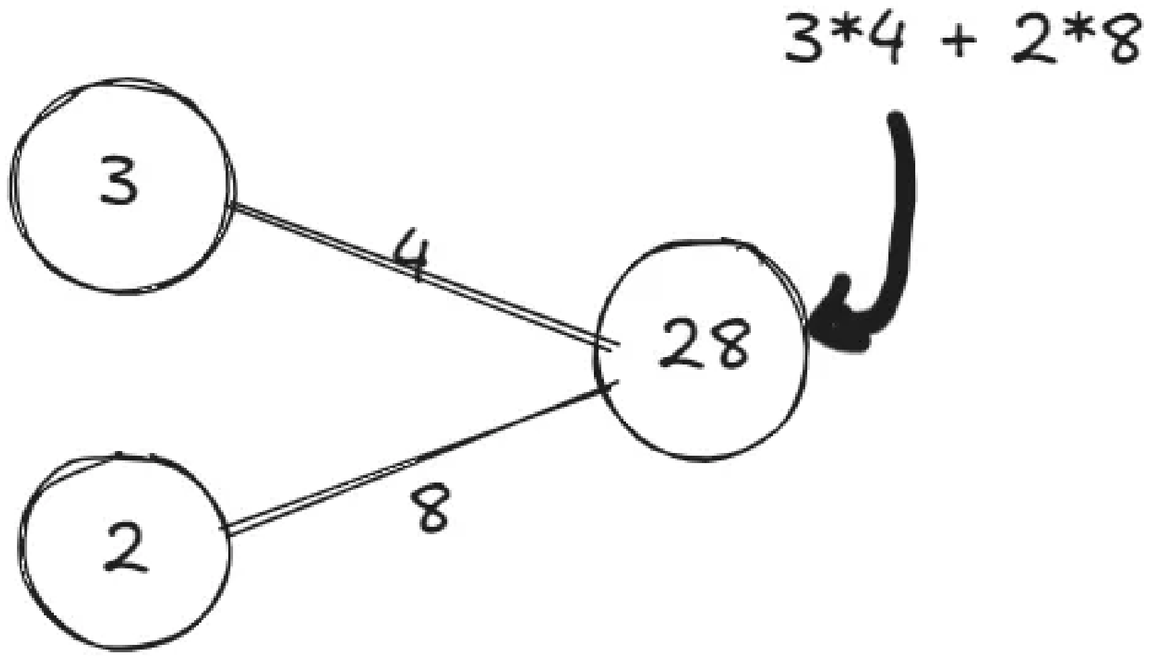


PDSC

WEIGHTS



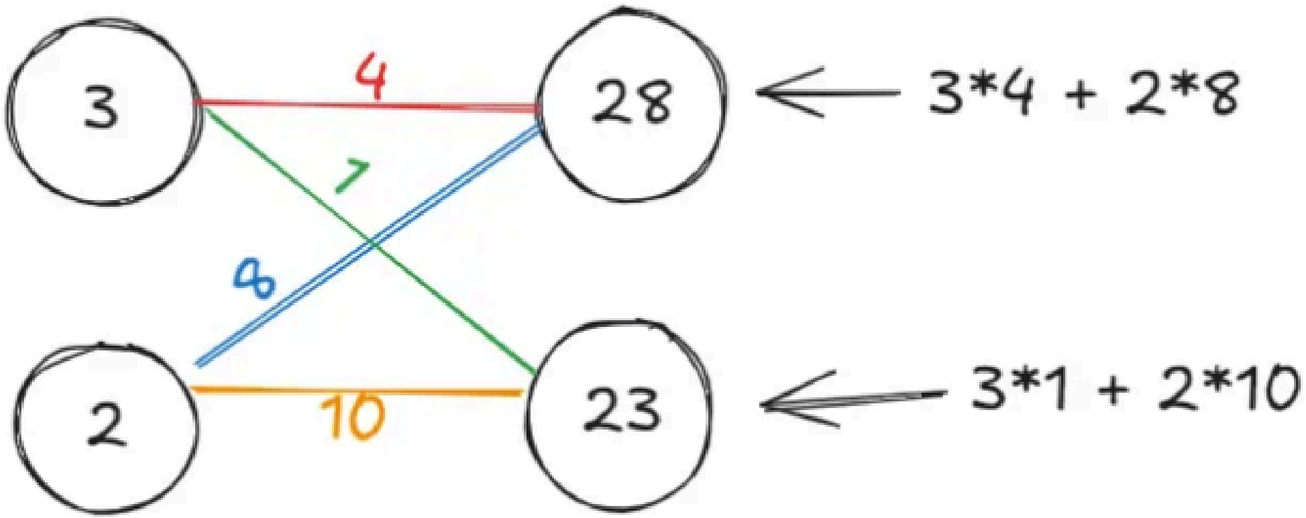
PDSC



$$[3, 2] * [4, 8] = 28$$

PDSC

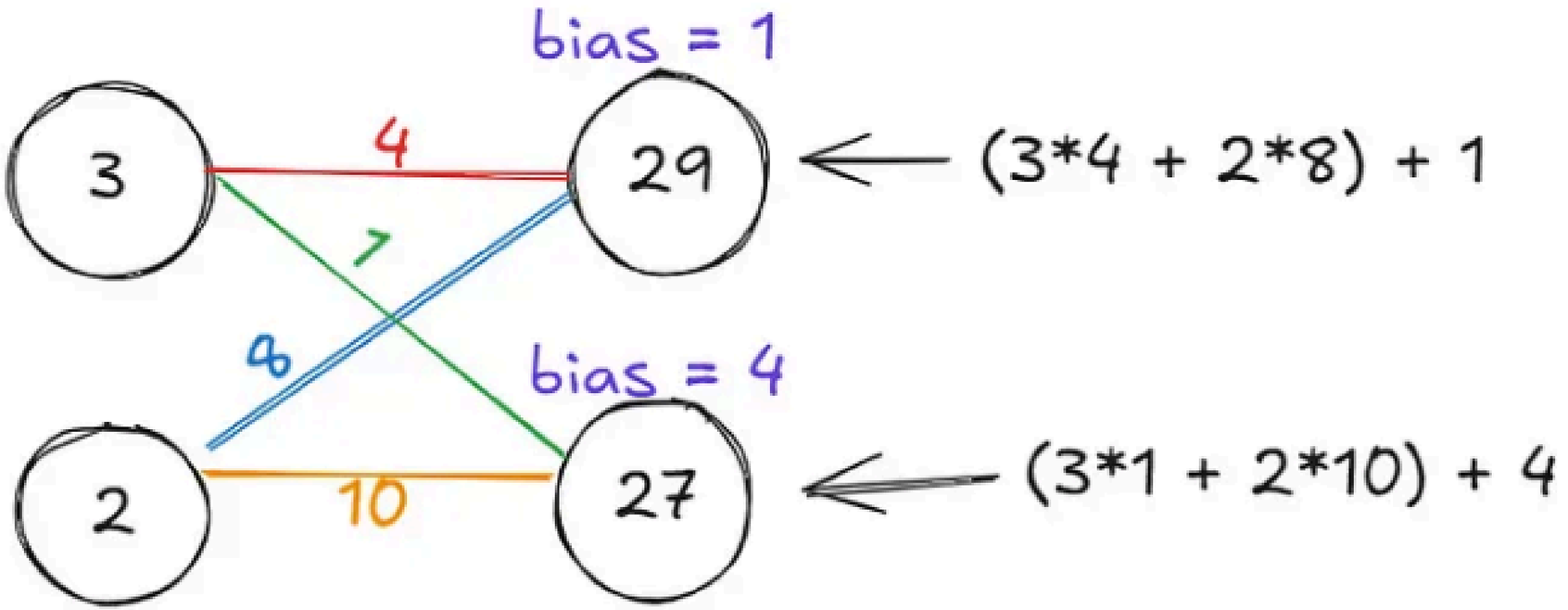
Let’s move on to a bigger example:



A 2 layer neural network

PDSC

Biases



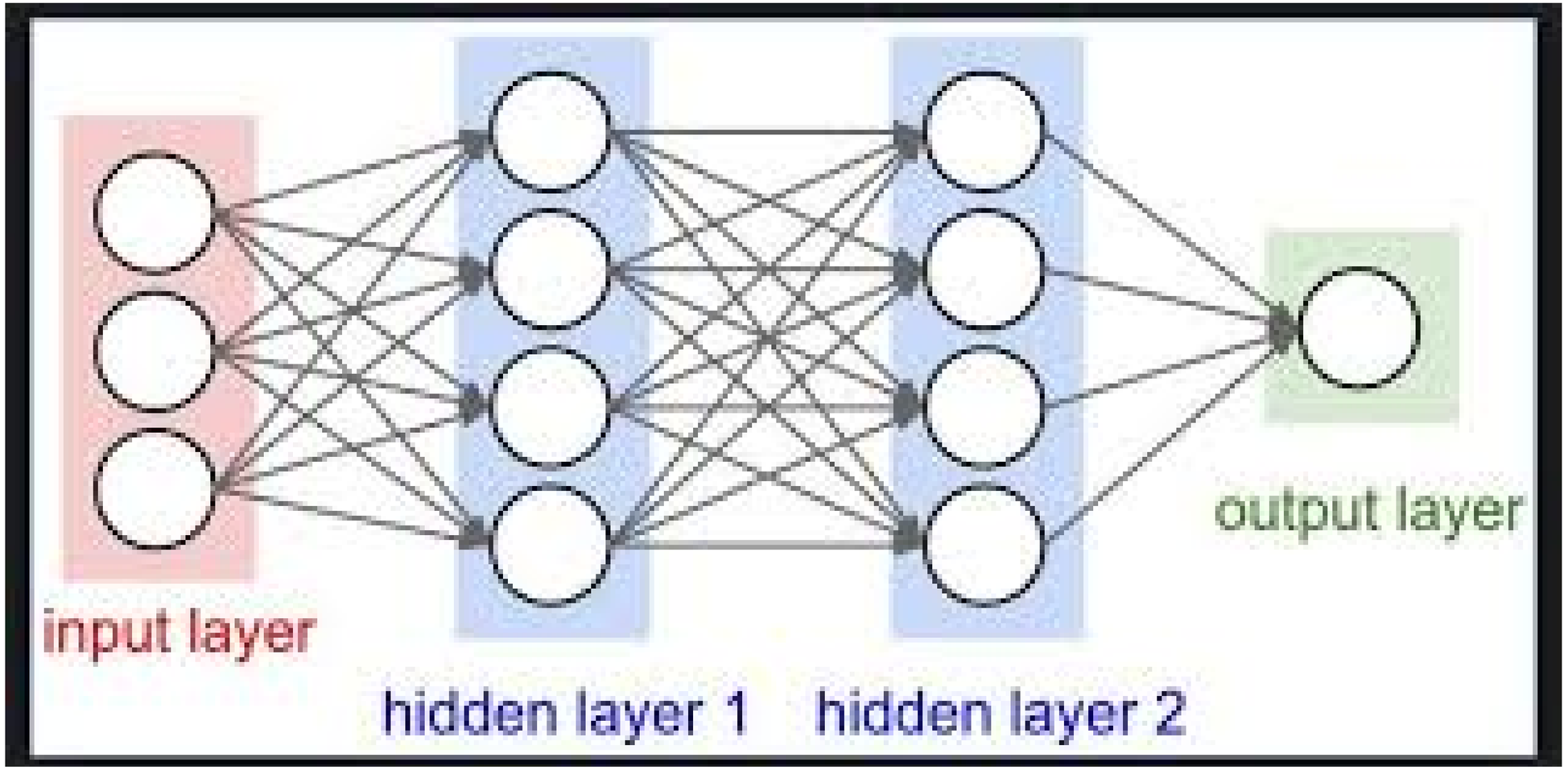
Example of a 2 layer network with biases

PDSC

LAYERS OF NETWORK

- Input Layer
- Hidden Layer
- Output Layer

PDSC

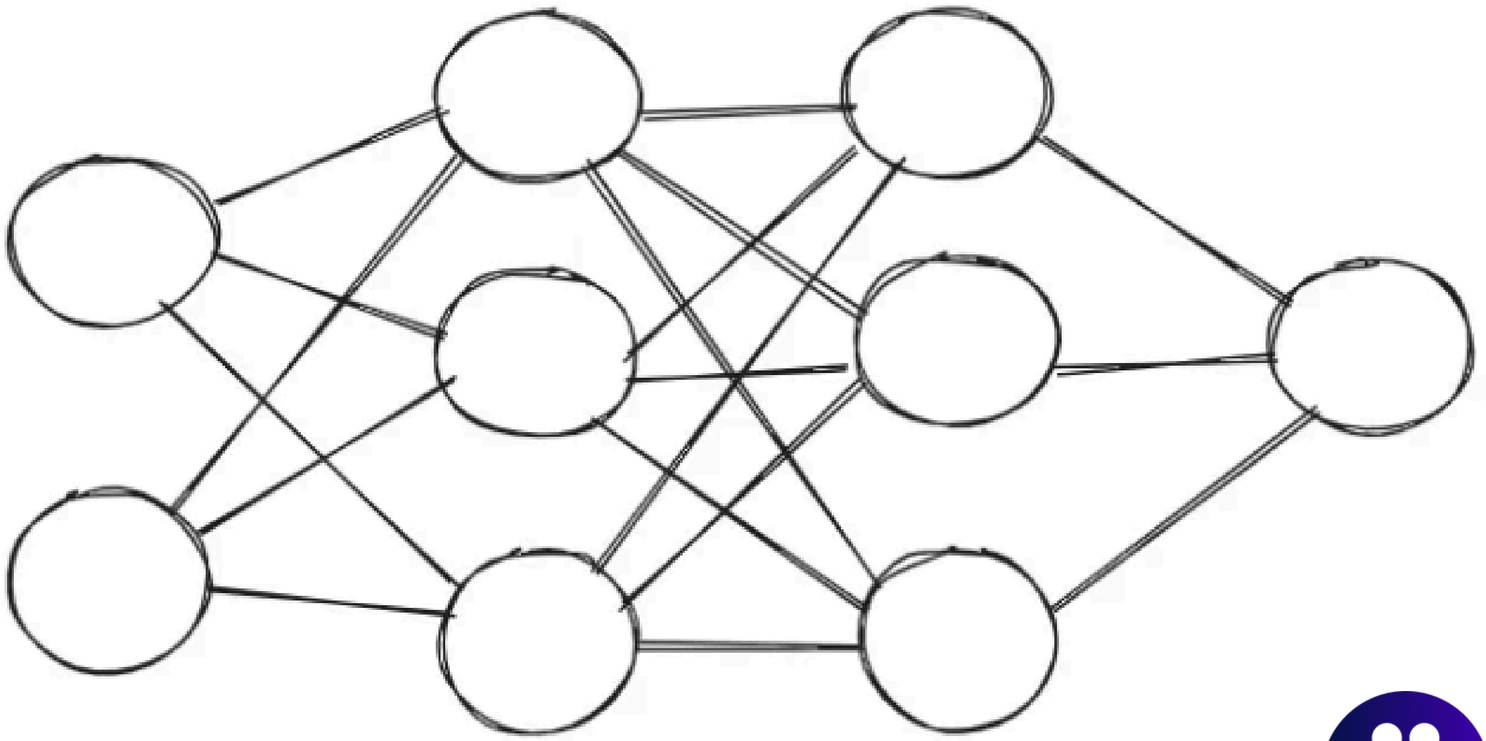


PDSC

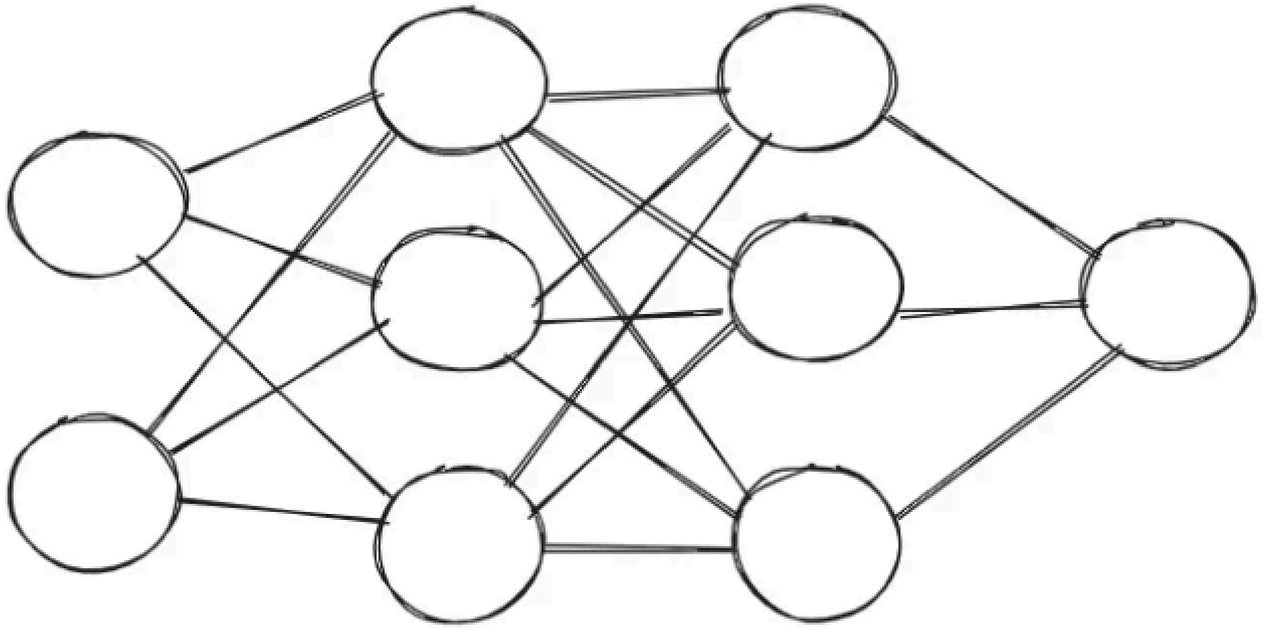
Neural Network Notation

PDSC

No of nodes in each layer



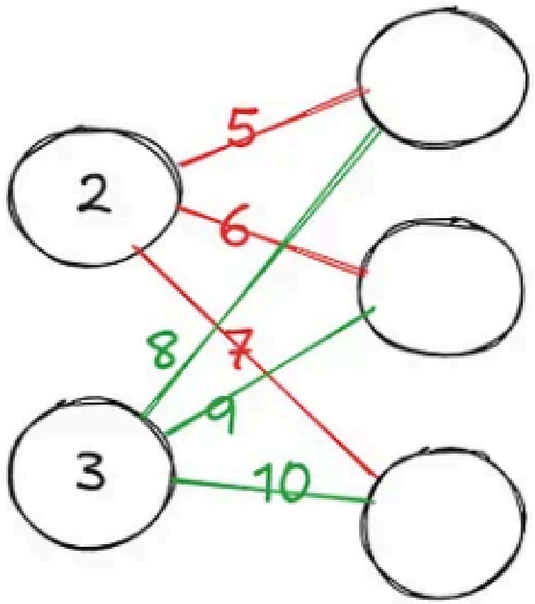
PDSC



$$n^{[0]} = 2, n^{[1]} = 3, n^{[2]} = 3, n^{[3]} = 1$$

PDSC

How to calculate weights ??



node 1 = [2, 5] dot [3, 8]

node 2 = [2, 6] dot [3, 9]

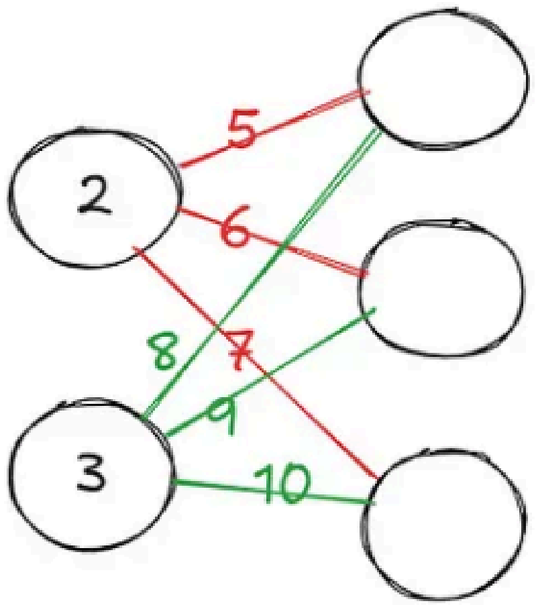
node 3 = [2, 7] dot [3, 10]

PDSC

1. Gather all the red weights and place them in a vector: [5, 6, 7]

2. Gather all the green weights and place them in a vector: [8, 9, 10]

3. input layer as [2,3]



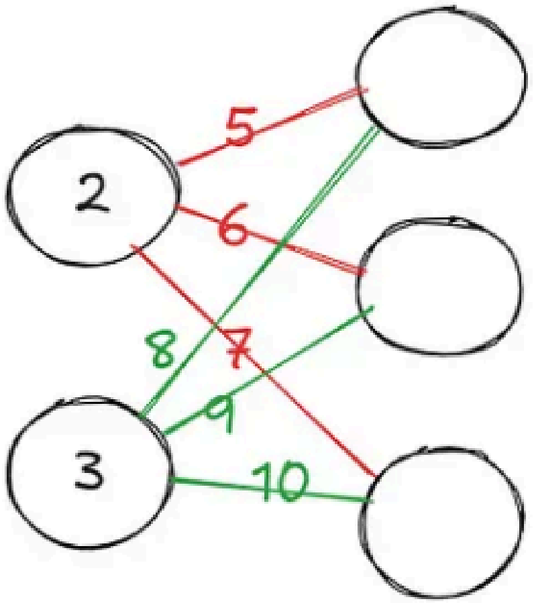
PDSC

$$\begin{bmatrix} 2 & 3 \end{bmatrix} \begin{bmatrix} 5 & 6 & 7 \\ 8 & 9 & 10 \end{bmatrix} = \begin{bmatrix} 2*5+3*8 \\ 2*6+3*9 \\ 2*7+3*10 \end{bmatrix}$$

With matching dimensions, we get what we want.

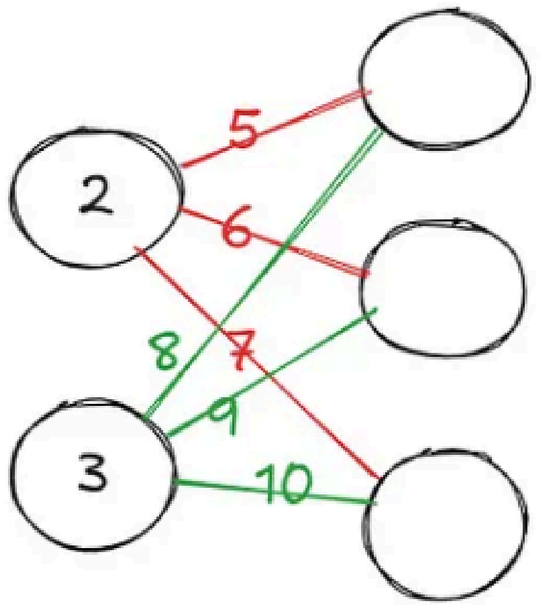
PDSC

- The 0th layer has 2 nodes, and since the 1st layer has 3 nodes, each node in the 0th layer has 3 weight connections.
- We treat the set of weight connections belong to a node in the 0th layer as a vector of size 3, since there are 3 connections.
- There are two nodes in the 0th layer, so we have 2 sets of weights.
- To form the weight matrix, we stack the vectors side by side vertically, so each vector is its own column. This gives us a 3 x 2 matrix.

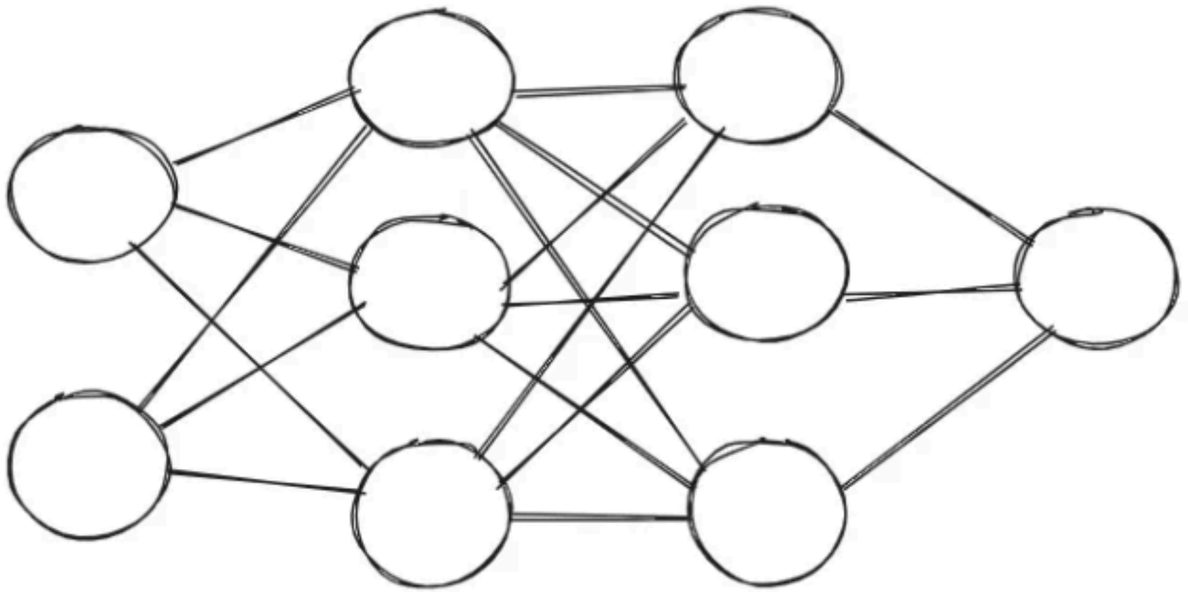


PDSC

W^[l] has dimensions n^[l] x n^[l-1] if n^[l] is the number of nodes in layer l and n^[l-1] is the number of numbers in the previous layer l-1.



PDSC

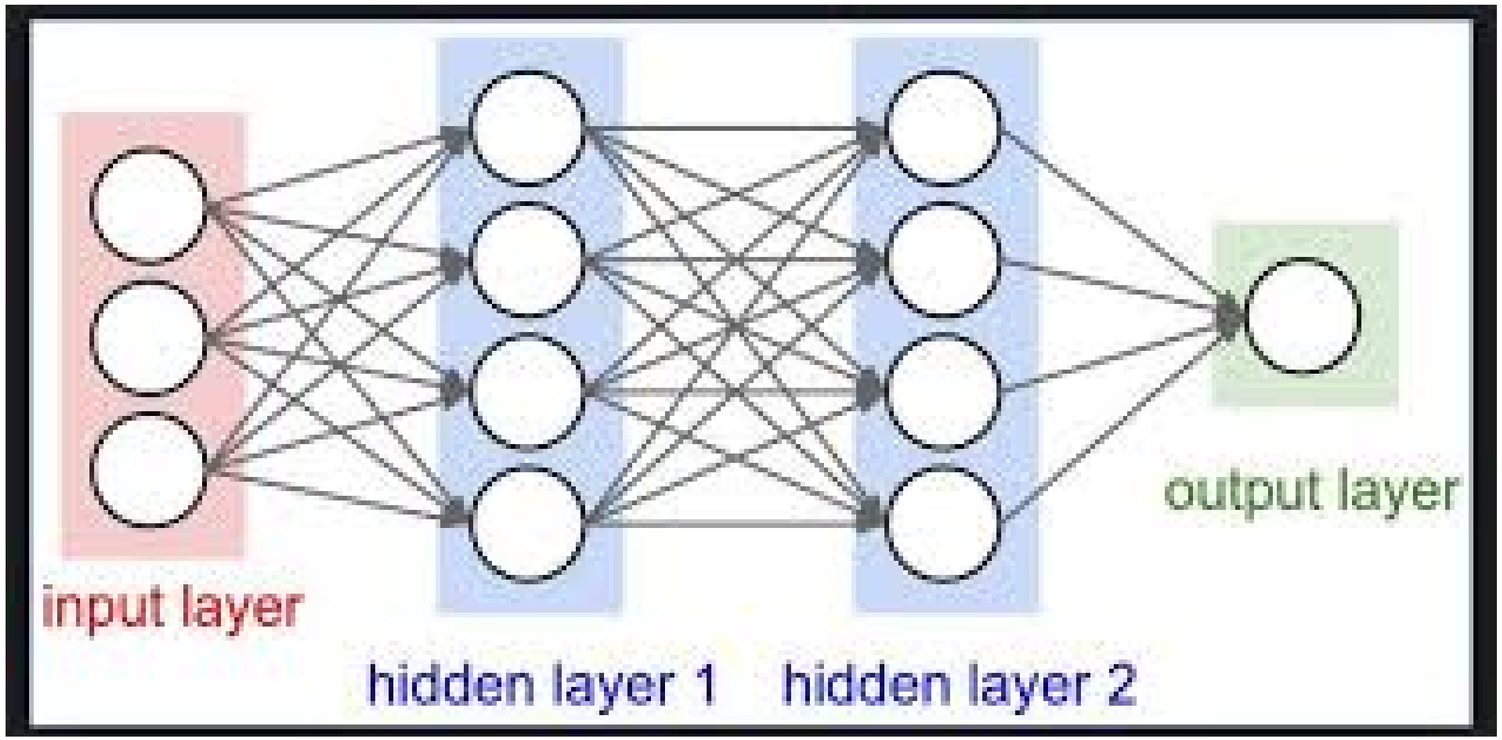


n^[0] = 2, n^[1] = 3, n^[2] = 3, n^[3] = 1

- `W^[1]` : 3 x 2 matrix
- `W^[2]` : 3 x 3 matrix
- `W^[3]` : 1 x 3 matrix

PDSC

Forward Propagation



PDSC

Forward Propagation

$$y_j = b_j + \sum_i x_i w_{ij}$$

PDSC

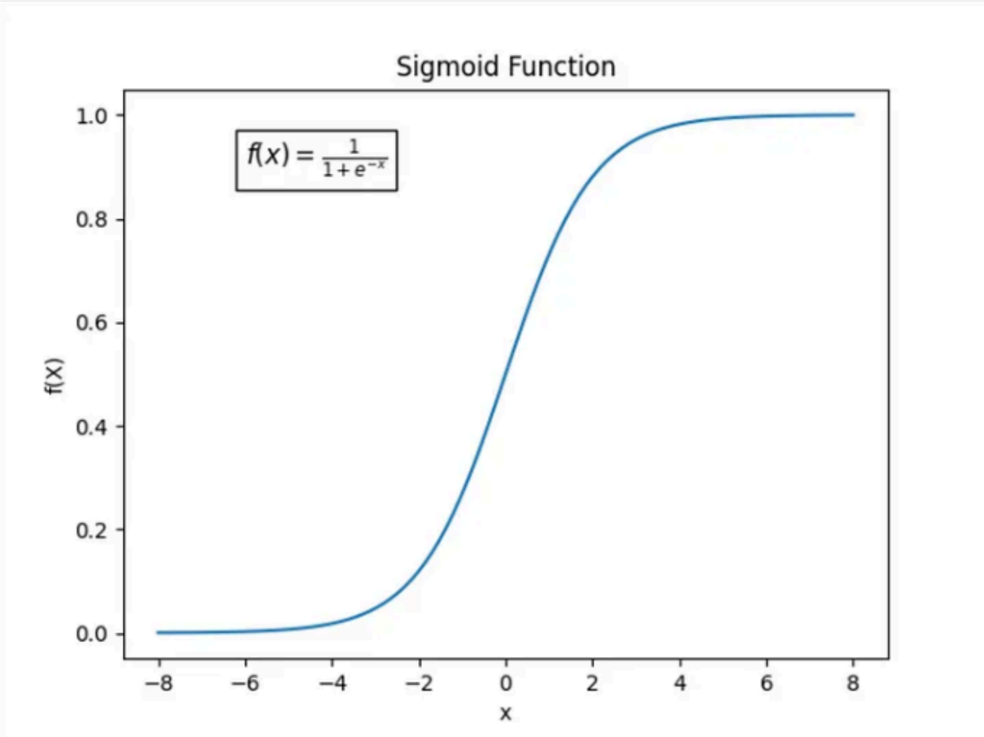
Activation Function

- Must be nonlinear, as in it CANNOT be in the form g(z) = mz + b.
- Must compress the input to a predetermined range, like 0 to 1.

PDSC

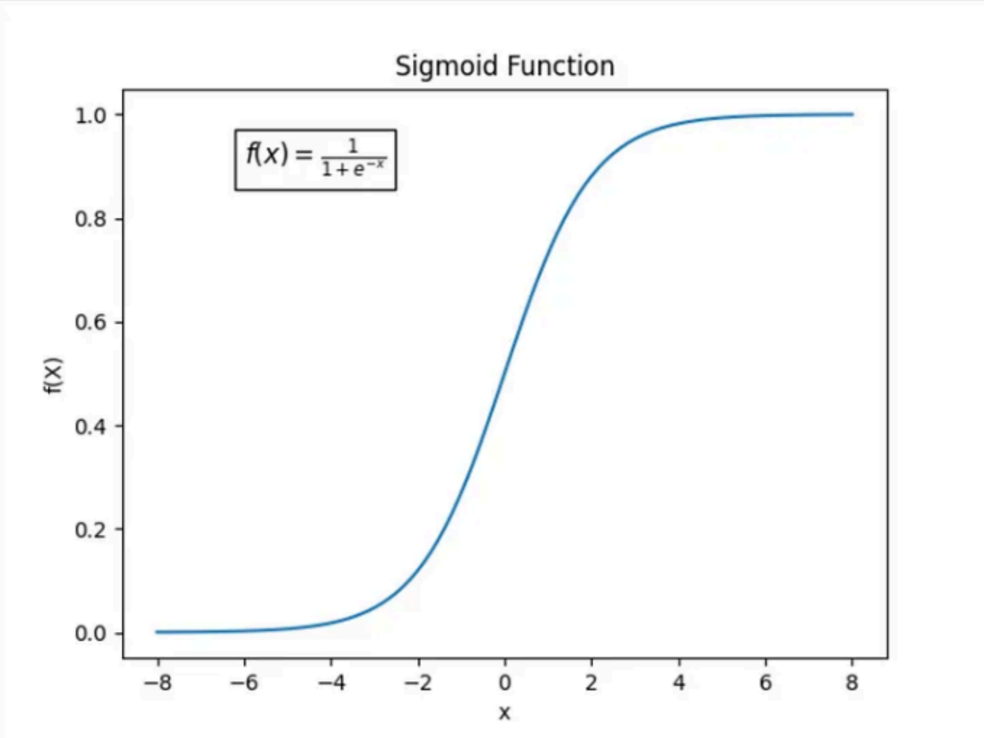
Activation Function

The activation function we will be using is the sigmoid function, which looks like so:



What the sigmoid function looks like

The sigmoid function ranges between 0 and 1. As you go to the right, the sigmoid function output flatlines at 1, and as you go to the left, the sigmoid function output flatlines at 0.

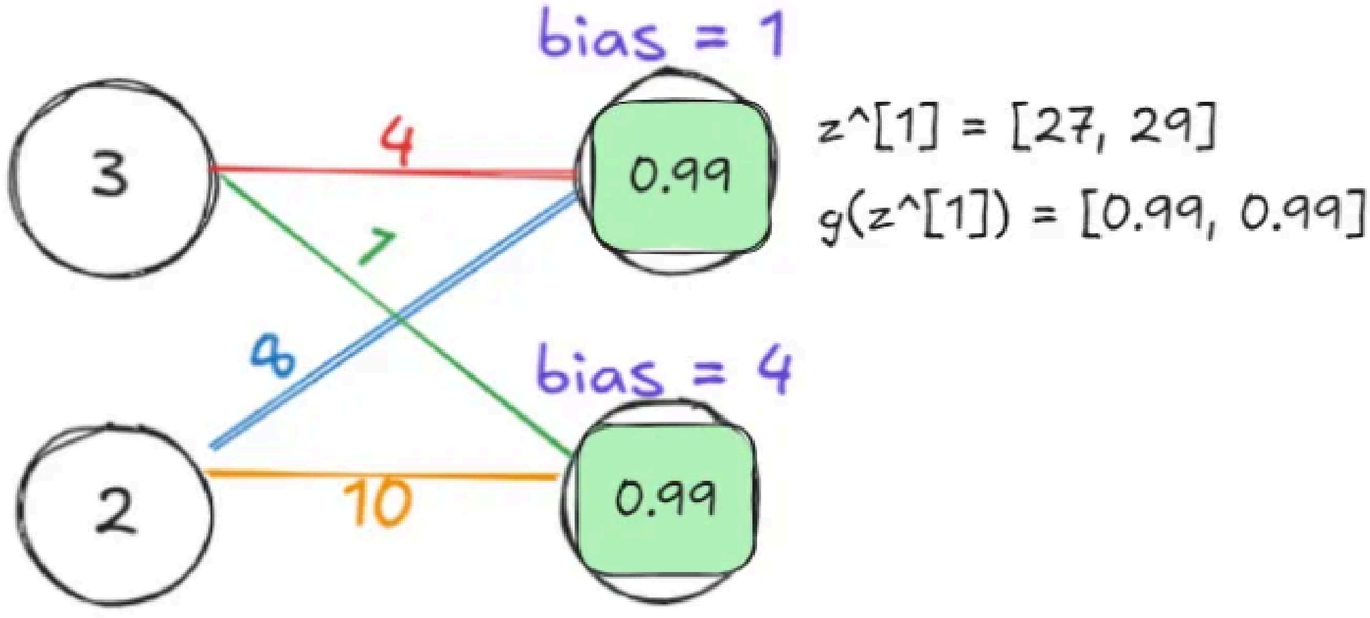


What the sigmoid function looks like

The z^[1] calculation is just an intermediary step, where the first node in layer 1 has value 27, and the second node in layer 1 has value 29.

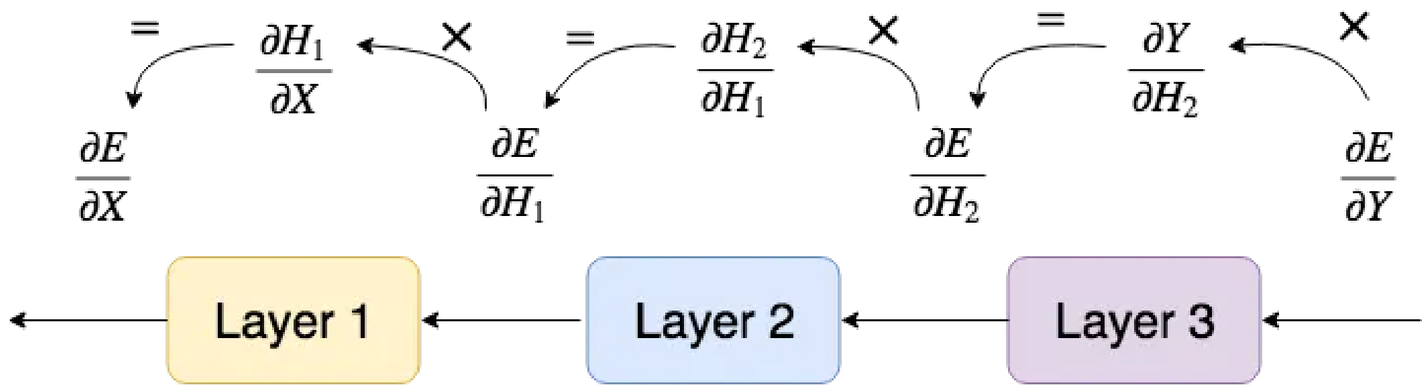
Now we pass those z values into our sigmoid activation function g(z) = 1/(1 + e^(-z)) to get back actual values for nodes in a layer, a^[l].

When we pass in 27 into g(z), we get g(27) = 0.99. For 29 into g(z), we get g(29) = 0.99.



Backward Propagation

$$w \leftarrow w - \alpha \frac{\partial E}{\partial w}$$



PDSC

Coding Time

PDSC