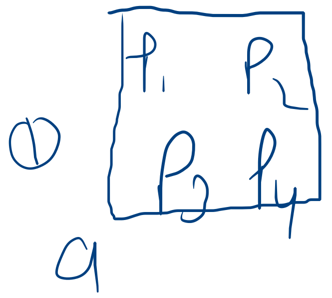
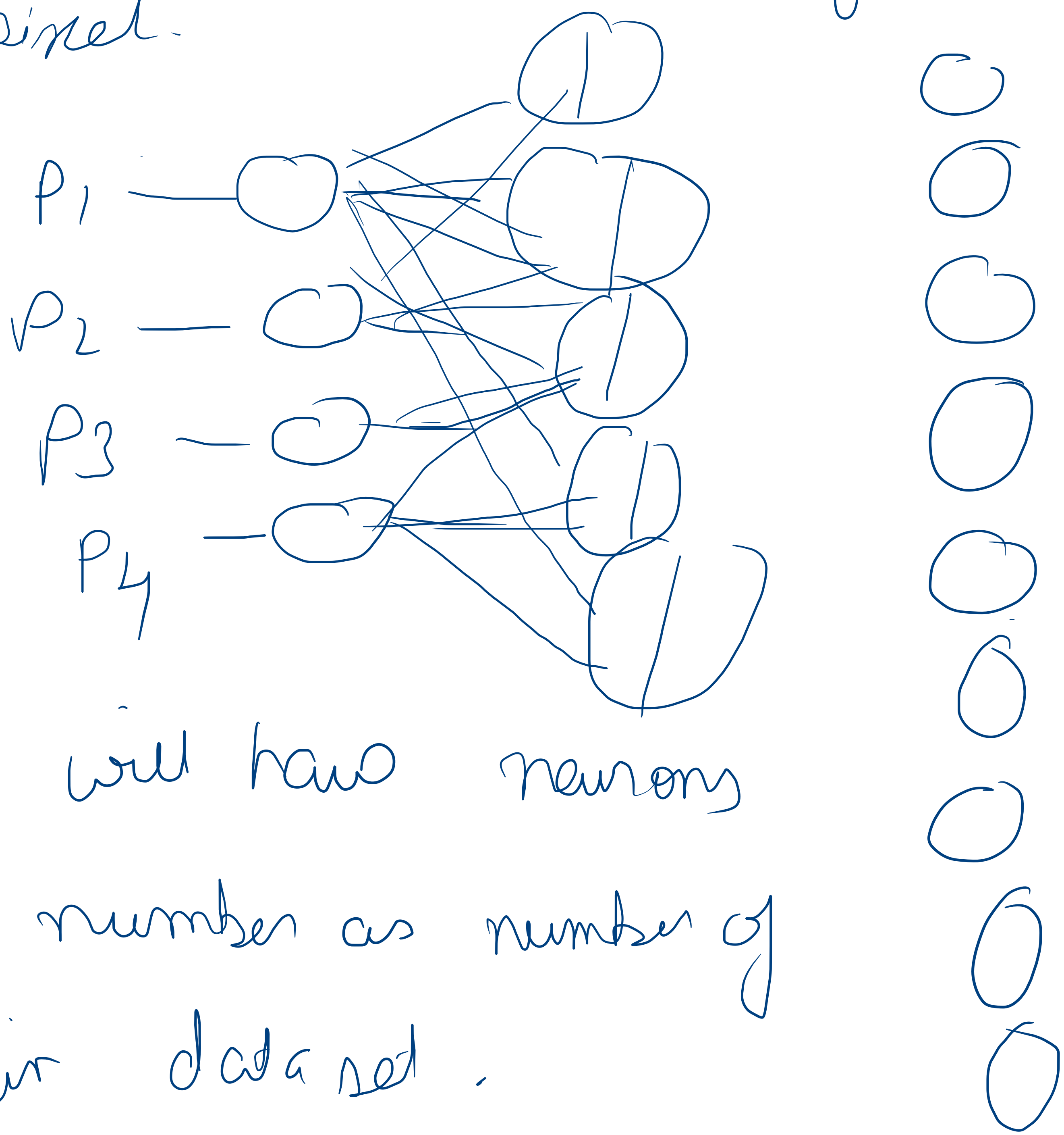
$2 \times 2 = 4$ pixel.

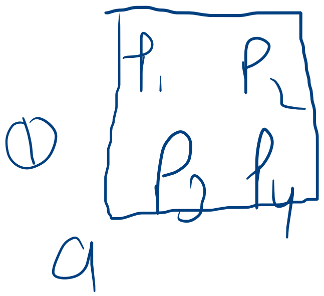


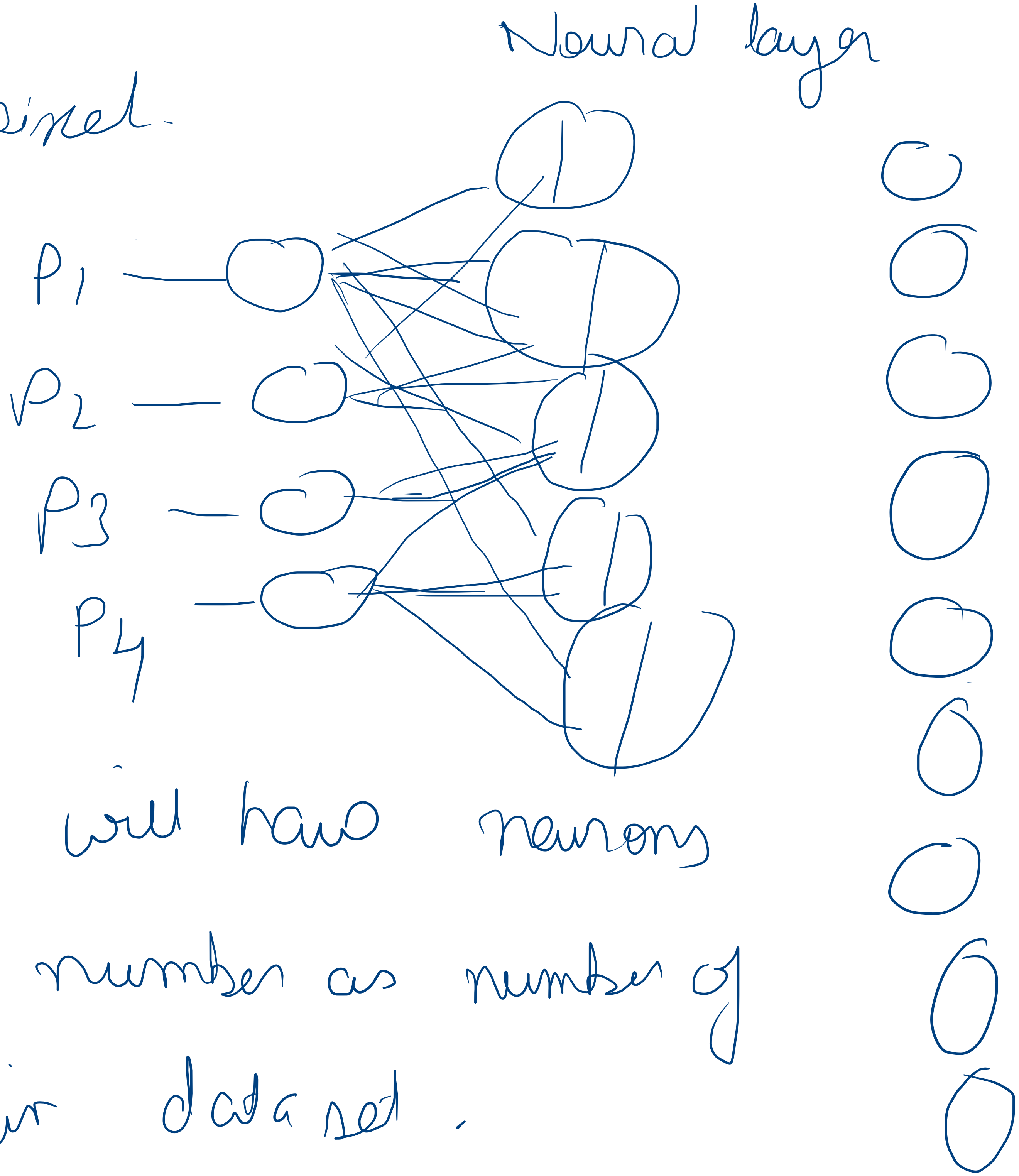
Neural layer



Input layer will have neurons in same number as number of features in dataset.

$2 \times 2 = 4$ pixel.





Input layer will have neurons in same number as number of features in dataset.

iris = input parameters (4)

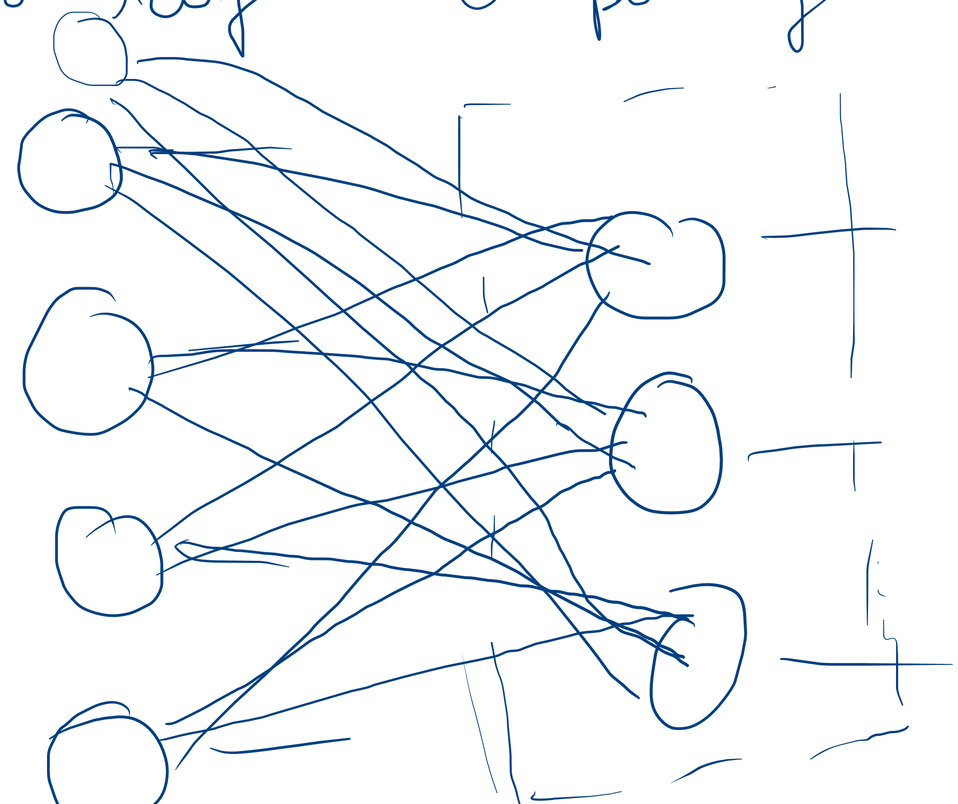
Perceptron

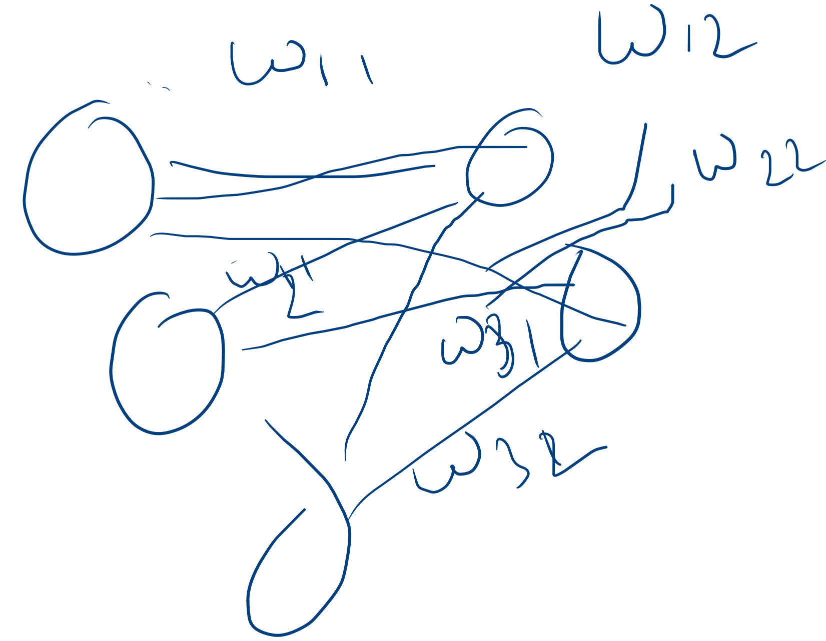
output class (3)

random

1 →
2
3

Input layer   output lay





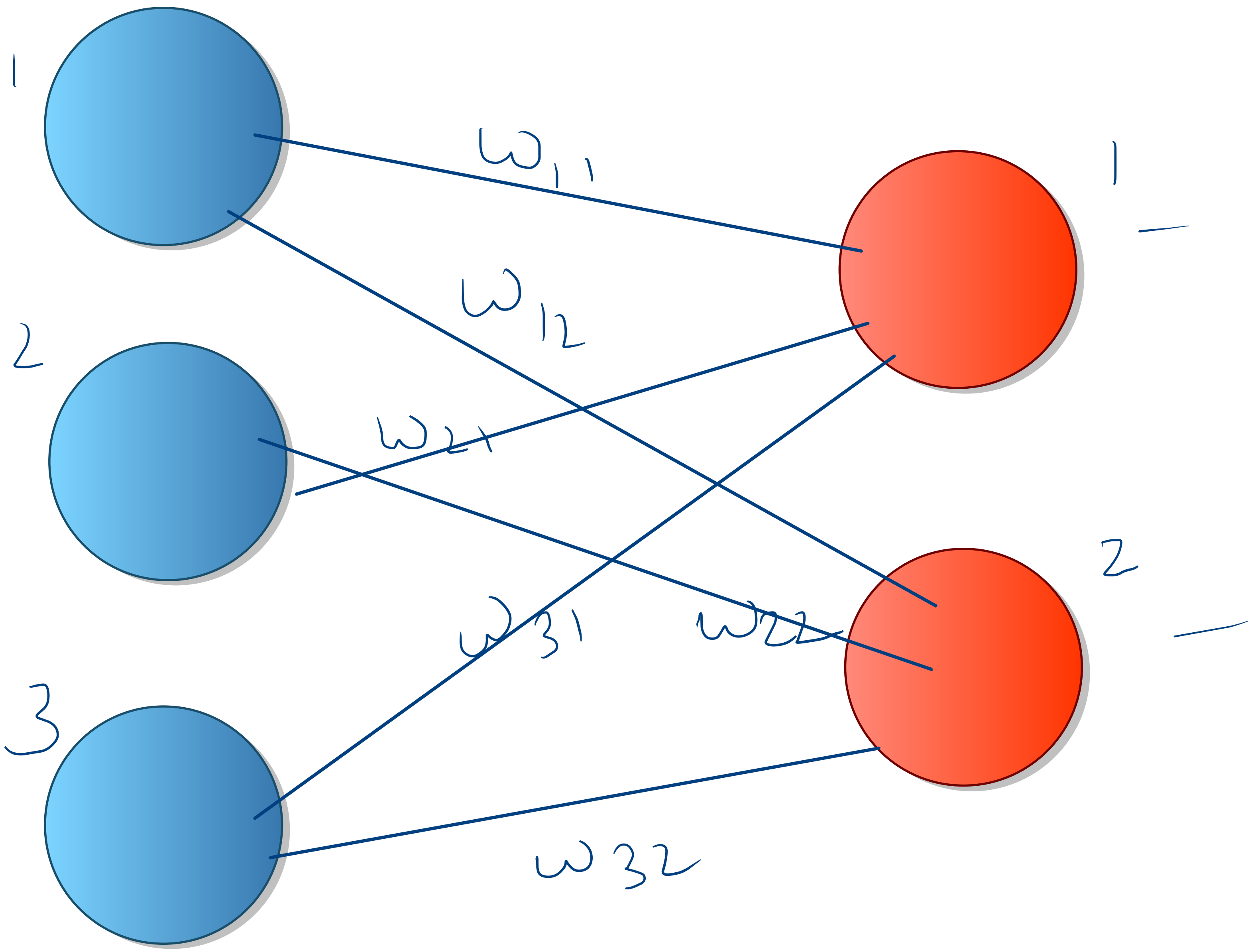
150

Perceptron model

For instance

1

2)



η

$$\theta^{\text{next step}} = \theta - \eta \nabla_\theta MSE(\theta)$$

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$

Vector = $y = \theta^T \cdot X$

$$\theta^T = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \cdot X = [1, x_1, x_2, \ldots x_n]$$

$\theta_0 \times 1 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$

$$\theta^{\text{next step}} = \theta - \eta \nabla_\theta MSE(\theta)$$

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$

Vector = $y = \theta^T \cdot X$

$$\theta^T = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \\ \theta_n \end{bmatrix}$$

$$X = [1, x_1, x_2, \dots x_n]$$

$$\theta_0 \times 1 + \theta_1 x_1 + \theta_2 x_2$$

$$+ \theta_n x_n$$

$$\hat{\theta} = (X^T \cdot X)^{-1} \cdot X^T \cdot y$$

best fun

To solve this equation how much time is req. is governed by $(X^T \cdot X)^{-1}$ size of matrix. = y ~~x~~ is how n feature

Data

The Time / Computational complexity =

$O(n^{2.4})$ or $O(n^3)$

If I double the number of points
Then time taken be 8 times.

For eg: 1 = time .01 sec

2 = .01×8 = .08

4 = .08 × 8 = .64

8 = .64 × 8 = 6.72

16 = 6.72 × 8 = 48

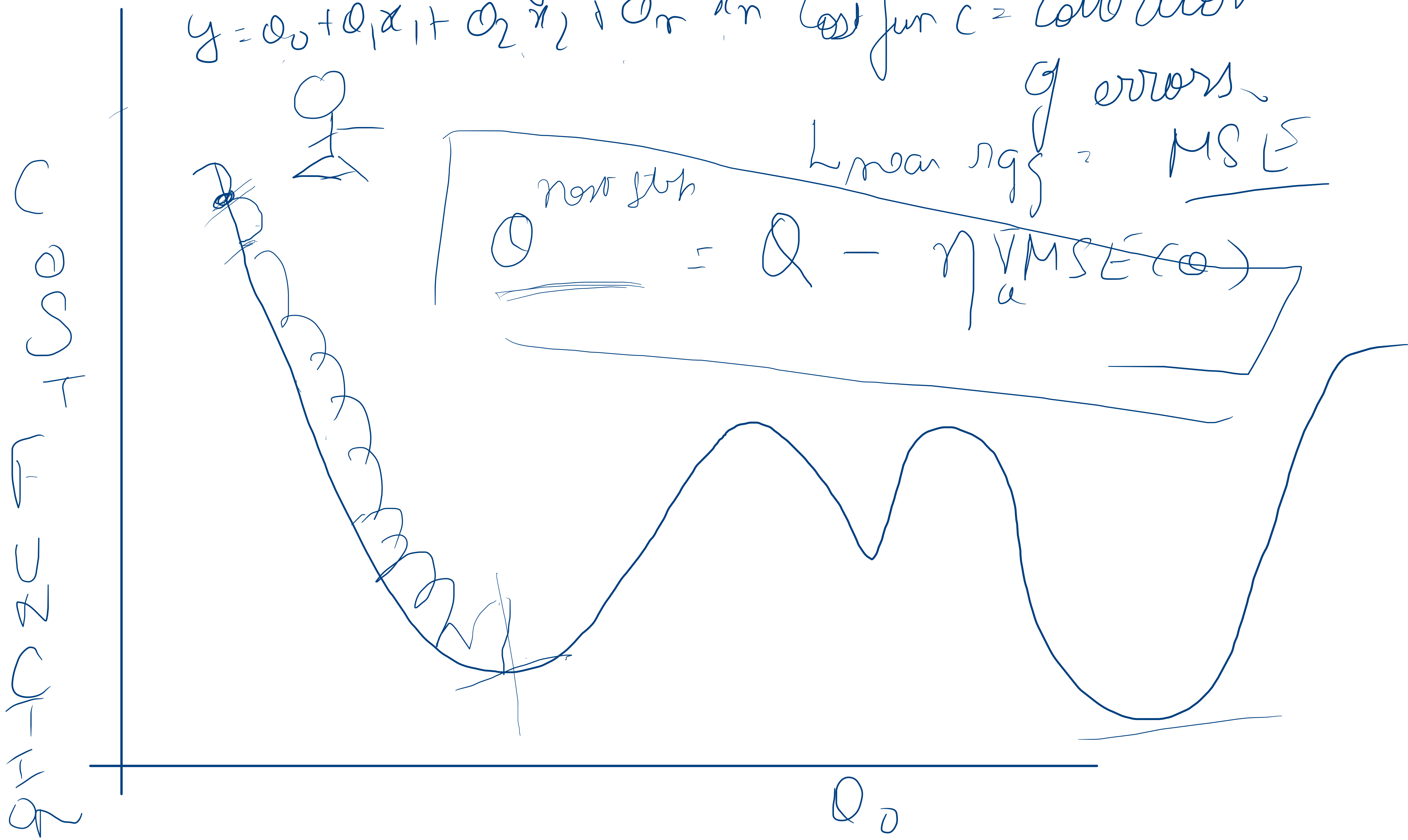
Method called gradient descent.

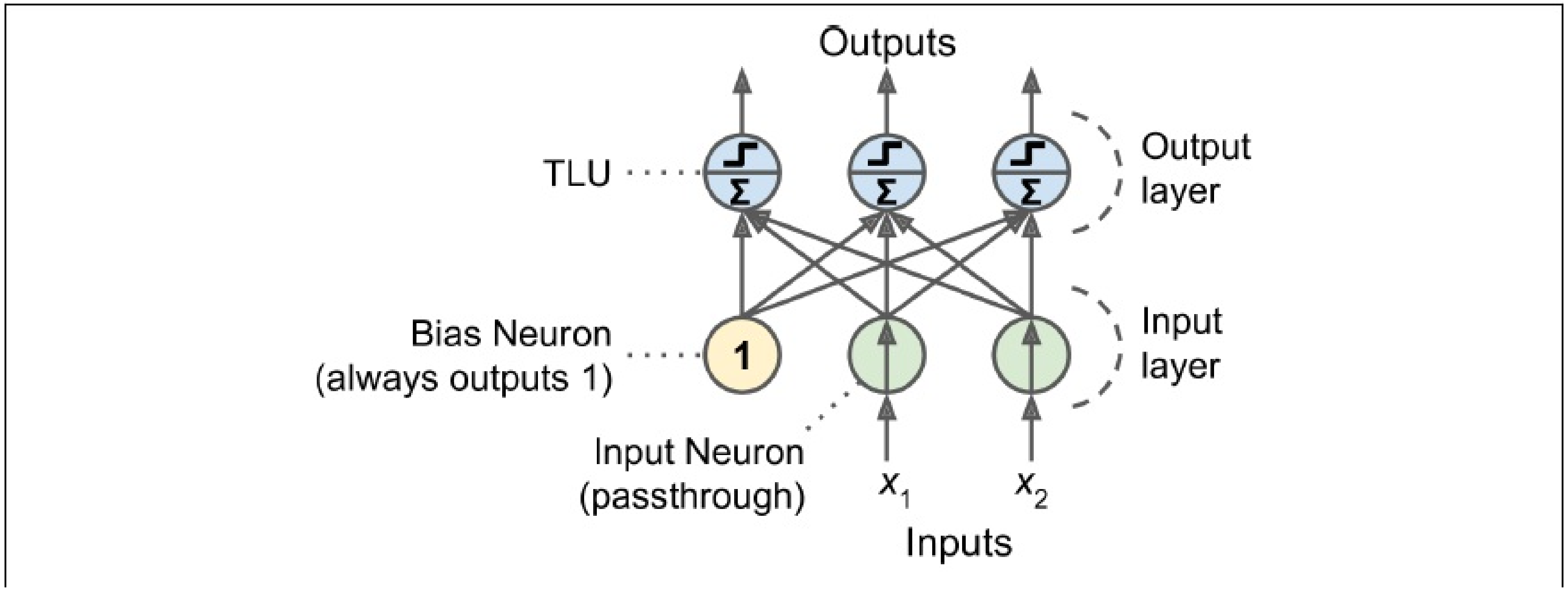
$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_n x_n$

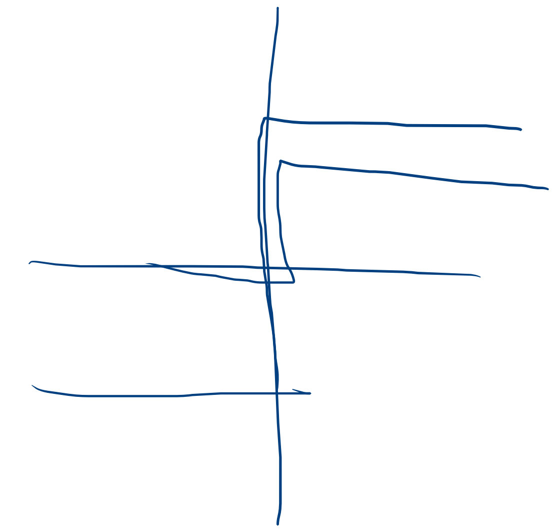
Cost fun c = collection of errors

Lmean rgs = MSE

$$\theta^{\text{new step}} = \theta - \eta \nabla_{\theta} MSE(\theta)$$







$$w_{i,j}^{(\text{next step})} = w_{i,j} + \eta\left(y_j - \hat{y}_j\right)x_i$$

- $w_{i,j}$ is the connection weight between the i^{th} input neuron and the j^{th} output neuron.
- x_i is the i^{th} input value of the current training instance.
- $\hat{y}_j$ is the output of the j^{th} output neuron for the current training instance.
- y_j is the target output of the j^{th} output neuron for the current training instance.
- η is the learning rate.

The decision boundary of each output neuron is linear, so Perceptrons are incapable of learning complex patterns (just like Logistic Regression classifiers). However, if the training instances are linearly separable, Rosenblatt demonstrated that this algorithm would converge to a solution.[7] This is called the *Perceptron convergence theorem*.

Scikit-Learn provides a `Perceptron` class that implements a single TLU network. It can be used pretty much as you would expect—for example, on the iris dataset (introduced in Chapter 4):