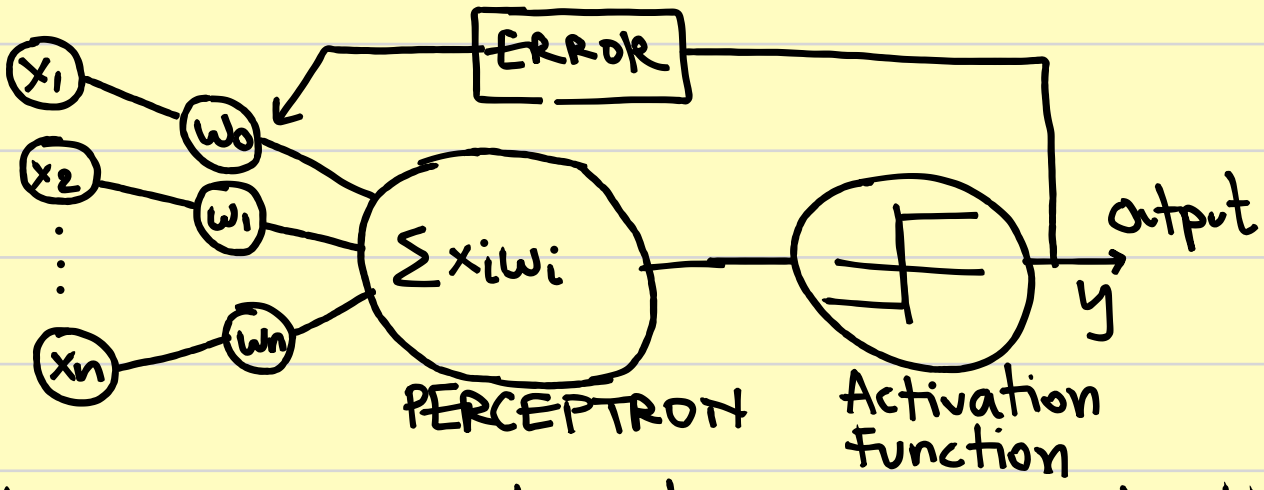
DEEP LEARNING by hand

SINGLE NEURON PERCEPTRON

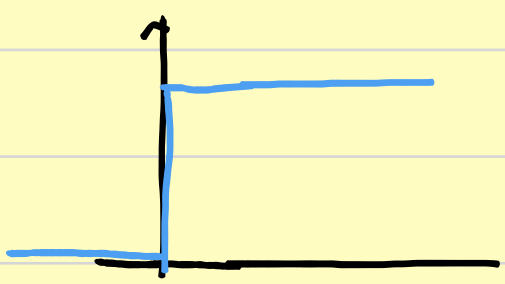


Step 1. Multiply all input values x_i with their corresponding weights w_i and then calculate the weighted sum.

$$\sum_{i=1}^{n} x_i w_i = x_1 w_1 + x_2 w_2 + \cdots + x_n w_n$$

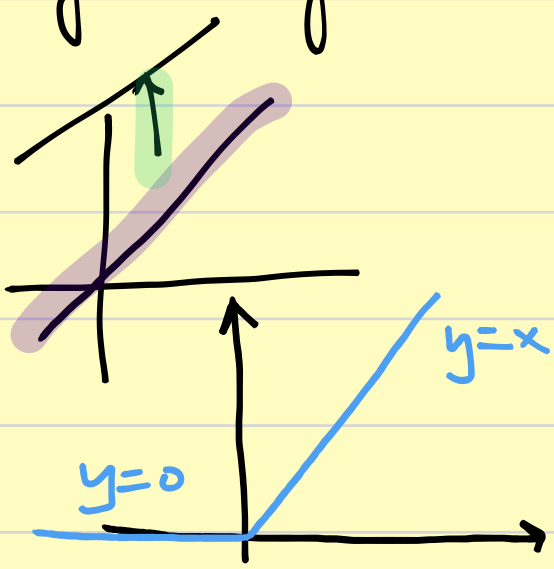
Step 2. An activation function is applied to the above giving us an output of the form

$$\hat{y} = f\left(\sum w_i x_i + b\right)$$



Step Function

Sigmoid Funct.



ReLU
Rectifier
Lined
Unit

Step 3. We need to calculate the weight values which make the equation $\hat{y} = f(\sum w_i x_i + b)$ as similar as possible to a prediction y.

We calculate the error: $C = \frac{1}{2}(y - \hat{y})^2 =$

$$= \frac{1}{2}(y - \sum w_i x_i - b)^2$$

We call this error "cost" and we want to find the values of $w_1, w_2, \dots, w_n$ that minimize the cost.

$$\frac{dC}{dw_1} = \frac{1}{2} \cdot 2 \cdot (y - w_1 x_1 - w_2 x_2 - \dots - w_n x_n)(-x_1)$$

$$\frac{dC}{dw_2} = \frac{1}{2} \cdot 2 (y - w_1 x_1 - w_2 x_2 - \dots - w_n x_n)(-x_2)$$

$$\dots$$

$$\frac{dC}{dw_n} = \frac{1}{2} \cdot 2 (y - w_1 x_1 - w_2 x_2 - \dots - w_n x_n)(-x_n)$$

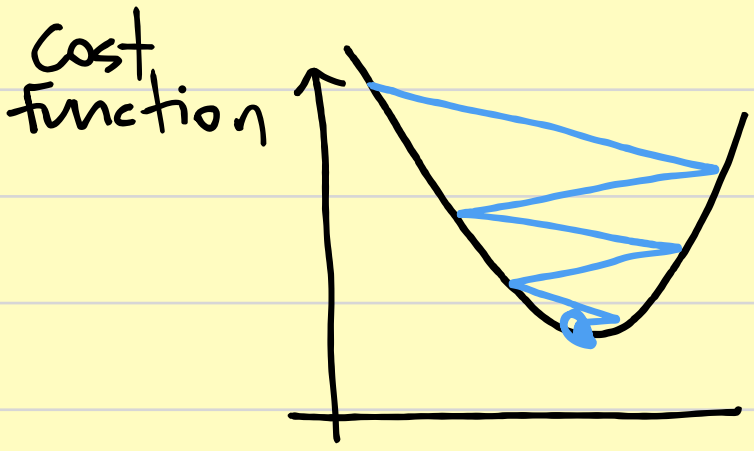
these derivatives are called gradients.

Step 4. Now to reach a minima we will start taking small steps towards the direction of the minima (opposite direction of the gradient).

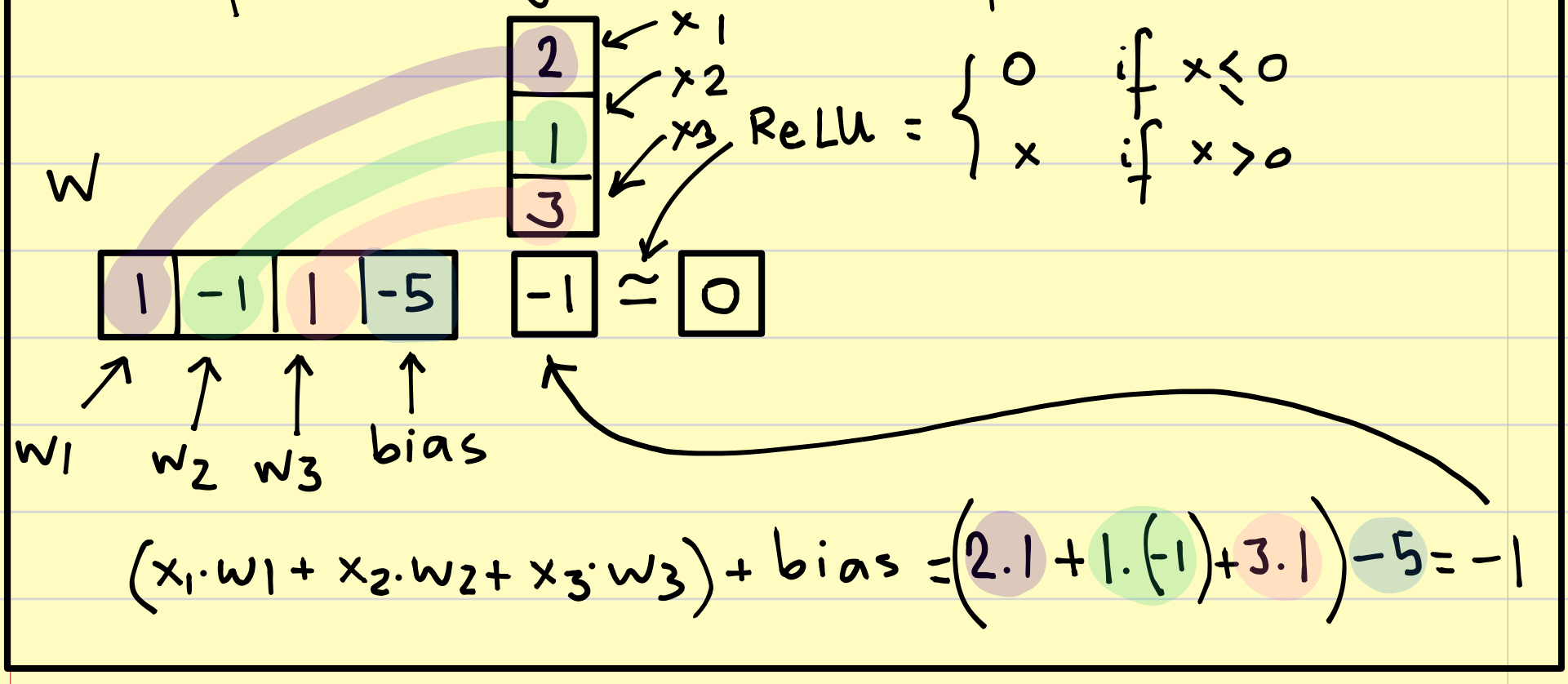
GRADIENT DESCENT.

We do so by updating the weights by a small

amount each time. This small amount is called LEARNING RATE.

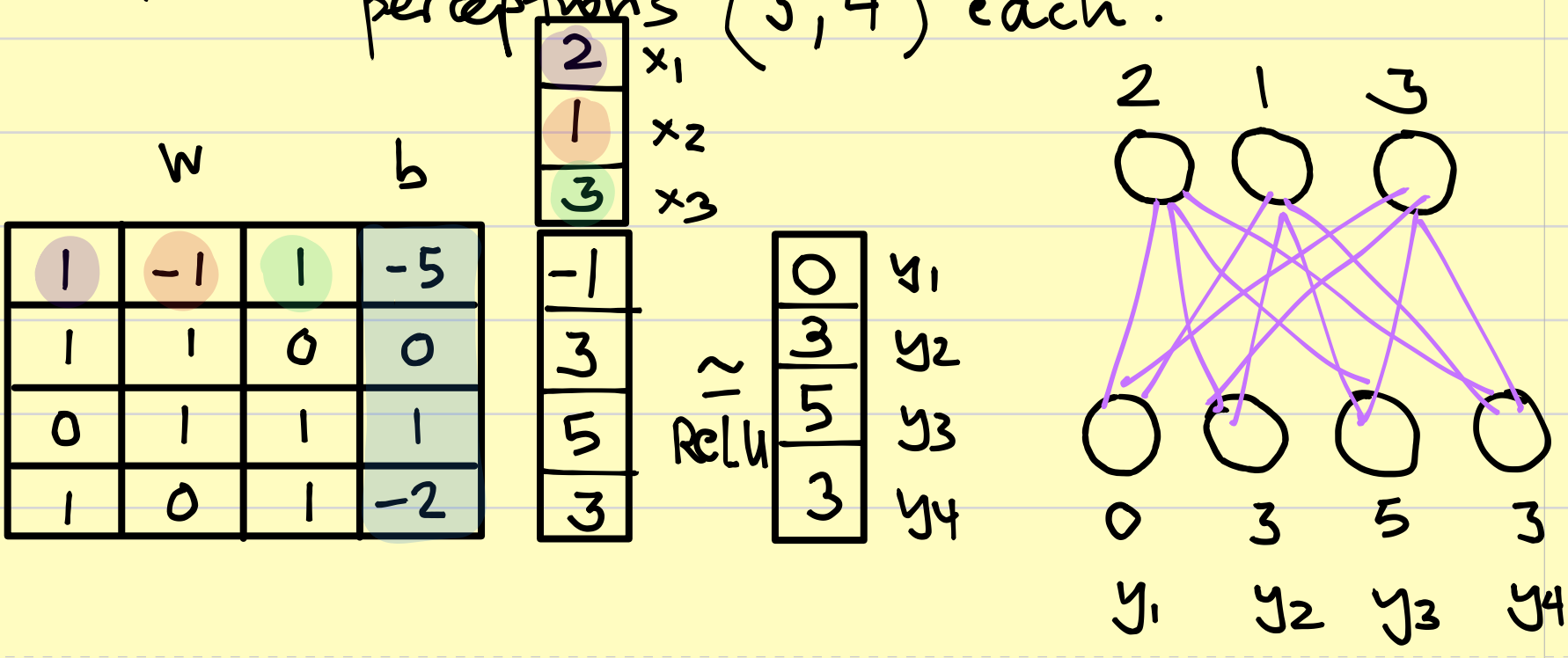


Example 1. Single Neuron Perceptron Forward Pass



$$(x_1 \cdot w_1 + x_2 \cdot w_2 + x_3 \cdot w_3) + bias = (2.1 + 1.(-1) + 3.1) - 5 = -1$$

Example 2. Calculate the output of two layes of perceptrons (3,4) each.



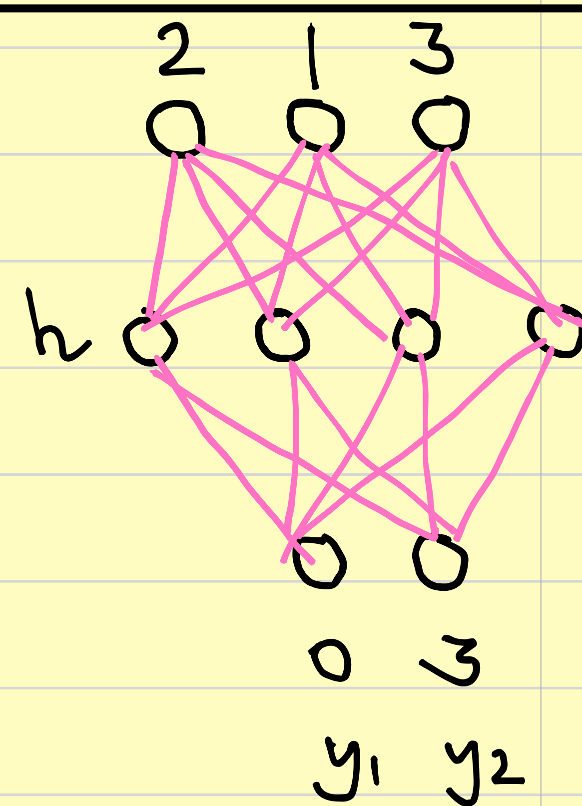
$$(2\cdot 1 - 1\cdot 1 + 1\cdot 3) - 5 = -1$$

$$(2\cdot 1 + 1\cdot 1 + 0\cdot 3) + 0 = 3$$

$$(2\cdot 0 + 1\cdot 1 + 1\cdot 3) + 1 = 5$$

$$(2\cdot 1 + 0\cdot 1 + 1\cdot 3) - 2 = 3$$

Example 3. HIDDEN LAYER



$$\underbrace{\begin{array}{ccc|c} 1 & -1 & 1 & -5 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & -2 \end{array}}_{3x}\ (4x) \quad \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix} \quad \begin{bmatrix} -1 \\ 3 \\ 5 \\ 3 \end{bmatrix} \underset{\text{ReLU}}{\sim} \begin{bmatrix} 0 \\ 3 \\ 5 \\ 3 \end{bmatrix} = h$$

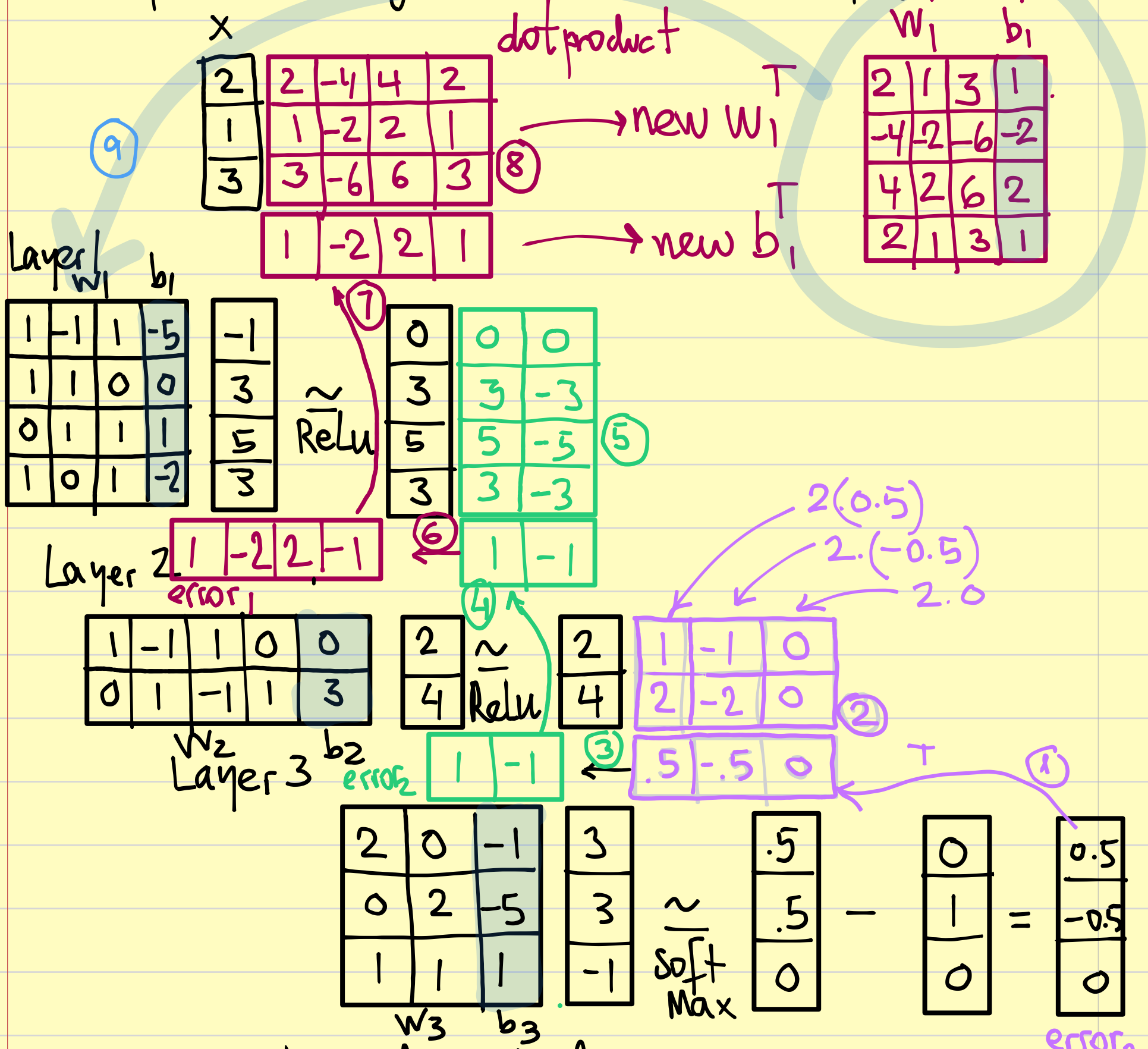
W_1 b_1

$$\underbrace{\begin{array}{cccc|c} 1 & 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 & 1 \end{array}}_{4x}\ (2x) \quad \begin{bmatrix} -2 \\ 3 \end{bmatrix} \underset{\text{ReLU}}{\sim} \begin{bmatrix} 0 \\ 3 \end{bmatrix}$$

$$(1\cdot 0 + 1\cdot 3 - 1\cdot 5 + 0\cdot 3) + 0 = -2$$

$$(0\cdot 0 + 0\cdot 3 + 1\cdot 5 - 1\cdot 3) + 1 = 3$$

Example 4. Feed forward and Backpropagation.



SoftMax: Positive values Σvalues=1
Negative values → 0.

① Transpose the vector predicted - vector target

② Multiply the vector predicted error$_3$ with Output of Layer 2. DOT-PRODUCT!

③ Multiply the predicted error$_3$ with W_3 matrix.

$(0.5).2 + (-0.5).0 + 0.1 = 1$

$(0.5).0 + (-0.5).2 + 0.1 = -1$

④ Transpose of the $error_3 \cdot W_3$

⑤ DOT PRODUCT of $(error_3 . W_3)^T \cdot$ output Layer 1.

⑥ Multiply the predicted $error_2$ with W_2 matrix.

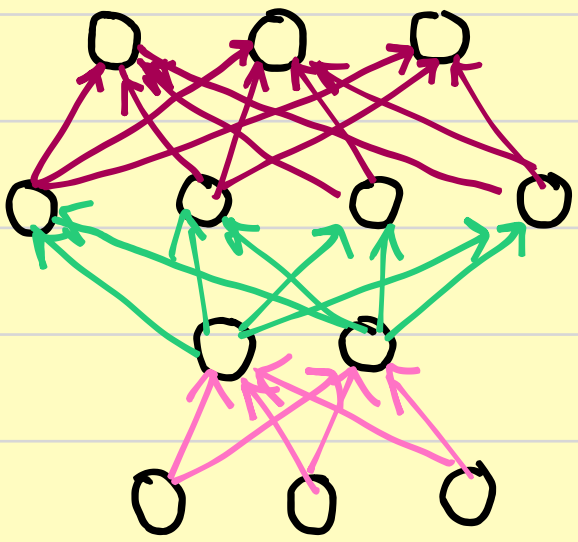
$(1.1) + (-1) . 0 = 1$

$(1)(-1) + (-1) \cdot 1 = -2$

⑦ Transpose of the $error_2 \cdot W_2$

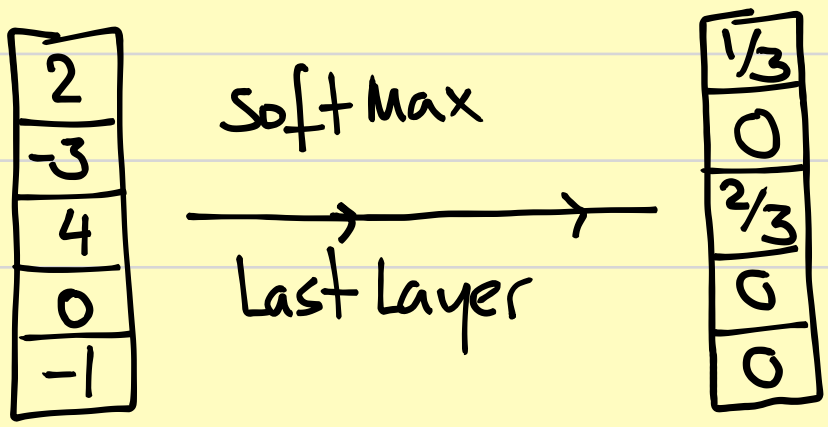
⑧ DOT PRODUCT of $(error_2 \cdot W_2)^T .$ Input

⑨ Substitute the new updated weights w_1' & b_1'



Graphical Explanation

Softmax explanation: ≤ 1



$$2 + 4 = 6$$

$$\frac{2}{6} + \frac{4}{6} = 1$$

$$\frac{1}{3} + \frac{2}{3}$$

TRANSFORMERS

Transformers are neural networks that eliminate the need for labeled datasets and convolution in the network.

The two main mechanisms that allow for this are:

1) ATTENTION WEIGHTING
2) FEEDFORWARD NETWORKS (FFN)

Attention weighting is a technique by which the model learns which part of the incomming sequence needs to be focused on. It screens the data and recognizes what is relevant.

FFN are in this context a regular multilayer perception acting on a batch of datavectors.

Combined FFN & AW, it produces "position dimension".

FFN mechanism (see above MLPerception).

What is the attention mechanism?

The transformer architecture can be seen as a

combination of two shells:

The outer shell is a combination of attention-weighting and a FFN.
The inner shell consist on a self-attention mechanism and is part of the attention-weighting feature.

Example: The attention weight matrix A (given) is obtained by feeding the input features into the architecture. This matrix tries to find the most relevant parts in the input sequence.

x_1	x_2	x_3	x_4
2	0	0	2
0	1	0	0
0	2	1	0
0	0	1	1
2	0	0	0
1	0	1	1

1. Weight W_Q

1	1	0	0	0	0
0	1	0	1	0	0
0	0	1	0	1	1

2	1	0	2
0	1	1	1
3	2	2	1

$q_1\ q_2\ q_3\ q_4$

$W_Q \cdot X = q$

QUERY VECTOR

2. key W_K

1	0	0	0	1	0	0
2	0	1	0	0	0	0
3	0	0	1	0	0	0

one-hot-Encoding

$W_K \cdot X = k$

3. Value W_V

$W_V \cdot X = V$

4. The next step is to multiply $k \cdot q$ with dot product

5 The next step is to scale/normalize each element

$$z_i = \frac{x_i - \mu}{\sigma_x}$$

6 The next step is to apply softmax to get the prediction

7 Backwards propagation as in FFN.

Apfel → 0 0 0 Mango → 0 1 1

Orange → 0 0 1 Ananas → 1 0 1

Erdbeer → 0 1 0 Zitrone → 1 1 0

Birne → 1 0 0 Melone → 1 1 1

$$\begin{bmatrix} 0{,}01 \\ 0{,}99 \\ 0{,}0 \end{bmatrix} \rightarrow \text{Erdbeer}$$

y_{pred}