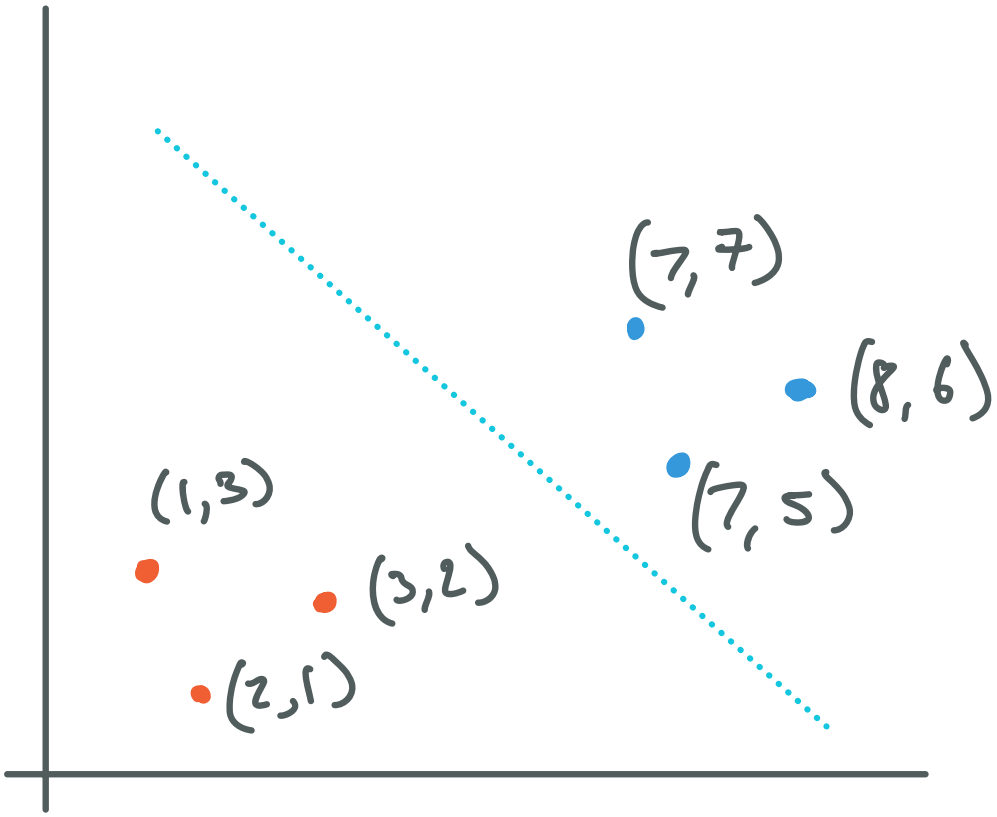
Linearly seperable data



"randomly" initialized weight matrix

working in 2D so weight matrix (really vector) has size n+1 w/ n = dimension

"randomly" initialized to 0.5

$$w = (0.5, 0.5, 0.5)$$

We also have a data matrix (this time an actual matrix)

that holds each data point as a row. We construct it as such

$$D = \begin{pmatrix} 1, x_1, y_1, \bullet \\ \vdots \\ 1, x_n, y_n, \bullet \end{pmatrix}$$

So, b/c we are working in 2D space, each row is four elements long. The first element of each row is always 1. This is b/c, what we are ultimately constructing is a $y = mx + b$ relation that splits the data. Therefore, in order for the b value to be non-zero, the first element of each row must be 1. This will become more obvious when we do the actual lin-alg. Then, the next two values are the x & y position, & the final element of each row is the data type,

(as the perceptron is ultimately a classification algorithm). In this case, our data types are blue & orange. We will call blue 1 & orange 0, so our real data matrix for this example becomes

$$D = \begin{pmatrix} 1 & 1 & 3 & 0 \\ 1 & 7 & 7 & 1 \\ 1 & 8 & 6 & 1 \\ 1 & 7 & 5 & 1 \\ 1 & 3 & 2 & 0 \\ 1 & 2 & 1 & 0 \end{pmatrix}$$

We can therefore see that

$D \in \mathbb{R}^{m \times n}$ s.t. m = spatial dimension + 2
&
n = # of data points

Now we 'train' the weight matrix. We take the dot product of the weight vector w/ the first row (& only first three entries) of the data matrix

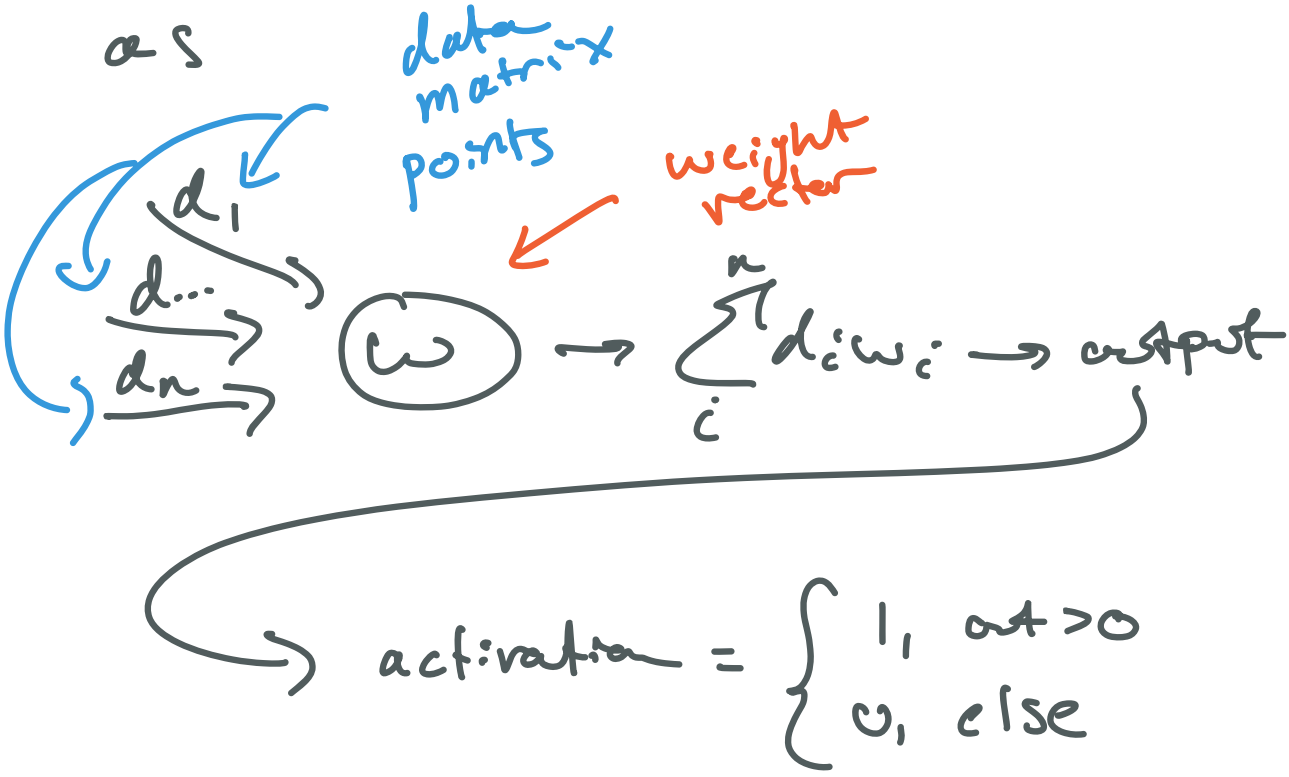
$$\text{So} \quad \text{output} = \begin{pmatrix} 0.5 \\ 0.5 \\ 0.5 \end{pmatrix} (1 \;\; 1 \;\; 3)$$

$$\Rightarrow \text{output} = 2.5$$

We then run this output through a simple activation function which for this model we define as

$$f(x) = \begin{cases} 1, & \text{out} > 0 \\ 0 & \text{else} \end{cases}$$

OR you can think of it as



Then, we take this final value (0 or 1) & compare it to what that row in the data matrix actually corresponded to. Remember, we only used the first 3 elements of each data matrix row, as the last value is what we test on. Also recall we defined orange = 0 & blue = 1 ... So...

we take the dot product of the weight vector w the i^{th} row (again, only first 3 elements) of the data matrix & test if we get the correct output.

→ If we do, we do <u>not</u> change the weight vector.

→ If we don't (prediction ≠ actual value), we update each element of the weight matrix according to

$$w_j = \eta \cdot error \cdot d_{ij}$$

where

w_j = j^{th} element of the weight matrix

η = learning rate (hyperparameter)

error = actual − prediction

(so either 1 or -1 or 0)

d_{ij} = j^{th} element of i^{th} row of the data matrix.

Calculating the Weight Matrix by hand

So now, lets calculate the weight matrix after 1 epoch

→ 1 epoch means one pass through the entire data matrix. OR, taking the dot product of the weight vector w/ each

data matrix row exactly once.

initially some arbitrary

Recall

$$\vec{w} = \begin{pmatrix} 0.5 \\ 0.5 \\ 0.5 \end{pmatrix}$$ and

$$D = \begin{pmatrix} 1 & 1 & 3 & 0 \\ 1 & 7 & 7 & 1 \\ 1 & 8 & 6 & 1 \\ 1 & 7 & 5 & 1 \\ 1 & 3 & 2 & 0 \\ 1 & 2 & 1 & 0 \end{pmatrix}$$

So...

$$dp_1 = \begin{pmatrix} .5 \\ .5 \\ .5 \end{pmatrix} (1 \; 1 \; 3) = 2.5 > 0 \rightarrow dp_1 = 1$$

however, we see $dp_1 \neq 1$,
so we update the weight vector

(for simplicity, η = learning rate = 1)

error = −1

$$\rightarrow \vec{w} = \begin{pmatrix} -.5 \\ -.5 \\ -1.5 \end{pmatrix}$$

$$dp_2 = \begin{pmatrix} -.5 \\ -.5 \\ -1.5 \end{pmatrix} (1 \ 7 \ 7) = -14.5 \neq 0 \rightarrow dp_2 = 0$$

which is false, so we update $\vec{w}$ again

error = 1

$$\rightarrow \vec{w} = \begin{pmatrix} -.5 \\ -3.5 \\ -10.5 \end{pmatrix}$$

$$dp_3 = \begin{pmatrix} -.5 \\ -3.5 \\ -10.5 \end{pmatrix} (1 \ 8 \ 6) = -91.5 \neq 0$$

$\rightarrow dp_3 = 0$ again, false so we update

error = 1

$$\rightarrow \vec{w} = \begin{pmatrix} -0.5 \\ -28 \\ -63 \end{pmatrix}$$

$$dp_4 = \begin{pmatrix} -0.5 \\ -28 \\ -63 \end{pmatrix} (1 \quad 7 \quad 5) = \text{idk but def}$$

not $> 0 \rightarrow dp_4 = 0$ which is false so we update again!

error = 1

☆ You can see here our weight values are quickly increasing. This is where the learning rate comes into play. If we chose a smaller learning rate, the weight values would be increasing much more slowly.

→ Too high a learning rate, & you may overshoot the optimal weights. Too small a learning rate, & it'll take a while for the weights to

find the optimum (wastes compute)

So now

$$\vec{w} = \begin{pmatrix} -0.5 \\ -196 \\ -315 \end{pmatrix}$$

$$dp5 = \begin{pmatrix} -0.5 \\ -196 \\ -315 \end{pmatrix} (1 \;\; 3 \;\; 2) = -1218.5 \neq 0$$

$\rightarrow dp5 = 0$ which is TRUE so no update!

$$dp6 = \begin{pmatrix} -0.5 \\ -196 \\ -315 \end{pmatrix} (1 \;\; 2 \;\; 1) = -707.5 \neq 0$$

$\rightarrow dp6 = 0$ which is TRUE so no update.

→ Therefore, after one epoch (pass through the data matrix) our weight vector is

$$\vec{w} = \begin{pmatrix} -0.5 \\ -196 \\ -315 \end{pmatrix}$$

Which is obviously not correct nor ideal, but that is why you have to choose an appropriate learning rate as well as do more than 1 epoch

→ After a successful training run, you will have a weight vector in the form

$$\vec{w} = \begin{pmatrix} \alpha \\ \beta \\ \delta \end{pmatrix}$$

From which you turn into a $y = mx + b$ equation of

α is associated with b, β with x & γ with y

$$\Rightarrow \gamma y + \beta x + \alpha b = 0$$

$$\Rightarrow y = -\left(\frac{\beta}{\gamma} x + \frac{\alpha}{\gamma}\right)$$

→ And there is your decision boundary obtained from the weight vector that gets updated during training!!

→ It is easy to see how this can scale to n-dimensions & why we usually get computers to do this – the math is very tedious.

→ Unfortunately, however, the perceptron can only separate linearly separable data, which most real world data/functions/patterns are not 😞

↳ To address this, we must add layers & non-linearities!

↳ Which we will get to next week!!