



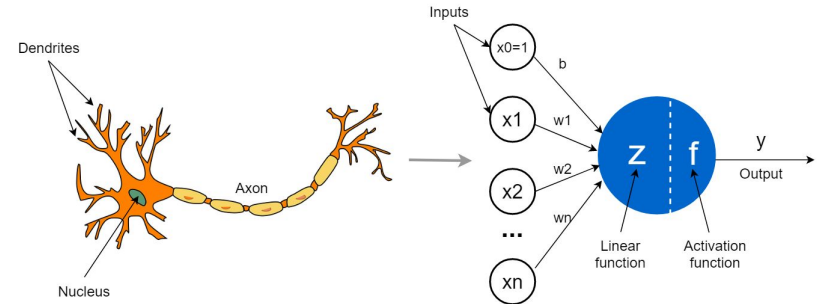
Introduction to Neural Networks



THE KNOWLEDGE HOUSE

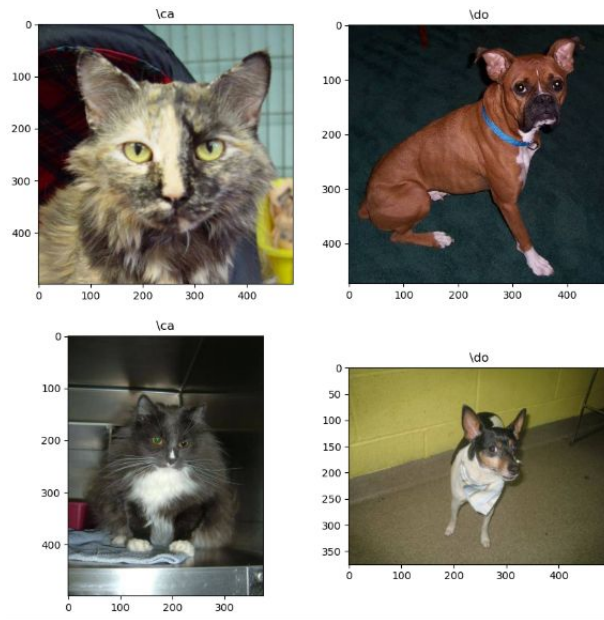
Agenda - Schedule

1. Historical Intro to Neural Nets
2. Perceptrons
3. Break
4. Gradient Descent
5. Simple Feed Forward Networks



Hyperplanes, bayes theorem, neighbors, why not just take inspiration from the brain?

Agenda - Goals

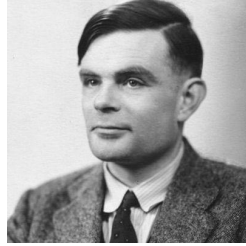


Historical/Biological Tangent



Automatons - Ancient Egypt/Greece/China

1000s of years later...



"Can Machines Think?"(1950) - Alan Turing

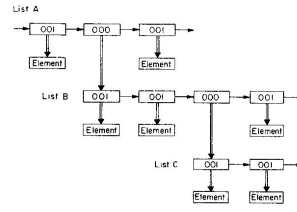
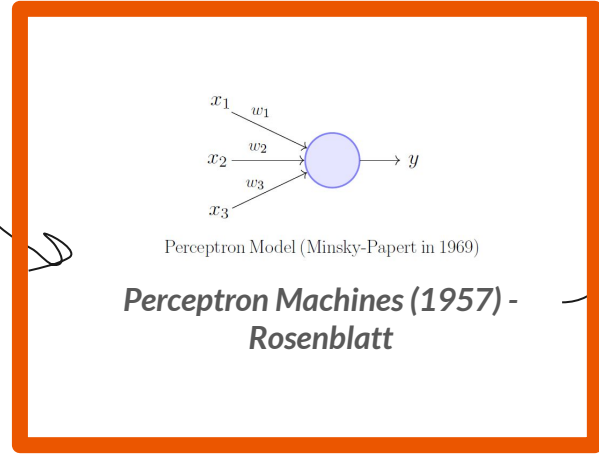


Fig. 5. Connect List B to the second box on List A; List C

Logic Theorist (1955) - Newell & Simon



Perceptron Machines (1957) - Rosenblatt

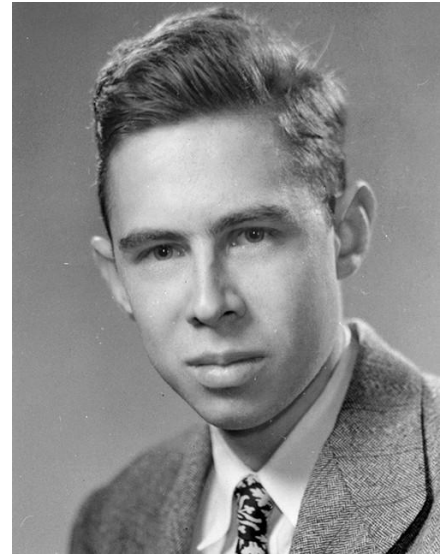
While all of these are interesting ideas and could entail years of study, let's focus in on the **"perceptron" as another attempt at "solving" intelligence.**

The Perceptron - An Imitation of our Neurons

In 1957, Frank Rosenblatt (cognitive researcher/psychologist) publishes a paper titled *"The Perceptron: A Probabilistic Model for Information Storage & Organization in the Brain"*

"If we are eventually to understand the capability of higher organisms for perceptual recognition, generalization, recall, and thinking, we must first have answers to three fundamental questions:

- 1. How is information about the ...world sensed...*
- 2. In what form is information stored...*
- 3. How does information contained in storage...influence recognition"*

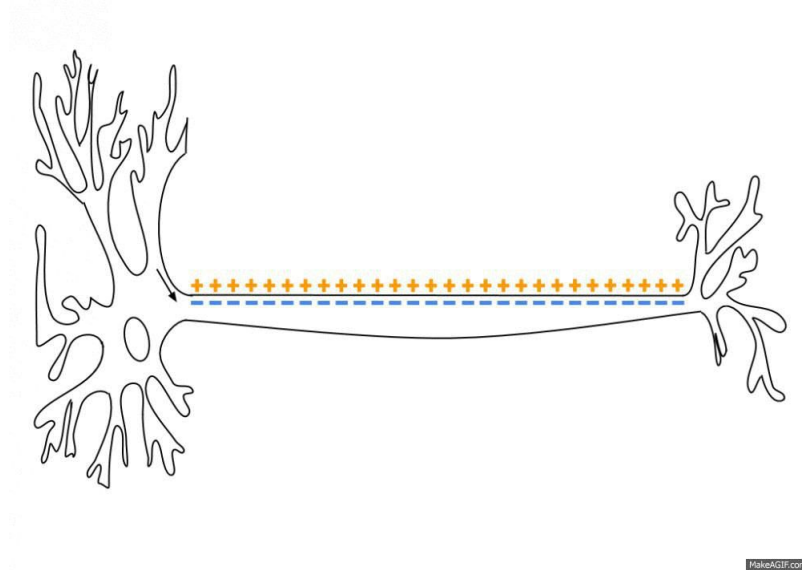


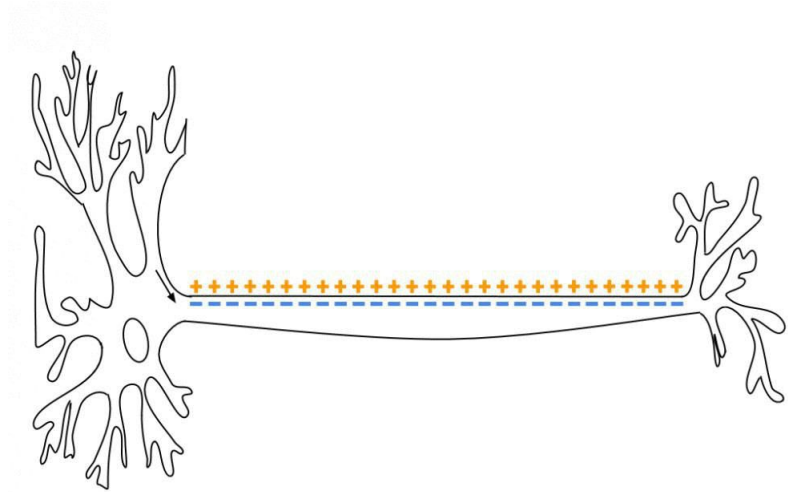
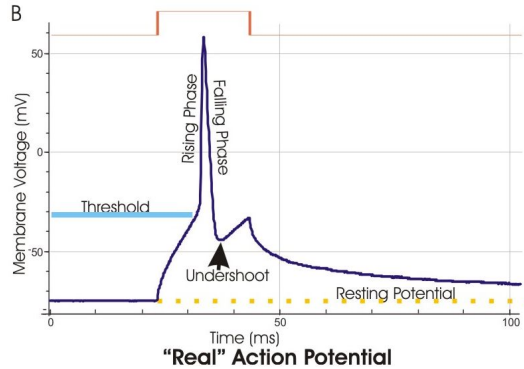
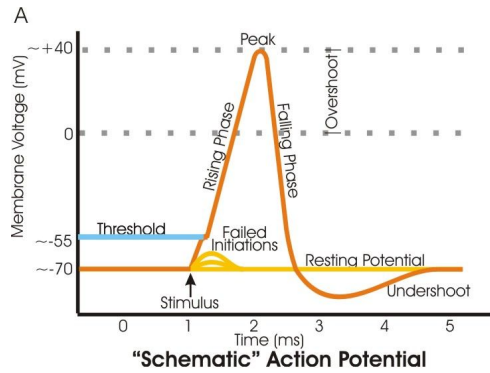
The Perceptron - An Imitation of our Neurons

Before exploring the features of a perceptron, let's recognize the most basic components of our brain: **the neuron**.

A neuron is a cell that **takes input** and provides output via electrical signals (action potential) **once we cross a certain threshold potential**.

This by no means is a biology fellowship, so we'll keep it **nice and short**

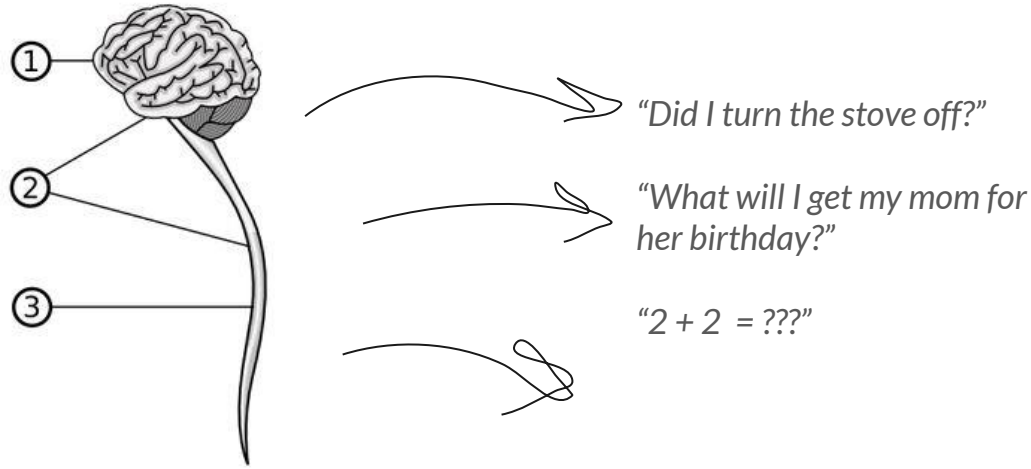




MakeAGIF.com

In rough terms, the neuron “fires” after it reaches a level of depolarization.

Once a neuron reaches around -55 microvolts, it “shoots off” a signal to another neuron in the central/peripheral nervous system.



A diagram showing the CNS:

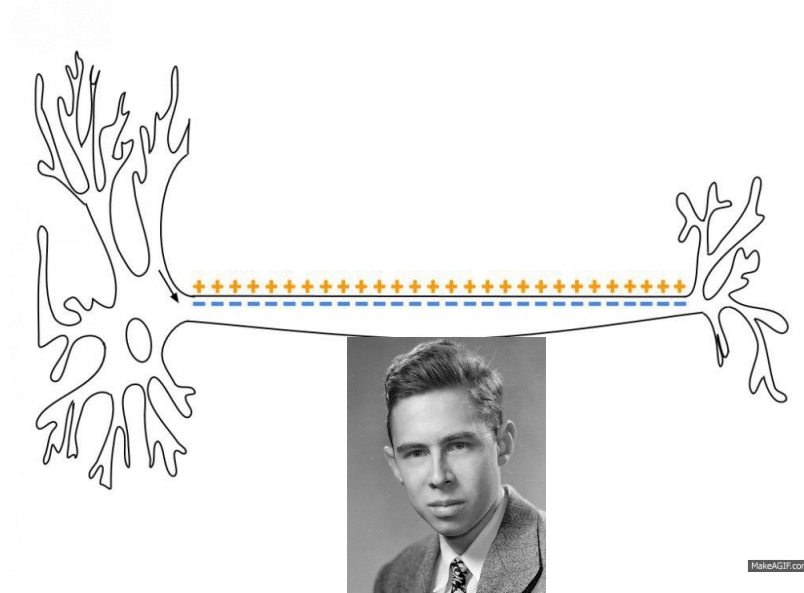
1. Brain
2. Central nervous system
(brain and spinal cord)
3. Spinal cord

(by no means a neurophysiology lesson) If you bundle enough of these neurons tightly together, you start **getting intelligent behavior**. For some reason “consciousness” also emerges, whatever that’s about.

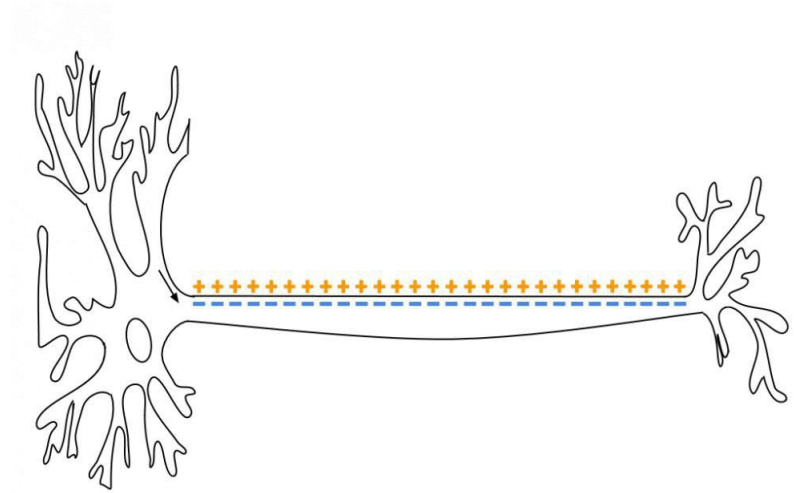
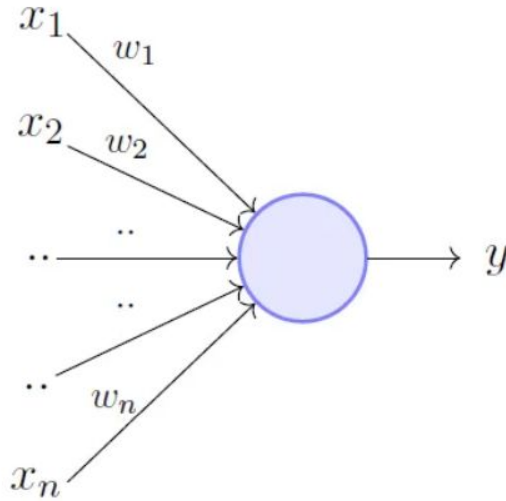
The Perceptron - An Imitation of our Neurons

Rosenblatt observed this biological phenomenon (and with the help of past research) and posited:

*“...if one understood the code or ‘wiring diagram’ of the nervous system, one should, in principle, be able to **discover exactly what an organism remembers by reconstructing the original sensory pattern...**”*



The Perceptron - An Imitation of our Neurons

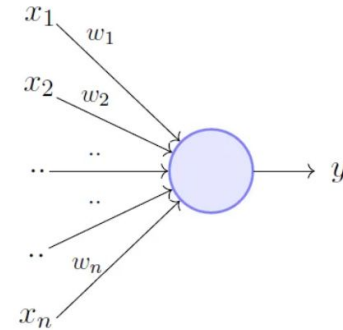


And so, that's exactly what
Rosenblatt did.

The Perceptron - An Imitation of our Neurons

The perceptron is a supervised binary learning classification algorithm that takes in input features, and learned weights.

If these summed weights surpass some activation function, we classify an output as "1." Just like a neuron!



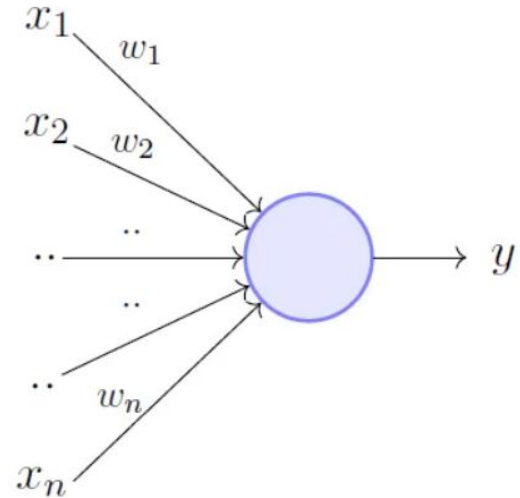
$$z = w_1x_1 + w_1x_2 + \dots + w_nx_n = X^TW$$

$$h(z) = \begin{cases} 0 & \text{if } z < Threshold \\ 1 & \text{if } z \geq Threshold \end{cases}$$

The Perceptron - An Imitation of our Neurons

However this only worked for linearly separable classifications.

Perhaps to achieve complex classifications, we don't use one neuron, but rather...

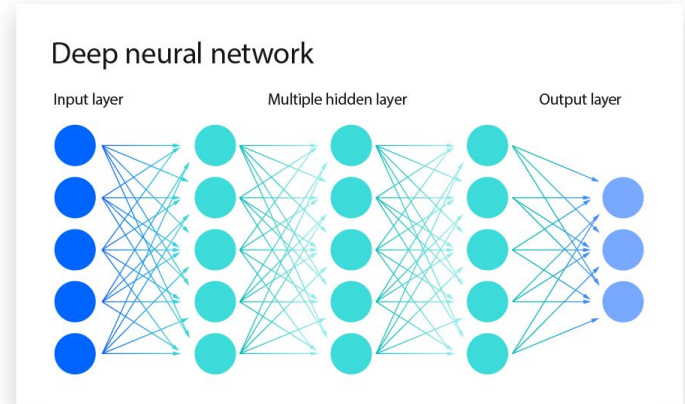


The Perceptron - An Imitation of our Neurons

However this only worked for linearly separable classifications.

Perhaps to achieve complex classifications, we don't use one neuron, but rather...

A network of neurons (a neural network?)

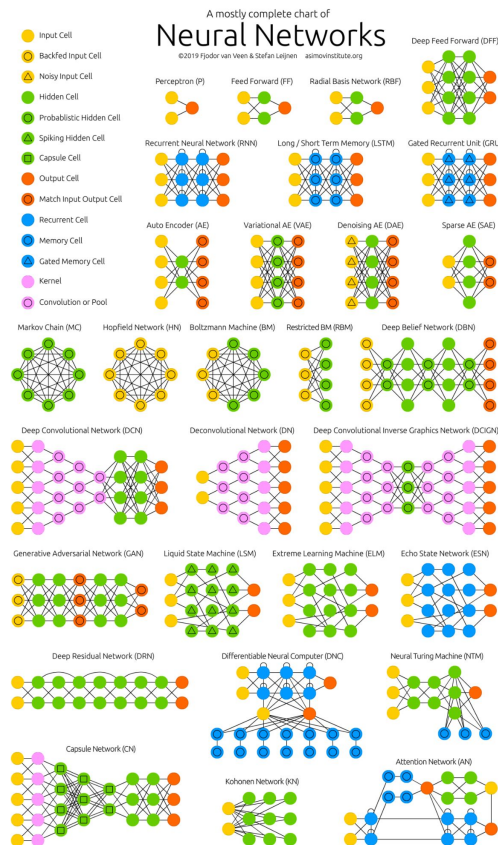


Applications of Neural Networks

There are many different applications of this powerful concept:

- Image recognition via **Convolutional Nets**
- Time series prediction via **Long-Short Term Memory Nets**
- Embedding generation via **Word2Vec**
- Word prediction via **Transformers**

Our focus will be on these last two concepts





Applications of Neural Networks

At least for the time being, almost all **novelty** (*at least the novelty that attracts the most funding*) comes from deep learning models (another word for neural nets).

Let's see a couple of ways in which deep learning methods are being used:

Neither of these women are real.



Face generation: <https://thispersondoesnotexist.com/>

Farukh can't speak Spanish



Voice generation: <https://elevenlabs.io/>






More voice generation

A Quick Aside - Neurons & Intelligence

Neural density is **probably not the end-all answer to intelligence.**

If this was the case, elephants & whales would be “more intelligent” than us. Plus, plenty of life-forms exhibit “intelligence” via other means such as **coordination (think of an ant colony)**.

However, for carbon-based life-forms like humans, **neural density seems to be a good start.**

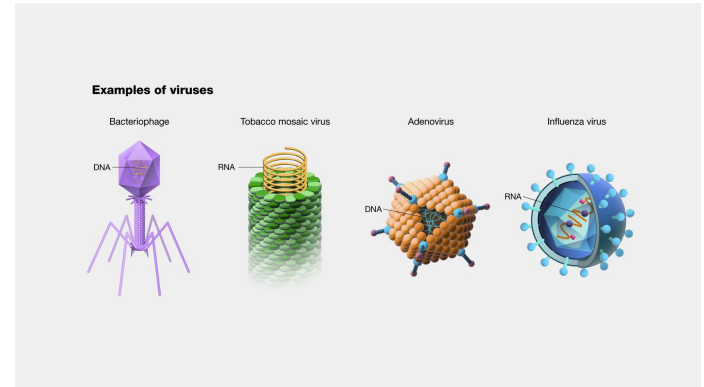
Human	8.6×10^{10}	$\sim 1.5 \times 10^{14}$	Neurons for average adult		[57][58][59]
Short-finned pilot whale	1.28×10^{11}			 [60]	[56]
African elephant	2.57×10^{11}				[60][61]

A Quick Aside - Neurons & Intelligence

Furthermore, whether or not something has to be **alive** to be “**intelligent**” (or vice-versa) **is up for debate.**

Think of a virus: a submicroscopic infectious agent that is **not alive**, but still exhibits “intelligent” behavior (mutations).

This topic eventually leads us to topics of consciousness, which while fun, is not yet applicable to our understanding of these concepts.



Perceptrons

Review - Classification: Binary Categories

In the context of **logistic regression** (and **perceptrons!**), we always consider a binary classifier to be an “on and off” switch. Is something present or is it not?

- Fraud vs not fraud
- Malignant vs nonmalignant
- Human vs not human
- Dog vs **not dog**



Is this a
dog or not
a dog?

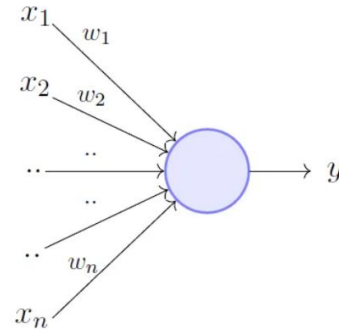
The Perceptron

Let's dive deeper into the internals of a perceptron before working through an example.

We have a few components to work with:

- **Coefficients (W)**
- **Inputs (X)**

We express these as **vectors** which we take the **dot-product** of.



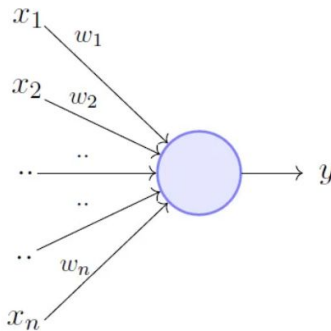
$$\hat{y} = W^T X$$

The Perceptron - A DISCLAIMER

No one uses perceptrons by themselves anymore.

It's important to KNOW what a perceptron is, and how it works, but you will never generate just one neuron (unless it's for fun).

We're all about 'networks' of neurons these days,



$$\hat{y} = W^T X$$

$$W = \begin{pmatrix} w0 \\ w1 \\ w2 \end{pmatrix}$$

$$X = \begin{pmatrix} 1 \\ x0 \\ x1 \end{pmatrix}$$

$$\hat{y} = W^T X$$

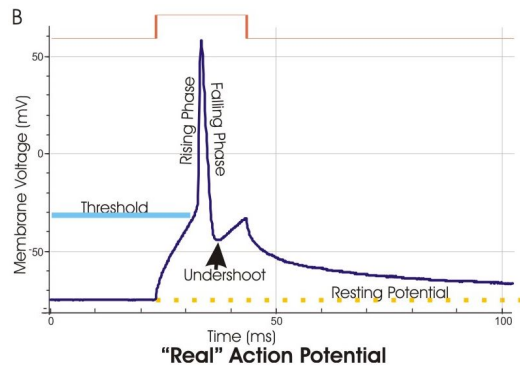
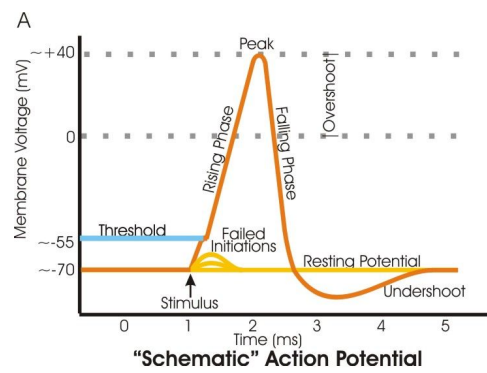
Just to break this notation down a little further. **Can anyone figure out the dot product of W (transpose) & X?** Remember, when we take the dot-product we just line up the elements, multiply them and then add them all together.

$$\hat{y} = W^T X$$

$$\hat{y} = w_0 + w_1 x_0 + w_2 x_1$$

Which model does this formula look like?
Hint: think back to linearity.

This is the formula that calculates the “*voltage*” or **output of our perceptron**.
Next we need to determine a function that “**activates**”, just our human neurons do.



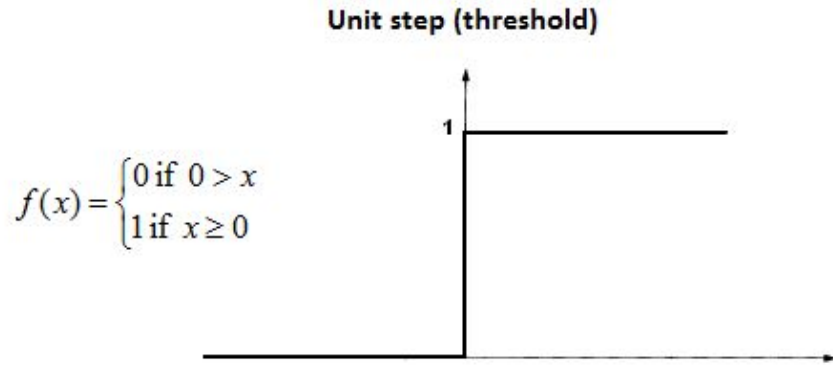
$$\hat{y} = W^T X$$

$$\hat{y} = w_0 + w_1 x_0 + w_2 x_1$$

$$h(z) = \begin{cases} 0 & \text{if } z < Threshold \\ 1 & \text{if } z \geq Threshold \end{cases}$$

Keep in mind that a neuron only “activates” once we get past a **certain threshold**. So therefore, we should probably use a function that behaves similarly to a neuron (**all or nothing**).

$$\hat{y} = W^T X$$

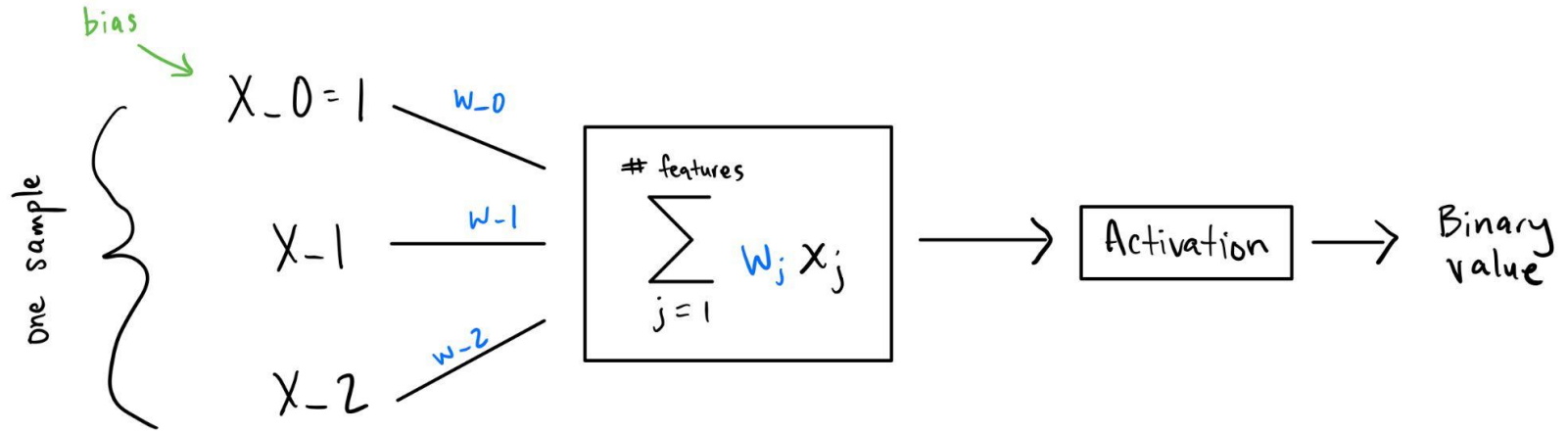


$$\hat{y} = w_0 + w_1 x_0 + w_2 x_1$$

$$h(z) = \begin{cases} 0 & \text{if } z < \text{Threshold} \\ 1 & \text{if } z \geq \text{Threshold} \end{cases}$$

There are actually **quite a few options** when it comes to choosing an activation function. However, for our **single perceptron**, we will use the **heaviside step function** (aka **binary step function**).

The idea for this function is simple: **classify a point as "1" (positive) if the output is greater than or equal to 0, otherwise classify it as "0" (negative).**

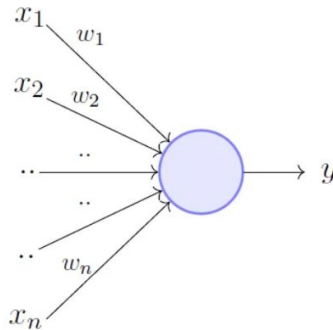


The basic idea of a perceptron is the following: using **learned weights** and **inputs**, calculate a **summed value** which will either **active or fail to activate** our neuron. Just like how our biological neurons work!

The Perceptron

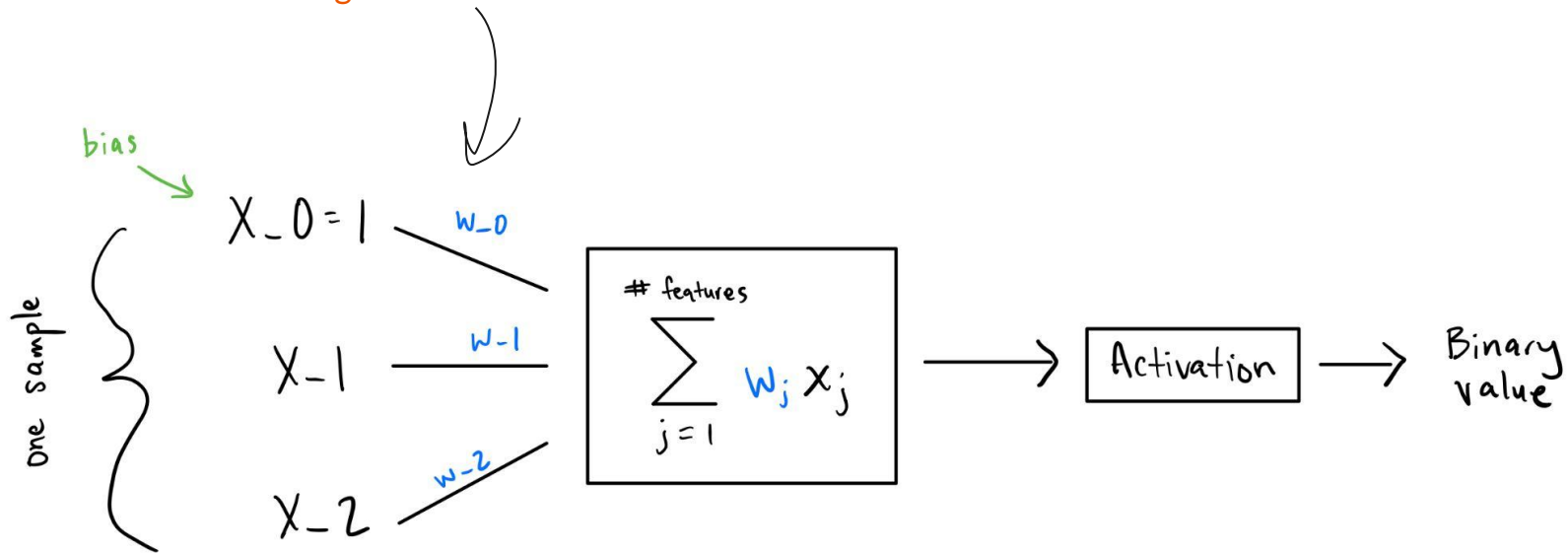
We're only scratching the surface of perceptrons, so let's take stock of terms:

- **Coefficients (W):** the **weights** we are trying to learn.
- **Inputs (X):** The **data** itself.
- **Activation Function:** A function to **"activate" the neuron**. We are currently using the **heaviside step function** as our activation.



$$\hat{y} = w_0 + w_1x_0 + w_2x_1$$

How do we actually learn these weights?



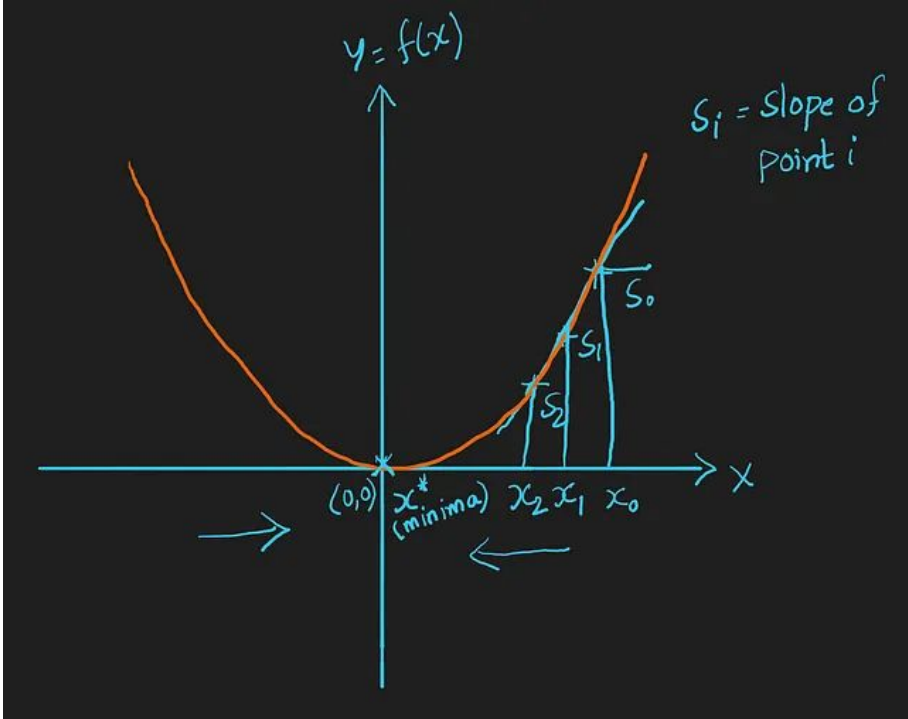
While we have seen the “structure” of a perceptron, how do we actually learn our weights? **Any guesses? Think back to previous strategies.**

One such approach is gradient descent.

Imagine our **solution space** is a **parabola**, we pick a random point and **calculate the slope (gradient)**.

We then follow that *slope down to the minima.*

That local minima tells us the best possible weights for our perceptron!





The Perceptron

Just like with LASSO & ridge, we utilize **gradient descent** to find our **optimal weights!**

We start out with some incorrect weights (typically $0, 0, 0$) and iteratively **“learn” better weights!**

$$w'_i = w_i + n(target_i - output_i)x_i$$

A diagram illustrating the perceptron weight update equation. The equation is $w'_i = w_i + n(target_i - output_i)x_i$. Handwritten annotations in orange text with arrows point to specific parts of the equation: 'learning rate' points to n ; 'difference b/w target & output' points to $(target_i - output_i)$; 'old weight' points to w_i ; 'new weight' points to w'_i ; and 'Relevant dimension of data sample' points to x_i .

$$w'_i = w_i + n(target_i - output_i)x_i$$

learning rate

difference b/w target & output

old weight

new weight

Relevant dimension of data sample

We usually aim to provide a visual example before diving into the maths. But their truly is **no way to discuss a perceptron without maths**, so let's break down these components.

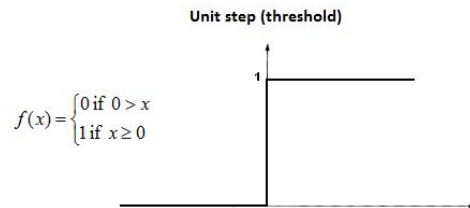
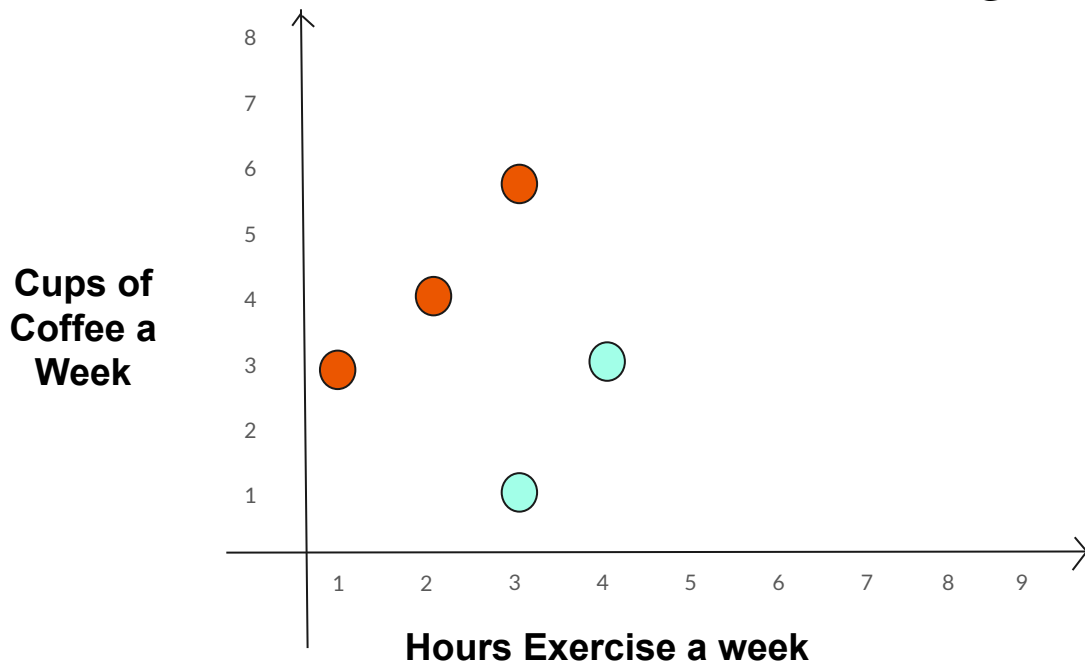
1 = Insomnia
0 = Good nights rest

What happens as we
increase
coffee/decrease
exercise?

Hours of exercise a week	Cups of coffee a week	Insomnia
1	3	1
2	4	1
3	6	1
3	1	0
4	3	0

Let's work through a simple example using a **dataset**. We'll take a look at **coffee consumed & hours of exercise**, as it relates to **insomnia**.

$$\hat{y} = w_0 + w_1 x_0 + w_2 x_1$$



● = Insomnia

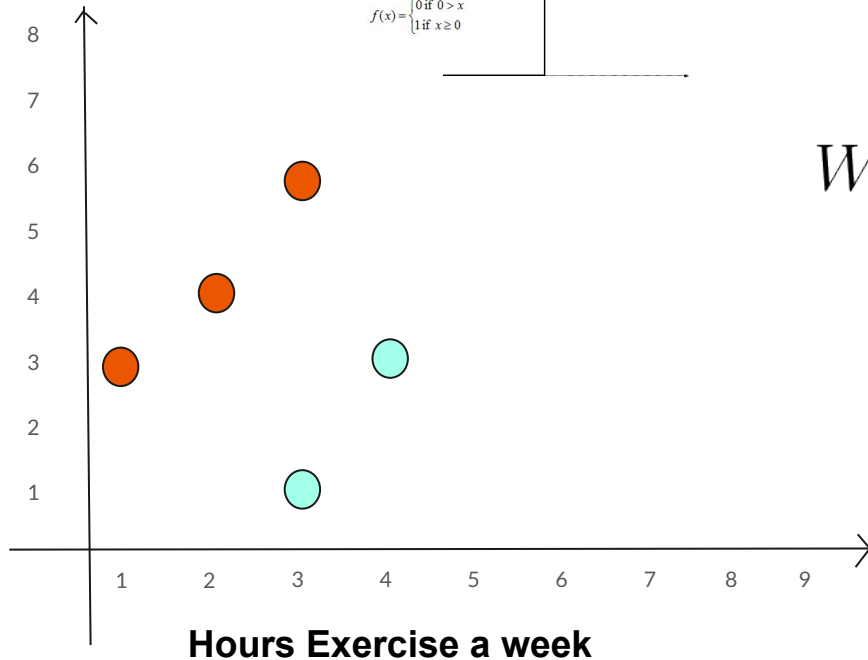
● = No Insomnia

We plot these data points on a scatter-plot. Would you state that these are linearly separable?

● = Insomnia

● = No Insomnia

Cups of
Coffee a
Week



$$\hat{y} = w_0 + w_1x_0 + w_2x_1$$

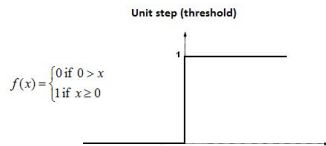
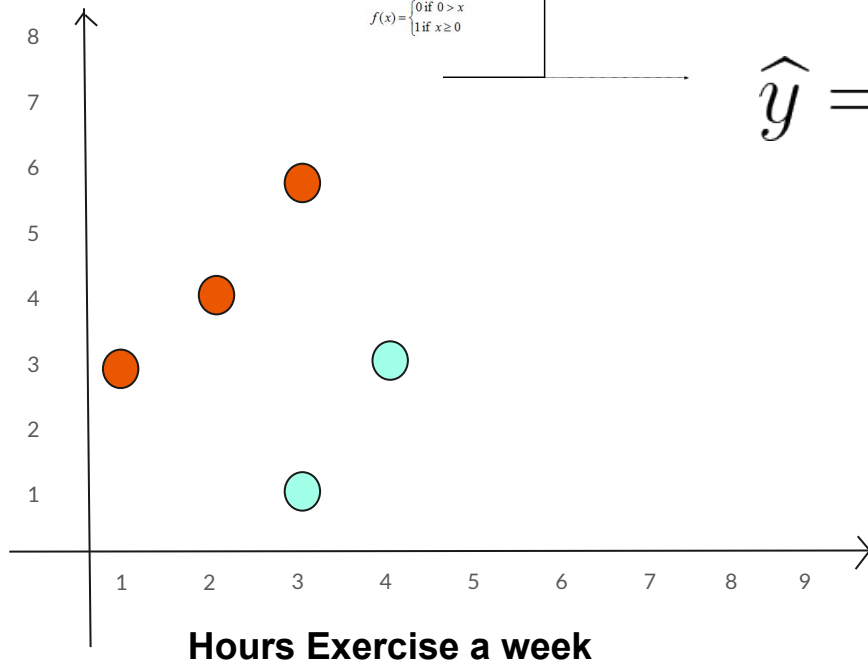
$$W = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

To start, we select an “incorrect” vector of weights such as $(0, 0, 0)$.
According to these weights, what will our weight function look like?

● = Insomnia

● = No Insomnia

Cups of
Coffee a
Week



$$\hat{y} = w_0 + w_1x_0 + w_2x_1$$

$$\hat{y} = 0 + 0 * x_0 + 0 * x_1$$

$$y_1 = 0 + 0(1) + 0(3)$$

$$y_2 = 0 + 0(2) + 0(4)$$

$$y_3 = 0 + 0(3) + 0(6)$$

$$y_4 = 0 + 0(3) + 0(1)$$

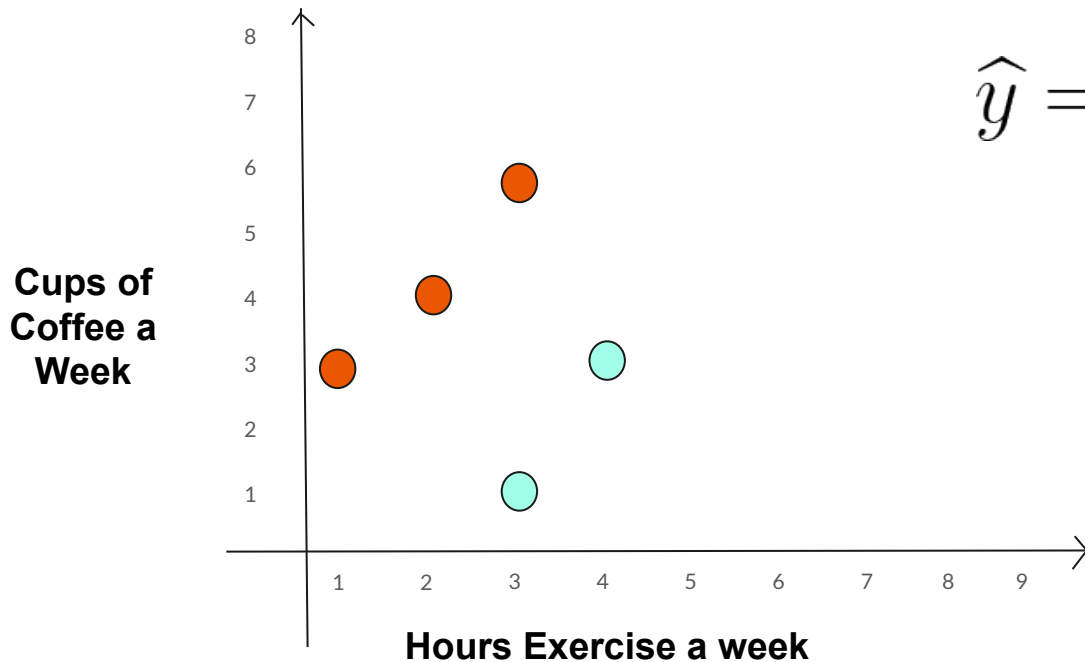
$$y_5 = 0 + 0(4) + 0(3)$$

This will give us an equation where all the coefficients are “0”. Next we will plug in all of our samples into this function and calculate their output.

What will all these samples calculate?

● = Insomnia

● = No Insomnia



$$\hat{y} = w_0 + w_1x_0 + w_2x_1$$

$$\hat{y} = 0 + 0 * x_0 + 0 * x_1$$

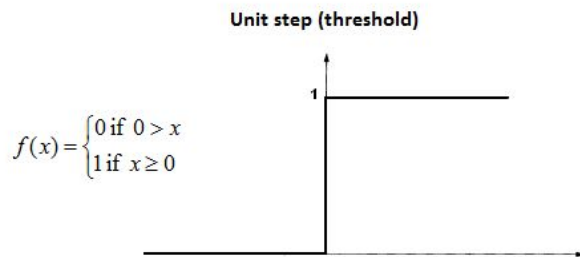
$$y_1 = 0 + 0(1) + 0(3) = 0$$

$$y_2 = 0 + 0(2) + 0(4) = 0$$

$$y_3 = 0 + 0(3) + 0(6) = 0$$

$$y_4 = 0 + 0(3) + 0(1) = 0$$

$$y_5 = 0 + 0(4) + 0(3) = 0$$



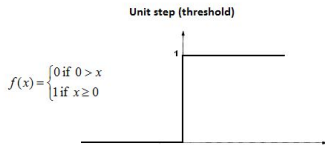
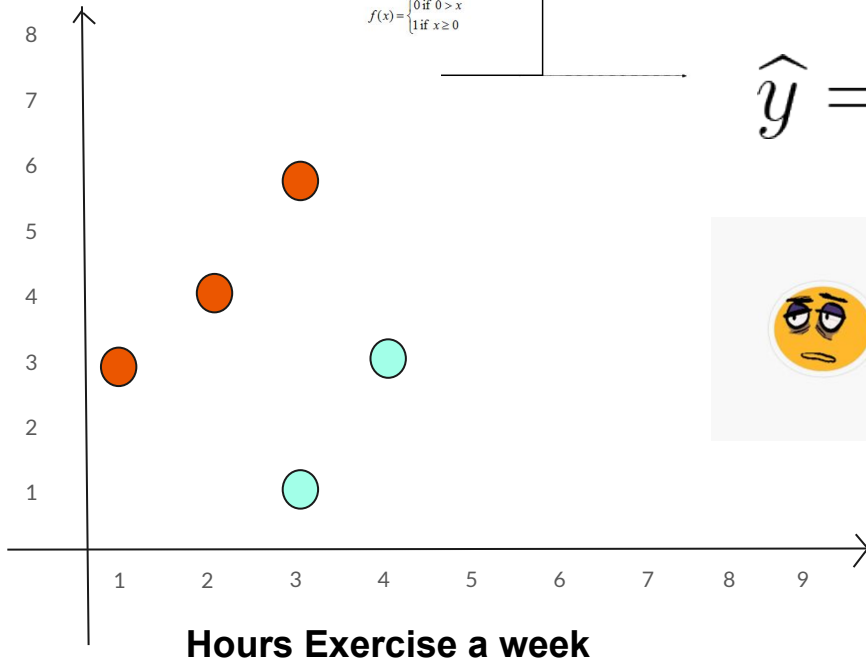
All of these outputs compute to “0.”

According to our step-function, what will all of our samples be classified as?

● = Insomnia

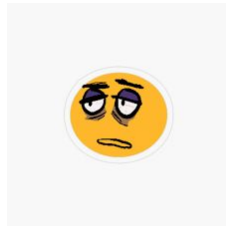
● = No Insomnia

Cups of
Coffee a
Week



$$\hat{y} = w_0 + w_1x_0 + w_2x_1$$

$$\hat{y} = 0 + 0 * x_0 + 0 * x_1$$



$y_1 = 0 + 0(1) + 0(3) = 0 \longrightarrow$ Insomnia (1)
 $y_2 = 0 + 0(2) + 0(4) = 0 \longrightarrow$ Insomnia (1)
 $y_3 = 0 + 0(3) + 0(6) = 0 \longrightarrow$ Insomnia (1)
 $y_4 = 0 + 0(3) + 0(1) = 0 \longrightarrow$ Insomnia (1)
 $y_5 = 0 + 0(4) + 0(3) = 0 \longrightarrow$ Insomnia (1)

Insomnia	Predicted Insomnia
1	1
1	1
1	1
0	1
0	1

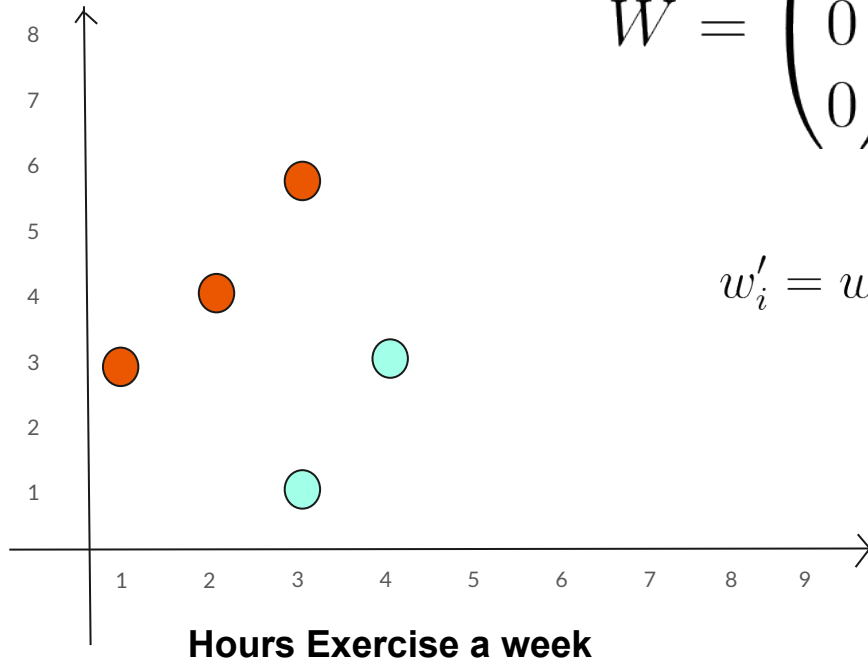
Our heaviside function tells us that this does not meet the threshold for 'activation' therefore **all of these samples get classified as 'no insomnia.'**

Obviously this is an incorrect model. Let's take note of these misclassifications...

● = Insomnia

● = No Insomnia

Cups of
Coffee a
Week



$$W = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Insomnia	Predicted Insomnia
1	1
1	1
1	1
0	1
0	1

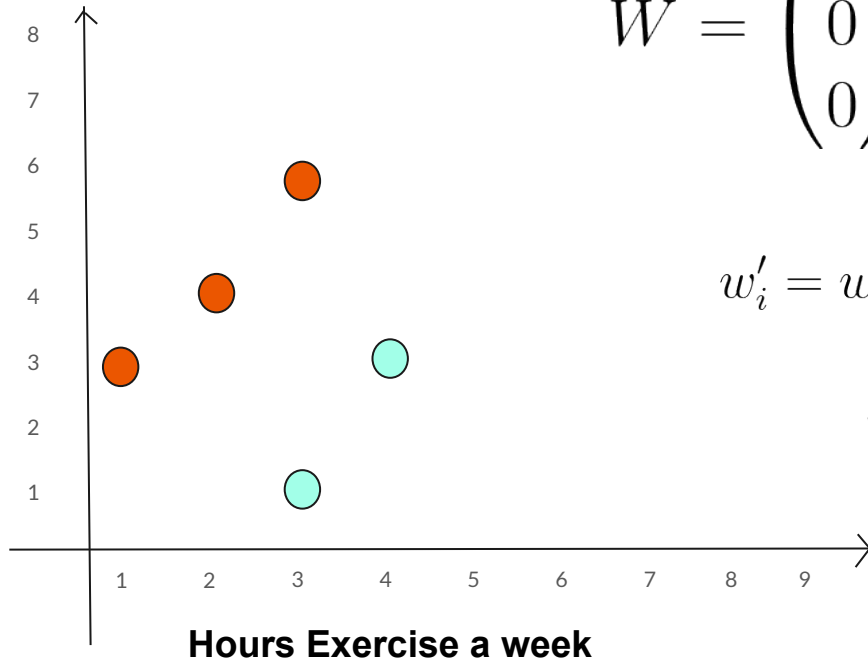
$$w'_i = w_i + n(target_i - output_i)x_i$$

We can update our weights via gradient descent by following the gradient of these weights! Let's do one round of this to see how our weights update.

● = Insomnia

● = No Insomnia

Cups of
Coffee a
Week



$$W = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Exercise	Coffee	Insomnia	Predicted Insomnia
1	3	1	1
2	4	1	1
3	6	1	1
3	1	0	1
4	3	0	1

$$w'_i = w_i + n(target_i - output_i)x_i$$

$w1' =$

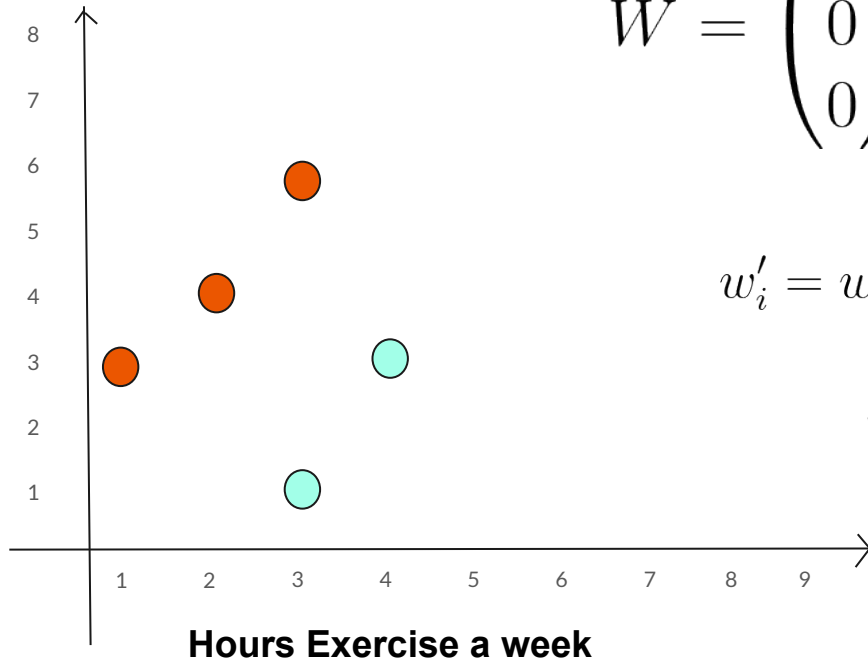
Let's start with our $w1$ weight. Can you fill in the variables of this gradient descent algorithm for the first sample?

We make use of the **previous weight calculation**, the **errors we've made**, the **value of the data-points themselves**, and finally the **learning rate** (we'll use 0.1 in this example)

● = Insomnia

● = No Insomnia

Cups of
Coffee a
Week



$$W = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Exercise	Coffee	Insomnia	Predicted Insomnia
1	3	1	1
2	4	1	1
3	6	1	1
3	1	0	1
4	3	0	1

$$w'_i = w_i + n(target_i - output_i)x_i$$

$$w1' = 0 + (0.1)(1 - 1)(1)$$

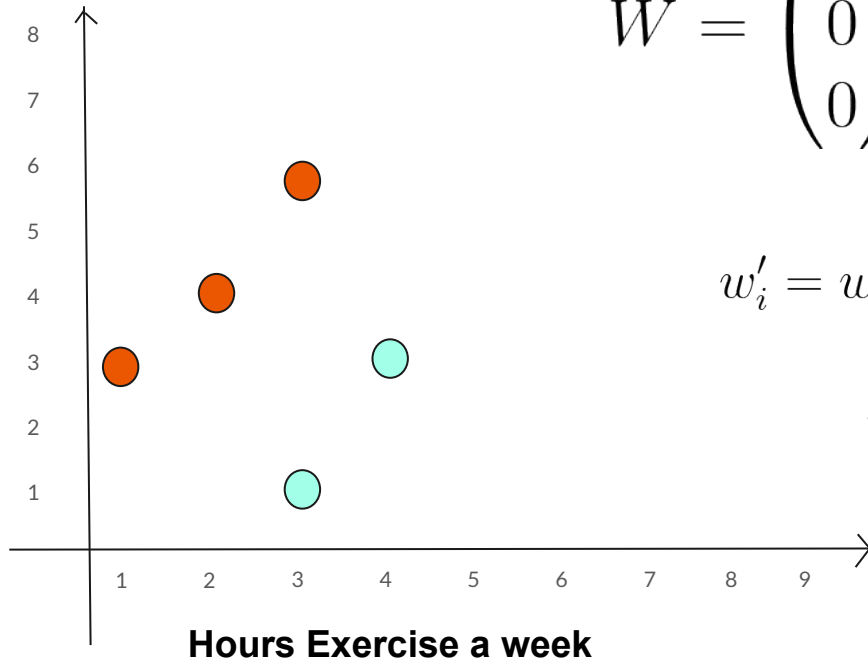
Since we look at “**weight 1**”, we only consider “**predictor 1**” (exercise) of our first sample

What will be the value of this new weight?

● = Insomnia

● = No Insomnia

Cups of
Coffee a
Week



$$W = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Exercise	Coffee	Insomnia	Predicted Insomnia
1	3	1	1
2	4	1	1
3	6	1	1
3	1	0	1
4	3	0	1

$$w'_i = w_i + n(target_i - output_i)x_i$$

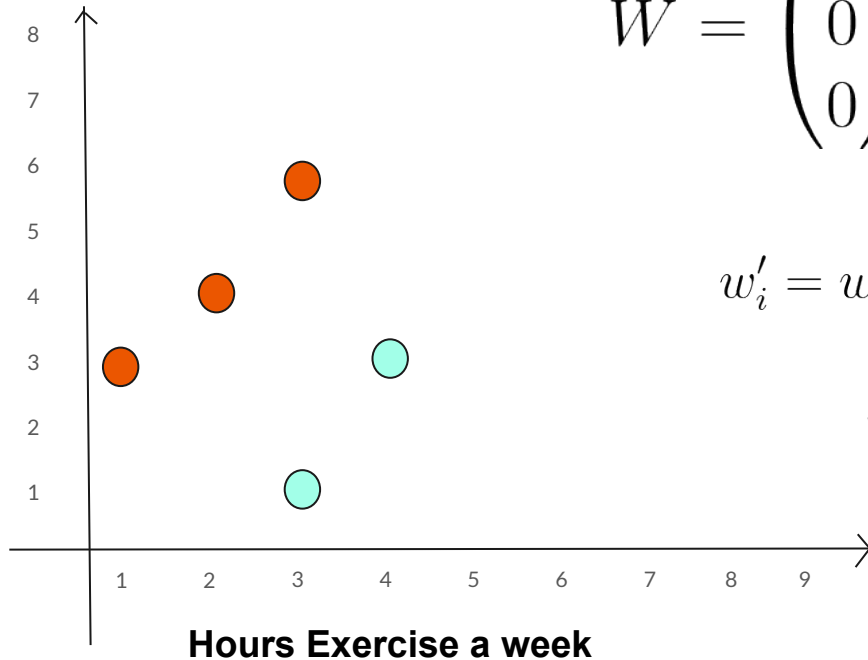
$$w1' = 0 + (0.1)(1 - 1)(1) = 0.0$$

We get a value of 0.0! However we are not done yet, we need to update our weights based on ALL SAMPLES (full-batch gradient descent)

● = Insomnia

● = No Insomnia

Cups of
Coffee a
Week



$$W = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Exercise	Coffee	Insomnia	Predicted Insomnia
1	3	1	1
2	4	1	1
3	6	1	1
3	1	0	1
4	3	0	1

$$w'_i = w_i + n(\text{target}_i - \text{output}_i)x_i$$

$$w1' = 0 + (0.1)(1 - 1)(1) = 0.0$$

$$w1' = 0.0 + (0.1)(1 - 1)(2)$$

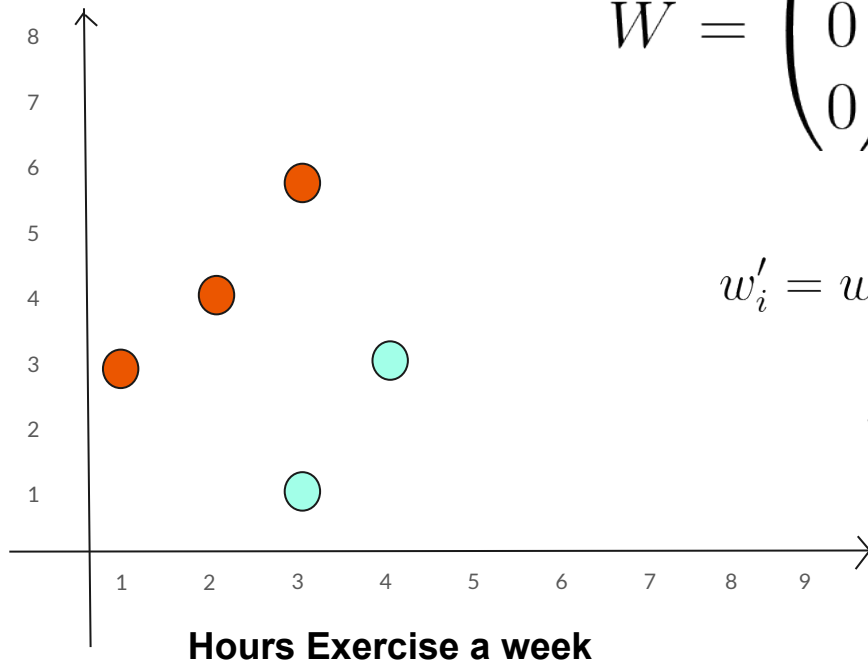
You might be thinking, “so do we just reuse ‘0’ as the ‘old weight’ in the next sample calculation?”

NO! You actually use the previous weight you calculated!!! Can anyone calculate this new value?

● = Insomnia

● = No Insomnia

Cups of
Coffee a
Week



$$W = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Exercise	Coffee	Insomnia	Predicted Insomnia
1	3	1	1
2	4	1	1
3	6	1	1
3	1	0	1
4	3	0	1

$$w'_i = w_i + n(\text{target}_i - \text{output}_i)x_i$$

$$w1' = 0 + (0.1)(1 - 1)(1) = 0.0$$

$$w1' = 0.0 + (0.1)(1 - 1)(2) = 0.0$$

$$w1' = 0.0 + (0.1)(1 - 1)(3) = 0.0$$

$$w1' = 0.0 + (0.1)(0 - 1)(3) = -0.3$$

$$w1' = -0.3 + (0.1)(0 - 1)(4) = -0.7$$

And then the cycle continues until we've observed all samples! Once we finish, we get our new weight.

What do you notice when we do (0 - 0)?

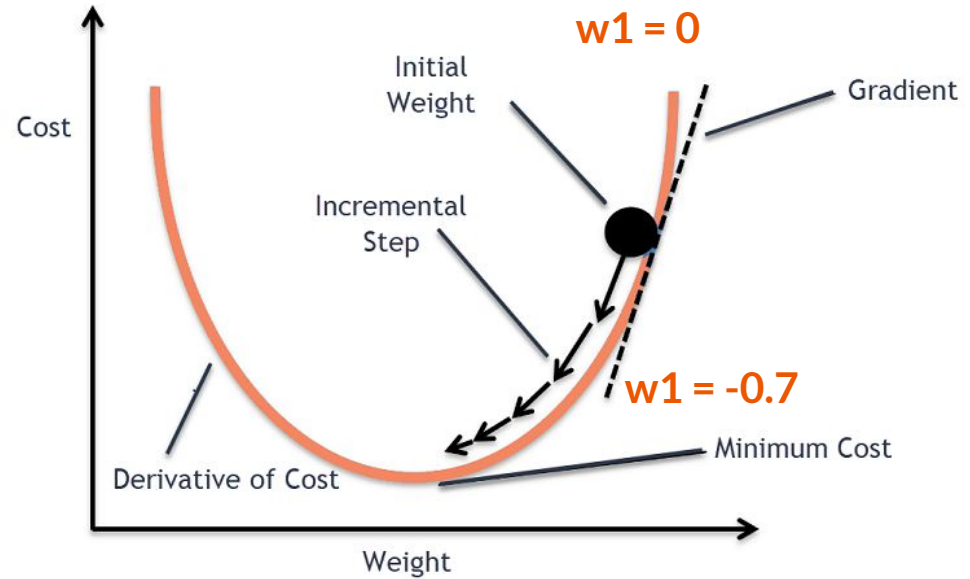
$$w1' = 0 + (0.1)(1 - 1)(1) = 0.0$$

$$w1' = 0.0 + (0.1)(1 - 1)(2) = 0.0$$

$$w1' = 0.0 + (0.1)(1 - 1)(3) = 0.0$$

$$w1' = 0.0 + (0.1)(0 - 1)(3) = -0.3$$

$$w1' = -0.3 + (0.1)(0 - 1)(4) = -0.7$$



In more visual terms, we just “followed the gradient” of our errors to create a coefficient that is “less wrong.” It’s not perfect, but it’s a step in the right direction! We keep doing this until **we get to a local minimum**.

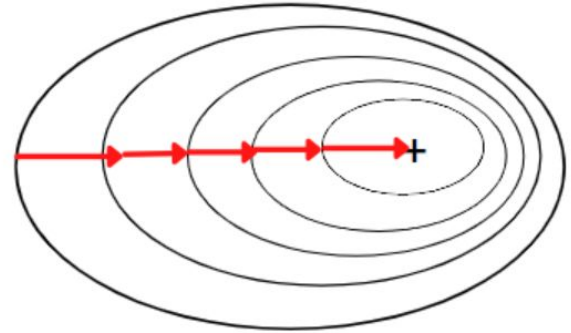
An Aside - Full Batch Gradient Descent

What we just did is called **FULL BATCH GRADIENT DESCENT**.

This involves going through all data-points and updating our weights.

Remember, as technologists we want to be 'cost-effective', and full-batch gradient descent is the **opposite of cost-effective when you have a bajillion rows**.

Batch Gradient Descent



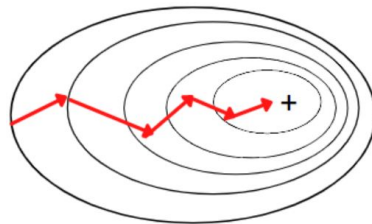
An Aside - Other Options for Gradient Descent

Instead, we utilize either **stochastic** or **mini-batch gradient descent**.

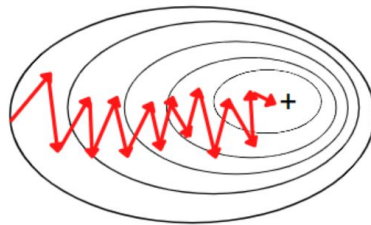
Both are **more cost-effective** and lead to approximately **the same optimum**.

We will explore these techniques later.

Mini-Batch Gradient Descent



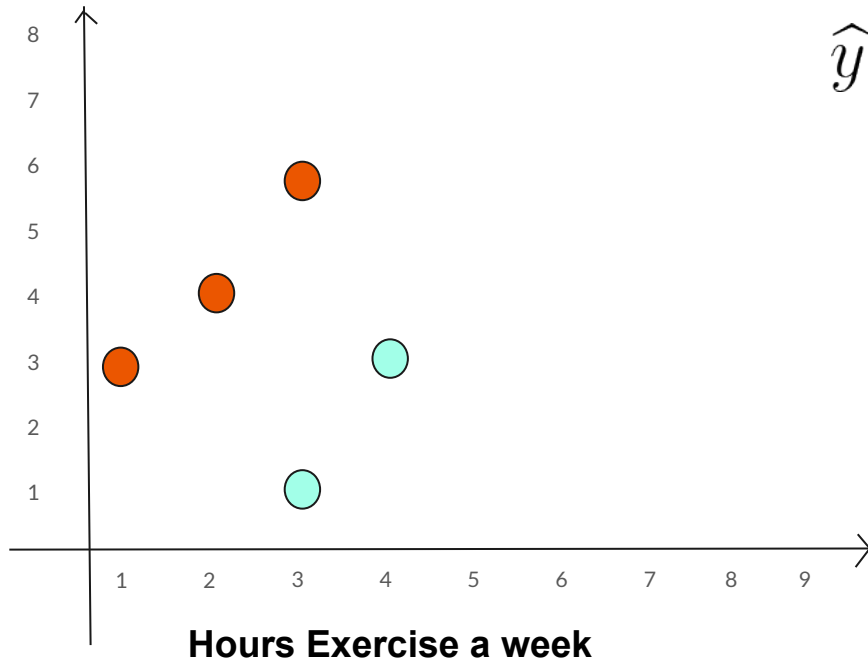
Stochastic Gradient Descent



● = Insomnia

● = No Insomnia

**Cups of
Coffee a
Week**



$$\hat{y} = w_0 + w_1x_0 + w_2x_1$$

$$\hat{y} = 0.3 - 0.7x_1 + 1.3x_2$$

$$y_1 = 0.3 - 0.7(1) + 1.3(3) = 3.5 \rightarrow 1$$

$$y_2 = 0.3 - 0.7(2) + 1.3(4) = 4.1 \rightarrow 1$$

$$y_3 = 0.3 - 0.7(3) + 1.3(6) = 6 \rightarrow 1$$

$$y_4 = 0.3 - 0.7(3) + 1.3(1) = -0.5 \rightarrow 0$$

$$y_5 = 0.3 - 0.7(4) + 1.3(3) = 1.4 \rightarrow 1$$

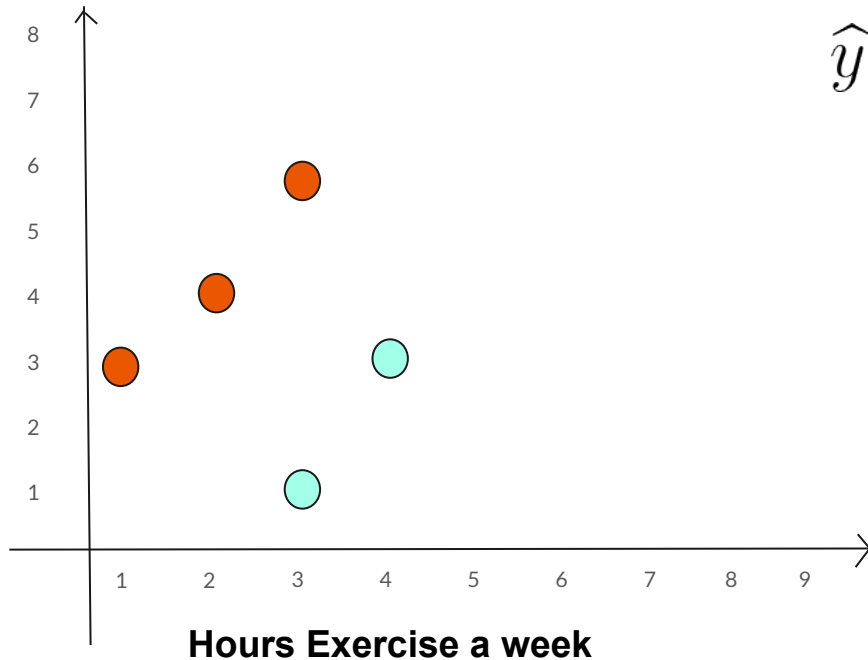
Eventually after applying these updates for each weight (**w0**, **w1**, **w2**) , we are left with (0.3, -0.7, 1.3)

Is this a better classifier???

● = Insomnia

● = No Insomnia

Cups of
Coffee a
Week



$$\hat{y} = w_0 + w_1x_0 + w_2x_1$$

$$\hat{y} = 0.3 - 0.7x_1 + 1.3x_2$$

$$y_1 = 0.3 - 0.7(1) + 1.3(3) = 3.5 \rightarrow 1$$

$$y_2 = 0.3 - 0.7(2) + 1.3(4) = 4.1 \rightarrow 1$$

$$y_3 = 0.3 - 0.7(3) + 1.3(6) = 6 \rightarrow 1$$

$$y_4 = 0.3 - 0.7(3) + 1.3(1) = -0.5 \rightarrow 0$$

$$y_5 = 0.3 - 0.7(4) + 1.3(3) = 1.4 \rightarrow 1$$

Yes! It seems that we get some nuance to our predictions. Notice that our 4th sample correctly gets predicted as “no insomnia”!

● = Insomnia

● = No Insomnia

Cups of
Coffee a
Week



$$\hat{y} = w_0 + w_1x_0 + w_2x_1$$

$$\hat{y} = 0.3 - 0.7x_1 + 1.3x_2$$

$$y_1 = 0.3 - 0.7(1) + 1.3(3) = 3.5 \rightarrow 1$$

$$y_2 = 0.3 - 0.7(2) + 1.3(4) = 4.1 \rightarrow 1$$

$$y_3 = 0.3 - 0.7(3) + 1.3(6) = 6 \rightarrow 1$$

$$y_4 = 0.3 - 0.7(3) + 1.3(1) = -0.5 \rightarrow 0$$

$$y_5 = 0.3 - 0.7(4) + 1.3(3) = 1.4 \rightarrow 1$$

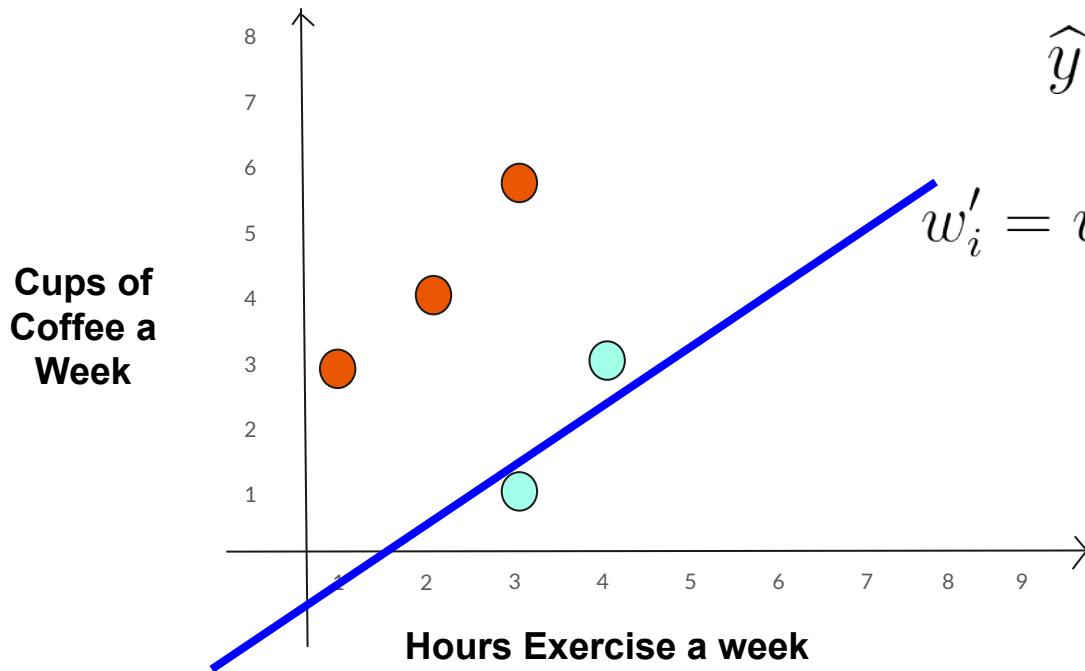
Note this hyperplane is calculated by setting our equation greater than 0

$$0.3 - 0.7x_1 + 1.3x_2 \geq 0$$

By solving the inequality b/w x_1 & x_2 we get the following hyperplane

● = Insomnia

● = No Insomnia



$$\hat{y} = w_0 + w_1x_0 + w_2x_1$$

$$\hat{y} = 0.3 - 0.7x_1 + 1.3x_2$$

$$w'_i = w_i + n(target_i - output_i)x_i$$

Since this hyperplane still makes a few misclassifications, we should allow it to keep training via gradient descent.

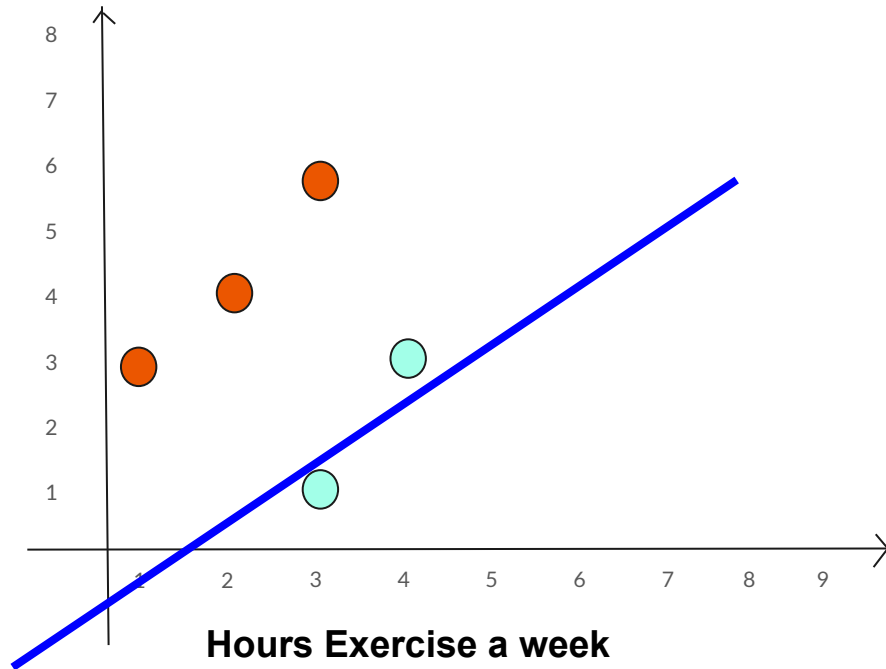
Once it reaches a point where the separating hyperplane makes **marginal updates**, we stop training. We control this through a **hyperparameter** called **epochs** (*how many times do we update our weights*).

● = Insomnia

● = No Insomnia

$$\hat{y} = w_0 + w_1x_0 + w_2x_1$$

Cups of
Coffee a
Week



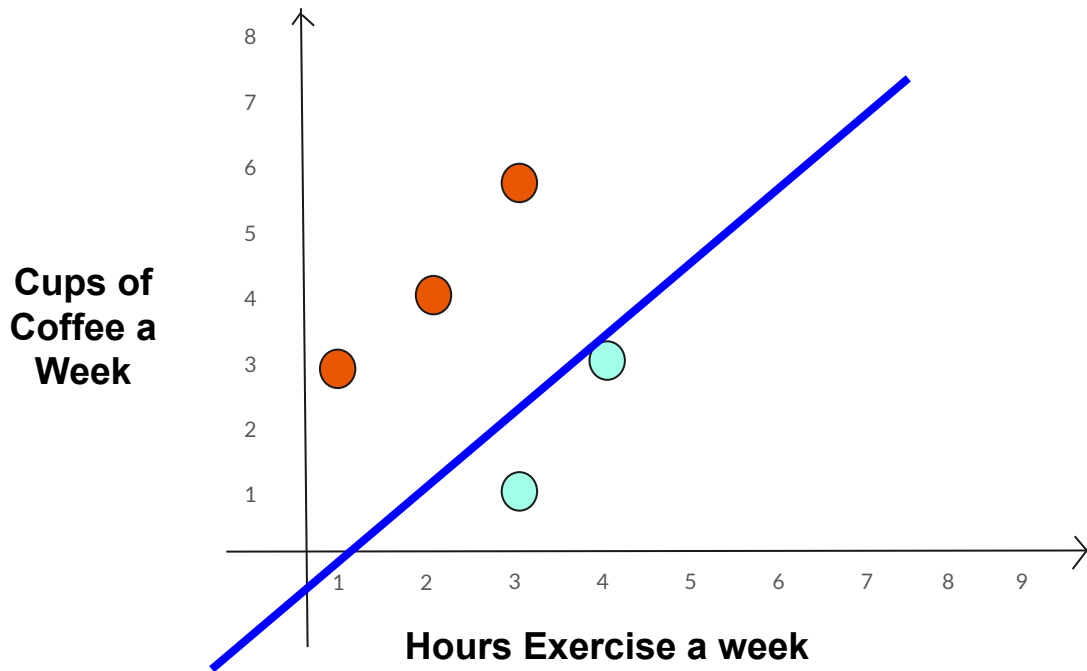
Epoch 1

Our previous (w_0, w_1, w_2)

● = Insomnia

● = No Insomnia

$$\hat{y} = w_0 + w_1x_0 + w_2x_1$$

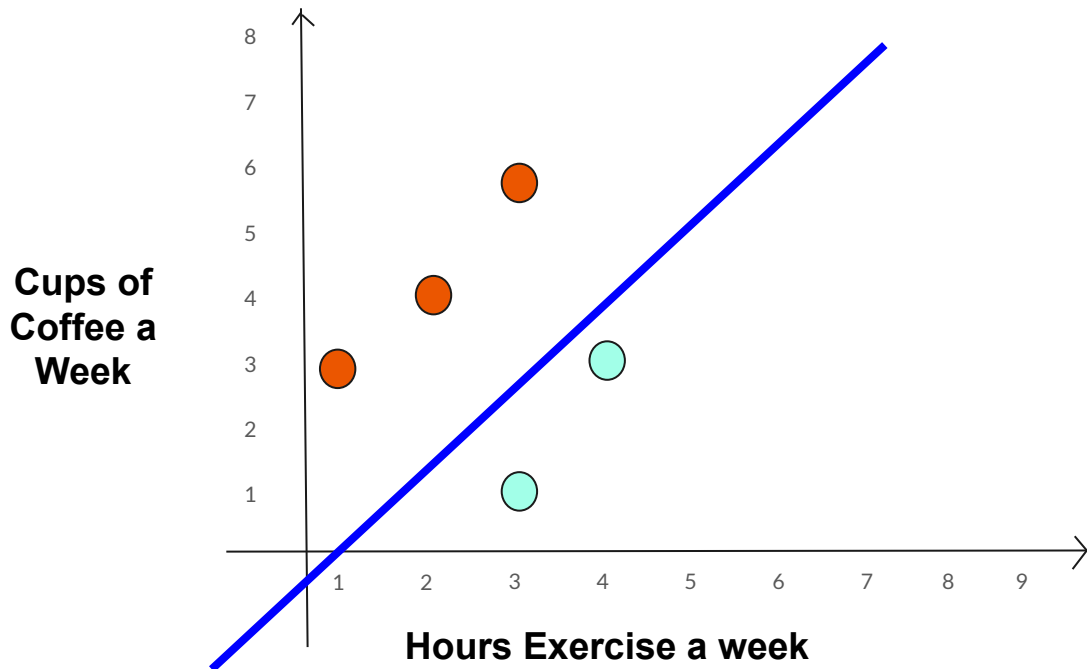


Epoch 2

The next (w_0, w_1, w_2) . Notice that it is now an even better linear separator!

● = Insomnia

● = No Insomnia



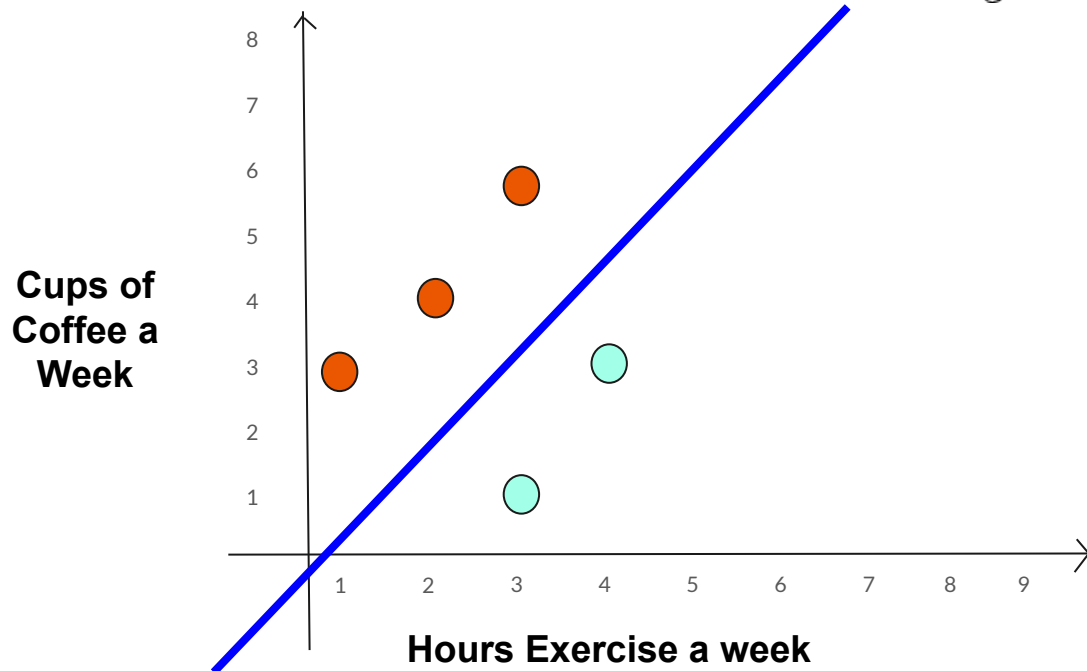
$$\hat{y} = w_0 + w_1x_0 + w_2x_1$$

Epoch 3

The third (w_0, w_1, w_2).

● = Insomnia

● = No Insomnia

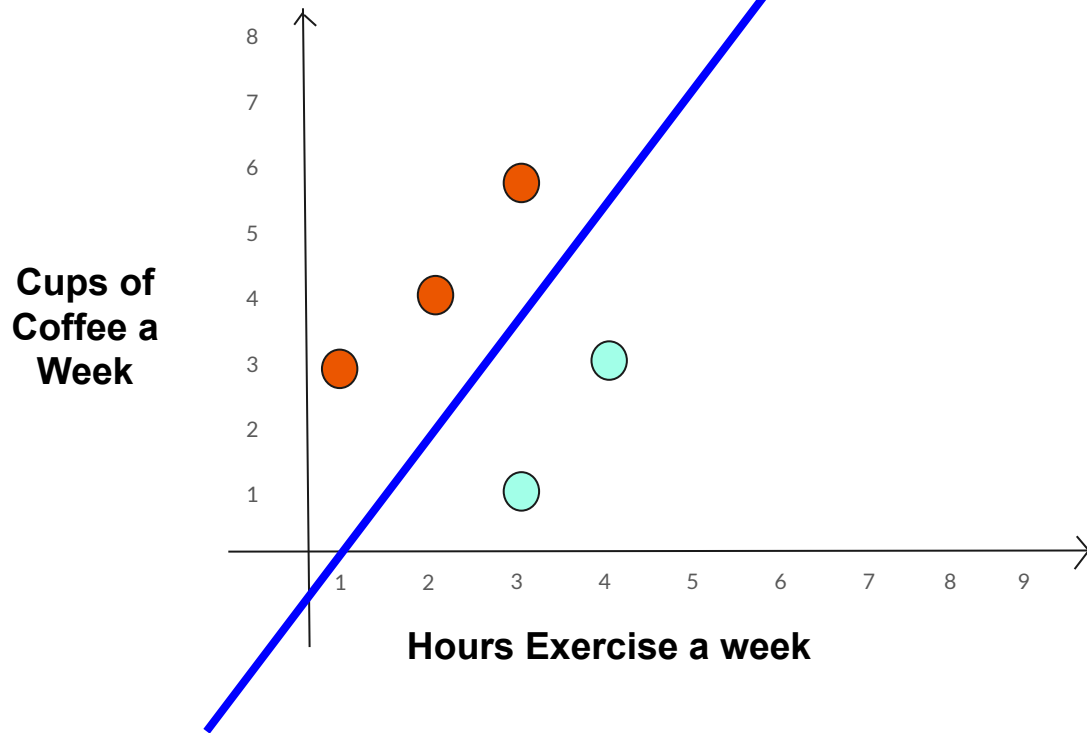


$$\hat{y} = w_0 + w_1x_0 + w_2x_1$$

Epoch 4

The fourth (w_0, w_1, w_2).

- = Insomnia
- = No Insomnia



$$\hat{y} = w_0 + w_1x_0 + w_2x_1$$

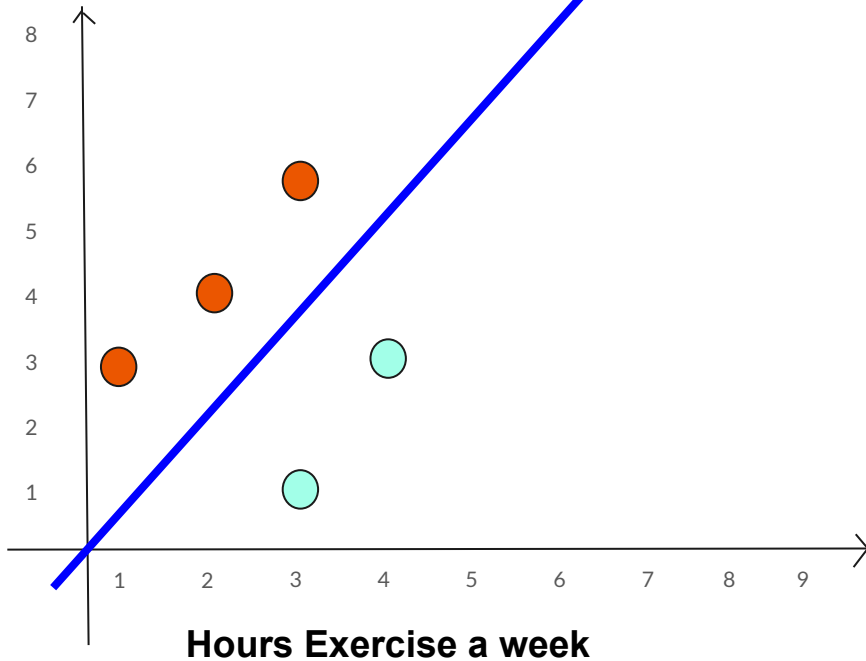
Epoch 5

The fifth (w_0, w_1, w_2). Notice that now we are making marginal improvements to our weights and we're basically **plateauing out**.

● = Insomnia

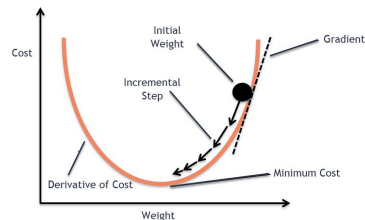
● = No Insomnia

Cups of
Coffee a
Week



$$\hat{y} = w_0 + w_1x_0 + w_2x_1$$

Epoch 6



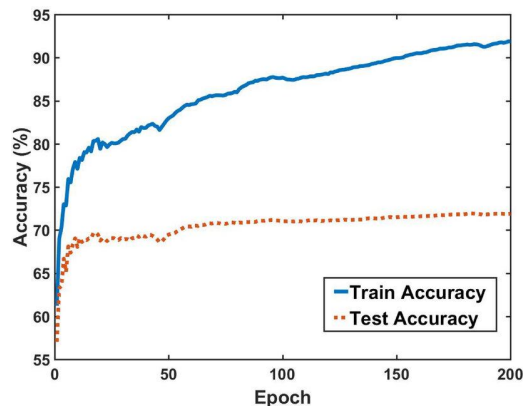
Gradient descent figures it out eventually.

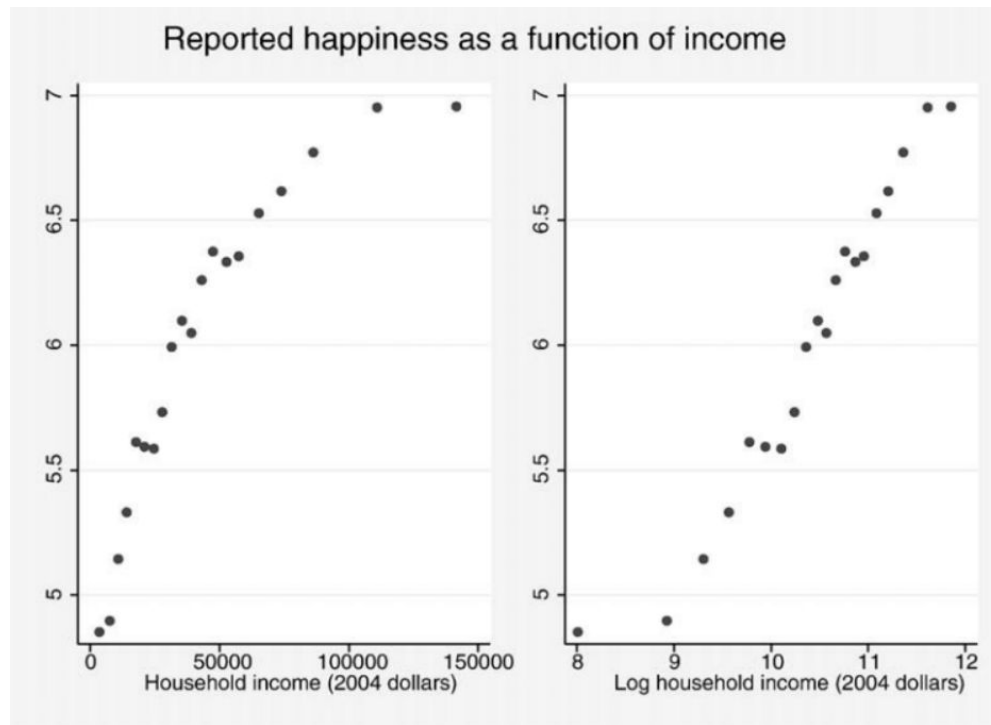
Hyperparameter - Epochs

Epoch count is a hyperparameter that you can control. However we will see that in **Keras** we can implement a *callback* which **halts training** as soon as we have a “good enough” model.

Notice that both **train & test accuracy** both plateau out after **enough** epochs!

This means that there is **some sweet-spot for epochs** (although we might not know where it is).





Think of this in terms of the *marginal gains in happiness across income*. For some individuals, happiness plateaus at a certain income level. Massive gains in income result in negligible gains in happiness. Our perceptrons behave the same way, we find this plateau by halting training at these marginal returns.

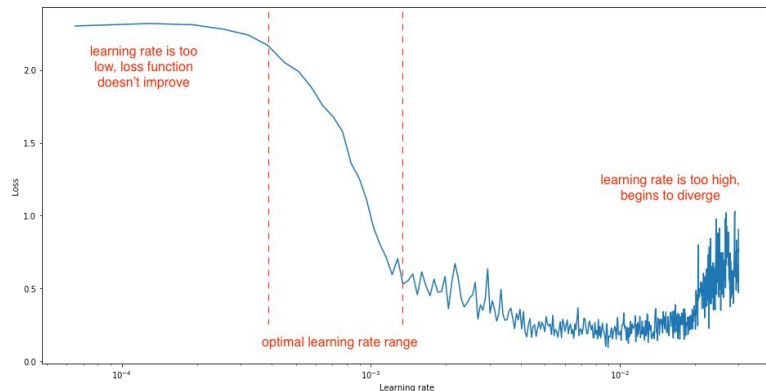
Hyperparameter - Learning Rate

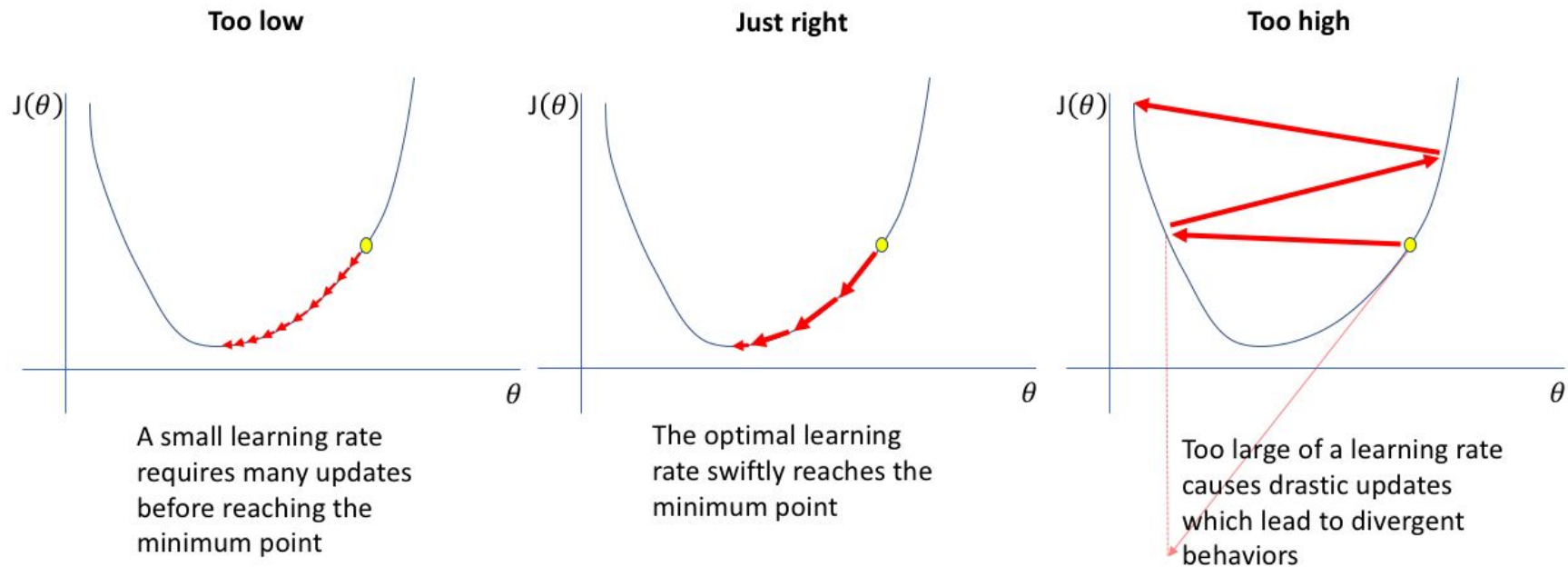
We also have **learning rate**. This controls how **well** your **gradient descent converges!** Notice that this hyperparam does not plateau, instead:

If your learning rate is too small, gradient descent converges slowly

If your learning rate is too large, the gradient descent never converges

0.1 is usually a safe bet, but the **best** hyperparam should be found via CV.



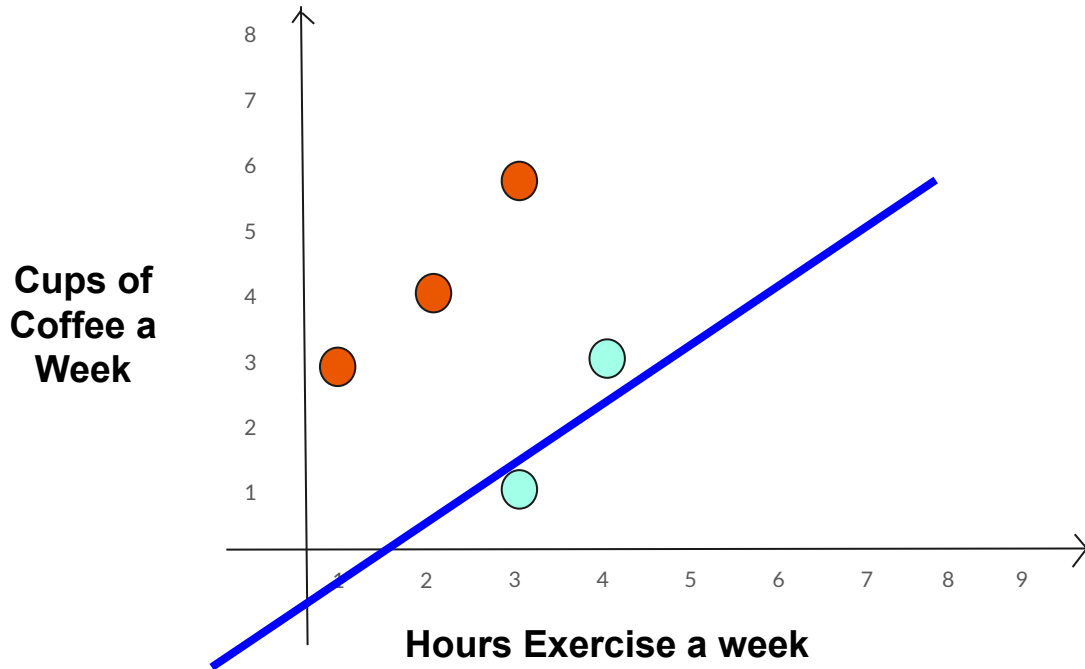


We'll see what this looks like for the separating hyperplane, but essentially, a large learning rate results in instability, while a small learning rate takes baby steps.

● = Insomnia

● = No Insomnia

$$\hat{y} = w_0 + w_1x_0 + w_2x_1$$



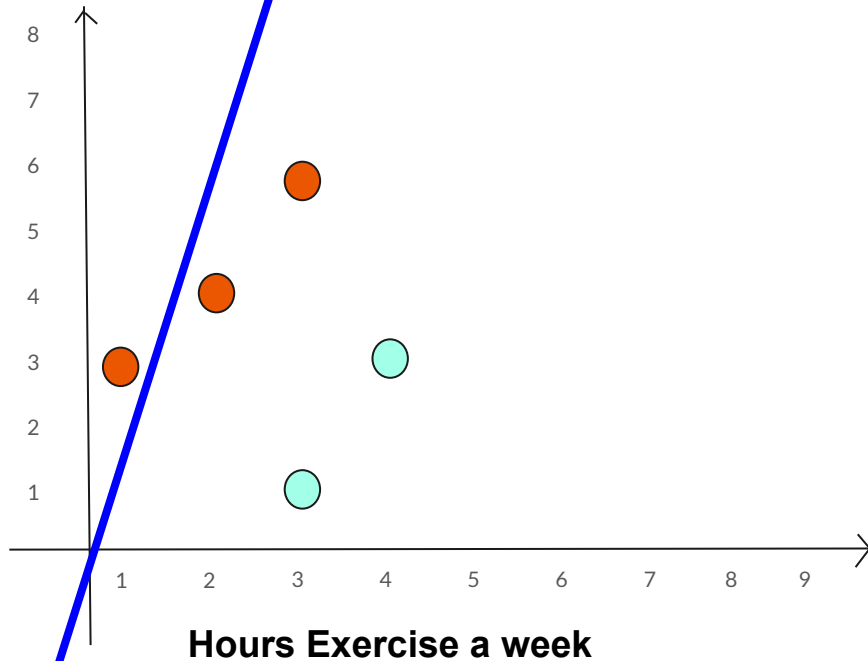
Epoch 1
n = 10

Only for visual demonstration: the behavior of our separating hyperplane with **TOO LARGE A LEARNING RATE**

● = Insomnia

● = No Insomnia

Cups of
Coffee a
Week



$$\hat{y} = w_0 + w_1x_0 + w_2x_1$$

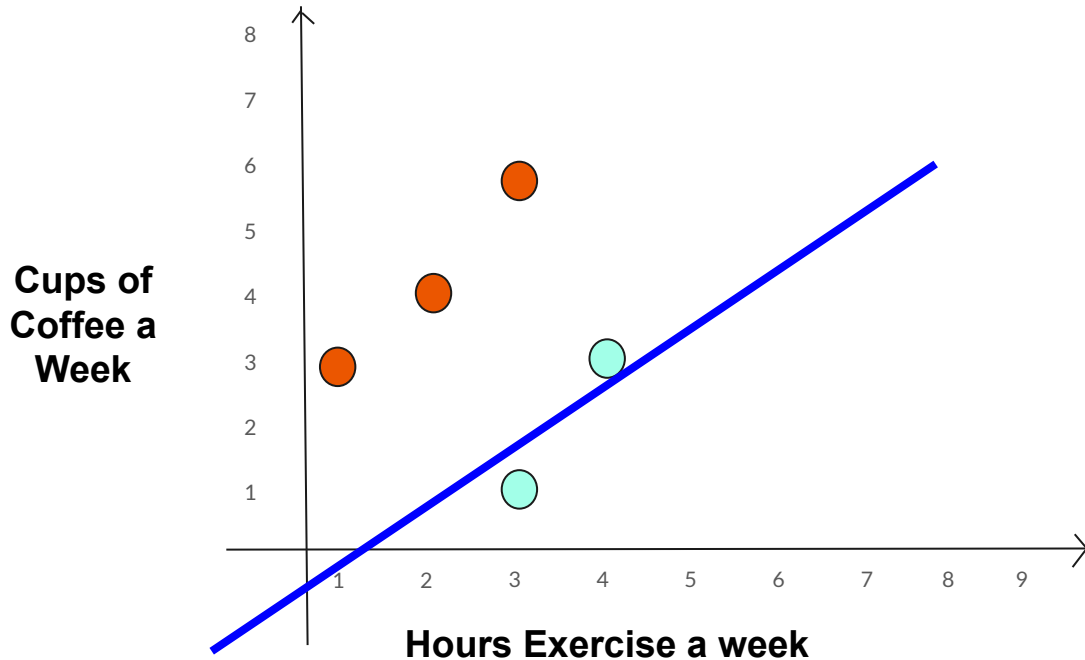
Epoch 2
n = 10

Only for visual demonstration: the behavior of our separating hyperplane with **TOO LARGE A LEARNING RATE**

● = Insomnia

● = No Insomnia

$$\hat{y} = w_0 + w_1x_0 + w_2x_1$$



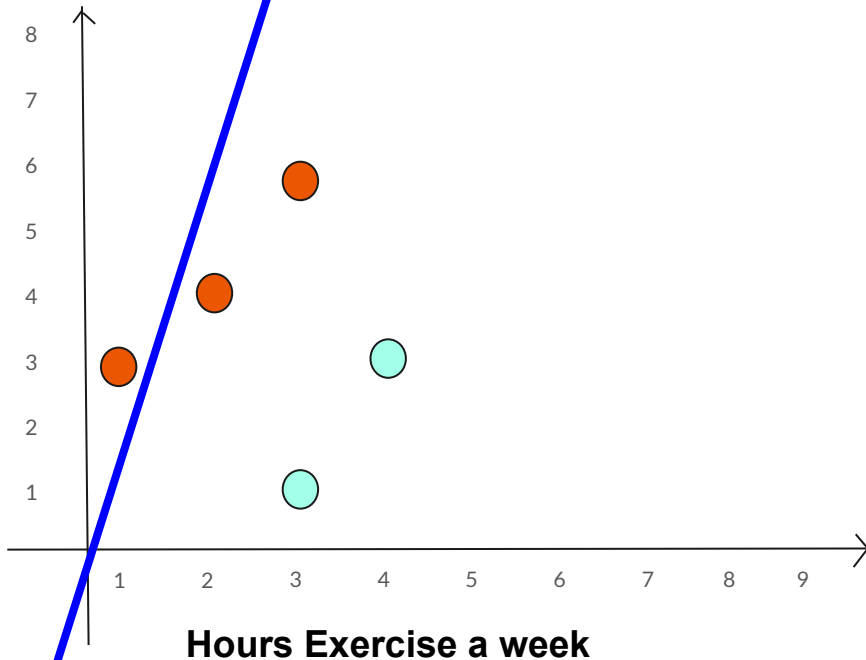
Epoch 3
n = 10

Only for visual demonstration: the behavior of our separating hyperplane with **TOO LARGE A LEARNING RATE**

● = Insomnia

● = No Insomnia

Cups of
Coffee a
Week



$$\hat{y} = w_0 + w_1x_0 + w_2x_1$$

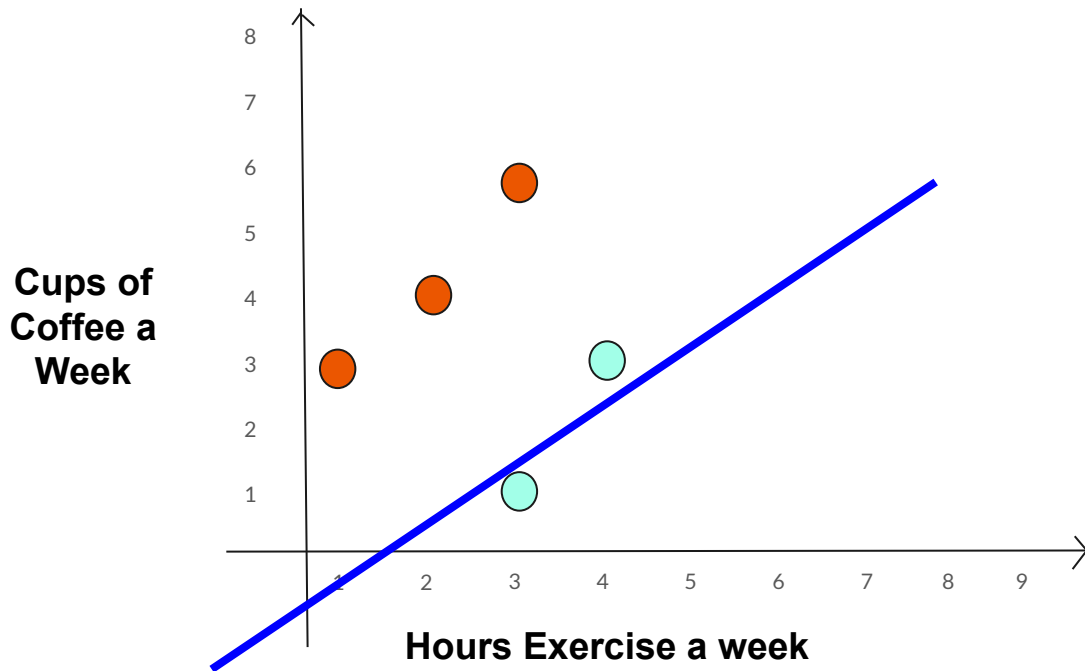
Epoch 4
n = 10

Only for visual demonstration: the behavior of our separating hyperplane with **TOO LARGE A LEARNING RATE**

● = Insomnia

● = No Insomnia

$$\hat{y} = w_0 + w_1x_0 + w_2x_1$$

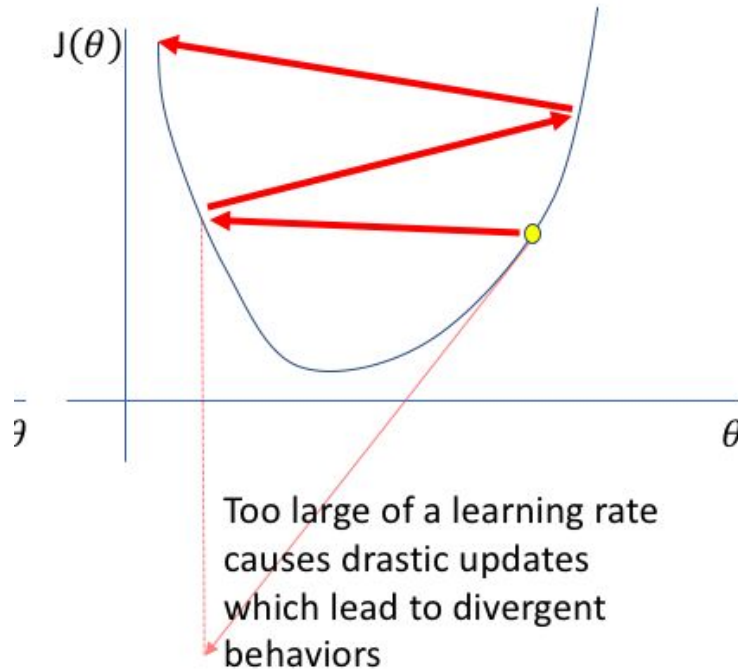


Epoch infinity
n = 10

The learning rate is too **ambitious**. It keeps **overshooting its goal** and **fails each time**.

Only for visual demonstration: the behavior of our separating hyperplane with **TOO LARGE A LEARNING RATE**

Too high

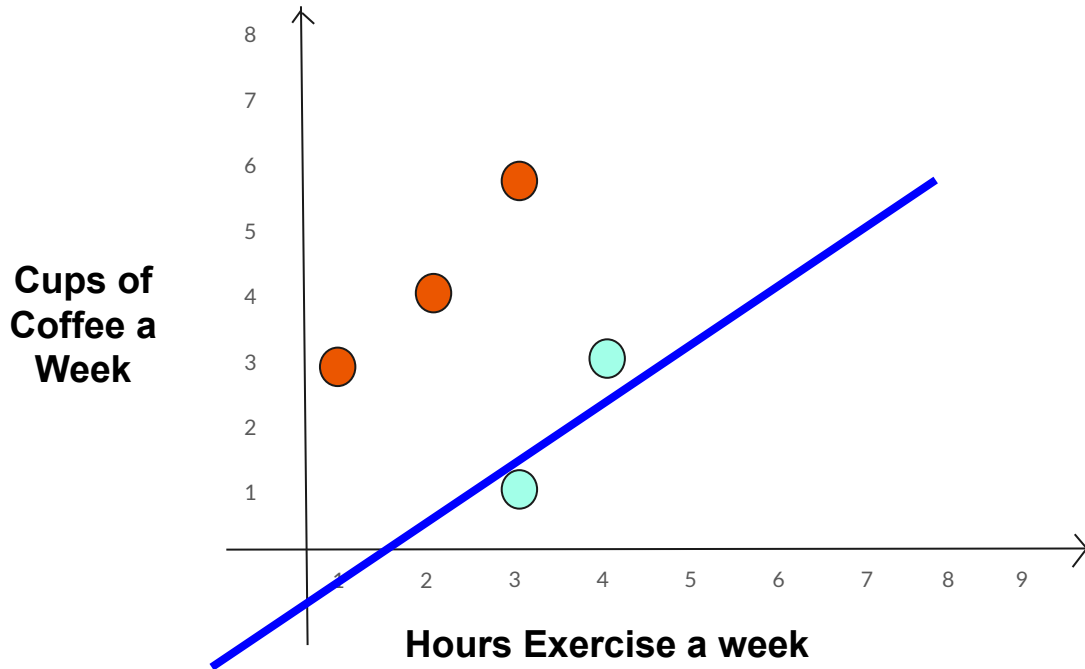


“Too enthusiastic” of a learner

● = Insomnia

● = No Insomnia

$$\hat{y} = w_0 + w_1x_0 + w_2x_1$$



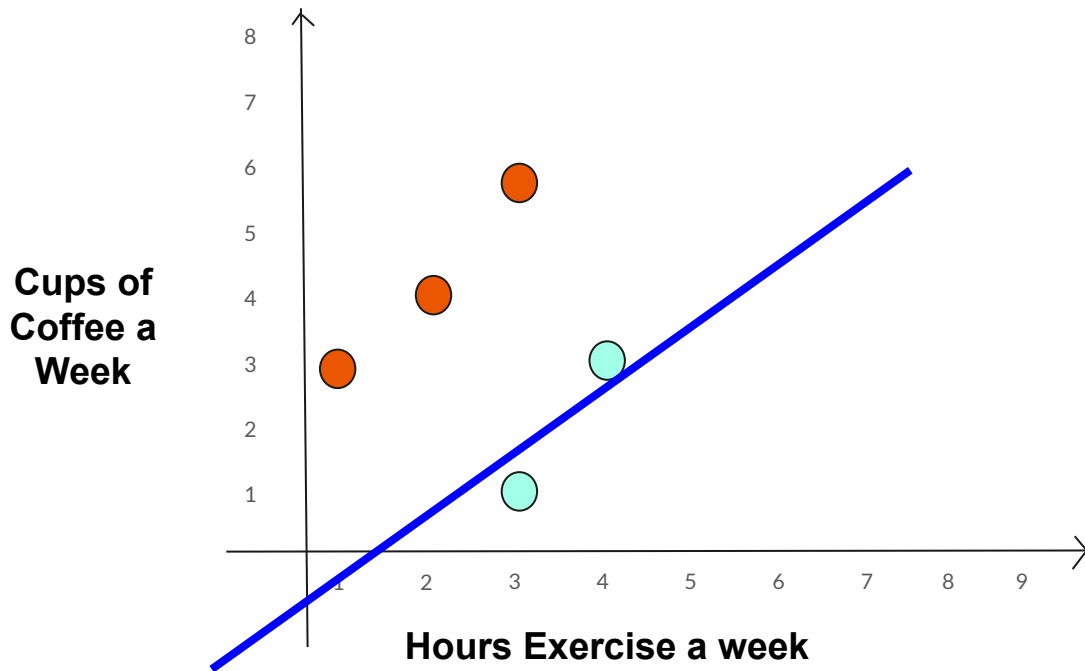
Epoch 1
n = 0.001

Only for visual demonstration: the behavior of our separating hyperplane with **TOO SMALL A LEARNING RATE**

● = Insomnia

● = No Insomnia

$$\hat{y} = w_0 + w_1x_0 + w_2x_1$$



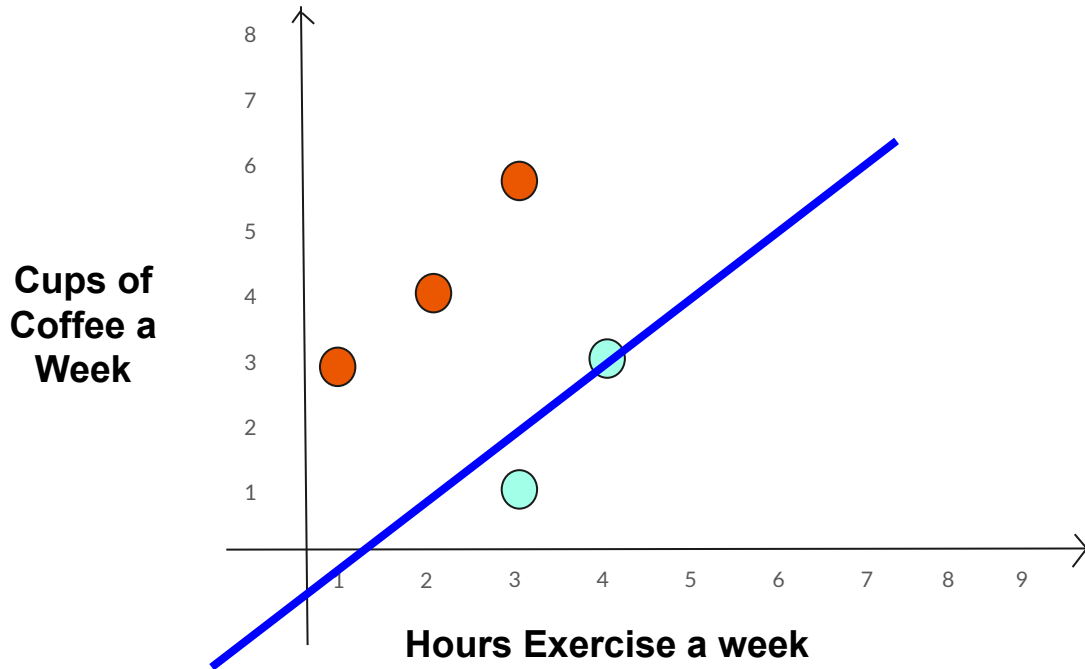
Epoch 2
n = 0.001

Only for visual demonstration: the behavior of our separating hyperplane with **TOO SMALL A LEARNING RATE**

● = Insomnia

● = No Insomnia

$$\hat{y} = w_0 + w_1x_0 + w_2x_1$$

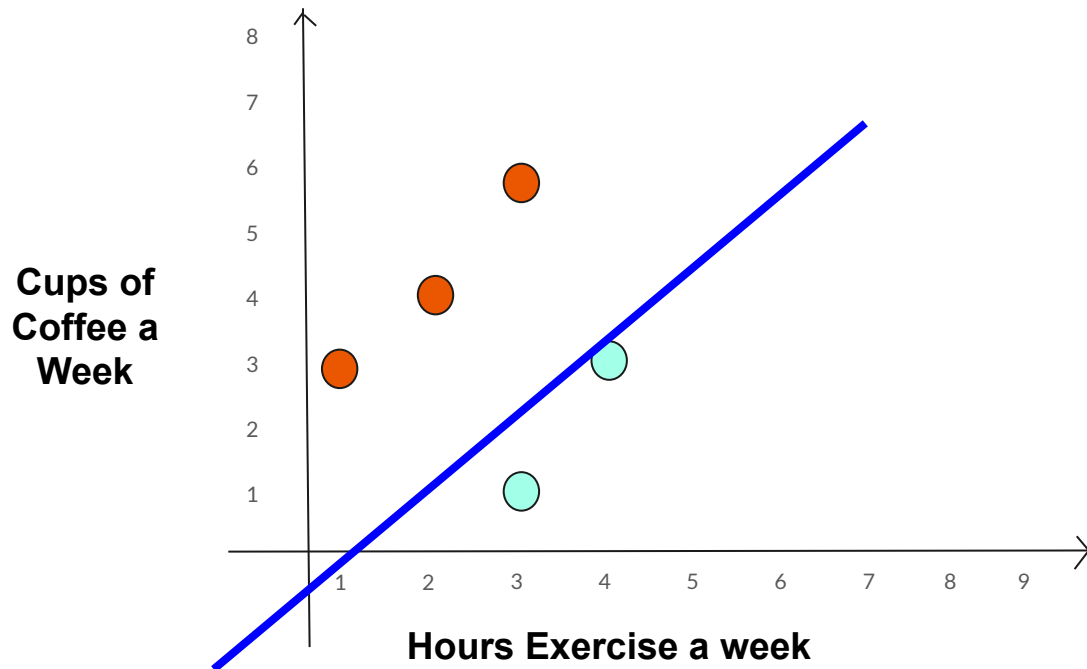


Epoch 3
n = 0.001

Only for visual demonstration: the behavior of our separating hyperplane with **TOO SMALL A LEARNING RATE**

● = Insomnia

● = No Insomnia



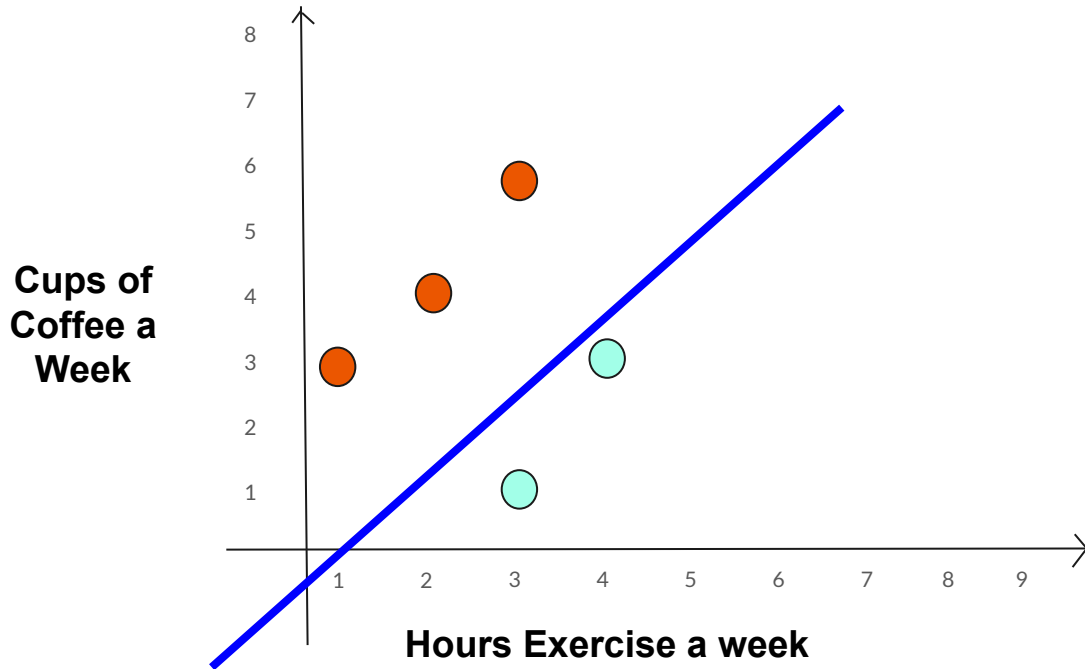
$$\hat{y} = w_0 + w_1x_0 + w_2x_1$$

Epoch 4
n = 0.001

Only for visual demonstration: the behavior of our separating hyperplane with **TOO SMALL A LEARNING RATE**

● = Insomnia

● = No Insomnia



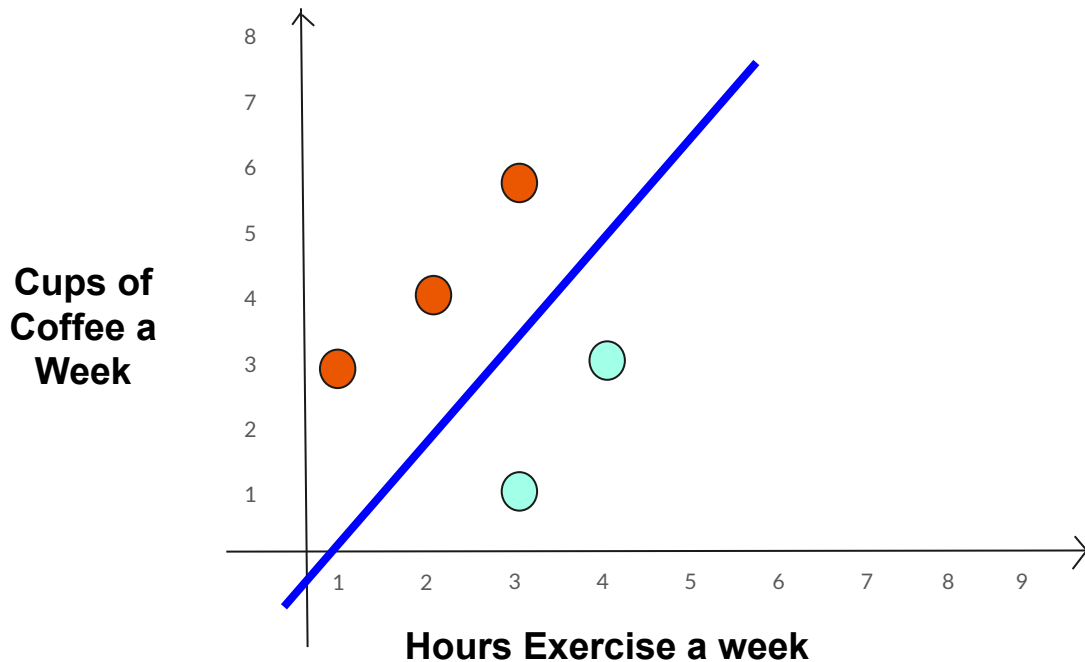
$$\hat{y} = w_0 + w_1x_0 + w_2x_1$$

Epoch 5
n = 0.001

Only for visual demonstration: the behavior of our separating hyperplane with **TOO SMALL A LEARNING RATE**

● = Insomnia

● = No Insomnia



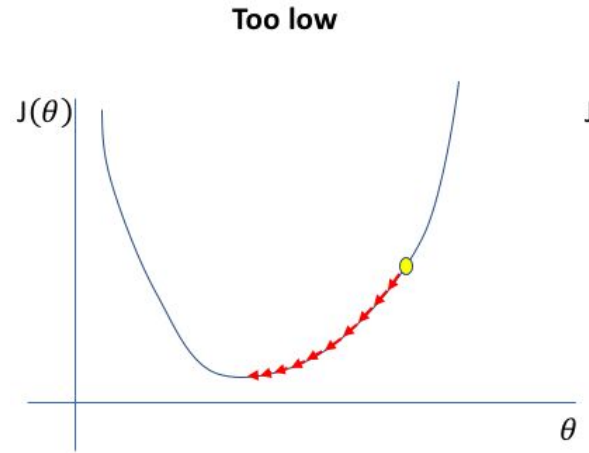
$$\hat{y} = w_0 + w_1x_0 + w_2x_1$$

Epoch ~1000000
n = 0.001

The learning rate is **too shy**. It will eventually find the best hyperplane.

But it will take a long time.

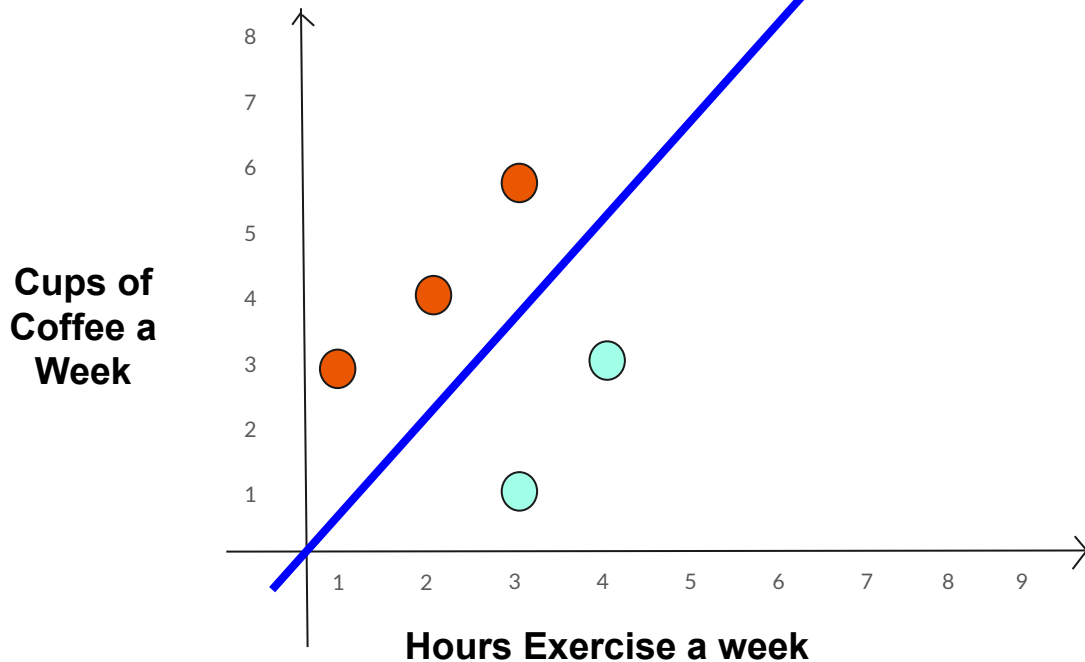
Only for visual demonstration: the behavior of our separating hyperplane with **TOO SMALL A LEARNING RATE**



A small learning rate
requires many updates
before reaching the
minimum point

“Too shy” of a learner

● = Insomnia
● = No Insomnia



$$\hat{y} = w_0 + w_1x_0 + w_2x_1$$

While the “perceptron” is an excellent model for binary classification, what are some drawbacks you can see to this model?

● = Insomnia

● = No Insomnia



$$\hat{y} = w_0 + w_1x_0 + w_2x_1$$

- Assumes “clean & linear” separation of data
- Only works for binary classes
- No ability to model complex patterns
- Prone to overfit

Just like with maximal margin classifiers and logistic regression, **our “single layer perceptron” can only model simple (and unrealistic) datasets.** We need more power aka more neurons (aka a network of neurons). We will explore this on Wednesday

End of Class Announcements



Lab (Due 04/22)



You are a data scientist working at an up & coming music platform startup that just secured its Series B financing.

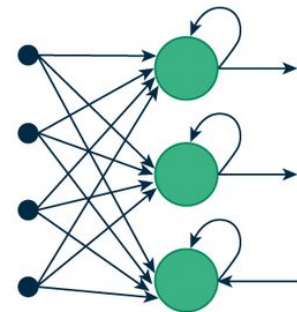
With this infusion of new cash flow, you are tasked with building an **unsupervised recommendation algorithm** that will recommend users new songs based on their previous listening history. As this is a brand new project, you will have to build this project from **scratch**.

Submit a link to your GitHub repository by **4/22**.

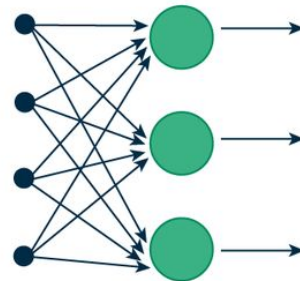
Tomorrow

Neural Networks

- *What is a “recurrent” neural network*
- *How do we build a neural network?*
- *How do we use neural nets to understand sentiment?*



(a) Recurrent Neural Network



(b) Feed-Forward Neural Network

How can we re-organize this structure for better predictions?