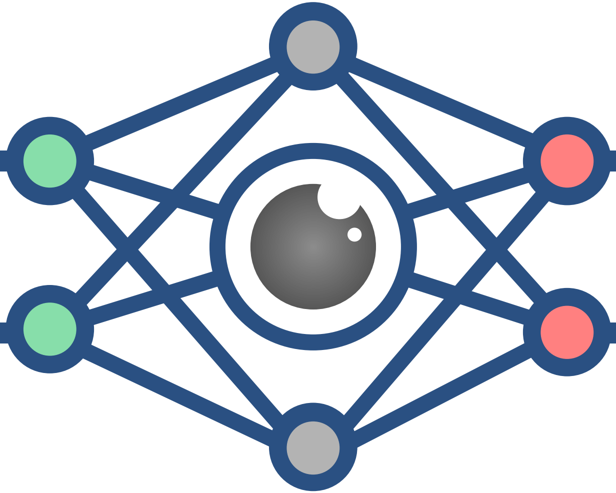
CS3485

Deep Learning for Computer Vision



Lec 2: Linear Classification and Perceptron

Announcements

- Lab 1 is out:
  - Make sure to find a pair to work on it with. If you can't find one, let me know **by Tuesday**.
  - It is an easy lab: you'll just need the basis of Python/Numpy + this slide deck. Feel free to ask me or come to my office hours if you have questions about either Python or Numpy.
  - The instructions carry a little info on what I expect in the report. I'll go easy on the grade this time, so you know what to improve for the next lab.
  - Keep in mind your late day budget (4 for **all labs**).
  - **Suggestion**: Use Overleaf for starting off with Latex.
- Staring today, we'll start seeing a bit of math. Make sure to check out our preliminary math materials, if you need help over there

Announcements

- Amazon Go is a flop! It seems that it won't grow bigger than small shops

In a statement, Amazon said it will continue using the Just Walk Out technology in Amazon Go stores, at smaller format Fresh stores in the UK, and third-party locations such as certain sports stadiums and college campuses

- On the other hand, another CV technology is taking place: Dash Carts!

(Tentative) Lecture Roadmap

Basics of Deep Learning

- Intro to Computer Vision
- Linear Classifiers and Perceptron
- Multilayer Perceptron
- Optimization in Deep Learning
- Pytorch I – MLPs
- Pytorch II – Images and Regularization
- Convolutional Neural Networks
- Data Augmentation and Deep CNNs
- Transfer Learning and Residual Nets
- Inception Net and what CNNs learn
- Adversarial Examples and Self-supervision

Deep Learning and Computer Vision in Practice

- Intro to MLOps
- Intro to Object Detection
- Fast Object Detection
- Intro to Image Segmentation
- Applications of Detection and Segmentation
- Autoencoders
- Image Generation with GANs
- Advanced GANs
- The Attention Mechanism
- Transformers and ChatGPT
- Image Generation by Prompt
- Misc. Topics

The image classification problem

- The first task in Computer Vision we are tackling is that of Image Classification:

  Image Classification the process of recognition, understanding, and grouping of images into preset categories or classes.

- We want a model (or rule) that effectively categorizes (unseen) images into a set of target classes.
- In order to find this rule, we have a set of **labeled example images** at our disposal that we can **train** our model on and **learn** that rule.
- This process of finding such a model from labeled data is called **Supervised Learning.**

Labeled images of cats

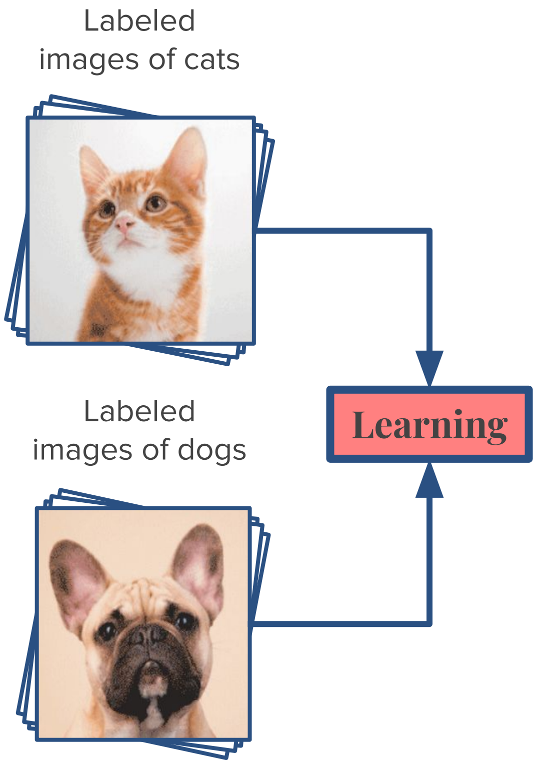
Labeled images of dogs

The image classification problem

- The first task in Computer Vision we are tackling is that of Image Classification:

  Image Classification the process of recognition, understanding, and grouping of images into preset categories or classes.

- We want a model (or rule) that effectively categorizes (unseen) images into a set of target classes.
- In order to find this rule, we have a set of **labeled example images** at our disposal that we can **train** our model on and **learn** that rule.
- This process of finding such a model from labeled data is called **Supervised Learning.**

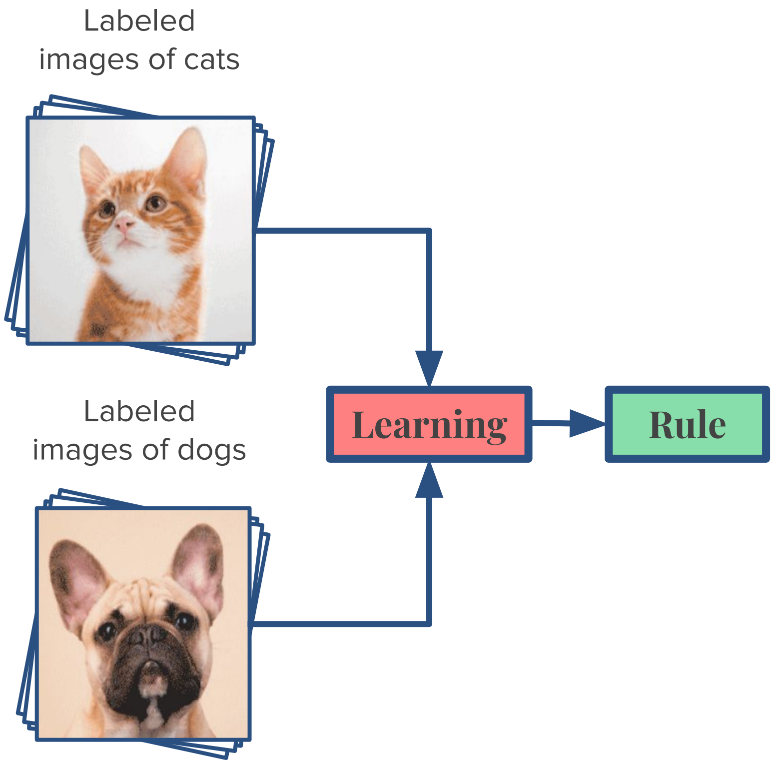


The image classification problem

- The first task in Computer Vision we are tackling is that of Image Classification:

  Image Classification the process of recognition, understanding, and grouping of images into preset categories or classes.

- We want a model (or rule) that effectively categorizes (unseen) images into a set of target classes.
- In order to find this rule, we have a set of **labeled example images** at our disposal that we can **train** our model on and **learn** that rule.
- This process of finding such a model from labeled data is called **Supervised Learning.**



The image classification problem

- The first task in Computer Vision we are tackling is that of Image Classification:

  Image Classification the process of recognition, understanding, and grouping of images into preset categories or classes.

- We want a model (or rule) that effectively categorizes (unseen) images into a set of target classes.
- In order to find this rule, we have a set of **labeled example images** at our disposal that we can **train** our model on and **learn** that rule.
- This process of finding such a model from labeled data is called **Supervised Learning.**

Unseen (unlabeled) images

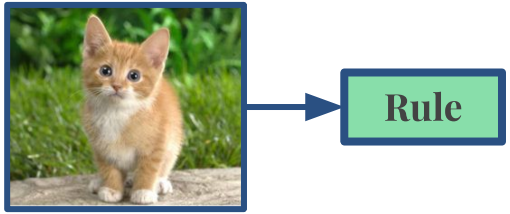
(Predicted) classes

The image classification problem

- The first task in Computer Vision we are tackling is that of Image Classification:

  Image Classification the process of recognition, understanding, and grouping of images into preset categories or classes.

- We want a model (or rule) that effectively categorizes (unseen) images into a set of target classes.
- In order to find this rule, we have a set of **labeled example images** at our disposal that we can **train** our model on and **learn** that rule.
- This process of finding such a model from labeled data is called **Supervised Learning.**

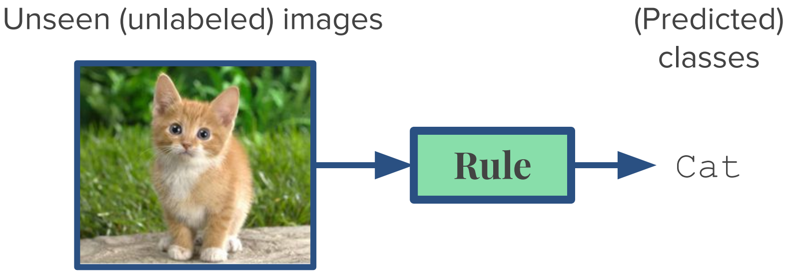


The image classification problem

- The first task in Computer Vision we are tackling is that of Image Classification:

  Image Classification the process of recognition, understanding, and grouping of images into preset categories or classes.

- We want a model (or rule) that effectively categorizes (unseen) images into a set of target classes.
- In order to find this rule, we have a set of **labeled example images** at our disposal that we can **train** our model on and **learn** that rule.
- This process of finding such a model from labeled data is called **Supervised Learning.**

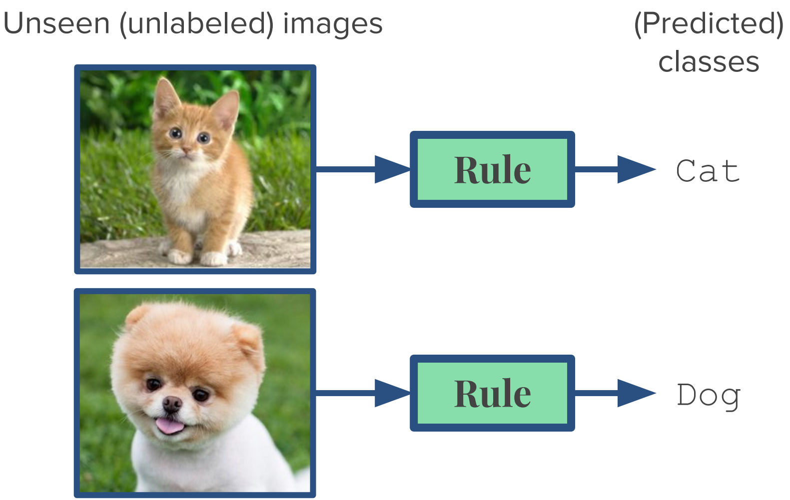


The image classification problem

- The first task in Computer Vision we are tackling is that of Image Classification:

  Image Classification the process of recognition, understanding, and grouping of images into preset categories or classes.

- We want a model (or rule) that effectively categorizes (unseen) images into a set of target classes.
- In order to find this rule, we have a set of **labeled example images** at our disposal that we can **train** our model on and **learn** that rule.
- This process of finding such a model from labeled data is called **Supervised Learning.**

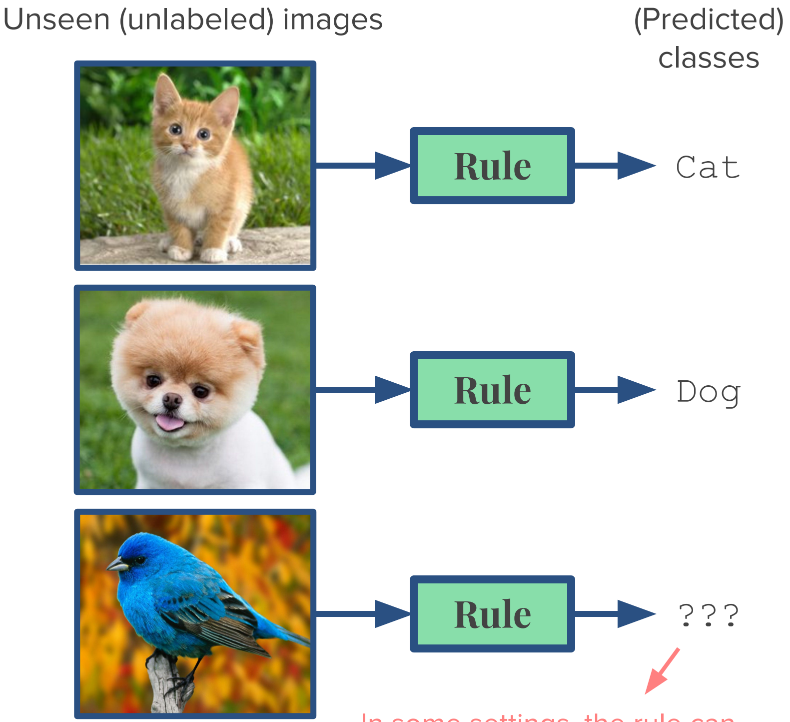


The image classification problem

- The first task in Computer Vision we are tackling is that of Image Classification:

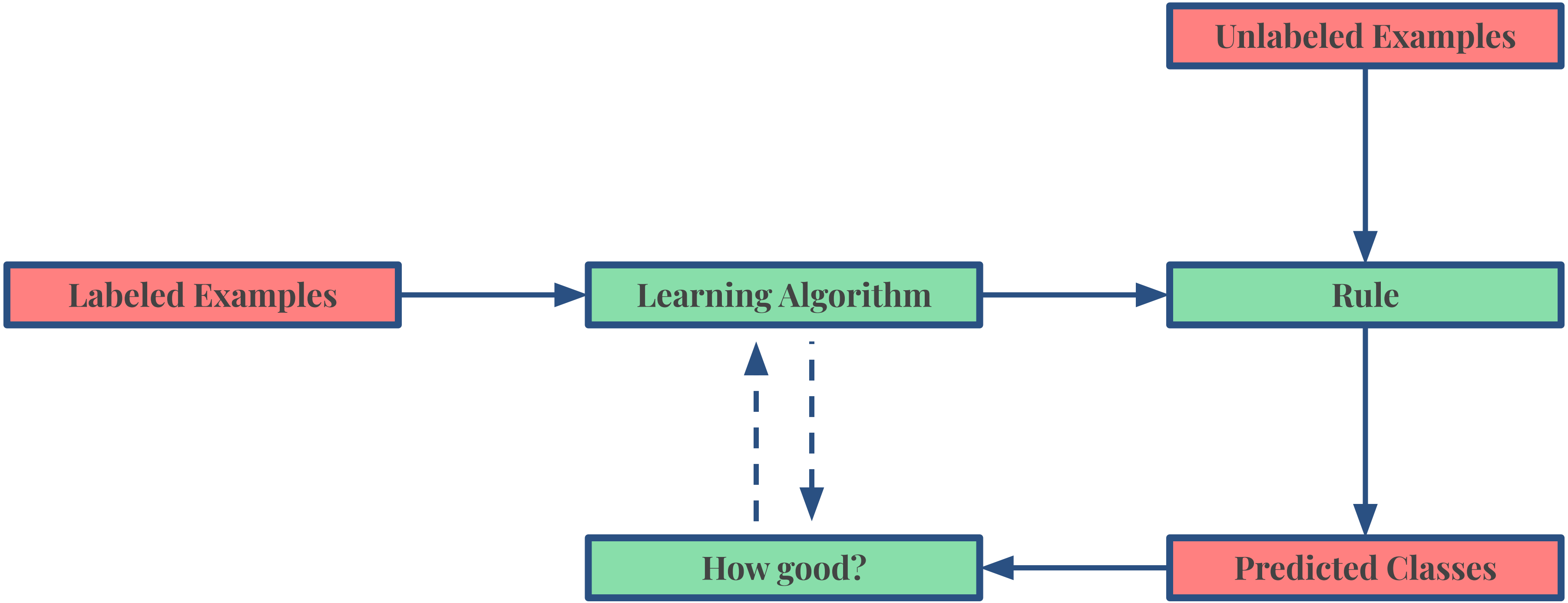
  Image Classification the process of recognition, understanding, and grouping of images into preset categories or classes.

- We want a model (or rule) that effectively categorizes (unseen) images into a set of target classes.
- In order to find this rule, we have a set of **labeled example images** at our disposal that we can **train** our model on and **learn** that rule.
- This process of finding such a model from labeled data is called **Supervised Learning.**



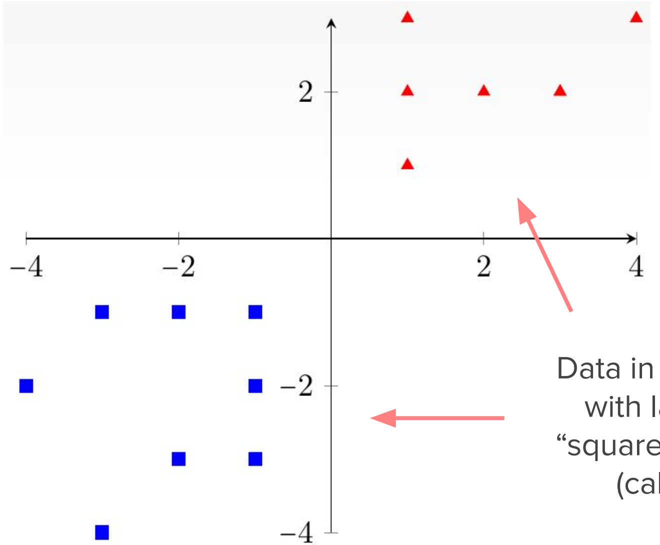
In some settings, the rule can also output “don’t know”!

Supervised Classification Pipeline



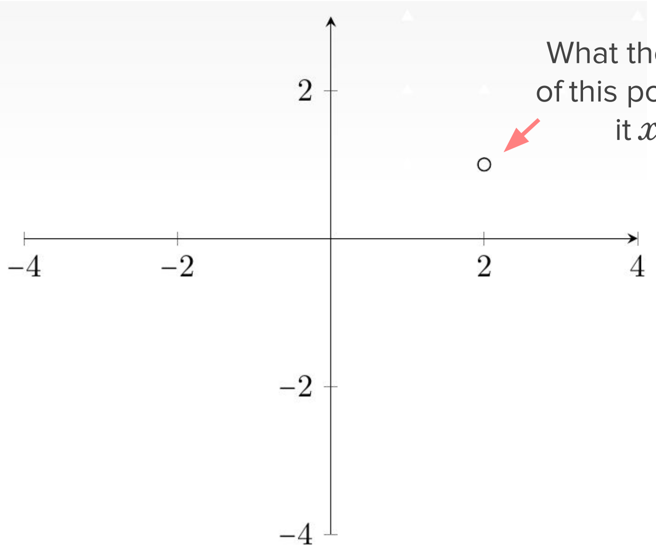
Example of Classification Problem

Original Data



Data in 2 dimensions
with labels either
“square” or “triangle”
(call them y).

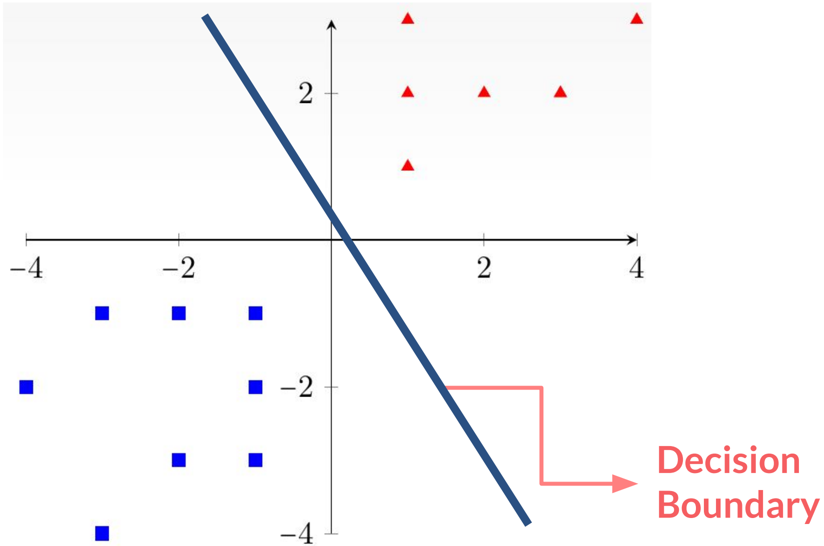
New Unlabeled Datapoint



What the class
of this point (call
it x)?

Linear Classifiers

Original Data

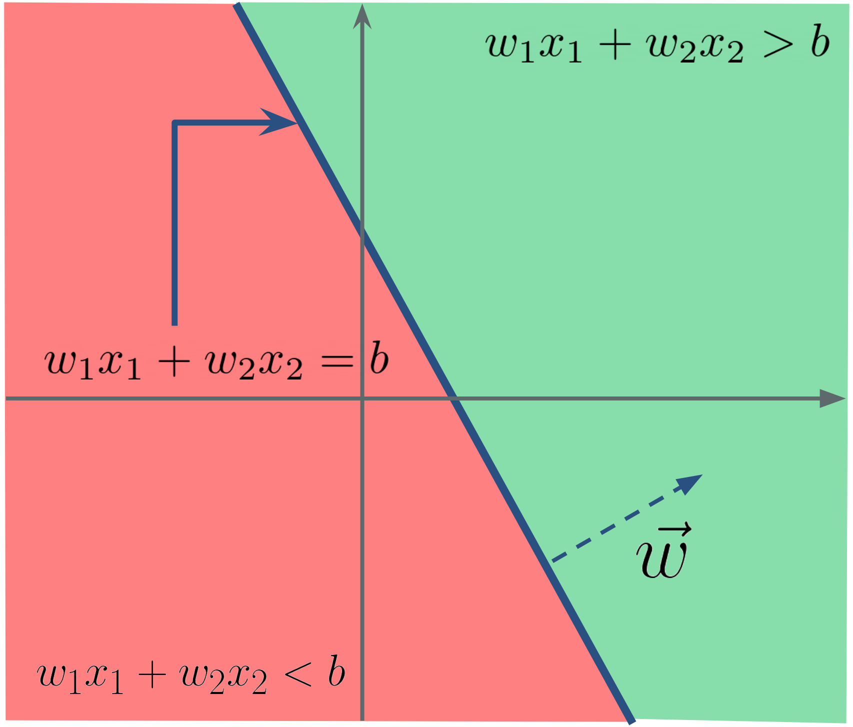


- We need to find a classification rule (**decision boundary**) based on the labeled data.
- Today's choice:

Linear Classifiers

- Which means: "*If x is on one side of the line, it is a triangle, otherwise it is a square*".
- How to define the line and its sides **mathematically**, so we can come up with algorithms?

A linear classifier in 2D

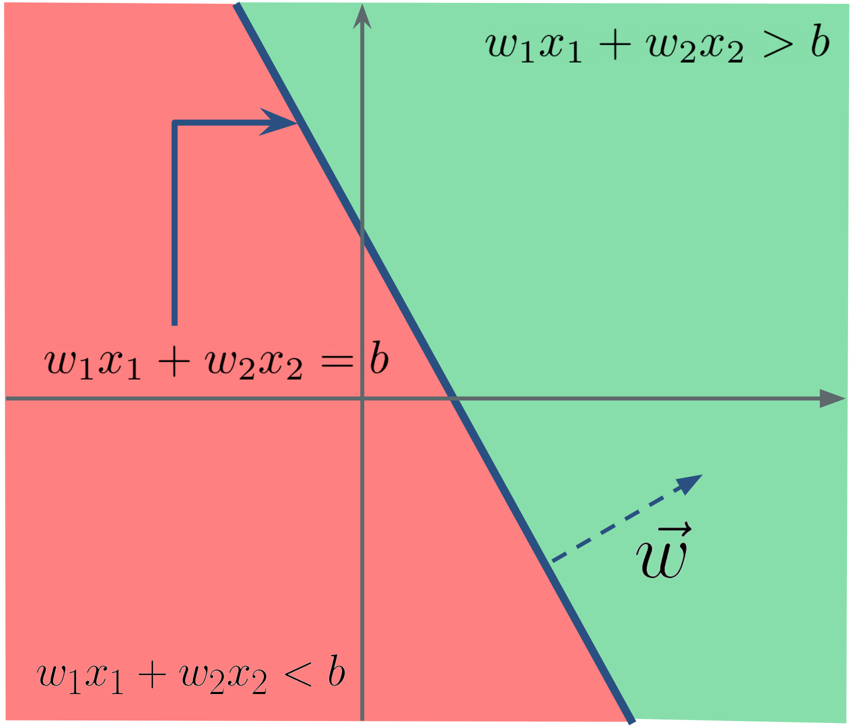


- In 2D, we represent a line using **three numbers**:
  - Two to form a vector $w = [w_1, w_2]$ called **weight vector**;
  - One number called **bias**, b.
- If a new point $x = [x_1, x_2]$ comes in, we just check whether:

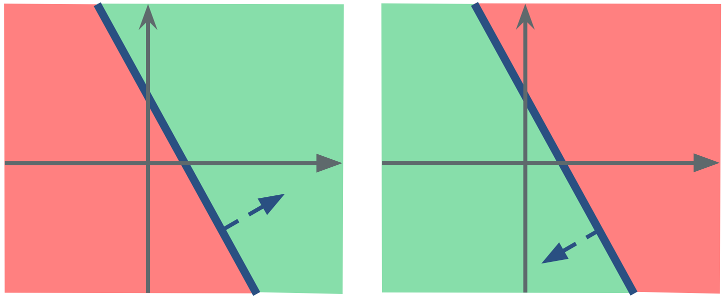
$$w_1x_1 + w_2x_2 > b$$

- If True, x lies on one side of the plane, if False it belongs to the other side
- If equal, it x is exactly **on the line**, and it can be classified as either True or False.

A linear classifier in 2D

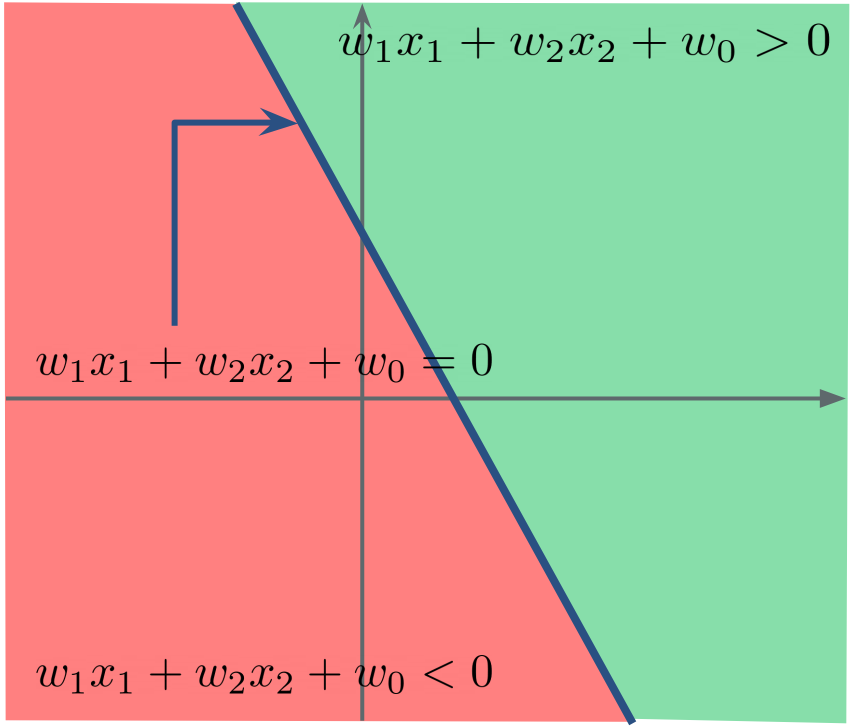


- The **direction of the vector of weights** plays a role here too.
- It always points to the side where the value of $w_1x_1 + w_2x_2 > b$ is True:



- The boundary is, however, the same in both cases, and one can change the direction of w by setting $w = -w$.

A linear classifier in 2D



- Now, we can also define the weight vector to include b, making:

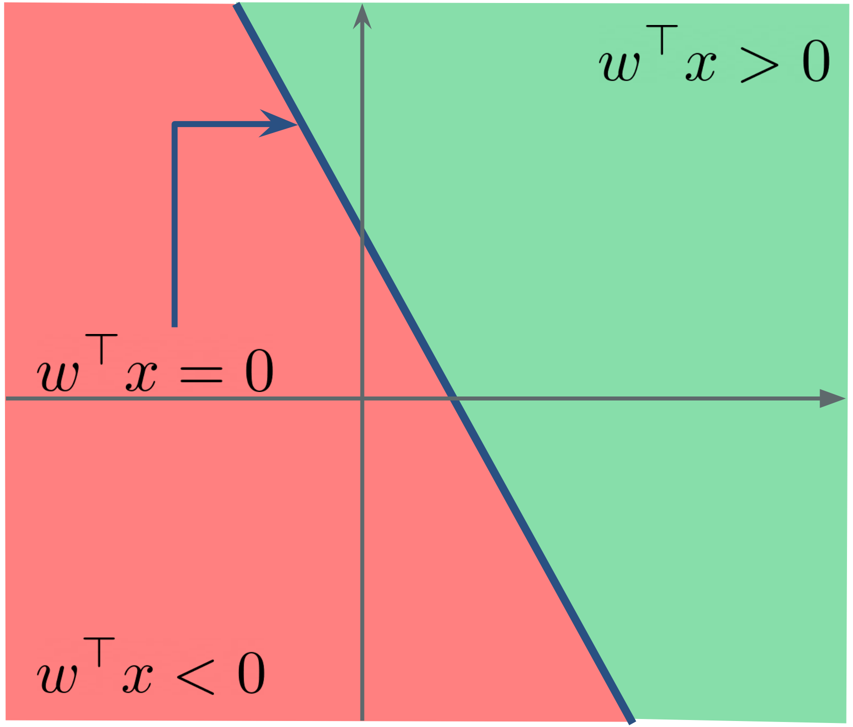
$$w = [w_0, w_1, w_2]$$

  where $b = -w_0$.
- Now, **very importantly**, because of that change in w, we need to add a new dimension with a "1" to all data points x:

$$x = [1, x_1, x_2]$$

- For example, if x was $[5, 7]$ initially, now it will be $[1, 5, 7]$.
- We'll use this change in today's examples.

A linear classifier in 2D



- Finally, we can use the following notation:

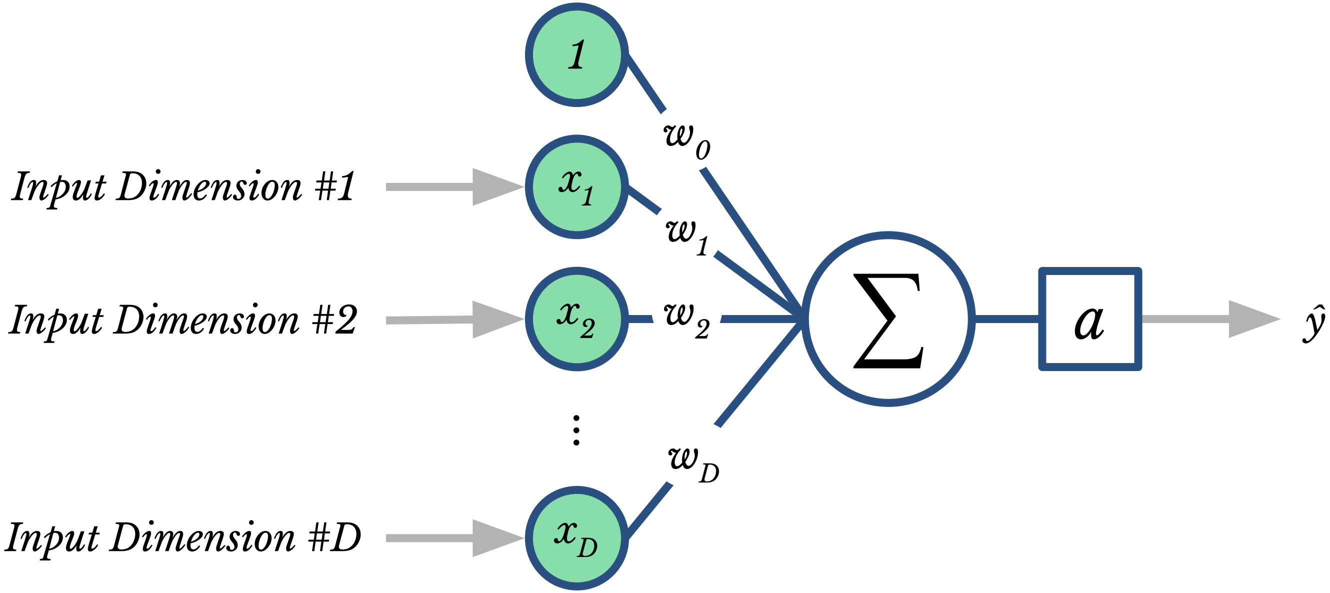
$$w^\top x = [w_0, w_1, w_2]^\top [1, x_1, x_2]$$
$$= w_0 + w_1 x_1 + w_2 x_2,$$

  where $^\top$ is the transpose operation.
- This notation is called the **inner product**, and it is handy since it is the same even if our data points are of $D > 2$ dimensions.
- **Mathematically**, the predicted class $\hat{y}$ of a point x by a linear classifier given by w is:

$$\hat{y} = \mathrm{sign}(w^\top x) = \begin{cases} 1, & \text{if } w^\top x \geq 0 \\ -1, & \text{if } w^\top x < 0 \end{cases}$$

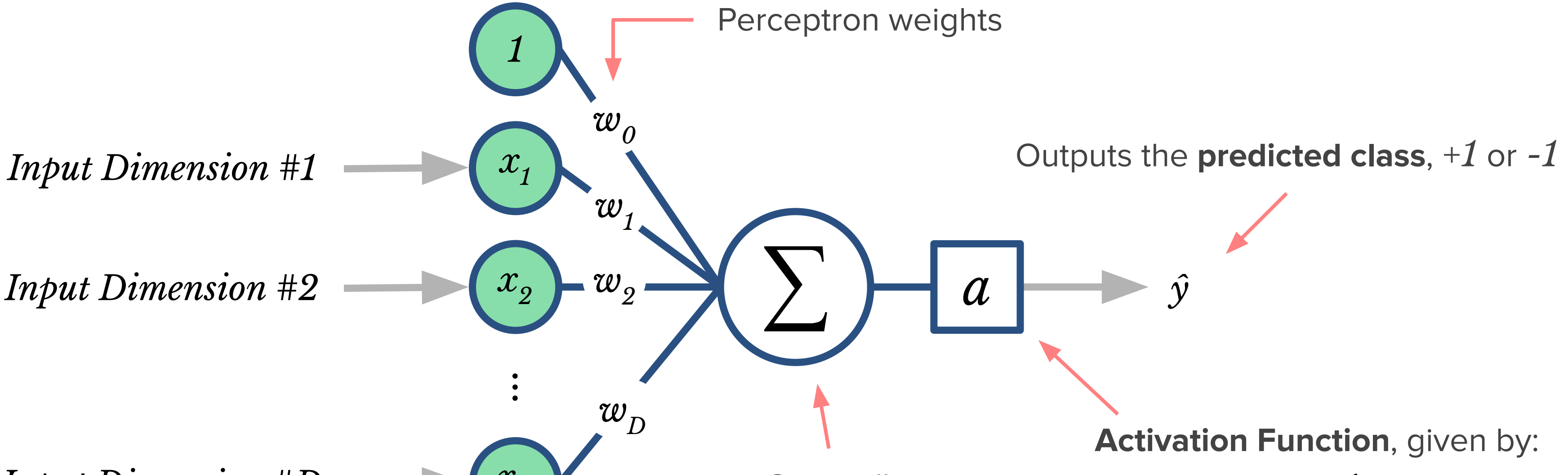
Linear Classifier Model: Perceptron

- Using these concepts, we can build a **model for classification** called **perceptron**!



Linear Classifier Model: Perceptron

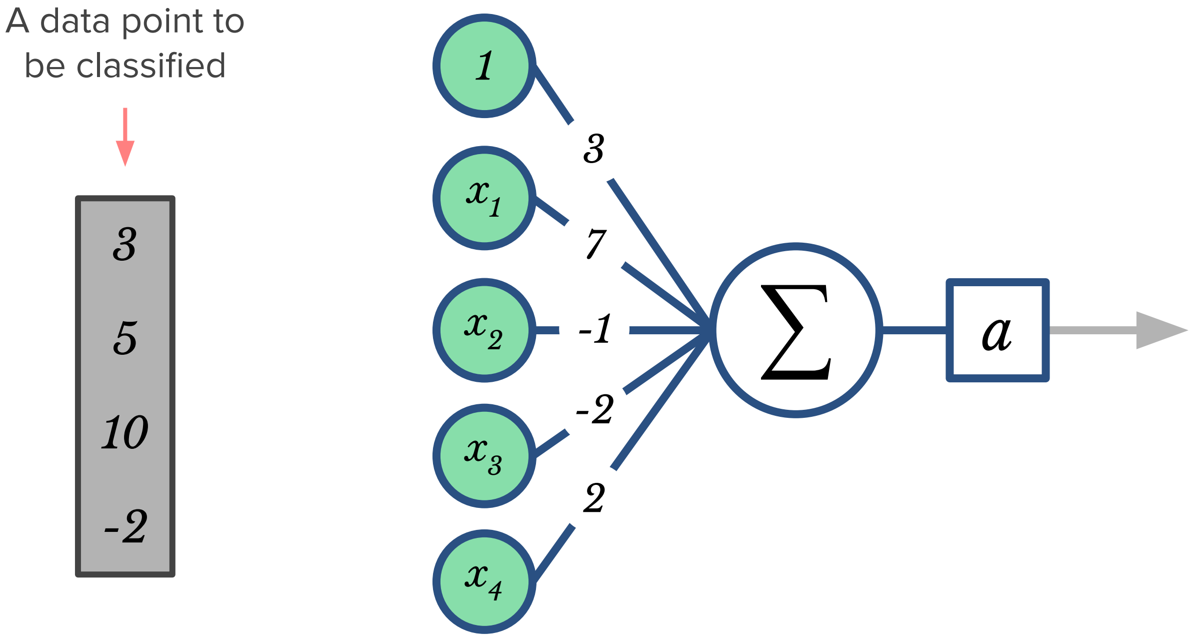
- Below, you have some important nomenclature of the inner workings of the perceptron:



$$a(z) = \text{sign}(z) = \begin{cases} 1, & \text{if } z \geq 0 \\ -1, & \text{if } z < 0 \end{cases}$$

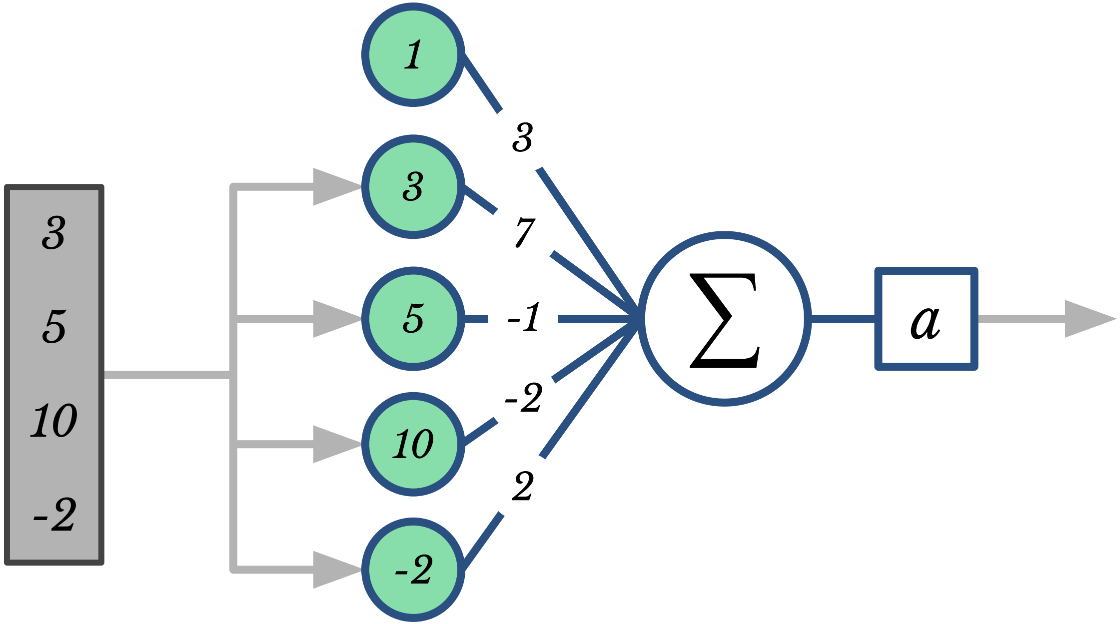
Linear Classifier Model: Perceptron

- Say **we know the perceptron weights**, then classifying a new point is easy. For example:



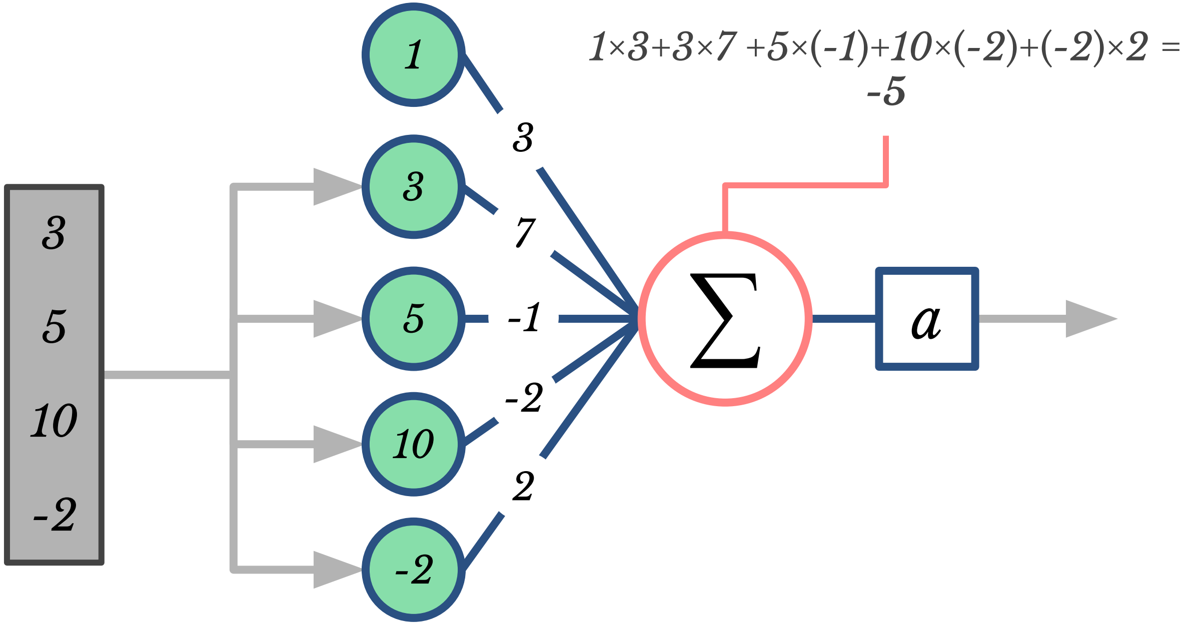
Linear Classifier Model: Perceptron

- Say **we know the perceptron weights**, then classifying a new point is easy. For example:



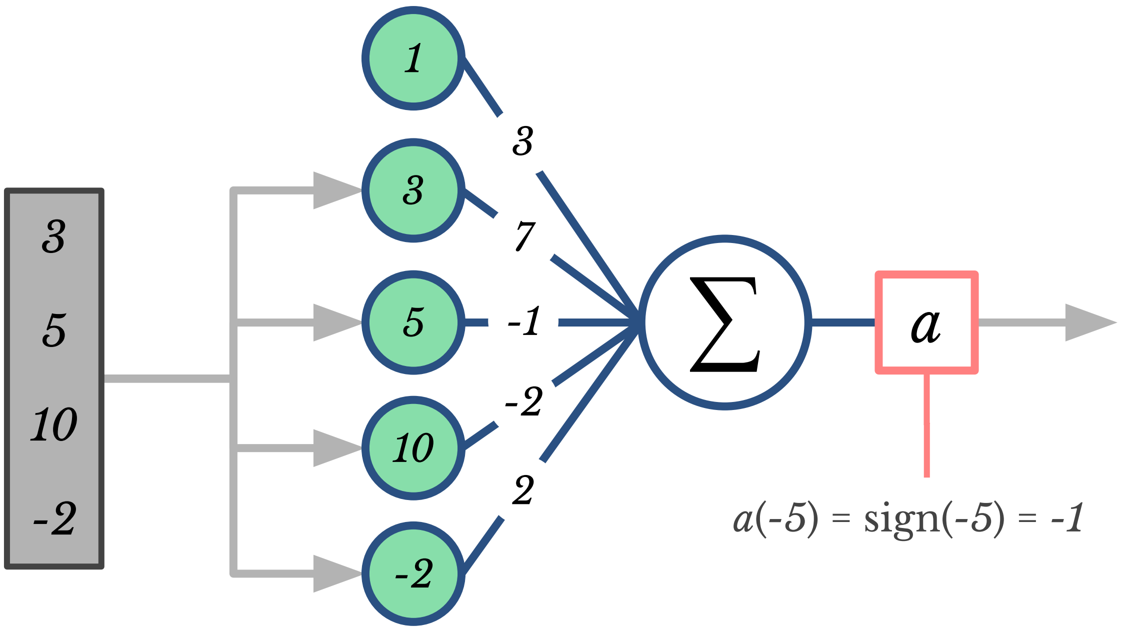
Linear Classifier Model: Perceptron

- Say **we know the perceptron weights**, then classifying a new point is easy. For example:



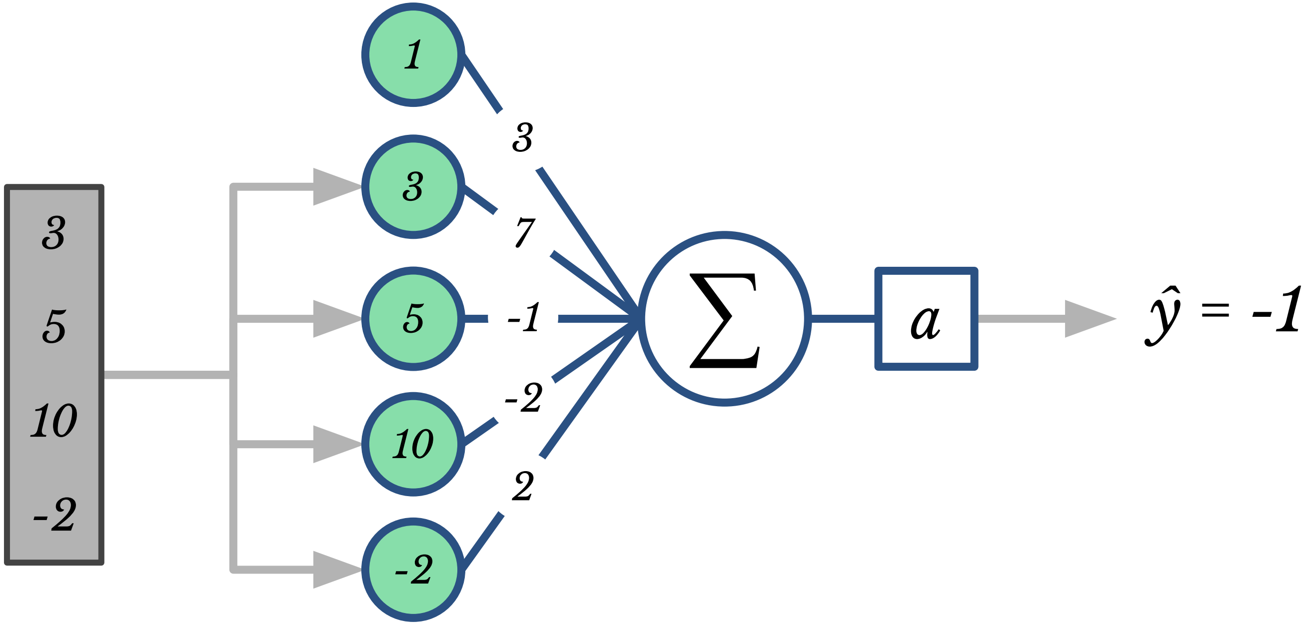
Linear Classifier Model: Perceptron

- Say **we know the perceptron weights**, then classifying a new point is easy. For example:



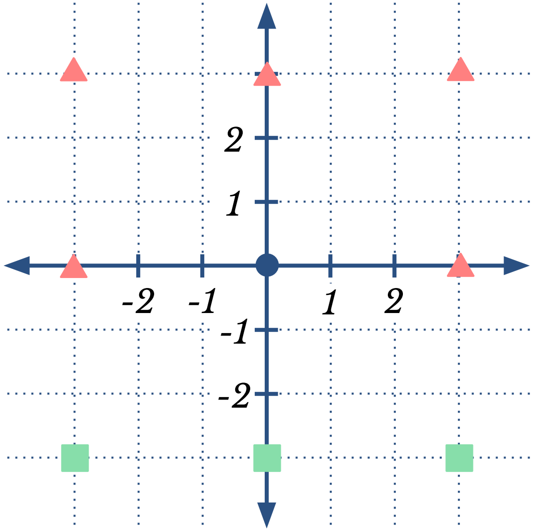
Linear Classifier Model: Perceptron

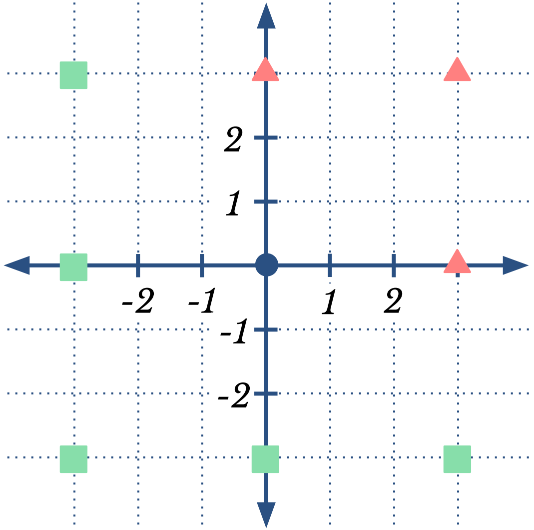
- This process is called **Forward Pass.**



Exercise (*In pairs*)

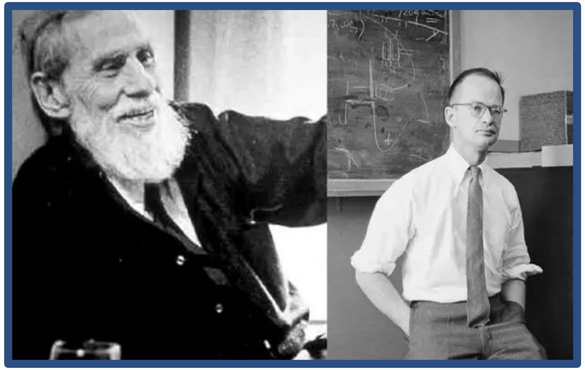
- Find weights $w = [w_0, w_1, w_2]$ for the lines that separate the triangles from the rectangles. After that, apply a perceptron model with weights given by w to $x = [2, 2]$ and $x = [-2, -2]$.

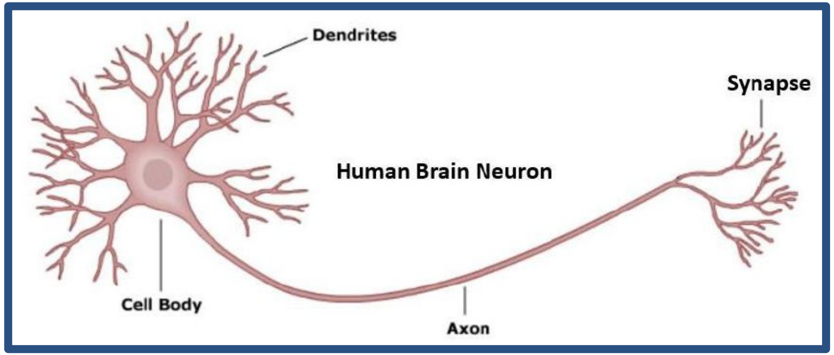




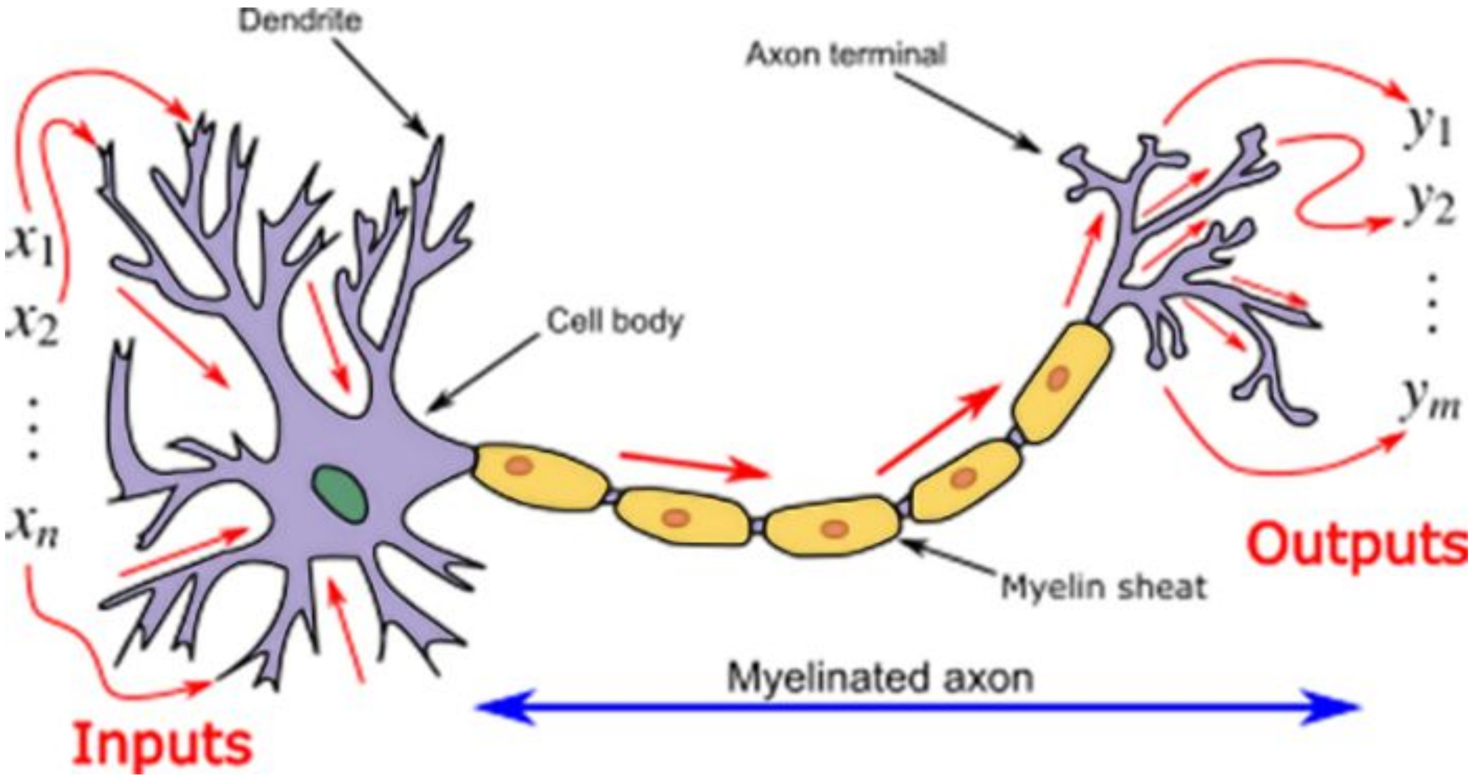
Neurons and the perceptron

- The perceptron model was developed to mathematically model **human neurons**!
- It was proposed **Warren MuCulloch** (neuroscientist) and **Walter Pitts** (logician) in 1943.
- It is considered the first **Artificial Neural Network** model and is the basis of deep learning.





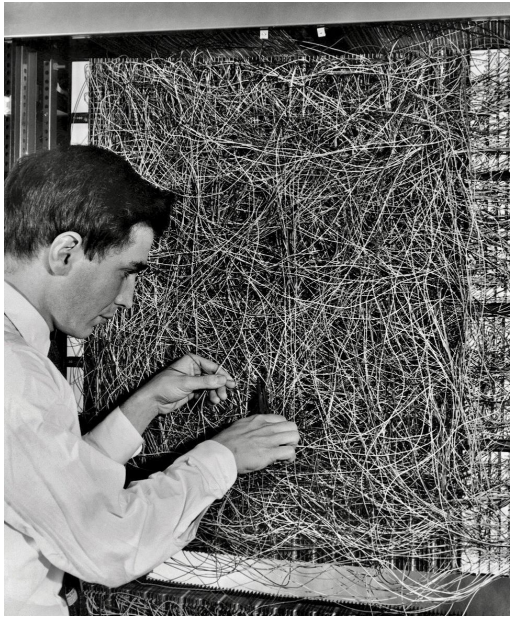
The Neuron



Dendrite
Axon terminal
x_1
x_2
$\vdots$
x_n
Cell body
y_1
y_2
$\vdots$
y_m
Outputs
Myelin sheat
Myelinated axon
Inputs

Supervised Learning with the Perceptron

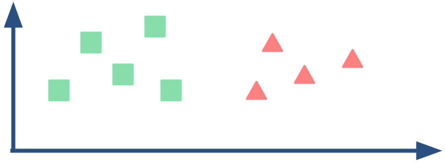
- The **perceptron** needs a linear classifier when classifying.
- We need then a way to **compute the perceptron weights** $w_0, w_1, w_2, \dots, w_D$.
- We can **learn** them from a training dataset S using the **Perceptron Algorithm**, first implemented by **Frank Rosenblatt** in 1958.
- We can show that, if S is **linearly separable**, it always finds an optimal decision boundary.



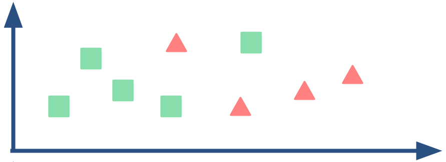
Frank Rosenblatt working on the perceptron algorithm implementation at Cornell in 1958.

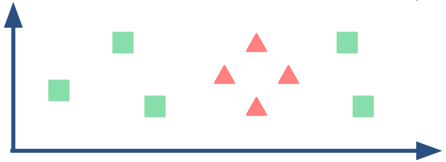
Examples of linear separability in datasets

Linearly separable dataset



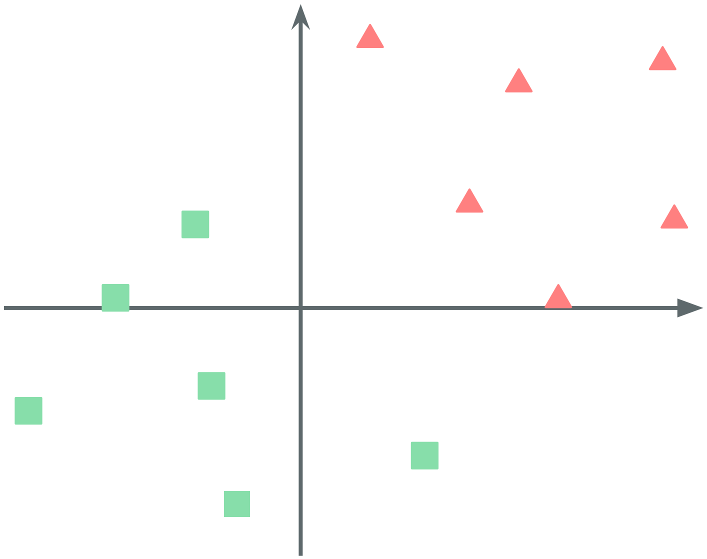
Non-Linearly separable datasets





The Perceptron Algorithm

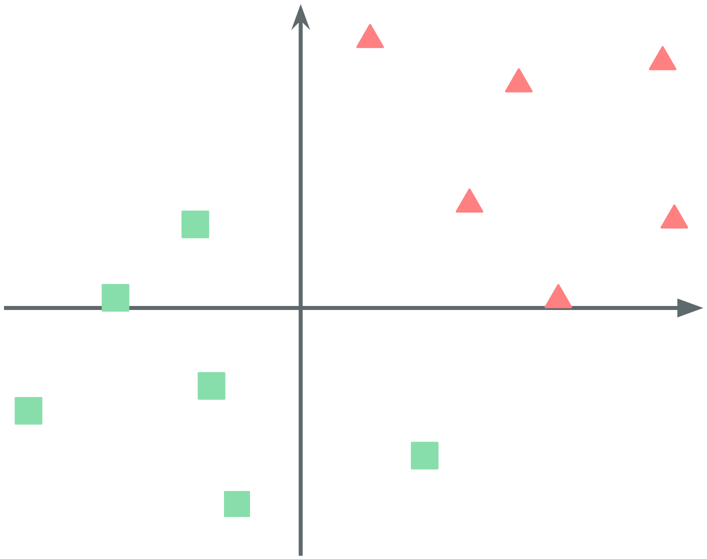
- There are n points $x^{(1)}, ..., x^{(n)}$ in D dimensions*, each with a class $y^{(1)}, ..., y^{(n)}$ of either -1 or $+1$.
- The perceptron algorithm is:
  1. Start with a random w in $D+1$ dimensions*.
  2. For i in 1 to n, do:
     a. Find the **predicted class**, $\hat{y}^{(i)} = a(w^{\mathsf{T}}x^{(i)})$.
     b. If $y^{(i)} = \hat{y}^{(i)}$, keep w the same ($x^{(i)}$ is correctly classified in this case).
     c. If $y^{(i)} = +1$ and $\hat{y}^{(i)} = -1$: Do $w = w + x^{(i)}$
     d. If $y^{(i)} = -1$ and $\hat{y}^{(i)} = +1$: Do $w = w - x^{(i)}$
  3. Repeat step 2 (go over the dataset again) until all points are correctly classified.



* Remember that the points are added a new dimension with a 1 to account for the bias term, go here for more details.

The Perceptron Algorithm

- There are n points $x^{(1)}$, ..., $x^{(n)}$ in D dimensions*, each with a class $y^{(1)}$, ..., $y^{(n)}$ of either -1 or $+1$.
- The perceptron algorithm is:
  1. Start with a random w in $D+1$ dimensions*.
  2. For i in 1 to n, do:
     a. Find the **predicted class**, $\hat{y}^{(i)} = a(w^\top x^{(i)})$.
     b. If $y^{(i)} = \hat{y}^{(i)}$, keep w the same ($x^{(i)}$ is correctly classified in this case).
     c. If $y^{(i)} = +1$ and $\hat{y}^{(i)} = -1$: Do $w = w + x^{(i)}$
     d. If $y^{(i)} = -1$ and $\hat{y}^{(i)} = +1$: Do $w = w - x^{(i)}$
  3. Repeat step 2 (go over the dataset again) until all points are correctly classified.

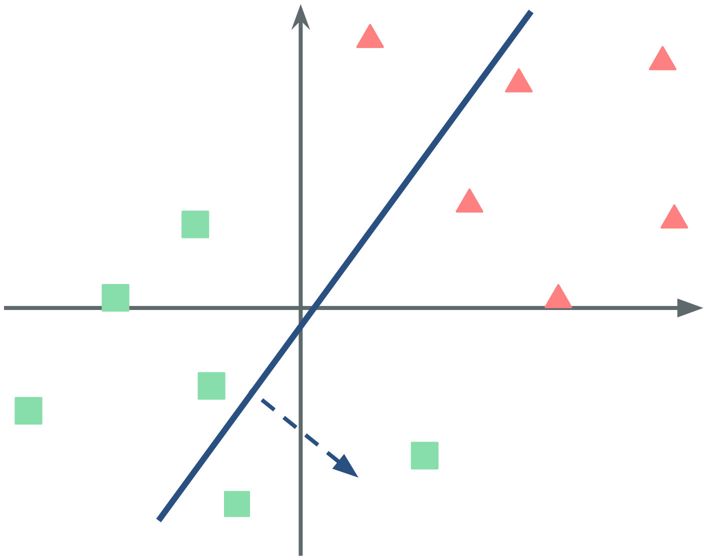


Set triangles to have label $+1$ and squares to have label -1.

* Remember that the points are added a new dimension with a 1 to account for the bias term, go here for more details.

The Perceptron Algorithm

- There are n points $x^{(1)}, ..., x^{(n)}$ in D dimensions*, each with a class $y^{(1)}, ..., y^{(n)}$ of either -1 or $+1$.
- The perceptron algorithm is:
  1. Start with a random w in $D+1$ dimensions*.
  2. For i in 1 to n, do:
     a. Find the **predicted class**, $\hat{y}^{(i)} = a(w^\top x^{(i)})$.
     b. If $y^{(i)} = \hat{y}^{(i)}$, keep w the same ($x^{(i)}$ is correctly classified in this case).
     c. If $y^{(i)} = +1$ and $\hat{y}^{(i)} = -1$: Do $w = w + x^{(i)}$
     d. If $y^{(i)} = -1$ and $\hat{y}^{(i)} = +1$: Do $w = w - x^{(i)}$
  3. Repeat step 2 (go over the dataset again) until all points are correctly classified.

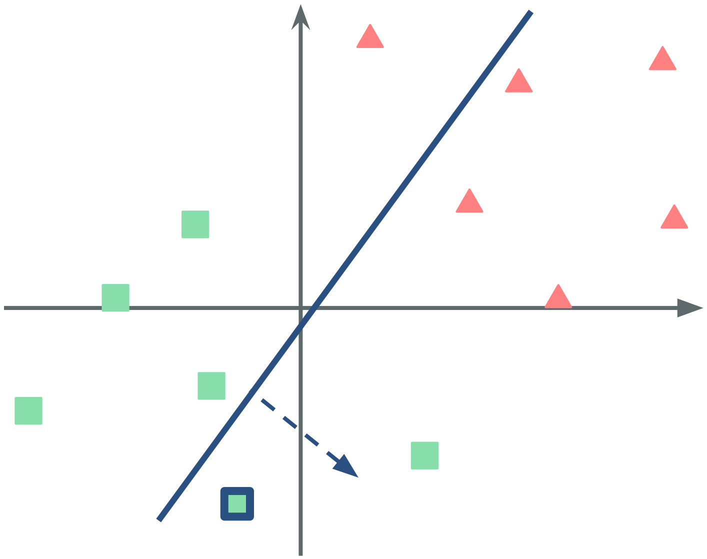


Start with a random w, which represents a random line.

* Remember that the points are added a new dimension with a 1 to account for the bias term, go [here](#) for more details.

The Perceptron Algorithm

- There are n points $x^{(1)}, ..., x^{(n)}$ in D dimensions*, each with a class $y^{(1)}, ..., y^{(n)}$ of either -1 or $+1$.
- The perceptron algorithm is:
  1. Start with a random w in $D+1$ dimensions*.
  2. For i in 1 to n, do:
     a. Find the **predicted class**, $\hat{y}^{(i)} = a(w^{\mathsf{T}}x^{(i)})$.
     b. If $y^{(i)} = \hat{y}^{(i)}$, keep w the same ($x^{(i)}$ is correctly classified in this case).
     c. If $y^{(i)} = +1$ and $\hat{y}^{(i)} = -1$: Do $w = w + x^{(i)}$
     d. If $y^{(i)} = -1$ and $\hat{y}^{(i)} = +1$: Do $w = w - x^{(i)}$
  3. Repeat step 2 (go over the dataset again) until all points are correctly classified.

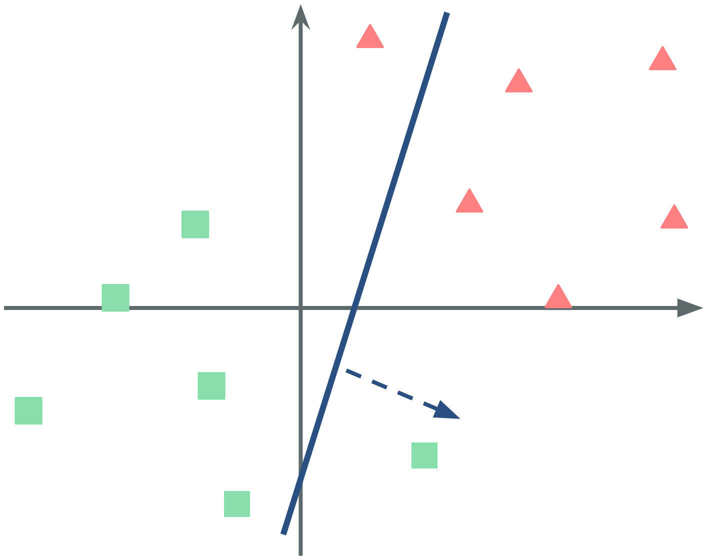


Go over the points, until you find one whose $\hat{y}_i$ does not match with its true class, y_i.

* Remember that the points are added a new dimension with a 1 to account for the bias term, go here for more details.

The Perceptron Algorithm

- There are n points $x^{(1)}, \dots, x^{(n)}$ in D dimensions*, each with a class $y^{(1)}, \dots, y^{(n)}$ of either -1 or $+1$.
- The perceptron algorithm is:
  1. Start with a random w in $D+1$ dimensions*.
  2. For i in 1 to n, do:
     a. Find the **predicted class**, $\hat{y}^{(i)} = a(w^{\top}x^{(i)})$.
     b. If $y^{(i)} = \hat{y}^{(i)}$, keep w the same ($x^{(i)}$ is correctly classified in this case).
     c. If $y^{(i)} = +1$ and $\hat{y}^{(i)} = -1$: Do $w = w + x^{(i)}$
     d. If $y^{(i)} = -1$ and $\hat{y}^{(i)} = +1$: Do $w = w - x^{(i)}$
  3. Repeat step 2 (go over the dataset again) until all points are correctly classified.

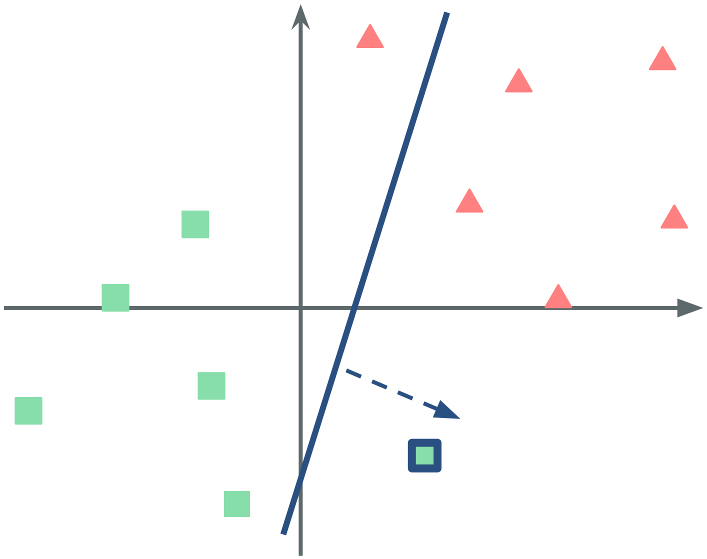


Change w according to the mismatch.

* Remember that the points are added a new dimension with a 1 to account for the bias term, go here for more details.

The Perceptron Algorithm

- There are n points $x^{(1)}, \ldots, x^{(n)}$ in D dimensions*, each with a class $y^{(1)}, \ldots, y^{(n)}$ of either -1 or $+1$.
- The perceptron algorithm is:
  1. Start with a random w in $D+1$ dimensions*.
  2. For i in 1 to n, do:
     a. Find the **predicted class**, $\hat{y}^{(i)} = a(w^{\top}x^{(i)})$.
     b. If $y^{(i)} = \hat{y}^{(i)}$, keep w the same ($x^{(i)}$ is correctly classified in this case).
     c. If $y^{(i)} = +1$ and $\hat{y}^{(i)} = -1$: Do $w = w + x^{(i)}$
     d. If $y^{(i)} = -1$ and $\hat{y}^{(i)} = +1$: Do $w = w - x^{(i)}$
  3. Repeat step 2 (go over the dataset again) until all points are correctly classified.

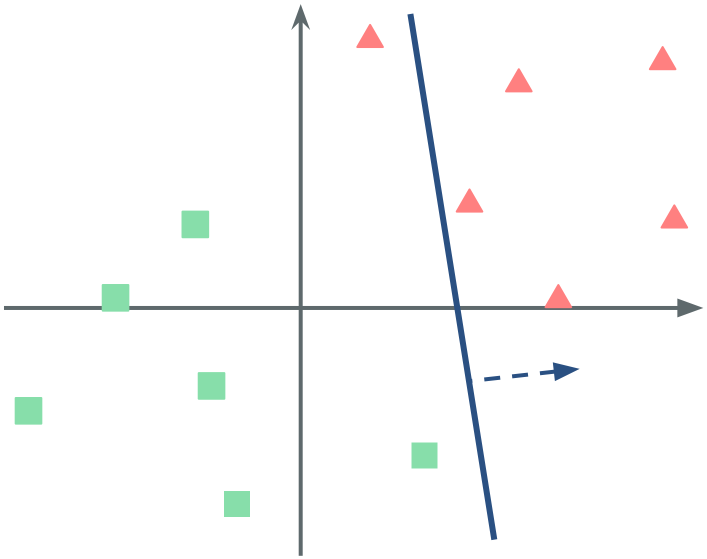


Go to the next data points where there is a mismatch.

* Remember that the points are added a new dimension with a 1 to account for the bias term, go here for more details.

The Perceptron Algorithm

- There are n points $x^{(1)}, ..., x^{(n)}$ in D dimensions*, each with a class $y^{(1)}, ..., y^{(n)}$ of either -1 or $+1$.
- The perceptron algorithm is:
  1. Start with a random w in $D+1$ dimensions*.
  2. For i in 1 to n, do:
     a. Find the **predicted class**, $\hat{y}^{(i)} = a(w^{\top}x^{(i)})$.
     b. If $y^{(i)} = \hat{y}^{(i)}$, keep w the same ($x^{(i)}$ is correctly classified in this case).
     c. If $y^{(i)} = +1$ and $\hat{y}^{(i)} = -1$: Do $w = w + x^{(i)}$
     d. If $y^{(i)} = -1$ and $\hat{y}^{(i)} = +1$: Do $w = w - x^{(i)}$
  3. Repeat step 2 (go over the dataset again) until all points are correctly classified.

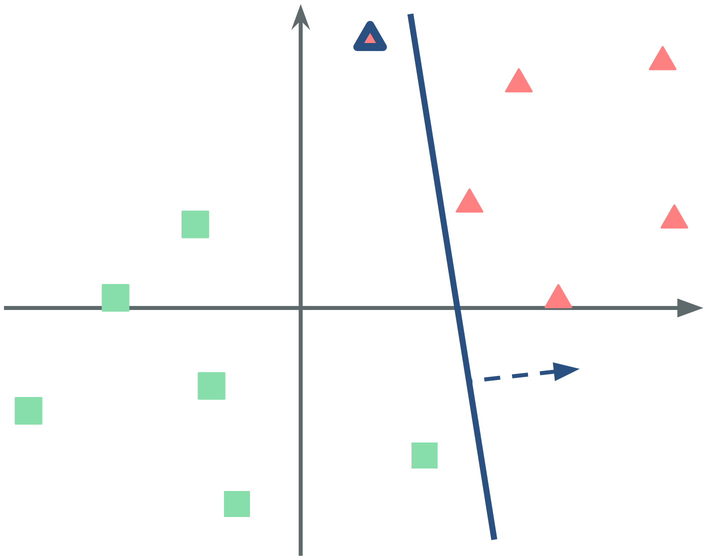


Change w according to the mismatch.

* Remember that the points are added a new dimension with a 1 to account for the bias term, go here for more details.

The Perceptron Algorithm

- There are n points $x^{(1)}, ..., x^{(n)}$ in D dimensions*, each with a class $y^{(1)}, ..., y^{(n)}$ of either -1 or $+1$.
- The perceptron algorithm is:
  1. Start with a random w in $D+1$ dimensions*.
  2. For i in 1 to n, do:
     a. Find the **predicted class**, $\hat{y}^{(i)} = a(w^{\mathsf{T}}x^{(i)})$.
     b. If $y^{(i)} = \hat{y}^{(i)}$, keep w the same ($x^{(i)}$ is correctly classified in this case).
     c. If $y^{(i)} = +1$ and $\hat{y}^{(i)} = -1$: Do $w = w + x^{(i)}$
     d. If $y^{(i)} = -1$ and $\hat{y}^{(i)} = +1$: Do $w = w - x^{(i)}$
  3. Repeat step 2 (go over the dataset again) until all points are correctly classified.

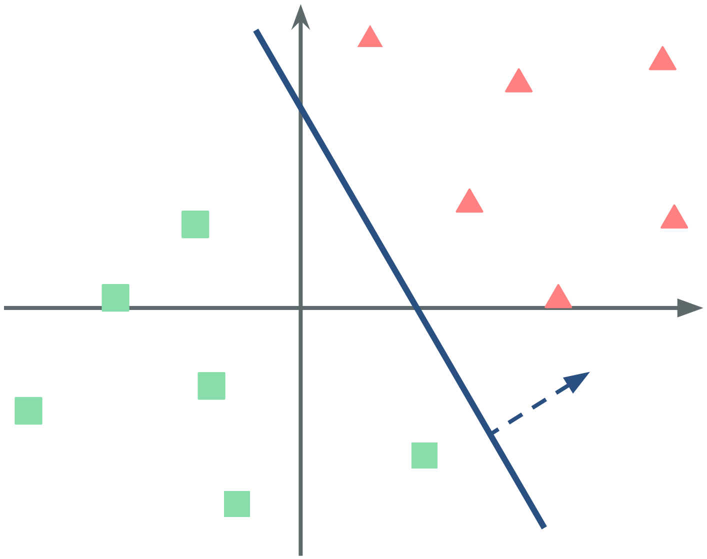


Do that until there are no mismatches.

* Remember that the points are added a new dimension with a 1 to account for the bias term, go here for more details.

The Perceptron Algorithm

- There are n points $x^{(1)}, \ldots, x^{(n)}$ in D dimensions*, each with a class $y^{(1)}, \ldots, y^{(n)}$ of either -1 or $+1$.
- The perceptron algorithm is:
  1. Start with a random w in $D+1$ dimensions*.
  2. For i in 1 to n, do:
     a. Find the **predicted class**, $\hat{y}^{(i)} = a(w^{\mathsf{T}}x^{(i)})$.
     b. If $y^{(i)} = \hat{y}^{(i)}$, keep w the same ($x^{(i)}$ is correctly classified in this case).
     c. If $y^{(i)} = +1$ and $\hat{y}^{(i)} = -1$: Do $w = w + x^{(i)}$
     d. If $y^{(i)} = -1$ and $\hat{y}^{(i)} = +1$: Do $w = w - x^{(i)}$
  3. Repeat step 2 (go over the dataset again) until all points are correctly classified.



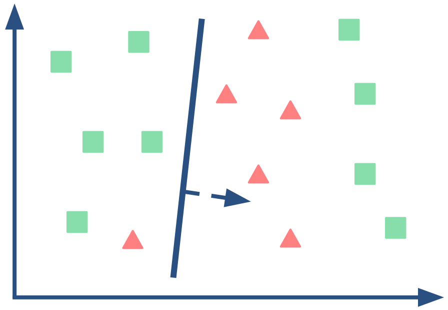
Do that until there are no mismatches.

* Remember that the points are added a new dimension with a 1 to account for the bias term, go here for more details.

Measuring classification efficiency

- For non-linearly separable datasets, the perceptron algorithm **won't find** a linear classifier that correctly classifies all points.
- If the classification isn't perfect, we need to find **a measure of how good it is.**
- One possible measure is our **Classification Accuracy** (Acc):

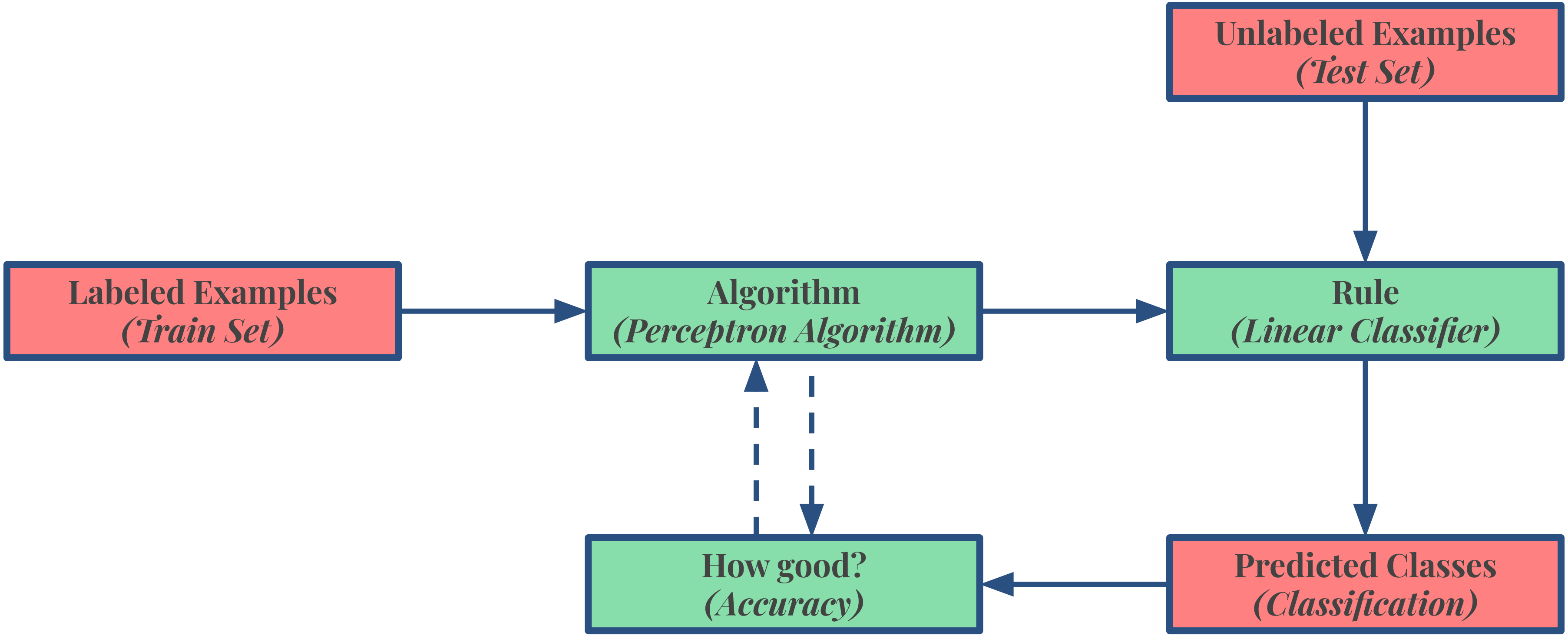
$$\mathrm{Acc} = \frac{\text{Number of correctly classified points}}{\text{Total number of points}}$$



If triangles are $+1$ and squares -1, the above classifier has an accuracy of 10/15 = 0.66%

- It is easy to evaluate a model's performance with it, since $0 \leq \mathrm{Acc} \leq 1$ and the accuracy higher the better.
- However, Acc only assumes "discrete" values, since we have a discrete number of points, which **can be a hindrance to many learning algorithms**.
- For that reason we may use a closely related measure called **loss** (*more on it next time*).

Classification Pipeline for the Perceptron



Exercise (*In pairs*)

- You have the points $x_1 = [-1, 0]$, $x_2 = [0, -1]$ and $x_3 = [1, 1]$. Assume rectangles are of class -1 and the triangle of class 1. Do the following:
  - Say we start with $w = [2, -1]$ and $b = 0$. Draw on the image above the linear separator that w and b generates.
  - Redefine w to be $w = [w_0, w_1, w_2]$. Change the definitions of x_1, x_2 and x_3, accordingly.
  - Perform each step of the perceptron algorithm to find the a new value w.
  - Draw on the image above the new linear separator defined by w.
  - Draw point $x_4 = [2, -2]$ and classify it using the new value for w.