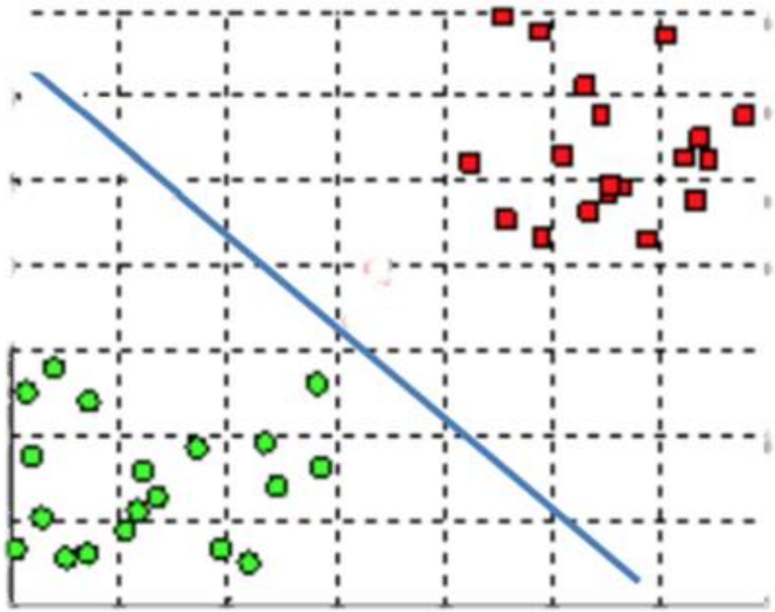
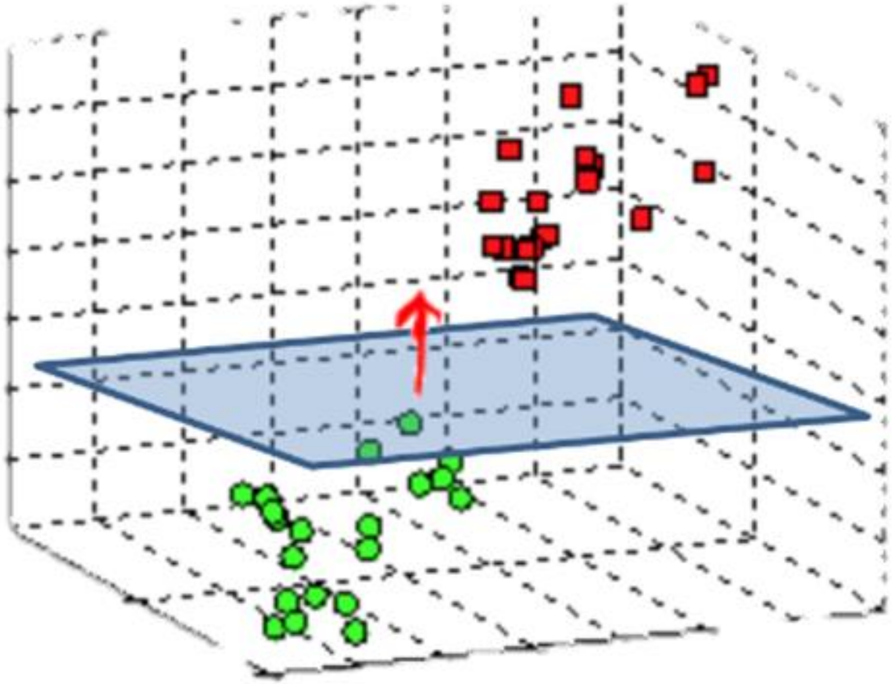




THE PERCEPTRON

Eitan Kosman

AGENDA

- The Problem
  - Linear Seperability – Def. 1
  - Linear Seperability – Def. 2
- A Biological Neuron
  - Structure
  - A Mathematical model
- The Solution – Perceptron
  - A brief history
  - The algorithm
  - Intuitive interpretation for weights update
  - Theorem: Mistake bound
  - The fall of perceptron
  - Stopping criterions
  - Kernel Perceptron

THE PROBLEM

Given a set of n points in $\mathbb{R}^d$ and labels:

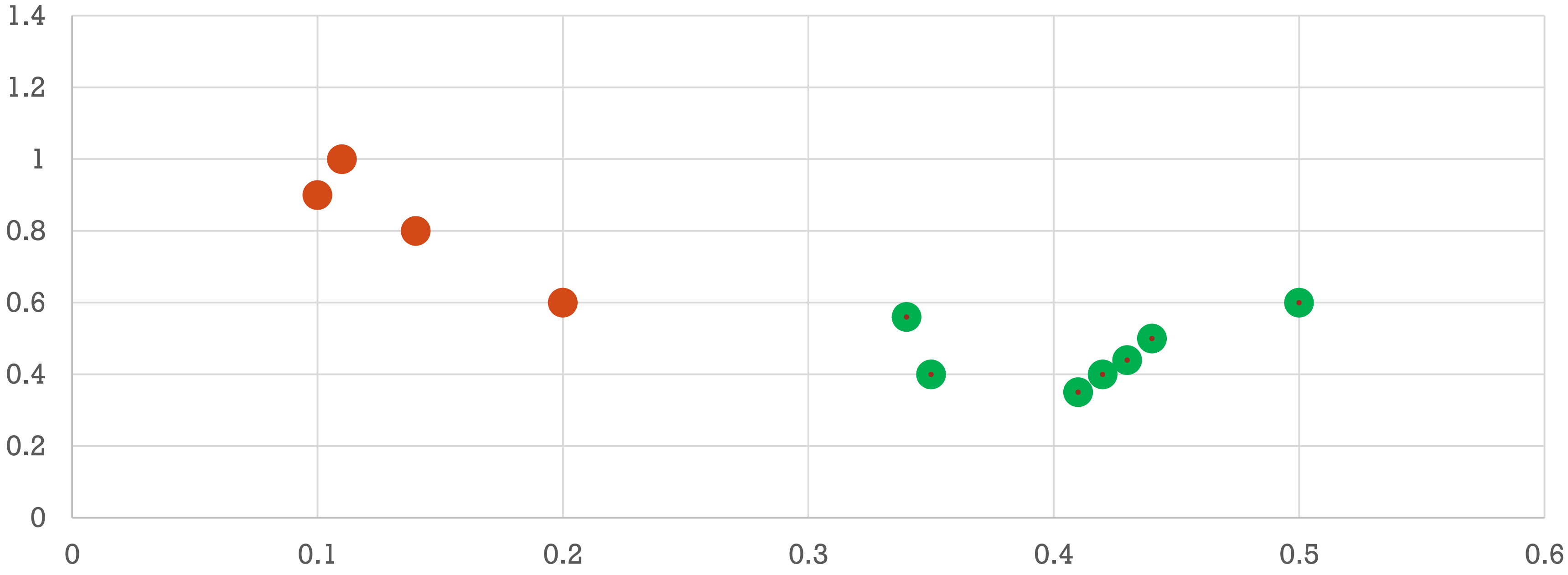
$$X = \{x_i | i \in [n], x \in \mathbb{R}^d\}, Y = \{y_i | i \in [n]\}$$

We want to find a transformation:

$$f : X \rightarrow \mathbf{Y}$$

s.t.

$$x_i \mapsto_f y_i$$



DEFINITION (1): LINEAR SEPARABILITY

Let $X_0, X_1 \subseteq \mathbb{R}^d$ be 2 sets of points. X_0, X_1 are linearly separable if there exist $n+1$ real numbers $w_1, w_2, \dots, w_n, k$ such that:

$$\forall x \in X_0: \sum_{i=1}^{n} w_i x_i > k$$

$$\forall x \in X_1: \sum_{i=1}^{n} w_i x_i < k$$

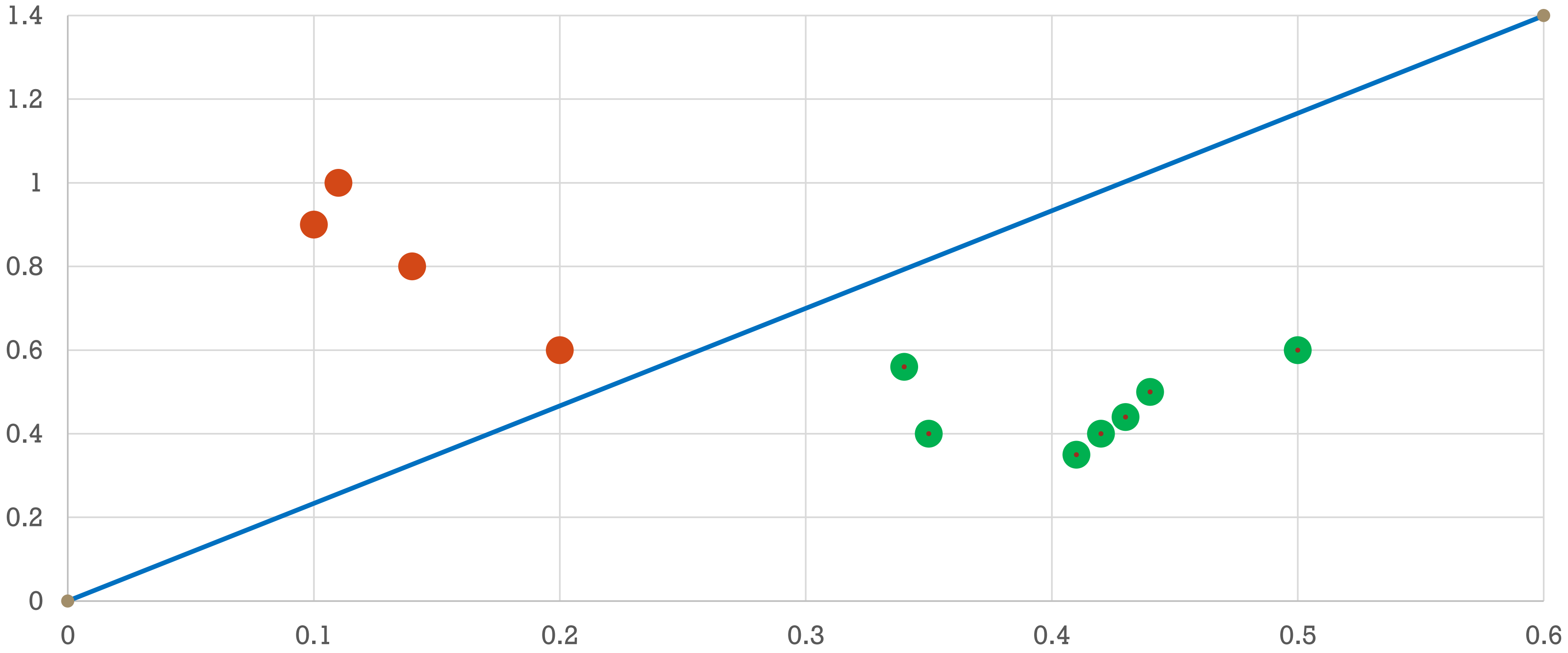
The above terms could also be represented as inner product: $\langle w, x \rangle$ where: $w = (w_1, w_2, .., w_n)$ and $x = (x_1, x_2, \dots, x_n)$

DEFINITION 1 INTERPRETATION

Given vector w, we can define a hyper-plane by:

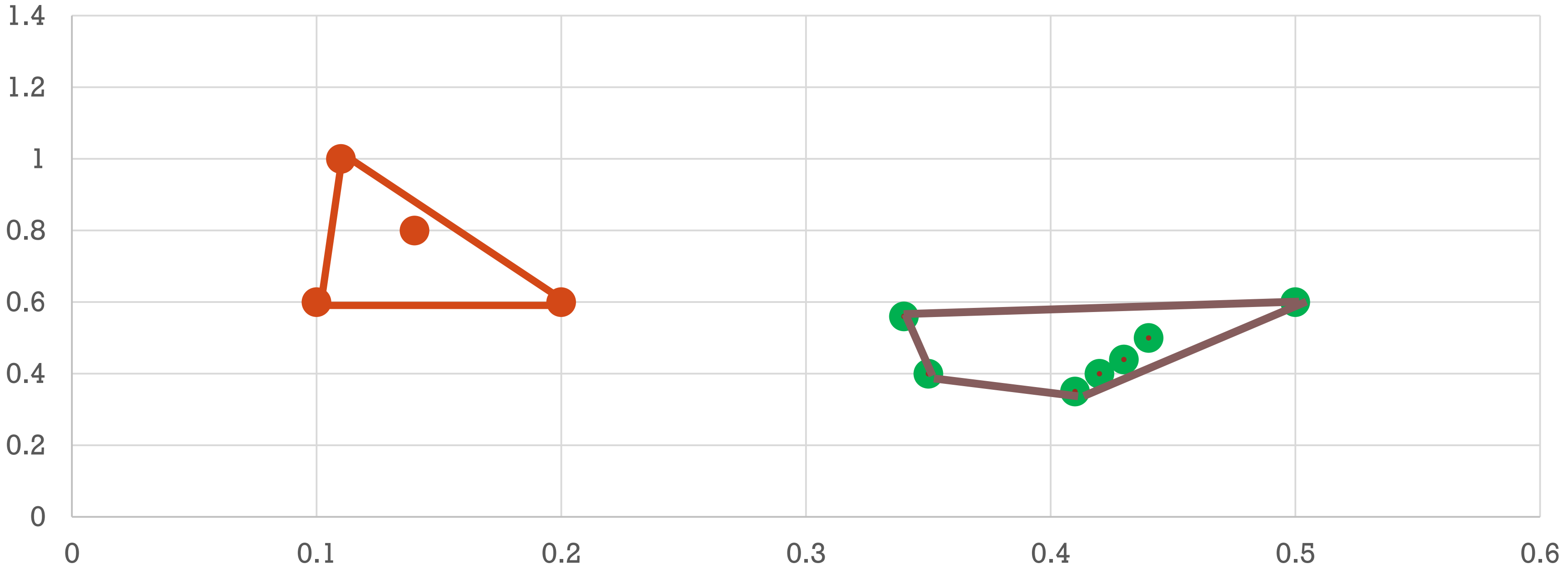
$$\langle w, x \rangle + d = 0$$

Thus, the hyper-plane separates the field into 2 regions such that all points belong to X_0 are in one region and all points belong to X_1 are in the other region.



DEFINITION (2): LINEAR SEPERABILITY

Let $X_0, X_1 \subseteq \mathbb{R}^d$ be 2 sets of points. X_0, X_1 are linearly separable precisely when their respective convex hulls are disjoint (do not overlap)



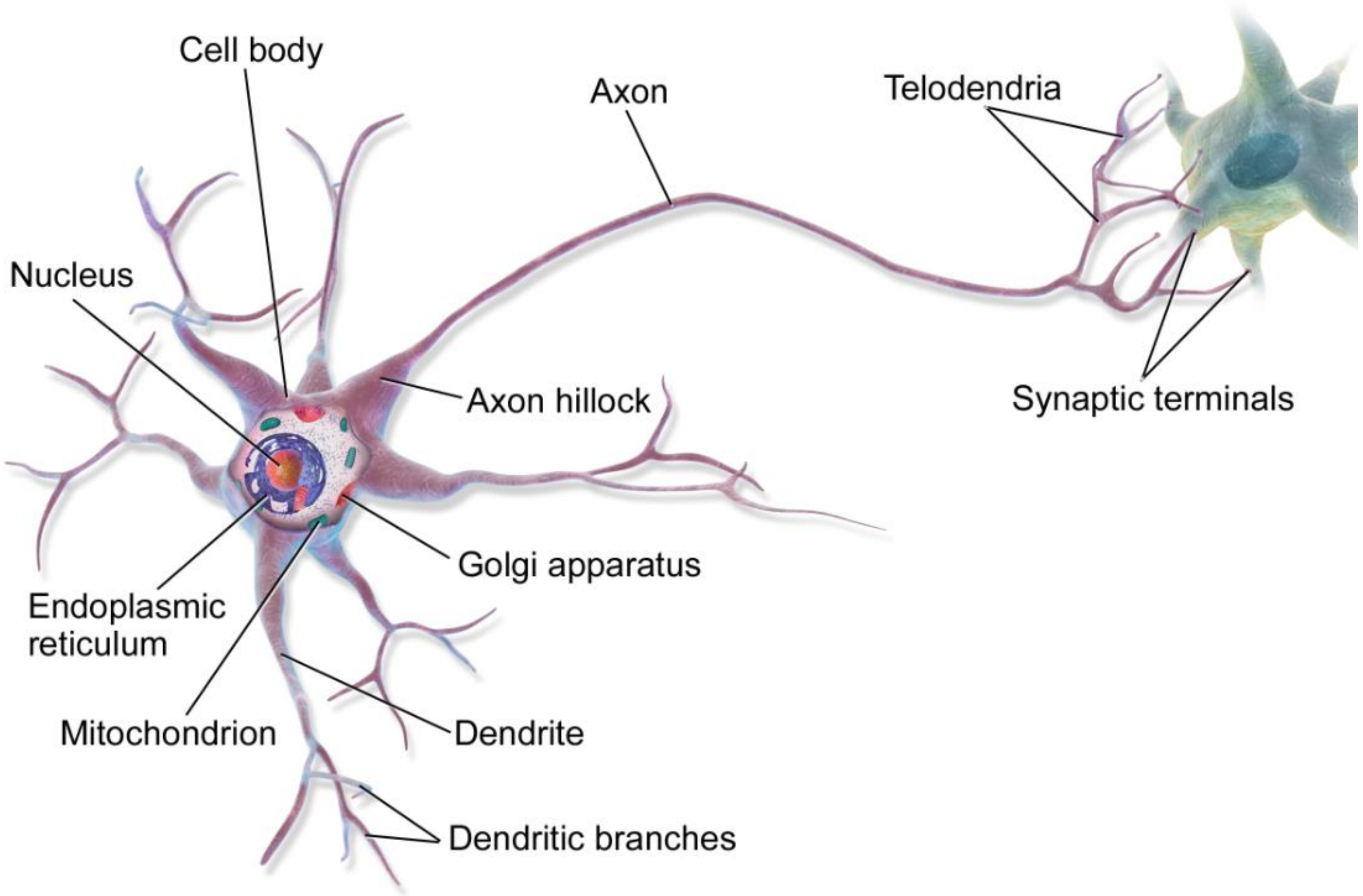
AGENDA

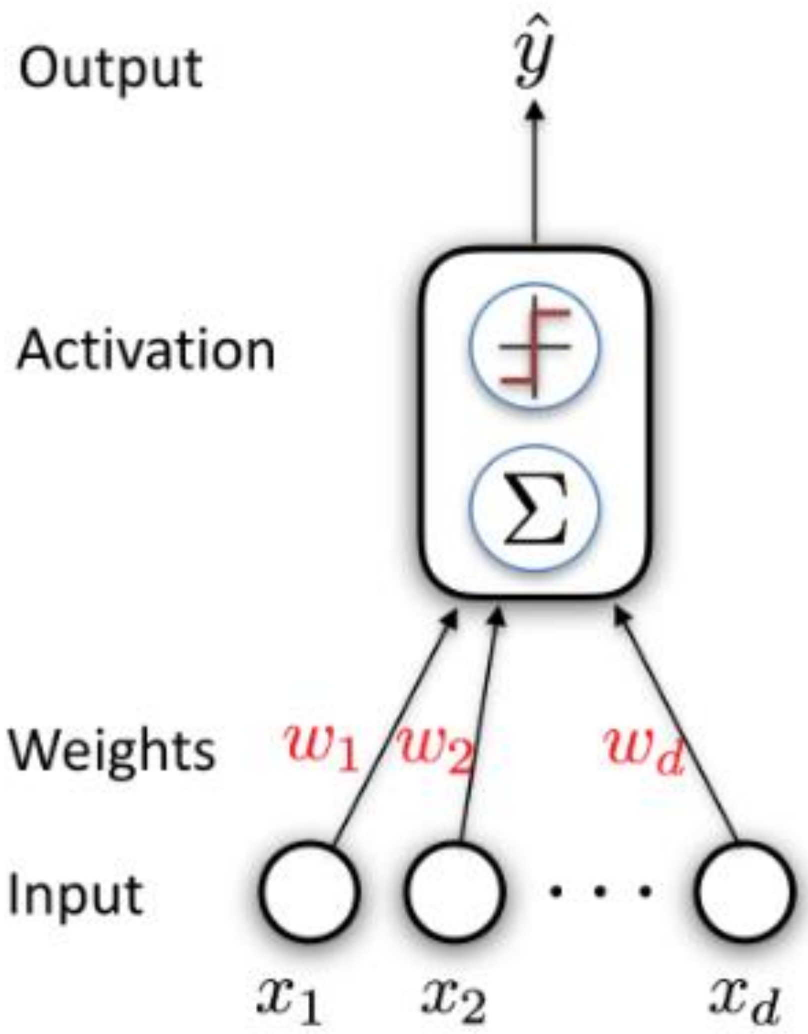
- The Problem
  - Linear Seperability – Def. 1
  - Linear Seperability – Def. 2
- The Biological Neuron
  - Structure
  - A Mathematical model
- The Solution – Perceptron
  - A brief history
  - The algorithm
  - Intuitive interpretation for weights update
  - Theorem: Mistake bound
  - The fall of perceptron
  - Stopping criterions
  - Kernel Perceptron

THE BIOLOGICAL NEURON - STRUCTURE

Like any other body cell, the neuron has a cell body which contains a nucleus where the DNA is stored.
From our perspective, the interesting parts are:

- Dendrites – make connections with tens of thousand of other cells; other neurons. The behave as "inputs".
- Axon – transmits information to different neurons, muscles, and other body cells based on the signals the cell receives. It's signals are received by other cells' dendrites.





THE BIOLOGICAL NEURON – A MATHEMATICAL MODEL

- We will try to mimic the function of a neuron using mathematical tools. Given an input vector x:
  - x will be the inputs of the neuron (dendrites).
  - Define a weight, w_i, for each input, and sum all the multiplications.
  - Output the result as $\hat{y}$ (Axon)

There's still a problem – How do we find the weights?

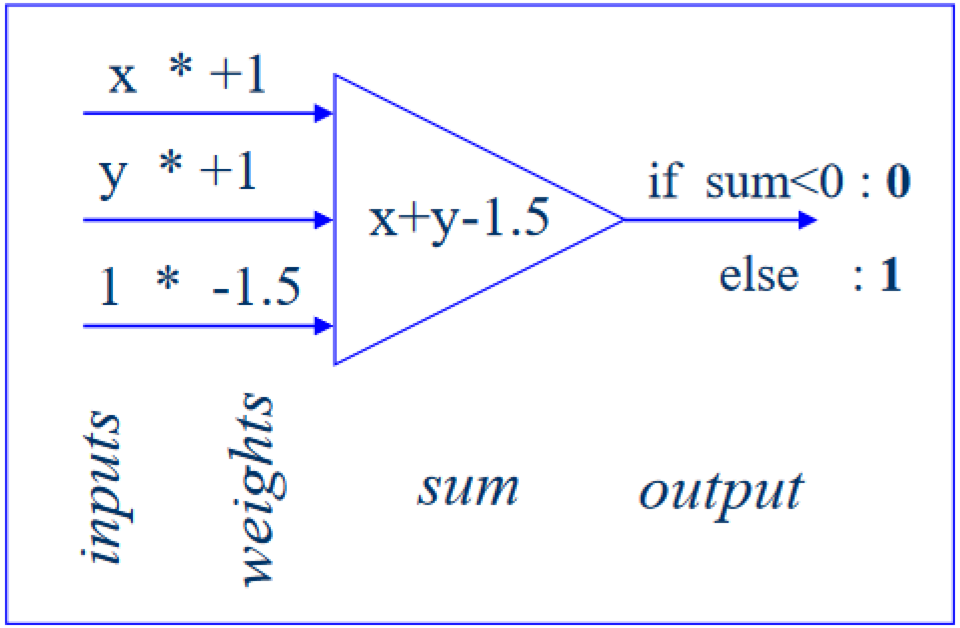
AGENDA

- The Problem
  - Linear Seperability – Def. 1
  - Linear Seperability – Def. 2
- The Biological Neuron
  - Structure
  - A Mathematical model
- The Solution – Perceptron
  - A brief history
  - The algorithm
  - Intuitive interpretation for weights update
  - Theorem: Mistake bound
  - The fall of perceptron
  - Stopping criterions
  - Kernel Perceptron

PREHISTORY

Because of the "all-or-none" character of nervous activity, neural events and the relations among them can be treated by means of propositional logic.

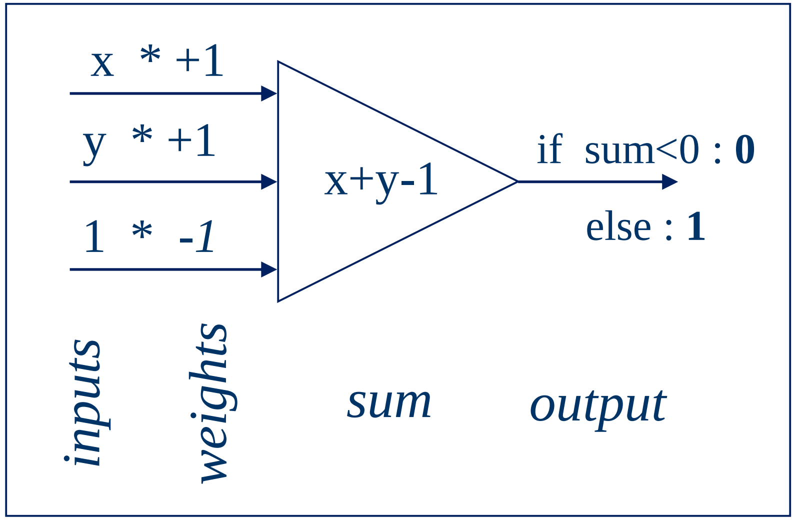
- W.S. McCulloch & W. Pitts (1943). "A logical calculus of the ideas immanent in nervous activity", Bulletin of Mathematical Biophysics, 5, 115-137
- This seminal paper pointed out that simple artificial "neurons" could be made to perform basic logical operations such as AND, OR and NOT



Truth Table for Logical AND

x	y	x & y
0	0	0
0	1	0
1	0	0
1	1	1

inputs *output*



Truth Table for Logical OR

x	y	x | y
0	0	0
0	1	1
1	0	1
1	1	1

inputs *output*

1958 – THE PERCEPTRON

Psychological Review
Vol. 65, No. 6, 1958

THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN[1]

F. ROSENBLATT

Cornell Aeronautical Laboratory

NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

WASHINGTON, July 7 (UPI) —The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

PERCEPTRON – THE ALGORITHM

- The goal is to find a hyper-plane separating 2 <u>known</u> classes.
- Consider definition (1) for linear separability:

$$\forall x \in X_0 : \langle w, x \rangle > \mathrm{k}$$

$$\forall x \in X_1 : \langle w, x \rangle < \mathrm{k}$$

$$\Downarrow$$

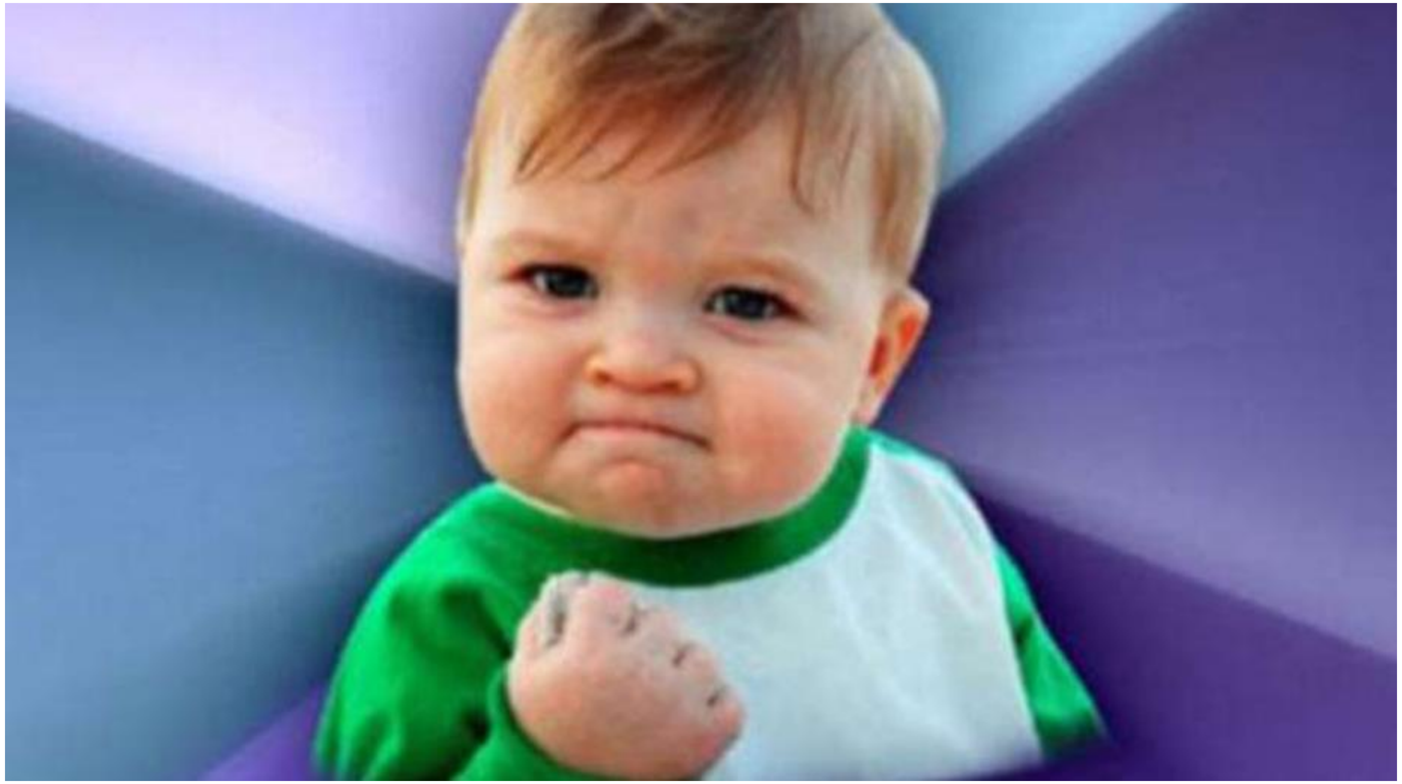
$$\forall x \in X_0 : \langle w, x \rangle - \mathrm{k} > 0$$

$$\forall x \in X_1 : \langle w, x \rangle - k < 0$$

We can eliminate k by augmenting representation with one dimension:

$$x' = (x, 1)$$
$$w' = (w, -k)$$

$$\langle w', x' \rangle = (w, -k) \binom{x}{1} = w \cdot x - k$$



Algorithm: Perceptron Learning Algorithm

$P \leftarrow inputs \quad with \quad label \quad 1;$

$N \leftarrow inputs \quad with \quad label \quad 0;$

Initialize $\mathbf{w}$ randomly;

while !*convergence* **do**

  Pick random $\mathbf{x} \in P \cup N$;

  if $\mathbf{x} \in P \quad and \quad \mathbf{w}.\mathbf{x} < 0$ **then**

    $\mathbf{w} = \mathbf{w} + \mathbf{x}$;

  end

  if $\mathbf{x} \in N \quad and \quad \mathbf{w}.\mathbf{x} \geq 0$ **then**

    $\mathbf{w} = \mathbf{w} - \mathbf{x}$;

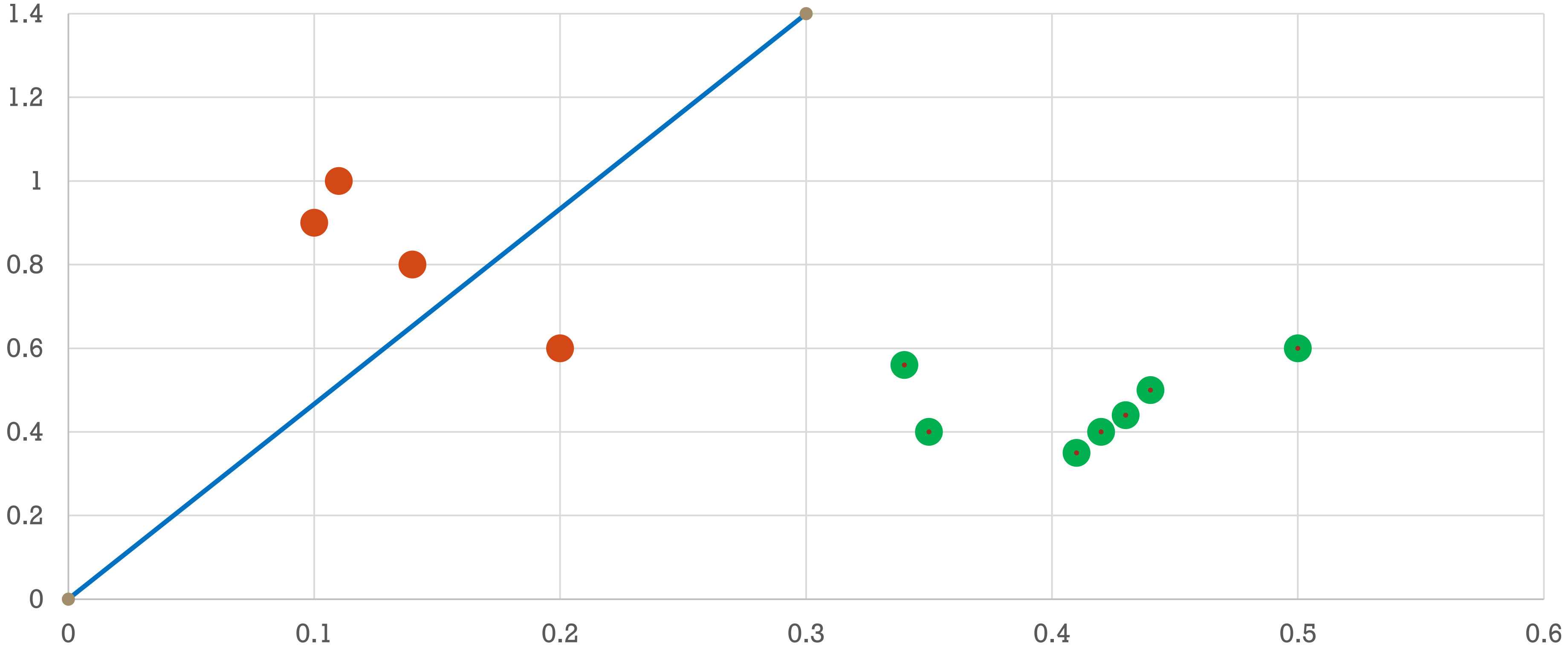
  end

end

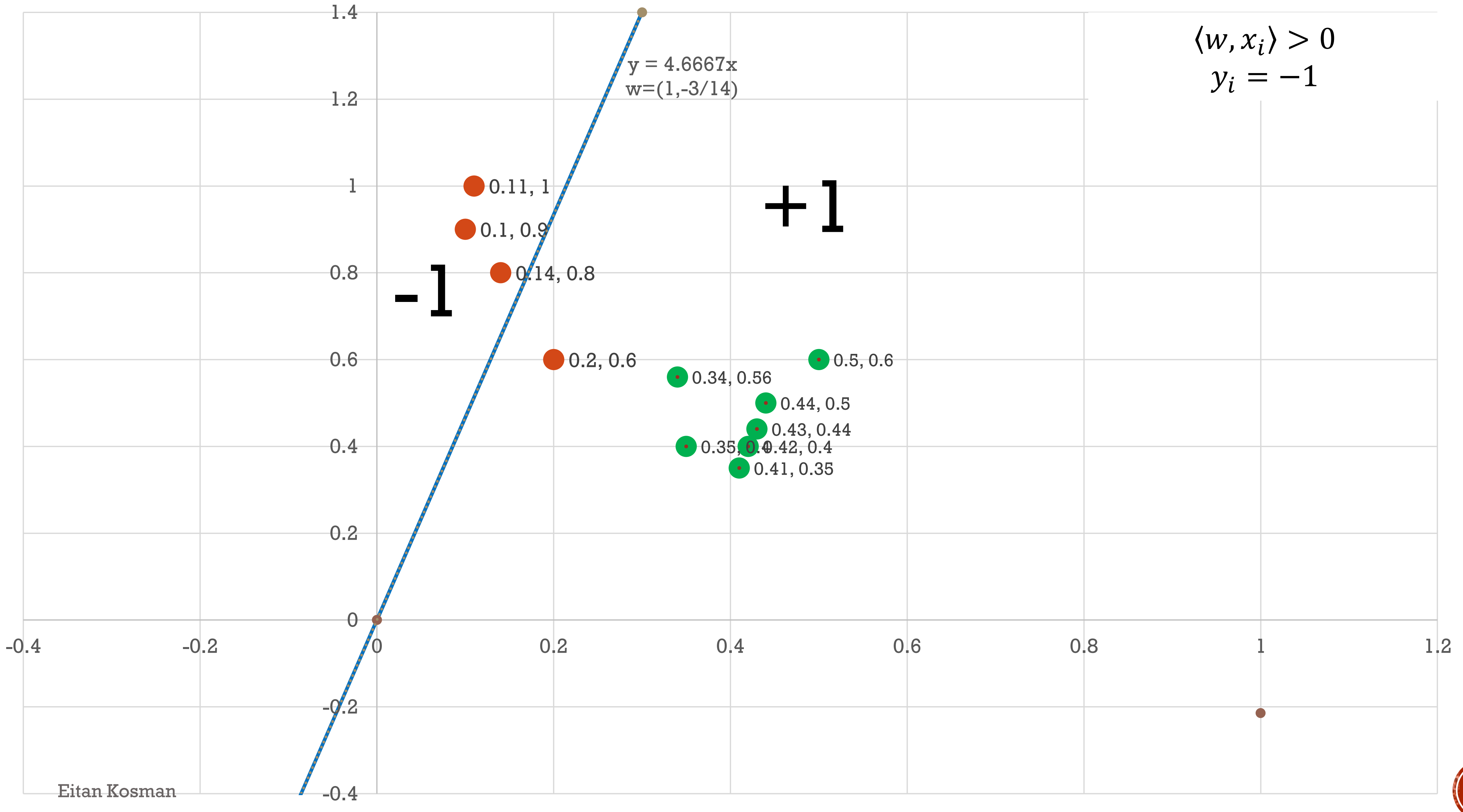
//the algorithm converges when all the inputs are classified correctly

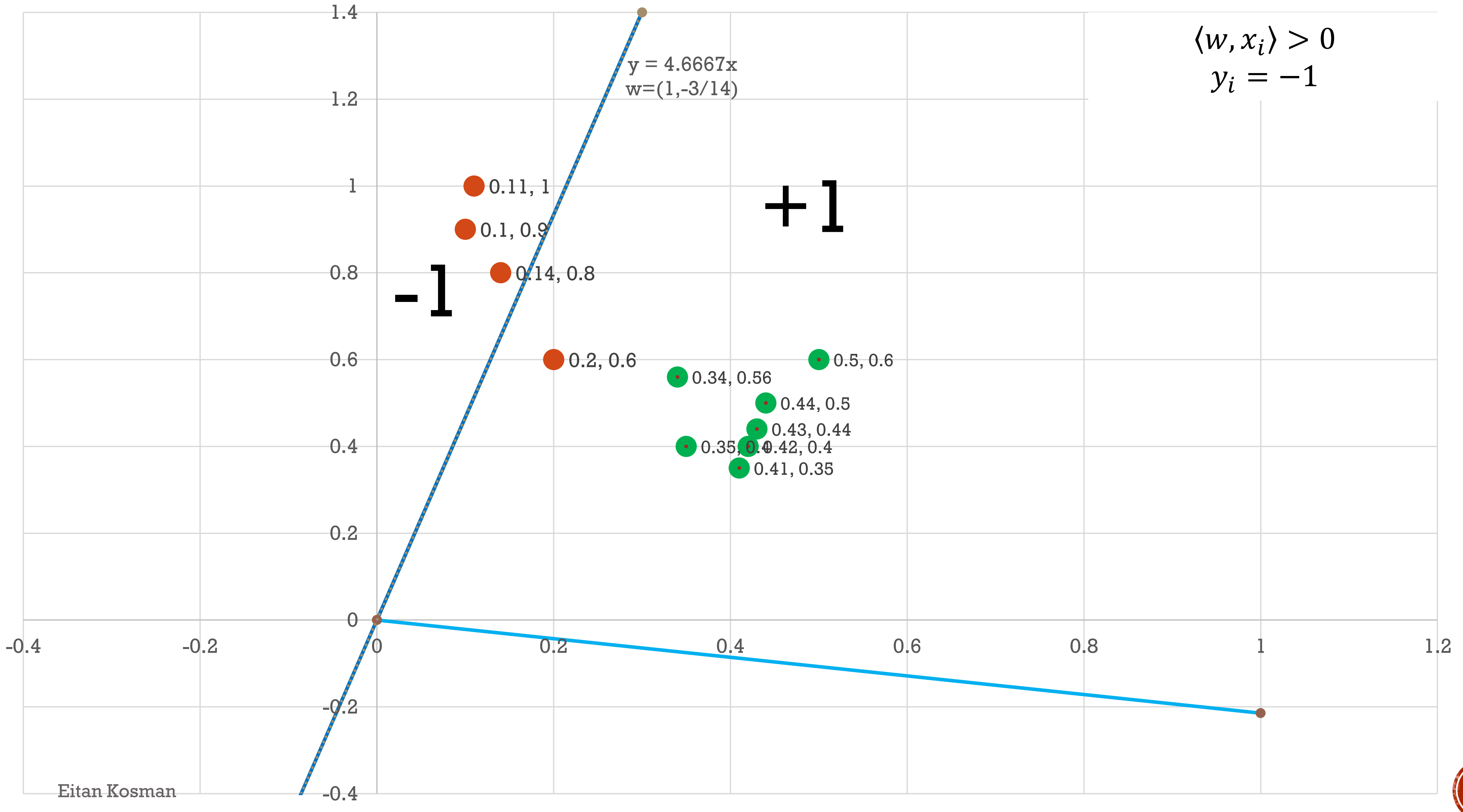
WEIGHTS UPDATE : INTUITION

The orange points are from class -1 and the green points are from class +1.
How would we update the decision line so that it classifies all the points correctly?

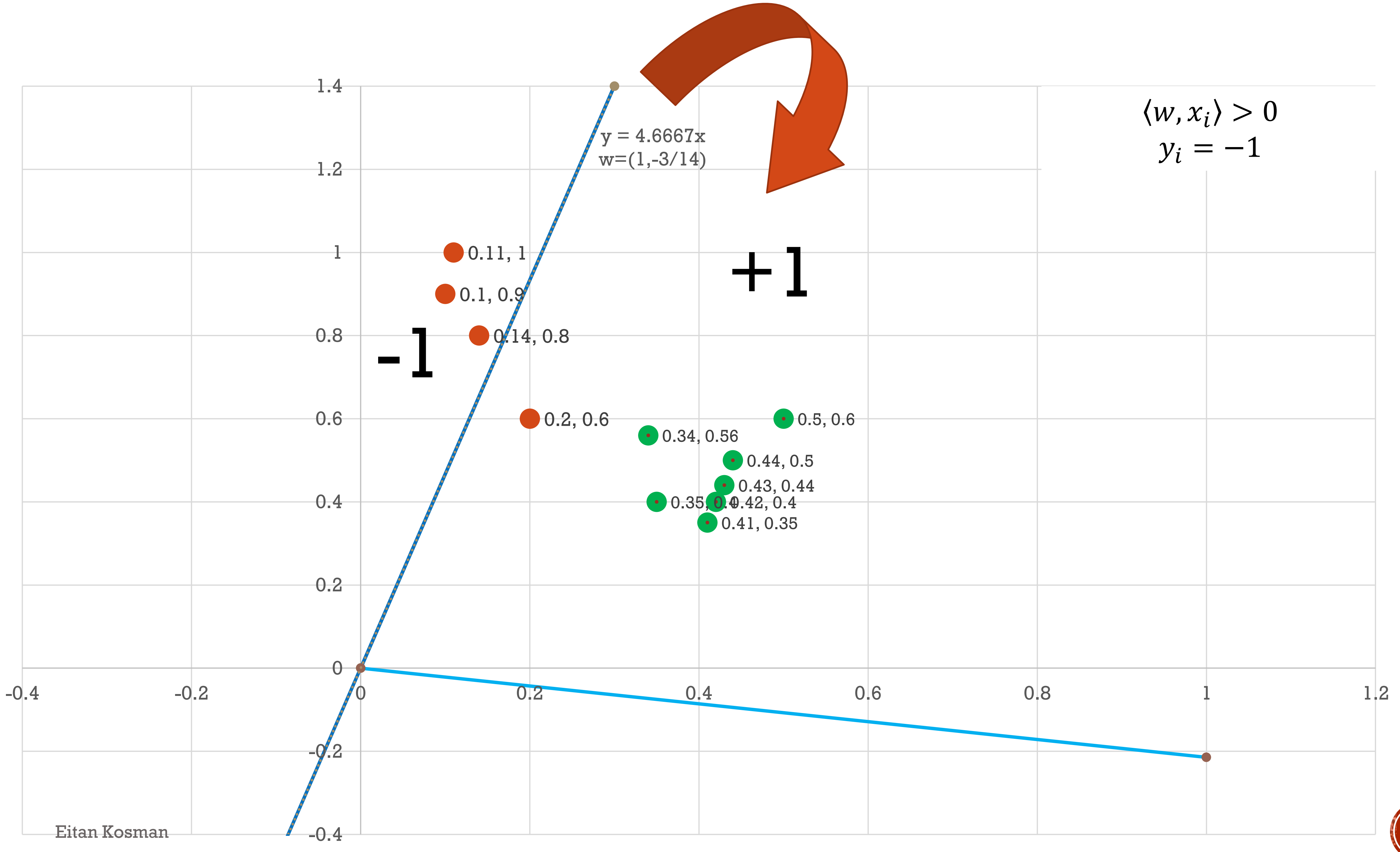


$$\langle w, x_i \rangle > 0$$
$$y_i = -1$$



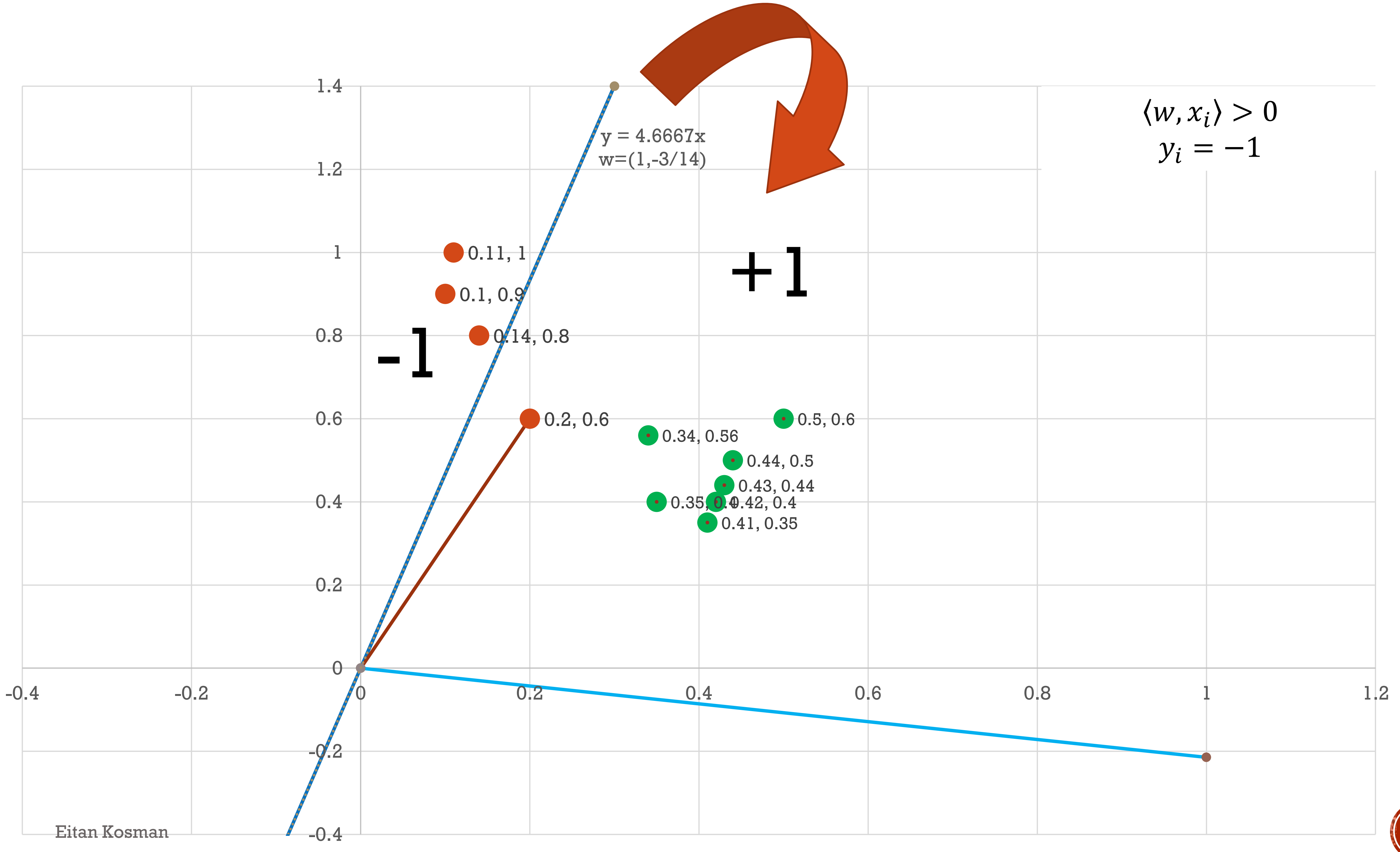


Eitan Kosman



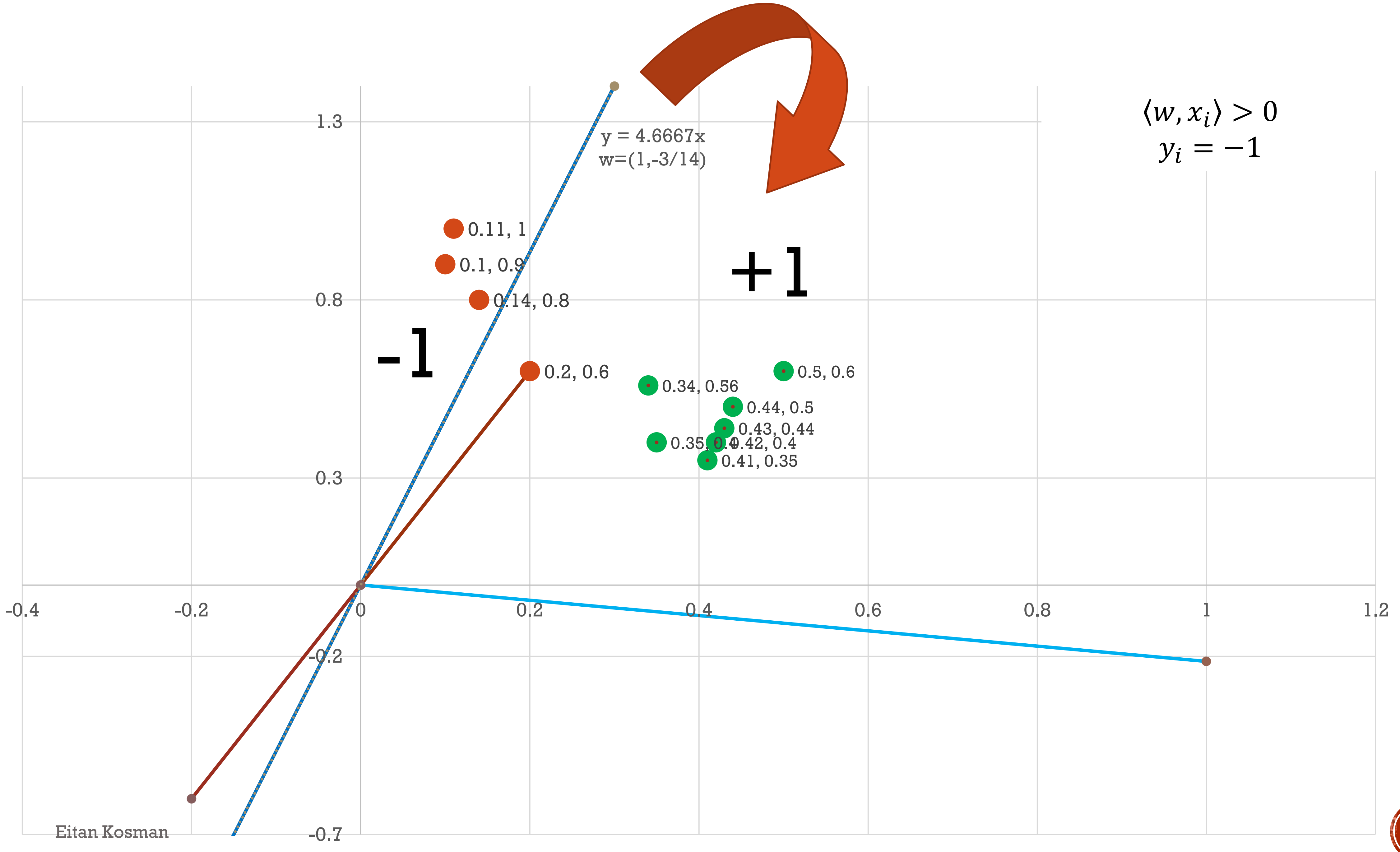
$\langle w, x_i \rangle > 0$

$y_i = -1$



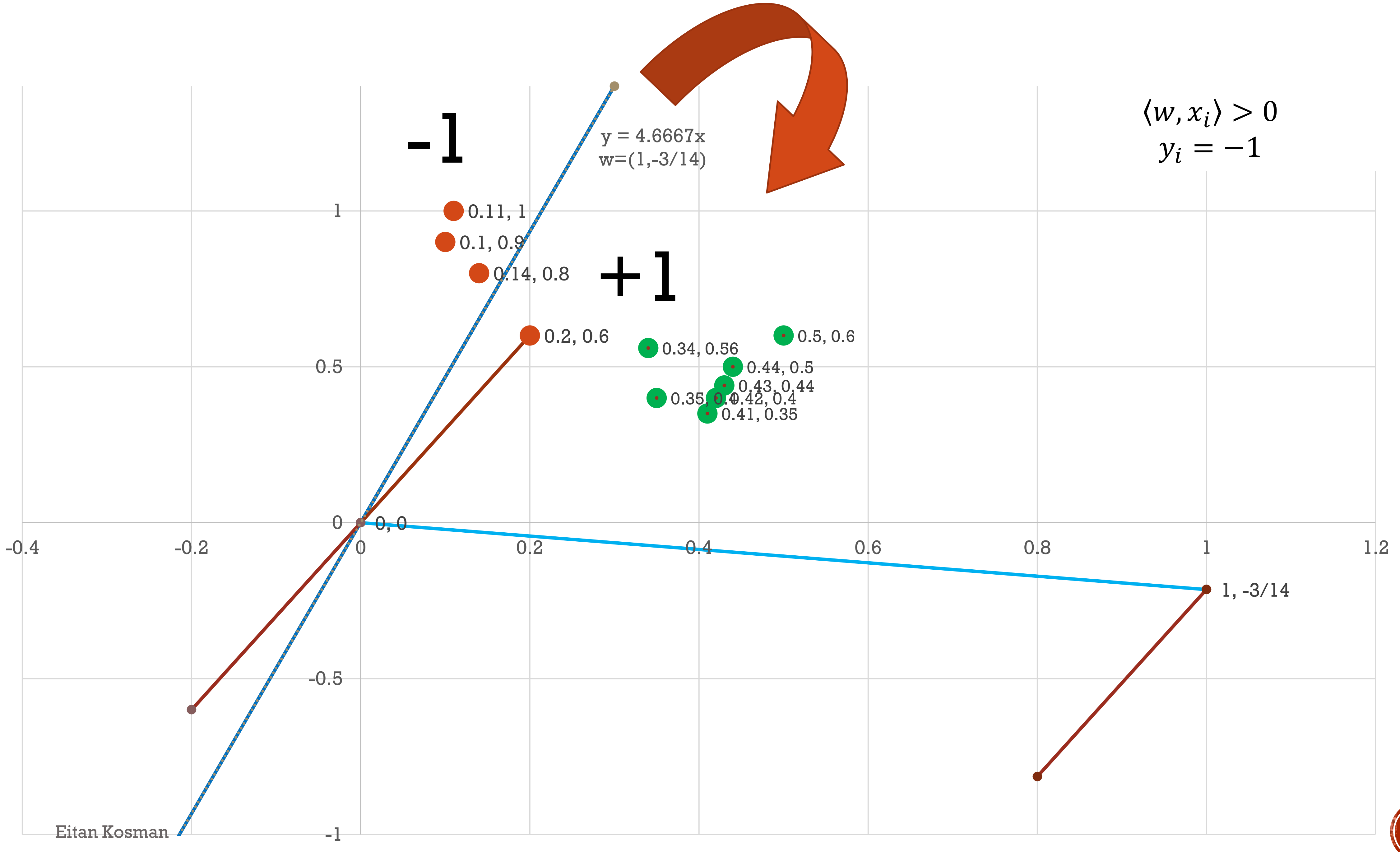
$$\langle w, x_i \rangle > 0$$
$$y_i = -1$$

Eitan Kosman

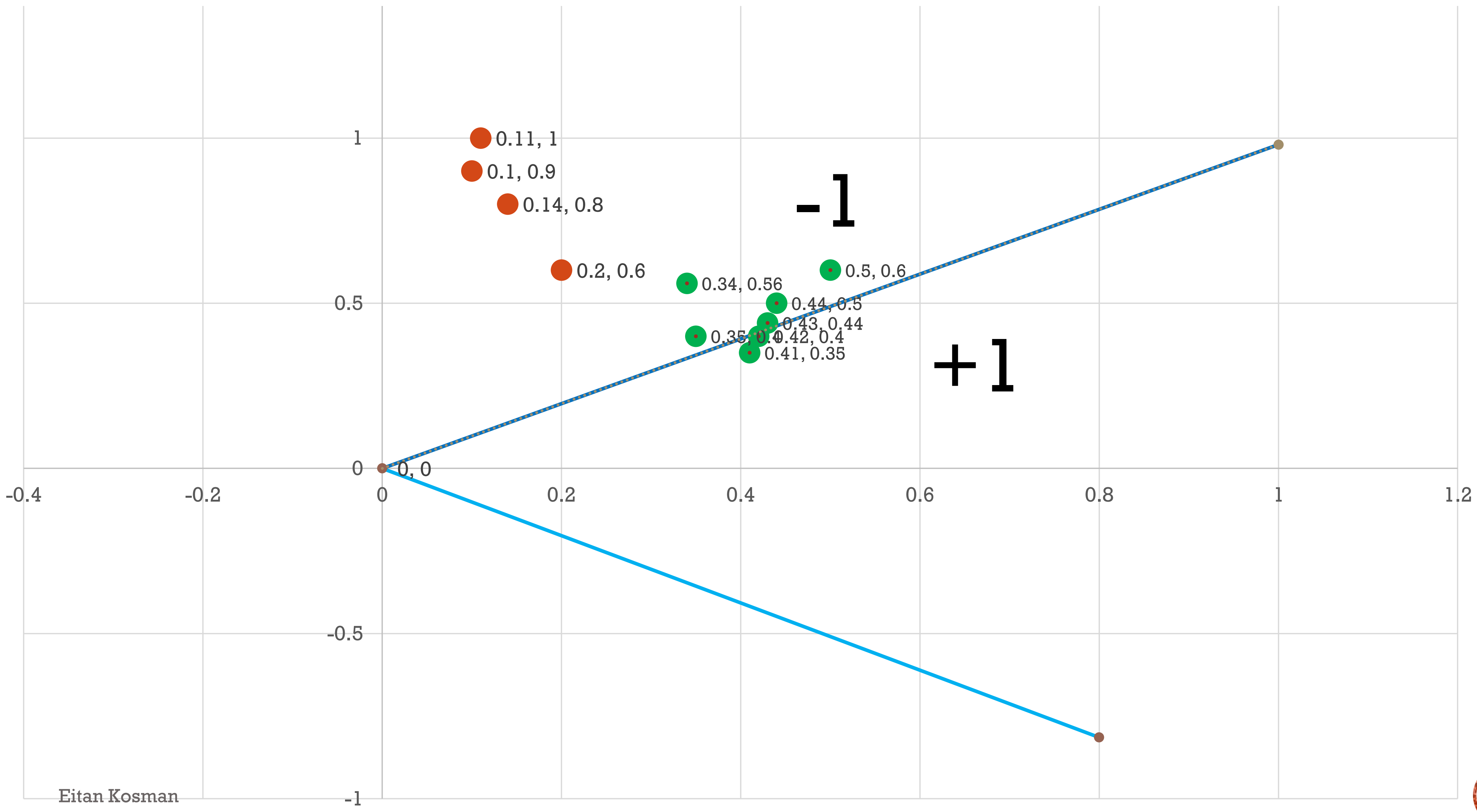


$\langle w, x_i \rangle > 0$
$y_i = -1$

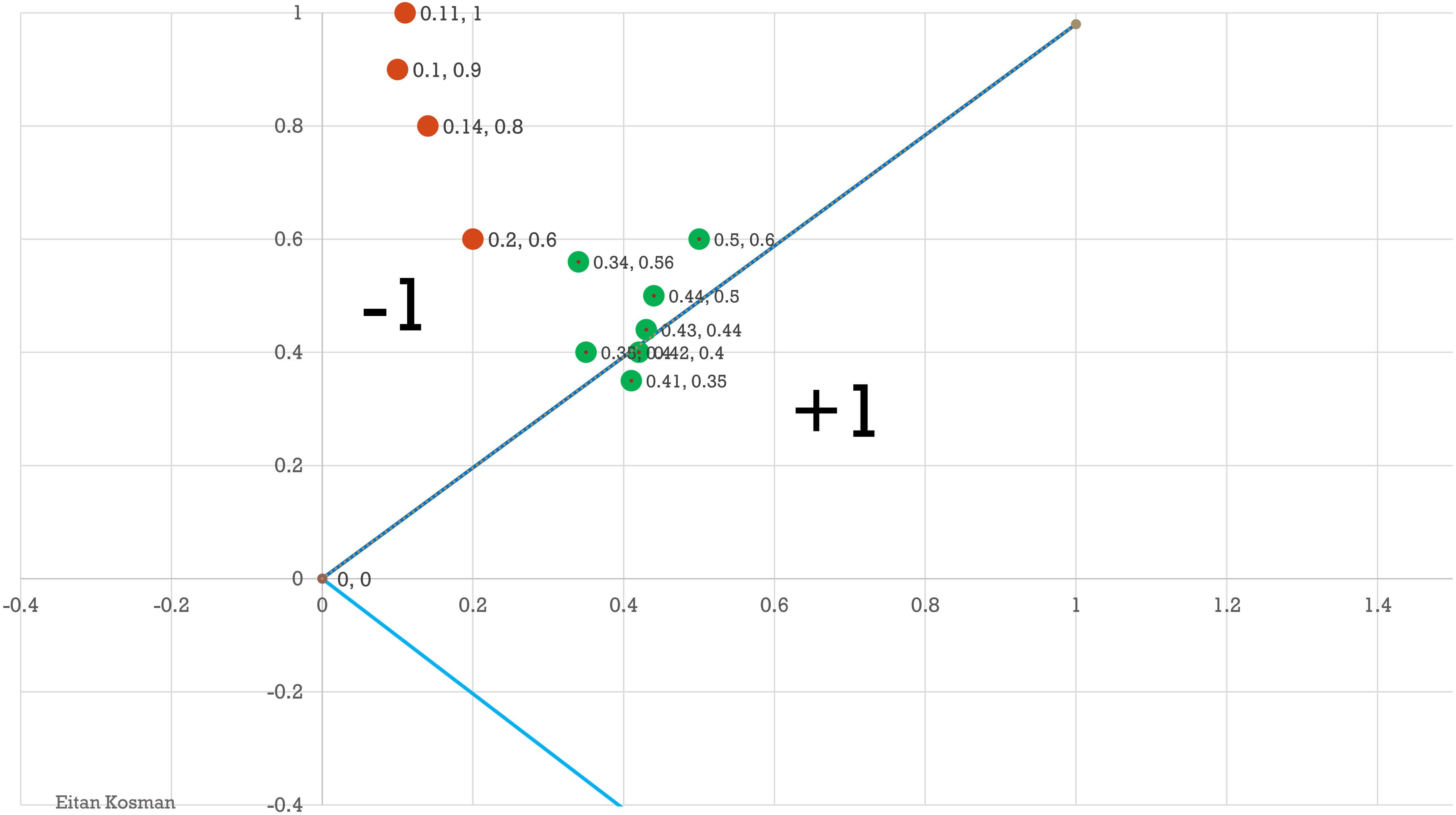
Eitan Kosman



$$\langle w, x_i \rangle > 0$$
$$y_i = -1$$



0.11, 1
0.1, 0.9
0.14, 0.8
0.2, 0.6
0.34, 0.56
0.5, 0.6
0.44, 0.5
0.43, 0.44
0.35, 0.4
0.42, 0.4
0.41, 0.35
0, 0
-1
+1
1
0.5
0
-0.5
-1
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1
1.2



0.11, 1
0.1, 0.9
0.14, 0.8
0.2, 0.6
0.34, 0.56
0.5, 0.6
0.44, 0.5
0.43, 0.44
0.35, 0.4
0.42, 0.4
0.41, 0.35
0, 0
-1
+1
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1
1.2
1.4

MISTAKE BOUND

Theorem:

Let $(x_1, y_1), \dots, (x_n, y_n)$, where $x_i \in \mathbb{R}^N$ and $y_i \in \{-1,1\}$ be a sequence of labeled examples and assume it is linearly separable.

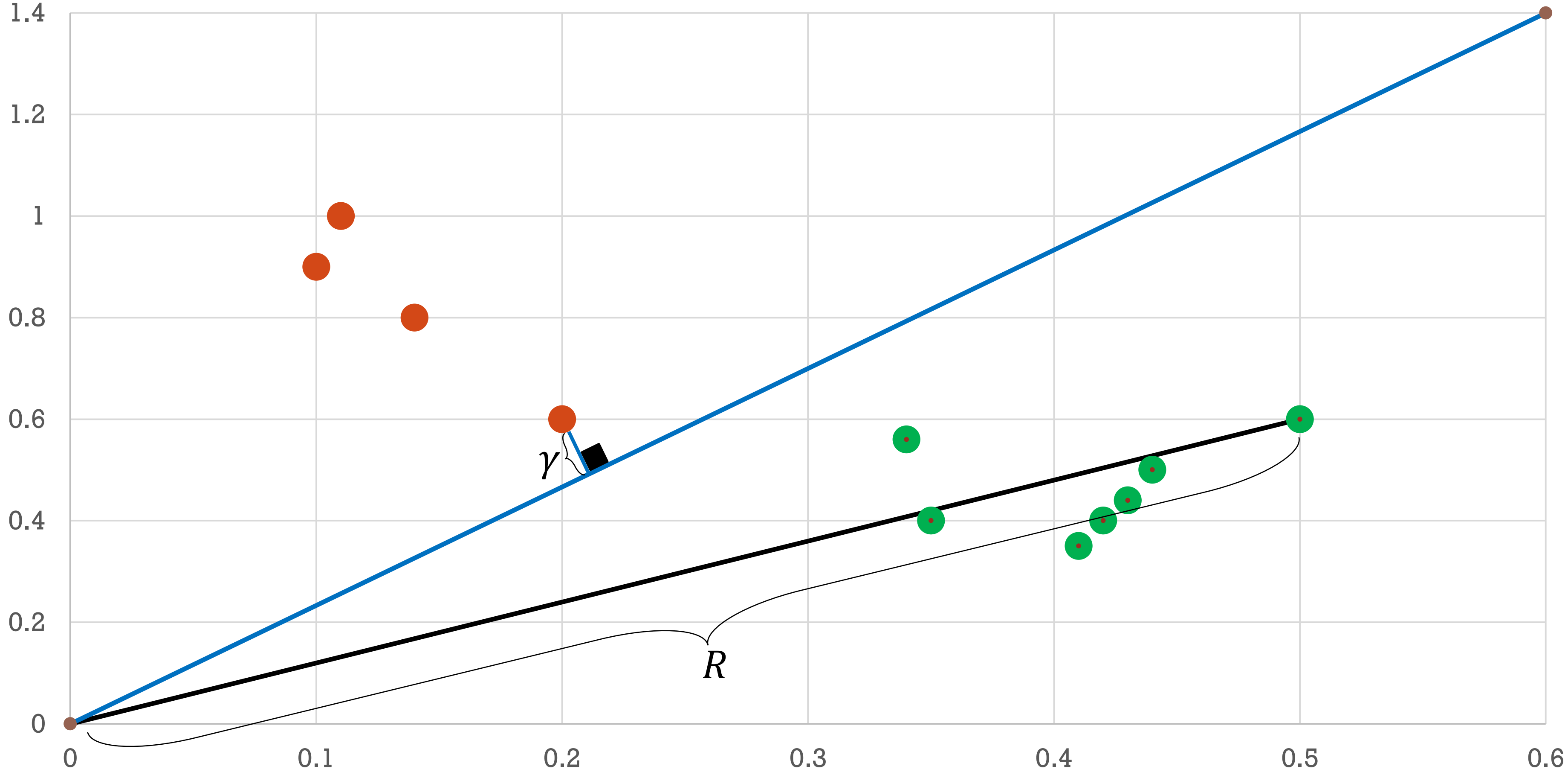
Denote:

$$R = \max_i ||x_i||$$

Suppose there exists a vector $w^*, \gamma > 0$ such that $||w^*|| = 1$ and $\forall i, y_i\left({w^*}^T x_i\right) \geq \gamma$, then the number of mistakes made by the Perceptron algorithm of this sequence of example is $O\left(\left(\frac{R}{\gamma}\right)^2\right)$

$$R = \max_i ||x_i||$$

$$\forall i, y_i\left(w^{*T} x_i\right) \geq \gamma$$



MISTAKE BOUND

Let $w_1 = 0$ (initial weight vector) and denote w_k the weight vector after the $k'th$ mistake.

<u>Lemma 1:</u> $w_{t+1} \cdot w^* \geq w_t \cdot w^* + \gamma$

<u>Lemma 2:</u> $||w_{t+1}||^2 \leq ||w_t||^2 + R^2$

MISTAKE BOUND

<u>Lemma 1:</u> $w_{t+1} \cdot w^* \geq w_t \cdot w^* + \gamma$

The $t's$ update occurred when the perceptron did a mistake on sample (x_i, y_i).

If $y_i = 1$:

$$w_{t+1} \cdot w^* = (w_t + x_i) \cdot w^* = w_t \cdot w^* + \underbrace{x_i \cdot w^*}_{\geq \gamma} = w_t \cdot w^* + \gamma$$

If $y_i = -1$:

$$w_{t+1} \cdot w^* = (w_t - x_i) \cdot w^* = w_t \cdot w^* - \underbrace{x_i \cdot w^*}_{\geq \gamma} = w_t \cdot w^* + \gamma$$

MISTAKE BOUND

<u>Lemma 2:</u> $||w_{t+1}||^2 \leq ||w_t||^2 + R^2$

The $t's$ update occurred when the perceptron did a mistake on sample (x_i, y_i).

If $y_i = 1$:

$$||w_{t+1}||^2 = ||w_t + x_i||^2 = ||w_t||^2 + 2\underbrace{w_t \cdot x_i}_{\substack{<0, since \\ a\ mistake \\ has\ occured}} + \underbrace{||x_i||^2}_{\leq R^2} \leq ||w_t||^2 + R^2$$

If $y_i = -1$:

$$||w_{t+1}||^2 = ||w_t - x_i||^2 = ||w_t||^2 - 2\underbrace{w_t \cdot x_i}_{\substack{>0, since \\ a\ mistake \\ has\ occured}} + \underbrace{||x_i||^2}_{\leq R^2} \leq ||w_t||^2 + R^2$$

MISTAKE BOUND

Now, equipped with the two lemmas, we know that from Lemma 1:

$$w_1 = \bar{0}$$
$$w_2 \cdot w^* \geq w_1 \cdot w^* + \gamma = \gamma$$
$$w_3 \cdot w^* \geq w_2 \cdot w^* + \gamma \geq \gamma + \gamma = 2\gamma$$

<u>Assume:</u> $w_t \cdot w^* \geq (t-1) \cdot \gamma$

Thus –

$$w_{t+1} \cdot w^* \geq w_t \cdot w^* + \gamma \geq (t-1) \cdot \gamma + \gamma = t \cdot \gamma$$

Moreover, from lemma 2:

$$|w_1| = 0$$
$$|w_2|^2 \leq |w_1|^2 + R^2 = R^2$$
$$|w_3|^2 \leq |w_2|^2 + R^2 \leq R^2 + R^2 = 2R^2$$

<u>Assume:</u> $|w_t|^2 \leq (t-1)R^2$

Thus –

$$|w_{t+1}|^2 \leq |w_t|^2 + R^2 \leq (t-1)R^2 + R^2 = tR^2$$

MISTAKE BOUND

<u>Recap:</u>

After T mistakes:

$$w_{T+1} \cdot w^* \geq T \cdot \gamma$$

$$|w_{T+1}|^2 \leq TR^2$$

$$\Downarrow$$

$$\gamma T \leq \underbrace{w_{T+1} \cdot w^*}_{scalar} = |w_{T+1} \cdot w^*| \underbrace{\leq}_{\substack{Cauchy \\ Schwarz}} |w_{T+1}| \cdot \underbrace{|w^*|}_{=1} = |w_{T+1}|$$

$$\Downarrow$$

$$\gamma^2 T^2 \leq |w_{T+1}|^2 \leq TR^2$$

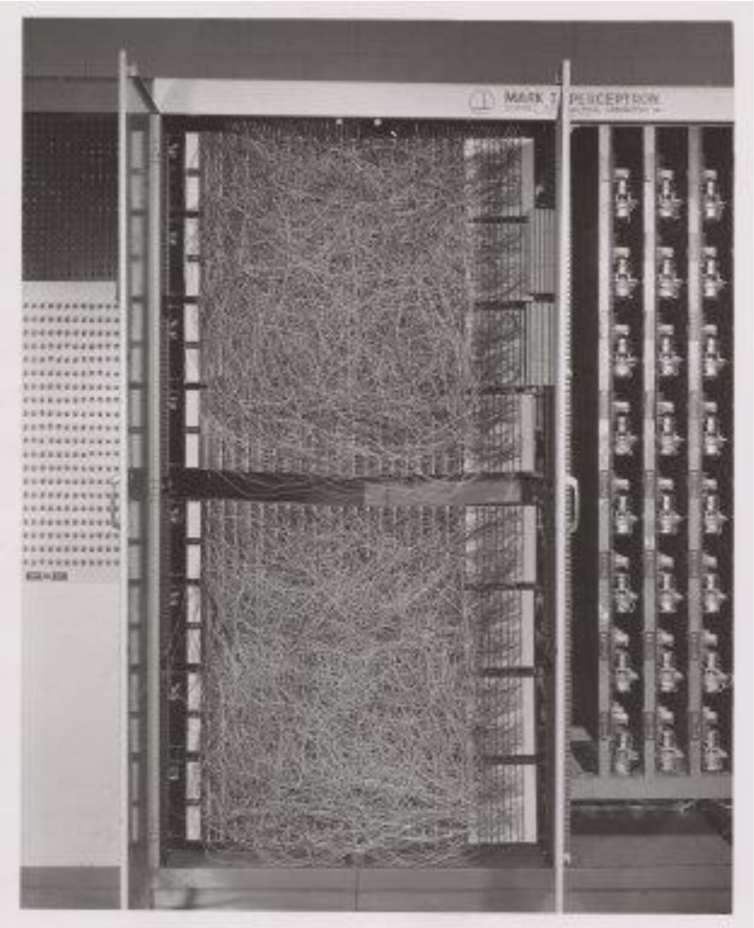
$$\Downarrow$$

$$T \leq \frac{R^2}{\gamma^2}$$

THE FALL OF THE PERCEPTRON

The first computer built around the concept of perceptron looked like this.

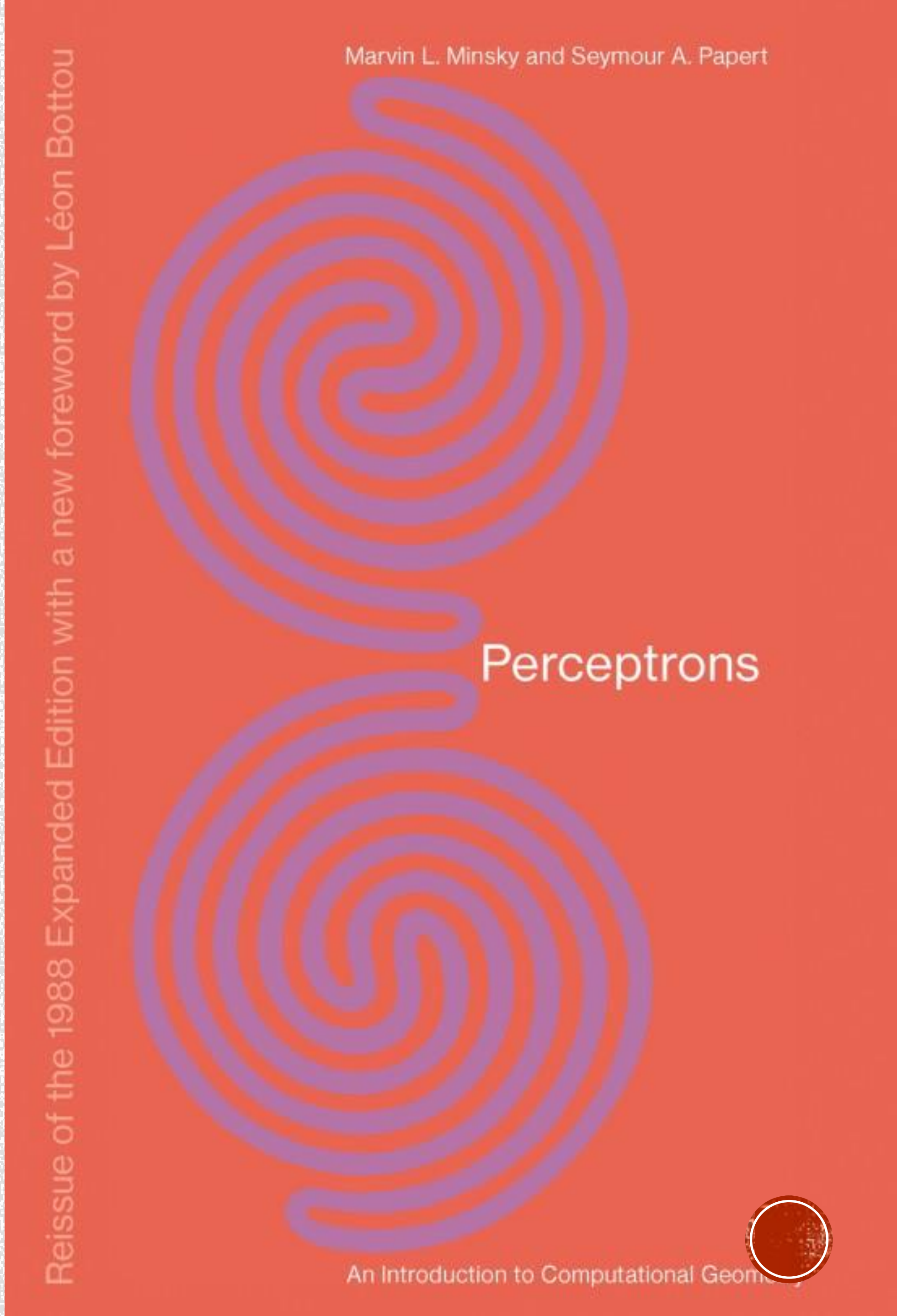
Even the wiring was supposed to simulate the connections of neurons.



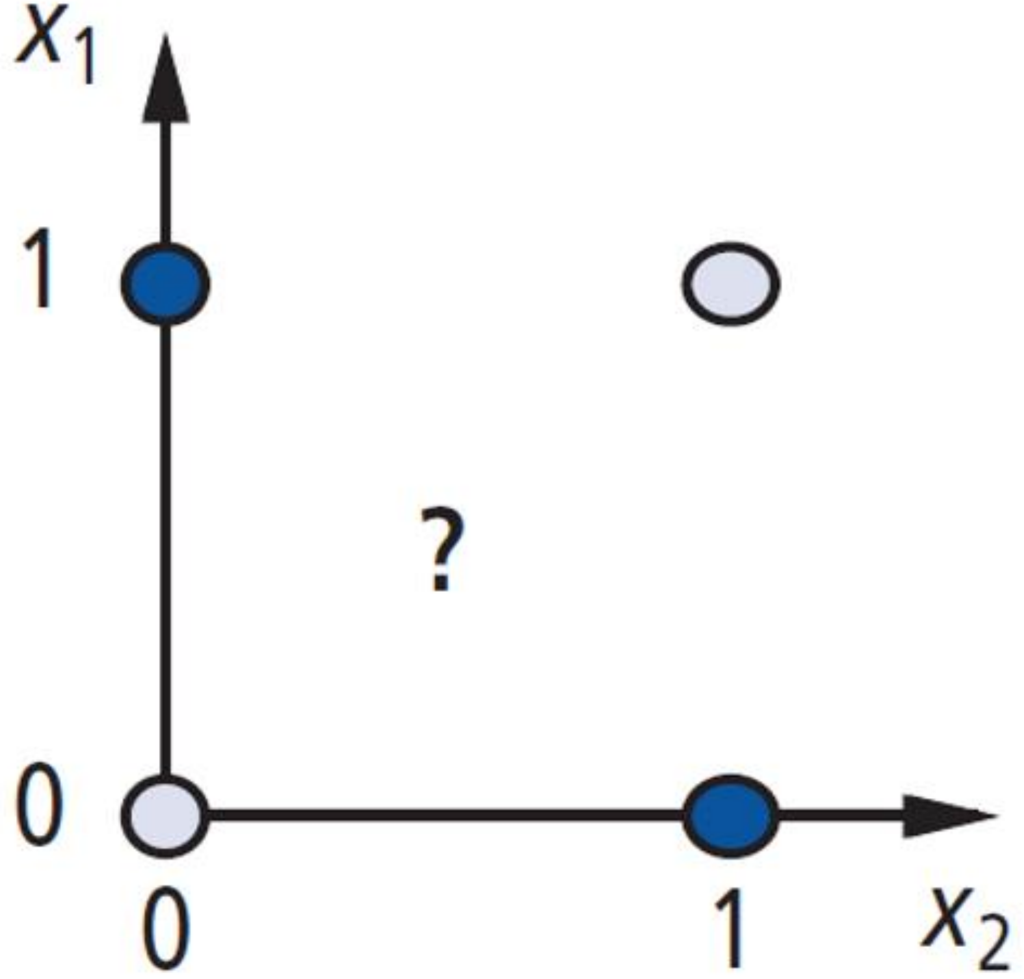
THE FALL OF THE PERCEPTRON

However, a paper describing the perceptron's shortcomings, particularly that it was effective only at solving simple problems, led to a drastic drop in interest in artificial neural networks in the 1960's.

Unless input categories were "linearly separable", a perceptron could not learn to discriminate between them. **Example:**



XOR

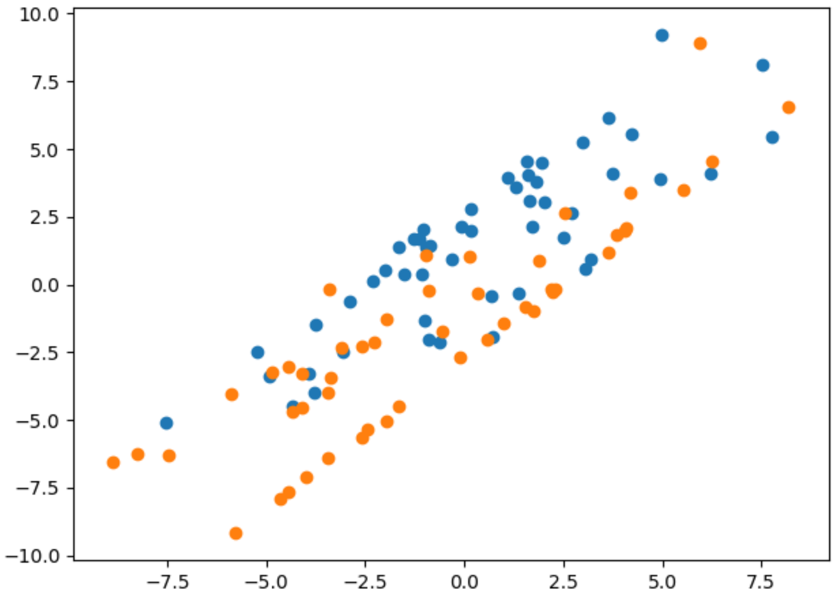


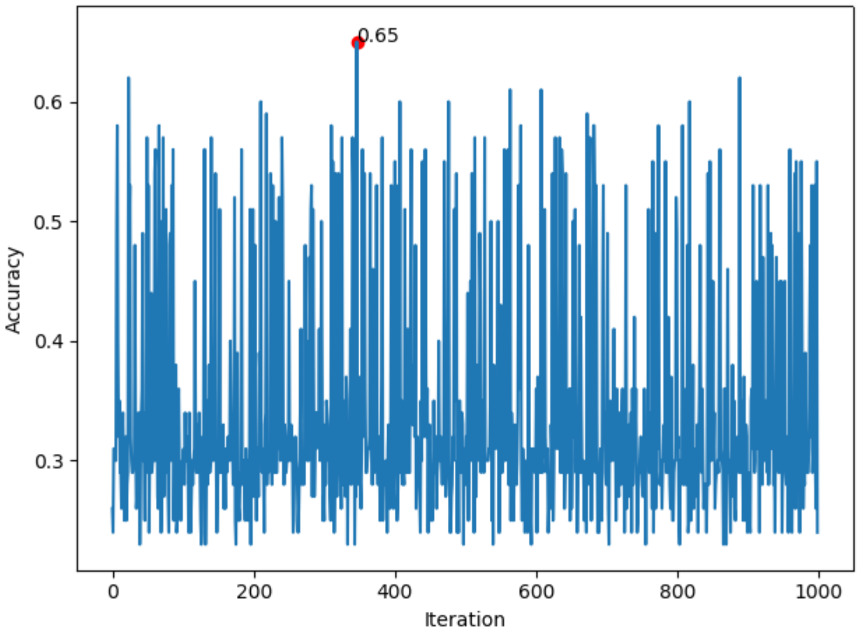
STOPPING CRITERIONS

The mistake bound holds only if the dataset is linearly separable.

If it's not the case, one could define other critertions:

- <u>Approach 1:</u> Consider the perceptron as an any-time algorithm. When the user is out of time or resources, return the current weights.
- <u>Approach 2:</u> After each update, calculate the accuracy, and remember the weights with highest accuracy:





KERNEL PERCEPTRON

- Lets have a look at the learning rule:

$$if\ \hat{y} \neq y_i:$$

$$w \leftarrow w + y_i x_i$$

- Thus, we can infer that the weights vector w learned by the Perceptron's algorithm is a linear combination of all the data points:

$$w = \sum_i \alpha_i y_i x_i$$

- α_i is a "mistake-counter" - how many times the perceptron made a mistake on sample x_i
- We can rewrite the prediction formula of a new observation x' as:

$$\hat{y} = sign\left[\left(\sum_i \alpha_i y_i x_i\right)^T x'\right] = sign\left[\sum_i \alpha_i y_i\, x_i^T x'\right]$$

KERNEL PERCEPTRON

$$\hat{y} = sign\left[\left(\sum_i \alpha_i y_i x_i\right)^T x'\right] = sign\left[\sum_i \alpha_i y_i \, x_i^T x'\right]$$

- In other words, the dual problem is finding the $\alpha_i' s$. The new learning algorithm would loop thru the samples and make predictions, but now it will update a "mistake counter" vector α rather than updating a weights vector w.
- <u>First observation:</u>

  We can define a threshold s, and during learning we can zero (and consider dropping samples from the dataset) any mistake-counter that reaches a value above it, i.e.:

  $if\ \alpha_i \geq s$:

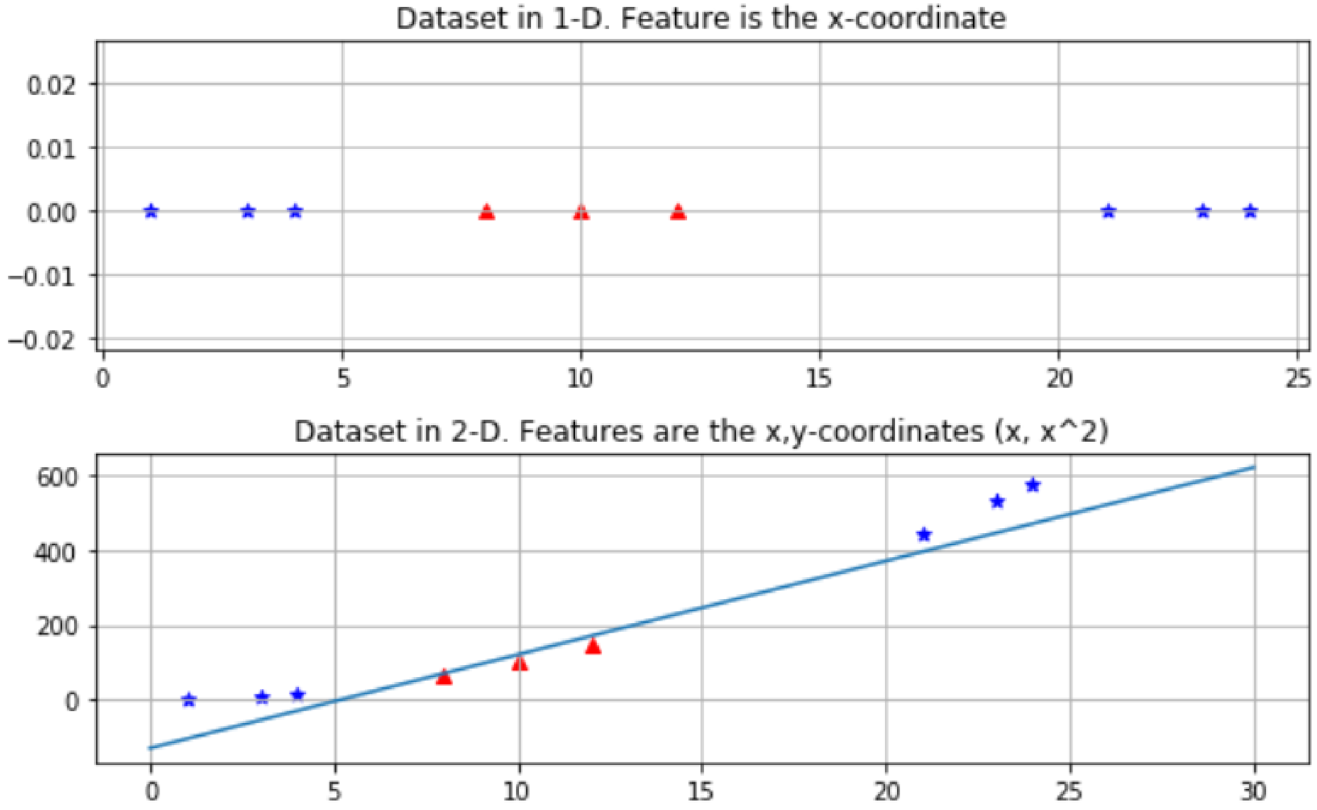
  $\alpha_i \leftarrow 0$

  $drop\ x_i\ from\ the\ dataset$

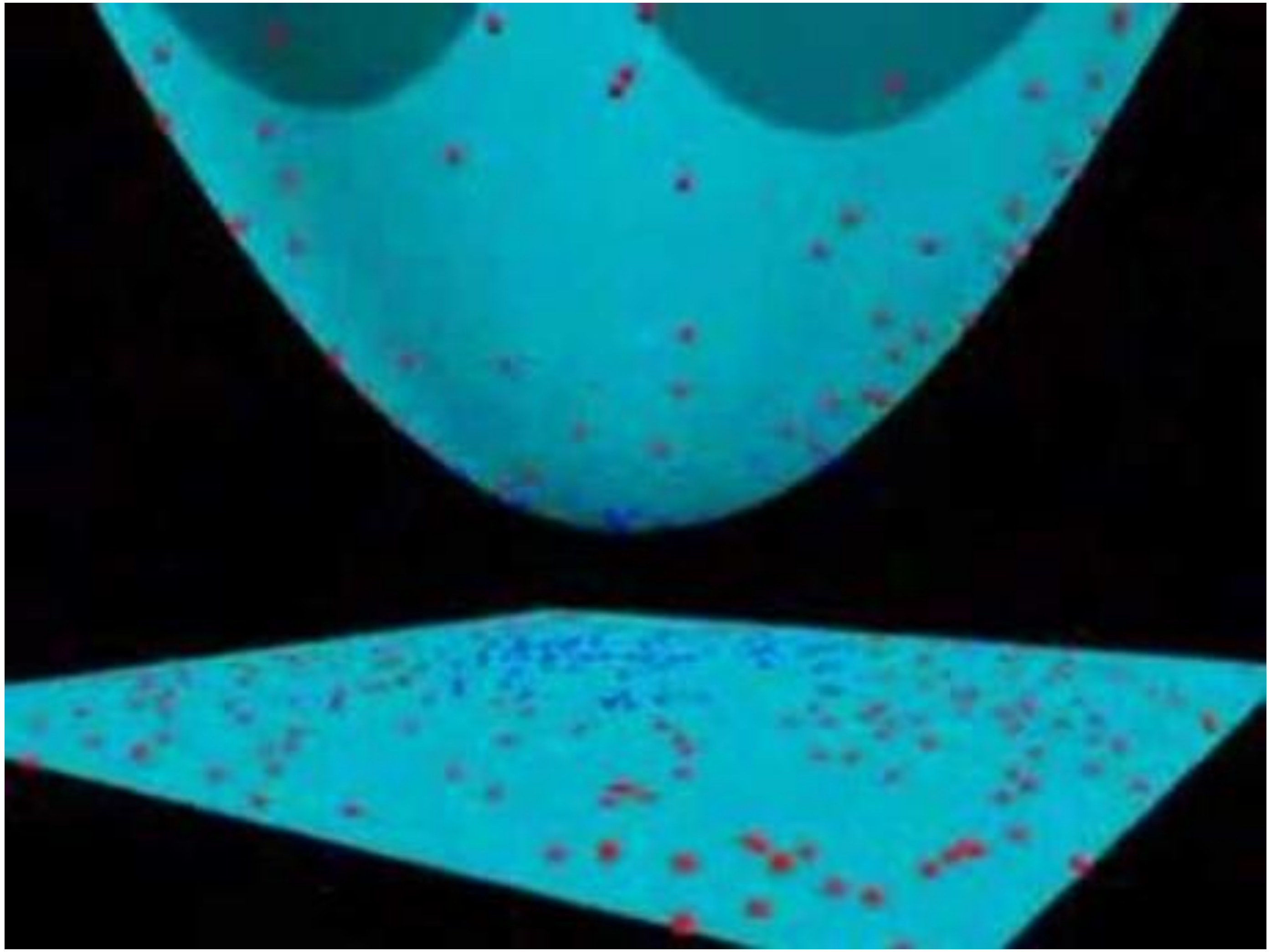
- This could help us detecting outliers

KERNEL PERCEPTRON

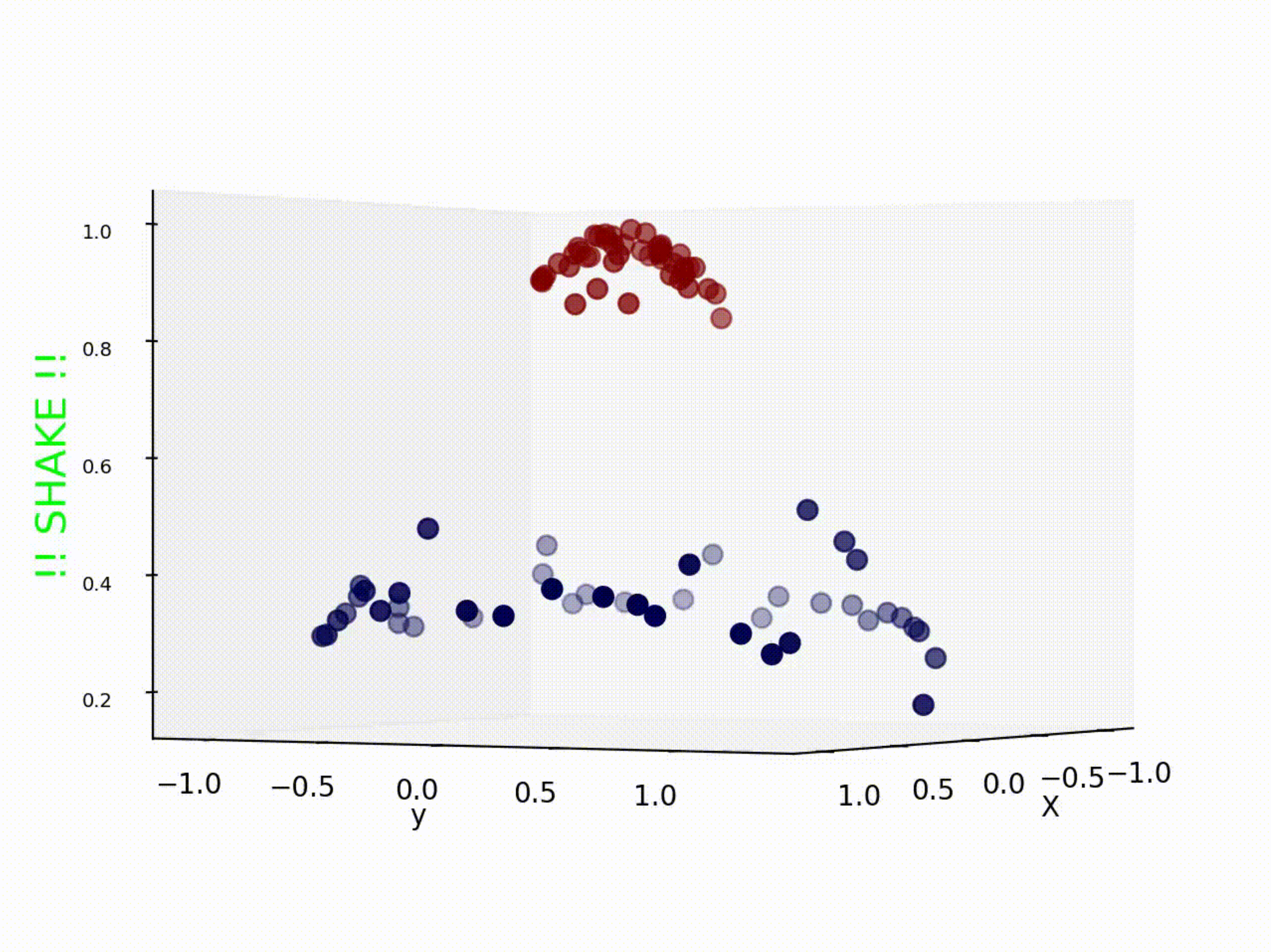
- <u>Second observation:</u>
- While there exist datasets that aren't linearly separable in a given form (set of features), we can find a transformations to another space where the points are linearly separable:



KERNEL PERCEPTRON



KERNEL PERCEPTRON



KERNEL PERCEPTRON

- Our next goal is to find transformations to spaces where our datasets are linearly separable. But firstly, we have to find a method to apply these transformations effectively.
- Let $\mathbb{R}^n$ be the features' space and $\mathbb{R}^m$. We want to find a transformation:
$$\varphi: \mathbb{R}^n \to \mathbb{R}^m$$
- The original prediction rule was:
$$\hat{y} = sign\left[\sum_i \alpha_i y_i \, x_i^T x'\right]$$
- In the new features' space, the prediction rule would become:
$$\hat{y} = sign\left[\sum_i \alpha_i y_i \, \varphi(x_i)^T \varphi(x')\right]$$
- In a low dimensions space, we cannot deal with more complex datasets.
- In a high dimensions space, the computations become very slow.
- A new trick called – "**<u>The Kernel Trick</u>**" – comes to the rescue!

KERNEL PERCEPTRON

- It makes it possible to get the same results as if you added many features, without adding them in practice.
- Usually, we define a kernel K such that $K(x, y) = \varphi^T(x)\varphi(y)$ and find a direct formula that doesn't involve any transformation to a higher dimension space.
- <u>Example:</u>

$$\varphi(y) = \varphi\left(\begin{bmatrix} u_1 \\ u_2 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ \sqrt{2}u_1 \\ \sqrt{2}u_2 \\ \sqrt{2}u_1u_2 \\ u_1^2 \\ u_2^2 \end{bmatrix}$$

The inner product:

$$\varphi^T(u)\varphi(v) = 1 + 2u_1v_1 + 2u_2v_2 + 2u_1u_2v_1v_2 + u_1^2v_1^2 + u_2^2v_2^2 = (1 + u^Tv)^2$$

Note that computing that dot project u^Tv is the original features' space is less expensive than computing the dot product in the transformed space.

KERNEL PERCEPTRON

- Examples of the most used kernel functions:

$$K(x, y) = (x^T y + 1)^p \ \ - \textit{Polynomial kernel of degree}$$

$$K(x, y) = e^{-\frac{1}{2\sigma^2}|x-y|^2} \ \ - \textit{Gaussian kernel}$$

$$K(x, y) = e^{-\gamma|x-y|^2} - \textit{RBG kernel}$$

$$K(x, y) = \tanh(\eta x^T y + \theta) \ \ - \textit{Sigmoid kernel}$$

KERNEL PERCEPTRON – THE ALGORITHM

Initialize a mistake-counter vector $\alpha \leftarrow 0$

While some stopping criterion isn't met:

For each x_j, y_j in the training set:

Predict $\hat{y} = sign\left(\sum_i \alpha_i y_i K(x_i, x_j)\right)$

If $\hat{y} \neq y_j$:

$\alpha_j \leftarrow \alpha_j + 1$

- The prediction rule:

$$\hat{y} = sign\left[\sum_i \alpha_i y_i \, \varphi(x_i)^T \varphi(x')\right]$$