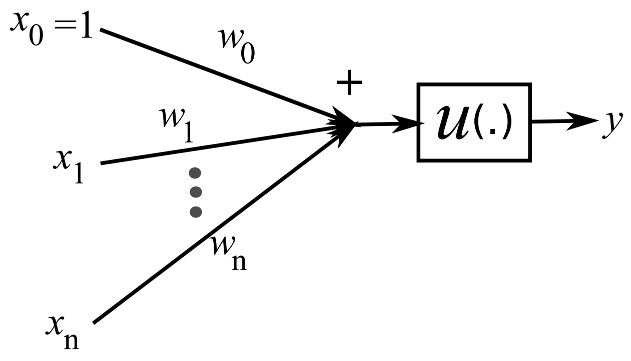
ECE/CS 559 - Neural Networks Lecture Notes #4
The perceptron and its training

Erdem Koyuncu

1 Rosenblatt's percepton

- Simplest case of a neural "network" consisting of one neuron with the McCulloch-Pitts model (step activation function).
- Can classify patterns that are linearly separable.
- Can learn to classify patterns. This is done by training the perceptron with labeled samples from each class.
- Recall that learning from labeled samples is called supervised learning.
- We typically use the symbol u for the step activation function. In other words, we define $u(x) = 1$ if $x \geq 0$, and otherwise, we define $u(x) = 0$ if $x < 0$.
- The block diagram of the perceptron is then

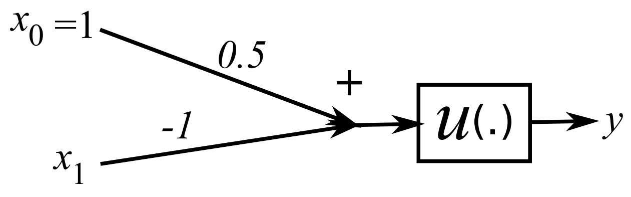


- The input output relationships are $y = u(\sum_{i=0}^{n} x_i w_i)$.

2 What can be done with the perceptron?

2.1 Logical NOT

- Input $x_1 \in \{0, 1\}$ with a 0 representing a FALSE, and 1 representing a TRUE as usual...
- We want $y = \text{NOT}(x_1)$ (sometimes also written $y = \overline{x_1}$). In other words, we want $y = \begin{cases} 1, & x_1 = 0 \\ 0, & x_1 = 1 \end{cases}$
- It turns out that the choices $w_0 = 0.5$ and $w_1 = -1$ work, i.e. we have the perceptron:



- $y = u(-x_1 + 0.5)$.
- Test: Feed $x_1 = 0$. We have $y = u(-0 + 0.5) = u(0.5) = 1$. OK.
- Test: Feed $x_1 = 1$. We have $y = u(-1+0.5) = u(-0.5) = 0$. OK. All desired input output relationships are satisfied. So, the perceptron works as intended.

2.2 Logical AND

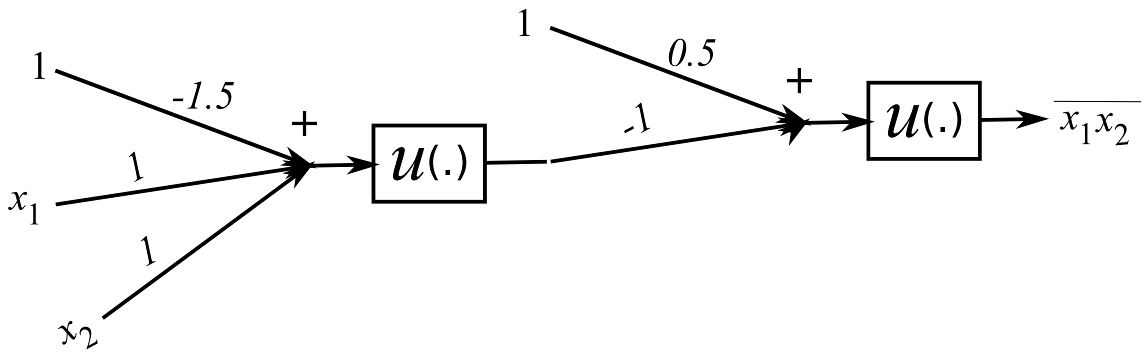
- We now have two inputs x_1 and x_2. We want $y = AND(x_1, x_2)$ or with a shorthand notation $y = x_1x_2$. The following are the input output relationships we want:

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

- Are there weights w_0, w_1, w_2 that satisfy the desired input output relationships?
- One option is again to guess to find appropriate weights.
- More systematically, one can determine the correct weights through solving linear inequalities. In our case,
- Given $x_1 = x_2 = 0$, we want the output $y = u(w_0)$ to be equal to 0. This yields the inequality $w_0 < 0$.
- Given $x_1 = 0$, $x_2 = 1$, we want $y = u(w_0 + w_2) = 0$. This yields $w_0 + w_2 < 0$.
- Given $x_1 = 1$, $x_2 = 0$, we want $y = u(w_0 + w_1) = 0$. This yields $w_0 + w_1 < 0$.
- Given $x_1 = x_2 = 1$, we want $y = u(w_0 + w_1 + w_2) = 1$. This yields $w_0 + w_1 + w_2 \geq 0$.
- One can formally solve this system of 4 inequalities. Actually, by inspection, it is not difficult to see that $w_0 = -\frac{3}{2}$, $w_1 = w_2 = 1$ is a solution.
- In general, for n inputs, setting $w_0 = -n + \frac{1}{2}$, $w_1 = \cdots = w_n = 1$ will implement the n-input AND gate, i.e. $y = x_1x_2 \cdots x_n$.

2.3 Building a digital computer using perceptrons

- We have the NOT, we have the AND, which means we can get the NAND by cascading the NOT and the AND:



- But the NAND gate is universal in the sense that every possible logic operation over an arbitrary number of inputs can be implemented using NAND gates only.
- One can thus build a computer using a sufficiently large number of perceptrons.

2.4 Logical OR

- We now have two inputs x_1 and x_2. We want $y = OR(x_1, x_2)$ or with a shorthand notation $y = x_1 + x_2$. The following are the input output relationships we want:

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

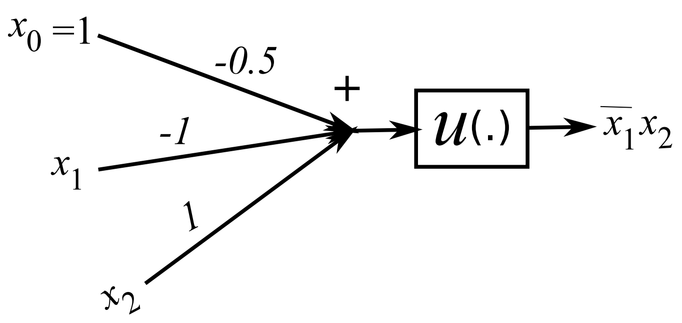
- We have the 4 inequalities corresponding to the 4 cases:
- $w_0 < 0$.
- $w_0 + w_2 \geq 0$.
- $w_0 + w_1 \geq 0$.
- $w_0 + w_1 + w_2 \geq 0$.
- By inspection, it is not difficult to see that $w_0 = -\frac{1}{2}$, $w_1 = w_2 = 1$ is a solution.
- In general, for n inputs, setting $w_0 = -\frac{1}{2}$, $w_1 = \cdots = w_n = 1$ will implement the n-input OR gate, i.e. $y = x_1 + x_2 + \cdots + x_n$.

2.5 Any logical product term

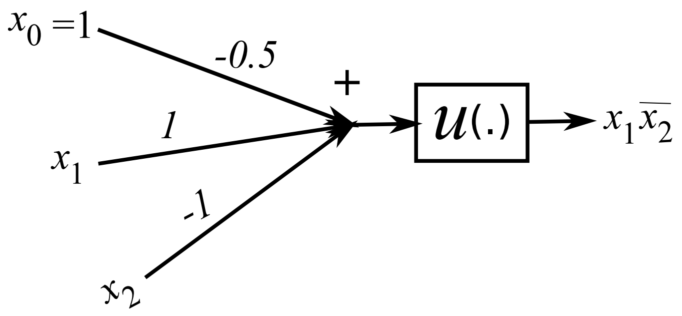
- Implementing the gate $y = \overline{x_1} \cdots \overline{x_n} x_{n+1} \cdots x_{n+m}$.
- By inspection, $w_0 = -m + \frac{1}{2}$, $w_1 = \cdots = w_n = -1$, and $w_{n+1} = \cdots = w_{n+m} = 1$ works.
- Actually this generalizes the NOT and AND gate examples earlier.
- Recall that any logic function can be implemented in a sum of products form. For example, suppose we want to implement the logic function of two variables with the following truth table (this is known as the XOR gate or the modulo-two addition):

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

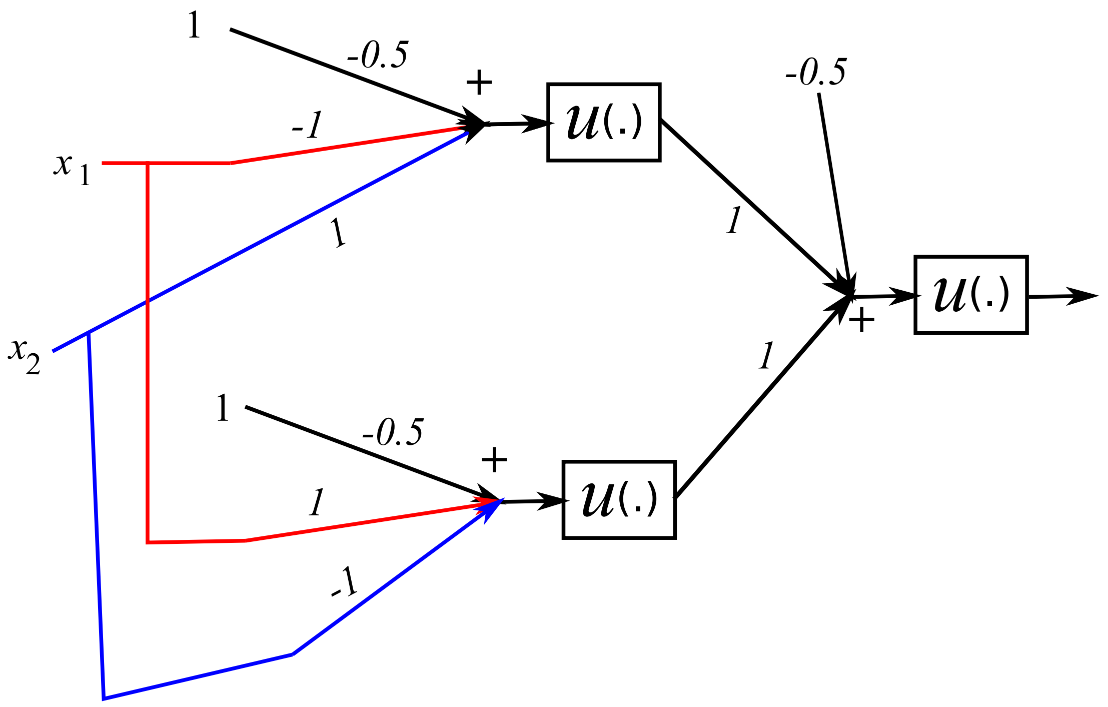
- We consider the inputs for which the output is equal to 1. The first set of inputs that provide an output of 1 are $x_1 = 0$ and $x_2 = 1$. The corresponding product term is $\overline{x_1} x_2$. The second set of inputs that provide an output of 1 are $x_1 = 1$ and $x_2 = 0$. The corresponding product term is $x_1 \overline{x_2}$. The function we are looking for is thus $y = \overline{x_1} x_2 + x_1 \overline{x_2}$ which is in the sum of products form.
- We know how to implement each product term, for example the following network provides the output $\overline{x_1} x_2$:



- On the other hand, the following network provides the output $x_1\overline{x_2}$:



- So put these two subnetworks in the first layer, and then OR them in the second layer to get the overall network that performs $y = \overline{x_1}x_2 + x_1\overline{x_2}$:



- This way, any arbitrary logic function over any number of inputs can be implemented using two layers only.

3 What cannot be done with a perceptron?

- Let us try implementing the XOR operation whose truth table was also provided above:

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

(We know how to implement XOR using two layers of perceptrons with a total of 3 perceptrons. Here we want to implement the XOR using just a single perceptron).

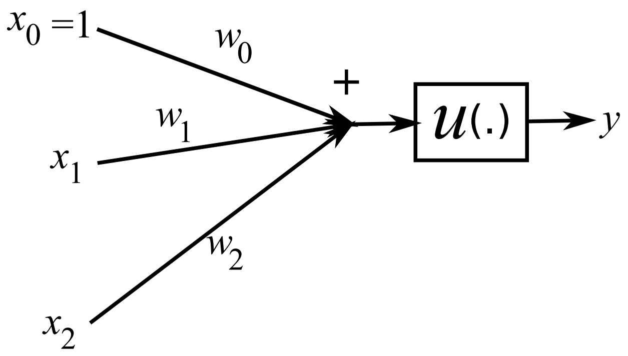
- The following inequalities should be satisfied:

$$w_0 < 0 \tag{1}$$
$$w_0 + w_2 \geq 0 \tag{2}$$
$$w_0 + w_1 \geq 0 \tag{3}$$
$$w_0 + w_1 + w_2 < 0 \tag{4}$$

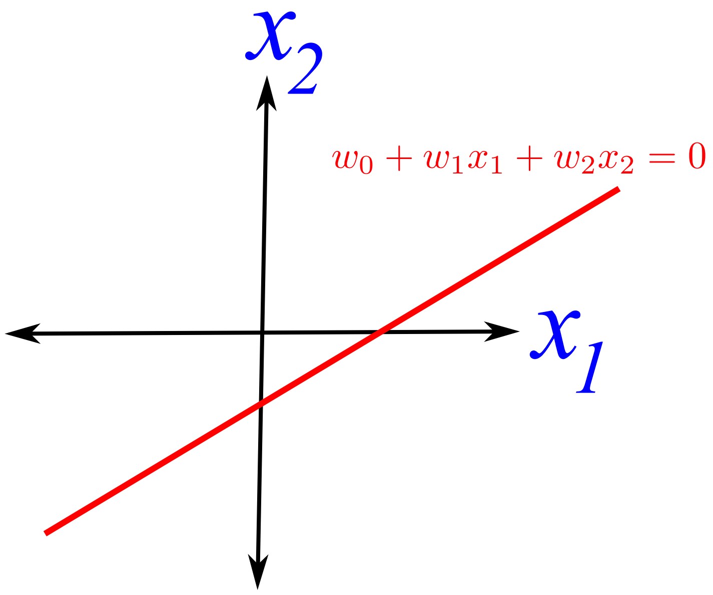
- $-(1) + (2) + (3) - (4)$, we get $0 > 0$, a contradiction! Which means, no choice of weights w_0, w_1, w_2 can satisfy the equations above. Or, a single perceptron, by itself, cannot implement the XOR gate.
- So, what is going on?

4 Geometric interpretation of the perceptron

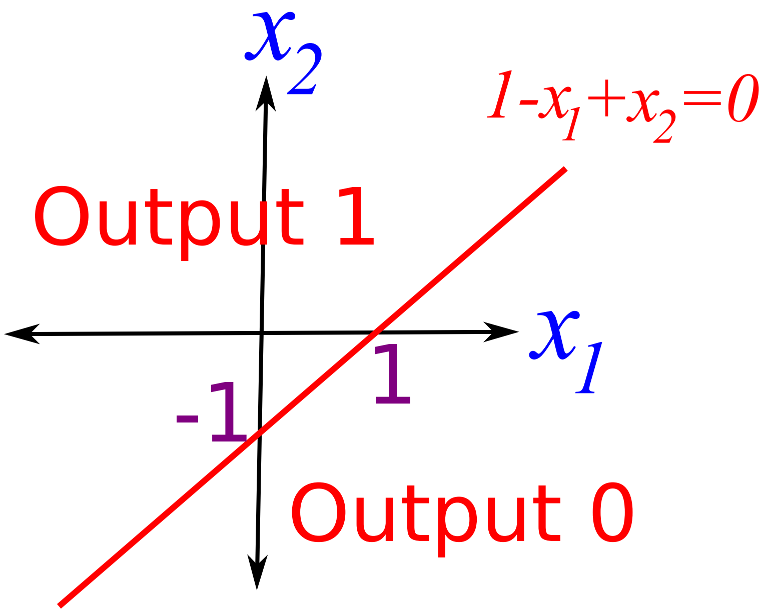
- Suppose we only have two inputs:



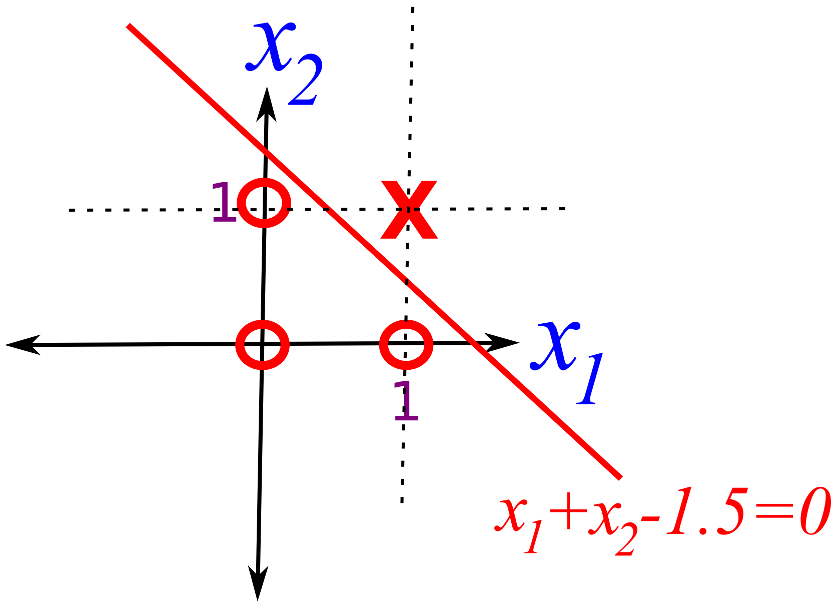
- We have $y = \begin{cases} 0, & w_0 + w_1x_1 + w_2x_2 < 0 \\ 1, & w_0 + w_1x_1 + w_2x_2 \geq 0 \end{cases}$
- Hence, the perceptron divides the input space $\mathbb{R}^2$ (the set of all x_1-x_2 pairs) to two disjoint subsets via the line $w_0 + w_1x_1 + w_2x_2 = 0$. It looks like as follows:



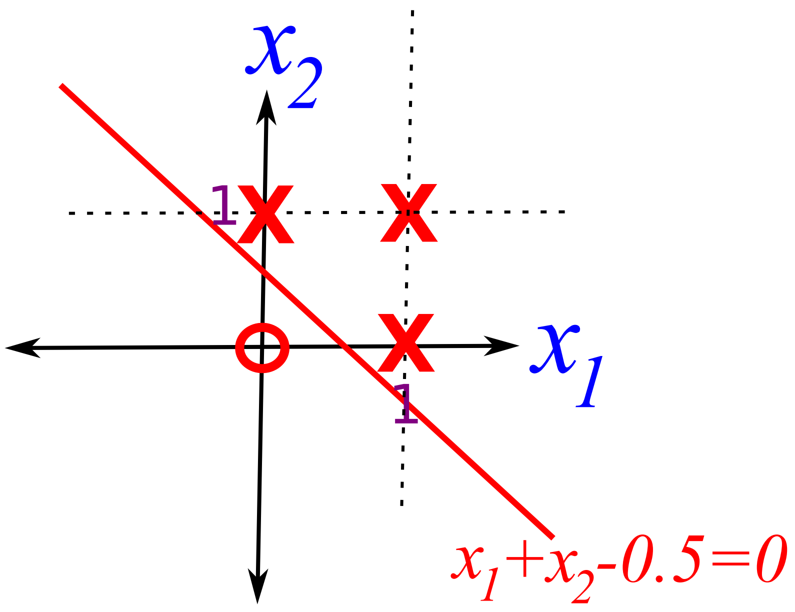
- Depending on the signs of w_1, w_2 for one side of the line, the output will be 0, and for the other side of the line, the output will be 1 (the separator always results in an output of 1 as $u(0) = 1$ by definition). For example, if $w_2 > 0$ the (x_1, x_2) pairs on the top of the line will result in an output of 1, and if $w_2 < 0$ the points on the bottom of the line will output 1.
- For example, if $w_0 = 1$, $w_1 = -1$, $w_2 = 1$, the separator is $1 - x_1 + x_2 = 0$.



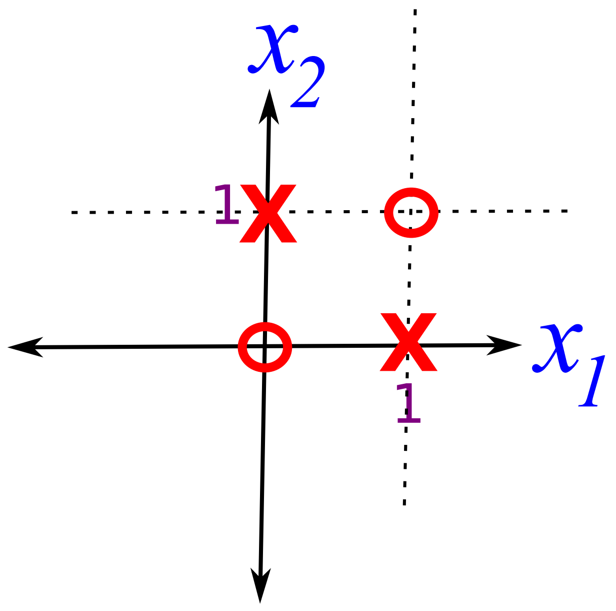
- If the signs of all weights are reversed, we would have the same separator, but now instead, the bottom of the separator would result in an output of 1.
- In general, for a perceptron with n inputs, the separator is a hyperplane, i.e. an $n-1$ dimensional subspace of $\mathbb{R}^n$. For example, for $n = 1$, you can easily verify that the separator is a mere point.
- In general, we observe that the classes should be linearly separable for the existence of a perceptron that can ultimately classify them. For the AND gate, the set of points and their desired respective classes, and the separator we found look as follows: We use an O to represent an output of 0, and an X to represent an output of 1.



For the OR gate example, the geometry looked like as follows:



- For the XOR gate, the classes we would like to separate look like as follows.



It should be clear that no line can separate the two classes. This serves as a "geometric proof" of the fact that the XOR gate cannot be implemented via a single perceptron.

5 Learning Algorithm for the Perceptron

- So, from now on, suppose we have two classes that are linearly separable. That is to say, let $\mathcal{C}_0$ and $\mathcal{C}_1$ be two subsets of $\mathbb{R}^{1+d}$ (column vectors) such that the first component of all vectors in $\mathcal{C}_0$ and $\mathcal{C}_1$ are equal to 1 (this is to introduce the bias term). We say that the classes $\mathcal{C}_0$ and $\mathcal{C}_1$ are linearly separable if there exists a weight vector $\Omega \in \mathbb{R}^{1+d}$ such that

$$\Omega^T \mathbf{x} \geq 0 \text{ for every } \mathbf{x} \in \mathcal{C}_1, \text{ and}$$
$$\Omega^T \mathbf{x} < 0 \text{ for every } \mathbf{x} \in \mathcal{C}_0.$$

Here $(\cdot)^T$ represents the matrix transpose.

- Now suppose Ω is a weight vector that can separate the classes $\mathcal{C}_0$ and $\mathcal{C}_1$ as described above. Given $\Omega = (w_0 \cdots w_n)^T$, a d-input perceptron with the step activation function, weights $w_1, \ldots, w_d$, and bias w_0 will output a 1 if the input is any vector that belongs to class $\mathcal{C}_1$. Otherwise, for any input vector that belongs to class $\mathcal{C}_0$, the perceptron will output a 0.
- Given that $\mathcal{C}_0$ and $\mathcal{C}_1$ are linearly separable, how to find a weight vector $\mathbf{w}$ that can separate them?
- **Option 1:** Write all the $|\mathcal{C}_0| + |\mathcal{C}_1|$ inequalities and solve them: Not a very good idea if the classes are large.
- **Option 2:** Instead, we use the Perceptron Training Algorithm (PTA) that finds the weight for us. It is one special case of supervised learning.
- Suppose n training samples $x_1, \ldots, x_n \in \mathbb{R}^{1+d}$ are given. Again, the first component of these vectors are assumed to be equal to 1 to take into account the bias. Let $d_1, \ldots, d_n \in \{0, 1\}$ represent the desired output for each of the input vectors. With this notation, we have $\mathcal{C}_0 = \{\mathbf{x}_i : d_i = 0\}$, and $\mathcal{C}_1 = \{\mathbf{x}_i : d_i = 1\}$. Suppose $\mathcal{C}_0$ and $\mathcal{C}_1$ are linearly separable. Consider any learning parameter $\eta > 0$. Then, the following algorithm finds a weight vector $\Omega \in \mathbb{R}^{1+d}$ that can separate the two classes:

```
1. Initialize $\Omega$ arbitrarily (e.g. randomly).
2. epochNumber $\leftarrow 0$.
3. While $\Omega$ cannot correctly classify all input patterns, i.e. while $u(\Omega^T \mathbf{x}_i) \neq d_i$ for some $i \in \{1, \ldots, n\}$,
   (a) epochNumber $\leftarrow$ epochNumber $+ 1$.
   (b) for $i = 1$ to $n$
       i. Calculate $y = u(\Omega^T x_i)$ (the output for the $i$th training sample with the current weights).
       ii. If $y = 1$ but $d_i = 0$,
           A. Update the weights as $\Omega \leftarrow \Omega - \eta \mathbf{x}_i$.
       iii. If $y = 0$ but $d_i = 1$,
           A. Update the weights as $\Omega \leftarrow \Omega + \eta \mathbf{x}_i$.
```

- Of course the variable epochNumber really does not do anything and it is just there for tracking purposes. We can also write the entire algorithm in a considerably simpler form:

```
1. Initialize $\Omega$ arbitrarily (e.g. randomly).
2. While $\Omega$ cannot correctly classify all input patterns, i.e. while $u(\Omega^T \mathbf{x}_i) \neq d_i$ for some $i \in \{1, \ldots, n\}$,
   (a) for $i = 1$ to $n$
       i. $\Omega \leftarrow \Omega + \eta \mathbf{x}_i (d_i - u(\Omega^T \mathbf{x}_i))$.
```

- Of course, the "while" conditioning can also be optimized - you can figure out those optimizations yourself.
- The parameter η is usually referred to as the learning rate parameter, or simply, the learning parameter.
  - Small η: The effects of the past inputs are more pronounced, fluctuations of the weights are lesser in magnitude, may take longer to converge.

  - Large η: Faster adaptations to any possibly changes in the underlying classes. May also take longer to converge (this time because of large fluctuations).

- Nevertheless, if the input classes are linearly separable, then the PTA converges for any $\eta > 0$ (and thus results in a weight vector that can separate the two classes).

- **Proof:** I am providing a proof here as the proofs I found in different references were at places incomplete/flawed. We will provide a proof for the initialization $\Omega = \Omega_0 = 0$ (all zero vector) and $\eta = 1$. The proof can be generalized to different initializations and other η.

  Let Ω_{opt} be a vector of weights that can separate the two classes (existence is guaranteed as the classes are assumed to be linearly separable). We have

$$\mathbf{x}^T\Omega_{\text{opt}} \geq 0 \text{ for every } \mathbf{x} \in \mathcal{C}_1, \text{ and}$$
$$\mathbf{x}^T\Omega_{\text{opt}} < 0 \text{ for every } \mathbf{x} \in \mathcal{C}_0.$$

  Step-1: In fact, we can assume, without loss of generality, that

$$\mathbf{x}^T\Omega_{\text{opt}} > 0 \text{ for every } \mathbf{x} \in \mathcal{C}_1,$$

  i.e. we have strict inequality. To see this, let $\delta = \min_{\mathbf{x}\in\mathcal{C}_0} |\mathbf{x}^T\Omega_{\text{opt}}|$. Then, for

$$\Omega'_{\text{opt}} = \Omega_{\text{opt}} + \frac{\delta}{2}[1 \quad 0 \quad \cdots \quad 0]^T,$$

  we have, for every $\mathbf{x} \in \mathcal{C}_1$,

$$\mathbf{x}^T\Omega'_{\text{opt}} = \mathbf{x}^T\Omega_{\text{opt}} + \frac{\delta}{2}\mathbf{x}^T[1 \quad 0 \quad \cdots \quad 0]^T = \underbrace{\mathbf{x}^T\Omega_{\text{opt}}}_{\geq 0} + \underbrace{\frac{\delta}{2}}_{>0} > 0,$$

  and if $\mathbf{x} \in \mathcal{C}_0$, we have

$$\mathbf{x}^T\Omega'_{\text{opt}} = \underbrace{\mathbf{x}^T\Omega_{\text{opt}}}_{\leq -\delta} + \frac{\delta}{2} \leq -\frac{\delta}{2} < 0.$$

  Hence, Ω'_{opt} is also a correct classifier.

  Step-2: Recall that we show the samples in the order $\mathbf{x}_1, \ldots, \mathbf{x}_n, \mathbf{x}_1, \ldots, \mathbf{x}_n, \ldots$. Let $j_1, j_2, \ldots, \in \{1, \ldots, n\}$ represent the sequence of indices at which we update the neuron weights. Without of loss generality, we assume $j_i = i$. Let Ω_i denote the neuron weights after the ith update. Recall that the initial weights we choose are $\Omega_0 = 0$. We have

$$\Omega_1 = \begin{cases} \Omega_0 + \mathbf{x}_1,\ \Omega_{\text{opt}}^T\mathbf{x}_1 \geq 0,\ \Omega_0^T\mathbf{x}_1 < 0 \\ \Omega_0 - \mathbf{x}_1,\ \Omega_{\text{opt}}^T\mathbf{x}_1 < 0,\ \Omega_0^T\mathbf{x}_1 \geq 0 \end{cases}$$
$$\Omega_2 = \begin{cases} \Omega_1 + \mathbf{x}_2,\ \Omega_{\text{opt}}^T\mathbf{x}_2 \geq 0,\ \Omega_1^T\mathbf{x}_2 < 0 \\ \Omega_1 - \mathbf{x}_2,\ \Omega_{\text{opt}}^T\mathbf{x}_2 < 0,\ \Omega_1^T\mathbf{x}_2 \geq 0 \end{cases}$$
$$\vdots$$
$$\Omega_N = \begin{cases} \Omega_{N-1} + \mathbf{x}_N,\ \Omega_{\text{opt}}^T\mathbf{x}_N \geq 0,\ \Omega_{N-1}^T\mathbf{x}_N < 0 \\ \Omega_{N-1} - \mathbf{x}_N,\ \Omega_{\text{opt}}^T\mathbf{x}_N < 0,\ \Omega_{N-1}^T\mathbf{x}_N \geq 0 \end{cases}$$

  Why do you have these updates? For example, in order to have $\Omega_1 = \Omega_0 + \mathbf{x}_1$, the desired output given $\mathbf{x}_1$ should be 1 (this is provided by the condition $\Omega_{\text{opt}}^T\mathbf{x}_1 \geq 0$), but the actual output given $\mathbf{x}_1$ with the current weights Ω_0 should be 0 (this is provided by the condition $\Omega_0^T\mathbf{x}_1 < 0$). Similar with others.

  Take the first equality and multiply both sides by Ω_{opt}^T, we obtain:

$$\Omega_{\text{opt}}^T\Omega_1 = \begin{cases} \Omega_{\text{opt}}^T\Omega_0 + \Omega_{\text{opt}}^T\mathbf{x}_1,\ \Omega_{\text{opt}}^T\mathbf{x}_1 \geq 0,\ \Omega_0^T\mathbf{x}_1 < 0 \\ \Omega_{\text{opt}}^T\Omega_0 - \Omega_{\text{opt}}^T\mathbf{x}_1,\ \Omega_{\text{opt}}^T\mathbf{x}_1 < 0,\ \Omega_0^T\mathbf{x}_1 \geq 0 \end{cases}$$

The first conditions tell you that you can just write this in the considerably simpler form:

$$\Omega_{\text{opt}}^T \Omega_1 = \Omega_{\text{opt}}^T \Omega_0 + |\Omega_{\text{opt}}^T \mathbf{x}_1|.$$

Doing the same for all equalities, and using the fact that $\Omega_0 = 0$, we obtain

$$\begin{aligned}
\Omega_{\text{opt}}^T \Omega_1 &= \Omega_{\text{opt}}^T \Omega_0 + |\Omega_{\text{opt}}^T \mathbf{x}_1| = |\Omega_{\text{opt}}^T \mathbf{x}_1| \\
\Omega_{\text{opt}}^T \Omega_2 &= \Omega_{\text{opt}}^T \Omega_1 + |\Omega_{\text{opt}}^T \mathbf{x}_2| = |\Omega_{\text{opt}}^T \mathbf{x}_1| + |\Omega_{\text{opt}}^T \mathbf{x}_2| \\
&\vdots \\
\Omega_{\text{opt}}^T \Omega_N &= \sum_{i=1}^{N} |\Omega_{\text{opt}}^T \mathbf{x}_i|
\end{aligned}$$

From Step 1, recall that $|\Omega_{\text{opt}}^T \mathbf{x}_i| > 0$ for every $i \in \{1, \ldots, n\}$. Therefore, letting

$$\alpha = \min_{i \in \{1,\ldots,n\}} |\Omega_{\text{opt}}^T \mathbf{x}_i|,$$

we obtain

$$\Omega_{\text{opt}}^T \Omega_N \geq N\alpha$$

We now recall Cauchy-Schwarz inequality: For vectors $\mathbf{x}, \mathbf{y}$, we have $\mathbf{x}^T \mathbf{y} \leq \|\mathbf{x}\| \|\mathbf{y}\|$. In particular, $\Omega_{\text{opt}}^T \Omega_N \leq \|\Omega_{\text{opt}}\| \|\Omega_N\|$ so that

$$\|\Omega_N\|^2 \geq \frac{N^2 \alpha^2}{\|\Omega_{\text{opt}}\|^2} \tag{5}$$

Step-3: Recall the first equality in the beginning of Step 2:

$$\Omega_1 = \begin{cases} \Omega_0 + \mathbf{x}_1, \; \Omega_{\text{opt}}^T \mathbf{x}_1 \geq 0, \; \Omega_0^T \mathbf{x}_1 < 0 \\ \Omega_0 - \mathbf{x}_1, \; \Omega_{\text{opt}}^T \mathbf{x}_1 < 0, \; \Omega_0^T \mathbf{x}_1 \geq 0 \end{cases}$$

Taking the squared Euclidean norms of both sides and expanding:

$$\|\Omega_1\|^2 = \begin{cases} \|\Omega_0\|^2 + \|\mathbf{x}_1\|^2 + 2\Omega_0^T \mathbf{x}_1, \; \Omega_{\text{opt}}^T \mathbf{x}_1 \geq 0, \; \Omega_0^T \mathbf{x}_1 < 0 \\ \|\Omega_0\|^2 + \|\mathbf{x}_1\|^2 - 2\Omega_0^T \mathbf{x}_1, \; \Omega_{\text{opt}}^T \mathbf{x}_1 < 0, \; \Omega_0^T \mathbf{x}_1 \geq 0 \end{cases}$$

Thus either way,

$$\|\Omega_1\|^2 = \|\Omega_0\|^2 + \|\mathbf{x}_1\|^2 - 2|\Omega_0^T \mathbf{x}_1| \leq \|\Omega_0\|^2 + \|\mathbf{x}_1\|^2$$

and likewise from the other equalities, we obtain

$$\begin{aligned}
\|\Omega_2\|^2 &\leq \|\Omega_1\|^2 + \|\mathbf{x}_2\|^2 \\
&\vdots \\
\|\Omega_N\|^2 &\leq \|\Omega_{N-1}\|^2 + \|\mathbf{x}_N\|^2
\end{aligned}$$

Summing all inequalities up, we obtain

$$\|\Omega_N\|^2 \leq \sum_{i=1}^{N} \|\mathbf{x}_i\|^2 \leq \beta N \tag{6}$$

where

$$\beta = \max_{i \in \{1,\ldots,n\}} \|\mathbf{x}_i\|^2$$

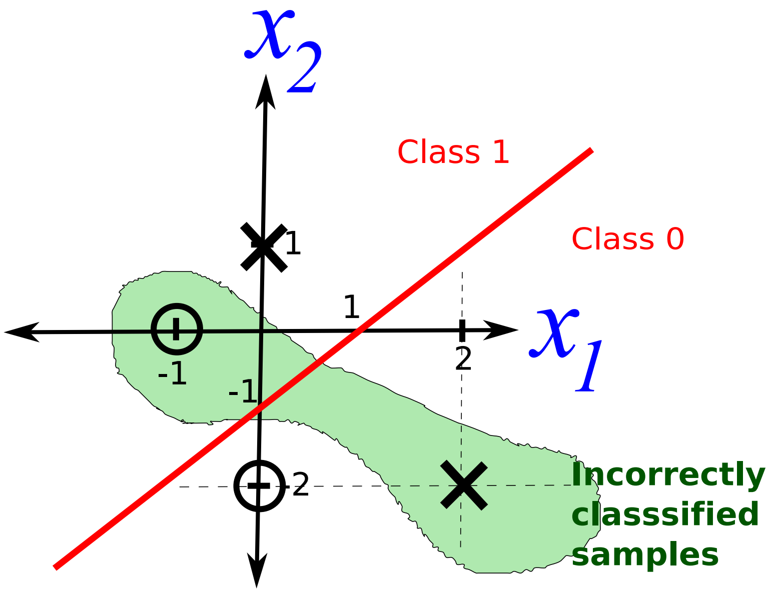
Step-4: Combining (5) and (6),

$$N \leq \frac{\beta \|\Omega_{\text{opt}}\|^2}{\alpha^2}$$

This means that the number of updates is finite, and thus the PTA converges. □

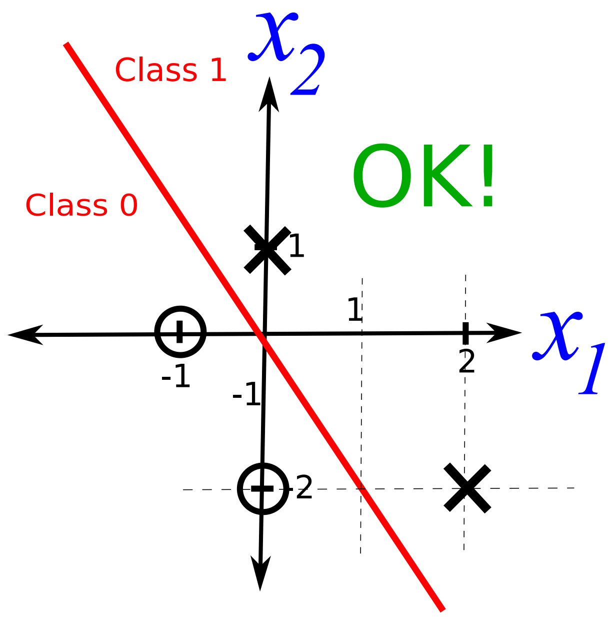
6 An Example

Given initial weights $\Omega_0 = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}$, training set (without 1s for the biases) $\begin{bmatrix} 2 \\ -2 \end{bmatrix}, \begin{bmatrix} 0 \\ -2 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ 0 \end{bmatrix}$ with the desired outputs $1, 0, 1, 0$.



Initial weights cannot correctly classify the samples/patterns. So, let $\mathbf{x}_1 = \begin{bmatrix} 1 \\ 2 \\ -2 \end{bmatrix}, \mathbf{x}_2 = \begin{bmatrix} 1 \\ 0 \\ -2 \end{bmatrix}, \mathbf{x}_3 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, \mathbf{x}_4 = \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}$ (I put the 1s for the biases).

- Epoch 1
  - We feed $\mathbf{x}_1$. $\Omega_0^T \mathbf{x}_1 = -3$, which implies an output of $u(-3) = 0$. But the desired output is 1. So, we update $\Omega_1 = \Omega_0 + \mathbf{x}_1 = \begin{bmatrix} 2 \\ 1 \\ -1 \end{bmatrix}$.
  - We feed $\mathbf{x}_2$. Now we use the updated weights: $\Omega_1^T \mathbf{x}_2 = 4$, which implies an output of $u(4) = 1$. But the desired output is 0. So, we update $\Omega_2 = \Omega_1 - \mathbf{x}_2 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$.
  - We feed $\mathbf{x}_3$. We have $u(\Omega_2^T \mathbf{x}_3) = 1$. The desired output is also 1. So, no update on the weights, or $\Omega_3 = \Omega_2$.
  - We feed $\mathbf{x}_4$. We have $u(\Omega_3^T \mathbf{x}_4) = 0$. But the desired output is 0. So, we update $\Omega_4 = \Omega_3 - \mathbf{x}_4 = \begin{bmatrix} 0 \\ 2 \\ 1 \end{bmatrix}$.
- Epoch 2: It turns out that there are no updates at this epoch, which means the PTA has converged. Let us now look at the final situation geometrically:


x_2
Class 1
Class 0
OK!
1
1
2
x_1
-1
-1
-2