Perceptrons

Courtesy: Prof. M. Khapra

The story ahead ...

What about non-boolean (say, real) inputs ?

Do we always need to hand code the threshold ?

Are all inputs equal ? What if we want to assign more weight (importance) to some inputs ?

What about functions which are not linearly separable ?

We will now try to answer the following questions:

Why are we trying to implement boolean functions?

Why do we need weights ?

Why is $w_0 = -\theta$ called the bias ?

1. Real-valued input, function

We discussed boolean input and boolean function

but will all problems have only boolean input?

Instead of the "go to the movie" decision problem.

Eg.

Say, you are in an oil mining company and

trying to decide

whether you should mine or drill at a particular station or not.

This could depend on various factors--

pressure on the ocean surface at that point,

salinity of the water at that point,

aquatic life at that point and so on.

Not Boolean function.

Salinity is a real number,

density would be a real number,

pressure would be a real number and so on.

A valid decision problem.

Companies would be interested in it.

2. Learn, not calculate the threshold

How did we decide the threshold with MP neurons?

Hand-computed them.

Won't work here.

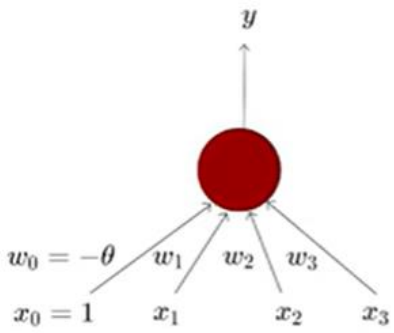
(i.e.) doesn't scale to larger problems where there are

many more dimensions,

the inputs are not boolean and so on.

We need a way of learning this threshold.

3. Weigh the inputs



Consider the task of predicting whether we would like a movie or not

Suppose, we base our decision on 3 inputs (binary, for simplicity)

Based on our past viewing experience (**data**), we may give a high weight to *isDirectorNolan* as compared to the other inputs

Specifically, even if the actor is not *Matt Damon* and the genre is not *thriller* we would still want to cross the threshold θ by assigning a high weight to *isDirectorNolan*

$x1 = isActorDamon$

$x2 = isGenreThriller$

$x3 = isDirectorNolan$

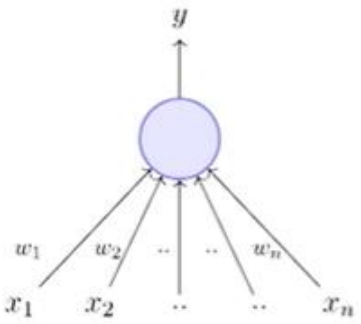
4. Functions not being linearly separable

Maybe even with the (restricted) boolean case.

Perceptron tries to fix some of these things.

So, first, perceptrons, then, others.

Perceptrons - the classical and the refined perceptrons



- Frank Rosenblatt, an American psychologist, proposed the **classical perceptron** model (1958)
- A more general computational model than McCulloch–Pitts neurons
- **Main differences:** Introduction of numerical weights for inputs and a mechanism for learning these weights
- Inputs are no longer limited to boolean values
- Refined and carefully analyzed by Minsky and Papert (1969) - their model is referred to as the **perceptron** model here

How do we know that the actor is what matters or
the director is what matters?

Given a lot of data about the movies watched in the past.
How do I know which are the weights to assign to this?
We will see an algorithm for it.

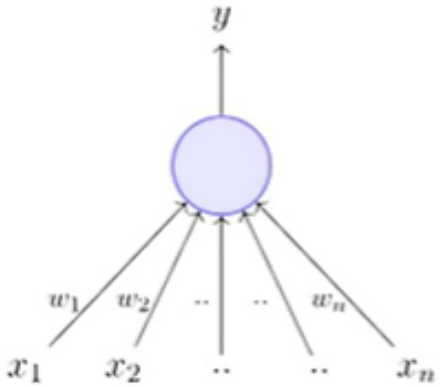
The inputs are no longer limited to boolean values.

This is the classical perceptron,
but what we will be discussing is the "refined" version
proposed by Minsky and Papert,
known as the perceptron model.

What does the perceptron do?

How does it operate?

It will give an output of 1
if the weighted sum of the inputs
is greater than a threshold and
0 if this weighted sum is less than threshold.



$$y = 1 \quad if \sum_{i=1}^{n} w_i * x_i \geq \theta$$

$$= 0 \quad if \sum_{i=1}^{n} w_i * x_i < \theta$$

Not very different from the MP neuron.

A trick to get it to a better notation (form).

Take θ to the left.

Rewriting the above,

$$y = 1 \quad if \sum_{i=1}^{n} w_i * x_i - \theta \geq 0$$

$$= 0 \quad if \sum_{i=1}^{n} w_i * x_i - \theta < 0$$

A more accepted convention,

$$y = 1 \quad if \sum_{i=0}^{n} w_i * x_i \geq 0$$

Indices are 0 to n and

θ has disappeared.

What happened?

$$x_0 = 1 \quad and \quad w_0 = -\theta$$

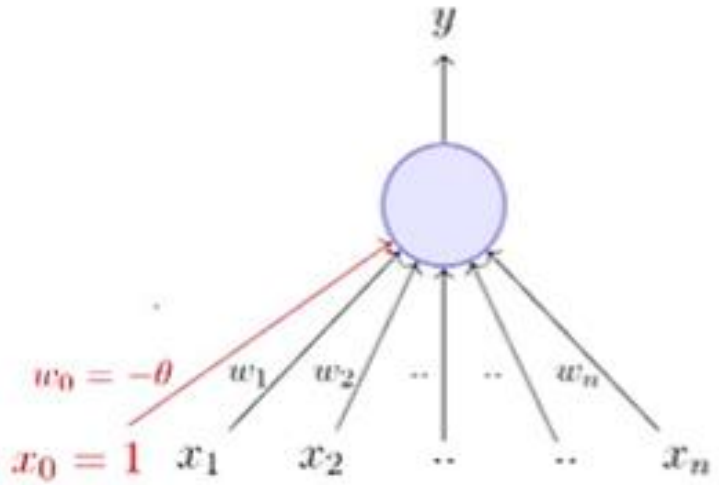
(i.e.) we are assuming that

instead of having the threshold as a separate quantity,

we think of it as one of the inputs

which is always on and

the weight of that input is -θ.



A more accepted convention,

$$y = 1 \quad if \sum_{i=0}^{n} w_i * x_i \geq 0$$

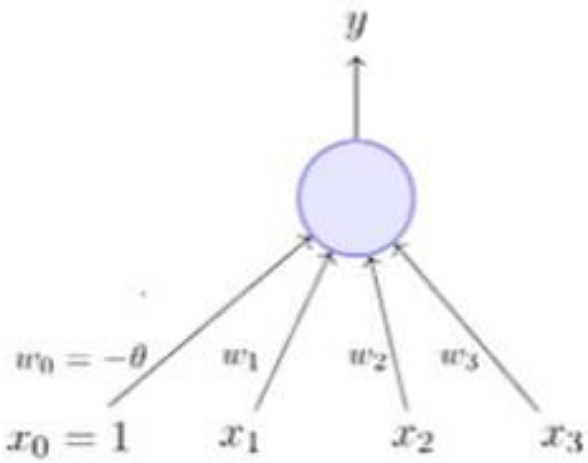
$$= 0 \quad if \sum_{i=0}^{n} w_i * x_i < 0$$

$$where, \quad x_0 = 1 \quad and \quad w_0 = -\theta$$

So, the job of all the other inputs and their weights is to make sure their sum is > this input we have.

The perceptron fires when this summation is >= 0,
and otherwise, not.

Back to the task of predicting whether you would like to watch a movie or not and
suppose we base our decisions on three simple inputs;
actor,
genre and
director.



$$x_1 = isActorDamon$$
$$x_2 = isGenreThriller$$
$$x_3 = isDirectorNolan$$

w0 = -θ is often called the bias because it represents the "prior" (prejudice).

Based on our past viewing experience,
we may give a high weight to Nolan as compared to the other inputs.
It means that
as long as the director is Christopher Nolan,
we are going to watch this movie
irrespective of who the actor is or
what the genre is.

Suppose you are a movie buff.
What would θ be?
0
cuz you will watch any movie irrespective of the actor,
the director and
the genre.

Suppose you are a very niche (selective) movie watcher
who only watches those movies whose genre is thriller,
the director was Christopher Nolan and
the actor was Damon,

What would θ be?
3.

High
(Interstellar)
The weights and the bias
will depend on the data
which in this case is the viewing history.
That's why we've the weights and biases and why we want to learn them.

Before we see whether or how we can learn these weights and biases,
what kind of functions can be implemented using the perceptron and
are these functions any different from the McCulloch Pitts neuron?

McCulloch Pitts Neuron
(assuming no inhibitory inputs)

$$y = 1 \quad if \sum_{i=0}^{n} x_i \geq \theta$$
$$= 0 \quad if \sum_{i=0}^{n} x_i < \theta$$

Perceptron

$$y = 1 \quad if \sum_{i=0}^{n} w_i * x_i \geq \theta$$
$$= 0 \quad if \sum_{i=0}^{n} w_i * x_i < \theta$$

- From the equations it should be clear that even a perceptron separates the input space into two halves
- All inputs which produce a 1 lie on one side and all inputs which produce a 0 lie on the other side
- In other words, a single perceptron can only be used to implement linearly separable functions
- Then what is the difference? The weights (including threshold) can be learned and the inputs can be real valued

The only difference from MC neuron:
weights and
a mechanism for learning these weights,
in addition to learning the threshold.
We'll not hand-code them.

We will first revisit some boolean functions and
see the perceptron learning algorithm
(for learning the weights).

x_1	x_2	OR	
0	0	0	$w_0 + \sum_{i=1}^{2} w_i x_i < 0$
1	0	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
0	1	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
1	1	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$

$$w_0 + w_1 \cdot 0 + w_2 \cdot 0 < 0 \implies w_0 < 0$$

(expanding the first condition)

Similarly, expanding all,

$$w_0 + w_1 \cdot 0 + w_2 \cdot 0 < 0 \implies w_0 < 0$$
$$w_0 + w_1 \cdot 0 + w_2 \cdot 1 \geq 0 \implies w_2 > -w_0$$
$$w_0 + w_1 \cdot 1 + w_2 \cdot 0 \geq 0 \implies w_1 > -w_0$$
$$w_0 + w_1 \cdot 1 + w_2 \cdot 1 \geq 0 \implies w_1 + w_2 > -w_0$$

What we've is

a system of linear inequalities.

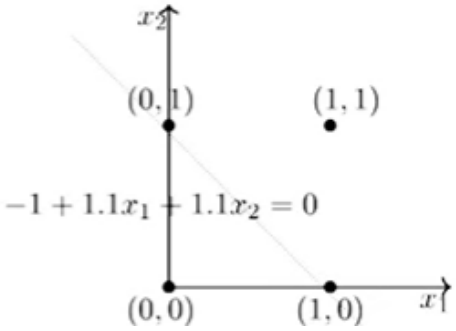
There are algorithms for solving these, though not always.

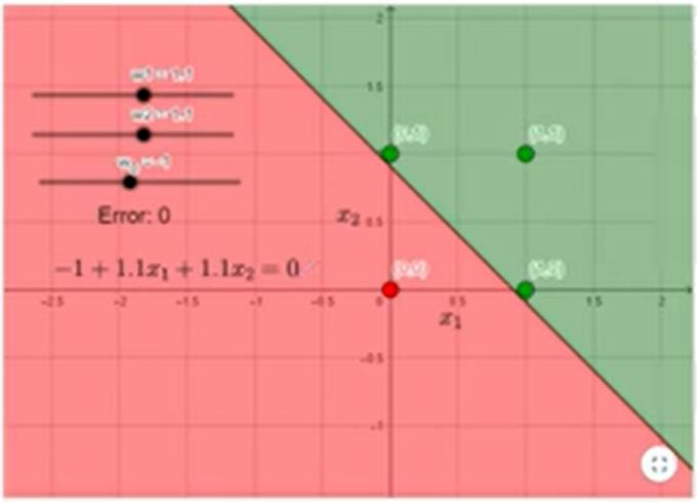
One possible solution here is

w0 = -1,
w1 = 1.1 and
w2 = 1.1.

Let us draw that line.

1.1x1 + 1.1x2 = 1.

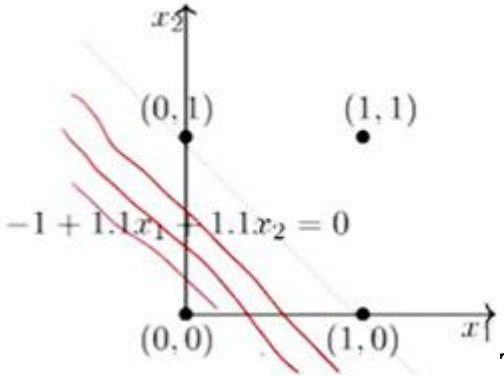




It satisfies the conditions.

Is it the only possible solution?
No.

Many more valid lines (solutions).



These result in different w1, w2, and w0s.

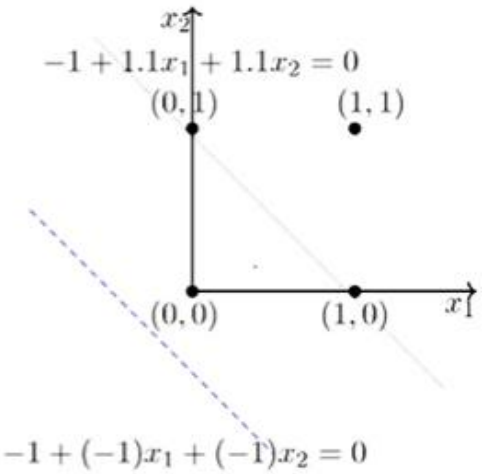
Note:
We can come up with
a similar set of inequalities
and find the value of θ (that we can set to solve them)
for a McCulloch Pitts neuron also.

Before we go to learning,
let's discuss errors and error surfaces and
how they relate to these multiple solutions.

3.2 Errors and error surfaces

The simple eg again:

- Let us fix the threshold $(-w_0 = 1)$ and try different values of w_1, w_2
- Say, $w_1 = -1, w_2 = -1$



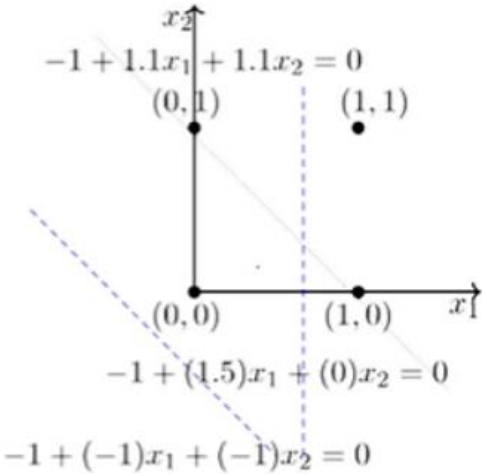
Is there anything wrong with this line?

Any errors?

One error on 1 of the 4 inputs.

- Let us try some more values of w_1, w_2 and note how many errors we make

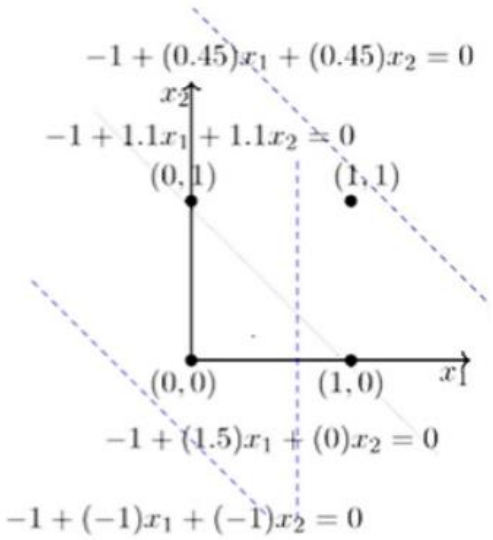
w_1	w_2	errors
-1	-1	1
1.5	0	1



Only one error because (0,0) should be anyway below the line.

- Let us try some more values of w_1, w_2 and note how many errors we make

w_1	w_2	errors
-1	-1	1
1.5	0	1
0.45	0.45	3



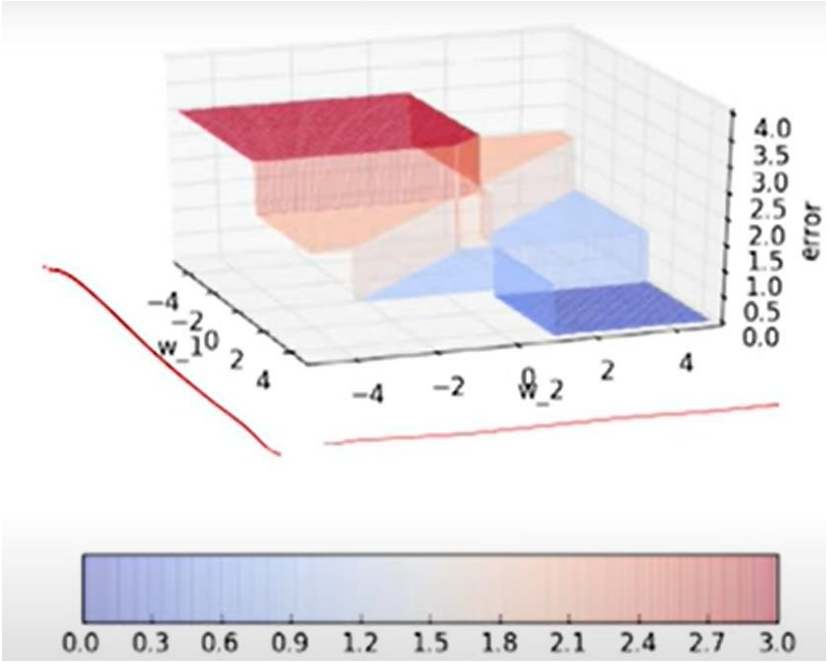
Again, only three errors because (0,0) should be anyway below the line.

Now that we understand about errors,
our goal to find w1, w2, …, w_n
can be stated as
find w1, w2, …, w_n
such that errors are minimized
or errors are 0 in the best case.

(i.e.) the objective of our search for w1, w2, …, w_n now is to
minimize these errors.

- We are interested in those values of w_0, w_1, w_2 which result in 0 error
- Let us plot the error surface corresponding to different values of w_0, w_1, w_2

- For ease of analysis, we will keep w_0 fixed (-1) and plot the error for different values of w_1, w_2



w1, w2 can go from -∞ to +∞ but
for the sake of showing it,
we have it from -4 to 4 here.

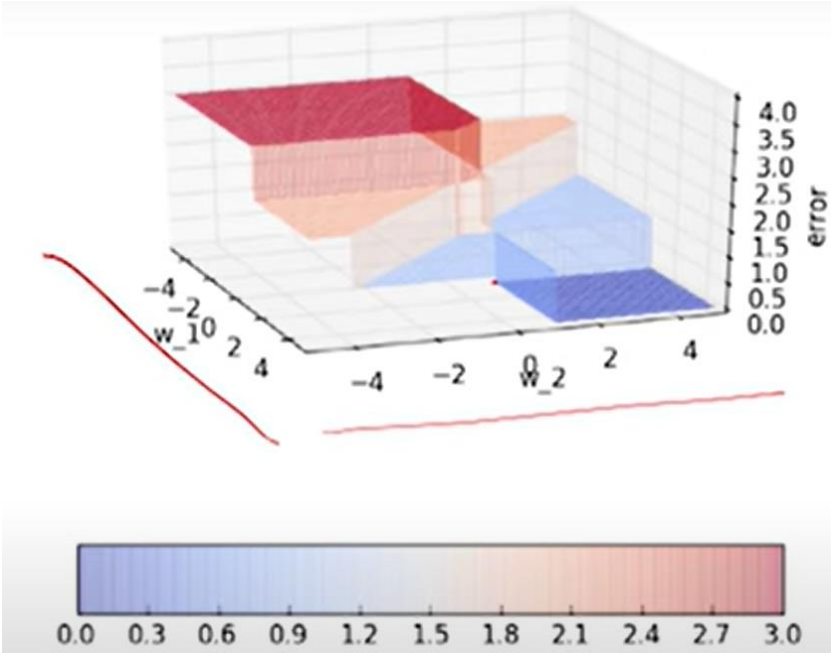
What we'll do is
we are searching for some values of w's--w1 and w2
so that the error is 0.

Do a brute force--try every value from -4 to 4.

In fact, one of the solutions I proposed earlier was this 1.1, 1.1.

The line on the previous slide led to 0 error.

That is the dark blue surface here.



So, how did I compute this error?

- For a given w_0, w_1, w_2 we will compute $-w_0+w_1*x_1+w_2*x_2$ for all combinations of (x_1, x_2) and note down how many errors we make

Substitute 1.1, 1.1 here and then,

put in all the four combinations for x1, x2 and

realized that I am able to satisfy all of them.

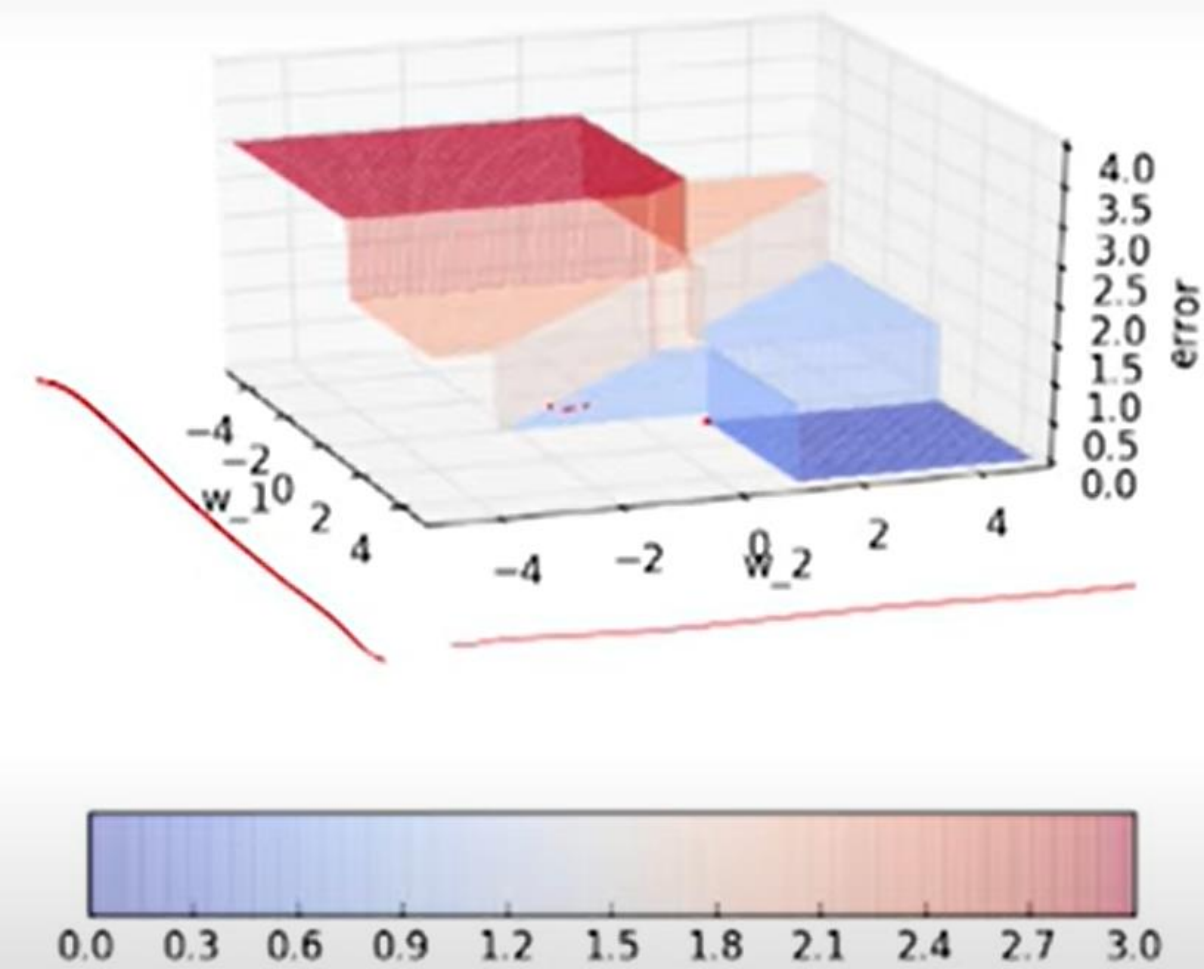
(i.e.) no error.

Now, instead of that if I had put something different?

Form the previous slide on this,

-1, -1

is in the light blue region.



In this light blue region,

the error is 1.

I made an error for one of the inputs.

This is a brute force way of finding.

Will not work because we have lots of inputs to check.

This is to just give an intuition

that we are looking at errors and

we are trying to find a value of w1, w2

which minimizes this error.

So, that is the idea behind errors and error surfaces.

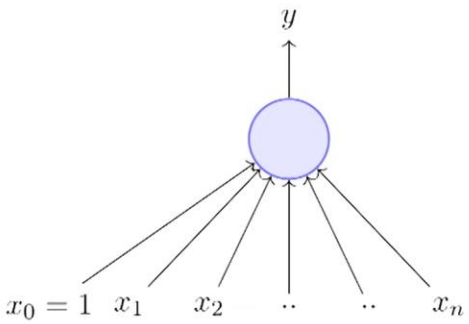
3.3 Perceptron learning algorithm

We will now see a more principled approach for learning these weights and threshold but before that let us answer this question...

Apart from implementing boolean functions (which does not look very interesting) what can a perceptron be used for ?

Our interest lies in the use of perceptron as a binary classifier. Let us see what this means...

The movie eg again but slightly more complicated.
Instead of three, n variables,
some are real-valued.
The rating could be any real number between 0 to 1.



$x_1 = isActorDamon$
$x_2 = isGenreThriller$
$x_3 = isDirectorNolan$
$x_4 = imdbRating(scaled\ to\ 0\ to\ 1)$
... ...
$x_n = criticsRating(scaled\ to\ 0\ to\ 1)$

- Let us reconsider our problem of deciding whether to watch a movie or not
- Suppose we are given a list of m movies and a label (class) associated with each movie indicating whether the user liked this movie or not : binary decision
- Further, suppose we represent each movie with n features (some boolean, some real valued)
- We will assume that the data is linearly separable and we want a perceptron to learn how to make this decision

In other words, we want the perceptron to find the equation of this separating plane (or find the values of $w_0, w_1, w_2, ..., w_m$)

If we say that the perceptron has learnt properly,
what would you expect it to do?
Perfect match.
Whenever we feed one of these movies,
it should give the same label
as is there in the data.
Again, there are
some movies for which the label is 1, which are positive, and
some movies which I have a label 0.
So, again, looking to separate the positives from the negatives.
So, it should adjust the weights in such a way that

it separates them.
That is the learning problem that we are interested in.

Now, the learning algo:

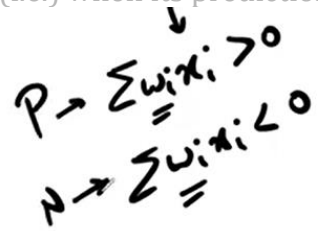
P are the positive inputs and
N, the negative.
We don't know what the weights are and
have no prior knowledge of what they are going to be.
Need to learn them from the data.
Initialize them with random values.

Algorithm: Perceptron Learning Algorithm

$P \leftarrow$ *inputs with label* 1;
$N \leftarrow$ *inputs with label* 0;
Initialize $\mathbf{w} = [w_0, w_2, \ldots, w_n]$ randomly;
while *!convergence* **do**
  Pick random $\mathbf{x} \in P \cup N$;
  if $\mathbf{x} \in P$ *and* $\sum_{i=0}^{n} w_i * x_i < 0$ **then**
    $\mathbf{w} = \mathbf{w} + \mathbf{x}$;
  end
  if $\mathbf{x} \in N$ *and* $\sum_{i=0}^{n} w_i * x_i \geq 0$ **then**
    $\mathbf{w} = \mathbf{w} - \mathbf{x}$;
  end
end
//the algorithm converges when all the inputs are classified correctly

- Why would this work ?
- To understand why this works we will have to get into a bit of Linear Algebra and a bit of geometry...

while not convergence, repeat:
But what's convergence?
When will we say it has converged?
When it is not making any more errors on the training data.
(i.e.) when its predictions are not changing on the training data.



Now, the algorithm.
Pick a random point from the data--could be positive or negative.
(data is the union of positive and negative).
If the point is positive, what is the condition we want?
The weighted sum >= 0.

But if the sum is < 0, the algo has made an error.
Then, correct it: just add x to w.
What is the dimensionality of w?
n but including w0, it's n+1.
x is also of the same dimensions
and hence, w+x is valid.
If the point is negative, what is the condition we want?
The weighted sum < 0.
But if the sum is >= 0, the algo has made an error.
Then, just subtract x from w.

Intuition for why this works.

Using two vectors that look similar to what we've--w and x.

- Consider two vectors **w** and **x**

$$\mathbf{w} = [w_0, w_1, w_2, ..., w_n]$$
$$\mathbf{x} = [1, x_1, x_2, ..., x_n]$$
$$\mathbf{w} \cdot \mathbf{x} = \mathbf{w}^\mathbf{T}\mathbf{x} = \sum_{i=0}^{n} w_i * x_i$$

This summation is just the dot product.

- We can thus rewrite the perceptron rule as

$$y = 1 \quad if \quad \mathbf{w}^\mathbf{T}\mathbf{x} \geq 0$$
$$= 0 \quad if \quad \mathbf{w}^\mathbf{T}\mathbf{x} < 0$$

using dot product instead of the summation.

- We are interested in finding the line $\mathbf{w}^\mathbf{T}\mathbf{x} = 0$ which divides the input space into two halves

$w^Tx = 0$ is the decision boundary.

- Every point ($\mathbf{x}$) on this line satisfies the equation $\mathbf{w}^\mathbf{T}\mathbf{x} = 0$ trivially.

Instead of y = mx + c, we have here x2 = x1 + w0 or w1x1 + w2x2 + w0 = 0

Eg. Suppose we've a line x1+x2=0

Every point on this line satisfies the equation.

Eg. 1, -1

2, -2

but not 2,2.

- What can you tell about the angle (α) between **w** and any point ($\mathbf{x}$) which lies on this line ?

Perpendicular.

When the dot product is 0, the vectors are orthogonal.

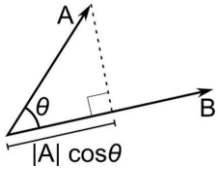
=> If we take this line,

vector w is orthogonal to every point and

hence, the line itself.

Angles (In Degrees)	0°	30°	45°	60°	90°	180°	270°	360°
cos	1	√3/2	1/√2	1/2	0	-1	0	1

- Since the vector **w** is perpendicular to every point on the line it is actually perpendicular to the line itself
- The angle is 90° ($\because cos\alpha = \frac{w^T x}{\|w\|\|x\|} = 0$) since



and a.b = ||a|| ||b|| cosθ where

a and b are the (Euclidean) magnitudes and

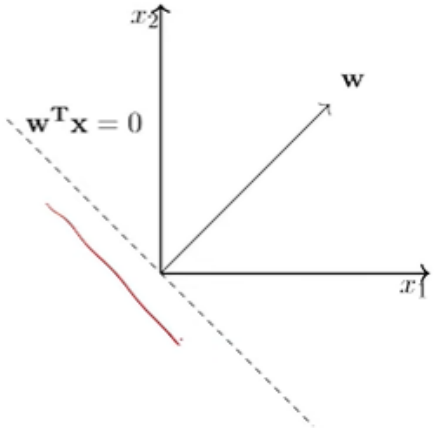
θ is the angle between the vectors,

in the direction of the vectors.

||a|| is the *norm* of *a*.

The geometric interpretation:

the decision boundary $w^T x$ is

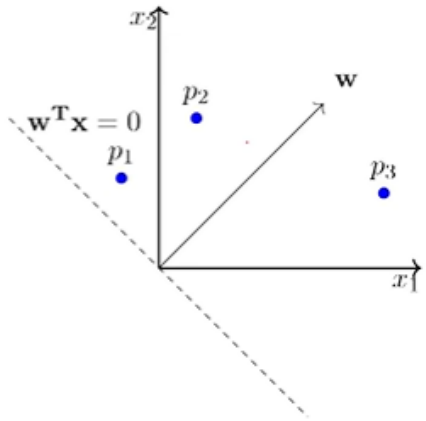


The vector w is actually orthogonal to this line.

That is the intuition built so far.

- Consider some points (vectors) which lie in the positive half space of this line (*i.e.*, $\mathbf{w^T x} \geq 0$)

These are the points for which the output is actually 1.

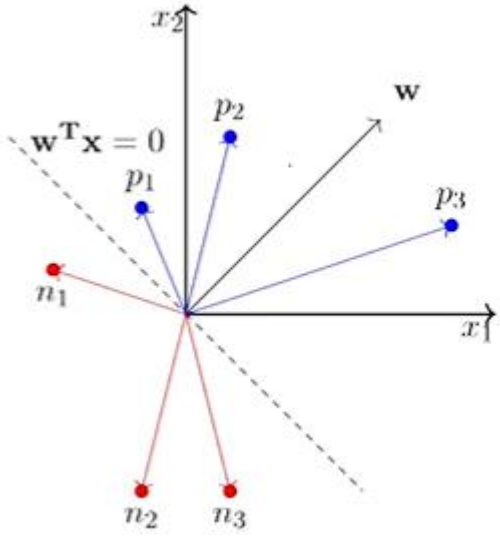


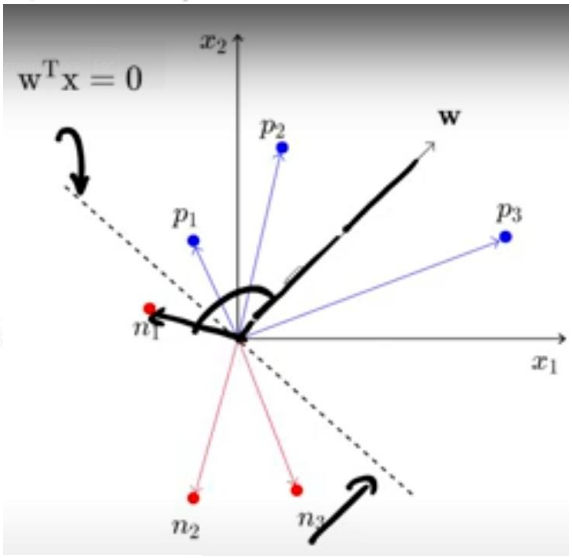
What is the angle between any of these points and w?

> 90, = 90 or < 90?

Obviously, <90.

- What about points (vectors) which lie in the negative half space of this line (*i.e.*, $\mathbf{w}^T\mathbf{x} < 0$)





- What will be the angle between any such vector and $\mathbf{w}$? Obviously, greater than 90°
- Of course, this also follows from the formula ($cos\alpha = \frac{w^T x}{||w||||x||}$)

For the positive points,

$w^Tx > 0$ => $\cos \alpha > 0$ => $\alpha < 90^0$

For the negative points,

$w^Tx < 0$ => $\cos \alpha < 0$ => $\alpha > 90^0$

Angle in degree	0	60	90	120	180
cos	1	1/2	0	-1/2	-1

Keeping this picture in mind, let's revisit the algo.

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

- For $\mathbf{x} \in P$ if $\mathbf{w}.\mathbf{x} < 0$ then it means that the angle (α) between this $\mathbf{x}$ and the current $\mathbf{w}$ is greater than 90° (but we want α to be less than 90°)

$$cos\alpha = \frac{\mathbf{w}^T\mathbf{x}}{||\mathbf{w}||||\mathbf{x}||}$$

Let's look at the first condition.

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

the angle between x and the current w is actually> 90 degrees

but what do we want it to be?

< 90.

Our solution:

- What happens to the new angle (α_{new}) when $\mathbf{w_{new}} = \mathbf{w} + \mathbf{x}$

Why does this work?

What's the cos(α_{new})?

Proportional to

$$cos(\alpha_{new}) \propto \mathbf{w_{new}}^T\mathbf{x}$$

Substitute w_{new}

$$\begin{aligned} cos(\alpha_{new}) &\propto \mathbf{w_{new}}^T\mathbf{x} \\ &\propto (\mathbf{w}+\mathbf{x})^T\mathbf{x} \\ &\propto \mathbf{w}^T\mathbf{x} + \mathbf{x}^T\mathbf{x} \\ &\propto cos\alpha + \mathbf{x}^T\mathbf{x} \\ cos(\alpha_{new}) &> cos\alpha \end{aligned}$$

where $\mathbf{x}^T\mathbf{x}$ is positive.

If cos(α_{new}) is > cos(α), what's α_{new}?

It will be < α.

That's exactly what we want.

This angle was actually > 90^0 and

we want to slowly move it such that it becomes < 90^0.

Not solved in one iteration and that's why "till convergence".

Keep picking the x's till convergence.

Till we are satisfied with that condition.

The other condition.

- For $\mathbf{x} \in N$ if $\mathbf{w}.\mathbf{x} \geq 0$ then it means that the angle (α) between this $\mathbf{x}$ and the current $\mathbf{w}$ is less than 90° (but we want α to be greater than 90°)
- What happens to the new angle (α_{new}) when $\mathbf{w_{new}} = \mathbf{w} - \mathbf{x}$

$$
\begin{aligned}
cos(\alpha_{new}) &\propto \mathbf{w_{new}}^T\mathbf{x} \\
&\propto (\mathbf{w} - \mathbf{x})^T\mathbf{x} \\
&\propto \mathbf{w}^T\mathbf{x} - \mathbf{x}^T\mathbf{x} \\
&\propto cos\alpha - \mathbf{x}^T\mathbf{x} \\
cos(\alpha_{new}) &< cos\alpha
\end{aligned}
$$

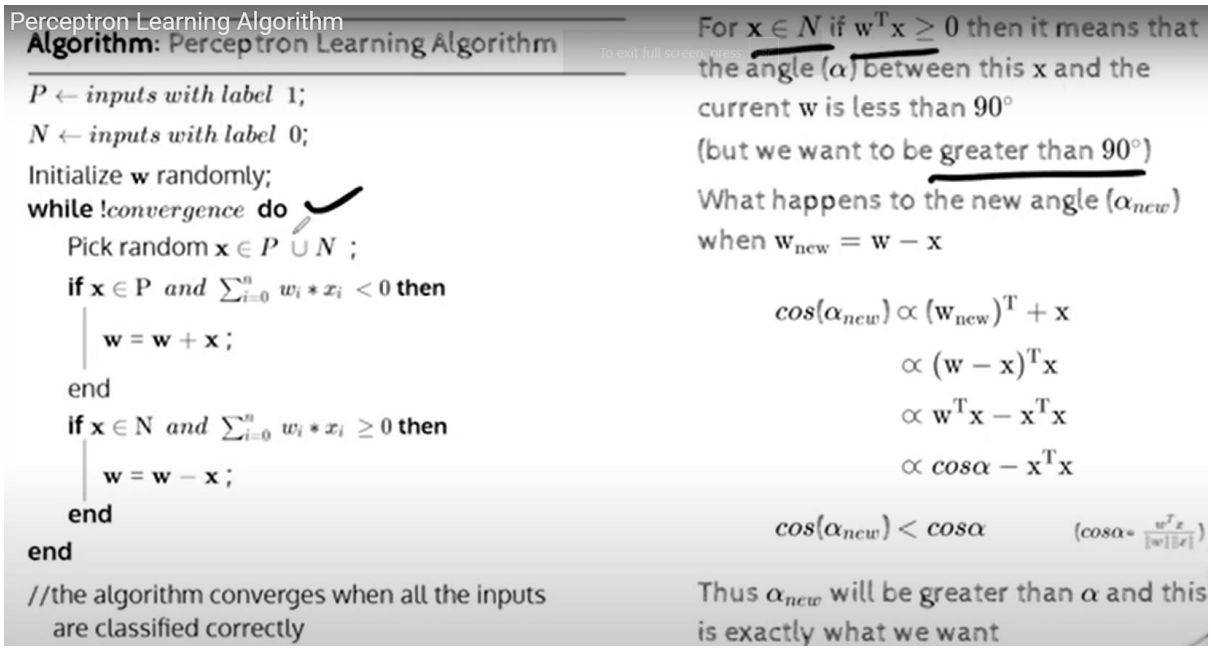
where $\mathbf{x}^T\mathbf{x}$ is positive.

We are conveniently ignoring the denominator and making it proportional to make it easier to show the intuition.

- Thus α_{new} will be greater than α and this is exactly what we want

The new cos(α) is going to be < the original cos(α);
means, the angle is going to be greater and
that's exactly what we wanted.

The other error case:



Something more is there?!

How do we not keep oscillating by keeping increasing and decreasing w (i.e.) whether this algo will converge or not..

Later.

Take a point, adjust w,
but now for the next point, may go back to the same w
because that point asked me to move it again.
Keep doing this again and again and
may never converge.

Not the case.
The algorithm will converge
because of the bounds on the cos value.

Coming back to our questions

What about non-boolean (say, real) inputs? **Real valued inputs are allowed in perceptron**

Do we always need to hand code the threshold? **No, we can learn the threshold**

Are all inputs equal? What if we want to assign more weight (importance) to some inputs?
A perceptron allows weights to be assigned to inputs

What about functions which are not linearly separable ? **Not possible with a single perceptron but we will see how to handle this ..**