CS167: Machine Learning

Linear Classifiers
Perceptron
Code for Perceptron

Tuesday, March 26th, 2024

Announcements

- Quiz # 2
  - will be released on Thursday **03/28/24**
  - due next Thursday **04/04/24 by 11:59pm**

Today's Agenda

- Perceptron

- Perceptron code

Linear Classifier

- In n-dimensional space the following equation represents a linear classifier

  $w_0+w_1*x_1+\ldots+w_n*x_n = 0$

  is a hyperplane

- Idea:
  - Use: $w_0+w_1*x_1+\ldots+w_n*x_n > 0$ to denote positive classifications
  - Use: $w_0+w_1*x_1+\ldots+w_n*x_n \leq 0$ to denote negative classifications

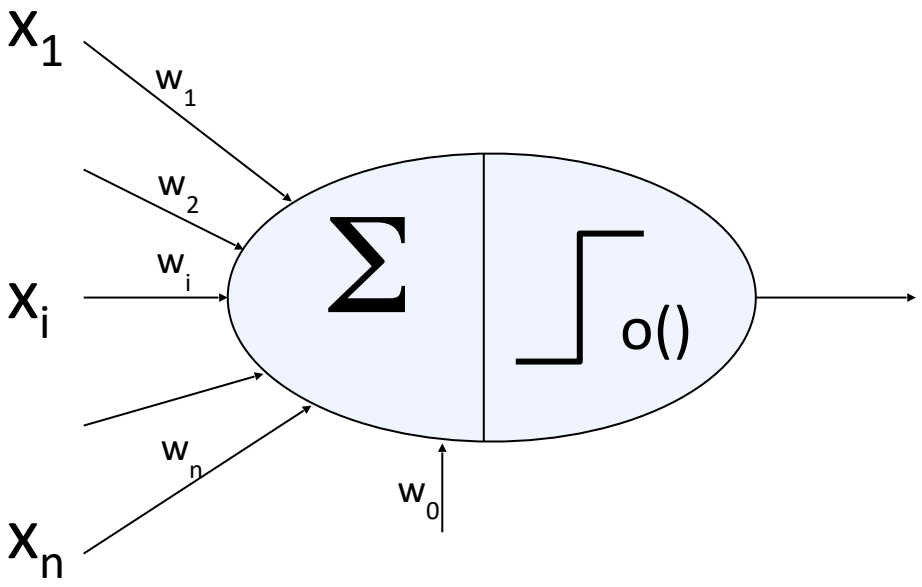
Perceptron

- The mathematical model of a *single neuron* is called a **perceptron**

- It has a two components:
  - Component 1: a linear model of the form we just saw in the previous slide

$$w_0+w_1*x_1+\ldots+w_n*x_n$$

  - Component 2: **a step function** which will produce 1 if the function value is positive and -1 otherwise

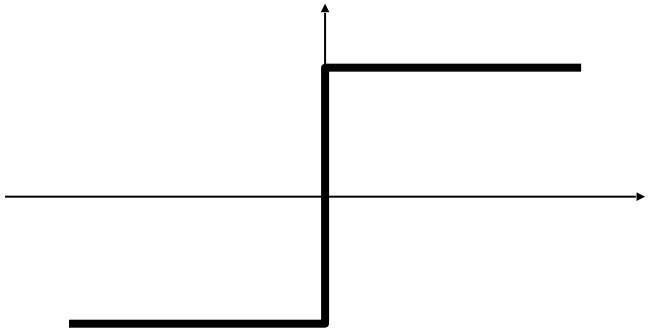


Big Question?
How do we determine the weights that best classify the data?

$$y = f(\mathbf{x},\mathbf{w}) = o\left(\Sigma_{i=1,\ldots,n}\, w_i\, x_i\right)$$

Perceptron

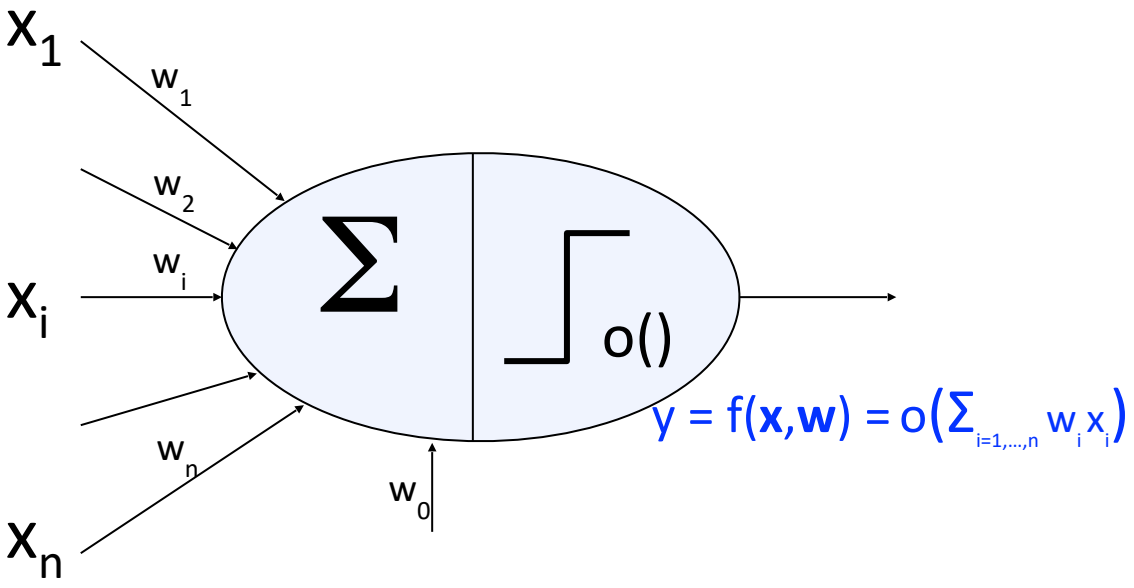
- The mathematical model of a *single neuron* is called a **perceptron**
- It has a two components:
  - Component 1: A **linear model** of the form we just saw in the previous slide
  - Component 2: **a step function** which will produce 1 if the function value is positive and -1 otherwise

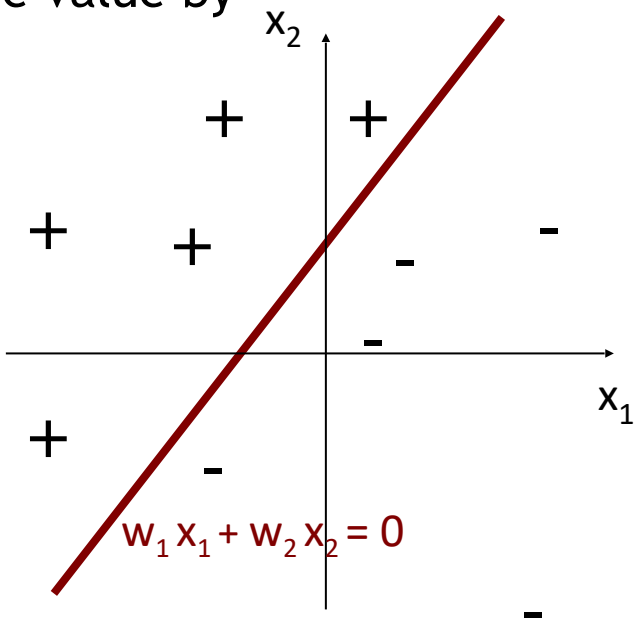


$$y = f(\mathbf{x},\mathbf{w}) = o\left(\Sigma_{i=1,\ldots,n}\, w_i\, x_i\right)$$

Perceptron

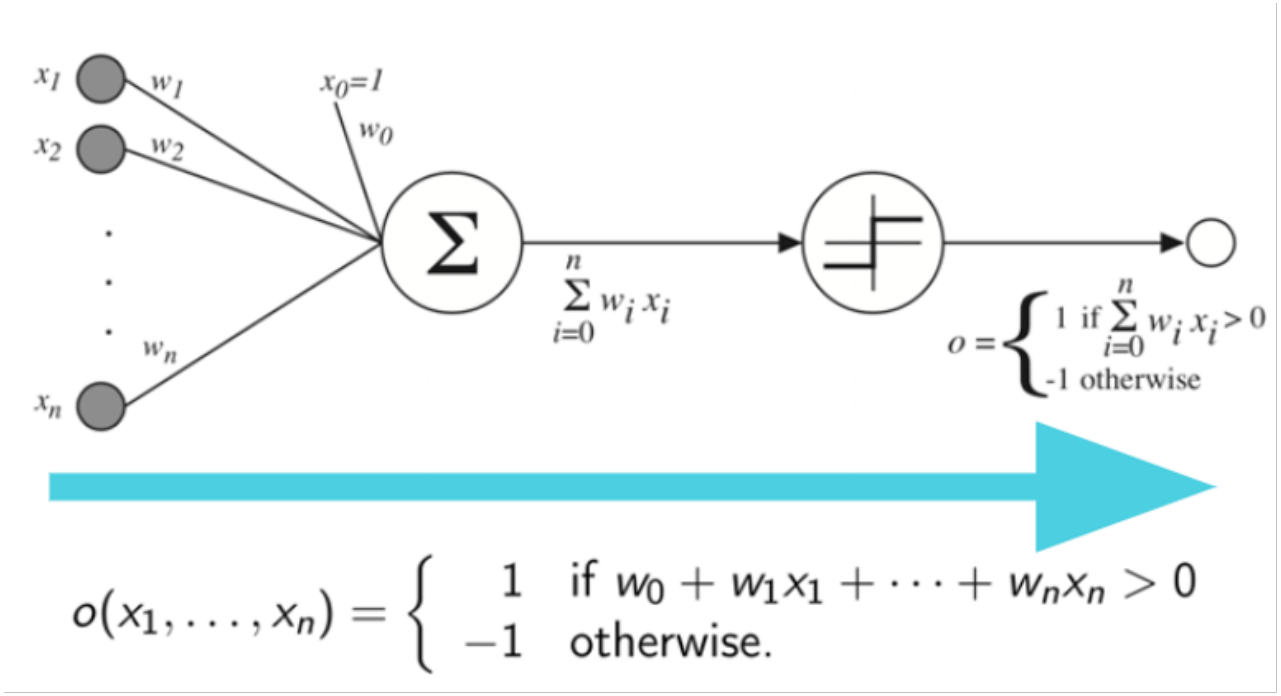
- when you are given some training samples, you can consider
  - each x_i is one of the predictor features --i.e. *sepal length, age, etc*
  - each w_i is some weight we multiply the feature value by





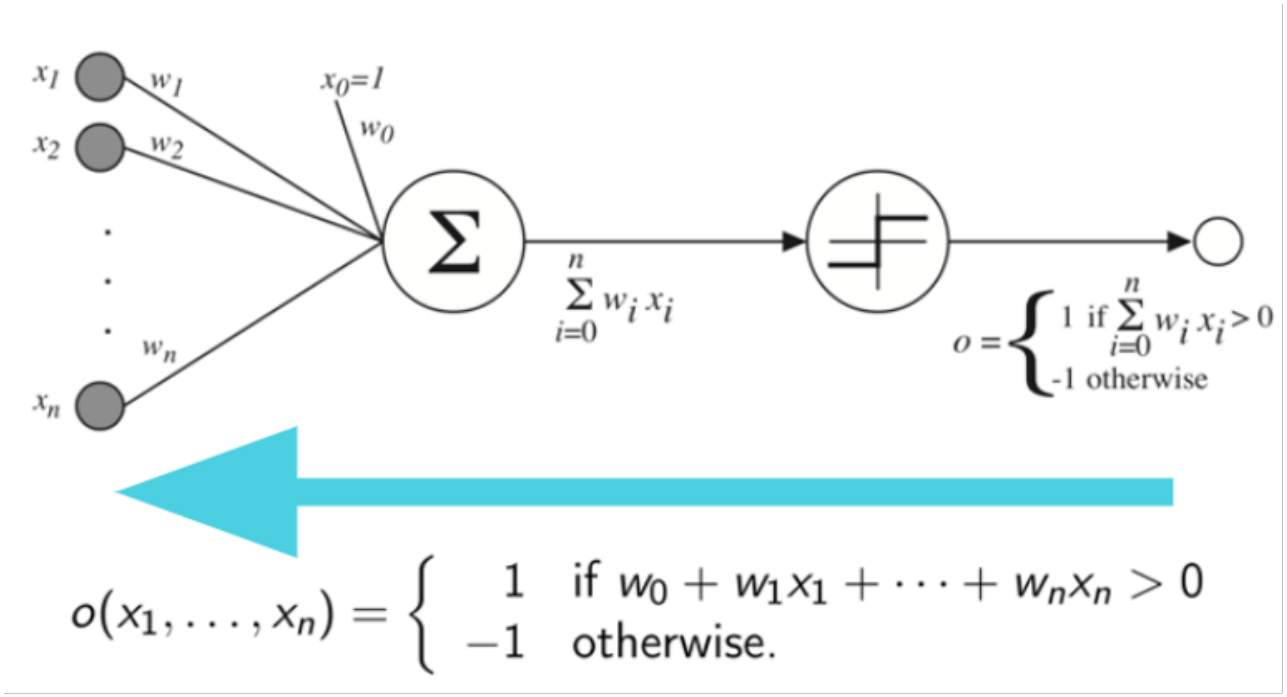
Forward Step

- The **forward step** in a perceptron is the step of receiving input, and getting an output based on the input. In other words, making a prediction from an input. This step is also known as the *inference step*



Backward Step

- The **backward step** in a perceptron is the step of checking to see if the model got the prediction correct, and if not, adjusting the weights to make a better prediction next time. We also call this step *updating the weights*



How to train your Perceptron

- How do we determine the weights that best classify the data?
- One strategy could be as follows:
  - Start with 0 values for all weights
  - Feed in a training example
    - If it is properly classified; great, move on
    - Else; update weights according to the training example
    - Repeat

How to train your Perceptron

- Feed a training example into the perceptron, then for each weight

$$\Delta w_i = \eta(t - o)x_i$$
$$w_i^{new} = w_i^{old} + \Delta w_i$$

t = target

o = output

- **target** is the thing in the target column for this example - what you're trying to predict
- **output** is the perceptron output - what the perceptron is currently set to predict for this example
- η**(eta)** : is a small constant (e.g. 0.1) called the **learning rate** (controls how quickly the model adapts to the training data)

How to train your Perceptron

- Feed a training example into the perceptron, then for each weight

$$\Delta w_i = \eta(t - o)x_i$$

$$w_i^{new} = w_i^{old} + \Delta w_i$$

t = target

o = output

- Note what happens when:
  - target and output are both 1
  - target and output are both -1
  - target is 1, output is -1
  - target is -1, output is 1

Group Activity on Perceptron Learning Rule

- Here is a sample dataset of binary classification problem:
  - `underground explosion vs. earthquake`

Seismometer Data		
Body Wave Magnitude	**Surface Wave Magnitude**	**Classification**
5.2	3.4	underground explosion
5.8	3.5	underground explosion
5.9	4.4	underground explosion
6.1	4.1	underground explosion
5.2	5	earthquake
4.5	4.9	earthquake
5.3	4.2	earthquake
5.5	5.5	earthquake
6.1	5.8	earthquake
5.6	3.9	?
5.7	4.6	?
4.7	4.7	?
6.4	4.5	?

Group Activity

Body Wave Mag X_1	Surface Wave Mag x_2	Classification	Target (-1=earthquake, +1=explosion)
5.2	3.4	Explosion	1
4.5	4.9	Earthquake	-1
6.1	5.8	Earthquake	-1
5.6	3.9	?	?

initial value for weight parameter w_0 { $w_0 = 0$

initial value for weight parameter w_1 { $w_1 = 0$

initial value for weight parameter w_2 { $w_2 = 0$

fixed value for learning rate parameter { $\eta = 0.1$

- Use the first training example and apply the perceptron learning rule to find the weight parameter values again

$$\Delta w_i = \eta(t - o)x_i$$
$$w_i^{new} = w_i^{old} + \Delta w_i$$

Iteration 1: Apply Forward Step

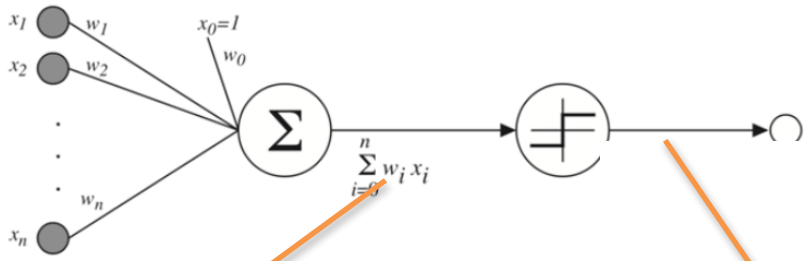
$w_0^{old} = 0$

$w_1^{old} = 0$

$w_2^{old} = 0$

Body Wave Mag X_1	Surface Wave Mag x_2	Classification	Target (-1=earthquake, +1=explosion)
5.2	3.4	Explosion	1

$\eta = 0.1$



x_0 is always 1

Constant	Body wave mag	Surface wave mag	Neuron's linear model output	Neuron's step function output: *o*	target: *t*
x_0	x_1	x_2	$w_0x_0 + w_1x_1 + w_2x_2$	$o = \begin{cases} +1 & : w_0x_0 + w_1x_1 + w_2x_2 > 0 \\ -1 & : w_0x_0 + w_1x_1 + w_2x_2 \leq 0 \end{cases}$	explosion vs. earthquake
1	5.2	3.4	0*1+0*5.2+0*3.4 = 0	-1	1

Iteration 1: Backward Step

$w_0^{old} = 0$

$w_1^{old} = 0$

$w_2^{old} = 0$

Body Wave Mag X_1	Surface Wave Mag x_2	Classification	Target (-1=earthquake, +1=explosion)
5.2	3.4	Explosion	1

$\eta = 0.1$

$$\Delta w_i = \eta(t - o)x_i$$

$$w_i^{new} = w_i^{old} + \Delta w_i$$

computation for weight parameter W_0

η	target: t	Neuron's output: o	x_0	$\Delta w_0 = \eta(t-o)x_0$	$w_o^{new} = w_o^{old} + \Delta w_0$
0.1	1	-1	1	0.1x(1-(-1))x1= 0.1x2x1= 0.2	0+0.2= 0.2

computation for weight parameter W_1

η	target: t	Neuron's output: o	x_1	$\Delta w_1 = \eta(t-o)x_1$	$w_1^{new} = w_1^{old} + \Delta w_1$
0.1	1	-1	5.2	0.1x(1-(-1))x5.2= 0.1x2x5.2= 1.04	0+1.04= 1.04

computation for weight parameter W_2

η	target: t	Neuron's output: o	x_2	$\Delta w_2 = \eta(t-o)x_2$	$w_2^{new} = w_2^{old} + \Delta w_2$
0.1	1	-1	3.4	0.1x(1-(-1))x3.4= 0.1x2x3.4= 0.68	0+0.68= 0.68

Results After Iteration 1

Body Wave Mag X_1	Surface Wave Mag x_2	Classification	Target (-1=earthquake, +1=explosion)
4.5	4.9	Earthquake	-1

updated weight parameter w_0 → $w_0 = 0.2$

updated weight parameter w_1 → $w_1 = 1.04$

updated weight parameter w_2 → $w_2 = 0.68$

fixed value for learning rate parameter → $\eta = 0.1$

- Now use the second training example and apply the perceptron learning rule to find the weight parameter values again

$$\Delta w_i = \eta(t - o)x_i$$
$$w_i^{new} = w_i^{old} + \Delta w_i$$

Iteration 2: Apply Forward Step

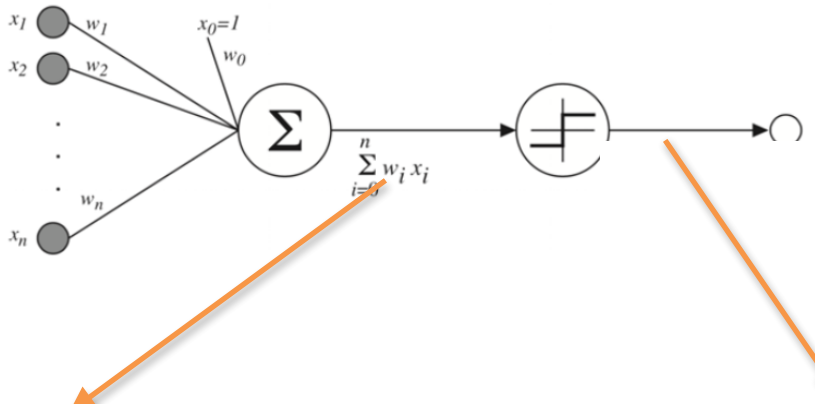
w_0^{old} = 0.2

w_1^{old} = 1.04

w_2^{old} = 0.68

Body Wave Mag X_1	Surface Wave Mag x_2	Classification	Target (-1=earthquake, +1=explosion)
4.5	4.9	Earthquake	-1

η = 0.1



Constant	Body wave mag	Surface wave mag	Neuron's linear model output	Neuron's step function output: *o*	target: *t*
x_0	x_1	x_2	$w_0x_0 + w_1x_1 + w_2x_2$	$o = \begin{cases} +1 & : w_0x_0 + w_1x_1 + w_2x_2 > 0 \\ -1 & : w_0x_0 + w_1x_1 + w_2x_2 \leq 0 \end{cases}$	explosion vs. earthquake
1	4.5	4.9	0.2*1+1.04*4.5+0.68*4.9 = 8.2	1	-1

Iteration 2: Backward Step

$w_0^{old} = 0.2$

$w_1^{old} = 1.04$

$w_2^{old} = 0.68$

Body Wave Mag X_1	Surface Wave Mag x_2	Classification	Target (-1=earthquake, +1=explosion)
4.5	4.9	Earthquake	-1

$\eta = 0.1$

$$\Delta w_i = \eta(t - o)x_i$$
$$w_i^{new} = w_i^{old} + \Delta w_i$$

computation for weight parameter W_0

η	target: t	Neuron's output: o	x_0	$\Delta w_0 = \eta(t-o)x_0$	$w_o^{new} = w_o^{old} + \Delta w_0$
0.1	-1	1	1	0.1x(-1-1)x1= 0.1x(-2)x1= -0.2	0.2+(-0.2)= 0

computation for weight parameter W_1

η	target: t	Neuron's output: o	x_1	$\Delta w_1 = \eta(t-o)x_1$	$w_1^{new} = w_1^{old} + \Delta w_1$
0.1	-1	1	4.5	0.1x(-1-1)x4.5= 0.1x(-2)x4.5= -0.9	1.04+(-0.9)= 0.14

computation for weight parameter W_2

η	target: t	Neuron's output: o	x_2	$\Delta w_2 = \eta(t-o)x_2$	$w_2^{new} = w_2^{old} + \Delta w_2$
0.1	-1	1	4.9	0.1x(1-(-1))x4.9= 0.1x(-2)x4.9= -0.98	0.68+(-0.98)= -0.30

Results After Iteration 2

Body Wave Mag X_1	Surface Wave Mag x_2	Classification	Target (-1=earthquake, +1=explosion)
6.1	5.8	Earthquake	-1

Your turn to update weight parameters using the last example

updated weight parameter w_0 { $w_0 = 0.0$

updated weight parameter w_1 { $w_1 = 0.14$

updated weight parameter w_2 { $w_2 = -0.3$

fixed value for learning rate parameter { $\eta = 0.1$

- Now use the third training example and apply the perceptron learning rule to find the weight parameter values again

$$\Delta w_i = \eta(t - o)x_i$$
$$w_i^{new} = w_i^{old} + \Delta w_i$$

Your Turn: Iteration 3: Apply Forward Step

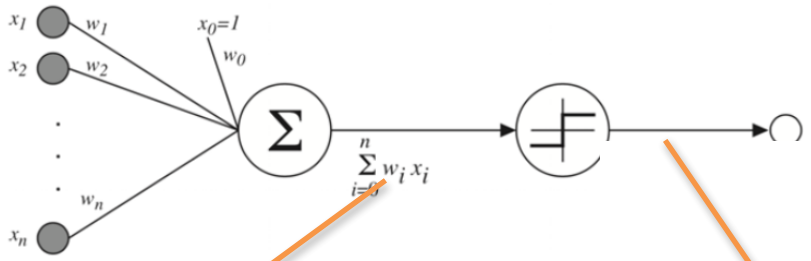
w_0^{old} = 0.0

w_1^{old} = 0.14

w_2^{old} = -0.3

Body Wave Mag X_1	Surface Wave Mag x_2	Classification	Target (-1=earthquake, +1=explosion)
6.1	5.8	Earthquake	-1

η = 0.1



Constant	Body wave mag	Surface wave mag	Neuron's linear model output	Neuron's step function output: *o*	target: *t*
x_0	x_1	x_2	$w_0x_0 + w_1x_1 + w_2x_2$	$o = \begin{cases} +1 & : w_0x_0 + w_1x_1 + w_2x_2 > 0 \\ -1 & : w_0x_0 + w_1x_1 + w_2x_2 \leq 0 \end{cases}$	explosion vs. earthquake
1	6.1	5.8	?	?	-1

Your Turn: Iteration 3: Backward Step

w_0^{old} = 0.0

w_1^{old} = 0.14

w_2^{old} = -0.3

Body Wave Mag X_1	Surface Wave Mag x_2	Classification	Target (-1=earthquake, +1=explosion)
6.1	5.8	Earthquake	-1

η = 0.1

$$\Delta w_i = \eta(t - o)x_i$$
$$w_i^{new} = w_i^{old} + \Delta w_i$$

computation for weight parameter W_0

η	target: t	Neuron's output: o	x_0	$\Delta w_0 = \eta(t-o)x_0$	$w_o^{new} = w_o^{old} + \Delta w_0$
0.1	-1	?	1	?	?

computation for weight parameter W_1

η	target: t	Neuron's output: o	x_1	$\Delta w_1 = \eta(t-o)x_1$	$w_1^{new} = w_1^{old} + \Delta w_1$
0.1	-1	?	6.1	?	?

computation for weight parameter W_2

η	target: t	Neuron's output: o	x_2	$\Delta w_2 = \eta(t-o)x_2$	$w_2^{new} = w_2^{old} + \Delta w_2$
0.1	-1	?	5.8	?	?

Your Turn: Results After Iteration 3

Body Wave Mag X_1	Surface Wave Mag x_2	Classification	Target (-1=earthquake, +1=explosion)
6.1	5.8	Earthquake	-1

updated weight parameter W_0 { $w_0 = ?$

updated weight parameter W_1 { $w_1 = ?$

updated weight parameter W_2 { $w_2 = ?$

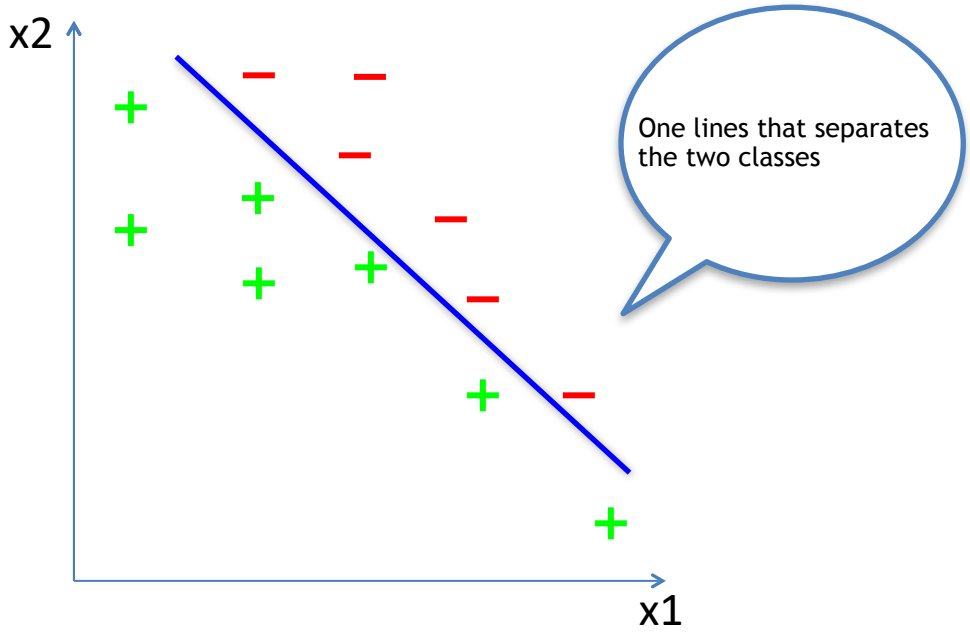
fixed value for learning rate parameter { $\eta = 0.1$

- Now use the third training example and apply the perceptron learning rule to find the weight parameter values again

$$\Delta w_i = \eta(t - o)x_i$$
$$w_i^{new} = w_i^{old} + \Delta w_i$$

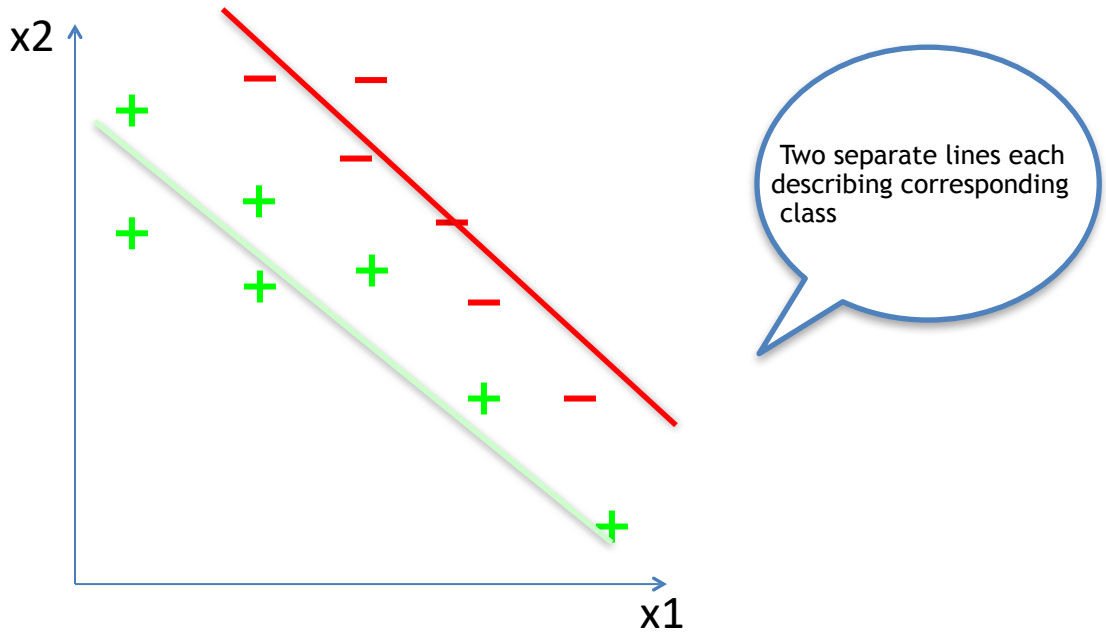
Discriminative Model for Classification

- **Discriminative Model:** Alternatively, we can build the decision functions (eg, line/plane/hyperplane) from the training samples using the ***difference between two classes***
  - For example, if we have two classes as shown below, we model one 2D line for class-1 examples and another 2D line for class-2 examples



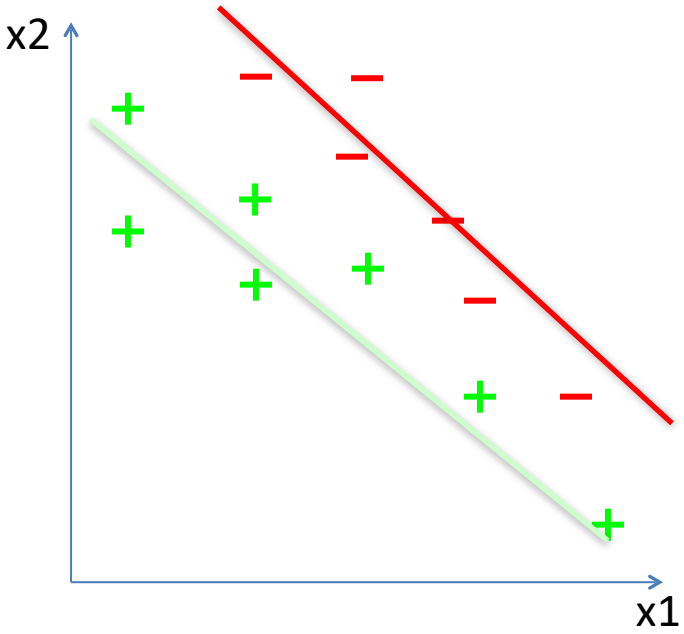
Generative Model for Classification

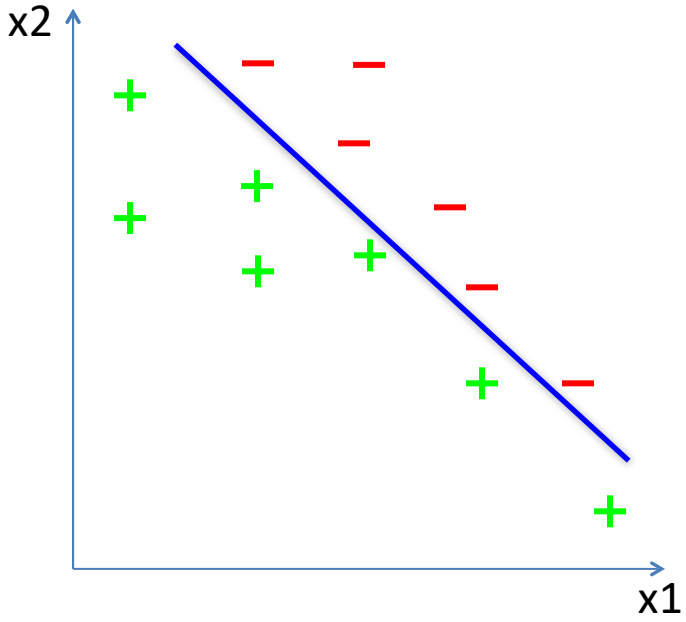
- **Generative Model:** We can build the decision functions (eg, line/plane/hyperplane) from the training samples using individual classes
  - For example, if we have two classes as shown below, we model one 2D line for class-1 examples and another 2D line for class-2 examples



Generative vs. Discriminative Model

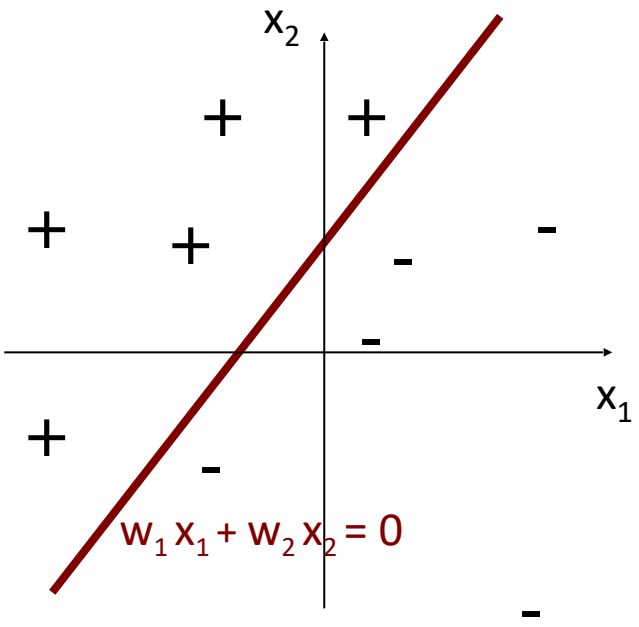
Generative vs **Discriminative**





Example: Generative vs. Discriminative

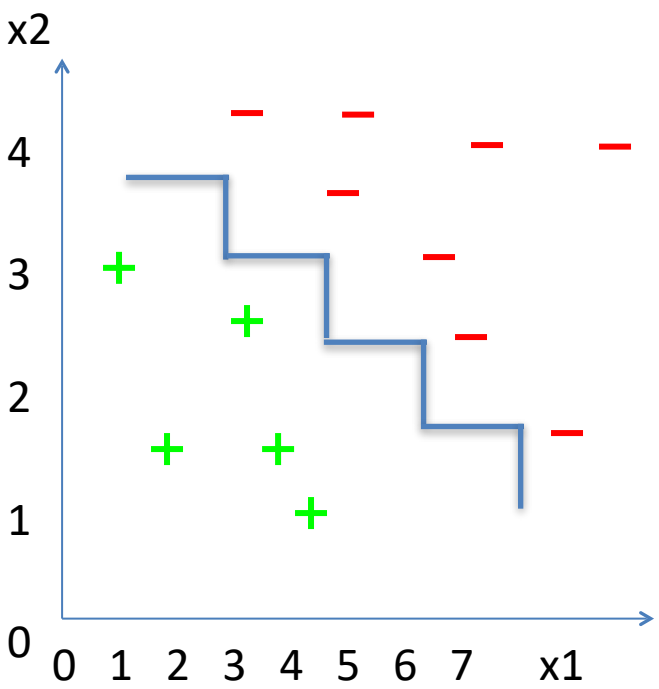
- Is **Perceptron** a generative model or a discriminative model?



Perceptron produces one lines (or a hyperplane) that separates the two classes

Example: Generative vs. Discriminative

- Is **decision tree (DT)** a generative model or a discriminative model?



DT produces collection of lines (axis-aligned decision boundaries) that separates the two classes

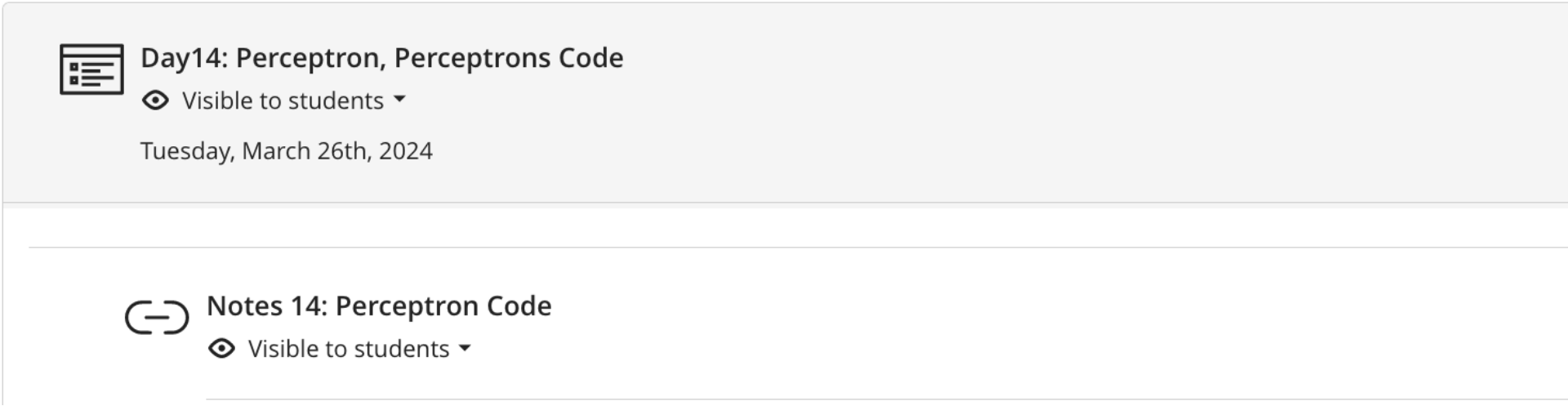
Today's Agenda

- Perceptron

- Perceptron code

Perceptron Code

- Go to Blackboard and open the notebook to do the coding activity



Perceptron Code

- Go to Blackboard and open the notebook to do the coding activity

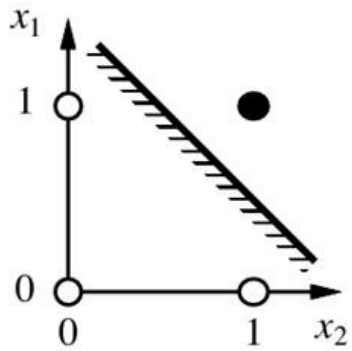
```
#load up scikit-learn Perceptron
from sklearn.linear_model import Perceptron
from sklearn import metrics
perc = Perceptron()
perc.fit(train_data,train_sln)
iris_perc_predictions = perc.predict(test_data)

#output accuracy
print("Setosa accuracy:", metrics.accuracy_score(test_sln, iris_perc_predictions
conf_mat = metrics.confusion_matrix(test_sln, iris_perc_predictions)
print("confusion matrix: \n", conf_mat)
print("confusion matrix display: \n")
disp = ConfusionMatrixDisplay(conf_mat)
disp.plot()
plt.show()
```

Let's Consider AND Function

- Let's consider the AND function.

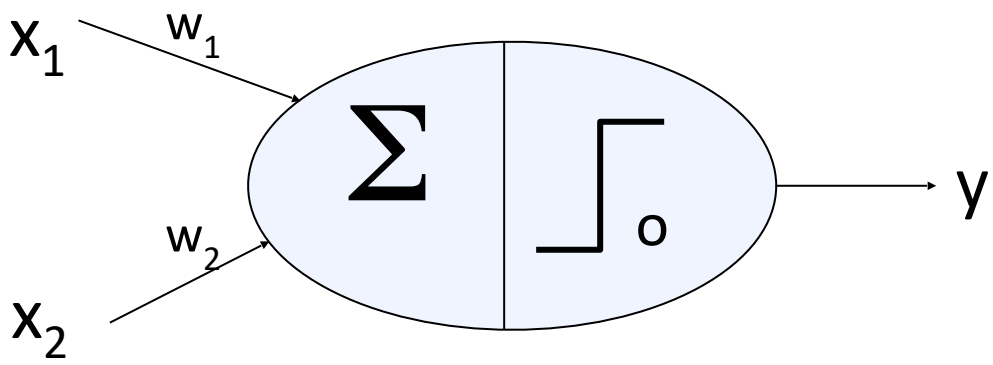
x	y	x and y
0	0	0
0	1	0
1	0	0
1	1	1



x_1 **and** x_2

YES, because we can find this 2D line

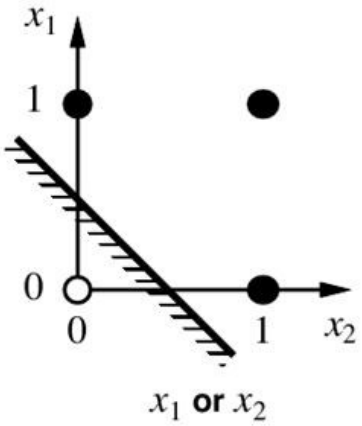
- Can a perceptron model AND function?



Let's Consider OR Function

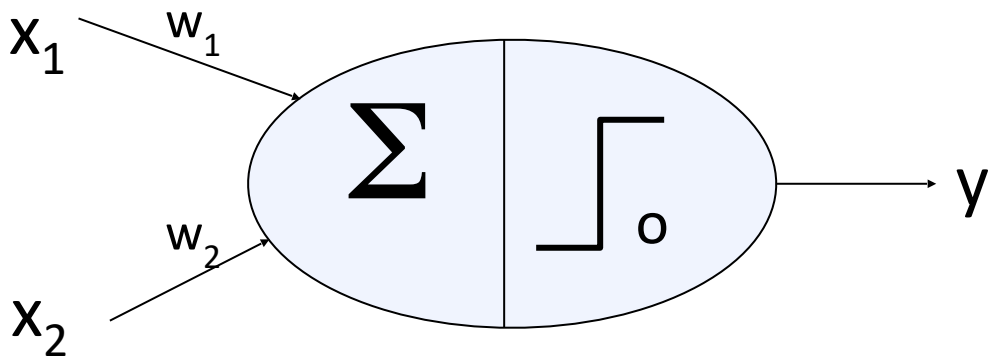
- Let's consider the OR function.

x	y	x or y
0	0	0
0	1	1
1	0	1
1	1	1

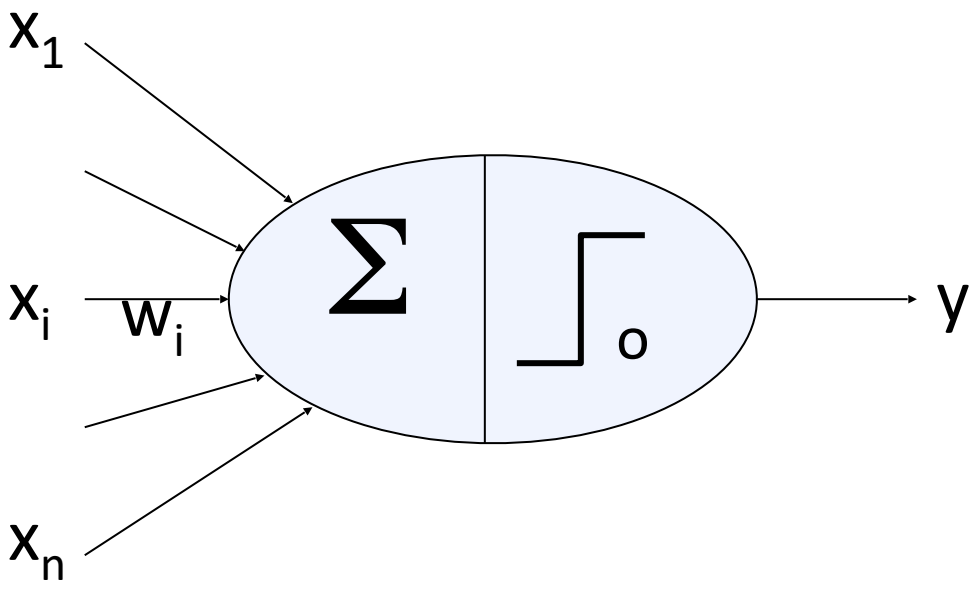


YES, because we can find this 2D line

- Can a perceptron model OR function?



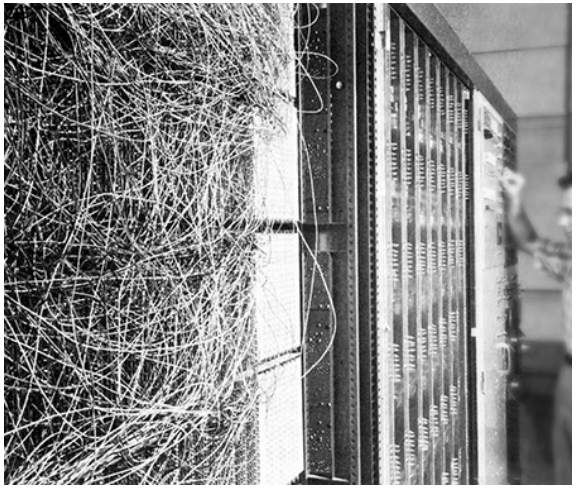
Can A Perceptron Model Any Function?

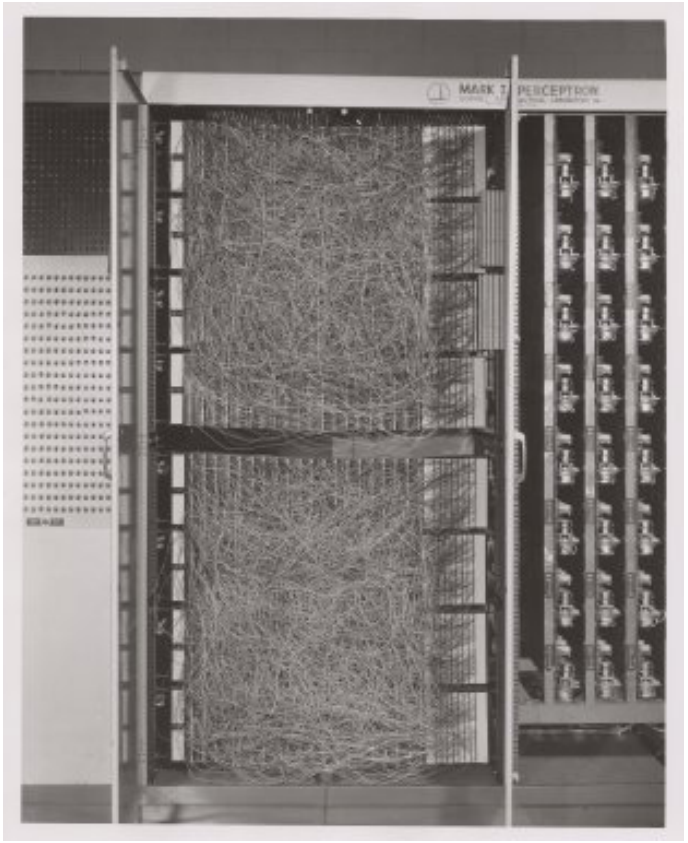


- Perceptron can also model other boolean functions such as ($x_1 \wedge x_2 \wedge \neg x_3$). We can tabulate all combinations of the three variables and their corresponding outputs; then find weight parameters (of the plane separating it two classes (1 and 0)) using perceptron update rule we just did

- But can a perceptron model any function?

Perceptron





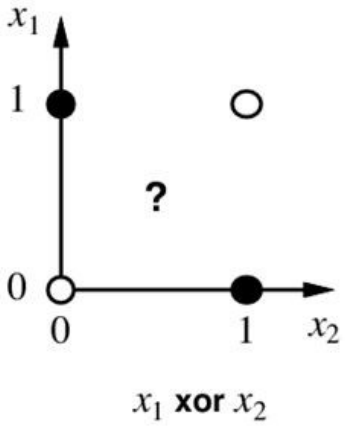
"the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence."

Frank Rosenblatt, 1958

Now Let's Consider XOR Function

- Let's consider the XOR function.

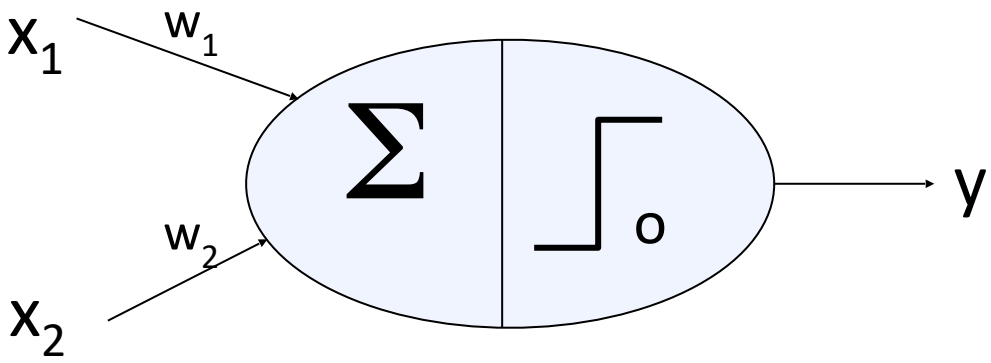
x	y	x xor y
0	0	0
0	1	1
1	0	1
1	1	0



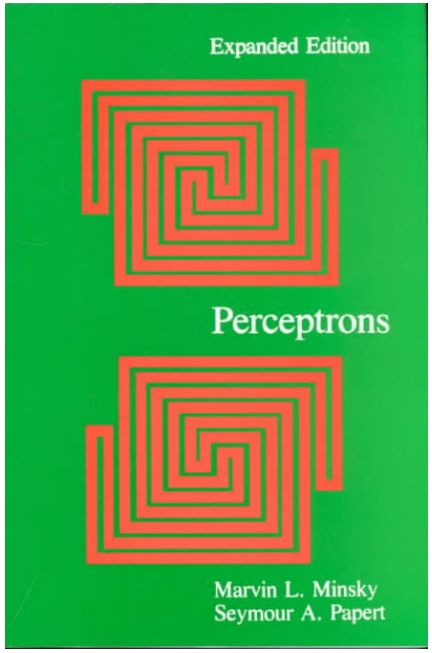
x_1 **xor** x_2

NO, because we can't find a single 2D line that separates the samples

- Can a perceptron model XOR function?



Perceptron



Marvin Minsky and Seymour Papert showed that they couldn't even learn XOR in 1969, which is why all the hype about the perceptron faded away