# PERCEPTRON LEARNING

David Kauchak
CS 158 – Fall 2019

## Admin

Assignment 1 grading

Assignment 2 Due Sunday at midnight

Meet with colloquium speaker today, 2:15-2:45pm in Edmunds 129

## Colloquium talk

**Detecting Bugs and Explaining Predictions of Machine Learning Models**

Machine learning is at the forefront of many recent advances in science and technology, enabled in part by the sophisticated models and algorithms that have been recently introduced. However, as a consequence of this complexity, machine learning essentially acts as a black-box as far as users are concerned, making it incredibly difficult to understand, predict, or detect bugs in their behavior. For example, determining when a machine learning model is "good enough" is challenging since held-out accuracy metrics significantly overestimate real-world performance. In this talk, I will describe our research on approaches that explain the predictions of any classifier in an interpretable and faithful manner, and automated techniques to detect bugs that can occur naturally when a model is deployed. In particular, these methods describe the relationship between the components of the input instance and the classifier's prediction. I will cover various ways in which we summarize this relationship: as linear weights, as precise rules, and as counter-examples, and present experiments to contrast them and evaluate their utility in understanding, and debugging, black-box machine learning algorithms, on tabular, image, text, and graph completion applications.

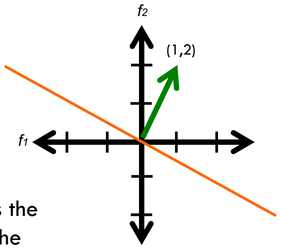
## Defining a line

Any pair of values ($w_1$,$w_2$) defines a line through the origin:

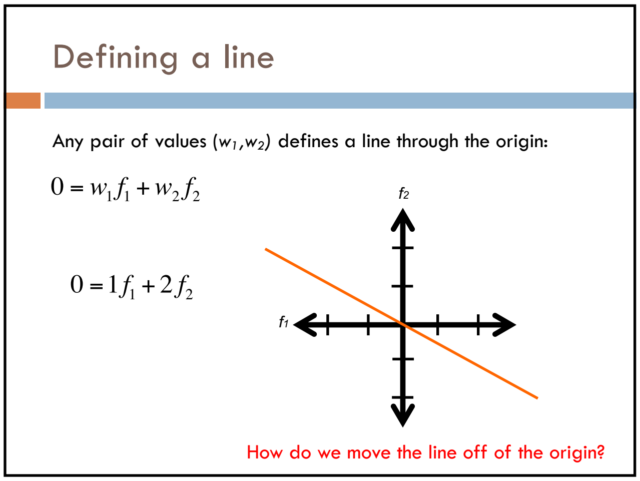
$$0 = w_1 f_1 + w_2 f_2$$

$$0 = 1 f_1 + 2 f_2$$

w=(1,2)



We can also view it as the line perpendicular to the *weight vector*



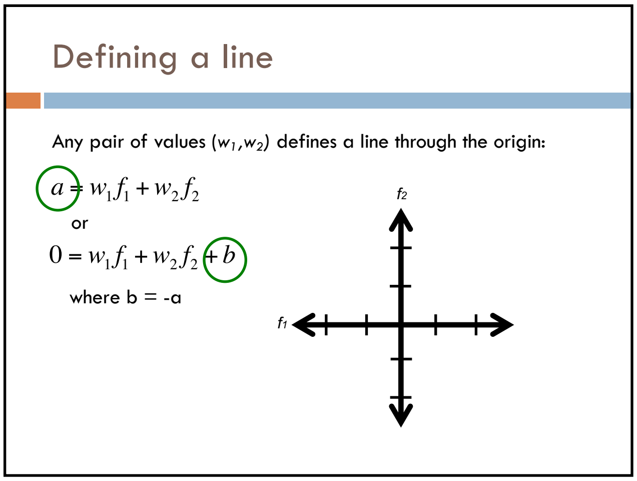
## Defining a line

Any pair of values ($w_1$,$w_2$) defines a line through the origin:

$$0 = w_1 f_1 + w_2 f_2$$

$$0 = 1 f_1 + 2 f_2$$

How do we move the line off of the origin?



## Defining a line

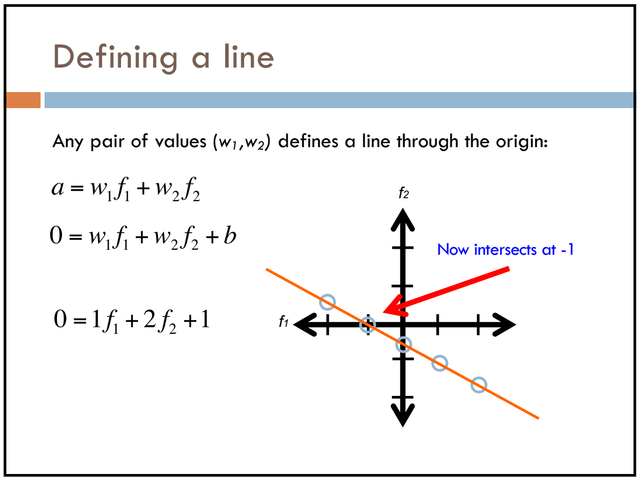
Any pair of values ($w_1$,$w_2$) defines a line through the origin:

$$a = w_1 f_1 + w_2 f_2$$

or

$$0 = w_1 f_1 + w_2 f_2 + b$$

where b = -a



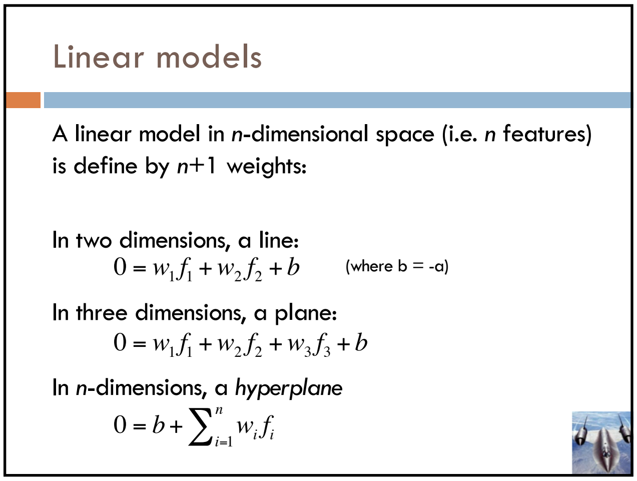
## Defining a line

Any pair of values ($w_1$,$w_2$) defines a line through the origin:

$$a = w_1 f_1 + w_2 f_2$$

$$0 = w_1 f_1 + w_2 f_2 + b$$

$$0 = 1 f_1 + 2 f_2 + 1$$



## Linear models

A linear model in *n*-dimensional space (i.e. *n* features) is define by *n*+1 weights:

In two dimensions, a line:

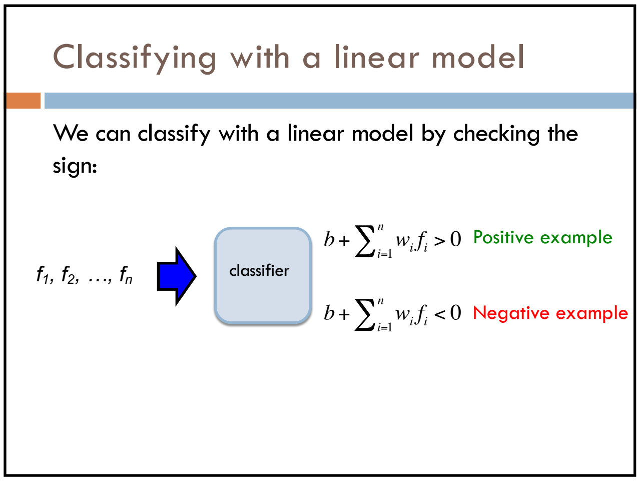
$$0 = w_1 f_1 + w_2 f_2 + b \quad \text{(where b = -a)}$$

In three dimensions, a plane:

$$0 = w_1 f_1 + w_2 f_2 + w_3 f_3 + b$$

In *n*-dimensions, a *hyperplane*

$$0 = b + \sum_{i=1}^{n} w_i f_i$$



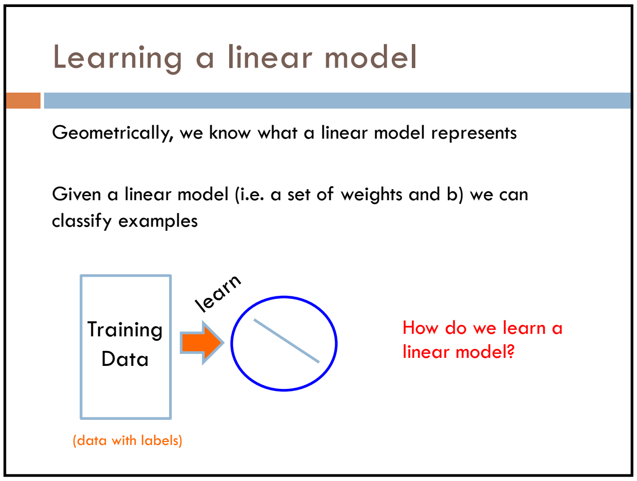
## Classifying with a linear model

We can classify with a linear model by checking the sign:

$f_1, f_2, \ldots, f_n$ → classifier

$b+\sum_{i=1}^{n} w_i f_i > 0$ Positive example

$b+\sum_{i=1}^{n} w_i f_i < 0$ Negative example

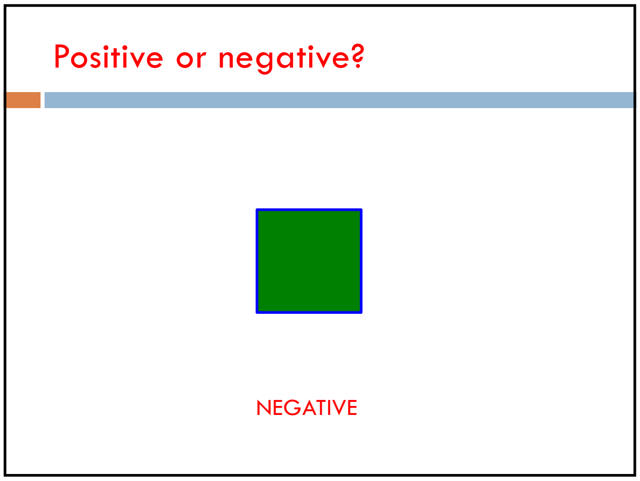


## Learning a linear model

Geometrically, we know what a linear model represents

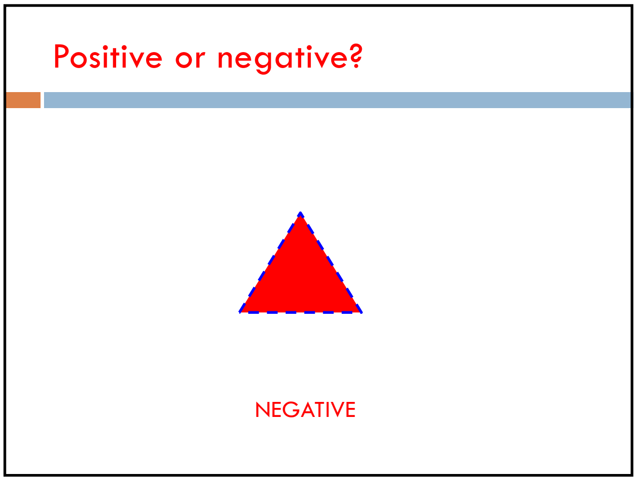
Given a linear model (i.e. a set of weights and b) we can classify examples

How do we learn a linear model?



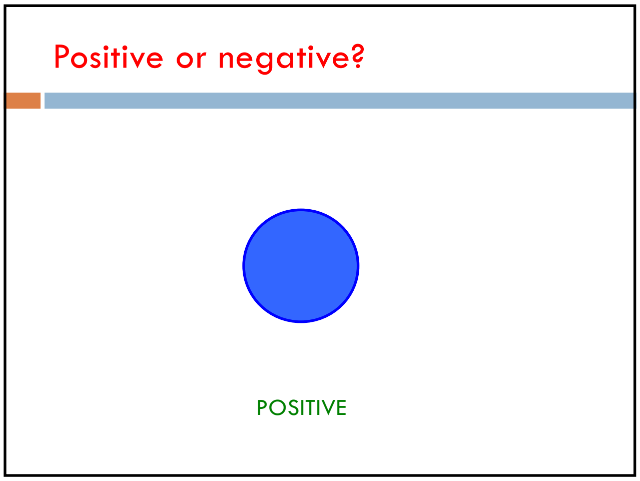
## Positive or negative?

NEGATIVE



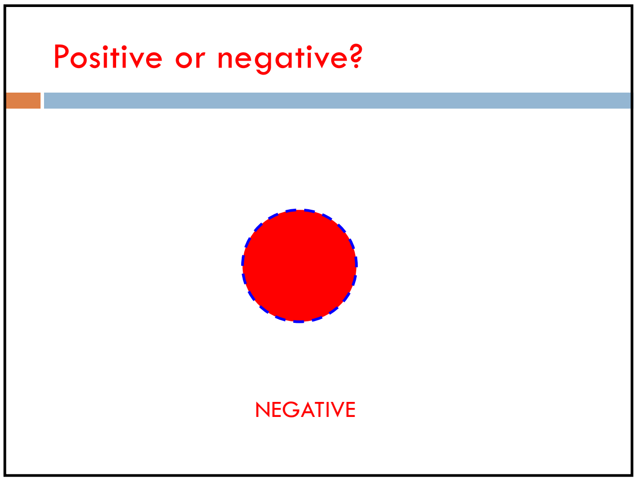
## Positive or negative?

NEGATIVE



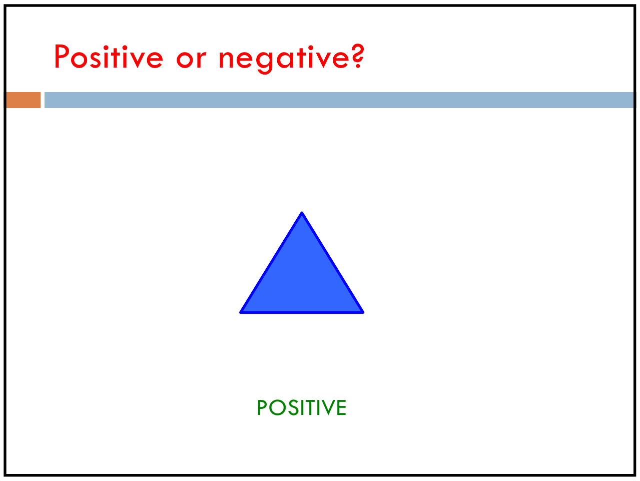
## Positive or negative?

POSITIVE



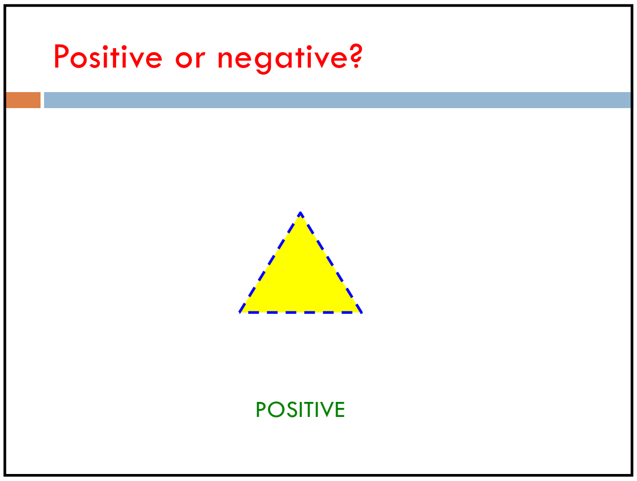
## Positive or negative?

NEGATIVE



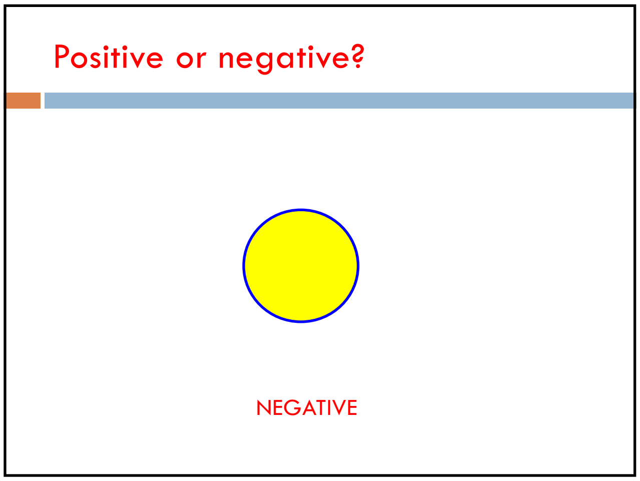
## Positive or negative?

POSITIVE



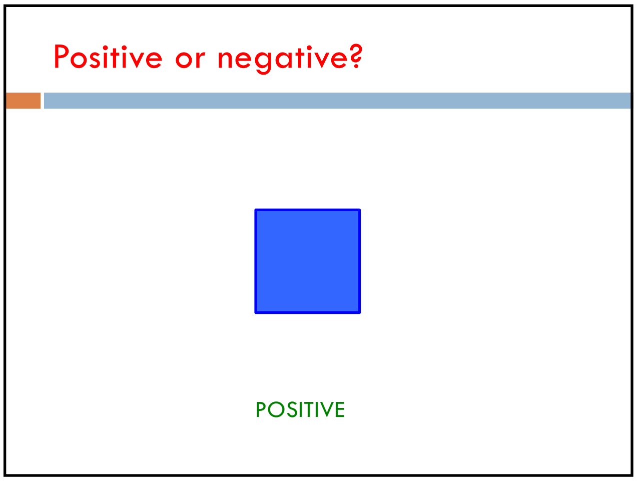
## Positive or negative?

POSITIVE



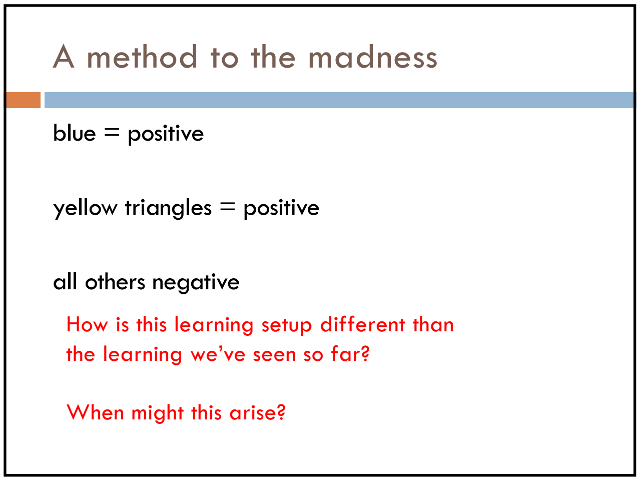
## Positive or negative?

NEGATIVE



## Positive or negative?

POSITIVE



## A method to the madness

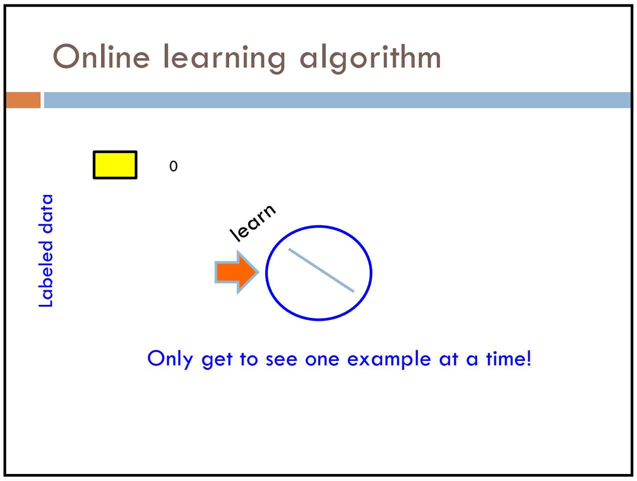
blue = positive

yellow triangles = positive

all others negative

How is this learning setup different than the learning we've seen so far?

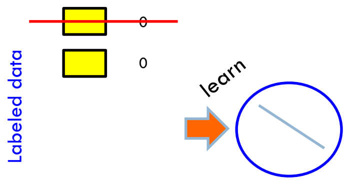
When might this arise?



## Online learning algorithm

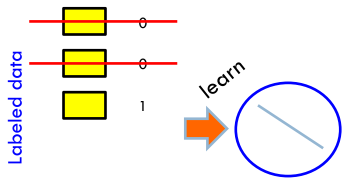
Only get to see one example at a time!

## Online learning algorithm



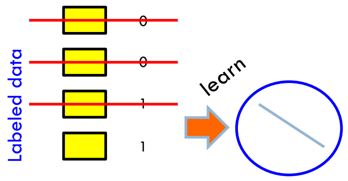
Only get to see one example at a time!

## Online learning algorithm



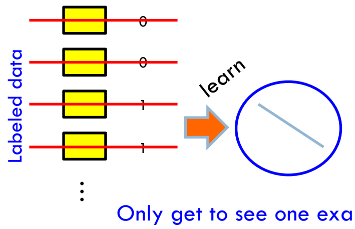
Only get to see one example at a time!

## Online learning algorithm

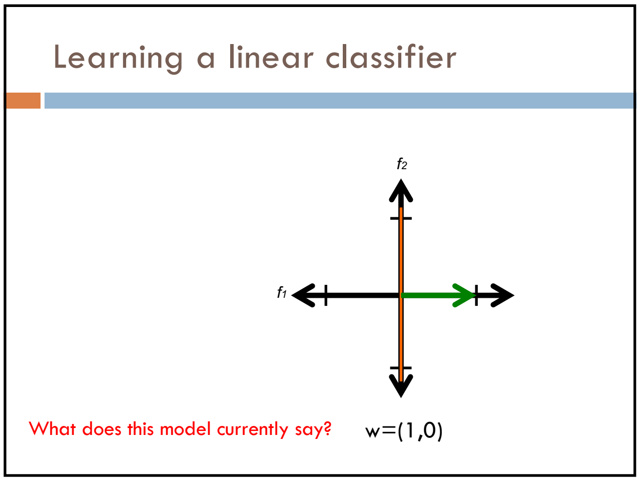


Only get to see one example at a time!

## Online learning algorithm



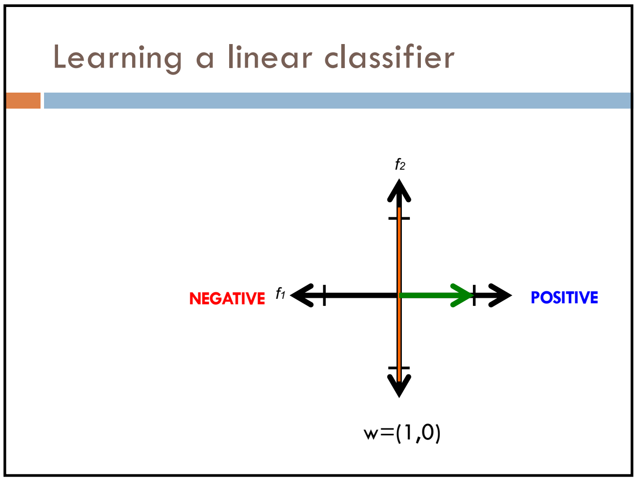
Only get to see one example at a time!



### Learning a linear classifier

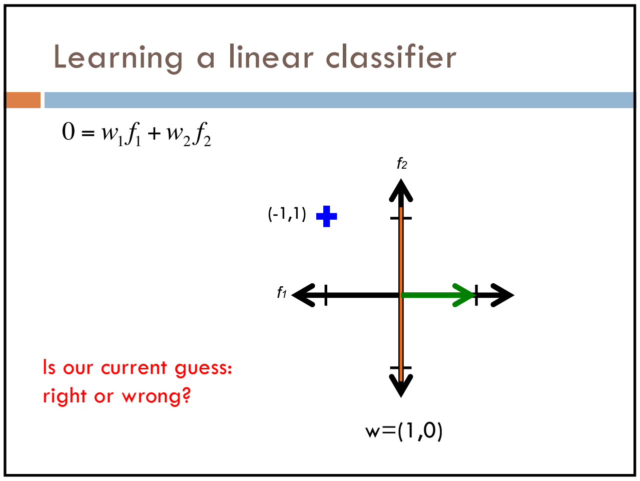
What does this model currently say?

w=(1,0)



### Learning a linear classifier

w=(1,0)

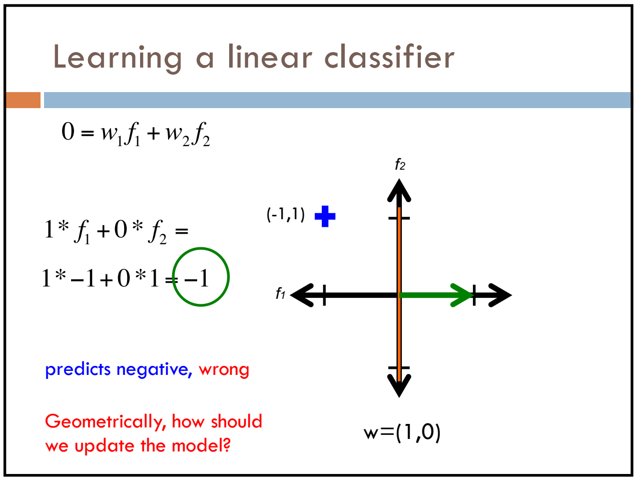


### Learning a linear classifier

$0 = w_1 f_1 + w_2 f_2$

Is our current guess: right or wrong?

w=(1,0)



### Learning a linear classifier

$0 = w_1 f_1 + w_2 f_2$

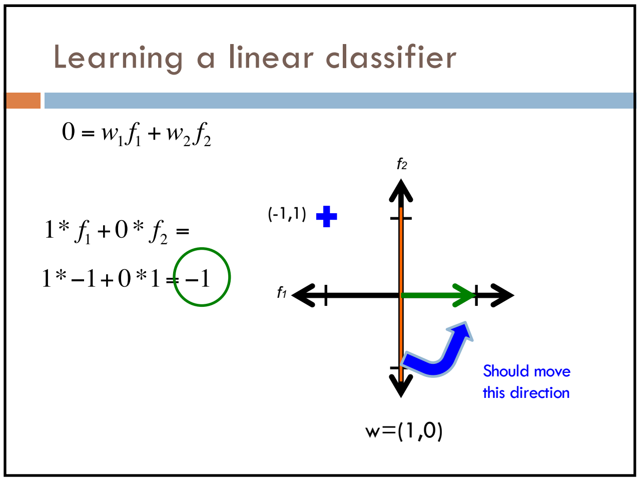
$1 * f_1 + 0 * f_2 =$

$1 * -1 + 0 * 1 = \boxed{-1}$

predicts negative, wrong

Geometrically, how should we update the model?

w=(1,0)



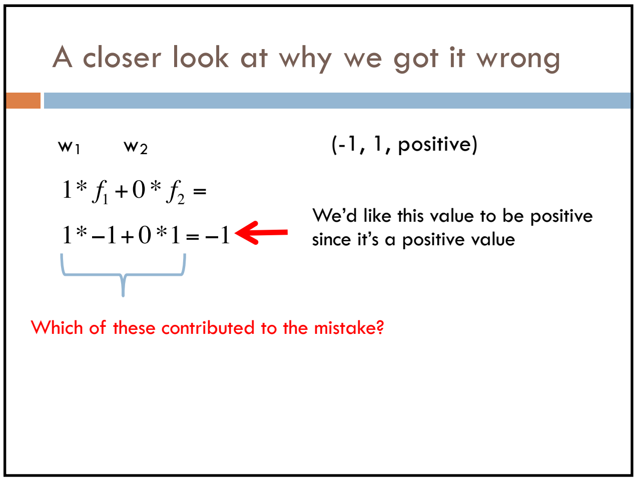
## Learning a linear classifier

$0 = w_1 f_1 + w_2 f_2$

$1 * f_1 + 0 * f_2 =$

$1 * -1 + 0 * 1 = -1$

w=(1,0)



## A closer look at why we got it wrong

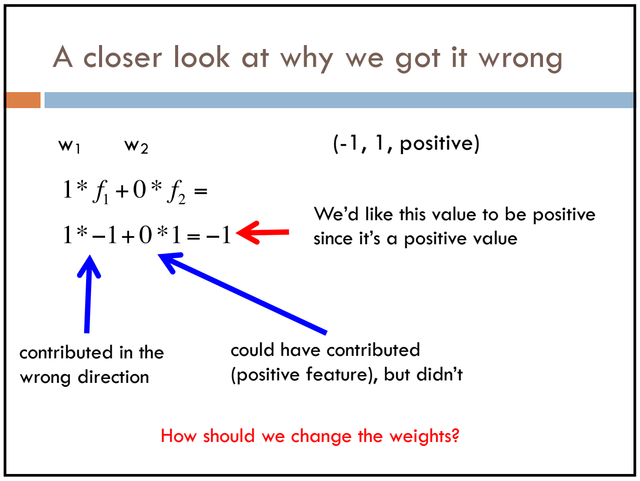
w1 w2 (-1, 1, positive)

$1 * f_1 + 0 * f_2 =$

$\underbrace{1 * -1 + 0 * 1}_{} = -1$

We'd like this value to be positive since it's a positive value

Which of these contributed to the mistake?



## A closer look at why we got it wrong

w1 w2 (-1, 1, positive)

$1 * f_1 + 0 * f_2 =$

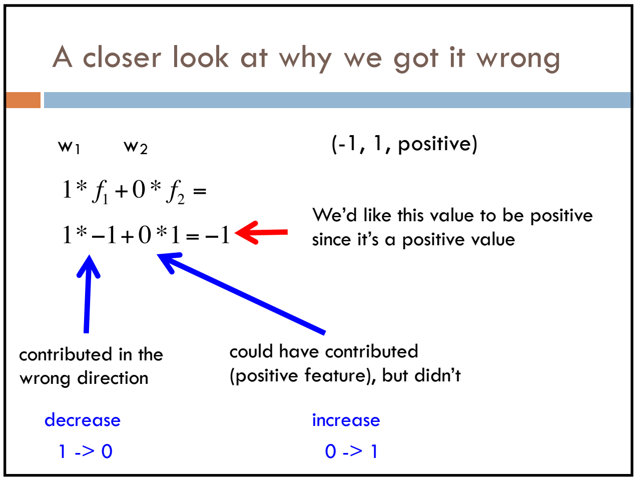
$1 * -1 + 0 * 1 = -1$

We'd like this value to be positive since it's a positive value

contributed in the wrong direction

could have contributed (positive feature), but didn't

How should we change the weights?



## A closer look at why we got it wrong

w1 w2 (-1, 1, positive)

$1 * f_1 + 0 * f_2 =$

$1 * -1 + 0 * 1 = -1$

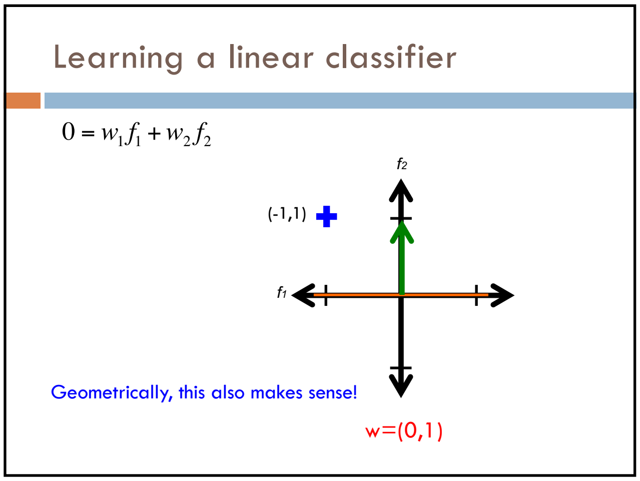
We'd like this value to be positive since it's a positive value

contributed in the wrong direction

could have contributed (positive feature), but didn't

decrease
1 -> 0

increase
0 -> 1

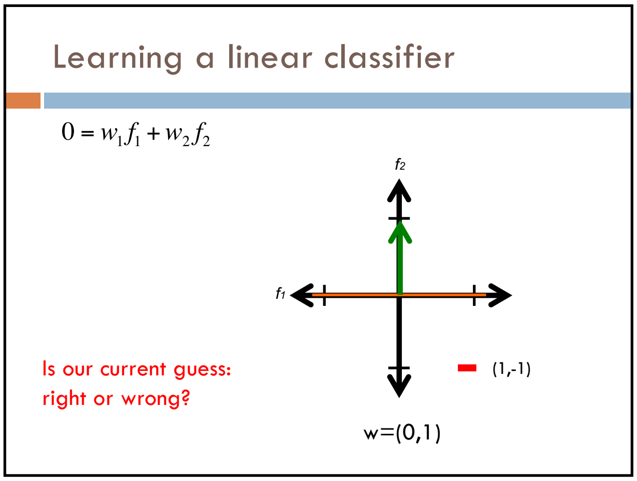


## Learning a linear classifier

$0 = w_1 f_1 + w_2 f_2$

Geometrically, this also makes sense!

w=(0,1)

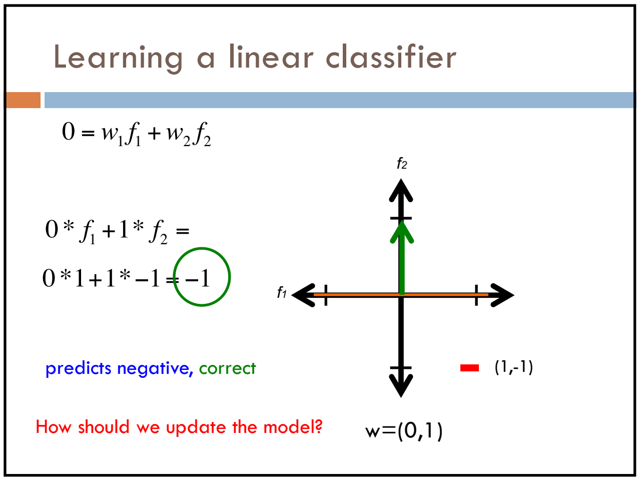


## Learning a linear classifier

$0 = w_1 f_1 + w_2 f_2$

Is our current guess: right or wrong?

w=(0,1)



## Learning a linear classifier

$0 = w_1 f_1 + w_2 f_2$

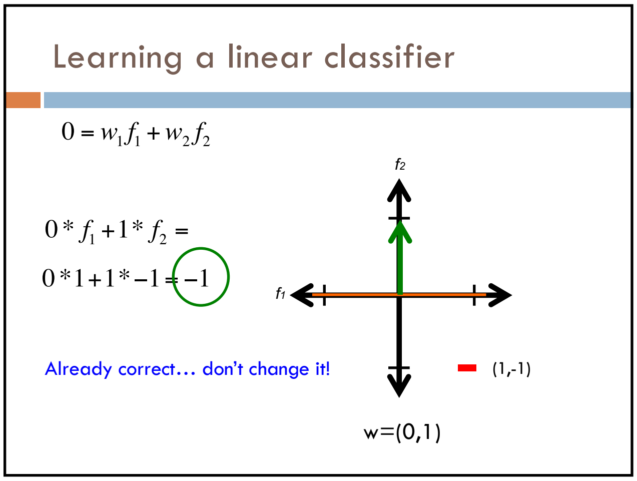
$0 * f_1 + 1 * f_2 =$

$0 * 1 + 1 * -1 = \boxed{-1}$

predicts negative, correct

How should we update the model?

w=(0,1)



## Learning a linear classifier

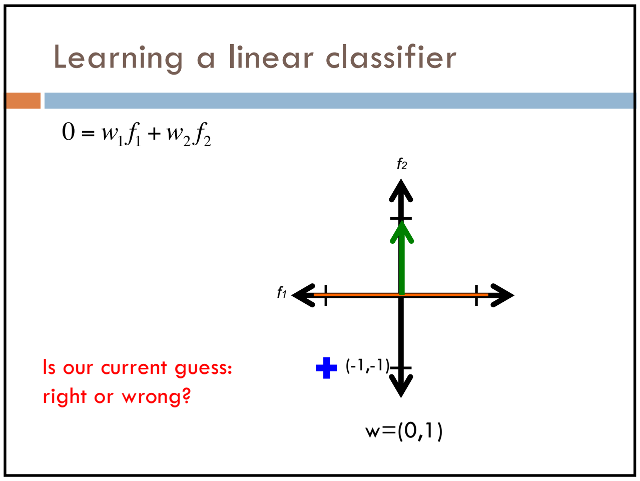
$0 = w_1 f_1 + w_2 f_2$

$0 * f_1 + 1 * f_2 =$

$0 * 1 + 1 * -1 = \boxed{-1}$

Already correct… don't change it!

w=(0,1)

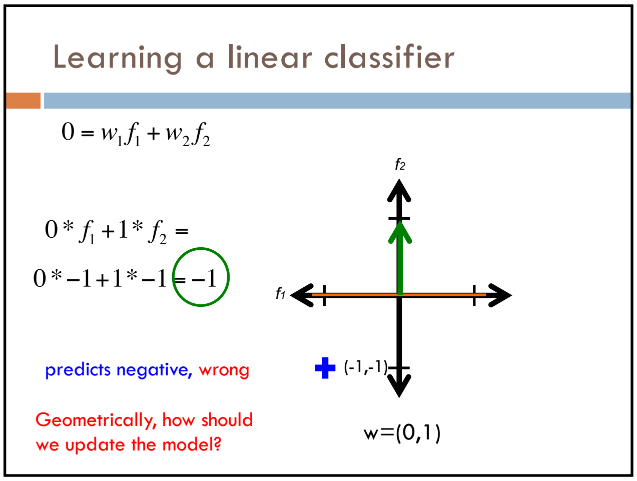


## Learning a linear classifier

$0 = w_1 f_1 + w_2 f_2$

w=(0,1)

Is our current guess: right or wrong?



## Learning a linear classifier

$0 = w_1 f_1 + w_2 f_2$

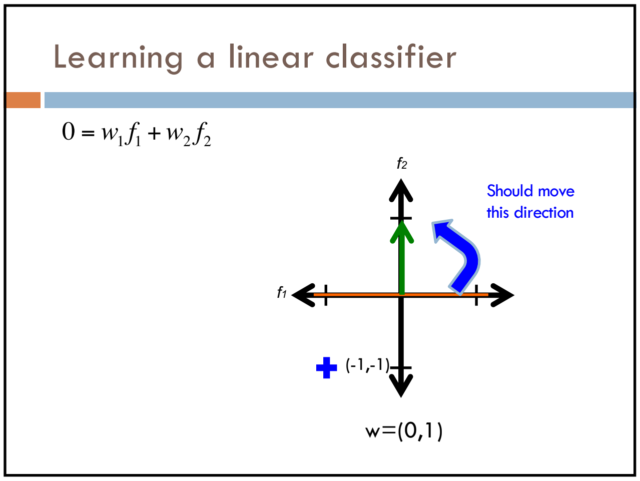
$0 * f_1 + 1 * f_2 =$

$0 * -1 + 1 * -1 = \boxed{-1}$

predicts negative, wrong

Geometrically, how should we update the model?

w=(0,1)

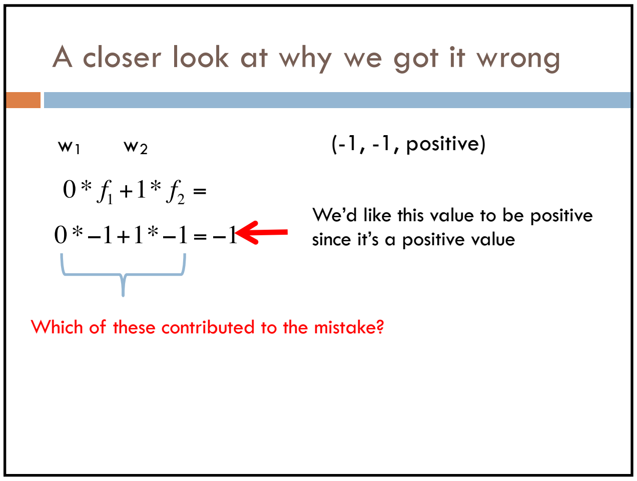


## Learning a linear classifier

$0 = w_1 f_1 + w_2 f_2$

Should move this direction

w=(0,1)



## A closer look at why we got it wrong

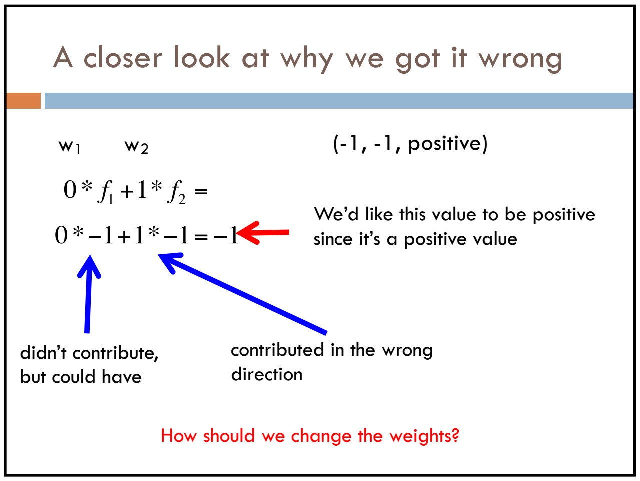
(-1, -1, positive)

$\overset{w_1}{0} * f_1 + \overset{w_2}{1} * f_2 =$

$\underbrace{0 * -1 + 1 * -1}_{} = -1$

We'd like this value to be positive since it's a positive value

Which of these contributed to the mistake?



## A closer look at why we got it wrong

$w_1$ $w_2$ (-1, -1, positive)

$0 * f_1 + 1 * f_2 =$

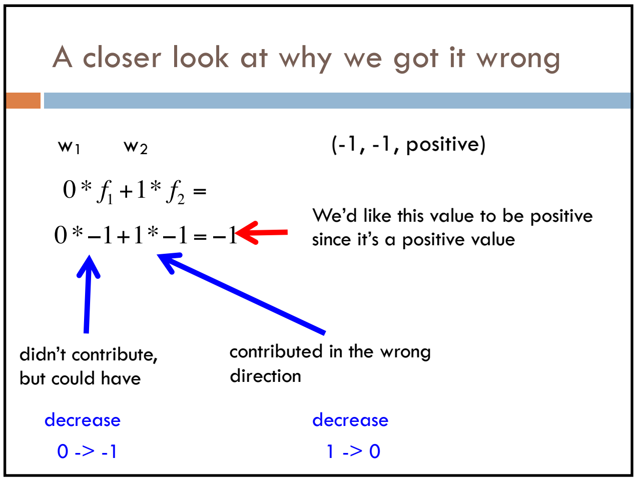
$0 * -1 + 1 * -1 = -1$

We'd like this value to be positive since it's a positive value

didn't contribute, but could have

contributed in the wrong direction

How should we change the weights?



## A closer look at why we got it wrong

$w_1$ $w_2$ (-1, -1, positive)

$0 * f_1 + 1 * f_2 =$

$0 * -1 + 1 * -1 = -1$

We'd like this value to be positive since it's a positive value

didn't contribute, but could have

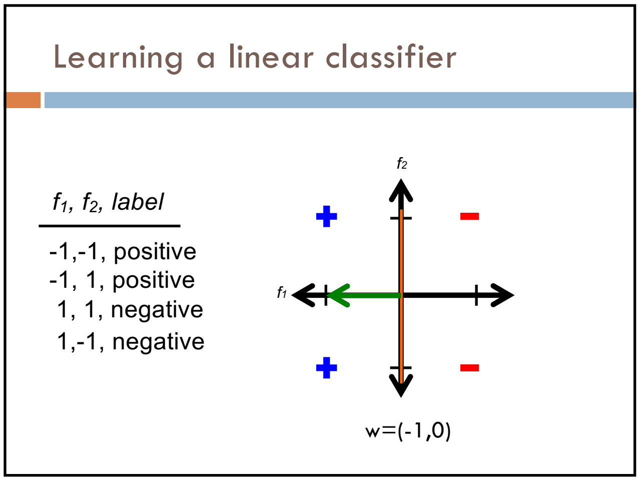
contributed in the wrong direction

decrease

0 -> -1

decrease

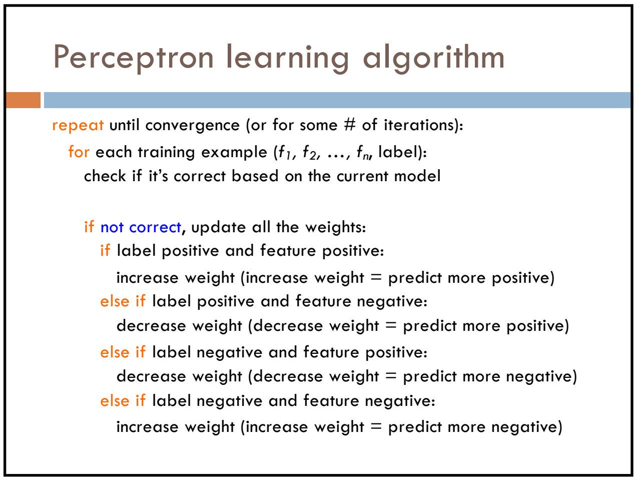
1 -> 0



## Learning a linear classifier

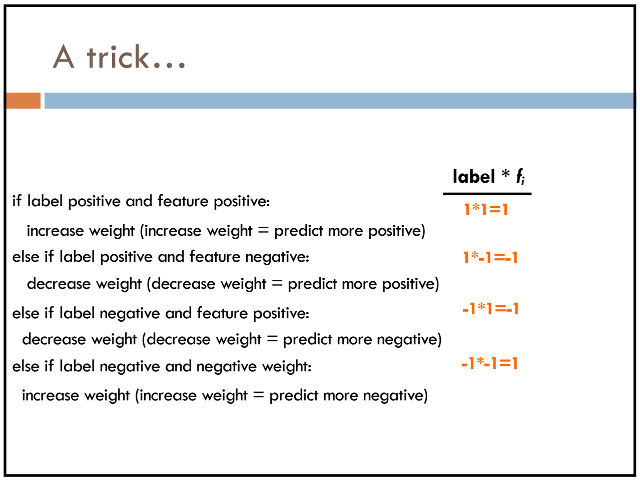
| $f_1$, $f_2$, *label* |
|---|
| -1,-1, positive |
| -1, 1, positive |
| 1, 1, negative |
| 1,-1, negative |

w=(-1,0)



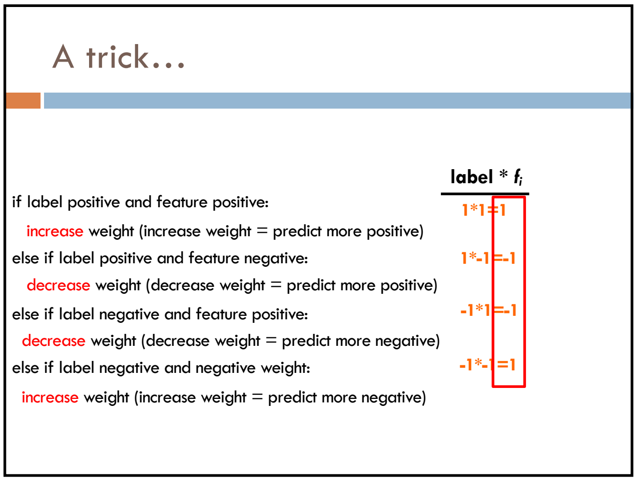
## Perceptron learning algorithm

```
repeat until convergence (or for some # of iterations):
  for each training example ($f_1$, $f_2$, …, $f_n$, label):
    check if it's correct based on the current model

    if not correct, update all the weights:
      if label positive and feature positive:
        increase weight (increase weight = predict more positive)
      else if label positive and feature negative:
        decrease weight (decrease weight = predict more positive)
      else if label negative and feature positive:
        decrease weight (decrease weight = predict more negative)
      else if label negative and feature negative:
        increase weight (increase weight = predict more negative)
```



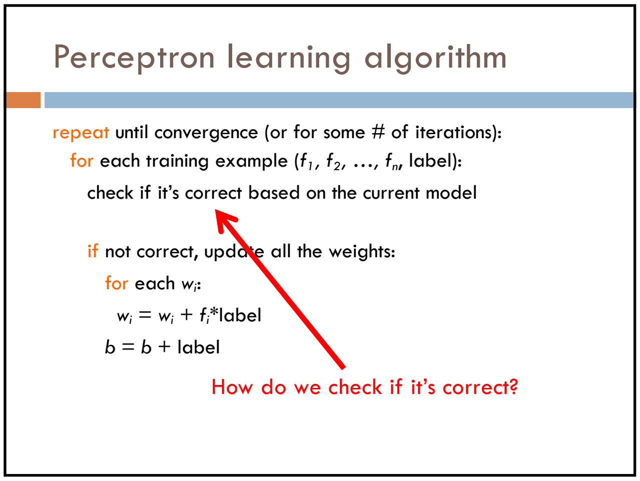
## A trick…

| | label * $f_i$ |
|---|---|
| if label positive and feature positive:<br>increase weight (increase weight = predict more positive) | 1*1=1 |
| else if label positive and feature negative:<br>decrease weight (decrease weight = predict more positive) | 1*-1=-1 |
| else if label negative and feature positive:<br>decrease weight (decrease weight = predict more negative) | -1*1=-1 |
| else if label negative and negative weight:<br>increase weight (increase weight = predict more negative) | -1*-1=1 |



## A trick…

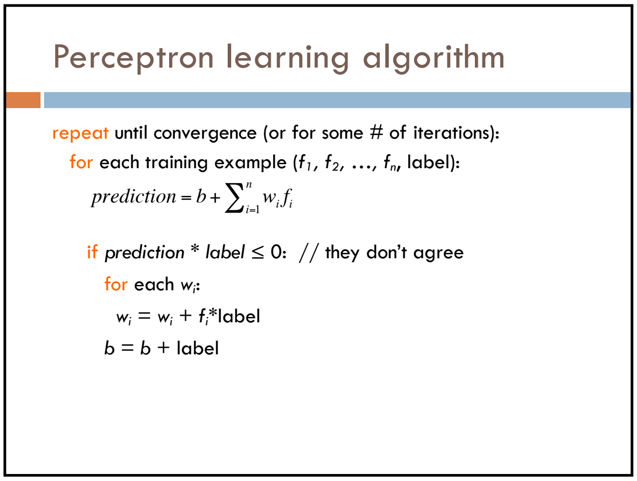
| | label * $f_i$ |
|---|---|
| if label positive and feature positive:<br>increase weight (increase weight = predict more positive) | 1*1=1 |
| else if label positive and feature negative:<br>decrease weight (decrease weight = predict more positive) | 1*-1=-1 |
| else if label negative and feature positive:<br>decrease weight (decrease weight = predict more negative) | -1*1=-1 |
| else if label negative and negative weight:<br>increase weight (increase weight = predict more negative) | -1*-1=1 |



## Perceptron learning algorithm

```
repeat until convergence (or for some # of iterations):
    for each training example ($f_1$, $f_2$, …, $f_n$, label):
        check if it's correct based on the current model

        if not correct, update all the weights:
            for each $w_i$:
                $w_i = w_i + f_i$*label
            $b = b$ + label
```

How do we check if it's correct?



## Perceptron learning algorithm

```
repeat until convergence (or for some # of iterations):
    for each training example ($f_1$, $f_2$, …, $f_n$, label):
        $prediction = b + \sum_{i=1}^{n} w_i f_i$

        if $prediction$ * $label \le 0$:  // they don't agree
            for each $w_i$:
                $w_i = w_i + f_i$*label
            $b = b$ + label
```

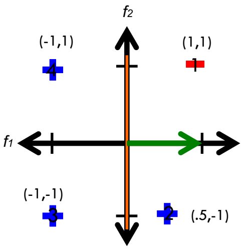
## Perceptron learning algorithm

```
repeat until convergence (or for some # of iterations):
  for each training example ($f_1$, $f_2$, …, $f_n$, label):
    $prediction = b + \sum_{i=1}^{n} w_i f_i$

    if $prediction$ * $label \leq 0$:  // they don't agree
      for each $w_i$:
        $w_i = w_i + f_i$*label
      $b = b$ + label
```

Would this work for non-binary features, i.e. real-valued?

## Your turn ☺

```
repeat until convergence (or for some # of iterations):
  for each training example ($f_1$, $f_2$, …, $f_n$, label):
    $prediction = \sum_{i=1}^{n} w_i f_i$

    if $prediction$ * $label \leq 0$:  // they don't agree
      for each $w_i$:
        $w_i = w_i + f_i$*label
```

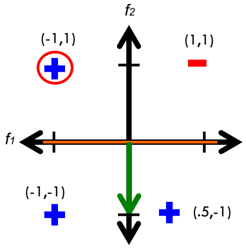


- Repeat until convergence
- Keep track of $w_1$, $w_2$ as they change
- Redraw the line after each step

w = (1, 0)

## Your turn ☺

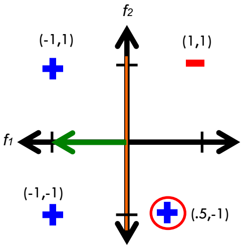
```
repeat until convergence (or for some # of iterations):
  for each training example ($f_1$, $f_2$, …, $f_n$, label):
    $prediction = \sum_{i=1}^{n} w_i f_i$

    if $prediction$ * $label \leq 0$:  // they don't agree
      for each $w_i$:
        $w_i = w_i + f_i$*label
```



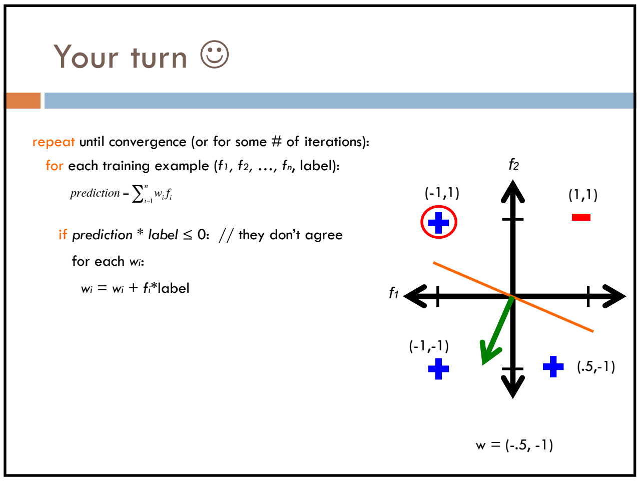
w = (0, -1)

## Your turn ☺

```
repeat until convergence (or for some # of iterations):
  for each training example ($f_1$, $f_2$, …, $f_n$, label):
    $prediction = \sum_{i=1}^{n} w_i f_i$

    if $prediction$ * $label \leq 0$:  // they don't agree
      for each $w_i$:
        $w_i = w_i + f_i$*label
```



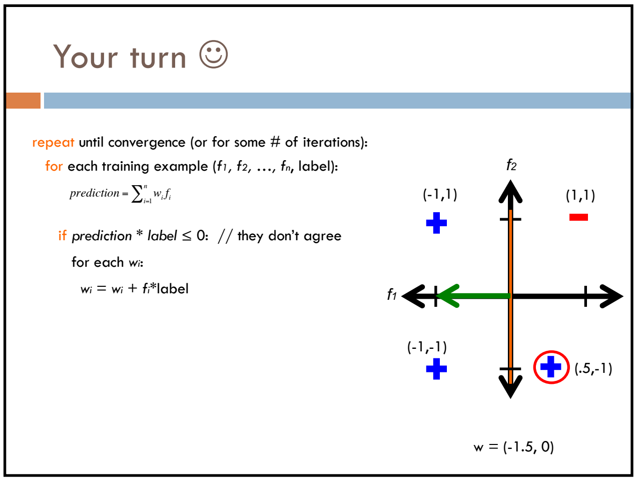
w = (-1, 0)



# Your turn ☺

```
repeat until convergence (or for some # of iterations):
  for each training example ($f_1$, $f_2$, …, $f_n$, label):
    $prediction = \sum_{i=1}^{n} w_i f_i$
    if prediction * label ≤ 0:  // they don't agree
      for each $w_i$:
        $w_i = w_i + f_i$*label
```

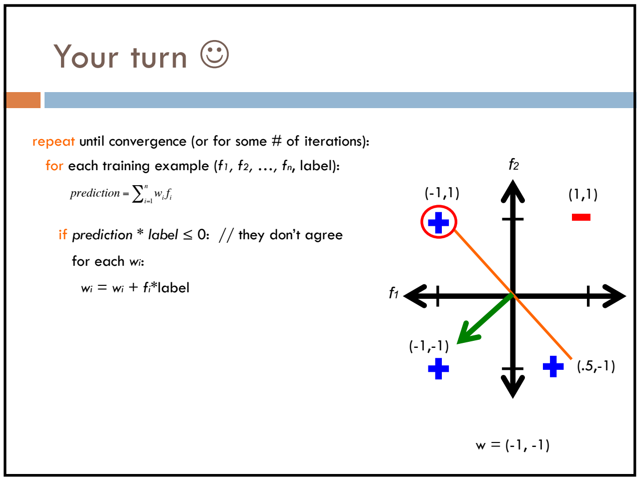
w = (-.5, -1)



# Your turn ☺

```
repeat until convergence (or for some # of iterations):
  for each training example ($f_1$, $f_2$, …, $f_n$, label):
    $prediction = \sum_{i=1}^{n} w_i f_i$
    if prediction * label ≤ 0:  // they don't agree
      for each $w_i$:
        $w_i = w_i + f_i$*label
```

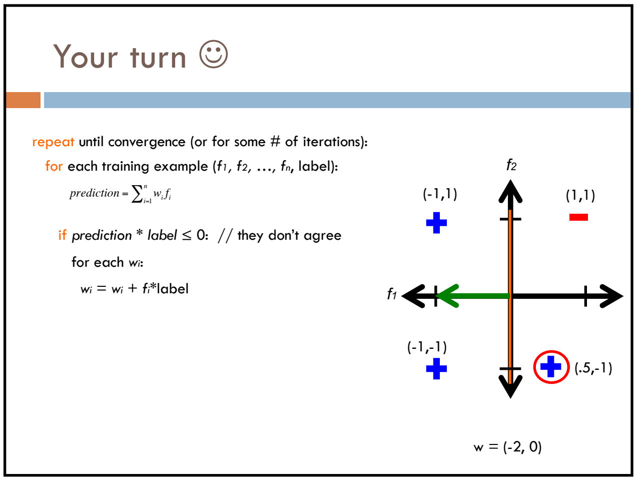
w = (-1.5, 0)



# Your turn ☺

```
repeat until convergence (or for some # of iterations):
  for each training example ($f_1$, $f_2$, …, $f_n$, label):
    $prediction = \sum_{i=1}^{n} w_i f_i$
    if prediction * label ≤ 0:  // they don't agree
      for each $w_i$:
        $w_i = w_i + f_i$*label
```

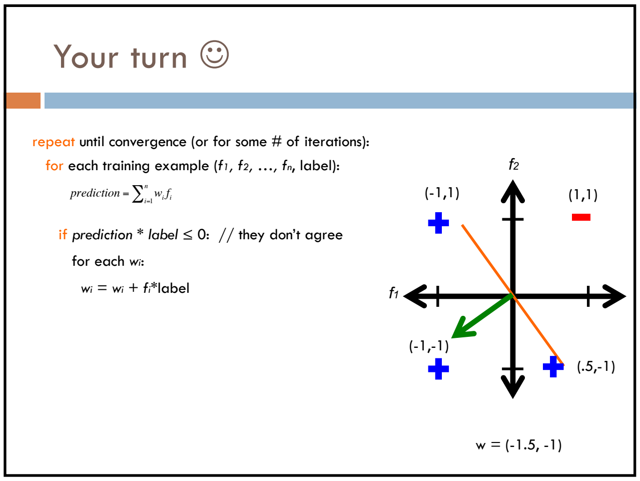
w = (-1, -1)



# Your turn ☺

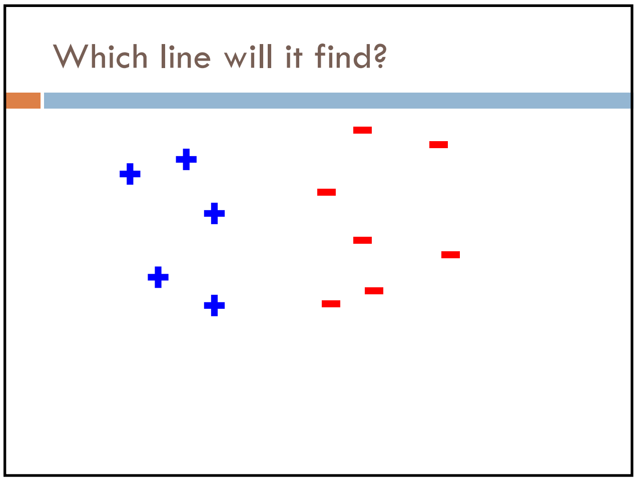
```
repeat until convergence (or for some # of iterations):
  for each training example ($f_1$, $f_2$, …, $f_n$, label):
    $prediction = \sum_{i=1}^{n} w_i f_i$
    if prediction * label ≤ 0:  // they don't agree
      for each $w_i$:
        $w_i = w_i + f_i$*label
```

w = (-2, 0)

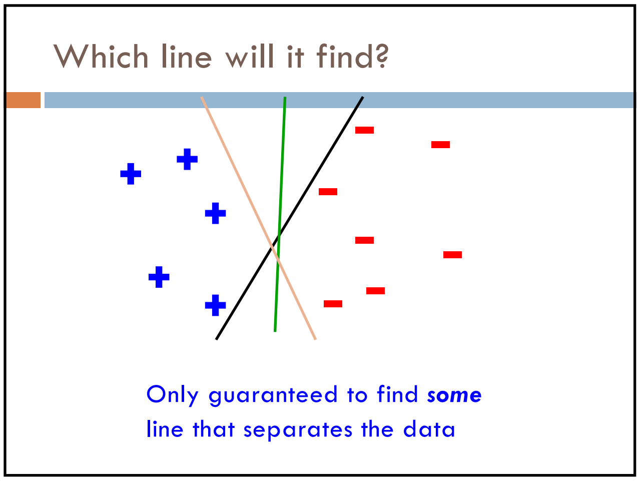


## Your turn ☺

```
repeat until convergence (or for some # of iterations):
  for each training example ($f_1$, $f_2$, …, $f_n$, label):
    $prediction = \sum_{i=1}^{n} w_i f_i$

    if $prediction$ * $label \le 0$:  // they don't agree
      for each $w_i$:
        $w_i = w_i + f_i$*label
```

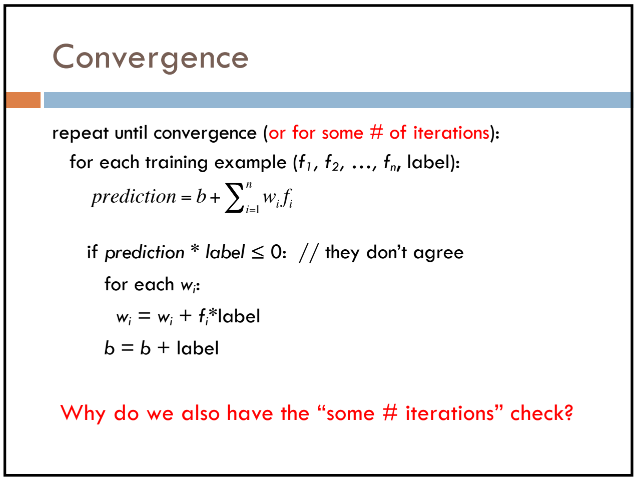


## Which line will it find?



## Which line will it find?

Only guaranteed to find ***some*** line that separates the data

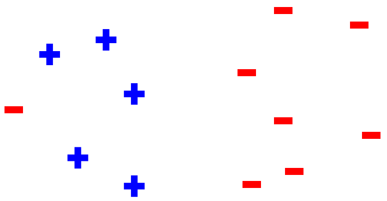


## Convergence

```
repeat until convergence (or for some # of iterations):
  for each training example ($f_1$, $f_2$, …, $f_n$, label):
    $prediction = b + \sum_{i=1}^{n} w_i f_i$

    if $prediction$ * $label \le 0$:  // they don't agree
      for each $w_i$:
        $w_i = w_i + f_i$*label
      $b = b +$ label
```

Why do we also have the "some # iterations" check?

## Handling non-separable data



If we ran the algorithm on this it would never converge!

## Convergence

```
repeat until convergence (or for some # of iterations):
  for each training example ($f_1$, $f_2$, …, $f_n$, label):
    $prediction = b + \sum_{i=1}^{n} w_i f_i$

    if $prediction$ * $label \leq 0$:  // they don't agree
      for each $w_i$:
        $w_i = w_i + f_i$*label
      $b = b$ + label
```

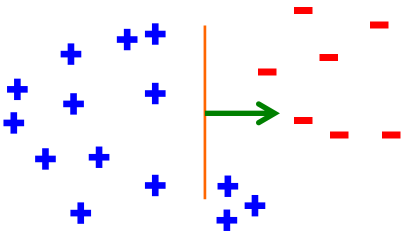
Also helps avoid overfitting!
(This is harder to see in 2-D examples, though)

## Ordering

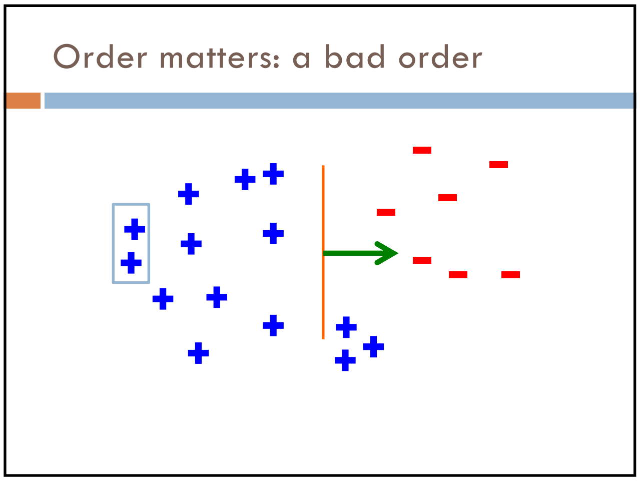
```
repeat until convergence (or for some # of iterations):
  for each training example ($f_1$, $f_2$, …, $f_n$, label):
    $prediction = b + \sum_{i=1}^{n} w_i f_i$

    if $prediction$ * $label \leq 0$:  // they don't agree
      for each $w_i$:
        $w_i = w_i + f_i$*label
      $b = b$ + label
```

What order should we traverse the examples?
Does it matter?

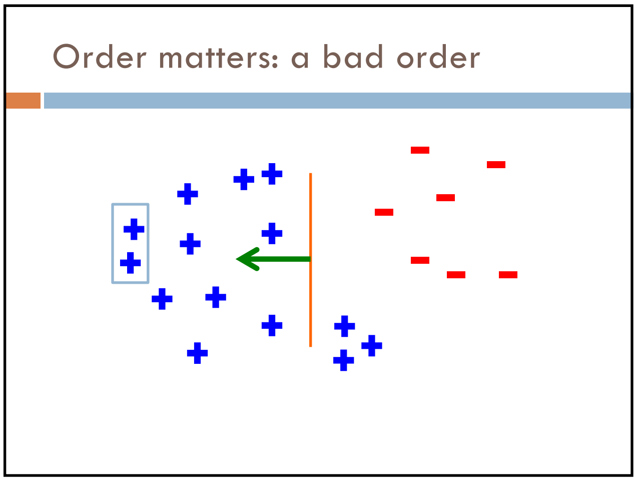
## Order matters



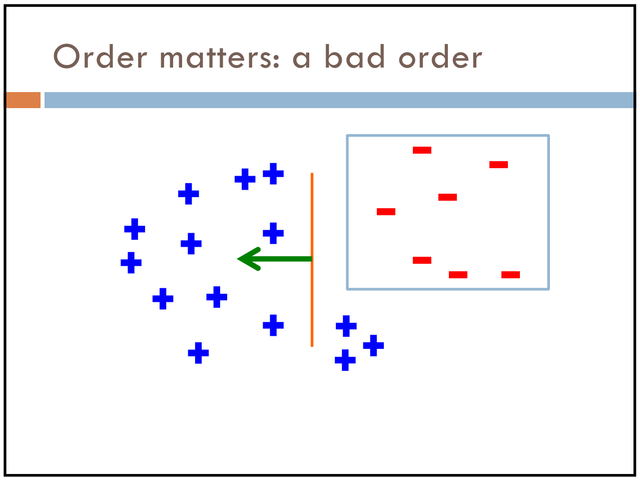
What would be a good/bad order?



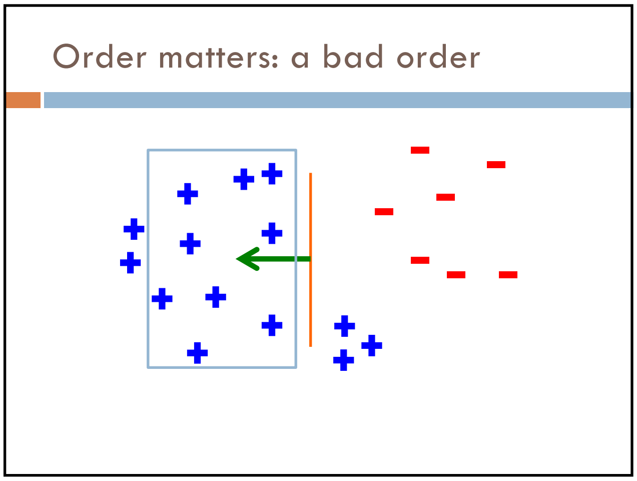
## Order matters: a bad order



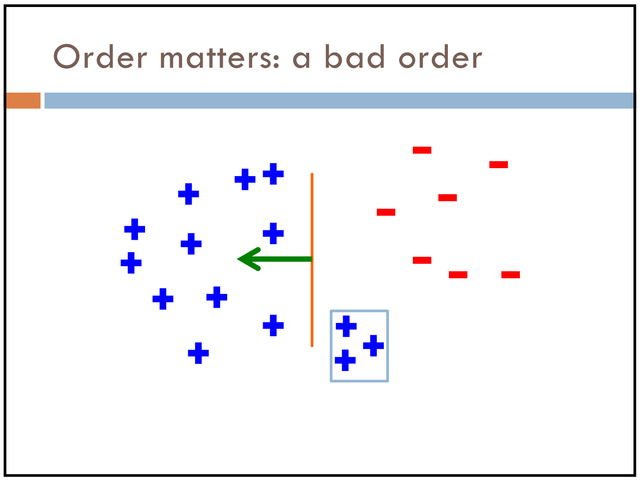
## Order matters: a bad order



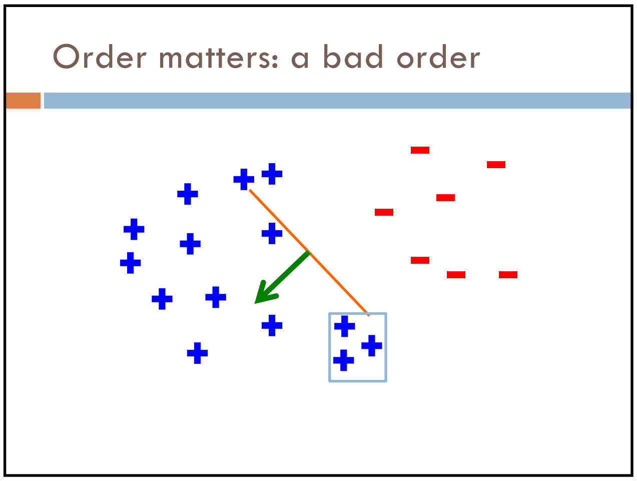
## Order matters: a bad order



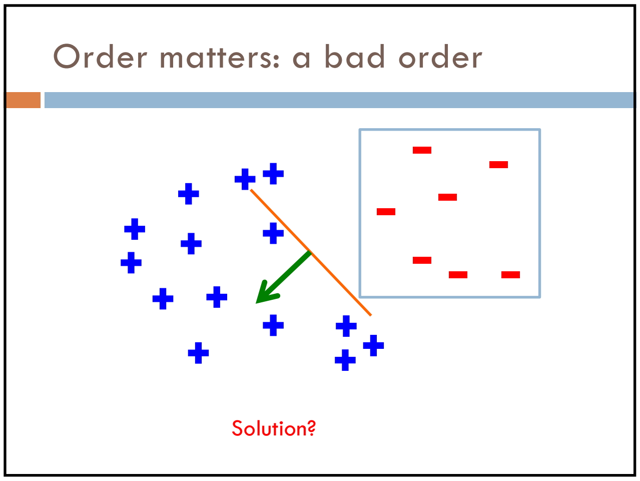
## Order matters: a bad order



## Order matters: a bad order

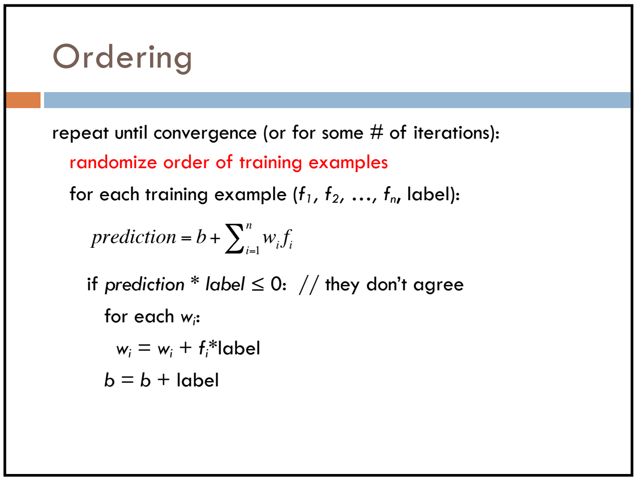


## Order matters: a bad order



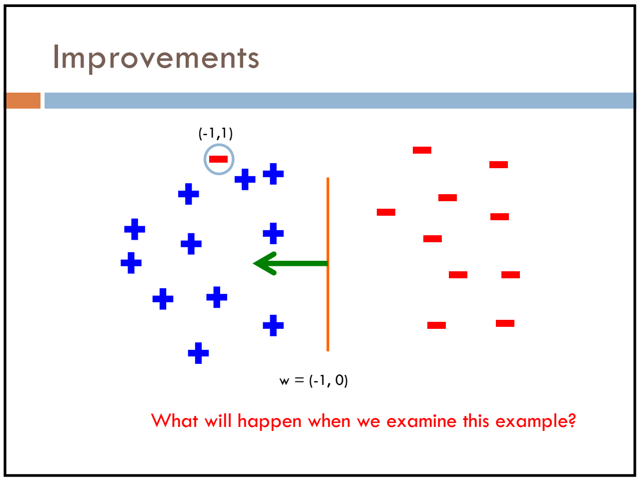
## Order matters: a bad order

Solution?



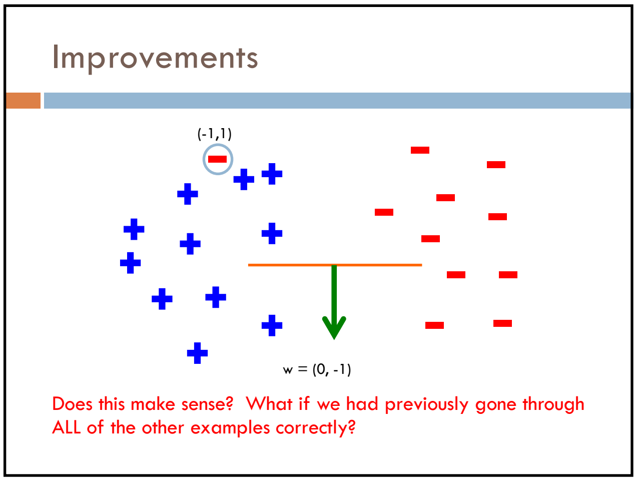
## Ordering

```
repeat until convergence (or for some # of iterations):
   randomize order of training examples
   for each training example ($f_1$, $f_2$, …, $f_n$, label):
      $prediction = b + \sum_{i=1}^{n} w_i f_i$
      if $prediction * label \leq 0$:  // they don't agree
         for each $w_i$:
            $w_i = w_i + f_i$*label
         $b = b$ + label
```



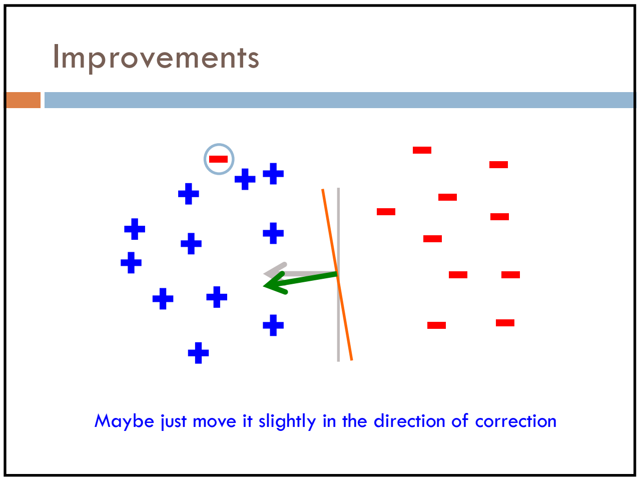
## Improvements

What will happen when we examine this example?



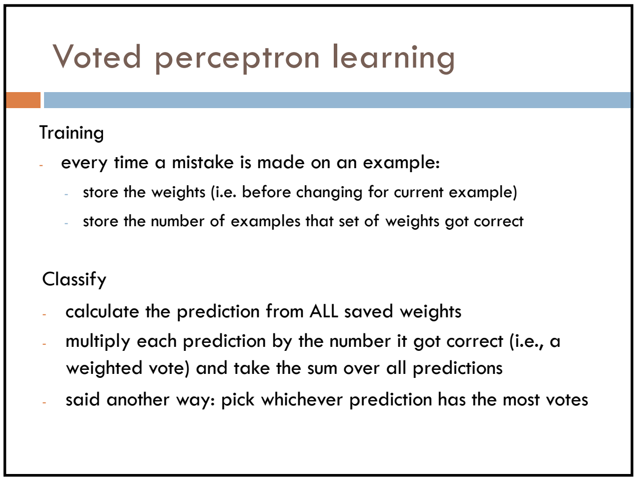
## Improvements

Does this make sense? What if we had previously gone through ALL of the other examples correctly?



## Improvements

Maybe just move it slightly in the direction of correction



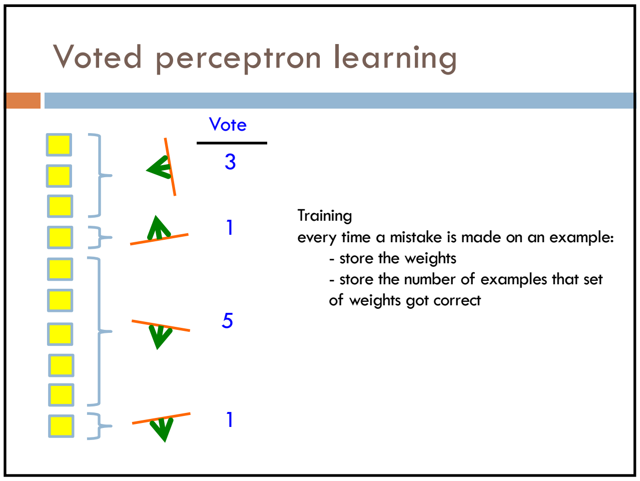
## Voted perceptron learning

Training

- every time a mistake is made on an example:
  - store the weights (i.e. before changing for current example)
  - store the number of examples that set of weights got correct

Classify

- calculate the prediction from ALL saved weights
- multiply each prediction by the number it got correct (i.e., a weighted vote) and take the sum over all predictions
- said another way: pick whichever prediction has the most votes



### Voted perceptron learning

Vote

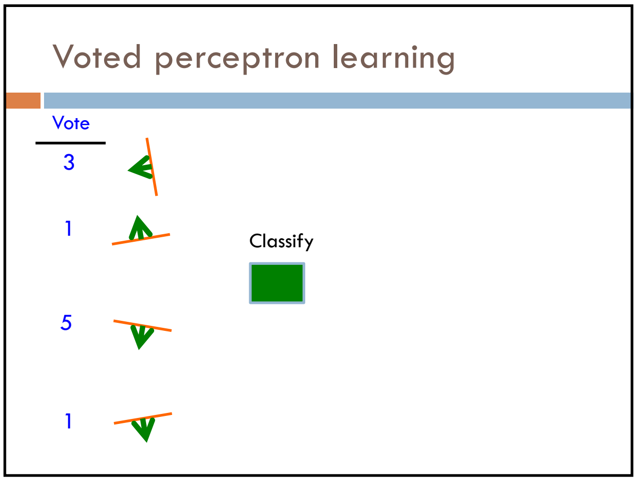
3

1

5

1

Training
every time a mistake is made on an example:
- store the weights
- store the number of examples that set of weights got correct



### Voted perceptron learning

Vote

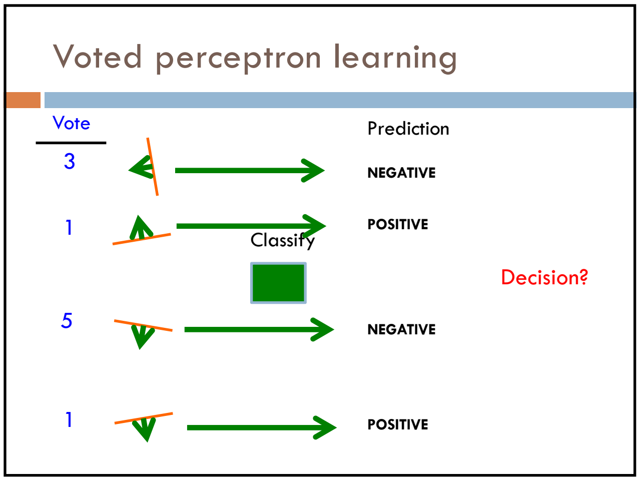
3

1

5

1

Classify

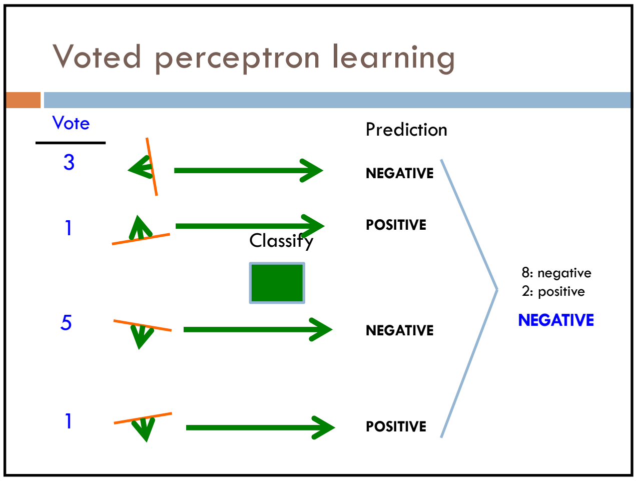


### Voted perceptron learning

| Vote | Prediction |
|---|---|
| 3 | NEGATIVE |
| 1 | POSITIVE |
| 5 | NEGATIVE |
| 1 | POSITIVE |

Classify

Decision?



### Voted perceptron learning

| Vote | Prediction |
|---|---|
| 3 | NEGATIVE |
| 1 | POSITIVE |
| 5 | NEGATIVE |
| 1 | POSITIVE |

Classify

8: negative
2: positive

NEGATIVE

## Voted perceptron learning

Works much better in practice

Avoids overfitting, though it can still happen

Avoids big changes in the result by examples examined at the end of training

## Voted perceptron learning

Training

- every time a mistake is made on an example:
  - store the weights (i.e. before changing for current example)
  - store the number of examples that set of weights got correct

Classify

- calculate the prediction from ALL saved weights
- multiply each prediction by the number it got correct (i.e a weighted vote) and take the sum over all predictions
- said another way: pick whichever prediction has the most votes

Any issues/concerns?

## Voted perceptron learning

Training

- every time a mistake is made on an example:
  - store the weights (i.e. before changing for current example)
  - store the number of examples that set of weights got correct

Classify

- calculate the prediction from ALL saved weights
- multiply each prediction by the number it got correct (i.e a weighted vote) and take the sum over all predictions
- said another way: pick whichever prediction has the most votes

1. Can require a lot of storage
2. Classifying becomes very, very expensive

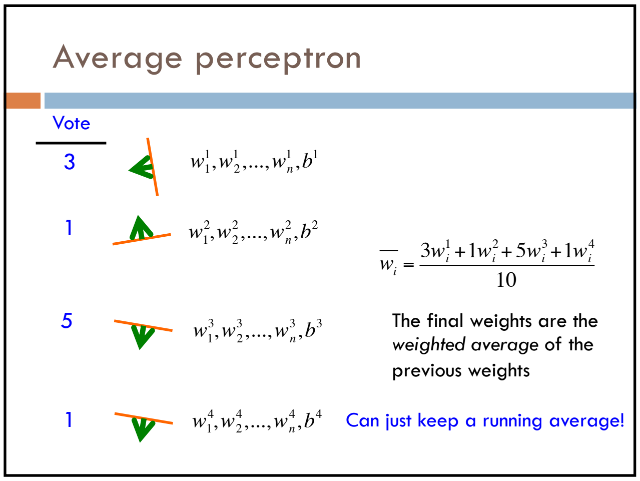
## Average perceptron

| Vote | | |
|---|---|---|
| 3 | | $w_1^1, w_2^1, \ldots, w_n^1, b^1$ |
| 1 | | $w_1^2, w_2^2, \ldots, w_n^2, b^2$ |
| 5 | | $w_1^3, w_2^3, \ldots, w_n^3, b^3$ |
| 1 | | $w_1^4, w_2^4, \ldots, w_n^4, b^4$ |

$$\overline{w_i} = \frac{3w_i^1 + 1w_i^2 + 5w_i^3 + 1w_i^4}{10}$$

The final weights are the *weighted average* of the previous weights

How does this help us?



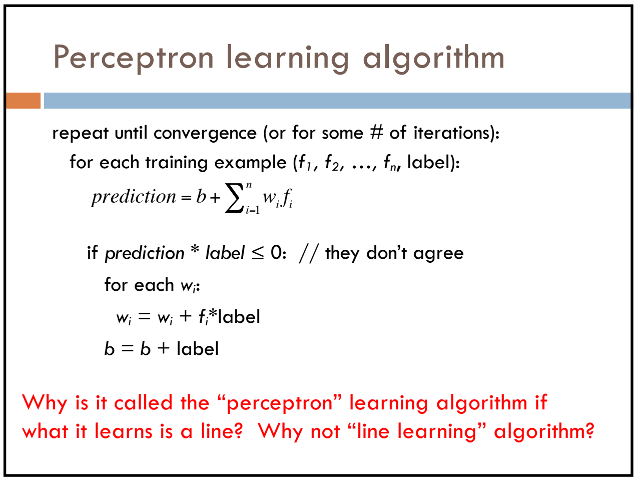
## Average perceptron

| Vote | | |
|---|---|---|
| 3 | | $w_1^1, w_2^1, \ldots, w_n^1, b^1$ |
| 1 | | $w_1^2, w_2^2, \ldots, w_n^2, b^2$ |
| 5 | | $w_1^3, w_2^3, \ldots, w_n^3, b^3$ |
| 1 | | $w_1^4, w_2^4, \ldots, w_n^4, b^4$ |

$$\overline{w_i} = \frac{3w_i^1 + 1w_i^2 + 5w_i^3 + 1w_i^4}{10}$$

The final weights are the *weighted average* of the previous weights

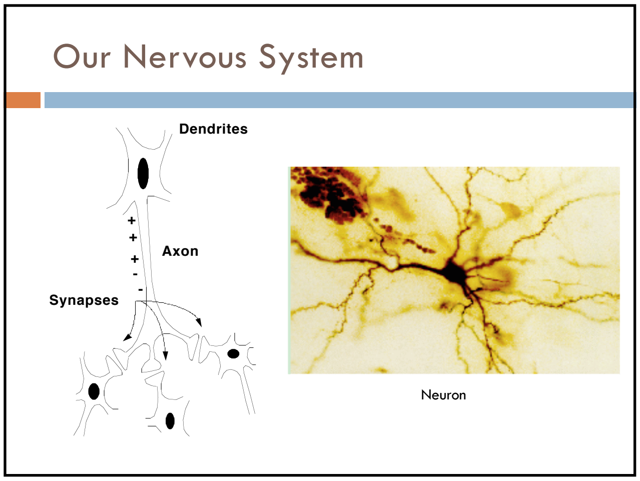
Can just keep a running average!



## Perceptron learning algorithm

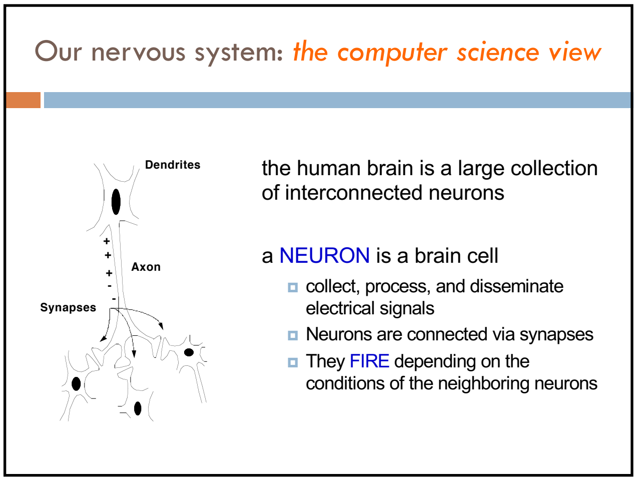
```
repeat until convergence (or for some # of iterations):
  for each training example ($f_1$, $f_2$, …, $f_n$, label):
    $prediction = b + \sum_{i=1}^{n} w_i f_i$
    if prediction * label ≤ 0:  // they don't agree
      for each $w_i$:
        $w_i = w_i + f_i$*label
      $b = b$ + label
```

Why is it called the "perceptron" learning algorithm if what it learns is a line? Why not "line learning" algorithm?



## Our Nervous System

Neuron

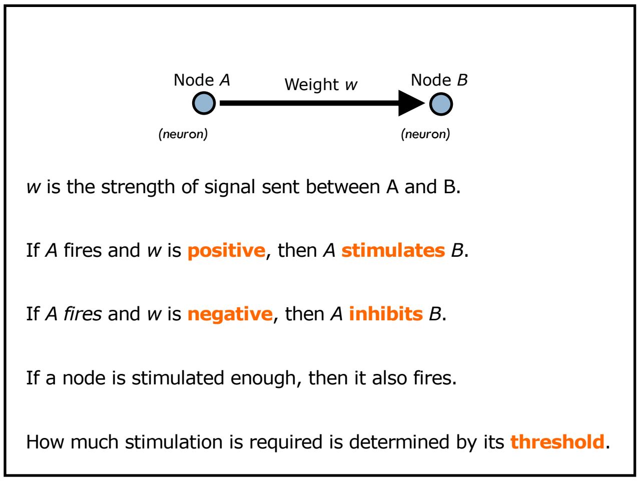


## Our nervous system: *the computer science view*

the human brain is a large collection of interconnected neurons

a NEURON is a brain cell

- collect, process, and disseminate electrical signals
- Neurons are connected via synapses
- They FIRE depending on the conditions of the neighboring neurons



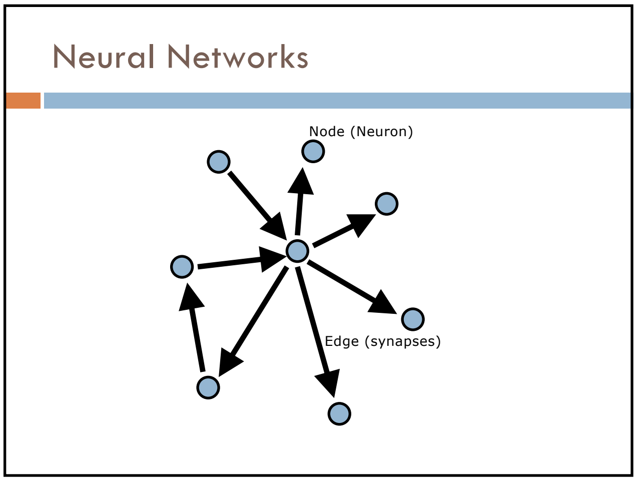
*w* is the strength of signal sent between A and B.

If *A* fires and *w* is **positive**, then *A* **stimulates** *B*.

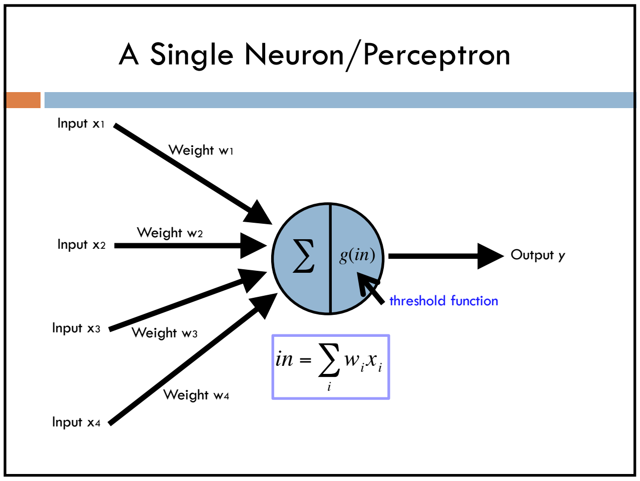
If *A fires* and *w* is **negative**, then *A* **inhibits** *B*.

If a node is stimulated enough, then it also fires.

How much stimulation is required is determined by its **threshold**.

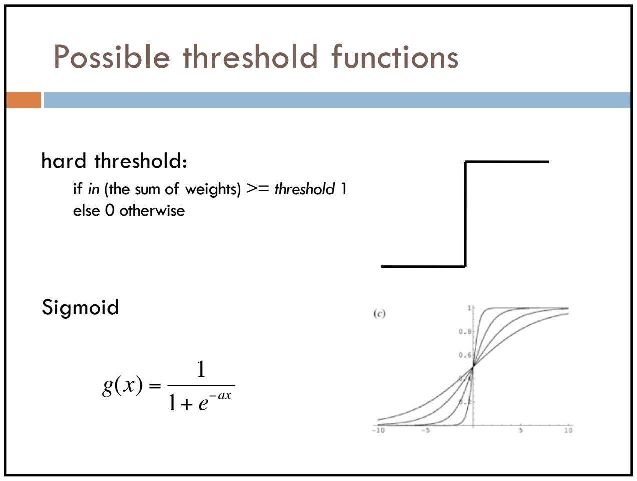


## Neural Networks



## A Single Neuron/Perceptron

$$in = \sum_i w_i x_i$$



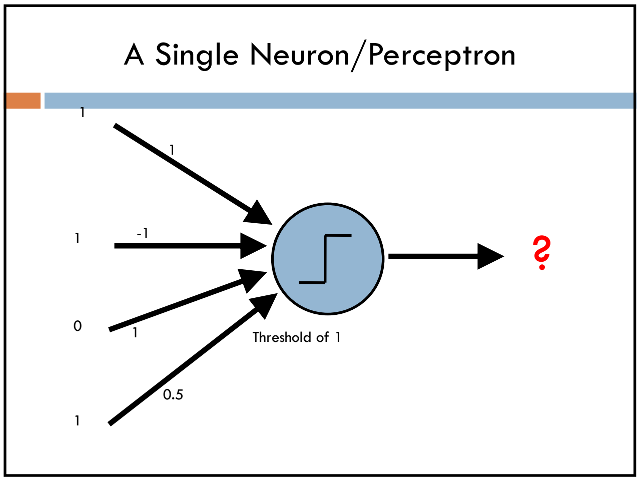
## Possible threshold functions

hard threshold:

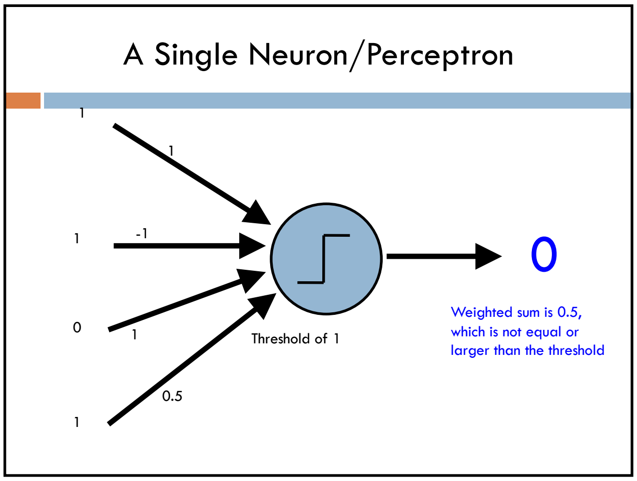
if *in* (the sum of weights) >= *threshold* 1
else 0 otherwise

Sigmoid

$$g(x) = \frac{1}{1 + e^{-ax}}$$

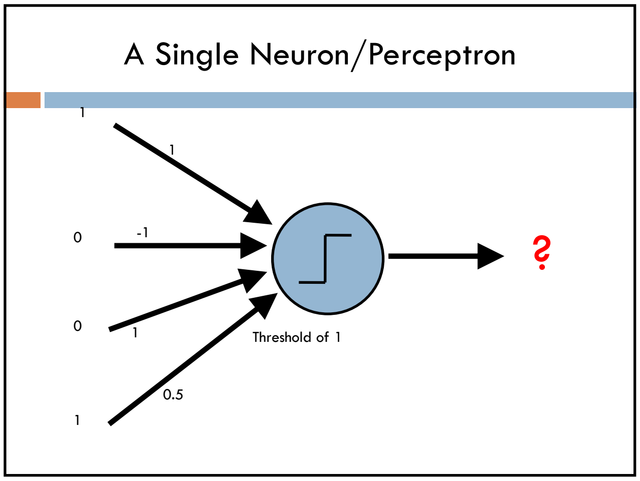


## A Single Neuron/Perceptron

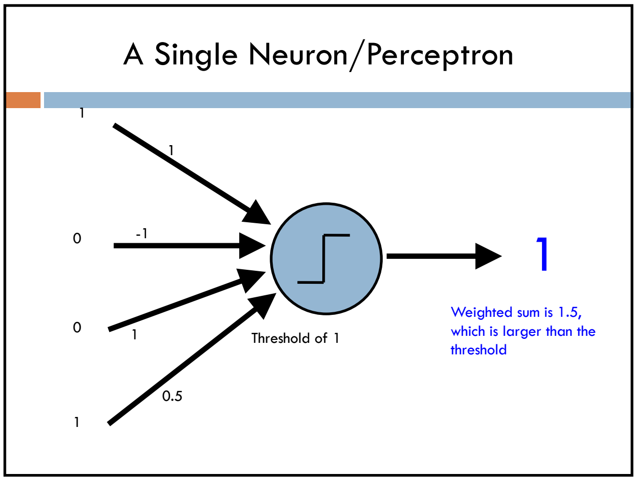


## A Single Neuron/Perceptron

Weighted sum is 0.5, which is not equal or larger than the threshold

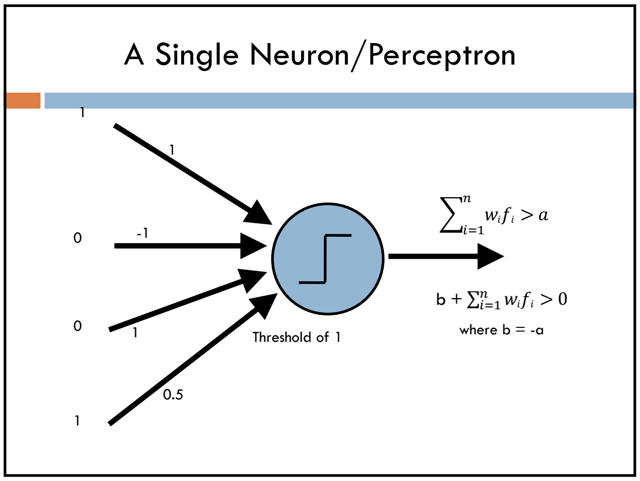


## A Single Neuron/Perceptron



## A Single Neuron/Perceptron

Weighted sum is 1.5, which is larger than the threshold



## A Single Neuron/Perceptron

$\sum_{i=1}^{n} w_i f_i > a$

$b + \sum_{i=1}^{n} w_i f_i > 0$

where b = -a