CS440/ECE448 Lecture 22: Linear Classifiers

Mark Hasegawa-Johnson, 3/2020
Including Slides by
Svetlana Lazebnik, 10/2016

Linear Classifiers

- Classifiers
- Perceptron
- Linear classifiers in general
- Logistic regression

Classifiers example: dogs versus cats

Can you write a program that can tell which ones are dogs, and which ones are cats?

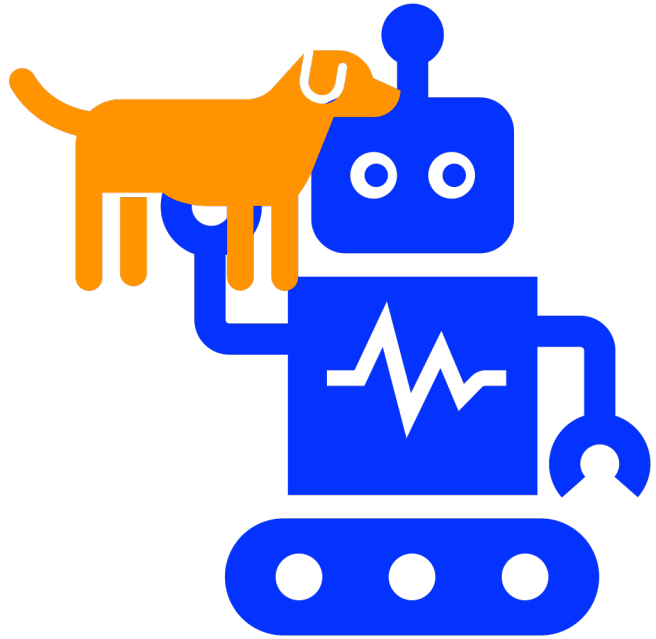
Classifiers example: dogs versus cats

Can you write a program that can tell which ones are dogs, and which ones are cats?

Idea #1: Cats are smaller than dogs.

Our robot will pick up the animal and weigh it.

If it weighs more than 20 pounds, call it a dog. Otherwise, call it a cat.



Classifiers example: dogs versus cats

Can you write a program that can tell which ones are dogs, and which ones are cats?

Oops.

Classifiers example: dogs versus cats

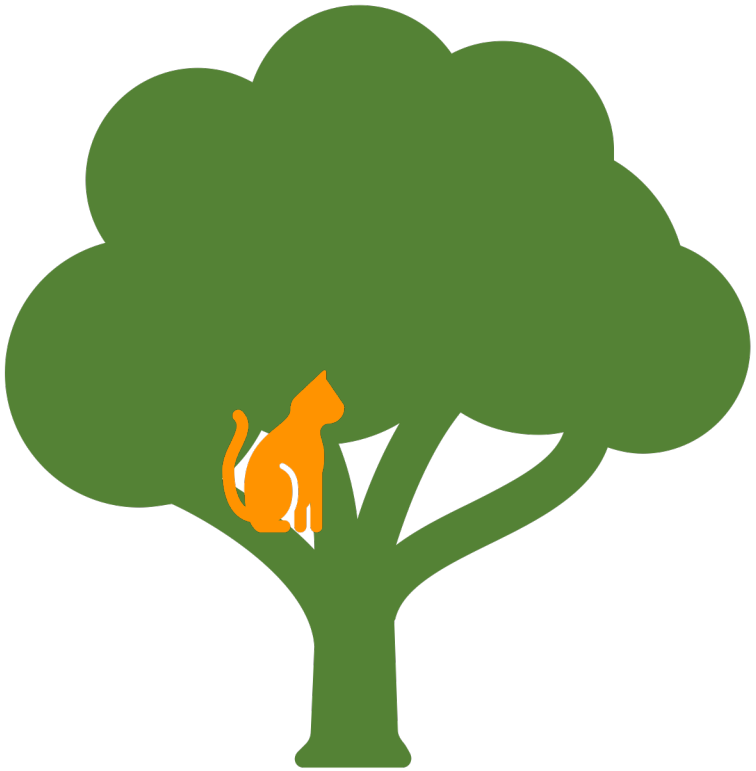
Can you write a program that can tell which ones are dogs, and which ones are cats?

Idea #2: Dogs are tame, cats are wild.

We'll try the following experiment: 40 different people call the animal's name. Count how many times the animal comes when called.

If the animal comes when called, more than 20 times out of 40, it's a dog.

If not, it's a cat.



Classifiers example: dogs versus cats

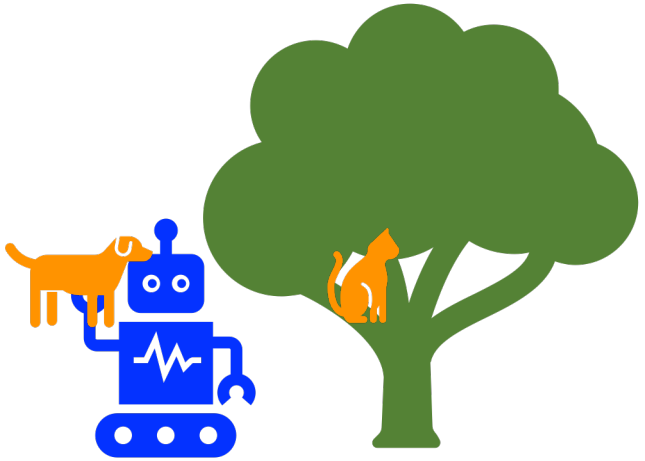
Can you write a program that can tell which ones are dogs, and which ones are cats?

Oops.

By Smok Bazyli - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=16864492

Classifiers example: dogs versus cats

Can you write a program that can tell which ones are dogs, and which ones are cats?



Idea #3:

$x_1 =$ # times the animal comes when called (out of 40).

$x_2 =$ weight of the animal, in pounds.

If $0.5x_1 + 0.5x_2 > 20$, call it a dog.

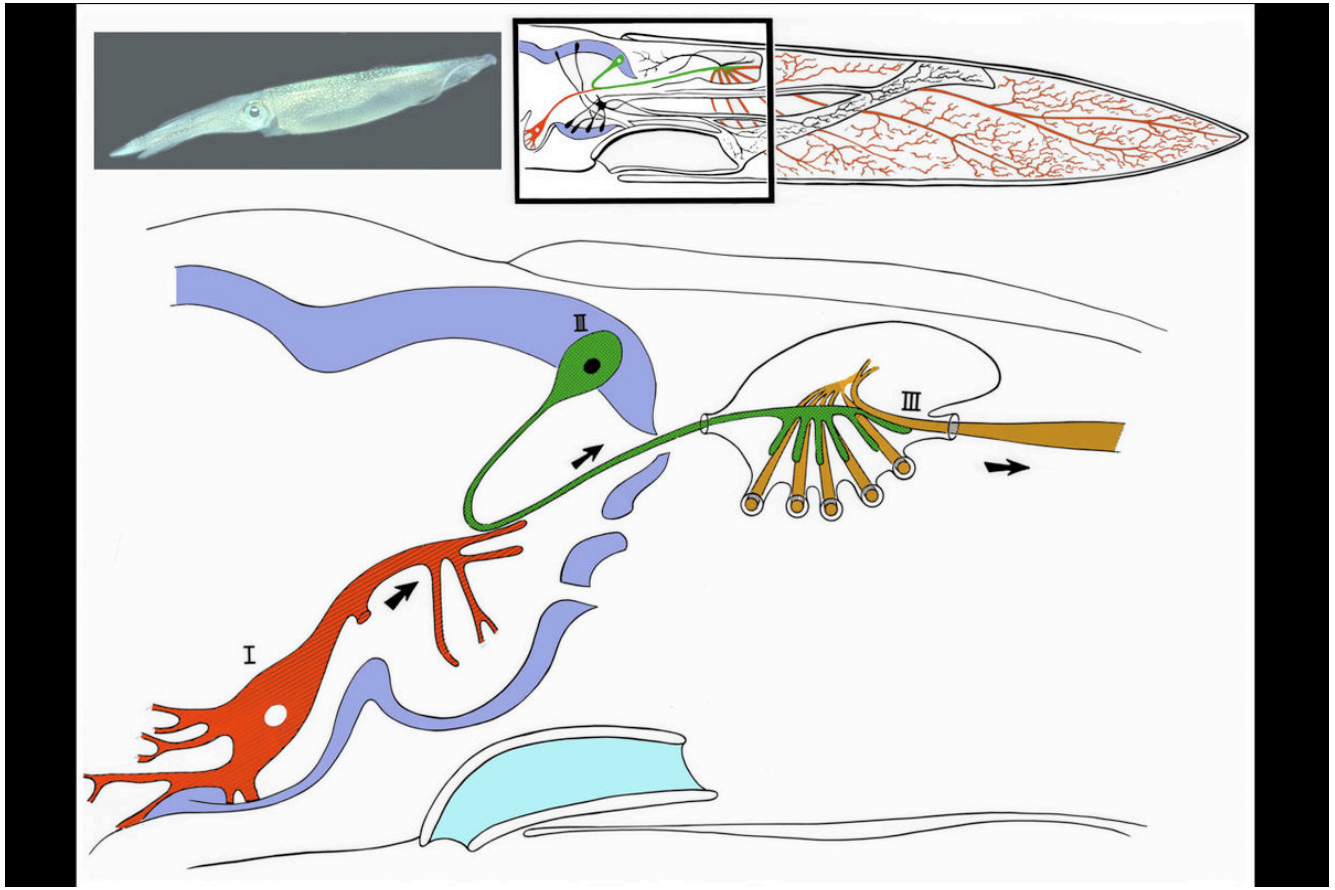
Otherwise, call it a cat.

This is called a "linear classifier" because $0.5x_1 + 0.5x_2 = 20$ is the equation for a line.

Linear Classifiers

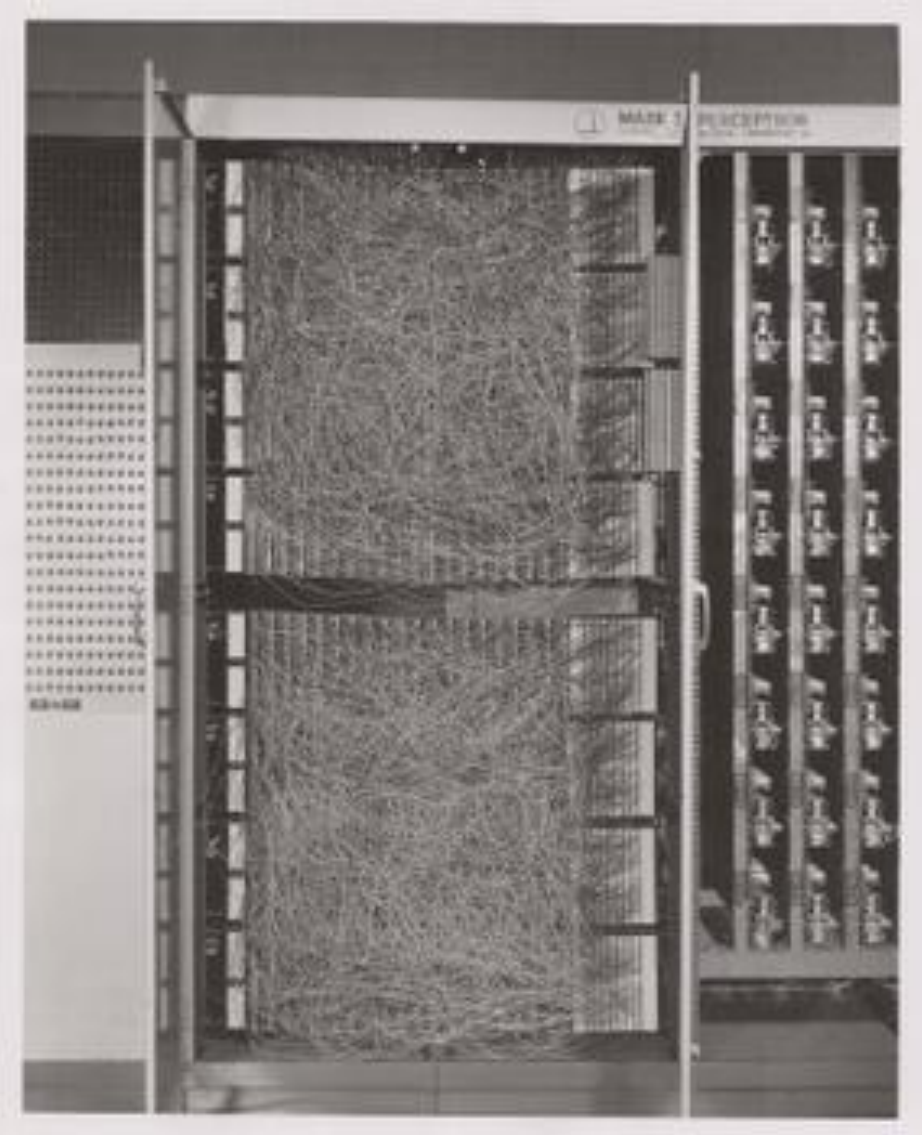
- Classifiers
- Perceptron
- Linear classifiers in general
- Logistic regression

The Giant Squid Axon



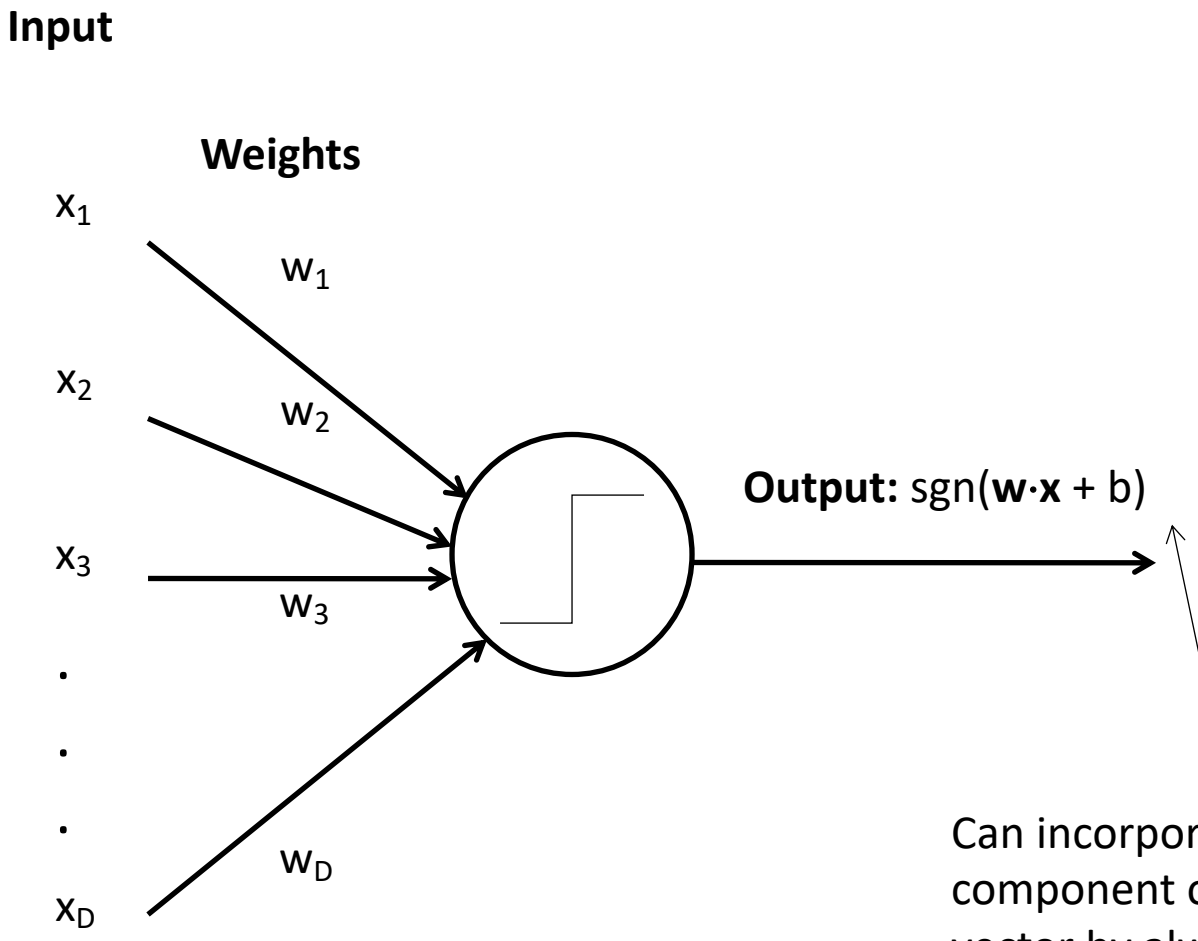
- 1909: Williams discovers that the giant squid has a giant neuron (axon 1mm thick)
- 1939: Young finds a giant synapse (fig. shown: Llinás, 1999, via Wikipedia). Hodgkin & Huxley put in voltage clamps.
- 1952: Hodgkin & Huxley publish an electrical current model for the generation of binary action potentials from real-valued inputs.

Perceptron



- 1959: Rosenblatt is granted a patent for the “perceptron,” an electrical circuit model of a neuron.

Perceptron



Perceptron model: action potential = signum(affine function of the features)

$y^* = \text{sgn}(w_1x_1 + w_2x_2 + \ldots + w_Dx_D + b)$
$= \text{sgn}(\vec{w}^T\vec{x})$

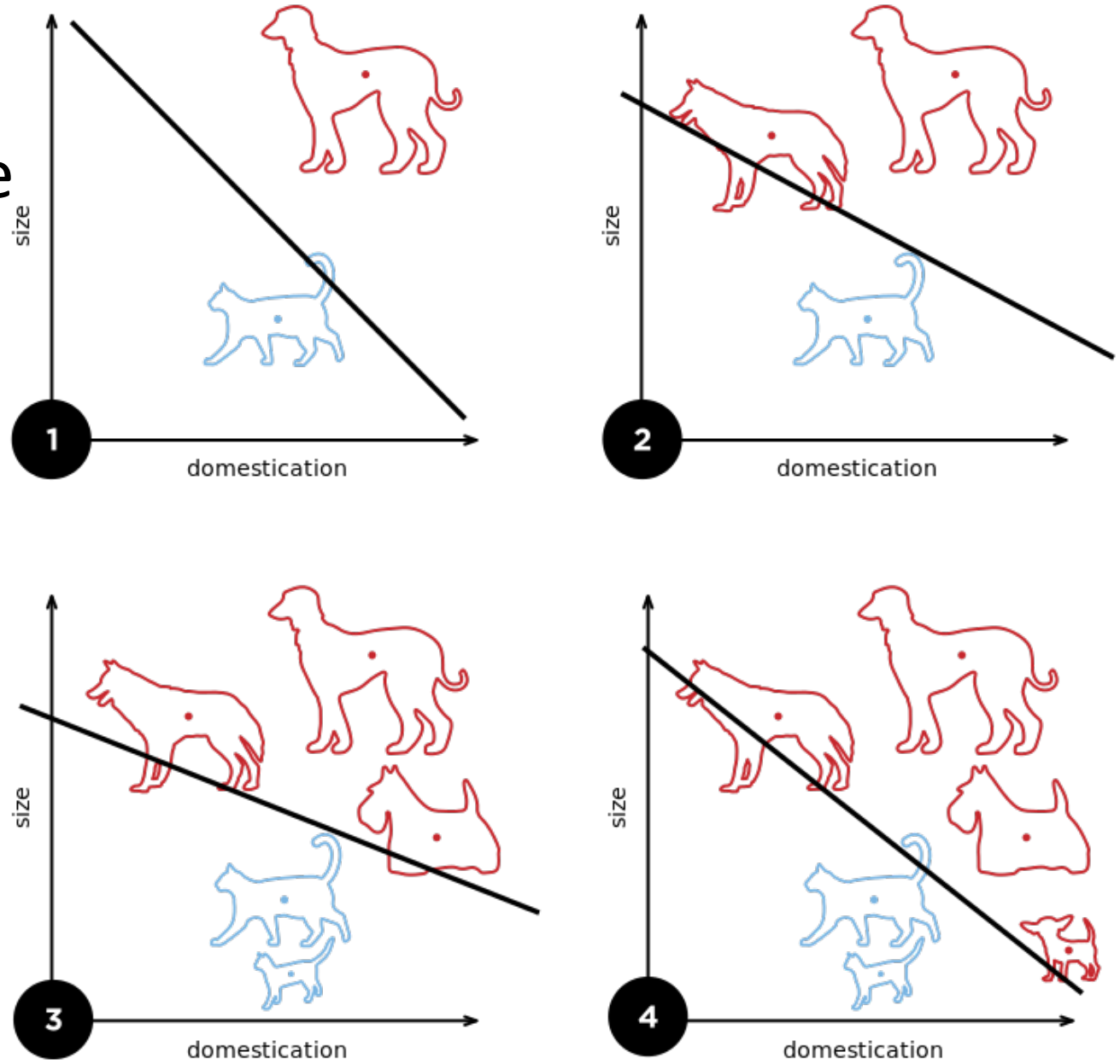
Where $\vec{w} = [w_1, \ldots, w_D, b]^T$
and $\vec{x} = [x_1, \ldots, x_D, 1]^T$

Can incorporate bias as component of the weight vector by always including a feature with value set to 1

Perceptron

Rosenblatt's big innovation: the perceptron learns from examples.

- Initialize weights randomly
- Cycle through training examples in multiple passes (*epochs*)
- For each training example:
  - If classified correctly, do nothing
  - If classified incorrectly, update weights



By Elizabeth Goodspeed - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=40188333

Perceptron

For each training instance $\vec{x}$ with ground truth label $y \in \{-1,1\}$:

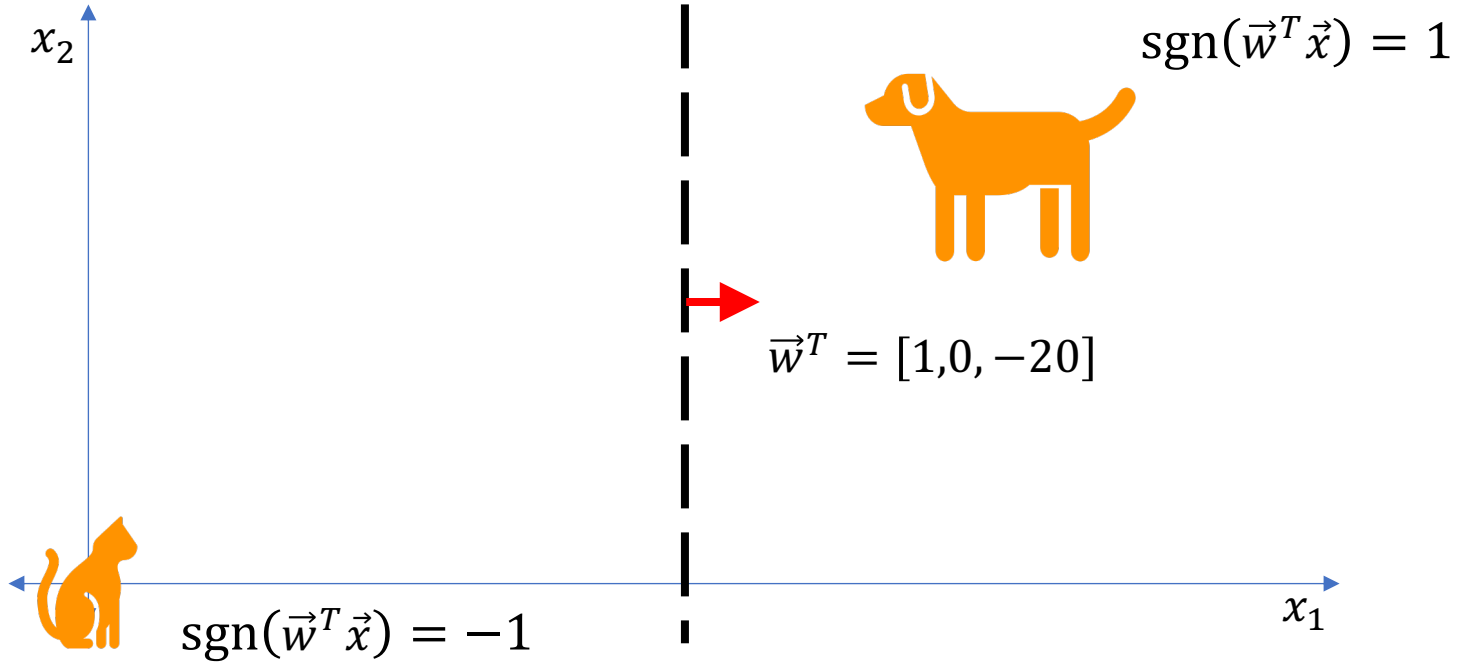
- Classify with current weights: $y^* = \text{sgn}(\vec{w}^T \vec{x})$
- Update weights:
  - if $y = y^*$ then do nothing
  - If $y \neq y^*$ then $\vec{w} = \vec{w} + \eta y \vec{x}$
  - η (eta) is a "learning rate." More about that later.

Perceptron training example: dogs vs. cats

- Let's start with the rule "if it comes when called (by at least 20 different people out of 40), it's a dog."
- So if $x_1 =$ # times it comes when called, then the rule is:

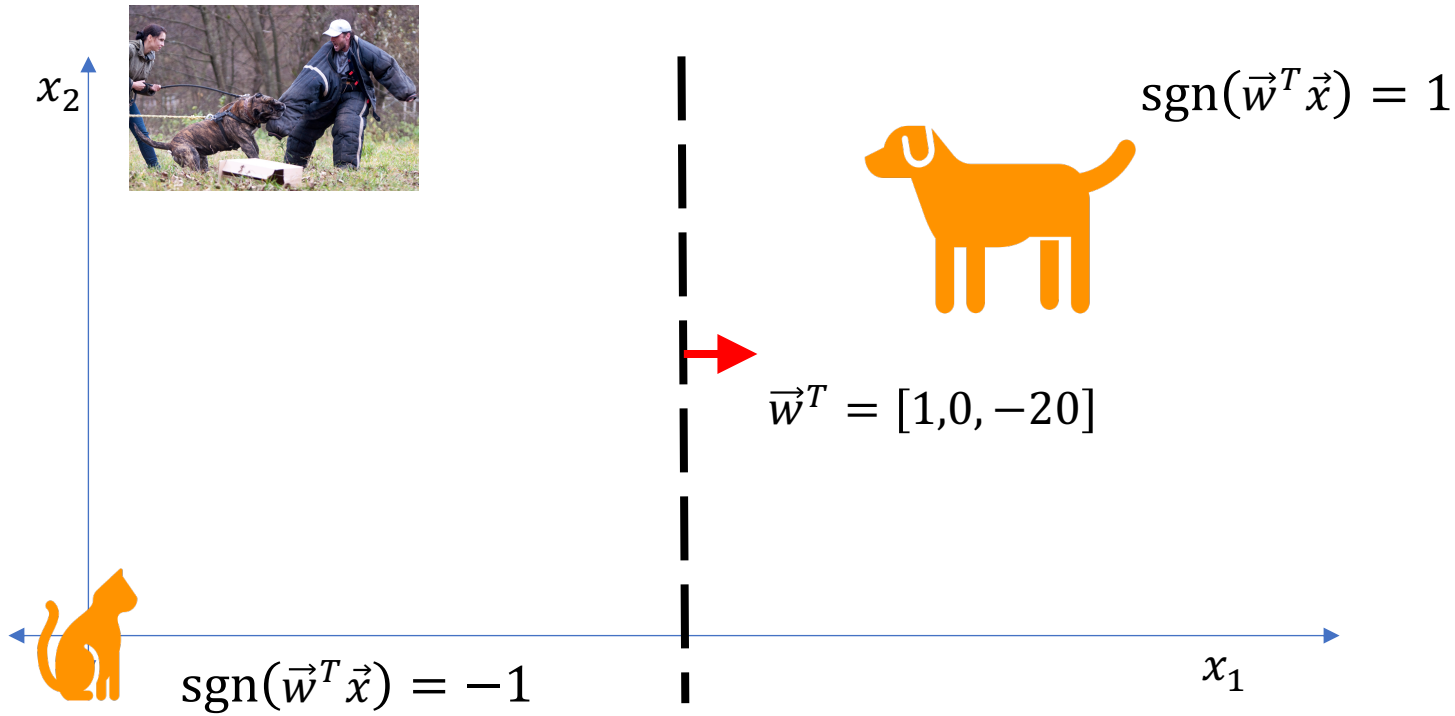
If $x_1 - 20 > 0$, call it a dog.

In other words, $y^* = \text{sgn}(\vec{w}^T \vec{x})$, where $\vec{w}^T = [1,0,-20]$, and $\vec{x}^T = [x_1, x_2, 1]$.



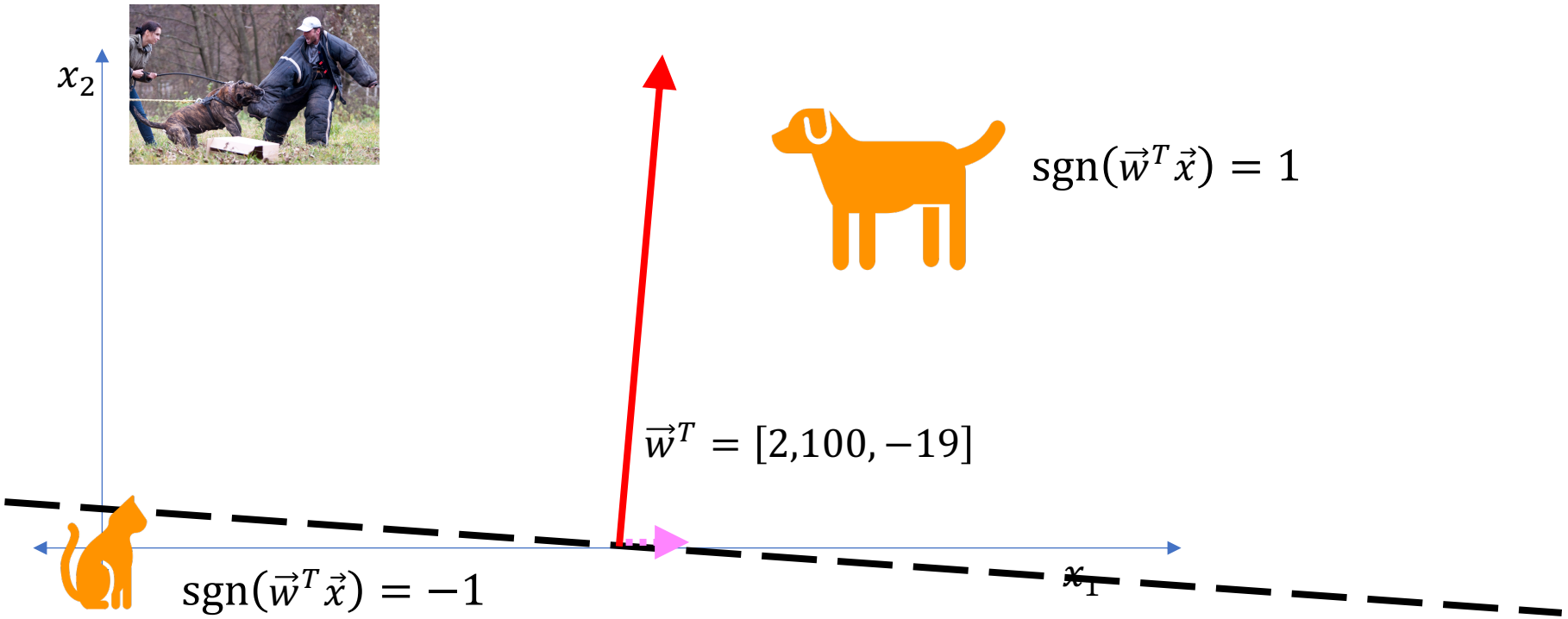
Perceptron training example: dogs vs. cats

- The **Presa Canario** gets misclassified as a cat ($y = 1$, but $y^* = -1$) because it only obeys its trainer ($x_1 = 1$), and nobody else. But we notice that the **Presa Canario**, though it rarely comes when called, is very large ($x_2 = 100$ pounds), so we have $\vec{x}^T = [x_1, x_2, 1] = [1,100,1]$.



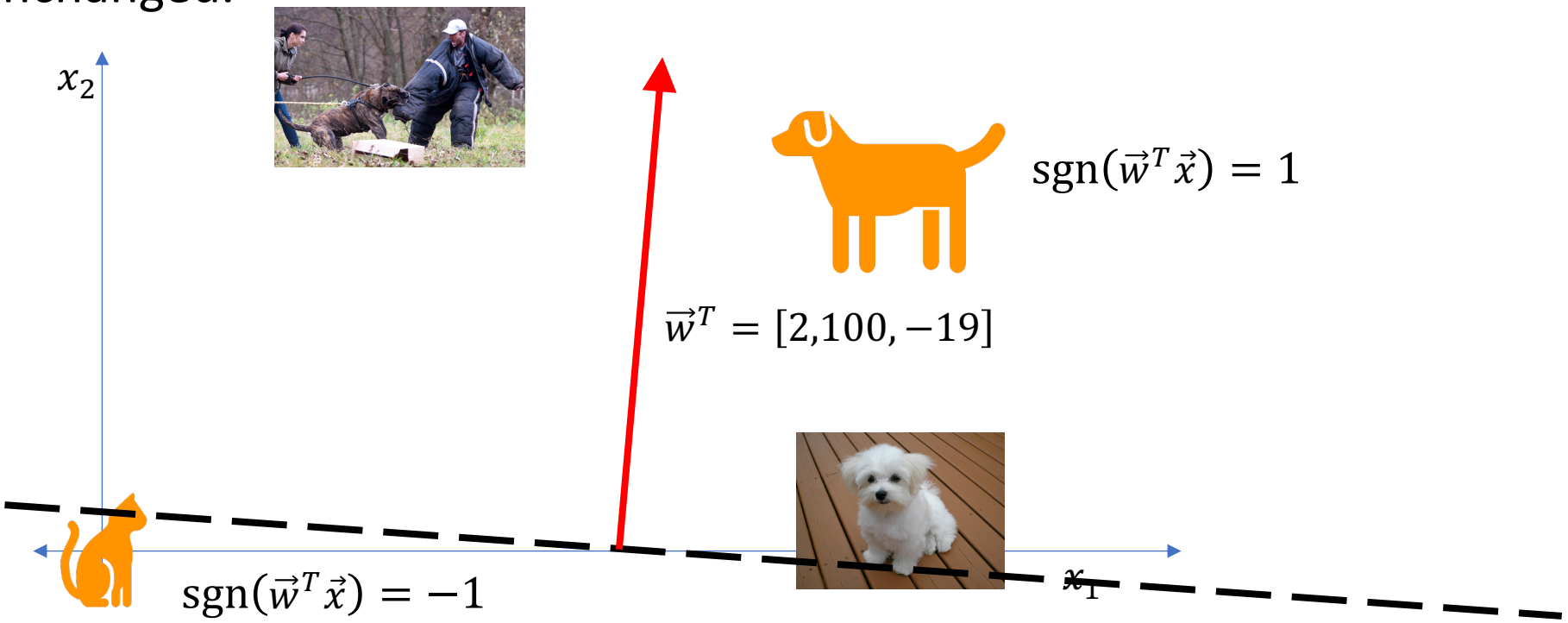
Perceptron training example: dogs vs. cats

- The **Presa Canario** gets misclassified as a cat ($y = 1$, but $y^* = -1$) because it only obeys its trainer ($x_1 = 1$), and nobody else. But we notice that the **Presa Canario**, though it rarely comes when called, is very large ($x_2 = 100$ pounds), so we have $\vec{x}^T = [x_1, x_2, 1] = [1,100,1]$.
- So we update: $\vec{w} = \vec{w} + y\vec{x} = [1,0,-20] + [1,100,1] = [2,100,-19]$



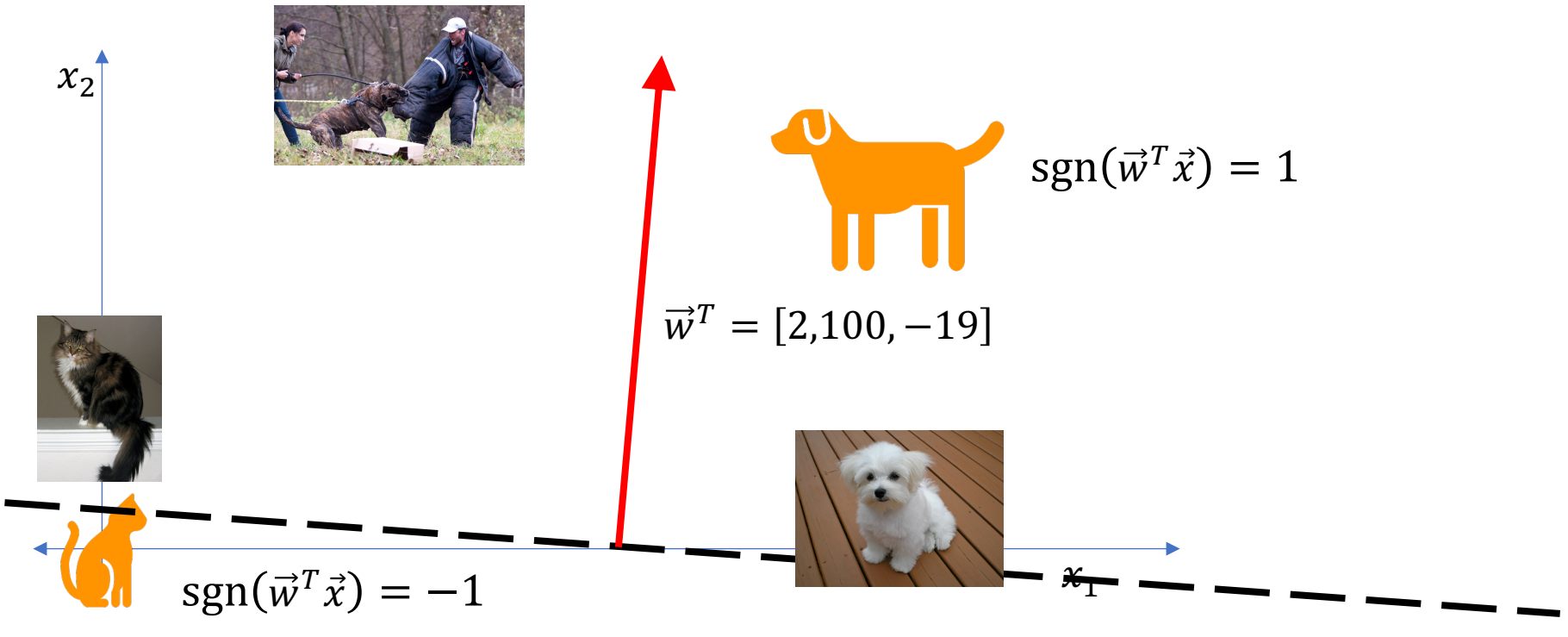
Perceptron training example: dogs vs. cats

- The **Maltese**, though it's small ($x_2 = 10$ pounds), is very tame ($x_1 = 40$): $\vec{x} = [x_1, x_2, 1] = [40,10,1]$.
- But it's correctly classified! $y^* = \text{sgn}(\vec{w}^T\vec{x}) = \text{sgn}(2\times40 + 100\times10 - 19) = +1$, which is equal to $y = 1$.
- So the $\vec{w}$ vector is unchanged.



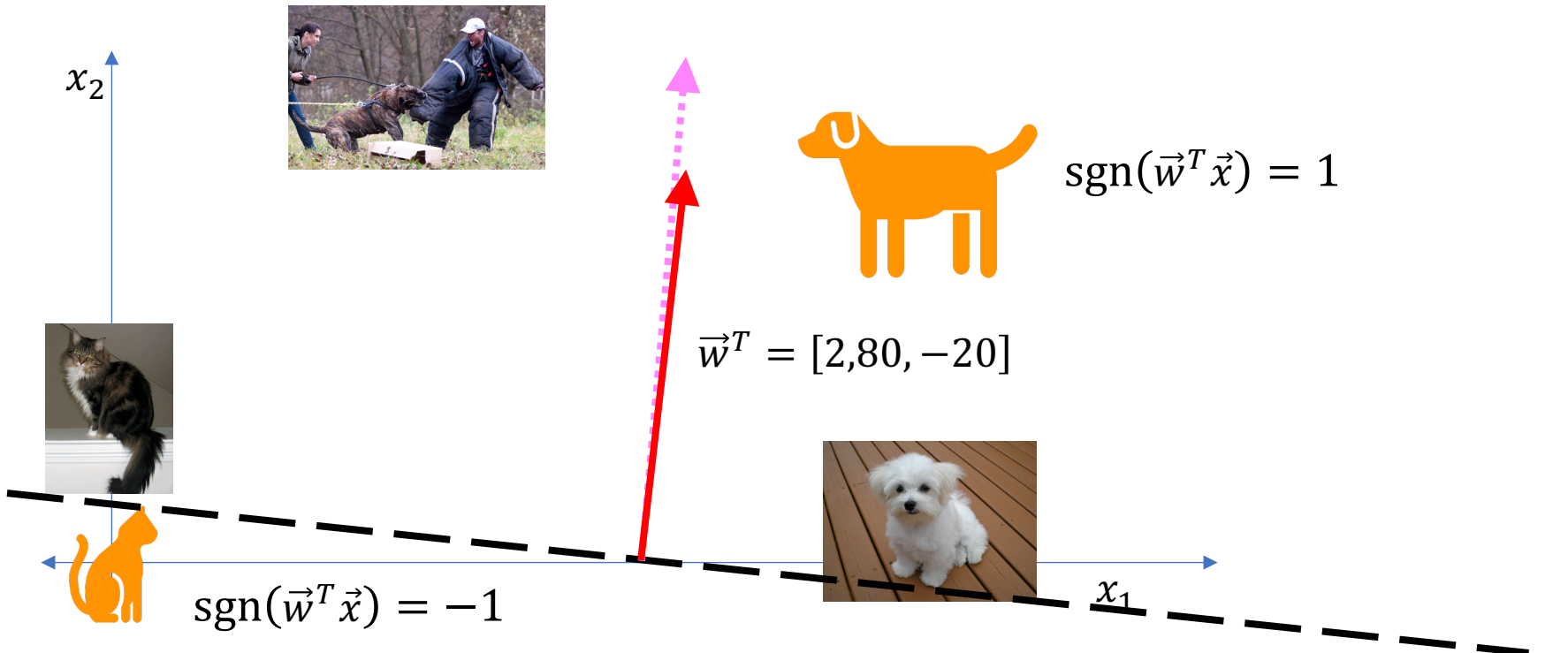
Perceptron training example: dogs vs. cats

- The **Maine Coon** cat is big ($x_2 = 20$ pounds: $\vec{x} = [0,20,1]$), so it gets misclassified as a dog (true label is $y = -1$="cat," but the classifier thinks $y^* = 1$="dog").



Perceptron training example: dogs vs. cats

- The **Maine Coon** cat is big ($x_2 = 20$ pounds: $\vec{x} = [0,20,1]$), so it gets misclassified as a dog (true label is $y = -1$="cat," but the classifier thinks $y^* = 1$="dog").
- So we update: $\vec{w} = \vec{w} + y\vec{x} = [2,100,-19] + (-1)\times[0,20,1] = [2,80,-20]$



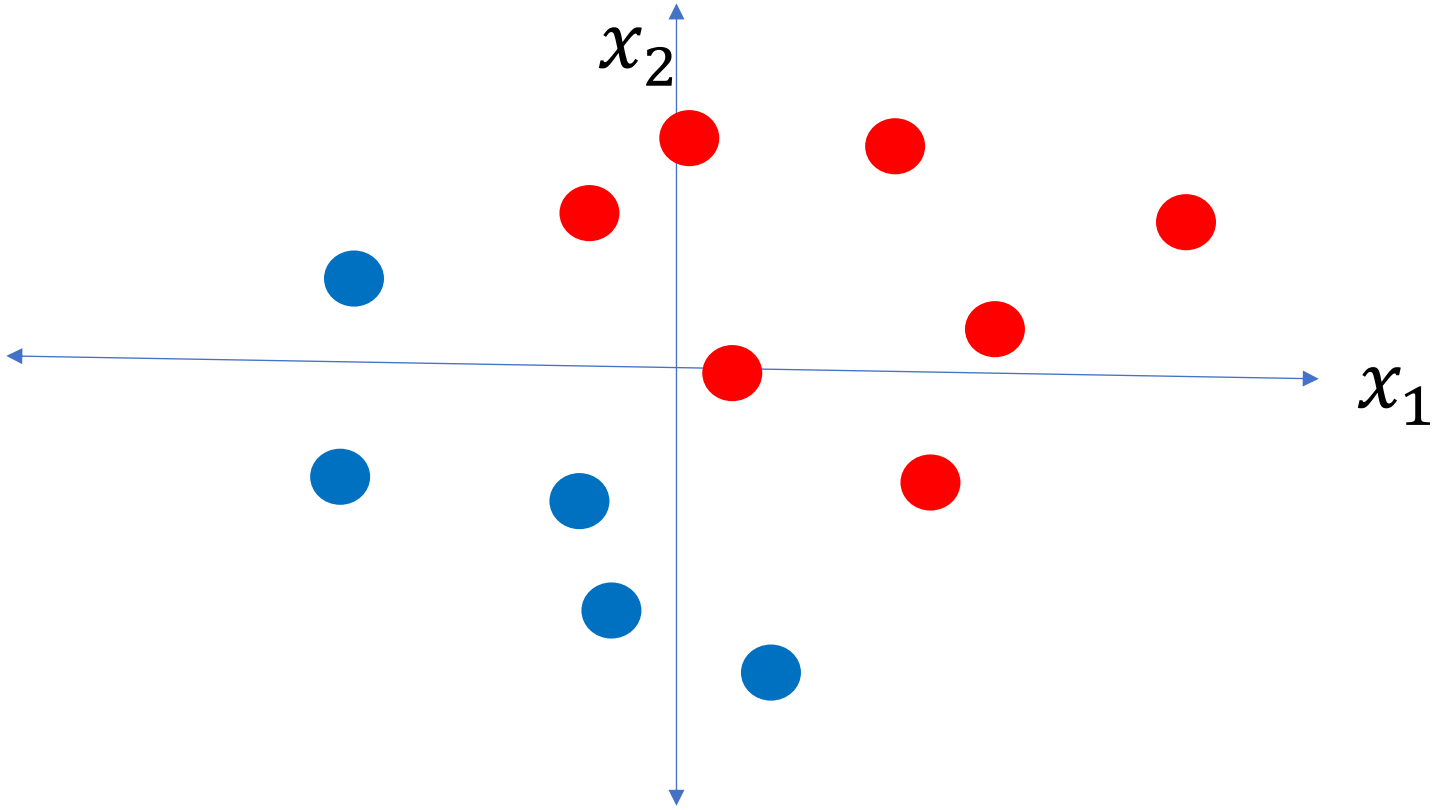
Perceptron: Proof of Convergence

- Definition: **linearly separable:**
  - A dataset is linearly separable if and only if there exists a vector, $\vec{w}$, such that the ground truth label of each token is given by $y = \text{sgn}(\vec{w}^T \vec{x})$.
- Theorem (proved in the next few slides):

If the data are linearly separable, then the perceptron learning algorithm converges to a correct solution, even with a learning rate of η=1.

Perceptron: Proof of Convergence

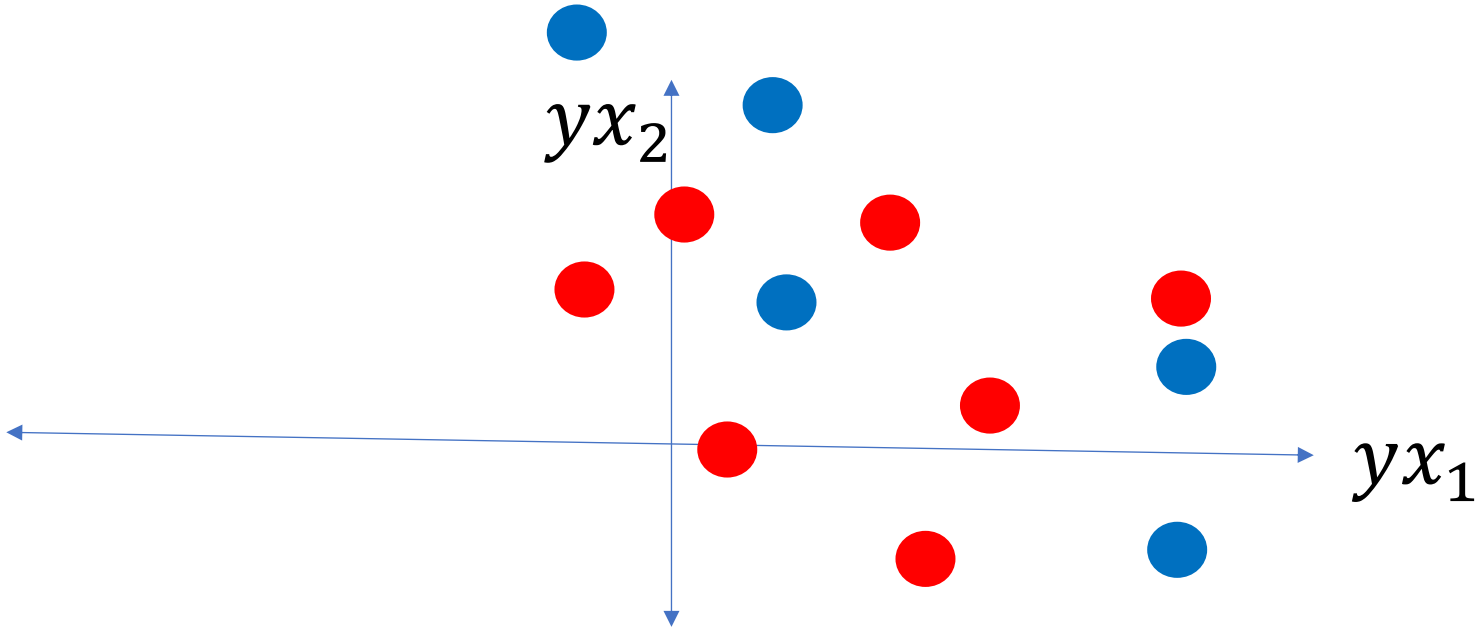
Suppose the data are linearly separable. For example, suppose red dots are the class y=1, and blue dots are the class y=-1:



Perceptron: Proof of Convergence

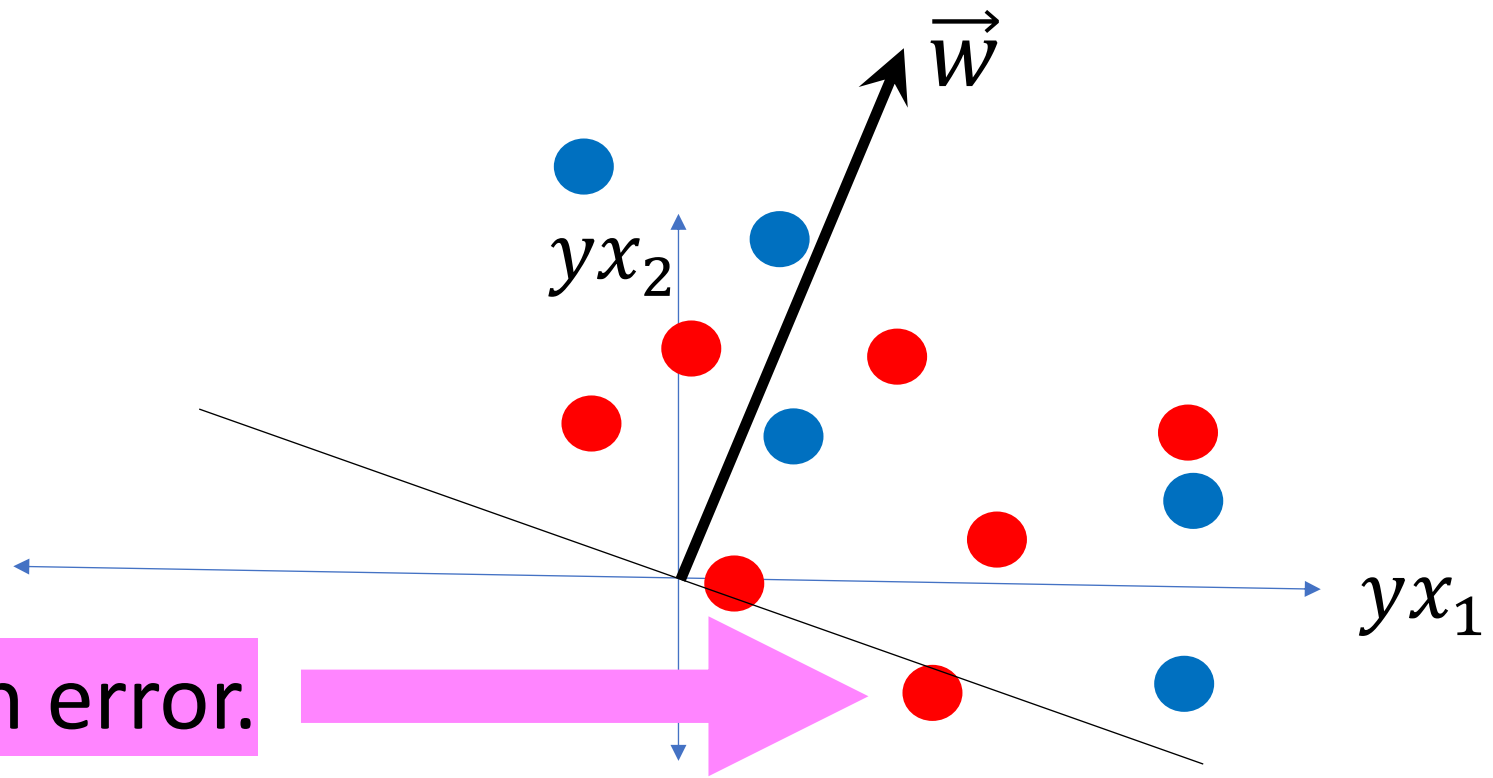
Instead of plotting $\vec{x}$, plot $y\vec{x}$. The red dots are unchanged; the blue dots are multiplied by -1.

- Since the original data were linearly separable, the new data are all in the same half of the feature space.



Perceptron: Proof of Convergence

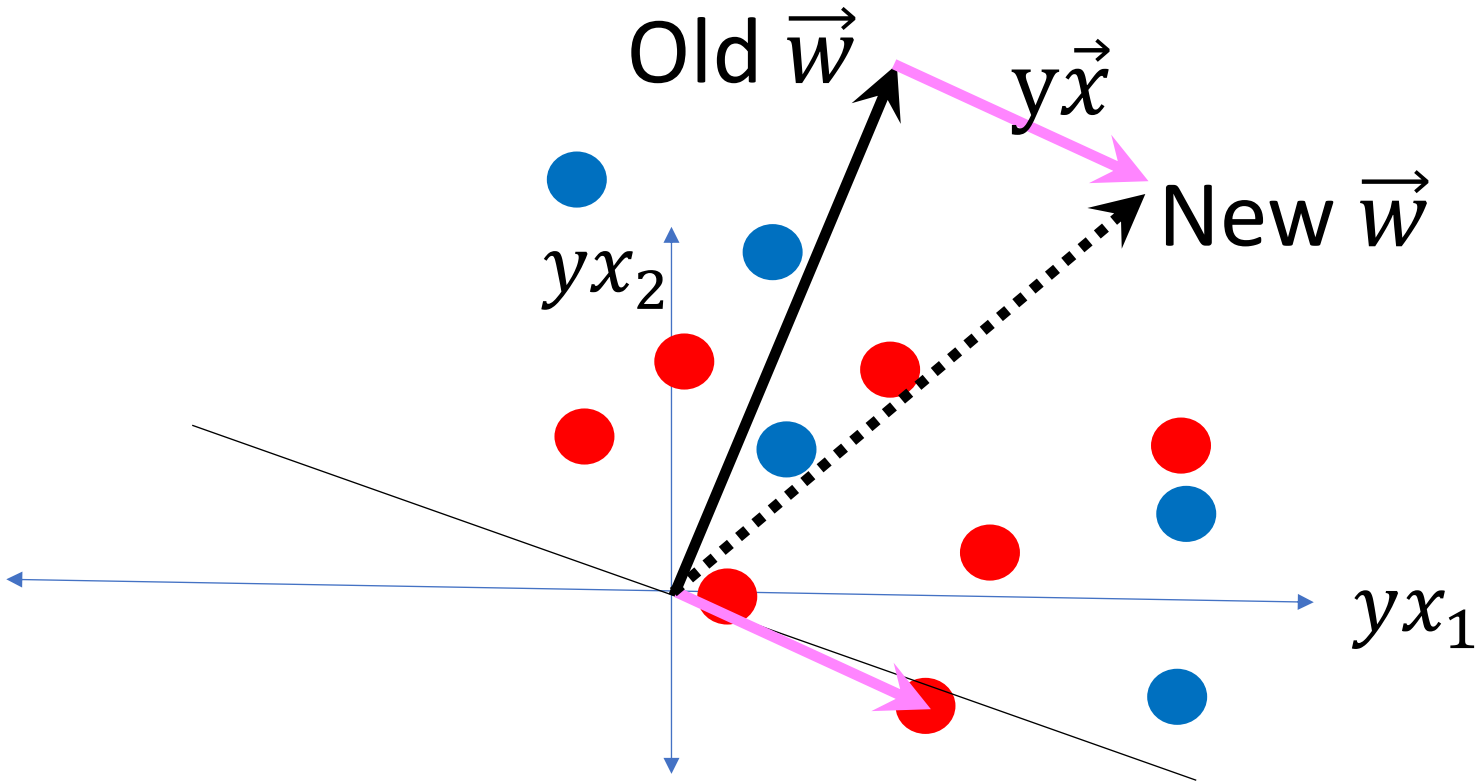
Suppose we start out with some initial guess, $\vec{w}$, that makes mistakes. In other words, $\text{sgn}(\vec{w}^T(y\vec{x})) = -1$ for some of the tokens.



Oops! An error.

Perceptron: Proof of Convergence

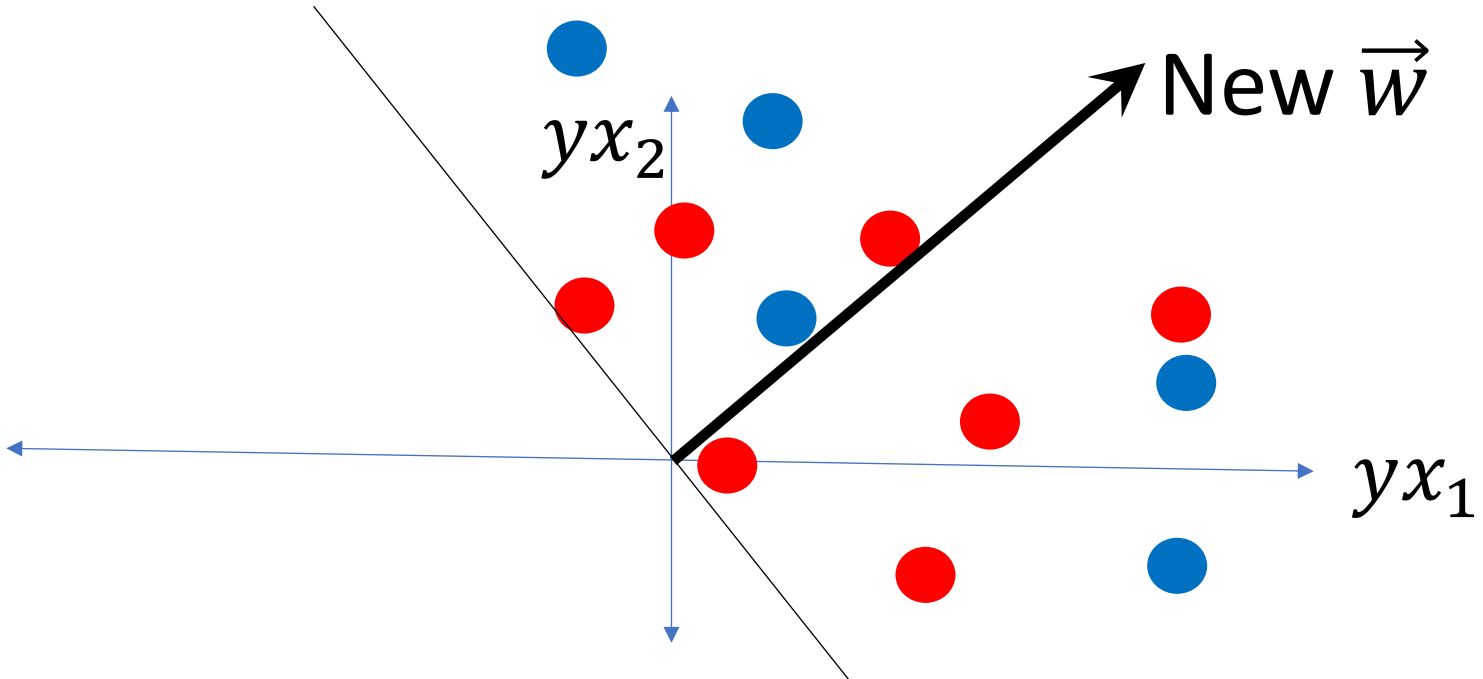
In that case, $\vec{w}$ will be updated by adding $y\vec{x}$ to it.



Perceptron: Proof of Convergence

If there is any $\vec{w}$ such that $\mathrm{sgn}(\vec{w}^T(y\vec{x})) = 1$ for all tokens, then this procedure will eventually find it.

- If the data are linearly separable, the perceptron algorithm converges to a correct solution, even with η=1.



What about non-separable data?

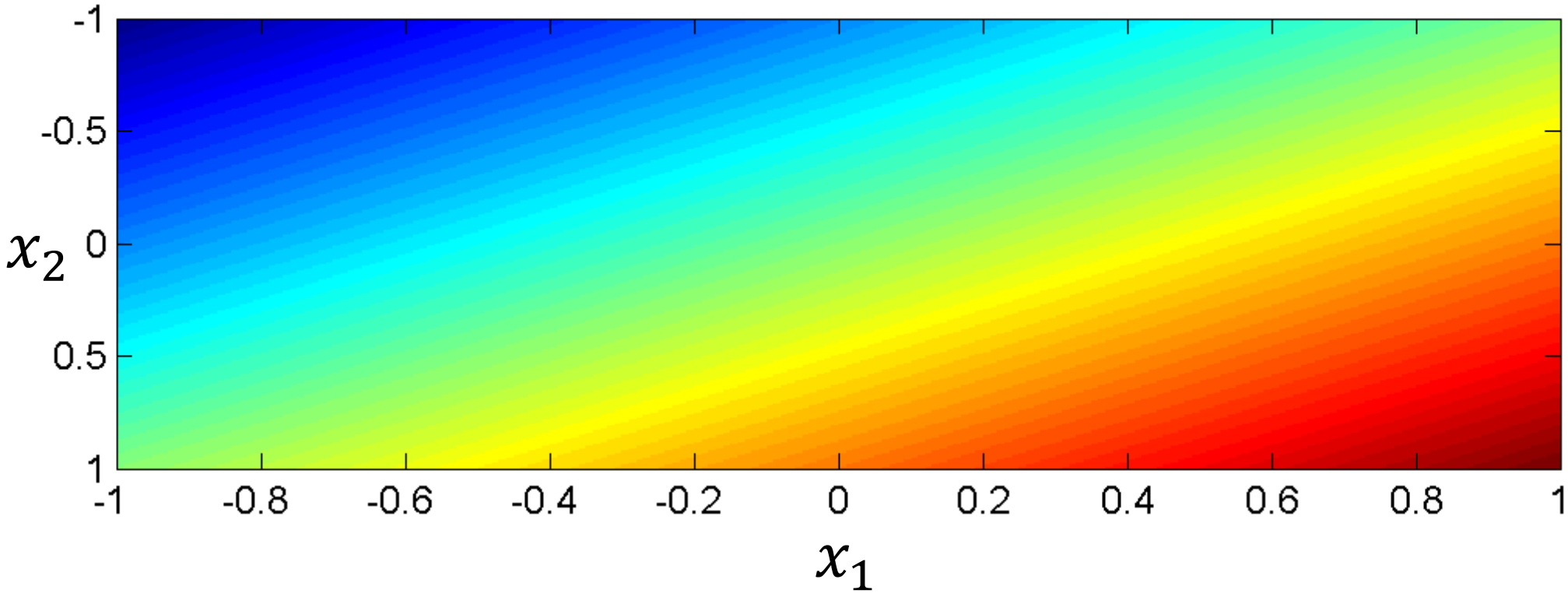
- If the data are NOT linearly separable, then the perceptron with η=1 doesn't converge.
- In fact, that's what η is for.
- Remember that $\vec{w} = \vec{w} + \eta y\vec{x}$.
- We can force the perceptron to stop wiggling around by forcing η (and therefore $\eta y\vec{x}$) to get gradually smaller and smaller.
- This works: for the n^{th} training token, set $\eta = \frac{1}{n}$.
- Notice: $\sum_{n=1}^{\infty} \frac{1}{n}$ is infinite. Nevertheless, $\eta = \frac{1}{n}$ works, because the $y\vec{x}$ tokens are not all in the same direction.

Linear Classifiers

- Classifiers
- Perceptron
- Linear classifiers in general
- Logistic regression

Linear Classifiers in General

The function $b + \sum_{j=1}^{D} w_j x_j$ is an affine function of the features x_j. That means that its contours are all straight lines. Here is an example of such a function, plotted as variations of color in a two-dimensional space x_1 by x_2:



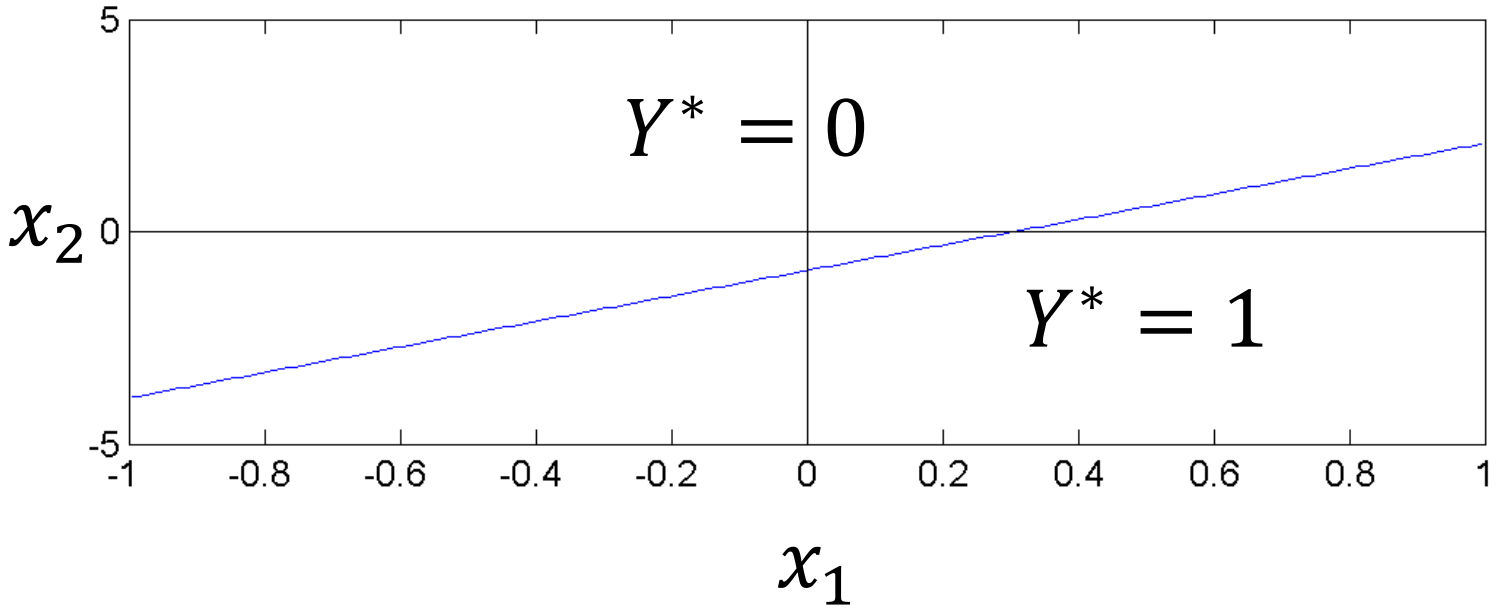
Linear Classifiers in General

Consider the classifier

$$Y^* = 1 \quad \text{if:} \quad b + \sum_{j=1}^{D} w_j x_j > 0$$

$$Y^* = 0 \quad \text{if:} \quad b + \sum_{j=1}^{D} w_j x_j < 0$$

This is called a "linear classifier" because the boundary between the two classes is a line. Here is an example of such a classifier, with its boundary plotted as a line in the two-dimensional space x_1 by x_2:

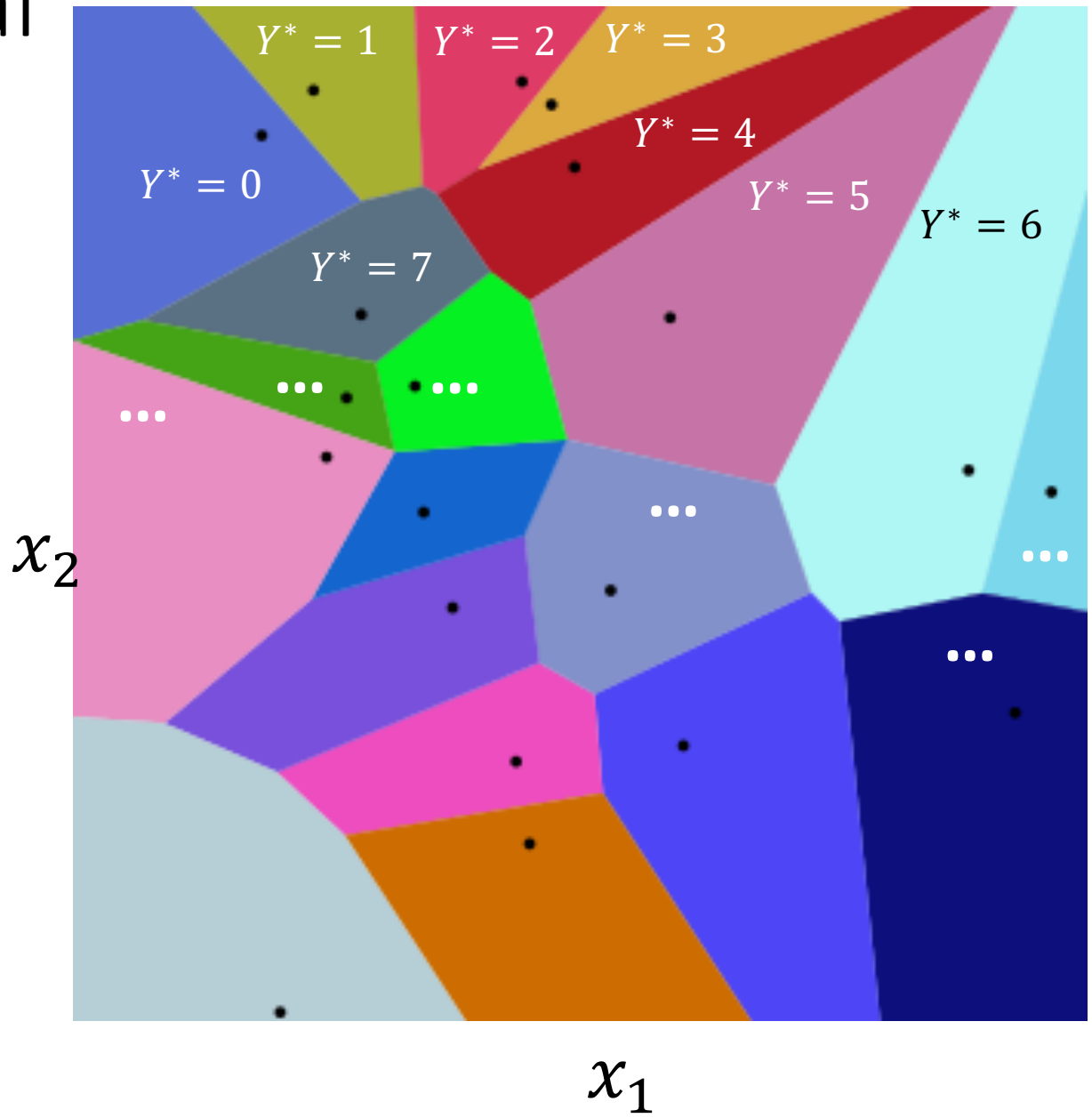


Linear Classifiers in General

Consider the classifier

$$Y^* = \arg\max_c \left(b_c + \sum_j w_{cj} x_j \right)$$

- This is called a "multi-class linear classifier."
- The regions $Y^* = 0,\ Y^* = 1,\ Y^* = 2$ etc. are called "Voronoi regions."
- They are regions with piece-wise linear boundaries. Here is an example from Wikipedia of Voronoi regions plotted in the two-dimensional space x_1 by x_2:



Linear Classifiers in General

When the features are binary ($x_j \in \{0,1\}$), many (but not all!) binary functions can be re-written as linear functions. For example, the function

$$Y^* = (x_1 \vee x_2)$$

can be re-written as

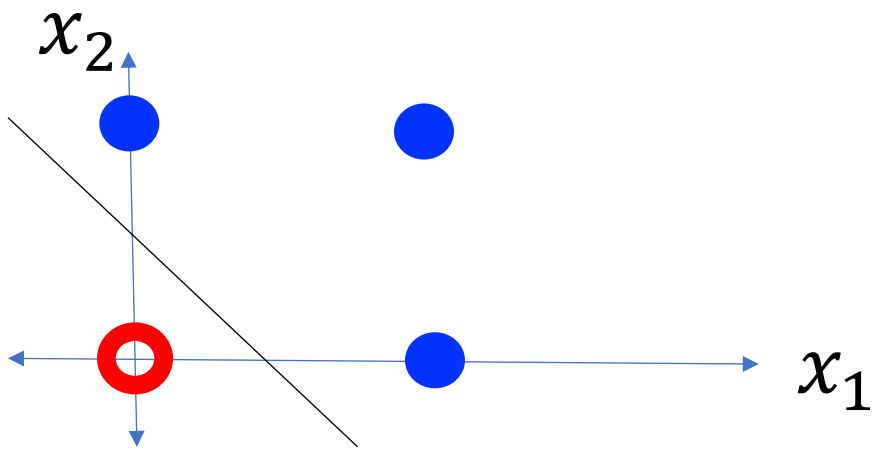
$$Y^* = 1 \quad \text{if: } x_1 + x_2 - 0.5 > 0$$

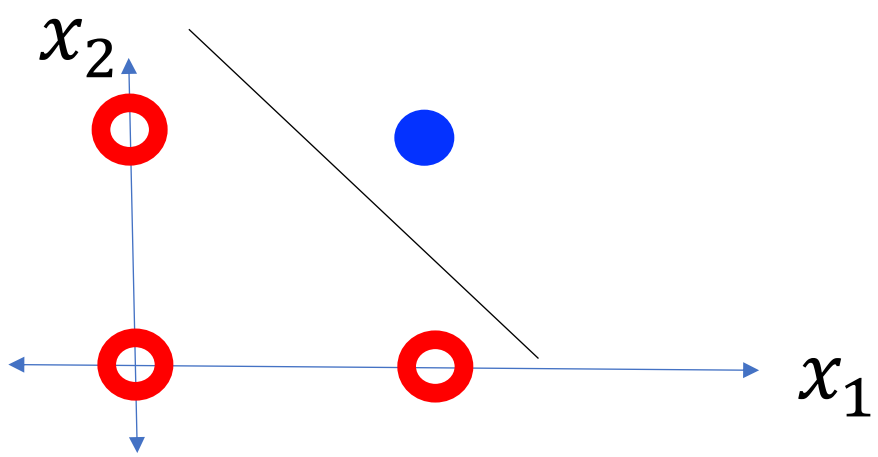
Similarly, the function

$$Y^* = (x_1 \wedge x_2)$$

can be re-written as

$$Y^* = 1 \quad \text{if: } x_1 + x_2 - 1.5 > 0$$



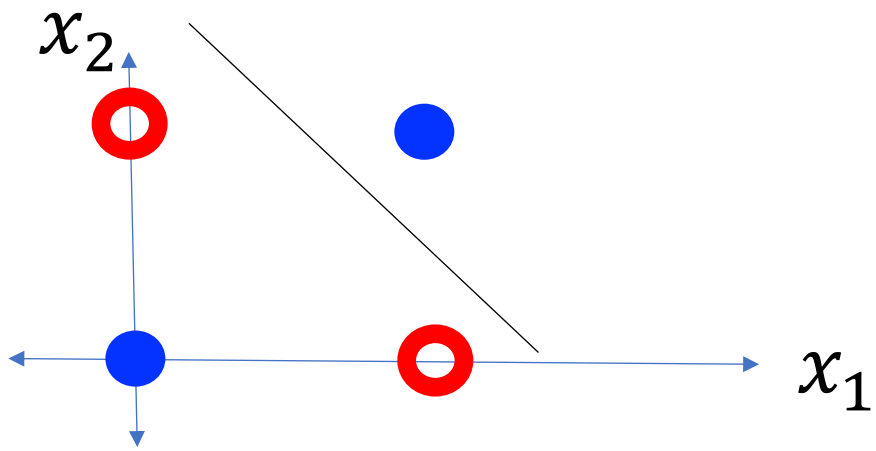


Linear Classifiers in General

- Not all logical functions can be written as linear classifiers!
- Minsky and Papert wrote a book called *Perceptrons* in 1969. Although the book said many other things, the only thing most people remembered about the book was that:

"A linear classifier cannot learn an XOR function."

- Because of that statement, most people gave up working on neural networks from about 1969 to about 2006.
- Minsky and Papert also proved that a two-layer neural net can learn an XOR function. But most people didn't notice.



Linear Classifiers

Classification:

$$Y^* = \arg\max_c \left(b_c + \sum_{j=1}^{D} w_{cj} x_j \right)$$

- Where x_j are the features (binary, integer, or real), w_{cj} are the feature weights, and b_c is the offset for the c^{th} class.

Linear Classifiers

- Classifiers
- Perceptron
- Linear classifiers in general
- Logistic regression

Differentiable Perceptron

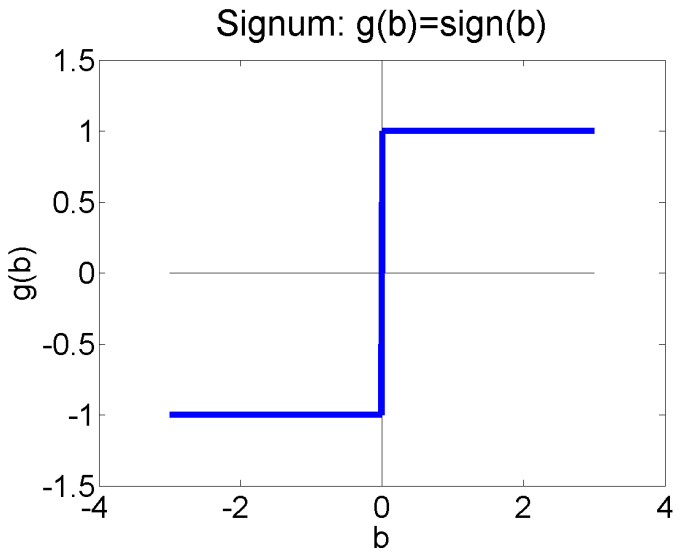
- Also known as a "one-layer feedforward neural network," also known as "logistic regression." Has been re-invented many times by many different people.
- Basic idea: replace the non-differentiable decision function

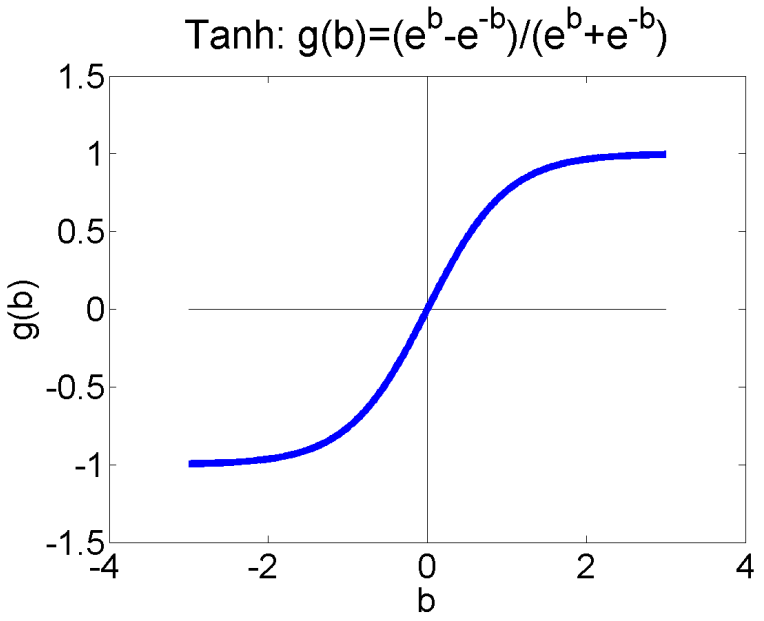
$$y^* = \text{sgn}(\vec{w}^T \vec{x})$$

with a differentiable decision function:

$$y^* = \tanh(\vec{w}^T \vec{x})$$

$$= \frac{1 - e^{-2w\vec{x}}}{1 + e^{-2\vec{w}^T \vec{x}}}$$





Why?

More about perceptron learning

Let's re-write the training data in a different way.

Suppose we have n training vectors, $\vec{x}_1$ through $\vec{x}_n$, where

$$\vec{x}_i = \left[x_{i1}, \ldots, x_{ij}, \ldots, x_{iD}, 1\right]^T$$

Each one has an associated ground-truth reference label $y_i \in \{-1,1\}$. The perceptron computes a classifier output $y_i^* = \mathrm{sgn}(\vec{w}^T \vec{x}_i)$ which is also $\in \{-1,1\}$.

The LOSS FUNCTION (a.k.a. the error rate on the training corpus) is

$$L(\vec{w}) = \frac{1}{4}\sum_{i=1}^{n}(y_i - y_i^*)^2$$

More about perceptron learning

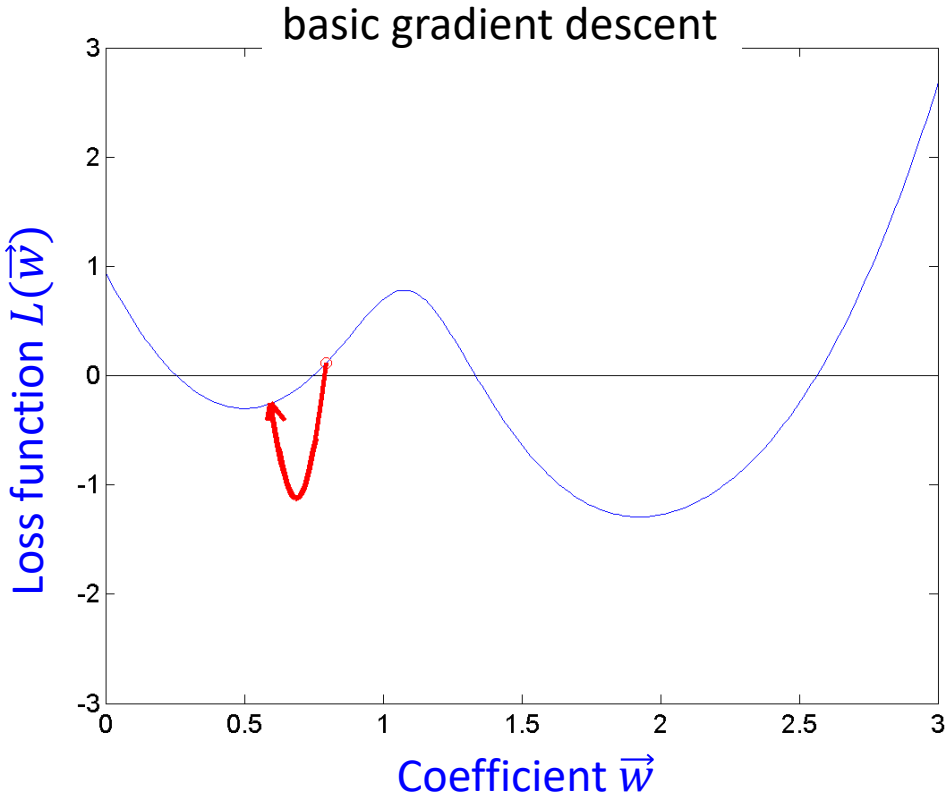
$$L(\vec{w}) = \frac{1}{4}\sum_{i=1}^{n}(y_i - y_i^*)^2$$

The perceptron learning algorithm tries to minimize the loss function using the following strategy:

- If $y_i = y_i^*$ then do nothing.
- If $y_i \neq y_i^*$ then set $\vec{w} = \vec{w} + \eta y_i \vec{x}_i$.

Why is the perceptron so weird?

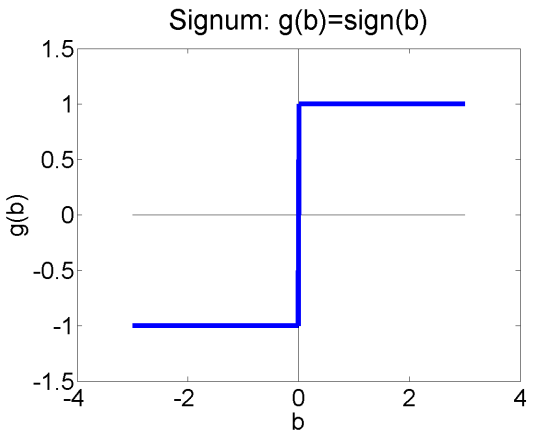
- If $y_i = y_i^*$ then do nothing.
- If $y_i \neq y_i^*$ then set $\vec{w} = \vec{w} + \eta y_i \vec{x}_i$.

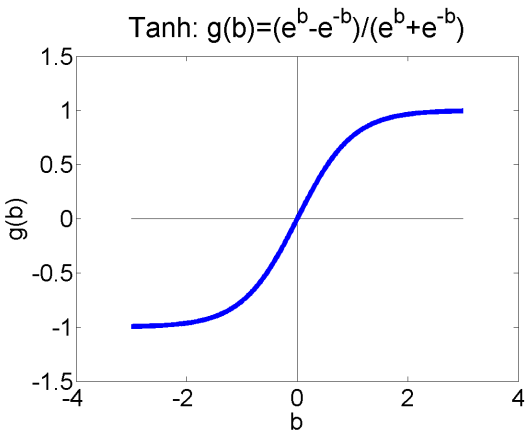


… that seems really weird. Why not just use gradient descent, i.e., why not just set $\vec{w} = \vec{w} - \eta \nabla_{\vec{w}} L$?

Answer: because $y_i^* = \text{sgn}(\vec{w}^T \vec{x}_i)$ is not differentiable.

Fixing the perceptron





Let's make $L(\vec{w})$ differentiable.

First, we make $y^*(\vec{w})$ differentiable.

Instead of $y_i^* = \text{sgn}(\vec{w}^T\vec{x}_i)$

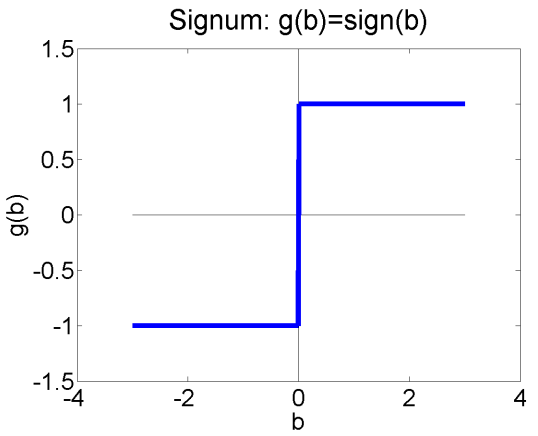
We'll use $y_i^* = \tanh(\vec{w}^T\vec{x}_i)$.

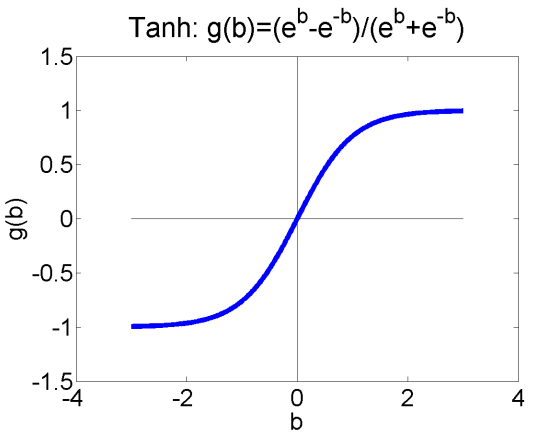
That's pronounced "tanch," it means

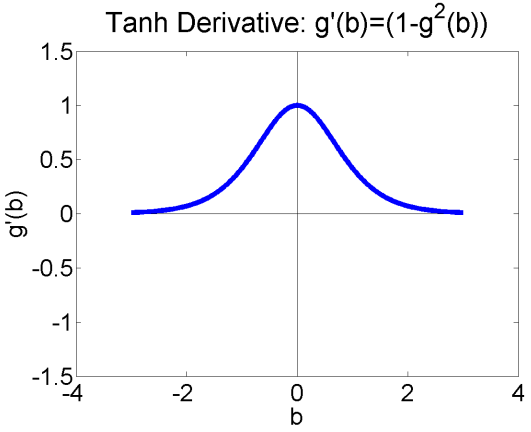
"hyperbolic tangent," and it looks like this:

$$y_i^* = \tanh(\vec{w}^T\vec{x}_i) = \frac{e^{\vec{w}^T\vec{x}} - e^{-\vec{w}^T\vec{x}}}{e^{\vec{w}^T\vec{x}} + e^{-\vec{w}^T\vec{x}}} = \frac{1 - e^{-2\vec{w}^T\vec{x}}}{1 + e^{-2\vec{w}^T\vec{x}}}$$

Fixing the perceptron







Let's make $L(\vec{w})$ differentiable.

First, we make $y^*(\vec{w})$ differentiable.

Instead of $y_i^* = \text{sgn}(\vec{w}^T\vec{x}_i)$

We'll use $y_i^* = \tanh(\vec{w}^T\vec{x}_i)$.

That's pronounced "tanch," it means

"hyperbolic tangent," and it looks like this:

$$y_i^* = \tanh(\vec{w}^T\vec{x}_i) = \frac{e^{\vec{w}^T\vec{x}} - e^{-\vec{w}^T\vec{x}}}{e^{\vec{w}^T\vec{x}} + e^{-\vec{w}^T\vec{x}}} = \frac{1 - e^{-2\vec{w}^T\vec{x}}}{1 + e^{-2\vec{w}^T\vec{x}}}$$

Its derivative is

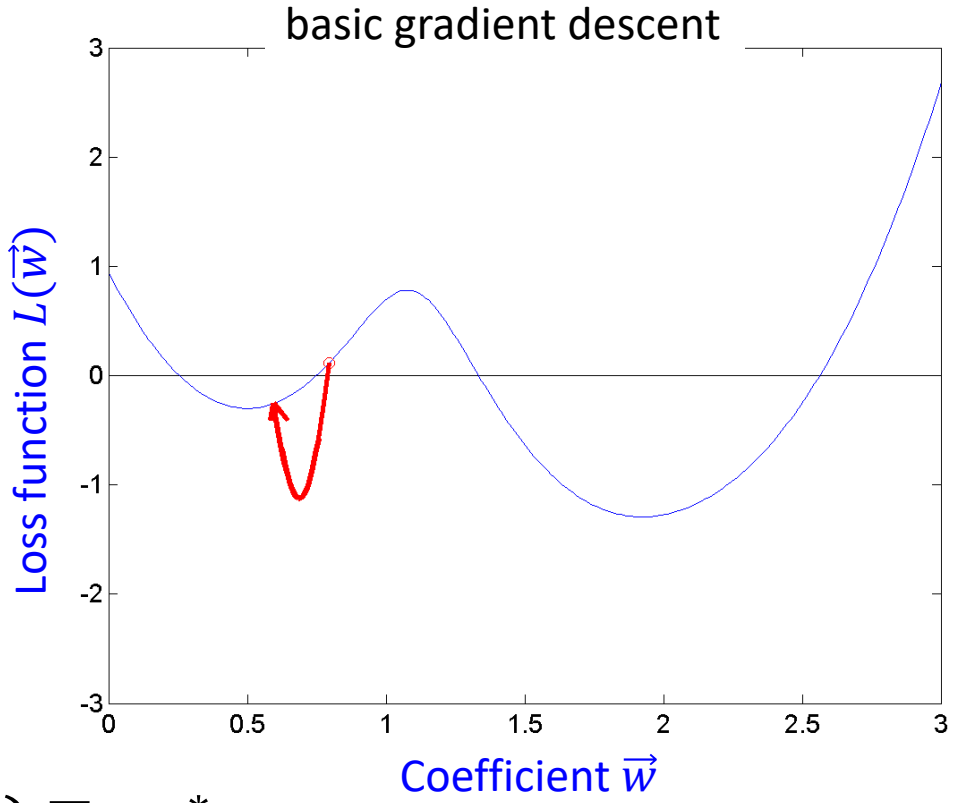
$$\frac{dy_i^*}{d\vec{w}^T\vec{x}_i} = \frac{d\tanh(\vec{w}^T\vec{x}_i)}{d\vec{w}^T\vec{x}_i} = 1 - \tanh^2(\vec{w}^T\vec{x}_i) = 1 - y_i^{*2}$$

Fixing the perceptron

Now, we just differentiate $L(\vec{w})$.

Remember that $L(\vec{w}) = \frac{1}{4}\sum_{i=1}^{n}(y_i - y_i^*)^2$.

Its derivative is:



$$\nabla_{\vec{w}} L = -\frac{1}{2}\sum_{i=1}^{n}(y_i - y_i^*)\,\nabla_{\vec{w}} y_i^*$$

$$= -\frac{1}{2}\sum_{i=1}^{n}(y_i - y_i^*)\left(1 - y_i^{*2}\right)\nabla_{\vec{w}}(\vec{w}^T\vec{x}_i) = -\sum_{i=1}^{n}\left(\frac{y_i - y_i^*}{2}\right)\left(1 - y_i^{*2}\right)\vec{x}_i$$

Comparing logistic regression vs. the perceptron

Logistic regression:

$$\vec{w} = \vec{w} - \eta \nabla_{\vec{w}} L = \vec{w} + \eta \sum_{i=1}^{n} \left(\frac{y_i - y_i^*}{2} \right) \left(1 - y_i^{*2}\right) \vec{x}_i$$

- If $y_i = y_i^*$ then do nothing.
- If $y_i \neq y_i^*$ then set $\vec{w} = \vec{w} + \eta \left(\frac{y_i - y_i^*}{2} \right) \left(1 - y_i^{*2}\right) \vec{x}_i$

Perceptron:

- If $y_i = y_i^*$ then do nothing.
- If $y_i \neq y_i^*$ then set $\vec{w} = \vec{w} + \eta y_i \vec{x}_i$

Conclusions

- Perceptron and Logistic Regression are similar in most ways:
  - They both implement linear classification rules.
  - They can both be initialized either using random weights, or using all zero weights, or setting the weight vector equal to the average of the y=+1 class, or any other reasonable initialization.
  - They can both be trained, one training token at a time. They only change when the classifier output is different from the ground truth label, i.e., $y_i \neq y_i^*$.
  - They both use a "learning rate," η, which should start at $\eta \approx 1$, and should gradually decay toward zero as you see more and more data.
- They differ only in the way the weight vector, w, is updated.
  - Perceptron just adds $\eta y_i \vec{x}_i$.
  - Logistic regression adds $-\eta \nabla_{\vec{w}} L = \eta \left(\frac{y_i - y_i^*}{2}\right)\left(1 - y_i^{*2}\right)\vec{x}_i$.