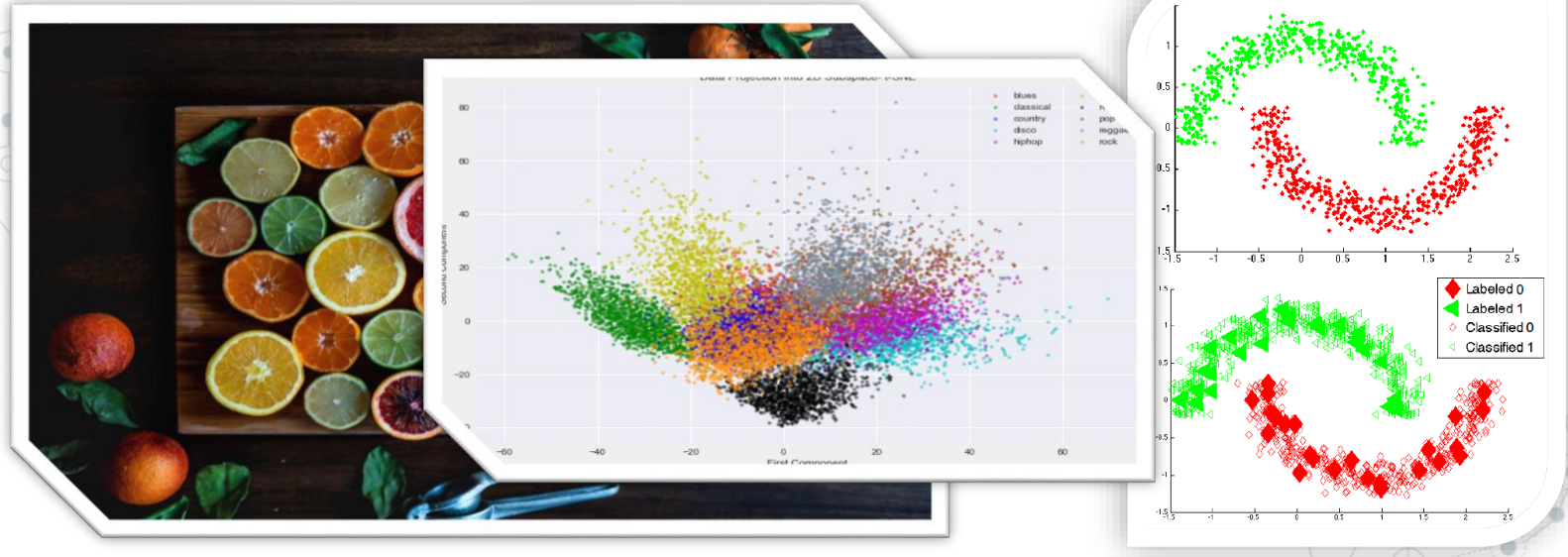
CLASSFICIATION



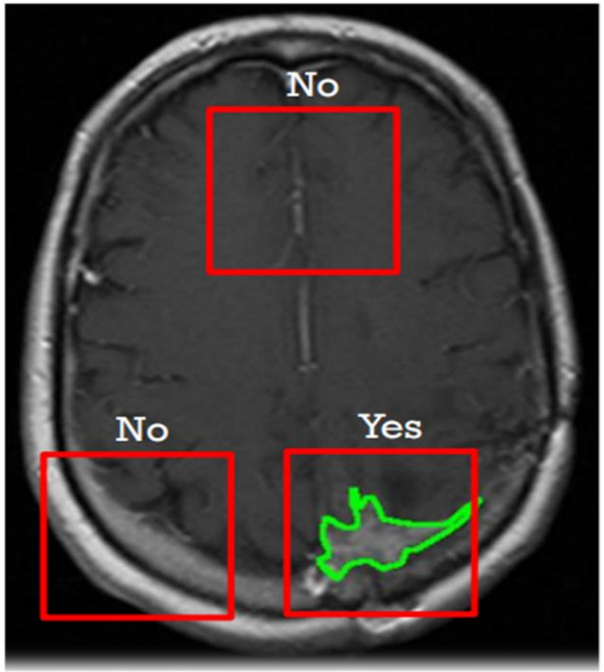
EXAMPLES

TUMOR CLASSIFICATION

Tumor?

Yes ~ +1

No ~ −1



EXAMPLES

SPAM FILTERS

Spam?

Yes ~ +1

No ~ −1

METHODOLOGY

Machine Learning
> Supervised Learning
> Classification

Hypothesis function

- **Task.** Find $h: \mathbb{R}^d \rightarrow \{-1, +1\}$ **such that** $y \approx h(x; \theta)$
- **Experience.** Training data $\left(x^{(1)}, y^{(1)}\right), \ldots, \left(x^{(n)}, y^{(n)}\right)$
- **Performance.** Prediction error on test data

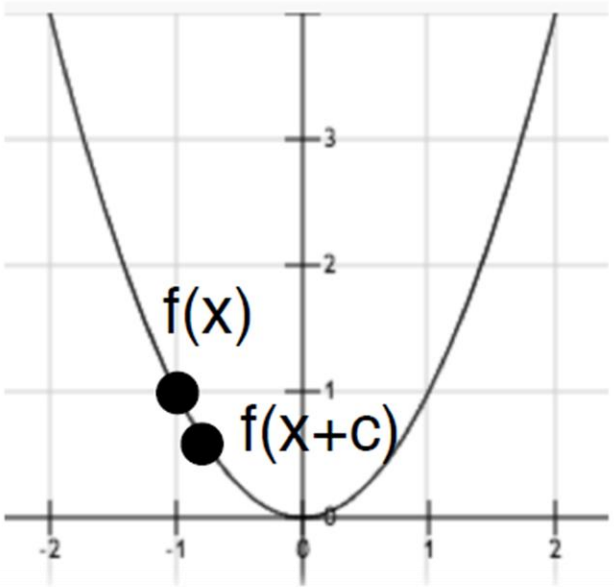
GRADIENT DESCENT

The slope at a point is called the **derivative** at that point

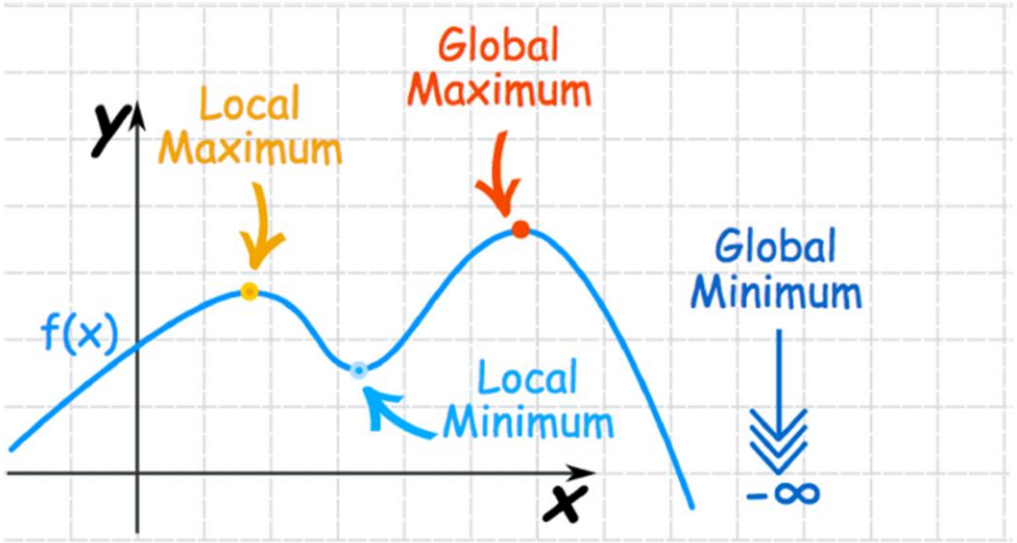
Intuition: Measure the slope between two points that are really close together

$$\frac{f(x + c) - f(x)}{c}$$

Limit as *c* goes to zero



GRADIENT DESCENT



From: https://www.mathsisfun.com/algebra/functions-maxima-minima.html

All global maxima and minima are also local maxima and minima

GRADIENT DESCENT

What if a function has multiple arguments?

Ex: $f(x_1, x_2) = 3x_1 + 5x_2$

$df/dx_1 = 3 + 5x_2$ The derivative "with respect to" x_1

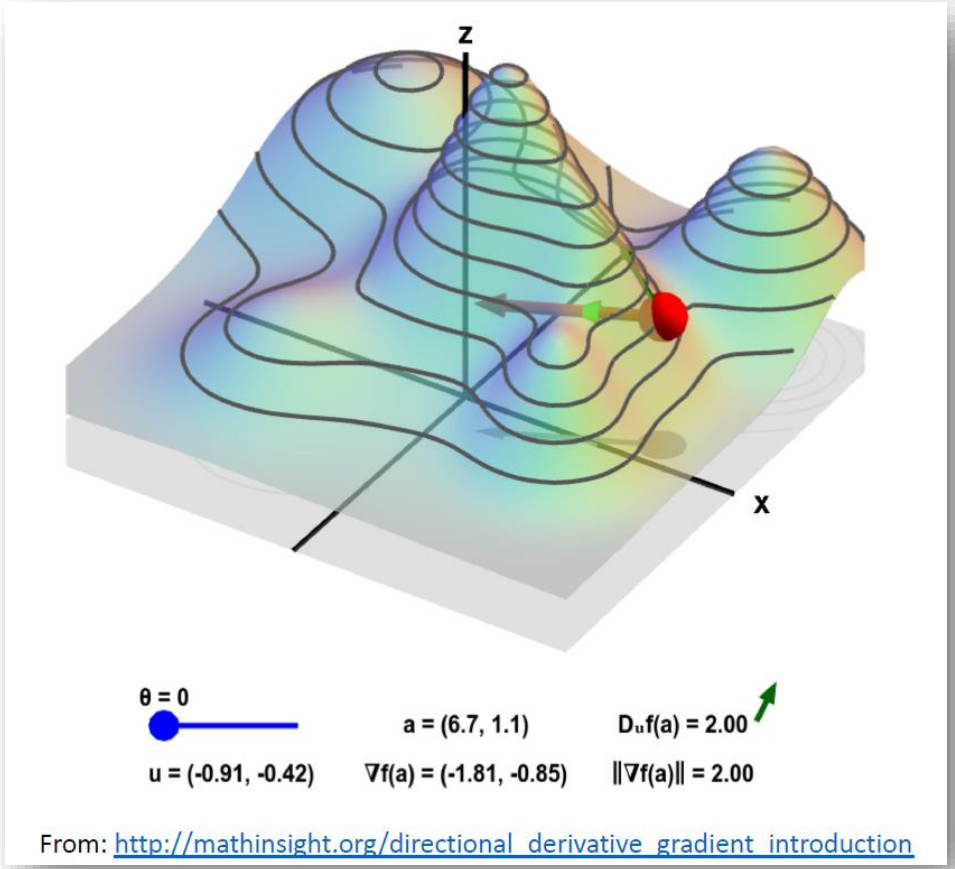
$df/dx_2 = 3x_1 + 5$ The derivative "with respect to" x_2

These two functions are called **partial derivatives**.

The vector of all partial derivatives for a function f is called the **gradient** of the function:

$$\nabla f(x_1, x_2) = < df/dx_1 , df/dx_2 >$$

GRADIENT DESCENT



From: http://mathinsight.org/directional_derivative_gradient_introduction

GRADIENT DESCENT

1. Initialize the parameters $\mathbf{w}$ to some guess (usually all zeros, or random values)
2. Update the parameters:
   $$\mathbf{w} = \mathbf{w} - \eta \nabla L(\mathbf{w})$$
3. Update the learning rate η (How? Later…)
4. Repeat steps 2-3 until $\nabla L(\mathbf{w})$ is close to zero.

GRADIENT DESCENT

Gradient descent is guaranteed to eventually find a *local* minimum if:

- the learning rate is decreased appropriately;
- a finite local minimum exists (i.e., the function doesn't keep decreasing forever).

What if we want to find a local *maximum*?

Same idea, but the update rule moves the parameters in the opposite direction:

$$\mathbf{w} = \mathbf{w} + \eta \nabla L(\mathbf{w})$$

GRADIENT DESCENT

In order to guarantee that the algorithm will converge, the learning rate should decrease over time. Here is a general formula.

At iteration t:

$\eta_t = c_1 / (t^a + c_2)$,

where $0.5 < a < 2$

$c1 > 0$

$c2 \geq 0$

GRADIENT DESCENT

For most functions, you probably won't get the gradient to be exactly equal to **0** in a reasonable amount of time.

Once the gradient is sufficiently close to **0**, stop trying to minimize further.

LINEAR CLASSFIERS

METHODOLOGY

Training data

$$\mathcal{S}_n = \left\{ \left(x^{(i)}, y^{(i)} \right) \middle| \, i = 1, \ldots, n \right\}$$

- Features/Inputs $x^{(i)} = \left(x_1^{(i)}, \ldots, x_d^{(i)} \right)^\top \in \mathbb{R}^d$
- Labels/Output $y^{(i)} \in \{-1, +1\}$

MODEL

Model

Set of linear classifiers $h: \mathbb{R}^d \to \{-1, +1\}$

$$h(x; \theta, \theta_0) = \text{sign}(\theta_d x_d + \cdots + \theta_1 x_1 + \theta_0)$$

$$= \text{sign}(\theta^\top x + \theta_0)$$

$$\text{sign}(z) = \begin{cases} +1 & \text{if } z \geq 0, \\ -1 & \text{if } z < 0. \end{cases}$$

Some folks define $\text{sign}(0) = 0$ *but we will not adopt that here.*

Model Parameters

$$\theta \in \mathbb{R}^d, \theta_0 \in \mathbb{R}$$

Also called the offset

TEST LOSS

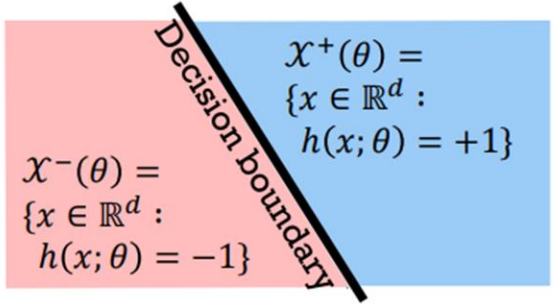
$[\![\cdot]\!]$ *is the indicator function that returns a* 1 *if its argument is true, and* 0 *otherwise.*

Test Loss

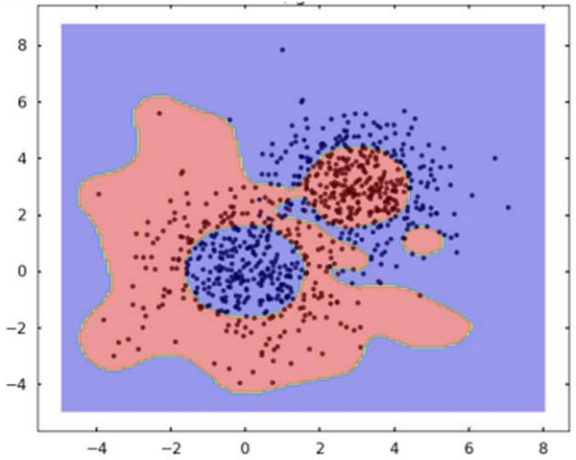
$$\mathcal{R}_1(\theta, \theta_0; x, y) = [\![y \neq h(x; \theta, \theta_0)]\!]$$

$$\mathcal{R}(\theta, \theta_0; \mathcal{S}_*) = \frac{1}{n} \sum_{(x,y) \in \mathcal{S}_*} \mathcal{R}_1(\theta, \theta_0; x, y)$$

DECISION REGIONS



linear classifier



non-linear classifier

A classifier h partitions the space into decision regions that are separated by decision boundaries. In each region, all the points map to the same label. Many regions could have the same label.

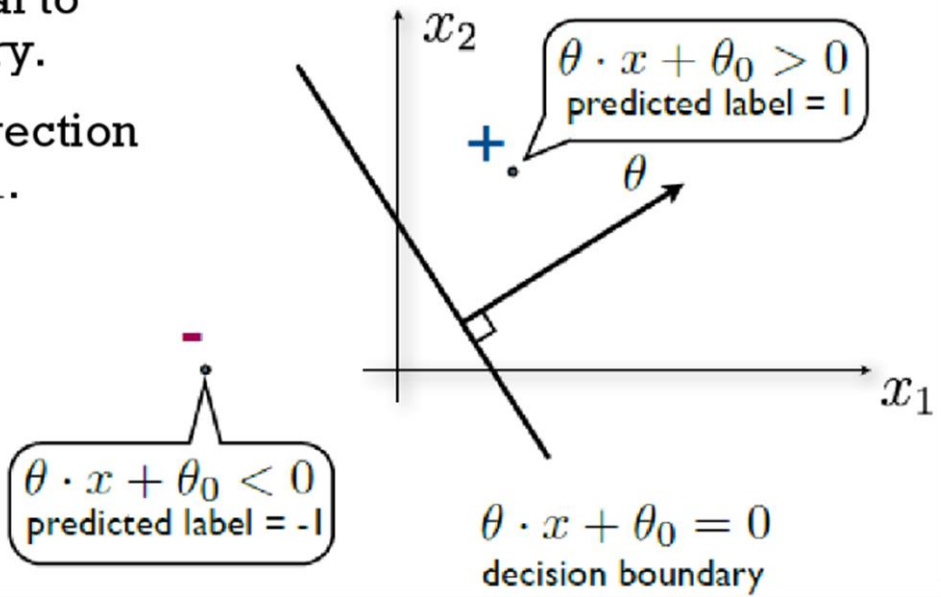
For linear classifiers, these regions are half spaces.

DECISION REGIONS

For linear classifiers, the decision boundary is a hyperplane of dimension $d - 1$.

Vector θ is orthogonal to the decision boundary.

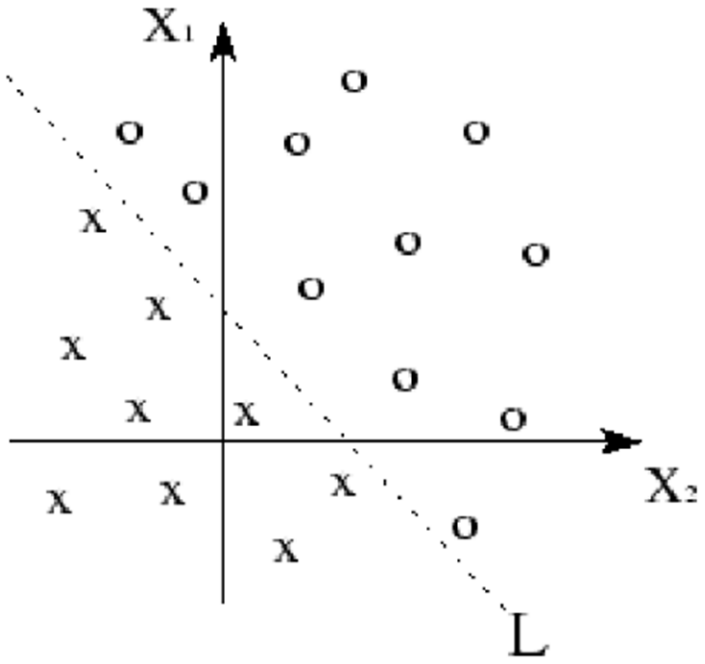
Vector θ points in direction of region labelled +1.



LINEARLY SEPARABLE

The training data $\mathcal{S}_n$ is linearly separable if there exists a parameters θ and θ_0 such that for all $(x, y) \in \mathcal{S}_n$,

$$y(\theta^\top x + \theta_0) > 0.$$



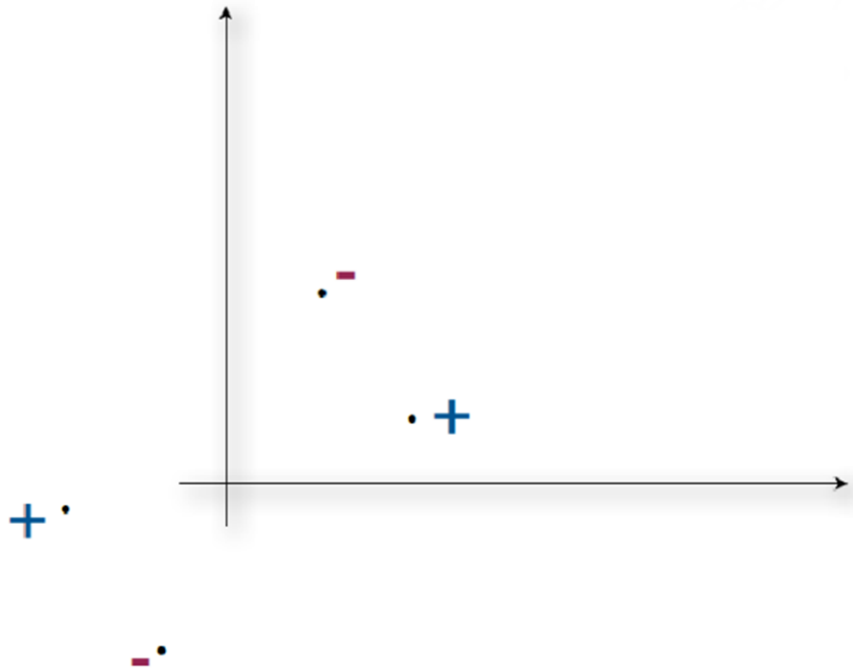
NOT LINEARLY SEPARABLE

Challenge

How do you prove the training data on the right is not linearly separable?

Hint

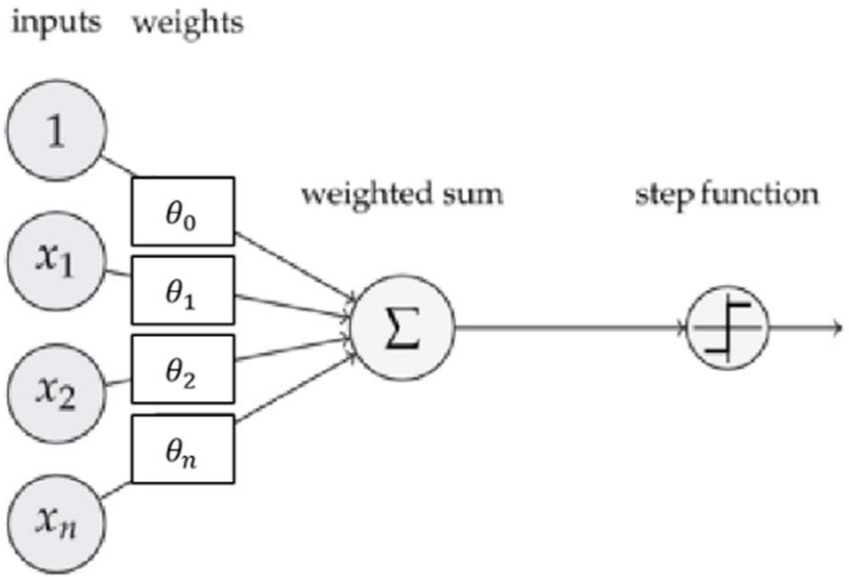
Draw a line between the points labelled +1, and a line between the points labelled −1.



PERCEPTRON ALGORITHM

PERCEPTRON

Linear classifiers are often also called perceptrons.



Perceptrons (1957) were designed to resemble neurons.

ZERO-ONE LOSS

Let $\mathcal{L}_1(\theta; x, y) = 1$ *(*0 *otherwise) if*

- $y \neq h(x; \theta)$, *or* *[misclassified]*
- (x, y) *is on decision boundary* *[boundary]*

Note that $y(\theta^\top x) \leq 0$ *if*

- $\theta^\top x$ *and* y *differ in sign, or* *[misclassified]*
- $\theta^\top x$ *is zero* *[boundary]*

$$\mathcal{L}_1(\theta; x, y) = [\![y(\theta^\top x) \leq 0]\!] = \text{Loss}\big(y(\theta^\top x)\big)$$

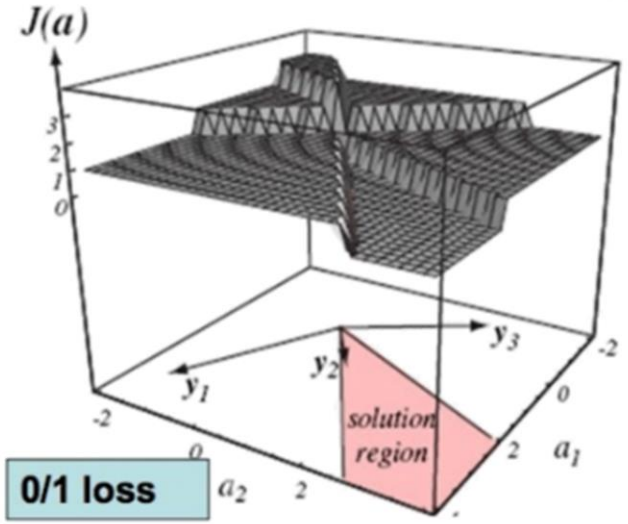
where $\text{Loss}(z) = [\![z \leq 0]\!]$ *is the zero-one loss.*

TRAINING LOSS

$$\text{Loss}(z) = [\![z \leq 0]\!]$$

$$\mathcal{L}_1(\theta; x, y) = \text{Loss}\big(y(\theta^\top x)\big)$$

$$\mathcal{L}_n(\theta; \mathcal{S}_n) = \frac{1}{n}\sum_{(x,y)\in\mathcal{S}_n} \mathcal{L}_1(\theta; x, y)$$



Gradient is zero almost everywhere!

Gradient descent not possible.

HYPOTHESIS FUNCTION

The hypothesis function is given by

$$h_\theta\left(x^{(i)}\right) = \operatorname{sgn}\left(\left\langle \theta, x^{(i)} \right\rangle\right),$$

where

$$\operatorname{sgn}(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{if } z < 0. \end{cases}$$

LOSS FUNCTION

- Using the count of the number of misclassified points as a loss function is not good as this is a piecewise constant, which means that its gradient is zero almost everywhere and gradient descent methods will not work.
- Instead we define the perceptron criterion:

$$\mathcal{L}_P(\theta) = -\sum_{i \in \mathcal{M}} y^{(i)} \left\langle \theta, x^{(i)} \right\rangle,$$

where $\mathcal{M}$ is the set of misclassified points.

PERCEPTRON ALGORITHM

- Step 1: pick a point $x^{(i)}$ and check if $y^{(i)} \langle \theta(t), x^{(i)} \rangle \geq 0$.
- Step 2: if yes, do nothing; if no perform the following update rule:

$$\begin{aligned} \theta(t+1) &= \theta(t) - \alpha \nabla \mathcal{L}_P(\theta(t)) \\ &= \theta(t) + \alpha x^{(i)} y^{(i)} \end{aligned}$$

- Cycle through the rest of the data with steps 1 and 2.

ERROR REDUCTION

- The update rule reduces the error with respect to the selected point because

$$-y^{(i)}\left\langle \theta(t+1), x^{(i)}\right\rangle = -y^{(i)}\left\langle \theta(t), x^{(i)}\right\rangle - y^{(i)}\left\langle \alpha x^{(i)} y^{(i)}, x^{(i)}\right\rangle$$
$$< -y^{(i)}\left\langle \theta(t), x^{(i)}\right\rangle$$

since $\alpha \left\| y^{(i)} x^{(i)} \right\|^2 > 0$.

- However, this does not guarantee that the total error function is reduced at each stage as:
  - the contribution to the error from other misclassified points may have increased;
  - previously correctly classified points may have become misclassified.

PERCEPTRON SUMMARY

MISTAKE-DRIVEN ALGORITHM

1. *Initialize* $\theta = 0$.
2. *For each data* $(x, y) \in \mathcal{S}_n$,
   a. *Check if* $h(x; \theta) = y$.
   b. *If not, update* θ *to improve the mistake.*
3. *Repeat Step (2) until no mistakes are found.*

CONVERGENCE

Theorem. If the training data is linearly separable, then the perceptron algorithm terminates after a finite number of steps.

Non linearly-separable. In this case, the perceptron algorithm will never terminate, because there will always be a mistake for all values of θ. Other learning algorithms are needed.

EXAMPLE

Training data

- $(x^{(1)}, y^{(1)}) = ((2,\ \ 2)^\top, +1)$
- $(x^{(2)}, y^{(2)}) = ((2, -1)^\top, -1)$

Apply the perceptron algorithm to the data to find a classifier $h(x;\theta), \theta = (\theta_1, \theta_2)$, that separates the data.

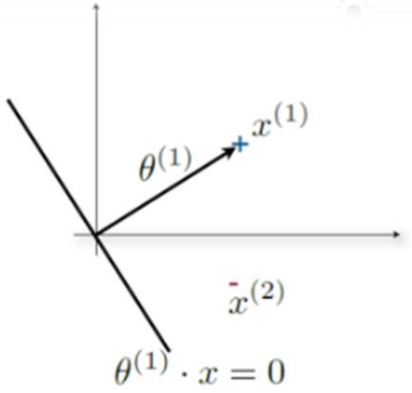
EXAMPLE

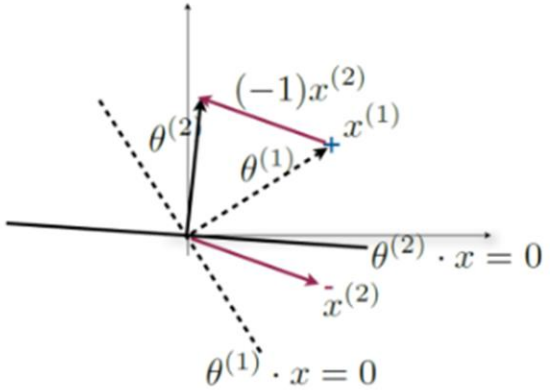
EXAMPLE

Training data

- $(x^{(1)}, y^{(1)}) = ((2, \ 2)^\top, +1)$
- $(x^{(2)}, y^{(2)}) = ((2, -1)^\top, -1)$

- *Initialize* $\theta = (0,0)$.
- *Since* $y^{(1)}\theta^\top x^{(1)} = 0$, *set* $\theta = (0,0) + (2,2)^\top = (2,2)^\top$.
- *Since* $y^{(2)}\theta^\top x^{(2)} = -2$, *set* $\theta = (2,2)^\top - (2,-1)^\top = (0,3)^\top$.
- $y^{(1)}\theta^\top x^{(1)} = 6 > 0$.
- $y^{(2)}\theta^\top x^{(2)} = 3 > 0$.
- *No more mistakes, so we are done.*





EXAMPLE

Example

Red circles: +1, Blue circles: -1 (see pg 195 in Bishop)

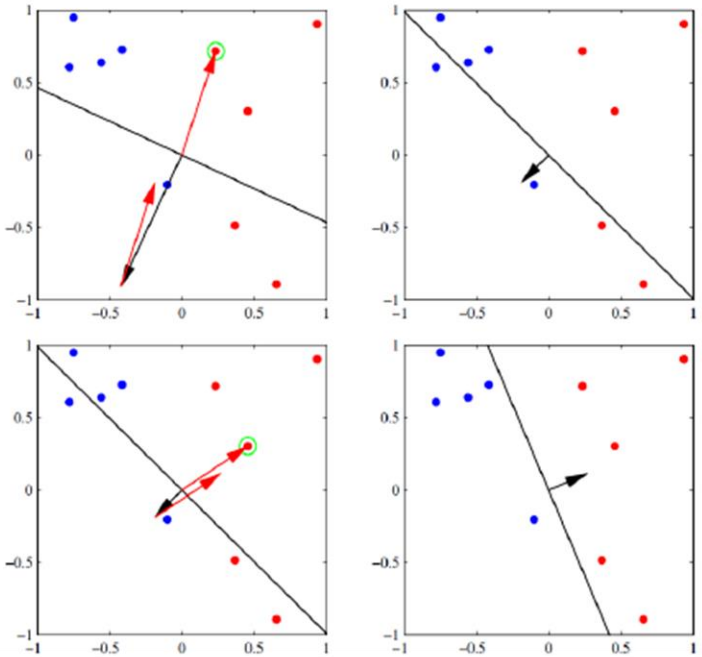


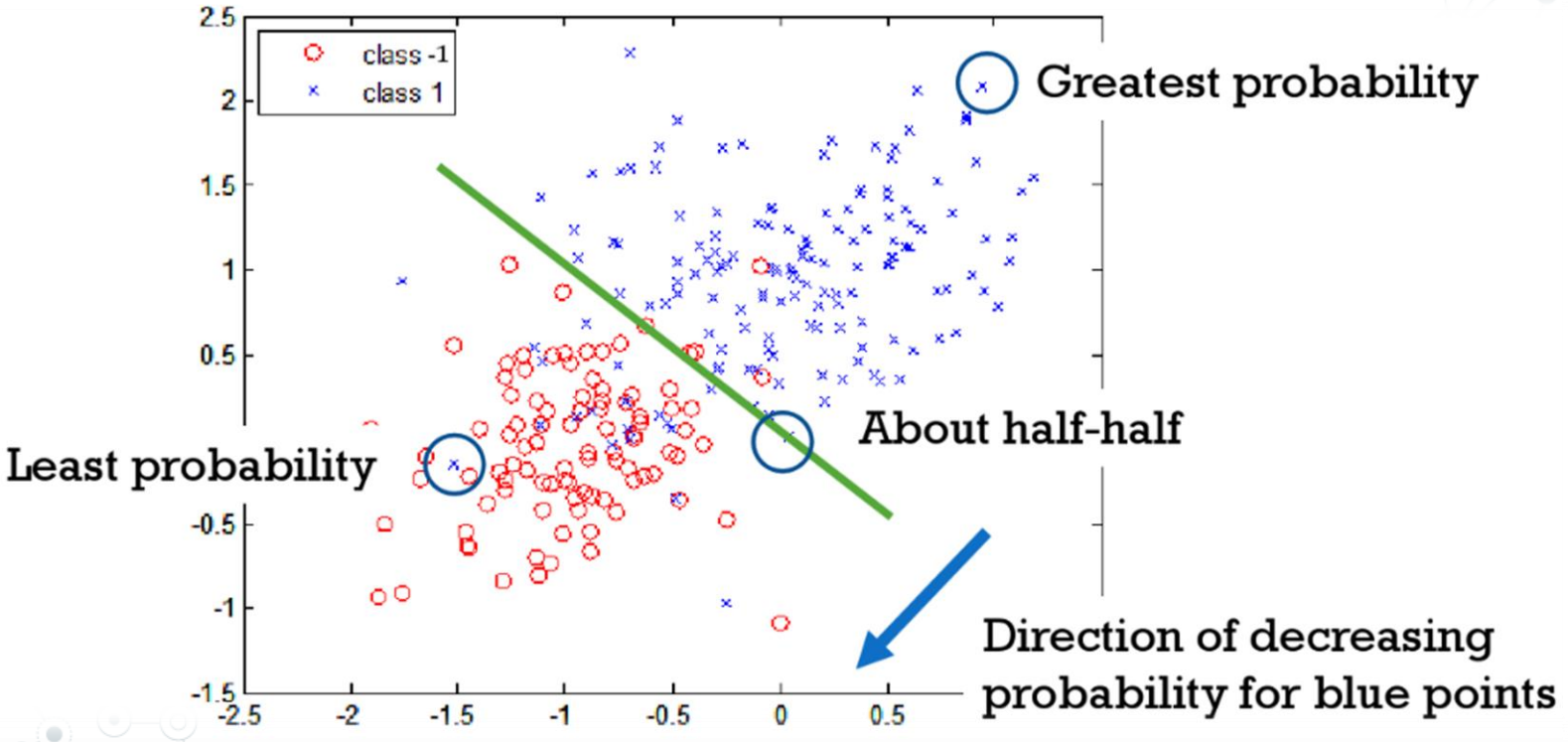
Figure 4.7 Illustration of the convergence of the perceptron learning algorithm, showing data points from two classes (red and blue) in a two-dimensional feature space (ϕ_1, ϕ_2). The top left plot shows the initial parameter vector $\mathbf{w}$ shown as a black arrow together with the corresponding decision boundary (black line), in which the arrow points towards the decision region which classified as belonging to the red class. The data point circled in green is misclassified and so its feature vector is added to the current weight vector, giving the new decision boundary shown in the top right plot. The bottom left plot shows the next misclassified point to be considered, indicated by the green circle, and its feature vector is again added to the weight vector giving the decision boundary shown in the bottom right plot for which all data points are correctly classified.

DISADVANTAGES

- Learning algorithm difficulties:
  - Not easy to differentiate slow convergence from cases where there will be no convergence due to not having linear separability.
  - Different initialization of the parameters and presentation of the data lead to different solutions.
- Does not provide probabilistic outputs.
- Does not generalize well to more than two classes.

LOGISTIC REGRESSION

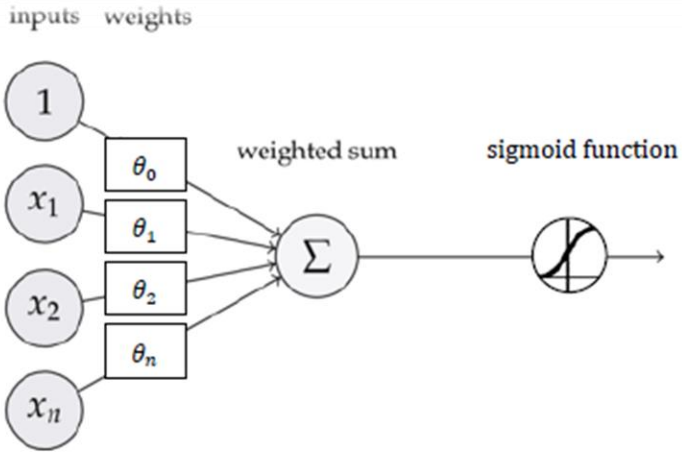
ALMOST LINEARLY SEPARABLE

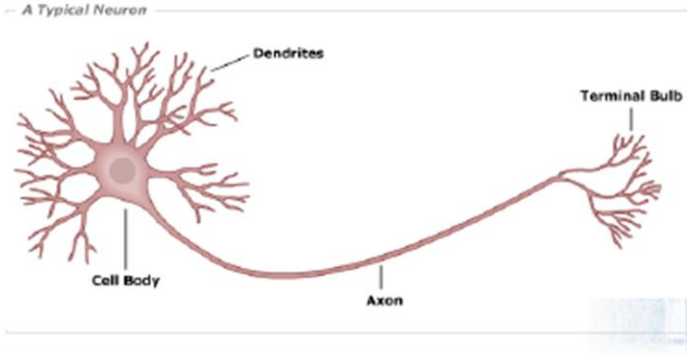


SIGMOID NEURON

Model consists of sigmoid neurons.

They were popular in the early days of deep learning (2006).





PROBABILISTIC MODEL

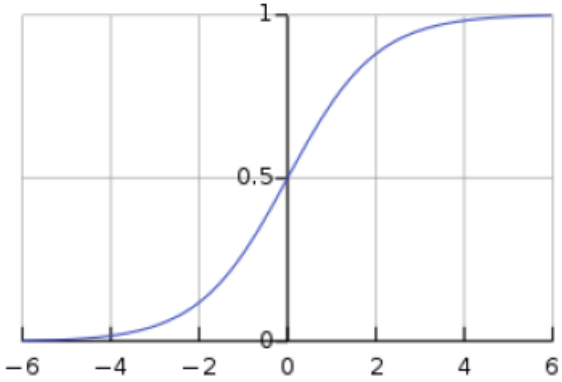
$[0, 1]$ denotes the interval $\{a \in \mathbb{R}: 0 \leq a \leq 1\}$

Model the probability that the label y is $+1$ given the feature is x.

$$h: \mathbb{R}^d \to [0, 1]$$

$$h(x; \theta) = \mathbb{P}(\, y = +1 \mid x \,) = \text{sigmoid}(\theta^\top x)$$

For small $\theta^\top x$, the label is very likely to be -1.



For large $\theta^\top x$, the label is very likely to be $+1$.

SIGMOID FUNCTION

- The formula for the sigmoid function $\sigma(z)$ is given by

$$\sigma(z) = \frac{1}{1 + e^{-z}},$$

and the hypothesis function is thus

$$h_\theta\left(x^{(i)}\right) = \sigma\left(\left\langle \theta, x^{(i)} \right\rangle\right) = \frac{1}{1 + e^{-\left\langle \theta, x^{(i)} \right\rangle}}.$$

- The sigmoid function is also known as the logistic function.

SOLVING FOR HYPERPLANE

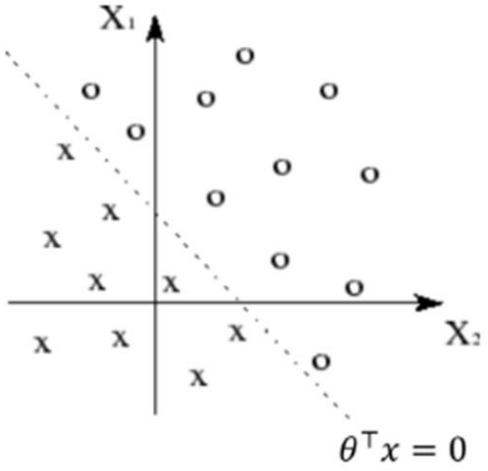
- Because of its probabilistic nature, solving for the best fit hyperplane requires us to maximize a likelihood function based on the sigmoid hypothesis function.

- Involves the use of gradient ascent to maximize the likelihood function.

DECISION BOUNDARY

$$h(x;\theta) \geq \frac{1}{2} \iff \text{sigmoid}(\theta^\top x) \geq \frac{1}{2} \iff \theta^\top x \geq 0$$

$$h(x;\theta) < \frac{1}{2} \iff \text{sigmoid}(\theta^\top x) < \frac{1}{2} \iff \theta^\top x < 0$$



The decision boundary is described by $\theta^\top x = 0$.

SIGMOID FUNCTION FORMULAS

(i) $$\sigma(z) = \frac{1}{1+e^{-z}} = \frac{e^{z}}{e^{z}+1} = 1 - \sigma(-z)$$

(ii) $$\sigma'(z) = \frac{e^{-z}}{(1+e^{-z})^2}$$

$$= \left(\frac{1}{1+e^{-z}}\right)\left(\frac{e^{-z}}{1+e^{-z}}\right)$$

$$= \left(\frac{1}{1+e^{-z}}\right)\left(1 - \frac{1}{1+e^{-z}}\right)$$

$$= \sigma(z)\,(1-\sigma(z)) = \sigma(z)\sigma(-z)$$

SIGMOID FUNCTION FORMULAS

- With $z^{(i)}$ denoting $\langle \theta, x^{(i)} \rangle$, we have

$$p_\theta \left(y = 1 \middle| x^{(i)} \right) = \sigma \left(z^{(i)} \right),$$

which means that

$$p_\theta \left(y = 0 \middle| x^{(i)} \right) = 1 - \sigma \left(z^{(i)} \right) = \sigma \left(-z^{(i)} \right).$$

- We can combine both expressions as

$$p_\theta \left(y = y^{(i)} \middle| x^{(i)} \right) = \sigma \left(z^{(i)} \right)^{y^{(i)}} \sigma \left(-z^{(i)} \right)^{1-y^{(i)}}.$$

LOG-LIKELIHOOD FUNCTION

The log-likelihood function is thus

$$\begin{aligned}\ell(\theta) &= \log \prod_{i=1}^{m} p\left(y = y^{(i)} \,\middle|\, x^{(i)}\right) \\ &= \log \prod_{i=1}^{m} \sigma\left(z^{(i)}\right)^{y^{(i)}} \sigma\left(-z^{(i)}\right)^{1-y^{(i)}} \\ &= \sum_{i=1}^{m} y^{(i)} \log\left(\sigma\left(z^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \log\left(\sigma\left(-z^{(i)}\right)\right).\end{aligned}$$

GRADIENT

The gradient of this loss function is

$$\begin{aligned}
\frac{\partial \ell(\theta)}{\partial \theta_j} &= \sum_{i=1}^{m} \frac{y^{(i)} x_j^{(i)}}{\sigma\left(z^{(i)}\right)} \sigma\left(z^{(i)}\right) \sigma\left(-z^{(i)}\right) - \frac{\left(1-y^{(i)}\right) x_j^{(i)}}{\sigma\left(-z^{(i)}\right)} \sigma\left(z^{(i)}\right) \sigma\left(-z^{(i)}\right) \\
&= \sum_{i=1}^{m} y^{(i)} x_j^{(i)} \sigma\left(-z^{(i)}\right) - \left(1-y^{(i)}\right) x_j^{(i)} \sigma\left(z^{(i)}\right) \\
&= \sum_{i=1}^{m} x_j^{(i)} \left[y^{(i)} \left(1-\sigma\left(z^{(i)}\right)\right) - \left(1-y^{(i)}\right) \sigma\left(z^{(i)}\right) \right] \\
&= \sum_{i=1}^{m} x_j^{(i)} \left(y^{(i)} - \sigma\left(z^{(i)}\right) \right).
\end{aligned}$$

GRADIENT ASCENT

- We can then use the gradient to perform gradient ascent to maximize the likelihood:

$$\theta_j(t+1) = \theta_j(t) + \alpha \sum_{i=1}^{m} \left(y^{(i)} - \sigma \left(\left\langle \theta(t), x^{(i)} \right\rangle \right) \right) x_j^{(i)}, \quad j = 1, \ldots, n.$$

MULTICLASS CLASSIFICATION

Example. Predict color preference (e.g. yellow, green, blue).

Solution.
Learn a 'one-vs-rest' function for each class.

$$h_{\text{yellow}}(x) = \text{sigmoid}(\alpha^\top x)$$

$$h_{\text{green}}(x) = \text{sigmoid}(\beta^\top x)$$

$$h_{\text{blue}}(x) = \text{sigmoid}(\gamma^\top x)$$

Rank the function values to predict the best class.