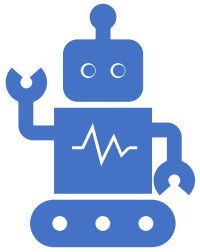
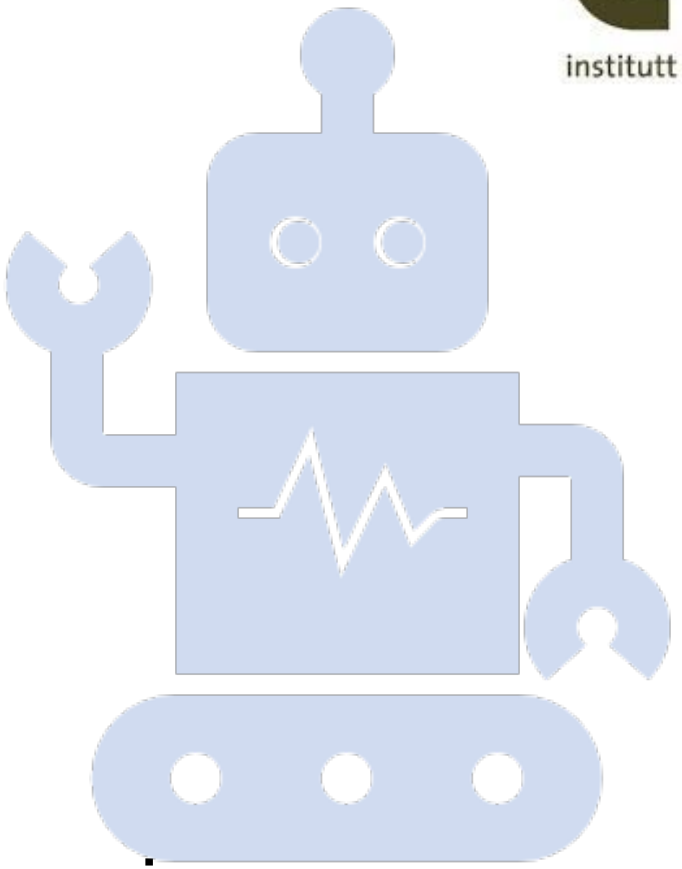


UiO : University of Oslo

institutt for informatikk

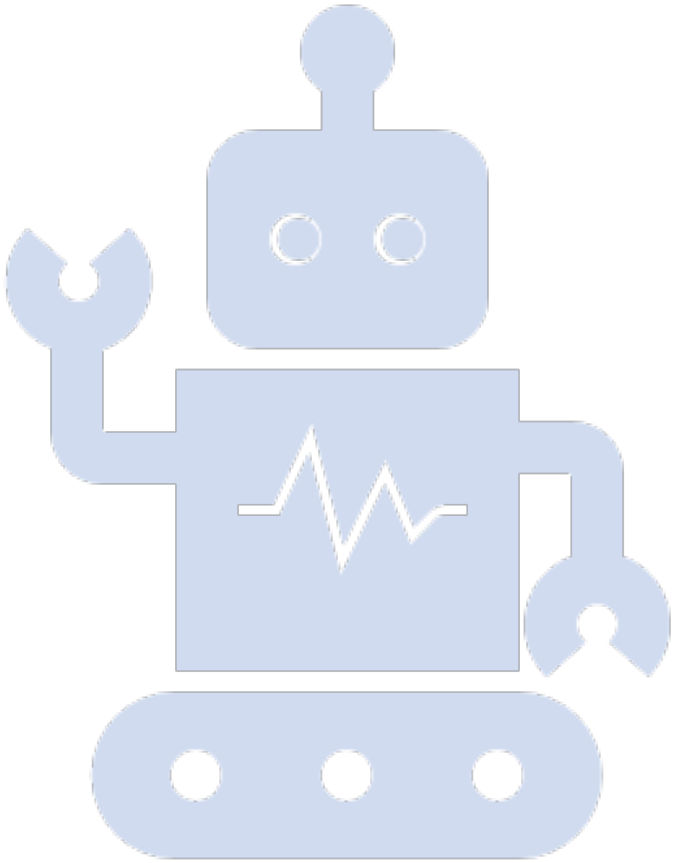


IN3050/IN4050 - Introduction to Artificial Intelligence and Machine Learning



Lecture 5: Perceptron and Linear Regression

Slides by Jan Tore Lønning with revisions by Andrey Kutuzov

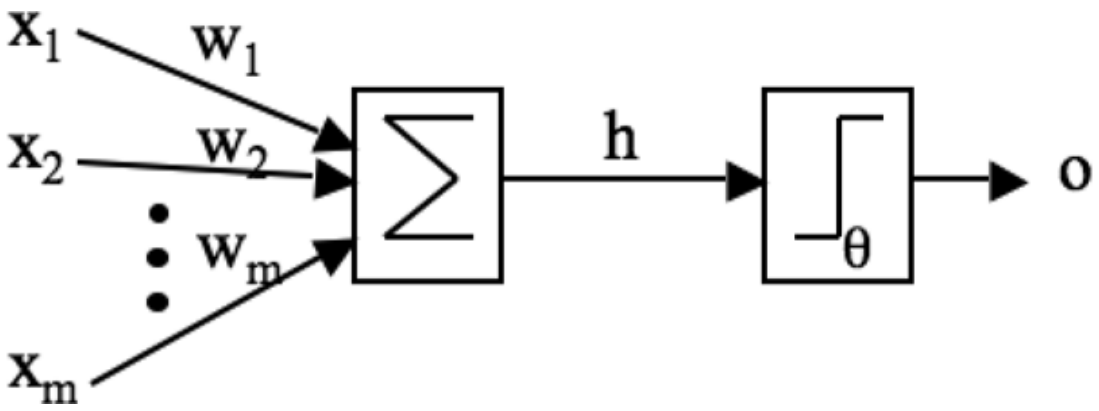


5.1 Overview

IN3050/IN4050 Introduction to Artificial Intelligence and Machine Learning

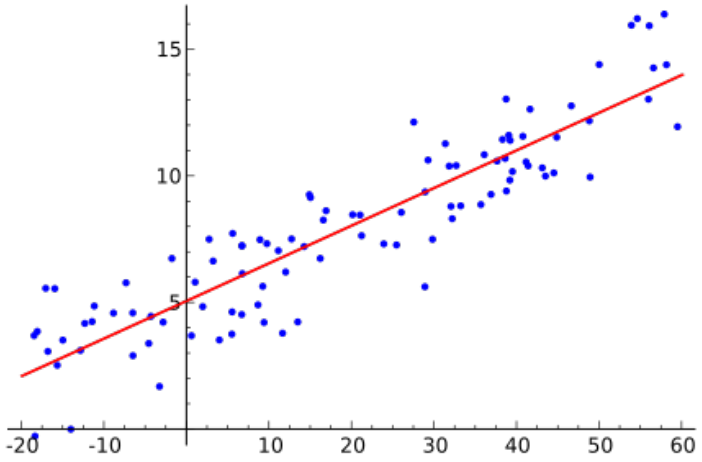
This week: Two main themes

Perceptron (classifier)



Linear Regression

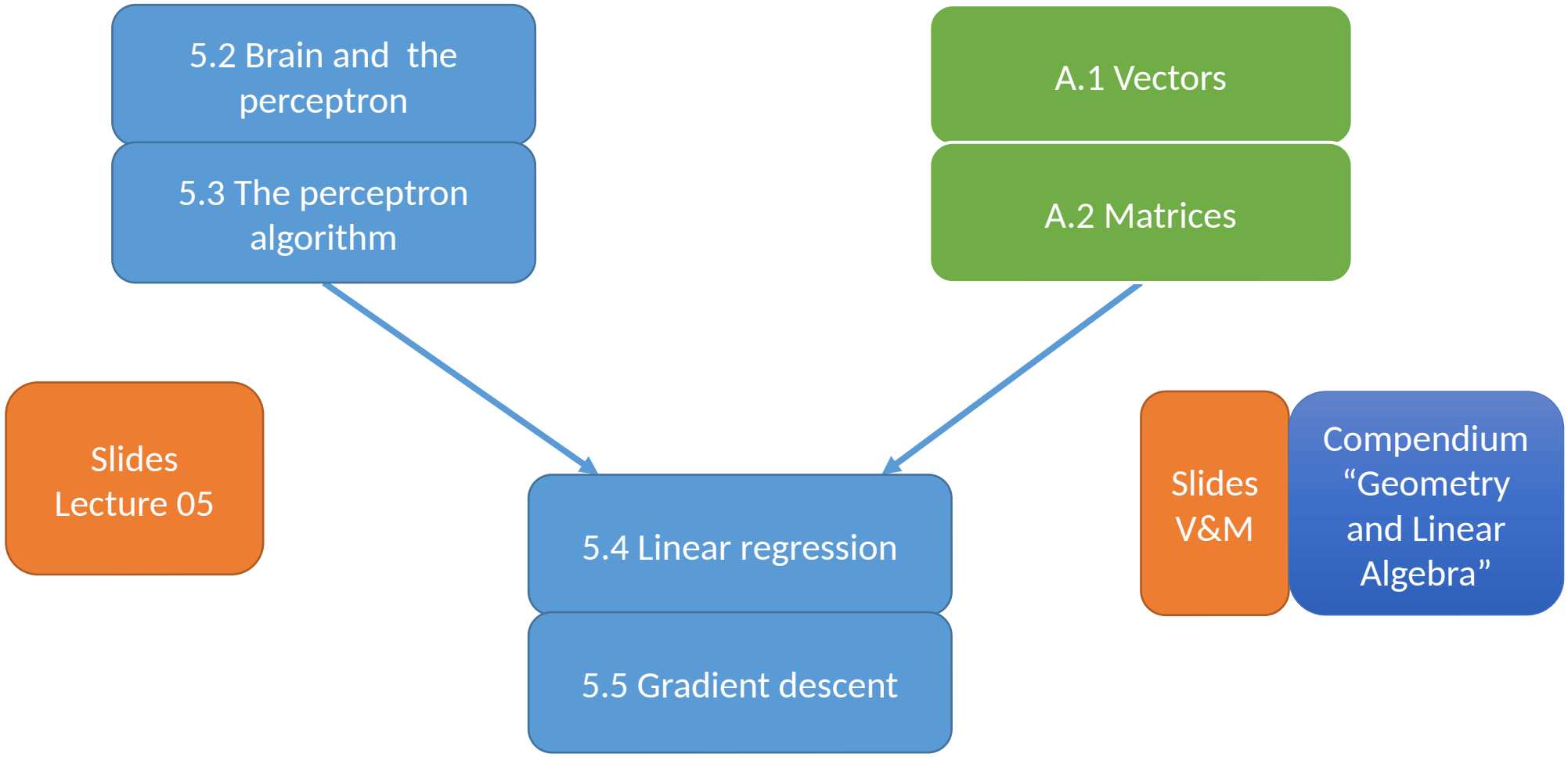
- Assign a numerical value to an observation

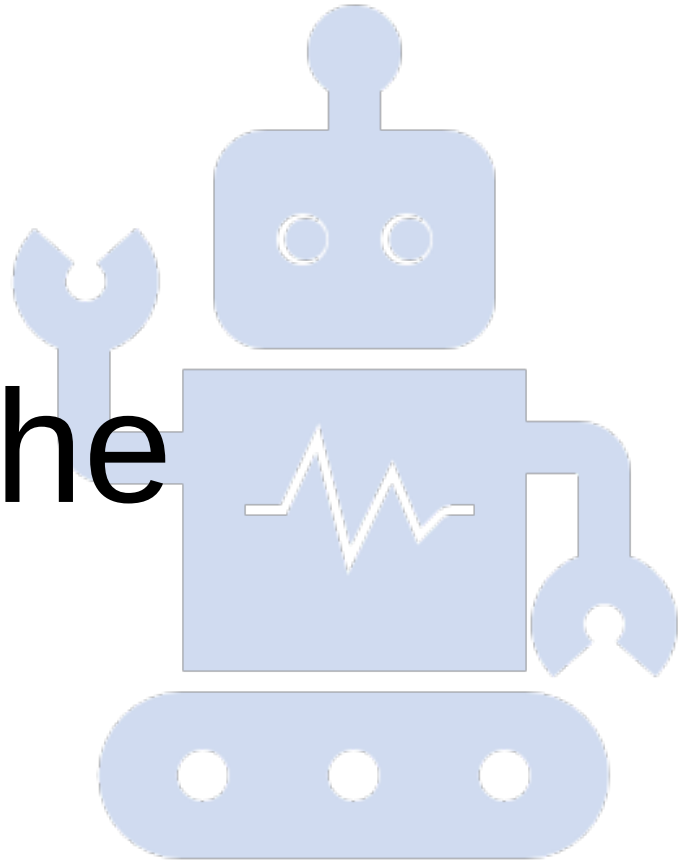


In addition: Vectors, matrices, NumPy

- Efficient code: both writing and execution
  - A@B can replace three nested loops
  - GPUs – parallel processing
- NumPy:
  - Based on vectors and matrices
  - Used by Marsland
  - Libraries for ML, including Deep Learning
- Necessary for a deeper understanding
  - of complex neural networks
    - Tensor generalizes vectors and matrices

Overview


5.2 Brain and the perceptron
5.3 The perceptron algorithm
A.1 Vectors
A.2 Matrices
Slides Lecture 05
5.4 Linear regression
5.5 Gradient descent
Slides V&M
Compendium "Geometry and Linear Algebra"



5.2 The Brain and the Perceptron

IN3050/IN4050 Introduction to Artificial Intelligence and Machine Learning

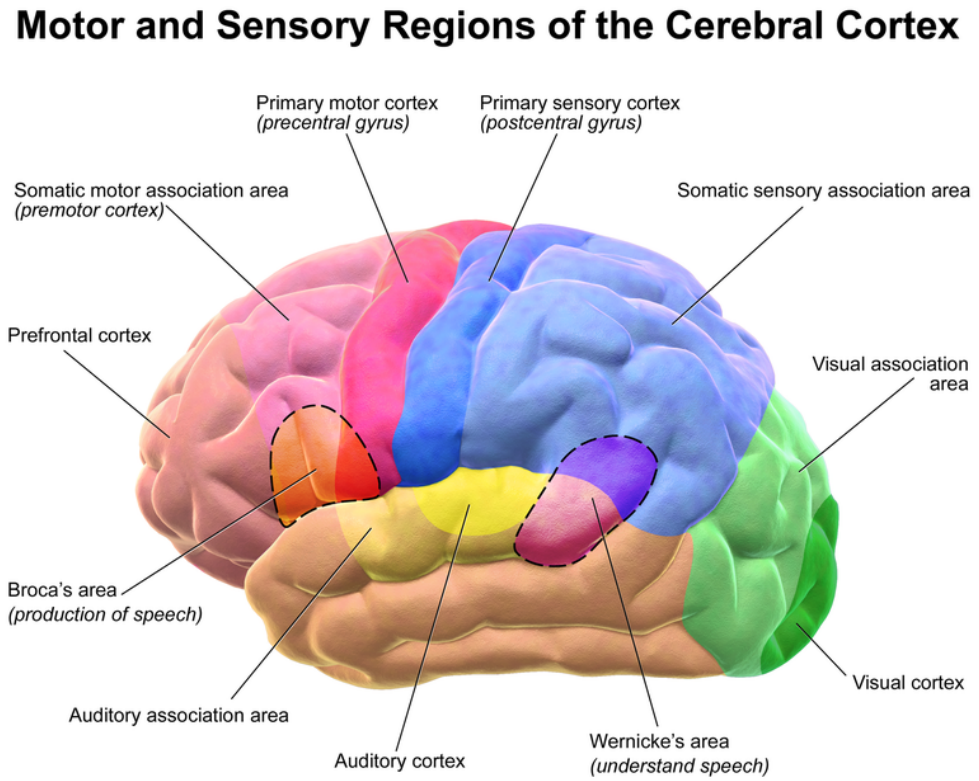
Inspiration for AI and ML

- Psychology
  - Ask people how they think
  - Observe how humans behave
- Logic
  - How should you think
- ‘’Hardware’’:
  - ‘’If we want to make a machine replicating humans, it should be built on similar hardware’’
- … and more …

The human brain

Rough figures

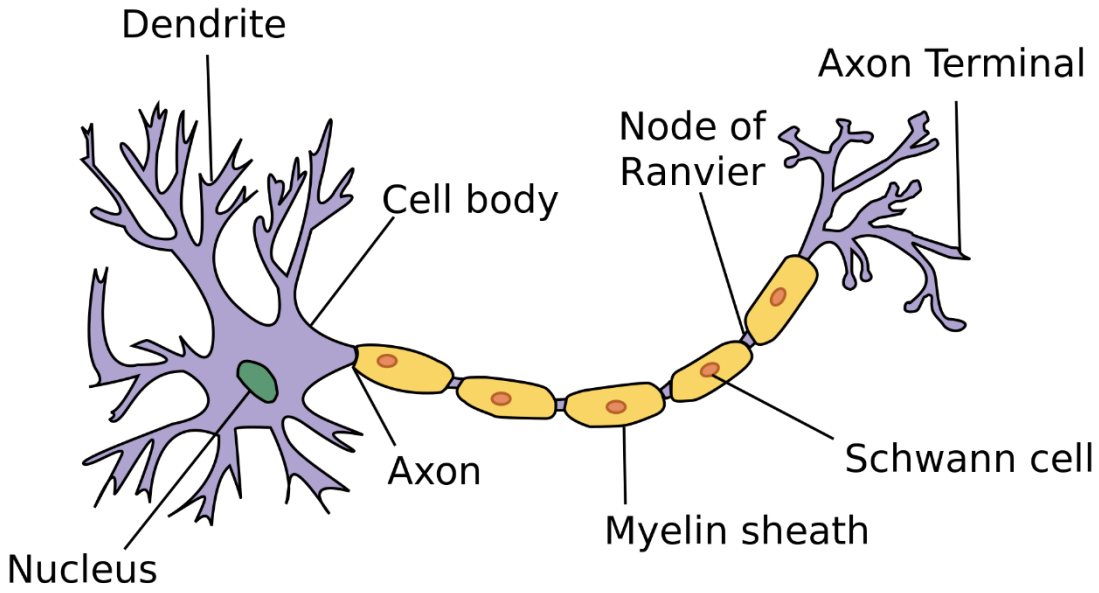
- 1.5 kilos
- 10^{11} Neurons
  - cells
- 10^{14} Synapses
  - connections between neurons
- Clock time: 10^{-3} seconds
  - Compared to computer: $1\ \text{GHz} = 10^{-9}$ seconds



"Medical gallery of Blausen Medical 2014". WikiJournal of Medicine 1 (2). DOI:10.15347/wjm/2014.010. ISSN 2002-4436.

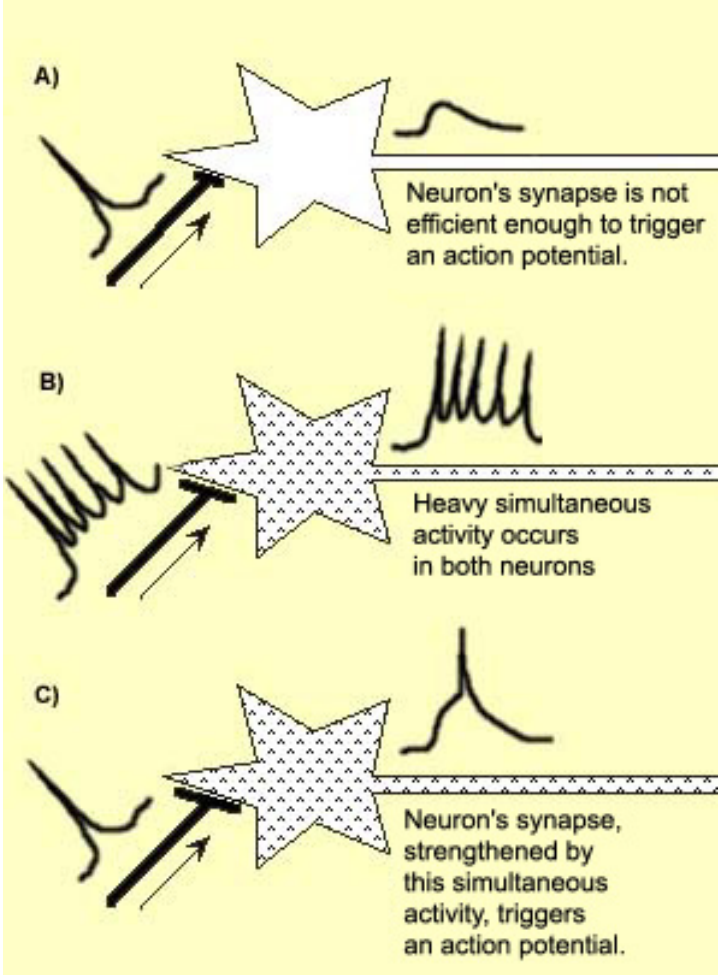
Neuron

- Axon
  - Transports signals to other cells
- Dendrites
  - Receive signals from other cells' axons at the synapses
- Soma (cell body):
  - "Sums" the signals from the dendrites
  - When membrane potential passes a threshold,
  - an action potential is sent down the axon, the cell "spikes" or "fires"



https://simple.wikipedia.org/wiki/Neuron#/media/File:Neuron.svg

Hebb's rule



When an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased

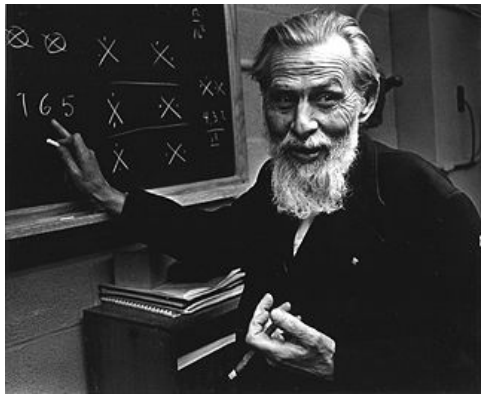
- Donald O. Hebb:
  The Organization of Behavior (1949)
  - Goal: The connections between behavior and neural activity
- Terje Lømo, UiO, 1966
  - *Long-term potentiation*
    - empirical confirmation

https://thebrain.mcgill.ca/flash/i/i_07/i_07_cl/i_07_cl_tra/i_07_cl_tra.html

McCulloch and Pitts

- Warren S. McCulloch and Walter Pitts, 1943:
  - "*A Logical Calculus of the Ideas Immanent in Nervous Activity*"
  - A formal simplified model of neurons
  - Showed how networks of these neurons could correspond to logical formulas

W.S. McCulloch b. 1898

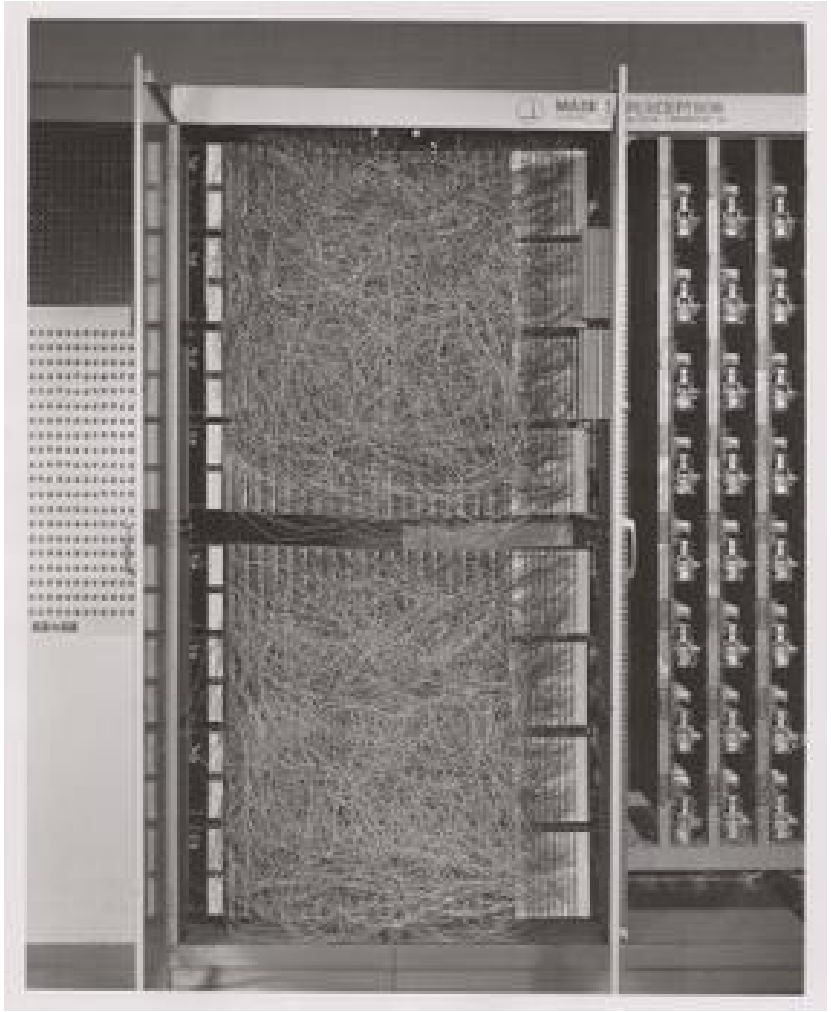




W. Pitts b. 1923

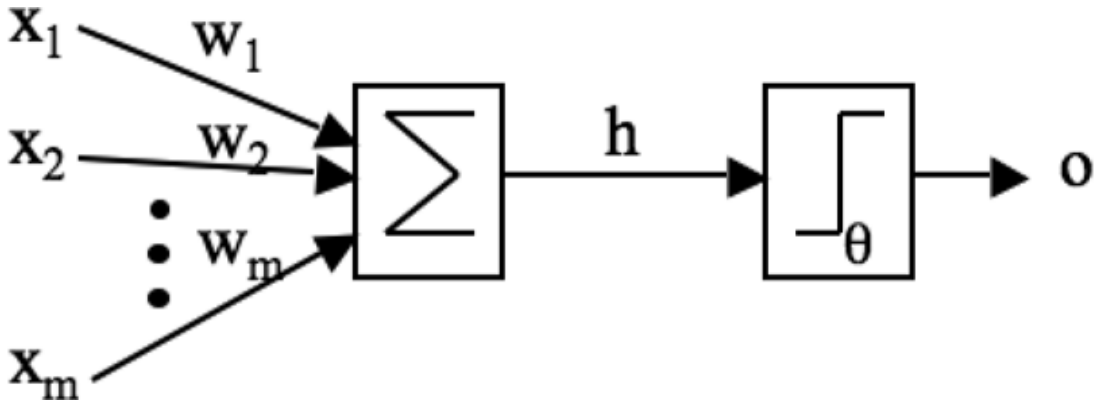
The Perceptron

- Frank Rosenblatt, 1958
- A learning algorithm,
  - which we will consider
- A custom-built machine
  - based on this algorithm
  - for image recognition



https://en.wikipedia.org/wiki/File:Mark_I_perceptron.jpeg

The Perceptron



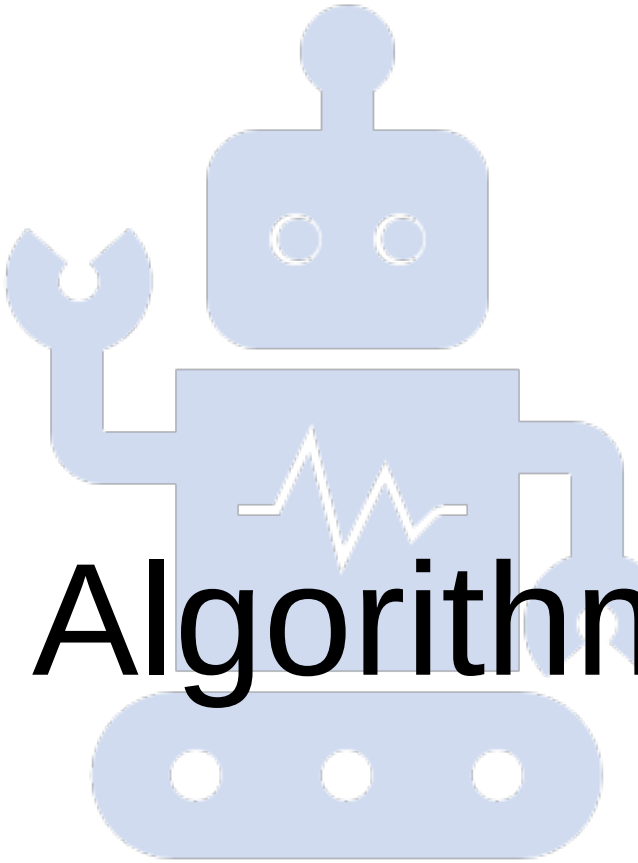
1. A set of inputs: $x_1, x_2, \ldots x_m$
2. A set of weights: $w_1, w_2, \ldots w_m$
3. An adder:

$$h = \sum_{i=1}^{m} w_i x_i = w_1 x_1 + w_2 x_2 + \cdots + w_m x_m$$

4. An activation function,

   Originally a step function:

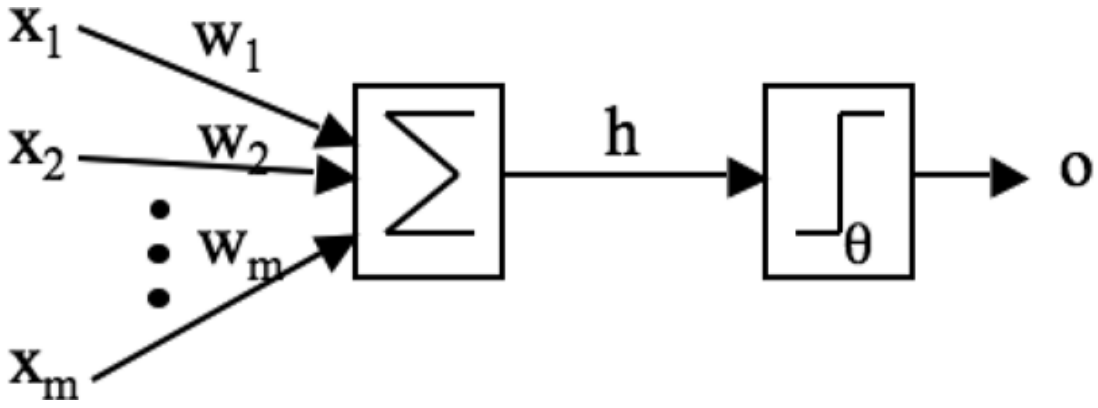
$$o = g(h) = \begin{cases} 1 \; if \; h > \theta \\ 0 \; if \; h \leq \theta \end{cases}$$



5.3 The Perceptron Algorithm

IN3050/IN4050 Introduction to Artificial Intelligence and Machine Learning

The Perceptron



1. A set of inputs: $x_1, x_2, \ldots x_m$
2. A set of weights: $w_1, w_2, \ldots w_m$
3. An adder:

$$h = \sum_{i=1}^{m} w_i x_i = w_1 x_1 + w_2 x_2 + \cdots + w_m x_m$$

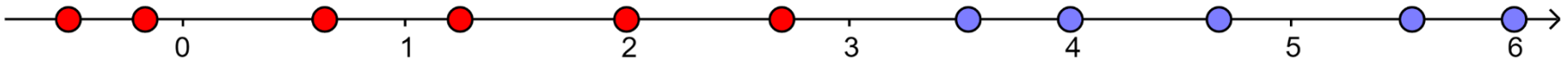
4. An activation function,

   Originally a step function:

$$o = g(h) = \begin{cases} 1 \; if \; h > \theta \\ 0 \; if \; h \leq \theta \end{cases}$$

Adjusting the threshold - example

- Consider the simplest situation with only one input: x_1
- Assume we have a fixed threshold, say $\theta = 1$
- Let blue be the positive class, and red the negative class

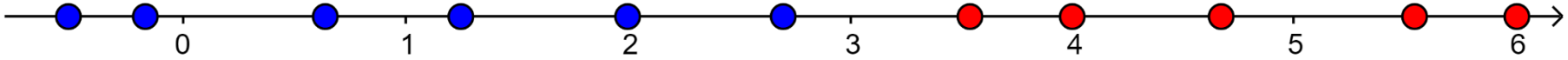


- We see that the positive class corresponds to $x_1 > 3$, or $\frac{1}{3}x_1 > 1$

$w_1 = \frac{1}{3}$ yield the desired outcome: $w_1 x_1 > 1$ iff $x_1 > 3$

Adjusting the threshold – example contd.

- Assume the same fixed threshold, say $\theta = 1$
- Let blue be the positive class, and red the negative class

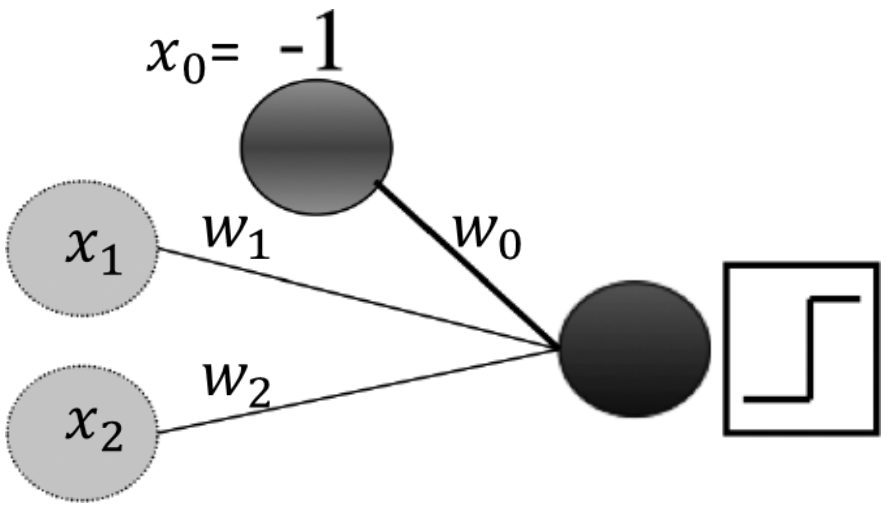


- We see that now the positive class corresponds to $x_1 < 3$
- Is there a w_1 such that $x_1 < 3$ if and only if $w_1 x_1 > 1$?
  - NO!
- If we instead can change the threshold to $\theta = -1$, we see that
- $x_1 < 3$ if and only if $w_1 x_1 > -1$ if $w_1 = -\frac{1}{3}$
- We must change the threshold as well as the weights!

The Bias Term

Since:

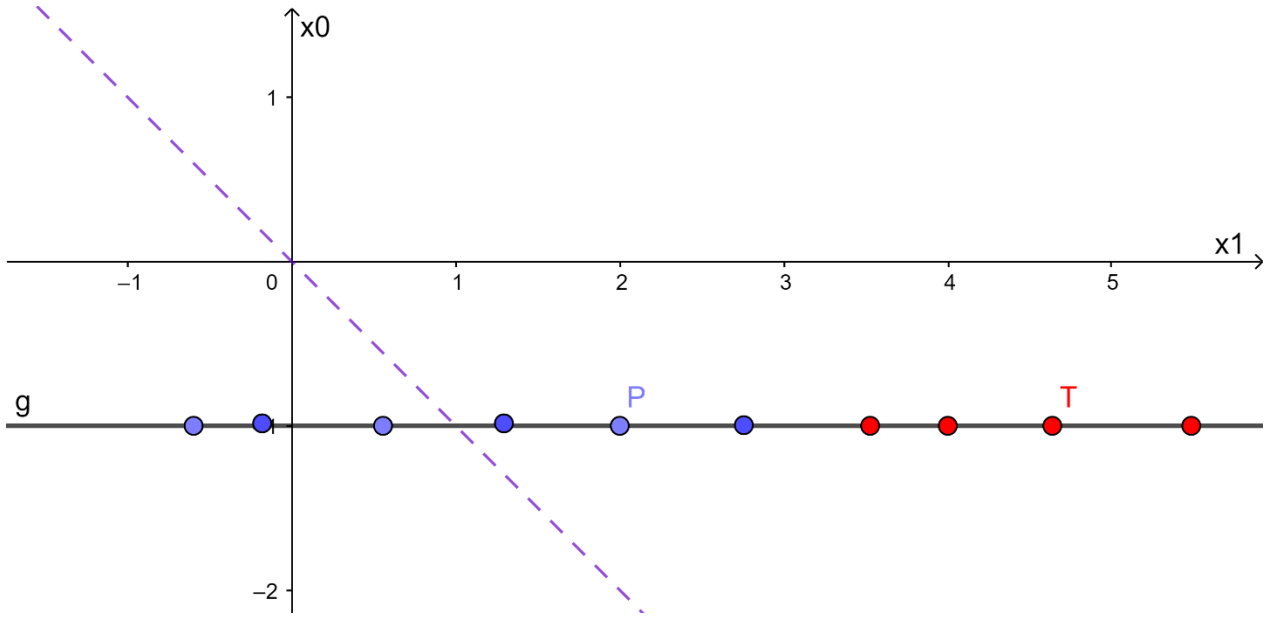
- $\sum_{i=1}^{m} w_i x_i > \theta$ is the same as
- $\sum_{i=1}^{m} w_i x_i - \theta > 0$, is the same as
- $\sum_{i=1}^{m} w_i x_i + w_0 x_0 = \sum_{i=0}^{m} w_i x_i > 0$
  - Provided $x_0 = -1$ (and $w_0 = \theta$)



We can

- add a new feature $x_0 = -1$ for all items
- Replace $(x_1, x_2, \ldots, x_m)$ with $(-1, x_1, x_2, \ldots, x_m)$
- Replace $\sum_{i=1}^{m} w_i x_i > \theta$ with $h = \sum_{i=0}^{m} w_i x_i > 0$

Redefined objective

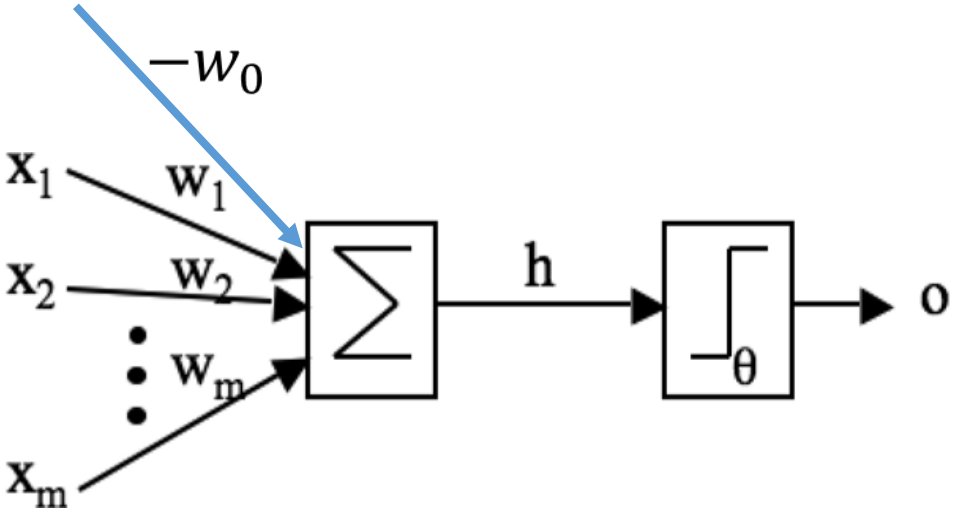


Geometric understanding

- Find a line $w_1 x_1 + w_0 x_0 = 0$
  - such that $h(-1, x_1) = w_1 x_1 + w_0 x_0 = w_1 x_1 - w_0 > 0$
  - if and only if $x_1 < 3$

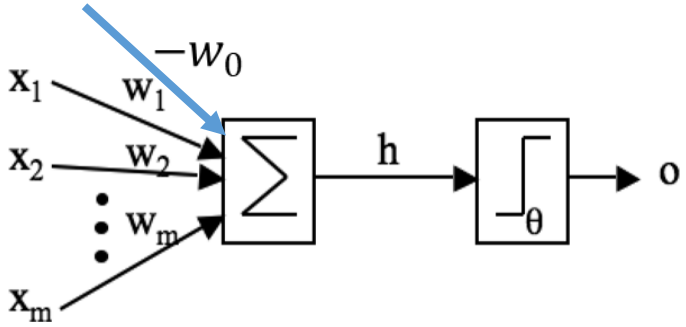
Training one perceptron

1. Initialize: set all weights to small random numbers, $w_0, w_1, \ldots w_m$
2. Repeat until <some criteria>:
   Consider one training instance
   - Inputs: $x_0, x_1, \ldots x_m$
   - Label: t, which is 1 or 0
   - Calculate the output of the perceptron
     $$y = o = g(\sum_{i=0}^{m} w_i x_i)$$
   - If $y = t$, do nothing, if $y \neq t$, update weights



Update weights

$\eta > 0$ is the fixed learning rate



if $t = 1, y = 0$

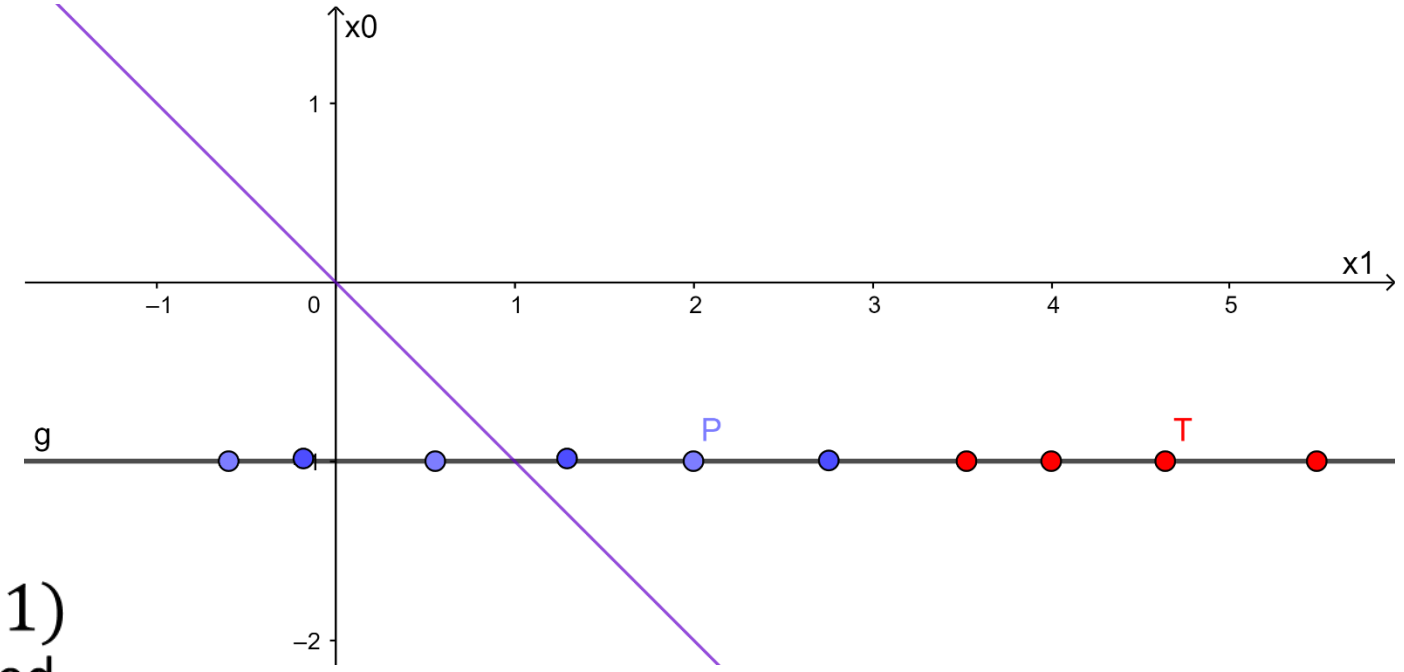
- increase $\sum_{i=0}^{m} w_i x_i$
  - by increasing each $w_i x_i$:
    - if $x_i > 0$: increase w_i
    - if $x_i < 0$: decrease w_i
  - $w_i = w_i + \eta x_i$
    - cover both cases

if $t = 0, y = 1$

- decrease $\sum_{i=0}^{m} w_i x_i$
  - by decreasing each $w_i x_i$:
    - if $x_i > 0$: decrease w_i
    - if $x_i < 0$: increase w_i
  - $w_i = w_i - \eta x_i$

$w_i = w_i + \eta(t - y)x_i = w_i - \eta(y - t)x_i$
(covers all cases)

Example

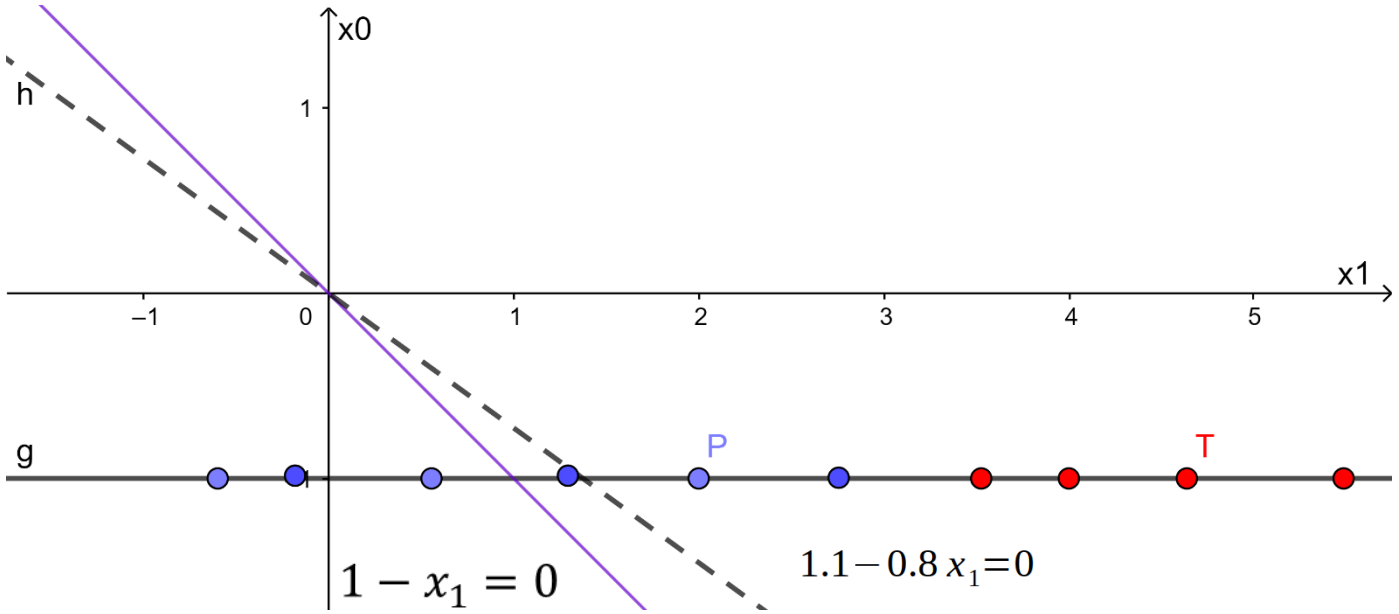


- We are in the middle of training
- Learning rate: $\eta = 0.1$
- Current weights: $(-1, -1)$
  - i.e., positive class provided
  - $h = -w_0 + w_1 x_1 = 1 - x_1 > 0$
  - $1 > x_1$

- Consider the point T=(-1,4.6):
  - $h(T) = 1 - 4.6 < 0$
  - Do nothing

- Consider the point P=(-1,2):
  - $h(P) = 1 - 2 < 0$
  - Wrongly classified
  - Update:
  - $w_0 = w_0 - \eta(y - t)x_0 = -1 - 0.1(0 - 1)(-1) = -1.1$
  - $w_1 = w_1 - \eta(y - t)x_1 = -1 - 0.1(0 - 1)(2) = -0.8$

Example

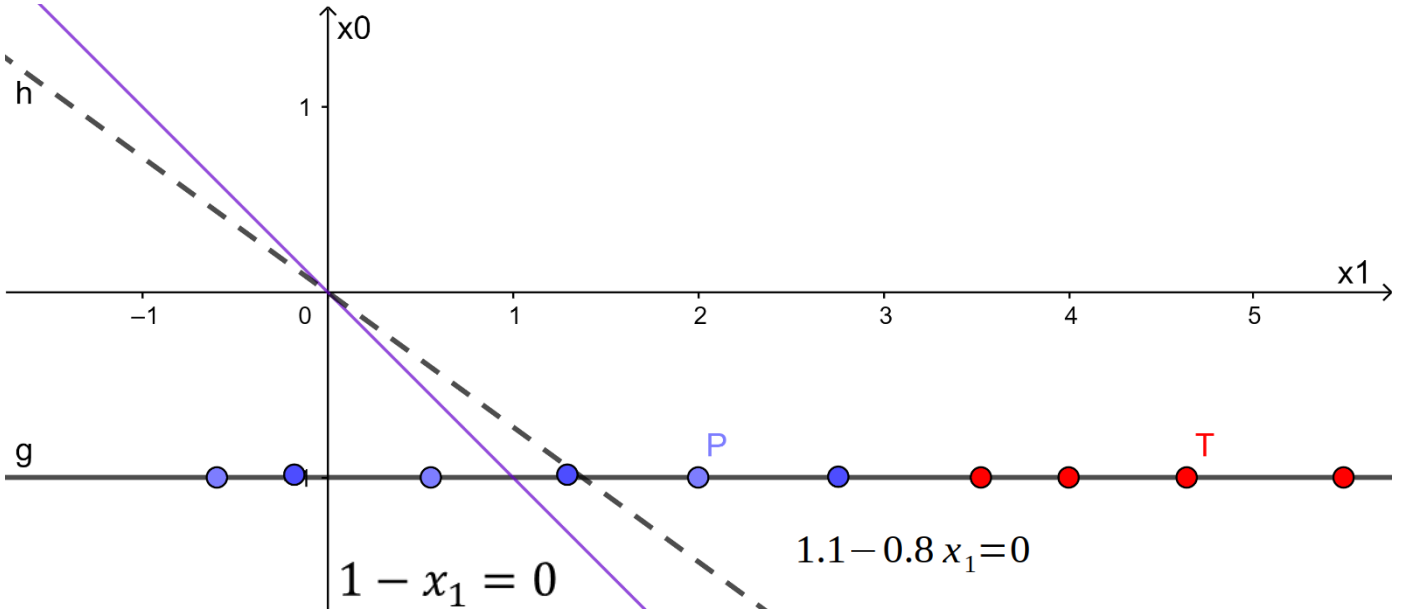


- We are in the middle of training
- Learning rate: $\eta = 0.1$
- Current weights: $(-1, -1)$
  - i.e., positive class provided
  - $h = -w_0 + w_1 x_1 = 1 - x_1 > 0$
  - $1 > x_1$

- Consider the point T=(-1,4.6):
  - $h(T) = 1 - 4.6 < 0$
  - Do nothing

- Consider the point P=(-1,2):
  - $h(P) = 1 - 2 < 0$
  - Wrongly classified
  - Update:
  - $w_0 = w_0 - \eta(y - t)x_0 = -1 - 0.1(0 - 1)(-1) = -1.1$
  - $w_1 = w_1 - \eta(y - t)x_1 = -1 - 0.1(0 - 1)(2) = -0.8$

Observe



- Many possible solutions
  - $w_0 = -1.1, w_1 = -0.8$
  - $w_0 = -2.2, w_1 = -1.6$
  - $w_0 = -5.5, w_1 = -4.0$
- Same line
- Same classifier

- But
  - $w_0 = 1.1, w_1 = 0.8$
- Same line
- But swaps the two classes!

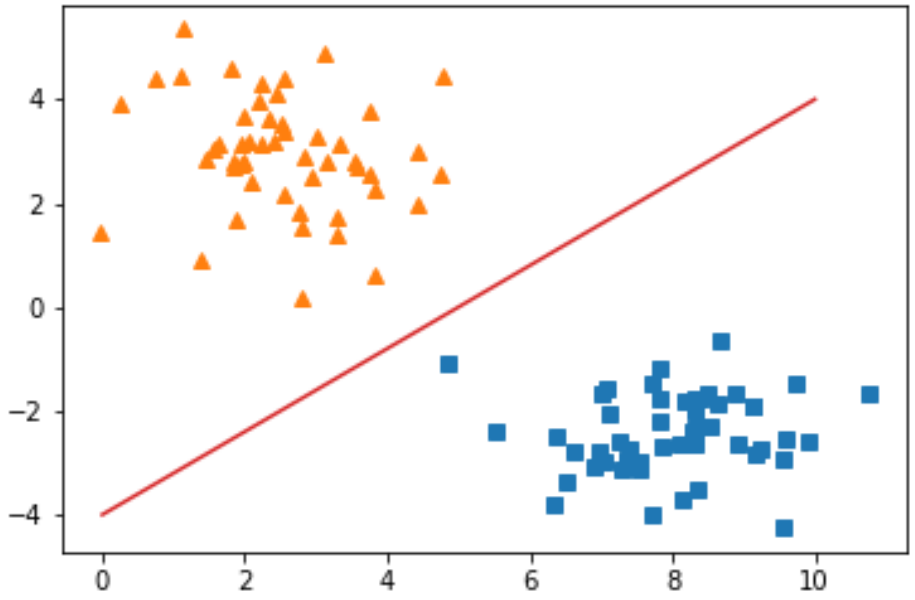
Properties

- With only one training item
  - the algorithm will sooner or later classify the item correctly
  - and no longer update

- When there are several training items, there might be disagreement:
  - one item will increase a certain weight w_i
  - another item will decrease it
- What then?

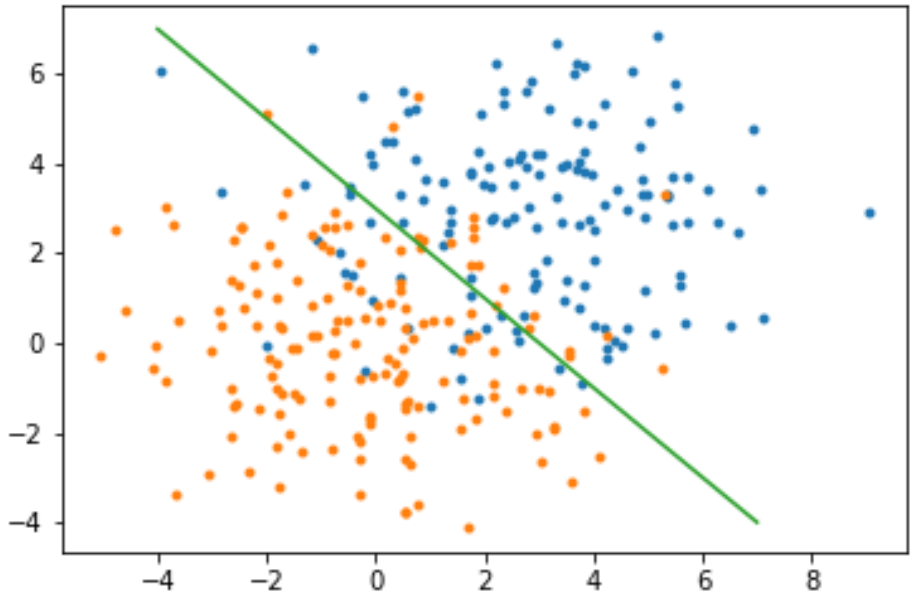
Linear separability

- A set is linearly separable if there is a straight line in the feature plane such that all points in one class fall on one side and all points in the other class fall at the other side
- For more than two features, this generalizes to a hyper-plane
- (With one dimension to a point, cf. the example so far)



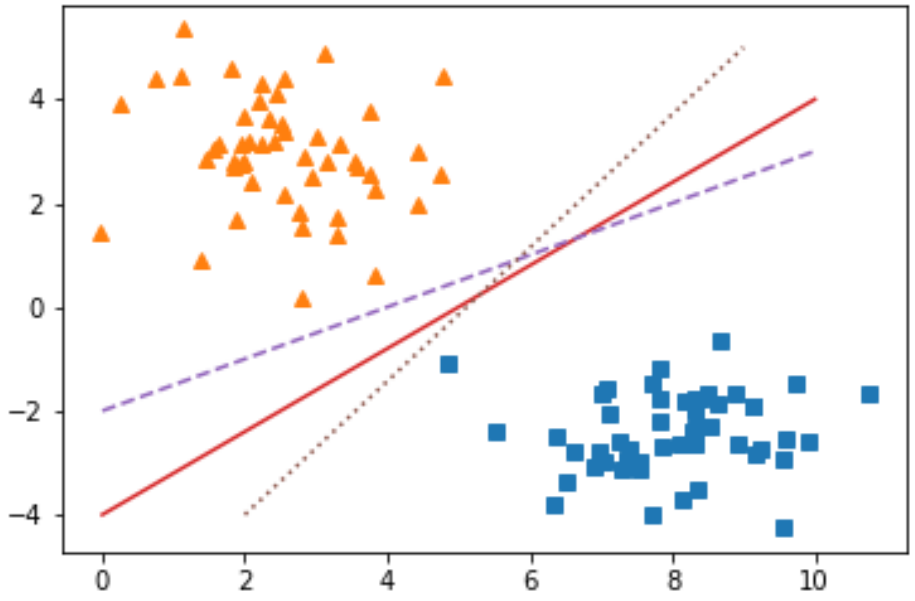
Linear classifier

- A linear classifier will always propose a linear decision boundary
  - (point, line, plane, hyper-plane)
  - whether the set is linearly separable or not
- The perceptron is a linear classifier



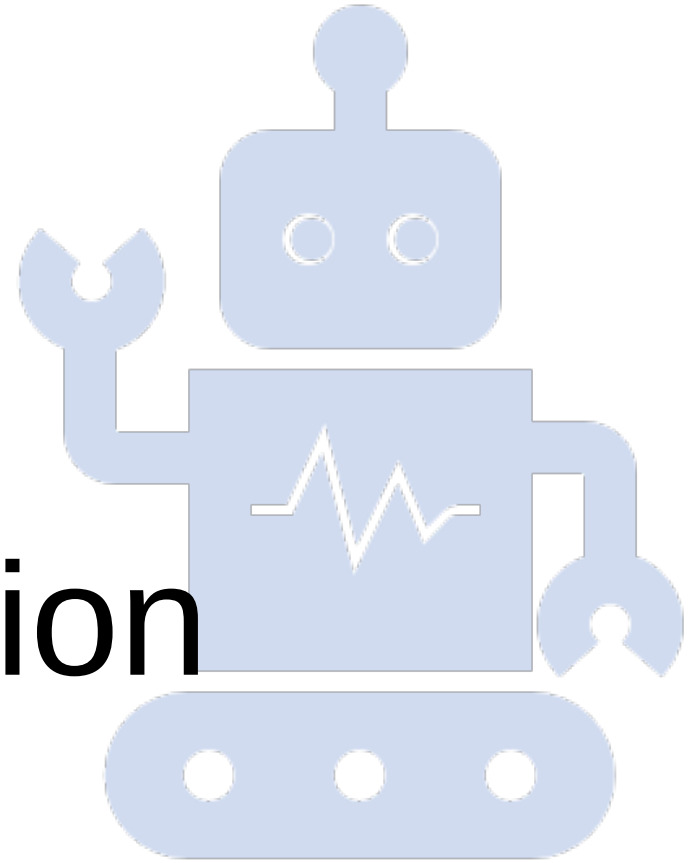
Perceptron Convergence Theorem

- If the training set is linearly separable, the perceptron algorithm will (sooner or later)
  - find a linear decision boundary
  - stop updating
- Unless the learning rate η is too large
- Comment:
  - There are normally more than one solution, which generalizes differently to test data



Perceptron summary

- The brain, neuron and synapsis
- The perceptron
- The bias term
- The perceptron algorithm
- Linear classifiers
- Linear separability



5.4 Linear Regression

IN3050/IN4050 Introduction to Artificial Intelligence and Machine Learning

Supervised learning – two types

Classification

- Assign a label (class) from a finite set of labels to an observation

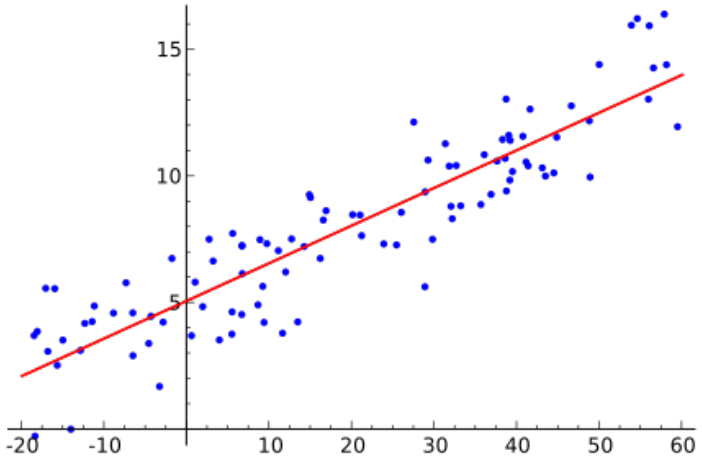




?

Regression

- Assign a numerical value to an observation
  - e.g., the temperature tomorrow



Supervised learning

- Each observation (datapoint) is described as a feature vector
  - $\boldsymbol{x}_j = (x_{j,1}, x_{j,2}, \ldots, x_{j,m})$
  - The "input"
- There is a well-defined set of possible target values, *T*
- The goal is for an input to predict a target value $f(\boldsymbol{x}_j)$ from T
- For supervised learning, we have a training set
  - $\{(\boldsymbol{x}_1, t_1), (\boldsymbol{x}_2, t_2), \ldots (\boldsymbol{x}_N, t_N)\}$
- We try to learn the function f from the training set

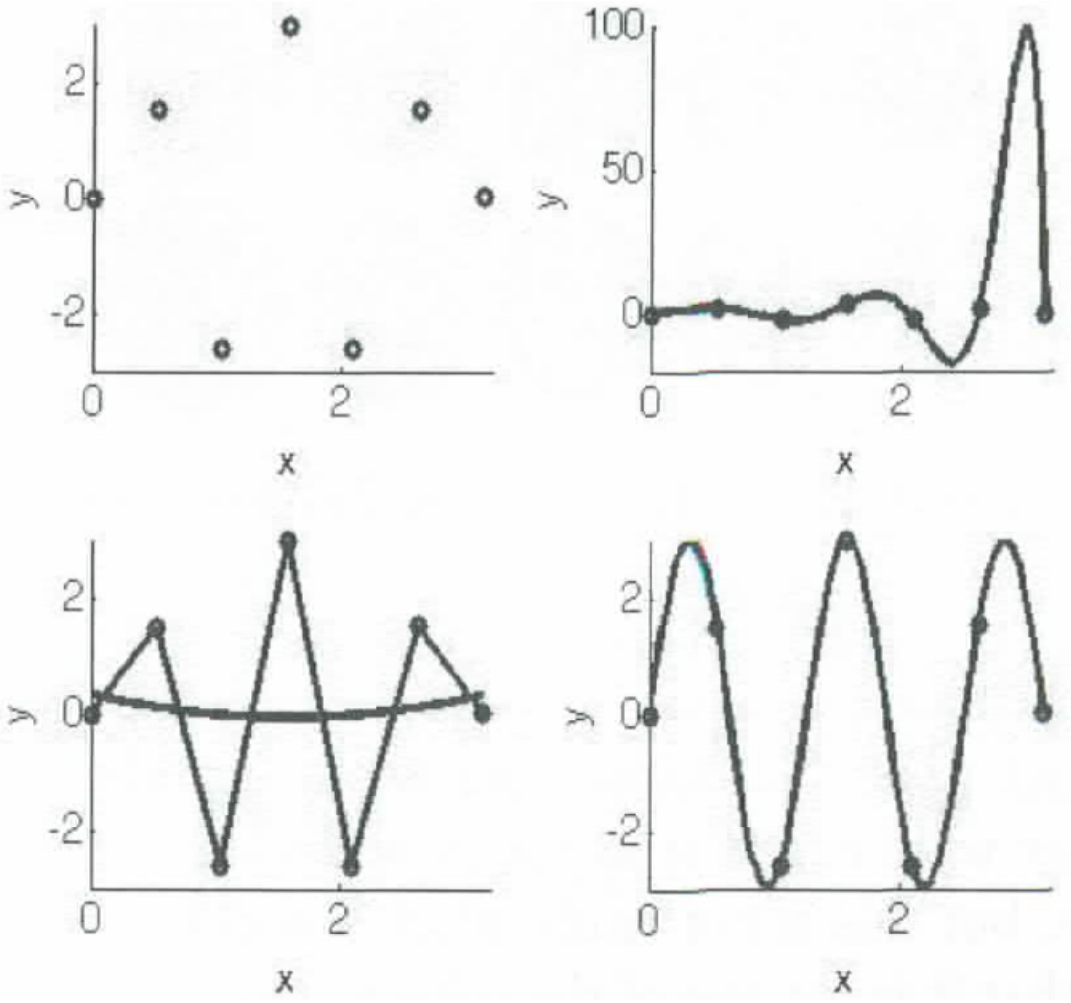
Regression

- In classification, the target set is a set of categories:
  - The goal is to predict one of these.
- In regression, the target set is real numbers:
  - The goal is to predict a $y_j = f(\boldsymbol{x}_j)$ which is close to the true t_j
- It is a general problem of function approximation:
  - working out the value between values that we know

An example from Marsland

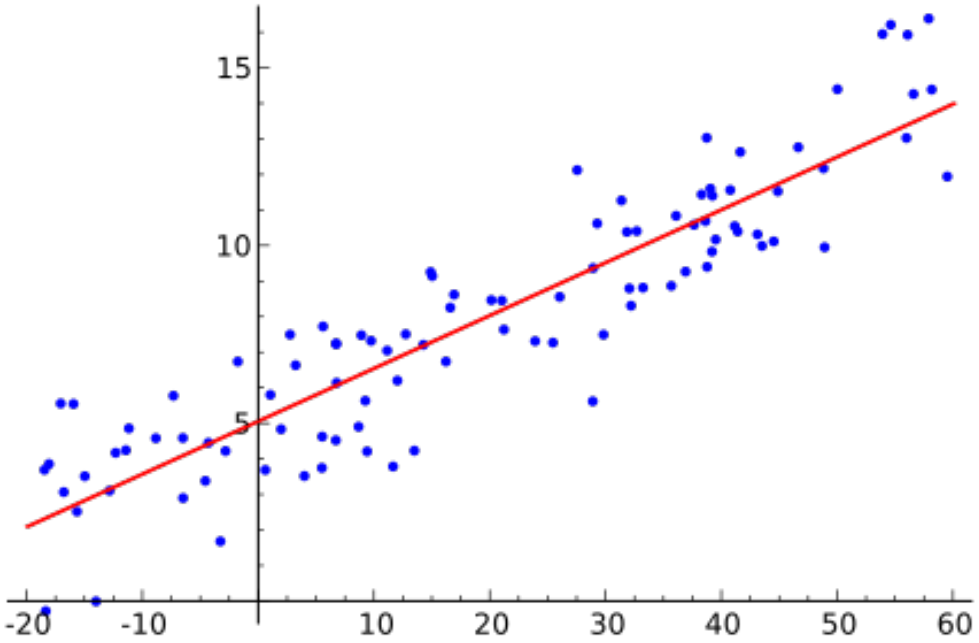
- Given, the following data, can we find the value of the output when $x = 0.44$?

x	t
0	0
0.5236	1.5
1.0472	-2.5981
1.5708	3.0
2.0944	-2.5981
2.6180	1.5
3.1416	0



Linear regression

- We need some idea regarding the kind of functions we are looking for
- The simplest is to assume a linear function
  - $f(\boldsymbol{x}) = f((x_1, x_2, \ldots, x_m)) = w_0 + w_1 x_1 + w_2 x_2 + \ldots w_m x_m$



Of course, this isn't always a good fit, but linear regression may also be adopted to some non-linear functions by feature engineering.

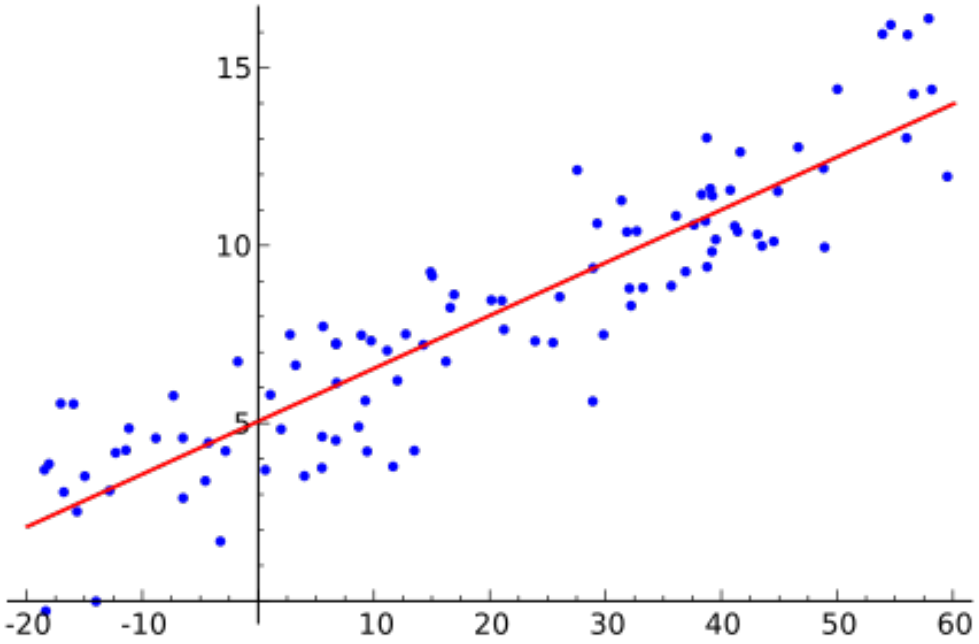
Inductive bias (read Daumé: ciml)

- To learn from data, you must have some idea regarding how the data are distributed.
- You choose a model.
- You try to find parameters which makes the model fit the training data well.

- Models carry with them inductive biases, e.g.,
  - Linear regression can only learn straight lines.
  - Perceptron can only learn linear decision boundaries

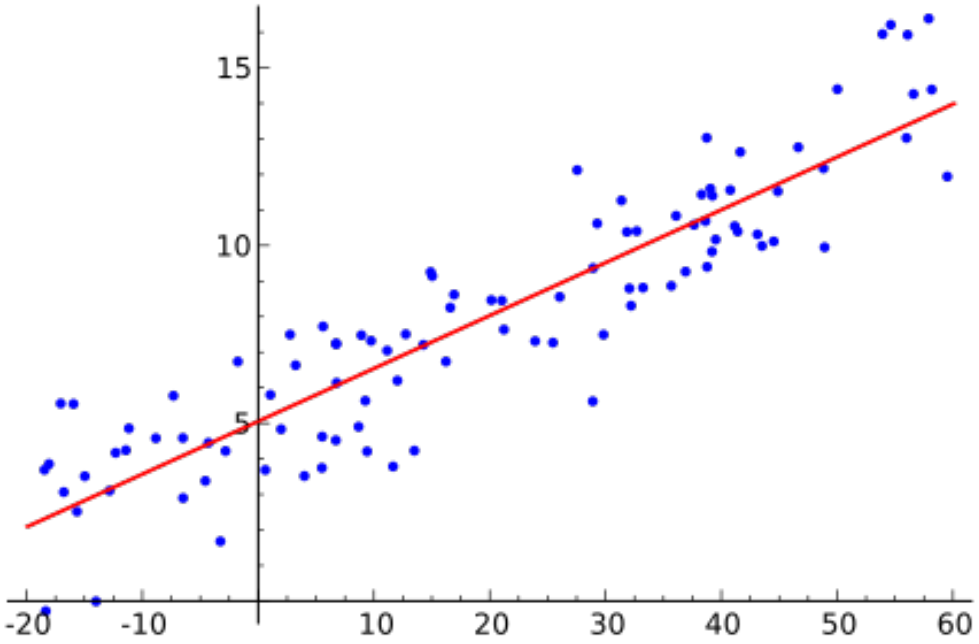
Linear regression

- When there is only one input variable
  - called simple linear regression
  - $f(x_1) = w_0 + w_1 x_1$
  - a straight line
  - easy to draw
- In the general case
  - $f(\boldsymbol{x}) = w_0 + w_1 x_1 + w_2 x_2 + \ldots w_m x_m$
  - describes a hyper-plane
  - harder to draw



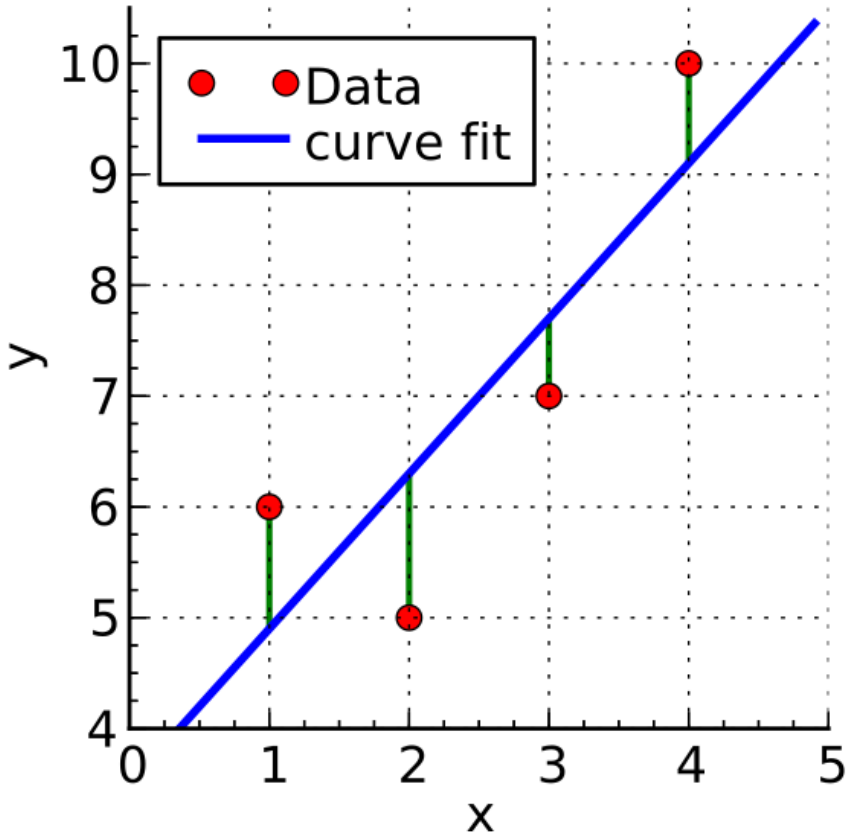
Notation

- $f(\boldsymbol{x}) = w_0 + w_1 x_1 + w_2 x_2 + \ldots w_m x_m$
- Often used notation:
- $f(\boldsymbol{x}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots \beta_m x_m$
- The β_i-s are parameters of the model
- $\beta_1, \beta_2 x_2, \ldots \beta_m$ called coefficients
- β_0 called intercept
- We have used w_i for weights
- We will assume a $x_0 = 1$ (bias)



Mean Square Error (MSE)

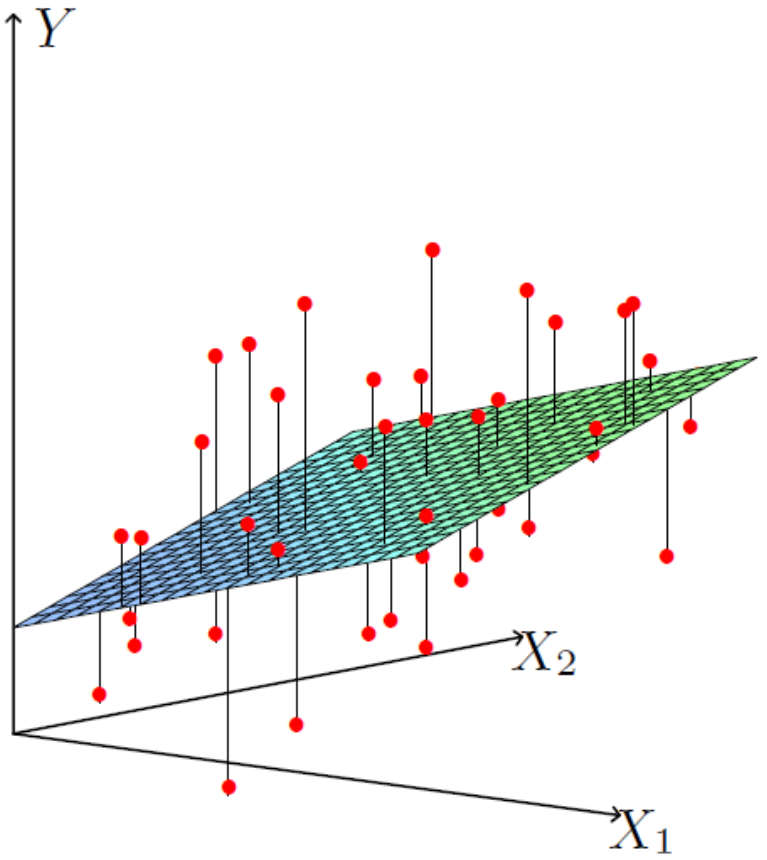
- We need a way to tell whether a line is a good fit to the data, and whether one line is better than another
- $\frac{1}{N}\sum_{j=1}^{N}\left(t_j - \sum_{i=0}^{m} w_i x_{j,i}\right)^2$
- $= \frac{1}{N}\sum_{j=1}^{N}\left(t_j - y_j\right)^2$
- The goal is to minimize this error.



https://en.wikipedia.org/wiki/Linear_least_squares

MSE in 3D

- With two input variables, we are trying to find a plane.
- The MSE is similar
- $\frac{1}{N}\sum_{j=1}^{N}\left(t_j - \sum_{i=0}^{m} w_i x_{j,i}\right)^2$



Trevor Hastie et al, **The Elements of Statistical Learning**

Footnote: variants

- Root Mean Square Error(RMSE):
  - $\sqrt{\frac{1}{N}\sum_{j=1}^{N}\left(t_j - \sum_{i=0}^{m} w_i x_{j,i}\right)^2}$
- MSE: $\frac{1}{N}\sum_{j=1}^{N}\left(t_j - \sum_{i=0}^{m} w_i x_{j,i}\right)^2$
- SE: $\sum_{j=1}^{N}\left(t_j - \sum_{i=0}^{m} w_i x_{j,i}\right)^2$
- $\frac{1}{2}\sum_{j=1}^{N}\left(t_j - \sum_{i=0}^{m} w_i x_{j,i}\right)^2$

- They all have minimum for the same values of x_i-s and w_i-s.
- RMSE has the ''right scale'', but not suited for finding min.
- Mean (MSE or RMSE) is needed for comparison across different training/test sets

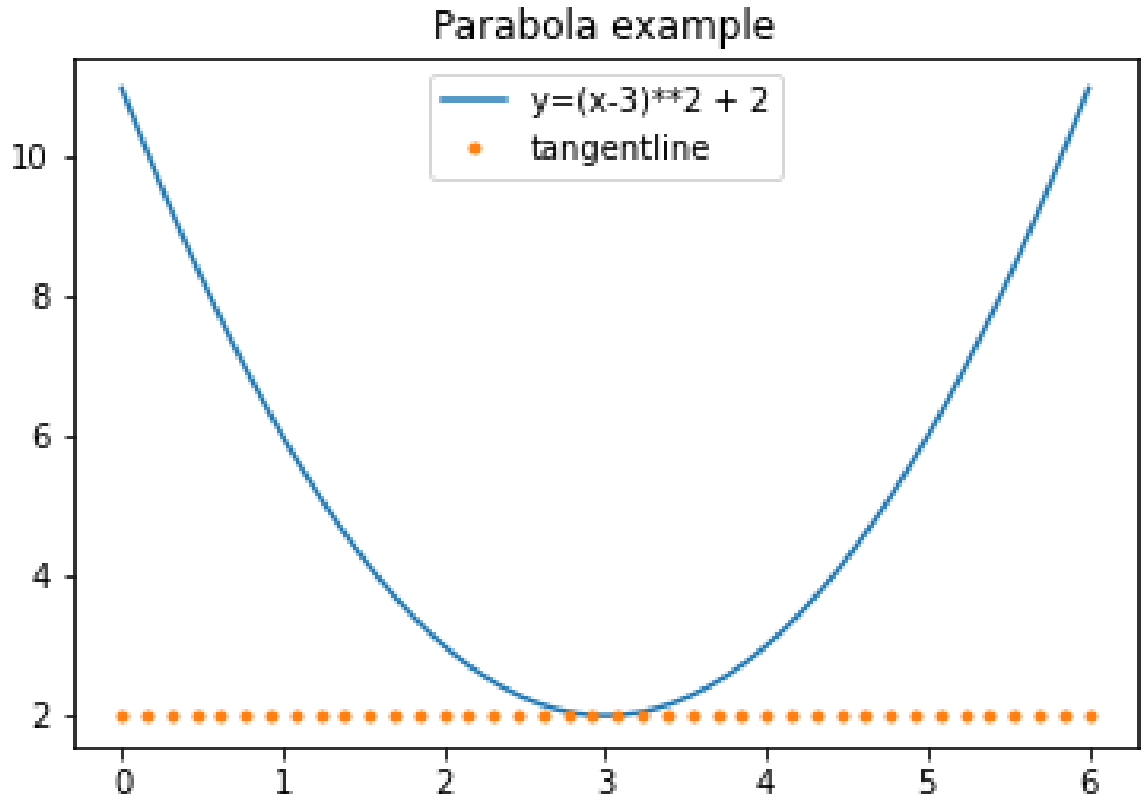
Goal

- The goal is to find the $w_0, w_1, \ldots\ w_m$ that minimizes the MSE

$$\frac{1}{N}\sum_{j=1}^{N}\left(t_j - \sum_{i=0}^{m} w_i x_{ji}\right)^2$$

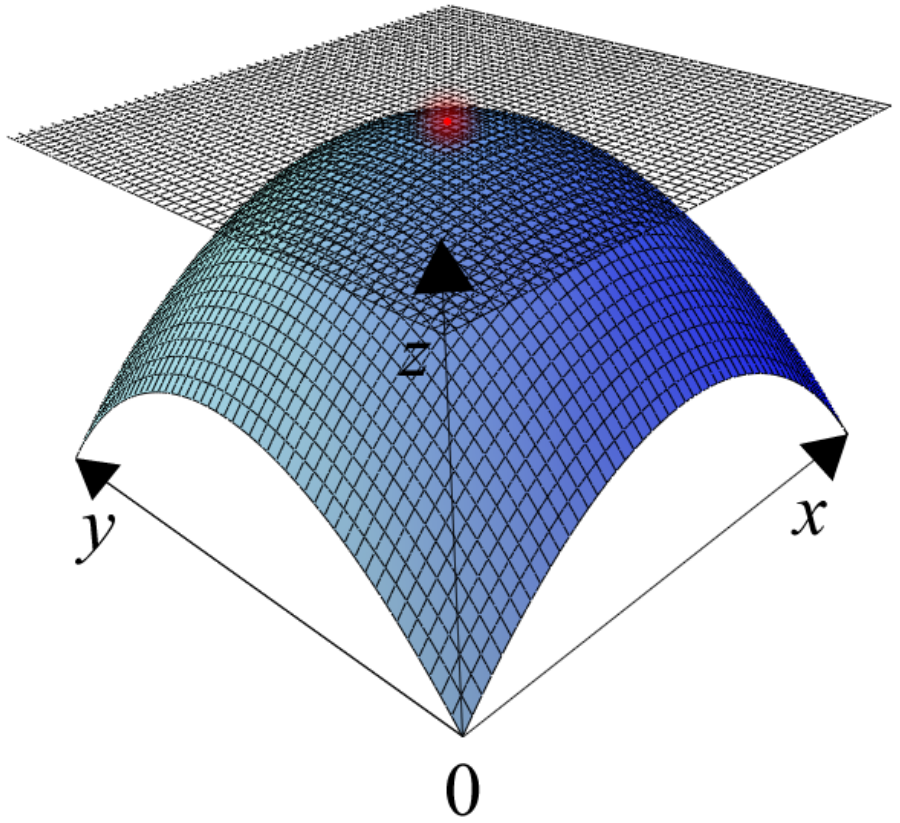
Minimizing: one variable

- We assume you know how to minimize with one variable:
- $f(x) = (x-3)^2+2$
- $f'(x) = 2(x-3)$
- $f'(x) = 0$ iff $x = 3$



The minimization problem

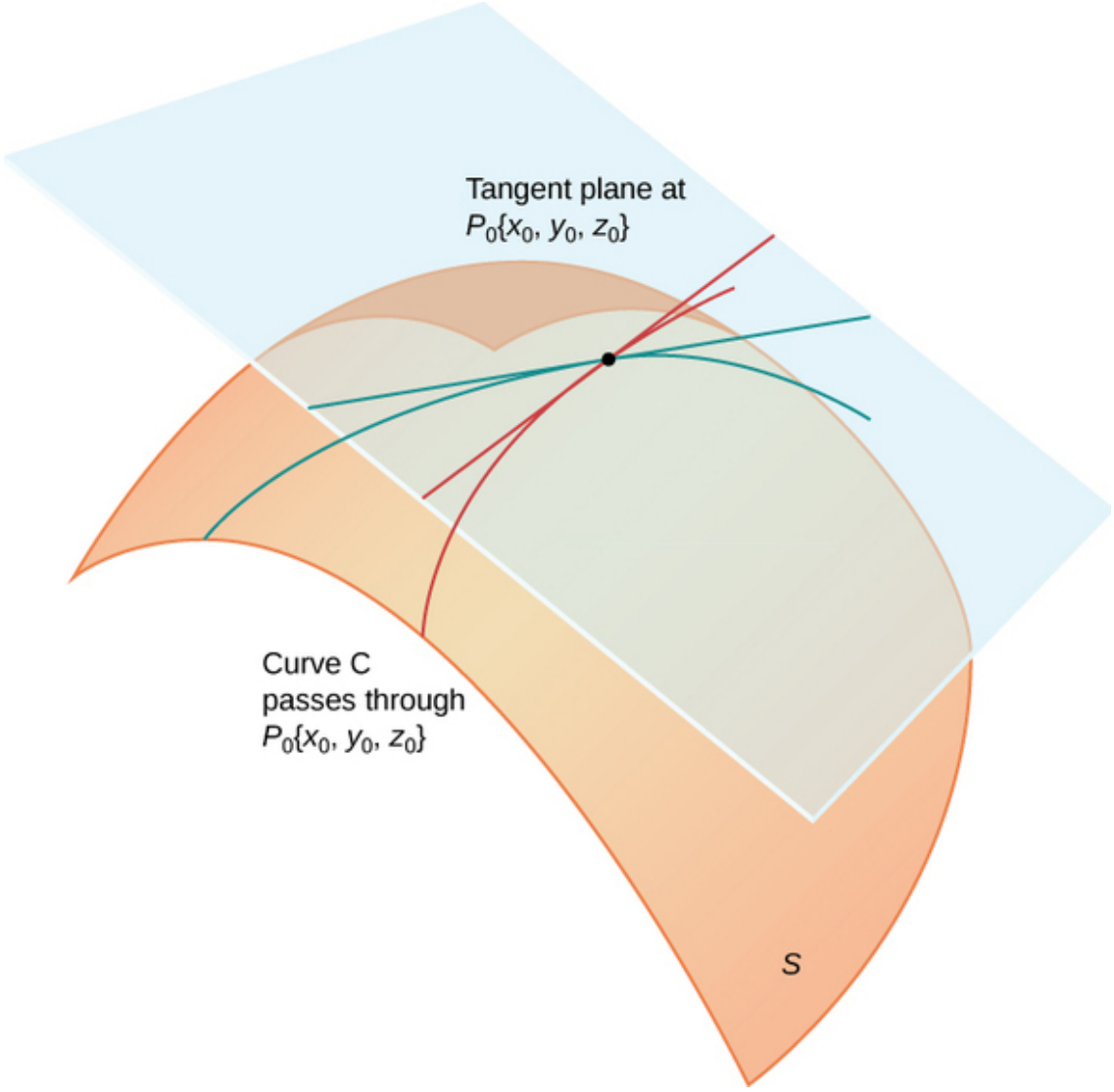
- (Here as a maximization problem, ‘’upside-down’’, easier to draw.)
- Looking for a point where the tangent plane is horizontal:
  - Where all the derivatives are 0
- The MSE is convex:
  - There is only one global minimum
  - No problem with local optima



https://en.wikipedia.org/wiki/File:Maximum_tangentplane_boxed.png

Tangent plane

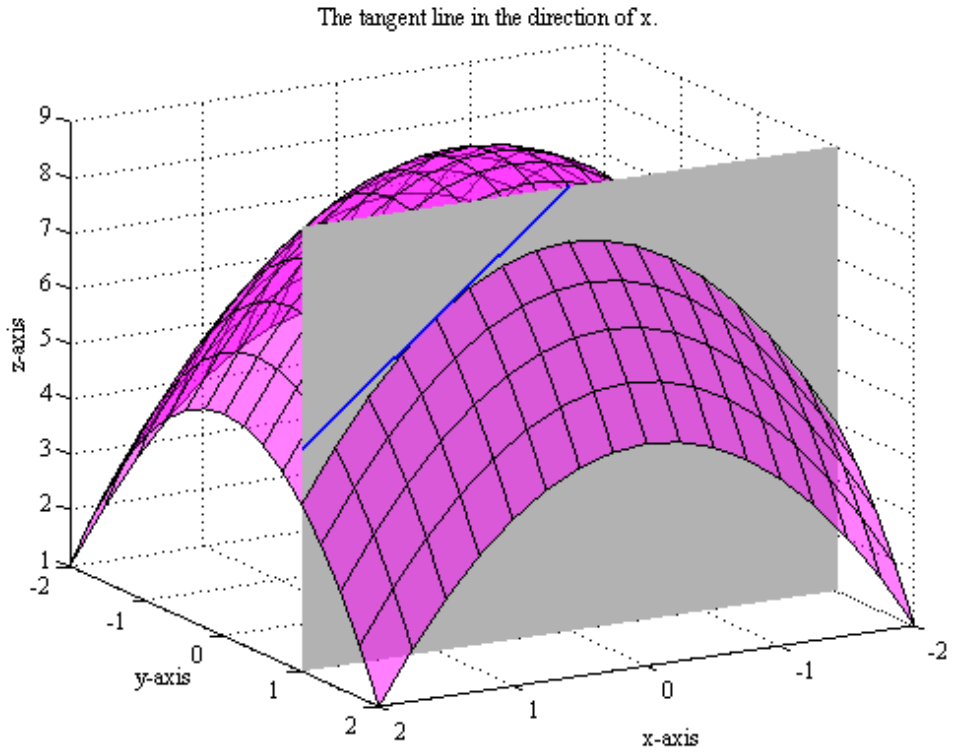
- Just like the derivative in 2D determines a tangent to the curve,
- the partial derivatives in more dimensions determines tangents to the surface parallel to the axes
- These tangents together determine a tangent (hyper-)plane to the surface
- It is the steepest direction in this plane, we follow when we follow gradient ascent/descent.



Picture source

Partial derivatives

- We assume you know
  - If $f(x) = (a + bx)^2$, then
  - $f'(x) = \frac{d}{dx} f(x) = 2(a + bx)b$
- Extended to more dimensions, we can construct partial derivatives, e.g.
  - $g(x, y) = (a + bx + cy)^2$
  - $\frac{\partial}{\partial x} g(x, y) = 2(a + bx + cy)b$
  - $\frac{\partial}{\partial y} g(x, y) = 2(a + bx + cy)c$



https://www.wikihow.com/Image:OyXsh.png

Minimizing the MSE

- To minimize $f = \frac{1}{N}\sum_{j=1}^{N}\left(t_j - \sum_{i=0}^{m} w_i x_{j,i}\right)^2$ with respect to the $w_0, w_1, \ldots w_m$
- we can calculate the partial derivatives
- $\frac{\partial}{\partial w_k} f = \frac{2}{N}\sum_{j=1}^{N}\left((t_j - \sum_{i=0}^{m} w_i x_{j,i})(-x_{j,k})\right)$ for $k = 1, 2, \ldots, m$
- $\frac{\partial}{\partial w_k} f = \frac{2}{N}\sum_{j=1}^{N}\left((t_j - y_j)(-x_{j,k})\right)$ for $k = 1, 2, \ldots, m$

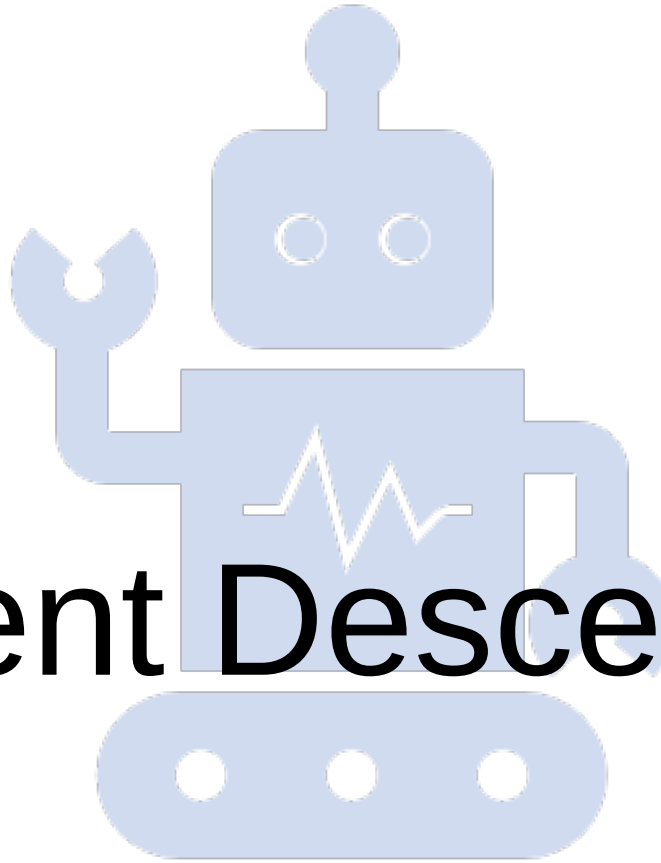
- (Observe that this is just a generalization of
  - $\frac{\partial}{\partial x} g(x, y) = 2(a + bx + cy)b$)

Closed form solution

- To minimize $f = \frac{1}{N}\sum_{j=1}^{N}\left(t_j - \sum_{i=0}^{m} w_i x_{ji}\right)^2$
- We can set these partial derivatives to equal 0:
  $\frac{\partial}{\partial w_k} f = \frac{2}{N}\sum_{j=1}^{N}\left((t_j - \sum_{i=0}^{m} w_i x_{ji})(-x_{jk})\right) = 0$, for $k = 1, 2, \ldots, m$
- We get m many linear equations of m unknown w_i-s
- We know how to solve such a system.

Some good and some bad news

- This has a closed form solution, i.e., there is a recipe for calculating the solution:
  - This works fine for low dimensions (few features for each observation)
- But it gets slow for more dimensions.
  - Standard algorithms are $O(m^3)$ where m is the number of dimensions.
  - In ML, we may have millions of features/dimensions
- And since it does not require ML, we will not go into it.
- But we can always use gradient descent – next video!

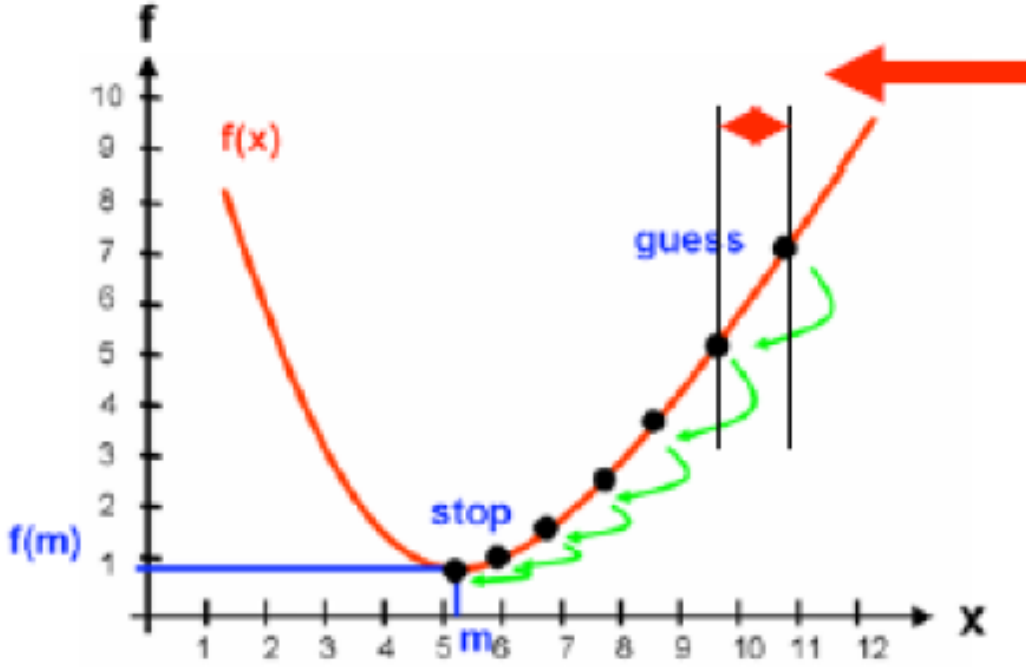


5.5 Applying Gradient Descent

IN3050/IN4050 Introduction to Artificial Intelligence and Machine Learning

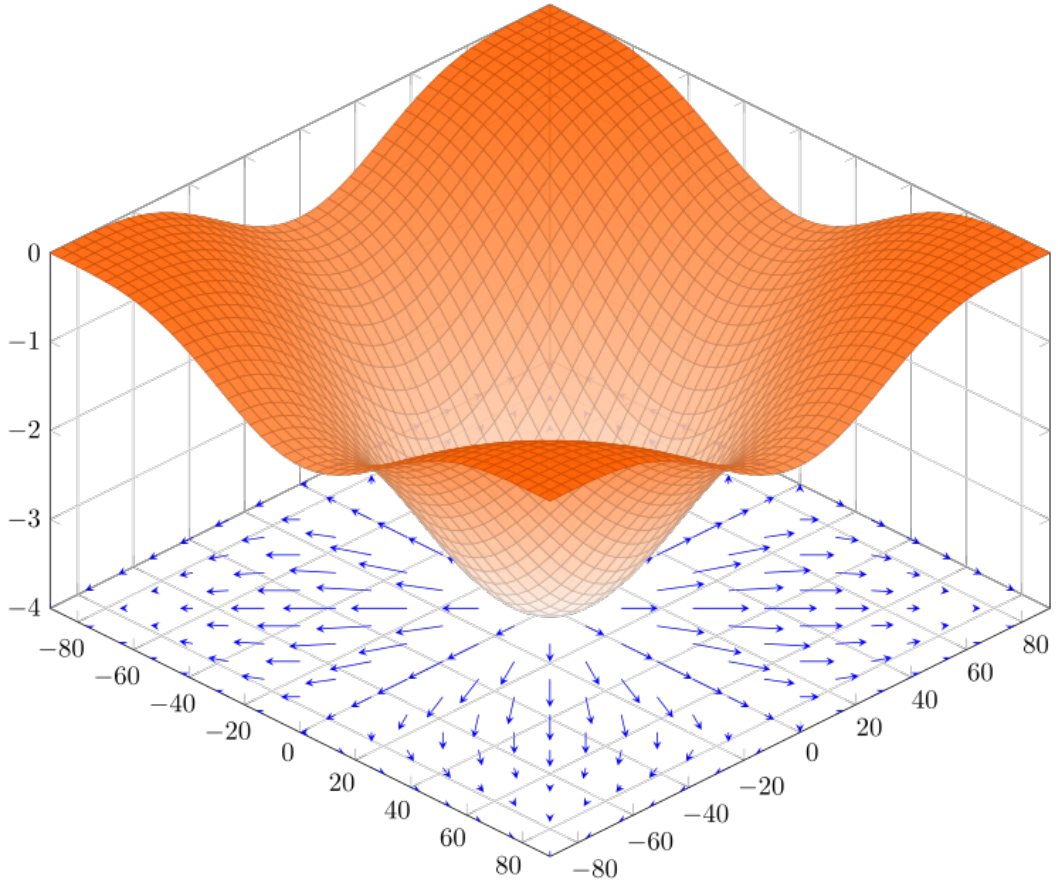
Gradient descent in one variable

- (Lecture 2)
- Start with x_0
- Iteratively find, $x_1, x_2, .., x_i, \ldots$ with decreasing $f(x_i)$ by setting
- $x_{i+1} = x_i - \gamma f'(x_i)$
- γ is the learning rate



The gradient in more dimensions

- We move in the opposite direction of the gradient.



Minimizing the MSE

- To minimize $f = \frac{1}{N}\sum_{j=1}^{N}\left(t_j - \sum_{i=0}^{m} w_i x_{j,i}\right)^2$ with respect to the $w_0, w_1, \ldots w_m$
- we can calculate the partial derivatives
- $\frac{\partial}{\partial w_k} f = \frac{2}{N}\sum_{j=1}^{N}\left((t_j - \sum_{i=0}^{m} w_i x_{j,i})(-x_{j,k})\right)$ for $k = 1, 2, \ldots, m$
- $\frac{\partial}{\partial w_k} f = \frac{2}{N}\sum_{j=1}^{N}\left((t_j - y_j)(-x_{j,k})\right)$ for $k = 1, 2, \ldots, m$

- (Observe that this is just a generalization of
  - $\frac{\partial}{\partial x} g(x, y) = 2(a + bx + cy)b$)

Prediction (matrix form)

$$y_j = y_{j,1} = \sum_{i=0}^{m} w_i x_{j,i} = (x_{j,0}, x_{j,1}, \ldots, x_{j,m}) \cdot (w_{0,1}, w_{1,1}, \ldots, w_{m,1})$$

$$\begin{bmatrix} x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,0} & x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{N,0} & x_{N,1} & x_{N,2} & \cdots & x_{N,m} \end{bmatrix} \begin{bmatrix} w_{0,1} \\ w_{1,1} \\ w_{2,1} \\ \vdots \\ w_{m,1} \end{bmatrix} = \begin{bmatrix} y_{1,1} \\ y_{2,1} \\ \vdots \\ y_{N,1} \end{bmatrix} \sim \begin{bmatrix} t_{1,1} \\ t_{2,1} \\ \vdots \\ t_{N,1} \end{bmatrix}$$

The gradient

$$\frac{\partial}{\partial w_k} f = \frac{2}{N} \sum_{j=1}^{N} \left((t_j - y_j)(-x_{j,k}) \right)$$

$$\nabla f = \begin{bmatrix} \frac{\partial}{\partial w_0} f \\ \frac{\partial}{\partial w_1} f \\ \frac{\partial}{\partial w_2} f \\ \vdots \\ \frac{\partial}{\partial w_m} f \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^{N} (t_j - y_j)(-x_{j,0}) \\ \sum_{j=1}^{N} (t_j - y_j)(-x_{j,1}) \\ \sum_{j=1}^{N} (t_j - y_j)(-x_{j,2}) \\ \vdots \\ \sum_{j=1}^{N} (t_j - y_j)(-x_{j,m}) \end{bmatrix} = X^T (Y - T)$$

$$\begin{bmatrix} x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,0} & x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{N,0} & x_{N,1} & x_{N,2} & \cdots & x_{N,m} \end{bmatrix} \begin{bmatrix} w_{0,1} \\ w_{1,1} \\ w_{2,1} \\ \vdots \\ w_{m,1} \end{bmatrix} = \begin{bmatrix} y_{1,1} \\ y_{2,1} \\ \vdots \\ y_{N,1} \end{bmatrix} \sim \begin{bmatrix} t_{1,1} \\ t_{2,1} \\ \vdots \\ t_{N,1} \end{bmatrix}$$

Implementing the gradient descent

- Input:
  - X, input, a $N \times m$ NumPy matrix:
    - N items, m features
  - T, corresponding target values,
    - $N \times 1$ column vector
    - (maybe it is given as a vector of length N and must be transformed)
  - Make a weight matrix, W,
    - $m \times 1$ column vector
- Forward step:
  - $Y = X@W$
- Update step:
  - $W = W - \eta \nabla f$
  - $W -= \eta X.T(Y - T)$
  - (η, *eta*, is a learning rate)

Summary

- Linear regression
- Inductive bias
- Mean Square Error
- Minimizing the MSE
- Using Gradient Descent
- in Matrix form