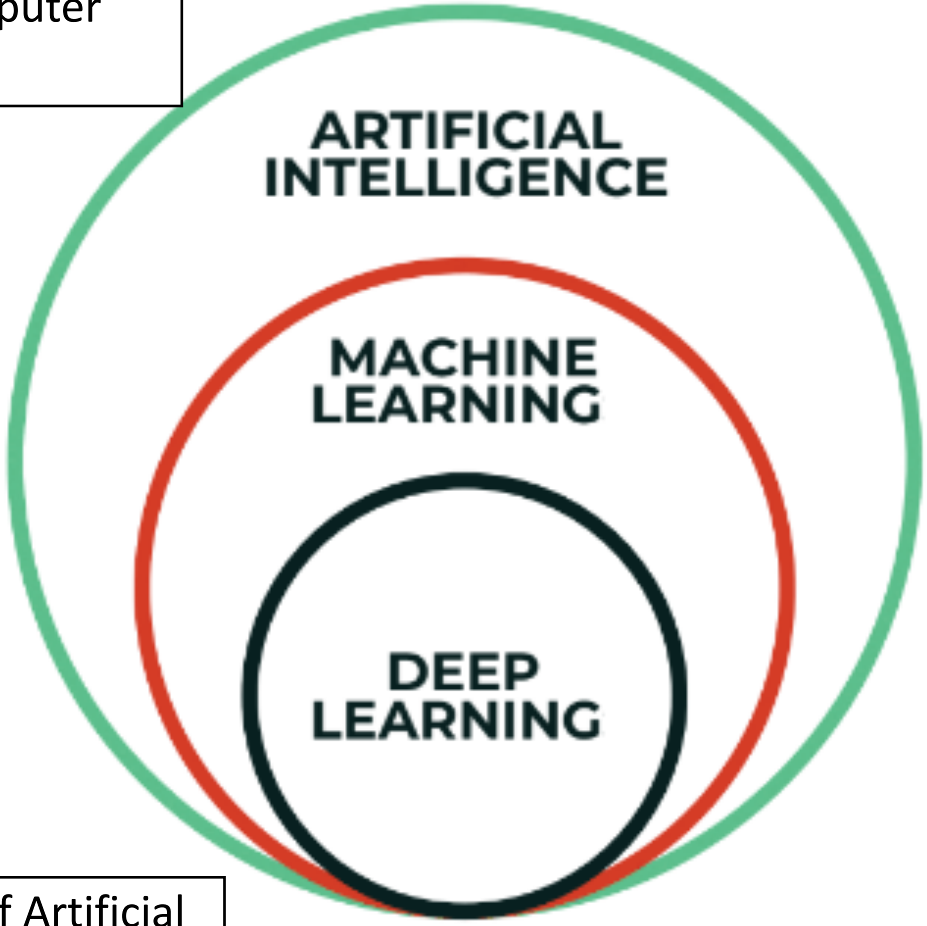
AI is a broader field (i.e.: the big umbrella) that contains several subfields such as machine learning, robotics, and computer vision

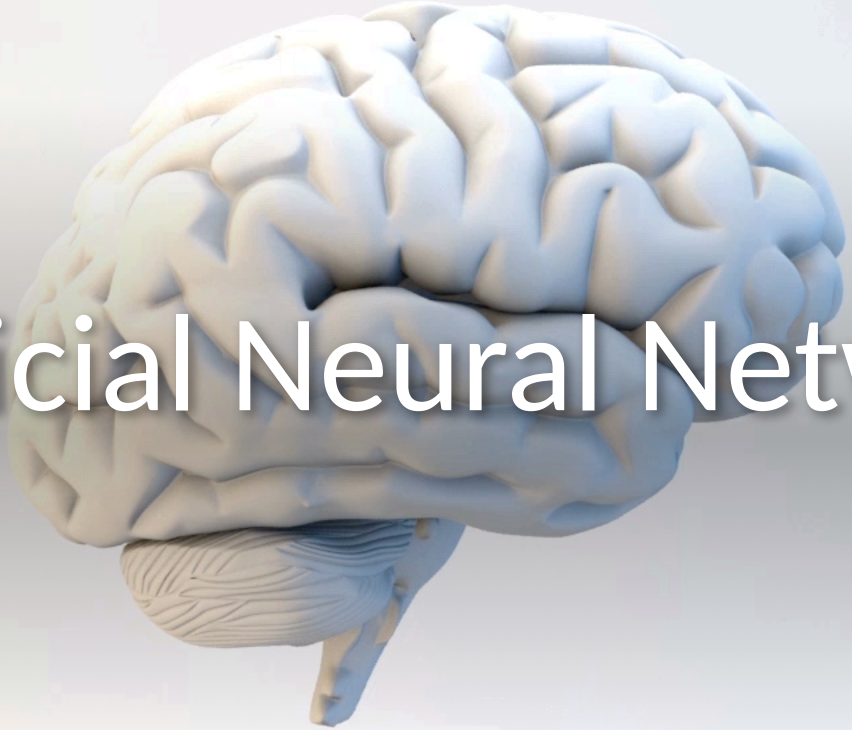


Machine Learning is a subfield of Artificial Intelligence that enables machines to improve at a given task with experience

Deep Learning is a subfield of ML, where the algorithm tries to mathematically mimic a biological neural network.

PLUS: it "learns" the features on its own

Artificial Neural Networks



Black box analogy

Training

There are 4 parts to understand

- ❑ Data
- ❑ Model
- ❑ Optimization Algorithm
- ❑ Objective Function

Training - Data

[{"index": 1, "x": 0.59857177734375, "y": 6.53887939453125, "z": 5.97772216796875}, {"index": 1, "x": 0.59857177734375, "y": 6.53887939453125, "z": 5.97772216796875}, {"index": 2, "x": 0.951507568359375, "y": 6.8403778076171875, "z": 5.99566650390625}, {"index": 3, "x": 0.9012603759765625, "y": 6.699188232421875, "z": 5.9083251953125}, {"index": 4, "x": 0.7373504638671875, "y": 6.416839599609375, "z": 5.448913574218750}, {"index": 5, "x": 0.34613037109375, "y": 6.3546295166015625, "z": 5.4130096435546875}, {"index": 6, "x": 0.846221923828125, "y": 6.6429595947265625, "z": 5.409423828125}, {"index": 7, "x": 0.92877197265625, "y": 6.609466552734375, "z": 5.5182952880859375}, {"index": 8, "x": 0.8593902587890625, "y": 6.4419708251953125, "z": 5.543426513671875}, {"index": 9, "x": 1.3116302490234375, "y": 6.421630859375, "z": 5.3484039306640625}, {"index": 10, "x": 1.54852294921875, "y": 6.263702392578125, "z": 5.152191162109375}, {"index": 11, "x": 1.1931915283203125, "y": 6.146453857421875, "z": 5.0612640380859375}, {"index": 12, "x": 0.78759765625, "y": 5.9825439453125, "z": 5.02178955078125}, {"index": 13, "x": 0.4741363525390625, "y": 5.6798553466796875, "z": 5.2132110595703125}, {"index": 14, "x": 0.3485260009765625, "y": 5.3209228515625, "z": 5.3424224853515625}, {"index": 15, "x": 0.374847412109375, "y": 5.2993927001953125, "z": 5.4178009033203125}, {"index": 16, "x": 0.5543060302734375, "y": 5.454925537109375, "z": 5.122283935546875}, {"index": 17, "x": 0.5315704345703125, "y": 5.779159545898375, "z": 4.77532958984375}, {"index": 18, "x": 0.623687744140625, "y": 6.0184326171875, "z": 4.4869842529296875}, {"index": 19, "x": 0.5734405517578125, "y": 6.1871337890625, "z": 4.118499755859375}, {"index": 20, "x": 0.1032562255859375, "y": 6.51495361328125, "z": 3.9641571044921875}, {"index": 21, "x": -0.2425079345703125, "y": 6.9169464111328125, "z": 4.1567840576171875}, {"index": 22, "x": -0.2544708251953125, "y": 7.29620361328125, "z": 4.68798828125}, {"index": 23, "x": 0.516021728515625, "y": 7.6264190673828125, "z": 5.0600738525390625}, {"index": 24, "x": 1.59637451171875, "y": 7.412261962890625, "z": 4.8327484130859375}, {"index": 25, "x": 1.78302001953125, "y": 7.1334991455078125, "z": 4.7908782958984375}, {"index": 26, "x": 0.9503173828125, "y": 7.015045166015625, "z": 5.4943695068359375}, {"index": 27, "x": 0.380828857421875, "y": 6.5472564697265625, "z": 6.4383392333984375}, {"index": 28, "x": 0.410736083984375, "y": 6.110565185546875, "z": 6.583099365234375}, {"index": 29, "x": 0.0003662109375, "y": 6.2493438720703125, "z": 6.299560546875}, {"index": 30, "x": -0.667236328125, "y": 6.9564208984375, "z": 6.376129150390625}, {"index": 31, "x": -

Training - Model

- There are several types of model.

  Example: Linear

  $$V_1 + V_2 + V_3 + \ldots$$

Training – Objective Function

- It is the error from the labeled prediction

  We want to minimize it!

Output	Real Value	Objective Fun. Score
		470
		50
		4

Training – Optimization Algorithm

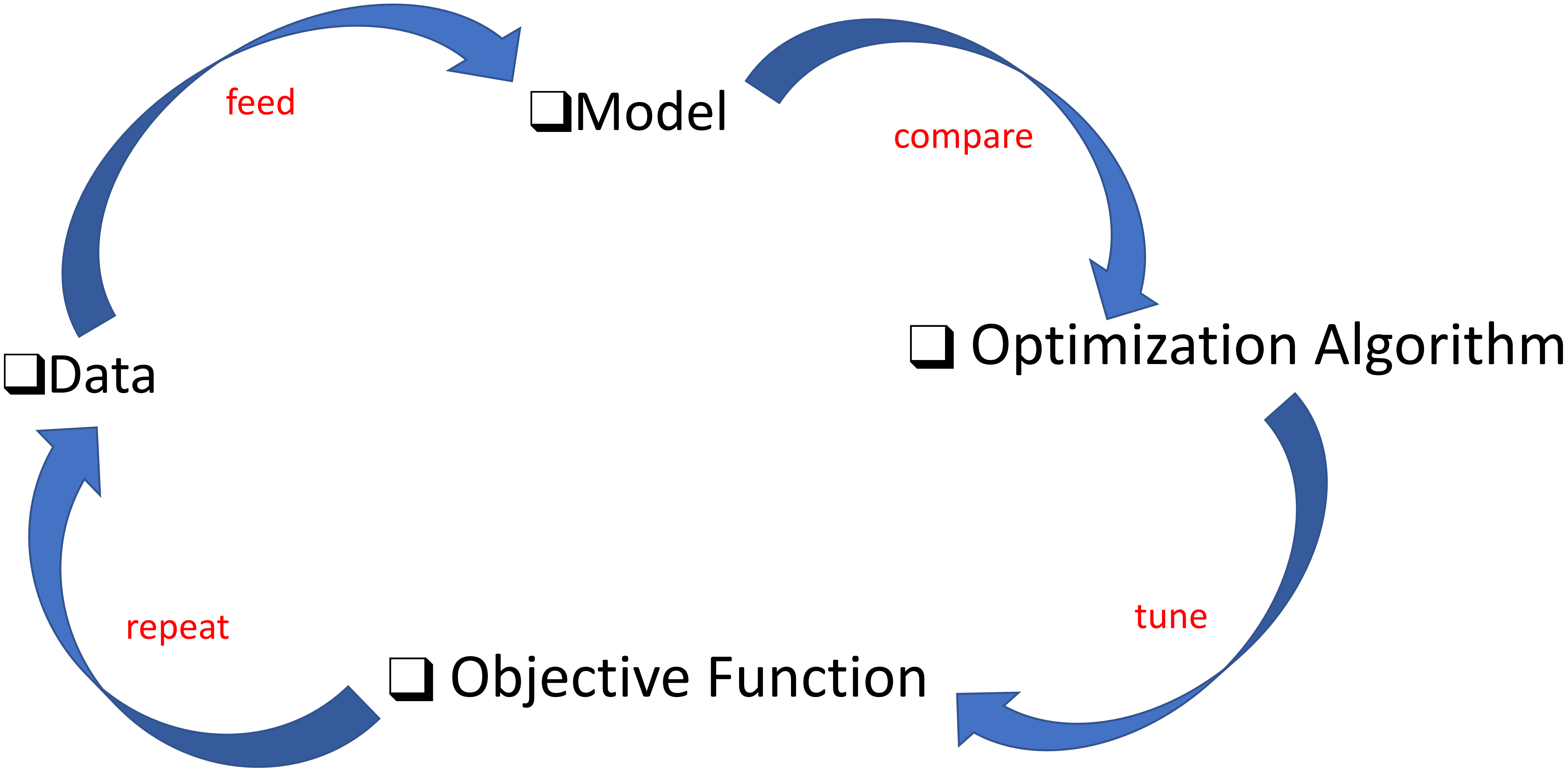
- The mechanics through which we vary the parameters of the model to optimize the Objective Function

For example, we can tune the α values in a Model that looks like this:

$$\boldsymbol{\alpha_0} V_0 + \boldsymbol{\alpha_1} V_1 + \boldsymbol{\alpha_2} V_2 + \ldots$$

Note that the α values can also be negative

Training


feed
Model
compare
Optimization Algorithm
Data
tune
repeat
Objective Function

Types of Machine Learning

Trial-and-error

- Kind of brute force attack to password guessing, but more optimized

  A trial-and-error model with feedbacks

- It could reach a “perfect” solution that is impossible to reach with a standard and schematic approach

  That’s one “type” of ML

Mimicking behavior

Example: Self driving vehicles

~~“A curve! I must turn”~~

“A curve, I saw others turning, I will do it too!”

That’s another “type” of ML

Types of ML

❑ Supervised

- We have inputs, target outputs (feedback)
- The **objective function** in supervised learning is called **loss function** (also, **cost** or **error**)
- Split in **Classification**, that provides outputs (categories)
    - The labels are not ordered and cannot be compared
- And **Regression**, that provides numerical values
    - The labels can be compared

Types of ML

❑ Unsupervised

- Only inputs, no target outputs
- Dependence or underline logic must be found by the algorithm
- Good for data labeling (**clustering**)

❑ Reinforcement

- Sit! Good boy/girl
- The **objective function** is called **reward function**
- Good for Decision process and Reward system

The Linear Model

Supervised

The Linear Model

$$f(x) \to \ y$$

- What is f?

These are observations that we found:

$f\,(2.4) = \ 12$

$$f(1.7) = \ 10$$

$$f(0.6) \ = \ -6$$

...

The Linear Model

In the classic **Linear Model**:

$f(x) = xa + b$

Where:

x is the **input**

a is the **coefficient** of x

b is the **intercept**

The Linear Model

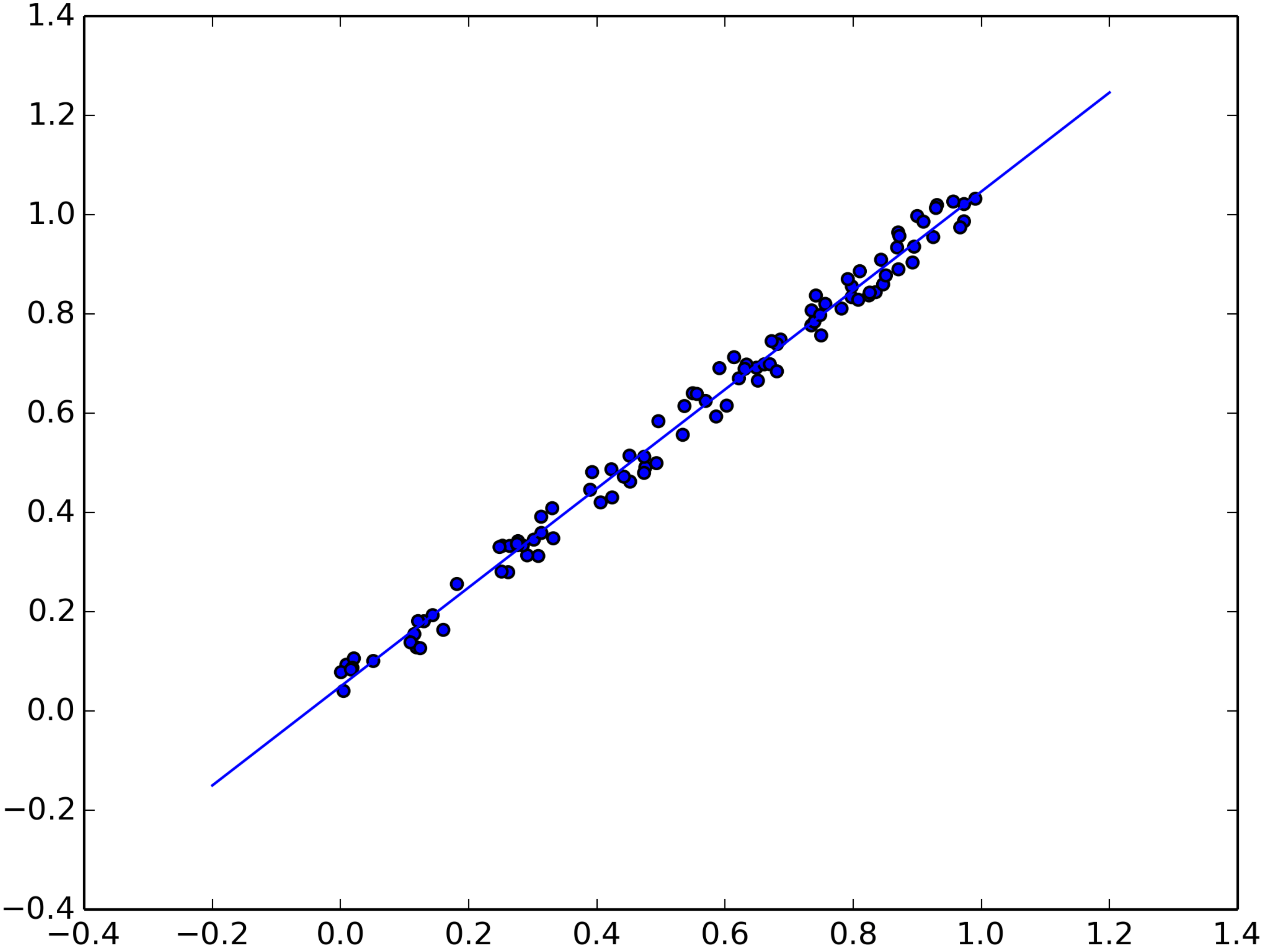
In Machine Learning, the formula is usually written as:

$$f(x) = x\boldsymbol{w} + \boldsymbol{b}$$

Where:

w is the **weight**

b is the **bias**


1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4

The Linear Model

$$f(x) = xw + b$$

➢ We want to find the values $\boldsymbol{w}$ and $\boldsymbol{b}$ so that the result $\boldsymbol{f}(\boldsymbol{x})$ is **as close as possible to the observations**

- Data that can be classified using a linear model is called **linearly separable**

The Linear Model

$f(x) = xw + b$

Example:

- Imagine that you want to identify the price of apartments around SJSU
- You think that the size of the apartments is a crucial factor
- So, the value $\mathbf{x}$ will represent the size of such apartments

The Linear Model

The formula may become:

$$f(x) = x * 100.08 + 9.98$$

Example:

- Imagine that you want to identify the price apartments around SJSU
- You think that the size of the apartment is a crucial factor
- So, the value **x** will represent the size of such apartments

The Linear Model

Ok, but what if you think that the size is not enough to define the price?
What about how close apartments are to SJSU?

- In this case we have two variables
  - But we still use this formula:

$$f(x) = xw + b$$

The Linear Model

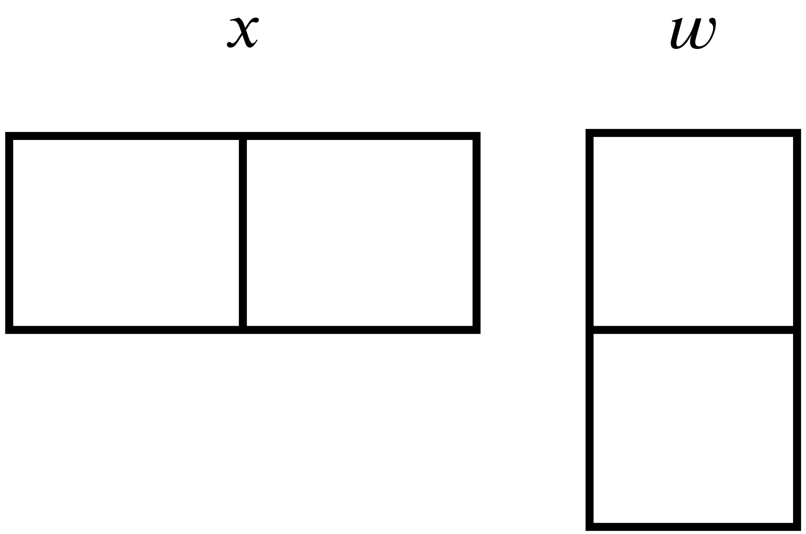
$f(x) = x\boldsymbol{w} + \boldsymbol{b}$

- In fact, $\boldsymbol{w}$ and $\boldsymbol{x}$ are actually **vectors**

  x is **1-by-2 vector**

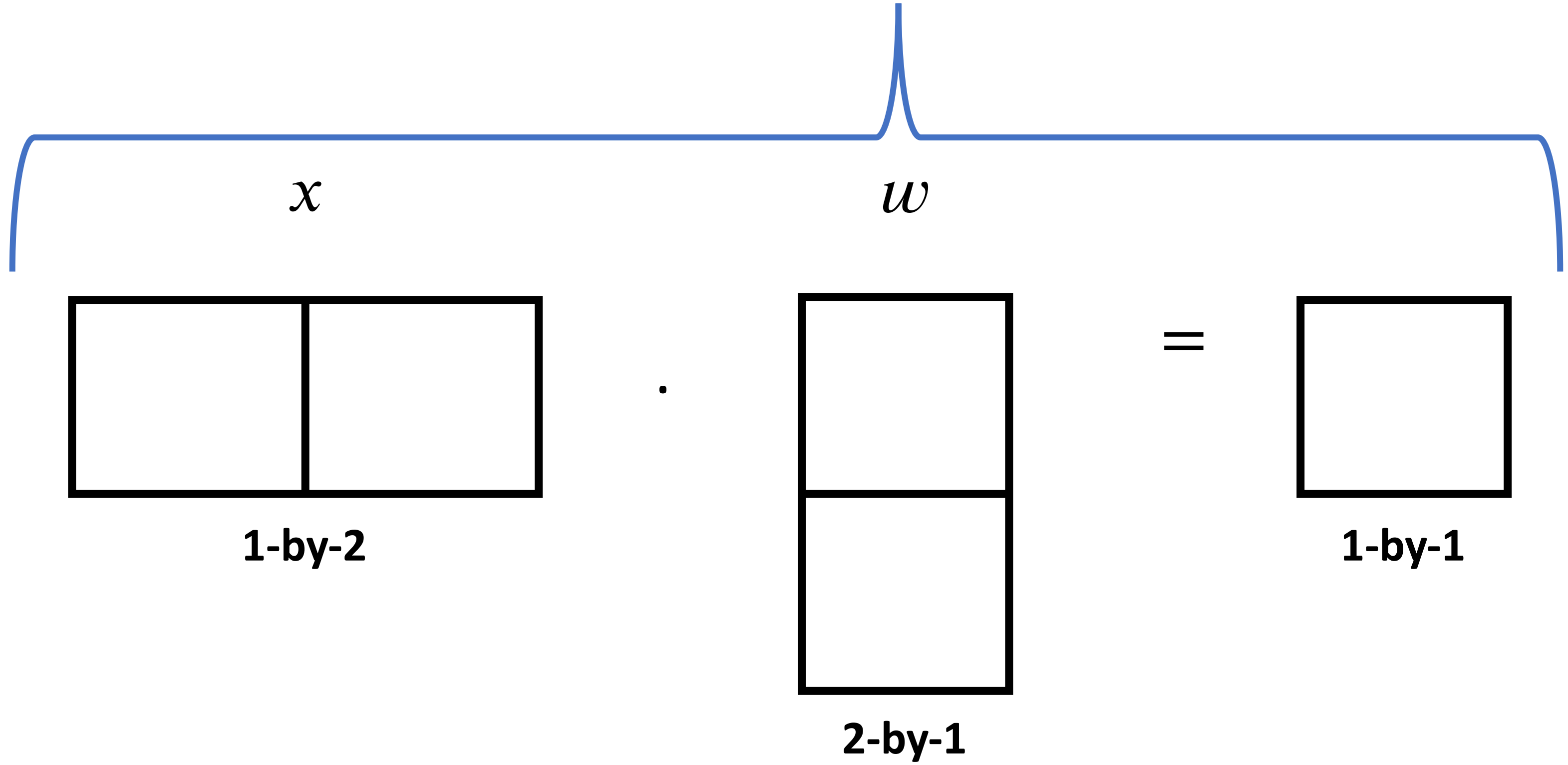
  w is **2-by-1 vector**

  b is **1-by-1 vector**



The Linear Model

$f(x) = x\mathbf{w} + \mathbf{b}$



The Linear Model

$f(x) = x_1 * w_1 + x_2 * w_2 + b$

$$f(x) = \begin{array}{c} x \\ \begin{bmatrix} x_1 & x_2 \end{bmatrix} \end{array} \cdot \begin{array}{c} w \\ \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \end{array} + \begin{bmatrix} b \end{bmatrix}$$

The Linear Model

$f(x) = x_1 * w_1 + x_2 * w_2 + b$

- In a realistic scenario, w_2 would be a negative weight

In fact, we expect the distance to SJSU to have a negative impact when we raise it

Multiple Inputs and Multiple Outputs

Now, you want to check the price for renting AND buying an apartment near SJSU

$y_1 = x_1 * w_{11} + x_2 * w_{21}$

2 linear models

$$y_2 = \quad {}_{12} + x_2 * w_{22} + b_2$$

The number of weights is equal to:

weights = # inputs * # outputs

The number of biases is equal to:

biases = # outputs

Multiple Inputs and Multiple Outputs

$$y_1 = x_1 * w_{11} + x_2 * w_{21} + b_1$$
$$y_2 = x_1 * w_{12} + x_2 * w_{22} + b_2$$

$$\begin{bmatrix} y_1 & y_2 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \cdot \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} + \begin{bmatrix} b_1 & b_2 \end{bmatrix}$$

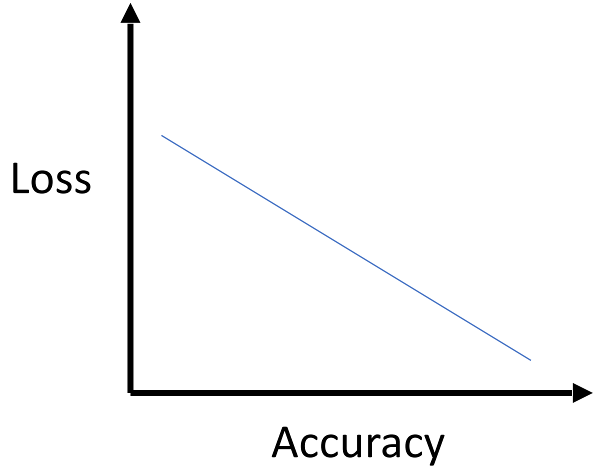
Objective Function

Objective Function

There are two categories of Objective Function: **Loss** and **Reward**

❑ **Loss** (or **Cost**) functions

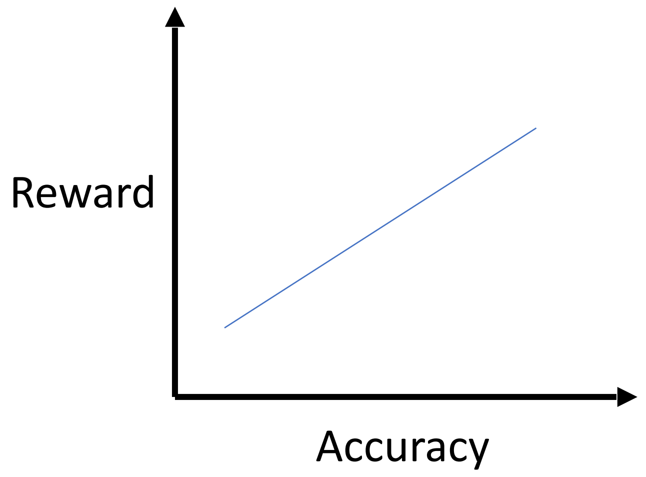
- The lower is the **Loss** function, the higher is the level of **accuracy**
- Usually used in Supervised learning
- It's nothing more than a function that has higher values for worse results and vice versa



Objective Function

There are two categories of Objective Function: **Loss** and **Reward**

- ❑ **Reward** functions
  - Usually used in Reinforcement learning
  - The exact opposite of a Loss function



Loss functions

Loss functions are used in Supervised Learning

- There are two types of Supervised Learning: Regression and Classification

Two very common **Loss functions** are:

➢ **L2-norm** (aka square loss or Least sum of squares)
  - Used for Regression

➢ **Cross-entropy**
  - Used for Classification

- We need to introduce the concept of **Target** (T)

  Ideally, we want: $y = T$

- The targets are the labels used to train the model

$f(x)$	Target (T)	Loss
		470
		50
		4

L2-norm Loss

$$\text{L2-norm} = \sum_i \left(y_i - t_i\right)^2$$

- **Vector norm** (aka **Euclidian distance**) of the outputs and the targets
- The lower the error, the lower the Loss
- Note that the results of this function would always be a number

Cross-entropy Loss

$$\text{Cross-entropy} = L(\mathrm{y}, \mathrm{t}) = -\sum_i t_i ln(y_i)$$

Let's see an example

- Imagine we have three categories: Apple, Pears and Pineapples
- The Target vector $[0\ ,\ 1\ ,\ 0]$ indicates that the input is labeled as Pear

$[0\ ,\ 1\ ,\ 0]$

$[0\ ,\ 0\ ,\ 1]$

Cross-entropy Loss

$$\text{Cross-entropy} = L(\mathrm{y}, \mathrm{t}) = -\sum_i t_i ln(y_i)$$

Y = [0.4, 0.4, 0.2]

$$T = [0\ ,\ 1\ ,\ 0]$$

Output of the model

Y = [0.1, 0.2, 0.7]

$$T = [0\ ,\ 0\ ,\ 1]$$

Cross-entropy Loss

$$\text{Cross-entropy} = L(\mathrm{y}, \mathrm{t}) = -\sum_i t_i ln(y_i)$$

Y = [0.4, 0.4, 0.2] → 40% it's an Apple
40% it's a Pear
20% it's a Pineapple

$$T = [0\ ,\ 1\ ,\ 0]$$

Y = [0.1, 0.2, 0.7] → 10% it's an Apple
20% it's a Pear
70% it's a Pineapple

$$T = [0\ ,\ 0\ ,\ 1]$$

Cross-entropy Loss

Cross-entropy $= L(\mathrm{y}, \mathrm{t}) = -\sum_i t_i ln(y_i)$

Y = [0.4, 0.4, 0.2] $\rightarrow -0 * \ln(0.4) - 1 * \ln(0.4) - 0 * \ln(0.2) = 0.9$

$T = [0\ ,\ 1\ ,\ 0]$

Y = [0.1, 0.2, 0.7] $\rightarrow -0 * \ln(0.1) - 0 * \ln(0.2) - 1 * \ln(0.7) = 0.3$

$T = [0\ ,\ 0\ ,\ 1]$

Cross-entropy Loss

$$\text{Cross-entropy} = L(\mathrm{y}, \mathrm{t}) = -\sum_i t_i ln(y_i)$$

Y = [0.4, 0.4, 0.2]

$T = [0\ ,\ 1\ ,\ 0]$

$L(y, t) = 0.92$

Y = [0.1, 0.2, 0.7]

$T = [0\ ,\ 0\ ,\ 1]$

$L(y, t) = 0.36$

That's the best!

Optimization Algorithm

Gradient Descent

The last piece of the puzzle is the Optimization Algorithm

- The easiest Optimization Algorithm is the **Gradient Descent**

- **Gradient Descent** is a form of trial-and-error in searching for a local minimum
  - Computers like a lot trial-and-error approaches

- Using a “well-designed” **update rule**, each trial is better than the previous one
  - It reaches the minimum faster, without oscillation

Gradient Descent

Let's see a simple example:

$f(x) = 5x^2 + 3x - 4$

We want to find the MIN of the function $f(x)$ using **Gradient Descent**

Step 1: Derivative of the function

$$f'(x) = 10x + 3$$

Step 2: Choose an arbitrary number x_0

$$x_0 = 4$$

Step 1: Derivative of the function

$$f'(x) = 10x + 3$$

Step 2: Choose an arbitrary number x_0

$$x_0 = 4$$

Step 3: Choose a value x_1 following this update rule:

$$x_{i+1} = x_i - \eta f'(x_i)$$

$$x_1 = x_0 - \eta[10 * x_0 + 3]$$

$$x_1 = 4 - \eta\left[10 * 4 + 4\right] = 4 - 43\eta$$

Step 1: Derivative of the function

$$f'(x) = 10x + 3$$

Step 2: Choose an arbitrary number x_0

$$x_0 = 4$$

Step 3: Choose a value x_1 following this rule:

$$x_1 = 4 - 43\eta$$

- η is the **learning rate** at which the ML algorithm forgets all beliefs for new ones

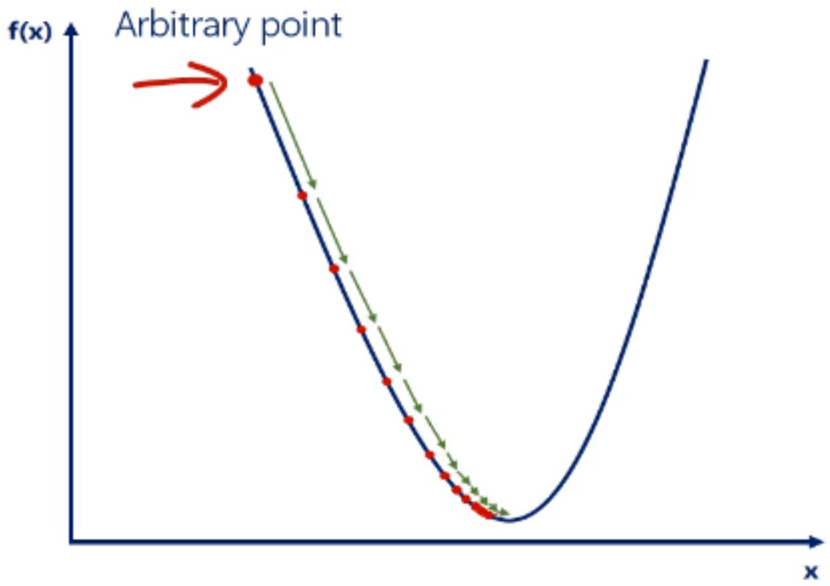
- Step 4: we keep searching for values $(x_2, x_3, \ldots)$ until we reach a point were **the values stop updating**

  We basically reached the minimum of the function, that is, where the derivative is equal to 0

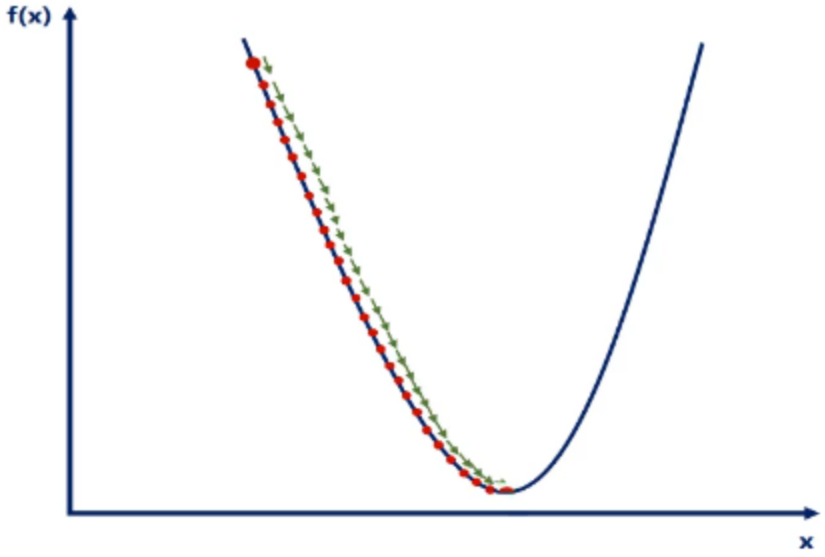
- If the derivative is 0, then the update rule: $x_{i+1} = x_i - \eta f'(x_i)$ becomes:

$$x_{i+1} = x_i - 0$$

That is, it stops updating



$\eta = 0.1$



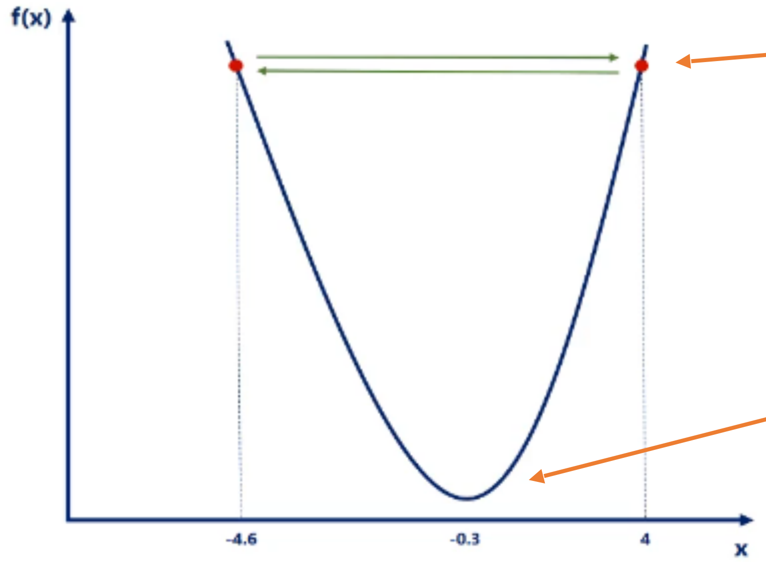
$\eta = 0.001$

- The speed of minimization depends on η

Oscillation

- For some η values, the result of the update rule could oscillate eternally between two (or more) values.

➢This is called **Oscillation**



We are stuck here!

But the Minimum is there…

η = 0.2

Which η should we chose?

To choose the right $\boldsymbol{\eta}$ value, follow these two simple rules:

- It should be **high enough**, so that the closest minimum is reached in a rational amount of time
- It should be **low enough**, so that we don’t oscillate *in aeternum* around the minimum

➢In other words, experiment with reasonable values…

How do I know when to stop?

- What if the results of the update rule are smaller and smaller and seem to continue forever?

- A rule of thumb is to stop when:

$$x_{i+1} - x_i = 0.001$$

This can be implemented easily adding a “breaking condition” in the loop

N-dimensional Gradient Descent

Up to now, we have:

- Data:

We select some data

- Model:

$$xw + b = y \text{ [Linear Model]}$$

- Optimization Function:

We'll see why over 2 soon

$$L(y,t) = \frac{L2 - norm}{2} = \frac{\sum_i (y_i - t_i)^2}{2}$$

N-dimensional Gradient Descent

- The update rule:

$$x_{i+1} = x_i - \eta f'(x_i)$$

- Becomes:

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \eta \boldsymbol{\nabla}_{\boldsymbol{w}} L(\boldsymbol{y}, \boldsymbol{t})$$
$$\boldsymbol{b}_{i+1} = \boldsymbol{b}_i - \eta \boldsymbol{\nabla}_{\boldsymbol{b}} L(\boldsymbol{y}, \boldsymbol{t})$$

$\boldsymbol{\nabla}_{\boldsymbol{w}} L(\boldsymbol{y}, \boldsymbol{t})$ is the gradient of the Loss function with respect to w_i for the weights
$\boldsymbol{\nabla}_{\boldsymbol{b}} L(\boldsymbol{y}, \boldsymbol{t})$ is the gradient of the Loss function with respect to b_i for the biases

N-dimensional Gradient Descent

- We update rule for N-dimensions:

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \eta \boldsymbol{\nabla}_{\boldsymbol{w}} L(\boldsymbol{y}, \boldsymbol{t})$$
$$\boldsymbol{b}_{i+1} = \boldsymbol{b}_i - \eta \boldsymbol{\nabla}_{\boldsymbol{b}} L(\boldsymbol{y}, \boldsymbol{t})$$

- To minimize the Loss function, we need to optimize regarding $\boldsymbol{w}$ and $\boldsymbol{b}$

  It's still a game of tuning the weights and the biases

Optimization

- We update rule for N-dimensions:

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \eta \nabla_w L(\boldsymbol{y}, \boldsymbol{t})$$
$$\boldsymbol{b}_{i+1} = \boldsymbol{b}_i - \eta \nabla_b L(\boldsymbol{y}, \boldsymbol{t})$$

$$\nabla_w L(\boldsymbol{y}, \boldsymbol{t}) = \sum_i \nabla_w \frac{1}{2}(\boldsymbol{y}_i - \boldsymbol{t}_i)^2 = \sum_i \boldsymbol{x}_i(\boldsymbol{y}_i - \boldsymbol{t}_i) = \sum_i \boldsymbol{x}_i \boldsymbol{\delta}_i$$

Recall that:

$$L(y,t) = \frac{\sum_i (y_i - t_i)^2}{2}$$

$y_i = x_i w + b$
Where w and x are matrices

δ is the default math symbol used to measure differences

$$
\begin{aligned}
\nabla_{\mathbf{w}} L = \nabla_{\mathbf{w}} \frac{1}{2} \sum_{i} (y_i - t_i)^2 &= \\
= \nabla_{\mathbf{w}} \frac{1}{2} \sum_{i} \left((\boldsymbol{x}_i \boldsymbol{w} + b) - t_i \right)^2 &= \\
= \sum_{i} \nabla_{\mathbf{w}} \frac{1}{2} (\boldsymbol{x}_i \boldsymbol{w} + b - t_i)^2 &= \\
= \sum_{i} \boldsymbol{x}_i (\boldsymbol{x}_i \boldsymbol{w} + b - t_i) &= \\
= \sum_{i} \boldsymbol{x}_i (y\mathrm{i} - t_i) &\equiv \\
\equiv \sum_{i} \boldsymbol{x}_i \delta_i &
\end{aligned}
$$

Optimization

- The update rule for N-dimensions:

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \eta \boldsymbol{\nabla}_w L(\boldsymbol{y}, \boldsymbol{t})$$
$$\boldsymbol{b}_{i+1} = \boldsymbol{b}_i - \eta \boldsymbol{\nabla}_b L(\boldsymbol{y}, \boldsymbol{t})$$

$$\boldsymbol{\nabla}_w L(\boldsymbol{y}, \boldsymbol{t}) = \sum_i \boldsymbol{x}_i \delta_i$$

$$\boldsymbol{\nabla}_b L(\boldsymbol{y}, \boldsymbol{t}) = \sum_i \delta_i$$

Finally, we have:

- Data: We select some data
- Model: $xw + b = y$ [Linear Model]
- Optimization Function:

$$L(y,t) = \frac{L2 - norm}{2} = \frac{\sum_i (y_i - t_i)^2}{2}$$

- Optimization Algorithm:

$$w_{i+1} = w_i - \eta \nabla_w L(y,t)$$
$$b_{i+1} = b_i - \eta \nabla_b L(y,t)$$