Machine Learning

- Gradient Descent Algorithm
- Linear Regression
- Non-Linear Regression
- Logistic Regression
- Decision Trees
  - Regression Trees
  - Classification Trees
- Clustering Algorithms
  - K-Means
  - Hierarchical clustering
  - DB-Scan
  - Mean Shift
  - GMM
- Support Vector Machine

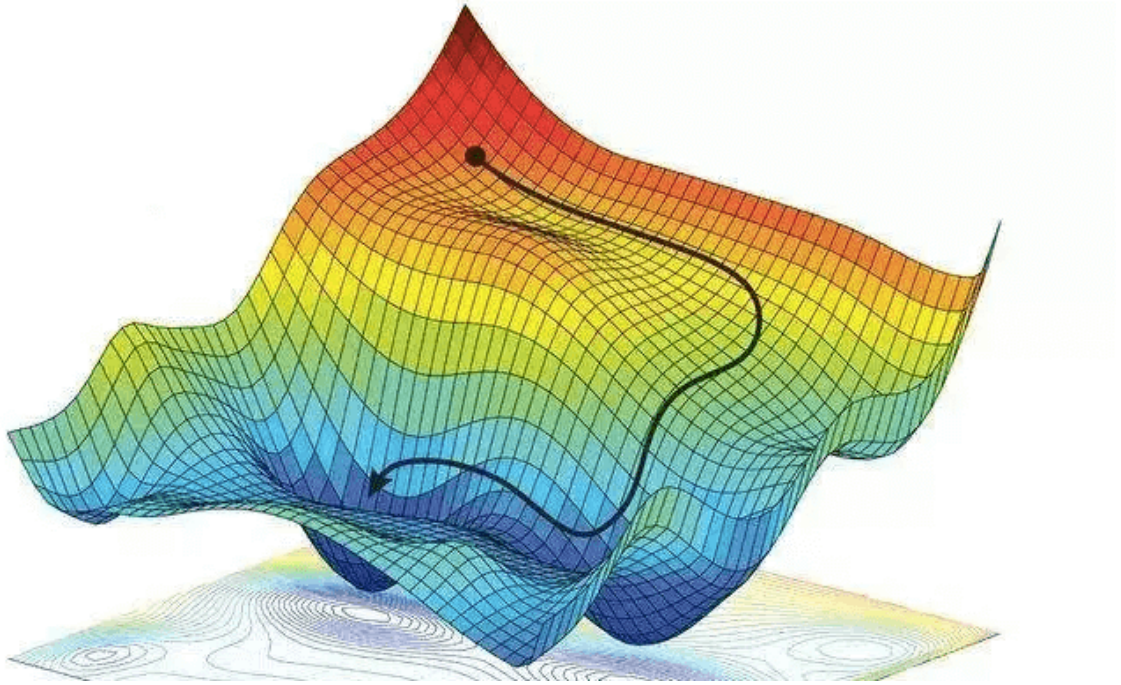
Deep Learning

- MLP
- CNN

Datasets

- Breast Cancer Wisconsin
- MIMIC-III
- Framingham Heart Study
- Alzheimer’s Disease Neuroimaging Initiative
- Drug discovery
- Microbiome

C1 - Gradient Descent Algorithm

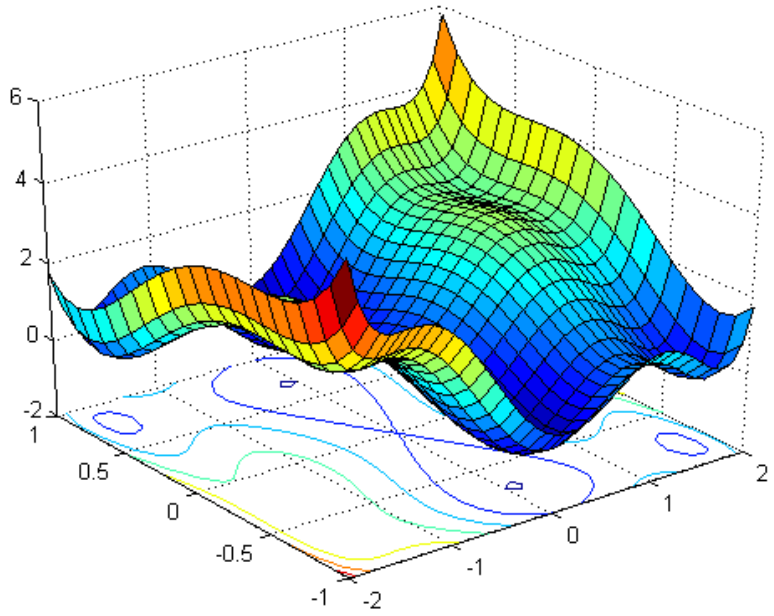


Vanilla Gradient Descent

First-order optimization algorithm to find a local minimum/maximum of a given function.

Gradient descent algorithm does not work for all functions.
There are two specific requirements. A functions must be:

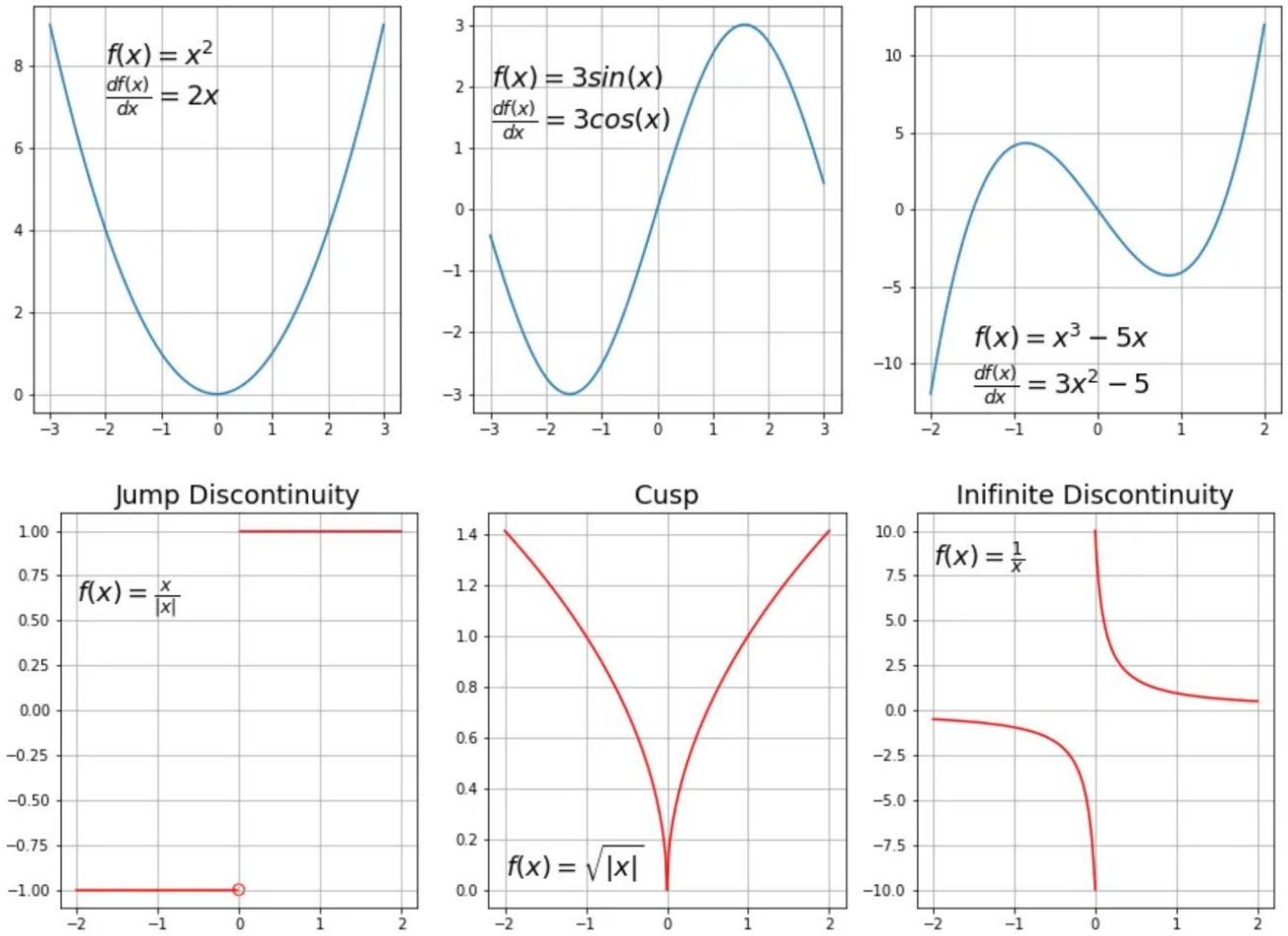
- differentiable
- convex



Vanilla Gradient Descent

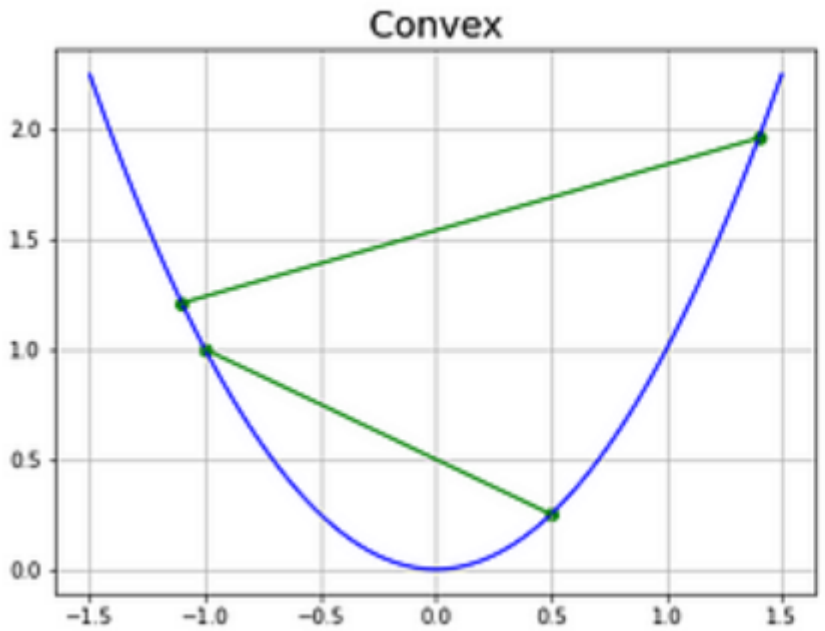
Differentiable:

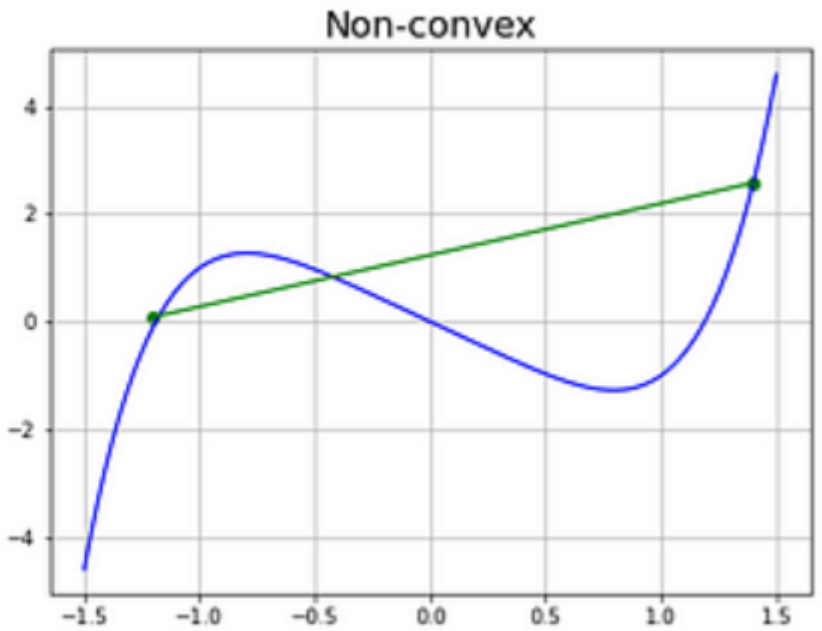
A function is differentiable it has a derivative for each point in its domain



Vanilla Gradient Descent

Convex:
A real function is convex on an interval (a,b), if the chord joining any two points in the function lies above the function.





Gradient

Vector whose components are the partial derivatives of f at p
Given $p = (x_1,\ x_2,\ \ldots,\ x_n)$

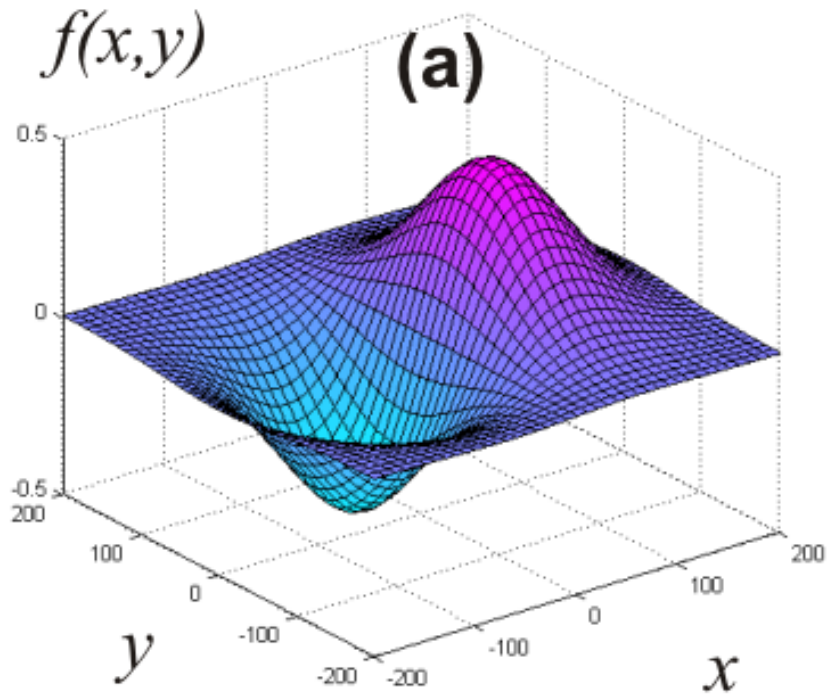
$$\nabla f(p) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(p) \\ \vdots \\ \frac{\partial f}{\partial x_n}(p) \end{bmatrix}.$$

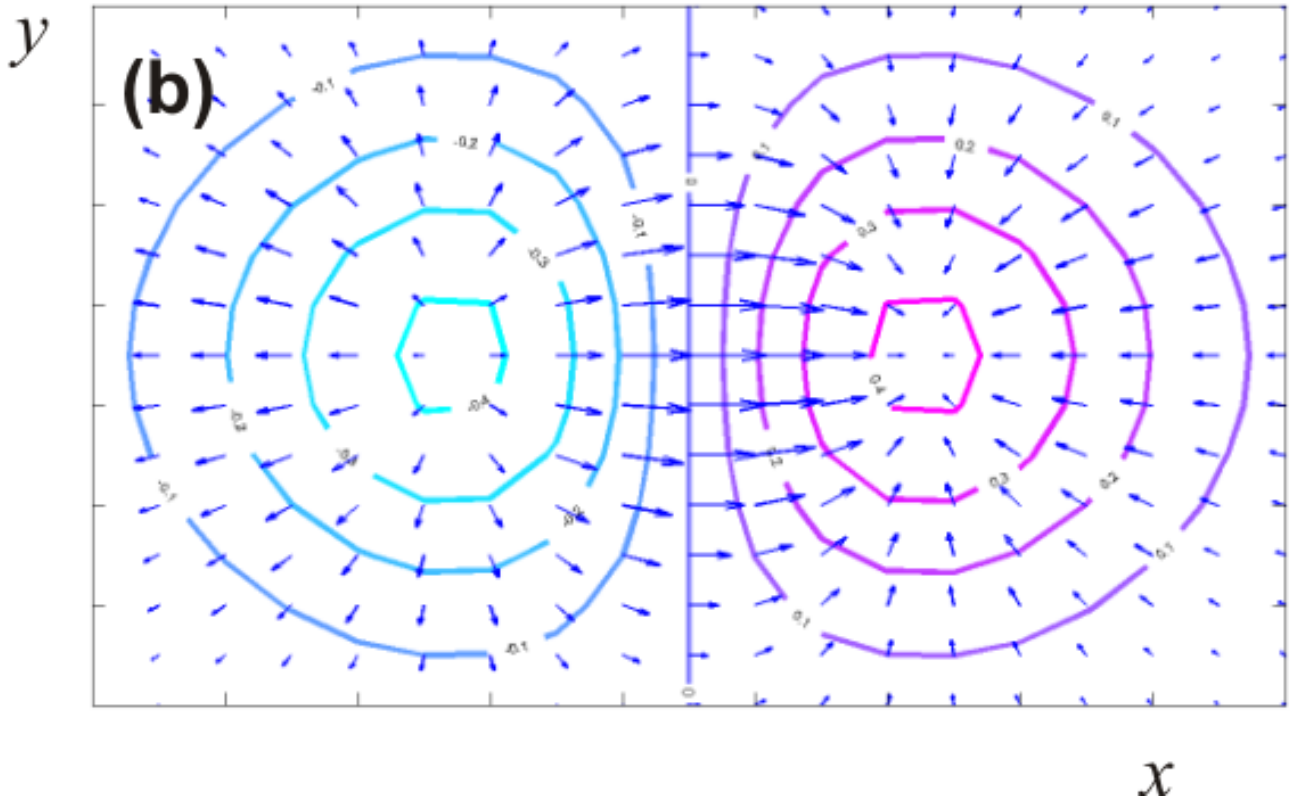
The gradient vector is used to determine the direction in which a function grows fastest at a given point, and it is also used in function optimization to find minimums and maximums.

If the gradient of a function at a point is zero, then that point may be a local minimum or maximum of the function, or an inflection point.

Gradient

$$\nabla f(p) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(p) \\ \vdots \\ \frac{\partial f}{\partial x_n}(p) \end{bmatrix}.$$





Gradient descent algorithm

Gradient Descent Algorithm iteratively calculates the next point using gradient at the current position scales it (by a learning rate) and subtracts obtained value from the current position (makes a step).

$$p_{n+1} = p_n - \eta \nabla f(p_n)$$

η is also called learning rate, which scales the gradient and thus controls the step size.
Learning rate have a strong influence on model performance.

- The smaller learning rate the longer GD converges, or may reach maximum iteration before reaching the optimum point
- If learning rate is too big the algorithm may not converge to the optimal point (jump around) or even to diverge completely.

Gradient descent algorithm

In summary, Gradient Descent method's steps are:

1. choose a starting point (initialisation)
2. calculate gradient at this point
3. make a scaled step in the opposite direction to the gradient (objective: minimise)
4. repeat points 2 and 3 until one of the criteria is met:

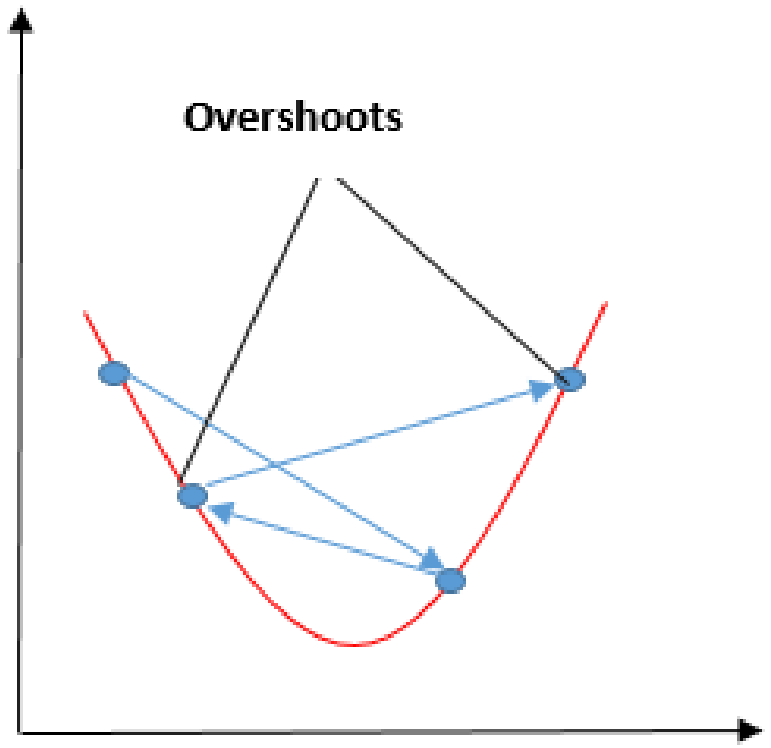
- maximum number of iterations reached
- step size is smaller than the tolerance (due to scaling or a small gradient).

$$\text{Repeat until convergence } \{$$

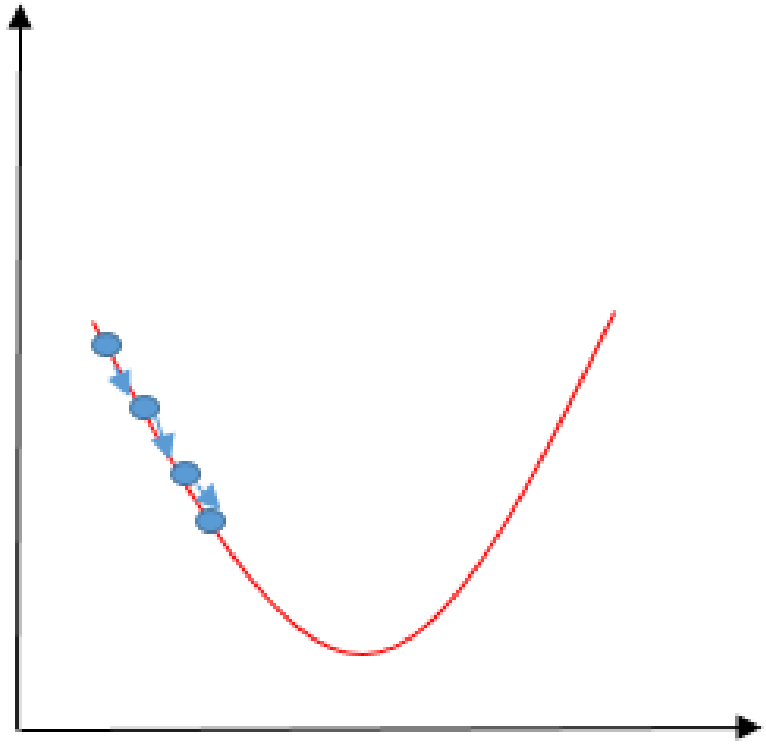
$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

$$\}$$

```python
def gradient_descent(start, gradient, learn_rate, max_iter, tol=0.01):
    """Gradient descent algorithm.

    args:
        start: starting point (maybe random initialisation)
        gradient: function to calculate gradient
        learn_rate: learning rate | scaling factor for step sizes
        max_iter: maximum number of iterations
        tol: tolerance for stopping criterion

    returns:
        steps: list of all steps
        x: final value
    """
    steps = [start]
    x = start

    for _ in range(max_iter):
        diff = learn_rate * gradient(x)
        if np.abs(diff) < tol:
            break
        x = x - diff
        steps.append(x)
    return steps, x
```

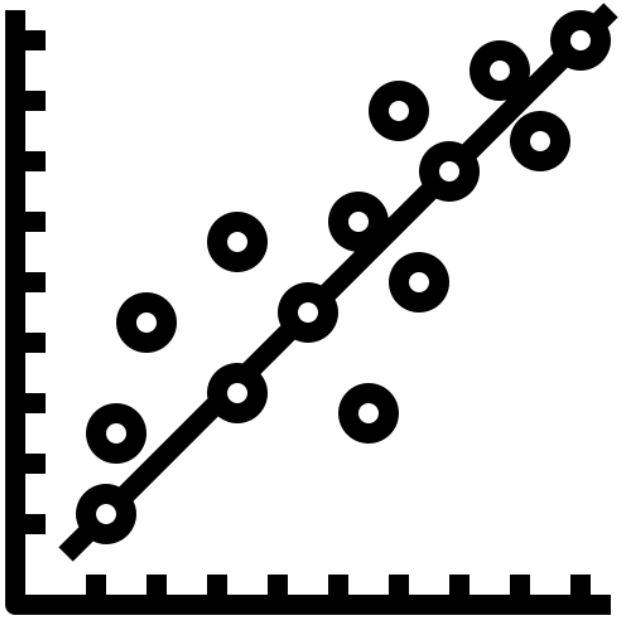


High learning rate



Low learning rate

C1 - Linear Regression

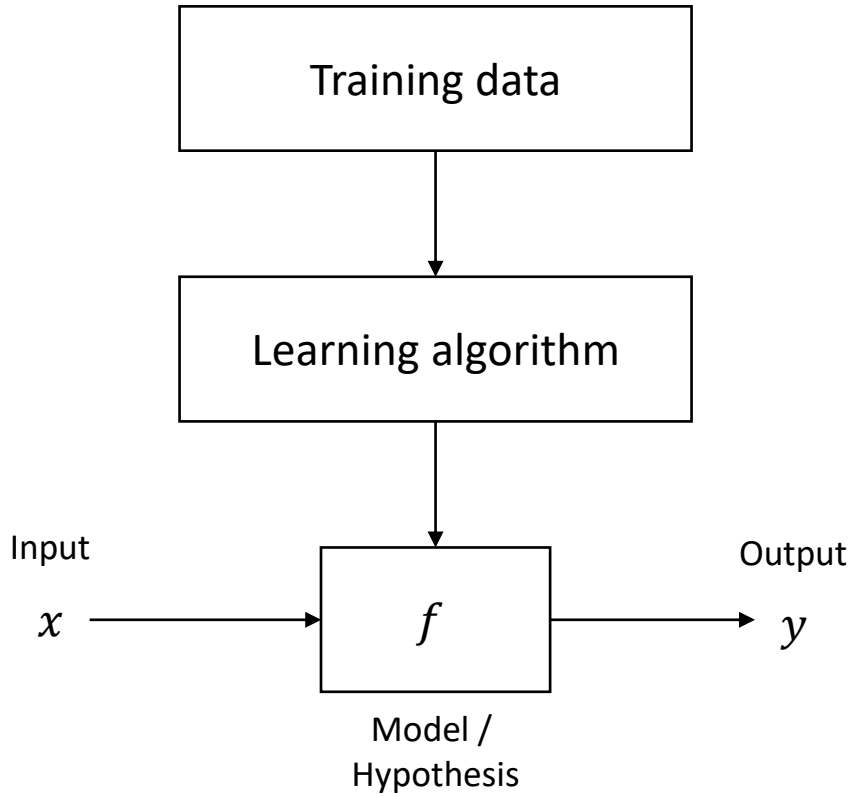


Model/Hypothesis

A machine learning hypothesis is a candidate model that approximates a target function for mapping inputs to outputs

For univariable linear regression:

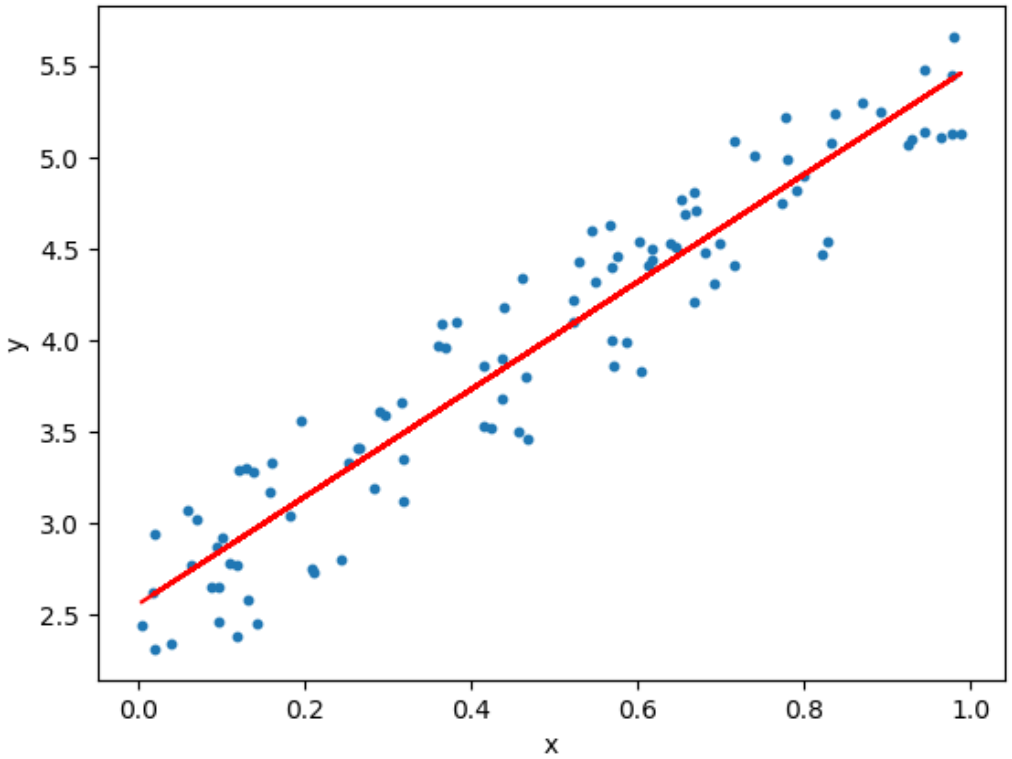
$$h(x) = x * w + b$$



Univariable Linear Regression

is an attempt to model the relationship between variables by fitting a linear equation to observed data.

GOAL: Find the best line that fits the data.



Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

Parameters: θ_0, θ_1

Loss function (MSE)

Mean

Error Squared

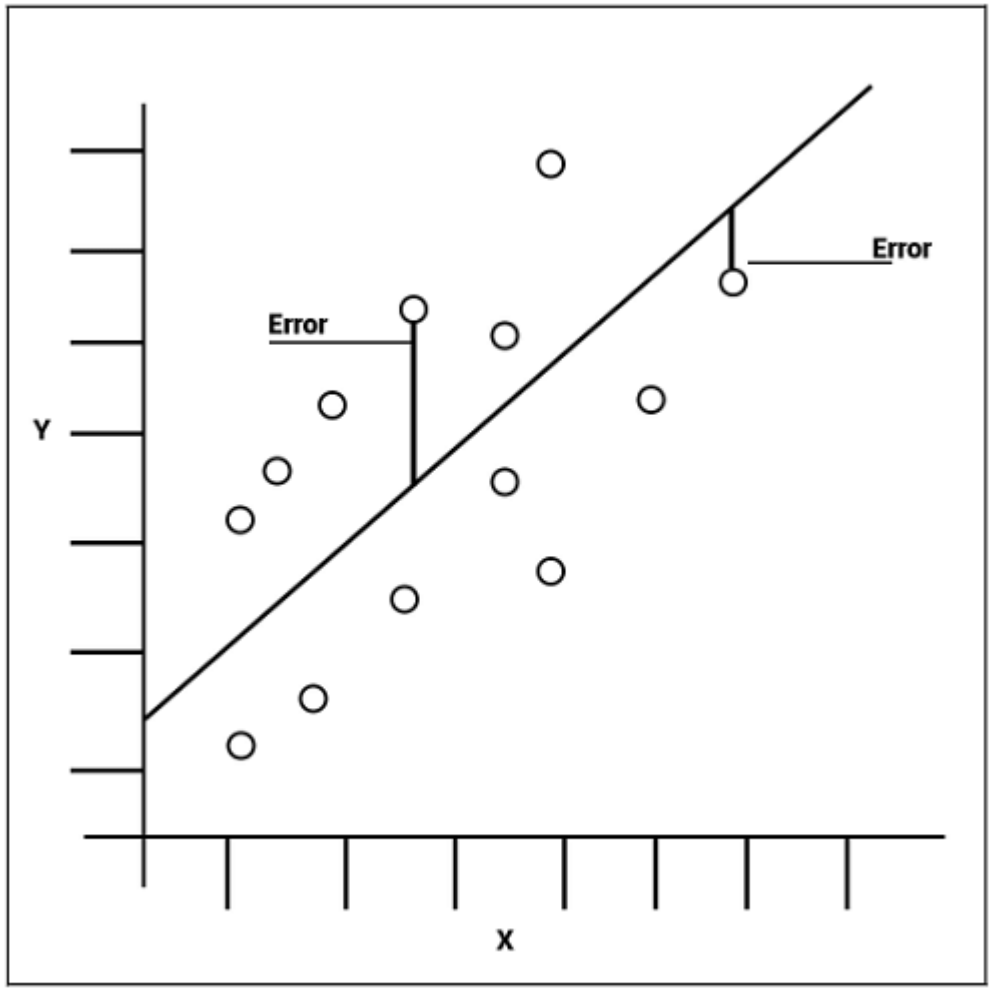
$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

MSE = mean squared error

n = number of data points

Y_i = observed values

$\hat{Y}_i$ = predicted values



Univariable Linear Regression

Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

Parameters: θ_0, θ_1

Cost Function: $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$

Goal: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

Partial derivatives / Gradient Vector

$$J = \frac{1}{n}\sum_{i=1}^{n}(a_0 + a_1 \cdot x_i - y_i)^2$$

$$\frac{\partial J}{\partial a_0} = \frac{2}{n}\sum_{i=1}^{n}(a_0 + a_1 \cdot x_i - y_i)$$

$$\frac{\partial J}{\partial a_1} = \frac{2}{n}\sum_{i=1}^{n}(a_0 + a_1 \cdot x_i - y_i) \cdot x_i$$

$$\frac{\partial J}{\partial a_0} = \frac{2}{n}\sum_{i=1}^{n}(pred_i - y_i)$$

$$\frac{\partial J}{\partial a_1} = \frac{2}{n}\sum_{i=1}^{n}(pred_i - y_i) \cdot x_i$$

Update parameters with GD algorithm

$$\frac{\partial J}{\partial a_0} = \frac{2}{n}\sum_{i=1}^{n}(pred_i - y_i)$$

$$a_0 = a_0 - \alpha \cdot \frac{2}{n}\sum_{i=1}^{n}(pred_i - y_i)$$

$$\frac{\partial J}{\partial a_1} = \frac{2}{n}\sum_{i=1}^{n}(pred_i - y_i) \cdot x_i$$

$$a_1 = a_1 - \alpha \cdot \frac{2}{n}\sum_{i=1}^{n}(pred_i - y_i) \cdot x_i$$

Pseudocode

1. Initialize the parameters.
2. Predict the value of a dependent variable by given an independent variable.
3. Calculate the error in prediction for all data points.
4. Calculate partial derivative w.r.t a0 and a1.
5. Calculate the cost for each number and add them.
6. Update the values of a0 and a1.

Code