

MACHINE LEARNING

SUPERVISED

CHAP 8

REGRESSION: OLS & Gradient Descent

Jean Tabet

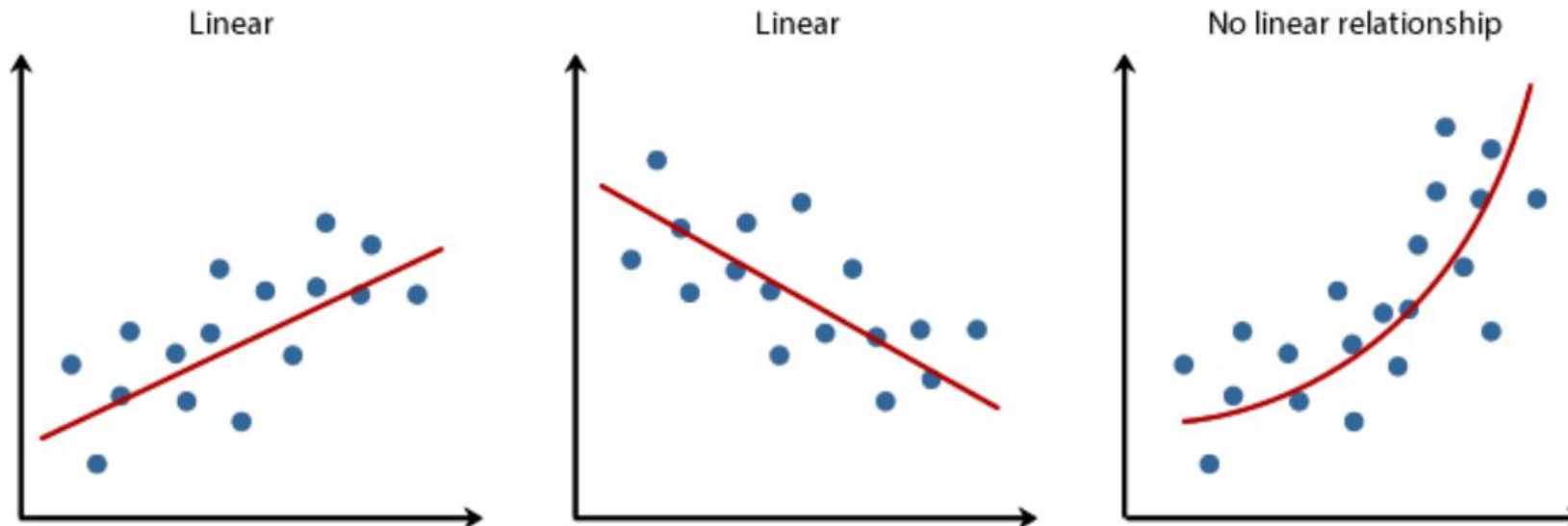
j.tabet@ul.edu.lb

➔ REGRESSION

- What is linear regression?
- The least-squares (closed-form) solution
 - Simple linear regression
 - Multivariate linear regression
 - Polynomial regression
- Solve linear regression by optimization
 - Gradient descent

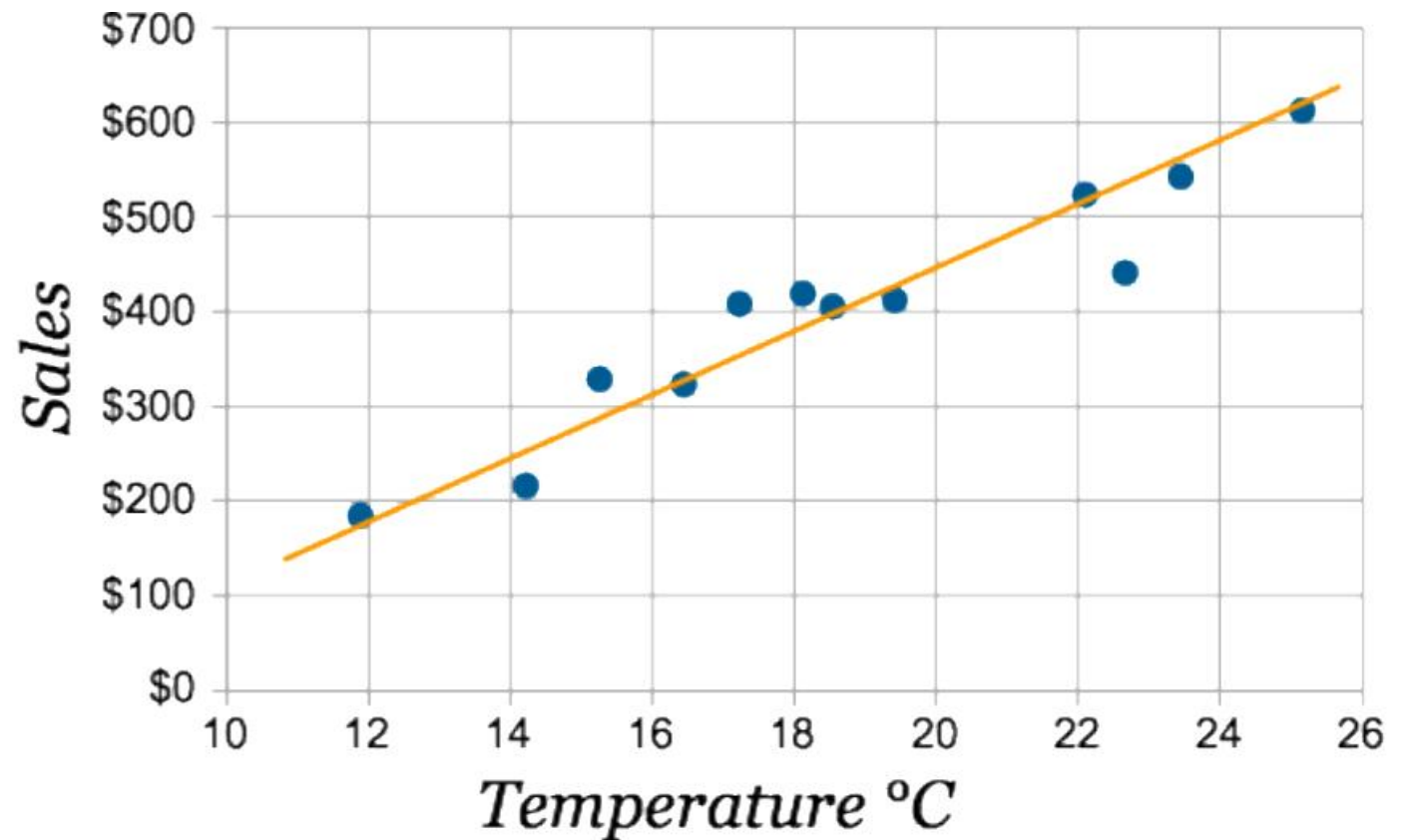
➔ WHAT IS LINEAR REGRESSION ?

- Given a set of observed values of the independent (input) variables and the corresponding values of the dependent (output) variable, determine a linear relation between the independent variable(s) and a continuous output variable.
- **Note:** to improve linear regression apply scaling on numerical data.



➔ LINEAR REGRESSION

- **Example** about simple linear regression



➔ LINEAR REGRESSION

- **Example** about multivariate linear regression



Prices of used cars: example data for regression

Price (US\$)	Age (years)	Distance (km)	Weight (pounds)
13500	23	46986	1165
13750	23	72937	1165
13950	24	41711	1165
14950	26	48000	1165
13750	30	38500	1170
12950	32	61000	1170
16900	27	94612	1245
18600	30	75889	1245
21500	27	19700	1185
12950	23	71138	1105

→ LINEAR REGRESSION

- General approach

- Regression function: $y = f(x, \theta)$

- Optimize θ such that the approximation error is minimized:

$$E = \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

Example

$$\text{Price} = a_0 + a_1 \cdot \text{Age} + a_2 \cdot \text{Distance} + a_3 \cdot \text{Weight}$$

$$x = \{\text{Age}, \text{Distance}, \text{Weight}\}$$

$$\theta = \{a_0, a_1, a_2, a_3\}$$

Prices of used cars: example data for regression

Price (US\$)	Age (years)	Distance (km)	Weight (pounds)
13500	23	46986	1165
13750	23	72937	1165
13950	24	41711	1165
14950	26	48000	1165
13750	30	38500	1170
12950	32	61000	1170
16900	27	94612	1245
18600	30	75889	1245
21500	27	19700	1185
12950	23	71138	1105

➔ DIFFERENT LINEAR REGRESSION MODELS

- Simple linear regression
 - Only one continuous independent variable

$$y = a + bx$$

- Multivariate linear regression
 - More than one independent variables

$$y = a_0 + a_1x_1 + \dots + a_nx_n$$

- Polynomial regression
 - Only one continuous independent variable

$$y = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$$

➔ SIMPLE LINEAR REGRESSION (OLS)

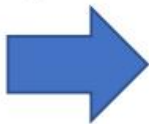
- Ordinary least squares

$$y = \alpha + \beta x$$

x	x_1	x_2	$\dots$	x_n
y	y_1	y_2	$\dots$	y_n

- Objective function

$$\begin{aligned} E &= \sum_{i=1}^n (y_i - \hat{y}_i)^2 \\ &= \sum_{i=1}^n [y_i - (\alpha + \beta x_i)]^2 \end{aligned}$$

$$\begin{aligned} \frac{\partial E}{\partial \alpha} &= 0 & \sum_{i=1}^n y_i &= n\alpha + \beta \sum_{i=1}^n x_i \\ \frac{\partial E}{\partial \beta} &= 0 & \sum_{i=1}^n x_i y_i &= \alpha \sum_{i=1}^n x_i + \beta \sum_{i=1}^n x_i^2 \end{aligned}$$


➔ SIMPLE LINEAR REGRESSION (OLS)

- Ordinary least squares

$$y = \alpha + \beta x$$

Final solution:

$$\beta = \frac{\text{Cov}(x, y)}{\text{Var}(x)}$$

$$\alpha = \bar{y} - \beta \bar{x}$$

$$\bar{x} = \frac{1}{n} \sum x_i$$

$$\bar{y} = \frac{1}{n} \sum y_i$$

$$\text{Var}(x) = \frac{1}{n-1} \sum (x_i - \bar{x})^2$$

$$\text{Cov}(x, y) = \frac{1}{n-1} \sum (x_i - \bar{x})(y_i - \bar{y})$$

→ SIMPLE LINEAR REGRESSION (OLS)

- Example

$$y = \alpha + \beta x$$

$$n = 5$$

$$\bar{x} = \frac{1}{5}(1.0 + 2.0 + 3.0 + 4.0 + 5.0) = 3.0$$

$$\bar{y} = \frac{1}{5}(1.00 + 2.00 + 1.30 + 3.75 + 2.25) = 2.06$$

$$\text{Cov}(x, y) = \frac{1}{4}[(1.0 - 3.0)(1.00 - 2.06) + \cdots + (5.0 - 3.0)(2.25 - 2.06)] = 1.0625$$

$$r(x) = \frac{1}{4}[(1.0 - 3.0)^2 + \cdots + (5.0 - 3.0)^2] = 2.5$$

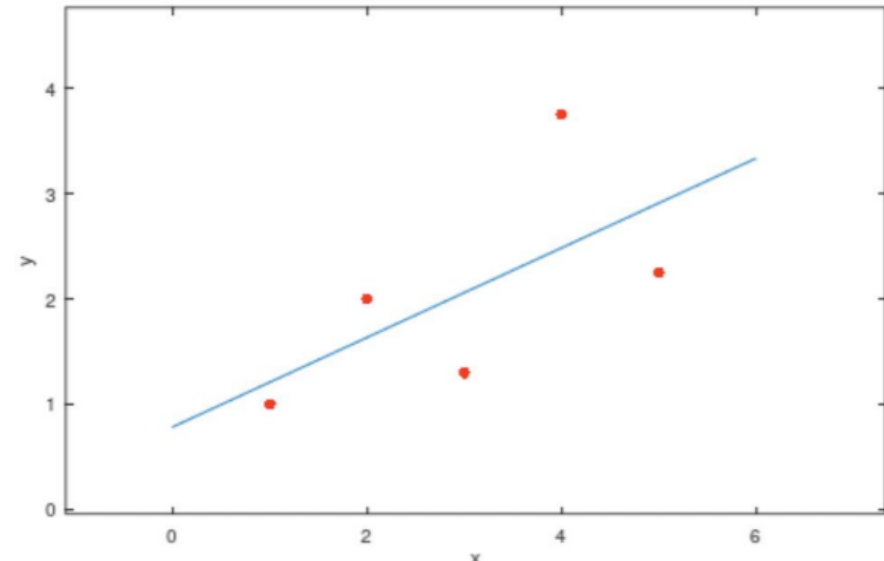
$$b = \frac{1.0625}{2.5} = 0.425$$

$$a = 2.06 - 0.425 \times 3.0 = 0.785$$



$$y = 0.785 + 0.425x$$

x	1.0	2.0	3.0	4.0	5.0
y	1.00	2.00	1.30	3.75	2.25



➔ SIMPLE LINEAR REGRESSION (OLS)

```
from sklearn.linear_model import LinearRegression
import numpy as np
```

```
# Data
```

```
X = np.array([[1], [2], [3], [4], [5]])
```

```
y = np.array([2, 4, 5, 4, 5])
```

```
# Model
```

```
model = LinearRegression()
```

```
model.fit(X, y)
```

```
# Results
```

```
print("Intercept:", model.intercept_)
```

```
print("Slope:", model.coef_)
```

```
# Prediction
```

```
y_pred = model.predict(X)
```

Regularization terms

$$RIDGE : \quad RSS + \lambda \sum_{i=1}^n \beta_i^2$$

$$LASSO : \quad RSS + \lambda \sum_{i=1}^n |\beta_i|$$

→ POLYNOMIAL REGRESSION (OLS)

- Model

$$y = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \cdots + \alpha_k x^k$$

x	x_1	x_2	$\dots$	x_n
y	y_1	y_2	$\dots$	y_n

- Ordinary least squares

- Objective

$$E = \sum_{i=1}^n [y_i - (\alpha_0 + \alpha_1 x_i + \alpha_2 x_i^2 + \cdots + \alpha_k x_i^k)]^2$$

- Solution can be obtained by solving

$$\frac{\partial E}{\partial \alpha_i} = 0, \quad \forall i = 0, 1, \dots, k.$$

➔ POLYNOMIAL REGRESSION (OLS)

- Ordinary least squares

$$y = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \cdots + \alpha_k x^k$$

$$\frac{\partial E}{\partial \alpha_i} = 0, \quad \forall i = 0, 1, \dots, k.$$



$$\begin{aligned}\sum y_i &= \alpha_0 n + \alpha_1 \left(\sum x_i \right) + \cdots + \alpha_k \left(\sum x_i^k \right) \\ \sum y_i x_i &= \alpha_0 \left(\sum x_i \right) + \alpha_1 \left(\sum x_i^2 \right) + \cdots + \alpha_k \left(\sum x_i^{k+1} \right) \\ \sum y_i x_i^2 &= \alpha_0 \left(\sum x_i^2 \right) + \alpha_1 \left(\sum x_i^3 \right) + \cdots + \alpha_k \left(\sum x_i^{k+2} \right) \\ &\vdots \\ \sum y_i x_i^k &= \alpha_0 \left(\sum x_i^k \right) + \alpha_1 \left(\sum x_i^{k+1} \right) + \cdots + \alpha_k \left(\sum x_i^{2k} \right)\end{aligned}$$

➔ POLYNOMIAL REGRESSION (OLS)

- Ordinary least squares

$$y = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \cdots + \alpha_k x^k$$

$$\frac{\partial E}{\partial \alpha_i} = 0, \quad \forall i = 0, 1, \dots, k.$$



$$\begin{aligned} \sum y_i &= \alpha_0 n + \alpha_1 \left(\sum x_i \right) + \cdots + \alpha_k \left(\sum x_i^k \right) \\ \sum y_i x_i &= \alpha_0 \left(\sum x_i \right) + \alpha_1 \left(\sum x_i^2 \right) + \cdots + \alpha_k \left(\sum x_i^{k+1} \right) \\ \sum y_i x_i^2 &= \alpha_0 \left(\sum x_i^2 \right) + \alpha_1 \left(\sum x_i^3 \right) + \cdots + \alpha_k \left(\sum x_i^{k+2} \right) \\ &\vdots \\ \sum y_i x_i^k &= \alpha_0 \left(\sum x_i^k \right) + \alpha_1 \left(\sum x_i^{k+1} \right) + \cdots + \alpha_k \left(\sum x_i^{2k} \right) \end{aligned}$$



$$\vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \quad D = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^k \\ 1 & x_2 & x_2^2 & \cdots & x_2^k \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^k \end{bmatrix}, \quad \text{and } \vec{\alpha} = \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_k \end{bmatrix}$$

Each row = one data point
Each column = a term in the model

$$\vec{\alpha} = (D^T D)^{-1} D^T \vec{y}$$



$$\vec{y} = D \vec{\alpha}$$

“Each output y_i is computed using the polynomial model”

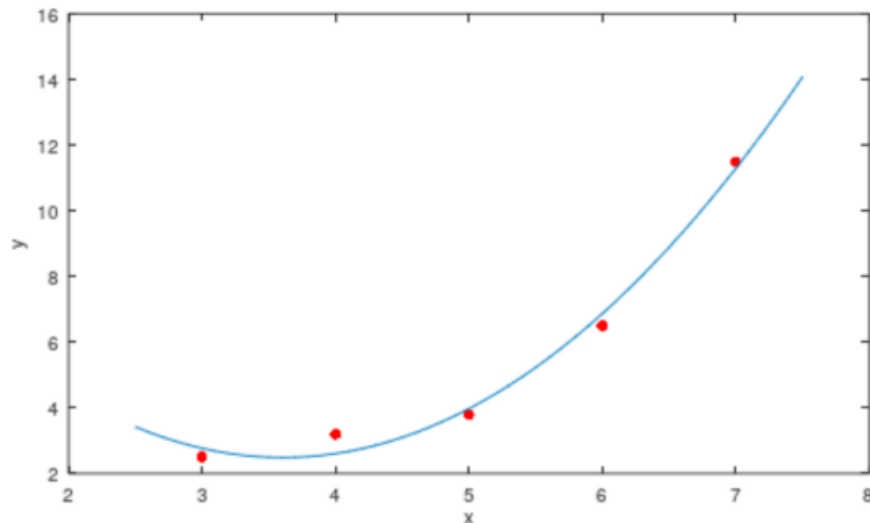
$$\begin{bmatrix} 1 & x_i & x_i^2 & \cdots \end{bmatrix} \cdot \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \end{bmatrix} = y_i$$

➔ POLYNOMIAL REGRESSION (OLS)

- Example

x	3.0	4.0	5.0	6.0	7.0
y	2.5	3.2	3.8	6.5	11.5

$$y = \alpha_0 + \alpha_1 x + \alpha_2 x^2$$



$$\begin{aligned}\sum y_i &= n\alpha_0 + \alpha_1 \left(\sum x_i\right) + \alpha_2 \left(\sum x_i^2\right) \\ \sum y_i x_i &= \alpha_0 \left(\sum x_i\right) + \alpha_1 \left(\sum x_i^2\right) + \alpha_2 \left(\sum x_i^3\right) \\ \sum y_i x_i^2 &= \alpha_0 \left(\sum x_i^2\right) + \alpha_1 \left(\sum x_i^3\right) + \alpha_2 \left(\sum x_i^4\right)\end{aligned}$$



$$27.5 = 5\alpha_0 + 25\alpha_1 + 135\alpha_2$$

$$158.8 = 25\alpha_0 + 135\alpha_1 + 775\alpha_2$$

$$966.2 = 135\alpha_0 + 775\alpha_1 + 4659\alpha_2$$



$$\alpha_0 = 12.4285714$$

$$\alpha_1 = -5.5128571$$

$$\alpha_2 = 0.7642857$$



$$y = 12.4285714 - 5.5128571x + 0.7642857x^2$$

➔ POLYNOMIAL REGRESSION (OLS)

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
```

```
# Data
```

```
X = np.array([[1], [2], [3], [4], [5]])
y = np.array([1, 4, 9, 16, 25]) #  $y = x^2$ 
```

```
# Transform  $X \rightarrow [1, x, x^2]$ 
```

```
poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X)
```

```
# Fit model
```

```
model = LinearRegression()
model.fit(X_poly, y)
```

```
# Predictions
```

```
y_pred = model.predict(X_poly)
```

→ MULTIPLE LINEAR REGRESSION (OLS)

- Model

$$y = \beta_0 + \beta_1 x_1 + \cdots + \beta_N x_N$$

- Ordinary least squares

Variables	Values (examples)			
	Example 1	Example 2	...	Example n
x_1	x_{11}	x_{12}	...	x_{1n}
x_2	x_{21}	x_{22}	...	x_{2n}
...				
x_N	x_{N1}	x_{N2}	...	x_{Nn}
y (outcomes)	y_1	y_2	...	y_n

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \quad X = \begin{bmatrix} 1 & x_{11} & x_{21} & \cdots & x_{N1} \\ 1 & x_{12} & x_{22} & \cdots & x_{N2} \\ \vdots & & & & \\ 1 & x_{1n} & x_{2n} & \cdots & x_{Nn} \end{bmatrix}, \quad \text{and } B = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_N \end{bmatrix}$$

$$B = (X^T X)^{-1} X^T Y.$$

→ MULTIPLE LINEAR REGRESSION (OLS)

- Example

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

x_1	1	1	2	0
x_2	1	2	2	1
y	3.25	6.5	3.5	5.0

$$B = (X^T X)^{-1} X^T Y$$

$$Y = \begin{bmatrix} 3.25 \\ 6.5 \\ 3.5 \\ 5.0 \end{bmatrix}, \quad X = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 2 \\ 1 & 2 & 2 \\ 1 & 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix}$$

$$y = 2.0625 - 2.3750x_1 + 3.2500x_2$$

$$X^T X = \begin{bmatrix} 4 & 4 & 6 \\ 4 & 6 & 7 \\ 6 & 7 & 10 \end{bmatrix} \Rightarrow (X^T X)^{-1} = \begin{bmatrix} \frac{11}{4} & \frac{1}{2} & -2 \\ \frac{1}{2} & 1 & -1 \\ -2 & -1 & 2 \end{bmatrix} \Rightarrow B = (X^T X)^{-1} X^T Y = \begin{bmatrix} 2.0625 \\ -2.3750 \\ 3.2500 \end{bmatrix}$$


➔ MULTIPLE LINEAR REGRESSION (OLS)

```
X = np.array([
    [1, 2],
    [2, 1],
    [3, 4],
    [4, 3],
    [5, 5]
])

y = np.array([5, 6, 11, 12, 15])    # Target

model = LinearRegression()           # Model
model.fit(X, y)

print("Intercept (b):", model.intercept_)    # Parameters
print("Coefficients (w1, w2):", model.coef_)

y_pred = model.predict(X)
print("Predictions:", y_pred)              # Prediction
```

➔ REGRESSION WITH GRADIENT DESCENT (GD)

- Linear regression

$$y = f(x, \theta)$$

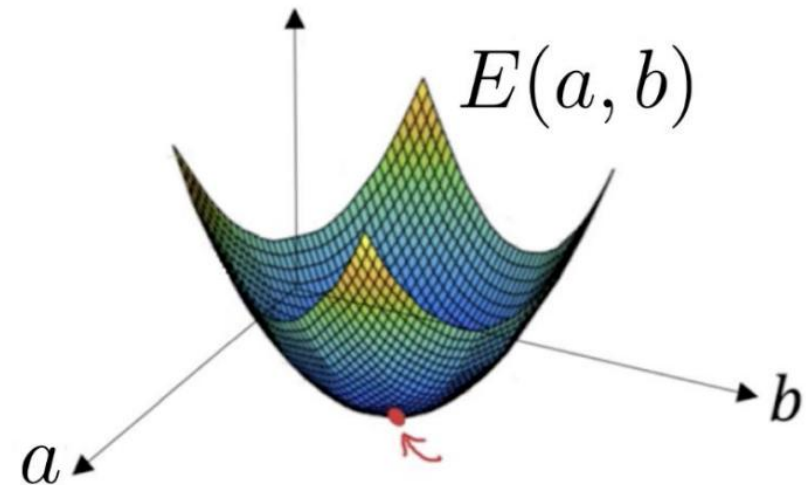
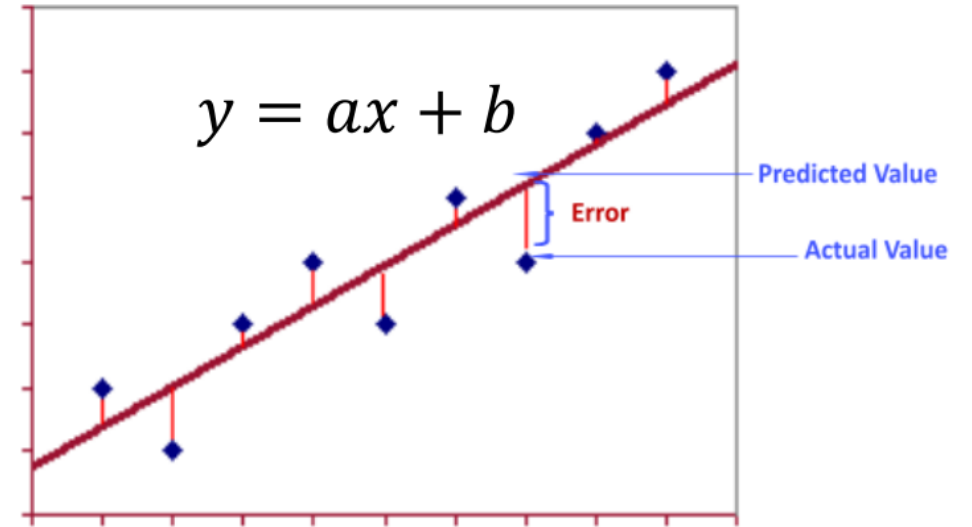
- Objective function

- Sum of squared error

$$E = \sum_{i=0}^n (y_i - (ax_i + b))^2$$

- Solution

$$\min \sum_{i=0}^n (y_i - (ax_i + b))^2$$



➔ REGRESSION WITH GRADIENT DESCENT (GD)

Step 1: Take the derivative of the Loss Function for each parameter in it. In machine learning terms, compute the gradient of the loss function.

Step 2: Pick random initial values for the parameters.
(Slope and intercept)

Step 3: Substitute the parameter values into the derivatives (the gradient).

Step 4: Calculate the step sizes:

Step Size = Gradient $\times$ Learning Rate



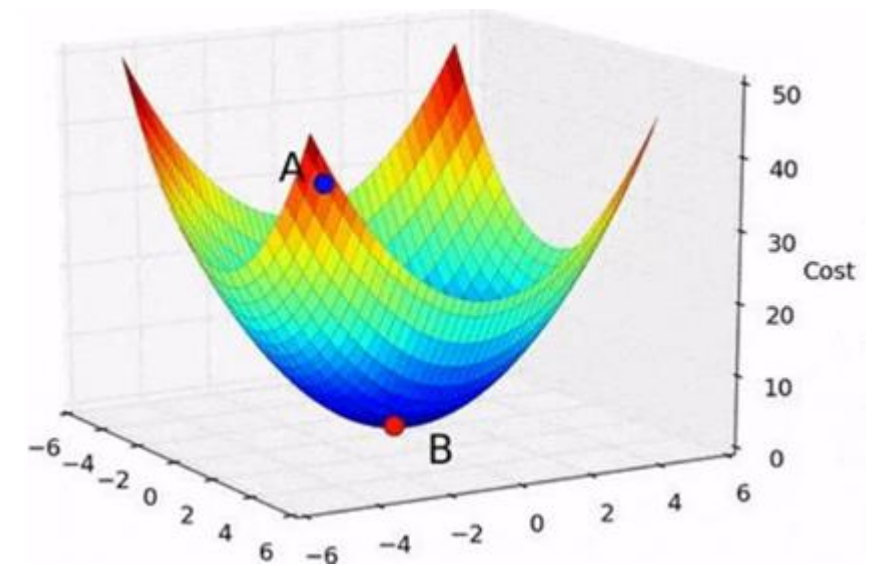
$$\eta \nabla f(\vec{p}_n)$$

Step 5: Update the parameters:

New Parameter = Old Parameter $-$ Step Size



$$\vec{p}_{n+1} = \vec{p}_n - \eta \nabla f(\vec{p}_n)$$



➔ REGRESSION WITH GRADIENT DESCENT (GD)

Then go back to **Step 3** and repeat until the step size becomes very small, or until you reach the maximum number of steps (iterations).

Step 3: Plug the parameter values into the derivatives (i.e., compute the gradient).

Step 4: Calculate the step sizes:

Step Size = Slope × Learning Rate

$$\eta \nabla f(\vec{p}_n)$$

Step 5: Update the parameters:

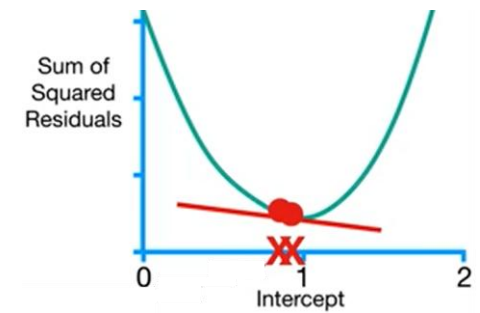
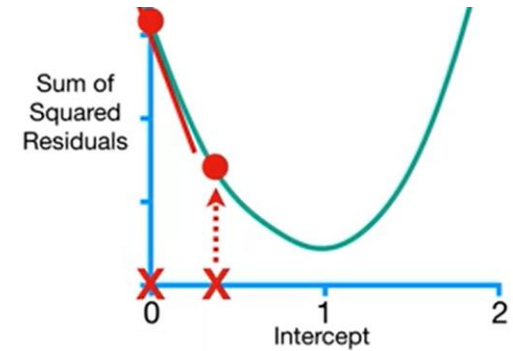
New Parameter = Old Parameter – Step Size

$$\vec{p}_{n+1} = \vec{p}_n - \eta \nabla f(\vec{p}_n)$$

➔ REGRESSION WITH GRADIENT DESCENT (GD)

Step size versus slope

- When the slope is far from 0, then we should take big steps because we are far from the optimal value
- The closer we get to the optimal value of the intercept, the closer the slope of the curve approaches zero.
- The size of the step should be related to the slope, as it indicates whether we should take a small step or a large step.

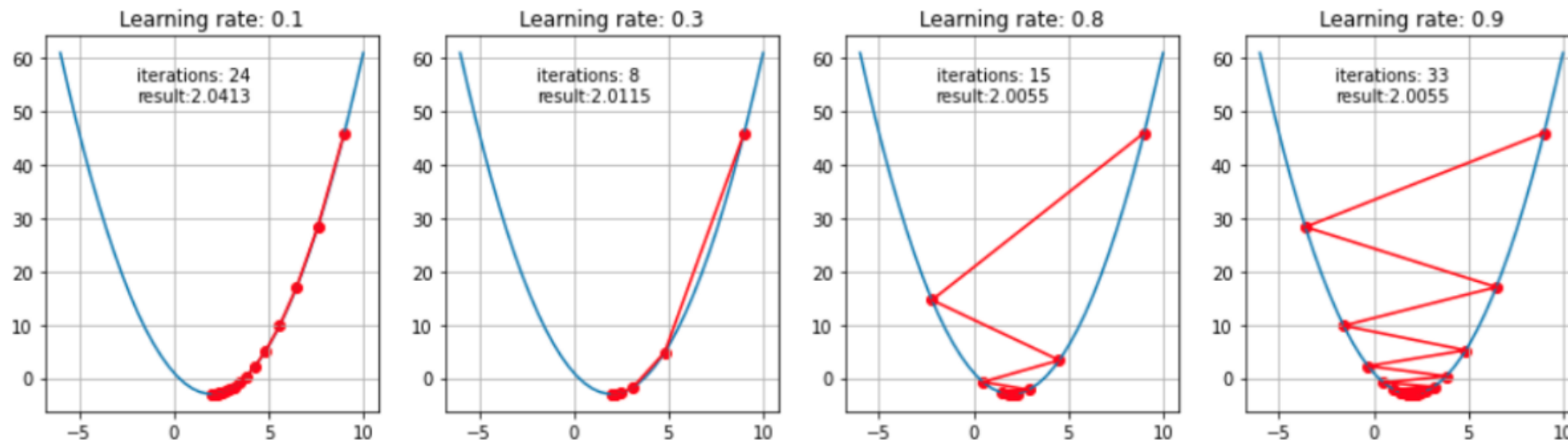


$$\eta \nabla f(\vec{p}_n)$$

➔ REGRESSION WITH GRADIENT DESCENT (GD)

Learning rate

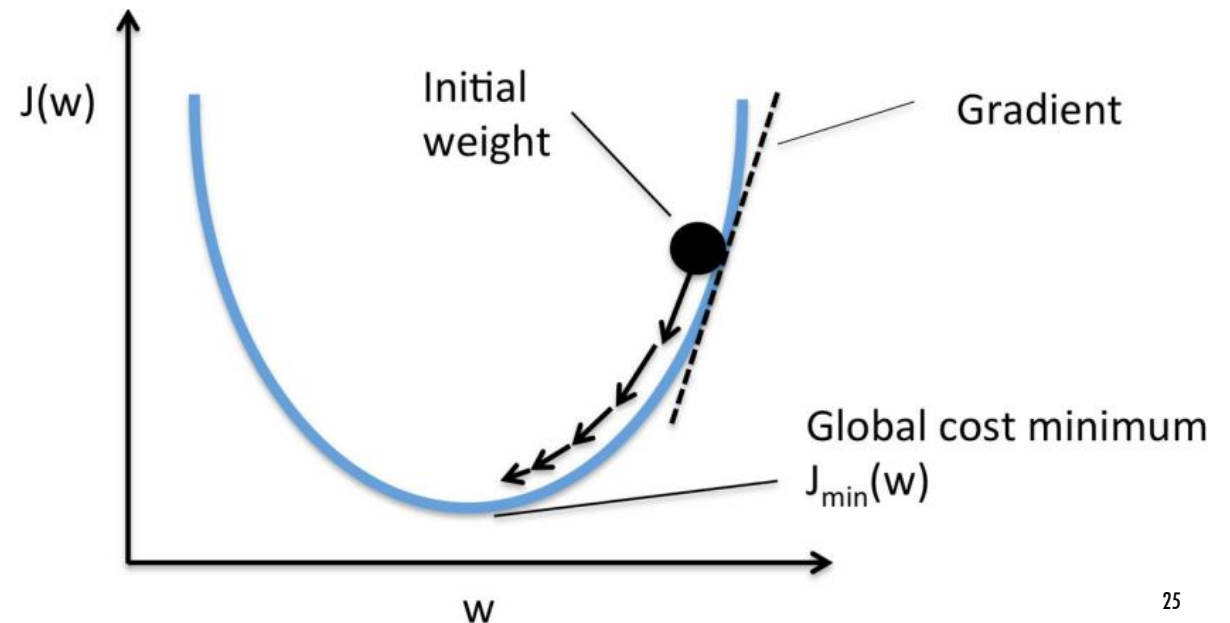
- Parameter update:
$$\vec{p}_{n+1} = \vec{p}_n - \eta \nabla f(\vec{p}_n)$$
- Learning rate η : scales the gradient and thus controls the step size
 - small: Slow to converge; may reach maximum iteration before convergence
 - big: May not converge to the optimal point or even to diverge completely



➔ REGRESSION WITH GRADIENT DESCENT (GD)

Learning rate

- Use a fixed learning rate
 - Try with a large value like 0.1
 - Try exponentially lower values: 0.01, 0.001, etc.
- Use an adaptive learning rate
 - Start with a larger value
 - Gradual decrease it



➔ REGRESSION WITH GRADIENT DESCENT (GD)

```
X = np.array([1, 2, 3, 4, 5])  
y = np.array([2, 4, 5, 4, 5])
```

```
# Initialize  
w = 0.0  
b = 0.0  
alpha = 0.01  
epochs = 100000  
n = len(X)  
loss_history = [ ]
```



```
for i in range(epochs):  
    y_pred = w * X + b  
    error = y - y_pred
```



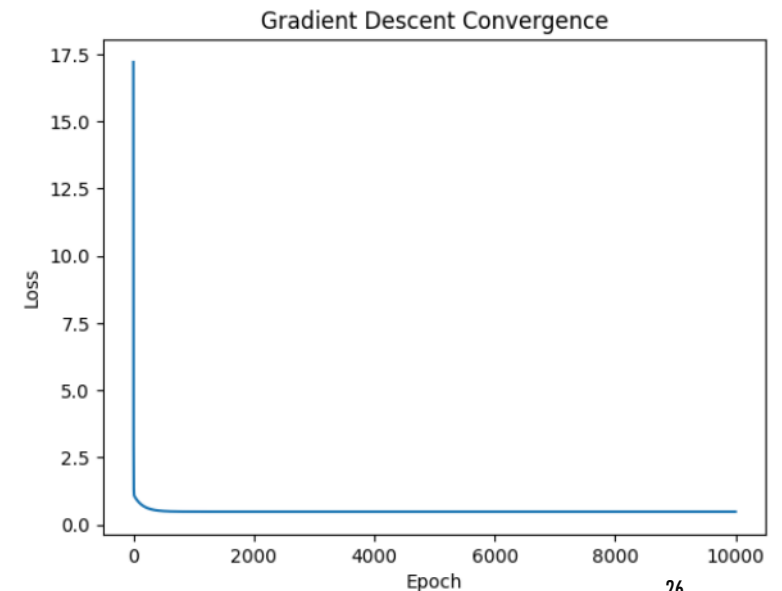
```
    dw = (-2/n) * np.sum(X * error)  
    db = (-2/n) * np.sum(error)
```

```
    w -= alpha * dw  
    b -= alpha * db
```

```
    loss = np.mean(error**2)  
    loss_history.append(loss)
```

```
Print(w, b)
```

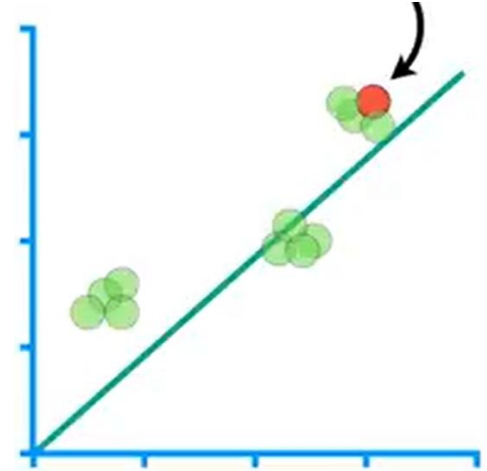
```
plt.plot(loss_history)  
plt.xlabel("Epoch")  
plt.ylabel("Loss")  
plt.title("Gradient Descent Convergence")  
plt.show()
```



➔ REGRESSION: STOCHASTIC GRADIENT DESCENT (SGD)

Stochastic Gradient Descent

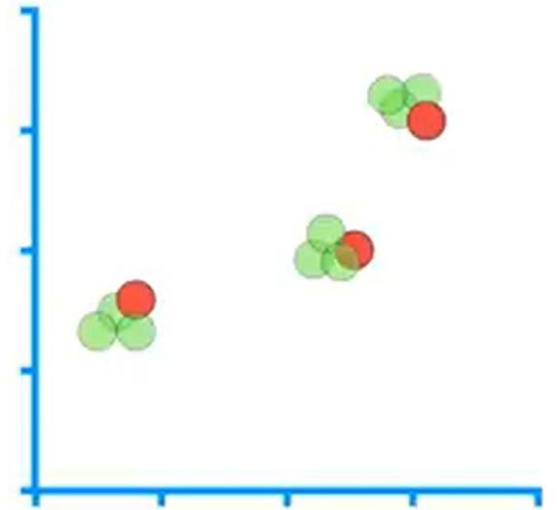
- For big data (tens of features + thousands of observations) gradient Descent becomes slow.
- **Solution:** Stochastic Gradient Descent
- SGD selects a one sample (1 observation) from the observations and evaluates the derivatives based on this sample. This will divide the time of computations by the number of observations.
- Just like in regular Gradient Descent, the general strategy is to start with a relatively large learning rate and gradually decrease it with each step.



➔ REGRESSION: STOCHASTIC GRADIENT DESCENT (SGD)

Mini-Batch Stochastic Descent

- Using a mini-batch at each step combines the advantages of using a single sample and using the entire dataset for each update.
- Similar to using the entire dataset, using a mini-batch can lead to more stable parameter estimates in fewer steps.
- Like using a single sample per step, using a mini-batch is much faster than using the entire dataset.



➡ REGRESSION: STOCHASTIC GRADIENT DESCENT (SGD)

Mini-Batch Stochastic Descent

```
def mini_batch_gradient_descent(X, y, lr=0.01, epochs=1000, batch_size=5):
    X = np.array(X, dtype=float)
    y = np.array(y, dtype=float)
    w = 0.0
    b = 0.0
    n = len(X)
    for _ in range(epochs):
        # Shuffle data
        indices = np.random.permutation(n)
        X_shuffled = X[indices]
        y_shuffled = y[indices]
        # Create mini-batches
        for i in range(0, n, batch_size):
            X_batch = X_shuffled[i:i+batch_size]
            y_batch = y_shuffled[i:i+batch_size]
            m = len(X_batch)
            # Predictions
            y_pred = w * X_batch + b
            error = y_batch - y_pred
            # Gradients
            dw = (-2/m) * np.sum(X_batch * error)
            db = (-2/m) * np.sum(error)
            # Update
            w -= lr * dw
            b -= lr * db
    return w, b
```

```
X = np.array([1, 2, 3, 4, 5])
y = np.array([2, 4, 5, 4, 5])
```



```
w, b = mini_batch_gradient_descent(X, y, lr=0.01, epochs=5000,
batch_size=2)
```

```
print("w =", w)
print("b =", b)
```