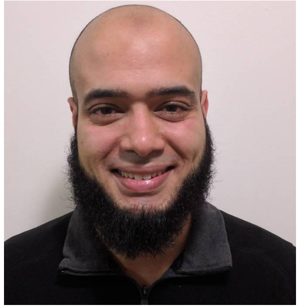
Machine Learning

Linear Regression Homework 1

Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / MSc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)



$$cost(W) = \frac{1}{2N} \sum_{n=1}^{N} (y(X^n, W) - t^n)^2$$

$$\frac{\partial cost(W)}{\partial W_j} = \frac{1}{N} \sum_{n=1}^{N} (y(X^n, W) - t^n) * X_j^n$$

Early Verifications

- You should always think how to verify your code
  - I take baby steps to compile, build and verify
- The easiest way is to start with a simple data for a **line (no noise)**
  - E.g. a 45 degree line
  - You should be able to perfectly fit it
- Consider the following data:
  - x = np.array([0, 0.2, 0.4, 0.8, 1.0])
  - t = 5 + x
  - Clearly the solution for such data is: slope = 1 and intercept = 5
- The right way: Assume data. Compute by hand the derivatives. Code and Compare
  - In the next slide, I give you data to compare!

Transform Data

- For each example [x] we convert it to [1, x]
  - This corresponds to learning [c, m] parameters

0
0.2
0.4
0.8
1.0

$\Rightarrow$

1	0
1	0.2
1	0.4
1	0.8
1	1.0

Early Verifications

- Start from
  - weights = [1, 1] for the [c, m]
  - step size = 0.1
- Cost function output: 8.0
  - Pred: ([1. , 1.2, 1.4, 1.8, 2.])
  - Target: ([5. , 5.2, 5.4, 5.8, 6.])
  - Error = array([-4., -4., -4., -4., -4.])
- Cost: (-4^2 + -4^2 + -4^2 + -4^2 + -4^2) / (2 * 5) = 16 * 5 / (2 * 5) = 8
  - 5 is the number of examples
  - 2 is the factor we use in the division in the equation

$$cost(W) = \frac{1}{2N} \sum_{n=1}^{N} (y(X^n, W) - t^n)^2$$

Early Verifications

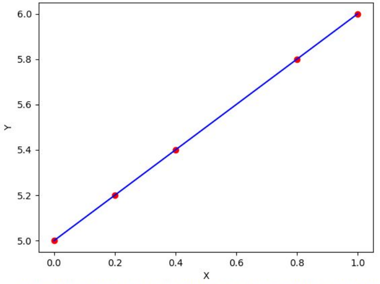
- Gradient: [-4.   -1.92] = Error * X
  - [-4 * 1 + -4 * 1 + -4 * 1 + -4 * 1 + -4 * 1] / 5 = -4
  - [-4 * 0 + -4 * 0.2 + -4 * 0.4 + -4 * 0.8 + -4 * 1.0] / 5 = -1.92
- Updated weights: [1.4   1.192]
  - 1 - 0.1 * -4 = 1.4
- For W = [0.8 0.5]
  - Cost = 9.8739 and Gradient = [-4.44 -2.2], updated weights = [1.244 0.72]

$$\frac{\partial cost(W)}{\partial W_j} = \frac{1}{N}\sum_{n=1}^{N}(y(X^n, W) - t^n) * X_j^n$$

Write 2 functions

- The first function takes input data X and its target output t
- It also takes the current weights (e.g. for a line)
- It computes the cost function
- The function should handle hyperplane in general
- Write another function that computes the derivative
- Code and compare with my numbers
- After that, use these 2 functions in the next homework

```python
def f(X, t, weights):...


def f_dervative(X, t, weights):  ...

if __name__ == '__main__':
    # Input is 1D feature, e.g. the price
    X = np.array([0, 0.2, 0.4, 0.8, 1.0])
    t = 5 + X   # Output linear, no noise

    X = X.reshape((-1, 1))  # let's reshape in 2D
    X = np.hstack([np.ones((X.shape[0], 1)), X])    # add 1 for c

    print(X.shape)  # 5 x 2: for line mx+c

    weights = np.array([1.0, 1.0])  # starting params

    print(f(X, t, weights)) # cost: 8

    print(f_dervative(X, weights))  # dervative: [-4.    -1.92]
```



$$cost(W) = \frac{1}{2N}\sum_{n=1}^{N}(y(X^n, W) - t^n)^2$$

$$\frac{\partial cost(W)}{\partial W_j} = \frac{1}{N}\sum_{n=1}^{N}(y(X^n, W) - t^n) * X_j^n$$

“Acquire knowledge and impart it to the people.”

“Seek knowledge from the Cradle to the Grave.”