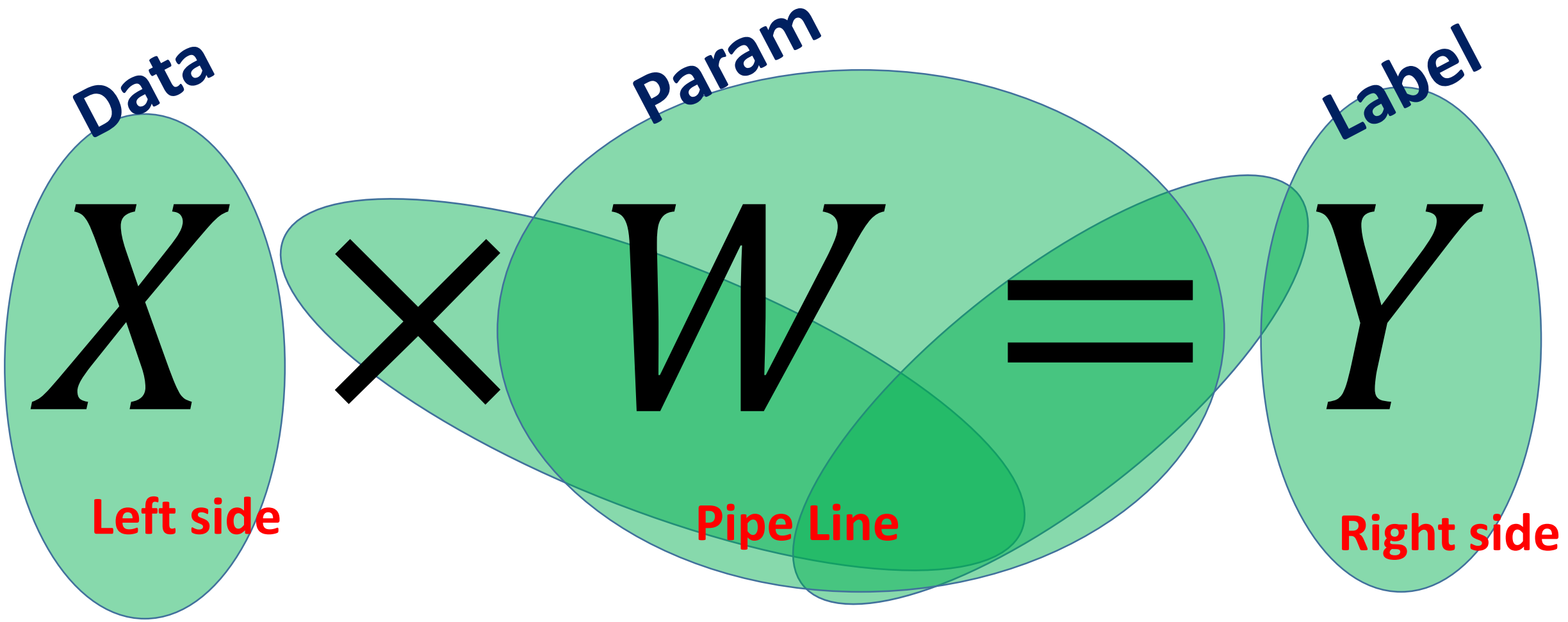
Pattern Mapping for Multivariate Loss Function in Linear Regression

Dr. Kalidas Y.

In this lecture you will understand how to formulate MULTI VARIATE steepest descent

Data, Label, Parameter – Simple supervised case



Fitting a Line Passing Through Origin

- $y = m\,x$
- $L(m) = \sum_{i=1}^{i=N} (y_i - m\,x_i)^2$
- $X = \begin{bmatrix} x_1 \\ \cdots \\ x_N \end{bmatrix}_{N\times 1}, Y = \begin{bmatrix} y_1 \\ \cdots \\ y_N \end{bmatrix}_{N\times 1}, W = [m\,]_{1\times 1}$
- $L([m]) = (XW\ - Y)^T (XW - Y)$
- $\nabla L\ = \left[\frac{\partial L}{\partial m}\right]$ //It's a function
- $W_{(new)} = W_{(old)} - \eta \nabla L|_{W=W_{(old)}}$

Squared error type loss function

Fitting a Line – slope and intercept

- $y = m\,x + c$
- $L(m) = \sum_{i=1}^{i=N} (y_i - (m\,x_i + c))^2$
- $X = \begin{bmatrix} x_1 & 1 \\ \dots & \\ x_N & 1 \end{bmatrix}_{N\times 2}$, $Y = \begin{bmatrix} y_1 \\ \dots \\ y_N \end{bmatrix}_{N\times 1}$, $W = \begin{bmatrix} m \\ c \end{bmatrix}_{2\times 1}$
- $L\left(\begin{bmatrix} m \\ c \end{bmatrix}\right) = (XW - Y)^T (XW - Y)$
- $\nabla L = \begin{bmatrix} \frac{\partial L}{\partial m} \\ \frac{\partial L}{\partial c} \end{bmatrix}$ //It's a function
- $W_{(new)} = W_{(old)} - \eta \nabla L|_{W=W_{(old)}}$

Squared error type loss function

Fitting a Parabola?

- $y = a\,x^2 + b\,x + c$
- $L(a, b, c) = \sum_{i=1}^{i=N} \left(y_i - (a\,x_i^2 +\right.$

Squared error type loss function

Fitting a Cubic curve?

- $y = a\,x^3 + b\,x^2 + c\,x + d$
- $L(a, b, c, d) = \sum_{i=1}^{i=N} \left(y_i - (a\,x_i^3 +\right.$

Squared error type loss function

Fitting a Degree-K polynomial?

- $y = a_k\, x^k + \cdots + a_0$
- $L(m) = \sum_{i=1}^{i=N} \left(y_i - \sum_{j=0}^{k} a_j\, x^j\right)^2$
- $X = \begin{bmatrix} x_1^k \ldots x_1^2 & x_1^1 & 1 \\ & \ldots & \\ x_N^k \ldots x_N^2 & x_N^1 & 1 \end{bmatrix}_{N\times(k+1)}$,
- $Y = \begin{bmatrix} y_1 \\ \ldots \\ y_N \end{bmatrix}_{N\times 1}$,
- $W = \begin{bmatrix} a_0 \\ \ldots \\ a_k \end{bmatrix}_{(k+1)\times 1}$
- $L(W) = (XW - Y)^T (XW - Y)$
- $\nabla L = \begin{bmatrix} \frac{\partial L}{\partial a_0} \\ \ldots \\ \frac{\partial L}{\partial a_k} \end{bmatrix}$ //It's a function
- $W_{(new)} = W_{(old)} - \nabla L|_{W=W_{(old)}}$

Squared error type loss function

Steepest Descent for Multi Variate Loss function

$$W_{(new)} = W_{(old)} - \eta \nabla L(W) \Big|_{W=W_{(old)}}$$

This is a ***step size***
a.k.a ***learning rate***

This is a ***function***

Function computed ***at***

specified point
of interest

Matrix form of Gradient function (2D)

- $L(W) = \sum_{i=1}^{i=n} \left(x_{i,1} w_1 + w_2 - y_i\right)^2$
- $\frac{\partial L(W)}{\partial w_1} = \sum_{i=1}^{i=n} 2 * \left(x_{i,1} w_1 + w_2 - y_i\right)^1 x_{i,1} \; = 2 * X[:,1]^T (XW - Y)$
- …
- $\frac{\partial L(W)}{\partial w_2} = \sum_{i=1}^{i=n} 2 * \left(x_{i,1} w_1 + w_2 - y_i\right)^1 = 2 * X[:,2]^T (XW - Y)$
- $\nabla L(W) = \begin{bmatrix} \frac{\partial L(W)}{\partial w_1} \\ \frac{\partial L(W)}{\partial w_2} \end{bmatrix} = \begin{bmatrix} 2 * X[:,1]^T (XW - Y) \\ 2 * X[:,2]^T (XW - Y) \end{bmatrix} = 2 * \begin{bmatrix} X[:,1]^T \\ X[:,2]^T \end{bmatrix} * (XW - Y)$
- $= 2\, X^T (XW - Y)$

Squared error type loss function

Matrix form of Gradient function (3D)

- $L(W) = \sum_{i=1}^{i=n} \left(x_{i,1} w_1 + x_{i,2}\, w_2 + x_{i,3} w_3 - y_i\right)^2$
- $\frac{\partial L(W)}{\partial w_1} = \sum_{i=1}^{i=n} 2 * \left(x_{i,1} w_1 + x_{i,2}\, w_2 + x_{i,3} w_3 - y_i\right)^1 x_{i,1} \; = 2 * X[:,1]^T (XW - Y)$
- …
- $\frac{\partial L(W)}{\partial w_3} = \sum_{i=1}^{i=n} 2 * \left(x_{i,1} w_1 + x_{i,2}\, w_2 + x_{i,3} w_3 - y_i\right)^1 x_{i,1} \; = 2 * X[:,3]^T (XW - Y)$
- $\nabla L(W) = \begin{bmatrix} \frac{\partial L}{\partial w_1} \\ \frac{\partial L}{\partial w_2} \\ \frac{\partial L}{\partial w_3} \end{bmatrix} = \begin{bmatrix} 2 * X[:,1]^T (XW - Y) \\ 2 * X[:,2]^T (XW - Y) \\ 2 * X[:,3]^T (XW - Y) \end{bmatrix} = 2 * \begin{bmatrix} X[:,1]^T \\ X[:,2]^T \\ X[:,3]^T \end{bmatrix} * (XW - Y)$
- $= 2\, X^T (XW - Y)$

Squared error type loss function

Matrix form of Gradient function (Multi-Variate)

- $L(W) = \sum_{i=1}^{i=n}\left(\left(\sum_{j=0}^{j=k} x_{i,j} w_j\right) - y_i\right)^2$
- $\frac{\partial L(W)}{\partial w_j} = \sum_{i=1}^{i=n} 2 * \left\{\left(\left(\sum_{j=0}^{j=k} x_{i,j} w_j\right) - y_i\right)^1 * x_{i,j}\right\} = 2 * X[:,j]^T (XW - Y)$
- $\nabla L(W) = \begin{bmatrix} \frac{\partial L}{\partial w_0} \\ \cdots \\ \frac{\partial L}{\partial w_k} \end{bmatrix} = \begin{bmatrix} 2 * X[:,0]^T (XW - Y) \\ \cdots \\ 2 * X[:,k]^T (XW - Y) \end{bmatrix} = 2 * \begin{bmatrix} X[:,0]^T \\ \cdots \\ X[:,k]^T \end{bmatrix} * (XW - Y)$
- $= 2\, X^T (XW - Y)$

Squared error type loss function

Steepest Descent for Multi Variate for Squared Error Loss function

$$\boldsymbol{W_{(new)} = W_{(old)} - \eta X^T (XW - Y)}$$

Multi Variate

Squared error type loss function

Write Code!

What are the different ways?

2nd argument is the *list of Parameters*

```
BruteForceSolver( L, [m1,c1])
```

$minval = +10000.0$

FOR $m \in [-100, \ldots, +100]$

  FOR $c \in [-100, \ldots, +100]$

    Compute $v = L(m, c)$

    IF $v < minval$**:**
      $v = minval$
      $m1 = m,$
      $c1 = c$

1st argument is the *Loss function*

Questions…

- *-100 to +100, who gave the range?*
- *What is the step size?*
- *What if the solution is highly fine, (3.451, -89.1123)*
- *Can we speed up?*

What are the different ways?

2nd argument is *the list of Parameters*

RandomSolver(L, [m1,c1])

$minval = +10000.0$

FOR ITER= 1:100000

FOR $m = RAND(-100, 100)$

FOR $c = RAND(-100, 100)$

Compute $v = L(m, c)$

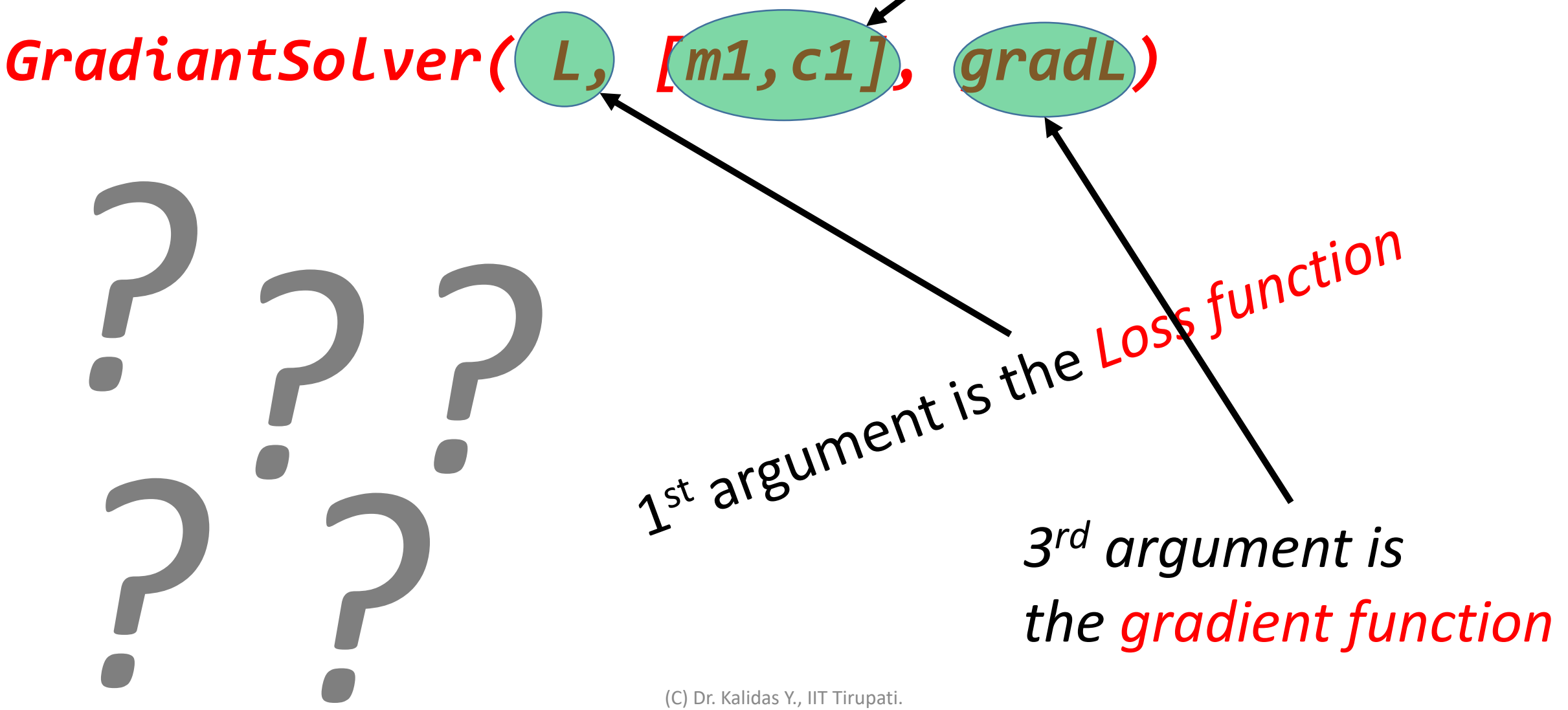
IF $v < minval$:
$v = minval$
$m1 = m$,
$c1 = c$

1st argument is the Loss function

Questions…

- *-100 to +100, who gave the range?*
- *What is the step size?*
- *What if the solution is highly fine, (3.451, -89.1123)*
- *Can we speed up?*

What are the different ways?



What are the different ways?

2nd argument is *the list of Parameters*

```
GradiantSolver( L, [m,c])
```

- **INIT m,c = 0**
- **Iterate 1000 steps (*or whatever*)**
  - $m_{(new)} = m_{(old)} - \frac{\partial L}{\partial m}|_{m=m_{(old)}}$
  - $c_{(new)} = c_{(old)} - \frac{\partial L}{\partial c}|_{c=c_{(old)}}$

1st argument is the Loss function

What are the different ways?

2nd argument is the *list of Parameters*

```
SteepestDescentSolver( L, [m1,c1], gradL)
```

```
//m1,c1 = some initial value is already input to the function
```

- **Iterate 1000 steps (*or whatever*)**
- **IF L(m1,c1) < Threshold**
  - **RETURN**
- **[m1,c1] = [m1,c1] – gradL(m1,c1)**

1st argument is the Loss function

3rd argument is the gradient function

This function outputs a list or vector

Defining a loss function

```
# Author: Kalidas Y <ykalidas at iittp dot a
c dot in>
# License: BSD 3 clause

def fun_error(w) :
  diff = np.matmul(w, X.T) - y.T
  err_val = np.matmul( diff, diff.T )/X.shap
e[0]
  return err_val[0][0]
```

Defining Gradient Function

```
# Author: Kalidas Y <ykalidas at iittp dot ac dot
in>
# License: BSD 3 clause

def fun_grad_error(w) :
  grad_val = 2*np.matmul( np.matmul(w, X.T) -
 y.T, X )/X.shape[0]
  return np.concatenate(grad_val)
```

Invoking Optimization function

```
# Author: Kalidas Y <ykalidas at iittp dot ac dot
in>
# License: BSD 3 clause

from scipy.optimize import minimize

w_init = np.random.randn(1,2)

res = minimize(fun=fun_error, x0=w_init, jac=fun_g
rad_error,method='BFGS')
```