Lecture 5 – Gradient Descent

ME494 – Data Science and Machine Learning for Mech Engg

Instructor – Subramanian Sankaranarayanan

Teaching Assistants

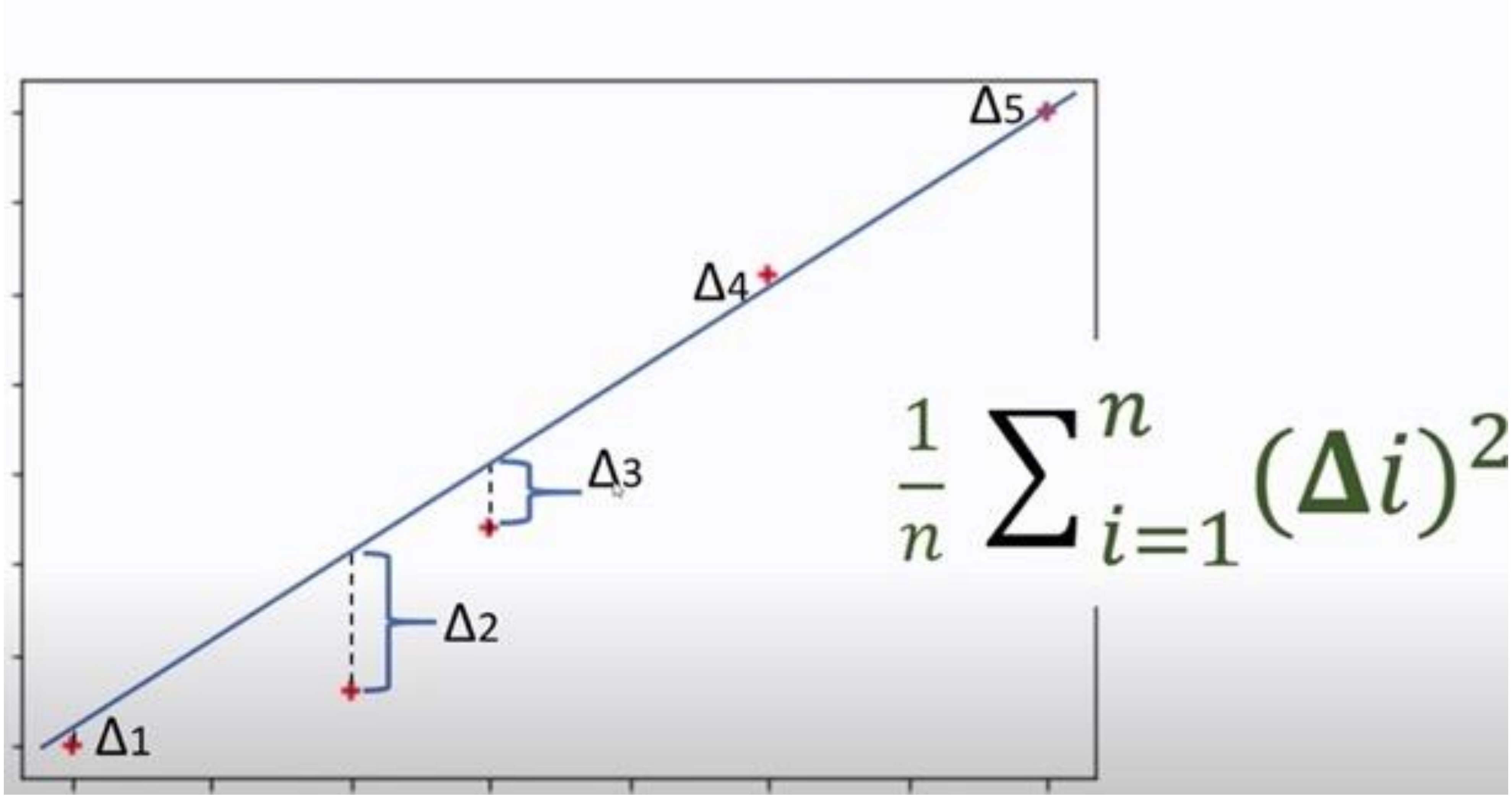
Aditya Koneru (akoner3@uic.edu)

Suvo Banik (sbanik2@uic.edu)

Learning objectives for this class

- Introduction to gradient descent for finding optimal solution
- Loss function and gradient descent
- Concept of learning rate
- Types of gradient descent - Stochastic vs. batch vs. mini batch
- Python implementation of the gradient descent approach

Loss Function – Mean Square Error



Y=f(X)

X

Mean Square Error

Mean square error $\frac{1}{N}\sum_{i=1}^{N}(e_i)^2$

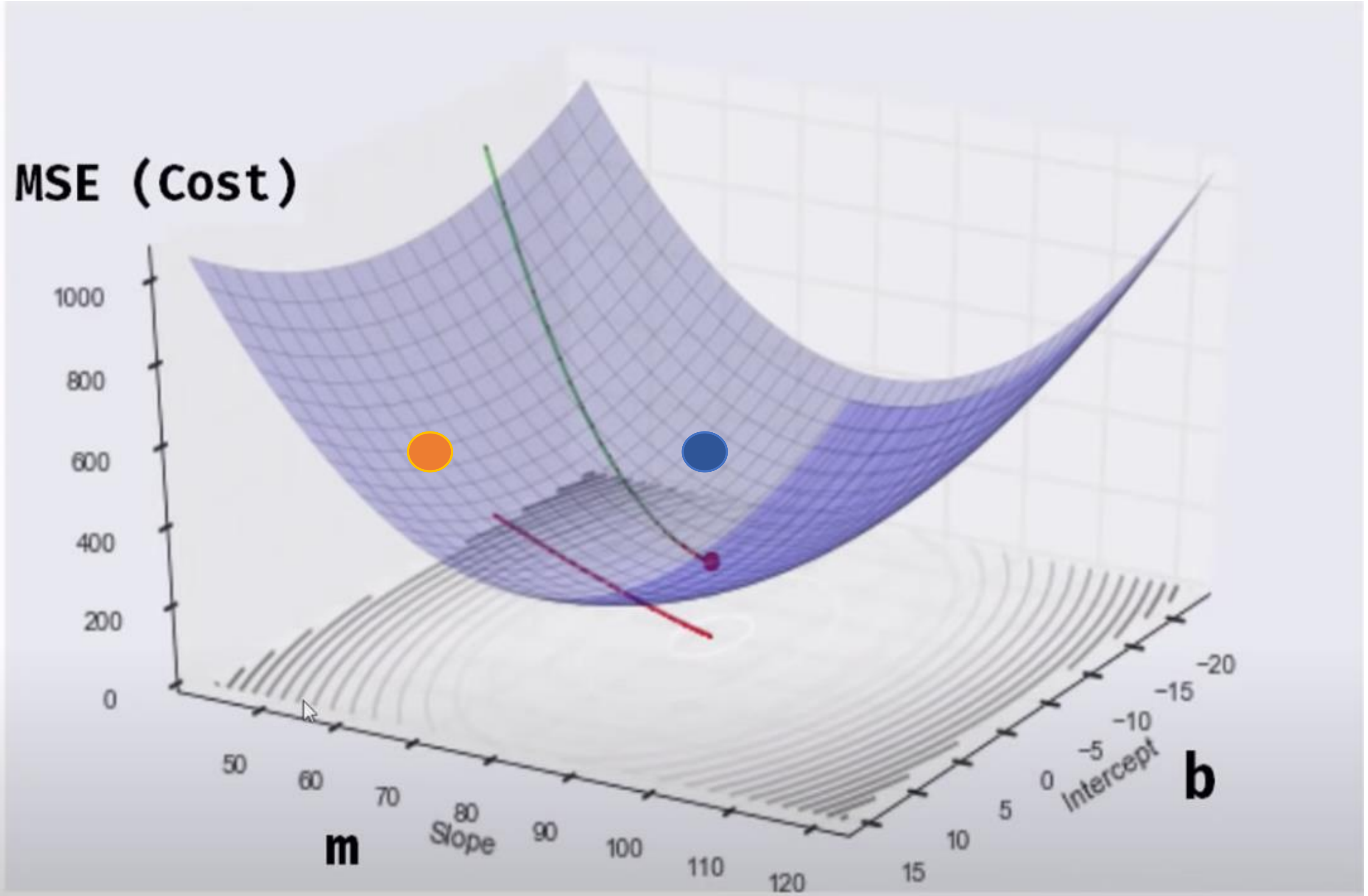
For any ML model, MSE = $\frac{1}{n}\sum_{i=1}^{n}\left(y_i - y_{predicted}\right)^2$

Loss Function MSE $= \frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i))^2$

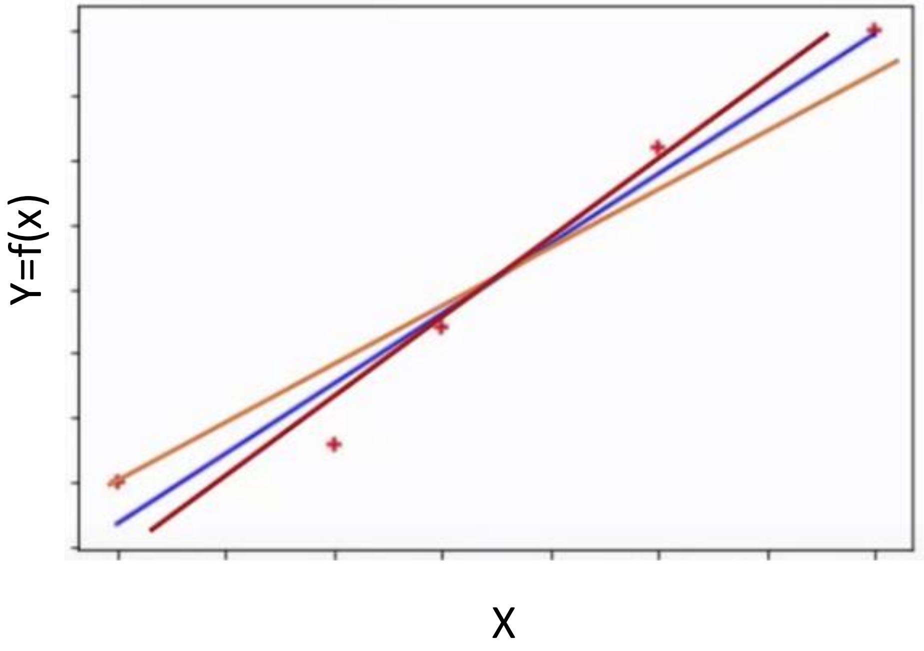
Loss Function MSE $= \frac{1}{N}\sum_{i=1}^{N}(y_i - \beta_0 - \beta_1 x_{i1})^2$

Gradient descent aims to find the best fit line for a given training dataset

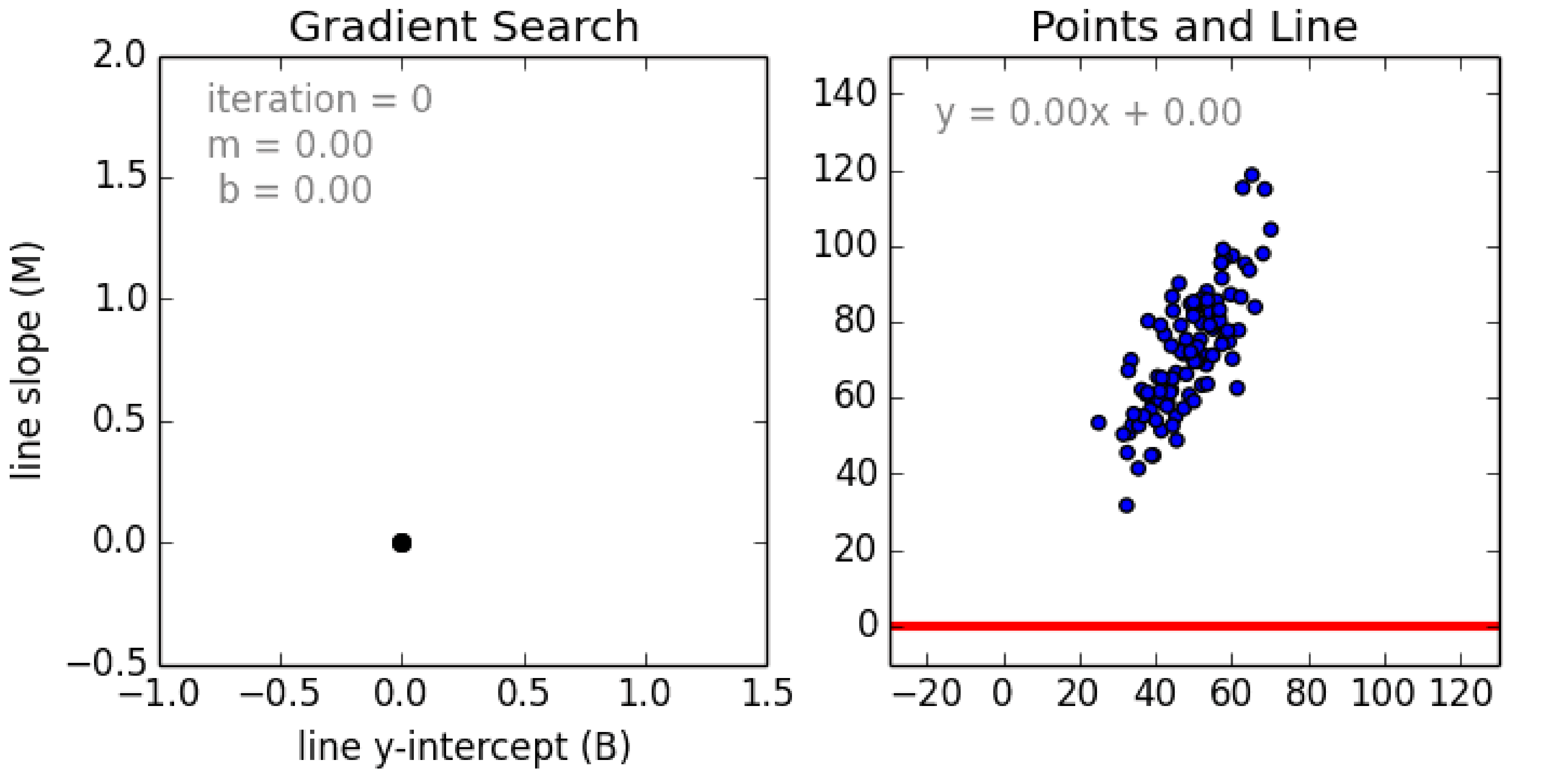
What is gradient descent?



Reference: https://am207.github.io/2017/wiki/gradientdescent.html

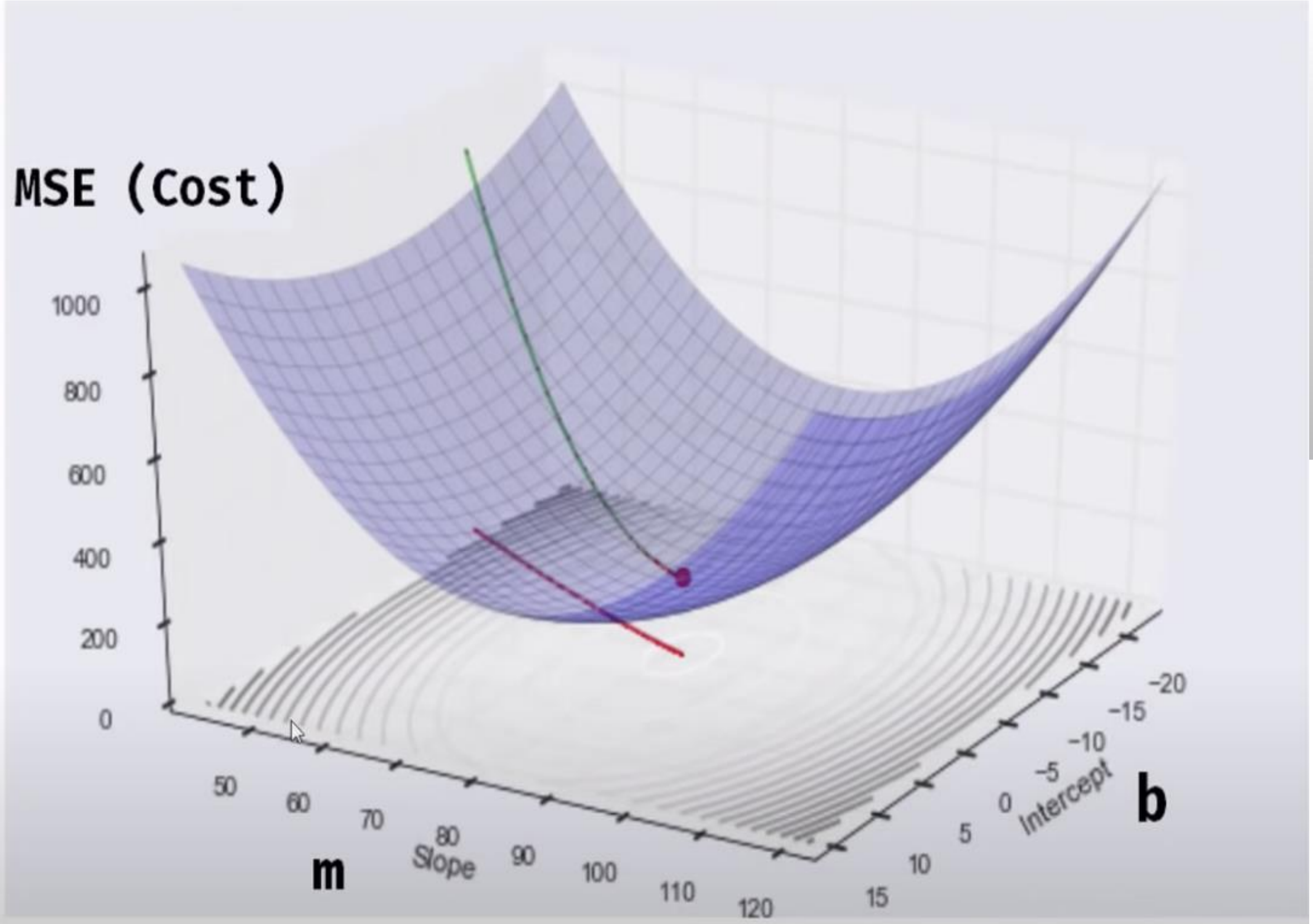


Example of a Gradient based search

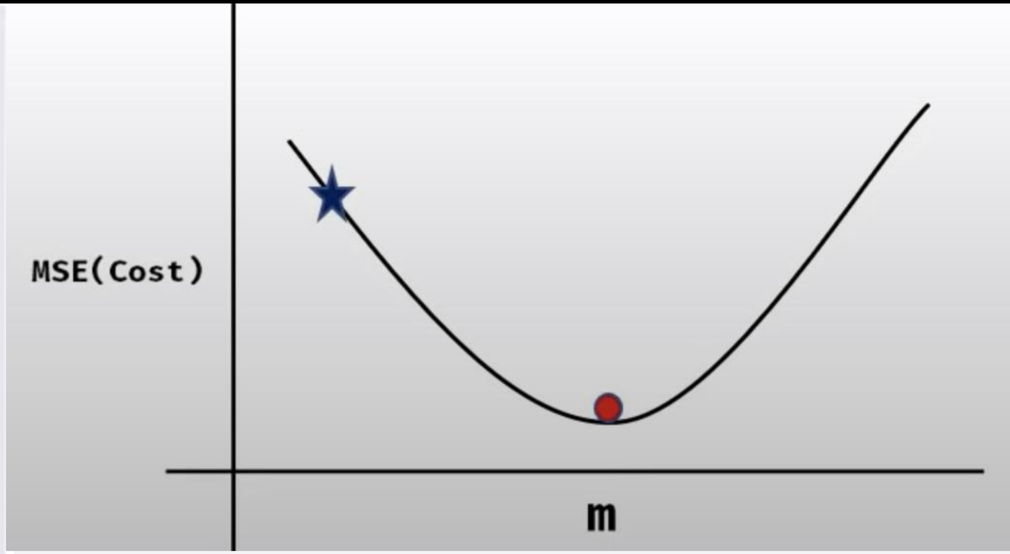


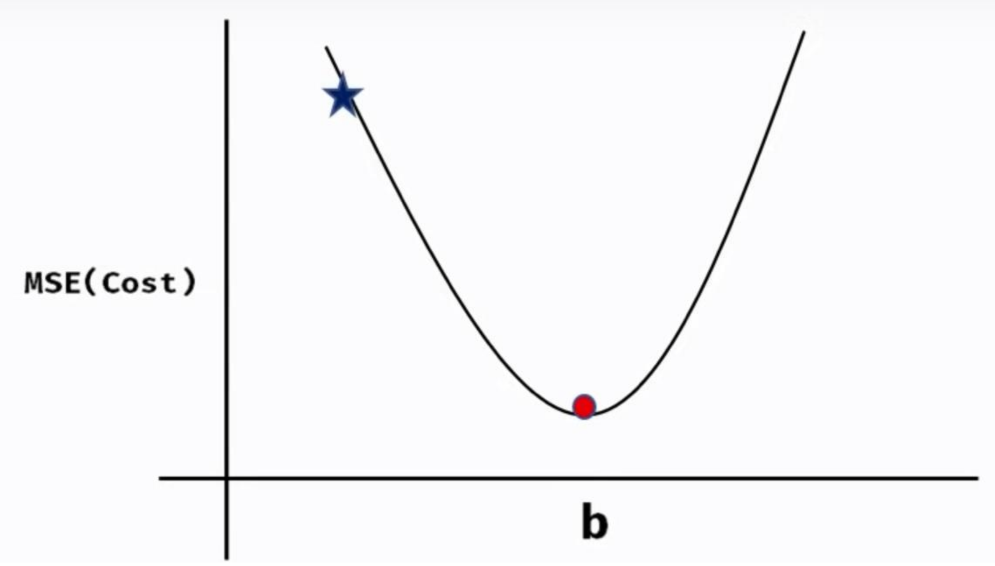
Reference - https://github.com/mattnedrich/GradientDescentExample/tree/master

Example of a Gradient based search

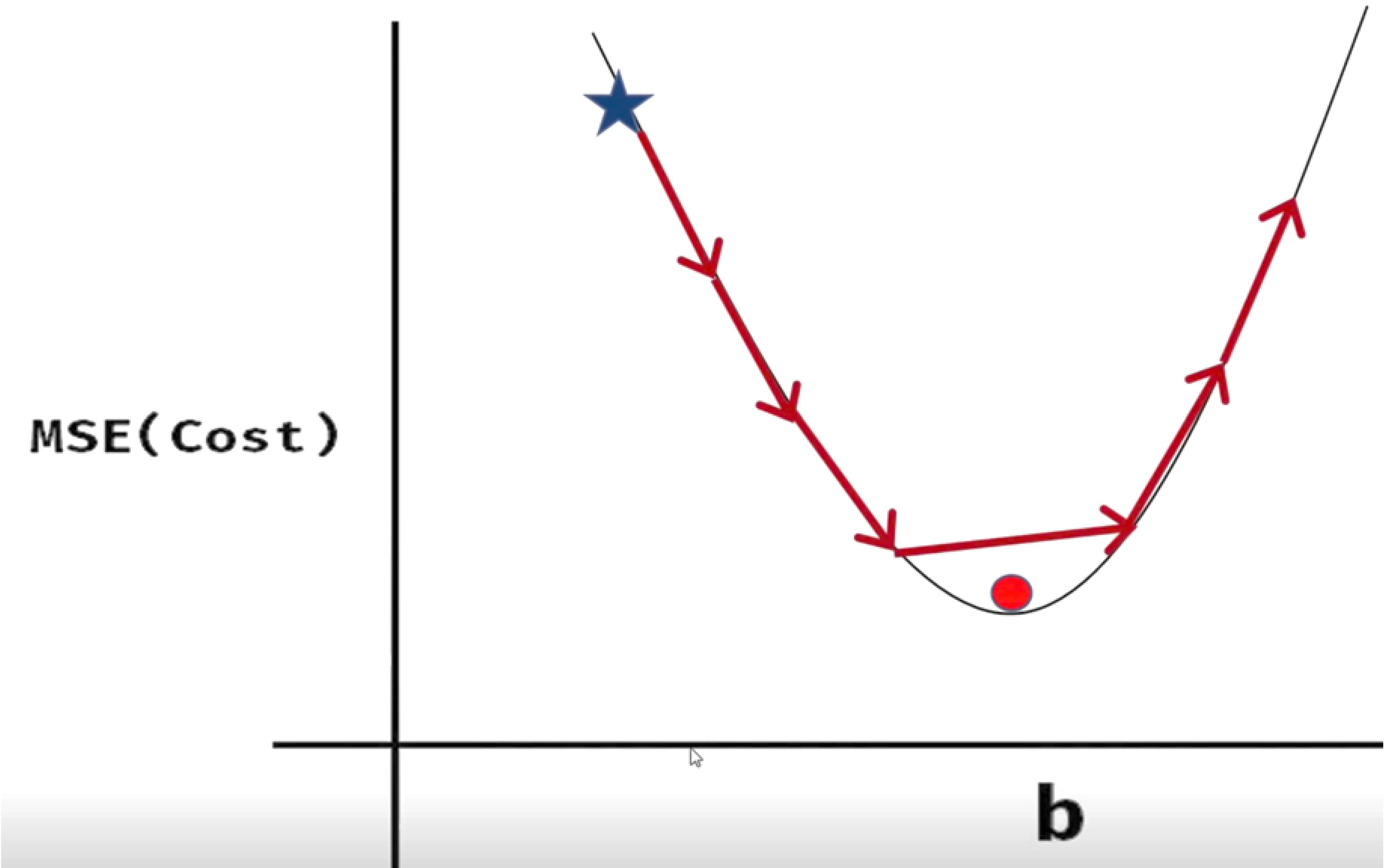


Reference: https://am207.github.io/2017/wiki/gradientdescent.html

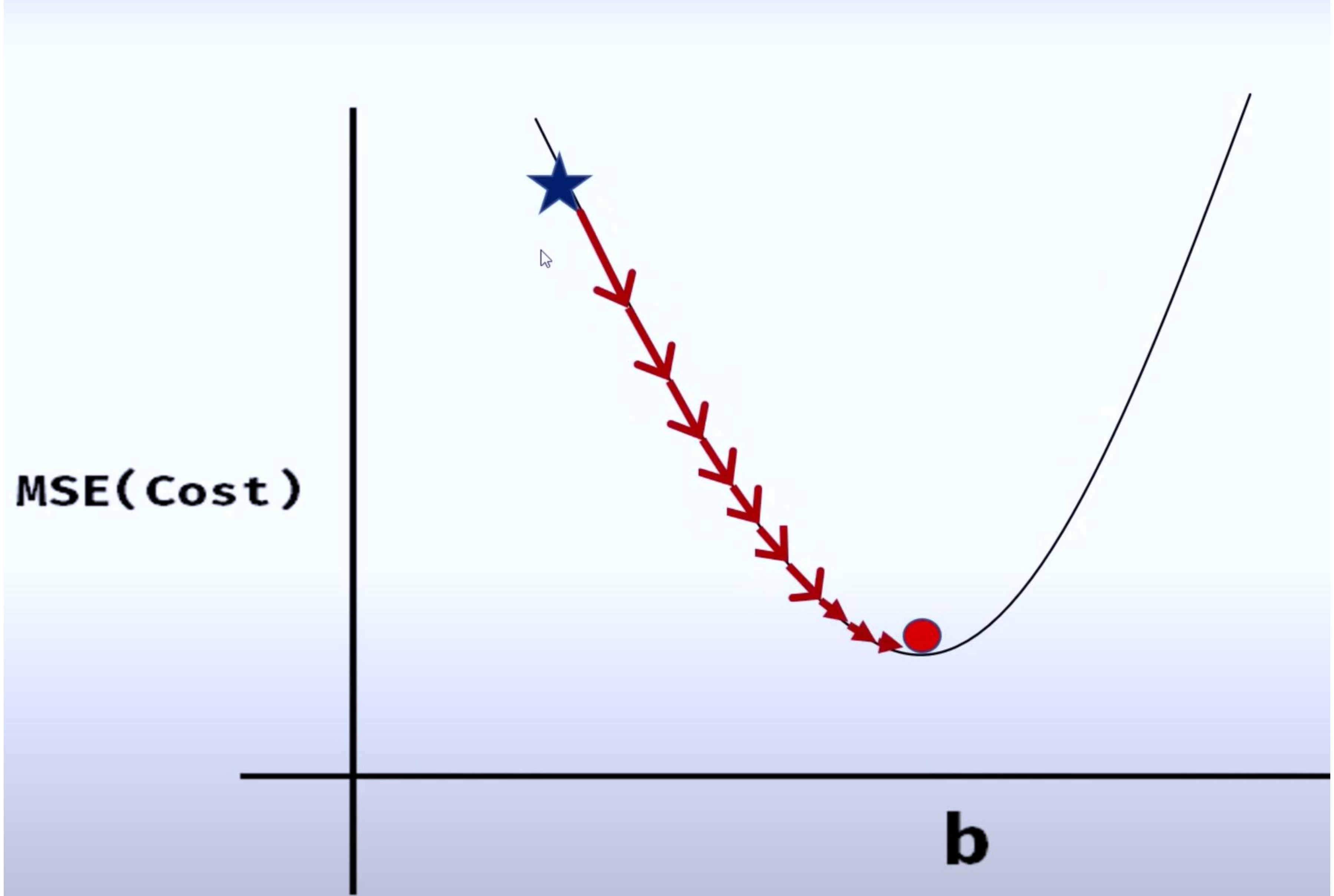




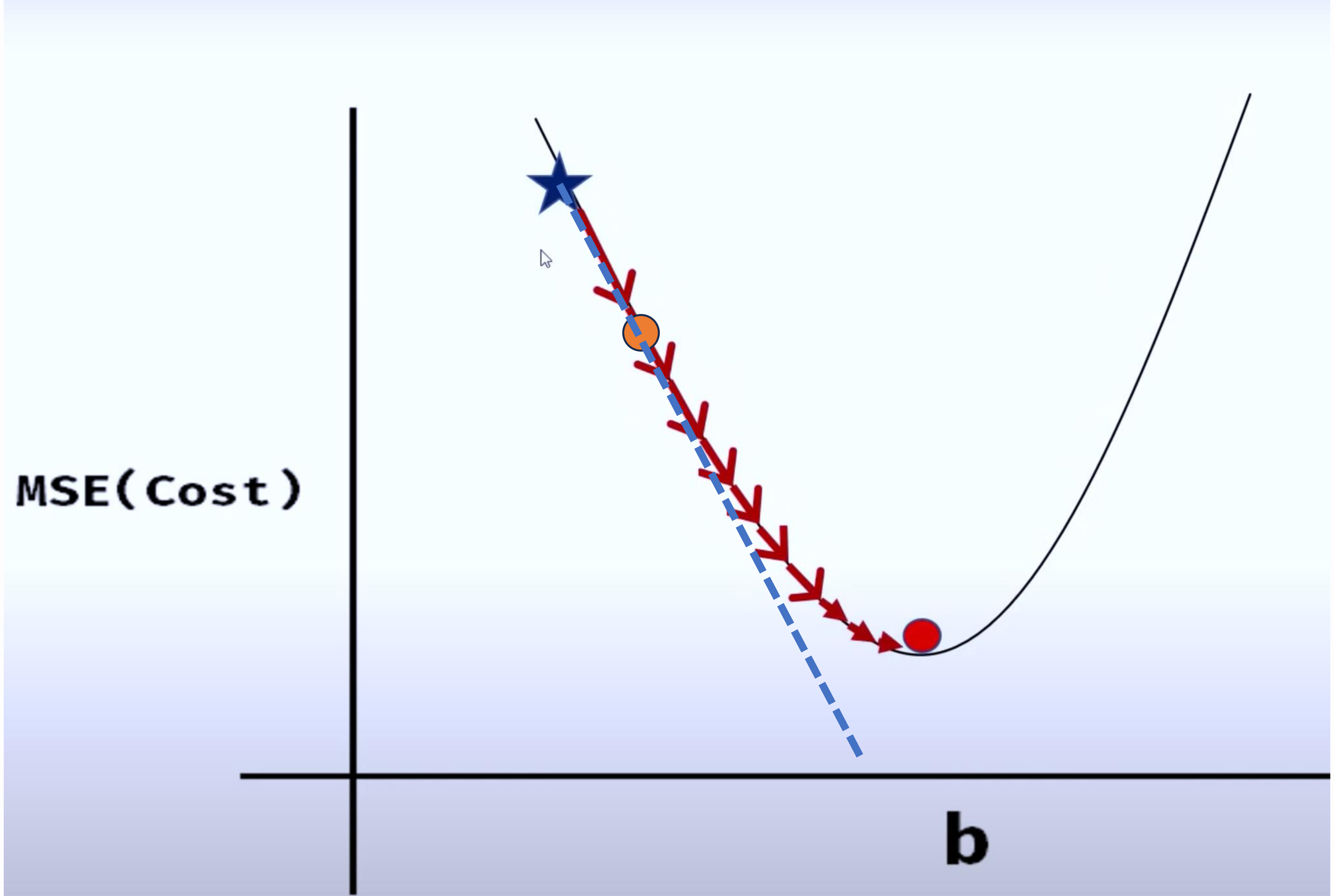
Gradient Descent and Steps Towards Minima



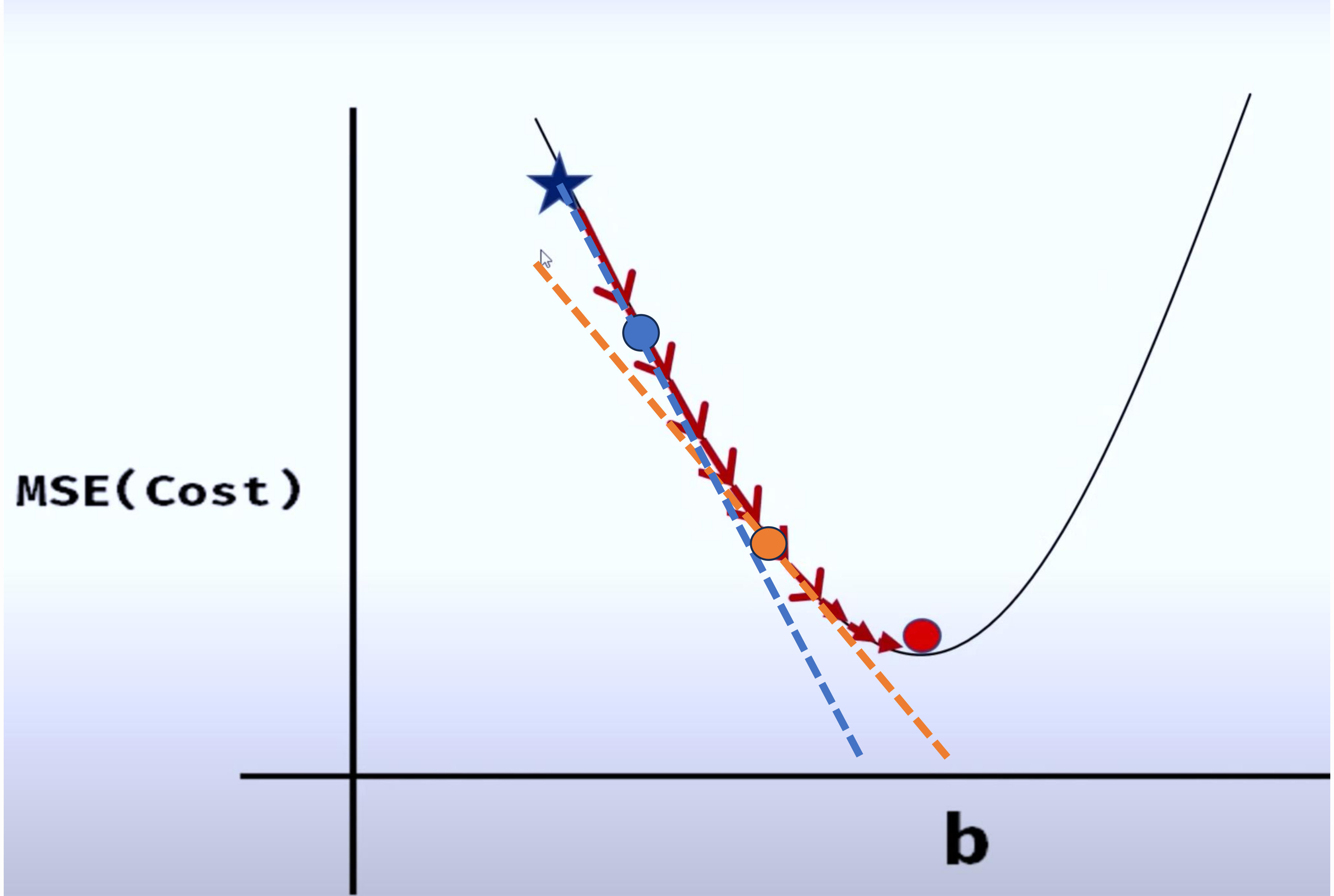
Gradient Descent and Steps Towards Minima



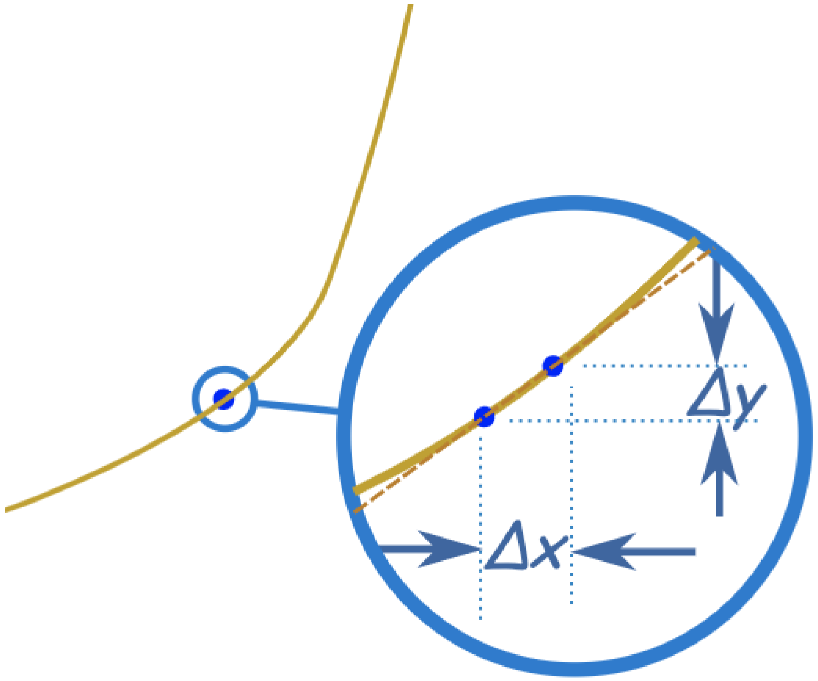
Gradient Descent and Steps Towards Minima

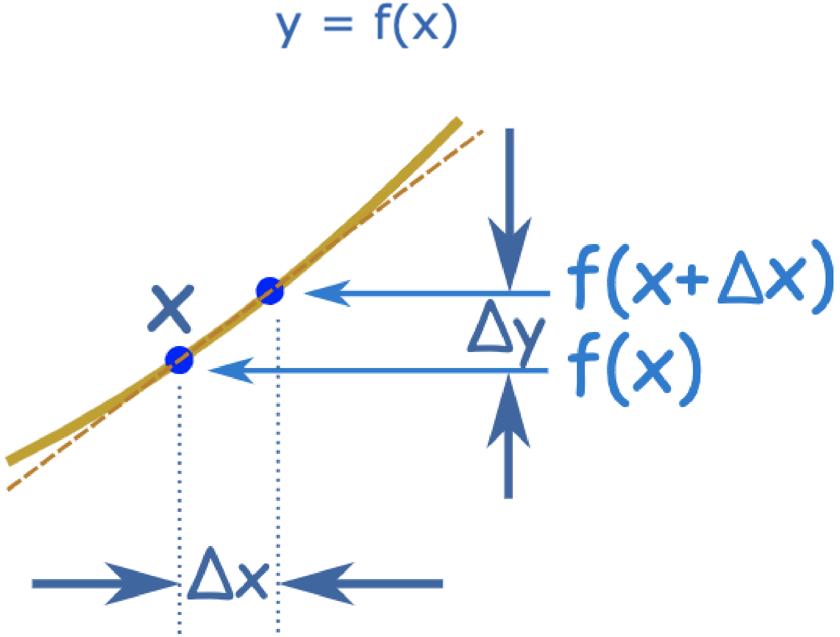


Gradient Descent and Steps Towards Minima



Basic Math – Let us revisit the derivatives

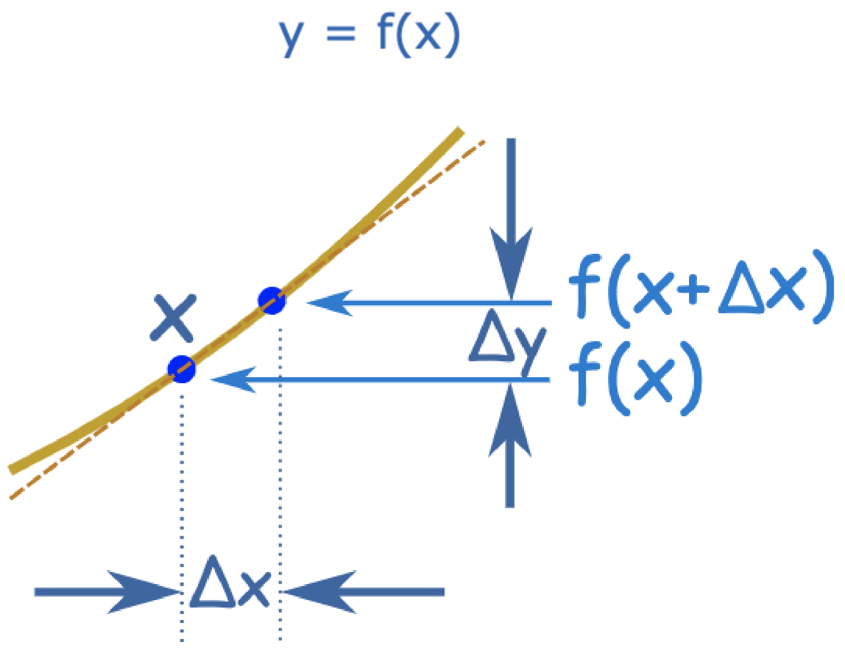




$$\frac{\Delta y}{\Delta x} = \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

Source credit - https://www.mathsisfun.com/calculus/derivatives-dy-dx.html

Basic Math – Let us revisit the derivatives



$$\frac{\Delta y}{\Delta x} = \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

As Δx tends to zero, $\Delta x \rightarrow dx$

$$\frac{dy}{dx} = \frac{f(x + dx) - f(x)}{dx}$$

Source credit - https://www.mathsisfun.com/calculus/derivatives-dy-dx.html

Basic Math – Let us revisit the derivatives

$f(x) = x^2$

$$\frac{dy}{dx} = \frac{f(x + dx) - f(x)}{dx}$$

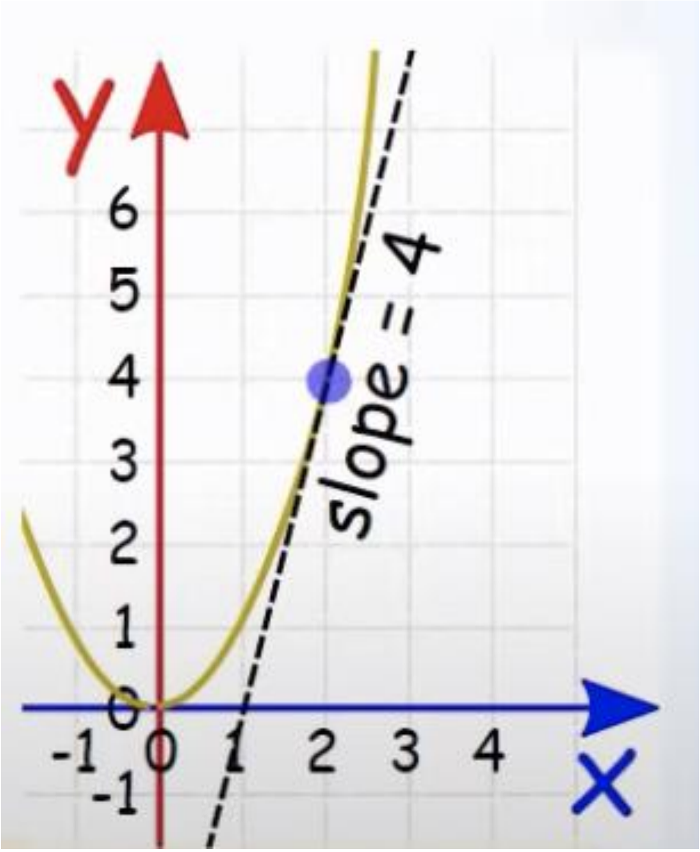
$$= \frac{(x + dx)^2 - x^2}{dx}$$ $f(x) = x^2$

$$= \frac{x^2 + 2x(dx) + (dx)^2 - x^2}{dx}$$ Expand $(x+dx)^2$

$$= \frac{2x(dx) + (dx)^2}{dx}$$ $x^2 - x^2 = 0$

$$= 2x + dx$$ Simplify fraction

$$= 2x$$ dx goes towards 0



Source credit - https://www.mathsisfun.com/calculus/derivatives-dy-dx.html

Recap - Partial derivatives

$$f(x, y) = x^3 + y^2$$

$$\partial f / \partial x = 3x^2 + 0 = 3x^2$$

$$\partial f / \partial y = 0 + 2y = 2y$$

Now let us calculate the slope

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - (mx_i + b)\right)^2$$

$$\partial/_{\partial m} = \frac{2}{\mathrm{n}}\sum_{i=1}^{n} -x_i\left(y_i - (mx_i + b)\right)$$

$$\partial/_{\partial b} = \frac{2}{\mathrm{n}}\sum_{i=1}^{n} -\left(y_i - (mx_i + b)\right)$$

1. Partial derivative with respect to m

Let

$$u = y - (mx + b).$$

Then

$$f(m, b) = u^2.$$

Step-by-step

1. Use the chain rule:

$$\frac{\partial f}{\partial m} = \frac{\partial}{\partial m}\left[u^2\right] = 2\,u \cdot \frac{\partial u}{\partial m}.$$

2. Compute $\frac{\partial u}{\partial m}$:

$$u = y - (mx + b) \quad \Longrightarrow \quad \frac{\partial u}{\partial m} = \frac{\partial}{\partial m}[y - (mx + b)] = -x.$$

3. Combine the results:

$$\frac{\partial f}{\partial m} = 2\left(y - (mx + b)\right) \cdot (-x) = -2x\left(y - (mx + b)\right).$$

2. Partial derivative with respect to b

Again using $u = y - (mx + b)$:

$$\frac{\partial f}{\partial b} = \frac{\partial}{\partial b}\left[u^2\right] = 2\,u \cdot \frac{\partial u}{\partial b}.$$

1. Compute $\frac{\partial u}{\partial b}$:

$$u = y - (mx + b) \quad \Longrightarrow \quad \frac{\partial u}{\partial b} = \frac{\partial}{\partial b}[y - (mx + b)] = -1.$$

2. Combine the results:

$$\frac{\partial f}{\partial b} = 2\,\bigl(y - (mx + b)\bigr) \cdot (-1) = -2\,\bigl(y - (mx + b)\bigr).$$

Final Answers

$$\frac{\partial}{\partial m}\left(y-(mx+b)\right)^2=-2x\left(y-(mx+b)\right),$$

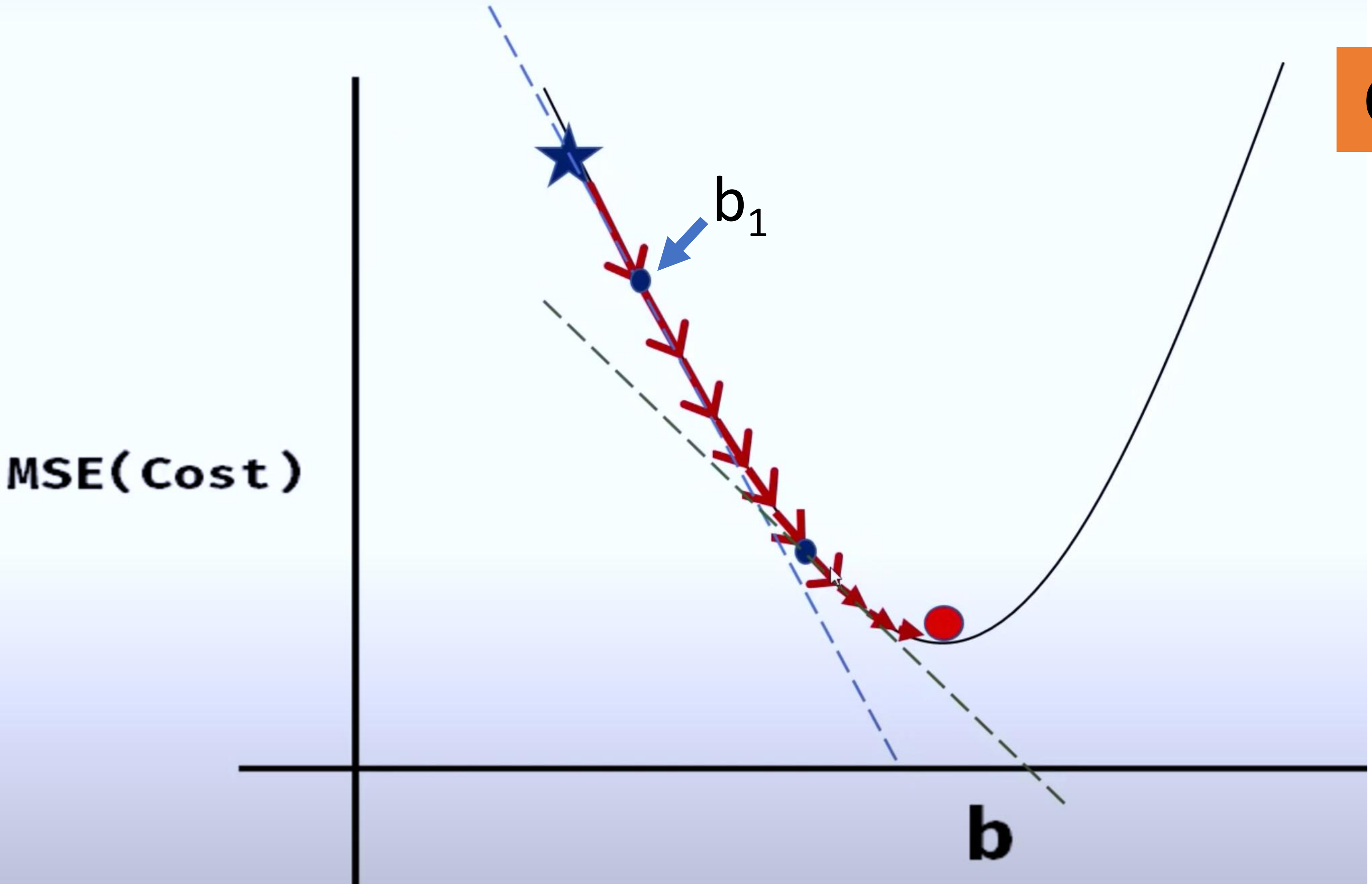
$$\frac{\partial}{\partial b}\left(y-(mx+b)\right)^2=-2\left(y-(mx+b)\right).$$

Concept of Learning Rate

$$\mathrm{m} = \mathrm{m} - \text{learning rate} * {}^{\partial}/_{\partial m}$$

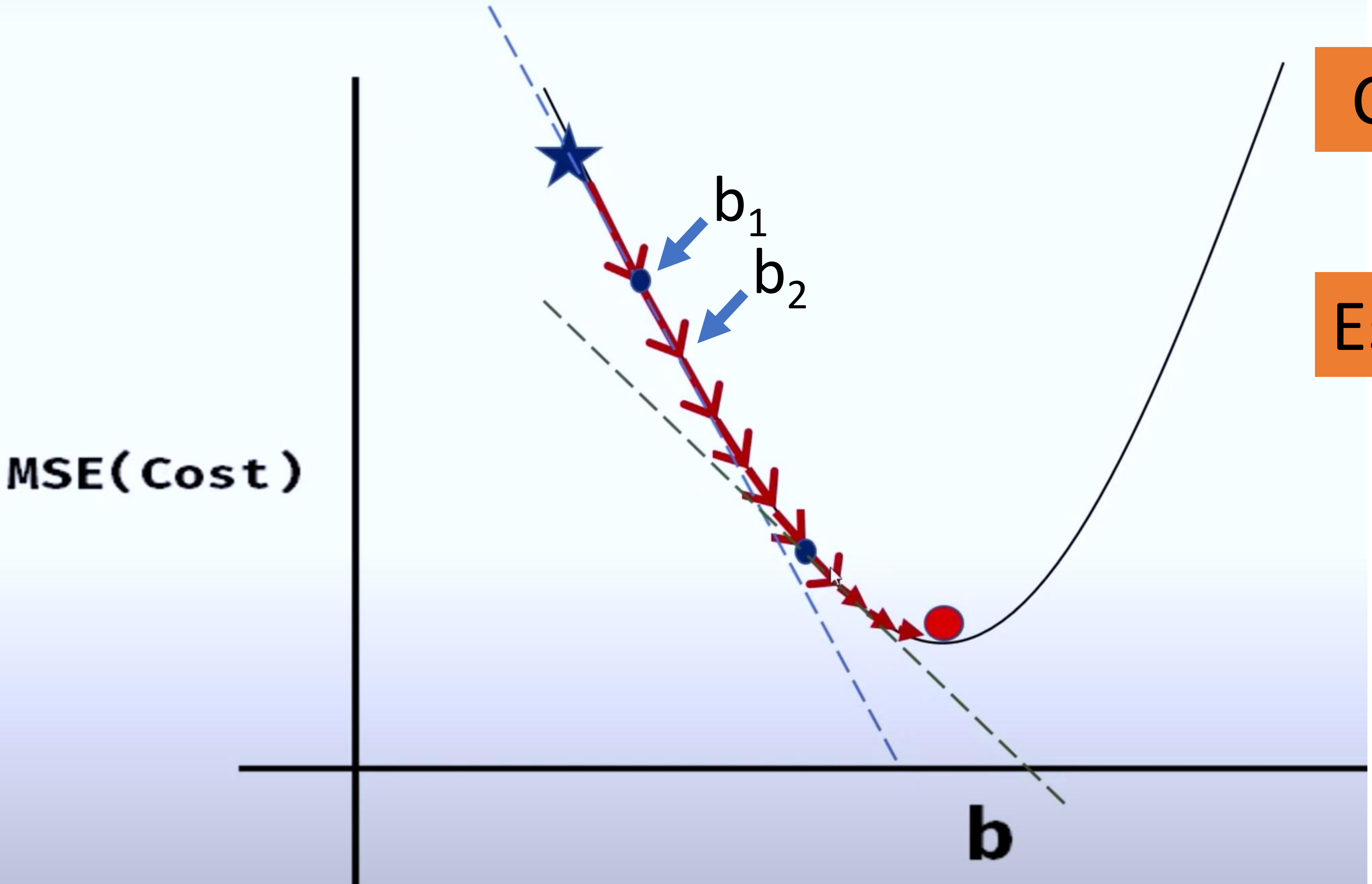
$$\mathrm{b} = \mathrm{b} - \text{learning rate} * {}^{\partial}/_{\partial b}$$

How does Gradient Descent work?



Compute the Slope

How does Gradient Descent work?

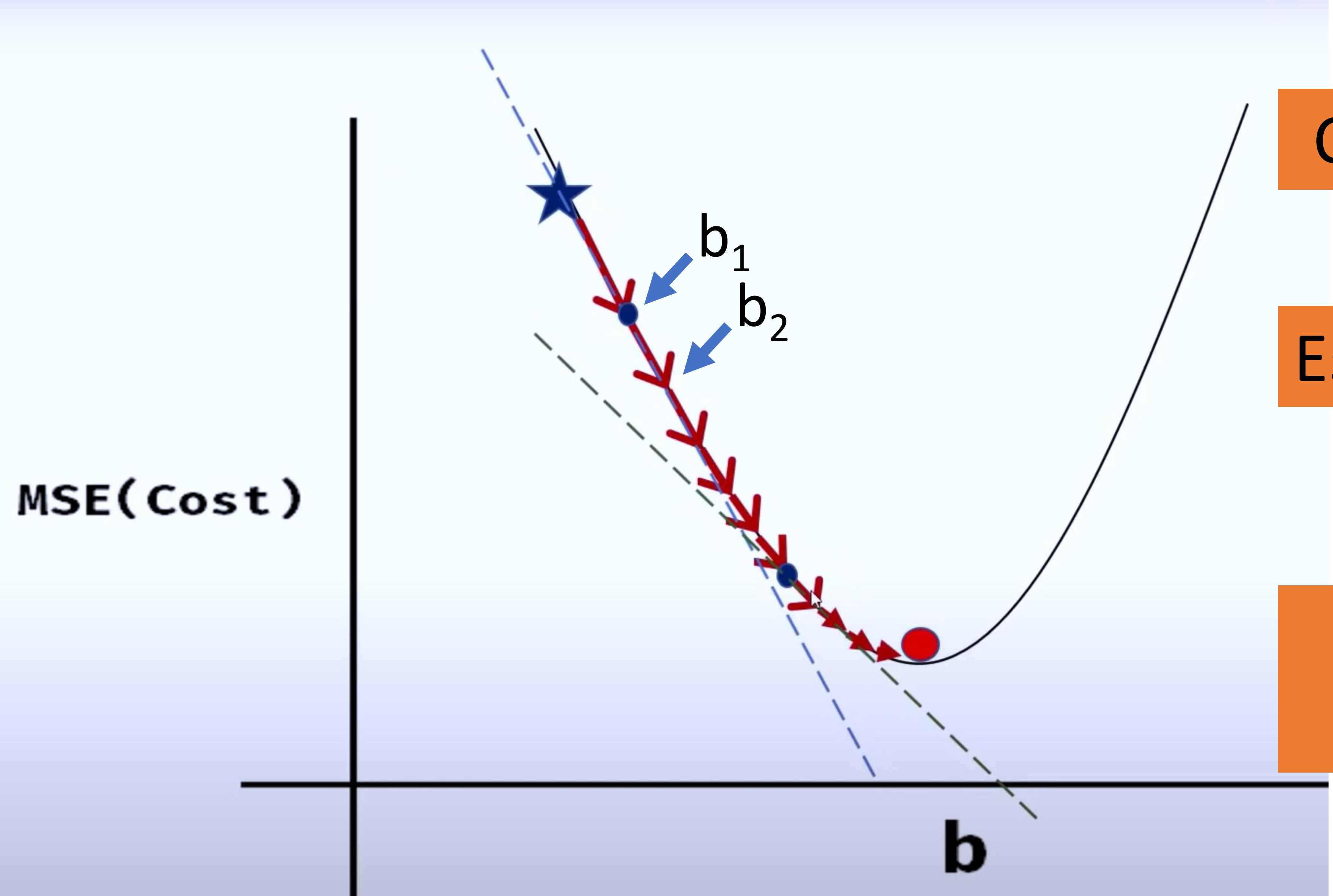


Compute the Slope

Estimate Step Size b_2

$b_2 = b1 - \text{learning rate} * \partial/\partial b$

How does Gradient Descent work?



Compute the Slope

Estimate Step Size b_2

$$b_2 = b1 - \text{learning rate} * \partial/\partial b$$

Iterate until convergence

Python Implementation

```
In [109]: import numpy as np
          import matplotlib.pyplot as plt
```

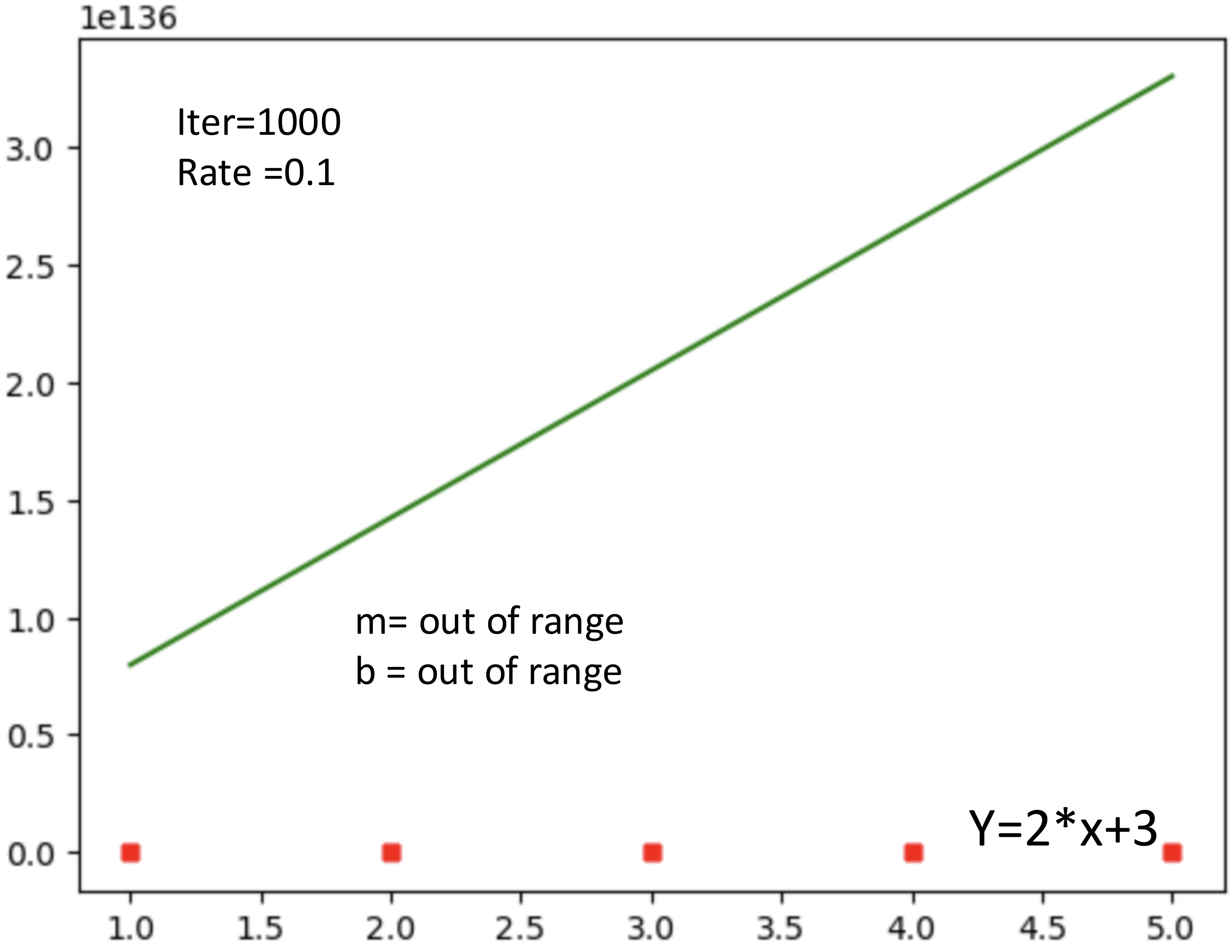
```
In [117]: %matplotlib inline
          def gradient_descent(x,y):
              m_curr = b_curr = 0
              rate = 0.008
              iter=1000
              n = len(x)
              plt.scatter(x,y,color='red',marker='+',linewidths=5)
              for i in range(iter):
                  y_predicted = m_curr * x + b_curr
                  #print (m_curr,b_curr, i)
                  plt.plot(x,y_predicted,color='green')
                  md = -(2/n)*sum(x*(y-y_predicted))
                  yd = -(2/n)*sum(y-y_predicted)
                  m_curr = m_curr - rate * md
                  b_curr = b_curr - rate * yd
```

```
In [ ]:
```

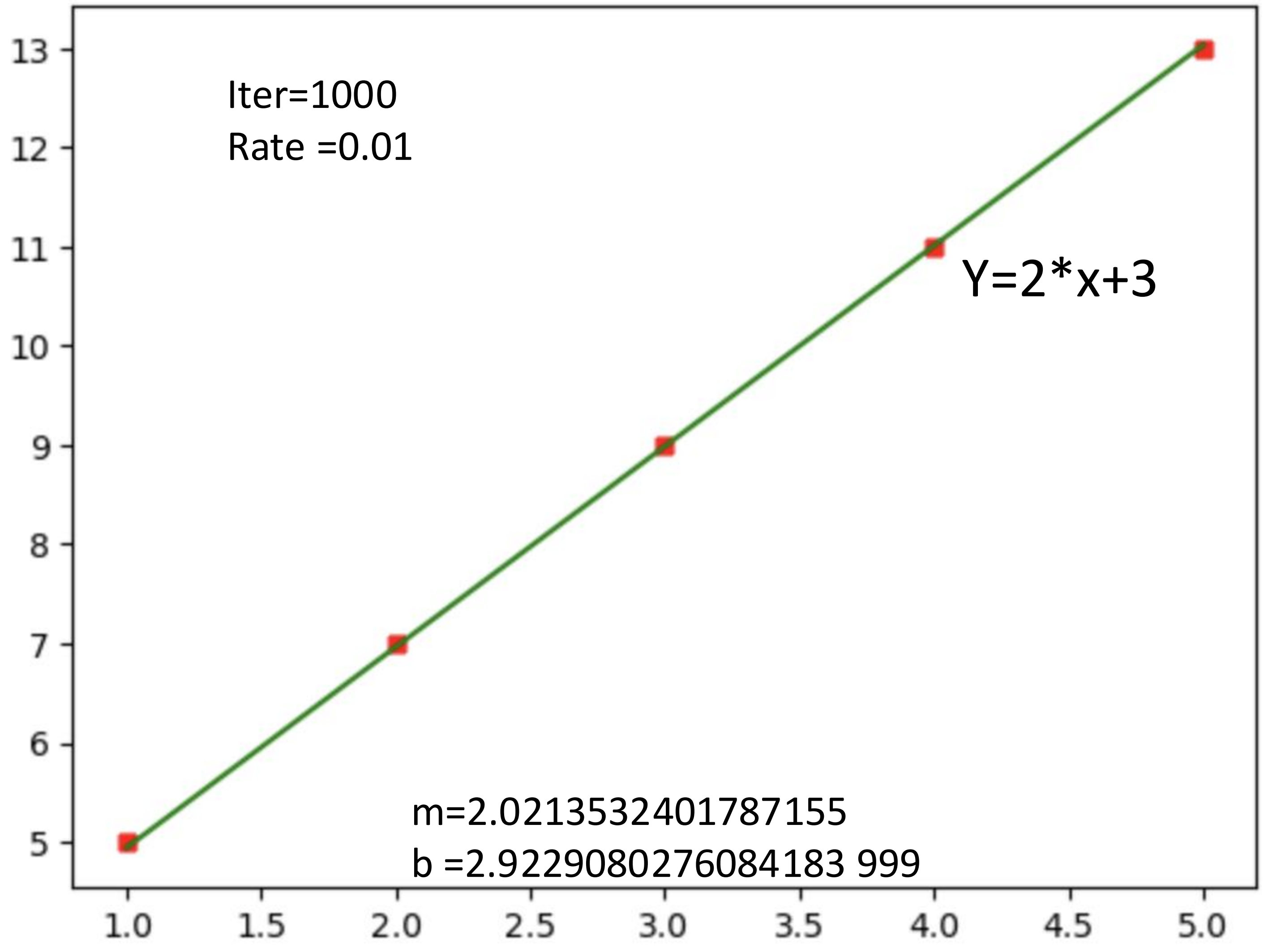
```
In [118]: #Create an array of independent variable x and dependent variable y based on a known linear model (y=m*x+c)

          x = np.array([1,2,3,4,5])
          y = np.array([5,7,9,11,13])

          y.shape
          #Solution for this (y=m*x+b) are m=2.0 b=3.0

Out[118]: (5,)
```

```
In [119]: gradient_descent(x,y)
```

Python Implementation

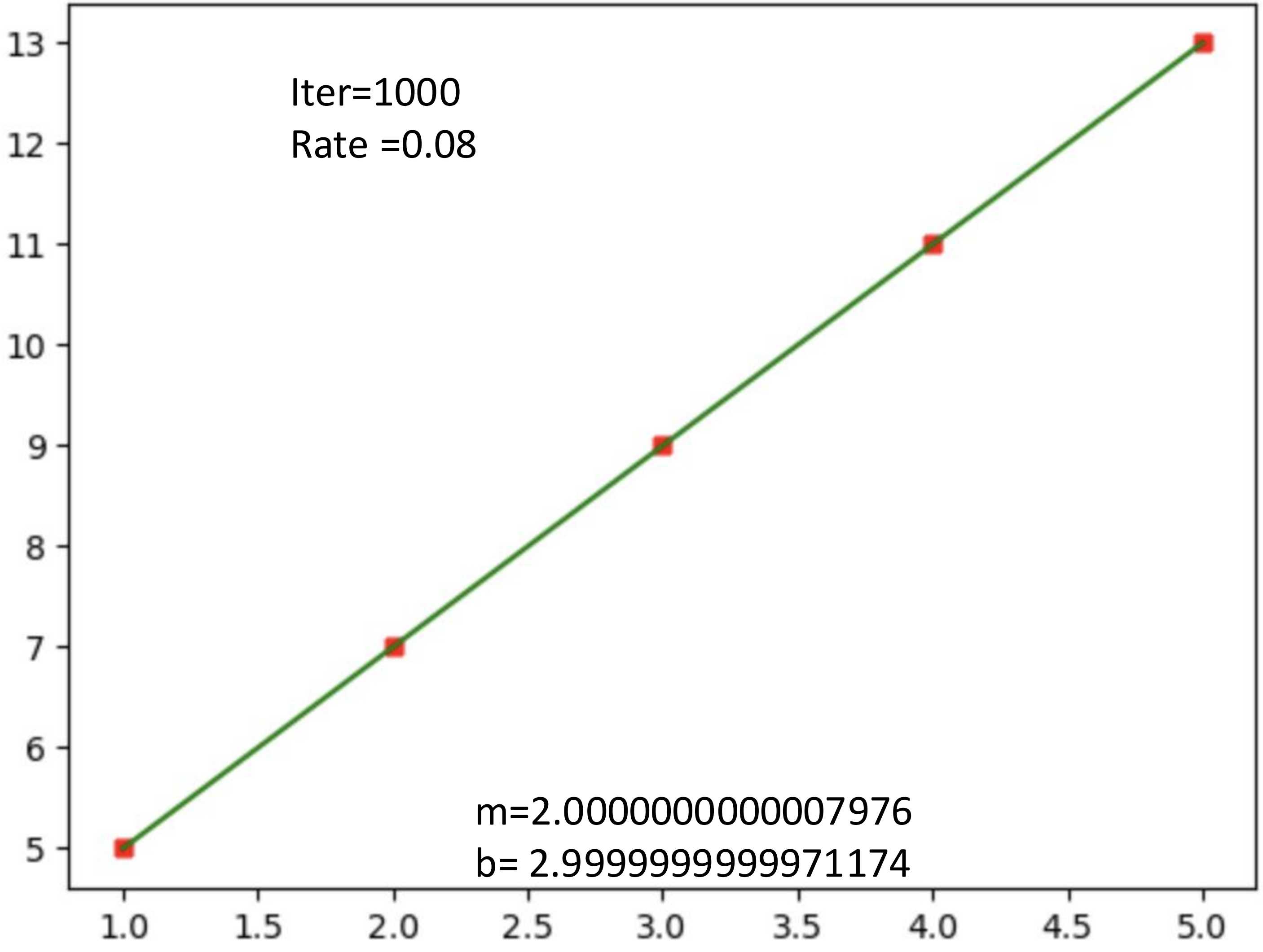


Python Implementation

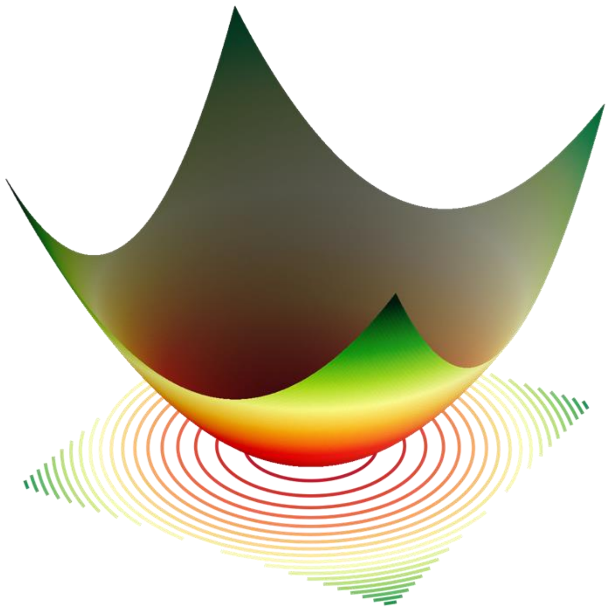


Python Implementation

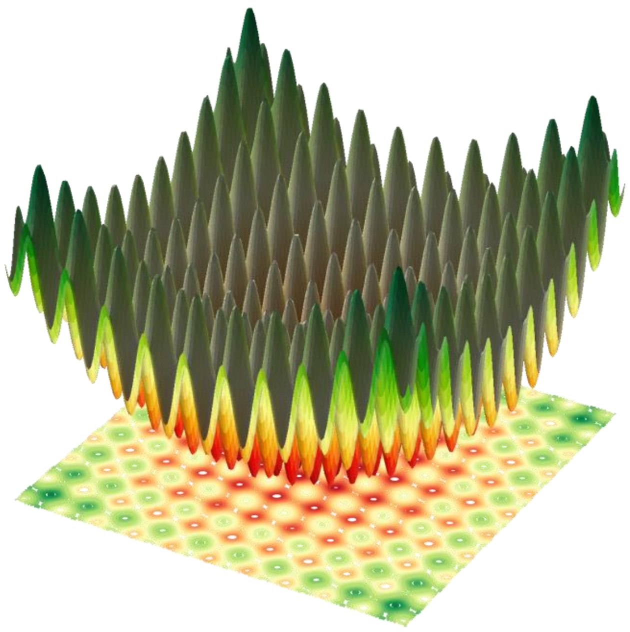
Gradient descent *is a* *<u>first-order</u>* *<u>iterative</u>* *<u>optimization</u>* *<u>algorithm</u>* *for finding the minimum of a function*



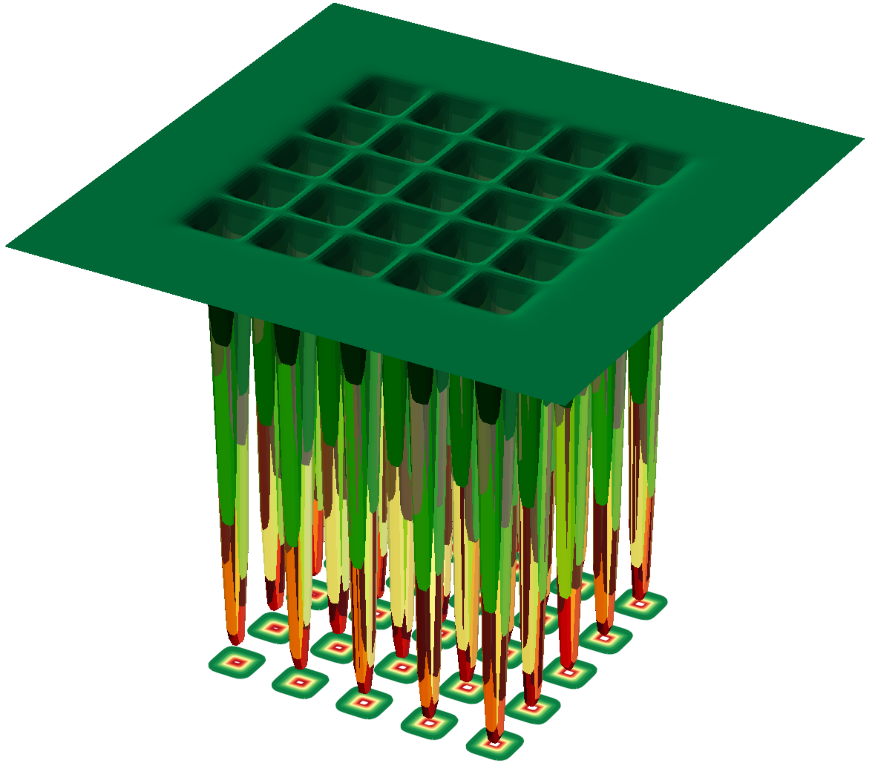
Most problems in practice



Sphere (Unimodal)



Rastrigin (Multimodal)



Dejong N5 (Fixed Dimension)

Need ways to implement gradient descent more efficiently

Stochastic vs. Batch vs. Mini batch Gradient Descent

HV	C.al	C.co
170	0.056604	0.000000
380	0.200000	0.266667
775	0.400000	0.200000
486	0.208333	0.000000
118	0.024390	0.243902

Machine Learning Model

Hardness = w1*C.al+w2*C.co+bias

Gradient Descent

Initialize w_1=1, w_2=1, bias=1

HV	C.al	C.co
170	0.056604	0.000000
380	0.200000	0.266667
775	0.400000	0.200000
486	0.208333	0.000000
118	0.024390	0.243902

Machine Learning Model

Hardness = w_1*C.al+w_2*C.co+bias

Sample 1

HV_predicted =1.0566

HV_true = 170

Error_1=(Hv_true-HV_predicted)^2

Gradient Descent

Initialize w_1=1, w_2=1, bias=1

HV	C.al	C.co
170	0.056604	0.000000
380	0.200000	0.266667
775	0.400000	0.200000
486	0.208333	0.000000
118	0.024390	0.243902

Machine Learning Model

Hardness = w_1*C.al+w_2*C.co+bias

Sample 2

HV_predicted = 1.4666

HV_true = 380

Error_2=(Hv_true-HV_predicted)^2

Gradient Descent

Initialize $w_1=1$, $w_2=1$, $bias=1$

HV	C.al	C.co
170	0.056604	0.000000
380	0.200000	0.266667
775	0.400000	0.200000
486	0.208333	0.000000
118	0.024390	0.243902

Machine Learning Model

Hardness = w_1*C.al+w_2*C.co+bias

Sample 5

HV_predicted = 1.27

HV_true = 118

Error_5=(Hv_true-HV_predicted)^2

Total Error = Error_1+Error_2+Error_3+Error_4+Error_5

End of First Epoch

Total Error = Error_1+Error_2+Error_3+Error_4+Error_5

Mean Squared Error (MSE) = $\frac{Total\ Error}{5}$

$$\text{w1} = \text{w1} - learning\ rate * {}^{\partial(\text{MSE})}/_{\partial w1}$$

W1= 10

$$\text{w2} = \text{w2} - learning\ rate * {}^{\partial(\text{MSE})}/_{\partial w2}$$

W2 = 23

$$\text{b} = \text{b} - learning\ rate * {}^{\partial(\text{MSE})}/_{\partial b}$$

b = 34

End of Second Epoch

HV	C.al	C.co
170	0.056604	0.000000
380	0.200000	0.266667
775	0.400000	0.200000
486	0.208333	0.000000
118	0.024390	0.243902

Machine Learning Model

Hardness = 10*C.al+23*C.co+34

Weights from the first Epoch

Sample 5

HV_predicted = 39.87

HV_true = 118

$Error\_5 = (Hv\_true - HV\_predicted)^2$

Total Error = Error_1+Error_2+Error_3+Error_4+Error_5

This is Batch Gradient Descent where we go through all the samples and compute total error. This is back propagated and weights are adjusted till convergence is achieved i.e. error is the minimum

Now the challenging part with Batch Mode

Think about neural networks and large ML models

HV	C.al	C.co
170	0.056604	0.000000
380	0.200000	0.266667
775	0.400000	0.200000
486	0.208333	0.000000
118	0.024390	0.243902

Say if you have a million samples

First Epoch would essentially require a forward pass for million samples

If we have 2 features or attributes, we have 2 million derivatives to compute

Remember the dataset last time, where we had 22 features

Too much computations required to converge

Stochastic Gradient Descent

First, randomly pick a single data training sample

Initialize $w_1=1$, $w_2=1$, bias=1

HV	C.al	C.co
170	0.056604	0.000000
380	0.200000	0.266667
775	0.400030	0.200000
486	0.208333	0.000000
118	0.024390	0.243902

Machine Learning Model

Hardness = w_1*C.al+w_2*C.co+bias

Sample 3

HV_predicted = 1.6

HV_true = 775

$$\text{Error}=(\text{Hv\_true}-\text{HV\_predicted})^2$$

Stochastic Gradient Descent

Next, adjust the weights by computing the derivatives

$$w1 = w1 - \textit{learning rate} * {}^{\partial(\mathrm{MSE})}/_{\partial w1}$$

$$w2 = w2 - \textit{learning rate} * {}^{\partial(\mathrm{MSE})}/_{\partial w2}$$

$$b = b - \textit{learning rate} * {}^{\partial(\mathrm{MSE})}/_{\partial b}$$

Stochastic Gradient Descent – iteration 2

Again, randomly pick a single data training sample

Weights from previous forward pass

HV	C.al	C.co
170	0.056604	0.000000
380	0.200000	0.266667
775	0.400000	0.200000
486	0.208333	0.000000
118	0.024390	0.243902

Machine Learning Model

Hardness = w_1*C.al+w_2*C.co+bias

Sample 4

HV_predicted = XXX

HV_true = 486

Error=(Hv_true-HV_predicted)^2

Stochastic Gradient Descent – iteration 2

Again, adjust the weights by computing the derivatives

$$w1 = w1 \ - learning\,rate \ * \ {\partial(\mathrm{MSE})}/{\partial w1}$$

$$w2 = w2 \ - learning\,rate \ * \ {\partial(\mathrm{MSE})}/{\partial w2}$$

$$b = b \ - learning\,rate \ * \ {\partial(\mathrm{MSE})}/{\partial b}$$

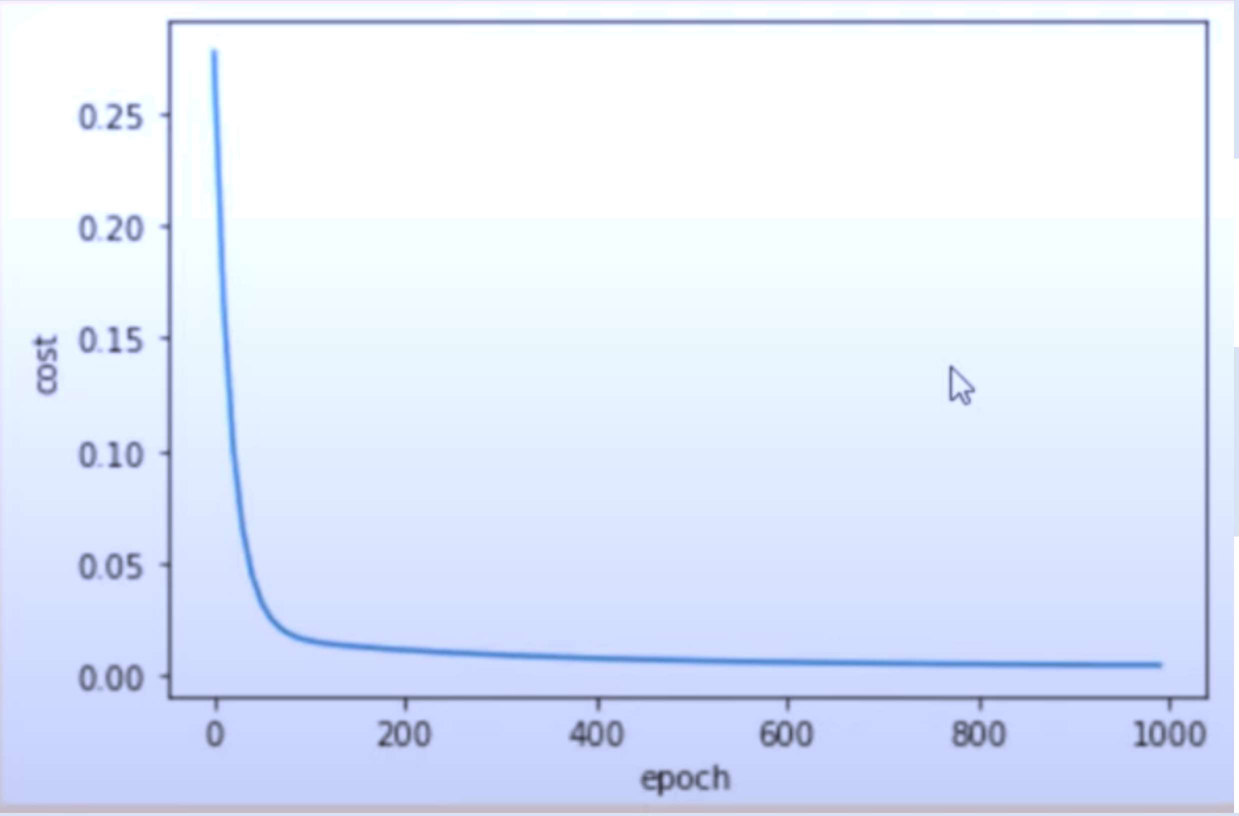
Continue this till convergence

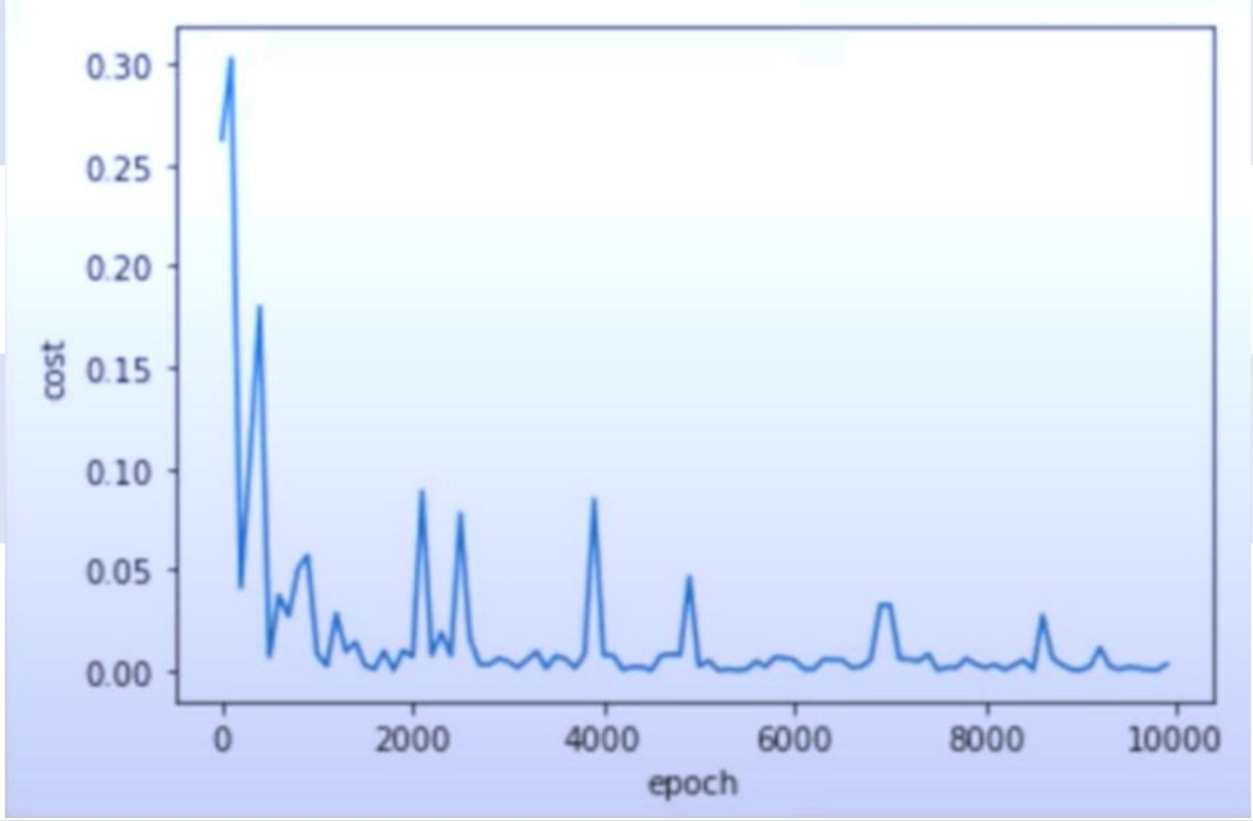
Stochastic vs. Batch Gradient Descent

Batch Gradient Descent	Stochastic Gradient Descent
All the samples are used for one forward pass	One sample is picked randomly for one forward pass
Good for smaller training data set	Good for large training data set. Saves compute time
Great for convex, or relatively smooth error manifolds or surfaces	SGD works well for error manifolds that have lots of local maxima/minima
Struggles when lot of local minima are present – eventually converges provided large enough compute times	Noisier gradients tend to avoid sub-optimal local minima

Stochastic vs. Batch Gradient Descent

Batch Gradient Descent	Stochastic Gradient Descent





Mini Batch Gradient Descent

Mini Batch is similar to a stochastic gradient descent. However, instead on one randomly chosen sample, we pick (randomly) a small batch of samples.

1. If the total samples are around 200
2. Let us pick 10 random samples for one forward pass to compute the cummulative error
3. Adjust the weights
4. Choose again 10 random samples
5. Adjust weights

:

:

Convergence is achieved

Summary

Batch gradient Descent	Stochastic Gradient Descent	Mini Batch Gradient Descent
Use **all** the training data or samples for one forward pass and adjust weights	Use **one randomly picked** training data or samples for one forward pass and adjust weights	Use **a batch of randomly picked** training data or samples for one forward pass and adjust weights

Next lecture

Python implementation of stochastic and batch gradient descent