Machine Learning

Lecture 2: Training Models

Asst. Prof. Dr. Santitham Prom-on

Department of Computer Engineering, Faculty of Engineering
King Mongkut's University of Technology Thonburi

Topics

- Direct approach: linear regression with ordinary least square
- Iterative approach
  - Gradient descent
  - Batch gradient descent
  - Stochastic gradient descent
  - Mini-batch gradient descent
- Logistic regression

Notebook:
https://colab.research.google.com/drive/1jcPGje7ymfGZLL9C2OV0VdqJQXWpe7h9?usp=sharing

Model training

Two different ways

- Using a direct “closed-form” equation.
- Using an iterative optimization approach.

Direct approach

Linear regression with Ordinary Least Square (OLS)

- Linear regression model can be expressed by the following formula

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$$

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n + \varepsilon$$

where

$\hat{y}$ is the predicted value

n is the number of features

x_i is the ith feature value

Simple linear regression

$$Y_i = \alpha + \beta X_i + \varepsilon_i$$
$$e_i \sim N(0, \sigma^2) \quad i.i.d.$$
$$\varepsilon_i \text{ is independent of } X_i$$

- The intercept is α
- The slope is β
- We use the normal distribution to describe the “error”

Method of least squares

- Choose the β's so that the sum of the squares of the errors, ε_i, are minimized
- The least squares function is

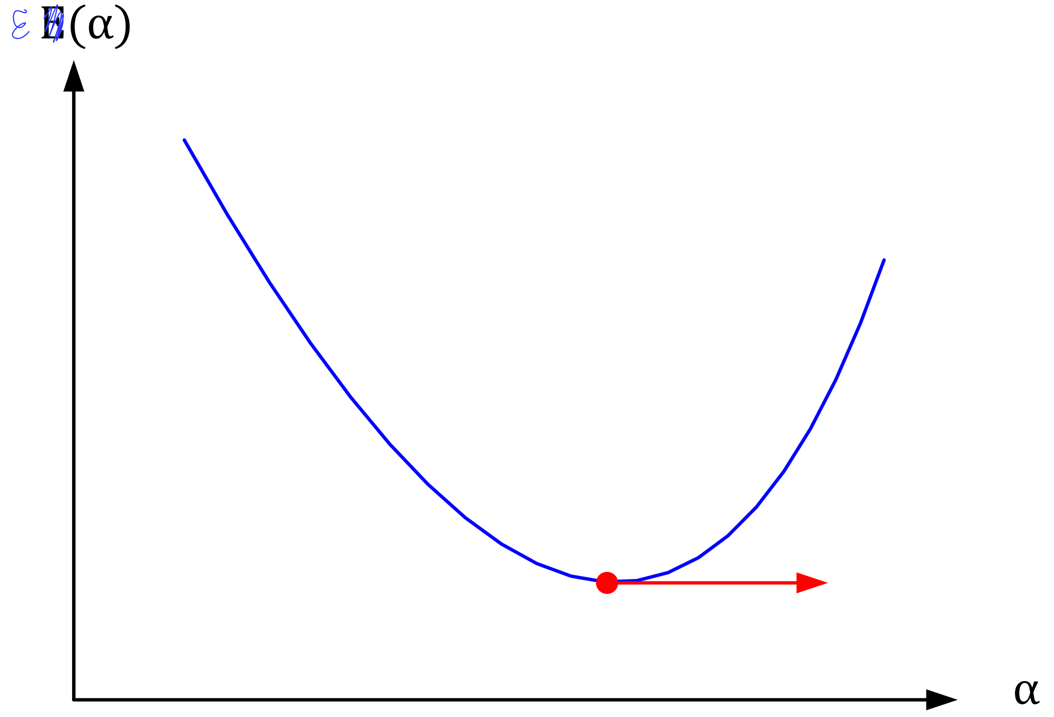
$$S = \sum_{i=1}^{n} \varepsilon_i^2$$

$$= \sum_{i=1}^{n} \left(y_i - \alpha - \beta x_i \right)^2$$

OLS solution

Minimum of a function is the point where the slope is zero

$\varepsilon_i(\alpha) = Y - \beta X_i - \alpha$



Derivative of the error functions

The function S is to be minimized with respect to β_0, β_1

and

$$\beta_0 = \frac{\partial S}{\partial \alpha} = -2\sum_{i=1}^{n}\left(y_i - \alpha - \beta x_i\right) = 0$$

$$\beta_1 = \frac{\partial S}{\partial \beta} = -2\sum_{i=1}^{n}\left(y_i - \alpha - \beta x_i\right)x_i = 0$$

Least square normal equation

$$n\alpha + \beta\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i$$

$$\alpha\sum_{i=1}^{n} x_i + \beta\sum_{i=1}^{n} x_i^2 = \sum_{i=1}^{n} x_i y_i$$

Find alpha (intercept)

$$\alpha = \frac{\begin{vmatrix} \sum_{i=1}^{n} y_i & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i y_i & \sum_{i=1}^{n} x_i^2 \end{vmatrix}}{\begin{vmatrix} n & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 \end{vmatrix}} = \frac{\sum_{i=1}^{n} x_i^2 \sum_{i=1}^{n} y_i - \sum_{i=1}^{n} x_i y_i \sum_{i=1}^{n} x_i}{n \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i \right)^2}$$

Find beta (slope)

$$\beta = \frac{\begin{vmatrix} n & \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i y_i \end{vmatrix}}{\begin{vmatrix} n & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 \end{vmatrix}} = \frac{n\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}$$

Example: Multiple regression

x_1	x_2	y
1	2	12
2	1	9
3	2	19
1	1	8

$$y = \overset{W}{c_1} x_1 + \overset{W}{c_2} x_2 + \overset{C}{c_3}$$

$$\begin{aligned}
\textcircled{1}\quad c_1 + 2c_2 + c_3 &= 12 \\
\textcircled{2}\quad 2c_1 + c_2 + c_3 &= 9 \\
\textcircled{3}\quad 3c_1 + 2c_2 + c_3 &= 19 \\
\textcircled{4}\quad c_1 + c_2 + c_3 &= 8
\end{aligned}$$

$$\underset{}{\overset{A}{\begin{bmatrix} 1 & 2 & 1 \\ 2 & 1 & 1 \\ 3 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix}}} \overset{x}{\begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix}} = \begin{bmatrix} 12 \\ 9 \\ 19 \\ 8 \end{bmatrix}$$

Pseudoinverse

$\mathbf{Y}$	$N \times 1$ vector
$\mathbf{A}$	$N \times M$ matrix, where M is the number of parameters
$\mathbf{B}$	$M \times 1$ vector

$$\mathbf{Y} = \mathbf{A}\mathbf{B}$$

$$\mathbf{A}\mathbf{B} = \mathbf{Y}$$

$$\overset{m \times m \text{ มิติ}}{\underline{\mathbf{A}^{\mathrm{T}}\mathbf{A}}}\mathbf{B} = \mathbf{A}^{\mathrm{T}}\mathbf{Y}$$

$$\underbrace{\left(\mathbf{A}^{\mathrm{T}}\mathbf{A}\right)^{-1}} \mathbf{A}^{\mathrm{T}}\mathbf{A}\mathbf{B} = \left(\mathbf{A}^{\mathrm{T}}\mathbf{A}\right)^{-1} \mathbf{A}^{\mathrm{T}}\mathbf{Y}$$

$$\mathbf{B} = \left(\mathbf{A}^{\mathrm{T}}\mathbf{A}\right)^{-1} \mathbf{A}^{\mathrm{T}}\mathbf{Y}$$

Python: pseudoinverse

```
A = np.array([[1,2,1],
              [2,1,1],
              [3,2,1],
              [1,1,1]])
print(A)
```

```
[[1 2 1]
 [2 1 1]
 [3 2 1]
 [1 1 1]]
```

```
B = np.array([12,9,19,8])
B.shape = (-1,1)
print(B)
```

```
[[12]
 [ 9]
 [19]
 [ 8]]
```

```
np.matmul(np.linalg.pinv(A),B)
```

```
array([[ 3. ],
       [ 5.5],
       [-1.5]])
```

Iterative approach Gradient descent

- A very generic optimization algorithm capable of finding optimal solutions to a wide range of problems.

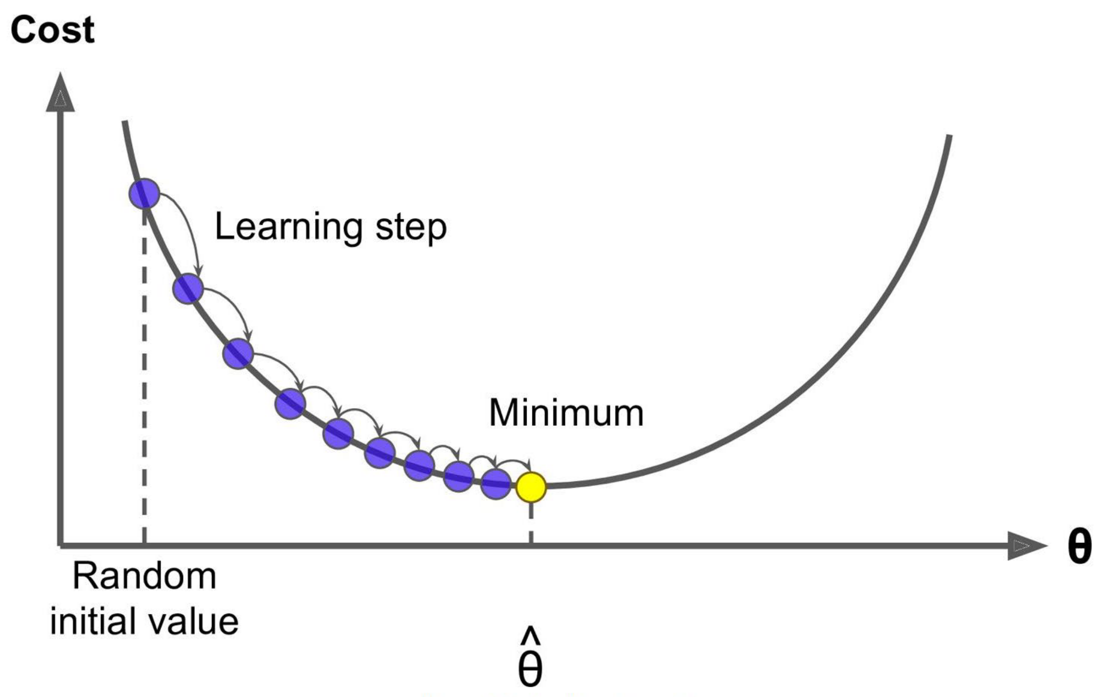


Figure 4-3. Gradient Descent

Suppose you are lost in the mountains in a dense fog; you can only feel the slope of the ground below your feet.

A good strategy to get to the bottom of the valley quickly is to go downhill in the direction of the steepest slope.

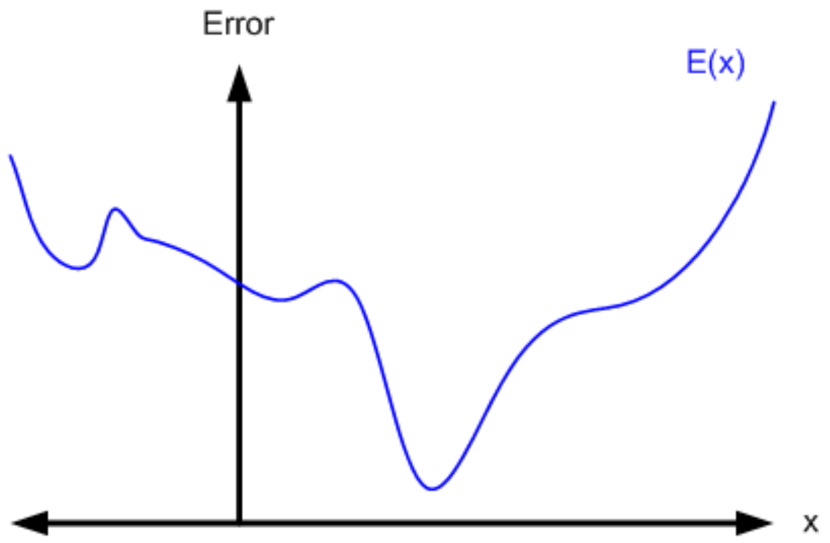
Nonlinear Least Square

- Suppose that we have a sample of n observations on the response and the regressor, say, y_i, x_{i1}, x_{i2}, …, x_{ik} for $i=1,2,\ldots,n$
- The least square method involves minimizing the least square function

$$S(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left[y_i - f(\mathbf{x_i}, \boldsymbol{\beta}) \right]^2$$

Nonlinear objective function

1 Non-linear objective function



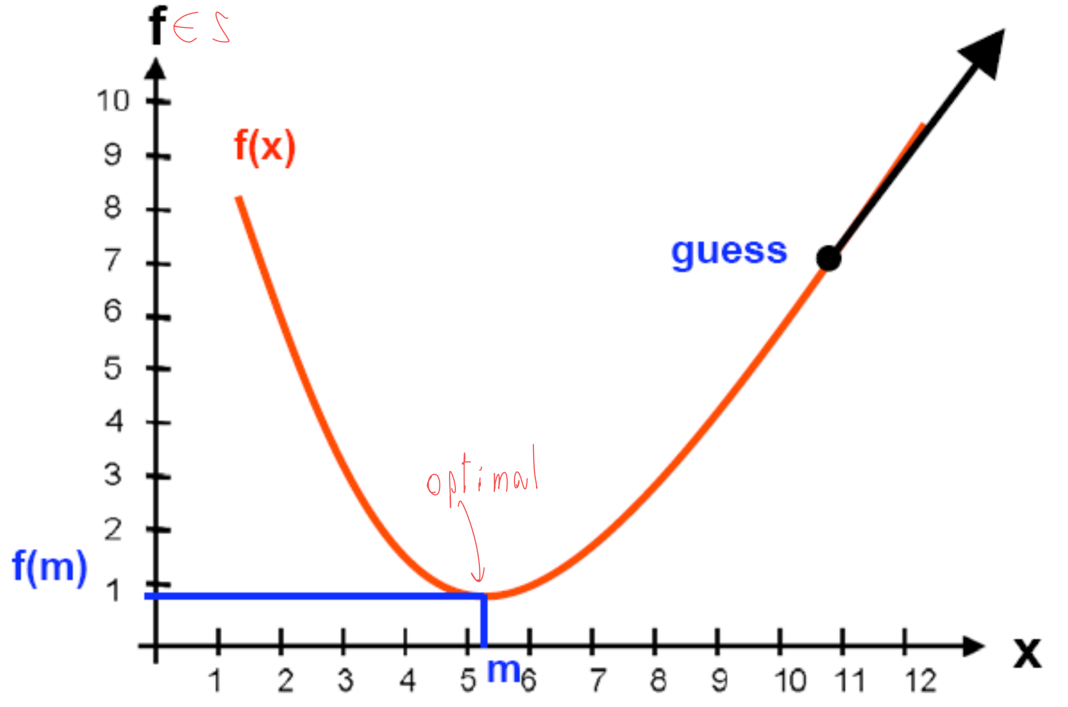
2 Unsolvable for roots of derivative of error

$$\frac{dE}{dx} = e^{-x}$$

Gradient Descent Basic 1 Objective function

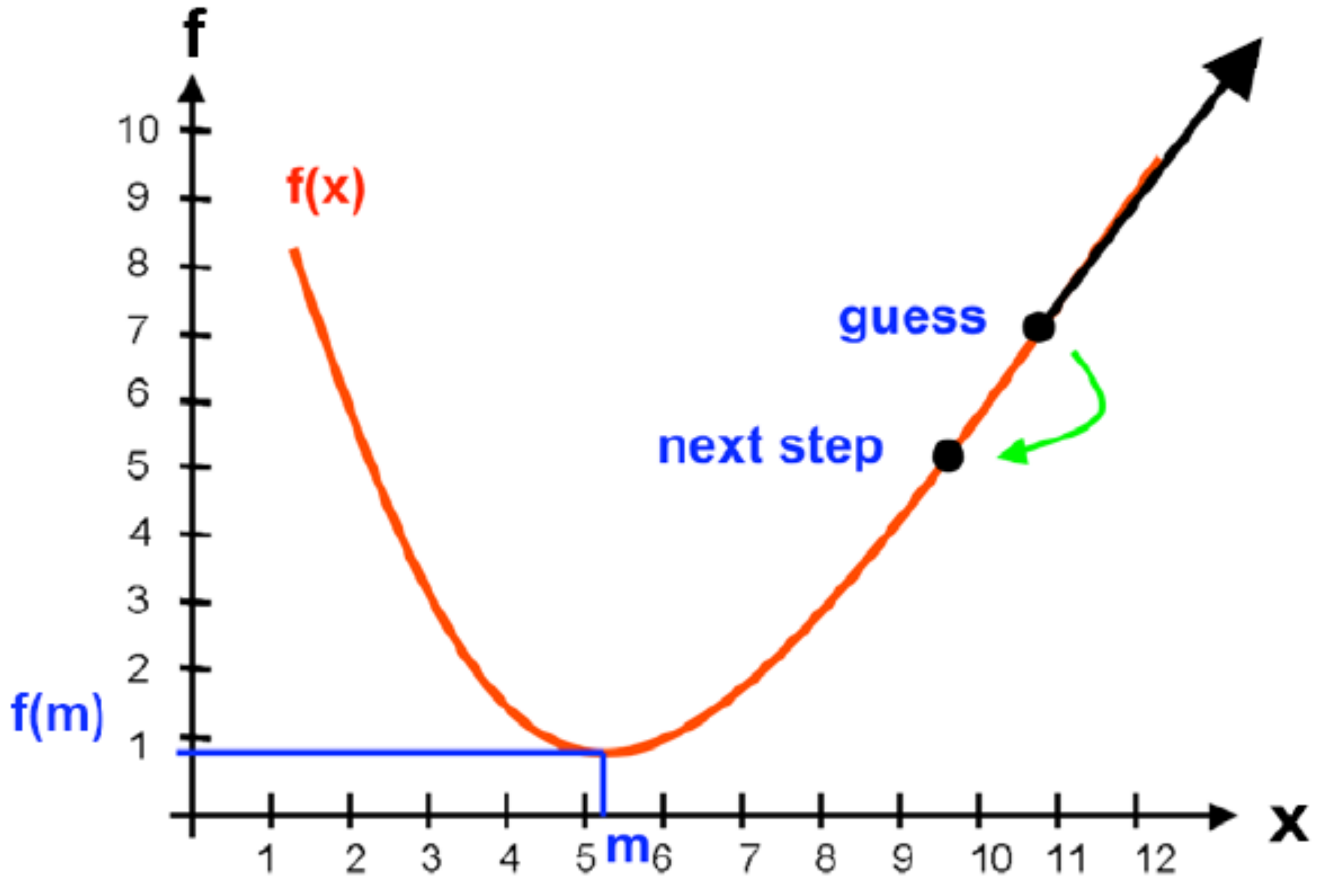
Minimum of a function is found by following the slope of the function

$S = \sum (y - y_{pred})^2$



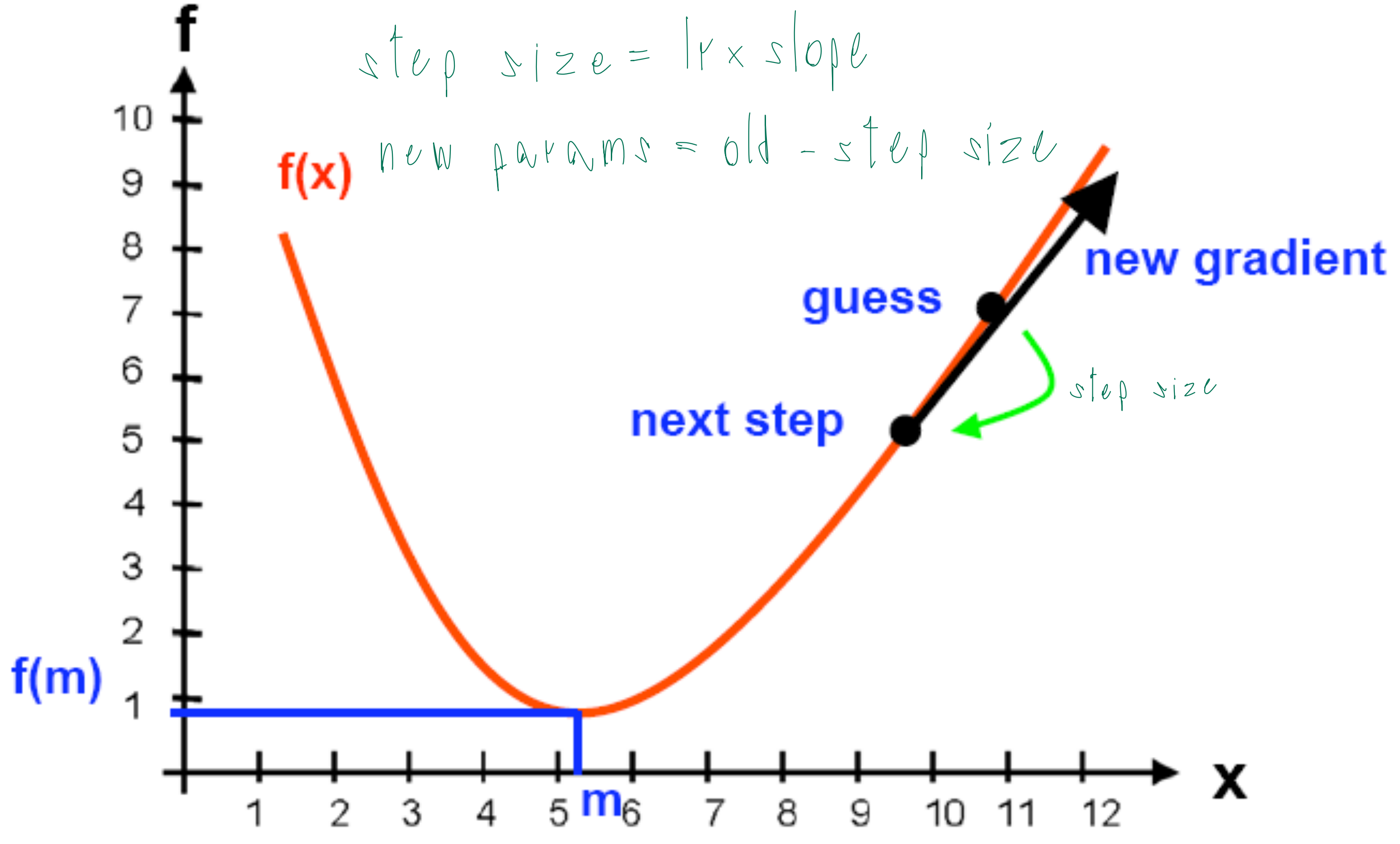
http://www.ce.berkeley.edu/~bayen/ce191www/lecturenotes/lecture10v01_descent2.pdf

Gradient Descent Basic 2
Moving opposite to gradient



http://www.ce.berkeley.edu/~bayen/ce191www/lecturenotes/lecture10v01_descent2.pdf

Gradient Descent Basic 3
Iterative gradient evaluation

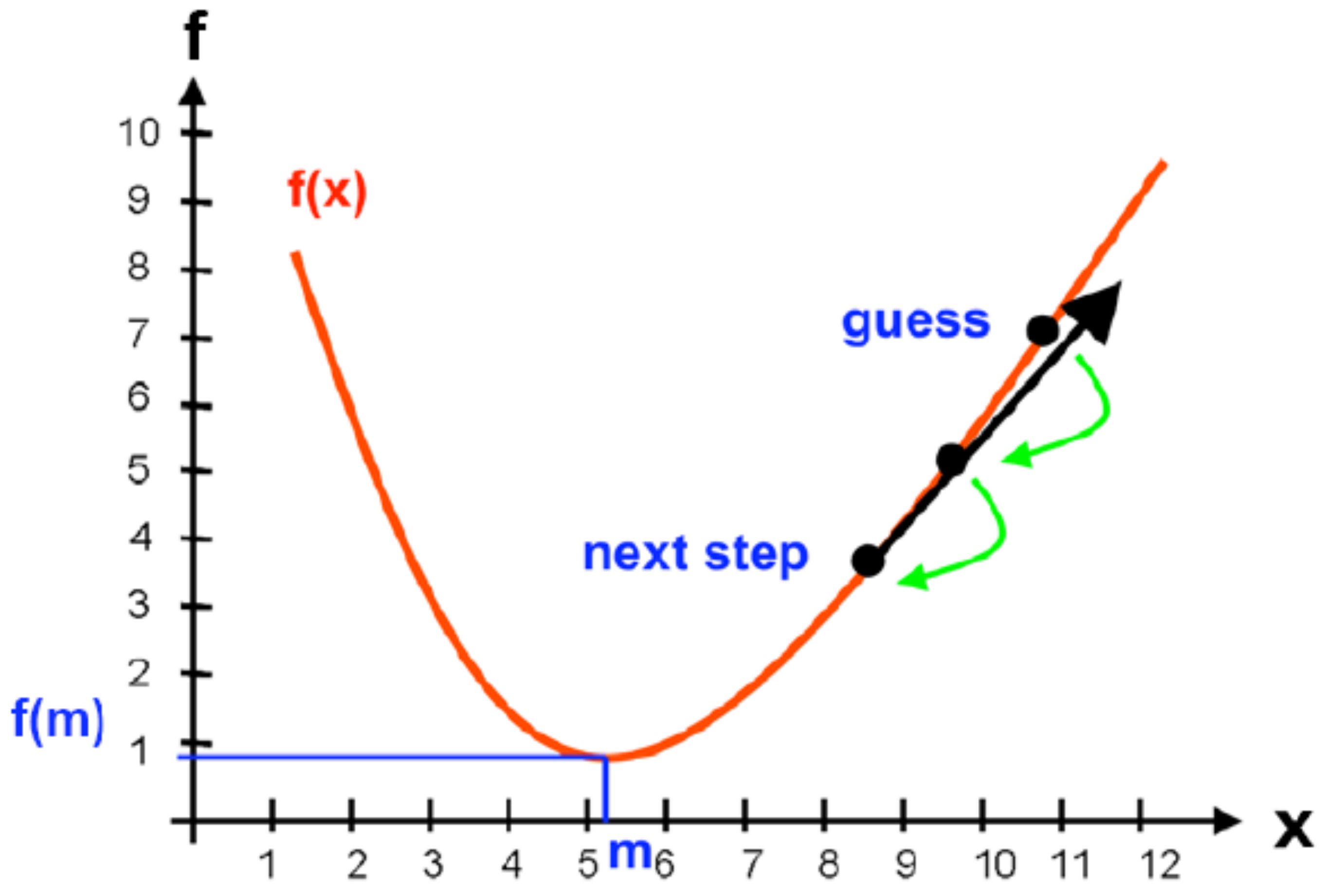


step size = lr × slope

new params = old − step size

http://www.ce.berkeley.edu/~bayen/ce191www/lecturenotes/lecture10v01_descent2.pdf

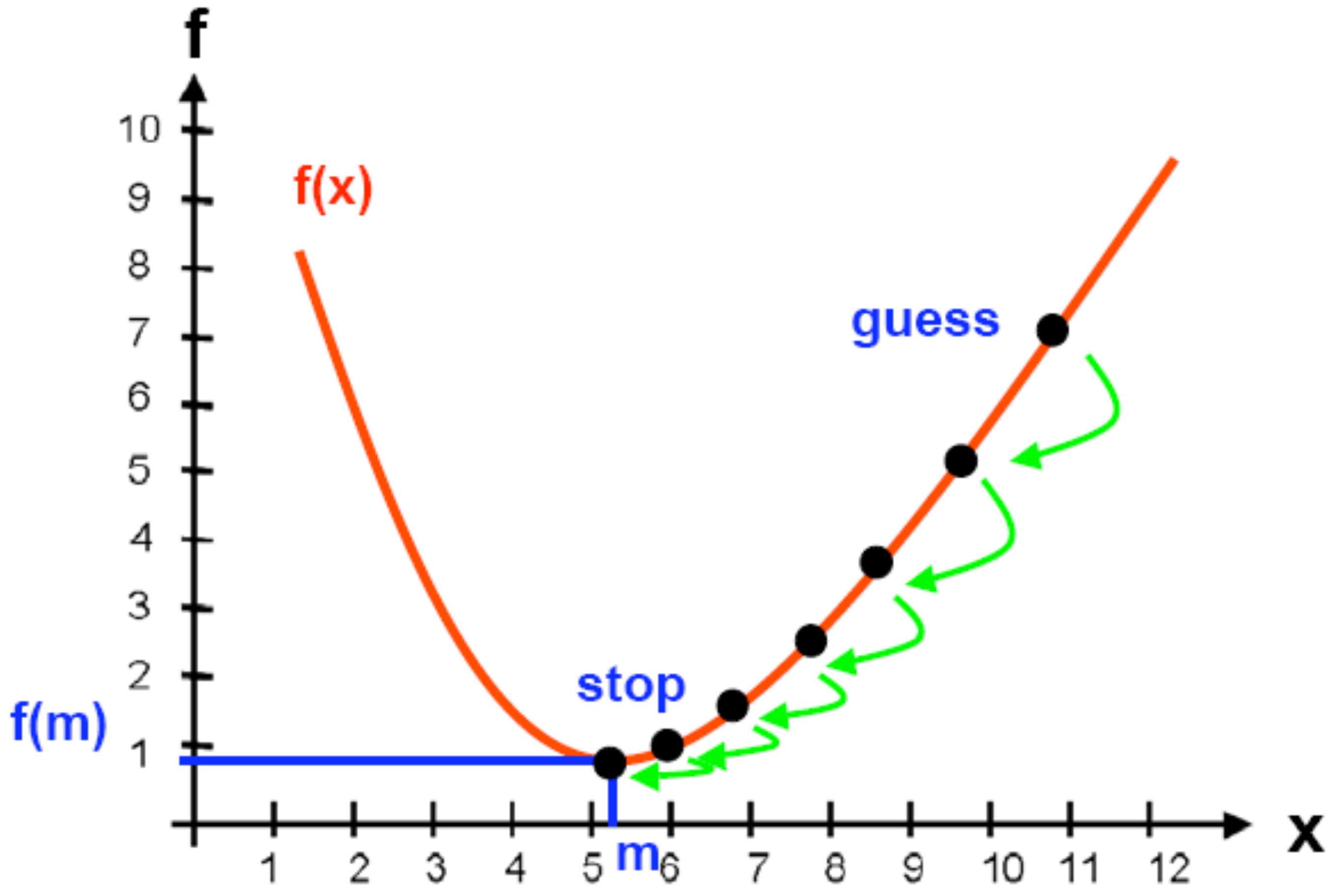
Gradient Descent Basic 4
Moving opposite to gradient



http://www.ce.berkeley.edu/~bayen/ce191www/lecturenotes/lecture10v01_descent2.pdf

Gradient Descent Basic 5

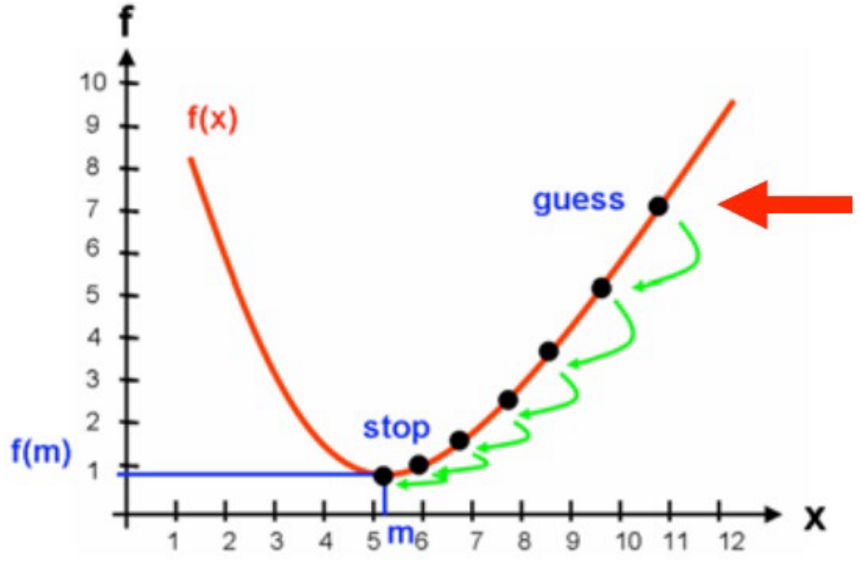
Iteratively descent opposite to the gradient



http://www.ce.berkeley.edu/~bayen/ce191www/lecturenotes/lecture10v01_descent2.pdf

Gradient Descent – algorithm

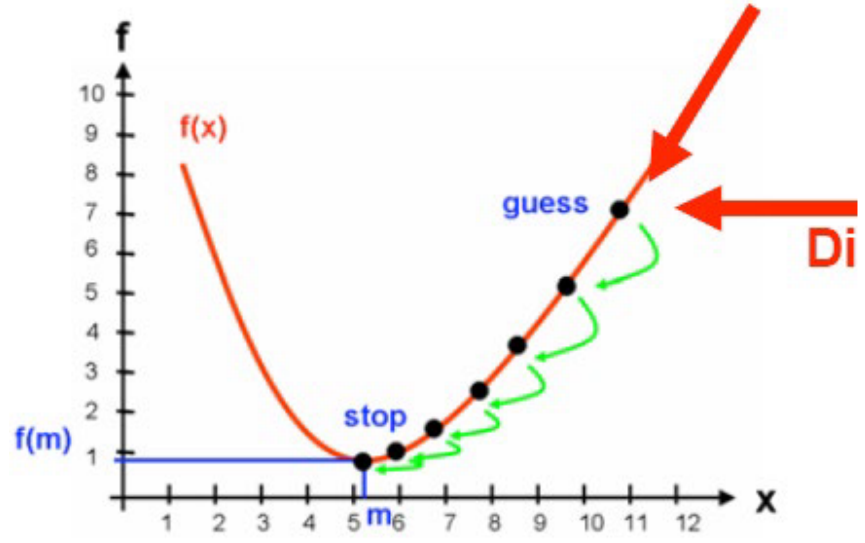
- Start with a point (randomly guessing)
- Repeat
  - Determine a descent direction
  - Choose a step
  - Update
- Until stopping criterion is satisfied



http://www.ce.berkeley.edu/~bayen/ce191www/lecturenotes/lecture10v01_descent2.pdf

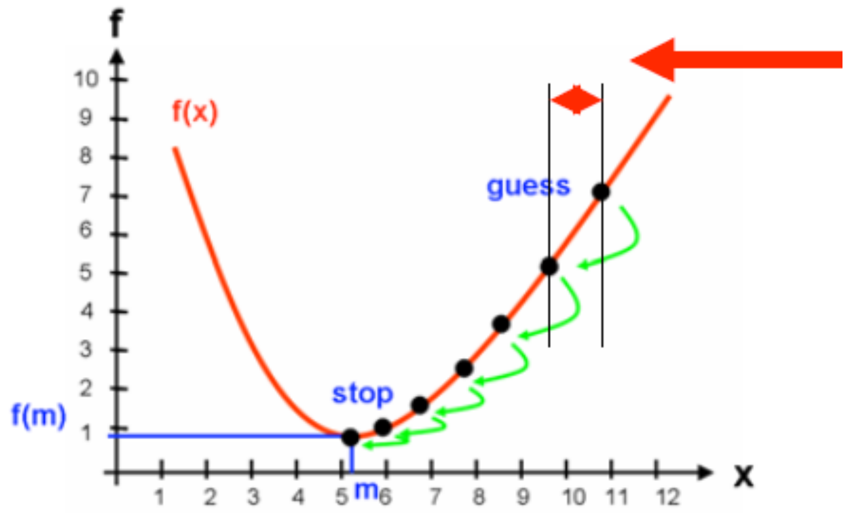
Gradient Descent – algorithm

- Start with a point (randomly guessing)
- Repeat
  - Determine a descent direction
  - Choose a step
  - Update
- Until stopping criterion is satisfied



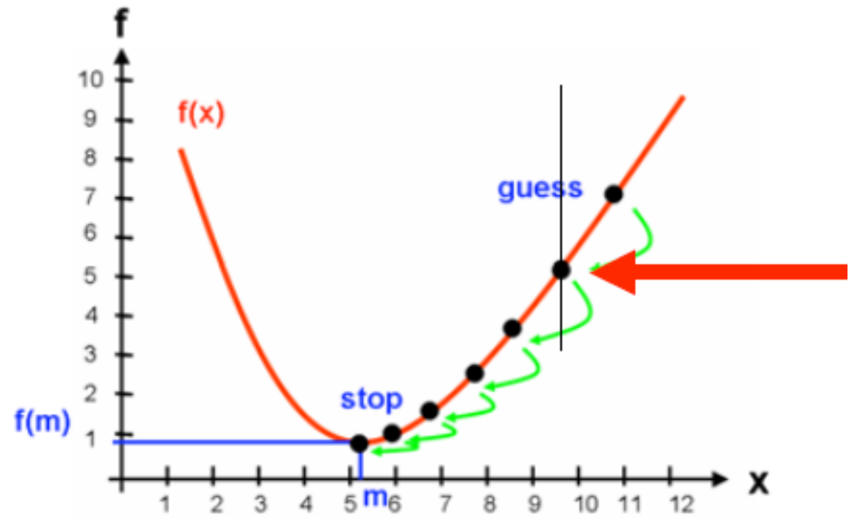
Gradient Descent – algorithm

- Start with a point (randomly guessing)
- Repeat
  - Determine a descent direction
  - Choose a step
  - Update
- Until stopping criterion is satisfied



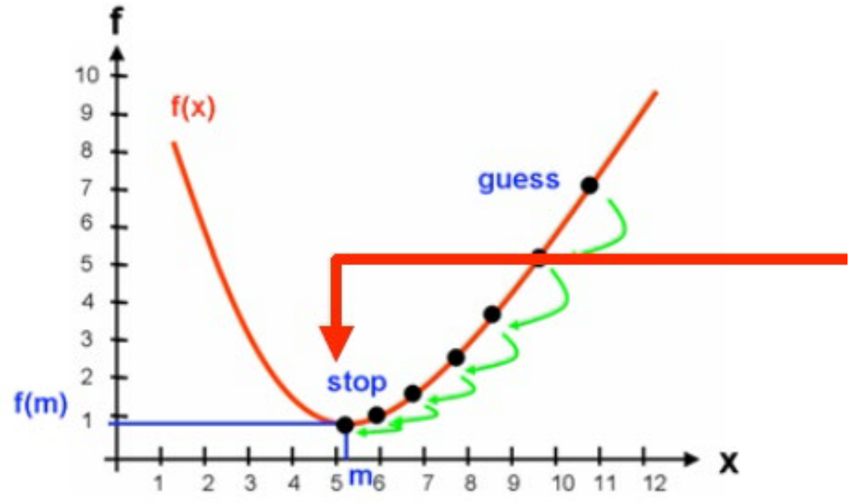
Gradient Descent – algorithm

- Start with a point (randomly guessing)
- Repeat
  - Determine a descent direction
  - Choose a step
  - Update
- Until stopping criterion is satisfied



Gradient Descent – algorithm

- Start with a point (randomly guessing)
- Repeat
    - Determine a descent direction
    - Choose a step
    - Update
- Until stopping criterion is satisfied



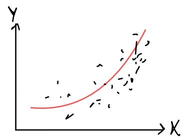
Gradient Descent – algorithm

- Start with a point (randomly guessing) $\rightarrow$ Randomly guessing β
- Repeat
  - Determine a descent direction $\rightarrow$ $\text{direction} = -\frac{dS(\beta)}{d\beta}$
  - Choose a step $\rightarrow$ $\text{step} > 0$
  - Update $\rightarrow$ $\beta^{t+1} = \beta^{t} - step\,\overset{\text{gradient}}{\frac{dS(\beta)}{d\beta}}$
- Until stopping criterion is satisfied $\rightarrow$ $\frac{dS(\beta)}{d\beta} \approx 0$

http://www.ce.berkeley.edu/~bayen/ce191www/lecturenotes/lecture10v01_descent2.pdf

Gradient Descent

$$f(x) = c_0 + c_1 e^{c_2 x}$$



$$S = \sum (y - f(x))^2$$

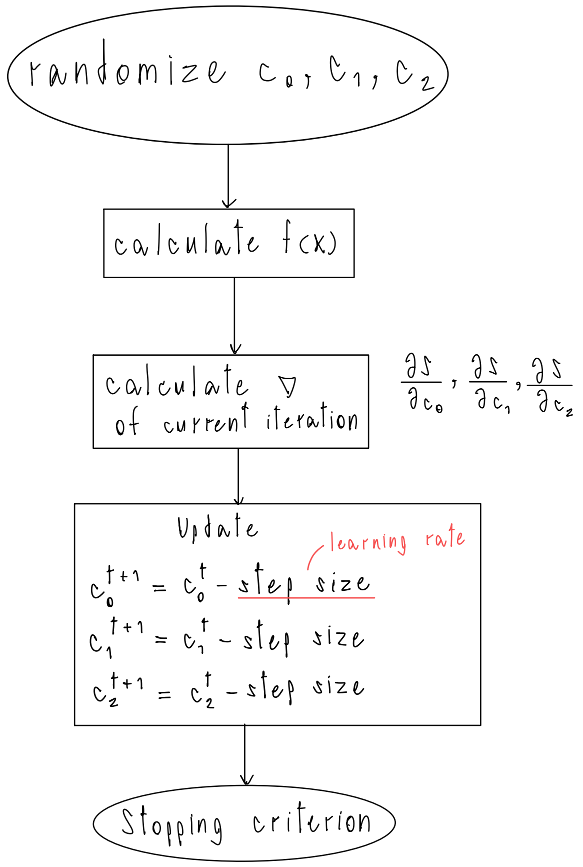
$$c_0 : \frac{dS}{dc_0} = 2 \sum (y - c_0 - c_1 e^{c_2 x})(-1)$$

$$= -2 \sum (y - c_0 - c_1 e^{c_2 x})$$

$$c_1 : \frac{dS}{dc_1} = -2 \sum (y - c_0 - c_1 e^{c_2 x})(e^{c_2 x})$$

$$c_2 : \frac{dS}{dc_2} = 2 \sum (y - c_0 - c_1 e^{c_2 x}) \cdot (-c_1 \cdot x e^{c_2 x})$$

Procedure



randomize c_0, c_1, c_2
calculate $f(x)$
calculate ∇ of current iteration
$\frac{\partial S}{\partial c_0}, \frac{\partial S}{\partial c_1}, \frac{\partial S}{\partial c_2}$
Update
learning rate
$c_0^{t+1} = c_0^t - \text{step size}$
$c_1^{t+1} = c_1^t - \text{step size}$
$c_2^{t+1} = c_2^t - \text{step size}$
Stopping criterion

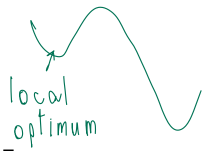
Batch Gradient Descent

- To implement Gradient Descent, you need to compute the gradient of the cost function with regards to each model parameter θ_j.
- Batch gradient descent uses data of the whole batch to compute gradient

$$\frac{\partial}{\partial \theta_j} \text{MSE}(\boldsymbol{\theta}) = \frac{2}{m} \sum_{i=1}^{m} (\underline{\boldsymbol{\theta}^T \mathbf{x}^{(i)}} - y^{(i)})\, x_j^{(i)}$$

$$\boldsymbol{\theta}^{(\text{next step})} = \boldsymbol{\theta} - \overset{\text{learning rate}}{\boxed{\eta}} \nabla_{\boldsymbol{\theta}}\ \text{MSE}(\boldsymbol{\theta})$$

ข้อดี :

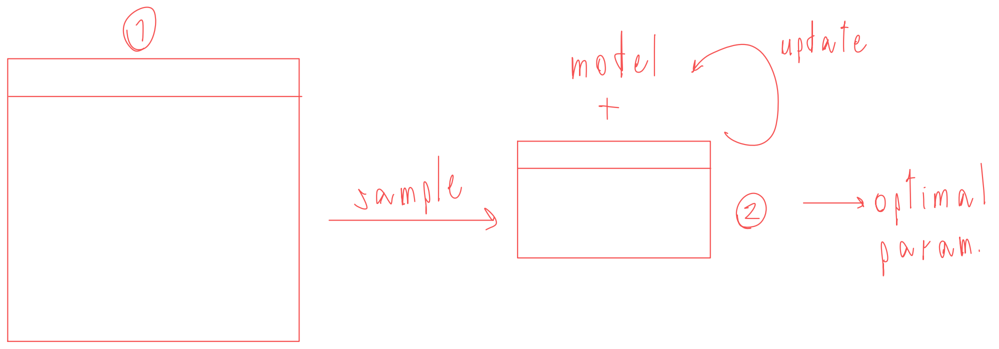


(ไม่สน value) (สน value)
exploration + exploitation

. สามารถหลุดจาก local optimum ได้

Stochastic gradient descent (SGD)

- Batch Gradient Descent uses the whole training set to compute the gradients at every step, which makes it very slow when the training set is large.
- Stochastic gradient descent samples random instances for training at each training step



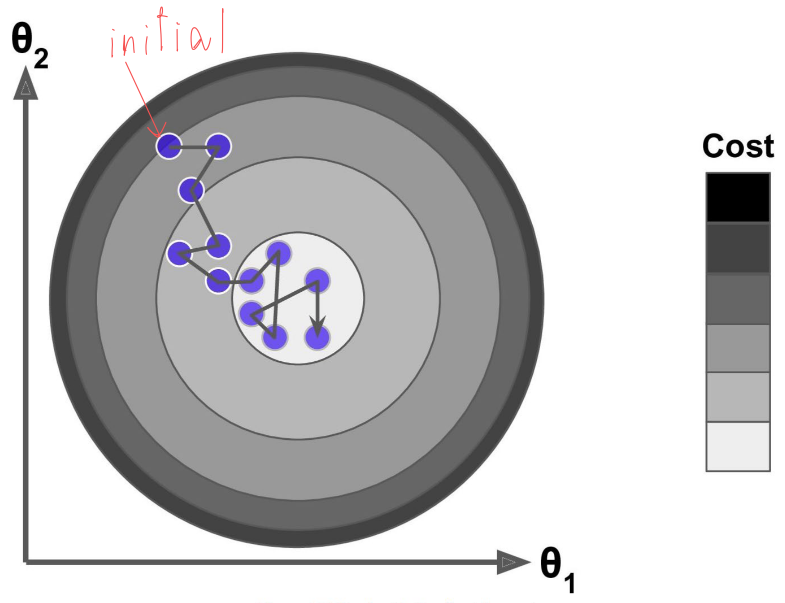
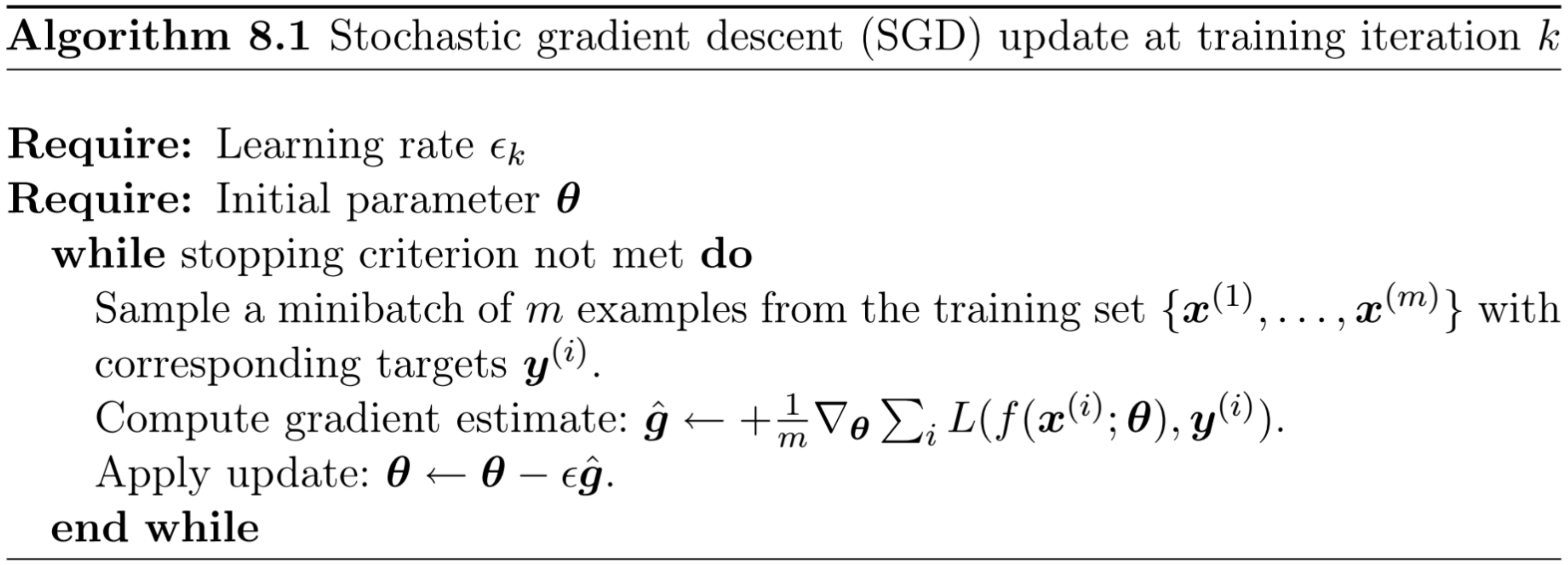


Figure 4-9. Stochastic Gradient Descent

Mini-Batch Gradient Descent

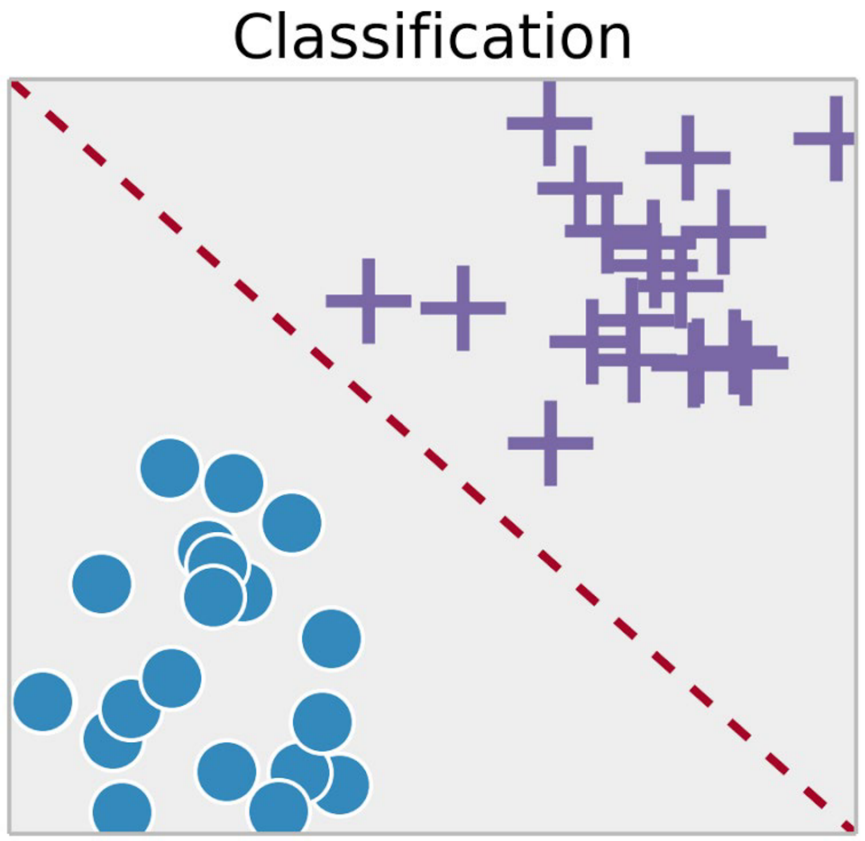
- Both stochastic and use small batch
- Apply for small batch instead of an instance.
- Faster by GPU computing



Algorithm 8.1 Stochastic gradient descent (SGD) update at training iteration k

```
Require: Learning rate $\epsilon_k$
Require: Initial parameter $\boldsymbol{\theta}$
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ with
    corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\boldsymbol{g}}$.
  end while
```

Example: Logistic Regression



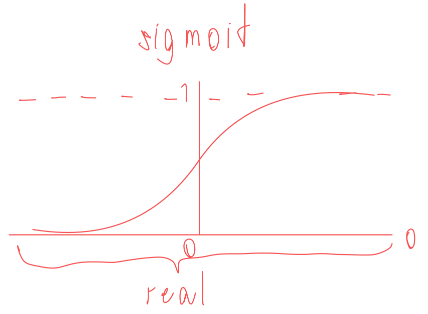
Logistic regression

$$\frac{H_0(x_i)}{1 - H_0(x_i)} = \text{odd ratio}$$

Logistic regression has the following mathematical formulae,

$$\log\left(\frac{\overset{\text{prob}}{H_{\boldsymbol{\theta}}(x_i)}}{1 - H_{\boldsymbol{\theta}}(x_i)}\right) = f(x_i) = \underbrace{\theta_0 + \sum_{i=1}^{n} \theta_i x_i}_{\text{regression}}$$

and



$$H_{\boldsymbol{\theta}}(x_i) = \frac{1}{1 + e^{-f(x)}}.$$

Loss function

This function has a nice property that,

$$\frac{\partial}{\partial \boldsymbol{\theta}} H_{\boldsymbol{\theta}}(x_i) = H_{\boldsymbol{\theta}}(x_i)(1 - H_{\boldsymbol{\theta}}(x_i))x_i.$$

For logistic regression, we can formulate the loss function as follows,

binary cross entropy

$$J(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n}[-y_i \log H_{\boldsymbol{\theta}}(x_i) - (1-y_i)\log(1 - H_{\boldsymbol{\theta}}(x_i))].$$

If $y_i = 0$: $H_\theta \longrightarrow 0$ $\quad J(\theta) \longrightarrow 0$ ✓

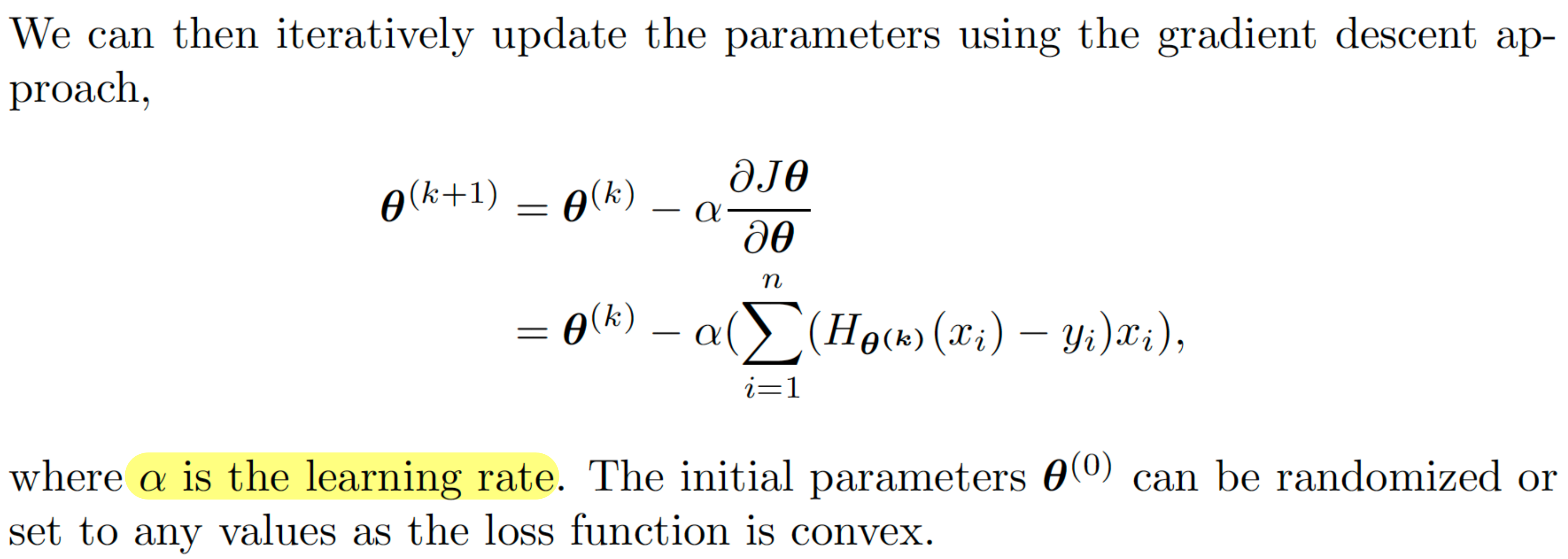
$H_1 \rightarrow 1$ $\quad J(\theta) \rightarrow 1$ ✗

Gradient

$$\frac{\partial}{\partial \theta} H_\theta(x_i) = H_\theta(x_i)(1 - H_\theta(x_i))\,x_i$$

$$\begin{aligned}
\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} &= \frac{\partial}{\partial \boldsymbol{\theta}}\Big(\frac{1}{n}\sum_{i=1}^{n}\big(-y_i \log H_{\boldsymbol{\theta}}(x_i) - (1-y_i)\log\left(1 - H_{\boldsymbol{\theta}}(x_i)\right)\big)\Big)\\
&= \frac{1}{n}\sum_{i=1}^{n}\Big(-y_i \frac{\partial}{\partial \boldsymbol{\theta}} \log H_{\boldsymbol{\theta}}(x_i) - (1-y_i)\frac{\partial}{\partial \boldsymbol{\theta}}\log\left(1 - H_{\boldsymbol{\theta}}(x_i)\right)\Big)\\
&= \frac{1}{n}\sum_{i=1}^{n}\Big(-\frac{y_i}{H_{\boldsymbol{\theta}}(x_i)}\frac{\partial}{\partial \boldsymbol{\theta}} H_{\boldsymbol{\theta}}(x_i) - \frac{1-y_i}{1-H_{\boldsymbol{\theta}}(x_i)}\frac{\partial}{\partial \boldsymbol{\theta}}(1 - H_{\boldsymbol{\theta}}(x_i))\Big)\\
&= \frac{1}{n}\sum_{i=1}^{n}\Big(-\frac{y_i}{H_{\boldsymbol{\theta}}(x_i)} H_{\boldsymbol{\theta}}(x_i)(1 - H_{\boldsymbol{\theta}}(x_i))x_i\\
&\qquad - \frac{1-y_i}{1-H_{\boldsymbol{\theta}}(x_i)} H_{\boldsymbol{\theta}}(x_i)(1 - H_{\boldsymbol{\theta}}(x_i))(-x_i)\Big)\\
&= \frac{1}{n}\sum_{i=1}^{n}\big(-y_i(1 - H_{\boldsymbol{\theta}}(x_i))x_i - (1-y_i)H_{\boldsymbol{\theta}}(x_i)(-x_i)\big)\\
&= \frac{1}{n}\sum_{i=1}^{n}\big(-y_i(1 - H_{\boldsymbol{\theta}}(x_i)) + (1-y_i)H_{\boldsymbol{\theta}}(x_i)\big)x_i\\
&= \frac{1}{n}\sum_{i=1}^{n}\big(-y_i + y_i H_{\boldsymbol{\theta}}(x_i) + H_{\boldsymbol{\theta}}(x_i) - y_i H_{\boldsymbol{\theta}}(x_i)\big)x_i\\
&= \frac{1}{n}\sum_{i=1}^{n}\big(\overset{\text{prob}}{H_{\boldsymbol{\theta}}(x_i)} - \overset{\text{label}}{y_i}\big)x_i
\end{aligned}$$

Gradient descent

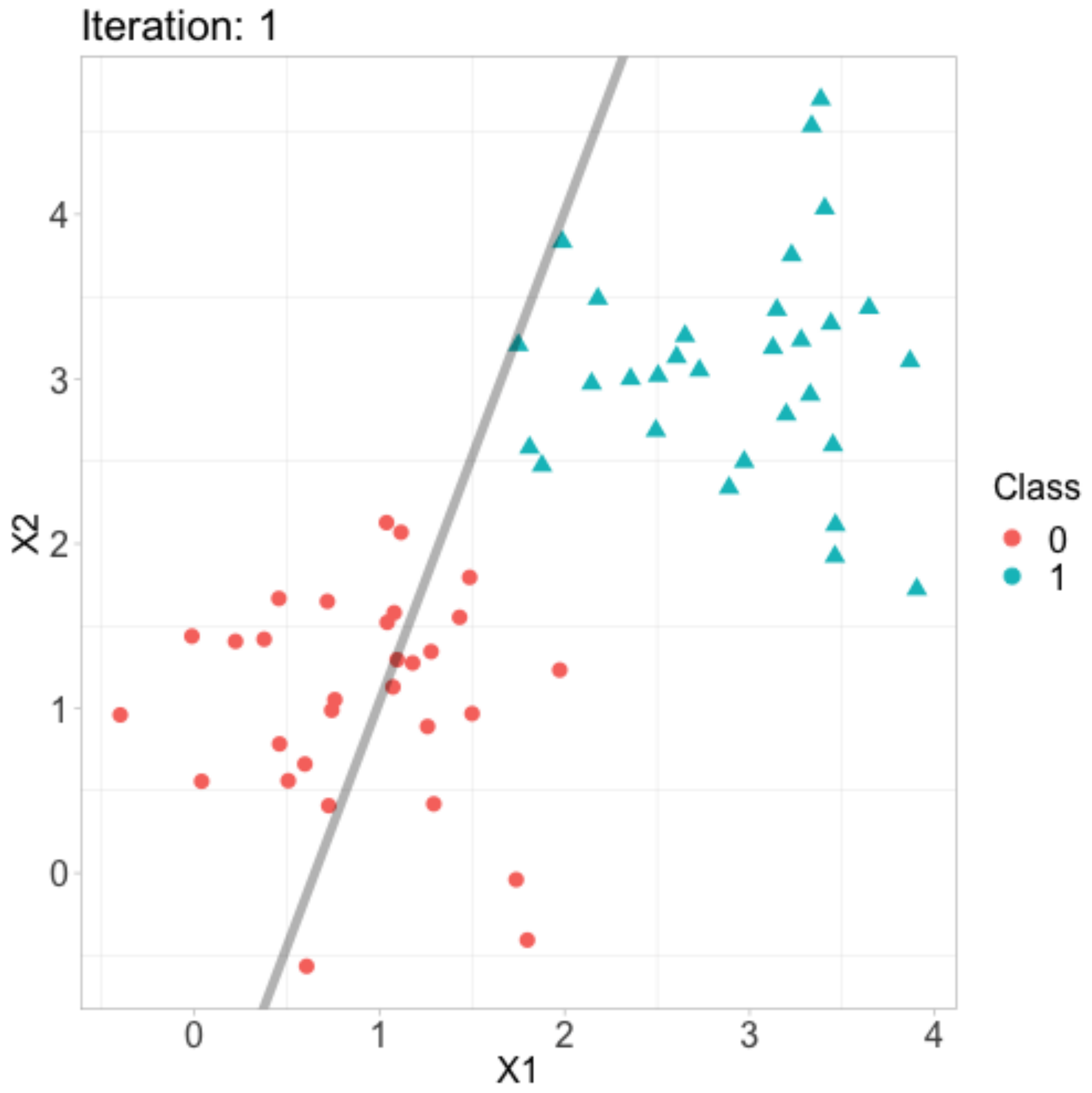


We can then iteratively update the parameters using the gradient descent approach,

$$\begin{aligned}\boldsymbol{\theta}^{(k+1)} &= \boldsymbol{\theta}^{(k)} - \alpha\frac{\partial J\boldsymbol{\theta}}{\partial\boldsymbol{\theta}} \\ &= \boldsymbol{\theta}^{(k)} - \alpha(\sum_{i=1}^{n}(H_{\boldsymbol{\theta}^{(k)}}(x_i) - y_i)x_i),\end{aligned}$$

where α is the learning rate. The initial parameters $\boldsymbol{\theta}^{(0)}$ can be randomized or set to any values as the loss function is convex.

Results



End of Lecture ~~3~~ 2

Question?