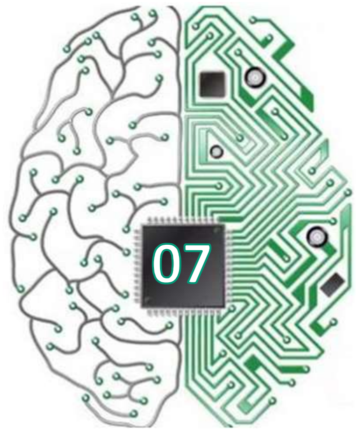
Open Elective Course [OE]
Course Code: CSO507
Winter 2023-24

Lecture#



Deep Learning

Unit-1: Machine Learning Basics [Part-III]

Course Instructor:

Dr. Monidipa Das

Assistant Professor
Department of Computer Science and Engineering
Indian Institute of Technology (Indian School of Mines) Dhanbad, Jharkhand 826004, India

Optimization

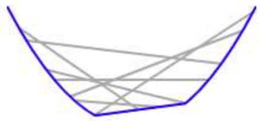
2

- ***Optimization*** is concerned with optimizing an objective function
  - finding the value of an argument that minimizes of maximizes the function
  - Most optimization algorithms are formulated in terms of minimizing a function $f(x)$
  - Maximization is accomplished vie minimizing the negative of an objective function (e.g., minimize $-f(x)$)
  - In minimization problems, the objective function is often referred to as a cost function or loss function or error function
- **Optimization is very important for machine learning**
  - The performance of optimization algorithms affect the model's training efficiency

Convex and Non-Convex Functions

3

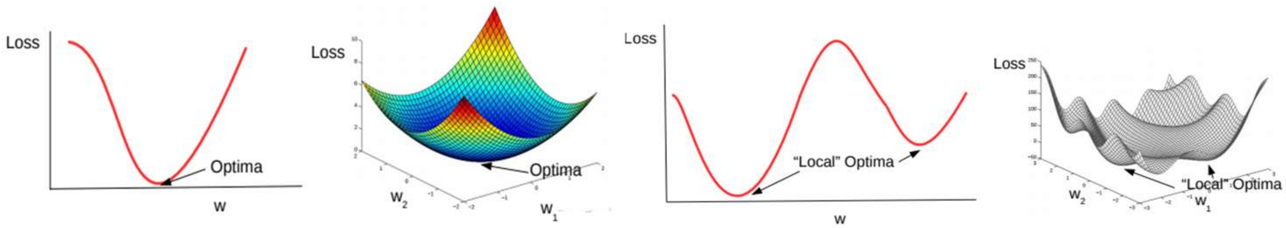
A concave function: no line segment joining two points on the graph lies above the graph at any point



A convex function: no line segment joining two points on the graph lies below the graph at any point

A function that is neither concave nor convex: the line segment shown lies above the graph at some points and below it at others

Negative of a convex function is called a concave function, which also has a unique optima (maxima)



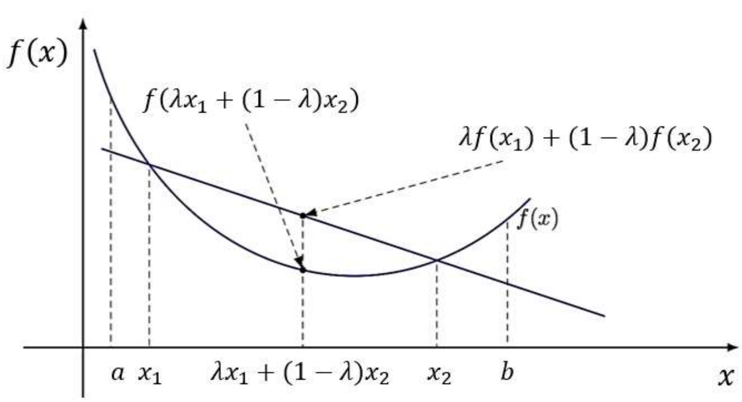
Convex functions are bowl-shaped. They have a unique optima (minima)

Non-convex functions have multiple minima. Usually harder to optimize as compared to convex functions

Convex Functions

4

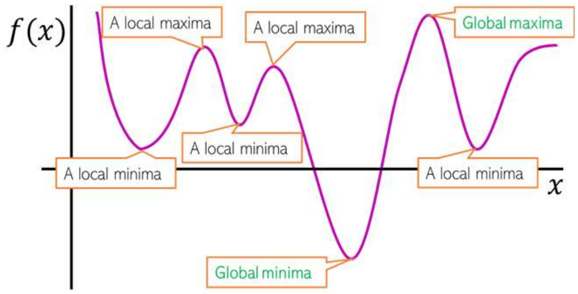
- In mathematical terms, the function f is a ***convex function*** if for all points x_1, x_2 and for all $\lambda \in [0,1]$: $\lambda f(x_1) + (1-\lambda) f(x_2) \geq f(\lambda x_1 + (1-\lambda) x_2)$

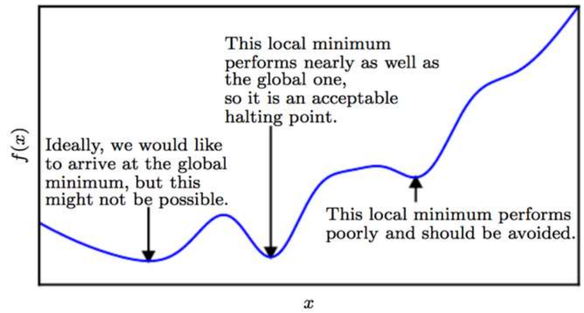


Functions and their optima

5

- Many ML problems require us to optimize a function f of some variable(s) x
- For simplicity, assume f is a scalar-valued function of a scalar $x (f: \mathbb{R} \rightarrow \mathbb{R})$



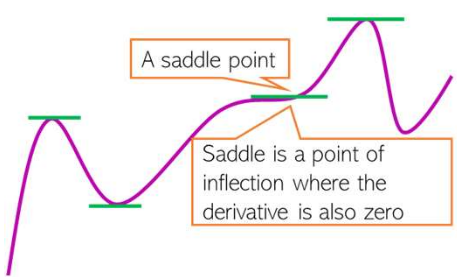


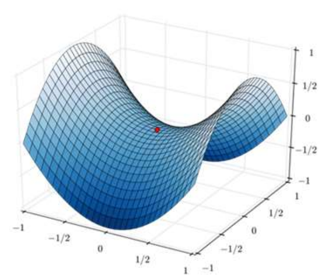
- Any function has one/more optima (maxima, minima), and maybe saddle points
- Finding the optima or saddle points requires derivatives/gradients of the function

Saddle Points

6

- Points where derivative is zero but are neither minima nor maxima

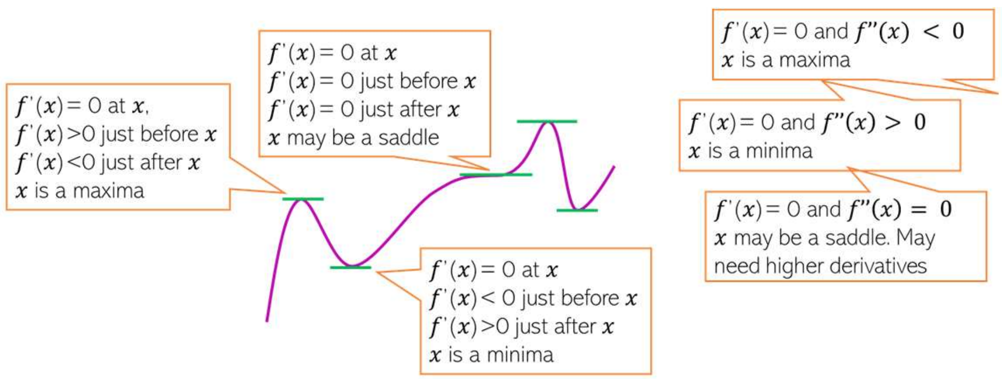




- Saddle points are very common for loss functions of deep learning models
  - The optimization algorithms may stall at saddle points, without reaching a minima
  - Need to be handled carefully during optimization
- Second or higher derivative may help identify if a stationary point is a saddle

Derivatives

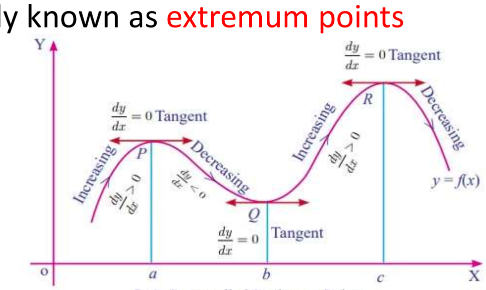
- How the derivative itself changes tells us about the function's optima



- The second derivative $f''(x)$ can provide this information

Stationary/Critical Points

- Are the points where the derivative of the function (say $f(x)$) is zero, i.e., $f'(x) = 0$
- The stationary points can be:
  - ***Minimum***, a point where the derivative changes from negative to positive
  - ***Maximum***, a point where the derivative changes from positive to negative
  - ***Saddle point***, derivative is either positive or negative on both sides of the point
- The minimum and maximum points are collectively known as extremum points
- The nature of stationary points can be determined based on the second derivative of $f(x)$ at the point
  - If $f''(x) > 0$, the point is a minimum
  - If $f''(x) < 0$, the point is a maximum
  - If $f''(x) = 0$, inconclusive, the point can be a saddle point, but it may not



P, Q, R are called Stationary Points

The same concept also applies to gradients of multivariate functions

9

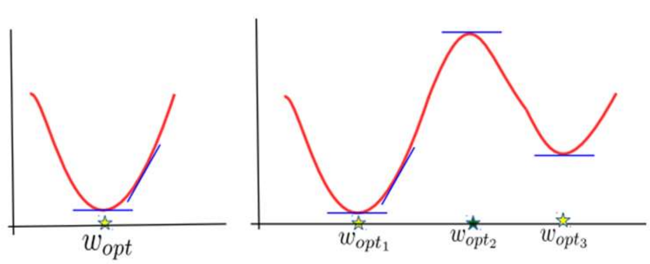
Methods for Solving Optimization Problems

Method 1: Non-iterative & Using First-Order Optimality

10

- First order optimality: The gradient $\boldsymbol{g}$ must be equal to zero at the optima

$$\boldsymbol{g} = \nabla_{\boldsymbol{w}}[L(\boldsymbol{w})] = \mathbf{0}$$



Called "first order" since only gradient is used and gradient provides the first order info about the function being optimized

The approach works only for very simple problems where the objective is convex and there are no constraints on the values $\boldsymbol{w}$ can take

- Sometimes, setting $\boldsymbol{g} = \mathbf{0}$ and solving for $\boldsymbol{w}$ gives a closed form solution
- If closed form solution is not available, the gradient vector $\boldsymbol{g}$ can still be used in iterative optimization algos, like gradient descent

Prof. Monidipa Das, Department of CSE, IIT (ISM) Dhanbad

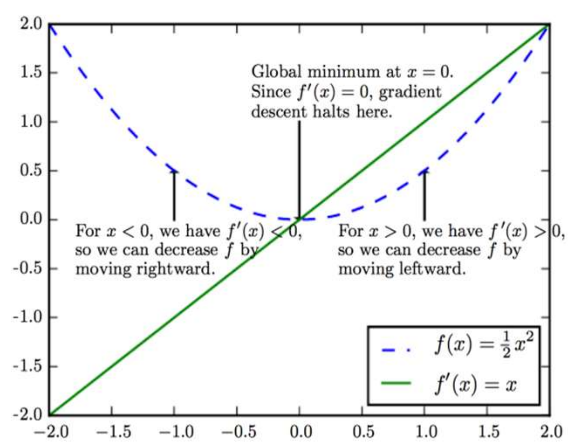
Method 2: Iterative Optimization via Gradient Descent

- Iterative since it requires several steps/iterations to find the optimal solution
- Suppose $y = f\ (x), x, y$ real nos.
  - Derivative of function denoted: $f'(x)$ or as $\frac{dy}{dx}$
    - Derivative $f'(x)$ gives the slope of $f(x)$ at point x
    - It specifies how to scale a small change in input to obtain a corresponding change in the output: $f\ (x\ +\ \varepsilon)\ \approx\ f\ (x)\ +\ \varepsilon\, f'\ (x)$
  - It tells how you make a small change in input to make a small improvement in y
  - We know that $f\ (x\ -\ \varepsilon\ sign\ (f'(x)))$ is less than $f\ (x)$ for small ε.
  - Thus we can reduce $f(x)$ by moving x in small steps with opposite sign of derivative
    - This technique is called **gradient descent (Cauchy 1847)**

How Gradient Descent uses derivatives

- Criterion $f(x)$ minimized by moving from current solution in direction of the negative of gradient

- For $x>0$, $f(x)$ increases with x and $f'(x)>0$
- For $x<0$, $f(x)$ is decreases with x and $f'(x)<0$
- Use $f'(x)$ to follow function downhill
- Reduce $f(x)$ by going in direction opposite sign of derivative $f'(x)$



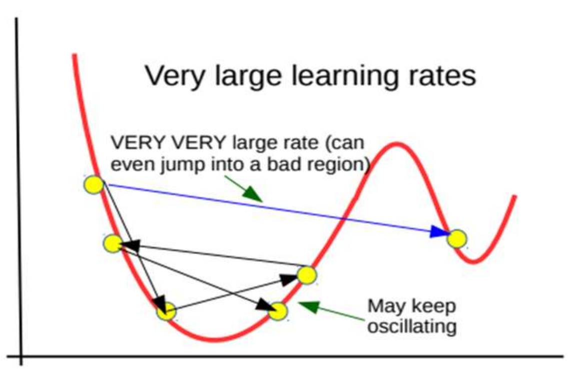
Method of Gradient Descent

13

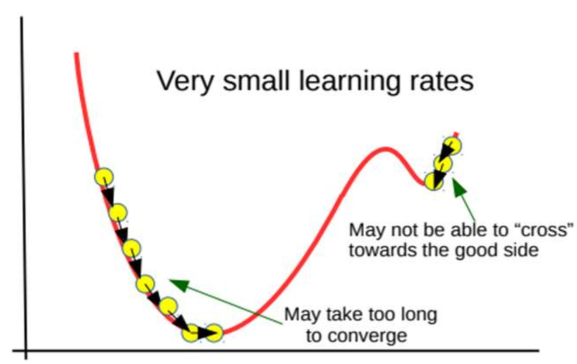
- The gradient points directly uphill, and the negative gradient points directly downhill
- Thus we can decrease f by moving in the direction of the negative gradient
  - This is known as the method of ***steepest descent*** or ***gradient descent***
- Steepest descent proposes a new point $\boldsymbol{x}' = \boldsymbol{x} - \alpha \nabla_{\boldsymbol{x}} f(\boldsymbol{x})$
  - where α is the ***learning rate***, a positive scalar. Set to a small constant.
- Steepest descent converges when every element of the gradient is zero
  - In practice, very close to zero
- We may be able to avoid iterative algorithm and jump to the critical point by solving the equation $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = 0$ for x

Learning rate is very important

14



Too large, the next point will perpetually bounce haphazardly across the bottom of the well



Too small learning rate will take too long

Gradient descent algorithm

15

Gradient Descent: Algorithmic representation

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad \text{(simultaneously update } j = 0 \text{ and } j = 1)$$

}

Parameters

Cost function, which is to be minimized

Correct updating of parameters

$$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$
$$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$
$$\theta_0 := \text{temp0}$$
$$\theta_1 := \text{temp1}$$

Incorrect updating of parameters

$$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$
$$\theta_0 := \text{temp0}$$
$$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$
$$\theta_1 := \text{temp1}$$

Gradient Descent for Linear Regression

16

$$h_\theta(x) = f_\theta(x) = \theta_0 + \theta_1 x \qquad J(\theta_0, \theta_1) = \frac{1}{2N} \sum_{i=1}^{N} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

m is the number of samples/ data points

repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \frac{1}{N} \sum_{i=1}^{N} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$$
$$\theta_1 := \theta_1 - \alpha \frac{1}{N} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)}$$

}

Update θ_0 and θ_1 simulteneously

Gradient Descent in Execution

17

Example Regression Problem:
Prediction of house price in a city based on living area.

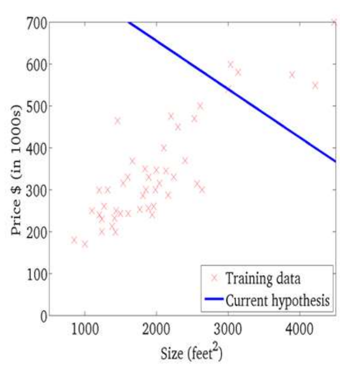
Living area (feet^2)	Price (1000\$s)
2104	400
1600	330
2400	369
1416	232
3000	540
$\vdots$	$\vdots$

Gradient Descent in Execution

18

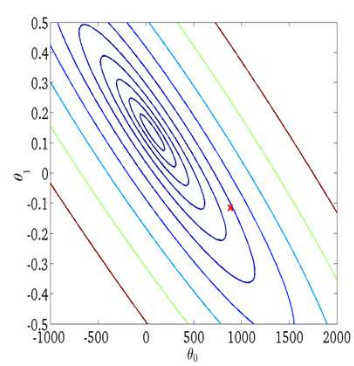
$h_\theta(x)$

(for fixed θ_0, θ_1 , this is a function of x)



$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



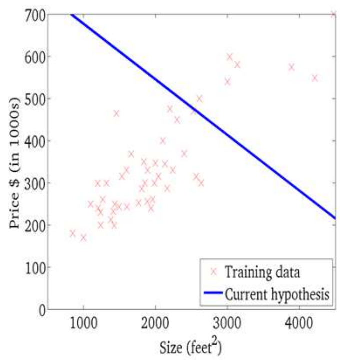
Courtesy: Andrew Ng

Gradient Descent in Execution

19

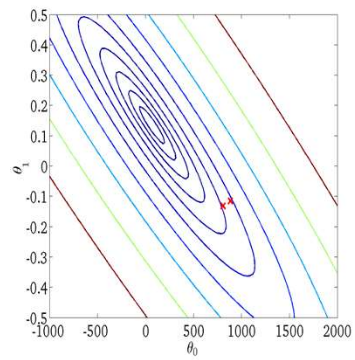
$h_\theta(x)$

(for fixed θ_0, θ_1 , this is a function of x)



$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)

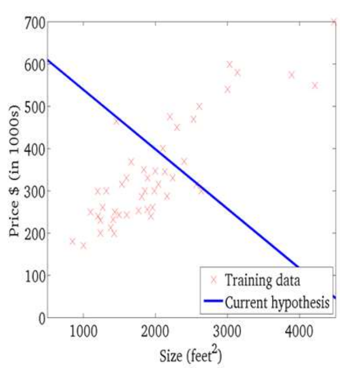


Gradient Descent in Execution

20

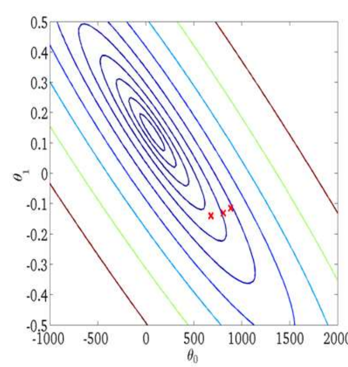
$h_\theta(x)$

(for fixed θ_0, θ_1 , this is a function of x)



$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



Gradient Descent in Execution

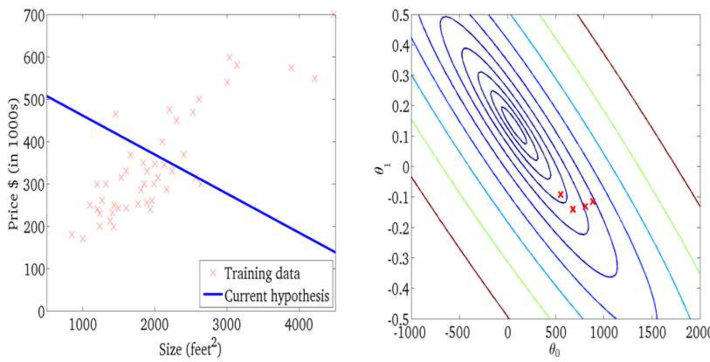
21

$h_\theta(x)$

(for fixed θ_0, θ_1 , this is a function of x)

$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



Gradient Descent in Execution

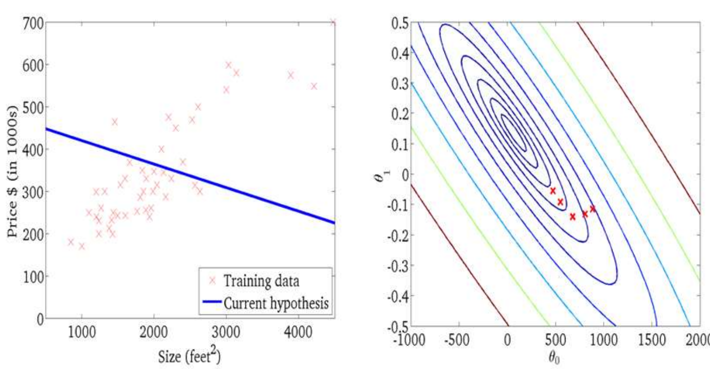
22

$h_\theta(x)$

(for fixed θ_0, θ_1 , this is a function of x)

$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)

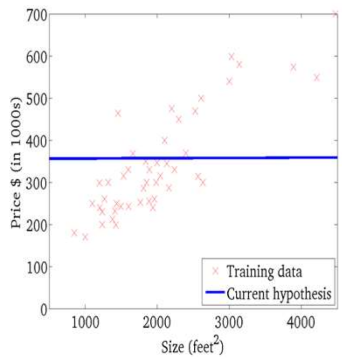


Gradient Descent in Execution

23

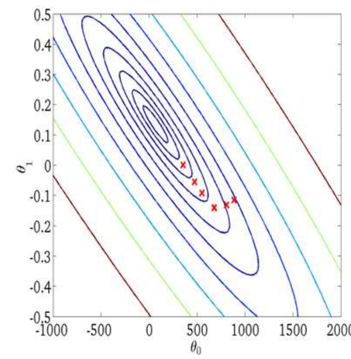
$h_\theta(x)$

(for fixed θ_0, θ_1 , this is a function of x)



$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)

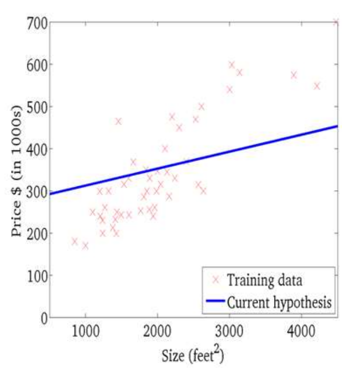


Gradient Descent in Execution

24

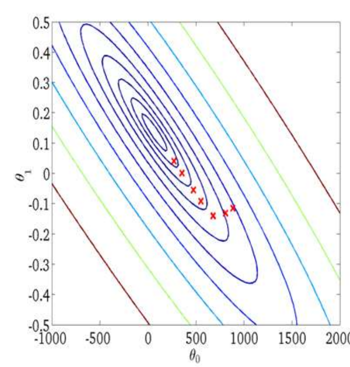
$h_\theta(x)$

(for fixed θ_0, θ_1 , this is a function of x)



$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)

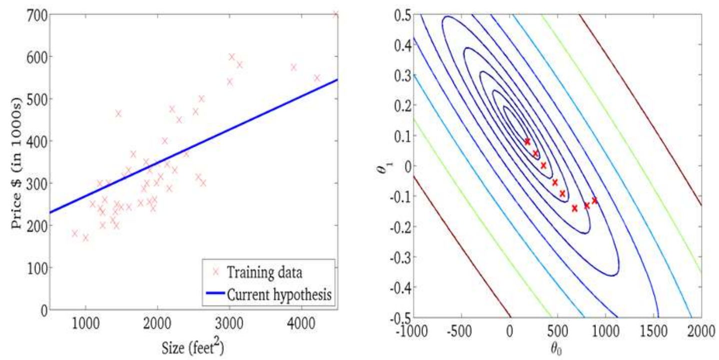


Gradient Descent in Execution

25

$h_\theta(x)$ (for fixed θ_0, θ_1, this is a function of x)

$J(\theta_0, \theta_1)$ (function of the parameters θ_0, θ_1)

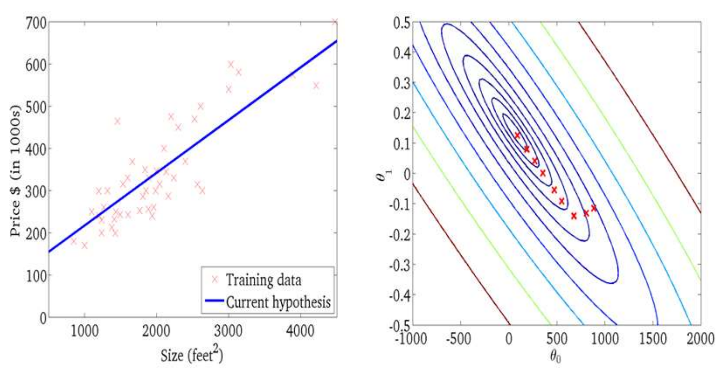


Gradient Descent in Execution

26

$h_\theta(x)$ (for fixed θ_0, θ_1, this is a function of x)

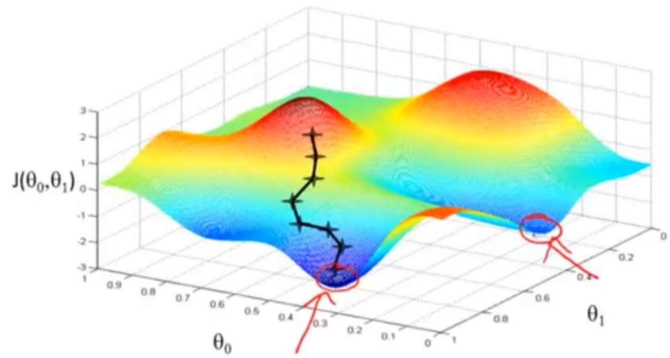
$J(\theta_0, \theta_1)$ (function of the parameters θ_0, θ_1)



GD: Advantages and Disadvantages

27

- Advantages
  - Ease of implementation
  - Convergence guarantee
  - Scalability
  - Versatility
- Disadvantages
  - Sensitivity to learning rate
  - Sensitivity to initialization
  - Convergence speed
  - Correctness

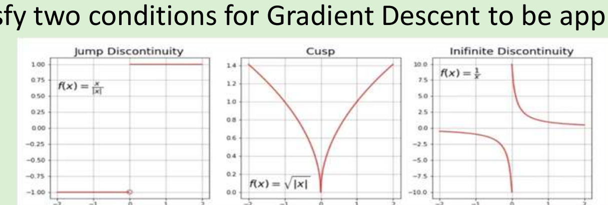


Courtesy: Andrew Ng

Gradient Descent: Facts (Summary)

28

- Gradient gives the direction of steepest change in function's value
- Iterative since it requires several steps/iterations to find the optimal solution
- For convex functions, GD will converge to the global minima
- The gradient descent algorithm is not written for all types of functions.
- The function has to satisfy two conditions for Gradient Descent to be applicable on it:
  - differentiable
  - convex function



- Good initialization needed for non-convex functions
- The learning rate is very imp. Should be set carefully (fixed or chosen adaptively).
- For max. problems we can use gradient ascent. Will move <u>in</u> the direction of the gradient

$$w^{(t+1)} = w^{(t)} + \alpha_t g^{(t)}$$ ***Gradient***

Types of Gradient Descent

29

- There are **three major types** of Gradient Descent Algorithms:
  - Batch Gradient Descent (BGD)
  - Stochastic Gradient Descent (SGD)
  - Mini-Batch Gradient Descent (MBGD)

Gradient Descent (GD): Pseudo-Code

Input: parameters (θ), gradient of the loss function with respect to the parameters ($d\theta$), learning rate (α)
Update parameters: $\theta = \theta - \alpha \times d\theta$
Output: updated parameters (θ)

Pseudo-Code: Batch Gradient Descent (BGD)

30

Step 1: Select a learning rate α

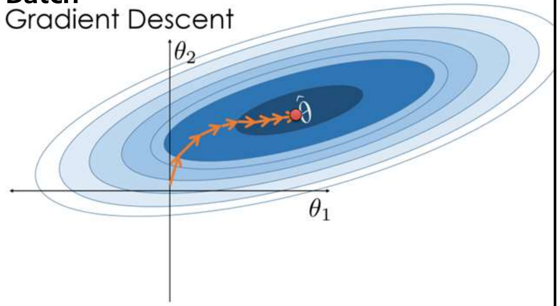
Step 2: Select initial parameter values θ as the starting point

Step 3: Update all parameters from the gradient of the training data set, i.e. compute

$$\theta = \theta - \alpha \times \nabla_\theta J(\theta)$$

Step 4: Repeat Step 3 until a local minima is reached

Batch
Gradient Descent



Pseudo-Code: Stochastic Gradient Descent (SGD)

31

Step 1: Randomly shuffle the data set of size N

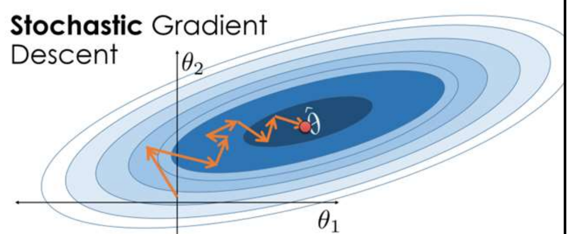
Step 2: Select a learning rate α

Step 3: Select initial parameter values θ as the starting point

Step 4: Update all parameters from the gradient of a single training example x^j, y^j, i.e. compute

$$\theta = \theta - \alpha \times \nabla_\theta J(\theta; x^j; y^j)$$

Step 5: Repeat Step 4 until a local minimum is reached



Common to decay learning rate linearly until iteration τ: $\alpha_i=(1-\epsilon)\alpha_0+\epsilon\alpha_\tau$ with $\epsilon=i/\tau$
After iteration τ, it is common to leave α constant Often a small positive value in the range 0.0 to 1.0

Pseudo-Code: Mini-Batch Gradient Descent (MBGD)

32

- **Estimate gradient on a batch of b samples ($b <$ N), N is the no. of training samples**

Step 1: Randomly shuffle the data set of size N

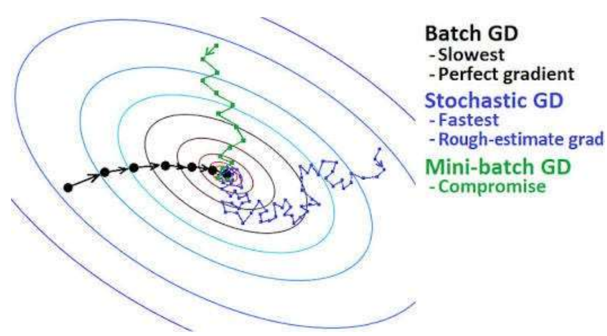
Step 2: Select a learning rate α

Step 3: Select initial parameter values θ as the starting point

Step 4: Update all parameters from the gradient calculated against a batch size of b training examples, i.e. compute

$$\theta = \theta - \alpha \times \nabla_\theta J(\theta; x^{j:j+b}; y^{j:j+b})$$

Step 5: Repeat Step 4 until a local minimum is reached



Momentum Method

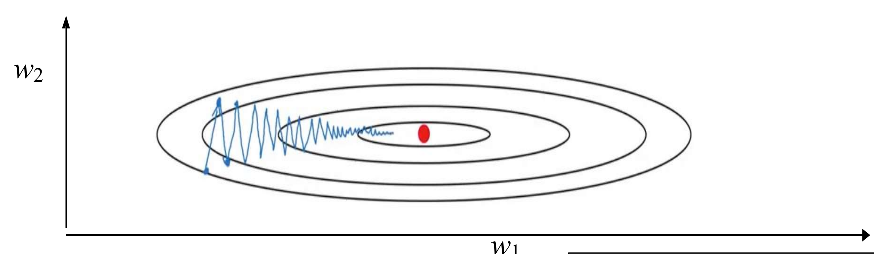
33

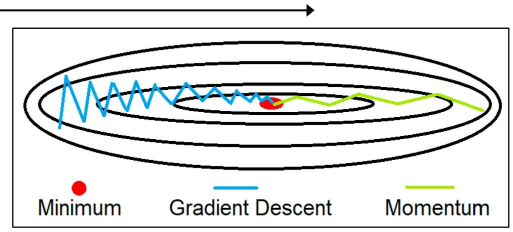
- SGD is a popular optimization strategy but it can be slow
- Momentum method accelerates learning, when:
  - Facing high curvature
  - Small but consistent gradients
  - Noisy gradients
- It works by accumulating the moving average of past gradients and moves in that direction while exponentially decaying

Gradient Descent with Momentum

34

- Gradient descent with momentum converges faster than standard gradient descent
- Taking large steps in w_2 direction and small steps in w_1 direction slows down algorithm





- Momentum reduces oscillation in w_2 direction
- Now can set a higher learning rate

Momentum Definition

35

- Introduce velocity variable $\boldsymbol{v}$
- This is the direction and speed at which parameters move through parameter space
- Name momentum comes from physics and is mass times velocity
  - The momentum algorithm assumes unit mass
- A hyperparameter $\delta \in [0,1)$ determines exponential decay of $\boldsymbol{v}$

Momentum Update Rule

36

- The update rule is given by

$$v \leftarrow \delta v - \alpha\nabla_{\theta}\left(\frac{1}{m}\sum_{i=1}^{m} L\left(f(x^{(i)};\theta), y^{(i)}\right)\right)$$
$$\theta \leftarrow \theta + v$$

- The velocity $\boldsymbol{v}$ accumulates the gradient elements

$$\nabla_{\theta}\left(\frac{1}{m}\sum_{i=1}^{m} L\left(f(x^{(i)};\theta), y^{(i)}\right)\right)$$

- The larger δ is relative to α, the more previous gradients affect the current direction

SGD Algorithm with Momentum

37

```
INPUT: cost function J(θ), learning rate α,  number of iterations: T,
 batch size B, initial velocity: v
INITIALIZE: random θ
FOR i = 1 to T DO
 Split the training examples into B mini-batches of size b:
 FOR j = 1 to number of mini-batches B DO
  Compute the gradient of J with respect to θ for a mini-batch
  of training examples:
  gradient = 1/b × ∇θ Σ(J(θ, x^j, y^j))
  Compute velocity update: v = δv − α × gradient
  Update the parameters θ:
  θ = θ + v
 END FOR
END FOR
OUTPUT: θ
```


Nesterov Momentum

38

- A variant to accelerate gradient, with update

$$\boldsymbol{v} \leftarrow \delta \boldsymbol{v} - \alpha \nabla_{\boldsymbol{\theta}} \left[\frac{1}{m} \sum_{i=1}^{m} L\left(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta} + \delta \boldsymbol{v}), \boldsymbol{y}^{(i)} \right) \right],$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v},$$

  - where parameters δ and α play a similar role as in the standard momentum method

– Difference between Nesterov and standard momentum is where gradient is evaluated.

  - **Nesterov gradient is evaluated after the current velocity is applied.**
  - One can interpret Nesterov as attempting to add a correction factor to the standard method of momentum

SGD with Nesterov Momentum

39

- A variant of the momentum algorithm
  - Nesterov's accelerated gradient method
- Applies a correction factor to standard method

```
INPUT: cost function J(θ), learning rate α,  number of iterations: T,
  batch size B, initial velocity: v
INITIALIZE: random θ
FOR i = 1 to T DO
  Split the training examples into B mini-batches of size b:
  FOR j = 1 to number of mini-batches B DO
    Apply interim update: θ̃ = θ + δv
    Compute the gradient of J with respect to θ for a mini-batch
    of training examples:
    gradient = 1/b × ∇θ̃ Σ(J(θ̃, x^j, y^j))
    Compute velocity update: v = δv − α × gradient
    Update the parameters θ:
    θ = θ + v
  END FOR
END FOR
OUTPUT: θ
```


40

Questions?