SJSU SAN JOSÉ STATE UNIVERSITY

DATA 220: Mathematical Methods for Data Analytics

Dr. Mohammad Masum

Optimization

- Optimization – finding the input parameters or arguments to a function that produce the minimum or maximum of the function
  - Continuous function optimization
  - Combinatorial optimization
- Continuous function optimization
  - Check whether the objective function can be differentiated at a point or not (finding gradient)

Optimization

- Objective function (aka, loss function or cost function)
  - Evaluate a candidate solution in terms of a given measure of quality or goodness
  - Primary goal is to find optimal value(s) of one or multiple variables (parameters) that minimize or maximize the value of the objective function
    - In other words, an objective function is a mathematical expression that takes one or more input variables and produces a scalar output value, which represents the degree of fitness or quality of the solution
  - It an be defined based on various criteria or goals: maximizing profit, minimizing cost, maximizing accuracy, or minimizing error
  - The choice of objective function depends on the specific problem at hand and the goals of the optimization process

Optimization

- Objective function (aka, loss function or cost function)
  - Optimizing the objective function can be challenging based on the selection of the function
    - If analytical solution using calculus or linear algebra is possible – easy
    - Local search algorithm – moderate
    - Global search algorithm – hard

Optimization- Local Optimization

- Local Optimization
  - Locate the optima for an objective function from a starting point believed to contain the optima
  - Finding a good point if not the very best
- A local optimization algorithm will locate the global optima
  - If the local optima is the global optima, or,
  - If the search space contain the global optima

Many numerical optimization methods seek local minima. Local minima are locally optimal, but we do not generally know whether a local minimum is a global minimum

Optimization- Global Optimization

- Global Optimization
  - The algorithm searches for the global optima by employing mechanisms to search larger parts of the search space
  - Global optimization is used for problems with a small number of variables, where computing time is not critical, and the value of finding the true global solution is very high

No Free Lunch Algorithm

- The No Free Lunch (NFL) algorithm is a concept in computer science that describes the limitations of optimization algorithms
- It states that there is no one-size-fits-all optimization algorithm that performs better than any other algorithm on all problems
  - Implies that every optimization algorithm is equally good when averaged over all possible problems.
- Therefore, the choice of an optimization algorithm should be based on the specific characteristics of the problem being solved.
  - the best algorithm for one problem may not be the best algorithm for another problem.

No Free Lunch Algorithm

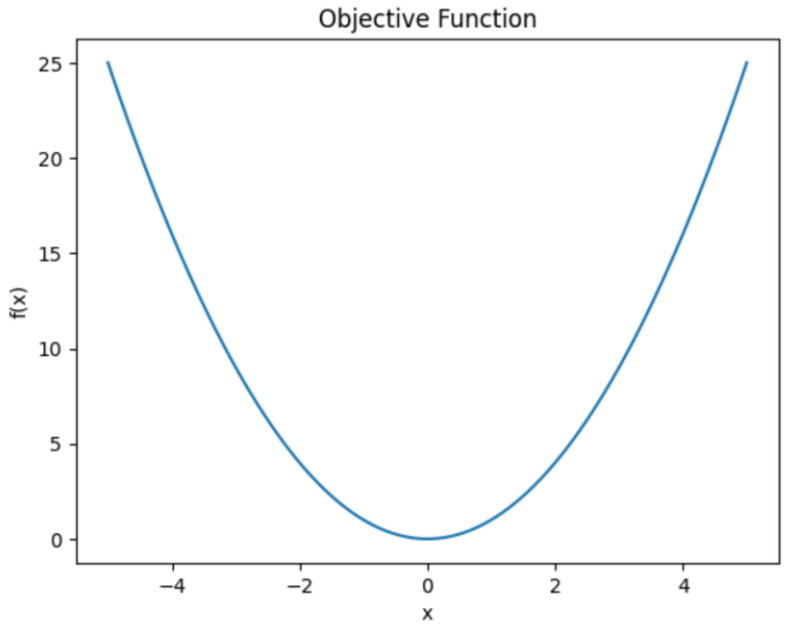
	Algorithm 1	Algorithm 2	Algorithm 3	Algorithm 4	Algorithm 5	Algorithm 6	Algorithm 7	Algorithm 8	...
Problem 1	78.90158096	38.18696053	83.9788141	3.128185533	93.71767489	3.612131384	38.02555482	46.02033283	...
Problem 2	63.63661246	51.21726878	6.915100117	92.46504485	20.63056606	90.15194724	6.628150576	88.92628997	...
Problem 3	5.467817525	78.82129795	19.01963224	16.18471759	59.57316925	26.61430506	41.45446652	62.38540108	...
Problem 4	40.96337067	55.59045049	25.47959077	77.75563723	90.98183523	42.23275523	92.4381591	80.17316672	...
Problem 5	17.32640301	80.17604054	48.01380213	9.378352179	13.25844413	66.24497877	17.39991202	46.86218446	...
Problem 6	2.90117365	14.18732284	88.12091607	28.32526953	88.17950692	43.16349405	78.48956349	76.09121009	...
Problem 7	74.22339559	71.35440724	46.26625983	69.9710712	66.9510279	68.97533166	14.29350951	56.8139594	...
Problem 8	69.06790479	89.53420767	17.7105817	71.3419208	48.8622438	3.348772613	70.81053152	3.855765825	...
Problem 9	19.94675498	3.137513385	10.68373549	4.011603637	49.49135388	37.92530089	99.49914362	54.10622766	...
Problem 10	7.510870987	58.55534993	57.60647147	80.17271882	80.41639739	25.77488384	55.59960103	94.67596268	...
Problem 11	98.30840803	40.16271408	15.063453	80.71102508	67.38435353	2.092705478	54.93369837	34.34560747	...
Problem 12	56.35291015	99.47783881	73.23060569	79.11112105	58.89165367	51.21548188	72.3854659	54.63516655	...
Problem 13	42.95441914	5.055088383	20.45995021	60.02150262	2.129162205	0.03549031414	90.26590811	1.821852475	...
Problem 14	44.26664262	55.68963431	33.72502344	56.30721179	88.24480947	42.89040502	29.76489645	6.234549423	...
Problem 15	91.00330356	24.51201295	90.63002494	53.41813975	93.87696033	28.00711639	23.69333881	40.15298867	...
...	...	...	...	...	...	...	...	...	...
Average	100	100	100	100	100	100	100	100	

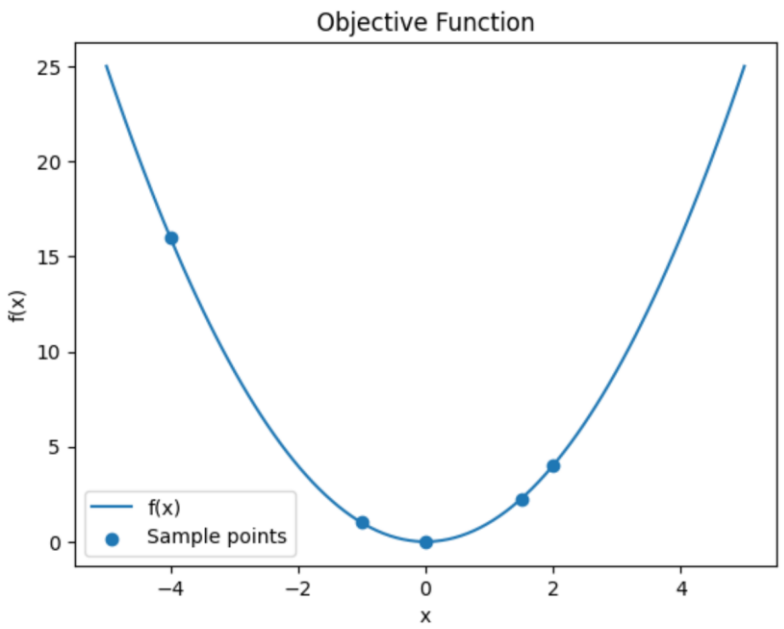
https://machinelearningmastery.com/no-free-lunch-theorem-for-machine-learning/

Optimization

- Visualization of the objective function – 2D

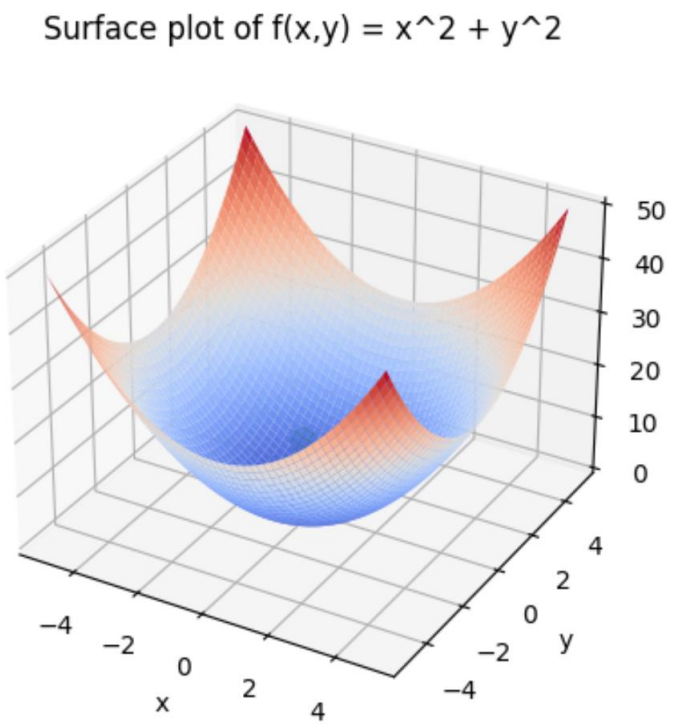
```python
import numpy as np
import matplotlib.pyplot as plt

# Define the objective function
def objective_function(x):
    return x**2

# Generate sample data for plotting
x = np.linspace(-5, 5, 100)
y = objective_function(x)

# Plot the objective function
plt.plot(x, y)

# Set the axis labels and title
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Objective Function')

# Display the plot
plt.show()
```

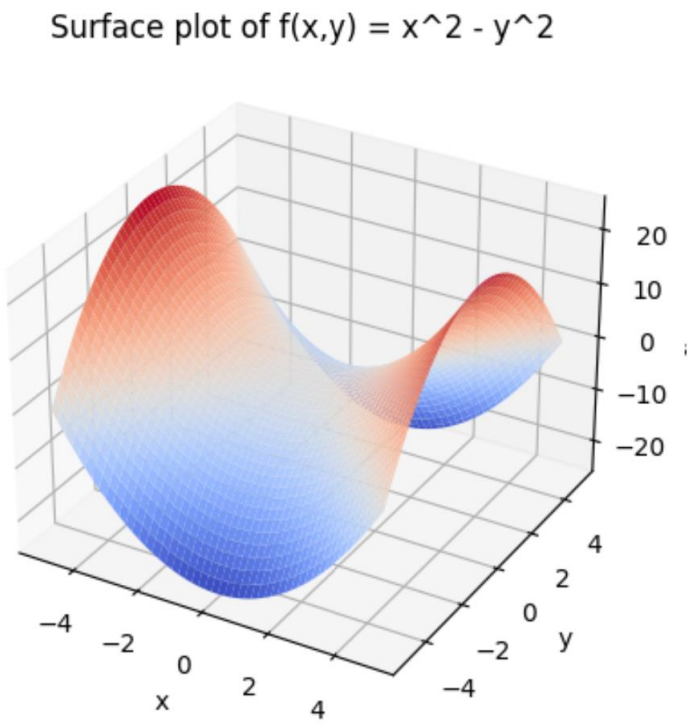




Optimization

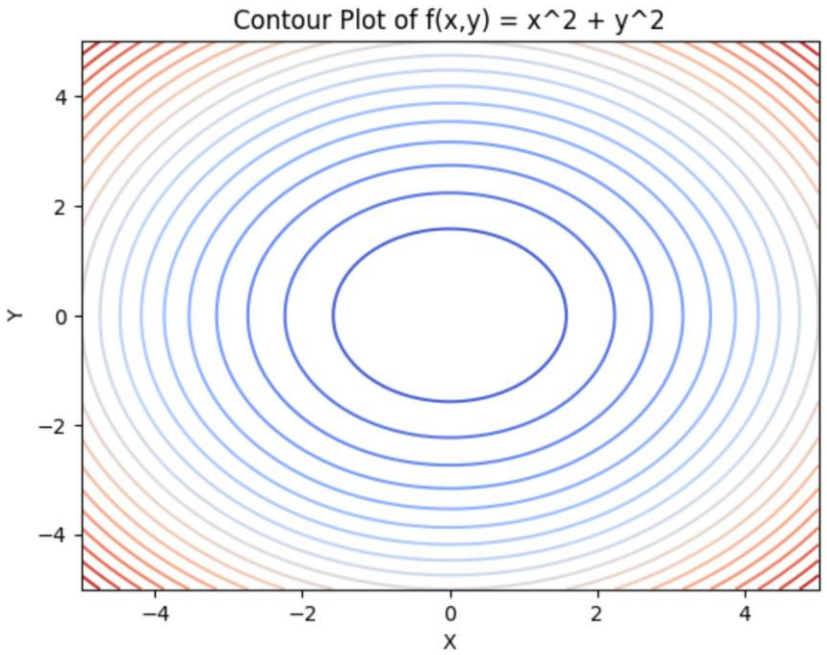
- Visualization of the objective function - Surface Plot

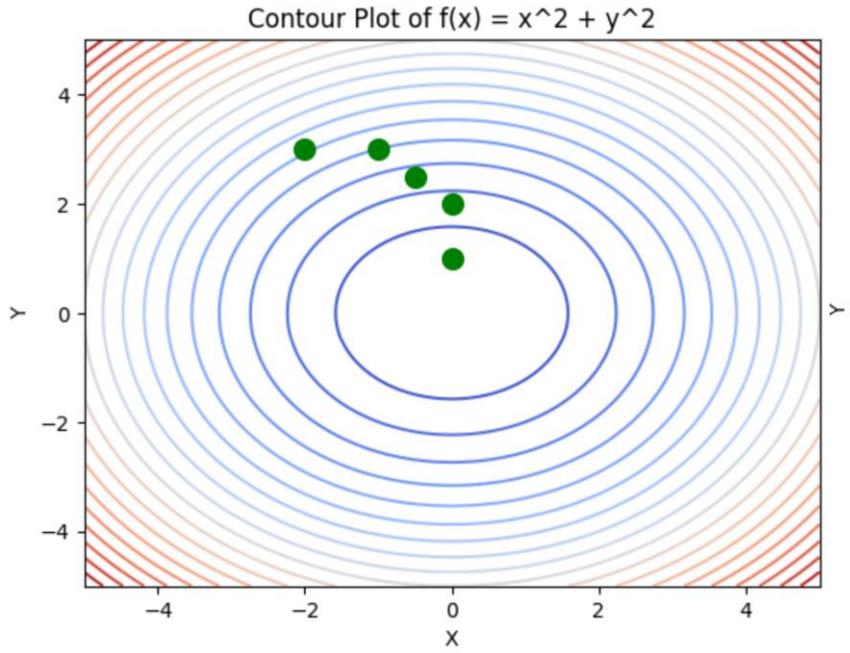


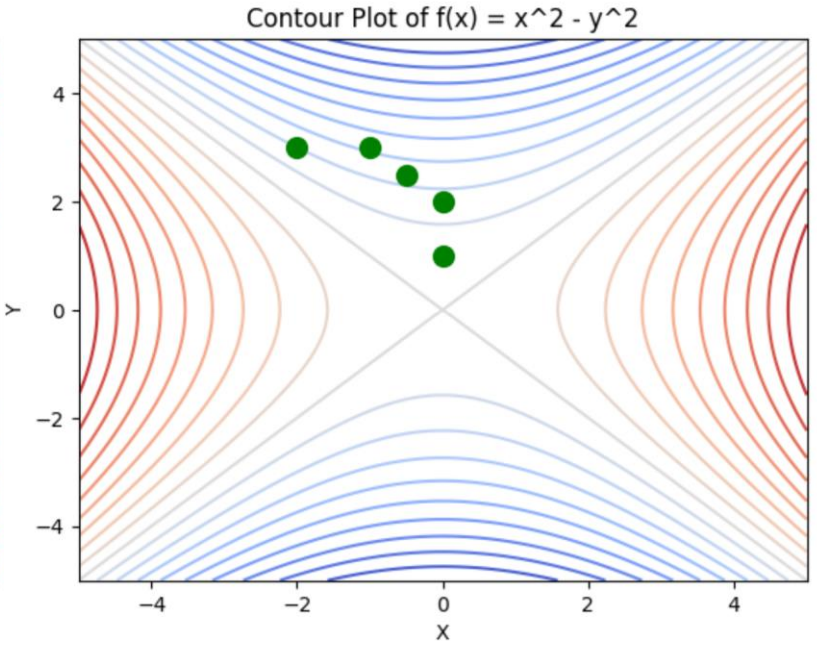


Optimization

- Visualization of the objective function - 2D







Optimization – Direct Approach / Non – Derivative Approach

- Random Search -
  - Finds the optimal input parameters for a given objective function
  - Generates random sets of input parameters and evaluate the objective function at each set
  - Starts by defining a search space, which is the range of values that each input parameter can take
  - Random sets of input parameters are then generated from the search space
  - The objective function is evaluated for each set of input parameters, and the results are recorded

Optimization - Random Search

- Continues to generate random sets of input parameters and evaluate the objective function until a stopping condition is met.
- The stopping condition can be a maximum number of evaluations, a target objective value, or a time limit, among others.
- The best set of input parameters found during the search is returned as the optimal solution.
- Random Search can be useful when the objective function is complex, non-convex, or has many local optima.

Optimization - Random Search

Algorithm [edit]

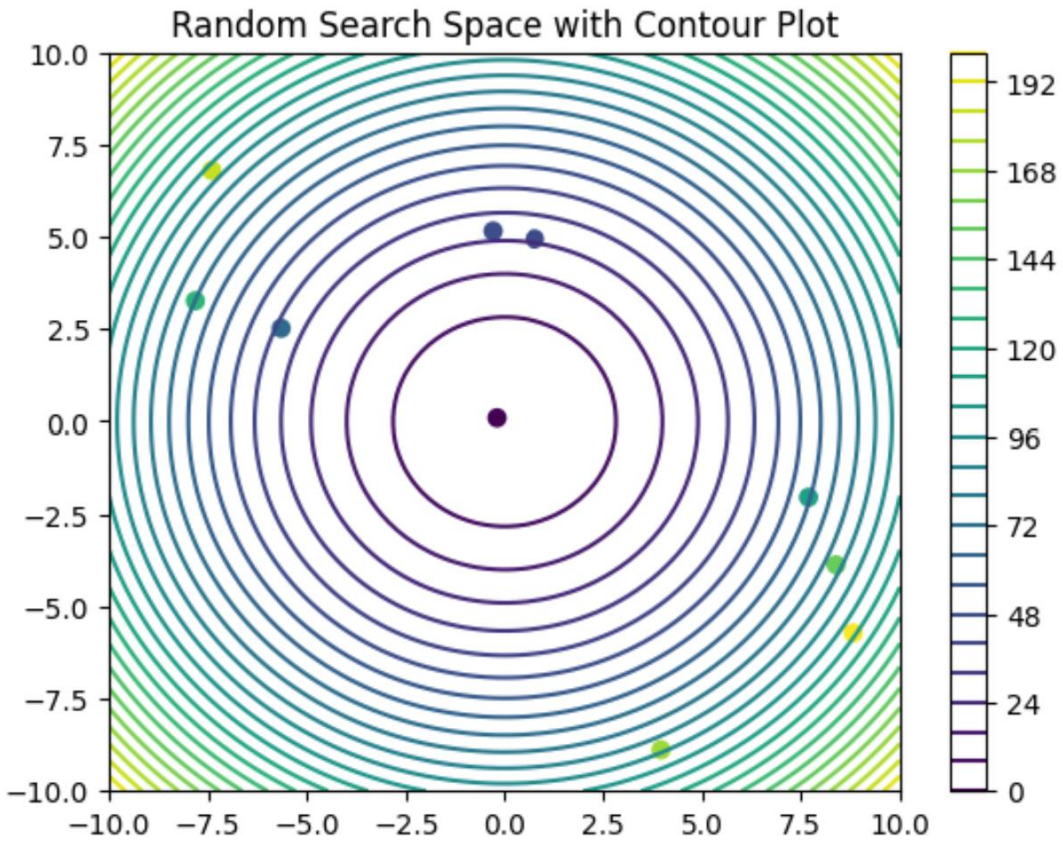
Let f: $\mathbb{R}^n \rightarrow \mathbb{R}$ be the fitness or cost function which must be minimized. Let $\mathbf{x} \in \mathbb{R}^n$ designate a position or candidate solution in the search-space. The basic RS algorithm can then be described as:

1. Initialize **x** with a random position in the search-space.
2. Until a termination criterion is met (e.g. number of iterations performed, or adequate fitness reached), repeat the following:
    1. Sample a new position **y** from the hypersphere of a given radius surrounding the current position **x** (see e.g. Marsaglia's technique for sampling a hypersphere.)
    2. If $f(\mathbf{y}) < f(\mathbf{x})$ then move to the new position by setting $\mathbf{x} = \mathbf{y}$

https://en.wikipedia.org/wiki/Random_search

Optimization - Random Search

```
# Define objective function
def f(x,y):
    return x**2 + y**2

# Define search space
x = np.linspace(-10, 10, 100)
y = np.linspace(-10, 10, 100)
X, Y = np.meshgrid(x, y)

# Generate random search points
num_points = 10
x_rand = np.random.uniform(-10, 10, num_points)
y_rand = np.random.uniform(-10, 10, num_points)
f_rand = f(x_rand, y_rand)
```

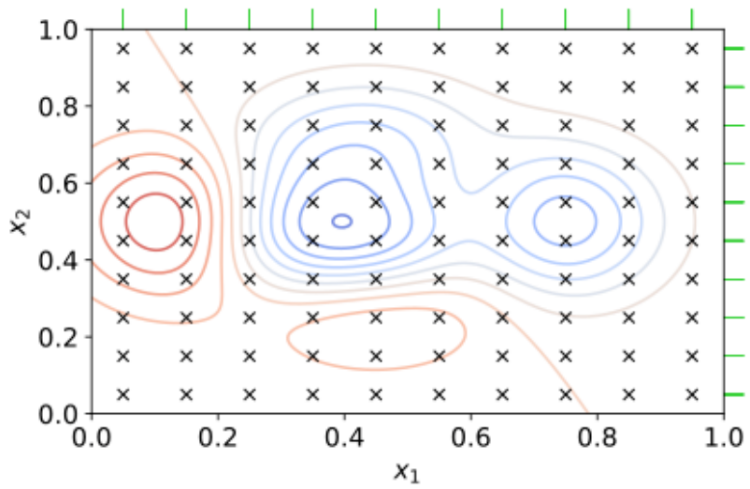


https://en.wikipedia.org/wiki/Random_search

Optimization – Grid Search

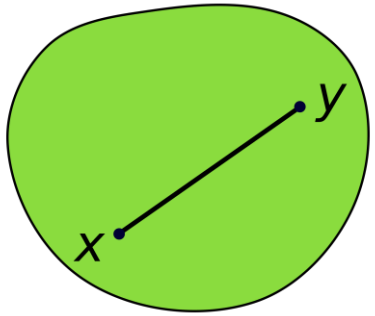
- Grid Search is a simple and systematic approach to finding the optimal set of parameters
- It Define a grid of (hyper) parameters with discrete values, where each point on the grid represents a unique combination of hyperparameters.
- The objective function is evaluated for each combination of hyperparameters - determine the best combination that maximizes or minimizes the objective function.
- It explores all possible combinations of hyperparameters - computationally expensive method for high-dimensional problems
- best suited for problems where the search space is relatively small and the number of hyperparameters is limited.
- It is commonly used as a benchmark for more advanced optimization algorithms such as random search and Bayesian optimization.

Optimization – Grid Search

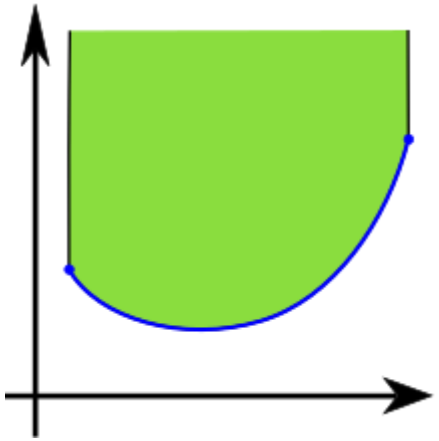


Grid search across different values of two hyperparameters. For each hyperparameter, 10 different values are considered, so a total of 100 different combinations are evaluated and compared. Blue contours indicate regions with strong results, whereas red ones show regions with poor results.

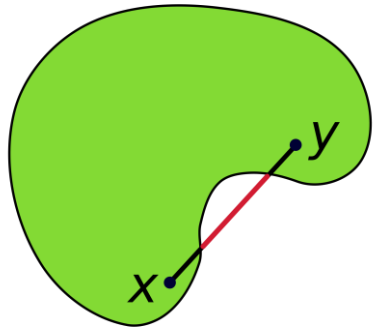
Optimization- convexity



The line segment, illustrated in black above, joining points x and y, lies completely within the set, illustrated in green. Since this is true for any potential locations of any two points within the above set, the set is convex



A function is convex if and only if its epigraph, the region (in green) above its graph (in blue), is a convex set.



a non-convex set. Illustrated by the above line segment whereby it changes from the black color to the red color. Exemplifying why this above set, illustrated in green, is non-convex.

https://en.wikipedia.org/wiki/Convex_set

Gradient Descent

- It is the backbone of to train deep learning
- It is a **first order** **minimization** **optimization** algorithm
- The derivative of multivariate (objective) function → "**Gradient**"
- Why we use gradient?
  - To find the direction of the minima
- The sign of the **gradient** tells whether the objective function is increasing/decreasing
  - Positive gradient- function is increasing at that point
  - Negative gradient- function is decreasing at that point
- Why it is important to know the sign of the gradient?
  - As this is a minimization problem, we search of the minima in the negative direction of the gradient

Gradient Descent

- Can we apply gradient descent on all objective functions – differentiable/ non-differentiable?
  - Hint – Gradient descent search based on first-order derivative

Gradient Descent

- It is good for finding global extremum if the function is convex
- It is good for finding local extremum if the function is not convex
  - In real-world, many function is not convex
- It is used for optimizing many Machine learning algorithms:
  - Deep learning
  - Linear Regression
  - Logistic Regression
  - Support Vector Machines

Gradient Descent

- Gradient descent algorithm:

$$x_{new} = x - \alpha f'(x)$$

  - Calculate gradient at a randomly selected point in the input space
  - Find the direction
  - How far to move – step size (learning rate)

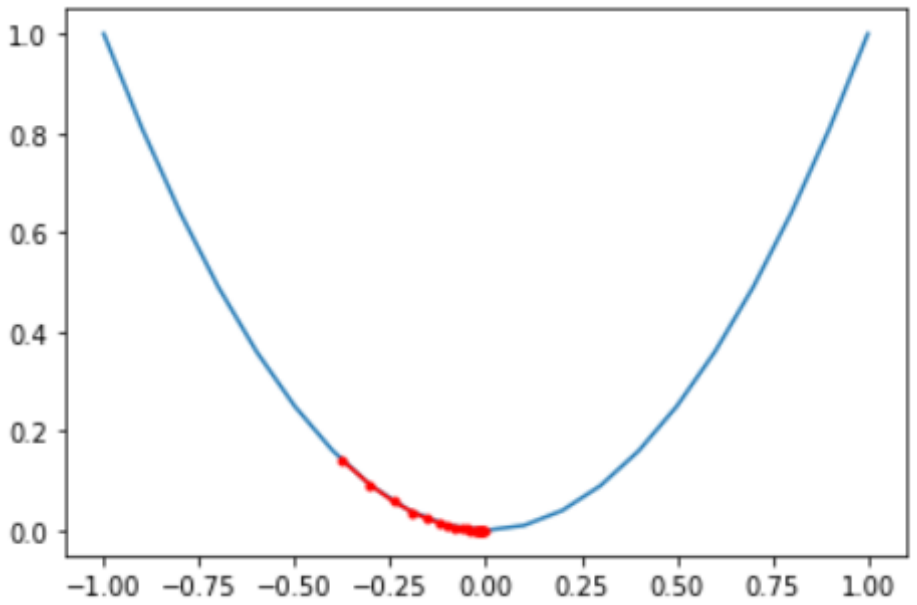
Gradient Descent

- Step size (learning rate)
  - Hyperparameter
  - If step size is too small – slow movement – take a long time to converge
  - If step size is too large – the search may bounce around the search space – may diverge
  - Finding good step size may require some trail and error

Gradient Descent

- Step size (learning rate)

Step size = .1

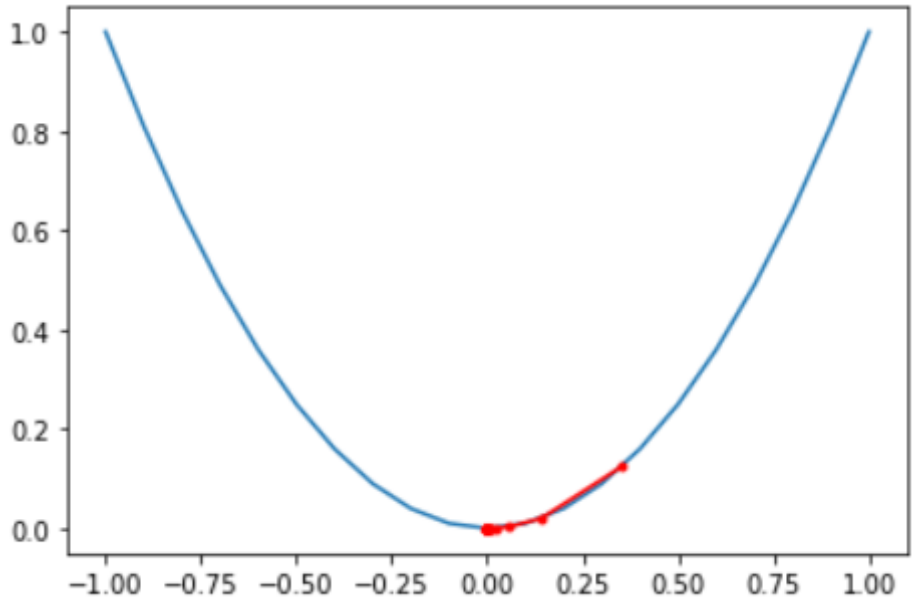



```
>0 f([-0.37803835]) = 0.14291
>1 f([-0.30243068]) = 0.09146
>2 f([-0.24194454]) = 0.05854
>3 f([-0.19355564]) = 0.03746
>4 f([-0.15484451]) = 0.02398
>5 f([-0.12387561]) = 0.01535
>6 f([-0.09910049]) = 0.00982
>7 f([-0.07928039]) = 0.00629
>8 f([-0.06342431]) = 0.00402
>9 f([-0.05073945]) = 0.00257
>10 f([-0.04059156]) = 0.00165
>11 f([-0.03247325]) = 0.00105
>12 f([-0.0259786]) = 0.00067
>13 f([-0.02078288]) = 0.00043
>14 f([-0.0166263]) = 0.00028
>15 f([-0.01330104]) = 0.00018
>16 f([-0.01064083]) = 0.00011
>17 f([-0.00851267]) = 0.00007
>18 f([-0.00681013]) = 0.00005
>19 f([-0.00544811]) = 0.00003
```

Gradient Descent

- Step size (learning rate)

Step size = .3

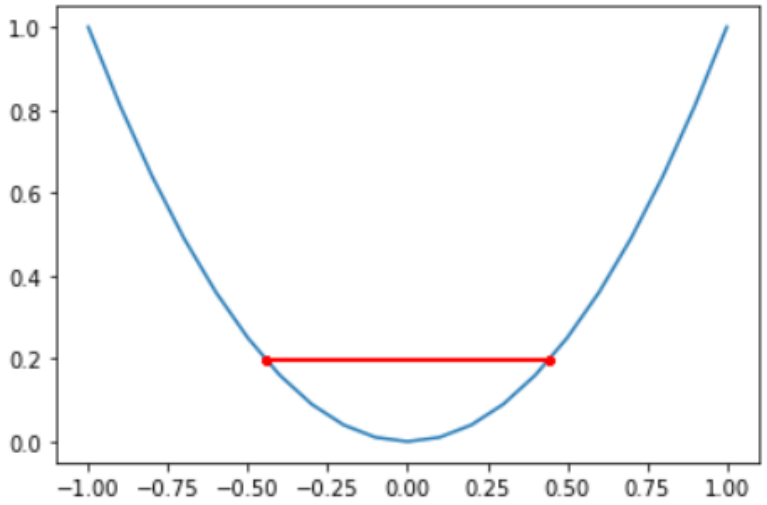



```
>0 f([0.35377041]) = 0.12515
>1 f([0.14150816]) = 0.02002
>2 f([0.05660327]) = 0.00320
>3 f([0.02264131]) = 0.00051
>4 f([0.00905652]) = 0.00008
>5 f([0.00362261]) = 0.00001
>6 f([0.00144904]) = 0.00000
>7 f([0.00057962]) = 0.00000
>8 f([0.00023185]) = 0.00000
>9 f([9.27387908e-05]) = 0.00000
>10 f([3.70955163e-05]) = 0.00000
>11 f([1.48382065e-05]) = 0.00000
>12 f([5.93528261e-06]) = 0.00000
>13 f([2.37411304e-06]) = 0.00000
>14 f([9.49645217e-07]) = 0.00000
>15 f([3.79858087e-07]) = 0.00000
>16 f([1.51943235e-07]) = 0.00000
>17 f([6.07772939e-08]) = 0.00000
>18 f([2.43109176e-08]) = 0.00000
>19 f([9.72436703e-09]) = 0.00000
```

Gradient Descent

- Step size (learning rate)

Step size = 1

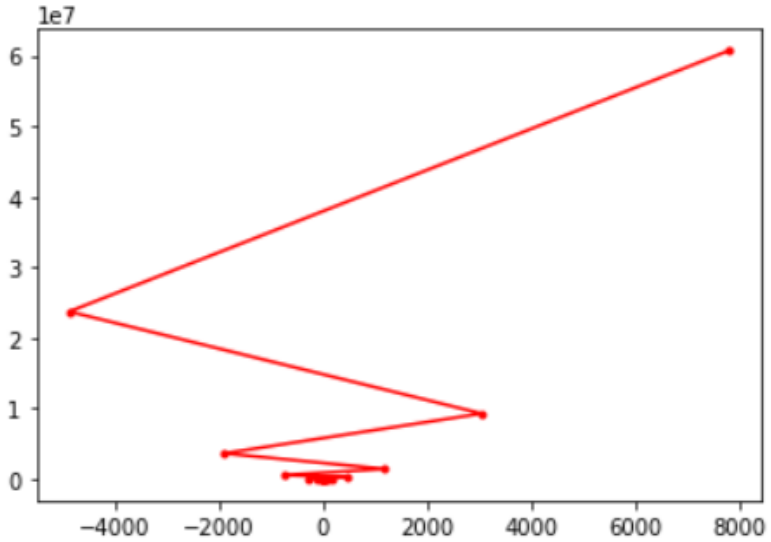



```
>0 f([-0.44198753]) = 0.19535
>1 f([0.44198753]) = 0.19535
>2 f([-0.44198753]) = 0.19535
>3 f([0.44198753]) = 0.19535
>4 f([-0.44198753]) = 0.19535
>5 f([0.44198753]) = 0.19535
>6 f([-0.44198753]) = 0.19535
>7 f([0.44198753]) = 0.19535
>8 f([-0.44198753]) = 0.19535
>9 f([0.44198753]) = 0.19535
>10 f([-0.44198753]) = 0.19535
>11 f([0.44198753]) = 0.19535
>12 f([-0.44198753]) = 0.19535
>13 f([0.44198753]) = 0.19535
>14 f([-0.44198753]) = 0.19535
>15 f([0.44198753]) = 0.19535
>16 f([-0.44198753]) = 0.19535
>17 f([0.44198753]) = 0.19535
>18 f([-0.44198753]) = 0.19535
>19 f([0.44198753]) = 0.19535
```

Gradient Descent

- Step size (learning rate)

Step size = 1.3

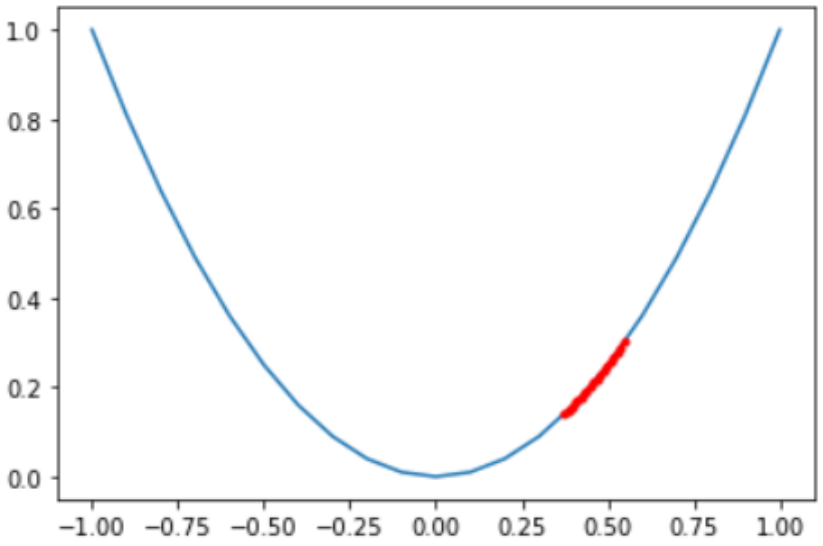



```
>0 f([-1.0314731]) = 1.06394
>1 f([1.65035696]) = 2.72368
>2 f([-2.64057113]) = 6.97262
>3 f([4.22491381]) = 17.84990
>4 f([-6.75986209]) = 45.69574
>5 f([10.81577935]) = 116.98108
>6 f([-17.30524695]) = 299.47157
>7 f([27.68839512]) = 766.64722
>8 f([-44.3014322]) = 1962.61689
>9 f([70.88229152]) = 5024.29925
>10 f([-113.41166643]) = 12862.20608
>11 f([181.45866629]) = 32927.24757
>12 f([-290.33386606]) = 84293.75378
>13 f([464.53418569]) = 215792.00968
>14 f([-743.25469711]) = 552427.54478
>15 f([1189.20751538]) = 1414214.51463
>16 f([-1902.7320246]) = 3620389.15745
>17 f([3044.37123936]) = 9268196.24307
>18 f([-4870.99398298]) = 23726582.38225
>19 f([7793.59037277]) = 60740050.89856
```

Gradient Descent

- Step size (learning rate)

Step size = 0.01 & iteration 20

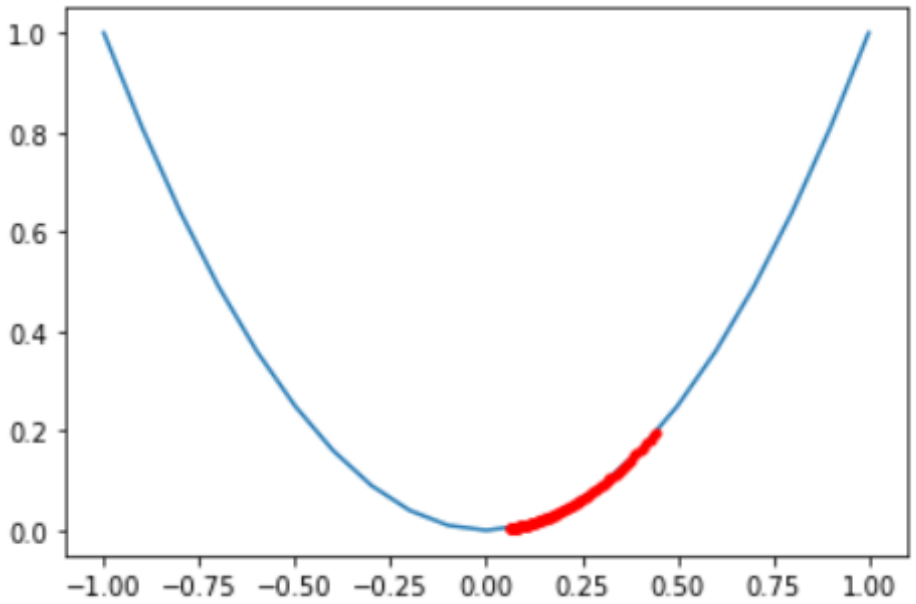



```
>0 f([0.54919471]) = 0.30161
>1 f([0.53821082]) = 0.28967
>2 f([0.5274466]) = 0.27820
>3 f([0.51689767]) = 0.26718
>4 f([0.50655972]) = 0.25660
>5 f([0.49642852]) = 0.24644
>6 f([0.48649995]) = 0.23668
>7 f([0.47676995]) = 0.22731
>8 f([0.46723455]) = 0.21831
>9 f([0.45788986]) = 0.20966
>10 f([0.44873207]) = 0.20136
>11 f([0.43975742]) = 0.19339
>12 f([0.43096228]) = 0.18573
>13 f([0.42234303]) = 0.17837
>14 f([0.41389617]) = 0.17131
>15 f([0.40561825]) = 0.16453
>16 f([0.39750588]) = 0.15801
>17 f([0.38955576]) = 0.15175
>18 f([0.38176465]) = 0.14574
>19 f([0.37412936]) = 0.13997
```

Gradient Descent

- Step size (learning rate)

Step size = 0.01 & iteration = 100

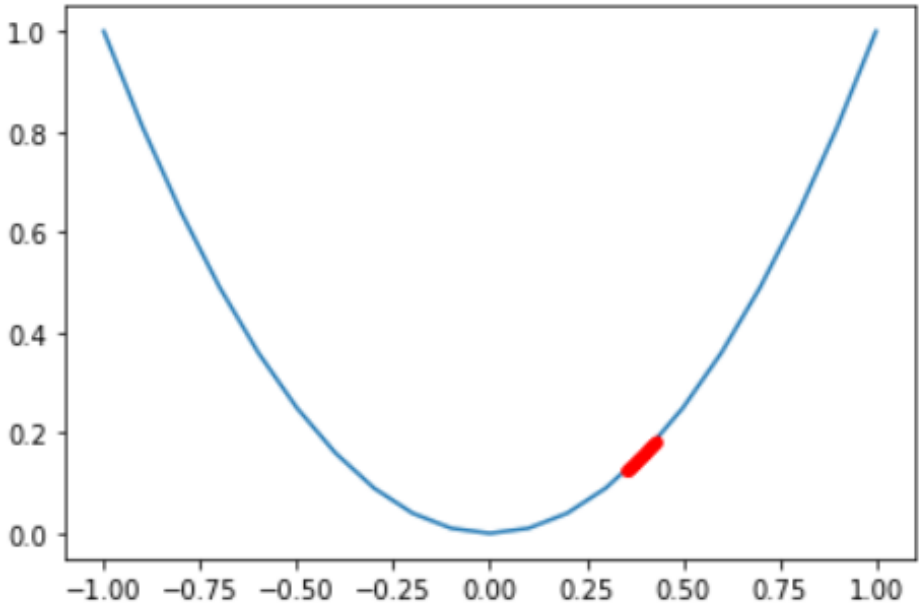



```
>80 f([0.08840912]) = 0.00782
>81 f([0.08664094]) = 0.00751
>82 f([0.08490812]) = 0.00721
>83 f([0.08320996]) = 0.00692
>84 f([0.08154576]) = 0.00665
>85 f([0.07991484]) = 0.00639
>86 f([0.07831655]) = 0.00613
>87 f([0.07675021]) = 0.00589
>88 f([0.07521521]) = 0.00566
>89 f([0.07371091]) = 0.00543
>90 f([0.07223669]) = 0.00522
>91 f([0.07079195]) = 0.00501
>92 f([0.06937611]) = 0.00481
>93 f([0.06798859]) = 0.00462
>94 f([0.06662882]) = 0.00444
>95 f([0.06529624]) = 0.00426
>96 f([0.06399032]) = 0.00409
>97 f([0.06271051]) = 0.00393
>98 f([0.0614563]) = 0.00378
>99 f([0.06022718]) = 0.00363
```

Gradient Descent

- Step size (learning rate)

Step size = 0.001 &
iteration = 100



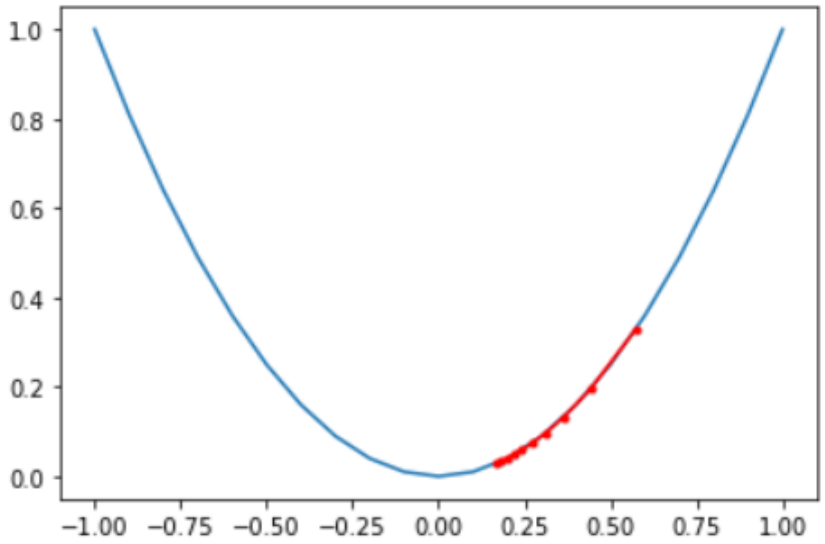

```
>80 f([0.36627158]) = 0.13415
>81 f([0.36553904]) = 0.13362
>82 f([0.36480796]) = 0.13308
>83 f([0.36407835]) = 0.13255
>84 f([0.36335019]) = 0.13202
>85 f([0.36262349]) = 0.13150
>86 f([0.36189824]) = 0.13097
>87 f([0.36117445]) = 0.13045
>88 f([0.3604521]) = 0.12993
>89 f([0.35973119]) = 0.12941
>90 f([0.35901173]) = 0.12889
>91 f([0.35829371]) = 0.12837
>92 f([0.35757712]) = 0.12786
>93 f([0.35686197]) = 0.12735
>94 f([0.35614824]) = 0.12684
>95 f([0.35543595]) = 0.12633
>96 f([0.35472507]) = 0.12583
>97 f([0.35401562]) = 0.12533
>98 f([0.35330759]) = 0.12483
>99 f([0.35260098]) = 0.12433
```

Gradient Descent

- Gradient-based dynamic learning rate
  - If the starting value is far away from the minima → the gradient is large
  - Approaches towards minima → gradients approaches towards 0
  - Learning rate = learning rate * abs(gradient)

Gradient Descent

- Step size (learning rate) – **Dynamic**
- Gradient-based dynamic learning rate



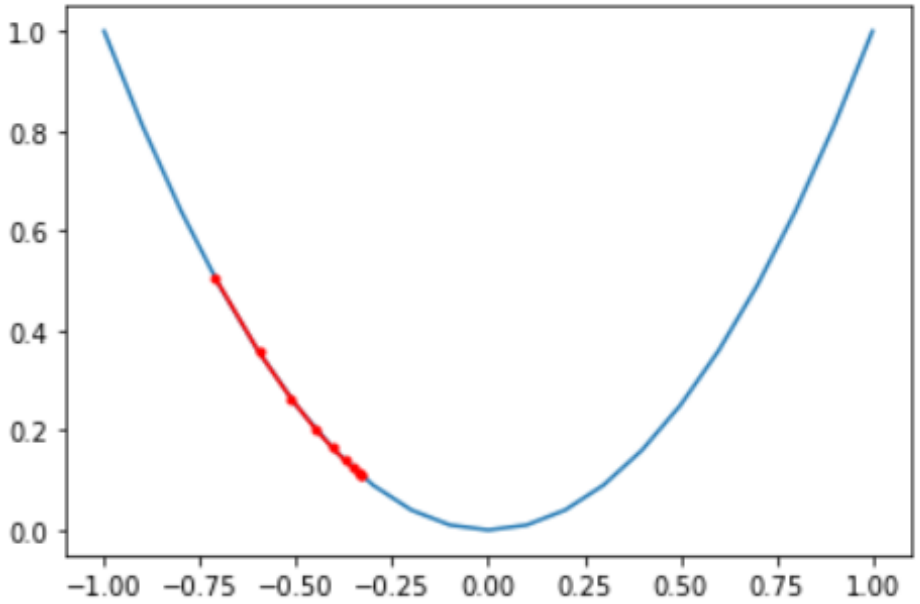
	gradient	step_size
0	[1.7791062749017068]	[0.1779106274901707]
1	[1.1460624474227812]	[0.11460624474227812]
2	[0.8833706207442422]	[0.08833706207442422]
3	[0.7273018900254287]	[0.07273018900254287]
4	[0.6215082821785165]	[0.06215082821785165]
5	[0.5442537732152184]	[0.05442537732152184]
6	[0.48501133928341794]	[0.048501133928341794]
7	[0.43796413943671897]	[0.0437964139436719]
8	[0.3996016219502098]	[0.039960162195020986]
9	[0.3676653306971621]	[0.036766533069716216]

Gradient Descent

- Step size (learning rate) – **Dynamic**
- Time-based dynamic learning rate
  - Adjusting the learning rate according to the training epoch
  - Learning rate = learning rate $(1 - \frac{epoch\ number}{total\ epochs})$

Gradient Descent

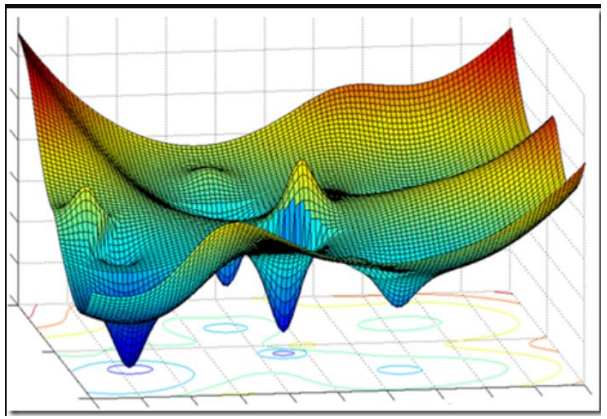
- Step size (learning rate) – **Dynamic**
- Time-based dynamic learning rate

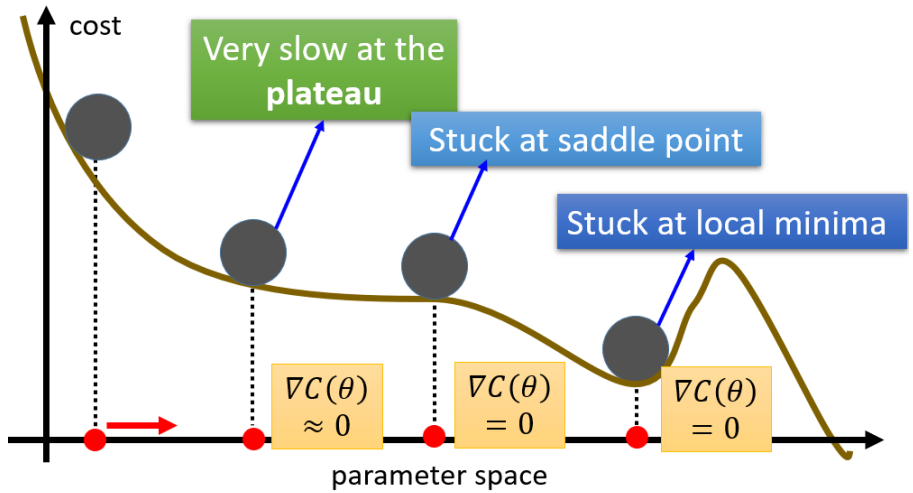


	step_size
0	0.10
1	0.09
2	0.08
3	0.07
4	0.06
5	0.05
6	0.04
7	0.03
8	0.02
9	0.01

Gradient Descent

- Challenges with gradient descent
  - Local minima
  - Vanishing gradient
  - Exploding gradient
  - Saddle point
- Gradient descent is guaranteed to go **downhill**
  - Can not guaranteed a best or near best solution



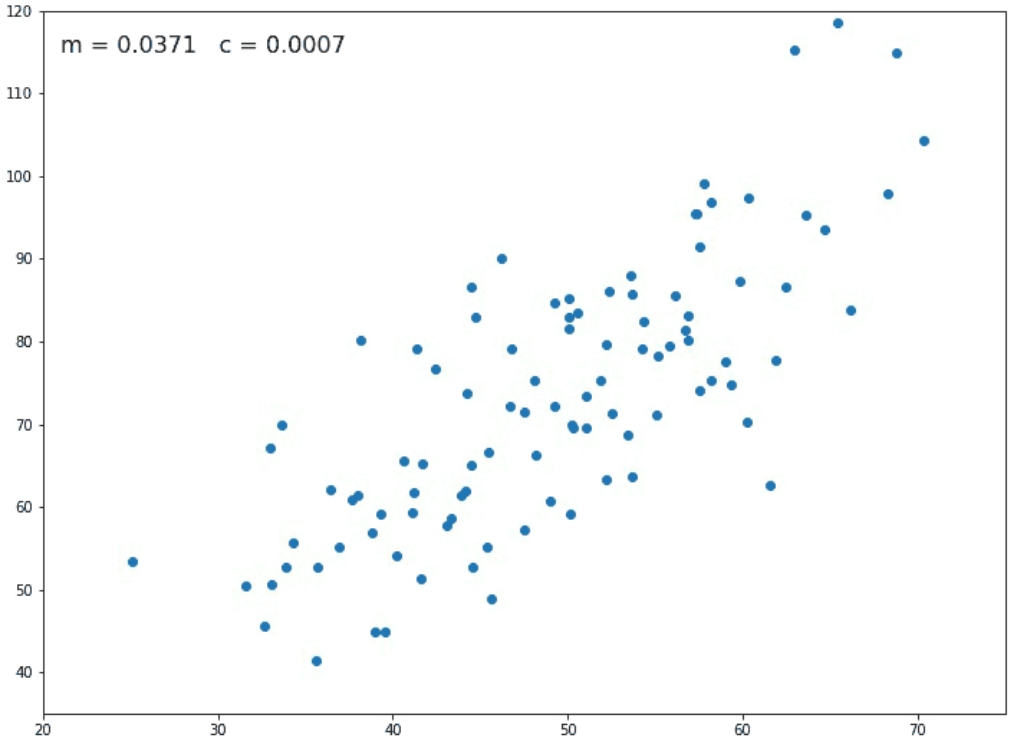


https://lh6.ggpht.com/-5RFMcz2bzWI/VA9NuUkB_vI/AAAAAAAAAdM/_DqqYO0nbX4/im2_thumb%25255B1%25255D.png?imgmax=800
Slide credit – Hung –yi Lee

Linear Regression with Gradient Descent

Steps

1. Find loss function
2. Set initial value of parameters and hyperparameters
3. Calculate the partial derivatives of the loss function
4. Define the update equation (GD)
5. Repeat the process



GIF source - https://towardsdatascience.com/linear-regression-using-gradient-descent-97a6c8700931