**An analysis of the Gradient Descent Algorithms - Final Project**

Assiya Karatay, Ayomide Awojobi, Ruiqi Chang

Graduate Students

BU MET College, Computer Science Department

Spring, 2022

CS566 - Professor Alexander Belyaev

**Table of Contents**

1. Abstract
2. Introduction
3. Explanation and implementation of algorithms
4. Pros and cons
5. Time and Space Complexity analysis
6. Conclusion
7. Glossary
8. References

**Abstract**

Gradient descent is an easy to understand and implement popular optimization strategy used in machine learning and deep learning. Gradient descent is used in machine learning to find the values of a function’s parameters which are basically the coefficients that minimize a cost function as far as possible. The goal of the project is review and analysis of the different types of this algorithm such as Stochastic, Batch, and Mini-Batch.

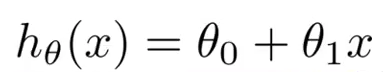
**Introduction**

Gradient Descent Algorithm was invented by famous French mathematician Augustin-Louis Cauchy to solve quadratic problems in astronomy (Cauchy, 1847). Gradient descent is not only limited to machine learning and deep learning. It is used widely in other areas such as control engineering(robotics, chemical, etc), computer games and mechanical engineering (Kwiatkowski, 2021).

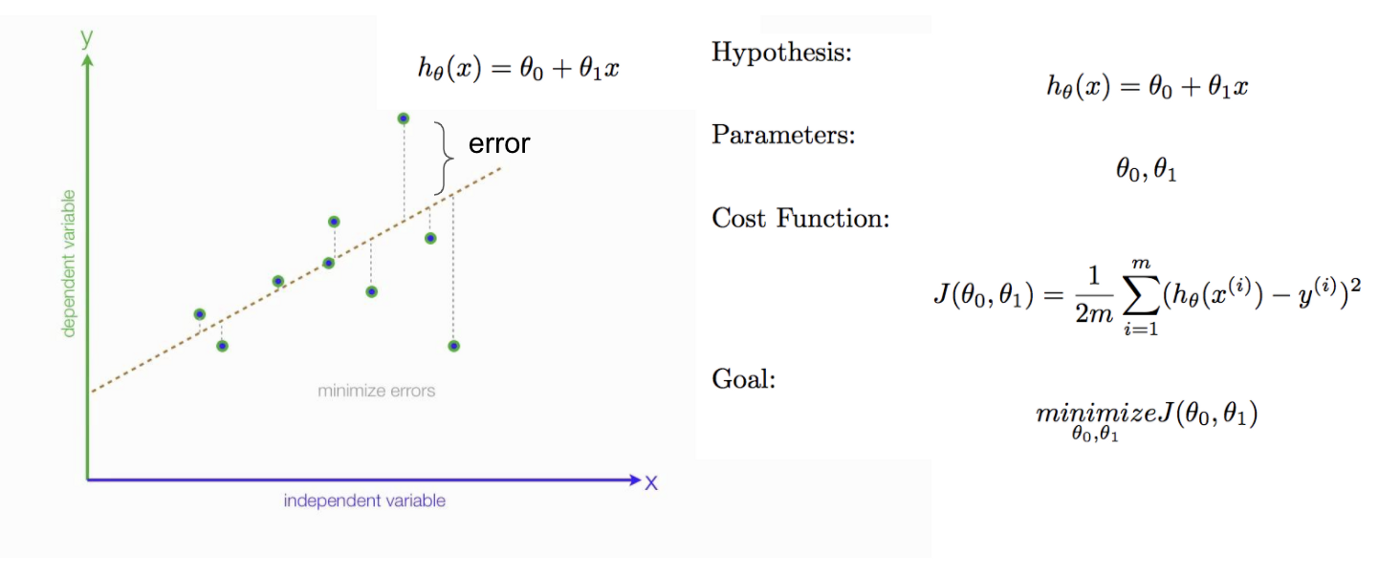
The basic concept of the Gradient Descent Algorithm can be explained in a simple way. It is like people go down from a mountain to the lowest point , and the easiest way is to follow the gradient which is the steepest slope down the mountain. This is the gradient descent formula. When the gradient descent algorithm is applied, this formula should be repeated until convergence. The α means the learning rate which represents the size of the next step, and the right part of the α represents direction. Thus, it is important to get the appropriate α and the right direction through iterating data points in a large database (Pandey, 2019).

The steps of Gradient Descent Algorithm include choosing the initial point and finding the gradient of it. Then we make a scaled step opposite to the direction of the gradient for the minimum or in the direction of the gradient for the maximum. We repeat these steps until we get the optimum which is minimize the Cost Function (Kwiatkowski, 2021). Below are the formulas.

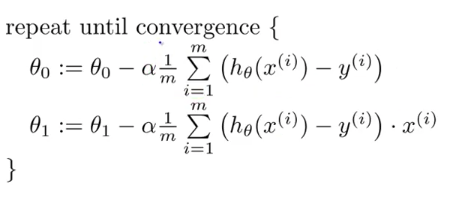
Hypothesis for Regression problem



Cost function(mean difference squared):



The derivation for the cost function(slop):



(Liu,2020)

In this project, we would examine different types of Gradient Descent Algorithms: Stochastic, Batch, and Mini-Batch Gradient Descent. The difference within these types is that Stochastic Gradient Descent takes one example in each iteration. Mini-Batch Gradient Descent takes a random sample in predetermined size while the Batch one takes the whole set in each iteration. In the last one, the batch size is equal to the data set size. With these different types of algorithms, we would compare them for design and complexity.

**Batch Gradient Descent**

Batch Gradient Descent is a type of gradient descent which processes all the training examples for each iteration of gradient descent. But if the number of training examples is large, then batch gradient descent is computationally very expensive. Since the gradients for the whole dataset to perform one parameter update, batch gradient descent can be very slow. Hence if the number of training examples is large, then batch gradient descent is not preferred. However, there are some advantages. First, We can use a fixed learning rate during training without worrying about learning rate decay. The most important is to have an unbiased estimate of gradients. The more the examples, the lower the standard error. (Dabbura,2017)

**Python Implementation of Stochastic Gradient Descent**(Hong)

alpha = 0.01 #learning rate, ep = 0.01 # convergence criteria

def gradient\_descent(alpha, x, y, ep, max\_iter=10000):

m = number of samples

# initial theta: t0, t1

# total error, J(theta) for check if current parameter converged

J = sum([(t0 + t1\*x[i] - y[i])\*\*2 for i in range(m)])

# Iterate Loop, for each training sample, compute the gradient (d/d\_theta j(theta))

grad0 = 1.0/m \* sum([(t0 + t1\*x[i] - y[i]) for i in range(m)])

grad1 = 1.0/m \* sum([(t0 + t1\*x[i] - y[i])\*x[i] for i in range(m)])

# update the theta

t0 = t0 - alpha \* grad0

t1 = t1 - alpha \* grad1

# mean squared error

e = sum( [ (t0 + t1\*x[i] - y[i])\*\*2 for i in range(m)] )

Converged,

if abs(J-e) <= ep

**Mini-Batch Gradient Descent**

Mini-Batch GD updates parameters after calculating the gradient of error with respect to a subset called mini-batch. Since we take a subset, there are advantages such as quick updates of parameters and vectorizing the code. Based on the batch size, the updates can be less noisy comparatively to SGD. Thus, this type of GD algorithm is more flexible and robust than other two types because of the fast convergence and the noise of the gradient update. (Khosla, 2019)

The pseudocode of the algorithm is represented below.

*Let theta = model parameters and max\_iters = number of epochs.*

*for itr = 1, 2, 3, …, max\_iters:*

*for mini\_batch (X\_mini, y\_mini):*

*Forward Pass on the batch X\_mini:*

*Make predictions on the mini-batch*

*Compute error in predictions (J(theta)) with the current values of the parameters*

*Backward Pass:*

*Compute gradient(theta) w.r.t. Theta*

*Update parameters:*

*theta = theta – learning\_rate\*gradient(theta)*

**The Python Implementation of Mini-Batch GD**

# function to perform mini-batch gradient descent

def miniBatchGradientDescent(X, y, learning\_rate = 0.001, batch\_size = 32):

theta = np.zeros((X.shape[1], 1))

error\_list = []

max\_iters = 3

for itr in range(max\_iters):

mini\_batches = create\_mini\_batches(X, y, batch\_size)

for mini\_batch in mini\_batches:

X\_mini, y\_mini = mini\_batch

theta = theta - learning\_rate \* gradient(X\_mini, y\_mini, theta)

error\_list.append(cost(X\_mini, y\_mini, theta))

return theta, error\_list

**Stochastic Gradient Descent**

Stochastic gradient descent is an iterative method often used for machine learning, optimizing the gradient descent during each search once a random weight vector is picked. This is a strategy that searches through a large infinite hypothesis space whenever there are hypotheses continuously being parameterized and the errors are differentiable based on the parameters. A problem with gradient descent is that converging to a local minimum takes extensive time and determining a global minimum is not guaranteed. Stochastic gradient descent is used in neural networks and decreases machine computation time while increasing complexity and performance for large-scale problems.

**Python Implementation of Stochastic Gradient Descent**

def SGD(f, theta0, alpha, num\_iters):

"""

Arguments:

f -- the function to optimize, it takes a single argument

and yield two outputs, a cost and the gradient

with respect to the arguments

theta0 -- the initial point to start SGD from

num\_iters -- total iterations to run SGD for

Return:

theta -- the parameter value after SGD finishes

"""

start\_iter = 0

theta = theta0

for iter in xrange(start\_iter + 1, num\_iters + 1):

\_, grad = f(theta)

# there is NO dot product ! return theta

theta = theta - (alpha \* grad)

**Pros & Cons**

Mini-Batch Gradient Descent pros:

* Convergence is more stable than stochastic gradient descent
* It is computationally efficient
* Fast learning since we perform more updates

Mini-Batch Gradient Descent cons:

* We have to configure the mini-batch size hyperparameter

Stochastic Gradient Descent pros:

* It is easier to fit into memory due to a single training sample being processed by the network.
* It is computationally fast as only one sample is processed at a time
* For larger datasets, it can converge faster as it causes updates to the parameters more frequently

Stochastic Gradient Descent cons:

* Can veer off in the wrong direction due to frequent updates
* Due to noisy steps, it will take longer to achieve convergence to the minima of the loss function
* Frequent updates are computationally expensive due to using all resources for processing one training sample at a time

Batch Gradient Descent pros:

* More stable convergence and error gradient than stochastic gradient descent
* Embraces the benefits of vectorization
* A more direct path is taken to the minimum

Batch Gradient Descent cons:

* Some of the data may be redundant and not contribute much to the update.
* Slower learning since an update is performed only after we go through all observations

**Time and Space Complexity analysis**

The computational cost of gradient descent depends on the number of iterations it takes to converge. The complexity of gradient descent is O(kn^2) where k is the number of iterations. When n is large, it is recommended to use gradient descent as it is more efficient compared to the closed form of linear regression.

**Conclusion**

The Gradient Descent algorithm is a powerful algorithm to get the best-optimized parameters in linear regression problems where the loss function value is minimized. The learning rate and Gradient are two more important values in this algorithm. The three main types of Gradient Descent algorithms apply different ways to minimize the Cost Function. Through the more detailed analysis, the three methods’ upsides and downsides list clearly. Finally, the python implementations and analysis are obvious to show the actual workings and performance behind the scenes of the famous gradient descent algorithm.

**Glossary**

GD - Gradient Descent

SGD - Stochastic Gradient Descent

**References**

Cauchy, A. (1847). 1. Methode generale pour la resolution des systemes d’equations simultanees. *Comptes rendus hebdomadaires des séances de l’Académie des sciences,* 536-544.<https://gallica.bnf.fr/ark:/12148/bpt6k2982c.image.f540.pagination.langEN>

Dabbura, I. (2019, September 3). Gradient descent algorithm and its variants. Medium. Retrieved April 24, 2022, from <https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3>

Donges, N. (n.d.). *Gradient descent: An introduction to 1 of Machine Learning's most popular algorithms*. Built In. Retrieved February 14, 2022, from<https://builtin.com/data-science/gradient-descent>

Hong,k(n.d.). *Python tutorial. Python Tutorial: batch gradient descent algorithm* - 2020. (n.d.). Retrieved April 24, 2022, from

<https://www.bogotobogo.com/python/python_numpy_batch_gradient_descent_algorithm.php>

Kwiatkowski, R. (2021, May 24). *Gradient descent algorithm - a deep dive*. Medium. Retrieved February 14, 2022,from [ttps://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21](https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21)

Khosla, S.(2019, January 23). *ML: Mini-batch gradient descent with python*. GeeksforGeeks. Retrieved March 6, 2022, from <https://www.geeksforgeeks.org/ml-mini-batch-gradient-descent-with-python/?ref=lbp>

Liu, C. (n.d.). 5 concepts you should know about gradient descent and cost function. KDnuggets. Retrieved April 24, 2022, from <https://www.kdnuggets.com/2020/05/5-concepts-gradient-descent-cost-function.html>

Oppermann, A. (2021, February 28). *Stochastic-, batch-, and mini-batch gradient descent demystified*. Medium. Retrieved April 11, 2022, from <https://towardsdatascience.com/stochastic-batch-and-mini-batch-gradient-descent-demystified-8b28978f7f5>

Pandey, P. (2019, March 18). *Understanding the mathematics behind gradient descent.* Medium. Retrieved February 21, 2022, from

<https://towardsdatascience.com/understanding-the-mathematics-behind-gradient-descent-dde5dc9be06e>

ML: Stochastic gradient descent (SGD). GeeksforGeeks. (2021, September 13). Retrieved March 20, 2022, from <https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/>