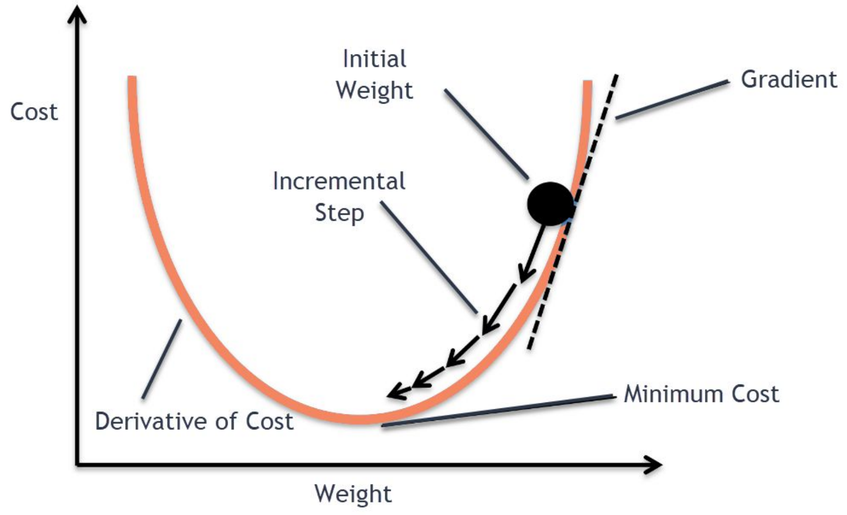
Gradient Descent

Lihau Pei, Peiyu Wang, Marie-Laure Brossay
Group 7

What is Gradient Descent ?

- Gradient Descent is an optimization technique which uses the calculus of steepest descent to find the minimum of a function.
- It uses the calculus of the negative of the gradient to move in the direction of descent.
- This is used in machine learning and optimization programs.

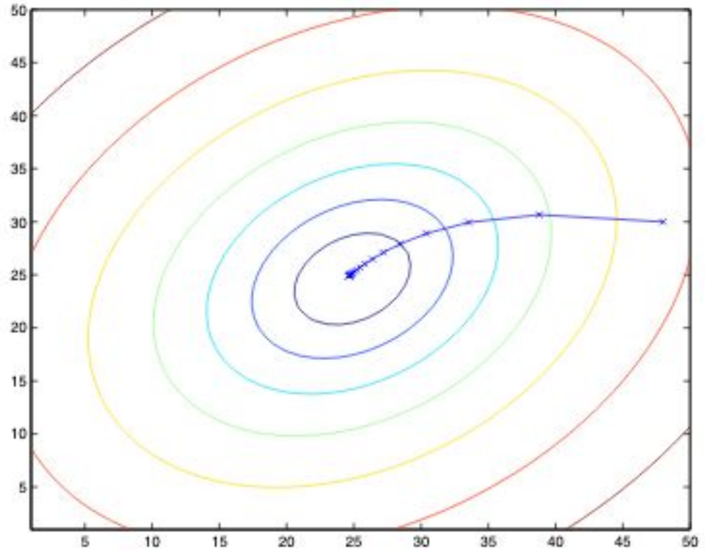


Why Gradient Descent?

- There are three different types of gradient descent:
  - Batch Gradient Descent
  - Stochastic Gradient Descent
  - Mini-batch Gradient Descent
- Gradient Descent is an extremely common optimization and learning algorithm, which is often used or combined with other algorithms in order to speed up an algorithm or increase accuracy.
- It is also possible to find both local and global minimums using gradient descent.
- Gradient Descent can allow for parallelization, be used to decrease amount of memory used, and simplify multivariable calculations.

Batch Gradient Descent (or Gradient Descent)

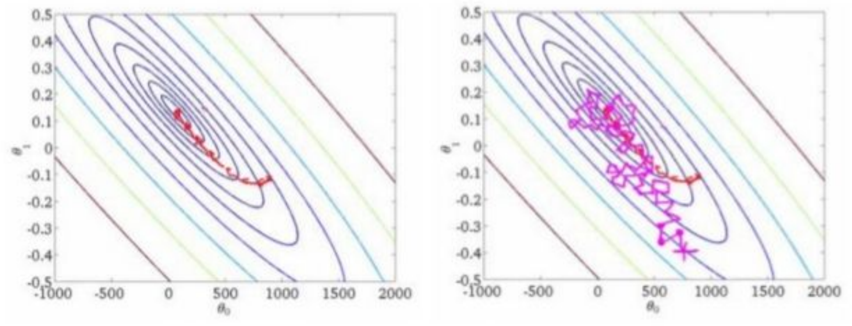
- Batch Gradient Descent uses the entire dataset to find the gradient with every iteration. This yields a very accurate solution/optimization.
- This type of gradient descent is best for very small data sets, as this is a very expensive algorithm.
- If extreme accuracy is needed, this is the best algorithm.
- This is also the most direct algorithm, where the iterations go in the exact direction of the minimum with every iteration.



Stochastic Gradient Descent

- Stochastic means random. This type of gradient descent randomly selects a sample from the training set to base approximations off of, rather than checking every data point during every iteration.
- This can be much faster than batch gradient descent, especially for very large data sets.
- Two problems with this are that the frequent updates can be computationally expensive, and the process of optimization has much more noise.

Illustration: batch gradient descent vs stochastic gradient descent



Batch: gradient

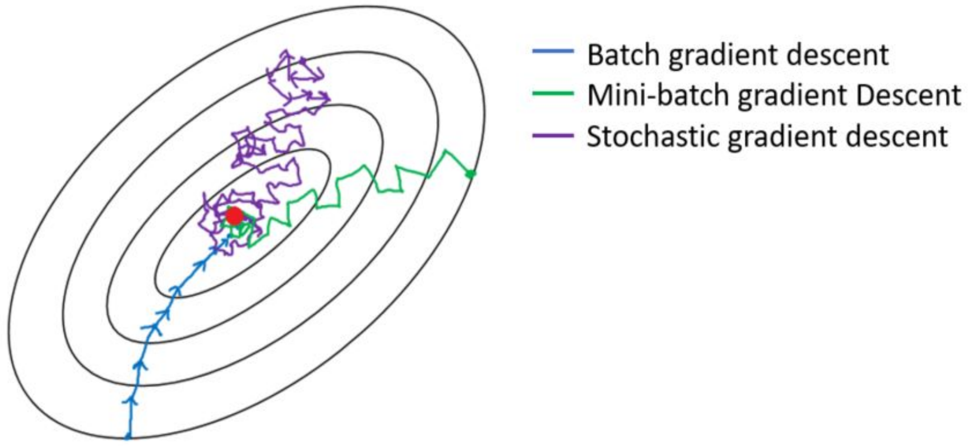
$$x \leftarrow x - \eta \nabla F(x)$$

Stochastic: single-example gradient

$$x \leftarrow x - \eta \nabla F_i(x)$$

Mini-batch Gradient Descent

- Mini-batch gradient descent is in between batch gradient descent and stochastic gradient descent.
- Mini-batch takes a sample of points from the training set and iterates using this sample.
- This is faster than batch gradient descent, and more accurate than stochastic gradient descent.
- There is still noise though, and does still take more time.
- This is the most common type for training an algorithm.



Differences in Algorithms

In the Batch Gradient Descent algorithm we did the gradients on each observation one by one for each iteration and changed the parameters to the direction of the least loss by one learning rate.

In Stochastic Gradient Descent we chose only one observation randomly for each observation.

In Mini batch gradient descent we used random samples but in batches. What this means is that we did not calculate the gradients for each observation but for a group of observations which results in a faster optimization.

Motivation

- Gradient descent is a popular topic within the realm of machine learning at the moment, and has an extremely wide breadth of applications.
- The three main types of gradient descent all have different purposes and priorities.
- Stochastic gradient descent specifically is very prominent and applicable, especially for very large data sets.
- The goal of this project is two-fold: analyzing the differences between the three main types of gradient descent, and comparing different stochastic gradient descent applications from journal papers.

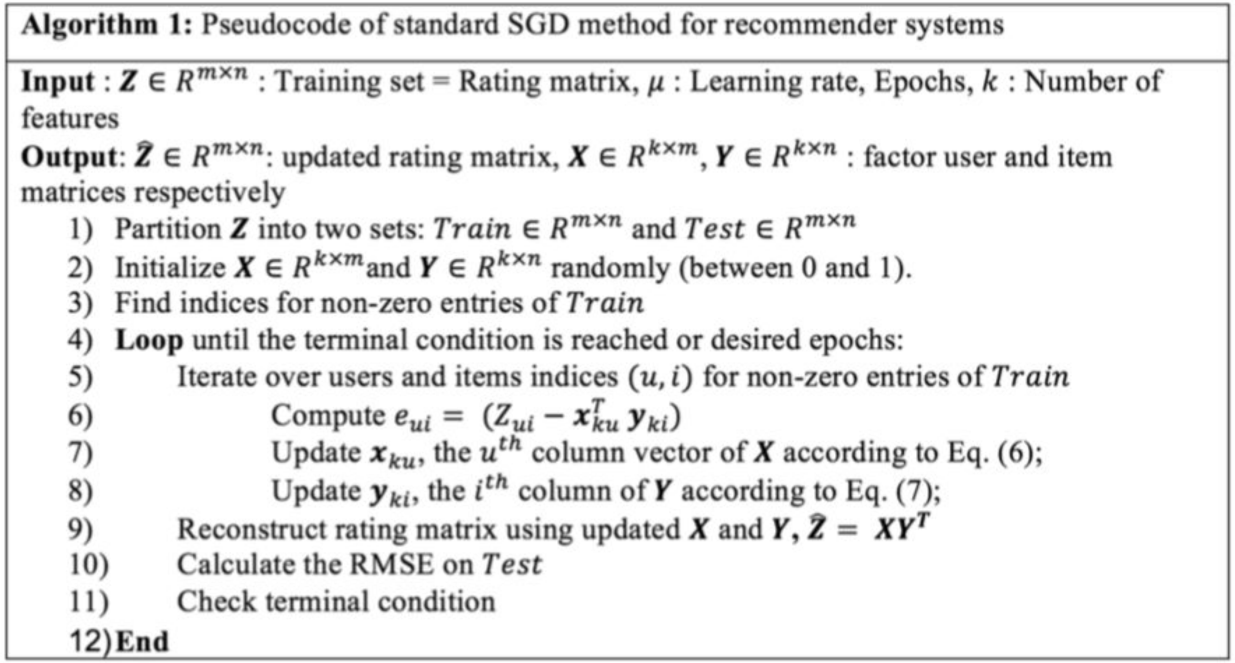
Methods of Altering Stochastic Gradient Descent

- One of the reasons that Stochastic Gradient Descent has become the most common type of Gradient Descent used is because of how many ways of changing it to suit an application there are.
- In the following slides, we will explain several of these adaptations by comparing journal papers which discuss these.

Papers Read

1. Fractional stochastic gradient descent for recommender systems
2. A simple Stochastic Gradient Descent Variational Bayes for the Correlated Topic Model
3. Preconditioned Stochastic Gradient Descent

Stochastic Gradient Descent for Recommender Systems



Algorithm 1: Pseudocode of standard SGD method for recommender systems

Input : $\boldsymbol{Z} \in R^{m \times n}$: Training set = Rating matrix, μ : Learning rate, Epochs, k : Number of features

Output: $\hat{\boldsymbol{Z}} \in R^{m \times n}$: updated rating matrix, $\boldsymbol{X} \in R^{k \times m}$, $\boldsymbol{Y} \in R^{k \times n}$: factor user and item matrices respectively

```
1)  Partition $\boldsymbol{Z}$ into two sets: $Train \in R^{m \times n}$ and $Test \in R^{m \times n}$
2)  Initialize $\boldsymbol{X} \in R^{k \times m}$ and $\boldsymbol{Y} \in R^{k \times n}$ randomly (between 0 and 1).
3)  Find indices for non-zero entries of $Train$
4)  **Loop** until the terminal condition is reached or desired epochs:
5)      Iterate over users and items indices $(u, i)$ for non-zero entries of $Train$
6)          Compute $e_{ui} = (Z_{ui} - \boldsymbol{x}_{ku}^T \boldsymbol{y}_{ki})$
7)          Update $\boldsymbol{x}_{ku}$, the $u^{th}$ column vector of $\boldsymbol{X}$ according to Eq. (6);
8)          Update $\boldsymbol{y}_{ki}$, the $i^{th}$ column of $\boldsymbol{Y}$ according to Eq. (7);
9)      Reconstruct rating matrix using updated $\boldsymbol{X}$ and $\boldsymbol{Y}$, $\hat{\boldsymbol{Z}} = \boldsymbol{X}\boldsymbol{Y}^T$
10)     Calculate the RMSE on $Test$
11)     Check terminal condition
12) **End**
```

$$\boldsymbol{x}_{ku} = \boldsymbol{x}_{ku} + \mu e_{ui} \boldsymbol{y}_{ki} \tag{6}$$

$$\boldsymbol{y}_{ki} = \boldsymbol{y}_{ki} + \mu e_{ui} \boldsymbol{x}_{ku} \tag{7}$$

Fractional Stochastic Gradient Descent For Recommender Systems:

Algorithm 2: Pseudocode of proposed FSGD method for recommender systems

Input: $\boldsymbol{Z} \in R^{m\times n}$: Training set = Rating matrix, μ : Learning rate, μ_{fr} : Fractional learning rate, Epochs, k: Number of features and fr: Fractional order

Output: $\widehat{\boldsymbol{Z}} \in R^{m\times n}$: updated rating matrix, $\boldsymbol{X} \in R^{k\times m}$, $\boldsymbol{Y} \in R^{k\times n}$: factor user and item matrices respectively

```
1)  Partition $\boldsymbol{Z}$ into two sets: $Train \in R^{m\times n}$ and $Test \in R^{m\times n}$
2)  Initialize $\boldsymbol{X} \in R^{k\times m}$ and $\boldsymbol{Y} \in R^{k\times n}$ randomly (between 0 and 1).
3)  Find indices for non-zero entries of $Train$
4)  Loop until the terminal condition is reached or desired epochs:
5)      Iterate over users and items indices $(u, i)$ for non-zero entries of $Train$
6)          Compute $e_{ui} = (Z_{ui} - \boldsymbol{x}_{ku}^T\,\boldsymbol{y}_{ki})$
7)          Update $\boldsymbol{x}_{ku}$, the $u^{th}$ column vector of $\boldsymbol{X}$ according to Eq. (14);
8)          Update $\boldsymbol{y}_{ki}$, the $i^{th}$ column of $\boldsymbol{Y}$ according to Eq. (15);
9)      Reconstruct rating matrix using updated $\boldsymbol{X}$ and $\boldsymbol{Y}$, $\widehat{\boldsymbol{Z}} = \boldsymbol{X}\boldsymbol{Y}^{\boldsymbol{T}}$
10)     Calculate the RMSE on $Test$
11)     Check terminal condition
12) End
```

$$\boldsymbol{x}_{ku} = \boldsymbol{x}_{ku} + \mu e_{ui}\boldsymbol{y}_{ki} + \frac{\mu_{fr}}{\Gamma(\boldsymbol{2}-fr)} e_{ui}\,\boldsymbol{y}_{ki} \odot |\boldsymbol{x}_{ku}|^{1-fr} \qquad (14)$$

$$\boldsymbol{y}_{ki} = \boldsymbol{y}_{ki} + \mu e_{ui}\boldsymbol{x}_{ku} + \frac{\mu_{fr}}{\Gamma(\boldsymbol{2}-fr)} e_{ui}\,\boldsymbol{x}_{ku} \odot |\boldsymbol{y}_{ki}|^{1-fr} \qquad (15)$$

Comparison of the two algorithms

RMSE: Root Mean Square Error (The smaller the better)

$$RMSE_{test} = \sqrt{mean\left(\sum_{(u,i)\in\Omega_{test}} e_{ui}^2\right)} \qquad (16)$$

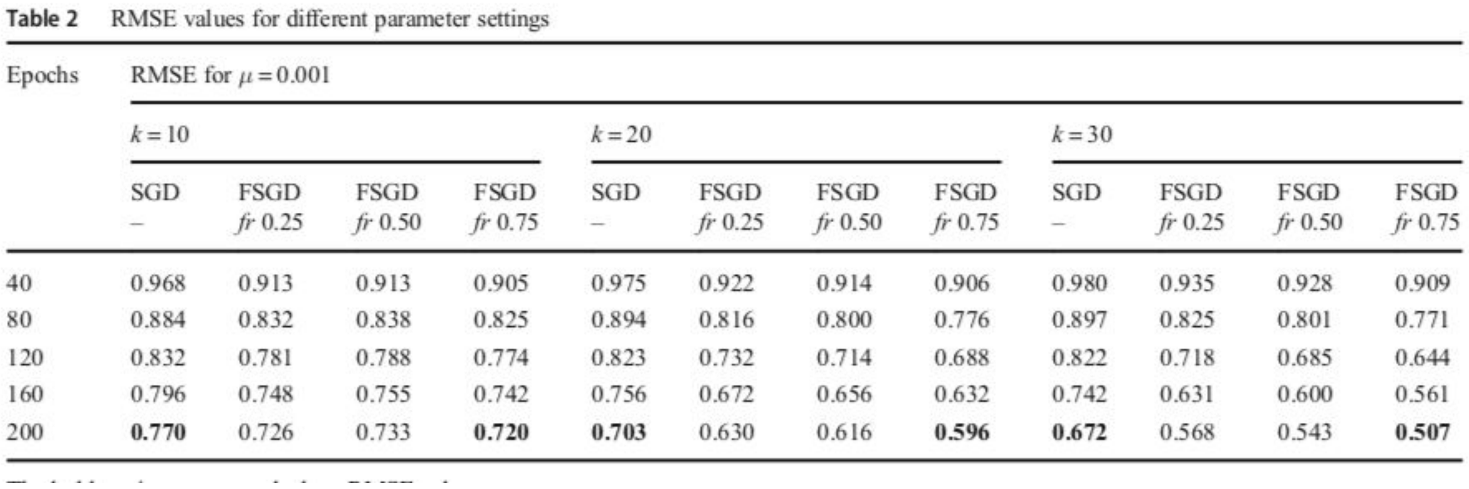


Table 2 RMSE values for different parameter settings

Epochs	RMSE for $\mu = 0.001$											
	$k = 10$				$k = 20$				$k = 30$			
	SGD –	FSGD *fr* 0.25	FSGD *fr* 0.50	FSGD *fr* 0.75	SGD –	FSGD *fr* 0.25	FSGD *fr* 0.50	FSGD *fr* 0.75	SGD –	FSGD *fr* 0.25	FSGD *fr* 0.50	FSGD *fr* 0.75
40	0.968	0.913	0.913	0.905	0.975	0.922	0.914	0.906	0.980	0.935	0.928	0.909
80	0.884	0.832	0.838	0.825	0.894	0.816	0.800	0.776	0.897	0.825	0.801	0.771
120	0.832	0.781	0.788	0.774	0.823	0.732	0.714	0.688	0.822	0.718	0.685	0.644
160	0.796	0.748	0.755	0.742	0.756	0.672	0.656	0.632	0.742	0.631	0.600	0.561
200	**0.770**	0.726	0.733	**0.720**	**0.703**	0.630	0.616	**0.596**	**0.672**	0.568	0.543	**0.507**

Possible Variation

- Instead of using Fractional Stochastic Gradient Descent, we propose using another variation of SGD which is called Elastic Averaging SGD (EASGD) which “links the parameters of the workers of asynchronous SGD with an elastic force. This allows the local variables to fluctuate further from the center variable, which in theory allows for more exploration of the parameter space” (Zhang et al.)
- Research shows that the increased capacity for exploration leads to improved performance by finding new local optima

Algorithm/Code Implementation still under construction….

A Simple Stochastic Gradient Descent Variational Bayes for the Correlated Topic Model

- This applies Stochastic Gradient Variational Bayes to the Correlated Topic Model. The inference also adopts a gradient-based optimization for estimating parameters.
- It uses Stochastic gradient based optimization.
- It doesn’t adopt the whole stochastic gradient descent, instead, it combined stochastic gradient descent with variational bayes and applied it to the Correlated Topic Model.

Preconditioned Stochastic Gradient Descent

- This paper attempts to tackle the issue in stochastic gradient descent of having too much noise by preconditioning the learning algorithm.
- It attempts to find a method that works on both convex and non-convex problems.
- This group tested out three different pre-conditioners, finding out which had the consistently best results in improving convergence, improving long term memory, and eliminating noise.

Code Demo

In machine learning and deep learning fields, the solution to most problems are based on matrix multiplication. If the input matrix has very large features and observation then it would become almost infeasible for computers to load very much data on CPU/GPU to solve such huge matrix multiplications. This is where gradient descent becomes useful.

Our code demo will show how to apply gradient descent algorithms to solve the best parameters of a linear regression model.

Questions?

Lihau Pei, Peiyu Wang, Marie-Laure Brossay
Group 7