Recommender Systems || Lecture 5

Summer 2022

Dr. Karthik Mohan

Univ. of Washington, Seattle

July 19, 2022

Logistics

1. Programming 2 due this Sunday

part 1 → Easy ✓

part 2 → Not Easy

Logistics

1. Programming 2 due this Sunday
2. Please spend good time on the programming assignment

Logistics

1. Programming 2 due this Sunday
2. Please spend good time on the programming assignment
3. Programming 2 part 1 focuses on SVD based recommendations. Programming 2 part 2 is based on SGD (stochastic gradient descent) and matrix factorization.

Logistics

1. Programming 2 due this Sunday
2. Please spend good time on the programming assignment
3. Programming 2 part 1 focuses on SVD based recommendations. Programming 2 part 2 is based on SGD (stochastic gradient descent) and matrix factorization.
4. Any questions on grading? →

Conceptual 20%

Programming 40%

Project 30-40%

Prog 2 has more weightage

Logistics

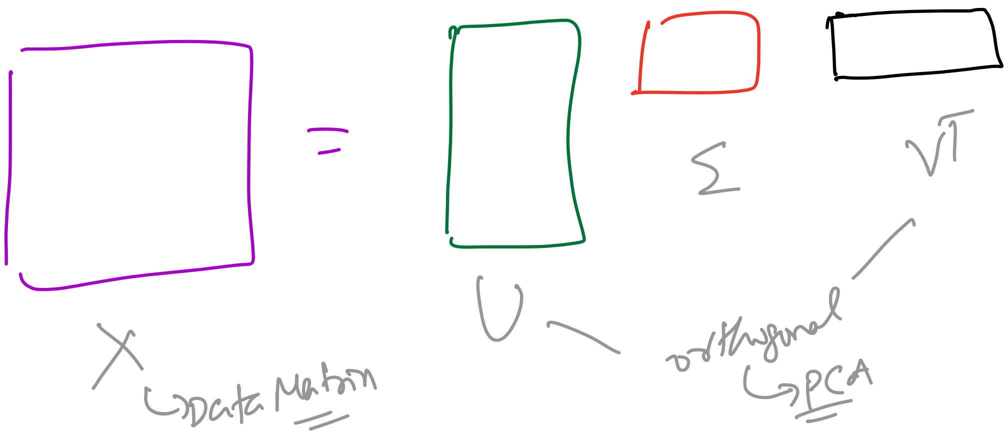
1. Programming 2 due this Sunday
2. Please spend good time on the programming assignment
3. Programming 2 part 1 focuses on SVD based recommendations. Programming 2 part 2 is based on SGD (stochastic gradient descent) and matrix factorization.
4. Any questions on grading?
5. Any other questions?

Limitations of SVD

1. PCA is based on SVD for dimensionality reduction. And we know PCA is not robust to outliers

Limitations of SVD

1. PCA is based on SVD for dimensionality reduction. And we know PCA is not robust to outliers
2. SVD is a nice baseline however it is very specific - Requires factors U, V to be orthogonal and is not flexible to regularization or other considerations.



Limitations of SVD

1. PCA is based on SVD for dimensionality reduction. And we know PCA is not robust to outliers
2. SVD is a nice baseline however it is very specific - Requires factors U, V to be orthogonal and is not flexible to regularization or other considerations.
3. Requires behavioral data - Can't handle hybrid formulations that include content as well

Limitations of SVD

1. PCA is based on SVD for dimensionality reduction. And we know PCA is not robust to outliers
2. SVD is a nice baseline however it is very specific - Requires factors U, V to be orthogonal and is not flexible to regularization or other considerations.
3. Requires behavioral data - Can't handle hybrid formulations that include content as well
4. **Solution and Extension?** Matrix Factorization!

→ Another Limitation :- can't work on an Incomplete matrix!

Matrix Factorization Methods

Matrix Factorization Problem

Let $X \in \mathcal{R}^{m\times n}$ be a data matrix, say for ratings of users vs movies. Rows represent users and columns represent movies, and entries represent the ratings. Then, mathematically, the matrix factorization problem is defined as:

more flexible on U, V than SVD!

$$\min_{U,V} \frac{1}{2}\|X - UV\|_F^2$$

CCA
NMF
LCA
⋮

where, $U \in \mathcal{R}^{m\times r}$ and r is the dimension corresponding to low-dimensional factors (U and V).

Matrix Completion

1M customers & 100K products ≃ 100 Billion possible entries!!

0.01% (In reality)

U V

1M × 1000 1000 × 100K

⟹ O(Number of parameters learned) = ? O(1B)

Matrix Factorization Methods

Matrix Factorization Problem

Let $X \in \mathcal{R}^{m \times n}$ be a data matrix, say for ratings of users vs movies. Rows represent users and columns represent movies, and entries represent the ratings. Then, mathematically, the matrix factorization problem is defined as:

$$\min_{U,V} \frac{1}{2}\|X - UV\|_F^2$$

where, $U \in \mathcal{R}^{m \times r}$ and r is the dimension corresponding to low-dimensional factors (U and V).

Quick question

Based on the set up above, What's the dimension of the matrix V?

Matrix Factorization vs Matrix Completion

Missing ratings

When we have missing ratings in X, instead of filling it in, we can actually only use the ratings we know to learn the factors U and V. This formulation is called the **matrix completion** problem.

Matrix Factorization vs Matrix Completion

→ Incomplete 1 Customer → rate | purchase 50 product (100K) $\frac{50}{100K} < 0.1\%$

Missing ratings

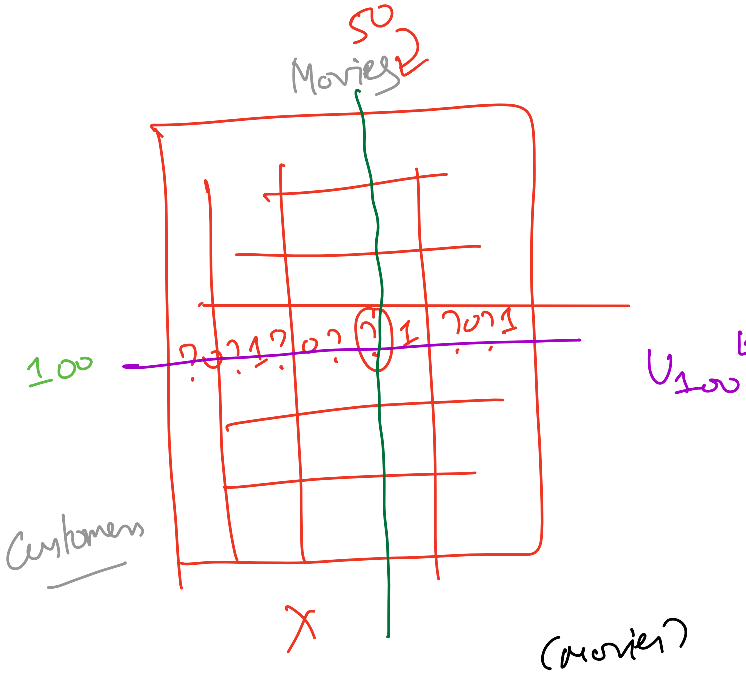
When we have missing ratings in X, instead of filling it in, we can actually only use the ratings we know to learn the factors U and V. This formulation is called the **matrix completion** problem.

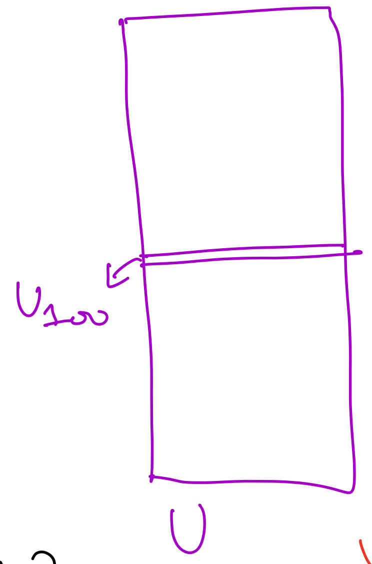
Matrix Completion

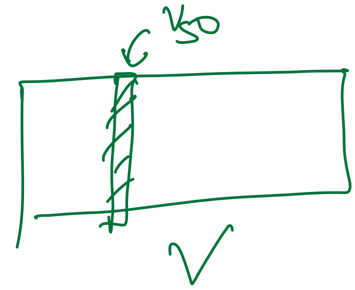
$$\min_{U,V} \frac{1}{2} \sum_{(i,j)\in R} (X_{ij} - U_{i,\cdot}^T V_{\cdot,j})^2,$$

optimization

where R is the set of tuples (i,j) for which a rating is known apriori. How does this compare with matrix factorization with imputation?







$U_{100} \cdot V_{50} = ?$

High ⇒ Customer 100 likes Movie 50

Low ⇒ Customer 100 may not like Movie 50!

(movies)

(Products)

1MM customers, 100k products

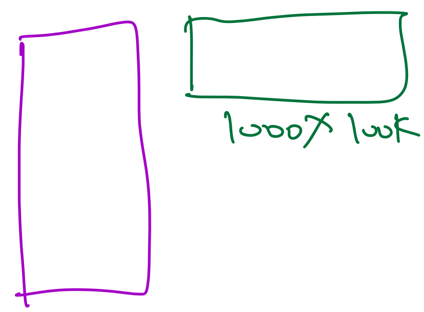
⇓

0.1% of this matrix! } 1 customer purchases 100 products

⇓

100MM training data points

Test data points ≃ O(100B) ~ unknown!!



1MM x 1000

⇓

parameters = O(1B)

O(100MM) ⟶ Train Data points

O(100B) ⟶ Unknown!! ≪

O(1B) ⟶ Parameters

under-fitting / overfitting

Matrix Factorization Extensions

overfitting

use

Flexible Formulation - Can add regularization!

$$\min_{U,V} \frac{1}{2}\|X - UV\|_F^2 + \lambda\|U\|_F^2 + \lambda\|V\|_F^2$$

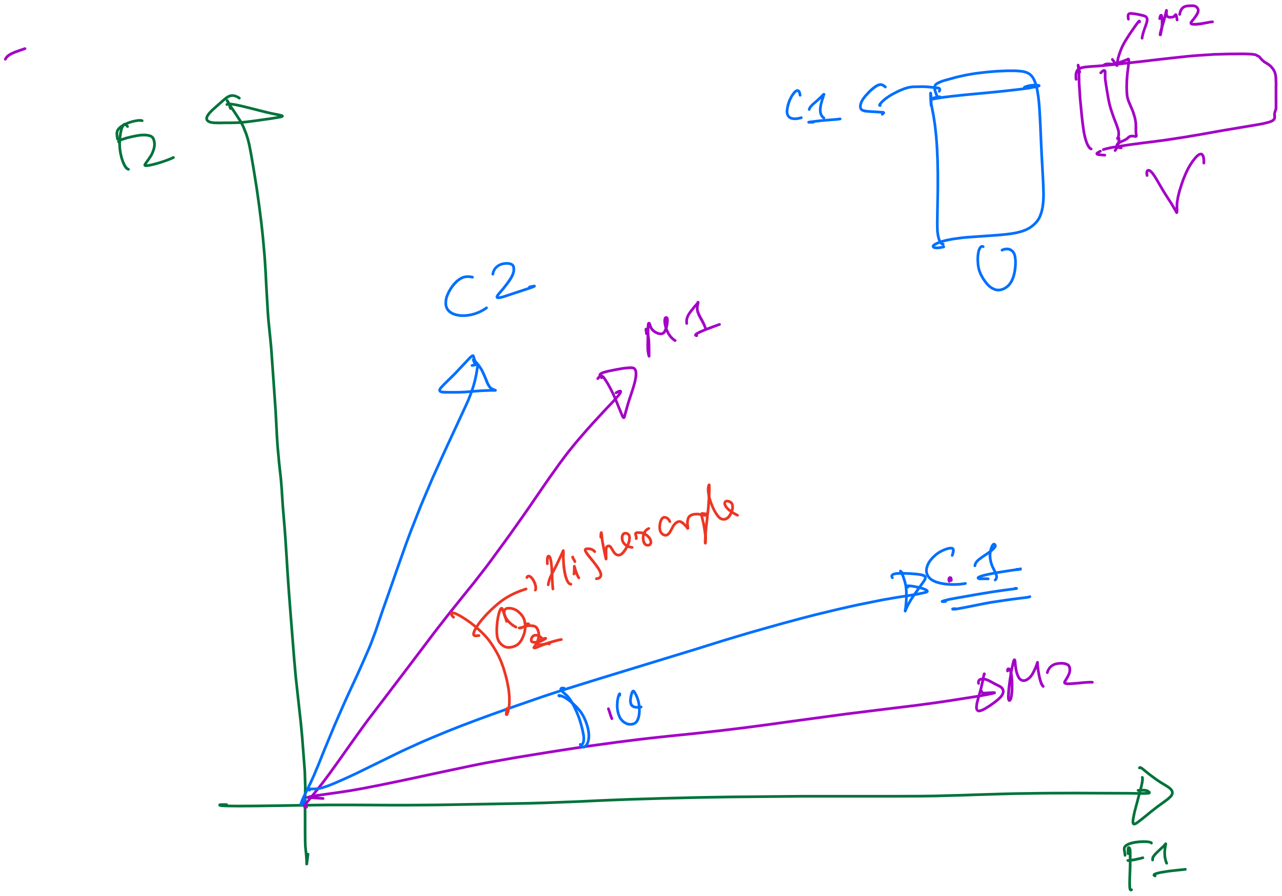
Minimizing overfitting

λ - Hyper-Parameter

$\lambda = 0 \Rightarrow$ Back to matrix completion!

$\lambda \uparrow \Rightarrow$ Reduce overfitting further

$\lambda \rightarrow \infty \Rightarrow U, V \Rightarrow 0$


F2
C2
M1
Highercangle
C1
M2
F1
C1
M2
U
V

Matrix Factorization Extensions

Flexible Formulation - Can add regularization!

$$\min_{U,V} \frac{1}{2}\|X - UV\|_F^2 + \lambda\|U\|_F^2 + \lambda\|V\|_F^2$$

Reduced rank matrix factorization

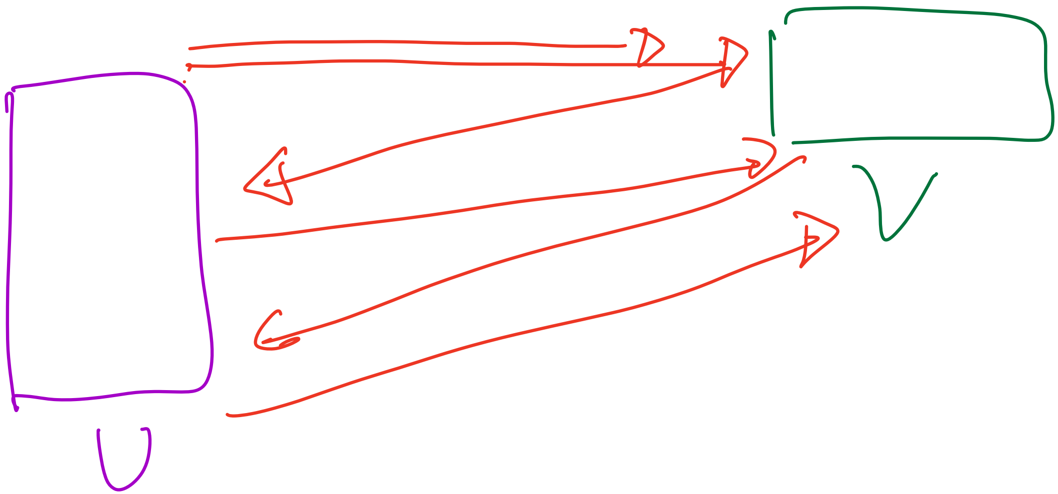
Above formulation with the regularization is known to reduce the dimensionality of the factors obtained.

Alternating Minimization Method for Matrix Completion

Skip to next lecture

$$\min_{V} \frac{1}{2} \|X - UV\|_F^2 + \dots$$

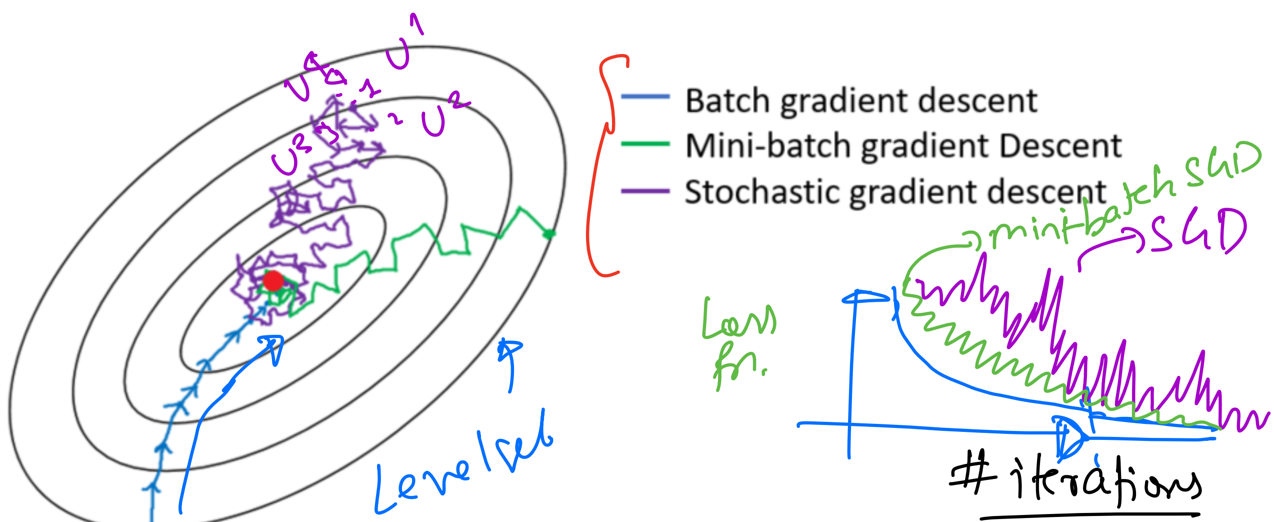
Two sets of parameters:- U, V



Algorithmic foundations to Machine Learning

Underlying Engine behind ML Training

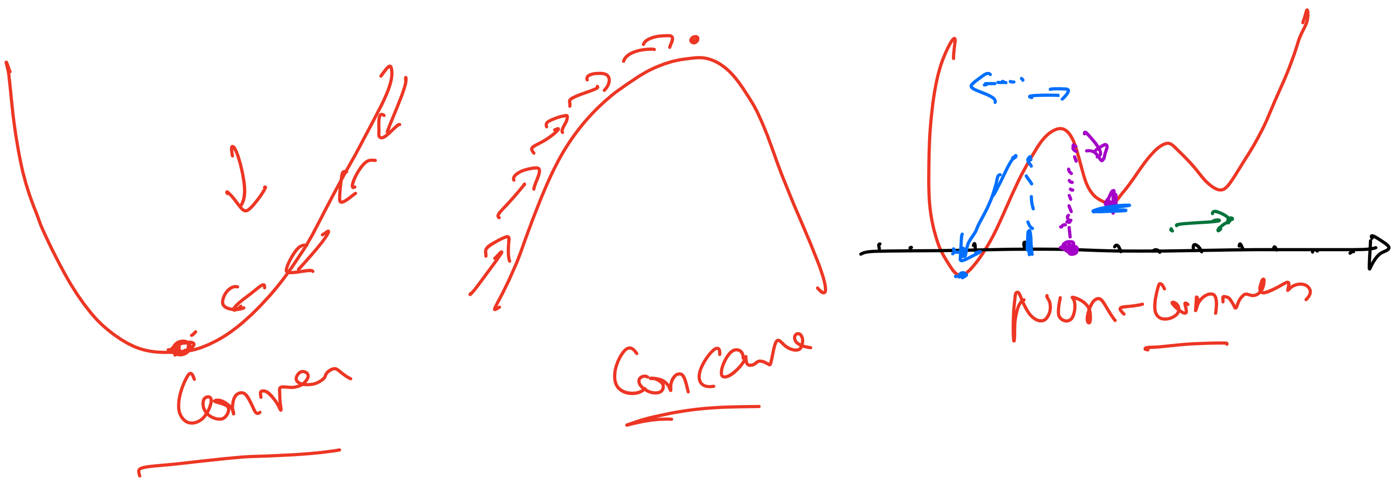
`(Mini-batch) Stochastic Gradient Descent` Almost every model and problem-space in ML uses SGD of some kind - Clustering, Regression, Deep Learning, Computer Vision and NLP to name a few. Almost every algorithm in every library - `Scikit-learn`, `Keras`, `Pytorch`, etc uses **mini-batch SGD under the hood**.



So what is Gradient Descent?

Fundamentally

Take a convex/non-convex function, f. GD allows you to find a local optimum to f.



So what is Gradient Descent?

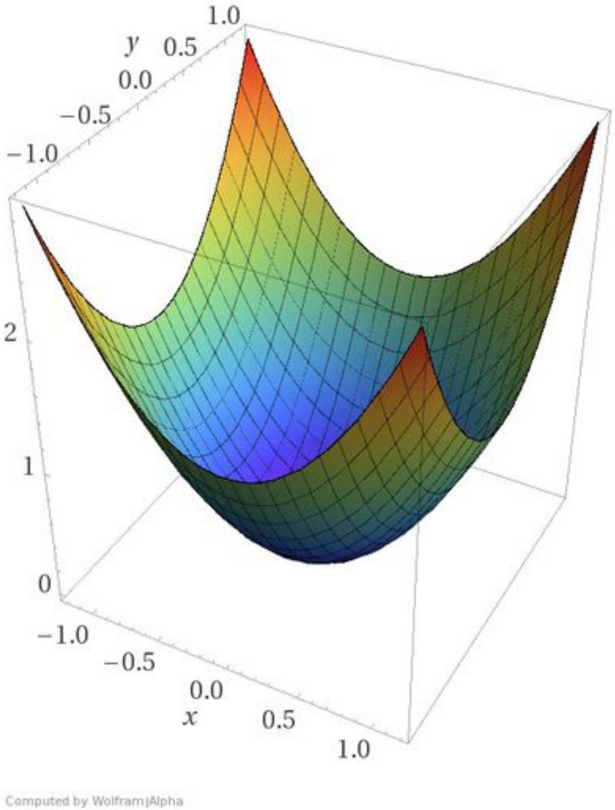
Fundamentally

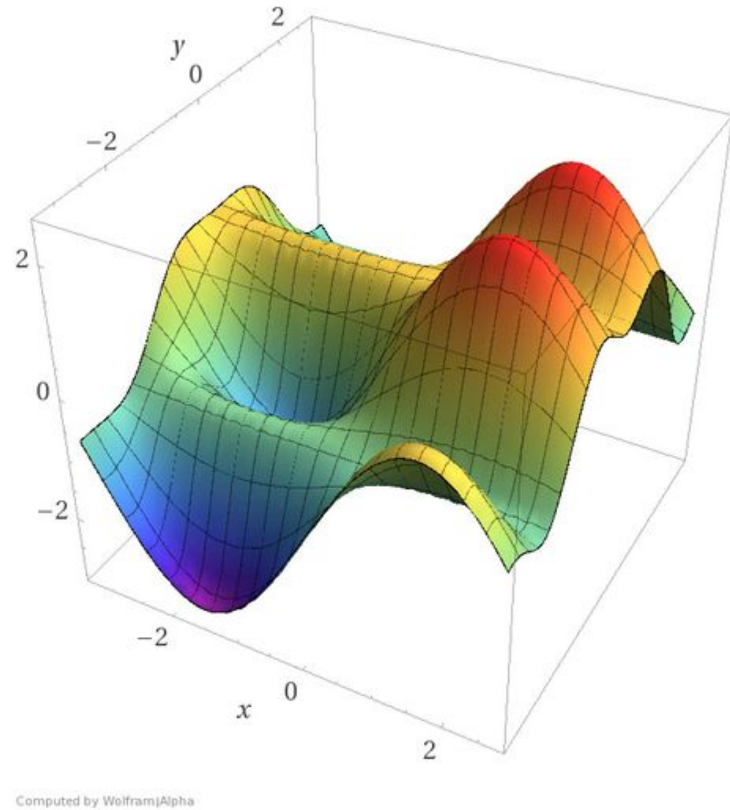
Take a convex/non-convex function, f. GD allows you to find a local optimum to f.

Why is this important?

Consider the Linear Regression problem. $\hat{w}$ is a local optimum to the function $f(w) = \frac{1}{2}\|Xw - y\|_2^2 + \lambda\|w\|_2^2$

Negative Gradient helps you view the direction of descent





$\underline{\underline{u}}^3 \leftarrow \underline{\underline{u}}^2 + \delta \left(- \nabla \ell \left(u^2, \underline{\underline{v}}^2 \right) \right)$

Gradient Descent

Batch Gradient Descent

Let us say we want to minimize $L(\hat{w})$ - Loss Function and find the best $\hat{w}$ that does that.

1. **Initialize** $w = w_0$ (maybe randomize)

Gradient Descent

Batch Gradient Descent

Let us say we want to minimize $L(w)$ - Loss Function and find the best $\hat{w}$ that does that.

1. **Initialize** $w = w_0$ (maybe randomize)
2. **Gradient Descent** $w \leftarrow w - lr * \nabla L(w)$

step size = Learning Rate

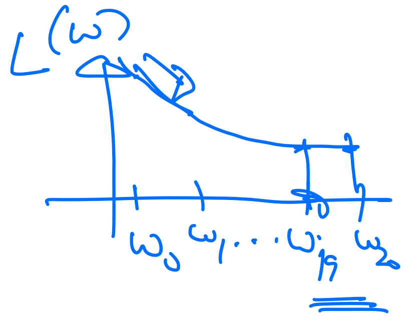
Gradient Descent

Batch Gradient Descent

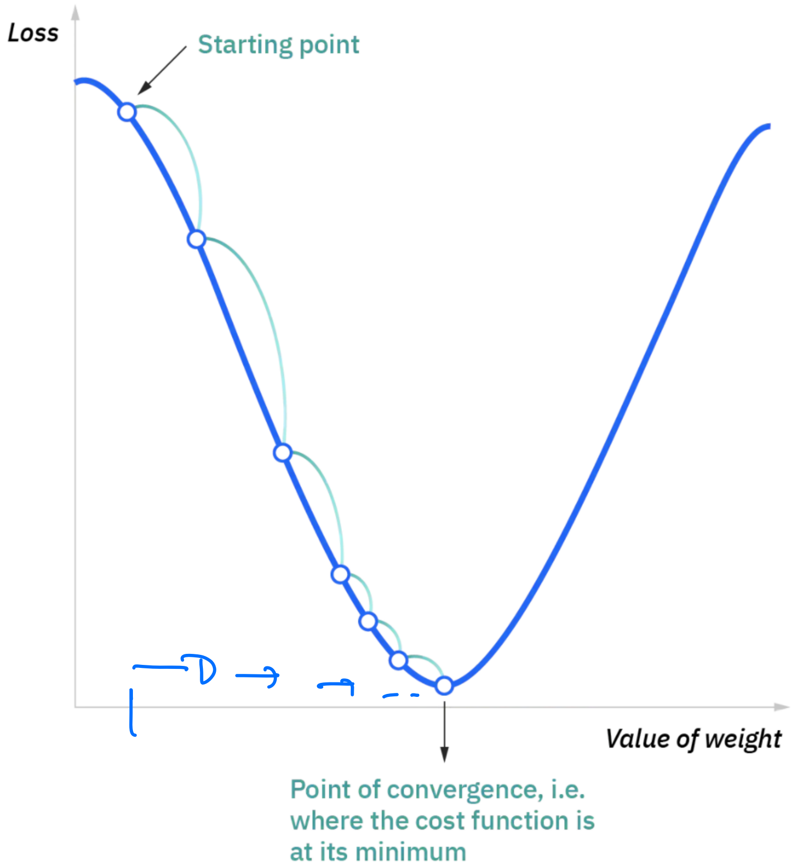
Let us say we want to minimize $L(w)$ - Loss Function and find the best $\hat{w}$ that does that.

1. **Initialize** $w = w_0$ (maybe randomize)
2. **Gradient Descent** $w \leftarrow w - lr * \nabla L(w)$
3. **Iterate** Repeat step 2 until w converges, i.e.

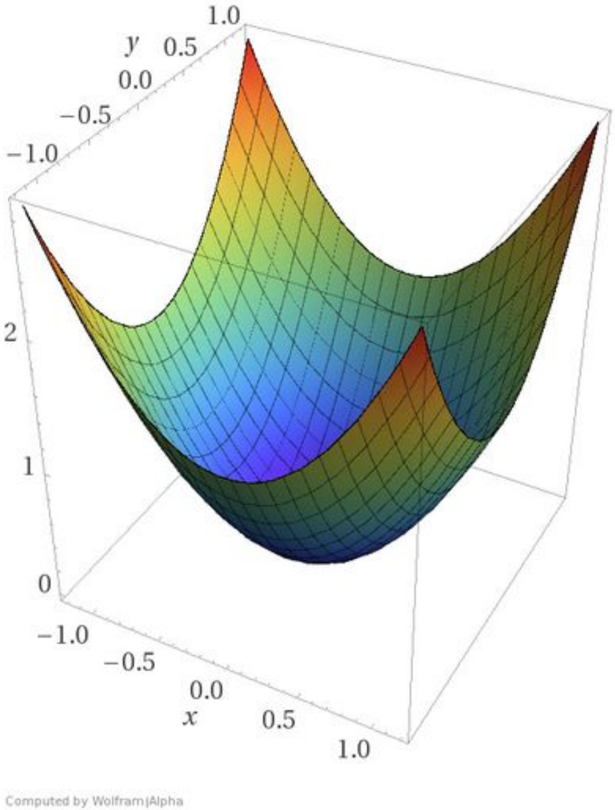
$$\|w^{k+1} - w^k\| / \|w^k\| \leq 10^{-3}$$

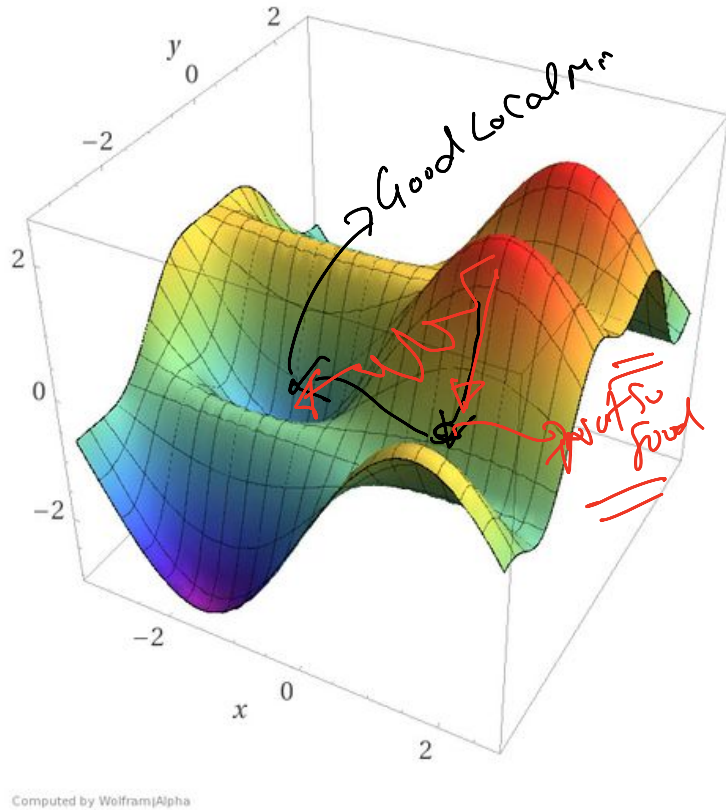


GD in one dimension



Loss function in 2 dimensions





Deep Learning and SGD!

Gradient in DL ← "Back propagation"

↓

Auto-Differentiation

Computing Gradients!

$$\ell(u) = u^T \alpha$$

$\hookrightarrow$ content vector

$\left[\begin{array}{l}\alpha \in \mathbb{R}^{10} \\ u \in \mathbb{R}^{10}\end{array}\right.$

$$\underline{\nabla_u \ell = \alpha}$$

$$\ell(u) = \underbrace{u_1 \alpha_1} + u_2 \alpha_2 + \cdots + u_{10} \alpha_{10}$$

$$\nabla_{u_1} \ell = \frac{d}{du_1}(u_1 \alpha_1) = \alpha_1$$

$$\nabla_{u_K} \ell = \alpha_K$$

$$\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_{10} \end{bmatrix} = \alpha$$

ICE #1

$\|w\|_2^2 = w_1^2 + w_2^2 + \cdots + w_d^2$

$w \in \mathbb{R}^d$

Gradient of Ridge Regularizer (2 mins)

Find the gradient of the regularization function, $R(w) = \lambda\|w\|_2^2$. I.e. obtain the expression for, $\nabla_w R(w)$?

- a) $2\lambda\|w\|_2$
- b) $\lambda\|w\|_2 w$
- c) $2\lambda w$
- d) $2\lambda\|w\|_2 w$

$$\begin{bmatrix} \frac{\partial}{\partial w_1} R \\ \frac{\partial}{\partial w_2} R \\ \vdots \\ \frac{\partial}{\partial w_d} R \end{bmatrix} = \nabla_w R$$

Poll Link

$$\frac{\partial}{\partial w_1} R = \frac{\partial}{\partial w_1} \lambda(\underline{\underline{w_1}}^2 + w_2^2 + \dots + w_d^2)$$

$$\Rightarrow \lambda(2w_1) = 2\lambda w_1$$

$$\frac{\partial}{\partial w_2} R_L = 2\lambda w_2$$

Entry/Scalar

$$2\lambda \underbrace{\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix}}_{\mathbf{w}} = 2\lambda \underline{\underline{\mathbf{w}}}$$

vector

Gradient Descent Properties

1. Gradient Descent converges to a local minimum

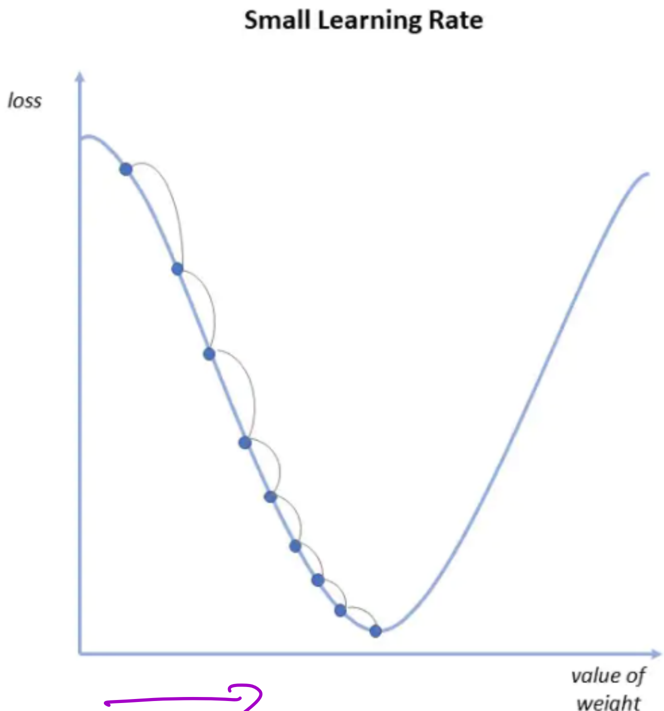
Gradient Descent Properties

1. Gradient Descent converges to a local minimum
2. If L is a convex function, all local minima become a global minima!

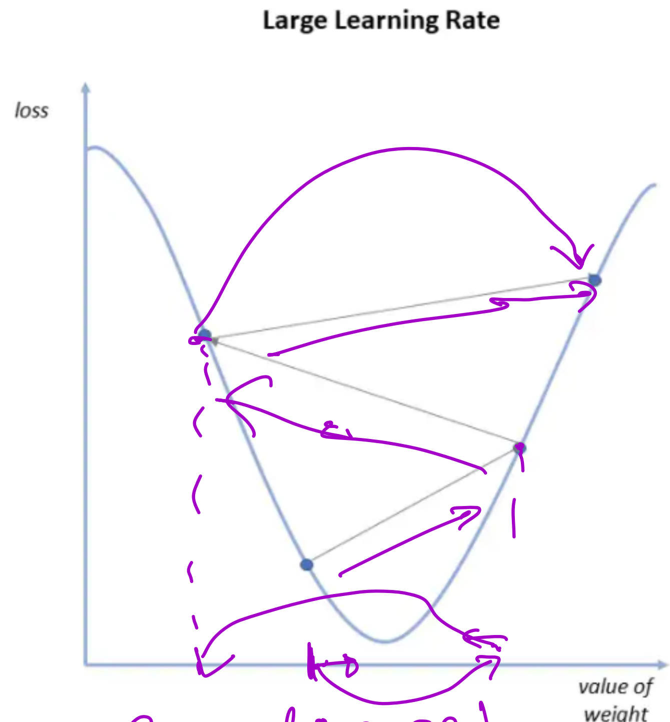
Gradient Descent Properties

1. Gradient Descent converges to a local minimum
2. If L is a convex function, all local minima become a global minima!
3. Wherever we start, gradient descent usually finds a local minima closest to the start.

Effect of Learning Rate

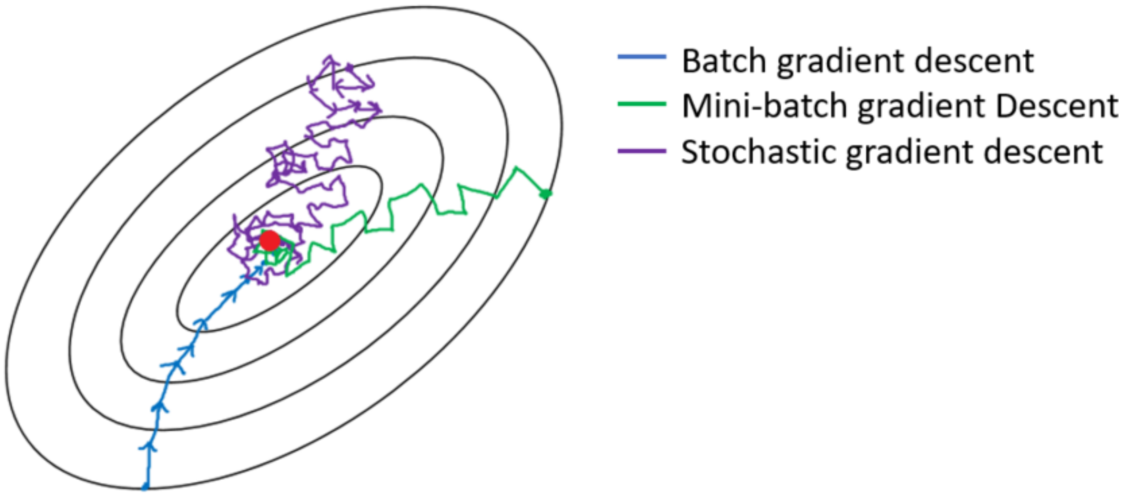


More time
to Converge



Can diverge!

GD behavior in the search space



Gradient descent in practice - SGD!

data pts.

SGD

Let $L(w) = \sum_{i=1}^{N} L_i(w)$ where L_i is a function of only the *ith* data point (x_i, y_i) and parameter w.

1. **Initialize** w^0 (randomize)

$(U^0, V^0) \rightarrow$ matrix completion

Gradient descent in practice - SGD!

SGD

Let $L(w) = \sum_{i=1}^{N} L_i(w)$ where L_i is a function of only the *ith* data point (x_i, y_i) and parameter w.

1. **Initialize** w^0 (randomize) Pick index i at random between 1 and N!

Gradient descent in practice - SGD!

SGD

Let $L(w) = \sum_{i=1}^{N} L_i(w)$ where L_i is a function of only the *ith* data point (x_i, y_i) and parameter w.

1. **Initialize** w^0 (randomize) Pick index i at random between 1 and N!
2. **Gradient Descent** $w^{k+1} \leftarrow w - lr * \nabla L_i(w^k)$

Gradient descent in practice - SGD!

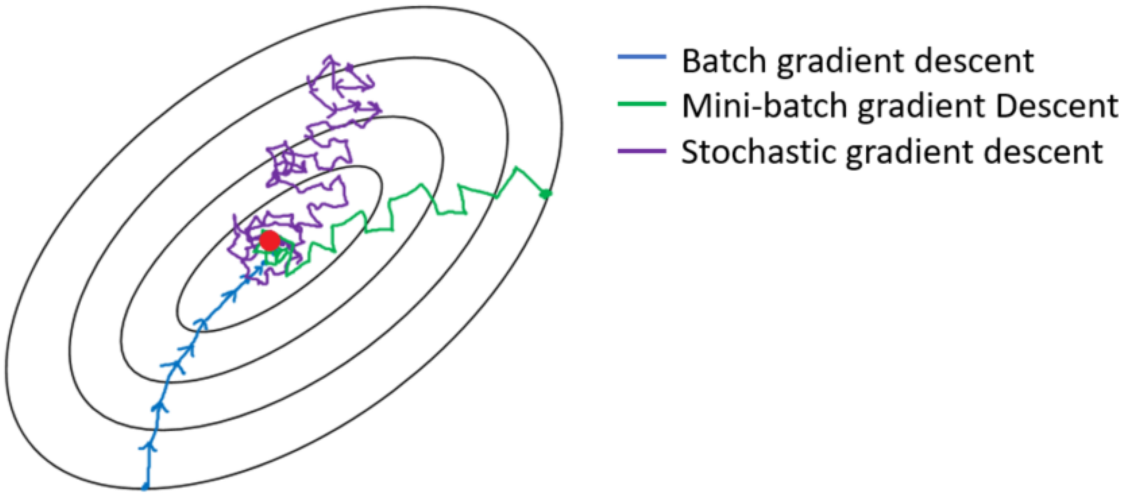
SGD

Let $L(w) = \sum_{i=1}^{N} L_i(w)$ where L_i is a function of only the *ith* data point (x_i, y_i) and parameter w.

1. **Initialize** w^0 (randomize) Pick index i at random between 1 and N!
2. **Gradient Descent** $w^{k+1} \leftarrow w - lr * \nabla L_i(w^k)$
3. **Iterate** Repeat step 2 and 3 until w converges, i.e.

$$\|w^{k+1} - w^k\| / \|w^k\| \leq 10^{-3}$$

SGD behavior in search space



SGD in practice - mini-batch SGD!

mini-batch SGD

Let $L(w) = \sum_{i=1}^{N} L_i(w)$ where L_i is a function of only the *ith* data point (x_i, y_i) and parameter w. Let B be the number of batches and k be the batch size.

$= N/k$

$32, 256$

1. **Initialize** $w = w_0$ (randomize)

SGD in practice - mini-batch SGD!

mini-batch SGD

Let $L(w) = \sum_{i=1}^{N} L_i(w)$ where L_i is a function of only the *ith* data point (x_i, y_i) and parameter w. Let B be the number of batches and k be the batch size.

256

1. **Initialize** $w = w_0$ (randomize) Pick a batch of k data points at random between 1 and N: $i_1, i_2, \ldots, i_k$!

SGD in practice - mini-batch SGD!

mini-batch SGD

Let $L(w) = \frac{1}{N}\sum_{i=1}^{N} L_i(w)$ where L_i is a function of only the *ith* data point (x_i, y_i) and parameter w. Let B be the number of batches and k be the batch size.

1. **Initialize** $w = w_0$ (randomize) Pick a batch of k data points at random between 1 and N: $i_1, i_2, \ldots, i_k$!
2. **Gradient Descent** $w^{k+1} \leftarrow w^k - lr * \underbrace{\sum_{j=1}^{k} \nabla_w L_{i_j}(w^k)}_{\text{pool info from k data pts.}}$

(SGD: $k = 1$)

SGD in practice - mini-batch SGD!

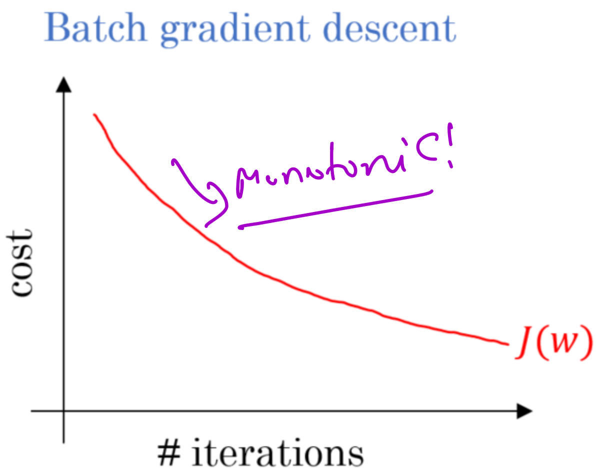
mini-batch SGD

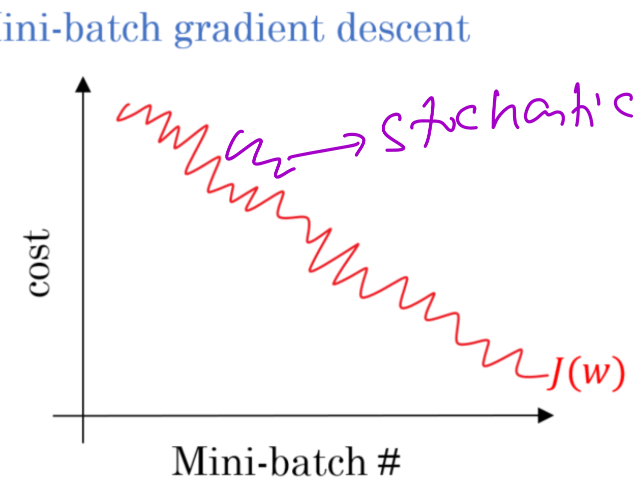
Let $L(w) = \sum_{i=1}^{N} L_i(w)$ where L_i is a function of only the *ith* data point (x_i, y_i) and parameter w. Let B be the number of batches and k be the batch size.

1. **Initialize** $w = w_0$ (randomize) Pick a batch of k data points at random between 1 and N: $i_1, i_2, \ldots, i_k$!
2. **Gradient Descent** $w^{k+1} \leftarrow w^k - lr * \sum_{j=1}^{k} \nabla_w L_{i_j}(w^k)$
3. **Iterate** Repeat step 2 and 3 until w converges, i.e.

$$\|w^{k+1} - w^k\| / \|w^k\| \leq 10^{-3}$$

GD vs Mini-batch convergence behavior





GD vs mini-batch SGD

Factor	GD	Mini-batch SGD
Data	All per iteration	Mini-batch (usually 128 or 256)
Randomness	Deterministic	Stochastic
Error reduction	Monotonic	Stochastic
Computation	High	Low
Memory big data	Intractable	Tractable
Convergence	Low relative error	Few "passes" on data ?
Local Minima traps	Yes	No

Stopping criteria

DL

Programming Assignment 2 Part 2

1. Implement SGD yourself, from scratch for recommender systems!

Programming Assignment 2 Part 2

1. Implement SGD yourself, from scratch for recommender systems!
2. Apply it with/without regularizers!

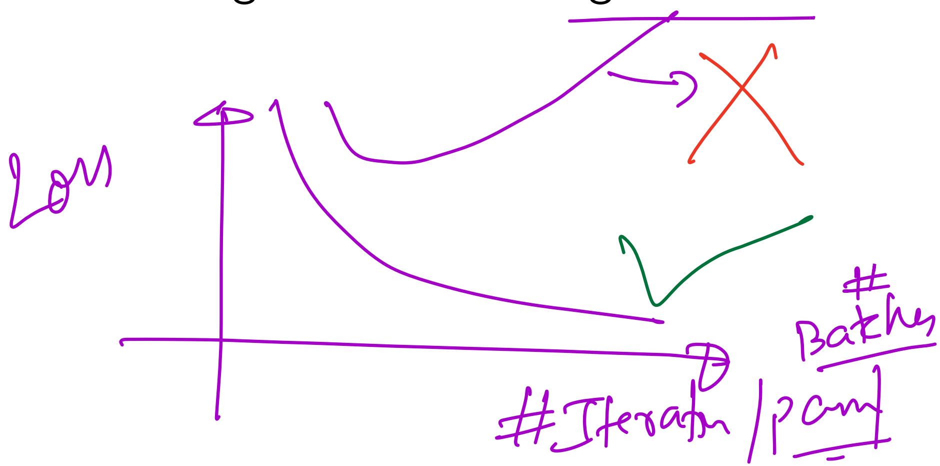
Programming Assignment 2 Part 2

MovieLens Data set 100k ratings

1. Implement SGD yourself, from scratch for recommender systems!
2. Apply it with/without regularizers!
3. Use "auto-differentiation" part of MXNet/Pytorch DL frameworks to compute gradients automatically!

Programming Assignment 2 Part 2

1. Implement SGD yourself, from scratch for recommender systems!
2. Apply it with/without regularizers!
3. Use "auto-differentiation" part of MXNet/Pytorch DL frameworks to compute gradients automatically!
4. Test that you have implemented it right as the training error will make it obvious to you!



Programming Assignment 2 Part 2

1. Implement SGD yourself, from scratch for recommender systems!
2. Apply it with/without regularizers!
3. Use "auto-differentiation" part of MXNet/Pytorch DL frameworks to compute gradients automatically!
4. Test that you have implemented it right as the training error will make it obvious to you!
5. In practice, SGD isn't implemented - But this is the fundamental building block behind any ML/DL algorithm/framework such as Scikit learn, Pytorch, Keras, etc.