Class 5- Machine Learning concepts Part II

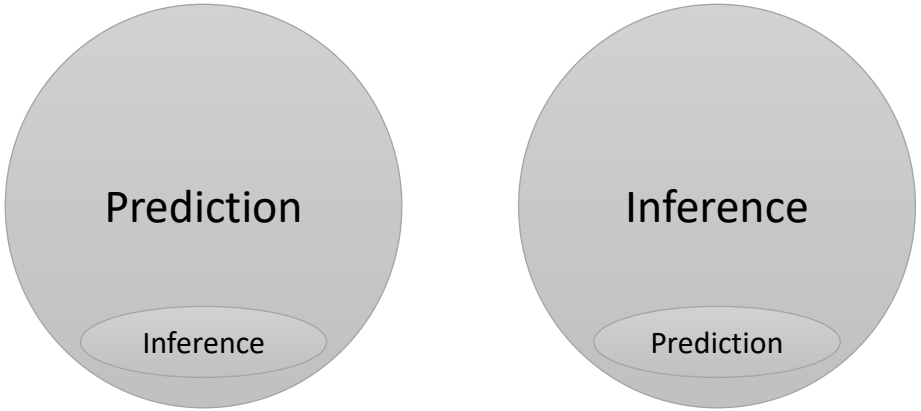
Agenda

- Fetch repo with updated slides!
- Slides
- HW discussion! EDA NYC!

Motivation

Machine learning fundamental concepts:

- Inference and prediction
- Part I: The Model
- Part II: Evaluation metrics
- Part III: Bias-Variance tradeoff
- Part IV: Resampling methods
- Part V: Solvers/learners (GD, SGD, Adagrad, Adam, …)
- Part VI: How do machines learn?
- Part VII: Scaling the features

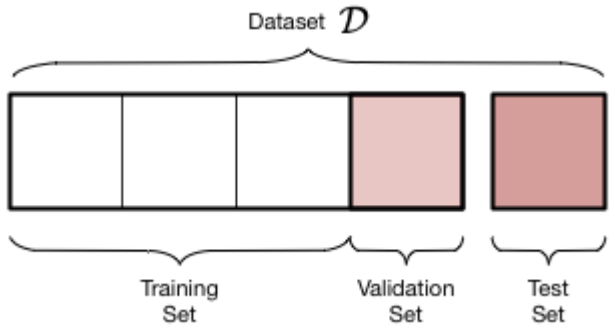


Part IV
Resampling methods

Partitioning of the dataset

The data set is typically divided into three non-overlapping samples:

1) Training set used to train the model
2) Validation set for validating and tuning the model
3) Test set (holdout set) for testing the model's ability to predict well on new data



To be valid and useful, any supervised machine learning model must generalize well beyond the training data.

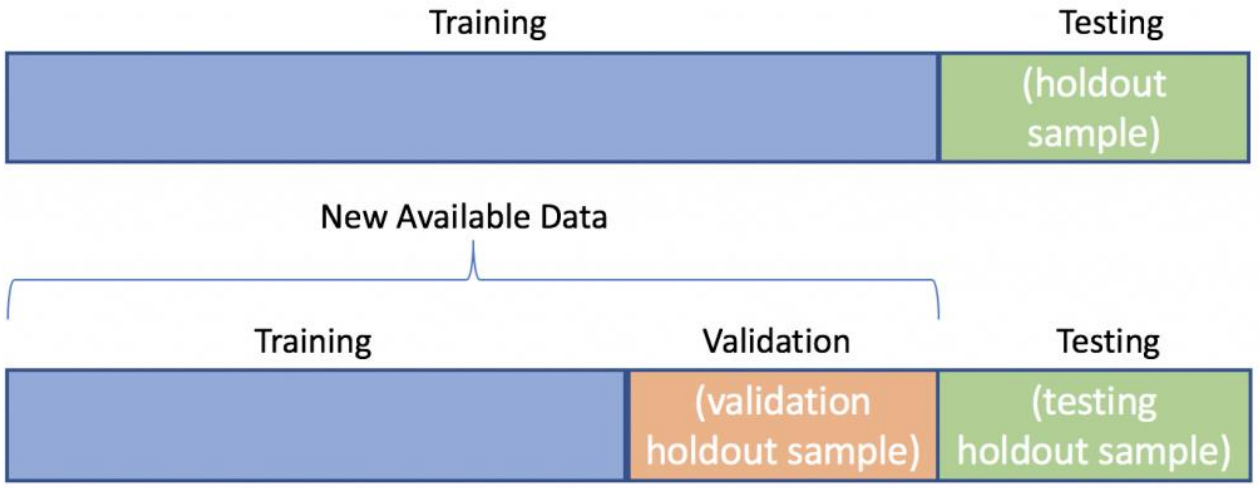
Large dataset is needed! But what if we don't have it?

Resampling methods

Cross validation

- Sometimes we cannot afford to split the data in three because the algorithm may not learn anything from a small training dataset!
- Small validation set is also problematic because we cannot tune the hyperparameters properly!

- Solution: combining the training and validation sets!
- The goal is to obtain additional information about the fitted model! For example, to provide estimates of test set prediction errors.



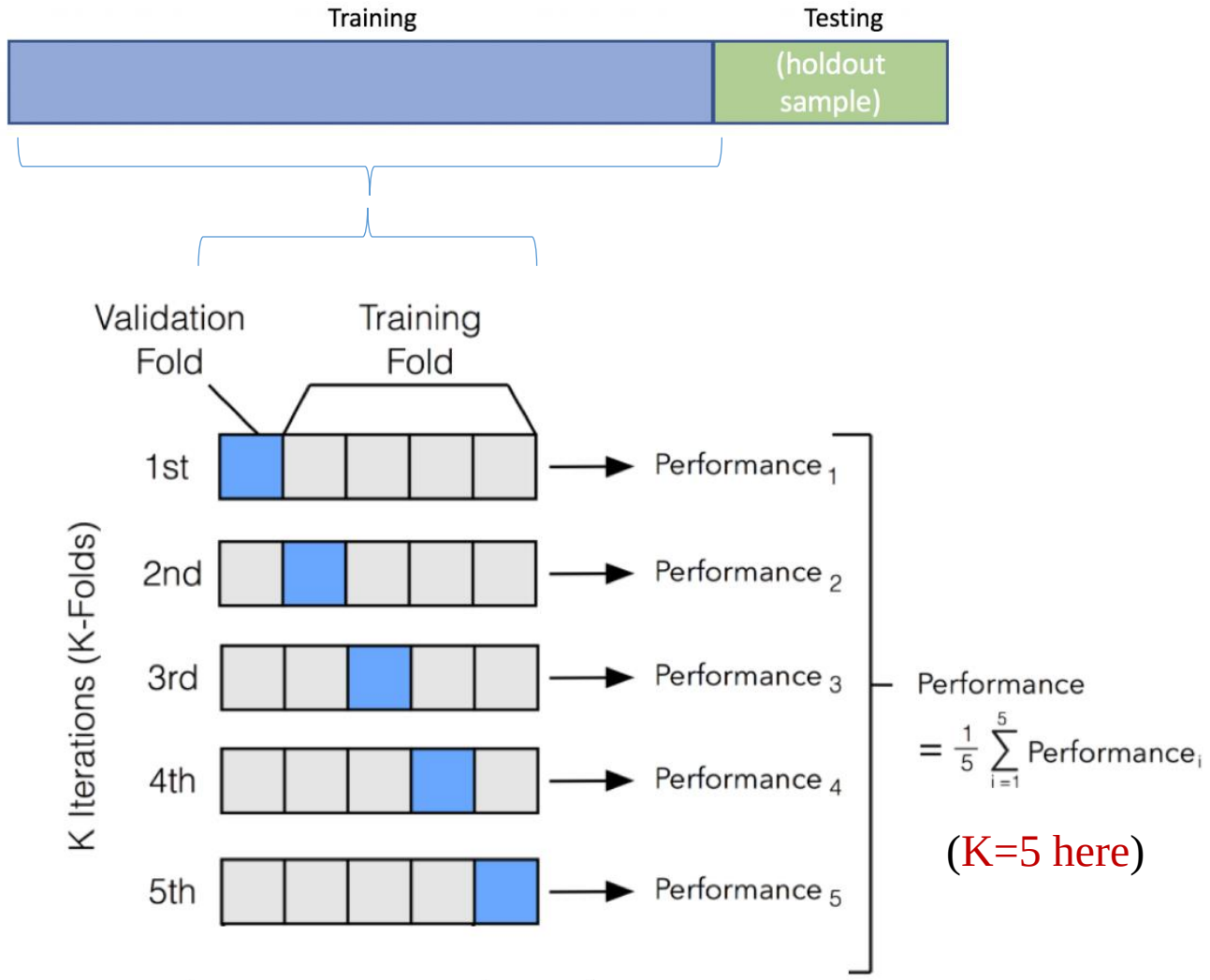
K-fold Cross Validation

1) Divide the training data into K roughly equal-sized non-overlapping groups. Leave out k^{th} fold and fit the model to the other $k-1$ folds. Finally, obtain predictions for the left-out k^{th} fold.

2) Performance can be any of the evaluation metrics for regression or classification models. For example, MSE, accuracy, …

3) This is done in turn for each part $k = 1,2, \ldots . K$, and then the results are combined.

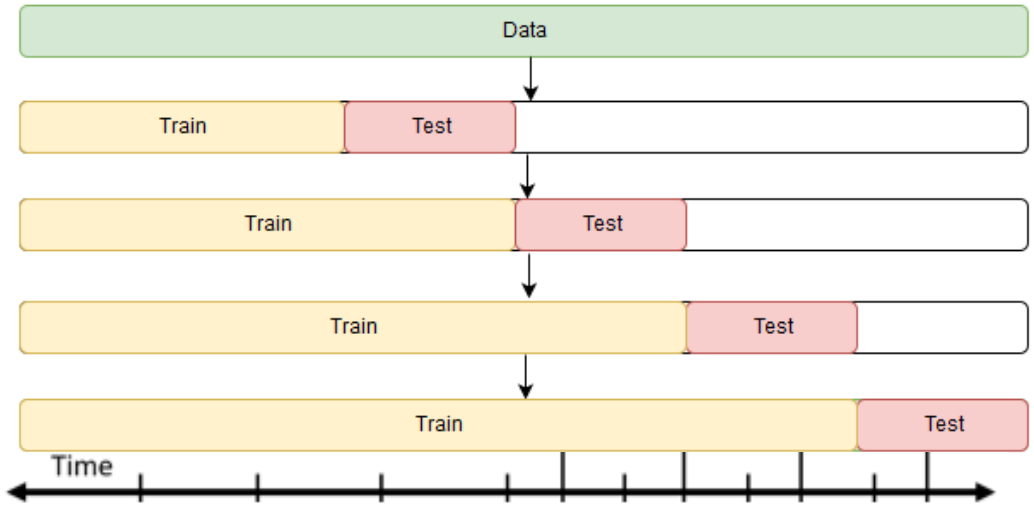
- Leave one out CV (LOOCV): if there is only 1 observation in each fold.



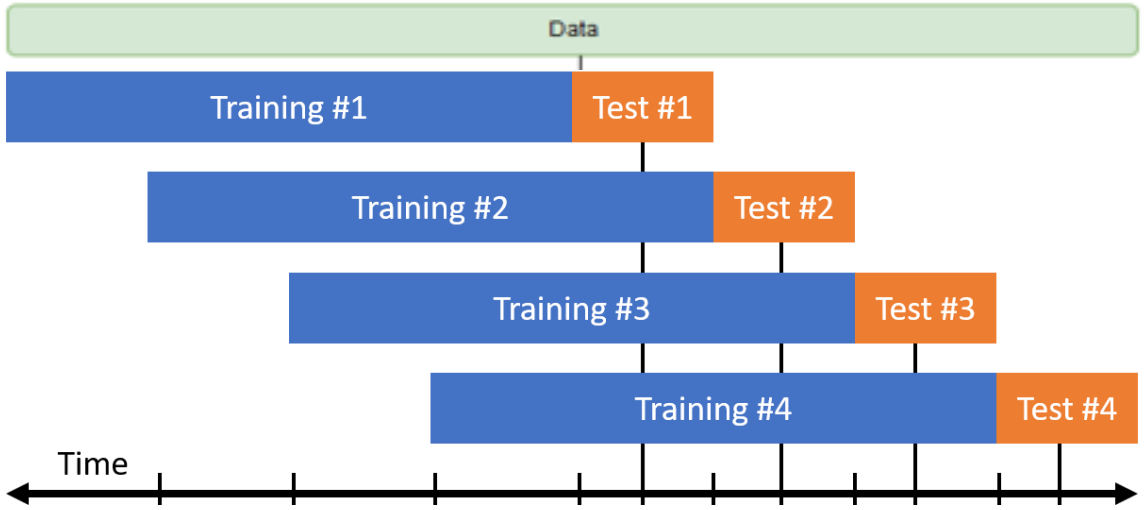
Time Series Cross Validation

With time series data, we cannot shuffle the data! We also need to avoid look-ahead bias!

Walk forward cross validation
Expanding windows



Walk forward cross validation
Rolling windows



Why do we use Cross Validation?

Cross validation (aka validation set, development set or dev set) is mainly used for two purposes:

1- **Model selection**

2- Fair estimate of the performance of the model in the test set

After selecting the best model, we estimate the generalization error using the test set.

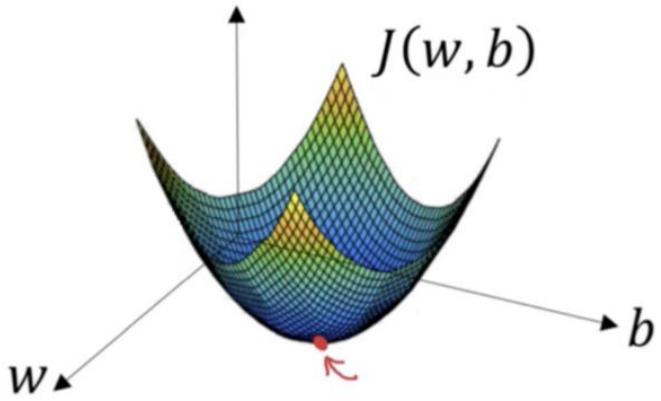
Part V
Solvers (Gradient Descent)

Cost Function

A Cost Function tells us "how good" our model is at making predictions for a given set of parameters. The cost function has its own curve and its own gradients. The slope of this curve tells us how to update our parameters to make the model more accurate.

Terminology alert: Loss function applies to a single training example; vs cost function is the average of all the loss function values!!!

Example: MSE



The two most frequently used optimization algorithms when the cost function is differentiable are:

1) Gradient Descent (GD)
2) Stochastic Gradient Descent (SGD)

Solvers (learners)!

Gradient Descent: is an iterative optimization algorithm for finding the minimum of a function. To find a local minimum of a function using gradient descent, one starts at some random point and takes steps proportional to the negative of the gradient of the function at the current point.

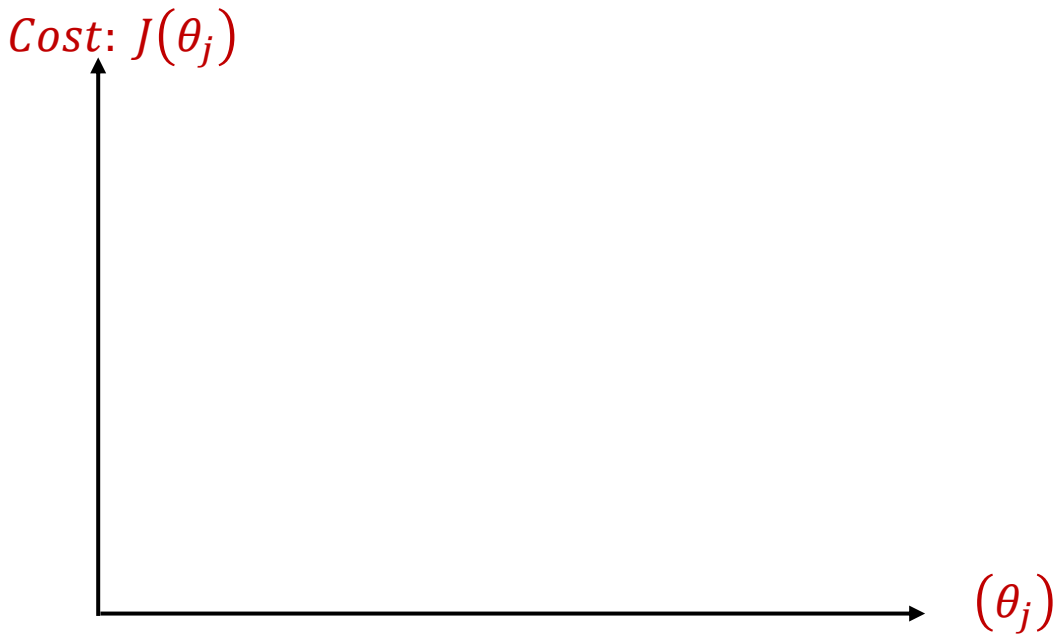
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

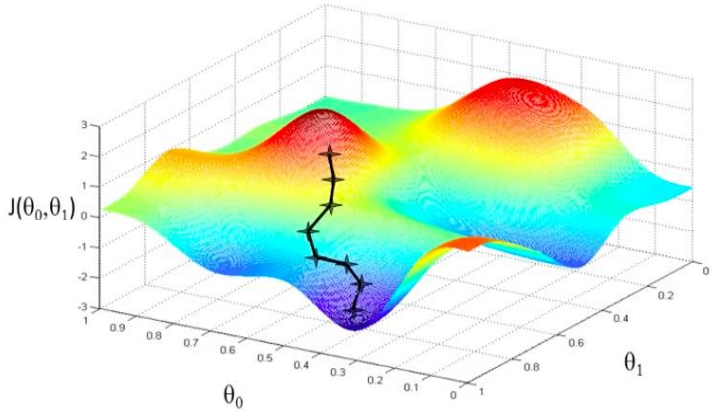
- θ_j is the model's j^{th} parameter
- α is the learning rate
- $J(\theta)$ is the cost function (which is differentiable)

Gradient Descent Visualization

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

Gradient descent proceeds in epochs. An epoch consists of using the training set entirely to update each parameter. The learning rate α controls the size of an update





repeat until convergence {

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

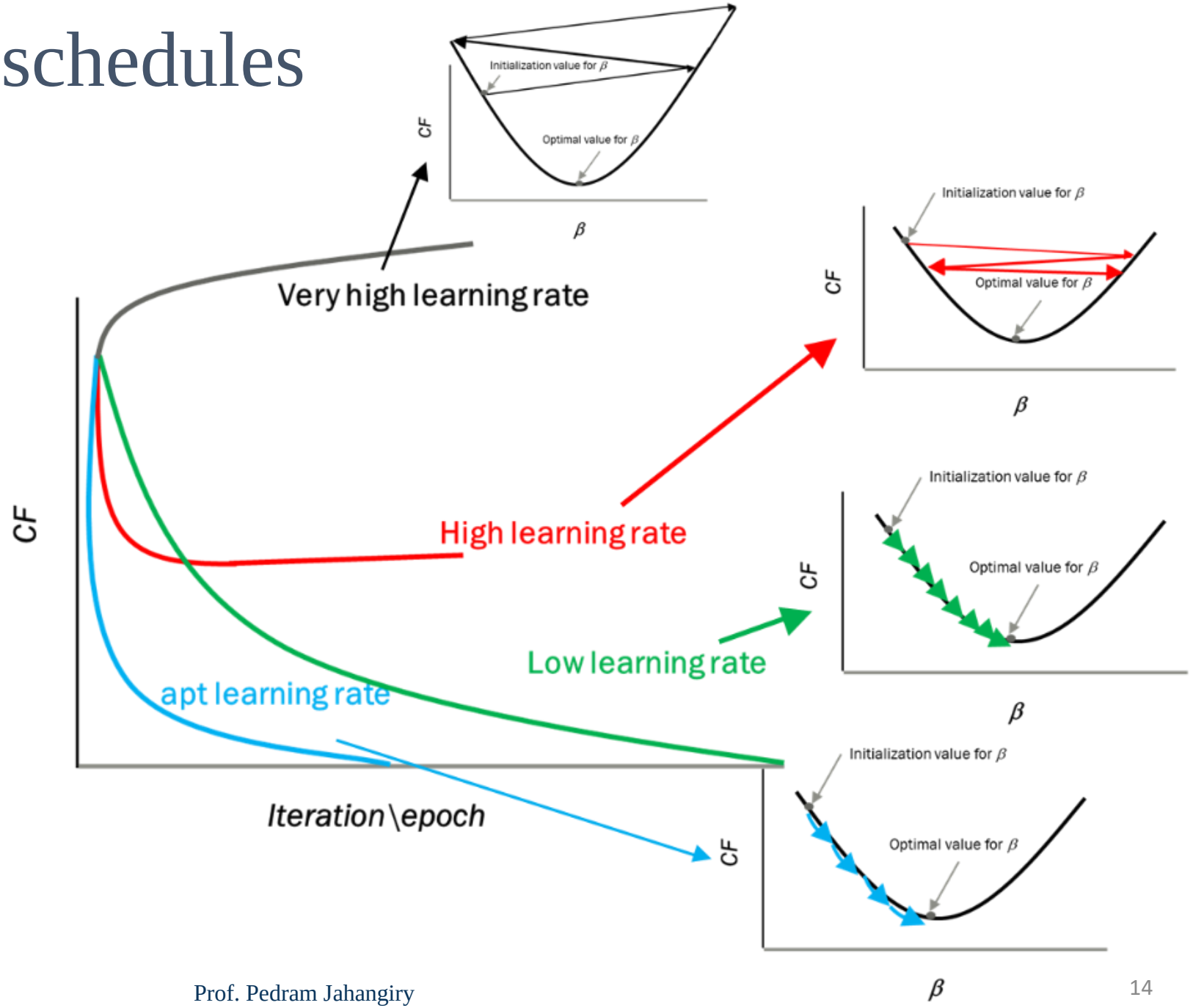
(for $j = 1$ and $j = 0$)

}

Learning rate schedules

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

- If α is too small, gradient descent can be slow
- If α is large, gradient descent can overshoot the minimum. It may fail to converge.
- If α is too large, the gradient descent can even **diverge**.



Beyond Gradient Descent?

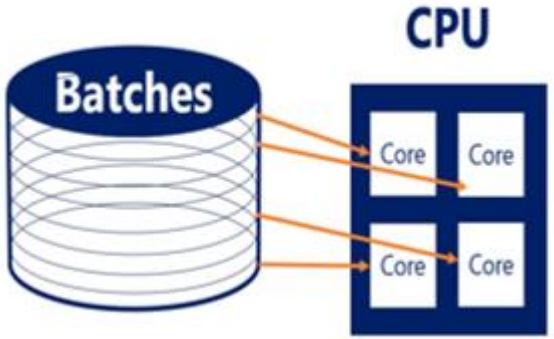
Disadvantages of gradient descent:

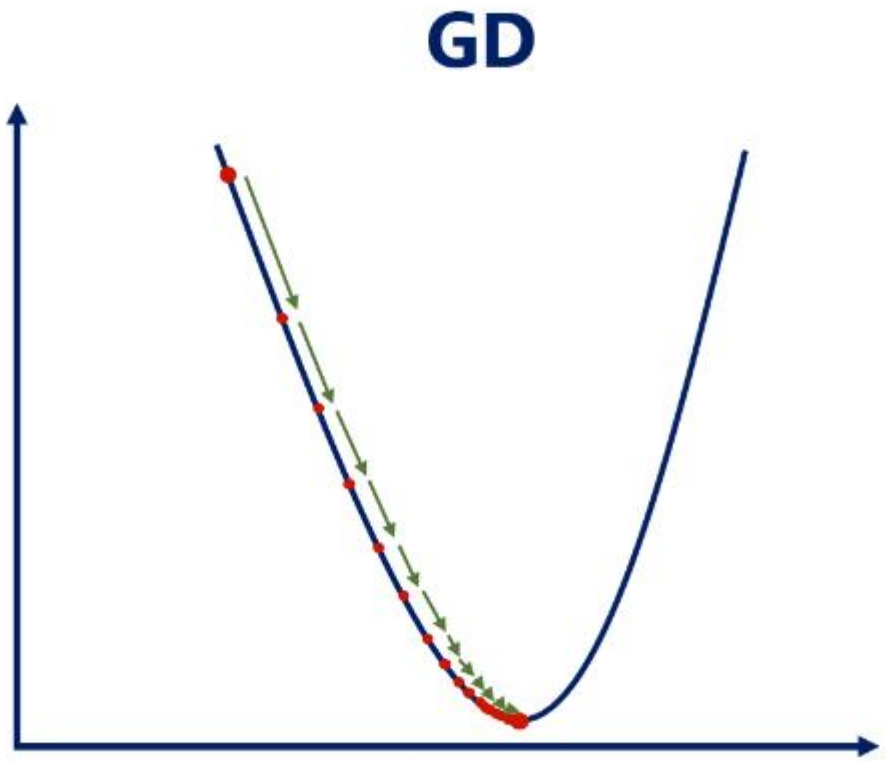
- Single batch: use the entire training set to update parameters!
- Sensitive to the choice of the learning rate
- Slow for large datasets

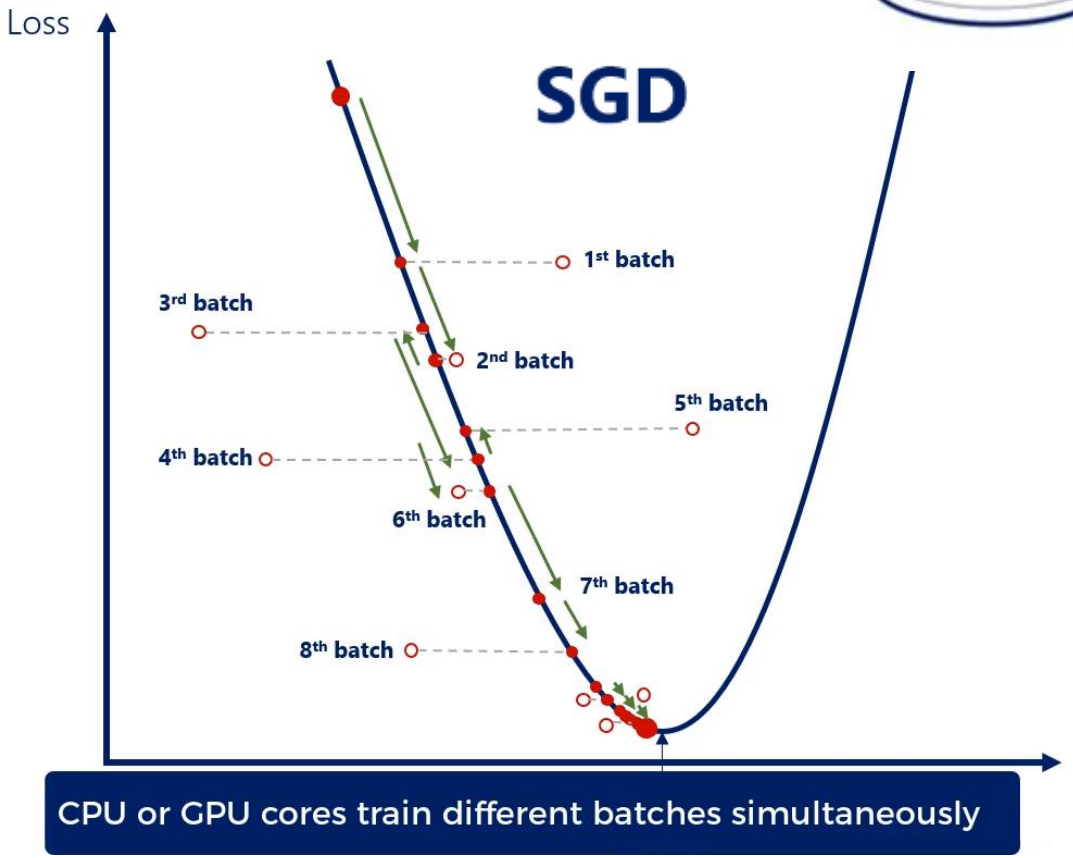
(Minibatch) Stochastic Gradient Descent: is a version of the algorithm that speeds up the computation by approximating the gradient using smaller batches (subsets) of the training data. SGD itself has various "upgrades".

1) Adagrad
2) Adam

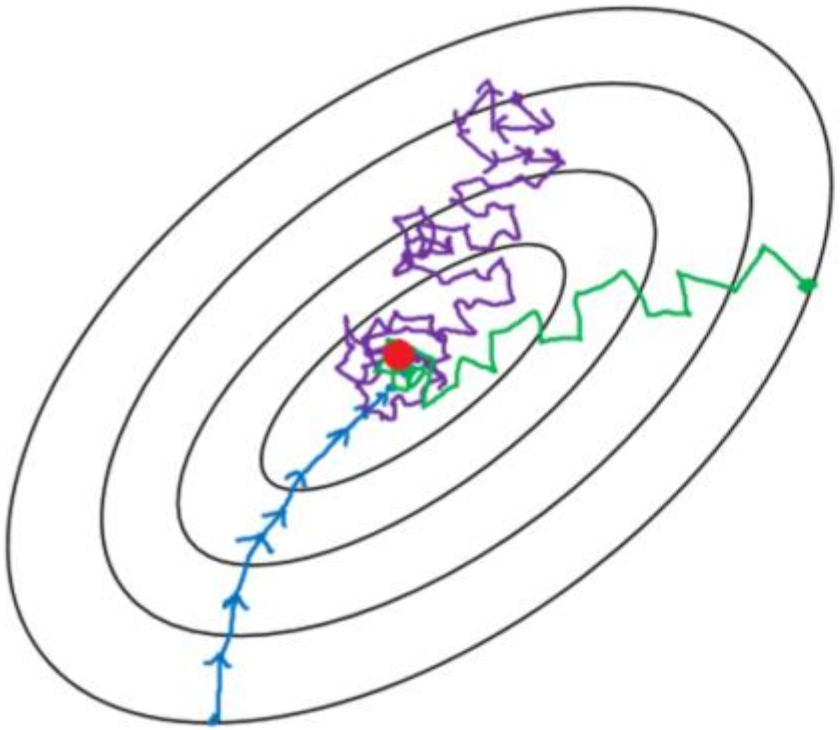
Why SGD?


CPU
Batches
Core
Core
Core
Core

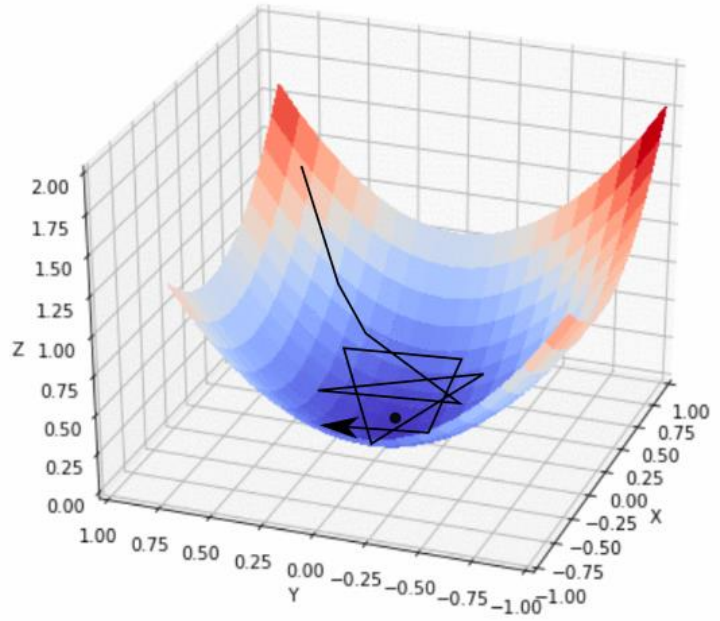

GD


Loss
SGD
1st batch
2nd batch
3rd batch
4th batch
5th batch
6th batch
7th batch
8th batch
CPU or GPU cores train different batches simultaneously

SGD vs GD



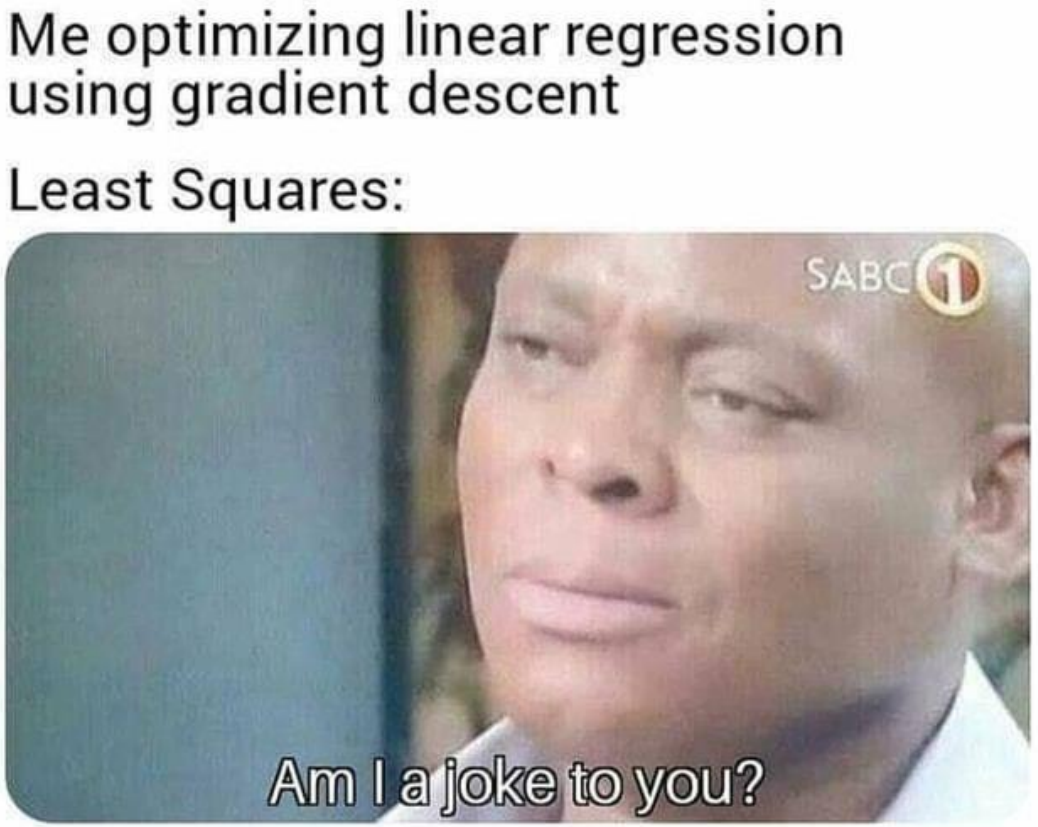
Batch gradient descent
Mini-batch gradient Descent
Stochastic gradient descent



Final message!

Notice that gradient descent and its variants are not machine learning algorithms. They are solvers of minimization problems in which the function to minimize has a gradient (in most points of its domain).

Question of the day!




class
DISMISSED