



Module 4 – Part I

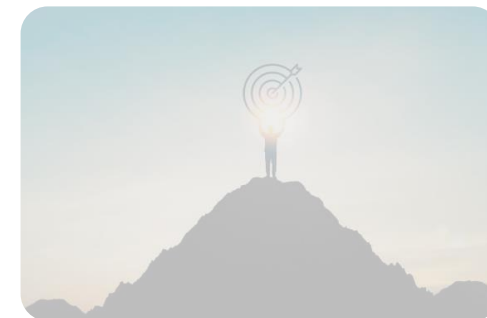
Machine Learning Fundamentals





Class Modules

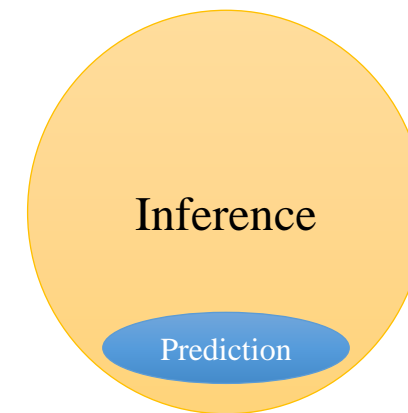
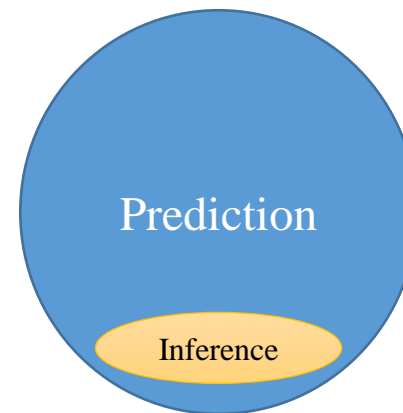
- Module 1- Introduction to Deep Learning
- Module 2- Setting up Machine Learning Environment
- Module 3- Linear Regression (Econometrics approach)
- **Module 4- Machine Learning Fundamentals**
- Module 5- Linear Regression (Machine Learning approach)
- Module 6- Penalized Regression (Ridge, LASSO, Elastic Net)
- Module 7- Logistic Regression
- Module 8- K-Nearest Neighbors (KNN)
- Module 9- Classification and Regression Trees (CART)
- Module 10- Bagging and Boosting
- Module 11- Dimensionality Reduction (PCA)
- Module 12- Clustering (KMeans – Hierarchical)





Topics

- Inference vs Prediction
- The Model
- Train, Test, Validation
- Resampling methods
- Evaluation metrics
- Bias-Variance tradeoff, overfitting
- How do machines actually learn?
 - Cost Function
 - Solvers/learners (GD, SGD)
- Scaling the features



The Model



The Model

$$y = f(X, \theta) + \epsilon = f(X_1, X_2, \dots, X_m, \theta_1, \theta_2, \dots, \theta_k) + \epsilon$$

y : response, dependent variables, output, **Target**

X : predictors, independent variables, input, **Features**

θ : estimates, specifications, **Parameters**

✓ It is all about estimating f by \hat{f} for two purposes:

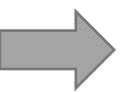
- 1) Inference (interpretable ML)
- 2) Prediction

→ Parameters and Hyperparameters

$$y = f(X, \theta) + \epsilon = f(X_1, X_2, \dots, X_m, \theta_1, \theta_2, \dots, \theta_k) + \epsilon$$

Model **parameters** are estimated from data automatically and model **hyperparameters** are set manually (prior to training the model) and are used in processes to help estimate model parameters.

Example?



Parametric Vs. Nonparametric models

$$y = f(X, \theta) + \epsilon$$

- The true relationship, $f(X)$ is unknown and the goal is to see which ML algorithm is better at approximating it. An algorithm learns/estimates $f(X)$ from training data.

$f(X)$ is **assumed**. Examples:
Linear regression, GLM,
logistic regression, simple
Neural networks,



Parametric
algorithms

Pros



Simpler

Easier to understand and to interpret

Faster

Very fast to fit your data

Less data

Require "few" data to yield good perf.

Cons



Limited complexity

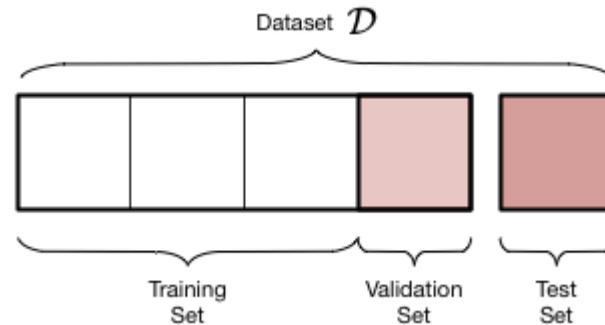
Because of the specified form, parametric algorithms are more suited for "simple" problems where you can guess the structure in the data

Train, Validation, Test Resampling Methods

→ Partitioning of the dataset

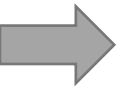
The data set is typically divided into three non-overlapping samples:

- 1) **Training set**: to train the model
- 2) **Validation set**: to validate and tune the model
- 3) **Test set**: to test the model's ability to predict well on new data (**generalize**)



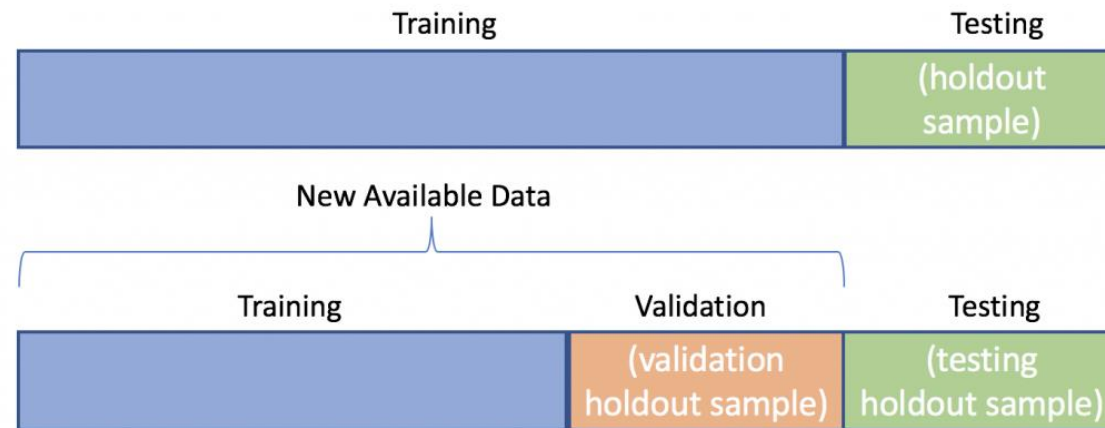
To be valid and useful, any supervised machine learning model **must** generalize well beyond the training data.

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



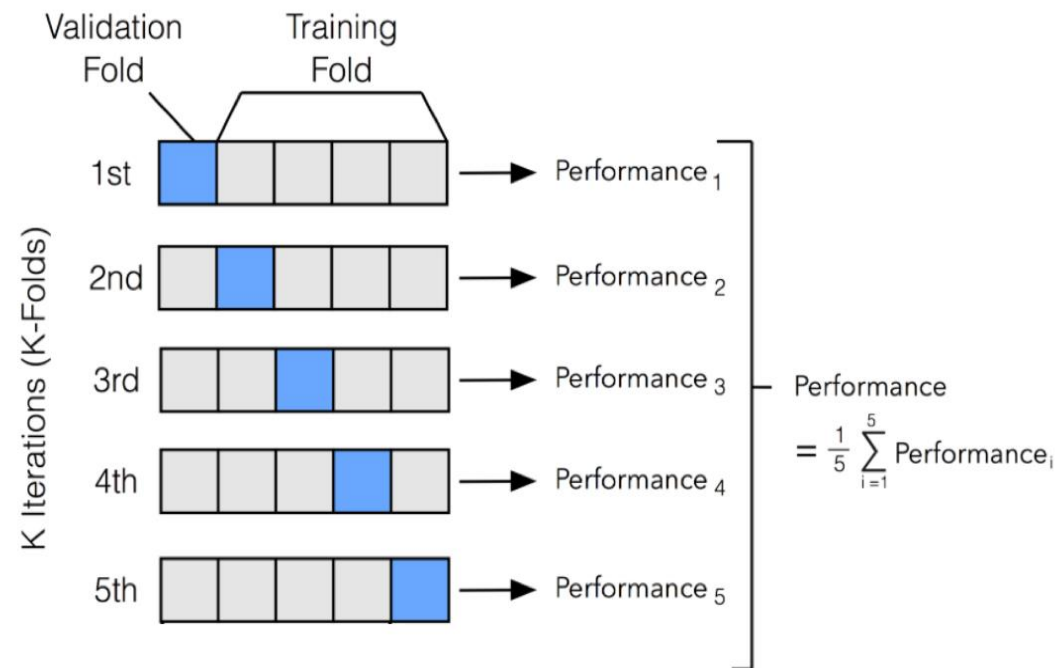
Resampling methods

- 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! **Unstable** model performance in validation set!
- **Solution**: combining the training and validation sets and use cross validation!



→ 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) This is done in turn for each part k and then the results are combined.



➔ Why do we use Cross Validation?

Cross validation is mainly used for two purposes:

1. Model **architecture** selection (optimization vs generalization)
2. **Estimation** of model performance in the test set



- After selecting the **best model architecture**, we estimate the generalization error using the test set.
- **Different model comparison** is based on **test set** performance!

Evaluation Metrics

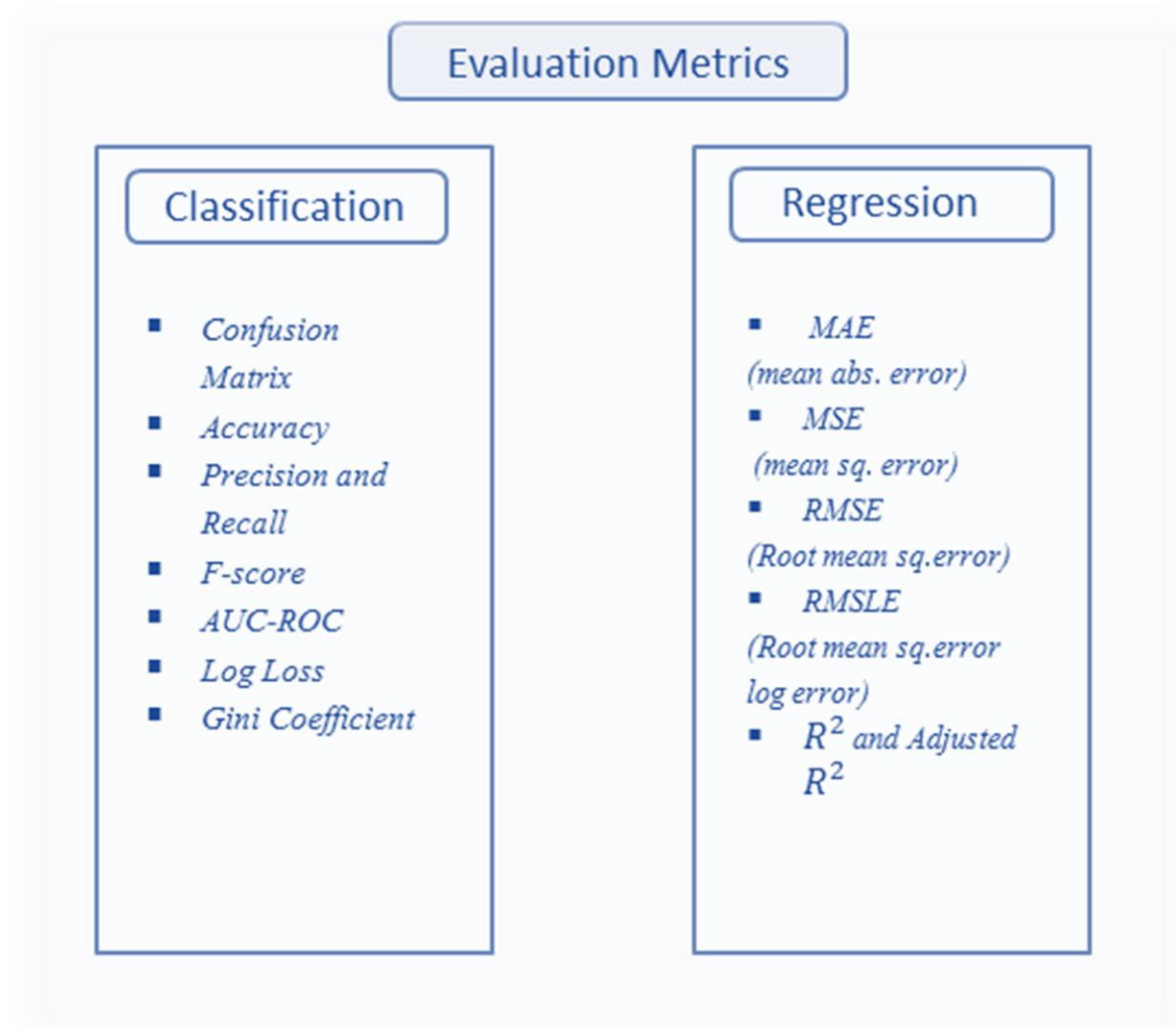


Evaluation metrics

In general, we want to compare how close are the predictions to the actual numbers in the **test set**.

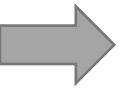
This is typically assessed using

- MSE for **quantitative** response
- Misclassification rate for **qualitative** response

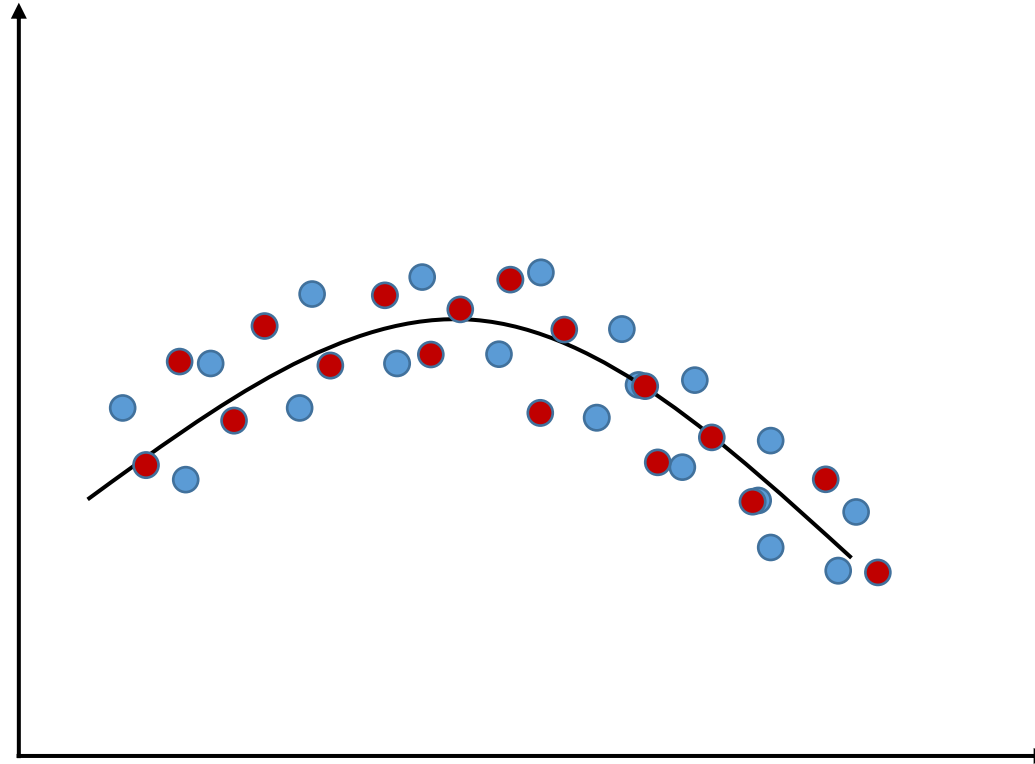


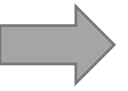
Bias-Variance Tradeoff

Optimization vs Generalization

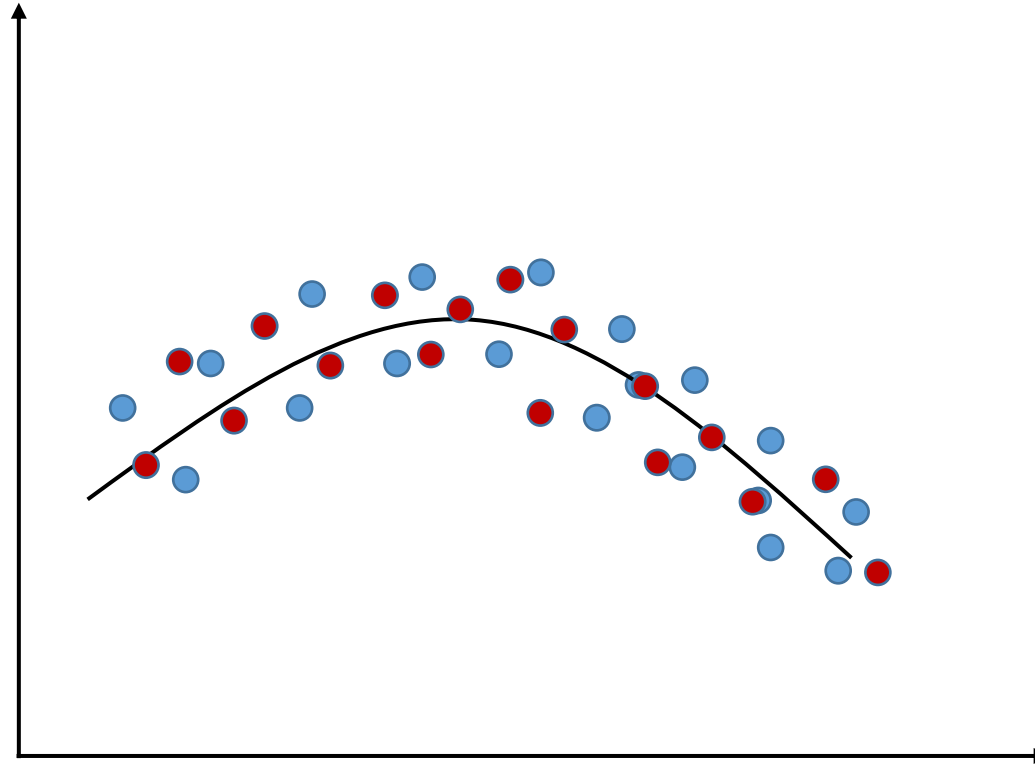


Model Bias & Model Variance in machine learning





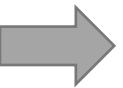
Model Bias & Model Variance in machine learning



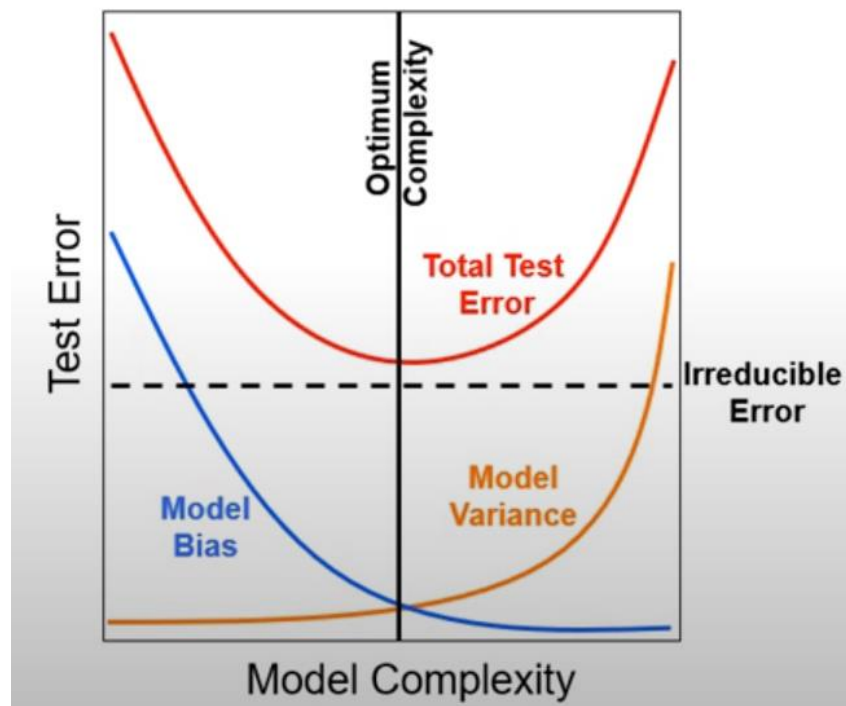
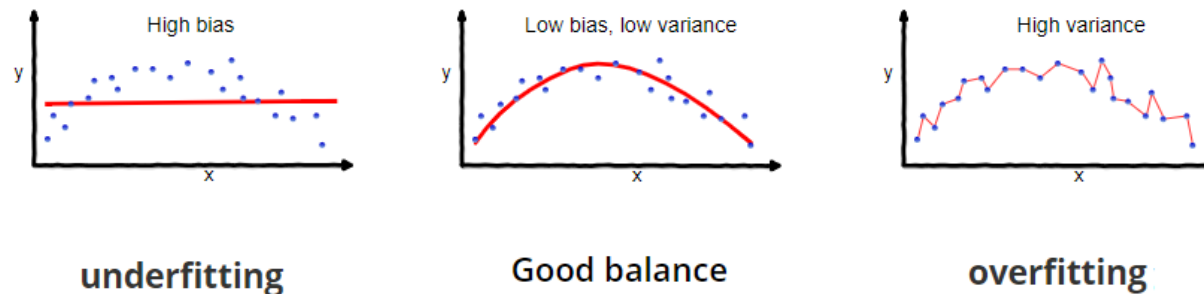
→ MSE decomposition

$$MSE = \text{model variance} + \text{model bias} + \text{irreducible error}$$

- 1) **Model variance** is the variance if we had estimated the model with a different **training set**
 - 2) **Model bias** is the error due to using an approximate model (model is too simple)
 - 3) **Irreducible error** is due to missing variables and limited samples. Can't be fixed with modeling
- The goal is to minimize the sum of **model variance** and **model bias**.
 - This is known as the bias-variance tradeoff because reducing one often leads to increasing the other.
 - Choosing the flexibility (complexity) of $\hat{f}(X)$, will amount to bias-variance tradeoff.



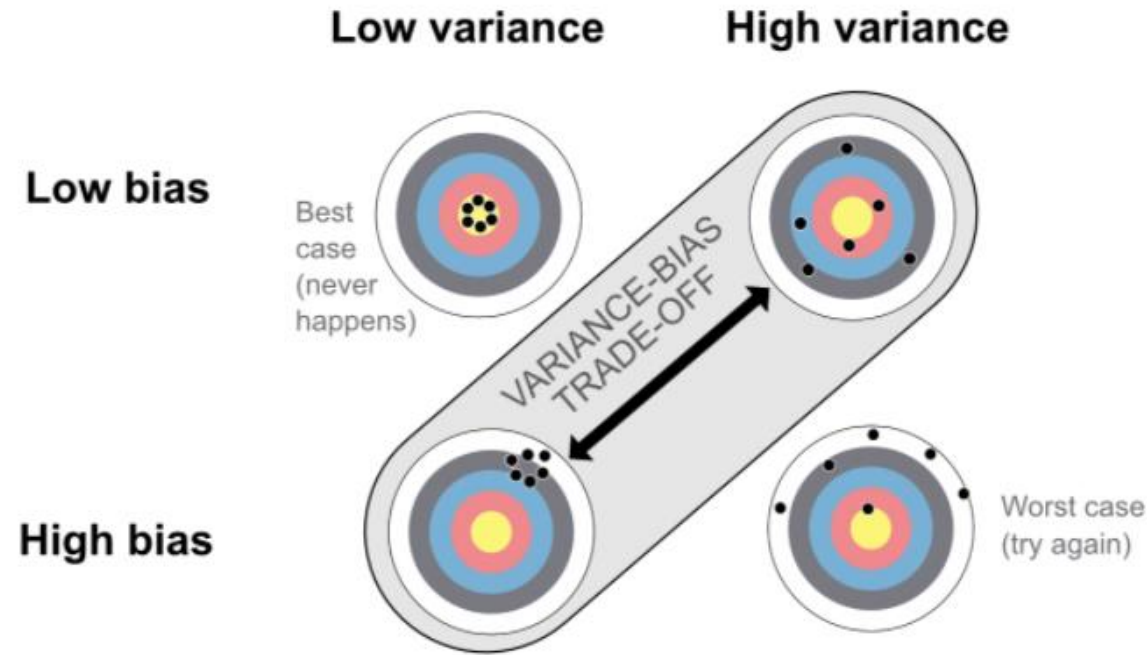
Representations of the bias-variance tradeoff



Optimization
Vs
Generalization

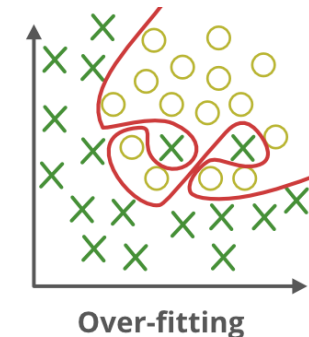
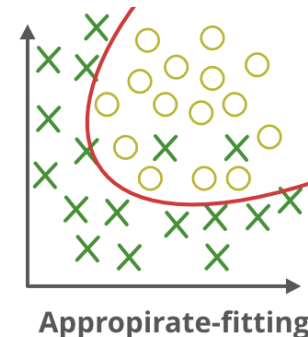
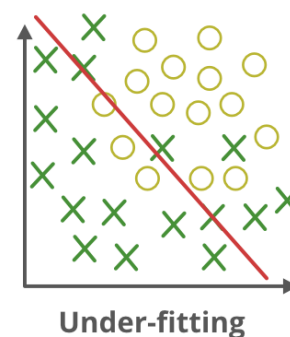
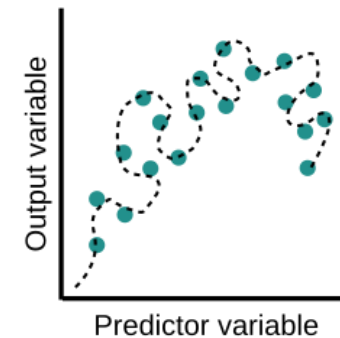
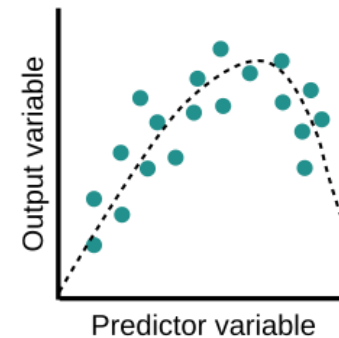
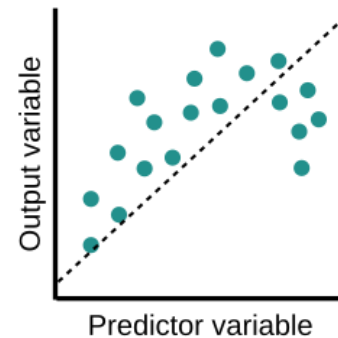


Other representations of the bias-variance tradeoff





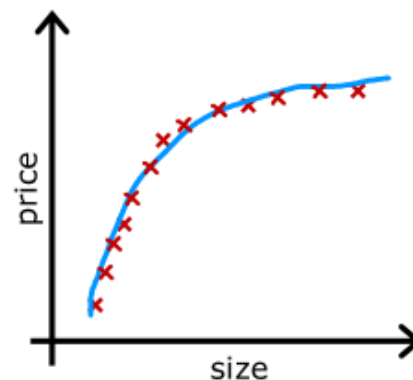
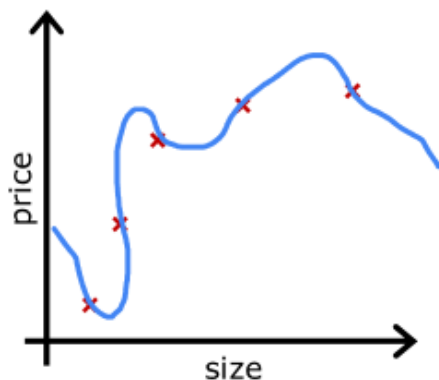
Overfitting



➔ Mitigate overfitting

The main techniques used to mitigate overfitting risk in a model construction are:

- 1) Collect **more data** (Can reduce AND variance)
- 2) **Complexity** reduction (regularization, feature selection)
- 3) **Cross validation** (estimate the performance in test set)



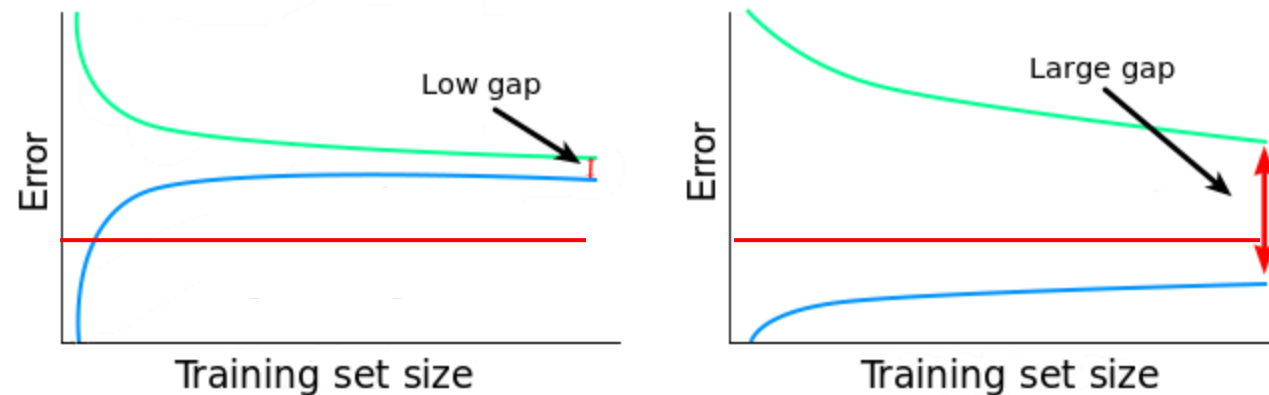
With more training example



The Learning Curve: Do we need to collect more data?

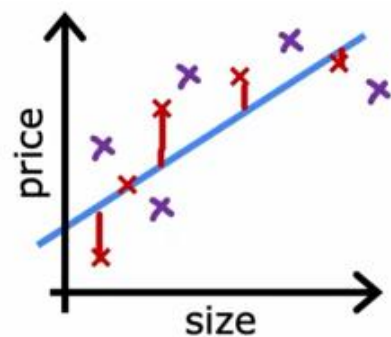
- A learning curve is a plot that shows the relationship between the amount of **training data** and the **performance** of a machine learning model.
- It is used to diagnose whether a model has high **bias**, high **variance**, or is **just right**.

-- Training score
-- Cross validation score
-- Benchmark performance
(common sense performance)





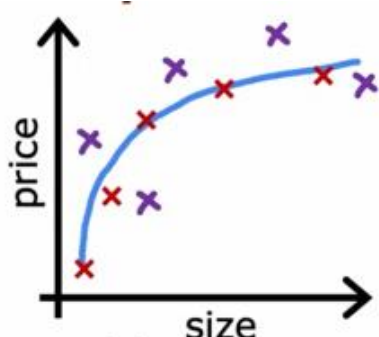
Class exercise



$$f_{\bar{w},b}(x) = w_1x + b$$

→ High bias
(underfit)

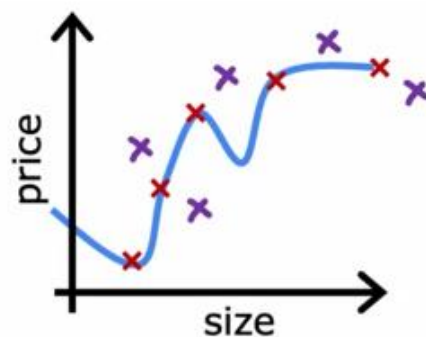
J_{train} J_{cv} ?



$$f_{\bar{w},b}(x) = w_1x + w_2x^2 + b$$

"Just right"

J_{train} J_{cv} ?

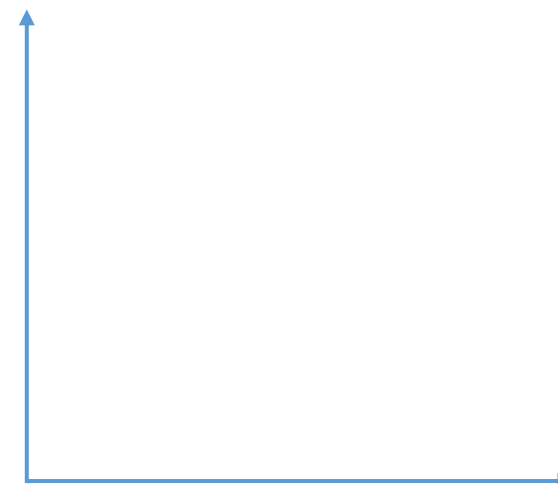


$$f_{\bar{w},b}(x) = w_1x + w_2x^2 + w_3x^3 + w_4x^4 + b$$

High variance
(overfit)

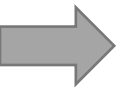
J_{train} J_{cv} ?

Cost



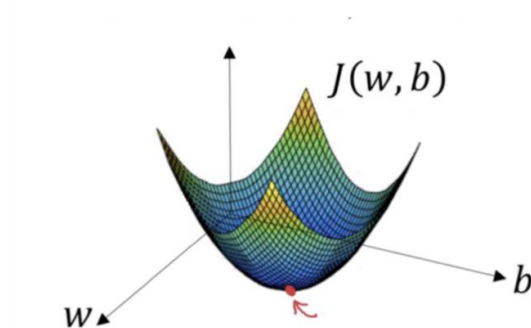
Model complexity

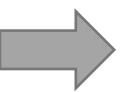
How do Machines Learn?



Terminology

- Learning: Finding the model **weights** (parameters' values)
- **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.
- The two most frequently used optimization algorithms when the cost function is **continuous** and **differentiable** are Gradient Descent (GD) and Stochastic GD.





Solvers (learners)!

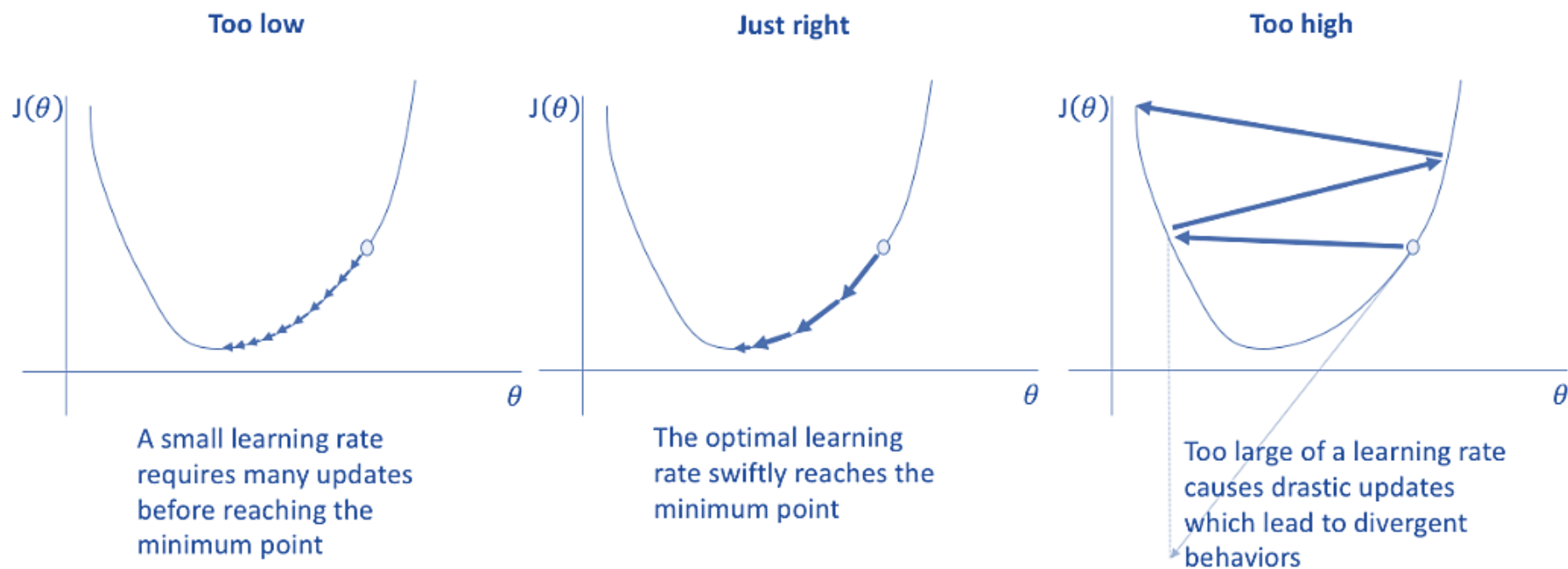
- **Gradient Descent**: is an **iterative** optimization algorithm for finding the minimum of a function.
- We start at some random point and take steps proportional to the **negative** of the gradient of the function at the current point.

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

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

Choice of learning rate

- If α is too small, gradient descent can be slow
- If α is too large, the gradient descent can even diverge.



➔ Beyond Gradient Descent?

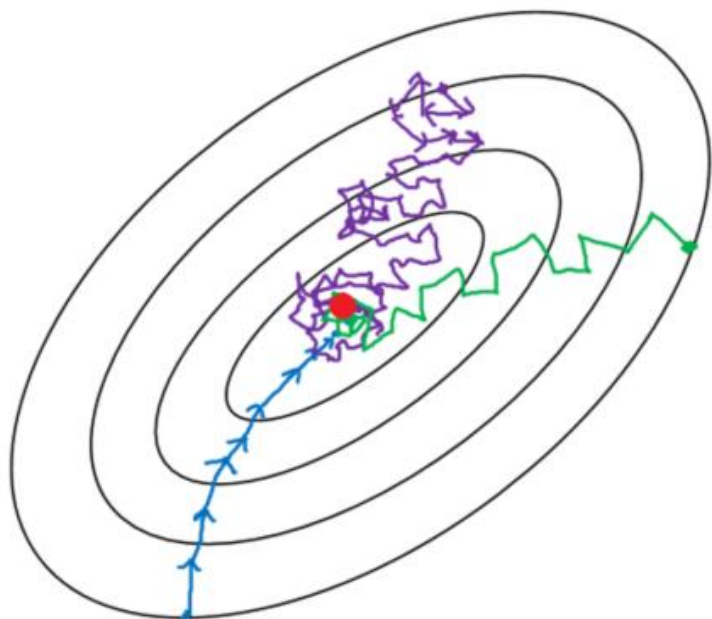
Disadvantages of gradient descent:

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

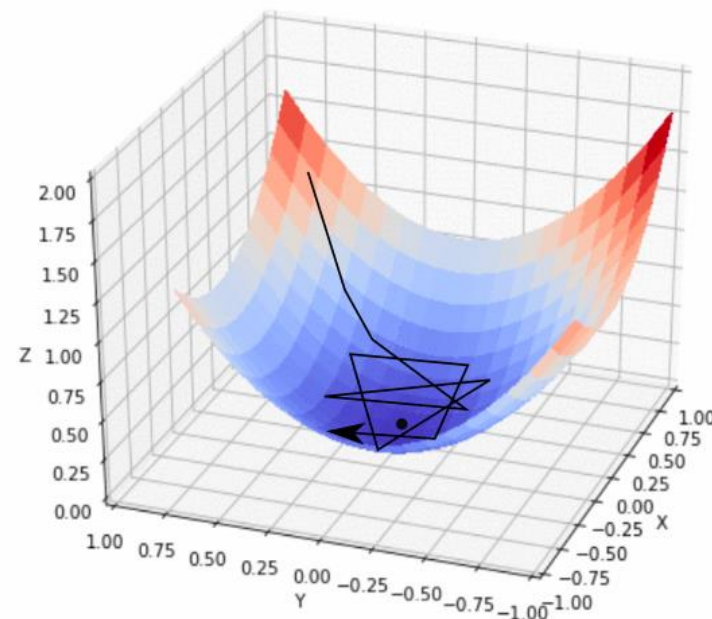
(Minibatch) Stochastic Gradient Descent: is a version of the algorithm that speeds up the computation by approximating the gradient using **smaller batches** (subsets) of the training data. SGD itself has various “upgrades”.



SGD vs GD



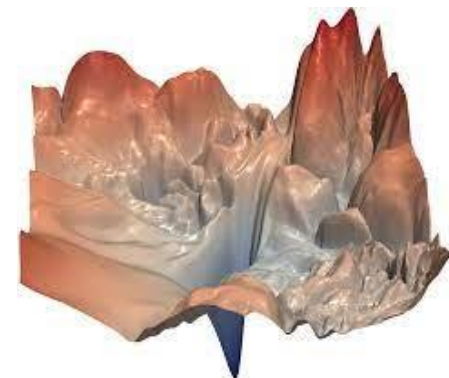
- Batch gradient descent
- Mini-batch gradient Descent
- Stochastic gradient descent





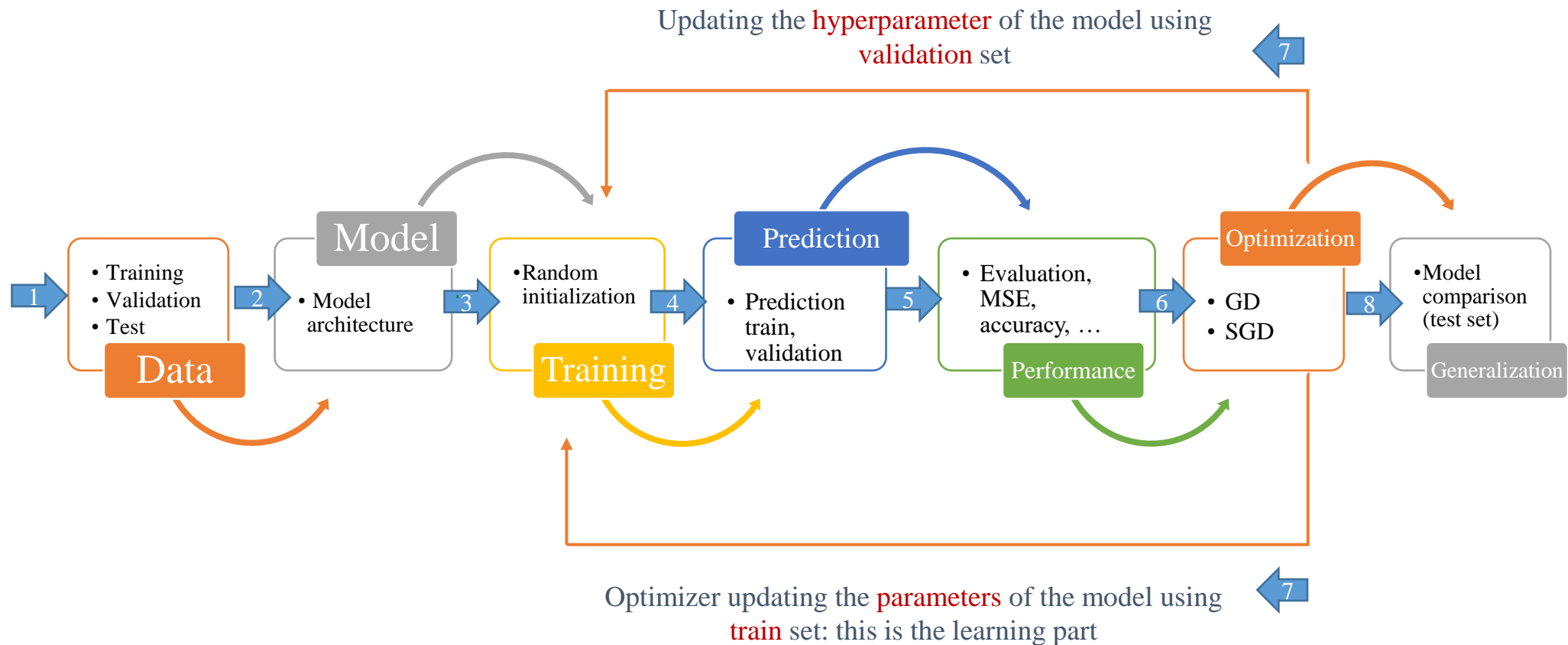
Beyond SGD?

- Loss functions can be difficult to optimize!
- *Visualizing the loss landscape of neural nets*, Li et al, 2018



- **Solution:** Designing an adaptive learning rate that can **adapt** to the loss landscape.
- Rather than just looking at **the current gradient**, consider the **previous weight updates**.
- This is called, **momentum**!
- Examples: Adam, Adadelata, Adagrad, RMSProp!

How do machines learn?

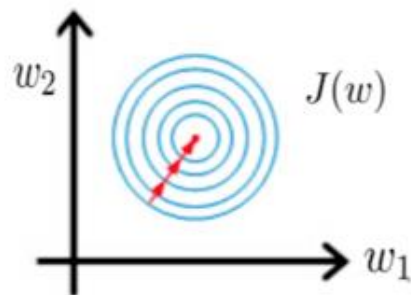


Feature Scaling

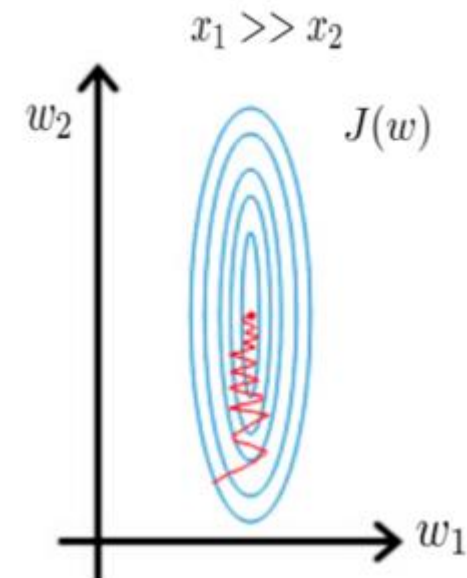
➔ Why feature scaling?

- Feature scaling in machine learning is a critical step during the **pre-processing** of data **before creating a machine learning model**.
- Feature scaling is essential for machine learning models that calculate **distances between data**.
- Feature scaling could:
 - Avoid numerical overflow and speed up the algo
 - Reduce dominant effects of specific variables

$$0 \leq x_1 \leq 1$$
$$0 \leq x_2 \leq 1$$



Both parameters could be updated in equal proportions

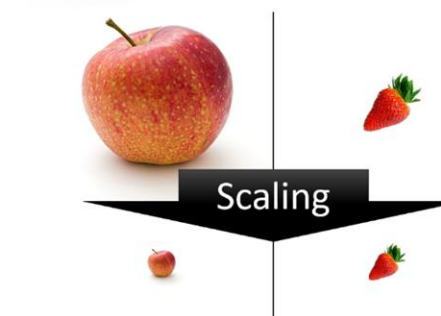


Gradient of larger parameters dominates the updates

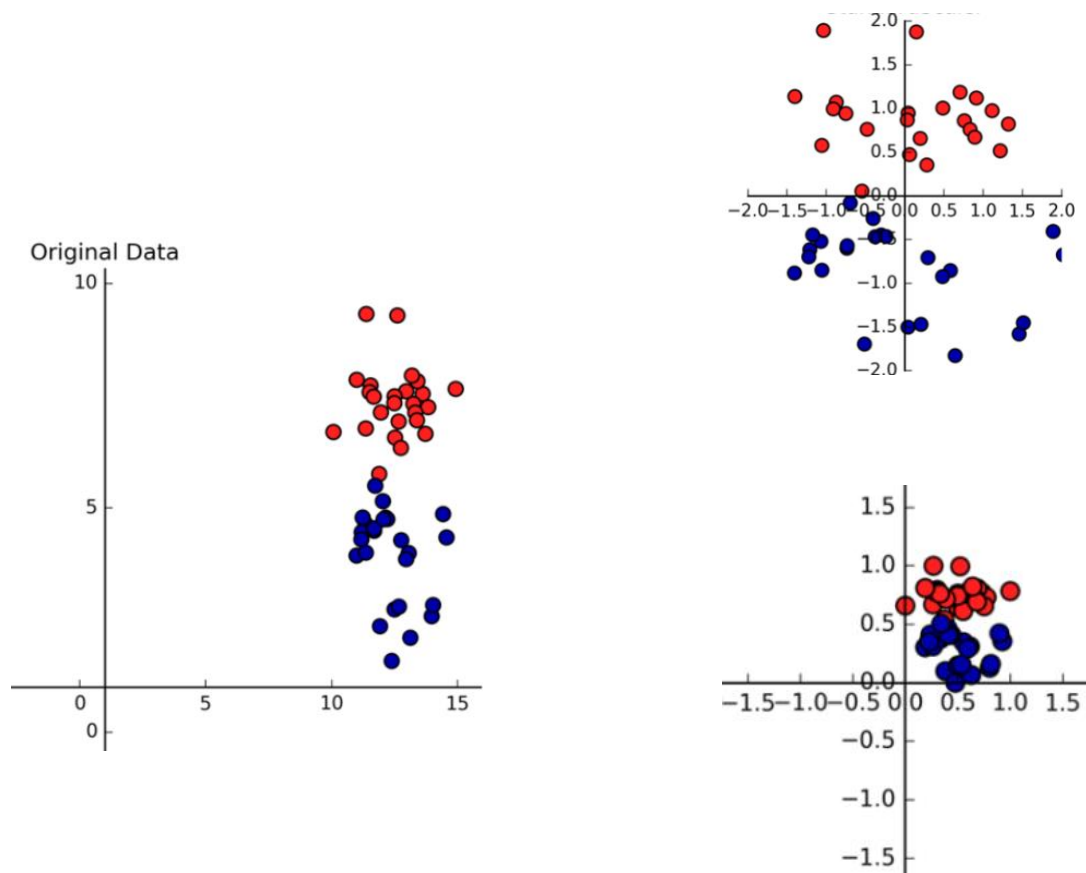
➔ Scaling the features

Let us use x_i for raw input and \tilde{x}_i for the transformed data.
Common scaling practices include:

- Standardization (Z-score):
$$\tilde{x}_i = \left(\frac{x_i - \mu_x}{\sigma_x} \right)$$
- Normalization:
 - Min-Max scaler over [0,1]:
$$\tilde{x}_i = \left(\frac{x_i - \text{Min}(X)}{\text{Max}(X) - \text{Min}(X)} \right)$$
 - Min-Max scaler over [-1,1]:
$$\tilde{x}_i = 2 * \left(\frac{x_i - \text{Min}(X)}{\text{Max}(X) - \text{Min}(X)} \right) - 1$$
 - Mean normalization:
$$\tilde{x}_i = \left(\frac{x_i - \text{Mean}(X)}{\text{Max}(X) - \text{Min}(X)} \right)$$



➔ Scaling the features (class exercise)



aim for about $-1 \leq x_j \leq 1$ for each feature x_j
 $-3 \leq x_j \leq 3$
 $-0.3 \leq x_j \leq 0.3$ } acceptable ranges

- $0 \leq x_1 \leq 3$
- $-2 \leq x_2 \leq 0.5$
- $-100 \leq x_3 \leq 100$
- $-0.001 \leq x_4 \leq 0.001$
- $98.6 \leq x_5 \leq 105$

Which of these
ranges need to be
scaled?

➔ Some general hints with scaling

- Be careful when scaling the **time series** data! Why?
- To avoid **data leakage**, it is a good practice to fit the scaler on the training data and then use it to transform the testing data.
- Scaling the data does NOT change the shape of the **distributions**.





Class Modules

- ✓ Module 1- Introduction to Deep Learning
- ✓ Module 2- Setting up Machine Learning Environment
- ✓ Module 3- Linear Regression (Econometrics approach)
- ✓ Module 4- Machine Learning Fundamentals
- Module 5- Linear Regression (Machine Learning approach)
- Module 6- Penalized Regression (Ridge, LASSO, Elastic Net)
- Module 7- Logistic Regression
- Module 8- K-Nearest Neighbors (KNN)
- Module 9- Classification and Regression Trees (CART)
- Module 10- Bagging and Boosting
- Module 11- Dimensionality Reduction (PCA)
- Module 12- Clustering (KMeans – Hierarchical)

