WELLS FARGO

Workshop on Machine Learning and Application

Module 1: Algorithms, Training, and Tuning

March 24, 2023

Vijay Nair
Joint with Anwesha Bhattacharyya and Rahul Singh

Advanced Technologies for Modeling (AToM) Group
Corporate Model Risk

Outline

- Machine Learning: Overview
- Background
- Algorithms
  - Architecture
  - Training
  - Tuning
  - Examples

References

Overview

- Hu, L., et al. (2021) Supervised Machine Learning Techniques: **An Overview** with Applications to Banking*, International Statistical Review*, Volume 89 (Issue3) pages, p.573 - 604.
- Breiman, L. (2001b). **Statistical Modeling: The Two Cultures** (with comments and a rejoinder by the author). *Statistical Science, 16*, 199-231.

Algorithms

- Breiman, L. (2001a). **Random Forests.** *Machine Learning, 45*, 5-32.
- Friedman, J. (2001). Greedy Function Approximation: **A Gradient Boosting Machine**. *The Annals of Statistics, 29*, 1189-1232.
- Chen, T., & Guestrin, C. (2016). **XGBoost:** A Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* San Francisco.
- Goodfellow, I., Bengio, Y., & Courville, A. (2015). ***Deep Learning***. Cambridge, MA: MIT Press.

Explanation Techniques

- Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. *Advances in neural information processing systems*, (pp. 4765-4774).
- Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). " Why should i trust you?" **Explaining the predictions** of any classifier. *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, (pp. 1135-1144).
- Song, E., Nelson, B. L., & Staum, J. (2016). **Shapley Effects for Global Sensitivity Analysis**: Theory and Computation. *J. Uncertainty Quantification, 4*(1), 1060-1083.

Machine Learning and Artificial Intelligence

- **Machine Learning:**
  - Term coined by **Arthur Samuel (IBM)** in 1959
  - ML gives "computers the ability to learn without being explicitly programmed”
  - Study and construction of algorithms that can learn from data, identify features, recognize patterns, make predictions, and take actions
  - A **key pathway to AI**

- **Artificial Intelligence:** concerned with making computers behave like humans
  - Term coined by **John McCarthy (MIT)** around 1956
  - Study of “ intelligent agents” [or systems] that “perceive” the environment and take actions that maximize [probability] of success [to achieve] some goal
  - **Long history**: formal reasoning in philosophy, logic, …
  - **Resurgence** of AI techniques in the **last decade**: advances in computing power, computing and data architectures, sizes of training data, and theoretical understanding
  - **Deep Learning Neural Networks**: At the core of recent advancements in AI, specifically for certain classes of ML tasks (Reinforcement L and Representation L)

Machine Learning Tasks

- **Supervised Learning**
  - Data with **“labels”**
  - **Regression and classification**

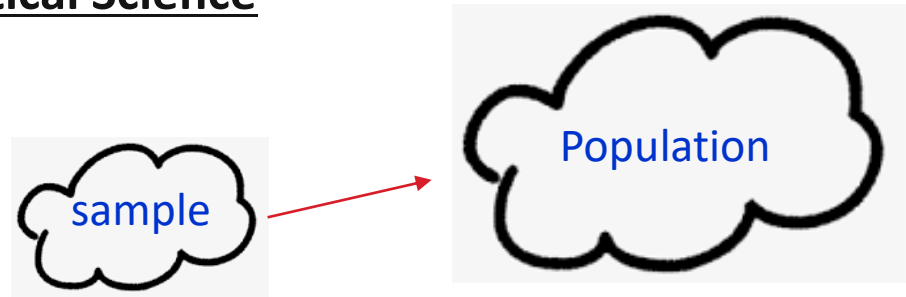
- Unsupervised Learning
  - Data with **no labels**
  - **Discover patterns or structure** in the data (anomalies, clusters, lower-dimensional representation)

- Reinforcement Learning
  - **Experiment** and **exploit** to **make “optimal” decisions** based on **reward** structure

- Others
  - Semi-supervised, Positive-Unlabeled Learning, …
  - Representation Learning
  - Transfer Learning

Supervised Learning: Statistics vs ML paradigms

- **Leo Breiman (2001) *Statistical Modeling: The Two Cultures*, Statistical Science**
  - Two paradigms: data model and algorithmic model



- Traditional statistics
  - Goal: "understand" the generative model
    - **Estimate** model **parameters** and **assess uncertainty**
    - Identify **key drivers** and **input-output relationships**
    - **Extensive tools and diagnostics** developed over time
    - Parametric **models** → **easier to interpret**

- Machine Learning
  - Goal: best predictive performance … generalization assessed on hold-out data
    - **Algorithmic** approach and **automation** of model building
      → variable selection, feature engineering, model training
    - Large samples
    - Not much focus on CI, hypothesis testing, …
  - **No** intrinsic **interest in** the **data generation process** (even if there's such a thing!)

- **For regulated industries and safety-critical applications:**
  - Model interpretability is important

Applications in Banking

Areas:

- **Credit Risk:** Predicting **losses** – customers **not repaying debts or loans**: Mortgages, Auto-Loans, Student Loans, Credit cards, Small business loans, …
- **Credit Decisions: Activities related to loan applications**: credit scoring, marketing, collections, …
- **Revenue and Transactions:** Interest, servicing fees, deposits, withdrawals, electronic payments, etc.
- **Financial Crimes:** Fraud detection, Money laundering
- **Fair Lending**: Ensuring fair treatment of customers
- **Text and speech**: Conversations, complaints, emails, voice messages, chat-bots for assisting customers and employees

Statistical and econometric techniques

- Dimension reduction; clustering, anomaly detection
- Parametric modelling for regression and classification
- Semi- and non-parametric regression models
- Regularization: Lasso, ridge, …
- Survival analysis; Time series forecasting

ML/AI techniques

- Auto-encoders → Dimension reduction
- Isolation Forest → Anomaly detection
- **Supervised ML**: Support vector machines; **Random forests; Gradient boosting; Neural networks** (FF and Deep NNs)
- **Natural language processing** of Text Data → Deep NNs
- Conversational AI: Chatbots, …

Computing Environment

- Python, C++, Java, R, SAS, and R
- Open Source Libraries, Tensorflow, PyTorch
- CPU and GPU Clusters, Cloud

Natural Language Processing (NLP)

- Methods, algorithms, and systems for **analyzing "human language" data** (text, speech, conversations)
  - **Very challenging** …
- **Interdisciplinary area** that combines computer science, statistics, optimization, AI, linguistics, logic …
  - Earlier version → computational linguistics, speech recognition, …
- **Evolution:**
  - Rule-based, statistical …now **largely driven by deep neural networks**
- **Diverse applications**

Text Summarization

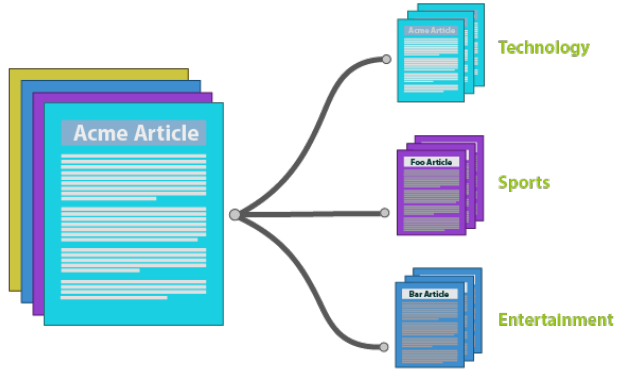
Machine Translation

Chatbots

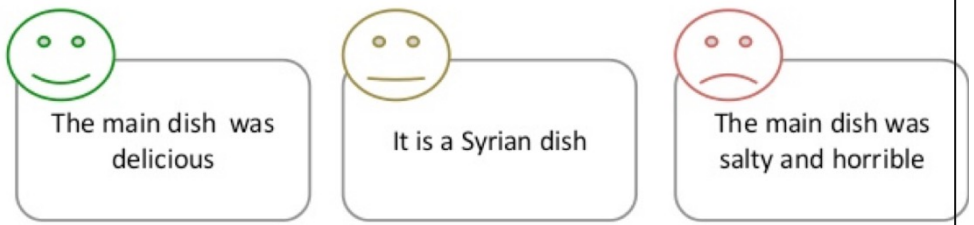
- Alexa and Siri-like
- Conversational AI

Natural Language Generation

Text classification



Sentiment Analysis



Opportunities with ML

General:

- **Advent of "Big Data"**
  - ✓ **New sources of data**: social media, sensor networks, intelligent systems, …
    - Text, conversations, …

- **Advances in computing and data storage technologies**
  - ✓ Infrastructure for data collection, warehousing, transfer, and management
  - ✓ Efficient and scalable algorithms and associated technologies for analyzing large datasets
  - ✓ Open-source algorithms
  - ✓ Cloud storage and computing
    - → Democratization of Data Science

Specific:

- Availability of large datasets and fast algorithms
  - → flexible modeling … move away from restrictive parametric models
- SML model:
  - ➢ Improved predictive performance
  - ➢ Semi-automated approach to feature engineering and model training → ideal for Big Data
- New data sources and computing technologies open up new opportunities
  - ➢ Text, speech, images, …
  - ➢ More timely information and decision making

Structured data

- Structured or tabular data with response/label
  - n observations
  - p covariates
  - static, time series, repeated measurements, …

- Examples in banking:
  - Banking transactions over time
    - Response: deposit and withdrawals (times and amounts)
    - Covariates: account holder's attributes, account type, location, etc.

  - Historical loan payment data
    - Response: default or not (binary 0 or 1)
    - Covariates – amount of original loan, balance over time, credit rating over time, delinquency status, etc.

Obs id	x_1	x_2	…	x_p	y
1					
2					
…					
n					

Estimation and loss functions

- **Continuous response**:$Y(x) = function(inputs, noise)$
- **Binary response:** $\boldsymbol{Y(x) = 0\ or\ 1}$;
  - Let $\mu(x) = E(Y(x))$
  - Model is: $g(\mu(x))m = f(\boldsymbol{x})$
  - Here $g(\cdot)$ is the link function

Goal:

Estimation of $f(\boldsymbol{x})$

Prediction: given $\boldsymbol{x}^*$, $predict\ f(\boldsymbol{x}^*)$

How to estimate $\boldsymbol{f(x)}$?

- Given data $\{y_i, \boldsymbol{x}_i, i = 1, \dots, n\}$,
- Find $\hat{y} = \hat{f}(x)$ that minimizes some loss/cost.
- Loss functions are specific to the learning task.

Common loss functions

Continuous response

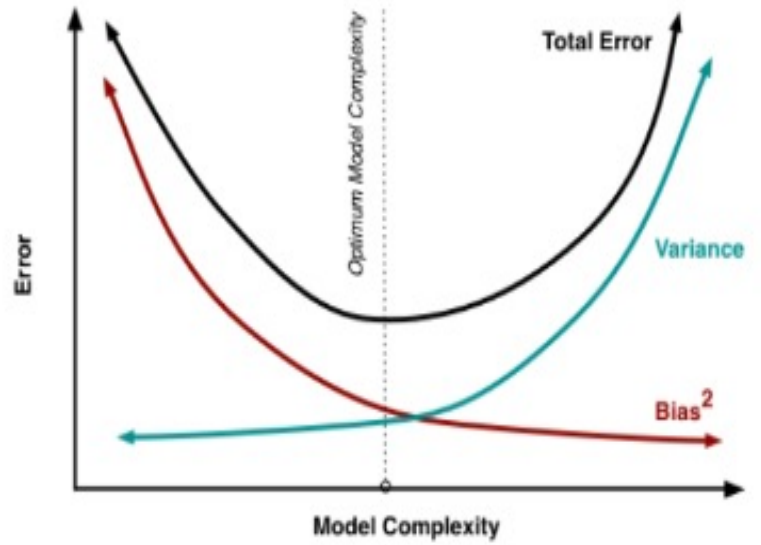
- Mean square loss : $\frac{1}{n}\sum_i (y_i - \hat{y}_i)^2$.
- MAE
- Quantile loss

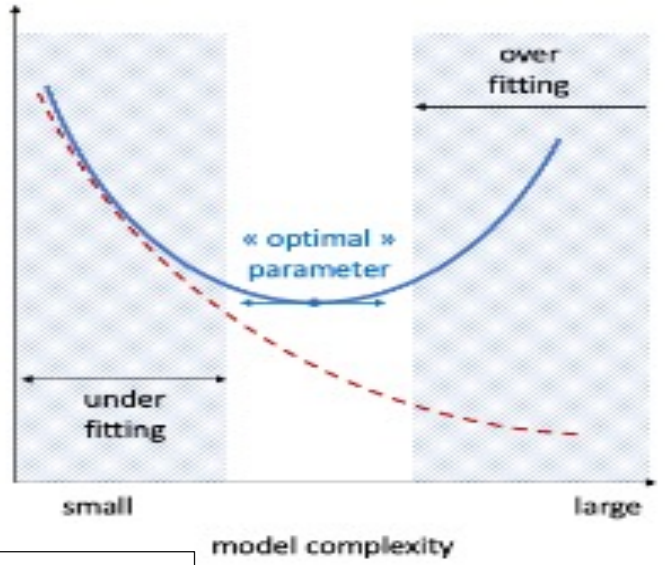
Binary Response

- Logloss:
- $\frac{1}{n}\sum_i [y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i)]$

Bias variance tradeoff and over-fitting

- All learning algorithms come with **hyper-parameters (HP)** which control complexity of the algorithm.
- **Examples**
  - High dimension in linear/logistic regression : Large number of predictors
  - Degree of polynomial in polynomial regression
  - Number of knots in splines
  - Tree based ensemble learners : depth, number of trees, learning rate, minimum samples per leaf
  - Neural Network : number of layers, layer sizes,
- **Complexity of a model is related to bias-variance trade-off**
  - Underlying model : $y = f_\star(x) + \epsilon$
  - $E\left(\hat{f} - f_\star\right)^2 = Var\left(\hat{f}\right) + Bias\left(E\left(\hat{f}\right), f_\star\right)^2$
  - Very complex (over-fitted) model – low bias, high variance
  - Simple (under-fitted) model – high bias, low variance





Regularization

- **Regularization**: Applying penalty to loss function to penalize for model complexity

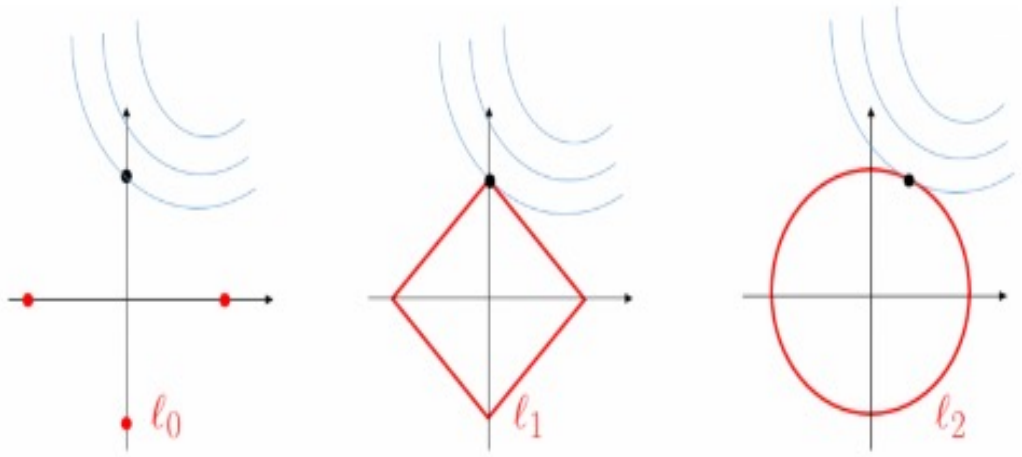
$$\boldsymbol{Regularized\ loss}$$

$$= \boldsymbol{Loss}(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{\theta}) + \boldsymbol{\lambda} \times \boldsymbol{Penalty}(\boldsymbol{complexity})$$

  - Penalty parameter λ is treated as hyper-parameter
  - Tune λ to obtain optimal model that minimizes hold-out validation error

- **Objective:** force some parameters to be small or 0 thereby reducing model complexity without compromising on performance too much.
- **Result :** Reduce over-fitting and generalizable model performance

- **L0:** $\boldsymbol{\lambda|\theta|_0}, \quad \boldsymbol{|\theta|_0 = \Sigma_j I\ (\theta_j \neq 0)}$
  - Exact sparsity
  - Non-convex, NP hard, not computationally tractable
- **L1:** $\boldsymbol{\lambda|\theta|_1}, \quad \boldsymbol{|\theta|_1 = \Sigma_j |\theta_j|}$
  - Exact sparsity - selection
  - Convex
- **L2:** $\boldsymbol{\lambda|\theta|_2}, \quad \boldsymbol{|\theta|_2 = \left(\Sigma_j \theta_j^2\right)^{\frac{1}{2}}}$
  - Convex, fast and efficient
  - Close to 0 but not exact sparsity
  - Effective in presence of multi-collinearity.



Overview of statistical approaches to regression

$$g\big(E(y(x))\big) = f(x)$$

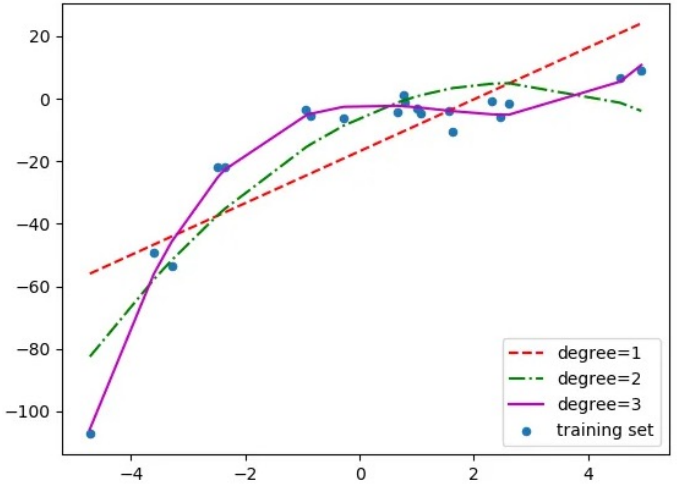
- Parametric
  - $f(x)$ is modelled as an explicit function of x and some parameter β
  - β is finite dimensional
  - Examples
    - Linear regression, logistic regression, polynomial regression
- Semi-parametric
  - $f(x)$ has a fixed dimensional component and an infinite dimensional component
  - Example: Cox proportional hazard models
- Non-parametric
  - $f(x)$ is general – infinite dimensional

Overview of statistical approaches – continuous response

Parametric modelling

Linear model with continuous response $y = X\beta + \epsilon$

- Assumptions
  - Linear
  - Constant variance of noise.
- Solved through least square approaches
  - Minimize $\left|\left|Y - X\beta\right|\right|_2^2$
  - $\hat{\beta} = (X'X)^{-1}X'y$
  - Requires $n > p$
  - Inference (hypothesis testing) – requires normality assumptions



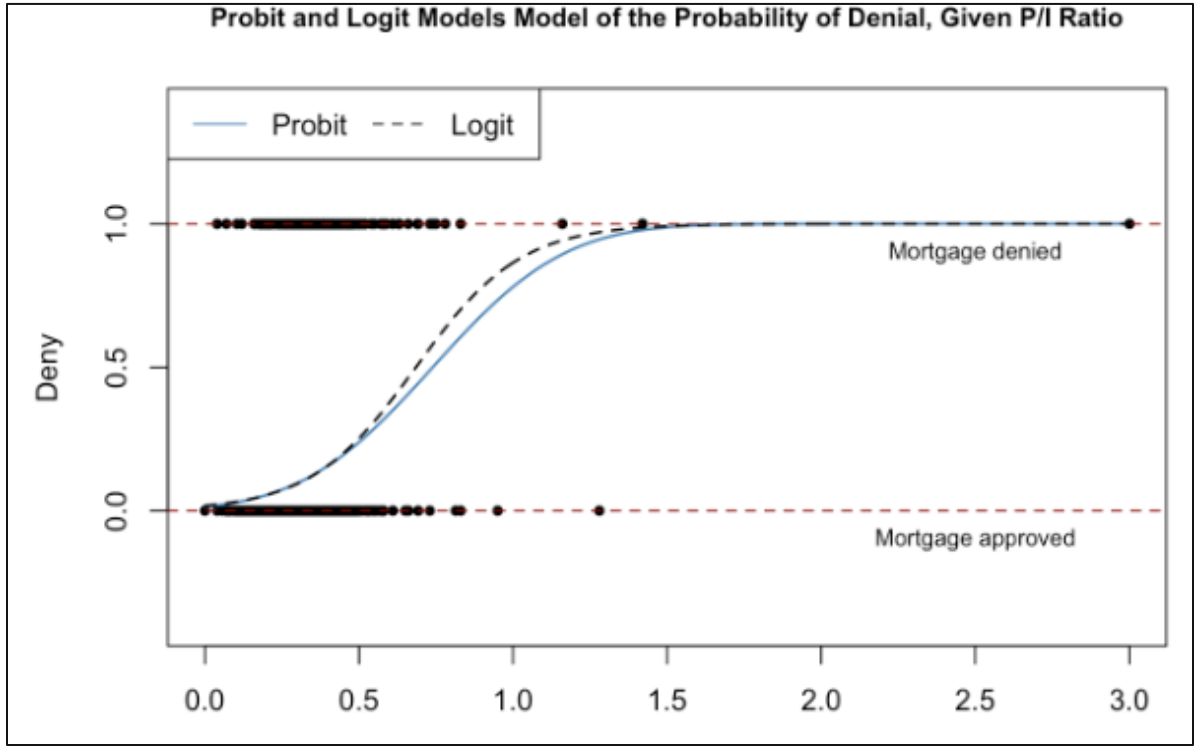
- Non-linear with continuous response
  - Polynomial regression X^2, X^3
  - Linear model with interactions $\mathrm{y} = \beta_0 + \beta_1 \mathrm{X}_1 + \beta_2 \mathrm{X}_2 + \beta_3 \mathrm{X}_3 + \beta_{12} X_1 X_2 + \beta_{123} X_1 X_2 X_3 + \epsilon$
  - Polynomial with interactions $\mathrm{y} = \beta_0 + \beta_1 \mathrm{X}_1^2 + \beta_2 \mathrm{X}_2^3 + \beta_3 \mathrm{X}_3 + \beta_{12} X_1 X_2 + \beta_{123} X_1 X_2 X_3 + \epsilon$

Overview of statistical approaches – binary response

- **Binary response**
  - Linear regression generates out of range values
  - Try to model the probability $\mathrm{p(x)} = Prob(y = 1|X)$
  - Assume $y|x \sim Ber(p(x))$
- Logistic Regression
  - $logit\left(p(x)\right) = log\left(\frac{p(x)}{1-p(x)}\right) = f(x)$
  - $f(x)$: linear in x
- Probit Regression
  - $\Phi^{-1}\big(p(x)\big) = f(x)$
  - Φ : cdf of normal distribution. Scales $R \rightarrow [0,1]$
  - $f(x)$: linear in x
- **Link functions**
  - $\Phi^{-1}\big(p(x)\big), \log\left(\frac{p(x)}{1-p(x)}\right)$

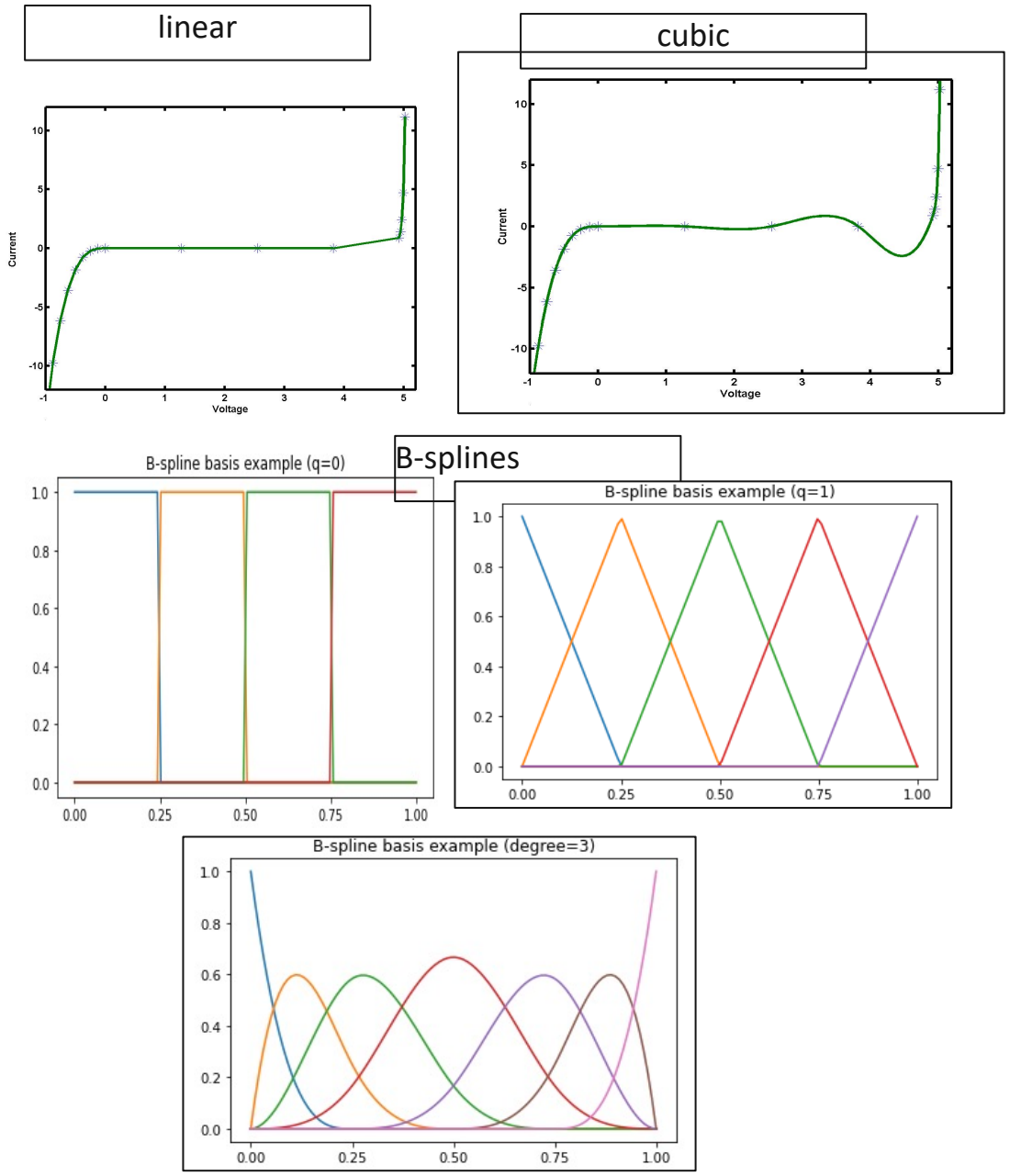
Generalized Linear Models

- Distribution of y
- $E(y|x) = \mu(x)$
- Link function : $\mathrm{g}\big(\mu(x)\big) = f(x)$
- $f(x)$: linear in x



Splines

- Piecewise functions used to model effects of each variable smoothly.
- Examples:
  - Linear splines (derivatives not continuous at knots)
    $f(x) = \beta_0 + \beta_1 x + \Sigma_{k=1}^{K} b_k (x - \xi_k)_+ \ , \xi_1 < \xi_2 < \cdots \xi_K$
  - Truncated power basis splines of order q (q-1 derivatives continuous)
    $f(x) = \beta_0 + \beta_1 x + \cdots \beta_q x^q + \Sigma_{k=1}^{K} b_k (x - \xi_k)^q{}_+ \ , \xi_1 < \xi_2 < \cdots \xi_K$
  - B-Splines
  - $f(x) = \Sigma_{k=1}^{K} b_k B_{kd}(x)$
  - $B_{kd}(x) : k_{th}$ spline of degree d
  - Defined through recursive relations with
  - $B_{k0}(x) = 1 \ if \ \xi_k \leq x < \xi_{k+1}$
  - $B_{k(j+1)}(x) = \alpha_{k(j+1)}(x) B_{kj}(x) + (1 - \alpha_{(k+1)(j+1)}(x)) B_{(k+1)j}(x)$
  - Efficiency in computation
  - Any spline function can be expressed as linear combination of b-splines



Generalized Additive Models (GAMs)

Hastie and Tibshirani

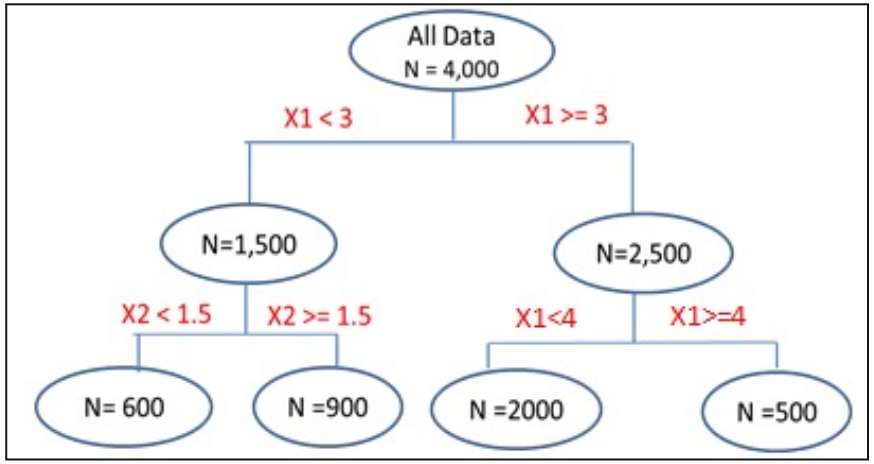
- $\boldsymbol{g(EY(x)) = f(x) = f_1(x_1) + f_2(x_2) + \ldots + f_p(x_p)}$
- $\boldsymbol{f_j(x_j)}$: **unspecified non-parametric functions**
- Usually assume they are smooth
- Each $f_j(X_j)$ is estimated nonparametrically
- Can use splines or some other semi- or nonparametric structures
- Example:

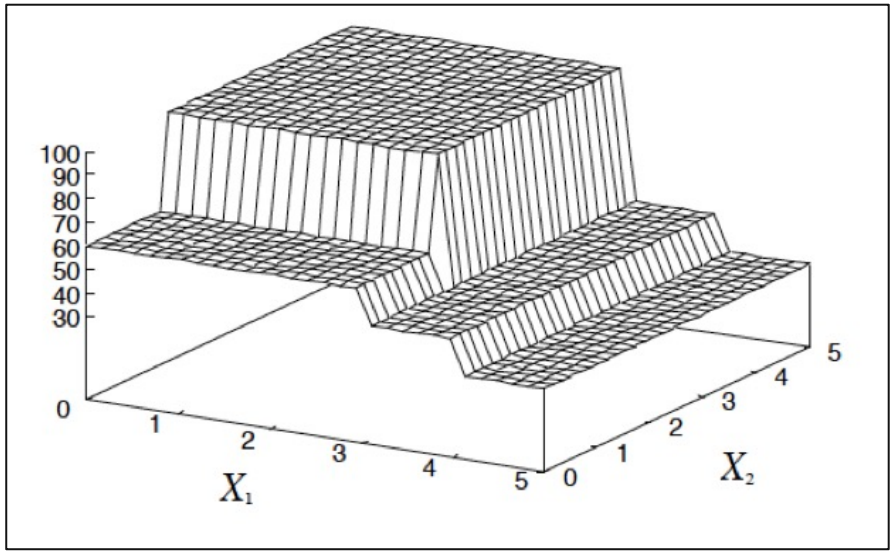
  $\hat{f}(X) = \Sigma_{j=1}^{p} \Sigma_{k=1}^{K(j)} \beta_{jk} B_{jkd}(X_j)$

- Number of knots and order define complexity of model

Regression Trees

- Regression tree partition the feature space into a set of rectangles and fit a simple model (e.g., constant) in each one.
- Advantages:
  - Fast, intuitive
  - Able to handle both numeric and categorical data
  - Robust to outliers in predictors
  - Model interaction and nonlinearity automatically (little data transformation)
- Disadvantage:
  - High bias for shallow trees, for example trying to model linear relationships
  - Unstable, high variance for deep trees. Small change in data can result in a completely different tree

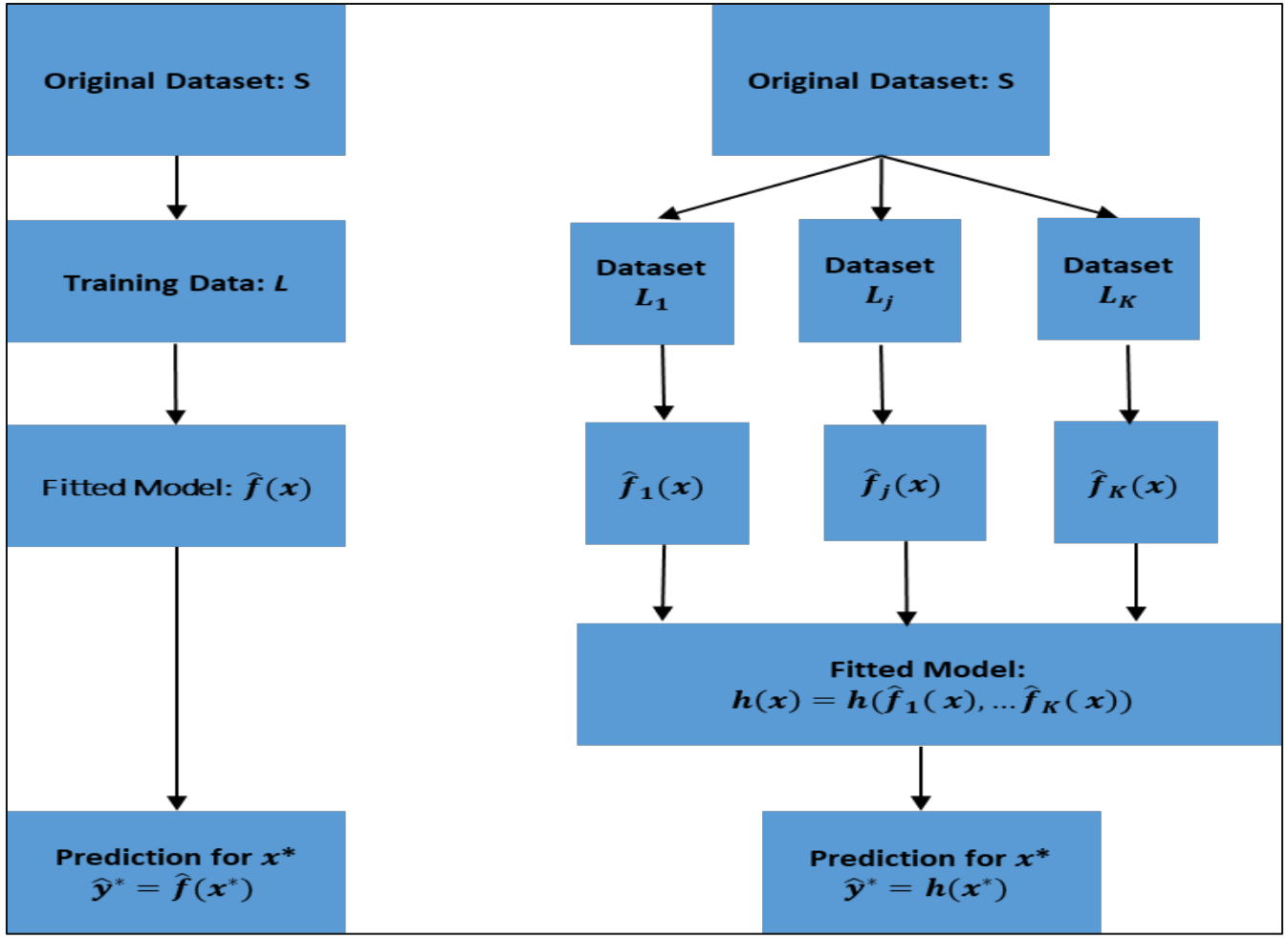




Outline

- ML algorithms
  - Tree based ensemble learners
    - Random Forest
    - Gradient Boosting
  - Feed Forward Neural Network
- Hyper-Parameter tuning
- Reproducibility
- Advanced neural network architecture

Ensemble algorithms

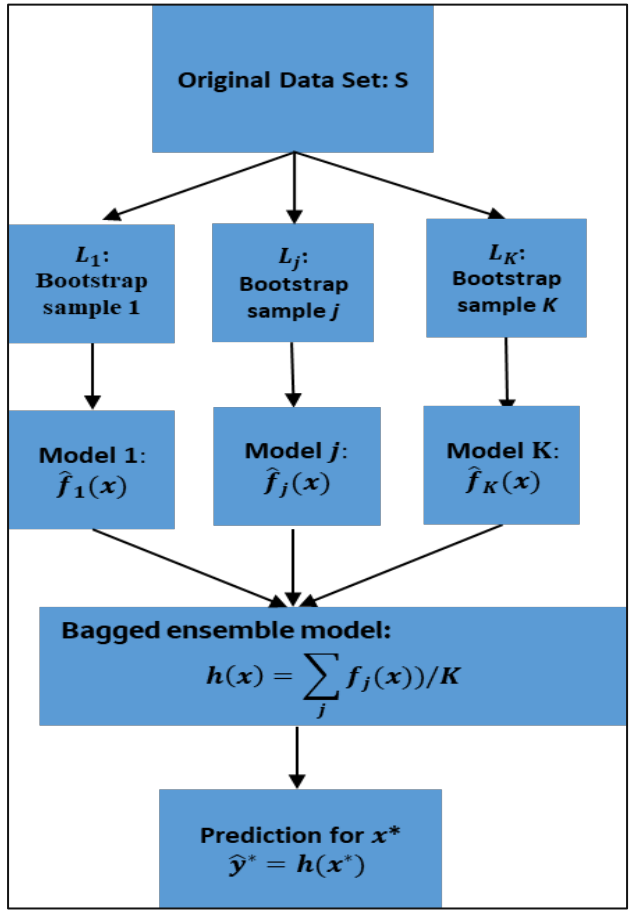


- Improve performance by combining the outputs of several individual predictors.
- Examples:
  - Bagging
  - Boosting
  - Model Averaging
  - Majority Voting
  - Ensemble Stacking
- Mostly bagging and boosting algorithms use tree-based learners.

Bagging

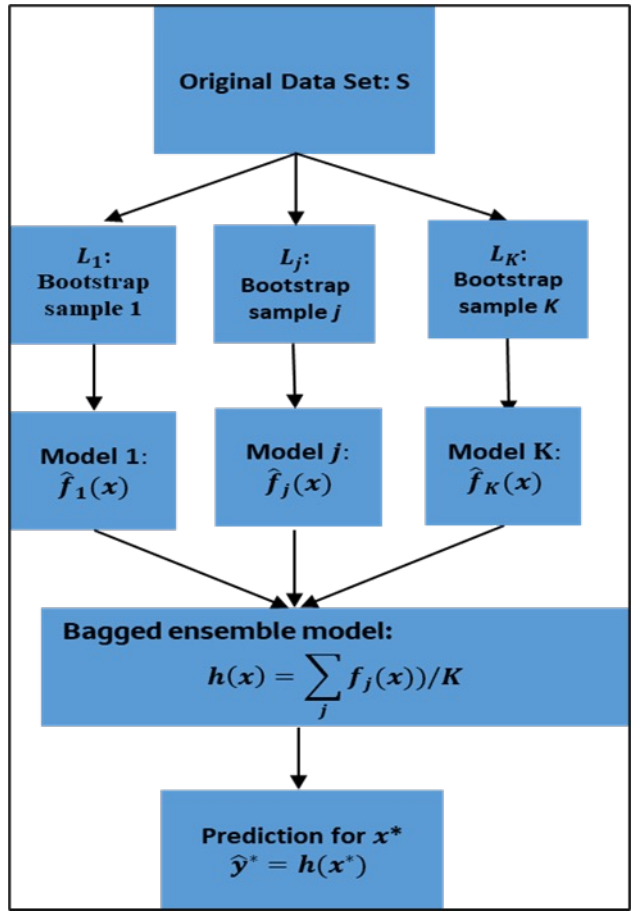
- Bagging: bootstrap aggregating (Breiman in 1994)
  - "improvements for unstable procedures" (deep decision tree)

- Algorithm:
  - bootstrap sample at each iteration $i, i = 1, 2, \ldots, n.$
  - Fit base learner to bootstrap sample → $\hat{f}_i(x)$
  - Combine base model predictions :
    - Averaging (Regression): $\hat{f}(x) = \frac{1}{n}\sum_i \hat{f}_i(x)$
    - Majority voting (Classification): $\hat{f}(x) = \arg\max_k \sum_i I\left(\hat{f}_i(x) == k\right)$
- Loss functions:
  - Loss functions for measuring homogeneity in leaves:
    - MSE (continuous)
    - Gini-Index/cross-entropy/logloss (binary/multi-class)
- Combining base model predictions reduces variance, making model stable
  - More base learners → better prediction, higher computational complexity
  - Base learners: low bias high variance



Random Forests

- **Random Forests** (Breiman 2001)
  - Bagging and random feature subsampling
- Deep Trees
  - Low bias, high variance
  - Reduce variance through bagging
  - A variant uses sample without replacement
- De-correlate trees
  - use random subset of features in each split instead of entire feature set
  - Tries to achieve maximum variance reduction
- Typically over-fits the data
- Typical Hyper-Parameters (HP) to be tuned (sklearn)
  - N_estimators : number of forests
  - Max_depth : maximum depth of the trees
  - Min_samples_leaf (MSL): the minimum samples required at each leaf
  - Max_features: number of features to consider at each split



Boosting

- Boosting is a different type of ensemble algorithm, based on removing bias of a simple learner.
- Given a simple learner, can you improve it to be a strong learner? (Kearns and Valiant 1988)
- Schapire (1989): Yes → by a technique called “boosting”
- Freund and Schapire (1995): AdaBoost for classification

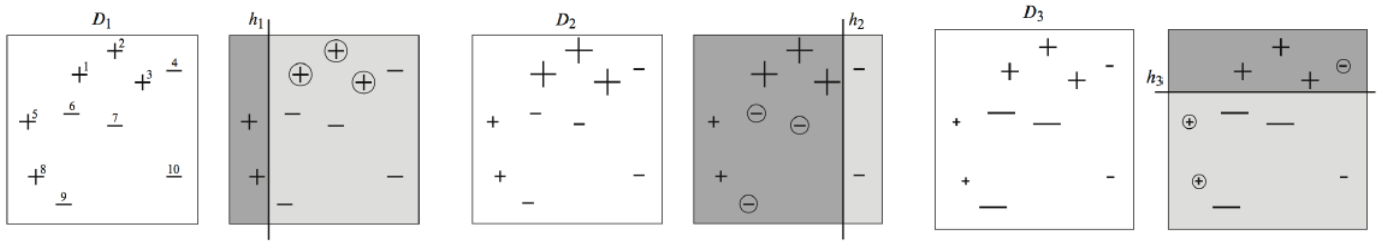
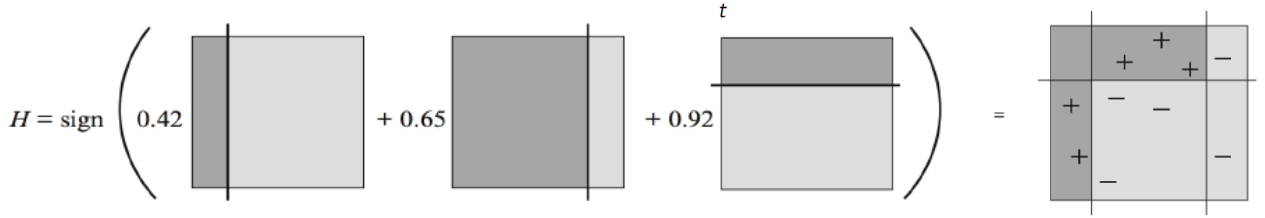


Figure : AdaBoost. Source: Figure 1.1 of [Schapire and Freund, 2012]

- “Base learner”: simple rectangular classification regions at each stage
- Reweighting at each stage – more weight to data that are misclassified
- Fit an additive model (ensemble)

$$H(x) = \sum_t \rho_t h_t(x)$$



Source: Figure 1.2 of [Schapire and Freund, 2012]

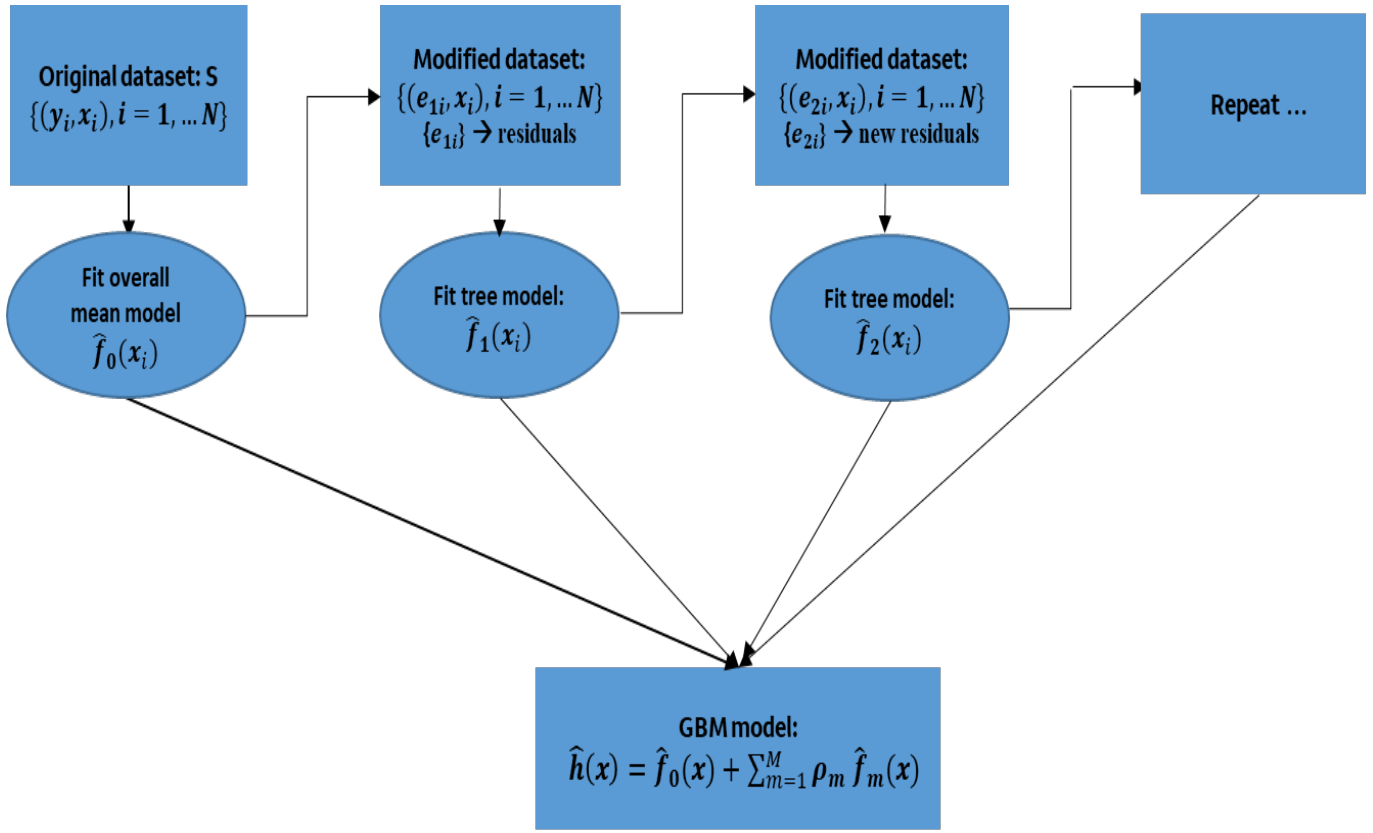
Gradient Boosting

- Breiman (1998+): Boosting → optimization algorithm
- Friedman (2000+): Extended concept to gradient boosting (gradient descent)
- Loss function to minimize: $L(y, f)$.
  - squared error loss $(y - f)^2$ for regression
  - deviance $yf - \log\left(1 + e^f\right)$ for binary classification (f is the log-odds) (deviance = logloss)
  - Other loss functions: absolute error loss, partial likelihood, etc
- Find the prediction function $f(x)$ that minimize the total loss $\sum_{i=1}^{N} L\left(y_i,\ f(x_i)\right)$.
  - $f(x)$ is optimized in an additive, stage-wise way:
  - $f(x) = T_0(x) + \sum_{m=1}^{M} \eta_m T_m(x)$, where $T_0(x)$ is baseline (e.g., overall mean in regression).
  - In each stage m, update $f(x)$ in the direction $T_m(x)$ where the total loss decreases, for a step size/learn rate of η_m.
  - Each base learner $T_m(x_i)$ is fit to the negative gradient (gradient descent) from the previous iteration
  - $T_m(x_i) = -\frac{\partial L(x_i, y_i)}{\partial\, \hat{f}_{m-1}(x_i)}$

Gradient Boosting

- Stochastic gradient boosting (Friedman 1999):
  - fit each tree with a subsample instead of the entire data.
  - can be more robust and lead to less overfitting.
- Hyper-Parameters (HP):
  - number of trees,
  - learn rate, tree
  - depth, …
- Implementation:
  - Scikit learn: GradientBoostingClassifier and GradientBoostingRegressor
  - R: gbm package
  - Spark: mllib library
  - H2o: h2o.gbm
- Popular variations
  - **XGBoost**
  - **LightGBM**
  - CatBoost

Illustration of GBM for continuous regression



XGBoost

- XGB = Extreme Gradient Boosting (Chen and Guestrin 2016) - a variant of GBM that uses second order Hessian for optimizing
- XGBoost adds a **penalty** term to the loss function

$$\sum_{i=1}^{N} L\big(y_i, f(x_i)\big) + \sum_{m=1}^{M} \Omega\big(T_m(x)\big)$$

  to control overfitting.
  - For example, $\Omega(T) = \gamma|T| + \frac{1}{2}\lambda\sum_j w_j^2$, where $|T|$ is the number of terminal nodes in the tree and w_j is the fit in terminal node j.
  - γ is a pruning parameter: any split with the improvement below γ is pruned.
  - λ acts like a shrinkage parameter: the prediction in each tree node is shrunk
  - $\mathrm{w_j} = -\frac{G_j}{H_j+\lambda}$, where $G_j = \sum_{i\in I_j} g_i$, $H_j = \sum_{i\in I_j} h_i$ are the total gradients and total **Hessian** in Node j.
  - An L1 penalty ($\alpha\sum_j |w_j|$) can be added as well.
  - The trees are built depth-wise.

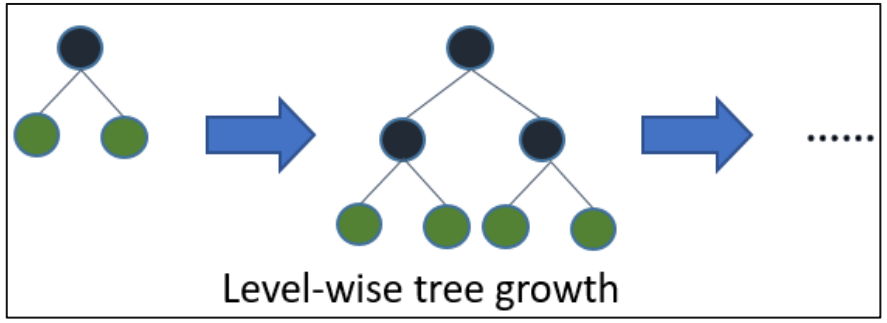
- XGBoost has several advantages:
  - Parallelized, so scalable.
  - Penalized, to reduce overfitting issue
- Originally implemented in C++ but it is available in Java, Python, R through APIs. Has Scikit-learn wrapper.
- Hyper-parameters (HP) :
  - max_depth, learning_rate (lr_rate),N_estimators (# trees), L1 (alpha),L2 (lambda), min_child_weight, …

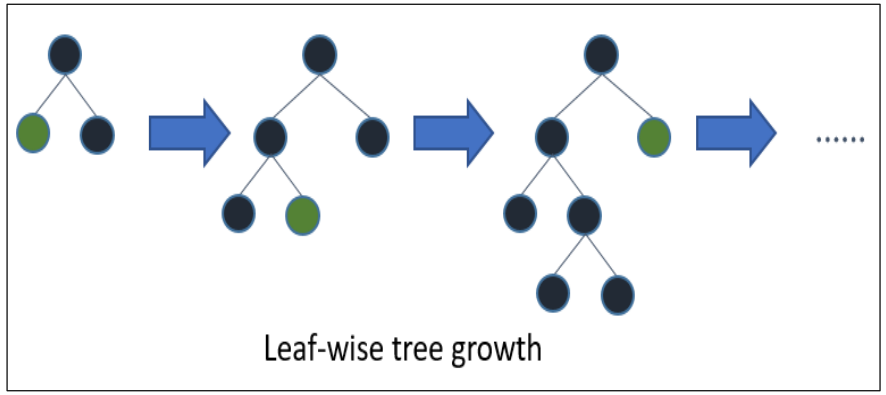
LightGBM

- Introduces two new concepts (Ke et al, 2017)
  - Gradient One-Sided sampling(GOSS)
    - At each iteration, keep all observations with large gradients and random subset on instances with smaller gradients.
  - Exclusive feature bundling (EFB)
    - Bundles up mutually exclusive features.
    - Helpful in feature reduction for high dimensional data with large number of 0-1 encoded columns.
- Builds trees leaf-wise
- Hyper-Parameters:
  - Lr_rate
  - Max_depth
  - Num_leaves
  - Min_data_in_leaf
- Developed by Microsoft.
- Built in C++, has python and R interface.

XGBoost vs LightGBM

- Numerous blogs on comparison
- Similar performance
- LightGBM is faster
- Default option in XGB: grows tree level-wise
- LightGBM grows trees leaf-wise
- Differences in the algorithm may lead to different feature importance and other diagnostics.





Neural Networks: history and inspiration from neuroscience

Wave 1: “Cybernetics”: [1940s-1960s]

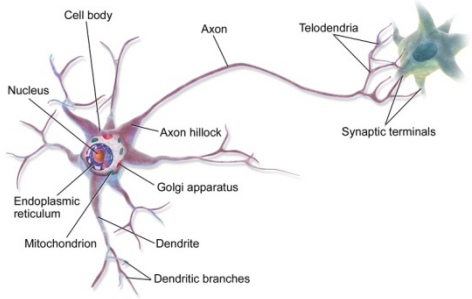
- Model of biological function of brain
- 1958 Rosenblatt: Developed “perceptron”, a training algorithm
- *Perceptrons* (Minsky and Papert): showed difficulties with approaches

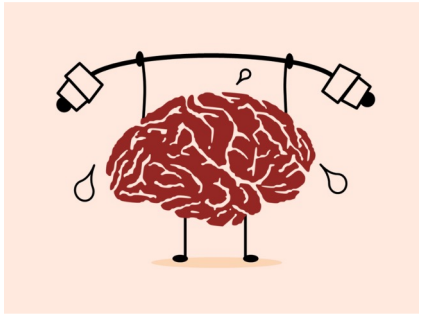
Wave 2: “Connectionism”: [1980s-1990s]

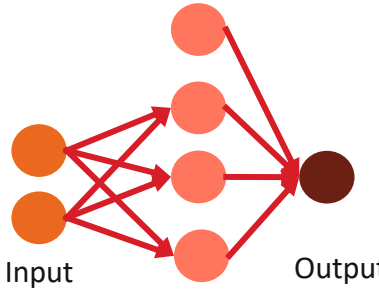
- From the study of how the brain can form connections
- Central Idea: A network of many simple units can learn complex patterns.
- The backpropagation algorithm provided an essential advance in training neural networks.

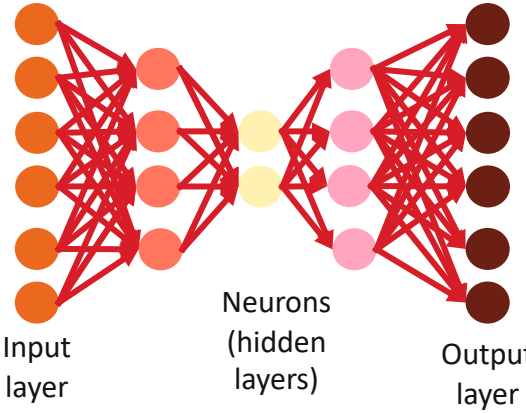
Third Wave [2006-present]: Deep Learning

- Ability to train neural networks with more layers.
- Able to outperform other ML systems:
  - Object recognition in pictures
  - Natural Language Processing



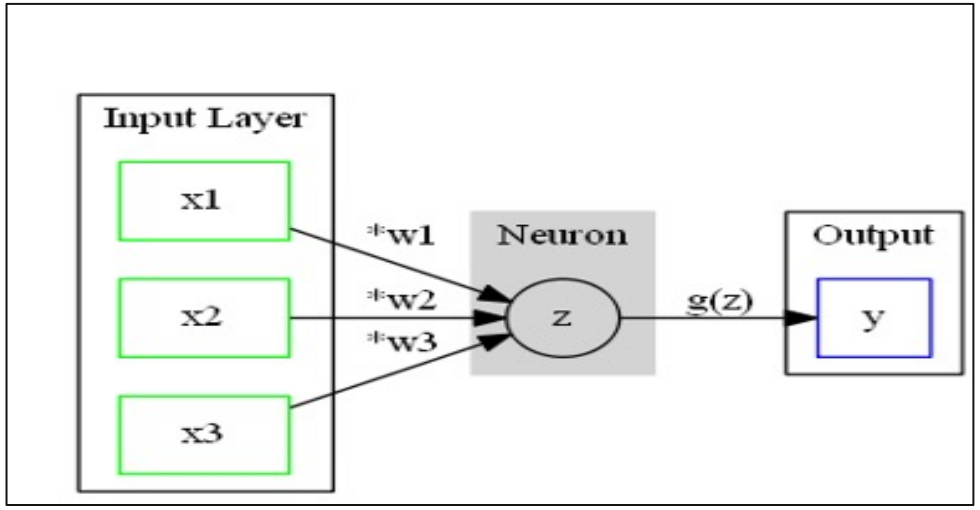


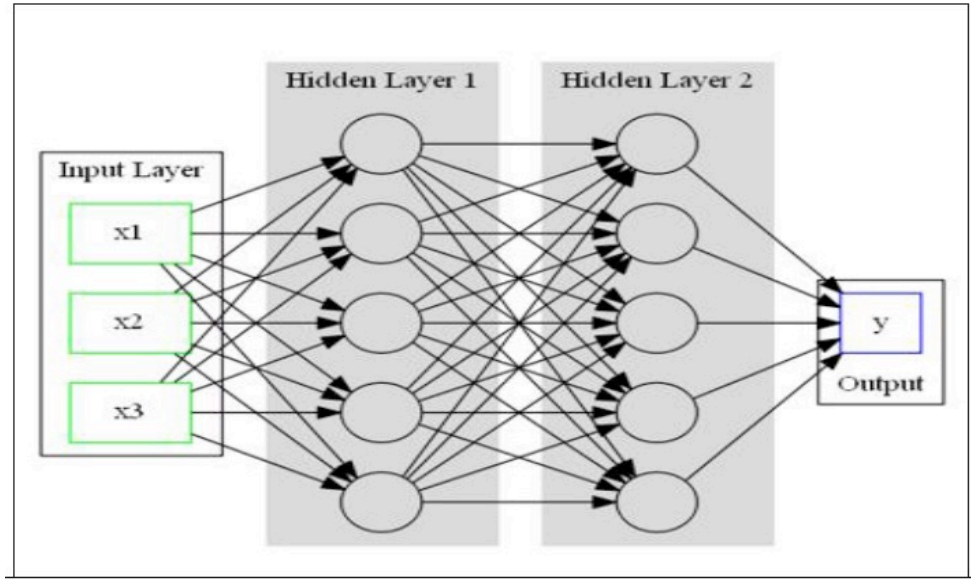




Feedforward Neural Networks (FFNN):

- Activation function: $g(w^T x)$
  - Sigmoidal, Hyperbolic Tan, ReLU
  - Connection to **additive index models:**
    $$f(\boldsymbol{x}) = g(w_1 x_1 + \ \ldots + w_P x_P)$$
- FFNN architecture
  - Nodes (Neurons)
  - Input, Output, and Hidden Layers
  - All nodes connected with others in next layer
- Hyper-Parameters(HP)
  - Learning rate
  - Number of layers
  - Neurons in each layer
  - L1, L2, dropout
  - batch-size
  - activation function…
- Implementation
  - Scikit-learn – MLP
  - Keras
  - PyTorch





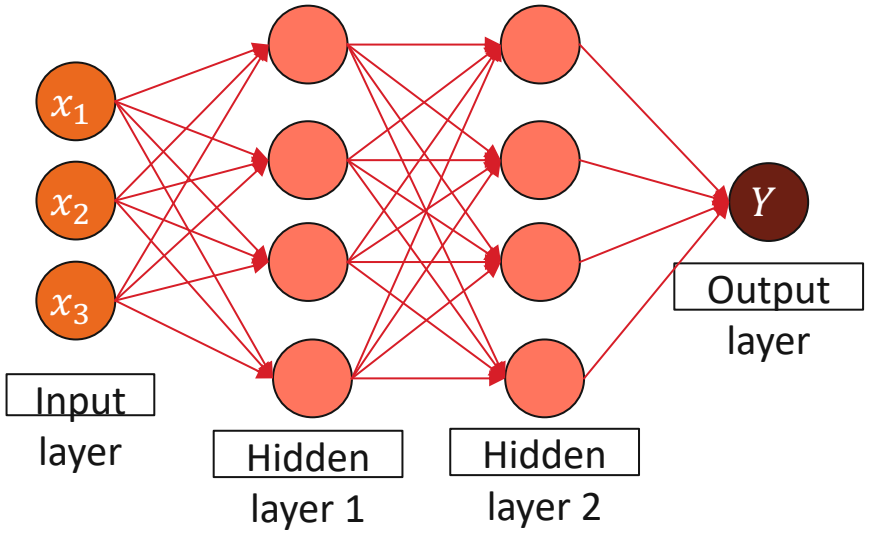
Fitting a Neural Network to Data: Learning the Weights

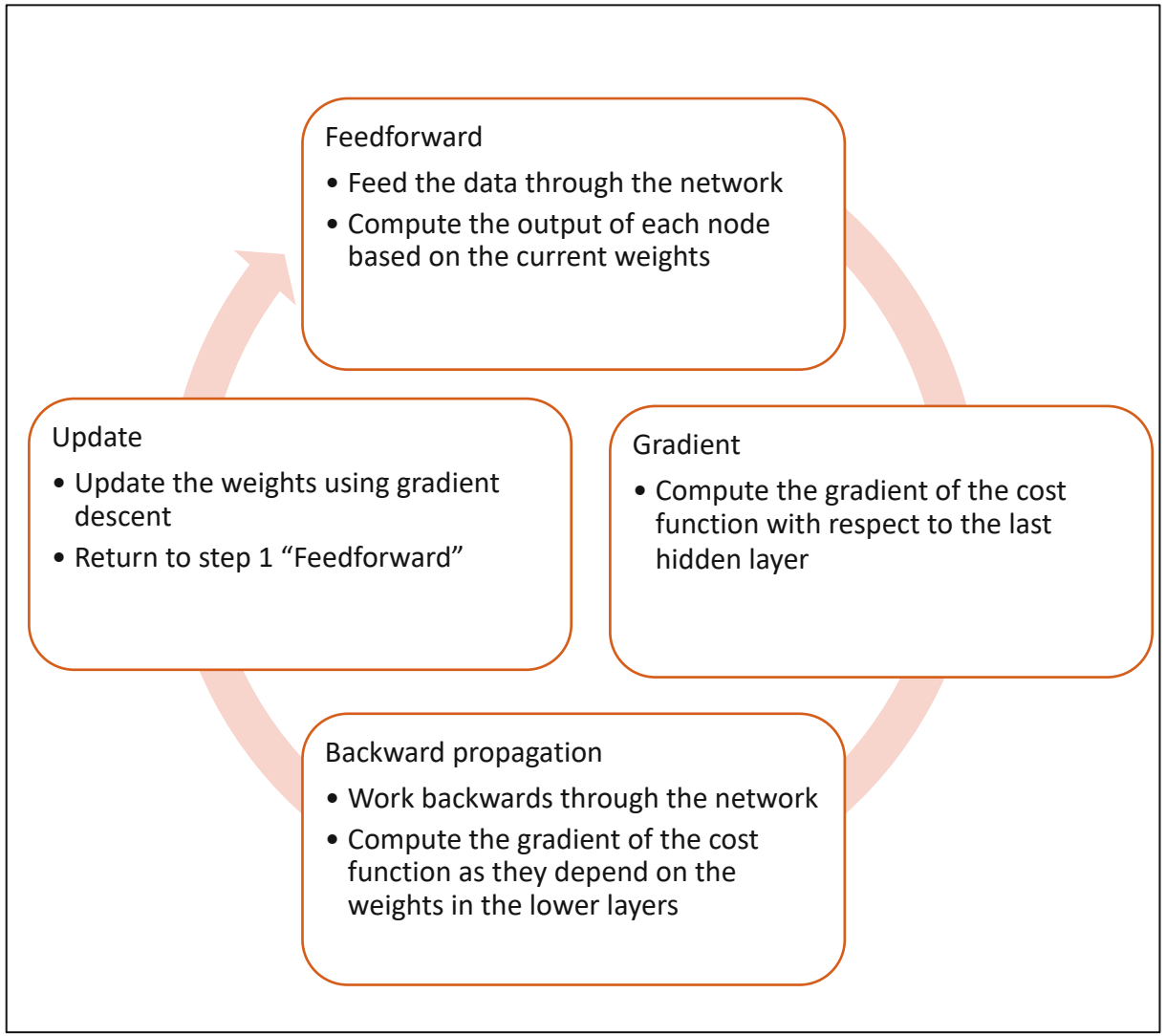
- The **weights** (and **bias**) of each neuron are unknown parameters in a NN
- They need to be "learned" from data.
- Need an appropriate cost (loss) function:
  - For continuous responses, typically use squared error loss: $\frac{1}{n}\sum_i (y_i - \hat{y}_i)^2$
  - For binary response, typically use cross entropy, or log loss: $-\frac{1}{n}\sum_i [y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i)]$

- Choose **weights** to minimize the cost (loss) function.

Optimization: Back Propagation Algorithm

- Gradient descent can be challenging in NNs due to computation of gradient.
- Initialization:
  - Input the data x
  - Initialize all weights in the network.
- There is a lot of research into optimization of NNs:
  - many other algorithms (Adam, SGD, etc)
  - mini-batch training



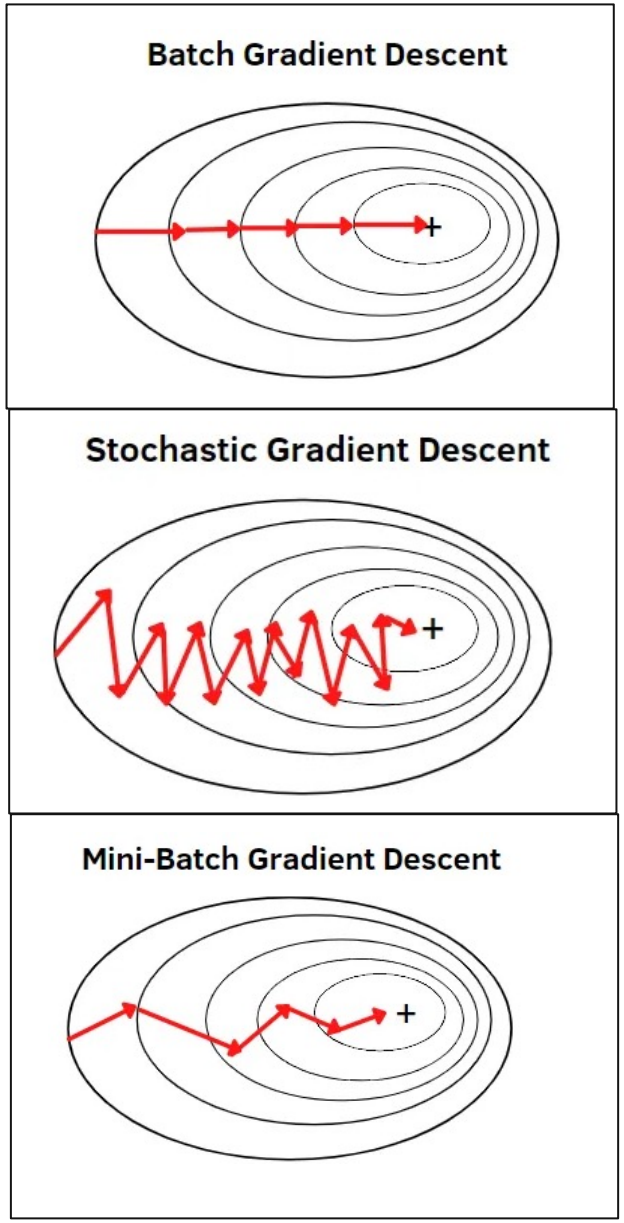


Different types of gradient descent

- Batch gradient descent
  - computes gradients of all samples at each iteration
- Stochastic gradient descent
  - computes gradient of a single random sample at each iteration
- Mini batch gradient descent
  - partitions data and computes gradient of one partition at each iteration

- Variations include
  - Adaptive learning rates
  - Adding momentum

- Different variations of gradient descent
  - Adagrad
  - RMSProp
  - Adam



Using NNs in Practice

General Usage

- Very flexible. Choices for:
  - Structure: Number of Hidden Layers, Number of nodes on Each Hidden Layer, Activation Functions for each Hidden Layer
  - Regularization Strategy/ Parameters
  - Additional Features: Skip connections, Batch Normalization, Dropout, Constraints
  - Training/Optimization Algorithms
- Can make grid search difficult

- Much of the literature available gives advice in the context of (images/text/speech)

- Not particularly relevant to tabular data

Training effectively

- Training can be challenging
  - Saddle points and local minima can result in a sub-optimal model.
- Tips:
  - Standardize or normalize data (X) before training. Avoids vanishing gradient problem.
    - Min/Max scaling
    - Gaussian standardization (perform better with large outliers)
  - Batch Normalize hidden layers.
  - optimization routine with **learning rate decay** (e.g., Adam).
  - **batch size** used in training - smaller batches can be slow and volatile, but help escape local minima/saddle points.
  - Use **early stopping** to determine number of training epochs.

Overfitting

- NNs are flexible models, with a (potentially) large number of parameters, therefore overfitting is a real concern.
- Strategies to avoid overfitting in include:
  - Multiple narrow layers vs. Single wide layer
  - Weight regularization: Penalizing large weights in the cost function.
  - Dropout: randomly set a fraction of neurons' activations to 0 during training. This forces redundancy of neurons, and reduces neuron specialization.
  - Early stopping via a validation set.

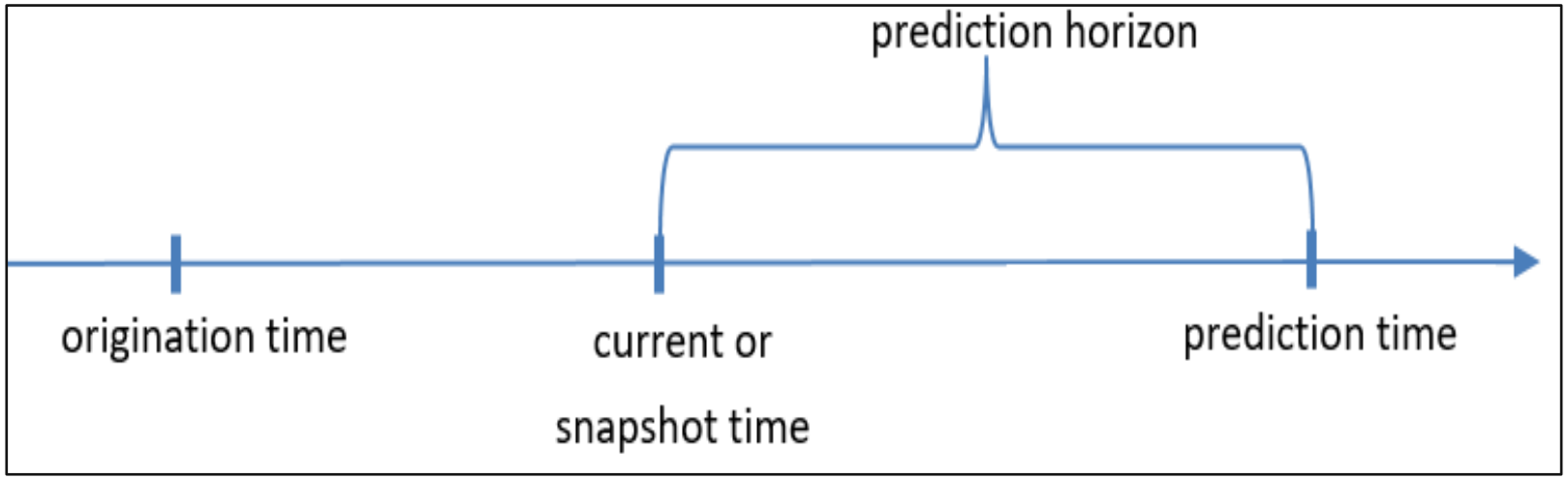
Comparison of ML Algorithms

	Preprocessing	Robustness to outliers in input space	Computational scalability (large N)	Predictive power	Smoothness of response surface	Feature engineering	Hyper parameter tuning
GBM	No, some require dummy coding	Yes, tree based methods are robust to outliers	Depend on software implementation. Xgb oost, LightGBM and H2O GBM are scalable	Good, often outperforms random forest and neural network in prediction	The response surface for tree based methods are often jumpy and not smooth, especially for small data	Good for manually created features; may not be good for raw features, e.g., transaction data, image data.	Relatively easy
Random Forest	No, some require dummy coding	Yes, tree-based methods are robust to outliers	Depend on software implementa tion. Scikit learn random forest, H2O random forest are scalable	Good	The response surface for tree-based methods are often jumpy and not smooth, especially for small data	Good for manually created features; may not be good for raw features, e.g., transaction data, image data.	Relatively easy
Neural Network	dummy coding and standardization	No	Large neural network with large data requires GPU. Computation with simple network or small data is scalable on CPU.	Good. Best for image, speech,…	Usually smooth	Powerful in feature engineering with a variety of deep neural network structures.	Complicated

Application to Home Mortgage: Modeling “In-Trouble” Loans

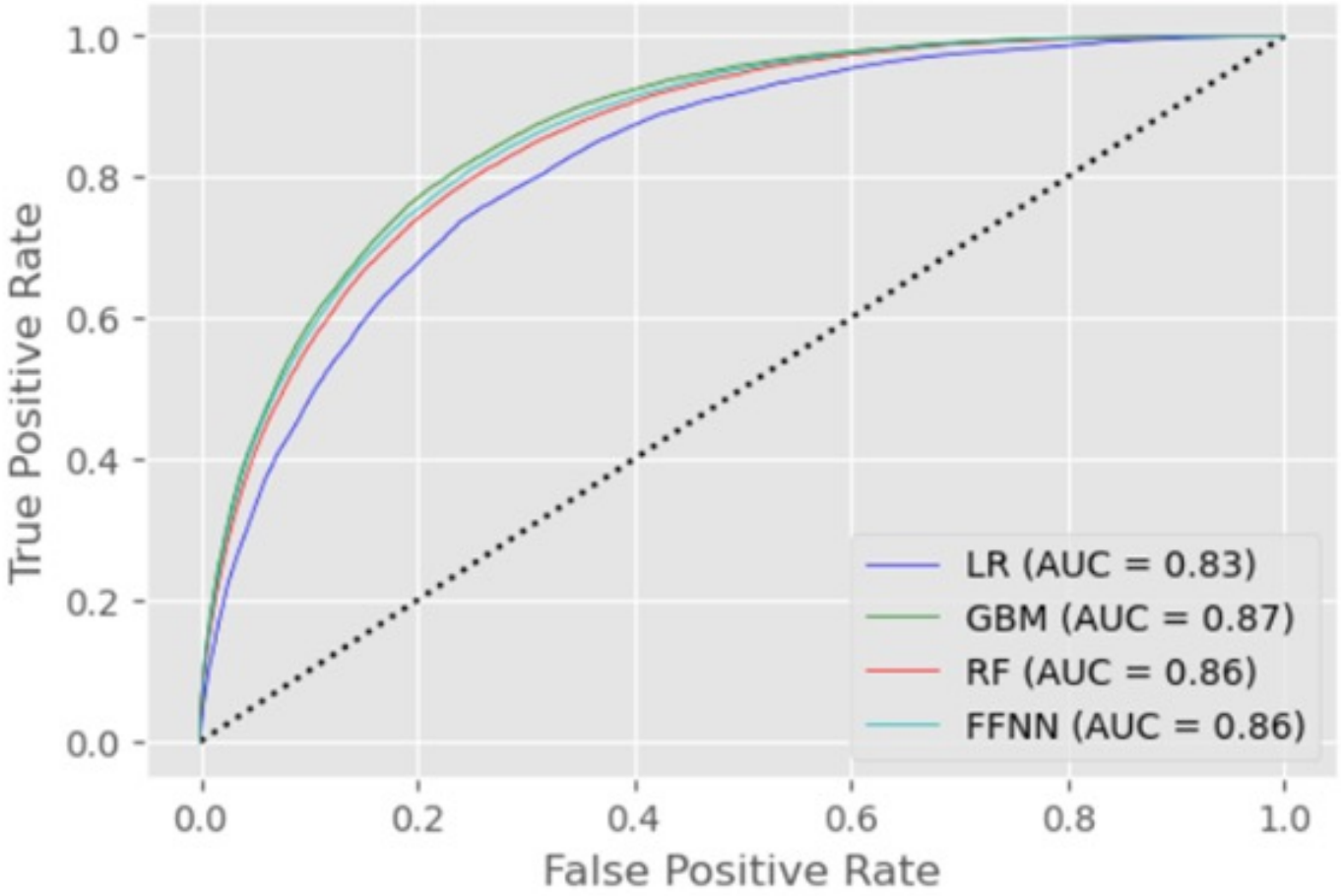
- One portfolio: ~ 5 million observations
- Response: binary = loan is “in trouble” (multiple failures and connections to competing risks)
- 20+ predictors: credit history, type of loan, loan amount, loan age, loan-to-value ratios, interest rates at origination and current, loan payments up-to-date, etc. (origination and over time)

Modeling framework



Loan origination, current (snapshot) and prediction times

Comparison of Predictive Performance: ROC and AUC on Test Data



- ML with 22 predictors
- LR model: eight carefully selected variables
  - snapshot fico (credit history);
  - ltv (loan-to-value ratio);
  - ind_financial-crisis;
  - pred_unemp_rate;
  - two delinquency status variables;
  - horizon

How typical is this "lift" in our applications?

Findings from internal study -XGB and FFNN are competitive and exhibit better model performance across a variety of functional forms than RF

https://arxiv.org/ftp/arxiv/papers/2204/2204.12868.pdf

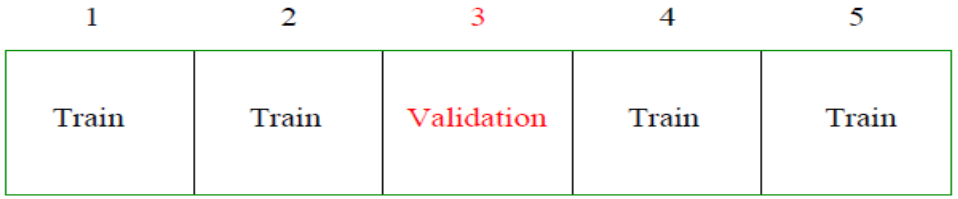
Hyper-Parameter Tuning

Tuning of Supervised Machine Learning Algorithms

- **Hyper-parameter tuning**
  - Find the best hyper-parameter configuration that gives the best results
- HPs are data dependent
  - They need to be tuned based on the particular dataset
- Tuning - **search** method and a **fitting-evaluation** method.
  - For each HP setting α, fit the model $\hat{f}(x;\alpha)$
  - Evaluate the model performance
  - Use a search method to search over the hyper-parameter space to find the hyper-parameter/model that optimizes performance
- Note: The model that minimizes the loss/error on the training data is likely to overfit.
- To avoid this, the performance is assessed on **a separate validation data** (when there is abundant data)
  - Split data into training, validation, and test datasets (example: 60:20:20)
- One uses **cross-validation** (when there is not enough data)

Evaluation Using Cross Validation

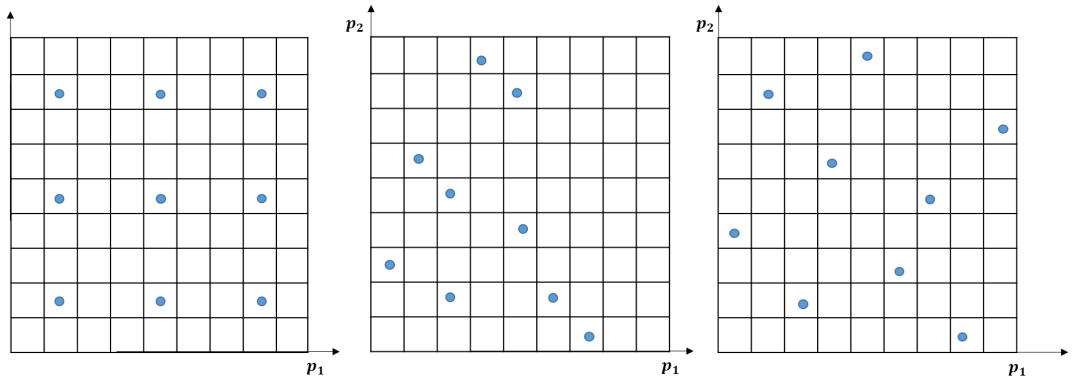
- Cross-validation. The typical **K-fold cross validation** works as follows:
  1. Randomly divide the data into K folds. (Stratification may be needed for imbalanced data)
  2. For each k = 1, …, K
     1. Leave the k-th fold out, build a model using the rest K-1 folds and given hyper parameter α, denote as $\hat{f}^{-k}(x;\alpha)$
     2. Predict on the k-th fold.
  3. After obtaining the cross-validation predictions for the entire data, compute the loss/error



- This is the cross-validation model performance
  - Sometime, people compute the performance for each k-th hold-out fold and compute and average
- Typical choices for K are 5 or 10.
- The case $K = N$ is known as **leave-one-out** cross validation.
  - With larger K, this becomes computationally expensive

Batch or non-sequential search methods

- **Grid search** and **random search :** two most widely-used batch (non-sequential) parameter tuning methods.
- **Grid search** - specify a set of grid points for each parameter and try all combinations in the parameter grid space. It is simple but not efficient:
  - The number of HP combinations increases exponentially; yet many parameters may not matter much.
  - On any single HP dimension, you only have a handful of points, this is risky.
- **Random search**, we randomly sample hyper-parameters from each parameter space.
  - "*Random Search for Hyper-Parameter Optimization*" by Bergstra and Bengio: random search is more efficient than grid search
- **Space filling designs:** latin hypercube design, orthogonal arrays, etc.

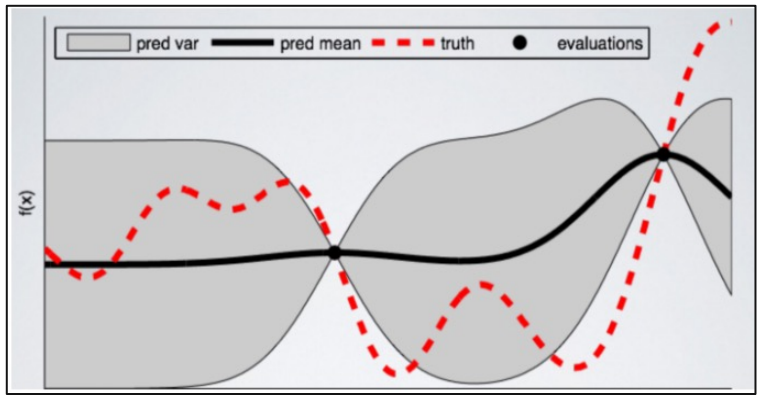


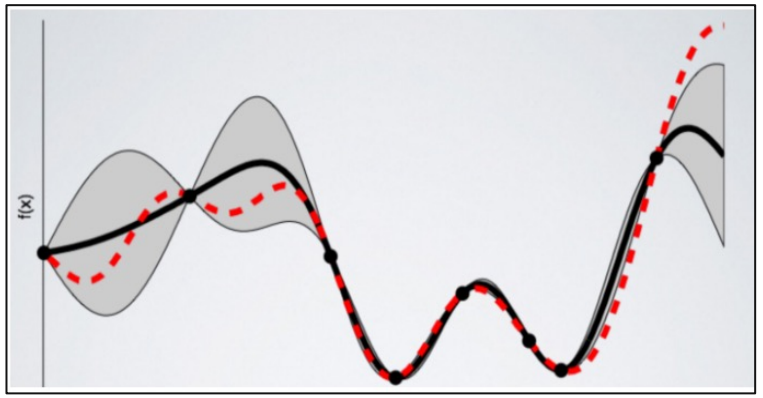
Grid search Random search Latin Hyper-cube search

- Grid search is implemented in GridSearchCV in sklearn in python.
- Random search is implemented in RandomizedSearchCV in sklearn in python
- GPyOpt – space filling designs

Sequential search methods: Bayesian optimization

- Grid and random search are not informed by past evaluations.
- Both approaches may waste time searching area that is unpromising based on past evaluations.
- Bayesian parameter optimization takes past evaluations into consideration.
- By evaluating hyperparameters that appear more promising from past results, Bayesian methods can find better model settings than random search in fewer iterations.
    1. Using past evaluation results, build a surrogate probability model $\hat{P}(score|parameter)$.
    2. Find the hyperparameters that perform best on the surrogate model
    3. Apply these hyperparameters to build the model and evaluate the scores
    4. Update the surrogate model incorporating the new results
    5. Repeat steps 2–4 until max iterations or time is reached.





Surrogate model with 2 evaluations (left) and 8 evaluations (right)

Summary of hyper-parameter search methods

- There are other sequential search methods like TPE (Tree-structured Parzen Estimators), Hyperband, etc.
- All sequential algorithms exploit previous information to do intelligent searches and hence are more efficient than batch techniques.
- The primary advantage of the batch methods is their simplicity.
- There are several open source software packages available for implementing the algorithms discussed above:
  - **Spearmint:** https://github.com/HIPS/Spearmint
  - **RoBO:** http://automl.github.io/RoBO/
  - **GPyOpt:** https://github.com/SheffieldML/GPyOpt
  - **GPflowOpt:** https://github.com/GPflow/GPflowOpt
  - **Hyperband:** https://github.com/zygmuntz/hyperband

Neural Networks – Other Architectures

Complex NNs

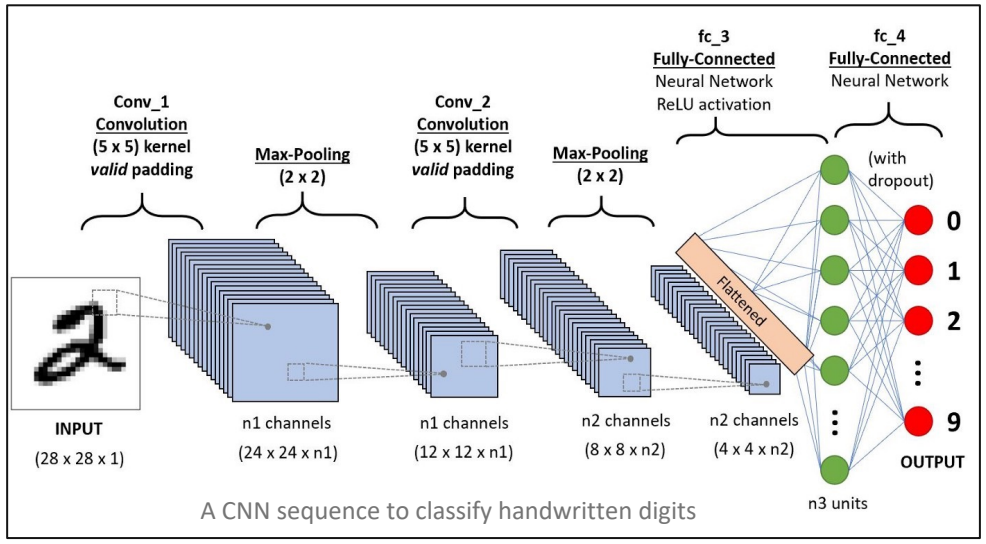
Convolutional NN

Used in images, text, time series

- Key Features:
  - Convolutional layers, where inputs are convolved with their neighbors.
  - Each output is a weighted average of the inputs:

$$\sum_{i,j} a_{i,j}\, x_{i,j}$$

  - The **weights** remain **constant** as the convolution is applied to successive windows of data.
  - Other tricks, like pooling

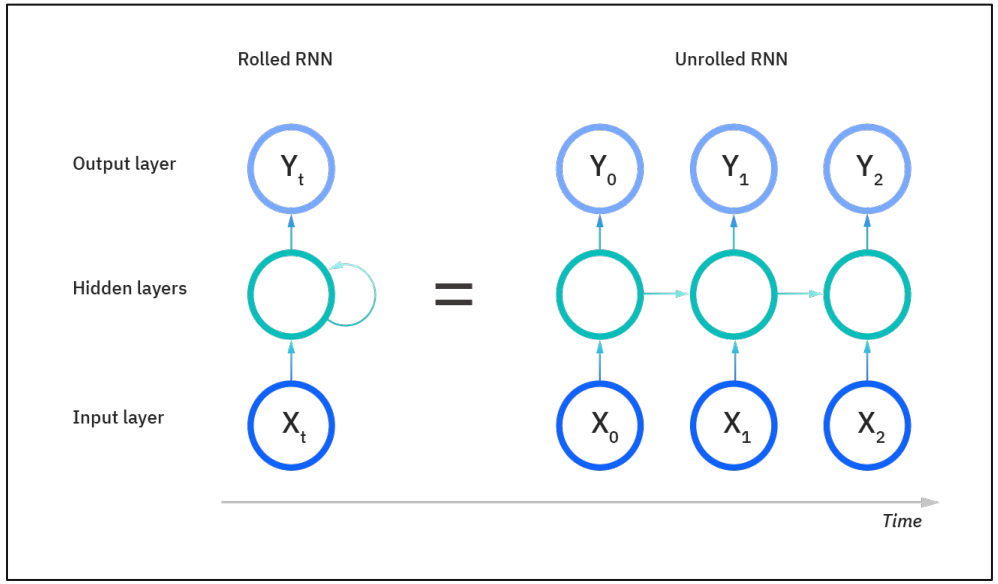


A CNN sequence to classify handwritten digits

a-comprehensive-guide-to-convolutional-neural-networks

Recurrent NNs

- Useful in studying sequences, such as in natural language context.
- Defined by "recurrent" connections, where the output of a downstream unit serves as input to an upstream neuron.
- Several variations; "Long Short-Term Memory" (LTSM) was popular
- Now focus is on Attention Networks.

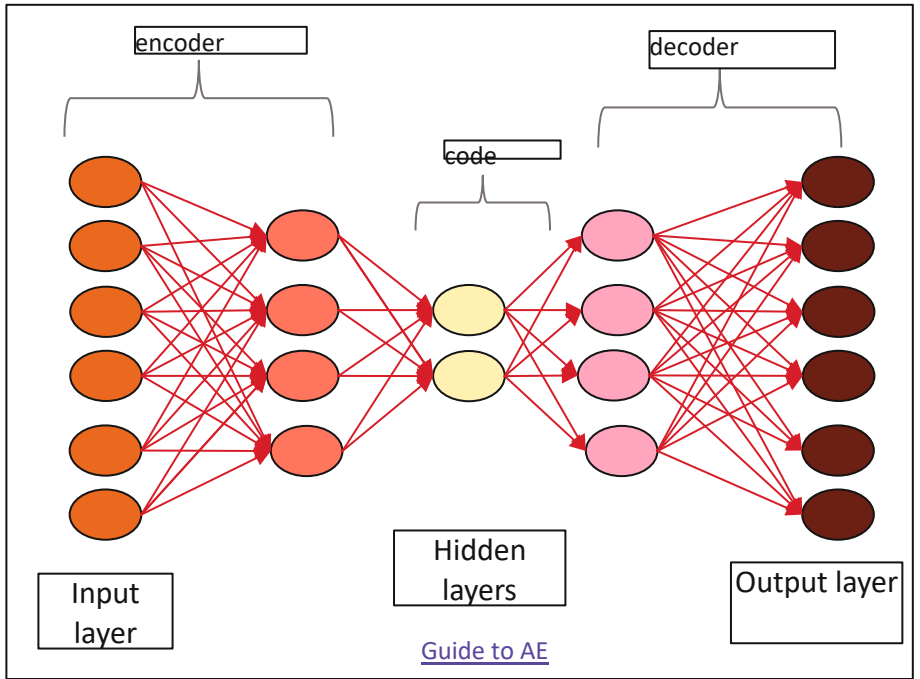


https://www.ibm.com/cloud/learn/recurrent-neural-networks

Complex NNs (contd.)

Auto-Encoders

- Dimension Reduction Network
- Predicts input from input.
- Bottleneck layer engineers lower-dimensional features.
- Different types
  - De-noising AE
  - Sparse AE
  - Variational AE …

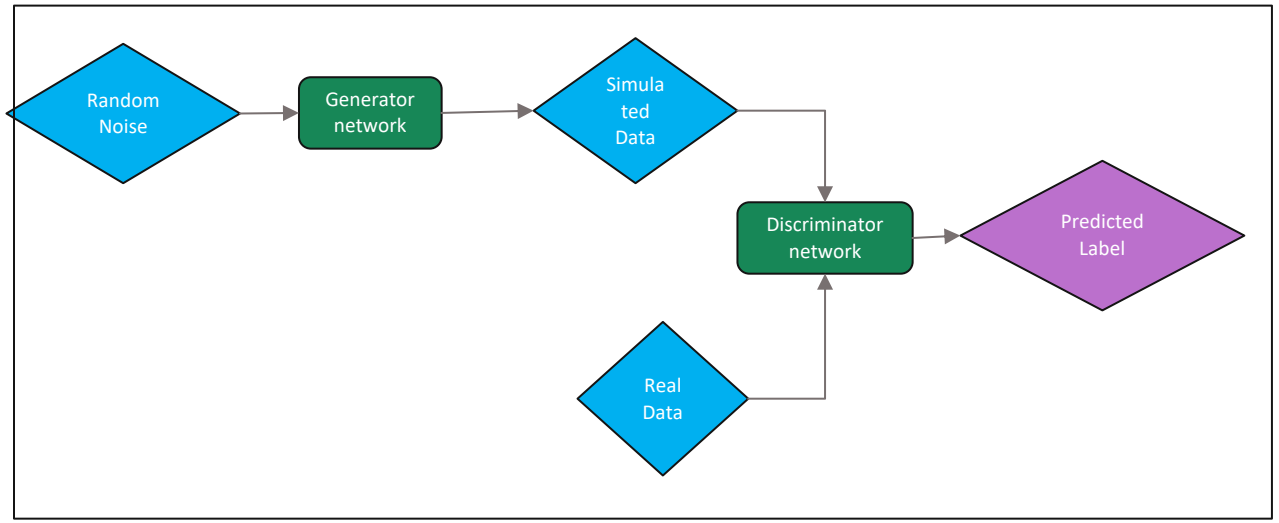


Guide to AE

Generative Adversarial Networks (GANs)

Unsupervised technique

- Pair of ANNs, trained with simultaneous backpropagation
- A Generator Network, which produces candidate data examples
- A Discriminator Network, which learns to distinguish the generated data from the real data. (Classification)
- Simultaneous training improves the performance of both networks.



Advanced Architectures and Ongoing Research

- Attention Networks
  - Added mechanisms to model dependencies across data regardless of positions.
  - Originally proposed as a component of RNN structures; now used independently.
  - Reference “Attention Is All You Need” https://arxiv.org/pdf/1706.03762.pdf
- Temporal Fusion Transformers
  - Use a combination of RNN and Attention structures to predict multi-horizon time series
  - Provides some measure of interpretability into temporal patterns
  - Reference: “Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting” https://arxiv.org/pdf/1912.09363.pdf
- Other exciting work, both in the bank and in the broader research community

Software

- Specialized software packages exploit computational graphs, symbolic differentiation, back propagation and mini-batch training.
- Popular Packages include:
  - **TensorFlow /Keras**(Google)
  - **PyTorch**/Torch
  - Caffe/Caffe2
  - CNTK (Cognitive Toolkit) (Microsoft)
  - Theano

- Wrappers exist to provide abstraction layers (Keras)
- Many general-purpose scientific packages have more limited implementations (R, MatLAB, scikit-learn)
- Many offer distributed or flexible computing
- NVidia providing growing support for GPU computation

Summary

- We have discussed several machine learning algorithms
  - Focus was on supervised machine learning methods: random forest, GBM and FFNN
- Random forests and GBM are tree-based ensemble methods
  - They were designed to improve the performance of a single tree using a collective set of trees.
- Neural network is a biologically inspired method designed to mimic the function of brain
- In structured data XGB and FFNN have competitive performances and both outperform RF
- Advantages of ML algorithms
  - Useful in large datasets where we can fit more flexible nonparametric models
  - Better predictive performance than traditional statistical techniques
  - (Semi-)automated approaches to variable selection and feature transformation → useful with large datasets
- Disadvantages:
  - Computationally intensive
    - Need access to fast computing environment for model training and hyper-parameter tuning
  - Model is a "black box" → no analytical expression for fitted model $\hat{f}(\boldsymbol{x})$
  - Goal is more than just prediction → must be able interpret results and explain to stakeholders
  - Model is flexible, so it can overfit → must assess robustness
  - Fitted model depends on various random elements and have to quantify these sources of variation

Reference

- Breiman, L. (1996). Bagging predictors. Machine Learning, 24 (2): 123–140.
- Quinlan, J. R. (1986). Induction of decision trees. Machine Learning, 1(1):81-106.
- Breiman, L.; Friedman, J. H.; Olshen, R. A.; Stone, C. J. (1984). Classification and regression trees. Monterey, CA: Wadsworth & Brooks/Cole Advanced Books & Software.
- Friedman, J. H (2001). Greedy Function Approximation: A Gradient Boosting Machine. The Annals of Statistics, 29 (5): 1189-1232.
- Friedman, J. H. (1999). *Stochastic gradient boosting.* Stanford University.
- Hastie, T.; Tibshirani, R.; Friedman, J. H. (2001). *The elements of statistical learning : Data mining, inference, and prediction.* New York: Springer Verlag.
- Breiman, Leo (2001). Random Forests. Machine Learning, **45** (1)
- Deep Learning, by I. Goodfellow, Y. Bengio, and A. Courville, available: http://www.deeplearningbook.org/
- Explained: Neural networks, by L. Hardesty, available: http://news.mit.edu/2017/explained-neural-networks-deep-learning-0414
- M Nielsen (2017), Neural Network and Deep Learning online book: http://neuralnetworksanddeeplearning.com/index.html
- Goldstein, A., Kapelner, A., Bleich, J., & Pitkin, E. (2013). Peeking Inside the Black Box: Visualizing Statistical Learning with Plots of Individual Conditional Expectation. *eprint arXiv:1309.6392*.
- Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). Why should I trust you?: Explaining the predictions of any classifier. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, (pp. 1135-1144)
- Hu, L et. al (2022). Surrogate locally-interpretable models with supervised machine learning algorithms, Journal of Indian Statistical Association.
- Chen, J. et al (2020). Adaptive Explainable Neural Networks, arXiv:2004.02352
- Vaughan, J. et al (2018). Explainable Neural Networks based on Additive Index Models. The RMA Journal