AI for Business: Insights from Corporate Data

Topic 6: Tree-based Models and Neural Networks

Miao Liu

Boston College

February 18, 2026

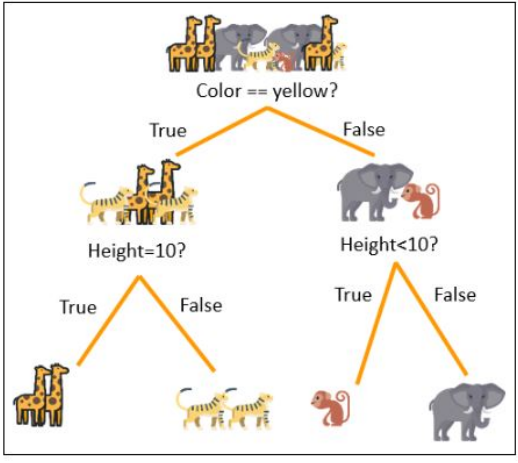
Overview: Topic 6

1. Tree-based Models
2. Neural Network and Deep Learning
3. General ML Workflow

Overview: Topic 6

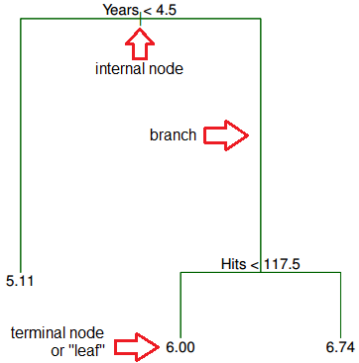
Decision Trees

- They segment the predictor space into distinct, non-overlapping regions using splitting rules



Regression Trees

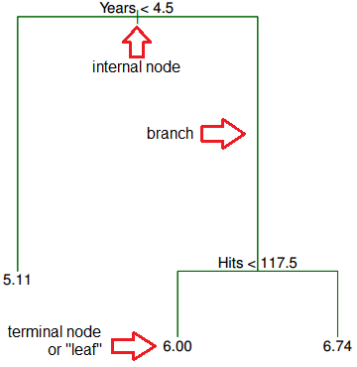
- For regression, decision trees predict the mean response in each terminal node
- Example: Predicting baseball salaries using years played (*Years*) and hits (*Hits*)

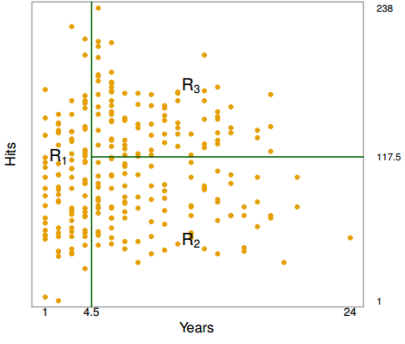


- But how do we know which predictor to use at each node and the threshold value?

Recursive Binary Splitting

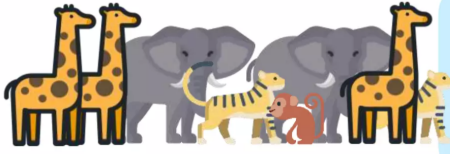
- At each node, search for the single predictor and a threshold (e.g.Years = 4.5) that best separates data to minimize the variance of Salary within resulting regions
- Repeat recursively to create smaller, more homogeneous groups where Salary values are as close as possible to their region's mean





Classification Trees

- For classification, decision trees predict most frequent class in each terminal node

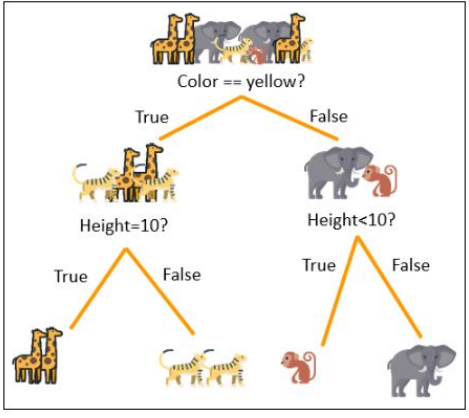


Training Dataset

Color	Height	Label
grey	10	elephant
Yellow	10	giraffe
brown	3	Monkey
grey	10	elephant
Yellow	4	Tiger

Classification Trees

- At each node: Search for the single predictor and a threshold (Color=yellow) that best splits the data to minimize classification error within the resulting regions
- Repeat recursively to create smaller, more homogeneous groups where data within each region predominantly belong to a single class



Advantages and Disadvantages of Decision Trees

Advantages

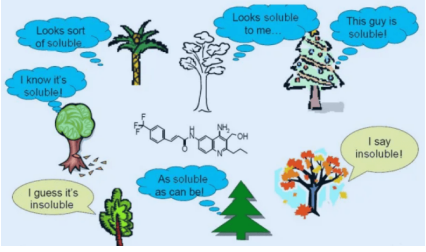
- **Interpretability:** Decision trees provide clear and intuitive decision-making processes that can be easily understood by non-experts
- **Visualization:** Trees can be visualized effectively, offering insights into the model's decision-making process and feature importance

Disadvantages

- **Overfitting:** Can easily **overfit** the training data, especially when the tree becomes too deep, leading to poor generalization on unseen data (high variance)
- **Greedy Splitting:** The splitting process is **greedy**, optimizing for local improvements at each node, which can result in suboptimal global solutions

Introduction to Ensemble Methods

- Ensemble methods combine multiple "weak learners" to create more robust model
- **Weak learners:** Simple models, like decision trees, that perform moderately well on their own



- Common ensemble techniques:
  - Bagging (Bootstrap Aggregation)
  - Random Forests
  - Boosting

Bagging (Bootstrap Aggregation)

- Bagging reduces variance by averaging predictions from multiple models
- Given models $\hat{f}_1(x), \hat{f}_2(x), \ldots, \hat{f}_B(x)$, the bagged prediction is:

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_b(x)$$

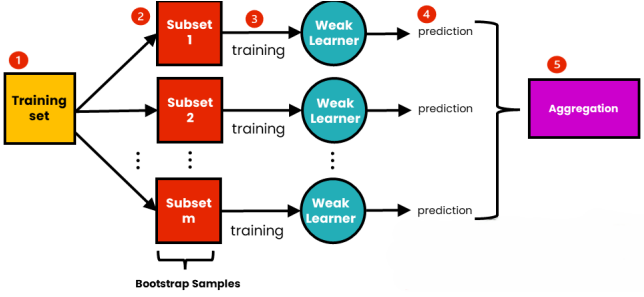
- Particularly effective for high-variance models like decision trees
- Each model is trained on a bootstrapped dataset (sampled with replacement)

Bootstrapping

- **Bootstrapping:** Create multiple datasets by sampling with replacement from the original training data
- Each bootstrap sample contains $\sim \frac{2}{3}$ of the original data, with duplicates
- Used to train individual models $\hat{f}_1(x), \hat{f}_2(x), \ldots, \hat{f}_B(x)$ in bagging

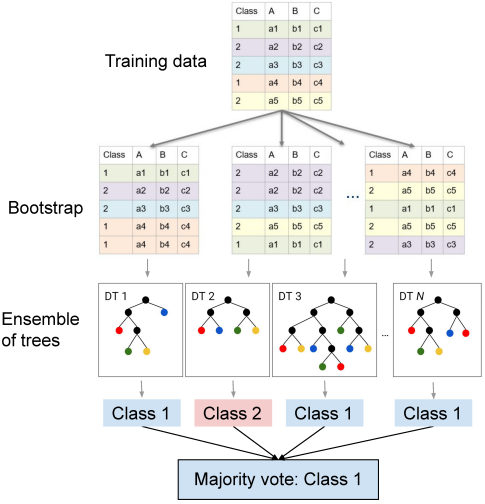
Bagging

- Steps:
  1. Create m bootstrapped datasets from training data
  2. Train a decision tree on each dataset
  3. Aggregate predictions:
     - Regression: Average predictions
     - Classification: Majority vote



Bagging: Why it works

- Intuition: Bagging smooths out individual weak model's prediction errors



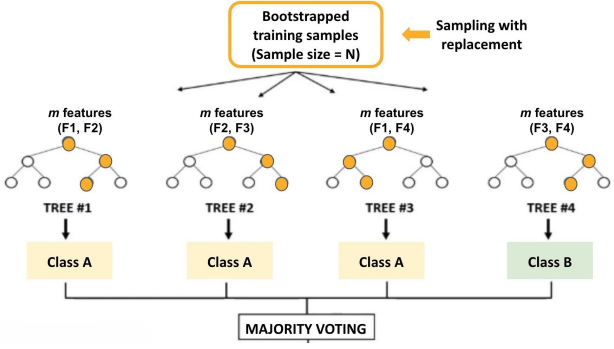
Introduction to Random Forests

- **Bagging** reduces variance but can produce similar trees, as strong predictors dominate splits
- Many similar trees cannot smooth out errors among individual trees

- **Random forests** improve on bagging by adding one feature: decorrelating trees
- At each split:
- Consider a random subset of predictors m (m $\approx \sqrt{p}$, $p =$ total # of predictors)
- Choose the best split only among these predictors
- Particularly useful for datasets with many correlated predictors

Random Forest Algorithm

1. Create B bootstrapped datasets from training data
2. Train a tree on each dataset **by randomly selecting m predictors at each split**
3. Aggregate predictions:
   - Regression: Average predictions
   - Classification: Majority vote

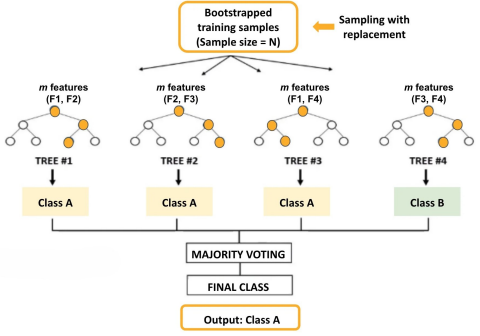


Random Forest Hyper-parameters

- Num of Trees; Max Depth; Random Feature Subset Size
- Can also consider Min num of samples required to split a node; Min num of samples required at a leaf node

Training Data
(Sample size, N= 6, No. of features, F = 4)

F1	F2	F3	F4	Y
2.1	0	400	-9	A
3.0	1	890	-42	B
2.2	1	929	0	B
4.0	0	324	-23	A
3.5	1	333	-15	A
6.0	0	215	-9	A

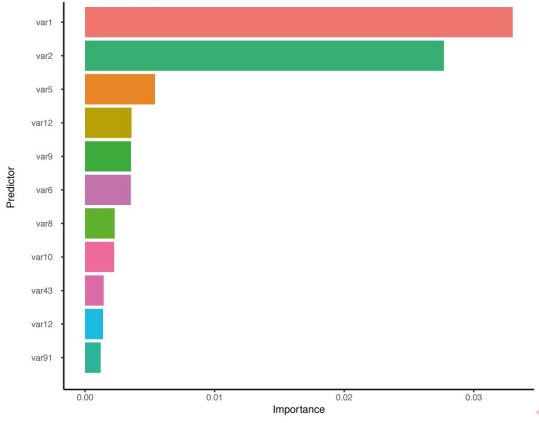


Key parameters of Random Forest Model are: (a) Number of trees , (b) Maximum depth of the trees (c) Size of the random subset of features
In this example, No. of trees = 4, Depth = 2, and Feature subset size, m = 2 (no. of features/2)

- **How do we choose hyper-parameters?**

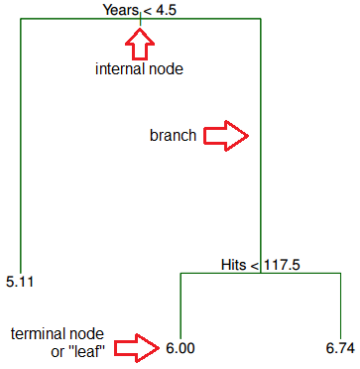
Variable Importance

- Bagging and Random Forest provides variable importance metrics:
- How much each feature contributes to decreasing the prediction error across all trees in the forest
- Visualized as a variable importance plot



Why does Random Forest work?

- **Captures Nonlinear Patterns**: like how baseball salaries increase with years played but plateau after a point
- **Handles Interactions Automatically**: identify feature interactions, such as how hits impact salaries more for experienced players than rookies



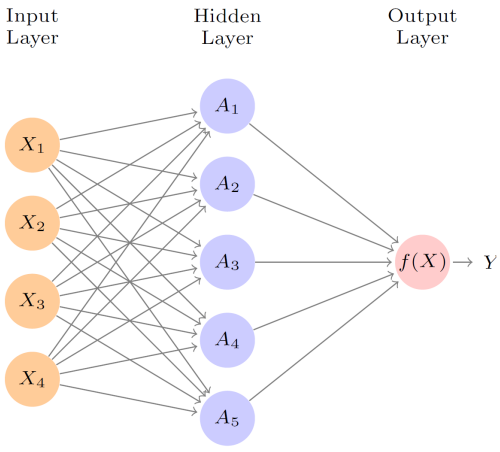
Overview: Topic 6

Introduction

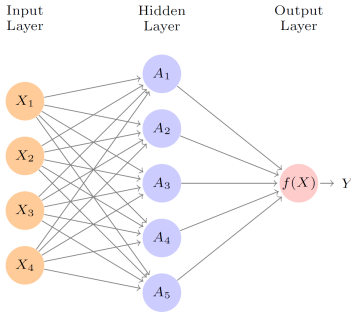
- Deep learning is a highly active research area in AI
- The cornerstone of deep learning is the neural network
- Neural networks rose to prominence in the late 1980s but lost favor until their resurgence after 2010
- Key factors for resurgence:
  - Increased computational power
  - Larger datasets due to digitization
  - Improved algorithms and architectures

What is a Neural Network?

- A neural network consists of Input layer, One or more hidden layers, Output layer



What is a Neural Network?



Key Takeaways:

- Nonlinear activation $g(\cdot)$ enables learning of complex relationships
- Weights w_{kj} and biases w_{k0} adjust during training to capture interactions
- Neural networks can model complex functions and interactions automatically

Neural Network Formulation:

$$f(X) = \beta_0 + \sum_{k=1}^{K} \beta_k A_k$$

(Learned parameters: β_0, β_k)

$$A_k = h_k(X) = g\left(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j\right)$$

(Learned parameters: w_{k0}, w_{kj})

ZX

Why Neural Networks Work I: Flexible Function Approximation

- Neural networks approximate $f(\cdot)$ as:

$$f(X) = \beta_0 + \sum_{k=1}^{K} \beta_k A_k$$

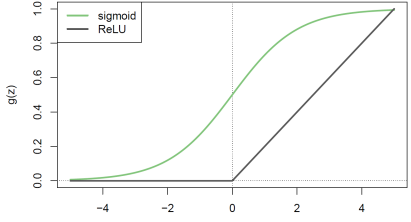
- Each activation A_k is a nonlinear transformation of the input:

$$A_k = h_k(X) = g\left(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j\right)$$

- The activation function $g(\cdot)$ enables approximation of complex $f(\cdot)$

Activation Functions:

- ReLU: $g(z) = \max(0, z)$
- Sigmoid: $g(z) = \frac{1}{1+e^{-z}}$



Why Neural Networks Work II: Automatic Feature & Interaction Learning

Traditional Model (Linear Regression)

- Predicts using: $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2$
- Cannot capture interactions unless we manually add $X_1 X_2$:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 (X_1 X_2)$$

Neural Network

- Hidden layers automatically learn interactions:

$$A_1 = g(w_{10} + w_{11} X_1 + w_{12} X_2), \quad A_2 = g(w_{20} + w_{21} X_1 + w_{22} X_2)$$

- Output combines these learned features:

$$Y = \beta_0 + \beta_1 A_1 + \beta_2 A_2$$

- No need to manually specify $X_1 X_2$—the network discovers interactions on its own

Example

Model:

$$A_1 = g(w_{10} + w_{11}X_1 + w_{12}X_2), \quad A_2 = g(w_{20} + w_{21}X_1 + w_{22}X_2)$$

$$Y = \beta_0 + \beta_1 A_1 + \beta_2 A_2$$

Consider specific weights:

$$w_{11} = 1, \quad w_{12} = 1, \quad w_{10} = -1$$
$$w_{21} = -1, \quad w_{22} = 2, \quad w_{20} = 0.5$$

Now, the activations are:

$$A_1 = g(-1 + X_1 + X_2) = \max(0, -1 + X_1 + X_2)$$
$$A_2 = g(0.5 - X_1 + 2X_2) = \max(0, 0.5 - X_1 + 2X_2)$$

Then, plugging these into Y:

$$Y = \beta_0 + \beta_1 \max(0, -1 + X_1 + X_2) + \beta_2 \max(0, 0.5 - X_1 + 2X_2)$$

Example

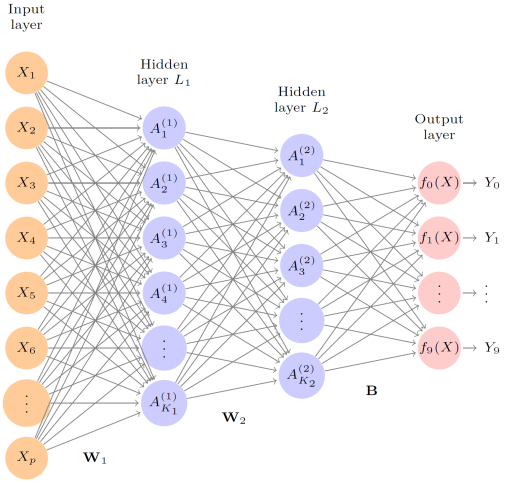
- How does this model capture nonlinearity and interaction?

$$Y = \beta_0 + \beta_1 \max(0, -1 + X_1 + X_2) + \beta_2 \max(0, 0.5 - X_1 + 2X_2)$$

- The ReLU function introduces nonlinearity, as activations can be zero for certain values of X_1 and X_2
- The interaction effect arises because the activation of each term depends on both X_1 and X_2. This means the relationship between X_1 and X_2 in Y changes dynamically based on which terms are activated, capturing cross-term effects that a simple linear model cannot
- This power becomes stronger when we have more neurons in the hidden layer and when we have multiple hidden layers

Towards Deep Learning: Multilayer Neural Networks

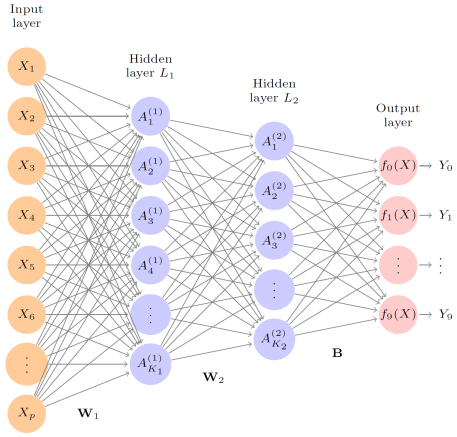
- Modern networks have multiple hidden layers for complex representations



Multilayer Neural Networks: Hidden Layers and Deep Representations

- By stacking multiple layers:

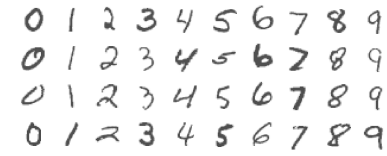
$$A_k^{(l)} = g\left(w_{k0}^{(l)} + \sum_{j=1}^{p} w_{kj}^{(l)} A_j^{(l-1)}\right)$$



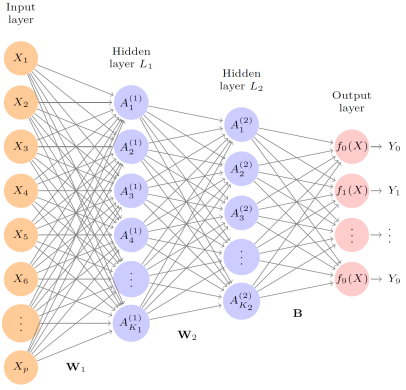
- Each layer extracts higher-order interactions and more abstract features, making deep networks more expressive than shallow models

Case Study: MNIST Dataset

- Dataset: 60,000 training images, 10,000 test images
- Features: 784 pixels per image (28x28 grid), each is an eight-bit grayscale value between 0 and 255
- Goal: build a model to classify the images into their correct digit class 0–9



MNIST: Neural Network Architecture



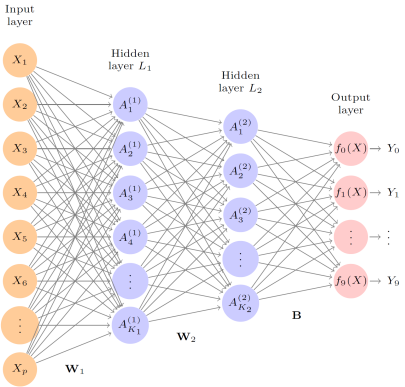
- Input layer has $p = 784$ units
- First hidden layer has $K_1 = 256$ units
- Second hidden layer has $K_2 = 128$ units
- The output layer has 10 units
- The first hidden layer computes:

$$A_k^{(1)} = g\left(w_k^{(1)} + \sum_{j=1}^{p} w_{kj}^{(1)} X_j\right), \quad k = 1, \ldots, K_1$$

- The second hidden layer computes:

$$A_k^{(2)} = g\left(w_k^{(2)} + \sum_{j=1}^{K_1} w_{kj}^{(2)} A_j^{(1)}\right), \quad k = 1, \ldots, K_2$$

MNIST: Neural Network Architecture



- Input layer has $p = 784$ units
- First hidden layer has $K_1 = 256$ units
- Second hidden layer has $K_2 = 128$ units
- Matrix W_1 has $785 \times 256 = 200,960$ parameters
- W_2 has $257 \times 128 = 32,896$ parameters
- B has $129 \times 10 = 1,290$ parameters
- Total of 235,146 parameters
- More than the 60,000 training images

- Large number of parameters allows the model to capture complex patterns
- Risk of overfitting with a small dataset; regularization techniques important

Fitting a Neural Network

Fitting neural networks is complex, but good software can automate much of the process
Here's a brief overview of the basic concepts:

- Given observations (x_i, y_i), $i = 1, \ldots, n$, the model is fit by solving:

$$\min_{\{\beta_k\}_{k=0}^{K}, \{w_k\}_{k=1}^{K}} \sum_{i=1}^{n} (y_i - f(x_i))^2, \quad \text{where} \quad f(x_i) = \beta_0 + \sum_{k=1}^{K} \beta_k g\left(w_{k0} + \sum_{j=1}^{p} w_{kj} x_{ij}\right).$$

Strategies to Overcome Overfitting

Three strategies help in fitting neural networks and preventing overfitting:

- **Regularization I**: Apply penalties on the parameters (e.g., Lasso or Ridge) Summing all parameters into a long vector θ, the objective function becomes:

$$J(\theta) = \sum_{i=1}^{n} (y_i - f_\theta(x_i))^2 + \lambda \sum_j \theta_j^2$$

- **Regularization II**: **Dropout Learning** is a regularization technique where a fraction of neurons are randomly dropped during training
- **Slow Learning**: Fit the model iteratively using **Gradient Descent** and stop when overfitting is detected

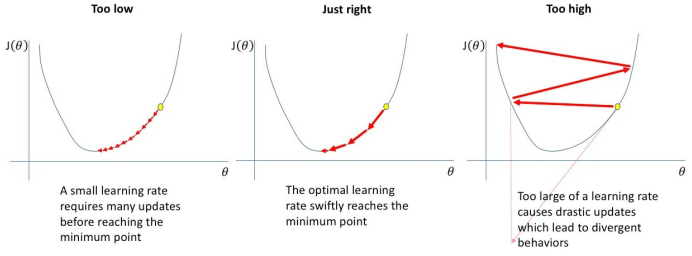
Gradient Descent

Gradient descent is used to minimize the objective function $J(\theta)$:

- Start with an initial guess θ_0 for the parameters.
- Iterate until the objective stops decreasing:

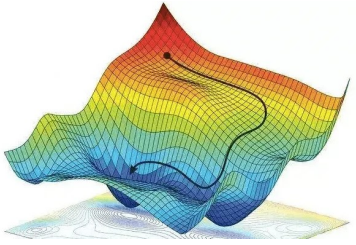
$$\theta_{t+1} = \theta_t - \alpha \nabla J(\theta_t).$$

- α is the learning rate, and $\nabla J(\theta_t)$ is the gradient of $J(\theta)$ at θ_t.



Gradient Descent: the Art of Slow Learning

- A slower learning rate keeps weight updates small and gradual, preventing the model from overfitting to noise or training-specific details



- Intuition: "Taking Small Steps to Avoid Wrong Turns" – Imagine walking down a mountain in the fog. If you take big steps (high learning rate), you might accidentally step into a ditch (overfitting to noise). If you take small, careful steps (low learning rate), you can find a smooth and steady path (better generalization)
- The model is less sensitive to small fluctuations in the training data and more focused on learning robust patterns

Alternatives to Gradient Descent

- **Computational Efficiency:** Processing the entire dataset at each update can be computationally expensive, especially with large datasets. SGD and mini-batch GD significantly reduce the computation per iteration

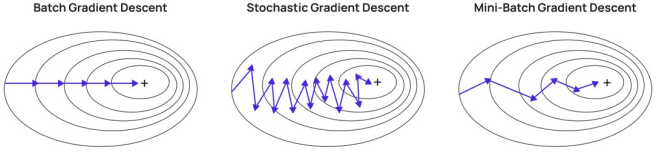
Stochastic Gradient Descent (SGD)

SGD minimizes the objective function $J(\theta)$ by updating the parameters using a *single* training example at a time:

- Start with an initial guess θ_0 for the parameters
- Iterate until the objective stops decreasing:

$$\theta_{t+1} = \theta_t - \alpha \nabla J\big(\theta_t; x^{(i)}, y^{(i)}\big).$$

- α is the learning rate, and $\nabla J\big(\theta_t; x^{(i)}, y^{(i)}\big)$ is the gradient computed on a single training example $(x^{(i)}, y^{(i)})$.



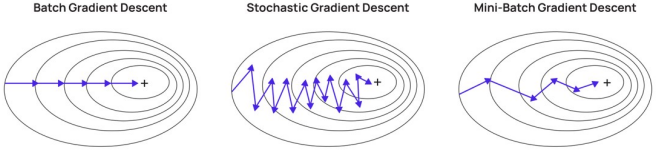
Mini-Batch Gradient Descent

Mini-Batch GD minimizes the objective function $J(\theta)$ by updating the parameters using a small subset (mini-batch) of the training data:

- Start with an initial guess θ_0 for the parameters.
- Iterate until the objective stops decreasing:

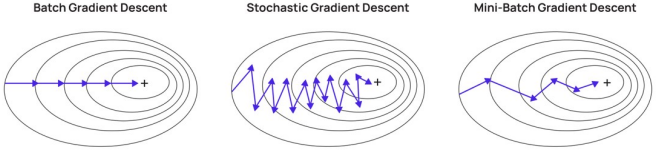
$$\theta_{t+1} = \theta_t - \alpha \nabla J\Big(\theta_t; \{x^{(j)}, y^{(j)}\}_{j=1}^{m}\Big).$$

- α is the learning rate, m is the mini-batch size, and $\nabla J\Big(\theta_t; \{x^{(j)}, y^{(j)}\}_{j=1}^{m}\Big)$ is the gradient computed on the mini-batch $\{(x^{(j)}, y^{(j)})\}_{j=1}^{m}$.



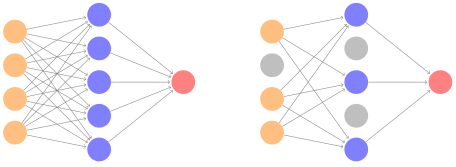
Alternatives to Gradient Descent

- **Faster Convergence in Practice:** Frequent updates in SGD and mini-batch GD allow the algorithm to start making progress before processing the entire dataset, which can lead to faster overall convergence
- **Improved Generalization:** The inherent noise in SGD and the balance of noise and stability in mini-batch GD can help the optimization process escape shallow local minima and potentially improve the model's generalization performance



Dropout Learning

A regularization technique where a fraction ϕ of neurons are randomly dropped during training



How It Works:

- During training, each neuron is independently dropped with probability ϕ
- At testing, all neurons are used

Dropout vs. Random Forest: A Similar Spirit

- **Reducing Overfitting by Bringing more Independence through Randomness**
  - **Random Forest**: Prevents overfitting by decorrelating individual trees, ensuring no single feature dominates
  - **Dropout**: Prevents co-adaptation of neurons, where specific neurons rely on each other's outputs rather than learning individually useful features

- Imagine studying in a group where you always rely on the same set of people for answers. If random members are removed each session, you're forced to learn more independently.

Summary of Neural Network and Deep Learning

Network Architecture

- Neurons, Layers, Activation Function, and why networks capture non-linearity, interactions, and complex patterns in data
- The resulting large number of parameters

Fitting a Network

- GD, SGD, and Mini-batch GD

Regularization

- L1 and L2 panelty
- Dropout Learning
- Slow Learning through GD

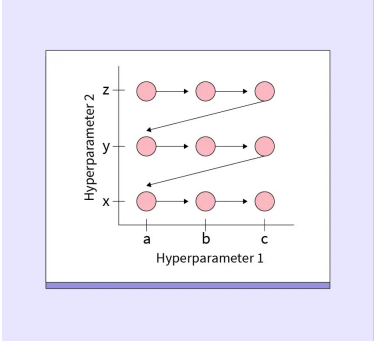
Hyper-parameters

- number of neurons in each layer, learning rate, L1 or L2 panelty (λ), dropout rate

Overview: Topic 6

What is Grid Search?

- Grid Search is an exhaustive search over a manually specified set of hyperparameters - a technique for hyperparameter tuning
- It systematically trains models for all possible parameter combinations and evaluate these models through cross-validation



Grid Search Procedure

1. Define a set of hyperparameters and their possible values.
2. Train and evaluate models for every combination using cross-validation.
3. Select the combination yielding the best performance.

Example: Random Forest Grid Search

- Number of trees: `n_estimators` $= \{50, 100, 200\}$
- Maximum depth: `max_depth` $= \{5, 10, \text{None}\}$
- Minimum samples per split: `min_samples_split` $= \{2, 5, 10\}$

Total combinations: $3 \times 3 \times 3 = 27$ models trained and evaluated.

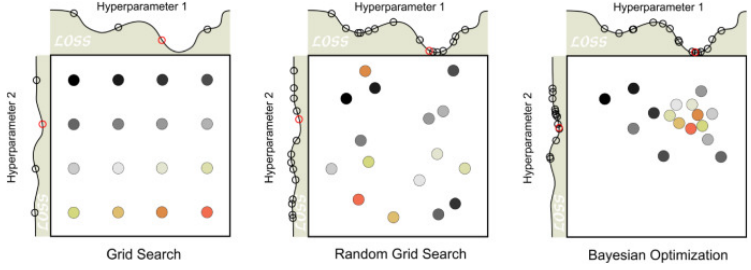
Why Look Beyond Grid Search?

- Computationally expensive when searching over many hyperparameters
- Some hyperparameters contribute more to performance than others
- Alternative methods can be more efficient and adaptive

Alternative 1 – Random Search

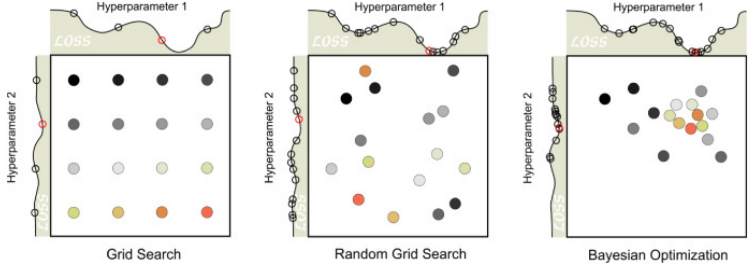
- Instead of testing all combinations, selects a subset of hyperparameter values randomly
- This avoids testing less important values and explores a wider variety of configurations with far fewer trials
- Often finds a near-optimal solution with far fewer computations

Example: Sampling 10 out of 27 configurations instead of all



Alternative 2 – Bayesian Optimization

- **Adaptive Sampling:** Uses past evaluations to build a surrogate model (e.g., Gaussian Process or Tree-structured Parzen Estimator) that predicts the performance of hyperparameters, guiding the selection of hyperparameters that are most likely to yield improvements
- **Sample-Efficient:** Requires fewer evaluations vs. Grid and Random Search



General ML Workflow

1. **Data Preparation**
   - Collect, clean, and preprocess data (handle missing values, standardize, encode categorical features, etc.)
   - Split data into training set (e.g., 80%) and testing set (e.g., 20%)
2. **Model Training & Hyperparameter Tuning**
   - Use training data to train ML models
   - Perform grid search (or methods like random search, Bayesian optimization) for hyperparameter tuning
   - Use cross-validation (e.g., k-fold CV) to evaluate different hyperparameter settings
3. **Final Model Training**
   - Once best hyperparameters determined, retrain model using entire training dataset
4. **Model Evaluation**
   - Test the trained model on the unseen testing data
   - Evaluate using appropriate metrics (e.g., accuracy, MSE, F1-score, AUC-ROC)

Why Final Model Evaluation on Testing Data is Crucial

- **Prevents Data Leakage & Overfitting to Validation Set**
  - Cross-validation helps tune model, but reuses training data multiple times
  - Best hyperparameters are selected based on CV performance, meaning the model has been indirectly exposed to the validation data
  - A separate unseen test set ensures the model's generalization ability is truly assessed
  - If the test performance is significantly worse than CV results, it indicates potential overfitting to the validation folds
- **Final Check Before Deployment**
  - Ensures there were no data leakage issues during training
  - Confirms the model is ready for production use or further tuning if necessary
- **Key Takeaway:** CV helps find the best model through hyperparameter tuning, but testing on unseen data ensures we get a true, unbiased estimate of how well the model will perform in practice

Variable Standardization: Neural Networks vs Random Forests

Neural Networks:

- **Sensitivity to Scale:** Gradient-based optimization (e.g., gradient descent) requires balanced feature scales for stable and efficient learning.
- **Activation Functions:** Functions such as sigmoid, tanh, and ReLU perform better with normalized inputs, preventing issues like saturation.
- **Optimization Efficiency:** Standardization helps the optimizer navigate the loss function surface more effectively, leading to faster convergence.

Random Forests:

- **Tree-Based Model:** Splits in decision trees are based on feature orderings rather than absolute values.
- **Robust to Scale:** Invariant to monotonic transformations; standardization is generally unnecessary.

Handling Missing Values in Machine Learning

Handling missing values is crucial in machine learning for several reasons:

- **Model Performance and Accuracy:** Many machine learning algorithms require complete data. Missing values can lead to inaccurate model estimates, reduce predictive power, or even cause algorithms to fail.
- **Bias and Inconsistencies:** Ignoring missing values can introduce bias into your model, especially if the missingness is not random. This bias can distort learned patterns and lead to poor generalization on new data.
- **Distorted Statistical Analysis:** Missing values can skew the statistical properties of your dataset (e.g., means, variances, and correlations), impacting feature engineering, variable selection, and model interpretation.

Why Encode Categorical Features?

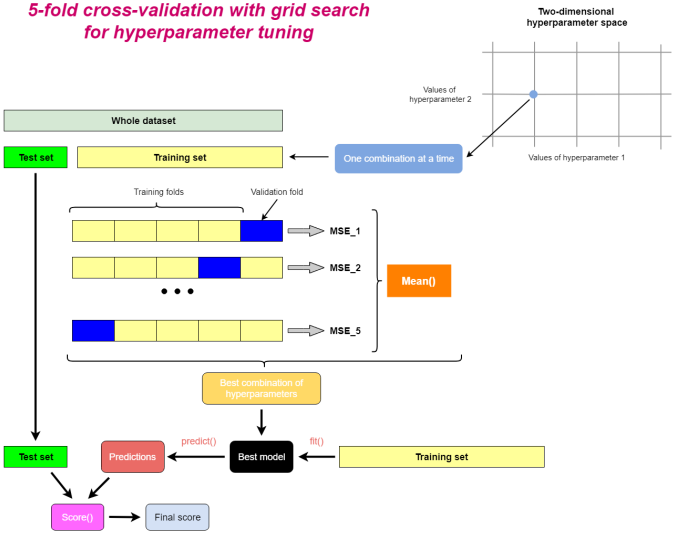
- **Numerical Requirement:** Most ML algorithms (e.g., linear regression, SVMs, neural networks) require numerical inputs for mathematical operations.
- **Meaningful Representation:** Encoding (e.g., one-hot or label encoding) transforms categorical data into numerical format, preventing erroneous ordinal assumptions.
- **Improved Model Performance:** Proper encoding reduces bias and inaccuracies, ensuring that the model correctly interprets each category.
- **Algorithm Compatibility:** Even for algorithms that handle categorical data (e.g., tree-based models), encoding can enhance performance and simplify feature handling.

One-Hot Encoding

- **Definition:** A method to convert categorical variables into a binary (0/1) vector representation.
- **Conversion Process:** For a categorical variable with n unique categories, one-hot encoding creates n new binary features. Each feature represents one category.
- **Representation:** For any observation, the binary vector has a value of 1 in the position corresponding to its category and 0 in all other positions.
- **Example:** Given a feature *Color* with categories: Red, Green, and Blue:
  - Color_Red
  - Color_Green
  - Color_Blue

  For an observation where Color is Green, the encoded vector is: $[0, 1, 0]$.
- **Advantages:** Avoids implying any ordinal relationship between categories and makes the data suitable for algorithms requiring numerical input.

General ML Workflow



Assignment for Next Week

- Lab Assignment: Titanic Competition at **Kaggle Titanic Competition**
- Required Reading: *Can Machines "Learn" Finance?* sections 1 to 5.1