Gradients

Numerical Gradients
Explanation: Numerical gradients are approximations of the gradient computed using finite differences, which are computationally expensive but straightforward to implement.
Formula: ∂f/∂x ≈ (f(x + ε) - f(x - ε)) / (2ε)
Applications: Gradient checking, numerical optimization, and debugging gradient computations.

Analytical Gradients
Explanation: Analytical gradients are exact derivatives of a function computed symbolically, which can be more efficient than numerical gradients but require careful derivation.
Formula: Derivative of the loss function with respect to model parameters
Applications: Efficient gradient computation, neural network training, and optimization algorithms.

Automatic Differentiation
Explanation: Automatic differentiation is a technique for efficiently computing gradients of functions specified by computer programs, often used in deep learning frameworks.
Formula: Uses chain rule to compute gradients recursively
Applications: Deep learning frameworks (e.g., TensorFlow, PyTorch), optimization, and complex model training.

Training Methods

Batch Gradient Descent
Explanation: Batch gradient descent updates model parameters by computing the gradient of the loss function with respect to the entire training dataset.
Formula: θ = θ - α * ∇J(θ))
Applications: Deep learning training, large-scale optimization problems, and regression analysis.

Stochastic Gradient Descent
Explanation: Stochastic gradient descent updates model parameters using the gradient of the loss function with respect to a single training example.
Formula: θ = θ - α * ∇J(θ;x(i);y(i)))
Applications: Online learning, large-scale machine learning, and neural network training.

Mini-Batch Gradient Descent
Explanation: Mini-batch gradient descent updates model parameters using the gradient of the loss function with respect to a small subset (mini-batch) of the training dataset.
Formula: θ = θ - α * ∇J(θ;x(i:i+n);y(i:i+n)))
Applications: Deep learning, efficient model training, and handling large datasets.

Early Stopping
Explanation: Early stopping is a regularization technique used to prevent overfitting by stopping the training process when the performance on a validation dataset starts to degrade.
Applications: Neural network training, model selection, and improving generalization.

L1 Regularization (Lasso)
- **Explanation:** L1 regularization adds a penalty equal to the absolute value of the magnitude of coefficients to the loss function, promoting sparsity in the model.
- **Formula:** L1_penalty = λ * ∑|w_i|
  λ: regularization parameter

  $\sum_{i=1}^{p} |w_i|$: This is the sum of the absolute values of the model parameters (weights)
- **Applications:** Feature selection, sparse modeling, and regression analysis.

L2 Regularization (Ridge)
- **Explanation:** L2 regularization adds a penalty equal to the square of the magnitude of coefficients to the loss function, promoting smaller weights and reducing overfitting.
- **Formula:** L2_penalty = λ * ∑(w_i^2)
- **Applications:** Ridge regression, improving model stability, and combating multicollinearity.

Dropout
- **Explanation:** Dropout is a regularization technique that randomly deactivates a fraction of neurons during training to prevent complex co-adaptations and improve generalization.
- **Formula:** During training, each neuron is retained with a probability p and dropped with probability 1 - p
- **Applications:** Neural network training, preventing overfitting, and improving model robustness.

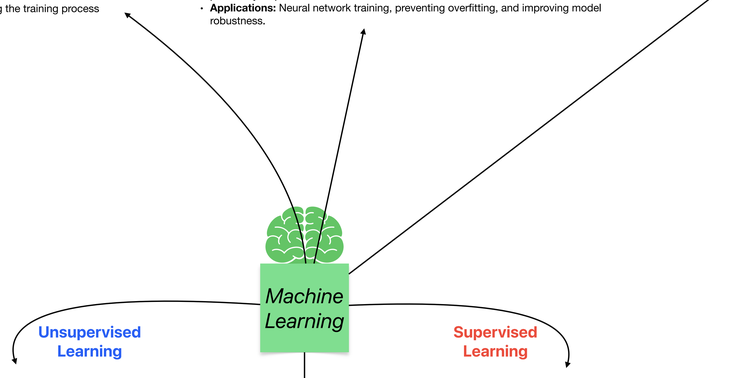
Error Handling

Bias-Variance Tradeoff
- **Explanation:** The bias-variance tradeoff refers to the balance between the error introduced by bias (underfitting) and variance (overfitting) in machine learning models.
- **Formula:** Error = Bias^2 + Variance + Irreducible Error
- **Applications:** Model selection, performance optimization, and understanding model behavior.

Overfitting
- **Explanation:** Overfitting occurs when a model learns the training data too well, including noise and outliers, resulting in poor generalization to new data.
- **Applications:** Model evaluation, improving generalization, and developing robust models.

Underfitting
- **Explanation:** Underfitting occurs when a model is too simple to capture the underlying patterns in the data, resulting in poor performance on both training and new data.
- **Applications:** Model evaluation, improving model complexity, and ensuring adequate learning.

Large Language Models
- They are basically computer programs that can understand and generate human-like text. They're like super-powered autocomplete models trained on large datasets to predict the answers of a given question, identify patterns or statistical data, or even create different creative text formats, like poems or code.
- LLMs can access, read, and process information using internet and keep the responses up-to-date based on current events.



Clustering
K-Means
- **Explanation:** K-Means is a clustering algorithm that partitions data into K clusters by iteratively assigning data points to the nearest cluster centroid and updating the centroids.
- **Formula:** Minimize the sum of squared distances from data points to their assigned cluster centroids
- **Applications:** Customer segmentation, market basket analysis, and image compression.

Hierarchical Clustering
- **Explanation:** Hierarchical clustering builds a tree of clusters by either merging (agglomerative) or splitting (divisive) clusters at each step based on their similarity.
- **Formula:** Dendrogram representation
- **Applications:** Gene expression data analysis, social network analysis, and document clustering.

Dimensionality Reduction
Principal Component Analysis (PCA)
- Explanation: PCA is a technique used to reduce the dimensionality of data while preserving most of its variance. It finds the orthogonal axes (principal components) that capture the most variance in the data.
- Formula: Eigenvalue decomposition or Singular Value Decomposition (SVD)
- Applications: Data visualization, noise reduction, and feature extraction.

t-SNE
- **Explanation:** t-SNE is a nonlinear dimensionality reduction technique particularly well-suited for visualization of high-dimensional data by preserving local similarities.
- **Formula:** Optimization based on minimizing the Kullback-Leibler divergence between high-dimensional and low-dimensional probability distributions
- **Applications:** Visualizing high-dimensional datasets, exploratory data analysis, and anomaly detection.

Association Rule Learning
Apriori Algorithm
- **Explanation:** Apriori is an algorithm for discovering frequent itemsets in transactional databases. It uses a bottom-up approach to generate candidate itemsets and prune infrequent ones.
- **Formula:** Support and confidence measures
- **Applications:** Market basket analysis, recommendation systems, and intrusion detection.

Linear Regression
- **Explanation:** Linear regression is a supervised learning algorithm used to model the relationship between a dependent variable and one or more independent variables. It assumes a linear relationship between the independent and dependent variables.
- **Formula:** y = mx + b
- **Applications:** Predicting housing prices, forecasting sales, and risk management.

Logistic Regression
- **Explanation:** Logistic regression is used for binary classification problems. It models the probability that a given input belongs to a particular category using a logistic function.
- **Formula:** p = 1 / (1 + e^(-z)), where z = mx + b
- **Applications:** Email spam detection, medical diagnosis (disease presence/absence), and credit scoring.

Decision Trees
- **Explanation:** Decision trees partition the feature space into regions, making decisions based on the values of input features. They are often used for classification and regression tasks.
- **Formula:** Decision rule based on feature thresholds
- **Applications:** Customer segmentation and recommendation systems.

Random Forests
- **Explanation:** Random forests are an ensemble learning method that constructs multiple decision trees during training and outputs the mode of the classes (classification) or mean prediction (regression) of the individual trees.
- **Formula:** Combination of decision trees
- **Applications:** Fraud detection, stock market prediction, and sentiment analysis.

Support Vector Machines (SVMs)
- **Explanation:** SVMs are supervised learning models used for classification and regression analysis. They find the hyperplane that best separates data points into different classes.
- **Formula:** w^T * x + b
- **Applications:** Spam filtering, sentiment analysis, and document classification.

Naive Bayes
- **Explanation:** Naive Bayes classifiers are based on Bayes' theorem with the assumption of independence between features. They are simple probabilistic classifiers often used for text classification.
- **Formula:** P(y|x) = P(x|y) * P(y) / P(x)
- **Applications:** Spam filtering, sentiment analysis, and document classification.

Neural Networks

Feedforward Neural Networks
- **Explanation:** Feedforward neural networks are composed of multiple layers of neurons, where information flows in one direction, from input to output. They are capable of approximating complex nonlinear functions.
- **Formula:** Output = Activation_Function(Weighted_Sum_of_Input)
- **Applications:** Image recognition, time series prediction, and regression analysis.

Convolutional Neural Networks (CNNs)
- **Explanation:** CNNs are designed for processing structured grid-like data, such as images. They use convolutional layers to automatically learn spatial hierarchies of features.
- **Formula:** Output = Convolution(Input, Filter) + Bias
- **Applications:** Image and video recognition, medical image analysis, and object detection.

Recurrent Neural Networks (RNNs)
- **Explanation:** RNNs are a type of neural network designed to handle sequential data. Unlike traditional neural networks that process each data point independently, RNNs can utilize information from previous data points in the sequence. This makes them well-suited for tasks like
- **Applications:** Language modeling, speech recognition, and time series forecasting.
  - RNNs can suffer from vanishing gradients, where information from earlier parts of the sequence gets washed out as it passes through the network.

LSTM (Long Short-Term Memory):
- Introduces special gates (forget gate, input gate, output gate) that control the flow of information within the network.
- These gates allow LSTMs to learn long-term dependencies more effectively than standard RNNs.
- **Formula:** --- LSTM (Long Short-Term Memory

  --- Forget Gate (f_t = σ(W_f * [h_(t-1), x_t] + b_f))
  --- Input Gate (i_t = σ(W_i * [h_(t-1), x_t] + b_i))
  --- Output Gate (o_t = σ(W_o * [h_(t-1), x_t] + b_o))
  --- Candidate Cell State (C_t^~ = tanh(W_c * [h_(t-1), x_t] + b_c))
  --- Cell State (C_t = f_t * C_(t-1) + i_t * C_t^~)
  --- Hidden State (h_t = o_t * tanh(C_t))

GRU (Gated Recurrent Unit):
- Similar to LSTMs, it uses gates to control information flow.
- GRUs have a simpler architecture compared to LSTMs, making them potentially faster to train.
- **Formula:** --- GRU (Gated Recurrent Unit

  --- Reset Gate (r_t = σ(W_r * [h_(t-1), x_t] + b_r))
  --- Update Gate (z_t = σ(W_z * [h_(t-1), x_t] + b_z))
  --- Candidate Hidden State (h_t^~ = tanh(W_h * [r_t * h_(t-1), x_t] + b_h))
  --- Hidden State (h_t = (1 - z_t) * h_(t-1) + z_t * h_t^~)

LLMs Architecture
- **Input Layer**
  - Text Data
    - Example: "The quick brown fox jumps over the lazy dog."
- **Tokenization**
  - Tokenizer
    - Definition: Splits the input text into smaller units (tokens).
    - Types:
      - Word Tokenization: Splits text into words.
      - Subword Tokenization: Splits text into subword units.
      - Character Tokenization: Splits text into individual characters.
  - Vocabulary
    - Definition: A predefined set of tokens the model understands.
    - Size: Typically ranges from 30,000 to 50,000 tokens.
    - Special Tokens:
      - [PAD]: Padding token
      - [UNK]: Unknown token
      - [CLS]: Classification token
      - [SEP]: Separator token
  - Formula: tokens = tokenizer(text)
    - Example:
      - Input: "The quick brown fox"
      - Tokens: [1012, 2532, 3456, 7896]
- Embedding Layer
  - Word Embeddings
    - Definition: Converts tokens into dense vectors representing the semantic meaning.
    - Types:
      - Static Embeddings: Pre-trained and fixed (e.g., Word2Vec, GloVe).
      - Contextual Embeddings: Dynamic and context-dependent (e.g., BERT, GPT).
  - Formula: embedding_vector = embedding_lookup(token)
    - Example:
      - Token: 1012
      - Embedding Vector: [0.25, -0.12, 0.43, ...]
- Positional Encoding
  - Purpose: Adds positional information to the embeddings to account for word order.
  - Characteristics:
    - Uses sine and cosine functions of different frequencies.
    - Ensures unique encoding for each position.
  - Formula: positional_encoding(position) = [sin(position/10000^(2i/dim)), cos(position/10000^(2i/dim))]
    - Example: For a position in the sequence, add respective sine and cosine values to the embedding vectors.
- Transformer Blocks
  - Multi-Head Self-Attention
    - Definition: Allows the model to focus on different parts of the sentence simultaneously.
    - Heads: Multiple attention heads (e.g., 8 or 12) to capture different relationships.
    - Formula: Attention(Q, K, V) = softmax((QK^T) / sqrt(d_k)) V
      - Example: Query (Q), Key (K), Value (V) matrices from the input embeddings.
  - Feed-Forward Neural Network
    - Definition: Applies non-linear transformations to the attention outputs.
    - Layers: Typically consists of two linear layers with a ReLU activation in between.
    - Formula: FFN(x) = max(0, xW1 + b1)W2 + b2
      - Example: Transforms the input through layers of weights and biases.
  - Layer Normalization and Residual Connections
    - Purpose: Stabilizes training and helps with gradient flow.
    - Layer Normalization: Normalizes inputs across the features.
    - Formula: LayerNorm(x + Sublayer(x))
      - Example: Normalizes the summed input and sublayer output.
- Stacking Transformer Layers
  - Depth: Multiple transformer layers (e.g., 12 or 24) stacked to build deeper understanding.
  - Purpose: Allows the model to capture more complex patterns and representations.
- Output Layer
  - Linear Layer
    - Definition: Maps the final hidden states to the vocabulary size.
    - Formula: logits = W_hidden * hidden_states + b_hidden
  - Softmax Layer
    - Definition: Converts scores (logits) to probabilities for each token in the vocabulary.
    - Formula: output_probs = softmax(logits)
    - Example:
      - Hidden State: [0.12, -0.34, 0.56, ...]
      - Output Probabilities: [0.01, 0.05, 0.9, ...]
- Decoding (for Text Generation)
  - Greedy Search
    - Definition: Selects the token with the highest probability at each step.
    - Pros and Cons: Simple but can lead to repetitive or suboptimal sequences.
  - Beam Search
    - Definition: Explores multiple possible sequences simultaneously.
    - Pros and Cons: Balances between exploration and exploitation but can be computationally expensive.
  - Sampling
    - Definition: Selects tokens based on their probability distribution.
    - Pros and Cons: Can produce more diverse and creative outputs but less deterministic.
    - Example: Choosing the highest probability tokens to form a sentence.
- Training (Backpropagation and Optimization)
  - Loss Function
    - Definition: Measures the difference between predicted and actual tokens.
    - Common Losses: Cross-Entropy Loss for classification tasks.
    - Formula: loss = -sum(actual * log(predicted))
  - Optimization Algorithm
    - Definition: Adjusts weights to minimize the loss.
    - Common Optimizers: Adam, SGD, RMSprop.
    - Example: Using Adam optimizer to update weights.
  - Backpropagation
    - Definition: Computes gradients of the loss with respect to model parameters.
    - Gradient Descent: Uses gradients to update parameters.
    - Formula: θ = θ - η * ∇L(θ)
      - Example: θ represents model parameters, η is the learning rate, and ∇L(θ) is the gradient of the loss.