Week 0: Introduction to Machine Learning & AI

Foundations, Algorithms, and Modern Applications

Machine Learning for Smarter Innovation

BSc-Level Course Series

September 28, 2025

Presentation Overview

Part 1: Machine Learning Foundations

Theory, Definitions, and Core Concepts

Formal Definition: What is Machine Learning?

Tom Mitchell's Definition (1997)

A computer program learns from:

- **Experience** E with respect to
- **Task** T and
- **Performance measure** P

if its performance at task T, as measured by P, improves with experience E.

Example:
E: Email database
T: Classify spam vs non-spam
P: Classification accuracy

Mathematical View

Learn function $f : \mathcal{X} \rightarrow \mathcal{Y}$
Given training set:

$$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$$

Find:

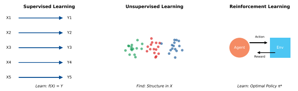
$$\hat{f} =_{f \in \mathcal{F}} \sum_{i=1}^{n} L(y_i, f(x_i)) + \lambda R(f)$$

where:

- L: Loss function
- R: Regularization term
- λ: Regularization strength

Three Paradigms of Machine Learning

Supervised



$$\{(x_i, y_i)\}_{i=1}^{n} \rightarrow \hat{f}$$

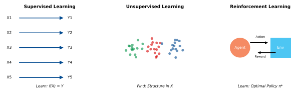
Applications:

- Email spam detection
- Medical diagnosis
- Stock price prediction
- Image recognition

Key Algorithms:

- Linear Regression
- Random Forest
- Neural Networks

Unsupervised



$$\{x_i\}_{i=1}^{n} \rightarrow \text{Structure}$$

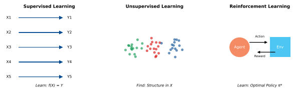
Applications:

- Customer segmentation
- Anomaly detection
- Data compression
- Market basket analysis

Key Algorithms:

- K-means clustering
- PCA
- Autoencoders

Reinforcement



$$(s_t, a_t, r_t, s_{t+1}) \rightarrow \pi^*$$

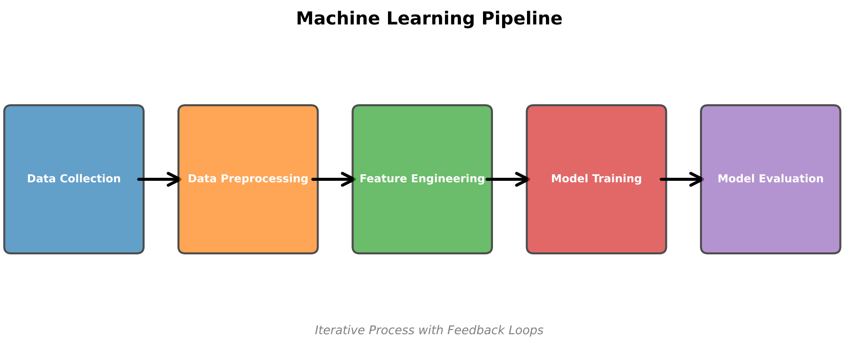
Applications:

- Game playing (Chess, Go)
- Autonomous vehicles
- Robotics control
- Resource allocation

Key Algorithms:

- Q-Learning
- Policy Gradients
- Actor-Critic

The Machine Learning Pipeline



Data Collection → Preprocessing → Feature Engineering → Model Training → Validation → Deployment

The Fundamental Tradeoff: Bias vs Variance

Mathematical Framework

For any learning algorithm, the expected error can be decomposed as:

$$\text{Error} = \text{Bias}^2 + \text{Variance} + \text{Noise}$$

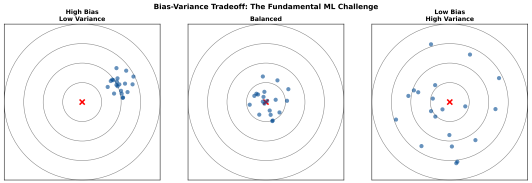
Bias: Error from oversimplifying assumptions

$$\text{Bias}[\hat{f}(x)] = E[\hat{f}(x)] - f(x)$$

Variance: Error from sensitivity to training data

$$\text{Var}[\hat{f}(x)] = E[(\hat{f}(x) - E[\hat{f}(x)])^2]$$

Key Insight: There's a fundamental tradeoff between bias and variance



Model Complexity Examples:

- High Bias: Linear models on nonlinear data
- Balanced: Regularized models
- High Variance: Deep trees, k-NN with small k

Machine Learning vs Traditional Programming

Traditional Programming

Process:

- Write explicit rules
- Code logic step by step
- Handle edge cases manually
- Deterministic outputs

Example: Email Classification

- IF contains “FREE” AND “LIMITED TIME”
- THEN classify as spam
- Requires manual rule updates

Limitations:

- Rules become complex
- Hard to handle exceptions
- Doesn’t adapt to new patterns

Machine Learning

Process:

- Provide example data
- Algorithm learns patterns
- Generalizes to new cases
- Probabilistic outputs

Example: Email Classification

- Train on 10,000 labeled emails
- Learn complex word patterns
- Automatically adapts to new spam

Advantages:

- Handles complex patterns
- Adapts to new data
- Discovers hidden relationships

Data Splitting: Training, Validation, and Test Sets

Why Split Data?

Training Set (60%):

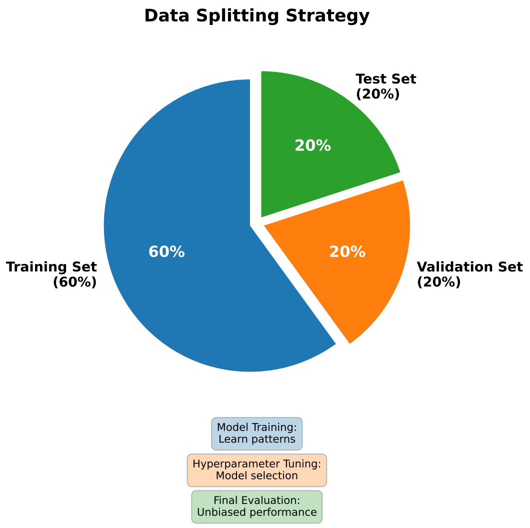
- Used to fit model parameters
- Algorithm learns from this data
- Larger is generally better

Validation Set (20%):

- Used for hyperparameter tuning
- Model selection and comparison
- Prevents overfitting to training data

Test Set (20%):

- Final unbiased evaluation
- Never seen during development
- Estimates real-world performance



Cross-Validation:

Evaluating Machine Learning Models

Classification Metrics

Accuracy:

$$\text{Accuracy} = \frac{\text{Correct Predictions}}{\text{Total Predictions}}$$

Precision:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

Recall (Sensitivity):

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

F1-Score:

$$F1 = 2 \cdot \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Regression Metrics

Mean Squared Error:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Root Mean Squared Error:

$$RMSE = \sqrt{MSE}$$

Mean Absolute Error:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

R-squared:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

Part 2: Supervised Learning Methods

Prediction and Classification Algorithms

Linear Methods: The Foundation

Linear Regression Family

Ordinary Least Squares:

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

$$\min_{\beta} ||y - X\beta||_2^2$$

Ridge Regression (L2):

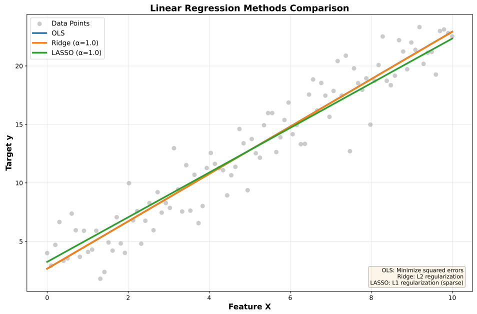
$$\hat{\beta}_{ridge} = (X^T X + \lambda I)^{-1} X^T y$$

$$\min_{\beta} ||y - X\beta||_2^2 + \lambda ||\beta||_2^2$$

LASSO (L1):

$$\min_{\beta} ||y - X\beta||_2^2 + \lambda ||\beta||_1$$

No closed form - use coordinate descent



Applications:

- House price prediction
- Sales forecasting
- Medical diagnosis
- Scientific modeling

Elastic Net (Best of Both):

Logistic Regression: Linear Classification

Mathematical Framework

Logistic Function:

$$p(y=1|x) = \frac{1}{1+e^{-(\beta_0+\beta^T x)}}$$

Odds Ratio:

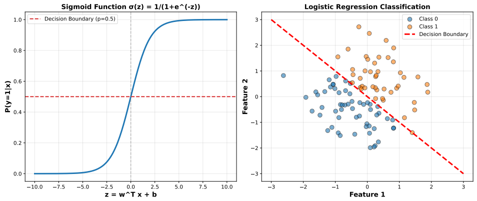
$$\frac{p}{1-p} = e^{\beta_0+\beta^T x}$$

Log-Likelihood:

$$\ell(\beta) = \sum_{i=1}^{n} [y_i \log p_i + (1-y_i)\log(1-p_i)]$$

No closed form solution

- Use gradient descent
- Newton-Raphson method
- Iteratively reweighted least squares



Decision Boundary:

$$\beta_0 + \beta^T x = 0$$

Applications:

- Email spam detection
- Medical diagnosis
- Marketing response
- Credit approval

Support Vector Machines: Maximum Margin Classification

Optimization Problem

Primal Form:

$$\min_{w,b} \frac{1}{2}||w||^2$$

$$\text{s.t. } y_i(w^T x_i + b) \geq 1, \forall i$$

Dual Form:

$$\max_{\alpha} \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j x_i^T x_j$$

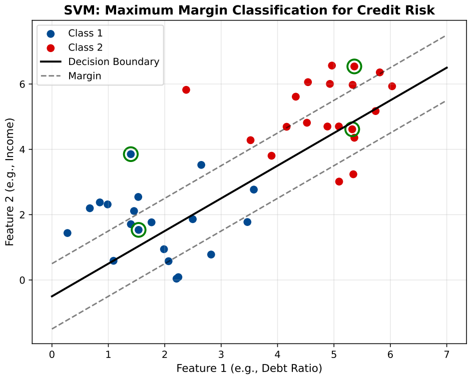
$$\text{s.t. } \alpha_i \geq 0, \sum_i \alpha_i y_i = 0$$

Kernel Trick:

$$K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$$

Common kernels:

- RBF: $K(x, z) = e^{-\gamma||x-z||^2}$



Key Concepts:

- **Support Vectors:** Data points on margin
- **Maximum Margin:** Optimal separating hyperplane
- **Kernel Trick:** Nonlinear classification

Advantages:

- Works well in high dimensions

Decision Trees: Interpretable Nonlinear Models

Tree Construction

Splitting Criterion:

Gini Impurity:

$$G = \sum_{k=1}^{K} p_k(1 - p_k)$$

Information Gain:

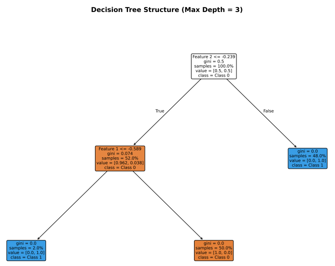
$$IG = H(parent) - \sum_{j} \frac{n_j}{n} H(child_j)$$

Entropy:

$$H = -\sum_{k=1}^{K} p_k \log_2 p_k$$

CART Algorithm:

1. Find best split across all features
2. Partition data based on split



Advantages:

- Highly interpretable
- Handles mixed data types
- No assumptions about distribution
- Captures interactions automatically

Disadvantages:

- Prone to overfitting
- Unstable (small data changes)

Random Forest: Ensemble of Trees

Ensemble Method

Bootstrap Aggregating (Bagging):

1. Draw B bootstrap samples
2. Train tree on each sample
3. Average predictions (regression)
4. Vote on class (classification)

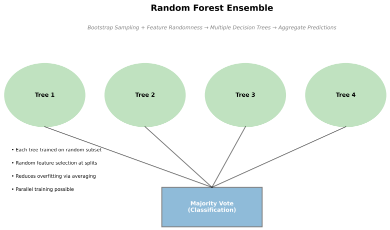
Random Feature Selection:

- At each split, randomly select m features
- Typically $m = \sqrt{p}$ for classification
- Typically $m = p/3$ for regression

Final Prediction:

$$\hat{y} = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$$

Key insight: Averaging reduces variance while maintaining low bias


Random Forest Ensemble
Bootstrap Sampling + Feature Randomness → Multiple Decision Trees → Aggregate Predictions
Tree 1
Tree 2
Tree 3
Tree 4
• Each tree trained on random subset
• Random feature selection at splits
• Reduces overfitting via averaging
• Parallel training possible
Majority Vote (Classification)

Advantages:

- Reduces overfitting
- Handles missing values
- Provides feature importance
- Works well out-of-the-box

Feature Importance:

- Mean decrease in impurity
- Permutation importance
- Out-of-bag importance

Gradient Boosting: Sequential Learning

Boosting Algorithm

Sequential Model Building:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$$

where h_m is trained on residuals:

$$r_{im} = -\frac{\partial L(y_i, F_{m-1}(x_i))}{\partial F_{m-1}(x_i)}$$

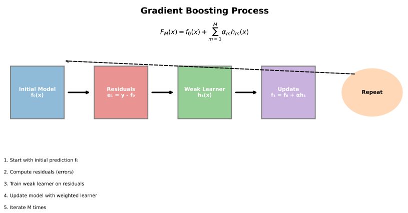
XGBoost Objective:

$$\mathcal{L} = \sum_i l(y_i, \hat{y}_i) + \sum_k \Omega(f_k)$$

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda||w||^2$$

Key Features:

- Regularization prevents overfitting
- Second-order derivatives
- Handles missing values


Gradient Boosting Process
$F_M(x) = f_0(x) + \sum_{m=1}^{M} \alpha_m h_m(x)$
Initial Model f₀(x)
Residuals e₁ = y - f₀
Weak Learner h₁(x)
Update f₁ = f₀ + αh₁
Repeat
1. Start with initial prediction f₀
2. Compute residuals (errors)
3. Train weak learner on residuals
4. Update model with weighted learner
5. Iterate M times

Popular Implementations:

- **XGBoost:** Extreme Gradient Boosting
- **LightGBM:** Fast gradient boosting
- **CatBoost:** Categorical features

Applications:

- Kaggle competitions
- Click-through rate prediction
- Risk modeling
- Ranking problems

k-Nearest Neighbors: Instance-Based Learning

Non-parametric Method

Algorithm:

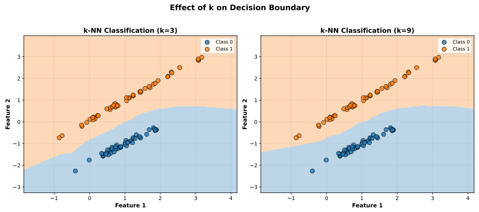
1. Store all training data
2. For new point, find k nearest neighbors
3. Classification: majority vote
4. Regression: average target values

Distance Metrics:

- Euclidean: $d(x, z) = \sqrt{\sum_i (x_i - z_i)^2}$
- Manhattan: $d(x, z) = \sum_i |x_i - z_i|$
- Minkowski: $d(x, z) = (\sum_i |x_i - z_i|^p)^{1/p}$

Choosing k:

- Small k: Low bias, high variance
- Large k: High bias, low variance
- Use cross-validation to select



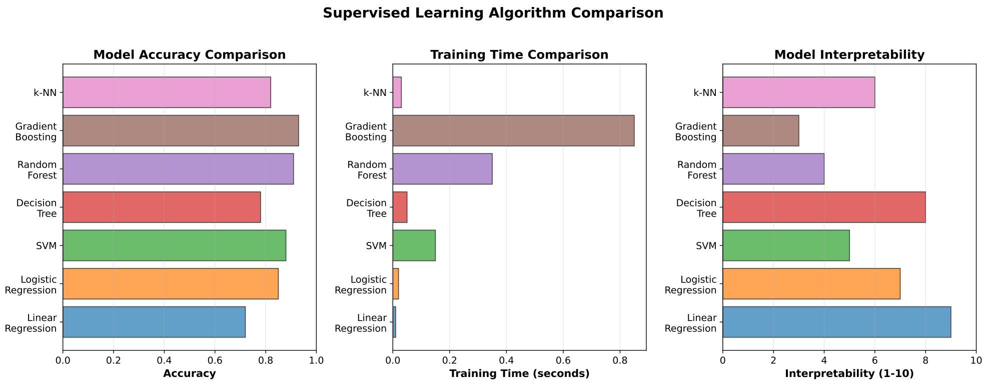
Advantages:

- Simple to understand and implement
- No assumptions about data distribution
- Adapts to local patterns
- Works well with small datasets

Disadvantages:

- Computationally expensive for large datasets
- Sensitive to irrelevant features
- Curse of dimensionality
- Sensitive to data scaling

Supervised Learning: Algorithm Comparison



Algorithm	Interpretability	Training Speed	Prediction Speed	Accuracy
Linear Regression	High	Fast	Fast	Low-Medium
Logistic Regression	High	Fast	Fast	Medium
Decision Tree	High	Medium	Fast	Medium
Random Forest	Medium	Slow	Medium	High
XGBoost	Low	Slow	Medium	Very High
SVM	Low	Slow	Fast	High
k-NN	Medium	Fast	Slow	Medium-High

Part 3: Unsupervised Learning Methods

Discovering Hidden Structure in Data

K-means Clustering: Partitioning Data

Algorithm

Objective Function:

$$J = \sum_{i=1}^{n} \sum_{k=1}^{K} w_{ik} ||x_i - \mu_k||^2$$

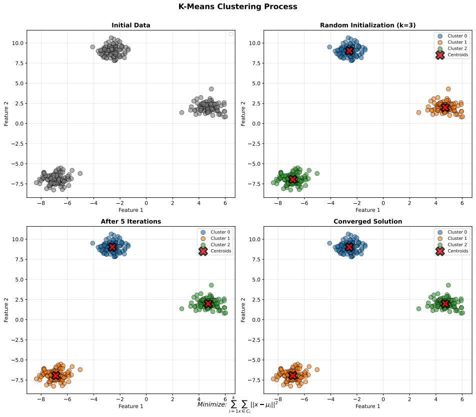
K-means Algorithm:

1. Initialize K cluster centers randomly
2. Assign each point to nearest center
3. Update centers to mean of assigned points
4. Repeat until convergence

Update Rules:

$$w_{ik} = \begin{cases} 1 & \text{if } k =_j ||x_i - \mu_j||^2 \\ 0 & \text{otherwise} \end{cases}$$

$$\mu_k = \frac{\sum_{i=1}^{n} w_{ik} x_i}{\sum_{i=1}^{n} w_{ik}}$$



Choosing K:

- **Elbow Method:** Plot within-cluster sum of squares
- **Silhouette Analysis:** Measure cluster separation
- **Gap Statistic:** Compare to random data

Advantages:

Hierarchical Clustering: Building Cluster Trees

Agglomerative Approach

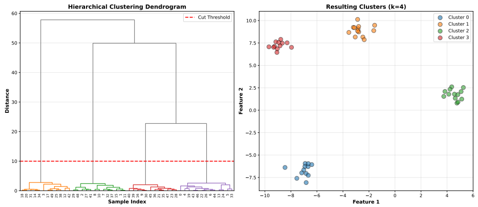
Algorithm:

1. Start with each point as its own cluster
2. Merge closest pair of clusters
3. Repeat until single cluster remains
4. Cut dendrogram at desired level

Linkage Criteria:

- **Single:** $\min(d(a,b))$ where $a \in A, b \in B$
- **Complete:** $\max(d(a,b))$ where $a \in A, b \in B$
- **Average:** $\frac{1}{|A||B|} \sum_{a \in A} \sum_{b \in B} d(a,b)$
- **Ward:** Minimize within-cluster variance

Time Complexity: $O(n^3)$ for naive implementation



Advantages:

- No need to specify number of clusters
- Produces hierarchy of clusters
- Deterministic results
- Works with any distance metric

Disadvantages:

- Computationally expensive
- Sensitive to outliers
- Difficult to handle large datasets

DBSCAN: Density-Based Clustering

Density-Based Approach

Key Concepts:

- **Core Point:** $\geq$ minPts neighbors within ϵ
- **Border Point:** In neighborhood of core point
- **Noise Point:** Neither core nor border

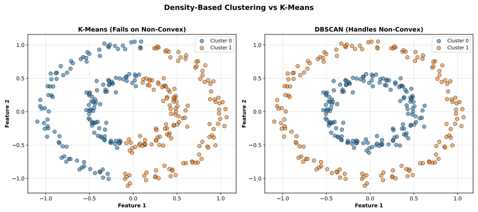
Algorithm:

1. For each unvisited point
2. If core point, start new cluster
3. Add all density-reachable points
4. Mark non-core points as noise

Parameters:

- ϵ: Neighborhood radius
- minPts: Minimum points for core

$$\text{Density} = \frac{\text{Points in } \epsilon\text{-neighborhood}}{|\epsilon\text{-neighborhood}|}$$



Advantages:

- Finds arbitrary-shaped clusters
- Automatically determines cluster count
- Robust to outliers
- Identifies noise points

Applications:

- Anomaly detection
- Image segmentation
- Fraud detection
- Social network analysis

Principal Component Analysis: Dimensionality Reduction

Mathematical Framework

Objective: Find directions of maximum variance

Covariance Matrix:

$$C = \frac{1}{n-1} X^T X$$

Eigendecomposition:

$$C = V \Lambda V^T$$

where V contains eigenvectors (principal components) and Λ contains eigenvalues.

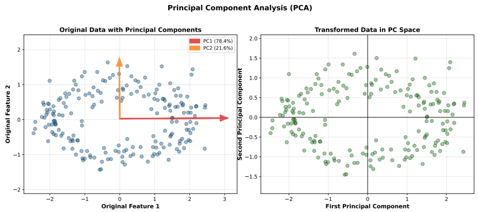
Dimensionality Reduction:

$$Z = XW$$

where W contains the first k principal components.

Variance Explained:

$$\text{Explained Variance} = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{p} \lambda_i}$$



Steps:

1. Standardize the data
2. Compute covariance matrix
3. Find eigenvalues and eigenvectors
4. Sort by eigenvalue magnitude
5. Select top k components
6. Transform data

Applications:

- Data visualization
- Noise reduction
- Feature extraction

Autoencoders: Neural Network Dimensionality Reduction

Architecture

Encoder:

$$z = f(Wx + b)$$

Decoder:

$$\hat{x} = g(W'z + b')$$

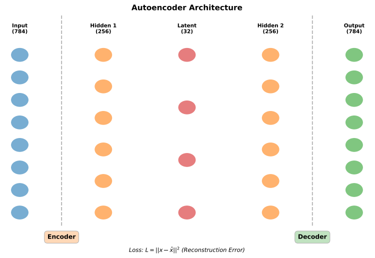
Objective:

$$\min_{W,W'} ||x - \hat{x}||^2$$

Types of Autoencoders:

- **Vanilla:** Basic encoder-decoder
- **Denoising:** Add noise to input
- **Sparse:** Encourage sparse representations
- **Variational:** Probabilistic latent space

Bottleneck Layer: Forces compression and learning of important features



Advantages over PCA:

- Nonlinear transformations
- Better reconstruction for complex data
- Can learn hierarchical features
- Flexible architecture

Applications:

- Image denoising
- Anomaly detection
- Data compression

t-SNE and UMAP: Nonlinear Visualization

t-SNE

t-Distributed Stochastic Neighbor Embedding
High-dimensional similarities:

$$p_{j|i} = \frac{\exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k\neq i}\exp(-||x_i - x_k||^2/2\sigma_i^2)}$$

Low-dimensional similarities:

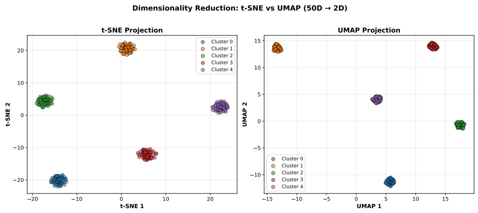
$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k\neq l}(1 + ||y_k - y_l||^2)^{-1}}$$

Objective: Minimize KL divergence

$$C = \sum_i KL(P_i||Q_i)$$

Key Features:

- Preserves local structure
- Heavy-tailed distribution in low-dim
- Stochastic optimization



UMAP (Uniform Manifold Approximation)

- Faster than t-SNE
- Preserves global structure better
- Consistent results across runs
- Can embed to higher dimensions

When to Use:

- **t-SNE:** Detailed local clustering
- **UMAP:** Balance of local and global
- **PCA:** Linear structure, fast

Evaluating Unsupervised Learning

Internal Metrics

Silhouette Score:

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

where $a(i)$ = avg distance within cluster, $b(i)$ = avg distance to nearest cluster

Calinski-Harabasz Index:

$$CH = \frac{SS_B/(k-1)}{SS_W/(n-k)}$$

Davies-Bouldin Index:

$$DB = \frac{1}{k}\sum_{i=1}^{k} \max_{j \neq i} \frac{\sigma_i + \sigma_j}{d(c_i, c_j)}$$

Inertia (Within-cluster sum of squares):

$$WCSS = \sum_{i=1}^{k} \sum_{x \in C_i} ||x - \mu_i||^2$$

External Metrics

Adjusted Rand Index:

$$ARI = \frac{RI - E[RI]}{\max(RI) - E[RI]}$$

Normalized Mutual Information:

$$NMI = \frac{MI(U, V)}{\sqrt{H(U)H(V)}}$$

Homogeneity and Completeness:

- Homogeneity: Each cluster contains only one class
- Completeness: All members of class in same cluster

V-measure: Harmonic mean of homogeneity and completeness

Note: External metrics require ground truth labels

Unsupervised Learning Applications

Clustering

Customer Segmentation:

- Group customers by behavior
- Targeted marketing campaigns
- Product recommendations

Market Basket Analysis:

- Find product associations
- Store layout optimization
- Cross-selling opportunities

Image Segmentation:

- Medical image analysis
- Computer vision
- Object recognition

Dimensionality Reduction

Data Visualization:

- Explore high-dimensional data
- Identify patterns and outliers
- Present insights to stakeholders

Feature Engineering:

- Reduce computational cost
- Remove noise and redundancy
- Improve model performance

Compression:

- Image and audio compression
- Efficient data storage
- Fast data transmission

Anomaly Detection

Fraud Detection:

- Credit card transactions
- Insurance claims
- Online account activity

Network Security:

- Intrusion detection
- Malware identification
- Unusual traffic patterns

Quality Control:

- Manufacturing defects
- System monitoring
- Predictive maintenance

Unsupervised learning reveals hidden patterns and structures in data without labeled examples

Part 4: Neural Networks and Deep Learning

From Perceptrons to Modern Architectures

The Perceptron: Foundation of Neural Networks

Mathematical Model

Linear Combination:

$$z = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b$$

$$z = \mathbf{w}^T \mathbf{x} + b$$

Activation Function:

$$y = \sigma(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

Decision Boundary:

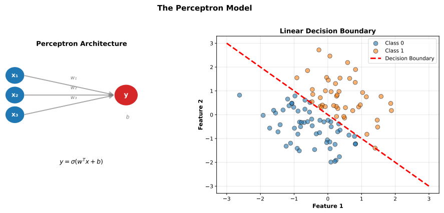
$$\mathbf{w}^T \mathbf{x} + b = 0$$

Learning Rule (Perceptron Algorithm):

$$w_i := w_i + \eta(y - \hat{y})x_i$$

$$b := b + \eta(y - \hat{y})$$

where η is the learning rate



Perceptron Limitations:

- Can only learn linearly separable functions
- Cannot solve XOR problem
- Single decision boundary

Historical Impact:

- First neural network model (1943)
- Led to "AI Winter" when limitations discovered
- Foundation for modern deep learning

Multi-Layer Perceptron: Universal Approximator

Architecture

Forward Propagation:

$$z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}$$

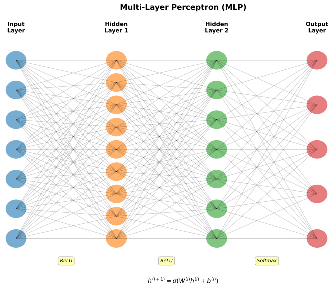
$$a^{[l]} = g^{[l]}(z^{[l]})$$

Universal Approximation Theorem: A neural network with:

- One hidden layer
- Finite number of neurons
- Non-linear activation function

can approximate any continuous function on a compact set to arbitrary accuracy.
Key Insight: Width vs depth tradeoff

- Wide shallow networks: Exponential width needed
- Deep narrow networks: Polynomial parameters


Multi-Layer Perceptron (MLP)
Input Layer
Hidden Layer 1
Hidden Layer 2
Output Layer
ReLU
ReLU
Softmax
$h^{(l+1)} = \sigma(W^{(l)}h^{(l)} + b^{(l)})$

Activation Functions:

- **Sigmoid:** $\sigma(x) = \frac{1}{1+e^{-x}}$
- **Tanh:** $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- **ReLU:** $\text{ReLU}(x) = \max(0, x)$
- **Leaky ReLU:** $\text{LeakyReLU}(x) = \max(0.01x, x)$

Backpropagation: Training Neural Networks

Algorithm

Chain Rule Application:

$$\frac{\partial L}{\partial W^{[l]}} = \frac{\partial L}{\partial z^{[l]}} \frac{\partial z^{[l]}}{\partial W^{[l]}}$$

Backward Pass:

$$\delta^{[l]} = \frac{\partial L}{\partial z^{[l]}}$$

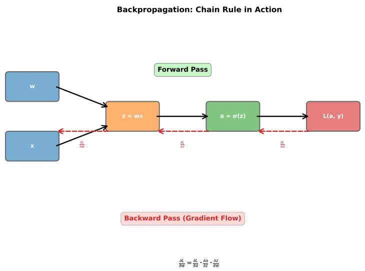
$$\delta^{[l-1]} = (W^{[l]})^T \delta^{[l]} \odot g'(z^{[l-1]})$$

Parameter Updates:

$$\frac{\partial L}{\partial W^{[l]}} = \delta^{[l]} (a^{[l-1]})^T$$

$$\frac{\partial L}{\partial b^{[l]}} = \delta^{[l]}$$

Gradient Descent:


Backpropagation: Chain Rule in Action
Forward Pass
w
x
z = wx
a = σ(z)
L(a, y)
$\frac{\partial L}{\partial w}$
$\frac{\partial L}{\partial z}$
$\frac{\partial L}{\partial a}$
Backward Pass (Gradient Flow)
$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial w}$

Computational Graph:

- Forward pass: Compute outputs
- Backward pass: Compute gradients
- Automatic differentiation
- Memory vs computation tradeoff

Challenges:

- Vanishing gradients
- Exploding gradients

Activation Functions: Nonlinearity in Neural Networks

Common Activations

Sigmoid:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Range: $(0, 1)$, smooth, vanishing gradients

Tanh:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Range: $(-1, 1)$, zero-centered, still vanishing gradients

ReLU:

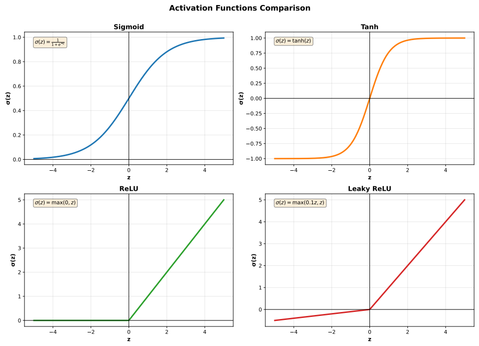
$$\text{ReLU}(x) = \max(0, x)$$

Simple, fast, sparse activations, dying ReLU problem

Leaky ReLU:

$$\text{LeakyReLU}(x) = \max(\alpha x, x)$$

Fixes dying ReLU, $\alpha = 0.01$ typically



Modern Activations:

- **ELU:** $f(x) = \begin{cases} x & x > 0 \\ \alpha(e^x - 1) & x \leq 0 \end{cases}$
- **Swish:** $f(x) = x \cdot \sigma(\beta x)$
- **GELU:** $f(x) = x \cdot \Phi(x)$

Choice Guidelines:

- Hidden layers: ReLU or variants

Convolutional Neural Networks: Spatial Feature Learning

Architecture Components

Convolution Operation:

$$(I * K)_{ij} = \sum_m \sum_n I_{i+m,j+n} K_{m,n}$$

Key Concepts:

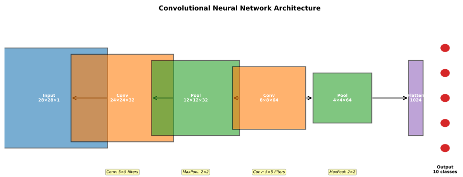
- **Local Connectivity:** Neurons connect to local regions
- **Parameter Sharing:** Same filter across all positions
- **Translation Invariance:** Features detected anywhere

Typical CNN Architecture:

1. Convolution + ReLU
2. Pooling (max or average)
3. Repeat multiple times
4. Flatten and fully connected layers
5. Final classification layer

Filter Parameters:

- Kernel size (3x3, 5x5, 7x7)
- Stride (step size)



Pooling Operations:

- **Max Pooling:** Take maximum in region
- **Average Pooling:** Take average in region
- **Global Pooling:** Pool entire feature map

Applications:

- Image classification
- Object detection
- Medical imaging
- Computer vision