Week 0: Introduction to Machine Learning & AI

Foundations, Algorithms, and Modern Applications

Machine Learning for Smarter Innovation

BSc-Level Course Series

December 11, 2025

Presentation Overview

1. Machine Learning Foundations
2. Supervised Learning Methods
3. Unsupervised Learning Methods
4. Neural Networks and Deep Learning
5. Generative AI and Modern Applications

Part 1: Machine Learning Foundations

Theory, Definitions, and Core Concepts

You Want a Program That Gets Better

Think about email spam detection:

- You **show it** 10,000 examples (spam and not spam)
- It learns patterns in the data
- It gets better at recognizing new spam

Tom Mitchell (1997) formalized this:
A program learns from **Experience** E at **Task** T measured by **Performance** P if its performance improves with experience.

Concrete Example:

E: 10,000 labeled emails

T: Classify spam vs non-spam

P: 85% -¿ 95% accuracy after training

The Mathematical Pattern

What the algorithm actually does:
Step 1: Given labeled examples

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

Step 2: Find function that maps inputs to outputs

$$f : \mathcal{X} \to \mathcal{Y}$$

Step 3: Minimize errors on training data

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

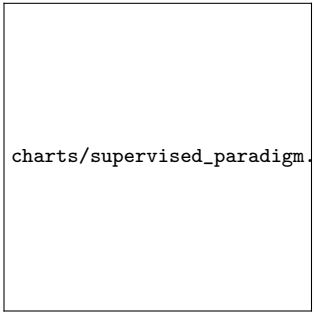
where L measures mistakes, R prevents overfitting

This optimization is what "learning" means mathematically

Learning formalizes improvement through optimization - mathematical frameworks transform intuitive experience into tractable computational problems

Three Paradigms of Machine Learning

Supervised

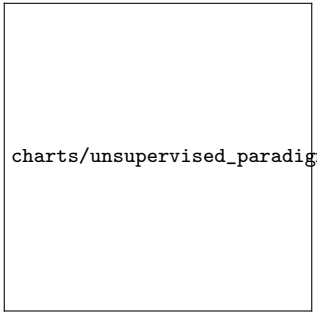


$\{(x_i, y_i)\}_{i=1}^{n} \rightarrow \hat{f}$

Applications:

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

Unsupervised

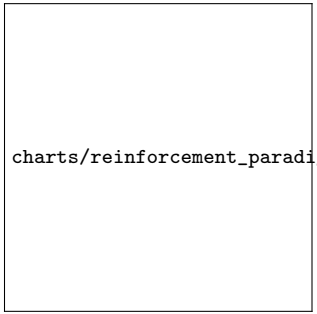


$\{x_i\}_{i=1}^{n} \rightarrow$ Structure

Applications:

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

Reinforcement



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

Applications:

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

The Machine Learning Pipeline

charts/ml_pipeline.pdf

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

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

Machine learning excels where explicit programming fails - pattern complexity and adaptation requirements favor data-driven approaches over rule-based systems

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:

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:

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

Evaluation metrics quantify model quality - metric selection aligns algorithmic optimization with domain-specific success criteria

Part 2: Supervised Learning Methods

Prediction and Classification Algorithms

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):

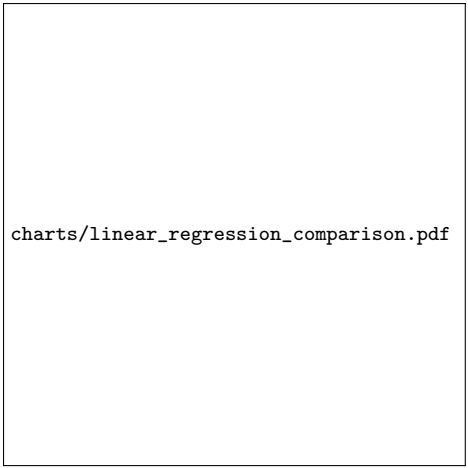
$$\hat{\beta}_{ridge} = (X^T X + \lambda I)^{-1} X^T y$$

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

LASSO (L1):

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

No closed form - use coordinate descent



Applications:

- House price prediction

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

charts/logistic_regression.pdf

Decision Boundary:

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

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

charts/svm_classification.pdf

Key Concepts:

- **Support Vectors:** Data points on margin
- **Maximum Margin:** Optimal separating hyperplane

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
3. Recursively apply to children

charts/decision_tree.pdf

Advantages:

- Highly interpretable
- Handles mixed data types

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

Advantages:

- Reduces overfitting
- Handles missing values

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

charts/gradient_boosting.pdf

Popular Implementations:

- **XGBoost:** Extreme Gradient Boosting
- **LightGBM:** Fast gradient boosting

Non-parametric Method

Algorithm:

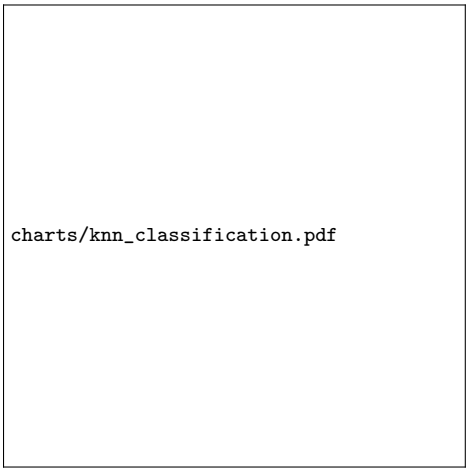
1. Store all training data
2. For new point, find k nearest neighbors
3. Classification: majority vote
4. Regression: average target values

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

Supervised Learning: Algorithm Comparison

charts/algorithm_comparison.pdf

Part 3: Unsupervised Learning Methods

Discovering Hidden Structure in Data

K-means: You Want to Group Similar Data Points

The Idea

You have customer data and want to find 3 natural groups:

Step-by-step:

1. **Start:** Place 3 center points randomly
2. **Assign:** Each customer joins nearest center
3. **Update:** Move centers to average of their group
4. **Repeat:** Until centers stop moving

Worked Example (2D):

- Point $x_1 = [2, 3]$, Centers: $\mu_1 = [1, 2]$, $\mu_2 = [5, 5]$
- Distance to μ_1: $\sqrt{(2-1)^2 + (3-2)^2} = 1.4$
- Distance to μ_2: $\sqrt{(2-5)^2 + (3-5)^2} = 3.6$
- Assign x_1 to cluster 1 (closer!)

The algorithm minimizes total distance from points to their cluster centers

charts/kmeans_clustering.pdf

The optimization:

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

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

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

Mathematical Framework

Objective: Find directions of maximum variance
Covariance Matrix:

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

Eigendecomposition:

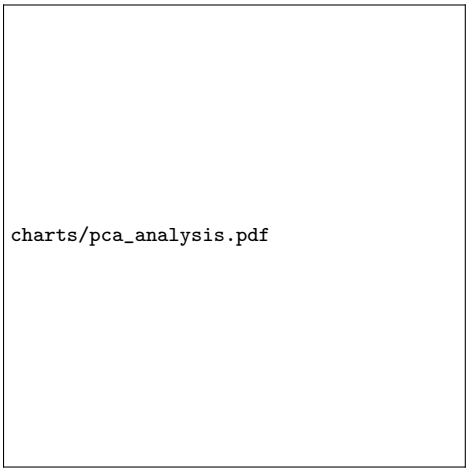
$$C = V \Lambda V^T$$

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

Architecture

Encoder:

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

Decoder:

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

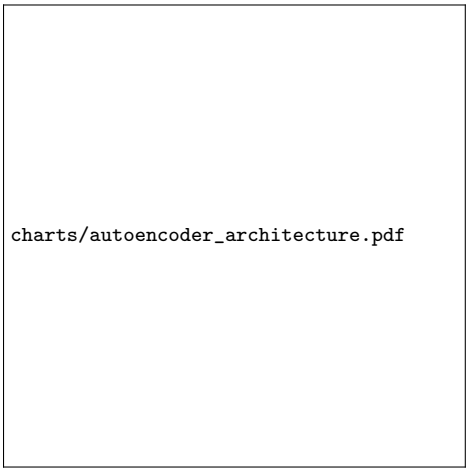
Objective:

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

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

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:

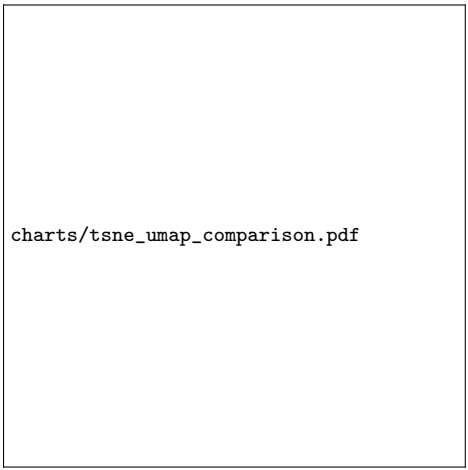
$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k \neq l}(1 + ||y_k - y_l||^2)^{-1}}$$

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

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

Unsupervised methods discover latent structure - clustering, dimensionality reduction, and anomaly detection operate without supervisory signal

Part 4: Neural Networks and Deep Learning

From Perceptrons to Modern Architectures

Mathematical Model

Linear Combination:

$$z = w_1x_1 + w_2x_2 + \ldots + w_nx_n + b$$

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

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

charts/perceptron_model.pdf

Perceptron Limitations:

- Can only learn linearly separable functions
- Cannot solve XOR problem

Architecture

Forward Propagation:

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

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

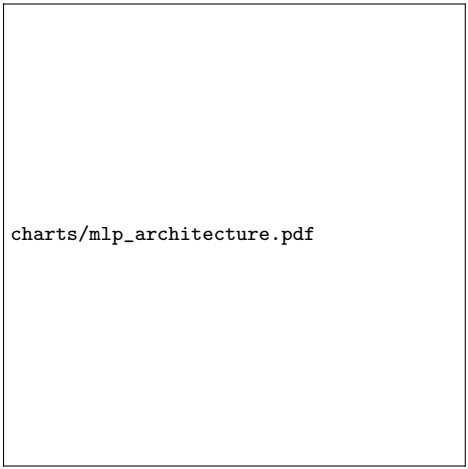
Universal Approximation Theorem: A neural network with:

- 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



Activation Functions:

- **Sigmoid:** $\sigma(x) = \frac{1}{1+e^{-x}}$
- **Tanh:** $\tanh(x) = \frac{e^{x}-e^{-x}}{}$

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:

charts/backpropagation.pdf

Computational Graph:

- Forward pass: Compute outputs
- Backward pass: Compute gradients

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

charts/activation_functions.pdf

Modern Activations:

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

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)

charts/cnn_architecture.pdf

Pooling Operations:

- **Max Pooling:** Take maximum in region
- **Average Pooling:** Take average in region

RNN Architecture

Recurrence Relation:

$$h_t = \tanh(W_h h_{t-1} + W_x x_t + b)$$

Output:

$$y_t = W_y h_t + b_y$$

Unfolded in Time:

- Share parameters across time steps
- Process variable-length sequences
- Memory of previous inputs

Training: Backpropagation Through Time (BPTT)

$$\frac{\partial L}{\partial W_h} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W_h}$$

Vanishing Gradient Problem:

$$\frac{\partial h_t}{} \quad \prod^{k} \frac{\partial h_{t-i+1}}{}$$

charts/rnn_architecture.pdf

LSTM (Long Short-Term Memory):

- Forget gate: What to forget from cell state
- Input gate: What new info to store

Optimization Algorithms

Stochastic Gradient Descent:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L(\theta_t)$$

Momentum:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta L(\theta_t)$$

$$\theta_{t+1} = \theta_t - v_t$$

Adam (Adaptive Moment Estimation):

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t} + \epsilon} m_t$$

Learning Rate Scheduling:

- Step decay
- Exponential decay

charts/training_curves.pdf

Regularization Techniques:

- **L1/L2 Regularization:** Add penalty to weights
- **Dropout:** Randomly set neurons to zero

The Deep Learning Revolution: Key Milestones

Historical Timeline

2012: AlexNet

- Won ImageNet competition
- 8-layer CNN
- GPU acceleration
- Dropout regularization

2014: VGGNet

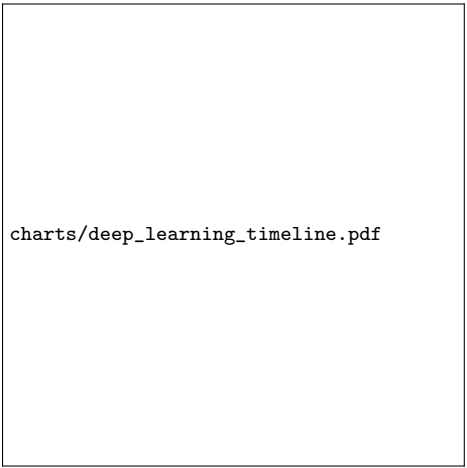
- Deeper networks (16-19 layers)
- Small 3x3 filters
- Showed depth importance

2015: ResNet

- Residual connections
- 152 layers deep
- Solved vanishing gradient
- Skip connections: $y = F(x) + x$

2017: Transformer

- Attention mechanism
- Parallelizable architecture



Key Enablers:

- **Big Data:** ImageNet, large text corpora
- **GPU Computing:** Parallel processing

Modern Neural Network Architectures

Vision

ResNet:

- Skip connections
- Very deep networks
- Identity mapping

EfficientNet:

- Compound scaling
- Balanced depth/width/resolution
- Mobile-friendly

Vision Transformer:

- Self-attention for images
- Patch-based processing
- Competitive with CNNs

Language

Transformer:

- Self-attention mechanism
- Encoder-decoder architecture
- Parallelizable training

BERT:

- Bidirectional encoding
- Pre-trained representations
- Fine-tuning for tasks

GPT:

- Autoregressive generation
- Scaling laws
- Few-shot learning

Multimodal

CLIP:

- Vision-language understanding
- Contrastive learning
- Zero-shot classification

DALL-E:

- Text-to-image generation
- Multimodal creativity
- Large-scale training

Flamingo:

- Few-shot multimodal learning
- Vision-language tasks
- In-context learning

Modern architectures combine multiple techniques for state-of-the-art performance across domains

Specialized architectures encode domain-specific inductive biases - CNNs for vision, RNNs for sequences, Transformers for attention

Part 5: Generative AI and Modern Applications

Creating New Content with Artificial Intelligence

Generative vs Discriminative Models

Discriminative Models

Goal: Learn decision boundary

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

Examples:

- Logistic regression
- Support Vector Machines
- Neural network classifiers
- Decision trees

Applications:

- Classification tasks
- Regression problems
- Prediction from features
- Pattern recognition

Advantages:

- Often better at classification
- More direct approach

Generative Models

Goal: Learn data distribution

$$p(x) \text{ or } p(x, y)$$

Examples:

- Generative Adversarial Networks
- Variational Autoencoders
- Autoregressive models
- Diffusion models

Applications:

- Data generation
- Image synthesis
- Text generation
- Data augmentation

Advantages:

- Can generate new samples
- Handle missing data
- Provide uncertainty estimates

Mathematical Framework

Generator: $G : \mathcal{Z} \to \mathcal{X}$

$$G(z) \text{ where } z \sim p_z(z)$$

Discriminator: $D : \mathcal{X} \to [0, 1]$

$$D(x) = \text{Probability } x \text{ is real}$$

Minimax Objective:

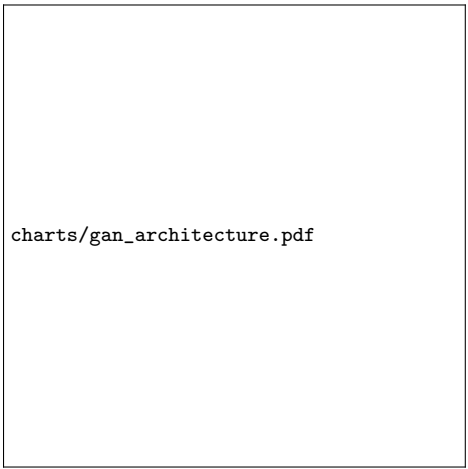
$$\min_G \max_D V(D, G)$$

where:

$$V(D, G) = \mathbb{E}_{x \sim p_{data}}[\log D(x)] + \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z)))]$$

Training Process:

1. Train D to distinguish real vs fake
2. Train G to fool D
3. Alternate until convergence



Training Challenges:

- Mode collapse
- Training instability

Probabilistic Framework

Encoder (Recognition Model):

$$q_\phi(z|x) \approx p(z|x)$$

Decoder (Generative Model):

$$p_\theta(x|z)$$

Evidence Lower Bound (ELBO):

$$\log p(x) \geq \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - KL(q_\phi(z|x)||p(z))$$

Loss Function:

$$\mathcal{L} = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - \beta \cdot KL(q_\phi(z|x)||p(z))$$

Reparameterization Trick:

$$z = \mu + \sigma \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$$

Enables backpropagation through stochastic node

charts/vae_architecture.pdf

Key Advantages:

- Stable training
- Meaningful latent space

Mathematical Formulation

Forward Process (Noise Addition):

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$$

Reverse Process (Denoising):

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$$

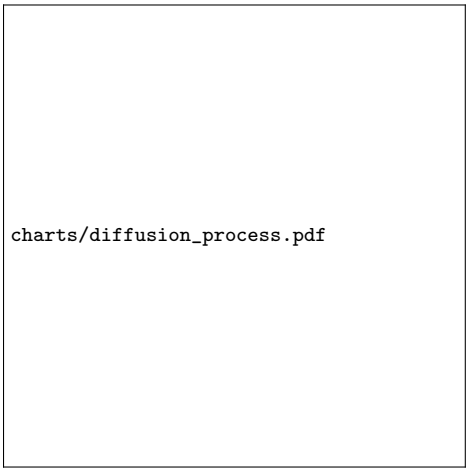
Training Objective:

$$L = \mathbb{E}_{t,x_0,\epsilon}[||\epsilon - \epsilon_\theta(x_t, t)||^2]$$

where ϵ_θ predicts noise added at step t

Sampling Process:

1. Start with random noise $x_T \sim \mathcal{N}(0, I)$
2. Iteratively denoise: $x_{t-1} = \mu_\theta(x_t, t) + \sigma_t \epsilon$
3. Continue until x_0 (clean sample)



Key Properties:

- High-quality generation
- Stable training

Self-Attention Mechanism

Attention Formula:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

where:

- Q: Queries matrix
- K: Keys matrix
- V: Values matrix
- d_k: Dimension of keys

Multi-Head Attention:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(head_1, \ldots, head_h)W^O$$

where:

$$head_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Position Encoding: Since no recurrence

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$$

charts/transformer_architecture.pdf

Architecture Components:

- **Encoder:** Self-attention + Feed-forward
- **Decoder:** Masked self-attention + Cross-attention

Model Evolution

GPT (Generative Pre-trained Transformer):

- Autoregressive generation
- Transformer decoder architecture
- Pre-train then fine-tune

BERT (Bidirectional Encoder Representations):

- Bidirectional context
- Masked language modeling
- Next sentence prediction

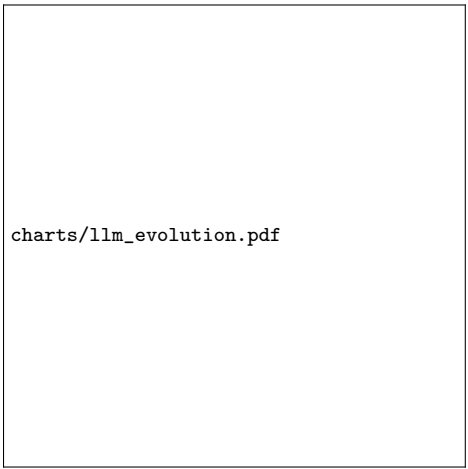
Scaling Laws:

$$L(N) = \left(\frac{N_c}{N}\right)^{\alpha}$$

where L is loss, N is parameters, $\alpha \approx 0.076$

Key Findings:

- Performance scales predictably with size
- Emergent abilities at scale
- Few-shot learning capabilities



Model Sizes:

- GPT-1: 117M parameters (2018)
- GPT-2: 1.5B parameters (2019)

Generative AI Applications: Transforming Industries

Content Creation

Text Generation:

- GPT-4: Advanced writing
- Claude: Helpful AI assistant
- Jasper: Marketing copy

Image Generation:

- DALL-E 2: Text-to-image
- Midjourney: Artistic images
- Stable Diffusion: Open-source

Video Generation:

- Runway: Video editing
- Synthesia: AI avatars
- Luma: 3D generation

Code & Development

Code Generation:

- GitHub Copilot: AI pair programmer
- CodeT5: Code understanding
- AlphaCode: Competitive programming

Software Engineering:

- Automated testing
- Bug detection
- Code refactoring
- Documentation generation

Low-code Platforms:

- Natural language to app
- Automated UI generation
- Database query generation

Science & Research

Drug Discovery:

- Molecule generation
- Protein folding (AlphaFold)
- Clinical trial optimization

Scientific Writing:

- Literature review
- Hypothesis generation
- Grant proposal writing

Data Analysis:

- Automated insights
- Report generation
- Visualization creation

Generative AI is revolutionizing how we create, code, and conduct research

Generative AI transforms creative industries - text, image, code, and scientific generation achieve practical utility across domains

Ethical Considerations in Generative AI

Key Challenges

Bias and Fairness:

- Training data bias propagation
- Underrepresentation of groups
- Stereotypical outputs
- Need for diverse datasets

Misinformation:

- Deepfakes and synthetic media
- Convincing false information
- Difficulty in detection
- Social and political implications

Copyright and Ownership:

- Training on copyrighted content
- Generated content ownership
- Artist and creator rights
- Legal framework gaps

Privacy Concerns:

- Memorization of training data

Mitigation Strategies

Technical Solutions:

- Bias detection and mitigation
- Differential privacy
- Adversarial testing
- Content authentication

Regulatory Approaches:

- AI governance frameworks
- Content labeling requirements
- Transparency mandates
- Algorithmic auditing

Industry Standards:

- Responsible AI principles
- Ethical review boards
- Impact assessments
- Stakeholder engagement

Education and Awareness:

- Media lit

The Future of Generative AI

Technical Frontiers

Multimodal Models:

- GPT-4V: Vision and language
- DALL-E 3: Improved text-image
- Video-language models
- Audio-visual generation

Efficiency Improvements:

- Model compression techniques
- Efficient architectures
- Faster sampling methods
- Edge deployment

Controllability:

- Fine-grained control
- Style and content separation
- Interactive generation
- User-guided creation

Reasoning Capabilities:

- Chain-of-thought reasoning

Societal Impact

Creative Industries:

- AI-human collaboration
- New creative workflows
- Democratized content creation
- Novel art forms

Education:

- Personalized learning content
- AI tutoring systems
- Automated assessment
- Curriculum generation

Business Transformation:

- Automated content pipelines
- Personalized marketing
- Customer service automation
- Product design assistance

Key Insight: Generative AI will become increasingly integrated into daily life, requiring thoughtful

Appendix: Mathematical Foundations

Essential Mathematics for Machine Learning

Vector Operations

Dot Product:

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i = ||\mathbf{a}||||\mathbf{b}|| \cos\theta$$

Vector Norm:

$$||\mathbf{x}||_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}$$

Matrix Multiplication:

$$(AB)_{ij} = \sum_{k=1}^{m} A_{ik} B_{kj}$$

Matrix Inverse:

$$AA^{-1} = A^{-1}A = I$$

Transpose Properties:

Eigendecomposition

Eigenvalue Equation:

$$A\mathbf{v} = \lambda\mathbf{v}$$

Characteristic Polynomial:

$$\det(A - \lambda I) = 0$$

Diagonalization:

$$A = P\Lambda P^{-1}$$

where P contains eigenvectors, Λ contains eigenvalues

Singular Value Decomposition:

$$A = U\Sigma V^T$$

where U, V are orthogonal, Σ is diagonal

Multivariable Calculus

Gradient:

$$\nabla f(x) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n} \right)$$

Chain Rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial u} \frac{\partial u}{\partial x}$$

Hessian Matrix:

$$H_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}$$

Taylor Expansion:

$$f(x + h) \approx f(x) + \nabla f(x)^T h + \frac{1}{2} h^T H h$$

Directional Derivative:

$$D_{\mathbf{u}} f = \nabla f \cdot \mathbf{u}$$

Optimization

Gradient Descent:

$$\theta_{t+1} = \theta_t - \eta \nabla L(\theta_t)$$

Newton's Method:

$$\theta_{t+1} = \theta_t - H^{-1} \nabla L(\theta_t)$$

Convexity: A function f is convex if:

$$f(\lambda x + (1 - \lambda) y) \leq \lambda f(x) + (1 - \lambda) f(y)$$

Lagrange Multipliers:

$$L(x, \lambda) = f(x) + \lambda g(x)$$

Optimality Conditions:

$$\nabla_x L = 0, \quad \nabla_\lambda L = 0$$

Basic Probability

Bayes' Theorem:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Expectation:

$$E[X] = \sum_x xP(X = x)$$

Variance:

$$\text{Var}(X) = E[X^2] - (E[X])^2$$

Covariance:

$$\text{Cov}(X, Y) = E[XY] - E[X]E[Y]$$

Independence:

$$P(X, Y) = P(X)P(Y)$$

Distributions

Gaussian (Normal):

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Multivariate Gaussian:

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{k/2}|\Sigma|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})}$$

Bernoulli:

$$P(X = k) = p^k(1-p)^{1-k}$$

Exponential Family:

$$p(x|\theta) = h(x)e^{\eta(\theta)^T T(x) - A(\theta)}$$

Probability theory quantifies uncertainty - distributions and expectations provide the statistical foundation for inference

Core Concepts

Entropy:

$$H(X) = -\sum_x P(x) \log P(x)$$

Cross-Entropy:

$$H(p, q) = -\sum_x p(x) \log q(x)$$

KL Divergence:

$$D_{KL}(p||q) = \sum_x p(x) \log \frac{p(x)}{q(x)}$$

Mutual Information:

$$I(X; Y) = \sum_{x,y} p(x, y) \log \frac{p(x, y)}{p(x)p(y)}$$

Properties:

Applications in ML

Maximum Likelihood:

$$\theta^* = \underset{\theta}{\text{argmax}} \sum_i \log p(x_i|\theta)$$

Equivalent to minimizing cross-entropy

Information Gain (Decision Trees):

$$IG = H(S) - \sum_v \frac{|S_v|}{|S|} H(S_v)$$

Variational Inference:

$$\log p(x) \geq \mathbb{E}_q[\log p(x, z)] - \mathbb{E}_q[\log q(z)]$$

Regularization via Information:

- Information bottleneck principle
- Minimum description length
- Occam's razor formalization

PAC Learning

Probably Approximately Correct: With probability $\geq 1 - \delta$, a learning algorithm outputs hypothesis h such that:

$$R(h) \leq R^*(h) + \epsilon$$

Sample Complexity:

$$m \geq \frac{1}{\epsilon}\left(\log|H| + \log\frac{1}{\delta}\right)$$

for finite hypothesis class H

VC Dimension: Largest set size that can be shattered by hypothesis class

Generalization Bound:

$$R(h) \leq \hat{R}(h) + \sqrt{\frac{d\log(m/d) + \log(1/\delta)}{m}}$$

where d is VC dimension

Regularization Theory

Structural Risk Minimization:

$$h^* = \underset{h}{\operatorname{argmin}}\left[\hat{R}(h) + \lambda\Omega(h)\right]$$

Rademacher Complexity:

$$\mathfrak{R}_m(F) = \mathbb{E}_\sigma\left[\sup_{f\in F}\frac{1}{m}\sum_{i=1}^{m}\sigma_i f(x_i)\right]$$

Concentration Inequalities:

Hoeffding's Inequality:

$$P(|\hat{\mu} - \mu| \geq t) \leq 2e^{-2mt^2}$$

McDiarmid's Inequality: For bounded differences, concentration around expectation

Statistical learning theory provides generalization guarantees - PAC bounds quantify sample complexity and algorithm performance

Optimization Algorithms

Stochastic Gradient Descent:

$$\theta_{t+1} = \theta_t - \eta_t \nabla L_i(\theta_t)$$

Momentum:

$$v_t = \gamma v_{t-1} + \eta \nabla L(\theta_t)$$

$$\theta_{t+1} = \theta_t - v_t$$

AdaGrad:

$$G_t = G_{t-1} + (\nabla L(\theta_t))^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \nabla L(\theta_t)$$

Adam:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla L(\theta_t)$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla L(\theta_t))^2$$

Numerical Stability

Softmax Numerical Stability:

$$\text{softmax}(x_i) = \frac{e^{x_i - \max_j x_j}}{\sum_j e^{x_j - \max_j x_j}}$$

Log-Sum-Exp Trick:

$$\log \sum_i e^{x_i} = a + \log \sum_i e^{x_i - a}$$

where $a = \max_i x_i$

Gradient Clipping:

$$\mathbf{g} = \begin{cases} \mathbf{g} & ||\mathbf{g}|| \leq \theta \\ \frac{\theta}{||\mathbf{g}||}\mathbf{g} & ||\mathbf{g}|| > \theta \end{cases}$$

Numerical Precision:

- Float32 vs Float64 tradeoffs
- Catastrophic cancellation
- Conditioning and stability