Machine Learning Interview Study Sheet

This document contains topics I have encountered during my M.Sc. and Ph.D. studies, and are (somewhat) commonly asked in machine learning and deep learning interviews. The document is written to convey the gist of topics, and a number of preliminaries, that can be used as a reminder when studying the material. It is not intended to capture all details of the approaches. If you find mistakes, please send them to `jechterh@ucsd.edu`.

Contents

1 Probability and Statistics for Machine Learning **5**

1.1 Type 1 and Type 2 Error . . . 5
1.2 Probability Distributions . . . 5
1.3 Common Conjugate Priors . . . 6
1.4 Maximum Likelihood Estimation (MLE) . . . 6
1.5 Bayesian Inference and Maximum A Posteriori (MAP) . . . 6
1.6 Kernel Density Estimation (KDE) . . . 6
1.7 KL Divergence and ELBO . . . 7
1.8 Expectation Maximization (EM) . . . 7
1.9 Markov Random Fields (MRFs) (Markov Networks) . . . 7
1.10 Probability Math Rules . . . 7
1.10.1 Independence in Probability . . . 7
1.10.2 Set Rules . . . 8
1.10.3 Bayes Rule . . . 8
1.10.4 Partition Function . . . 8
1.10.5 Complement Rule . . . 8
1.10.6 Variance and Probability Rules . . . 8
1.11 Frequentist vs Bayesian Approaches . . . 8
1.12 Cumulative Distribution Function (CDF) vs Probability Density Function (PDF) . . . 9
1.13 Central Limit Theorem . . . 9
1.14 Law of Large Numbers . . . 9
1.15 Random Variables . . . 9
1.16 IID . . . 9
1.17 KDE . . . 9
1.18 Bayesian Information Criterion (BIC) . . . 10
1.19 Collinearity in Regression . . . 10
1.20 Bayesian Networks . . . 10
1.21 Variational Inference . . . 10
1.22 Gibbs Sampling . . . 10
1.23 Latent Dirichlet Allocation (LDA) . . . 11

1.24 Hypothesis Testing . . . 11
1.24.1 Z-Test . . . 11
1.24.2 T-Test . . . 12
1.24.3 Chi-Square Test . . . 12

2 Linear Algebra in Machine Learning 13
2.1 Orthogonality/-normality . . . 13
2.2 Determinant: . . . 13
2.3 Rank: . . . 13
2.4 Eigenvalues and Eigenvectors: . . . 13
2.5 Singular Values: . . . 13
2.6 Jacobian: . . . 13
2.7 Hessian: . . . 14
2.8 Taylor Series Approximation . . . 14
2.9 Matrix Operations in ML . . . 14
2.10 Derivative Rules . . . 14
2.11 Optimization with Constraints . . . 15
2.12 Lipschitz Continuity . . . 15
2.13 Jensen's Inequality . . . 15

3 Traditional Machine Learning 16
3.1 Imbalanced Data Techniques . . . 16
3.2 Bias-Variance Trade-Off . . . 16
3.3 K-Nearest Neighbors (KNN) . . . 16
3.4 Decision Trees . . . 16
3.5 Support Vector Machines (SVM) . . . 17
3.6 Matrix Factorizations . . . 17
3.7 Principal Component Analysis (PCA) . . . 17
3.8 Curse of Dimensionality . . . 17
3.9 Logistic and Linear Regression . . . 18
3.10 Bayesian Linear Regression . . . 18
3.11 Ensemble Methods . . . 18
3.12 XGBoost . . . 19
3.13 Performance Metrics . . . 19
3.14 Clustering and Distance Metrics . . . 19
3.15 Regularization . . . 20
3.16 Naive Bayes Classification . . . 20
3.17 Data Augmentation . . . 20
3.18 Data Synthesis . . . 20
3.19 Overfitting Prevention . . . 21
3.20 Gaussian Processes (GP) . . . 21
3.21 Feature Selection . . . 21
3.22 Uncertainty Quantification . . . 21
3.23 Clustering . . . 22
3.24 Gaussian Mixture Models (GMM) . . . 22
3.24.1 t-SNE (Stochastic Neighbor Embedding) . . . 22
3.24.2 Spectral Clustering . . . 22
3.24.3 Density-Based Spatial Clustering (DBSCAN) . . . 23

3.24.4 Self-Organizing Maps (SOM) . . . 23
3.25 Feature Scaling . . . 24

4 Deep Learning **25**
4.1 Backpropagation Algorithm . . . 25
4.2 Gradient Descent Variants . . . 25
4.3 Gradient Dynamics . . . 25
4.4 Gradient Accumulation . . . 26
4.5 Gradient Checkpointing . . . 26
4.6 Weight Initialization . . . 26
4.7 Loss Functions . . . 26
4.8 Activation Functions . . . 28
4.9 Layer Normalization . . . 28
4.10 Batch Normalization . . . 29
4.11 Dropout . . . 29
4.12 Learning Rates . . . 29
4.13 Optimizers . . . 29
4.14 Weight Initialization . . . 30
4.14.1 Break Symmetry Problem . . . 30
4.15 Regularization Techniques . . . 30
4.16 Learning Rate Schedulers . . . 30
4.17 Adaptive Gradient Algorithm (AdaGrad) . . . 31
4.18 Transformer Architecture . . . 31
4.18.1 Positional Encoding . . . 31
4.18.2 Attention Mechanisms . . . 32
4.18.3 Transformer Architectures . . . 32
4.19 Speculative Decoding . . . 32
4.20 Vision Transformers (ViTs) . . . 32
4.21 Convolutional Neural Networks (CNNs) . . . 33
4.22 Autoencoder . . . 33
4.23 Variational Autoencoders (VAEs) . . . 33
4.24 LoRA . . . 34
4.25 Reconstruction Loss and Regularization . . . 34
4.26 Generative Adversarial Networks (GANs) . . . 34
4.27 Jensen-Shannon Divergence . . . 34
4.28 Gumbel-Softmax Trick . . . 35
4.29 YOLO (You Only Look Once) . . . 35
4.30 Evaluation Metrics for GANs . . . 35
4.31 Conditional GANs (cGANs) . . . 35
4.32 Knowledge Distillation . . . 36
4.33 Mixture of Experts . . . 36
4.34 Recurrent Neural Networks (RNNs) . . . 36
4.35 LSTM . . . 36
4.36 Stable Diffusion . . . 37
4.37 Network Pruning . . . 37
4.38 UNET: CNN for Image Segmentation . . . 37
4.39 Graph Neural Networks (GNNs) . . . 38
4.40 Retrieval-Augmented Generation (RAG) Systems . . . 38

4.41 Hebbian Learning . . . 38
4.42 Hopfield Networks . . . 39
4.43 Restricted Boltzmann Machines (RBMs) . . . 39
4.44 Belief Propagation . . . 39
4.45 Deep Belief Networks (DBNs) . . . 39
4.46 Interpretable Model-Agnostic Explanations . . . 40
4.47 Internal Covariate Shift . . . 40
4.48 Saliency Maps . . . 40
4.49 Parameter Tying . . . 41

5 Reinforcement Learning **42**
5.1 Model Based RL . . . 42
5.2 Value vs. Policy Based RL . . . 42
5.3 Markov Decision Processes (MDPs) . . . 42
5.4 Markov Chain . . . 43
5.5 Hidden Markov Models (HMMs) . . . 43
5.6 Markov Decision Processes (MDPs) . . . 43
5.7 Multi-Armed Bandits: . . . 43
5.8 SARSA: . . . 44
5.9 Policy Gradient . . . 44
5.10 TD Learning . . . 44
5.11 Reinforce . . . 45
5.12 Q-Learning: . . . 45
5.13 Deep Q-Learning . . . 46
5.14 Actor-Critic Methods: . . . 46
5.15 Advantage Actor Critic . . . 47
5.16 Monte Carlo Methods: . . . 47
5.17 Monte Carlo Tree Search (MCTS): . . . 47
5.18 AlphaGo . . . 47
5.19 Proximal Policy Optimization (PPO) . . . 47
5.20 Trust Region Policy Optimization (TRPO) . . . 48
5.21 RLHF . . . 48

1 Probability and Statistics for Machine Learning

1.1 Type 1 and Type 2 Error

- Type 1 error: False Positives
- Type 2 error: False Negatives

1.2 Probability Distributions

- **Beta:**
- **Bernoulli:** Takes the value 1 with probability p and the value 0 with probability $q = 1 - p$

$$f(k; p) = p^k(1-p)^{1-k} \quad \text{for } k \in \{0, 1\}$$

- **Normal Distribution:**
  - Probability density function (PDF):

$$f(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

  - Defined by mean μ and variance σ^2.
- **Poisson Distribution:**
  - Models the number of events occurring in a fixed interval.
  - PMF:

$$P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!},$$

    where λ is the average event rate.
- **Gamma Distribution:**
  - Used to model waiting times.
  - PDF:

$$f(x|\alpha, \beta) = \frac{\beta^\alpha x^{\alpha-1} e^{-\beta x}}{\Gamma(\alpha)},$$

    where α is the shape parameter and β is the rate parameter.
- **Multivariate Normal Distribution:**
  - Generalization of the normal distribution to multiple variables.
  - PDF:

$$f(x|\mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^k |\Sigma|}} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)},$$

    where μ is the mean vector and Σ is the covariance matrix.
- **Wishart Distribution:**
  - Distribution of covariance matrices.
  - PDF is used in Bayesian statistics and multivariate analysis.

1.3 Common Conjugate Priors

Likelihood	Prior	Posterior
Binomial/Bernoulli	Beta	Beta
Normal	Normal	Normal
Normal	Normal-Inverse-Gamma	Normal-Inverse-Gamma
Poisson	Gamma	Gamma
Categorical/Multinomial	Dirichlet	Dirichlet
Multivariate Normal	Wishart	Wishart

Table 1: Common Conjugate Priors

1.4 Maximum Likelihood Estimation (MLE)

- Determines parameters θ that maximize the likelihood of observing given data:

$$\hat{\theta} = \arg\max_{\theta} P(D|\theta),$$

  where D is the observed data.

- Often uses the log-likelihood for simplicity:

$$\ell(\theta) = \log P(D|\theta).$$

- Assumes data comes from a known distribution.

1.5 Bayesian Inference and Maximum A Posteriori (MAP)

- MAP estimation incorporates a prior distribution $P(\theta)$:

$$\hat{\theta}_{MAP} = \arg\max_{\theta} P(\theta|D) = \arg\max_{\theta} P(D|\theta)P(\theta).$$

- Balances likelihood and prior knowledge.

1.6 Kernel Density Estimation (KDE)

- Non-parametric method to estimate the probability density function of a random variable.

- KDE formula:

$$\hat{f}(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right),$$

  where K is the kernel function, h is the bandwidth, and x_i are the data points.

- Common kernels include Gaussian, Epanechnikov, and Uniform.

- Bandwidth h controls the smoothness of the estimate.

1.7 KL Divergence and ELBO

- KL Divergence measures the difference between two distributions P and Q:

$$D_{KL}(P\|Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)}.$$

- Evidence Lower Bound (ELBO):

$$\mathcal{L}(q) = \mathbb{E}_{q(z)}[\log P(x|z)] - D_{KL}(q(z)\|P(z)).$$

  Used in variational inference to approximate posterior distributions.

1.8 Expectation Maximization (EM)

- Useful for problems with latent unobserves variables (e.g. cluster assignments)
- Iterative method to find the Maximum Likelihood Estimates (MLE) of parameters in probabilistic models with latent variables.
- **E-Step:** Compute the expected value of the log-likelihood with respect to the conditional distribution of the latent variables:

$$Q(\theta|\theta^{(t)}) = \mathbb{E}_{z\sim p(z|x,\theta^{(t)})}[\log p(x, z|\theta)].$$

- **M-Step:** Maximize $Q(\theta|\theta^{(t)})$ with respect to θ to update the parameters:

$$\theta^{(t+1)} = \arg\max_\theta Q(\theta|\theta^{(t)}).$$

1.9 Markov Random Fields (MRFs) (Markov Networks)

- Models the joint distribution of a set of random variables using an undirected graph.
- Nodes represent random variables, edges represent dependencies between variables.
- Satisfies the Markov property: Each node is conditionally independent of all other nodes given its neighbors.
- Probability distribution is defined using cliques (fully connected subsets of nodes):

$$P(X) = \frac{1}{Z} \prod_{C\in\mathcal{C}} \phi_C(X_C),$$

  where ϕ_C is the potential function for clique C, and Z is the partition function ensuring normalization.

1.10 Probability Math Rules

1.10.1 Independence in Probability

- Two random variables X and Y are independent if:

$$P(X \cap Y) = P(X)P(Y).$$

- Independence implies that the occurrence of one event does not affect the probability of the other.

1.10.2 Set Rules

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$
$$P(A \cap B) = P(A|B)P(B)$$

1.10.3 Bayes Rule

$$P(A|B) = \frac{P(A \cap B)}{P(B)}, \text{ if } P(B) \neq 0,$$

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}, \text{ if } P(B) \neq 0.$$

1.10.4 Partition Function

- Ensures the distribution sums to 1:

$$Z = \sum_{X} \prod_{C \in \mathcal{C}} \phi_C(X_C).$$

- Used in probabilistic graphical models to normalize probabilities.

1.10.5 Complement Rule

- Relates the probability of an event to its complement:

$$P(A^c) = 1 - P(A).$$

1.10.6 Variance and Probability Rules

- **Variance:**

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

- Probability rules for distributions:
  - **Bernoulli:** $P(X = 1) = p$, $P(X = 0) = 1 - p$.
  - **Binomial:** $P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$.

1.11 Frequentist vs Bayesian Approaches

- **Frequentist:** Defines probability as the long-run frequency of events over repeated trials.
- **Bayesian:** Interprets probability as a degree of belief or uncertainty, updated using Bayes' theorem.
- Frequentist inference relies on sampling distributions, while Bayesian inference incorporates prior beliefs.

1.12 Cumulative Distribution Function (CDF) vs Probability Density Function (PDF)

- **PDF:** Describes the density of probabilities at each point (for continuous variables).
- **CDF:** Represents the cumulative probability that a random variable takes on a value less than or equal to x:

$$F(x) = P(X \leq x).$$

1.13 Central Limit Theorem

- The sum of independent random variables leads to a normal distribution, regardless of the original distribution
-

$$\sum_{i=1}^{n} \frac{X_i - n\mu}{\sqrt{n\sigma^2}}$$

1.14 Law of Large Numbers

The average of the results obtained from a large number of independent random samples converges to the true value, if it exists.

1.15 Random Variables

Random variables quantify uncertainty in probabilistic models.

1.16 IID

Each random variable has the same probability distribution as the others and all are mutually independent.

1.17 KDE

- Kernel Density estimation estimates the probability density function of random variable based on finite sample of data.
- Non-parametric
- Intuition: Place kernel (e.g. Gaussian "bump") at each data point and sum the kernels to estimate the density

$$\widehat{f}_h(x) = \frac{1}{n}\sum_{i=1}^{n} K_h(x - x_i) = \frac{1}{nh}\sum_{i=1}^{n} K\Big(\frac{x - x_i}{h}\Big),$$

- For n samples x; h width of the "bump"; K is symmetric, non negative function whose integral sums to 1

1.18 Bayesian Information Criterion (BIC)

- Model selection criterion among a finite set of models.
- Penalizes complexity to prevent overfitting:

$$\text{BIC} = k \log(n) - 2\ell,$$

where k is the number of parameters, n is the number of data points, and ℓ is the log-likelihood.

1.19 Collinearity in Regression

- Occurs when predictor variables in a regression model are highly correlated.
- Results in near-singularity of the design matrix, increasing variance of coefficient estimates.

1.20 Bayesian Networks

- Graphical model representing conditional dependencies between random variables using directed edges.
- Contains no directed cycles.
- Captures latent variables and topic distributions in collections of documents:

$$P(x_i|\text{Parents}(x_i)).$$

1.21 Variational Inference

- Approximates posterior distributions by minimizing the KL divergence:

$$q^*(z) = \arg\min_q D_{KL}(q(z)\|p(z|x)).$$

- Optimizes Evidence Lower Bound (ELBO):

$$\mathcal{L}(q) = \mathbb{E}_{q(z)}[\log P(x|z)] - D_{KL}(q(z)\|P(z)).$$

1.22 Gibbs Sampling

- A Markov Chain Monte Carlo (MCMC) method for sampling from a joint distribution $P(X_1, X_2, \ldots, X_n)$.
- Iteratively samples from the conditional distribution of each variable:

$$X_i^{(t+1)} \sim P(X_i|X_{-i}),$$

where X_{-i} represents all variables except X_i.

- Converges to the true joint distribution after sufficient iterations.
- Commonly used in Bayesian inference and probabilistic graphical models.

1.23 Latent Dirichlet Allocation (LDA)

- A generative probabilistic model used for topic modeling.
- Assumes each document is a mixture of topics, and each topic is a mixture of words.
- Parameters:
  - α: Dirichlet prior on the per-document topic distributions.
  - β: Dirichlet prior on the per-topic word distributions.
- Generative process:
  1. For each document d:
     - Sample topic distribution $\theta_d \sim \text{Dir}(\alpha)$.
     - For each word w in d:
       * Sample a topic $z \sim \text{Multinomial}(\theta_d)$.
       * Sample a word $w \sim \text{Multinomial}(\beta_z)$.
- Inference aims to estimate the hidden topic structure using methods like Variational Inference or Gibbs Sampling.

1.24 Hypothesis Testing

- Null hypothesis (H_0): Assumes no effect or relationship exists.
- Alternative hypothesis (H_a): Opposes the null hypothesis, indicating an effect or relationship.
- P-value measures the probability of observing results as extreme as the actual results under H_0.
- Reject H_0 if the p-value is less than the significance level α.

1.24.1 Z-Test

- Used for large samples where the population variance is known.
- Tests the null hypothesis that the sample mean is equal to a population mean:

$$Z = \frac{\bar{X} - \mu}{\sigma/\sqrt{n}},$$

  where $\bar{X}$ is the sample mean, μ is the population mean, σ is the population standard deviation, and n is the sample size.
- Assumes data is normally distributed.

1.24.2 T-Test

- Used for small samples where the population variance is unknown.
- Tests whether the means of two groups are significantly different:

$$T = \frac{\bar{X}_1 - \bar{X}_2}{s\sqrt{\frac{1}{n_1} + \frac{1}{n_2}}},$$

  where s is the pooled standard deviation.
- Assumes the samples are independent and drawn from normal distributions.

1.24.3 Chi-Square Test

- Tests for independence between categorical variables.
- Compares observed and expected frequencies:

$$\chi^2 = \sum \frac{(O - E)^2}{E},$$

  where O is the observed frequency and E is the expected frequency.
- Assumes expected frequencies are sufficiently large (e.g., $E \geq 5$).

2 Linear Algebra in Machine Learning

2.1 Orthogonality/-normality

- Orthogonal: $u \cdot v = 0$ for vectors u, v
- Orthonormal: Orthogonal and $||u|| = ||v|| = 1$

2.2 Determinant:

Indicates if a matrix is invertible. A square matrix A is singular if:

$$\det(A) = 0,$$

e.g.

$$A = \begin{vmatrix} a & b \\ c & d \end{vmatrix} \rightarrow det(A) = ad - bc,$$

which implies A has no inverse and is rank-deficient.

2.3 Rank:

Measures the number of linearly independent rows or columns.

- A rank-deficient matrix has redundant information.
- $\text{rank}(A) = \min(\text{rows}, \text{cols})$ for a full-rank matrix.

2.4 Eigenvalues and Eigenvectors:

- Solve $Av = \lambda v$ for scalar λ (eigenvalue) and vector v (eigenvector).
- Eigenvalues indicate the scaling in the direction of eigenvectors.

2.5 Singular Values:

Derived from the singular value decomposition (SVD):

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

where Σ contains the singular values, which measure the magnitude of each dimension.

2.6 Jacobian:

Matrix of first-order partial derivatives:

$$J_{ij} = \frac{\partial f_i}{\partial x_j}.$$

Useful for analyzing transformations and gradients.

2.7 Hessian:

Matrix of second-order partial derivatives:

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

Describes the curvature of a function, often used in optimization.

2.8 Taylor Series Approximation

- Approximates a function $f(x)$ near a point a:

$$f(x) \approx f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \dots$$

- Common in machine learning for approximating nonlinear functions locally.

2.9 Matrix Operations in ML

- **Dot Product:** Measures similarity between vectors:

$$a \cdot b = \sum_i a_i b_i.$$

- **Cross Product:** Produces a vector perpendicular to two input vectors (in 3D):

$$a \times b.$$

- **Orthogonality:** Two vectors a and b are orthogonal if:

$$a \cdot b = 0.$$

2.10 Derivative Rules

- **Chain Rule:**

$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx}.$$

- **Power Rule:**

$$\frac{d}{dx}[x^n] = nx^{n-1}.$$

- **Sum Rule:**

$$\frac{d}{dx}[f(x) + g(x)] = f'(x) + g'(x).$$

- **Quotient Rule**

$$h(x) = \frac{f(x)}{g(x)} \to h'(x) = \frac{f'(x)g(x) - f(x)g'(x)}{(g(x))^2}.$$

- **Product Rule**

$$(u \cdot v)' = u' \cdot v + u \cdot v'$$

- **Log Trick**:
$$f(x) = \ln(g(x)) \to f'(c) = \frac{g'(x)}{g(x)}$$
- **Approximation for small changes:**
$$f(x + \Delta x) \approx f(x) + f'(x)\Delta x$$

2.11 Optimization with Constraints

- **Lagrangian Optimization:** Combines the objective function and constraints into a single function:
$$\mathcal{L}(x, \lambda) = f(x) + \lambda g(x),$$
where $g(x) \leq 0$ represents constraints.
- Solution lies on the constraint surface where gradients of f and g align.
- Natural log trick simplifies optimization problems:
$$\ln(g(x)) \text{ for } g(x) > 0.$$

2.12 Lipschitz Continuity

- A function $f(x)$ is Lipschitz continuous if:
$$|f(x_1) - f(x_2)| \leq L\|x_1 - x_2\|,$$
where L is the Lipschitz constant.
- Ensures small changes in input lead to bounded changes in output.
- Used in stability and robustness analysis of machine learning models.

2.13 Jensen's Inequality

- For a convex function f:
$$f\left(\mathbb{E}[X]\right) \leq \mathbb{E}[f(X)],$$
where X is a random variable.
- Often applied in expectation maximization and probabilistic models.

3 Traditional Machine Learning

3.1 Imbalanced Data Techniques

- **Oversampling:** Increase the representation of minority classes by duplicating data.
- **Subsampling:** Reduce the majority class by sampling a subset of its data.
- **SMOTE:** Synthesize new examples for the minority class, interpolates between existing instances. Take minority class sample, find neighbor of another minority point and interpolate between.
- **Weighted Loss:** Assign higher penalties to minority class errors.
- **Ensemble Methods:** Combine multiple models to improve performance.
- **Use Balanced Metrics:** Focus on metrics like F1-score or AUC-ROC.

3.2 Bias-Variance Trade-Off

- **High Bias:** Leads to underfitting.
- **High Variance:** Leads to overfitting.
- **Reducing Bias:** Increase model complexity.
- **Reducing Variance:** Decrease model complexity or use regularization.

3.3 K-Nearest Neighbors (KNN)

- Supervised, feature-based separation.
- Calculate distance between the query point and all data points using

$$d(p, q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

- Select the k closest points and vote or average for classification or regression.

3.4 Decision Trees

- Split based on the best feature using Gini coefficient

$$G = 1 - \sum_{i=1}^{n} p_i^2$$

or entropy

$$H = -\sum_{i=1}^{n} p_i \log p_i$$

.

- Continue splitting until a stopping criterion is met.

3.5 Support Vector Machines (SVM)

- Maximize margin between classes:

$$f(x) = \text{sign}(w^T x + b)$$

- Trade-off parameter C controls margin and misclassification.
- Kernel trick for high-dimensional feature spaces, e.g., polynomial kernel

$$K(x, x') = (x^T x' + c)^d$$

or radial basis function (RBF) kernel

$$K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right).$$

3.6 Matrix Factorizations

- **SVD:**

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

where Σ contains singular values.

- **LU Decomposition:** Factorize A into LU for square matrices.
- **QR Decomposition:** Factorize A into QR, where Q is orthogonal and R is upper triangular.

3.7 Principal Component Analysis (PCA)

- Reduce dimensionality while retaining variance.
- Compute covariance matrix:

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

.

- Find eigenvectors and eigenvalues:

$$\Sigma v = \lambda v$$

.

- Select top components based on explained variance.

3.8 Curse of Dimensionality

- Data becomes sparse as dimensions increase.
- Distances lose meaning in high-dimensional spaces.

3.9 Logistic and Linear Regression

- **Logistic Regression:** Models the probability of binary outcomes using the log-odds:
$$\log\left(\frac{P(y=1|x)}{1-P(y=1|x)}\right) = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n.$$

- **Binary Cross-Entropy Loss:** For n samples:
$$\text{Loss} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(p_i) + (1-y_i)\log(1-p_i)\right].$$

- **Linear Regression:** Predicts continuous outputs by minimizing the Mean Squared Error (MSE):
$$\hat{y} = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n, \quad \text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2.$$

3.10 Bayesian Linear Regression

- Adds prior distributions to weights:
$$w \sim \mathcal{N}(0, \lambda^{-1} I).$$

- Posterior distribution:
$$P(w|X,y) \propto P(y|X,w)P(w).$$

- Predictive distribution:
$$P(\hat{y}|X) = \int P(\hat{y}|w,X)P(w|X,y)dw.$$

3.11 Ensemble Methods

- **Bagging:** Combines predictions from subsets of data to reduce variance.

- **Boosting:** Sequentially adjusts weights of samples to reduce bias, e.g., Gradient Boosting:
$$F_{m+1}(x) = F_m(x) + \eta \sum_{i=1}^{n} \nabla \text{Loss}(y_i, F_m(x_i)),$$
where η is the learning rate.

- **Stacking:** Combines multiple models using a meta-model.

3.12 XGBoost

- Emsemble gradient boosting, ensemble of trees
- Each tree corrects errors of previous tree
- **Goal:** Minimize loss by adding trees f_t in step t
- Uses 2nd order Taylor Approximation
- For each split, calculate the gain
- **Algorithm:**
  - Init model f_0 with constant
  - Compute gradients $\hat{g}$ and Hessians $\hat{h}$
  - Fit base learner using training set optimizing a base learner (or weak learner, e.g. tree) using the training set

$$\hat{\phi}_m = \arg\min_{\phi \in \mathbf{\Phi}} \sum_{i=1}^{N} \frac{1}{2} \hat{h}_m(x_i) \left[\phi(x_i) - \frac{\hat{g}_m(x_i)}{\hat{h}_m(x_i)} \right]^2.$$

  - Update model:

$$\hat{f}_{(m)}(x) = \hat{f}_{(m-1)}(x) - \hat{f}_m(x).$$

  - Output

$$\hat{f}(x) = \hat{f}_{(M)}(x) = \sum_{m=0}^{M} \hat{f}_m(x).$$

3.13 Performance Metrics

- **F1 Score:** Harmonic mean of precision and recall:

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

- **ROC-AUC:** Area under the Receiver Operating Characteristic curve.
- **Mean Average Precision (mAP):** Used for evaluating ranking and object detection tasks.

3.14 Clustering and Distance Metrics

- **Euclidean Distance:**

$$d(p, q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}.$$

- **Intersection Over Union (IoU):** Evaluates overlap in object detection:

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}.$$

3.15 Regularization

- **L2 Regularization:** Adds a penalty proportional to the square of coefficients:

$$\text{Loss} = \text{Original Loss} + \lambda \sum_{j=1}^{n} \beta_j^2.$$

- **L1 Regularization:** Adds a penalty proportional to the absolute values of coefficients:

$$\text{Loss} = \text{Original Loss} + \lambda \sum_{j=1}^{n} |\beta_j|.$$

3.16 Naive Bayes Classification

- Naive because: Assumes conditional independence of features.
- Calculate $P(X|C)$ using Gaussian, Multinomial, or Bernoulli distributions:

$$P(C|X) = \frac{P(X|C)P(C)}{P(X)}.$$

- Use Laplace smoothing to handle zero probabilities:

$$P(X|C) = \frac{\text{count}(X, C) + \alpha}{\text{count}(C) + \alpha \cdot \text{number of classes}}.$$

- choose $c = \text{argmax}_c P(C) \prod_i P(X_i|C)$

3.17 Data Augmentation

- Techniques:
  - Image: Cropping, Flipping, Rotation, Scaling, Shearing, Brightness, Contrast, Adding Noise, Padding.
  - Text: Synonym replacement, Back-translation.
  - Time-series: Scaling, Time-warping.
- Use generative models (e.g., GANs) to create new samples.

3.18 Data Synthesis

- Generate data from a known distribution.
- Methods:
  - Synthetic Minority Oversampling Technique (SMOTE).
  - Kernel Density Estimation (KDE).
  - Gaussian Mixture Models (GMMs).

3.19 Overfitting Prevention

- Reduce model complexity.
- Use regularization:
  - **L1 Regularization:** Adds a penalty proportional to the sum of absolute coefficients.
  - **L2 Regularization:** Adds a penalty proportional to the sum of squared coefficients.
- Apply dropout to randomly deactivate neurons.
- Ensembling: Combine predictions from multiple models.
- Cross-validation: Use a validation set to tune hyperparameters.

3.20 Gaussian Processes (GP)

- Non-parametric Bayesian model.
- Defines a distribution over functions:

$$f(x) \sim \mathcal{GP}(m(x), k(x, x')),$$

  where $m(x)$ is the mean function and $k(x, x')$ is the kernel (covariance function).
- Predictions are random variables with a multivariate Gaussian distribution.
- Kernels calculate covariance, e.g., RBF kernel:

$$k(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right).$$

3.21 Feature Selection

- Remove irrelevant or redundant features to improve model performance.
- Use methods like:
  - Regularization (L1, L2).
  - Recursive Feature Elimination (RFE).
  - Principal Component Analysis (PCA).

3.22 Uncertainty Quantification

- Quantify model confidence in predictions.
- Useful for small datasets or non-linear relationships.
- Incorporates prior knowledge using Bayesian methods.

3.23 Clustering

3.24 Gaussian Mixture Models (GMM)

- **Soft Clustering:** Assigns probabilities of membership for each point to different clusters.
- **Maximize Log-Likelihood:** Given data x, parameters μ_k and Σ_k, maximize:

$$\mathcal{L}(x) = \sum_{i=1}^{n} \log \left(\sum_{k=1}^{K} \pi_k \mathcal{N}(x_i | \mu_k, \Sigma_k) \right).$$

- Uses Expectation Maximization

3.24.1 t-SNE (Stochastic Neighbor Embedding)

- Projects data to lower dimensions while preserving pairwise similarities.
- Computes pairwise similarity probabilities in high dimensions:

$$P_{ij} = \frac{\exp(-\|x_i - x_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq l} \exp(-\|x_k - x_l\|^2 / 2\sigma_k^2)}.$$

- Maps points in lower dimensions using a t-distribution:

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

  Uses the t-distribution because it better models the distribution of pairwise distances in the low-dimensional space and effectively mitigates the "crowding problem" that arises when visualizing high-dimensional data.

- Minimizes the KL divergence between P_{ij} and Q_{ij}:

$$KL(P \| Q) = \sum_{i \neq j} P_{ij} \log \frac{P_{ij}}{Q_{ij}}.$$

- Optimized using gradient descent.
- Useful for non-linear data structures and preserving local structures.

3.24.2 Spectral Clustering

- Graph-based clustering method.
- Represents data points as nodes and similarities as edges.
- Constructs a similarity graph (e.g., RBF kernel):

$$W_{ij} = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2} \right).$$

- Computes graph Laplacian:

$$L = D - W, \quad \text{where } D_{ii} = \sum_j W_{ij}.$$

- Finds eigenvectors and eigenvalues of L.
- Uses rows of top k eigenvectors as features for clustering.
- Suitable for non-linearly separable data.

3.24.3 Density-Based Spatial Clustering (DBSCAN)

- Groups points based on density rather than distance.
- Parameters:
  - ϵ: Radius for neighborhood.
  - MinPts: Minimum number of neighbors to form a dense region.
- Identifies core points, border points, and noise.
- Advantages:
  - Robust to outliers.
  - Handles clusters of arbitrary shapes.

3.24.4 Self-Organizing Maps (SOM)

- Unsupervised neural network that preserves topological properties of data.
- Neuron weights arranged in a grid indexed by position.
- Process:
  1. Initialize weights randomly.
  2. Sample input data.
  3. Find best matching unit (BMU):

$$\text{BMU} = \arg\min_j \|x - w_j\|.$$

  4. Update weights of BMU and its neighbors:

$$w_j(t+1) = w_j(t) + \eta(t) h_{j,i}(t)(x - w_j(t)).$$

  5. Repeat until convergence.

3.25 Feature Scaling

- Normalize or standardize features to ensure equal weighting.
- Methods:
  - Min-Max Scaling:
$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}.$$
  - Standardization:
$$x' = \frac{x - \mu}{\sigma}.$$
- Benefits:
  - Faster convergence during optimization.
  - Reduces bias introduced by different feature ranges.

4 Deep Learning

4.1 Backpropagation Algorithm

- Used to compute gradients for neural network training.
- **Steps:**
  1. **Forward Pass:** Compute activations and output values layer by layer.
  2. **Loss Computation:** Calculate the error between predicted and true values using a loss function L.
  3. **Backward Pass:** Compute gradients of the loss with respect to weights using the chain rule:
$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial w}.$$
  4. **Weight Update:** Update weights using gradient descent:
$$w \leftarrow w - \eta \frac{\partial L}{\partial w},$$
where η is the learning rate.

4.2 Gradient Descent Variants

- **Gradient Descent:** Updates weights using gradients of the loss function.
- **Stochastic Gradient Descent (SGD):** Updates weights one data point at a time, reducing memory usage.
- **Momentum:** Accumulates past gradients to smooth updates and escape local minima:
$$v_t = \beta v_{t-1} + (1-\beta)\nabla L(w), \quad w \leftarrow w - \eta v_t.$$
- **Nesterov Accelerated Gradient:** Adjusts gradients using a lookahead mechanism:
$$w_t = w_{t-1} - \eta \nabla L(w_t - \eta v_t).$$
- **Gradient Clipping:** Limits the gradient magnitude to avoid exploding gradients:
$$\nabla L(w) \leftarrow \text{clip}(\nabla L(w), -c, c).$$

4.3 Gradient Dynamics

- Gradient points in the direction of steepest descent toward the local minimum.
- At equilibrium:
$$\nabla L(w) = 0$$
- Stable states occur at minima, unstable states at maxima and saddle points.

4.4 Gradient Accumulation

- Accumulate gradients over multiple steps to simulate a larger batch size when GPU memory is limited:

$$G_{accumulated} = \sum_{i=1}^{N} G_i$$

- Normalizes gradients per feature, especially in CNNs, for spatial invariance.

4.5 Gradient Checkpointing

Memory optimization technique which recomputes intermediate activations instead of storing them in memory (layer-wise).

4.6 Weight Initialization

- Zero init leads to failure and random init can lead to vanishing/exploding gradients
- **Xavier Initialization:** Scales weights by:

$$W \sim \mathcal{N}\left(0, \frac{1}{n_{\text{in}}}\right).$$

  and maintains variance of activations and gradients constantly to avoid gradients exploding/vanishing and each layer receives data with similar variance; for sigmoid and tanh

- **He Initialization:** Scales weights for ReLU activation:

$$W \sim \mathcal{N}\left(0, \frac{2}{n_{\text{in}}}\right).$$

4.7 Loss Functions

- **Mean Squared Error (MSE):**

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

- **Cross-Entropy Loss:**

$$L = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right].$$

- **Kullback-Leibler Divergence (KL Divergence):**

$$D_{KL}(P\|Q) = \sum_{i} P(i)\log\frac{P(i)}{Q(i)}.$$

  Measures how one probability distribution P diverges from a second distribution Q.

- **Hinge Loss:** Used for SVM:

$$L = \sum_{i=1}^{n} \max(0, 1 - y_i \hat{y}_i).$$

  Encourages correct classification with a margin of at least 1.

- **Huber Loss:** A loss function that is robust to outliers. Defined as:

$$L = \begin{cases} \frac{1}{2}(y_i - \hat{y}_i)^2, & \text{if } |y_i - \hat{y}_i| \leq \delta, \\ \delta \cdot |y_i - \hat{y}_i| - \frac{1}{2}\delta^2, & \text{otherwise.} \end{cases}$$

  The parameter δ controls the threshold for treating differences as outliers.

- **Contrastive Loss:** Used in tasks like metric learning. Defined as:

$$L = \frac{1}{2} \sum_{i=1}^{n} \Big(y_i d(x_i, x_j)^2 + (1 - y_i) \max(0, m - d(x_i, x_j))^2 \Big),$$

  where $d(x_i, x_j)$ is a distance metric, y_i is 1 if the pair is similar and 0 otherwise, and m is the margin.

- **Dice Loss:** Commonly used in segmentation tasks to maximize overlap between predicted and ground truth regions:

$$L = 1 - \frac{2 \sum_i p_i g_i}{\sum_i p_i^2 + \sum_i g_i^2},$$

  where p_i and g_i are the predicted and ground truth values, respectively.

- **Focal Loss:** Designed to address class imbalance in classification tasks:

$$L = -\alpha (1 - p_t)^{\gamma} \log(p_t),$$

  where p_t is the predicted probability for the correct class, α is a weighting factor, and γ controls the down-weighting of well-classified examples.

- **Triplet Loss:** Encourages embeddings to bring similar pairs closer while pushing dissimilar ones apart:

$$L = \sum_{i=1}^{n} \max(0, d(a_i, p_i) - d(a_i, n_i) + \alpha),$$

  where a_i, p_i, and n_i represent anchor, positive, and negative samples, $d(\cdot)$ is a distance metric, and α is the margin.

- **Cosine Similarity:** Measures the similarity between two vectors:

$$\text{Cosine Similarity}(u, v) = \frac{u \cdot v}{\|u\| \|v\|},$$

  where $u \cdot v$ is the dot product of u and v, and $\|u\|$ and $\|v\|$ are their magnitudes.

4.8 Activation Functions

- **Sigmoid:** $f(z) = \frac{1}{1+e^{-z}}$.
- **ReLU:** $f(z) = \max(0, z)$.
- **Leaky ReLU:** $f(z) = \max(\alpha z, z), \alpha > 0$.
- **GELU:** $f(z) = z\Phi(z)$, where $\Phi(z)$ is the Gaussian CDF.
- **Softmax:** Converts logits into probabilities:

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}.$$

- **ELU (Exponential Linear Unit):** An activation function designed to improve gradient flow. Defined as:

$$\text{ELU}(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha(e^x - 1), & \text{if } x \leq 0 \end{cases}$$

  where $\alpha > 0$ is a parameter controlling the function's negative saturation.

- **Tanh (Hyperbolic Tangent):** A non-linear activation function that maps inputs to the range $(-1, 1)$. Defined as:

$$\text{Tanh}(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

  It is symmetric around the origin and often used when zero-centered outputs are desired.

- **Swish:** A smooth, non-monotonic activation function that often outperforms ReLU in practice. Defined as:

$$\text{Swish}(x) = x \cdot \text{Sigmoid}(\beta x) = \frac{x}{1 + e^{-\beta x}},$$

  where β is a trainable or fixed parameter. When $\beta = 1$, it simplifies to $\text{Swish}(x) = x \cdot \text{Sigmoid}(x)$.

4.9 Layer Normalization

- Normalizes the outputs of a layer, each sample across features.
- Typically used in Transformers
- Normalizes activations to prevent vanishing/exploding gradients:

$$\hat{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}.$$

4.10 Batch Normalization

- Normalize per feature
- For CNN due to spatial invariance
- Reduces internal covariate shift and helps gradient flow

4.11 Dropout

Randomly deactivates neurons during training.

4.12 Learning Rates

- Fixed learning rates may overshoot or converge too slowly.
- Step Decay:
$$\eta_t = \eta_0 \cdot \gamma^{\lfloor t/T \rfloor}$$
- Cyclical Learning Rates:
$$\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min})(1 + \cos(\frac{t\pi}{T})).$$
- Warm-up Learning Rates: Gradually increase the learning rate over a predefined number of steps.

4.13 Optimizers

- **SGD:**
- **Adam:** Combines RMSprop and momentum with bias correction:
$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$
$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$
$$w \leftarrow w - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}.$$
- **RMSprop:** Uses a moving average of squared gradients:
$$v_t = \beta v_{t-1} + (1 - \beta) g_t^2, \quad w \leftarrow w - \eta \frac{g_t}{\sqrt{v_t} + \epsilon}.$$

4.14 Weight Initialization

- Proper initialization prevents vanishing or exploding gradients:
  - **Xavier Initialization:** For sigmoid/tanh activations:

$$W \sim \mathcal{N}\left(0, \frac{1}{n_{\text{in}}}\right).$$

  - **He Initialization:** For ReLU activations:

$$W \sim \mathcal{N}\left(0, \frac{2}{n_{\text{in}}}\right).$$

- Uniform initialization is also an option.

4.14.1 Break Symmetry Problem

If two hidden units with same activation are connected to same inputs, they must have different parameters as otherwise they will be updated the same way.

4.15 Regularization Techniques

- **L1 Regularization:** Encourages sparsity in weights:

$$L = L_0 + \lambda \sum |w_i|.$$

- **L2 Regularization:** Penalizes large weights:

$$L = L_0 + \lambda \sum w_i^2.$$

- **Elastic Net:** Combines L1 and L2:

$$L = L_0 + \lambda_1 \sum |w_i| + \lambda_2 \sum w_i^2.$$

- **Dropout:** Randomly deactivates neurons during training to reduce overfitting.
- **Batch Normalization:** Normalizes activations within a batch to stabilize and accelerate training:

$$\hat{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}.$$

4.16 Learning Rate Schedulers

- **Step Decay:** Reduces learning rate by a factor after a fixed number of epochs.
- **Cyclic Scheduler:** Oscillates learning rates between minimum and maximum bounds.
- **Warm-up and Cool-down:** Gradually increase or decrease learning rates during training.

4.17 Adaptive Gradient Algorithm (AdaGrad)

- Adjusts learning rates based on the accumulation of squared gradients:

$$G_t = \sum_{i=1}^{t} g_i^2, \quad \eta_t = \frac{\eta}{\sqrt{G_t} + \epsilon}.$$

- Parameters with larger gradients accumulate higher G_t, resulting in smaller learning rates.
- Parameters with smaller gradients retain larger learning rates.
- Benefits:
  - Handles sparse gradients well.
  - Automatically adapts learning rates during training.

4.18 Transformer Architecture

- **Input Embedding:** Maps tokens to high-dimensional vectors.
- **Positional Encoding:** Encodes token positions using sine and cosine functions:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right), \quad PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right).$$

- **Multi-Head Attention:** Computes attention scores using queries Q, keys K, and values V:

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

- **Residual Connections:** Adds the input back to the output to address vanishing gradients:

$$y = x + f(x).$$

- **Feed-Forward Neural Network:** Applies transformations independently to each token.

4.18.1 Positional Encoding

- Captures token order using sine and cosine functions.
- Periodic encoding allows differentiation between relative positions.
- Added to token embeddings before feeding into attention layers.

4.18.2 Attention Mechanisms

- Scales dot-product attention to prevent large values:

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

- **Multi-Head Attention:** Splits attention into multiple heads for parallelization:

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

- Heads are concatenated and projected using W^O:

$$\text{output} = \text{concat}([\text{head}_1, \ldots, \text{head}_h])W^O.$$

4.18.3 Transformer Architectures

- **Longformer:** Efficient attention mechanism for long sequences:
  - Combines sliding window attention with global attention.
  - Reduces computational complexity compared to standard transformers.
- **Decoder-only vs. Encoder-Decoder:**
  - **BERT:** Encoder-only for understanding tasks.
  - **GPT:** Decoder-only for generation tasks.
  - **Encoder-Decoder:** Suitable for sequence-to-sequence tasks.

4.19 Speculative Decoding

- Speed up autoregressive text generation
- Use small model to speculate and large model to confirm
- Uses log probability for evaluation

4.20 Vision Transformers (ViTs)

- **Architecture:**
  - Input image divided into patches of size $P \times P$.
  - Each patch flattened into a vector and linearly embedded.
  - Learnable positional encodings added to embeddings.
- **Transformer Encoder:** Processes embeddings with:
  - Multi-head attention layers.
  - Feed-forward layers with residual connections and layer normalization.
- **Classification:** Prepend a special [CLS] token to embeddings.
  - Final hidden state of [CLS] used as input to classifier head.

4.21 Convolutional Neural Networks (CNNs)

- **Convolutional Layer:** Applies filters to extract spatial features:

$$y = \text{conv}(x, W) + b.$$

- **Pooling Layer:** Reduces spatial dimensions and computations while maintaining shift invariance.
- **Key Features:**
  - **Local connectivity:** Each neuron connects to a local region
  - **Translation invariance:** Pooling makes CNN robust to shifts and distortions.
  - **Hierarchical feature extraction:** From low to high level
  - **Objects are made of parts; Receptive Field:** Neurons process information from a local region, gets bigger deeper in the net.
  - **Weight Sharing:** Filters are shared across input regions.
- **Applications:**
  - Hierarchical feature extraction from low to high levels.
  - Object detection and image recognition.

4.22 Autoencoder

- Minimize reconstruction loss

$$||x - \hat{x}||^2$$

- Encoder Decoder architectures are neural networks

4.23 Variational Autoencoders (VAEs)

- **Architecture:**
  - **Encoder:** Maps input x to latent space z using a posterior distribution:

$$q(z|x) \sim \mathcal{N}(\mu(x), \sigma^2(x)).$$

  - **Decoder:** Reconstructs x from z by modeling:

$$p(x|z) \sim \mathcal{N}(\hat{x}, \sigma^2).$$

- **Objective: Evidence Lower Bound (ELBO):**

$$\mathcal{L}_{\text{ELBO}} = \mathbb{E}_{q(z|x)}[\log p(x|z)] - D_{KL}(q(z|x)\|p(z)).$$

  - Reconstruction term: $\mathbb{E}_{q(z|x)}[\log p(x|z)]$.
  - Regularization term: $D_{KL}(q(z|x)\|p(z))$ enforces Gaussian prior.
- **Reparameterization Trick:** Rewrites $z \sim q(z|x)$ as:

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

4.24 LoRA

- Pretrained $W \in \mathbb{R}^{d \times k}$
- Learn $A \in \mathbb{R}^{d \times r}$; $B \in \mathbb{R}^{r \times k}$
- $W' = W + \Delta W$; $\Delta W = AB$
- Number of params $r(d + k)$
- Typically query and projection matrix in transformer

4.25 Reconstruction Loss and Regularization

- Reconstruction loss measures difference between input x and output $\hat{x}$:

$$\mathcal{L}_{\text{recon}} = \|x - \hat{x}\|^2.$$

- Regularization term encourages latent space structure.

4.26 Generative Adversarial Networks (GANs)

- **Minimax Objective:**

$$\min_G \max_D \mathbb{E}_{x \sim p_{\text{data}}}[\log D(x)] + \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z)))].$$

- **Components:**
  - **Generator (G):** Takes random noise z and generates data $G(z)$ to fool the discriminator.
  - **Discriminator (D):** Classifies data as real or fake.
- **Challenges:**
  - Mode collapse: Generator produces limited outputs.
  - Training instability.

4.27 Jensen-Shannon Divergence

- Measuring the similarity between two probability distributions
- Based on the Kullback–Leibler divergence, with some notable differences, including that it is symmetric and it always has a finite value
- $$\text{JSD}(P \parallel Q) = \frac{1}{2} D(P \parallel M) + \frac{1}{2} D(Q \parallel M),$$

  with

$$M = \frac{1}{2}(P + Q)$$

4.28 Gumbel-Softmax Trick

- Differentiable approximation to sampling from a categorical distribution.
- Adds Gumbel noise g_i to logits $\log p_i$:

$$y_i = \frac{\exp((\log p_i + g_i)/\tau)}{\sum_j \exp((\log p_j + g_j)/\tau)}.$$

- Temperature τ controls sharpness of the distribution.

4.29 YOLO (You Only Look Once)

- Single-stage object detection framework.
- **Process:**
  - Divide image into grid cells.
  - Each cell predicts bounding boxes, class probabilities, and confidence scores.
- **Loss Function:**

$$\mathcal{L} = \mathcal{L}_{\text{coord}} + \mathcal{L}_{\text{conf}} + \mathcal{L}_{\text{class}}.$$

- Uses a CNN backbone (e.g., Darknet) for feature extraction.

4.30 Evaluation Metrics for GANs

- **Inception Score (IS):** Evaluates image quality and diversity.

$$\text{IS} = \exp\left(\mathbb{E}_x[D_{KL}(p(y|x)\|p(y))]\right).$$

- **Fréchet Inception Distance (FID):** Measures similarity between real and generated distributions:

$$\text{FID}(X_r, X_g) = \|\mu_r - \mu_g\|^2 + \text{Tr}(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{1/2}).$$

4.31 Conditional GANs (cGANs)

- Conditions generation on additional information such as labels or classes.
- Generator and discriminator receive the label y as input:

$$G(z, y), \quad D(x, y).$$

- Objective function:

$$\min_G \max_D \mathbb{E}_{x,y\sim p_{\text{data}}}[\log D(x, y)] + \mathbb{E}_{z\sim p_z, y\sim p_y}[\log(1 - D(G(z, y), y))].$$

4.32 Knowledge Distillation

- Enables a smaller student model to learn from a larger teacher model by mimicking its outputs.
- Distillation loss combines soft predictions from the teacher and supervised loss:

$$\mathcal{L}_{\text{distill}} = \alpha \cdot \text{KL}(p_t \| p_s) + (1 - \alpha) \cdot \mathcal{L}_{\text{CE}}.$$

- Applications:
  - Model compression.
  - Transfer learning.
  - Multi-modal learning.

4.33 Mixture of Experts

- Combines multiple sub-models (experts) for specific tasks.
- Gating network assigns probabilities to experts based on input.
- Load balancing ensures equal utilization of experts.

4.34 Recurrent Neural Networks (RNNs)

- Processes sequences one step at a time while maintaining a hidden state:

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

- **Challenges:**
  - Vanishing and exploding gradients.
  - Requires backpropagation through time (BPTT).
- **Solutions:**
  - Layer normalization.
  - Gated architectures like LSTMs and GRUs.

4.35 LSTM

- Handle the vanishing gradient problem through their specialized architecture, which includes mechanisms to preserve information over long time periods and manage the flow of gradients effectively.
- Gates: LSTMs use three primary gates (input gate, forget gate, and output gate) to control the flow of information. These gates are implemented using sigmoid activations, which allow the network to selectively add or remove information from the cell state:
  - Forget Gate: Decides what information to discard from the cell state.

  - Input Gate: Decides what new information to add to the cell state.
  - Output Gate: Controls the output based on the updated cell state.
- Cell state uses a direct connection with only elementwise additions and multiplications; enables gradients to flow relatively unchanged during backprop, effectively preventing the exponential decay of gradients.

4.36 Stable Diffusion

- Step-by-step diffusion process maps data to noise and back.
- UNet predicts noise added to latent representations at each step.
- Pretrained VAEs and text encoders extract semantic features from input.

4.37 Network Pruning

- Reduces model complexity by removing redundant weights.
- **Lottery Ticket Hypothesis:** A smaller, trainable subnetwork exists within the original network.
  - Iteratively prune smallest-magnitude weights.
  - Reset remaining weights to original values.
- Benefits:
  - Speeds up training and inference.
  - Reduces memory and computational requirements.

4.38 UNET: CNN for Image Segmentation

- Successive convolution and pooling layers reduce spatial dimensions and increase feature depth.
- Skip connections combine encoder and decoder features for better reconstruction.
- Common loss functions:
  - Cross-Entropy Loss:
$$\mathcal{L}_{\text{CE}} = -\sum_i y_i \log \hat{y}_i.$$
  - Dice Loss:
$$\mathcal{L}_{\text{Dice}} = 1 - \frac{2|P \cap G|}{|P| + |G|},$$
    where P is the predicted set and G is the ground truth.

4.39 Graph Neural Networks (GNNs)

- Operate on graph-structured data.
- **Message Passing:** Aggregates information from node neighbors:

$$h_v^{(k+1)} = \text{update}\left(h_v^{(k)}, \text{aggregate}(\{h_u^{(k)} : u \in \mathcal{N}(v)\})\right).$$

- Incorporates adjacency and degree matrices:

$$A, \quad D = \text{diag}(\deg(v)).$$

- Applications:
  - Node classification.
  - Link prediction.
  - Graph classification.

4.40 Retrieval-Augmented Generation (RAG) Systems

- Combines retrieval and generation for improved responses.
- Uses a similarity function (e.g., cosine similarity) to retrieve relevant context from a database.
- Small speculative models guess multiple tokens at once, confirmed by larger models.
- Objective function for RAG includes retrieval and generation terms:

$$\mathcal{L} = \log P(r|q) + \log P(c|D),$$

  where r is the retrieved context, q is the query, and c is the response.

4.41 Hebbian Learning

- Used to store and recall patterns given partial patterns (e.g. noisy signals) for patterns p
- Idea: Neurons which fire together, wire together
- $$w_{ij} = \frac{1}{N} \sum_p s_i^p s_j^p$$
- Neurons update with activation rule

$$s_i = \text{sign} \sum_j w_{ij} s_j - \theta_i$$

4.42 Hopfield Networks

- Stores and recalls patterns using binary nodes.
- Energy function:

$$E = -\frac{1}{2}\sum_{i,j} w_{ij} s_i s_j.$$

- Learns patterns by minimizing the energy function.
- Recalls patterns with partial input.

4.43 Restricted Boltzmann Machines (RBMs)

- Bipartite graph structure with visible and hidden units.
- Energy function:

$$E(v,h) = -\sum_i a_i v_i - \sum_j b_j h_j - \sum_{i,j} v_i w_{ij} h_j.$$

- Trained using Contrastive Divergence to reconstruct data.

4.44 Belief Propagation

- Represent dependencies between variables using a graph structure.
- Nodes represent random variables, edges encode conditional dependencies.
- Utilize message-passing algorithms to compute beliefs about variables.
  - Messages are exchanged between nodes iteratively:

$$m_{ij}(x_j) = \sum_{x_i} \psi_{ij}(x_i, x_j)\psi_i(x_i) \prod_{k \in \text{ne}(i)\setminus j} m_{ki}(x_i).$$

  - Messages are updated iteratively, passing information between connected nodes.
  - After convergence, compute approximate marginal probabilities:

$$b_i(x_i) \propto \psi_i(x_i) \prod_{j \in \text{ne}(i)} m_{ji}(x_i).$$

- Example: Belief Propagation in Bayesian Networks and Markov Random Fields.

4.45 Deep Belief Networks (DBNs)

- Stacks of RBMs for unsupervised pretraining.
- Greedy layer-wise training.
- Fine-tuned with supervised learning for downstream tasks.

4.46 Interpretable Model-Agnostic Explanations

- Uses surrogate models to approximate complex models locally for interpretability.
- Example: Linear approximation in the neighborhood of a prediction:

$$g(x') = \sum_i w_i x'_i,$$

  where $g(x')$ is the surrogate model, w_i are the coefficients, and x'_i are the input features.

- **SHAP (SHapley Additive exPlanations):**
  - Computes the contribution of each feature by leveraging Shapley values from cooperative game theory.
  - Shapley value formula:

$$\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!} \left[v(S \cup \{i\}) - v(S) \right],$$

    where $v(S)$ is the value of subset S.
  - Explains individual predictions and provides global insights.
- **LIME (Local Interpretable Model-agnostic Explanations):**
  - Approximates the behavior of a complex model locally with a simpler surrogate model (e.g., linear regression).
  - Steps:
    1. Sample perturbed instances around the input of interest.
    2. Weigh these instances by proximity to the input.
    3. Train a surrogate model to fit these weighted instances.
  - Outputs feature importance scores to explain predictions.

4.47 Internal Covariate Shift

Change in distribution of activations within neural network parameter which slows down learning, makes gradients unstable.

4.48 Saliency Maps

- How much is output changing with respect to input
- Process: 1. Forward Pass, 2. Backward pass, 3. Visualize absolute value of gradients

4.49 Parameter Tying

- Instead of learning separate parameters for different parts of the model, parameter tying forces parts to share the same parameters
- Reduces complexity, increases symmetry and consistency
- Examples: RNN (shared hidden states for steps); CNN (kernel weights)

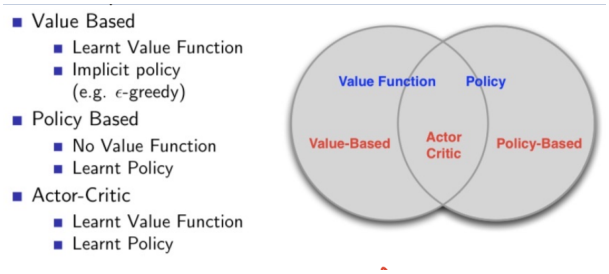
5 Reinforcement Learning

- Agent interacts with the environment at discrete timesteps t.
- Observes state s_t, takes action a_t, receives reward r_t, and transitions to state s_{t+1}.
- Discounted reward:
$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k},$$
where γ is the discount factor.
- Policy gradient methods optimize the objective function $J(\theta)$:
$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q(s,a)].$$
- Monte Carlo policy gradient estimates $Q(s,a)$ using returns.

5.1 Model Based RL

Needs model of the environment.

5.2 Value vs. Policy Based RL



5.3 Markov Decision Processes (MDPs)

- Defined by states S, actions A, transition probabilities $P(s'|s,a)$, and rewards $R(s,a)$.
- Markov property: Future state depends only on the current state and action.
- Bellman equation for state-value function:
$$V(s) = \mathbb{E}_{a \sim \pi}[R(s,a) + \gamma V(s')].$$
- Deep Q-Learning approximates $Q(s,a)$ using a neural network and minimizes:
$$\mathcal{L}(\theta) = \mathbb{E}[(Q(s,a;\theta) - y)^2],$$
where $y = r + \gamma \max_{a'} Q(s', a'; \theta')$.

5.4 Markov Chain

Next state depends on current state. Uses **states, transition probabilities and is memoryless**.

5.5 Hidden Markov Models (HMMs)

- Represents processes with hidden states Z and observations X.
- Components:
  - Initial state probabilities $P(z_1)$.
  - Transition probabilities $P(z_{t+1}|z_t)$.
  - Emission probabilities $P(x_t|z_t)$.
- Forward algorithm computes observation likelihood:

$$\alpha_t(z) = \sum_{z'} \alpha_{t-1}(z')P(z|z')P(x_t|z).$$

5.6 Markov Decision Processes (MDPs)

- Defined by states S, actions A, transition probabilities $P(s'|s,a)$, and rewards $R(s,a)$.
- Markov property: Future state depends only on the current state and action.
- Bellman equation for state-value function:

$$V(s) = \mathbb{E}_{a\sim\pi}[R(s,a) + \gamma V(s')].$$

- Deep Q-Learning approximates $Q(s,a)$ using a neural network and minimizes:

$$\mathcal{L}(\theta) = \mathbb{E}[(Q(s,a;\theta) - y)^2],$$

  where $y = r + \gamma \max_{a'} Q(s',a';\theta')$.

5.7 Multi-Armed Bandits:

Models decision-making with k independent actions (arms):

- **Epsilon-Greedy:** Chooses random actions with probability ϵ, otherwise exploits the best-known action.
- **Upper Confidence Bound (UCB):** Balances exploration and exploitation based on action uncertainty:

$$a_t = \arg\max_a \left(\hat{Q}(a) + c\sqrt{\frac{\log t}{N(a)}} \right),$$

  where $\hat{Q}(a)$ is the estimated reward, $N(a)$ is the count of action a, and c controls exploration.
- **Thompson Sampling:** Samples actions proportionally to the probability of being optimal.

5.8 SARSA:

On-policy algorithm that updates Q-values based on the current policy:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma Q(s',a') - Q(s,a) \right].$$

5.9 Policy Gradient

- Use sign of reward on gradient. Encourages actions leading to win, discourage others.
- Generalizes likelihood ratio approach to multi step MDP
- Replaces instanaeous reward r with long-term value Q

5.10 TD Learning

1. **Combines Monte Carlo and Dynamic Programming (DP):**
   - TD learning estimates value functions by combining ideas from Monte Carlo methods (sampling) and DP (bootstrapping).
   - It updates estimates based on sampled experience and partial future estimates.
2. **Incremental Updates:**
   - Updates are made after every step rather than waiting until the end of an episode, enabling real-time learning.
   - The update rule is given by:

     $$V(s) \leftarrow V(s) + \alpha \big[R_{t+1} + \gamma V(s') - V(s) \big]$$

     where:
     - α: Learning rate.
     - γ: Discount factor.
     - R_{t+1}: Reward received after the transition.
     - $V(s)$: Current estimate of the value of state s.
     - $V(s')$: Current estimate of the value of the next state s'.
3. **Bootstrapping:**
   - TD learning uses the current estimate of the value of the next state ($V(s')$) to improve the current estimate ($V(s)$).
   - This is more efficient than waiting for the final outcome of an episode.
4. **Key Algorithms:**
   - TD learning includes important algorithms such as:
     - **TD(0)**: One-step TD learning.
     - **SARSA**: On-policy TD control.
     - **Q-learning**: Off-policy TD control.
   - These algorithms are widely used in reinforcement learning tasks.

5.11 Reinforce

1. **Policy Gradient Method:**

   - REINFORCE is a Monte Carlo policy gradient algorithm used to optimize a stochastic policy by directly adjusting its parameters.
   - It maximizes the expected cumulative reward $J(\theta)$ by following the gradient of the objective:
   $$\nabla_\theta J(\theta) = \mathbb{E}\big[\nabla_\theta \log \pi_\theta(a|s) G_t\big],$$
   where $\pi_\theta(a|s)$ is the policy, and G_t is the cumulative reward from time t.

2. **Monte Carlo Sampling:**

   - The algorithm uses sampled trajectories to estimate the gradient without relying on a value function.
   - Rewards are collected from full episodes, making it suitable for episodic tasks.

3. **Updates Based on Rewards:**

   - The policy is updated to increase the probability of actions that lead to higher rewards.
   - The update rule for the policy parameters is:
   $$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a|s) G_t,$$
   where α is the learning rate.

4. **Key Characteristics:**

   - **Variance**: The method can have high variance due to its reliance on full-episode returns.
   - **Baseline**: Adding a baseline (e.g., value function) can reduce variance without introducing bias.
   - It is widely used in reinforcement learning tasks with discrete action spaces.

5.12 Q-Learning:

Off-policy algorithm that learns the optimal policy by maximizing future rewards. The Q-value is updated using the Bellman equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big[R + \gamma \max_a Q(s',a) - Q(s,a)\big],$$

where:

- α: Learning rate.
- γ: Discount factor.
- R: Immediate reward.
- s': Next state.

5.13 Deep Q-Learning

1. **Combines Q-Learning with Deep Neural Networks:**
   - Deep Q-Learning extends traditional Q-learning by using a deep neural network (DNN) to approximate the Q-value function $Q(s, a)$, which predicts the expected cumulative reward for taking action a in state s.
   - This approach is effective in handling high-dimensional state spaces, such as images.
2. **Q-Value Update Rule:**
   - In DQL, the neural network approximates $Q(s, a)$ and is updated based on the temporal difference error.
3. **Experience Replay:**
   - A replay buffer stores past experiences (s, a, R, s'), which are sampled randomly to train the neural network.
   - This reduces correlation between training samples and improves learning stability.
4. **Target Network for Stability:**
   - A separate target network is used to compute the target Q-values, which are periodically updated to match the primary network.
   - This mitigates instability and divergence during training by preventing the same network from bootstrapping off unstable predictions.

5.14 Actor-Critic Methods:

- Actor updates the policy.
- Critic evaluates the action using a value function.

1. **Combines Policy and Value-Based Methods:**
   - Actor-Critic algorithms combine the strengths of policy-based methods (the "Actor") and value-based methods (the "Critic").
   - The **Actor** updates the policy $\pi_\theta(a|s)$ directly to maximize expected rewards.
   - The **Critic** estimates the value function $V_w(s)$ to guide the Actor's updates.
2. **Two Neural Networks:**
   - The **Actor Network** outputs the policy (a probability distribution over actions).
   - The **Critic Network** predicts the value function (e.g., $V(s)$ or $Q(s, a)$).
3. **Policy Gradient Update:**

- The Actor is updated using the policy gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}\big[\nabla_\theta \log \pi_\theta(a|s) A(s,a)\big],$$

where $A(s,a)$ is the advantage function, which can be defined as:

$$A(s,a) = Q(s,a) - V(s),$$

or an approximation of the temporal difference error.

5.15 Advantage Actor Critic

- A2C is a synchronous, improved version of Actor-Critic that uses multiple agents (workers) to collect experience in parallel.
- Experiences are aggregated to compute more stable gradient updates, improving training efficiency and convergence.

5.16 Monte Carlo Methods:

- Estimate value functions using episodic returns.
- Updates policies based on sampled trajectories:

$$\pi(a|s) \leftarrow \pi(a|s) + \alpha \nabla \log \pi(a|s) G_t,$$

where G_t is the return from timestep t.

5.17 Monte Carlo Tree Search (MCTS):

Explores decision trees by simulating play and propagating values up the tree.

5.18 AlphaGo

- Uses two neural networks:
  - Policy network selects actions.
  - Value network predicts the winner of a given state.
- Trains by self-play and reinforcement learning.
- Monte Carlo simulations explore possible moves and refine strategies.

5.19 Proximal Policy Optimization (PPO)

- Enhances policy gradient methods by ensuring stable learning.
- Clipping restricts the extent of policy changes:

$$\mathcal{L}(\theta) = \mathbb{E}[\min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)],$$

where $r_t(\theta)$ is the probability ratio and A_t is the advantage function.

- Balances exploration and exploitation effectively.

5.20 Trust Region Policy Optimization (TRPO)

- Ensures stable updates to the policy by enforcing a constraint on the change in the policy distribution.
- Uses the KL-divergence as a constraint:

$$\mathbb{E}[\mathrm{KL}[\pi_{\mathrm{old}}(a|s)\|\pi_{\mathrm{new}}(a|s)]] \leq \delta,$$

  where δ is a small positive constant.
- Optimizes the surrogate objective:

$$\mathcal{L}(\theta) = \mathbb{E}\left[\frac{\pi_{\mathrm{new}}(a|s)}{\pi_{\mathrm{old}}(a|s)} A(s,a)\right].$$

5.21 RLHF

- Intergrate human feedback into RL
- Reward Modelling: $r_\theta(x, y)$ given prompts x and answers y_w and y_l (preferred and less preferred)
- It minimizes $\mathcal{L} = -\log(r_\theta(x, y_w) - r_\theta(x, y_l))$
- Policy Optimization: Model generates response given prompt. Objective is expected reward from reward model while having similarity to initial fine tuned model.