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

1.1 Type 1 and Type 2 Error

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

1.2 Probability Distributions

- **Beta:**
  - The Beta distribution, Beta(α, β), is a continuous probability distribution on $x \in (0, 1)$ parameterized by shape parameters $\alpha, \beta > 0$. Its probability density function (PDF) is:

    $$f(x; \alpha, \beta) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha, \beta)}$$

    where $B(\alpha, \beta) = \int_0^1 t^{\alpha-1}(1-t)^{\beta-1}dt$ is the Beta function.
  - It is a conjugate prior for the Bernoulli, Binomial, and related distributions in Bayesian statistics, with the expected value:

    $$\mathbb{E}[X] = \frac{\alpha}{\alpha + \beta}$$

- **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.

- **Beta Distribution:**
  - **Probability Distribution over a Unit Interval** – The Beta distribution is a continuous probability distribution defined on the interval $[0, 1]$, making it useful for modeling probabilities and proportions.
  - **Parameterized by Two Shape Parameters** – It is controlled by two positive parameters α and β, which determine the shape of the distribution:

$$P(x|\alpha,\beta) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)},$$

where $B(\alpha, \beta)$ is the Beta function, ensuring normalization.
  - **Used in Bayesian Inference** – The Beta distribution is commonly used as a conjugate prior for the Bernoulli and Binomial distributions, allowing easy posterior updates in Bayesian statistics.
- **Dirichlet Distribution:**
  - **Distribution over Probability Vectors** – The Dirichlet distribution is a multivariate generalization of the Beta distribution and defines a probability distribution over a simplex, meaning it generates valid probability vectors where elements sum to 1.
  - **Parameterized by Concentration Parameters** – It is defined by a vector of positive concentration parameters $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_K)$, which control the distribution of probabilities among K categories.
  - **Common in Bayesian Statistics and Topic Modeling** – The Dirichlet distribution is widely used as a prior in Bayesian models (e.g., Latent Dirichlet Allocation for topic modeling), where it acts as a prior for categorical distributions like the multinomial.
- **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.

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

1.3 Common Conjugate Priors

1.4 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:

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

- The weights in M step come from the probabilities computed in the E-Step.

1.5 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.
- **Define the Likelihood Function:** The likelihood function represents the probability of the observed data given the model parameters. For independent and identically distributed (i.i.d.) data, the likelihood is the product of the individual probabilities:

$$L(\theta) = \prod_{i=1}^{n} p(x_i \mid \theta)$$

  where θ are the parameters of the model.

- **Log-Likelihood Simplification:** To simplify calculations, the logarithm of the likelihood function is often used. This converts the product into a sum:

$$\ell(\theta) = \log L(\theta) = \sum_{i=1}^{n} \log p(x_i \mid \theta)$$

- **Optimization:** The goal is to find the parameter values $\hat{\theta}$ that maximize the (log-)likelihood function. This involves solving:

$$\hat{\theta} = \arg\max_{\theta} \ell(\theta)$$

- If the likelihood function has a closed-form solution, we can solve for θ directly (e.g., in Gaussian distributions, MLE has an analytical solution).

  However, in most complex models (e.g., neural networks, deep learning, logistic regression, latent variable models like GMMs), there is no closed-form solution.

- **Interpretation:** The resulting $\hat{\theta}$ are the parameters that make the observed data most likely under the model, assuming the data comes from the distribution specified by $p(x \mid \theta)$.

1.6 Bayesian Inference and Maximum A Posteriori (MAP)

- Balances likelihood and prior knowledge.
- **Bayesian Extension of MLE** – Unlike Maximum Likelihood Estimation (MLE), MAP incorporates a prior belief about parameters using Bayes' theorem.
- **Objective Function** – MAP maximizes the posterior probability:

$$\theta_{\text{MAP}} = \arg\max_{\theta} P(\theta|D) = \arg\max_{\theta} P(D|\theta)P(\theta),$$

  where $P(D|\theta)$ is the likelihood and $P(\theta)$ is the prior.

- **Regularization Connection** – In many cases, MAP estimation corresponds to adding a regularization term in optimization (e.g., Gaussian priors in MAP lead to L2 regularization in linear regression).
- **Optimization with Gradient Descent** – When no closed-form solution exists, MAP estimation is solved using iterative methods like gradient ascent or Expectation-Maximization (EM).

1.7 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
- Non-parametric method to estimate the probability density function of a random variable.
- **Kernel Function Smoothing** – Uses a kernel function (e.g., Gaussian, Epanechnikov) to smooth data points and create a continuous density estimate.
- Common kernels include Gaussian, Epanechnikov, and Uniform.
- Bandwidth h controls the smoothness of the estimate.

1.8 KL Divergence

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

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

1.9 ELBO

- 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 by maximizing a tractable lower bound on the log-marginal likelihood.

- The ELBO is derived from Jensen's inequality and is given by:

$$\log P(X) \geq \mathbb{E}_{q(Z|X)}\left[\log P(X, Z) - \log q(Z|X)\right]$$

  Maximizing ELBO improves the approximation $q(Z|X)$ of the true posterior $P(Z|X)$, making it a central tool in variational inference and Bayesian deep learning.

- It is called the Evidence Lower Bound (ELBO) because:
  - "Evidence" refers to the observed data likelihood $logP(X)$ (also called the marginal likelihood).
  - "Lower Bound" because ELBO provides a lower bound on $logP(X)$, meaning that optimizing ELBO indirectly maximizes the log-likelihood.

1.10 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):

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

1.11 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.

1.12 Probability Math Rules

1.12.1 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.

1.12.2 Set Rules

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

1.12.3 Bayes Rule

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

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

1.12.4 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.

1.12.5 Complement Rule

- Relates the probability of an event to its complement:

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

1.12.6 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:** $P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$.

1.13 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.

1.14 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:

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

1.15 Central Limit Theorem

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

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

1.16 Random Variables

Random variables quantify uncertainty in probabilistic models.

1.17 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.

1.18 IID

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

1.19 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.

1.20 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.

1.21 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:

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

1.22 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., $E \geq 5$).

1.24.4 Wilcoxon Test

- **Non-Parametric Statistical Test** – The Wilcoxon test is a rank-based, non-parametric test used to compare paired or independent samples without assuming a normal distribution.
- **Two Main Variants** – The Wilcoxon signed-rank test compares paired (dependent) samples, while the Wilcoxon rank-sum test (also called the Mann-Whitney U test) compares two independent samples.
- **Wilcoxon Signed-Rank Test Formula** – For paired samples $X_1, X_2, \ldots, X_n$, compute differences $d_i = X_i - Y_i$, rank their absolute values R_i, and sum the ranks for positive and negative differences:

$$W = \sum_{i:d_i>0} R_i.$$

  The test statistic W is compared against a critical value or approximated using a normal distribution for large samples.
- **Wilcoxon Rank-Sum Test (Mann-Whitney U Test) Formula** – Given two independent samples of sizes n_1 and n_2, rank all observations together and compute:

$$U = n_1 n_2 + \frac{n_1(n_1 + 1)}{2} - R_1,$$

  where R_1 is the sum of ranks for the first sample. The test assesses whether the two groups come from the same distribution.

- **Ranks Instead of Raw Values** – Instead of comparing raw data values, the test ranks observations and assesses differences in distributions, making it robust to outliers and non-normal data.
- **Common in Small-Sample Analysis** – Frequently used in medical research, psychology, and other fields where sample sizes are small, and parametric assumptions (e.g., normality) may not hold.

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. A matrix is invertible if and only if its determinant is nonzero. The determinant of a transformation matrix represents scaling in geometry. For example, in 2D or 3D, $|det(A)|$ gives the factor by which areas or volumes are scaled under the transformation. The determinant of a matrix is the product of its eigenvalues.

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.
- Second-order expansion (using the Hessian) can help refine step directions for faster convergence

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

- **Splitting Criterion:** At each node, the algorithm selects the best feature and threshold to split the data based on metrics like Gini impurity, entropy (for classification), or variance reduction (for regression). Split based on the best feature using Gini coefficient

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

or entropy

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

.

- **Recursive Partitioning:** The dataset is recursively split into subsets, forming a tree structure where each split aims to increase the purity of child nodes.
- **Stopping Criteria:** The tree stops growing when a predefined condition is met, such as reaching a maximum depth, a minimum number of samples per node, or no further gain in information.
- **Pruning:** To avoid overfitting, post-processing techniques like pruning remove less significant branches, improving generalization on unseen data.

3.5 Support Vector Machines (SVM)

- Find the optimal hyperplane that separates data points from different classes
- Aims to maximize the margin (distance between the hyperplane and the nearest data points of any class). These nearest data points are called support vectors.
- SVM solves an optimization problem to find the hyperplane that maximizes the margin.

  The optimization problem for linearly separable data is formulated as:

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

  Subject to:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, \quad \forall i$$

  where:

  - $\mathbf{x}_i$: Feature vector for the i-th sample,
  - y_i: Label ($+1$ or -1),
  - b: Bias term.

- 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

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

- If the data cannot be perfectly separated, SVM introduces a *slack variable* to allow for some misclassification.

  The optimization problem becomes:

$$\min_{\mathbf{w},b,\xi} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \xi_i$$

  Subject to:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

  where:

- ξ_i: Slack variable for the i-th data point,
- C: Regularization parameter controlling the trade-off between margin maximization and the penalty for misclassification.

- After training, the decision function is given by:

$$f(\mathbf{x}) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b)$$

For multi-class problems, SVM can be extended using techniques such as *one-vs-one* or *one-vs-all*.

3.6 Matrix Factorizations

- **SVD:** Use for low-rank approximation, PCA, ill-conditioned problems. Works for any matrix but is expensive. Best for stability and compression.

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

where Σ contains singular values, U left singular vector and V right singular vectors. Singular vectors represent the principal directions in which a matrix stretches or compresses data when applied as a transformation

- **LU Decomposition:** Factorize A into LU for square matrices. Use for solving linear systems efficiently if the matrix is square and non-singular. Fast but less stable.

- **QR Decomposition:** Factorize A into QR, where Q is orthogonal and R is upper triangular. Use for least squares problems and orthogonalization. More stable than LU but cheaper than SVD. Best for tall matrices (more rows than columns).

3.7 Principal Component Analysis (PCA)

- Reduce dimensionality while retaining variance.
- Mean center data if necessary
- 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:** Trains multiple models on subsets of data to reduce variance. Combines their predictions
- **Boosting:**
  - Sequential Training: Boosting builds models one at a time, where each subsequent model is trained to correct the errors of the previous ones.
  - Weighted Data Points: Each data point is assigned a weight to indicate its importance. Initially, all data points have equal weight. As the iterations proceed, the weights of misclassified data points are increased, so subsequent models focus more on them.
  - Model Combination: The outputs of all weak learners are combined to make the final prediction. The combination is often a weighted sum of the individual learners' predictions.
- **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

- Precision $\frac{TP}{TP+FP}$ also TPR
- Recall: $\frac{TP}{TP+FN}$
- FPR $\frac{FP}{TN+FP}$
- Prec-Recall Curve: May be better for imbalanced data, precision on y, recall on x axis
- **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. x: FPR, y: TPR

- **Mean Average Precision (mAP):** Used for evaluating ranking and object detection tasks. P_n precision at threshold n, R_n is recall at threshold n

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i$$

$$AP = \sum_{n} (R_n - R_{n-1}) P_n$$

$$AP = \int_0^1 P(R)\, dR$$

$$AP = \sum_{k} P(k) \Delta R(k)$$

- **Euclidean Distance:**

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

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

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

3.14 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:

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

- **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:

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

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

3.16 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.

3.17 Data Synthesis

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

3.18 Overfitting Prevention

- Reduce model complexity.
- Use regularization: L1, L2
- Apply dropout to randomly deactivate neurons.
- Ensembling: Combine predictions from multiple models.
- Cross-validation: Use a validation set to tune hyperparameters.
- Early stopping
- Feature selection
- More data

3.18.1 Label Smoothing

Makes hard probabilities soft

$$y_{smooth} = (1 - \alpha)y + \frac{\alpha}{\text{num}_{\text{classes}}}$$

3.19 Feature Selection

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

3.20 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.

3.21 Uncertainty Quantification

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

3.22 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:

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

3.23 Clustering

3.23.1 K-Means

- **Clustering Algorithm:** k-means is an iterative algorithm that partitions a dataset into k clusters by minimizing the within-cluster variance.
- **Objective Function:** The algorithm minimizes the total within-cluster sum of squared distances (WCSS), given by:

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

  where C_i is the set of points in the i-th cluster, μ_i is the centroid of C_i, and $\|\cdot\|$ denotes the Euclidean norm.
- **Centroid Update:** The centroid of each cluster C_i is updated as the mean of all points assigned to that cluster:

$$\mu_i = \frac{1}{|C_i|} \sum_{\mathbf{x} \in C_i} \mathbf{x}.$$

- **Iterative Process:** The algorithm alternates between:
  1. **Assignment Step:** Assign each data point $\mathbf{x}$ to the nearest cluster based on the centroid:
  $$C_i = \left\{\mathbf{x} : \|\mathbf{x} - \mu_i\|^2 \leq \|\mathbf{x} - \mu_j\|^2, \forall j \neq i\right\}.$$
  2. **Update Step:** Recompute centroids using the updated cluster assignments.

3.23.2 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

3.23.3 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.

3.23.4 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.
- Typically uses k-means on eigenvectors to assign clusters
- Suitable for non-linearly separable data.

3.23.5 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 (points with at least MinPts neighbors within ϵ distance), border points (Points within ϵ of a core point but with fewer than MinPts neighbors), and noise (all other).
- Start with unvisited points, check neighbors. If core, form new cluster, add neighbors to cluster, continue until all points are allocated.
- Doesn't require number of clusters as inputs.
- Advantages:
  - Robust to outliers.
  - Handles clusters of arbitrary shapes.

3.23.6 Self-Organizing Maps (SOM)

- Unsupervised neural network for dimensionality reduction and clustering that preserves topological properties of data.
- Neuron weights arranged in a 2D or 3D grid indexed by position.
- Process:
  1. Initialize weights randomly.
  2. Sample input data.
  3. Find best matching unit (BMU) (closest node):
  $$\text{BMU} = \arg\min_j \|x - w_j\|.$$
  4. Update weights of BMU and its neighbors:
  $$w_j(t+1) = w_j(t) + \eta(t) h_{j,i}(t)(x - w_j(t)).$$
     - $\mathbf{w}_j(t+1)$: The updated weight vector of the j-th neuron at time step $t+1$.
     - $\mathbf{w}_j(t)$: The current weight vector of the j-th neuron at time step t.
     - $\eta(t)$: The learning rate at time t, which controls the magnitude of the update and typically decreases over time.
     - $h_{j,i}(t)$: The neighborhood function, which determines the influence of the BMU (i) on the j-th neuron. It depends on the distance between neurons i and j on the grid and decreases over time. A common choice is the Gaussian function:
     $$h_{j,i}(t) = \exp\left(-\frac{\text{dist}(j,i)^2}{2\sigma(t)^2}\right),$$
     where:

    - $*$ $\text{dist}(j, i)$: The distance between neurons j and i on the grid.
    - $*$ $\sigma(t)$: The neighborhood radius, which also decreases over time.
  - $\mathbf{x}$: The input vector being processed at the current iteration.
  - $\mathbf{x} - \mathbf{w}_j(t)$: The difference vector, representing the error between the input vector and the current weight vector of neuron j.
5. Repeat until convergence.

- Dimensionality Reduction: The weights are the lower dimensional representation
- Clustering: Nodes (or groups of nodes) that respond strongly to similar data points form clusters.

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.
- **Batch Gradient Descent:** Uses multiple input instances to compute the gradient.
- **Momentum:** Accumulates past gradients to smooth updates and escape local minima:
$$v_t = \beta v_{t-1} - \eta \nabla L(w_t), \quad w_{t+1} = w_t + v_t.$$
- **Nesterov Accelerated Gradient:** Adjusts gradients using a lookahead mechanism:
$$v_{t+1} = \beta v_t - \eta \nabla L(w_t - \beta v_t).$$
$$w_{t+1} = w_t + v_{t+1}$$
where γ is the momentum term and η is the learning rate. By anticipating the future position, NAG reduces oscillations and improves convergence speed, particularly in convex optimization and deep learning training scenarios.
- **Gradient Clipping:** Limits the gradient magnitude to avoid exploding gradients:
$$\nabla L(w) \leftarrow \text{clip}(\nabla L(w), -c, c).$$

4.3 Optimizers

- **SGD:**
  - **Optimization Algorithm:** Stochastic Gradient Descent (SGD) is an iterative optimization algorithm used to minimize a given loss function by updating parameters in the opposite direction of the gradient.
  - **Objective Function:** Given a loss function $L(\theta)$, where θ represents the parameters, the goal is to find:

$$\theta^* = \arg\min_{\theta} L(\theta).$$

  - **Gradient Update Rule:** The parameters θ are updated using the gradient of the loss function with respect to a single (or a small batch of) data point(s):

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L(\theta; \mathbf{x}_i, y_i),$$

    where:
    - η is the learning rate,
    - $\nabla_\theta L(\theta; \mathbf{x}_i, y_i)$ is the gradient of the loss with respect to the i-th data point $(\mathbf{x}_i, y_i)$.
  - **Stochastic Approximation:** Unlike full-batch gradient descent, SGD uses a single data point (or a mini-batch) at each iteration, which provides a noisy estimate of the gradient:

$$\nabla_\theta L(\theta) \approx \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \nabla_\theta L(\theta; \mathbf{x}_i, y_i),$$

    where $\mathcal{B}$ is the mini-batch of size $|\mathcal{B}|$.
  - **Convergence:** The learning rate η often follows a schedule (e.g., decaying over time) to ensure convergence to a local or global minimum.

- **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:

$$v_t = \beta v_{t-1} + (1 - \beta) g_t^2, \quad w \leftarrow w - \eta \frac{g_t}{\sqrt{v_t} + \epsilon}.$$

4.4 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.
- We want a stable gradient flow across the network

4.5 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.

4.6 Gradient Checkpointing

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

4.7 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.

4.8 Weight Initialization

- Zero init leads to failure and random init can lead to vanishing/exploding gradients
- The variance of the outputs of each layer should be equal to the variance of its inputs. Additionally, the gradients should maintain equal variance before and after flowing through a layer during backpropagation. This helps prevent the gradients from vanishing or exploding.
- **Xavier Initialization:** Scales weights by:
$$W \sim \mathcal{N}\left(0, \sqrt{\frac{2}{n_{\text{in}} + n_{\text{out}}}}\right).$$
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/Kaiming Initialization:** Scales weights for ReLU activation:
$$W \sim \mathcal{N}\left(0, \frac{2}{n_{\text{in}}}\right).$$

4.9 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.

  In a ranking scenario, you deal with pairs of items (i, j), where the goal is to rank i higher than j. The hinge loss is extended to:

$$\text{Ranking Hinge Loss} = \sum_{(i,j)\in\mathcal{P}}\max(0, 1 - (s_i - s_j))$$

  Where:

  - $\mathcal{P}$ is the set of all pairs such that i should be ranked higher than j,
  - s_i and s_j are the predicted scores for items i and j, respectively.

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

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

  The parameter δ controls the threshold for treating differences as outliers. Combines the benefits of Mean Squared Error (MSE) and Mean Absolute Error (MAE) by being quadratic for small errors (like MSE) and linear for large errors (like MAE)

- **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. Range [-1,1].

- **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

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

4.10 Activation Functions

- **Sigmoid:** $f(z) = \frac{1}{1+e^{-z}}$. For binary tasks, vanishing gradient problem.
- **ReLU:** $f(z) = \max(0, z)$. Helps with vanishing and exploding gradient, dying ReLU problem
- **Leaky ReLU:** $f(z) = \max(\alpha z, z), \alpha > 0$. Allows small negative values, reducing the dying ReLU issue.

- **GELU:** $f(z) = z\Phi(z)$, where $\Phi(z)$ is the Gaussian CDF. Smooth activation improves training dynamics, computationally more expensive than ReLU. Used in NLP.
- **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. Situations where negative activations help learning, helps with vanishing gradients better than ReLU, computationally more complex.
- **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 $\text{Swish}(x) = x \cdot \text{Sigmoid}(x)$. Vision tasks requiring better gradient flow, self-gating mechanism enhances learning.

4.11 Layer Normalization

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

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

4.12 Batch Normalization

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

4.13 Dropout

Randomly deactivates neurons during training.

4.14 Learning Rates

- Fixed learning rates: Reduces learning rate by a factor after a fixed number of epochs, may overshoot or converge too slowly.
- Step Decay:
$$\eta_t = \eta_0 \cdot \gamma^{\lfloor t/T \rfloor}$$
- Cyclical Learning Rates:
$$\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min})(1 + \cos(\frac{t\pi}{T})).$$
- **Exponential Decay:** Decay LR exponentially over time, may reset.
- **Cosine Annealing:** Gradually decrease using cosine.
- **Cyclic Scheduler:** Oscillates learning rates between minimum and maximum bounds.
- **Warm-up and Cool-down:** Gradually increase or decrease learning rates during training.

4.15 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.

4.16 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.

4.17 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 for hidden state $h_t = o_t \odot \tanh(c_t)$. These gates are implemented using sigmoid activations, which allow the network to selectively add or remove information from the cell state $c_t = f_t \odot c_{t-1} + i_t \odot g_t$ for cell gate $g_t = \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg})$:
  - Forget Gate: Decides what information to discard from the cell state. $f_t = \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf})$
  - Input Gate: Decides what new information to add to the cell state. $i_t = \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi})$
  - Output Gate: Controls the output based on the updated cell state. $o_t = \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho})$
- 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.

4.18 GRU

- **Update Gate**: Controls how much of the previous hidden state h_{t-1} should be carried forward to the next hidden state:

$$z_t = \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{t-1} + b_{hz})$$

- **Reset Gate**: Determines how much of the previous hidden state h_{t-1} is forgotten when computing the candidate hidden state:

$$r_t = \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{t-1} + b_{hr})$$

- **New Gates**:

$$n_t = \tanh(W_{in}x_t + b_{in} + r_t \odot (W_{hn}h_{t-1} + b_{hn}))$$

- **Final Hidden State**: The final hidden state is a linear interpolation between the previous hidden state h_{t-1} and the candidate hidden state $\tilde{h}_t$, controlled by the update gate z_t:

$$h_t = (1 - z_t) \odot n_t + z_t \odot h_{t-1}$$

4.19 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:

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

- Q is each word; K is each word against each word in vocab, how it is "indexed"; V: actual information you retrieve (content)
- **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.

4.19.1 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.

4.19.2 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:

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

4.19.3 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.

4.20 Speculative Decoding

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

4.21 LLM Evaluation Metrics

- BLEU
  - Computes precision-based n-gram similarity between candidate and reference text.

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

    where p_n is the precision for n-grams, w_n is the weight, and BP is the brevity penalty.
  - The brevity penalty (BP) is given by:

$$\text{BP} = \begin{cases} 1, & \text{if } c > r \\ e^{(1-\frac{r}{c})}, & \text{if } c \leq r \end{cases}$$

    where c is the length of the candidate text and r is the length of the reference text.
- ROUGE
  - **ROUGE-N:** Measures the overlap of n-grams between the candidate text and reference text.

$$\text{ROUGE-N} = \frac{\sum_{\text{gram}_n \in \text{Ref}} \text{Count}_{\text{match}}(\text{gram}_n)}{\sum_{\text{gram}_n \in \text{Ref}} \text{Count}(\text{gram}_n)}$$

  - **ROUGE-L:** Considers the longest common subsequence (LCS) to capture sentence-level structure.

$$\text{ROUGE-L} = \frac{\text{LCS}(\text{Candidate}, \text{Reference})}{\text{Length}(\text{Reference})}$$

4.22 Model Quantization

- **Definition:** Quantization is the process of mapping continuous values to a finite set of discrete values, often used in signal processing and neural network optimization.
- **Uniform Quantization:** Divides the range of values into equal-sized intervals, with each value mapped to the nearest representative level.

$$Q(x) = \Delta \cdot \left\lfloor \frac{x}{\Delta} + 0.5 \right\rfloor$$

  where Δ is the quantization step size.
- **Non-Uniform Quantization:** Uses variable-sized intervals, typically based on a logarithmic or learned distribution, to allocate more precision to frequently occurring values.
- **Application in Deep Learning:** Reduces model size and inference latency by representing weights and activations with lower bit-widths (e.g., 8-bit integers instead of 32-bit floats).

4.23 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.

4.24 Contrastive Learning

- **Objective:** Contrastive learning aims to learn representations by bringing similar data points closer together in the embedding space while pushing dissimilar data points further apart.
- **Loss Function:** The general loss function for contrastive learning can be written as:

$$\mathcal{L}_{\text{contrastive}} = -\log \frac{\exp(\text{sim}(\mathbf{z}_i, \mathbf{z}_j)/\tau)}{\sum_k \exp(\text{sim}(\mathbf{z}_i, \mathbf{z}_k)/\tau)},$$

  where:
  - $\text{sim}(\mathbf{z}_i, \mathbf{z}_j)$ is a similarity function, often cosine similarity,
  - τ is the temperature scaling hyperparameter,

  - $\mathbf{z}_i$ and $\mathbf{z}_j$ are representations of positive pairs, and $\mathbf{z}_k$ includes negative samples.

- **Key Components:** Positive pairs (similar data points) are generated using augmentations or natural relationships, while negative pairs (dissimilar data points) are sampled from other examples in the batch.

- **Applications:** Contrastive learning is used for tasks like self-supervised image representation learning, NLP (e.g., sentence embeddings), and cross-modal learning.

4.24.1 InfoNCE

- **Purpose:** InfoNCE (Information Noise Contrastive Estimation) is designed to maximize the mutual information between input data and its learned representation.

- **Loss Function:** The InfoNCE loss is defined as:

$$\mathcal{L}_{\text{InfoNCE}} = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{\exp(\text{sim}(\mathbf{z}_i,\mathbf{z}_j)/\tau)}{\sum_{k=1}^{N}\exp(\text{sim}(\mathbf{z}_i,\mathbf{z}_k)/\tau)},$$

  where N is the number of samples in the batch.

- **Key Idea:** Positive pairs are constructed using augmented views of the same data point, and all other points in the batch serve as negatives.

- **Applications:** InfoNCE is used in methods like CPC (Contrastive Predictive Coding) and SimCLR to learn self-supervised representations.

4.24.2 MaskedLM

- **Purpose:** Masked Language Modeling (MaskedLM) trains models to predict masked tokens in input sequences, enabling contextual representation learning.

- **Objective:** The loss function for MaskedLM is:

$$\mathcal{L}_{\text{MLM}} = -\sum_{i\in\mathcal{M}}\log P(x_i|\mathbf{x}_{\setminus i}),$$

  where $\mathcal{M}$ is the set of masked token indices and $P(x_i|\mathbf{x}_{\setminus i})$ represents the probability of the masked token.

- **Approach:** Random tokens in the input are masked, and the model is trained to recover them using surrounding context.

- **Applications:** MaskedLM is a foundational technique for pretraining transformer-based models like BERT and RoBERTa.

4.24.3 SimCLR

- **Purpose:** SimCLR is a framework for self-supervised learning that maximizes similarity between augmented views of the same image while minimizing similarity with other images.
- **Loss Function:** SimCLR uses the InfoNCE loss:

$$\mathcal{L}_{\text{SimCLR}} = -\log \frac{\exp(\text{sim}(\mathbf{z}_i, \mathbf{z}_j)/\tau)}{\sum_k \exp(\text{sim}(\mathbf{z}_i, \mathbf{z}_k)/\tau)}.$$

- **Augmentations:** Key augmentations include cropping, color distortions, and Gaussian blur to generate positive pairs.
- **Applications:** SimCLR demonstrates that high-quality image representations can be learned without labels when paired with strong augmentations and a large batch size.

4.24.4 MOCO

- **Purpose:** Momentum Contrast (MOCO) learns representations by maintaining a dynamic memory bank and using a momentum-updated encoder to generate consistent features.
- **Key Innovation:** MOCO replaces the need for large batch sizes by storing negative examples in a queue and updating the encoder with momentum:

$$\theta_{\text{momentum}} \leftarrow m\theta_{\text{momentum}} + (1-m)\theta_{\text{encoder}},$$

  where m is the momentum coefficient.
- **Loss Function:** MOCO uses a contrastive loss similar to InfoNCE:

$$\mathcal{L}_{\text{MOCO}} = -\log \frac{\exp(\text{sim}(\mathbf{q}, \mathbf{k}_{\text{pos}})/\tau)}{\sum_{\mathbf{k}\in\mathcal{K}} \exp(\text{sim}(\mathbf{q}, \mathbf{k})/\tau)},$$

  where $\mathcal{K}$ is the memory bank of keys.
- **Applications:** MOCO is widely used for learning visual representations in tasks like object detection and image recognition.

4.25 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.

4.26 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.

4.27 Autoencoder

- Minimize reconstruction loss
$$||x - \hat{x}||^2$$
- Encoder Decoder architectures are neural networks

4.28 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.
  - 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.

- **Reparameterization Trick:** Rewrites $z \sim q(z|x)$ as:

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

4.29 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 attention and projection layers in transformer

4.30 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.
- Allows gradients to flow through discrete variables during backpropagation, making it useful for reinforcement learning and neural network architectures requiring discrete decisions
- As temperature approaches zero, the softmax output approximates a one-hot vector, while higher temperatures produce softer distributions

4.31 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:**

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

$$\mathcal{L} = \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbf{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbf{1}_{ij}^{\text{obj}} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbf{1}_{ij}^{\text{obj}} (\hat{C}_i - C_i)^2 + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbf{1}_{ij}^{\text{noobj}} (\hat{C}_i - C_i)^2$$

$$+ \sum_{i=0}^{S^2} \mathbf{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (\hat{p}_i(c) - p_i(c))^2$$

  Localization Loss: Penalizes errors in bounding box coordinates (x, y, w, h), with emphasis on the square root of width and height for better scale invariance.

  Confidence Loss: Measures how well the predicted confidence score $\hat{C}_i$ matches the actual object presence.

  Classification Loss: Ensures that the predicted class probabilities $\hat{p}_i(c)$ align with the ground truth.

  λ_{coord} emphasizes coordinate accuracy.

  λ_{noobj} downweights confidence loss for background cells.

  $\mathbf{1}_{ij}^{\text{obj}}$ is an indicator function for whether object j is in cell i.

  $\mathbf{1}_{ij}^{\text{noobj}}$ applies when no object is present.

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

4.32 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.

4.33 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:

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

4.34 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:

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

4.35 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.

4.36 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.

4.37 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 text.
- Decoder maps latent back to image space
- Inference: Iteratively denoise with model

4.38 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.
  - Process:
    * Random init of weights
    * Train for j iteractions
    * Prune p% of parameters and create mask
    * Reset remaining params to values of random init
    * Return the optimal subnetwork
- Benefits:
  - Speeds up training and inference.
  - Reduces memory and computational requirements.

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.
- 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 and context G.

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 Boltzmann Machines

- Mimics how particles behave in a thermal system at thermal equilibrium.
- Every node is connected to every other node.
- Nodes are binary.
- Learns by minimizing the energy function:

$$E(v, h) = \sum_{i,j} w_{ij} v_i v_j - \sum_{i} b_i v_i$$

  where:

  - b_i is the bias of the node.
  - v_i is the state of the node.

4.44 Restricted Boltzmann Machines (RBMs)

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

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

- Trained using Contrastive Divergence to reconstruct data.

4.45 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.

4.46 Deep Belief Networks (DBNs)

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

4.47 Interpretable Model-Agnostic Explanations

- **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).
  - 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.

- 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.

4.48 Internal Covariate Shift

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

4.49 Saliency Maps

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

4.50 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)

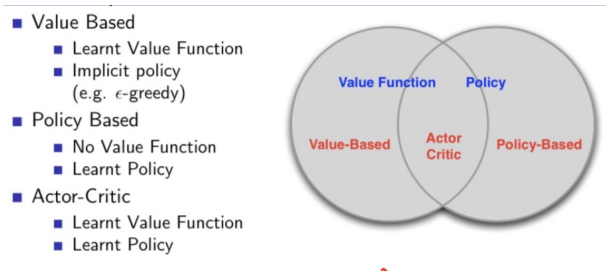
5 Reinforcement Learning

- 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.
- A Q-value represents the expected total future reward an agent will receive if it takes a specific action in a given state and follows the optimal policy afterward. It helps the agent decide which action is best by estimating long-term rewards rather than just immediate gains.

5.1 Model Based RL

Needs model of the environment.

5.2 Value vs. Policy Based RL



5.3 Bellman Equation

The Bellman equation expresses the value of a state as the immediate reward plus the expected discounted value of future states, enabling recursive computation of optimal policies in dynamic programming. It is written as:

$$V(s) = \max_a \left[R(s,a) + \gamma \sum_{s'} P(s'|s,a) V(s') \right]$$

where $V(s)$ is the value function, $R(s,a)$ is the reward, γ is the discount factor, and $P(s'|s,a)$ is the transition probability.

5.4 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

5.5 Markov Chain

Stochastic process where next state depends on current state. Uses **states, transition probabilities and is memoryless**.

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 is algorithn to solve MDPs and 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 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 $P(x_t|z_t)$, represent the probability of observing a particular output (or emission x_t) given a hidden state.
- Forward algorithm computes observation likelihood:

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

$\alpha_t(z)$ is the probability of observing the first t observations and ending up in state z

5.8 Monte Carlo Methods:

- Monte Carlo methods estimate values by running many random simulations and averaging the results, making them useful for solving problems where exact solutions are difficult or infeasible.
- Estimate value functions using episodic returns.
- Updates policies parameters based on sampled trajectories:

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

where G_t is the return from timestep t.

5.9 Monte Carlo Tree Search (MCTS):

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

5.10 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 number of times action a has been chosen so far, and c controls exploration. Upper Confidence Bound (UCB) is named because it selects actions based on an upper bound on the estimated reward, ensuring that actions with uncertain rewards are explored.

- **Thompson Sampling:** Samples actions proportionally to the probability of being optimal.

5.11 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). Bootstrapping refers to the process of updating value estimates using other current estimates rather than relying solely on actual rewards from complete episodes
   - 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.

5.12 SARSA

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

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

5.13 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. Introducing a **baseline** $b(s)$, typically the state value function $V(s)$, modifies the update:

   $$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a|s)(G_t - b(s))$$

   Since $G_t - b(s)$ centers the updates around zero, it reduces variance without changing the expected value of the gradient.
   - It is widely used in reinforcement learning tasks with discrete action spaces.

5.14 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.

5.15 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.

5.16 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.

5.17 Advantage Actor Critic

- **Combines Policy and Value-Based Methods** – Uses an *actor* (policy $\pi(a|s)$) to choose actions and a *critic* (value function $V(s)$) to estimate how good a state is, improving learning stability.
- **Uses Advantage Function** – Instead of the full return, it computes the *advantage*:

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

  to reduce variance, guiding updates based on how much better an action is than expected.
- **More Sample Efficient** – The critic reduces variance in policy updates, allowing faster and more stable learning compared to pure policy gradient methods like REINFORCE.
- **Supports Parallel Training** – A3C (Asynchronous Advantage Actor-Critic) enables multiple agents to explore simultaneously, improving training speed and robustness.

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:

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

  where δ is a small positive constant.

- Optimizes the surrogate objective:

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

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.