Part I: Machine Learning Foundations (Week 1)

1. Linear Regression & Projections

Geometric Interpretation

The goal of linear regression is to find a coefficient vector β such that $X\beta$ approximates the target vector y. Geometrically, the column space of X forms a subspace (a plane). Since y usually does not lie perfectly on this plane, we seek the orthogonal projection of y onto the space of X.

- **Orthogonality Condition:** The error vector (residual) $y - X\beta$ must be orthogonal to the column space of X.

$$X^T(y - X\beta) = 0$$

- **The Normal Equation:** Solving for β:

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

Probabilistic Interpretation (MLE)

If we assume the data is generated with Gaussian noise: $Y = X\beta + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$. The likelihood of observing the data is:

$$p(Y|X, \beta) \propto \exp\left(-\frac{||y - X\beta||^2}{2\sigma^2}\right)$$

Maximizing this likelihood (Maximum Likelihood Estimation - MLE) is equivalent to minimizing the negative log-likelihood, which results in the **Least Squares** objective:

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

2. Bayesian Viewpoint: MLE vs. MAP

Maximum Likelihood Estimation (MLE)

MLE estimates parameters θ by maximizing the likelihood of the data D:

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

- **Example (Bernoulli):** For coin flips with heads probability θ, if we observe H heads and T tails, $\hat{\theta}_{MLE} = \frac{H}{H+T}$.

Maximum A Posteriori (MAP)

MAP incorporates a **prior** belief $P(\theta)$ using Bayes' Theorem:

$$P(\theta|D) \propto P(D|\theta)P(\theta)$$

$$\hat{\theta}_{MAP} = \arg\max_{\theta}[\log P(D|\theta) + \log P(\theta)]$$

- **Conjugate Priors:** If the likelihood is Bernoulli, a **Beta distribution** prior ($\text{Beta}(\alpha, \beta)$) is computationally convenient.
- **Result:** The prior acts as "virtual counts." If prior is $\text{Beta}(\alpha, \beta)$, the estimate becomes:

$$\hat{\theta}_{MAP} = \frac{H + (\alpha - 1)}{H + T + (\alpha - 1) + (\beta - 1)}$$

  - *Insight:* As data size $N \to \infty$, the prior's influence vanishes, and $\hat{\theta}_{MAP} \to \hat{\theta}_{MLE}$.

3. Information Theory

- **Entropy ($H(X)$):** Measure of uncertainty. $H(X) = \mathbb{E}[-\log P(X)]$.
- **KL Divergence (D_{KL}):** Measures the distance between two distributions P and Q.

$$D_{KL}(P||Q) = \sum P(x) \log \frac{P(x)}{Q(x)} \geq 0$$

  - *Note:* It is asymmetric ($D_{KL}(P||Q) \neq D_{KL}(Q||P)$).
- **Cross Entropy ($H_P(Q)$):** Used as a loss function in classification.

$$H_P(Q) = -\sum P(x) \log Q(x) = H(P) + D_{KL}(P||Q)$$

Since $H(P)$ (the entropy of the true labels) is fixed, minimizing Cross Entropy is equivalent to minimizing the KL Divergence between the predicted distribution and the true distribution.

4. Regularization

Overfitting occurs when a model fits noise or is too complex for the data.

- **Ridge Regression (L2 Regularization):** Adds a penalty term $\lambda||\beta||^2$.

$$Loss = ||Y - X\beta||^2 + \lambda||\beta||^2$$

  - **Solution:** $\beta^* = (X^T X + \lambda I)^{-1} X^T y$. The λI term ensures the matrix is invertible (non-singular).

Kernel Methods

Regularization allows the use of the **Kernel Trick** to map data into high-dimensional spaces without computing coordinates explicitly.

- Represent β as a linear combination of data points: $\beta = \Phi(X)^T \alpha$.
- Prediction involves only dot products $K(x_1, x_2) = \phi(x_1)^T \phi(x_2)$.

5. Logistic Regression

Used for binary classification.

- **Sigmoid Function:** Squashes input z to $(0, 1)$.

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

  - *Derivative Property:* $\sigma'(z) = \sigma(z)(1 - \sigma(z))$.
- **Loss Function:** Negative Log-Likelihood (Binary Cross Entropy).

$$L = -\sum [y_i \log \sigma(w^T x_i) + (1 - y_i) \log(1 - \sigma(w^T x_i))]$$

- **Gradient:** The derivative is intuitive (Prediction - Target) $\times$ Input:

$$\nabla_w L = \sum(\sigma(w^T x_i) - y_i)x_i$$

- **Hessian:** $H = X^T S X$ where S is a diagonal matrix of variances. Since S is positive semi-definite, the loss surface is convex (guaranteeing a global minimum).

6. Support Vector Machines (SVM)

SVM seeks a hyperplane $w^T x + b = 0$ that maximizes the **margin** (distance to the nearest data points).

- **Geometry:** The distance from a point x_i to the hyperplane is $\frac{|w^T x_i + b|}{||w||}$.
- **Hard Margin Formulation:**
  We impose constraint $y_i(w^T x_i + b) \geq 1$. To maximize the margin $\frac{1}{||w||}$, we minimize $||w||^2$.

$$\min_{w,b} \frac{1}{2}||w||^2 \quad \text{subject to} \quad y_i(w^T x_i + b) \geq 1$$

- **Soft Margin:** Introduces slack variables for non-linearly separable data.
- **Dual Formulation:** Solved using Lagrange Multipliers (α_i). The solution depends only on support vectors (where $\alpha_i > 0$).

Part II: Deep Learning Optimization (Week 1)

1. Weight Initialization

Proper initialization is crucial to prevent vanishing or exploding gradients. The goal is to keep the variance of activations constant across layers.
Let $y = \sum w_i x_i$. If variances are independent: $Var(y) = N_{in} \cdot Var(w) \cdot Var(x)$.

- **Xavier Initialization:** Designed for **Sigmoid/Tanh** (linear regions).
  To maintain variance ($Var(y) = Var(x)$), we need $N_{in} Var(w) = 1$.

$$Var(w) = \frac{1}{N_{in}}$$

- **He Initialization:** Designed for **ReLU**.
  ReLU zeroes out half the neurons, halving the variance. We must double the weight variance to compensate.

$$Var(w) = \frac{2}{N_{in}}$$

2. Gradient Descent (GD) Theory

Smoothness and Convergence

A function f is β**-smooth** if its gradient does not change too fast (bounded by β).

$$||\nabla f(x) - \nabla f(y)|| \leq \beta||x - y||$$

Key Lemma: For a β-smooth function:

$$f(y) \leq f(x) + \langle \nabla f(x), y - x \rangle + \frac{\beta}{2}||y - x||^2$$

Substituting the GD update rule $x_{t+1} = x_t - \eta \nabla f(x_t)$:

$$f(x_{t+1}) \leq f(x_t) - \eta||\nabla f(x_t)||^2 + \frac{\beta\eta^2}{2}||\nabla f(x_t)||^2$$

If we choose step size $\eta = 1/\beta$, we guarantee descent.

Stochastic Gradient Descent (SGD)

Uses a mini-batch or single sample.

- **Convergence:** $O(1/\sqrt{T})$. Noise prevents perfect convergence to the critical point but allows escape from shallow local minima.

3. Advanced Optimizers

- **Momentum:** Accumulates a velocity vector v. helps dampen oscillations and pass through flat regions.

$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t)$$

$$x_{t+1} = x_t + v_{t+1}$$

- **Nesterov Momentum:** Calculates gradient at the "lookahead" position ($x_t + \rho v_t$) rather than current position.
- **Adagrad:** Adapts learning rates for each parameter. Large gradients $\rightarrow$ reduced learning rate.

$$x_{t+1} = x_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$$

  Drawback: Learning rate shrinks monotonically to zero.
- **RMSProp:** Solves Adagrad's shrinking issue by using an exponential moving average of squared gradients.
- **Adam:** Combines Momentum (1st moment) and RMSProp (2nd moment). Includes bias correction for early steps.

4. Batch Normalization (BN)

Addresses **Internal Covariate Shift** (distribution of layer inputs changing during training).

- **Mechanism:** Normalizes a batch $B = \{x_1, \dots, x_m\}$ to have mean 0 and variance 1, then scales and shifts.

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$y_i = \gamma \hat{x}_i + \beta$$

- **Benefits:** Allows higher learning rates, acts as a regularizer, and makes the loss landscape smoother (Lipschitz constant improves).

Part III: Convolutional Neural Networks (Week 5)

1. CNN Basics

- **Convolution Operation:** Translation equivariant. Defined by Kernel size (K), Stride (S), and Padding (P).
- **Output Dimension Calculation:**
  For input size N, the output size N' is:

$$N' = \left\lfloor \frac{N - K + 2P}{S} \right\rfloor + 1$$

- **Pooling:** (Max or Average) Reduces spatial dimensions, providing translation invariance.

2. Key Architectures

AlexNet (2012)

The breakthrough architecture.

- **Input:** $227 \times 227 \times 3$.
- **Key Features:** Used ReLU (solved saturation), Dropout (reduced overfitting), and Max Pooling.
- **Structure:** 5 Conv layers + 3 Fully Connected (FC) layers.
- **Stride dynamics:** Aggressive stride (4) in the first layer reduces dimension quickly (55×55).

ZFNet

An improvement on AlexNet based on visualizing feature maps.

- **Insight:** AlexNet's first layer filter (11×11) was too large and stride (4) too coarse, missing details.
- **Change:** Reduced 1st layer filter to 7×7 and stride to 2. Resulted in better feature capture.

GoogLeNet (Inception)

Addressed the problem of computational cost while going deeper (22 layers).

- **Inception Module:** Instead of choosing one filter size (3×3 or 5×5), use them all in parallel and concatenate results.
- **1x1 Convolution (Bottleneck):** Crucial innovation. Used to reduce depth (number of channels) *before* expensive 3×3 and 5×5 convolutions.
    - *Example:* Reducing $28 \times 28 \times 256$ input to a smaller depth saves parameters and computation (e.g., $854M$ ops vs $112M$ ops).
- **Auxiliary Classifiers:** Added small output networks at intermediate layers to inject gradients and combat the vanishing gradient problem during training.

ResNet (Residual Networks)

addressed the degradation problem: simply adding layers to "plain" networks caused training error to *increase* (not just overfitting, but optimization difficulty).

- **Residual Block:** Instead of learning mapping $H(x)$, the network learns the residual function $F(x) = H(x) - x$.

$$H(x) = F(x) + x$$

    - The "skip connection" (identity shortcut) allows gradients to flow directly through the network during backpropagation.
- **Initial Identity:** If optimal $H(x) \approx x$, the weights of $F(x)$ are driven to 0, which is easier than learning an identity matrix.
- **Architecture:** Heavily uses 3×3 convolutions and Global Average Pooling (no heavy FC layers at the end).
- **Bottleneck Block:** In deeper ResNets (50+ layers), uses 1×1 conv to reduce dimensions, 3×3 to process, and 1×1 to restore dimensions.