Part 1: Linear Models & Estimation Theory (Week 1)

1. Least Squares Estimation (LSE)

- **Objective:** Find β to minimize the squared error between observed y and predicted $X\beta$.
- **Geometric Interpretation:** Finding the orthogonal projection of y onto the column space of X. The error vector $y - X\beta$ must be orthogonal to the column space of X.
- **Derivation:**
  - Loss function: $L(\beta) = \|y - X\beta\|^2 = (y - X\beta)^T(y - X\beta)$
  - Expansion: $y^T y - 2\beta^T X^T y + \beta^T X^T X\beta$
  - Gradient: $\nabla_\beta L(\beta) = -2X^T y + 2X^T X\beta = 0$
  - **Normal Equation:** $X^T y = X^T X\beta \implies \beta = (X^T X)^{-1} X^T y$

2. Maximum Likelihood Estimation (MLE)

- **Assumption:** Gaussian noise. $Y = X\beta + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$.
- **Likelihood:** $P(Y|X, \beta) \propto \exp\left(-\frac{\|y-X\beta\|^2}{2\sigma^2}\right)$
- **Maximization:** Maximizing the likelihood is equivalent to minimizing the negative log-likelihood (NLL), which results in the **Least Squares** term $\|y - X\beta\|^2$. Thus, under Gaussian noise, $\beta_{MLE} = \beta_{LSE}$.

3. Bayesian Estimation (MAP)

- **Concept:** Incorporates a prior distribution $P(\theta)$.
- **Formula:** $P(\theta|Data) = \frac{P(Data|\theta)P(\theta)}{P(Data)}$
  - Posterior $\propto$ Likelihood $\times$ Prior.
- **Bernoulli Example (Coin Toss):**
  - Likelihood: $P(Data|\theta) = \theta^{\sum x_i}(1-\theta)^{n-\sum x_i}$
  - Prior: Beta Distribution $Beta(\alpha, \beta) \propto \theta^{\alpha-1}(1-\theta)^{\beta-1}$
  - Posterior: $Beta(\alpha + \sum x_i, \beta + n - \sum x_i)$
- **Estimates:**
  - $\theta_{MLE} = \frac{\sum x_i}{n}$ (Frequency ratio).
  - $\theta_{MAP} = \frac{\sum x_i + (\alpha-1)}{n+(\alpha+\beta-2)}$.
  - *Note:* If the Prior is Uniform ($\alpha = 1, \beta = 1$), then $\theta_{MAP} = \theta_{MLE}$.

4. Information Theory

- **Entropy:** $H(X) = E[-\log P(X)] \geq 0$. Measure of uncertainty.
- **KL Divergence:** $D_{KL}(P\|Q) = \sum P(x) \log \frac{P(x)}{Q(x)}$.

    - Properties: Non-negative ($D_{KL} \geq 0$, via Jensen's inequality), Asymmetric ($D_{KL}(P||Q) \neq D_{KL}(Q||P)$).
- **Mutual Information:** $I(X;Y) = D_{KL}(P(x,y)||P(x)P(y)) = H(X) - H(X|Y)$. Represents reduction in uncertainty of X given Y.
- **Cross Entropy:** $H_P(Q) = -\sum P(x) \log Q(x) = H(P) + D_{KL}(P||Q)$. Minimizing Cross Entropy is equivalent to minimizing KL Divergence (if $H(P)$ is fixed).

5. Regularization & Kernel Methods

- **Ridge Regression (L2):** Adds penalty $\lambda||\beta||^2$.
    - Objective: $||Y - X\beta||^2 + \lambda||\beta||^2$.
    - Solution: $\beta = (X^T X + \lambda I)^{-1} X^T y$.
- **Kernel Trick:** Mapping low-dimensional x to high-dimensional feature space $\phi(x)$.
    - Kernel function: $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$.
    - **Kernel Ridge Regression:**
        - Dual coefficients: $\alpha^* = (K(X,X) + \lambda I)^{-1} Y$.
        - Prediction: $f^*(x) = K(x, X)\alpha^*$.

6. Logistic Regression

- **Sigmoid Function:** $\sigma(z) = \frac{1}{1+e^{-z}}$.
    - Derivative: $\sigma'(z) = \sigma(z)(1 - \sigma(z))$.
- **Binary Classification (MLE):**
    - Model: $P(y=1|x) = \sigma(w^T x)$.
    - Loss: Negative Log Likelihood (Cross Entropy).
    - Gradient: $\nabla_w L = X^T(\sigma(Xw) - y)$.
    - **Hessian:** $H = X^T S X$ where S is a diagonal matrix of $\sigma_i(1-\sigma_i)$. Since $S \succ 0$, the Hessian is positive definite, implying the loss function is **convex** (global minimum exists).

7. Multiclass Classification (Softmax)

- **Softmax Function:** $p_k = \frac{e^{z_k}}{\sum_j e^{z_j}}$ where $z_k = w_k^T x$.
- **Gradient Derivation:**
    - $\frac{\partial p_m}{\partial z_k} = p_k(\delta_{mk} - p_m)$.
    - Loss Gradient: $\nabla_{w_m}(-\log p_{target}) = (p_m - y_m)x$.

8. Support Vector Machine (SVM)

- **Hyperplane:** $w^T x + b = 0$.
- **Margin:** The distance from the hyperplane to the nearest point is $d = \frac{1}{||w||}$.

- **Primal Problem:**
  - Minimize $\frac{1}{2}||w||^2 + C\sum \xi_i$ (soft margin formulation with slack variables).
  - Constraint: $y_i(w^T x_i + b) \geq 1 - \xi_i$.
- **Dual Problem:**
  - Use Lagrange Multipliers (α_i).
  - Maximize: $\sum \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j x_i^T x_j$.
  - Subject to: $\sum \alpha_i y_i = 0$ and $0 \leq \alpha_i \leq C$.
  - The optimal w is a linear combination of support vectors: $w = \sum \alpha_i y_i x_i$.

Part 2: Neural Networks & Optimization (Week 1)

1. Activation Functions

- **Sigmoid:** $\frac{1}{1+e^{-z}}$. Range $[0, 1]$. Problem: Vanishing gradient.
- **Tanh:** $\frac{e^z - e^{-z}}{e^z + e^{-z}}$. Range $[-1, 1]$. Zero-centered.
- **ReLU:** $\max(0, z)$. Solves vanishing gradient for $z > 0$. Derivative is 1 or 0.

2. Weight Initialization

- Goal: Keep the variance of activations consistent across layers to prevent exploding/vanishing gradients.
- **Variance Analysis:** Let $Var(y) = D_{in} Var(w) E[x^2]$.
- **Xavier (Glorot) Initialization:**
  - For Sigmoid/Tanh (linear region assumption).
  - Set $Var(w) = \frac{1}{D_{in}}$ (or $\frac{2}{D_{in} + D_{out}}$).
  - Uniform Dist: $U(-\sqrt{\frac{3}{D_{in}}}, \sqrt{\frac{3}{D_{in}}})$.
- **He Initialization:**
  - For ReLU. Since ReLU zeroes out half the inputs, variance is halved.
  - Set $Var(w) = \frac{2}{D_{in}}$.

3. Learning Rate Schedulers

- **Step Decay:** Reduce LR at fixed intervals.
- **Linear Warmup:** Linearly increase LR from 0 to target at the start.
- **Cosine Decay:** $\eta_t = \frac{1}{2}\eta_0(1 + \cos(\frac{t\pi}{T}))$.

4. Batch Normalization (BN)

- **Algorithm:**
  i. Calculate mean μ_B and variance σ_B^2 of the mini-batch.
  ii. Normalize: $\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$.
  iii. Scale and Shift: $y_i = \gamma \hat{x}_i + \beta$ (Learnable parameters).
- **Inference:** Use moving averages of μ and σ^2 collected during training.
- **Benefits:** Reduces internal covariate shift, allows higher learning rates, smoother loss landscape.

5. Optimization Theory & Algorithms

- **Gradient Descent (GD) Analysis:**
  - **β-Smoothness:** $\|\nabla f(x) - \nabla f(y)\| \leq \beta \|x - y\|$.
  - **Convergence:** For convex, smooth functions, error decreases at rate $O(1/T)$ or $O(1/\sqrt{T})$ depending on setting.
- **Optimizers:**
  - **SGD:** $x_{t+1} = x_t - \eta \nabla f(x_t)$. Noisy but faster per step.
  - **Momentum:** Accumulates velocity vector. $v_{t+1} = \rho v_t - \alpha \nabla f(x_t)$.
  - **Nesterov Momentum:** Computes gradient at the "lookahead" position ($x_t + \rho v_t$).
  - **AdaGrad:** Scales learning rate by sum of squared past gradients. (Good for sparse data, but LR decays to 0).
  - **RMSProp:** Uses exponential moving average of squared gradients to fix AdaGrad's decay.
  - **Adam:** Combines Momentum (1st moment) and RMSProp (2nd moment).
    - $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$
    - $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
    - Bias correction: $\hat{m}_t = m_t / (1 - \beta_1^t)$, $\hat{v}_t = v_t / (1 - \beta_2^t)$.
    - Update: $x_{t+1} = x_t - \frac{\eta \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$.

Part 3: Convolutional Neural Networks (Week 5)

1. Basic Components

- **Convolution Layer:** Preserves spatial structure.
  - Input: $N \times N \times C$. Filter: $K \times K \times C$. Number of filters: D.
  - Output Size: $N_{out} = \frac{N - K + 2P}{S} + 1$.
  - Output Volume: $N_{out} \times N_{out} \times D$.
- **Pooling:** Downsampling (Max Pooling, Average Pooling). No parameters to learn.

- 1×1 **Convolution:** Used to change channel depth (C) without changing spatial resolution.
- **Properties:** Translation Invariance/Equivariance.

2. Key Architectures

- **AlexNet (2012):**
    - First major Deep CNN success.
    - Input: $227 \times 227 \times 3$.
    - Architecture: 5 Conv layers, 3 Fully Connected (FC) layers.
    - Key features: ReLU, Dropout (50% in FC layers to prevent overfitting), Local Response Normalization (obsolete now), Data Augmentation.
- **ZFNet:** Refined AlexNet (smaller strides/filters in early layers) based on visualization.
- **GoogLeNet (Inception):**
    - **Inception Module:** Concatenates outputs from 1×1, 3×3, 5×5 convs and 3×3 pooling.
    - **Bottleneck:** Uses 1×1 convolutions *before* expensive 3×3 or 5×5 convs to reduce channel dimensions and computational cost.
    - **Auxiliary Classifiers:** Inject gradients at intermediate layers to help training deep networks.
- **ResNet (Residual Networks):**
    - **Problem:** Deep networks were harder to train (degradation problem, not just overfitting).
    - **Solution:** Residual Block. Learn mapping $F(x) + x$ instead of direct mapping $H(x)$.
    - **Architecture:**
        - **Plain Block:** 3×3 Conv $\rightarrow$ ReLU $\rightarrow$ 3×3 Conv $\rightarrow$ Addition.
        - **Bottleneck Block:** 1×1 (reduce dim) $\rightarrow 3 \times 3 \rightarrow 1 \times 1$ (restore dim). Used in deeper ResNets (e.g., ResNet-50/101).
    - **He Initialization** is crucial here.

3. Visualization

- **Saliency Maps:** Compute gradient of class score with respect to input image pixels to visualize which parts of the image influenced the decision.