Probability Basics:
$\mathbb{P}((X,Y) \in A) = \int_A p(x,y)\,dx\,dy$.
$p(x) = \int_{-\infty}^{\infty} p(x,y)\,dy$. $p(y|x) = \frac{p(x,y)}{p(x)}$.
$p(x,y) = p(x)p(y|x)$, If X and Y are independent, then $p(x,y) = p(x)p(y)$.
Likelihood of i.i.d. dataset: $p(\mathcal{D}) = \prod_{i=1}^n p(x_i)$ or $\log p(\mathcal{D}) = \sum_{i=1}^n \log p(x_i)$. where dataset $\mathcal{D} = \{x_1, \ldots, x_n\}$ is i.i.d.
Bayes' theorem: $p(x|y) = \frac{p(x,y)}{p(y)} = \frac{p(y|x)p(x)}{p(y)}$.
Expectation:
$\mathbb{E}[X] = \int_{-\infty}^{\infty} xp(x)\,dx$, $\mathbb{E}[g(X)] = \int_{-\infty}^{\infty} g(x)p(x)\,dx$,
$\mathbb{E}[X+Y] = \mathbb{E}[X] + \mathbb{E}[Y]$,
$\mathbb{E}[X \mid Y=y] = \int_{-\infty}^{\infty} xp(x \mid y)\,dx$,
$\mathbb{E}[f(X)g(Y) \mid Y=y] = \mathbb{E}[f(X) \mid Y=y]g(y)$.
Variance and Covariance:
$\mathrm{var}(X) = \mathbb{E}[(X-\mathbb{E}[X])^2] = \mathbb{E}[X^2] - \mathbb{E}[X]^2$, $\mathrm{cov}(X,Y) = \mathbb{E}[(X-\mathbb{E}[X])(Y-\mathbb{E}[Y])] = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y]$,
$\mathrm{var}(X+Y) = \mathrm{var}(X) + \mathrm{var}(Y) + 2\mathrm{cov}(X,Y)$.
Vector Variables: $\mathbf{x} = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}^\top$, $\mathbb{E}[\mathbf{x}] = \begin{bmatrix} \mathbb{E}[x_1] & \mathbb{E}[x_2] & \cdots & \mathbb{E}[x_n] \end{bmatrix}^\top$
If $\mathbf{A}$ is a deterministic matrix, $\mathbb{E}[\mathbf{A}\mathbf{x}] = \mathbf{A}\mathbb{E}[\mathbf{x}]$.
Covariance: $\mathrm{cov}(\mathbf{x}) = \mathbf{\Sigma}_{xx} = \mathbb{E}[(\mathbf{x} - \mathbb{E}\mathbf{x})(\mathbf{x} - \mathbb{E}\mathbf{x})^\top]$,
$\mathrm{cov}(\mathbf{A}\mathbf{x}) = \mathbf{A}\,\mathrm{cov}(\mathbf{x})\mathbf{A}^\top$
Cross-covariance: $\mathrm{cov}(\mathbf{x},\mathbf{y}) = \mathbf{\Sigma}_{xy} = \mathbb{E}[(\mathbf{x} - \mathbb{E}\mathbf{x})(\mathbf{y} - \mathbb{E}\mathbf{y})^\top]$
Matrix Calculus: $\frac{\partial(\mathbf{a}^\top\mathbf{x})}{\partial\mathbf{x}} = \mathbf{a}$, $\frac{\partial(\mathbf{x}^\top\mathbf{A}\mathbf{x})}{\partial\mathbf{x}} = (\mathbf{A} + \mathbf{A}^\top)\mathbf{x}$
$\frac{\partial(\mathbf{a}^\top\mathbf{X}\mathbf{b})}{\partial\mathbf{X}} = \mathbf{a}\mathbf{b}^\top$, $\frac{\partial\det(\mathbf{X})}{\partial\mathbf{X}} = \det(\mathbf{X})(\mathbf{X}^{-1})^\top$,
$\frac{\partial(\mathbf{a}^\top\mathbf{X}^{-1}\mathbf{b})}{\partial\mathbf{X}} = -(\mathbf{X}^{-1})^\top\mathbf{a}\mathbf{b}^\top(\mathbf{X}^{-1})^\top$
Bayesian Inference:
Given a parametric model, the posterior is derived as:
$p(\theta \mid x) = \frac{p(x|\theta)p(\theta)}{p(x)} \propto p(x \mid \theta)p(\theta)$, where $p(x)$ is a normalization constant. If $p(x) \propto f(x)$ for some function $f(x)$, then $p(x) = cf(x)$ with $c = (\int f(x)dx)^{-1}$.
Conjugate Distributions:
If prior and posterior share the same form, they are conjugate: $p(\theta \mid x) \propto p(x \mid \theta)p(\theta)$. The prior $p(\theta)$ is called the conjugate prior of the likelihood $p(x \mid \theta)$.
1. Allows for analytical closed form solutions and easy to interpret.
2. May lack flexibility to complex data, requiring MCMC.
Conjugate Prior for Binomial:
Given $s \sim \mathrm{Bin}(n,\theta)$ and prior $\theta \sim \mathrm{Beta}(a,b)$, the posterior is:
$p(\theta \mid s) \propto \mathrm{Bin}(s \mid \theta, n) \cdot \mathrm{Beta}(\theta \mid a,b)$
$\propto \theta^s(1-\theta)^{n-s} \cdot \theta^{a-1}(1-\theta)^{b-1}$
$\propto \theta^{s+a-1}(1-\theta)^{n-s+b-1}$. Thus, the posterior follows:
$p(\theta \mid s) = \mathrm{Beta}(\theta \mid s+a, n-s+b)$
Categorical Distribution:
• A categorical variable X follows: $\mathrm{Cat}(x \mid \theta_1, \ldots, \theta_K)$ with parameters $\theta_k \geq 0, \sum_{k=1}^K \theta_k = 1$; $\mathrm{Cat}(x \mid \theta_1, \ldots, \theta_K) = \theta_x$.
• Given i.i.d. samples $X_i \sim \mathrm{Cat}(\theta_1, \ldots, \theta_K)$, the joint probability of $\mathcal{D} = \{X_1, \ldots, X_n\}$ is: $p(\mathcal{D}) = \prod_{i=1}^n \mathrm{Cat}(x_i \mid \theta_1, \ldots, \theta_K) = \prod_{i=1}^n \prod_{k=1}^K \theta_k^{\mathbb{1}\{x_i=k\}}$. Using count notation $N_k = \sum_{i=1}^n \mathbb{1}\{x_i = k\}$, we get: $p(\mathcal{D}) = \prod_{k=1}^K \theta_k^{N_k}$.
Gaussian (Normal) Distribution:
A random variable X follows a normal distribution:
$\mathcal{N}(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(x-\mu)^2}$.
If $X \sim \mathcal{N}(\mu,\sigma^2)$, then: $aX \sim \mathcal{N}(a\mu, a^2\sigma^2)$ for any $a \in \mathbb{R}$ $X + c \sim \mathcal{N}(\mu+c, \sigma^2)$. If $Z \sim \mathcal{N}(0,1)$, then $X = \sigma Z + \mu \sim \mathcal{N}(\mu,\sigma^2)$. If $X \sim \mathcal{N}(\mu,\sigma^2)$ and $Y \sim \mathcal{N}(\xi,\nu^2)$ are independent: $X + Y \sim \mathcal{N}(\mu+\xi, \sigma^2+\nu^2)$.
Multivariate Gaussian Distribution:
A random vector $\mathbf{X}$ follows a multivariate normal distribution:
$\mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}, \mathbf{\Sigma}) = \frac{1}{(2\pi)^{K/2}(\det\mathbf{\Sigma})^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top\mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)$.
• $\mathbf{X} = [X_1, \ldots, X_n]^\top$ is *jointly Gaussian* if for any vector $\mathbf{a} \in \mathbb{R}^n$, the linear combination: $\mathbf{a}^\top\mathbf{X} = \sum_{i=1}^n a_i X_i$ is Gaussian.
• If $\mathbf{Z} \sim \mathcal{N}(\mathbf{0},\mathbf{I})$, then: $\mathbf{X} = \mathbf{A}\mathbf{Z} + \boldsymbol{\mu}$ is jointly Gaussian with mean and covariance: $\mathbb{E}[\mathbf{X}] = \boldsymbol{\mu}$, $\mathrm{cov}(\mathbf{X}) = \mathbf{A}\mathbf{A}^\top$.
• $\boldsymbol{\mu} = \begin{bmatrix} \mu_X \\ \mu_Y \end{bmatrix}$, $\mathbf{\Sigma} = \begin{bmatrix} \sigma_X^2 & \mathrm{cov}(X,Y) \\ \mathrm{cov}(X,Y) & \sigma_Y^2 \end{bmatrix}$.
$\rho_{X,Y} = \frac{\mathrm{cov}(X,Y)}{\sigma_X\sigma_Y}$.
Likelihood Functions for Common Distributions:
1. Bernoulli: If $x_i \sim \mathrm{Bern}(\theta)$ i.i.d., where $\theta \in [0,1]$, then:
$p(\mathcal{D} \mid \theta) = \theta^{N_1}(1-\theta)^{N_0}$, $N_k = \sum_{i=1}^n \mathbb{1}\{x_i = k\}$.
2. Exponential: If $x_i \sim \mathrm{Exp}(\lambda)$ i.i.d., where $\lambda > 0$, then:
$p(\mathcal{D} \mid \lambda) = \lambda^n \exp\left(-\lambda\sum_{i=1}^n x_i\right)$.
3. Gaussian: If $x_i \sim \mathcal{N}(\mu,\sigma^2)$ i.i.d., then $\theta = (\mu,\sigma^2)$ and:
$p(\mathcal{D} \mid \theta) = \frac{1}{(2\pi)^{n/2}\sigma^n} \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^n (x_i-\mu)^2\right)$.
Maximum Likelihood Estimation (MLE):
The MLE of parameter θ is: $\theta_{\mathrm{ML}} = \arg\max_\theta p(\mathcal{D} \mid \theta)$. For i.i.d. data $\mathcal{D} = \{x_1, \ldots, x_n\}$, we maximize the log-likelihood:
$\log p(\mathcal{D} \mid \theta) = \sum_{i=1}^n \log p(x_i \mid \theta)$.
Bernoulli MLE: If $x_i \sim \mathrm{Bern}(\theta)$, then:
$p(\mathcal{D} \mid \theta) = \theta^{N_1}(1-\theta)^{N_0}$,
$\log p(\mathcal{D} \mid \theta) = N_1\log\theta + N_0\log(1-\theta)$. Solving
$\frac{\partial}{\partial\theta}\log p(\mathcal{D} \mid \theta) = 0$: $\theta_{\mathrm{ML}} = \frac{N_1}{N_0+N_1} = \frac{N_1}{n}$.
Exponential MLE: If $x_i \sim \mathrm{Exp}(\lambda)$, then:
$\log p(\mathcal{D} \mid \lambda) = n\log\lambda - \lambda\sum_{i=1}^n x_i$. Solving
$\frac{\partial}{\partial\lambda}\log p(\mathcal{D} \mid \lambda) = 0$: $\lambda_{\mathrm{ML}} = \frac{n}{\sum_{i=1}^n x_i}$.
Linear Regression Model:
$\mathbf{x} = (x_1, x_2, \ldots, x_D)$, $y = \mathbf{w}^\top\mathbf{x} + \epsilon$, $\epsilon \sim \mathcal{N}(0,\sigma^2)$.
Thus, the likelihood is: $p(y \mid \mathbf{x}, \mathbf{w}) = \mathcal{N}(y \mid \mathbf{w}^\top\mathbf{x}, \sigma^2)$.
Basis Function Expansion: model non-linear relationships using basis functions: $\phi(\mathbf{x}) = \begin{bmatrix} \varphi_1(\mathbf{x}) & \varphi_2(\mathbf{x}) & \cdots & \varphi_M(\mathbf{x}) \end{bmatrix}^\top$,
$y = \mathbf{w}^\top\phi(\mathbf{x}) + \epsilon$; For $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$, $\phi(\mathbf{x}) = [1, x_1, x_2, x_1^2, x_2^2]$,
$y = w_1 + w_2x_1 + w_3x_2 + w_4x_1^2 + w_5x_2^2 + \epsilon$.
MLE for w: Given i.i.d. training data $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}$,
$\log p(\mathbf{y} \mid \mathbf{\Phi}, \mathbf{w}) = -\frac{1}{2\sigma^2}\|\mathbf{\Phi}\mathbf{w} - \mathbf{y}\|^2 + \text{const}$, where $\mathbf{\Phi} = [\mathbf{x}_1 \ldots \mathbf{x}_n]^\top$ is the design matrix. Maximizing w.r.t. $\mathbf{w}$ gives the least squares solution: $\mathbf{w}_{\mathrm{ML}} = (\mathbf{\Phi}^\top\mathbf{\Phi})^{-1}\mathbf{\Phi}^\top\mathbf{y}$.
Model Evaluation Metrics:
• $\mathrm{RSS} = \sum_{i=1}^n (y_i - \hat{y}_i)^2$. • $\mathrm{RMSE} = \sqrt{\frac{1}{n}\mathrm{RSS}}$.
• $R^2 = 1 - \frac{\mathrm{RSS}}{\mathrm{TSS}} = 1 - \frac{\mathrm{RSS}}{\sum_{i=1}^n (y_i - \bar{y})^2}$,
Maximum A Posteriori (MAP) Estimation:
• The MAP estimate maximizes the posterior:
$\theta_{\mathrm{MAP}} = \arg\max_\theta p(\theta \mid \mathcal{D})$. Using Bayes' theorem:
$p(\theta \mid \mathcal{D}) \propto p(\mathcal{D} \mid \theta)p(\theta)$,
$\log p(\theta \mid \mathcal{D}) = \log p(\mathcal{D} \mid \theta) + \log p(\theta) + \text{const}$.
• Given: $p(\mathcal{D} \mid \theta) = \theta^{N_1}(1-\theta)^{N_0}$, $p(\theta) = \mathrm{Beta}(\theta \mid a,b)$,
$\log p(\mathcal{D} \mid \theta)p(\theta) = (N_1 + a - 1)\log\theta + (N_0 + b - 1)\log(1-\theta)$. Solving
$\frac{\partial}{\partial\theta}\log p(\mathcal{D} \mid \theta) = 0$: $\theta_{\mathrm{MAP}} = \frac{N_1+a-1}{n+a+b-2}$.
Classification and Naïve Bayes:
• Classification Rule: Given feature vector $\mathbf{x}$ and class label $y \in \{1, \ldots, K\}$, $\delta(\mathbf{x}) = k$ if $p(y = k \mid \mathbf{x})$ is maximized.
• Naïve Bayes Classifier: Assuming conditional independence, $p(\mathbf{x} \mid y = c, \theta) = \prod_{d=1}^D p(x_d \mid \theta_{dc})$. Using Bayes' rule:
$p(y = c \mid \mathbf{x}, \theta) \propto \pi(c)\prod_{d=1}^D p(x_d \mid \theta_{dc})$, where $\pi(c)$ is the prior probability of class c.
Mixture Models and Gaussian Mixture Model (GMM):
• Suppose an observation $\mathbf{x}$ can be generated from one of K possible probability density functions (pdfs): $p(\mathbf{x} \mid \boldsymbol{\eta}_1), \ldots, p(\mathbf{x} \mid \boldsymbol{\eta}_K)$.
The generating index z follows a categorical distribution:
$p(z) = \mathrm{Cat}(z \mid \boldsymbol{\pi})$. Since z is unobserved, it is a **latent variable**.
The marginal distribution is: $p(\mathbf{x} \mid \boldsymbol{\theta}) = \sum_{k=1}^K \pi(k)p(\mathbf{x} \mid \boldsymbol{\eta}_k)$, where $\boldsymbol{\theta} = (\boldsymbol{\pi}, \{\boldsymbol{\eta}_k\}_{k=1}^K)$.
• GMM: $p(\mathbf{x} \mid \boldsymbol{\theta}) = \sum_{k=1}^K \pi(k)\mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \mathbf{\Sigma}_k)$, where $\boldsymbol{\theta} = (\boldsymbol{\pi}, \{\boldsymbol{\mu}_k, \mathbf{\Sigma}_k\}_{k=1}^K)$. Given i.i.d. observations $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$, $p(\mathbf{x}_i \mid \boldsymbol{\theta}) = \sum_{k=1}^K \pi(k)p(\mathbf{x}_i \mid \boldsymbol{\eta}_k)$.
$\log p(\mathbf{x}_1, \ldots, \mathbf{x}_n \mid \boldsymbol{\theta}) = \sum_{i=1}^n \log\sum_{k=1}^K \pi(k)p(\mathbf{x}_i \mid \boldsymbol{\eta}_k)$.
Algorithmic Issues in Mixture Models:
• Singularity: If for some k, we set $\boldsymbol{\mu}_k = \mathbf{x}_i$ and $\sigma_k \to 0$, then: $\mathcal{N}(\mathbf{x}_i \mid \boldsymbol{\mu}_k, \sigma_k\mathbf{I}) \propto \frac{1}{\sigma_k} \to \infty$.
• Unidentifiability: no unique global optimum for log-likelihood function.
• Optimization Challenges: non-convex, hard to solve.
• If latent variables $z_1, \ldots, z_n$ are observed, the likelihood simplifies to: $\log p((\mathbf{x}_1, z_1), \ldots, (\mathbf{x}_n, z_n) \mid \boldsymbol{\theta}) = \sum_{i=1}^n \left(\log\pi[z_i] + \log p(\mathbf{x}_i \mid \boldsymbol{\eta}_{z_i})\right)$. much easier to maximize.
Gaussian Mixture Model (GMM):
• Observed data $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^D$ are generated from a mixture of K Gaussian distributions:
$p(\mathbf{x} \mid \boldsymbol{\theta}) = \sum_{k=1}^K \pi(k)\mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \mathbf{\Sigma}_k)$
$\boldsymbol{\theta} = (\pi(k), \boldsymbol{\mu}_k, \mathbf{\Sigma}_k)_{k=1}^K$.
• Complete data likelihood: $\log p(\mathbf{y}_1, \ldots, \mathbf{y}_n \mid \boldsymbol{\theta}) = \sum_{k=1}^K \sum_{i:z_i=k} \left(\log\pi(k) + \log\mathcal{N}(\mathbf{x}_i \mid \boldsymbol{\mu}_k, \mathbf{\Sigma}_k)\right)$..
• The MLE estimates for $\boldsymbol{\mu}_k$ and $\mathbf{\Sigma}_k$: $\hat{\boldsymbol{\mu}}_k = \frac{1}{n}\sum_{i:z_i=k}\mathbf{x}_i$,
$\hat{\mathbf{\Sigma}}_k = \frac{1}{n}\sum_{i:z_i=k}(\mathbf{x}_i - \hat{\boldsymbol{\mu}}_k)(\mathbf{x}_i - \hat{\boldsymbol{\mu}}_k)^\top$.
Expectation-Maximization (EM) Algorithm:
Complete data $\mathbf{y}$ can't be known directly. Still since $\log p(\mathbf{x} \mid \boldsymbol{\theta})$ is hard to optimize. Try maximize the expectation
$\mathbb{E}_{p(\mathbf{y}|\mathbf{x},\hat{\boldsymbol{\theta}})}\left[\log p(\mathbf{y} \mid \boldsymbol{\theta}) \mid \mathbf{x}, \hat{\boldsymbol{\theta}}\right]$.
EM Steps: 1. Initialize $\boldsymbol{\theta}^{(0)}$. 2. **E-step**: Compute
$Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}^{(m)}) = \mathbb{E}_{p(\mathbf{y}|\mathbf{x},\boldsymbol{\theta}^{(m)})}\left[\log p(\mathbf{y} \mid \boldsymbol{\theta}) \mid \mathbf{x}, \boldsymbol{\theta}^{(m)}\right] = \int \log p(\mathbf{y} \mid \boldsymbol{\theta})p(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta}^{(m)})d\mathbf{y}$.
3. **M-step**: Update $\boldsymbol{\theta}$ by $\boldsymbol{\theta}^{(m+1)} = \arg\max_{\boldsymbol{\theta}\in\Theta} Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}^{(m)})$.
4. Repeat until convergence.
EM for GMM:
Given $\pi^{(0)}(k), \boldsymbol{\mu}_k^{(0)}, \mathbf{\Sigma}_k^{(0)}$ for $k = 1, \ldots, K$.
$L^{(0)} = \frac{1}{n}\sum_{i=1}^n \log\left(\sum_{k=1}^K \pi^{(0)}(k)\mathcal{N}(\mathbf{x}_i \mid \boldsymbol{\mu}_k^{(0)}, \mathbf{\Sigma}_k^{(0)})\right)$.
Repeat:
1. **E-step:** Compute responsibilities:
$r_{ik}^{(m)} = \frac{\pi^{(m)}(k)\mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_k^{(m)}, \mathbf{\Sigma}_k^{(m)})}{\sum_{k'}\pi^{(m)}(k')\mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_{k'}^{(m)}, \mathbf{\Sigma}_{k'}^{(m)})}$, $n_k^{(m)} = \sum_{i=1}^n r_{ik}$.
2. **M-step:** Update parameters: $\pi^{(m+1)}(k) = \frac{n_k^{(m)}}{n}$,
$\boldsymbol{\mu}_k^{(m+1)} = \frac{1}{n_k^{(m)}}\sum_{i=1}^n r_{ik}^{(m)}\mathbf{x}_i$, $\mathbf{\Sigma}_k^{(m+1)} = \frac{1}{n_k^{(m)}}\sum_{i=1}^n r_{ik}^{(m)}(\mathbf{x}_i - \boldsymbol{\mu}_k^{(m+1)})(\mathbf{x}_i - \boldsymbol{\mu}_k^{(m+1)})^\top$.
3. **Compute:** $L^{(m+1)} = \frac{1}{n}\sum_{i=1}^n \log\left(\sum_{k=1}^K \pi^{(m+1)}(k)\mathcal{N}(\mathbf{x}_i \mid \boldsymbol{\mu}_k^{(m+1)}, \mathbf{\Sigma}_k^{(m+1)})\right)$.
4. **untill:** $|L^{(m+1)} - L^{(m)}| \leq \epsilon$.
K-Means Algorithm
GMM with $\Sigma_k = \sigma^2 I$ and $\pi(k) = 1/K$ are fixed. Only μ_k are inferred. 1. **E-step:** Assign each x_i to its nearest cluster center:
$k_i = \arg\min_k \|x_i - \mu_k^{(m)}\|^2$. Define hard assignment:
$r_{ik}^{(m)} = 1$ if $k = k_i$, otherwise $r_{ik}^{(m)} = 0$.
$Q(\theta \mid \theta^{(m)}) = -\frac{1}{2\sigma^2}\sum_{i=1}^n \|x_i - \mu_{k_i}^{(m)}\|^2 + \text{const}$.
2. **M-step:** Update cluster centers: $\mu_k^{(m+1)} = \frac{1}{N_k}\sum_{i:k_i=k} x_i$, where $N_k = \sum_{i=1}^n \mathbb{1}\{k_i = k\}$.
3. Update each cluster center using the mean of its assigned points.
EM Algorithm for MAP Estimation
Given data $\mathbf{x}$, the posterior is $p(\theta \mid \mathbf{x}) = \frac{p(\mathbf{x}|\theta)p(\theta)}{p(\mathbf{x})}$. The MAP estimate is $\theta_{MAP} = \arg\max_\theta(\log p(\mathbf{x} \mid \theta) + \log p(\theta))$.
Algorithm:
1. Pick initial guess $\theta^{(0)}$.
2. **E-step:** At iteration $m+1$, compute
$Q(\theta \mid \theta^{(m)}) = \int p(\mathbf{y} \mid \mathbf{x}, \theta^{(m)})\log p(\mathbf{y} \mid \theta)d\mathbf{y}$.
3. **M-step:** Update
$\theta^{(m+1)} = \arg\max_\theta\left(Q(\theta \mid \theta^{(m)}) + \log p(\theta)\right)$.
4. Repeat until convergence.
Transition Matrix
• Each row of $\mathbf{T}$ sums to one:
$\sum_{j=1}^M T(i,j) = \sum_{j=1}^M p_{x[t]|x[t-1]}(j \mid i) = 1$. $\mathbf{T}$ is a (row) stochastic matrix.
• Given initial distribution $\mathbf{p}_0 = [p_0(1), p_0(2), \ldots, p_0(M)]$, $\mathbf{p}_1 = \mathbf{p}_0\mathbf{T}$. For general t, let $\mathbf{p}_t = [p_t(1), \ldots, p_t(M)]$, then: $\mathbf{p}_t = \mathbf{p}_0\mathbf{T}^t$.
MLE for Transition Matrix
• Estimate prior π and transition matrix $\mathbf{T}$ from training data:
$p(x[0], \ldots, x[t] \mid \pi, \mathbf{T}) = \pi(x[0])\mathbf{T}(x[0], x[1]) \ldots \mathbf{T}(x[t-1], x[t])$. Given n observed sequences $\mathcal{D} = \{\mathbf{x}_1[0:t_1], \ldots, \mathbf{x}_n[0:t_n]\}$, each of varying length $t_i + 1$, assume all data points follow the same $\mathbf{T}$.
• $\log p(\mathcal{D} \mid \pi, \mathbf{T}) = \sum_{i=1}^n \log\pi(x_i[0]) + \sum_{i=1}^n\sum_{t=1}^{t_i} \log\mathbf{T}(x_i[t-1], x_i[t])$
$= \sum_{x=1}^M N_x\log\pi(x) + \sum_{x=1}^M\sum_{y=1}^M N_{xy}\log\mathbf{T}(x,y)$.
$N_x = \sum_{i=1}^n \mathbb{1}(x_i[0] = x)$,
$N_{xy} = \sum_{i=1}^n\sum_{t=1}^{t_i} \mathbb{1}(x_i[t-1] = x, x_i[t] = y)$.
• $\hat{\pi}(x) = \frac{N_x}{n}$, $\hat{\mathbf{T}}(x,y) = \frac{N_{xy}}{\sum_{z=1}^M N_{xz}}$.
• May predict certain strings are impossible if data have zero count of certain states (this is a form of overfitting).
HMM: Hidden Markov Model
A hidden Markov model consists of:
• A discrete state Markov chain with **hidden states** or latent variables $z[t] \in \{1, \ldots, M\}$, $t = 0, 1, \ldots$, with initial pdf π and transition matrix $\mathbf{T}$.
• An observation model with emission probabilities $p(\mathbf{x}[t] \mid z[t]) = p(\mathbf{x}[t] \mid \phi_{z[t]})$, where $\phi = (\phi_1, \ldots, \phi_M)$.
Applications: (1) Long-range dependencies; (2) Speech recognition; (3) Gene finding; (4) Emission probabilities (Gaussian example).
Baum-Welch Algorithm for HMM Training
1. Initialize $\theta^{(0)}$.
2. **E step**: Use Forward-Backward Algorithm to compute
$\gamma_{i,t}(z) = p(z_i[t] = z|\mathbf{x}_i[0:t_i], \theta^{(m)}) \propto \alpha_j(z)\beta_j(z)$.
$\xi_{i,t}(z,z') = p(z_i[t-1] = z, z_i[t] = z'|\mathbf{x}_i[0:t_i], \theta^{(m)}) \propto \alpha_{t-1}(z)p(\mathbf{x}_i[t]|z_i[t] = z')\beta_t(z')p(z_i[t] = z'|z_i[t-1] = z)$.
3. **M step**: Update parameters $\hat{\pi}(z) = \frac{\sum_{i=1}^n \gamma_{i,0}(z)}{n}$,
$\hat{T}(z,z') = \frac{\sum_{i=1}^n\sum_{t=1}^{t_i}\xi_{i,t}(z,z')}{\sum_u\sum_{i=1}^n\sum_{t=1}^{t_i}\xi_{i,t}(z,u)}$, $\hat{\phi}_z$ = emission probability parameters. 4. Repeat E and M steps.
Inference in HMMs
• Filtering: Estimate latent state $p(z[t]|\mathbf{x}[0:t])$ using observations up to time t (Forward Algorithm).
• Smoothing: Estimate $p(z[t]|\mathbf{x}[0:T])$, using both past and future observations (Forward-Backward Algorithm).
• Fixed-lag smoothing: Estimate $p(z[t-l]|\mathbf{x}[0:t])$ for online inference (Forward-Backward Algorithm).
• Prediction: Estimate $p(z[t+h]|\mathbf{x}[0:t])$, where $h > 0$ is the prediction horizon: $p(z[t+h]|\mathbf{x}[0:t]) = \sum_{z[t],\ldots,z[t+h-1]} p(z[t+h]|z[t+h-1])p(z[t+h-1]|z[t+h-2]) \cdots p(z[t+1]|z[t]) \cdot p(z[t]|\mathbf{x}[0:t])$.
• MAP sequence: using Viterbi Algorithm, most probable sequence $\mathbf{z}^*[0:T] = \arg\max_{z[0:T]} p(z[0:T]|\mathbf{x}[0:T])$.
Sampling Using Cdf
If $X \sim F$, then: $\mathbb{P}(X \leq x) = F(x)$, Let $U \sim \mathrm{Unif}(0,1)$, define: $X = F^{-1}(U) \Rightarrow X \sim F$.
$\mathbb{P}(F^{-1}(U) \leq x) = \mathbb{P}(U \leq F(x)) = F(x)$.
Transformations
If $Y = f(X)$, then $p_Y(y) = \sum_{k=1}^K \frac{p_X(x_k)}{|f'(x_k)|}$, where x_k are the solutions to $f(x) = y$ **Note:** Requires solving $f(x) = y$ and knowing $f'(x)$
Rejection Sampling
• $p(z) = \frac{1}{M}\tilde{p}(z)$, where M unknown. Choose proposal $q(z)$, and constant $k \geq \frac{\tilde{p}(z)}{q(z)}$ for all z.
• $\mathrm{supp}(p) \subseteq \mathrm{supp}(q)$, where $\mathrm{supp}\,p = \overline{\{z : p(z) > 0\}}$.
• **Sampling Procedure:** 1. Sample $z \sim q(z)$ 2. Sample $u \sim \mathrm{Unif}[0, kq(z)]$ 3. Accept z if $u \leq \tilde{p}(z)$.
• $\mathbb{P}(z \text{ accepted}) = \int \frac{\tilde{p}(z)}{kq(z)}q(z)\,dz = \frac{M}{k}$.
• Choose smallest possible k s.t. $kq(z) \geq \tilde{p}(z)\ \forall z$.
• Accepted $z \sim p(z)$
$\mathbb{P}(z \leq z_0 \mid \text{accepted}) = \frac{1}{M}\int_{z\leq z_0}\tilde{p}(z)\,dz \rightarrow$ Accepted samples follow the CDF of $p(z)$
Importance Sampling
• Estimate expectation $\mathbb{E}_p[f(z)] = \int f(z)p(z)\,dz$, where $p(z)$ is hard to sample.
• Use proposal $q(z)$ and importance weights $w(z) = \frac{p(z)}{q(z)}$, rewrite as $\mathbb{E}_p[f(z)] = \int f(z)w(z)q(z)\,dz \approx \frac{1}{n}\sum_{i=1}^n w(z_i)f(z_i)$, where $z_i \sim q(z)$.
• $\mathrm{supp}(f(\cdot)p(\cdot)) \subseteq \mathrm{supp}(q)$. Keep all samples, no need $q(z) \geq p(z)$. Better matches well, if $q(z)$ is large where $|f(z)|p(z)$ is large.
• If only know how to compute $p(z)$ and $q(z)$ up to normalizing constants, use normalized weights:
$\mathbb{E}_p[f(z)] \approx \sum_{i=1}^n w_n(z_i)f(z_i)$, $w_n(z_i) = \frac{w(z_i)}{\sum_j w(z_j)}$.
Sampling Importance Resampling (SIR)
• Convert importance weighted samples into unweighted samples

from $p(z)$. • **Steps:**
1. Sample $z_1, \dots, z_n$ from $q(z)$.
2. Compute weights $w_n(z_1), \dots, w_n(z_n)$.
3. Resample with replacement from $\{z_1, \dots, z_n\}$ using weights $(w_n(z_1), \dots, w_n(z_n))$.
• Each sample $\tilde{z}_i$ drawn from multinomial over $\{z_1, \dots, z_n\}$ with weights $w_n(z_i)$. asymptotically for large $n \to \infty$, $\mathbb{P}(\tilde{z} \le a) = \sum_{i=1}^n w_n(z_i) \mathbb{1}_{\{z_i \le a\}} \to \int_{z \le a} p(z)\, dz$.

SIR for Bayesian Inference
• Take unnormalized $\tilde{p}(\theta) = p(\mathcal{D} \mid \theta) p(\theta)$, sample $\theta_1, \dots, \theta_n$ from $q(\theta) = p(\theta)$.
• $w_n(\theta_i) = \frac{\tilde{p}(\theta_i)/q(\theta_i)}{\sum_j \tilde{p}(\theta_j)/q(\theta_j)} = \frac{p(\mathcal{D}|\theta_i)}{\sum_j p(\mathcal{D}|\theta_j)}$.
• Resampling $\theta_1, \dots, \theta_n$ according to weights $(w_n(\theta_1), \dots, w_n(\theta_n))$.

Sampling for EM
• Observed incomplete data $\mathbf{x}$, while complete is $(\mathbf{x}, \mathbf{z})$. Need to compute: $Q(\theta \mid \theta^{(m)}) = \int p(\mathbf{z} \mid \mathbf{x}, \theta^{(m)}) \log p(\mathbf{x}, \mathbf{z} \mid \theta)\, dz$.
• $Q(\theta \mid \theta^{(m)}) \approx \frac{1}{n} \sum_{i=1}^n \log p(\mathbf{x}, \mathbf{z}_i \mid \theta)$, where $\mathbf{z}_i \sim p(\mathbf{z} \mid \mathbf{x}, \theta^{(m)})$.
• Rejection and importance sampling not suitable for high-dimensional $\mathbf{z} \to$ need MCMC methods

Stationary Distribution
Consider homogeneous Markov chain with transition probability $p(x_t = y \mid x_{t-1} = x) = \mathbf{T}(x, y)$. π is a stationary distribution if $\sum_x \pi(x)\mathbf{T}(x, y) = \pi(y)$ for all states y. Also called invariant distribution — does not change over time in the chain. If M states, $\mathbf{T}$ is an $M \times M$ matrix with $\pi\mathbf{T} = \pi$.

Asymptotic Steady State
Initial distribution: π_0. Markov evolution: $\pi_1 = \pi_0 \mathbf{T}$, $\pi_2 = \pi_1 \mathbf{T}$, $\dots$, $\pi_k = \pi_0 \mathbf{T}^k$. If the limit $\pi = \lim_{k\to\infty} \pi_k = \lim_{k\to\infty} \pi_0 \mathbf{T}^k$ exist,π must satisfy $\pi\mathbf{T} = \pi \to$ a stationary distribution

Reversible Markov Chain (MC)
• Sufficient condition but not necessary condition for π to be stationary: $\pi(x)\mathbf{T}(x, y) = \pi(y)\mathbf{T}(y, x)$ for all $x, y \to$ This chain is *reversible*. Summing both sides over x: $\sum_x \pi(x)\mathbf{T}(x, y) = \sum_x \pi(y)\mathbf{T}(y, x) = \pi(y) \sum_x \mathbf{T}(y, x) = \pi(y)$.
• Sampling from a distribution $\pi(x)$: Design transition $\mathbf{T}(x, y)$ such that (1) All the ergodicity conditions hold (2) Usually make $\mathbf{T}$ reversible and aperiodic (3) $\pi(x)$ is stationary distribution.
• MCMC basis: Generate sample path for the Markov chain $z_0, z_1, \dots, z_n$ starting from any z_0. If n large, then $p(z_n) \approx \pi(z_n) \to$ obtain samples from π

Metropolis-Hastings Algorithm
• Sample from $\pi(\mathbf{x})$ (e.g., $\mathcal{X} = \mathbb{R}^{1000}$). Assume we can compute unnormalized density $\tilde{\pi}(\mathbf{x})$. Choose transition probability $q(\mathbf{x}, \mathbf{y})$ that is irreducible, aperiodic, and easy to sample — this is the *proposal distribution*.
• Let $Z_0, Z_1, \dots$ be the chain states. At step m:
1. Let $\mathbf{x} = Z_{m-1}$. 2. Sample $\mathbf{y} \sim q(\mathbf{x}, \cdot)$.
3. Accept $\mathbf{y}$ with probability $A(\mathbf{x}, \mathbf{y}) = \min\left(1, \frac{\tilde{\pi}(\mathbf{y})q(\mathbf{y},\mathbf{x})}{\tilde{\pi}(\mathbf{x})q(\mathbf{x},\mathbf{y})}\right)$.
4. If accepted, $Z_m = \mathbf{y}$; else $Z_m = Z_{m-1}$.
• (1) Sequence $Z_0, Z_1, \dots$ is a Markov chain (only depends on Z_{m-1}) (2) Like rejection/importance sampling, does not need normalization constant of $\tilde{\pi}$ (3) Suitable for high-dimensional $\pi(\mathbf{x})$ since sampling is from simple $q(\mathbf{x}, \cdot)$

Proposal Distributions
• MH chain $Z_0, Z_1, \dots$ is designed to converge to stationary distribution $\pi(\cdot)$. For large m, $Z_m \sim \pi$ approximately.
• Burn-in period: discard first 1000 to 5000 samples
• Choice of Proposal Distribution:
1. $q(\mathbf{x}, \mathbf{y}) = q(\mathbf{y} - \mathbf{x})$ — *random walk MH*.
(1) $\mathbf{y} - \mathbf{x} \sim \mathcal{N}(0, \Sigma)$ — Gaussian centered at $\mathbf{x}$.
(2) $\mathbf{y} - \mathbf{x} \sim \text{Unif}[-\delta, \delta]^d$ — Uniform around $\mathbf{x}$.
(3) If $q(\mathbf{x}, \mathbf{y}) = q(\mathbf{y}, \mathbf{x})$, then $A(\mathbf{x}, \mathbf{y}) = \min\left(1, \frac{\tilde{\pi}(\mathbf{y})}{\tilde{\pi}(\mathbf{x})}\right)$ known as the Metropolis Algorithm.
(4) Variance affects — small $\Rightarrow$ slow, high $\Rightarrow$ high rejection rates.
2. Independence Chain MH: $q(\mathbf{x}, \mathbf{y}) = q(\mathbf{y})$, i.e. next state is independent of current. Works well if $q(\mathbf{y})$ closely approximates $\pi(\mathbf{y})$ and is heavy-tailed
3. Exploiting Structure of π: Suppose $\pi(\mathbf{x}) \propto \psi(\mathbf{x})h(\mathbf{x})$, with $h(\mathbf{x})$ is a density that can be sampled from, bounded function $\psi(\mathbf{x})$. Choose $q(\mathbf{x}, \mathbf{y}) = h(\mathbf{y})$, then:
$A(\mathbf{x}, \mathbf{y}) = \min\left(1, \frac{\tilde{\pi}(\mathbf{y})q(\mathbf{y},\mathbf{x})}{\tilde{\pi}(\mathbf{x})q(\mathbf{x},\mathbf{y})}\right) = \min\left(1, \frac{\psi(\mathbf{y})h(\mathbf{y})h(\mathbf{x})}{\psi(\mathbf{x})h(\mathbf{x})h(\mathbf{y})}\right) = \min\left(1, \frac{\psi(\mathbf{y})}{\psi(\mathbf{x})}\right)$

Proposal Variance • Important to tune proposal variance σ.
• σ too small $\to$ slow convergence, high acceptance rate, takes a long time to explore the whole space.
• σ too large $\to$ big steps, low acceptance rate, stuck for long time.
• Rules of thumb: (1) Random walk MH: target acceptance rate of 0.25 to 0.5. (2) Independence chain MH: acceptance rate close to 1

Burn-In Discard early samples before chain reaches stationary distribution. Hard to know when. E.g. $x_0 \sim \text{Unif}(\{0, 1, \dots, 20\})$, takes over 400 steps to "forget" initial state.

Thinning Break dependencies between samples by taking every d-th sample. Useful when σ is too large $\to$ MC stuck for long time at same location. Subsample every d samples: $z_0, z_d, z_{2d}, \dots$

Gibbs Sampling
Special case of Metropolis-Hastings, for multivariate $p(z_1, \dots, z_d)$ (hard to sample jointly when d is large). For each i, define $\mathbf{z}_{-i} = \{z_1, \dots, z_{i-1}, z_{i+1}, \dots, z_d\}$. If we can compute full conditionals $p(z_i \mid \mathbf{z}_{-i})$, we can perform Gibbs sampling:
• Initialize $(z_1^{(0)}, \dots, z_d^{(0)})$.
• For each iteration k, sequentially sample:
$z_1^{(k)} \sim p(\cdot \mid z_2^{(k-1)}, \dots, z_d^{(k-1)})$
$z_2^{(k)} \sim p(\cdot \mid z_1^{(k)}, z_3^{(k-1)}, \dots)$ ...
$z_j^{(k)} \sim p(\cdot \mid z_1^{(k)}, \dots, z_{j-1}^{(k)}, z_{j+1}^{(k-1)}, \dots)$ ...
$z_d^{(k)} \sim p(\cdot \mid z_1^{(k)}, \dots, z_{d-1}^{(k)})$.
• Discard burn-in samples.
• Generating Approximate i.i.d. Samples: (1) Run r independent Gibbs chains of length m, use final sample from each sequence. (2) Run one long sequence, discard burn-in, take every d-th sample.
• Getting conditionals distributions:
$p(z_1 \mid z_2, \dots, z_d) = \frac{p(z_1, \dots, z_d)}{p(z_2, \dots, z_d)}$ (1) Start with joint $p(z_1, \dots, z_d)$. (2) Drop constants not depend on z_1. (3) Use knowledge of well-known distributions to find the distribution $p(z_1 \mid z_2, \dots, z_d)$.

Introduction to NN and DL • An artificial neuron computes $y = f\left(\sum_{j=0}^m w_j x_j\right)$ with $x_0 = 1, w_0 = b$. Common activations: sigmoid $f(x) = \frac{1}{1+e^{-x}}$, ReLU $f(x) = \max(0, x)$.
• Binary output: 1 if $\mathbf{w}^T\mathbf{p} + b > 0$, else 0. Decision boundary: $\mathbf{w}^T\mathbf{p} + b = 0$. +1 from negative to positive.
• OR Gate Perceptron Design: Given $\mathbf{w} = [1, 1]^T$, point $\mathbf{p} = [0, 0.5]^T$ lies on the boundary. Then $1 \cdot 0 + 1 \cdot 0.5 + b = 0 \Rightarrow b = -0.5$.
• AND Gate Perceptron Design: Given $\mathbf{w} = [1, 1]^T$, point $\mathbf{p} = [0, 1.5]^T$ lies on the boundary. Then $1 \cdot 0 + 1 \cdot 1.5 + b = 0 \Rightarrow b = -1.5$.
• XOR and Multi-Layer Perceptron: Single-layer fails on XOR. Use hidden layer: Neuron 1: $\mathbf{w}_1 = [1, 1]^T, b = -0.5$; Neuron 2: $\mathbf{w}_2 = [-1, -1]^T, b = 1.5$. Output neuron AND: $\mathbf{w}_{\text{out}} = [1, 1], b = -1.5$.
• Forward Propagation: Given $W_1, W_2, W_3, b_1, b_2, b_3$, input $\mathbf{p}$, compute: $\mathbf{h}_1 = \sigma(W_1\mathbf{p} + b_1)$, $\mathbf{h}_2 = \sigma(W_2\mathbf{h}_1 + b_2)$, $\hat{y} = \sigma(W_3\mathbf{h}_2 + b_3)$.
• Universal Approximation Theorem: (1) Approximating Arbitrary Decision Regions with 3-Layer MLP; Approximating a Function with 2-Layer MLP. (2) A hidden layer of nonconstant, bounded, and continuous neurons (with weights and biases), plus a linear output neuron (with weights but without bias) can approximate any continuous function to any arbitrary accuracy if there are enough hidden neurons.

Feedforward Computation
A multilayer NN maps input x to output y via: $y = f(x) = \sigma(W_L\sigma(W_{L-1} \dots \sigma(W_1x + b_1) \dots + b_{L-1}) + b_L)$. All layers use weight matrices W_i and bias vectors b_i.

NN terminology Basics
• Classification: last layer = softmax, loss = cross-entropy.
• Regression: last layer = linear, loss = MSE.
• "2-layer NN(1-hidden-layer NN)" = input + 1 hidden + output.
• "3-layer NN(2-hidden-layer NN)" = input + 2 hidden + output.

Activation Functions
• Sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$, output in $[0, 1]$, saturates.
• Tanh: $\tanh(x) = 2\sigma(2x) - 1$, output in $[-1, 1]$, zero-centered.
• ReLU: $\max(0, x)$, avoids vanishing gradient, fast to compute.
• Leaky ReLU: $f(x) = \alpha x$ if $x < 0$, else x, avoids dead neurons.
• Linear: $f(x) = cx$, used in regression output layer. Also known as identity activation function.
• Softmax: $\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$, output is probability distribution. • Maxout. • Exponential Linear Unit (ELU).

Data Preprocessing
• Standardization: zero mean, unit variance. $x' = \frac{x-\mu}{\sigma}$
• Normalization: scale to $[0, 1]$ or $[-1, 1]$: $x' = \frac{x - \min(x)}{\max(x) - \min(x)}$

Loss Functions
• Classification: Cross-entropy loss: $\mathcal{L}(\theta) = -\sum_{k=1}^K y_k \log \hat{y}_k$.
• Regression:
Mean squared error (MSE): $\mathcal{L}(\theta) = \frac{1}{n}\sum_i (y_i - \hat{y}_i)^2$.
Mean absolute error (MAE): $\frac{1}{n}\sum_i |y_i - \hat{y}_i|$

Gradient Descent (GD)
1. **Initialization:** Randomly initialize the model parameters θ^0. Set $\theta^{\text{old}} = \theta^0$.
2. Compute the gradient of the loss function at θ^{old}: $\nabla\mathcal{L}(\theta^{\text{old}})$.
3. Update the parameters: $\theta^{\text{new}} = \theta^{\text{old}} - \alpha\nabla\mathcal{L}(\theta^{\text{old}})$ where α is the learning rate.
4. Set $\theta^{\text{old}} = \theta^{\text{new}}$ and return to step 2 (repeat until terminating).
GD does not guarantee reaching a global minimum.
• Backpropagation: Combines forward pass + backward pass using chain rule to compute gradients of loss w.r.t. weights.
• Batch GD: compute gradients over full dataset.
• Mini-batch GD: over subset (e.g., 32 to 256 samples). Mini-batch gradient approximates the full training set gradient well.
• SGD: mini-batch size = 1, fast but noisy. Less used.

GD with Momentum:
• Momentum: $v_t = \beta v_{t-1} + (1 - \beta)\nabla\mathcal{L}$, improves convergence.
• Nesterov Momentum: lookahead at next step before computing gradient. (Skip)
• Adam: computes a weighted average of past gradients and weighted average of past squared gradients. • Other: RMSprop, Adagrad, Adadelta, Nadam, etc. • Most used: Adam, SGD with momentum.

Learning Rate (LR)
• Too small $\to$ slow convergence; Too large $\to$ divergence or oscillations. • LR scheduling: reduce α over time. Approaches: step decay, exponential/cosine decay, reduce by constant, warmup(increase then cool down).

Underfitting and Overfitting
• Underfitting: high train + validation error (too simple).
• Overfitting: low train error, high validation error (too complex).

Regularization Techniques
• ℓ_2 regularization: $\mathcal{L}_{\text{reg}} = \mathcal{L}(\theta) + \lambda\sum_k \theta_k^2$, penalty for large.
• ℓ_1 regularization: $\sum_k |\theta_k|$, perform worse than ℓ_2.
• Elastic net: mix of ℓ_1 and ℓ_2. • Dropout: randomly deactivate neurons during training (e.g., $p = 0.5$). • Early stopping: monitor validation loss, stop if no improvement after n epochs.

Batch Normalization • Normalize each mini-batch: $\hat{x} = \frac{x-\mu}{\sigma}$
• Less dependent on initialization. Normalizes activations. Reduces vanishing/exploding gradients. Reduces internal covariate shift. Stabilizes training, enables higher learning rates. Common in CNNs.

Hyperparameter Tuning • layer size, lr(schedule), optimizer, regularization, batch size, activation, loss f.
• Methods: grid search(check all with step), random search, Bayesian optimization. Use k-fold cross-validation if data is limited.

Why MLP Fails on Images MLPs are not translation invariant: shifting the input changes the output drastically. CNNs solve this with local receptive fields and shared weights.

Convolution Operation • 1D: Convolution flips and slides a kernel w over input x, $(x * w)[n] = \sum_{k=-\infty}^{\infty} x[k]w[n - k]$
$x = [1, 2, 3, 4]$, $w = [5, 6, 7] \to z = [5, 16, 34, 52, 45, 28]$
• 2D: Apply 2D filter over 2D image by elementwise multiplication and sum. Edge detectors highlight directional features.

Convolutional Neural Networks(CNNs) • Parameter sharing via convolution filters. • Sparse connections via local receptive fields. • Translation invariance. • Fewer parameters than MLPs: faster train.

Residual Networks (ResNets) • Use identity skip connections: output = layer(x) + x. • Help prevent vanishing gradients.
• Enable very deep models (e.g., 18, 50, 152 layers).

CNN Details • Each filter spans full input depth. • Produces one activation map per filter. • Multiple filters produce stacked output (e.g., 6 filters $\to$ output depth = 6).

CNN Architecture Typical: [(CONV - ReLU)$\times N$ - POOL]$\times M$ - [FC - ReLU]$\times K$ - Softmax. Modern trends: deep models (VGG, ResNet), small filters (3×3), less pooling.

Spatial Dimensions For input of size $W_1 \times H_1 \times C$, filter size F, stride S, padding P, number of filters K:
it will produce an output of $W_2 \times H_2 \times K$
$W_2 = \frac{W_1 - F + 2P}{S} + 1$. Same for H_2.
Number of parameters per filter: $F^2C + 1$ (bias), total: $K(F^2C + 1)$ for K filters.
Example Calculation: Input: $32 \times 32 \times 3$, Filters: 10 of size 5×5, stride 1, padding 2 Output spatial size: $(32 + 2 * 2 - 5)/1 + 1 = 32 \to 32 \times 32 \times 10$ Params: $5 * 5 * 3 + 1 = 76$ per filter $\to$ total 760 parameters.

Padding Strategy Zero-padding preserves spatial size. To preserve size: use padding $P = \frac{F-1}{2}$ if $S = 1$

Pooling Layer • Reduces spatial size, parameters, overfitting.
• Max pooling: keep largest value in window. • Avg pooling: keep mean value. • Output size: $W_2 = \frac{W_1 - F}{S} + 1$, same for H_2.
• 0 learnable parameters.

Flatten and FC Layer • Flatten: convert activation map to 1D before FC. • FC: standard dense layer connects to all inputs.

```
import torch
X = torch.tensor([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=torch.
      float)
Y = torch.tensor([0, 1, 1, 0], dtype=int)
network = torch.nn.Linear(2, 2)
loss_fn = torch.nn.CrossEntropyLoss()
optim   = torch.optim.SGD(network.parameters(), lr=0.1)
epochs  = 50
for epoch in range(epochs):
    for x, y in zip(X, Y):
        optim.zero_grad()
        y_pred = network(x)
        loss = loss_fn(y_pred, y)
        loss.backward()
        optimizer.step()
------------------------------------------------------------
import numpy as np
class NeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size,
      learning_rate, momentum=None):
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.learning_rate = learning_rate
        self.momentum = momentum
        self.weights_input_hidden = np.random.rand(input_size,
      hidden_size)
        self.bias_hidden = np.zeros((1, hidden_size))
        self.weights_hidden_output = np.random.rand(hidden_size
      , output_size)
        self.bias_output = np.zeros((1, output_size))
        if self.momentum:
            self.velocity_input_hidden = np.zeros((input_size,
      hidden_size))
            self.velocity_hidden_output = np.zeros((hidden_size
      , output_size))
    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))
    def sigmoid_derivative(self, x):
        return x * (1 - x)
    def forward_pass(self, X):
        self.hidden_output = self.sigmoid(np.dot(X, self.
      weights_input_hidden) + self.bias_hidden)
        self.output = self.sigmoid(np.dot(self.hidden_output,
      self.weights_hidden_output) + self.bias_output)
        return self.output
    def backward_pass(self, X, y, output):
        error = y - output
        output_delta = error * self.sigmoid_derivative(output)
        error_hidden = output_delta.dot(self.
      weights_hidden_output.T)
        hidden_delta = error_hidden * self.sigmoid_derivative(
      self.hidden_output)
        if self.momentum:
            self.velocity_input_hidden = self.momentum * self.
      velocity_input_hidden + X.T.dot(hidden_delta) * self.
      learning_rate
            self.velocity_hidden_output = self.momentum * self.
      velocity_hidden_output + self.hidden_output.T.dot(
      output_delta) * self.learning_rate
            self.weights_input_hidden += self.
      velocity_input_hidden
            self.weights_hidden_output += self.
      velocity_hidden_output
        else:
            self.weights_hidden_output += self.hidden_output.T.
      dot(output_delta) * self.learning_rate
            self.weights_input_hidden += X.T.dot(hidden_delta)
      * self.learning_rate
        self.bias_output += np.sum(output_delta, axis=0,
      keepdims=True) * self.learning_rate
        self.bias_hidden += np.sum(hidden_delta, axis=0,
      keepdims=True) * self.learning_rate
    def train(self, X, y, epochs):
        for epoch in range(epochs):
            output = self.forward_pass(X)
            self.backward_pass(X, y, output)
            if epoch % 1000 == 0:
                loss = np.mean(np.square(y - output))
                print(f'Epoch {epoch}, Loss: {loss:.4f}')
```