1 Supervised Learning

1.1 Definitions

Given $\mathcal{D} \coloneqq \{(\boldsymbol{x}^{(1)}, y^{(1)}), \ldots, (\boldsymbol{x}^{(N)}, y^{(N)})\}$, $y^{(i)} = f(\boldsymbol{x}^{(i)}) + \epsilon$, find $\hat{f}$ s.t. $\hat{f}(\boldsymbol{x}^{(i)}) \approx f^{(i)}(\boldsymbol{x}^{(i)})$ by $\hat{f} = \arg\min_{h \in \mathcal{H}} \mathcal{L}(h)$ where $\mathcal{H}$ is parameterized by $\boldsymbol{w}$.

Regression losses: $l(y, y') =$

Squared (l_2): $(y - y')^2$ Huber: $\begin{cases} \frac{1}{2}(y-y')^2 & |y-y'| \le \delta \\ \delta(|y-y'| - \frac{1}{2}\delta) & \text{otherwise} \end{cases}$

Absolute (l_1): $|y - y'|$ ϵ-insensitive: $\begin{cases} 0 & |y-y'| \le \epsilon \\ |y-y'| - \epsilon & \text{otherwise} \end{cases}$

Classification losses: $l(y, y') =$

Zero-one: $\begin{cases} 1 & y \neq y' \\ 0 & \text{otherwise} \end{cases}$ Hinge: $\max(0, 1 - yy')$

Expected loss consists of bias (average difference from best predictor), variance (sensitivity to choice of $\mathcal{D}$) and Bayes error (irreducible). **Bayes optimal** predictors achieve Bayes error (impossible in practice).

$$\mathbb{E}_{\mathcal{D}}[\mathbb{E}_y[(h(\boldsymbol{x}; \mathcal{D}) - y)^2 | \boldsymbol{x}]] = \underbrace{(\mathbb{E}_{\mathcal{D}}[h] - h_*)^2}_{\text{bias}} + \underbrace{\text{Var}(h)}_{\text{variance}} + \underbrace{\text{Var}(y)}_{\text{Bayes error}}$$

$$\mathcal{L}_{\text{gen}}(h) = \mathbb{E}_{(\boldsymbol{x},y) \sim \Pr(\boldsymbol{x},y)}[l(h(\boldsymbol{x}), y)] \qquad \mathcal{L}_{\text{emp}}(h) = \frac{1}{N} \sum_{i=1}^{N} l(h(\boldsymbol{x}^{(i)}), y^{(i)})$$

We want to minimize $L_{\text{gen}}(h)$ but we can't because we don't know $\Pr(\boldsymbol{x}, y)$. **Empirical risk minimization** minimizes $L_{\text{emp}}(h)$ instead. $\mathcal{L}_{\text{emp}}(h) \to \mathcal{L}_{\text{gen}}(h)$ as $N \to \infty$ due to WLLN.

ν-fold cross validation: Split $\mathcal{D}$ into ν equal folds, for each fold train on all others and evaluate on current. Increasing ν lowers bias but increases variance of error estimate. $\nu = N$ is **leave-one-out cross validation**.

1.2 k-Nearest Neighbours

$$\hat{f}(\boldsymbol{x}^*) = \frac{1}{k} \sum_{i \in \mathcal{N}_k(\boldsymbol{x}^*)} y^{(i)} \quad \text{or} \quad \frac{1}{\sum_i w_i} \sum_{i \in \mathcal{N}_k(\boldsymbol{x}^*)} w_i y^{(i)}, \quad w_i = \frac{1}{\text{dist}(\boldsymbol{x}^*, \boldsymbol{x}^{(i)})}$$

where $\mathcal{N}_k(\boldsymbol{x}^*)$ are k training samples closest to $\boldsymbol{x}^*$. Use most common label for classification, decrease k if tied or use odd k. Smaller k gives more complex model. Rule of thumb: $k < \sqrt{N}$. **Normalize** $x_i \leftarrow (x_i - \mu_i)/\sigma_i$ per-dimension if scale doesn't matter. Distance metrics: $\text{dist}(\boldsymbol{x}, \boldsymbol{z}) =$

Minkowski: $\left(\sum_{i=1}^{D} |x_i - z_i|^p\right)^{\frac{1}{p}}$ Mahalanobis: $\sqrt{(\boldsymbol{x} - \boldsymbol{z})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{z})}$

Complexity: $\mathcal{O}(ND + N \log N)$ or $\mathcal{O}(D \log N)$ (k-d tree, $D \ll N$).

Curse of dimensionality: Number of points needed to keep neighbour distance constant grows exponentially with D. (Use dimensionality reduction.) Result can be made **probabilistic** by considering distribution of labels. Apply Laplace smoothing to avoid zero probabilities.

1.3 Linear Regression

$$\hat{f}(\boldsymbol{x}, \boldsymbol{w}) = w_0 + \sum_{j=1}^{D} w_j x_d = \boldsymbol{w}^T \boldsymbol{x}$$

where $x_0 = 1$, $\boldsymbol{x} = \{x_0, \ldots, x_D\}^T$, $\boldsymbol{X}_{ij} = x_j^{(i)}$, $\boldsymbol{y} = \{y^{(1)}, \ldots, y^{(N)}\}^T$ then

$$\hat{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{w} \qquad \hat{\boldsymbol{w}} = \underset{\boldsymbol{w} \in \mathbb{R}^{D+1}}{\arg\min} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2 \implies \boldsymbol{X}^T\boldsymbol{X}\hat{\boldsymbol{w}} = \boldsymbol{X}^T\boldsymbol{y}$$

Requires $\mathcal{O}(ND)$+$\mathcal{O}(N)$ memory to solve (additional $\mathcal{O}(ND)$ for some methods). Compute $\boldsymbol{X}^T\boldsymbol{X}$ and $\boldsymbol{X}^T\boldsymbol{y}$ term-by-term for large problems.

Overdetermined problems: $N > D + 1$, $\boldsymbol{X}$ is tall. Solve for $\boldsymbol{w}$ by:

Cholesky: $\boldsymbol{X}^T\boldsymbol{X} = \boldsymbol{R}^T\boldsymbol{R}$ where $\boldsymbol{R} \in \mathbb{R}^{(D+1)\times(D+1)}$ upper-triangular. Then $\boldsymbol{z} = \boldsymbol{R}^{-T}\boldsymbol{X}^T\boldsymbol{y}$ and $\boldsymbol{w} = \boldsymbol{R}^{-1}\boldsymbol{z}$ by back substitution. Add $\lambda\boldsymbol{1}$ to $\boldsymbol{X}^T\boldsymbol{X}$ to avoid singularity (l_2 regularization). Fastest but squares condition number. Complexity: $\mathcal{O}(N(D+1)^2 + \frac{1}{3}(D+1)^3)$

Economic QR: $\boldsymbol{X} = \boldsymbol{Q}\boldsymbol{R}$ where $\boldsymbol{Q} \in \mathbb{R}^{N\times(D+1)}$ is orthonormal, $\boldsymbol{R}$ as above. Then $\boldsymbol{w} = \boldsymbol{R}^{-1}\boldsymbol{Q}^T\boldsymbol{y}$. Can fail when $\boldsymbol{X}$ is nearly rank-deficient. Complexity: $\mathcal{O}(2N(D+1)^2 + \frac{2}{3}(D+1)^3)$

SVD: $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T = \boldsymbol{U}_1\boldsymbol{\Sigma}_1\boldsymbol{V}^T$ where $\boldsymbol{U} \in \mathbb{R}^{N\times N}, \boldsymbol{V} \in \mathbb{R}^{(D+1)\times(D+1)}$ are orthogonal and $\boldsymbol{\Sigma} \in \mathbb{R}^{N\times(D+1)}$ is diagonal. Then $\boldsymbol{w} = \boldsymbol{V}\boldsymbol{\Sigma}_1^{-1}\boldsymbol{U}_1^T\boldsymbol{y} = \sum_{i=1}^{D+1} \boldsymbol{v}_i \frac{\boldsymbol{u}_i^T\boldsymbol{y}}{\sigma_i}$ where $\boldsymbol{u}_i, \boldsymbol{v}_i$ denotes column i. Truncate for rank-deficient $\boldsymbol{X}$. Slowest and most stable. Complexity: $\mathcal{O}(2N(D+1)^2 + 11N(D+1)^3)$

Pseudoinverse: $\boldsymbol{X}^\dagger = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T = \boldsymbol{V}\boldsymbol{\Sigma}^\dagger\boldsymbol{U}^T$ then $\boldsymbol{w} = \boldsymbol{X}^\dagger\boldsymbol{y}$.

Condition number: $\kappa(\boldsymbol{X}) = \|\boldsymbol{X}\|\|\boldsymbol{X}^\dagger\| = \frac{\sigma_{\max}}{\sigma_{\min}}$. Computing $\boldsymbol{X}^T\boldsymbol{X}$ squares it. Adding $\lambda\boldsymbol{1}$ lowers it. 1 digit of precision is lost per power of 10 in κ. Cholesky squares κ.

Underdetermined problems: $N < D + 1$, $\boldsymbol{X}$ is wide. Instead solve $\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w} \in \mathbb{R}^{D+1}} \|\boldsymbol{w}\|_2^2$ subject to $\boldsymbol{X}\boldsymbol{w} = \boldsymbol{y} \implies \boldsymbol{w} = \boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{X}^T)^{-1}\boldsymbol{y}$.

For QR, $\boldsymbol{X}^T = \boldsymbol{Q}\boldsymbol{R}$ and $\boldsymbol{w} = \boldsymbol{Q}\boldsymbol{R}^{-T}\boldsymbol{y}$; for SVD, $\boldsymbol{X}^T = \boldsymbol{U}_1\boldsymbol{\Sigma}_1\boldsymbol{V}^T$ and $\boldsymbol{w} = \boldsymbol{U}_1\boldsymbol{\Sigma}_1^{-1}\boldsymbol{V}^T\boldsymbol{y}$.

For classification, use model $\hat{y} = \sigma(\boldsymbol{w}^T\boldsymbol{x})$ where $\sigma(z) = \frac{1}{1+e^{-z}}$. Use cross-entropy instead of squared loss (logistic regression).

$$\mathcal{L}_{CE}(\boldsymbol{w}) = \frac{1}{N} \sum_{i=1}^{N} \left[-y^{(i)} \log \hat{y}^{(i)} - (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})\right]$$

No closed-form solution exists. Gradient is $\sum_{i=1}^{n}(y^{(i)} - \sigma(\boldsymbol{w}^T\boldsymbol{x}^{(i)}))\boldsymbol{x}^{(i)}$.

1.4 Generalized Linear Models (GLMs)

$$\hat{f}(\boldsymbol{x}, \boldsymbol{w}) = w_0 + \sum_{i=1}^{M-1} w_i \phi_i(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x})$$

where $\phi_0(\boldsymbol{x}) = 1$, $\boldsymbol{\phi}(\boldsymbol{x}) = \{\phi_0(\boldsymbol{x}), \ldots, \phi_{M-1}(\boldsymbol{x})\}^T$, $\boldsymbol{\Phi}_{ij} = \phi_j(x^{(i)})$ then

$$\hat{\boldsymbol{y}} = \boldsymbol{\Phi}\boldsymbol{w} \qquad \hat{\boldsymbol{w}} = \underset{\boldsymbol{w} \in \mathbb{R}^{D+1}}{\arg\min} \|\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{w}\|_2^2 \implies \boldsymbol{\Phi}^T\boldsymbol{\Phi}\hat{\boldsymbol{w}} = \boldsymbol{\Phi}^T\boldsymbol{y}$$

Large M leads to overfitting. Can be counteracted by **regularization**:

$$\boldsymbol{w} = \underset{\boldsymbol{w} \in \mathbb{R}^M}{\arg\min} |\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{w}|_2^2 + \lambda\|\boldsymbol{w}\|_2^2 \implies (\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda\boldsymbol{1})\boldsymbol{w} = \boldsymbol{\Phi}^T\boldsymbol{y}$$

SVD: $\boldsymbol{\Phi} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T \implies \boldsymbol{w}(\lambda) = \boldsymbol{V}(\boldsymbol{\Sigma}^T\boldsymbol{\Sigma} + \lambda\boldsymbol{1})^{-1}\boldsymbol{\Sigma}^T\boldsymbol{U}^T\boldsymbol{y} = \sum_{i=1}^{M} \boldsymbol{v}_i \frac{\sigma_i \boldsymbol{u}_i^T\boldsymbol{y}}{\sigma_i^2 + \lambda}$

1.5 Dual Representation of GLMs

$$\hat{f}(\boldsymbol{x}, \boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i k(\boldsymbol{x}, \boldsymbol{x}^{(i)}) = \boldsymbol{k}^T(\boldsymbol{x})\boldsymbol{\alpha}, \quad \boldsymbol{\alpha} = (\boldsymbol{K} + \lambda\boldsymbol{1})^{-1}\boldsymbol{y}$$

where $\boldsymbol{k}(\boldsymbol{x}) = \{k(\boldsymbol{x}^{(1)}, \boldsymbol{x}), \ldots, k(\boldsymbol{x}^{(N)}, \boldsymbol{x})\}^T$, $K_{ij} = k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$. Relation to primal: $k(\boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{\phi}^T(\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{z})$, $\boldsymbol{w} = \boldsymbol{\Phi}^T\boldsymbol{\alpha}$, $\alpha_i = -\frac{1}{\lambda}(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}^{(i)}) - y^{(i)})$. Setting $\lambda = 0$ leads to an interpolation model, i.e. it reproduces training targets given training inputs.

Kernels define an inner product (measure of distance) over Hibert space $\mathcal{F}$:

$$k(\boldsymbol{x}, \boldsymbol{z}) = k(\boldsymbol{z}, \boldsymbol{x}) \qquad \iint u(\boldsymbol{x}) k(\boldsymbol{x}, \boldsymbol{z}) u(\boldsymbol{z}) \, \mathrm{d}\boldsymbol{x} \, \mathrm{d}\boldsymbol{z} \ge 0$$

$$k(\boldsymbol{x}, \boldsymbol{x}) \ge 0 \qquad k(\boldsymbol{z}, \boldsymbol{z}) \le \sqrt{k(\boldsymbol{x}, \boldsymbol{x}) k(\boldsymbol{z}, \boldsymbol{z})}$$

Example kernels: $r = \|\boldsymbol{x} - \boldsymbol{z}\|$, $k(\boldsymbol{x}, \boldsymbol{z}) =$

Linear: $\boldsymbol{x}^T\boldsymbol{z}$ Polynomial: $(1 + \boldsymbol{x}^T\boldsymbol{z})^n$

Iso. Gaussian: $e^{-\frac{1}{2}r^2}$ Aniso. Gaussian: $e^{-(\boldsymbol{x}-\boldsymbol{z})^T\boldsymbol{\Theta}^{-1}(\boldsymbol{x}-\boldsymbol{z})}$

Multiquadratic: $\sqrt{1 + \frac{r^2}{\theta}}$ Inv. MQ: $\frac{1}{\sqrt{1 + \frac{r^2}{\theta}}}$

Matern C^0: $\exp\left(-\frac{r}{\theta}\right)$ Matern C^2: $\frac{1}{1 + \frac{r}{\theta}} \exp\left(-\frac{r}{\theta}\right)$

Matern C^4: $\left(3 + 3\frac{r}{\theta} + \left(\frac{r}{\theta}\right)^2\right) \exp\left(-\frac{r}{\theta}\right)$

All **radial basis function** (RBF) kernels depend only on r and produce positive definite $\boldsymbol{K}$ (except multiquadrics). $\boldsymbol{K}$ is non-singular if data points are distinct.

Kernel selection is informed by prior knowledge of the target, e.g. degree of smoothness, periodicity, other function properties. Use kernel methods when $M \gg N$ and explicit basis functions when $N \gg M$; if both are high use sparse models (e.g. orthogonal marching pursuit). Time and memory complexity of kernel GLMs scale as $\mathcal{O}(N^3)$.

Any algorithm can be kernelized by mapping from input to feature space via a basis $\boldsymbol{\phi}$: $\mathcal{X} \to \mathcal{F}$, then replacing any inner product $\boldsymbol{\phi}^T(\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{z})$ with the kernel $k(\boldsymbol{x}, \boldsymbol{z})$.

1.6 Sparse GLMs

Orthogonal marching pursuit: Greedy sparse regression. Pick basis functions one at a time using metric $J(\phi_i) = \frac{(\boldsymbol{\Phi}_i^T\boldsymbol{r}^{(k)})^2}{\boldsymbol{\Phi}_i^T\boldsymbol{\Phi}_i}$ where $\boldsymbol{\Phi}_i$ denotes column i. Solve $\boldsymbol{\phi}^{(k)}\boldsymbol{w}^{(k)} \approx \boldsymbol{y}$ for updated weights and compute residual $\boldsymbol{r}^{(k)} = \boldsymbol{y} - \boldsymbol{\Phi}^{(k)}\boldsymbol{w}^{(k)}$.

Leave-one-out-error: Let $\boldsymbol{A} = \boldsymbol{K}(\boldsymbol{K} + \lambda\boldsymbol{1})^{-1} = \boldsymbol{\Phi}(\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T$ then

$$y^{(i)} - \hat{f}^{\setminus i}(\boldsymbol{x}^{(i)}) = \frac{y^{(i)} - \hat{f}(\boldsymbol{x}^{(i)})}{1 - A_{ii}} \implies \text{LOO} = \frac{1}{N} \sum_{i=1}^{N} \left(\frac{y^{(i)} - \hat{f}(\boldsymbol{x}^{(i)})}{1 - A_{ii}}\right)^2$$

for least-squares error with l_2 regularization.

l_1 regularization typically gives sparser models, but no analytic solution.

1.7 Parameter/Density Estimation

Given $\mathcal{D} = \{\boldsymbol{x}^{(i)}\}_{i=1}^N$ find its generating distribution, assuming it is a member of $\mathcal{P}_{\boldsymbol{\theta}} = \{p(\boldsymbol{x} \mid \boldsymbol{\theta}) \mid \boldsymbol{\theta} \in \Gamma\}$ parameterized by $\boldsymbol{\theta}$.

Maximum likelihood: Assuming no prior, i.i.d. data:

$$\hat{\boldsymbol{\theta}}_{\text{ML}} = \underset{\boldsymbol{\theta} \in \Gamma}{\arg\max}\, p(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(N)} | \boldsymbol{\theta}) = \underset{\boldsymbol{\theta} \in \Gamma}{\arg\max} \sum_{i=1}^{N} \log p(\boldsymbol{x}^{(i)} | \boldsymbol{\theta})$$

Assuming additive Gaussian noise on $\hat{f}(\boldsymbol{x}^{(i)}, \boldsymbol{w})$ is equivalent to regression with l_2 loss; Laplacian noise is l_1 loss. However resulting variance is constant, which is not reasonable (should be smaller where there are more data points).

Maximum a posteriori: Assuming prior $p(\boldsymbol{\theta})$, i.i.d. data:

$$\hat{\boldsymbol{\theta}}_{\text{MAP}} = \underset{\boldsymbol{\theta}}{\arg\max}\, p(\mathcal{D}|\boldsymbol{\theta}) p(\boldsymbol{\theta}) = \underset{\boldsymbol{\theta} \in \Gamma}{\arg\max} \log p(\boldsymbol{\theta}) + \sum_{i=1}^{N} \log p(\boldsymbol{x}^{(i)} | \boldsymbol{\theta})$$

Assuming Gaussian prior is equivalent to l_2 regularization (effective $\lambda = \sigma^2/\alpha$ where σ^2 is the variance of data, α the variance of prior). Assuming Laplacian prior leads to l_1 regularization.

2 Unsupervised Learning

Dimensionality reduction: Given a dataset $\mathcal{D} = \{\boldsymbol{x}^{(i)}\}_{i=1}^N$ where $\boldsymbol{x}^{(i)} \in \mathbb{R}^D$, find a mapping $f \colon \mathbb{R}^D \mapsto \mathbb{R}^d$ where $d < D$ so key characteristics of the data are retained. Can be used to save computational effort (as pre-processing), reduce overfitting, and for visualization of high-dimensional data.

2.1 Principal Component Analysis

Linear model $\boldsymbol{z} = \boldsymbol{U}^T(\boldsymbol{x} - \boldsymbol{b})$ where where $\boldsymbol{U} \in \mathbb{R}^{D\times d}$ is an orthonormal matrix and $\boldsymbol{b} \in \mathbb{R}^D$. $\boldsymbol{z}$ is the **representation**/**code**; the projection of $\boldsymbol{x}$ into $\mathcal{S} = \text{col}\,\boldsymbol{U}$ is the **reconstruction** $\tilde{\boldsymbol{x}}$.

$$\boldsymbol{b} = \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{x}^{(i)} \quad \boldsymbol{\Sigma} = \frac{1}{N} \sum_{i=1}^{N} (\boldsymbol{x}^{(i)} - \boldsymbol{b})(\boldsymbol{x}^{(i)} - \boldsymbol{b})^T$$

The k-th principal component (column of $\boldsymbol{U}$) is the eigenvector of $\boldsymbol{\Sigma}$ corresponding to the k-th largest eigenvalue. All principal components are orthogonal. Code vectors produced by PCA are de-correlated (diagonal covariance). Minimizing reconstruction error is equivalent to maximizing variance of code vectors. The first principal component is in the direction of maximum variation, then every other component is normal to all previous ones and accounts for as much of the remaining variation as possible.

If $\boldsymbol{X}$ is centered (subtract mean from each row), take SVD $\boldsymbol{X} = \boldsymbol{U}_1\boldsymbol{S}_1\boldsymbol{V}_1^T$, then columns of $\boldsymbol{V}$ contain principal components and $\boldsymbol{S}_1^2/N$ are eigenvalues of $\boldsymbol{\Sigma}$.

3 Optimization

Given $f(\boldsymbol{\theta})$ where $\boldsymbol{\theta} \in \mathbb{R}^n$ and $f \colon \mathbb{R}^n \mapsto \mathbb{R}$ find $\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} f(\boldsymbol{\theta})$. Define:

$$\nabla f(\boldsymbol{\theta}) = \boldsymbol{g}(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial f}{\partial \theta_1} \\ \vdots \\ \frac{\partial f}{\partial \theta_n} \end{bmatrix} \qquad \nabla^2 f(\boldsymbol{\theta}) = \boldsymbol{H}(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial^2 f}{\partial {\theta_1}^2} & \cdots & \frac{\partial^2 f}{\partial \theta_n \partial \theta_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial \theta_1 \partial \theta_n} & \cdots & \frac{\mathrm{d}^2 f}{\mathrm{d} {\theta_n}^2} \end{bmatrix}$$

Karush-Kuhn-Tucker (KKT) conditions:

- **First-order condition:** If $\boldsymbol{\theta}^*$ is a minimum, then $\nabla f(\boldsymbol{\theta}^*) = \boldsymbol{0}$ if it exists.
- **Second-order condition:** If $\boldsymbol{\theta}^*$ is a minimum, then $H(\boldsymbol{\theta}^*)$ is positive semi-definite, if it exists.
- **Sufficient conditions:** If $\nabla f(\boldsymbol{\theta}^*) = \boldsymbol{0}$ and $H(\boldsymbol{\theta}^*)$ is positive definite and both exist, then $\boldsymbol{\theta}^*$ is a local minimum.

3.1 Unconstrained Optimization

Start with $k = 0$ and an initial guess $\boldsymbol{\theta}_0$, repeat:

1. Test for convergence of $\boldsymbol{\theta}_k$.
2. Compute new search direction $\boldsymbol{p}_k$ s.t. $\boldsymbol{p}_k^T\boldsymbol{g}_k(\boldsymbol{\theta}_k) < 0$.
3. Compute step length $\alpha_k > 0$ s.t. $f(\boldsymbol{\theta}_k + \alpha_k\boldsymbol{p}_k) < f(\boldsymbol{\theta}_k)$.
4. Update $\theta_{k=1} \leftarrow \boldsymbol{\theta}_k + \alpha_k\boldsymbol{p}_k$, $k \leftarrow k + 1$.

Stopping criteria: gradient tolerance ϵ_g, absolute tolerance ϵ_a, relative tolerance ϵ_r. Stop if $\|\boldsymbol{g}(\boldsymbol{\theta}_k)\|_2 \le \epsilon_g$ or $|f(\boldsymbol{\theta}_{k+1}) - f(\boldsymbol{\theta}_k)| \le \epsilon_a + \epsilon_r|f(\boldsymbol{\theta}_k)|$ for two successive iterations.

Search direction: $\boldsymbol{p}_k = -\boldsymbol{B}\boldsymbol{g}_k$ is always valid. Steepest descent: $\boldsymbol{B} = \boldsymbol{1}$; Newton's method: $\boldsymbol{B} = \boldsymbol{H}_k^{-1}$; quasi-Newton: $\boldsymbol{B} \approx \boldsymbol{H}_k^{-1}$.

Armijo sufficient decrease condition:

$$f(\boldsymbol{\theta}_k + \alpha_k\boldsymbol{p}_k) \le f(\boldsymbol{\theta}_k) + \mu_1\alpha_k\boldsymbol{g}_k^T\boldsymbol{p}_k$$

where μ_1 is typically $\sim 10^{-4}$. **Backtracking line search**: start with $\alpha \in (0, 1)$, check Armijo, and if not satisfied, $\alpha \leftarrow \rho\alpha$ and repeat (typically $\rho \in [0.1, 0.5]$).

Steepest descent: Use $\boldsymbol{p}_k = -\frac{\boldsymbol{g}(\boldsymbol{\theta}_k)}{\|\boldsymbol{g}(\boldsymbol{\theta}_k)\|_2}$ at each step. Can show $\boldsymbol{p}_k \perp \boldsymbol{p}_{k-1}$ so it zig zags (inefficient). Linear convergence: $\lim_{k\to\infty} \frac{|f(\boldsymbol{\theta}_{k-1}) - f(\boldsymbol{\theta}^*)|}{|f(\boldsymbol{\theta}_k) - f(\boldsymbol{\theta}^*)|} = K$.

Conjugate gradient methods: Use $\boldsymbol{p}_k = -\boldsymbol{g}_k + \beta_k\boldsymbol{p}_{k-1}$ for $k \ge 1$, where $\beta_k = \frac{\boldsymbol{g}_k^T\boldsymbol{g}_k}{\boldsymbol{g}_{k-1}^T\boldsymbol{g}_{k-1}}$ (Fletcher-Reeves) $\beta_k = \frac{\boldsymbol{g}_k^T(\boldsymbol{g}_k - \boldsymbol{g}_{k-1})}{\boldsymbol{g}_{k-1}^T\boldsymbol{g}_{k-1}}$ (Polak-Ribieve). More efficient due to less zig zag and still only uses first-order information.

Newton's method: Use $\boldsymbol{p}_k = \boldsymbol{H}_k^{-1}\boldsymbol{g}_k$ at each step. Step direction only valid if $\boldsymbol{H}_k$ is SPD. Quadratic convergence: $\lim_{k\to\infty} \frac{|f(\boldsymbol{\theta}_{k-1}) - f(\boldsymbol{\theta}^*)|}{|f(\boldsymbol{\theta}_k) - f(\boldsymbol{\theta}^*)|^2} = K > 0$ if $\boldsymbol{\theta}_0$ close to $\boldsymbol{\theta}^*$ and $\boldsymbol{H}(\boldsymbol{\theta}^*) > 0$. Possible failure and numerical issues if $\boldsymbol{H}_k$ is non-SPD or quadratic approximation is poor.

Quasi-Newton methods: Use approximate $\boldsymbol{B}_k^{-1} \approx \boldsymbol{H}_k^{-1}$ and update approximate Hessian as $\boldsymbol{B}_{k+1}^{-1} = \boldsymbol{B}_k^{-1} + \Delta\hat{\boldsymbol{B}}_k$ at each step. Convergence between linear and quadratic.

SR1 update: Update $\boldsymbol{B}_k$ by a symmetric rank-1 matrix at each step.

$$\boldsymbol{B}_{k+1}^{-1} = \boldsymbol{B}_k^{-1} + \frac{(\boldsymbol{s}_k - \boldsymbol{B}_k^{-1}\boldsymbol{y}_k)(\boldsymbol{s}_k - \boldsymbol{B}_k^{-1}\boldsymbol{y}_k)^T}{(\boldsymbol{s}_k - \boldsymbol{B}_k^{-1}\boldsymbol{y}_k)^T\boldsymbol{y}_k}$$

3.2 Unconstrained Optimization

Minimize $f(\boldsymbol{\theta})$ subject to constraints $g_i(\boldsymbol{\theta}) \ge 0, h_j(\boldsymbol{\theta}) = 0$ where $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, q$ and $\boldsymbol{\theta}_l \le \boldsymbol{\theta} \le \boldsymbol{\theta}_u$.

Penalty methods: Minimize $\pi(\boldsymbol{\theta}, \rho_k) = f(\boldsymbol{\theta}) + \rho_k\phi(\boldsymbol{\theta})$, where $\boldsymbol{\phi}(\boldsymbol{\theta})$ is ero when constraints are met, positive when constraints violated. e.g. quadratic penalty function $\phi(\boldsymbol{\theta}_k) = \sum_{i=1}^{m}(\max(0, -g_i(\boldsymbol{\theta})))^2 + \sum_{i=1}^{q}(h_i(\boldsymbol{\theta}))^2$.

At each step, check convergence conditions, find $\boldsymbol{\theta}_{k+1} = \arg\min_{\boldsymbol{\theta} \in \Gamma} \pi(\boldsymbol{\theta}, \rho_k)$, increment ρ if constraints not met well enough (typical factor of 1.4 to 10). Objective and constraint need appropriate scaling so one doesn't dominate the other.

3.3 Nonlinear Least Squares

Find $\arg\min_{\boldsymbol{\theta} \in \mathbb{R}^n} = \frac{1}{2}\sum_{i=1}^{N} r_i(\boldsymbol{\theta})^2$ where $r_i(\boldsymbol{\theta}) = \hat{f}(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}) - y^{(i)}$. Let $\boldsymbol{r} = \{r_1, \ldots, r_n\}^T \in \mathbb{R}^N$ then minimize $f(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{r}(\boldsymbol{\theta})\|_2^2$.

$$\nabla f(\boldsymbol{\theta}) = \boldsymbol{J}(\boldsymbol{\theta})^T\boldsymbol{r}(\boldsymbol{\theta}) \qquad \nabla^2 f(\boldsymbol{\theta}) = \boldsymbol{J}(\boldsymbol{\theta})^T\boldsymbol{J}(\boldsymbol{\theta}) + \sum_{j=1}^{N} r_j(\boldsymbol{\theta}) \nabla^2 r_j(\boldsymbol{\theta})$$

Gauss-Newton method: Use approximate Hessian $\nabla^2 f(\boldsymbol{\theta}) = \boldsymbol{J}(\boldsymbol{\theta})^T\boldsymbol{J}(\boldsymbol{\theta})$. Solve $\boldsymbol{J}(\boldsymbol{\theta})^T\boldsymbol{J}(\boldsymbol{\theta})\boldsymbol{p}_k = -\boldsymbol{J}(\boldsymbol{\theta})^T\boldsymbol{r}(\boldsymbol{\theta}_k)$ for search direction and line search α_k. Where initial residual is small or linear in $\boldsymbol{\theta}$, this performs similar to full Newton's method, using only first-order derivatives. If $\boldsymbol{J}(\boldsymbol{\theta}_k)$ is full-rank and $\nabla f(\boldsymbol{\theta}_k) \neq 0$, the search direction is always valid.

3.4 Gradient Descent (Stochastic, Full, Mini-Batch)

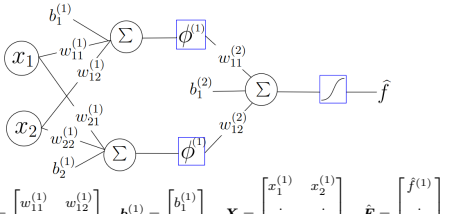
$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta} \in \Gamma}{\arg\min}\, \mathcal{L}(\boldsymbol{\theta}; \mathcal{D}) = \frac{1}{N} \sum_{i=1}^{N} l(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)}) + \lambda R(\boldsymbol{\theta})$$

Full-batch gradient descent: $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta_k\nabla\mathcal{L}(\boldsymbol{\theta}_k; \mathcal{D})$ where η_k is a typically constant learning rate. Expensive since gradient is computed over entire dataset for each step.

Stochastic gradient descent: $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta\nabla l(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)})$ for $i = 1, \ldots, N$, after shuffling. Each full $i = 1 \to N$ is an **epoch**. Cost per step is independent of dataset size. Works because $\mathbb{E}[l(\boldsymbol{\theta}_k; \boldsymbol{x}^{(i)}, y^{(i)})] = \mathcal{L}(\boldsymbol{\theta}_k; \mathcal{D})$. More iterations to converge but each takes less time; runs faster overall.

Mini-batch gradient descent: $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \frac{\eta}{N}\sum_{\mathcal{D}} \nabla l(\boldsymbol{\theta}; \boldsymbol{x}^{(t)}, y^{(t)})$ Compromise between full-batch and SGD. Compute gradient averaged over batches of multiple data points.

4 Neural Networks



$$\boldsymbol{W}^{(1)} = \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} \end{bmatrix} \quad \boldsymbol{b}^{(1)} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \end{bmatrix} \quad \boldsymbol{X} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} \\ \vdots & \vdots \\ x_N^{(1)} & x_N^{(1)} \end{bmatrix} \quad \hat{\boldsymbol{F}} = \begin{bmatrix} \hat{f}^{(1)} \\ \vdots \\ \hat{f}^{(N)} \end{bmatrix}$$

$$\boldsymbol{H}^{(i)} = \phi^{(i)}(\boldsymbol{H}^{(i-1)}{\boldsymbol{W}^{(i)}}^T + \boldsymbol{b}^{(i)}) \qquad \hat{\boldsymbol{F}} = \sigma(\boldsymbol{H}^{(N-1)}{\boldsymbol{W}^{(N)}}^T + \boldsymbol{b}^{(N)})$$

Logistic sigmoid: for binary classification: $\sigma(z) = \frac{1}{1+e^{-z}}$

Softmax: for multi-class classification: $\text{softmax}(\boldsymbol{z}) = \frac{e^{z_j}}{\sum_k e^{z_k}}$

$\text{logsumexp}(\boldsymbol{z}) = \log\left(\sum_i e^{z_i}\right) \implies \log \text{softmax}(\boldsymbol{z})_j = z_j - \text{logsumexp}(\boldsymbol{z})$

Classification dataset likelihoods:

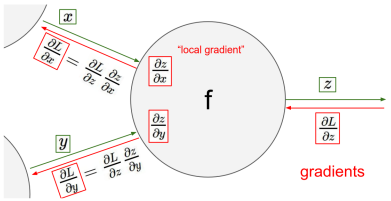
Binary: i.i.d. Bernoulli, $\Pr(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{w}) = \hat{f}(\boldsymbol{x}; \boldsymbol{w})^y (1 - \hat{f}(\boldsymbol{x}; \boldsymbol{w}))^{1-y}$

$$\hat{\boldsymbol{w}} = \arg\max_{\boldsymbol{w}} \sum_{i=1}^{N} y^{(i)} \log f(\boldsymbol{x}^{(i)}; \boldsymbol{w}) + (1 - y^{(i)}) \log(1 - f(\boldsymbol{x}^{(i)}; \boldsymbol{w}))$$

Multiclass: i.i.d. Categorical, $\Pr(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{w}) = \prod_{j=1}^{K} \hat{f}_j(\boldsymbol{x}; \boldsymbol{w})^{y_j}$

$$\hat{\boldsymbol{w}} = \arg\max_{\boldsymbol{w}} \sum_{i=1}^{N} \sum_{j=1}^{K} y_j^{(i)} \log f_j(\boldsymbol{x}^{(i)}; \boldsymbol{w})$$

Backpropagation: Each node $z = f(x_1, \ldots, x_N)$ in the computational graph is passed an upstream gradient $\frac{\partial L}{\partial z}$ (sum of all downstream gradients of the nodes that z connects to) and computes local gradients $\frac{\partial z}{\partial x_i}$; pass gradients downstream as $\frac{\partial L}{\partial x_i} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x_i}$; full gradient is obtained when leaf is reached. $+$: distribute gradients; max/min: route gradient to 1 input; $\times$: swap gradients.



Weight initialization: Aim for zero mean, unit variance of initial hidden states to avoid vanishing or exploding gradients. *Xavier:* sample from unit Gaussian, divide by square root of number of neuron inputs; *ReLU:* divide by square root of half the number of inputs.

4.1 Regularization Techniques

Weight decay: Equivalent to l_2 regularization; keeps weights small.

Early stopping: Stopping when validation loss increases, before convergence in (training) gradient.

Data augmentation: Creating new samples from existing ones through class-preserving transforms, e.g. rotation, blurring, cropping, etc. or **adversarial training** samples designed to mislead the model.

Bagging/bootstrap aggregation: Generate k datasets of equivalent size from $\mathcal{D}$ by sampling with replacement, and train k models on datasets; at inference time, average predictions (regression) or vote on class (classification).

Dropout: During training, drop weights randomly with probability $1 - \pi$ (typ. $\pi \in [0.5, 0.8]$) at each SGD iteration; during inference, scale weights by π. Equivalent to bagging with a k that increases exponentially with network size (but same model across all datasets).

Weight sharing: Using prior knowledge to constrain weights to be similar; CNNs are an example.

4.2 Network Architectures

Convolutional Neural Networks (CNNs): Convolving 2D (or higher) filters with the input to produce output for the next layer, keeping structure. Use on highly structured data, e.g. images. Pooling layers between convolutional layers perform spacial downsampling (but keeps depth). Layers will learn progressively higher-level features until linearly separable data at the last layer (or passed to MLP).

Autoencoders: Unsupervised model consisting of an encoder from high-dimensional input space to low-dimensional feature space, and a decoder from feature space back to output space. During training the model is made to reconstruct its input. Can be used for dimensionality reduction, new sample generation, noise filtering, anomaly detection, etc. or trained further on labelled data for final task (semi-supervised learning).

5 Bayesian Inference

Given the **likelihood** $p(y^{(1)}, \ldots, y^{(N)}|\theta) = p(\mathcal{D}|\theta)$ and the **prior distribution** $p(\theta)$, we wish to find:

1. **Posterior distribution:** $p(\theta|\mathcal{D}) = \dfrac{p(\mathcal{D}|\theta)p(\theta)}{\int p(\mathcal{D}|\theta')p(\theta')\,\mathrm{d}\theta'}$
2. **Posterior predictive distribution:** $p(y'|\mathcal{D}) = \displaystyle\int p(y'|\theta)p(\theta|\mathcal{D})\,\mathrm{d}\theta$

Conjugate prior: A prior with similar mathematical form as the likelihood; e.g. beta for Bernoulli/geometric, Gaussian for Gaussian, Dirichlet for multinomial, gamma for Poisson/exponential.

Evidence/marginal likelihood: Likelihood of observing the data when marginalizing across all possible parameters $\boldsymbol{w}$ according to the prior:

$$p(\boldsymbol{y}|\boldsymbol{X}) = \int p(\boldsymbol{y}|\boldsymbol{w}, \boldsymbol{X})p(\boldsymbol{w})\,\mathrm{d}\boldsymbol{w}$$

$$p(\boldsymbol{y}|\boldsymbol{X}, \alpha, \sigma^2) = \int p(\boldsymbol{y}|\boldsymbol{w}, \boldsymbol{X}, \sigma^2)p(\boldsymbol{w}|\alpha)\,\mathrm{d}\boldsymbol{w}$$

5.1 Bayesian Linear Regression

Given dataset $\mathcal{D} = \{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^N$, assume $y \sim \mathcal{N}(y|\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}), \sigma^2)$ giving likelihood $\log p(\boldsymbol{y}|\boldsymbol{w}, \boldsymbol{X}, \sigma^2) = \sum_{i=1}^N \log \mathcal{N}(y^{(i)}|\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}^{(i)}), \sigma^2)$, with prior $p(\boldsymbol{w}|\alpha) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{0}, \alpha\boldsymbol{1})$, Bayesian linear regression gives:

- Weight Posterior: $p(\boldsymbol{w}|\mathcal{D}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$
- Posterior predictive: $p(y'|\boldsymbol{x}', \mathcal{D}) = \mathcal{N}(y'|\boldsymbol{\mu}^T\boldsymbol{\phi}(\boldsymbol{x}'), \boldsymbol{\phi}(\boldsymbol{x}')^T\boldsymbol{\Sigma}\boldsymbol{\phi}(\boldsymbol{x}') + \sigma^2)$ where

$$\boldsymbol{\mu} = \frac{1}{\sigma^2}\boldsymbol{\Sigma}\boldsymbol{\phi}^T y \qquad \boldsymbol{\Sigma} = \left(\frac{1}{\sigma^2}\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \frac{1}{\alpha}\boldsymbol{1}\right)^{-1}$$

Complexity: $\mathcal{O}(NM^2 + M^3)$ time, $\mathcal{O}(NM + M^2)$ memory.

Type-II inference: Estimate σ^2 and α through numerical maximization of the (log) evidence/marginal likelihood (where N is number of samples):

$$\log p(\boldsymbol{y}|\boldsymbol{X}, \alpha, \sigma^2) = -\frac{N}{2}\log\alpha - \frac{N}{2}\log(\sigma^2) - \frac{1}{2\sigma^2}\boldsymbol{y}^T\boldsymbol{y} + \frac{1}{2}\boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} + \frac{1}{2}\log\det(\boldsymbol{\Sigma}) - \frac{N}{2}\log(2\pi)$$

5.2 Gaussian Processes

Define: **Gram matrix** $\boldsymbol{K}_{\boldsymbol{X},\boldsymbol{X}} = \alpha\boldsymbol{\Phi}\boldsymbol{\Phi}^T \in \mathbb{R}^{N\times N}$, where entry ij is $[\boldsymbol{K}_{\boldsymbol{X},\boldsymbol{X}}]_{ij} = \alpha\boldsymbol{\phi}^T(\boldsymbol{x}^{(i)})\boldsymbol{\phi}(\boldsymbol{x}^{(j)}) = k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$; k: $\mathcal{X}\times\mathcal{X} \mapsto \mathbb{R}$ is the **kernel**. Let $\boldsymbol{k}_{\boldsymbol{X},\boldsymbol{x}'} = \begin{bmatrix}k(\boldsymbol{x}^{(1)}, \boldsymbol{x}') & k(\boldsymbol{x}^{(2)}, \boldsymbol{x}') & \ldots & k(\boldsymbol{x}^{(N)}, \boldsymbol{x}')\end{bmatrix}^T$. Then the kernelized posterior predictive is

$$p(y'|\boldsymbol{x}', \mathcal{D}) = \mathcal{N}(y'|\mu_p, \sigma_p^2) \qquad \mu_p = \boldsymbol{k}_{\boldsymbol{x}',\boldsymbol{X}}(\boldsymbol{K}_{\boldsymbol{X},\boldsymbol{X}} + \sigma^2\boldsymbol{1})^{-1}\boldsymbol{y}$$
$$\sigma_p^2 = k(\boldsymbol{x}', \boldsymbol{x}') - \boldsymbol{k}_{\boldsymbol{x}',\boldsymbol{X}}(\boldsymbol{K}_{\boldsymbol{X},\boldsymbol{X}} + \sigma^2\boldsymbol{1})^{-1}\boldsymbol{k}_{\boldsymbol{X},\boldsymbol{x}'} + \sigma^2$$
$$\text{cov}(y_1', y_2') = k(\boldsymbol{x}_1', \boldsymbol{x}_2') - \boldsymbol{k}_{\boldsymbol{x}_1',\boldsymbol{X}}(\boldsymbol{K}_{\boldsymbol{X},\boldsymbol{X}} + \sigma^2\boldsymbol{1})^{-1}\boldsymbol{k}_{\boldsymbol{X},\boldsymbol{x}_2'} + \sigma^2$$

which is equivalent to a GLM with l_2 error and regularization, $\lambda = \sigma^2/\alpha$.

$k(\cdot,\cdot)$ specifies covariance between two observations; any collection of observations are jointly Gaussian, making this a **Gaussian process**.

Complexity: $\mathcal{O}(N^3)$ time, $\mathcal{O}(N^2)$ memory. Much more efficient than normal Bayesian linear regression when $M \gg N$ or possibly infinite (same kernels as GLMs).

Combining kernels: Through summation $k(x, y) = k_1(x, y) + k_2(x, y)$ or $k(\boldsymbol{x}, \boldsymbol{y}) = k_1(x_1, y_1) + k_2(x_2, y_2)$, multiplication $k(x, y) = k_1(x, y)k_2(x, y)$ or $k(\boldsymbol{x}, \boldsymbol{y}) = k_1(x_1, y_1)k_2(x_2, y_2)$, or composition $k(x, y) = k_1(f(x), f(y))$, all preserving positive-definiteness.

Type-II inference: In function-space,

$$\log p(\boldsymbol{y}|\boldsymbol{X}, \alpha, \sigma^2) = -\frac{N}{2}\log(2\pi) - \frac{1}{2}\log\det(\boldsymbol{K}_{\boldsymbol{X},\boldsymbol{X}} + \sigma^2\boldsymbol{1}) - \frac{1}{2}\boldsymbol{y}^T(\boldsymbol{K}_{\boldsymbol{X},\boldsymbol{X}} + \sigma^2\boldsymbol{1})^{-1}\boldsymbol{y}$$

5.3 Approximate Bayesian Inference

Given observed evidence, X_E and unobserved variables, X_F, find

$$p(X_F|X_E) = \frac{p(X_E, X_F)}{p(X_E)}$$

$p(X_E, X_F) = p(X_E|X_F)p(X_F)$ is often known from the model and prior assumptions, but $p(X_E) = \int p(X_E, X_F)\,\mathrm{d}X_F$ is intractable to compute.

Quadrature: Numerically integrate $p(X_E) = \int p(X_E, X_F)\,\mathrm{d}X_F$ by sampling a finite number of points. Infeasible due to the exponential growth in points required. Error usually scales as $\mathcal{O}(n^{-\frac{1}{d}})$ where d is the dimensionality of the integral. Beyond $d \geq 3$ Monte Carlo is better.

Laplace approximation: Let $X_F = \boldsymbol{z}$ and $\hat{\boldsymbol{z}}_{\text{MAP}} = \arg\max_{\boldsymbol{z}} \log\tilde{p}(\boldsymbol{z})$ where $\tilde{p}(\boldsymbol{z}) = p(X_E, \boldsymbol{z})$, then $p(\boldsymbol{z}|X_E) = \frac{1}{Z}p(X_E, \boldsymbol{z}) = \frac{1}{Z}\tilde{p}(\boldsymbol{z})$. Then:

$$p(\boldsymbol{z}|X_E) \approx \mathcal{N}(\boldsymbol{z}|\hat{\boldsymbol{z}}_{\text{MAP}}, \boldsymbol{A}^{-1}) \qquad \boldsymbol{A} = -\vec{\nabla}^2\log\tilde{p}(\boldsymbol{z})\Big|_{\boldsymbol{z}=\hat{\boldsymbol{z}}_{\text{MAP}}}$$

Note: from a second-order Taylor expansion:

$$\log p(\boldsymbol{z}|X_E) \approx \log\tilde{p}(\hat{\boldsymbol{z}}_{\text{MAP}}) - \frac{1}{2}(\boldsymbol{z} - \hat{\boldsymbol{z}}_{\text{MAP}})^T\boldsymbol{A}(\boldsymbol{z} - \hat{\boldsymbol{z}}_{\text{MAP}}) + \text{const.}$$

Simple but poor performance, since it does not account for global information. However, if posterior is Gaussian, this is exact. Requires existence of Hessian for likelihood and prior, so certain distributions (e.g. Laplace) might be bad.

5.3.1 Stochastic Variational Inference (SVI)

Kullback-Leibler (KL) divergence: For two distributions $p(\boldsymbol{z})$ and $q(\boldsymbol{z})$,

$$KL(q\parallel p) = \mathbb{E}_{\boldsymbol{z}\sim q(\boldsymbol{z})}\left[\log\frac{q(\boldsymbol{z})}{p(\boldsymbol{z})}\right] \implies \begin{cases} KL(q\parallel p) \geq 0 \\ KL(q\parallel p) = 0 \iff q = p \\ KL(q\parallel p) \neq KL(p\parallel q)\end{cases}$$

Reverse-KL (information projection): take $KL(q\parallel p)$ which penalizes q having mass where p has none (compressing q to fit to one peak of p); *forward-KL (moment projection):* take $KL(p\parallel q)$ which penalizes q missing mass where p has mass (stretching out q to cover all peaks of p). Reverse KL commonly used for computational reasons.

Stochastic Variational Inference (SVI): Approximating the true conditional $p(\boldsymbol{z}|\boldsymbol{x}) = \frac{p(\boldsymbol{z},\boldsymbol{x})}{p(\boldsymbol{x})}$ by a simpler distribution, $q(\boldsymbol{z}|\boldsymbol{\theta})$, from a known family of distributions, such that $KL(q(\boldsymbol{z}|\boldsymbol{\theta})\parallel p(\boldsymbol{z}|\boldsymbol{x}))$ is minimized.

Evidence lower bound (ELBO): $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{x}) = -\mathbb{E}_{\boldsymbol{z}\sim q}\left[\log\frac{q(\boldsymbol{z}|\boldsymbol{\theta})}{p(\boldsymbol{z},\boldsymbol{x})}\right]$ is a lower bound for $\log p(\boldsymbol{x})$; maximizing ELBO minimizes KL. Has gradient

$$\vec{\nabla}_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{x}) = \vec{\nabla}_{\boldsymbol{\theta}}\int q(\boldsymbol{z}|\boldsymbol{\theta})\log\frac{p(\boldsymbol{x},\boldsymbol{z})}{q(\boldsymbol{z}|\boldsymbol{\theta})}\,\mathrm{d}\boldsymbol{z}$$

Score function/REINFORCE gradient estimator: By Monte Carlo:

$$\vec{\nabla}_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{x}) = \mathbb{E}_{\boldsymbol{z}\sim q}\left[\vec{\nabla}_{\boldsymbol{\theta}}\log q(\boldsymbol{z}|\boldsymbol{\theta})\log\frac{p(\boldsymbol{x},\boldsymbol{z})}{q(\boldsymbol{z}|\boldsymbol{\theta})}\right] \approx \frac{1}{B}\sum_{i=1}^{B}\vec{\nabla}_{\boldsymbol{\theta}}\log q(\boldsymbol{z}^{(i)}|\boldsymbol{\theta})\log\frac{p(\boldsymbol{x},\boldsymbol{z}^{(i)})}{q(\boldsymbol{z}^{(i)}|\boldsymbol{\theta})}$$

Pathwise/reparametrization gradient estimator: Find $T(\boldsymbol{\varepsilon}, \boldsymbol{\theta})$ such that for $\boldsymbol{\varepsilon} \sim p(\boldsymbol{\varepsilon})$, then $\boldsymbol{z} = T(\boldsymbol{\varepsilon}, \boldsymbol{\theta}) \implies \boldsymbol{z} \sim q(\boldsymbol{z}|\boldsymbol{\theta})$. Then Monte Carlo:

$$\vec{\nabla}_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{x}) = \mathbb{E}_{\boldsymbol{\varepsilon}\sim p(\boldsymbol{\varepsilon})}\left[\vec{\nabla}_{\boldsymbol{\theta}}\log\frac{p(\boldsymbol{x}, T(\boldsymbol{\varepsilon},\boldsymbol{\theta}))}{q(T(\boldsymbol{\varepsilon},\boldsymbol{\theta})|\boldsymbol{\theta})}\right] \approx \frac{1}{B}\sum_{i=1}^{B}\vec{\nabla}_{\boldsymbol{\theta}}\log\frac{p(\boldsymbol{x}, T(\boldsymbol{\varepsilon}^{(i)},\boldsymbol{\theta}))}{q(T(\boldsymbol{\varepsilon}^{(i)},\boldsymbol{\theta})|\boldsymbol{\theta})}$$

e.g. for a Gaussian, $\boldsymbol{\theta} = \{\mu, \sigma^2\}$, let $\varepsilon \sim \mathcal{N}(\varepsilon|0, 1)$ and $T(\varepsilon, \boldsymbol{\theta}) = \sigma\varepsilon + \mu$, then $z \sim \mathcal{N}(z|\mu, \sigma^2)$.

5.3.2 Expectation Approximation

Instead of $p(\boldsymbol{x})$ sometimes we only need $I = \mathbb{E}_{\boldsymbol{x}\sim p(\boldsymbol{x})}[\phi(\boldsymbol{x})]$.

Monte Carlo: Collect R random samples from $p(\boldsymbol{x})$, denoted $\boldsymbol{x}^{(i)}$, then

$$I = \int\phi(\boldsymbol{x})p(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} \approx \hat{I} = \frac{1}{R}\sum_{i=1}^{R}\phi(\boldsymbol{x}^{(i)})$$

Has standard deviation proportional to $1/\sqrt{R}$, independent of dimension; unbiased estimator. For dimension $d \geq 3$ this generally outperforms quadrature. However, we might only know the unnormalized $\tilde{p}(\boldsymbol{x}) = Zp(\boldsymbol{x})$, or sampling may be difficult due to high dimensionality, in which case we need importance sampling.

Importance Sampling: Let $q(\boldsymbol{x})$ be the *sampler density*, a density easy to sample from; then for some unnormalized $\tilde{p}(\boldsymbol{x}) = Zp(\boldsymbol{x})$,

$$I = \frac{\mathbb{E}_{\boldsymbol{x}\sim q(\boldsymbol{x})}\left[\frac{\phi(\boldsymbol{x})\tilde{p}(\boldsymbol{x})}{q(\boldsymbol{x})}\right]}{\mathbb{E}_{\boldsymbol{x}\sim q(\boldsymbol{x})}\left[\frac{\tilde{p}(\boldsymbol{x})}{q(\boldsymbol{x})}\right]} \implies \hat{I} = \frac{\frac{1}{R}\sum_{r=1}^{R}\frac{\phi(\boldsymbol{x}^{(r)})\tilde{p}(\boldsymbol{x}^{(r)})}{q(\boldsymbol{x}^{(r)})}}{\frac{1}{R}\sum_{r=1}^{R}\frac{\tilde{p}(\boldsymbol{x}^{(r)})}{q(\boldsymbol{x}^{(r)})}} = \frac{\sum_r w_r\phi(\boldsymbol{x}^{(r)})}{\sum_r w_r}$$

where each $w_r = \tilde{p}(\boldsymbol{x}^{(r)})/q(\boldsymbol{x}^{(r)})$ is an *importance weight*, which is larger if sampling from $q(\boldsymbol{x})$ tends to under-represent the point, or smaller for over-represent. Note if $\tilde{p}(\boldsymbol{x}) = p(\boldsymbol{x})$ is normalized, then

$$\mathbb{E}_{\boldsymbol{x}\sim q(\boldsymbol{x})}\left[\frac{\tilde{p}(\boldsymbol{x})}{q(\boldsymbol{x})}\right] = 1 \implies \hat{I} = \frac{1}{R}\sum_{r=1}^{R}\frac{\phi(\boldsymbol{x}^{(r)})\tilde{p}(\boldsymbol{x}^{(r)})}{q(\boldsymbol{x}^{(r)})} = \frac{1}{R}\sum_r w_r\phi(\boldsymbol{x}^{(r)})$$

Sampler should have heavy tails (e.g. Cauchy instead of Gaussian) to compensate for differences between $p(\boldsymbol{x})$ and $q(\boldsymbol{x})$. If the sampler is chosen improperly, the variance of the result can be extremely high. In high dimensions, if the sampler distribution is not a near-perfect approximation of the target, then the entire sum will likely be dominated by a few samples with a huge weight, leading to a very bad estimate.

6 Useful Identities

Matrix differentiation: $\frac{\partial}{\partial\boldsymbol{z}}(\boldsymbol{z}^T\boldsymbol{A}\boldsymbol{z}) = (\boldsymbol{A} + \boldsymbol{A}^T)\boldsymbol{z}$ $\qquad$ $\frac{\partial}{\partial\boldsymbol{z}}(\boldsymbol{A}\boldsymbol{z}) = \boldsymbol{A}^T$

Laplace distribution: $\text{Lap}(\epsilon|\mu, b) = \frac{1}{2b}e^{-\frac{|\epsilon-\mu|}{b}}$ with $\sigma^2 = 2b^2$.

Gaussian distribution: $\mathcal{N}(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma}}e^{-\frac{1}{2\sigma^2}(x-\mu)^2}$; multi-var ver.:

$$\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{\frac{n}{2}}\det(\boldsymbol{\Sigma})^{\frac{1}{2}}}\exp\left[-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu})\right]$$

Partition Gaussian $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ as:

$$\boldsymbol{x} = \begin{bmatrix}\boldsymbol{x}_1\\ \boldsymbol{x}_2\end{bmatrix}, \boldsymbol{\mu} = \begin{bmatrix}\boldsymbol{\mu}_1\\ \boldsymbol{\mu}_2\end{bmatrix}, \boldsymbol{\Sigma} = \begin{bmatrix}\boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12}\\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22}\end{bmatrix} \implies \boldsymbol{x} \sim f(\boldsymbol{x}_1, \boldsymbol{x}_2)$$

Then $\boldsymbol{x}_1 \sim \mathcal{N}(\boldsymbol{x}_1|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_{11}), \boldsymbol{x}_2 \sim \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_{22})$, and

$$f(\boldsymbol{x}_1|\boldsymbol{x}_2) = \mathcal{N}(\boldsymbol{x}_1|\boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\boldsymbol{x}_2 - \boldsymbol{\mu}_2), \boldsymbol{\sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21})$$

Beta distribution: $p(\theta|\alpha, \beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\theta^{\alpha-1}(1-\theta)^{\beta-1}$ ($\alpha = \beta = 1$ gives uniform dist.) with mean $\frac{\alpha}{\alpha+\beta}$ and mode $\frac{\alpha-1}{\alpha+\beta-2}$.

Dirichlet distribution: $p(\boldsymbol{\theta}|\alpha_1, \ldots, \alpha_K) = \frac{\Gamma(\sum_k\alpha_k)}{\prod_k\Gamma(\alpha_k)}\prod_k\theta_k^{\alpha_k-1}$ where $\alpha_k > 0, \sum_k\alpha_k = 1$.

Sigmoid & Derivative: $\sigma(x) = \frac{1}{1+e^{-x}} \implies \frac{\mathrm{d}}{\mathrm{d}x}\sigma(x) = \sigma(x)(1 - \sigma(x))$