Introduction to Machine Learning 252-0220-00L

Exam cheatsheet for the FS24 exam at ETH Zürich by Wu, You

Basics

$\binom{n}{p} = \frac{n!}{p!(n-p)!}$, $(x+y)^n = \sum_{p=0}^n x^p y^{n-p}, x, y \in \mathbb{R}$

- $(a+b)^3 = a^3 + 3a^2b + 3ab^2 + b^3$

$\dim \mathcal{P}_p(\mathbb{R}^d) = \binom{d+p}{p}$ for all $p \in \mathbb{N}_0, d \in \mathbb{N}$ with lead. order

- $p \to \infty : \dim \mathcal{P}_p(\mathbb{R}^d) = O(p^d); d \to \infty : \dim \mathcal{P}_p(\mathbb{R}^d) = O(d^p)$

$f(\boldsymbol{x}) = f(\boldsymbol{x}_0) + J_f(\boldsymbol{x}_0)(\boldsymbol{x} - \boldsymbol{x}_0) + o(\|\boldsymbol{x} - \boldsymbol{x}_0\|)$

$$\begin{pmatrix} a \\ b \end{pmatrix} \begin{pmatrix} c & d \end{pmatrix} = \begin{pmatrix} ac & ad \\ bc & bd \end{pmatrix}$$

- $\boldsymbol{\nabla}_{\boldsymbol{x}} \|\boldsymbol{x}\|_2^2 = 2\boldsymbol{x}$, $\frac{\partial}{\partial \boldsymbol{x}} \boldsymbol{b}^T \boldsymbol{x} = \frac{\partial}{\partial \boldsymbol{x}} \boldsymbol{x}^T \boldsymbol{b} = \boldsymbol{b}$
- $\frac{\partial}{\partial \boldsymbol{x}} (\boldsymbol{b}^T \boldsymbol{A} \boldsymbol{x}) = \boldsymbol{A}^T \boldsymbol{b}$
- $\frac{\partial}{\partial \boldsymbol{x}} (\boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x}) = (\boldsymbol{A} + \boldsymbol{A}^T)\boldsymbol{x} \overset{\text{if A sym.}}{=} 2\boldsymbol{A}\boldsymbol{x}$
- $\frac{\partial}{\partial \boldsymbol{X}} (\boldsymbol{c}^T \boldsymbol{X} \boldsymbol{b}) = \boldsymbol{c}^T \boldsymbol{b}$
- $\frac{\partial}{\partial \boldsymbol{X}} (\boldsymbol{c}^T \boldsymbol{X}^T \boldsymbol{b}) = \boldsymbol{b}\boldsymbol{c}^T$

$\boldsymbol{g} = \boldsymbol{W_2 h} \overset{\frac{\partial}{\partial \boldsymbol{h}}}{\to} \boldsymbol{W_2}$

$\boldsymbol{h} = \sigma(\boldsymbol{z}) \overset{\frac{\partial}{\partial \boldsymbol{z}}}{\to} \odot \sigma'(\boldsymbol{z})$

$\boldsymbol{z} = \boldsymbol{W}_1 \boldsymbol{x} \overset{\boldsymbol{\delta} \frac{\partial}{\partial \boldsymbol{W_1}}}{\to} \boldsymbol{\delta}^T \boldsymbol{x}^T$

LSE with Squared Matrix $A^{n \times n}$

- $\boldsymbol{Ax} = \boldsymbol{0}$, if $\det(A) \neq 0 \Rightarrow \boldsymbol{x} = \boldsymbol{0}$, else $(n-r)$ free variables
- $\boldsymbol{Ax} = \boldsymbol{b}$, if $\det(A) \neq 0 \Rightarrow$ unique solution
  - else $\begin{cases} (n-r) \text{ free variables,} & \text{Rank}(A) = \text{Rank}(A|b) = r < n \\ \text{no solutions} & \text{Rank}(A) \neq \text{Rank}(A|b) \end{cases}$

LSE with General Matrix $A^{m \times n}$

- $\boldsymbol{Ax} = \boldsymbol{b}$, if $\text{Rank}(A) < \text{Rank}(A|b) \Rightarrow$ no solutions
- else if $\text{Rank}(A) = \text{Rank}(A|b) = r \Rightarrow \begin{cases} (n-r) \text{ free variables,} & r < n \\ \text{Unique solutions} & r = n \end{cases}$

Supervised Learning

- $f : \mathbb{R}^n \to \mathbb{R}$ cont diff and x^* stationary point $\Rightarrow \nabla f(x^*) \overset{!}{=} 0$
- x is a local minimum $\Rightarrow \nabla f[x] = 0$ and $Hf[x] \geq 0$
- $f : \mathbb{R}^n \to \mathbb{R}$ continuous is **convex**: for all $t \in (0,1)$
  - $f(tx + (1-t)y) \leq tf(x) + (1-t)f(y)$
- strictly convex if $<$ holds, concave if $>$ holds, Hessian > 0

Composition: $X \subset \mathbb{R}^n$ **convex**, $f_1, ..., f_s : X \to \mathbb{R}$ **convex**

1. $\sum_{i=1}^s \lambda_i f_i$ **is convex for all** $\lambda_i \in \mathbb{R}_{\geq 0}$
2. $\max\{f_i, ..., f_s\}$ **convex**
3. $f_1(f_2(x))$ **convex for** f_1 **being non-decreasing**

(Batch) GD: $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t \boldsymbol{\nabla} L(\boldsymbol{w}_t)$ with $\boldsymbol{w}^{(0)}$ picked arbitrary

- high momentum → long-lasting U-turn, oscillation↓
- **loss** should be decreasing, convex in its argument & diff-able
  - only exponential and logistic suffice
- $L(\boldsymbol{w}^{t+1}) = L(\boldsymbol{w}^t + \eta \boldsymbol{v}) = L(\boldsymbol{w}^t) + \eta \langle \nabla L(\boldsymbol{w}^t), \boldsymbol{v} \rangle + o(\eta)$
- $\|\boldsymbol{w}^{t+1} - \boldsymbol{w}^t\|_2^2 = \|-\eta \boldsymbol{\nabla} L(\boldsymbol{w}^t)\|_2 = \eta \|\boldsymbol{\nabla} L(\boldsymbol{w}^t)\|_2$

SGD at each iteration, decrease of loss not guaranteed

Minibatch GD batch size large ⇒ loss plot smoothens out. Average gradients from diff. images are expected to lead in the optimal direction in the weight space

Batch Normalization standardize only batches of activation units v_i at t_0 and training. Regularize by limit magnitude of v_i, avoids internal covariate shift, vanishing/exploding gradient → larger learning rates, faster convg.

Dropout large weights often come as a result of overfitting and dropout helps by randomly dropping weights during training

Depth reduction multiplicative nature of the chain rule → small network depth prevents vanishing/exploding gradients

Regression Objective: $\boldsymbol{w}^* := \arg\min_{\boldsymbol{w}} L(w)$

$$\boldsymbol{y} = \boldsymbol{w}^{*T} \boldsymbol{x} + \varepsilon \Leftrightarrow \boldsymbol{y} = \boldsymbol{X}\boldsymbol{w}^* + \varepsilon$$

Ordinary Least Squares with closed form solution

$\boldsymbol{w}^* = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{y}$ $\qquad \Theta(nd^2 + d^3)$

$L(\boldsymbol{w}) := \sum_{i=1}^n (y_i - \boldsymbol{w}^T \boldsymbol{x}_i)^2 = \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|_2^2$

$$\boldsymbol{\nabla}_{\boldsymbol{w}} L(\boldsymbol{w}) = -2 \sum_{i=1}^n (y_i - \boldsymbol{w}^T \boldsymbol{x}_i) \cdot \boldsymbol{x}_i$$

$$= 2\boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}) \sim \Theta(nd)$$

Ridge Regression with closed form solution. Lasso uses $\lambda \|\boldsymbol{w}\|_1$

$\boldsymbol{w}^* = (\boldsymbol{X}^T \boldsymbol{X} + \lambda \boldsymbol{I})^{-1} \boldsymbol{X}^T \boldsymbol{y}$

$L(\boldsymbol{w}) := \sum_{i=1}^n (y_i - \boldsymbol{w}^T \boldsymbol{x}_i)^2 + \lambda \|\boldsymbol{w}\|_2^2$

$\boldsymbol{\nabla}_{\boldsymbol{w}} L(\boldsymbol{w}) = -2 \sum_{i=1}^n (y_i - \boldsymbol{w}^T \boldsymbol{x}_i) \cdot \boldsymbol{x}_i + 2\lambda \boldsymbol{w}$

Logistic Regression $\mathbb{P}[Y = y | \boldsymbol{x}] = \frac{1}{1 + \exp(-y\boldsymbol{w}^T \boldsymbol{x})}$

$L(\boldsymbol{w}) := \log[1 + \exp(-y\boldsymbol{w}^T \boldsymbol{x})]$

$\boldsymbol{\nabla}_{\boldsymbol{w}} L(\boldsymbol{w}) = \frac{1}{1 + \exp(y\boldsymbol{w}^T \boldsymbol{x})}(-y\boldsymbol{x})$

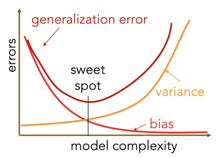
Regularized Logistic Regression

- **L2:** $\min_{\boldsymbol{w}} \sum_{i=1}^n \log[1 + \exp(-y\boldsymbol{w}^T \boldsymbol{x})] + \lambda \|\boldsymbol{w}\|_2^2$
- **GD Step:** $\boldsymbol{w} \leftarrow \boldsymbol{w}(1 - 2\lambda \eta_t) - \eta_t \boldsymbol{\nabla}_{\boldsymbol{w}} L(\boldsymbol{w})$

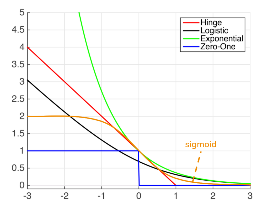
Kernelized Logistic Regression

- $\hat{\boldsymbol{\alpha}} = \arg\min_{\alpha} \sum_{i=1}^n \log[1 + \exp(-y_i \boldsymbol{\alpha}^T \boldsymbol{K}_i)] + \lambda \boldsymbol{\alpha}^T \boldsymbol{K} \boldsymbol{\alpha}$
- **Classification** $\hat{P}(y | \boldsymbol{x}, \hat{\boldsymbol{\alpha}}) = \frac{1}{1 + \exp\left(-y \sum_{j=1}^n \alpha_j k(\boldsymbol{x}_j, \boldsymbol{x})\right)}$)

Cross-entropy loss $l(\boldsymbol{W}; \boldsymbol{x}, y) = -\log \frac{\exp(f_y)}{\sum_{y'} \exp(f_{y'})}$



Bias, training, validation error
Variance, validation error



Classification Objective: $\boldsymbol{w}^* := \arg\min_{\boldsymbol{w}} \ell(\boldsymbol{w}; \mathcal{D})$

0/1 Loss: $\ell_{0/1}(\boldsymbol{w}; y_i, \boldsymbol{x}_i) = \begin{cases} 1 \text{ if } y_i \neq \text{sign}(\boldsymbol{w}^T \boldsymbol{x}_i) \\ 0 \text{ else} \end{cases}$

Perceptron: algorithm uses together with SGD

- lin. separable ⇔ ∃ lin. separator (not necessarily optimal)

$\ell_p(\boldsymbol{w}; y_i, \boldsymbol{x}_i) \overset{\text{def}}{=} \max(0, -y_i \boldsymbol{w}^T \boldsymbol{x}_i)$

$\boldsymbol{\nabla} \ell_p(\boldsymbol{w}; y_i, \boldsymbol{x}_i) = \begin{cases} 0 & \text{if } y_i \boldsymbol{w}^T \boldsymbol{x}_i \geq 0 \\ -y_i \boldsymbol{x}_i & \text{else} \end{cases}$

Objective: $\boldsymbol{w}^* := \arg\min_{\boldsymbol{w}} \ell(\boldsymbol{w}; \mathcal{D}) + \lambda C(\boldsymbol{w})$

Support Vector Machine (SVM) (Hinge & $C(\boldsymbol{w}) = \|\boldsymbol{w}\|_2^2$)

$\ell_H(\boldsymbol{w}; y_i, \boldsymbol{x}_i) \overset{\text{def}}{=} \max(0, 1 - y_i \boldsymbol{w}^T \boldsymbol{x}_i)$

$\boldsymbol{\nabla} \ell_H(\boldsymbol{w}; y_i, \boldsymbol{x}_i) = \begin{cases} 0 & \text{if } y_i \boldsymbol{w}^T \boldsymbol{x}_i \geq 1 \\ -y_i \boldsymbol{x}_i & \text{else} \end{cases}$

L1-SVM (Hinge loss & $C(\boldsymbol{w}) = \|\boldsymbol{w}\|_1$)

	y=+1	y=−1
ŷ=+1	TP	FP
ŷ=−1	FN	TN

- $n = \underbrace{n_+}_{\text{TP+FN}} + \underbrace{n_-}_{\text{TN+FP}}$

- **Accuracy:** $\frac{TP+TN}{n}$
- **Precision:** $\frac{TP}{TP+FP} = \frac{TP}{p_+}$
- **Recall/TPR:** $\frac{TP}{n_+}$ **FPR:** $\frac{FP}{n_-}$
- **F1:** $\frac{2TP}{TP+FP+FN} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$
- **Precision Recall Curve:**
  - **y = Precision, x = Recall**
- **ROC Curve: y = TPR, x = FPR**

k-nearest neighbors (nonlinear) no weights & training, class. during test time. High dim works bad (distance↑).Large n to perform well but $O(nd) \to O(n^\rho), \rho < 1$ by allowing error probability. Decision boundary curvilinear. missing data calculate dist. to all neighbors and classified based on values from closest k neighbors. Choose k via CV. Small $k \Rightarrow$ ↓bias, ↑variance (overfit)

$$y = \text{sign}\left(\sum_{i=1}^n y_i [x_i \text{ among k nearest neighbors of } \boldsymbol{x}]\right)$$

Decision Tree (nonlinear) no. level/depth chosen unwisely → small leaf nodes, overfit to noise. ☠️Greedy with bad top nodes

Kernel Methods $k(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})^T \phi(\boldsymbol{x}')$

$$\underbrace{\boldsymbol{K}}_{\in \mathbb{R}^{n \times n}} \overset{\text{def}}{:=} \begin{pmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_1, \boldsymbol{x}_n) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{x}_n, \boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_n, \boldsymbol{x}_n) \end{pmatrix} \text{p.s.d} \Leftrightarrow \forall \boldsymbol{x} \in \mathbb{R}^n : \underbrace{\boldsymbol{x}^T \boldsymbol{K} \boldsymbol{x} \geq 0}_{\lambda_i \geq 0}$$

Composition $k(\boldsymbol{x}, \boldsymbol{x}')$ as ① $k_1 + k_2$ ② $k_1 \cdot k_2$ ③ $c \cdot k_1$ for $c > 0$ or $f(k_1)$, with $f \equiv \exp$ or polynomial with pos coeff.

Important Kernels

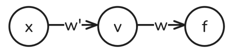
- $k(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^T \boldsymbol{y} + C)^n$, with $C \geq 0, n \geq 1$
- $k(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(\frac{-\|\boldsymbol{x} - \boldsymbol{y}\|_2^2}{2\sigma^2}\right)$, $\sigma > 0$ (Gaussian)
- $k(\boldsymbol{x}, \boldsymbol{y}) = \exp(-\alpha \|\boldsymbol{x} - \boldsymbol{y}\|_p)$, $\alpha > 0$ (Lapl. $p = 2$, Abel $p = 1$)

Feature Map may be asymmetric, used to find correct mapping given data. If induced by a valid kernel, may be ∞-dimensional

Neural Network ① $\boldsymbol{z} := w'x$ ② $\boldsymbol{v} = \phi(z)$ ③ $f = wv$ **(Forward)**

- linear NN with id more layers $\not\Rightarrow$ complexity↑

- **Back Propagation** $\boldsymbol{\nabla}_{\boldsymbol{w}} \ell(w_i, x, y) = [\frac{\partial}{\partial w} \ell, \frac{\partial}{\partial w'} \ell]$



$$\frac{\partial}{\partial w} \ell = \underbrace{\frac{\partial \ell}{\partial f}}_{\delta} \cdot \underbrace{\frac{\partial f}{\partial w}}_{v}$$

$$\frac{\partial}{\partial w'} \ell = \underbrace{\frac{\partial \ell}{\partial f}}_{\delta} \cdot \underbrace{\frac{\partial f}{\partial v}}_{w} \cdot \underbrace{\frac{\partial v}{\partial z}}_{\phi'(z)} \cdot \underbrace{\frac{\partial z}{\partial w'}}_{x}$$

Activation	$\phi(z)$	$\phi'(z)$
Sigmoid $1 - \sigma(z) = \sigma(-z)$	$\frac{1}{1+\exp(-z)}$	$\phi(z)(1 - \phi(z))$
Tanh	$\frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)}$	$1 - \phi^2(z)$
ReLU (nonlinear)	$\max(z, 0)$	$\begin{cases} 1 \text{ if } z > 0 \\ 0 \text{ otherwise} \end{cases}$

Vanishing Gradient $\varphi'(z) \approx 0$ Keep variance of weights approximately constant across layers to avoid vanishing and exploding gradients and network activations (most-prone to VG sigmoid/tanh > ReLU)

Zero-centered $\mathbb{E}[f(X)] \overset{!}{=} 0$ (id, tanh symmetric around zero)

CNN: For **each dimension**, the resulting dimension:

$$l = \frac{n + 2\overbrace{p}^{\text{padding}} - \overbrace{f}^{\text{filter}}}{\underbrace{s}_{\text{stride}}} + 1$$

- Number of trainable parameters (default) a filter has the same number of channels as the input

$$\#\text{trainable} = \#\text{filter} * \prod_i \#\text{filter\_dim\_i} * \#\text{input channels}$$

Unsupervised Learning

k-Means: CV works badly, low loss if centers close to test set

$$L(\boldsymbol{\mu}) = L(\boldsymbol{\mu}_1, ..., \boldsymbol{\mu}_k) := \sum_{i=1}^n \min_{j \in [k]} \|\boldsymbol{x}_i - \boldsymbol{\mu}_j\|_2^2$$

$$\hat{\boldsymbol{\mu}} = \arg\min_{\boldsymbol{\mu}} L(\boldsymbol{\mu}) \quad \text{non-convex, NP-hard}$$

Lloyd's Heuristics: ∼ exponential, in practice not that bad

Initialize cluster center $\boldsymbol{\mu}^{(0)} := \left[\boldsymbol{\mu}_1^{(0)}, ..., \boldsymbol{\mu}_k^{(0)}\right]$

While not converged, each iteration $O(nkd)$

- assign points to **closest center** $z_i^{(t)} \leftarrow \arg\min_j \left\|\boldsymbol{x}_i - \boldsymbol{\mu}_j^{(t-1)}\right\|_2^2$
- update centers to **mean of each cluster** $\boldsymbol{\mu}_j^{(t)} \leftarrow \frac{1}{n_j} \sum_{i : \boldsymbol{z}_i^{(t)} = \boldsymbol{j}} \boldsymbol{x}_i$

as Hard-EM assume identical, spherical covariance matrices, uniform weights over mixture components. as soft-EM same assumption, with additionally variances → 0

k-Means++: start with random point as center and add centers randomly, propor. to the squared distance to closest center. Opt expected cost within $O(\log k)$ $(L(\boldsymbol{\mu}) \leq O(\log k) \min_{\mu} L(\boldsymbol{\mu}))$

Dimensional Reduction

PCA Problem: $W \in \mathbb{R}^{d\times k}$ orthogonal and $x_i \in \mathbb{R}^d$, $z_i \in \mathbb{R}^k$

$$(\boldsymbol{W}, \boldsymbol{z}_1, \dots, \boldsymbol{z}_n) = \arg\min_{\boldsymbol{W}^T\boldsymbol{W}=\boldsymbol{I}_k} \sum_{i=1}^n \left\|\boldsymbol{W}\underline{\boldsymbol{W}^T\boldsymbol{x}_i} - \boldsymbol{x}_i\right\|_2^2$$
$$= \arg\min \sum_{i=1}^n \left\|\boldsymbol{W}\underline{\boldsymbol{z}_i} - \boldsymbol{x}_i\right\|_2^2$$

Special Case with Closed-form Solution $(k > 1)$

- centered data $\boldsymbol{\mu} = \frac{1}{n}\sum_i \boldsymbol{x}_i \overset{!}{=} 0$ and $\boldsymbol{\Sigma} = \frac{1}{n}\sum_{i=1}^n \boldsymbol{x}_i\boldsymbol{x}_i^T$

PCA Solution non-convex, both $\boldsymbol{w}$ and $-\boldsymbol{w}$ are optimal solution

$\boldsymbol{W} := \underbrace{(\boldsymbol{v}_1|\dots|\boldsymbol{v}_k)}_{1\le k\le d}$ and $\boldsymbol{z}_i = \boldsymbol{W}^T\boldsymbol{x}_i$ hereby $\boldsymbol{\Sigma} = \sum_{i=1}^d \lambda_i \boldsymbol{v}_i\boldsymbol{v}_i^T$

Kernel PCA

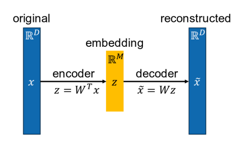
- $\boldsymbol{z}_i \overset{\text{def}}{:=} \sum_{j=1}^n \alpha_j^{(i)} k(\boldsymbol{x}, \boldsymbol{x}_j)$
- $\alpha_j^{(i)}$ is the j-th component of vector $\boldsymbol{\alpha}^{(i)} \overset{\text{def}}{:=} \frac{1}{\sqrt{\lambda_i}}\boldsymbol{v}_i$
- $\boldsymbol{K} = \sum_{i=1}^n \lambda_i \boldsymbol{v}_i\boldsymbol{v}_i^T$

Projection Matrix $\underbrace{\boldsymbol{X}}_{\in\mathbb{R}^{m\times n}} \boldsymbol{X}^T \underbrace{\boldsymbol{b}}_{\in\mathbb{R}^m} = \boldsymbol{X}\begin{pmatrix}\boldsymbol{x}_1^T\boldsymbol{b}\\ \vdots \\ \boldsymbol{x}_n^T\boldsymbol{b}\end{pmatrix} = \sum_{i=1}^n \boldsymbol{x}_i^T\boldsymbol{b}\boldsymbol{x}_i$

- Orthogonal square matrix fulfills $\boldsymbol{P}^2 = \boldsymbol{P}$

Find projected point given eigen vector $\boldsymbol{v}_1$

$$\boldsymbol{x}^{\text{PCA}} := \frac{\boldsymbol{v}_1\boldsymbol{v}_1^T}{\|\boldsymbol{v}_1\|_2}\boldsymbol{x} = \frac{\boldsymbol{v}_1^T\boldsymbol{x}}{\|\boldsymbol{v}_1\|_2}\boldsymbol{v}_1$$



Autoencoder find w^*

$$\arg\min_{\boldsymbol{w}} \sum_{i=1}^n \underbrace{\|\boldsymbol{x}_i - f(\boldsymbol{x}_i;\boldsymbol{W})\|_2^2}_{\text{square loss check output } \approx \text{ input}}$$

$$f(x;\theta) \overset{\text{def}}{:=} f_{\text{dec}}(f_{\text{enc}}(x;\theta_{\text{enc}});\theta_{\text{dec}})$$

- **Equivalence of fitting a NN autoencoder to PCA (when $\varphi \equiv \text{id}$)**

Generative Models $p(x, y)$ can be more powerful (detect outliers, missing values) with met assumptions, typically less robust against outliers

Discriminative Models $p(y|x)$ ~~detect outliers~~, but more robust

GANS finds saddle point instead of local minimum

Probabilistic Modeling: choose distribution family $\mathcal{P}$

- conditional $\mathbb{P}[A, B] = \mathbb{P}[A|B] \cdot \mathbb{P}[B]$
  - $p(\boldsymbol{x}_i, y_i; \theta) = p(y_i|\boldsymbol{x}_i;\theta)p(\boldsymbol{x}_i)$
- Independent $\mathbb{P}[A, B] = \mathbb{P}[A]\mathbb{P}[B]$
- Bayes' rule

$$\mathbb{P}[B_i|A] = \frac{\mathbb{P}[A|B_i]\mathbb{P}[B_i]}{\mathbb{P}[B]} = \frac{\mathbb{P}[A|B_i]\mathbb{P}[B_i]}{\sum_{j=1}^n \mathbb{P}[A|B_j]\mathbb{P}[B_j]}$$

  - $p_{X|Y}(x, y) = \frac{p_{Y|X}(y|x)p_X(x)}{\int p_{Y|X}(y|x')p_X(x')dx'}$
- $$\mathbb{E}[X] := \begin{cases}\sum_{x\in E} x \cdot \mathbb{P}[X = x] & \text{if discrete } X : \Omega \to E \\ \int_{-\infty}^{\infty} x \cdot f(x)dx & \text{if cont. } X : \Omega \to \mathbb{R}\end{cases}$$
- $\sigma^2 = \text{Var}[X] = \mathbb{E}[(X - \mathbb{E}[X])^2] = \mathbb{E}[X^2] - \mathbb{E}[X]^2$

$X \sim$	$p(x) = \mathbb{P}[X = x]$	$\mathbb{E}[X]$	$\text{Var}[X]$
$\text{Ber}(p)$	$p^x(1-p)^{1-x}$	p	$p(1-p)$
$\text{Bin}(n, p)$	$\binom{n}{k}p^k(1-p)^{n-k}$	np	$np(1-p)$
$\text{Poisson}(\lambda)$	$e^{-\lambda}\frac{\lambda^k}{k!}$	λ	λ
$\text{Geom}(p)$	$p(1-p)^{k-1}$	$\frac{1}{p}$	$\frac{1-p}{p^2}$
$\mathcal{U}([a, b])$	$\begin{cases}\frac{1}{b-a} & \text{if } x \in [a, b]\\ 0 & \text{else}\end{cases}$	$\frac{a+b}{2}$	$\frac{1}{12}(b-a)^2$
$\text{Exp}(\lambda)$	$\begin{cases}\lambda e^{-\lambda x} & \text{if } x \ge 0\\ 0 & \text{else}\end{cases}$	$\frac{1}{\lambda}$	$\frac{1}{\lambda^2}$
$\mathcal{N}(\mu, \sigma^2)$	$\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$	μ	σ^2

Multivariate Gaussian, empirically, $\hat{\boldsymbol{\Sigma}} \overset{\text{def}}{:=} \frac{1}{n}\sum_{i=1}^n \boldsymbol{x}_i\boldsymbol{x}_i^T$

$$\frac{1}{2\pi\sqrt{|\boldsymbol{\Sigma}|}} \cdot \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu})\right)$$

Statistical Inference: Estimating $\mathbb{P}^{\hat{\theta}}_{X,Y}$

- we used **discriminative model**, where θ only parameterizes $\hat{\mathbb{P}}^{\theta}_{Y|X} \approx \mathbb{P}^{\theta}_{Y|X}$

KL-Divergence $D_{KL}(P\|Q) = \mathbb{E}_p\left[\log\left(\frac{p(x)}{q(x)}\right)\right]$ (asymmetric)

Likelihood function of $\boldsymbol{x} \in \mathbb{R}^n$

$L(\theta) = \mathbb{P}[X_1 = x_1, \dots, X_n = x_n] \overset{\text{i.i.d}}{=} p_\theta(x_1)\dots p_\theta(x_n)$

- e.g. $p_{Y|X}(D;\theta) = \prod_i p_{Y_i|X_i}(y_i|\ x_i;\theta)$
- $p(\boldsymbol{x}_{1:n}, \boldsymbol{z}_{1:n}|\theta) = \prod_{i=1}^n p(\boldsymbol{x}_i, \boldsymbol{z}_i|\theta) = \prod_{i=1}^n p(\boldsymbol{x}_i|\boldsymbol{z}_i) \cdot p(\boldsymbol{z}_i)$

Frequentist Paradigm: experiments-based only. Precision of estimator unknown if experiment only performed once!

$$\hat{\theta}_{\text{MLE}} = \arg\max_{\theta\in\Theta} p(\mathcal{D};\theta) \overset{\text{iid}}{=} \arg\max_{\theta\in\Theta} \prod_{i=1}^n p(\boldsymbol{x}_i, y_i;\theta)$$
$$= \arg\min_{\theta\in\Theta} \sum_{i=1}^n -\log \underbrace{p(\boldsymbol{x}_i, y_i;\theta)}_{=p(y_i|\boldsymbol{x}_i;\theta)\cancel{p(\boldsymbol{x}_i)}}$$
$$= \arg\min_{\theta\in\Theta} \sum_{i=1}^n -\log p(y_i|\boldsymbol{x}_i;\theta)$$

Bayesian Paradigm: with "prior belief"

$$\underbrace{p(\theta \mid \mathcal{D})}_{\text{posterior belief}} := \underbrace{\frac{p(\mathcal{D} \mid \theta)}{p(\mathcal{D})}}_{\text{update}} \underbrace{p(\theta)}_{\substack{\text{prior}\\ \text{belief}\\ (\exists\text{bias})}}$$

- ① $p(\mathcal{D}|\theta) \overset{\text{def}}{:=} \prod_i p(X_i = \dots \mid \theta)$
- ② $p(\mathcal{D}) \overset{\text{def}}{:=} \int p(\mathcal{D}|\theta)p(\theta)d\theta$ (e.g. $\int_0^1 d\theta$ for Bernoulli)
- posterior distribution $\mathbb{P}_{\theta^*|\mathcal{D}}$ computationally hard to compute!

$$\hat{\theta}_{\text{MAP}} = \arg\max_{\theta\in\Theta} \underbrace{p(\theta|\mathcal{D})}_{p(\mathcal{D}|\theta)p(\theta)} \overset{\text{iid}}{=} \arg\max_{\theta\in\Theta} \left(\prod_{i=1}^n p(\boldsymbol{x}, y_i|\theta)\right) \cdot p(\theta)$$
$$= \arg\min_{\theta\in\Theta} \sum_{i=1}^n -\log \underbrace{p(\boldsymbol{x}, y_i|\theta)}_{p(y_i|\boldsymbol{x}_i,\theta)\cancel{p(\boldsymbol{x}_i|\theta)}} - \log(p(\theta))$$
$$= \arg\min_{\theta\in\Theta} \sum_{i=1}^n -\log p(y_i|\boldsymbol{x}_i, \theta) - \log(p(\theta))$$

Equivalence: MAP with uniform prior coincides with MLE

$$\hat{\theta}_{\text{MAP}} = \arg\max_{\theta\in\Theta} p(\mathcal{D}|\theta)\cancel{p(\theta)} \overset{!}{=} \arg\max_{\theta\in\Theta} p(\mathcal{D};\theta) = \hat{\theta}_{\text{MLE}}$$

Bayes Optimal Predictor: optimal when knowing $\mathbb{P}_{Y|X}$

$$\underbrace{f^\star}_{\hat{f}}(x) \overset{\text{def}}{:=} \arg\min_{a\in\mathcal{Y}} \underbrace{\mathbb{E}}_{\hat{\mathbb{E}}}[\ell(a, Y)|\ X = x]$$
$$= \arg\min_{a\in\mathcal{Y}} \int \underbrace{p(y|x)}_{\hat{p}} \cdot \ell(a, y)dy$$

In practice when $\mathbb{P}_{Y|X}$ unknown due to finite dataset, one estimate it by replacing with $\hat{f}$, and $\hat{p}(y|x)$ is obtained from $\hat{\mathbb{P}}_{Y|X}$ using prob. modeling

Gaussian Mixture Model

Weights

$$\sum_{i=1}^K w_i \overset{!}{=} 1$$

Overall probability

$$p(x) = \sum_{j=1}^K w_j \overbrace{\mathcal{N}(x;\mu_j,\Sigma_j)}^{\text{probability of j-th Gaussian}}$$

Hard-EM Initialize the parameter $\theta^{(0)}$. For t = 1,2...

- E-Step: predict most likely class for each data points

$$z_i^{(t)} := \arg\max_z P(z|\ \theta^{(t-1)})P(\boldsymbol{x}_i\ |z, \theta^{(t-1)})$$

- Now data is complete! $D^{(t)} = \left\{\left(\boldsymbol{x}_1, z_1^{(t)}\right), \dots, \left(\boldsymbol{x}_n, z_n^{(t)}\right)\right\}$
  - M-Step: compute closed-form MLE as for Gauss. Bayes classifier

$$\theta^{(t)} = \arg\max_\theta P[D^{(t)}|\ \theta]$$

- i.e. $\boldsymbol{\mu}_j^{(t)} = \frac{1}{n_j}\sum_{i:z_i=j}\boldsymbol{x}_j$

Soft-EM (EM): better deals with overlapping clusters

- Initailize parameters $\mu^{(0)}, \Sigma^{(0)}, \boldsymbol{w}^{(0)}$. While not converged:
- E-Step: prob. sample i-th belongs to Gaussian k-th

$$\gamma_k^{(t)}(\boldsymbol{x}_i) = \frac{w_k^{(t-1)}\overbrace{\mathcal{N}\left(\boldsymbol{x}_i \mid \mu_k^{(t-1)}, \Sigma_k^{(t-1)}\right)}^{\substack{\text{prob. k-th Gaussian}\\ \text{generated i-th sample}}}}{\sum_{i=1}^K w_j^{(t-1)}\mathcal{N}\left(\boldsymbol{x}_i \mid \mu_j^{(t-1)}, \Sigma_j^{(t-1)}\right)}$$

- M-Step: fit clusters to weigh. points (closed form MLE sol)

k-th weight: ave. prob. that a point belongs to Gaussian

$$w_j^{(t)} = \frac{1}{N}\sum_{i=1}^N \gamma_j^{(t)}(\boldsymbol{x}_i)$$

k-th mean: weighted average of all points

$$\mu_j^{(t)} = \frac{\sum_{i=1}^N \gamma_j^{(t)}(\boldsymbol{x}_i)\boldsymbol{x}_i}{\sum_{i=1}^N \gamma_j^{(t)}(\boldsymbol{x}_i)}$$

k-th variance: weighted variance σ^2 of all points

$$\Sigma_j^{(t)} = \frac{\sum_{i=1}^N \gamma_j^{(t)}(\boldsymbol{x}_i)\left(\boldsymbol{x}_i - \mu_j^{(t)}\right)^2}{\sum_{i=1}^N \gamma_j^{(t)}(\boldsymbol{x}_i)}$$

- to avoid degenracy, one can add $\nu^2\boldsymbol{I}$ to the diagonal of MLE in update, equivalent to placing a (conjugate) Wishart prior on the covariance matrix

LLM

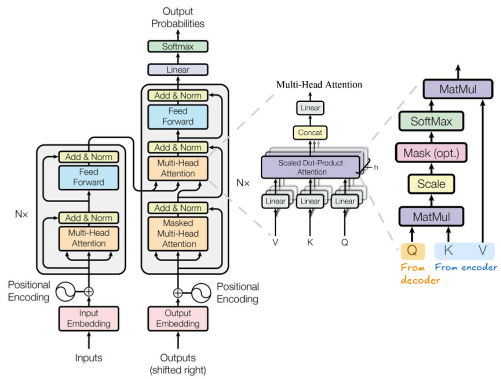
RNN: Recall **feedforward NN** neglecting

①order ②context ⇒ cannot capture sequential data

- **RNN** has recurrent connection, $\text{input}^{(t+1)} := \text{output}^{(t)}$(embed memory, eg. long-short-term memory **LSTM**)
- Sequence-to-Sequence Model Disadvantage: long path of info flow, hard to capture long range dependencies (vanishing gradient)

Transformer process in **parallel** where everything can be passed at once, efficiency allows to train more data (**Note:** but no direct effects on training data set)

- Architecture: also with Residual skip connections; **multi-headed blocks** for diff. meanings



Matrix Calculation of Self-Attention With word embedding matrix $\boldsymbol{X}$, apply *MatMul* with trained weights $(\boldsymbol{W}^{\boldsymbol{Q}}, \boldsymbol{W}^{\boldsymbol{K}}, \boldsymbol{W}^{\boldsymbol{V}})$ and get $(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V})$. Then obtain matrix product Z with: $\text{softmax}\left(\frac{\boldsymbol{Q}\times\boldsymbol{K}^T}{\sqrt{d_k}}\right) \cdot \boldsymbol{V} = \boldsymbol{Z}$

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Positional Encoding: attention neglected orders! Encode with trig. functions; each position/index is mapped to a vector, output of the layer is a matrix with each row as an encoded object

Fine-Tuning: **traditionally** requires gradient updates, **Methods with no grad updates:** Zero-shot predicts the answer given only a natural language description of the task. One-shot a single example of the task provided Few-shot a few examples provided