CS 5489: Machine Learning Cheat Sheet

December 10, 2025

Supervised Learning

Naive Bayes

Generative model: model the prior and class-conditional distribution (each class has a particular distribution of features) by observation / assumption to use MLE on class conditional probability and Bayes decision rule (BDR) to get target probability:

$$p(y|x) = \frac{p(x,y)}{p(x)} = \frac{p(x|y)\cdot p(y)}{p(x)} = \frac{p(x|y)\cdot p(y)}{\sum_y p(x|y)} \quad (1)$$

Naive Bayes: assume features are conditionally independent, i.e. $p(x = (x_1, x_2)|y) = p(x_1|y)p(x_2|y)$.

(class) **Model** $p(y)$. *Bernoulli distribution*: $p(y) = \pi^{\mathbb{1}(y=1)} \cdot (1-\pi)^{\mathbb{1}(y=0)}$.

MLE: $\pi^* = \arg\max \sum_{i=1}^{N} \log p(y_i) \Rightarrow \pi = \frac{\#(y=1)}{\#(y=0)+\#(y=1)}$.

(observation) **Model** $p(x|y)$. ① *Gaussian*: $p(x|y=c) = \frac{1}{\sqrt{2\pi\sigma_c^2}} e^{(-\frac{1}{2\sigma_c^2}(x-\mu_c)^2)}$.

MLE: $\mu_c^* = \frac{1}{N}\sum_{i=1}^{N} x_i, {\sigma_c^*}^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_c^*)^2$. $(x_i|y=c)$

② *Poisson*: $p(x|y=c) = \frac{1}{x!}e^{-\mu_c}{\mu_c}^x, x \in \{0,1,2,\cdots\}, (\mu_c = \bar{x}|y=c)$.

MLE: $\mu_c^* = \frac{1}{N}\sum_{i=1}^{N} x_i$. $(x_i|y=c)$

Laplace smoothing (smoothed MLE): $\pi_j = \frac{N_j+\alpha}{N+2\alpha}$ *to prevent overfitting*.

Naive Bayes Example – SMS spam

Bag-of-Words (BoW) model: 1. build vocabulary $\mathcal{V}$. 2. text t to vector $x \in \mathbb{R}^V$
`sklearn.feature_extraction.text.`
① count occurrence to vectorize: `CountVectorizer(stop_words=xx, max_features=xx)`
② *Term-Frequency (TF):* $x_j = \frac{w_j}{|D|} = \frac{\#\text{ word } j}{\#\text{ words in document}}$.
③ *TF Inverse Document Frequency (TF-IDF):* $x_j = \frac{w_j}{|D|}\log\frac{N}{N_j}$
$IDF(j) = \log\frac{N}{N_j} = \frac{\#\text{ documents}}{\#\text{ documents with word } j}$. `TfidfVectorizer`
④ `HashingVectorizer`

NB Gaussian $p(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} e^{-\frac{1}{2}\|\boldsymbol{x}-\boldsymbol{\mu}\|^2_{\boldsymbol{\Sigma}}}$

MLE: $\boldsymbol{\mu}^* = \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{x}_i$, $\boldsymbol{\Sigma}^* = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{x}_i - \boldsymbol{\mu})(\boldsymbol{x}_i - \boldsymbol{\mu})^T$.

NB Multinomial: $p(\boldsymbol{x}|y) = \frac{(\sum_j x_j)!}{\prod_j x_j!}\left(\prod_j \pi_{j,y}^{x_j}\right)$, where x_j is the frequency of word j ($\sum_j x_j = 1$), $\pi_{j,y}$ is the probability of word w_j in class y ($\sum_{j=1}^{V}\pi_{j,y} = 1$).

L_1 & L_2

L_1 *regularization:* 1. treat all errors equally 2. encourage more sparsity 3. decision is more likely to be aligned with the coordinate axis

L_1 *error:* robust to outlier, no preference to reduce the larger errors.

Generative → Discriminative

Generative: class-conditional distribution (CCD) $\xrightarrow{\text{Bayes rule}}$ classifier $p(y|x)$, can be used to small dataset.
Discriminative: classifier $p(y|x)$ directly

$$\frac{p(y=1|x)}{p(y=2|x)} > 1 \Leftrightarrow \log\frac{p(y=1|x)}{p(y=2|x)} \overset{\text{Bayes}}{=} \log\frac{p(x|y=1)p(y=1)}{p(x|y=2)p(y=2)} > 0 \quad (1)$$

sharing σ^2: $\log\frac{p(x|y=1)}{p(x|y=2)} = \log\frac{\prod_{i=1}^{D}\mathcal{N}(x_i|\mu_i,\sigma^2)}{\prod_{i=1}^{D}\mathcal{N}(x_i|\nu_i,\sigma^2)} = \frac{1}{2\sigma^2}\sum_{i=1}^{D}\left[2(\mu_i - \nu_i)x_i - \mu_i^2 + \nu_i^2\right]$

$$\frac{p(y=1|x)}{p(y=2|x)} > 1 \Leftrightarrow \sum_{i=1}^{D}\underbrace{\frac{1}{\sigma^2}(\mu_i - \nu_i)}_{w_i}x_i + \underbrace{\frac{1}{2\sigma^2}\sum_{i=1}^{D}(\nu_i^2 - \mu_i^2) + \log\frac{\pi_1}{\pi_2}}_{b} > 0 \quad (2)$$

Linear (Binary) Classifier

$f(x) = w^\top x + b$. $f(x) > 0 \rightarrow$ class $1(y=1)$, $f(x) < 0 \rightarrow$ class $2(y=-1)$

Logistic Regression (Binary) Classifier

$f(x) = w^\top x + b, \sigma(z) = \frac{1}{1+e^{-z}}, p(y|x) = \sigma(y\cdot f(x))$:

$$p(y=+1|x) = \sigma(f(x)), \quad p(y=-1|x) = 1-\sigma(f(x)) = \sigma(-f(x)) \quad (3)$$

MLE: $(w^*, b^*) = \arg\max_{w,b}\sum_{i=1}^{N}\log p(y_i|x_i) = \arg\min_{w,b}\sum_{i=1}^{N}\log(1+e^{-y_i f(x_i)})$
$z_i \triangleq y_i f(x_i) > 0 \Rightarrow$ classified correctly, $z_i < 0 \Rightarrow$ wrongly, $z_i = 0 \Rightarrow$ boundary
Logistic regression only forms a linear decision surface.

Logistic loss function: $L(z_i) = \log(1+\exp(-z_i))$. convex → one optimum

Regularization: $(w^*, b^*) = \arg\max_{w,b}\left[\log p(w) + \sum_{i=1}^{N}\log p(y_i|x_i)\right]$.
① Gaussian: $p(w) = \mathcal{N}(0, \frac{C}{2}I) \Rightarrow \log p(w) = -\frac{1}{C}w^T w + \text{constant}$
$\Rightarrow (w^*, b^*) = \arg\max_{w,b}\frac{1}{C}w^T w + \sum_{i=1}^{N}\log(1+\exp(-y_i f(x_i))), \because w^T w = \sum_{j=1}^{d} w_j^2$.
Large $C \rightarrow$ big $w \Leftrightarrow$ Small $C \rightarrow$ small w. Equal to L_1 regularization.

Multiclass logistic regression

$$p(y=c|x) = \text{softmax}(f(x)) = \frac{e^{f_c(x)}}{e^{f_1(x)} + \cdots + e^{f_K(x)}}, f_c(x) = w_c^T x. \quad (4)$$

MLE: ① likelihood: $p(y|x) = \prod_{j=1}^{K} p(y=j|x)^{y_j}$. ② log-likelihood: $\log p(y|x)$.
③ (min) cross-entropy loss: $-\log p(y|x) = -\sum_{j=1}^{K} y_j \log p(y=j|x)$

$$\max_{\{w_j\}}\sum_{i=1}^{N}\log p(y_i|x_i) = \max_{\{w_j\}}\sum_{i=1}^{N}\sum_{j=1}^{K} y_{ij}\log p(y=j|x_i) \quad (5)$$

where $y = [y_1, \cdots, y_K]$ is one-hot vector.

Support Vector Machine (SVM)

Margin distance: $\gamma = \min_i d_i = \min_i \frac{|f(x_i)|}{\|w\|} = \min_i \frac{|w^\top x_i + b|}{\|w\|}$
Support vector: points on $y = w^\top x + b$.
Normalization: $\frac{|aw^T x_i + ab|}{||aw||} = \frac{|w^T x_i + b|}{||w||} \Rightarrow$ just let $f(x_i) = w^T x_i + b = 1$.

Maximize margin: $(\hat{w}, b) = \arg\max_{w,b}\frac{1}{\|w\|}$ s.t. $\min_i |f(x_i)| = 1$
$\Leftrightarrow (\hat{w}, b) = \arg\min_{w,b}\frac{1}{2}\|w\|^2 = \frac{1}{2}w^\top w$ s.t. $y_i f(x_i) \geqslant 1, \forall\, i$. (for binary class)
Lagrangian: $L(x,\lambda) = f(x) - \lambda g(x)$
$\Rightarrow$ *SVM dual*: $\min L(w,\alpha) = \frac{1}{2}w^T w - \sum_i \alpha_i[y_i(w^T x_i + b) - 1], w = \sum_{i=1}^{N}\alpha_i y_i x_i$

$$\arg\max_{\alpha}\sum_i \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j x_i^T x_j \text{ s. t.} \sum_{i=1}^{N}\alpha_i y_i = 0, \quad \alpha_i \geqslant 0 \quad (1)$$

non-separable data: $(\hat{w}, b) = \arg\min_{w,b}\frac{1}{2}w^\top w$, s.t. $y_i f(x_i) \geqslant 1 - \xi_i, \xi_i \geqslant 0, \forall\, i$

soft-SVM: $(\hat{w}, b) = \arg\min_{w,b}\frac{1}{2}w^\top w + \sum_{i=1}^{N}C\xi_i$, s.t. $y_i f(x_i) \geqslant 1 - \xi_i, \xi_i \geqslant 0, \forall\, i$

objective function: $\arg\min_{w,b}\frac{1}{C}w^T w + \underbrace{\sum_{i=1}^{N}\max(0, 1 - y_i f(x_i))}_{\text{hinge loss}}$

prediction: need training data to calculate similarity $f(x) = \sum_i \alpha_i y_i k(x_i, x) + b$.

Tricks

① Normalization for Logistic regression and SVM, due to their sensitivity to absolute value. ② Apply weights to loss of unbiased data. ③ Bigger weights to loss of more important class. ④ Change threshold to one class.

Kernel SVM

idea: learn linear classifier in high-dim space $\phi(x) \in \mathbb{R}^D$ rather than $x \in \mathbb{R}^d$
kernel function: $k(x_i, x_j) = \phi(x_i)^\top\phi(x_j)$, which is $x_i^\top x_j$ in *dual SVM*
example: ① polynomial kernel: $k(x,x') = (x^T x')^p = (\sum_{i=1}^{d} x_i x_i')^p$
② RBF(radial basis function): $k(x,x') = e^{-\gamma\|x-x'\|^2}$. $\gamma\uparrow$: smooth function.
③ Laplacian kernel: $k(x,x') = \exp(-\alpha||x-x'||)$

kernel SVM: $\hat{y} = \text{sign}(\sum_{i=1}^{N}\alpha_i y_i k(x_i, x_*) + b)$, where

$$\alpha = \arg\max_{\alpha}\sum_i \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j k(x_i, x_j) \text{ s. t.} \sum_{i=1}^{N}\alpha_i y_i = 0, \quad \alpha_i \geqslant 0 \quad (2)$$

Kernel matrix: $K = \left[k(x_i, x_j)\right]_{i,j}$, where $k(x_i, x_j) = \phi(x_i)^\top\phi(x_j)$.

K is a positive semi-definite matrix, i.e. $z^T K z \geq 0, \forall z$.
Kernel computation for high dimension is of high cost. Memory usage is based on the number of support vectors kept.

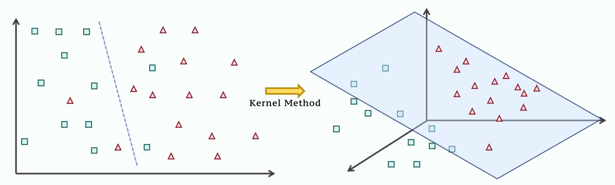


Figure 1: Kernel trick

Regression

Linear Regression. $\hat{y} = w_0 + w_1 x_1 + \ldots + w_d x_d = \boldsymbol{w}^\top \boldsymbol{x} = f(\boldsymbol{x}_i)$, where $\boldsymbol{w} = [w_0, w_1, \ldots, w_d]^\top$, $\boldsymbol{x} = [1, x_1, \ldots, x_d]^\top$
Ordinary Least Squares (OLS). $\min_w \sum_{i=1}^N (y_i - f(\boldsymbol{x}_i))^2 = \min_w \|\boldsymbol{y} - \boldsymbol{X}^\top \boldsymbol{w}\|^2$, where $\boldsymbol{X} = [\boldsymbol{x}_1 \cdots \boldsymbol{x}_N]$ is the data matrix, $\boldsymbol{y} = [y_1, \ldots, y_N]$.
$\Rightarrow \min_w \boldsymbol{y}^T\boldsymbol{y} - 2\boldsymbol{y}^T\boldsymbol{X}^T\boldsymbol{w} + \boldsymbol{w}^T\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{w} \Rightarrow \boldsymbol{w}^* = (\boldsymbol{X}\boldsymbol{X}^\top)^{-1}\boldsymbol{X}$, i.e. *pseudo-inverse*

Ridge Regression. $\min_w \alpha||\boldsymbol{w}||_2^2 + \sum_{i=1}^N (y_i - f(\boldsymbol{x}_i))^2$, where $||\boldsymbol{w}||_2^2 = \sum_{j=0}^d w_j^2$
$\Rightarrow \boldsymbol{w}^* = (\boldsymbol{X}\boldsymbol{X}^T + \alpha\boldsymbol{I})^{-1}\boldsymbol{X}\boldsymbol{y}$. (In OLS, $\boldsymbol{X}\boldsymbol{X}^\top$ may be non-invertible)

LASSO *(Least absolute shrinkage and selection operator).* $\min_w \alpha||\boldsymbol{w}||_1 + \sum_{i=1}^N (y_i - f(\boldsymbol{x}_i))^2$, where $||\boldsymbol{w}||_1 = \sum_{j=0}^d |w_j|$. *LASSO* encourage more sparsity

Sparsity Constraints. $\min_w \sum_{i=1}^N (y_i - f(\boldsymbol{x}_i))^2$, s.t. $||\boldsymbol{w}||_0 \leqslant K$, where $||\boldsymbol{w}||_0 =$ # non-zero entries. nonconvex and NP-hard v.s. *Ridge* and *LASSO* are convex

OMP *(Orthogonal Matching Pursuit).* Iteratively and greedily selects the most related feature to current residual error.
RANSAC *(**RAN**dom **SA**mple **C**onsensus).* Split the data into inliers (good data) and outliers (bad data) and learn the model only from the inliers.
Nonlinear Regression. $\hat{y} = \boldsymbol{w}^\top \phi(\boldsymbol{x})$, where $\boldsymbol{w} = [w_0, w_1, \ldots, w_d]^\top$.

Kernel Ridge Regression. $\min_w \|\boldsymbol{y} - \Phi\boldsymbol{w}\|_2^2 + \lambda\|\boldsymbol{w}\|_2^2$, where $\Phi = [\phi(\boldsymbol{x}_1), \ldots, \phi(\boldsymbol{x}_N)]^\top \in \mathbb{R}^{N\times d}, \boldsymbol{w} \in \mathbb{R}^d \Rightarrow \boldsymbol{w} = (\Phi^\top\Phi + \lambda\boldsymbol{I})^{-1}\Phi^\top\boldsymbol{y} \Rightarrow \hat{y} = \hat{\boldsymbol{k}}^\top(K + \lambda I)^{-1}\boldsymbol{y}$, where $K = [k(\boldsymbol{x}_i, \boldsymbol{x}_j)] \in \mathbb{R}^{N\times N}$, $\hat{\boldsymbol{k}} = [k(\boldsymbol{x}_1, \hat{\boldsymbol{x}}), \ldots, k(\boldsymbol{x}_N, \hat{\boldsymbol{x}})]^\top \in \mathbb{R}^N$.

Gaussian Process Regression (GPR). *Gaussian process:* an sequential random variable set whose any finite subset is joint Gaussian distributed. $f(x) \sim \mathcal{GP}(m(x), k(x, x'))$ where f, m, k are value, mean and covariance function separately. $f_1, \ldots, f_N | x_1, \ldots, x_N \sim \mathcal{N}(0, K)$, where K is the kernel matrix.

GPR: ① observation noise: $y = f + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma^2)$, $p(y|f) = \mathcal{N}(y|f, \sigma^2 I)$ ② function prior: $f \sim \mathcal{GP}(0, k(x, x'))$ ③ train: $p(f|X, y) = \frac{p(y|X,f)p(f|X)}{p(y|X)} = \frac{p(y|f)p(f)}{p(y|X)}$ ④ inference: $p(\hat{f}|\hat{x}, X, y) = \int p(\hat{f}|\hat{x}, f)p(f|X, y)df$. ⑤ prediction: $p(f_*|x_*, \boldsymbol{X}, \boldsymbol{y}) = \mathcal{N}(f_*|\mu_*, \sigma_*^2)$, $\mu_* = \boldsymbol{k}_*^T(\boldsymbol{K} + \sigma^2 I)^{-1}\boldsymbol{y}$, $\sigma_*^2 = k_{**} - \boldsymbol{k}_*^T(\boldsymbol{K} + \sigma^2 I)^{-1}\boldsymbol{k}_*$, where $k_{**} = k(\boldsymbol{x}_*, \boldsymbol{x}_*)$.

Support Vector Regression (SVR). ① objective: $\min\limits_{w,b} \sum\limits_{i=1}^N \left|y_i - (\boldsymbol{w}^\top\boldsymbol{x}_i + b)\right|_\epsilon + \frac{1}{C}||\boldsymbol{w}||^2$, where $|z|_\epsilon = \begin{cases} 0, & |z| \leqslant \epsilon \\ |z| - \epsilon, & |z| > \epsilon \end{cases}$

Regression – Model Ensemble

Idea: combine multiple regression model together to form a better algorithm. ① ***bagging***: train multiple models from random selection of training data. ② ***boosting***: train multiple models which focus on errors made by previous one.

① ***Random Forest Regression.*** Use **bagging** to make an ensemble of *Decision Tree Regressor*. Random subset of data is used to train different tree, whose nodes use different features. The prediction is the average value of each tree. **RFR is sensitive to outliers.**
② ***XGBoost Regression.*** *(e**X**treme **G**radient Boosting)* Use **boosting** to combine a set of less accurate models to create a accurate model. Weak learner fits the gradient of the loss: $h_t(\boldsymbol{x}) \approx \frac{dL}{d\boldsymbol{f}}$, $f_t(\boldsymbol{x}) = f_{t-1}(\boldsymbol{x}) - \alpha_t h_t(\boldsymbol{x}) \approx f_{t-1}(\boldsymbol{x}) - \alpha_t \frac{dL}{d\boldsymbol{f}_{t-1}}$.
Note: ① XGBoost / AdaBoost can run in parallel. ② not necessary to use different datasets to train *NN* when bagging.

Tricks

Improve accuracy: more features + complex features + complex model

Optimization

Adaptive learning rates: AdaGrad, RMSProp, Adam.

Unsupervised Learning

Dimensionality Reduction

Unsupervised learning. only considers the input data, with no output values, trying to discover inherent properties in the data.
Original data point $\boldsymbol{x} = \sum_{j=1}^p w_j\boldsymbol{v}_j \in \mathbb{R}^d$ can be represented as its corresponding weights: $\boldsymbol{w} = [w_1, \cdots, w_P] \in \mathbb{R}^p$.

Dimensionality Reduction. ① pre-process dataset for easier using ② make the results easier to understand ③ reduce computational cost and remove noise.
Goal: Transform high-dim vectors into low-dim vectors, where dimensions in the low-dim data ① represent co-occurring features in high-dim data ② may have semantic meaning.
Note: Not all dimensionality reduction algorithms are unsupervised.
Linear Dimensionality Reduction (LDR). Project the original data onto a lower-dim hyperplane and represent the data with coordinates in it.

LDR - Principal Component Analysis (PCA)

Principal Component Analysis (PCA). ① unsupervised method ② *goal:* preserve the variance of data as much as possible = minimize reconstruction error ③ choose basis vectors along the max variance (longest extent) of data, called principal components (PC).
Alg: ① subtract the mean of the data: $\bar{\boldsymbol{x}}_i = \boldsymbol{x}_i - \mu$ ② data covariance matrix: $\Sigma = \frac{1}{N}\sum_i \bar{\boldsymbol{x}}_i\bar{\boldsymbol{x}}_i^T = \frac{1}{N}\bar{\boldsymbol{X}}\bar{\boldsymbol{X}}^T$ ③ select top K eigenvectors of Σ as PC: $\boldsymbol{V} = [\boldsymbol{v}_1 \cdots \boldsymbol{v}_K]$ ④ project the data onto the PC: $\boldsymbol{w}_i = \boldsymbol{V}^T\boldsymbol{x}_i \in \mathbb{R}^K \to \hat{\boldsymbol{x}}_i = \boldsymbol{V}\boldsymbol{w}_i$.
Explained variance. each PC explains a percentage of the original data called explained variance.
Number of PC: till ① sum(explained variance) $\geqslant$ 95% ② performance is good.
Lack. preserving the variance sometimes won't help for classification: ① PCA doesn't consider which class the data belongs to ② When the "classification" signal is less than the "noise", PCA will make classification more difficult.

Random Projections. ① For k-components PCA in $\mathbb{R}^d$, complexity is $O(dk^2)$ ② generate random basis vectors sampled from a Gaussian ③ reduce computation at the expense of losing some accuracy (adding noise).
Johnson-Lindenstrauss lemma. carefully selecting the distribution of the random projection matrices will preserve the pairwise distances between any two samples of the dataset, within some error epsilon: $(1 - \epsilon)\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2 < \|\boldsymbol{w}_i - \boldsymbol{w}_j\|^2 < (1 + \epsilon)\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2$.

Fisher's Linear Discriminant (FLD) / Linear Discriminant Analysis (LDA). ① supervised method ② *goal:* find a lower-dim space to maximize the class separation: 1. maximize the distance between projected means 2. minimize the projected variances ③ *model:* data from each class is modeled as a Gaussian.

Latent Semantic Analysis (LSA). ① document vector $\boldsymbol{x}_i$ is a BoW representation ② approximate $\boldsymbol{x}$ as a weighted sum of topic vectors $\hat{\boldsymbol{x}} = \sum_{n=1}^p w_p\boldsymbol{v}_p$ ③ *goal:* $\min_{v,w}\sum_i ||\boldsymbol{x}_i - \hat{\boldsymbol{x}}_i||^2$ ④ *Advantage:* distances / similarities compare **topics** rather than words, a higher-level semantic representation. ⑤ *Lack:* frequency of a word can be negative in topic + weights for each topic can be negative.

Non-negative Matrix Factorization (NMF). ① $\min_{v\geqslant0,w\geqslant0}\sum_j ||\boldsymbol{x}_j - \hat{\boldsymbol{x}}_j||^2$ ② *sparse representation:* most topic weights for a document are zero. ③ not a probabilistic model and difficult to interpret.
Note: 1. linear Kernel *PCA* can reduce memory and computation ($\to O(d^2k)$) 2. *PCA* and *LSA* have closed-form solutions.

Non-Linear Dimensionality Reduction

Linear DR: model the data as "living" on a linear manifold (PCA, NMF, LSA)

Kernel PCA. ① run PCA on high-dim feature transformation $\phi(\boldsymbol{x}_i)$ ② *principal component:* $\boldsymbol{v} = \sum_{i=1}^n a_i\phi(\boldsymbol{x}_i)$ where a_i are learned weights ③ coefficient of new point $\boldsymbol{x}_*$ for $\boldsymbol{v}$: $w = \phi(\boldsymbol{x}_*)^T\boldsymbol{v} = \sum_{i=1}^n a_i\phi(\boldsymbol{x}_*)^T\phi(\boldsymbol{x}_i) = \sum_{i=1}^n a_i k(\boldsymbol{x}_*, \boldsymbol{x}_i) = \boldsymbol{k}_*^T\boldsymbol{a}$ ④ *learn weights:* center kernel $\tilde{\boldsymbol{K}} = (\boldsymbol{I} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T)\boldsymbol{K}(\boldsymbol{I} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T)$ where $\boldsymbol{K} = [k(\boldsymbol{x}_i, \boldsymbol{x}_j)]_{ij} \to$ select top-K eigenvalues and scale $\frac{1}{\sqrt{\lambda_j}}\boldsymbol{a}_j \to$ project $w_j = \boldsymbol{k}_*^T\boldsymbol{a}_j$.

Clustering – Parametric clustering

Data is set of vectors $\{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n\}$, where each data point is a vector $\boldsymbol{x} \in \mathbb{R}^d$.
Clustering. ① *Goal:* group similar data together ② *Cluster:* groups are called **clusters**, each data point is assigned with a cluster index ($\{1, \cdots, K\}$).

K-Means. ① *Idea:* each cluster is represented by a cluster center $\boldsymbol{c}_j \in \mathbb{R}^d$, $j \in \{1, \cdots, K\} \to$ assign each data point to the closest center: $(\boldsymbol{x}_i, z_i)$, $z_i \in \{1, \cdots, K\}$. ② *Objective:* minimize the total sum-squared difference between points and their centers: $\min\sum_{i=1}^n ||\boldsymbol{x}_i - \boldsymbol{c}_{z_i}||^2$. ③ *Method 1:* assume the assignments z_i are known ($C_j = \{\boldsymbol{x}_i | z_i = j\}$) $\to$ pick the cluster centers $\boldsymbol{c}_j = \text{argmin}\sum i \in C_j||\boldsymbol{x}_i - \boldsymbol{c}_j||^2 = \frac{1}{|C_j|}\sum \boldsymbol{x}_i \in C_j\boldsymbol{x}_i$. ④ *Method 2:* assume the clusters $\{\boldsymbol{c}_1, \cdots, \boldsymbol{c}_K\}$ are known $\to$ pick the assignments $z_i = \arg\min\limits_{j\in\{1,\cdots K\}} ||\boldsymbol{x}_i - \boldsymbol{c}_j||^2$. ⑤ *Note:* result depends on initialization $\to$ random initializations & k-means++. ⑥ *Problem:* assume each cluster has a circular shape (Euclidean distance $\|\cdot\|$), thus can't handle skewed (elliptical) clusters. ⑦ *Note:* always converges.

Bag-of-X Representation. ① *Goal:* create a new quantized representation for samples. ② get k cluster center in many examples (vocabulary) $\to$ assign each part in samples a center $\to$ count the number of each center $\to$ k-dim vector.

Gaussian mixture model (GMM). ① use multivariate Gaussian model a cluster with an elliptical shape, where covariance matrix controls elliptical shape and mean controls location. ② *GMM:* weighted sum of Gaussian $p(\boldsymbol{x}) = \sum_{j=1}^K \pi_j N(\boldsymbol{x}|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$, where π_j reflects how likely is the j-th cluster / Gaussian. ③ *MLE:* $\max_{\pi,\boldsymbol{\mu},\boldsymbol{\Sigma}}\sum_{i=1}^N \log\sum_{j=1}^K \pi_j N(\boldsymbol{x}_i|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$. $\leftarrow$ hard to optimize $\to$ *EM* alg. ④ *Note:* use Mahalanobis distance; handle ellipse clusters

Expectation-Maximization (EM) algorithm. ① *Goal:* find the *MLE* solution $\hat{\theta} = \text{argmax}_\theta \log p_\theta(\boldsymbol{x})$ when there are hidden variables z, where $p_\theta(\boldsymbol{x}) = \sum_z p(\boldsymbol{x}|z)p(z)$. ② *Solution:* iterate between estimating the hidden variables z and maximizing w.r.t the parameters θ. 1. *initialize* $\to$ 2. *E-step*: estimate the hidden variables as their expected value $\mathcal{Q}(\theta) \leftarrow \mathbb{E}_{z|\boldsymbol{x},\hat{\theta}}[\log p_\theta(\boldsymbol{x}, z)] \to$ 3. *M-step:* $\hat{\theta} \leftarrow \text{argmax}_\theta\,\mathcal{Q}(\theta) \to$ 4. repeat until convergence.

EM alg. for GMM. ① *E-step:* soft assignment of point i to cluster j: $\hat{z}_{ij} = p(z_i = j|\boldsymbol{x}_i) = \frac{\pi_j\mathcal{N}(\boldsymbol{x}_i|\mu_j,\Sigma_j)}{\sum_k \pi_k\mathcal{N}(\boldsymbol{x}_i|\mu_k,\Sigma_k)}$ ② *M-step:* soft count $N_j = \sum_{i=1}^N \hat{z}_{ij} \to$ weight $\pi_j = N_j/N$ $\to$ mean $\boldsymbol{\mu}_j = \frac{1}{N_j}\sum_{i=1}^N \hat{z}_{ij}\boldsymbol{x}_i \to$ variance $\Sigma_j = \frac{1}{N_j}\sum_{i=1}^N \hat{z}_{ij}(\boldsymbol{x}_i - \boldsymbol{\mu}_j)^2$.

Bayesian GMM. ① *Goal:* automatically selects K ② use Dirichlet distribution to model prior $p(\pi)$, mathematically $\pi_j = \frac{N_j+\alpha}{N+K\alpha}$ (similar to Laplace smoothing).
Clustering using KDE. ① automatically select cluster centers ② *Mode:* local maximum of probability density $\boldsymbol{\mu} = \text{argmax}_x\,p(\boldsymbol{x})$, considered as centers. ③ *Find:* $\hat{\boldsymbol{\mu}} \leftarrow \hat{\boldsymbol{\mu}} + \eta\frac{d}{dx}p(\boldsymbol{x})$, convergence guaranteed by "Mean-shift algorithm".

Comparison of GMM and K-means. ① *Similarity:* 1. use iterative algorithms where first step assigns points, and the second step recalculates the cluster means 2. assume compact clusters (elliptical or circular shaped) 3. sensitive to initialization 4. can't decide clusters number K by itself ② *Difference:* 1. K-means uses hard-assignment (0/1) while *GMM* uses soft-assignments ($0 \sim 1$) 2. K-means assumes circular shape while *GMM* elliptical 3. *GMM* factor in the cluster size but K-means not.

Non-parametric clustering – Mean-shift algorithm

①*Idea:* iteratively shift towards the largest concentration of points ② *Alg.* start from an initial point $\boldsymbol{x}$ → find the neighbors to $\boldsymbol{x}$ within some radius (bandwidth) → set $\boldsymbol{x}$ as the mean of the neighbors points → repeat till convergence. ***Getting Clusters.*** ① Run for many initial points $\{x_i\}$ → converged points are centers, points that converge to the same center belong to the same cluster. ② *Number of clusters:* implicitly controlled by the bandwidth, where larger bandwidth creates less clusters.

Non-parametric clustering – Spectral Clustering

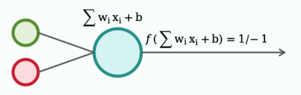
Non-compact clusters. K-means, GMM, and Mean-Shift assume that all clusters are compact, i.e. circles or ellipses, thus can't handle non-compact clusters. ***Spectral Clustering.*** ① *Idea:* 1. estimate clusters using pair-wise *affinity* between points 2. clustering with a graph formulation 3. cut edges that are small ② *Affinity / Similarity:* use kernel function $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, like RBF kernel ③ *Graph formulation:* node is data point, edge weight is affinity $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. ④ *Sensitivity to gamma:* small γ in RBF → far away points are still considered similar. ⑤ *Advantage:* can handle non-compact shape clusters / on a manifold which can form a connected graph.

Name	Cluster Shape	Principle	Advantages	Disadvantages
K-Means	circular	minimize distance to center	scalable (MiniBatchKMeans)	sensitive to initialization; local minima. must choose K.
GMM	elliptical	maximum likelihood	supports elliptical shapes	sensitive to initialization; local minima. must choose K.
Bayesian GMM	elliptical	maximum marginal likelihood	automatically selects K via concentration parameter	slow; sensitive to initialization.
Mean-Shift	compact	move toward local mean	automatically selects K via bandwidth	can be slow.
Spectral Clustering	irregular	graph-based	handles any connected cluster shape	must choose K; cannot assign new points; kernel matrix expensive.

Deep Learning

Multi-layer Perceptron

Perceptron. Model a single neuron $y = f(w^T x) = f(\sum_{j=0}^{d} w_j x_j) = \begin{cases} 1 & \text{if } w^\top x \geq 0 \\ -1 & \text{otherwise} \end{cases}$, *loss:* $E(w) = \sum_{i=1}^{N} L(z_i) = \sum_{i=1}^{N} \max(0, -z_i), z_i = y_i w^\top x_i$.



Perceptron Alg. / SGD. $w \leftarrow w + \eta y_i x_i = w - \eta g_w(x_i)$, where x_i is misclassified *Result (1st in ML)*: 1. only converge on linearly separable data; 2. # iteration $\sim \frac{1}{m}$, where m is the separation (margin) between classes.
Multi-layer Perceptron now called *neural network*.

Multi-class Logistic regression. class labels $y \in \{1, \cdots, C\}$, class vector $\boldsymbol{y} \in \mathbb{R}^C$ is one-hot vector. Output $p(y = j|\boldsymbol{x}) = f_j(\boldsymbol{x}) = s_j(\boldsymbol{g}(\boldsymbol{x})) = \frac{\exp(g_j(\boldsymbol{x}))}{\sum_{k=1}^{C} \exp(g_k(\boldsymbol{x}))}$, $g_j(\boldsymbol{x}) = \boldsymbol{w}_j^T \boldsymbol{x}$, for $j \in \{1, \cdots, C\}$.
MLE: $\log p(\boldsymbol{y}|\boldsymbol{x}) = \log \prod_{j=1}^{C} f_j(\boldsymbol{x})^{y_j} = \sum_{j=1}^{C} y_j \log f_j(\boldsymbol{x}) = \boldsymbol{y}^T \log \boldsymbol{f}(\boldsymbol{x})$
$\boldsymbol{W}^* = \arg\max_{\boldsymbol{W}} \sum_{i=1}^{N} \log p(\boldsymbol{y}_i|\boldsymbol{x}_i) = \arg\max_{\boldsymbol{W}} \sum_{i=1}^{N} \boldsymbol{y}_i^T \log \boldsymbol{f}(\boldsymbol{x}_i) = \arg\min_{\boldsymbol{W}} \sum_{i=1}^{N} \{-\sum_{j=1}^{C} y_{ij} \log f_j(\boldsymbol{x}_i)\}$.
cross-entropy loss: $L(\boldsymbol{y}, \boldsymbol{f}) = -\sum_{j=1}^{C} y_j \log f_j(\boldsymbol{x})$.
chain rule: $\frac{dL}{dw_j} = \frac{dL}{dg} \frac{dg^T}{dw_j} = \frac{dL}{df} \frac{df}{dg} \frac{dg^T}{dw_j} = \boldsymbol{x}(f_j(\boldsymbol{x}) - y_j)$.

Multi-layer Perceptron(MLP). Add hidden layers between inputs and outputs. $\boldsymbol{h} = f(\boldsymbol{W}^T \boldsymbol{x})$, where $f(\cdot)$ is the *activation function*.

Activation function: ① Sigmoid: $\mathbb{R} \mapsto [0, 1], f(x) = \frac{1}{1+e^{-x}}$ ② Tanh: $\mathbb{R} \mapsto [-1, 1], f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ ③ Rectifier Linear Unit (ReLU): $f(x) = \max(0, x)$. ④ *Note:* ReLU yield sparse representations and train fast.

MLP - Part 2

Overfitting: the training loss decreases, but the validation loss increases.
Early stopping: stop training when the validation loss is stable (change below a threshold) for a number of iterations to prevent overfitting.

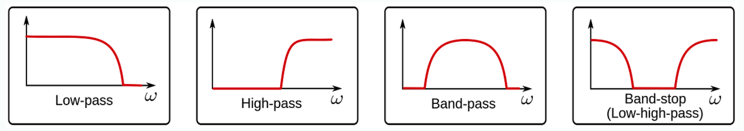
Universal Approximation Theorem: A *MLP* with one hidden layer and finite nodes can approximate any continuous function up to a desired error.
Vanishing Gradient problem: ① *Meaning:* Back-prop recursively multiplies gradients causing numerical values get smaller. ② *Reason:* 1. successive multiplications of small gradients + summation of large numbers of gradient paths for deeper network 2. activation functions that saturate at some points.

Convolutional Neural Networks (CNNs)

Idea: Signal's features may have **locality** and **translation**, which can not be dealt with by *MLP*. To deal with it, use **local feature extractor** in the signal.

1-D discrete convolution: $s(t) = x * w = \sum_a x(a) w(t - a)$.
2-D discrete convolution: $s(n, m) = x * w = \sum_a \sum_b x(a, b) w(n - a, m - b)$.
Property: $s = x * w \Rightarrow s(t - a) = x(t - a) * w(t)$. *Note:* convolution operation is equivalent to multiplication in the frequency domain.

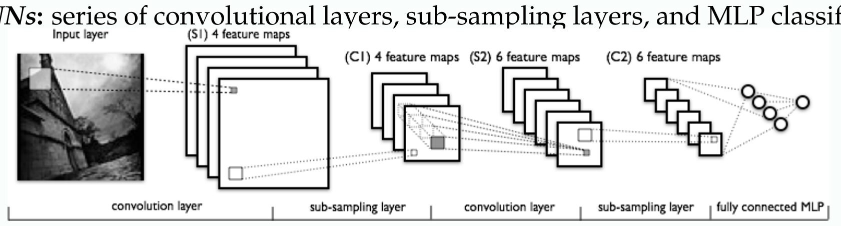
Interpretation 1: w is a filter on the frequency spectrum of signal x, e.g. *low-pass, high-pass, band-pass, low-high-pass, moving-average.*



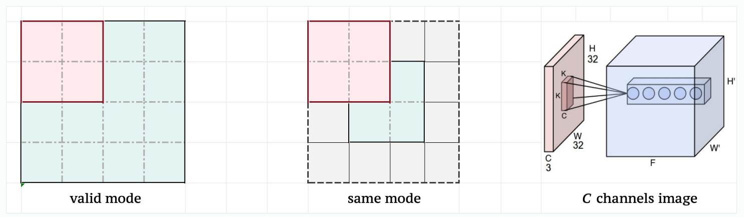
① transform to frequency domain $X(w)$ by *DFT* $X(\omega) = \sum_t x(t) e^{-i\omega t}$ where $e^{-i\omega t} = \cos(\omega t) - i \sin(\omega t)$ ② Do filtering by w ③ Back to time domain $x(t) = \frac{1}{2\pi} \int X(\omega) e^{i\omega t} d\omega$. Then $s(t) = x(t) * w(t) \iff S(\omega) = X(\omega) W(\omega)$.

Interpretation 2: w is a pattern (template) and try to find this pattern i.e. pattern w matches the local $x \Rightarrow x = \frac{w}{||w||}$.

CNNs: series of convolutional layers, sub-sampling layers, and MLP classifier.



2D convolution filter: $h = f(\sum_{x,y} W_{x,y} P_{x,y})$, where W: template, P: part of image
Convolution modes: ① *valid mode*: all $P_{x,y}$ has valid image values → shrinks by $(k - 1)/2$ ② *same mode*: zero-pad the border of the image → same size as the input image.



C channels image: $C \times H \times W \xrightarrow[C \times F \times K \times K]{F \text{ convolution filter}}$ F channels feature: $F \times H' \times W'$

Spatial sub-sampling: ① *stride* (s) → $(m - k_x + 1 - s) \times (n - k_y + 1 - s)$ ② *max-pooling* → $W_{x,y} P_{x,y} = \max(P_{x,y})$, introduce translation invariance (robust) ③ *Effect:* reduce spatial resolution of feature maps thus faster inference speed.
Receptive field: what pixels in the input affect a particular node → $k_{i-1} + k_i - 1$
Advantages: ① extract the same features throughout the image ② pooling is robust to changes ③ less parameters than *FCNN*: $C \times H \times W \to F \times H \times W$, *FCNN* is $(CHW + 1) \times (FHW)$ and *CNN* is $(CKK + 1) \times F$

Regularization Methods

L2 regularization: $\hat{L} = L + \lambda \frac{1}{2} ||\boldsymbol{w}||^2$. *SGD:* $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \eta \frac{d\hat{L}}{d\boldsymbol{w}_k} = \boldsymbol{w}_k - \eta(\frac{dL}{d\boldsymbol{w}_k} + \lambda \boldsymbol{w}_k) = (1 - \eta\lambda)\boldsymbol{w}_k - \eta \frac{dL}{d\boldsymbol{w}_k}$, which is also called ***weight-decay***.
*Note:*① *L2 regularization should be applied to all layers.* Because when applying L2 regularization on only one layer, the other layer can be adjusted to get the same network: $f(x) = \boldsymbol{A}^T r(\boldsymbol{B}^T \boldsymbol{x}) = \boldsymbol{A}^T r(\frac{\epsilon}{\epsilon} \boldsymbol{B}^T \boldsymbol{x}) = \frac{1}{\epsilon} \boldsymbol{A}^T r(\epsilon \boldsymbol{B}^T \boldsymbol{x})$. ② equivalent to assuming a Gaussian prior on the weights and using MAP estimation.

Early Stopping: limit the number of training iterations (how far $\boldsymbol{w}$ can move)
Bagging: combine predictions from several models (model averaging).
Combine K models whose error is ϵ_i and $\text{MSE}_i = \mathbb{E}[\epsilon_i^2] = V$, then $\text{MSE}_c = \frac{V}{K} + \frac{K-1}{K} C$, where $C = \mathbb{E}[\epsilon_i \epsilon_j]$ is the correlation between errors of the i-th and j-th models. So MSE decreases when $C = 0$.

For *NN*, can use same dataset, but different initializations / hyper-parameters.
Dropout: ① *Goal:* approximate a large ensemble of models by randomly removing an input/hidden node with some probability. ② *Training:* (1) "drop out" each node with probability p (2) train a reduced network in each iteration ③ *Testing:* use all the nodes for prediction and scale the output by $1 - p$ ④ *Effect:* (1) block some paths when computing gradients → so reduce vanishing gradient problem (2) reduces prediction error (3) reduces the effective capacity of the network (4) increases classifier robustness (not just rely on one feature).

Data augmentation: artificially permute the data, using translation, flip, rotation, deformation, add pixel noise, scale, color, brightness, contrast, etc. to make 1. the dataset larger / model sees more variations of the data, 2. the network invariant to the permutations.

How to Go Deeper?

How many samples do we need? Suppose $\mathcal{N}$ is a feed-forward network with W weights, L layers, with a fixed piecewise activation function. Then $VCdim(\mathcal{N}) = O(WL \log W + WL^2)$. (Bartlett, Maiorov, Meir, 1998)
VC dimension is a statistical measure of function class capacity, and sample size needs to scale proportionally to. With same number of parameters, the deeper network requires more data (WL^2).

Rectified Linear Unit. $\text{ReLU}(z) = \max(0, z)$. *Advantage:* ① Easy and fast to train. ② Sparse representation, i.e. most nodes will output zero, reduce the vanishing gradient problem.

Batch Normalization. ① *Why?* Distribution of activations changes in 1 layer, then all layers are affected and need to change. ② *Batch-norm*: For each node in each layer, normalize outputs to $\hat{y}_i = \frac{y_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$ over each mini-batch → scale-shift to $z_i = \gamma \hat{y}_i + \beta$, where γ, β are learnable. ③ *Effects*: reduce the effect of covariate shift → accelerate training + stable gradients + better generalization (no need for dropout / L_2-reg). ④ *Note:* better to put after a linear layer.

Learning Rate Schedule. ① *Linear change*: $\eta_k = (1 - \frac{k}{T})\eta_0 + \frac{k}{T}\eta_T$. ② *Decay*: $\eta_k = \frac{1}{1+\delta k}\eta_0$, where $0 < \delta < 1$. ③ *Adaptive schedule*: reduce the *lr* when the validation loss no longer improves.

Momentum. keep a running average of the gradients across mini-batches: $\boldsymbol{v}^{(t)} = \alpha \boldsymbol{v}^{(t-1)} - \eta \frac{dL}{d\boldsymbol{w}}, \boldsymbol{w}^{(t)} \leftarrow \boldsymbol{w}^{(t-1)} + \boldsymbol{v}^{(t)}$.

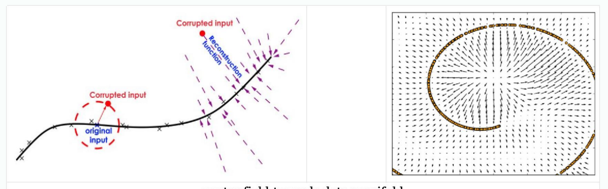
Transformer Learning & Fine-tuning

Transformer Learning. Rather than retrain the whole pre-trainied network, ① fix the lower layers that extract features ② train the last few layers to perform the new task. `*list(model.children())[:-1]`

Auto-Encoders

Auto-Encoder (AE). ① *Goal:* use the hidden layer to learn the lower-dimensional latent representation of the data ② *Method:* encoder $\boldsymbol{z} = f(\boldsymbol{x}) = h_1(\boldsymbol{A}^T\boldsymbol{x}) \rightarrow$ decoder $\hat{\boldsymbol{x}} = g(\boldsymbol{z}) = g(f(\boldsymbol{x})) = h_2(\boldsymbol{B}^T\boldsymbol{z})$ ③ *Target:* minimize the difference between $\boldsymbol{x}$ and $\hat{\boldsymbol{x}}$: 1. cross-entropy: $\boldsymbol{x}\log\hat{\boldsymbol{x}} + (1-\boldsymbol{x})\log(1-\hat{\boldsymbol{x}})$ 2. MSE: $||\boldsymbol{x} - \hat{\boldsymbol{x}}||^2$ ④ *Weight Sharing:* decoder's weights are the transpose of encoder's weights, $\boldsymbol{A}^T = \boldsymbol{B}$. ⑤ *Note:* if h_1, h_2 are linear then it's similar to PCA ⑥ *Better:* 1. regularization like $L = ||\boldsymbol{x} - g(f(\boldsymbol{x}))||^2 + \lambda||f(\boldsymbol{x})||_1$ 2. denoising.

Denoising Auto-Encoder (DAE). ① *Problem of AE:* minimum occurs when $g(f(\boldsymbol{x})) = \boldsymbol{x}$ means learn nothing ②*Idea:* add noise to corrupt the inputs to make encoder learn more about the data manifold: minimize $||\boldsymbol{x} - g(f(\tilde{\boldsymbol{x}}))||^2$ ③ *Interpretation (1):* DAE is learning a **vector field** from the corrupted $\tilde{\boldsymbol{x}}$ to the data manifold (clean $\boldsymbol{x}$) ③ *Interpretation (2):* learn a reconstruction distribution $p_{\text{recon}}(\boldsymbol{x}|\tilde{\boldsymbol{x}})$: sample training example $\boldsymbol{x} \rightarrow$ add noise $\tilde{\boldsymbol{x}} \rightarrow$ maximize $p_{\text{recon}}(\boldsymbol{x}|\tilde{\boldsymbol{x}}) = p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{z}), \boldsymbol{z} = f(\tilde{\boldsymbol{x}}) \rightarrow$ *MLE:* $\min \sum_i ||\boldsymbol{x}_i - g(f(\tilde{\boldsymbol{x}}_i))||^2$ (assume $p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{z})$ is Gaussian)



vector field towards data manifold

Convolutional Auto-Encoder. encoder - a standard *CNN* w/o classifier → decoder - the opposite architecture, replace max-pooling with upsampling.

Deep Generative Models

For data-label pair (x, z), forward generative process: $p(x|z)$ and discriminative classifier: $p(z|x)$. We want *NN* to be generative model (multiple outputs). ① *Idea:* use *NN* to predict the parameters of a distribution, e.g. $y \sim \mathcal{N}(\mu(\boldsymbol{x}), \sigma(\boldsymbol{x})^2)$ ② *Reparameterization:* decompose y into parameters and "randomness", e.g. $y = \mu(\boldsymbol{x}) + \sigma(\boldsymbol{x})z, z \sim \mathcal{N}(0,1)$. In general $y \sim p(y|x) \Rightarrow y = f(z, x), z \sim p(z)$ where x are the inputs, z is the source of randomness, f transforms samples of z to samples of y which modeled as a *NN* and learned. ③ *Note:* Computing $p(y)$ requires computing the inverse of the transformation f, learning f is equal to learn the inverse CDF of y.

Variational AutoEncoder (VAE). ① *Problem of AE:* decoder never sees encoded latent vectors outside of the training set thus hard to decode new images. ② *Idea:* introduce noise in the latent vectors ③ *Model:* let latent variable $\boldsymbol{z}$, observation $\boldsymbol{x}$, prior $p(\boldsymbol{z}) \sim \mathcal{N}(0, \boldsymbol{I})$, likelihood $p(\boldsymbol{x}|\boldsymbol{z})$, loss function $-\log p(\boldsymbol{x}|\boldsymbol{z})$; given $\boldsymbol{x}_i$ to infer (posterior distribution) $\boldsymbol{z}_i$: $q(\boldsymbol{z}_i|\boldsymbol{x}_i) \sim \mathcal{N}(\mu(\boldsymbol{x}_i), \text{diag}(\sigma(\boldsymbol{x}_i)^2))$ ($\mu(\cdot), \sigma(\cdot)$ are *NN* outputs) ④ *Reparameterization:* $\boldsymbol{z}_{il} = \mu(\boldsymbol{x}_i) + \sigma(\boldsymbol{x}_i) \circ \epsilon_{il}$ where $\epsilon_{il} \sim \mathcal{N}(0, \boldsymbol{I})$ ⑤ *Note:* decoder and encoder don't share weights ⑥ *VAE loss:*

$$L(\boldsymbol{x}_i) = \underbrace{-\frac{1}{L}\sum_{l=1}^{L}\log p(\underbrace{\boldsymbol{x}_i|\boldsymbol{z}_{il}}_{\text{decoder}})}_{\text{reconstruction loss}} + \underbrace{KL(q(\boldsymbol{z}|\boldsymbol{x}_i), p(\boldsymbol{z}_i))}_{\text{regularizer}},$$

where $\boldsymbol{z}_{il} = \mu(\boldsymbol{x}_i) + \sigma(\boldsymbol{x}_i) \circ \epsilon_{il}$, $\epsilon_{il} \sim \mathcal{N}(0, \boldsymbol{I})$(encoder).
Convolutional VAE. replace the Dense layers with Conv2D and Pooling.
Comparison with AE. ① *Similarities:* 1. aim to encode the input into a latent space, and then decode it to reconstruct the original input 2. use reconstruction loss (MSE, cross-entropy) for training. ② *Differences:* 1. *AE* predicts a single latent vector z, while *VAE* predicts a Gaussian posterior distribution of the latent vector z 2. For training, *VAE* also has a regularization term that keeps the posterior near to the prior distribution 3. *VAE* samples the latent vector z during training, according to the posterior. ③ *Advantage:* *VAE* samples from the latent space, it sees more latent vectors during training, which prevents the model from overfitting to each latent vector. This helps interpolation between the latent vectors, creating a smoother output space.

Deep Generative Models – Generative Adversarial Networks

① *Goal:* generate samples from a probability density $p_{data}(x)$ which is represented by a training set $\{x_i\}$. ② *Setup:* true data samples: $x \sim p_d(x)$ but only have samples $\mathcal{X} = \{x_i\}$; generated samples: $x \sim p_m(x)$ ③ *Model:* reparam $x = g(z), z \sim N(0, I)$ where $g(\cdot)$ is a *NN* → max $p_m(x)$ ④ *Problem:* cannot explicitly compute $p_m(x)$ due to *NN* ⑤ *Solution:* look at the samples directly → build a classifier to predict whether a sample is *real* or *fake* (generated).

GAN. ① *Architecture:* *Generator (G)* - make fake samples to fool the discriminator ↔ *Discriminator (D)* - classify x as real or fake ② *Model:* define "real" ($y = 1$) and "fake" ($y = 0$), discriminator (classifier) $D(x) = p(y = 1|x)$ ③ *Train:* D: $\min_D -\sum_i y_i \log D(x_i) + (1-y_i)\log(1-D(x_i)) = \max_D[\mathbb{E}_{x\sim p_d}[\log D(x_i)] + \mathbb{E}_{z\sim\mathcal{N}(0,I)}[\log(1-D(G(z)))]]$; G to fool D → final training problem: $\min_G \max_D \mathbb{E}_{x\sim p_d}[\log D(x_i)] + \mathbb{E}_{z\sim\mathcal{N}(0,I)}[\log(1-D(G(z))]$. ④ *Note:* 1. *GANs* don't provide an explicit likelihood estimation, but *VAEs* do. 2. training *GAN* is hard due to its adversarial nature.

Analysis. ① *Optimal D:* $D^*(x) = \frac{p_d(x)}{p_d(x)+p_m(x)} = p(\text{real}|x)$ ② *Optimal G:* $p_m^*(x) = p_d(x)$ ③ *Advantage:* don't need to approximate the likelihood function; don't need the partition function; easy to generate samples ④ *Lack:* cannot compute the likelihood value (probability density) of a sample; hard to train ④ *Mode collapse problem:* when $p_d(x)$ have more than 2 modes, G samples from one mode → D catches this mode → G switches to other mode → $\cdots$.
Application: Style transfer (CycleGAN), Face Synthesis . . ..

Diffusion Model. ① *Goal:* define forward diffusion process gradually turns image into noise → learn reverse process gradually remove noise and recover image ② *Forward diffusion process:* $\boldsymbol{x_t} = \sqrt{1-\beta_t}\boldsymbol{x}_{t-1} + \sqrt{\beta_t}\epsilon_t$, $\epsilon_t \sim \mathcal{N}(0, \boldsymbol{I})$ where $\boldsymbol{x_0}$ = image and $\{\beta_1, \cdots, \beta_T\}$ is *noise variance schedule,* $q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}) = \mathcal{N}(\boldsymbol{x}_t|\sqrt{1-\beta_t}\boldsymbol{x}_{t-1}, \beta_t\boldsymbol{I}) \rightarrow q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0) = \prod_{t=1}^{T} q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$ ③ *Reverse diffusion process:* $p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) = \mathcal{N}(\boldsymbol{x}_{t-1}|\boldsymbol{\mu}_\theta(\boldsymbol{x}_t, t), \Sigma_\theta(\boldsymbol{x}_t, t)) \rightarrow p(\boldsymbol{x}_{0:T}) = p_\theta(\boldsymbol{x}_T)\prod_{t=1}^{T} p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$ ④ *Train:* $\mathbb{E}[-\log p_\theta(\boldsymbol{x}_0)] \leqslant \mathbb{E}_q\left[-\log\frac{p_\theta(\boldsymbol{x}_{0:T})}{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)}\right] = \sum_{t=0}^{T} L_t \rightarrow \min\sum_{t=0}^{T} L_t$ where $L_0 = -\log p_\theta(\boldsymbol{x}_0|\boldsymbol{x}_1), L_{t-1} = KL(q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0), p(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)), L_T = KL(q(\boldsymbol{x}_T|\boldsymbol{x}_0, p(\boldsymbol{x}_T))$.

Model	Description
Autoencoder	Unsupervised dimensionality reduction and clustering.
Convolutional Autoencoder	AE for images.
Masked Autoencoder (MAE)	Self-supervised representation learning.
Variational Autoencoder (VAE)	Improve interpolation ability, generative model.
Generative Adversarial Networks (GAN)	Learn to generate samples from a distribution.
Diffusion Model	Transform noise into image, step by step.

ResNet

The shortest path of ResNet contains most of the gradient signal.
A ResNet with k residual blocks can be viewed as equivalent to an ensemble of 2^k networks.

Training Problems

Overfitting. Try regularization, weight decay, early stopping, dropout, ensembling, data augmentation, (max-pooling, increase channels, decrease kernel size)
Generalization gap. Try regularization, data augmentation, and dropout, to improve generalization
Underfitting. Try training for more epochs, or increase the learning rate.

Large Language Models

Tokenization. Split natural language input into a sequence of subwords (tokens). ① *Rule-based:* Split by Spaces + Special Handling of Punctuations, default for English, see NLTK ② *Dictionary/Model-based:* for languages that there are no easy rules to split the sentence, e.g. jieba for Chinese ③ *Statistics-based:* most widely used in NLP, e.g. Byte-pair encoding (BPE), repeat the process of merging most frequently appearing nearby subtokens.

Langauge Models (LMs). ① *Task:* build a probablistic model $p(T)$ for the input tokens sequence $T = [t_0, t_1, ..., t_{N-1}]$ ② *Decomposition:* $p(T) = p(t_0) * p(t_1|t_0) * p(t_2|t_0, t_1) * ... * p(t_{N-1}|t_0, t_1, ..., t_{N-2}) \rightarrow$ next token prediction: $p(t_i|t_0, ..., t_{i-1})$.
LM evaluation. ① *perplexity (PPL):* $PPL = b^{-\frac{1}{N}\sum_x p(x)\log_b p(x)}$ ② *2-based PPL:* $PPL = 2^{-\frac{1}{N}\sum_i \log_2 p(t_i)}$
Next Token Prediction. ① *N-gram:* assume $p(t_i|t_0, ..., t_{i-1}) \approx p(t_i|t_{i-n}, ..., t_{i-1})$, e.g. Unigram: $p(t_i)$, Bi-gram: $p(t_i|t_{i-1})$, Tri-gram: $p(t_i|t_{i-2}, t_{i-1})$, ... ② *N-gram problem:* strong assumption, sparsity with OOV words, semantic meaning lack ③ *Neural LM:* utilizes a vector to represent each word/token to allow better modeling of the semantics, e.g. *word embedding*, *word2vec*, *Glove* ④ *Transformer LM:* attention-based, easy to parallesize and can scale.

Large Language Models (LLMs). *Scaling Laws:* ① ELmo and BERT (2018) w/ Large-scale Pre-training → ② GPT-3 (2020) w/ In-Context Learning ③ InstructGPT (2022) w/ RLHF → ③ OpenAI-o1 (2024) and Deepseek-R1 (2025) w/ Reasoning.

Vision Language Models (VLM). Region-Based CNN (RCNN / R-CNN) → Fast R-CNN → Faster R-CNN → Region Proposal Network (RPN) → You Only Look Once (YOLO) → DEtection TRansformer (DETR) . . ..

Algorithm 1 Orthogonal Matching Pursuit (OMP)

```
1: Initialize the residual: $\boldsymbol{r} = \boldsymbol{y}$
2: for $t = 1$ to $K$ do
3:     Find the most correlated feature: $j = \arg\max_j |r^T \boldsymbol{x}_j|$, where $\boldsymbol{x}_j$ is the $j$-th row of $\boldsymbol{X}$ (the $j$-th feature).
4:     Compute the weight: $\boldsymbol{w}_j = \arg\min_{\boldsymbol{w}_j} \|\boldsymbol{r} - \boldsymbol{x}_j \boldsymbol{w}_j\|^2$
5:     Update the residual: $\boldsymbol{r} = \boldsymbol{r} - \boldsymbol{x}_j \boldsymbol{w}_j$
6: end for
```

Algorithm 2 RANSAC (Random Sample Consensus)

```
1: Given: training set $D = \{\boldsymbol{x}_i, y_i\}$, threshold $\epsilon$, loss function $L = \sum l(x, y)$
2: Classify all data as inlier or outlier by calculating the prediction errors $l$ and comparing to the threshold $\epsilon$. (typically set as the median absolute deviation (MAD) of $y$, i.e. median$(|y_i - \text{median}(y)|)$)
3: The set of inliers is called the *consensus set*.
4: Save the model with the highest number of inliers.
5: Use the largest consensus set to learn the final model.
```

Algorithm 3 k-means Clustering

```
1: Input: Data points $\{\boldsymbol{x}_i\}_{i=1}^n \subset \mathbb{R}^d$, number of clusters $k$.
2: Initialization: Choose initial cluster centers $\{\boldsymbol{c}_j^{(0)}\}_{j=1}^k \subset \mathbb{R}^d$.
3: Set iteration counter $t \leftarrow 0$.
4: repeat
5:     Assignment step: For each $i = 1, \ldots, n$, update the cluster label $z_i^{(t)} \leftarrow \arg\min_{j \in \{1,\ldots,k\}} \|\boldsymbol{x}_i - \boldsymbol{c}_j^{(t)}\|_2^2$.
6:     Update step: For each $j = 1, \ldots, k$, update the cluster center $\boldsymbol{c}_j^{(t+1)} \leftarrow \frac{1}{|\mathcal{C}_j^{(t)}|} \sum_{i \in \mathcal{C}_j^{(t)}} \boldsymbol{x}_i, \quad \mathcal{C}_j^{(t)} := \{\, i \in \{1, \ldots, n\} : z_i^{(t)} = j \,\}$.
7:     $t \leftarrow t + 1$.
8: until Convergence.
```

Algorithm 4 EM Algorithm for Gaussian Mixture Model (GMM)

```
1: Input: Data points $\{\boldsymbol{x}_i\}_{i=1}^N \subset \mathbb{R}^d$, number of clusters $K$, initial parameters for means $\{\boldsymbol{\mu}_j\}_{j=1}^K$, covariances $\{\Sigma_j\}_{j=1}^K$, and weights $\{\pi_j\}_{j=1}^K$.
2: Set iteration counter $t \leftarrow 0$.
3: repeat
4:     E-step: Calculate cluster membership for each point $\boldsymbol{x}_i$: $\hat{z}_{ij} = p(z_i = j|\boldsymbol{x}_i) = \frac{\pi_j \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu}_j, \Sigma_j)}{\sum_{k=1}^K \pi_k \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu}_k, \Sigma_k)}$.
5:     M-step: Update the parameters of each Gaussian cluster: Soft count of points in cluster $j$: $N_j = \sum_{i=1}^N \hat{z}_{ij}$.
6:     $\pi_j \leftarrow \frac{N_j}{N}$, $\boldsymbol{\mu}_j \leftarrow \frac{1}{N_j} \sum_{i=1}^N \hat{z}_{ij} \boldsymbol{x}_i$, $\Sigma_j \leftarrow \frac{1}{N_j} \sum_{i=1}^N \hat{z}_{ij} (\boldsymbol{x}_i - \boldsymbol{\mu}_j)(\boldsymbol{x}_i - \boldsymbol{\mu}_j)^\top$.
7:     $t \leftarrow t + 1$.
8: until Convergence.
```

Algorithm 1 Mean-shift Algorithm

```
1: Input: Data points $\{\boldsymbol{x}_i\}_{i=1}^N \subset \mathbb{R}^d$, bandwidth parameter $h$, initial point $\boldsymbol{x}^{(0)}$.
2: Set iteration counter $t \leftarrow 0$.
3: repeat
4:     1) Find the neighbors of $\boldsymbol{x}^{(t)}$: For each data point $\boldsymbol{x}_i$, calculate the distance to $\boldsymbol{x}^{(t)}$. Find the set of neighbors within a bandwidth radius $h$: $\mathcal{N}(\boldsymbol{x}^{(t)}) = \left\{ \boldsymbol{x}_i : \|\boldsymbol{x}_i - \boldsymbol{x}^{(t)}\|_2 \le h \right\}$.
5:     2) Shift the point towards the mean of the neighbors: Update $\boldsymbol{x}^{(t+1)}$ to the weighted mean of the points in $\mathcal{N}(\boldsymbol{x}^{(t)})$ using the Gaussian kernel:
           $$\boldsymbol{x}^{(t+1)} = \frac{\sum_{\boldsymbol{x}_i \in \mathcal{N}(\boldsymbol{x}^{(t)})} \boldsymbol{x}_i \exp\left( -\frac{\|\boldsymbol{x}_i - \boldsymbol{x}^{(t)}\|_2^2}{2h^2} \right)}{\sum_{\boldsymbol{x}_i \in \mathcal{N}(\boldsymbol{x}^{(t)})} \exp\left( -\frac{\|\boldsymbol{x}_i - \boldsymbol{x}^{(t)}\|_2^2}{2h^2} \right)}.$$
6:     $t \leftarrow t + 1$.
7: until $\|\boldsymbol{x}^{(t+1)} - \boldsymbol{x}^{(t)}\|_2$ is below a given threshold (convergence).
```