MATH REVIEW

PSD $A \succeq 0$

if $\forall v \neq 0: v^T A v \geq 0$

SVD $\boxed{M = U \Sigma V^T}$

U cols = eigenvectors of MM^T

V cols = eigenvectors of $M^T M$

$\lambda_i = \Sigma_{ii}^2 \qquad \Sigma_{ii} = \sigma_i$

Spectral thm

$\boxed{A = Q \Lambda Q^T}$

for square symmetric matrices

Λ = eigenvalues of A

Q = eigenvectors of A

JENSEN INEQUALITY
$f(t\theta_1 + (1-t)\theta_2) \leq t f(\theta_1) + (1-t) f(\theta_2)$

Iso contours = ellipses centered around dist's mean

To find direction of major axis:

1. Find eigenvalues / eigenvectors
2. Find eigenvector corresponding to largest eigenvalue

covariance matrix

$$\Sigma = \begin{bmatrix} \mathrm{cov}(X_1, X_1) & \dots & \mathrm{cov}(X_1, X_n) \\ \dots & & \dots \\ \mathrm{cov}(X_n, X_1) & \dots & \mathrm{cov}(X_n, X_n) \end{bmatrix}$$

$X = AZ + \mu$ where $Z \sim N(0, I_n)$

$\boxed{\Sigma = AA^T}$ * zero mean

* diagonal entries = variances
* off-diagonal = covariances
* uncorrelated: $E[XY] = E[X]E[Y]$

Lagrangian multiplier

$\mathcal{L}(x, y, \lambda) = f(x,y) - \lambda g(x,y)$

example: constrained optimization

$\|w_1\|_2 = 1 \leftarrow$ constraint

$\underset{w_1}{\mathrm{argmin}} \; - w_1^T S w_1 + \underbrace{\lambda (w_1^T w_1 - 1)}_{\text{Lagrange}}$

Bayes Rule!!!

$$\boxed{\underbrace{P(Y|X)}_{\text{posterior}} = \frac{P(X|Y)\overset{\text{prior}}{P(Y)}}{P(X)}}$$

$$P(Y=0|X) = \frac{P(X|Y=0)\pi_1}{P(X|Y=0)\pi_1 + P(X|Y=1)(1-\pi_1)}$$

EXPECTATION

$E[f] = \sum_x p(x) f(x)$

variance

$\mathrm{var}[X] = E[f(x)^2] - E[f(x)]^2$

covariance

$\mathrm{cov}[X, Y] = E[XY] - E[X]E[Y]$

* maximize negative = minimize

orthonormal vectors

If W orthogonal, then $w_1^T w_1 = 1$, $w_1^T w_2 = 0$

L1 Norm

$$\boxed{\|x\|_1 = \sum_{i=1}^n |x_i|}$$

L2 Norm

$$\|x\|_2 = \sqrt{\sum_{i=1}^n x_i^2}$$

$\|x\|_2^2 = x^T x$

* gradient = transpose of partials

$\nabla_x (w^T x) = w$

$\nabla_x (w^T A x) = A^T w$

$\nabla_A (w^T A x) = w x^T$

$\nabla_x (x^T A x) = (A + A^T) x$

Matrix Derivatives

$$\frac{\partial \mathbf{x}^T \mathbf{a}}{\partial \mathbf{x}} = \frac{\partial \mathbf{a}^T \mathbf{x}}{\partial \mathbf{x}} = \mathbf{a}$$

$$\frac{\partial \mathbf{a}^T \mathbf{X} \mathbf{b}}{\partial \mathbf{X}} = \mathbf{a}\mathbf{b}^T$$

$$\frac{\partial \mathbf{a}^T \mathbf{X}^T \mathbf{b}}{\partial \mathbf{X}} = \mathbf{b}\mathbf{a}^T$$

$$\frac{\partial \mathbf{a}^T \mathbf{X} \mathbf{a}}{\partial \mathbf{X}} = \frac{\partial \mathbf{a}^T \mathbf{X}^T \mathbf{a}}{\partial \mathbf{X}} = \mathbf{a}\mathbf{a}^T$$

$$\frac{\partial \mathbf{X}}{\partial X_{ij}} = \mathbf{J}^{ij}$$

$$\frac{\partial (\mathbf{X}\mathbf{A})_{ij}}{\partial X_{mn}} = \delta_{im}(\mathbf{A})_{nj} = (\mathbf{J}^{mn}\mathbf{A})_{ij}$$

$$\frac{\partial (\mathbf{X}^T\mathbf{A})_{ij}}{\partial X_{mn}} = \delta_{in}(\mathbf{A})_{mj} = (\mathbf{J}^{nm}\mathbf{A})_{ij}$$

$$\frac{\partial}{\partial X_{ij}} \sum_{klmn} X_{kl} X_{mn} = 2 \sum_{kl} X_{kl}$$

$$\frac{\partial \mathbf{b}^T \mathbf{X}^T \mathbf{X} \mathbf{c}}{\partial \mathbf{X}} = \mathbf{X}(\mathbf{b}\mathbf{c}^T + \mathbf{c}\mathbf{b}^T)$$

$$\frac{\partial (\mathbf{B}\mathbf{x} + \mathbf{b})^T \mathbf{C} (\mathbf{D}\mathbf{x} + \mathbf{d})}{\partial \mathbf{x}} = \mathbf{B}^T \mathbf{C}(\mathbf{D}\mathbf{x} + \mathbf{d}) + \mathbf{D}^T \mathbf{C}^T (\mathbf{B}\mathbf{x} + \mathbf{b})$$

$$\frac{\partial (\mathbf{X}^T \mathbf{B} \mathbf{X})_{kl}}{\partial X_{ij}} = \delta_{lj}(\mathbf{X}^T\mathbf{B})_{ki} + \delta_{kj}(\mathbf{B}\mathbf{X})_{il}$$

$$\frac{\partial (\mathbf{X}^T \mathbf{B} \mathbf{X})}{\partial X_{ij}} = \mathbf{X}^T \mathbf{B} \mathbf{J}^{ij} + \mathbf{J}^{ji} \mathbf{B} \mathbf{X} \qquad (\mathbf{J}^{ij})_{kl} = \delta_{ik}\delta_{jl}$$

$$\frac{\partial \mathbf{x}^T \mathbf{B} \mathbf{x}}{\partial \mathbf{x}} = (\mathbf{B} + \mathbf{B}^T)\mathbf{x}$$

$$\frac{\partial \mathbf{b}^T \mathbf{X}^T \mathbf{D} \mathbf{X} \mathbf{c}}{\partial \mathbf{X}} = \mathbf{D}^T \mathbf{X} \mathbf{b} \mathbf{c}^T + \mathbf{D} \mathbf{X} \mathbf{c} \mathbf{b}^T$$

$$\frac{\partial}{\partial \mathbf{X}} (\mathbf{X}\mathbf{b} + \mathbf{c})^T \mathbf{D} (\mathbf{X}\mathbf{b} + \mathbf{c}) = (\mathbf{D} + \mathbf{D}^T)(\mathbf{X}\mathbf{b} + \mathbf{c})\mathbf{b}^T$$

Jacobian

$$J = \begin{bmatrix} \frac{\partial z_1}{\partial x} \\ \vdots \\ \frac{\partial z_k}{\partial x} \end{bmatrix} = \begin{bmatrix} \frac{\partial z_1}{\partial x_1} & \dots & \frac{\partial z_1}{\partial x_d} \\ \vdots & \ddots & \\ \frac{\partial z_k}{\partial x_1} & & \frac{\partial z_k}{\partial x_d} \end{bmatrix}$$

Frobenius Norm

$$\|A\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n |A_{ij}|^2} = \sqrt{tr(A^T A)} = \sqrt{\sum_{i=1}^r \sigma_i^2}$$

singular values

$\sigma_i = \sqrt{\lambda_i}$ where λ_i = eigenvals of $A^T A$

Hessian

$$H = \nabla^2 f(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \dots & \frac{\partial^2 f}{\partial x_1 \partial x_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_d \partial x_1} & \dots & \frac{\partial^2 f}{\partial x_d^2} \end{bmatrix}$$

OUTER PRODUCT FORM

$X^T X = \sum_j x_j x_j^T$

$\underbrace{S = \frac{1}{N} \sum_{n=1}^N x_n x_n^T}_{\text{sample covariance if data centered}}$

Normal Gaussian PDF

$$\boxed{N(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)}$$

RELU

$$RELU(x) = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases}$$

MULTIVARIATE GAUSSIAN

$$p(x) = \frac{1}{(2\pi)^{\frac{n}{2}} \underset{\uparrow \text{covariance}}{|\Sigma|^{1/2}}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x - \mu)\right)$$

$\nabla_w (Xw - y)^T (Xw - y)$
$= \nabla_w ((Xw)^T(Xw) - (Xw)^T(y) - y^T(Xw) + y^Ty)$
$= \nabla_w (w^T X^T X w - 2 w^T X^T y + y^T y)$
$= (X^T X + X^T X) w - 2X^T y$
$= 2X^T (Xw - y)$

$0 = 2X^T X w - 2X^T y$

$2X^T y = 2X^T X w$ invertible

$w^* = (X^T X)^{-1} X^T y$

SIGMOID FUNCTION

$$\sigma(x) = \frac{1}{1 + \exp(-z)}$$

* prone to vanishing gradients

LIKELIHOOD FUNCTION

$$P(X|\mu, \sigma^2) = \prod_{n=1}^N N(x_n | \mu, \sigma^2)$$

$$L(\theta) = P(\overset{\text{event}}{x_1}|\theta)^{\overset{\text{count}}{k_1}} \cdot P(x_2|\theta)^{k_2} \dots P(x_m|\theta)^{k_m}$$

examples:

$L(p) = P(\text{coin A}|T)^2 \cdot P(\text{coin A}|H) \cdot P(\text{coin B}|H)^3$

$L(\mu, \Sigma) = \prod_{i=1}^n p(x_i | \mu, \Sigma)$

$L(w) = \prod_{i=1}^n P(Y = y_i | x_i)$

LOG-LIKELIHOOD

Take log of $L(\theta)$ since log is monotonically increasing

$\ell(\mu, \Sigma) = \log\left(\prod_{i=1}^n p(x_i | \mu, \Sigma)\right)$

* exponents → multiplication
* products → sums

MLE = max likelihood estimation = finding param vals to maximize likelihood function

$$\boxed{\underset{\mu, \sigma}{\mathrm{argmax}} \; p(x_1, x_2 \dots x_n | \mu, \sigma)}$$

$\ell(p) = \log 27 + 4 \log p + 2 \log(1-p)$

$\frac{\partial \ell(p)}{\partial p} = \frac{4}{p} + \frac{-2}{1-p} = 0 \longrightarrow p = \frac{2}{3}$

take derivative wrt p (or other vars) and set = 0

* $X^T X$ invertible when full rank
* features > num data points = undetermined
  * ↳ use regularization
* data pts > features = overdetermined
  * ↳ least squares

MAP = maximum a posteriori

* point estimate of param to maximize posterior

$$\hat{\theta}_{MAP} = \underset{\theta}{\mathrm{argmax}} \; P(\theta | X) = \underset{\theta}{\mathrm{argmax}} \; \frac{P(X|\theta)P(\theta)}{P(X)}$$

* uniform prior: MAP = MLE
* incorporates prior but MLE does NOT

$$\boxed{\hat{\theta}_{MAP} = \underset{\theta}{\mathrm{argmax}} \; P(X|\theta)P(\theta)} = \underset{\theta}{\mathrm{argmax}} \left[\prod_{i=1}^n p(x_i | \theta)\right] p(\theta)$$

LINEAR REGRESSION

$y = Xw$

$\mathcal{L} = \underset{w}{\mathrm{argmin}} \; \|y - Xw\|_2^2$

$w^* = (X^T X)^{-1} X^T y$

$\hat{y} = X^T (X^T X)^{-1} X^T y$

RIDGE REGRESSION

* underdetermined models

$\mathcal{L} = (y - Xw)^T (y - Xw) + \underbrace{\lambda \|w\|_2^2}_{\text{penalty term}}$

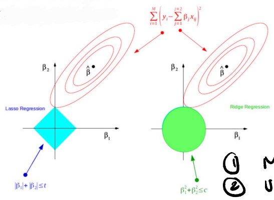
$w^* = \underbrace{(X^T X + \lambda I)^{-1}}_{\text{invertible when } \lambda > 0} X^T y$

LASSO REGRESSION

* induces sparsity w/ L1 norm

$$\boxed{w^* = \underset{w}{\mathrm{argmin}} (y - Aw)^T (y - Aw) + \lambda \|w\|_1}$$

* sharper corners = more sparse



GENERATIVE CLASSIFIER

Features = X, Labels = Y

* model probability dist for both classes & learning params that best fit this distribution

① Model each class P(X|Y) and P(Y)

② Use Bayes to compute P(Y|X)

DISCRIMINATIVE CLASSIFIER

Model conditional probability dist P(Y|X) directly

* uses sigmoid function for binary

$P(Y=1|X)=\sigma(w^T x)=\frac{1}{1+e^{-(w^T x)}}$

* softmax for multiclass classification

$$P(Y=k|X)=\frac{e^{z_k}}{\sum_{j=1}^{k} e^{z_j}}$$

LOGISTIC REGRESSION

Goal: find probability that input x belongs to a certain class P(Y=1|X)

BINARY

$$P(y=1|x;w)=\frac{1}{1+e^{-w^T x}}$$

$$P(y=0|x;w)=\frac{e^{-w^T x}}{1+e^{-w^T x}}$$

when $z=w^T x$

$$\text{log-odds}=\log\left(\frac{P(Y=1|X)}{P(Y=0|X)}\right)$$

Neural network

① $a_j^{(\ell)}=\sum_{i\in[d_{\ell-1}]}\theta_{ji}^{(\ell)}x_i^{(\ell-1)}$ (pre activation value for j^{th} neuron in layer ℓ

weights

neurons from prev layer

② $x_j^{(\ell)}=g_\ell(a_j^{(\ell)})$

activation function

pre-activation value

③ $g_\ell=\mathbb{R}\to\mathbb{R}$ activation function

CROSS ENTROPY LOSS

pred

$$\mathcal{L}=-y\cdot\ln(\hat{y})$$

batch of m samples

$$J=-\frac{1}{m}\left(\sum_{i=1}^{m}y_i\cdot\ln(\hat{y}_i)\right)$$

negative log likelihood $\mathcal{L}(\theta)=-y\log p(z)-(1-y)\log(1-p(z))$

* want to minimize loss
* minimize using SGD

for 1 layer NN → $\frac{\partial L}{\partial\theta_{ki}^{(1)}}=\underbrace{\frac{\partial L}{\partial a_k^{(1)}}\cdot\frac{\partial a_k^{(1)}}{\partial\theta_{ki}^{(1)}}}_{\text{chain rule}}$

gradient of loss function wrt weight $\theta_{ki}^{(1)}$

partial of loss wrt pre-activation value

partial of pre-activation value wrt weight

STOCHASTIC GRADIENT DESCENT

$$\theta_{t+1}=\theta_t-\epsilon_t\nabla_\theta\mathcal{L}(x_i,y_i;\theta)$$

Backprop:

① Forward pass: calculate preactivation value
apply activation function

② Backward pass:
- Take gradients of loss wrt params of each layer
- use chain rule to propagate backwards
- update weights w/ gradient descent after gradients calculated

loss

$$\left[\frac{\partial L}{\partial Z}\right]_{ij}=\frac{\partial L}{\partial Z_{ij}}=\frac{\partial L}{\partial Y_{ij}}\frac{\partial Y_{ij}}{\partial Z_{ij}}=\frac{\partial L}{\partial Y_{ij}}\mathbb{1}_{Z_{ij}>0}\rightarrow\frac{\partial L}{\partial Z}=\frac{\partial L}{\partial Y}\odot\mathbb{1}_{Z>0}$$

activation function

indicator matrix

element wise

Normalization when training deep networks

MEAN $\mu=\frac{1}{n}\sum_{i\in[n]}x_i$ | STANDARD DEV $\sigma^2=\frac{1}{n}\sum_{i\in[n]}(x_i-\mu)^2$ | $x\leftarrow\frac{x-\mu}{\sigma}$

subtract

convolutional neural networks

- convolutional filter: same weights applied across many different locations

$$Z[d_1,d_2,n]=(X*W)[d_1,d_2,n]=\sum_i\sum_j\sum_c W[i,j,c,n]X[d_1+i,d_2+j,c]+b[n].$$

# of channels

- pooling layer: downsample image
  ↳ introduces translational invariance

* skip connections to deal w/ vanishing gradients

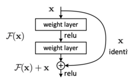
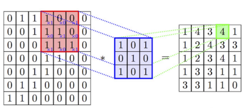
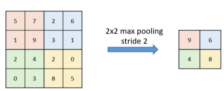


Figure 2. Residual learning: a building block.

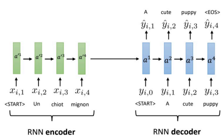


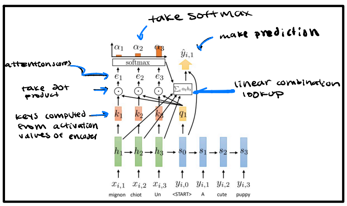
CONVOLUTION



POOLING

SEQ2SEQ MODELS





ATTENTION

① Transform encoder activation to key

② Transform decoder activation to query

③ attention sore = dot product of query w/ keys

④ Softmax to normalize attention scores

⑤ send all of $h_1\cdots h_n$ through linear combo dictated by softmax of attn scores

PCA = principal component analysis

step by step:

① center the data around the mean

② compute covariance matrix $\Sigma=X^T X$

③ Eigendecomp: $X^T X=QDQ^T$ or use SVD $X=U\Sigma V^T$ V = eigenvectors $X^T X$

④ keep top k eigenvectors $Q_k\equiv Q_{:,1:k}$

⑤ Project points down to subspace → principal component scores

* final dim reduced data: $\bar{X}_k=\bar{X}Q_k\in\mathbb{R}^{n\times k}$
  ↳ to reconstruct: $\bar{X}_{recon_k}=\bar{X}_k Q_k^T=XQ_kQ_k^T$

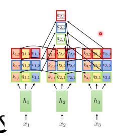
* recon loss: $\|\bar{X}_{recon-k}-\bar{X}\|_F$ ← want smallest error possible

reconstructed

original

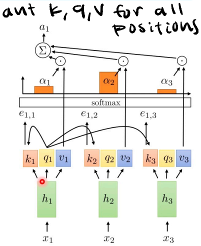
MULTIHEAD ATTN

* multiple keys, queries, values for each time step
* nonlinearities
* masked decoding
  ↳ not allowed to look at future values



SELF ATTENTION

* want k, q, v for all positions



t-SNE = computed shortest pairwise distance between points

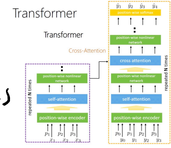
* t-distribution = distribution heavier tail

$$Q_{j\leftarrow i}=\frac{(1+\|y_i-y_j\|^2)^{-1}}{\sum_{k\neq\ell}(1+\|y_k-y_\ell\|^2)^{-1}}$$ change in computing stochastic neighbors

Algorithm 1: Simple version of t-Distributed Stochastic Neighbor Embedding.

Data: data set $X=\{x_1,x_2,...,x_n\}$,
cost function parameters: entropy constant, C to set σ_{ij}
optimization parameters: number of iterations T, learning rate η
Result: low-dimensional data representation $\mathcal{Y}^{(T)}=\{y_1,y_2,...,y_n\}$.
begin
compute pairwise affinities $p_{j|i}$ with entropy constant, C to set σ_{ij}
set $p_{ij}=\frac{p_{j|i}+p_{i|j}}{2n}$
sample initial solution $\mathcal{Y}^{(0)}=\{y_1,y_2,...,y_n\}$ from $\mathcal{N}(0,10^{-4}I)$
for t=1 **to** T **do**
compute low-dimensional affinities q_{ij} (using Equation 4)
compute gradient $\frac{\delta C}{\delta\mathcal{Y}}$ (using Equation 5)
set $\mathcal{Y}^{(t)}=\mathcal{Y}^{(t-1)}+\eta\frac{\delta C}{\delta\mathcal{Y}}$
end
end

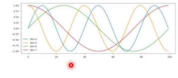
Transformer



TRANSFORMER

* stacked self attn layers w/ pos wise nonlinearities
* easily parallelizable

$$p_t=\begin{bmatrix}\sin(t/10000^{2*1/d})\\\cos(t/10000^{2*1/d})\\\sin(t/10000^{2*2/d})\\\cos(t/10000^{2*2/d})\\\cdots\\\sin(t/10000^{2*\frac{d}{2}/d})\\\cos(t/10000^{2*\frac{d}{2}/d})\end{bmatrix}$$



d, is the dimensionality of positional encoding

POSITIONAL ENCODING

* keep ordering
→ use periodicity

$h_t=f(x_t,t)$

K means clustering

$$\underset{\substack{S=\{U_1 \ldots U_{C_k}\}, \{c_1 \ldots c_k\} \\ \text{cluster partition} \quad \text{cluster centroids}}}{\operatorname{argmin}} \sum_k \sum_{x \in C_k} \|x - c_k\|^2$$

Steps:

1. Compute partition by choosing closest centroid
2. Compute centers by averaging over partition
3. Continue until centers do not change

Voronoi Tesselation

For euclidean metric all boundaries = linear

$\|(x - x^A)\|_2^2 = \|x - x_B\|_2^2$

$\sum_{i=1}^{d} (x_i - x_i^A)^2 = \sum_{i=1}^{d} (x_i - x_i^B)^2$

Soft K-means: use softmax of distances

soft partition

$r_{ik} = \text{softmax}(\{-\beta \|x_i - c_k\|^2\})$

$$\hat{c}_k = \frac{\sum_i r_{ik} x_i}{\sum_i r_{ik}}$$

Mixture of Gaussians: model cluster as non-spherical gaussian

likelihood: $L_i = p(x_i) = \sum_{k=1}^{K} p(x_i, z_i = k)$

$= \sum_{k=1}^{K} P(x_i \mid z_i = k) P(z_i = k) = \sum_{k=1}^{K} N(x_i \mid \mu_k, \Sigma_k) P(z_i = k)$

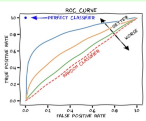
learn these params

Kleinberg Impossibility Thm

1. scale invariance: stretching data = same cluster
2. consistency: stretching space between cluster = same cluster
3. richness: clustering to produce any arbitrary partition

Classifier decision outcomes

		MODEL PREDICTIONS	
		Negative	Positive
GROUND TRUTH	Negative	TN	FP
	Positive	FN	TP



sensitivity = $\frac{TP}{\#\text{ actual pos}}$

specificity = $\frac{TN}{\#\text{ actual neg}}$

miss rate = $\frac{FN}{\#\text{ actual pos}}$

fallout = $\frac{FP}{\#\text{ actual neg}}$

Bias-Variance

Bias = avg diff between model output ⟺ truth

↳ underfitting = HIGH BIAS

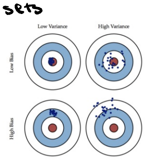
overfitting = LOW BIAS

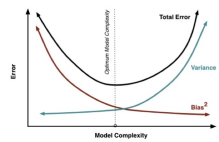
variance = variance over all possible train sets

For regularization: increasing λ ↓ less flexible model prevent overfitting ↓ higher bias lower variance

$\text{Bias} = E[\hat{X} - \mu] = E[\hat{p}] - p$

variance = $\text{Var}[\hat{X}]$





KNN Algorithm (supervised)

1. for each training sample $(x_i, y_i) \in D$ compute distance between x and x_i
2. choose set of training samples with k smallest distance N
3. return majority label of samples in N

Pros:
- No training
- learns complex nonlinear functions

Cons:
- High storage cost
- slow inference
- curse of dimensionality (worse in higher dim)

Bayes Optimal Classifier

$$P(Y = c \mid X) = \frac{P(X \mid c) P(c)}{\sum_{c \in [C]} P(X \mid c) P(c)}$$

Conditional indep

$a \perp\!\!\!\perp (b, c) \mid d \xrightarrow{\text{implies}} a \perp\!\!\!\perp b \mid d$

$P(a, b, c \mid d) = P(a \mid d) P(b, c \mid d)$

$\sum_c P(a, b, c \mid d) = \sum_c P(a \mid d) P(b, c \mid d)$

$P(a, b \mid d) = P(a \mid d) P(b \mid d)$

Bayes Decision Boundary

when 2 probabilities weighted w/ cost are equal

$L(1,0) P(Y = 0 \mid X) = L(0,1) P(Y = 1 \mid X)$

Triplet Loss: minimize distance between reference & pos sample, maximize distance between reference & neg sample

$$\mathcal{L}_{triplet} = \max(0, d(x_{anchor}, x_{pos})^2 - d(x_{anchor}, x_{neg})^2 + \text{margin})$$

N-pair Loss = compare w/ multiple neg samples at same time

$$\mathcal{L}_{n\text{-pair}} = \frac{1}{n} \sum_{i=1}^{n} \log\left(1 + \sum_{k \neq i} \exp(\langle x_i, x_k \rangle - \langle x_i, x_j \rangle)\right)$$

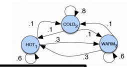
Markov chains

Markov assumption: $P(q_i = a \mid q_1 \cdots q_{i-1}) = P(q_i = a \mid q_{i-1})$

$Q = q_1 q_2 \cdots q_N$ → N states

$A = a_{11} a_{12} \cdots a_{nn}$ → transition probability matrix A where a_{ij} = prob of moving from state i to j

$\pi = \pi_1 \cdots \pi_N$ → initial probability dist over states



for states $X_1 \cdots X_n$: $\boxed{\text{Joint dist}: P(X_1, X_2, \cdots X_n) = P(X_1) \prod_{i=2} P(X_i \mid X_{i-1})}$

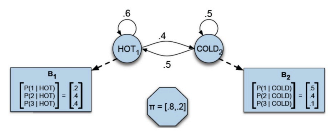
w/ no indepence assumptions: min # params = $k^T - 1$ (# of states)

Hidden Markov models

= events interested in cannot be observed directly

HMM has

1. set of K states in $[K] = \{1 \cdots k\}$
2. Transition matrix A where rows sum to 1
3. seq of observations $Y_{1:T} = (Y_1 \cdots Y_T)$
4. Emission prob: $B_{kl} = P(Y_t = l \mid X_t = k)$ — prob of observation $Y_t = l$ generated from state $X_t = k$
5. initial probability distribution (π_i)

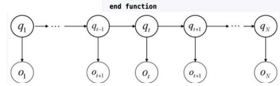


Viterbi Algorithm = find the most likely sequence of hidden states in HMM

Viterbi Pseudocode

- T1 stores prob of most likely path so far ending in state i.
- T2 stores the most recent observation in this path.
- We populate these matrices, computing a distribution over states at each timestep.
- Finally, we find the most likely path by working backwards from the final state.

```
function VITERBI(O, S, Π, Y, A, B) : X
    for each state i = 1,2,...,K do
        T1[i,1] ← π_i · B_{iy_1}
        T2[i,1] ← 0
    end for
    for each observation j = 2,3,...,T do
        for each state i = 1,2,...,K do
            T1[i,j] ← max_k (T1[k,j−1] · A_{ki} · B_{iy_j})
            T2[i,j] ← arg max_k (T1[k,j−1] · A_{ki} · B_{iy_j})
        end for
    end for
    z_T ← arg max_k (T1[k,T])
    x_T ← s_{z_T}
    for j = T, T−1, ..., 2 do
        z_{j−1} ← T2[z_j, j]
        x_{j−1} ← s_{z_{j−1}}
    end for
    return X
end function
```



$\delta_t(j) = \max_i [\delta_{t-1}(i) \cdot a_{ij}] \cdot b_j(o_t)$

* Dynamic programming + backtracking

Decision Trees

At each node: split by feature → traverse until you hit a leaf node = output

Next best attribute = feature + split that MAX info gain / MIN entropy

Greedy Algorithm:

Start with empty tree

For each node:

- If stopping condition reached:
  - Leaf label = average of data at that node
- Else:
  - Split by **next best attribute**
  - Recurse to child nodes

Entropy: $H(Y) = E_Y[-\log P(Y)]$

$= -\sum_k P(Y = k) \log P(Y = k)$ (proportion of k out of total)

surprise = $-\log(P(Y = k))$

conditional entropy: $H(Y \mid X) = \sum_{x \in X} p(x) H(Y \mid X = x)$

Info gain: $I(S, A) = \text{Entropy}(S) - \sum_{v \in \text{values}(A)} \frac{|S_v|}{|S|} \text{entropy}(S_v)$

Binary (left + right) Info gain: $H(L+R) - \left(\frac{|L|}{|L|+|R|} H(L) + \frac{|R|}{|L|+|R|} H(R)\right)$

Bagging: Train M models with n' samples, sample with replacement → $\boxed{\text{REDUCES VARIANCE}}$

Random forests: same as bagging but at each split, choose only random subset $p' = \sqrt{p}$ features

* trained in parallel

Avg model performance: $\hat{Y}_{bagging} = \frac{1}{M} \sum_{m=1}^{M} \hat{y}_m(x)$

weighted avg model: $\hat{Y}_{\text{weighted bagging}} = \sum_{m=1}^{M} \alpha_m \hat{y}_m(x)$

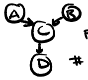
Boosting:

1. train next model conditioned on all prev models + weights
2. reweight models to minimize loss
3. repeat

* trained sequentially
* $\boxed{\text{reduces bias}}$ by weighting misclassified points more
* stumps better for boosting

Probabilistic Graph Models

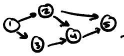
Node = random variable
Edge = dependence relationship



Joint Dist:

$P(a,b,c,d) = \overset{1}{P(a)}\overset{1}{P(b)}\overset{4}{P(c|a,b)}\overset{2}{P(d|c)}$

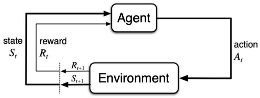
# of params = 1 + 1 + 4 + 2 = 8



$P(1{:}5) = P(1)P(2|1)P(3|1)P(4|2,3)P(5|2,4)$

# of params = 1+2+2+4+4 = 13

Markov Decision Process (MDP)



Transition dynamic: $P(S_t, R_t \mid S_{t-1}, A_{t-1})$

* conditioning on all history = conditioning on just previous state
* maximize sum of discounted rewards / return

Return: $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$

$= \boxed{\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} = R_{t+1} + \gamma G_{t+1}}$

State value: $v_\pi(s) = E_\pi[G_t \mid S_t = s] = E_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s]$

Action value: $q_\pi(s,a) = E_\pi[G_t \mid S_t = s, A_t = a] = E_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a]$

Bellman equation

value function: $V_\pi(s) = E_\pi[q_\pi(s,a)] = \sum_{a \in A} \pi(a|s) q(s,a)$

for $R(s)$

$V^*(s) = R(s) + \gamma \max_{a \in A} \sum_{s' \in S} P(s'|s,a) V^*(s')$

for $R(s,a)$

$V^*(s) = \max_{a \in A} [R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^*(s')]$

Policy iteration:
1. Init value func & policy randomly
2. Policy evaluation
3. Policy improvement

(2)–(3): repeat until converged

for $R(s,a,s')$

$V^* = \max_{a \in A} \sum_{s' \in S} P(s'|s,a)[R(s,a,s') + \gamma V^*(s)]$

Q function: $Q(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) V(s')$

Policy Eval: $V(s) = \sum_a \pi(a|s) Q(s,a)$

Policy improvement: $\pi(s) = \arg\max_a Q(s,a)$

Value Iteration: $V(s) = \max_a Q(s,a)$

for each state s

α-algorithm

1. initialization: $\alpha(x_t) = P(x_t, y_{1:t})$
2. recursive: $\alpha(x_t) = P(y_t|x_t) \sum_{x_{t-1}} \alpha(x_{t-1}) P(x_t|x_{t-1}) \quad t > 1$

Kernels: run model of high dim set of features w/o blowing up complexity

1. Project features to higher dim space ($x \to \phi(x)$)
2. Rewrite all training / inference w/ only inner products between features $\phi(x_i)^T \phi(x_j)$
3. Write kernel function that computes inner products between high-dim features
$k(x_i, x_j) = \phi(x_i)^T \phi(x_j)$

2 conditions for kernel

1. k has inner product rep: $\exists \Phi: \mathbb{R}^d \to \mathcal{H}$ st $\forall x_i, x_j \in \mathbb{R}^d \quad k(x_i, x_j) = \langle \Phi(x_i), \Phi(x_j) \rangle$
2. for every sample $x_1 \cdots x_n \in \mathbb{R}^d$

$$K = \begin{bmatrix} k(x_1,x_1) & \cdots & k(x_1,x_n) \\ \vdots & k(x_i,x_j) & \vdots \\ k(x_n,x_1) & \cdots & k(x_n,x_n) \end{bmatrix} \text{ is PSD } (a^T K a \geq 0)$$

** cond 1 implies cond 2 **

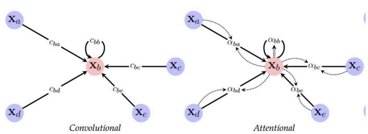
conv layer output: $H' = \left\lfloor \frac{H - K + 2P}{S} \right\rfloor + 1$

$S(z) = \frac{1}{1+e^{-z}} \to \frac{\partial S(z)}{\partial z} = S(z)(1 - S(z))$

$\frac{d}{dx} f(x)$ where $f(x)$ is PDF: $-x f(x)$

Graph Neural Networks

Msg passing w/ either convolutional / attentional mechanisms



$h_u^{(k)} = \phi\left(h_u^{(k-1)}, \bigoplus\{W^k h_v^{(k-1)} \mid v \in \mathcal{N}(u)\}\right)$ $\quad h_u^{(k)} = \phi\left(h_u^{(k-1)}, \bigoplus\{a(h_u^{(k-1)}, h_v^{(k-1)}) h_v^{(k-1)} \mid v \in \mathcal{N}(u)\}\right)$

Aggregation function: permutation invariant

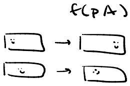
* will it change if you permute input? → No: $f(PA) = f(A)$; → yes

Neural network: permutation equivariant

permutation of argument = same permutation output

* $f(PA) = Pf(A)$

* translational equivariance — has diff result
rotational invariance — has same result



Langevin MCMC

score-based generative models: $S_\theta(x) \approx \nabla_x \log P_{data}(x)$

Langevin dynamics to sample from dist & get new samples: $\boxed{x_{t+1} = x_t + \eta \nabla_x \log P_{data}(x_t) + \sqrt{2\eta}\, z_t \quad \text{where } z_t \sim N(0,1)}$

Learning score-based models:

1. Max likelihood: $\min_\theta E_{P_{data}}[\log P_\theta(x)]$
2. Score matching: $\min_\theta E_{P_{data}}[\|\nabla_x \log P(x) - S_\theta(x)\|^2]$
3. Denoising approaches

Least Squares Denoising

Easier to sample if we
1. Add Gaussian noise
2. Sample from noisy distribution
3. Denoise noisy to clean

$\mathcal{L}(\ell(y)) = \int \|x - \ell(y)\|^2 p(x) p(y|x)\, dx$

$\ell^*(y) = E[x|y] = \int x\, p(x|y)\, dx$

$= \int x \frac{P(y|x)P(x)}{P(y)} dx$

$= \frac{\int x P(y|x) p(x)\, dx}{P(y)}$

$\nabla_y \log P(y) = \frac{\nabla_y P(y)}{P(y)} = \frac{\int \nabla_y P(y|x) p(x) dx}{P(y)}$

* $y = x + z \to$

$P(y|x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(y-x)^2}{2\sigma^2}\right]$

$\boxed{\ell^*(y) = y + \sigma^2 \nabla \log P(y)}$

Score func: $\nabla_y \log P(y) = \frac{1}{\sigma^2}[\ell^*(y) - y]$

Langevin to sample:

$y_{t+1} = y_t + \frac{\eta}{\sigma^2}[\ell^*(y_t) - y_t] + \sqrt{2\eta}\,\varepsilon_t$

step size (η); Normal (ε_t)