Vector Calculus

$f: \mathbb{R}^n \to \mathbb{R}$ $\quad a \in \mathbb{R}^n$

$\frac{df}{dx}(a) \in \mathbb{R}^{1\times n}$ (row vector)

Taylor's approximation:

$f(x) \approx f(a) + \left[\frac{df(a)}{dx}\right](x-a)$

→ Gradient is derivative transposed

$f(x) \approx f(a) + [\nabla_x f(a)]^T (x-a)$

for matrices:

$f(x) \approx f(A) + \langle \nabla_x f(A), X-A\rangle$

$\approx f(A) + Tr(\nabla_x f(A)^T (X-A))$

$[\nabla_x F(x)]_{ij} = \frac{\partial F(x)}{\partial x_{ij}}$

$\nabla_x F(x) = \begin{bmatrix} \frac{\partial F(x)}{\partial x_{11}} & \cdots & \frac{\partial F(x)}{\partial x_{1m}} \\ \frac{\partial F(x)}{\partial x_{n1}} & \cdots & \frac{\partial F(x)}{\partial x_{nm}} \end{bmatrix}$

Hessian: derivative of the gradient

$[\nabla_x^2 F(x)]_{ij} = \frac{\partial^2 F(x)}{\partial x_i \partial x_j} = \frac{\partial}{\partial x}[\nabla_x F(x)]$

best quadratic approximation:

$f(x) \approx f(a) + [\nabla_x f(a)]^T (x-a) + \frac{1}{2}(x-a)^T [\nabla_x^2 f(a)](x-a)$

derivative chain rule

$\frac{d(f\circ g)(x)}{dx} = \left[\frac{df(g(x))}{dz}\right]\cdot\left[\frac{dg(x)}{dx}\right]$

examples

- $\frac{d(w^T x)}{dx} = w^T = [\nabla_x (w^T x)]^T$
- $\frac{d(x^T A\ y)}{dx} = (A + A^T)x$
- $\nabla_x^2 (x^T A x) = A + A^T$
- $\nabla_x \|Ax - y\|_2^2 = 2A^T(Ax-y)$
- $\nabla_A\ v^T A v = vv^T$
- $\nabla_A \|Ax-y\|_2^2 = 2(Ax-y)x^T$
- $\nabla_x LSE(x) = \nabla_x \ln\left(\sum_{i=1}^n e^{x_i}\right) = \frac{1}{\sum_{i=1}^n e^{x_i}}\begin{bmatrix} e^{x_1} \\ \vdots \\ e^{x_n}\end{bmatrix}$
- $\nabla_x^2 LSE(x) = diag(\nabla_x LSE(x)) - [\nabla_x LSE(x)][\nabla_x LSE(x)]^T$

Miscellaneous

$x^T A y = \sum_i \sum_j a_{ij} x_i y_j$

$\begin{bmatrix} x \\ y\end{bmatrix}^T \begin{bmatrix} A & B \\ C & D\end{bmatrix}\begin{bmatrix} x \\ y\end{bmatrix} = x^TAx + x^TBy + y^TCx + y^TDy$

Lagrangian

$L(\vec{x}, \vec{\lambda}, \vec{\nu}) = f_0(\vec{x}) + \sum_{i=1}^m \lambda_i f_i(\vec{x}) + \sum_{j=1}^p \nu_j h_j(\vec{x})$

→ for $p^* = \min_{x\in\mathbb{R}^n} f_0(\vec{x})$ s.t. $f_i(\vec{x}) \le 0$, $h_j(\vec{x}) = 0$

→ if you have a constrained optimization problem, you can swap it with its Lagrangian to find its argmin/max

Positive Semidefiniteness

$A \in \mathbb{R}^{n\times n}$ symmetric

① $\forall x \in \mathbb{R}^n \quad x^TAx \ge 0$

② $\forall \lambda_i \ge 0$ s.t. $Av_i = \lambda_i v_i$

③ $\exists U \in \mathbb{R}^{n\times n}$ s.t. $A = UU^T$

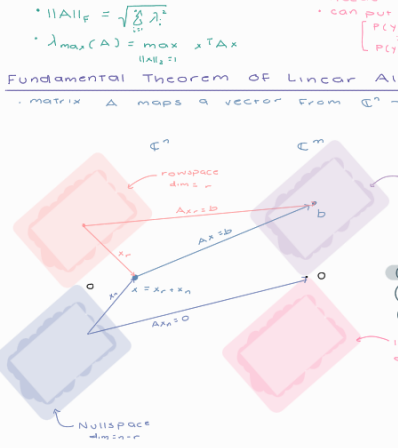
} definition of $A \succeq 0$

Properties of PSD matrices

- $A \succeq 0$ & $B \succeq 0 \Rightarrow \alpha A + \beta B \succeq 0$
- $A \succeq 0 \Rightarrow A_{ii} \ge 0 \ \forall i \in [n]$
- $A \succeq 0 \Rightarrow \sum_{i=1}^n \sum_{j=1}^n A_{ij} \ge 0$
- $A \succeq 0$ & $B \succeq 0 \Rightarrow Tr(AB) \ge 0$
- $A \succeq 0$ & $B \succeq 0 \Leftrightarrow Tr(AB) = 0 \Leftrightarrow AB = 0$
- $\|A\|_F = \sqrt{\sum_i \lambda_i^2}$
- $\lambda_{max}(A) = \max_{\|x\|_2=1} x^TAx$

Fundamental Theorem of Linear Algebra

- matrix A maps a vector from $\mathbb{C}^n \to \mathbb{C}^m$



- rowspace (X) = columnspace (X^T)
- rowspace $(X) \perp$ nullspace (X)
- $N(X) = N(X^TX)$
- columspace (X^TX) = rowspace (X^TX) = rowspace (X)

Probability

Bernoulli: biased coin flip [Bern(p)]

$P_X(x) = \begin{cases} p & \text{if } x=1 \\ (1-p) & \text{o/w}\end{cases}$ $\quad \mathbb{E}[X] = p$ $\quad Var[X] = p(1-p)$

Bin(n,p): number of heads (k) in n biased flips

$P_X(k) = \binom{n}{k} p^k (1-p)^{n-k}$ $\quad \mathbb{E}[X] = np$ $\quad Var[X] = np(1-p)$

Pois(λ): # mutations k in population w/mean mutation rate λ over fixed time interval

$P_X(k) = \frac{e^{-\lambda}\lambda^k}{k!}$ $\quad \mathbb{E}[X] = \lambda$ $\quad Var[X] = \lambda$

Unif([a,b]):

$P_X(k) = \frac{1}{b-a+1}$ $\quad \mathbb{E}[X] = \frac{a+b}{2}$ $\quad Var[X] = \frac{[(b-a+1)^2 - 1]}{12}$

Multivariate Gaussians

- density of N-dim MVG $\sim\mathcal{N}(\mu, \Sigma)$:

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

- IF $X \in \mathbb{R}^d$ is distributed as an MVG, then $\forall i,j \in \{1,\dots,d\}$ $cov(X_i, X_j) = 0$ iff X_i, X_j independent
- Σ for MVG is symmetric & PSD $\Rightarrow \lambda_i \ge 0 \ \forall i$
- a random vector follows the MVG distribution iff
  - each coordinate marginally & conditionally Gaussian
  - $Z_1 \sim \mathcal{N}(\mu_1, \sigma_1)$ & $Z_2 | Z_1 \sim \mathcal{N}(\mu_{2|1}, \sigma_{2|1}^2)$

MLE

- goal: find a hypothesis model that maximizes the probability of the data
- parameterize the set of hypothesis models with θ:

$\hat{\theta}_{mle} = \arg\max_\theta \mathcal{L}(\theta; D) = p(data = D \mid true\ model = \theta)$

$= \arg\max_\theta \log \prod_{i=1}^n p(y_i \mid x_i, \theta)$

$= \arg\max_\theta \sum_{i=1}^n \log(p(y_i \mid x_i, \theta))$

Gaussian log-likelihood $y_i | \theta \sim \mathcal{N}(h_\theta(x_i), \sigma^2)$

$\hat{\theta}_{mle} = \arg\max_\theta -\sum_{i=1}^n \frac{(y_i - h_\theta(x_i))^2}{2\sigma^2} - \ln(\sqrt{2\pi}\sigma)$

$= \arg\min_\theta \sum_{i=1}^n (y_i - h_\theta(x_i))^2$

→ now, in the case of regression, $h_\theta(x_i) = x_i^T\theta$

$\hat{\theta}_{mle} = \arg\min_\theta \sum_{i=1}^n (y_i - x_i^T\theta)^2 = (X^TX)^{-1}X^Ty$

→ which is just OLS!

- MLE properties:
  - consistent (more data → convergence of true θ value for D)
  - asymptotically efficient (least variance parameter estimates)
  - value of $p(D|\theta_{mle})$ is invariant to reparameterization
    - eg $\mathcal{N}(x|\mu, \sigma)$ vs $\mathcal{N}(x|z\mu, \sigma)$
  - can yield a parameter estimate even when data wasn't generated from family
  - only get a point estimate of parameter instead of distribution (which captures uncertainty)
  - MLE equivalent to maximizing cross-entropy / minimizing relative entropy

Linear Regression

examples

Gaussian Linear Regression $p(y|x) = \mathcal{N}(y | w^Tx, \sigma^2)$ → OLS

$\theta_{mle} = (w_{mle}, \sigma^2_{mle}) = \arg\max_{w,\sigma^2} \log p(D|\theta)$

$L_w = \|y - Aw\|_2^2$; $\sigma^2_{mle} = \frac{1}{N}\sum_i (y_i - w^Tx_i)^2$

$w_{mle} = (A^TA)^{-1}A^Ty$

not invertible if columns in A are linearly dependent → when this happens, so many solns to w_{mle} (underdetermined)

Ridge Regression

Loss becomes $L_w = \|y - Aw\|_2^2 + \lambda\|w\|_2^2$ → makes you choose w_{rr} as the w_{mle} with the smallest norm (with smaller coefficients the model is less sensitive to noise)

$w_{rr} = (A^TA + \lambda' I)^{-1}A^Ty$ ($\lambda' \ge 0$ → $(A^TA + \lambda' I)$ invertible)

MAP: Maximum a Posteriori

- Find a point estimate that maximizes the posterior probability

$\theta_{MAP} = \arg\max_\theta p(\theta|D)$

$= \arg\max_\theta p(D|\theta)\,p(\theta)$

- Laplace prior ⟹ L1 regularization
- Gaussian prior ⟹ L2 regularization

MAP for linear regression using Gaussian prior

- zero mean prior $p(w) = \mathcal{N}(w; 0, \lambda I)$
- Bayesian posterior $p(w|D) = p(D|w)p(w)$

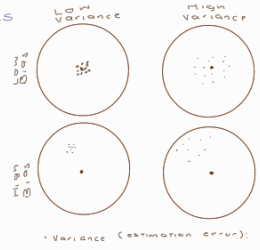
$w_{map} = \arg\max_w \log p(D|w)p(w) = \arg\max_w \log p(D|w) + \log \mathcal{N}(w; 0, \lambda I)$

$= \arg\max_w \sum_{i=1}^N \log \mathcal{N}(y_i; w^Tx_i, \sigma^2) + \sum_{i=1}^d \log\left(\frac{1}{\sqrt{2\pi}\lambda}\exp\left(-\frac{w_i - 0}{2\lambda}\right)^2\right)$

$= \arg\min_w \frac{1}{2\sigma^2}(y-Aw)^T(y-Aw) + \sum_{i=1}^d \frac{w_i^2}{2\lambda}$

$= \arg\min_w (y-Aw)^T(y-Aw) + \frac{\sigma^2}{\lambda}\|w\|_2^2$

Bias-Variance Tradeoff



- Bias of a model: $\mathbb{E}[f(\hat{x}) - f(x)]$
- Variance (estimation error) of a model: $V[f(\hat{x})] = \mathbb{E}[(f(\hat{x}) - \mathbb{E}[f(\hat{x})])^2]$
- MSE of a model: $\mathbb{E}[(f(\hat{x}) - f(x))^2]$
  $= \text{bias}^2 + \text{variance}$
  $= \mathbb{E}[f(\hat{x}) - f(x)]$

Natural Log Rules

- $\ln(1) = 0$
- $\ln(a^b) = b\ln a$
- $\ln(ab) = \ln a + \ln b$
- $\ln\left(\frac{a}{b}\right) = \ln a - \ln b$

Regression Fcns

① Linear: $h(x; w, a) = wx + a$

② Polynomial: equiv. to linear w/added polyn. feat.

③ Logistic: $h(x; w, a) = s(wx + a)$, $s(y) = \frac{1}{1+e^{-y}}$

Loss Fcns: z = prediction $h(x)$; y true label

Ⓐ Squared error: $L(z, y) = (z-y)^2$

Ⓑ Absolute error: $L(z, y) = |z-y|$

Ⓒ Cross-entropy*: $L(z, y) = -y\ln z - (1-y)\ln(1-z)$

cost fcns to minimize

Ⓐ Mean loss: $J(h) = \frac{1}{n}\sum_i L(h(x_i), y_i)$

Ⓑ Max loss: $J(h) = \max_i L(h(x_i), y_i)$

Ⓒ Weighted sum: $J(h) = \sum_i w_i L(h(x_i), y_i)$

Ⓓ L2 penalized: $J(h) =$ Ⓐ or Ⓑ or Ⓒ $+ \lambda\|w\|_2$

Ⓔ L1 penalized: $J(h) =$ Ⓐ or Ⓑ or Ⓒ $+ \lambda\|w\|_1$

Softmax Regression: classification

- logistic regression → 2 classes

$\begin{bmatrix} P(Y=1|X) \\ P(Y=0|X)\end{bmatrix} = \begin{bmatrix} M \\ 1-M\end{bmatrix} = \begin{bmatrix} \frac{1}{1+\exp(-\beta^Tx)} \\ \frac{\exp(-\beta^Tx)}{1+\exp(-\beta^Tx)}\end{bmatrix}$

- softmax/multinomial logistic regression
- needs iterative optimization (eg: ∇ descent)
- can put at top of NN to get softmax loss

$\begin{bmatrix} P(y=1|x) \\ P(y=0|x)\end{bmatrix} = \frac{1}{\sum_{c=1}^2 e^{\beta_c x}}\begin{bmatrix} e^{\beta_1 x} \\ e^{\beta_2 x}\end{bmatrix}$

$P(y=i|x) = \frac{e^{\text{logit}_i}}{\sum_j e^{\text{logit}_j}}$

$\text{logit}_i = \log\left(\frac{P(y=i|x)}{P(y\ne i|x)}\right)$

Famous Regression Methods

- Least-sq. linear regr: ① + Ⓐ + ⓐ
- Weighted Least-sq. linear regr: ① + Ⓐ + Ⓒ } quadr. cost → minimize w/calculus
- ridge regression: ① + Ⓐ + ⓓ
- LASSO: ① + Ⓐ + Ⓔ } Quadratic Program
- Logistic regression: ③ + Ⓒ + ⓐ } convex cost → minimize w/gradient descent

Gradient Descent

① choose $x_0 \in \mathbb{R}^d$ & $k=0$

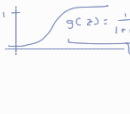
② choose $t_k > 0$ & $x_{k+1} = x_k - t_k \nabla f(x_k)$

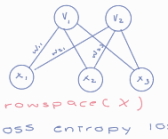
③ Repeat

Neural Networks

- one layer

$y_i = g\left(\sum_k w_{jk} x_k\right)$

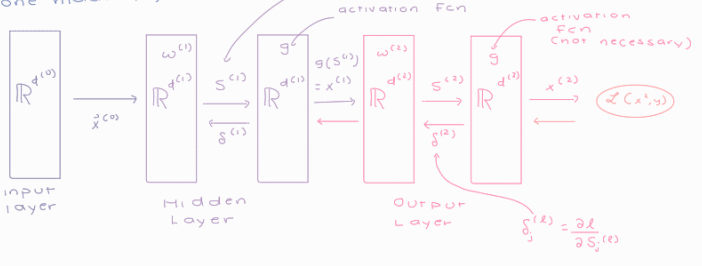




- cross entropy loss:

$L = -\sum_{\text{input data}} (y_i \ln v_i + (1-y_i)\ln(1-v_i))$

- one hidden layer



$S_j^{(\ell)} = \sum_{i=1}^d w_{ij}^{(\ell)} x_i^{(\ell-1)}$

$\delta_j^{(\ell)} = \frac{\partial L}{\partial S_j^{(\ell)}}$

PCA

- n data points, dimension d
- $X \in \mathbb{R}^{n\times d}$
- Procedure:
  ① Zero-mean the matrix X to get $\bar{X}$
  ② Covariance matrix: $\Sigma = \bar{X}^T\bar{X}$
  ③ Eigendecomposition, compute $\Sigma = QDQ^T$
  ④ Keep k eigenvectors $Q_k := Q_{:,1:k}$ with most variance (highest eigenvalues in D)
  ⑤ Project your points down to this subspace

$\bar{X}_k = \bar{X}Q_k \in \mathbb{R}^{n\times k}$, these are your principal components

- if we want our data in the original dimension but only using info retained from PCA, we expand back from PCA to original basis:

$\bar{X}_{reconstruct} = \bar{X}_k Q_k^T \in \mathbb{R}^{n\times d}$

- Algorithm 2 (SVD)
  ① Pre-process: reduce mean, normalize variance
  ② Calculate SVD of X: $X = U\Sigma V^T$
  ③ New basis is 1st k right singular vectors (1st k columns of V)

Explained variance

for $X^TX = QDQ^T$, the explained variance V_i for some eigenvalue $(\lambda_i = D_{i,i})$ is:

$V_i = \frac{D_{ii}}{\sum_j D_{jj}} = \frac{\lambda_i}{\sum_j \lambda_j}$

PCA limitations

- assumes only μ & Σ affect distribution (ie, Gaussian)
- due to orthogonality, only exploits linear redundancy
- isn't robust to scale of features/outliers
- unsupervised

Neighborhood Embedding (NE) / Isomap

Basic NE

① For each point, find nearest neighbors & draw edge btwn them

② Compute distances btwn points (using only edges)

③ Given this set of pairwise distance, put into matrix M & perform MDS/PCA

Stochastic NE

① Probably x_i chooses x_j as its neighbor:

$P_{j\leftarrow i} = \frac{\exp\{-\|x_i - x_j\|^2/2\sigma_i^2\}}{\sum_{k\ne i}\exp\{-\|x_i - x_k\|^2/2\sigma_i^2\}}$ $\quad P_{j\leftarrow j} = 0$

② Symmetrize & Normalize

$P_{ij} = P_{i|j} = \frac{1}{2n}(P_{j\leftarrow i} + P_{i\leftarrow j})$

$P = \{P_{ij}\}$

③ Set σ_i^2 adaptively s.t. entropy of $P_{\cdot\leftarrow i} = \sum_j P_{j\leftarrow i}\log(P_{j\leftarrow i})$

④ Posit low-dim representations $Y \in \mathbb{R}^{n\times k}$ & define stochastic neighborhoods for them $Q = \{Q_{ij}\}$

$Q_{ij} = \frac{\exp\{-\|y_i - y_j\|^2\}}{\sum_{k\ne l}\exp\{-\|y_k - y_l\|^2\}}$ (more $\sigma = \frac{1}{2}$)

Goal: Find Y s.t. SN structure preserved → Join: KL-div

⑤ $\hat{Y} = \arg\min_Y KL(P\|Q) = \sum_{ij} P_{ij}\log\frac{P_{ij}}{Q_{ij}}$

→ use gradient descent to find embedded pts $\{y_i\}$

→ $\frac{\partial Loss}{\partial y_i} = \sum_j (P_{ij} - Q_{ij})(y_i - y_j)$

→ nonconvex → many local minima

t-SNE

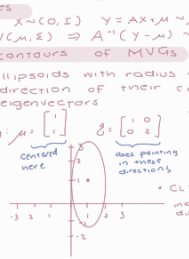
- new $Q_{j\leftarrow i} = \frac{(1+\|y_i - y_j\|^2)^{-1}}{\sum_{k\ne l}(1+\|y_k - y_l\|^2)^{-1}}$
- everything else stays the same; this prevents crowding in low-dim space

Key MVG properties

① $X_1, \dots, X_n \overset{\text{indep}}{\sim} \mathcal{N}(0,1)$ $X \sim \mathcal{N}(0, I)$ $\quad Y = AX + \mu \sim \mathcal{N}(\mu, AA^T)$

② if $\Sigma \succ 0$ & $Y \sim \mathcal{N}(\mu, \Sigma) \Rightarrow A^{-1}(Y - \mu) \sim \mathcal{N}(0, I)$

- isocontours of MVGs



- ellipsoids with radius $\sqrt{\lambda_i}$ in direction of their corresponding eigenvectors
- eg: $\mu = \begin{bmatrix} 1 \\ 1\end{bmatrix}$, $\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 2\end{bmatrix}$
- CLT: $\sum$ (large # indep RVs) → Gaussian distribution

- in order for a zero-mean RV $Z = [Z_1, Z_2]^T$ to be JG:
  - Z_1 & Z_2 marginally Gaussian
  - $Z_1|Z_2$ & $Z_2|Z_1$ are Gaussian
  - can be written as $Z = AX$ where X is iid std normal RVs
- two Gaussian RVs can be uncorr. but not indep

- $X \sim \mathcal{N}(\mu, \Sigma) \Leftrightarrow AX + b \sim \mathcal{N}(A\mu + b, A\Sigma A^T)$
- $MGF(x) = \mathbb{E}[\exp\{\lambda^T X\}] = \exp\{\lambda^T\mu + \frac{1}{2}\lambda^T\Sigma\lambda\}$
- $Z = \sqrt{\Sigma}^{-1}(X - \mu) \sim \mathcal{N}(0, I_n)$

Diagonalizing an Ellipse

① Compute eigenvalues & eigenvectors of A

② $Q = \begin{bmatrix} v_1/\|v_1\| & v_2/\|v_2\| \\ | & | \end{bmatrix}$ (orthonormal) $\quad D = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2\end{bmatrix}$

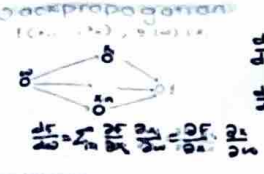
③ Change coordinate system along eigenvectors

$\begin{bmatrix} x_1' \\ x_2'\end{bmatrix} = Q^T\begin{bmatrix} x_1 \\ x_2\end{bmatrix}$

→ Axis points along eigenvectors (major & minor axis lengths are $1/\sqrt{\lambda_1}$ & $1/\sqrt{\lambda_2}$)

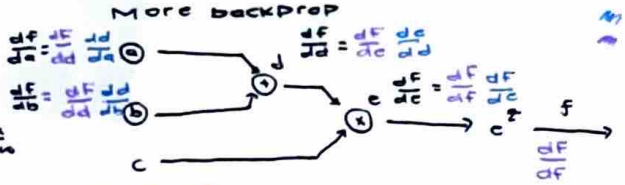
⇒ Finding coordinate system axis-aligned ≡ diagonalizing A

Backpropagation



$\frac{df}{dx} = \sum_{i=1}^{n} \frac{df}{dy_i}\frac{dy_i}{dx} = \frac{df}{dy}\frac{dy}{dx}$

More backprop



$\frac{df}{da} = \frac{df}{dc}\frac{dc}{da}$

$\frac{df}{db} = \frac{df}{dc}\frac{dc}{db}$

$\frac{df}{dc} = \frac{df}{dd}\frac{dd}{dc}$

$\frac{df}{dc} = \frac{df}{de}\frac{de}{dc}$

$\frac{df}{de} = \frac{df}{dc}\frac{dc}{de}$

= compute during forward pass

= computed during backward pass

Convolution

- Image $I * J$ color channels I_1, I_2, I_3 (each channel size $W \times H$)
- Filter $G * J$ color channels G_1, G_2, G_3 (each mask size $k \times k$)
- convolution operation at each (x,y):

$$(I*G)[x,y] = \sum_{c=1}^{3} \sum_{x'=x}^{x+k} \sum_{y'=y}^{y+k} I_c[x',y'] \, G_c[x'-x, y'-y]$$

- size of output of convolution
  - no stride: $(W-k+1) \times (H-k+1)$
  - stride L: $\lfloor (W-k)/L \rfloor + 1 \times \lfloor (H-k)/L \rfloor + 1$

$$\frac{\partial L}{\partial G_c[x',y']} = \sum_{x=1}^{W-k+1} \sum_{y=1}^{H-k+1} \frac{\partial L}{\partial R[x,y]} \frac{\partial R[x,y]}{\partial G_c[x',y']}$$

$$= \sum_{x=1}^{W-k+1} \sum_{y=1}^{H-k+1} \frac{\partial L}{\partial R[x,y]} \frac{\partial}{\partial G_c[x',y']} \sum_{x''=1}^{k} \sum_{y''=1}^{k} I_c[x+x'', y+y''] G_c[x'', y'']$$

$$= \sum_{x=1}^{W-k+1} \sum_{y=1}^{H-k+1} \frac{\partial L}{\partial R[x,y]} I_c[x+x', y+y']$$

- Max pooling, output: R or (i,j)

$$R[i,j] = \max_{x=i*k,\ldots,(i+1)*k} \; \max_{y=j*k,\ldots,(j+1)*k}$$

- conv layers share weights
- # parameters in conv layer depends on # of filters, filter width, # of color channels in previous layer
- CNNs have more than FCNs

Backprop takes $O(\#\text{edges})$ time

- convolutions - translationally equivariant (shift input ⇒ shift output fts)
- NN w/ identity fcn as activation fcn is just linear regression
- 2D conv: image $D \times D$, N filters $K \times K$. No padding, output from 1 filter has size $(D-K+1)(D-K+1)$ ⇒ for all N, multiply by N.

Vanishing Gradient

gain:

① normalize data in mini-batch

$$\hat{z}^{(i)} = (z^{(i)} - \mathbb{E}[z^{(i)}]) / \mathrm{Var}(z^{(i)})$$

② add scale & shift params

$$h^{(i)} = \sigma(\gamma \hat{z}^{(i)} + \beta)$$

K-means clustering:

① Partition n points into K disjoint clusters

cluster i's mean $= \mu_i = \frac{1}{n_i} \sum_{j:i} x_j$

⇒ Find y that min $\sum_{i=1}^{K} \sum_{y_j=i} \|x_j - \mu_i\|_2^2$

- K-means heuristic:
  - ⇒ Alt. btwn
    - ① y_i fixed; update μ_i
    - ② μ_i fixed; update y_i

Classification:

- principle: minimize $\mathbb{P}[\text{error}]$

$$\mathbb{P}[\text{error}] = \int p(\text{error}|x)\, p(x)\, dx$$

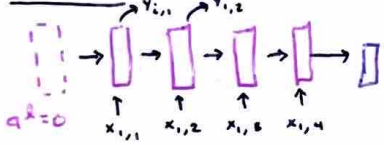
ResNets:

- skip connections!

$$x^{(l)} = \sigma(z^{(l-1)}) + x^{(l-1)}$$

(instead of $x^{(l)} = \sigma(z^{(l-1)})$)

RNNs:



- each layer

$$\hat{a}^{l-1} = \begin{bmatrix} a^{l-1} \\ x_{i,t} \end{bmatrix}$$

$$z^{l} = W^{l} \hat{a}^{l-1} + b^{l}$$

$$a^{l} = \sigma(z^{l})$$

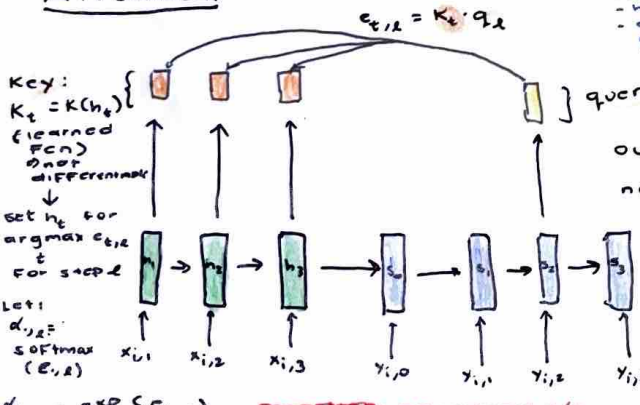
$$\hat{y}^{l} = F(a^{l})$$

} variable # inputs

} decoder ⇒ variable # outputs

↳ autoregressive gen. model (gen. each new word $\hat{y}_{i,t}$ having fed in previous ones)

Attention



$e_{t,l} = K_t \cdot q_l$

Key: $K_t = K(h_t)$ (learned fcn) (not differentiable)

set h_t for $\text{argmax}_t e_{t,l}$ for step l

Let: $\alpha_{t,l} = \text{softmax}(e_{t,l})$

$$\alpha_{t,l} = \frac{\exp(e_{t,l})}{\sum_{t'} \exp(e_{t',l})}$$

send $a_l = \sum_t \alpha_{t,l} h_t$

query: $q_l = q(s_l)$

output: $\hat{y}_l = f(s_l, a_l)$

next decoder step: $s_l = \begin{bmatrix} s_{l-1} \\ \alpha_{l-1} \\ y_l \end{bmatrix}$

- multi-head attn: have mult. keys, queries & values
- self-attn is linear
- masked attn typically only used in decoders (when computing attn weights @ time t, weights corr. to times $> t$ are set to 0)
- self-attn runtime $O(d^2 n + n^2 d)$
- self-attn α value computation:

$$\alpha(Q,K,V) = \text{softmax}(QK^T)V$$

- DT ensemble w/ bagging has higher variance than rand. forest
- bagging reduces variance, not necessarily bias/training acc.
- random forests tend to ↑ bias

learnable params in NN:

- each image has $28 \times 28 = 784$ fts
- 2 hidden layers w/ 100 hidden units each
- all layers (incl. output) have a bias term

⇒ $\underbrace{784 \cdot 100 + 100}_{\text{layer 1}} + \underbrace{100^2 + 100}_{\text{layer 2}} + \underbrace{101}_{\text{output}}$

- For a FC layer l with $n_i(l)$ inputs & $n_o(l)$ outputs, ω_l contains $n_i(l) \times n_o(l)$ parameters

Entropy & Information

- entropy: = "expected surprise"

$$H(Y) := \mathbb{E}[-\log P(Y)]$$

$$= -\sum_{k} P(X=k) \log(P(Y=k))$$

- conditional entropy

$$X_{j,v} := \begin{cases} 1 & \text{if } x_j \le v \\ 0 & \text{o/w} \end{cases}$$

$H(Y|X_{j,v}=1)$ = entropy of distribution of pons s.t. $x_j \le v$

- information gain / mutual information:

$$I(X_{j,v}; Y) := H(Y) - H(Y|X_{j,v})$$

$$= H(Y) - P(X=1) H(Y|X=1) - P(X=0) H(Y|X=0)$$

- high entropy:
  - ↳ Y from a uniform dist.
  - ↳ sampled values less predictable
  - ↳ flat histogram

- $\hat{\omega}_{rr} = (X^T X + \lambda I)^{-1} X^T y$
- $\hat{\omega}_{ols} = (X^T X)^{-1} X^T y$
- $\hat{\lambda}_{mle} = \frac{1}{n} \sum_{i=1}^{N} x_i$ ⇒ unbiased
- $\hat{\lambda}_{map} = \underbrace{\frac{n}{n+\varepsilon}}_{<1} \hat{\lambda}_{mle}$ ⇒ biased

Nearest Neighbors

- non-parametric
- used for regression & classification
- K-NN algo (regression):
  - ↳ Training:
    - ① store $\forall (x_i, y_i)$ in dataset D
  - ↳ Testing:
    - input: x_q (query), K
    - ① Find closest $(\tilde{x}_1, \ldots, \tilde{x}_K)$ in training dataset
    - ② set $\hat{y} = \frac{1}{K} \sum_{j=1}^{K} f(\tilde{x}_j)$
    - output: $\hat{y}$ (prediction).
- K-NN algo (classification)
  - ↳ Training:
    - ① store $\forall (x_i, y_i)$ in D
  - ↳ Testing:
    - input: x_q, K
    - ① Find closest $(\tilde{x}_1, \ldots, \tilde{x}_K)$
    - ② $\hat{y}$ = majority vote of $(f(\tilde{x}_1), \ldots, f(\tilde{x}_K))$
    - output: $\hat{y}$
- Nearest neabors (in the limit)
  - $\mathcal{E}^*(x)$:= avg. error of Bayes classifier
  - $\mathcal{E}_{nn}(x)$:= error of NN
  - + Thm 1: $\lim_{n \to \infty} \mathcal{E}_{NN}(x) \le 2\mathcal{E}^*(x)$
  - + Thm 2: $\lim_{\substack{n \to \infty \\ K \to \infty}} \mathcal{E}_{KNN}(x) = \mathcal{E}^*(x)$ if $\frac{K}{n} \to 0$
- Pros of NN:
  - Fast training
  - learns complex fcns easily
- cons of NN:
  - slow @ test time
  - high storage cost
  - curse of dimensionality (high dim data hard to effectively measure distance)

- Bayes rule:

$$P(Y|X) = \frac{P(X|Y)\,P(Y)}{P(X)}$$

$$P(X) = \sum_i P(X|Y=i)$$

RVs Z, W indep: $\mathbb{E}[ZW] = \mathbb{E}[Z]\mathbb{E}[W]$

Decision Trees

- work well w/ default hyperparams
- highly interpretable
- NNs generally may perform better
- axis-aligned (can't easily model diagonal boundary)
- model that makes predictions by posing a series of simple tests on the point
- trained in greedy, recursive fashion, downward from root
- goal: at each recursion, find the Feature & split that minimizes conditional entropy (maximizing info. gain)
- Algorithm for building DTs:
  - ① start w/ empty tree
  - ② for each node:
    - IF stopping condition reached:
      - leaf label = avg. of data at that node
    - else:
      - split by next best attribute $\text{argmax}\, I(X_i) = \text{argmax}\, H(Y) - H(Y|X_i)$
      - recurse to child nodes
- stopping conditions:
  - limited depth: don't split if node if beyond some fixed depth
  - node purity: don't split if proportion of training point in some class is sufficiently high
  - info gain criteria: don't split if info gain/purity is sufficiently close to 0
  - ↳ problem: using greedy heuristic (XOR)
- pruning:
  - backward-greedy: greedily remove node & descendants that most improves validation performance

Ensembling Decision Trees

- slightly diff samples ⇒ v diff trees (high variance models)
- bagging (bootstrap aggregation)
  - train M models, each with n' (usually $n'=n$) samples, sampled w replacement
  - avg M predictions to get bagged prediction ($\hat{y} = \frac{1}{m} \sum_m^M G_m(x)$)
- random forests - same as bagging except at each split, choose only a random subset p' (usually $p' = \sqrt{p}$) features to split on
- boosting: reweight training points to emphasize ones incorr. classified
  - algo:
    - ① train next model conditioned on all prev models & their weights
    - ② reweight models to minimize loss: $\hat{y} = \sum_{m=1}^{M} \alpha_m G_m(x)$
    - ③ repeat
- random forests w/ decision trees of same depth have lower variance than single decision tree

Bias-Variance : $\hat{\lambda}$: estimate constructed from samples of X; x': new sample

$E[(\hat{\lambda} - x')^2] = E[(\hat{\lambda} - \mu)^2] + E[(x' - \mu)^2]$
$= E[\hat{\lambda}^2 - 2\hat{\lambda}\mu + \mu^2] + Var(x')$
$= E[(\hat{\lambda} - \mu)^2] + \sigma^2 = E[(\hat{\lambda} - E[\hat{\lambda}] + E[\hat{\lambda}] - \mu)^2] + \sigma^2$
$= E[(\hat{\lambda} - E[\hat{\lambda}])^2] + E[\hat{\lambda} - \mu]^2 + \sigma^2$
$= Var(\hat{\lambda}) + bias^2 + \sigma^2$

$E[(\hat{\lambda} - \mu)^2] = (E[\hat{\lambda}] - \mu)^2 + Var(\hat{\lambda})$
$= bias^2(\hat{\lambda}) + bias^2$

- bias: error due to inability of hypothesis h to fit g perfectly
- variance: error due to fitting random noise in data
  - adding ft usually ↑ variance (don't want to add ft. unless it reduces bias more)
- noise in test set only affects $Var(\hat{\lambda})$ but noise in training set can affect bias & variance
- adding bad ft rarely increases bias (∃ set coeff = 0)

eg: semantic segm, node classification
eg: noise/wrong classification

Markov Models

$Q = q_1 q_2 \dots q_N$: set of N states

$A = a_{11} a_{12} \dots a_{N1} \dots$: Transition prob matrix A, each a_{ij} repr. $P(q_j \mid q_i)$ s.t. $\sum_{j=1}^{n} a_{ij} = 1$

$\pi = \pi_1, \dots, \pi_N$: initial prob distribution over states
↳ π_i = prob MC will start in state i, ↳ $\sum_{i=1}^{N} \pi_i = 1$

→ HMMs have the above but also have

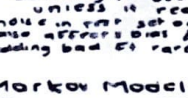
$O = o_1 \dots o_T$: sequence of T observations, each drawn from a vocab $V = v_1, v_2, \dots, v_V$

$B = b_i(o_t)$: sequence of observation likelihoods/ emission probabilities $P(o_t$ generated from state $i)$

- HMM probs:
  1) Likelihood: Given specified HMM, compute likelihood of observation sequence O
  2) Decoding: given HMM, find best seq. of hidden states
  3) Learning: given observation seq. O & set of states, learn HMM params A & B

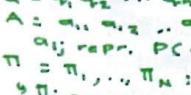
Probabilistic Graphical Models

- each node repr. random var. & edges repr. dependence relationships
- PAGs help us achieve tractability thru cond. independence



$P[R=T \mid G=T] = \frac{P[G=T, R=T]}{P[G=T]} = \frac{\sum_{x \in \{T,F\}} P[G=T, S=x, R=T]}{\sum_{x,y \in \{T,F\}} P[G=T, S=x, R=y]}$

$P[G=T, S=T, R=T] = P[G=T \mid S=T, R=T] P[S=T \mid R=T] P[R=T]$



Joint factorization $P_{S,X,Y,Z}(S,X,Y,Z) := P$
$P = P[X=x] P[Z=z] P[Y=y \mid X=x, Z=z] P[S=s \mid X=x, Y=y]$

- each node is conditionally indep. of all of its ancestor nodes, given all of its parents

$S \perp Z \mid Y$? No.
$S \perp X \mid Y$? No.

conditional independence: $X \perp Y \mid Z \Rightarrow P(X \mid Y, Z) = P(X \mid Z)$

Markov Decision Processes

- $S_t \in S$: state at timestep t. $A_t \in A$ action @time t, γ: discount factor
- $R_{t+1} \in R$: reward generated from A_t; S_{t+1} : state generated from A_t
- one step dynamics: $p(s', r \mid s, a) = P[S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a]$
- return following time t: $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} = R_{t+1} + \gamma G_{t+1}$
- policy π maps states to probability of actions: $\pi(a \mid s) = P[A_t = a \mid S_t = s]$
- state-value fcn v_π of state s under policy π: $v_\pi(s) = E_\pi[G_t \mid S_t = s]$ (expected future return when following π after starting at state s)
- action-value fcn $q_\pi(s,a)$ of taking action a in state s under policy π: $q_\pi(s,a) = E_\pi[G_{t+1} \mid S_t = s, A_t = a] = E_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a]$

$E_\pi[R_{t+1} \mid S_t = s] = \sum_r r P[r \mid s] = \sum_{s',a} r \, p(s', r \mid s, a) \pi(a \mid s)$

$v_\pi(s) = E_\pi[q_\pi(s,a)] = \sum_a q_\pi(s,a) \pi(a \mid s)$

$q_\pi(s,a) = E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] = \sum_{s',r} [r + \gamma v_\pi(s')] P[s', r \mid s, a]$

- Bellman expectation eqn:
$v_\pi(s) = \sum_a \pi(a \mid s) q_\pi(s,a) = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a) [r + \gamma v_\pi(s')]$
- optimal state-value fcn (Bellman optimality eqn)
$v_*(s) = \max_\pi v_\pi(s) = \max_a \sum_{s',r} p(s', r \mid s, a) [r + \gamma v_*(s')] = \max_a q_*(s,a)$

↳ idea: optimal policy yields max expected total reward

- Value iteration: compute optimal values of states by iterative updates until convergence (ie $v_{k+1}(s) = v_k(s)$)
  1. ∀s, initialize $v_0(s) = 0$
  2. ∀s ∈ S, until convergence: $v_{k+1}(s) = \max_a \sum_{s',r} p(s', r \mid s, a) [r + \gamma v_k(s')]$
- Policy iteration: use policy evaluation & extraction to iteratively converge to optimal policy; usually outperforms value iteration bc policies usually converge faster than state's values
  1. Define initial policy (can be arbitrary)
  2. Until convergence (ie $\pi_{i+1} = \pi_i$)
     A) Policy evaluation: $v_{k+1}(s) = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a) [r + \gamma v_k(s')]$
     B) Policy improvement: $\pi'(s) = \arg\max_a \sum_{s',r} p(s', r \mid s, a) [r + \gamma v_k(s')]$

Decision Theory

- classifier/decision rule $f: X \to \{1, \dots, K\}$ where X is space of inputs & we have K classes
- loss function $L(i,j)$ = penalty classifier receives for predicting class i when true class is j
- risk $R(f)$ returns avg loss of classifier for entire data distribution

$R(f) = E_{X,Y}[L(f(x), y)] = E_X[E_{Y|X}[L(f(x), y) \mid X]] = E_X[\sum_{j=1}^{K} L(f(x), j) P(Y = j \mid X)]$

$= \int_x \sum_{j=1}^{K} L(f(x), y) P(Y = j \mid X) P(x) \, dx = \int_x \sum_{j=1}^{K} L(f(x), j) P(X \mid Y = j) P(Y = j) dx$

- Bayes classifier: attains minimum risk. a classifier can achieve
- Bayes risk/error rate: lowest risk any classifier can attain

$f^*(x) = \arg\min_i E_{Y|X}[L(i, y) \mid X] = \arg\max_i P[Y = i \mid X]$

- equating the risks of 2 classes should help solve for Bayes classifier decision boundary, eg:

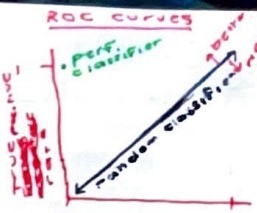
$\begin{bmatrix} \lambda_{11} & \lambda_{12} \\ \lambda_{21} & \lambda_{22} \end{bmatrix}$ λ_{ij} - loss for classifying j as i; ω_i - class i

$\lambda_{11} P[\omega_1 \mid x] + \lambda_{12} P[\omega_2 \mid x] = \lambda_{22} P[\omega_2 \mid x] + \lambda_{21} P[\omega_1 \mid x]$

$\frac{P[\omega_1 \mid x]}{P[\omega_2 \mid x]} = \frac{\lambda_{22} - \lambda_{12}}{\lambda_{11} - \lambda_{21}}$

Misc

- FC NNs - more expressive & have greater model capacity due to their incr # of weights (rel. to CNNs)
- CNNs - less weights & less data needed but need ↑ architecture params to work well
- capacity of a model is a hyperparam that we choose based on the error on a validation set
  - ↳ if capacity < optimal ⇒ underfitting
  - ↳ if capacity > optimal ⇒ overfitting
- convolution: translation equivariant
- global pooling: translation invariant
- self-attn: permutation equivariant
- GNN: permutation invariant bc of the aggregation fcn taking a set as input
- representational containment: conv ⊂ attn ⊂ msg pass (msg pass can be general msg passing allows arbitrarily learnable msg fcns but conv & attn pass extra constraints
  - ↳ translational equiv - useful for pixels & node-level tasks
  - ↳ rotational invariant - useful for graph & img-level tasks
- tasks:
  - ↳ GNNs (image-lvl): classification/regression, one output/target for entire img. eg: dog, cat
  - ↳ GNNs (pixel-lvl): semantic seg for classification of each individual pixel
  - ↳ GNNs (graph-lvl): classification (is this molecule poisonous/not? At what temp will it melt?)
  - ↳ GNNs (node/edge lvl): graph of customers & products - deciding pricing of products/ recommender systems)
- building decision trees w/ K-ary split; decide K for each node by calculating info gain for diff values of K & optimizing over thresholds & K. Algo will prefer high values for K (↑ K ⇒ ↑ complex decision boundaries ⇒ detailed/nuanced model)
- precision = TP/(TP+FP)
- recall = TP/(TP+FN) = TPR
- true neg. rate = TN/(TN+FP) = 1−FPR
  - ↳ "specificity"
- deep DT ⇒ model overfit ⇒ ↑ var
- shallow DT ⇒ model underfit ⇒ ↑ bias



- FP = (1 − specificity)
- TP = sensitivity
- AUC (area under curve) → larger area = better model
- FP: pred +1, true −1
- FN: P −1, T +1
- TP: P +1, T +1
- TN: P −1, T −1
- ROC curve generated by varying discriminative threshold of our classifier

GNNs

- compute vv msgs across edges ⇒ harder to train, runtime, req. ↑ memory
- might have more param dep. on complexity of parameterization of ψ
- grid-like data ⇒ conv better bc of regular neighborhood structure
- can use msg passing in middle ground by allowing complex interactions between neighborhoods but only computing scalar-valued attention coefficients between pairs of nodes

Graph Neural Networks

- nn that can operate on arbitrary relational structures. Able to learn repr. of nodes that depend on structure of the graph & message passing used to learn these representations
- $G = (V, \mathcal{E})$; node set V, edge set $\mathcal{E}$, set of node features $X \in \mathbb{R}^{|V| \times d}$
  - ↳ want to generate learned node embeddings (z_u) $z_u, \forall u \in V$
  - ↳ during each message passing operation a hidden embedding $h_u^{(k)}$ is generated, representing the updated embedding of node $u \in V$ in the k^{th} iteration, based on the info aggregated from u's graph neighborhood $\mathcal{N}(u)$ (which can incl. u)
- Message Passing Update:

$h_u^{(k)} = \phi^{(k)}\left[h_u^{(k-1)}, \bigoplus(\{\psi^{(k)}(h_v^{(k-1)}, h_u^{(k-1)}), \forall v \in \mathcal{N}(u)\}\right)$

update fcn; ⊕: aggregation operator (sum, mean, max); ψ: message fcn (arbitrarily diff'able (parameterized by NNs))

$= \phi^{(k)}\left[h_u^{(k-1)}, \bigoplus(\{m_{vu}^{(k)}, \forall v \in \mathcal{N}(u)\})\right)$

$m_{vu}^{(k)} = \psi^{(k)}(h_v^{(k-1)}, h_u^{(k-1)})$: message from sender node v to receiver u

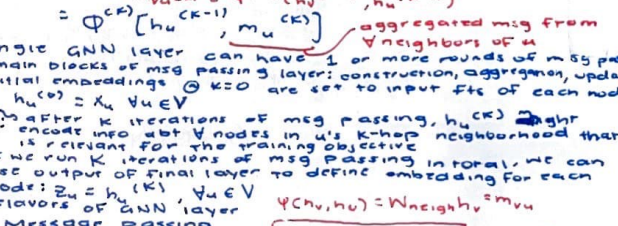
$= \phi^{(k)}\left[h_u^{(k-1)}, m_u^{(k)}\right]$ $m_u^{(k)}$: aggregated msg from ∀ neighbors of u

- single GNN layer can have 1 or more rounds of msg passing
- 3 main blocks of msg passing layer: construction, aggregation, update
- initial embeddings @ k=0 are set to input fts of each node: $h_u^{(0)} = x_u \; \forall u \in V$
- after K iterations of msg passing, $h_u^{(K)}$ might encode info about ∀ nodes in u's K-hop neighborhood that is relevant for the training objective
- if we run K iterations of msg passing in total, we can use output of final layer to define embedding for each node: $z_u = h_u^{(K)}, \forall u \in V$
- 3 Flavors of GNN layer

① Message passing

ex: $h_u^{(k)} = \sigma\left(W_{self}^{(k)} h_u^{(k-1)} + W_{neig}^{(k)} \sum_{v \in \mathcal{N}(u)} h_v^{(k-1)} + b^{(k)}\right)$

$\phi(h_u, h_v) = \sigma(W_{self} h_u + m_u + b)$ $\quad m_u = \oplus(\{m_v\}) = \sum_{v \in \mathcal{N}(u)} m_v = \sum_{v \in \mathcal{N}(u)} W_{neigh} h_v$



Note: if $W_i^{(k)} \in \mathbb{R}^{d \times m}$ the # of learnable params in the MP update fcn above is $2 \cdot d \cdot m$

② Convolutional

$h_u^{(k)} = \phi(h_u^{(k-1)}, \oplus(\{\omega^{(k)} h_v^{(k-1)}, \forall v \in \mathcal{N}(u)\}))$

- eg: input $I \in \mathbb{R}^{L \times H \times d^{(k-1)}}$ (eg image (LxH) w/ $d^{(k-1)}$ input channels)
  $W^{(k)} \in \mathbb{R}^{L \times H \times d^{(k-1)} \times d^{(k)}}$ (eg filter (LxH) w/ $d^{(k-1)}$ in & $d^{(k)}$ out channels)
- for any point (i,j) let input embedding be $h_{ij}^{(k-1)} = [h_{ij1}, h_{ij2}, \dots, h_{ijd^{(k-1)}}] \in \mathbb{R}^{d^{(k-1)}}$ (this is value of i^{th}, j^{th} pixel, l^{th} of input channels)
- suppose $L = H = 3$: $h_u = \sigma\left(\sum_{i \in [-1,0,1]} \sum_{j \in [-1,0,1]} W_{ij} h_{v,l} + b_j\right)$

* msg only dep on sender node v but not u & weights don't dep on either u or v

③ Attentional

$h_u^{(k)} = \phi(h_u^{(k-1)}, \oplus(\{\alpha(h_u^{(k-1)}, h_v^{(k-1)}), \forall v \in \mathcal{N}(u)\}))$

- $\alpha(h_u, h_v)$ are attn. weights
- suppose attn weights are computed by single-head scaled dot-prod self-attn. ⇒ $\mathcal{N}(u) = V$ bc self-attn is msg passing on FC graph. can parameterize attn weights as

$\alpha(h_u, h_v) = \frac{\exp\left\{\frac{1}{\sqrt{d}} (W_k h_v)^T (W_q h_u)\right\}}{\sum_{v' \in \text{Nodes}} \exp\left\{\frac{1}{\sqrt{d}} (W_k h_{v'})^T (W_q h_u)\right\}}$

- to emulate self-attn more, could also multiply h_v by W_v

* weights are fx-dep. & messages dep on sender & receiver

EM

- iterative method to find local ML/MAP estimates of params in statistical models
- coordinate ascent algo
- 2 steps: initialize θ^0

① Expectation

$$q^{t+1} = \arg\max_q F(q, \theta^t) = \arg\max_q H(q(z_i|x_i)) + E_q[(L_c(x_i, z_i; \theta)]$$

$$= \arg\max_q \sum_{k=1}^{K} q(z_i=k|x_i) \log[q(z_i=k|x_i)] + \sum_{k=1}^{K} q(z_i=k|x_i) \log p(x_i, z_i=k; \theta)$$

$$q^{t+1}(z_i=k|x_i) := p(z_i=k|x_i; \theta^t)$$ → used to comput $E_{q^{t+1}}[L_c(x, z; \theta^t)]$

② Maximization (param estimation)

$$\theta^{t+1} = \arg\max_\theta F(q^{t+1}, \theta) = \arg\max_\theta E_{q^{t+1}}[L_c(x, z; \theta)]$$

EM for MoG

- MoG model: $x|z \sim \mathcal{N}(\mu_z, \Sigma_z)$, $p(z=k) = \alpha_k$

$\theta := \{\mu_k, \Sigma_k, \alpha_k\}$ $x_1, \dots, x_n \in \mathbb{R}^d$ observed data

- $q^t_{ki} := q^t(z_i = k | x_i)$

① E step: $$q^{t+1}(z_i=k|x_i) = \frac{\alpha_k^t p(x_i|z_i=k; \theta^t)}{\sum_{j=1}^{K} \alpha_j^t p(x_i|z_i=j; \theta^t)}$$

② M step:

$$\mu_k^{t+1} = \frac{\sum_{i=1}^{N} q_{ki}^{t+1} x_i}{\sum_{i=1}^{N} q_{ki}^{t+1}} \quad ; \quad \Sigma_k^{t+1} = \frac{\sum_{i=1}^{N} q_{ki}^{t+1} (x_i - \mu_k^{t+1})(x_i - \mu_k^{t+1})^T}{\sum_{i=1}^{N} q_{ki}^{t+1}}$$

$$\alpha_k^{t+1} = \frac{1}{N} \sum_{i=1}^{N} q_{ki}^{t+1}$$

Jensen's inequality: $f(E[x]) \leq E[f(x)]$

Distance Fcns

satisfy the following:

① $d(x,y) = 0$ iff $x = y$

② $d(x,y) = d(y,x)$ $\forall x, y$

③ $d(x,z) \leq d(x,y) + d(y,z)$ (triangle ineq.)

- dissimilarity measure $d(x,y)$ satisfies the above except possibly ③
- similarity measur $s(x,y) = -d(x,y)$

computational complexity $z \in \mathbb{R}^d$, $D = \{x_1, \dots, x_n\} \in \mathbb{R}^d$

naive exhaustive search (computes dist btwn z & $\forall$ pts in D & returns k nearest points x_i using quickselect)

- $O(d)$: time computing distances btwn z & x_i
- $O(dn)$: time " " " " & $\forall x_i$
- $O(n)$: time spent finding k shortest distances
- overall: $O(dn + n) = O(dn)$

divide & conquer: place Cartesian grid over Ft. space. in this case, assume $k=1$

- in d dimensions need to search 3^d cells
- overall $O(3^d + d3^d) = O(d3^d)$

PCA & L-S

$X \in \mathbb{R}^{n \times d}$, $y \in \mathbb{R}^n$ $\quad X = U \Sigma V^T = \sum_{i=1}^{d} \sigma_i u_i v_i^T$

$U \in \mathbb{R}^{n \times d}$, $\Sigma \in \mathbb{R}^{d \times d}$, $V \in \mathbb{R}^{d \times d}$, $\sigma_1 \geq \sigma_2 \dots \geq \sigma_d \geq 0$

- $\hat{w}_{ridge} = \sum_{i=1}^{d} \frac{\sigma_i}{\sigma_i^2 + \lambda} v_i u_i^T y$ if $\lambda = 0$, $\hat{w}_{rr} = \hat{w}_{ols}$, if $\lambda > 0$ the ↑ the value of σ_i the less penalized it will be in rr
- $\hat{w}_{ols} = \sum_{i=1}^{d} \frac{1}{\sigma_i} v_i u_i^T y$
- $\hat{w}_{PCA\text{-}ols} = \sum_{i=1}^{K} \frac{1}{\sigma_i} v_i u_i^T y = V_K \Sigma_K^{-1} U_K^T y$ (multiply by V_K to bring w/ orig to d dimensions)
  - large σ_i's kept intact, while small ones (after k) are completely removed. equivalent to $\lambda = 0$ for 1st K components & $\lambda = \infty$ for rest
  - ⇒ rr is "smooth" version of PCA regr.

Misc

- compared to ols, rr has ↑ bias & ↓ variance

Backpropogation

$f(x_1, \cdots, x_n)$; $g_i(w) = x_i$

$$\frac{df}{dw} = \sum_{i=1}^{n} \frac{\partial f}{\partial x_i} \cdot \frac{\partial x_i}{\partial w} = \frac{\partial f}{\partial x} \frac{\partial x}{\partial w}$$

Convolution

- Image $I \to 3$ color channels I_r, I_b, I_g
  - each channel size $W \times H$
- Filter $G \to 3$ color channels G_r, G_b, G_g
  - each mask size $\omega \times h$
- convolution operation at point (x, y):

$$(I * G)[x,y] = \sum_{a=0}^{\omega-1} \sum_{b=0}^{h-1} \sum_{c \in \{r,g,b\}} I_c[x+a, y+b] \cdot G_c[a,b]$$

- size of output of convolution:
  - no stride: $(1+W-\omega) \times (1+H-h)$
  - stride: $\lfloor 1 + (W-\omega)/s \rfloor \times \lfloor 1 + (H-h)/s \rfloor$

- $$\frac{\partial L}{\partial G_c[x,y]} = \sum_{i,j} \frac{\partial L}{\partial R[i,j]} \cdot \frac{\partial R[i,j]}{\partial G_c[x,y]}$$

  $$= \sum_{i,j} \frac{\partial L}{\partial R[i,j]} \cdot \frac{\partial}{\partial G_c[x,y]} \sum_{c} \sum_{a} \sum_{b} I_c[i+a, j+b] G_c[a,b]$$

  $$= \sum_{i,j} \frac{\partial L}{\partial R[i,j]} \cdot I_c[i+x, j+y]$$

- Max pooling; output = R or (i,j):

$$R[i,j] = \max_{a,b} (I * G)[s*i+a, s*j+b]$$

Vanishing Gradient

- when output s is close to 0 or 1
  $s' \approx 0 \longrightarrow$ GD change s slowly & the unit is stuck
- mitigation: