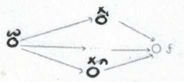
Backpropogation

$F(x_1, \dots, x_n)$; $g_i(\omega) = x_i$



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

Convolution

- Image I → 3 color channels I_r, I_b, I_g
  - ↳ each channel size W×H
- Filter G → 3 color channels G_r, G_b, G_g
  - ↳ each mask size w×h
- convolution operation at point (x,y):

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

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

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

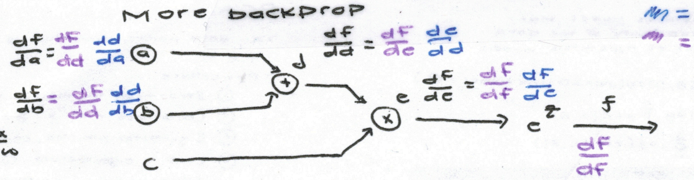
- Max pooling; output = R at [i,j]:
  $R[i,j] = \max_{a,b} (I * G)[s*i+a, s*j+b]$

- # params in conv lays = (# filters) x (filter width) x (filter height) x (# filters in prev. layer + 1)
- CNNs have more architecture params than FCNs

Vanishing Gradient

→ soln:
1. normalize data in mini-batch
   $\hat{z}^{(l)} = (z^{(l)} - \mathbb{E}[z^{(l)}])/Var(z^{(l)})$
2. add scale & shift params
   $h^{(l)} = \sigma(\gamma\,\hat{z}^{(l)} + \beta)$

More backprop



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

= computed during forward pass
= computed during backward pass

- Backprop takes O(#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×D. N filters K×K. No padding. output from 1 filter has size (D−K+1)(D−K+1) → for all N, mult by N.

- # learnable params in NN:
  - each image has 28 × 28 = 784 fts
  - 2 hidden layers w/100 hidden units each
  - all layers (incl. output) have a bias term

  ⇒ $\underbrace{784\cdot100+100}_{\text{layer 1}} + \underbrace{100^2+100}_{\text{layer 2}} + \underbrace{10\ldots}_{\text{output}}$

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

K-means clustering:

1. Partition n points into K disjoint clusters
   cluster i's mean $= \mu_i = \frac{1}{n_i}\sum_{y_j=i} x_j$

→ 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

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

Entropy & Information

- entropy := "expected surprise"
  $H(Y) := \mathbb{E}[-\log P(y)]$
  $= -\sum_k P(X=k)\log(P(Y=k$
- conditional entropy
  $X_{j,v} := \begin{cases}1 & \text{if } x_j \le 0\\ 0 & \text{o/w}\end{cases}$
  $H(Y|X_{j,v}=1)$ = entropy of distribution of points s.t. $x_j \le v$
- information gain / mutual information:
  $I(X_{j,v}; Y) := H(Y) - H(Y|X$
  $= H(Y) - P(X=1)H(Y|X=1) - P(X=0)H(Y|X=0)$
- high entropy:
  - ↳ Y from a uniform dist.
  - ↳ sampled values less predictable
  - ↳ flat histogram

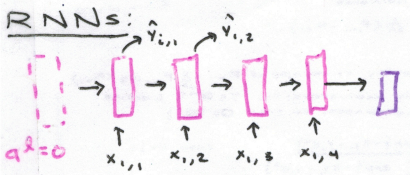
Classification:

- principle: minimize P[error]
  $P[error] = \int p(error|x)\,p(x)\,dx$

ResNets:

- skip connections!
  $x^{(\ell)} = \sigma(z^{(\ell-1)}) + x^{(\ell-1)}$
  (instead of $x^{(\ell)} = \sigma(z^{(\ell-1)})$)

RNNs:



- each layer

$$\hat{a}^{\ell-1} = \begin{bmatrix} a^{\ell-1} \\ x_{i,\ell} \end{bmatrix}$$
$$z^\ell = W^\ell \hat{a}^{\ell-1} + b^\ell$$
$$a^\ell = \sigma(z^\ell)$$
} variable # inputs

$$\hat{y}^\ell = f(a^\ell)$$
} decoder → variable # outputs

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

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 $(\tilde{x}_1, \dots, \tilde{x}_K)$ 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 $(\tilde{x}_1, \dots, \tilde{x}_K)$
    2. $\hat{y}$ = majority vote of $(f(\tilde{x}_1), \dots f(\tilde{x}_K))$
    - output: $\hat{y}$
- Nearest neighbors (in the limit)
  - ↳ $\varepsilon^*(x)$:= avg. error of Bayes classifier
  - ↳ $\varepsilon_{nn}(x)$:= error of NN
  - * Thm 1: $\lim_{n\to\infty} \varepsilon_{NN}(x) \le 2\varepsilon^*(x)$
  - * Thm 2: $\lim_{n\to\infty, K\to\infty} \varepsilon_{KNN}(x) = \varepsilon^*(x)$ 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)

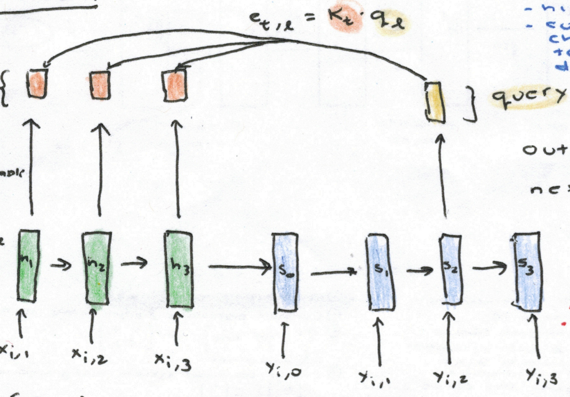
- Bayes rule:
  $P[Y|X] = \frac{P[X|Y]\,P[Y]}{P[X]}$
  $P[X] = \sum_i P[X|Y=i]$

- 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 boundarie
- 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 no
     - else:
       - split by next best attribute
         $\arg\max_i I(X_i) = \arg\max_i H(Y) - H(Y|X_i)$
       - 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

Attention



key: $k_t = K(h_t)$ (learned fcn) ⇒ not differentiable ↓ ...et h_t for ...rgmax$_t$ $e_{t,\ell}$ for step ℓ

query: $q_\ell = q(s_\ell)$

...et: $\alpha_{\cdot,\ell}$ = softmax$(e_{\cdot,\ell})$

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

...send $a_\ell = \sum_t \alpha_{t,\ell} h_t$

output: $\hat{y}_\ell = f(s_\ell, a_\ell)$

next decoder step:
$$\vec{s}_\ell = \begin{bmatrix} s_{\ell-1} \\ \alpha_{\ell-1} \\ y_\ell \end{bmatrix}$$

- self-attn runtime $O(d^2n + n^2d)$
- self-attn α value computation:
  $\alpha(Q, K, V) = \text{softmax}(QK^T)V$
- 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

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 $(y = \frac{1}{m}\sum_m^M G_m(x))$
- random forests – same as bagging except at each split, choose only a random subset p′ (usually $p' = \sqrt{p}$) features to split
- boosting: reweight training points to emphasize ones incorr. classif
  - algo:
    1. train next model conditioned on all prev models & their weights
    2. reweight models to minimize loss: $y = \sum_{m=1}^{M}\alpha_m G_m(x)$
    3. repeat

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

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

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

$f(X) \approx f(A) + \langle \nabla_x f(A), X - A \rangle$
$= f(A) + Tr(\nabla_x f(A)^T (X-A))$

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

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

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

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

Miscellaneous

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

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

Lagrangian

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

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

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

Positive Semidefiniteness

$A \in \mathbb{R}^{n\times n}$ symmetric
1. $\forall x \in \mathbb{R}^n \quad x^T A x \geq 0$
2. $\forall \lambda_i \geq 0$ s.t. $Av_i = \lambda_i v_i$
3. $\exists U \in \mathbb{R}^{n\times n}$ s.t. $A = UU^T$

} Definition of $A \succeq 0$

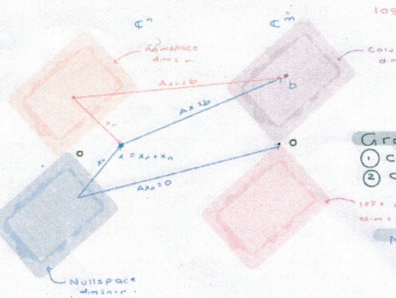
Properties of PSD matrices

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

- must be symmetric
- diagonals must be non-0
- PSD ($v^T \Sigma v \geq 0 \; \forall v$)
- determinant must be ≥ 0

Fundamental Theorem of Linear Algebra

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



P(B)

- rowspace (X) = columnspace (X^T)
- rowspace (X) ⊥ nullspace (X)
- $N(X) = N(X^T X)$
- columnspace ($X^T X$) = rowspace ($X^T X$) = rowspace (X)

Probability

Bernoulli: biased coin flip [Bern(p)]

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

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

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

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

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

Unif([a,b])

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

Multivariate Gaussians

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

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

- if $X \in \mathbb{R}^d$ is distributed as an MVG, then $\forall i,j \in \{1, \ldots, d\}$ $cov(X_i, X_j) = 0$ iff X_i, X_j independent
- Σ for MVG is symmetric & PSD ⇒ $\lambda \geq 0 \; \forall i$
- a random vector follows the MVG distribution iff each coordinate marginally & conditionally Gaussian
$z_i \sim \mathcal{N}(\mu_i, \sigma)$ & $z_i | z \sim \mathcal{N}(\mu_i, \sigma_i^2)$

MLE

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

$\hat{\theta}_{mle} = \arg\max_\theta \mathcal{L}(\theta; D) = p(data = D \mid \text{true model} = \theta)$
$= \arg\max_\theta \log \prod_{i=1}^n p(y_i \mid x_i, \theta)$
$= \arg\max_\theta \sum_{i=1}^n \log(p(y_i \mid x_i, \theta))$

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

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

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

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

→ which is just OLS!

MLE properties:

- consistent (more data ⇒ convergence of true θ value for D)
- statistically efficient (least variance parameter estimates)
- value of $p(D \mid \theta_{mle})$ is invariant to reparameterization
 → eg $\mathcal{N}(x \mid \mu, \sigma)$ vs $\mathcal{N}(x \mid 2\mu, \sigma)$
- 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

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

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

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

$w_{mle} = (A^T A)^{-1} A^T y$

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)

MAP: Maximum a Posteriori

- find a point estimate that maximizes the posterior probability

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

- 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 $p(w \mid D) \propto p(D \mid w) p(w)$

$w_{map} = \arg\max_w \log p(D \mid w) p(w) = \arg\max_w \log p(D \mid w) + \log \mathcal{N}(w, 0, \lambda I)$
$= \arg\max_w \sum_{i=1}^N \log \mathcal{N}(y_i \mid w^T x_i, \sigma^2) + \sum_{i=1}^d \log\left(\frac{1}{\sqrt{2\pi}\lambda} \exp\left(-\frac{w_i - 0}{2\lambda}\right)^2\right)$
$= \arg\min_w \frac{1}{2\sigma^2}(y - Aw)^T(y - Aw) + \sum_{i}^d \frac{w_i^2}{2\lambda}$
$= \arg\min_w (y - Aw)^T(y - Aw) + \frac{\sigma^2}{\lambda}\|w\|_2^2$

$w_{rr} = (A^T A + \lambda' I)^{-1} A^T y$
$\lambda' > 0 \Rightarrow (A^T A + \lambda I)$ invertible

Softmax Regression classification

- logistic regression — 2 classes

$\begin{bmatrix} p(y=1 \mid x) \\ p(y=0 \mid x) \end{bmatrix} = \begin{bmatrix} \mu \\ 1-\mu \end{bmatrix} = \begin{bmatrix} \frac{1}{1+\exp(-\beta^T x)} \\ \frac{\exp(-\beta^T x)}{1+\exp(-\beta^T x)} \end{bmatrix}$

- softmax/multinomial logistic regression ↳ use MLE to get param β
- needs iterative optimization (eg. ∇ descent)
- can put at top of NN to get softmax loss

$\begin{bmatrix} P(y=1 \mid x) \\ \vdots \\ P(y=n \mid x) \end{bmatrix} = \frac{1}{\sum_{i=1}^n e^{\beta_i x}} \begin{bmatrix} e^{\beta_1 x} \\ \vdots \\ e^{\beta_n x} \end{bmatrix}$

- for class i, softmax is: $P[y=i \mid x] = \frac{e^{logit_i}}{\sum_j e^{logit_j}}$

$logit_i = \log\left(\frac{P[y=i \mid x]}{P[y \neq i \mid x]}\right)$

Bias-Variance Tradeoff

- Bias of a model:
$E[f(\hat{x}) - f(x)]$
↳ measure of avg diff btwn model output & ground truth over ∀ possible training sets

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

Model error (3 components):
$\mathcal{E}(x, h) = (E[h(x;D) - f(x)])^2 + Var(h(x;D)) + Var(z)$
= bias² + variance + irreducible error

Natural Log Rules

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

Regression Fcns

1. Linear: $h(x; w, a) = wx + a$
2. Polynomial: equiv. to linear w/added polyn. feat.
3. Logistic: $h(x; w, a) = s(wx + a)$, $s(\gamma) = \frac{1}{1+e^{-\gamma}}$

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

- (A) squared error: $L(z, y) = (z - y)^2$
- (B) Absolute error: $L(z, y) = |z - y|$
- (C) Cross-entropy: $L(z, y) = -y\ln z - (1-y)\ln(1-z)$

Cost Fcns to minimize

- (a) Mean loss: $J(h) = \frac{1}{n}\sum_i L(h(x_i), y_i)$
- (b) Max loss: $J(h) = \max L(h(x_i), y_i)$
- (c) Weighted sum: $J(h) = \sum_i w_i L(h(x_i), y_i)$
- (d) L2 penalized: $J(h) =$ (a) or (b) or (c) $+ \lambda\|w\|_2$
- (e) L1 penalized: $J(h) =$ (a) or (b) or (c) $+ \lambda\|w\|_1$

Famous Regression Methods

- Least-sq. linear regr: (1) + (A) + (a)
- Weighted Least-sq. linear regr: (1) + (A) + (c)
- ridge regression: (1) + (A) + (d)

} quadr. cost → minimize w/ calculus

- LASSO: (1) + (A) + (E) } Quadratic Program
- Logistic regression: (3) + (C) + (a) } convex cost → minimize w/gradient descent

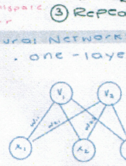
Gradient Descent

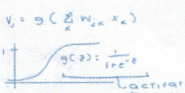
1. choose $x_0 \in \mathbb{R}^d$ & $k = 0$
2. choose $t_k > 0$ & $x_{k+1} = x_k - t_k \nabla f(x_k)$
3. Repeat

Neural Networks

- one-layer

$V_j = g(\sum_k w_{jk} x_k)$





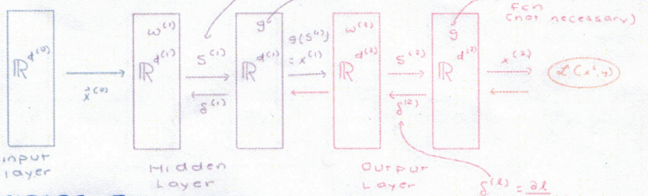
- cross entropy loss:

$L = -\sum (y_i \ln V_i + (1 - y_i)\ln(1 - V_i))$
input data

- $Var(A+B) = Var(A) + Var(B)$
- $Var(kA) = k^2 Var(A)$

- one hidden layer

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

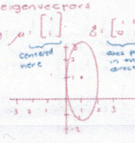


Key MVG properties

1. $x_1, \ldots, x_n \overset{indep}{\sim} \mathcal{N}(0,1)$, $X \sim (0, I)$, $Y = AX + \mu \sim \mathcal{N}(\mu, AA^T)$
2. if $\Sigma > 0$ & $Y \sim \mathcal{N}(\mu, \Sigma) \Rightarrow A^{-1}(Y - \mu) \sim \mathcal{N}(0, 1)$

- isocontours of MVGs
 - ellipsoids with radius $\sqrt{\lambda}$, in direction of their corresponding eigenvectors

eg $\mu = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ (centered here) $\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$ (axes pointing in these directions)



scaling an MVG's Σ matrix by X scales the isocontour radius by $\sqrt{X}$

- CLT: Σ(large # indep RVs) → Gaussian distribution

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

1st PC (w) of n×d data matrix X:
- $\min_w \sum_{i=1}^n \|X_i - \frac{X_i^T w}{\|w\|^2} w\|^2$

or $\max_{\|w\|=1} w^T X^T X w$

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

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

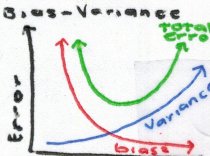
PCA limitations

- 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



- ↑ residuals ⇒ ↑ bias
- ↑ complexity ⇒ ↑ v ar

Stochastic NE

1. Probability x_i chooses x_j as its neighbor:

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

$P_{j \leftarrow j} = 0$

2. Symmetrize & Normalize

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

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

3. Set σ_i^2 adaptively s.t. entropy of $P_{\cdot \leftarrow i} = \sum_{j \neq i} P_{j \leftarrow i} \log(P_{j \leftarrow i})$
4. Posit low-dim representations $Y \in \mathbb{R}^{n\times k}$ & define stochastic neighborhoods for them $Q = \{Q_{ij}\}$

$Q_{ji} = \frac{\exp\{-\|y_i - y_j\|^2\}}{\sum_{l \neq k} \exp\{-\|y_l - y_k\|^2\}}$ (note $\sigma = \frac{1}{2}$)

Goal: Find Y s.t. SN structure preserved → soln: 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 $Q_{j \leftarrow i} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{i \neq k}(1 + \|y_i - y_k\|^2)^{-1}}$
- everything else stays the same; this prevents crowding in low-dim space

- SNE & tSNE can be viewed as doing MLE to obtain reduced dimensionality repr of data bc minimizing KL divergence objective is equivalent to maximizing MLE objective

Bias-Variance Takeaways

- underfitting ≡ high bias; overfitting ≡ high variance
- training error reflects bias, not variance. test error reflects both
 ↳ if training err << test err ⇒ overfitting
- var → 0 as n → ∞
- if f is in set of hypothesis fcns, bias will ↓ with ↑ data. if not, then underfitting problem & ↑ data won't help
- **bias: $E[\hat{X}] - \mu$**
- **Variance: $Var(\hat{X})$**

- 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 ($cov(X,Y) = 0$) but not indep
- $X \sim \mathcal{N}(\mu, \Sigma) \Rightarrow AX + b \sim \mathcal{N}(A\mu + b, A\Sigma A^T)$
- $MGF(X) = E[\exp\{\lambda^T X\}] = \exp\{\lambda^T \mu + \frac{1}{2}\lambda^T \Sigma \lambda\}$
- $Z = \sqrt{\Sigma^{-1}}(X - \mu) \sim \mathcal{N}(0, I_n)$

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
$\begin{bmatrix} x_1' \\ x_2' \end{bmatrix} = Q \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$

→ Axis points along eigenvectors (major & minor axis lengths are $1/\sqrt{\lambda_1}$ & $1/\sqrt{\lambda_2}$)
⇒ Finding coordinate system axis-aligned ≡ diagonalizing A

Bias-Variance : $\hat{X}$: estimator constr. from sample ... X': new sample

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

$\mathbb{E}[(\hat{X}-\mu)^2] = (\mathbb{E}[\hat{X}]-\mu)^2 + var(\hat{X})$
$= var(\hat{X}) + bias^2$

- 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(z) but noise in training set also affects bias & variance
- adding bad ft rarely increases bias (j set coeff = 0)

eg: semantic segment. node classification
eg: molecule/img classification

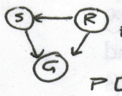
Markov Models

$Q = q_1 q_2 \dots q_n$: set of N states
$A = a_{11} a_{12} \dots a_{nn}$: Transition prob matrix A, each a_{ij} repr. $P(q_i \to q_j)$ 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
     ↳ Viterbi
  3) Learning: given observation seq. O & set of states, learn HMM params A & B

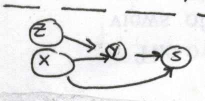
Probabalistic Graphical Models

- 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 it's parents

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

conditional independence: $X \perp Y \mid Z \Rightarrow P[X \mid Y, Z] = P[X \mid Z]$

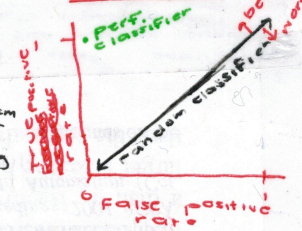
Markov Decision Processes

- $S_t \in S$: 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:
  $\pi(a \mid s) = \mathbb{P}[A_t = a \mid S_t = s]$
- state-value fcn v_π of state s under policy π:
  $v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s]$ (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:
  $q_\pi(s,a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a]$
- $\mathbb{E}_\pi[R_{t+1} \mid S_t = s] = \sum_r r \mathbb{P}[r \mid s] = \sum_r r \sum_{s',a} p(s', r \mid s, a) \pi(a \mid s)$
- $v_\pi(s) = \mathbb{E}_\pi[q_\pi(s,a)] = \sum_a q_\pi(s,a) \pi(a \mid s)$
- $q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] = \sum_{s',r} [r + \gamma v_\pi(s')] p[s', r \mid s, a]$
- 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 $\forall s$, initialize $V_0(s) = 0$
  ② $\forall s \in S$, 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
  $R(f) = \mathbb{E}_{X,Y}[L(f(X), Y)] = \mathbb{E}_X[\mathbb{E}_{Y|X}[L(f(X), Y) \mid X]] = \mathbb{E}_X[\sum_{j=1}^{K} L(f(X), j) P(Y=j \mid X)]$
  $= \int_X \sum_{j=1}^{K} L(f(x), Y) P(Y=j \mid X) P(x)\, dx = \int_X \sum_{j=1}^{K} L(f(x), j) \mathbb{P}(X \mid Y=j) P(Y=j)\, dx$
- Bayes classifier: attains minimum risk a classifier can achieve
- Bayes risk/error rate: lowest risk any classifier can attain
  $f^*(x) = \arg\min_i \mathbb{E}_{Y|X}[L(i, Y) \mid X] = \arg\max_i \mathbb{P}[Y = i \mid X]$
- 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
  $\lambda_{11} \mathbb{P}[\omega_1 \mid x] + \lambda_{12} \mathbb{P}[\omega_2 \mid x] = \lambda_{22} \mathbb{P}[\omega_2 \mid x] + \lambda_{21} \mathbb{P}[\omega_1 \mid x]$
  $\frac{\mathbb{P}[\omega_1 \mid x]}{\mathbb{P}[\omega_2 \mid x]} = \frac{\lambda_{22} - \lambda_{12}}{\lambda_{11} - \lambda_{21}}$

- 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
  " > " ⇒ over "
- 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 bc general msg passing allows arbitrarily learnable msg fcns but conv & attn pose extra constraints
- translational equiv – useful for pixel & node-lvl tasks
- rotational inv – useful for graph & img-lvl tasks
- tasks:
  ↳ CNNs (image-lvl): classification/regression, one output/target for entire img. eg: dog, cat
  ↳ CNNs (pixel-lvl): semantic seg for classification of each individual pixel
  ↳ GNNs (graph-lvl): classification (eg 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



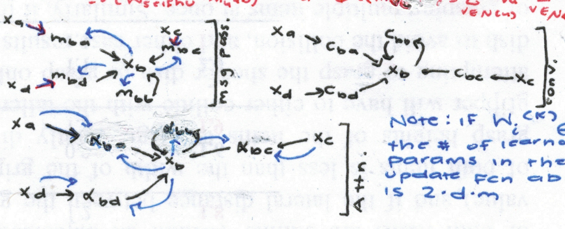
- FP = (1 − specificity)
- TP = sensitivity
- AUC (area under curve) → larger area = better model
- FP : pred +1, True −1
- FN : P −1, T +1
- TP : P +1, T +1
- TN : P −1, T −1
- ROC curve generated by varying discriminative threshold of our classifier

GNNs

- compute VV msgs across edges ⇒ harder to train, ↑ runtime, ↑ memory
- might have more params dep. on complexity of parameterization of ψ
- grid-like data ⇒ conv better bc of regular neighborhood structure
- attn msg passing ⇒ middle ground by allowing complex interactions btwn neighborhoods but only computing scalar-valued quantities across edges

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 fts $X \in \mathbb{R}^{d \times |V|}$
  ↳ 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 iteration, based on the info aggregated from u's graph neighborhood $\mathcal{N}(u)$ (which can incl. u)
- Message Passing Update:
  $h_u^{(k)} = \phi^{(k)}\left[h_u^{(k-1)}, \bigoplus\left(\{\psi^{(k)}(h_v^{(k-1)}, h_u^{(k-1)}), \forall v \in \mathcal{N}(u)\}\right)\right]$
  ϕ: update fcn, ↑ arbitrarily diff'able; $\bigoplus$: aggregation operator (usually non-parameteric); ψ: message fcn (parameterized by NNs)
  $= \phi^{(k)}\left[h_u^{(k-1)}, \bigoplus(\{m_{vu}^{(k)}, \forall v \in \mathcal{N}(u)\})\right]$
  message from sender node v to receiver u: $m_{vu}^{(k)} = \psi^{(k)}(h_v^{(k-1)}, h_u^{(k-1)})$
  $= \phi^{(k)}[h_u^{(k-1)}, m_u^{(k)}]$ — 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 abt ∀ 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: $h_u^{(k)} = \sigma\left(W_{self}^{(k)} h_u^{(k-1)} + W_{neig}^{(k)} \sum_{v \in \mathcal{N}(u)} h_v^{(k-1)} + b^{(k)}\right)$
  $\psi(h_v, h_u) = W_{neigh} h_v = m_{vu}$
  $\phi(h_u, h_v) = \sigma(W_{self} h_u + m_u + b)$
  $m_u = \bigoplus(\{m_{vu}\}) = \sum_{v \in \mathcal{N}(u)} m_{vu} = \sum_{v \in \mathcal{N}(u)} W_{neigh} h_v$



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
  $h_u^{(k)} = \phi\left(h_u^{(k-1)}, \bigoplus(\{W^{(k)} h_v^{(k-1)}, \forall v \in \mathcal{N}(u)\})\right)$
  - eg: input $I \in \mathbb{R}^{L \times H \times d^{(k-1)}}$ (eg image (L×H) w/ $d^{(k-1)}$ input channels)
    $W^{(k)} \in \mathbb{R}^{l \times h \times d^{(k)} \times d^{(k-1)}}$ (eg filter (l×h) w/ d^{k-1} in & d^k out channels)
    for any point (i,j) let input embedding be $h_{ij}^{(k-1)} = [h_{ij1}, h_{ij2}, \dots h_{ij d^{(k-1)}}] \in \mathbb{R}^{d^{(k-1)}}$ (h_{ijc} is value of input channel)
  - suppose l = h = 3: $h_u = \sigma\left(\sum_{i,j \in \{-1,0,1\}} W_{ij} h_{x+i, y+j}\right)$
  * msg only dep on sender node v but not u & weights don't dep on either u or v
  ③ Attentional
  $h_u^{(k)} = \phi\left(h_u^{(k-1)}, \bigoplus(\{\alpha(h_u^{(k-1)}, h_v^{(k-1)}), \forall v \in \mathcal{N}(u)\})\right)$
  - $\alpha(h_u, h_v)$ 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
    $\alpha(h_u, h_v) = \frac{\exp\{\frac{1}{\sqrt{d}}(W_k h_v)^T(W_q h_u)\}}{\sum_{v' \in \mathcal{N}(u)} \exp\{\frac{1}{\sqrt{d}}(W_k h_{v'})^T(W_q h_u)\}}$
  - to emulate self-attn more, could also multiply h_v by ...
  * weights are ft-dep. & messages dep on sender & receiver

Expectation-Maximization

iterative method to find local ML/MAP estimates of params in statistical models
coordinate ascent algo

2 steps:
① initialize θ^0
① Expectation

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

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

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

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

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

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

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

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: $f(\mathbb{E}[x]) \leq \mathbb{E}[f(x)]$

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 measure $s(x,y) = -d(x,y)$

computational complexity
naive exhaustive search (computes dist btwn $z \in \mathbb{R}^d$, $D = \{x_1, \dots, x_n\} \subseteq \mathbb{R}^d$ & pts in D & returns k nearest points (using quickselect))
↳ $O(cd)$: time computing distances btwn z & x_i
↳ $O(cdn)$: time " " " & $\forall x_i$
↳ $O(cn)$: time spent finding k shortest distances
↳ overall: $O(cdn + cn) = O(cdn)$
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

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

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

$\hat{w}_{ridge} = \sum_{i=1}^{d} \frac{\sigma_i}{\sigma_i^2 + \lambda} v_i u_i^T y$; 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 weights 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.

LSC

compared to ols, rr has ↑ bias & ↓ variance