CS 189 Midterm Notes

Multivariate Gaussians: iff each $X_i \in N(\mu_i, \sigma_i^2)$ marginally, and each $x_i | X_j \sim N(\mu_k, \sigma_k^2)$ conditionally.
[or jointly gaussian] iff $p_{\vec{x}}(\vec{x}) = \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} e^{-\frac{1}{2}(\vec{x}-\vec{\mu})^T \Sigma^{-1}(\vec{x}-\vec{\mu})}$

iff $\vec{x} = A\vec{z} + \vec{\mu}$, where A is some real-valued matrix

MLE for Lin Reg: $\vec{w}^* = (X^TX)^{-1}X^T\vec{y}$ is $MLE[\vec{w}|X,\vec{y}]$, where $\vec{y} \sim N(X\vec{w}, \sigma^2)$

MAP for L2 Reg: $\vec{w}^* = (X^TX+\lambda)^{-1}X^T\vec{y} = \arg\min_{\vec{w}} \|\vec{y} - X\vec{w}\|_2^2 + \lambda\|\vec{w}\|_2^2 = \arg\max_{\vec{w}} \sum_{i=1} \log P_{N(\vec{x}_i^T\vec{w}, \sigma_1^2)}(y_i) + \log P_{N(0,\sigma_2^2 I)}(\vec{w})$
[$w_i \overset{iid}{\sim}$ Laplace dist gives L1 Reg] $[\lambda = \frac{\sigma_1^2}{\sigma_2^2}]$ [Assuming $w_i \overset{iid}{\sim} N(0, \sigma_2^2)$] $= \arg\max_{\vec{w}} P(\vec{y}|\vec{w}, X) \cdot P(\vec{w}) = \arg\max_{\vec{w}} P(\vec{w}|X,\vec{y}) = MAP[\vec{w}]$.

Principal Component Analysis: (unsupervised, linear, assumes Gaussian, skewed by scale of features, sensitive to outliers).
$\Sigma = Cov(X) = X^TX = \Theta D \Theta^T$. Then, columns of Θ_k are k principal components, $X\Theta_k$ = projected data. (k-dim). $D = diag(\vec{\lambda})$. Then, $\frac{\lambda_i}{\sum \lambda_i}$ = % variance explained by i^{th} PC. $X\Theta_k\Theta_k^T$ = d-dim data but only with k-dim information.
$Tr(X^TX) = \sum \lambda_i \propto var(X)$. [assuming zero mean]

Stochastic Neighbour Embeddings: $P_{j \leftarrow i} = \mathbb{P}[\vec{x}_j$ is chosen as $\vec{x}_i$'s neighbour$] = \frac{e^{-\|\vec{x}_i - \vec{x}_j\|^2/2\sigma_i^2}}{\sum_k e^{-\|\vec{x}_i - \vec{x}_k\|^2/2\sigma_i^2}}$, $P_{i\leftarrow i} = 0$
Then, adapt σ_i^2: $\sum_j P_{j\leftarrow i} \log P_{j \leftarrow i}$ is constant. (for all i)
Then, symmetrize: $P_{ij} = P_{ji} = \frac{1}{2n}(P_{j\leftarrow i} + P_{i \leftarrow j})$.
Then, for SNE, model Q on $Y [\in \mathbb{R}^{n\times k}$, where $X \in \mathbb{R}^{n\times d}$, $d >> k]$ and find $Y = \arg\min_Y KL(P\|Q) = \sum_{ij} P_{ij} \log \frac{P_{ij}}{Q_{ij}}$.

For t-SNE, use $Q_{j\leftarrow i} = \frac{(\|\vec{y}_i - \vec{y}_j\|^2)^{-2}}{\sum_k (\|\vec{y}_i - \vec{y}_k\|^2)^{-2}}$ (in place of e^{-x^2}, use $\frac{1}{x^2}$). This makes tails heavier.

KL-divergence = Relative Entropy
$= H(P,Q) - H(P)$
$= -\sum P_i \log(Q_i) + \sum P_i \log(P_i)$
$= \sum P_i \log\left(\frac{P_i}{Q_i}\right)$

Cross-Entropy $= H(P,Q) = -\sum P_i \log Q_i$.
KL-divergence measures difference in distributions. (deviation of Q from P).

k-means clustering: (unsupervised, linear, scale-sensitive on axes)
For each point $\vec{x}_i$, let $y_i \in [k]$ be the predicted cluster. Then, k-means minimizes $\sum_{j=1} \|\vec{x}_j - \vec{\mu}_{y_j}\|^2$.
while !converged: 1) calculate cluster memberships $[y_i = \arg\min_k \|\vec{x}_i - \vec{\mu}_k\|^2]$ 2) Reset means based on y_i $[\vec{\mu}_k = (\sum_{i:y_i=k} \vec{x}_i)\frac{1}{|\{i: y_i = k\}|}]$

Risk: $R(\theta) = \mathbb{E}[\ell(\theta(X), Y)]$.

Logistic Regression: $y_i \in \{0,1\}$, $\hat{y}_i = \sigma(\vec{w}^T\vec{x}_i)$, $\sigma(z) = \frac{1}{1+e^{-z}}$. Loss = BCE. Optim - GD or **Newton's Method**
$\vec{w}_{k+1} = \vec{w}_k - \eta_k \nabla_{\vec{w}}\ell(\vec{w})\big|_{\vec{w}_k}$, $= \vec{w}_k - \ell(\vec{w}_k)[\nabla_{\vec{w}}\ell(\vec{w}_k)]^{-1}$

Binary Cross Entropy: $y \ln \hat{y} + (1-y)\ln(1-\hat{y})$.

Softmax Regression: $y_i \in [k]$. $\hat{y}_i = \arg\max_j s(\vec{w}_j^T\vec{x}_i)$. $s(\vec{z}) = \frac{e^{-z_j}}{\sum e^{-z_j}}$. Loss = Cross Entropy. (negative cross entropy).

Neural Networks: Repeated application of linear map $W^{(\ell)}$, bias $\vec{b}^{(\ell)}$, non-linear activation $g^{(\ell)}$.
$\vec{x}^{(\ell+1)} = g(W^{(\ell)}\vec{x}^{(\ell)} + \vec{b}^{(\ell)})$

Back propagation:
$\vec{x}^{(i)} \to \boxed{\vec{z}^{(i)} = \vec{w}^T\vec{x}^{(i)} + b} \xrightarrow{\vec{z}^{(i)}} \boxed{\vec{a}^{(i)} = \frac{1}{1+e^{-\vec{z}^{(i)}}}} \xrightarrow{\vec{a}^{(i)} = \hat{y}^{(i)}} BCE(\hat{y}^{(i)}, \vec{y}^{(i)})$

$\boxed{\text{Perceptron Example (log reg)}}$

$\frac{\partial L}{\partial \vec{a}^{(i)}} = -\frac{\vec{y}^{(i)}}{\vec{a}^{(i)}} + \frac{1-\vec{y}^{(i)}}{1-\vec{a}^{(i)}}$

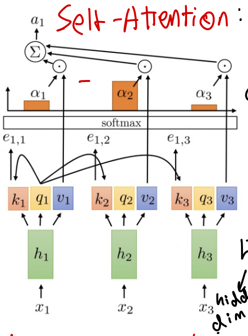
$\frac{\partial \vec{a}^{(i)}}{\partial z^{(i)}} = \vec{a}^{(i)}(1-\vec{a}^{(i)})$, $\frac{\partial L}{\partial z^{(i)}} = \left[\frac{\partial L}{\partial \vec{a}^{(i)}}\right]^T \frac{\partial \vec{a}^{(i)}}{\partial z^{(i)}}$ $\xleftarrow{\frac{\partial L}{\partial z^{(i)}}}$

$\frac{\partial z^{(i)}}{\partial \vec{w}} = x^{(i)}$, $\frac{\partial L}{\partial \vec{x}^{(i)}} = \frac{\partial L}{\partial z^{(i)}}\vec{w}$, $\frac{\partial L}{\partial \vec{w}} = \frac{\partial L}{\partial z^{(i)}} \cdot \frac{\partial z^{(i)}}{\partial \vec{w}}$ $\xrightarrow{\frac{\partial L}{\partial \vec{x}^{(i)}}}$

Residual Connections:
- $x^{(\ell)} = \sigma(z^{(\ell-1)}) + x^{(\ell-1)}$
- smooths out loss landscape
- allows deeper models
- ResNet: CNN with RC.

Convolution: $Out_{N_i, C_j} = bias_j + \sum_{k=0}^{C_{in}-1} W_{C_j,k} * In_{N_i,C_k}$ ($H'\times W'$, $H'\times W'$, $a\times b$, $H\times W$)
$(N, C_{in}, H, W) \longrightarrow (N, C_{out}, H', W')$
$C_{in} \cdot C_{out}$ filters of size a, b.
Using stride (s_a, s_b), padding (P_a, P_b)
$H' = \frac{H + 2P_a - a}{s_a} + 1$, $W' = \frac{W + 2P_b - b}{s_b} + 1$
(default. $P_a = P_b = 0$, $s_a = s_b = 1$, $a = b$).
- Conv layers are translationally equivariant.
- Conv layers use element-wise mult.
- # Params $= \underbrace{C_{in} \cdot C_{out} \cdot a \cdot b}_{\text{weights}} + \underbrace{C_{out} \cdot a \cdot b}_{\text{bias}}$

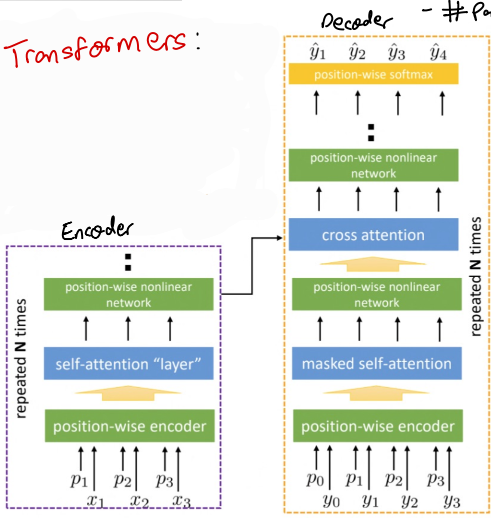
Self-Attention:



$\vec{a}_\ell = \sum_t \alpha_{\ell,t}\vec{v}_t$ attention
$\alpha_{\ell,t} = \frac{e^{e_{\ell,t}}}{\sum_{t'} e^{e_{\ell,t'}}}$ prob. dist.
$e_{\ell,t} = \vec{q}_\ell^T \vec{k}_t$ similarity score
$\vec{v}_t = v(\vec{h}_t)$ value
$\vec{q}_t = q(\vec{h}_t)$ query
$\vec{k}_t = k(\vec{h}_t)$ key
$h_t = g(W\vec{x}_t + \vec{b})$ — hidden dim., shared across time-steps, non-linear function

Positional Encodings:
$\vec{p}_t = \begin{bmatrix} \sin(\frac{t}{10000^{2 \cdot 1/d}}) \\ \cos(\frac{t}{10000^{2\cdot 1/d}}) \\ \sin(\frac{t}{10000^{2\cdot 2/d}}) \\ \vdots \\ \cos(\frac{t}{10000}) \end{bmatrix}$
- bounded magnitude
- non-constant
- usually uses sin / cos.

Transformers:



Batch Normalization:
- Learnable scale and shift
- compute batch statistics (scale, shift) at train time, use to rescale at eval time
- Standardization against vanishing gradient.

Masked Self-attention: Send $k_t \to 0$ $\forall$ t not yet predicted

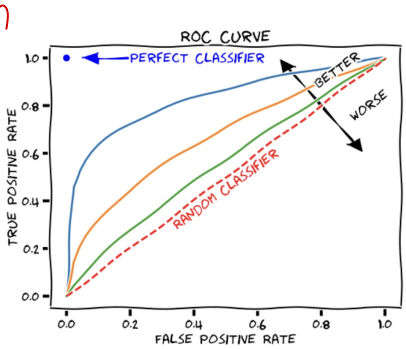
Derivatives Cheats:
- $\sigma'(x) = \sigma(x)(1-\sigma(x))$
- $\frac{\partial}{\partial \vec{x}} \|\vec{x}\|_2^2 = 2\vec{x}$
- $\frac{\partial}{\partial X} \vec{a}^T X \vec{b} = \vec{a}\vec{b}^T$
- $\frac{\partial Y^{-1}}{\partial x} = -Y^{-1}\frac{\partial Y}{\partial x}Y^{-1}$
- $\frac{\partial \vec{b}^T X^T X \vec{c}}{\partial X} = X(\vec{b}\vec{c}^T + \vec{c}\vec{b}^T)$

$|\lambda_{max}(A)| = \|A\|_2$

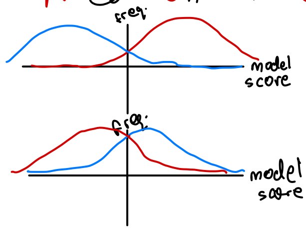
Classification

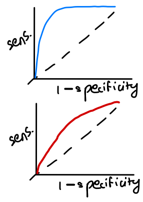
y \ $\hat{y}$	N	P
N	TN	FP
P	FN	TP

- specificity $= \frac{TP}{TP+FP}$ (precision)
- sensitivity $= \frac{TP}{TP+FN}$ (recall)



- Area Under Curve ↑ ⇒ better model





Nearest neighbors

Algorithm The k-nearest neighbors classification algorithm

Input:

D: a set of training samples $\{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$
k: the number of nearest neighbors
$d(\mathbf{x}, \mathbf{y})$: a distance metric
$\mathbf{x}$: a test sample

1: **for each** training sample $(\mathbf{x}_i, y_i) \in D$ **do**
2: Compute $d(\mathbf{x}, \mathbf{x}_i)$, the distance between $\mathbf{x}$ and $\mathbf{x}_i$
3: Let $N \subseteq D$ be the set of training samples with the k smallest distances $d(\mathbf{x}, \mathbf{x}_i)$
4: **return** the majority label of the samples in N

- Pros:
  - no training
  - non-linear functions can be learned
  - 1-NN error <= 2× Bayes error [$\lim n \to \infty$]
  - kNN error = Bayes error [$\lim k, n \to \infty$]
- cons:
  - Slow
  - curse of dimensionality [Neighbors far away and hard to find in higher dim. data]

Decision Trees

- Entropy, $H(Y) = \mathbb{E}_Y[-\log_2 P(Y)] = -\sum_k P(Y=k)\log_2 P(Y=k) \Rightarrow H(Y|X) = \sum_{x\in X} p(x) H(Y|X=x)$
- Surprise$(Y=k) = -\log_2 P(Y=k)$
- Information Gain: $I(X_{j,v}; Y) := H(Y) - H(Y|X_{j,v})$
- Decision tree nodes split on one attribute based on threshold to maximize Information gain.
  ⇒ Axis-alligned decision boundaries. ⇒ Recursive splitting, greedy for Info.
- Easily overfit ⇒ Ensemble to reduce variance
  - Bootstrap Aggregation-ing
    - Train M models with n' (usually $= n$) samples each (w/ replacement)
    - Average M predictions $\left[y = \frac{1}{M}\sum_{m=1}^{M} G_m(x)\right]$
  - Random Forests: same as Bagging but choose $p' < p$ features to split on, each time (more trees, less variance).
  - Boosting:
    - Adaptive Boosting:
      - initialize $w_i = \frac{1}{n}\ \forall i \in [n]$
      - for m in $[M]$:
        - Build model $G_m: \mathbb{R}^d \to \{-1, 1\}$, where training point x_i is weighted w_i.
        - $e_m = \frac{\sum_i \text{misclassified } w_i}{\sum_i w_i}$.
        - Reweight points: $w_i \mathrel{*}= \begin{cases} \sqrt{\frac{1-e_m}{e_m}} & \text{if misclassified} \\ \sqrt{\frac{e_m}{1-e_m}} & \text{otherwise.} \end{cases}$
      - during inference, $y = \frac{1}{M}\sum_{m=1}^{M} \alpha_m G_m(x)$. $\alpha_m = \frac{1}{2}\ln\left(\frac{1-e_m}{e_m}\right)$
        [$e_m \to 0$, $\frac{1-e_m}{e_m} = \infty$, $\alpha_m = \infty$; $e_m \to 1$, $\frac{1-e_m}{e_m} = 0$, $\alpha_m \to -\infty$]
        [higher weight to good classifiers].
    - Gradient Boosting: seek m^{th} model output $\vec{g} = \arg\max_{\vec{g}\in G} |\langle -\nabla_{\vec{y}} L(\vec{y}, F_{m-1}(x)), \vec{g}\rangle|$
      where F_{m-1} is averaged output of m-1 models so far.

Bias-Variance Tradeoff

- with true distribution $\overbrace{\vec{Y}}^{\text{observation}} = \underbrace{f(\vec{x})}_{\text{true relationship}} + \overbrace{\vec{Z}}^{\text{noise}}$, data D, and model h:

$$\underbrace{\mathcal{E}(\vec{x}; h)}_{\text{model error}} = \underbrace{\left(\mathbb{E}[h(\vec{x}; D)] - f(\vec{x})\right)^2}_{\text{bias}^2 \text{ of } h} + \underbrace{Var(h(\vec{x}; D))}_{\text{Variance of } h} + \underbrace{Var(\vec{Z})}_{\text{irreducible error}} = |\mathbb{E}[\hat{y}] - f(\vec{x}) + Var(\hat{y}) + Var(y - f(\vec{x}))$$.

Viterbi for HMMs

```
function VITERBI(O, S, Π, Y, A, B) : X
    for each state i = 1, 2, ..., K do
        T_1[i, 1] ← π_i · B_{i y_1}
        T_2[i, 1] ← 0
    end for
    for each observation j = 2, 3, ..., T do
        for each state i = 1, 2, ..., K do
            T_1[i, j] ← max_k (T_1[k, j − 1] · A_{ki} · B_{i y_j})
            T_2[i, j] ← argmax_k (T_1[k, j − 1] · A_{ki} · B_{i y_j})
        end for
    end for
    z_T ← argmax_k (T_1[k, T])
    x_T ← s_{z_T}
    for j = T, T − 1, ..., 2 do
        z_{j−1} ← T_2[z_j, j]
        x_{j−1} ← s_{z_{j−1}}
    end for
    return X
end function
```

Probabilistic Graph Models

- Generalization of Bayes' Nets / Markov Models
- Directed Acyclic Graphs allow tractability.

Markov Decision Processes

- Return: $G_t = R_{t+1} + \gamma R_{t+2} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} = R_{t+1} + \gamma G_{t+1}$
- Value: $V_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$
- Action-value: $q_\pi(s,a) = \mathbb{E}_{\pi,p}[G_t | S_t = s, A_t = a]$
- Policy: $\pi(a|s)$ is prob. with which we act a from s.
- Bellman Eqns:
  - $\rightarrow V_\pi(s) = \mathbb{E}_\pi[q_\pi(s,a)] \rightarrow q_\pi(s,a) = \sum_{s',r} (r + \gamma V_\pi(s')) p(s', r | s, a)$
  - $\Rightarrow V_\pi(s) = \sum_{a \in A} \pi(a|s) \sum_{s',r} (r + \gamma V_\pi(s')) p(s', r | s, a)$.

Value Iteration: $V_{k+1}(s) = \max_{a \in A} \sum_{s',r} p(s', r | s, a)[r + \gamma V_k(s')]$

Policy Iteration

- Initialize π and loop to converge:
  - Policy Evaluation: Solve Bellman Eqns. for $\vec{V}_\pi$ (value iteration)
  - Policy Improvement: Solve for $\pi'(s) = \arg\max_a q_\pi(s, a)$

Graph Neural Nets

- General CNNs w/o ordered input
- $\vec{h}_u^{(k)}$ is embedding for node u at hidden layer k.
- $\psi^{(k)}(\vec{h}_v^{(k-1)}, \vec{h}_u^{(k-1)})$ is a message function that takes in embeddings at layer $k-1$ for nodes u, v.
- $\oplus$ is some message aggregator. could add, multiply, vectorize, etc.
- $N(u)$ are neigbors of u. — $\vec{m}_{u}^{(k)} = \psi^{(k)}(\vec{h}_v^{(k-1)}, \vec{h}_u^{(k-1)})$ is message between u and v.
- $\phi^{(k)}(\vec{h}_u^{(k-1)}, \vec{m}_u^{(k)})$ is some update function enacting layer transitions in GNN.
- General form: $h_u^{(k)} = \phi^{(k)}\left(\vec{h}_u^{(k-1)}, \oplus\left(\left\{\underbrace{\psi^{(k)}(\vec{h}_v^{(k-1)}, \vec{h}_u^{(k-1)})}_{\vec{m}_{vu}^{(k)}} \ \forall v \in N(u)\right\}\right)\right)$
- Example: CNN layer with 3x3 kernel, W, sigmoid activation, σ, and "node" (= coords), (x, y):

$$h_u^{(k)} = \sigma\left(\sum_{i,j \in \{-1,0,1\}} W_{i,j} \, h_{x+i,y+j}^{(k-1)}\right)$$

- Aggregators must be permutation invariant [permutation of inputs irrelevant].
- Hence, also translational, rotational, perspective, reflective in/equivariance.

Miscelloneous

- ISOMAP is a non-linear dim. reduction method.
- Lasso can be used to remove unwanted features (promotes zeroing out weights).
- t-SNE non-linear + stochastic, PCA linear + deterministic.
- OLS is MLE, Ridge is MAP with gaussian prior on weights, Lasso is MAP with laplace prior on weights.
- decision trees are non-differentiable w.r.t params.
- SVMs are trained using quadratic programming solvers.
- SVMs have hinge loss w/ ridge penalty.
- Hard margin SVM is soft-margin SVM with $\lim_{C \to \infty}$.
- KL minimization in SNE / t-SNE is an MLE objective.
- Decision trees are built greedily for practicality > optimality
- 1 v.s. All classification $\leq$ softmax multiclass for mutually excl.