Gaussian Distribution:

normal: $p(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$

multi: $p(x) = \frac{1}{\sqrt{|\Sigma|}\sqrt{2\pi}^d} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$

The covariance matrix Σ is a matrix such that each entry is $\Sigma_{ij} = \text{cov}(x_i, x_j)$. Diagonal is $\Sigma_{ii} = \text{var}(x_i)$. If Σ is diagonal, all x_i are independent

$\Sigma = E[XX^T] - \mu\mu^T$; PSD $x^T\Sigma x \geq 0$

$x \sim N(\mu, \Sigma)$; $Ax+b \sim N(A\mu+b, A\Sigma A^T)$

Unsupervised: clustering, partitioning, kmeans
K means ++, Kmedians, Vector Quantization, PCA,
Parametric Density, Non Parametric Density

SVM-like classifiers work with a boundary, a hyperplane (a line for 2-D data) that seperates two classes. Support vectors are the points closest to the boundaries. γ is the margin, the distance between the boundaries and the support vectors. The parameter θ is a vector. $\theta \cdot x$ gives predictions. About θ:

- Direction of θ defines the boundary. We can choose this.
- $\|\theta\|$ must be $1/\gamma$, as restricted by $\forall i : y^i\theta \cdot x^i \geq 1$. We cannot explicitly choose this; it depends on boundary. The restriction is turned into a cost in soft-margin SVM.

Perception picks misclassified points and updates θ just enough to classify it right
$\theta \leftarrow \theta + x^i$ or $\theta \leftarrow \theta - \nabla J(\theta)$; overfits w/outliers skew the boundary. Converges iff seperable.
Batch Eq: $\theta \cdot x = \sum \alpha^i y^i x^i \cdot x$; α = # of times point i was misclassified.

Hard Margin SVM maximizes the margin around the boundary. Technically, it minimizes the distance between boundary and the vectors closest to it. Batch eqn: $\theta = \sum \alpha_i y_i x_i$, where
$\min_\theta \|w\|^2$ such that $\forall i : y_i\theta \cdot x_i \geq 1$ Converges iff seperable. $\alpha_i = 1_i$ is support vector

Sometimes removing a few outliers lets us find a much higher margin, or margin at all

Soft Margin SVM is like hard margin SVM but penalizes misclassifications:
$\min_\theta \|w\|^2 + C\sum(1-y^i\theta \cdot x^i)$; Hyper parameter C is the hardness of the margin. Low C means more misclassification but larger soft margin.

	small C	big C
desire	maximize margin $1/\|w\|$	Keeps most slack variables zero or small
outliers	less sensitive	very sensitive
boundary	more "flat"	more curvy
danger	underfitting (misclassifies much training data)	overfitting (good training, bad testing)

KNN: Given an item x, find the "k" training items closest to x and return the result of a vote. Hyperparameter k, the number of neighbors. "Closest" can be defined by some norm (L2 by default). Overfits when k is super small.

Descision Tree: Recursively split on features that yeild the best split. Each tree has many nodes, which either split on a feature at a threshold or all the data the same way. Hyper parameters typically restrict complexity (max tree depth, min points at node) or penalize it. One particular one of interest d, the max # of nodes.
Overfits when tree is deep or when we are allowed to split on a very small # of items.

Bagging: Make multiple trees, each with a random subset of training items. To predict, take votes from trees. Hyperparameters: # of trees, proportion of items to subset.

Random forest is bagging, except, for each node, consider a random subset of features to split on. Hyperparameters: proportion of features to consider.

Support Vector Machines

Hyperplane = $\{w^Tx + b = 0\}$

Optimization: $\min_{w,b} \frac{1}{2}\|w\|_2$ s.t. $y^i(w^Tx^i + b) \geq 1, i = 1, \ldots, m$

Primal: $L_p(w,b,\alpha) = \frac{1}{2}\|w\|_2 - \sum_{i=1}^m \alpha_i(y^i(w^Tx^i+b) - 1)$

$\frac{\partial L_p}{\partial w} = w - \sum \alpha_i y^i x^i = 0 \Rightarrow w = \sum \alpha_i y^i x^i$

$\frac{\partial L_p}{\partial b} = -\sum a_i y^i = 0$ Note: $\alpha_i \neq 0$ only for Support vectors

→ substitute derivatives into primal to get the dual

Dual: $L_d(\alpha) = \sum_{i=1}^m \alpha_i - \frac{1}{2}\sum_{i=1}^m \sum_{j=1}^m y^i y^j \alpha_i \alpha_j (x^i)^T x^j$

KHT says $\alpha_n(y_n(w^Tx_n + b) - 1) = 0$ where $\alpha_n > 0$

Non seperable case, we add boundaries + penalize
$\min_{w,b} \frac{1}{2}\|w\|_2 + C\sum \xi_i$ s.t. $y^i(w^Tx^i + b) \geq 1 - \xi_i$

* each α_i has upperbound of $C \Rightarrow 0 \leq \alpha_i \leq C$

Regression

In general, loss function consist of two parts, the loss term and Regularization term
$J(\omega) = \sum_i \text{loss}_i + \lambda R(\omega)$

L2 regularization results in RIDGE REGRESSION
- used when A contains a null space

$\min_x \|Ax - y\|_2^2 + \lambda\|x\|_2^2 \rightarrow x^* = (A^TA + \lambda I)^{-1}X^Ty$

- works well for LARGE parameters

L1 regularization results in lasso regression
- used when x has a laplace Prior

→ gives sparce results - not good for feature reduction

Logistic Regression

Classify $y \in \{0,1\}$ → model $P(y=1|x)$

$h_\theta(x) = \frac{1}{1+e^{-\theta^Tx}} = P(y=1|x)$

$\frac{\partial h_\theta}{\partial \theta} = \left(\frac{1}{1+e^{\theta^Tx}}\right)^2 e^{-\theta^Tx} = h_\theta(1-h_\theta)$

$P(y|x;\theta) = (h_\theta(x))^y(1-h_\theta(x))^{1-y} \rightarrow$

$L(\theta) = \prod^m (h_\theta(x^i))^{y_i}(1-h_\theta(x^i))^{1-y_i} \rightarrow$

$\ell(\theta) = \sum^m y_i \log(h_\theta(x_i)) + (1+y_i)\log(1-h_\theta(x_i)) \rightarrow$

$\nabla_\theta \ell = \sum(y_i - h_\theta(x_i))x_i = X^T(y - h_\theta(x))$, (want max $\ell(\theta)$)

Stochastic: $\theta_{t+1} = \theta_t + \alpha(y_t^i - h_\theta(x_t^i))x_t^i$

Batch: $\theta_{t+1} = \theta_t + \alpha X^T(y - h_\theta(x))$

LDA and QDA

Classify $y \in \{0,1\}$, model $p(y) = \phi^y\phi^{1-y}$ and

$\ell(\theta, \mu_0, \mu_1, \Sigma) = \log(\prod_{i=1}^m p(x^i|y^i;\mu_0,\mu_1,\Sigma)p(y^i;\phi)) \rightarrow$

$\phi_{MLE} = \frac{1}{m}\sum^m 1\{y^i = 1\}$ and $\mu_{k,mle} = \text{avg}(x_i)$

$\Sigma_{MLE} = \frac{1}{m}\sum(x^i - \mu_{y^i})(x^i - \mu_{y^i})^T$

If $p(x|y)$ is multivariate gaussian w/ shared Σ, then $p(y|x)$ is logistic function. The converse is NOT true.

LDA makes stronger assumptions about the data than does linear regression.

$h(x) = \text{argmax}_k -\frac{1}{2}(x-\mu_k)^T\Sigma^{-1}(x-\mu_k) + \log(\pi_k)$

For QDA, it is the same except for each class has a unique covariance matrix.

$h(x) = \text{argmax}_k -\frac{1}{2}\log(|\Sigma_k|) - \frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k) + \log(\pi_k)$

Underfitting = high bias; overfitting = high variance

Curse of Dimensionality:

As the number of dimensions increases, everything becomes farther apart. Our low dimension intuition falls apart.

Consider a hypercube/hypersphere ratio, its close to zero at d=10.

How to deal w/ this:

1. Get more data to fill the space
2. Get better features, reducing the dimensionality and pathing the data closer together.
3. Use better distance metric

Decision Trees Cont.

- to reduce overfitting, we can prune using a validation set and we can limit the depth.

Hueristic: For classification, maximize the information gain
$\max_j H(D) - \sum_{x \in X_j} P(X_j = x) \cdot H(D|X_j = x)$
where $H(D) = \sum_{c \in C} P(y=c)\log[P(y=c)]$ is the entropy of the dataset. C is the set of classes each data point can take, and $P(C=c)$ is prop of data points w/class c.

→ For regression, we want to minimize the variance. Same as above but H is replaced w/ Var. Pure leaves correspond to low variance, and result is mean of the current leaf.

Clustering: Unsupervised
Distance functions. Suppose we have two sets of points.

- **Single linkage** is minimum distance between members.
- **Complete linkage** is maximum distance between members.
- **Centroid Linkage** is distance between centroids.
- **Average Linkage** is average distance between all pairs.

Hierarchical:

- **Agglomerative:** Start w/ n points, merge 2 closest clusters with some measure, such as: Single-link (closest pair), Complete Link (furthest pair), Average Link (avg of all pairs), Centroid (centroid distance).

Note: Single and Complete Link are sensitive to outliers.

Divisive: Start with single cluster, recursively divide clusters into two subclasses.

Probability + Matrix Review

Bayesian Decision Theory

Bayes Rule: $P(\omega|x) = \frac{P(x|\omega)P(\omega)}{P(x)}$; $P(x) = \sum P(x|\omega_i)P(\omega_i)$

$P(x,w) = P(x|w)P(w) = P(w|x)P(x)$

$P(\text{error}) = \int_{-\infty}^{\infty} P(\text{error}|x)P(x)\,dx$

$P(\text{error}|x) = \begin{cases} P(\omega_1|x) & \text{if we decide } \omega_2 \\ P(\omega_2|x) & \text{if we decide } \omega_1 \end{cases}$

0-1 Loss: $\lambda(\alpha_i, \omega_j) = \begin{cases} 0 & i=j \text{ (correct)} \\ 1 & i \neq j \text{ (mismatch)} \end{cases}$

Expected Loss (Risk): $R(\alpha_i|x) = \sum_{j=1}^c \lambda(\alpha_i|\omega_j)P(\omega_j|x)$

0-1 Risk: $R(\alpha_i|x) = \sum_{j \neq i} P(\omega_j|x) = 1 - P(\omega_i|x)$

Generative vs Discriminate Model

Generative: Model class conditional density $p(x|y)$ for $p(y|x) \propto p(x|y)p(y)$ or model joint density $p(x,y)$ and marginalize to find $p(y=k|x) = \int_x p(x, y=k)dx$

Discriminative: Model condition $p(y|x)$

class conditional $P(x|y)$; posterior $P(y|x)$

prior $P(y)$; evidence $P(x)$

Loss functions

Binomial deviance $= \log[1 + e^{-yf(x)}]$
- minimizing function $f(x) = \log\frac{P(y=+1|x)}{P(y=-1|x)}$

SVM hinge loss $= [1 - yf(x)]_+$
- minimizing function $f(x) = \text{sign}\left(P[y=+1|x] - \frac{1}{2}\right)$

Squared Error $= [y - f(x)]^2 = [1 - yf(x)]^2$
- minimizing function $f(x) = 2P[y=+1|x] - 1$

"Huberized" Squared hinge loss
$= \begin{cases} -4yf(x) & \text{if } yf(x) < -1 \\ [1-yf(x)]_+^2 & \text{otherwise} \end{cases}$ ↕ same
- minimizing function $f(x) = 2P[y=+1|x] - 1$

Gradient Descent

$\theta_{t+1} = \theta_t - \alpha\nabla f(\theta_t)$, for minimizing

Gradients:

$\frac{\partial y}{\partial x} \triangleq \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_1}{\partial x_n} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}$

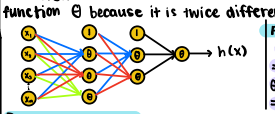
$\frac{\partial(Ax)}{\partial x} = A^T$ $\frac{\partial(x^TA)}{\partial x} = A$

$\frac{\partial(x^Tx)}{\partial x} = 2x$ $\frac{\partial(x+BA)}{\partial A} = B^T$

$\frac{\partial(x^TAx)}{\partial x} = (A + A^T)x$

Neural Networks

- Neural Networks explore what you can do by combining perceptrons, each of which is a simple linear classifer. We use a soft threshold for each activation function θ because it is twice differential.



Activation functions:

$\theta(s) = \tanh(\omega) = \frac{e^s - e^{-s}}{e^s + e^{-s}}$

$\Rightarrow \theta'(s) = 1 - \theta^2(s)$

$\theta(s) = \sigma(s) = \frac{1}{1+e^{-s}}$

$\Rightarrow \theta'(s) = \sigma(s)(1-\sigma(s))$

Error Functions:

Cross Entropy Loss: $\sum_{i=1}^{num} y\log(h_\theta(x)) + (1-y)\log(1-h_\theta(x))$

Mean Squared Error: $\sum_{i=1}^{num} (y - h_\theta(x))^2$

Notations:

1. w_{ij}^ℓ is the weight from neuron i in layer $(\ell-1)$ to neuron j in layer ℓ. There are d^ℓ nodes in the ℓ^{th} layer.
2. L layers, where L is the output layer and data is the 0^{th} layer.
3. $x_j^\ell = \theta(s_j^\ell)$ is the output of a neuron. Its the activation function applied to the input signal. $s_j^\ell = \sum_i w_{ij}^\ell x_i^{(\ell-1)}$
4. $e(w)$ is the error as a function of the weights.

The goal is to learn the weights w_{ij}^ℓ. We use gradient descent, but the error is non-convex so we tend to local minima. The naive version takes $O(w^2)$. Backpropagation, an algorithm for efficient computation of the gradient, takes $O(w)$.

$\nabla e(w) = \frac{\partial e(w)}{\partial w_{ij}^\ell} = \frac{\partial e(w)}{\partial s_j^\ell}\frac{\partial s_j^\ell}{\partial w_{ij}^\ell} = \delta_j^\ell x_i^{\ell-1}$

Final Layer: $\delta_j^L = \frac{\partial e(w)}{\partial s_j^L} = \frac{\partial e(w)}{\partial x_j^L}\cdot\frac{\partial x_j^L}{\partial s_j^L} = e'(x_j^L)\theta'_{out}(s_j^L)$

General: $\delta_i^{\ell-1} = \frac{\partial e(w)}{\partial s_i^{\ell-1}} = \sum_{j=1}^{d(\ell)} \frac{\partial e(w)}{\partial s_j^\ell} \times \frac{\partial s_j^\ell}{\partial x_i^{\ell-1}} \times \frac{\partial x_i^{\ell-1}}{\partial s_i^{\ell-1}}$
$= \sum_{j=1}^{d(\ell)} \delta_j^\ell \cdot w_{ij}^\ell \cdot \theta'(s_i^{\ell-1})$

Process:

1. Initialize all weights w_{ij}^ℓ at random
2. for $t = 0, 1, 2, \ldots, d$
   - pick $n \in \{1, 2, \ldots, N\}$
   - Forward: compute all x_{ij}^ℓ
   - Backwards: compute all δ_{ij}^ℓ
3. Update the weights $w_{ij}^\ell \leftarrow w_{ij}^\ell - \eta x_i^{\ell-1}\delta_j^\ell$
4. Iterate to the next step until you reach goal, d
5. Return the final weights w_{ij}^ℓ

LDA vs Logistic Regression:

LDA Advantages:

- LDA stable for well seperated classes. Logistic regression is surprisingly unstable
- Easy and elegant for more than 2 classes.
- LDA slightly more accurate when class nearly normal, especially small n

Logistic Regression:

- More emphasis on decision boundary; always seperates linearly seperable points thru weighting of points.
- More robust than non-Gaussian distributions.
- Naturally fits labels between 0 and 1

Clustering (cont.)

Partitioning: Partition the data into k mutually exclusive exhaustive groups (ie: encode $h = C(i)$). Iteratively reallocate to minimize some loss function. Finding the correct partitions is hard. Use a greedy algorithm called k-means (coordinate descent). Loss function is non-convex and thus we find a local minima.

k-means: Choose clusters at random, calculate centroid of each cluster, reallocate objects to nearest centroid, then repeat. Works with spherical, well seperated clusters of similar count and volumes.

k-means ++: Initilize clusters one by one. $D(x)$ = distance of point X to nearest cluster. $P(x \text{ is new cluster center}) \propto D(x)^2$

k-medians: Works with arbitrary distance/dissimilarity metric, the centers μ_k are represented by data points. Is more restrictive and thus has higher loss.

General Loss:
$\sum_{n=1}^N \sum_{k=1}^K d(x_n, \mu_k) r_{nk} = 1$ if x_n is in cluster k, and 0 otherwise.

Parametric Density Estimation

Mixture Models: Assume PDF is made up of multiple gaussians with different centers.

$P(y) = \sum^{n_c} P(c_i)P(x|c_i)$ with objective function as log likelyhood of data. Use EM to estimate this model

E-step: $P(\mu_i|x_n) = \frac{P(\mu_i)P(x|\mu_i)}{\sum_j P(\mu_j)P(x|\mu_j)}$

M-step: $P(c_i) = \frac{1}{n_e}\sum_{k=1}^{n_e} P(\mu_i|x_k)$

$\mu_i = \frac{\sum_k x_k P(\mu_i|x_k)}{\sum_k P(\mu_i|x_k)}$

$\sigma_i^2 = \frac{\sum_k (x_k - \mu_i)^2 P(\mu_i|x_k)}{\sum_k P(\mu_i|x_k)}$

Non Parametric Density Estimation

Can use histogram or kernal density estimation (KDE).

KDE: $P(x) = \frac{1}{n}\sum k(x - x_i)$ is a function of the data.

The kernel k has the following properties: symmetric, Normalized $\int_{\mathbb{R}^d} k(x)dx = 1$ and $\lim_{\|x\|\to\infty} \|x\|^a K(x) = 0$

The bandwidth is the width of the kernel function. Too small = jagged results, too large = smoothed out results.

Principle Component Analysis

First run singular value decomposition on pattern matrix, X.

1. Subtract mean from each point
2. (sometimes) Scale by variance.
3. Compute covariance $\Sigma = X^TX$
4. Compute eigenvalues/vectors of $\Sigma = VSV^T$ (spectral theorem)
5. get back $X = X\Sigma = (XV)SV^T = USV^T$

S contains the eigen values of transformed features. The larger the S_{ii}, the larger the variance of that feature. We want the K largest features, so we find the indices of the K largest items in S and we only keep these entries in U and V.

MLE: To do gaussian discriminant analysis, we must fit gaussians to the sample points and estimate the class prior probabilities.

Definition: MLE is method of estimating the parameters of a statistical model by picking the parameters that maximize the likelyhood function $\mathcal{L}$

- one way of density estimation

Likelyhood of a Gaussian given Sample points $X_1 \cdots X_n$ (all probs multiplied)
- likelyhood $= \prod^n p(x_i = x_i)$
- Log Likelyhood $= \log(\prod^n p(x_i = x_i))$

Then we can use gradients to find out $\hat{\mu} = \bar{X}$ and $\hat{\sigma}^2 = \frac{1}{nn}\sum\|x_i - \mu\|^2$

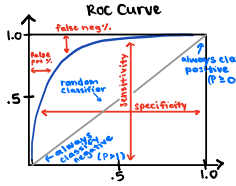
- We dont know μ exactly, so we must use the estimator μ to solve for $\hat{\sigma}^2$
- We use the sample mean and variance for $N(\mu, \sigma^2)$

good luck! -Jilli from 4:39am

ROC curves

- Reciever Operating Characteristics
- Way to evaluate classifier after it is trained by running on the test set or validation set.
- We can change the posterior probability threshold for Gaussian discriminant Analysis or logistic regression to trade off false positives vs false negatives.

Specificity: Horizontal distance from curve to the right (1 - false pos rate)
Sensitivity: Vertical distance from bottom to curve (1 - false neg rate)
Upper right: Always classify positive
Lower left: Always classify negative
Diagonal: Random Classifiers
- Rough measure of effectiveness is area under the curve. Max area is 1, for a classifier that is always right. ½ for random classifier.



Generative / Discriminative cont.

Advantages:
A+G - P(Y|X) tells you probability guess was wrong, you can diagnose outliers

Disadvantages:
G - often hard to estimate distributions accurately. real distributions rarely match.

Neural Networks (cont)

Input standarization: batch norm makes sure each feature has similar statistics over a minibatch
LayerNorm makes sure each data point has similar statistics over all its features.
Skip Connections: add input to result later in the network
Dropout: randomly 0 out some nodes during training and is a form of Regularization encouraging network to learn diffuse rep.

Transformers:

- motivated by natural language processing, since one word may not translate directly, as well as the sentences having possibly irrelevant words (ie: a, the, etc)
- inductive bias - focus on pairings of words

Encoder:
- full transformer includes cross-attention
  - Similar mechanism, just quering into a different sequence
- Multihead Attention
  - Include multiple key/query/value embedding, each is independant
  - Allows for learning different features
- LayerNorms
  - standardizes features at each position in a sentence
  - stabalizes training/inputs to each embedding

Transformer VIT
- take each patch of the image, flatten it, and embed it with a linear layer
- Now treat those embeddings like you would treat word embeddings and proceed as usual.
- Can put a MLP onto some output to classify

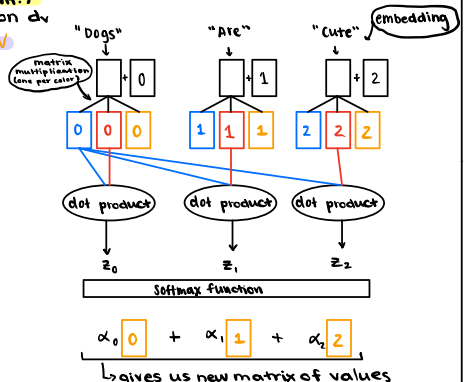
Self Attention (transformer cont.)

- each vector has dimension d_v

$\text{Attention}(Q,K,V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_K}}\right)V$

Queries - used to index information from other positions
Keys - indexed into to release the "value" from this position
Values - weighted together to form output vector

Multihead Attention:
- learn different pairwise dependancies
- each head is computed on its own, so we can parallelize
- stack output vectors and project to desired output dimensions
  - ↳ has dimension of $n_{heads} * d_v$
- project downwards w/ matrix multiplication



↳ gives us new matrix of values

Bias-Variance Decomposition

- Model $X_i \sim D$, $\epsilon_i \sim D'$, $y_i = g(x_i) + \epsilon_i$ (D' has 0 mean).

We fit hypothesis h (a random variable) to X,y
There are two sources of error in a hypothesis, h:

1. Bias: err due to inability of hypothesis h to fit g perfectly (ie: fitting a quadratic to a linear function)

2. Variance: err due to fitting random noise in data

- Consider arbitrary point $z \in R^d$, $y = g(z) + \epsilon$, $\epsilon \sim D'$
- The mean comes from g and the variance comes from ϵ: $E(y) = g(z)$; $Var(y) = Var(\epsilon)$
- $R(h) = E[L(h(z), y)]$. We are taking a mean over the probability distribution of the hypothesis. The bias-variance decomposition of the risk function is as follows:

$$= E[(h(z) - y)^2]$$
$$= \underbrace{(E[h(z)] - g(z))^2}_{\text{bias}^2 \text{ of method}} + \underbrace{\text{var}(h(z))}_{\text{var of method}} + \underbrace{\text{var}(\epsilon)}_{\text{irreducible error}}$$

Visually the bias is the expected hypothesis. Variance is the expected square distance between a random test point and the actual curve.

Consequences:
- Training error reflects bias but not variance; test error reflects both
- We cant precisely measure bias or variance because we dont know g and our noise model might be wrong. However, we can test learning algorithms by choosing g and making synthetic data.

Bias:
- underfitting means too much bias
- if h can fit g exactly, for many distributions bias → 0 as n → ∞
- if h cannot fit g well, bias is large at most points
- Adding a good feature reduces bias, adding a bad feature rarely increases it.
- Bias can be 0 when hypothesis function can fit the real one.

Variance:
- most overfitting caused by too much variance
- For most distributions, var → 0 as n → ∞
- Adding a feature usually increases variance (aka dont add feature unless it reduces bias more)
- Variance portion of RSS (overfitting) decreases with 1/n sample points, increases with d features,

Irreducible Noise
- Cant reduce irreducible error
- Noise in test set only affects Var(ε). Noise in the training set only affects bias and Var(h)

Gaussian Discriminant Analysis cont.

Normal PDF: $f(x) = n(q(x))$, $n(q) = \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} e^{-q/2}$, $q(x) = (x-\mu)^T \Sigma^{-1} (x-\mu)$
($R \to R$, exponential; $\det \Sigma$; $R^d \to R$, quadratic)

- $\Sigma = V \Gamma V^T$ is the covariance matrix, where eigenvalues of Σ are the variances along the eigenvectors
- $\Gamma_{ii} = \sigma_i^2$
- $\Sigma^{1/2} = V \Gamma^{1/2} V^T$ maps spheres to elipsoids, where eigen values of Σ are the Gaussian widths/ellipsoid radii / standard deviations
- $\sqrt{\Gamma_{ii}} = \sigma_i$
- $\Sigma^{-1} = V \Gamma^{-1} V^T$ is the precision matrix.

Sphering a Gaussian

$X \sim N(\mu, \Sigma)$; we want $z = f(x)$ s.t. $Z \sim N(0, I)$

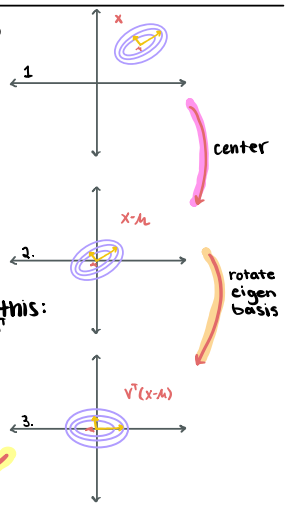
$p(x) \propto \exp(\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu))$
$p(z) \propto \exp(\frac{1}{2} z^T z)$
$q(x) = (x-\mu)^T \Sigma^{-1} (x-\mu)$

Then we want $q(f(x)) = f(x)^T f(x)$ for some transformation.

$$q(x) = (x-\mu)^T \Sigma^{-1} (x-\mu)$$
$$= (x-\mu)^T V \Lambda^{-1} V^T (x-\mu)$$
$$= [(x-\mu)^T V \Lambda^{-1/2}][\Lambda^{-1/2} V^T (x-\mu)]$$
$$= f(x)^T f(x)$$

↳ where $f(x) = \Lambda^{-1/2} V^T (x-\mu)$

So $\therefore z = \underbrace{\Lambda^{-1/2}}_{\text{divide each eigen vector direction by standard deviation in that direction}} \underbrace{V^T}_{\text{rotate the eigen basis}} \underbrace{(x-\mu)}_{\text{center the distribution around 0}}$



Another way to write this:
$z = A^{-1}(x-\mu)$ for $\Sigma = AA^T$
$A = V\Lambda^{1/2}$
$\therefore A^{-1} = \Lambda^{-1/2} V^T$

Convolution Neural Network

- Inductive Bias: exploiting human knowledge of a problem in order to simplify our models
- Fully connected networks (MLP) can in theory learn any function, however, they have many parameters (lots of memory + run time constraints) and they require ridiculous amounts of data to learn high dimensions
- Some problems have structural invariance and equivariance
- Want # of parameters to be constant wrt image size

Pooling Layer - used to reduce spacial dimensions
- Sliding window method similar to convolution but has no parameters.
- Avg pooling (take average value)
  - extracts features smoothly
- max pooling (take maximum value)
  - extracts most important features

Ex. $\begin{bmatrix} 0&0&0&0&0 \\ 0&1&1&1&0 \\ 0&1&0&1&0 \\ 0&1&1&1&0 \\ 0&0&0&0&0 \end{bmatrix}$ 3x3 filter w/ stride = 2

max pooling = $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$

avg pooling = $\begin{bmatrix} \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} \end{bmatrix}$

# of parameters = $(s+1) \times k$
k = # of filters; s = dimensions of filter (ie: 3x5x2 = 30 = s)

shape of output tensor w/ k x k filter
$[H', W'] = 1 + ([H, W] + 2 \times pad - k)/stride)$

- Fully connected layers typically go on the end of models w/ reduced dimension features

Math Appendix

• Vectors
- Dot Product: $x \cdot y = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n$
- Euclidean norm: $\|x\| = \sqrt{x \cdot x} = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$
  - Normalizing a vector means replacing x with $\frac{x}{\|x\|}$
- $\cos(\theta) = \frac{x \cdot y}{\|x\| \|y\|} = \frac{x}{\|x\|} \cdot \frac{y}{\|y\|}$

• Continous Distributions
- $Pr[X \in [x_1, x_2]] = \int_{x_1}^{x_2} f(x)\,dx$
- $\mu = E(X) = \int_{-\infty}^{\infty} x f(x)\,dx$
- $E[g(x)] = \int_{-\infty}^{\infty} g(x) \cdot f(x)\,dx$
- $\sigma^2 = E[(X-\mu)^2] = E(X^2) - \mu^2$

Linear Algebra
- Basics
  - The column space, also called the range or span of X is Range(X) := {y | y = Xv}
  - The row space is Row(X) := {y | y = X^Tv}
  - The null space, or the kernel of X is defined as N(X) := {v | Xv = 0}
  - The orthogonal compliment of a subspace U_1 is a subspace $U^\perp$ such that $u \in U, u' \in U^\perp \to u \cdot u' = 0$
  - Row(X) = Range(X^T)
  - $N(X)^\perp$ = Row(X)
  - $N(X^T X) = N(X)$
  - Row($X^T X$) = Range($X^T X$)
  - Rank Nullity Theorem: Rank(X) + N(X) = m
  - Rank(A) + Rank(B) − n ≤ Rank(AB) ≤ min{rank(A), rank(B)}