Contents

Machine Learning CS 189

Contents

Classification: August 30

Scribe: Brandon McKinzie

- **Goal:** Want to prove that w is normal to decision boundary.
  - Starting axiom: Any vector x along the decision boundary satisfies, by definition,

$$w \cdot x + \beta = 0 \tag{1}$$

  - Let x and x' be two such vectors that lie on the decision boundary. Then the vector $x' - x$ points from x to x' and is parallel to the decision boundary. If w really is normal to the decision boundary line, then

$$\begin{aligned} w \cdot (x' - x) &= 0 \\ &= w \cdot x' - w \cdot x \\ &= (w \cdot x' + \beta) - (w \cdot x + \beta) \\ &= 0 + 0 \end{aligned} \tag{2}$$

  - Euclidean distance of x to decision boundary:

$$\tau = -\frac{(w \cdot x + \beta)}{||w||} = -\frac{f(x)}{||w||} \tag{3}$$

  - The **margin** is can be found as the minimum over all training data τ:

$$M = \min_{i \in 1 \cdots n} \frac{|f(x_i)|}{||w||} \tag{4}$$

Gradient Descent: September 1

Scribe: Brandon McKinzie

- **Optimization:** maximize goals/minimize cost, subject to constraints. However, can model a lot while ignoring constraints.
- Main optimization algorithm is **stochastic gradient descent**.
- The **SVM**[1] is just another cost function. Want to minimize[2]

$$C\sum_{i=1}^{n}\Big(1-y_i(w\cdot x_i+\beta)\Big)_+ + ||w||^2 \tag{5}$$

with respect to the **decision variables** (w, β); Note that C is a **hyperparameter**.

[1] so-called because you could represent the decision boundary as a set of vectors pointing to the hyperplane.

[2] Notation: $(z)_+ = max(z, 0)$.

Machine Learning **Fall 2016**

Stochastic Gradient Descent: September 6

Table of Contents Local *Scribe: Brandon McKinzie*

- Review: minimize cost function $f(w)$ over w. Take gradient; set to zero to solve for w.
- If can't solve analytically, then Gradient Descent:

$$w_{k+1} = w_k - \alpha_k \nabla f(w_k) \tag{6}$$

- For convex f, can always find solution. Guaranteed global minimum.
- Cost functions of form: minimize $\sum \text{loss}(w_i(x_i, y_i)) + \text{penalty } (w)$.
- SVM example:

$$\min C/n \sum (1 - y_i w^T x_i)_+ + ||w||^2$$

. Add squared norm because better margins and better classifications. Also, because algorithms converge faster. C is the **regularization parameter**. "Do I fit the data, or make w simple?". Doesn't change optimal set, just changes the "Cost" (wat).

- Want algorithm constant in number of data points n[3].
- Unbiased estimate of the gradient:
  - Want expected vlaue of g to be gradient of cost function.
  - Sample i uniformly at random from $\{1, \ldots, n\}$.
  - Then set g to gradient of loss at ith data point.
- SGD:
  - initialize w_0, $k = 0$.
  - (Repeat) sample i at uniform. Do weight update on the loss for i term. Until converged.
  - Follow the expected value of the gradient (rather than the true gradient) until converge. Following a noisy version. As long as variance is bounded, direction will be more or less correct. For large number of data points n, will be pretty good.

[3]Regular GD is linear in n

- Numerical example:
  - $f(w) = 1/2n \sum (w - y_i)^2$. Assumes x always 1.
  - Solve $\nabla f(w) = 0 = 1/n \sum (w - y_i) = 0$.
  - Optimal $w = 1/n \sum y_i$. The empirical mean.
  - Init $w_1 = 0$. Set $\alpha_k = 1/k$. Where k is kth update reference.
  - $w_2 = w_1 - \alpha_k \nabla \text{loss } () = y_1$. Where loss the grad of f.
  - $w_3 = w_2 - \alpha_2(w_2 - y_2) = y_1 - \frac{1}{2}(y_1 - y_2) = \frac{y_1 + y_2}{2}$.
  - $w_4 = \cdots =$. idk
  - Lesson: order we passed through data didn't matter. One pass over all data points leads to optimal w. Why advocate randomness then? He uses sum of trig function example to illustrate how SGD can struggle if done in order, but converge much quicker when randomly sampled.
- Illustrates "**region of confusion**". Coined by Bertsekas. Different convex functions along w. Rapid decrease in error early on iterations means we are far outside this region. Constant α means you'll jiggle around later iterations. That is why you do diminishing α; helps in region of confusion.
- Most important rules of SGD: (buzzwords)
  - shuffle! Can speed up by as much as 20x.
  - diminishing stepsize (α learning rate decay). After n steps, set $\alpha = \beta \cdot \alpha$.
  - **momentum**. $w_{k+1} = w_k - \alpha \nabla l_i(w_k) + \beta_k(w_k - w_{k-1})$. Momentum is in final term. Typical value is 0.9.
- Notation: $e(z) = (z < 1)$. Evaluates to 1 or 0.

Machine Learning **Fall 2016**

Risk Minimization & Optimization Abstractions: September 8

Table of Contents Local

Scribe: Brandon McKinzie

- Where do these optimization problems come from?
  - General framework: minimizing an average loss + λ penalty.
  - Loss: measures data fidelity.
  - Penalty: Controls model complexity.
  - Features/representation: How to write (x_i, y_i).
  - Algorithms: ∇ cost(w) = 0.
  - **Risk**: Integral of loss over probability(x, y).
  - Empirical risk: Sample average. Converges to true Risk with more points; variance decreases.
- Begins discussion of splitting up data.
  - Let some large portion be the **Training set** and the small remaining data points be the **Validation set**.

$$R_T = \frac{1}{n_T} \sum_{train} loss(w, (x_i, y_i)) \tag{7}$$

$$R_V = \frac{1}{n_V} \sum_{val} loss(w, (x_i, y_i)) \tag{8}$$

  - By law of large numbers, can say the R_V will go like $\frac{1}{n_V}$. Looking lots of times at validation set becomes a problem with $n_V \approx 10^4$.
- Classification example:
  - **Hinge** loss: $(1 - yw^T x)_+$. Means you're solving a SVM.
  - Least-Squares: $(1 - yw^T x)^2$. Bayes classifiers.
  - In practice, hinge and LS perform basically the same.
  - Logistic loss useful for MLE.
- Most important theorem in machine learning: Relates risk with empirical risk:

$$R[w] = \underbrace{R[w] - R_T[w]}_{\text{generalization err}} + \underbrace{R_T[w]}_{\text{train err}} \tag{9}$$

$$\approx R_V[w] - R_T[w] + R_T[w] \tag{10}$$

- One vs. all classifation MNIST example: Have a classifier for each digit that treats as (their digit) vs. (everything else). Choose classifier with highest margin when classifying digit.
- **Maximum Likelihood**:
  - Have $p(x, y; w)$. Pick the w that makes data set have highest probability.
  - Assumes data points come independently.
  - Can get same result by minimizing the negative log avg.
- More than most things (like loss functions) are choosing the **features**. **LIFT THE D**. Conic sections because why not.
- N-grams. **Bag of words**.
  - x_i = number occurrences of term i. Count number of times each word appears in some document.
  - The two-gram is the *lifted* version. x_{ij} = number occurrences of terms i, j in same context. Count number of terms two words, e.g. appear in the same sentence, or next to each other. Like a quadratic model.
- Histograms
  - $\hat{x}_{ij} = 1$ if $x_i \in bin(j)$ else 0.
  - e.g. histograms of image gradients in the notes.
- "If you have too many features, then you have to have a penalty." - D.J. Khaled. i.e. if $d > n$, must use pen(w). Never need to have $d > n$, becaues of **Kernel trick**.

Machine Learning **Fall 2016**

Decision Theory: September 13

Table of Contents Local *Scribe: Brandon McKinzie*

- **Decision Theory:**
  - Given feature distributions conditioned on class values (e.g. ± 1).
  - Goes over Bayes rule like a pleb.
  - Classification depends on loss function.
- Loss function can be **asymmetric**. e.g. would rather misclassify as cancer — normal rather than misclassify normal — cancer. So to minimize expected loss, may prefer to be wrong more on some misclassification than another.
- Can minimize on probability of error.

$$\min \left[Pr(\text{error}) = \int Pr(\text{error}|x) Pr(x) dx \right] \tag{11}$$

  The area (under) of overlap between conditionals (think hw problem w/Gaussians) is Pr(error). If K classes, similarly, classify as the maximum conditional, and error is 1 - $\Pr()_{max}$.
- **Modified rule**:

$$\min \sum_k L_{kj} P(c_k|x) \tag{12}$$

  where L_{kj} is loss where true class is k but classify as j. "by integrating over x, we can compute the expected loss."
- Loss function in regression pretty clear (e.g. least squared loss). Classification is less so. Can use the **"Doubt option"**. Classifier humility (lolololol)[4]. In some range of inputs near the decision boundary, just say "idk".
- Good classifiers have expected loss closer to Bayes risk, given certain choice of features.
- **NOTE:** Jitendra praises to the lord Gauss. Note 2: Jitendra makes another god joke.
- Three ways of building classifiers:
  - **Generative:** Model the class conditional distribution $P(\widetilde{x}|c_k)$. Model priors $P(c_k)$. Use bayes duh. Want to understand distributions under which data were *generated*. Physicists can get away with this shit. They have "models" [5]

[4] Wanna see my posterior

[5] fucking magnets how do they work?

  - **Discriminative:** "Fuck it" method. Model $P(c_k|\widetilde{x})$ directly.
  - Find decision boundaries.

- Posterior for Gaussian class-conditional densities :
  - $P(x|c_1)$ and $P(x|c_2) \sim \mathcal{N}(\mu_i, \sigma^2)$.
  - Univariate gaussian example *like a bitch*.
  - Posterior probabilities $P(c_i|x)$ turn out to be logistic σ functions.

Machine Learning **Fall 2016**

Multivariate Gaussians and Random Vectors: September 15

Table of Contents Local *Scribe: Brandon McKinzie*

⇒ Recht's Decision notation: $P(x|H_0)$. **RECHT CAN'T GET THE MIC ON HERE WE GO**

⇒ Want to discuss case of x being non-scalar. ***Random Vectors.***

- Def: vector x with probability density $p(x) : \mathbb{R}^n \to \mathbb{R}$.
- Example density is the **multivariate gaussian**.
- Usually want to know $Pr(x \in A) = \int_A p(x)dx_1 \dots dx_n$, the prob that x lives in set A.
- Properties of random vectors: **mean and covariance**.

⇒ Let $f : \mathbb{R}^n \to \mathbb{R}^m$. Expected value of f:

$$\mathbb{E}[f(x)] = \int \int f(x)p(x)dx_1 \dots dx_n \tag{13}$$

⇒ **Covariance Matrix**. A matrix. $\Lambda = \Lambda^T$.

$$\Lambda = \mathbb{E}[(x-\mu)(x-\mu)^T] = \mathbb{E}[xx^T] - \mu_x\mu_x^T \tag{14}$$

$$var(x_1) = \mathbb{E}[(x_1 - \mu_{x_1})2] = \Lambda_{11} \tag{15}$$

⇒ Let $v \in \mathbb{R}^n$. Then $var(v^Tx) = v^T\Lambda v \geq 0 \Rightarrow \Lambda_x$ is ***positive semidefinite.***

- Suppose A is some square matrix, λ is an eigval of A with corresponding eigvec x if $Ax = \lambda x$. Larger eigenvalues tell about how much variance in a given direction.
- Eigenvectors are, like, eigen*directions*, man.
- **Spectral Theorem**: If $A = A^T$, then $\exists S = v_1, \dots, v_n \in \mathbb{R}^n$ such that $v_i^Tv_j = 0$, $i \neq j$, and $Av_i = \lambda_i v_i$ and $\lambda_i \in \mathbb{R}$.
- Because matrix of eigenvectors has vectors linearly independent, invertible.
- A p.d $\Rightarrow B^TAB$ is p.s.d. $\forall B$.
- If A is p.s.d., then $f(x) = x^TAx$ is *convex.*

$\Rightarrow$ **Multivariate Gaussian**

$$p(x) = \frac{1}{det(2\pi\Lambda_x)^{1/2}} \exp\Big[-\frac{1}{2}(x-\mu_x)^T\Lambda_x^{-1}(x-\mu_x)\Big] \tag{16}$$

If $\mu_x \in \mathbb{R}^n,\ \Lambda_x$ p.d. $\Rightarrow$ p(x) is a density.

$\Rightarrow$ What happens if covariance is diagonal? Then the the vars are indepenedent.

- Λ_x diagonal,
- x Gaussian
- $\Rightarrow x_1, \ldots, x_n$ independent.

Machine Learning **Fall 2016**

Maximum Likelihood: September 20

Table of Contents Local *Scribe: Brandon McKinzie*

- **Estimation**: hypothesis testing on the continuum.
- **Maximum Likelihood:** Pick the model so that $p(\text{data}|\text{model})$, the likelihood function, is maximized.
- Treat model as random var. Then maximize p(model—data), "**maximum a posteriori**". Assume uniform priors over all models. Flaw is assuming model is a random variable.
- ML examples:
  - *Biased Coin.* Flipping a coid.

$$P(X = x) = \binom{n}{x} p^x (1-p)^{n-x} = P(x|p) \tag{17}$$

    See $n = 10, 8$ heads. Choose estimate $\hat{p} = \frac{4}{5}$. Binomial is not concave, when you take the log it becomes concave.

- Gaussians.

$$x1, \ldots, x_n \sim \mathcal{N}(\mu, \sigma^2) \tag{18}$$

  independent samples.

$$P(\{x_i\}|\mu, \sigma^2) = \prod P(x_i|\mu, \sigma^2) \tag{19}$$

  where each term in prod is standard Gaussian PDF. Next, take the log.

$$\log P(\{x_i\}|\mu, \sigma^2) = \sum -\frac{x_i - \mu}{2\sigma^2} - \log \sigma - 1/2 \log 2\pi \tag{20}$$

  Ideally, best estimates for mean and variance:

$$\hat{\mu} = \frac{1}{n} \sum_i x_i \tag{21}$$

$$\hat{\sigma}^2 = \frac{1}{n} \sum_i \left(x_i - \hat{\mu}\right)^2 \tag{22}$$

- Multivariate Gaussian:

$$P(x|\mu, \Lambda) = \frac{1}{\det 2\pi\Lambda^{1/2}} \exp(-\frac{1}{2}(x-\mu)^T \Lambda^{-1}(x-\mu) \tag{23}$$

Machine Learning **Fall 2016**

LDA & QDA: September 22

Table of Contents Local *Scribe: Brandon McKinzie*

- How do we build classifiers?
  - ERM. Minimize

$$\frac{1}{n}\sum(\text{loss}) + \lambda\text{pen} \tag{24}$$

    Equivalent to discriminative
  - Generative models. Fit model to data, use that model to classify. For each class C, fit $p(x|y = C)$. Estimate $p(y = C)$. To minimize $Pr(err)$, pick y that maximizes $P(y|x)$ via Bayes rule.
  - Discriminative. Fit $p(y|x)$. Function fitting. Fitting each data point. For cost function, want to maximize $\prod P(y_i|x_i) \equiv \max 1/n \sum \log p(y_i|x_i)$. *Equivalent to ERM.*
- Generative example: What is a good model of x given y. Fit blobs of data given their labels.
  - Let

$$p(x|y = C) = \mathcal{N}(\mu_c, \Lambda_c) \tag{25}$$

    where $\hat{\mu}_c = 1/n_c \sum_{i \in I_C} X_i$, and $\Lambda_c = 1/n_c \sum (x_i - \hat{\mu}_c)(x_i - \hat{\mu}_c)^T$. Only have one index i because it is a dxd sum of matrices.
  - Decision rule:

$$\arg\max_c -1/2(x - \hat{\mu}_c)^T \hat{\Lambda}^{-1}(x - \hat{\mu}_c) - 1/2 \log\det \hat{\Lambda}_c - \log \hat{\pi}_c \tag{26}$$

    where last two terms are apparently constant. First term is a quadratic. $Q_c(x)$ denotes the whole thing. Decision boundary is set $\{x : Q_{c=-1}(x) = Q_{c=1}(x)\}$.
  - Need to make sure $n >> d^2$ in order to avoid overfitting.
  - called **Quadratic Discriminant Analysis**.
- **Linear Discriminant Analysis**. Assume Λ_c same for every class. They all have the same covariance matrix.

- How to find Λ?

$$\hat{\Lambda}_c = \sum_C \frac{n_c}{n} \frac{1}{n} \sum_{i \in C} (x_i - \hat{\mu}_c)(x_i - \hat{\mu}_c)^T \tag{27}$$

$$= \frac{1}{n} \sum_i^n (x_i - \hat{\mu}_{y_i})(x_i - \hat{\mu}_{y_i})^T \tag{28}$$

- Extremely similar to QDA, have (sort of; don't rely on this)

$$\arg\max_c -(x - \hat{\mu}_c)^T \hat{\Lambda}^{-1} (x - \hat{\mu}_c) - 1/2 \log\det \hat{\Lambda} - \log \hat{\pi}_c \tag{29}$$

- end up with linear boundaries.

- Did something called **method of centroids**
- ppl like LDA bc fuck optimization

Machine Learning **Fall 2016**

Regression: September 27

Table of Contents Local *Scribe: Brandon McKinzie*

- Model $p(y|x) \sim \mathcal{N}(w^T x, \sigma^2)$, where $y = w^T x + \epsilon$. Epsilon is noise causing data points to fluctuate about hyperplane. Assume noise is gaussian with zero mean, some variance. Variance of y is the variance of the noise ϵ.
- Unknown we want to estimate: w. Estimate by using maximum likelihood. **Q:** says this maximizes $p(y|x)$. Figure out how this is same as maximizing $p(x|y)$...
- $P(data|\theta) = p(y_1, \dots, y_n | x_1, \dots, x_n, \theta) = \prod p(y_i|x_i, \theta)$.
- "Hit that bitch with a log so she split." - Jitendra
- Use matrices so you can express as

$$\sum (y_i - w^T x_i)^2 = ||y - Aw||^2 \tag{30}$$

  where A is typically denoted as the designed matrix X.
- Take gradient of loss like usual...

$$\nabla_W \mathcal{L} = -A^T y + A^T A w \tag{31}$$

  where, if we differentiate again, yields the hessian $H = A^T A$.
- Implicitly want $y \approx Aw$ here. Re-interpret A as a bunch of columns now (rather than a bunch of rows).

$$A = \begin{bmatrix} a_1 & \cdots & a_d \end{bmatrix} \tag{32}$$

  and so

$$||y - Aw||^2 = ||y - (w_1 a_1 + \cdots + w_d a_d)||^2 \tag{33}$$

- **column space** of A refers to this type of linear combination of columns a_i.
- References 3.2 figure in ESL. Error is vertical component of y in figure. This "error vector" is perpendicular to the subspace spanned by the x's.
- y - Aw is perpendicular to each and every column of A. $A^T(y - Aw) = \mathbf{0}$.

Machine Learning **Fall 2016**

Bias-Variance Tradeoff: October 4

Table of Contents Local *Scribe: Brandon McKinzie*

- Fitting the data to model is called **bias**. Bias summarizes the fact that the model is wrong, but we want to know how wrong.
- **Variance** is robustness to changes in data.
- Good model has both low bias and low variance.
- Example:
  - Sample one point $x \sim \mathcal{N}(\mu, \sigma^2 I_d)$.
  - What is most likely estimate for $\hat{\mu}$? Just x (only have one point).
  - Since, the expected value of x is (by definition) μ, we have that

$$\mathbb{E}[\hat{\mu} - \mu] = 0 \tag{34}$$

  - How about squared error

$$\begin{aligned} \mathbb{E}[||\hat{\mu} - \mu||^2] &= \mathbb{E}[(\hat{\mu} - \mu)^(\hat{\mu} - \mu)] && (35)\\ &= \mathbb{E}[Tr(x-\mu)(x-\mu)^T] && (36)\\ &= Tr(\Lambda) && (37) \end{aligned}$$

    which uses the **cyclic property of the trace**: if dot product is scalar, then it is equal to trace of outer product[6].
  - What is the trace of the covariance matrix Λ? Here (only) it is $d\sigma^2$.
  - What if I'm bored and I define $\hat{\mu} = \alpha x$, where $0 < \alpha < 1$? Then

$$\begin{aligned} \mathbb{E}[\hat{\mu}] &= \alpha\mu && (38)\\ \mathbb{E}[\hat{\mu} - \mu] &= (\alpha - 1)\mu && (39) \end{aligned}$$

    which isn't zero (woaAHhhh!)
  - Variance won't go down.

$$\mathbb{E}[||\hat{\mu} - \mu||^2] = \mathbb{E}[||\hat{\mu} - \mathbb{E}[\hat{\mu}] + \mathbb{E}[\hat{\mu} - \mu||^2]] \tag{40}$$

[6] Oh, it is just the fact that Tr(AB) = Tr(BA). Moving on...

- Me making sense of **Newton's Method** (as defined in this lecture):
  - Slow for high dimensional probs; Better than gradient descent though.
  - Gradient descent models func with first order taylor approx. Newton's method uses *second order*.

$$f(x) \approx f(x_k) + \nabla f(x_k)^T (x - x_k) + \frac{1}{2}(x - x_k)^T \nabla^2 f(x_k)(x - x_k) \tag{41}$$

    where grad-squared is the **Hessian**.
  - ***Actual derivation by Wikipedia***:

$$\text{Let } f(\alpha) = 0 \tag{42}$$
$$f(\alpha) = f(x_n) + f'(x_n)(\alpha - x_n) + R_1 \tag{43}$$
$$\text{where } R_1 = \frac{1}{2} f''(\xi_n)(\alpha - x_n)^2 \tag{44}$$
$$\frac{f(x_n)}{f'(x_n)} + (\alpha - x_n) = -\frac{f''(\xi_n)}{2f'(x_n)}(\alpha - x_n)^2 \tag{45}$$
$$\tag{46}$$

    and all the x_n represent the nth approximation of some root of $f(x)$.

- Oh I get it now:
  - **Gradient descent**: Find optimal w iteratively by assuming first-order taylor expansion of $\nabla J(w*)$:

$$\nabla J(w^*) \approx \nabla J(w_k) \tag{47}$$
$$\tag{48}$$

    where w_k is the current best guess for the minimum of J. If this gradient is zero, we are done. If it is not, then we continue to iterate closer and closer via the update

$$w_{k+1} = w_k - \eta \nabla J(w_k) \tag{49}$$

    until our first-order approximation (appears) valid.
  - **Newton's method** goes a step further and expands to second order:

$$\nabla J(w*) \approx \nabla J(w_k) + \nabla J(w_k)^2 (w^* - w_k) \tag{50}$$
$$= \nabla J(w_k) + \mathbf{H}(w^* - w_k) \tag{51}$$

where[7], implicit in all these optimization algorithms, is the hope that $w_{k+1} \approx w^*$, and so we can set this derivative to 0 to "solve" for $w_{k+1} = w^*$ as

$$(w_{k+1} - w_k) = -\mathbf{H}^{-1}\nabla J(w_k) \tag{52}$$

$$w_{k+1} = w_k - \mathbf{H}^{-1}\nabla J(w_k) \tag{53}$$

where equatoin 53 is **Newton's Update**. It is computationally better to compute e, where

$$\mathbf{H}e = -\nabla J(w_k) \longrightarrow e = -\mathbf{H}^{-1}\nabla J(w_k) \tag{54}$$

[7]Remember that we are dealing with matrices now, so keep the order of $\mathbf{H}$ before $(w - w_k)$ even if you don't like it.

Machine Learning **Fall 2016**

Regularization: October 6

Table of Contents Local *Scribe: Brandon McKinzie*

- bias = $\mathbb{E}[f(x) - y]$
- Risk = $\mathbb{E}[\text{loss}(f(x), y)]$
- Variance = $\mathbb{E}[(f(x) - \mathbb{E}[f(x)])^2]$
- Regularization: Minimize empirical loss + penalty term.

Machine Learning **Fall 2016**

Neural Networks: October 20

Table of Contents Local

Scribe: Brandon McKinzie

Basics/Terminology. Outputs can be computed for, say, a basic neuron to output 2 as $S_2 = \sum_i w_{2i} x_i$. We can also feed this through **activations functions** g such as the logistic or RELU. Why shouldn't we connect linear layers to linear layers? *Because that is equivalent to one linear layer.* If we want to stack (multilayer) need some nonlinearity. Want to find good weights so that output can perform classification/regression.

Learning and Training. Goal: Find w such that O_i is as close as possible to y_i (the labeled/desired output). Approach:

→ Define loss function $\mathcal{L}(w)$.
→ Compute $\nabla_w \mathcal{L}$.
→ Update $w_{new} \leftarrow w_{old} - \eta \nabla_w \mathcal{L}$.

and so training is all about *computing the gradient.* Amounts to computing partial derivatives like $\frac{\partial \mathcal{L}}{\partial w_{jk}}$. Approach for **training a 2-layer neural network**:

→ Compute $\nabla_w \mathcal{L}$ for all weights from input to hidden, hidden output.
→ Use SGD. Loss function **no longer convex** so can only find local minima.
→ Naive gradient computation is quadratic in num. weights. **Backpropagation** is a trick to compute it in linear time.

Computing gradients [for a two layer net]. The value of the ith output neuron can be computed as

$$O_i = g\Big(\sum_j W_{ij} \; g\big(\sum_k W_{jk} x_k \big) \Big) \tag{55}$$

where let's focus on the weight W_{12}. *Simple idea:*

- Consider some situation where we have value for output O_i as well as another value O_i' which is the same as O_i except one of the weights is slightly changed:

$$O_i = g(\ldots, w_{jk}, \ldots, x) \tag{56}$$
$$O_i' = g(\ldots, w_{jk} + \Delta w_{jk}, \ldots, x) \tag{57}$$

- Then we can compute numerical approx to derivative for one of the weights:

$$\frac{\mathcal{L}(O_i') - \mathcal{L}(O_i)}{\Delta w_{jk}} \tag{58}$$

  a process typically called the **forward pass**[8] This is $\mathcal{O}(n)$ if there are n weights in the network. But since this is just the derivative for one of the weights, the total cost over all weights is $\mathcal{O}(n^2)$.

- This is why we need backprop: to lower complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$.

Backpropagation. Big picture: A lot of these computations [gradients] seem to be shared. Want to find some way of avoiding computing quantities more than once.

→ **Idea:** Want to compute some quantity δ^i at output layer for each of the i output neurons. Then, find the δ^{i-1} for the layer below, repeat until reach *back* to input layer [hence name backprop]. Key idea is the **chain rule**.

→ Notation:[9]

$$x_j^{(l)} = g\left(\sum_i w_{ij}^{(l)} x_i^{(l-1)}\right) \equiv g\left(S_j^{(l)}\right)$$

where now w_{ij} is *from* i *to* j. **We will also denote** e **for 'error'.**

→ Define partial derivative of error with respect to the linear combination input to neuron j as

$$\delta_j^{(l)} \triangleq \frac{\partial e}{\partial S_j^{(l)}} \tag{59}$$

which carry the information we want about the partial derivatives along the way.

→ Consider simple case of

$$x_i^{(l-1)} \to w_{ij}^{(l)} \to x_j^{(l)}$$

and we want to calculate

$$\frac{\partial e}{\partial w_{ij}^{(l)}} = \frac{\partial e}{\partial S_j^{(l)}} \frac{\partial S_j^{(l)}}{\partial w_{ij}^{(l)}} \tag{60}$$

$$= \delta_j^{(l)} \frac{\partial S_j^{(l)}}{\partial w_{ij}^{(l)}} \tag{61}$$

[8] Not sure why he says this. See pg 396 of ESL. Forward Pass: the current weights are fixed and the predicted values $\hat{f}_k(x_i)$.

[9] Note: he really screws this up.

→ "Inductive step" for calculating δ with chain rule [for **regression problem using squared error loss of** $e = \frac{1}{2}\Big(g(S_i^{(l)}) - y\Big)^2$ **corresponds to a given example**]:

$$\delta_i^{(l)} = \frac{\partial e}{\partial S_i^{(l)}} \tag{62}$$

$$= \frac{1}{2}\Bigg[2\ \Big(g(S_i^{(l)}) - y\Big)\ g'(S_i^{(l)})\Bigg] \tag{63}$$

Don't confuse the above expression for sigmoid deriv. It is not assuming anything about the functional form of g.

Machine Learning **Fall 2016**

Neural Networks II: October 25

Table of Contents Local *Scribe: Brandon McKinzie*

Backprop Review. Cross-entropy loss derivation uses the following. Note that we are defining all $y_i = 0$ or $y_i = 1$.

$$O_i^{y_i}(1-O_i)^{1-y_i} \quad \rightarrow \quad y_i \ln O_i + (1-y_i)\ln(1-O_i) \tag{64}$$

where the expression on the RHS is the log (likelihood) of the LHS. We want to take partial derivatives of loss function with respect to weights. The δ terms represent layer-specific derivatives of error with respect to the S values (the summed input). See previous lecture note for more details on this. Note that, in order to get the values of the error in the first place, need to first perform the **forward pass**.

Clarifying the notation. In the last lecture, we barely scratched the surface of actually calculating $\delta_i^{(l-1)}$, the partial of the error with respect to $S_i^{(l-1)}$. Recall that the subscript on $S_i^{(l-1)}$ means the weighted sum *into* the ith neuron at layer. Specifically

$$S_j^{(l-1)} = \sum_i w_{ij}^{(l)} x_i^{(l-2)} \quad \rightarrow \quad x_j^{(l-1)} \tag{65}$$

Calculating the δ terms. **Setup**: Only consider the following portion of the network: A summation value $S_i^{(l-1)}$ is fed into a single neuron $x_i^{(l-1)}$ at the $l-1$ layer. From this neuron, $g(S_i^{(l-1)})$ is fed to the neurons at the layer above (l) by connection weights w. We calculate the partial derivative of the error *corresponding to these particular weights* with respect to the summation fed to $x_i^{(l-1)}$ as

$$\delta_i^{(l-1)} = \frac{\partial err(w)}{\partial S_i^{(l-1)}} \tag{66}$$

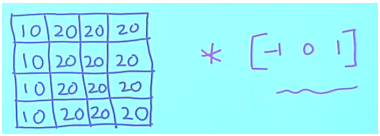
$$= \sum_j \frac{\partial err(w)}{\partial S_j^{(l)}} \frac{\partial S_j^{(l)}}{\partial x_i^{(l-1)}} \frac{\partial x_i^{(l-1)}}{\partial S_i^{(l-1)}} \tag{67}$$

$$= \sum_j \delta_j^{(l)} w_{ij}^{(l)} g'(S_i^{(l-1)}) \tag{68}$$

where we've already calculated all δ value in the layers above (i.e. we are somewhere along the backward pass).

[Inspiration for] Convolutional Neural Networks. [1 hr into lec]. Reviews biology of brain/neuron/eye. Rods and cones are the eye's pixels. Think of as 2D sheet of inputs. Such sheets can be thought of as 1D layers. Bipolar cell gets direct input (center input) from two photo-receptors, and gets indirect input (surround input) from horizontal cell. "Disc where you're getting indirect input from horizontal cell." Weights between neurons can be positive (excitatory) or negative (inhibitory). Assume center input is excitatory, surround input in inhibitory.

- Very small spot of light means neuron fires, as you increase size of spot, inhibition from surround cells kick in, and its output is diminished. Uses example of ON/OFF cells in retinal ganglia. Neurons can only individually communicate positive values, but multiple neurons can "encode" negative values.
- **Receptive Fields**. The receptive field of a receptor is simply the area of the visual field from which light strikes that receptor. For any other cell in the visual system, the receptive field is determined by which receptors connect to the cell in question.
- Relation to **Convolution**. Consider convolving an image with a filter.



  Each output unit gets the weighted sum of image pixels. The [-1, 0, 1] is a "weighting mask."

Machine Learning **Fall 2016**

Convolutional Neural Networks: October 27

Table of Contents Local

Scribe: Brandon McKinzie

Shewchuck Notes

Contents

Shewchuck Notes **Fall 2016**

Perceptron Learning:

Table of Contents Local *Scribe: Brandon McKinzie*

Perceptron Algorithm

- Consider n sample points $X_1, \ldots, X_n$.
- For each sample point, let

$$y_i = \begin{cases} 1 & X_i \in \text{class C} \\ -1 & X_i \notin \text{class C} \end{cases}$$

- **Goal**: Find weights w that satisfy the constraint

$$y_i X_i \cdot w \geq 0 \tag{69}$$

- In order to minimize the number of constraint violations, need a way to quantify how "good" we are doing. Do this with the **loss function**

$$L(z,\ y_i) = \begin{cases} 0 & y_i z \geq 0 \\ -y_i z & \text{otherwise} \end{cases} \tag{70}$$

  Notice that this can only be ≥ 0 by definition. The larger L is, the worse you are as a human being.

- The **Risk/Objective/Cost** function is a sum total of your losses.

$$R(w) = \sum_{i=1}^{n} L(X_i \cdot w, y_i) = \sum_{i \in V} (-y_i\ X_i \cdot w) \tag{71}$$

  where $(\forall i \in V)(y_i X_i \cdot w < 0)$.

- **Goal**: Find w that minimizes $R(w)$.

Hyperplanes with Perceptron

- Notice the different between the two (purple) Goals stated in the previous subsection. We went from constraining w to certain **hyperplanes** in x-space ($y_i X_i \cdot w \geq 0$) to constraining w to certain **points** in w-space ($\min_w R(w)$).

- Figure **??** illustrates how the data points constrain the possible values for w in our optimization problem. For each sample point x, the constraints can be stated as
  - x in the "positive" class $\Rightarrow$ x and w must be on the **same** side of the hyperplane that x transforms into[10].
  - x in the "negative" class $\Rightarrow$ x and w must be on the **opposite** side of x's hyperplane.

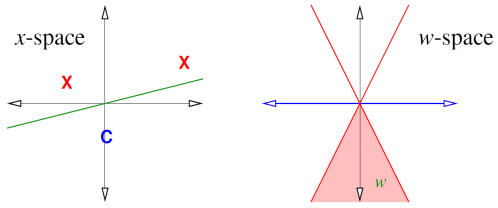


Figure 1: Illustration of how three sample points in x-space (left) can constrain the possible values for w in w space (right).

Algorithm: Gradient Descent

- GD on our risk function R is an example of an **optimization algorithm**. We want to *minimize* our risk, so we take successive steps in the *opposite* direction of $\nabla R(w)$.[11]

$$\nabla R(w) = \nabla \sum_{i \in V} (-y_i \; X_i \cdot w) \tag{72}$$

$$= -\sum_{i \in V} (y_i \; X_i) \tag{73}$$

- **Algorithm:**
  - $w \leftarrow$ arbitrary nonzero (e.g. any $y_i X_i$)
  - while $R(w) > 0$
    - $V \leftarrow$ all i for which $y_i X_i \cdot w < 0$
    - $w \leftarrow w + \epsilon \sum_{i \in V} (y_i \; X_i)$

  where ϵ is the **learning rate/step size**. Each step is $O(nd)$ time.

[10] x transforms into a hyperplane in w space defined as all w that satisfy $x \cdot w = 0$.
[11] Recall that the gradient points in direction of steepest ascent.

Algorithm: Stochastic GD

- Procedure is simply GD on one data point only per step, i.e. no summation symbol. Called the **perceptron algorithm**.
- **Algorithm:**
  - while some $y_i X_i \cdot w < 0$
    - $w \leftarrow w + \epsilon y_i X_i$
  - return w.
- **Perceptron Convergence:** If data is linearly separable, perfect linear classifier will be found in at most $O(R^2/\gamma^2)$ iterations, where
  - $R = \max_i |X_i|$ is radius of the data
  - γ is the maximum margin.

Maximum Margin Classifiers

- **Margin**: (of a linear classifier) the distance from the decision boundary to the nearest sample point.
- **Goal**: Make the margin as large as possible.
  - Recall that the margin is defined as $|\tau_{min}|$, the magnitude of the smallest euclidean distance from a sample point to the decision boundary, where for some x_i,

    $$\tau_i = \frac{|f(x_i)|}{||w||}$$

    and our goal is to maximize the value of the smallest τ in the dataset.
  - Enforce the (seemingly arbitrary?) constraints that $|f(x_i)| \geq 1$, or equivalently

    $$y_i(w \cdot x_i + \alpha) \geq 1 \tag{74}$$

    which can also be stated as requiring all $\tau_i \geq 1/||w||$.
  - **Optimize:** Find w and α that minimize $||w||^2$, subject to $y_i(w \cdot x_i + \alpha) \geq 1$ for all $i \in [1, n]$. "Called a **quadratic program** in d+1 dimensions and n constraints. It has **one unique solution.**"
- The solution is a **maximum margin classifier** aka a **hard SVM**.

Shewchuck Chapter 4 **Fall 2016**

Soft-Margin SVMs:

Table of Contents Local *Scribe: Brandon McKinzie*

- Hard-margin SVMs fail if not linearly separable.
- **Idea:** Allow some points to violate the margin, with **slack variables** ξ

$$y_i(X_i \cdot w + \alpha) \geq 1 - \xi_i \tag{75}$$

  where $\xi_i \geq 0$. Note that each sample point is assigned a value of ξ_i, which is only nonzero iff x_i violates the margin.

- To prevent abuse of slack, add a **loss term** to our objective function[12].
  - Find w, α, and ξ_i that minimize our objective function,

$$|w|^2 + C\sum_{i=1}^{n} \xi_i \tag{76}$$

    subject to

$$y_i(X_i \cdot w + \alpha) \geq 1 - \xi_i \quad \text{for all } i \in [1, n] \tag{77}$$
$$\xi_i \geq 0 \quad \text{for all } i \in [1, n] \tag{78}$$

    a quadratic program in d + n + 1 dimensions and 2n constraints. The relative size of C, the **regularization hyperparameter** determines whether you are more concerned with getting a large margin (small C) or keeping the slack variables as small as possible (large C).

[12] Before this, looks like our objective function was just $|w|^2$ since that is what we wanted to minimize (subject to constraints).

Shewchuck Chapter 6 **Fall 2016**

Decision Theory:

Table of Contents Local *Scribe: Brandon McKinzie*

- For when "a sample point in feature space doesn't have just one class". Solution is to classify with probabilities.
- **Important terminology:**
  - **Loss Function** $L(z, y)$: Specifies badness of classifying as z when the true class is y. Can be **asymmetrical**. We are typically used to the *0-1 loss function* which is symmetric: 1 if incorrect, 0 if correct.
  - **Decision rule (classifier)** $r : \mathbb{R}^d \to \pm 1$. Maps feature vector x to a class (1 if in class, -1 if not in class for binary case).
  - **Risk**: Expected loss over *all* values of x, y:

$$R(r) = \mathbb{E}\Big[L(r(X), Y)\Big] \tag{79}$$

$$= \sum_y P(Y = y) \sum_x P(X = x|Y = y) L(r(x), y) \tag{80}$$

    In ESL Chapter 2.4, this is denoted as the **expected prediction error**.
  - **Bayes decision rule/classifier** $r*$: Defined as the decision rule $r = r*$ that minimizes $R(r)$. If we assume $L(z, y) = 0$ when $z = y$, then

$$r^*(x) = \begin{cases} 1 & L(-1, 1)P(1|x) > L(1, 1)P(-1|x) \\ -1 & \text{otherwise} \end{cases} \tag{81}$$

    which has *optimal risk*, also called the **Bayes risk** $R(r^*)$.
- Three ways to build classifiers:
  - **Generative models (LDA)**: Assume sample points come frome class-conditioned probability distributions $P(x|c)$, different for each class. Guess the form of these dists. For each class C, fit (guessed) distributions to points labeled as class C. Also need to estimate (basically make up?) $P(C)$. Use bayes rule and classify on $\max_C P(Y = C|X = x)$. **Advantage**: Can diagnose outliers (small P(x)). Can know the probability that prediction is wrong. **Real definition**: A full probabilistic model of all variables.

- **Discriminative models**. Model $P(Y|X)$ directly. (I guess this means don't bother with modelling all the other stuff like X—Y, just go for it bruh.) **Advantage:** Can know probability of prediction being wrong. **Real definition**: A model only for the target variables.
- **Decision boundary finding**: e.g. SVMs. Model $r(x)$ directly. **Advantage**: Easier; always works if linearly separable; don't have to guess explicit distributions.

Shewchuck Chapter 7 **Fall 2016**

Gaussian Discriminant Analysis:

Table of Contents Local *Scribe: Brandon McKinzie*

- **Fundamental assumption:** Each class C comes from a normal distribution.
- For a given x, want to maximize $P(X = x|Y = C)\pi_C$, where π_C prior probability of class c. Easier to maximize $ln(z)$ since increases monotonically for $z > 0$. The following gives the "quadratic in x" function $Q_C(x)$,

$$Q_C(x) = \ln\left((\sqrt{2\pi})^d P(x)\pi_C\right) \tag{82}$$

$$= -\frac{|x-\mu_C|^2}{2\sigma_C^2} - d\ln\sigma_C + \ln\pi_C \tag{83}$$

  where $P(x)$, a normal distribution, is what we use to estimate the class conditional $P(x|C)$.
- The Bayes decision rule r^* returns the class C that maximizes $Q_C(x)$ above.

Quadratic Discriminant Analysis (QDA)

- Suppose only 2 classes, C and D. Then

$$r^*(x) = \begin{cases} C & Q_C(x) - Q_D(x) > 0 \\ D & \text{otherwise} \end{cases} \tag{84}$$

  which is quadratic in x. The Baye's Decision Boundary (BDB) is the solution of $Q_C(x) - Q_D(x) = 0$.
- In 1D, BDB may have 1 or 2 points (solution to quadratic equation)
- In 2D, BDB is a ***quadric*** (e.g. for d=2, conic section).
- In 2-class problems, naturally leads to **logistic/sigmoid** function for determining $P(Y|X)$.

Newton's Method

- Iterative optimization for some smooth function $J(w)$.
- Can Taylor expand gradient about v:

$$\nabla J(w) = \nabla J(v) + (w - v)\nabla^2 J(v) + \mathcal{O}(|w - v|^2) \tag{85}$$

  where $\nabla^2 J(v)$ is the **Hessian matrix** of $J(w)$ at v, which I'll denote $\mathbf{H}$.

- Find critical point w where $\nabla J(w) = 0$:

$$w = v - H^{-1}\nabla J(v) \tag{86}$$

- Shewchuck defines **Newton's method** algorithm as:

  1. Initialize w.
  2. until convergence do:

     $e := solve\_linear\_system\Big(\mathbf{H}e = -\nabla J(w)\Big)$.

     $w := w + e$.

  where starting w must be "close enough" to desired solution.

Justifications & Bias-Variance (12)

- Overview: Describes models, how they lead to optimization problems, and how they contribute to underfitting/overfitting.
- Typical model of reality:

$$y_i = f(X_i) + \epsilon_i \tag{87}$$

  where $\epsilon_i \sim D'$ has mean zero.

- Goal of regression: find h that estimates f.

Fall 2016

Probability Review:

Table of Contents Local

Scribe: Brandon McKinzie

Notation:

- **Sample Space** Ω: Set of all outcomes of a random experiment. For six-sided die, $\Omega = \{1, \dots, 6\}$.
- **Event Space** $\mathcal{F}$: Set whose *elements* are *subsets* of Ω. Appears that $\mathcal{F}$ is required to be complete in a certain sense, i.e. that it should contain *all* possible events (combinations of possible individual outcomes).
- **Probability measure**: Function $P : \mathcal{F} \to \mathbb{R}$. Intuitively, it tells you what fraction of the total space of possibilities that $\mathcal{F}$ is in, where if $\mathcal{F}$ *is* the full space, $P(F) = P(\Omega) = 1$. Also required: $P(A) \geq 0 \;\; \forall A \in \mathcal{F}$.

Random Variables:[13]

- Consider experiment: Flip 10 coins. An example element of Ω would be of the form

$$\omega_0 = (H, H, T, H, T, H, H, T, H, T) \in \Omega \tag{88}$$

  which is typically a quantity too specific for us to really care about. Instead, we prefer real-valued *functions* of outcomes, known as **random variables**.

- R.V. X is defined as a function $X : \Omega \to \mathbb{R}$. They are denoted as $X(\omega)$, or simply X if ω dependence is obvious.
- Using our definition of the probability measure, we define the probability that $X = k$ as the probability measure over the space containing all outcomes ω where $X(\omega) = k$.[14]

$$P(X = k) := P(\{\omega : X(\omega) = k\}) \tag{89}$$

- **Cumulative Distribution Function:** $F_X : \mathbb{R} \to [0, 1]$ defined as[15]

$$F_X(x) \triangleq P(X \leq x) \tag{90}$$

[13] TIL I had no idea what a random variable really was.
[14] Oh my god yes, this is what I came here for.
[15] The symbol $\triangleq$ means equal by definition (hnnggg). In continuous case, $F_X(x) = \int_{-\infty}^{x} p_X(u)du$.

- **Probability Mass Function**: When X is a *discrete* RV, it is simpler to represent the probability measure by directly saying the probability of each possible value X can assume. It is a function $p_X : \Omega \to \mathbb{R}$ such that

$$p_X(x) \triangleq P(X = x) \tag{91}$$

- **Probability Density Function**: The derivative of the CDF.

$$f_X(x) \triangleq \frac{\mathrm{d}F_X(x)}{\mathrm{d}x} \tag{92}$$

$$P(x \leq X \leq x + \delta x) \approx f_x(x)\delta x \tag{93}$$

EXPECTATION VALUE

- Discrete X: (PMF $p_X(x)$) Can either take expectations of X (the mean) or of some function $g(X) : \mathbb{R} \to \mathbb{R}$, also a random variable.

$$\mathbb{E}[g(X)] \triangleq \sum_{x \in Val(X)} g(x)p_X(x) \tag{94}$$

$$\mathbb{E}[X] \triangleq \sum_{x \in Val(X)} xp_X(x) \tag{95}$$

- Continuous X: (PDF $f_X(x)$), then

$$\mathbb{E}[g(X)] \triangleq \int_{-\infty}^{\infty} g(x)f_X(x)\mathrm{d}x \tag{96}$$

- **Properties**:

$$\mathbb{E}[a] = a \quad \forall a \in \mathbb{R} \tag{97}$$

$$\mathbb{E}[a\ f(X)] = a\mathbb{E}[f(X)] \tag{98}$$

$$\mathbb{E}[f(X) + g(X)] = \mathbb{E}[f(X)] + \mathbb{E}[g(X)] \tag{99}$$

$$\mathbb{E}[bool(X == k)] = P(X = k) \tag{100}$$

Variance: Measure of how concentrated the dist of a RV is around its mean.

$$Var[X] \triangleq \mathbb{E}[(X - \mathbb{E}[X])^2] \tag{101}$$

$$= \mathbb{E}[X^2] - \mathbb{E}[X]^2 \tag{102}$$

with properties:

$$Var[a] = 0 \quad \forall a \in \mathbb{R} \tag{103}$$

$$\Delta[af(X)] = a^2 Var[f(X)] \tag{104}$$

Distribution	PDF or PMF	Mean	Variance
$Bernoulli(p)$	$\begin{cases} p, & \text{if } x = 1 \\ 1-p, & \text{if } x = 0. \end{cases}$	p	$p(1-p)$
$Binomial(n,p)$	$\binom{n}{k} p^k (1-p)^{n-k}$ for $0 \leq k \leq n$	np	npq
$Geometric(p)$	$p(1-p)^{k-1}$ for $k = 1, 2, \ldots$	$\frac{1}{p}$	$\frac{1-p}{p^2}$
$Poisson(\lambda)$	$e^{-\lambda}\lambda^x/x!$ for $k = 1, 2, \ldots$	λ	λ
$Uniform(a,b)$	$\frac{1}{b-a}$ $\forall x \in (a,b)$	$\frac{a+b}{2}$	$\frac{(b-a)^2}{12}$
$Gaussian(\mu, \sigma^2)$	$\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$	μ	σ^2
$Exponential(\lambda)$	$\lambda e^{-\lambda x}$ $x \geq 0, \lambda > 0$	$\frac{1}{\lambda}$	$\frac{1}{\lambda^2}$

Covariance

- Recognize that the covariance of two random variables X and Y can be described as a *function* $g : \mathbb{R}^2 \to \mathbb{R}$. Below we define the expectation value for some multivariable function[16], and then we can define the covariance as a particular example.

$$\mathbb{E}[[]g(X,Y)] \triangleq \sum_{x \in Val(X)} \sum_{y \in Val(Y)} g(x,y) p_{XY}(x,y) \tag{105}$$

$$Cov[X,Y] = \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])] \tag{106}$$

- Properties:

$$Cov[X,Y] = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y] \tag{107}$$

$$Var[X+Y] = Var[X] + Var[Y] + 2Cov[X,Y] \tag{108}$$

Random Vectors

- Suppose we have n random variables $X_i = X_i(\omega)$ all over the same general sample space Ω. Convenient to put them into a **random vector** X, defined as $X : \Omega \to \mathbb{R}^n$ and with $X = [X_1 X_2 \ldots X_n]^T$.

- Let g be some function $g : \mathbb{R}^n \to \mathbb{R}^m$. We can define expectations with notation laid out below.

$$g(X) = \begin{bmatrix} g_1(X) \\ g_2(X) \\ \vdots \\ g_m(X) \end{bmatrix} \qquad \mathbb{E}[g(X)] = \begin{bmatrix} \mathbb{E}[g_1(X)] \\ \mathbb{E}[g_2(X)] \\ \vdots \\ \mathbb{E}[g_m(X)] \end{bmatrix} \tag{109}$$

$$\mathbb{E}[g_i(X)] = \int_{\mathbb{R}^n} g(x_1, \ldots, x_n) f_{X_1,\ldots,X_n} dx_1 \ldots dx_n \tag{110}$$

- For a given $X : \Omega \to \mathbb{R}^n$, its **covariance matrix** Σ is the $n \times n$ matrix with $\Sigma_{ij} = Cov[X_i, X_j]$. Also,

[16] Discrete case shown only. Should be obvious how it would look for continuous.

$$\Sigma = \mathbb{E}[XX^T] - \mathbb{E}[X]\mathbb{E}[X]^T \tag{111}$$
$$= \mathbb{E}[(X - \mathbb{E}[X])(X - \mathbb{E}[X])^T] \tag{112}$$

and it satisfies: (1) $\Sigma \succeq 0$ (pos semi-def), (2) Σ is symmetric.

Fall 2016

Probability Review:

Table of Contents Local *Scribe: Brandon McKinzie*

The eigenvalues of a matrix are the zeros of its **characteristic polynomial**, defined as $f(\lambda) = \det(A - \lambda I)$. A vector $v \neq 0$ is an **eigenvector** iff $v \in Null(A - \lambda I)$.

Regardless of offset of plane, the normal vector to ax + by + cz = d is $w = (a, b, c)$. For any point A not on the plane, closest point B to P where B *is* on the plane, is determined by the value of α that solves

$$(A - \alpha(a, b, c)) \cdot (a, b, c) = d \tag{113}$$

since, given α satisfies the equation, $B = (A - \alpha(a, b, c))$ is a point on the plane, constructed by following the direction of w "backwards" from A.

Direct comparison between MLE and **MAP**:[17]

$$\theta_{MLE} = \arg\max_\theta \sum_i \log\left(p_X(x|\theta)\right) \qquad \begin{aligned} p(\theta|x) &\propto p_X(x|\theta)p(\theta) \\ \theta_{MAP} &= \arg\max_\theta \sum_i \log\left(p_X(x|\theta)p(\theta)\right) \\ &= \arg\max_\theta \{\log\left[p_X(x|\theta)\right] + p(\theta)\} \end{aligned} \tag{114}$$

Support Vector Machines

[18]

- Note that (logistic regression)

$$g(\theta^T) \geq 0.5 \iff \theta^T x \geq 0 \tag{115}$$

- Switch to perceptron algorithm where

$$h_{w,b}(x) = g(w^T x + b) = \begin{cases} 1 & w^T x + b \geq 0 \\ -1 & \text{otherwise} \end{cases} \tag{116}$$

- Given a training example $(x^{(i)}, y^{(i)})$, define the **functional margin** of (w, b) w.r.t the training example as

[17] MAP also known as Bayesian Density Estimation

[18] Based off Andrew Ng's CS 229 Notes.

$$\hat{\gamma}^{(i)} = y^{(i)}(w^T x^{(i)} + b) \tag{117}$$

where $\hat{\gamma}^{(i)} > 0$ means prediction is correct.[19] We can also define with respect to $S = \{(x^{(i)}, y^{(i)}) : i = 1, \ldots, m\}$ to be the *smallest* of the individual functional margins:

$$\hat{\gamma} = \min_{i=1,\ldots,m} \hat{\gamma}^{(i)} \tag{118}$$

- Now we move to **geometric margins**. First, consider figure 2[20]

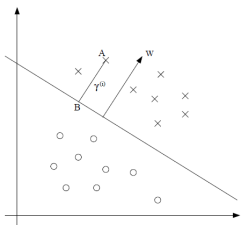


Figure 2: Decision boundary

- If we consider A as the ith data point, what is value of $\gamma^{(i)}$? The point B that is closest to A on the plane is given by $A - \tau \cdot w/||w||$ where τ is the distance $|AB|$ that we want to solve for. Since B is on the plane, we can solve for τ via

$$0 = w^T \left(x^{(i)} - \tau \frac{w}{||w||} \right) + b \tag{119}$$

$$\tau = \frac{w^T x^{(i)} + b}{||w||} \tag{120}$$

which leads to the general definition for the **geometric margin**, denoted *without the hat* $\gamma^{(i)}$ as

$$\gamma^{(i)} = y^{(i)} \left[\left(\frac{w}{||w||} \right)^T x^{(i)} + \frac{b}{||w||} \right] \tag{121}$$

[19] Possible insight relating to regularization: Notice how perceptron classification $g(x)$ only depends on the sign of it's argument, and not on the *magnitude*. However, performing $x \to 2x$ causes our functional margin to double $\hat{\gamma}^{(i)} \to 2\hat{\gamma}^{(i)}$ and so it seems "*we can make the functional margin arbitrarily large without really changing anything meaningful.* This leads to, perhaps, defining a normalization condition that $||w||_2 = 1$. Hmmm...

[20] Alright retard, time to settle this once and for all. The plane containing point P_0 and the vector $\boldsymbol{n} = (a, b, c)$ consists of all points P with corresponding position vector $\boldsymbol{r}$ such that the vector drawn from P_0 to P is perpendicular to $\boldsymbol{n}$, i.e. the plane contains all points $\boldsymbol{r}$ such that $\boldsymbol{n} \cdot (\boldsymbol{r} - \boldsymbol{r_0}) = 0$

where clearly if $||w|| = 1$ is the same as the functional margin. Also can define the geometric over the whole training set similarly as was done for the functional margin.

- ***Optimizing (maximizing) the margin.***
  - Pose the following optimization problem

$$\max_{\gamma,w,b} \frac{\hat{\gamma}}{||w||} \quad \textit{S.T.} \tag{122}$$

$$y^{(i)}\left(w^T x^{(i)} + b\right) \geq \hat{\gamma} \tag{123}$$

  - Due to reasons primarily regarding how computing $||w||$ is non-convex/hard, we translate the problem as follows: (1) impose (on the *functional margin*[21]) constraint that[22], $\hat{\gamma} = 1$ which we can always satisfy with some scaling of w and b; (2) Instead of maximizing $1/||w||$, minimize $||w||^2$.

$$\min_{\gamma,w,b} \frac{1}{2}||w||^2 \quad \textit{S.T.} \tag{124}$$

$$y^{(i)}\left(w^T x^{(i)} + b\right) \geq 1 \tag{125}$$

    which gives us the **optimal margin classifier**.
  - Lot of subtleties: For SVM (at least in this class) we want maximize the margin, which we **define** to be $2/||w||$. Note that this is *not* a fixed scalar value, it changes as $||w||$ changes! The **support vectors** are any points $x^{(i)}$ such that $y^{(i)}(w^T x^{(i)} + b) = 1$.

PROOF THAT W IS ORTHOGONAL TO THE HYPERPLANE

- Claim: If w is a vector that classifies according to perceptron algorithm (equation 116), then w is orthogonal to the separating hyperplane.
- Proof: We proceed, using only the definition of a plane, by finding the plane that w is orthogonal to, and show that this plane must be the separating hyperplane.
- If we plug in w to the *point-normal form* of the equation of a plane, **defined** as the plane containing all points $\boldsymbol{r} = (x, y, z)$ such that w is orthogonal to the **PLANE**[23]

$$w_x x + w_y y + w_z z + d = 0 \tag{126}$$

$$\boldsymbol{w}^T \boldsymbol{r} + d = 0 \tag{127}$$

$$\tag{128}$$

[21] Recall that the functional margin alone does NOT tell you a distance.

[22] Also remember that $\hat{\gamma}$ is over the WHOLE training set, and evaluates to the smallest $\hat{\gamma}^{(i)}$

[23] NOTE HOW I SAID PLANE AND NOT ANY VECTOR POINTING TO SOME POINT IN THE PLANE

where, denoting $\boldsymbol{r}_0 = (x_0, y_0, z_0)$ as the vector pointing to some arbitrary point P_0 in the plane,

$$d = -(w_x x_0 + w_y y_0 + w_z z_0) \tag{129}$$
$$= -(\boldsymbol{w}^T \boldsymbol{r}_0) \tag{130}$$

which means that

$$0 = \boldsymbol{w}^T \boldsymbol{r} + d \tag{131}$$
$$= \boldsymbol{w}^T \boldsymbol{r} - (\boldsymbol{w}^T \boldsymbol{r}_0) \tag{132}$$
$$= \boldsymbol{w}^T (\boldsymbol{r} - \boldsymbol{r}_0) \tag{133}$$

QED

Spring 2016 Midterm

- Hard-margin SVM and perceptron will not return a classifier if data not linearly separable.
- Soft-margin SVM uses $y_i(X_i \cdot w + \alpha) \geq 1 - \xi_i$, so $\xi_i \neq 0$ for both (1) misclassified samples and (2) all samples inside the margin.
- Large value of C in (Soft-margin SVM) $|w|^2 + C \sum \xi_i$ is prone to **overfitting training** data. Interp: Large C means we want most $\xi_i \to 0$ or small, and therefore the **decision boundary will be sinuous**, something we currently don't know how to do.
- **Bayes classifier** classifies to the most probable class, using the conditional (discrete) distribution $P(G|X)$.

$$\hat{G}(x) = \arg\max_{g \in G} Pr(g|X = x) \tag{134}$$

- $\Sigma^{1/2} = U \Lambda^{1/2} U^T$.

Multivariate Gaussians

- The covariance matrix $\Sigma \in \mathbf{S}_+^n +$, the space of all symmetric, positive definite nxn matrices.
- Due to this, and since the inverse of any pos. def matrix is also pos. def, we can say that, for all $x \neq \mu$:

$$-\frac{1}{2}(x - \mu)^T \Sigma^{-1} (x - \mu) < 0 \tag{135}$$

- **Theorem:** *For any random vector X with mean μ and covariance matrix Σ* [24]

[24] Also in Probability Review

$$\Sigma = \mathbb{E}[(X-\mu)(X-\mu)^T] = \mathbb{E}[XX^T] - \mu\mu^T \tag{136}$$

- **Theorem:** *The covariance matrix Σ of any random vector X is symmetric positive semidefinite.*
  - In the particular case of Gaussians, which require existence of Σ^{-1}, we also have that Σ is then full rank. "Since any full rank symmetric positive semidefinite matrix is necessarily symmetric positive definite, it follows that Σ must be **symmetric positive definite**.
- The DIAGONAL COVARIANCE MATRIX case. An n-dimensional Gaussian with mean $\mu \in \mathbb{R}^n$ and diagonal $\Sigma = \text{diag}(\sigma_1^2,\ \sigma_2^2, \ldots, \sigma_n^2)$ is the same as n independent Gaussian random variables with mean μ_i and σ_i^2, respectively. (i.e. $P(X) = P(x_1) \cdot P(x_2) \cdots P(x_n)$ where each $P(x_i)$ is a univariate Gaussian PDF.
- **ISOCONTOURS**. General intuitions listed below[25]
  - For random vector $X \in \mathbb{R}^2$ with $\mu \in \mathbb{R}^2$, isocontours are **ellipses** centered on (μ_1, μ_2).
  - *If* Σ diagonal, then principal axes lie along x and y axis. Otherwise, in more general case, they are along the covariance eigenvects. (right?)
- **Theorem:** *Let $X \sim \mathcal{N}(\mu, \Sigma)$ for some $\mu \in \mathbb{R}^n$ and $\Sigma \in \mathbf{S}^n_{++}$. Then there exists a matrix $B \in \mathbb{R}^{nxn}$ such that if we define $Z = B^{-1}(X-\mu)$, then $Z \sim \mathcal{N}(0, I)$.*

Misc. Facts

- The sum of absolute residuals is less sensitive to outliers than the residual sum of squares. [Todo: study the flaws of least-squares regression.]
- In LDA, the discriminant functions $\delta_k(x)$ are an *equivalent* description of the decision rule, classifying as $G(x) = \arg\max_k \delta_k(x)$, where (for LDA),

$$\delta_k(x) = x^T\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + \log \pi_k \tag{137}$$

- Large value of C in soft-margin SVM objective function $|w|^2 + C\sum\xi_i$ is likely to **overfit** training data. This is because it will drive the ξ_i very low/zero, which means *it constructed a (likely nonlinear) decision boundary such that most points were either on or outside the margin.* The key here is that changing the ξ_i associated with points doesn't mean you're ignoring them or something, it means you are manipulating the decision boundary to more closely resemble your training distribution.

[25] Disclaimer: The following were based on an example with $n = 2$ and diagonal Σ. I've done my best to generalize the arguments they made here. I'm like, pretty sure I'm right, but...you know how things can go.

- Can't believe this is necessary, but remember that the sum in the following denominator is over y (not x):

$$P(Y = y_i|X = x_i) = \frac{f_i(x_i)\pi_i}{\sum_{y_j \in Y} f_j(x_i)\pi_j} \tag{138}$$

  If binary class classification, decision boundary is at $x = x*$ where $P(Y = 1|x*) = P(Y = 0|x*) = \frac{1}{2}$. If logistic regression, this occurs when the argument $h(x*)$ to the exponential in denominator is $\exp(h(x*)) = \exp(0) = 1$. So, to find the values of x along decision boundary, in this particular case, you'd solve $h(x) = 0$.

- **[DIS3.2]** Ok. First, never forget that

$$1 = \int_{x \in X|Y_i} f_{X|Y=Y_i}(x)dx \tag{139}$$

  and, therefore, if you're told that x_n sampled

  > iid and uniformly at random from 2 equiprobable classes, a disk of radius 1 (Y = +1) and a ring from 1 to 2 (Y = -1)

  then you should be able to see why (hint: the equation I just wrote) $f_{x|Y=+1} = 1/\pi$ for $||X|| \leq 1$ and $f_{x|Y=-1} = 1/3\pi$ for $1 \leq ||X|| \leq 2$. The fact that they are equiprobable mean $f_Y(Y = +1) = f_Y(Y = -1) = \frac{1}{2}$ which means you can write the density of X, f_X.

Elements of Statistical Learning

Contents

ESL **Fall 2016**

Linear Regression:

Table of Contents Local *Scribe: Brandon McKinzie*

- Assumption: The **regression function** $\mathbb{E}[Y|X]$ is linear[26] in the inputs $X_1, \ldots, X_p$.
- Perform well for...
  - Small numbers of training cases.
  - Low signal/noise.
  - Sparse data.

Models and Least-Squares

- The **model**:

$$f(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j \tag{3.1}$$

- Most popular **estimation method** is least-squares.

$$RSS(\beta) = \sum_{i=1}^{n} \Big(y_i - f(x_i)\Big)^2 \tag{3.2}$$

$$= (\boldsymbol{y} - \boldsymbol{X}\beta)^T (\boldsymbol{y} - \boldsymbol{X}\beta)^T \tag{3.3}$$

  which is reasonable if training observations (x_i, y_i) represent independent random draws from their population[27].

- First two derivatives wrt to parameter vector β:

$$\begin{aligned} \frac{\partial RSS}{\partial \beta} &= -2\boldsymbol{X}^T(\boldsymbol{y} - \boldsymbol{X}\beta) \\ \frac{\partial^2 RSS}{\partial \beta \partial \beta^T} &= 2\boldsymbol{X}^T\boldsymbol{X} \end{aligned} \tag{3.4}$$

[26] or reasonably approximated as linear

[27] and/or if y_i's conditionally indep given the x_i's.

- Assuming that $\mathbf{X}$ has full column rank so that $\mathbf{X}^T\mathbf{X}$ is positive definite[28], set first derive to 0 to obtain the unique solution:

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{3.6}$$

- **Geometry of Least Squares** I GET IT NOW!
  - The $(p+1)$ column vectors of $\mathbf{X}$ span a subspace of $\mathbb{R}^N$.[29].
  - Minimizing $RSS(\beta) = ||\mathbf{y} - \mathbf{X}\beta||^2$ is choosing $\hat{\beta}$ such that the **residual vector** $\mathbf{y} - \hat{\mathbf{y}}$ is orthogonal to this subspace[30]. Stated another way, (the optimal) $\hat{y}$ is the *orthogonal projection of* $\mathbf{y}$ *onto the column space of* X.
  - Since $\hat{\mathbf{y}} = \mathbf{X}\hat{\beta}$, we can define this projection matrix (aka hat matrix), denoted as $\mathbf{H}$, where

$$\begin{aligned} \hat{\mathbf{y}} &= \mathbf{X}\hat{\beta} \\ &= \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \\ &= \mathbf{H}\mathbf{y} \end{aligned} \tag{3.7}$$

Why $Var(\hat{\beta}) = (\mathbf{X}^T\mathbf{X})^{-1}\sigma^2$**.**

- Note: **Variance-covariance matrix** ≡ Covariance matrix.
- Can express the **correlation matrix** in terms of the covariance matrix:

$$corr(\mathbf{X}) = \Big(diag(\Sigma)\Big)^{-1/2}\Sigma\Big(diag(\Sigma)\Big)^{-1/2} \tag{140}$$

  or, equivalently, the correlation matrix can be seen as the covariance matrix of the standardized random variables $X_i/\sigma(X_i)$.

- Recall from decision theory that, when we want find a function $f(X)$ for predicting some $Y \in \mathbb{R}$, we can do this by *minimizing the risk* (aka the expected prediction error EPE(f)). This is accomplished first by defining a loss function. Here we will use the

[28] A matrix is positive definite if it's symmetric and all its eigenvalues are positive. **What would we do here if X were *not* full column rank?**. **Answer:** $\mathbf{X}$ columns may not be linearly independent if, e.g., two inputs were perfectly correlated $\mathbf{x_2} = 3\mathbf{x_1}$. The fitted $\hat{\mathbf{y}}$ will still be projection onto $C(\mathbf{X})$, but there will be more than 1 way (not unique) to express that projection. Occurs most often when one or more (qualitative) inputs are coded in a redundant fashion.

[29] This is the **column space** $C(\mathbf{X})$ of $\mathbf{X}$. It is the space of $\mathbf{X}v\ \forall v \in \mathbb{R}^N$, since the produce Xv is just a linear combination of the columns in X with coefficients v_i.

[30] Interpret: $X\beta$ will always lie *somewhere* in this subspace, but we want β such that, when we subtract each component (WOAH JUST CLICKED) from the prediction, they cancel exactly,i.e. $y_i - (\mathbf{X}\beta)_i = 0$ for all dimensions i in $C(X)$. The resultant vector $\mathbf{y} - \hat{y}$ will only contain components outside this subspace, hence it is orthogonal to it by definition.

squared error loss $L(Y, f(X)) = (Y - f(X))^2$. We can express $EPE(f)$ as an integral over all values that Y and X may take on (i.e. the joint distribution). Therefore, we can factor the joint distribution and define $f(x)$ via minimizing EPE piecewise (meaning at each value of $X = x$.) This whole description is written mathematically below.

$$EPE(f) = \mathbb{E}[Y - f(X)]^2 \tag{2.9}$$

$$= \int \Big[y - f(x)\Big]^2 f_{XY}(x, y)dxdy \tag{2.10}$$

$$= \mathbb{E}_X\Big[\mathbb{E}_{Y|X}\Big[(Y - f(X))^2|X\Big]\Big] \tag{2.11}$$

and therefore, the best predictor of Y is a function $f : \mathbb{R}^p \to \mathbb{R}$ that satisfies, for each x value separately

$$f(x) = \arg\min_c \mathbb{E}_{Y|X}\Big[(Y - c)^2|X\Big] \tag{2.12}$$

$$= \mathbb{E}[Y|X = x] \tag{2.13}$$

which essentially defines what is meant by $\mathbb{E}[Y|X = x]$, also referred to as the **conditional mean**[31].

Bias-Variance Tradeoff

The expected test MSE, for a given value x_0, can always be decomposed into the sum of three fundamental quantities:

$$\mathbb{E}[y_0 - \hat{f}(x_0)]^2 = Var\Big(\hat{f}(x_0)\Big) + \Big[Bias(\hat{f}(x_0))\Big]^2 + Var(\epsilon) \tag{141}$$

which is interpreted as the *expected test MSE*: the average test MSE that we would obtain if we repeatedly estimated f using a large number of training sets, and tested each at x_0. The **overall test MSE** can be computing the average (of this average) over all possible values of x_0 in the TEST set.

- What **bias** means here: On the other hand, bias refers to the error that is introduced by approximating a real-life problem, which may be extremely complicated, by a much simpler model. For example, linear regression assumes that there is a linear relationship between Y and $X_1, \ldots, X_p$. It is unlikely that any real-life problem truly has such a simple linear relationship, and so performing linear regression will undoubtedly result in some bias in the estimate of f. In Figure 2.11, the true f is substantially non-linear, so no matter how many training observations we are given, it will not be possible to produce an accurate estimate using linear regression. In other words, linear regression results in high bias in this example. However, in Figure 2.10 the true f is very close to linear, and so given enough data, it should be possible for linear regression to produce an accurate estimate. Generally, more flexible methods result in less bias.

[31] At the same time, don't forget that least-squared error assumption was built-in to this derivation.

- Returning now to the case where know (aka assume) that the true relationship between X and Y is linear

$$Y = X^T\beta + \epsilon \tag{2.26}$$

and so *in this particular case* the least squares estimates are unbiased.

- oh my fucking god. This is the proof: (relies on the fact that $Var(\beta) = 0$ since β is the true (NON RANDOM) vector we are estimating)[32]

$$\begin{aligned}
Var[\hat{\beta}] &= Var\Big[(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}\Big] && (142)\\
&= Var\Big[(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\Big(\mathbf{X}\beta + \epsilon\Big)\Big] && (143)\\
&= Var\Big[(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{X}\beta + (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\epsilon\Big] && (144)\\
&= Var\Big[\beta + (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\epsilon\Big] && (145)\\
&= Var\Big[(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\epsilon\Big] && (146)\\
&= \mathbb{E}\Big[\Big((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\epsilon\Big)\Big((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\epsilon\Big)^T\Big] && (147)\\
&= \Big((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\Big)\ \mathbb{E}[\epsilon\epsilon^T]\ \Big(\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\Big) && (148)\\
&= \Big((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\Big)\ \sigma^2\ \Big(\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\Big) && (149)\\
&= \sigma^2\Big((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\Big)\Big(\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\Big) && (150)\\
&= \sigma^2(\mathbf{X}^T\mathbf{X})^{-1} && (151)
\end{aligned}$$

where we have assumed that the X are FIXED (not random)[33] and so the variance of (some product of Xs) $\times\ \epsilon$ is like taking the variance with a constant out front. We've also assumed that X^TX (and thus its inverse too) is symmetric, apparently.

Subset Selection (3.3)

- Two reasons why we might not be satisfied with 3.6:

[32] Also, $\forall a \in \mathbb{R} : Var(a + X) = Var(X)$

[33] another way of stating this is that we took the variance given (or conditioned on) each X

1. Prediction accuracy. Often have low bias, high variance. May improve if shrink coefficients. Sacrifices some bias to reduce variance.
2. Interpretation. Sacrifices some of the small details.

- Appears that subset selection refers to retaining a subset of the *predictors* $\hat{\beta}_i$ and discarding the rest.
- Doing this can often exhibit high variance, even if lower prediction error.

Shrinkage Methods (3.4)

- Shrinkage methods are considered *continuous* (as opposed to subset selection) and don't suffer as much from high variability.

ESL **Fall 2016**

Naive Bayes:

Table of Contents Local *Scribe: Brandon McKinzie*

Appropriate when dimension p of feature space is large. It assume that given a class $G = j$, the features X_k are independent:

$$f_j(X) \equiv f_j((X_1, X_2, \ldots, X_p)^T) = \prod_{k=1}^{p} f_{jk}(X_k) \tag{152}$$

which can simplify estimation [of the class-conditional probability densities $f_j(X)$] dramatically: The individual class-conditional marginal densities f_{jk} can each be estimated *separately* using 1D kernel density estimates.

Neural Computation VS 265

Contents

Supervised Learning: September 15

Scribe: Brandon McKinzie

- Perceptron review for the plebs.
- XOR problem not linearly separable.
- Learning rules for two-layer network:
  - $E^{(\alpha)} = \frac{1}{2}\sum_i [T_i^\alpha - z_i(x^\alpha)]^2$.
  - $\Delta V_{ij} = \Big[[T_i - z_i(x)]\Big]\frac{\partial z_i}{\partial V_{ij}} = \delta_{ij} y_j$.
  - That was outer layer. For hidden layer:

$$\Delta W_{kl} = \eta \sum_i [T_i - z_i(x)] \frac{\partial z_i}{\partial W_{kl}}$$

  - Chain rule the fuck outta ur shit
- **Second-order Methods**:
  - $E(w_0 + \Delta w) \approx E(w_0) + \Delta w^T \nabla E + \frac{1}{2}\Delta w^T H \Delta w$.
  - Minimized when $\nabla E + H\Delta w = 0$, thus $\Delta w* = -H^{-1}\nabla E$. Hessian, rather than approximating function as a line (like the gradient), approximates as a quadratic function.
  - **Momentum** is kinda second order.

$$\Delta w_{kl}(t+1) = -\eta \frac{\partial E}{\partial w_{kl}} + \alpha \Delta w_{kl}(t) \tag{153}$$

$$\Delta w_{kl} \approx -\frac{\eta}{d\text{fuck}en} \tag{154}$$

Neural Computation **Fall 2016**

Competitive Learning & Sparse Coding: September 27

Table of Contents Local *Scribe: Brandon McKinzie*

- Competitive Learning (what follows is unrelated)
  - Most real data non-Gaussian. (bad for PCA/Hebbian).
  - Try **non-linear Hebbian learning**.

$$\Delta w_i \propto y x_i \tag{155}$$

$$= f\left(\sum_j w_j x_j\right) x_i \tag{156}$$

    where we can expand f in a taylor series.

- Winner-take-all learning.
  - output neurons are connected to each other. fighting it out.

$$y_i = \begin{cases} 1 & u_i > u_j \forall j \neq i \\ 0 & \text{otherwise} \end{cases} \tag{157}$$

    where u are standard weighted sum of inputs.
  - Learning rule:

$$\Delta w_{ij} = \eta y_i (x_j - w_{ij}) \tag{158}$$

    where, if y_i wins, moves towards vector x_j.
  - Visually, weight vectors shift to align with clusters in data.
  - weight vector with highest inner product wins competition, and is therefore the one that moves toward the given data point (where each output has an associated weight vector).
  - Algorithm $\equiv$ K-means.
  - energy function:

$$E(\{w_i\}) = \frac{1}{2} \sum_{i,\mu} M_i^\mu |x^{(\mu)} - w_i|^2 \tag{159}$$

$$\Delta w_i = \eta \sum_\mu M_i^\mu (x^\mu - w_i) \tag{160}$$

- **Sparse Coding:**
  - Allow multiple units to be active.
  - Barlow paper in 1972 is genesis of the idea.
  - Sensory system is organized to achieve as complete a representation of the sensory stimulus as possible w/min number of active neurons. i.e. minimize number of neurons k constrained by wanting to preserve max amount of information as possible about the input.
  - Adapt the coding strategy to the data structure.
  - is in the middle of *local codes* (grandmother cells) and *dense codes* (e.g. ascii). i.e. in the middle of easy-to-read-out and maximum-combinatorial capacity.
- Autoencoder networks.

$$\min_{W,M} |x - \hat{x}|^2 \tag{161}$$

  Want to train output $\hat{x}$ to be same as input x, where data is passed through some bottleneck y bridged by W and M from input-¿middle-¿out. Idea is to exploit correlations in the input so that can pass through smaller space while preserving most of the information, and then passed back to the original (larger) space with a more sparse encoding. Basically is a compression algorithm. Also, see 'retinal bottleneck.'
- Bottleneck may have same number of units but with lower capacity (e.g. less bits per neuron).
- **sparse code bottleneck** limits the number of active units. i.e. middle space may in fact be much larger, but only allowed to use a subset of it when mapping $x \to \hat{x}$.

Neural Computation **Fall 2016**

Sparse Distributed Coding: September 29

Table of Contents Local *Scribe: Brandon McKinzie*

- VI simple-cell receptive fields are localized, oriented, and bandpass.
- PCA is really bad for such situations.
- To detect sharp edges in images, need high frequency and in-phase combinations.
- Higher-order image statistics:
  - phase alignment
  - orientation
  - motion
- want to move beyond pairwise correlations.
- WTA is too greedy, want more distributed strategy.
- Idea: **Projection pursuit**.
  - Look for low-dimensional projections that are as non-Gaussian as possible.
  - Projections tend to result in Gaussian distributions by the C.L.T.
  - Want to explore projections onto a weight vector until find something Non-Gaussian. Why? Because such a distribution could not have happened by accident.
- **Gabor-filter** response histograms are highly non-Gaussian.
- (Lab-related) Paper on *Forming sparse representations by local anti-Hebbain learning.*
  - Each neuron takes weighted input sum, as well as getting lateral inhibitaion by neighbors, but where the lateral weights are all negative. Put all through f, some sigmoidal non-linearity. "leaky integrator"
  - Want population-sparcity, so need neurons decorrelated. Have three learning rules: anti-Hebbian, Hebbian, and threshold modification.
  - Threshold modification resembles homeostasis.

    $$\Delta t_i = \gamma(y_i - p) \tag{162}$$

    which is essentially SGD. Think about average behavior, as it relates to y_i output and p. p is a constant to be determined. Feedback loop. Adjusts spiking threshold.

  - Anti-hebb guarantees neurons are decorrelated.

$$\Delta w_{ij} = \alpha(y_i y_j - p^2) \tag{163}$$

    where p^2 because this is what we would expected if i and j were decorrelated. There more coactive two neurons are, the more this drives them to repulse one another.

  - Standard hebbian rule

$$\Delta q_{ij} = \beta y_i (x_j - q_{ij}) \tag{164}$$

    relates to sparsity fraction of neurons.

- Problems:
  - Don't know how to deal with graded (i.e. non-binary) input signals. Non-discrete stuff.
  - No objective function. Would like way to characterize how well system is performing.
- Led to Bruno's work: **sparse coding for graded signals**
  - Data described by

$$I(x, y) = \sum_i a_i \ \phi_i(x, y) + \epsilon(x, y) \tag{165}$$

  - basis decomposition of input. neuron i with activity a_i means that need feature functions ϕ to describe model. Want the **neural activities a_i to be sparse**.
  - Constrain sparseness of a_i by imposing cost function on the activity:

$$E = \frac{1}{2}|I - \Phi a|^2 + \lambda \sum_i C(a_i) \tag{166}$$

    where first term: preserve information and second term: I want to be sparse.
  - Penalty function C shaped like really steep parabola on zero. Or could do $C = |a_i|$, v-shaped thing.
  - Energy function determines dynamics of system. Want neuron activity to be expressible as a function of the input I.
  - Compute coefficients a_i by gradient descent.

$$\tau \dot{a_i} = -\frac{dE}{da_i} \tag{167}$$

  - Neuron i inhibited by neuron j proportioanl to their functions *phi* inner products.

– self-inhibition of neuron back on itself makes it sparse.

– Learning rule:

$$\Delta\phi_i = -\eta \frac{\partial E}{\partial \phi_i} \tag{168}$$

$$= \Big[I - \Phi\hat{a}\Big]\hat{a}_i \tag{169}$$

Neural Computation **Fall 2016**

Foldiak Paper - Sparse Coding:

Table of Contents Local

Scribe: Brandon McKinzie

- Abstract: A layer of simple Hebbian units connected by modifiable anti-Hebbian feedback connections can learn to code a set of patterns in such a way that statistical dependency between the elements of the representation is reduced, while information is preserved.
- ***Introduction***
  - Input-space that is our surrounding is enormous, but most inputs are highly correlated, which the brain may exploit to transform the high-dimensional pattern inputs to symbolic representations. Objects may be defined as conjunctions of highly correlated sets of components that are relatively independent of other such conjunctions[34]
- ***Unsupervised Learning***
  - The complexity of the mapping to be learnt $\Leftarrow$ complexity of the input.
  - Unsupervised learning exploits statistical regularities in input to learn a more meaningful symbolic representation.
- ***The Hebb Unit***
  - Simple model of cell (basically perceptron)

$$y = \begin{cases} 1 & \sum_j w_j x_j > \text{thresh} \\ 0 & \text{otherwise} \end{cases} \tag{170}$$

  - Can be thought of as pattern matching; y is maximal when weight vector = input vector pattern.
  - Hebb proposed: connection should become stronger if the two units being connected are active simultaneously: $\Delta w_j = x_j y$.
- ***Competitive Learning***
  - Out of the units receiving weighted sums of the input, only activate the unit with the largest weighted sum; suppress the output of all others.

[34]Translation: objects are clumps of stuff that are usually found clumped together, and such that these clumps tend not to clump with other clumps.

- Results in a local, "**grandmother-cell**" representation.
- Limited in number of different inputs it can discriminate, and in ability to generalize.

- ***Sparse Coding***
  - Distributed coding: instead, code each input state by a set of active units (rather than just one).
  - Pros: combinatorics of input states increases representational capacity. Cons: situations where many units are active per input pattern, and fact that learning can be extremely slow.
  - **Sparse Coding** is a compromise between distributed and local representations.
- ***Decorrelation***
  - Units *within* a layer are connected by modifiable inhibitory weights, governed by an **Anti-Hebbian learning rule**: if two units in same layer are active, connection becomes more inhibitory[35].

[35] which discourages their joint activity

Neural Computation **Fall 2016**

Comprehensive Review:

Table of Contents Local *Scribe: Brandon McKinzie*

Unsupervised Learning

- ***Bruno:PCA***
  - First, let's get this straight. Difference between **covariance** and **correlation**:

$$\mathbf{COV}[X,Y] \triangleq \mathbb{E}[(X-\mu_X)(Y-\mu_Y)] \tag{171}$$

$$\mathbf{CORR}[X,Y] \equiv \rho_{XY} \triangleq \frac{Cov[X,Y]}{\sigma_X\ \sigma_Y} \tag{corr}$$

  - Consider input stream $\mathbf{x}$ that has linear pairwise correlations[36] among its elements. Mathematically, correlation between elements x_i and x_j would imply that

$$\langle x_i x_j \rangle = \frac{\mathbb{E}[x_i x_j]}{\sqrt{\mathbb{E}[x_i]\mathbb{E}[x_j]}} \neq 0 \tag{172}$$

    or, equivalently, that $\mathbb{E}[x_i x_j] \neq \mathbb{E}[x_i]\mathbb{E}[x_j] = 0$. Bruno is correct that linear pairwise correlations imply that $c_{ij} \neq 0$, he is *absolutely incorrect* to say that c_{ij} is an "average over many examples." That is nothing more than academic sloppiness at its finest.

- ***HKP:PCA***
  - Goal: Find a set of M orthogonal vectors in data space that account for as much as possible of the data's variance. Projecting the data from original N-dimensional space onto the M-dimensional subspace spanned by these vectors then performs a **dimensionality reduction**.
  - HKP actually states accurately what Bruno meant to state: The kth principal component direction is along an eigenvector direction belonging to the kth largest eigenvalue of the full **covariance matrix**

$$\langle (\xi_i - \mu_i)(\xi_j - \mu_j) \rangle \tag{173}$$

  - **For zero-mean data this reduces to the corresponding EIGENVECTORS of the correlation matrix[37] C**

[36] This is exactly what is meant by eq corr, Pearson's correlation coefficient. *Linear* because "it is a measure of the linear dependence between two variables X and Y."

[37] NOTICE HOW I DIDN'T SAY REDUCES TO THE CORRELATION MATRIX.

  - Note: I am now going to start from beginning of CH8 of HKP since I'm not understanding the stuff they are referencing FML

- ***HKP Ch8: Unsupervised Hebbian Learning***
  - Units need to learn patterns/correlations/categories in inputs and code the output. Units and connections display some degree of **self-organization**.
  - Redundancy provides knowledge: w/o redundancy there would be no patters to learn.

$$\text{MaxInfoPossible} - \text{InputContent} = \text{DegreeOfRedundancy} \tag{174}$$

  - **PLAIN HEBBIAN LEARNING.** Context: output will be continuous-valued and DO NOT have a winner-take-all character[38], and so the **purpose** is to measure familiarity or projecting onto principal components of input data.
  - Setup: Draw at each time step an input vector $\boldsymbol{\xi}$ from (multivariate) probability distribution $P(\boldsymbol{\xi})$ that has N components[39]. Network will learn to tell us - as output - how well an input conforms to the distribution[40]
  - (One linear output unit): Let V be a scalar-valued continuous output with a bunch of inputs pointing to it, with

$$V = \sum_j w_j \xi_j = \boldsymbol{w}^T \boldsymbol{\xi} = \boldsymbol{\xi}^T \boldsymbol{w} \tag{175}$$

  - Want large (on average) $V \leftrightarrow$ more probable $\boldsymbol{\xi}$. Why? Because then we can use the relative size of the output as a way of characterizing the sort of input it just received (see footnote 26 below). The weight update to do this is **plain Hebbian learning update:**

$$\Delta w_i = \eta V \xi_i \tag{176}$$

    where it is perhaps easier to think about the situation where $\Delta w_i = 0$ when analyzing, i.e. If $\xi_i = 0$ (which means it had nothing to do with the output), then don't increase it's weight[41].

[38] TODO: Come back and explain why this is true, because current Brandon thought otherwise.

[39] Confusingly, here N refers to the dimension of space that each input vector lives in (usually denoted by d.)

[40] **Q:** Come back and explain why we would want a network to do this. Biological relevance/analog? **A**: You need to view it in the context of the grandmother-cell. That's what this is all about. If a given neuron has a large linear output, then we have a good idea of what type of input went in; it was an input really similar to the weight vector. This begs the question, though: how does one determine a reasonable initialization for a given connected layer of weights to a single output? I suppose the answer is that this is the wrong question. Rather, we should interpret the outcome as resulting from a stream of particular inputs and, based on its future responses to inputs, we can determine what type of input went in. With the brain, this is like the jennifer aniston neuron: if that neuron fires, we can assume the person just saw something that resembled Jennifer Aniston.

[41] Minor TODO: Analyze case of non-binary (i.e. continuous both pos/neg) inputs/outputs.

- Problem: $\boldsymbol{w}$ grows without bound. However, suppose stable equilib exists for $\boldsymbol{w}$. This could happen for example, when considering that the update just performs $\boldsymbol{w} = \eta V \boldsymbol{\xi}$, where eventually $||\boldsymbol{w}|| >> ||\boldsymbol{\xi}||$ in addition to the fact that ξ is quite likely to be along $\boldsymbol{w}$. So at equilib, expect the updates to average to 0:

$$0 = \langle \Delta w_i \rangle \tag{177}$$
$$= \langle \sum_j w_j \xi_j \xi_i \rangle \tag{178}$$
$$= \sum_j \mathbf{C}_{ij} w_j \tag{179}$$

  where the brackets are *expectation values* in the sense that

$$\langle \xi_i \xi_j \rangle = \iint_{-\infty}^{\infty} \xi_i \xi_j f_{\xi_i \xi_j}(\xi_i, \xi_j) d\xi_i d\xi_j \tag{180}$$

  where f is the PDF for the two random variables in question. I suppose that, since strictly speaking $\boldsymbol{w}$ isn't a random variable, that it can be pulled out along with the summation. That satisfies me for now.

- Given that $\boldsymbol{\xi}$ can be interpreted as a column vector, we have

$$\mathbf{C}_{ij} \equiv \langle \xi_i \xi_j \rangle \tag{181}$$
$$\mathbf{C} \equiv \langle \boldsymbol{\xi} \boldsymbol{\xi}^T \rangle \tag{182}$$

  Now, to be perfectly clear, this is NOT the correlation, but I am so sick and tired of caring that I'm just going to accept their absolutely incorrect definition and move on.

- Since I've read ahead, I know that the following property will be important to remember:

$$\forall \boldsymbol{x},\ \boldsymbol{x}^T \mathbf{C} \boldsymbol{x} = \boldsymbol{x}^T \langle \boldsymbol{\xi} \boldsymbol{\xi}^T \rangle \boldsymbol{x} \tag{183}$$
$$= \langle \boldsymbol{x}^T \boldsymbol{\xi} \boldsymbol{\xi}^T \boldsymbol{x} \rangle \tag{184}$$
$$= \langle (\boldsymbol{\xi}^T \boldsymbol{x})^2 \rangle \tag{185}$$

- There are *only* unstable fixed points (unstable equilib) for the plain Hebbian learning procedure.
- **OJA'S RULE**. Goal: Modify plain Hebb rule such that $|\boldsymbol{w}| = 1$.
- Solution: Add a **weight decay** proportional to V^2:

$$\Delta w_i = \eta V (\xi_i - V w_i) \tag{186}$$

  and we see that Δw depends on the difference between the input and the back-propagated output[42]

[42] Say 'back-propagated output' because we are subtracting what was put into the network by the resultant output *times* the connection (weight) between the input and said output. Dwelling on this *would* be overly pedantic, so move on.

– Informal analysis for zero-mean data: The average component of ξ along w will be zero, but since this is an algorithm depending on an unstable equilibrium, it will tend to fall along the maximal eigenvector of $\mathbf{C}$.

– Oja's rule chooses the direction of $\boldsymbol{w}$ to maximize $\langle V^2 \rangle$.

– **Sanger's Learning Rule**. Setup: Now, instead of 1 output, have M output neurons with the hopes that they gives us the first M principal components of the input data. Architecture is ONE LAYER fully connected.

– The ith output is a linear neuron as usual given by

$$V_i = \sum_j w_{ij}\xi_j = \boldsymbol{w}_i^T \boldsymbol{\xi} = \boldsymbol{\xi}^T \boldsymbol{w}_i \tag{187}$$

– The Sanger's learning rule update for the connection *from* the jth input component *to* the ith output neuron (so we are only updating a single edge/line in the following) is

$$\Delta w_{ij} = \eta V_i \left(\xi_j - \sum_{k=1}^{i} V_k w_{kj} \right) \tag{188}$$

where the (converged) weight vectors to the output neurons are orthonormal and converge to the normalized eigenvectors in order of largest to smallest eigvals:

$$\boldsymbol{w}_i^T \boldsymbol{w}_j = \delta_{ij} \tag{189}$$

$$\boldsymbol{w}_i \to \pm \boldsymbol{c}^i \tag{190}$$

Neural Computation **Fall 2016**

Lab 4 & LCA Handout:

Table of Contents Local

Scribe: Brandon McKinzie

- Want to learn a "dictionary" from data[43]
- Encode input data such that it can be reconstructed from that code, where dim(encoding) ¿ dim(input).
- Given N-dimensional input, build $N \times M$ dictionary[44] (matrix) $\mathbf{\Phi}$ where each column ϕ_i is a dictionary element with corresponding coefficient[45] a_i. Want to assemble $a_i \phi_i$ into a vector of **activations**.
- **GOAL:** Minimize energy function E, defined as

$$E = \frac{1}{2}||S - \hat{S}||_2^2 + \lambda \sum_i^M C(a_i) \tag{191}$$

  where $\hat{S} = \sum_i^M a_i \phi_i$ is for some reason called the image reconstruction. View this like a regularization procedure where the terms mean: (1) smallest difference between true image and reconstructed image (**reconstruction quality**); and (2) limit the number of **active elements**[46] a_i.

- Want to minimize E such that reconstructs data with fewest number of active elements, expressed as

$$\underset{a,\ \mathbf{\Phi}}{\arg\min} \Big(E\Big) \tag{argminE}$$

  where I guess the double argmin means "minimize E by changing a and $\mathbf{\Phi}$ only and then give me the values of a and $\mathbf{\Phi}$.

- Popular cost function is the ℓ_1 penalty:

$$\sum_i^M C(a_i) = \sum_i^M |a_i| \tag{192}$$

[43] TODO:wtf does this mean.
[44] M ¿ N.
[45] Looks like $a_i \notin \mathbf{\Phi}$
[46] A.k.a sparsity constraints a.k.a limit activations.

- We compute coeff vector a using a "dynamic process"[47] that minimizes argminE.
- Method for computing the sparse code[48] from a given input signal S and dictionary element ϕ_i is the **Locally Competitive Algorithm**.

  > The model describes an activation coefficient, a_k, as the thresholded output of some model neuron's **internal state**, u_k, which is analogous to the neuron's membrane potential.[49]

- Here we compute the equation for state transitions (updates) from the energy function. First, for grad descent on an individual neuron's activity, $a_k(t)$:

$$-\frac{\partial E(t)}{\partial a_k(t)} = \sum_i^N \left[S_i \Phi_{ik} - \sum_{j \neq k}^M \Phi_{ik} \Phi_{ij} a_j \right] - a_k - \lambda \frac{\partial C(a_k)}{\partial a_k} \tag{193}$$

  where the constants are S and $\mathbf{\Phi}$. Want system to evolve over time to produce optimal set of activations $a(t)$.

- Meaning of ϕ_k. Associated with kth (output?) neuron. Indicates the connection strength [between that neuron and] each pixel in the input.
- What the fuck is the following shit

  > In this model, we are going to
  > nd a sparse code for one patch of an image at a time, so that all M neurons are connected to the same image patch, S.

[47] Okay well what the fuck is it?
[48] what the fuck is this
[49] what. the. fuck.

HKP 9.4 - Feature Mapping:

 Scribe: Brandon McKinzie

Nearby (similar) outputs corresponding to nearby (similar) input patterns. Such a map (similar inputs → similar outputs) is a **feature map**. The conventional case: 2 continuous-valued inputs x and y map (fully-connected) to a two-dimensional x,y grid. Want nearby input values (in the actual euclidean sense) (x, y) to be mapped closely in the output 2D grid.

Kohonen's Algorithm implements the self-organizing (feature) map by using competitive learning, where now we update weights going to the *neighbors* of the winning unit as well as those of the winning unit itself.

- Setup: N continuous-valued inputs ξ_1 to ξ_N, defining a point $\boldsymbol{\xi}$ in N-dimensional space. Outputs O_i are arranged in (typically) a 1-D or 2-D array fully connected via w_{ij} to the inputs.
- A competitive learning rule is used, choosing output O_i^* as winner, determined by

$$|\boldsymbol{w_i^*} - \boldsymbol{\xi}| \leq |\boldsymbol{w}_i - \boldsymbol{\xi}| \quad \textbf{(for all i)} \tag{194}$$

- The **Kohonen Learning Rule** is

$$\Delta w_{ij} = \eta \Lambda(i, i^*)(\xi_j - w_{ij}) \tag{195}$$

  where $\Lambda(i, i^*)$ is the **neighborhood function**, equal to 1 for $i = i^*$ and falls off with distance $|\boldsymbol{r} - \boldsymbol{r_i^*}|$.

- A typical choice for $\Lambda(i, i^*)$ is

$$\Lambda(i, i^*) = \exp\left(- \frac{|\boldsymbol{r} - \boldsymbol{r_i^*}|^2}{2\sigma^2} \right) \tag{196}$$

  where σ is width parameter that *is gradually decreased*. Apparently $\eta(t) \propto t^{-\alpha}$ where $0 < \alpha \leq 1$ is a good choice.

Neural Computation **Fall 2016**

Locally Linear Embedding: October 19

Table of Contents Local *Scribe: Brandon McKinzie*

LLE is an unsupervised learning algorithm for dimensionality reduction. Similar to PCA and MDS[50], LLE is called an *eigenvector method*. The basic idea is illustrated below in figure 3.

The **LLE algorithm**:

1. Compute the neighbors of each data point, X_i.
2. Compute the weights W_{ij} that best reconstruct each X_i from its neighbors, minimizing the cost in
$$ReconErr(W) = \sum_i |X_i - \sum_j W_{ij} X_j|^2 \tag{197}$$
by constrained linear fits.
3. Compute the Y_i reconstructed by the weights W_{ij}, minimizing the quadratic form in
$$\Phi(Y) = \sum_i |Y_i - \sum_j W_{ij} Y_j|^2 \tag{198}$$
by its bottom nonzero eigenvectors.

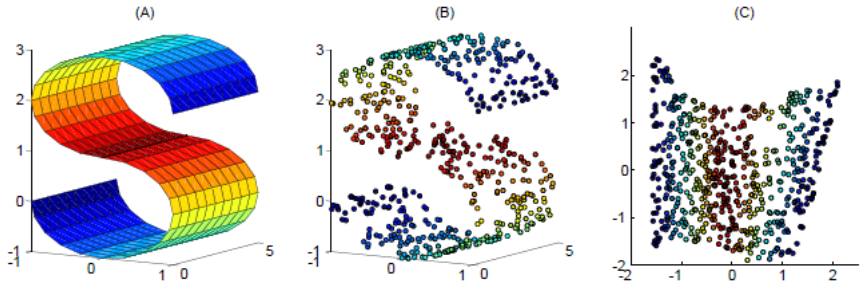


Figure 3: (A) Multidimensional sampling distribution with clear underlying manifold representation. (B) Points that were sampled. (C) The neighborhood-preserving mapping discovered by LLE.

[50]Multidimensional scaling

Some intuition/overview of the algorithm. We expect each X_i and its neighbors to lie on or close to a **locally linear** patch of the manifold. We characterize these patches by linear coefficients W_{ij} that reconstruct each X_i from its neighbors. As seen in eq. 197, the reconstructed point X_i is given by $\sum_j W_{ij} X_j$.

Computing/analyzing the weights W_{ij}. Minimize eq 197 subject to

$$\forall X_j \notin \text{Neighbors}(X_i) \ : \ W_{ij} = 0 \tag{199}$$

$$\sum_j W_{ij} = 1 \tag{200}$$

where the optimal weights are found by solving a least squares problem. Note that for a given data point, *the weights are invariant to rotations, rescalings, and translations of that data point and its neighbors.*[51] If the data lie on some nonlinear manifold of $d << D$, then there exists a *linear mapping* (approx) from the high-D coordinates of each neighborhood to global ('internal') coordinates on the manifold. Lucky for us, W can also do this![52]

Explanation of eqs. 197 198. Note that eq. 197 is minimized over the W_{ij}, while equation 198 is minimized over the Y_i. In English: We first want the weights W that reconstruct each X_i by its neighbors in the high-D space. Then, we want the low-d coordinates Y_i, representing the global coordinates on the manifold, that correspond to each X_i from the original space. **How it is minimized:**

> it can be minimized by solving a sparse $N \times N$ eigenvector problem, whose bottom d non-zero eigenvectors provide an ordered set of orthogonal coordinates centered on the origin.

Implementation of algorithm. Only one free parameter: number of neighbors per data point K. W_{ij} and Y_i are computed by 'standard linear algebra'.

[51] In other words, since the weights just characterize the local patch of the given data point, that patch shouldn't change if we shift the data, rotate it, or scale it. The neighboring points should remain the same.

[52] In particular, the same weights W_{ij} that reconstruct the ith data point in D dimensions should also reconstruct its embedded manifold coordinates in d dimensions

Neural Computation **Fall 2016**

Recurrent Neural Networks: October 20

Table of Contents Local *Scribe: Brandon McKinzie*

Lab 6 Overview. Briefly goes over how we can corrupt some number of bits and reconstruct a desired image [with hopfield nets]. Unfortunately, can get "spurious basins of attractions." Pushing down on some region of landscape causes pushing up of some other region. Want to carve energy landscape so that we push down only where we want.

Bump circuits and ring attractors. Want family of solutions (e.g. a line) that solutions drawn to (called line attractors). Head-direction neurons[53] look like an internal compass for animals; encode direction of head in *world coordinate system*. Different dots represent a single neuron's firing rate at different relative head directions. **Ring attractors**: population of neurons that with bumps that are stable (?). Convergence/stability because T_{ij} matrix is symmetric. Symmetric = fixed stable; Asymmetric =

Bruno shows simulation:

- 32 neurons where bar is activity of neuron.
- Start with random symmetric weight hopfield net.
- Eventually weights converge to gaussian-like bump; an equipotential pattern.
- If we add small asymmetry (gamma) to weights, then population (bump) would shift. Bump change is shifting position, and when the asymmetry stops (we stop moving our head) the population stays fixed. In English: moving head causes bump to move but when we stop moving, they stay put.
- **For more**: Read "catcher and zong" paper. I misspelled that.

[*Enter guest lecturer Alex Anderson*] ***Recurrent Neural Networks***:

→ Starts with handwriting network.
→ RNNs good for sequence prediction tasks with "long-term dependencies."

[53] Literally referring to direction of [e.g. some animal's] head

Backprop Review. Blobs do activation computation and transformers do propagations. Note: A^t is target output values.

Problem to Solve. Feed net a bunch of sentences and have it fill in the blank somewhere, based on the previous info it was fed. Mad libs. Have network understand particular frame of movie by exploiting context; just showing it a bunch of frames isn't enough/good approach.

RNN loops/Notation. Feed *time sequence* x_t to block A. Two figures in this slide are different reps of same thing; instructor prefers the right fig. H_k is hidden state we want to predict[54]. f can be some nonlinearity like *tanh*. In RNNs, cost function typically broken up over time; so C_k is cost at timestep k. Usually want hidden state to *summarize* the past. Hidden state traces out a trajectory over time [wut].

Unroll a RNN. Can basically turn RNN into a linearized hidden markov chain, where time proceeds to the right. Total cost is given by cost at each time step.

Long-term Dependencies. Shows toy model. Imagine ur an ant walking along graph. Given string of nodes, predict next letter each timestep [solve the question mark in slide]. Don't necessarily want/need whole past as input. Want to remember past [hidden] states, but they usually get overwritten; want to save it more efficiently. Key: want to make function simple, give the network parameterization.

Exploding/vanishing gradients. Local dependencies easy to learn.

→ Once we get to B, want network to output a U.
→ To learn, errors need to propagate back [in time], so we can change the weights that started the error: gradient of cost at timestep k with respect to initial weights using chain rule. Basically a product of k matrices.
→ If k large and matrices have eigvals less than 1, gradients *vanish*. If eigvals above 1, gradients *explode*. So what we want is for eigvals to be very near 1.
→ **Todo**: lookup relationship between eigval magnitudes and determinant.

[54] Analogy to hopfield: H is like hopfield B. X is like external I in hopfield.

Solution: Multiplicative Gating. Helps protect hidden state. MultGate can be either 0 or 1, and we multiply the hidden state by that value; if we 0 lose the hidden state; if 1 we keep the hidden state. Since binary functions not smooth/differentiable, continuous gating is better. [slide note: top row is w/o multgate, lower row is with multgate]. Key equation:

$$c_t = f_t \odot c_{t-1} + i_t \odot j_t$$

where $\odot$ is elementwise product.

Note: This is in TensorFlow now.

Neural Computation **Fall 2016**

Hopfield Networks Handout: October 26

Table of Contents Local

Scribe: Brandon McKinzie

Energy Function. The following governs the dynamic of pairwise recurrently connected networks.

$$E = \frac{1}{2}\sum_i \sum_{j \neq i} T_{ij} V_i V_j \tag{201}$$

For symmetric weights $T_{ij} = T_{ji}$, consider the change in energy ΔE resulting from making a positive change to V_k[55]

$$\Delta E = -\Delta V_k \sum_{i \neq k} T_{ki} V_i \tag{202}$$

which will be *negative* if both ΔV_k and the sum are positive, thus decreasing the overall energy (good). Conversely, if sum is negative, we should decrease value of V_k. **Critical assumption**: Symmetric $T_{ij} = T_{ji}$. Without this assumption, impossible to show the system will have fixed points.

> For a network with symmetric connections though, the dynamics will converge to so-called **basins of attraction**.

Setting the Weights. Goal: store pattern $\mathbf{V}^\alpha$ as basin of attraction in network. One approach: the **Hebbian prescription** $T_{ij} = V_i^\alpha V_j^\alpha$.

→ **Single memory storage**. Now, the summed input sent to, say, the ith unit in response to some $\mathbf{V}^\beta$ will be given by

$$U_i = V_i^\alpha \sum_{j \neq i} V_j^\alpha V_j^\beta \tag{203}$$

and thus if $\mathbf{V}^\alpha = \mathbf{V}^\beta$, U_i won't flip sign and the networks stays put.

→ **Multiple memories**. Now, need to form as many basins of attractions as memories we want stored. Set weights with a superposition over each desired *memory* $\mathbf{V}^\alpha$: $T_{ij} = \sum_\alpha V_i^\alpha V_j^\alpha$, and the corresponding response of the ith neuron is

$$U_i = \sum_\alpha V_i^\alpha \sum_{j \neq i} V_j^\alpha V_j^\beta \tag{204}$$

[55] If it is -1, change to +1, else just keep where it is.

Capacity for a Hopfield Network. If the patterns to store (memories) have *few elements in common*, then cross terms $\sum_{j \neq i} V_j^\alpha V_j^\beta$ tend to zero for $\alpha \neq \beta$ (since each V_j^α is ± 1 and a random average over ± 1 is zero) and U_i won't change. As we store more patterns which are *similar*, memories degrade and basins gone from desired locations. This **capacity** for Hopfield is $\approx 15\%$ of the number of neurons in network[56].

[56] Assuming the stored patterns are relatively dissimilar.

Discrete Math
CS 70

Contents

Discrete Math and Probability **Fall 2016**

RSA & Bijections: September 23

Table of Contents Local *Scribe: Brandon McKinzie*

RSA Alice communicates to Bob. Eve wants to figure it out. The message is

$$m = D\Big(E(m, s), s\Big) \tag{205}$$

Bijections. A **bijective** function $f : S \to T$ is defined as

- One-to-one: $f(x) \neq f(x')\forall x, x' \neq x \in S$.
- Onto: $\forall y \in T \exists x \in S$ where $f(x) = y$.

Theorem: *Two sets have same size iff there is a bijection between them.* Relation to modular arithmetic:

→ Can reverse mapping from S to T with inverse function $g : T \to S$ that maps outputs of f back to their input.

→ Consider $f(x) = x + 1 \mod m$. Is it 1-1?

→ Well, consider $g(x) = x - 1 \mod m$. It is the inverse of f, and so the function is one-to-one. **TIP:** To show a function is one-to-one, trying finding its inverse.

→ **Theorem:** If $\gcd(a, m) = 1$, $ax \neq ax'(mod m)$ for $x \neq x' \in \{0, \ldots, m-1\}$

→ Consider output space $T = \{0a \mod m, \ldots, (m-1)a \mod m\}$ and input $S = \{0, 1, \ldots, (m-1)\}$. Want to show that $S = T$.

- $T \subseteq S$, obvi.
- one-to-one mapping from S to T, so $|T| \geq |S|$ and T is superset of S.
- $\therefore S = T$.

→ Result: Since $S = T$, inverse of $a \mod m$ must exist because $1 \mod m \in T$.

Discrete Math and Probability **Fall 2016**

More RSA: September 26

Table of Contents Local *Scribe: Brandon McKinzie*

Example: RSA

- Public key: $(N = 77, e = 7)$ and $d = 43$ and $p \times q = 11 \times 7$.
- $E(2) = 2^e \mod 77 = 51 \mod 77 \longrightarrow D(51) = 51^{43} \mod 77$
- 51^{43} is big. **Repeated squaring** to the rescue.
- $51^{43} = 51^{2^5+2^3+2^1+2^0} \mod 77$. Calculate each factor alone $\mod 77$ and use results from lower powers to calculate higher powers.
- How to actually do it[57]: To compute $n^e \mod p$, divide exponent e repeatedly by 2, flooring each time [Save sequence of numbers this produces]. Starting from smallest number (probably 1), successively take n raised to that power mod 7. Use past results to help future ones. The last number in the sequence is e and you'll have $n^e \mod p$.

Properties of e, d, and exponents in modular arithmetic.

- **Theorem:**

$$m^{ed} = m \mod pq \text{ if } ed = 1 \mod (p-1)(q-1) \tag{206}$$

- **Corollary:**

$$a^{k(p-1)+1} = a \mod p \tag{207}$$

- **Lemma 1**: For any prime p and any a, b:[58] $a^{1+b(p-1)} \equiv a \mod p$
- **Lemma 2**: $\forall$ primes $p, q \neq p$ and $\forall x, k$: $x^{1+k(p-1)(q-1)} \equiv x \mod pq$
- **Prime Number Theorem:** Let $\pi(N)$ denote the number of primes less than or equal to N. For all $N \geq 17$

$$\pi(N) \geq N / \ln N \tag{208}$$

[57] See Discussion 5B
[58] Think Fermat's little theorem.

Important Notes on FLT[59]

- $gcd(a, pq) = 1 \Leftarrow gcd(a, p) = gcd(a, q) = 1$
- Before expanding the exponent in $a^{(p-1)(q-1)}$, realize that it's the same as $(a^{(p-1)})^{q-1}$

[59] Ctr-f: Fermat's Little Theorem fermat Fermats little theorem

Discrete Math and Probability **Fall 2016**

Polynomials: September 28

Table of Contents Local *Scribe: Brandon McKinzie*

Polynomials in modular arithmetic $P(x) \mod p$ consist only of points in the domain $\{0, 1, \ldots, p-1\}$.

Solve intersection of polynomials by equating and solving for x, use multiplicative inverses rather than dividing. "Whole world is $\mod p$.

Theorem: There is exactly one polynomial of degree $\leq d$ ([optionally] with arithmetic modulo prime p) that **contains** $d+1$ (particular/given) points.

Secret: I'm going to give you $2+1$ points of a parabola, and the *secret* is that parabola's y-intercept.

Shamir's **k out of n scheme:**

1. Choose secret $s = a_0 \in \{0, \ldots, p-1\}$ and randomly $a_1, \ldots, a_{k-1}$.
2. Let $P(x) = a_{k-1}x^{k-1} + \cdots + a_0$.
3. The ith shared point is $(i, P(i) \mod p$.

- **Robustness**: Any k shares gives secret.
- **Secrecy:** Knowing $\leq k-1$ points $\Rightarrow$ any $P(0)$ is possible.

Solving polynomial given enough points $\equiv$ **General linear system:**

- Given points: $(x_1, y_1), \ldots, (x_k, y_k)$, Solve...

$$a_{k-1}x_1^{k-1} + \cdots + a_0 \equiv y_1 \mod p \tag{209}$$
$$\vdots \tag{210}$$
$$a_{k-1}x_k^{k-1} + \cdots + a_0 \equiv y_k \mod p \tag{211}$$

Interpolation

- **Goal**: Want to find $P(x) = a_2x^2 + a_1x + a_0 \mod 5$ that contains $(1,3), (2,4), (3,0)$.

1. Find $\Delta_1(x)$ defined such that, for all x above except $x = 1$, $\Delta_1(x) = 0 \mod 5$ and evaluates to 1 at $x = 1$. Solution, as shown below, is to factor all $x - x_i$ together, evaluate at $x = 1$, and multiply the inverse of that to force/normalize $\Delta_1(x = 1) = 1 \mod 5$.

$$\Delta_1(x) = 3(x-2)(x-3) \mod 5 \tag{212}$$

   where 3 is inverse of $(1-3)(1-2) \mod 5$.

2. Repeat, constructing $\Delta_i(x) \forall x \in$ given points.

3. Now we have 3 polynomials that each evaluate to 1 only and 0 else for each given point. To make the $y - values$ align and get desired polynomial, compute result:

$$P(x) = y_1\Delta_1(x) + 4\Delta_2(x) + 0\Delta_3(x) \mod 5 \tag{213}$$

- **General interpolation:**

$$\Delta_i(x) = \frac{\prod_{j \neq i}(x - x_j)}{\prod_{j \neq i}(x_i - x_j)} \tag{214}$$

  where, if in modular field, you don't technically "divide" the lower product; rather, you should read that as a multiplication by $\text{denom}^{-1} \mod p$ (the multiplicative inverse).

- Construction via interpolation proves existence of unique solution.

Theorem: Any degree d polynomial has at most d roots.

Polynomial division

- Problem: Divide $4x^2 - 3x + 2$ by $(x - 3) \mod 5$.
- One approach is calculating while ignoring mod, then modding at end

$$\begin{array}{r|l} & 4x + 9 \\ \hline x-3 & 4x^2 - 3x + 2 \\ & \underline{-4x^2 + 12x} \\ & 9x + 2 \\ & \underline{-9x + 27} \\ & 29 \end{array}$$

  and answer is then $29 \mod 5 = 4$. You can also just mod 5 everything as you go, too.

- In general, dividing $P(x)$ by $(x-a)$ gives $Q(x)$ and remainder r. i.e.

$$P(x) = (x-a)\ Q(x) + r \tag{215}$$

Lemma 1: $P(x)$ has root a iff $P(x)/(x-a)$ has remainder 0.[60]

Lemma 2: $P(x)$ has d roots; $r_1, \ldots, r_d$ then[61]

$$P(x) = c(x-r_1)(x-r_2)\cdots(x-r_d) \tag{216}$$

Polynomials Discussion

1. **How many polynomials?** (I'll express my degree of certainty for each of my answers as a footnote)

   (a) Strictly speaking, $P(2)$ can only have 5 values since $GF(5)$. The number of distinct polynomials is $5 \times 5 \times 5 = 125$.[62]

   (b) The number of different pairs are $5^2 = 25$. The number of polynomials here is the number of distinct pairs of $P(i \neq 0), P(j \neq 0, i)$. This is $(5 \times 4) \times (5 \times 3) = 300$.[63]

   (c) ~~If we know k values, then we need $(d+1) - k = (d-k)+1$ more points to uniquely determine any polynomial. The next point can have $p-k$ possible values for x, and each of those can have p possible y values, for a total of $(p-k) \times p$ unique choices for the next point alone. For subsequent choices, the number of possibles decreases by a factor of p. Therefore, the number of different polynomials we could obtain, given that we are in $GF(p)$, is~~[64]

   Error: The main error in your line of thought is that many of those polynomials would be the same one. Although polynomials are indeed definable by a set of points, many such sets can define a single polynomial. If you're going to take this approach, you need to say more like: We have (d-k)+1 points, each of which could take on p different values, so the number of *distinct* polynomials is $p^{(d-k)+1}$. Ta-da.

2. **Lagrange Interpolation**. I have an issue with their wording: Should just say "of degree 3" since it says unique. Whatever[65]

   (a) $\Delta_{-1}(x) = \frac{(x-0)(x-1)(x-2)}{(-1-0)(-1-1)(-1-2)}$

   (b) $\Delta_0(x) = \frac{(x+1)(x-1)(x-2)}{(1)(-1)(-2)}$

   (c) $\Delta_1(x) = \frac{(x+1)(x-0)(x-2)}{(2)(1)(-1)}$

   (d) $\Delta_2(x) = \frac{(x+1)(x-0)(x-1)}{(3)(2)(1)}$

[60] To prove: use 215
[61] To prove: induction on number of roots. Take advantage of Lemma 1.
[62] Certainty: 95 percent.
[63] Certainty: 90 percent
[64] Certainty: ~~95 percent~~ ~~More like 40 percent~~ 0 Percent because I know I was wrong now.
[65] Certainty:90 percent only because algrebra errors.

(e) $p(x) = 3\Delta_{-1}(x) + 1\Delta_0(x) + 2\Delta_1(x) + 0\Delta_2(x)$

3. **Secret sharing** Generate a degree 2 polynomial. Give each TA two points of it. Give each reader 1 point of it.[66]

Polynomials Note

- ***General Definitions***
  - **Polynomial division**: If we have a polynomial p(x) of degree d, we can divide by a polynomial q(x) of degree *le* by using long division. The result will be: $p(x) = q(x)q'(x) + r(x)$ where[67] $\deg(r) < \deg(p)$. Subtlety: When you rewrite p in quotient/remainder form like this, where you've explictly said what you're dividing by (q), then $\deg(r) < \deg(q)$ by definition.
- ***Property 1***: A non-zero polynomial of degree d has at most d roots.
  - **Claim 1** $\Big[p(a) = 0\Big] \Rightarrow \Big[p(x) = (x-a)q(x)\Big]$ where $\deg(p) = d$ and $\deg(q) = d-1$.
  - **Claim 2**:[68] If $p(x)$ has d distinct roots a_i, then $p(x)$ can be written as $p(x) = c(x-a_1)(x-a_2)\cdots(x-a_d)$.
- ***Property 2***: Given $d+1$ pairs with all x_i distinct $\exists$ unique $p(x)$ of degree (at most) d such that $p(x_i) = y_i \forall i \in \{1, \ldots, d+1\}$.
- ***Counting***
  - Can specify any $d+1$ polynomial with either (a) it's coefficients (coefficient representation) a_i, or (2) a set of $d+1$ points (value representation) contained by the polynomial. Can convert rep (a) to rep (b) by evaluating at the points. Can convert (b) to (a) with lagrange interpolation.
  - IMPORTANT: When they say "how many distinct polynomials go through these.." and whatever, they apparently always assume that the x points are ordered, and you're only interested in the value of $p(x)$ at the next, as of yet unspecified, x point. Wtf?
- ***Exhaustive List of PROOF TECHNIQUES:***
  - Rewriting $p(x)$ in quotient + remainder form and exploiting properties of roots,degree of the quotient, etc.
  - Induction on the degree d of a polynomial.

[66] Certainty: 70 percent. Question seems open-ended and the wording is shit
[67] Check Piazza for followup on my question regarding this
[68] Claim 2 $\implies$ Property 1

– When thinking about number of polynomials in $[\dots]$, remember that a polynomial can be uniquely defined by its *coefficients*. Equivalently, can think of as defined by $d+1$ points; Note that there can be *many* such sets of $d+1$ points that define the same polynomial.

Discrete Math and Probability **Fall 2016**

Erasure Coding: September 30

Table of Contents Local *Scribe: Brandon McKinzie*

- Lecture outline:
  - Finish polynomials and secret sharing
  - Finite fields: Abstract Algebra
  - Erasure Coding
- Note: the $d+1$ points needed to specify any polynomial must have different x values (obvi).
- ***Finite Fields***
  - Proofs of uniqueness haven't depended on whether x is reals, rationals, complex numbers... but not integers since no multiplicative inverses. Only works if modulo a prime p and finite element sets.
  - Can still generalize all to **finite fields**. Denote arithmetic mod p as field F_p or $GF(p)$.
  - Field def (informal): set with operations corresponding to addition/mult/div.
  - **Fact:** The number of degree d polynomials over $GF(m)$ is m^{d+1}.
- Revisit **efficiency** of polynomial secret sharing (k of n).
  - Need $p > n$ to hand out n shares.
  - For b-bit secret, need[69] $p > 2^b$.
  - **Theorem:** There is always a prime between n and $2n$.
- ***Erasure Codes*** (error correcting codes)
  - **Problem:** Want to send message with n packets. Lossy channel: loses k packets.
  - **Question:** Can you send $n+k$ packets and recover message?[70]
  - Solution Idea: Use polynomials. "*Any* n packets (out of the $n+k$) should allow reconstruction of original n packet message."[71]

[69] so you can share any secret you want. Good to choose p $= 2^b + 1$.

[70] $n+k$ because, since we know k packets out of the n will be lost, we should send $n+k$ packets if we want a total of n packets to be received.

[71] Think polynomial secret sharing.

  - Restated: Any n **point values** allow reconstruction of degree $n-1$ polynomial.
  - **Erasure coding scheme:** Message consists of n packets denoted $m_0, m_1, \ldots, m_{n-1}$. Each m_i is packet.
    1. Choose prime $p > 2^b$ for packet size b (num bits).
    2. $P(x) = m_{n-1}x^{n-1} + \cdots + m_0 \mod p$.
    3. Send $P(1), P(2), \ldots, P(n+k)$.
  - Any n of the $n+k$ gives polynomial, and thus the message.

- Comparison: Erasure codes vs. secret sharing.
  - Secret sharing: each share is size of whole secret.
  - Erasure: each share (a packet) is size $1/n$ of whole secret.

- **Example**: Erasure codes
  - Send message $1, 4, 4$ containing $n = 3$ numbers, up to $k = 3$ of which can be lost.
  - Make $P(1) = 1$, $P(2) = 4$, and $P(3) = 4$.
  - Work modulo 7 to accommodate at least $n + k = 6$ packets.
  - Can construct via linear system:[72]

$$P(1) = a_2 + a_1 + a_0 \equiv 1 \mod 7 \quad (217)$$
$$P(2) = 4a_2 + 2a_1 + a_0 \equiv 4 \mod 7 \quad (218)$$
$$P(3) = 2a_2 + 3a_1 + a_0 \equiv 4 \mod 7 \quad (219)$$
$$(220)$$

    so $P(x) = 2x^2 + 4x + 2$. Send packets $(1,1), (2,4), (3,4), (4, P(4)), (5, P(5)), (6, P(6))$.
    Don't forget to take mods

Error Correcting Codes

- ***Erasure Errors***: (missing packets)
  - Note: I'm only writing info here that I didn't write in the previous section.
  - If each packet is a b-bit string, choose prime p to be any prime larger than 2^b.
  - Be careful to ensure that $n + k \leq p$, which is usually pretty easy.
  - If receiver only gets $n-1$ of the packets, there are exactly p polynomials of degree at most $n-1$ that agree with the received packets.

[72] Form is always the same: Plug in values for x into $a_{k-1}x^{k-1} + \cdots + a_1x + a_0 \mod p$. Don't forget to take mod on all coefficients!

  - "This error-correcting scheme is therefore **optimal**: it can recover the n characters of the transmitted message from any n received characters, but recovery from any fewer characters is impossible."
  - To prove that the linear system always has a solution and that it is unique (which is true), hint is to show that a certain determinant is non-zero.

- ***General Errors*** (individual packets may be corrupted, but all are transmitted)
  - **DISTINCTION BETWEEN ERASURE**: Rather than the message being the coefficients of the polynomial, now want to encode as what polynomial evaluates to. fml.
  - One can still guard against k general errors by transmitting only 2k additional packets or characters[73].
  - Encoded message: $c_1, c_2, \ldots, c_{n+2k}$ where $c_j = P(j)$ for $1 \leq j \leq n+2k$. At least $n+k$ of these are received uncorrupted[74]
  - Receiver has to find $P(x)$. Know that $P(i) = r_i$ on at least $n+k$ points, where r_i denotes the ith *received* value. There are k points where $P(i) \neq r_i$ because they have been corrupted (changed) during the transmission process.
  - If $e_1, \ldots, e_k$ packets corrupted, define degree k polynomial $E(x)$ as follows, and with relationship to $P(x)$:

$$E(x) = (x - e_1)(x - e_2) \cdots (x - e_k) \tag{221}$$
$$P(i)\ E(i) = r_i\ E(i) \tag{222}$$

    for $1 \leq i \leq n+k$ where received points are of form (i, r_i). For any $i = e_i$, $E(i) = 0$. This is true because: (1) out of the $n+2k$ received, $n+k$ match the desired $P(x)$ correctly, i.e. $P(i) = r_i$ for $n+k$ points and eq 222 is obviously true. For the other points (the ones that got corrupted), $P(i)$ will be some (as of yet unknown) value that is not r_i. However, eq 222 is still true because $E(x) = 0$ for any x that was corrupted.
  - Eq 222 is really $n+2k$ linear equations with $n+2k$ unknowns.
    * Unknowns are the coefficients of $E(x)$ and $Q(x) := P(x)E(x)$.

$$Q(x) = a_{n+k-1}x^{n+k-1} + \cdots + a_1 x + a_0 \tag{223}$$
$$E(x) = (1)x^k + b_{k-1}x^{k-1} + \cdots + b_1 x + b_0 \tag{224}$$

  - Convention seems to be that, if we want to send a message of size n, we encode that message directly ***in order*** as $P(1), \cdots, P(n)$, starting for some reason at 1. We then encode the extra k parts as ordered eval of $P(n+1), \cdots, P(n+k)$.

[73] only twice as many as in the erasure case

[74] Goal is still for receiver to determine the unique polynomial $P(j)$.

- The **degree of P(x) is** $deg(P) = n - 1$. In other words, we map the desired n-point message to $(n-1)+1$ points defining the degree $n-1$ polynomial.
- **Exhaustive procedure/example**:
  - Setup: Working over $GF(7)$. Message has $n = 3$ characters.
  - **UNKNOWN TO RECEIVER:** Desired message: $3, 0, 6$. Then we need $P(x)$ uniquely defined by the points $(1, 3), (2, 0), (3, 6)$. Therefore, $P(x)$ is degree $n - 1 = 2$ with $P(x) = x^2 + x + 1 \pmod 7$.
  - **KNOWN TO RECEIVER**: Know that $n = 3$, $k = 1$, and therefore they know that the received message of size $n + 2k = 5$ has 1 corrupted letter. They know that the following polynomials take the respective forms[75]

$$E(x) = x + e_0 \tag{225}$$
$$Q(x) = q_3x^3 + q_2x^2 + q_1x + q_0 \tag{226}$$
$$= r_xE(x) \tag{227}$$

  - Don't forget to take mods of coefficients along the way.
  - **Q**: Given that we know $k = 1$ points will be corrupted, why is it *exactly* that we need to send $n + 2k = 5$ points? **A**: See below. Basically, it is so we can guarantee that the recovered polynomial P is unique (and the one we sent).

[75] Fact: For any polynomials P and Q, it is true that deg(PQ) = deg(P) + deg(Q).

General Errors: October 3

Scribe: Brandon McKinzie

- Only going to write new information here.
- **Problem**:Communicate n packets $m_1 \dots m_n$ on noisy channel that corrupts $\leq k$ packets. Notice that it is $\leq k$ now.
- **Reed Solomon Code**: Make $P(x)$ of degree $n-1$.

$$P(1) = m_1; \dots; P(n) = m_n \tag{228}$$

- Send $P(1), \dots, P(n+2k)$.
- **Why n + 2k?**
- [76]. Okay I think I know why we need $n+2k$ points. It is related to the fact that we need to guarantee the receiver will reconstruct the *unique* polynomial $P(x)$ as opposed to some other polynomial.
- Claim: If two polynomials $P(x)$ and $P'(x)$ satisfy $P(i) = r_i$ and $P'(i') = r'_i$ for their own (separate) sets of $\geq n+k$ points in the received message of size $n+2k$, then $P(x) = P'(x)$.
- Proof: We know that $\leq k$ (so at most k) packets are corrupted. This means that $P(x)$ and $P'(x)$ share *at least* n points in common (out of their respective $n+k$ point sets), i.e. where for any of these points r_j, it is true that $P(j) = r_j = P'(r_j)$. Since they are degree $n-1$ polynomials that are uniquely defined by n points, it must be that $P(x) = P'(x)$.
- Lec then goes over example of $3, 0, 6$ from the note and works through it.
- jargon: calls $E(x)$ the **error locator polynomial**.
- kind of annoyed that he keeps saying things like $P(x)$ is degree $\leq n-1$, when the note seems to just say "equals". Come back later and explain whether or not I should care.
- However, says deg(E) = k.

[76] Paused lec at 24:20

Discrete Math and Probability **Fall 2016**

Error Review & Infinity: October 5

Table of Contents Local *Scribe: Brandon McKinzie*

- Continues on general-error encoding example from note.
- Technique is called **Berlekamp-Welch**.[77]
- Wants to answer existence and uniqueness of $P(x)$ and $Q(x)$. Existence is easy. n+2k in n+2k unknowns can be solved so yes it exists.
- uniqueness requires proof by contradiction assuming two different solutions exist. I don't see how this is any different from my claim/proof in the previous lecture. Time: 17:00. Identical proof as in note though regarding EQ = Q'E'.
- **Infinity an Uncountability**. Proof techniques are enumeration and constructing bijections.
- **Countably infinite**: A set is countably infinite if its elements can be put in one-to-one correspondence with the set of natural numbers.
- Determining if two sets are **same size**.
  - Make function $f : A \to B$.
  - Show f is one-to-one, defined as $\forall x, y \in A, x \neq y \implies f(x) \neq f(y)$. Show f is onto, i.e. $\forall s \in B, \exists c \in A,\ s = f(c)$.
  - **Isomorphism principle:** If there exists bijection $f : A \to B$, then $|A| = |B|$ (the cardinality of A is the same as cardinality of B).
- ***Number of subsets of*** $S = \{a_1, \ldots, a_n\}$**.**
  - Equal to number of binary n-bit strings. In other words, there exists a bijection f : subsets $\to$ n-bit strings.
  - **Proof**: For some subset x of $\{a_1, \ldots, a_n\}$, define

$$f(x) = \Big(g(x, a_1), \ldots, g(x, a_n)\Big) \tag{229}$$

$$g(x, a) = \begin{cases} 1 & a \in x \\ 0 & \text{otherwise} \end{cases} \tag{230}$$

[77] This technique, i guess, *uses* reed-solomon code. Whatever.

- Example: $S = \{1, 2, 3, 4, 5\}, x = \{1, 3, 4\}$. Then $f(x) = (1, 0, 1, 1, 0)$.
- The cardinality of the **Power set** of S is

$$|\mathcal{P}(S)| = |\{0, 1\}^n| = 2^n \tag{231}$$

  which is the number of n-bit binary strings, and *therefore* the number of subsets is also 2^n since f is a bijection.

- ***Infinity*** [38:00]
  - Natural numbers = "the counting numbers".
  - Any set S is **countable** if there exists a bijection between S and *some subset of* $\mathbb{N}$.
  - If the subset of $\mathbb{N}$ is finite, then S has **finite cardinality**. If infinite subset then countably infinite and say it has "the same cardinality as $\mathbb{N}$".
  - Note, if a bijection exists from A to B, then we automatically know one exists from B to A because function inverse guaranteed.
  - Comparing cardinality of $\mathbb{Z}$ to that of $\mathbb{N}$: Define $f : \mathbb{N} \to \mathbb{Z}$ where

$$f(n) = \begin{cases} n/2 & \text{if n even} \\ -(n+1)/2 & \text{odd} \end{cases} \tag{232}$$

    and check (1) one-to-one by proof by cases on $x, y \in \mathbb{N}$ and combinations of one/both being even/odd, and (2) onto by for $z \in \mathbb{Z}$, cases where its negative/nonnegative and showing that it's pre-image would be $\in \mathbb{N}$.

Discrete Math and Probability **Fall 2016**

Countability & Computability: October 7

Table of Contents Local *Scribe: Brandon McKinzie*

- **Lists** have natural ordering property where position of item in list is a natural number. One way of showing if list is countable is by **enumeration** of elements in that set. Enumerability $\equiv$ countability.
- When enumerating, need to be careful that each element has a *finite* specified position in the list.
- **Lemma:** Any subset T of a countable set S is countable.
- All countably infinite sets have the same cardinality.
- For finite sets S_1 and S_2, cardinality of $S_1 \times S_2$ is $|S_1| \times |S_2|$.[78]
- **Cantor's diagonalization** for analyzing the cardinality of $\mathbb{R}$.
  - Try enumerating. View as a table. Construct a number along the diagonal: digit i is 7 if row i's ith digit is not 7, 6 otherwise. Implies that the diagonal number is not in the list[79], but it is somehow in $\mathbb{R}$, which is a **contradiction**.
  - Note: We can say that, *since* the numbers in the range $[0, 1]$ are uncountable, and since they are a subset of $\mathbb{R}$, that $\mathbb{R}$ is uncountable.
- Can show a bijection between two uncountable sets, e.g. $f : \mathbb{R}^+ \to [0, 1]$.

  Computability:
  - **Barber Paradox**. Why is this supposed to be interesting? Proof by cases leads to contradiction.
  - Any definable collection is a set. Example:

$$\exists Y \forall x (x \in Y \iff P(x)) \tag{233}$$

    and "y is the set of elements that satisfies P(x)." Can apply to barber paradox.
  - Key notion here is **self-reference**.

[78] Note: seems to suggest that $\mathbb{N} \times \mathbb{N}$ is undefined. But countable... Check.

[79] If it were, say, the jth element of the list, then by definition its jth element could not be its jth element. Don't hurt yourself, it's simple.

- The **halting problem**: write program that checks if other program halts: $HALT(P, I)$ where P is a program, I is input. Determines if $P(I)$ [P run on I] halts or loops forever. Program itself is some text string, which is why it (a program) can be fed as input to a program. *This enables self-reference in computation. One program executing on itself is possible.*

- HALT does **not** exist. Proof: Assume there is a program called HALT and a program TURING(P).

  1. If HALT(P, P) = "**halts**". then define Turing such that it goes into an infinite loop.
  2. Otherwise, Turing halts immediately. It basically does the opposite.
  3. Assumptions: there is a program HALT and text that are both the programs TURING and HALT.
  4. Question: Does Turing(Turing) halt? Proof by cases.
     - Assume it does halt. Then HALT(Turing, Turing) = halts. Then we TURING(turing) loops forever. Contradiction.
     - Assume it loops forever. Then HALT(turing, turing) $\neq$ halts. Then Turing(turing) halts. Contradiction.

     and so program HALT does not exist.

Discrete Math and Probability **Fall 2016**

Counting: October 10

Table of Contents Local *Scribe: Brandon McKinzie*

Computability Wrap-up:

- Goes over Turing machine. Infinite tape with characters. Can be in a state, read a character. More left/right and read/write charcter.
- Universal turing machine: tape could be a description of a . . . turing machine.
- Church proved equivalent theorem about **Lambda calculus**.
- Godel proved his **incompleteness theorem**: any formal system is either inconsistent [false statement can be proven] or incomplete [the is no proof for some sentence in the system]. Godel also showed every statement corresponds to a natural number. wtf.

Counting:

- Related to questions of the form "How many . . . given [condition]?"
- **Product Rule**: Objects made by choosing from n_1 then n_2, . . . , then n_k, then the number of objects is

$$n_1 \times n_2 \times \cdots \times n_k \tag{234}$$

- **Permutations:** General case is "how many different samples of size k from n numbers **without replacement**." Answer:

$$n \times (n-1) \times \cdots \times (n-k+1) = \frac{n!}{(n-k)!} \quad [{}^nP_k] \tag{235}$$

- If order doesn't matter, count ordered objects and then divide by number of orderings[80]. Have n objects and want to choose k?

$$\frac{n!}{(n-k)! \times k!} = \binom{n}{k} \tag{236}$$

[80] Calls this "second rule of counting." The first rule is the produce rule.

- Suppose sampling with replacement but order doesn't matter. Famous example is **Stars and bars**: *How many ways can Bob and Alice split 5 dollar bills?* For each of 5 dollars pick Bob or Alice (2^5), "then divide out order??" Let a denote number of dollars for Alice, similarly for bob such that $a+b=5$, or in more general case $a+b=k$. There are apparently $k+1$ ways.
- General case[48:00]: If want to split up between, say, $k=3$, can split with **stars and bars**: $**|*|**$. Each sequence of stars and bars $\implies$ split.
- **Counting rule:** If there is a 1-to-1 mapping between two sets, they have the same size.
- **Sum rule**: For disjoint S and T, $|S \cup T| = |S| + |T|$.
- **Inclusion/Exclusion**: $\forall S,\ T,\ \ |S \cup T| = |S| + |T| - |S \cap T|$.

General stars and bars: k stars $n-1$ bars. There are

$$\binom{n+k-1}{n-1} = \binom{(n-1)+k}{n-1} = \binom{n+k-1}{k} \tag{237}$$

...in other words, $n+k-1$ positions from which to choose $n-1$ bar positions. WIKIPEDIA VERSION:

Theorem one

$\forall n, k \in \mathbb{Z}^+$: the number of k-tuples of **positive** integers, whose sum = n, is $\binom{n-1}{k-1}$ Translation: If each person must get something, there are $\binom{n-1}{k-1}$ ways to split n stars up among $k+1$ people.

Theorem two

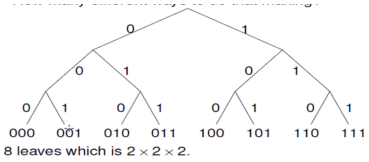
$\forall n, k \in \mathbb{Z}^+$: the number of k-tuples of **non-negative** integers, whose sum = n, is $\binom{n+k-1}{k-1} = \binom{n+k-1}{n}$. Translation: In general case, there are $\binom{n+k-1}{k-1} = \binom{n+k-1}{n}$ ways to split n stars up among $k+1$ people.

Since the above is confusing, here is the clearest possible way I can state it: If asked, how many ways to split up n [things] among k [people]? The answer is always

$$\binom{n+k-1}{k-1} \tag{238}$$

Examples

- How many 3-bit strings?



- How many outcomes for k coin tosses? 2^k.
- How many 10 digit numbers? 10^k.
- How many n digit base m numbers? m^n.
- How many **functions** f mapping S to T? $|T|^{|S|}$, because $\forall s_i \in S$ have $|T|$ choices for $f(s_i)$.
- How many **polynomials** of degree d modulo p? p^{d+1} coefficient choices and/or choices of the unique $d+1$ points (both lead to same answer).
- How many 10 digit numbers *without repeating a digit?*. $10 \times 9 \times \cdots \times 1 = 10!$.
- How many 1-to-1 functions from $|S|$ to $|S|$? $|S|!$.
- How many poker hands? Number of orderings for a given poker hand is 5!, so answer is $52!/(5!47!)$.
- How many different 5 star and 2 bar diagrams? 7 positions in which to place the 2 bars. $\binom{7}{2}$ ways splitting 5 dollars among 3 people.

Combinatorial Proofs

Let $|A| = n$. Prove $\binom{n}{k+1} = \binom{n-1}{k} + \binom{n-2}{k} + \cdots + \binom{k}{k}$.

- LHS. Number of subsets of size $k+1$ from set of size n.
- RHS. Ask yourself: What's another way I could find all subsets of size $k+1$?
  - Well, I could count the number of subsets that include the element $\min(A)$. This means I have k elements out of the remaining $n-1$ to choose from, i.e. $\binom{n-1}{k}$. That takes care of all subsets including $\min(A)$.
  - What about subsets where the smallest element is the *second-smallest* element in A?[81] Now we have k elements out of the remaining $n-2$ to choose from, i.e. $\binom{n-2}{k}$, and the pattern emerges.

[81] Notice that all such subsets do not include *any* of the subsets counted in the previous bullet point.

- Therefore, the jth term on the RHS represents the number of subsets of size k where the smallest item in the (jth) subset is the jth smallest element in A.

Textbook (Rosen) Notes

- If $A_1, \ldots, A_m$ are finite sets, then number of elements in the Cartesian product of these sets is

Equation

$$|A_1 \times \cdots \times A_m| = |A_1| \cdots |A_m| \tag{239}$$

- An **r-combination** of elements of a set is an unordered selection of r elements from the set. Thus, an r-combination is simply a subset of the set with r elements. The number of r-combinations from a set of n elements is often denoted as $\binom{n}{r}$.

Binomial theorem and related stuff.

Binomial Theorem

$$(x+y)^n = \sum_{j=0}^{n} \binom{n}{j} x^{n-j} y^j \tag{240}$$

which can be proved by counting the number of $x^{n-j}y^j$ terms. Since we have n products of sums $x+y$, we would need to *choose* $n-j$ x's from the n sums. But this is just $\binom{n}{n-j} = \binom{n}{j}$. Damn.

Corollaries to the Binomial Theorem

$$\sum_{k=0}^{n} \binom{n}{k} = 2^n \tag{241}$$

$$\sum_{k=0}^{n} (-1)^k \binom{n}{k} = 0 \tag{242}$$

$$\sum_{k=0}^{n} (2)^k \binom{n}{k} = 3^n \tag{243}$$

where all of these can be proven very easily using the Binomial Theorem (Hint: Think about what each implies about the values of x and y).

Other useful Identities.

Pascal's Identity and Vandermonde's Identity

$$\binom{n+1}{k} = \binom{n}{k-1} + \binom{n}{k} \quad PASCAL \tag{244}$$

$$\binom{m+n}{r} = \sum_{k=0}^{r} \binom{m}{r-k}\binom{n}{k} \quad VAND. \tag{245}$$

Note: It seems pretty popular to think about $\binom{n}{k}$ as "the number of bit strings of length n containing k ones."

Discrete Math and Probability **Fall 2016**

Midterm 2 Review: October 22

Table of Contents Local *Scribe: Brandon McKinzie*

Bijections/Sets

→ **[FA15.4.a]** If need bijection $f : (1, \infty) \to (0, 1)$, don't get too caught up with how any particular number should be mapped. Instead, think about what functions *over the given domain* map a positive real number above 1 to the interval 0, 1. The function they use is $1/x$. Then show it's one-to-one and onto in order to prove bijection.

→ To check if two sets A, B are *equal* (not just same size), check both that $A \subseteq B$ and $B \subseteq A$.

RSA/Modular Arithmetic

→ **Q [FA15.1.d]**: Given just N and e, how to quickly find d? **A:** You can't unless you know the factors of N.

→ **Q [FA15.1.e]**: What is the general meaning of 'signature of x'?

→ Write everything here about meaning of *relatively prime to [a number]* and what it implies/how to think about it.

- ⋆ Definition: a rel prime to b iff $\gcd(a, b) = 1$
- ⋆ Means that the two numbers share no common factor.
- ⋆ Multiplicative inverse of a exists mod b and vice versa.[82]
- ⋆ If inverse exists, then it is *also* relatively prime with the other number. This should be obvious because the inverse of the inverse exists (it is the original number) which means it must be rel prime.
- ⋆ **GENERAL FLT**: For any modulus n and any integer a coprime to n,

$$a^{\varphi(n)} \equiv 1 \pmod{n} \tag{246}$$

where $\varphi(n)$ denotes **Euler's totient function** which counts the number of integers

[82] Does it matter if one number is bigger than the other? **A: No it does not matter.**

between 1 and n *that are coprime with* n.

$$\varphi(n) = n\prod_{p|n}\left(1-\frac{1}{p}\right) \tag{247}$$

$$\gcd(m,n) = 1 \implies \varphi(mn) = \varphi(m)\varphi(n) \tag{248}$$

$$\varphi(p^k) = p^k\left(1-\frac{1}{p}\right) \tag{249}$$

→ **Chinese Remainder Theorem**: a theorem of number theory, which states that, if one knows the remainders of the division of an integer n by several integers, then one can determine uniquely the remainder of the division of n by the product of these integers, under the condition that the divisors are pairwise coprime.

→ Any RSA scheme is considered broken/breakable if knowing N allows one to deduce the value of $(p-1)(q-1)$, where you're only given N, not its factors. This is because, equivalently, breaking RSA means figuring out the value of $d = e^{-1} \pmod{(p-1)(q-1)}$.

- Also, unbreakable means at least as difficult as ordinary RSA. So, if you can make a bridge between the problem you're doing and the problem of ordinary RSA (given just N, e, find d), that suffices.
- **Q**: How to prove correctness of RSA?

Polynomials/Modular Arithmetic

→ Walkthrough of how smart person would approach "What is $3^{240} \pmod{77}$"

1. *Oh, 77 is 11 × 7, so I could think of as* $\pmod{77} = \pmod{pq}$.
2. *From things theorems like 206, I know that*

$$x^y \pmod{pq} \equiv_{pq} (x^y)^1 \pmod{(p-1)(q-1)} \equiv_{pq} x^y \pmod{(p-1)(q-1)}$$

3. *So I can rewrite and solve as*

$$3^{240} \equiv_{pq} 3^{240} \pmod{(10-1)(7-1)} \equiv_{pq} 3^{240} \pmod{6}0 \equiv_{pq} 3^0 \equiv_{pq} 1$$

→ **[FA15.2.b]** Write about polynomial intersections here. $P(x) - Q(x) = 0$ is max deg 4, so it has 4 roots, answer is 4.

→ Note: $n + x \equiv_n x \pmod{n}$.

→ Note: Modulo over polynomials should be *prime*.

→ General errors. Remember that for $E(x) = \prod_i (x - err_i)$, the err_i is an x value (!!!) and NOT a y value. It is an index.

Counting

→ **Stars and Bars**. If k bars and n stars, $\binom{n+k}{k} = \binom{n+k}{n}$ ways. I promise.

→ **Bins**. Convert to stars and bars problem with (numBins - 1) bars.

→ Don't forget the general sum rule: $\forall S,\ T,\ \ |S \cup T| = |S| + |T| - |S \cap T|$.

Computability

→ **Q [FA15.5.a]** Meaning of "undecidable"? **A:** an undecidable problem is a decision problem for which it is known to be impossible to construct a single algorithm that always leads to a correct yes-or-no answer.

→ **[FA15.5.a]** Master: halting problem, programs that return themselves.

→ **Quine**: A program that prints itself.

```
Print out the following sentence twice, the second time in quotes:
``Print out the following sentence twice, the second time in quotes:''
```

⇝ We can always write quines in any programming language.
⇝ Another example:

(Quine "s") (s "s")

which, if passed in $s = Quine$, will output (Quine "s"), which means we run the string s (now interpreted as a program) on itself.

→ **Theorem:** *Given any program $P(x, y)$, we can always "convert it" to another program $Q(x)$ such that $Q(x) = P(x, Q)$, i.e. Q behaves exactly as P would if its second input is the description of the program Q.*

→ **Halting Problem**.

- Proof relies on (1) self-reference, and (2) fact that we can't separate programs from data.
- Problem: Given the **description P of a program** and its input, write a program `TestHalt` that behaves as:

$$TestHalt(P, x) = \begin{cases} \text{"yes"} & \text{if P halts on input x} \\ \text{"no"} & \text{if P loops on input x} \end{cases} \tag{250}$$

- Proof: Try feeding program P the input P (itself as bitstring). Define

```
def Turing(P):
   if TestHalt(P, P) == "yes":
      loop forever
   else:
      halt
```

and consider behavior of Turing(Turing). It leads to proof by contradiction that TestHalt(P, P) cannot exist, since that was our main assumption this whole time.

→ **Reduction/TestEasyHalt** [**HARD**]

☞ General pattern to recognize for problem-solving: Try **reducing** (changing) the problem into the general form of the halting problem.

General Tips

⋆ Repeated squaring: It's easier if you write within the equation as you go. Example:

$$x^{16} \pmod{y} = (x^2)^8 \pmod{y} = ((x^2)^2)^4 \pmod{y} = \cdots$$

⋆ Write down cardinality of as many sets as possible and whether or not they are countable.
⋆ Rational numbers have decimal expansions that are either finite or periodic.

Discrete Math and Probability **Fall 2016**

Bayes' Rule, Independence, Mutual Independence: October 19

Table of Contents Local *Scribe: Brandon McKinzie*

Note: This lecture (23) corresponds to ***Note 14*** *(Combinations of Events).*

Conditional Probability Review.

- A and B positively correlated: $Pr(A|B) > Pr(A)$; Negatively correlated if $Pr(A|B) < Pr(A)$
- $B \subset A \implies$ A and B positively correlated.
- $A \cap B = \emptyset \implies$ A and B negatively correlated.
- Total probability rule: $Pr(B) = Pr(A \cap B) + Pr(\bar{A} \cap B)$.
- **True**: If $Pr(A|B) > Pr(A)$, then $Pr(B|A) > Pr(B)$.
- **False**: If $Pr(C|A) > Pr(C|B)$, then $Pr(A|C) > Pr(B|C)$.
- See lec at [18:00] for square-space probability illustration.

Independence. Two events A and B are independent if any of the (equivalent) statements hold:

$$Pr(A \cap B) = Pr(A)Pr(B) \tag{251}$$
$$Pr(A|B) = Pr(A) \tag{252}$$
$$Pr(B|A) = Pr(B) \tag{253}$$

Examples:

→ When rolling two dice, one blue and one red, define events A = sum is 7 and B = red die is 1. **Q**: Are these independent events?[83] **A**: Yes.

→ Now define events A = sum is 3 and B = red die is 1. **Q**: Are these independent events? **A**: no.

[83] I'm predicting yes they are, ~~because having the sum be seven doesn't tell us any information about which colored die was what.~~ You were right but *for the wrong reason.* The sum does actually give us some info in general, but the only reason it doesn't here is because it is 7, which is a possibility regardless of what the first die says. See the next example, which shows a case where they are not independent.

Mutual Independence. Events $\{A_j,\ j \in J\}$ are mutually independent if

$$Pr(\cap_{k\in K}) = \prod_{k\in K} Pr(A_k) \tag{254}$$

for all finite $K \subseteq J$.

- **Theorem:** *If all K_n are pairwise disjoint finite subsets of J, then events V_n defined by $\{A_j, j \in K_n\}$ are mutually independent.* Proof is in Note 25 example 2.7.
- **Fact:** $(A, B, C, \ldots, G, H$ mutually indep. $) \implies (A, B^C, C, \ldots, G^C, H$ mutually indep. $)$. **Inductive Proof**. Need to show eq 254 holds regardless of which events we take complement of or not. Proceed by induction on n, *the number of complements*. Base case For $n = 0$, this is the normal definition of mutual independence. Hypothesis: Assume true for n. Step. For $n+1$, need[84]

$$A \cap B^c \cap C \cap \cdots \cap G^c \cap H = X \cap H \ \setminus \ X \cap G \cap H \tag{255}$$

where $X := A \cap B^c \cap C \cap \cdots \cap F$. Recognize that $X \cap G \cap H \ \subset \ X \cap H$.

[84]Note: The **relative complement** of A with respect to B, denoted as $A \setminus B$, is defined as all objects that belong to A and not to B.

Discrete Math and Probability **Fall 2016**

Balls, Coupons, and Random Variables: October 26

Table of Contents Local *Scribe: Brandon McKinzie*

*Note: This lecture (25) corresponds to **Note 16** (Random Variables, Distribution, Expectation).*

Balls in bins. Have n bins and $m < n$ balls. Randomly (uniformly) throw balls, one by one, into bins. **Q**: What is the probability that after some m balls, that we don't have any collisions? (no two balls in same bin)[85]. Result:

$$Pr(\text{no collision}) \approx e^{-\frac{m^2}{2n}} \tag{256}$$

Coupons. Say there are large $n >> 1$ number of unique possible baseball cards. Each cereal box has a random card. You buy m boxes. The probability that you don't get a particular card (approx), and also a bound on the probability that you miss at least one card is shown below.

$$Pr(\text{miss a specific card}) \approx e^{-\frac{m}{n}} \tag{257}$$

$$Pr(\text{miss at least one card}) \leq ne^{-\frac{m}{n}} \tag{258}$$

Random Variables. Define random variable X to be the function $X : \Omega \to \mathbb{R}$ that assigns the value $X(\omega)$ to outcome ω. For more, see portion of section 2.7 on random variables. The **expected value** of a (discrete) random variable X is

$$\mathbb{E}[X] = \sum_a a \; Pr(X = a) \tag{259}$$

$$= \sum_\omega X(\omega) \; Pr(\omega) \tag{260}$$

where subscript a denotes all possible values of X, and ω denotes all possible outcomes in the sample space.

[85] Similar to having m people in room and wanting probability that no two people have same birthday ($n = 365$)

This suggests that if we repeat an experiement a large number N of times and denote $X_1, \ldots, X_n$ as the successive values we get, then

$$\mathbb{E}[X] \approx \frac{\sum_i X_i}{N} \tag{261}$$

Summary. If asked on final the definition of random variable X, write the following:

> X is a real-valued function of the outcome of a random experiment.

and some useful properties:

- $Pr(X = a) := Pr(X^{-1}(a)) = Pr(\{\omega | X(\omega) = a\})$ "The probability that X takes on the value a = The probability that random outcome of experiment happens to map into a"
- $Pr(X \in A) := Pr(X^{-1}(A))$.
- The **distribution** of X is the list of possible values and their probability:

$$\{(a, Pr(X = a)),\ a \in \mathcal{A}\}$$