6.034 10/14

Lecture

Parents' Weekend

☐ The Lay of the Land

☐ Nearest neighbors

- covers + Leukocytess
- Information retrieval
- Arm Control

☐ . Problems

* No cake w/o flour

* Simple ≠ Trivial

* Guard your sleep like you guard your wallet

First lecture on learning

We (humans) are always learning

- Some things fast learned
- some slow

Split learning into 2 parts

Learning

Reasoning Models

One shot experiences Language

Bulldozer Regularity

Classical (nearest neighbor)

most time

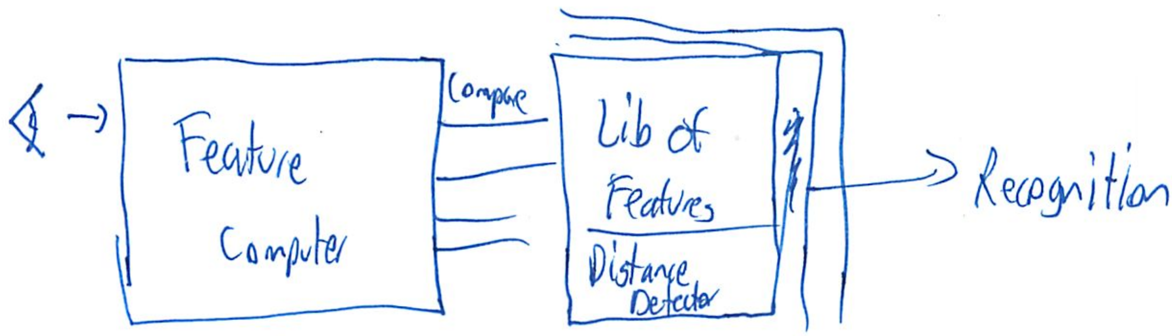
Stats Bayesian Search

Computer theory

Biology (Neural networks)

②

Nearest neighbor

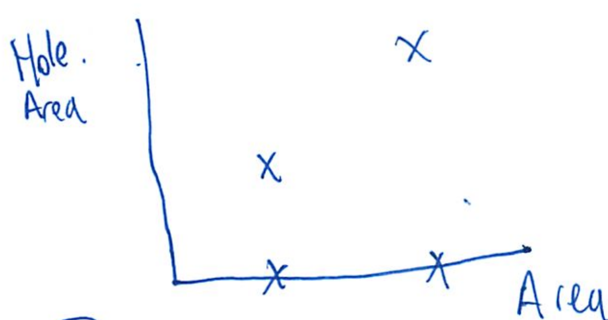


Calc features

Have pic of electrical covers

What features to calculate?

- cover area
- hole area



Populate w/ examples

If good in some respects will prob be ~~s~~ similar in other respects

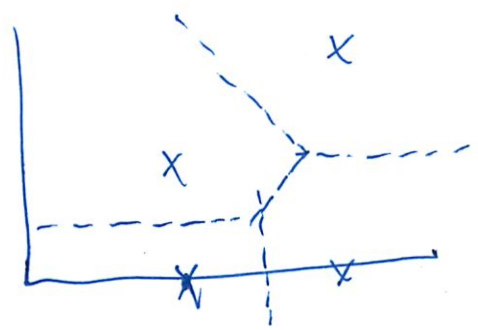
Like ~~w~~ weight - could prob guess from info above

3

Need to install some decision boundries

If two things - draw line in middle between

Draw a line between every pair

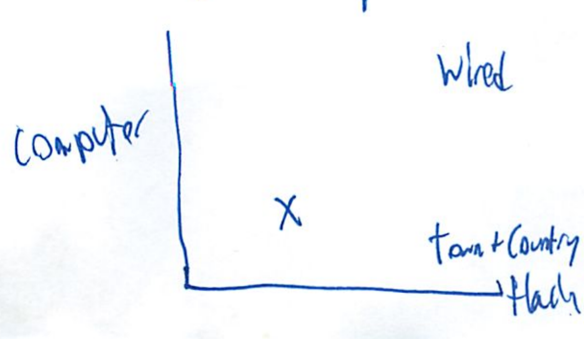


You start at w/ this

Frequently something better comes along

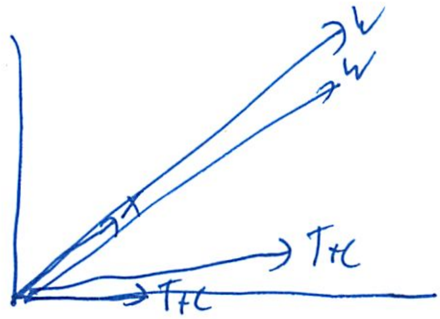
Watson is just this at its base

Find journal articles by comparing Stat. Improbable Words
- vs your probe



Town+Country-hack is a word for horse

Nearest neighbor would not work
Probe is about computers
Could draw vectors



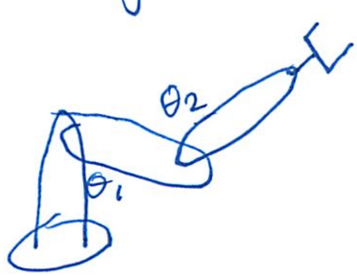
Min angles b/w probe vector and nearest article vector

L so Min θ

$$\text{Max } \theta = \frac{\bar{P} \cdot \bar{A}}{|\bar{P}| \times |\bar{A}|}$$ ← faster
can just do it once

Example 3 Robot Arm Control

- 2 degrees of freedom



- Want to take ball
- and ~~then~~ move it horizontally till it cant

We have 2 problems

- Kinematics: relate the x, y of ball to θ of arm
- Dynamics $\tau_1, \tau_2 \to \ddot{\theta}_1, \ddot{\theta}_2$

Need to solve a ~~bunch of~~ really long expression

Even if get expressions – it won't work

- need right inertia, etc
- real live variables
  - friction
  - vibration
  - worn parts

So need another approach

Get a gigantic table

θ_1	θ_2	θ_1'	θ_2'	θ_1''	θ_2''	τ_1	τ_2

(6)

Divide trajectory up into little increments

Each interval is a row of table

How fill table?

- Try moving arm and recording into table
- Then look for closest motion
- So it learns from testing

But is this pratical? Do we have enough memory?

For a baseball pitcher over his life time

100 Joints

100 Segments for each pitch

100 Bytes

100 Pitches/Day

100 Days/Year

100 Years

10^{12} bytes - not big anymore

≈ 1 TB

No exponential blow up anymore

7

10^{10} Nurons/Brain

10^{11} Nurons in Sarubelum

10^{5} Synapes in a Nuron

10^{16} So we cold easily have a look up table in our brain

Can a compter bounce a ball on a rachet

Sleep

Lots of types of learning does not work w/o sleep

Playing piano good before you go to bed

Custodians of sleep studies are in US Army

- US Troops in Gulf War friendly fire incidents

After 36 hrs very bad at firing

Also at MIT

8

After ~~8~~ 20 days w/ 6 hrs at 80% capability

Naps help - 30 min or more

Cafine helps
- didn't look so on chart

Hrs w/o sleep vs alcohol consumption
↳ similar / correlated

6.034 Lecture

10/17

Classification Trees

- ☐ Disorder
- ☐ Trees → Rules
- ☐ Rules → Fewer Rules
- * Occam's Razor
- * No Cake Without Flour

Romanian national anthem
- Berlin Wall
- Vampires

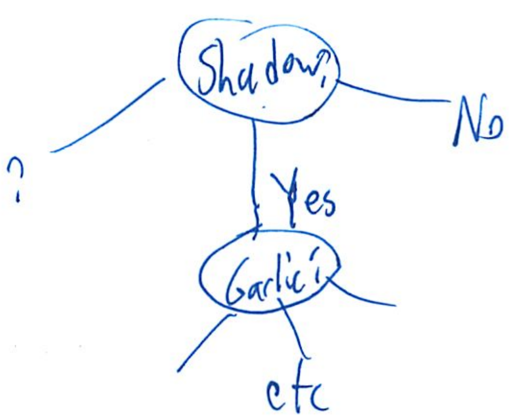
- Symptoms of Vampirism
  - Table

Vampire	Shadow	Garlic	Complexion	Accent ?
	can be, can't tell	Some non vampires might not have garlic	Some have gotten rid of accident	

In reality, many more items needed in table to be sure

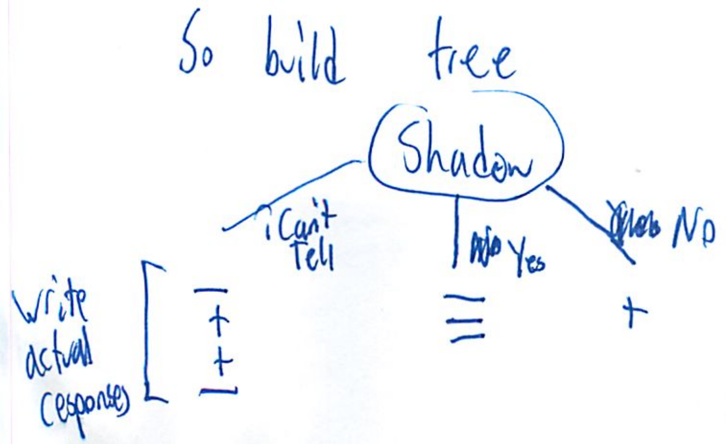
Can we do nearest neighbor
- but no #, just symbols!
- could make a spectrum sometimes
- but what else can we do

So can make a tree of tests



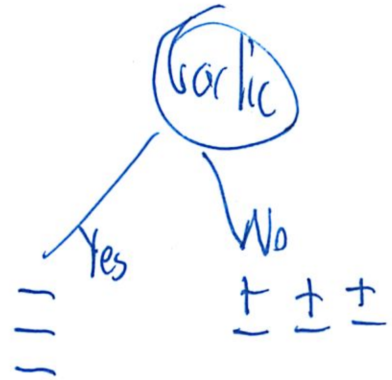
Simpler is better
L Occam's Razor
- also its NP - too hard to build

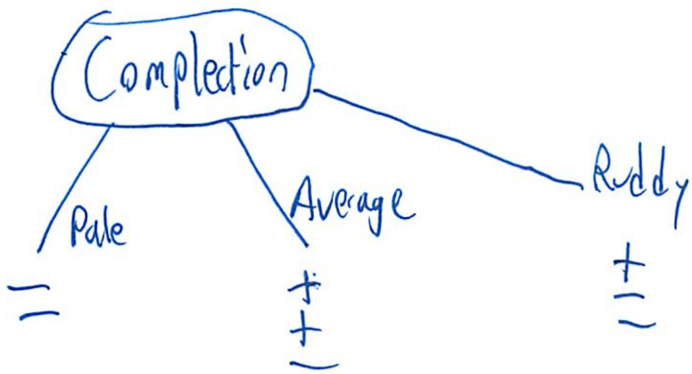
So build tree

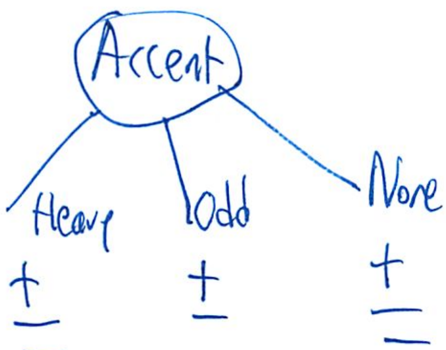


3

Look at all the tests first







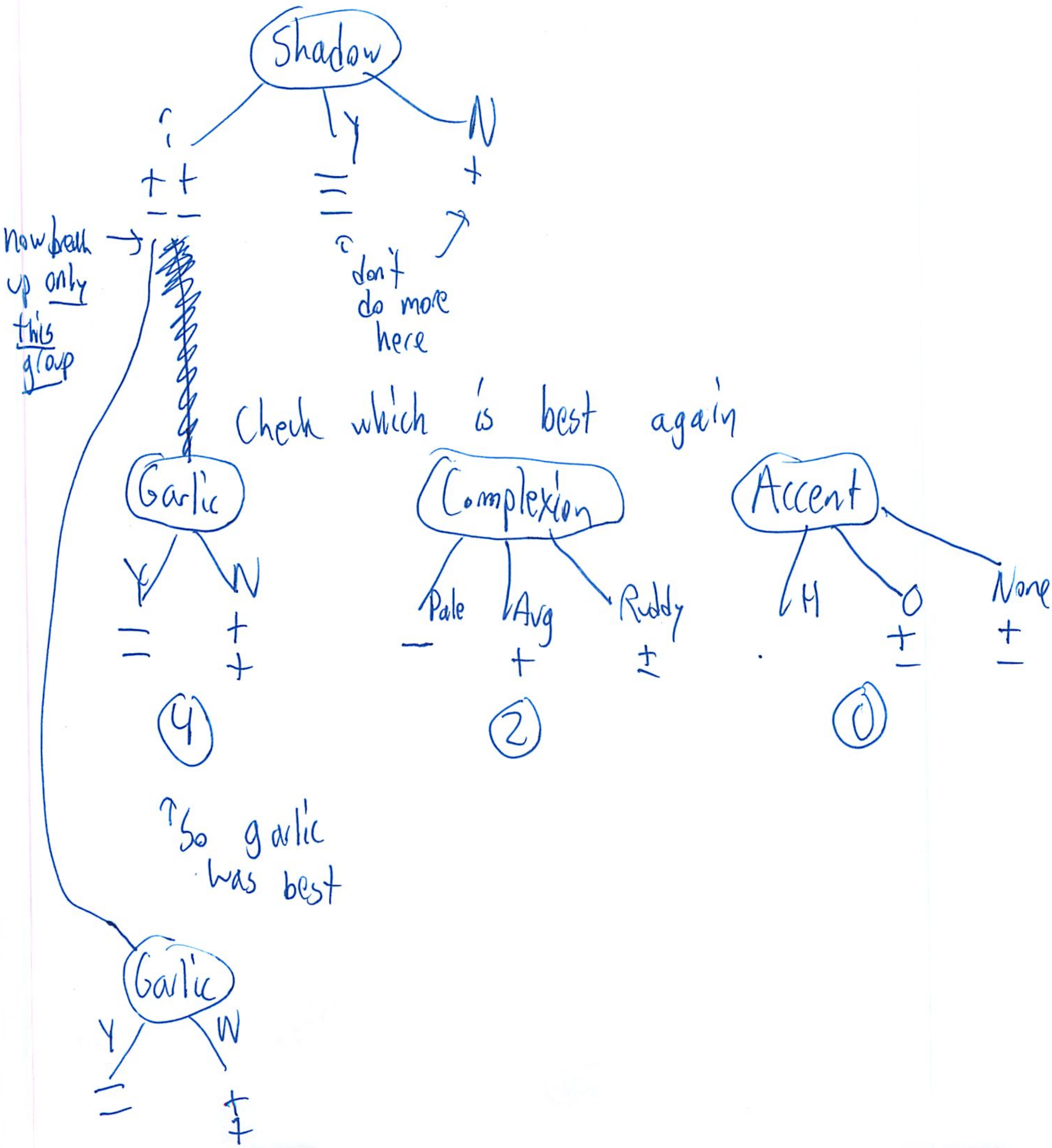
Now evaluate each test w/ a measure of "goodness"

- only works in classroom
- That it divides into homogeneous groups
- Count # items in homogeneous set

Shadow: 4   Complexion: 2
Garlic: 3   Accent: 0

(4)

So the first chat we want is shadow



(0) Done

The reason this test won't work is sometimes **no** homogenous sets

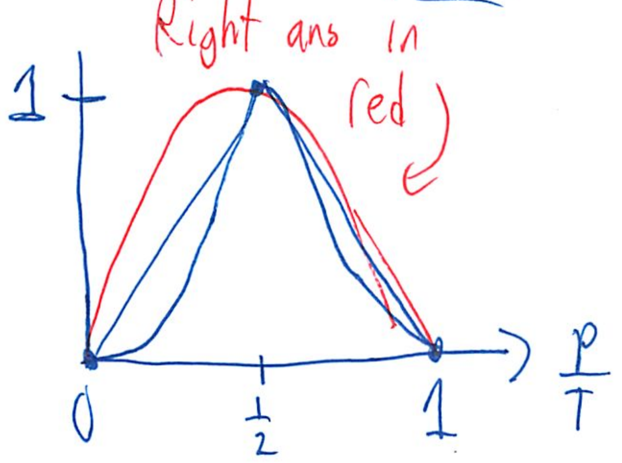
5

Need a measure of disorder

- 2 ways to do
- Thermodynamics
- Information Theory
- which is the best?

<u>Info Theory</u>

$\underline{D(set)} = -\frac{P}{T}\log_2\frac{P}{T} - \frac{N}{T}\log_2\frac{N}{T}$



If $P=N \rightarrow -\frac{1}{2}\log_2\frac{1}{2}\cdot 2$

$= +\frac{1}{2}\log_2 2\cdot 2$

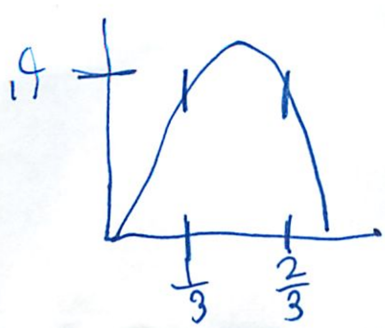
$= 1$

If $N=0 \rightarrow -1\log_2 1 - 0\log_2 0$

$= 0$ L'Hopitals Rule

Will see this a lot on exams

Note: Gets up pretty fast



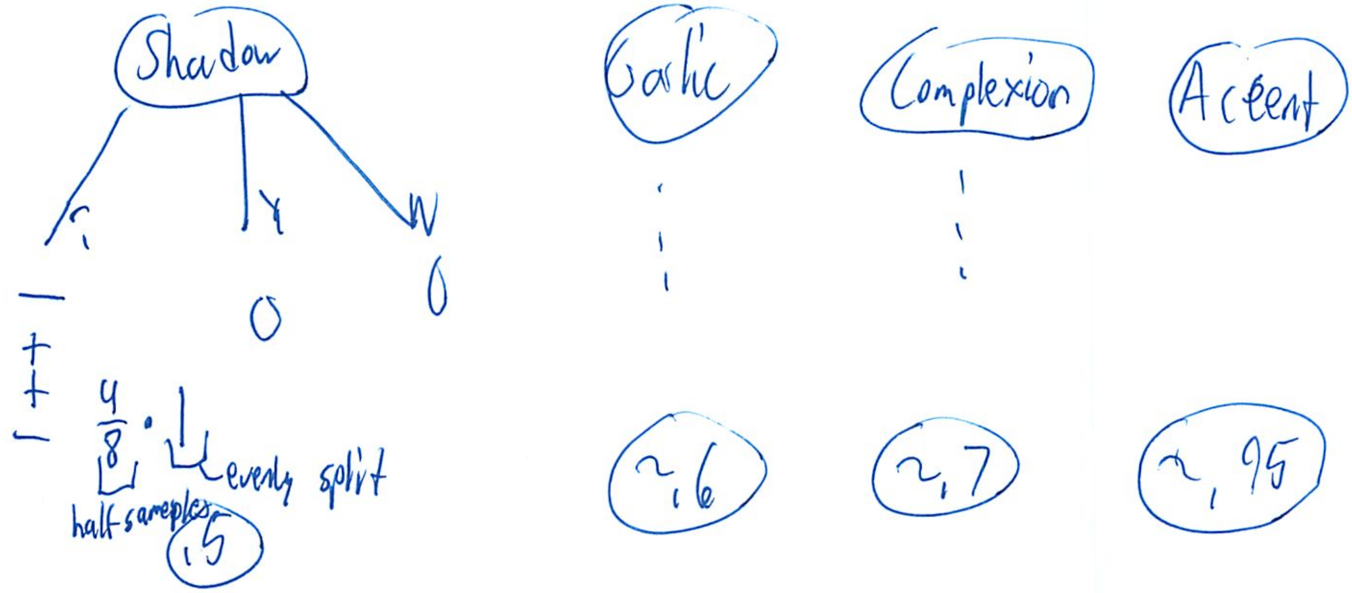
7 ← forgot pic

Now need to know how good a test is

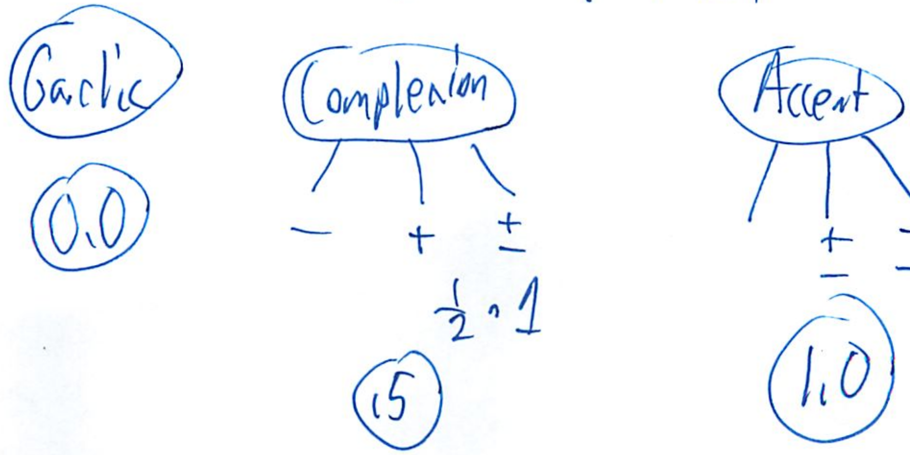
Weighted avg!

$$D(\text{Test}) = \sum_{\text{sets}} \frac{\#\text{ of samples in a set}}{\#\text{ of samples tested}} D(\text{set})$$

So back to original tests



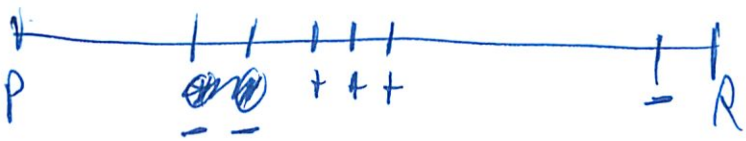
Then ~~next~~ ~~path~~ from shadow ? branch



8

How does this relate to nearest neighbor

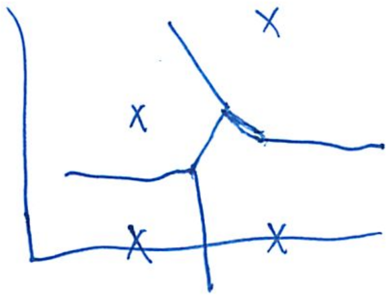
- this works w/ symbolic data
- this lets you ignore useless tests
  - ~~Real~~ this lets you identify those
- this lets you identify that some tests are <u>sometimes</u> useless
- this lets you take cost into account
- nearest neighbor works on numerical data
  - this can do numerical data as well
    - use a threshhold w/ the average ← bad idea



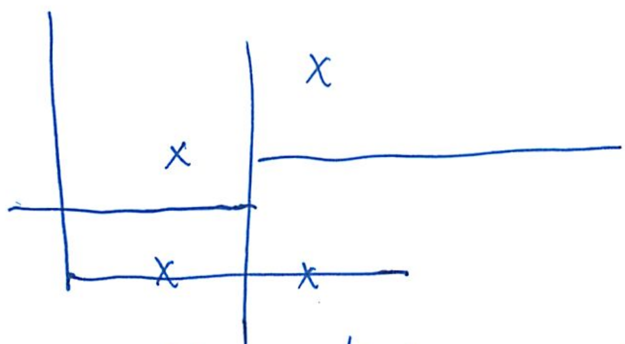
- min-max? – no
- best idea: threshhold b/w every point!
  - computers good at doing a lot of work

(9)

Remember our electrical cover nearest neighbor



It can put decision boundries



Divide up things that are still confused

Doctors don't like decision trees

Want simple rules

Can go from top to bottom and write rule

(1) If shadow test = ?
AND Garlic = Y
Then Not Vampire

(2) If Shadow = ?
Garlic = N
Then Vampire

(3) If shadow = Y
Then Not Vampire

(4) If shadow = N
Then vampire

10

But these rules can be too complicated

So can we condense?

Assume garlic = Y

		Vampire? +	−
Can't tell →	Shadow=?	+	=
Certain about shadow →	Shadow ≠?		−

So we know if it has a shadow or not - if it eats garlic its not a vampire

So can remove "if shadow = ?" from rule 1

Now we can combine (1) and (3)

If Garlic = Y || shadow = Y
 Then not a vampire
Else
 Is a vampire

Can only do all this if regularity is in data set
- if have wrong data, can't do anything

Why do martians think Diet Coke makes you fat?
 Since they see fat people drink it
 Confuse correlation + causation

6.034 Lecture

10/19

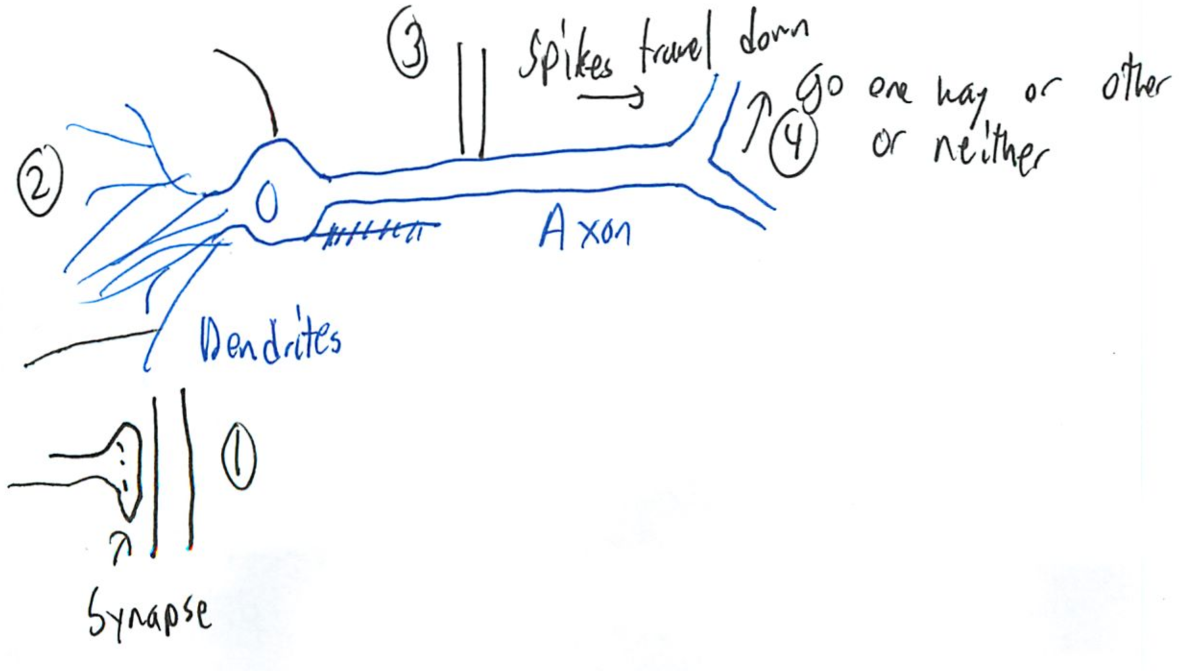
Neural networks

- ☐ Naive Neurobiology
- ☐ Mimicing ⟶
- ☐ Problems
- * 3 steps → Genius
- * Ask why 5 times

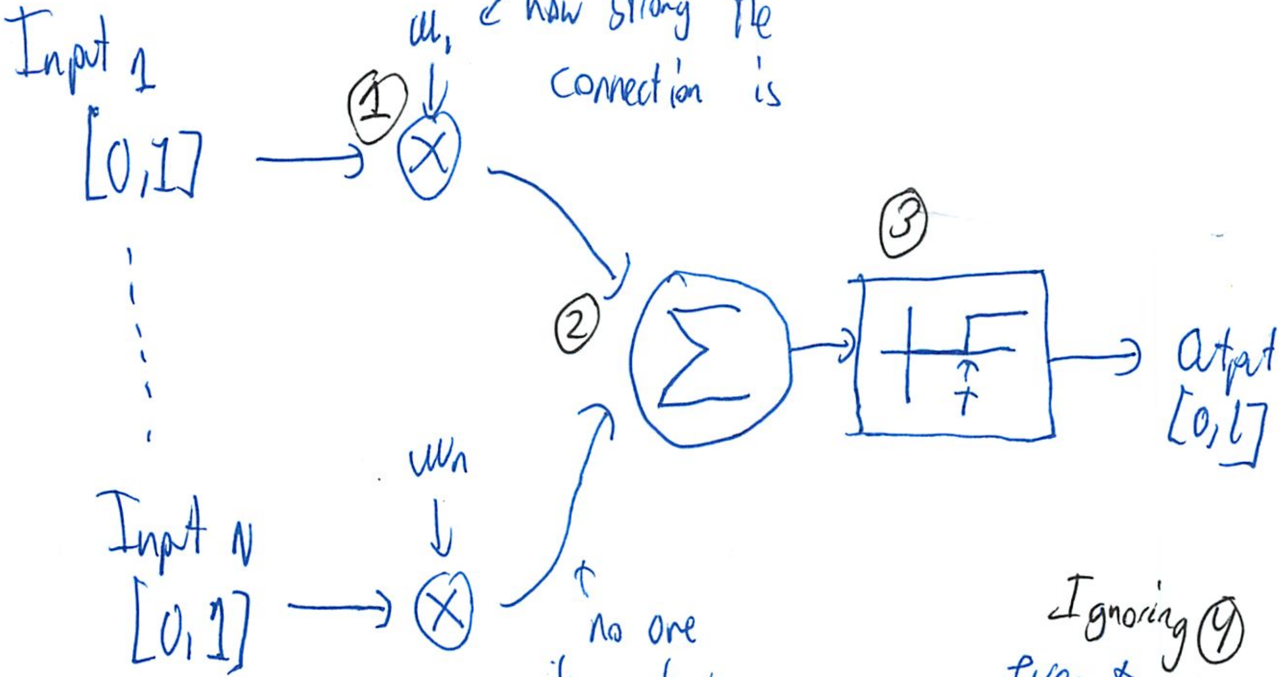
Neurons are what let humans think

If can't make a machine smart
- Can we teach it to be smart

Neuron

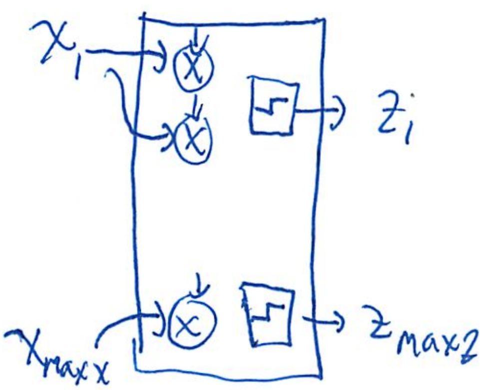


2



↑ no one knows how combined in humans

Ignoring 4 even though some people think its a big deal



$$\bar{z} = f(\bar{x}, \bar{w}, T)$$
$$\bar{d} = f(\bar{x}, \bar{w}, \bar{T})$$

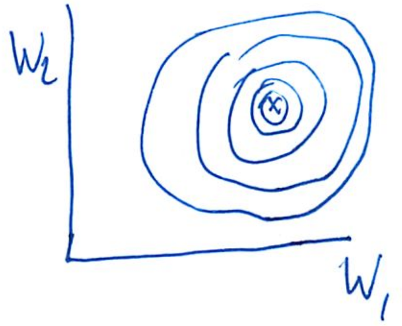
Change weights w, T so bring actual out closer to desired out

↳ performance metric $||\bar{d} - \bar{z}||$

But not convienent, so

$$-\frac{1}{2}||\bar{d} - \bar{z}||^2$$

(3) Output of performance Fn



To find peak, use hill climbing

Not good if lots of dimensions
└ 1000 neurons, ouch!

Find the gradient - 18.02

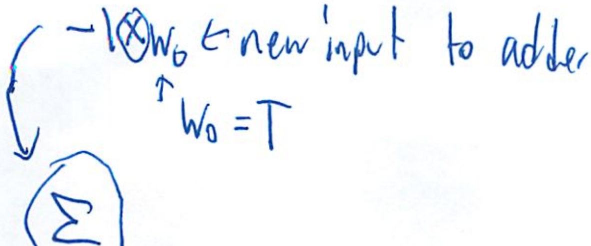
$$\Delta \bar{w} = \left(\frac{\partial P}{\partial w_1} \hat{i} + \frac{\partial P}{\partial w_2} \hat{j} \right) r$$

↑ rate constant

- helps us go to center - where Fn changes most

This will help us train our neural net

Can get rid of Threshhold to make problem simpler



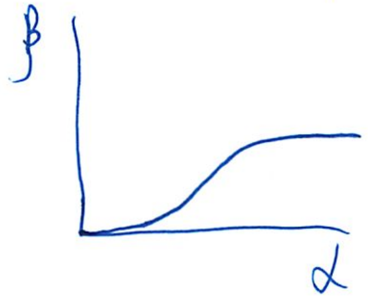
(4)

What must be true to use gradient ascent?

It must be differentiable everywhere

 is not continuous!

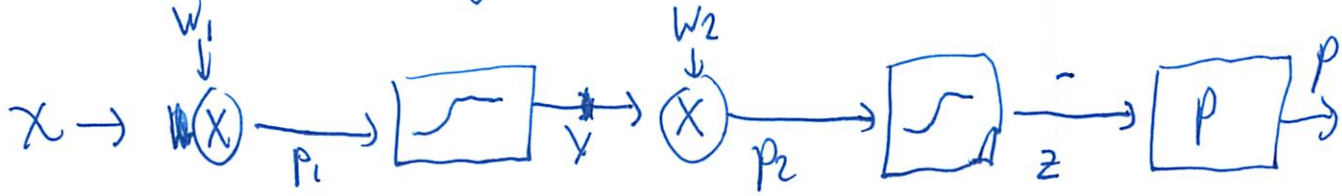
So change threshhold function
- smooth it out



$$\beta = \frac{1}{1+e^{-\alpha}}$$

Example
- 2 Neurons
- no threshholds

2nd simplest neural net



So can we compute the partial derivs?

$\frac{\partial P}{\partial W_2}$ → P is not a fn of w_2 ← but can use chain rule

$= \frac{\partial P}{\partial z} \frac{\partial z}{\partial W_2}$ how much does P change w/ W_2 changes

(5)

just scalars
no vectors

$$= \frac{\partial -\frac{1}{2}(d-z)^2}{\partial z} \cdot \frac{\partial z}{\partial w_2}$$

$$= (d-z)\frac{\partial z}{\partial w_2}$$

So now another chain rule

$$= (d-z)\frac{\partial z}{\partial p_2}\frac{\partial p_2}{\partial w_2}$$ Now can actually compute

$$= (d-z)\frac{\partial z}{\partial p_2} Y$$

$$\beta = \frac{1}{1+e^{-\alpha}}$$

"How do you differentiate a quotent?

$$= (1+e^{-\alpha})^{-1}$$

$$\frac{d\beta}{d\alpha} = -1(1+e^{-\alpha})^{-2} \times e^{-\alpha} \times (-1)$$

The -1 cancel each other out

$$= \frac{e^{-\alpha}}{(1+e^{-\alpha})^2}$$

$$= \frac{1}{1+e^{-\alpha}} \cdot \frac{e^{-\alpha}}{(1+e^{-\alpha})}$$

(e)

$$= \frac{1}{(1+e^{-\alpha})} \frac{1+e^{-\alpha}-1}{(1+e^{-\alpha})}$$

$$= \frac{1}{(1+e^{-\alpha})}\left[\frac{(1+e^{-\alpha})}{(1+e^{-\alpha})} - \frac{1}{1+e^{-\alpha}}\right]$$

$$= \beta(1-\beta)$$ ← simple resullt
deriv of output w/r to input
can be expressed in terms of output

$$= (d-z)\, z\, (1-z)\, y$$

Now

$$\frac{\partial P}{\partial W_1} = \frac{\partial P}{\partial z}\frac{\partial z}{\partial W_1}$$

$$= \frac{\partial P}{\partial z}\frac{\partial z}{\partial P_2}\frac{\partial P_2}{\partial W_1}$$

$$= (d-z)\, z(1-z)\frac{\partial P_2}{\partial W_1}$$

$$= (d-z)\, z(1-z)\, W_2 \frac{\partial y}{\partial W_1}$$

$$= (d-z)\, z\, (1-z)\, W_2 \frac{\partial y}{\partial P_1}\frac{\partial P_1}{\partial W_1}$$

$$= (d-z)\, z(1-z)\, W_2\, y(1-y)\, x$$

This was relativly easy
- picked very convient values
- end w/ 2 relativly simple formulas

Even starting to realize a pattern

$$(d-z)\, z(1-z)\, W_2\, y(1-y)\, x$$

etc

It's linear as you add layers horizontally or vertically

This is called a back propagation algorithm

Mechanism for training neural nets

Aren't magic - only useful in some situations

Animation of training
- desired outputs are close
- it can over fit - as it tries to go to peaks

(8)

- Increase rate function for bigger steps
  - L Converges fast
  - But it can get lost
  - Gets <u>way</u> off
  - Positive feedback - system ossilates - unstable

Quiz Wed: Games, Constraints, Steep, Object Recognition

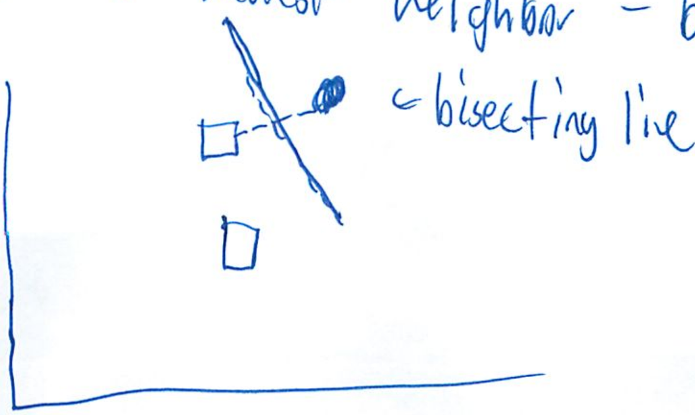
1. k-N Neighbors + Bandies
2. ID classification
   - info theory
   - trees
   - Wega - actual program

Nearest neighbors practice: credit cards p11

look at x nearest neighbors

look at what the majority of these are

like for 1 nearest neighbor - bandry drawling

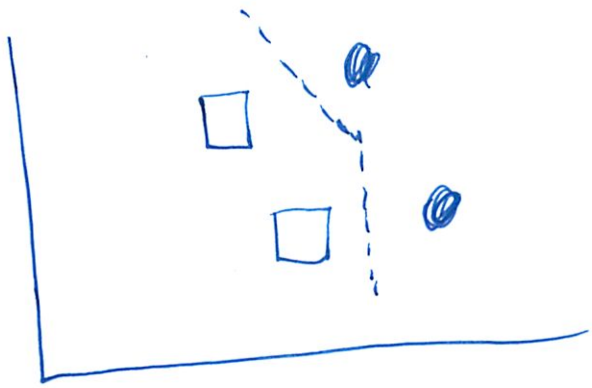


②

Temp. extend it far

Then pull it back when do more

when other lines ~~can~~ intersect

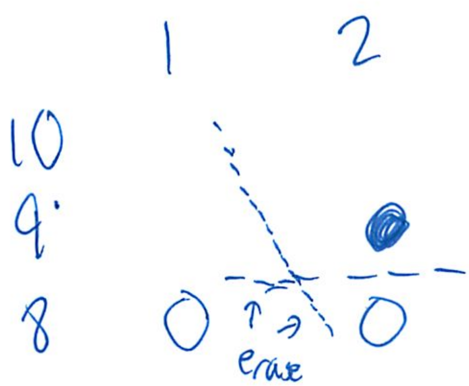


Never curved!

Why boundries - easier to see

---

p12



Then can test by arbitrally picking a pt

Which pair matters? - nearest neighbors

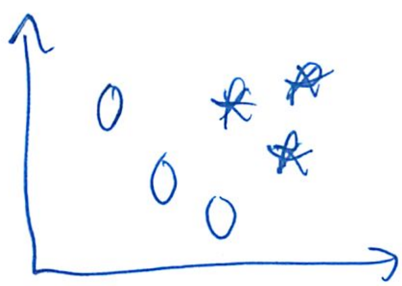
(on other sheet)

③

Classification Trees

Uses info theory

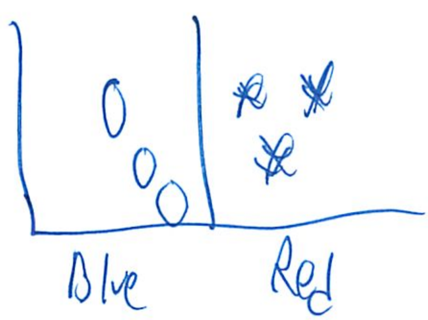
Trying to ↓ disorder when make cut in a plane



↑ this has 3 each

high disorder

So paramratize



So this is a very good cut

Use measures of entrophy

$$\sum_i -p_i \log_2 p_i$$

(4)

So for $\boxed{0\ 0\ +\ +}$

$$H = \underbrace{-\frac{1}{2}\log_2\frac{1}{2}}_{0} - \underbrace{\frac{1}{2}\log_2\frac{1}{2}}_{+}$$

$$= -\frac{1}{2}\cdot -1 - \frac{1}{2}\cdot -1$$

$= 1$ ← max disorder/entropy

For $\boxed{0\ 0\ 0\ 0}$

$$H = \frac{1}{1}\log_2 1$$

$$= \frac{1}{1}\cdot 0$$

$= 0$ ← min disorder/entropy

P7 table 9 possible cuts

- table above has calculated entropy for us

So base case

$\frac{4}{9}$ of 0 $\quad$ $\frac{5}{9}$ of *

.99 ← from table

5

Greedy 1 cut at a time

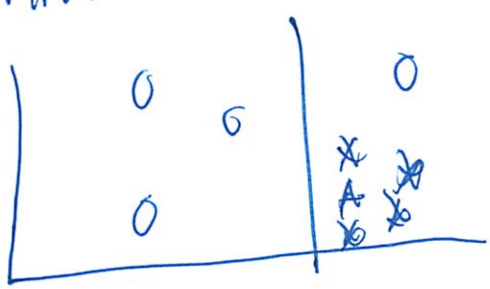
↳ lowest avg entropy

- takes the single best first step
- might not always work

Here we've already descitize

- otherwise need a mesh

So first



Now calc our entropy

- want weighted avg of entropy after cut
  - └ how many data pts in region

$$W_1 H_1 + W_2 H_2$$

$$\frac{3}{9} H_1 + \frac{6}{9} H_2$$

$$\frac{3}{9} \cdot 0 + \frac{6}{9} \cdot [-p \log p]$$

(6)

$$0 + \frac{2}{3}\left[-\frac{1}{6}\log_2\frac{1}{6} - \frac{5}{6}\log_2\frac{5}{6}\right]$$

Use handy dandy table for $\frac{1}{6}$
Table includes both terms

$\frac{2}{3} \cdot .65$

0.34

Goal is 0 remember

Now cut the remaining area ← if both areas ~~partial~~ mixed divide each sepertly!
– several solutions from here

Now can write rules from cuts

- If $x > 3$
  - No
    - 0
  - Yes
    - If $x > 3$
      - No
        - *
      - Yes
        - 0

Do w/ as few cuts as possible – otherwise over fit

7

p 13

2 possible cuts

Cut 1	Left	Right
$X = 1.01$	$W_1 H_1 + W_2 H_2$	
or 2, $X = 0$		

$n = 12$ pts

... (leave 10 min early)

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.034 Artificial Intelligence, Fall 2011
Recitation 6, October 20

K-NN and ID Tree Notes, Cliff Notes Version Prof. Bob Berwick, 32D-728

0. Basics

The general goal of machine learning = make accurate predictions about *unknown* data after being trained on *known* data.

There are two kinds of training: **supervised**, where the desired output is provided along with the input; and **unsupervised**, where the desired output is not provided. We will focus on **supervised** learning methods here.

Data comes in the form of examples, in the format: $(x_1, \ldots, x_n, y)$

Here, $x_1, \ldots, x_n$, are also known as **features, inputs,** or **dimensions**, while y is the desired (or observed) output or class label. A feature is a descriptor or property used to characterize the input for learning. We call the space where feature values define the coordinate axes a **feature space.** The input vector for each example defines a point in feature space

Both the x's and the y's can be **discrete** (taking on values from, say, $\{0, 1\}$ or some fixed set of label names or classes) or **continuous.**

In machine learning **training** we are given some (finite) set of $(x_1, \ldots, x_n, y)$ tuples. From this we output some learned classification or prediction function.

Note that K-Nearest Neighbors (KNN) and ID Trees are both **supervised, classification** learning algorithms

In machine learning **testing** we are given just $(x_1, \ldots, x_n)$ and the goal is to **predict** y with high accuracy.

Training error is the classification error measured using training data to test.

Testing error is classification error on data not seen in the training phase.

Checking for over-fitting - Cross-validation: split sample data into N subsets, use each subset as test set, the rest as training set; use average and standard deviation of performance on test sets to characterize prediction performance.

1. k-Nearest Neighbors

Training – Store all feature vectors in the training set, along with each class label.

Prediction – Given a query feature vector, find "nearest" stored feature vector and return the associated class.

"Distance" = $\sqrt{w_1(v_{a1} - v_{b1})^2 + w_2(v_{a2} - v_{b2})^2 + \ldots + w_n(v_{an} - v_{bn})^2}$

v_{a1} is the value of feature 1 in vector a

v_{b1} is the value of feature 1 in vector b

…

w_n is the weight for feature n (see below for some common metrics used for distance and other points about weighting)

1-NN: Given an unknown point, pick the closest 1 neighbor by some distance measure.
Class of the unknown is the 1-nearest neighbor's label.

k-NN: Given an unknown, pick the k closest neighbors by some distance function.
Class of unknown is the **mode** of the k-nearest neighbor's labels.
k is usually an odd number to facilitate tie breaking.

Normalization? To separate values clustered close together, divide by the standard deviation

Relevant features? All features are used; to find relevant ones, have to cross-validate, dropping features out.

What's the k? Can find best value using cross-validation

Voting for vectors? k-Nearest Neighbors votes on class for query feature vector; reduces sensitivity to noise

k-NN fixes a set of **decision boundaries** for whether a point is/is not in a given class. (We will see that other learning methods also fix decision boundaries).

How to draw 1-NN decision boundaries

Decision boundaries are defined as lines on which it is **equally likely** for a data point to be in any of the classes

1. Examine the region where you think decision boundaries should occur.
2. Find oppositely labeled points (+/-) and connect them, forming a line.
3. Draw perpendicular bisectors of these lines. (Use a pencil)
4. Extend and join all bisectors. Erase extraneously extended lines.
5. Remember to **draw boundaries to the edge of the graph** and indicate it with arrows! (a very common mistake).
6. Your 1-NN boundaries generally should have sharp edges and corners (otherwise, you are doing something wrong or drawing boundaries for a higher order k-NN).

Let's practice drawing k–NN boundaries. Turn to the end of the handout where we show you how the 'recipe' works; then we have a practice problem for you to try.

Here are some standard distance metrics to use

Euclidean Distance (common)	$D(\vec{w},\vec{v})=\sqrt{\sum_i^n (w_i-v_i)^2}$
Manhattan Distance (Block distance) - Sum of distances in each dimension	$D(\vec{w},\vec{v})=\sum_i^n \lvert w_i-v_i\rvert$
Hamming Distance - Sum of differences in each dimension	$D(\vec{w},\vec{v})=\sum_i^n I(w_i,v_i)$ $I(x,y)$ = 0 if identical, 1 if different.
Cosine Similarity - Used in Text classification; words are dimensions; documents are vectors of words; vector component is 1 if word i exists.	$D(\vec{w},\vec{v})=\dfrac{\vec{w}\cdot\vec{v}}{\lVert\vec{w}\rVert\lVert\vec{v}\rVert}=\cos\theta$

Note that it is also sometimes helpful to *transform* the data from one space to another. For example, if data are scattered in ring-like patterns of classes, then a transformation to polar coordinates typically helps. (Why?)

This is true of the practice problem we just did, as we will show in more detail below when using another learning method, ID trees.

Nearest neightbors, optional: How to weigh dimensions differently
In Euclidean distance all dimensions are treated the same. But in practice not all dimensions are equally important or useful!

Example: Suppose we represent documents as vectors of words. Consider the task of classifying documents related to *Red Sox*. If all words are equal, then the word *the* weighs the same as the word *Sox*. But almost every English document contain the word *the*. But only sports related documents have the word *Sox*. So we want the *k*-NN distance metrics to weight **meaningful** words like *Sox* more than functional words like *the*.

For text classification, a weight scheme used to make some dimensions (words) more important than others is known as: TF-IDF

$$tf \cdot idf(w_i,d) = tf(w_i,d) \cdot idf(w_i)$$

$$tf(w_i,d) = \frac{\#(w_i) \in d}{|d|}$$

$$idf(w_i) = \log \frac{|D|}{\# d \in D \, with \, w_i}$$

Here:
tf: Words that occur frequently should be weighed more.
idf: Words that occur in all the documents (functional-words like the, of etc) should be weighed less.

Using this weighing scheme with a distance metric, knn would produce better (more relevant) classifications.

Another way to vary the importance of different dimensions is to use: Mahalanobis Distance

$$D(\vec{x},\vec{y}) = \sqrt{(\vec{x}-\vec{y})S^{-1}(\vec{x}-\vec{y})}$$

Here S is a covariance matrix. Dimensions that show more variance are weighted more heavily.

2. Identification Trees (ID trees or decision trees)

Algorithm: Build a decision tree by **greedily** picking the "lowest disorder" feature tests. The best split for a set of data *minimizes* the average disorder (more precisely, we want the split that decreases the average disorder the most). We define these terms immediately below.

Training – **Divide** the feature space into boxes that have uniform labels. Split the space recursively along each axis to define a tree. (Note this forms a set of boundaries that 'tile' the plane in terms of perpindiculars.)

NOTE: This algorithm is greedy (local hill climbing) so it does **not** guarantee that the tree will have the minimum total disorder!

The notion of "disorder" is defined using **entropy,** ***H***.

We define the entropy (disorder), following Shannon's definition, of a discrete random variable X that has the probability mass function p, as follows:

$$-\sum_{i=1}^{n} p(x_i)\log_2 p(x_i)$$

So for example, suppose a drawer contains 3 red socks and 7 green socks. Then the entropy of this collection of socks in one drawer is (see the graph on the next page for a plot of this function where there are only two classes and one 'bin'):

$$-3/10 \log_2 3/10 + -7/10 \log_2 7/10 = -0.3(-1.7369) - 0.7(-0.5145) = +0.902570$$

Note that the disorder here is at a maximum when the two kinds of socks are equally distributed; and a minimum when either color is absent (uniform color), so the probability of one possibility is 0, and $-p\log_2 p$ of the other color is 1 x 0=0, so H is 0.

For ID trees, we will need to find the *weighted average* of disorder across a *set* of *classes or 'bins'*. The *average entropy* or *disorder for a split* = Entropy for *each* region (bin) times the fraction of the total data points that are in that region (bin) – a weighted average of the disorder, weighted by the # of data points in each class or bin.

$$\text{Average disorder} = \sum_b \left(\frac{n_b}{n_t}\right) \times \left(\sum_c -\frac{n_{bc}}{n_b}\log_2\left(\frac{n_{bc}}{n_b}\right)\right)$$

n_b is the total number of samples in branch b
n_t is the total number of samples in all branches
n_{bc} is the total of samples in branch b of class c

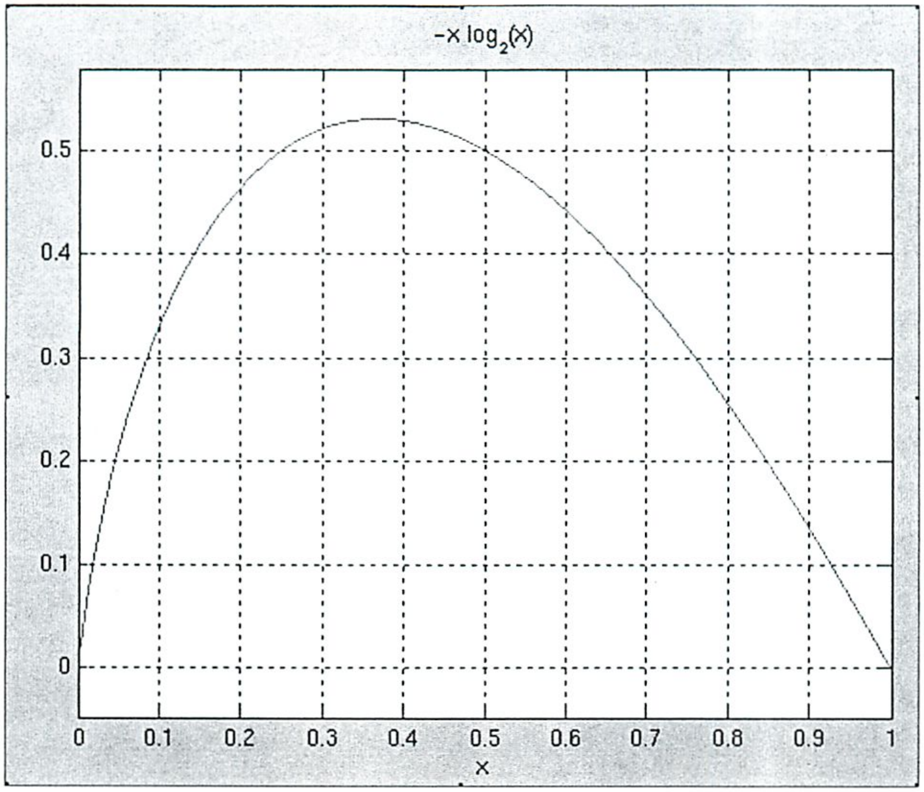
Let's practice calculating this. A simple example with 3 bins (classes), and 2 possibilities, + or O:

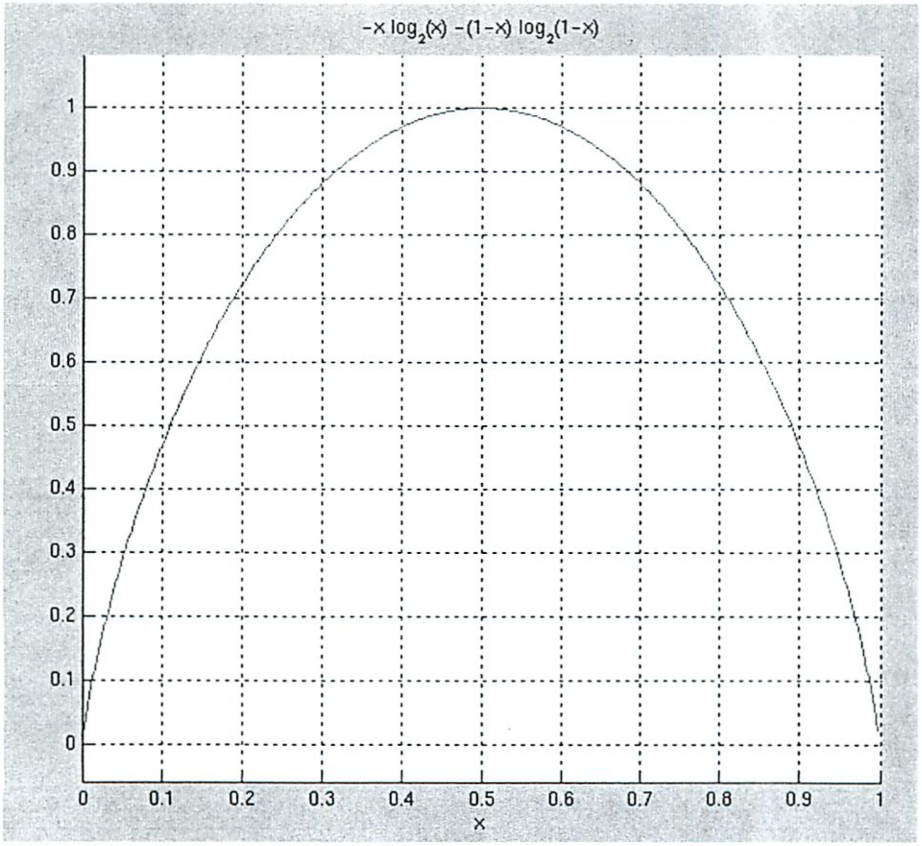
+ O O O + + O	+ + O O O +	+ O + + + + +

We calculate the entropy H in each of the three classes:
Class 1: 3 +, 4 O, 8 total, so + probability is 3/8 = 0.375, so from our 2nd graph: H_1= 0.95
Class 2: 3 +, 2 O, 6 total, so + probability is 3/6 = ½, so H_2= ½ + ½ = 1.0
Class 3: 7 +, 1 O, 8 total, so + probability is 7/8 =
Now we compute the *weighted average* of these three H values. There are 8+6+8 objects in all, or 22, so:

$$8/22(H_1)+6/22(H_2)+8/22(H_3) = 0.36(0.95)+0.18(1)+ 0.36(0.54)= 0.34+0.18+0.19=0.71$$

This is the *average disorder* of this particular split into 3 classes. This is the number used to 'drive' the algorithm, which attempts to find the split that achieves the *lowest* average disorder.





See also the table of binary entropy values a few pages later on.

Example formulas.

The disorder equation for a test with two branches, left and right, (l, r), with each branch having 2 (binary) classes or bins.

Let a = count of class 1 on the left side; b = count of class 2 on the left side;

Let c = count of class 1 on the right side; d = count of class 2 on the right side

$a + b = l \quad c + d = r; r+l = T$

$$\text{Disorder} = \frac{l}{T}\left(\left[-\frac{a}{l}\log_2\frac{a}{l}\right]+\left[-\frac{b}{l}\log_2\frac{b}{l}\right]\right)+\frac{r}{T}\left(\left[-\frac{c}{r}\log_2\frac{c}{r}\right]+\left[-\frac{d}{r}\log_2\frac{d}{r}\right]\right)$$

For a test with 3 branches, and 2 binary class outputs (this is the formula for the example we explicitly did earlier):

$$\text{Disorder} = \frac{b_1}{T}H\left(\frac{a}{b_1}\right)+\frac{b_2}{T}H\left(\frac{c}{b_2}\right)+\frac{b_3}{T}H\left(\frac{e}{b_3}\right)$$

a = count of class 1 on branch 1 $\quad$ b = count of class 2 on branch 1

c = count of class 1 on branch 2 $\quad$ d = count of class 2 on branch 2

e = count of class 1 on branch 3 $\quad$ f = count of class 2 in branch 3

$a+b = b_1 \quad c + d = b_2 \quad e + f = b_3$

Homogeneous Partitioning Trick

A time-saving heuristic shortcut to picking the lowest disorder test.

1. Pick tests that break the space into a *homogeneous portion* and a *non-homogeneous* portion
2. Pick the test that partitions out the **largest** homogeneous portion; that test will most likely have the lowest disorder.

Caution! when the homogeneous portions are **about the same size**, you should compute the full disorder value. This is where this shortcut might break down!

ID trees and Prediction – Test features of a query feature vector according to the identification tree generated during training, return the class at the leaf of the tree.

Relevant features? Irrelevant features are ignored because have large disorders.

Whose Razor? Occam's: The world is inherently simple. Choose the smallest consistent tree.

Why greedy? Finding the simplest tree is computationally intractable; so we use a greedy search using minimum average disorder as a heuristic.

Table of common Binary Entropy values $-\frac{1}{3}\lg_2\left(\frac{1}{3}\right) - \frac{2}{3}\lg_2\left(\frac{2}{3}\right)$

Note: because $H(x)$ is a symmetric function, i.e. $H(1/3) = H(2/3)$, fractions > 1/2 are omitted.

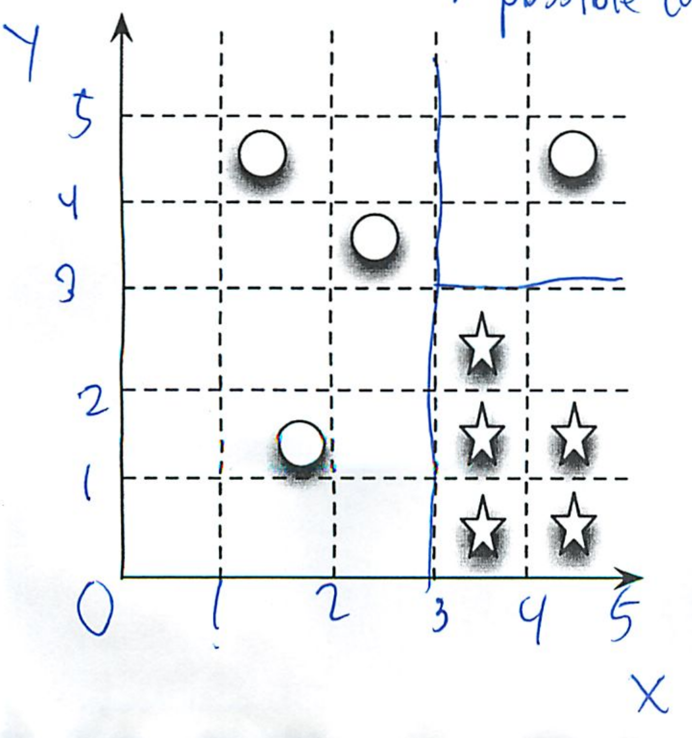
/3 to /9				/10 to /13			
numerator	denominator	fraction	H(fraction)	numerator	denominator	fraction	H(fraction)
1	3	0.33	0.92	1	10	0.10	0.47
2	3	0.67	0.92	2	10	0.20	0.72
1	4	0.25	0.81	3	10	0.30	0.88
2	4	0.50	1.00	4	10	0.40	0.97
1	5	0.20	0.72	1	11	0.09	0.44
2	5	0.40	0.97	2	11	0.18	0.68
3	5	0.60	0.97	3	11	0.27	0.85
1	6	0.17	0.65	4	11	0.36	0.95
2	6	0.33	0.92	5	11	0.45	0.99
3	6	0.50	1.00	1	12	0.08	0.41
1	7	0.14	0.59	2	12	0.17	0.65
2	7	0.29	0.86	3	12	0.25	0.81
3	7	0.43	0.99	5	12	0.42	0.98
1	8	0.13	0.54	1	13	0.08	0.39
2	8	0.25	0.81	2	13	0.15	0.62
3	8	0.38	0.95	3	13	0.23	0.78
4	8	0.50	1.00	4	13	0.31	0.89
1	9	0.11	0.50	5	13	0.38	0.96
2	9	0.22	0.76	6	13	0.46	1.00
3	9	0.33	0.92				
4	9	0.44	0.99				

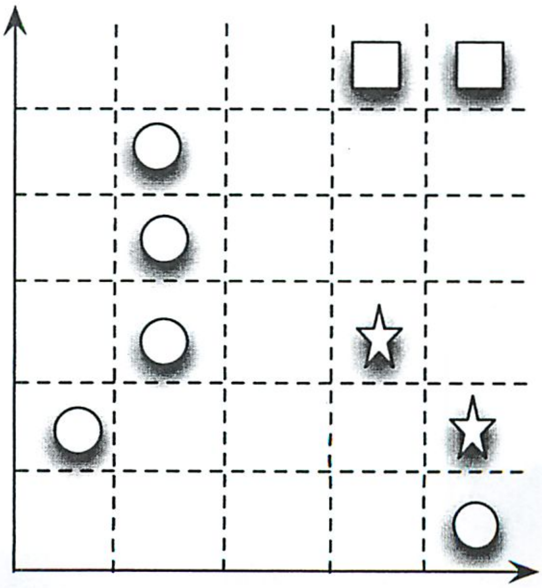
H is for both terms

Only for 2 types of symbols

Try some sample 'cuts' in these two figures….which is the best single cut(s) in each? Why? And the next cut?

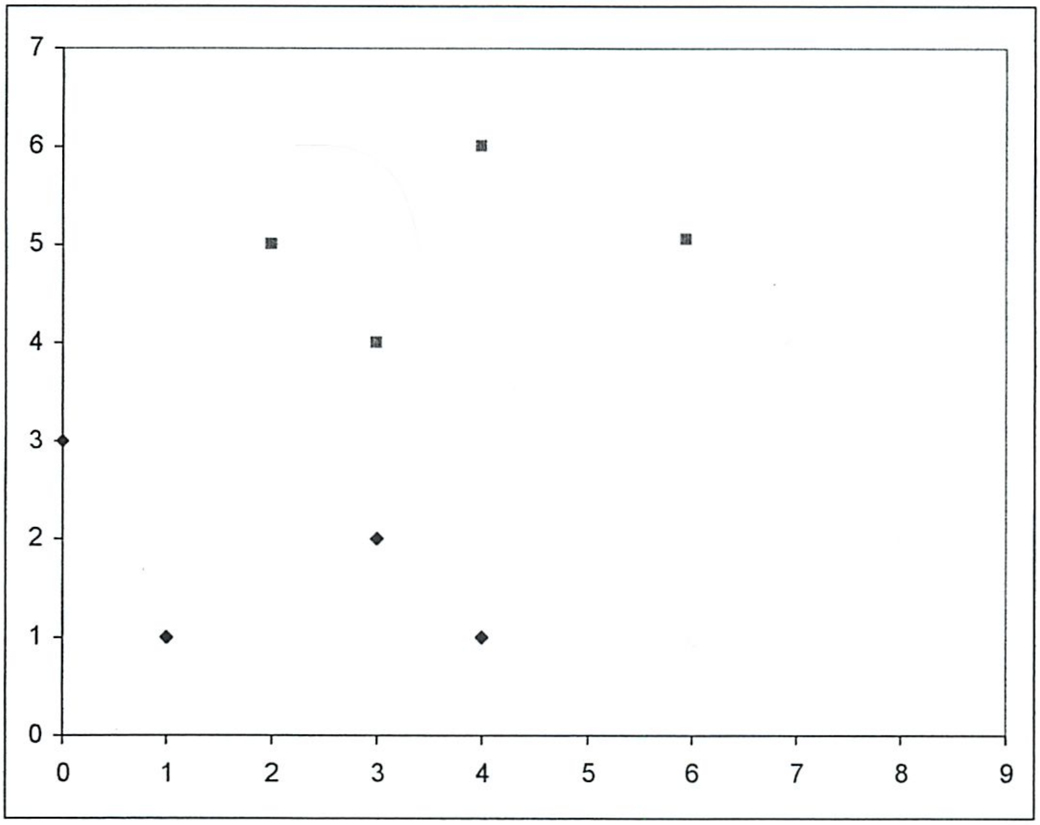
4 possible cuts

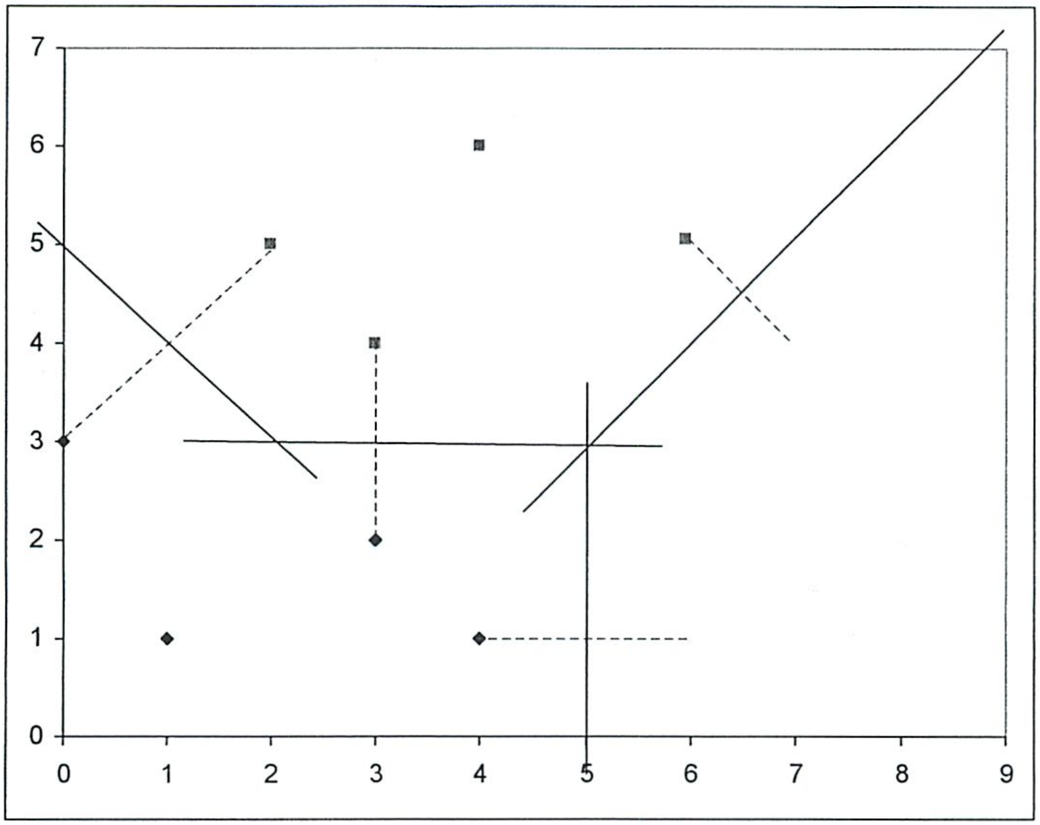




6.034 Recitation October 20: Nearest Neighbors, Drawing decision boundaries

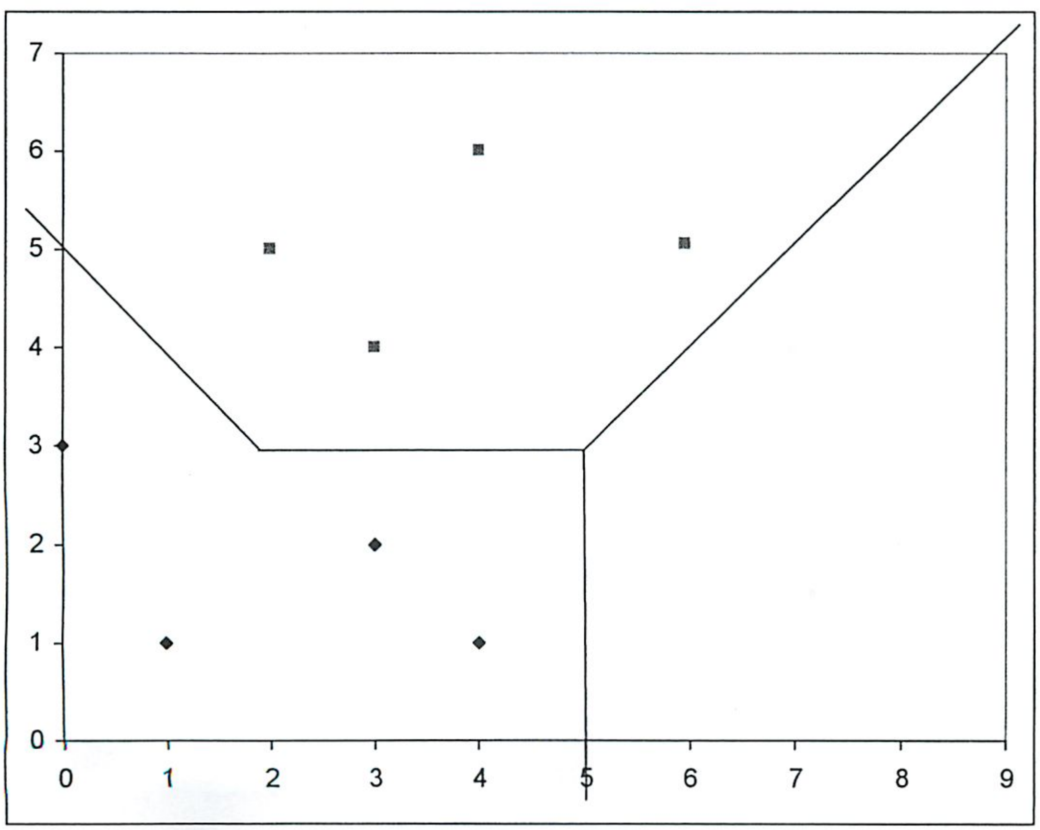
Boundary lines are formed by the intersection of perpendicular bisectors of every pair of points. Using pairs of closest points in different classes gives a good enough approximation. (To be absolutely sure about the boundaries, one would draw perpendicular bisectors between each pair of neighboring points to create a region for each point, then consolidate regions belonging to the same class, i.e., remove the boundaries separating points in the same class. This technique is unnecessary for our purposes.)





Construct lines between closest pairs of points in different classes.

Draw perpendicular bisectors.



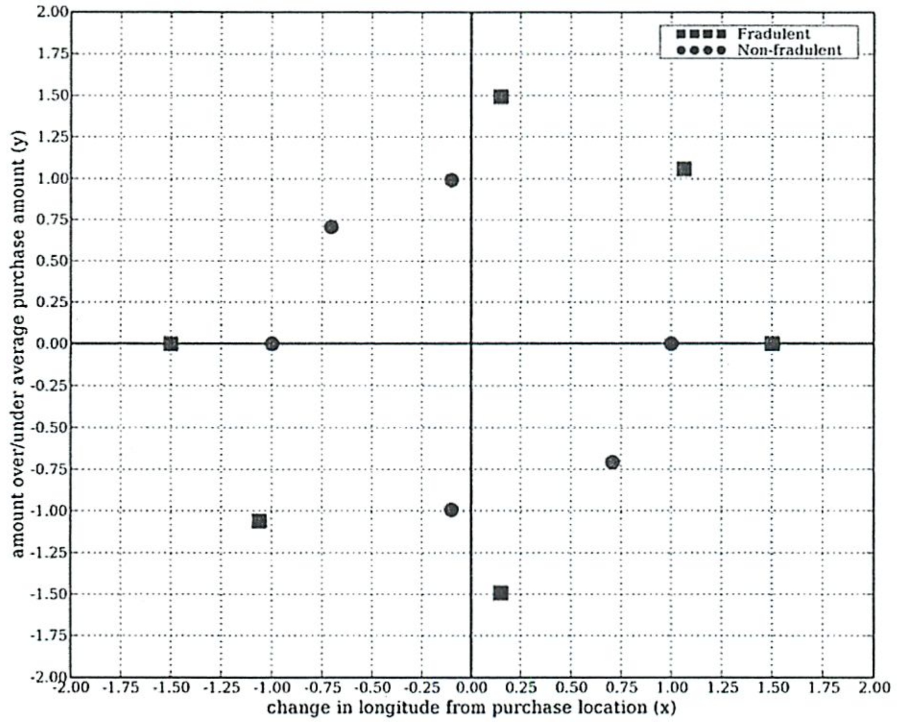
End bisectors at intersections; extend beyond axes (to infinity).

10/20/11 Nearest Neighbors Practice Problem 1

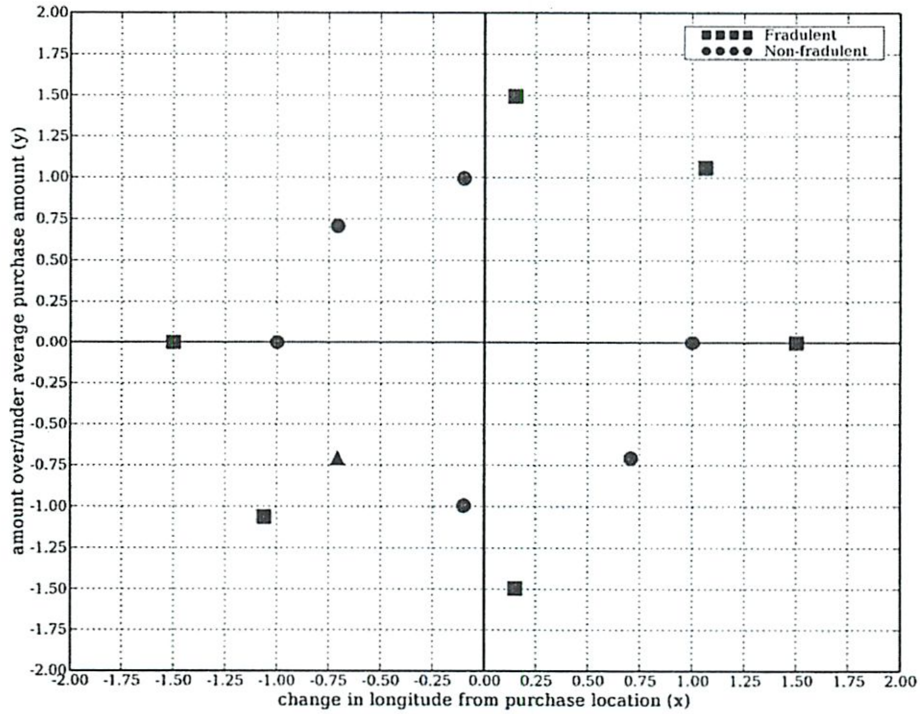
Lucy has been working hard for the credit card companies to detect fraud. They have asked her to analyze a number of classification methods to determine which one is best suited to their problem. The two quantities that they have provided her are the change in longitude from the purchase location to the registered address and the amount that the purchase is over or under the average purchase that the customer usually makes.

Part A: Nearest Neighbors (15 pts)

Lucy decides to use nearest neighbors to solve this problem and plots the fraudulent / non-fraudulent data. Squares are fraudulent and circles are non-fraudulent. Sketch the resulting decision boundary on the figure below.



It is the end of the month and Lucy's boss comes over with new data hot off the presses (the triangle). He wants Lucy to analyze whether or not the new charge is fraudulent.



What is the nearest neighbor classification of the new charge, fraudulent or non-fraudulent?

She's not too sure about this classification and decides to rerun it using k-nearest neighbors for k=3 and then for k=5. Is the charge fraudulent for these values of k?

K= 3:

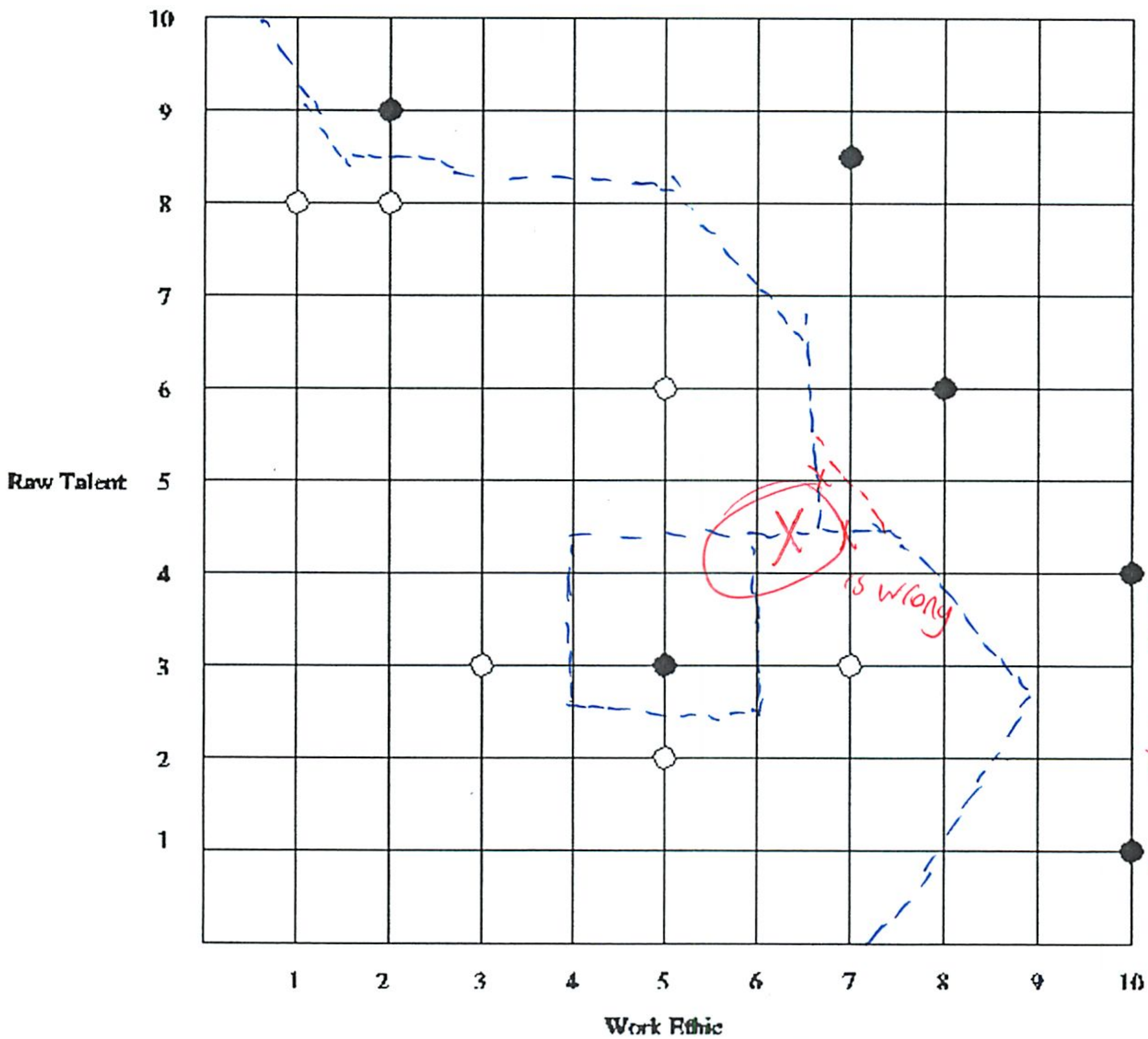
K= 5:

6.034 Recitation Thursday, October 20, 2011

Practice Problem 2: k-Nearest Neighbors

The 6.034 staff has decided to launch a search for the newest AI superstar by hosting a television show that will make one aspiring student an *MIT Idol*. The staff has judged two criteria important in choosing successful candidates: work ethic (W) and raw talent (R). The staff will classify candidates into either potential superstar (black dot) or normal student (open circle) using a nearest-neighbors classifier.

On the graph below, draw the decision boundaries that a 1-nearest-neighbor classifier would find in the R-W plane.



Can have just a box

~~black area must connect to other black areas~~

I forgot some connections

no T junctions just polygons

Identification trees Problem 1 (same credit card problem as k-NN above)

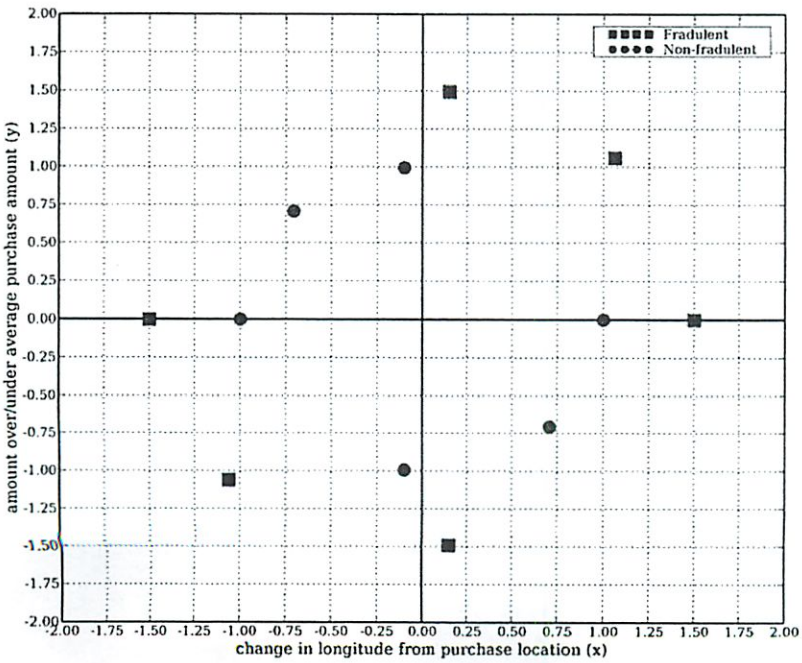
B1 The boundaries (18 pts)

Lucy now decides that she'll try to use identification trees on the data. There are three likely candidates for splitting the data: x=0.0, x=-1.01 and x=1.01. Note that the -1.01 and 1.01 values lie half-way between a square and a circle with nearby x values. Compute the average disorder for the decision boundary x=1.01. Your answer may contain logarithms.

Compute the average disorder for the decision boundary x=0.0. Again, your answer may contain logarithms.

Which of the two decision boundaries, x= 0.0 and x=1.01, is added first?

Sketch all of the decision boundaries on the figure below. Assume that x= 0.0 and x=1.01, in the order you determined above, are the first two decision boundaries selected (this may or may not be true, but assume it is).



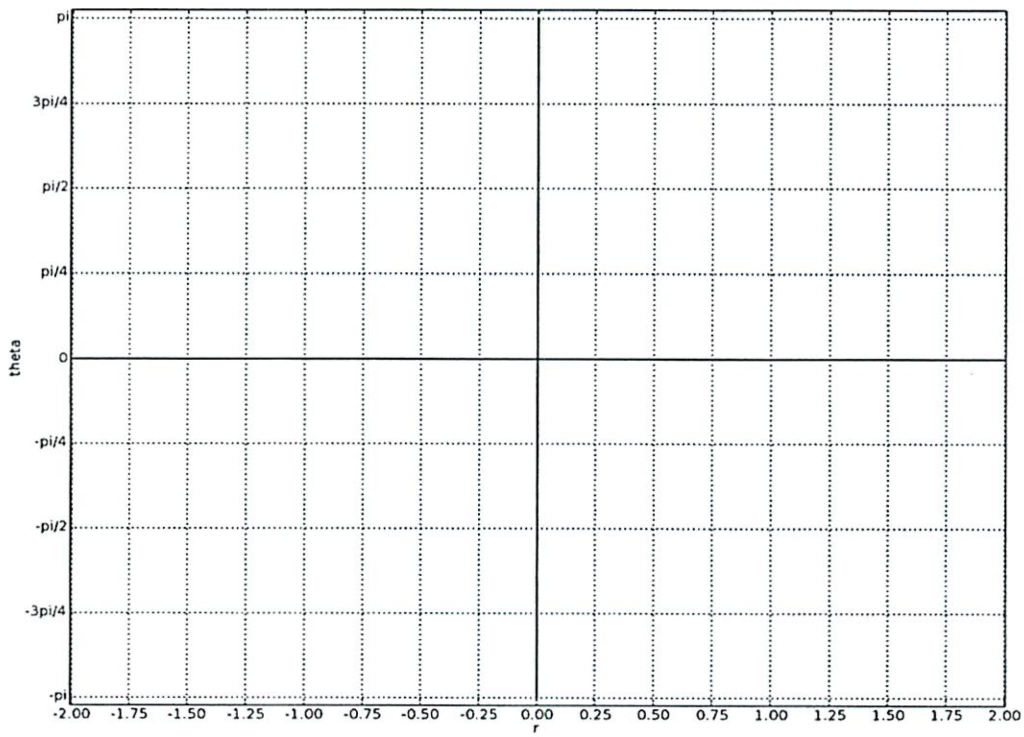
B2 The identification tree (7 pts)

Draw the identification tree corresponding to your decision boundaries.

What is the classification of the new charge (triangle)?

Part C: Polar coordinates (10 pts)

Lucy gets smart and decides to try a different space for each of the points. That is, she converts all of the points to polar coordinates. Sketch the data below. **You may assume that *r* value of each point is very close to a multiple of 0.25 and that the theta value of each point is very close to a multiple of pi/4.**



How many decision boundaries do we need in this case?

Draw the resulting identification tree and sketch the decision boundary on the graph above.

Identification Trees Practice Problem 2

Part B1 (2 Points)

Now, leaving nearest neighbors behind, you decide to try an identification-tree approach. In the space below, you have two possible initial tests for the data. Calculate the average disorder for each test. Your answer may contain $\log_2$ expressions, but no variables. The graph is repeated below.

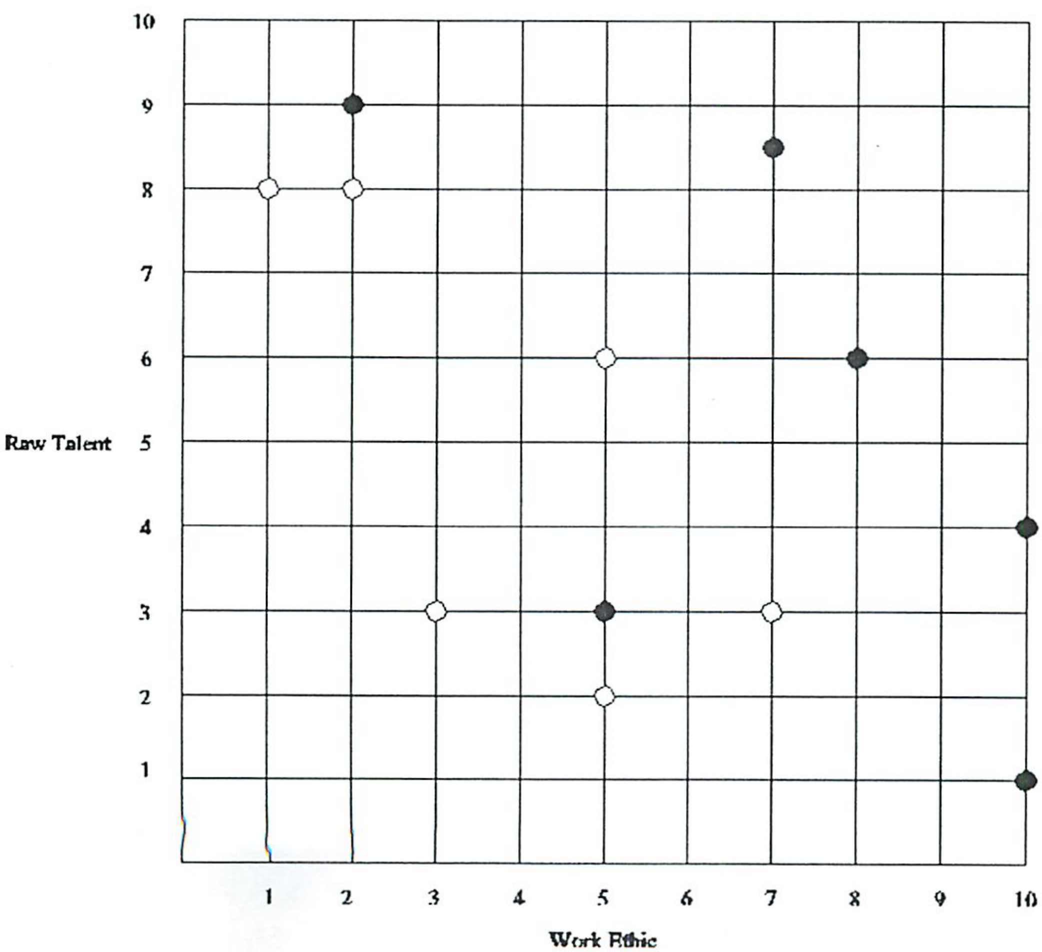
Test A: $R > 5$:

Test B: $W > 6$:

Part B2 (2 Points)

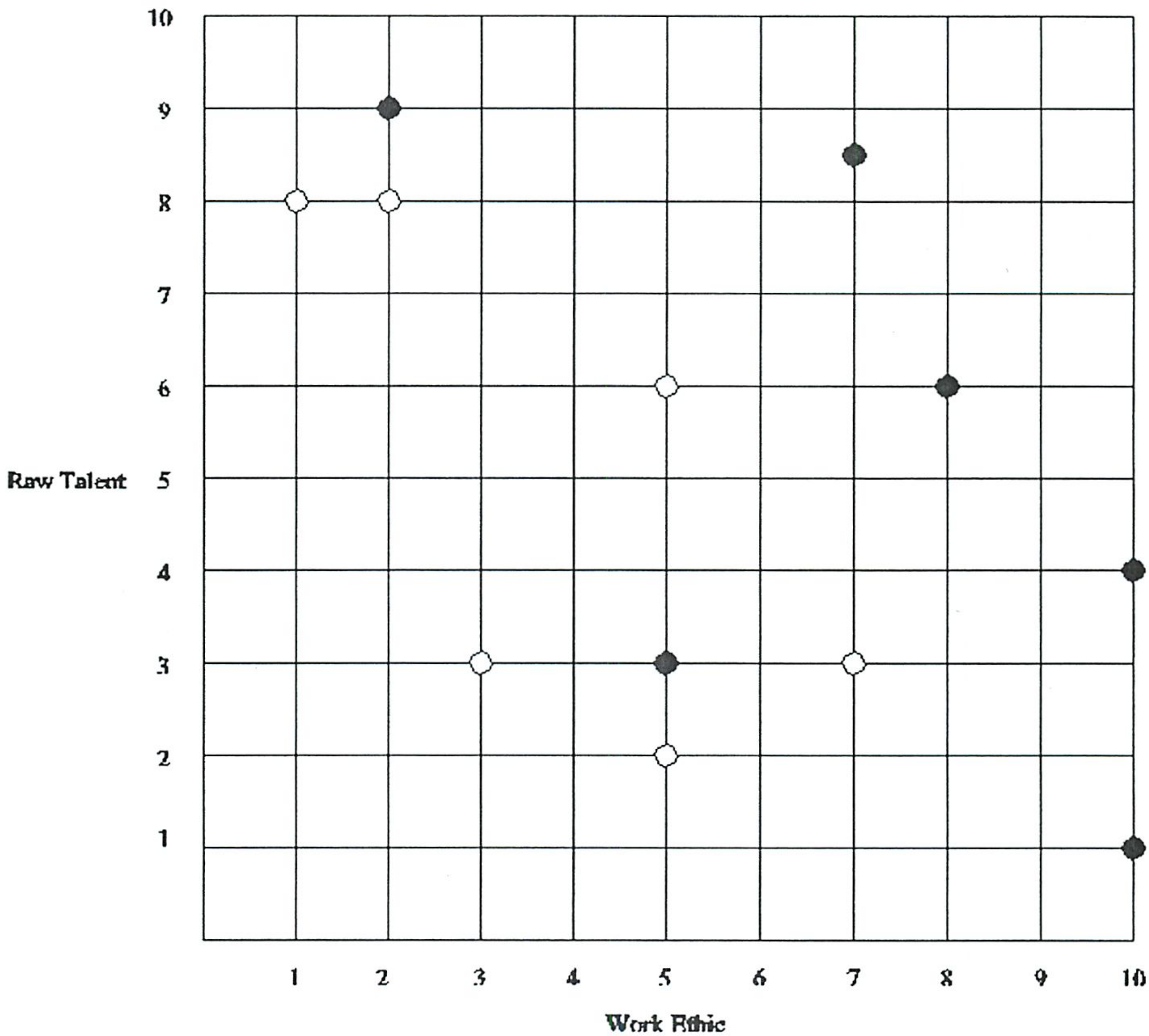
Now, indicate which of the two tests is chosen first by the greedy algorithm for building identification trees.

We include a copy of the graph below for your scratch work.



Part C: Identification Trees (4 Points)

Now, assume R > 5 is the first test selected by the identification-tree builder (which may or may not be correct). Then, draw in all the rest of the decision boundaries that would be placed (correctly) by the identification-tree builder:



6.034 Mega Recitation

10/21

Quiz next week
- this recitation is not on the quiz

Today: Nearest neighbors
Identification trees

~~Nearest neighbor~~

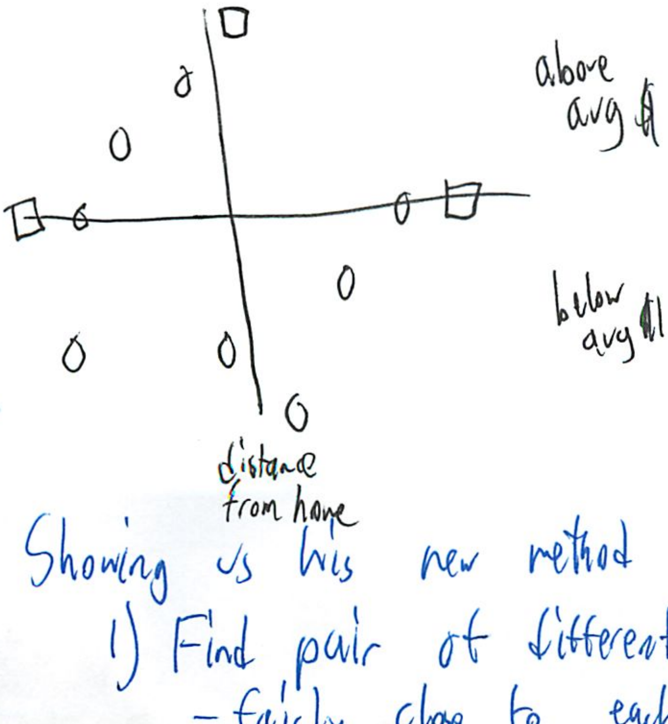
Silver Stars

* Knn shortcut
* $\frac{L}{N}\left(-\frac{L\oplus}{L}\lg\frac{L\oplus}{L} - \frac{L\ominus}{L}\lg\frac{L\ominus}{L}\right) + \frac{R}{N}\left(-\frac{R\oplus}{R}\lg\frac{R\oplus}{R} - \frac{R\ominus}{R}\lg\frac{R\ominus}{R}\right)$
* 

* Transform wisely
* Tie breakers
* No curved lines

Nearest neighbors

Draw in boundry lines

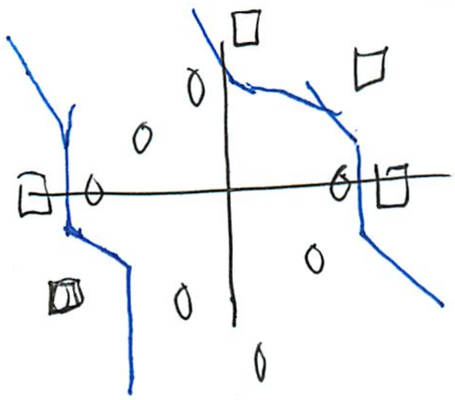


□ = fradulent
O = Legitimate

Showing us his new method knn
1) Find pair of different items
   - fairly close to each other

②

2. Draw arrow up and down until circle ~~or~~ square you just drew between are not the closest



Go ∞ when off graph

I drew it a bit wrong

Then they ask you qu

Like we have a new tx'm Δ

What is the 1-nearest neighbor?

- look at the lines

3-nearest neighbors

- look at 3 closest
- What type are the majority of the symbols

This is not a ~~standard~~ really smart real life idea

③

Decision trees

Formula for entropy

L = # Left   Symbols: $\oplus$ $\ominus$

N = #

R = # Right

$L\oplus$ = # $\oplus$ left

left of lines ↘   ↙ # that are $\oplus$ on L

$$\frac{L}{N}\left(-\frac{L\oplus}{L}\lg\frac{L\oplus}{L}-\frac{L\ominus}{L}\lg\frac{L\ominus}{L}\right)+$$

$$\frac{R}{N}\left(-\frac{R\oplus}{R}\lg\frac{R\oplus}{R}-\frac{R\ominus}{R}\lg\frac{R\ominus}{R}\right)$$

0 entropy - perfect

50/50 entropy is worse case - just flip coin

← gives you value for left and right

← look on table

then add to get value that can compare w/ other lines

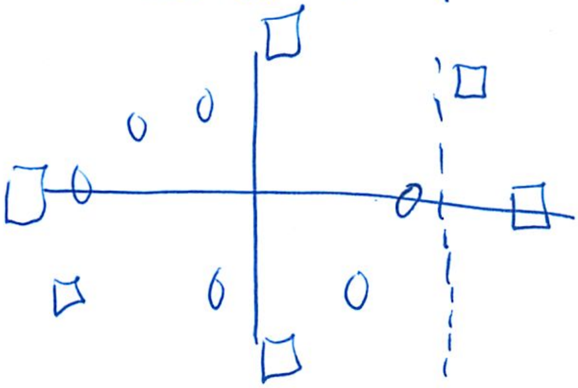
Vertical and horizontal lines only

- no diagonal

No curve

(4)

Draw decision boundry we were told



$+ = O$

$- = \square$

$$\frac{L}{N} = \frac{10}{12}$$

$$\frac{10}{12}\left(-\frac{6}{10}\lg\frac{6}{10}-\frac{4}{10}\lg\frac{4}{10}\right)+\frac{2}{12}\left(\underbrace{-\frac{0}{2}\lg\frac{0}{2}}_{0}-\underbrace{\frac{2}{2}\lg\frac{2}{2}}_{0}\right)$$

$$-\frac{1}{2}\lg\frac{3}{5}-\frac{1}{3}\lg\frac{2}{5}$$

Doing this to check which one is the best

Now try y axis 

$$\frac{6}{12}\left(-\frac{4}{6}\lg\frac{4}{6}-\frac{2}{6}\lg\frac{2}{6}\right)+$$

$$\frac{6}{12}\left(-\frac{2}{6}\lg\frac{2}{6}-\frac{4}{6}\lg\frac{4}{6}\right)$$

$$-\frac{2}{3}\lg\frac{2}{3}-\frac{1}{3}\lg\frac{1}{3}$$

5

Which is more?

L can use a calculator

First one is better

- since it seperates two out

Draw that one

Now quiz says draw at 0

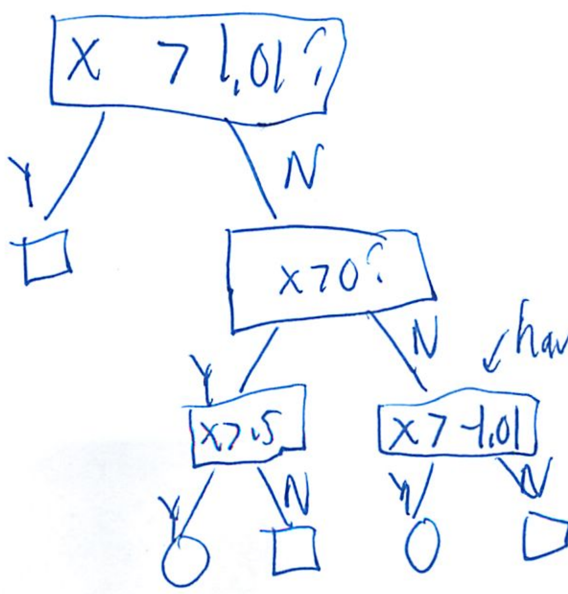
L not the best idea

Important to know how to break ties

L they will tell you

- different each time
- follow directions

ID Tree



have a tie, says vertical lines first

6

Now can ask about triangle again?

- with the new system

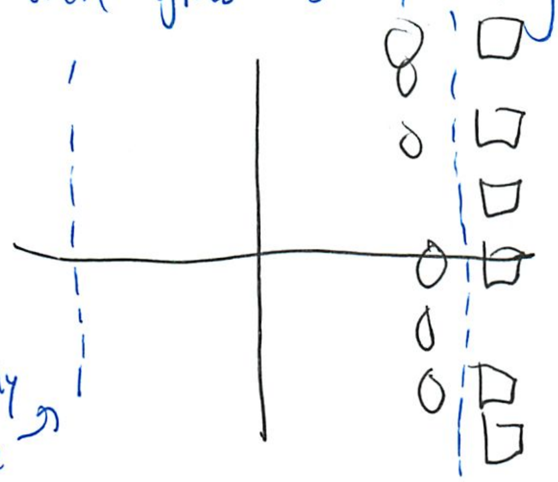
Could also convert to polar coords.

└ convert wisley

- usually polar, but not always
- can shift up + down and then do polar

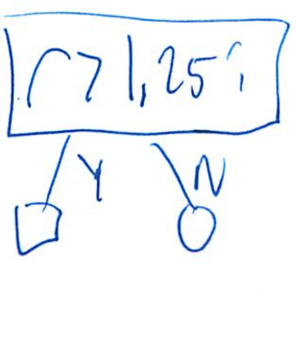
Quiz says just do polar

- even gives us rounding



technically a line here too

So draw line



6.034 · Lecture 10/22

Genetic algorithms

☐ Naive Darwinism

☐ Mimicking ♪

☐ Problems

* Diversity essential

* Ask where the credit lies

We still don't have a computer as smart of us

Can we build one to learn to do so

Neural nets (~~yesterday~~ last lecture)

But some issues

- local maxing
- instability
- overfitting - it does poorly on them
- Coding - need to implement data

Neural nets are functional approximations

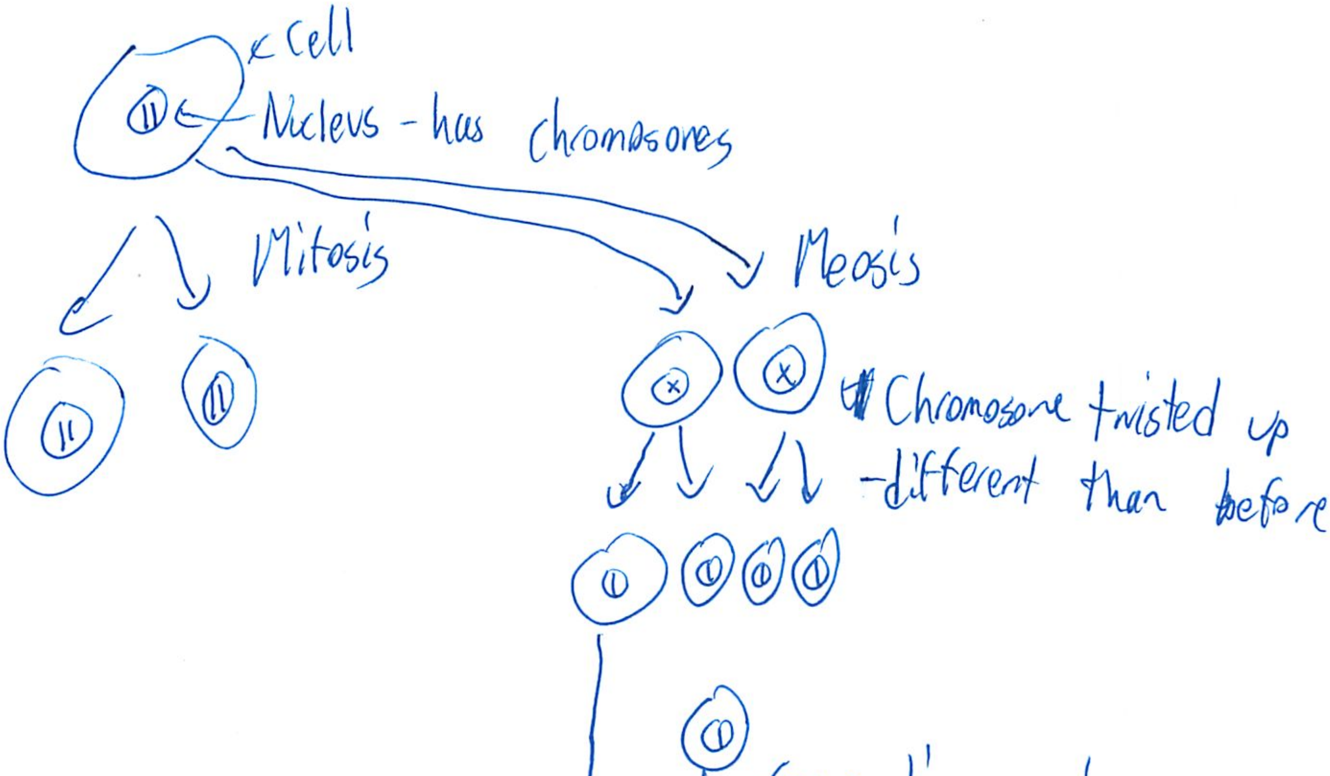
$$\bar{z} = f(\bar{x})$$

Sometimes functional approx is not best
[neural nets]
- but will be on next exam

Wed: Exam up to nearest neighbor

Can we build a system that evolves to our will

Sexual Reproduction

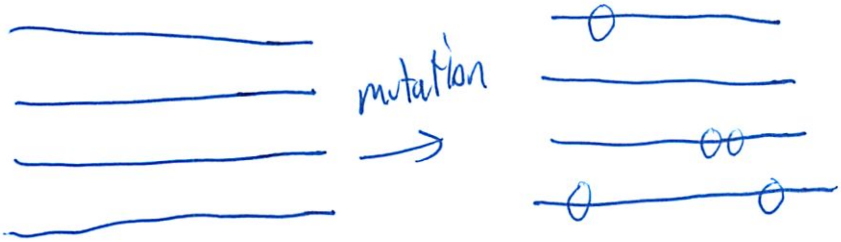


Happens in males
Lots of randomness - how much crossover?
- how much mutation?

3

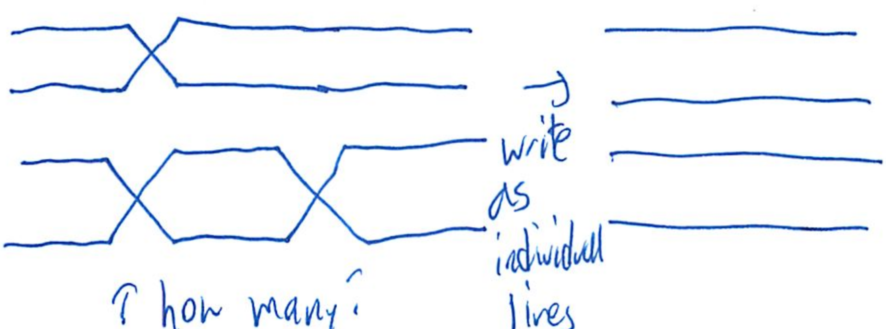
Have a chromonsore code

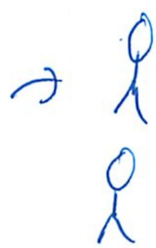
0 1 1 0 0 1 1 1 0 1



? how many?
Close together?

Cross over step





? how many?
how long

This is all a big mystery
How would you code this in?
L The big problem

Fitness		Fitness → Prob
88	→	P_{88}
75		P_{75}

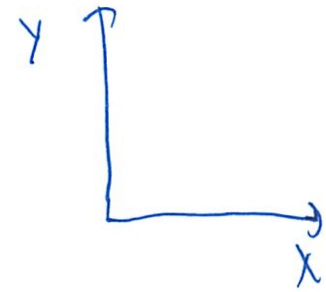
repeats for
next generation

(4)

Mutation is like hill climbing
2 like nural nets

¿But abat cross over

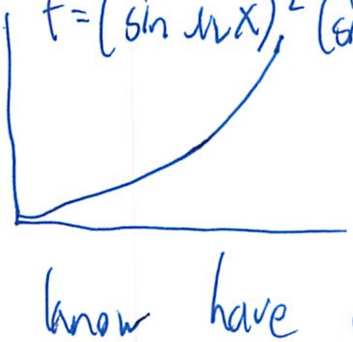
<u>Examples</u> Find max of function



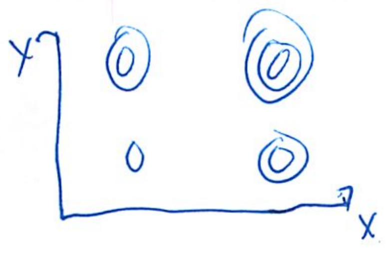
fitness → $f = (\sin wx)^2 (\sin w y)^2$

Want one to be spream maximum

$f = (\sin wx)^2 (\sin w y)^2 \cdot e^{\frac{x+y}{\sigma}}$

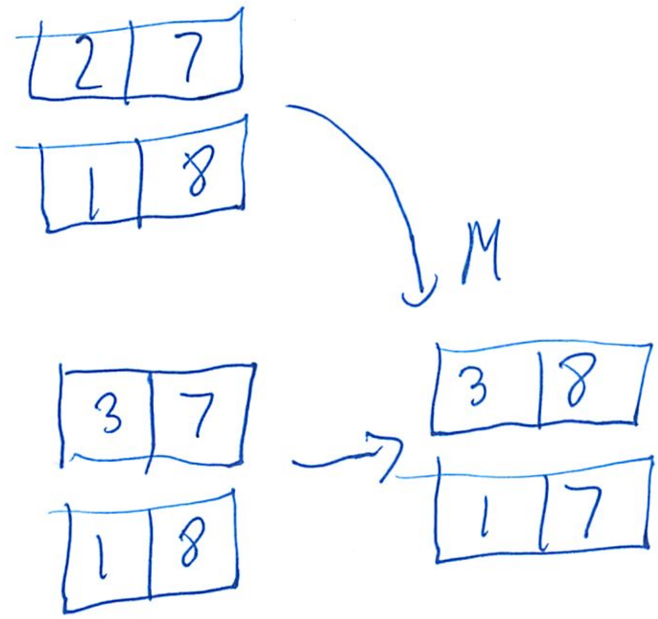


So know have contour map



5

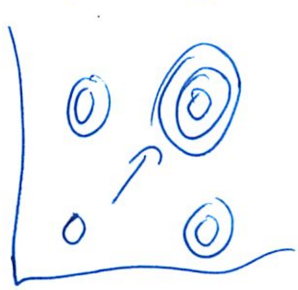
Chromosomes



How should prob of survival be related to who goes to next generation

$$P_x = \frac{\text{fitness}_x}{\sum_i f_i} \quad f = \text{fitness}$$

Has hard time getting off local maxima on each pt

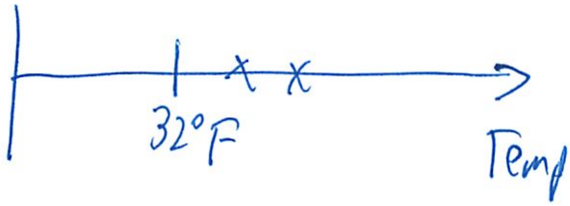


6

So tinker w/ system

approach 2

~~prob~~ Suppose



If chose 0°C instead do more survive?

└ Does measurement ~~math~~ matter?

So instead just write the #s 1 ... N

Then prob is off their ranking not exact values

Rank	
1	P_c ← prob that 1st ranked individual will survive
2	$(1-P_c)P_c$
⋮	
N-1	$(1-P_c)^{N-2}P_c$
N	$(1-P_c)^{N-1}$ ← must take is prob that none of the other guys were selected

all # should add to 1

7

$$X = 1 - P_c$$

$$S = 1 + x + x^2 + \ldots\ldots + x^{n-2}$$

$$Sx = x + \ldots\ldots\ldots + x^{n-2} + x^{n-1}$$

$$S(x-1) = \frac{x^{n-1} - 1}{x - 1} = \frac{(1-P_c)^{n-1} - 1}{1 - P_c - 1}$$

$$= \frac{(1-P_c)^{n-1} - 1}{P_c}$$

$$P_c \, S(x-1) = \frac{(1-P_c)^{n-1} - 1}{P_c} P_c$$

$$= (1-P_c)^{n-1} - 1$$

$$= (1 - P_c^{n-1}) - [1 - P_c^{n-1} - 1]$$

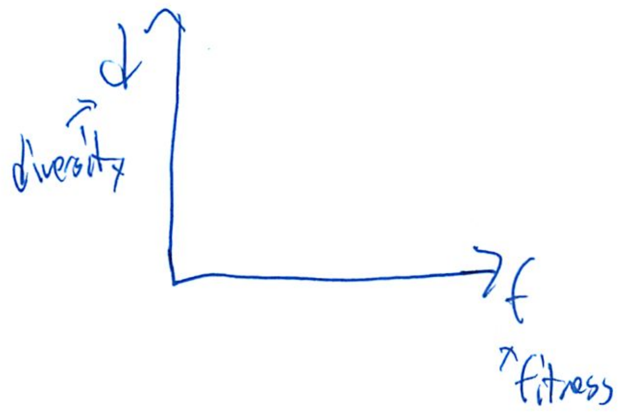
$$= 1 \quad ✓$$

So now rerun w/ ranking - not abs values

So obsessed w/ fitness just staying on local maximum approach #3

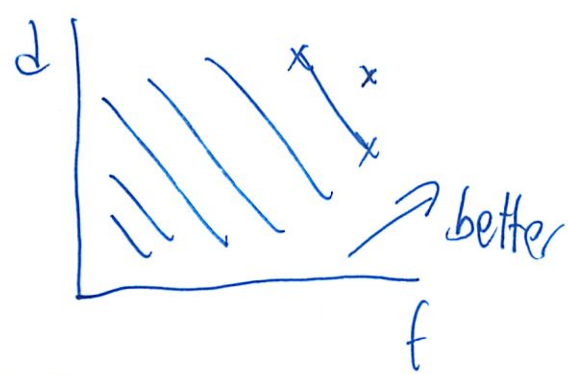
So add some diversity in

are both ranks

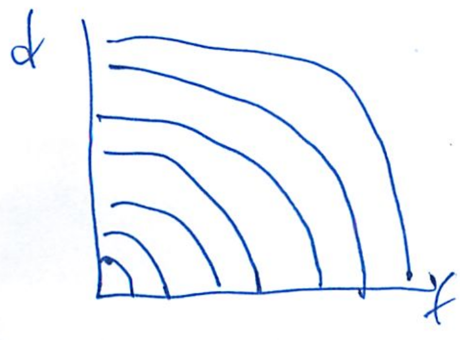


So how do you calculate a diversity rank?
best: very fit and very different
L rare

Can draw isofitness curve



Could also do



9

Not just crawling up hill - but away from each other

Currently it does well (approach 4) in either x or y 

Introduce crossover!

- ~~good~~ In
- ~~take~~ randomly pull over x and y coordinates

Works much better!

So when does this stuff actually work

1) Scheduling

S_1	S_2	S_3	S_4

good begining

S_{1P}	S_{2P}	S_{3P}	S_{4P}

good at end

crossover → good at begining and end

S_1	S_2	S_{3P}	S_{4P}

his company does this

2) Horse Racing

If x and y   x = post position, jockey weight

Then z

If A, and B

Then Z

(10)

If A' and B
~~and~~

Then Z

If A' and Y
~~X~~

Then Z

So evolved a system that was just as good

3. Blocks

- One part size
- one part where connected
- ~~other~~ one part how they move around

Object: to create better + better creatures

Funny video

Ask where credit lies
- 1. in genetic algorithm
  2. in head of Karl Sims
     L he programmed it all
  3. It's the space
     L any mechanism produces something interesting

6.034 Recitation 10/28

1. ID trees
2. Neural nets
    1. Simple input-output
    2. 2 layers + Back prop

Quiz back at end

Breakpts by problem

	P1	P2	P3
5	40	40	20
	35	35	15
4	30	31	10

Back to credit card fraud problem - Part 1 packet

1a) $n = 12$ $\quad x \overset{+}{=} 1.01$

10 $\quad$ 2 □

H_1 $\quad$ $H_2 = 0$

↓

4 □

6 ○

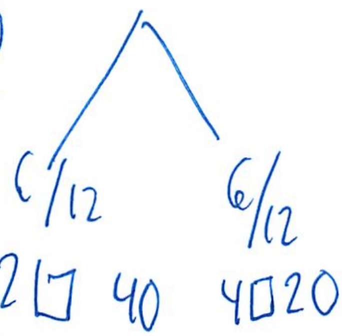
↓

$$\frac{10}{12}\underbrace{\left[-\frac{2}{5}\lg_2\frac{2}{5} - \frac{3}{5}\lg_2\frac{3}{5}\right]}_{H_1} + 0$$

$\frac{5}{6} \times 0.97 \to .8033$

(2)

b) $\underline{X = 0}$



$x = -1.01$ didn't do

H_1 H_2

$$\frac{6}{12}\underbrace{\left[-\frac{2}{6}\lg_2\frac{2}{6}-\frac{4}{6}\lg_2\frac{4}{6}\right]}_{H_1}+\frac{6}{12}\left[\quad\right]$$

$\frac{1}{2}\cdot 0.92 + \frac{1}{2}\cdot 0.92$

0.92

c) Now what to add a vertical division

Can use table to make calculation

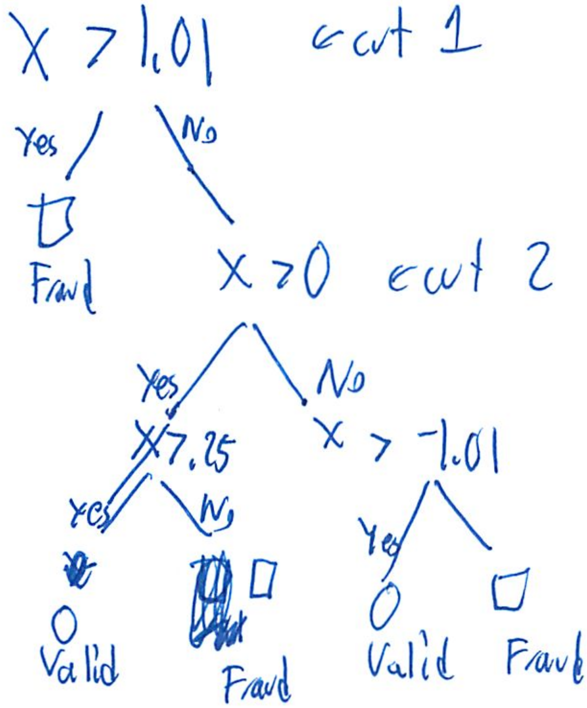
if does both parts no need to do addition

.25 is last cut

(3)

Now we can make rules

Order matters



Part C Polar Coords

Seperating data in nice clean way is always nice

You can niceley see a circle works well

└ still credit card fraud

So transform to polar

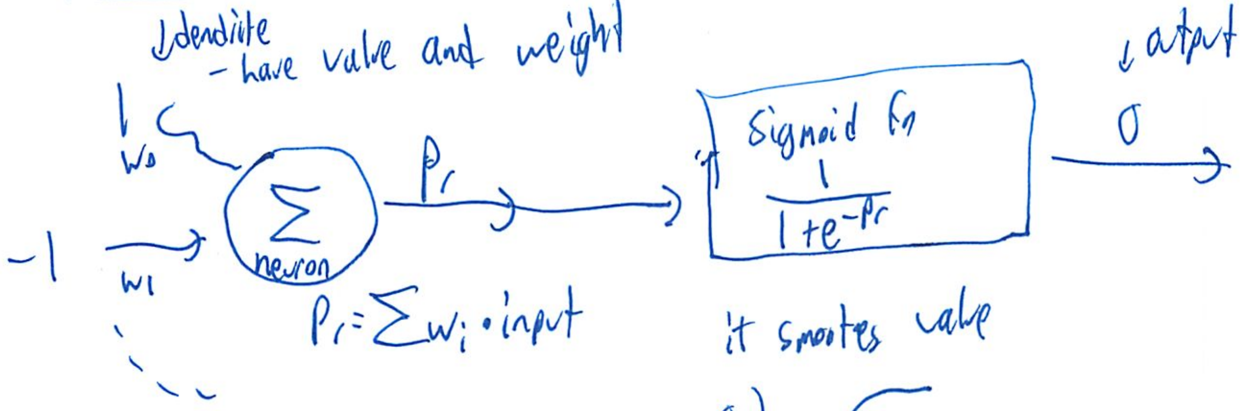
So we can still make straight lines

Happens to be easy divison here

Exams often ask ya to do some transform

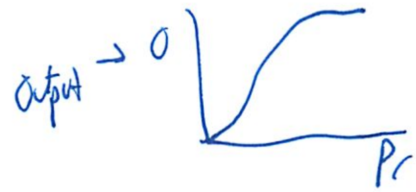
(4)

Neural Networks



$P_r = \sum w_i \cdot input$

it smoothes value



Wire together

To get input, output pair, ~~change weight to train~~

change weight to train

- only a PC can actually do
- twiddle weights till output matches input

1. Forward Propagation
   - w/ current weights
2. Back propagate
   - go back though network
   - to see which weight we should twiddle

5

d = desired output

O_f = output from forward propagation

twiddle weights if $\underbrace{d - O_f}_{\text{measure of how close}}$

- actually use something slightly difference
- a Performance function P

$$P = \frac{1}{2}(d - O_f)^2$$

↑ will have particularly nice deriv of P w/ respect to O_f

---

page 3

inputs

-1 $w_0 = 0$

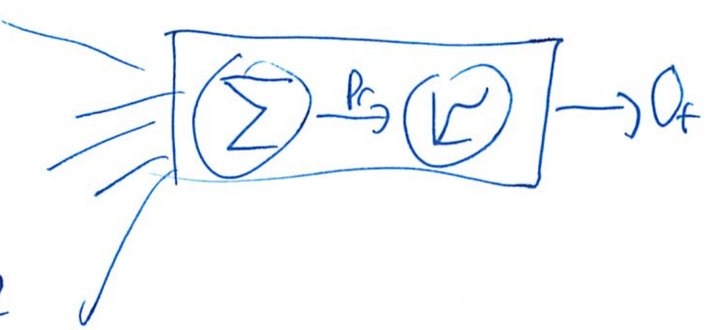
-1 $w_1 = 2$

1 $w_2 = 2$

1 $w_3 = 3$

.5 $w_4 = 2$

0 $w_5 = 1$



(6)

So desired is 1.0

but what does network actually compute?

Calculate $\frac{\partial P}{\partial W_i}$ ← want to see how P changes when w changes ← correct

- if partial is big - use that weight to twiddle
- since changes output a lot
- like gradiant assent

For each W_i calc W_i'

$$W_i' = W_i + \Delta W_i$$

↑ related to partial deriv

Read Neural net math
- on homepage, near play list

We got .9, wanted 1

Need $\frac{\partial P}{\partial W_i}$

(7)

Go backwards through network

For Each piece

$$P = -\frac{1}{2}(d-o)^2$$

$$\frac{\partial P}{\partial o_f} = d - o$$

[so that is after sigmoid

Now must go backwards here too

$$\frac{\partial o_f}{\partial p_r} = o_f(1-o_f)$$ after a lot of math - see notes

Then one more piece - undo summation

$$\frac{\partial p_r}{\partial w_i}$$ ~~multiply each~~

So multiply all 3 together

$$\frac{\partial p_r}{\partial w_i} \times \frac{\partial o_f}{\partial p_r} \times \frac{\partial P}{\partial o_f}$$

cancel

$$= \frac{\partial P}{\partial w_i}$$

8

Since P_r is linear combo $\quad P_r = \sum W_i \cdot input_i$

- W drops out when take deriv

$input_r = i_R$

each of input weights

So

$$= i_R \times O_f(1-O_f) \times (d-O_f)$$

need to do forward first

So can compute this

But how do you change weights?

First will rename $\underbrace{O_f(1-O_f) \times (d-O)} = \delta_f$

So $\Delta W_i = \alpha \cdot \delta_f \cdot input\ i$

magical

Constant

α

Called

learning rate

Fixed upfront

~2 to 1000

here $\alpha = 100$

This is called gradient accent,

Trying to ↓ d-o to 0

α Controls if gradual ← low

large, but over jumps ← high
↳ oscilates

- Controlled by external programmer

How do you set weights initally?

- Bad to set them all the same

So do all this for each input

- See table p4

$$O_f \cdot (1-O_f) \cdot (d-O_f) = \delta$$

So $.9 \cdot .1 \times .1 = .009$

This gets plugged in for each weight calc
the same

$$\Delta W_i = \alpha \cdot \delta_f \cdot input_i$$

10

~~At weight is 0 nothing will change~~

Calc new weight values for each

for W_0 $100 \cdot .099 \cdot -1 = -.9$

Part 1 10/27

2. Identification Trees (ID trees or decision trees) - Recitation 7, part 1, 10/27/11

Algorithm: Build a decision tree by **greedily** picking the "lowest disorder" feature tests. The best split for a set of data *minimizes* the average disorder (more precisely, we want the split that decreases the average disorder the most). We define these terms immediately below.

Training – **Divide** the feature space into boxes that have uniform labels. Split the space recursively along each axis to define a tree. (Note this forms a set of boundaries that 'tile' the plane in terms of perpindiculars.)

NOTE: This algorithm is greedy (local hill climbing) so it does **not** guarantee that the tree will have the minimum total disorder!

The notion of "disorder" is defined using **entropy, *H***.

We define the entropy (disorder), following Shannon's definition, of a discrete random variable X that has the probability mass function p, as follows:

$$-\sum_{i=1}^{n} p(x_i)\log_2 p(x_i)$$

So for example, suppose a drawer contains 3 red socks and 7 green socks. Then the entropy of this collection of socks in one drawer is (see the graph on the next page for a plot of this function where there are only two classes and one 'bin'):

$$-3/10 \log_2 3/10 + -7/10 \log_2 7/10 = -0.3(-1.7369) - 0.7(-0.5145) = +0.902570$$

Note that the disorder here is at a maximum when the two kinds of socks are equally distributed; and a minimum when either color is absent (uniform color), so the probability of one possibility is 0, and $-p\log_2 p$ of the other color is 1 x 0=0, so H is 0.

For ID trees, we will need to find the *weighted average* of disorder across a *set* of *classes or 'bins'*. The *average entropy* or *disorder for a split* = Entropy for *each* region (bin) times the fraction of the total data points that are in that region (bin) – a weighted average of the disorder, weighted by the # of data points in each class or bin.

$$\text{Average disorder} = \sum_b \left(\frac{n_b}{n_t}\right) \times \left(\sum_c -\frac{n_{bc}}{n_b}\log_2\left(\frac{n_{bc}}{n_b}\right)\right)$$

n_b is the total number of samples in branch b
n_t is the total number of samples in all branches
n_{bc} is the total of samples in branch b of class c

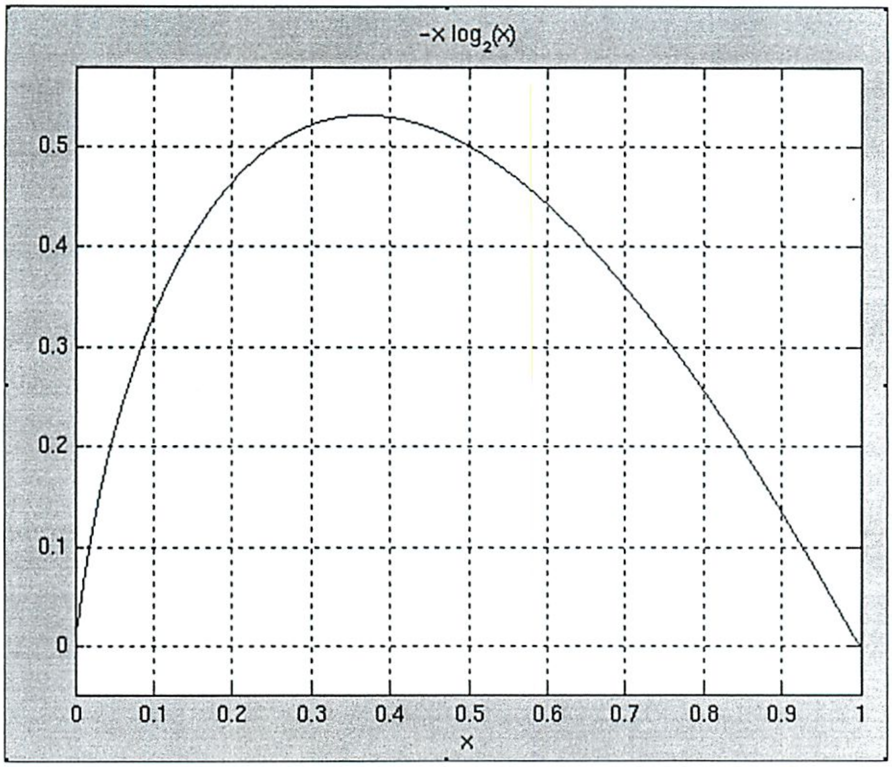
Let's practice calculating this. A simple example with 3 bins (classes), and 2 possibilities, + or O:

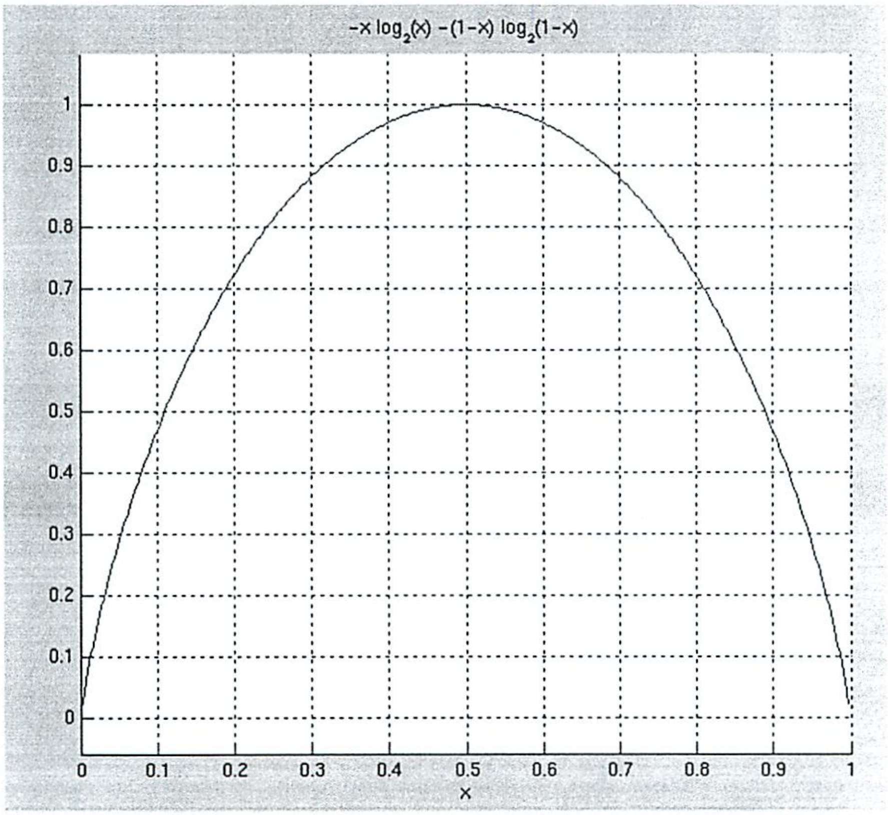
+ O O O + + O	+ + O O O +	+ O + + + + +

We calculate the entropy H in each of the three classes:
Class 1: 3 +, 4 O, 8 total, so + probability is 3/8 = 0.375, so from our 2nd graph: H_1= 0.95
Class 2: 3 +, 2 O, 6 total, so + probability is 3/6 = ½, so H_2= ½ + ½ = 1.0
Class 3: 7 +, 1 O, 8 total, so + probability is 7/8 =
Now we compute the *weighted average* of these three H values. There are 8+6+8 objects in all, or 22, so:

$$8/22(H_1)+6/22(H_2)+8/22(H_3) = 0.36(0.95)+0.18(1)+ 0.36(0.54)= 0.34+0.18+0.19=0.71$$

This is the *average disorder* of this particular split into 3 classes. This is the number used to 'drive' the algorithm, which attempts to find the split that achieves the *lowest* average disorder.





See also the table of binary entropy values a few pages later on.

Example formulas.
The disorder equation for a test with two branches, left and right, (l, r), with each branch having 2 (binary) classes or bins.
Let a = count of class 1 on the left side; b = count of class 2 on the left side;
Let c = count of class 1 on the right side; d = count of class 2 on the right side
$a + b = l \quad c + d = r; r+l= T$

$$\text{Disorder}= \frac{l}{T}\left(\left[-\frac{a}{l}\log_2\frac{a}{l}\right]+\left[-\frac{b}{l}\log_2\frac{b}{l}\right]\right)+\frac{r}{T}\left(\left[-\frac{c}{r}\log_2\frac{c}{r}\right]+\left[-\frac{d}{r}\log_2\frac{d}{r}\right]\right)$$

For a test with 3 branches, and 2 binary class outputs (this is the formula for the example we explicitly did earlier):

$$\text{Disorder} = \frac{b_1}{T}H\left(\frac{a}{b_1}\right)+\frac{b_2}{T}H\left(\frac{c}{b_2}\right)+\frac{b_3}{T}H\left(\frac{e}{b_3}\right)$$

a = count of class 1 on branch 1 $\quad$ b = count of class 2 on branch 1
c = count of class 1 on branch 2 $\quad$ d = count of class 2 on branch 2
e = count of class 1 on branch 3 $\quad$ f = count of class 2 in branch 3
$a+b = b_1 \quad c + d = b_2 \quad e + f = b_3$

Homogeneous Partitioning Trick
A time-saving heuristic shortcut to picking the lowest disorder test.
1. Pick tests that break the space into a *homogeneous portion* and a *non-homogeneous* portion
2. Pick the test that partitions out the **largest** homogeneous portion; that test will most likely have the lowest disorder.
Caution! when the homogeneous portions are **about the same size**, you should compute the full disorder value. This is where this shortcut might break down!

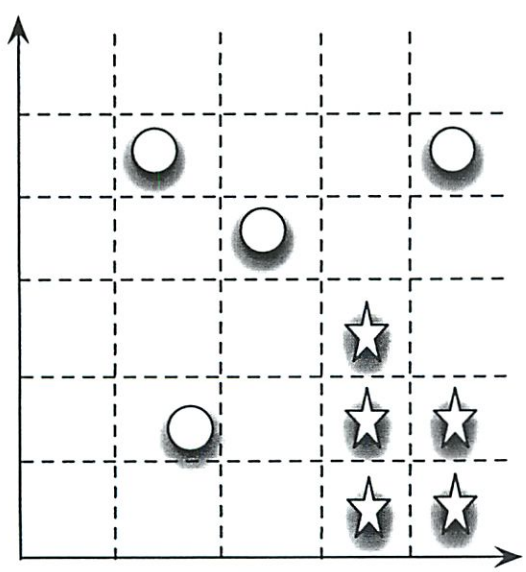
ID trees and Prediction – Test features of a query feature vector according to the identification tree generated during training, return the class at the leaf of the tree.
Relevant features? Irrelevant features are ignored because have large disorders.
Whose Razor? Occam's: The world is inherently simple. Choose the smallest consistent tree.
Why greedy? Finding the simplest tree is computationally intractable; so we use a greedy search using minimum average disorder as a heuristic.

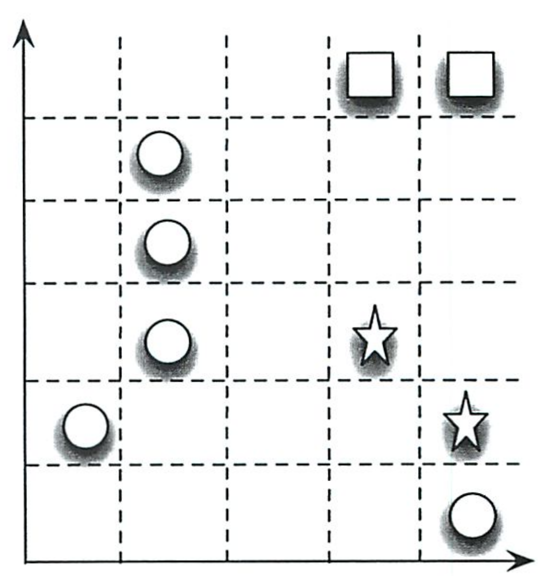
Table of common Binary Entropy values

Note: because $H(x)$ is a symmetric function, i.e. $H(1/3) = H(2/3)$, fractions > 1/2 are omitted.

/3 to /9				/10 to /13			
numerator	denominator	fraction	*H*(fraction)	numerator	denominator	fraction	*H*(fraction)
1	3	0.33	0.92	1	10	0.10	0.47
2	3	0.67	0.92	2	10	0.20	0.72
1	4	0.25	0.81	3	10	0.30	0.88
2	4	0.50	1.00	4	10	0.40	0.97
1	5	0.20	0.72	1	11	0.09	0.44
2	5	0.40	0.97	2	11	0.18	0.68
3	5	0.60	0.97	3	11	0.27	0.85
1	6	0.17	0.65	4	11	0.36	0.95
2	6	0.33	0.92	5	11	0.45	0.99
3	6	0.50	1.00	1	12	0.08	0.41
1	7	0.14	0.59	2	12	0.17	0.65
2	7	0.29	0.86	3	12	0.25	0.81
3	7	0.43	0.99	5	12	0.42	0.98
1	8	0.13	0.54	1	13	0.08	0.39
2	8	0.25	0.81	2	13	0.15	0.62
3	8	0.38	0.95	3	13	0.23	0.78
4	8	0.50	1.00	4	13	0.31	0.89
1	9	0.11	0.50	5	13	0.38	0.96
2	9	0.22	0.76	6	13	0.46	1.00
3	9	0.33	0.92				
4	9	0.44	0.99				

Try some sample 'cuts' in these two figures....which is the best single cut(s) in each? Why? And the next cut?





Identification trees Problem 1 (same credit card problem as k-NN above)

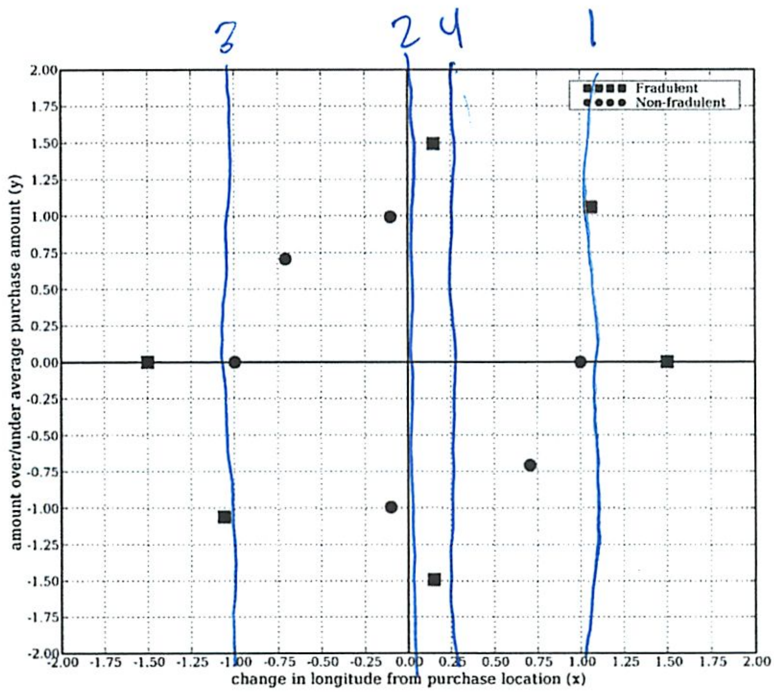
B1 The boundaries (18 pts)

Lucy now decides that she'll try to use identification trees on the data. There are three likely candidates for splitting the data: x=0.0, x=-1.01 and x=1.01. Note that the -1.01 and 1.01 values lie half-way between a square and a circle with nearby x values. Compute the average disorder for the decision boundary x=1.01. Your answer may contain logarithms.

Compute the average disorder for the decision boundary x=0.0. Again, your answer may contain logarithms.

Which of the two decision boundaries, x= 0.0 and x=1.01, is added first?

Sketch all of the decision boundaries on the figure below. Assume that x= 0.0 and x=1.01, in the order you determined above, are the first two decision boundaries selected (this may or may not be true, but assume it is).



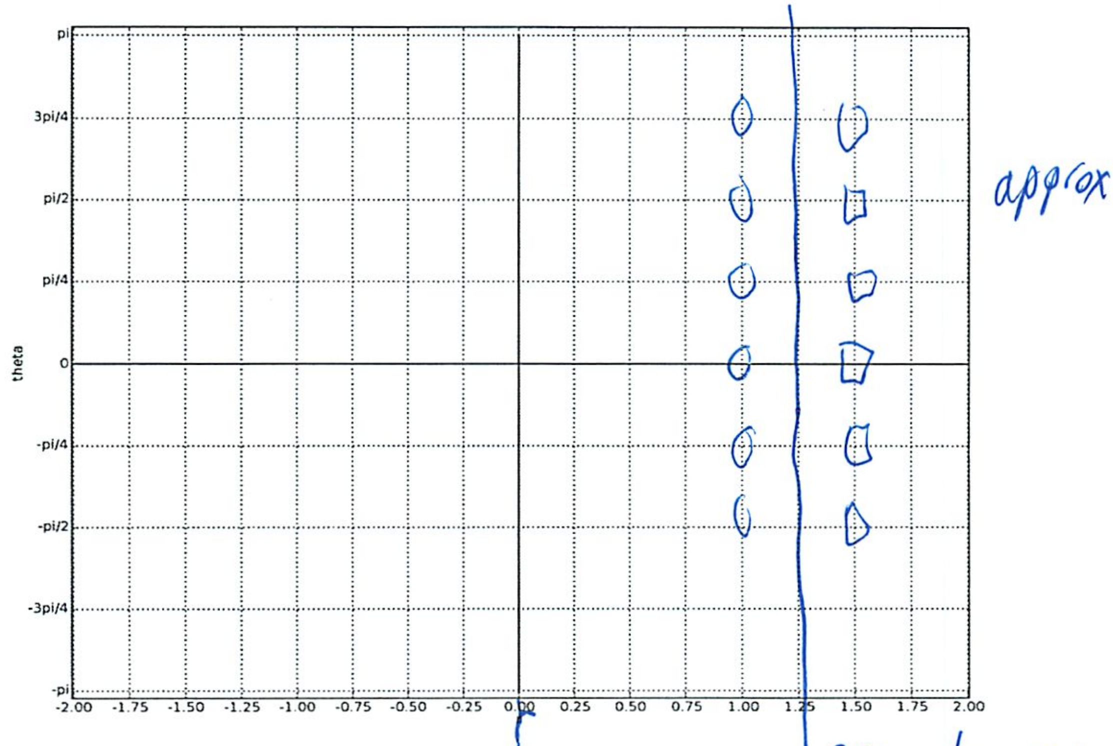
B2 The identification tree (7 pts)

Draw the identification tree corresponding to your decision boundaries.

What is the classification of the new charge (triangle)?

Part C: Polar coordinates (10 pts)

Lucy gets smart and decides to try a different space for each of the points. That is, she converts all of the points to polar coordinates. Sketch the data below. **You may assume that *r* value of each point is very close to a multiple of 0.25 and that the theta value of each point is very close to a multiple of pi/4.**



easy to separate

How many decision boundaries do we need in this case?

Draw the resulting identification tree and sketch the decision boundary on the graph above.

Identification Trees Practice Problem 2

Part B1 (2 Points)

Now, leaving nearest neighbors behind, you decide to try an identification-tree approach. In the space below, you have two possible initial tests for the data. Calculate the average disorder for each test. Your answer may contain $\log_2$ expressions, but no variables. The graph is repeated below.

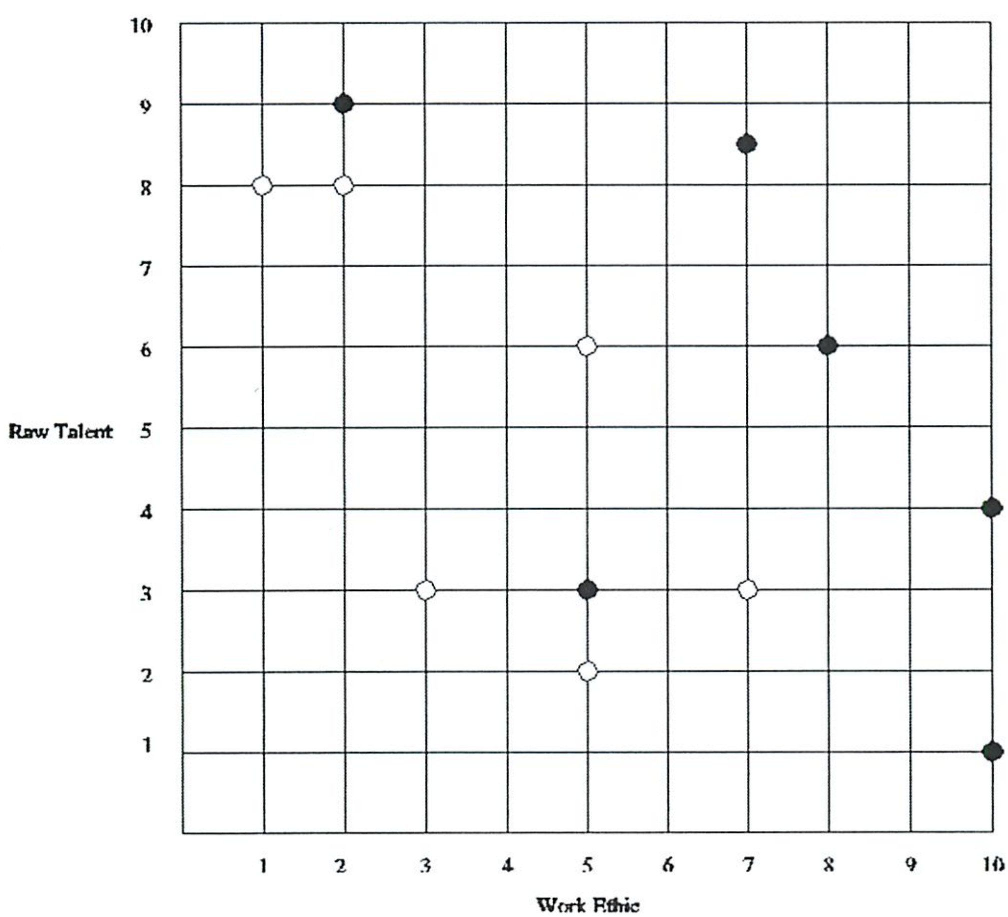
Test A: R > 5:

Test B: W > 6:

Part B2 (2 Points)

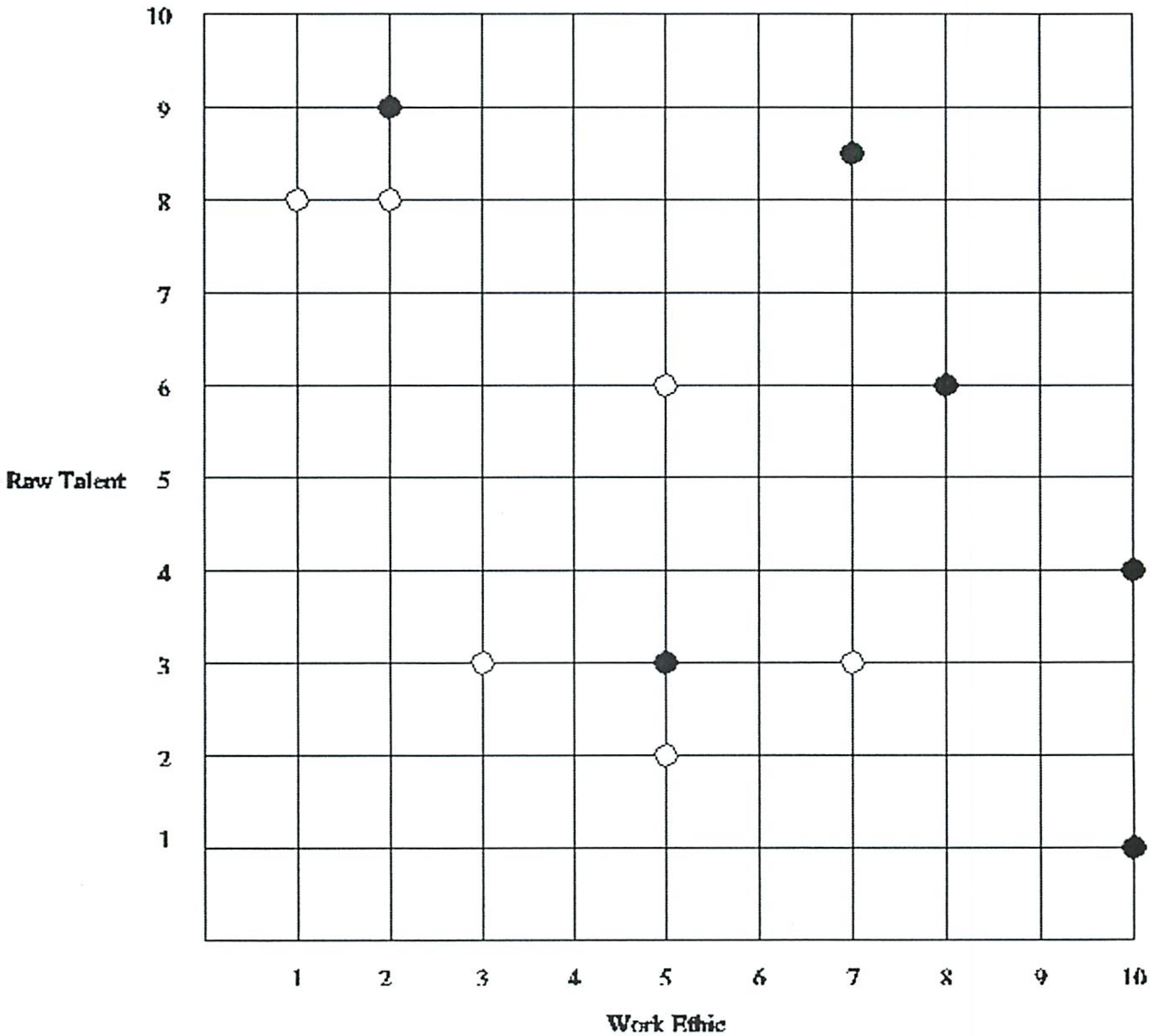
Now, indicate which of the two tests is chosen first by the greedy algorithm for building identification trees.

We include a copy of the graph below for your scratch work.



Part C: Identification Trees (4 Points)

Now, assume R > 5 is the first test selected by the identification-tree builder (which may or may not be correct). Then, draw in all the rest of the decision boundaries that would be placed (correctly) by the identification-tree builder:

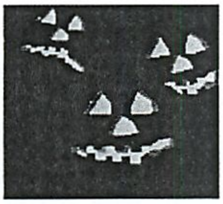


10/27

Massachusetts Institute of Technology

Department of Electrical Engineering and Computer Science
6.034 Artificial Intelligence, Fall 2011
Recitation 7, October 27

Neural networks Prof. Bob Berwick, 32D-728

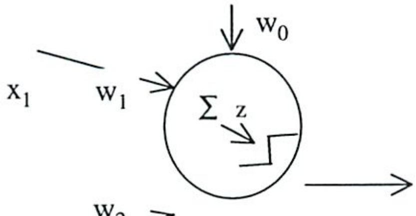


Part 2

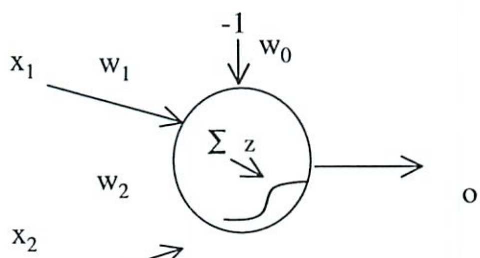
0. Introduction

Neural nets are networks of simulated neurons, each of which has weighted inputs, an adder that sums the weighted inputs, a threshold function that checks the sum against a threshold and returns an output. By means of *forward propagation*, inputs are run through the network to produce outputs: At each level of the network, weighted inputs are summed, then run through either a step function or a sigmoid to produce an output. As with other machine learning techniques, the goal in using a neural net is to classify unknown inputs, i.e., assign a known class to an unknown input. (Output that is continuous rather than discrete implements regression rather than classification.)

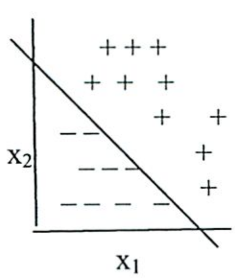
A simulated neuron that employs a *step function* returns 1 or 0, depending on whether the input is above or below a specified threshold value, respectively. Neural nets using these simulated neurons, sometimes called perceptrons, can be thought of as *linearly* dividing a space of input vectors into regions, thereby creating decision boundaries. The weights in multi-layer perceptron nets cannot be automatically computed, i.e., learned, because of the discontinuity in the derivative of the threshold function.
The more usual kind of neuron in a neural net employs a *sigmoid function* for its threshold, $y=1/(1+e^{-x})$. As Winston says, this solves several issues: it is continuous, and so differentiable everywhere, which we'll need in order to learn by using gradient ascent (or descent); the value y approaches 1 as x becomes highly positive; 0 as x becomes large and negative; and exactly ½ when x=0. As Winston shows, the really great news is that the derivative is: $dy/dx=y(1-y)$.
However, such sigmoid neural nets are extremely difficult to design by hand. Weights are learned via a *backpropagation* algorithm.



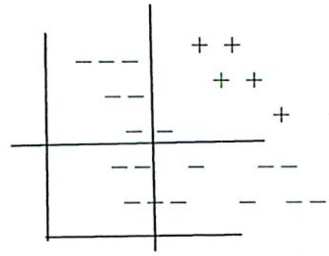
step function neuron



sigmoid neuron



(a) one decision boundary



(b) two decision boundaries (neurons) needed for an AND

We can think of a step function net as defining a *linear* decision boundary, $ax + by - c = 0$. The sum of the weights of a step function neuron can represent the equation of the line: $w_1x_1 + w_2x_2 - w_0 = 0$. In example (a) above, to the left of the line can be represented by a 0 output; to the right a 1 output. Each decision line is represented by one neuron in the lowest layer of the net. Each class is represented by a neuron in the final layer, with two classes only requiring one neuron. Interior neurons implement Boolean combinations, which allow for classifying data sets such as that shown in (b) above.

Backpropagation is a method for adjusting the weights in a network according to training data. The training data comes in the form of sample (input, desired output) pairs (these of course could be multi-dimensional vectors). To use backpropagation, we first compute for each input item what the network outputs with its **current** weights. **Remember this!** This is called **forward propagation**, and is always the **first** step in working with neural nets. It will yield a (perhaps vector) output value, *o*, one output value for each sample, e.g., o_{sample}. We then compare the desired output for each sample, d_{sample} to the observed output o_{sample} that the network computes. If the network is already 'on target' then the computed outputs from sample points o_{sample} will already equal the desired output. But typically this will not be the case, and there will be some difference. Based on a performance function *P* of this difference of *d–o*, we will adjust the network's current weights to come closer to the desired values, for each sample.

But how to change the weights? The idea is to use *partial derivatives* to see how much the output will change if we tweak the weights just a tiny bit, using the idea of *gradient ascent* with respect to the particular performance function $P = -1/2(d-o)^2$. (We have dropped the 'sample' subscript here.) Why do we use this function *P* instead of just $(d-o)$? As Winston notes, this formula also has nice mathematical properties: (i) it yields a maximum when $o=d$; (ii) it monotonically decreases as *o* deviates from *d* (so we can use it for "hill-climbing" or gradient 'ascent', or 'descent'); and (iii) the derivative of *P* with respect to *o* is very simple:

$$\frac{dP}{do} = \frac{d}{do}\left[-\frac{1}{2}(d-o)^2\right] = -\frac{2}{2}\times(d-o)^1\times -1 = d-o$$

Backpropagation works by trying to find the maximum of *P* (recall this is where $d=o$) by tweaking the weights *w*, so you move in the direction of the gradient in a space that gives *P* as a function of the weights, *w*. The following formula shows how much to change a weight, *w*, in terms of a partial derivative, i.e.:

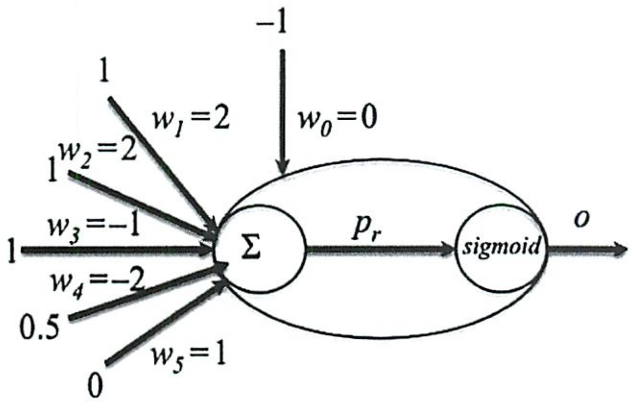
$$\Delta w \propto \frac{\partial P}{\partial w} \text{ so } w' = w + \alpha \times \frac{\partial P}{\partial w} \text{ where } \alpha \text{ is a rate constant (also } r \text{ in the literature); } \Delta w = \delta$$

The rate constant is like the step-size in hill climbing: too small, and you don't converge fast enough; too big, and you might overstep a solution, and oscillate; so this value must be chosen empirically.

We start at the very last, output layer of the network, and work backwards, calculating the change layer by layer; at each layer calculating the *change* δ we should best make at that layer of the net to most rapidly climb the performance function 'hill'. Then we must decide how to distribute the total change at this layer over all the inputs feeding this particular layer; we do this by dividing up the total δ according to the weights of the previous layer and their corresponding inputs (thus the previous layer's weights that are largest, and whose inputs are largest, get proportionately more change–the 'squeaky wheel' model). This method is called **backpropagation** because we 'propagate' the error *d–o* backwards through the net.

You should work through the details in Winston's notes, but here we'll do a bit of the same, reviewing as we go. For any neural network, you should know (1) how to do forward propagation (this is a matter of taking the assigned weights and inputs and running the system forward to its output value); (2) how to run a few steps of backpropagation, for each of three cases: (i) the final output layer case; (ii) an interior neuron case, where the neuron is fed by more than one previous neuron; and (iii) an output or interior node case where the neuron itself feeds more than one following neuron (so including more than one output). This will usually involve thinking about the chain rule and partial derivatives. This covers all the possible network topologies. Let's try the first two.

Case 1. The final output layer case. Consider the following single neuron system. How should we adjust the weights w_{ij} ? in this example, assume the initial weights w_0 through w_5 are as given in the figure just below, and the sample inputs on the 6 'input lines' are as shown. Note the "–1" line associated with weight w_0. This is called a 'bias weight' and is standardly used in the field. The inputs x_i are multiplied by their corresponding weights w_i and the result is summed (as shown by the sum box). This output is then passed through the sigmoid threshold function, resulting in the output from the neuron, o. (Sometimes the output is labeled y or z in the literature – just to alert you.) Finally, let us suppose that the **desired**_value for the network, d, is 1.0, and that the **learning rate constant**, denoted α in Winston's notes (and r in the traditional literature) is 100. (Why do we need this thing anyway?)



d=1.0, α=100

Step 0: Forward Propagation. Calculate the output o given the input values shown. Useful data point: *sigmoid*(2)= 0.9

Answer: $-1 \times w_0 +$ __1__ $\times w_1$ __1__ $\times w_2 +$ __1__ $\times w_3 +$ __.5__ $\times w_4 +$ __0__ $\times w_5 = p_r =$ __2__

Sigmoid $(p_r) = o_f =$ __.9__

So, does the *desired* output d equal the *observed* output o in this case? How far apart are they?

Step 1: Backpropagation to find delta for output layer.

Calculate the final (and here, only) node's effect on the performance function, P, which we will call δ_f. You can think of this as this neuron's contribution to P. It is the derivative of P with respect to the node's total input, **given** what is feeding the sigmoid function p_r, so may be found via the chain rule, with $P = -1/2(d-o)^2$. Recall: the derivative of P with respect to o is just $(d-o)$. Also recall the derivative of the sigmoid function with input x with respect to output y is $y(1-y)$, so with input $x = p_r$ this is just $o(1-o)$. So we therefore have this equation, where δ_f indicates that this is the δ for the *final* output (in the last page of the notes, Winston also denotes this δ_o, for 'output layer':

$$\delta_f = \frac{\partial P}{\partial p_r} = \frac{\partial P}{\partial o}\frac{\partial o}{\partial p_r} = (d - o_f) \times [o_f(1 - o_f)]$$

Given that d=1.0 and we now know o_f= ____________, then δ_f= __________________________

Step 2: Distribute changes to the weights using the chain rule. We next determine how to distribute contributions (or 'blame'), of this 'total' δ_f in order to tweak the weights.

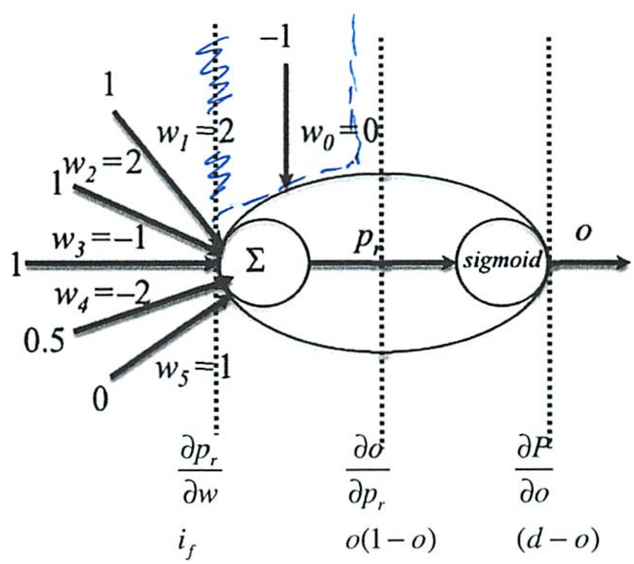
This is the full formula in the Winston notes for an output layer, where the factor i comes from the partial derivative of p_r with respect to each weight w_i. Thus, if we have multiple weights, w_i, as in this case, and are adding them as usual, this partial derivative is simply i_f, the input for each weight.

$$\frac{\partial P}{\partial w} = \frac{\partial P}{\partial o}\frac{\partial o}{\partial p_r}\frac{\partial p_r}{\partial w} = (d - o_f) \times [o_f(1 - o_f)] \times i_f = \delta_f \times i_f$$

We adjust the weights w according to the formula given earlier, $w'_i = w_i + \alpha \times \delta_f \times$ input to w_i. With the learning factor set to 100, you should be able to compute all the new w's. Try to understand why some weights go **up** and other weights go **down** and why others do not change at all. What happens if the **input** to a particular weight is **zero**? Should that weight change? (Think of its potential contribution to the output.) After you have figured out the new weights, calculate the **new** value of the network's output, to see if we have gotten any closer to the desired output.

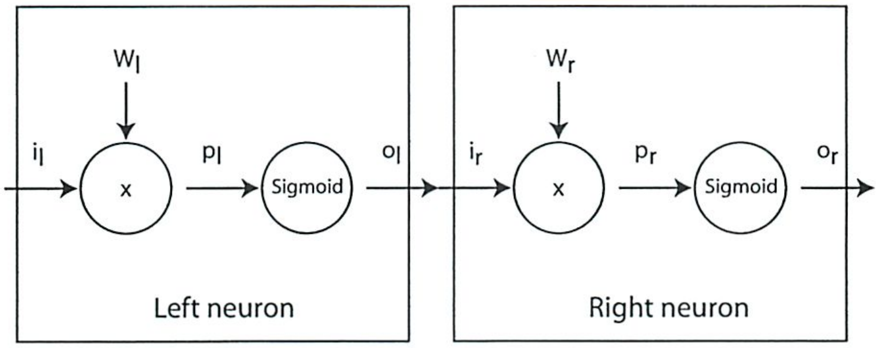
New Weight	ORIGINALWEIGHT +	RATE ×	δ_f ×	INPUT =	NEW WT VALUE
w'	w	α	$(d\text{-}o)(o)(1\text{-}o)$	i_f	
$w'_0 =$	0	100		−1	−0.9
$w'_1 =$	2	100		1	
$w'_2 =$	2	100		1	
$w'_3 =$	−1	100		1	
$w'_4 =$	−2	100		0.5	
$w'_5 =$	1	100		0	

OK, before moving on, let's see a picture of this simple output stage, containing three parts, with the chain of three partial derivatives below to help you picture how the partial derivatives do their work: **each** partial derivative "**pushes backwards**" through **one part** of the network at a time. Working through the formula, $\frac{\partial P}{\partial w}$, from left to right: first $\frac{\partial P}{\partial o}$ pushes backwards from the output through the sigmoid function telling us how given some change in o; $\frac{\partial o}{\partial p_r}$ pushes back through the summed p_r, telling us how o changes given some change in the sum; and finally, $\frac{\partial p_r}{\partial w}$ pushes back through the weights, telling us how the sum changes given a change in a particular weight.



Case 2. Two nodes connected together (Winston's simple case).

This time, the network is two neurons deep: one output neuron, on the *right*, and one 'hidden' neuron on the *left* ('hidden' because it does not **directly** connect to input or output nodes). Following the Winston notes, we will call the final output an output on the *right*, o_r and the input to the last neuron on the *right* will receive an input, i_r that feeds it, which we will also label o_l, since this is also the output from the left-hand neuron. We have one inputs, i_l, feeding the left neuron.



Now let's add weights. Assume **all initial weights are 0** (it is actually a bad idea to set all initial weights the same for neural nets; why?). Assume a sample input of $i_l = 1$, and that the *desired* output value d is 1.0. Assume a learning rate of 8.0. (Useful data point: *sigmoid*(0) = 0.5) Let's run one step of backpropagation on this and see what's different about this case. First, as before, we must carry out **forward** propagation: compute the inputs and outputs for each node.

Step 1: Forward Propagation. OK, you should know the drill by now. First compute the outputs at each node:

$p_l = w_l i_l$ = 0 × = ________ So o_l = *sigmoid*(   ) = ____________.

$p_r = w_r\, i_r$ = 0 × = ________ So o_r = *sigmoid*(   ) = ____________.

(**Important:** recall that instead of o_r Winston uses the notation o_f to denote the *final* output from the right neuron.)

Step 2: Calculate the δ for the output, final layer, δ_f (for use in changing w_r)

Recall the formula for δ_f is: $o_r \times (1 - o_r) \times (d - o_r)$ = ______ × (______) × (______) = ______.
Recall that $d = 1.0$; and we have just computed o_r

Step 3: Calculate the δ_l for the hidden neuron

Now we will march one more step backwards through the network, working our way through the sigmoid on the left, and then weight on the left so that we can calculate the partial of the performance function with respect to the weight on the left, w_l; remember, this is what tells us how much to 'tweak' that weight. When we do that, we get the following equation (following Winston):

$$\begin{aligned}
\frac{\partial P}{\partial w_l} &= \frac{\partial P}{\partial o_r} \times \frac{\partial o_r}{\partial w_l} \\
&= \frac{\partial P}{\partial o_r} \times \frac{\partial o_r}{\partial p_r} \times \frac{\partial p_r}{\partial w_l} \\
&= \frac{\partial P}{\partial o_r} \times \frac{\partial o_r}{\partial p_r} \times \frac{\partial p_r}{\partial o_l} \times \frac{\partial o_l}{\partial w_l} \\
&= \frac{\partial P}{\partial o_r} \times \frac{\partial o_r}{\partial p_r} \times \frac{\partial p_r}{\partial o_l} \times \frac{\partial o_l}{\partial p_l} \times \frac{\partial p_l}{\partial w_l}
\end{aligned}$$

This looks complicated, but it isn't! Recall the way we work back through the network, right to left. The first three terms after the equals sign correspond to derivatives of the performance function P with respect to the following parts of the network: (1) the output; (2) the right-hand sigmoid; and (3) the right-hand weights. The two new factors are derivatives of P with respect to the last two parts of this network, (4) the left-hand sigmoid; and (5) the left-hand weight. So what this formula does, in effect, is figure how much P 'jiggles' when we 'jiggle' the left-hand weight. Note that we've already computed terms (1) and (2) when we did the calculation for δ_f. As Winston notes, we can see how the left and right parts fit together by comparing the two derivatives for the left and right parts:

$$\frac{\partial P}{\partial w_r} = (d-o_r)\times o_r(1-o_r)\times i_r$$

$$\frac{\partial P}{\partial w_l} = (d-o_r)\times o_r(1-o_r)\times w_r \times o_l(1-o_l)\times i_l$$

Note again that the *first two* terms of the first equation above are the same as the *first two* terms of the second equation – this is the same 'jiggle' working its way back through the network, from output, through the right-hand sigmoid. We can simplify these by defining δ as follows, either for the left or right neuron.

$$\delta_r = o_r(1-o_r)\times(d-o_r)$$

$$\delta_l = \delta_r \times w_r \times o_l(1-o_l)$$

So again, we get the delta on the left simply by taking the delta on the right and then *adding* new terms to accommodate the partial derivatives working through the weights on the right and then the sigmoid on the left. As Winston notes, we can now write the weight change equations this way:

$$\Delta w_r = \Delta w_f = \alpha \times \delta_f \times i_f \text{ where } \delta_f = o_f(1-o_f)\times(d-o_f)$$

$$\Delta w_l = \alpha\times\delta_l\times i_l = \alpha\times\delta_l\times i_l \text{ where } \delta_l = o_l(1-o_l)\times w_r\times\delta_f = o_l(1-o_l)\times w_r\times o_f(1-o_f)\times(d-o_f)$$

$$\Delta w_l = \alpha\times o_l(1-o_l)\times w_r\times\delta_f\times i_l$$

Step 4. Calculating the new weights.

Now we can calculate the weight changes for both weights, and add these to the starting weights to get the new, updated weights. For the first weight change, the right neuron's weight, we use the δ_f and i_f values from our final layer calculation from above. To calculate the second weight change, we need the values of δ_f, w_r, the output from the left neuron, o_l, and finally the input to the left neuron, i_l:

$w_f' = w_f + \alpha \times \delta_f \times i_f$ = ______ + 8.0 × ______ × ______ = ______.

$w_l' = w_l + \alpha \times o_l(1-o_l) \times w_r \times \delta_f \times i_l$ = ______ + 8.0 × ______ × (______) × ______ × ______ × ______ = ______.

Have the new weights gotten us any *closer* to our target value of 1.0?

The next iteration!

To continue the process, using the new weights, we again do forward propagation and then backpropagation.

Step 1. Forward propagation.

$p_l = w_l i_l$ = ______ × ______ = ______ So o_l= *sigmoid*(______) = ______.

$p_r = w_r i_r$ = ______ × ______ = ______ So o_r= *sigmoid*(______) = ______.

So, have we gotten any *closer* to the desired output of 1, as we wanted from gradient ascent?

Step 2. Backpropagation.

$\delta_f = o_f \times (1-o_f) \times (d-o_f)$ = ______ × (______) × (______) = ______.

Compare this value of delta to the first iteration. Which one is larger? Now for the weight change on the left neuron:

$\delta_l = o_l \times (1-o_l) \times w_r \times \delta_f$ = ______ × (______) × ______ × ______ = ______.

Now we can compute the weight changes for the left and right neurons:

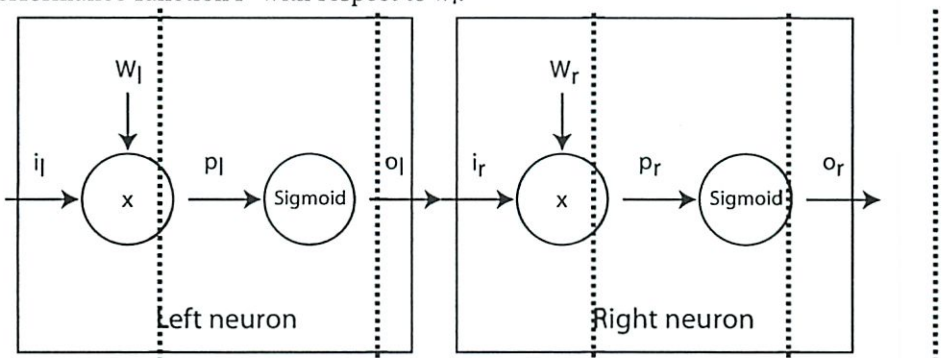
$\Delta w_f = \alpha \times i_f \times \delta_f$ = 8.0 × ______ × ______ = ______.

$\Delta w_l = \alpha \times i_l \times \delta_l$ = 8.0 × ______ × ______ = ______.

So the **new** weights on this second iteration become:

$w_f' = w_f + \Delta w_f$ = ______ + ______ = ______; $w_r' = w_r + \Delta w_r$ = ______ + ______ = ______.

For completeness, and to cement our understanding, let's see how the various terms in the partials are arrayed over this two-neuron diagram, pushing back from the output o_r, so you can see why it is called **backpropagation.** Make sure you **understand** where each of the five terms comes from. Multiplied together, they give the partial derivative of the performance function P with respect to w_l.



$\frac{\partial p_l}{\partial w_l}$	$\frac{\partial o_l}{\partial p_l}$	$\frac{\partial p_r}{\partial o_l}$	$\frac{\partial o_r}{\partial p_r}$	$\frac{\partial P}{\partial o_r}$
i_l	$o_l(1-o_l)$	w_r	$o_r(1-o_r)$	$(d-o_r)$
left weight deriv	left sigmoid deriv	right weight deriv	right sigmoid deriv	output deriv

6.034 Mega Recitation

10/28

Note $4^7 \neq 7^4$

Neural Nets

2009 was really hard

Silver Star 2009 = Great White Whale

Today hodgepage of 3 different questions

Core quesion: running forward then back tracking

Core Formula get new weight $w_{ij}' = W_{ij} + r\, o_i\, \delta_j$

not final output

(or α) direction

$\delta = (O^* - O)(O_1(1-O_1))$

final output deriv of performance fn

$P = \frac{1}{2}(O^* - O)^2$

Neural nets : hillclimbing in multiple dimensions form of machine learning

---

$\delta_i = O_i(1-O_i) \sum_{\substack{j \text{ s.t.} \\ i \to j}} w_{ij}\delta_j$

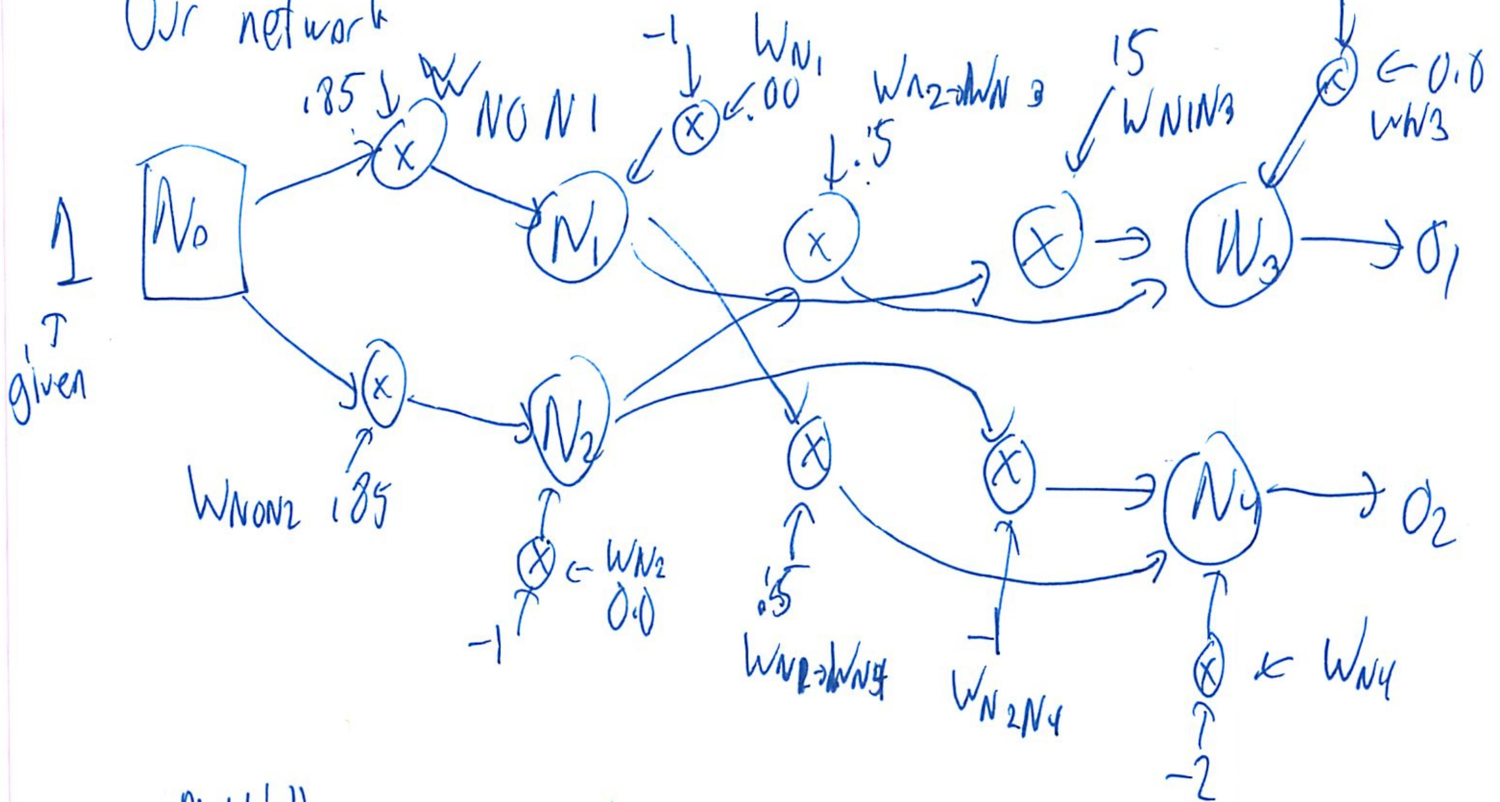
non final output

leads into

iterative backwards propagation

②

Our network



Multipliers are alwas there

No threshholds - but remember it

Run it through for $N_0 = 1$

$O_{N_1} = .3$

$O_{N_2} = .7$

$O_1 = O_{N_3} = .6$

$O_2 = O_{N_4} = .6$

(3)

Now need to run backwards

Desired $O_1 = 1$ Had .6

$O2 = 1$ .6

Looks fairly symetric

Need new weights

So need to calculate deltas

$\delta_{N3} = .4 \cdot .4 \cdot .6 = .096$

$= (O^*_{N3} - O_{N3}) O_{N3} (1 - O_{N3})$

$(1 - .6) \quad .6 \quad (.4)$

$\delta_{N4} = .096 \leftarrow$ symetric

$\delta_{N1} = (1 - .3) \; .3 \cdot \underset{\uparrow}{(.5 \cdot .096) + (.5 \cdot .096)} = .096$

$\sum_{i \to j}$ here N_3, N_4

Now quiz wants new weights

$W_{N1N3}' = W_{N1N3} + r (O_{N1} \delta_{N3})$

$= .5 + \underset{\substack{\uparrow \\ \text{told}}}{.1} (.3 \cdot .096)$

(4)

$$W_{N0N1}' = W_{N0N1} + r \cdot O_{N0} \cdot \delta_{N1}$$

$$= -.85 + (.1)(1)(.21)(.096)$$

$.3 \cdot .7$

If change sigmoid to inv tangent

~~[illegible]~~ Weight Fn won't change $\leftarrow W_{ij}'$

δ_1 would change

- Since deriv of sigmoid would change

$(1-O_1)^2$ instead $(1-O_1)$

Pinaple problem in 2006 ~~[illegible]~~ is hard

2011: Match game

Nets + Outputs

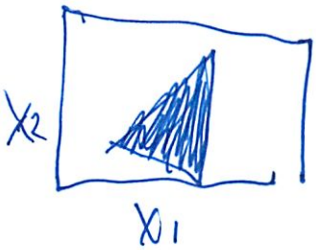
Can match each net ~~[illegible]~~ to each graphical ~~[illegible]~~ output

5

All sigmoids are step fn here

- below threshhold out
- above " in

1



A is simplelest

$ax_1 + by = t$

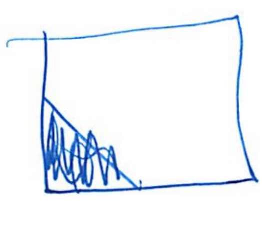
↑ cut off pt

$$y = \frac{t - ax}{b}$$

↑ a line

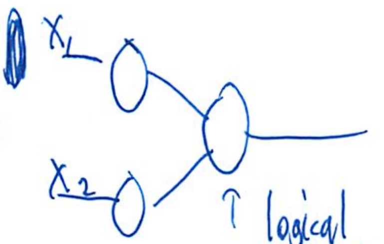
Can only do 6

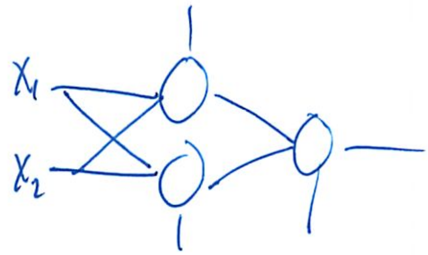
6



B 6

B, D are similar

D


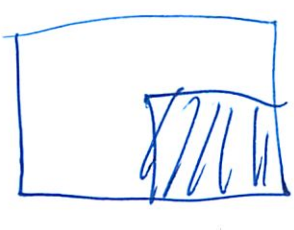
B


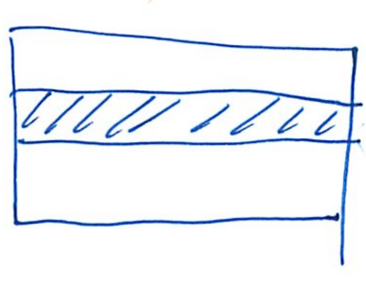
↑ logical operator
- can have it be AND - high thresholds
- Or NOT ≤ 0

Deeper things in graph are logic
Can't do XOR w/ 1 node

So B, D can draw 2 lines

D - must be vertical lines
B can be any ~~line~~ vertical + horiz line

5  D

4  B

E

What can we do for 3?

C

- can do XOR

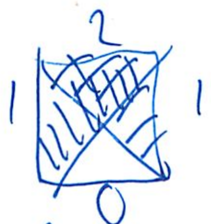
E, F left for 1, 2

E - any triangle

F - one line must be horiz, other vertical

XOR can't be done w/ 1 layer

- do it w/ double layer

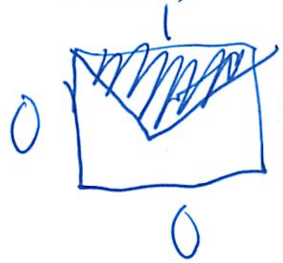


first
2 nodes

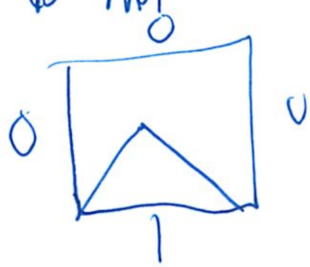
8

But have another layer

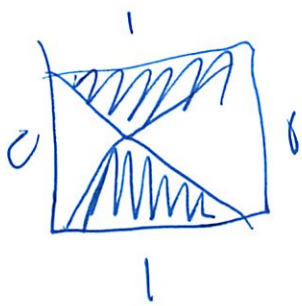
- Could be and
- Only accepts



- Can do Not



- Then Or



Lab 4

From 6.034 Fall 2011

Contents

This problem set is due Friday, November 4th. If you have questions about it, ask the list *6034tas@csail.mit.edu*.

To work on this problem set, you will need to get the code:

- You can view it at: http://web.mit.edu/6.034/www/labs/lab4/
- Download it as a ZIP file: http://web.mit.edu/6.034/www/labs/lab4/lab4.zip
- Or, on Athena, `attach 6.034` and copy it from `/mit/6.034/www/labs/lab4/`.

This lab has two parts, the first part is on CSPs and the second part is on learning algorithms, specifically KNN and decision trees.

Constraint Satisfaction Problems

In this portion of Lab 4, you are to complete the implementation of a general constraint satisfaction problem solver. You'll test it on problems we've worked out by hand in class.

We have provided you a basic CSP implementation in `csp.py`. The implementation has the Depth-first-search already completed. It even has a basic built in constraint checker. So it will produce the search trees of the kind for DFS w/ back tracking with basic constraint checking.

However, it doesn't do forward checking or forward checking + singleton propagation!

So your job is to complete:

```
forward_checking(state):
```

and

we already had the test on this un

```
forward_checking_prop_singleton(state):
```

in the file `lab4.py`. Here `state` is an instance of `CSPState` an object that keep track of the current variable assignments and domains. These functions are called by the Search algorithm at every node in the search tree. These functions should return False at points at which the Domain Reduction Algorithm would backtrack, and True otherwise (i.e. continue extending).

As a hint, here is the (unrefined) pseudocode for the two algorithms.

Forward Checking

1. Let X be the variable currently being assigned.
2. Let x be the value being assigned to X.
3. Find all the binary constraints that are associated with X.
4. For each constraint:
    1. For each neighbor variable, Y, connected to X by a binary constraint.
        1. For each variable value y in Y's domain
            1. If constraint checking fails for X=x and Y=y
                1. Remove y from Y's domain
            2. If the domain of Y is reduced down to the empty set, then the entire check fails: return False.
5. If all constraints passed declare success, return True

If you get a state with no current variable assignment (at the Root of the search tree) then you should just True, since forward checking could only be applied when there is some variable assignment.

Forward Checking with Propagation through Singletons

1. Run forward checking, fail if forward checking fails.
2. Find variables with domains of size 1.
3. Create a queue of singleton variables.
4. While single queue is not empty
    1. Pop off the first singleton variable X (add X to list of visited singletons)
    2. Find all the binary constraints that singleton X is associated with.
    3. For each constraint therein:
        1. For each neighbor variable, Y, connected to X by a binary constraint:
            1. For each value of y in Y's domain:
                1. If constraint check fails for X = (X's singleton value) and Y = y:

1. Remove y from Y's domain
2. If the domain of Y is reduced down to the empty set, then the entire check fails, return False.

4. Check to see if domain reduction produced any new and unvisited singletons; if so, add them to the queue.
5. return True.

API

These are some useful functions defined in `csp.py` that you should use in your code to implement the above algorithms:

`CSPState`: representation of one of the many possible search states in the CSP problem.

- `get_current_variable()` - gets the Variable instance being currently assigned. Returns None if we are in the root state, when there are no variable assignments yet.
- `get_constraints_by_name(variable_name)` - retrieves all the BinaryConstraint objects associated with `variable_name`.
- `get_variable_by_name(variable_name)` - retrieves the Variable object associated with `variable_name`.
- `get_all_variables()` - gets the list of all Variable objects in this CSP problem.

`Variable`: representation of a variable in these problems.

- `get_name()` - returns the name of this variable.
- `get_assigned_value()` - returns the **assigned** value of this variable. **Returns None if `is_assigned()` returns False, that is if the variable hasn't been assigned yet.**
- `is_assigned()` - returns True if we've made an assignment for this variable.
- `get_domain()` - returns a copy of the list of the current domain of this variable. Use this to iterate over values of Y.

You might want to consider using this method to get the singular value of a variable with domain size reduced to 1.

- `reduce_domain(value)` - remove `value` from this variable's domain.
- `domain_size()` - returns the size of this variable's domain

`BinaryConstraint`: a binary constraint on variable i, j: i -> j.

- `get_variable_i_name()` - name of the i variable
- `get_variable_j_name()` - name of the j variable
- `check(state, value_i=value, value_j=value)` - checks the binary constraint for a given CSP state, with variable i set by value i, and variable j set by value j. Returns False if the constraint fails. Raises an exception if value_i or value_j are not set or cannot be inferred from state.

NOTE: in our implementation of CSPs, constraints are symmetrical; a constraint object exists for each "direction" of a constraint, so you can check for the presence of a constraint by substituting for i and/or j in the most convenient fashion for you.

Here is how you might use the API to get the value of a variable currently being assigned.

```
var = state.get_current_variable()
value = None
if var is not None:    # we are not in the root state
   value = var.get_assigned_value()
   # Here value is the value of the variable current being assigned.
```

Here is how you might use the API to get the singular value from a singleton variable:

```
if singleton_var.domain_size() == 1
   value = singleton_var.get_domain()[0]
```

Testing

For unit testing, we have provided `moose_csp.py`, an implementation of the seating problem involving a Moose, Palin, McCain, Obama, Biden and You -- in terms of the framework as defined in `csp.py`.

Running:

```
python moose_csp.py dfs
```

will return the search tree for DFS with constraint checking. When you have finished your implementation, running `python moose_csp.py fc` or `python moose_csp.py fcps` should return the correct search trees under forward checking and forward checking with singleton propagation.

Similarly

Running:

```
python map_coloring_csp.py [dfs|fc|fcps]
```

Should return the expected search trees for the B,Y,R, state coloring problem from the 2nd Quiz in 2006.

There are also other fun solved CSP problems in the directory that you can test and play around with. You can submit your own unique solution to an interesting CSP problem to get extra credit!

EXTRA CREDIT

As extra credit, try to follow the code in `moose_csp.py` or `map_coloring_csp.py`, and implement a `problem()` function that returns a CSP instance for a problem of your own choosing.

You may do one of the problems from past quizzes: the 2009 Time Traveler scheduling problem or the 2010 Jigsaw puzzle question. Alternately, you may implement something that you find useful or interesting, ideas include: scheduling classes, seating guests for a wedding or dinner party (to maximize harmony), solving crypt-arithmetic puzzles, the 8-queens problem, or crossword puzzles.

You may also try to extend `csp.py`. For instance, you can add ability to find an optimal solution rather than just a constraint-satisfying solution (i.e. replace DFS with one of the optimal searches we've learned). Or you can add support for multi-variable constraints, and make the code solve the Max-flow problem from the

2006 final.

When you've succeeded in implementing such a problem or extension, send your working code to 6034tas@csail. Your reward: either a 1-to-3-day extension (depending on difficulty) on one of the previous or future labs, possibly erasing any late penalties. Or if your lab grade is already perfect, praise and recognition from the 6.034 staff.

Learning

Now for something completely different. Learning!

Classifying Congress

During Obama's visit to MIT, you got a chance to impress him with your analytical thinking. Now, he has hired you to do some political modeling for him. He seems to surround himself with smart people that way.

He takes a moment out of his busy day to explain what you need to do. "I need a better way to tell which of my plans are going to be supported by Congress," he explains. "Do you think we can get a model of Democrats and Republicans in Congress, and which votes separate them the most?"

"Yes, we can!" You answer.

The Data

You acquire the data on how everyone in the previous Senate and House of Representatives voted on every issue. (These data are available in machine-readable form via `voteview.com`. We've included it in the lab directory, in the files beginning with `H110` and `S110`.)

`data_reader.py` contains functions for reading data in this format.

`read_congress_data("FILENAME.ord")` reads a specially-formatted file that gives information about each Congressperson and the votes they cast. It returns a list of dictionaries, one for each member of Congress, including the following items:

- 'name': The name of the Congressperson.
- 'state': The state they represent.
- 'party': The party that they were elected under.
- 'votes': The votes that they cast, as a list of numbers. 1 represents a "yea" vote, -1 represents "nay", and 0 represents either that they abstained, were absent, or were not a member of Congress at the time.

To make sense of the votes, you will also need information about what they were voting on. This is provided by `read_vote_data("FILENAME.csv")`, which returns a list of votes in the same order that they appear in the Congresspeople's entries. Each vote is represented a dictionary of information, which you can convert into a readable string by running `vote_info(vote)`.

The lab file reads in the provided data, storing them in the variables `senate_people`, `senate_votes`, house_people, and house_votes.

Nearest Neighbors

You decide to start by making a nearest-neighbors classifier that can tell Democrats apart from Republicans in the Senate.

We've provided a nearest_neighbors function that classifies data based on training data and a distance function. In particular, this is a third-order function:

- First, call nearest_neighbors(distance, k), with distance being the distance function you wish to use and k being the number of neighbors to check. This returns a *classifier factory*.
- A classifier factory is a function that makes classifiers. You call it with some training data as an argument, and it returns a classifier.
- Finally, you call the classifier with a data point (here, a Congressperson) and it returns the classification as a string.

Much of this is handled by the evaluate(factory, group1, group2) function, which you can use to test the effectiveness of a classification strategy. You give it a classifier factory (as defined above) and two sets of data. It will train a classifier on one data set and test the results against the other, and then it will switch them and test again.

Given a list of data such as senate_people, you can divide it arbitrarily into two groups using the crosscheck_groups(data) function.

One way to measure the "distance" between Congresspeople is with the *Hamming distance*: the number of entries that differ. This function is provided as hamming_distance.

An example of putting this all together is provided in the lab code:

```
senate_group1, senate_group2 = crosscheck_groups(senate_people)
evaluate(nearest_neighbors(edit_distance, 1), senate_group1, senate_group2, verbose=1)
```

Examine the results of this evaluation. In addition to the problems caused by independents, it's classifying Senator Johnson from South Dakota as a Republican instead of a Democrat, mainly because he missed a lot of votes while he was being treated for cancer. This is a problem with the distance function -- when one Senator votes yes and another is absent, that is less of a "disagreement" than when one votes yes and the other votes no.

You should address this. Euclidean distance is a reasonable measure for the distance between lists of discrete numeric features, and is the alternative to Hamming distance that you decide to try. Recall that the formula for Euclidean distance is:

[(x1 - y1)^2 + (x2 - y2)^2 + ... + (xn - yn)^2] ^ (1/2)

- Make a distance function called euclidean_distance that treats the votes as high-dimensional vectors, and returns the Euclidean distance between them.

When you evaluate using euclidean_distance, you should get better results, except that some people are being classified as Independents. Given that there are only 2 Independents in the Senate, you want to avoid classifying someone as an Independent just because they vote similarly to one of them.

- Make a simple change to the parameters of `nearest_neighbors` that accomplishes this, and call the classifier factory it outputs `my_classifier`.

ID Trees

So far you've classified Democrats and Republicans, but you haven't created a model of which votes distinguish them. You want to make a classifier that explains the distinctions it makes, so you decide to use an ID-tree classifier.

`idtree_maker(votes, disorder_metric)` is a third-order function similar to `nearest_neighbors`. You initialize it by giving it a list of vote information (such as `senate_votes` or `house_votes`) and a function for calculating the disorder of two classes. It returns a classifier factory that will produce instances of the `CongressIDTree` class, defined in `classify.py`, to distinguish legislators based on their votes.

The possible decision boundaries used by `CongressIDTree` are, for each vote:

- Did this legislator vote YES on this vote, or not?
- Did this legislator vote NO on this vote, or not?

(These are different because it is possible for a legislator to abstain or be absent.)

You can also use `CongressIDTree` directly to make an ID tree over the entire data set.

If you `print` a `CongressIDTree`, then you get a text representation of the tree. Each level of the ID tree shows the minimum disorder it found, the criterion that gives this minimum disorder, and (marked with a +) the decision it makes for legislators who match the criterion, and (marked with a -) the decision for legislators who don't. The decisions are either a party name or another ID tree. An example is shown in the section below.

An ID tree for the entire Senate

You start by making an ID tree for the entire Senate. This doesn't leave you anything to test it on, but it will show you the votes that distinguish Republicans from Democrats the most quickly overall. You run this (which you can uncomment in your lab file):

```
print CongressIDTree(senate_people, senate_votes, homogeneous_disorder)
```

The ID tree you get here is:

```
Disorder: -49
Yes on S.Con.Res. 21: Kyl Amdt. No. 583; To reform the death tax by setting the
exemption at $5 million per estate, indexed for inflation, and the top death
tax rate at no more than 35% beginning in 2010; to avoid subjecting an
estimated 119,200 families, family businesses, and family farms to the death
tax each and every year; to promote continued economic growth and job creation;
and to make the enhanced teacher deduction permanent.:
+ Republican
```

```
- Disorder: -44
  Yes on H.R. 1585: Feingold Amdt. No. 2924; To safely redeploy United States
  troops from Iraq.:
  + Democrat
  - Disorder: -3
    No on H.R. 1495: Coburn Amdt. No. 1089; To prioritize Federal spending to
    ensure the needs of Louisiana residents who lost their homes as a result of
    Hurricane Katrina and Rita are met before spending money to design or
    construct a nonessential visitors center.:
    + Democrat
    - Disorder: -2
      Yes on S.Res. 19: S. Res. 19; A resolution honoring President Gerald
      Rudolph Ford.:
      + Disorder: -4
        Yes on H.R. 6: Motion to Waive C.B.A. re: Inhofe Amdt. No. 1666; To
        ensure agricultural equity with respect to the renewable fuels standard.:
        + Democrat
        - Independent
      - Republican
```

Some things that you can observe from these results are:

- Senators like to write bills with very long-winded titles that make political points.
- The key issue that most clearly divided Democrats and Republicans was the issue that Democrats call the "estate tax" and Republicans call the "death tax", with 49 Republicans voting to reform it.
- The next key issue involved 44 Democrats voting to redeploy troops from Iraq.
- The issues below that serve only to peel off homogenous groups of 2 to 4 people.

Implementing a better disorder metric

You should be able to reduce the depth and complexity of the tree, by changing the disorder metric from the one that looks for the largest homogeneous group to the information-theoretical metric described in lecture.

You can find this formula on page 429 of the reading (http://courses.csail.mit.edu/6.034f/ai3/ch21.pdf) .

- Write the `information_disorder(group1, group2)` function to replace `homogeneous_disorder`. This function takes in the lists of classifications that fall on each side of the decision boundary, and returns the information-theoretical disorder.

Example:

```
information_disorder(["Democrat", "Democrat", "Democrat"], ["Republican", "Republican"])
 => 0.0
```

```
information_disorder(["Democrat", "Republican"], ["Republican", "Democrat"])
 => 1.0
```

Once this is written, you can try making a new CongressIDTree with it. (if you're having trouble, keep in mind you should return a float or similar)

Evaluating over the House of Representatives

Now, you decide to evaluate how well ID trees do in the wild, weird world of the House of Representatives.

You can try running an ID tree on the entire House and all of its votes. It's disappointing. The 110th House began with a vote on the rules of order, where everyone present voted along straight party lines. It's not a very informative result to observe that Democrats think Democrats should make the rules and Republicans think Republicans should make the rules.

Anyway, since your task was to make a tool for classifying the newly-elected Congress, you'd like it to work after a relatively small number of votes. We've provided a function, `limited_house_classifier`, which evaluates an ID tree classifier that uses only the most recent *N* votes in the House of Representatives. You just need to find a good value of *N*.

- Using `limited_house_classifier`, find a good number *N_1* of votes to take into account, so that the resulting ID trees classify at least 430 Congresspeople correctly. How many training examples (previous votes) does it take to predict at least 90 senators correctly? What about 95? **To pass the online tests**, you will need to find close to the minimum such values for N_1, N_2, and N_3. Keep guessing to find close to the minimum that will pass the offline tests. Do the values surprise you? Is the house more unpredictable than the senate, or is it just bigger?

- Which is better at predicting the senate, 200 training samples, or 2000? Why?

The total number of Congresspeople in the evaluation may change, as people who didn't vote in the last *N* votes (perhaps because they're not in office anymore) aren't included.

Survey

Please answer these questions at the bottom of your `ps4.py` file:

- How many hours did this problem set take?
- Which parts of this problem set, if any, did you find interesting?
- Which parts of this problem set, if any, did you find boring or tedious?

(We'd ask which parts you find confusing, but if you're confused you should really ask a TA.)

Errata

If you find what you think is an error in the problem set, tell 6034tas@csail.mit.edu about it.

Retrieved from "http://ai6034.mit.edu/fall11/index.php?title=Lab_4"

- This page was last modified on 23 October 2011, at 00:04.
- *Forsan et haec olim meminisse iuvabit.*

6.034 Lab 4

10/30

CSP

Work in lab 4
But other code in csp.py
Need to figure out what is going on
behind the secens

Just build it with API
what is value being assigned to x?

Only when variable assignment
- Oh I see something else controlling testing
- this only tests

So a variable gets assigned here?
 L before calling?
 Yeah it handles DFS
We just check

So it gives a state
 L that is like our table?

②

X is Variable
+ type

gets values

then is assigned one

Then we get Constraints of that variable

L need .get_name()

Then we have some binary Constraints

- Check each one

- ok Y is the other value

- so get value ;(

- :(only work one way

- or in there twice
and

But get Domain seems to be failing ...

Oh need to turn Y string to variable

(Its all there - need to do it!

Put line break everywhere to see

Oh need check

(3)

What is CP again?

- cross things out
  - L we call reduce
- then when search have less to search

Seems to be working...

(✓) Passes test!

FC w/ Prop Singleton Domains

So this runs through FC
- ok that reduces

Then this looks at if 1 left

Same True if continue
False to back track) returning

Need ~~the~~ queue

FIFO

actually

↑ pop first value

(Nice eclipse helps)

4

(As I am writing this, I'm visualizing it
– better than last time)

Visited singletons too?

Find all constraints
└ then same as before

So it mostly works ...
– why does it stop?
Seems stuck in ∞ loop
– not poping?
Or re added instantly
– so need unvisited – ah I see now

How does their tree display?
– weird

So still bug in tester that neither is set
So X is assigned none
└ need to check for true as well
Oh need to assign it insted!
└ to singletons' only value

(5)

✓ correct

Learning Section

10/31

data already read

need nearest negipor

- returns classifier factory

    - makes classifier

        call w/ data pt and returns string

ok when call evaluate it tells you if you are right

hamming distance

└ pre built

euclidian distance

└ write this

$$\sqrt{(x1-y1)^2+(x_2-y2)^2}$$

See data format first

- list of 1s and 0s

(6)

Zip gets each character

(X) still incorrect

Need to treat votes as ¿high dimensional vectors?

like each vote is a dimension?

I don't see what is wrong!

Look at tests

1 2 3   4 5 6

want 5.19

So WA gives me 5.19 !

Test debugging by deleting other fns from tests.py!

- should have thought of earlier

its not squaring

↳ need $\cancel{**}$ $\cancel{*}$

Now not taking $\wedge\frac{1}{2}$

(✓) That took <u>way</u> <u>too</u> long!

7

Still gets 5 wrong answers !
Now change to params

~~- Still gets 8 ticton~~

Try param 3

works better

But its only running 2 times when 3 rules

ID Trees

Id tree maker

give it vote info and returns fn for call disorder

returns classifier factory
└ w/ instances Congress ID Tree

yes or no vote

can print a tree
(oh prebuilt)

Should do better one w/ info disorder -
not homogeneous disorder

(8)

(oh they have sudoku CSP too!)
(online tester still slow at end....)
So homogeneous returns # that are homogeneous

Disorder $\sum_{c} -\frac{n_{bc}}{n_b} \log_2 \frac{n_{bc}}{n_c}$

n_b = # in branch b

n_{bc} = # in branch b of class c

So our classes are yes and no?

So each branch is counted seperatly
than avg it

? So 4 logs
└ two for yes, two for no

Just R + D ?

Avg Disorder

$$\sum_{b} \frac{n_b}{n_t} \times \sum_{c} -\frac{n_{bc}}{n_b} \log_2 \frac{n_{bc}}{n_b}$$

n_t = # total

? better

9

Getting some math errors ....

hmm

Can't take $\log_2 0$

$-\infty$

We define as 0

- need to manually do I guess

(X) Some tests right

- others wrong

All my counts = 0

it still does!

float values?

ahhh

(X) still often wrong

but values no longer all 0

So [0, 2]

[0, 2]

Should be $\frac{1}{2}$

(✓) Needed floats there

(10)

So now 10/13 tests ~~online~~ off line
- takes 11 min for online to run ← weird since this is fast

Next is in the ~~wild~~ wild
limited_hase_classifier

Find N_1 to classify
How many training values
Guess the values

N1 Hase

---

10 → 288 correct ~~wait~~
want 480 ~~o~~

50 → ✓ works

N2 Senate

---

10 → 63
want 90
50 → 87
70 → ✓ correct

(11)

N3 Last senate

10 → 42
want 45
50 ✓ passes

✓ Passes offline

2 hrs - very short
for p-set

Now must wait to resubmit online....

N3 failed online ~~50~~
~~70~~
~~30~~
20 - too small on offline
23 ✓ Checkoff 5.0/5.0

6.034 Lecture

10/31

The Right Way (Chronology)

- ☐ Phonemes + distinctive features
- ☐ God, viewed as engineer
- ☐ Sussman - Yip machine
- ☐ Yip Sussman Rule learner
- ☐ Reflection
- * Hallucination Principle
- * Marr's catechism

One of most important lectures if in to 6.034

Halloween

Chocolate is one of the best legal highs you can have
- eat before exam

Saying "dogs" "cats"

vibration of vocal cord

no vibration

Has become natural - no one taught differently

He is distracted by laptops
- non verbal communication

(2)

Humans have 5 muscels to control speech

- Create wave form

But diff people's saying same words → wave form very different

Goes into an ear

└ becomes seq of phonemes

- meaning

Vector of distinctive feature

- 14 of them for each phonemes

2^{14} possible phonemes

eng has < 15

no lang has > 100

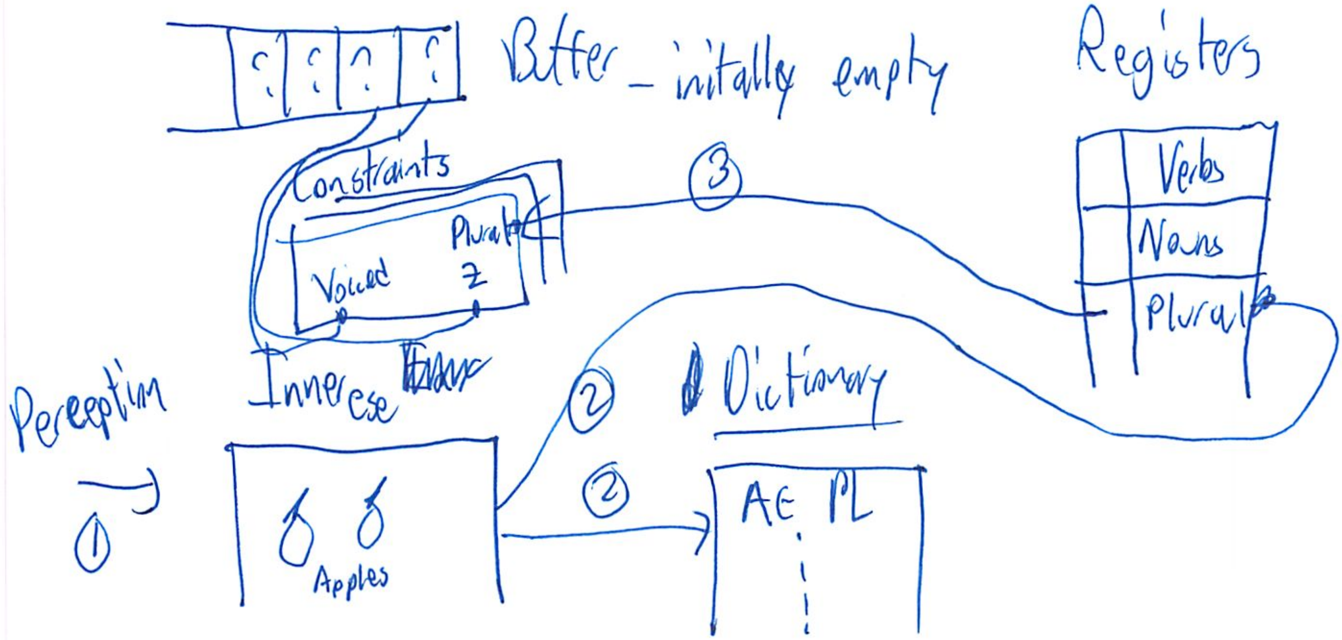
No one has been able to build distinctive feature system

Interplay in brain b/w meaning to ear + sequence

└ Hallucination principal

3

Sussman: How could I build a system like this?



- Constraint tries to force connection between register and buffer

3 Fill buffer

	AE	P	L

Do I have enough data on one to make a decision?

4 ← shift

AE	P	L	

Now 2 bins happy

5 Can we make everything happy

AE	P	L	Z

(4)

Works bi-directional

↳ from word to object

For every bundle of nerves going one way, some go the other way in humans

So draw dog and ~~cat~~ cat out as distinctive features

	KAT	(s)		DOG		
~~Slab~~ Slabic	−	+	−	−	+	−
Voiced	−	+	−	+	+	+
Continuent	−	+	−	−	+	+
	−	−	−	−	−	−

Strident – lots of white noise (s, z)

Phoneme

Each col is distinctive feature

Each word is sequence of phonemes ~~so phoneme~~

Each row is phoneme

Now how write a program to do this?

(5)

Search over a generations
  L patterns that must be matched for it to tell them apart
  - write don't cares

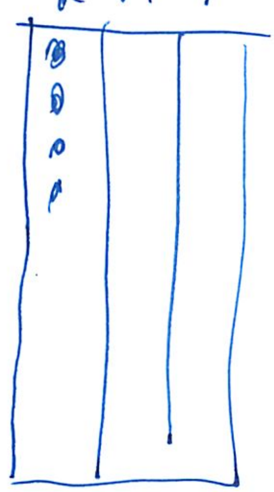
But 14.3 branching factor

So generalize from l to r
  L later phonemes matter more to determination plural

Seed w don't cares

k A T

till matches something



Do Beam (2) search

(6)

uncovered samples

Collect + − samples

↓

Pick + sample as seed

↓

Generalize Left + right → No more generalization is possible

+ → • − → •

↓ Rule

Reject No

Matches − sample

- may match

Computer example

---

(1) k E t d E k	(2) d Q g g E n	(3) b E J

↓ generalize down column

As soon as excluded < 3

- dont want to match other things

(1) Get s is end is not voiced + not strident − −

(2) Get z if end is voiced +

(3) Get es if end is strident +

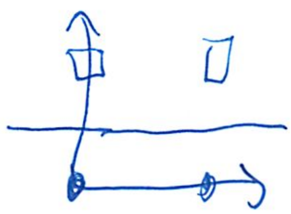
(7)

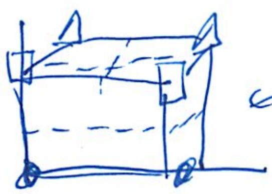
Works for a lot of stuff
 └ like past tenses

It was too good?

What aspects are important?

 Say 2 charastics/phonemes





← seperate w/ planes

Easy to seperate when sparse field

Marr's Catechism

1. Understand problem
2. Get a good representation
3. Develop a method/algorithm

Looks obvious
 But lots of people start at 3

8

What is a good representation

1. Makes something explicit
   - like in farmer, goose, grain it doesn't matter how tall farmer is
2. Exposes Constraint
3. Localness
4. Computable

At the time people were suffering w/ mechanism envy

Distinctive feature is right thing to make explicit

Show the constraint

Now need to exploit representation to solve problem

Does our model look biologically plausable?
   Representation is adaquet for task
   Not really bio plausable

<u>Vocab</u>

← Chomsky

Competance
- have knowlge needed

Performance
- actually compute it

9

Can have good competence theory w/o performance

This is a performance theory

Halucination principle: everything we do is
a Halucination principle
- people mix stuff they hear and stuff they think

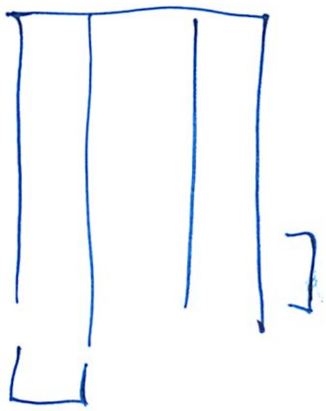
Profs are Working on this now

6.034 Tutorial

10/31

Today lecture incoherent

- Line drawing was most incoherent
- Can do line drawing from book



Phoneme

- basic unit of sound
- a letter or two

14 distinctive features for each phoneme

- a vector of the 14 describes a phoneme

	k	A	T
Voiced	−	+	−
Silabic	−	+	−
Strident	−	−	−
etc			
⋮			

this col always true for A phoneme

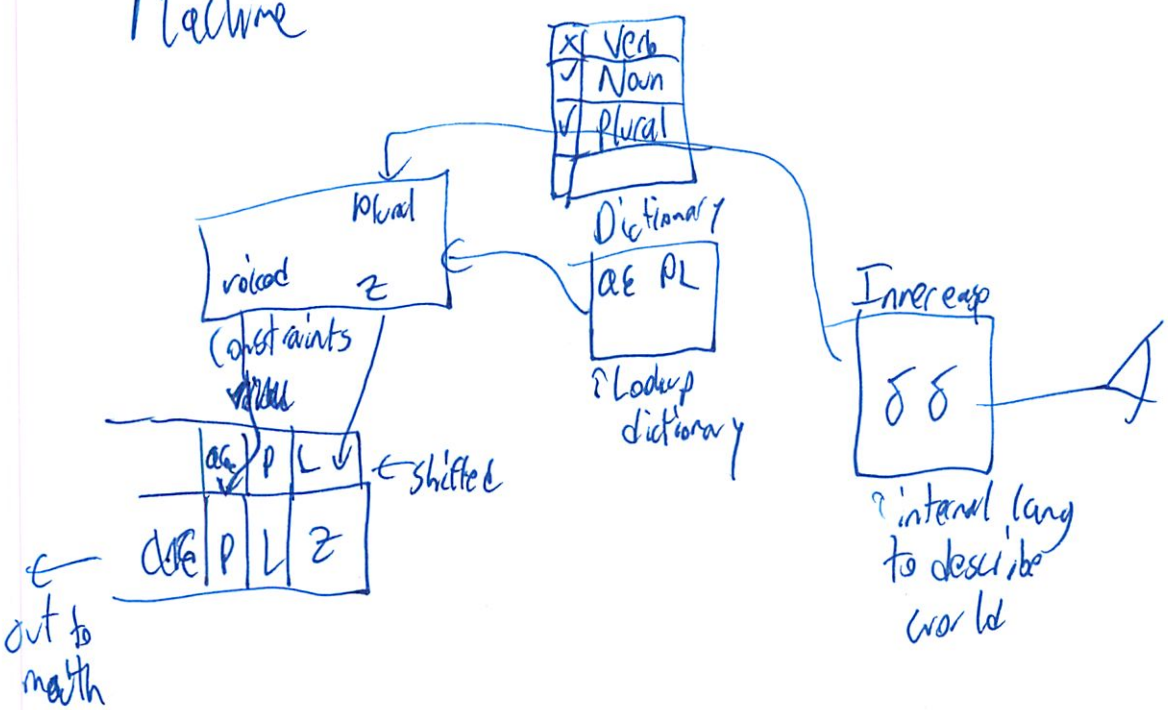
②

So I can build rules about ending sound if plural

Dots mean doesn't matter
- lay out KAT, DOG tables
- put dot where items are same

If have multiple examples for each ending sound
I can see that some differences are not important

Machine



Mostly conceptual
Once you see it - hopefully it makes sense

③

Go over quiz 2

- Very hard to get a 5 - need to miss < 1 on each problem
- Games
  - easest part
  - most like past tests
- Have to rerun search for oponent
  - Can't just write A+I
- If don't understand A,B - study
  - people did not prune enough
  - Only 9 nodes to actually evaluate

CSP

Not as straight forward

Propagating through domains reduced by any # care

- Only one prior exam

(4)

Interesting problem

A) 4^7 ← since problem was constructed like this

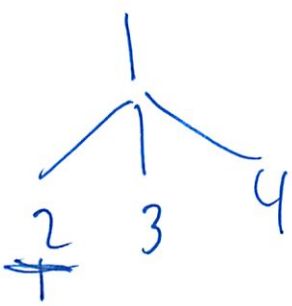
- not 7^4
- # nodes on tree $\frac{4^8-4}{3}$
- size was cage

B) Hardest part

- have given domain (pre-reduced)
- most people missed at least 5 pts

~~Constraints~~

- lose constraints



* most propage through any reduced domain

→ Propagate EL, H4

* try and see but does not work, fail, backtrack

- do you write the possible value and then cross out
  - like we did MA problem
    - state coloring

(3)

Only stop if that assignment is not possible

- don't extend node

~~Only check~~

If check through CP - we haven't checked on tree

Draw ~~BT~~ node if extend w/ CP ✓

CP w/ singleton ✓

CP w/ any reduced X

CA w/ any (ac)

Best rule

Think of code - if FC w/ CP reduces - (return False)

└ don't write

return True

└ write

At bottom Boar - must be w/ nearcat

So its a constraint

We crossed out 3,4 Boar in previous step (assigning friend)

→ Friendly constraint

(6)

Political problem - Palin wants ~~the~~ next to McCain

- when assigning one ?

↳ cross ~~oth~~ out the others' other positions

Crossing out domain is like the constraint prop before hand

~~That you can~~ Don't cross out Hyenna

- wording issue

Hyenna can only be w/ EL

but can be by himself

So can't cross 3 out till assign

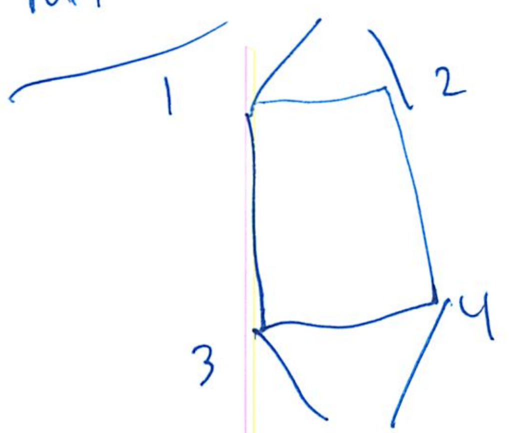
Someone else to it

Next tree is almost all singletons

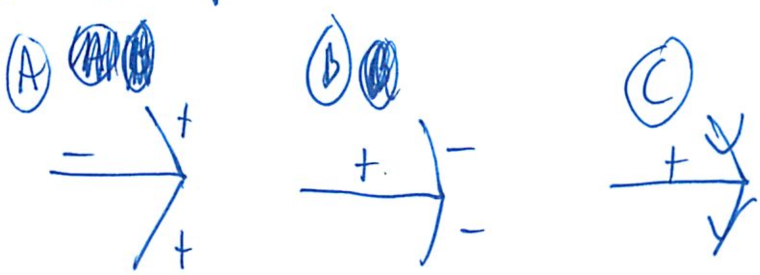
E) Accepted any valid constraint

7

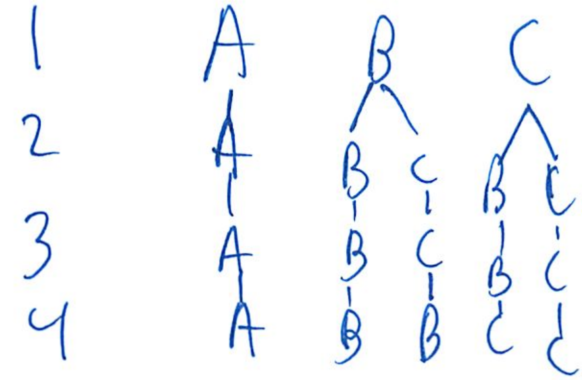
Part 3



Have 3 possibilites



Best way i draw a tree



Solving for every possibility

← count answer = 5

Classification of line drawings

So if 3 is A

does the other ones work

8

Arrow heads can be rotated to any possible angle

- Don't count these permutations

Should read book on this

- or could reason through

Test is long

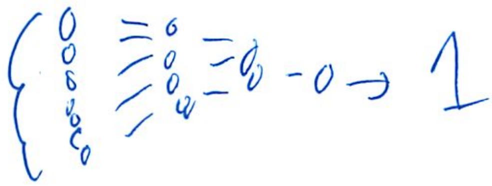
6.034 Recitation 11/3

1. Neural Networks
2. Examples what NN can do and can't
   - perceptions / XOR

<u>Next week</u> Support Vector Machines

So for recognizing a 1

Neural net for each



SVM actually throws out factors that don't matter
Does not happen in NN

Need a lot of training data

How to use? training
test ← set some aside to test
Can't train w/

80 90
20 10

(2)

Math to update weights

derivitive of perfomance function

$$P = -\frac{1}{2}(d-o)^2 \quad - \quad \frac{\partial P}{\partial o} = d - o$$

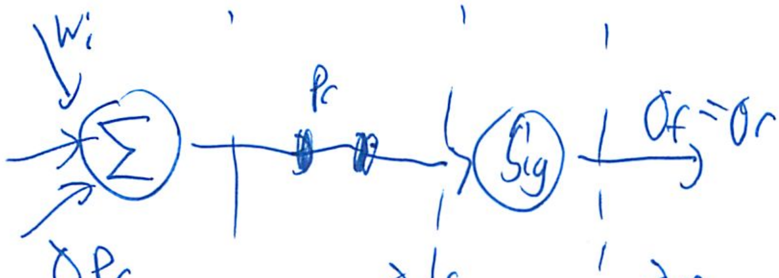
$$\frac{\partial P}{\partial w_i} = \frac{\partial P}{\partial w_i} \times \frac{\partial O_f}{\partial P_r} \times \frac{\partial P}{\partial O_f}$$

$$\Delta w_r = \overset{\text{(arbitrary}}{\alpha} \times L_f \times \delta_f$$

Δw_f calling

$$\Delta w_l = \alpha \times i_l \times \underbrace{O_l \times (1-O_l) \times w_r \times \delta_f}_{\delta_l}$$

$$sig = \frac{1}{1+e^{-x}}$$



$$\frac{\partial P_r}{\partial w_i} \qquad \frac{\partial o_f}{\partial p_r} \qquad \frac{\partial P}{\partial O_f}$$

← only if sigmoid

$$\underbrace{\left[o_f(1-o_f)\right] \quad (d-o_f)}_{} \rightarrow \delta_f$$

(3)

Forward chain the values

Then backwards chain the values

$$P = -\frac{1}{2}(d - o_f)^2$$

$$\xrightarrow{i_l} (\Sigma) \xrightarrow{p_l} (Sig) \xrightarrow{o_l = i_r} (\Sigma) \xrightarrow{p_r} (Sig) \xrightarrow{o_r = o_f}$$

$$\underbrace{o_f(1-o_f) \times (d - o_f)}_{} = \delta_f$$

etc repeat $\underbrace{o_l(1-o_l) \quad w_r \quad \ldots}_{} = \delta_l$

just to make calc look pretty

(Totall mess - just coping - not listening)

(Need to review on my own)

One deriv for each component of network

Add on new partials

$\Delta w_r = \alpha \times i_f \times \delta_f$

$\delta_o = 8 \times .5 \times \frac{1}{8} = .5$

that is weight change

$w_r' = w_r + \Delta w_r$ | weight new = weight old + .5 $\quad 0 + .5 = .5$

(4) Now next part

$$\Delta w_l = \alpha \times i_l \underbrace{o_l \times (1 - o_l) \times w_r \times \sigma_f}_{\sigma_l}$$

before update

$$= 8 \times 1 \times .5 \cdot (.5) \cdot 0 \cdot .5$$

$$= 0$$

So $W_1' = W_1 + \Delta w_1$

$0 = 0 + 0$

Looking at graphs is hard

Thats why do it w/ python

Getting closer to 1 but not fast

So ? α

8 is too small

Too big and you overshoot

Several rounds before you converge to 1

(5)

10,000 → 100,000 rounds possible

Don't Panic! on exam

What if change sigmoid to sum of 2

$$\frac{1}{1+e^{-\alpha x}} + \frac{1}{1+e^{-\beta x}}$$

Where will that change back propagation

Chane the deriv we had

<u>Old</u> $O_t(1-O_t)$

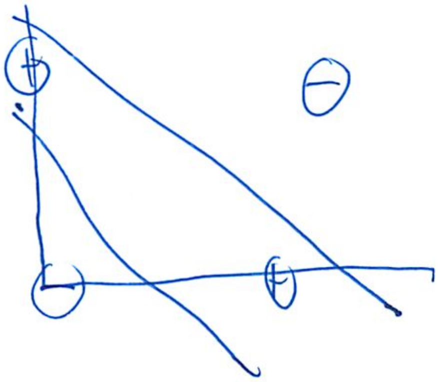
<u>new</u> --- (will be hw problem)

Must change both sigmoids

Applications of Neural Nets

p4 in handout

6



Classification problem

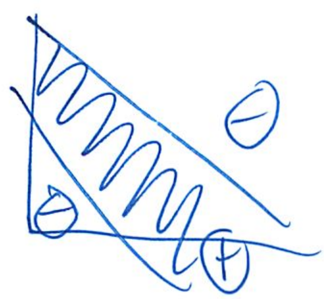
- like previous problems! k-NN, ~~or~~ Decision Trees, etc

X Y			aka
(1,0)	→	1	$\oplus$
(0,1)	→	1	$\oplus$
(1,1)	→	0	$\ominus$
(0,1)	→	0	$\ominus$

So as a binary fn this is XOR

But its non digital

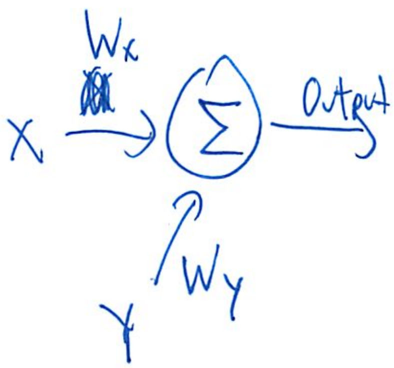
- ~~has~~ region where $\oplus$



7

What type of network do we need?

Perceptrons are single layers neural nets

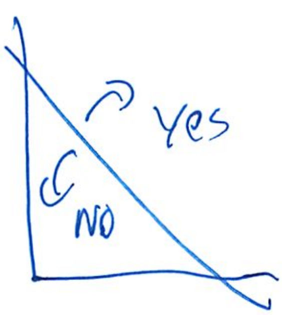


is it > c ?

What type of regions can this define?

In Patrick's book perceptrons
"beutiful math"

So only 1 cut
- can be horiz/vertical or combo since x + y



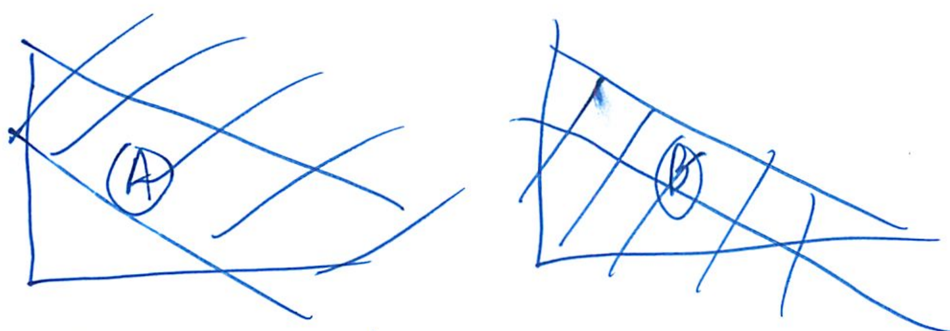
8

Can't use single ~~layer~~ layer ~~...~~ Neural net for the pattern on previous page

↓ opps Nearest Neighbor Neural Nets same

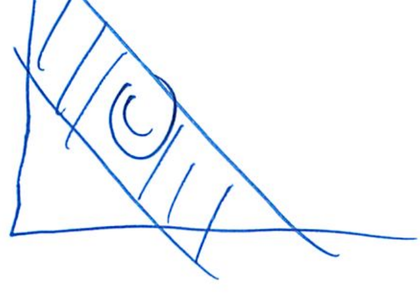
Need multi layer system

fully written out in packet



So C combines

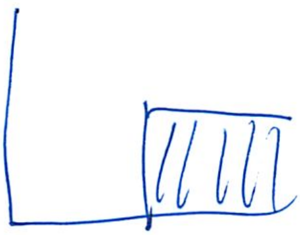
AND unit



Percitrion = 1 layer
└ No can't do it

2 layer system
└ yeah

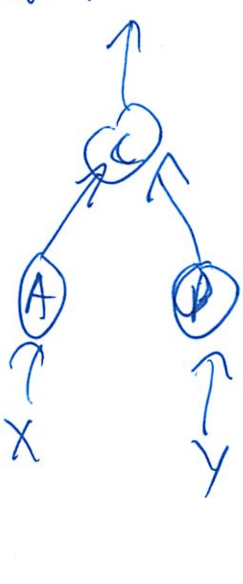
A bunch on p 7, 8



Perceptron No
2 layer Yes

$W_1 X + W_2 y + C = 0$
└ is possible

What about a system that does



← A, B only gets 1 variable
Can only do certain type of cuts
Can only do vertical + horizontal

(10)

How do you figure out what weights are?

(A) $W_{XA}x + W_{YA}y - W_A > 0$

(B) $W_{XB}X + W_{YB}y - W_B > 0$

(A) $y < -x + \frac{3}{2} \rightarrow -2x - 2y > 3$

(B) $y > -x + \frac{1}{2} \rightarrow 2x + 2y > 1$

~~mea~~

Now correspond coefficents

$W_{XA} = -2$ $W_{XB} = 2$ $W_A = 3$

$W_{YA} = -2$ $W_{YB} = 2$ $W_B = 1$

Can be ∞ valid values

But if want small ints $\rightarrow$ it's

Pg 8 - Sequence of shaded regions

Massachusetts Institute of Technology

Department of Electrical Engineering and Computer Science
6.034 Artificial Intelligence, Fall 2011
Recitation 8, November 3

Lots of typos
Will be re issued
later

Neural networks II

Prof. Bob Berwick, 32D-728

0. Introduction: the summary so far (and solutions from last time)

Summary of neural network update rules:

To update the weights in a neural network, we use *gradient ascent* of a performance function P by comparing what a network outputs given a sample data point and given the network's current weights, via **forward propagation**. We compare the network's **o**utput value against the **d**esired value in terms of the partial derivative of $P = -1/2(d-o)^2$ with respect to particular weights w_i. This is called **backpropagation.** Recall that the first step is to find the derivative of P with respect to the output, which turns out to be: (d–o).

The general formula for the change in weights is:

$$\Delta w \propto \frac{\partial P}{\partial w} \text{ so } w' = w + \alpha \times \frac{\partial P}{\partial w} \text{ where } \alpha \text{ is a rate constant (also } r \text{ in the literature \& quizzes)}$$

The value of alpha (aka r) is also the "step size" in the hill-climbing done by gradient ascent, using the performance fn.

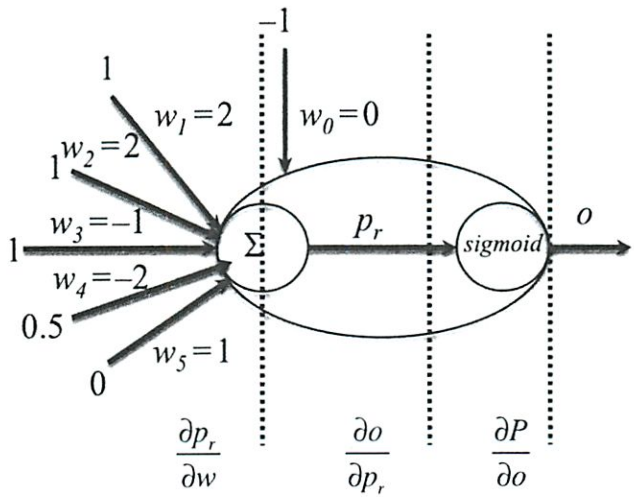
For the **final** layer in a neural network, whose output from forward propagation is o_f and where the desired output value is d, the required change in weight value for a (single) final weight is:

1. $\Delta w_r = \Delta w_f = \alpha \times \delta_f \times i_f$ where $\delta_f = o_f(1-o_f)\times(d-o_f)$

For the **previous** layer in a neural network (just the rightmost layer if a single neuron), the required update equation is:

2. $\Delta w_l = \alpha \times o_l(1-o_l)\times \delta_f \times i_l$

Example 1. Last time we computed the weight updates for a single-layer neural network with 6 inputs and 6 weights. Each partial derivative in the figure below corresponds to a different part of the network, with their product yielding the derivative of P with respect to the weights w, where the desired output was 1, and the learning rate alpha was (arbitrarily) set to 100:



Step 1: Forward Propagation. Calculate the output o given the input values shown. Useful data point: *sigmoid*(2)= 0.9

Answer: $-1 \times w_0 +$ __1__ $\times w_1$ __1__ $\times w_2 +$ __1__ $\times w_3 +$ __0.5__ $\times w_4 +$ __0__ $\times w_5 = p_r = 2$

Sigmoid $(p_r) = o_f =$ __0.9__

Step 2: Backpropagation to find delta for final, output layer.

$$\delta_f = \frac{\partial P}{\partial p_r} = \frac{\partial P}{\partial o}\frac{\partial o}{\partial p_r} = (d - o_f)\times[o_f(1-o_f)]$$

$$\frac{\partial P}{\partial w} = \frac{\partial P}{\partial o}\frac{\partial o}{\partial p_r}\frac{\partial p_r}{\partial w} = (d - o_f)\times[o_f(1-o_f)]\times i_f = \delta_f \times i_f$$

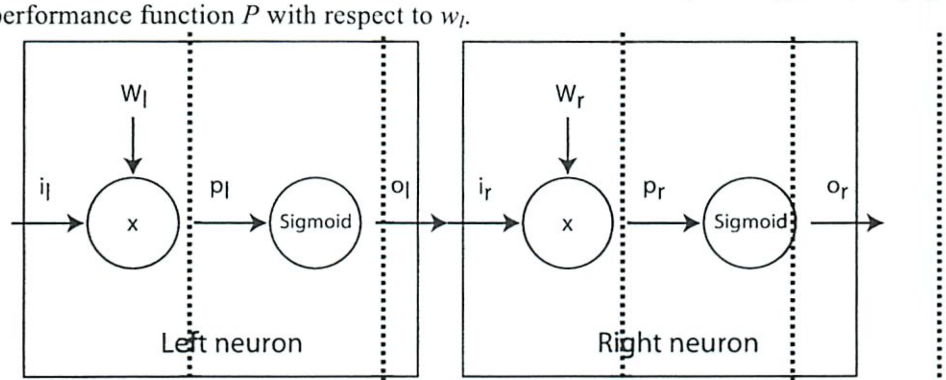
$\Delta w_f = \alpha \times i_f \times \delta_f$ (for each input line to the neuron, i)

$w_i' = w_i + \Delta w_f$ (for each input line to the neuron, i)

NEW WEIGHT	ORIGINALWEIGHT +	RATE ×	δ ×	INPUT =	NEW WT
w'	w	α	$(d\text{-}o)(o)(1\text{-}o)$	i_f	
$w_0' =$	0	100	0.009	−1	−0.9
$w_1' =$	2	100	0.009	1	2.9
$w_2' =$	2	100	0.009	1	2.9
$w_3' =$	−1	100	0.009	1	−0.1
$w_4' =$	−2	100	0.009	0.5	−1.55
$w_5' =$	1	100	0.009	0	1

Note how weights that have an input of 0 to them can't affect the performance, so they remain unchanged.
So, do these new weights get us closer to the desired output? If we run forward propagation again we can find out: $-1 \times 0.9 + 1 \times 2.9 + 1 \times 2.9 + 1 \times -0.1 + 0.5 \times -1.55 + 0 \times 1 = 6.65$; sigmoid(6.65) = 0. 99870765

Example 2. Last time, we also computed the value for the weight change for the final, output neuron of the simple two-neuron net below. We initially have i_l=1; both weights w_l and w_r= 0; and the desired output is 1. We will finish up this problem now.
For completeness, and to cement our understanding, let's see how the various terms in the partials are arrayed over this two-neuron diagram, pushing back from the output o_r, so you can see why it is called **backpropagation.** Make sure you **understand** where each of the five terms comes from. Multiplied together, they give the partial derivative of the performance function P with respect to w_l.



$\frac{\partial p_l}{\partial w_l}$	$\frac{\partial o_l}{\partial p_l}$	$\frac{\partial p_r}{\partial o_l}$	$\frac{\partial o_r}{\partial p_r}$	$\frac{\partial P}{\partial o_r}$
i_l	$o_l(1-o_l)$	w_r	$o_r(1-o_r)$	$(d-o_r)$
left weight deriv	left sigmoid deriv	right weight deriv	right sigmoid deriv	output deriv

This is to find the partial of the performance function P with respect to the left right, w_l.
Remember that to find the corresponding partial for the **final** output layer we computed something a bit different: the partial of p_r with respect to o_l is **replaced** with the partial of p_r with respect to w_r (so finding the partial of P with respect to w_r.) But this partial is just the derivative of $i_r \times w_r$ with respect to w_r, which is simply i_r.

Assume **all initial weights are 0** (it is actually a bad idea to set all initial weights the same for neural nets; why?). Assume a sample input of $i_l = 1$, and that the *desired* output value d is 1.0. Assume a learning rate of 8.0. (Useful data point: *sigmoid*(0) = 0.5) Let's run one step of backpropagation on this and see what's different about this case. First, as before, we must carry out **forward** propagation: compute the inputs and outputs for each node.

Step 1: Forward Propagation. OK, you should know the drill by now. First compute the outputs z at each node:

$p_l = w_l i_l$ = 0 × 1 = 0 So o_l (= i_r) = *sigmoid*(0) = 0.5

$p_r = w_r\, i_r$ = 0 × 0.5 = 0 So o_r = *sigmoid*(0) = 0.5

Step 2: Calculate the δ for the output, final layer, δ_f (i.e., the neuron on the right, for use in changing w_r)

Recall the formula for δ_f is: $o_r \times (1-o_r) \times (d-o_r)$= 0.5 × (1–0.5) × (1–0.5) = 0.125 .
Recall that $d = 1.0$; we have just computed o_r.
So, the change in the right-most weight w_f is: $\alpha_f \times i_r \times \delta_f = 8.0 \times 0.5 \times 0.125 = 0.5$

Step 3: Calculate δ_l for the hidden neuron on the <u>left</u>, recursively using the delta from the previous layer:
$\delta_l = o_l(1-o_l) \times w_r \times \delta_f$ = 0.5(1–0.5) × 0.125 = 0.03125
Now use this value to compute the weight change for the left neuron:
$\Delta w_l = \alpha \times i_l \times \delta_l = 8.0 \times 1 \times 0.03125 = 0.25$

Thus the two **new weights** are:
$w_f' = 0 + 0.5 = 0.5$
$w_l' = 0 + 0.25 = 0.25$

Let's see how much closer this has gotten us to the desired output value of 1.0. We do this by another round of forward propagation (and then typically, we would do back-propagation again to get us even closer, many thousands of times.) Your turn now.... (See the tear-off page on the back to estimate the sigmoid to 2 decimal places, or better, user a calculator or python on your laptop...)

Next iteration, forward propagation:
$p_l = w_l i_l$ = 0.25 × ____ = ____ So o_l (=i_r) = *sigmoid*(____) = ________.

$p_r = w_r i_r$ = 0.5 × ____ = ____ So $o_r = o_f$ = *sigmoid*(____) = ________.
So, we have definitely gotten a bit closer to our output goal of 1.0.

Next iteration, back-propagation:
Now you try it:

$\delta_f = o_f \times (1-o_f) \times (d-o_f)$ = ____ ×(1– ____) ×(1– ____) = ________.

$\delta_l = o_l \times (1-o_l) \times w_r \times \delta_f$= ____ ×(1– ____) × 0.5 × ____ = ________.

$\Delta w_f = \alpha \times i_f \times \delta_f$ = 8.0 × ____ × ____ = ____

$\Delta w_l = \alpha \times i_l \times \delta_l$ = 8.0 × 1.0 × ____ = ____

$w_f' = w_f + \Delta w_f =$ 0.5 + ______________ =

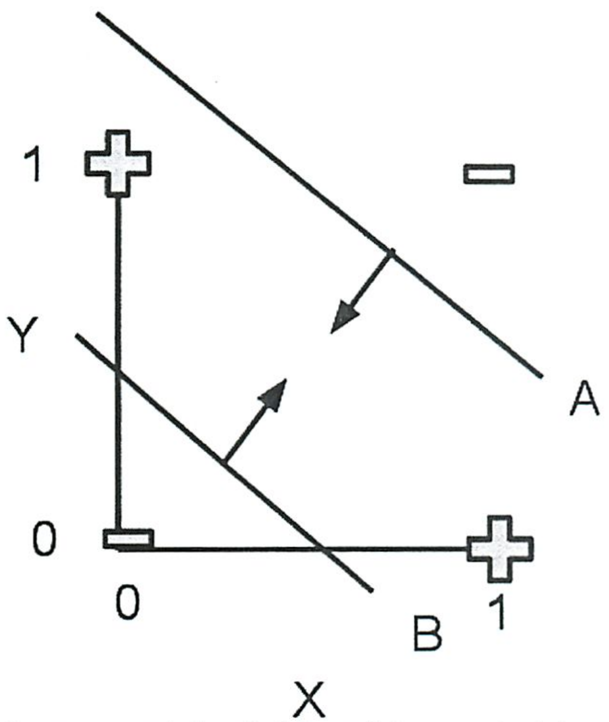
$w_l' = w_l + \Delta w_l =$ 0.25 + ______________ =

Do the new weights get us closer to the goal? Calculate this by **forward propagation** again:

$p_l = w_l i_l =$ 1 × ; o_l (= i_r) = sigmoid( ) =

$p_r = w_r i_r =$; $o_r =$ sigmoid( ) =

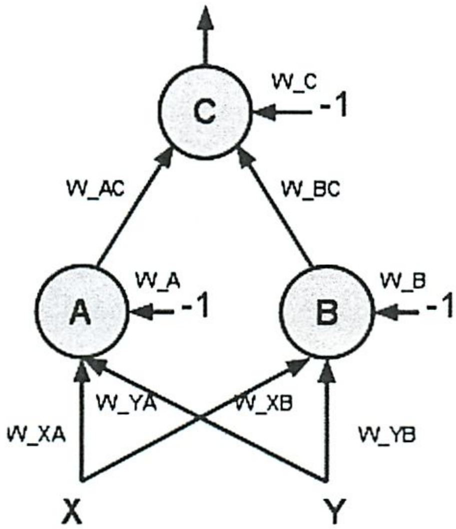
Example 3. What multilayer neural networks can learn that single layer networks cannot learn.
Why did people invent multi-layer neural networks? Consider a classification problem such as the one depicted below, which represents the predicate or the 'concept' of excusive-OR (XOR), i.e., the value of this function is 1 if either of the two inputs is 1; and the value of this function is 0 if both inputs are 0 *or* both inputs are 1:



Suppose we tried to find the weights to a *single* layer neural network to 'solve' this classification problem. Then the general formula for this network would be, output = $w_1x + w_2y + c$. But what weights would work? Do you see that this kind of equation can only define a *single* line? Thus, it says that we must classify the + and – regions in the graph above into regions that are all + (positive) and all – (negative), by making a *single* cut through the plane. Can this be done? Try it – why can't it be done?

Question: Can a perceptron encode function $|x–y| <$ *epsilon*, for some positive epsilon? Why or why not?

However, if we are allowed *two* network layers, then we *can* formulate a set of weights that does the job. Let's see how, by considering the network below, and then finding the weights that do the job. (This was a sample quiz problem previously.)



Step 1. First, think of input-level neurons (neurons A and B) as defining *regions* (that divide positive data points from negative data points) in the *X*, *Y* graph. These regions should be depicted as linear boundary lines with arrows pointing towards the positive data points. Next, think of hidden level neural units (neuron C) as some logical operator (a linearly separable operator) that combines those *regions* defined by the input level units.
(We will see later on a few more examples of this sort to show you how multi-layer networks can 'carve up' regions of the plane in this way.)

So in this case: units A, and B represent the **diagonal** boundaries (with arrows) on the graph (definition two distinct ways of separating the space). Unit C represents a logical AND that intersects the two regions to create the bounded region in the middle.

Step 2. Write the line equations for the regions you defined in the graph.

A) The boundary line equation for the region defined by line A:

$$y < -1 \times x + 3/2$$

B) The boundary line equation for the region defined by line B:

$$y > -1 \times x + 1/2$$

Step 3. Rewrite the line equations into the form: $ax + by > c$, where a, b, and c are integers:

A) $y < -1 \times x + 3/2$
$x + x \quad < 3/2$
$-2x + -2y \quad > 3$

B) $y > -1\ \mathrm{x} + 1/2$
$x + y \quad > 1/2$
$2x + 2y \quad > 1$

Now note that the sum of the weights times the inputs for each unit can also be written in a similar form. (We will call this summed product of weights times the inputs for a neuron its "z" value).

For Unit A: $z =$

$$W_{XA}\,x + W_{YA}\,y + W_A(-1) > 0$$
$$W_{XA}\,x + W_{YA}\,y > W_A$$

For Unit B: $z =$

$$W_{XB}\,x + W_{YB}\,y + W_B(-1) > 0$$
$$W_{XB}\,x + W_{YB}\,y > W_B$$

Why do we set $W_{XA}\,x + W_{YA}\,y + W_A(-1) > 0$ and not < 0? Look at the graph on the tear-off sheet!
When $z = W_{XA}\,x + W_{YA}\,Y + W_A(-1) > 0$, then sigmoid($z > 0$), and z grows and approaches 1, which corresponds to the *positive* points
When $z = W_{XA}\,x + W_{YA}\,Y + W_A(-1) < 0$, then sigmoid($z < 0$), z decreases and approaches 0, which corresponds to the negative points.
Thus, when expressed as > 0 the region is defined as **pointing towards** the positive points.
But when expressed as < 0, the region is defined as pointing towards negative points.
We want the defined region to point to the positive points. That is why we pick the inequality as $>$.

Step 5. Easy! Just read off the weights by correspondence.

$-2\;x\;\;+ -\;2\,y > 3$ line A's inequality
$W_{XA}\,x + W_{YA}\,y > W_A$ z equation for unit A. Therefore, $W_{XA} = -2$ $W_{YA} = -2$ $W_A = 3$

$2\,x\;+\;2\,y > 1$ line B's inequality
$W_{XB}\,x + W_{YB}\,y > W_B$ z equation for unit B. Therefore, $W_{XB} = 2$ $W_{YB} = 2$ $W_B = 1$

Step 6. Solve the logic in the second neuron layer
We now want to compute (A AND B), for the next layer. So we build a Truth table and solve for the constraints!

A	B	desired output	Equations	Simplified
0	0	0	$-W_C < 0$	$W_C > 0$
0	1	0	$W_{BC} - W_C < 0$	$W_{BC} < W_C$
1	0	0	$W_{AC} - W_C < 0$	$W_{AC} < W_C$
1	1	1	$W_{AC} + W_{BC} - W_C > 0$	$W_{AC} + W_{BC} > W_C$

We notice the symmetry in W_{BC} and W_{AC}, so we can make a guess that they have the same value:
$W_{BC} = 2$ and $W_{AC} = 2$

Then the inequalities in the table above condense down to the following:
$W_C > 0$ $W_C > 2$ (twice) $W_C < 2{+}2 = 4$

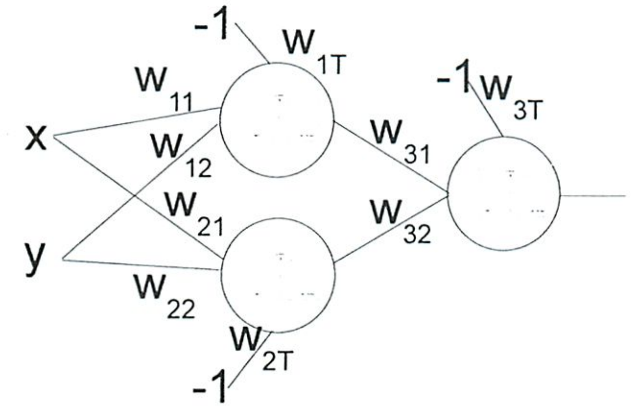
Therefore, $2 < W_C < 4$. Let's make life easy and pick $W_C = 3$. This gives us one acceptable solution:
$W_{BC} = 2$ $W_{AC} = 2$ $W_C = 3$

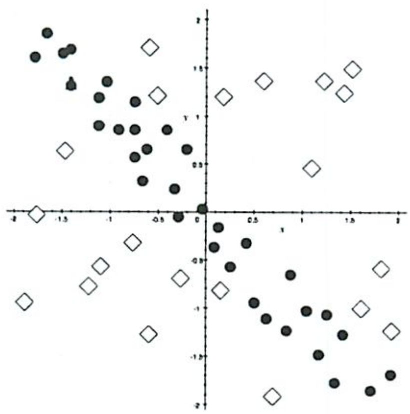
Of course, there are many solutions. The following solution also works, because it still obeys the inequalities and the constraints in the table:
$W_{BC} = 109$ $W_{AC} = 109$ $W_C = 110$

Quizzes will always ask for the *smallest* integer solutions.

This particular problem also illustrates how to combine networks using a logic gate. Thus, to compute more complex regions, we need more neurons either at one level or at the output level. But first, to cement our understanding of this problem, let's look at a related quiz problem, from quiz 3, 2009.

Given this three-node neural network, and the training data on the right





Question: which of the following sets of weights will correctly separate the dots from the diamonds? (Think about what cuts the various weights make at the left neuron....)

Weight set A:

w_{11}	w_{12}	w_{1T}	w_{21}	w_{22}	w_{2T}	w_{21}	w_{22}	w_{2T}
−2	−2	−1	3	3	−1.5	128	128	173

Weight set B:

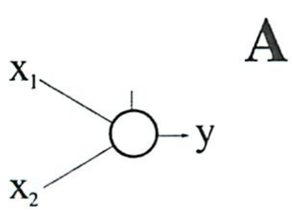
w_{11}	w_{12}	w_{1T}	w_{21}	w_{22}	w_{2T}	w_{21}	w_{22}	w_{2T}
2	−1	1	2	-2	1	100	100	50

Why does weight set A work but not weight set B?

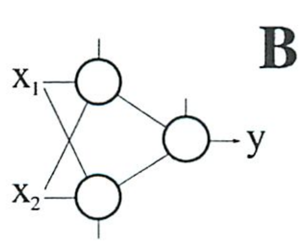
Example 4. Some other examples of carving up the *x-y* plane & the associated multi-layer networks
Now let's consider some other patterns in the *x-y* (or x_1, x_2) plane and what sort of qualitative network might be required to encode them. (This was an exam question in 2008.)

First, let's give the schematic pictures for (i) a perceptron; and then (ii) the simplest 2-layer neural net we have just seen – note that we have removed all the clutter of the *w*'s, etc.:

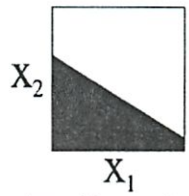
(A) Perceptron:



(B) Simplest 2-layer neural net:

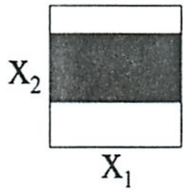


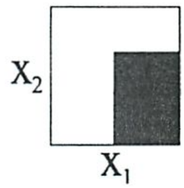
Here is a basic picture of the kind of classification regions a perceptron (A) can describe: any single cut, at any angle:



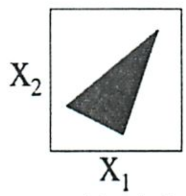
4.1 Question: Can a 2-layer network (B) also describe such a classification region? Why or why not?

4.2 Now consider these two sorts of classification regions.
Question: Can a perceptron (net A) describe these kinds of regions? Can the 2-layer network (B) also describe these kinds of regions? Why or why not?





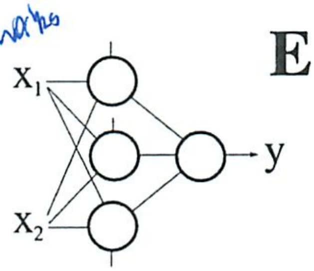
4.3 Now let's hone our intuitions by making the region more complex, and by considering different neural networks.
Question: The 2-layer network (B) cannot **describe this kind of region. Why not?**



3 cuts

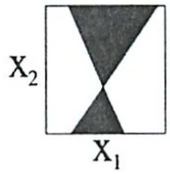
So, we must complicate our neural network to capture this kind of more complex region.
Question: Please explain *why* the following neural network *can* successfully describe the region just above. (Think about how we classified the region in our worked-out example earlier.)

works



Can make 3 cuts w/ x_1, x_2

Question: This network *cannot* successfully describe the region below. Why not? (Think about this, and for the next recitation, try to come up with the reason, and a modification, that is, a more complex neural network, that can describe this region.

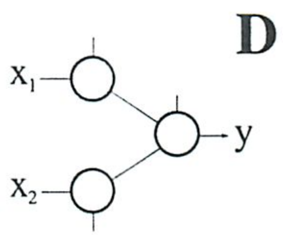


NAND

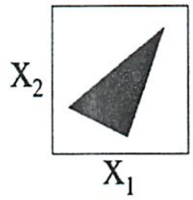
– next time

4.4 Finally, let us consider a *simpler* two layer neural network, where the inputs to the top, leftmost hidden neuron receives input *only* from x_1, and the bottom, leftmost hidden neuron receives inputs *only* from x_2. So the network looks like the following. Can you intuit how this will restrict what regions the network can describe?

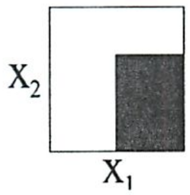
2 logic units
– combine outputs



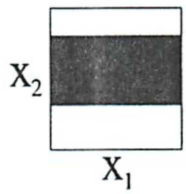
Question: Can this (restricted) neural network classify the region below? Why or why not?

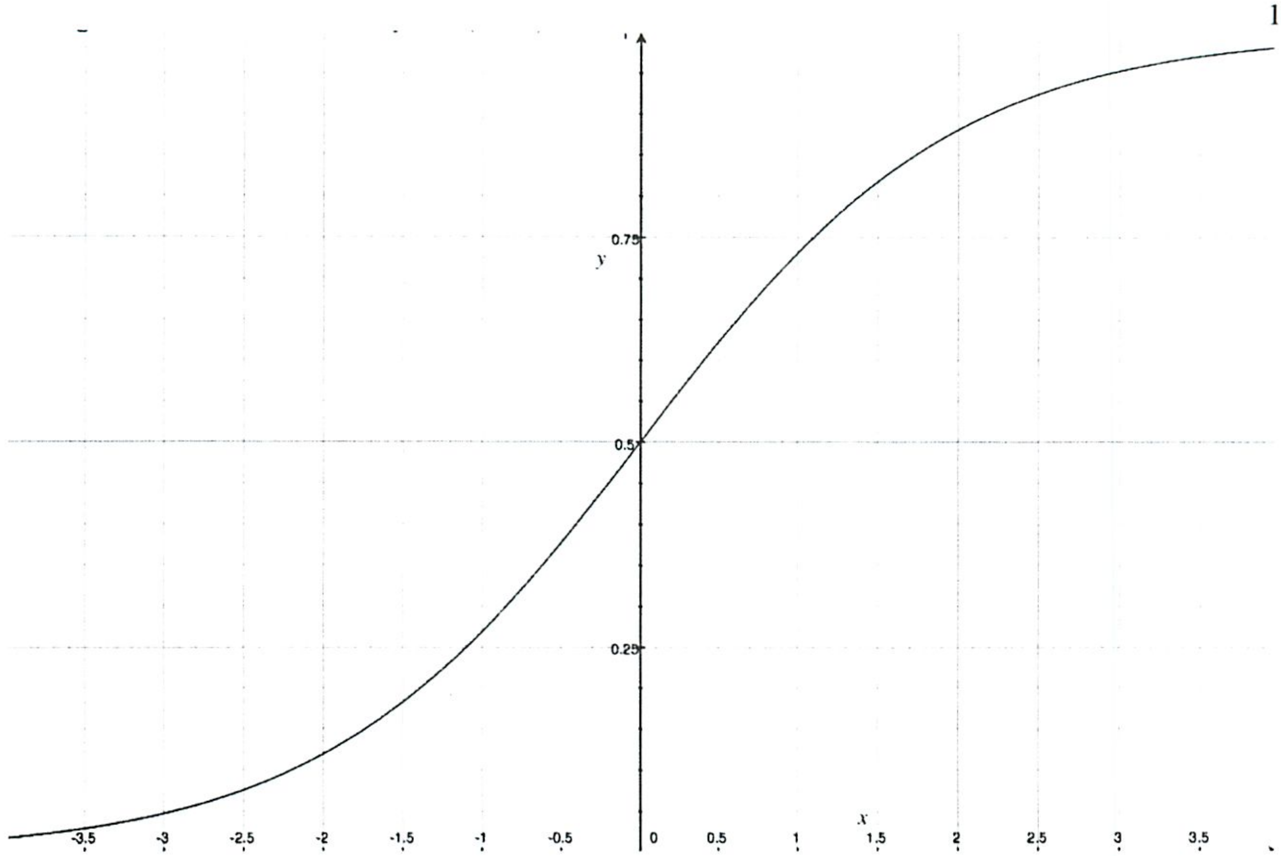


Can network (D) describe this region that we already saw above? Why or why not?



Finally, for next time, you might want to think about *why* network (D) **cannot** describe this region that we saw before (while we have already discussed what the usual 2-layer network (B) can do in this case):

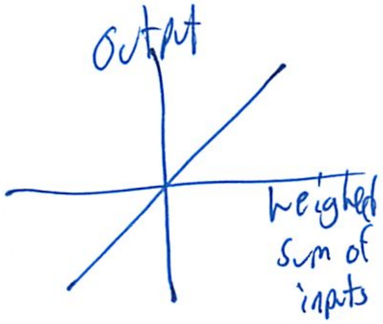


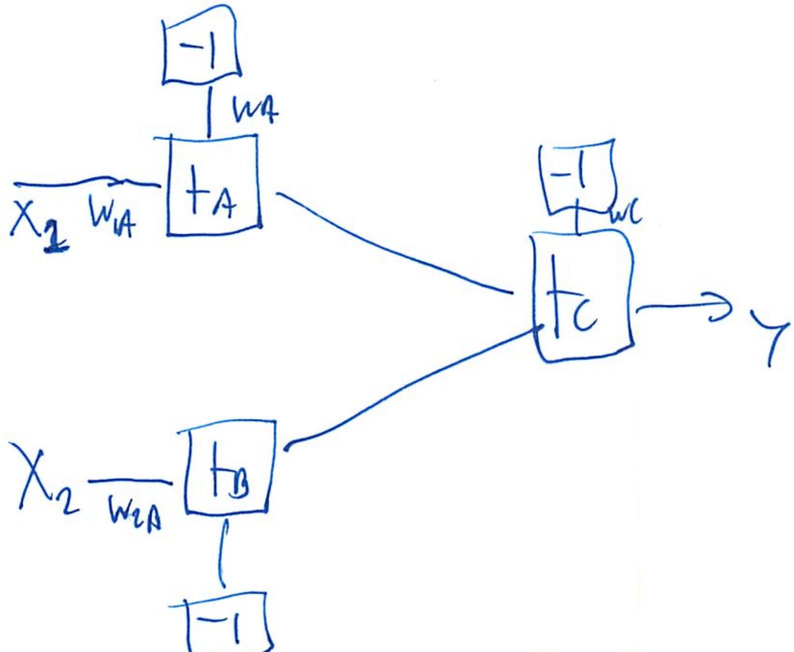


Sigmoid function $y = 1/(1-e^{-x})$

6.034 Tutorial 11/[illegible]7

Neural Networks





Just uses adder - no sigmoid
Contines to use standard performance fn

$$P = -\frac{1}{2}(\underset{\text{optimal}}{y^*} - y)^2$$

Can't use normal formula here for δ_C

(2)

$$\nabla_f = \frac{\partial P}{\partial z}$$

weighted sum function

Now need

$$\frac{\partial P}{\partial z}$$ = can't solve directly

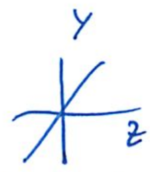
Series of chain rules

$$= \overset{\text{know}}{\frac{\partial P}{\partial y}} \cdot \frac{\partial y}{\partial z} \leftarrow \text{know}$$

only thing in P equation

so add more chain rules or solve it

$$= (y^* - y) \cdot 1$$

since 

if we label graph

before deriv of sigmoid $(y(1-y))$

$$= (y^* - y)$$

(3)

What is new equation for ~~[illegible]~~ δa

└ notes they gave you are mostly useless

$$\delta_i = \text{output}_i (1 - o_i) \sum_j w_{ij} \delta_j$$

this is slighty different for us

$$\delta_{a_i} = \underline{\quad 1 \quad} \sum_j w_{ij} \delta_j$$

$$\delta_A = W_{AC} (y_* - y)$$

$$\frac{\partial P}{\partial z_A} = \frac{\partial P}{\partial y} \frac{\partial y}{\partial z_a}$$

↑ same result as before

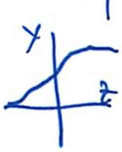
↑ z is ~~output~~ input of sigmoid

- here just

$$= (y_* - y) \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial z_a}$$

$$= (y_* - y) \cdot 1 \cdot \frac{\partial z}{\partial z_a}$$

Normally



$$z = W_{AC} \cdot z_a + W_{BC} \cdot z_B - W_C$$

↑ but shortcut w/ line graph

$$\text{general} \quad z = W_{AC}\, O_{AC} + W_{BC} \cdot O_{AB} - W_C$$

* Here $z = y$

~~[illegible]~~

(9)

$$= (y_* - y) \cdot 1 \cdot \frac{\partial z}{\partial A_C} \frac{\partial A_C}{\partial z_A}$$

$\partial A_C \downarrow W_{AC}$

$$= (y_* - y) \cdot 1 \cdot W_{AC} \cdot 1$$

key insight: we did this earlier

$$\delta_A = (y^* - y)\, W_{AC}$$

same as earlier

Here $\delta_i = \underbrace{\cancel{\quad}}_{\text{since this is 1}} \sum_j w_{ij}\, \delta_j$

like we saw earlier

So write our rules

final nodes $\rightarrow$ $\delta_f = (y_* - y)$ ← weights don't affect this

W_c is for -1

$\delta_i = (y_* - y)\, W_{ic}$ - just wrong formula here

internal node

(5)

So we were told

$W_i = 1$

$W_c = -.5$

$X_1 = X_2 = 1$

$r = 1$ ← learning constant aka α

So then our equation

$W_i' = W_i + \alpha\, i_i \cdot \delta_i$

↑ input of node

So first do normal foward propagation

$Y = \cancel{\;}$

$(X_1 \cdot W_{1A} + (-1) W_A) \cdot W_{AC} + (X_2 \cdot W_{2B} + (-1) W_B) \cdot W_{BC}$
$+ (-1) W_C$

$= (1 \cdot 1 + (-1) 1) \cdot 1 + (1 \cdot 1 + (-1)\, 1) \cdot 1$
$+ (-1) \cdot .5$

$= +.5$

(6)

Next backwards propagation

Find all the new weights

$W_c' = W_c + r \cdot i_t \cdot \delta_c$

↑ we called δ_f since its final node

↑ The input being modified by that weight

$= -.5 + 1 \cdot -1 \cdot (y^* - y)$

$= -.5 + 1 \cdot -1 \cdot (1 - .5)$

↑ input W_c is converting

$=$

$W_{AC}' = W_{AC} + r \cdot i_{AC} \cdot \delta_i$

$= W_{AC} + r \cdot (X_1 \cdot W_{1A} + (-1) W_A) \cdot (y^* - y) W_{AC}$

↑ input is A

$= 1 + 1 \cdot 0 \cdot .5$

$W_{1A} = 1 + 1 \cdot 1 \cdot (.5)$

$= 1.5$

Then can re forward propagate, ~~and~~ etc. Repeat.

No graphs.

~~[illegible]~~ 6.034 OH

11/8

Line Labeling

- specific case constraint prop
- think about more rigorously
  - don't think it about what it might be
- 3 face worlds
  - not all are
    - in class 6 face world
    - can do 8, 15, etc
- read chap on line labeling

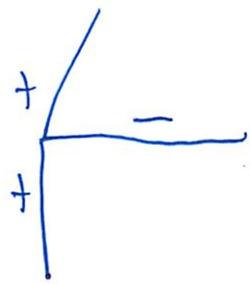
---

label each corner to one of 3

- ~~[illegible]~~ variables : the 4 corners
- domains : 3 possibilities
- very small space - so could do full tree
  - but don't need
  - but obvious

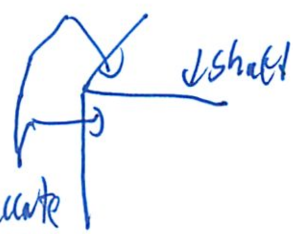
② Obvious way

1. ~~Tep~~ Temp assign UL to A



Since  2 acute angles

shaft

So then  shaft

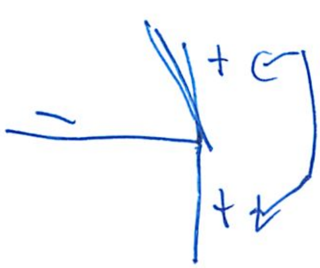
2 acute angles

So temp ~~[illegible]~~ assign that

2. Go UR

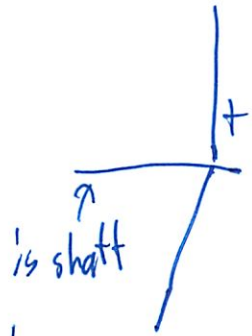
which of 3 fit

must share shaft which is —

Have  can say that

(3)

3. BQ



Since 2 prongs
Come back from shaft

otherwise ~~fork~~  fork

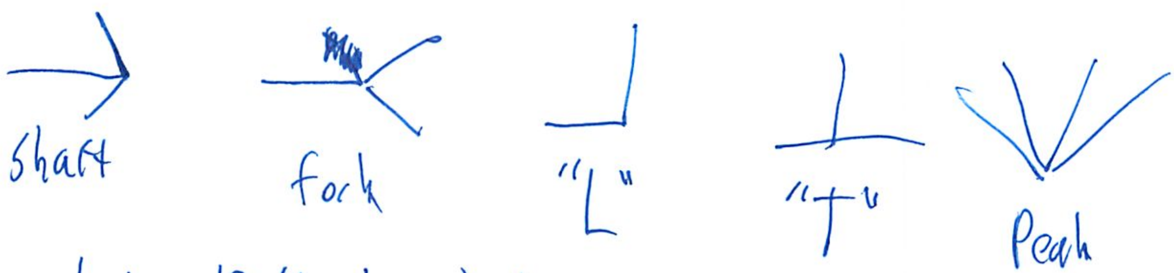
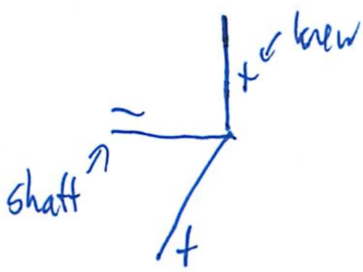


chart 12.4 in book

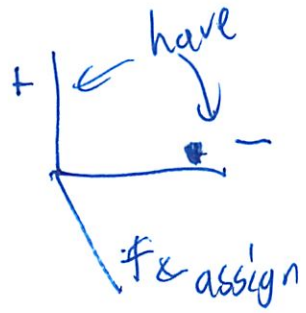
etc

Could assign A again



4

4. BL



0 works

1 possibility

+ vs 7

both say what line is

bounries can't see past

~~Looking~~

Leaning head on table looking off table

just see boundry edge

Labled w/ → w/ stuff on right

?table

(5)

If sitting in chair back a few feet looking
then looks like

✓ top edge looks + * since we can see 2 faces

bottom edge can't see 2nd face so arrow

+ can be on either side.

This problem is simple so B fails on ~~first step~~ step 2

Try for fail

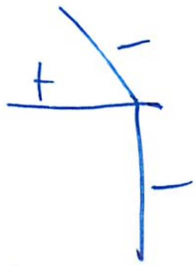
1. TL

2. TR try to assign A

? Contidiction

(a)

2b. Try (B) on TR



Could continue

Because we did fail at 1 point - There are some constraints

So must enumerate each way
 └ we did in tutorial

7

A,B Search

(Confused so many ways to ~~[illegible]~~ describe)

~~[illegible]~~

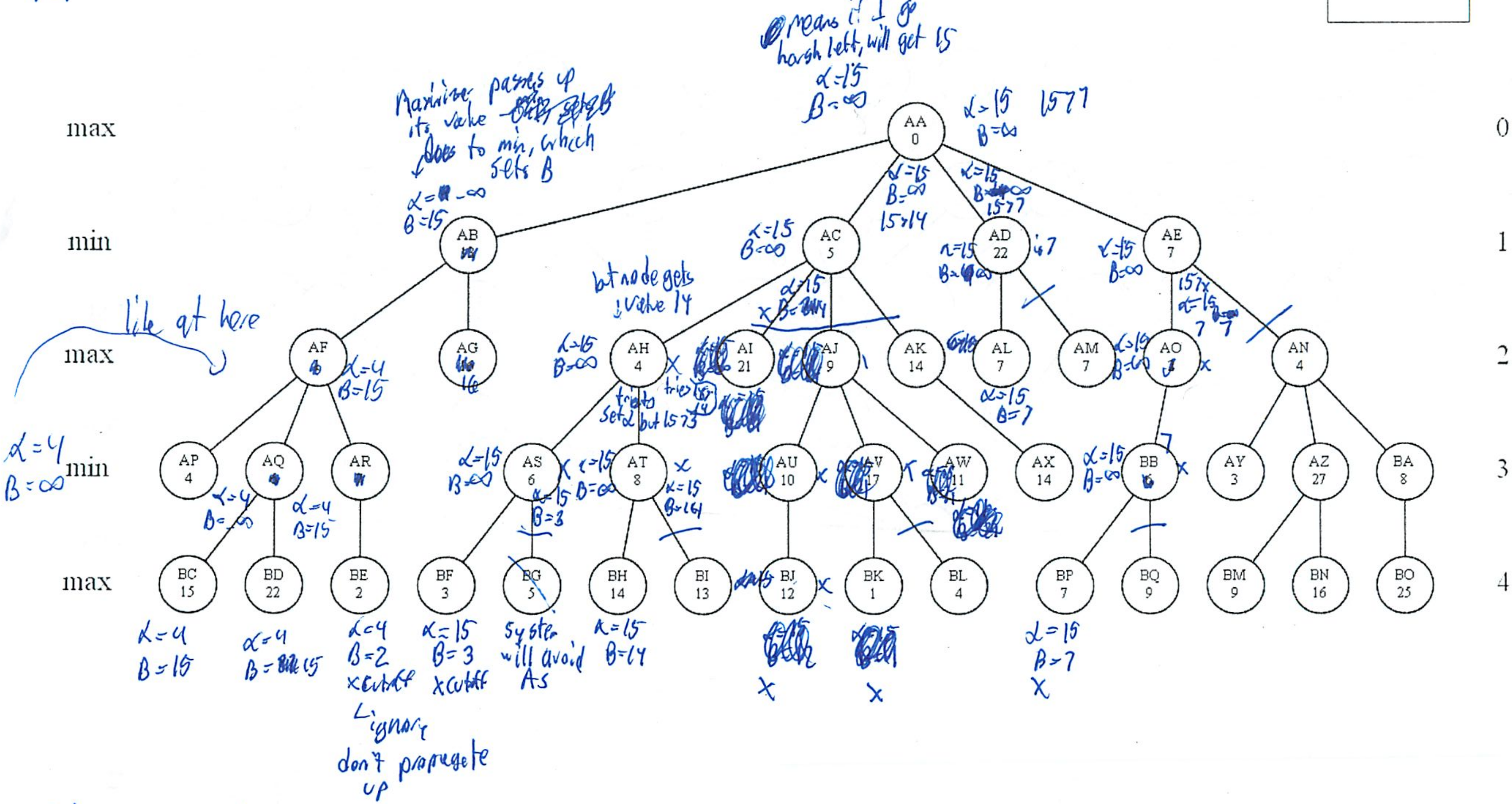
13

Tear off sheet

α,β pratice OH

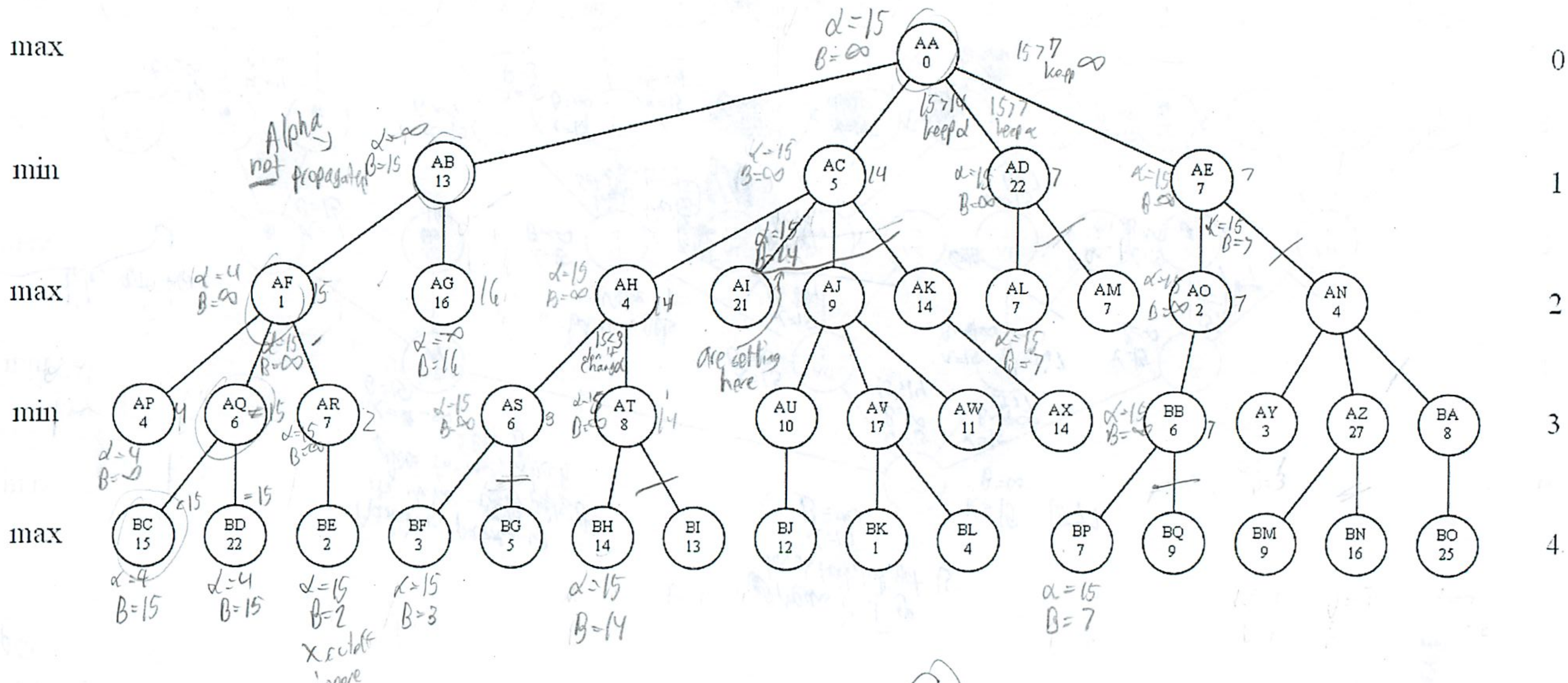
*if top node(root) is max β is always ∞

no flipping on paper



α,β propage ↓ , Values propagate up

HW4 try 2



0 correct

$\alpha \geq \beta$

6

(10)

Constraint Prop

Write something down if it is in the domain

~~the~~ #2 part first

Hornbill = 2

then FC w/ prop through reduced any domains

See Ant = 3 x doesn't work

Ant = 4 x " "

Don't write b/c its prop checking

Think back to P-set

- if tree ~~search~~ tries it → write
  code
  (prewritten)

- if ~~CP~~ propagation code (we wrote) → don't write

So Hornbill 2 ← write line

Lall CP

(11)

For D

Don't write 3,4 for Boar

└ Since Boar must be w/ meercat

– ~~[illegible]~~ consider before writing

– 3,4 are ~~[illegible]~~ eliminated w/ f. c. └ after assigning

So don't write for Meer = 1

Read 11/13

6.034f Neural Net Notes
October 28, 2010

These notes are a supplement to material presented in lecture. I lay out the mathematics more prettily and extend the analysis to handle multiple-neurons per layer. Also, I develop the back propagation rule, which is often needed on quizzes.

I use a notation that I think improves on previous explanations. The reason is that the notation here plainly associates each input, output, and weight with a readily identified neuron, a left-side one and a right-side one. When you arrive at the update formulas, you will have less trouble relating the variables in the formulas to the variables in a diagram.

One the other hand, seeing yet another notation may confuse you, so if you already feel comfortable with a set of update formulas, you will not gain by reading these notes.

The sigmoid function

The sigmoid function, $y = 1/(1 + e^{-x})$, is used instead of a step function in artificial neural nets because the sigmoid is continuous, whereas a step function is not, and you need continuity whenever you want to use gradient ascent. Also, the sigmoid function has several desirable qualities. For example, the sigmoid function's value, y, approaches 1 as x becomes highly positive; 0 as x becomes highly negative; and equals $1/2$ when $x = 0$.

Better yet, the sigmoid function features a remarkably simple derivative of the output, y, with respect to the input, x:

$$\begin{aligned}
\frac{dy}{dx} &= \frac{d}{dx}\left(\frac{1}{1+e^{-x}}\right)\\
&= \frac{d}{dx}(1+e^{-x})^{-1}\\
&= -1 \times (1+e^{-x})^{-2} \times e^{-x} \times -1\\
&= \frac{1}{1+e^{-x}} \times \frac{e^{-x}}{1+e^{-x}}\\
&= \frac{1}{1+e^{-x}} \times \frac{1+e^{-x}-1}{1+e^{-x}}\\
&= \frac{1}{1+e^{-x}} \times \left(\frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}}\right)\\
&= y(1-y)
\end{aligned}$$

Shows deriv is simple

Thus, remarkably, the derivative of the output with respect to the input is expressed as a simple function of the output.

The performance function

The standard performance function for gauging how well a neural net is doing is given by the following:

$$P = -\frac{1}{2}(d_{\text{sample}} - o_{\text{sample}})^2$$

desired output

where P is the performance function, d_{sample} is the desired output for some specific sample and o_{sample} is the observed output for that sample. From this point forward, assume that d and o are the desired and observed outputs for a specific sample so that we need not drag a subscript around as we work through the algebra.

The reason for choosing the given formula for P is that the formula has convenient properties. The formula yields a maximum at $o = d$ and monotonically decreases as o deviates from d. Moreover, the derivative of P with respect to o is simple:

$$\begin{aligned}\frac{dP}{do} &= \frac{d}{do}[-\frac{1}{2}(d-o)^2] \\ &= -\frac{2}{2} \times (d-o)^1 \times -1 \\ &= d - o\end{aligned}$$

deriv is easy

Gradient ascent

Backpropagation is a specialization of the idea of gradient ascent. You are trying to find the maximum of a performance function P, by changing the weights associated with neurons, so you move in the direction of the gradient in a space that gives P as a function of the weights, w. That is, you move in the direction of most rapid ascent if we take a step in the direction with components governed by the following formula, which shows how much to change a weight, w, in terms of a partial derivative:

$$\Delta w \propto \frac{\partial P}{\partial w}$$

proportion to

Δw is directly proportion to $\frac{\partial P}{\partial w}$

The actual change is influenced by a rate constant, α; accordingly, the new weight, w', is given by the following:

↑ given

$$w' = w + \alpha \times \frac{\partial P}{\partial w}$$

$\frac{\partial P}{\partial w} = \delta \cdot i$

$w' = w + \alpha \cdot \delta \cdot i$

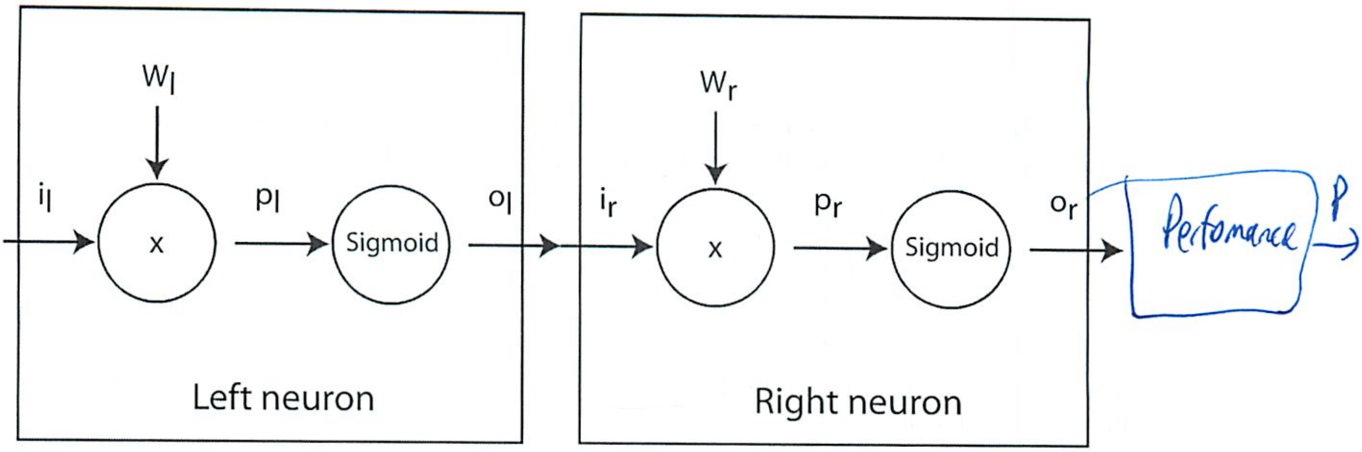
Gradient descent

alt way

If the performance function were $\frac{1}{2}(d_{\text{sample}} - o_{\text{sample}})^2$ instead of $-\frac{1}{2}(d_{\text{sample}} - o_{\text{sample}})^2$, then you would be searching for the minimum rather than the maximum of P, and the change in w would be subtracted from w instead of added, so w' would be $w - \alpha \times \frac{\partial P}{\partial w}$ instead of $w + \alpha \times \frac{\partial P}{\partial w}$. The two sign changes, one in the performance function and the other in the update formula cancel, so in the end, you get the same result whether you use gradient ascent, as I prefer, or gradient descent.

The simplest neural net

Consider the simplest possible neural net: one input, one output, and two neurons, the left neuron and the right neuron. A net with two neurons is the smallest that illustrates how the derivatives can be computed layer by layer.



Note that the subscripts indicate layer. Thus, i_l, w_l, p_l, and o_l are the input, weight, product, and output associated with the neuron on the left while i_r, w_r, p_r, and o_r are the input, weight, product, and output associated with the neuron on the right. Of course, $o_l = i_r$.

Suppose that the output of the right neuron, o_r, is the value that determines performance P. To compute the partial derivative of P with respect to the weight in the right neuron, w_r, you need the chain rule, which allows you to compute partial derivatives of one variable with respect to another in terms of an intermediate variable. In particular, for w_r, you have the following, taking o_r to be the intermediate variable:

$$\frac{\partial P}{\partial w_r} = \overbrace{\frac{\partial P}{\partial o_r}}^{\text{performance}} \times \overbrace{\frac{\partial o_r}{\partial w_r}}^{\text{right neuron}}$$

Now, you can repeat, using the chain-rule to turn $\frac{\partial o_r}{\partial w_r}$ into $\frac{\partial o_r}{\partial p_r} \times \frac{\partial p_r}{\partial w_r}$:

$$\frac{\partial P}{\partial w_r} = \overbrace{\frac{\partial P}{\partial o_r}}^{\text{perf}} \times \overbrace{\frac{\partial o_r}{\partial p_r}}^{\text{sigmoid}} \times \overbrace{\frac{\partial p_r}{\partial w_r}}^{\text{multiply}}$$

Conveniently, you have seen two of the derivatives already, and the third, $\frac{\partial p_r}{\partial w_r} = \frac{\partial (w_r \times o_l)}{\partial w_r}$, is easy to compute:

$$\frac{\partial P}{\partial w_r} = \overbrace{[(d - o_r)]}^{\text{perf}} \times \overbrace{[o_r(1 - o_r)]}^{\text{sigmoid}} \times \overbrace{[i_r]}^{\text{multiply}}$$

Repeating the analysis for w_l yields the following. Each line is the same as the previously, except that one more partial derivative is expanded using the chain rule:

$$\begin{aligned}
\frac{\partial P}{\partial w_l} &= \overbrace{\frac{\partial P}{\partial o_r}}^{\text{perf}} \times \overbrace{\frac{\partial o_r}{\partial w_l}}^{\text{both neurons}} \\
&= \frac{\partial P}{\partial o_r} \times \frac{\partial o_r}{\partial p_r} \times \frac{\partial p_r}{\partial w_l} \\
&= \frac{\partial P}{\partial o_r} \times \overbrace{\frac{\partial o_r}{\partial p_r} \times \frac{\partial p_r}{\partial o_l}}^{\text{right}} \times \frac{\partial o_l}{\partial w_l} \quad \text{left} \\
&= \frac{\partial P}{\partial o_r} \times \frac{\partial o_r}{\partial p_r} \times \frac{\partial p_r}{\partial o_l} \times \overbrace{\frac{\partial o_l}{\partial p_l} \times \frac{\partial p_l}{\partial w_l}} \\
&= [(d - o_r)] \times [o_r(1 - o_r)] \times [w_r] \times [o_l(1 - o_l)] \times [i_l]
\end{aligned}$$

? $w_l \cdot i_l$

its given in terms of w_l

Thus, the derivative consists of products of terms that have already been computed and terms in the vicinity of w_l. This is clearer if you write the two derivatives next to one another:

$$\frac{\partial P}{\partial w_r} = (d - o_r) \times o_r(1 - o_r) \times i_r$$
$$\frac{\partial P}{\partial w_l} = (d - o_r) \times o_r(1 - o_r) \times w_r \times o_l(1 - o_l) \times i_l$$

You can simplify the equations by defining δs as follows, where each delta is associated with either the left or right neuron:

$$\left.\begin{aligned} \delta_r &= o_r(1 - o_r) \times (d - o_r) \\ \delta_l &= o_l(1 - o_l) \times w_r \times \delta_r \end{aligned}\right\} \text{rename}$$

Then, you can write the partial derivatives with the δs:

$$\frac{\partial P}{\partial w_r} = i_r \times \delta_r$$
$$\frac{\partial P}{\partial w_l} = i_l \times \delta_l$$

If you add more layers to the front of the network, each weight has a partial derivatives that is computed like the partial derivative of the weight of the left neuron. That is, each has a partial derivative determined by its input and its delta, where its delta in turn is determined by its output, the weight to its right, and the delta to its right. Thus, for the weights in the final layer, you compute the change as follows, where I use f as the subscript instead of r to emphasize that the computation is for the neuron in the final layer:

$$\Delta w_f = \alpha \times i_f \times \delta_f$$

where

$$\delta_f = o_f(1 - o_f) \times (d - o_f)$$

final layer

For all other layers, you compute the change as follows:

$$\Delta w_l = \alpha \times i_l \times \delta_l$$

where

$$\delta_l = o_l(1 - o_l) \times w_r \times \delta_r$$

other layers

More neurons per layers

This was my question earlier

Of course, you really want back propagation formulas for not only any number of layers but also for any number of neurons per layer, each of which can have multiple inputs, each with its own weight. Accordingly, you need to generalize in another direction, allowing multiple neurons in each layer and multiple weights attached to each neuron.

The generalization is an adventure in summations, with lots of subscripts to keep straight, but in the end, the result matches intuition. For the final layer, there may be many neurons, so the formula's need an index, k, indicating which final node neuron is in play. For any weight contained

oh have not tried this yet

Show example

in the final-layer neuron, f_k, you compute the change as follows from the input corresponding to the weight and from the δ associated with the neuron:

$$\begin{aligned}\Delta w =& \alpha \times i \times \delta_{f_k}\\ \delta_{f_k} =& o_{f_k}(1 - o_{f_k}) \times (d_k - o_{f_k})\end{aligned}$$

Note that the output of each final-layer neuron output is subtracted from the output desired for that neuron.

For other layers, there may also be many neurons, and the output of each may influence all the neurons in the next layer to the right. The change in weight has to account for what happens to all of those neurons to the right, so a summation appears, but otherwise you compute the change, as before, from the input corresponding to the weight and from the δ associated with the neuron:

$$\begin{aligned}\Delta w =& \alpha \times i \times \delta_{l_i}\\ \delta_{l_i} =& o_{l_i}(1 - o_{l_i}) \times \sum_j w_{l_i \to r_j} \times \delta_{r_j}\end{aligned}$$

Note that $w_{l_i \to r_j}$ is the weight that connects the j^{th} right-side neuron to the output of the i^{th} left-side neuron.

Summary

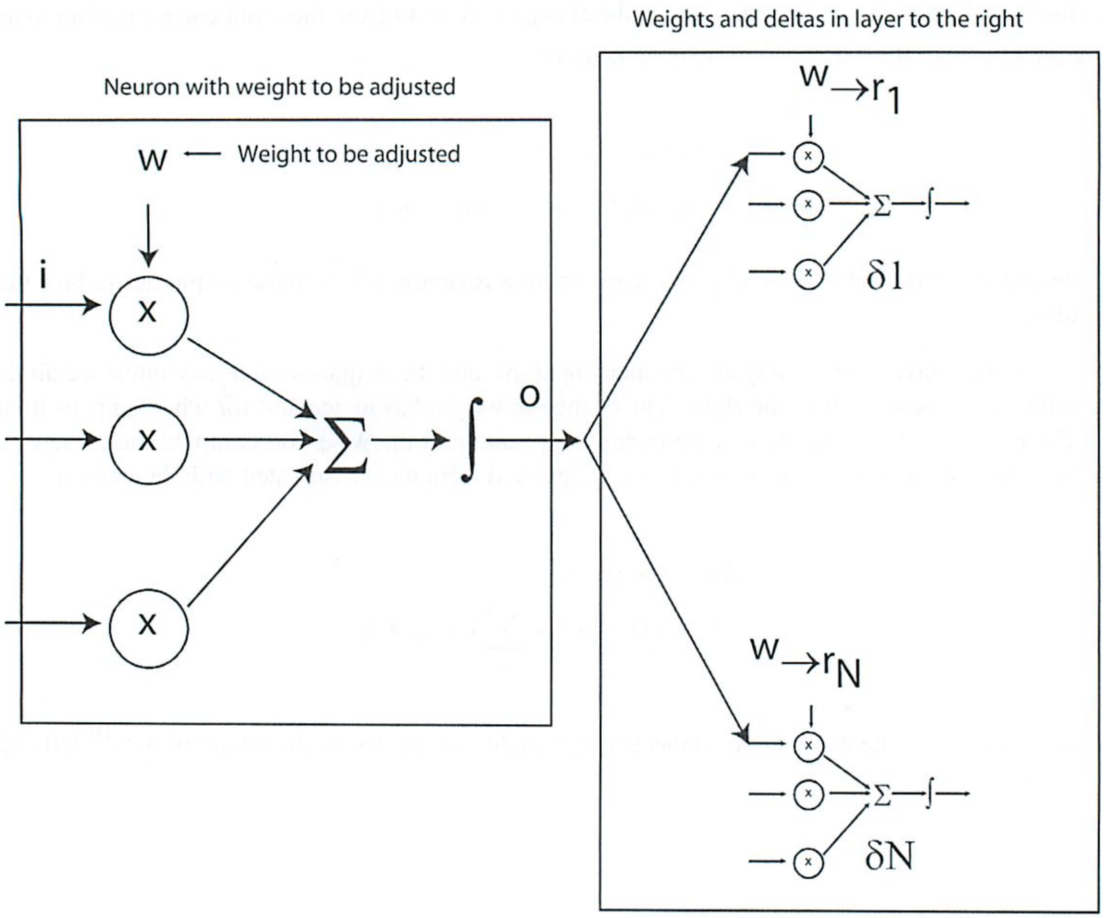
Once you understood how to derive the formulas, you can combine and simplify them in preparation for solving problems. For each weight, you compute the weight's change from the input corresponding to the weight and from the δ associated with the neuron. Assuming that δ is the delta associated with that neuron, you have the following, where $w_{\to r_j}$ is the weight connecting the output of the neuron you are working on, the i^{th} left-side neuron, to the j^{th} right-side neuron, and δ_{r_j} is the δ associated with that right-side neuron.

previous

$$\left[\begin{aligned}\delta_o =& o(1 - o) \times (d - o) && \text{for the final layer}\\ \delta_{l_i} =& o_{l_i}(1 - o_{l_i}) \times \sum_j w_{l_i \to r_j} \times \delta_{r_j} && \text{otherwise}\end{aligned}\right.$$

& multiple

That is, you computed change in a neuron's w, in every layer, by multiplying α times the neuron's input times its δ. The δ is determined for all but the final layer in terms of the neuron's output and all the weights that connect that output to neurons in the layer to the right and the δs associated with those right-side neurons. The δ for each neuron in the final layer is determined only by the output of that neuron and by the difference between the desired output and the actual output of that neuron.


Weights and deltas in layer to the right
Neuron with weight to be adjusted
w
Weight to be adjusted
i
x
x
x
Σ
∫
o
$w_{\rightarrow r_1}$
δ1
$w_{\rightarrow r_N}$
δN

6.034 Quiz 3 Studying

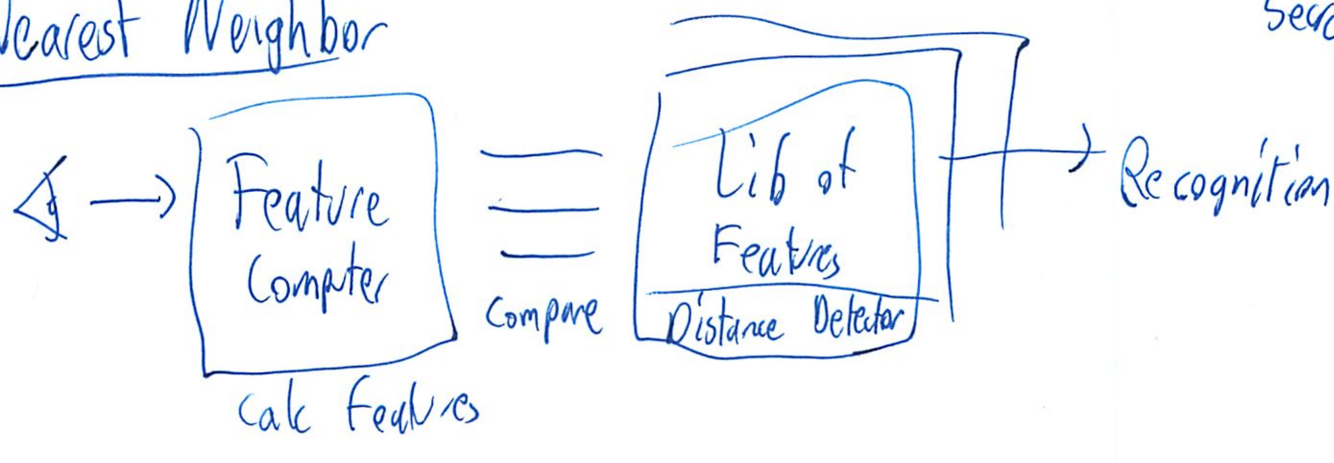
1/13

- k Nearest Neighbors
- ID Trees
- Neural Nets
- __NOT__ SVM
- __NOT__ Boosting

Lecture 10/14 Learning

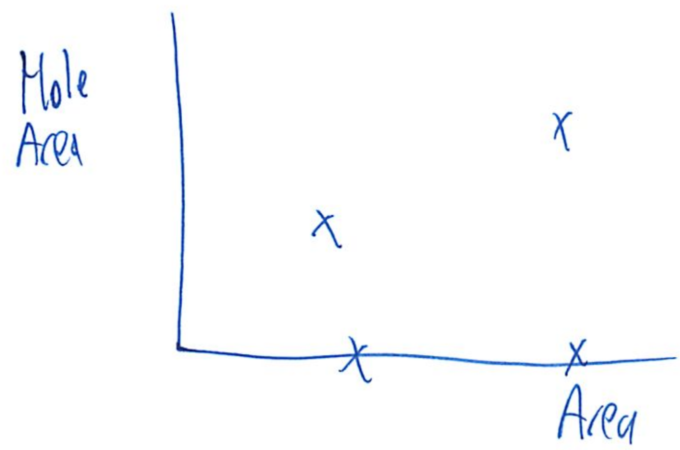
- Reasoning
- Learning
  - Models
    - One Shot
    - Language
  - Bulldozer
    - Regularity
      - Classification / Nearest neighbors
      - Biology / Neural Nets
      - Computer Theory
      - Stats / Baysean Search

Nearest Neighbor

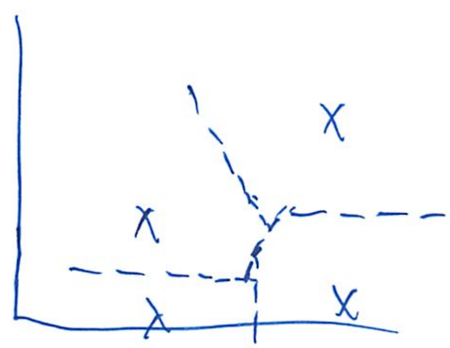


(2)

Compare 2 features



Draw lines b/w each pair



Want min angle b/w probe vector + nearest neighbor

Can use to train an arm to throw a ball

Learns from testing

<u>Sleep</u>

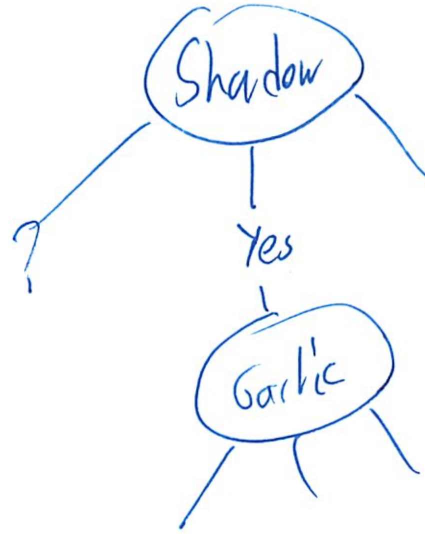
After 36 hrs - capability ↓

Similar to alcohol consumption

3

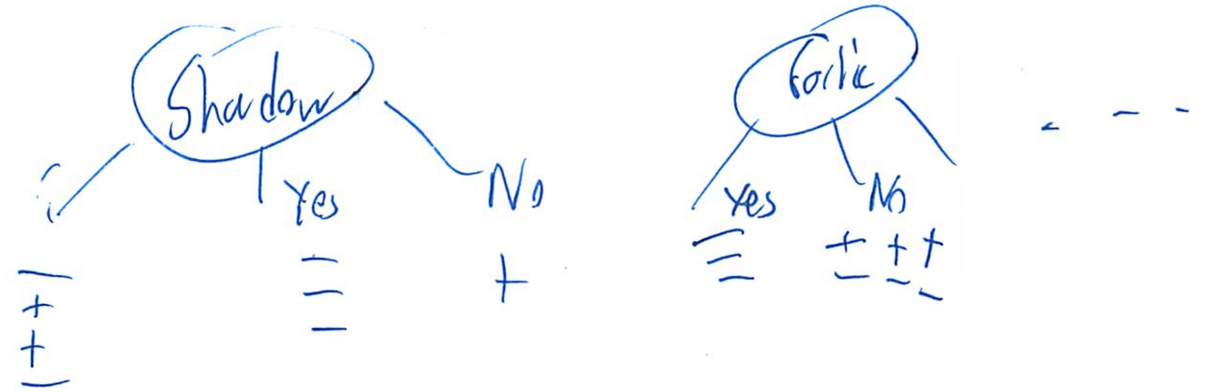
Lecture: Classification Trees

- Symptoms of Vampirism
- Can make tree of tests



etc

So build tree by trying each as root el



Eval each one for "goodness"

$$D(set) = -\frac{P}{T}\log_2\frac{P}{T} - \frac{N}{T}\log_2\frac{N}{T}$$

for 2 types P, N
$T = P + N$

(3) (4)

Then for remaining items test from remaining items
for each split branch
ii Pick the closest to .5
Verify

Then write simple rules from top to bottom

If shadow test = ? AND Garlic = Y
Then Not vampire

Can condence to square

Vampire	+	−
shadow = ?	+	=
shadow ≠ ?		−

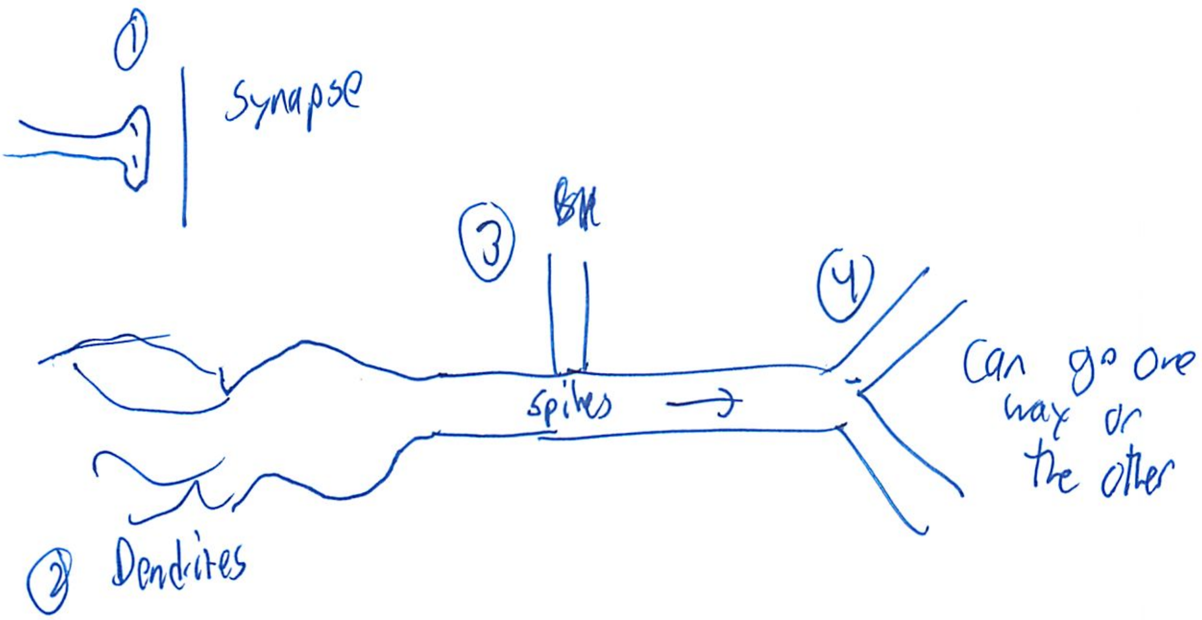
Assume garlic = Y

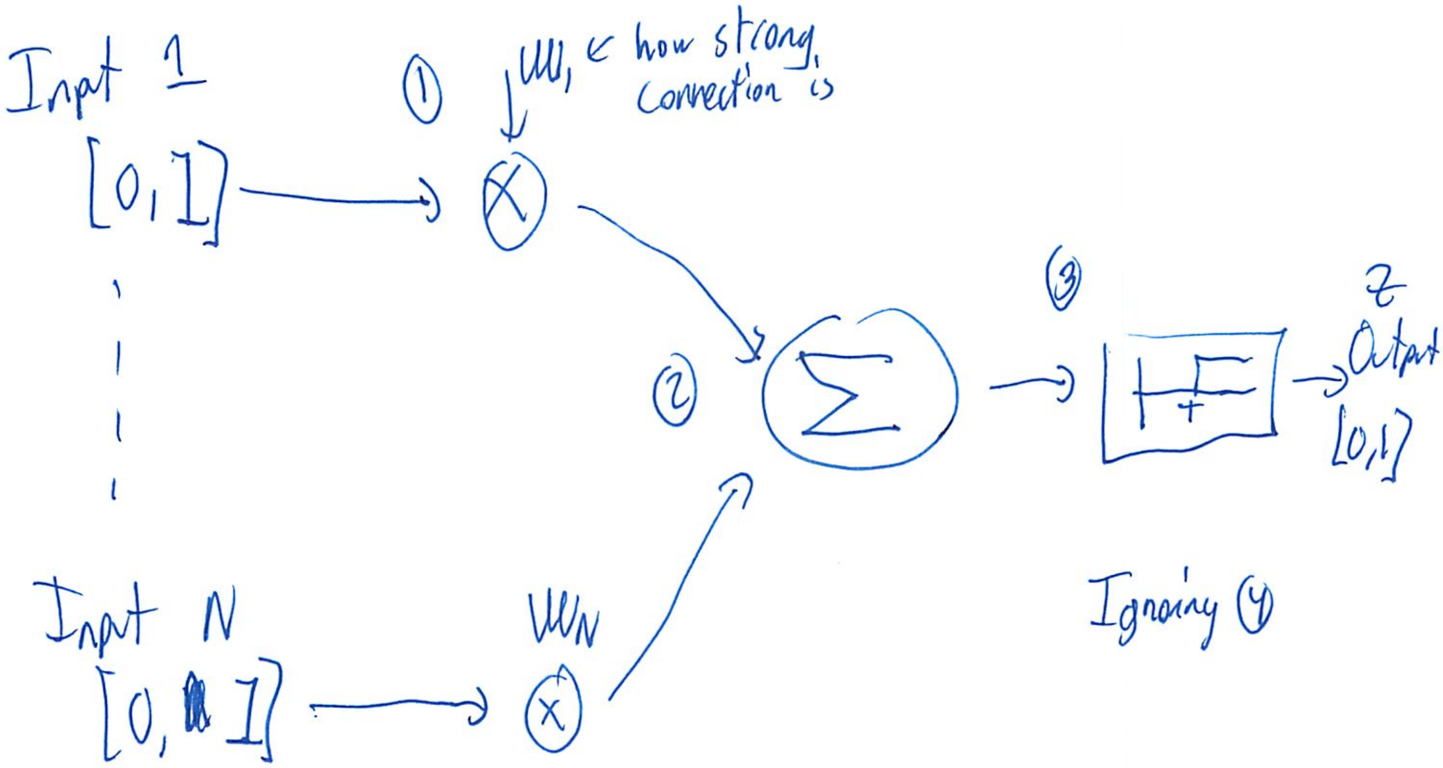
So Can combine rules

If garlic = Y || Shadow = Y
Then Not vampire
Else Is Vampire

5

Lecture: Native Neurobiology





Ignoring 4

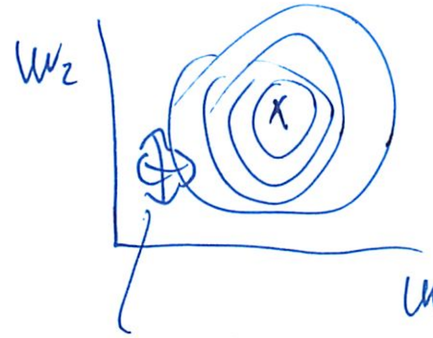
(6)

$\bar{z} = f(\bar{x}, \bar{w}, \bar{T})$

output, input, weight, threshol

Performence Metric $-\frac{1}{2}\|\bar{d}-\bar{z}\|^2$

Output



use hill climbing to find right weights

Find the gradient (18.02)

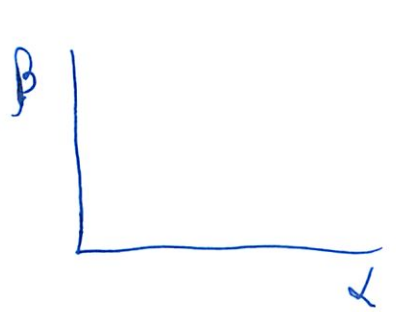
$$\Delta \bar{w} = \left(\frac{\partial P}{\partial w_1}\hat{i} + \frac{\partial P}{\partial w_2}\hat{j}\right) r$$

r: rate change constant

Can get rid of threshold to make problem simpler

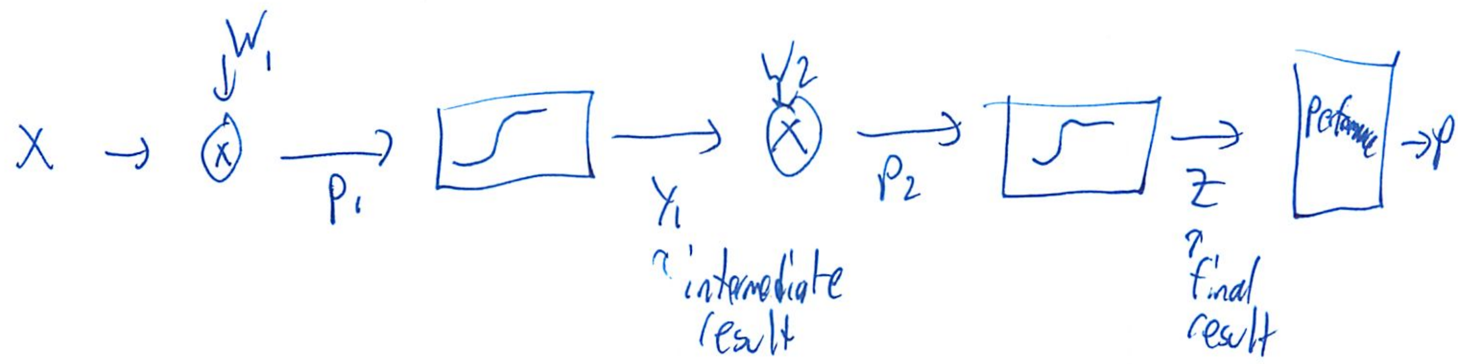
~~Sometimes~~

Smooth it out so differentiable



$$\beta = \frac{1}{1+e^{-\alpha}}$$

7



$$\frac{\partial P}{\partial W_2} = \frac{\partial P}{\partial Z}\frac{\partial Z}{\partial W_2}$$

Remember open book!

$$= \frac{\partial -\frac{1}{2}(d-z)^2}{\partial z}$$

$$= (d-z)\frac{\partial z}{\partial W_2}$$

So now need another chain rule

$$= (d-z)\frac{\partial z}{\partial P_2}\frac{\partial P_2}{\partial W_2}$$

$$= (d-z)\frac{\partial z}{\partial P_2}\; Y_1$$

how we get this?

$P_2 = W_2 \cdot Y_1$

differenate P_2 w/ respect to W_2

so Y just constant in front ✓

(8)

$$\beta = \frac{1}{1+e^{-\alpha}}$$

$$= (1+e^{-\alpha})^{-1}$$ rewrite to make it eaiser to differentiate

$$\frac{d\beta}{d\alpha} = -1(1+e^{-\alpha})^{-2} \times e^{-\alpha} \cdot -1$$

inside

-1 is constant in front of $\wedge$ take outside

$$= \frac{e^{-\alpha}}{(1+e^{-\alpha})^2}$$

$$= \frac{1}{(1+e^{-\alpha})} \cdot \frac{e^{-\alpha}}{(1+e^{-\alpha})}$$

$$= \frac{1}{(1+e^{-\alpha})} \cdot \frac{1+e^{-\alpha}-1}{(1+e^{-\alpha})}$$

$$= \frac{1}{1+e^{-\alpha}} \left[\frac{(1+e^{-\alpha})}{(1+e^{-\alpha})} - \frac{1}{1+e^{-\alpha}} \right]$$

$$= \beta(1-\beta)$$

That was silly ~~

9

$$= (d-z)\, z\,(1-z)\, y$$

$$\frac{\partial P}{\partial w_1} = \frac{\partial P}{\partial z}\frac{\partial z}{\partial w_1}$$

$$= \frac{\partial P}{\partial z}\frac{\partial z}{\partial p_2}\frac{\partial p_2}{\partial w_1}$$

$$= (d-z)\, z(1-z)\, \frac{\partial p_2}{\partial w_1}$$

$$= (d-z)\, z(1-z)\, w_2 \frac{\partial y}{\partial w_1}$$

$$= (d-z)\, z(1-z)\, w_2 \frac{\partial y}{\partial p_1}\frac{\partial p_1}{\partial w_1}$$ splitting this out

$$= \underbrace{(d-z)}_{\text{Performance}}\ \underbrace{z(1-z)\, w_2}_{\text{~~first~~ 2nd part}}\ \underbrace{y(1-y)\, x}_{\text{~~2nd~~ 1st part}}$$

where did w_1 go

I think

$$\underbrace{y(1-y)\, w_1\, x}$$

together all this is input to 2nd part

(10)

Overfits a lot

Recitation Newest Neighbor

Bisecting lines

Erase when goes over

Never curved!

Classification tree

Want lowest avg entropy

Take single best step

Weighted avg of entropy after cut

$$\underbrace{W_1 H_1}_{} + \underbrace{W_2 H_2}_{\text{other half}}$$

W_1: $\frac{3}{4}$ points in this half

H_1: entropy of this half

+ −

$-p \log_2 p$

Or fully

$$\underbrace{-\frac{1}{3} \lg_2 \frac{1}{3}}_{+\text{ pts}} \underbrace{-\frac{2}{3} \lg_2 \frac{2}{3}}_{-\text{ pts}}$$

Do as few cuts as possible

11

k-nearest neighbor

- k usually odd
- look at k closest
- don't draw lines

Usually Euclidian Distance

- but can be Manhatten (blocks)
- Hamming

No T junctions (see packet) - can have just a box
- does not have to be connected

Avg Disorder

$$\sum_b \left(\frac{n_b}{n_t}\right) \times \left(\sum_c -\frac{n_{bc}}{n_b} \lg_2 \left(\frac{n_{bc}}{n_b}\right)\right)$$

n_b is samples in branch b

n_{bc} is " " " " of class c

n_t is all samples

Table to make calculation easy

(12)

0 - 1 entropy perfect
.5 is worst

They tell you how to break ties

Can convert to polar

Need to try this stuff

Lecture: Genetic Algorithms

- Neural nets are functional approximations
- Sexual Reproduction

Mitosis - divides chromosomes into 2 identical sets
- get 2 identical **diploid** cells

Meosis - divides to allow sexual reproduction (sperm egg cells)
- get 4 unique **haploid** cells

Dipoid - one chromosome from M, one from F (2n)

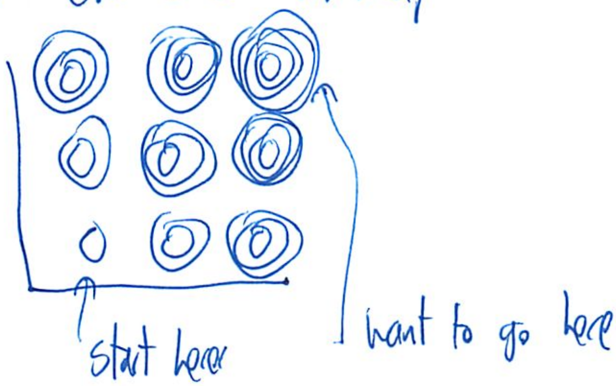
Haploid - one chromosome (n)
- Combine again

Lots of randomness in where crosses over

Want to find max

- through random crossovers

Have our crossover map



Hard time getting off local maxima

Approach 2

From ranking - not exact values

Rank	P
1	P_c
2	$(1-P_c)P_c$
⋮	
N-1	$(1-P_c)^{N-2}P_c$
N	$(1-P_c)^{N-1}$

(14)

$$S(x-1) = \frac{x^{n-1}-1}{x-1} = \frac{(1-P_c)^{n-1}-1}{P_c}$$

$$P_c \cdot S(x-1) = 1$$

Is this just proof?

Approach 3

Add some diversity

best: very fit AND very diverse

Can do isofitness curves

Approach 4

Add some crossover + test

His company does for scheduling

Funny blocks video

Neural Nets Recitation

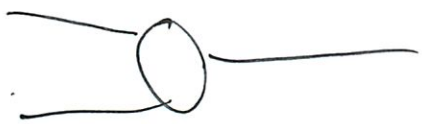
d = desired

Calc w_i' for each w_i

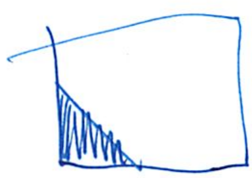
$$W_i' = W_i + \Delta W_i$$

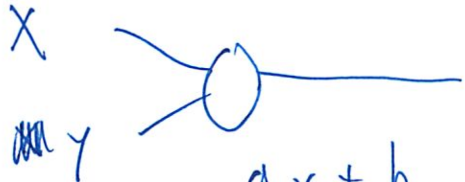
See printed handout

Now just need to practice w/ multiple inputs



Making these pictures





$$ax + by = t$$

t = cutoff pt

$$y = \frac{t - ax}{b}$$

Can have 1 node be AND, Not → XOR in 2

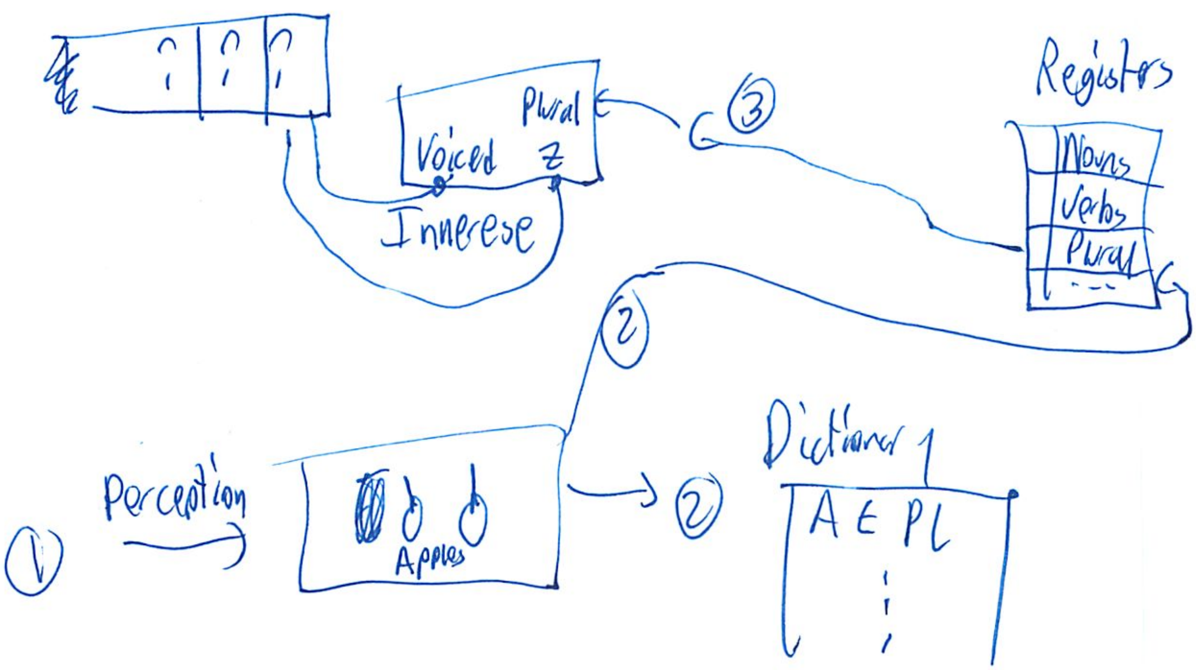
16

Lecture: The Right Way Chronology

Phonemes - distinctive features of words
- smallest segment of sound that forms meaningful contrasts b/w utterances
- depends on language

Vector of distinctive features
. 14 for each phoneme

Build System



17

3 Fill buffer

AE	P	L

4 Do I have enough info to make decision?
If not shift

5 | AE | P | L | Z | ✓ Good

Works Bidirectional

Table of distinctive features

KAT

Syllabic	−	+	−
Voiced	−	+	−
Continuant	−	+	−

DOG

Syllabic	−	+	−
Voiced	+	+	+
Continuant	−	+	+

Col = distinctive feature
Row = phoneme
Each word is a sequence of phonemes

(18)

Do Beam(2) Search

uncovered samples

Collect ⊕ ⊖ samples

Pick ⊕ sample as seed

Reject

No

Generalize left → right → No more generalization possible

⊕ → 0 ⊖ → 0

Matches ⊖ Sample

Rule
- may match

Marr's Catechism

1. Understand problem
2. Get a good representation
3. Develop a method / algorithm

Good representations
1. Make something explicit
2. Express constraint
3. Localness
4. Computable

Want

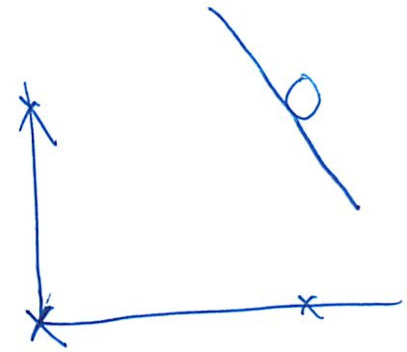
Competence
- done right

Performance
- done fast

Perceptron Applet

1/13

AND



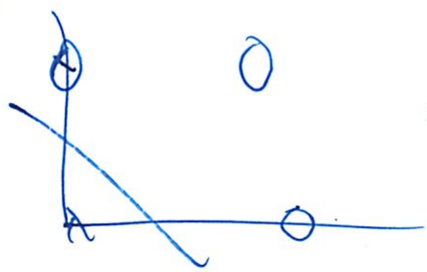
$w_1 = .25$
$w_2 = .25$

✓ ahh think of like this

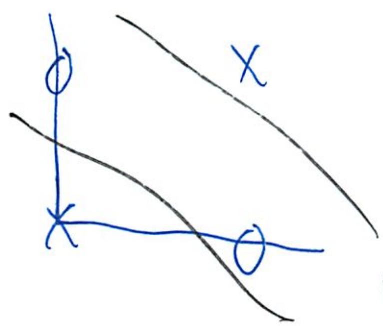
Then threshold to get

$$\begin{cases} 0 & x \le 0 \\ 1 & x \ge 1 \end{cases}$$

OR



XOR



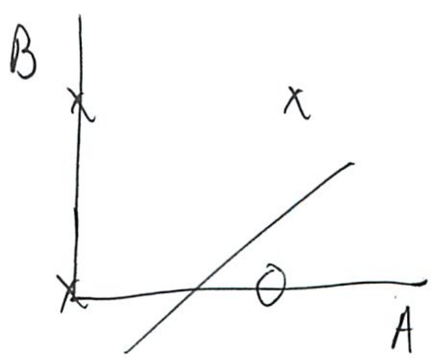
Can't do

✓ need 2 lines

So $A < B$

So TT · strange since not binary

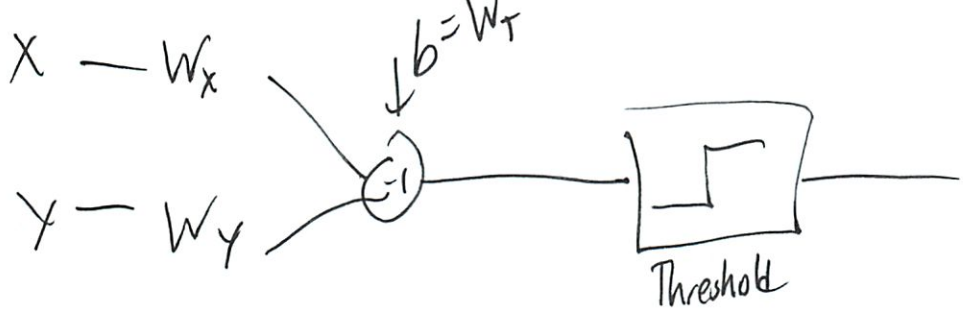
A	B	A<B
0	0	0
0	1	1
1	0	0
1	1	0



(2)

But what do weights do here

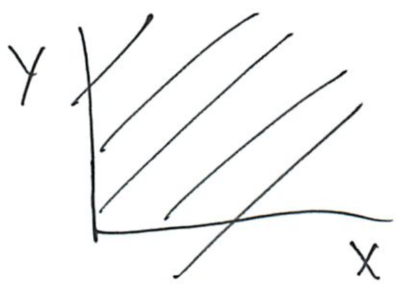
So



So $XW_x + YW_y + W_T = $ line

So for a

$-2 \quad 2 \quad -1$



What I saw earlier

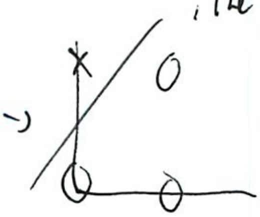
try values

but why -2 ??

TT

A	B	¬A	¬A ∨ B
0	0	1	1
0	1	1	1
1	0	0	0
1	1	0	1

+1 →



; the line I guess

Updating Neural Nets

11/13

(how could I miss in notes!?!)

$$\Delta W_i = \alpha \cdot \delta_f \cdot \text{input } i$$

for each W_i — learning constant (given) — $O_f(1-O_f)(d-O)$ final

$$W_i' = W_i + \Delta W_i$$

for each

11/13
Practice

6.034 Quiz 3
November 9, 2009

Name	
EMail	

Circle your TA and recitation time, if any, so that we can more easily enter your score in our records and return your quiz to you promptly.

TAs
Erica Cooper
Matthew Peairs
Charles Watts
Mark Seifter
Yuan Shen
Jeremy Smith
Olga Wichrowska

Thu	
Time	Instructor
11-12	Gregory Marton
12-1	Gregory Marton
1-2	Bob Berwick
2-3	Bob Berwick
3-4	Bob Berwick

Fri	
Time	Instructor
1-2	Randall Davis
2-3	Randall Davis
3-4	Randall Davis

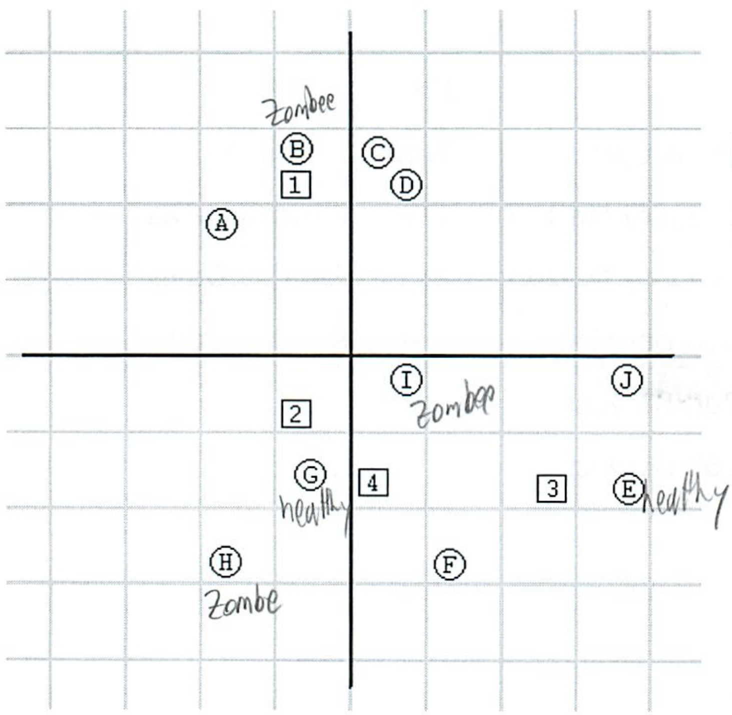
Problem number	Maximum	Score	Grader
1	50		
2	50		
Total	100		

There are ?? pages in this quiz, including this one. In addition, tear-off sheets are provided at the end with duplicate drawings and data.

As always, open book, open notes, open just about everything.

Problem 1:KNN & ID Trees (50 points)

Part A: K Nearest Neighbors, backwards (15 pts)
Shaun has been hired by the Joint Intelligence Committee to investigate the recent zombie infection in his hometown. The first thing Shaun needs to do is to make sense of the incomplete data the JIC has provided him. In the graph below, the circles correspond to observed people, but their labels, "zombie" or "healthy", were lost during the initial investigation. The square points represent people who still need to be classified (they are not themselves used to classify any other points).



A, C 1 or 2 zombie
D - no data

J, F 1 or 2 healthy

Shaun is also given the table below, showing how the square points would have been classified using 1- and 3-nearest neighbors before the labels were lost. Given the map and the table below, Shaun needs to recover the original labels.

Square point	Using 1-nearest-neighbors	Using 3-nearest-neighbors
1	zombie	zombie
2	healthy	zombie
3	healthy	healthy
4	? heathy	?

Clever!
← 1 of the 2 may be healthy
← Closest health, next 2 zombee

Can't get F · ? will be zombee if F zombee
healthy F healthy

A1: Write down whether the following specimens are zombies (Z), healthy (H), or if it's unknown (U).

Circle A: U ✓

Circle B: Z ✓

Circle C: U ✓

Circle D: U ✓

Circle E: H ✓

Circle F: U ✓

Circle G: H ✓

Circle H: Z ✓

Circle I: Z ✓

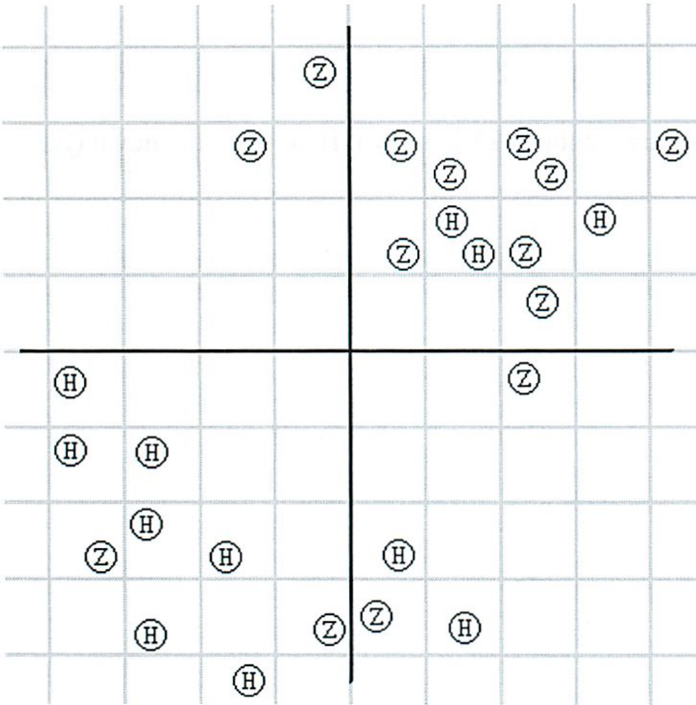
Circle J: U ✓

A2: How would point 4 be classified? (Again, choose Z, H, or U)

Using 1-nearest neighbor: H ✓

Using 3-nearest neighbors: U ✓

A3: Shaun is wondering whether this k-nearest-neighbor algorithm is really reliable. He decides to check it on some labeled zombie data from a neighboring town. In the graph below, zombies are labeled Z, and healthy people are labeled H.



Describe, in a sentence or two, what happens to the accuracy of k-nearest neighbors as k increases from 1 to 26 (the total number of samples).

Was thinking this

increases initially (to k6)

The data becomes less reliable because you are using less and less reliable data

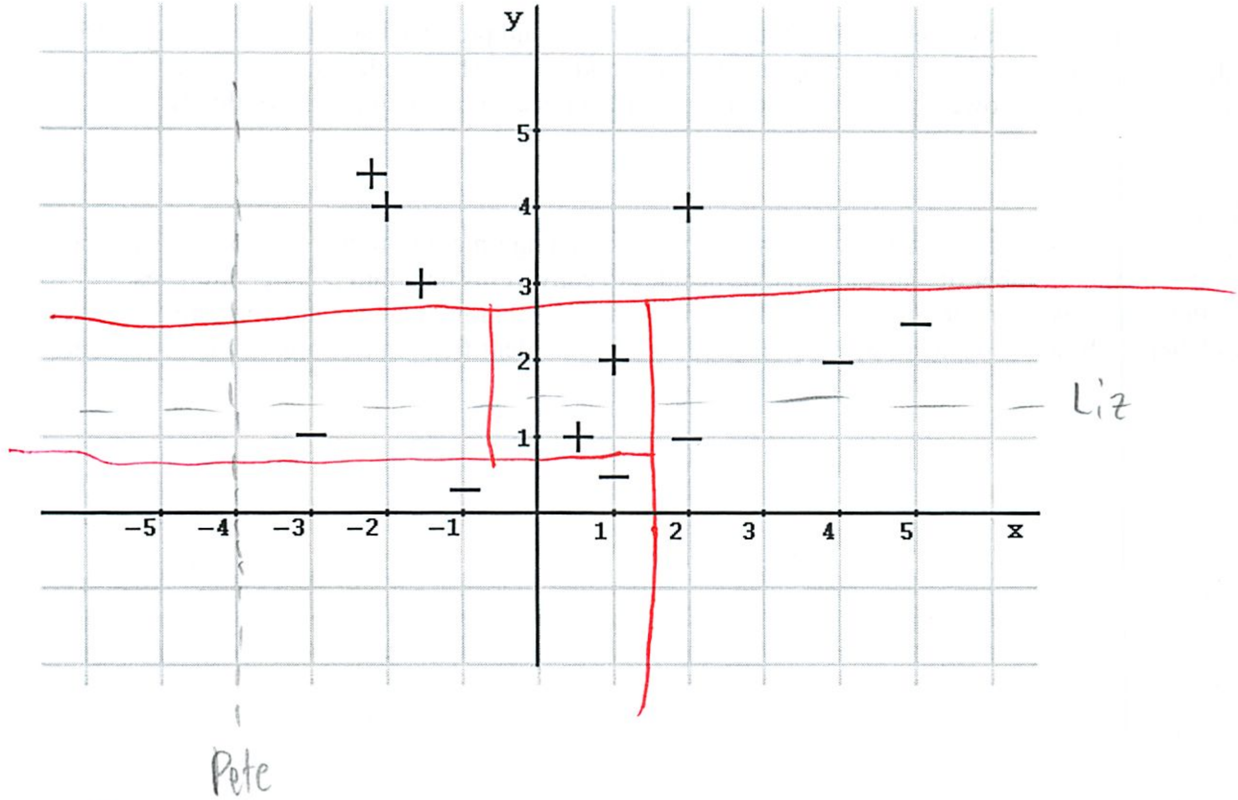
Then thought lim in math

Part B: ID Trees (35 pts)

Shaun quickly realizes that he will not be able to recover all the zombie infection information from the data he is given. Fortunately, Shaun's best friend Ed, who was in the middle of a reconnaissance mission in the town, managed to send in a bit more data before he was bitten. Shaun overlays the locations of the known zombies (marked with a +) and known healthy people (marked with a -) on a grid representing the town. (The zombies are currently not biting anyone, so you can trust the points not to change over time.) The JIC has tasked Shaun with figuring out where to build a series of walls separating the healthy people from the zombies. The walls will be built along the decision boundaries created by the identification tree algorithm.

On strike

hahhaha



B1. (13 pts)

Shaun's girlfriend Liz suggests building a wall at **y=1.5**. Compute the disorder at this decision boundary. Leave your answer only in terms of integers, fractions, arithmetic operations, and logarithms.

$$\frac{5}{12}\left(-\frac{4}{5}\lg_2\left(\frac{4}{5}\right)-\frac{1}{5}\lg_2\left(\frac{1}{5}\right)\right)+\frac{7}{12}\left(-\frac{2}{7}\lg_2\left(\frac{2}{7}\right)-\frac{5}{7}\lg_2\left(\frac{5}{7}\right)\right) \checkmark$$

bottom ⊖ ⊕ top ⊖ ⊕

Shaun's flatmate Pete loudly insists that, instead, a wall should be built at **x=-4**. Compute the disorder at this boundary.

$$\frac{0}{12}+\frac{12}{12}\left(-\frac{6}{12}\lg_2\left(\frac{6}{12}\right)-\frac{6}{12}\lg_2\left(\frac{6}{12}\right)\right)=\cancel{.5}\ 1$$

left

worst

don't do mental math

Whose idea is better, according to the heuristic described in class? (circle one)

Liz's ✓ Pete's

time to get a new flatmate

B2. (12 pts)
On the diagram above, draw the decision boundaries Shaun would produce using the identification tree algorithm. In case two decision boundaries are equally good, use the horizontal one. If there is still a tie, use the one with the lower-valued threshold. You will not need a calculator to solve this problem.

hmm how
- want largest contigeous grouping

-oh lines don't need to go all the way through

confused

B3. (10 pts)
Shaun realizes that building all these walls is going to take a long time. In order to find out whether he can build a smaller number of walls instead, he decides to convert his data into polar coordinates. **Sketch** the 12 points from the previous graph on the polar graph below (making sure they still show the + and - labels). Show the decision boundaries produced by the identification tree algorithm on the same graph.

will be much better

approx ← do better job

Describe in one sentence (or function) how the decision boundaries translate to walls on the original, x-y-plane graph.

diagonal from origin ✓

WP: Perceptron: simplest artifical neural net: a linear classifier

$f(x) = \begin{cases} 1 & \text{if } w*x+b \geq 0 \\ 0 & \text{otherwise} \end{cases}$

$w \cdot x$ = weighted sum

b = bias (constant)

basically just add?

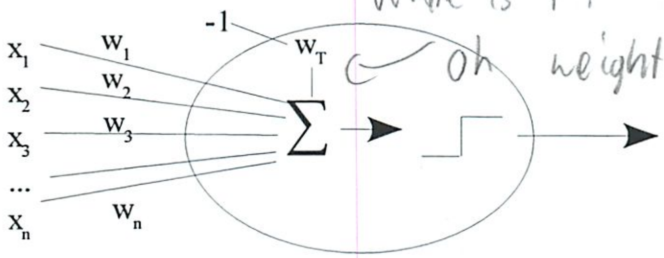
Question 2: Neural Networks and Genetic Algorithms (50 points)

Part A (20 points)

those block things

Perceptrons are the basic units of neural networks as we have seen them. They take a list of inputs **x**, multiply them by a list of corresponding weights **w**, compute the sum of those products, and pass the result through a decision function. We use a "fake" input T, usually -1, times an associated weight w_T, as part of the sum. When using a single perceptron for classification, one usually uses a threshold decision function: if the sum $z > 0$, the output is 1, otherwise 0.

Where is T?



To explore what perceptrons can and cannot do, we will ask you to make up weights, rather than training them. Consider the boolean function A→B, and note that it is the same as ¬A∨B.

1. Can a single perceptron with inputs A and B output 1 iff A→B?

Not A or B

If yes give weights: w_A= ~~-1~~ -2 w_B= ~~1~~ 2 w_T= ~~1~~ -1

If not, why not?

? can be -? ? how to do or ? +1 - AND is hard part

2. Can a perceptron capture inequalities: given two real-valued inputs A and B, can the perceptron output 1 if A<B and 0 otherwise?

I don't have a good sense of component pieces

If yes give weights: w_A= ? -1 w_B= 1 w_T= 0 ← add?

If not, why not?

? No

3. You wonder about the real-valued function A=B within epsilon E, that is, with three inputs, A, B, and and E, can a perceptron capture whether $|A-B| < E$?

If yes give weights: w_A=_______ w_B=_______ w_E=_______ w_T=_______

If not, why not?

No - a perceptron can encode 1 linear boundary
This has 2

$y = x \pm \epsilon$

? So strategy: try to draw?

$A \cdot W_A + B W_B + T \cdot W_T$

sigmoid

desired

4. The questions in part A have asked for solutions using a threshold decision function. Do the first round of training for A→B with a **sigmoid** decision function. Let the learning rate=1.

more training

A	B	T	W_A	W_B	W_T	Σ	y	y*
0	0	-1	0	0	1	-1	.27	1
0	1	-1	0	0	.86	-.86	.32	1
1	0	-1	0	.147	.713	-.713	.35	0
1	1	-1	-.079	.147	.7926	-.7252	.35	1

1-.27=.73

1-.32=.68

-.35

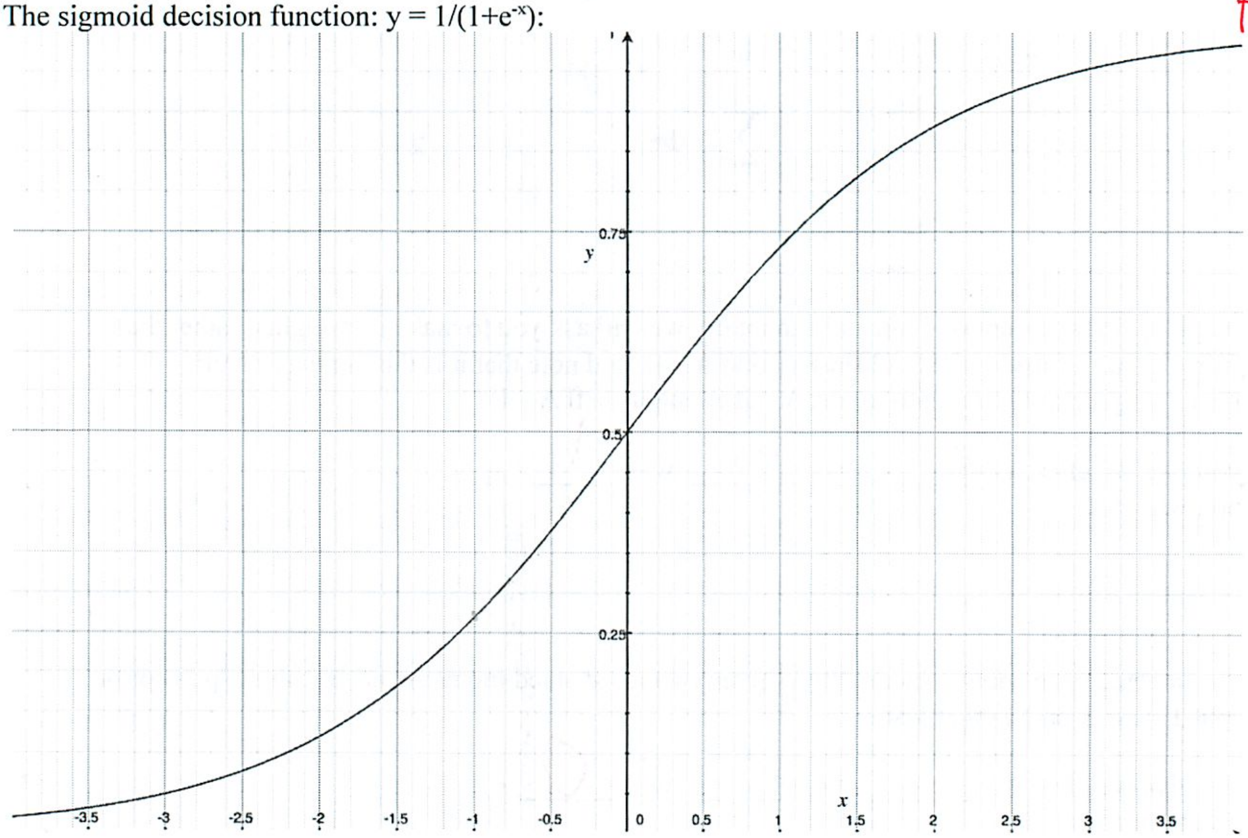
1-.35=.65

how how to do backward prop

Table diff after that

but I did the ones here right!

The sigmoid decision function: $y = 1/(1+e^{-x})$:



output

input

5. Which of the following is true about a perceptron when we use a sigmoid instead of a threshold?
   A. The perceptron can learn XOR
   B. The perceptron can no longer learn all linear classification boundaries
   C. The perceptron will learn A→B in fewer training steps
   D. The perceptron will learn A→B in more training steps
   E. None of these is true

8

Sigmoid etc

$$\Delta w_A = 1 \cdot .27(1-.27)(1-.27) \cdot 0$$

$= 0$

$w_A' = 0 + 0$

$w_B' = 0$

$$\Delta w_T = 1 \cdot .27(1-.27)(1-.27) \cdot -1$$

$= .14$

$w_T' = +1 - .14$

$= .86$

Answers seem a bit different
L The table is different too

Just try w/ this

$-.86$

sigmoid $.32$

$1-$ $.68$

(2)

$W_A = 0$

$\Delta W_B = 1 \cdot .32(1-.32)(1-.32) \cdot 1$

$= .147$

$W_B' = 0 + .147$

$\Delta W_T = 1 \cdot .32(1-.32)(1-.32) \cdot -1$

$= -.147$

$W_T' = .86 - .147$

$= .713$

$\Delta W_A = 1 \cdot .35(1-.35)(0-.35) \cdot 1$

$\Delta W_A = -.0796$

$W_A = 0 - .796$

$\Delta W_B = 1 \cdot .35(1-.35)(0-.35)\ \cancel{.147}\ 0$

$W_B = \cancel{.147}\ .147 + 0$

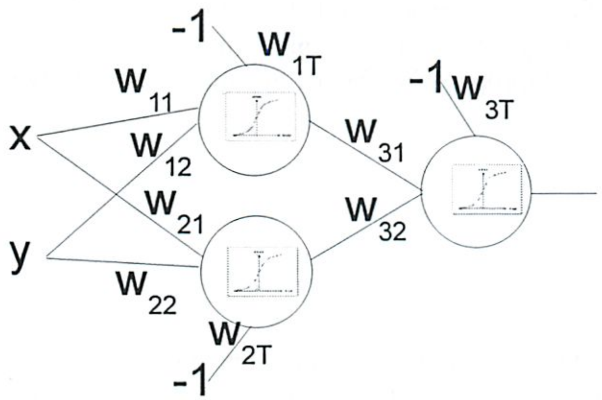
$\Delta W_T = \quad '' \quad -1$

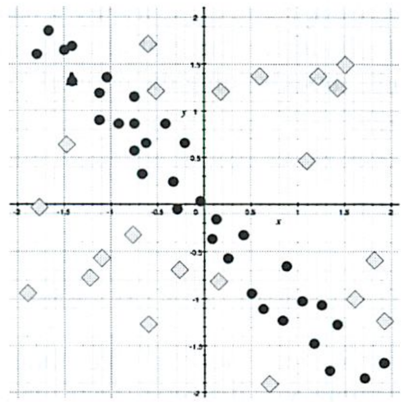
$= .0796$

$.713 + .0796 = .7926$

Part B (20 points)

Given this three-node neural network, and the training data on the right





1. Indicate which of the following sets of weights will separate the dots from the diamonds by **circling** their letters. NOTE: more than one may work!

oh duh - thats what it is

$-x-y > -\frac{1}{2}$

how are we supposed to see that?

	w11	w12	w1T	w21	w22	w2T	w31	w32	w3T
A	3	2	1	4	5	-2	-100	-100	-150
B	-2	-2	-1	3	3	-1.5	128	128	173
C	1	1	0.5	1	1	-0.5	97	97	128
D	4	4	2	6	6	-3	96	-95	-52
E	2	-2	1	2	-2	1	100	100	50
F	4	4	2	6	6	-3	-101	102	148

$3x+3y > -1.5$

So take all and try to plot

x=2 y=2 ✓
x=1 y=2 ✓
x=1 y=1 ✓
x=.5 y=.5 ✓
x=0 y=.5 ✓
x=0 y=-.5 x edge
x=-.5 y=0 x

I guess just look for edge

But then need to combine values is 1 above that - don't see how can do that fast

So $-y > -\frac{1}{2}$ is $2y < 1$

$y < \frac{1}{2}$

plot y

$x+y < \frac{1}{2}$

So

2. For (at least) one of the sets of weights you chose above, write the simplest mathematical representation for it using + - * /, inequalities, and/or boolean operations.

Which line?	mathematical expression:

3. When training the three-node neural network at the top of the page using back-propagation, if the current weights are the ones given in choice A, then, for the training example x=0 y=0, if y*=1, what is δ_1? See the tear-off sheet for notes on back-propagation. You can leave your solution expressed as a product, and it may help us assign partial credit.

Why are sols different?

Part C (10 points) ↓ seems silly, can't wrap my head arand

Your friend is building a device to unlock a door when it hears a secret knock pattern, and realizes that a neural network could do it, if only it had a sense of timing. You suggest feeding the output of one neuron into the input of one of its ancestors, and thereby get a dependence on timing.

1. You think about training the network by standard back-propagation, but decide that you can't. Why?

¿ time matters

hillclimbing

The solution is clear: Genetic Algorithms! You'll set up a population of identical neural networks with random weights, you discretize your input every 100 milliseconds into a sequence $k_1 \ldots k_n$ of 0 if silence and 1 if a knock was heard, ensuring that k_1 is always 1, and timing out eventually. You'll choose the fittest few neural networks at each step. Your friend jots down a few ideas for fitness functions:

A. Whether the full knock pattern was correctly classified
B. The length of the subsequence $k_1 \ldots k_t$ that is correctly classified
C. The length of the longest subsequence $k_i \ldots k_j$ that is correctly classified
D. The number of k_i correctly classified
E. The number of knock subsequences $k_i \ldots k_j$ that are correctly classified

¿ more

2. Select all of these fitness functions that one cannot evaluate using the neural net as a black box:

A

3. Select all of the fitness functions that will immediately trap the genetic algorithm in a fitness plateau:

B C

4. Select all of the fitness functions that do not correlate with the actual fitness: describe

No answers

5. Having selected a fitness function, you decide to mutate weights randomly, and choose about half of the weights from each parent for crossover. Your friend uses the GA to train an NN on the example knock sequence, and it consistently says true for that knock sequence. Excited, he installs it, goes outside, waits for it to lock, someone runs by, and the door opens. What was missing from his training data?

¿ When not to open

6. Having added that, he retrains the system on all the training data, and it's classifying things perfectly, and he goes outside and waits for it to lock, knocks the secret pattern, trying again and again, but you

Over fit?

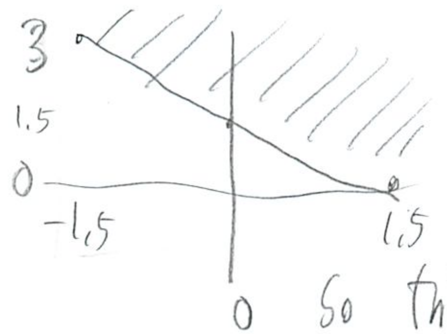
10

Remember broad Plan

And

$128x + 128y > 173$

WA



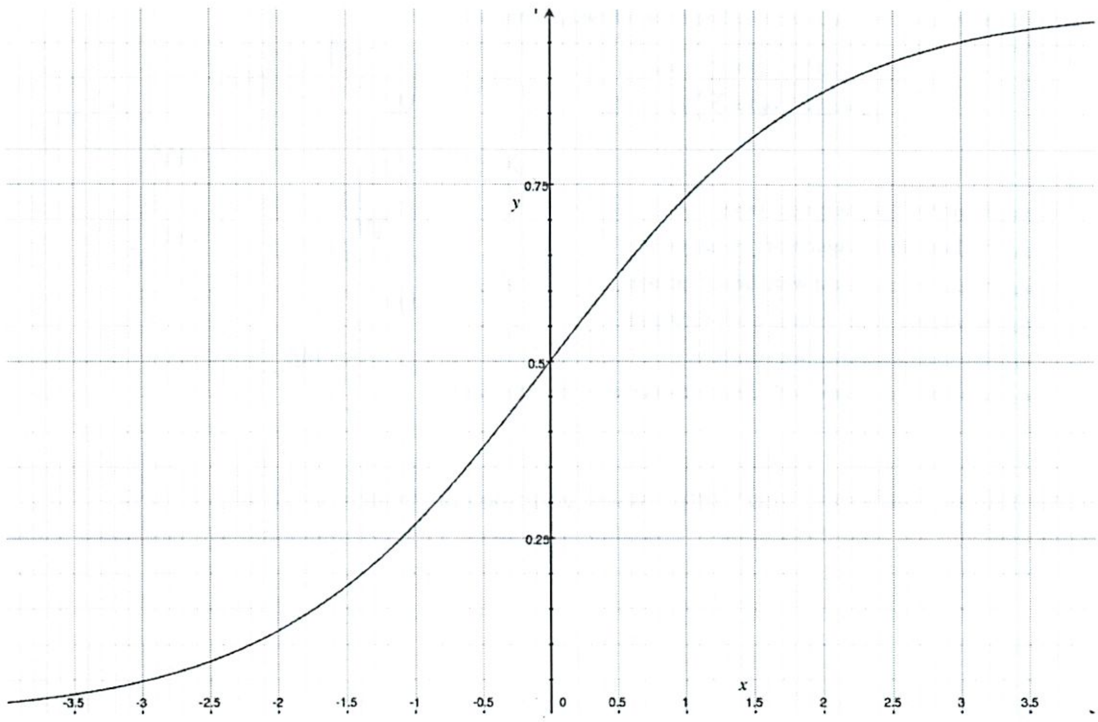
So this is AND

- Only 1,1 inside

Basically where is (0,0)
(0,1)
(1,0)
(1,1)
?

eventually have to let him back in. What happened?

Tear-off sheet



A	B	T	W_A	W_B	W_T	Σ	y	y*
0	0	-1	0	0	1			1
0	1	-1						1
1	0	-1						0
1	1	-1						1

Perceptron update:
if $|y^*-y| > 0$:
for each w_i:
$w'_i = w_i + r(y^*-y)x_i$

$$E = \frac{1}{2}\sum_k (o_k - d_k)^2$$

$$w_{i\to j} = w_{i\to j} - \Delta w_{i\to j}$$

$$\Delta w_{i\to j} = R \times o_{l_i} \times \delta_{r_j}$$

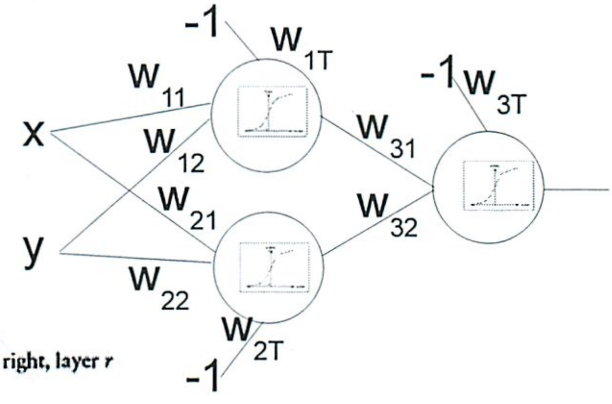
where R is a rate constant and the δs are computed with the following formulas:

$$\delta_k = o_k(1 - o_k) \times (o_k - d_k)$$

$$\delta_{l_i} = o_{l_i}(1 - o_{l_i}) \times \sum_j w_{i\to j} \times \delta_{r_j}$$

where

- o_k is output k of the output layer
- d_k is the desired output k of the output layer
- δ_k is a delta associated with the output layer
- o_{l_i} is output i of left layer in a left-right pair
- δ_{l_i} is a delta associated with the layer l
- δ_{r_j} is a delta associated with the adjacent layer to the right, layer r



For your entertainment after the quiz: http://www.youtube.com/watch?v=zE5PGeh2K9k

6.034 Quiz 3
November 9, 2009

Name	RONNIE WOOD
EMail	

Circle your TA and recitation time, if any, so that we can more easily enter your score in our records and return your quiz to you promptly.

TAs
Erica Cooper
Matthew Pealrs
Ronnie Wood
Mark Seifter
Yuan Shen
Jeremy Smith
Olga Wichrowska

Thu	
Time	Instructor
12-1	Gregory Marton
1-2	Berwick/Marton
2-3	Berwick/Marton
3-4	Berwick/Marton

Fri	
Time	Instructor
1-2	Randall Davis
2-3	Randall Davis
3-4	Randall Davis

Problem number	Maximum	Score	Grader
1	50		
2	50		
Total	100		

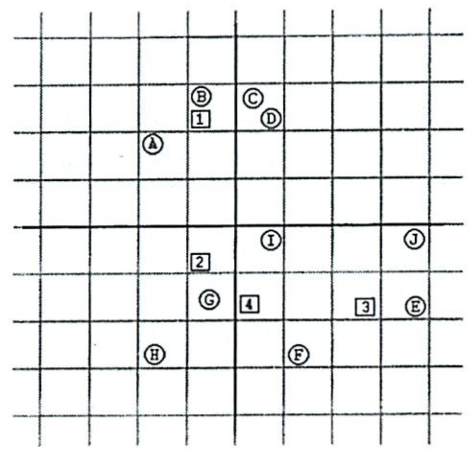
There are 12 pages in this quiz, including this one. In addition, tear-off sheets are provided at the end with duplicate drawings and data.

As always, open book, open notes, open just about everything.

Problem 1:KNN & ID Trees (50 points)

Part A: K Nearest Neighbors, backwards (15 pts)

Shaun has been hired to investigate the recent zombie infection in his hometown. The first thing Shaun needs to do is to make sense of the incomplete data provided to him. In the graph below, the circles correspond to observed people, but their labels, "zombie" or "healthy", were lost during the initial investigation. The square points represent people who still need to be classified (they are not themselves used to classify any other points).



Shaun is also given the table below, showing how the square points would have been classified using 1- and 3-nearest neighbors before the labels were lost. Given the map and the table below, Shaun needs to recover the original labels.

Square point	Using 1-nearest-neighbors	Using 3-nearest-neighbors
1	zombie	zombie
2	healthy	zombie
3	healthy	healthy
4	?	?

A1: Write down whether the following specimens are certain to be zombies (Z) or healthy (H) If you cannot be sure, write down unknown (U).

Circle A: U

Circle B: Z

Circle C: U

Circle D: U

Circle E: H

Circle F: U

Circle G: H

Circle H: Z

Circle I: Z

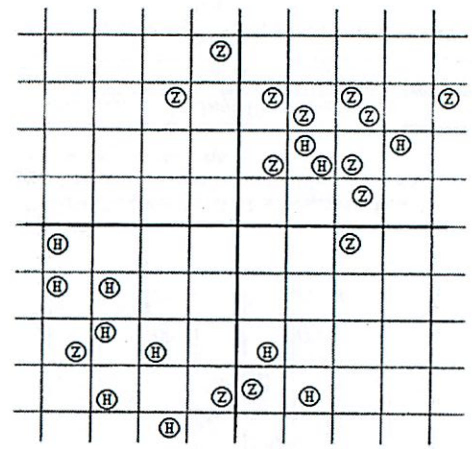
Circle J: U

A2: How would point 4 be classified? (Again, choose Z, H, or U)

Using 1-nearest neighbor: H

Using 3-nearest neighbors: U

A3: Shaun is wondering whether this k-nearest-neighbor algorithm is really reliable. He decides to check it on some labeled zombie data from a neighboring town. In the graph below, zombies are labeled Z, and healthy people are labeled H.

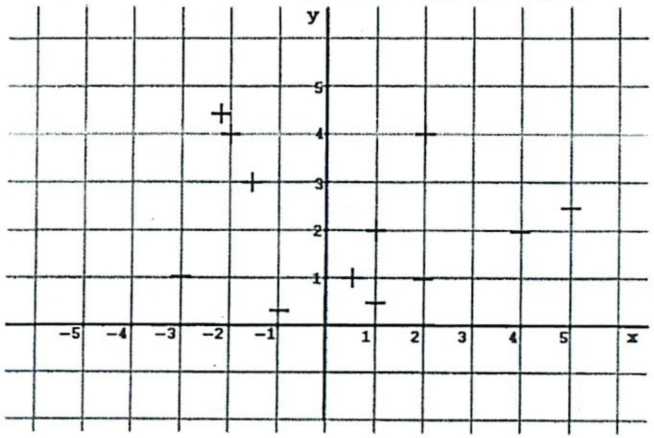


Describe, in a sentence or two, what happens to the accuracy of k-nearest neighbors as k increases from 1 to 26 (the total number of samples).

The accuracy increases initially (e.g. from k=1 to k=6) but decreases as k gets large (e.g. at k=26, everyone is classified as a zombie)

Part B: ID Trees (35 pts)

Shaun's best friend Ed, who was in the middle of a reconnaissance mission in another town, managed to send in a bit more data before he was bitten. Shaun overlays the locations of the known zombies (marked with a +) and known healthy people (marked with a -) on a grid representing the town. **The zombies are currently not biting anyone, so you can trust the points not to change over time.** Shaun' boss has tasked him with figuring out where to build a series of walls separating the healthy people from the zombies. The walls will be built along the decision boundaries created by the identification tree algorithm.



B1. (13 pts)

Shaun's girlfriend Liz suggests building a wall at y=1.5. Compute the disorder at this decision boundary. Leave your answer only in terms of integers, fractions, arithmetic operations, and logarithms.

$$\frac{5}{12}\left(-\frac{1}{5}\lg\frac{1}{5}-\frac{4}{5}\lg\frac{4}{5}\right)+\frac{7}{12}\left(-\frac{2}{7}\lg\frac{2}{7}-\frac{5}{7}\lg\frac{5}{7}\right)$$
$$=-\frac{1}{12}\lg\frac{1}{5}-\frac{1}{3}\lg\frac{4}{5}-\frac{1}{6}\lg\frac{2}{7}-\frac{5}{12}\lg\frac{5}{7}$$

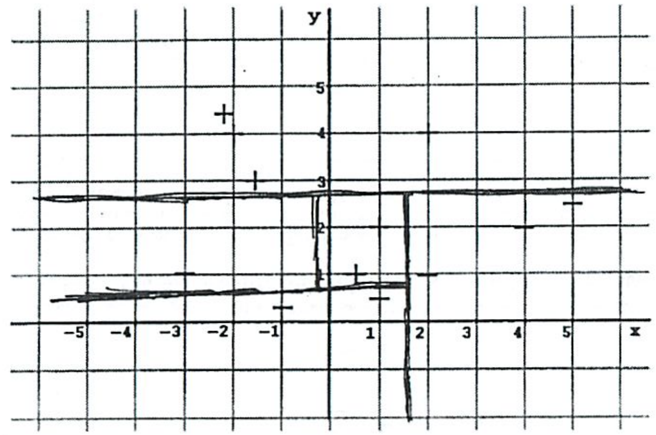
Shaun's flatmate Pete strangely insists that, instead, a wall should be built at x=-4. Compute the disorder at this boundary.

1

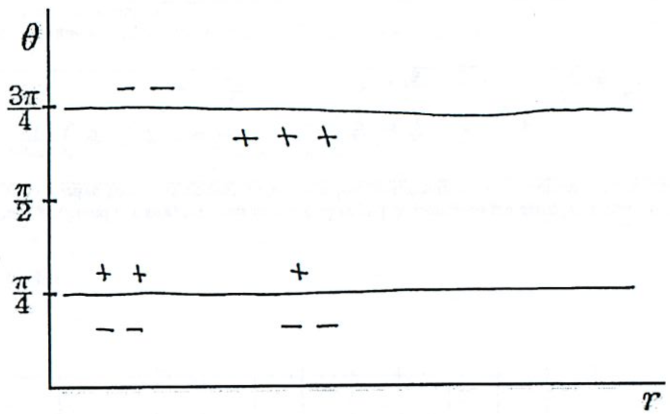
Whose idea is better, according to the heuristic described in class? (circle one)

Liz's Pete's

B2. (12 pts)
On the diagram below, ignoring the suggestions of Pete and Liz, draw the decision boundaries Shaun would produce using the identification tree algorithm. In case two decision boundaries are equally good, use the horizontal one. If there is still a tie, use the one with the lower-valued threshold. You will not need a calculator to solve this problem.

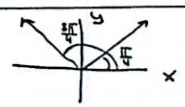


B3. (10 pts)
Shaun realizes that building all these walls is going to take a long time. In order to find out whether he can build a smaller number of walls instead, he decides to convert his data into polar coordinates. **Sketch** the 12 points from the previous graph on the polar graph below (making sure they still show the + and - labels). Show the decision boundaries produced by the identification tree algorithm on the same graph.



Describe in one sentence (or function) how the decision boundaries translate to walls on the original, x-y-plane graph.

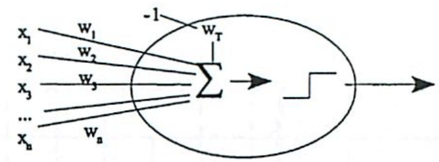
$y = |x|$



Question 2: Neural Networks and Genetic Algorithms (50 points)

Part A (10 points)

Perceptrons are the basic units of neural networks as we have seen them. They take a list of inputs **x**, multiply them by a list of corresponding weights **w**, compute the sum of those products, and pass the result through a decision function. We use a "fake" input T, usually -1, times an associated weight w_T, as part of the sum. When using a single perceptron for classification, one usually uses a threshold decision function: if the sum $z > 0$, the output is 1, otherwise 0.



To explore what perceptrons can and cannot do, we will ask you to make up weights, rather than training them. **You must use integer weights if integer weights are possible.**

1. Consider the boolean function A→B (same as "B or not A"). For inputs and output, true is represented as 1 and false as 0. Can a single perceptron with inputs A and B output 1 iff A→B?
If yes give weights: w_A= -2 w_B= 2 w_T= -1 for example
If not, why not?

2. Can a perceptron capture inequalities: given two real-valued inputs A and B, can the perceptron output 1 if A<B and 0 otherwise?
If yes give weights: w_A= -1 w_B= 1 w_T= 0 for example
If not, why not?

3. You wonder about the real-valued function A=B within epsilon E, that is, with three inputs, A, B, and E, can a perceptron capture whether |A-B| < E?
If yes give weights: w_A=______ w_B=______ w_E=______ w_T=______
If not, why not?

a perceptron can encode one linear boundary, and this has 2: $y = x \pm \epsilon$

Part B (16 points)

In this part of the question, we ask you to train a perceptron to learn A→B. We provide initial values for the weights. You are to use a learning rate, **r = 1.**

For the first sample, find z and y using the perceptron update algorithm (given on tear off sheet) with a **sigmoid decision function.** Use y to determine the next set of weights (which you are to write on the line for sample 2). Then, repeat using the second sample.

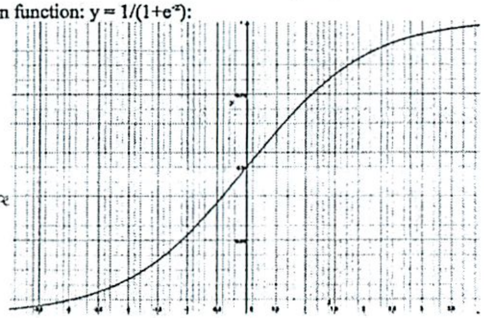
For the third and fourth samples, repeat, but use the **threshold decision function.**

Sample	A	B	T	W_A	W_B	W_T	$\Sigma x_i w_i = z$	use decision function:	y	y*	$y^* - y$
1	0	0	-1	0	0	1	-1	$y=1/(1+e^{-z})$	.27	1	.73
2	0	1	-1	0	0	.27	-.27	$y=1/(1+e^{-z})$	.43	1	.57
3	1	0	-1	0	.57	-.3	.3	y=1 if z>0; else y=0	1	0	-1
4	1	1	-1	-1	.57	.7	-1.14	y=1 if z>0; else y=0	0	1	1
				0	1.57	-.3					

W_B same for 3 & 4: B was 0

y at 3 and 4 must be 0 or 1!

The sigmoid decision function: $y = 1/(1+e^{-z})$:



Note:

	w_A	w_B
2	0	0
3	0	

because A & B are zero there.

Alternate, incorporating $y(1-y)$ into the first two updates:

A	B	T	W_A	W_B	W_T	$\Sigma x_i w_i = z$	use decision function:	y	y*
0	0	-1	0	0	1	-1	$y=1/(1+e^{-z})$	.27	1
0	1	-1	0	0	.86	-.86	$y=1/(1+e^{-z})$	.3	1
1	0	-1	0	.15	.71	-.71	y=1 if z>0; else y=0	0	0
1	1	-1	0	.15	.71	-.56	y=1 if z>0; else y=0	0	1
			1	1.15	-.29				

weights same, no update

Part C (4 points)

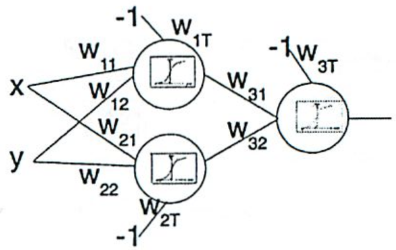
Which of the following is true about any perceptron when we use a sigmoid instead of a threshold?

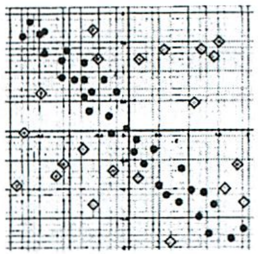
A. The perceptron can learn XOR
B. The perceptron can no longer learn all linear classification boundaries
C. The perceptron will learn A→B in fewer training steps (within, say, epsilon = 0.01 of 0 or 1)
D. The perceptron will learn A→B in more training steps (within, say, epsilon = 0.01 of 0 or 1)
E. None of these is true

The decision function doesn't change fundamental characteristics of what kinds of functions a perceptron can learn. However, a threshold decision function pushes weights out to the rails of a decision, while a sigmoid approaches the weights a little at a time, as you saw in part B. It therefore takes more iterations of training to get to a more "certain" classification, in those cases where learning is possible. With thresholds, A→B converges in 6 rounds of training. In 17 rounds the y from a sigmoid perceptron is within .01 of 0 or 1. In >150 it is within 0.0001. Option D is true.

Part D (12 points)

Given this three-node neural network, and the training data on the right





D1 Which of the following sets of weights will correctly separate the dots from the diamonds?

	W11	W12	W1T		W21	W22	W2T		W31	W32	W3T	
A:	-2	-2	-1	-x-y>-½	3	3	-1.5	x+y>-½	128	128	173	AND
B:	2	-1	1	wrong slope	2	-2	1		100	100	50	

Circle one: neither | A only | B only | both

dots are 1s, diamonds are 0s

D2. The following weights will separate the dots from the diamonds:

W11	W12	W1T		W21	W22	W2T		W31	W32	W3T	
4	4	2	x+y>½	6	6	-3	x+y>-½	96	-95	-52	B→A

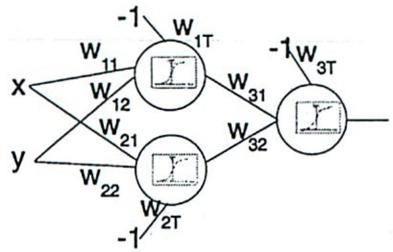
from page 8

Write an expression for how this works using inequalities and boolean operations:

$(x+y>-\frac{1}{2}) \rightarrow (x+y>\frac{1}{2})$

dots are 0s, diamonds are 1s

Part E (8 points)



While training the three-node neural network at the top of this page (same as in part D), you may start an iteration with the weights:

W11	W12	W1T	W21	W22	W2T	W31	W32	W3T
3	2	1	4	5	-2	-100	-100	-150

For the training example x=0 y=0, if y*=1, what is δ_1(the delta for the top-left node)? See the tear-off sheet for notes on back-propagation. You can leave your solution expressed as a product, and it may help us assign partial credit.

$\emptyset$

$y_1 = .27$

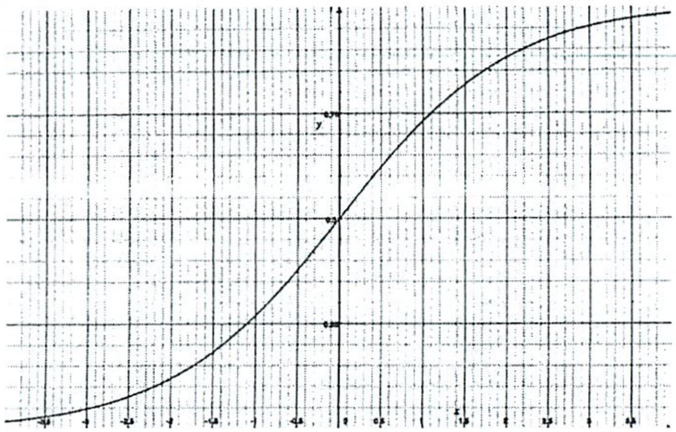
$y_2 = .88$

$y_3 = 1$

$y^* - y = \emptyset \qquad \delta_3 = 1(1-1)(1-1) = \emptyset$

$\delta_1 = y_1(1-y_1) \cdot \sum_{i \in \{3\}} w_{1i} \cdot \delta_i = .27(1-.27)(-100)(0) = \emptyset$

Tear-off sheet



Perceptron update:

if |y*-y| > 0:

for each w_i:

$$w'_i = w_i + r(y^* - y)x_i$$

Neural net update:

$$E = \frac{1}{2}\sum_k (o_k - d_k)^2$$

$$w_{l\to r}' = w_{l\to r} - \Delta w_{l\to r}$$

$$\Delta w_{l\to r} = R \times o_l \times \delta_r$$

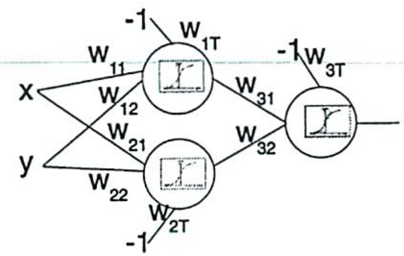
where R is a rate constant and the δs are computed with the following formulas:

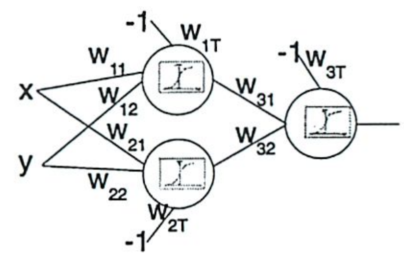
$$\delta_k = o_k(1 - o_k) \times (o_k - d_k)$$

$$\delta_l = o_l(1 - o_l) \times \sum_r w_{l\to r} \times \delta_r$$

where

- o_k is output k of the output layer
- d_k is the desired output k of the output layer
- δ_k is a delta associated with the output layer
- o_l is output l of left layer in a left-right pair
- δ_l is a delta associated with the layer l
- δ_r is a delta associated with the adjacent layer to the right, layer r



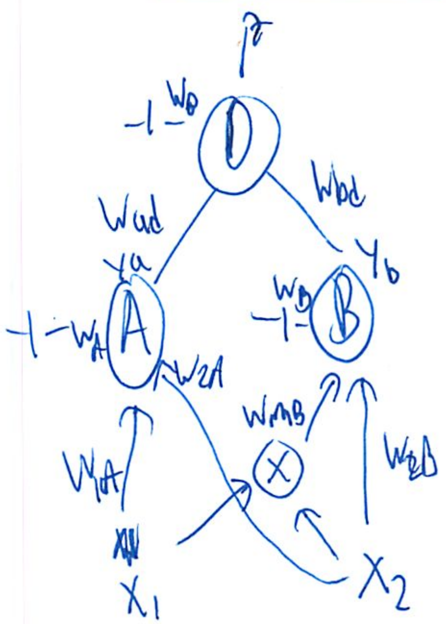


6.034 Tutorial

11/14

Trend this year: TAs writing very hard problems
- need to connect the dots + jump

Pineapples Neural Net problem



Pretty standard
- except for multiplier

Sigmoid function

A) ΔW_{bd} in terms of variables

$$\Delta W_{bd} = r\, \partial_i\, i_i$$

$$= r \left(o_f (1-o_f) \underbrace{(d-o)}_{\frac{\partial P}{do}} \right) i_i$$

$\frac{\partial o_f}{\partial p_r}$ for sigmoid

$$= r \left(z(1-z)(d-z) \right) y_b$$

②

$\partial_{final} \neq \partial_{internal}$

See last week - different math

internal nodes - take into outputs it effects

Should be able to see when do derivites

└ Should be able to do

---

internal

$$\partial_l = O_l(1-\partial_l) \cdot \underbrace{\sum_j W_{ij}\, \partial_j}$$

note the sum

we sometimes don't have this

$$\Delta W_{mb} = r\, \partial_i\, i$$

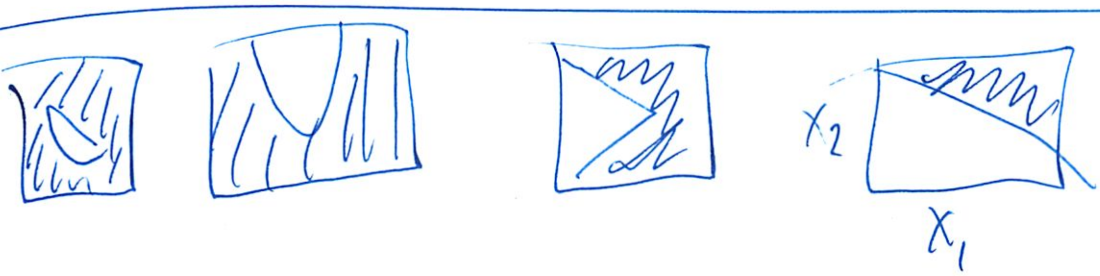
$$= r\left[Y_b(1-Y_b)\, W_{bd}\, z(1-z)\,(d-z)\right](x_1\, x_2)$$

↑ output of multiply node

③

(I was one of the only people who studied for this
- now I know the concepts here)

TA: But this is easiest way they could ask this problem
- no derivs, other sigmoid fns, etc



Which things could be produced by our perceptron?

- we can make 2 since its not a simple perceptron
- curvey part: can be done by the multiply
  - gives you radial function
  - or some name to that effect

$X_1 \cdot X_2 W_{mb} + X_2 W_{213} - W_b$

W_{mb}: can set to 0 or 1, set to 1
W_{213}: set to 0
W_b: -1

$X_1 X_2 - 1 > 0$

(4)

$X_2 = \frac{1}{X_1^2} \left(\cancel{W} (1-z) \right) (etc \quad)$ not clear on this

But can't do parabola

— need squared term

 Is easy

Set W_{bd} to 0

And only work w/ simple perceptron A

$X_1 W_{1A} + X_2 W_{2A} - W_A = 0$

? if did not have could only do vertical cut (depends on axis)

? just make line at first

Then can do shading < >

depends on what is (+) (−)

 Can we do this?

No? Multiplier needs radial curve

This needs 2 simple perceptrons

But can set W_{mb} to 0?

(5)

But then connection would not exist simply $X_1 \to B$

So now can do 1 simple perceptron (diagonal)

now but other one can only be vertical

Can we control the constants to make the line we want?

$$X_1 W_{1A} + X_2 W_{2A} - W_A = 0$$

Equation: rewrite

$$X_2 = \frac{-W_{1A}}{W_{2A}} X_1 + \frac{W_A}{W_{2A}}$$

X_2 labled as Y

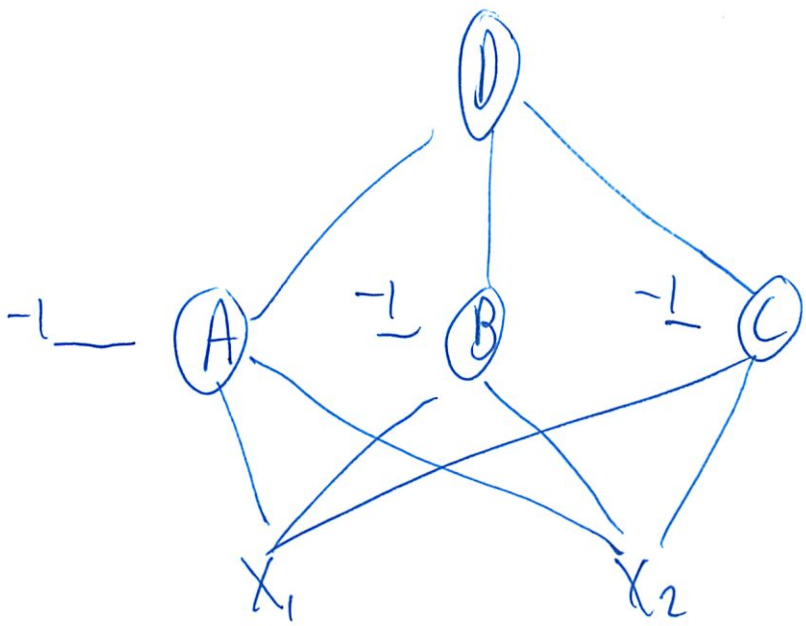
write in terms of Y

control these

like in 6th grade line drawings

(This is 2006 quiz 3)

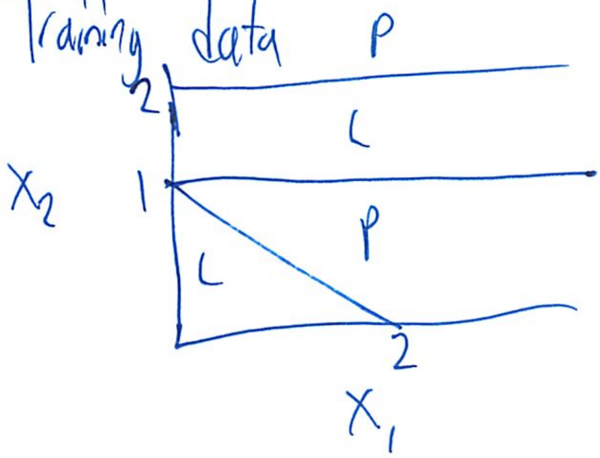
6



Weights on evey line
as ya expect

Now threshholds
- make eaiser
- even if says sigmoid - threshhold easy to calc

Training data



P=Pinaple

$W_a = -2$	$W_b =$	$W_c =$	$W_d =$
$W_{1a} = -1$	$W_{1b} =$	$W_{1c} =$	$W_{ad} = 4$
$W_{2a} =$	$W_{2b} =$	$W_{2c} = -1$	$W_{bd} = 2$
			$W_{cd} =$

Lots of stuff not given

(7)

First figure out what lines need to draw

$X_2 = 1$

Which is secretly y

$X_2 \, \mathbb{R} = 2$

$X_2 = -\frac{1}{2} X + 1$

So basically have (A), (B), (C) (simple perceptrons) do each of the lines

And then (D) be or ;

We have $W_{1A} X_1 + W_{2A} X_2 - W_A = 0$

$$W_{1B} X_1 + W_{2B} X_2 - W_B = 0$$

$$W_{1C} X_1 + W_{2C} X_2 - W_C = 0$$

But have constraints - plug in

$$-X_1 + W_2 A \, X_2 + 2 = 0$$

$$W_{1B} X_1 + W_{2B} X_2 - 1 = 0$$

$$W_{1C} X_1 - X_2 - W_C = 0$$

8

Now make one of them $X_2 = 1$

└ Need to pick which one - lets say the 2nd one

$$0\,x_1 + 1\,x_2 - 1 = 0$$

$\hookrightarrow X_2 = \underline{1}$

So $W_{1B} = 0$

$W_{2B} = 1$

$W_B = -1$

Now put third one to $x_2 = 2$

$$0\,x_1 - x_2 - 2 = 0$$

$\hookrightarrow X_2 = -2$

Must be integers!

Now first one $x_2 = -\frac{1}{2}x_1 + 1$

$$-1\,x_1 + -2\,x_2 - 2 = 0$$

$\hookrightarrow x_2 = -\frac{1}{2}x_1 + 1$

(9)

Now need (D)

$W_0 = ?$

$W_{ad} = 4$

$W_{bd} = 2$

$W_{cd} = ?$

So need to

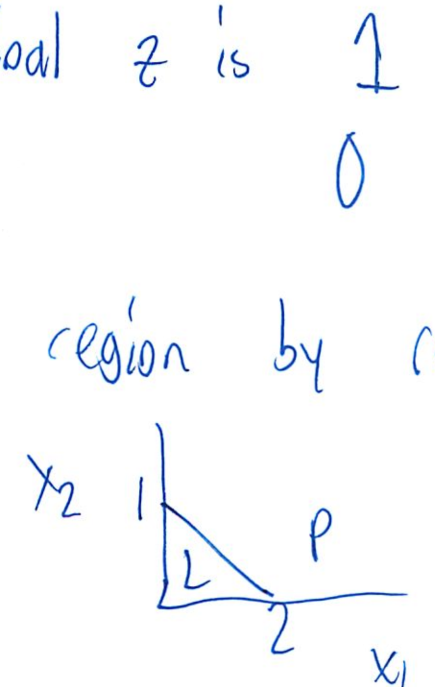
A	B	C	D
✓	x	✓	✓

✓ = True

Sols are not unique

Goal z is 1 in L region) given
0 in P region

Go region by region



(10)

Note

C $X_2 = 1$

B $X_2 = 2$

A $X_2 = -\frac{1}{2} X_1 + 1$

Want $\cancel{A}$ z to 1 in L region

- so if $-\frac{1}{2} x_1 + 1 < 0$ ← below line true

A $\quad 2 < 0$ ← below line true

B $\quad 1 > 0$ ← above line true

C

↙ tells us which nodes are on in each region

what we want

~~Don't go on region by region~~

Have our 2 lines weights already

↙ B off since 7

$4 + W_{CD} - W_D > 0$

↑ when on times 4

↑ must be true since what we wrote above

Threshold fns 1 if on 0 otherwise

So for P region

$W_{cd} - W_d < 0$

↑ want off

$\boxed{< 0 \text{ is off}}$

$\boxed{> 0 \text{ is on}}$

Write inequalities for each region so that it is what we want

Which of A, B, C is on for this

(11)

For L region

$W_{bd} + W_{cd} - W_d > 0$

For P region

everything off except B

$$2 - W_d \le 0$$

A is on only in Bottom region

~~D is on every where~~

C is on everywhere except top

WD is hidden - always there

So now need values

- solving 2 unknowns w/ 4 eqns

$W_d > 2$

$W_{cD} < W_D$

which is > 2

$W_{cD} < 2$ so can be 1 or 2

12

Since $W_{cd} > W_d - 2$

$\cancel{W_{cd} \neq W_d}$

$W_{cd} = W_d$ or $W_d - 1$

So $W_d = 2$

$W_{cd} = 1$ that works

Neural Nets Review + Practice

11/15

$$X_2 = \frac{-W_{1A}}{W_{2A}} X_1 + \frac{W_A}{W_{2A}}$$

↑
like y
on axis

From

$$X_1 W_{1A} + X_2 W_{2A} - W_A = 0$$

Try to get

+ sign error

$$X_2 = \frac{W_A}{W_{2A}} \overset{+}{-} \frac{X_1 W_{1A}}{W_{2A}}$$

So figure out lines to draw

$X_2 = 1$

$X_2 = 2$

$X_2 = -\frac{1}{2}x + 1$

have some weights preset

(2)

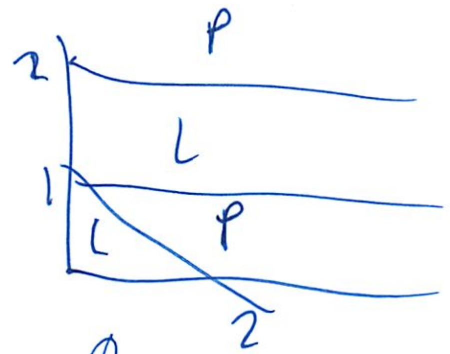
Like $0x_1 + 1x_2 - 1 = 0$

$\uparrow w_{1B}$ $\uparrow w_{2B}$ $x_2 = 1$ $\uparrow w_B = -1$

This I all got

But for (D)

Look at regions



Goal $z = \begin{cases} 1 & L \text{ region} \\ 0 & P \text{ region} \end{cases}$

So for this L region

$0 > -\frac{1}{2}x + 1$

since less than line, is threshhold

↳ becomes 1

$0 < 2$

$0 < 1$ So $w_{AD} + w_{BD} + w_{CD} > 0$ where

w_{AD} given 4, w_{BD} given 2

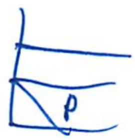
Why did I write $w_{AD} + w_{CD} - w_D > 0$

(3)

He went over this too fast...

No b since ~~a~~ ?

└ so essentially it does not apply?

P region 

$W_{a} - W_{d} < 0$

? want it off

?, So $-\frac{1}{2}x_1 + 1 < 0$ ← below line true

but we above line so don't write?

~~aa~~ $2 < 0$ ← below line true

? but we are within here

Or wrong value

└ but how do those have values?

4

$$W_{BD} + W_{cd} - W_d > 0$$

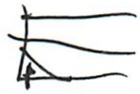
$$W_{bd} - w_d \leq 0$$

A is on ~~also~~ only in bottom region
C is on everywhere except top
?? So we look at where the rule is "on"?

So boundries were

A	$X_2 = -\frac{1}{2}x_1 + 1$	$-\frac{1}{2}x_1 + 1 < 0$	below line
B	$X_2 = 2$	$2 < 0$	below line
C	$X_2 = 1$	$1 > 0$	above line

(5)

So then for 

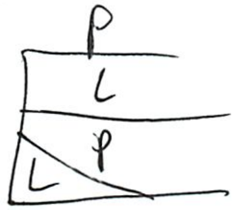
A, B is on

$W_{AB} + W_{BD} - W_D < 0$

want to be off

I had A, C on

So where did we get earlier lines?



want 1 in L

So $-\frac{1}{2}x_1 + 1 < 0$

A

below line = 1

B

$2 < 0$

C

below line = 1

$1 > 0$

above line → 1

So for our L

really only need $-\frac{1}{2}x_1 + 1 < 0$

Or on is all the $<$?

6

So let me try that

L is 1

So > 0

; So all < 0

$W_{AO} + W_{BO} + W_{CO} - W_O < 0$

Nah - does not make

let me try next one

Want Off

A - want off

B - ? could have on

C - want on ← they just use C

> 0

B, C on ? ✓

< 0

just B on ? ✓

(7)

So I feel  shall all be on?

 < 0

B + C on

So then what? (even though this is not resolved?

Write inequalities

? did he make mistake
— he was rushing on board

$$4 + W_{CD} - W_D > 0$$

$$W_{CD} - W_d \leq 0$$

$$2 + W_{CD} - W_C > 0$$

$$2 - W_D \leq 0$$

$\rightarrow 2 < W_D$

$\rightarrow W_{CD} < W_d$

So $W_{cd} < 2$

can be 1 or 2

8

Then fill in

$$W_{co} \geq W_d - 2$$

Multiple possible integers

Try some

I still don't get the first 2 sections

L think he might have done it wrong

Emailed in

- was class example wrong

Michael E Plasmeier

From: Erek Speed <espeed@MIT.EDU>
Sent: Tuesday, November 15, 2011 10:20 PM
To: Michael E Plasmeier
Subject: Re: Question on Example in Class

I don't think there are any hard and fast rules for choosing the inequalities. As in there are possibly more than one possible ways to choose them. Just like α, β!

My heuristic is to choose inequalities so that whichever side of the boundary is positive is 'on' (outputs a 1 if it's a threshold function). It has always worked for me so far, though I have no analysis for it.

This leads to the equations that you listed in your email from tutorial. I looked at your notes and you have 2<0 and 1>0 for your inequalities. These should be x2 <2 and X2 >1 for C and B respectively. missed up what is 0
It looks like you have the C and B line switched around which might be some cause of the confusion. The B line should be X2 =1. I think I wrote it wrong at one point and someone corrected me later.

Given that, you have what that means written in words. ok mis wrote some stuff

For A: "below line true" A
For B: "Below line true" (actually C) C
For C: "above line true" (actually B) B

"Tells us which nodes are on in each region."

This is correct. So for the bottom region because it is below the A line and below C but below B (when it should be above) the active nodes are just A and C. So I mixed it up

Does this make sense?

Erek

2011/11/15 Michael E Plasmeier <theplaz@mit.edu>:
> Did you make a mistake on the first 2 sections? (Page 10 of PDF are my
> notes from that day)
>
>
>
> So I see how we get the lines (to get where 1 or 0)
>
> A −(1/2)+1 < 0
>
> B 2 > 0
>
> C 1 > 0
>
>
>
> But then shouldn't the first diagonal region be all rules on. (You had
> A, C)

>
> The next region (P) should be B and C (You had just C)
>
> Then my thinking gets the same answers for section 3 (B and C)
>
> And the last section (just B)
>
>
>
> What am I doing wrong? How do we select which rules are active for
> each section? Is this like an OR gate?
>
>
>
> If it is faster, my phone # is 610 513 0390
>
>
>
> Thanks!!!
>
>

Massachusetts Institute of Technology

Department of Electrical Engineering and Computer Science
6.034 Artificial Intelligence, Fall 2011
Recitation 8, November 3 **Corrected Version & (most) solutions**

Neural networks II Prof. Bob Berwick, 32D-728

0. Introduction: the summary so far (and solutions from last time)

Summary of neural network update rules:

To update the weights in a neural network, we use *gradient ascent* of a performance function P by comparing what a network outputs given a sample data point and given the network's current weights, via **forward propagation**. We compare the network's output value against the desired value in terms of the partial derivative of $P = -1/2(d-o)^2$ with respect to particular weights w_i. This is called **backpropagation.** Recall that the first step is to find the derivative of P with respect to the output, which turns out to be: $(d-o)$.

The general formula for the change in weights is:

$$\Delta w \propto \frac{\partial P}{\partial w} \text{ so } w' = w + \alpha \times \frac{\partial P}{\partial w} \text{ where } \alpha \text{ is a rate constant (also } r \text{ in the literature \& quizzes)}$$

The value of alpha (aka r) is also the "step size" in the hill-climbing done by gradient ascent, using the performance fn.

For the **final** layer in a neural network, whose output from forward propagation is o_f and where the desired output value is d, the required change in weight value for a (single) final weight is:

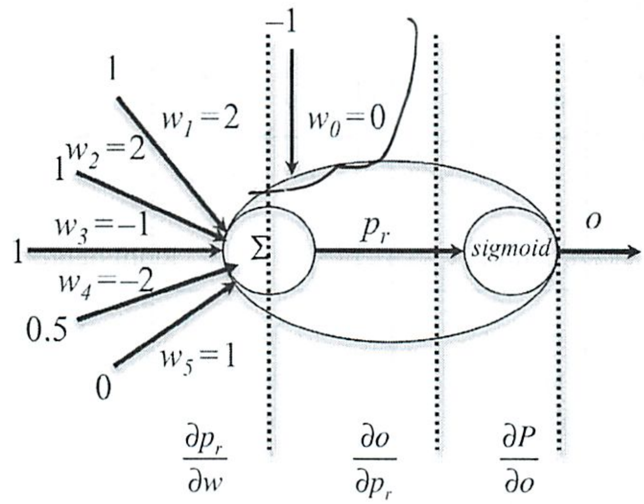
1. $\Delta w_r = \Delta w_f = \alpha \times \delta_f \times i_f$ where $\delta_f = o_f(1-o_f)\times(d-o_f)$

So many diff ways to write!

For the **previous** layer in a neural network (just the rightmost layer if a single neuron), the required update equation is:

2. $\Delta w_l = \alpha \times o_l(1-o_l)\times w_r \times \delta_f \times i_l$

Example 1. Last time we computed the weight updates for a single-layer neural network with 6 inputs and 6 weights. Each partial derivative in the figure below corresponds to a different part of the network, with their product yielding the derivative of P with respect to the weights w, where the desired output was 1, and the learning rate alpha was (arbitrarily) set to 100:



Step 1: Forward Propagation. Calculate the output o given the input values shown. Useful data point: *sigmoid*(2)= 0.9

Answer: $-1 \times w_0 +$ __1__ $\times w_1$ __1__ $\times w_2 +$ __1__ $\times w_3 +$ __0.5__ $\times w_4 +$ __0__ $\times w_5 = p_r = 2$

Sigmoid $(p_r) = o_f =$ __0.9__

stick in sigmoid collect inputs get val

Step 2: Backpropagation to find delta for final, output layer.

$$\delta_f = \frac{\partial P}{\partial p_r} = \frac{\partial P}{\partial o}\frac{\partial o}{\partial p_r} = (d - o_f) \times [o_f(1 - o_f)]$$

$$\frac{\partial P}{\partial w} = \frac{\partial P}{\partial o}\frac{\partial o}{\partial p_r}\frac{\partial p_r}{\partial w} = (d - o_f) \times [o_f(1 - o_f)] \times i_f = \delta_f \times i_f$$

$\Delta w_f = \alpha \times i_f \times \delta_f$ (for each input line to the neuron, i)

$w_i' = w_i + \Delta w_f$ (for each input line to the neuron, i)

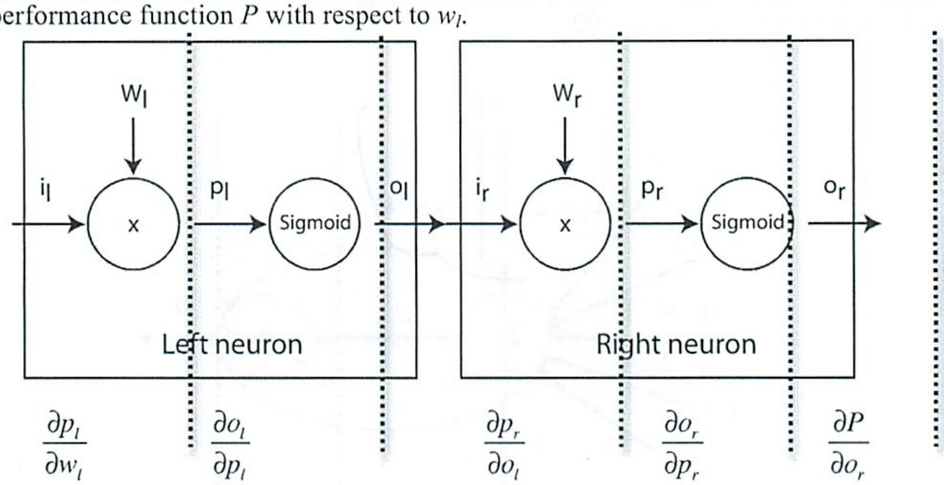
here same in all since feeds to same at

NEW WEIGHT	ORIGINALWEIGHT +	RATE ×	δ ×	INPUT =	NEW WT
w'	w	α	(d-o)(o)(1-o)	i_f	
w_0' =	0	100	0.009	–1	–0.9
w_1' =	2	100	0.009	1	2.9
w_2' =	2	100	0.009	1	2.9
w_3' =	–1	100	0.009	1	–0.1
w_4' =	–2	100	0.009	0.5	–1.55
w_5' =	1	100	0.009	0	1

Note how weights that have an input of 0 to them can't affect the performance, so they remain unchanged. So, do these new weights get us closer to the desired output? If we run forward propagation again we can find out: $-1 \times 0.9 + 1 \times 2.9 + 1 \times 2.9 + 1 \times -0.1 + 0.5 \times -1.55 + 0 \times 1 = 6.65$; sigmoid(6.65) = 0. 99870765 then repeat "training"

Example 2. Last time, we also computed the value for the weight change for the final, output neuron of the simple two-neuron net below. We initially have i_l=1; both weights w_l and w_r = 0; and the desired output is 1. We will finish up this problem now.

For completeness, and to cement our understanding, let's see how the various terms in the partials are arrayed over this two-neuron diagram, pushing back from the output o_r, so you can see why it is called **backpropagation.** Make sure you **understand** where each of the five terms comes from. Multiplied together, they give the partial derivative of the performance function P with respect to w_l.



$\frac{\partial p_l}{\partial w_l}$	$\frac{\partial o_l}{\partial p_l}$	$\frac{\partial p_r}{\partial o_l}$	$\frac{\partial o_r}{\partial p_r}$	$\frac{\partial P}{\partial o_r}$
i_l	$o_l(1-o_l)$	w_r	$o_r(1-o_r)$	$(d-o_r)$
left weight deriv	left sigmoid deriv	right weight deriv	right sigmoid deriv	output deriv

break it down →

This is to find the partial of the performance function P with respect to the left right, w_l. Remember that to find the corresponding partial for the **final** output layer we computed something a bit different: the partial of p_r with respect to o_l is **replaced** with the partial of p_r with respect to w_r (so finding the partial of P with respect to w_r.) But this partial is just the derivative of $i_r \times w_r$ with respect to w_r, which is simply i_r. **Note how** if we decided to use a different threshold function other than a sigmoid, the **only two** things we would have to change are the two partial derivatives, the right and the left sigmoid derivatives with respect to p_r and p_l, respectively. For example, if we changed the sigmoid threshold from $1/(1+e^{-x})$ to, say, x^2, then the derivatives would change from, e.g., $o_r(1-o_r)$ (the derivative of the sigmoid function with respect to its input), to just $2o_r$ (and the same for the left threshold derivative).

harder to train

For this example, assume **all initial weights are 0** (it is actually a bad idea to set all initial weights the same for neural nets; why?). Assume a sample input of $i_l = 1$, and that the *desired* output value d is 1.0. Assume a learning rate of 8.0. (Useful data point: *sigmoid*(0) = 0.5) Let's run one step of backpropagation on this and see what's different about this case. First, as before, we must carry out **forward** propagation: compute the inputs and outputs for each node.

Step 1: Forward Propagation. OK, you should know the drill by now. First compute the outputs z at each node:

$p_l = w_l i_l$ = 0 × 1 = 0 So o_l (= i_r) = *sigmoid*(0) = 0.5

$p_r = w_r\, i_r$ = 0 × 0.5 = 0 So o_r = *sigmoid*(0) = 0.5

Step 2: Calculate the δ for the output, final layer, δ_f (i.e., the neuron on the right, for use in changing w_r)

Recall the formula for δ_f is: $o_r \times (1-o_r) \times (d - o_r)$= 0.5 × (1–0.5) × (1–0.5) = 0.125 .
Recall that $d = 1.0$; we have just computed o_r.
So, the change in the right-most weight w_f is: $\alpha_f \times i_r \times \delta_f$ = 8.0 × 0.5 × 0.125 = 0.5

Step 3: Calculate δ_l for the hidden neuron on the left, recursively using the delta from the previous layer:
$\delta_l = o_l(1-o_l) \times w_r \times \delta_f$ = 0.5(1–0.5) ×0 × 0.125 = 0
So the weight change for the left neuron at the first iteration of back propagation is 0

the output from previous

Thus the two **new weights** are:
w_f' = 0 + 0.5 = 0.5
w_l' = 0 + 0 = 0

Let's see how much closer this has gotten us to the desired output value of 1.0. We do this by another round of forward propagation (and then typically, we would do back-propagation again to get us even closer, many thousands of times.) Your turn now…. (See the tear-off page on the back to estimate the sigmoid to 2 decimal places, or better, user a calculator or python on your laptop…)

Next iteration, forward propagation:
$p_l = w_l i_l$ = 0 × 1 = 0 So o_l (=i_r) = *sigmoid*(0) = 0.5 .

$p_r = w_r i_r$ = 0.5 × 0.5 = 0.25 So $o_r = o_f$ = *sigmoid*(0.25) = 0.56218 .
So, we have definitely gotten a bit closer to our output goal of 1.0.

Next iteration, back-propagation:
Now you try it:
$\delta_f = o_f \times (1-o_f) \times (d - o_f)$ = 0.56218 ×(1– 0.56218) ×(1– 0.56218) = 0.10776 .

$\delta_l = o_l \times (1-o_l) \times w_r \times \delta_f$ = 0.5 ×(1– 0.5) × 0.5 × 0.10776 = 0.01347 .

$\Delta w_f = \alpha \times i_f \times \delta_f$ = 8.0 × 0.5 × 0.10776 = 0.43104

$\Delta w_l = \alpha \times i_l \times \delta_l$ = 8.0 × 1.0 × 0.0134 = 0.1072

$w_f' = w_f + \Delta w_f =$ $\underline{0.5 + \quad 0.43104}$ $= \quad 0.943104$

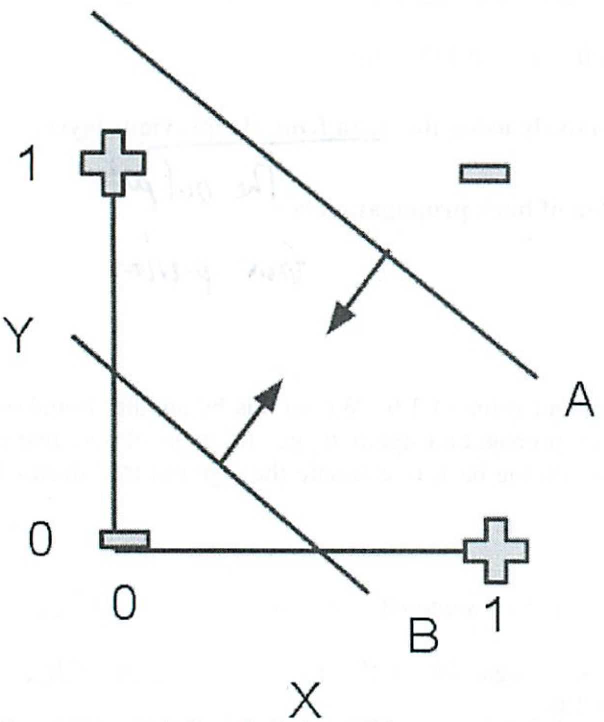
$w_l' = w_l + \Delta w_l =$ $\underline{0 \; + \; 0.1072}$ $= \quad 0.1072$

Do the new weights get us closer to the goal? Calculate this by **forward propagation** again:

$p_l = w_l i_l = \; 1 \times \; 0.1072$; $o_l \; (= i_r) \; = \;$ sigmoid(0.1072) $= 0.52677$

$p_r = w_r i_r = \; 0.943104 \; \times \; 0.52677 = \; 0.4968$; $o_r =$ sigmoid(0.4968) $= 0.62171$

Example 3. What multilayer neural networks can learn that single layer networks cannot learn.
Why did people invent multi-layer neural networks? Consider a classification problem such as the one depicted below, which represents the predicate or the 'concept' of excusive-OR (XOR), i.e., the value of this function is 1 if either of the two inputs is 1; and the value of this function is 0 if both inputs are 0 *or* both inputs are 1. The line A goes through the points (0, 3/2) and (3/2, 0); while the line B goes through the points (0, 1/2), (1/2, 0). The 'plus' signs are at (0,1) and (1, 0); the minus signs at (0,0) and (1,1).

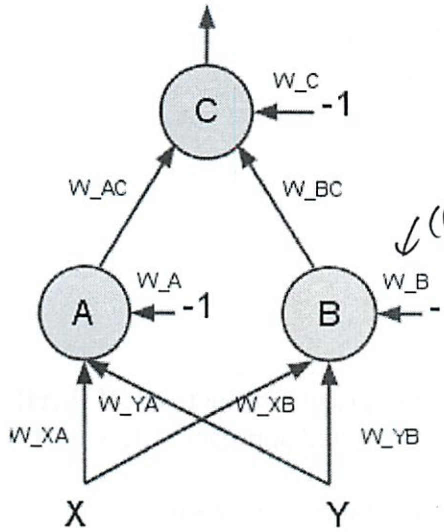


L what were these machines called?
Perceptrons
? a real 50s era name

Suppose we tried to find the weights to a *single* layer neural network to 'solve' this classification problem. Then the general formula for this network would be, output = $w_1 x + w_2 y + c$. But what weights would work? Do you see that this kind of equation can only define a *single* line? Thus, it says that we must classify the + and – regions in the graph above into regions that are all + (positive) and all – (negative), by making a *single* cut through the plane. Can this be done? Try it – why can't it be done? **Answer: you cannot do it, because a perceptron can only define a single line 'cut' through the plane, and this region of + is defined by two lines.**

Question: Can a perceptron encode function $|x–y| < epsilon$, for some positive epsilon? Why or why not?
Answer: No, again because the absolute value function requires two cuts through the plane.

However, if we are allowed *two* network layers, then we *can* formulate a set of weights that does the job. Let's see how, by considering the network below, and then finding the weights that do the job. (This was a sample quiz problem previously.)



can set these values

can weight it to whatever

can input value

Step 1. First, think of input-level neurons (neurons A and B) as defining *regions* (that divide positive data points from negative data points) in the X, Y graph. These regions should be depicted as linear boundary lines with arrows pointing towards the positive data points. Next, think of hidden level neural units (neuron C) as some logical operator (a linearly separable operator) that combines those *regions* defined by the input level units. (We will see later on a few more examples of this sort to show you how multi-layer networks can 'carve up' regions of the plane in this way.)

So in this case: units A, and B represent the **diagonal** boundaries (with arrows) on the graph (definition two distinct ways of separating the space). Unit C represents a logical AND that intersects the two regions to create the bounded region in the middle.

Step 2. Write the line equations for the regions you defined in the graph.

A) The boundary line equation for the region defined by line A:

$y < -1 \times x + 3/2$ ← write eqs for line

B) The boundary line equation for the region defined by line B:

$y > -1 \times x + 1/2$

Step 3. Rewrite the line equations into the form: $ax + by > c$, where a, b, and c are integers:

inequalities

A) $y < -1 \times x + 3/2$
$x + y \quad < 3/2$
$2x + 2y \quad < 3$ ←

B) $y > -1\ x + 1/2$
$x + y \quad > 1/2$
$2x + 2y \quad > 1$ ←

two sets of lines

Now note that the sum of the weights times the inputs for each unit can also be written in a similar form. (We will call this summed product of weights times the inputs for a neuron its "z" value).

For Unit A: $z =$

$W_{XA}\, x + W_{YA}\, y + W_A(-1) > 0$

$W_{XA}\, x + W_{YA}\, y \;\; > W_A$

For Unit B: $z =$

$W_{XB}\, x + W_{YB}\, y + W_B(-1) > 0$

$W_{XB}\, x + W_{YB}\, y \;\; > W_B$

Why do we set $W_{XA}\, x + W_{YA}\, y + W_A(-1) > 0$ and not < 0? Look at the graph on the tear-off sheet!
When $z = W_{XA}\, x + W_{YA}\, Y + W_A(-1) > 0$, then sigmoid($z > 0$), and z grows and approaches 1, which corresponds to the *positive* points/
When $z = W_{XA}\, x + W_{YA}\, Y + W_A(-1) < 0$, then sigmoid($z < 0$), z decreases and approaches 0, which corresponds to the negative points.
Thus, when expressed as > 0 the region is defined as **pointing towards** the **positive** points.
But when expressed as < 0, the region is defined as **pointing towards** the **negative** points.
We want the defined region to point to the positive points. So, we must adjust the equation for line (A) so that it has the inequality in the form > (rather than as <). We can do this by multiplying through by a –1, which will reverse the inequality, so the equation for line A becomes:
$-2x-2y > -3$
Now we are ready for the next step.

Step 5. Easy! Just read off the weights by correspondence.

$-2\, x \; + -\; 2\, y > 3$ line A's inequality

$W_{XA}\, x + W_{YA}\, y > W_A$ z equation for unit A. Therefore, $W_{XA} = -2$ $W_{YA} = -2$ $W_A = -3$

$2\, x \; + \; 2\, y > 1$ line B's inequality

$W_{XB}\, x + W_{YB}\, y > W_B$ z equation for unit B. Therefore, $W_{XB} = 2$ $W_{YB} = 2$ $W_B = 1$

Step 6. Solve the logic in the second neuron layer
The equation for the second layer unit C is W_{AC}(output from A) + W_{BC}(output from B) – W_C < or > 0 (where we pick the inequality to satisfy the sigmoid output description mentioned above – if we want the output to be 0 from the logic unit, then we want the sigmoid to go negative, so we want < 0; if we want he output to be 1, then we want the sigmoid to go positive, so we want > 0.) look at sigmoid when deciding
We now want to compute (A AND B), for the next layer. (Remember, the final region we want is the **intersection** of the regions defined by unit (line) A, and unit (line) B. So we build a Truth table for **And** and solve for the constraints. (**Note** that we are **not** building a truth table for **XOR** – we want **And.**) So we want the output from C to be true (1) iff the outputs from units A and B are both 1, as below.

A	B	desired output	Equations	Simplified
0	0	0	$-W_C < 0$	$W_C > 0$
0	1	0	$W_{BC} - W_C < 0$	$W_{BC} < W_C$
1	0	0	$W_{AC} - W_C < 0$	$W_{AC} < W_C$
1	1	1	$W_{AC} + W_{BC} - W_C > 0$	$W_{AC} + W_{BC} > W_C$

We notice the symmetry in W_{BC} and W_{AC}, so we can make a guess that they have the same value:

$W_{BC} = 2$ and $W_{AC} = 2$

Then the inequalities in the table above condense down to the following:

$W_C > 0$ $\quad$ $W_C > 2$ (twice) $\quad$ $W_C < 2+2 = 4$ $\quad$ what I saw earlier

Therefore, $2 < W_C < 4$. Let's make life easy and pick $W_C = 3$. This gives us one acceptable solution:

$W_{BC} = 2$ $\;W_{AC} = 2$ $\;W_C = 3$

Of course, there are many solutions. The following solution also works, because it still obeys the inequalities and the constraints in the table:

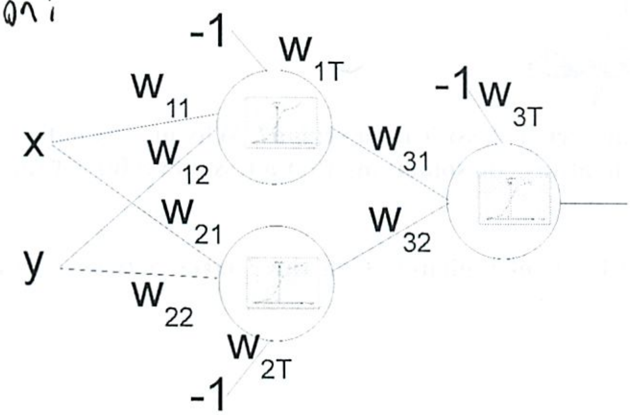
$W_{BC} = 109$ $\;W_{AC} = 109$ $\;W_C = 110$

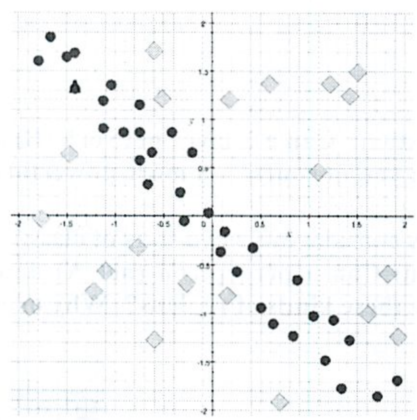
Quizzes will always ask for the *smallest* integer solutions.

This particular problem also illustrates how to combine networks using a logic gate. Thus, to compute more complex regions, we need more neurons either at one level or at the output level. But first, to cement our understanding of this problem, let's look at a related quiz problem, from quiz 3, 2009.

Oh this is what I was working on!

Given this three-node neural network, and the training data on the right





Question: which of the following sets of weights will correctly separate the dots from the diamonds? (Think about what cuts the various weights make at the left neuron....)

Weight set A:

w_{11}	w_{12}	w_{1T}	w_{21}	w_{22}	w_{2T}	w_{21}	w_{22}	w_{2T}
−2	−2	−1	3	3	−1.5	128	128	173

Ok thats how to do fast

Weight set B:

w_{11}	w_{12}	w_{1T}	w_{21}	w_{22}	w_{2T}	w_{21}	w_{22}	w_{2T}
2	−1	1	2	-2	1	100	100	50

Why does weight set A work but not weight set B?

Answer: weight set A defines two negatively sloping lines, similar to the XOR case, which are required to separate the dots from the diamonds.

Weight set B has as its first set of 3 weight a line that has positive slope – this is the wrong slope for that 'cut'. (Same for the next weight set). We need the weights to be both negative or both positive, so that the 'cuts' slope downwards, as required to separate the dot region from the diamonds.

diff so fi

same so —

Example 4. Some other examples of carving up the x-y plane & the associated multi-layer networks

Now let's consider some other patterns in the x-y (or x_1, x_2) plane and what sort of qualitative network might be required to encode them. (This was an exam question in 2008.)

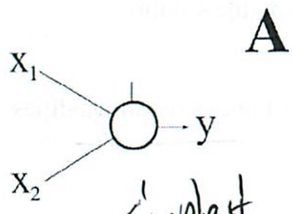
well work it out

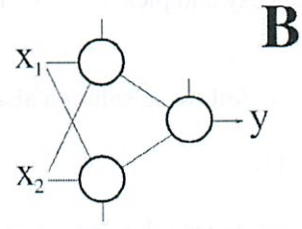
First, let's give the schematic pictures for (i) a perceptron; and then (ii) the simplest 2-layer neural net we have just seen – note that we have removed all the clutter of the w's, etc.:

y= .. x+..

(A) Perceptron:

(B) Simplest 2-layer neural net:

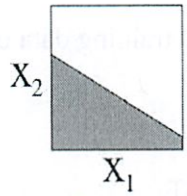




Simplest — 1 cut any direction

— any 2 cuts

Here is a basic picture of the kind of classification regions a perceptron (A) can describe: any single cut, at any angle:



4.1 Question: Can a 2-layer network (B) also describe such a classification region? Why or why not?
Answer: yes, of course – a more powerful network can always do something that a less powerful net can do.

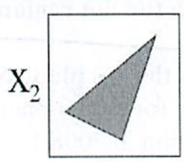
4.2 Now consider these two sorts of classification regions.
Question: Can a perceptron (net A) describe these kinds of regions? Can the 2-layer network (B) also describe these kinds of regions? Why or why not?

Answer: perceptrons can't do these – they require two cuts. A perceptron can only do one. The two-layer network for XOR can be modified with different weights to classify both of these figures above. A <u>simplified</u> two-layer network where unit A is fed just from X_1 and unit B is fed just from X_2 can describe the region on the RIGHT side (because unit A can describe any single vertical cut, X_1= some constant; and unit B can describe any single horizontal cut, X_2= some constant. Then the logic unit C can combine the two regions, as before. (See a picture of this net below, labeled "D".) But this kind of simplified network <u>cannot</u> describe the region on the left, because this requires different two horizontal cuts using the input X_2, so we would need a net with A and B units where the X_2 input connects to <u>both</u> units A and B (the X_1 input is irrelevant and can be set to 0).

one does vertical one does horizontal

— picture missing

4.3 Now let's hone our intuitions by making the region more complex, and by considering different neural networks.
Question: The 2-layer network (B) cannot describe this kind of region. Why not?

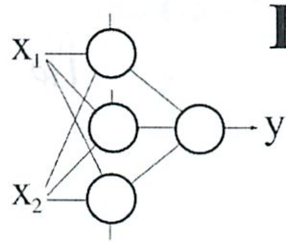


3 cuts

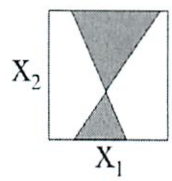
So, we must complicate our neural network to capture this kind of more complex region.
Question: Please explain *why* the following neural network *can* successfully describe the region just above.
(Think about how we classified the region in our worked-out example earlier.)

Answer: this region requires three separate cuts, not just two. So we need three basic input units, and then a second logic unit (as before) to combine their results via AND, like this one:

E



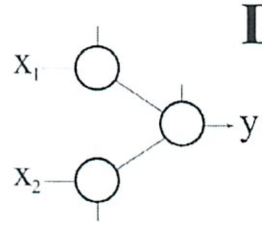
BUT: This network *cannot* successfully describe the region below. Why not? (Think about this, and for the next recitation, try to come up with the reason, and a modification, that is, a more complex neural network, that can describe this region.)



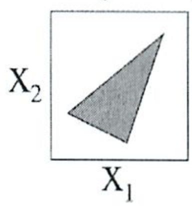
hmm - not sure why not
Since its > < issues

4.4 Finally, let us consider a *simpler* two layer neural network, where the inputs to the top, leftmost hidden neuron receives input *only* from x_1 , and the bottom, leftmost hidden neuron receives inputs *only* from x_2. So the network looks like the following. Can you intuit how this will restrict what regions the network can describe?

D

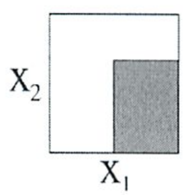


Question: Can this (restricted) neural network classify the region below? Why or why not?



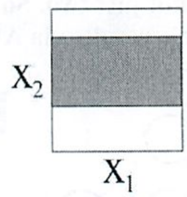
No - 3 cuts

Can network (D) describe this region that we already saw above? Why or why not? (We answered this already above.)



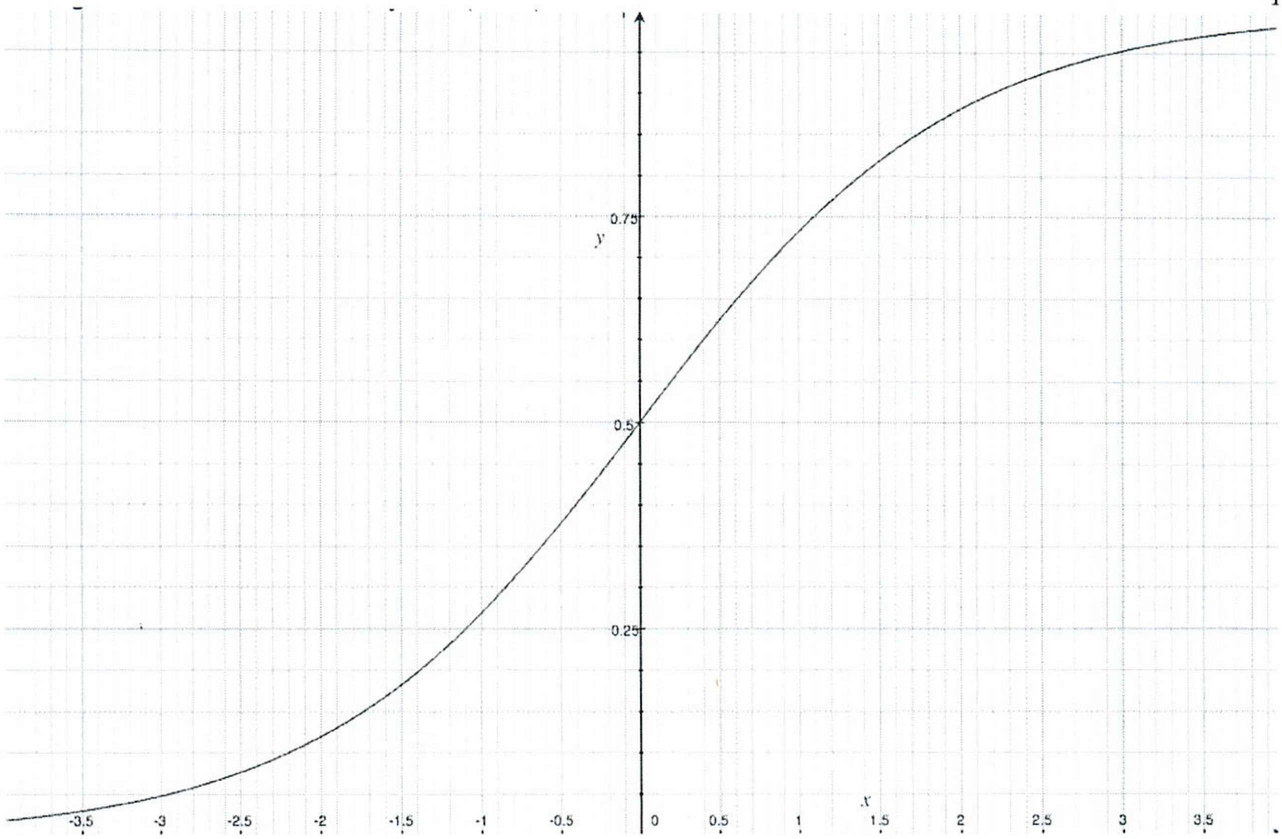
Yes

Finally, for next time, you might want to think about *why* network (D) **cannot** describe this region that we saw before (while we have already discussed what the usual 2-layer network (B) can do in this case) **(We also answered this question already, above).**



No one vert
one horiz only

output



Sigmoid function $y = 1/(1-e^{-x})$

input

Practice 11/15

6.034 Quiz 3
10 November 2010

Name	
email	

Circle your TA and recitation time **(for 1 point)**, so that we can more easily enter your score in our records and return your quiz to you promptly.

TAs
Martin Couturier
Kenny Donahue
Daryl Jones
Gleb Kuznetsov
Kendra Pugh
Mark Seifter
Yuan Shen

Thu	
Time	Instructor
1-2	Bob Berwick
2-3	Bob Berwick
3-4	Bob Berwick

Fri	
Time	Instructor
1-2	Randall Davis
2-3	Randall Davis
3-4	Randall Davis

Problem number	Maximum	Score	Grader
1	50		
2	50		
Total	100		

There are 8 pages in this quiz, including this one, but not including blank pages and tear-off sheets. Tear-off sheets are provided at the end with duplicate drawings and data. As always, open book, open notes, open just about everything, including a calculator, but no computers.

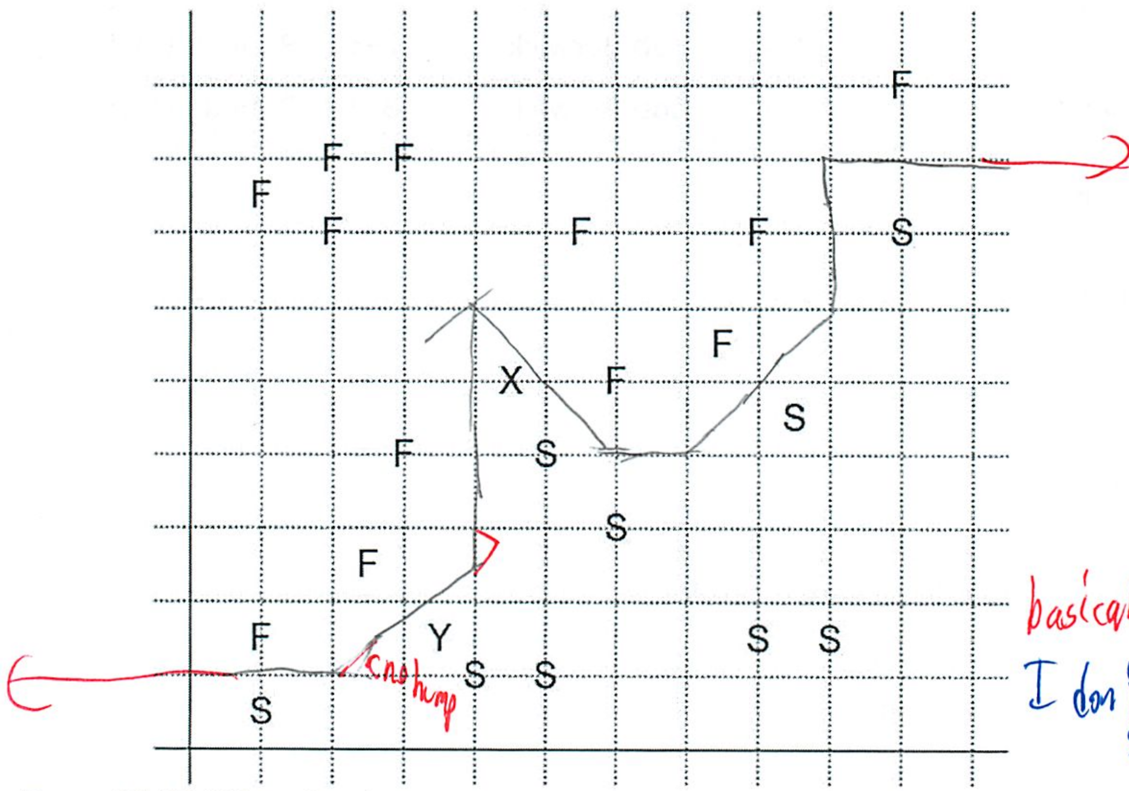
Problem 1: Learning (50 points)

Alice and Bob, a pair of 6.034 students, traveled to DC last weekend for a rally. Over Saturday night, they attended a cocktail party for rally-goers. They knew almost everyone there, but there were a couple of really interesting party crashers.

Alice and Bob decided to use their 6.034 skills to figure out whether the party crashers were at the rally to promote Fear, or restore Sanity.

Part A: Nearest Neighbors (25 points)

During the party, Bob suggests that they look at who the party crashers were spending their time with, given what Bob and Alice know about their friends' reasons to attend the rally (with either Fear/"F" or Sanity/"S").



basically Yes

I don't know how strict they will be

A1: Draw KNN (10 points)

Unknowns

Alice sketches the above drawing on a napkin, indicating the party crashers, X and Y, and the leanings of their friends, indicated by "F" or "S." She then draws nearest neighbor decision boundaries.

On the above graph, draw the decision boundaries produced by k-nearest-neighbors where k=1 and distance measure is Euclidean distance.

A2: More KNN (15 points)

Based on Alice's decision boundaries, **what are the classifications for X and Y?**

X = S ✓

Y = S ✓

Alice changes her mind and decides that those boundaries aren't quite right, and tells Bob they should switch to using k=3. "Why? That's so hard to draw!" Says Bob. "I think k=1 boundaries are too specific," says Alice. **What's the name for the problem with k=1 decision boundaries?** name for - I misinterpted

I would review in notes

Overfitting

She decides to classify the party goers using k= 3. **If k=3, what are the classifications for X and Y?**

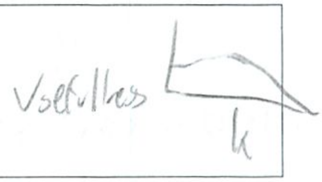
X = F ✓

Y = S ✓

"Okay okay! Based on what you just said, how about k=21?" Alice says "I don't think that'd be a good classifier either." **What's the problem Alice has with k=21?**

No - considers all pts - useless

So good up to a point then drops

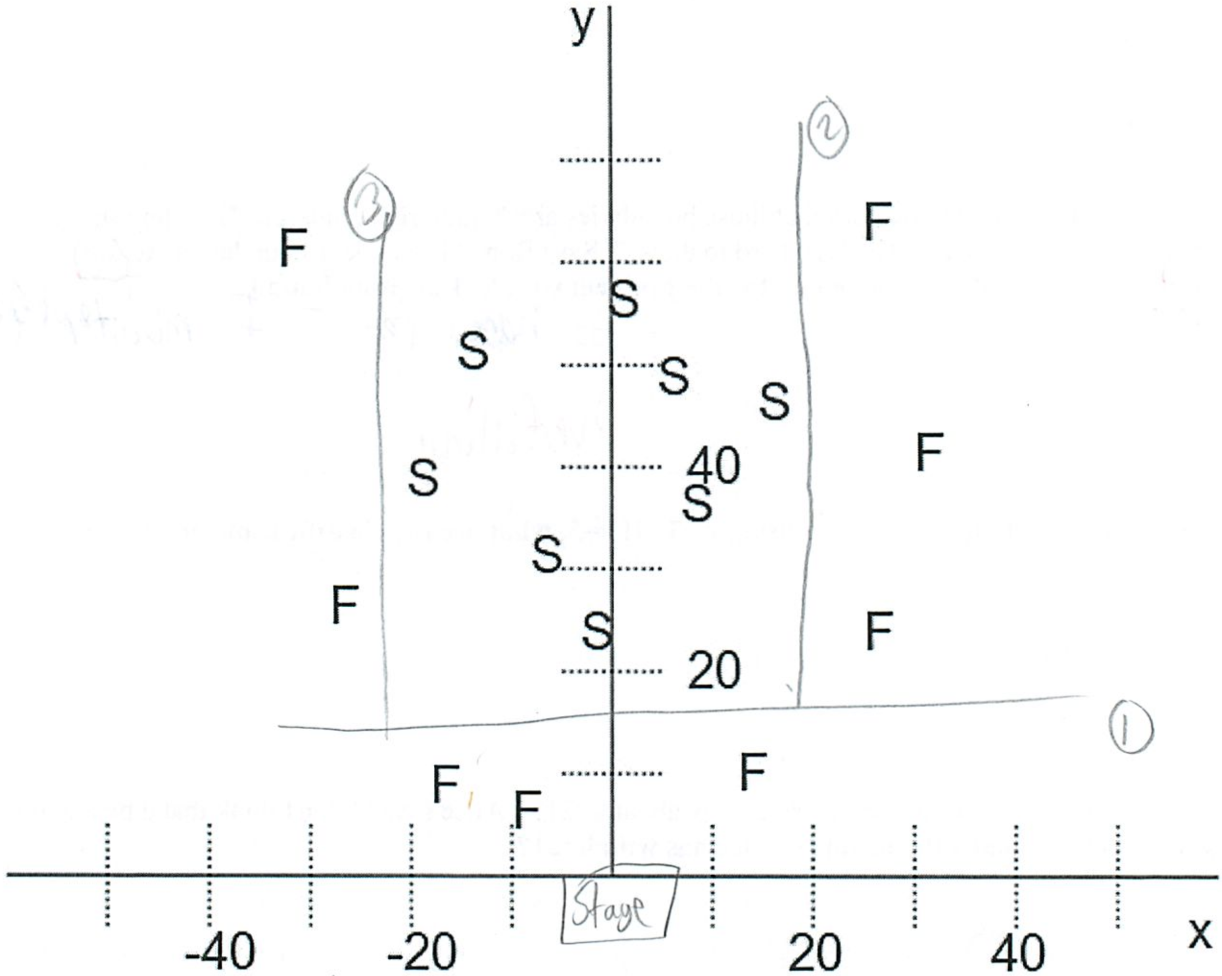


Part B: ID Trees (25 points)

Alice and Bob give up on classifying the party goers using who they stand near at the party. "Why not look at where they stood during the event?" says Bob. He then pulls up a high-resolution satellite image of the event on his smart phone, zooms and enhances, picks out his and Alice's friends, and sketches all their relative positions on a separate napkin.

Here's the picture he gets. He and Alice argue about the distance their friends were spread out over the event, so he puts in distance from the stage, as well as spread from the center of the mall:

NOTE: lowercase x and y are axes, measuring distance from the stage(y) and the center of the mall(x). There are 16 friends total.



B1: ID Trees (15 points)

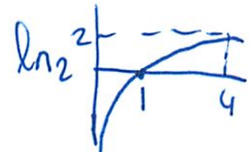
So most even split

Using the greedy heuristic, determine the decision boundaries Bob draws for ID trees. **Ties are broken by: vertical lines before horizontal lines, lesser values before greater values. Draw them on the picture, above, and write the equations in the box below. The numbers in your equations need only be approximate values; we know you cannot produce exact values from the diagram.**

I mixed up vertical + horizontal (facepalm)

Y = 15
X = -25
X = 20

Other than sure



So $\ln_2(1) = 0$

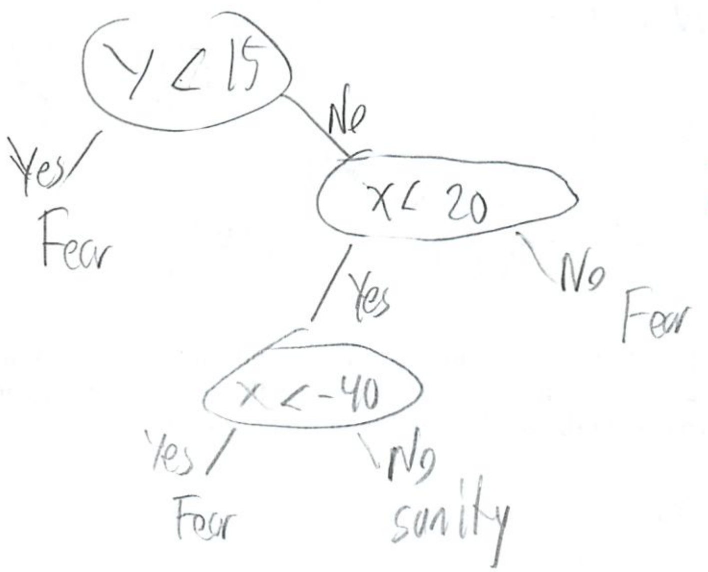
What's the disorder associated with the **first** of the decision boundaries? You may express your answer in terms of logarithms.

0

forgot − sign

$$\frac{3}{16}\cancel{\left(\frac{3}{3}\ln\left(\frac{3}{3}\right)\right)} + \frac{13}{16}\left(-\frac{5}{13}\ln\left(\frac{5}{13}\right) + \frac{8}{13}\ln\left(\frac{8}{13}\right)\right)$$

Draw the resulting decision tree in the space below. Order your branches such that the less-than-threshold branch is left of the other branch.



If I would have got points

B2: A Better Way (10 points)

Alice suggests that if they change their representation of the data, she and Bob may have an easier time creating decision boundaries. **Briefly describe how you would change how this data is represented.**

?, abs value x   polar w/ (0, 40) translated as origin

I guess I didn't see
But it's usually always polar

Based on your representation, what are the new decision boundaries and associated disorder?

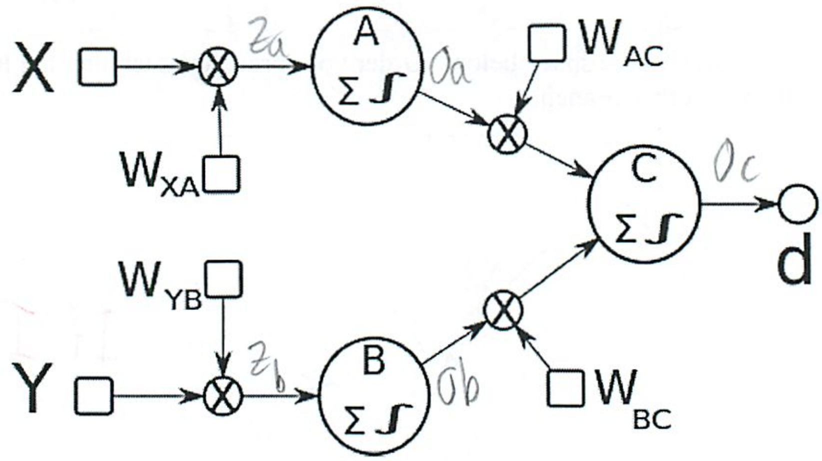
$y = 15$
$|x| = 21$ same disorder

$r \simeq 20$ disorder $= 0$

Problem 2: Neural Nets (50 points)

Part A: Warmup (25 points)

For the network below, answer the following questions:



been easy so far

A1: Simulate Forward Propagation. (15 points) Compute and fill in the values in the table below. Leave numerical answers to 2-decimal precision. You may use the sigmoid table to help with your calculations. **Note that there are no threshold weights in this network.**

tables would have been nice

X	Y	W_{XA}	z_A	o_A	W_{YB}	z_B	o_B
60	70	1	60	1.0	-1	-70	0

? what is? before sigmoid

		W_{AC}	W_{BC}	z_C	o_C	d	d - o_C
		1	0	1.0	.73	1	.27

Table of relevant values of the sigmoid function

x	s(x)	x	s(x)
< -50	0.00	0	0.5
-10	4.5×10^{-5}	1	0.73
-5	0.01	2	0.88
-3	0.05	3	0.95
-2	0.12	5	0.99
-1	0.27	> 50	1.00

A2: Back Propagation: (10 points)

Compute numerical values for weight updates for back propagation. Write out the full expressions you are calculating for partial credit. Assume that the learning rate $\alpha = 1$.

ΔW_{AC}

$= \alpha \cdot \delta_f \cdot \text{input}$

$= 1 \underbrace{(.73)(1-.73)(.27)} \cdot 1.0$ ✓

Should actually do $\approx .0532$

ΔW_{XA}

$= \alpha \cdot o(1-o)\, w_c \cdot \delta_c$

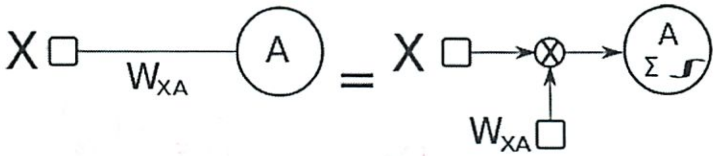
$= 1 \cdot (1.0)(1-1)(1)(.73)(1-.73)(.27)$

duh is 1, I was thinking — but didn't write it!

$= 0$

Part B: Multi-class Output (25 points)

NOTE: For Networks from this point on, we will adopt the abbreviated network notation. ??



One possible method for making neural nets capable of multi-class classification is to change the sigmoid function. Inspired by the 6.034 GPA function, Yuan decides to adopt a 2-step sigmoid function as the output of the sigmoid unit, creating neural nets that can output roughly 3 values, 0, 1, and 2. The 2-step sigmoid $S_2(x)$ has the following equation.

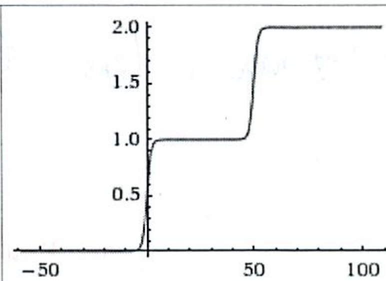
New Output function:

$$s_2(x)=\frac{1}{1+e^{-x}}+\frac{1}{1+e^{-(x-k)}}=s(x)+s(x-k)$$

what is this? old sigmoid

New performance Function:

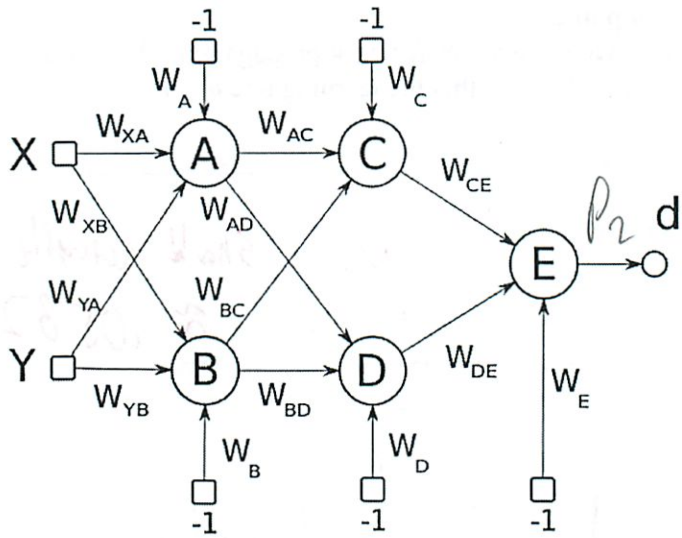
$$P_2(o)=-\frac{1}{2}\left(\frac{d-o}{2}\right)^2$$



(x from −50 to 100)

Wolfram alpha

For instance, when k = 50, the sigmoid $S_2(x)$ would have the graph shown at the right. Thus, the output is roughly, 0 when x is < 0, 1 when 0 < x < 50, and 2 when x > 50. Changing the sigmoid function triggers a similar change in the performance function in order to normalize range of values of the error.



the hard part of exam

B1. (9 points) For the neural network, given above, where the sigmoid units use S_2 and the performance function is P_2, write out the equation for $\delta_E = \frac{\partial P_2}{\partial o_E}\frac{\partial o_E}{\partial z_E}$. Express your answer in terms of d, k, s(), and z_E (the sum of weights times the inputs at node E). Hint: $o_E = s(z_E) + s(z_E\text{-}k)$.

deriv perf fcn

more a qu in taking derivs...

$\frac{\partial P_2}{\partial o_E}$ = deriv of performance $= \frac{d}{do_E} -\frac{1}{2}\left(\frac{d-o}{2}\right)^2$

$-2 \cdot \frac{1}{2}\left(\frac{d-o}{2}\right) \cdot -1$

$\frac{d-o}{2}$

$\frac{\partial o_E}{\partial z_e}$ = lot take deriv

use right variables

$\delta_E = [s(z_e)(1-s(z_e)) + s(z_e-k)(1-s(z_e-k))] \cdot \frac{d-(s(z_e)+s(z_e-k))}{4}$

B2. (8 points) Write out the equation for δ_C. Express you answer in terms of d, k, s(), z_c, and any weights in the network or any answer you've computed before.

$\delta_c = s(z_c)(1-s(z_c)) + s(z_c-k)(1-s(z_c-k))] \cdot W_{CE} \cdot \delta_F$

? So they have in terms of this

B3. (8 points) List all the weights that would be used in the **fully expanded calculation** for W_B' (the new value of weight W_B).

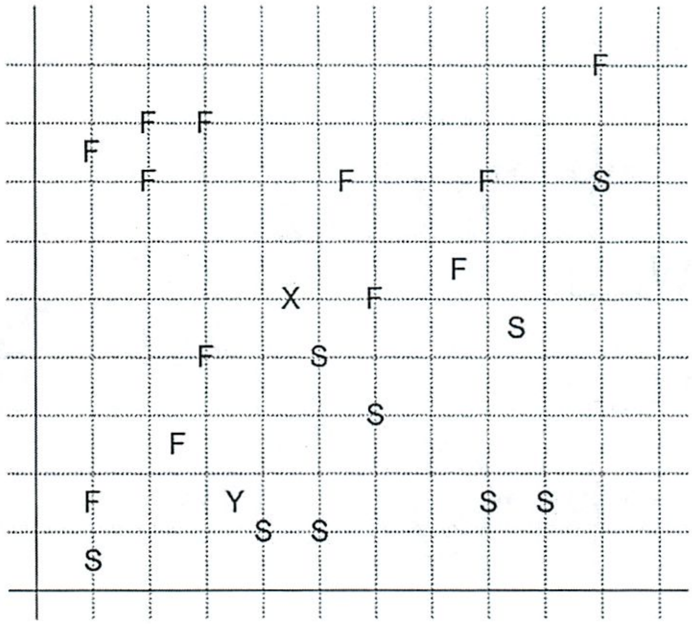
~~WE~~ WBC ✓ ~~WDE~~ - have already
WCE ✓ WBD ✓ WD
WDE ✓ WB ✓ ~~WAD~~

Oh duh

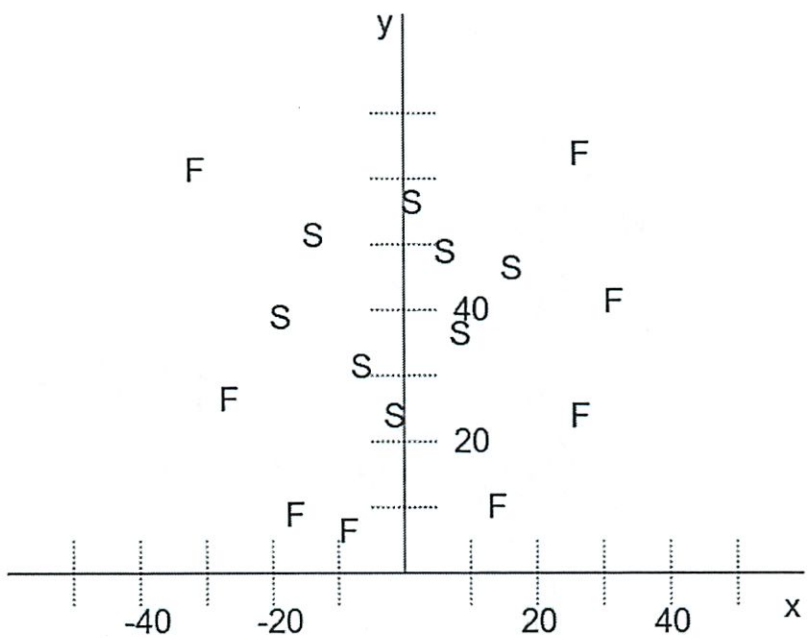
This is just a math complex ~~...~~ problem

Tear off sheet, you need not hand this in.

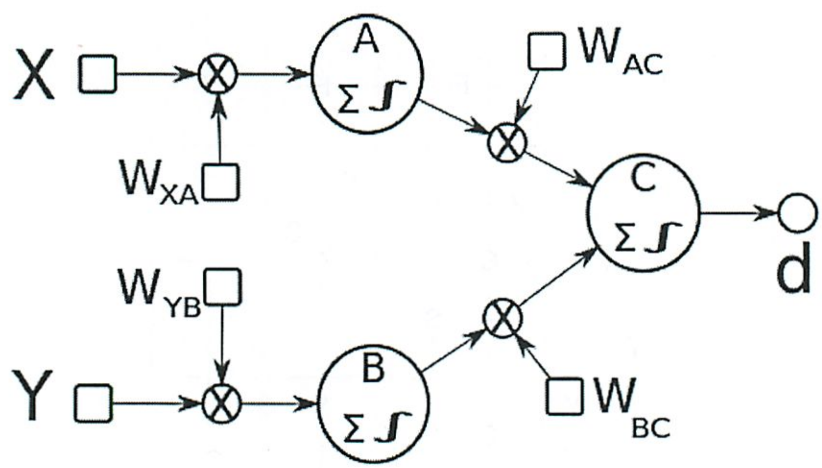
1A



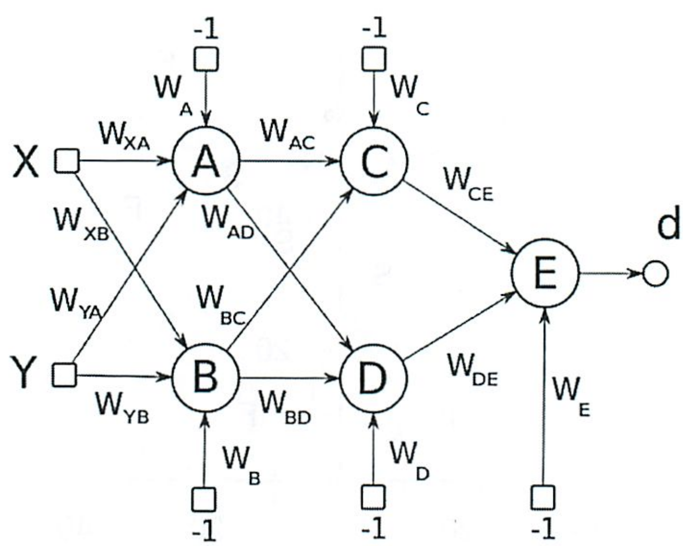
1B



2A


X
Y
A
B
C
W_{XA}
W_{YB}
W_{AC}
W_{BC}
d

2B


-1
-1
W_A
W_C
X
Y
A
B
C
D
E
W_{XA}
W_{AC}
W_{CE}
W_{XB}
W_{AD}
W_{YA}
W_{BC}
W_{YB}
W_{BD}
W_{DE}
W_E
W_B
W_D
d
-1
-1
-1

6.034 Quiz 3
10 November 2010

Name	Stephen Colbert
email	

Circle your TA and recitation time **(for 1 point)**, so that we can more easily enter your score in our records and return your quiz to you promptly.

TAs
Martin Couturier
Kenny Donahue
Daryl Jones
Gleb Kuznetsov
Kendra Pugh
Mark Seifter
Yuan Shen

Thu	
Time	Instructor
1-2	Bob Berwick
2-3	Bob Berwick
3-4	Bob Berwick

Fri	
Time	Instructor
1-2	Randall Davis
2-3	Randall Davis
3-4	Randall Davis

Problem number	Maximum	Score	Grader
1	50		
2	50		
Total	100		

There are 8 pages in this quiz, including this one, but not including blank pages and tear-off sheets. Tear-off sheets are provided at the end with duplicate drawings and data. As always, open book, open notes, open just about everything, including a calculator, but no computers.

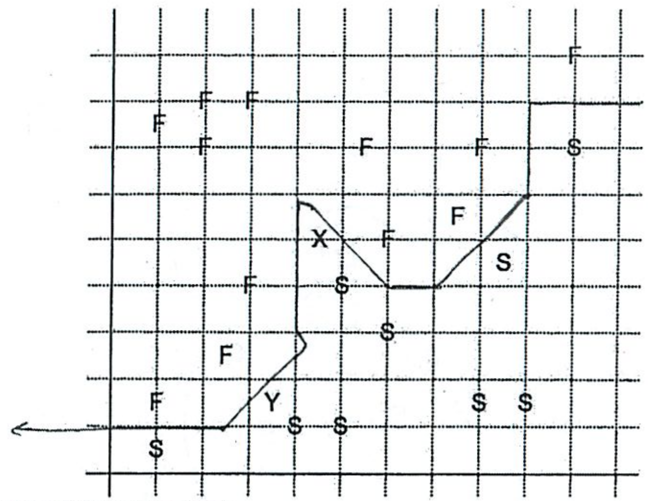
Problem 1: Learning (50 points)

Alice and Bob, a pair of 6.034 students, traveled to DC last weekend for a rally. Over Saturday night, they attended a cocktail party for rally-goers. They knew almost everyone there, but there were a couple of really interesting party crashers.

Alice and Bob decided to use their 6.034 skills to figure out whether the party crashers were at the rally to promote Fear, or restore Sanity.

Part A: Nearest Neighbors (25 points)

During the party, Bob suggests that they look at who the party crashers were spending their time with, given what Bob and Alice know about their friends' reasons to attend the rally (with either Fear/"F" or Sanity/"S").



A1: Draw KNN (10 points)

Alice sketches the above drawing on a napkin, indicating the party crashers, X and Y, and the leanings of their friends, indicated by "F" or "S." She then draws nearest neighbor decision boundaries.

On the above graph, draw the decision boundaries produced by k-nearest-neighbors where k=1 and distance measure is Euclidean distance.

A2: More KNN (15 points)

Based on Alice's decision boundaries, **what are the classifications for X and Y?**

X = S
Y = S

Alice changes her mind and decides that those boundaries aren't quite right, and tells Bob they should switch to using k=3. "Why? That's so hard to draw!" Says Bob. "I think k=1 boundaries are too specific," says Alice. **What's the name for the problem with k=1 decision boundaries?**

OVERFITTING

She decides to classify the party goers using k= 3. **If k=3, what are the classifications for X and Y?**

X = F
Y = S

"Okay okay! Based on what you just said, how about k=21?" Alice says "I don't think that'd be a good classifier either." **What's the problem Alice has with k=21?**

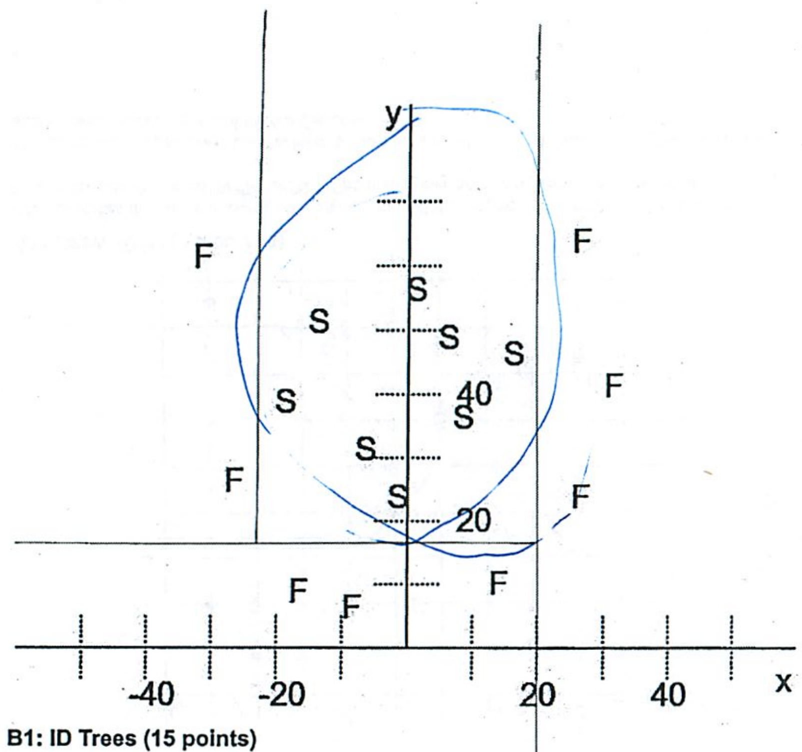
Underfitting, mode of. your training set

Part B: ID Trees (25 points)

Alice and Bob give up on classifying the party goers using who they stand near at the party. "Why not look at where they stood during the event?" says Bob. He then pulls up a high-resolution satellite image of the event on his smart phone, zooms and enhances, picks out his and Alice's friends, and sketches all their relative positions on a separate napkin.

Here's the picture he gets. He and Alice argue about the distance their friends were spread out over the event, so he puts in distance from the stage, as well as spread from the center of the mall:

NOTE: lowercase x and y are axes, measuring distance from the stage(y) and the center of the mall(x). There are 16 friends total.



B1: ID Trees (15 points)

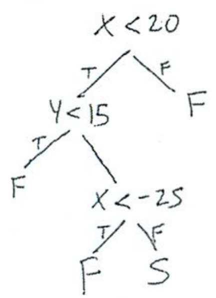
Using the greedy heuristic, determine the decision boundaries Bob draws for ID trees. **Ties are broken by: vertical lines before horizontal lines, lesser values before greater values. Draw them on the picture, above, and write the equations in the box below. The numbers in your equations need only be approximate values; we know you cannot produce exact values from the diagram.**

x=20 17 to 23
y=17 12 to 21
x=-22 -19 to -25

What's the disorder associated with the **first** of the decision boundaries? You may express your answer in terms of logarithms.

$\frac{13}{16}\left(-\frac{5}{13}\lg\frac{5}{13}-\frac{8}{13}\lg\frac{8}{13}\right)$

Draw the resulting decision tree in the space below. Order your branches such that the less-than-threshold branch is left of the other branch.



B2: A Better Way (10 points)

Alice suggests that if they change their representation of the data, she and Bob may have an easier time creating decision boundaries. **Briefly describe how you would change how this data is represented.**

Translate (0,40) to origin
Convert to polar

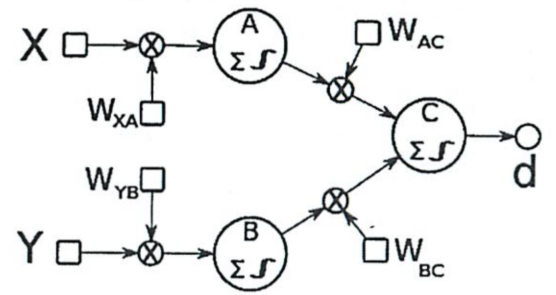
Based on your representation, what are the new decision boundaries and associated disorder?

$r \simeq 20$; disorder = 0

Problem 2: Neural Nets (50 points)

Part A: Warmup (25 points)

For the network below, answer the following questions:



A1: Simulate Forward Propagation. (15 points) Compute and fill in the values in the table below. Leave numerical answers to 2-decimal precision. You may use the sigmoid table to help with your calculations. **Note that there are no threshold weights in this network.**

X	Y	W_{XA}	z_A	o_A	W_{YB}	z_B	o_B
60	70	1	60	1.00	-1	−70	0.00

	W_{AC}	W_{BC}	z_C	o_C	d	d - o_C
	1	0	1.00	0.73	1	0.27

Table of relevant values of the sigmoid function

x	s(x)	x	s(x)
<-50	0.00	0	0.5
-10	4.5×10^{-5}	1	0.73
-5	0.01	2	0.88
-3	0.05	3	0.95
-2	0.12	5	0.99
-1	0.27	> 50	1.00

A2: Back Propagation: (10 points)
Compute numerical values for weight updates for back propagation. Write out the full expressions you are calculating for partial credit. Assume that the learning rate $\alpha = 1$.

ΔW_{AC}

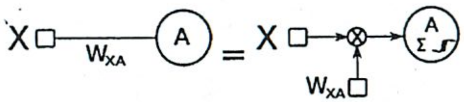
$$\Delta W_{AC} = \alpha \cdot \delta_c \cdot o_c = 1 \cdot \overbrace{(0.27)}^{d-o_c}\overbrace{(0.73 \cdot (1-0.73))}^{o_c(1-o_c)} \cdot 1.00$$
$$\boxed{= (0.27)^2 \cdot 0.73 \cong 0.0532}$$

ΔW_{XA}

$$\Delta W_{XA} = \alpha \cdot \delta_A \cdot o_c = 1 \cdot (o_A(1-o_A) \cdot W_{AC} \cdot \delta_c) \cdot x$$
$$= 1 \cdot 1 \cdot (1-1)^0 \cdot 1 \cdot (0.0532) \cdot 60$$
$$\boxed{= 0}$$

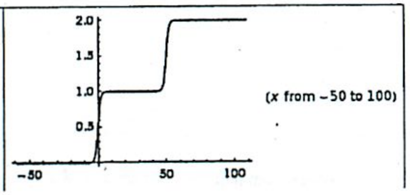
Part B: Multi-class Output (25 points)

NOTE: For Networks from this point on, we will adopt the abbreviated network notation.

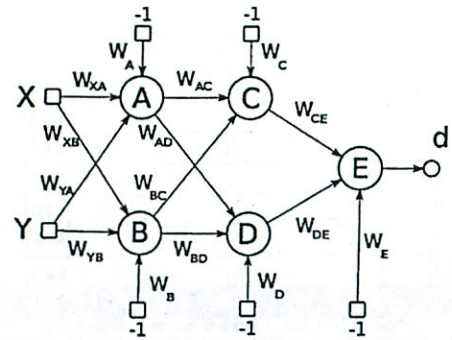


One possible method for making neural nets capable of multi-class classification is to change the sigmoid function. Inspired by the 6.034 GPA function, Yuan decides to adopt a 2-step sigmoid function as the output of the sigmoid unit, creating neural nets that can output roughly 3 values, 0, 1, and 2. The 2-step sigmoid $S_2(x)$ has the following equation.

New Output function:
$$s_2(x) = \frac{1}{1+e^{-x}} + \frac{1}{1+e^{-(x-k)}} = s(x) + s(x-k)$$
New performance Function:
$$P_2(o) = -\frac{1}{2}\left(\frac{d-o}{2}\right)^2$$



For instance, when $k = 50$, the sigmoid $S_2(x)$ would have the graph shown at the right. Thus, the output is roughly, 0 when x is < 0, 1 when 0 < x < 50, and 2 when x > 50. Changing the sigmoid function triggers a similar change in the performance function in order to normalize range of values of the error.



B1. (9 points) For the neural network, given above, where the sigmoid units use S_2 and the performance function is P_2, write out the equation for $\delta_E = \frac{\partial P_2}{\partial o_E}\frac{\partial o_E}{\partial z_E}$. Express your answer in terms of d, k, s(), and z_E (the sum of weights times the inputs at node E). Hint: $o_E = s(z_E) + s(z_E\text{-}k)$.

$$\delta_E = \left[s(z_E)(1-s(z_E)) + s(z_E-k)(1-s(z_E-k))\right] \cdot \frac{d-(s(z_E)+s(z_E-k))}{4}$$

5 pts 4 pts.

B2. (8 points) Write out the equation for δ_C. Express you answer in terms of d, k, s(), z_c, and any weights in the network or any answer you've computed before.

$$\delta_c = \left[s(z_c)(1-s(z_c)) + s(z_c-k)(1-s(z_c-k))\right] \cdot W_{CE} \cdot \delta_E$$

4pts 4pts.

B3. (8 points) List all the weights that would be used in the fully expanded calculation for W_B' (the new value of weight W_B).

① W_B, W_{BD}, W_{BC}, W_{CE}, W_{DE} if all o_Ns are already computed

② Alternately saying all weights is also okay if you account for Z calculations.

Michael E Plasmeier

From:	6034_t5_f11-bounces@MIT.EDU on behalf of Erek Speed <espeed@MIT.EDU>
Sent:	Tuesday, November 15, 2011 11:43 PM
To:	6034_t13_f11@mit.edu; 6034_t5_f11@mit.edu; 6034_t2_f11@mit.edu
Subject:	Re: [6034_t5_f11] Notes on Neural Nets

I've gotten several questions about 2006 q3 today. Especially when it comes to choosing w_d and w_{cd}. A lot of my answers have come down to relying on intuition and some heuristics I have but in my last write up I give a bit more direction. If it's confusing at all and you already had good grasp of this type of question ignore the following.

****

Right now we have equations for our 3 lines and we've solved all of their weights.
A: $-x_1 - 2x_2 + 2 = 0$
B: $x_2 - 1 = 0$
C: $-x_2 + 2 = 0$

These equations come from the weights we already found which is why they aren't 'simplified.'

These lines divide up our graph into 4 regions. For each region we want D to output the correct thing. Remember that we are dealing with a threshold and not a sigmoid so we know that any input to a node > 0 will be 1 and 0 otherwise.

Because the problem insists that that L regions should be 1 and P regions 0 this means for ever L region the inputs to D need to sum to greater than 0. In general it looks like this:
$o_a * w_{ad} + o_b * w_{bd} + o_c * w_{cd} - w_d > 0$

For P regions this should be <= to 0. (the <= is from the problem statement.)

For each region, some of the outputs will be 1 and some will be 0 (or all or none) which means each region will provide a inequality. After examining every region you will get 4 inequalities which restrict w_d and w_{cd}.

Before we can do this, we must decide when each node will be 1 and when it will be 0. In the past, I used some hand waving to do this and not everyone got the intuition. If you did, great! If not, read on.

For each line, test a point below or above it. For instance (0,0) is pretty easy usually. For your tests point, if the value is negative then that node is 0 for ALL regions on the same side of the line as your test point. It is 1 otherwise. (Due to the threshold function any point will work because we don't have to worry about the sigmoid taking time to go from 0 to 1.)

For example, let's plug in 0,0 for this problem:
A: 0+0+2 => 2 which is greater than 0 so A will be 1 for the bottom region and 0 for the above regions.
B: 0 + -1 => -1 which is less than 0 so B will be 0 for the two regions below it and 1 for the two regions above it.
C: 0 + 2 => 2 which is greater than 0 so C will be 1 for the 3 regions below it and 0 for the region above it.

nice

thats it!

Now we just need to use this information to find an inequality for each region.

Bottom Corner Region:
A is 1, B is 0, C is 1, we want the output to be 1.
Using the inequality from above:
$1*4 + 0*2 + 1*w_{cd} - w_d > 0$

simplifies to:
4 + w_cd -w_d > 0

The other regions follow a similar path.

The final step is to use your 4 inequalities to determine what w_d and w_cd must be.

Erek

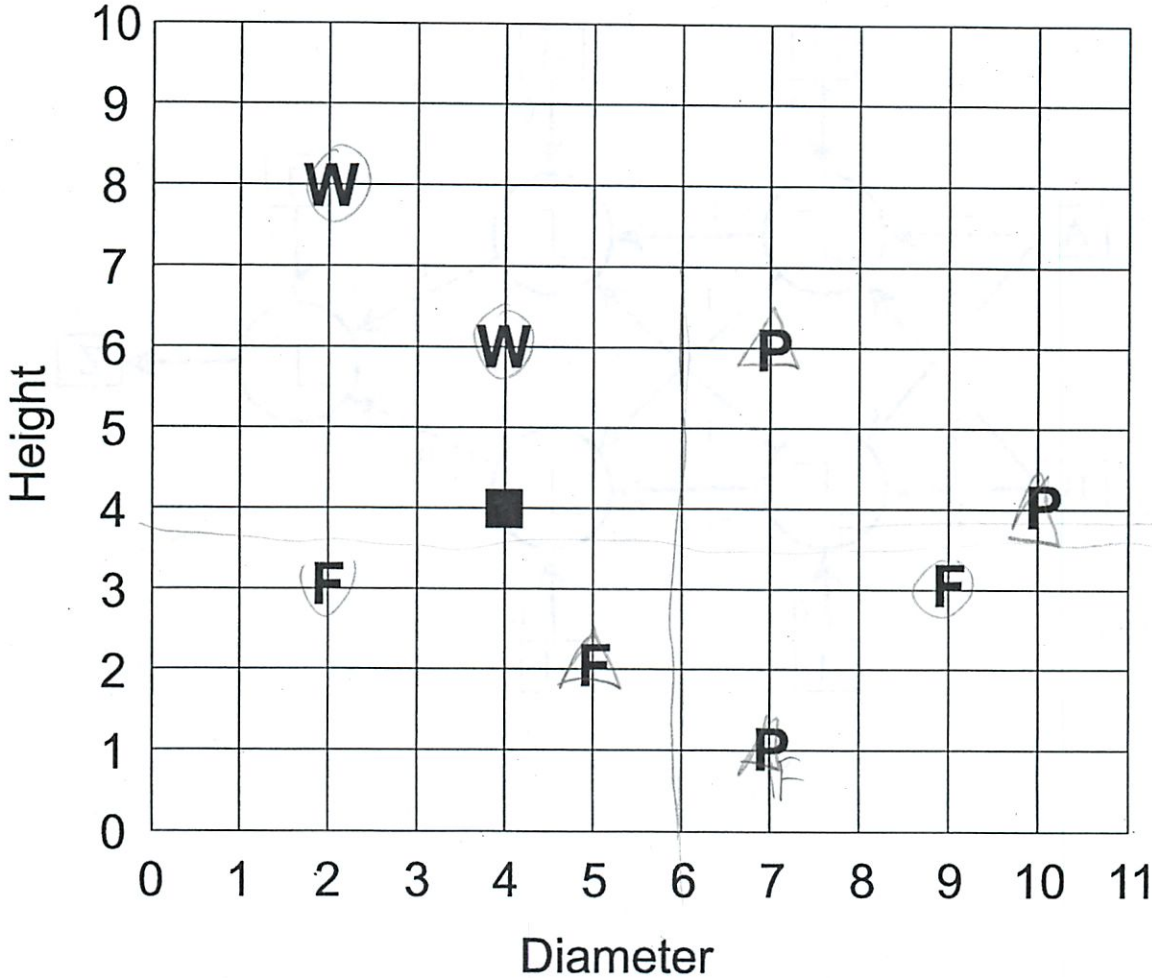
2011/11/15 Erek Speed <espeed@mit.edu>:
> I'm gone but ask via email etc.
>
> On Nov 15, 2011 5:01 PM, "Erek Speed" <espeed@mit.edu> wrote:
>>
>> I decided to extend my office hours to the future. I'll be here for
>> several hours. 24-323.
>>
>> If you're unsure of whether I'm still here email or text me.
>>
>> 2011/11/14 Erek Speed <espeed@mit.edu>:
>> > Hi,
>> >
>> > These are the notes on neural nets which I mentioned (will mention)
>> > in tutorial should be absorbed:
>> > http://web.mit.edu/6.034/wwwbob/recitation8-fall11.pdf
>> >
>> > I think this test will be really hard so I want to help you guys as
>> > much as possible.
>> >
>> > Always you can email me to set up a time to meet during the day.
>> > In general, I prefer to be off campus at night but if it's
>> > necessary I will make exceptions.
>> >
>> > I can do group office hours but I think they're only useful if
>> > everybody has similar questions and they're like an extra tutorial.
>> > Feel free to use these lists to organize amongst yourselves.
>> >
>> > If you contact me via email I'll probably reply pretty fast. You
>> > can even send it to the TAs list and either I'll reply or someone
>> > will if I'm feeling slow.
>> >
>> > You can find me on gtalk at melink14@gmail.com. You can even
>> > text/call me at 785-546-0123. At 3AM if you want.
>> >
>> > I could even setup a google hangout and do problems in gimp or
>> > something with a tablet.
>> >
>> > The bottom line is this: There are some pretty hard tests in the
>> > archive. This test is looking to be as hard as any of them and you
>> > will need to know all the concepts really well. Study now. Take
>> > tests. Time them. If anything is confusing ask me or another TA.
>> >

>> >
>> > That's all.
>> >
>> > Erek
>> >
>

6034_t5_f11 mailing list
6034_t5_f11@mit.edu
http://mailman.mit.edu/mailman/listinfo/6034_t5_f11

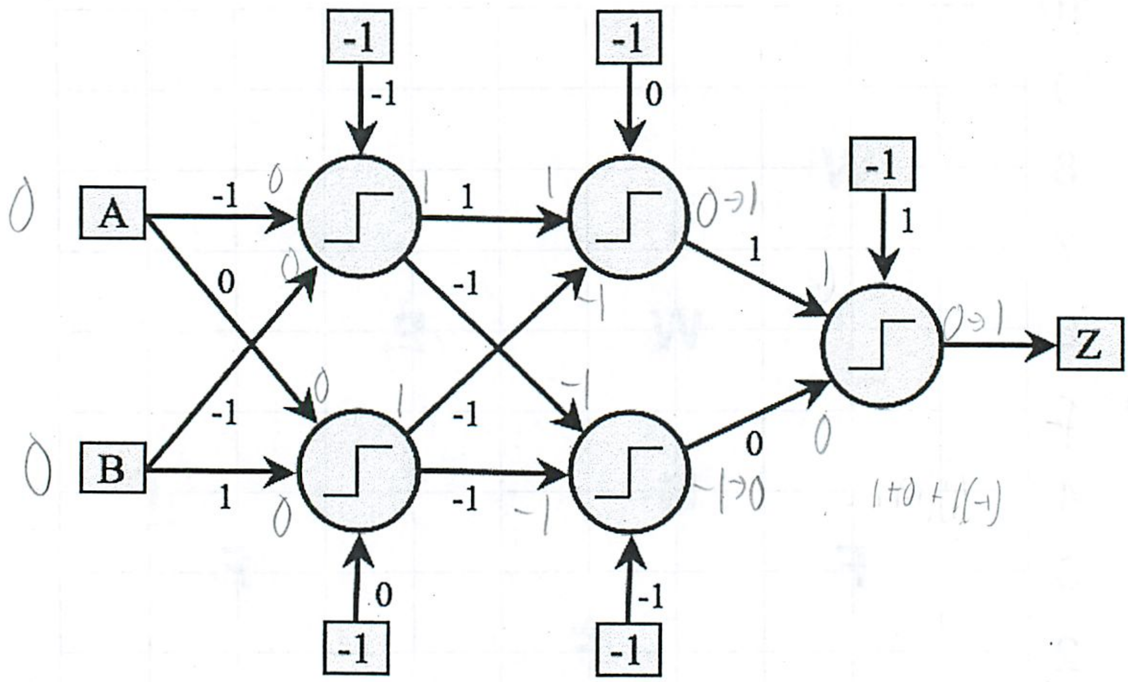
Tear Off Sheet

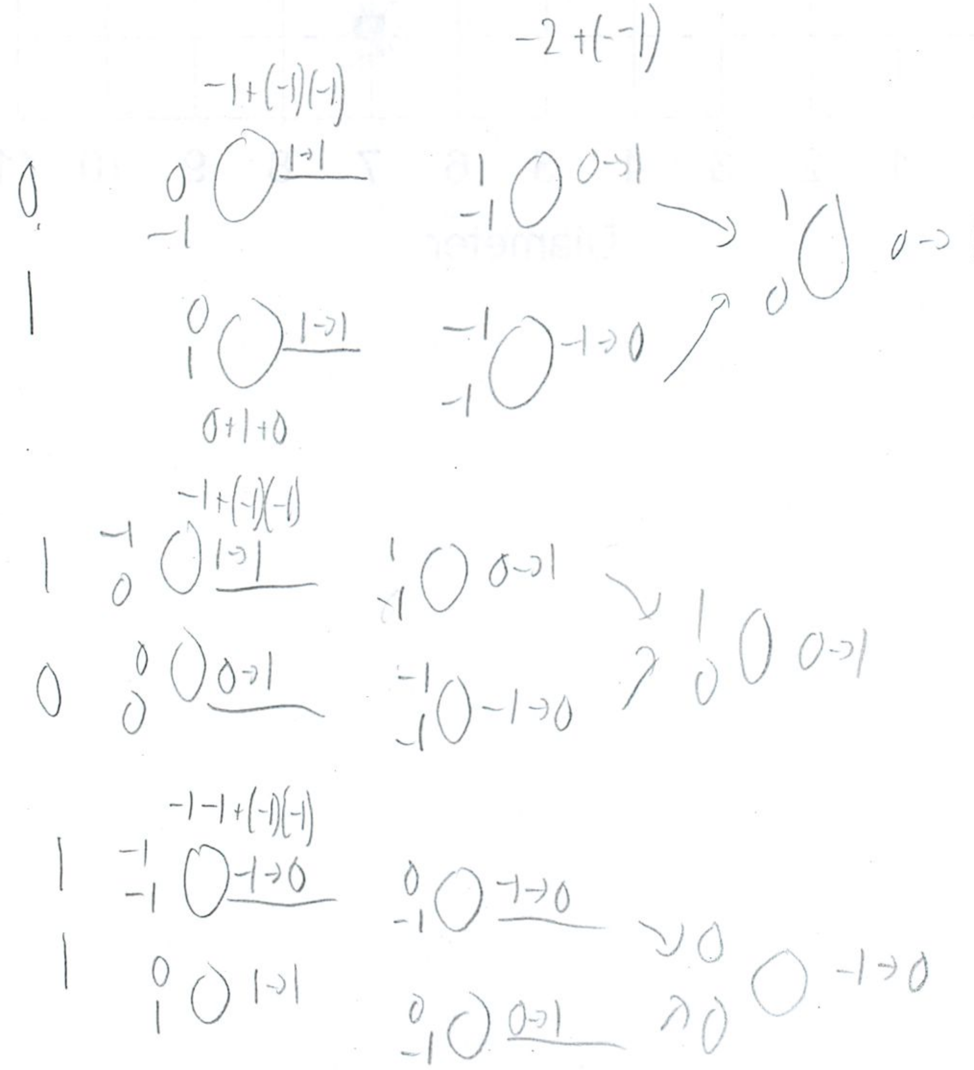
Figure from problem 1, part A1:



Tear Off Sheet

Figure from problem 2, part A: $W_A A + W_B B + W_T = \text{out}$

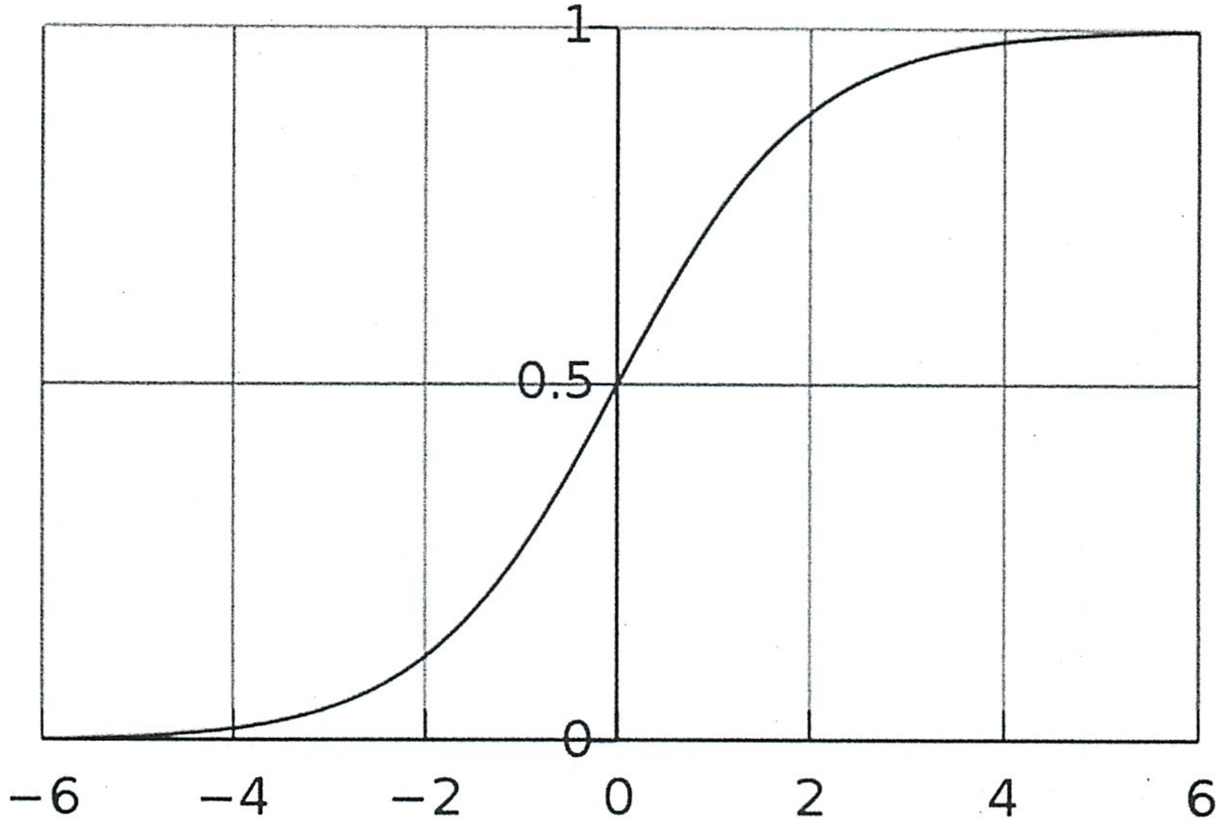




Tear Off Sheet

Pertinent to problem 2, part B:

Graph of the output (vertical axis) of the sigmoidal logistic transfer function versus its input. At an input of 0, this transfer function outputs a value of 0.5.



Tear Off Sheet – Blank Page

6.034 Quiz 3
November 16, 2011

Name	Michael Plasmeier
Email	theplaz @ mit.edu

Circle your TA *and* recitation (for 1 extra credit point), so that we can more easily enter your score in our records and return your quiz to you promptly.

TAs	Recitations
Avril Kenney	Thu. 1-2, Bob Berwick
Adam Mustafa	Thu. 2-3, Bob Berwick
Caryn Krakauer	Thu. 3-4, Bob Berwick
Erek Speed	Fri. 1-2, Randall Davis
Gary Planthaber	Fri. 2-3, Randall Davis
Peter Brin	Fri. 3-4, Randall Davis
Tanya Kortz	

Problem	Maximum	Score	Grader
Extra Credit	1	+1	TMK
1	40	27	TMK
2	40	38	AFK
3	20	4	ES
Total	101	70	ES

= 4

There are a total of 10 pages in this quiz not including one or more tear off sheets that may be provided at the end with duplicate drawings and data. As always, open book, open notes, open just about everything, including a calculator, but no computers.

Problem 1: Nearest Neighbors and ID Trees (40 points)

You move into a new house, and discover that the garden is overgrown with all kinds of plants. You decide to figure out what they all are, and then deal with them accordingly. Fortunately, you know some information about the characteristics of other types of plants to compare them with:

Classification	Diameter	Height	Leaf shape
food	2	3	round
food	9	3	round
food	5	2	pointy
weed	2	8	round
weed	4	6	round
psychoactive	7	1	pointy
psychoactive	7	6	pointy
psychoactive	10	4	pointy

Part A: Nearest Neighbors (16 points) +13

First, you decide to use nearest-neighbor classification to categorize your plants. For this part, you will **only use the continuous features (height and diameter)**, ignoring the binary feature (leaf shape).

A1 (12 points) +9

The following graph shows the known data points in a two-dimensional space of height and diameter. Draw the decision boundaries produced by nearest-neighbor classification (1 nearest neighbor). Ignore the unlabeled (square) point.

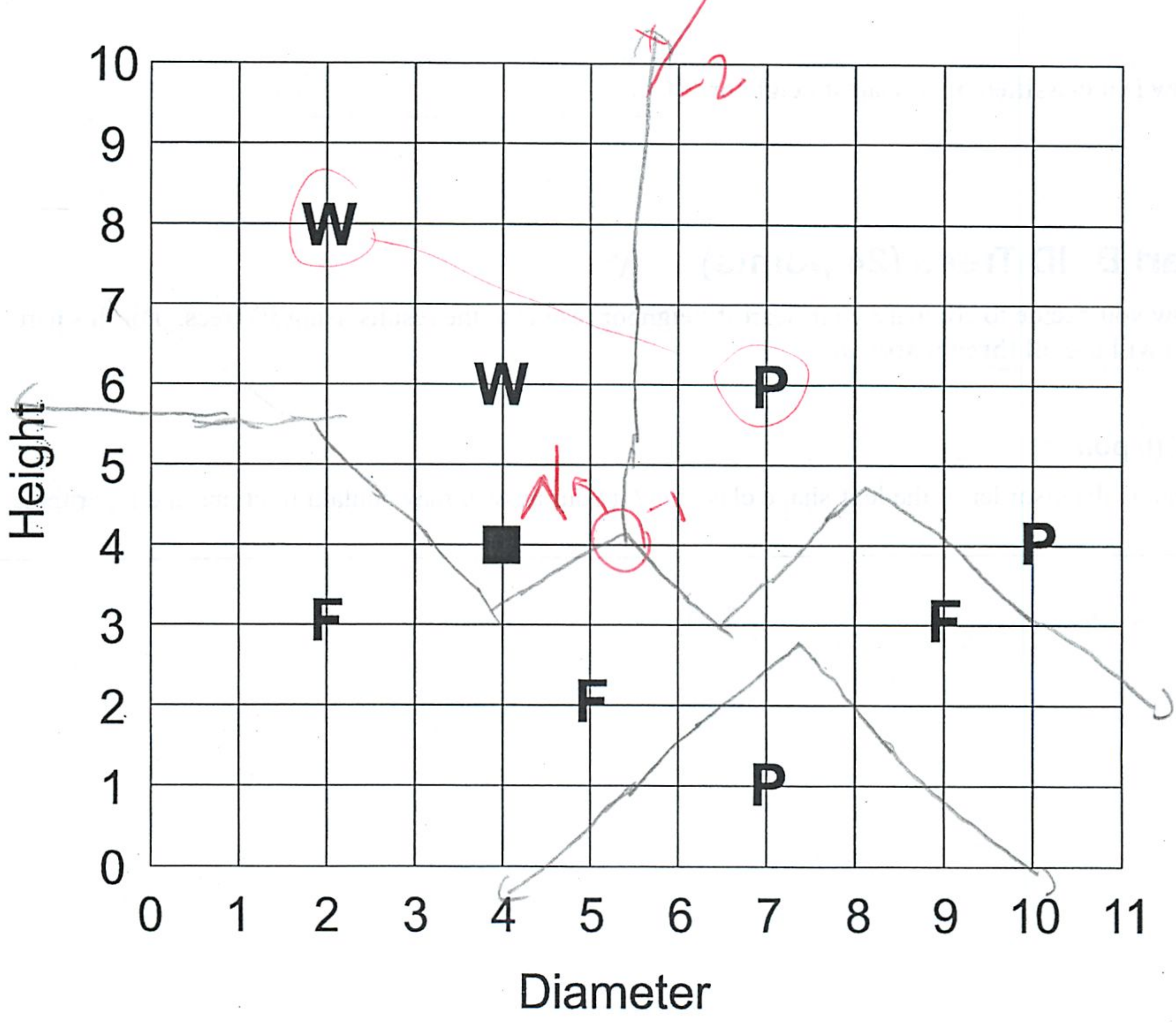


Figure duplicated on tear-off sheet.

A2 (4 points) +4

The unlabeled (square) point is one of the plants you have observed in your garden and want to classify.

How is it classified by 1-nearest neighbor? W ✓

How is it classified by 3-nearest neighbors? F ✓

Part B: ID Trees (24 points) +14

Now you decide to compare your nearest-neighbor results to the results using ID trees. For this part, you will use **all three features**.

B1 (8 points) +0

What is the disorder of the leaf-shape classifier? (Your answer may contain fractions and logarithms.)

$$\left(-\frac{4}{8}\ln\left(\frac{4}{8}\right) - \frac{4}{8}\ln\left(\frac{4}{8}\right)\right)$$

$$= .5$$

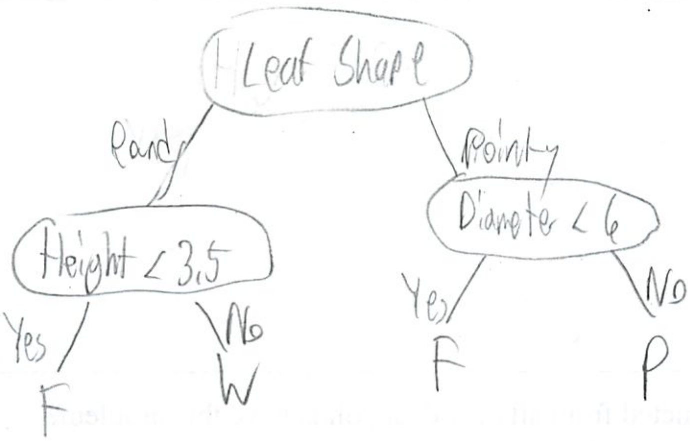
+0

↑ worst possible (does nothing!)

B2 (10 points) +10

You can now use **any horizontal or vertical thresholds** for the height and diameter features, in addition to the leaf-shape feature. Construct an ID tree that correctly classifies all of the labeled examples, using **no more than 3 classifiers** (multiple uses of the same classifier count as multiple). Your ID tree does NOT have to be constructed according to the greedy disorder-minimizing algorithm covered in class. ? So construct any way?

Draw your tree here:



B3 (6 points) +4

Suppose there were an additional example **in the training data** with the following characteristics:

Classification	Diameter	Height	Leaf shape
Food	7	1	pointy

If you tried to build an ID tree classifier based on all nine data points (the eight given initially plus this one), what problem would you encounter?

This food plant is growing exactly on top of the psycoactive plant (who also has a pointy leaf shape) So using leaf shape, diamater, hight won't seperate them!

Would the 1-nearest-neighbor classifier constructed from all nine data points have this problem? (Assume that you are still ignoring the leaf-shape feature.)

(Circle One) **YES** **NO**

would be P

Would the 3-nearest-neighbors classifier constructed from all nine data points have this problem? (Assume that you are still ignoring the leaf-shape feature.)

(Circle One) **YES** **NO**

Would be F

-2

Optionally, use this space to explain why your answer about 3-nearest-neighbors is correct (but don't waste time writing something if it's not helpful).

Since the new point is on top of a P point One of them will always be classified wrong 3-nearest neighbors would correctly classify new point as F, but now P point should be F according to 3NN

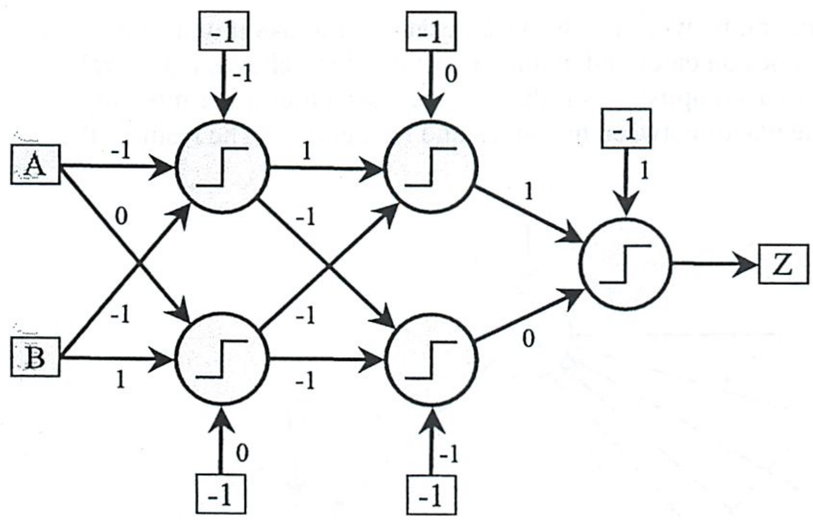
Problem 2: Neural Networks (40 points)

Part A: Forward Propagation (15 points)

Patty Luvbits is new to neural networks and really never wanted to leave her cozy world of binary logic. When Patty heard she could emulate binary logic using neural networks, she was ecstatic and promptly created several networks. Unfortunately, Patty forgot to label one of them and can no longer remember what logic function the network performs.

NOTE: This network uses the following unit step transfer function: $t(i) = \begin{cases} 0, & \text{for } i < 0 \\ 1, & \text{for } i \geq 0 \end{cases}$, where i is the sum of the weighted inputs and $t(i)$ is the output of the neuron. **Don't get confused:** $t(0) = 1$.

Help Patty by **CIRCLING** the logic function (on the right) emulated by the network:



AND **OR**

NAND **NOR**

XOR **NOT(B)**

Something Else

Figure duplicated on tear-off sheet.

Write in your calculated values for z below and compare them against the provided values for each of the prospective logic functions to find a match if one exists.

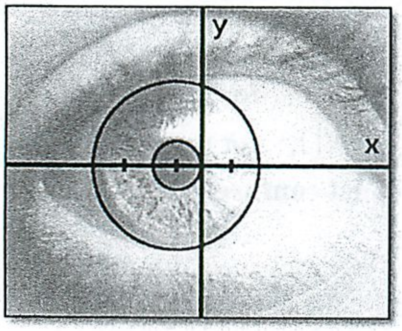
A	B	Z	AND	OR	NAND	NOR	XOR	NOT(B)
0	0	1	0	0	1	1	0	1
0	1	1	0	1	1	0	1	0
1	0	1	0	1	1	0	1	1
1	1	0	1	1	0	0	0	0

So Just run for each in

23/25

Part B: Manual Classification (25 points)

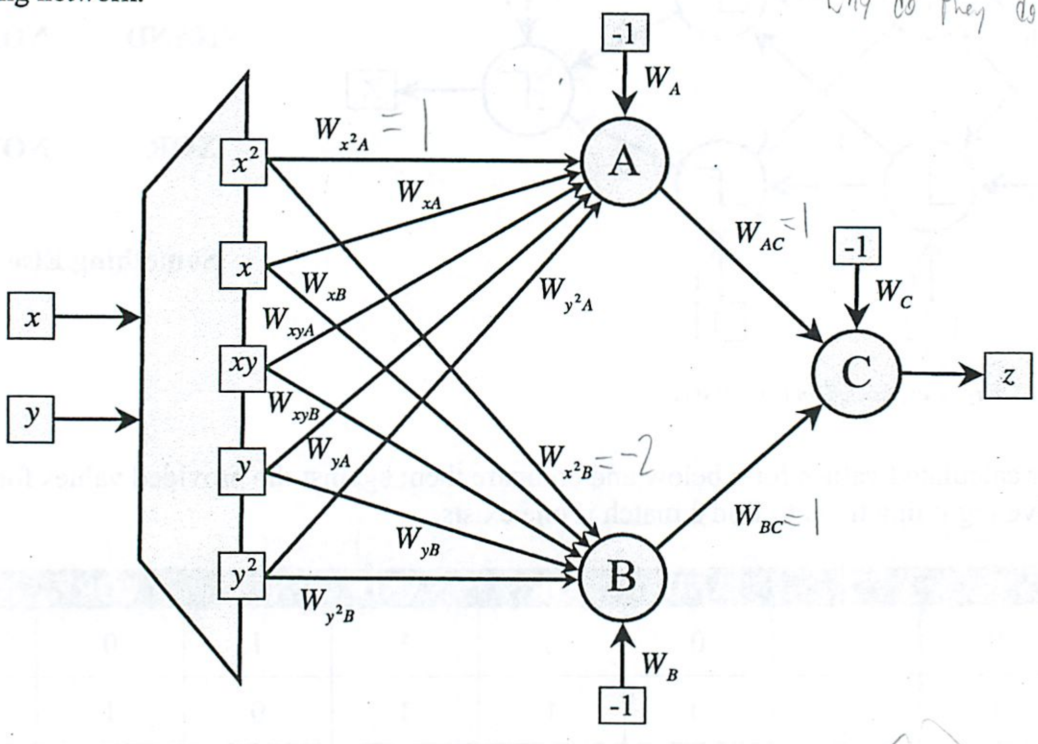
Ben Überfitz has just joined a new biometric research group and his first task is to construct a system that can accurately distinguish between points corresponding to iris pixels from those corresponding to non-iris pixels in pictures of eyes. Ben is given a sample picture of an eye which he marked up (shown below), and from which he derived the following classification function:



$$h(x,y) = \begin{cases} \text{iris}, & 1 \le (x+1)^2 + y^2 \le 9 \\ \text{not iris}, & \text{otherwise} \end{cases}$$

Ben is chomping at the bit to use a neural network for this task because he thinks it will sound extra neato when people ask him what he does on dates and at dinner parties. Ben believes his neural network is going to need some help to accomplish this task, so he devises a multiplier module containing a set of multipliers that he places between his inputs and his neurons. The result is the following network:

why do they do this silly stuff



NOTE: All neurons in Ben's network use the sigmoidal transfer function: $t(i) = \dfrac{1}{1+e^{-i}}$. See tear-off sheet for a graph of this sigmoid function's output. **Ben will indicate a positive classification (iris) if $z \ge 0.5$ and a negative classification (not iris) otherwise. Remember, $t(0) = 0.5$.**

-1

B1 (22 points) Below are tables, which map the names of weights in the diagram to corresponding values. Some weights have already been provided for you. Fill in the remaining weights *consistent with* the provided ones such that the neural network will classify points as iris or not iris according to the classification function given earlier.

Neuron A	
Weight	Value
W_{x^2A}	1
W_{xA}	2
W_{xyA}	0
W_{yA}	0
W_{y^2A}	1
W_A	1

Neuron B	
Weight	Value
W_{x^2B}	-2
W_{xB}	-4
W_{xyB}	0
W_{yB}	0
W_{y^2B}	-2
W_B	-16

-18-2

Neuron C	
Weight	Value
W_{AC}	1
W_{BC}	1
W_C	0 1.5

is it in bands

Neatly show work / formulas (for possible partial credit):

$W_C' = W_C + (\alpha \cdot i_C \cdot o_C(1-o_C)(d-o_C))$

but this isn't that

We have formula

other - but start w/ -2

so want inverse (what ever that is)

$-2x^2 - 4x$

call factor of 2

We want $(x+1)^2$, This is x^2+2x+1

A - adds

$W_{xA}=2$

$W_{xyA}=0$

$W_{yA}=0$

$W_{y^2}=1$

So have $x^2+2x+1+y^2$

But then want true if that ≥ 1

$x^2+2x+1+y^2 \geq 1$

$x^2+2x+y^2+1 \geq 0$

sigmoid (told to ignore)

$-2x^2-4x-2-2y^2 \leq 18$

$-x^2-2x-1-y^2 \leq 9$

opposite

$+x^2+2x+1+y^2 \geq 9$

Want − above 9

+ below 9

>1

B2 (3 points) Is Ben's solution likely to work properly on arbitrary eye images?

(Circle One) YES NO

Briefly explain your reasoning:

This always extracts this specific circle relative to center of image - it won't actually find the iris! Its a glorified "mask"

↓ outside time period! ~~100~~ 4

Problem 3: Near-miss Learning (20 points)

6.034 is so much fun, you decide to ask Professor Winston if he could use a UROP student. "Do you have any UROPs projects available?" He replies, "I was thinking of writing a system that would learn concepts like *revenge*, using near miss learning."

? genetic algorithms?

"Think about this," he says, handing you a sheet of paper with some scribbles on it. Then, he rushes off to the airport. "Words in CAPS indicate elements that must be present," he says over his shoulder.

Unfortunately, he has spilled coffee on the paper and several cells in the table have become unreadable.

Fill in the blank cells in the table making reasonable assumptions. ??

Example	Near Miss?	What is learned	Heuristic
Macbeth murders Duncan leads to Macduff kills Macbeth.	No	Initial model.	None.
Macbeth swindles Duncan leads to Macduff sues Macbeth.	Yes -4	Macbeth HARMS Duncan leads to Macduff HARMS Macbeth. add Z	Climb tree.
Pat pinches Chris leads to Chris hits Pat.	No	Pat HARMS Chris Chris HARMS Pat -4	Extend set -4
Macbeth pinches Duncan leads to Macduff hits Macbeth -4	Yes	PERSON X HARMS PERSON Y LEADS TO PERSON Z HARMS PERSON X.	Require link

??

6.034 Quiz 3 - Solutions
November 16, 2011

Name	Charles Babbage
Email	

Circle your **TA** *and* **recitation (for 1 extra credit point)**, so that we can more easily enter your score in our records and return your quiz to you promptly.

TAs	Recitations
Avril Kenney	Thu. 1-2, Bob Berwick
Adam Mustafa	Thu. 2-3, Bob Berwick
Caryn Krakauer	Thu. 3-4, Bob Berwick
Erek Speed	Fri. 1-2, Randall Davis
Gary Planthaber	Fri. 2-3, Randall Davis
Peter Brin	Fri. 3-4, Randall Davis
Tanya Kortz	

Problem	Maximum	Score	Grader
Extra Credit	1		
1	40		
2	40		
3	20		
Total	101		

There are a total of 10 pages in this quiz not including one or more tear off sheets that may be provided at the end with duplicate drawings and data. As always, open book, open notes, open just about everything, including a calculator, but no computers.

Problem 1: Nearest Neighbors and ID Trees (40 points)

You move into a new house, and discover that the garden is overgrown with all kinds of plants. You decide to figure out what they all are, and then deal with them accordingly. Fortunately, you know some information about the characteristics of other types of plants to compare them with:

Classification	Diameter	Height	Leaf shape
food	2	3	round
food	9	3	round
food	5	2	pointy
weed	2	8	round
weed	4	6	round
psychoactive	7	1	pointy
psychoactive	7	6	pointy
psychoactive	10	4	pointy

Part A: Nearest Neighbors (16 points)

First, you decide to use nearest-neighbor classification to categorize your plants. For this part, you will **only use the continuous features (height and diameter)**, ignoring the binary feature (leaf shape).

A1 (12 points)

The following graph shows the known data points in a two-dimensional space of height and diameter. Draw the decision boundaries produced by nearest-neighbor classification (1 nearest neighbor). Ignore the unlabeled (square) point.

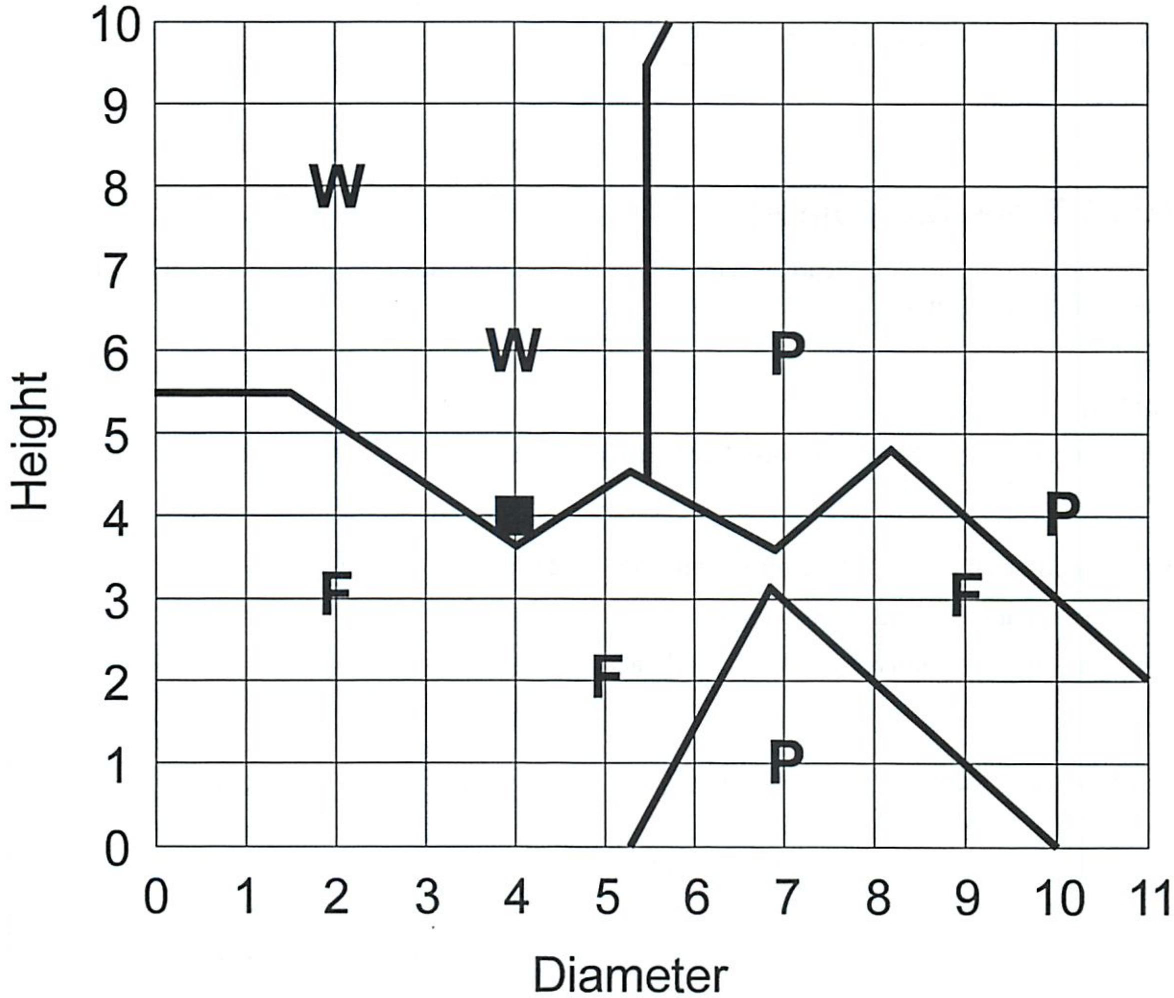


Figure duplicated on tear-off sheet.

A2 (4 points)

The unlabeled (square) point is one of the plants you have observed in your garden and want to classify.

How is it classified by 1-nearest neighbor?	**weed (W)**

How is it classified by 3-nearest neighbors?	**food (F)**

Part B: ID Trees (24 points)

Now you decide to compare your nearest-neighbor results to the results using ID trees. For this part, you will use **all three features**.

B1 (8 points)

What is the disorder of the leaf-shape classifier? (Your answer may contain fractions and logarithms.)

Leaf-shape classifier takes a set of 8 samples and creates 2 branches of 4 samples each:

Leaf shape = round: 2 food, 2 weed.

Leaf shape = pointy: 1 food, 3 psychoactive.

$$\text{Disorder (average entropy)} = \frac{4}{8}\left(-\frac{2}{4}\log_2\frac{2}{4}-\frac{2}{4}\log_2\frac{2}{4}\right)+\frac{4}{8}\left(-\frac{1}{4}\log_2\frac{1}{4}-\frac{3}{4}\log_2\frac{3}{4}\right)$$

$$= \frac{1}{2}+\frac{1}{2}\left(-\frac{1}{4}\log_2\frac{1}{4}-\frac{3}{4}\log_2\frac{3}{4}\right)$$

$$= \frac{3}{2}-\frac{3}{8}\log_2 3$$

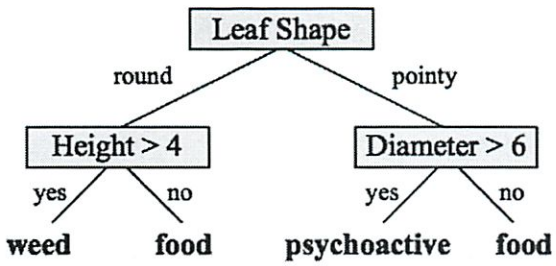
(NOTE: Simplification was not required)

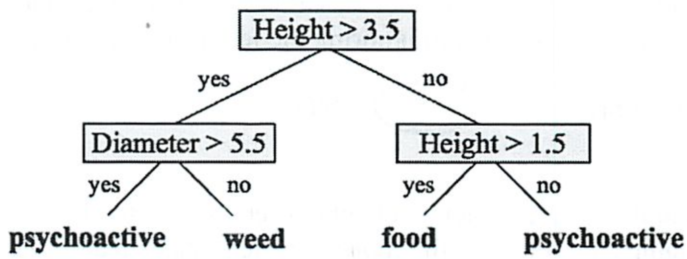
B2 (10 points)

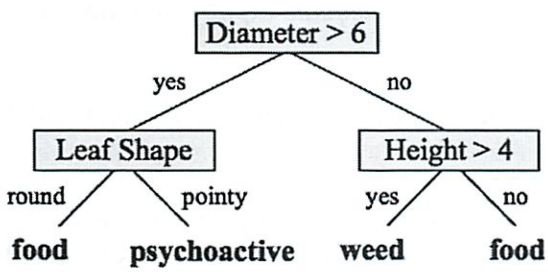
You can now use **any horizontal or vertical thresholds** for the height and diameter features, in addition to the leaf-shape feature. Construct an ID tree that correctly classifies all of the labeled examples, using **no more than 3 classifiers** (multiple uses of the same classifier count as multiple). Your ID tree does NOT have to be constructed according to the greedy disorder-minimizing algorithm covered in class.

Draw your tree here:

Several distinct ID tree solutions are possible for this problem. Some examples include:







For Diameter > 6, threshold can be anything between 5 and 7.

For Height > 4, threshold can be anything between 3 and 6.

B3 (6 points)

Suppose there were an additional example **in the training data** with the following characteristics:

Classification	Diameter	Height	Leaf shape
food	7	1	pointy

If you tried to build an ID tree classifier based on all nine data points (the eight given initially plus this one), what problem would you encounter?

The new sample with classification "food" has all the same feature values as another, existing sample with classification "psychoactive", so the ID tree would have no way to separate these two samples.

Would the 1-nearest-neighbor classifier constructed from all nine data points have this problem? (Assume that you are still ignoring the leaf-shape feature.)

(Circle One) **YES** **NO**

Would the 3-nearest-neighbors classifier constructed from all nine data points have this problem? (Assume that you are still ignoring the leaf-shape feature.)

(Circle One) **YES** **NO** *(Yes was accepted, if an adequate explanation was given)*

Optionally, use this space to explain why your answer about 3-nearest-neighbors is correct (but don't waste time writing something if it's not helpful).

YES: The 3-nearest-neighbors classifier will significantly reduce the size of the non-classifiable region, but there are still locations in the space where points would continue to be non-classifiable. Consider, for example, the point at Diameter = 8.5 and Height = 2.5: An attempt to classify at this point would yield not 3, but 4 nearest-neighbors with no majority classification (2 psychoactive and 2 food). **NOTE: This is a decision boundary and this is always trivially the case at any decision boundary, including the ones drawn for unproblematic 1-nearest neighbor problems.**

NO: With the exception of its resulting decision boundaries, the 3-nearest neighbors classifier removes the classification ambiguity through its majority voting mechanism.

Problem 2: Neural Networks (40 points)

Part A: Forward Propagation (15 points)

Patty Luvbits is new to neural networks and really never wanted to leave her cozy world of binary logic. When Patty heard she could emulate binary logic using neural networks, she was ecstatic and promptly created several networks. Unfortunately, Patty forgot to label one of them and can no longer remember what logic function the network performs.

<u>NOTE:</u> This network uses the following unit step transfer function: $t(i) = \begin{cases} 0, & \text{for } i < 0 \\ 1, & \text{for } i \geq 0 \end{cases}$, where i is the sum of the weighted inputs and $t(i)$ is the output of the neuron. **Don't get confused: $t(0) = 1$.**

Help Patty by **CIRCLING** the logic function (on the right) emulated by the network:

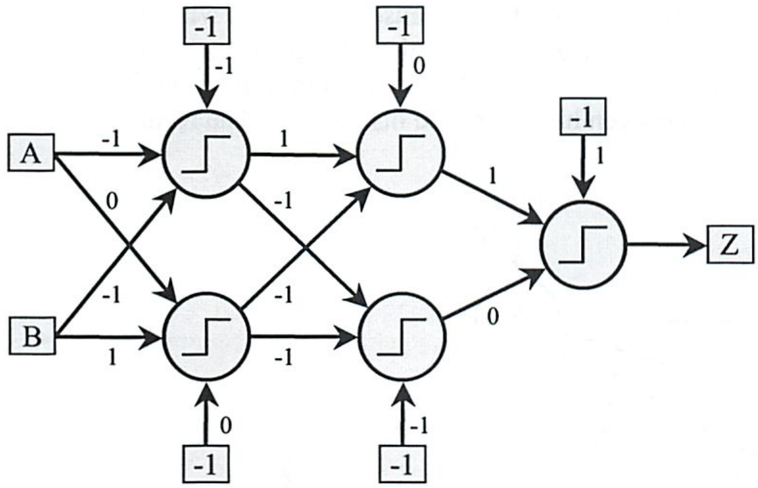


Figure duplicated on tear-off sheet.

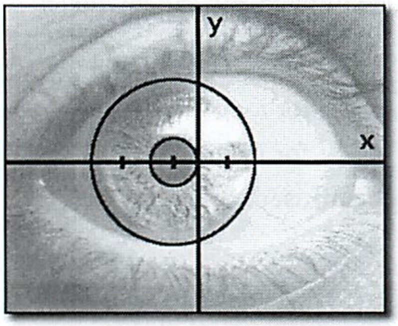
AND	**OR**
NAND	**NOR**
XOR	**NOT(B)**
Something Else	

Write in your calculated values for z below and compare them against the provided values for each of the prospective logic functions to find a match if one exists.

A	B	Z	AND	OR	NAND	NOR	XOR	NOT(B)
0	0	1	0	0	1	1	0	1
0	1	1	0	1	1	0	1	0
1	0	1	0	1	1	0	1	1
1	1	0	1	1	0	0	0	0

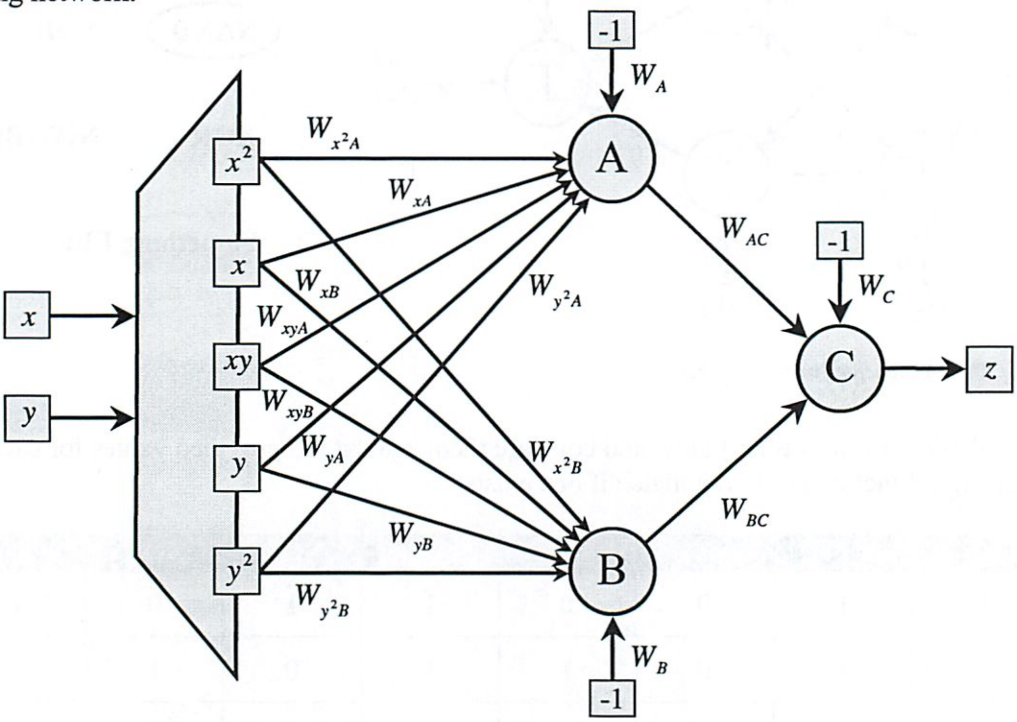
Part B: Manual Classification (25 points)

Ben Überfitz has just joined a new biometric research group and his first task is to construct a system that can accurately distinguish between points corresponding to iris pixels from those corresponding to non-iris pixels in pictures of eyes. Ben is given a sample picture of an eye which he marked up (shown below), and from which he derived the following classification function:



$$h(x,y) = \begin{cases} \text{iris}, & 1 \le (x+1)^2 + y^2 \le 9 \\ \text{not iris}, & \text{otherwise} \end{cases}$$

Ben is chomping at the bit to use a neural network for this task because he thinks it will sound extra neato when people ask him what he does on dates and at dinner parties. Ben believes his neural network is going to need some help to accomplish this task, so he devises a multiplier module containing a set of multipliers that he places between his inputs and his neurons. The result is the following network:



NOTE: All neurons in Ben's network use the sigmoidal transfer function: $t(i) = \frac{1}{1+e^{-i}}$. See tear-off sheet for a graph of this sigmoid function's output. **Ben will indicate a positive classification (iris) if $z \ge 0.5$ and a negative classification (not iris) otherwise. Remember, $t(0) = 0.5$.**

B1 (22 points) Below are tables, which map the names of weights in the diagram to corresponding values. Some weights have already been provided for you. Fill in the remaining weights *consistent with* the provided ones such that the neural network will classify points as iris or not iris according to the classification function given earlier.

Neuron A	
Weight	Value
W_{x^2A}	**1**
W_{xA}	2
W_{xyA}	0
W_{yA}	0
W_{y^2A}	1
W_A	0

Neuron B	
Weight	Value
W_{x^2B}	**-2**
W_{xB}	-4
W_{xyB}	0
W_{yB}	0
W_{y^2B}	-2
W_B	-16

Neuron C	
Weight	Value
W_{AC}	**1**
W_{BC}	**1**
W_C	1.5

Neatly show work / formulas (for possible partial credit):

Inner Circle Boundary:

$(x+1)^2 + y^2 = 1$

$x^2 + 2x + y^2 = 0$

Want to activate neuron from circle *outward.*

- ⇨ Coefficients should be positive.
- ⇨ Consistent with given W_{x^2A} of neuron A.

The coefficient of the x^2 term is 1, which is the same as W_{x^2A}. No scaling needed. This is neuron A's input. Read off weights.

Outer Circle Boundary:

$(x+1)^2 + y^2 = 9$

$x^2 + 2x + y^2 - 8 = 0$

Want to activate neuron from circle *inward.*

- ⇨ Coefficients should be negative.
- ⇨ Consistent with given W_{x^2B} of neuron B.

So, multiply through by -2 to match the given weight:

$-2x^2 - 4x - 2y^2 + 16 = 0$

This is neuron B's input. Read off weights.

Careful, though, W_B multiplied by its -1 input = 16.

To find W_C, we need to ensure that the C neuron still crisply cuts off at the two boundaries. At each boundary, one region has value 0.5 while the other is very nearly 1. We want $t(0)$ at the boundaries, so $W_C \approx 1.5$ to offset.

B2 (3 points) Is Ben's solution likely to work properly on arbitrary eye images?

(Circle One) **YES** **NO**

Briefly explain your reasoning:

Ben's solution is very specific to the geometry of the iris of this single sample image. Unless all arbitrary eye images had irises that fit inside the precise region that this one did, it would likely fail to properly classify iris pixels from non-iris pixels for most other images because the irises in those images may have different sizes and proportions. His solution is not very general at all even though it is engineered to work perfectly for this specific example.

Problem 3: Near-miss Learning (20 points)

6.034 is so much fun, you decide to ask Professor Winston if he could use a UROP student. "Do you have any UROPs projects available?" He replies, "I was thinking of writing a system that would learn concepts like *revenge*, using near miss learning."

"Think about this," he says, handing you a sheet of paper with some scribbles on it. Then, he rushes off to the airport. "Words in CAPS indicate elements that must be present," he says over his shoulder.

Unfortunately, he has spilled coffee on the paper and several cells in the table have become unreadable.

Fill in the <u>blank cells</u> in the table making reasonable assumptions.

Example	Near Miss?	What is learned	Heuristic
Macbeth murders Duncan leads to Macduff kills Macbeth.	No	Initial model.	None.
Macbeth swindles Duncan leads to Macduff sues Macbeth.	No	Macbeth HARMS Duncan leads to Macduff HARMS Macbeth.	Climb tree.
Pat pinches Chris leads to Chris hits Pat.	No	PERSON X HARMS PERSON Y leads to PERSON Z HARMS PERSON X. The individuals become variables with PERSON Y and PERSON Z not necessarily being the same individual.	Climb tree.
Jack punches John. Fred kicks Jack. This example causes the model to learn that "leads to" is required, and it is a near miss, so it has to be a negative example that matches the model on everything except the "leads to".	Yes	PERSON X HARMS PERSON Y LEADS TO PERSON Z HARMS PERSON X.	Require link.