CSC 236 HW 2

Check in period: 6/8 - 6/13

About

Please see the guide to hw, and guide to check ins for information and tips for the HW and the check ins!

These homeworks are on lectures and tutorials 1-4 inclusive.

1 Song Order

In this problem, you've been hired as a consultant for guitar player extraordinaire - Andy McKee. You've been asked to help him figure out the 'best' song order for a live gig. Here is some more context:

A guitar has six strings, each of which can be tuned to one of 88 musical pitches[1]. The strings are usually tuned to specific pitches known as 'standard tuning.' However, Andy uses many non-standard tunings and, for this problem, assume that every song he plays is in a different tuning. See here for an example of Andy tuning his guitar.

Here is an example. In standard tuning, strings 1 through 6 are tuned to the pitches 44, 39, 35, 30, 25, and 20, respectively. Strings 1 through 6 are tuned to 42, 37, 35, 30, 25, and 18 in one altered tuning. To get from standard tuning to this altered tuning, you'd tune the first, second and sixth strings down by 2 pitches each.

Let $\mathrm{Tunings}$ be the set of possible guitar tunings. Let $\mathrm{Songs} \subseteq \mathrm{Tunings}$ be the subset of $\mathrm{Tunings}$ that Andy would like to play for the show. Let S be standard tuning and assume $S \in \mathrm{Songs}$. Andy would like you to find the ordering of songs in Songs such that

- The guitar starts and ends in standard tuning S.
- The total amount of 'shifting' required is minimized.

Question 1a. Model this problem as one of the graph problems we studied in class. In particular, mathematically formalize the definitions of tunings, and fully specify the vertex set, the edge set, and edge weights (if any). Then, explain how a solution to the graph problem corresponds to the optimal song ordering described above.

Hint: define a notion of the cost of moving from tuning x to tuning y.

[1] In reality, the set of possible pitches for each string is way smaller, but this assumption is OK for this problem since we restrict our attention to only the tunings that Andy uses

2 Trees

Question 2a. Let G be an undirected graph. Show that if G is maximally acyclic, then G is a tree.

Question 2b. Let G be an undirected graph. Show that if G is minimally connected, then G is a tree.

3 Induction

Question 3a. Show that for all $n \in \mathbb{N}$ with $n \geq 1$,

$$\left(1+\frac{1}{1}\right)\left(1+\frac{1}{2}\right)\left(1+\frac{1}{3}\right)...\left(1+\frac{1}{n}\right) = n+1$$

Question 3b. Show by induction $\forall n \in \mathbb{N}.(n^3 \leq 3^n)$

Question 3c.

Your friend claims the following. **Claim:** $a^n = 1$ for all $n \in \mathbb{N}, a \in \mathbb{R}_{>0}$, and proposes the following proof by complete induction.

Proof. **Base case.** The base case holds since $a^0 = 1$ for any non-zero a.

Inductive step. Let $k \in \mathbb{N}$ be any natural number and assume for every $m \leq k$, and $a \in \mathbb{R}_{>0}$, $a^m = 1$. Then we have

$$a^{k+1} = a^{k+k-(k-1)} = \frac{a^k \cdot a^k}{a^{k-1}} = \frac{1 \cdot 1}{1} = 1,$$

which completes the induction. □

Find the flaw in the following argument, and explain the mistake.

4 Round Robin

A round-robin tournament is a tournament in which every player plays a single game (rock-paper-scissors, tennis, chess, etc.) with every other player.

If V is a set of players, we encode the results of a tournament as a directed graph $G = (V, E)$, $E = \{(p, q) \in V \times V : p \text{ beats } q\}$

Question 4a. Let $G = (V, E)$ be a round-robin tournament. Here are two more definitions.

- A player p is a winner if for all other players q, either p beat q or p beat someone who beat q.
- A player p has a maximal number of wins if there is no other player q with strictly more wins than p.

Is a player with the maximal number of wins always a winner? Prove your answer!

Question 4b. Let $G = (V, E)$ be any round-robin tournament encoded as a directed graph. Show that if $|V| \geq 3$, and G has some cycle, G has a cycle of length exactly 3. Hint: Try the Well Ordering Principle!

5 Rubik's Inevitability

A Rubik's cube is a fun puzzle/toy. If you don't have a Rubik's cube, check out this simulator.

Assume the blue face of the cube is always facing you. A state of the cube describes the current position of all the stickers. For example, the solved state of the cube is the one where every face has just one color (namely, the one matching the center square of the face).

The cube has 6 faces. Let f_i be the operation of turning the ith face of the cube a quarter turn clockwise. Let $\text{Turns} = \{f_1, ..., f_6\}$ be the set of turns one can perform on the Rubik's cube.

Question 5a. Let s be the solved state of the Rubik's cube. Define the set of possible Rubiks cube states, Rubiks, recursively or inductively.

Question 5b. Model the task of solving a Rubik's cube in the minimum number of moves as a graph problem. Fully specify the vertex set, edge set, and edge weights (if any), and explain why a solution to the graph problem corresponds to solving the cube.

Start with a solved Rubik's cube and consider any fixed sequence of turns. Now repeat that sequence of turns many times. **Eventually, somewhat magically, the Rubik's cube will return to the solved configuration!** In this problem, we will prove this fact.

Question 5c. Show that forall $f \in \text{Turns}$, f is an injective function from Rubiks to Rubiks.

Question 5d. Prove by induction that for all natural numbers $n \geq 1$, if $g_1, g_2, ..., g_n \in \text{Turns}$, the composition $g_n \circ g_{n-1} \circ ... \circ g_1$ is injective.

Question 5e. Give an upper bound on $|\text{Rubiks}|$ i.e. find some k such that $|\text{Rubiks}| \leq k$

Let g be an arbitrary sequence of turns. Denote

$$g^m = g \underbrace{\circ ... \circ}_{m \text{ times}} g$$

to be the function that applies g m times. Also, let $s \in \text{Rubiks}$ be the solved state of the cube.

Question 5f. Show that for some $m \in \mathbb{N}$ with $m \geq 1$, $g^m(s) = s$. That is, after m applications of the seqeunce of moves defined by g, we return to the solved state!

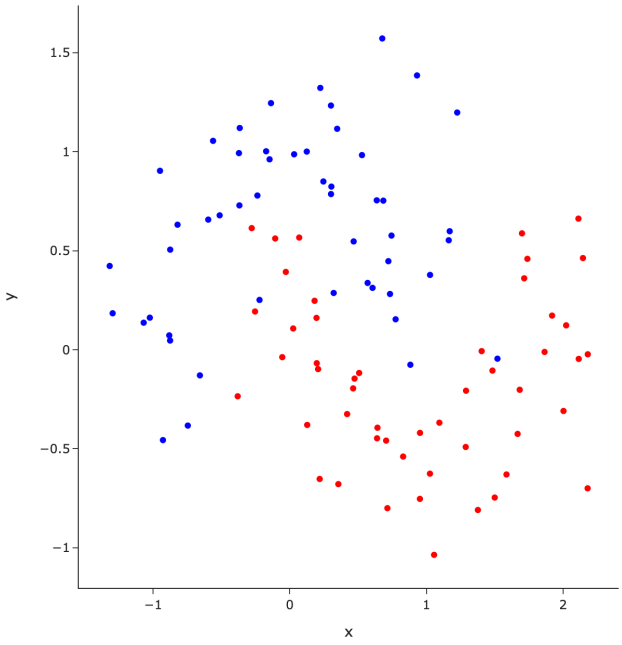
Hint: Consider the sequence $s_0, s_1, s_2, ...$ where $s_0 = s$, and $s_i = g^i(s)$.

6 A Neural Network Breakthrough?

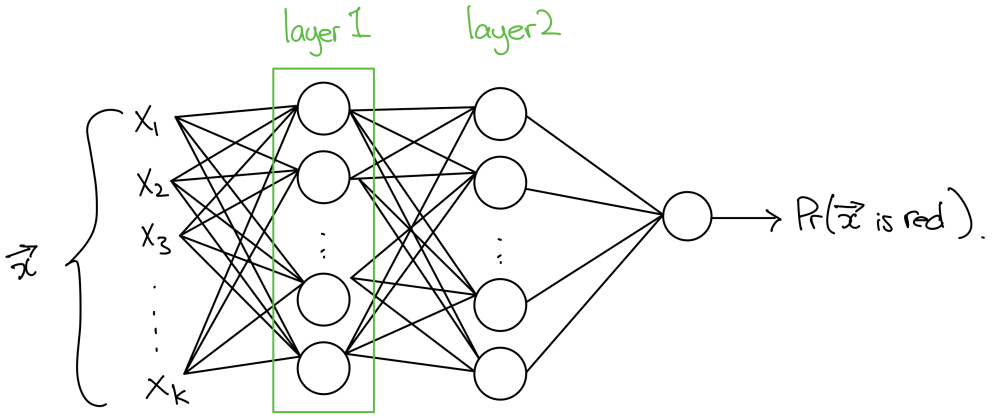
Your friend approaches you with a new idea, claiming it is possibly a breakthrough in neural networks for binary classification.

Binary classification is the problem where you're given some input and asked to classify the input as one of two classes. One example of binary classification is to predict whether or not the patient has pneumonia based on X-ray images. In this problem, we will focus on predicting whether a point is blue or red given the x and y coordinates.

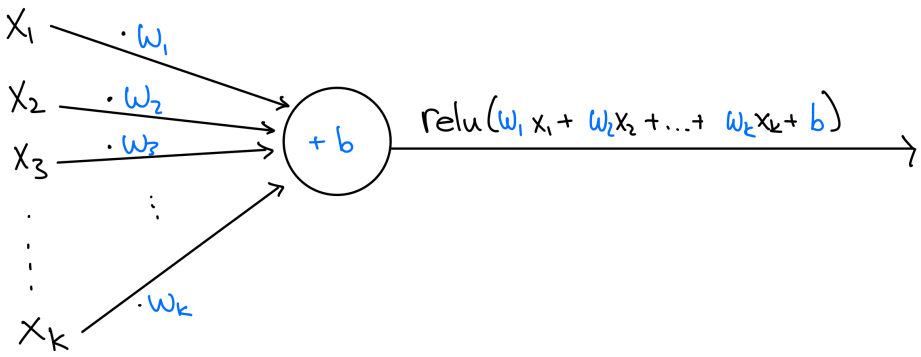
Our data looks like this:



A neural network looks like the following.



The circles are called neurons and compute as follows. Zooming on a single neuron:



Let k be the number of inputs for a particular neuron. Each neuron stores $k+1$ parameters, one weight, $w_i \in \mathbb{R}$, for each input, and one additional parameter called the bias, $b \in \mathbb{R}$. The output of the neuron is $\text{relu}(w_1x_1+w_2x_2+...+w_kx_k+b)$. In words, the neuron computes the weighted sum of the inputs, adds the bias term, and applies relu. $\text{relu} : \mathbb{R} \to \mathbb{R}$ is function defined as follows

$$\text{relu}(x) = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases}$$

Training a neural network amounts to finding the 'best' setting of the weights and biases according to the data.

Neurons are then arranged into layers so that the first layer gets the inputs from the actual inputs, and subsequent layers get their inputs from the outputs of the previous layers! One reason neural networks are so powerful is that you can have many layers, each transforming the data in more complex ways. The size of a layer is the number of neurons in it.

The final layer contains a single neuron whose output is the probability that the input is red. Then, if the probability is ≥ 0.5, we'll predict the point is red and otherwise blue.

The final neuron is very slightly different from the other neurons. It still computes a weighted average of its inputs plus a bias. However, it applies a different function called σ ('sigmoid') instead of relu.

Here is your friend's idea:

Look carefully at what relu *does. If the input to relu is positive, we report the value, but if it is negative, relu just outputs 0. Isn't that wasting a lot of information? Shouldn't it be useful for later neurons to know how negative the input was? So here is my idea, instead of having a neuron output* $\text{relu}(w_1x_1 + w_2x_2 + ... + w_kx_k + b)$*, have it output* $w_1x_1 + w_2x_2 + ... + w_kx_k + b$*.*

Question 6a. Test this theory out for yourself here.

First, run the model as is to check that it works.

Then, to test your friend's theory, remove the `activation="relu"` optional arguments in the definition of the model and try rerunning it. Try again with more and bigger layers.

How does the model do? Does your friend's idea work?

Question 6b. Unfortunately, your friend's idea doesn't work. Prove the following statement to show this:

If neurons compute in the suggested way, then any neural network with an arbitrary number of layers with arbitrary sizes is equivalent to some single output neuron (and thus, can't be all that powerful!)