Neural Network Lab, May 2017

Name:

Period:

Date Submitted:

**Instructions: Complete the parts below. You may leave the instructions in tact or delete them but please leave the 4 parts clearly separate. Submit final version on dropbox, as indicated on BB.**

**Part 1:**

Using the 14 linearly separable Boolean functions on 2 variables, how many 2-2-1 networks can you construct to calculate XOR? List all the solutions you find. How many unique solutions exists if you consider symmetry (e.g. ABC is just a mirror image of BAC)? Describe how your algorithm works.

0

node\_3 weights [1.3391615240814208, 1.5562279357686384] 2.66077135154

node\_4 weights [0.99999385323081169, 1.0000001603099296] 0.499942367354

node\_5 weights [-1.50000019992872, 1.49999980007128] 0.500000399859

Solution: 1 6 4

1

node\_3 weights [1.3391615240814208, 1.5562279357686384] 2.66077135154

node\_4 weights [-1.0000001587048899, -1.0000001999276482] -0.499999464112

node\_5 weights [-1.0000001587048899, -1.0000001999276482] -0.499999464112

Solution: 1 7 7

2

node\_3 weights [1.499999956715806, -2.0000000412230539] 0.499999808315

node\_4 weights [-1.50000019992872, 1.49999980007128] 0.500000399859

node\_5 weights [0.99999385323081169, 1.0000001603099296] 0.499942367354

Solution: 2 4 6

3

node\_3 weights [1.499999956715806, -2.0000000412230539] 0.499999808315

node\_4 weights [1.5000000020611568, -1.0000002205436762] -0.50000014634

node\_5 weights [1.5000000020611568, -1.0000002205436762] -0.50000014634

Solution: 2 9 9

4

node\_3 weights [-1.50000019992872, 1.49999980007128] 0.500000399859

node\_4 weights [1.499999956715806, -2.0000000412230539] 0.499999808315

node\_5 weights [0.99999385323081169, 1.0000001603099296] 0.499942367354

Solution: 4 2 6

5

node\_3 weights [-1.50000019992872, 1.49999980007128] 0.500000399859

node\_4 weights [-1.0000001958088691, 1.9999999999999993] -0.500000002062

node\_5 weights [1.5000000020611568, -1.0000002205436762] -0.50000014634

Solution: 4 11 9

6

node\_3 weights [0.99999385323081169, 1.0000001603099296] 0.499942367354

node\_4 weights [1.3391615240814208, 1.5562279357686384] 2.66077135154

node\_5 weights [1.499999956715806, -2.0000000412230539] 0.499999808315

Solution: 6 1 2

7

node\_3 weights [0.99999385323081169, 1.0000001603099296] 0.499942367354

node\_4 weights [-1.9999999979381347, -1.0000001978700204] -2.5

node\_5 weights [1.3391615240814208, 1.5562279357686384] 2.66077135154

Solution: 6 12 1

8

node\_3 weights [-1.0000001587048899, -1.0000001999276482] -0.499999464112

node\_4 weights [1.3391615240814208, 1.5562279357686384] 2.66077135154

node\_5 weights [-1.0000001587048899, -1.0000001999276482] -0.499999464112

Solution: 7 1 7

9

node\_3 weights [-1.0000001587048899, -1.0000001999276482] -0.499999464112

node\_4 weights [-1.9999999979381347, -1.0000001978700204] -2.5

node\_5 weights [-1.50000019992872, 1.49999980007128] 0.500000399859

Solution: 7 12 4

10

node\_3 weights [1.5000000020611568, -1.0000002205436762] -0.50000014634

node\_4 weights [1.499999956715806, -2.0000000412230539] 0.499999808315

node\_5 weights [-1.0000001958088691, 1.9999999999999993] -0.500000002062

Solution: 9 2 11

11

node\_3 weights [1.5000000020611568, -1.0000002205436762] -0.50000014634

node\_4 weights [-1.0000001958088691, 1.9999999999999993] -0.500000002062

node\_5 weights [-1.9999999979381347, -1.0000001978700204] -2.5

Solution: 9 11 12

12

node\_3 weights [-1.0000001958088691, 1.9999999999999993] -0.500000002062

node\_4 weights [-1.50000019992872, 1.49999980007128] 0.500000399859

node\_5 weights [-1.0000001958088691, 1.9999999999999993] -0.500000002062

Solution: 11 4 11

13

node\_3 weights [-1.0000001958088691, 1.9999999999999993] -0.500000002062

node\_4 weights [1.5000000020611568, -1.0000002205436762] -0.50000014634

node\_5 weights [-1.9999999979381347, -1.0000001978700204] -2.5

Solution: 11 9 12

14

node\_3 weights [-1.9999999979381347, -1.0000001978700204] -2.5

node\_4 weights [0.99999385323081169, 1.0000001603099296] 0.499942367354

node\_5 weights [1.3391615240814208, 1.5562279357686384] 2.66077135154

Solution: 12 6 1

15

node\_3 weights [-1.9999999979381347, -1.0000001978700204] -2.5

node\_4 weights [-1.0000001587048899, -1.0000001999276482] -0.499999464112

node\_5 weights [1.499999956715806, -2.0000000412230539] 0.499999808315

Solution: 12 7 2

**Part 2:**

Construct a NN using any topology you like to recognize coordinates within the unit circle. Implement your best model in code and compute its accuracy over 10,000 randomly selected points within the square -3/2 <= x,y <= 3/2. Describe your best model and report its accuracy.

Describe the first NN you came up with and how your design evolved into its final form. If your best model uses sigmoid actuator functions, what is the optimal “k” and “c” values you found (exponential coefficient, and rounding threshold). If your best model uses step actuators, how do you design the nodes and find their weights? Discuss your implementation challenges and how you determined your best model.

We first began by creating 4 perceptrons which checked if X and Y less than 1 and greater than -1. Then we created 2 and nodes, which would then go to another and node. After creating the square, the use of the sigmoid function was used to increase the accuracy of the nodes. It was found that when k = 4.2 the accuracy was the highest of 97.7%.

**Part 3:**

Implement an ad-hoc learning algorithm to discover the weights in a 2-2-1 network topology for XOR, using sigmoid actuator functions. Show the final weights learned by your algorithm (rounded and formatted nicely). Write pseudocode that clearly describes how your search algorithm works. Also discuss how your ideas evolved and any failed attempts along the way.

**Analysis**: Explore the convergence rate of your algorithm as a function of the learning rate. Count the number of iterations required for your algorithm to converge, averaged over several samples (say 10-100 per trial), as you vary the learning rate, lambda, and produce a graph. The appropriate range of lambda will depend on your implementation, but generally a lambda of “1” means you add “1” to the weights at each step. Find the approximate lambda which seems to minimize convergence time.

Describe how your algorithm depends on lambda and show a graph of “lambda vs. iterations”.

**Weights**: Node 1: [-2, 2, 1] Node 2: [2, -3, 2] Node 3: [ -4, -4, 6]

**Pseudo Code:**

While error is greater than .001

W = array of 9 numbers (random number between -1,1)

Delta = array of 9 numbers (number between -lambda,lambda)

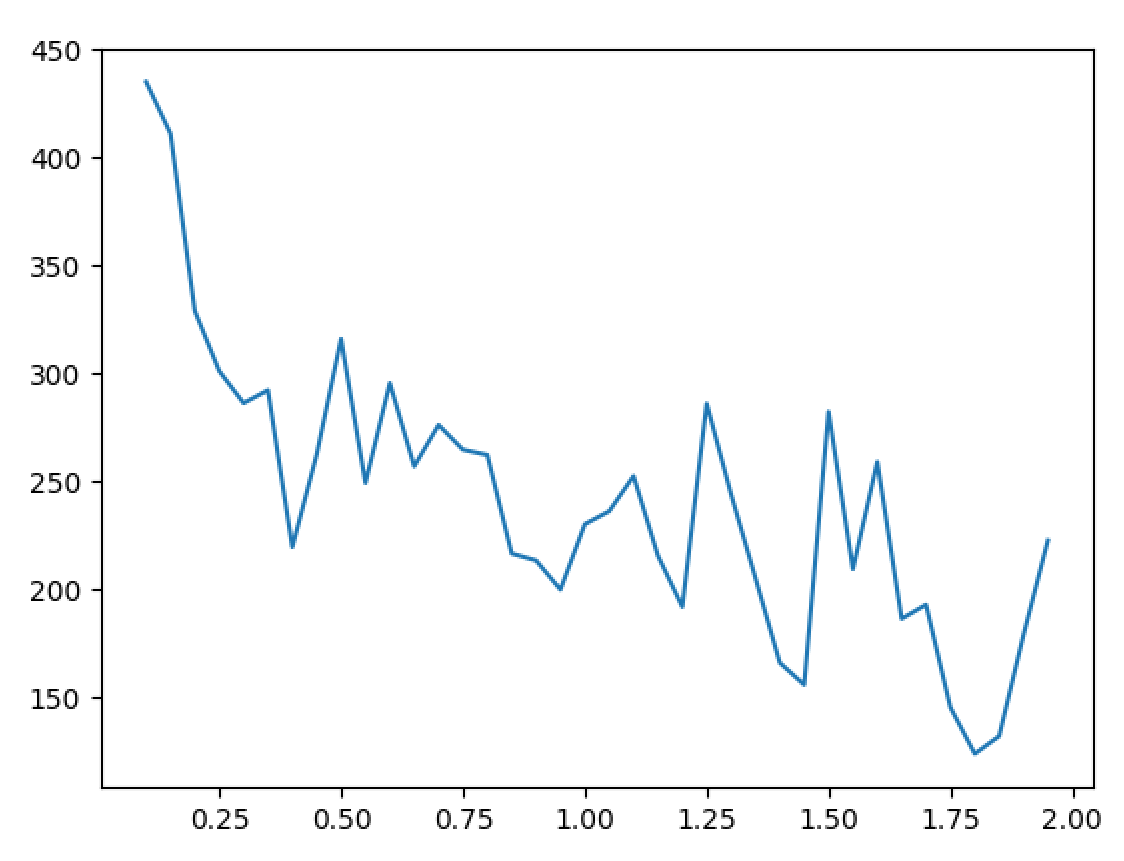
If error(w + delta) < error(w)

Then w = w+delta

Run for maximum of 3000 times

The best way to understand the algorithm is to think of each weight/bias as a dimension. After random selecting to move in a random direction in these 9 dimensions if this random selection is closer to the correct value then we make that the new value for w.

**Graph Lambda vs Iterations**



**Part 4: (Bonus/Extension)**

Modify your learning algorithm (or create a new one) to find optimal weights for part 2. You may also want to include “k” and “c” in your search space. Describe your results.