# CS-362 Homework 3

Jacob Hurst

April 15, 2019

# 1 Introduction

Nim is a game in which two players alternately take one or more objects from one of a number of heaps, each trying to take, or to compel the other to take, the last remaining object. The neural network designed in 'nim_network.py' is designed to play an optimal game of nim against users.

# 2 Read Me

```
# Nim Neural Network

A neural network to optimally play nim against. Converted from a logical circuit based solver which
uses Sprague-Grundy theorem to optimally play nim. See sampleruns.txt for sample usages and tests.

## Usage

Run with 'python3 nim_network.py <pile_1_size> <pile_2_size> <pile_3_size>' to start a game of nim
against the neural network with the supplied pile sizes as arguments.

## Authors

* **Jacob Hurst** - jhurst@cs.unm.edu
```

# 3 Sample Outputs

```
RUN 1
>> python3 nim_network.py 8 15 5

Pile sizes: 8, 15, 5.
Computer took 2 from pile 1!
Pile sizes: 8, 13, 5.
Human turn | provide your move 'amount,index': 8,0
Human took 8 from pile 0!
Pile sizes: 0, 13, 5.
Computer took 8 from pile 1!
Pile sizes: 0, 5, 5.
Human turn | provide your move 'amount,index': 2,1
Human took 2 from pile 1!
Pile sizes: 0, 3, 5.
Computer took 2 from pile 2!
Pile sizes: 0, 3, 3.
Human turn | provide your move 'amount,index': 1,2
Human took 1 from pile 2!
Pile sizes: 0, 3, 2.
Computer took 1 from pile 1!
Pile sizes: 0, 2, 2.
Human turn | provide your move 'amount,index': 1,1
Human took 1 from pile 1!
Pile sizes: 0, 1, 2.
Computer took 1 from pile 2!
Pile sizes: 0, 1, 1.
Human turn | provide your move 'amount,index': 1,1
Human took 1 from pile 1!
Pile sizes: 0, 0, 1.
Computer took 1 from pile 2!
Computer won!

RUN 2
```

```
>> python3 nim_network.py 10 10 10

Pile sizes: 10, 10, 10.
Computer has 3 available moves.
Computer took 10 from pile 0!
Pile sizes: 0, 10, 10.
Human turn | provide your move 'amount,index': 10,2
Human took 10 from pile 2!
Pile sizes: 0, 10, 0.
Computer took 10 from pile 1!
Computer won!

RUN 3
>> python3 nim_network.py 63 63 63

Pile sizes: 63, 63, 63.
Computer has 3 available moves.
Computer took 63 from pile 0!
Pile sizes: 0, 63, 63.
Human turn | provide your move 'amount,index': 32,2
Human took 32 from pile 2!
Pile sizes: 0, 63, 31.
Computer took 32 from pile 1!
Pile sizes: 0, 31, 31.
Human turn | provide your move 'amount,index': 1,1
Human took 1 from pile 1!
Pile sizes: 0, 30, 31.
Computer took 1 from pile 2!
Pile sizes: 0, 30, 30.
Human turn | provide your move 'amount,index': 16,2
Human took 16 from pile 2!
Pile sizes: 0, 30, 14.
Computer took 16 from pile 1!
Pile sizes: 0, 14, 14.
Human turn | provide your move 'amount,index': 4,1
Human took 4 from pile 1!
Pile sizes: 0, 10, 14.
Computer took 4 from pile 2!
Pile sizes: 0, 10, 10.
Human turn | provide your move 'amount,index': 10,2
Human took 10 from pile 2!
Pile sizes: 0, 10, 0.
Computer took 10 from pile 1!
Computer won!
```

# 4 Code

```
import sys
from numpy import *

bias = -1

class Neuron:
    def __init__(self, num_inputs):
        self.num_inputs = num_inputs
        self.weights = [0 for _ in range(0, num_inputs+1)]

    def set_weights(self, weights):
```

```
        self.weights = weights

    def set_weight(self, index, weight):
        self.weights[index] = weight

    def compute(self, inputs):
        return dot(self.weights[:-1], inputs)

    def __str__(self):
        return (str(len(self.weights[:-1]))+" Weights: "+str(self.weights[:-1])+", Bias: "+str(self.
            ↪ weights[-1]))

class NeuronLayer:
    def __init__(self, num_neurons, num_inputs):
        self.num_neurons = num_neurons
        self.neurons = [Neuron(num_inputs) for _ in range(0, self.num_neurons)]

    def __str__(self):
        return ("Layer:\n\t"+"\n\t".join([str(i+1)+"| "+str(neuron) for i,neuron in enumerate(self.
            ↪ neurons)]))

class NeuralNetwork:
    def __init__(self, dimensions):
        self.num_inputs = dimensions[0]
        self.num_outputs = dimensions[-1]
        self.num_neurons_per_hl = dimensions[1:-1]

        self.construct()

    def construct(self):
        self.layers = [NeuronLayer(self.num_neurons_per_hl[0], self.num_inputs)]
        for i in range(1, len(self.num_neurons_per_hl)-1):
            self.layers += [NeuronLayer(self.num_neurons_per_hl[i], self.num_neurons_per_hl[i-1])]
        self.layers += [NeuronLayer(self.num_outputs, self.num_neurons_per_hl[-1])]

        for i in range(0,18):
            self.fwd_weights(0, i, i)
            self.fwd_weights(1, i, i)
            self.fwd_weights(2, i, i)
            self.fwd_weights(3, i, i)
            self.fwd_weights(4, i, i)
            self.fwd_weights(5, i, i)

        self.xor_weights(0, 18, 18, 0, 6)
        self.xor_weights(0, 21, 21, 1, 7)
        self.xor_weights(0, 24, 24, 2, 8)
        self.xor_weights(0, 27, 27, 3, 9)
        self.xor_weights(0, 30, 30, 4, 10)
        self.xor_weights(0, 33, 33, 5, 11)

        self.xor_weights(2, 18, 18, 18, 12)
        self.xor_weights(2, 21, 19, 21, 13)
        self.xor_weights(2, 24, 20, 24, 14)
        self.xor_weights(2, 27, 21, 27, 15)
        self.xor_weights(2, 30, 22, 30, 16)
        self.xor_weights(2, 33, 23, 33, 17)

        self.xor_weights(4, 18, 18, 18, 0)
        self.xor_weights(4, 21, 19, 19, 1)
        self.xor_weights(4, 24, 20, 20, 2)
```

```
        self.xor_weights(4, 27, 21, 21, 3)
        self.xor_weights(4, 30, 22, 22, 4)
        self.xor_weights(4, 33, 23, 23, 5)

        self.xor_weights(4, 36, 24, 18, 6)
        self.xor_weights(4, 39, 25, 19, 7)
        self.xor_weights(4, 42, 26, 20, 8)
        self.xor_weights(4, 45, 27, 21, 9)
        self.xor_weights(4, 48, 28, 22, 10)
        self.xor_weights(4, 51, 29, 23, 11)

        self.xor_weights(4, 54, 30, 18, 12)
        self.xor_weights(4, 57, 31, 19, 13)
        self.xor_weights(4, 60, 32, 20, 14)
        self.xor_weights(4, 63, 33, 21, 15)
        self.xor_weights(4, 66, 34, 22, 16)
        self.xor_weights(4, 69, 35, 23, 17)

        for i in range(18,36):
            self.fwd_weights(6, i, i)
            self.fwd_weights(7, i, i)
            self.fwd_weights(8, i, i)

        self.sub_weights(6, 0, 0, 0, 18)
        self.sub_weights(6, 6, 1, 6, 24)
        self.sub_weights(6, 12, 2, 12, 30)

        for i in range(0,3):
            self.fwd_weights(8, i, i)
            self.fwd_weights(9, i, i)

        for i in range(0,6):
            self.and_weights(8, 3+i, 0, 18+i)
            self.and_weights(8, 9+i, 1, 24+i)
            self.and_weights(8, 15+i, 2, 30+i)

        self.sum_weights(9, 3, 3, 9, 15)

    def xor_weights(self, which, to1, to2, from1, from2):
        self.and_weights(which, to1, from1, from2)

        self.layers[which].neurons[to1+1].weights[from1] = 1
        self.layers[which].neurons[to1+2].weights[from2] = 1

        self.layers[which+1].neurons[to2].weights[to1] = -2
        self.layers[which+1].neurons[to2].weights[to1+1] = 1
        self.layers[which+1].neurons[to2].weights[to1+2] = 1

    def sub_weights(self, which, to1, to2, from1, from2):
        for i in range(0,6):
            j = 6-i
            self.layers[which].neurons[to1+i].weights[from1+i] = (2**j)*1
            self.layers[which].neurons[to1+i].weights[from2+i] = (2**j)*-1
            self.layers[which].neurons[to1+i].weights[-1] = -100
            self.layers[which+1].neurons[to2].weights[to1+i] = 1
            self.layers[which+1].neurons[to2].weights[-1] = -1

    def and_weights(self, which, to, from1, from2):
        self.layers[which].neurons[to].weights[from1] = 1
        self.layers[which].neurons[to].weights[from2] = 1
```

```
        self.layers[which].neurons[to].weights[-1] = 1

    def sum_weights(self, which, to, from1, from2, from3):
        for i in range(0,6):
            self.layers[which].neurons[to+i].weights[from1+i] = 1
            self.layers[which].neurons[to+i].weights[from2+i] = 1
            self.layers[which].neurons[to+i].weights[from3+i] = 1
            self.layers[which].neurons[to+i].weights[-1] = 100

    def fwd_weights(self, which, to, from_):
        self.layers[which].neurons[to].weights[from_] = 1

    def evaluate(self, inputs):
        for layer in self.layers:
            outputs = []
            for neuron in layer.neurons:
                total = neuron.compute(inputs) + neuron.weights[-1]*bias
                if(neuron.weights[-1] == 100): outputs.append(total+100)
                elif(neuron.weights[-1] == -100): outputs.append(total-100)
                else: outputs.append(self.sigmoid(total))
            inputs = outputs
        return outputs

    def sigmoid(self, activation):
        if activation < 0.5: return 0
        else: return 1

    def __str__(self):
        return '\n'.join([str(i+1)+": "+str(layer) for i,layer in enumerate(self.layers)])

def nim(piles):
    nim_network = NeuralNetwork([18,72,72,72,72,72,72,72,72,72,72,72,9])
    #print(str(nim_network))

    turn = False

    while(piles[0] > 0 or piles[1] > 0 or piles[2] > 0):
        print("Pile sizes: "+str(piles[0])+", "+str(piles[1])+", "+str(piles[2])+".")

        if(turn):
            move = [int(x) for x in input("Human turn | provide your move \'amount,index\': ").split(","
                ↪ )]
            selection = move[1]
            remaining = piles[selection]-move[0]
            print("Human took "+str(move[0])+" from pile "+str(move[1])+"!")
            piles[selection] = remaining
        else:
            state = to_bin(piles[0])
            state += to_bin(piles[1])
            state += to_bin(piles[2])
            move = nim_network.evaluate(state)
            selection = which(move[:3])
            if(selection == -2):
                selection = 0
                remaining = 0
            elif(selection == -1): break
            else: remaining = from_bin(move[3:])
            print("Computer took "+str(piles[selection]-remaining)+" from pile "+str(selection)+"!")
            piles[selection] = remaining
```

```
        turn = not turn

    if(turn): print("Computer won!")
    else: print("Human won!")

def to_bin(pile):
    binstr = "{0:06b}".format(pile)
    return [int(x) for x in binstr]

def from_bin(pile):
    binstr = ''
    for bit in pile:
        if(bit >= 1): bit = 1
        else: bit = 0
        binstr += str(bit)
    return(int(binstr, 2))

def which(selection):
    if(sum(selection)==1):
        if(selection[0]==1): return 0
        elif(selection[1]==1): return 1
        elif(selection[2]==1): return 2
    elif(sum(selection)==0):
        print("Computer has no remaining moves.")
        return -1
    else:
        print("Computer has 3 available moves.")
        return -2

piles = [int(sys.argv[1]),int(sys.argv[2]),int(sys.argv[3])]
nim(piles)
```