Andrew Simonetta
04/06/2020
CIS400M001 HW4

Learning Classification System on Neural Network

The assignment was to train a Neural Network with a Learning Classification System and compare the results with the Swarm Algorithm, Genetic Algorithm, Evolution Algorithm and the use of back propagation method. I started this process on the Google cloud where I used one of their GPU. I used the neural network from the previous assignment. It has an input layer, a hidden layer, and an output layer. It functions with a binary classifier where weights and correct samples are randomized.

I implemented my LCS with a population of 5 where each of their weight layers were randomized, assessed and tracked throughout the training process. The policies that were identified were found through a repetitious cycle on my part. I leveraged the ability of the numpy array to slice and index the layers in different patterns. The policy became how I incremented the slices in different patterns. I first started off with rather straightforward patterns, such as increment a range on each layer by some constant like 0.01. The learning part of this policy system was based on applying a particular policy to the whole population and determining how many improved. If the policy affected above 60% of the populace in a positive way it stayed, anything below was replaced. This offered consistent results across the population.

Just to mention a few notes about the structure of my program. I stored the particles, the weights, in a Numpy array with their loss values and when I needed to perform a function I iterated over them. I set up several helper functions to print, set, or get weights based on what I needed. Through each repetition of training I applied the policy to each member of the population, checked to see if it improved them, if so, applied it, and tracked the statistics for further fine tuning of my algorithm. I tracked effectiveness, or how many of the populace it affected, and the degree of change, the overall increase in fitness. I mainly used effectiveness in my application, but if expanded upon I would consider the degree of change more closely.

This was the first neural network where I could truly envision many improvements. I developed two ideas that would work in tandem with one another for great effect, but didn't have the time to implement it. The core idea was to incorporate a traits system that would cause each individual to explore the map in a unique complex testable pattern. Each individual would have a list of traits and their weights would be processed through each and their location would be updated. I had a fair list of traits, but some include Strong: or the ability to persist through negative change in a particular direction to achieve greater results, Intelligence: the ability to mimic for this phase the best trait amongst the populace, Agility: varying degrees of increments applied to find improvement, and many others with explorative properties.

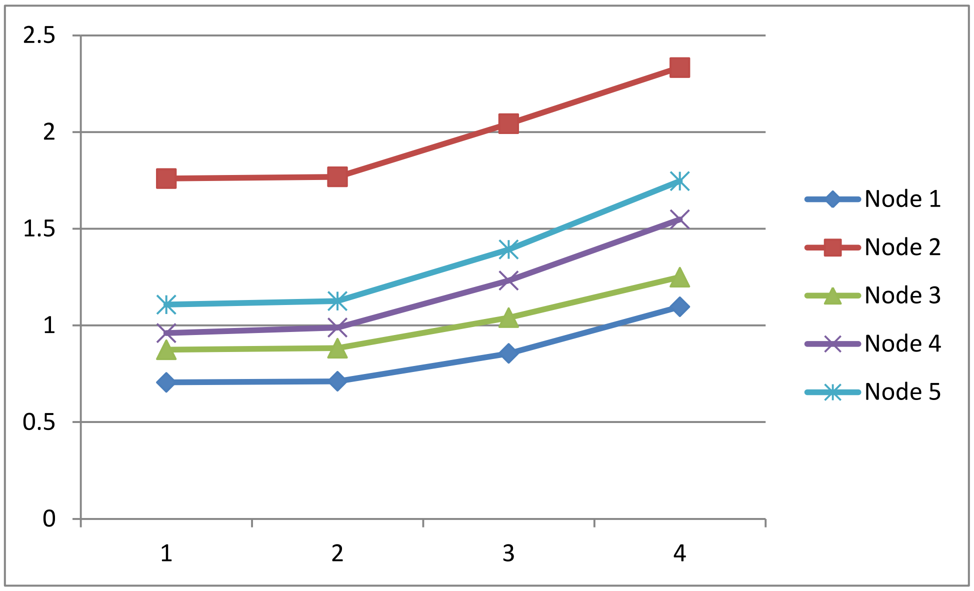
The results of my application were defined by steady improvement. Through each policy cycle a majority of the population improved and, as well as, the effectiveness of my policies. On average 70% of the population increased in fitness and by around a factor of 0.001. On my last cycle I increased the degree of improvement , on average, by 12%. There were still plenty of patterns I could apply to the weights. If I could automate the process of policy creation I could of had a profound effect, I believe. I just didn't have time to implement a function to do so. In the end I would be able to automatically create and swap policies.

In comparison to other learning applications I believe policy based to be superior. The way I was able to conceptualize the policy based learning classification system allows for more intricate patterns of traversing the loss map. By utilizing numpy arrays and vectorization the limit would really be processing power. If you think about adhering multiple policies together with different indices and skips you could probably development some complex mathematical models to identify overarching themes and localized pattern. With the ability to remove policies that begin to underperform and inject new ideas the system would truly be a living breathing craft of exploring on the vast expanse of the loss map.

In consideration of GAs, they are more linear in application. You can mix and match chromosomes and create new offspring, but it doesn't have the depth to truly wander freely through the map such as LCS. It's restrictive within its structural component. EA is almost a part of LCS in the fact that you want to continually update policies and swap them out to evolve its traversal based on where it is on the map. They share a fluidity of traversal which is certainly superior trait as I've found. Where Swarm seems to just be a more remedial policy which relies more on luck then effective pattern identification.

Graph and Data attached

Node 1	0.7056	0.7115	0.8555	1.0965
Node 2	1.7593	1.7684	2.043	2.333
Node 3	0.8744	0.8823	1.0405	1.2495
Node 4	0.9605	0.9892	1.232	1.5486
Node 5	1.1074	1.126	1.3928	1.7461



```
########## Entity 0  Accuracy:  0.7115098237991333 ##########
########## Entity 1  Accuracy:  1.7684202194213867 ##########
########## Entity 2  Accuracy:  0.8823484182357788 ##########
########## Entity 3  Accuracy:  0.9892172813415527 ##########
########## Entity 4  Accuracy:  1.1260185241699219 ##########

policy ('input', 0, 3, 1, 2, 5, 1, 0.1)

1  imp to  1.7702338695526123
Population: 5 (0.2, 0.001813650131225586)

policy ('hidden', 0, 7, 2, 7, 15, 2, -0.01)

1  imp to  1.7703330516815186

4  imp to  1.1261570453643799
Population: 5 (0.4, 0.0002377033233642578)
policy ('output', 0, 18, 2, 0, 1, 1, -0.1)

0  imp to  0.8408842086791992
1  imp to  2.0386288166046143
```

```
policy ('hidden', 0, 15, 3, 0, 15, 1, -0.01)
Population: 5 (0.0, 0)
policy ('hidden', 0, 15, 2, 16, 31, 3, -0.1)
4  imp to  1.416395664215088
Population: 5 (0.2, 0.0234979391098022
policy ('output', 0, 18, 2, 0, 1, 1, -0.1)
0  imp to
1.0627262592315674
1  imp to  2.320910692214966
2  imp to
1.2186930179595947
3  imp to
1.4819858074188232
4  imp to
1.6960504055023193
Population: 5 (1.0, 1.1927917003631592
policy ('input', 3, 9, 1, 7, 12, 1, 0.1)
0  imp to
1.0667250156402588
1  imp to  2.325803518295288
```

```
2  imp to  1.0242979526519775

3  imp to  1.192244291305542
4  imp to  1.3682823181152344
Population: 5 (1.0, 0.9847719669342041)
policy ('input', 3, 9, 1, 7, 12, 1, 0.1)

0  imp to  0.8422830700874329
1  imp to  2.043051242828369

2  imp to  1.0334694385528564

Population: 5 (0.6, 0.014992773532867432)
policy ('hidden', 4, 14, 2, 0, 15, 3, 0.1)

0  imp to  0.8555434942245483

2  imp to  1.0405272245407104

3  imp to  1.2320568561553955

4  imp to  1.3928977251052856

Population: 5 (0.8, 0.08474618196487427)

policy ('output', 0, 18, 3, 0, 1, 1, 0.15)
Population: 5 (0.0, 0)



########## Entity 0  Accuracy:  0.8555434942245483 ##########
########## Entity 1  Accuracy:  2.043051242828369
##########

########## Entity 2  Accuracy:  1.0405272245407104 ##########

########## Entity 3  Accuracy:  1.2320568561553955 ##########
########## Entity 4  Accuracy:  1.3928977251052856 ##########
```

```
2  imp to
1.2291746139526367
3  imp to
1.4844167232513428
Population: 5 (0.8, 0.0218040943145751
policy ('hidden', 4, 14, 2, 0, 15, 3, 0.1)
0  imp to  1.093242883682251
2  imp to
1.2465147972106934
3  imp to  1.541395664215088
4  imp to
1.7307615280151367
Population: 5 (0.8,
0.13554811477661133)
policy ('input', 0, 7, 3, 0, 15, 2, 0.15)
0  imp to
1.0965790748596191
1  imp to
2.3330881595611572
2  imp to
1.2495111227035522
3  imp to
1.5486396551132202
4  imp to
1.7461868524551392
Population: 5 (1.0,
0.03628647327423096)

########## Entity 0  Accuracy:  1.096579
##########
########## Entity 1  Accuracy:  2.333088
##########
########## Entity 2  Accuracy:  1.24951
##########
########## Entity 3  Accuracy:  1.548639
##########
########## Entity 4  Accuracy:  1.746186
##########
```