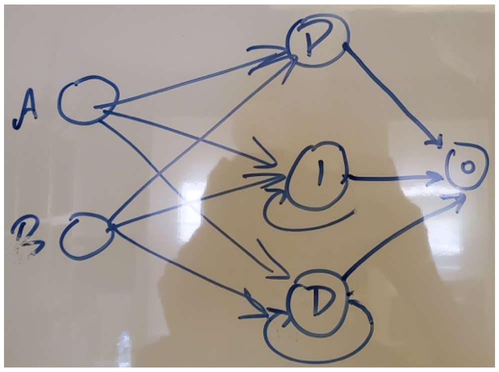
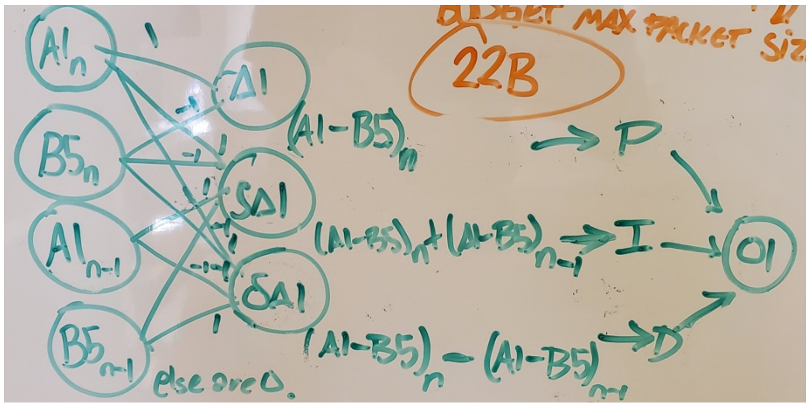
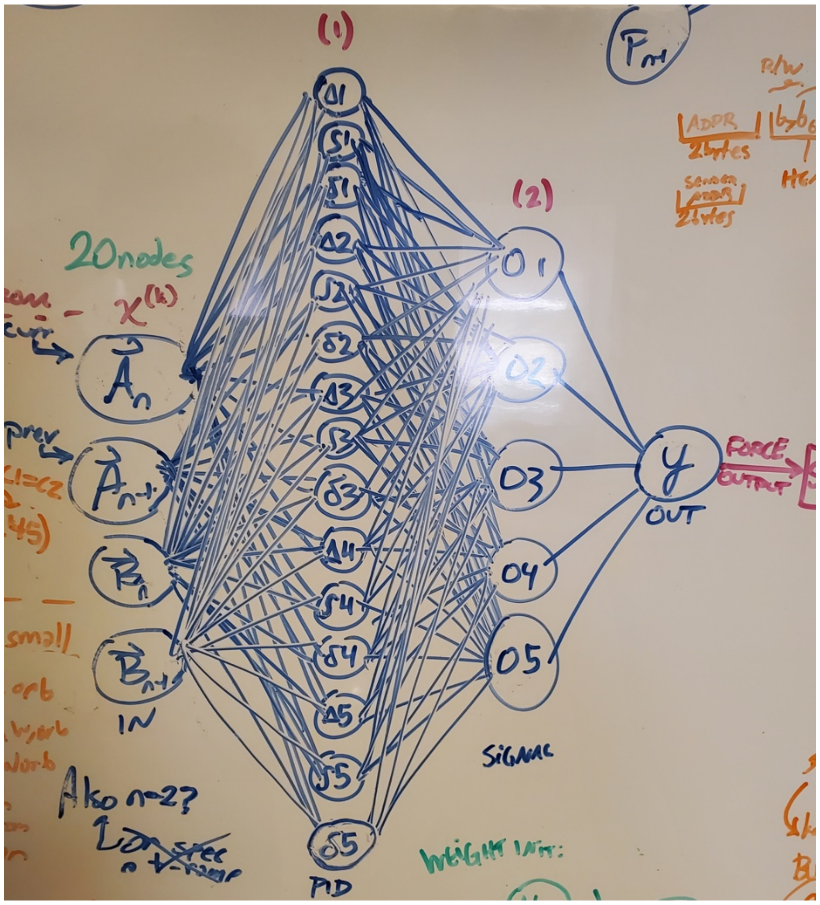
1. Function written.
2. An basic strategy would involve monitoring the distance to the walls above and below the flyer, the difference between these two will be the input to the controller. When the flyer gets closer to the upper wall than the lower wall, the flyer should thrust downwards, and when it's closer to the upper wall, it should thrust upwards (a P-controller). This strategy can be improved by reducing the commanded thrust if it would cause a significant sudden change to the input (a P-D controller). However, this strategy is prone to leaving a steady state error, which can be eliminated by adding a thrust component which is proportional to the integral of this error (a P-I-D controller).
   A downside to this approach, however, is that there is still information given which is being left unused. Namely, there are a 5 sample points given for both the top and bottom walls. An ideal strategy would use the above P-I-D technique to balance the flyer between all these points. While 5 separate P-I-D loops could be set up, this sort of control task with a high dimensionality on the sensor inputs is perfect for a neural network. So, a neural network was devised (detailed in part 3) whose architecture is capable of emulating 5 concurrent P-I-D loops. One problem with this approach, however, is that since the gradient for the environment is unknowable, a simple back-propagation technique won't work for training the network; so, a genetic algorithm can be employed instead.
3. A basic P-I-D algorithm would work as follows:
   $$F_{out} = k_p\left(d_{top} - d_{bottom}\right) + k_d \frac{d}{dt}\left[d_{top} - d_{bottom}\right] + k_i \int \left(d_{top} - d_{bottom}\right) dt$$
   One possible architecture for a neural network which would replicate this behavior would be:
   

   Herein, to produce a behavior of $P = k_p(A - B)$, the weights for the P neuron would be $+k_p$ for the A neuron and $-k_p$ for the B neuron. However, simple neural network algorithms deal with layers in a progressive manner and thus don't handle neurons who have their own state as an input. This can be resolved by passing in the previous state of each neuron in the input layer as shown below.



Thus, the integral neuron would be $I = k_i((A_n - B_n) + (A_{n-1} - B_{n-1}))$ and $D = k_i\big((A_n - B_n) - (A_{n-1} - B_{n-1})\big)$ for the derivative neuron.

With this in mind, the following architecture would be able to emulate 5 PID loops.



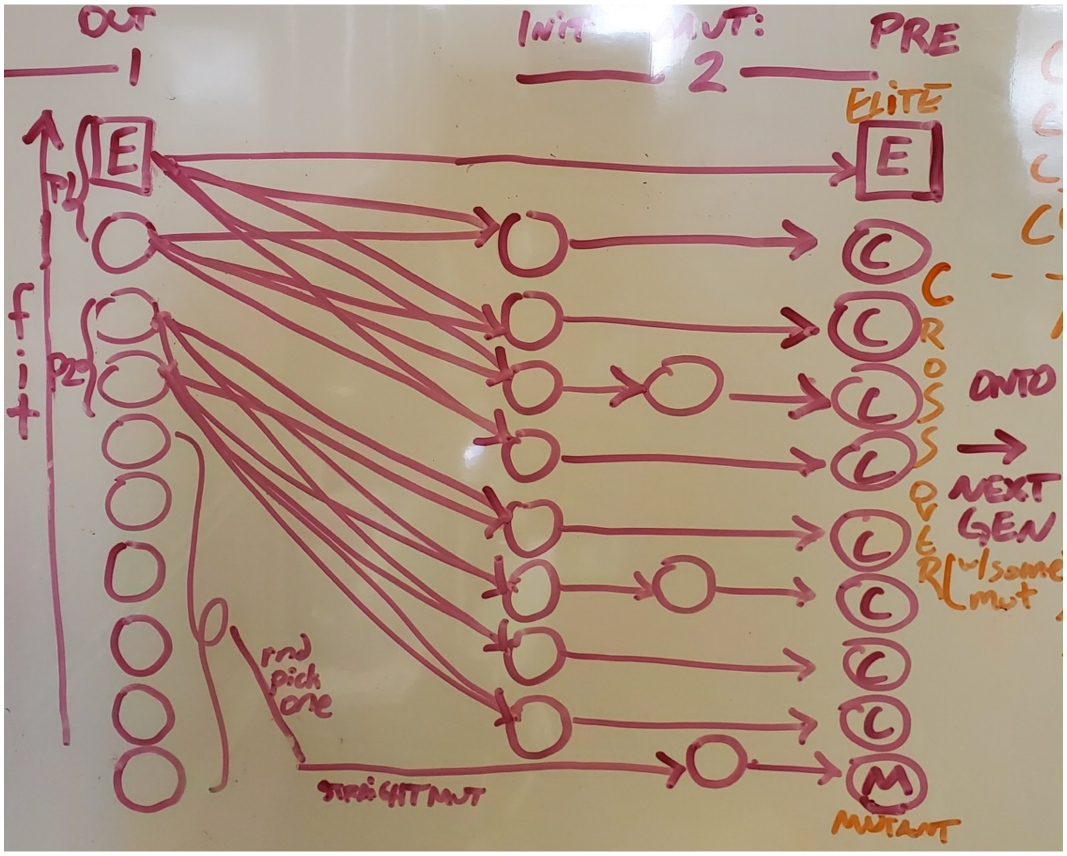
Here there are 20 input neurons (4 sets of 5 points) linked to 15 neurons in layer 1 (3 neurons comprising a P-I-D set for each set of 4 points representing $A_n, B_n,\ A_{n-1}, B_{n-1}$ as detailed above), then 5 neurons in layer 2 (1 for each P-I-D set, representing the signal from that control loop, and finally 1 output neuron.

4. And 6. A naïve version of the controller was implemented using the above architecture where each k value was set to one and the output was scaled so that the max output would be 2. This completely failed and only produced small output signals. However, after multiplying the networks output signal by a gain of 1,000,000, the network achieved a score of 458. From there

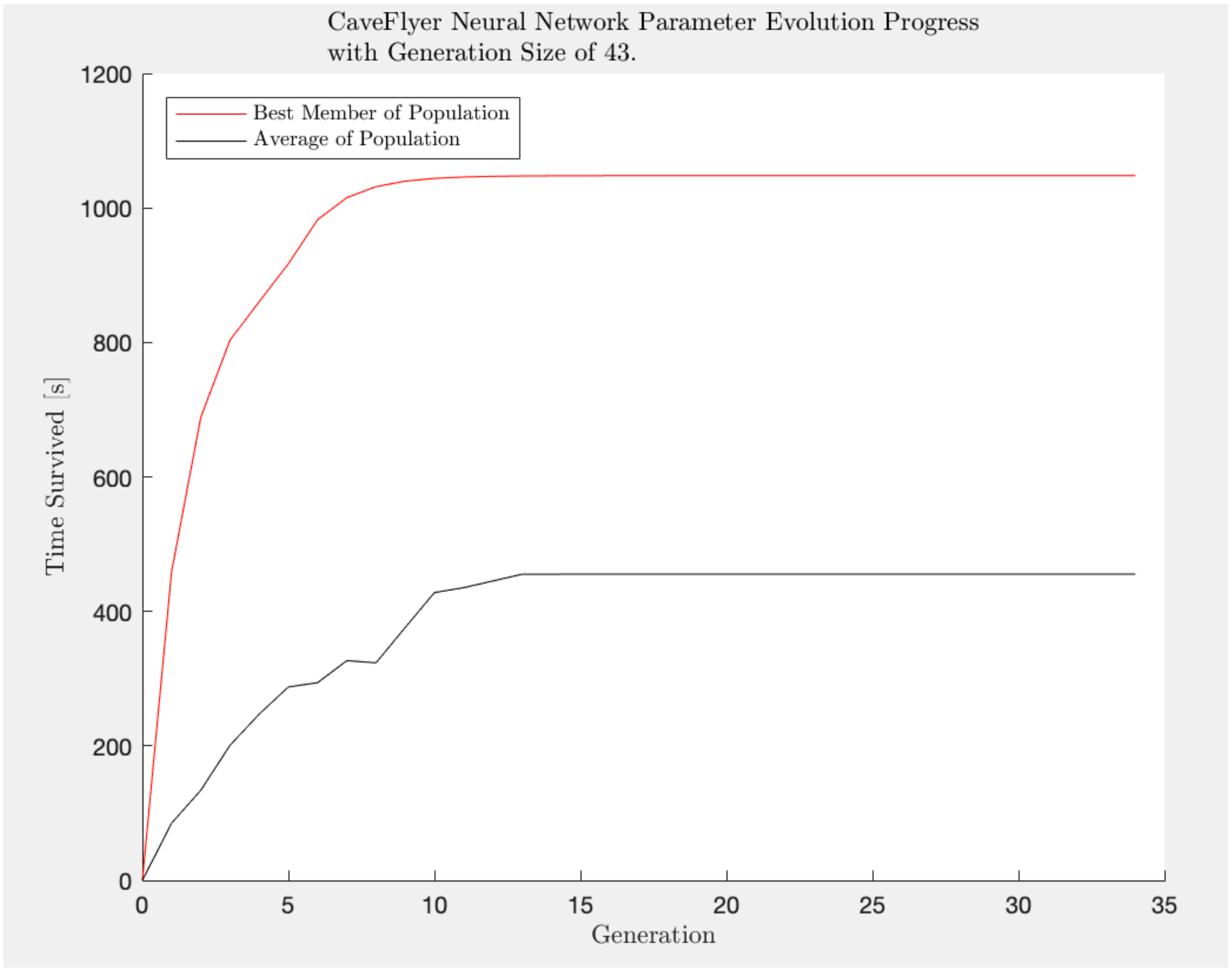
a genetic algorithm was used (the function evolveNN). This algorithm is initialized with a population of N flyers, sets of weights and biases for the network. All of these are then run on levels 5 and 10 and their scores are averaged. The top 10% of flyers are allowed to remain onto the next generation, from there all but a small number of the next generation is filled with sets of 4 flyers bred from pairs of the most successful flyers of the previous generation. In this case breeding means producing four children – C1,C2,C3,C4 – from two parents – P1, P2 – where the weights and biases are computed as follows:

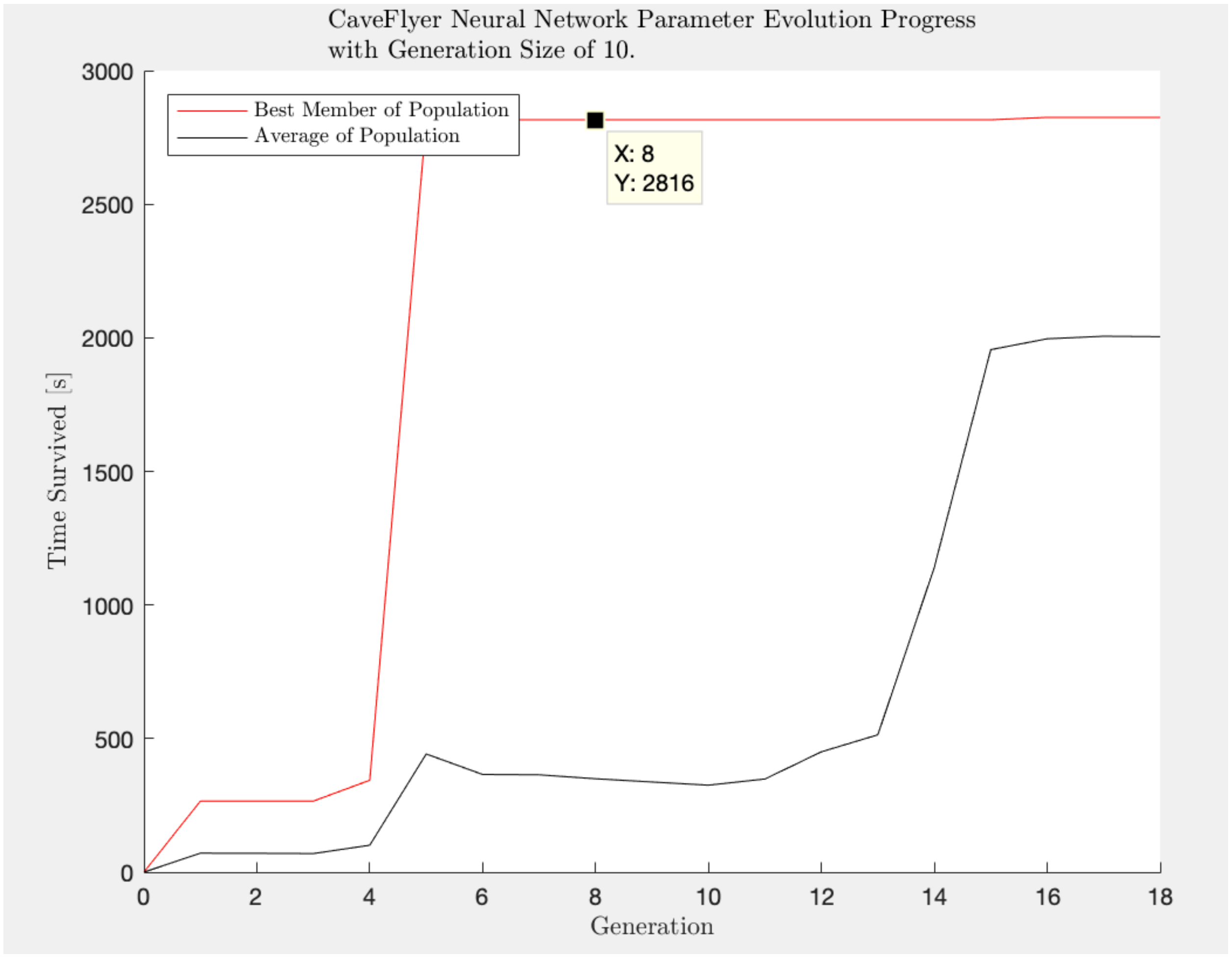
$$C1 = \alpha P1 + (1-\alpha)P2$$
$$C2 = \alpha P2 + (1-\alpha)P1$$
$$C3 = \frac{\alpha}{2}P1 + \left(1-\frac{\alpha}{2}\right)P2$$
$$C4 = \frac{\alpha}{2}P2 + \left(1-\frac{\alpha}{2}\right)P1$$

Where P1 is the better performing parent and α is chosen at random in each crossover set from between 0.2 and 0.45 (note that 0.5 means that C1 and C2 would be identical).
The remainder of the next generation is then filled with flyers composed of the most successful flyers of the prior generation which were not bred, having been first subjected to random mutations that consist of either swapping weights around, shuffling weights, or replacing weights with weights randomly selected from a reasonable normal distribution based upon. The hypothesized ideal network architecture proposed in question 3 (mean of 1, std of 75).



Training graphs:


CaveFlyer Neural Network Parameter Evolution Progress
with Generation Size of 43.
Best Member of Population
Average of Population
Time Survived [s]
Generation
0
200
400
600
800
1000
1200
0
5
10
15
20
25
30
35


CaveFlyer Neural Network Parameter Evolution Progress
with Generation Size of 10.
Best Member of Population
Average of Population
X: 8
Y: 2816
Time Survived [s]
Generation
0
500
1000
1500
2000
2500
3000
0
2
4
6
8
10
12
14
16
18

5. Done.