Experiments on Mixed Genetic and Gradient-Driven Evolution for Machine Learning

Gavin Cooke, Luca de Alfaro

Introduction:

In classic machine learning, a model is improved by using gradient descent to adapt its parameters and decrease the loss of the model's predictions. This process is repeated for a number of iterations until the model reaches a satisfactory performance on its training data. In this experiment we investigated whether any advantages exist for combining this standard approach with recombination steps inspired by genes in nature and evolving a population of models. Specifically, we were curious whether this approach worked well, how much variety there would be in the parameters of the final models, and whether it provided models with any resistance to adversarial attacks.

Method:

The general idea of this genetic recombination is that, instead of training a single model, a population of models is trained using gradient descent for a few epochs and then a new population is generated by recombining the parameters of the models from the original population. Then these new models are trained for a few epochs and the process is repeated for as many generations as required.

An important concern of an experiment like this is how one defines a gene. In our experiments we tried two different implementations of this. The first was to define each layer of the model as its own gene. Thus, a three layer model such as what we used would have three genes. One for the input, middle, and output layer. Referred to as l1, l2, and l3 respectively. In the recombination step, two models, the parents, are selected from the final population and a new model, the child, is created. For each of the child's layers, it receives a copy of the corresponding layer from one of its parents, chosen randomly. We refer to this first approach as horizontal genes, picturing the division between the genes as horizontal lines between the layers.

Our second implementation functioned in a similar manner. The input and output layers are handled the same as above, the difference comes from how the middle layer is handled. Here the middle layer is split into two pieces. Instead of the l1, l2, and l3 above, here there would be l1, l2a, l2b, and l3. Both halves of the middle layer take the full output of the l1 as input and their output is concatenated together to form the input for l3. During recombination l2a and l2b are treated as separate genes, the child model could receive one from one parent and the other from

the other. We refer to this approach as vertical genes, picturing the division in the middle layer as a vertical split.

Another important detail is how to select models from the final population for recombination. In genetic algorithms this is done by defining a fitness function to rank the members of the population in terms of some metric. In our experiment the fitness of each model was determined by its accuracy on the test data, and then the best half of the population was used to perform the recombination step to get the next population.

Experimental Results:

To get out results we ran experiments for both the horizontal gene implementation as well as the vertical one. For each of these, we simulated a population of twenty models over six generations where each model was trained for one epoch every generation. For comparison, we used a single model with the same number of parameters trained traditionally for six epochs. The dataset we used for this was MNIST.

In terms of accuracy, the difference between the models of the final population and the traditional model was negligible. All of them generally ended with accuracies in the 96%-97% range. The main difference is that the population simulation takes significantly longer to run, due to training twenty models in parallel.

For the question of variety, we measured this using the cosine similarities of each gene across the population. At the beginning, this similarity value is near zero, since each model is a unique initialization, but as the simulation progresses they become more similar overall due to learning from the same data and possibly inheriting from the same parent models. What we found was that after six generations, the variety in the models often disappeared completely and the entire population was nearly identical in terms of cosine similarity. At six generations, generally the population divided itself into a few somewhat different groups, similar to alleles that occur in genes in nature. For some reason, this behavior was most pronounced in the output layer of the models.

None of the models in the populations we simulated showed any resistance to adversarial attacks on their own, the accuracy of each model was generally less than 1%. We also tried testing attacks developed against one model from the final population against the others, the models showed little resistance to this kind of attack either, accuracies were generally around 1%-3%. For a final experiment, we tried developing an attack against a child of the final population and then testing it against other children of the final population. In a population where the cosine similarities had not completely converged yet, this got accuracies around 5%-10%. In a population where the cosine similarities had converged, the accuracies were around 4%.

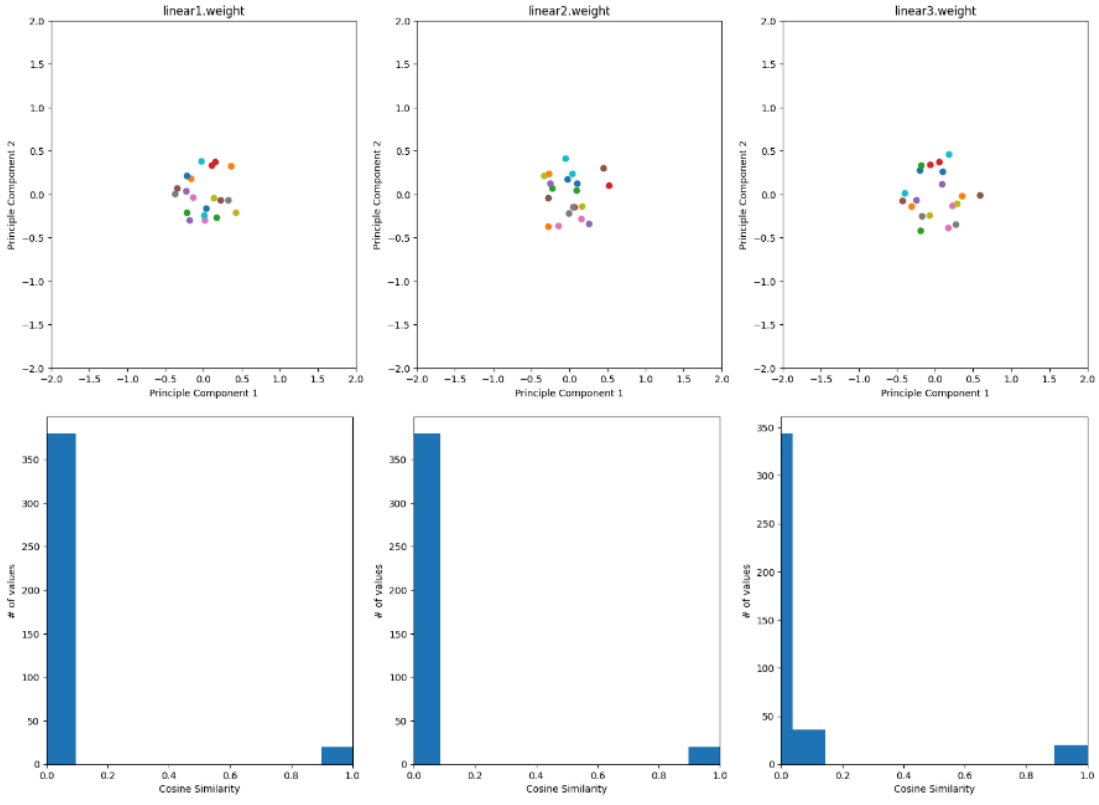
Conclusion:

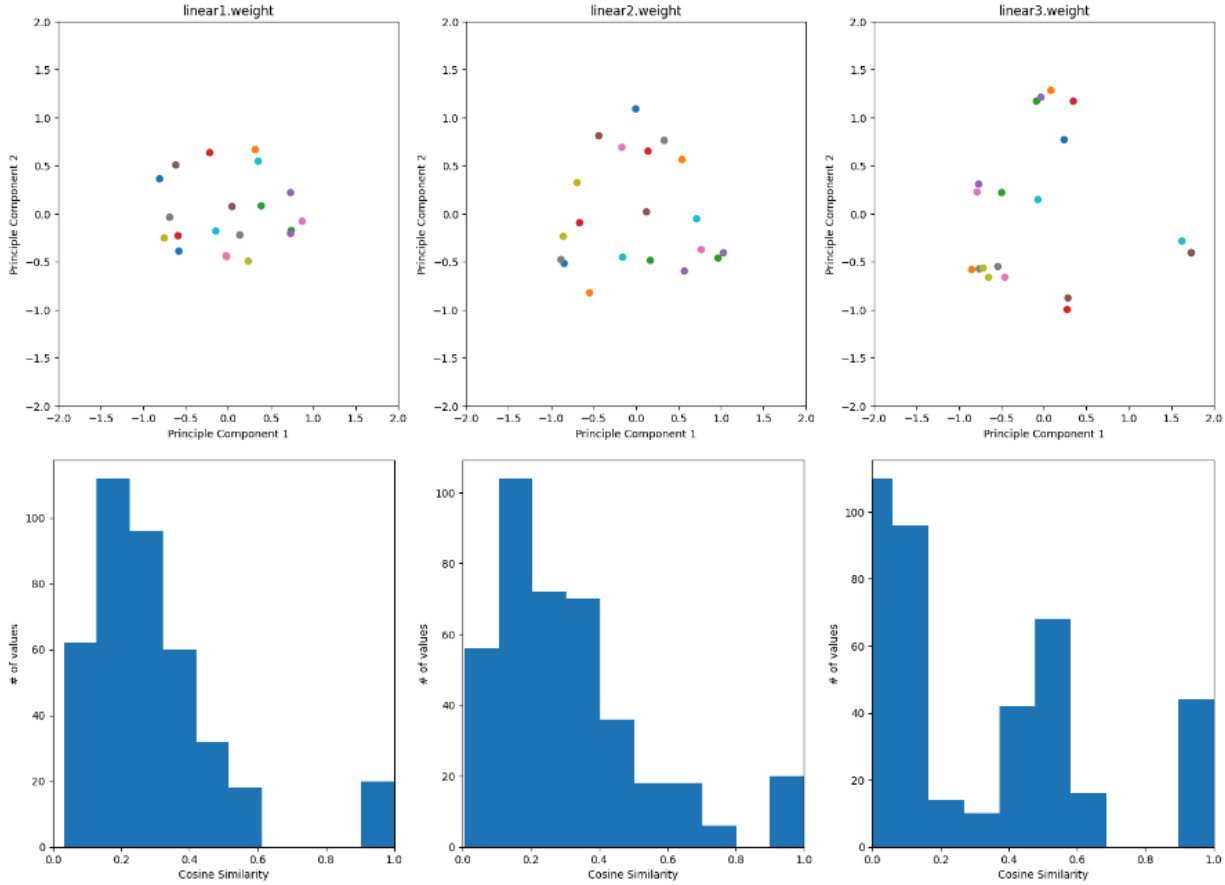
In conclusion, simply adding genetic crossing like we did in this experiment does not provide any meaningful advantages to a machine learning model compared to training it in a traditional manner using gradient descent. A potential cause of this is that there is not enough incentive for diversity in our implementation, since we only use accuracy to determine fitness. In the adversarial attacks test, the population that retained some diversity did perform slightly better than the others. To test this more thoroughly, there would need to be some motivation given for diversity in the parameter values of the final population. Perhaps by changing the fitness function to model advantages given to genetic hybrids in nature, or by adding something such as mutations to offset the loss in diversity which occurs in each generation.

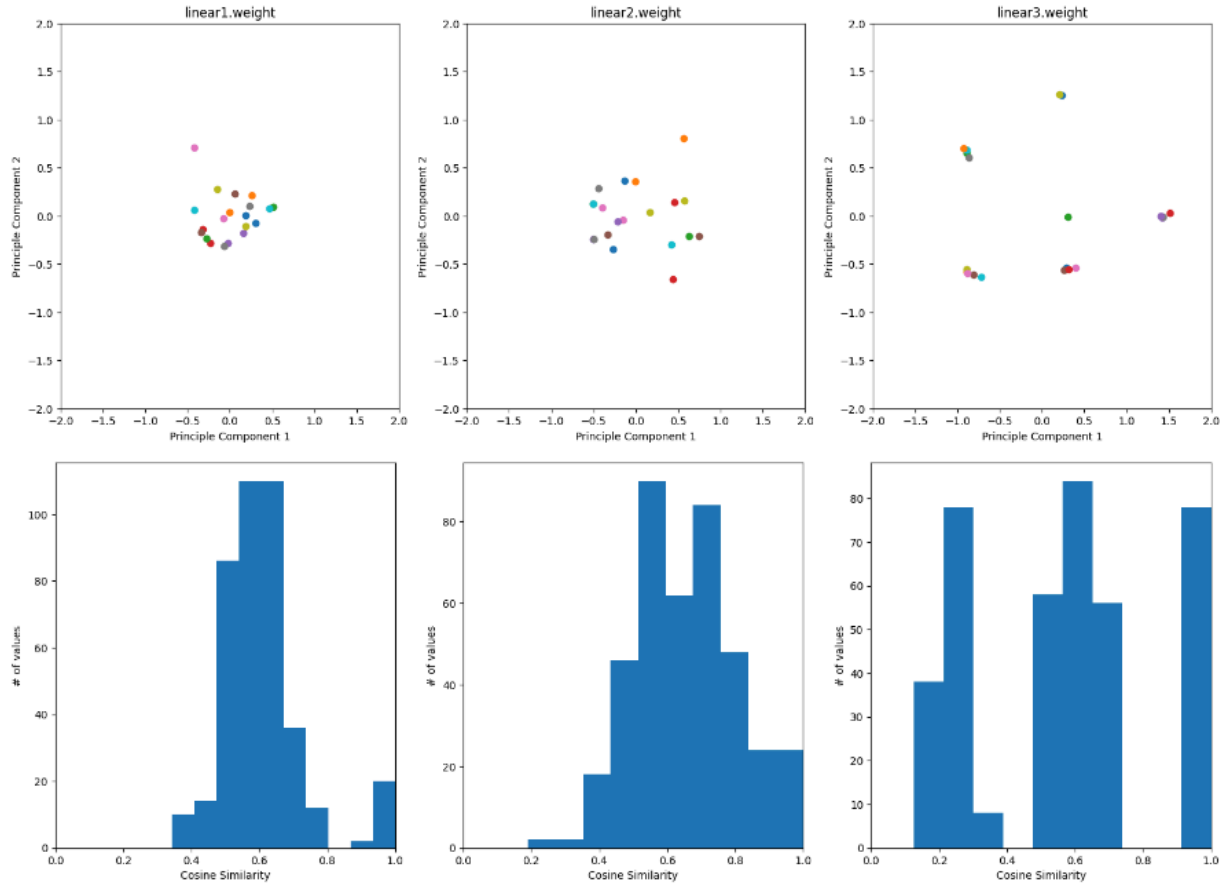
Cosine Similarity Graphs:

Below are some of the plots from the last simulation run as of the time of this report. Each set of two rows of graphs represents a generation in terms of the cosine similarity between the layers in the population of models. The first row of graphs in each generation plots the cosine similarities in a scatter plot by performing principal component analysis on the 20 by 20 cosine similarity matrix for each layer. The second row of graphs consists of histograms which count the frequency of each cosine similarity value in the matrix mentioned above.

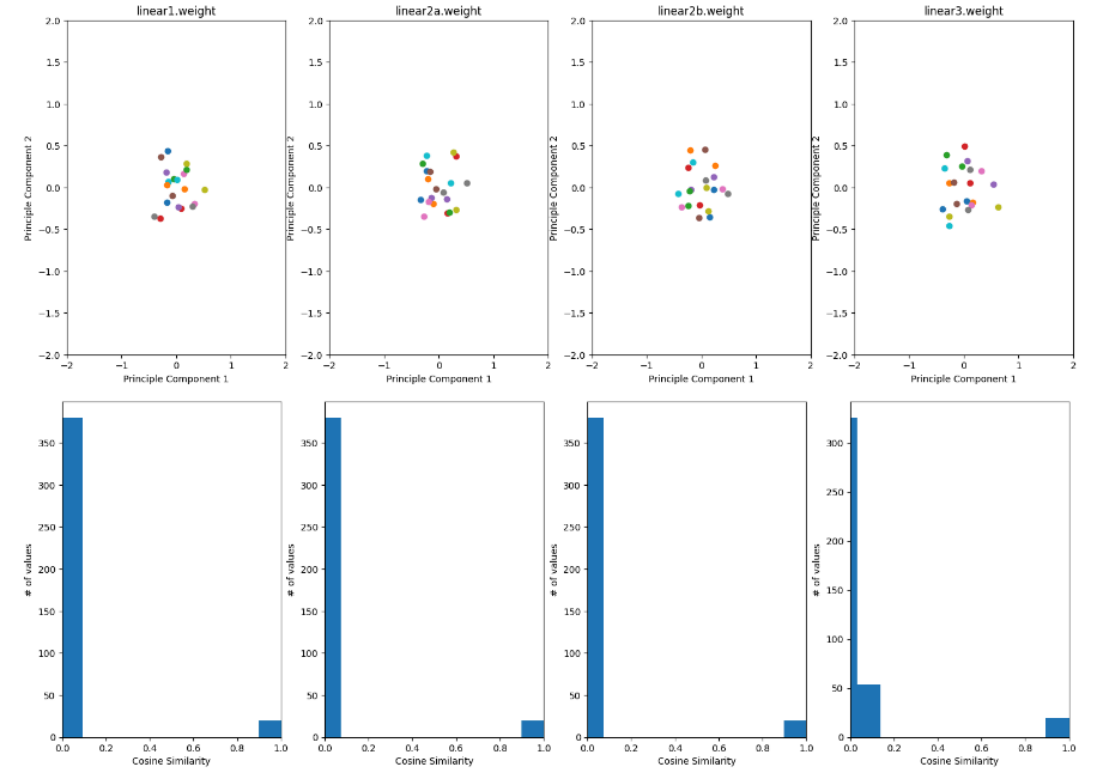
This first set of three generations represents generations zero, three, and six from the last simulation run of the horizontal gene population. In this simulation the population did not converge and there remained some diversity as of the final generation.

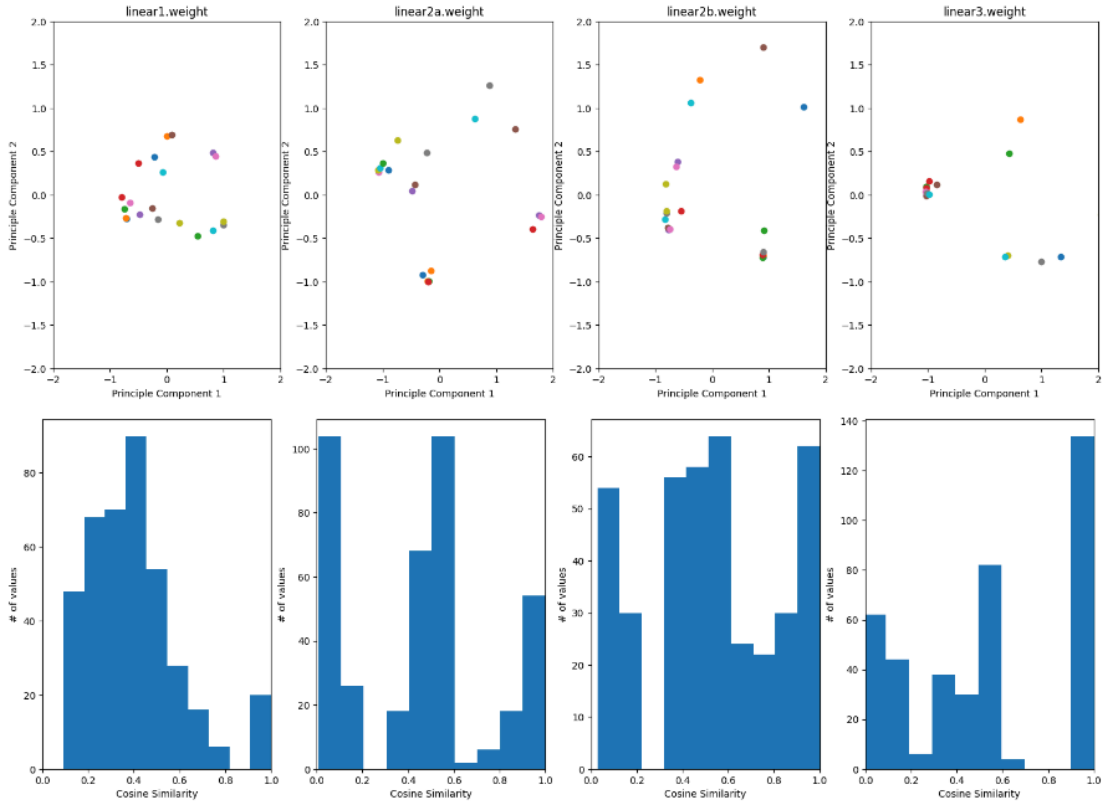


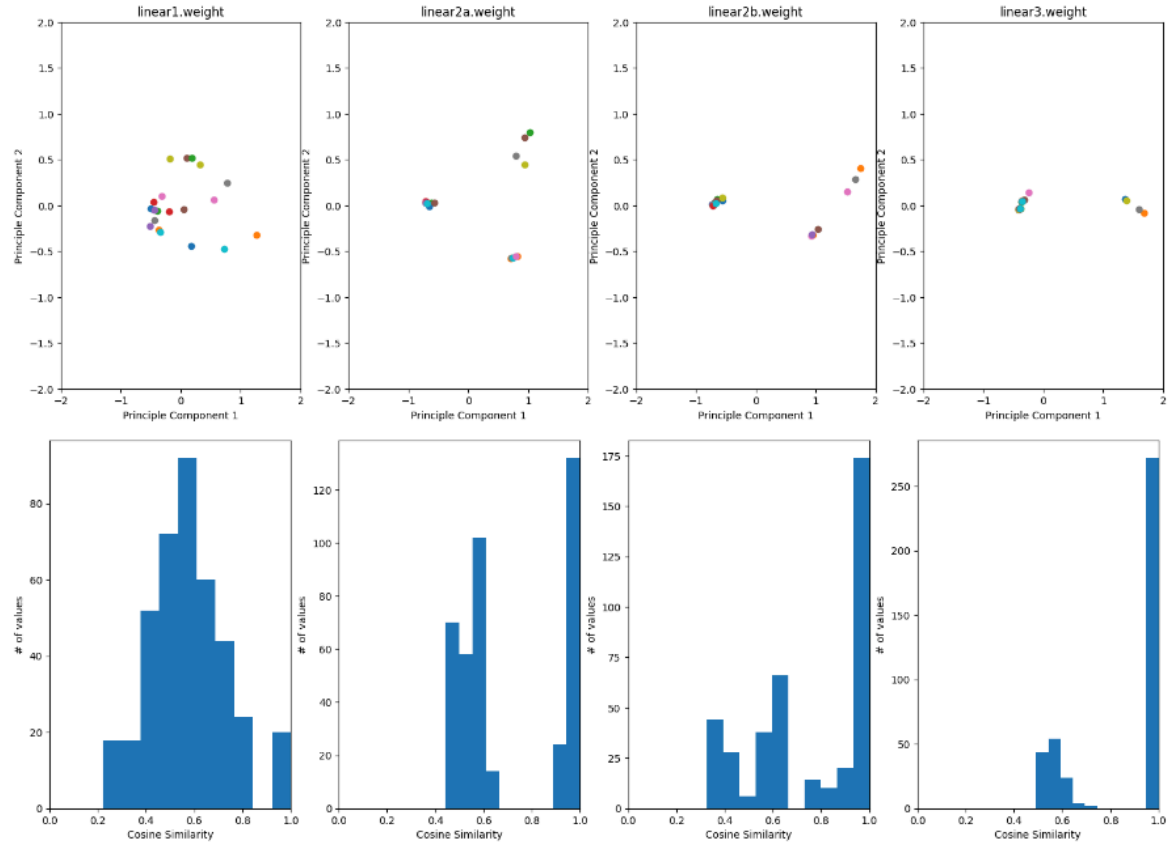

linear1.weight
linear2.weight
linear3.weight
Principle Component 2
Principle Component 1
# of values
Cosine Similarity

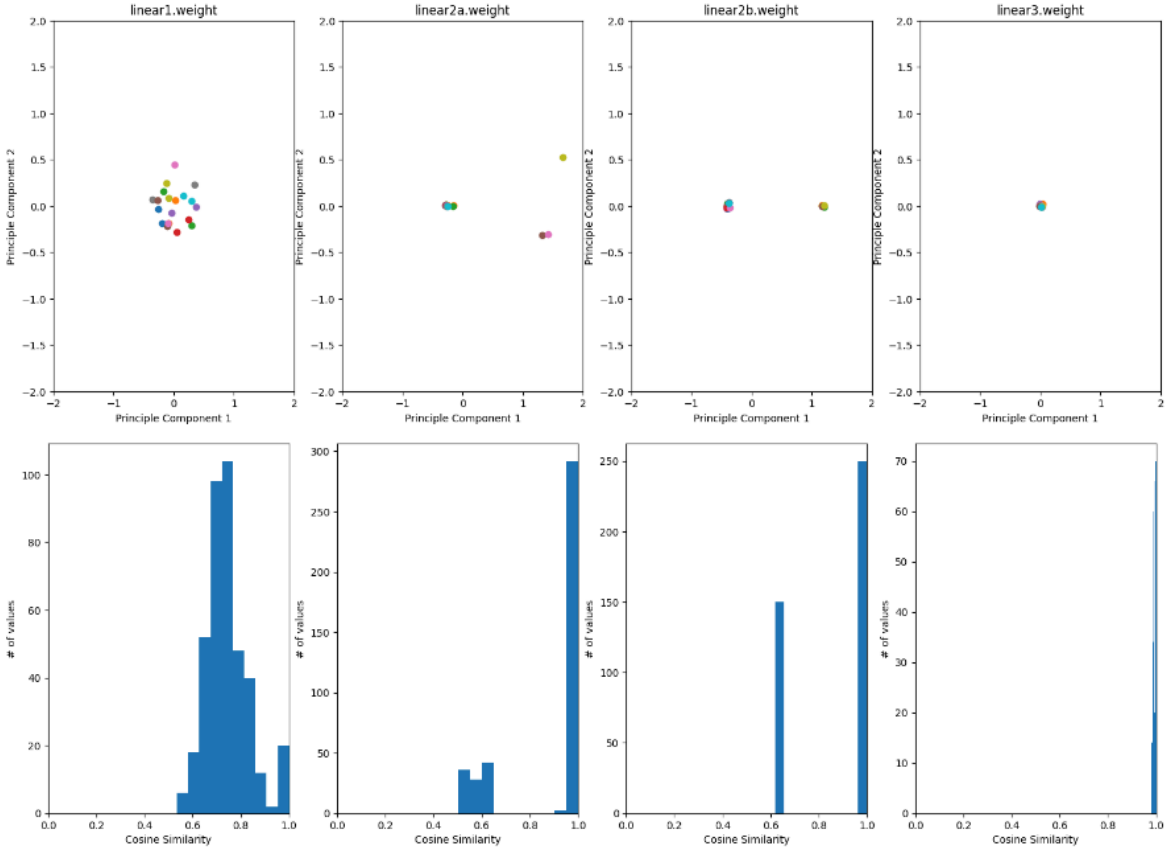

linear1.weight
linear2.weight
linear3.weight
Principle Component 2
Principle Component 1
# of values
Cosine Similarity

This next set of three generations represents generations zero, four, five, and six from the most recent simulation of the vertical gene population as of the time of this report. In this simulation the population converged to have nearly identical parameters for the third layer, and very similar values for the other layers as well.






linear1.weight
linear2a.weight
linear2b.weight
linear3.weight
Principle Component 2
Principle Component 1
# of values
Cosine Similarity


linear1.weight
linear2a.weight
linear2b.weight
linear3.weight
Principle Component 2
Principle Component 1
# of values
Cosine Similarity

Resources:

The code for this project, as well as a the complete collection of graphs and other output from the last experiment run using it, can be found in the Jupyter Notebook in the following Github repository: https://github.com/GavinCooke2000/Genetic_ML.