Development of Modularity in Neural Networks

Daniel H. Cox
Harvard University, IACS, CS299R Spring 2021, final project.

Introduction

There is considerable evidence that many neural systems are composed of modular components that are specialized for processing specific aspects of sensory stimuli [1]. In primary visual cortex (V1), for example, columns of neurons respond specifically to stimuli of certain orientation and from a certain eye [2]. And areas around V1 are specialized to respond to distinct attributes of a visual seen [3]. How do such modular neural systems arise? While much of the organization of the brain is undoubtedly based in genetics, there is an extensive literature that indicates that during development the visual system is highly plastic. Its structural connectivity and functional efficacy are heavily influenced by the electrical activity in the network and the visual experience of the developing human [4, 5]. Thus, is seems relevant to ask: do neural networks naturally segregate into modular systems as they are trained to understand the world?

Recently, in the context of evolution, we read papers by Parter, Kashtan and Alon [6-8] in which they suggested —and with model systems demonstrated— that modular systems naturally evolve when systems are evolving toward more than one goal. And, when only a single goal is present, modularity does not arise. Further, they suggested that modularity becomes more pronounced when the series of goals the system evolves toward are themselves modular. That is, the later goals can be thought of as a composite of the earlier goals. This suggests that perhaps neural networks also develop modularity naturally when they are trained on shifting goals but not on static goals. And perhaps this modularity is more profound if the goals they are trained on are also modular.

To test these hypotheses here I have performed a series of experiments with artificial neural networks, altering their structure and the way they are trained, and measuring their modularity. The results of these experiments are described below.

Experimental Design

Creating a series of artificial feedforward multilayered neural networks.

I created artificial feedforward neural networks using the python package *keras* with a *tensor-flow* backend. Their general structure was modeled after those of Filan et al. (2020) [9], who recently demonstrated a difference in modularity between trained and untrained neural networks. Each network was composed of an input layer with 756 nodes to represent each pixel of a 28x28 image, 4 hidden layers whose number of nodes varied, and an output layer that contained 4 nodes for classification. A summary of a typical network as generated by *keras* is shown below.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 256)               200960
_________________________________________________________________
dense_1 (Dense)              (None, 256)               65792
_________________________________________________________________
dense_2 (Dense)              (None, 256)               65792
_________________________________________________________________
dense_3 (Dense)              (None, 256)               65792
_________________________________________________________________
dense_4 (Dense)              (None, 4)                 1028
=================================================================
Total params: 399,364
Trainable params: 399,364
Non-trainable params: 0
```

Figure 1. Summary generated by *keras* of a typical neural network used for these experiments. Each line represents a layer of the network. The first is the input: then there are four hidden layers with 256 nodes each and an output layer with four nodes.

Training the networks to classify images with a either a single goal or a series of shifting goals.

For all experiments I used the well-known MNIST dataset, which is composed of hand-written numerals 0 through 9. An example of some MNIST numerals is shown below.

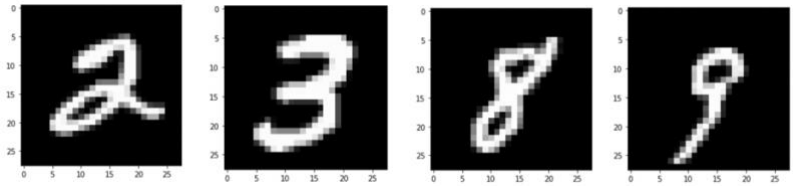

Figure 2. Series of MNIST handwritten numerals

The above series (2, 3, 8, 9) is one I used in my experiments. With this series, training on a single goal was done as follows. A subset of 40,000 MNIST images containing images of just the numerals 2, 3, 8 or 9 was generated. A neural network was then trained via gradient descent to determine if a given input is of the numeral 2, 3, 8 or 9. In this case the network would be trained on all numerals simultaneously. That is, for each input the network would have four classification choices. The network would be trained for 10 epochs (cycles through the data), and then its accuracy would be determined on a separate test set of the same general composition as the training set. Then, the modularity of the network would be measured using the n-cut metric (see below).

I refer to the above type of training as simultaneous, meaning the network was trying to classify all numerals at once. I also refer to this experiment as one performed with a single goal, meaning the task the network was performing did not change during training. This is as opposed to other experiments where the network was trained on the same series of numerals but sequentially and therefore with shifting goals. This was done as follows. First the series (2, 3, 8, 9) would be

randomly reordered to for example (3, 8, 9, 2). Then, the network would first be trained to distinguish a 3 from all other digits, and then after five epochs, an 8 from all other digits, and then after another five epochs a 9 from all other digits, and then finally a 2. So, here the series of goals would be (3, 8, 9, 2). This series would be presented as described but in random sequence 3 times, and finally the network's accuracy and modularity would be determined.

Note, however, in order to train the network sequentially it was necessary to use during training only a single output node whose value determined the classification. But to assess the networks accuracy on all four numerals, four output nodes were required. To accommodate these two demands, the accuracy of a network trained sequentially was assessed as follows. After training, the single output node was replaced with four output nodes, and then all weights in the network but those to the output nodes were frozen. Then, the weights to the four new nodes were determined by additional training for 10 epochs. The resulting network's accuracy could then be determined as usual. Fortunately, this procedure is not expected to influence to any significant extent the network's modularity, as only weights to 4 out of a total of 1812 nodes are altered. Cost functions for training were either categorical or binary cross-entropy.

Notice above that the set of goals used (2, 3, 8, 9) are modular in that the numeral 3 shares certain structural components with the other numerals —the top curve. And the numeral 8 shares many components with the numeral 3, and the numeral 9 shares components with 8, 3, and 2. Thus, a network learning to distinguish between these numerals might evolve modules to identify common structural components. Conversely, the other series I used in my experiments (2, 4, 6, 7) is not composed of modular goals, as these numerals share little structurally one to the next. Thus, by using these two series, I was able to assess the influence of modular as oppose to non-modular goals on network modularity after training.

Assessing the performances of the neural networks.

All networks' performances were assessed based on the percent of images they correctly classified on a separate test dataset composed of randomly selected MNIST numerals of the appropriate values.

Computing network modularity

To interpret my experiments, I needed some mechanism for quantifying the degree of modularity of a network. In the literature several methods have been discussed[9-11]. Most notably, Filan et al (2020)[9] recently developed a measure of neural network modularity that involves the use of a spectral graph clustering algorithm and yields what they refer to as a network's n-cut, a measure inversely proportional to modularity. With this metric a network is more modular when neurons in the same partition share strong synaptic weights with each other and weak synaptic weights with neurons in different partitions. I used this metric to quantify modularity.

To understand how a network's n-cut is calculated, it is first necessary to state some definitions (after Filan et al. 2020[9]).

1) A neural network can be represented by a graph *G* composed of vertices *V*, each representing a neuron, and edges *E*, each representing a feedforward connection. We

consider these edges undirected. A network with N neurons will then have N vertices and a variable number of edges M, dependent on the network's structure.

2) With such a graph we can then define an N x N adjacency matrix A, whose elements A_{ij} are the values of the weights of the network from neuron i to neuron j.
3) Also, we define the degree of each neuron d_i as the sum of the weights associated with edges connected to and from neuron i and the degree matrix D as the diagonal matrix with the degrees of each neuron on the diagonal.
4) Also, we define a subset of V as X, and a set of disjoint subsets as (X_1, X_2, ...X_i). Such a set represents a partitioning.
5) The complement of any X_i is referred to as $\overline{X}_i$ and is equal to all X in a partitioning but X_i.
6) Next, we define the volume of a given X_i, $vol(X_i)$, as the sum of the degrees of the neurons in X_i.
7) And we define the weight between two partitions of G, $W(X_1, X_2)$, as the sum over all A_{ij} representing edges between elements of X_1 and elements of X_2.

With these definitions we can then define the n-cut of a particular partitioning of G as

$$\text{n-cut}(X_1, \ldots, X_k) := \sum_{i=1}^{k} W(X_i, \bar{X}_i)/\text{vol}(X_i)$$

So, the n-cut is a sum over partitions of the strengths of connections between the partitions, each normalized by partition volume. The smaller this sum is, the fewer strong connections between partitions relative to within partitions, and thus we say that given two graphs, the one with the smaller n-cut is more modular.

Having the n-cut as a measure of modularity, it was then necessary to choose a method to partition a given network in a way that minimizes the number and strength of the connections between partitions. This was done with spectral graph clustering which in brief is done as follows.

1) An adjacency matrix A for the graph is constructed.
2) Its Laplacian L is calculated as $D - A$, where D is the degree matrix described above, and A is the adjacency matrix.
3) An Eigen decomposition is performed on L.
4) Some number of clusters k to partition the graph into is chosen.
5) The matrix U composed of the first k eigenvectors of L with the smallest eigenvalues is created.
6) The values of the elements of these eigenvectors are used to perform k-means clustering of the nodes of the network in k dimensional space.

These are the methods I used to partition each network after training and to calculate each network's modularity. Following Filan et al. 2020, k was set equal to 4.

Code

All calculations were done with python. Spectral clustering was performed with the *sci-kit-learn* package and its function *SpectralClustering()* using the arpack eigensolver and adjacency matrices constructed separately. Code to calculate n-cuts was developed specifically for this project. To facilitate these experiments, I created a python package called 'n_cut' that contains two modules. n_cut.py, which contains code for making all calculations involved in network analysis, and MNIST_helper.py, which contains code for manipulating MNIST data. The experiments themselves and their analyses were done in Jupyter notebooks, 24 in all. They can be viewed in the public Github repo at: https://github.com/dcox01/CS229R_Project_Cox

Varying network pruning and architecture.

In addition to examining the influence that training method —simultaneous vs sequential— and goals type —modular vs non-modular— have on network accuracy and n-cut, I also examined the extent to which two other factors affect these metrics: the size of a network and the degree to which it is pruned during training.

It is well established that in the brain connections are typically formed in excess and then pruned back as the connections mature [12]. A similar procedure can be done in an artificial neural network where during training, connections with small weights are gradually removed [13]. Indeed, Filan et al (2020) found that such a procedure enhances the development of network modularity when networks are trained in the standard manner. I have implemented a pruning procedure for the networks I have built with *keras* and varied the final sparcity of the networks from 0 to 40 to 80%, meaning 0, 40 or 80% of the network's weights are gradually reduced to zero during the course of training.

Also, it seemed to me possible that network modularity might become more or less significant as the size of the network changes. To examine this possibility, I varied the number of nodes in each of the four hidden layers in my experiments from of 64 to 128 to 256.

Results

Training method, pruning and size affect modularity

In total there were four variables associated with my experiments: the size of the network's hidden layers, the degree of pruning, the method of training, and the modularity of the goal sequence. Varying all of these factors I performed in total 20 experiments with each individual condition repeated 10 times. Listed in Fig. 3 below are the n-cut and accuracy values for 10 networks built and trained in the same way for a single condition. The name of each run in the leftmost column indicates that these results are for a series of networks that contained 4 hidden layers, 256 neurons per layer. No pruning was done. The networks were trained sequentially, and they were trained on the series (2, 4, 6, 7). Nineteen similar tables were generated.

	N_cut	Accuracy
2467_seq_prune_0_4_256_run0	2.21569	0.9124
2467_seq_prune_0_4_256_run1	2.12253	0.88975
2467_seq_prune_0_4_256_run2	2.15474	0.959225
2467_seq_prune_0_4_256_run3	2.17256	0.94665
2467_seq_prune_0_4_256_run4	2.16549	0.957025
2467_seq_prune_0_4_256_run5	2.1385	0.937225
2467_seq_prune_0_4_256_run6	2.09562	0.95765
2467_seq_prune_0_4_256_run7	2.16955	0.93655
2467_seq_prune_0_4_256_run8	2.15247	0.945175
2467_seq_prune_0_4_256_run9	2.06714	0.953675

Figure 3. Result table for 10 experiments done under a single condition.

.

From tables like that of Fig. 3 four condensed tables were generated representing the mean results of the individual trials. These tables are shown below in Fig 4. Fig. 4A&B show results for experiments done with the goal sequence (2, 4, 6, 7) and trained either simultaneously (Fig. 4A) or sequentially (Fig. 4B). Fig. 4C&D shows similar results but for experiments done with the modular goal sequence (2, 3, 8, 9).

A

	Series	Train_type	H_layers	H_nodes	prunning	Mean_N_cut	N_cut_std	N_cut_ste	Accuracy
0	2467	sim	4	64	0.0	2.182	0.028	0.009	0.986715
1	2467	sim	4	128	0.0	2.290	0.045	0.014	0.987572
2	2467	sim	4	256	0.0	2.370	0.046	0.014	0.988113
3	2467	sim	4	64	0.4	2.070	0.028	0.009	0.985058
4	2467	sim	4	128	0.4	2.191	0.019	0.006	0.987405
5	2467	sim	4	256	0.4	2.307	0.017	0.005	0.987582
7	2467	sim	4	64	0.8	1.764	0.033	0.010	0.978952
8	2467	sim	4	128	0.8	1.964	0.041	0.013	0.982345
9	2467	sim	4	256	0.8	2.126	0.025	0.008	0.985000

B

	Series	Train_type	H_layers	H_nodes	prunning	Mean_N_cut	N_cut_std	N_cut_ste	Accuracy
0	2467	seq	4	64	0.0	1.918	0.068	0.021	0.937587
1	2467	seq	4	128	0.0	2.029	0.026	0.008	0.948008
2	2467	seq	4	256	0.0	2.145	0.042	0.013	0.939533
3	2467	seq	4	64	0.4	1.767	0.059	0.019	0.958443
4	2467	seq	4	128	0.4	1.929	0.050	0.016	0.963833
5	2467	seq	4	256	0.4	2.048	0.025	0.008	0.966095
7	2467	seq	4	64	0.8	1.622	0.061	0.019	0.965780
8	2467	seq	4	128	0.8	1.794	0.035	0.011	0.970645
9	2467	seq	4	256	0.8	1.970	0.038	0.012	0.975987

C

	Series	Train_type	H_layers	H_nodes	prunning	Mean_N_cut	N_cut_std	N_cut_ste	Accuracy
0	2389	sim	4	64	0.0	2.200	0.043	0.014	0.984540
1	2389	sim	4	128	0.0	2.306	0.038	0.012	0.983975
2	2389	sim	4	256	0.0	2.343	0.041	0.013	0.984218
3	2389	sim	4	64	0.4	2.054	0.021	0.007	0.982012
4	2389	sim	4	128	0.4	2.192	0.020	0.006	0.983450
5	2389	sim	4	256	0.4	2.306	0.028	0.009	0.985335
7	2389	sim	4	64	0.8	1.735	0.050	0.016	0.974185
8	2389	sim	4	128	0.8	1.947	0.024	0.008	0.980563
9	2389	sim	4	256	0.8	2.121	0.021	0.007	0.982660

D

	Series	Train_type	H_layers	H_nodes	prunning	Mean_N_cut	N_cut_std	N_cut_ste	Accuracy
0	2389	seq	4	64	0.0	1.932	0.027	0.009	0.930065
1	2389	seq	4	128	0.0	2.026	0.058	0.018	0.902837
2	2389	seq	4	256	0.0	2.119	0.035	0.011	0.900710
3	2389	seq	4	64	0.4	1.799	0.063	0.020	0.933230
4	2389	seq	4	128	0.4	1.937	0.030	0.009	0.944160
5	2389	seq	4	256	0.4	2.065	0.022	0.007	0.949262
7	2389	seq	4	64	0.8	1.584	0.056	0.018	0.934917
8	2389	seq	4	128	0.8	1.834	0.040	0.013	0.954348
9	2389	seq	4	256	0.8	1.976	0.050	0.016	0.959985

Figure 4. Combined result for sets of experiments done with goal series (2,4,6,7), *A* and ***B,*** or goal series (2,3,8,9), *C* and *D*. A and C report data for experiments where the networks were trained simultaneously, B and D where they were trained sequentially.

As shown in Fig. 5 below, these results are best interpreted graphically. Each panel reports mean n-cut as a function of hidden-layer size for different combinations of training series and training method. Each color represents a different degree of pruning. Viewed in this way it becomes apparent that under all conditions pruning enhances modularity. That is, as pruning was increased from 0 to 40 to 80%, regardless of the other conditions, the mean n-cut decreased, and often substantially. In Fig. 5A for example, among networks containing 64 nodes in their hidden layers, changing from no pruning (blue) to 80% (green) reduced the n-cut 19%. Note on these and all other plots error bars indicate standard error of the mean, and they are often smaller than the data markers. Thus, in general differences between points that do not overlap are statistically significant

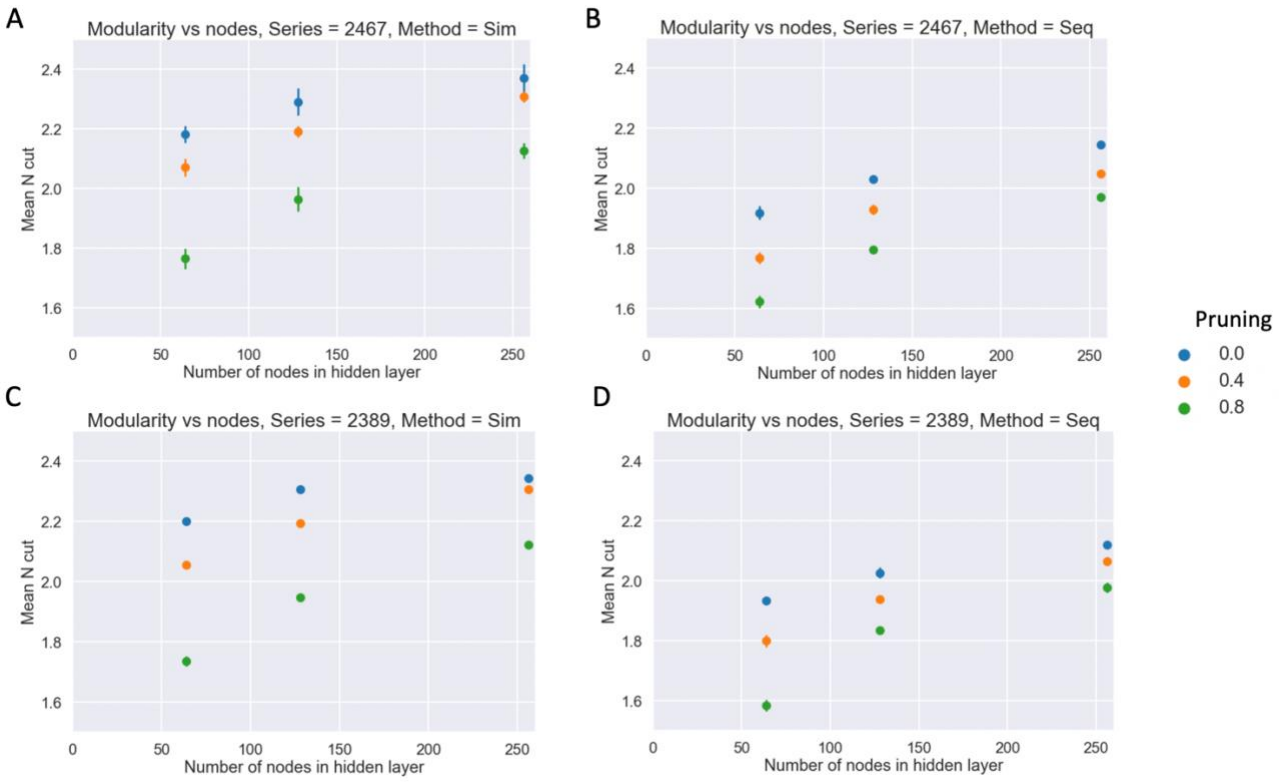

Figure 5. Plots are of Mean n-cut value from 10 experiments as a function of nodes in each hidden layer. Color indicates pruning level. In *A* and *B,* the goal series (2, 4, 6, 7) was used. In *B* and *C* the goal series (2, 3, 8, 9) was used. In *A* and *C* the networks were trained simultaneously, in *B* and D sequentially. Error bars = ste

Also, under all conditions, increasing the number of nodes in each hidden layer increased the n-cut in a monotonic way. The larger the network, all other things being equal, the less modular. This effect could also be quite pronounced. Again in Fig. 5A, among networks with 80% pruning (green), increasing the size of each hidden layer from 64 to 256 nodes increased the n-cut 20%.

Further, comparing the top graphs in Fig. 5 to the bottom, we see little difference in mean n-cut regardless of training method. Changing the training goals from (2, 4, 6, 7) to (2, 3, 8, 9) appears to have little or no effect on network modularity, even though to my eye the series (2, 3, 8, 9) is more modular than (2, 4, 6, 7).

And perhaps most importantly, comparing the graphs in Fig. 5 now left to right, it is also evident that training the networks sequentially, one numeral and then another and so on, does yield networks significantly more modular than those trained on the same series simultaneously. For example, for networks trained on the series (2, 3, 8, 9), with 40% pruning, and hidden layers containing 64 nodes, changing the method of training from sequential (Fig. 5D) to simultaneous (Fig. 5C), increased the mean n-cut from 1.799 to 2.054, 14.2%.

These last two affects can be seen more clearly if we just focus on a single level of pruning (Fig. 6 below). Here, all points are from networks with 80% pruning, and color now indicates the manner of training. As is evident for both series —(2, 3, 8, 9) and (2, 4, 6, 7)— regardless of network size, sequential training leads to more modular networks. The blue points are lower than the orange.

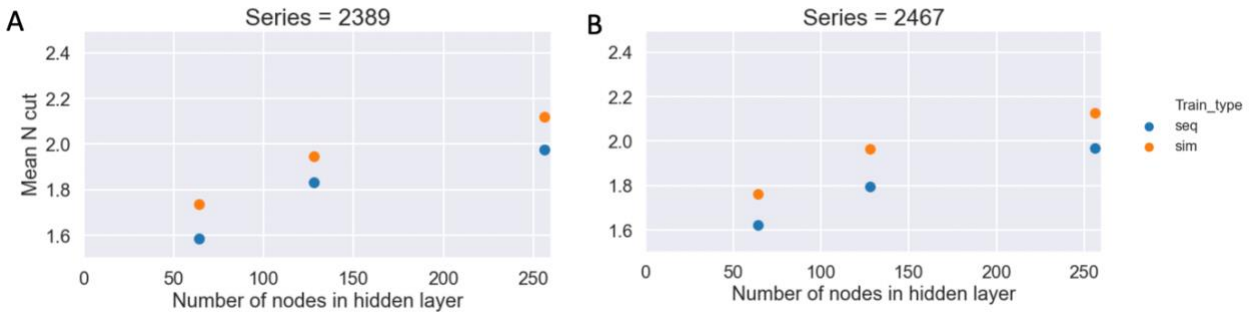

Figure 6. Plots are of Mean n-cut value from 10 experiments as a function of nodes in each hidden layer. Color indicates training method. In *A* the goal series (2, 3, 8, 9) was used. In *B* the goal series (2, 4, 6, 7) was used. All data are from networks with 80% pruning. Error bars = ste

.

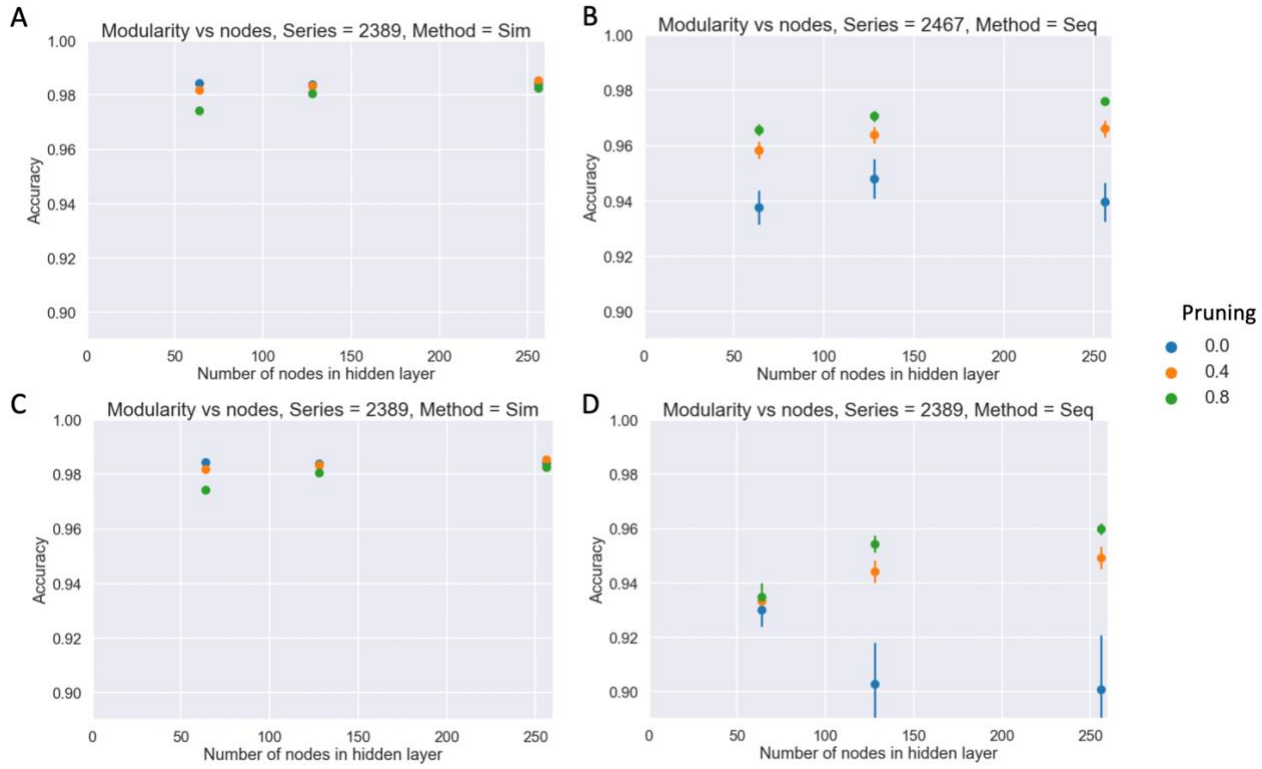

Figure 7. Plots are of mean accuracy on the test set from 10 experiments as a function of nodes in each hidden layer. Color indicates pruning level. In *A* and *B*, the goal series (2, 4, 6, 7) was used. In *B* and *C* the goal series (2, 3, 8, 9) was used. In *A* and *C* the networks were trained simultaneously, in *B* and D sequentially. Error bars = ste.

Training method, pruning and size also affect accuracy

It is also interesting to examine the effects of all these variables on network accuracy. This is shown in Fig. 7 above. First, from examining all plots it is clear that all networks were fairly accurate, classifying at least 90% of inputs correctly (see also tables in Fig. 4). But there were distinctions. First, the method of training clearly affects accuracy. Simultaneously trained networks were always more accurate than sequentially trained (compare panels left to right). Second, in simultaneously trained networks, pruning hurt accuracy —as one might expect when connections are being lost— however, remarkably the opposite was the case for networks trained sequentially. Here, pruning increased accuracy. For example, in networks with 256 nodes per hidden layer that were trained sequentially on the series (2, 4, 6, 7), network accuracy increased from 90% to 96% when pruning was increased from 0 to 80% (Fig. 7B, blue to green). Thus, pruning has opposite effects depending on training method.

Finally, while which series of goals was used did not show much effect on network modularity, regardless of training method (Figs. 5 and 6), it did affect accuracy. When networks were trained sequentially, accuracy was higher on the non-modular series (2, 4, 6, 7) than on the modular series (2, 3, 8, 9). When they were trained simultaneously, there was no difference. This is perhaps best seen in Fig. 8 below where plotted are points for networks with just 80% pruning, and now color indicates method of training. Here, it is immediately apparent that simultaneously trained networks are more accurate than sequentially trained ones (orange points are higher than blue), and that this effect is more pronounced with the modular goal series (2, 3, 8, 9) (Fig. 8A) than with the non-modular series (2, 4, 6, 7) (Fig. 8B).

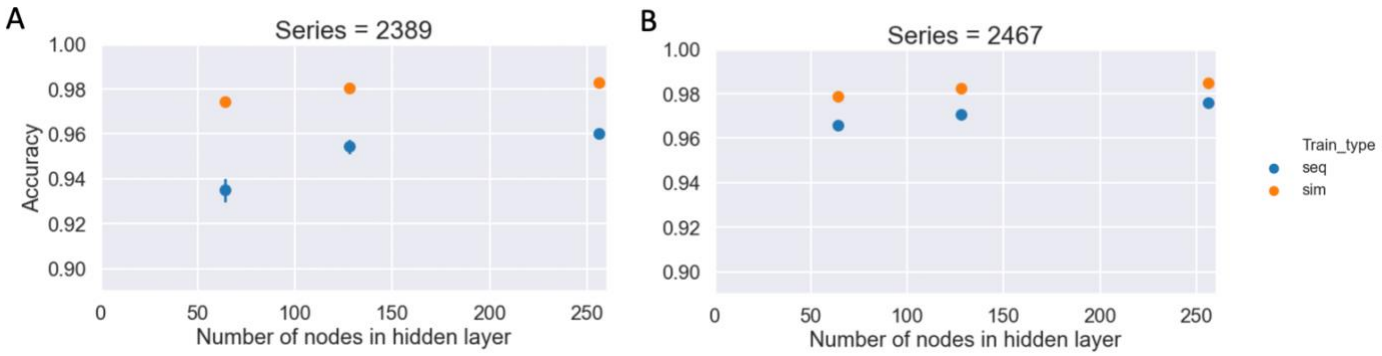

Figure 8. Plots are of mean accuracy value on the test set from 10 experiments as a function of nodes in each hidden layer. Color indicates training method. In *A* goal series (2, 3, 8, 9) was used. In *B* the goal series (2, 4, 6, 7) was used. All data are from networks with 80% pruning. Error bars = ste

Discussion

This study yielded several interesting conclusions. One is that pruning under all conditions enhances modularity. The origins of this effect are not known, but they are not unexpected. Filan et al 2020, found that in networks similar to the ones used here, trained networks were more modular than untrained and that pruning promoted modularity under both conditions [9]. Further, Clune et al 2013 found that networks evolved to detect images on a simulated retina became more modular when a cost was imposed for both inaccuracy and the number of connections, in effect when pruning was part of the evolutionary process [14]. Perhaps, pruning promotes modularity because it reduces the number of ways a series of goals can be met and therefore promotes reuse of common elements when faced with multiple goals.

Also, with regard to pruning, I observed that in networks trained sequentially, pruning enhanced accuracy. That is, pruned networks that had lost 80% of their connections were more accurate than those that remained intact during training. While this result is perhaps counter intuitive, it is not unprecedented. Clune et al 2013 found that networks trained sequentially on shifting goals with pruning were often more accurate than those trained without pruning. They concluded that pruning promotes modularity which in turn moves networks out of local minima with less-than-optimal accuracy[14]. Perhaps this is what is going on in my experiments as well. Similarly, Gerum et al 2020 found, that when they evolved neural networks to solve a maze, "random severing of connections (evolutionary pruning), without explicitly rewarding sparsity, lead to a general sparsification of the networks and a better generalization performance"[15]. Thus, pruning appears to increase both modularity and accuracy in sequentially trained networks. In my experiments, accuracy was always very high in simultaneously trained network (98%). Here, pruning did little harm or good. However, I did also observe that larger networks were generally less modular than smaller networks. This may be the converse of the effect of pruning. Larger networks may be less constrained to reuse parts when addressing shifting goals than are smaller networks.

The original hypotheses I set out to test with these experiments were based on the conclusions of Parter, Ashton and Alon [6-8] who worked with evolutionary models of digital circuits and tRNA molecules. Extrapolated to neural networks they may be stated as follows.

1) Modular neural networks will develop when the networks are trained on shifting goals, and less so when they are trained on fixed goals. In the context of my experiments this means that networks trained sequentially on either of my test series (2, 3, 8, 9) or (2, 4, 6, 7) will be more modular if the training is done sequentially.

2) The Modularity of neural networks trained on shifting goals will be greater, if those goals are themselves modular. In my experiments this means that networks trained sequentially on the goal series (2, 3, 8, 9) should be more modular than those trained on the series (2, 4, 6, 7).

Of these hypotheses, the first I have confirmed. As best shown in Fig 6, all other things being equal, networks develop with smaller n-cuts when trained sequentially as opposed to simultaneously, and it is tempting to conclude that this is due to pressure to reuse similar structures on different tasks when goals are shifting. Further, given that it is via shifting goals that biological

nervous systems develop —in the visual system, for example, one image after another— it may be that their modularity is due in part to something fundamental about the effects of this kind of training on the formation of synaptic connections and their subsequent pruning.

The second hypothesis listed above, however, was not confirmed. I observed no difference in network modularity when training networks sequentially on (2, 3, 8, 9) as opposed to (2, 4, 6, 7), a result that seems to suggest that modular goals do not lead more effectively to modular networks, and indeed in the context of artificial neural networks this may be true. However, I have defined the modular goals here subjectively, based on my opinions about what numerals share common substructures. Perhaps the neural networks see things differently. Perhaps they are focusing on other aspects of these numerals that my human eye is not drawn to. If I were to pursue these experiments further, I would want to see if networks trained on other types of images would yield the same results, and whether I could find some objective measure of common image substructures that I could use to test again hypothesis 2.

Finally, it is reasonable to ask what do these experiments, done with artificial neural networks, have to do with real biological neural networks? They share a common general structure, nodes connected by edges with synaptic weights of varying strength leading to some interpretable output. But real nervous systems contain a great many feedback connections that I have not included, and more concerning, real neural networks are not trained via gradient descent. The networks I have used were trained by following the gradient of the cost function for each task, typically categorical cross-entropy or binary cross-entropy. Conversely, a neuron in the brain has no notion of derivatives, so one might think that whatever guiding principles are followed by developing artificial neural networks —guided necessarily by derivatives— they may not be relevant to brain development. And it is hard to discount this concern completely. But one might also consider that following a gradient is just one mechanism of finding an optimum. And while biological nervous systems many not follow the gradient of their cost function with respect to their inputs. They very nearly must be doing something of the sort, as they must be comparing some output to some desired correct output learned by experience, and then adjusting themselves in such a way that the difference is smaller. That is, they must be trying to optimize some measurement based on feedback, and if the optimum has anything to do with modularity, then both real and artificial networks may reflect this fact regardless of the specifics of their training.

References

1. Zeki, S. and A. Bartels, *The autonomy of the visual systems and the modularity of conscious vision.* Phil. Trans.R. Soc. Lond 1998. **353**: p. 1911-1914.
2. Hubel, D.H. and T.N. Wiesel, *Early exploration of the visual cortex.* Neuron, 1998. **20**(3): p. 401-12.
3. Siu, C.R. and K.M. Murphy, *The development of human visual cortex and clinical implications.* Eye Brain, 2018. **10**: p. 25-36.
4. Berry, K.P. and E. Nedivi, *Experience-Dependent Structural Plasticity in the Visual System.* Annu Rev Vis Sci, 2016. **2**: p. 17-35.

5. Cramer, K.S. and M. Sur, *Activity-dependent remodeling of connections in the mammalian visual system.* Curr Opin Neurobiol, 1995. **5**(1): p. 106-11.
6. Kashtan, N. and U. Alon, *Spontaneous evolution of modularity and network motifs.* Proc Natl Acad Sci U S A, 2005. **102**(39): p. 13773-8.
7. Kashtan, N., E. Noor, and U. Alon, *Varying environments can speed up evolution.* Proc Natl Acad Sci U S A, 2007. **104**(34): p. 13711-6.
8. Parter, M., N. Kashtan, and U. Alon, *Facilitated variation: how evolution learns from past environments to generalize to new environments.* PLoS Comput Biol, 2008. **4**(11): p. e1000206.
9. Filan, D., et al., *Pruned Neural Networks Are Surprisingly Modular.* arXiv, 2020. **2003.04881v4**: p. 1 - 24.
10. Davis, B., et al., *NIF:A framework for quantifying neural information flow in deep networks.* arXiv:, 2019., 2019. **1901.08557**.
11. Watanabe, C., K. Hiramatsu, and K. Kashino, *Understanding community structure in layered neural networks.* Neurocomputing, 2019. **367**: p. 84–102.
12. Low, L.K. and H.J. Cheng, *Axon pruning: an essential step underlying the developmental plasticity of neuronal connections.* Philos Trans R Soc Lond B Biol Sci, 2006. **361**(1473): p. 1531-44.
13. Zhu, M. and S. Gupta, *To prune, or not to prune: exploring the efficacy of pruning for model compression.* arXiv, 2017. **1710.01878v2**: p. 1-11.
14. Clune, J., J.B. Mouret, and H. Lipson, *The evolutionary origins of modularity.* Proc Biol Sci, 2013. **280**(1755): p. 20122863.
15. Gerum, R.C., et al., *Sparsity through evolutionary pruning prevents neuronal networks from overfitting.* Neural Netw, 2020. **128**: p. 305-312.