Analyzing Groups of Neurons in Neural Networks: Comparing Information from Input and Output Perspectives

Abstract

The concept of a "modular" structure in artificial neural networks has been suggested as beneficial for learning, the ability to combine elements, and applying knowledge to new situations. However, a clear definition and measurement of modularity are still open questions. This paper reframes the identification of functional modules as the identification of groups of units with similar functions. This raises the question of what constitutes functional similarity between two units. To address this, we examine two main categories of methods: those that define similarity based on how units react to variations in inputs (upstream), and those that define similarity based on how changes in hidden unit activations affect outputs (downstream). We perform an empirical analysis to measure the modularity of hidden layer representations in simple feedforward, fully connected networks across various settings. For each model, we assess the relationships between pairs of hidden units in each layer using a range of upstream and downstream metrics, then group them by maximizing their "modularity score" with established network science tools. We find two unexpected results: first, dropout significantly increased modularity, while other forms of weight regularization had smaller effects. Second, while we observe general agreement on clusters within upstream methods and within downstream methods, there is limited agreement on cluster assignments between these two categories. This has significant implications for representation learning, as it implies that finding modular representations that reflect input structure (e.g., disentanglement) may be a different objective from learning modular representations that reflect output structure (e.g., compositionality).

1 Introduction

Modularity, a principle where complex systems are broken down into simpler subsystems, allows for independent analysis, debugging, and recombination for new tasks. This design approach offers benefits like enhanced robustness and quicker adaptation to new challenges. It is recognized that learning systems gain advantages from structures tailored to the specific problem, and many real-world problems can indeed be divided into sub-problems. Consequently, modularity is viewed as a standard design principle in evolved biological systems, including biological neural networks, and one that can be advantageous for artificial neural networks (ANNs).

Despite the intuitive appeal, formally defining and quantifying the modularity of a given system remains an unresolved issue. It is generally agreed that modular systems, by definition, break down into subsystems that carry out functions to solve sub-problems. Defining modules in ANNs, therefore, requires us to determine when two parts of a network are involved in the same "function". In this paper, we address this question at the level of pairs of hidden units. We explore various methods for assessing the "functional similarity" of any two hidden units, and we define a "module" as a group of units with similar functions. This definition is not intended to be the definitive answer to what constitutes a module, but rather to offer a practical foundation for experimenting with different concepts related to modularity, such as how regularization affects it.

A key objective of this paper is to highlight the differences between "upstream" and "downstream" perspectives when considering neural representations and their functions. In Section 3, we provide precise definitions and detail our method for identifying and quantifying functional modules in the hidden layers of trained neural networks by grouping units into functionally similar sets. This framework enables us to directly compare various indicators of a network's modularity. Section 4 describes the experimental results. Besides quantitatively evaluating modularity, we further examine whether different similarity measures agree on the assignment of units to modules. Surprisingly, we find that modules identified using "upstream" measures of functional similarity are consistently different from those found using "downstream" measures. Although we do not examine regularization methods specifically designed to create modular designs, these initial findings call for a more in-depth examination of how the "function" of a representation is defined, as well as why and when modules might be beneficial.

2 Related Work

The investigation of modularity in neural networks has a rich history. A frequent source of inspiration from biology is the separation of "what" and "where" pathways in the ventral and dorsal streams of the brain, respectively. Each pathway can be viewed as a specialized module (and can be further divided into submodules). Numerous prior experiments on modularity in artificial neural networks have investigated principles that would lead to similarly distinct what/where information processing in ANNs. A significant distinction from this line of work is that, instead of predefining the functional role of modules, such as one module handling "what" and another handling "where," our research aims to discover distinct functional groups in trained networks.

Generally, there are two categories of approaches to modularity in neural networks, each corresponding to a different way of understanding the function of network components. The structural modularity approach defines function based on network weights and the connections between sub-networks. Modules are thus defined as sub-networks with dense internal connections and sparse external connections. The functional modularity approach focuses on network activations or the information represented by those activations, rather than weights. This includes concepts like disentanglement, compositionality, and invariance. The connection between structural and functional modules is not entirely clear. While they seem to be (or should be) correlated, it has been observed that even very sparse inter-module connectivity does not always ensure functional separation of information processing. In this study, we adopt the functional approach, assuming that structural modularity is only useful to the extent that it supports distinct functions of the units, and that often distinct functions must share information, making strict structural boundaries potentially detrimental. For instance, in a complex visual scene, knowing "what" an object is can aid in determining "where" it is, and vice versa.

Our work is most closely related to a series of papers by Watanabe and colleagues in which trained networks are decomposed into clusters of "similar" units with the aim of understanding and simplifying those networks. They quantify the similarity of units using a combination of both incoming and outgoing weights. This is similar in spirit to our goal of identifying modules by clustering units, but an interesting contrast to our approach, where we find stark differences between "upstream" and "downstream" similarity.

3 Quantifying modularity by clustering similarity

We divide the task of identifying functional modules into two phases: evaluating the pairwise similarity of units, and then clustering based on this similarity. For simplicity, we apply these steps separately to each hidden layer, although in principle, modules could be assessed in the same way after combining layers. Section 3.1 defines the set of pairwise functional similarity methods we use, and Section 3.2 describes the clustering phase.

While we concentrate on similarity between pairs of individual units, our method is connected to, and inspired by, the question of what makes neural representations "similar" when comparing entire populations of neurons to each other. Instead of finding clusters of similar neurons as we do here, one could define modules in terms of dissimilarity between clusters of neurons. In preliminary work, we explored such a definition of functional modules, using representational (dis)similarity between sub-populations of neurons. The primary challenge with this approach is that existing representational similarity methods are highly sensitive to dimensionality (the number of neurons in each cluster), and it is not clear how best to account for this when calculating dissimilarity between clusters so that the method is not biased towards larger or smaller cluster sizes. To further justify our method, note that representational similarity analysis is closely related to tests for statistical (in)dependence between populations of neurons, and so the problem of finding mutually "dissimilar" modules is analogous to the problem of finding independent subspaces. In Independent Subspace Analysis (ISA), there is a similar issue of determining what constitutes a surprising amount of dependence between subspaces of different dimensions, and various methods have been proposed with different inductive biases. However, Palmer Makeig showed that a solution to the problem of detecting independent subspaces is to simply cluster the individual dimensions of the space. This provides some justification for the methods we use here: some technicalities notwithstanding, the problem of finding subspaces of neural activity with "dissimilar" representations is, in many cases, reducible to the problem of clustering individual units based on pairwise similarity, as we do here.

3.1 Quantifying pairwise similarity of hidden units

What constitutes "functional similarity" between two hidden units? In other words, we are looking for a similarity function S that takes a neural network N, a dataset D, and a task T as inputs, and produces an n x n matrix of non-negative similarity scores for all pairs among the n hidden units. We also require that the resulting matrix is symmetric, meaning Sij = Sji. Importantly, allowing S to depend on the task T opens up the possibility of similarity measures where units are considered similar based on their downstream contribution to a specific loss function.

Similarity by covariance. The first similarity measure we examine is the absolute value of the covariance of hidden unit activities across inputs. Let xk be the kth input in the dataset, and hi(x) be the response of the ith hidden unit to input x, with i in 1, 2, ..., n. Then, we define similarity as

$$S_{ij}^{cov} = \frac{1}{K} \sum_{k=1}^{K} |(h_i(x_k) - \bar{h_i})(h_j(x_k) - \bar{h_j})| \tag{1}$$

where K is the number of items in D and $\bar{h_i}$ is the mean response of unit i on the given dataset. Intuitively, the absolute value covariance quantifies the statistical dependence of two units across inputs, making it an upstream measure of similarity.

Similarity by input sensitivity. While S^{cov} measures similarity of responses across inputs, we next consider a measure of similar sensitivity to single inputs, which is then averaged over D. Let $J^h_{x_k}$ denote the n x d Jacobian matrix of partial derivatives of each hidden unit with respect to each of the d input dimensions. Then, we say two units i and j are similarly sensitive to input changes on input x_k if the dot product between the ith and jth row of $J^h_{x_k}$ has high absolute-value magnitude. In matrix notation over the entire dataset, we use

$$S^{i-sens}_{ij} = \frac{1}{K}\sum_{k=1}^{K} |J^h_{x_k}(J^h_{x_k})^T| \tag{2}$$

where the superscript "i-sens" should be read as the "input sensitivity."

Similarity by last-layer sensitivity. Let y denote the last-layer activity of the network. Using the same Jacobian notation as above, let J^y_h denote the o x n matrix of partial derivatives of the last layer with respect to changes in the hidden activities h. Then, we define similarity by output sensitivity as

$$S^{o-sens}_{ij} = \frac{1}{K}\sum_{k=1}^{K} |J^y_h(J^y_h)^T| \tag{3}$$

likewise with "o-sens" to be read as "output-sensitivity." Note that both h and y depend on the particular input x_k, but this has been left implicit in the notation to reduce clutter.

Similarity by the loss Hessian. The "function" of a hidden unit might usefully be thought of in terms of its contribution to the task or tasks it was trained on. To quote Lipson, "In order to measure modularity, one must have a quantitative definition of function... It is then possible to take an arbitrary chunk of a system and measure the dependency of the system function on elements within that chunk. The more that the dependency itself depends on elements outside the chunk, the less the function of that chunk is localized, and hence the less modular it is."

Lipson then goes on to suggest that the "dependence of system function on elements" can be expressed as a derivative or gradient, and that the dependence of that dependence on other parts of the system can be expressed as the second derivative or Hessian. Towards this conception of modular functions on a particular task, we use the following definition of similarity:

$$S^{hess}_{ij} = \frac{1}{K}\sum_{k=1}^{K} |\frac{\partial^2 L}{\partial h_i \partial h_j}| \tag{4}$$

where L is the scalar loss function for the task, and should be understood to depend on the particular input x_k. Importantly, each Hessian on the right hand side is taken with respect to the activity of hidden units, not with respect to the network parameters as it is typically defined.

To summarize, equations (1) through (4) provide four different methods to quantify pairwise similarity of hidden units. S^{cov} and S^{i-sens} are upstream, while S^{o-sens} and S^{hess} are downstream. All four take values in [0,). However, it is not clear if the raw magnitudes matter, or only relative (normalized) magnitudes. For these reasons, we introduce an optional normalized version of each of the above four un-normalized similarity measures:

$$S'_{ij} = \frac{S_{ij}}{max(S_{ii}, S_{jj}, \epsilon)} \tag{5}$$

where $\breve{2}$0ac is a small positive value included for numerical stability. Whereas S_{ij} is in [0,), the normalized values are restricted to S'_{ij} in [0,1]. In total, this gives us eight methods to quantify pairwise similarity. These can be thought of as 2x2x2 product of methods, as shown in the color scheme in Figure 2: the upstream vs downstream axis, the unnormalized vs normalized axis, and the covariance vs gradient (i.e. sensitivity) axis. We group together both S^{cov} and S^{hess} under the term "covariance" because the Hessian is closely related to the covariance of gradient vectors of the loss across inputs.

3.2 Quantifying modularity by clustering

Decomposing a set into clusters that are maximally similar within clusters and maximally dissimilar across clusters is a well-studied problem in graph theory and network science. In particular, Girvan Newman proposed a method that cuts a graph into its maximally modular subgraphs, and this tool has previously been used to study modular neural networks.

We apply this tool from graph theory to our problem of detecting functional modules in neural networks by constructing an adjacency matrix A from the similarity matrix S by simply removing the diagonal (self-similarity):

$$A_{ij} = \begin{cases} S_{ij} & if\ i \neq j \\ 0 & otherwise \end{cases} \tag{6}$$

Given A, we can simplify later notation by first constructing the normalized adjacency matrix, $\tilde{A}$, whose elements all sum to one:

$$\tilde{A}_{ij} = \frac{A_{ij}}{\sum_{ij} A_{ij}} \tag{7}$$

or, more compactly, $\tilde{A} = A/1_n^T A 1_n$ where 1_n is a column vector of length n containing all ones. Let P be an n x c matrix that represents cluster assignments for each of n units to a maximum of c different clusters. Cluster assignments can be "hard" (P_{ij} in 0, 1) or "soft" (P_{ij} in [0, 1]), but in either case the constraint $P1_c = 1_n$ must be met, i.e. that the sum of cluster assignments for each unit is 1. If an entire column of P is zero, that cluster is unused, so c only provides an upper-limit to the number of clusters, and in practice we set c = n. Girvan Newman propose the following score to quantify the level of "modularity" when partitioning the normalized adjacency matrix $\tilde{A}$ into the cluster assignments P:

$$Q(\tilde{A}, P) = Tr(P^T \tilde{A} P) - Tr(P^T \tilde{A} 1_n 1_n^T \tilde{A} P) \tag{8}$$

The first term sums the total connectivity (or, in our case, similarity) of units that share a cluster. By itself, this term is maximized when P assigns all units to a single cluster. The second term gives the expected connectivity within each cluster under a null model where the elements of $\tilde{A}$ are interpreted as the joint probability of a connection, and so $\tilde{A} 1_n 1_n^T \tilde{A}$ is the product of marginal probabilities of each unit's connections. This second term encourages P to place units into the same cluster only if they are more similar to each other than "chance." Together, equation (8) is maximized by partitioning $\tilde{A}$ into clusters that are strongly intra-connected and weakly inter-connected.

We define the modularity of a set of neural network units as the maximum achievable Q over all P:

$$P^*(\tilde{A}) = argmax_P Q(\tilde{A}, P) Q^*(\tilde{A}) = Q(\tilde{A}, P^*) \tag{9}$$

To summarize, to divide a given pairwise similarity matrix S into modules, we first construct $\tilde{A}$ from S, then we find the cluster assignments P^* that give the maximal value Q^*. Importantly, this optimization process provides two pieces of information: a modularity score Q^* which quantifies the amount of modularity in a set of neurons, for a given similarity measure. We also get the actual cluster assignments P^*, which provide additional information and can be compared across different similarity measures. Given a set of cluster assignments P^*, we quantify the number of clusters by first getting the fraction of units in each cluster, $r(P^*) = 1_n^T P^*/n$. We then use the formula for discrete entropy to measure the dispersion of cluster sizes: $H(r) = -\sum_{i=1}^{c} r_i log r_i$. Finally we say that the number of clusters in P^* is

$$numclusters(P^*) = e^{H(r(P^*))} \tag{10}$$

We emphasize that discovering the number of clusters in P^* is included automatically in the optimization process; we set the maximum number of clusters c equal to the number of hidden units n, but in our experiments we find that P^* rarely uses more than 6 clusters for hidden layers with 64 units (Supplemental Figure S4).

It is important to recognize that the sense of the word "modularity" in graph theory is in some important ways distinct from its meaning in terms of engineering functionally modular systems. In graph-theoretic terms, a "module" is a cluster of nodes that are highly intra-connected and weakly inter-connected to other parts of the network, defined formally by Q. This definition of graph modularity uses a particular idea of a "null model" based on random connectivity between nodes in a graph. While this null-model of graph connectivity enjoys a good deal of historical precedence in the theory of randomly-connected graphs, where unweighted graphs are commonly studied in terms of the probability of connection between random pairs of nodes, it is not obvious that the same sort of null model applies to groups of "functionally similar" units in an ANN. This relates to the earlier discussion of ISA, and provides a possibly unsatisfying answer to the question of what counts as a "surprising" amount of statistical independence between clusters; using Q makes the implicit choice that the product of average pairwise similarity, $\tilde{A} 1_n 1_n^T \tilde{A}$, gives the "expected" similarity between units. An important problem for future work will be to closely reexamine the question of what makes neural populations functionally similar or dissimilar, above and beyond statistical similarity, and what constitutes a surprising amount of (dis)similarity that may be indicative of modular design.

Finding P^* exactly is NP-complete, so in practice we use a variation on the approximate method proposed by Newman. Briefly, the approximation works in two steps: first, an initial set of cluster assignments is constructed using a fast spectral initialization method that, similar to other spectral clustering algorithms, recursively divides units into clusters based on the sign of eigenvectors of the

matrix $B = \tilde{A} - \tilde{A}1_n 1_n^T \tilde{A}$ and its submatrices. Only subdivisions that increase Q are kept. In the second step, we use a Monte Carlo method that repeatedly selects a random unit i then resamples its cluster assignment, holding the other n-1 assignments fixed. This resampling step involves a kind of exploration/exploitation trade-off: Q may decrease slightly on each move to potentially find a better global optimum. We found that it was beneficial to control the entropy of each step using a temperature parameter, to ensure that a good explore/exploit balance was struck for all $\tilde{A}$. Supplemental Figure S2 shows that both the initialization and the Monte Carlo steps play a crucial role in finding P^*, consistent with the observations of Newman. Full algorithms are given in Appendix A.1.

4 Experiments

4.1 Setup and initial hypotheses

Because our primary goal is to understand the behavior of the various notions of modularity above, i.e. based on the eight different methods for quantifying pairwise similarity introduced in the previous section, we opted to study a large collection of simple networks trained on MNIST. All pairwise similarity scores were computed using held-out test data. We trained 270 models, comprising 9 runs of each of 30 regularization settings, summarized in Table 1. We defined x (input layer) as the raw 784-dimensional pixel inputs and y (output layer) as the 10-dimensional class logits. We used the same basic feedforward architecture for all models, comprising two layers of hidden activity connected by three layers of fully-connected weights: Linear(784, 64), ReLU, dropout(p), Linear(64, 64), ReLU, dropout(p), Linear(64, 10). We analyzed modularity in the two 64-dimensional hidden layers following the dropout operations. We discarded 21 models that achieved less than 80

Before running these experiments, we hypothesized that

1. Dropout would decrease modularity by encouraging functions to be "spread out" over many units. 2. L2 regularization (weight decay) would minimally impact modularity since the L2 norm is invariant to rotation while modularity depends on axis-alignment. 3. L1 regularization on weights would increase modularity by encouraging sparsity between subnetworks. 4. All similarity measures would be qualitatively consistent with each other.

As shown below, all four of these hypotheses turned out to be wrong, to varying degrees.

4.2 How modularity depends on regularization

Figure 3 shows the dependence of trained networks' modularity score (Q^*) as a function of regularization strength for each of three types of regularization: an L2 penalty on the weights (weight decay), an L1 penalty on the weights, and dropout. The top row of Figure 3 shows four example $\tilde{A}$ matrices sorted by cluster, to help give an intuition behind the quantitative values of Q^*. In these examples, the increasing value of Q^* is driven by an increasing contrast between intra-cluster similarity and inter-cluster similarity. In this example, it also appears that the number and size of clusters remains roughly constant; this observation is confirmed by plotting the number of clusters versus regularization strength in Supplemental Figure S4.

Figure 3 shows a number of surprising patterns that contradict our initial predictions. First, and most saliently, we had predicted that dropout would reduce modularity, but found instead that it has the greatest effect on Q^* among the three regularization methods we tried. This is especially apparent in the upstream methods (first two columns of the figure), and is also stronger for the first hidden layer than the second (Supplemental Figure S3). In general, Q^* can increase either if the network partitions into a greater number of clusters, or if the contrast between clusters is exaggerated. We found that this dramatic effect of dropout on Q^* was accompanied by only minor changes to the number of clusters (Supplemental Figure S4), and so we can conclude that dropout increases Q^* by increasing the redundancy of hidden units. In other words, hidden units become more clusterable because they are driven towards behaving like functional replicas of each other, separately for each cluster. This observation echoes, and may explain, why dropout also increases the "clusterability" of network weights in a separate study.

The second surprising result in Figure 3 is that L2 regularization on the weights did, in fact, increase Q^*, whereas we had expected it to have no impact. Third, L1 regularization had a surprisingly weak effect, although its similarity to the L2 regularization results may be explained by the fact that they actually resulted in fairly commensurate sparsity in the trained weights (Supplemental Figure S1 bottom row). Fourth, we had expected few differences between the eight different methods for computing similarity, but there appear to be distinctive trends by similarity type both in Figure S3 as well as in the number of clusters detected (Supplemental Figure S4). The next section explores the question of similarity in the results in more detail.

4.3 Comparing modules discovered by different similarity methods

The previous section discussed idiosyncratic trends in the modularity scores Q^* as a function of both regularization strength and how pairwise similarity between units (S) is computed. However, such differences in the quantitative value of Q^* are difficult to interpret, and would largely be moot if the various methods agreed on the question of which units belong in which cluster. We now turn to the question of how similar the cluster assignments P^* are across our eight definitions of functional modules. To minimize ambiguity, we will use the term "functional-similarity" to refer to S, and "cluster-similarity" to refer to the comparison of different cluster assignments P^*.

Quantifying similarity between cluster assignments is a well-studied problem, and we tested a variety of methods in the clusim Python package. All cluster-similarity methods we investigated gave qualitatively similar results, so here we report only the "Element Similarity" method of Gates et al., which is a value between 0 and 1 that is small when two cluster assignments are unrelated, and large when one cluster assignment is highly predictive of the other. Note that this cluster-similarity analysis is applied only to P^* cluster assignments computed in the same layer of the same model. Thus, any dissimilarity in clusters that we see is due entirely to the different choices for functional-similarity, S.

Figure 4a summarizes the results of this cluster-similarity analysis: there is a striking difference between clusters of units identified by "upstream" functional-similarity methods (S^{cov}, $\tilde{S}^{cov}$, S^{i-sens}, $\tilde{S}^{i-sens}$) compared to "downstream" functional-similarity methods (S^{hess}, $\tilde{S}^{hess}$, S^{o-sens}, $\tilde{S}^{o-sens}$). This analysis also reveals secondary structure within each class of upstream and downstream methods, where the choice to normalize not (S vs $\tilde{S}$) appears to matter little, and where there is a moderate difference between moment-based methods (S^{cov}, S^{hess}) and gradient-based methods (S^{i-sens}, S^{o-sens}). It is worth noting that some of this secondary structure is not robust across all types and levels of regularization; in particular, increasing L2 or L1 regularization strength appears to lead to (i) stronger dependence on normalization in the downstream methods, and (ii) a stronger overall agreement among the upstream methods (Supplemental Figure S5).

We next asked to what extent these cluster-similarity results are driven by training. As shown in Figure 4b, much of the structure in the downstream methods is unaffected by training (i.e. it is present in untrained models as well), while the cluster-similarity among different upstream methods only emerged as a result of training. Interestingly, this analysis further shows that the main upstream-vs-downstream distinction seen in Figure 4a is, in fact, attenuated slightly by training.

5 Conclusions

The prevalence of "modular" designs in both engineered and evolved systems has led many to consider the benefits of modularity as a design principle, and how learning agents like artificial neural networks might discover such designs. However, precisely defining what constitutes a "module" within a neural network remains an open problem. In this work, we operationalized modules in a neural network as groups of hidden units that carry out similar functions. This naturally leads to the question of what makes any two units functionally similar. We introduced eight functional similarity measures designed to capture various intuitions about unit similarity and empirically evaluated cluster assignments based on each method in a large number of trained models.

One unexpected observation was that dropout increases modularity (as defined by Q^*), although this has little to do with the common-sense definition of a "module." Instead, it is a byproduct of dropout causing subsets of units to behave like near-copies of each other, perhaps so that if one unit is dropped out, a copy of it provides similar information to the subsequent layer. To our knowledge, this redundancy-inducing effect of dropout has not been noted in the literature previously.

Our main result is that there is a crucial difference between defining "function" in terms of how units are driven by upstream inputs, and how units drive downstream outputs. While we studied this distinction between upstream and downstream similarity in the context of modularity and clustering, it speaks to the deeper and more general problem of how best to interpret neural representations. For example, some sub-disciplines of representation-learning (e.g. "disentanglement") have long emphasized that a "good" neural representation is one where distinct features of the world drive distinct sub-populations or sub-spaces of neural activity. This is an upstream way of thinking about what is represented, since it depends only on the relationship between inputs and the unit activations and does not take into account what happens downstream. Meanwhile, many have argued that the defining characteristic of a neural representation is its causal role in downstream behavior; this is, of course, a downstream way of thinking. At a high level, one way to interpret our results is is that upstream and downstream ways of thinking about neural representations are not necessarily aligned, even in trained networks. This observation is reminiscent of recent empirical work finding that "disentangled" representations in auto-encoders (an upstream concept) do not necessarily lead to improved performance or generalization to novel tasks (a downstream concept).

Despite its theoretical motivations, this is an empirical study. We trained over 250 feedforward, fully-connected neural networks on MNIST. While it is not obvious whether MNIST admits a meaningful "modular" solution, we expect that the main results we show here are likely robust, in particular (i) the effect of weight decay, an L1 weight penalty, and dropout, and (ii) misalignment between upstream and downstream definitions of neural similarity.

Our work raises the important questions: are neural representations defined by their inputs or their outputs? And, in what contexts is it beneficial for these to be aligned? We look forward to future work applying our methods to larger networks trained on more structured data, as well as recurrent networks. We also believe it will be valuable to evaluate the effect of attempting to maximize modularity, as we have defined it, during training, to see to what extent this is possible and whether it leads to performance benefits. Note that maximizing Q during training is challenging because (i) computing S may require large batches, and more importantly (ii) optimizing Q is highly prone to local minima, since neural activity and cluster assignments P will tend to reinforce each other, entrenching accidental clusters that appear at the beginning of training. We suspect that maintaining uncertainty over cluster assignments (e.g. using soft P_{ij} in [0, 1] rather than hard P in 0, 1 cluster assignments) will be crucial if optimizing any of our proposed modularity metrics during training.

References

Mohammed Amer and Tomás Maul. A review of modularization techniques in artificial neural networks. Artificial Intelligence Review, 52(1):527-561, 2019.

Jacob Andreas. Measuring compositionality in representation learning. arXiv, pp. 1-15, 2019.

Farooq Azam. Biologically inspired modular neural networks. Phd, Virginia Polytechnic Institute and State University, 2000.

Francis R. Bach and Michael I. Jordan. Kernel independent component analysis. Journal of Machine Learning Research, 3(1):1-48, 2003.

Francis R. Bach and Michael I. Jordan. Beyond independent components: Trees and clusters. Journal of Machine Learning Research, 4(7-8):1205-1233, 2004.

Shahab Bakhtiari, Patrick Mineault, Tim Lillicrap, Christopher C Pack, and Blake A Richards. The functional specialization of visual cortex emerges from training parallel pathways with self-supervised predictive learning. NeurIPS, 3, 2021.

Gabriel Béna and Dan F. M. Goodman. Extreme sparsity gives rise to functional specialization. arXiv, 2021.

Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. IEEE Transactions on Pattern Analysis and Machine Intelligence, 35(8):1798-1828, 2013.

U. Brandes, D. Delling, M. Gaertler, R. Gorke, M. Hoefer, Z. Nikoloski, and D. Wagner. On Modularity Clustering. IEEE Transactions on Knowledge and Data Engineering, 20(2):172-188, 2008.

Jeff Clune, Jean Baptiste Mouret, and Hod Lipson. The evolutionary origins of modularity. Proceedings of the Royal Society B, 280, 2013.

Rion B Correia, Alexander J Gates, Xuan Wang, and Luis M Rocha. Cana: A python package for quantifying control and canalization in boolean networks. Frontiers in physiology, 9:1046, 2018.

Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. Algorithms for learning kernels based on centered alignment. Journal of Machine Learning Research, 13:795-828, 2012.

Róbert Csordás, Sjoerd van Steenkiste, and Jürgen Schmidhuber. Are Neural Nets Modular? Inspecting Functional Modularity Through Differentiable Weight Masks. ICLR, 2021.

J Denker, D Schwartz, B Wittner, S Solla, R Howard, L Jackel, and J Hopfield. Large Automatic Learning, Rule Extraction, and Generalization. Complex Systems, 1:877-922, 1987.

Andrea Di Ferdinando, Raffaele Calabretta, and Domenico Parisi. Evolving Modular Architectures for Neural Networks. Proceedings of the sixth Neural Computation and Psychology Workshop: Evolution, Learning, and Development, pp. 253-262, 2001.

Cian Eastwood and Christopher K.I. Williams. A framework for the quantitative evaluation of disentangled representations. ICLR, 2018.

Daniel Filan, Stephen Casper, Shlomi Hod, Cody Wild, Andrew Critch, and Stuart Russell. Clusterability in Neural Networks. arXiv, 2021.

Justin Garson and David Papineau. Teleosemantics, Selection and Novel Contents. Biology Philosophy, 34(3), 2019.

Alexander J. Gates, Ian B. Wood, William P. Hetrick, and Yong Yeol Ahn. Element-centric clustering comparison unifies overlaps and hierarchy. Scientific Reports, 9(1):1-13, 2019.

M. Girvan and M. E.J. Newman. Community structure in social and biological networks. Proceedings of the National Academy of Sciences of the United States of America, 99(12):7821-7826, 2002.

Melvyn A. Goodale and A. David Milner. Separate visual pathways for perception and action. TINS, 15(1): 20-25, 1992.

Arthur Gretton, Olivier Bousquet, Alex Smola, and Bernhard Schölkopf. Measuring statistical dependence with Hilbert-Schmidt norms. In S. Jain, H. U. Simon, and E. Tomita (eds.), Lecture Notes in Artificial Intelligence, volume 3734, pp. 63-77. Springer-Verlag, Berlin, 2005.

Harold W Gutch and Fabian J Theis. Independent Subspace Analysis is Unique, Given Irreducibility. In Mike E Davies, Christopher J James, Samer A Abdallah, and Mark D Plumbley (eds.), Independent Component Analysis and Signal Separation, volume 7. Springer, Berlin, 2007.

Irina Higgins, David Amos, David Pfau, Sebastien Racaniere, Loic Matthey, Danilo Rezende, and Alexander Lerchner. Towards a Definition of Disentangled Representations. arXiv, pp. 1-29, 2018.

Aapo Hyvärinen, Patrik O. Hoyer, and Mika Inki. Topographic independent component analysis. Neural Computation, 13(7):1527-1558, 2001.

Robert A Jacobs, Michael I Jordan, and Andrew G Barto. Task Decomposition Through Competition in a Modular Connectionist Architecture:The What and Where Vision Tasks. Cognitive Science, pp. 219-250, 1991.

Nadav Kashtan and Uri Alon. Spontaneous evolution of modularity and network motifs. Proceedings of the National Academy of Sciences of the United States of America, 102(39):13773-13778, 2005.

Nadav Kashtan, Elad Noor, and Uri Alon. Varying environments can speed up evolution. Proceedings of the National Academy of Sciences of the United States of America, 104(34):13711-13716, 2007.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of Neural Network Representations Revisited. ICML, 36, 2019.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-Based Learning Applied to Document Recognition. Proceedings of the IEEE, 86(11):2278-2324, 1998.

H Lipson. Principles of modularity, regularity, and hierarchy for scalable systems. Journal of Biological Physics and Chemistry, 7(4):125-128, 2007.

Francesco Locatello, Stefan Bauer, Mario Lucic, Sylvain Gelly, Bernhard Schölkopf, and Olivier Bachem. Challenging Common Assumptions in the Unsupervised Learning of Disentangled Representations. arXiv, pp. 1-33, 2019.

Milton Llera Montero, Casimir JJ Ludwig, Rui Ponte Costa, Guarav Malhotra, and Jeffrey Bowers. The role of disentanglement in generalization. ICLR, 2021.

M. E.J. Newman. Modularity and community structure in networks. Proceedings of the National Academy of Sciences of the United States of America, 103(23):8577-8582, 2006.

M. E.J. Newman and M. Girvan. Finding and evaluating community structure in networks. Physical Review E - Statistical, Nonlinear, and Soft Matter Physics, 69(2 2):1-15, 2004.

Jason A. Palmer and Scott Makeig. Contrast functions for independent subspace analysis. In Fabian J. Theis, A. Cichocki, A. Yeredor, and M. Zibulevsky (eds.), Independent Component Analysis and Signal Separation, volume LNCS 7191, pp. 115-122. Springer-Verlag, Berlin, 2012.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high- performance deep learning library. In Advances in Neural Information Processing Systems 32, pp. 8024-8035. Curran Associates, Inc., 2019.

Barnabás Póczos and András Lőrincz. Independent Subspace Analysis Using Geodesic Spanning Trees. ICML, 22:673-680, 2005.

Karl Ridgeway and Michael C. Mozer. Learning deep disentangled embeddings with the F-statistic loss. Advances in Neural Information Processing Systems, pp. 185-194, 2018.

J. G. Rueckl, K. R. Cave, and S. M. Kosslyn. Why are "what" and "where" processed by separate cortical visual systems? A computational investigation. Journal of Cognitive Neuroscience, 1(2):171-186, 1989.

Bernhard Scholkopf, Francesco Locatello, Stefan Bauer, Nan Rosemary Ke, Nal Kalchbrenner, Anirudh Goyal, and Yoshua Bengio. Toward Causal Representation Learning. Proceedings of the IEEE, 109(5): 612-634, 2021.

Herbert A Simon. The Architecture of Complexity. Proceedings of the American Philosophical Society, 106 (6), 1962.

O. Tange. Gnu parallel - the command-line power tool. ;login: The USENIX Magazine, 36(1):42-47, Feb 2011.

Günter P. Wagner, Mihaela Pavlicev, and James M. Cheverud. The road to modularity. Nature Reviews Genetics, 8(12):921-931, 2007.

Chihiro Watanabe. Interpreting Layered Neural Networks via Hierarchical Modular Representation. Communications in Computer and Information Science, 1143 CCIS:376-388, 2019.

Chihiro Watanabe, Kaoru Hiramatsu, and Kunio Kashino. Modular representation of layered neural networks. Neural Networks, 97:62-73, 2018.

Chihiro Watanabe, Kaoru Hiramatsu, and Kunio Kashino. Understanding community structure in layered neural networks. Neurocomputing, 367:84-102, 2019.

Chihiro Watanabe, Kaoru Hiramatsu, and Kunio Kashino. Knowledge discovery from layered neural networks based on non-negative task matrix decomposition. IEICE Transactions on Information and Systems, E103D(2):390-397, 2020.

Zongze Wu, Chunchen Su, Ming Yin, Zhigang Ren, and Shengli Xie. Subspace clustering via stacked independent subspace analysis networks with sparse prior information. Pattern Recognition Letters, 146: 165-171, 2021.

A Appendix

A.1 Algorithms

This section provides pseudocode for the algorithm used to compute clusters P^* from the normalized matrix of pairwise associations between units, $\tilde{A}$. Before running these algorithms, we always remove all-zero rows and columns from $\tilde{A}$; we consider these units to all be in a separate "unused" cluster.

L2 (weight decay)	L1 weight penalty	dropout prob.
logspace(-5,-1,9)	0.0	0.0
1e-5	logspace(-5,-2,7)	0.0
1e-5	0.0	linspace(0.05,0.7,14)

Table 1: Each row describes one hyperparameter sweep, for a total of 30 distinct hyperparameter values. First row: varying weight decay (L2 weight penalty) with no other regularization (9 values). Second row: varying L1 penalty on weights along with mild weight decay (7 values). Third row: varying dropout probability in increments of 0.05 along with mild weight decay (14 values).

Algorithm 1 Full clustering algorithm.

Require: Normalized pairwise associations $\tilde{A}$

$P \leftarrow$ GreedySpectralModules($\tilde{A}$) . Initialize P using spectral method

$P^* \leftarrow$ MonteCarloModules($\tilde{A}$, P) . Further refine P using Monte Carlo method

return P^*

Algorithm 2 Pseudocode for greedy, approximate, spectral method for finding modules

function GreedySpectralModules($\tilde{A}$)

    $B \leftarrow \tilde{A} - \tilde{A}1_n 1_n^T \tilde{A}$. . B is analogous to the graph Laplacian, but for modules

    $P \leftarrow [1 \quad 0 \quad 0 \quad ... \quad 0]_n^T$. Initialize P to a single cluster, which will be (recursively) split later.

    $queue \leftarrow [0]$. FILO queue keeping track of which cluster we'll try splitting next

    $Q \leftarrow Tr(P^T B P)$. Compute Q for the initial P

    while queue is not empty **do**

        $c \leftarrow queue.pop()$. Pop the next (leftmost) cluster id

        $i \leftarrow$ indices of all units currently in cluster c according to P

        $v \leftarrow eig(B(i,i))$. Get the leading eigenvector of the submatrix of B containing just units in c

        $i^+ \leftarrow$ subset of i where v was positive . Split v by sign (if not possible, continue loop)

        $i^- \leftarrow$ subset of i where v was negative

        $c0 \leftarrow$ index of the next available (all zero) column of P

        $P^0 \leftarrow P$ but with all i^- units moved to cluster $c0$. Try splitting c into c, c0 based on sign of v

        $Q^0 \leftarrow Tr(P^{0T} B P^0)$. Compute updated Q for newly-split clusters P^0

        if $Q^0 > Q$ **then**

            $Q, P \leftarrow Q^0, P^0$. Update Q and P

            $queue.append(c, c0)$. Push c and c0 onto the queue to consider further subdividing them

        else

            . Nothing to do - splitting c into c0 did not improve Q, so we don't add further subdivisions to the queue, and we keep the old P, Q values

        end if

    end while

    return P . Once the queue is empty, P contains a good initial set of cluster assignments

end function

Algorithm 3 Pseudocode for Monte Carlo method for improving clusters.

function MonteCarloModules($\tilde{A}$, P, n)

    for n steps **do**

        $i \leftarrow$ index of a single a randomly selected unit

$c \leftarrow$ index of the first empty cluster in P

$Q^*, P^* \leftarrow Tr(P^T(\tilde{A} - \tilde{A}1_n 1_n^T \tilde{A})P), P$. Keep track of best Q, P pair found so far

for $j = 1...c$ **do** . Try moving unit i to each cluster j, including a new cluster at c

$P^0 \leftarrow P$ with i reassigned to cluster j

$Q_j^0 \leftarrow Tr(P^{0T}(\tilde{A} - \tilde{A}1_n 1_n^T \tilde{A})P^0)$. Compute updated Q with re-assigned unit

if $Q_j^0 > Q^*$ **then**

$Q^*, P^* \leftarrow Q_j^0, P^0$. Update Q^*, P^* pair, even if we don't select this j later

end if

end for

$\tau \leftarrow$ whatever temperature makes $p \propto e^{Q^0/\tau}$ have entropy $H = 0.15$

$p \leftarrow e^{Q^0/\tau} / \sum_j e^{Q_j^0/\tau}$. We found H = 0.15 strikes a good balance between exploration and greedy ascent.

$j^* \sim p$. Sample new cluster assignment j from categorical distribution p

$P \leftarrow P$ with unit i reassigned to cluster j^*, ensuring only the leftmost columns have nonzero values

end for

return P^*

end function

A.2 Supplemental Figures

[width=]images.png

Figure 1: Basic performance metrics as a function of regularization strength. Each column corresponds to a different regularization method, as in Table 1. Each row shows a metric calculated on the trained models. Thin colored lines are individual seeds, and thick black line is the average ± standard error across runs. Horizontal gray line shows each metric computed on randomly initialized network. Sparsity (bottom row) is calculated as the fraction of weights in the interval [-1e-3, +1e-3].

[width=0.45]image1.png [width=0.45]image2.png

Figure 2: Both spectral initialization and Monte Carlo optimization steps contribute to finding a good value of Q^*. Left: The x-axis shows modularity scores (Q^*) achieved using only the greedy spectral method for finding P^*. The y-axis shows the actual scores we used in the paper by combining the spectral method for initialization plus Monte Carlo search. The fact that all points are on or above the y=x line indicates that the Monte Carlo search step improved modularity scores. Right: The x-axis now shows modularity scores (Q^*) achieved using 1000 Monte Carlo steps, after initializing all units into a single cluster (we chose a random 5% of the similarity-matrices that were analyzed in the main paper to re-run for this analysis, which is why there are fewer points in this subplot than in the left subplot). The fact that all points are on or above the y=x line indicates that using the spectral method to initialize improved the search.

[width=]image3.png

Figure 3: Modularity score (Q^*) versus regularization, split by layer. Format is identical to Figure 3, which shows modularity scores averaged across layers. Here, we break this down further by plotting each layer separately. The network used in our experiments has two hidden layers. The first two rows (white background) shows modularity scores for the first hidden layer h1, and the last two rows (gray background) shows h2.

Figure 4: Number of clusters in P^* versus regularization, split by layer. Layout is identical to Figure S3. Gray shading in the background shows 1σ, 2σ, and 3σ quantiles of number of clusters in untrained (randomly initialized) networks. Note that, for the most part, training has little impact on the number of clusters detected, suggesting that consistently finding on the order of 2-6 clusters is more a property of the MNIST dataset itself than of training. We computed the number of clusters using equation (10). This measure is sensitive to both the number and relative size of the clusters.

Figure 5: Further breakdown of cluster-similarity by regularization strength (increasing left to right) and type (L2/L1/dropout). Results in Figure 4 reflect an average of the results shown here. The six rows of this figure should be read in groups of two rows: in each group, the top row shows the similarity scores (averaged over layers and runs), and the bottom row shows the difference to untrained models. A number of features are noteworthy here: (i) at low values of all three types of regularization, there is little cluster- similarity within the upstream methods, but it becomes very strong at as regularization strength grows; (ii) at the highest values of L2 and L1 regularization, the pattern inside the 4x4 block of downstream methods changes to depend more strongly on normalization; (iii) a moderate amount of agreement between upstream and downstream methods is seen for large L1 regularization strength, but curiously only for unnormalized downstream methods.