Neural network response in learning from synthetic data and *network motifs* formation

Contents

General scope information

Code and documentation available here: GitHub repo.
Main references:

(1) Kashtan, N., Alon, U., *Spontaneous evolution of modularity and network motifs*, 2005. PNAS.

(2) Kashtan, N., Itzkovitz, S., Milo, R., Alon, U., *Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs*, 2004. NCBI.

(3) Kemp, C., Tenenbaum, J.B., *The discovery of structural form*, 2008. PNAS.

(4) Saxe, A.M., McClelland, J.L., Ganguli, S., *A mathematical theory of semantic development in deep neural networks*, 2018. arXiv.

(5) Saxe, A.M., McClelland, J.L., Ganguli, S., *Exact solutions to the nonlinear dynamics of learning in deep linear neural networks*, 2014. arXiv

(6) Goodfellow, I., Bengio, Y., Courville, A., *Deep Learning*, 2015. Online book.

(7) Newman, M.E.J., *Fast algorithm for detecting community structure in networks*, 2003. arXiv.

(8) Newman, M.E.J., Girvan, M., *Finding and evaluating community structure in networks*, 2003. arXiv.

(9) Kirkpatrick S., Gelatt, C.D., Vecchi, M.P., *Optimization by simulated annealing*, 1983. Science

Most of the figures, grouping different plots, are provided at the end of the document, in order not to make the text body heavy to read.

What's new

As made by Kashtan and Alon (2005), modularly varying goal (MVG) has been implemented. However, in the present setting this may be an end to itself exercise, in that in the cited work networks systems have been submitted to an **evolutionary process**, by means of genetics-inspired algorithms. Here the *evolutionary force*, almost, its far analogous, is the **gradient descend** algorithm. This is the main difference one could immediately spot between evolution by means of environmental stimuli and evolution of network parameters driven by gradient descend. In this latter case the task compliance is carried out by the clear definition of **how** weights must change. Otherwise, if random mutation were performed on the system with the aim of finding an optimal configuration, it could take an exaggerated amount of time for the algorithm to converge to a good configuration, or not to at all.

A new data set has been introduced in this version. Now the neural system is trained both on the binary tree data set and the independent clusters data set. The creation of this latter underlies some physical resemblance and embeds the use of Probabilistic Graphical Models. All is presented below.

At the present stage the `mfinder1.2` program is still used, but it is in order to develop from scratch a new method that finds features of the evolved model which fits better in this present framework: With respect to previous work on network motifs (Kashtan and Alon, 2005 and Kashtan *et al.*, 2004) here one can not neglect the connection strengths of the neural network once it is trained.

1 Introduction, aim and scope

In brief

Two data sets are fed to the neural system. These are generated with the purpose to embody different statistical structures. This particular feature has been put in relationship with the dynamics of learning in neural networks, shedding light on the speed of learning semantic distinctions as a function of the singular values of the input-output covariances (Saxe *et al.*, 2018). This proves the relevance of input statistical signature, but nothing is said about the emergence of particular topological patterns in the connections between neurons in the neural network.

Thus the task here is to inspect the topology of the neural system in response to input data sets with sharply different statistical morphology.

Slighly more verbose

In the full swing of Kashtan and Alon (2005), in the following a simple feed forward *deep* neural network is inspected as breeding ground for *network motifs* emergence. The model and the methods here to be deployed however differ from those presented in the cited paper, inasmuch

- Here the neural system set up is intended to solve a classification problem;
- There the system is set to evolve by using **genetics-inspired** algorithms, whilst here the neuronal network undergoes the standard learning process of parameters

(i.e. weights of the neurons connections, strengths of the edges in graph theoretic language) optimization, via back-propagation of classification error;

Although some methods differ, the foremost core of investigation can be summarized as follows: Synthetic data are used, these are then fed to an artificial neuronal network. In the last stage, emergence of network motifs is inspected.

It is not expected that the results produced match those reported in previous work on network motifs (Kashtan and Alon, 2005, and references therein), by the observations made above. But it could be however interesting to investigate the response in terms of topology of a network that *evolves* under the guidance of standard optimization algorithms used in Machine Learning. In particular, it may seem obvious that some kind of pattern emerge in the neural system and, by another viewpoint, it could be argued that the spectrum of such patterns is constrained to the densely fully connected architecture of the network itself. On the other hand it seems also reasonable to expect that these motifs, though being subjected to architectural constraints, can not be independent on the particular *statistical signature* of the data set.

This is indeed the whole point of training the model with synthetic data. By generating a toy data set, one has control on the statistical internal morphology of the data, and hence the motifs emerged may be imputed to such statistical trace. Or, almost, it is possible to verify which motifs are more present after training the model with one of the available data sets.

In a nutshell, it would be interesting inspect the veracity of the following heuristic idea: The statistical signature of a data set influences the parameters evolution in some way, as demonstrated by Saxe, *et al.* (2018), and this evolution drives the system, intended holistically as the neuronal network, to settle in a parameters configuration (that is: strengths of the edges of the resulting weighted graph) that somehow reflects the statistical signature of the data set fed to the system. For example, the binary tree data generating structure turns out to show a hierarchical structure.

The ultimate goal, from a real world perspective, could be an preventive optimization approach. While a neat choice of the parameters of course helps, if one knows the statistical properties of the data set, the could infer the kind of motifs that shall surface in the learning stage. Thus a preventive architecture design to encounter such an hypothetical outcome may resolve in a **faster convergence** to the optimum, thus saving learning time, which is one of the downsides of Deep Learning.

2 Data sets

Some previous work is followed for the zero stage (Kemp and Tenenbaum, 2008 and Saxe *et al.*, 2018). A difference however is that in these cited publications is that there synthetic data consists in categories, and the learning system should guess each item's feature. This leads to a difference in the covariance structure (cfr. Figures 3 below and for example Figure 9 in Saxe *et al.*, 2018), and is due to the fact that, for example in the binary tree data structure, in the present case correlation patterns tie together, in some extent, all of the nodes in the binary tree.

The main difference is that here each node of the PGM generated is associated with a *feature*, whilst class labels are assigned according to whether a data item matches some of the previously created, in the case of the binary tree, and according to which one of the independent clusters is selected in the case of the second data set.

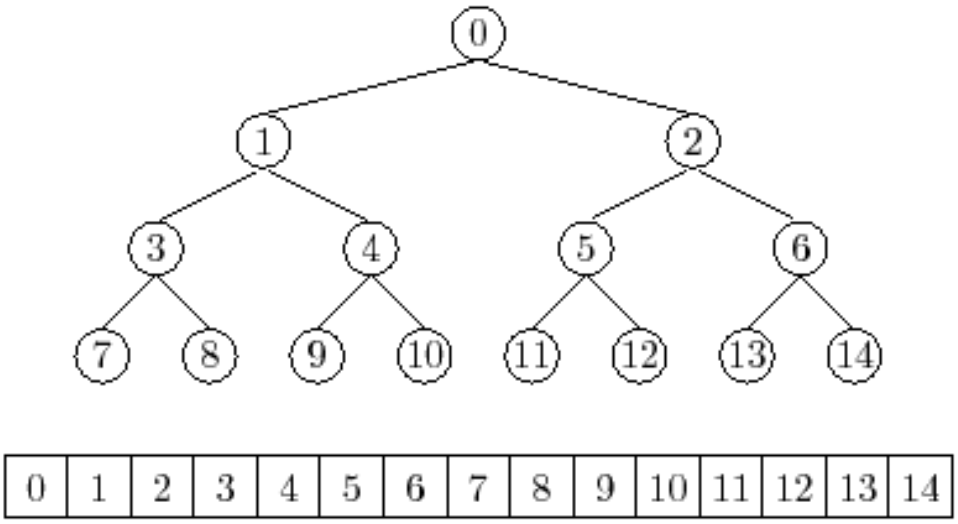


Figure 1: From OpenDataStructure site.

In the real world, data often come as rows of a so called *design matrix*. Each one datum is then an array of some *features* characterizing the observation. Each one of these features[1] is a **random variable**, distributed according to some unknown distribution. In this spirit the data set can be characterised by a multivariate probability distribution.

An interesting way to represent multivariate distributions is provided by **probabilistic graphical models** (PGMs). These models represent the causality of the random variables involved by means of a graph: Nodes encode random variables, while edges encode the relationships that tie these variables together (Chapter 16 of the Deep Learning Book by Goodfellow *et al.*, 2016). In this language, the sets of synthetic data that may be interesting for the sake of the present work can be formalised as PGMs and also by this representation the statistical structure may be more evident.

2.1 Binary tree data set

The first example is the **binary tree** data generating structure. The root node is a random variable, which attains one among the values $\{-1, +1\}$ with equal probability $p = 0.5$. According to the outcome of such random variable, the children inherit the ± 1 value according to some probabilistic decision rule and in the same fashion the children of the children, and so forth down the dynasty. As the user specifies the depth D of the tree to be created, the tree structure, i.e. the collection of the $N = 2^D - 1$ random variables that constitute one data vector are indeed one data instance. An advantage of the PGM representation is that it renders graphical visualisation ease: Data are often many-dimensional, i.e. points in an N-dimensional space.

In this case the collection of M of such vectors could be thought as an ensemble of living species. The root node determines whether one item (pattern, data example) can move or not. The children of the root node determine whether *if it moves, does*

[1]What in the language of Machine Learning is dubbed *feature* can be translated in Mathematical/Physical terms as *entry* of a vector. Data are often represented as vectors, i.e. a data example is a collection of features. Further, what is called *pattern* is essentially a data item, that is: one possible outcome of a sampling from the data generating distribution. **Not** to be confused with the pattern intended as *network motif*.

it swim? or *if it does not move, does it have bark?*, and so forth. Clearly, the levels deeper in the tree structure, bear more information about the data items. Better: In the following, it is shown how the choice of a particular level resolves in the presence of more or less classes. As one considers the leaves level, then **all** of the nodes of a tree (i.e. all of the features in a pattern) **must** equal, for two items to belong to a given class. On the other hand, if one considers a shallow level in the tree structure, the nodes which must equal for two data vectors to belong to the same class, are all the nodes **up to the last node** of the level considered. All the subsequent nodes could in principle attain different values but this does not matter. As an example, if one wants to differentiate living things based on the fact that such items *can move or not*, what matters is the value attained by the root node. Then whether two items are respectively a whale or a deer, this does not affect the belongingness to the *living thing that can move* super-class. In contrast, if one has to differentiate *living thing that move* based on the fact that such an item *does swim or not*, then a further level of detail is needed. Such a finer granularity is encoded by the values the nodes of the next levels attain. If the left children of the root node happens to inherit its $+1$ value, that means that, other that *being a moving living thing*, that item *does swim*. Therefore, the second tree level encodes this subsequent level of detail. The more detail is embedded (the higher level is chosen), the more the possible classes the data examples may in principle belong to.

The rationale behind such a data generator is first and foremost related to its transparency and statistical structure clarity: There is no real-world consistency in such data, but in this fashion it is easy to perform classification on them. As explained below, one single pattern generation happen to be a value diffusion down to the tree branches. In this way, one ends up with a N-dimensional binary array, in which (**important**) many of the slots bear the -1 value. The $+1$ values on the other hand lays in correspondence of the slots associated with those nodes which happen to represent a positive answer to the distinction question associated with that node. Consistently with the discussed example: if the living thing encoded in such a $N = 15$ dimensional vector is a moving thing (roughly speaking, an animal), then the root node has the $+1$ value, which in turn means that the 0th slot in the data vector has such value. If this is a water animal, it swims, then the left child of the root node has inherited the $+1$ value, then the slot 1 in the data vector has the value $+1$ and it implies that the right child of root inherited the value -1, so the slot 2 of the data vector has the value -1. Assume further that other than swimming, this animal *is not a mammal*. Then the left child of the 3-labelled node has inherited the -1 value and this same value is found in the slot 7 of the data vector. It means that the $+1$ value is inherited by the right child of node 3, then in the final data vector the $+1$ value appears in slot 8.

At the end of the day, the final data vector is made up by -1s, except for these said slots, where the $+1$ value ended up in, encoding the positive outcome of those criteria associated with the respective nodes. As terminal (leaves) level, it could be imagined as the *one-hots* stratum, that is: all of the leaves attain the -1 value, except for one single leaf, where the $+1$ got to settle, as consequence of the (stochastic) outcome of all the aforementioned decisions. This lonely $+1$ determines the final category in which the data vector fits in, **as one sets the leaves level to be the distinction granularity**. In such case, for two vectors to belong to the same class, it must be that **all of the features** equal. Otherwise, it could in principle be that a whale, echoing the previously discussed example, has the root node positive, but in another data row

it could be negative. This would mean that a whale is a *not moving living being* that *swims*. So, in the label generation stage, one shall differentiate according to all the nodes of the level under consideration **and** all of their ancestry.

2.1.1 *Single pattern generation*

One pattern is the collection of *all* the node values of the array-represented tree (that entity formerly dubbed a *data vector*). As an example, to the non-leaves nodes are associated decision rules, intended to discriminate samples (e.g.: *does the object move?*, which can be answered with *yes* or *no*, ± 1, is the primal decision rule, i.e. axis along which one can set distinctions). The initial value of the root node is inherited and eventually flipped according to probabilistic decision rules with respect to a fixed probabilistic threshold ϵ.

In this spirit, referring again to Figure 1, the (non-leaves) nodes ranging from 0 to 6 encode decision rules, (leaves) nodes indexed with $i = 7, \ldots 14$ represent the final category of that particular pattern. The following criteria are implemented:

(1) The probabilistic threshold is fixed a priori. The smaller its value, the less variability in the data set.

(2) Root attains the values ± 1 with probability $p = 0.5$.

(3) Root's children attain values $+1$ or -1 in a mutually exclusive fashion. The following convention is adopted: *if the root node attains the value* $+1$*, then the left child inherits the same value. Else, the left child attains the value* -1 *and the right child has assigned the value* $+1$.

(4) From the third level (children of root's children), the progeny of any node that has value -1 also has to have -1 value. On the other hand, if one node has value $+1$, its value is inherited (again mutually exclusively) by its children according to a probabilistic decision rule. This enforces the one-to-one correspondence between a pattern and the belongingness to a category, consistently with the ancestry of the leaves.

The aforementioned probabilistic decision rule is a Metropolis-like criterion: Sample a random variable $p \sim U([0, 1])$, then, given the probabilistic threshold ϵ,

- If $p > \epsilon$, the left child inherits the $+1$ value, and the right child, alongside with its progeny, assume the opposite value;
- Else, is the right child to assume the value $+1$.

2.1.2 *The complete data set*

Repeating the above procedure M times, one ends up with a data matrix $\boldsymbol{X} \in \{-1, +1\}^{M \times N}$, i.e. each row of $\boldsymbol{X}$, $\boldsymbol{x}^{\mu}$, $\mu = 1, \ldots, M$, is one single N-dimensional data vector, in the same terminology as above: a N-featured data vector (one pattern).

To complete the creation of a synthetic set of data, one needs the *label* associated to each one of the data items. Here the choice of the probabilistic threshold ϵ turns out to be crucial. The higher this quantity, the more the total number of different classes

the data example may fall into. On the other hand if ϵ is small enough, there is low probability of flipping a feature value, then it is more likely to observe repeatedly the same exact configuration.

The major drawback of the distinct categories population has been observed to impact on the **learning dynamics**.

To create the labels, encoded as *one-hot* activation vectors, one arbitrarily assumes the identity matrix to be the labels matrix. Then the whole data set is explored in a row-wise fashion. Since the data set has a **hierarchical structure**, it is possible to select the **granularity** of the distinction made in order to differentiate patterns in different classes. It depends on the choice of a level in the binary tree: If the level chosen is high (far away from the root node) then one ends up with a fine-grained distinction. On the other hand, if the level chosen is low, the distinction is made according to *super-classes*, e.g. whether a given object *can move*. The finer the granularity, the more detailed the distinction between patterns. Obviously, in this latter case the data set exhibit a greater number of distinct classes.

By this observation, the label matrix is created according to the level of distinction chosen. The node values to be considered (i.e. the entries of each $\boldsymbol{x}$ data vector) are all those that encode the values of the nodes up to the last one of the level selected. Referring again to the tree in Figure 1, if it suffices to identify the *move or not* alongside with the further *if it moves, does it swim?* and *if it does not move, does it have bark?* distinctions, then one should consider the nodes 0, 1, and 2. Hence to determine whether two data items fall in the same category, we check that all the first $2^{L+1} - 2$ nodes have the same value. Here $L = 1$, in fact we consider nodes $i \in [0, 2^{L+1} - 2] \equiv [0, 2] = \{0, 1, 2\}$.

By thus doing the data set is generated. The matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$ are saved to a proper data structure which can be easily managed by the program that implements the artificial neural network described below.

2.2 Independent clusters data set

The generation of the second data set is performed as follows: Generating some cloud of points distributed according to a bivariate Gaussian distribution, with means spread apart and covariances sufficiently small, in such a way that the points of different groups do not overlap with the others. The 2-dimensionality has of course nothing to do with the number of features, which as said before is the total number of points generated, that is the nodes of the probabilistic graph representation. This 2-dimensionality serves solely to draw the PGM and subsequently to partition the graph.

Once points are generated, are turned in a fully connected graph, i.e. create edges between each pair of nodes. In the spirit of the *simulated annealing* algorithm, here it is imagined that such a fully connected graph is a sort of mineral structure, and it is in order to increase the temperature, to simulate a melting process that destroys some of the over-abundant edges, according to some metric, for example the distance between points. For this reason it comes handy the 2-dimensional representation: Distance is simply the norm of the vector from a node to another. The distance for which the edge is removed is temperature-dependent: the higher the temperature, the shortest the maximum edge length allowed. At the end of this *simulated melting* process, it is expected the graph to exhibit some independent components, provided the melting schedule is properly set. Moreover these independent groups are not fully connected

Algorithm 1 Binary tree. Single feature generation

Compute $N = N_{\text{leaves}}$, $n = N_{\text{not leaves}}$. M is a free parameter.
$\texttt{tree} = \mathbf{0}^N$
Define a small $\epsilon \sim O(10^{-1})$ as probabilistic threshold
Value of root $\eta^{(0)} \sim U(\{-1, +1\})$
if Root node has value $+1$ **then**
    The left child inherits the value $+1$
    And the right child inherits the value -1
else
    The left child inherits the value -1
    And the right child inherits the value $+1$
end if
for All the other nodes indexed $i = 1, \ldots, n$ **do**
    if Node i has $+1$ value **then** Sample $p \sim U([0, 1])$
        if $p > \epsilon$ **then**
            Value of the left child of i = value of i
        else
            Value of the left child of i = flip the value of i
        end if
    else
        Both the children of i inherit its -1 value
    end if
end for
$\boldsymbol{x}^\mu \leftarrow$ values generated, $\quad \mu = 1, \ldots, M$

Algorithm 2 Binary tree. *One-hot* activation vectors, i.e. labels

Choose level of distinction L
$\boldsymbol{Y} = \mathbb{I}$
for $\mu = 1, \ldots, M$ **do**
    for $\nu = i, \ldots, M$ **do**
        if the first $2^{L+1} - 2$ entries of $\boldsymbol{x}^\mu$ and $\boldsymbol{x}^\nu$ equal **then**
            $\boldsymbol{y}^\nu \leftarrow \boldsymbol{y}^\mu$
        end if
    end for
end for
for $i = 1, \ldots, N$ **do**
    if $\boldsymbol{Y}[:, i]$ equals $\mathbf{0}^N$ **then**
        Eliminate column i of $\boldsymbol{Y}$
    end if
end for

within themselves. The melting schedule is designed in a way to remove some of these intra-edges. This simulates the random variables of each group not to be dependent on all of the others in the same could. **Note** that, unlike how exposed in Kirkpatrick, *et al.* (1983), in this melting simulation there is not, strictly speaking, an *optimization* perspective inasmuch what matters is the removal of some edges. The physics of the procedure could be revised.

2.2.1 *Single pattern generation*

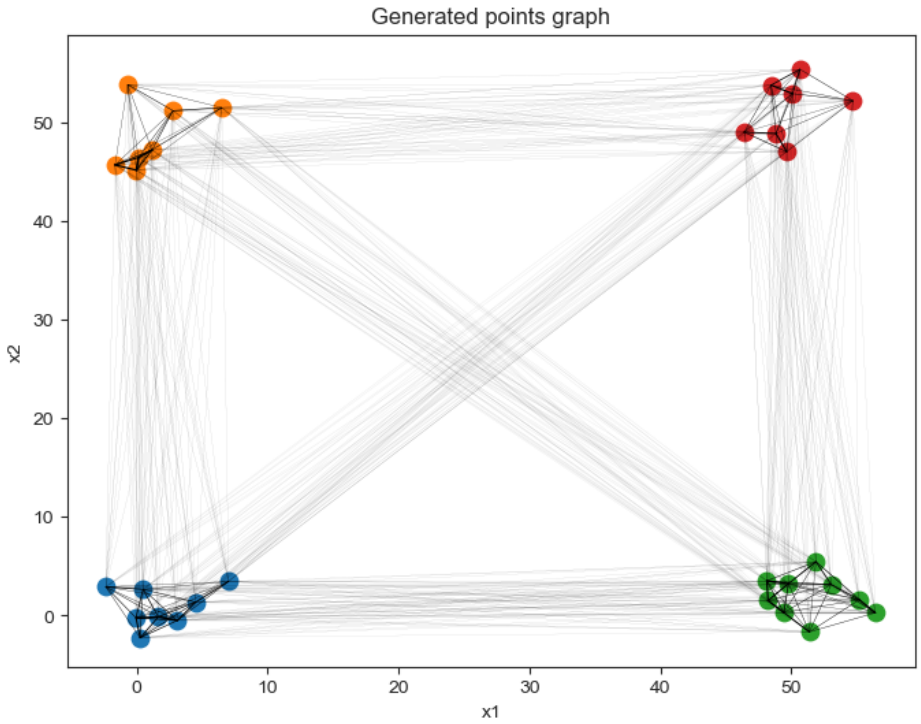
Once the independent clusters come to form, it is in order to assign each of the nodes a **topological ordering** in such a way to perform the **ancestral sampling** (Chapter 16 of Goodfellow, *et al.*, 2016). Since the graphs are directed, in the edges data structure created each edge is in the form of a couple (i, j), i.e. edge from node i to node j. Then if one node appears only on the left slot of such representation, it has topological order 1, in that no edge ends up at that node. Conversely, each node appearing on the right has almost one ancestor. For each edge then, each right node is saved to a proper data structure, and it is kept track of the ancestors of each node. In this way it is possible to assign both the topological order and to keep a list of all the ancestors. It will be useful in the stage of sampling to dispose of such list.

As a zero model however it is done as follows: A data item is initially initialized with all the features values of -1. Since each vertex in the graph encodes a feature, and the belongingness of each vertex to a group is an information known from the points generation stage, an integer ranging from 1 to the number of classes $N_c = 4$ is sampled uniformly. The nodes corresponding to this label number are assigned different values, according to their topological order. This is trivial to do since for each vertex belonging to the selected group one simply puts in the corresponding slots in the data vector the topological order of such vertices.

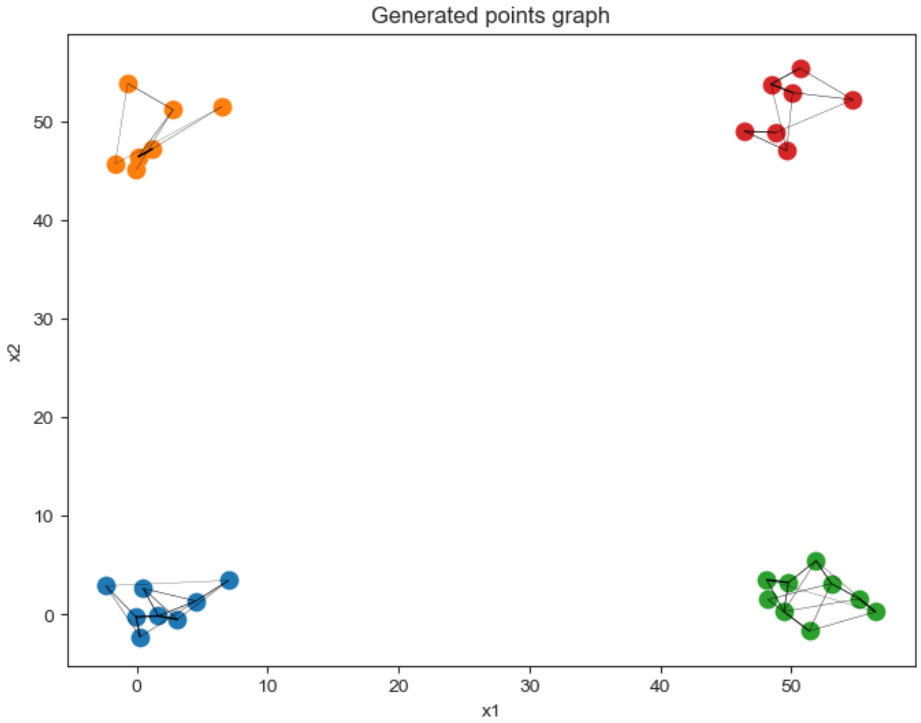
A further improvement could be rather this approach: once a label is sampled, one could sample from the distribution $p(x_i)$, for the vertices with topological order 1 in that cluster. The values associated with nodes having topological order 2 is still sampled from that distribution, but must be conditioned to the values sampled for their ancestors (nodes of order 1), i.e. $p(x_i \,|\, \text{Ancestors}(x_i))$. This is explained by recalling the very purpose of graphical models: to show (even graphically) the *causality* of the random variables involved. As distribution it could be chosen a Gaussian with mean zero and variance proportional to the degree of that node. Gaussian is believed to fit since nearby features are expected to have similar values (Kemp, 2008, but differently from this work, here one does not generate the features vector, hence sampling from the multivariate Gaussian having zero mean and variance dependent on the inverse of the Laplacian matrix of the graph. Here it suffices to sample a value for a single node, and hence the degree of a node could be a good compromise, being such quantity one of the ingredients of the Laplacian).

The following nomenclatural convention is adopted: vectors are ***bold-italic*** typefaced, such as one single data item. Since all the entries are random variables, in this case one refers to a single entry of the data item vector as x_i, even if it is widespread to typeface random variables with roman capitalisation. So in the following $p(x_i)$ refers to the probability associated with the outcome x_i of the random variable that is encoded by the i-th entry of the data item.

To sample from the conditional $p(x_i \,|\, \text{Ancestors}(x_i))$ the following rationale may



(a) Fully connected graph



(b) *Molten* graph

Figure 2: Two subsequent stages of the clusters data set generation

be implemented: The distribution is referred to all the nodes up to i, then could be viewed as a multivariate distribution. Then a value is sampled from that multivariate distribution, but keeping constants the values of the random variables sampled yet. As

an example: Assume that node 3 of cluster 1 is to be assigned the value x_3 and that Ancestors$(x_3) = [1, 2]$. Then the pdf to sample from is

$$p(x_3 \,|\, x_1, x_2) \sim \exp\left(-\frac{1}{2}(x_1, x_2, x_3)^T \, \mathbf{\Sigma}^{-1} \, (x_1, x_2, x_3)\right) \tag{1}$$

with $\mathbf{\Sigma} = \text{diag}(k_i)$, $i = 1, \ldots, 3$, being k_i the degree of node i. The above formula may be broken in products, owing to the fact that the variance matrix is diagonal, that is

$$\begin{aligned} p(x_3 \,|\, x_1, x_2) &\sim \exp\left(-\frac{1}{2}\frac{x_1^2}{k_1^2}\right) \exp\left(-\frac{1}{2}\frac{x_2^2}{k_2^2}\right) X_3 \\ X_3 &\sim \mathcal{N}\left(0, k_3^{-2}\right) \end{aligned} \tag{2}$$

the first two factors being **the values** that the Gaussian probability density function attains at the values sampled for the ancestors x_1 and x_2 and the third factor is the value sampled from the Gaussian having zero mean and variance k_3^2.

2.2.2 *Complete data set*

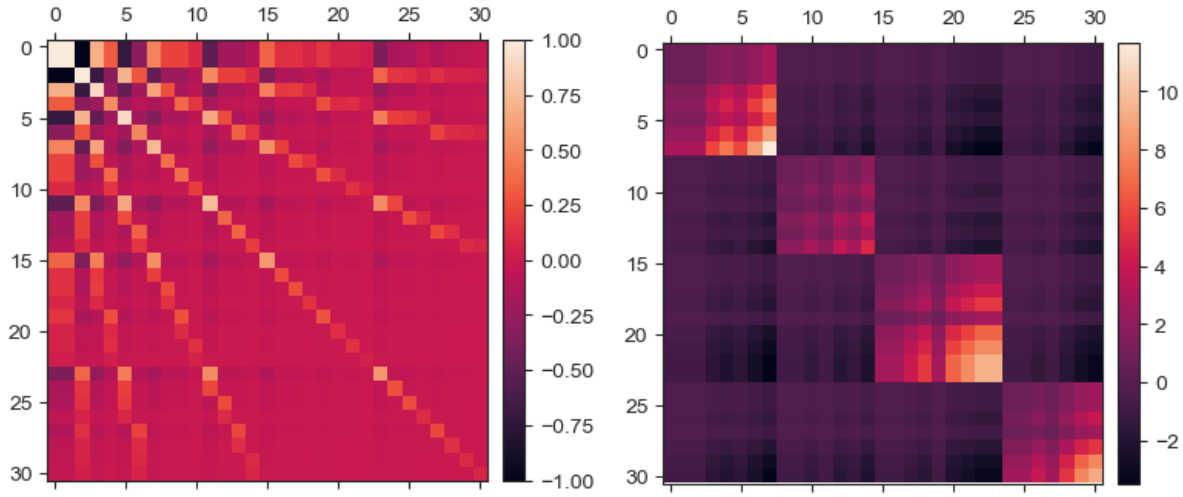
This procedure is repeated many times as specified by the user. Here a good number is, as in the case of binary tree, $M = 2000$ items. In the complete data set hence one has features in which the only values not being -1 lay in correspondence of the indexes of the data array that match with the nodes of the graph that belongs to the category given by the label of that feature. Labels are again *one-hot* vectors. For example, assume that the first cluster is selected. If this first cluster comprises the vertices ranging from 1 to 5, where node 1 has order 1, 2 and 3 have order 2, 4 has order 3 and five has order 4, then that data item has values $[1, 2, 2, 3, 4, -1, \ldots, -1]$ and the corresponding label is $[1, 0, \ldots, 0]$.

Algorithm 3 Independent clusters. Simulated melting to *partition the graph*

1: Choose the number of classes N_C
2: Set $\boldsymbol{\mu}^{(k)} \in \mathbb{R}^2$, $\mathbf{\Sigma}^{(k)} \in \mathbb{R}^{2\times 2}$, $k = 1, \ldots, N_C$
3: Generate $\boldsymbol{X}$ s.t. $\boldsymbol{x}_i \sim \mathcal{N}(\boldsymbol{\mu}^{(k)}, \mathbf{\Sigma}^{(k)})$, $i = 1, \ldots, M$
4: Include the indexes of the points generate in a list, which is the set of the vertices $\mathcal{V}$ of the graph $\mathcal{G}$
5: Fully connect the vertices to form a fully connected graph and group the vertices and the set of the edges $\mathcal{E}$ in the graph data structure, $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$. **Note** that since 2-dimensional coordinates will be useful, $\mathcal{V}$ is a dictionary of keys (nodes indexes $i = 1, \ldots, M$) and values (list with the point coordinates, $(x_i^{(1)}, x_i^{(2)})$).
6: **for** T increasing **do**
7: **for** All the edges $e = 1, \ldots, |\mathcal{E}|$ **do**
8: **if** Length of edge $e > \frac{1}{T}$ (for example) **then**
9: Remove edge e
10: **end if**
11: **end for**
12: **end for**
13: Plot the remaining edges and check if only independent fully connected components have survived.

Algorithm 4 Independent clusters. Single pattern generation

Here i indexes a single random variable. This kernel is used as many times as the number of samples the user wants to generate. $\boldsymbol{x}$ is the whole data item, initialised with each slot set to -1.
Set $\boldsymbol{x} = \{-1\}^N$
Sample $L \sim \mathcal{U}(\{1, \ldots, N_c\})$
for all the vertices $i = 1, \ldots, n_k$ in cluster L **do**
    if Topological Order of i is 1 **then**
        $x_i \sim p(x_i) \sim \mathcal{N}(0, k_i^{-2})$
    else
        $x_i \sim p(x_i) \prod_{j \in \text{Ancestors}(x_i)} (4\pi\, k_j^2)^{-1/2} \exp\left(-\frac{1}{2}\frac{x_j^2}{k_j^2}\right)$
    end if
end for
$\boldsymbol{y}_i = \text{one-hot}(L)$



(a) Binary tree data set

(b) Clusters data set

Figure 3: Covariances matrices visualisation of the two different data sets.

3 Neural network model

The two data sets are hand-crafted to have the same number of features and overall classes. This choice could be flexible however. The program that implements the simulation is capable to self-adjust the number of neurons according to the input size and the categories number. First table below gives an overview of the network architecture. The second table reports the setting of the optimization algorithm used.

4 Results

It is not striking that the model learns in just few iterations to classify correctly the samples. Data sets are easily tractable, in particular the clusters data sets. Here the systems sees non--1 values in correspondence of a class, then it is straight forward to learn what is the underlying pattern, regardless the fact that the non--1 values are

Architecture		
Layer	Units	Activation
1 (input)	$N = 31$	-
2 (hidden)	20	ReLU
3 (hidden)	10	ReLU
4* (output)	4	Softmax

NASGD	
Learing rate	0.01
Decay	10^{-6}
Momentum	0.6

real numbers, if one implements an ancestral sampling using normal distributions, or as in the present case, where features are simply the topological order of the vertices.

4.1 Binary tree data set

The simple neural network illustrated is trained with the data set. A 0.7 fraction is used for training, a 0.1 of which is used as validation set, and the 0.3 kept apart before is used as test set. Being the data set linearly separable, in few epochs the accuracy metric attains the top value of 1.0.

Test set accuracy	1.0
Test set loss	0.0025

As final result, weights and biases distributions are also reported for the trained network, alongside with the connections strengths visualisation, Figure 12b.

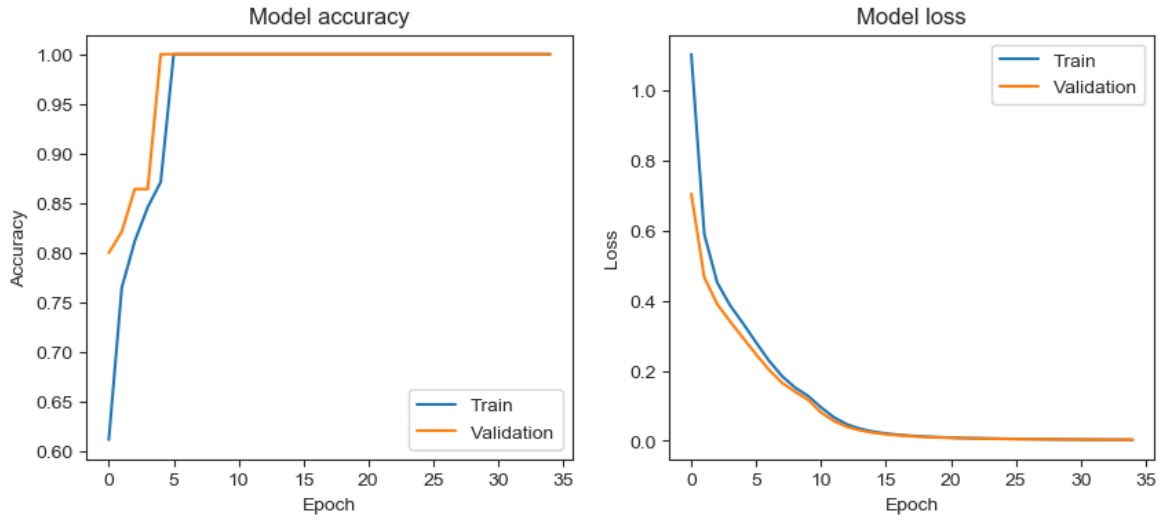
4.2 Independent clusters data set

Since here the map to be learned from the *states space* (i.e. the domain of the data) to the *target space* is straightforward (it does not take a neural network to understand the game), learning takes two epochs to be over, with a smaller value of the test loss.

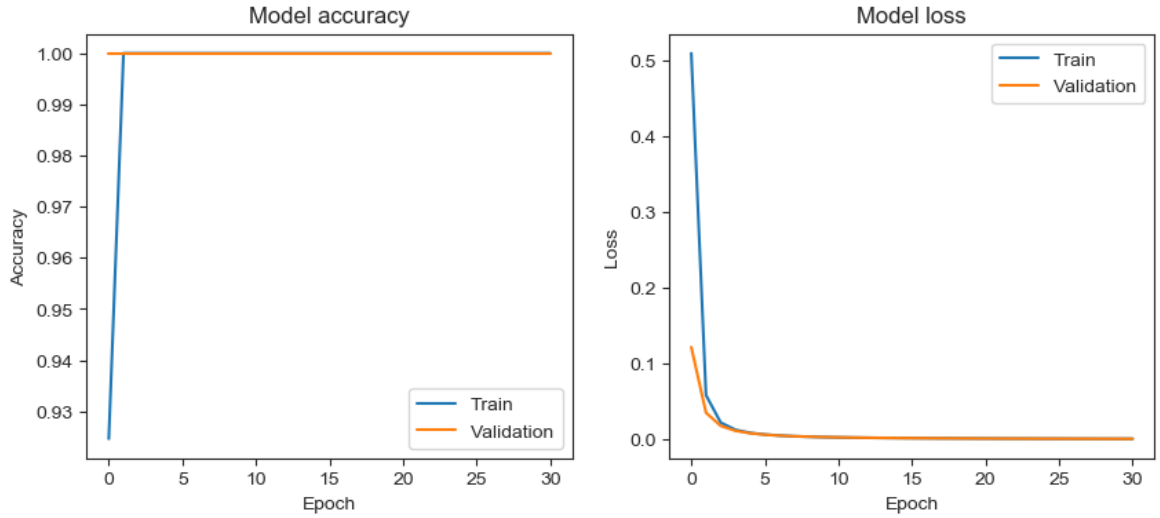
Test set accuracy	1.0
Test set loss	0.00065

4.3 Modularly varying goals

It is interesting also to inspect the response of the system once exposed to two different data sets formerly discussed. What is immediately clear is that more motifs are observed, likely due to the fact that the neural network undergoes a longer training stage and then the weights are updated forward the optimum for a prolonged period. The fact that the cost is monotonically decreasing for both the two data sets suggest



(a) First layer



(b) Second layer

Figure 4: Frequency plots of the parameters before and after training. Binary tree data set.

that even an optimal configuration is found, the cost function hyper-surface resembles a mild hill slope, where the parameters configuration point keeps descending.

An explanation could be that the data sets used are indeed so good that the more the model is trained, the more it is specialized on such tasks. It could be in some sense an occurrence of a subtle over-fitting. Attached to this document the complete log of the training agenda is provided.

4.4 Motifs detection

The `mfinder1.2` executable program (documentation here) is exploited in the next stage, that is inspecting the system in search of significant patterns.

As an introductory note: the weights of the neural network as obtained by the Keras build-in function, are real-valued of course. The cited program only can deal with +1 valued edges strengths. Then, as an exploratory experiment, the weights exceeding (both in positive and negative value), a given cutoff threshold, are set to +1 and to

0 otherwise. By thus doing the resulting graph item can be fed to the `mfinder1.2` program.

Nodes	Motif ID	N_{real}	N_{rand} stats	N_{real} Z-score	N_{real} P-val	Unique Val	C real
4	14	25	17.8 ± 2.8	2.61	0.000	4	15.63
4	76	316	267.4 ± 21.9	2.22	0.010	8	197.62
4	204	12	3.0 ± 1.6	5.66	0.000	4	7.50
4	280	224	183.0 ± 13.9	2.95	0.000	7	140.09
4	392	424	337.3 ± 25.4	3.42	0.000	7	265.17
4	904	38	3.7 ± 2.1	16.14	0.000	4	23.76
4	2184	107	91.2 ± 6.3	2.50	0.000	8	66.92

Table 1: Results of the `mfinder1.2` program for the level 2 data set.

Nodes	Motif ID	N_{real}	N_{rand} stats	N_{real} Z-score	N_{real} P-val	Unique Val	C real
4	280	147	127.8 ± 9.4	2.04	0.010	8	129.29
4	392	373	265.6 ± 23.0	4.66	0.000	8	328.06
4	904	18	2.3 ± 1.5	10.74	0.000	6	15.83

Table 2: Results of the `mfinder1.2` program for the clusters data set.

Nodes	Motif ID	N_{real}	N_{rand} stats	N_{real} Z-score	N_{real} P-val	Unique Val	C real
4	14	56	36.0 ± 4.7	4.27	0.000	6	15.44
4	28	294	206.6 ± 17.8	4.91	0.000	5	81.06
4	204	47	9.1 ± 3.2	11.83	0.000	5	12.96
4	280	534	409.9 ± 23.8	5.22	0.000	9	147.23
4	392	1022	762.3 ± 45.9	5.65	0.000	7	281.78
4	904	95	11.7 ± 3.6	23.25	0.000	5	26.19
4	2184	298	246.0 ± 12.2	4.27	0.000	11	82.16

Table 3: Results of the `mfinder1.2` program for the MVG-undergone training.

5 Conclusions and remarks

As pointed out in Figure 11, binary tree data set-fed neural network exhibits a greater number of motifs, both in diversity and in quantity of patterns found. In particular, the *diamond* motif (id204) is the mostly occurring one.

To sketch an hypothesis, one could start from Figure 3, in which sharply different structures of the covariance matrices are appreciable. While 3a shows a periodic pattern in the covariances between random variables in the tree structure, thus meaning that there are no such variables which do not share anything with some others, Figure 3b clearly shows that there are some blocks within which correlation may be found while all of the variables inter-blocks are not correlated. Recall that neural network owe their

power and seemingly almightiness to their capacity to extract the statistical features of the data set. In turn, it seems to fit neatly that a less *trivial* statistical signature translates in a more entangled topology in the evolved system. In the binary tree, being this structure hierarchical *per se*, all of the nodes are dependent by the outcome of the root node, thus correlation embraces almost all of the random variables. On the other hand, the independent clusters data set show a simple statistical morphology, a **sparser**, glassier covariance matrix, and then it comes with no surprise that the motifs detected are less, and less statistically significant. In this sense, less abstraction is required for the model to understand the underlying nature of the data set.

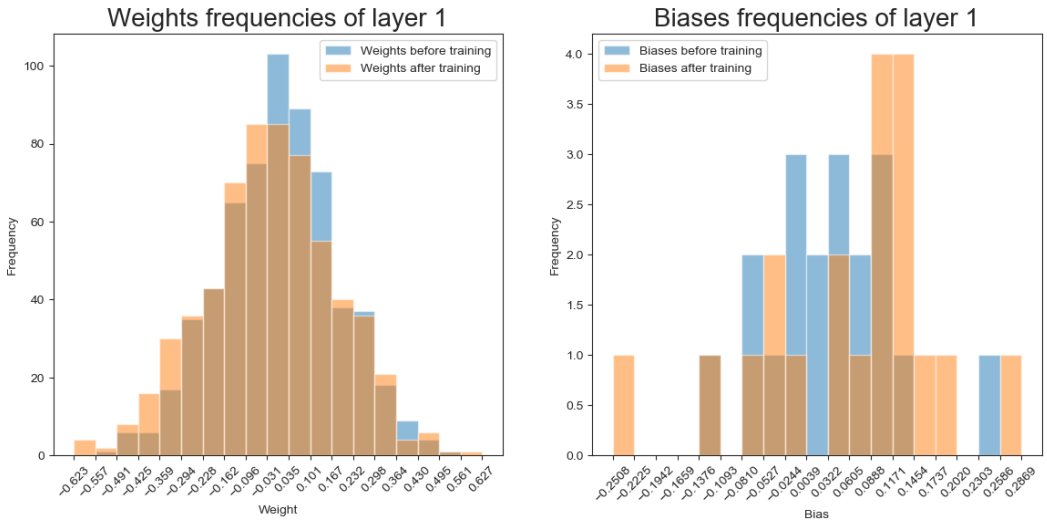
It is also worth a glance to the MVG-undergone system. In this case, a new motif showed up (id28) and again the diamond and the *bifan* (id 904 and 204 respectively) are the motifs that are more frequently observed. Referring to Figure 10c, it could be spotted a greater *diffusion* of the initial weights distribution, which means that the connection of the last layer experience a greater increase of magnitude. However, the fact that an unseen motifs pop up should not be matter of concern: If one repeats the simulations many times, he/she may appreciate some fluctuations in the results.

As a final remark, in the MVG case, even if it is observed the mentioned weights increase, there are no edges exhibiting much greater values. This may suggest that the different statistical signatures of the chosen data sets are different enough not to enforce certain network motifs, indeed related to one particular statistical structure.

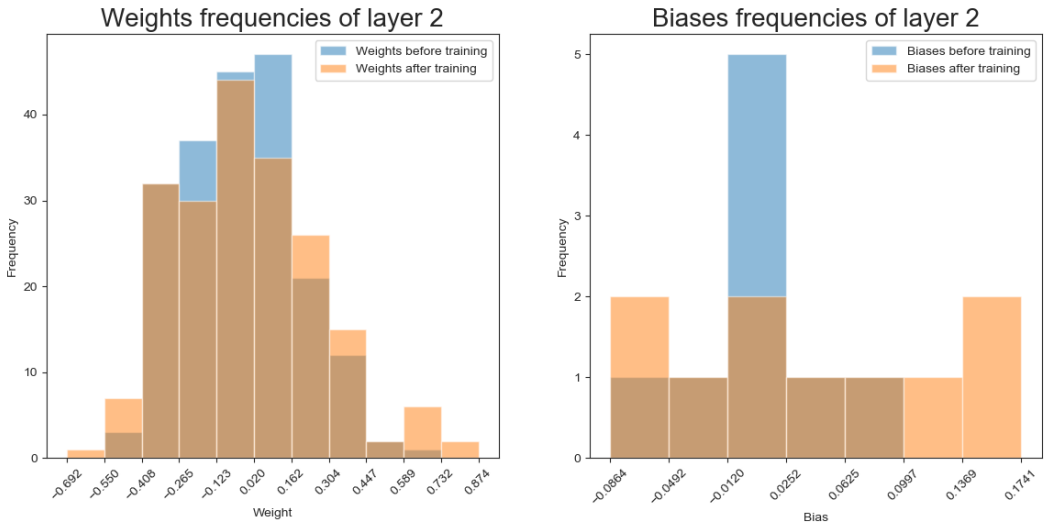
5.1 Further improvements

At the present stage the main concern can not be ignored any further: The instrument used to detect network patterns needs to be enhanced in such a way to include the edges strengths information in the motifs inspection stage. The next step in this work then is to create from scratch a procedure to accomplish such a task. Some literature is available for the sake of motif mining in weighted network, see e.g. Choodbar *et al.*, (2012) and Onnela *et al.*, (2004).

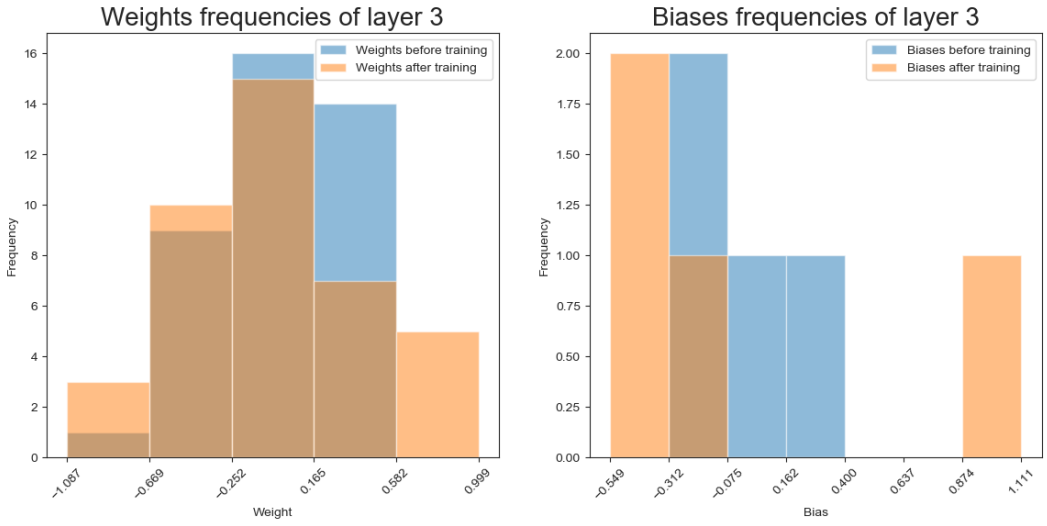
Although, another interesting perspective may be fruity: By repeating many times a random walk on the weighted graph, one may hope to find which are the more walked-through edges, thus yielding information about the preferential pathways, analogously to motifs. This approach is being worked on currently



(a) First layer

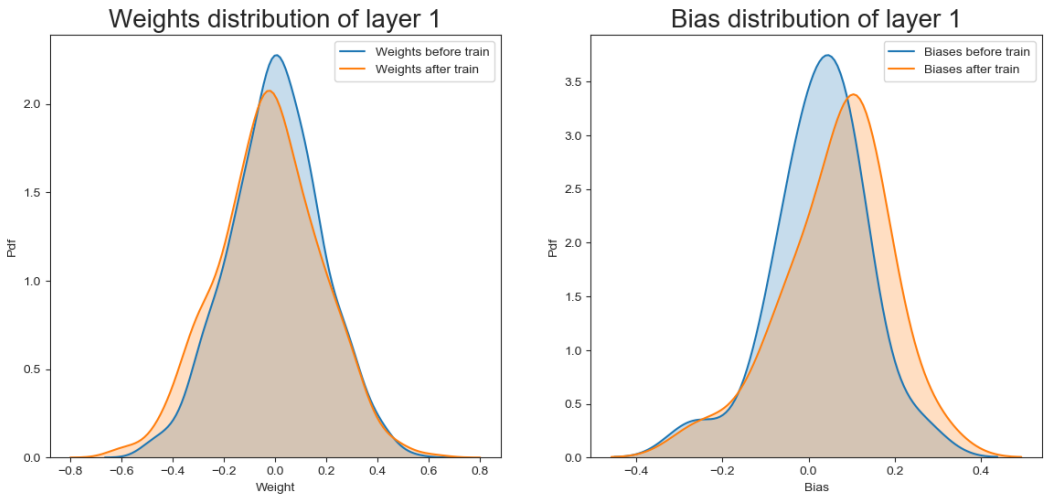


(b) Second layer

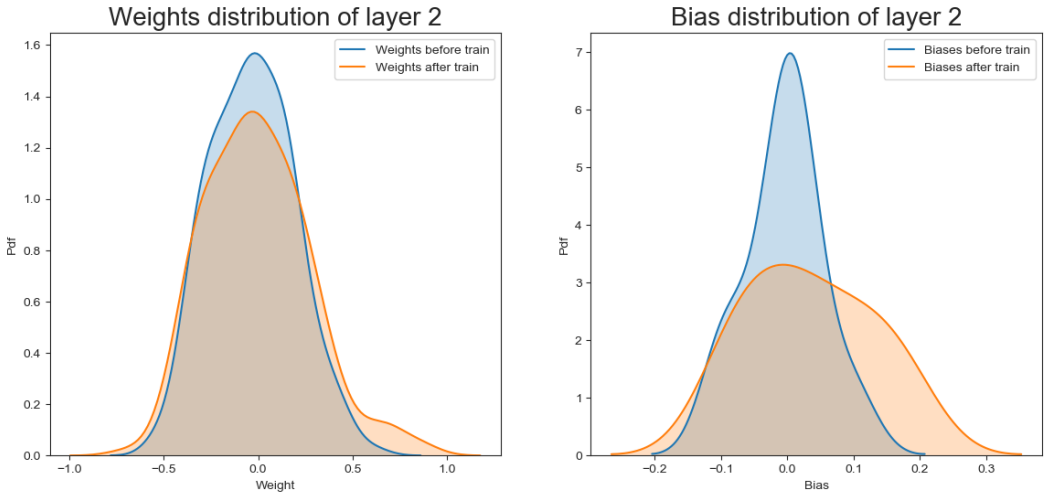


(c) Third layer

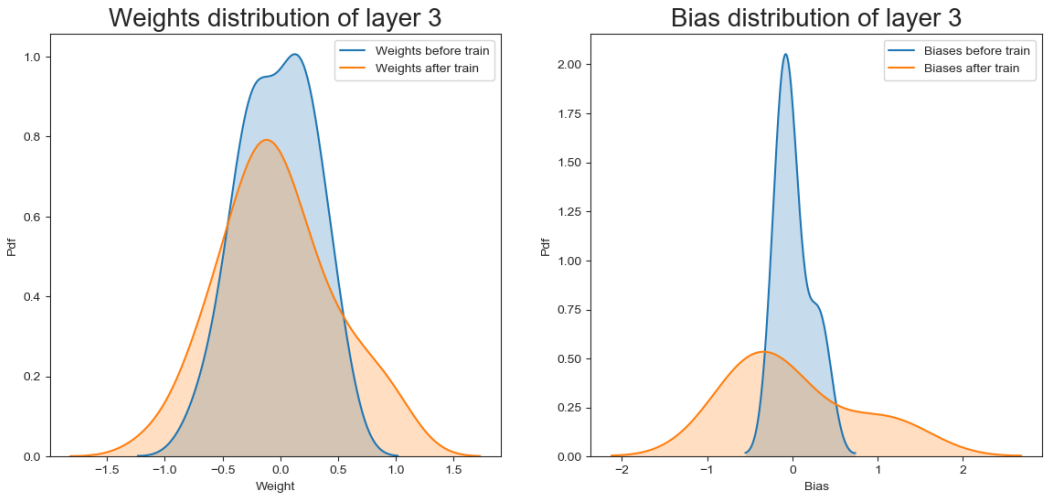
Figure 5: Frequency plots of the parameters before and after training. Binary tree data set.



(a) First layer

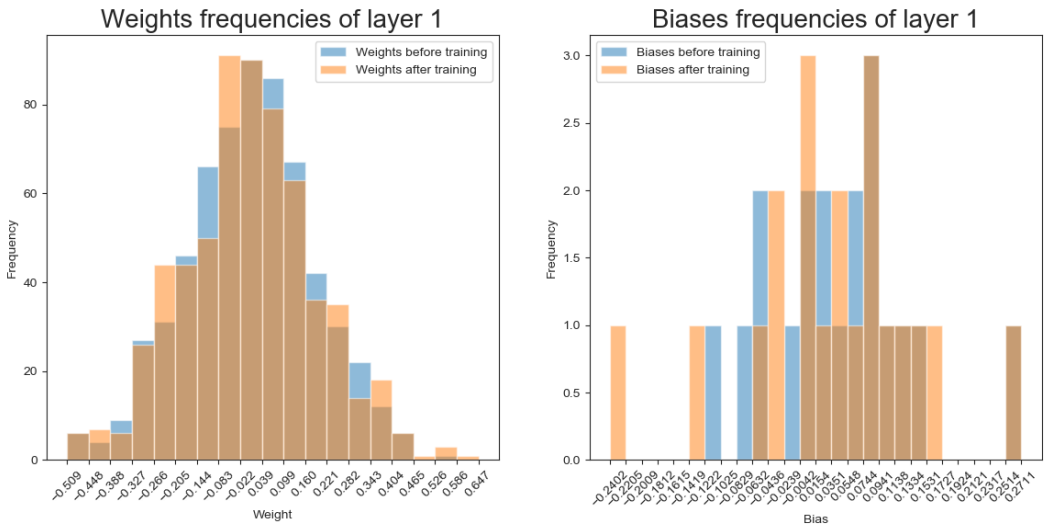


(b) Second layer

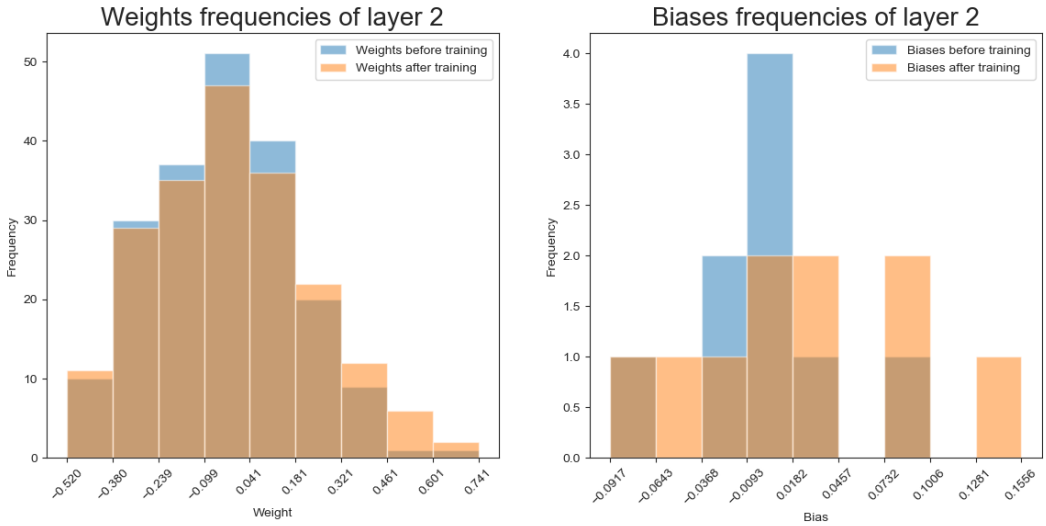


(c) Third layer

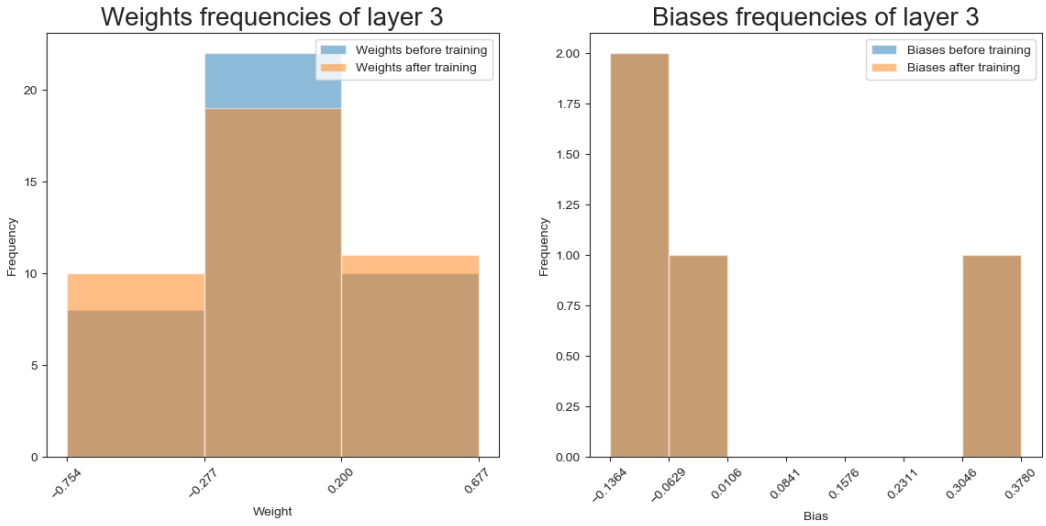
Figure 6: KDE plots of the parameters before and after training. Binary tree data set.



(a) First layer

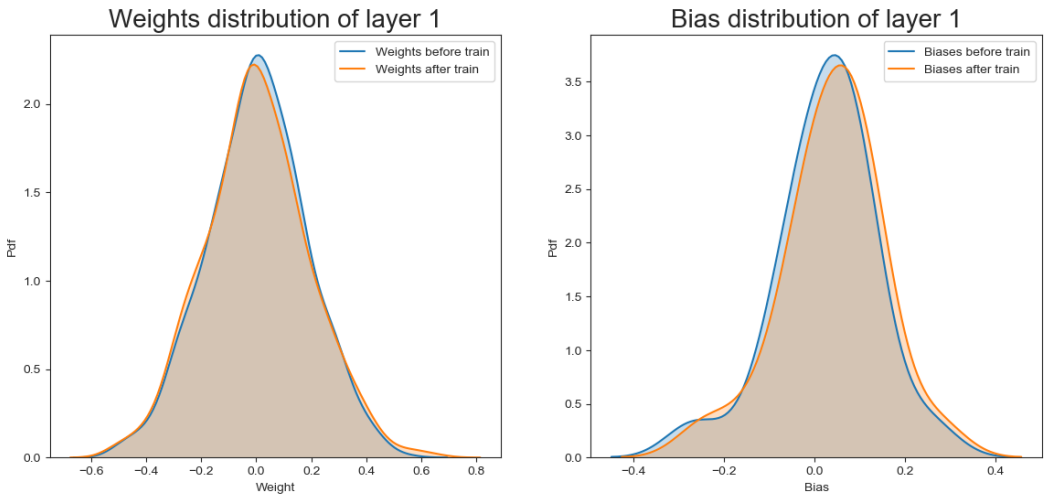


(b) Second layer

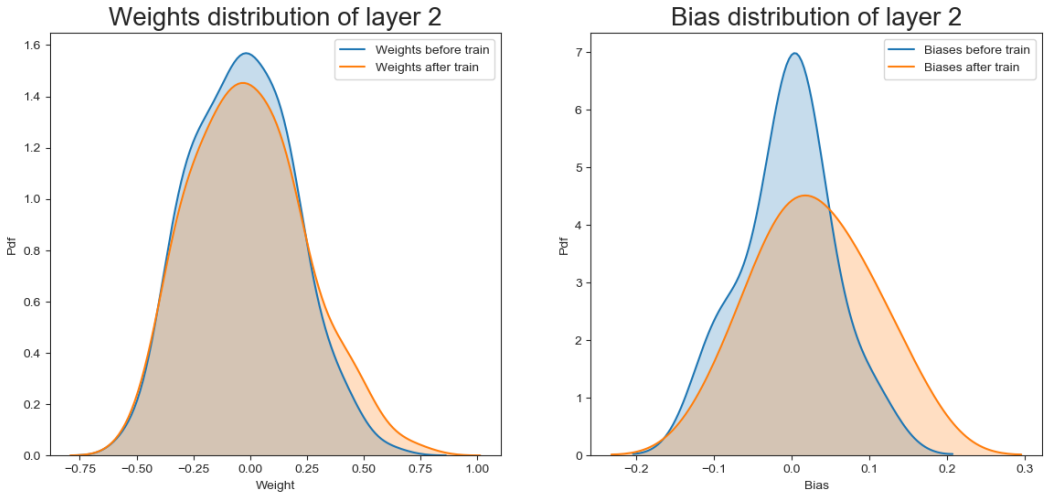


(c) Third layer

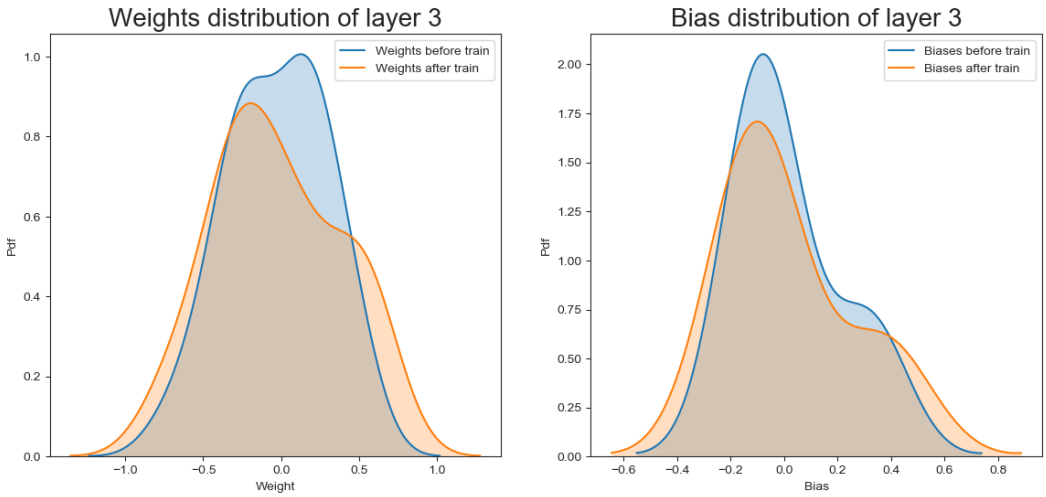
Figure 7: Frequency plots of the parameters before and after training. Clusters data set.



(a) First layer

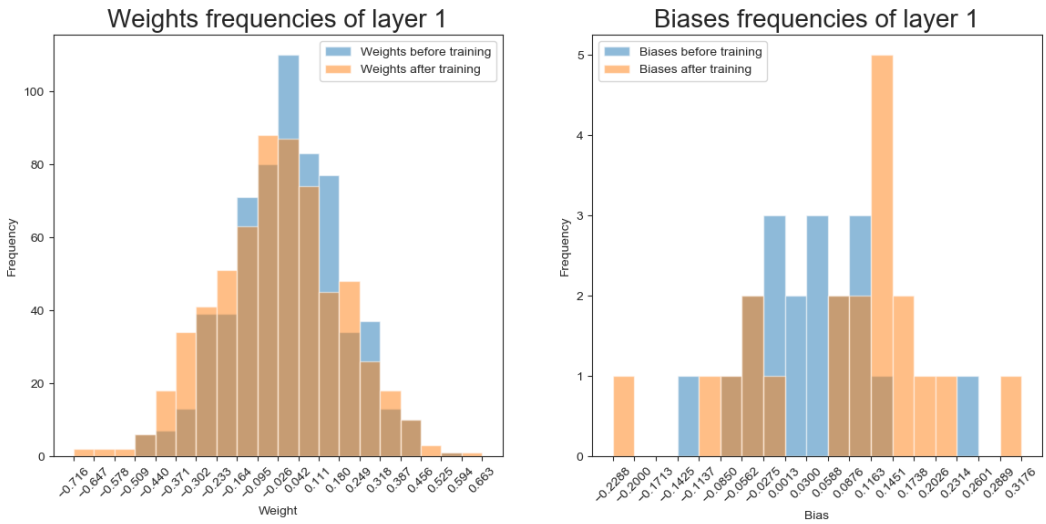


(b) Second layer

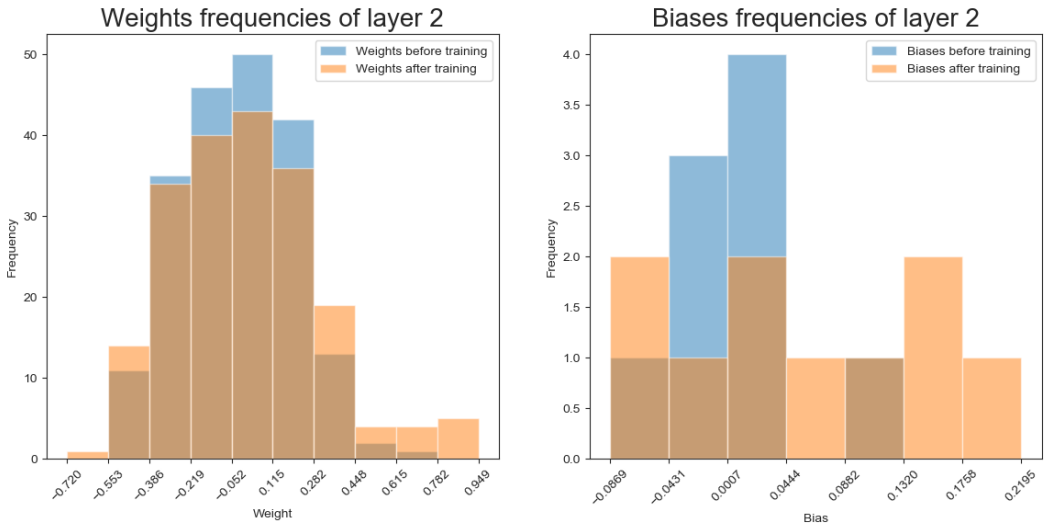


(c) Third layer

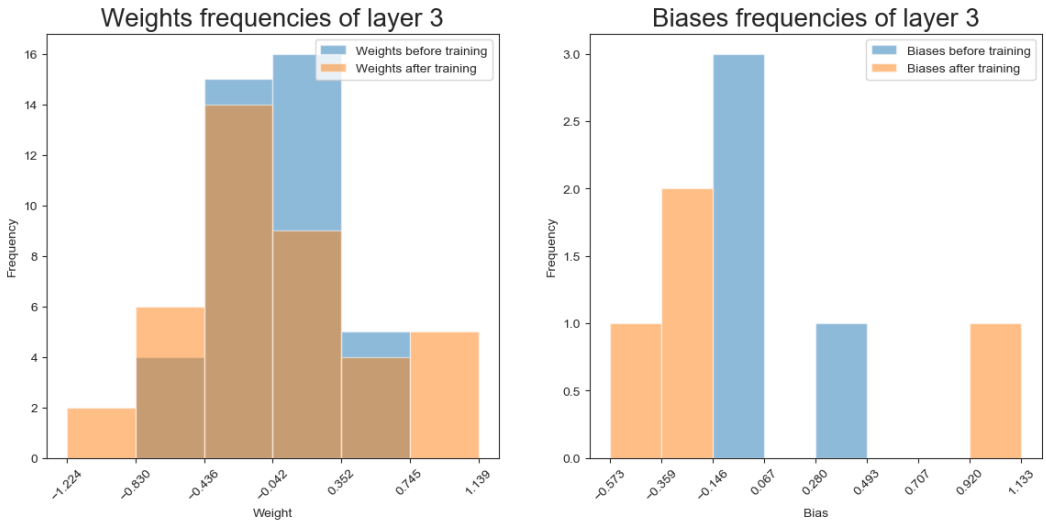
Figure 8: KDE plots of the parameters before and after training. Clusters data set.



(a) First layer

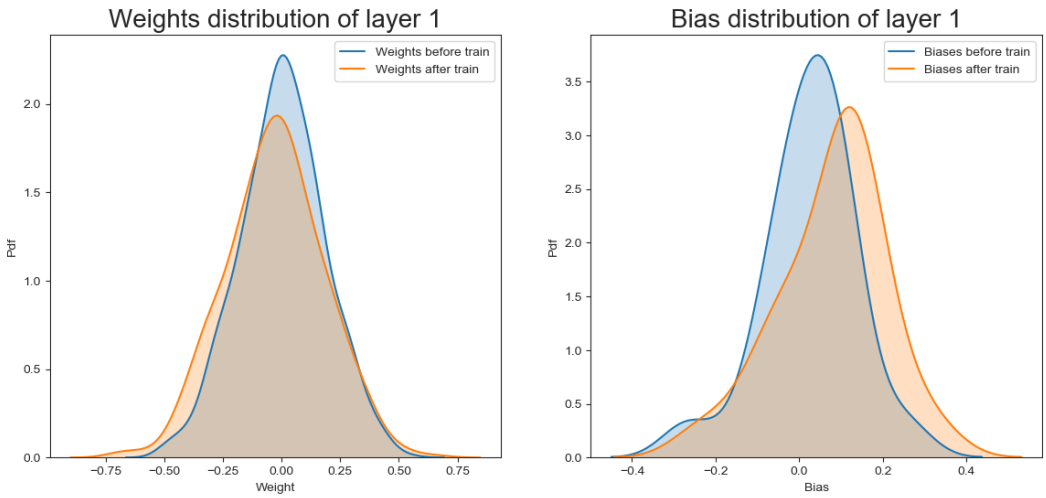


(b) Second layer

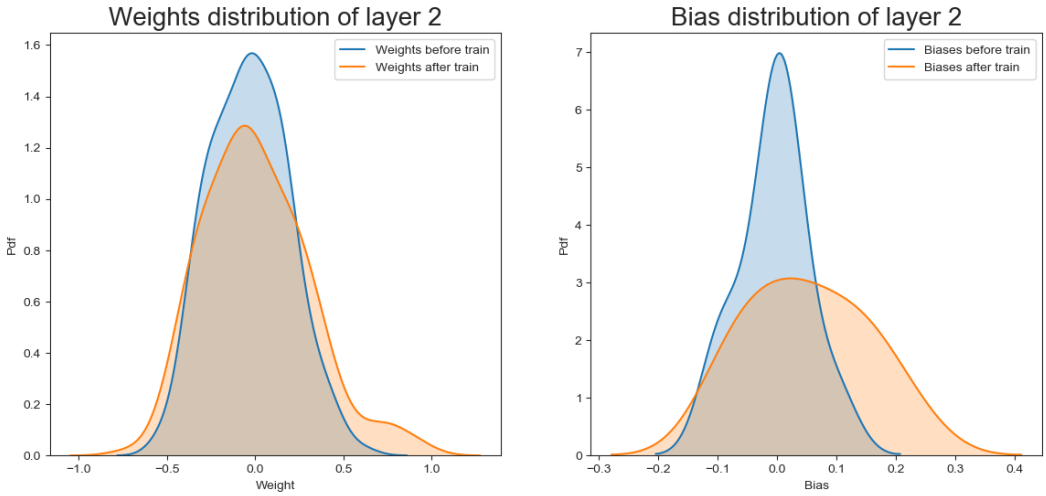


(c) Third layer

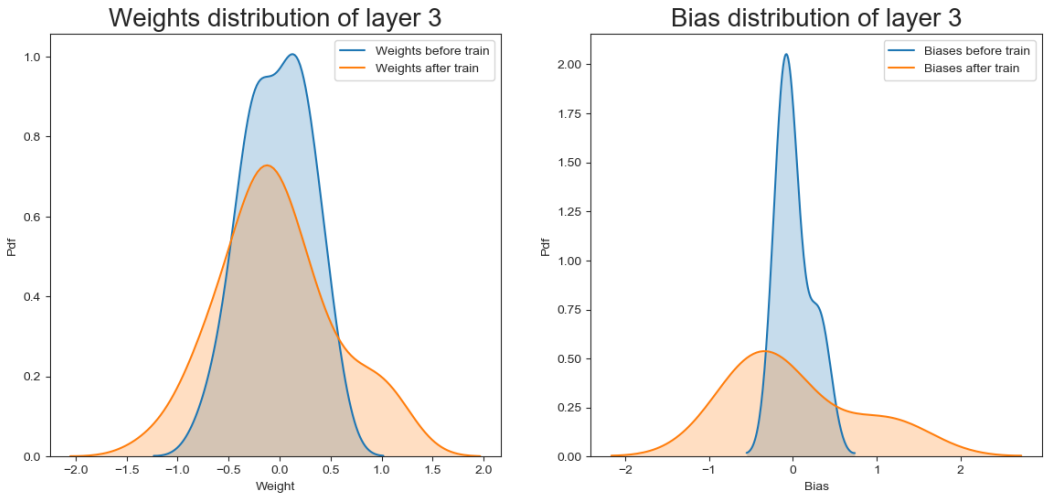
Figure 9: Frequency plots of the parameters before and after training. MVG.



(a) First layer

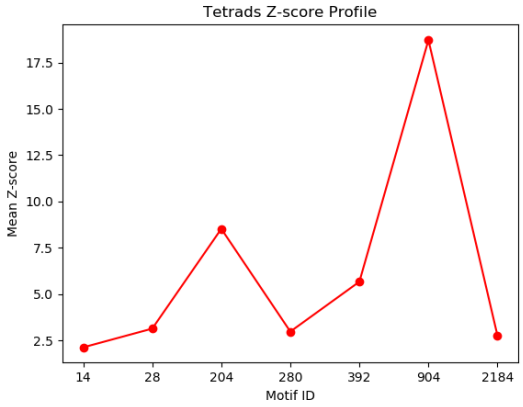


(b) Second layer

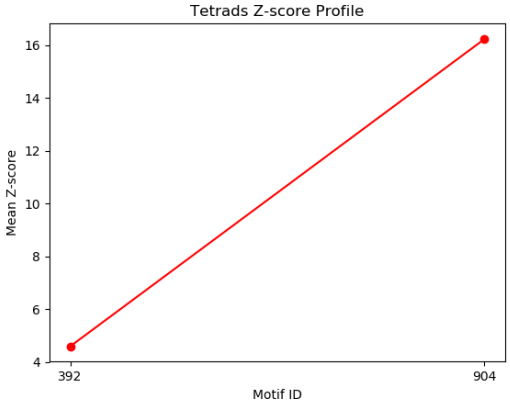


(c) Third layer

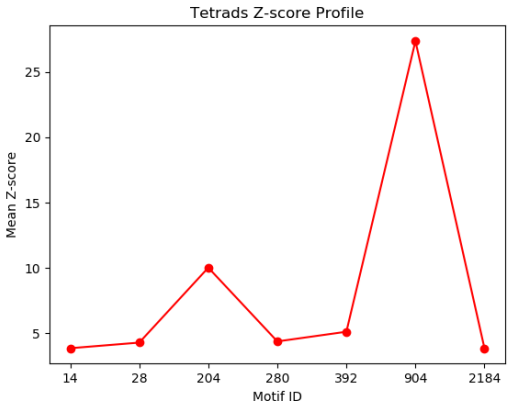
Figure 10: KDE plots of the parameters before and after training. MVG.



(a) Z-scores of the motifs found for the network fed with the tree data set

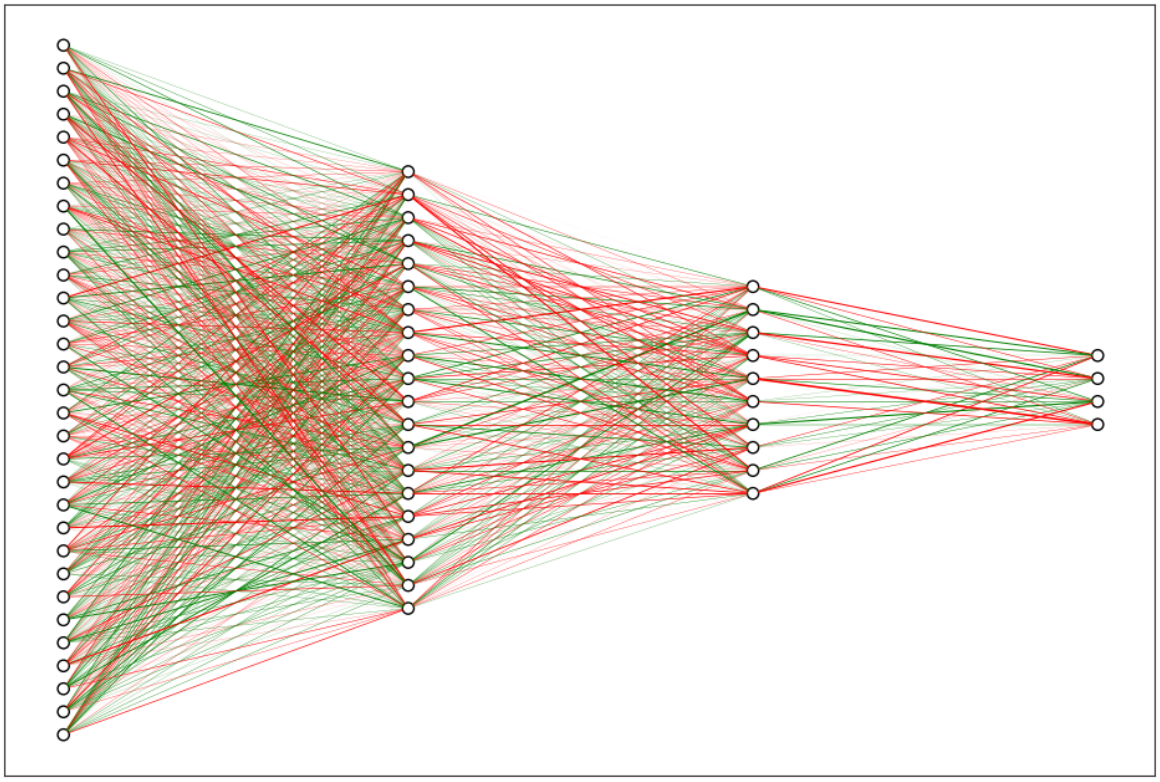


(b) Z-scores of the motifs found for the network fed with the clusters data set

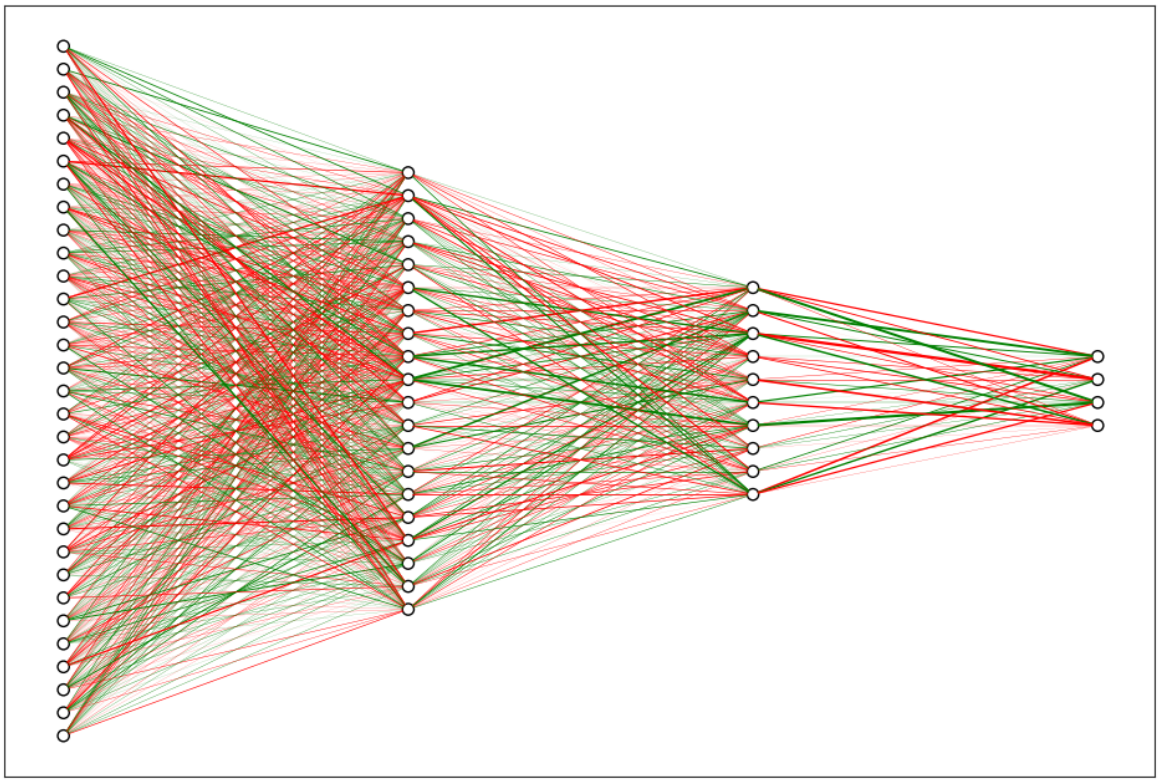


(c) Z-scores of the motifs found for the network undergone the MVG training

Figure 11: Z-scores for the two different data sets and the MVG version of training.

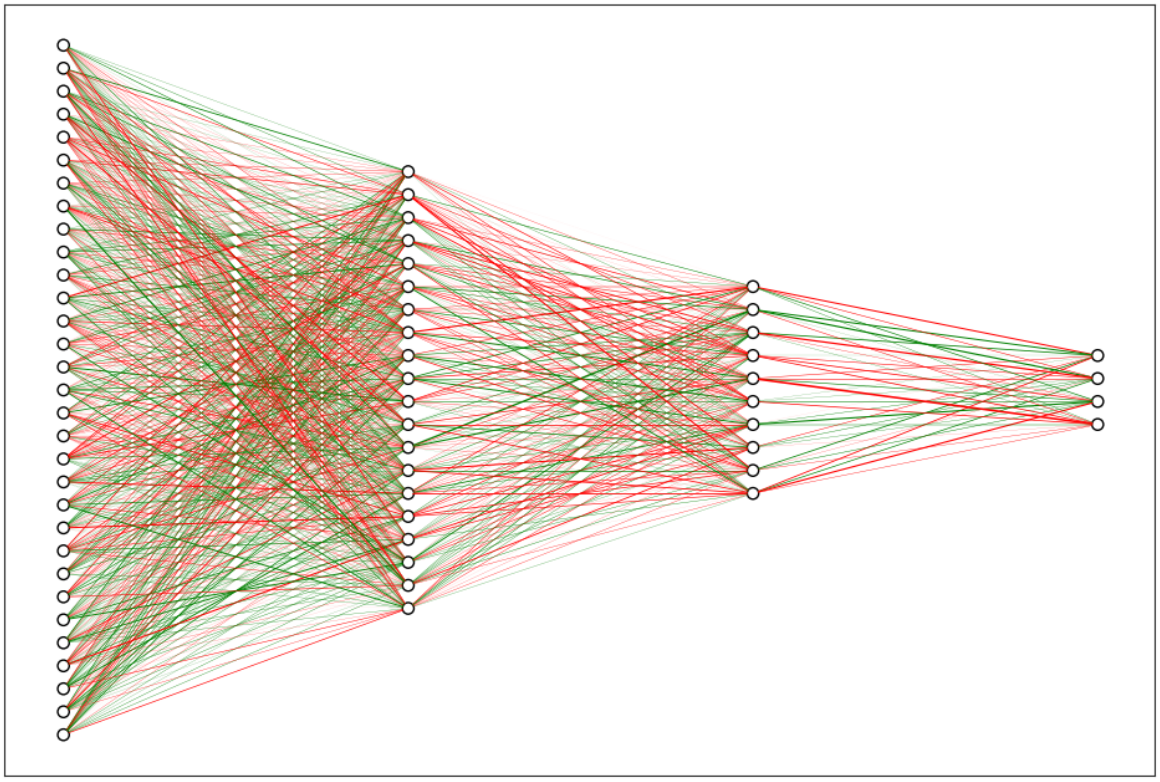


(a) Visualisation of the untrained network topological structure with connection strengths highlighted in red for negative valued weights and green for positive values ones.

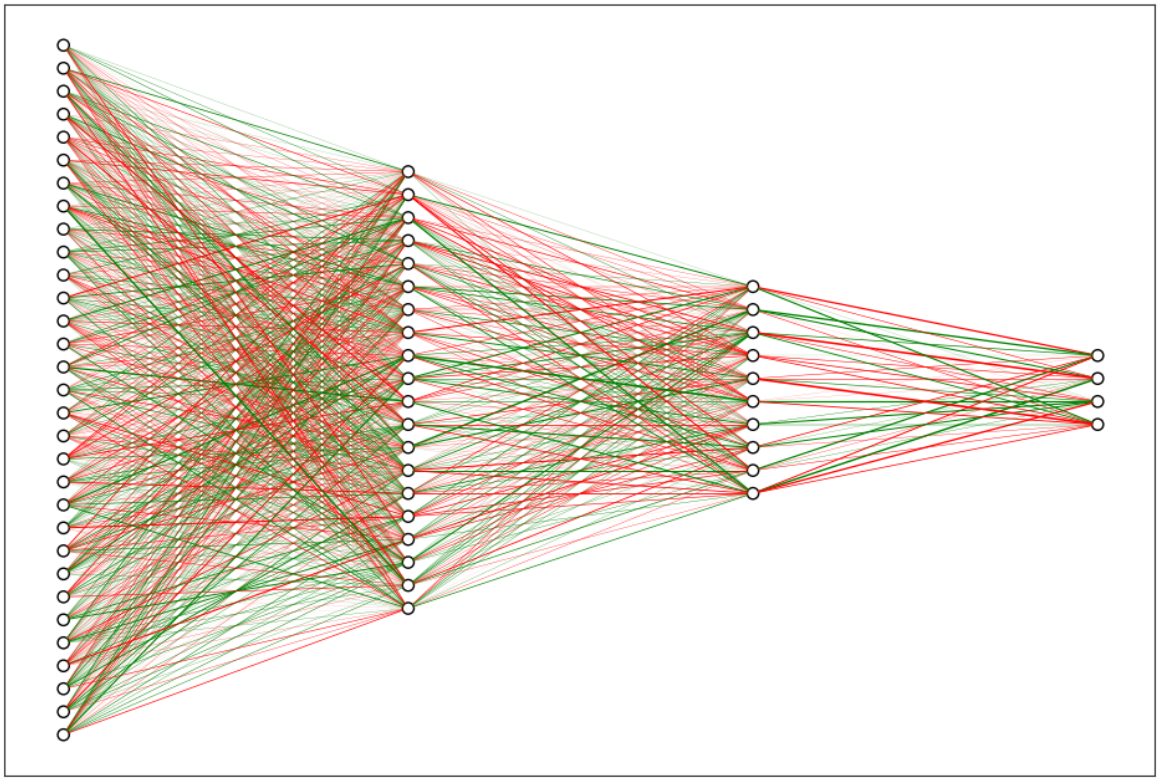


(b) Visualisation of the trained network topological structure with connection strengths highlighted in red for negative valued weights and green for positive values ones.

Figure 12: Visualisation of the network strengths for the binary tree data set.

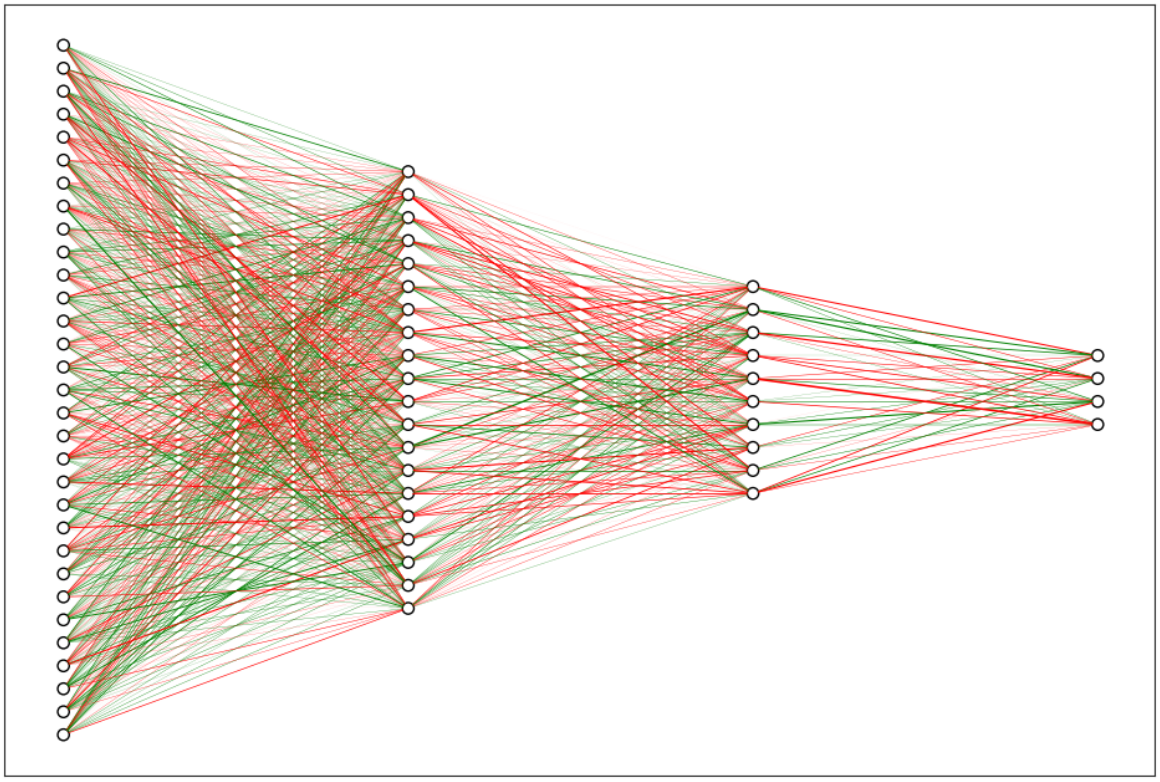


(a) Visualisation of the untrained network topological structure with connection strengths highlighted in red for negative valued weights and green for positive values ones.

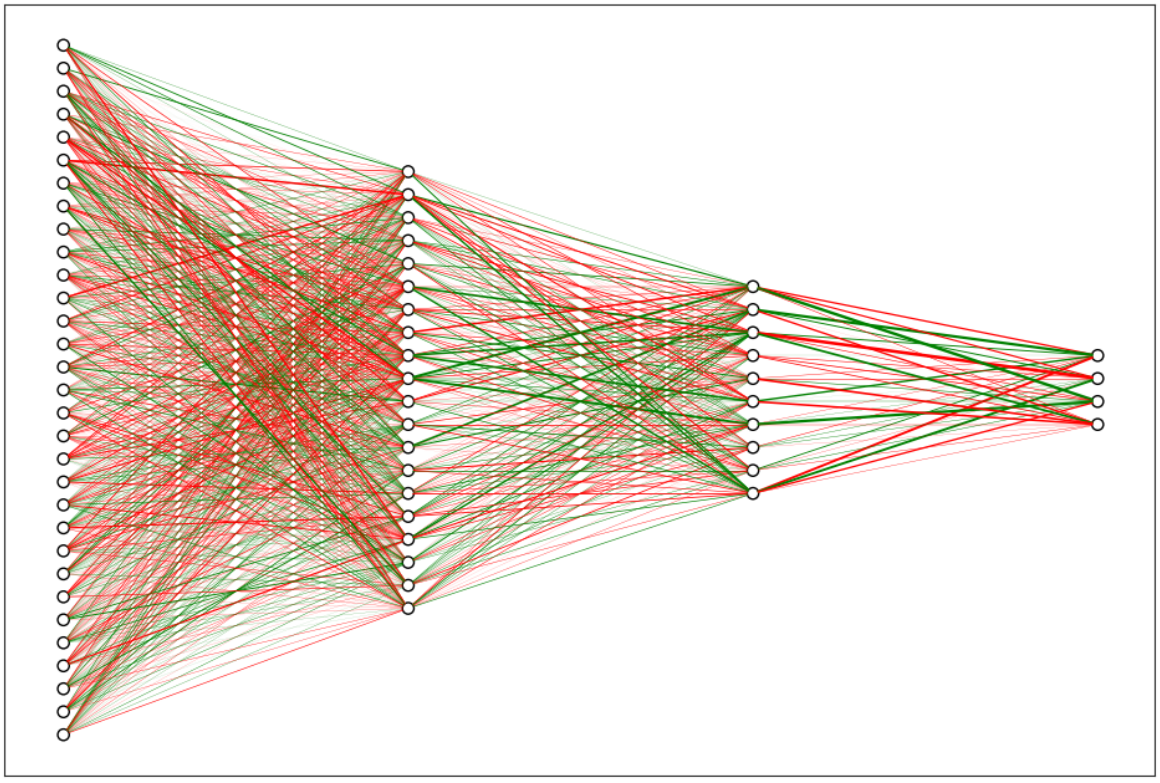


(b) Visualisation of the trained network topological structure with connection strengths highlighted in red for negative valued weights and green for positive values ones.

Figure 13: Visualisation of the network strengths for the clusters data set.



(a) Visualisation of the untrained network topological structure with connection strengths highlighted in red for negative valued weights and green for positive values ones.



(b) Visualisation of the trained network topological structure with connection strengths highlighted in red for negative valued weights and green for positive values ones.

Figure 14: Visualisation of the network strengths for the MVG-training undergone system.