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.

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, 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. MVG is discarded.

As a second betterment: Noising of the data set as formerly worked, has been discarded. At its place the alternative data set used is another binary tree-structured data set, but having chosen a finer level of detail. That is, in the first one the level selected in the binary tree is the level 2, in the second set of data has been selected the level 4. Respectively, these choices result in 4 and 16 categories the system shall classify samples into.

1 Introduction, aim and scope

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

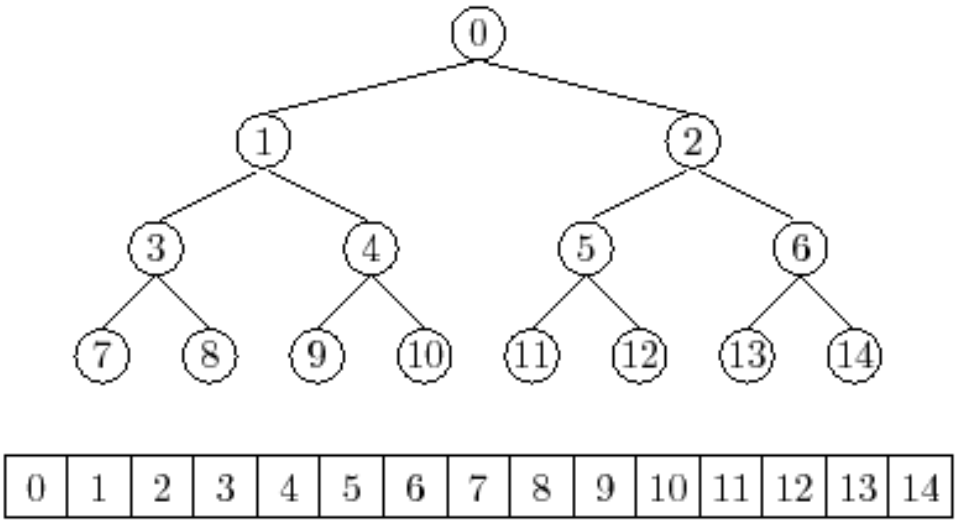


Figure 1: From OpenDataStructure site.

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 set

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

[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*.

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.

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 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 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 $i = 7, \dots 14$ 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 $+1$.

2.2 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.

2.3 Two data sets used

For the sake of this stage of the analysis, the two sets of synthetic data used are essentially the same, what changes is the level of granularity selected, respectively level 2 and level 4. In the former, nodes up to node 2 in Figure 1 are considered, whilst

in the latter the relevant nodes range up to the 14 indexed node. Note that saying *nodes up to* n means that we consider the outcome of the first n random variables of the PGM.

3 Baseline accuracy assessment

Owing to the sharp and clear hierarchical inner structure of the data themselves, the data sets is linearly separable, that is, items can be classified with almost no error. To assess an expectancy range for the accuracy attained by the neural network subsequently presented, it is in order first to test a linear model. Two models are used: a linear kernel Support Vector Machine (SVM) and a Classification Decision Tree. This latter is not, strictly speaking, falling under the class of linear model because of its intrinsic non-linearity, it is rather used to double check the result of the SVM.

3.1 Linear kernel SVM classifier

SVM classifier yields good results. Moreover, the probabilistic threshold beforehand introduced plays also here a crucial role, in that as this is kept at 0.1, few occurrences of some classes are generated and this may give rise to the non-presence, or presence of too few examples, in the training set, subsequently to the TRTE7030 split. This in turn translated in the potential absence of a number of classes in the test set, with consequent non correct classification of such samples. This motivates to rise ϵ to 0.5. In this case, any of the categories is left apart and all of them are included in the training, of course, and in the test stages. If the classes are 4, as in the present case, this peril is all the way avoided. Accuracy ranks to 1.0 for a 0.3 fraction of test samples out of the entire data set.

3.2 Decision Tree classifier

It is not expected that the formal consistency of the data set structure and this classifier is always producing such good results. Accuracy of 1.0 is attained also by this classifier.

4 Neural system setup

In general, a Machine Learning model (neural networks do not except) should embed as much complexity as how much is needed to solve the problem at hand. By complexity is meant the number of hyper-parameters, comprehending: number of layers, number of hidden units (input and output units are held fixed), learning rate, gradient descend momentum, weight decay are the most typical quantities to fiddle with in exploring the space of variability of these. The parameters of the model, those that are properly *learned* by the system, are assessed by running the optimization algorithm chosen, such as Stochastic Gradient Descend (SGD) and subsequently Back-propagation of errors, and are computed automatically. On the other hand, the above listed hyper-parameters are usually manually tuned in order to find the optimal combination for which best performance is attained.

If the case is that of linearly separable data, few parameters are needed (the problem can be transposed in the linear regression setting, or better dealt with Support Vector Machines) and the model should not be built improperly complex. But if the datum

itself shows non-trivial structure, is made of a number of features and bears noise, its complexity forces the system architecture to be complex as well.

The two tasks discussed involve a different number of output neurons, then the architecture is in some extent dynamic. In the table below, what may change (layers indexed with the asterisk) is the terminal layer, among the numerosity of 4 or 16 units.

Other that merely the topological setting, it has been used as optimization algorithm a Nesterov Accelerated SGD (NASGD), with the parameters reported in the table following. Such a setting for the SGD algorithm is kept through the whole set of experiments.

5 Results

The data set previously assessed to be linearly separable is fed to a feed forward neural network (DFFNN) having the following architectural characteristics:

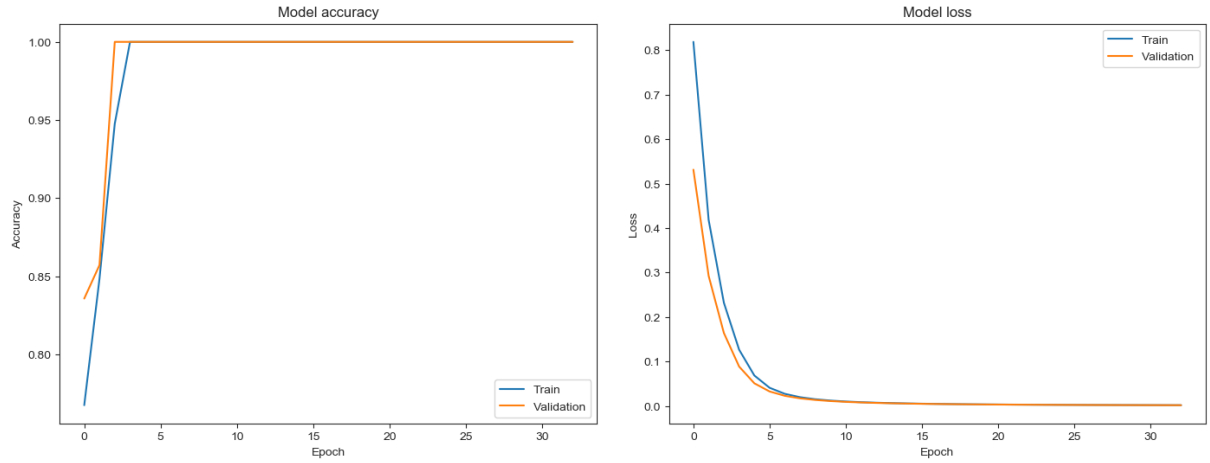
Architecture		
Layer	Units	Activation
1 (input)	$N = 31$	-
2 (hidden)	20	ReLU
3 (hidden)	10	ReLU
4* (output)	4	Softmax
4* (output)	16	Softmax

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

5.1 Initial values of weights and biases

It may happen that the above mentioned algorithm gets stuck in **local minima**, minima in the cost function hyper-surface which do not correspond to the optimal parameters configuration. A good choice of initial parameters values is crucial for the learning process not to experience such a problem. The choice made for biases is Gaussian random values with $\mu = 0.0$ and $\sigma^2 = 0.1$, while for weights it is believed to be a better choice to use the orthogonal initialisation with 1.0 gain. This holds for all of the layers. Orthogonal initialisation is known to render learning time independent of the network depth in a linear regime (Saxe, *et al.*, 2014).

In Figures 6, 7 and 8, 9 histograms and Kernel Density Estimate of these parameters are presented superimposed to the final parameters frequency plots and distributions. It is crucial that the initial parameters are the same for both the level 2 data set and the level 4 data set cases, in that, being the system non-linear *per se*, different initial conditions would imply different dynamics. So by keeping fixed the initial status, the final parameters distributions for both level 2 and 4 cases refer to solutions that evolve



(a) Accuracy **(b)** Loss

Figure 2: Accuracy and Loss profiles, level 2 data set.

from the same initial configuration. In Figures 10a, 11a the **connections strengths** of the initial states are reproduced, in conjunction with the network topological structure.

A **design care** adopted is the following: in the parameters initialisation stage, the first two connections layers were initialised but the numerosity of the last layer is not known yet, it depends on the data set chosen. In order to fix the initial weights of the first two layers, the model is saved and then fetched once the number of neurons of the output layer is known. In this way it is avoided to generate new initial values so the initial status is as similar as possible for both the cases.

5.2 System performance and final configuration

In both the following settings, the training set accuracy ranked to 1.0. In the first case such value is attained in two or three epochs, in the second it takes a while more to attain the maximum accuracy value.

5.2.1 *Level 2 data set*

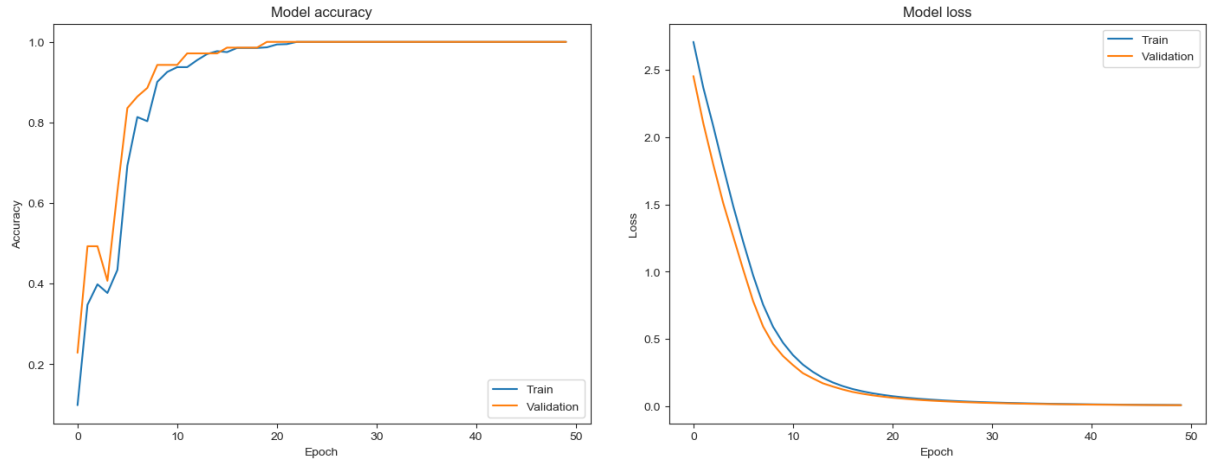
The simple neural network illustrated is trained with the data set. As said, a 0.7 fraction is used for training, a 0.1 of which is used as validation set, and the 0.3 kept apart before is used as test set. Being the data set linearly separable, in few epochs the accuracy metric attains the top value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

Test set accuracy	1.0
Test set loss	0.0015

As final result, weights and biases distributions are also reported for the trained network, alongside with the connections strengths visualisation, Figure 10b.

5.2.2 *Level 4 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 slightly more complex, training takes some epochs more. Again



(a) Accuracy **(b)** Loss

Figure 3: Accuracy and Loss profiles, noisy data set.

visualisation of the connections strengths in Figure 11b.

Test set accuracy	1.0
Test set loss	0.01

5.3 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. It is thought wise to report results concerning: initial weights configuration, weights once the system trained with the level 2 data set configuration, weights once the system is trained with the level 4 set configuration. Here the results refer to a cut-off threshold of $C = 0.35$. Observing the distributions in Figures 8, 9 it seem a value that does not imply the loss of much information.

The following tables report the results of the software execution. It is clear that in the second case the motifs found are more numerous and there are two pattern emerging that did not show up in the case of level 2.

6 Conclusions

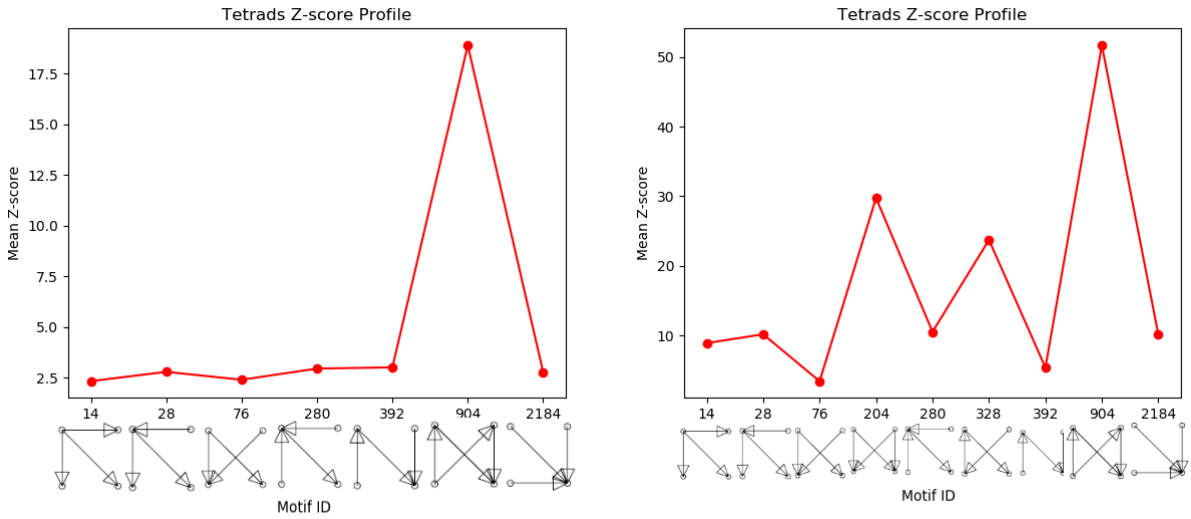
The baseline of accuracy is established by classification with simpler models. Also in these cases the performance is satisfactory. However, what matters is the following: The structure of the data set is simple and transparent, easily assessable in terms of mathematics and statistics. The final outcome of the learning process is the accuracy on unseen examples, but nothing is said about the behaviour and the evolution of

Nodes	Motif ID	N_{real}	N_{rand} stats	N_{real} Z-score	N_{real} P-val	Unique Val	C real
4	14	25	18.1 ± 2.9	2.34	0.010	5	12.91
4	28	150	118.1 ± 11.4	2.80	0.000	4	77.48
4	76	369	318.8 ± 20.9	2.41	0.010	7	190.60
4	280	291	242.0 ± 16.5	2.96	0.000	8	150.31
4	392	504	411.3 ± 30.7	3.02	0.000	7	260.33
4	904	48	5.2 ± 2.3	18.88	0.000	5	24.79
4	2184	125	108.3 ± 6.0	2.78	0.000	8	64.57

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	14	949	555.0 ± 44.0	8.94	0.000	10	41.87
4	28	2872	1784.0 ± 106.6	10.21	0.000	5	126.72
4	76	3777	3354.5 ± 121.4	3.48	0.000	12	166.65
4	204	479	89.7 ± 13.1	29.76	0.000	9	21.13
4	280	3585	2313.1 ± 120.5	10.56	0.000	10	158.18
4	328	3613	1745.3 ± 78.6	23.75	0.000	10	159.42
4	392	3488	2939.0 ± 101.2	5.43	0.000	8	153.90
4	904	726	86.3 ± 12.4	51.65	0.000	6	32.03
4	2184	1521	1017.0 ± 49.5	10.18	0.000	14	67.11

Table 2: Results of the `mfinder1.2` program for the level 4 data set.



(a) Accuracy

(b) Loss

Figure 4: Accuracy and Loss profiles, noisy data set.

the neural network in the stage of learning. By looking at Figures 10b and 11b one could not say whether some relevant topological and structural *motif* came to form. Previous studies shed light on this phenomenon, namely the recurrence of well defined patterns (network motifs), in some network systems, ubiquitous is biology, engineering

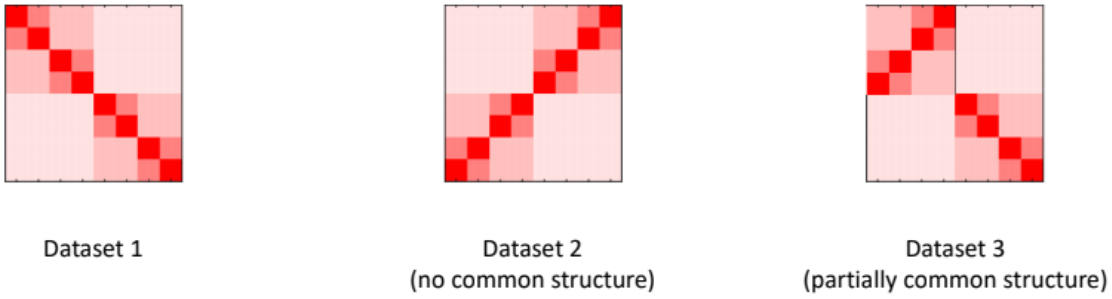


Figure 5: Covariance matrices of some statistically different data sets. The first corresponds to the binary tree case.

and social sciences, refer to Kashtan and Alon, 2005.

What is immediately observable is an increase in weights magnitude, both positive and negative, but it is only visual inspection. It is clear enough however that this weights increase affects in large part the **terminal layer**, that one in charge to activate the final layer neuron identifying a class belongingness. It can be seen both in the neural architectures sketches in Figures 10b, 11b and in the fatter distributions tails in Figures 8b, 9c, left panels. Restricting uniquely to connections weights, few changes for the edges of the inner layers seem to occur. This in part may suggest that the complexity of this system is over-abundant.

6.1 Differences between the first version of this report

Both in the previous version (clean and noised data set) and in the present (different levels of detail), results are not striking. It it obvious that weights come to increase in magnitude as a consequence of the learning process. In Figures 4a and 4b the occurrence of motifs, as counting of occurrences of patterns in the evolved systems with respect to random networks with the same degree sequences and structure, is depicted. In the case of level 4 data set the numerosity of some of these motifs is higher but it could be trivially due to the greater number of links between neurons, owing to the extended output layer. Moreover, the statistical structure of the two data sets is essentially equal. In the level 2 case the covariance matrix is made of diagonal blocks and in case of level 4 the structure is the same, but the diagonal block contain further sub-blocks as the first column in Figure 5, also in diagonal fashion. Thus it is not expected that in the two cases the influence of the covariance impacts significantly.

6.2 Further improvements

6.2.1 *A different data set*

An alternative data set is in order. In particular, a set of synthetic data the covariance of which exhibits a different shape, for example one of the block in the diagonal should be anti-diagonal as for example the second column of Figure 5.

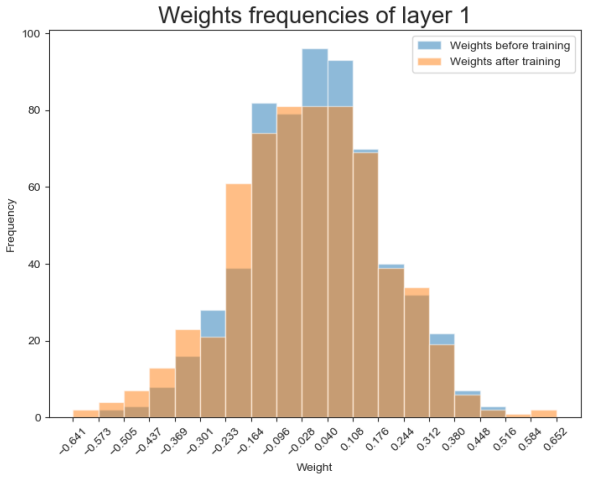
A data set coming from an independents clusters PGM could resolve in some different outcomes. One possible way to generate such an ensemble could be as follows

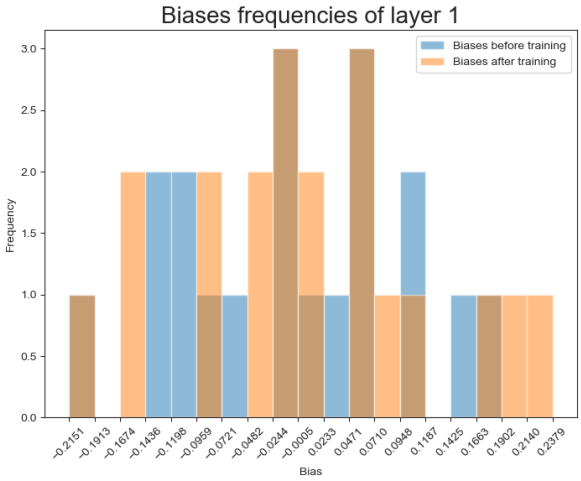
(**TL;DR**: See Algorithm 1, Inspired by the works of Newman, *et al.*, 2003, and Newmann, 2003): Generating some cloud of points distributed according to a bivariate Gaussian distribution, with means spread apart and covariances sufficiently small, in such a way that the points of the different groups 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, turn them 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.

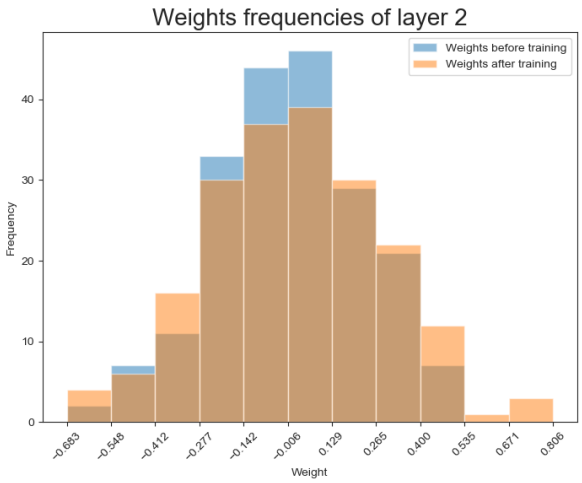
Algorithm 1 Simulated melting to generate independent clusters

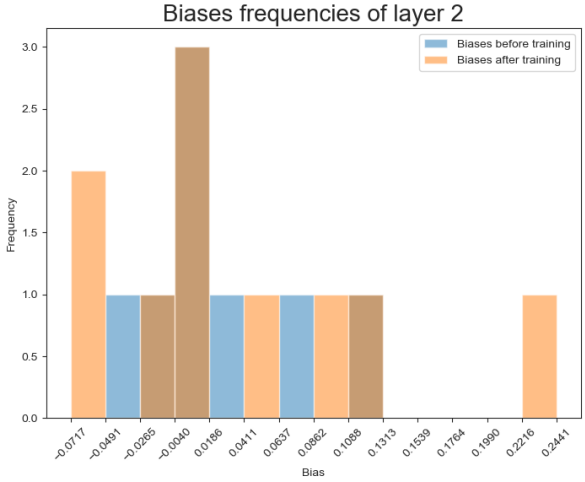
Choose the number of classes N_C
Set $\boldsymbol{\mu}^{(k)} \in \mathbb{R}^2$, $\boldsymbol{\Sigma}^{(k)} \in \mathbb{R}^{2\times 2}$, $k = 1, \ldots, N_C$
Generate $\boldsymbol{X}$ s.t. $\boldsymbol{x}_i \sim \mathcal{N}(\boldsymbol{\mu}^{(k)}, \boldsymbol{\Sigma}^{(k)})$, $i = 1, \ldots, M$
Include the indexes of the points generate in a list, which is the set of the vertices $\mathcal{V}$ of the graph $\mathcal{G}$
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, $(x_i^{(1)}, x_i^{(2)})$).
for T increasing **do**
    for All the edges $e = 1, \ldots, |\mathcal{E}|$ **do**
        if Length of edge $e > \frac{1}{T}$ (for example) **then**
            Remove edge e
        end if
    end for
end for
Plot the remaining edges and check if only independent fully connected components have survived.



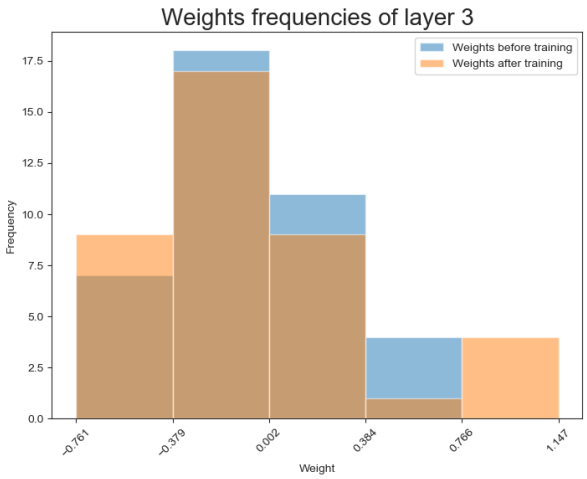


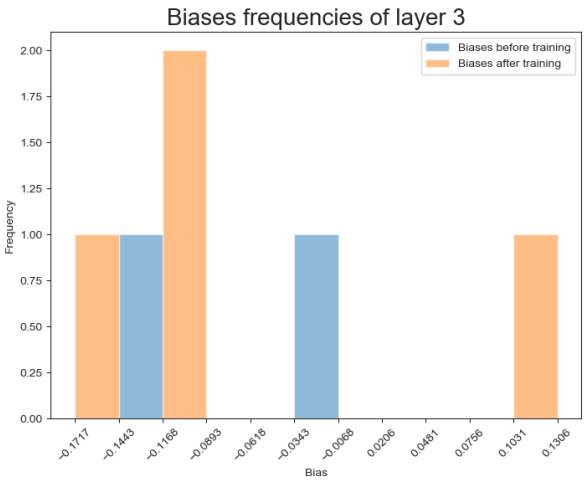
(a) First layer





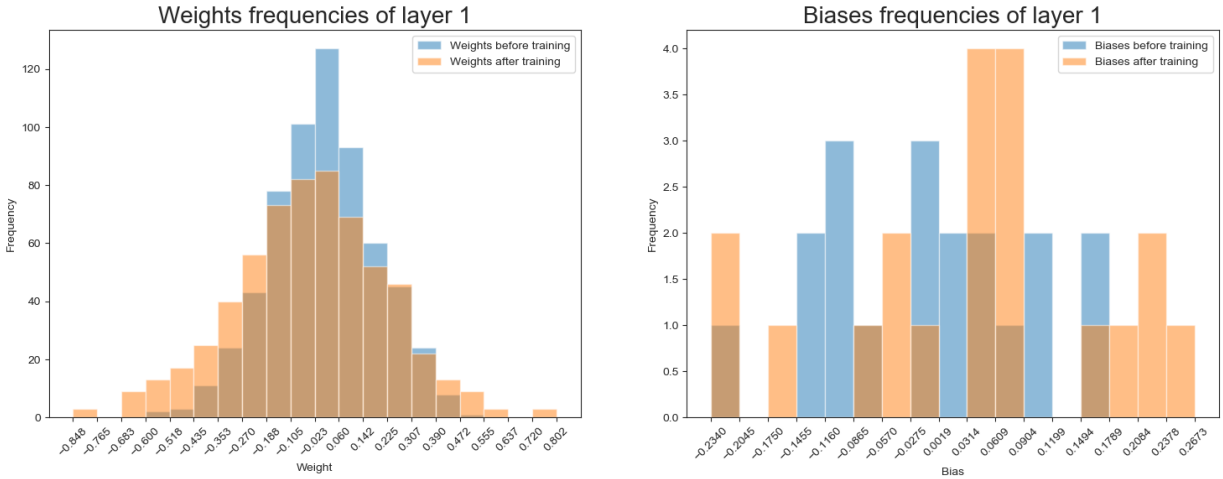
(b) Second layer



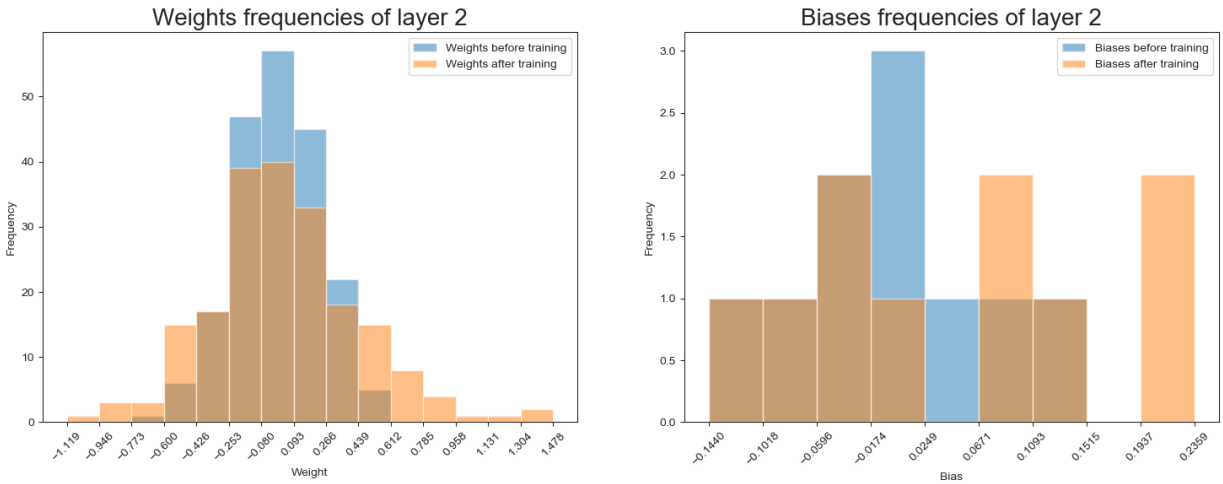


(c) Third layer

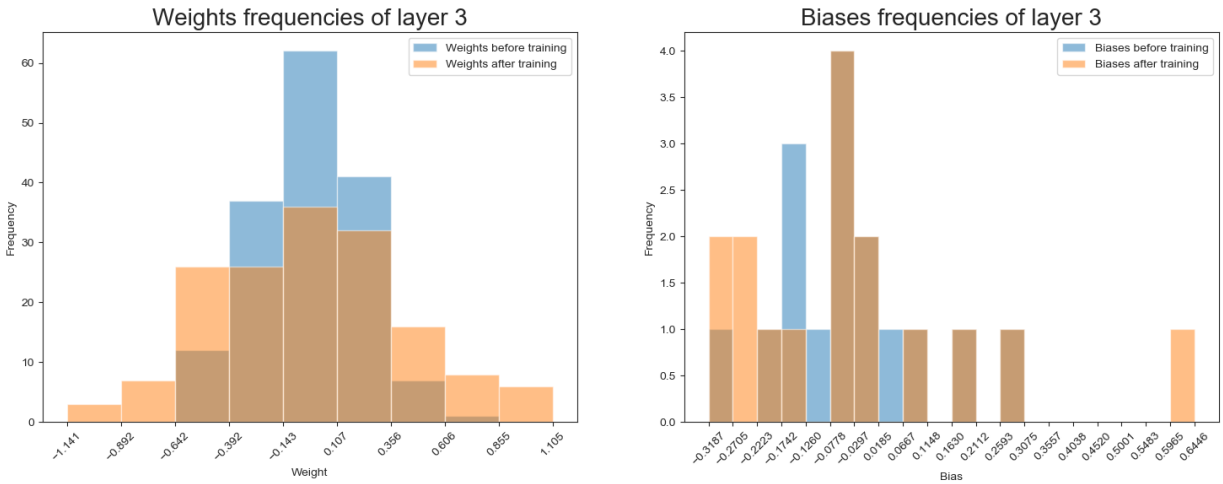
Figure 6: Frequency plots of the parameters before and after training. Level 2 data set.



(a) First layer

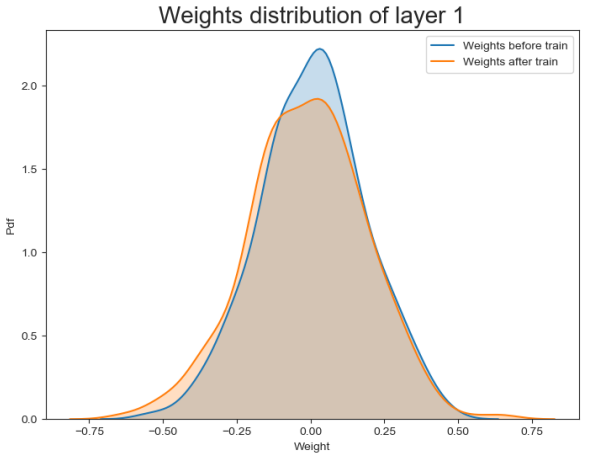


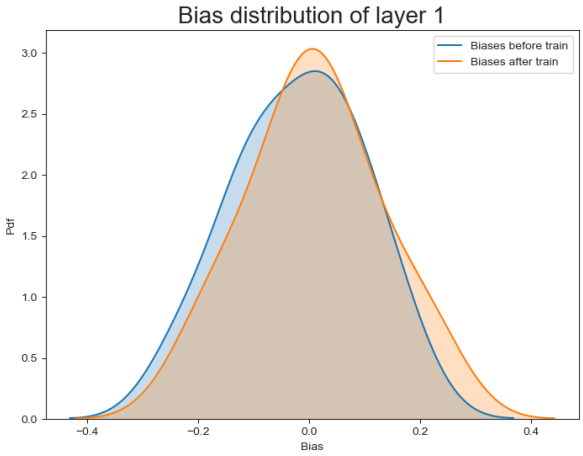
(b) Second layer



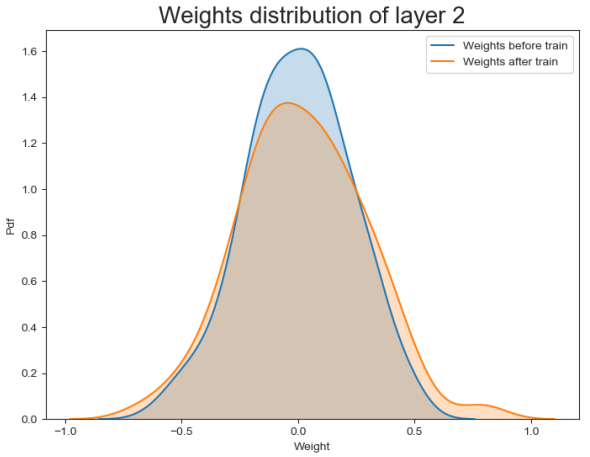
(c) Third layer

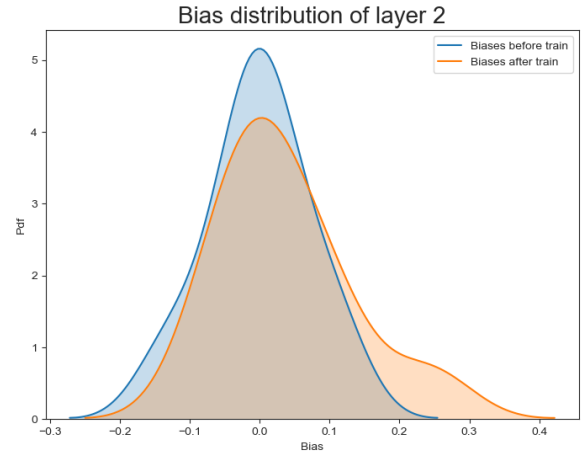
Figure 7: Frequency plots of the parameters before and after training. Level 4 data set.



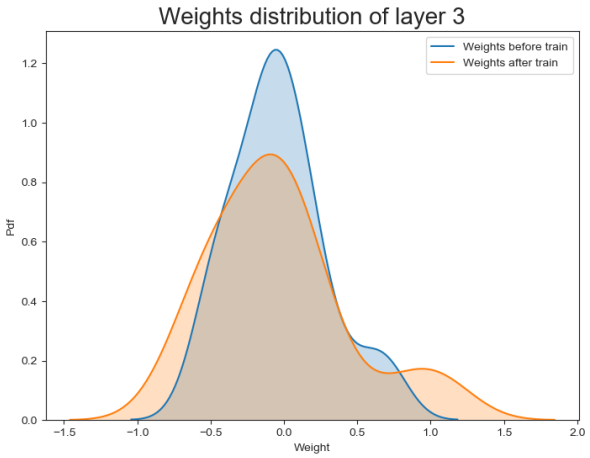


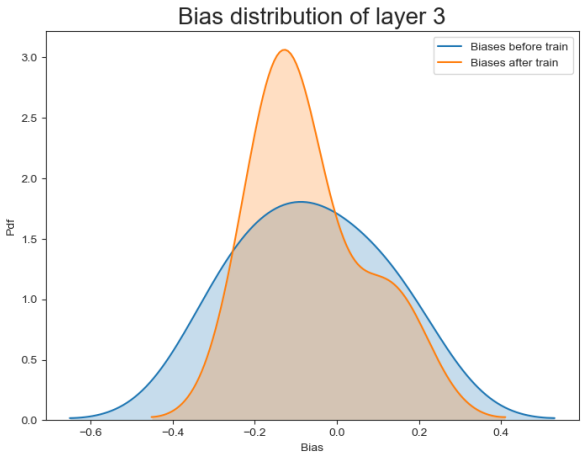
(a) First layer





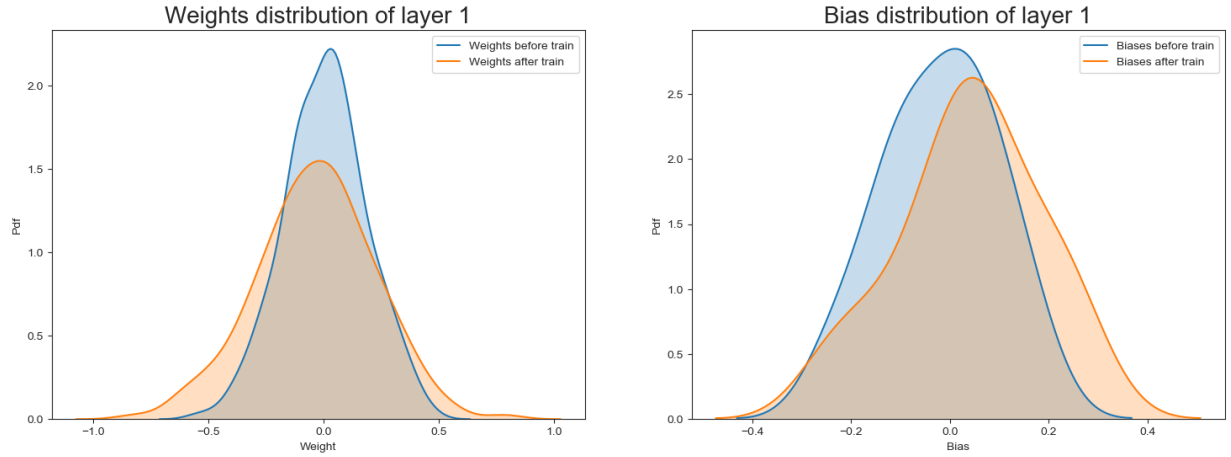
(b) Second layer



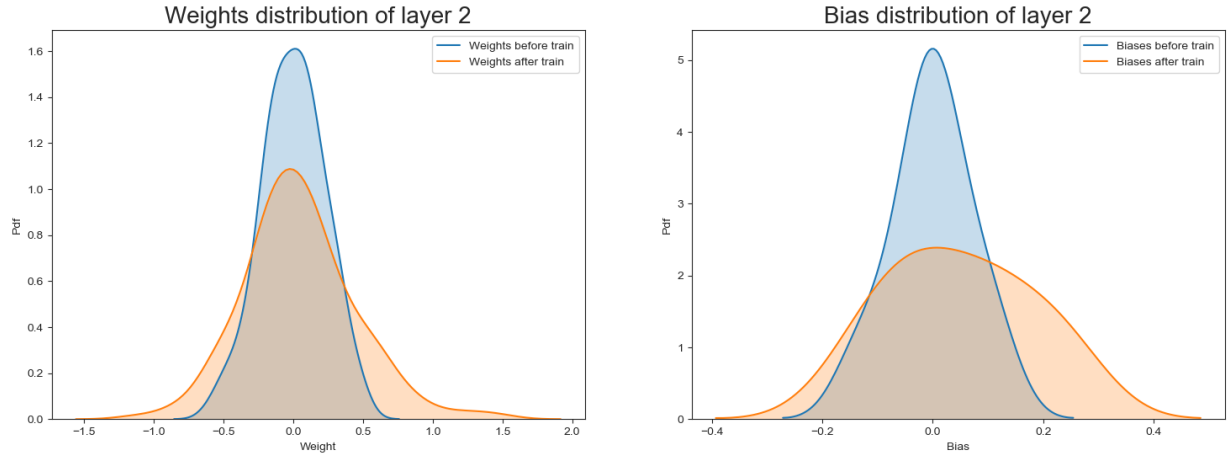


(c) Third layer

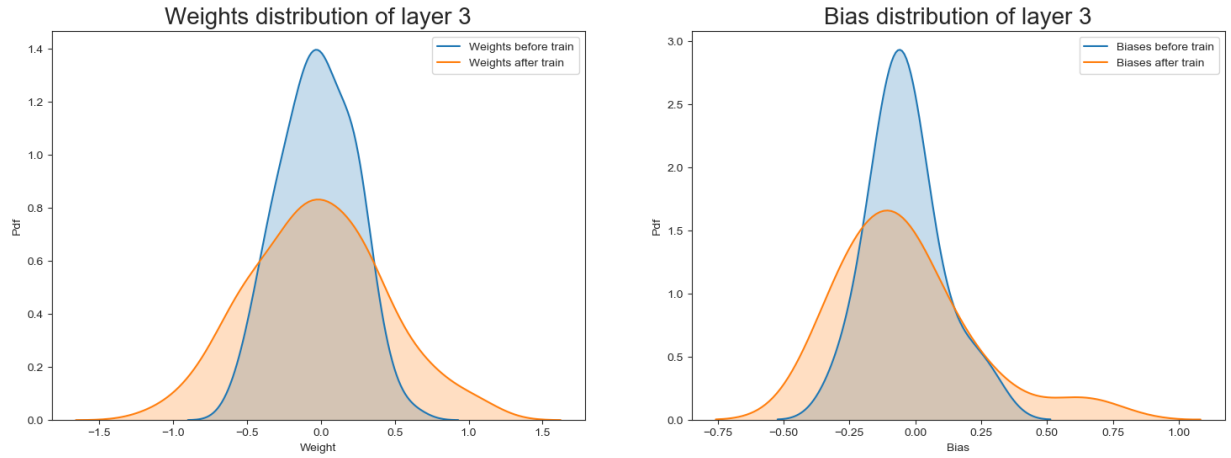
Figure 8: KDE of the parameters before and after training. Level 2 data set.



(a) First layer

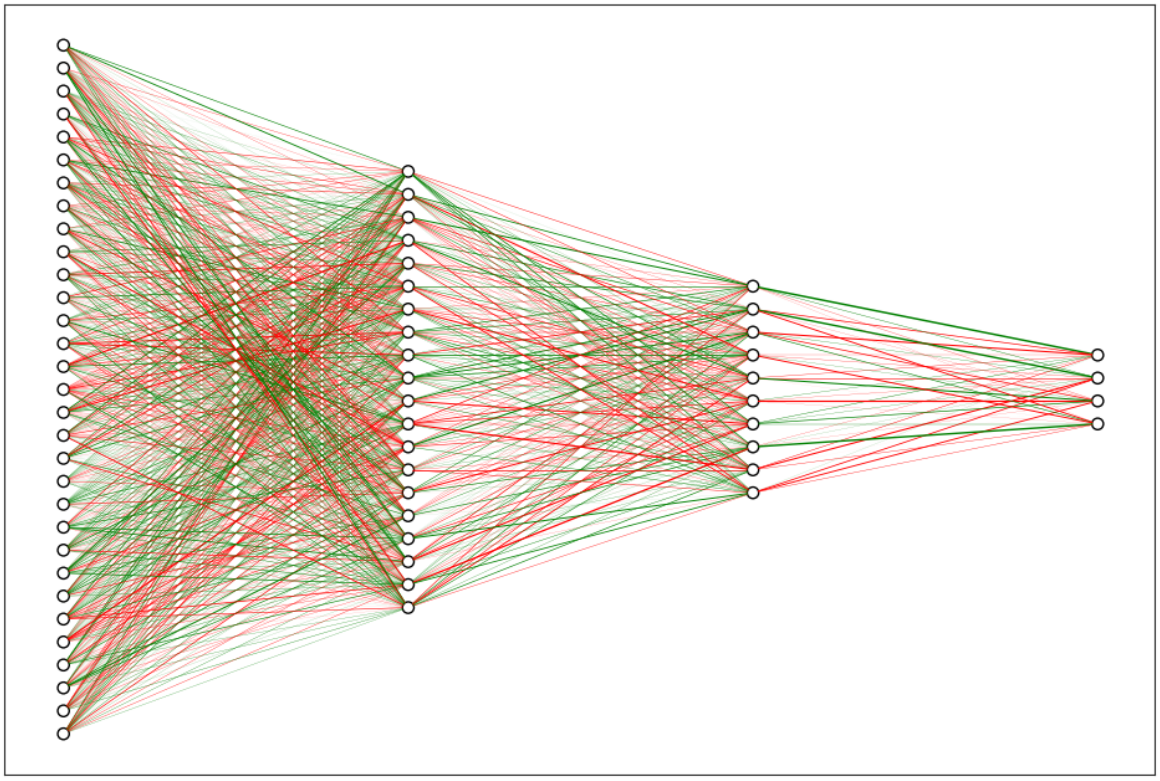


(b) Second layer

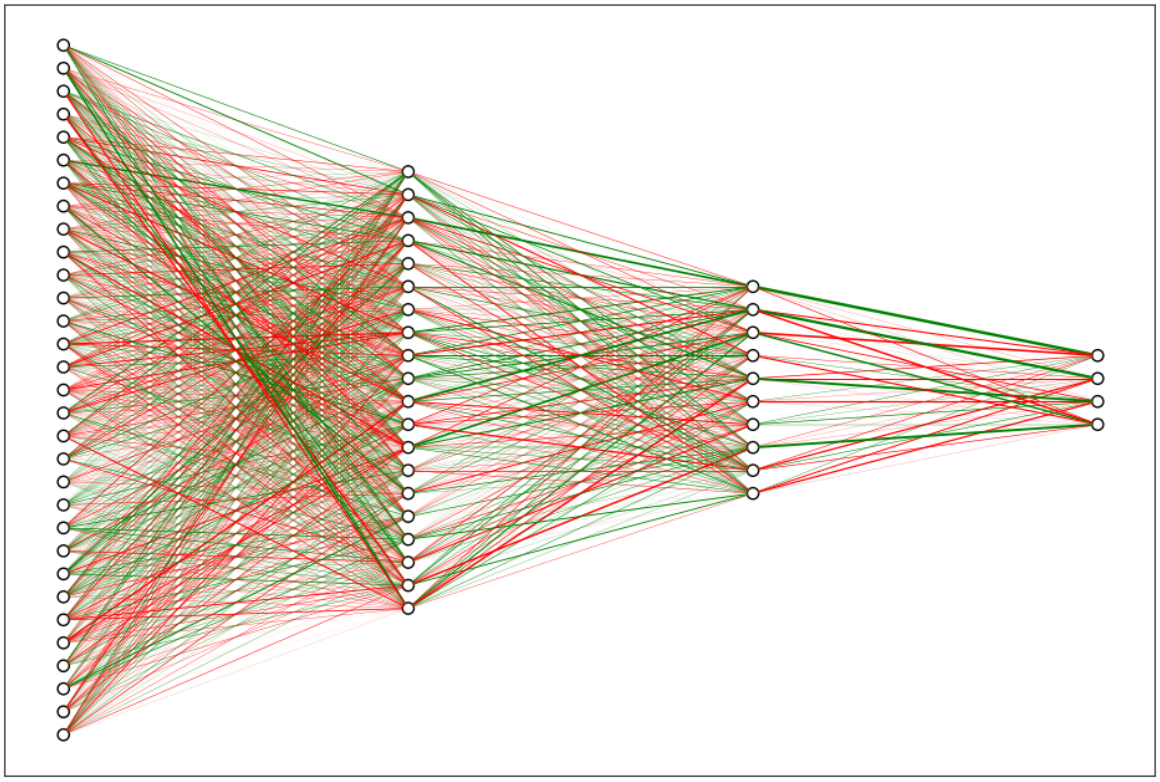


(c) Third layer

Figure 9: KDE of the parameters before and after training. Level data set.

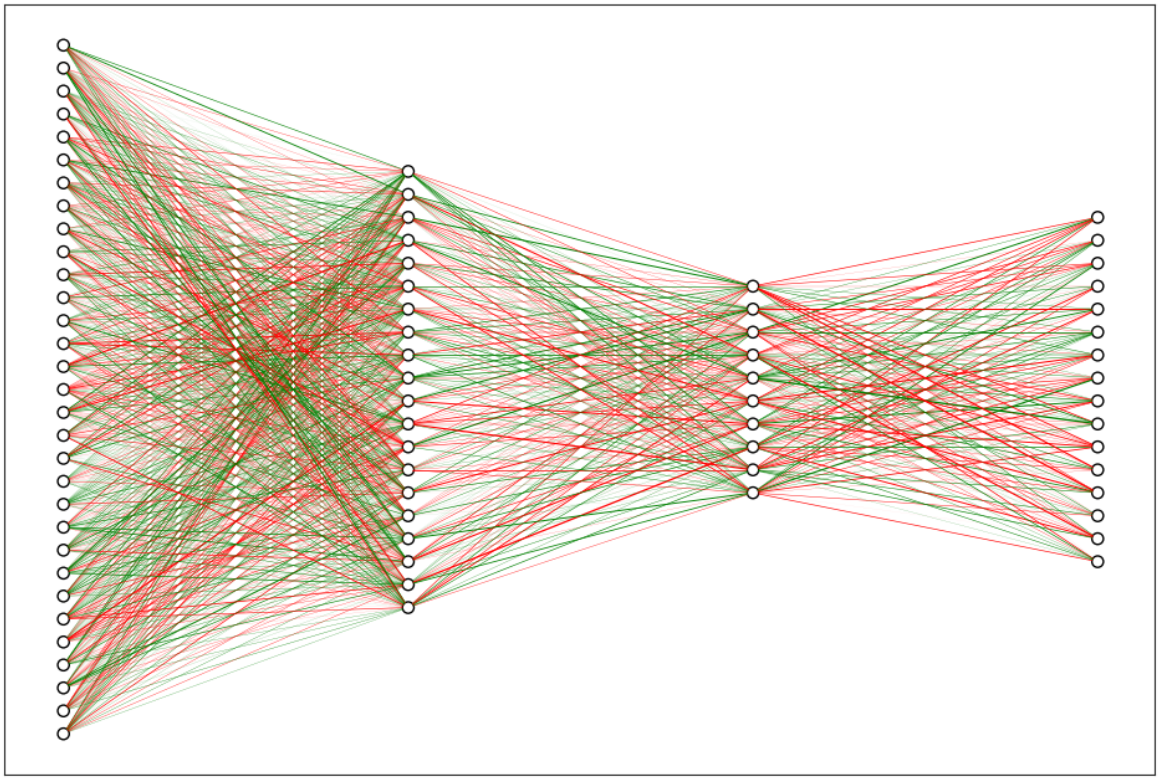


(a) Visualisation of the untrained network topological structure with connection strengths highlighted in red for negative valued weights and green for positive values ones.

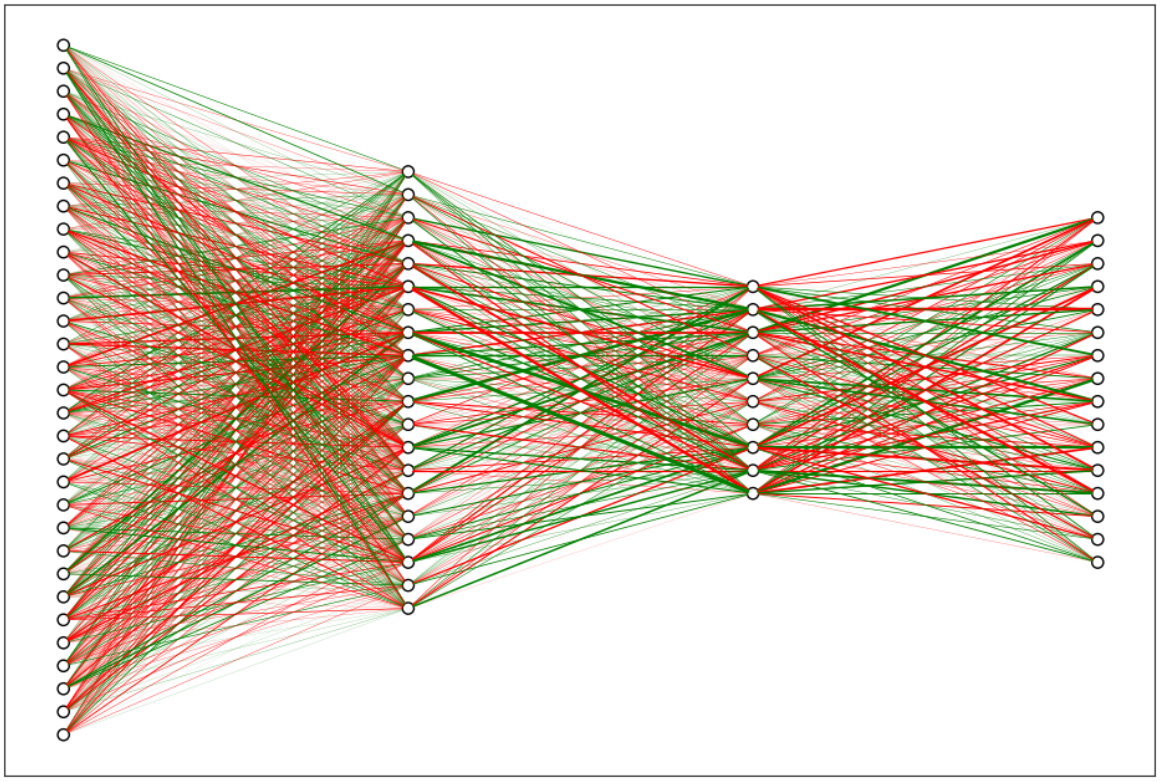


(b) Visualisation of the trained network topological structure with connection strengths highlighted in red for negative valued weights and green for positive values ones. This plot refers to the clean data set.

Figure 10: Visualisation of the network strengths.



(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. This plot refers to the clean data set.

Figure 11: Visualisation of the network strengths.