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

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

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.

1 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;

The consistence with the above mentioned study is recovered in the final stage: the foremost aim of this present work is to assess whether a neural network exhibits some peculiar topological characteristics, once exposed cyclically and periodically to mutating environment. This translates in training the system on different tasks, repeatedly, in a modular fashion. Otherwise, if the task exposure does not follow a scheme and the environments are randomly changed, no significant motif and/or modular structure can be identified (cfr. Kashtan and Alon, 2005).

As a rough example: If the system is intended to be trained to classify image of cats, in the first stage it shall be exposed to a dataset containing cats pictures. It eventually attains the best performance. This translates in a certain internal representation of the data, and hence in a certain weights and biases configuration. Then the task changes: The system is required to classify elephants. The training stage is reiterated, until the requested subject is correctly recognised by the network classifier.

If these two goals are repeatedly switched, the system is expected to develop an *adaptable* behaviour: It would finally not take much time in terms of epochs to perform such a goal switch and see the system to attain steadily this new task. Such a behaviour is expected to translate in terms of network topology in the affirmation of certain pattern that **encode the information common to both the tasks**. It this cats and elephants-recognition fashion: both cats and elephants walk on four legs. Further resemblances are, if any, rather weak. Then one could find reasonable that the network system in the end shall present an internal structure showing a pattern that is associated with this common feature of *having four legs*. It could then be interesting to investigate the relationship between the **overlap** of two different tasks and the subsequent topological encoding of such common traits.

As a zero model, the data set described below is used. That choice is made for classification ease and because in this way the overlap between this data set and another (similar) one is easily controllable. The very purpose of this work is to be in the position to put in clear relationship such an overlap and its signature in the topology of a neural network model trained on such data. Classification ease mentioned is intended to be a sanity check: Once one is sure that the model is correctly trained, then can proceed further and inspect the topology. Otherwise investigation could happen to be carried out on a system that fundamentally is not reliable and able to accomplish its task.

2 Data set

An ensemble of $M = 2000$ data vectors is generated. Each one of these data vectors consists in $N = 2^D - 1$ entries, D is the depth of the tree, $D = 4$ in Fig. 1. These entries can attain one value among $\{-1, +1\}$ and then belongs to a 2^N-dimensional space, where N is the total number of nodes of a binary tree, indexed by counting from 0 to $N - 1$: $i = 0, \ldots, N - 1$.

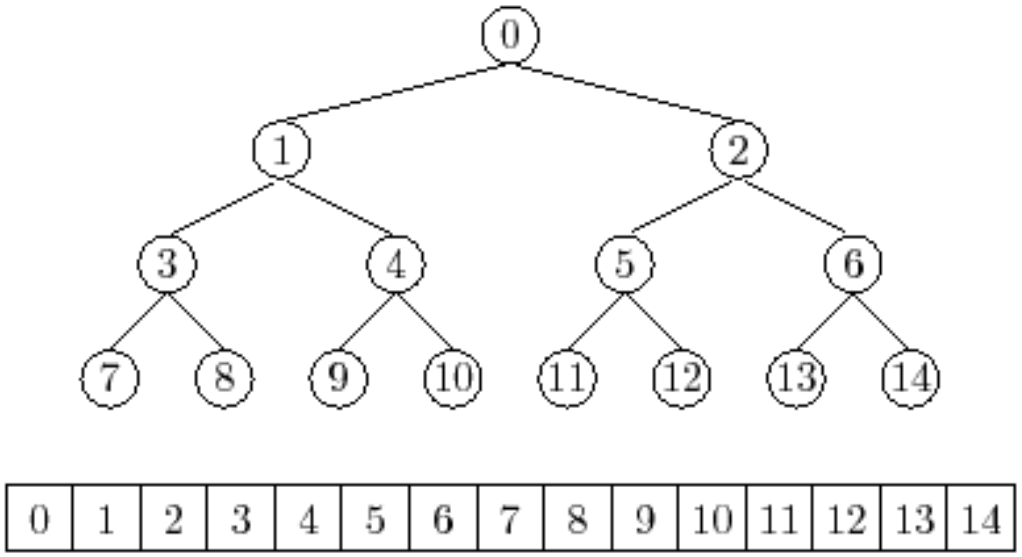


Figure 1: From OpenDataStructure site.

In fact, each one of these data points (2^N-dimensional vectors) is generated according to a **binary tree** fixed structure, refer to Figure 1. All of the nodes in such tree are the **features** of one single **pattern**[1]. 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

[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*.

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 Noising the data set TO BE CLARIFIED AND IMPROVED

It may in some cases be that there is *overlap* between some data items, that is, the problem is not linearly separable any further. This obviously implies a different learning dynamics and performance. In order to do this, an arbitrary number of entries of data items are flipped them values, $x_i^{\mu} \leftarrow -x_i^{\mu}$, for some i, μ. This flip is performed for a fraction of $\frac{N\,M}{5}$ of the total entries of $\boldsymbol{X}$.

Here the main concern is the following: whilst being the data set separable and being possible to select the detail granularity, it can not be asserted that two data sets embedding different details level constitute two potentially overlapping "environments". As a first instance: different granularity means different number of overall classes and, in turn, **more or less neurons in the terminal neural network layer**, and this renders a further problem of **system architecture**, which, for the sake of motif detection, would be preferred to be, for now, fixed. Other than that, although selecting two different detail level, the **statistical**

Algorithm 1 Single feature generation

Compute $N = N_{\text{leaves}}$, $n = N_{\text{not leaves}}$. M is a free parameter.
`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, $\mu = 1, \ldots, M$

Algorithm 2 *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

internal morphology of the datum is not changing, therefore the two different data set generated in this spirit **are perfectly overlapping**. In this light: another data set should be generated, with a different statistical character.

3 Baseline accuracy assessment

As long as the data set is not noised, it is **linearly separable**: The data items can be classified with almost no error, owing to the sharp and clear hierarchical inner structure of the data themselves. 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 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.

3.1.1 *Clean data set*

Accuracy value of 1.0 is attained. Here, in contrast with the neural system below presented, accuracy is assessed solely on the test examples. No misclassification error are made and all the categories are learned by the model.

3.1.2 *Noised data set*

A linear kernel SVM attains 0.90333 of accuracy, while a Gaussian kernel SVM gains some accuracy more, namely 0.935.

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.

3.2.1 *Clean data set*

Also this model attains the best in accuracy, 1.0.

3.2.2 *Noised data set*

Things changes for non linear separable data. Here the best that the Decision Tree can do is 0.81667 in terms of accuracy on test samples.

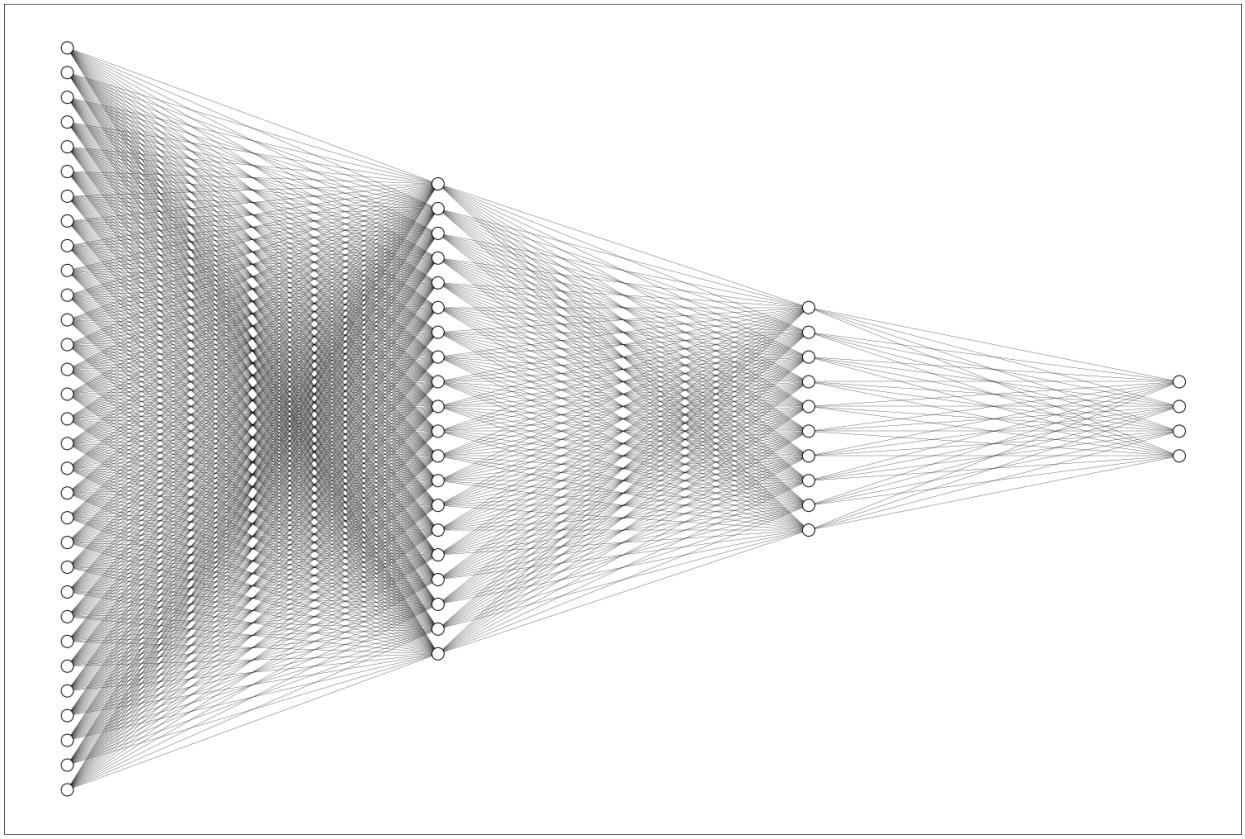


Figure 2: Visualisation of the network topological structure.

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.

In this specific case, being assessed the linearly separability character of the data set, it could, in principle, suffice a neural system with no hidden layer and linear activation function, thus returning to the linear regression setting. However, since the very purpose of the present is to investigate the structure of the neural system as a **complex network**, it is thought appropriate to start with a multilayer model. Owing to the presence of 31 features and 4 classes, these quantities define the sizes of the input and output layers respectively. A sketch of the network architecture is given in Figure 2.

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

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.

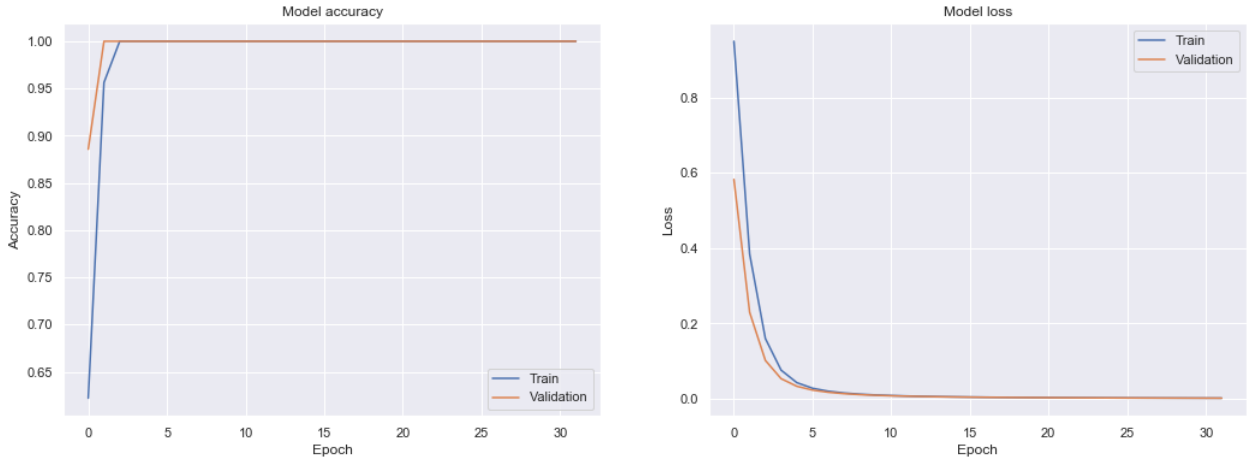
In Figures 6 and 8 histogram and Kernel Density Estimate of these parameters are presented superimposed to the final parameters frequency plots and distributions. It is crucial that the initial parameters are the same for both the clean data set and the noised data set cases, in that, being the system non-linear *per se*, different initial conditions would imply different dynamics. So by keeping fixed the initial status, the final parameters distributions for both linearly separable and not cases refer to solutions that evolve from the same initial configuration. In Figure 10a the **connections strengths** of the initial status are reproduced, in conjunction with the network topological structure.

5.2 System performance and final configuration

5.2.1 *Clean data set*

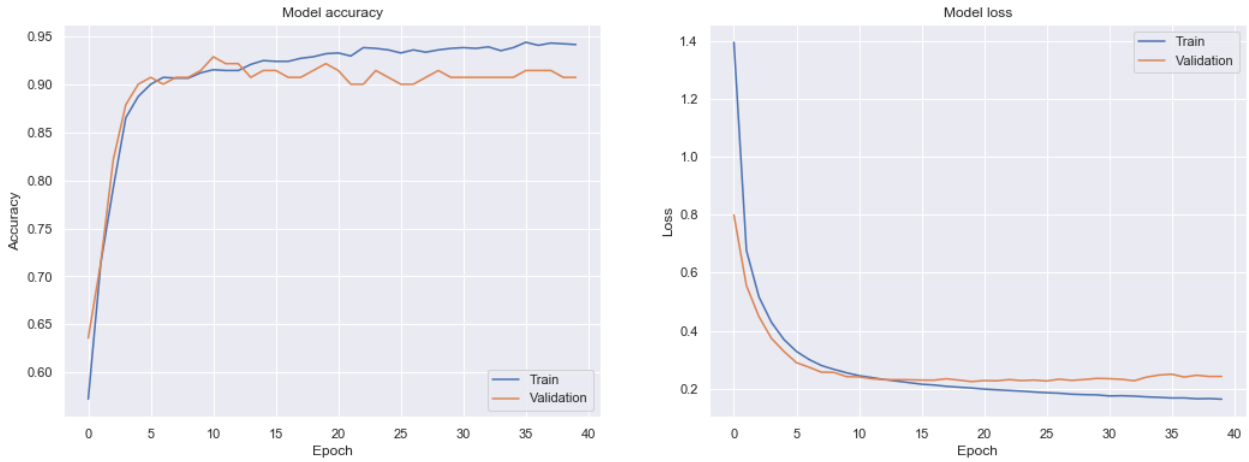
The simple neural network illustrated is trained with the data set. As said, a 0.7 fraction is used for training, a 0.1 of which is used as validation set, and the 0.3 kept apart before is used as test set. Being the data set linearly separable, in few epochs the accuracy metric attains the top value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

Learning time (epochs)	33
Training set accuracy	1.0
Test set accuracy	1.0
Test set loss	0.002



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

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



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

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

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 *Noised data set*

As expected, learning process in presence of noisy data is not as fast and accurate as in the former case. Again visualisation of the connections strengths in Figure 10c.

Learning time (epochs)	39 (Early stopped)
Training set accuracy	0.9413
Test set accuracy	0.9133
Test set loss	0.2283

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 linearly separable data set configuration, weights once the system is trained with the noisy data set configuration. Here the results refer to a cut-off threshold of $C = 0.35$. Observing the distributions in Figures 6 and 8 it seem a value that does not imply the loss of much information.

5.3.1 *Initial weights configuration*

Nodes	Motif ID	N_{real}	N_{rand} stats	N_{real} Z-score	N_{real} P-val	Unique Val	C real
4	392	104	78.6 ± 9.6	2.64	0.010	6	320.00

5.3.2 *Clean data set post-train configuration*

Nodes	Motif ID	N_{real}	N_{rand} stats	N_{real} Z-score	N_{real} P-val	Unique Val	C real
3	6	69	61.1 ± 2.6	3.00	0.000	10	173.37
4	14	25	17.3 ± 3.0	2.54	0.00	5	13.11
4	28	177	132.4 ± 13.5	3.30	0.00	5	92.82
4	280	264	213.3 ± 15.1	3.35	0.00	7	138.44
4	392	538	420.1 ± 28.6	4.12	0.00	6	282.12
4	904	51	5.6 ± 2.5	18.11	0.00	6	26.74
4	2184	122	102.6 ± 7.2	2.70	0.00	9	63.97

5.3.3 *Noisy data set post-train configuration*

Nodes	Motif ID	N_{real}	N_{rand} stats	N_{real} Z-score	N_{real} P-val	Unique Val	C real
3	6	85	73.3 ± 3.2	3.61	0.000	10	163.46
4	14	34	22.5 ± 3.9	2.98	0.000	6	11.84
4	28	257	187.8 ± 17.6	3.94	0.000	5	89.48
4	76	493	411.2 ± 26.1	3.13	0.000	8	171.66
4	204	24	5.3 ± 2.1	8.79	0.000	4	8.36
4	280	478	378.7 ± 26.4	3.76	0.000	9	166.43
4	392	765	613.0 ± 39.1	3.88	0.000	7	266.36
4	904	67	8.6 ± 3.4	17.18	0.000	6	23.33
4	2184	218	183.3 ± 9.8	3.55	0.000	9	75.91

6 Conclusions

6.1 Differences between the first version of this report

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 the neural network in the stage of learning. By looking at Figures 10a and 10c 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 and social sciences, refer to Kashtan and Alon, 2005.

What is immediately observable is an increase in weights magnitude, both positive and negative, and a sharper cross-like theme, 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, 10c and in the fatter distributions tails in Figures 8c, 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.

6.2 Further improvements

6.2.1 *Steps accomplished*

It has been implemented the following:

- Since in the work made by Kashtan and Alon (2005) the neural network setting was supposed to give the connections the only values $\{-1, +1\}$ (and, in the deployment of the mentioned program, negativity of edges strengths were not considered, then –**I believe**– all the connections are set the $+1$ value). In the present work the same is made, almost in this first experimental stage. The weights matrices have been binarized according to the above reported cut-off threshold.

- Once that done, the same motif-search analysis on the resulting (semi-sparse) graph in Figure 11 and observe which are the patterns emerging has been performed: Using the software published some motifs have been identified. The tables above shown report the motif identifier, through which one can check in the motif dictionary companion document the actual shape of the sub-graph discovered.

About this latter aspect: The documented motifs are composed of 3 and 4 nodes. Almost all of the pattern made of 4 nodes may seem not appropriate for this kind of neural network, in that no connections between nodes of the same layer are allowed. However, if one looks at motif labelled with "id28" in the motif dictionary document (available at the page the last reference above points to), it could be that the top-right node belongs to a first layer, the top-left node belongs to a second layer and both the bottom nodes instead belong to a third layer. One of the most recurrent motifs found (the only one, for the initial weights configuration), tagged "id392", could be interpreted as: the bottom-left node belongs to a first layer, both the top nodes belong to a second layer and the bottom-right node belongs to the third layer. This, in particular, referring to the above tables, the most recurring pattern.

It is in order to underline that these results are the basic backbone of the whole picture: As pointed out by Kashtan and Alon (2005), one could expect to observe some significant and informative trace of the environment in the network topology once he/she performed modularly varying goal (MVG) training. Here the way to follow to implement this latter is still not implemented, and is work in progress.

(a) Id28 **(b)** Id392

Figure 5: Typical motifs.

Nevertheless it could be interesting to investigate a possible relationship between

- the type of motif surfaced;
- the probabilistic signature of the data set;
- the same quantity for the motifs and/or weight in trained configurations.

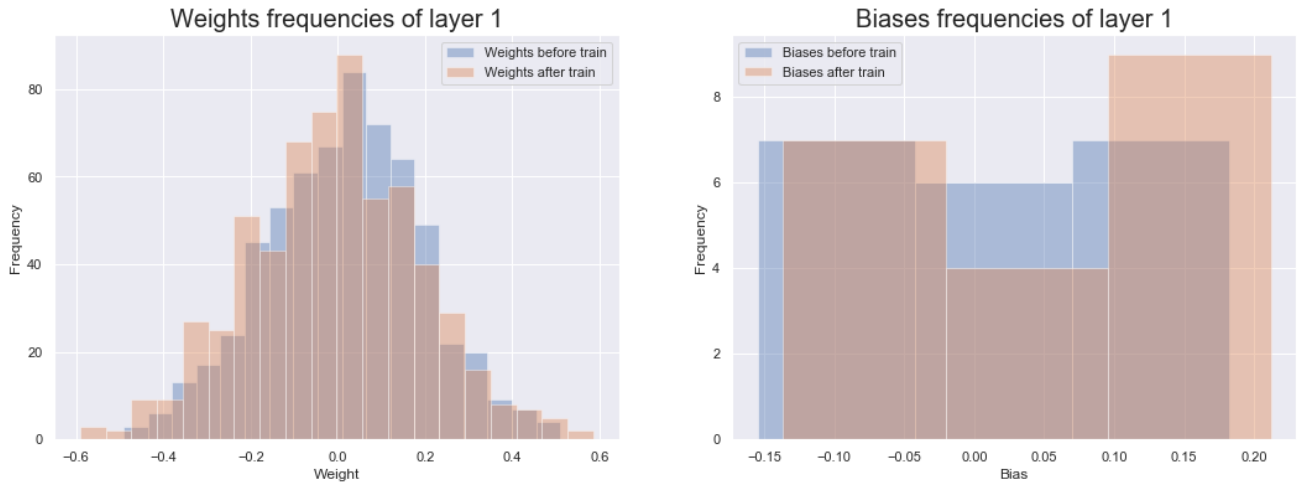
It has been proven that the data set statistical structure affects the dynamics of learning, as well as the development of semantic distinction of the internal representation (referring to Saxe *et al.*, 2018), but the evolution of the neural network in terms of structural and topological shape in response to a given and statistically known data set has not yet.

6.2.2 *For the sake of the broader scope of the whole work*

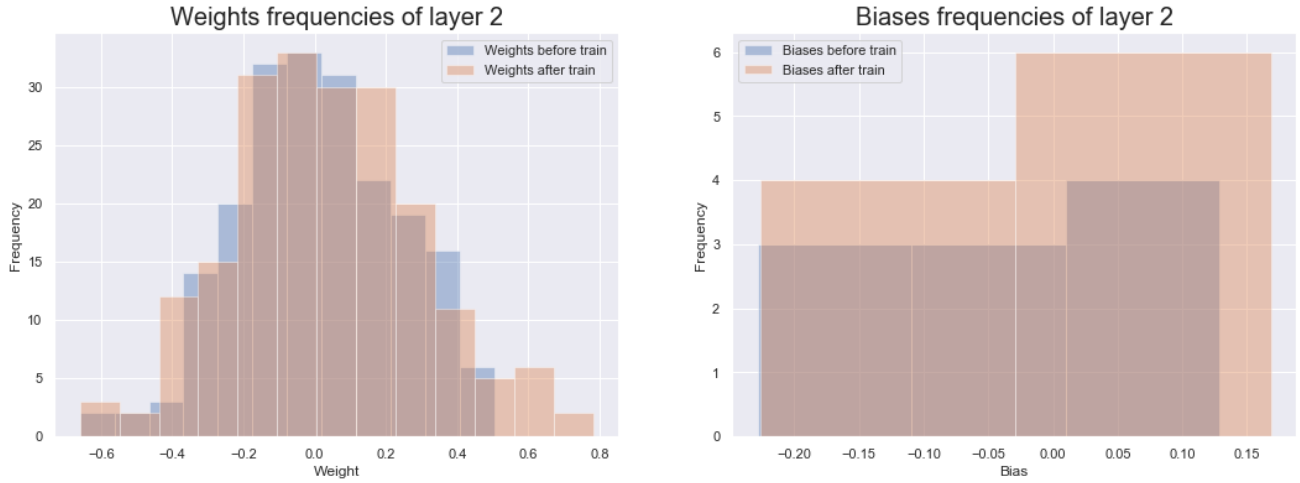
As pointed out in Kashtan and Alon, 2005, the final outcome of this kind of investigation should be the detection of some structures that come to exist in a **mutating environment**, that is: There should be not one single fixed task for the system to accomplish. Rather, if this is sequentially and periodically exposed to two different tasks, partially overlapping, it is expected that in such an environment one observes the emergence of some characteristic pattern in the network topology.

This expectation stems from the fact that the system learns cyclically the two representations: let us assume that it is to be learned by the network to detect a cat in a set of pictures and an elephant. These two categories share some common features, in that both have four legs. But other than that, there are few, if any, more resemblances (ears differ a lot, the former has hair whilst the latter does not. Elephants have fangs and cats do not, and so forth). What one naively expects then is that there should be a certain internal encoding of the *four legs* feature, and this could translate in a certain motif. In this fashion, motifs encode the **overlap** between two different tasks. If these two tasks exactly match, then all of the internal representation of the first one is *reused* for the second. There should not be, in principle, matter of difficulty in learning the second task. On the other hand if the two goals are completely disjoined, the system must start the training from scratch at every task switch, in that there is no common trait between the two goals.

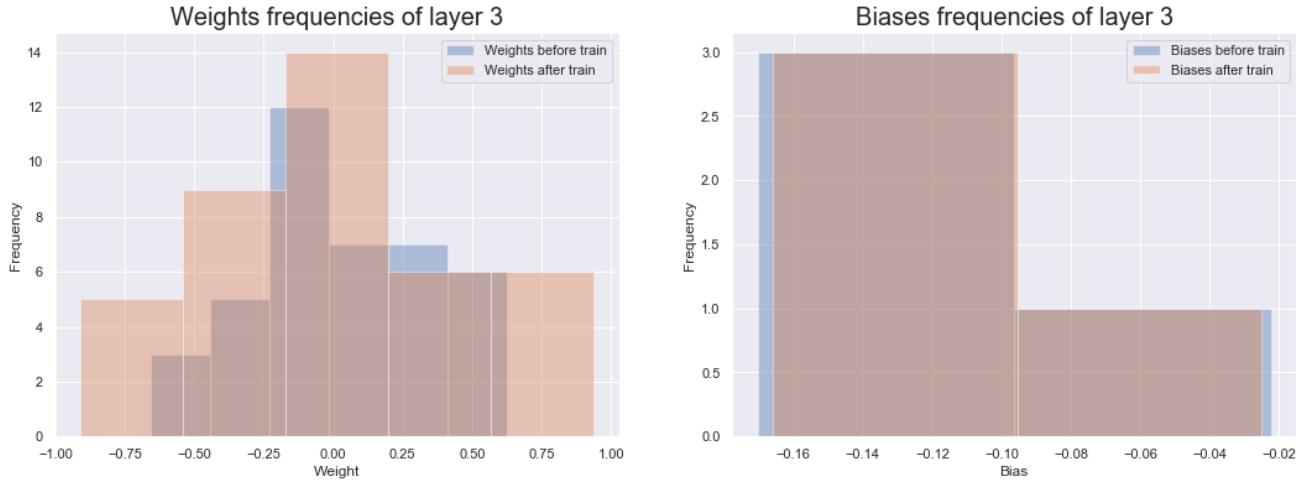
So, how can one quantify such an overlap, and then put in clear relationship the motif, its role in the task accomplishment (i.e. correct classification)? Is it feasible through the parameters of the trained network? Does it make any sense to search for a probabilistic footprint in both the parameters distributions and the input data distribution? Where should one look for proper tools? Graph theory? Unsupervised learning via Autoencoders and/or distribution inference?



(a) First layer

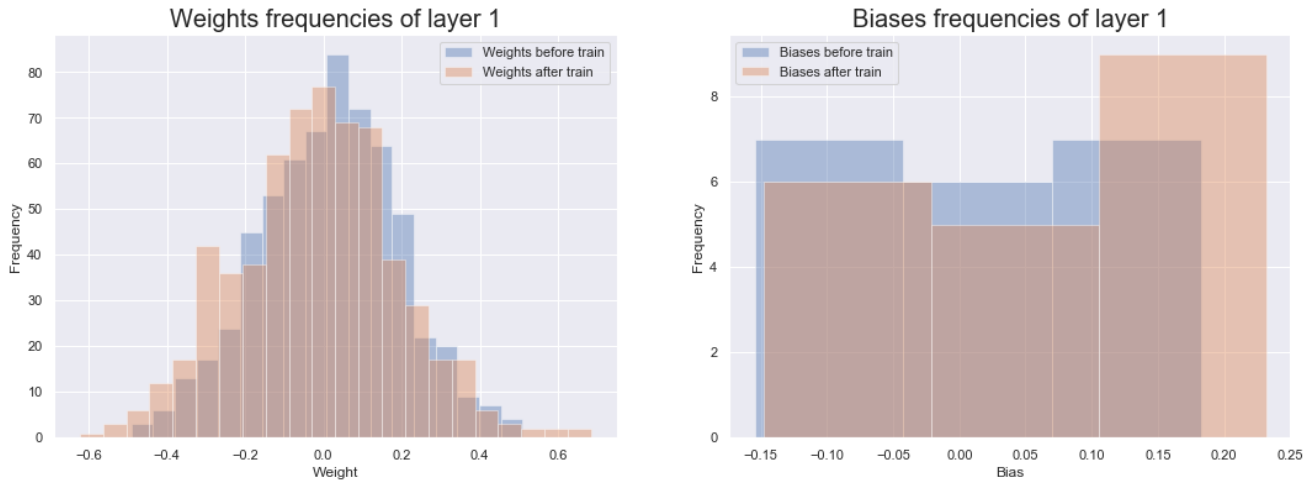


(b) Second layer

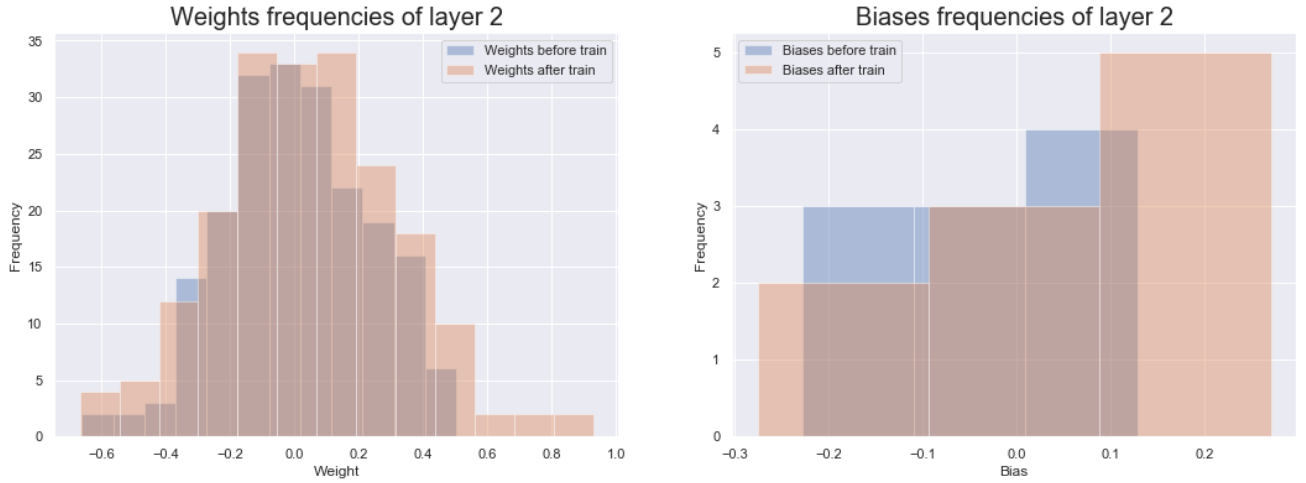


(c) Third layer

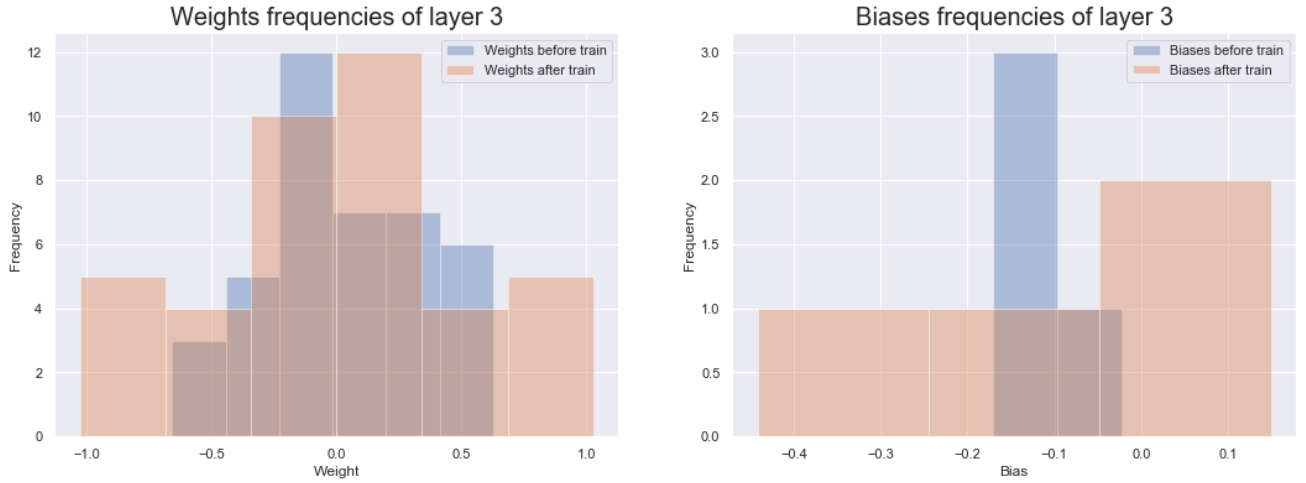
Figure 6: Frequency plots of the parameters before and after training. Linearly separable data set.



(a) First layer

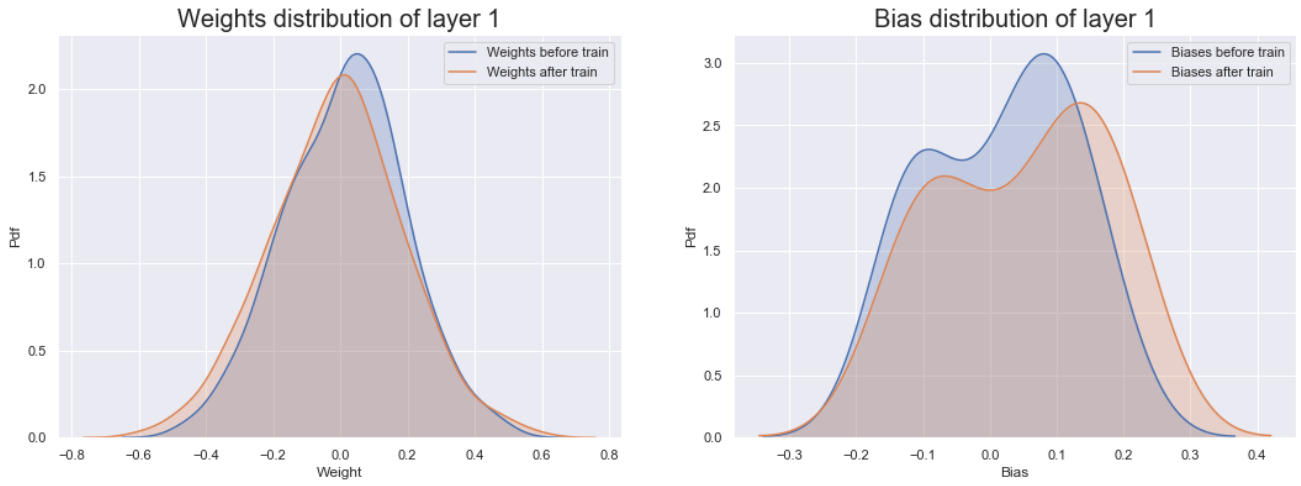


(b) Second layer

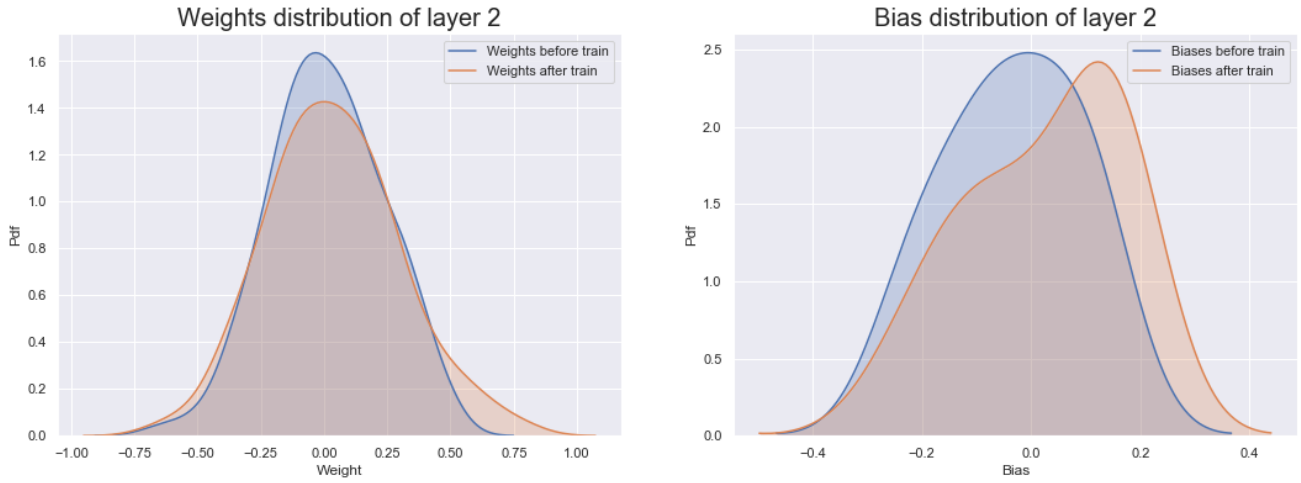


(c) Third layer

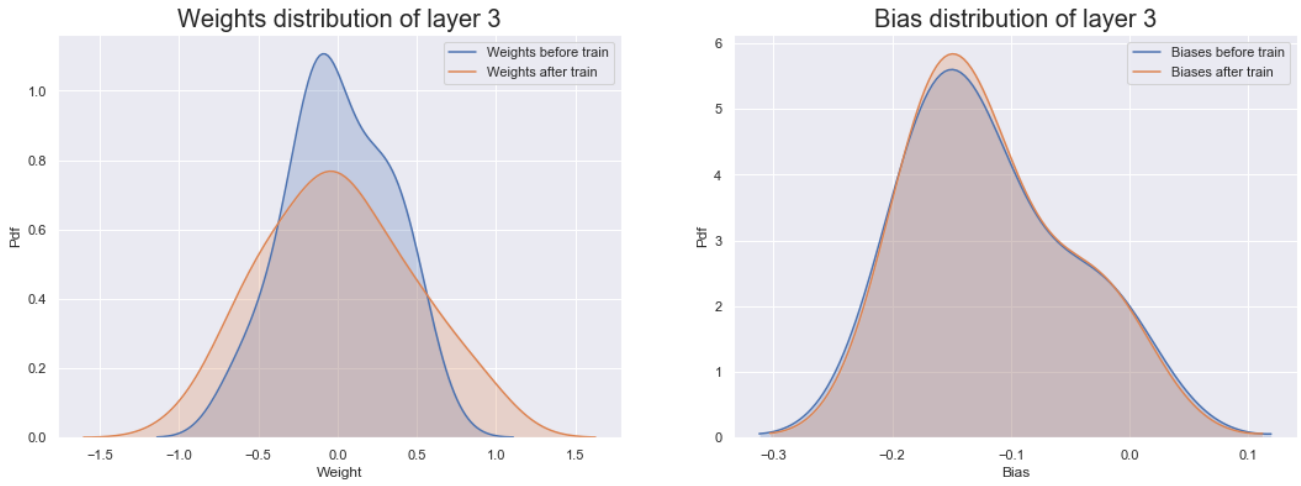
Figure 7: Frequency plots of the parameters once the model is trained on non separable data.



(a) First layer

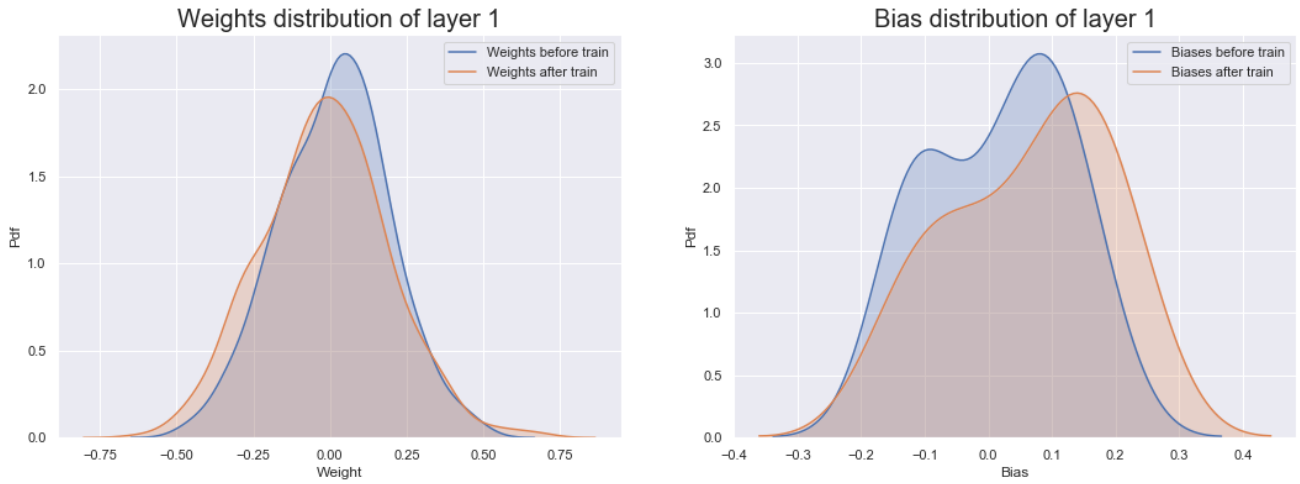


(b) Second layer

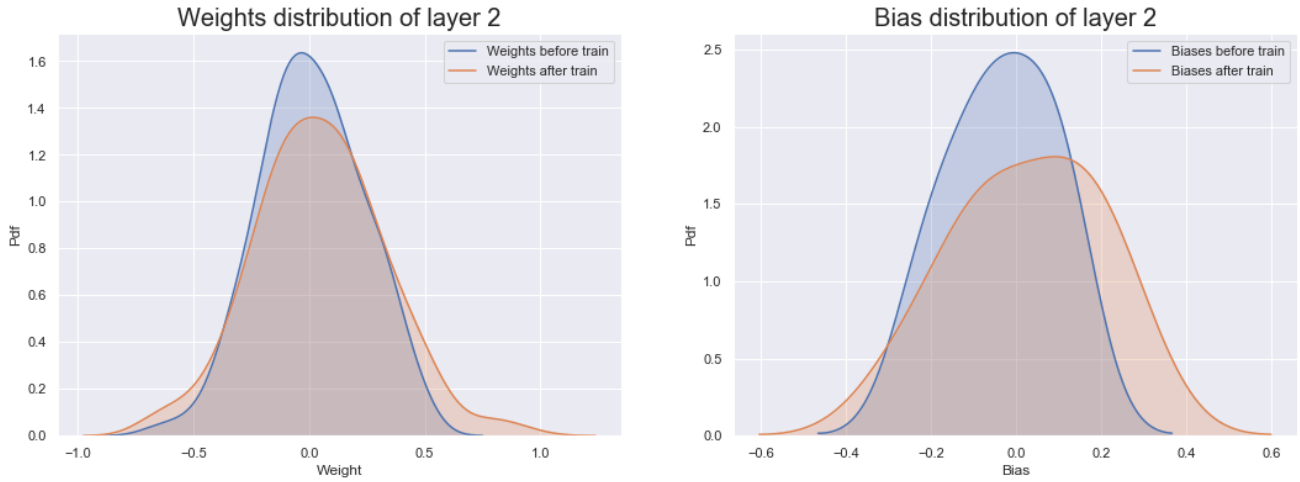


(c) Third layer

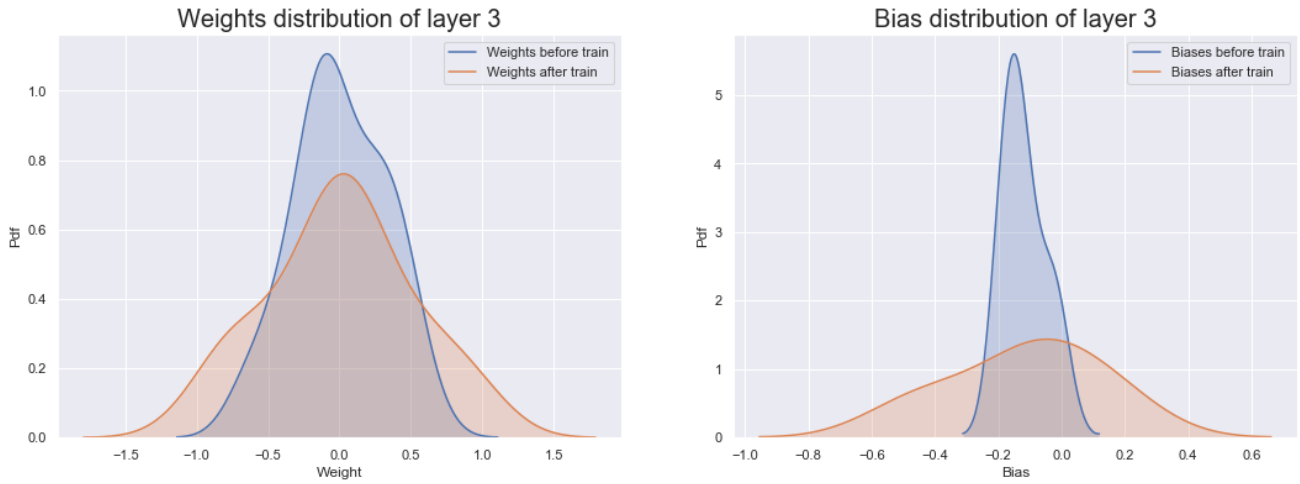
Figure 8: KDE of the parameters before and after training. Linearly separable data set.



(a) First layer

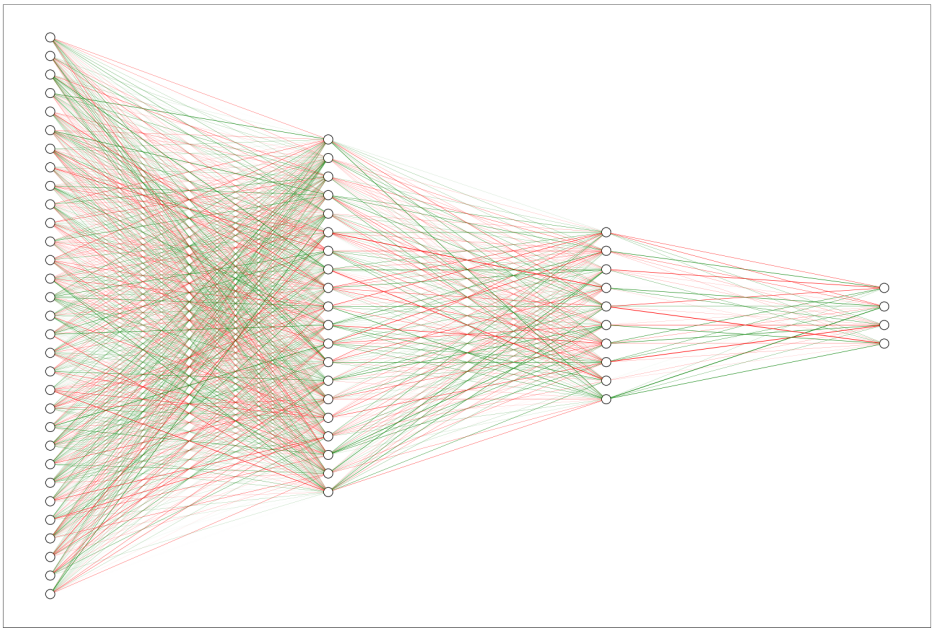


(b) Second layer

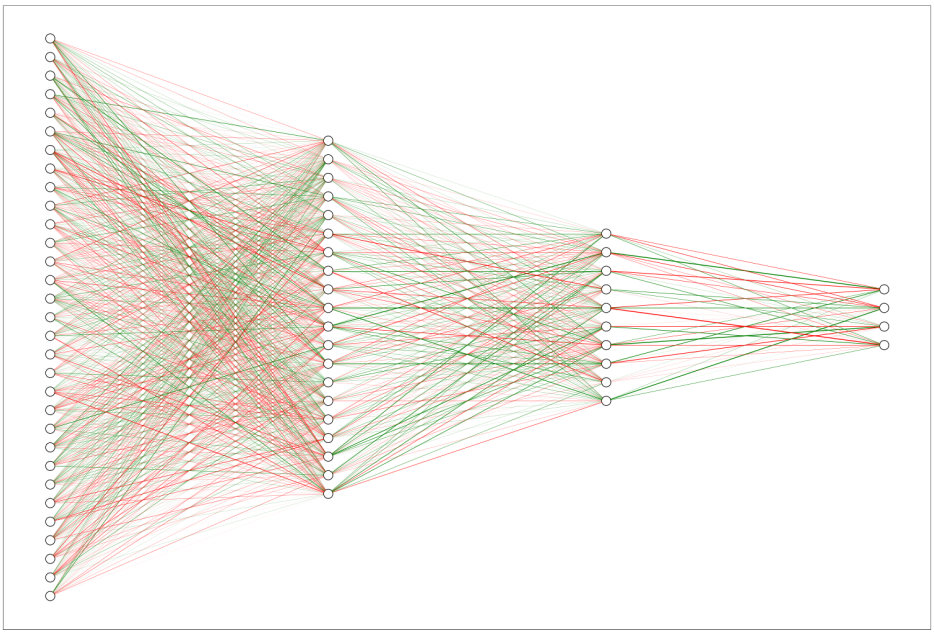


(c) Third layer

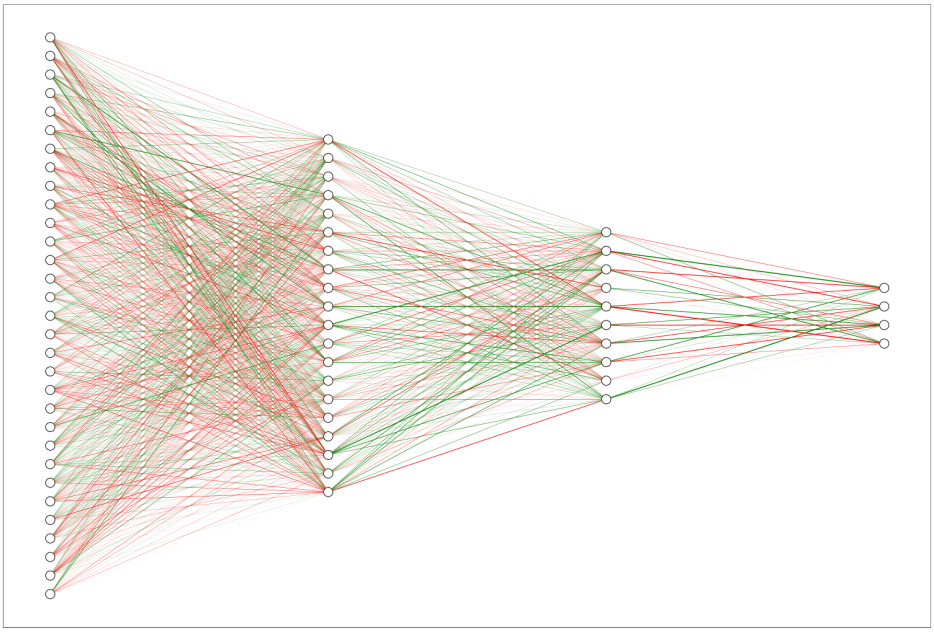
Figure 9: KDE of the parameters once the model is trained on noisy data.



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

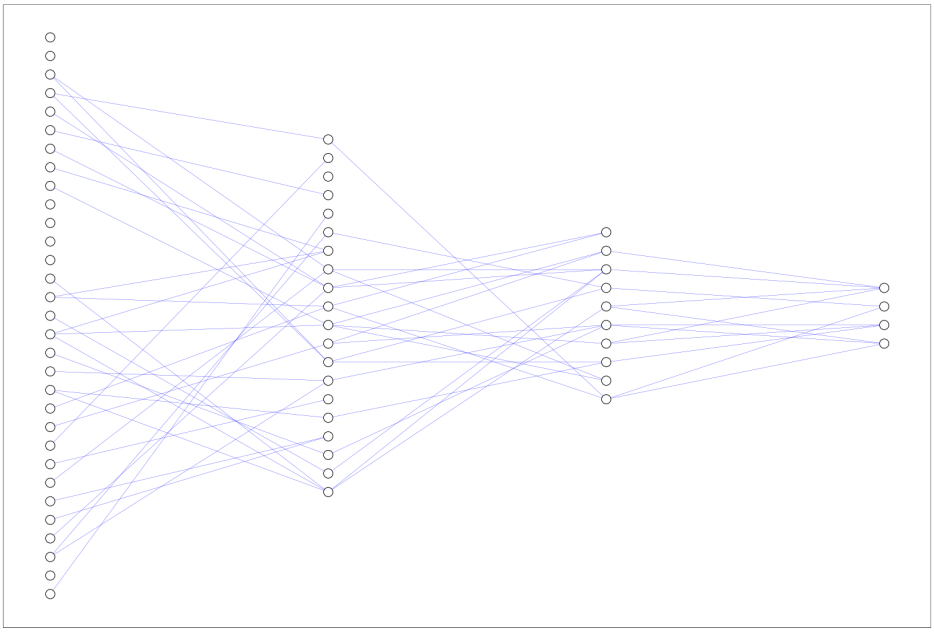


(b) Visualisation of the trained network topological structure with connection strengths highlighted in red for negative valued weights and green for positive values ones. This plot refers to the clean data set.

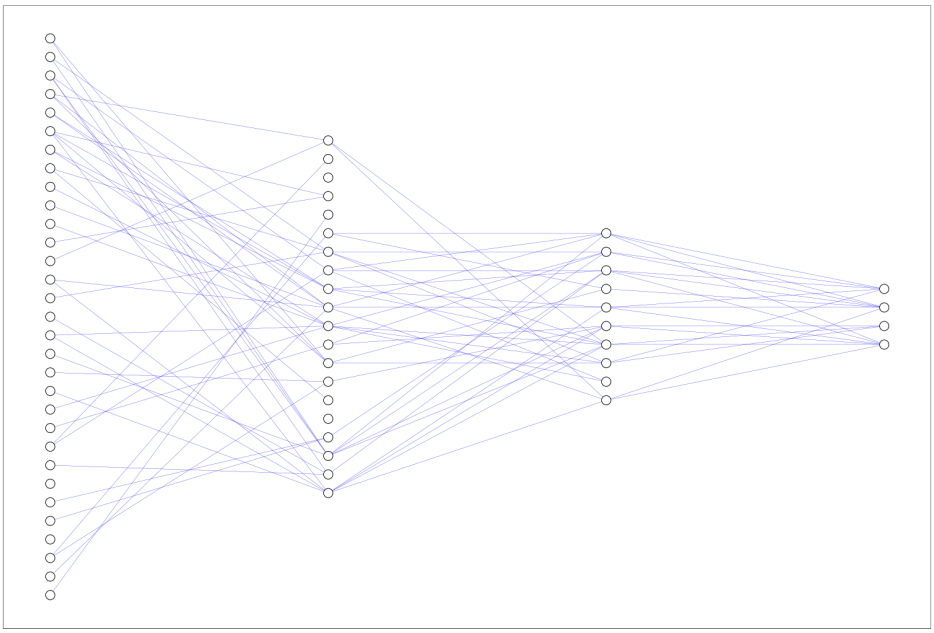


(c) Visualisation of the trained network topological structure with connection strengths highlighted in red for negative valued weights and green for positive values ones. This plot refers to the noisy data set.

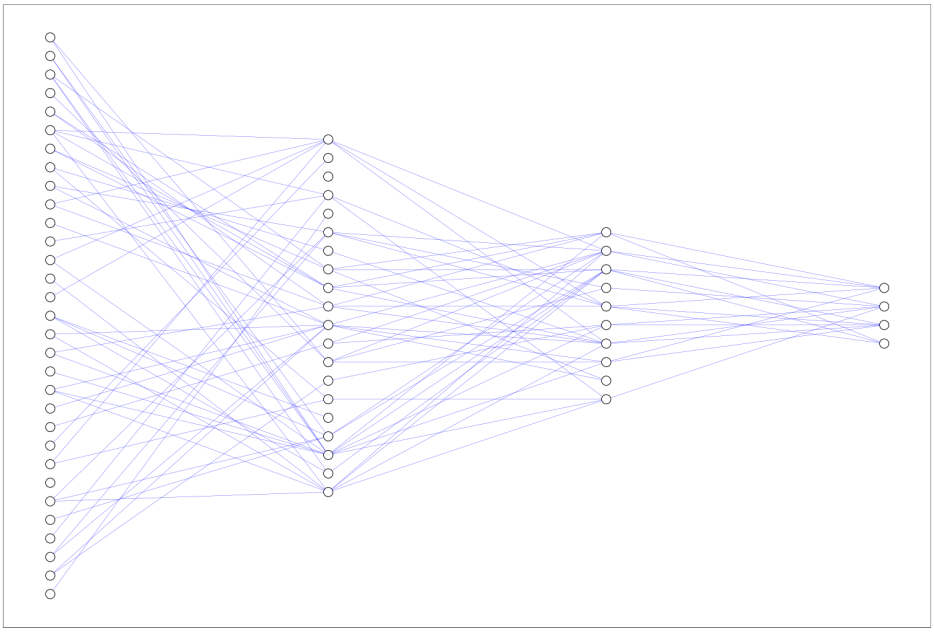
Figure 10: Visualisation of the network strengths.



(a) Visualisation of the untrained network representation as graph. $C = 0.35$ used as cutoff threshold.



(b) Visualisation of the trained on the clean data network representation as graph. $C = 0.35$ used as cutoff threshold. threshold.



(c) Visualisation of the trained on the noisy data network representation as graph. $C = 0.35$ used as cutoff threshold.

Figure 11: Visualisation of the graph representations of the untrained and trained networks.