Neural network response in learning from synthetic data

Contents

1 What's new

Whit respect to the previous version, some aspects have been rectified.

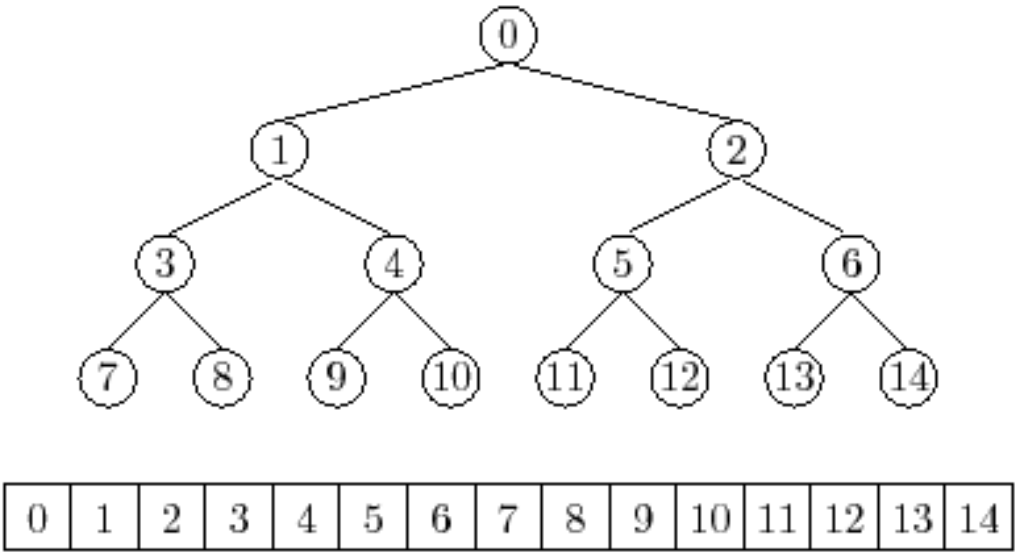


Figure 1: From OpenDataStructure site.

The results shown are heavily computational-reliant. Here are reported comments, rationales and descriptions of the process as well as graphical results, but it is thought better to keep the Python code apart from this document. It is checkable on this GitHub repo in Jupyer notebook format. Alongside with the code, markdown documentation is feasible. All of the plots herein reported are in the notebooks as well. Further comments are restricted to these companion notebooks.

2 Data set

Data set is generated following a **binary tree** fixed structure, refer to Figure 1. All of the nodes of the tree are the **features** of one single **pattern**. As a simple (an incomplete) example consider the following **decision rules** (associated with the non-leaves nodes) and **final classes** associated with the leaves. Further considerations on this convention are given below.

2.1 Single pattern generation

One pattern is the collection of *all* the node values of the array-represented tree. 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. a row of $\boldsymbol{X}$, $\boldsymbol{x}^{\mu}$, $\mu = 1,\ldots,M$, is one single N-dimensional data entry.

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.

The amount of data items generated is of 2000 examples.

Algorithm 1 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 *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

2.3 Noising the data set

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 $\boldsymbol{X}$.

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.

As an additional operation, PCA-preprocess is performed. It is not helpful to train the models on the preprocessed data, in that, restricting ourselves to the case of two Principal Components (PCs), it comes handy in visualising the data scatter in the PCs space, but it does not grant that in such a dimensionally-reduced space there exist a well defined decision boundary. Luckily SVM overcome this obstacle via the *kernel trick*. Even if the data were not linearly separable, this upgraded dot product allows to project data points in higher dimensional space, thus finding a way to set a sharp decision locus.

3.1 PCA preprocessing

Data set characteristics. For the data here presented a number of 127 features is present, corresponding to a tree depth of $D = 7$ levels. From now on, however, for further neural system visualisation ease, it is though to be sufficient to dispose of $N = 31$ features (tree depth $D = 4$) and 4 classes, for $M = 2000$ data items. Results in the next two Subsections are produced in such a fashion.

3.1.1 *Clean data set*

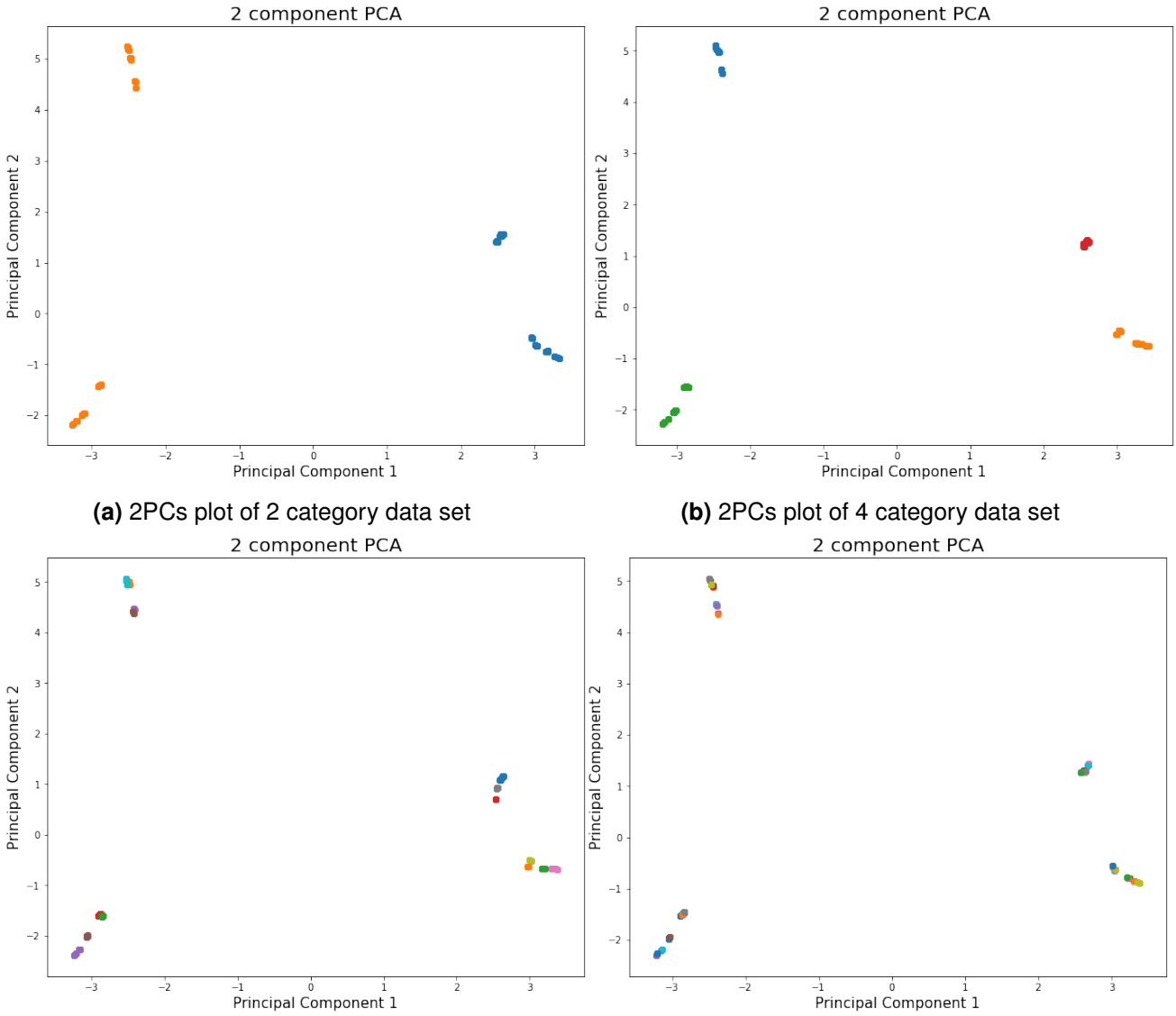
In the linearly separable case, data points in reduced space seem to settle along *invariant manifolds*, hyperbola-shaped. Regardless of the number of classes, and features, this behaviour is observed, Figure 2.

3.1.2 *Noised data set*

Here it is observed a greater scatter, in that not all the data patterns in one class have the same exact features values. However, the recurrent structure is also here observable, Figure 3.

3.2 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



(a) 2PCs plot of 2 category data set

(b) 2PCs plot of 4 category data set

(c) 2PCs plot of 16 category data set

(d) 2PCs plot of 63 category data set

Figure 2: Visualisation of PCA preprocessed data, different number of categories.

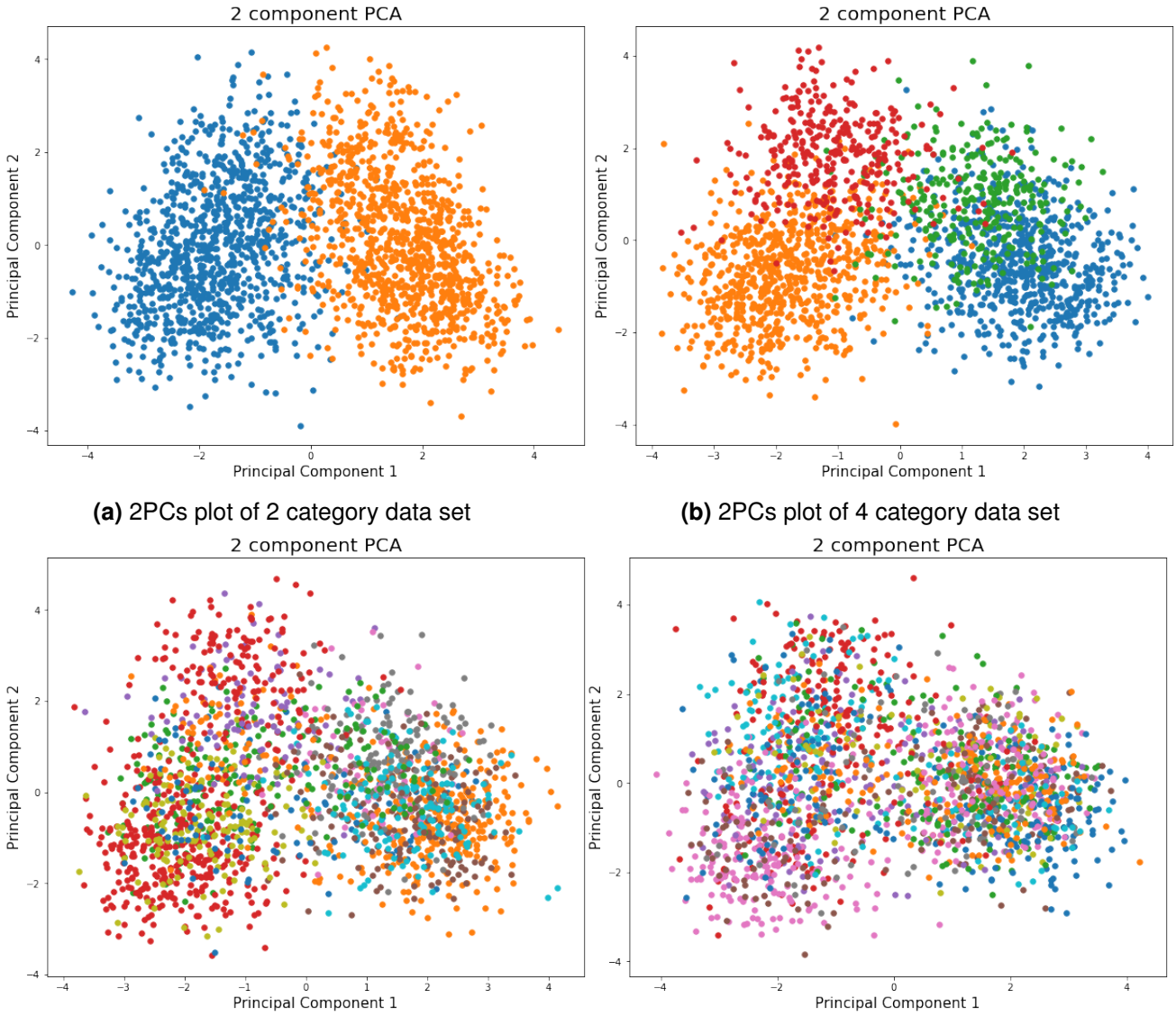
in the potential absence of a number of classes in the test set, with consequent non correct classification of such samples. This motivates to rise ϵ to 0.3. In this case, any of the categories is left apart and all of them are included in the training, of course, and in the test stages. If the classes are 4, as in the present case, this peril is all the way avoided.

3.2.1 *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.

3.2.2 *Noised data set*

By looking at Figure 3b, it is clear that in two dimensions, linear separability is not feasible, but kernel SVM operates in higher dimensions (as said, PCA is used for visualisation purpose, training is performed on *raw* data). A linear kernel SVM attains 0.90333 of accuracy, while a Gaussian kernel SVM gains some accuracy more, namely 0.935.



(a) 2PCs plot of 2 category data set

(b) 2PCs plot of 4 category data set

(c) 2PCs plot of 16 category data set

(d) 2PCs plot of 64 category data set

Figure 3: Visualisation of PCA preprocessed noisy data, different number of categories.

3.3 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.

3.3.1 *Clean data set*

Also this model attains the best in accuracy, 1.0.

3.3.2 *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.

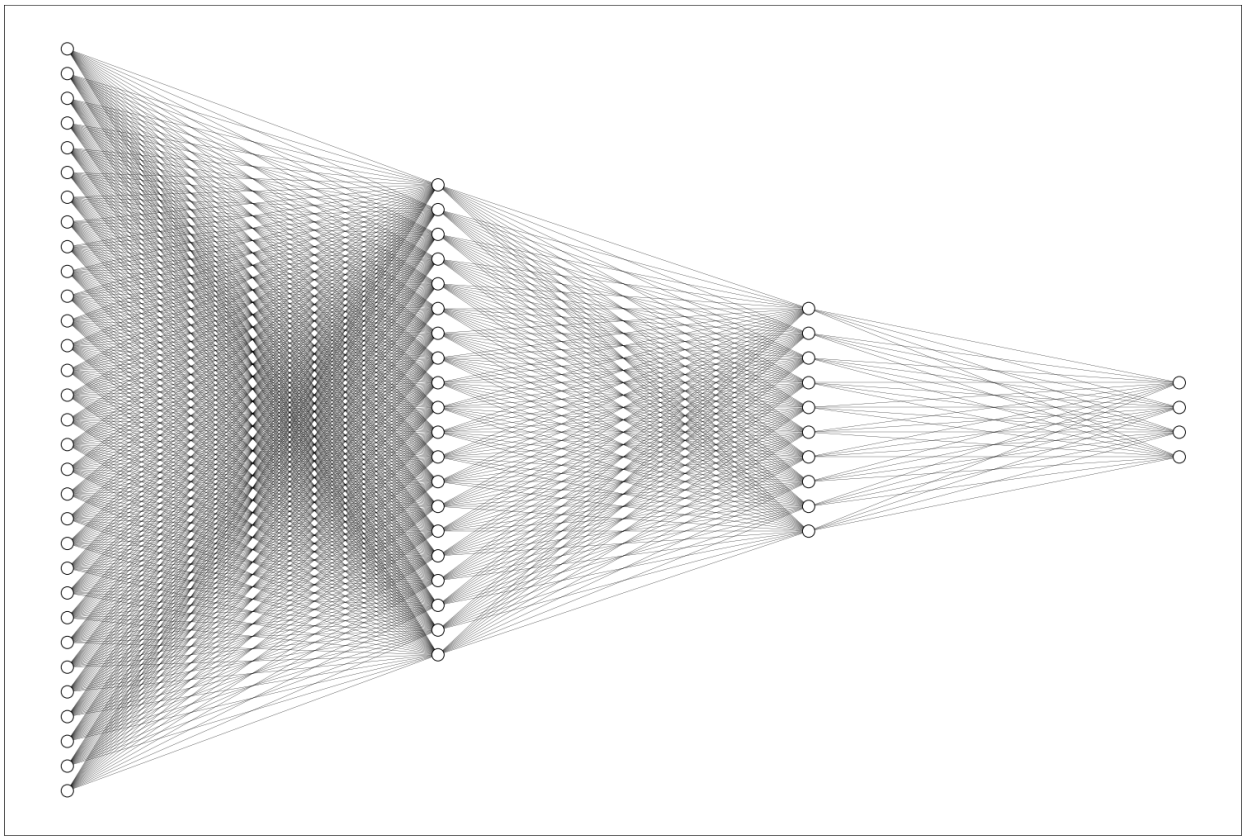


Figure 4: Visualisation of the network topological structure.

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 4.

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:

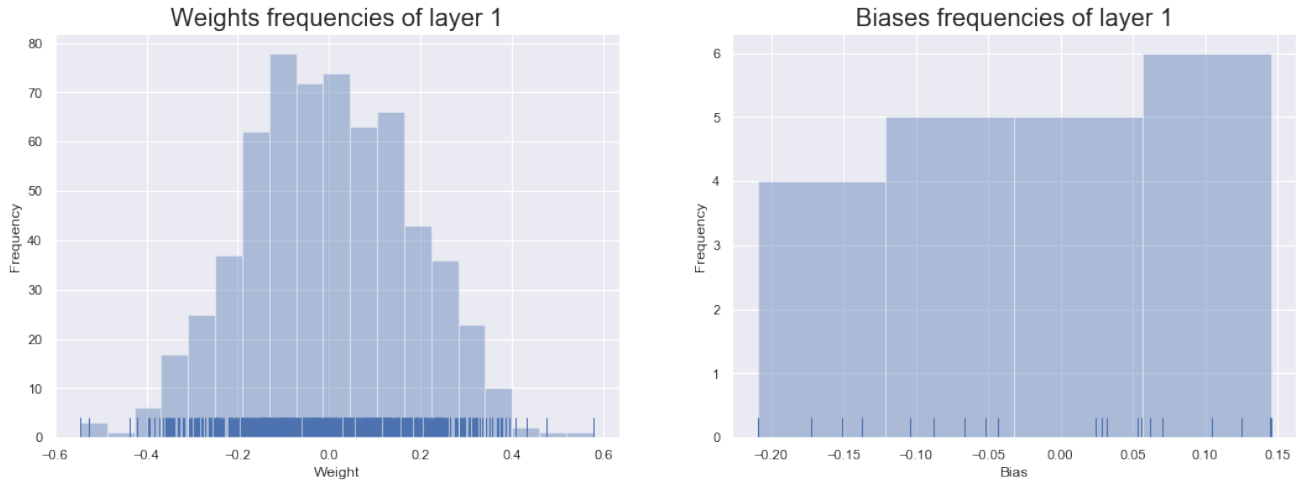
Architecture		
Layer	Units	Activation
1 (input)	$N = 31$	-
2 (hidden)	20	ReLU
3 (hidden)	10	ReLU
4 (output)	4	Softmax

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

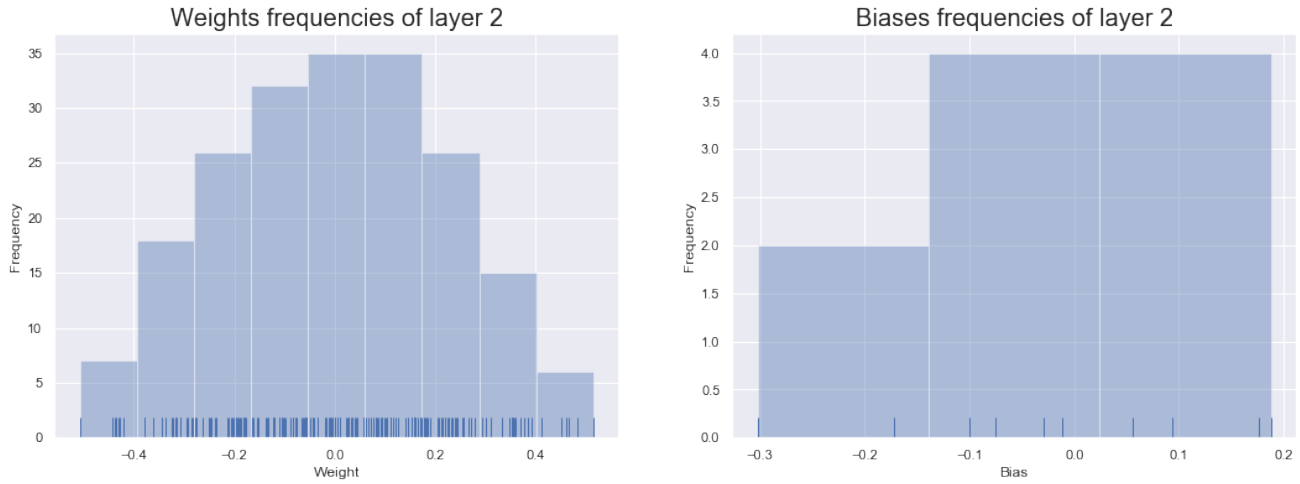
5.1 Initial values of weights and biases

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.

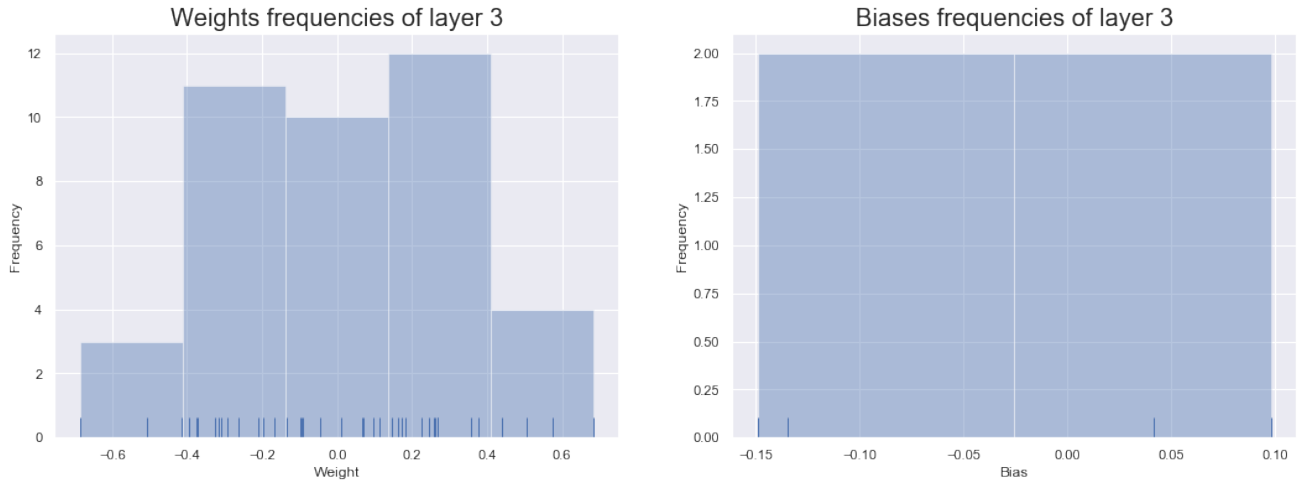
In Figures 5 and 6 histogram and Kernel Density Estimate of these parameters are presented, for the **untrained** network. In Figure 7 the **connections strengths** are reproduced, in conjunction with the network topological structure.



(a) First layer

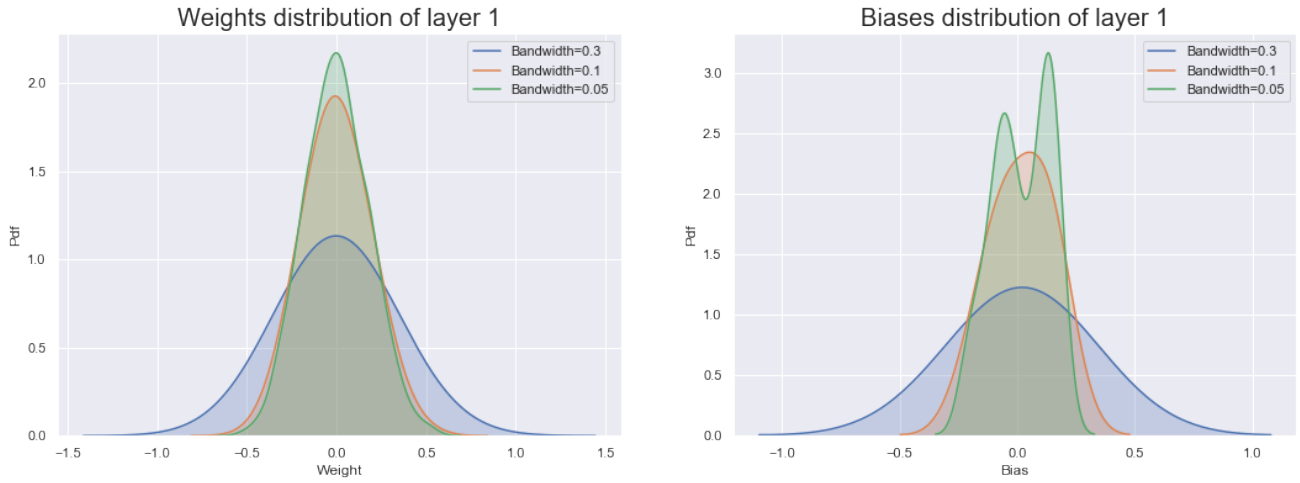


(b) Second layer

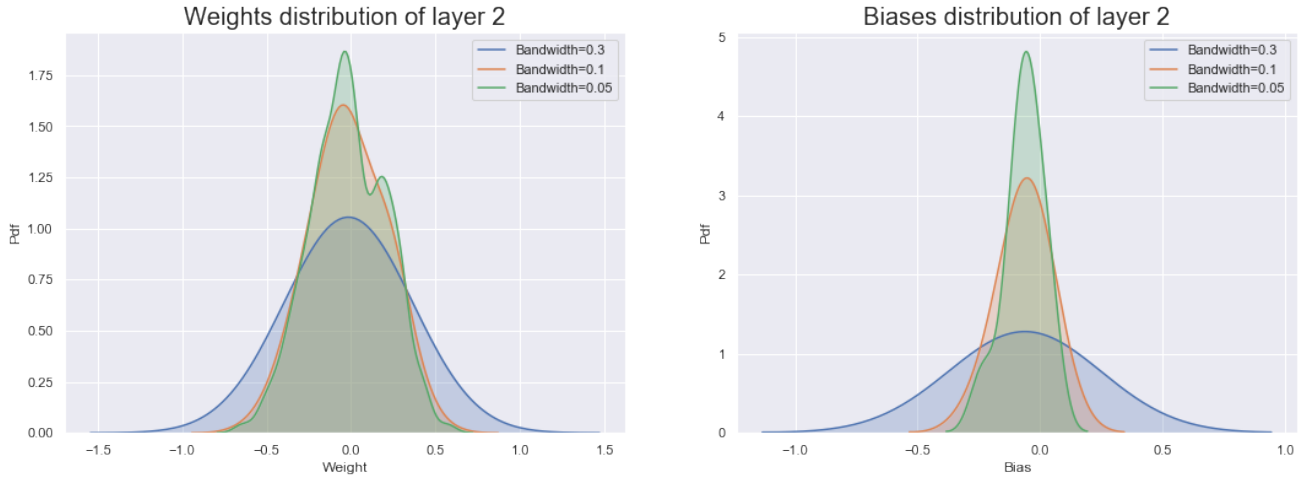


(c) Third layer

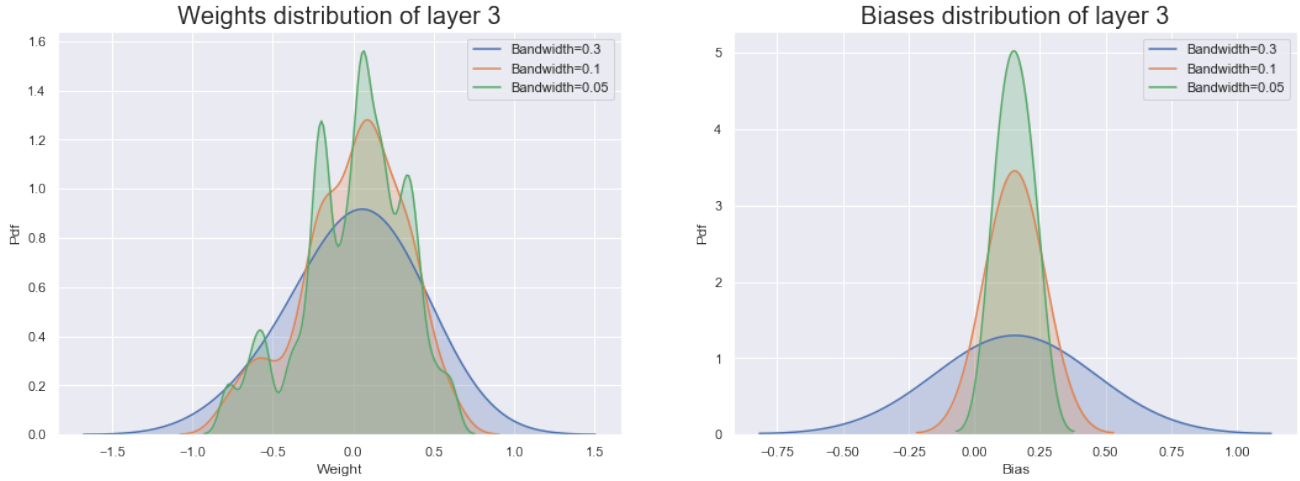
Figure 5: Frequency plots of the initial parameters.



(a) First layer



(b) Second layer



(c) Third layer

Figure 6: KDE of the initial parameters.

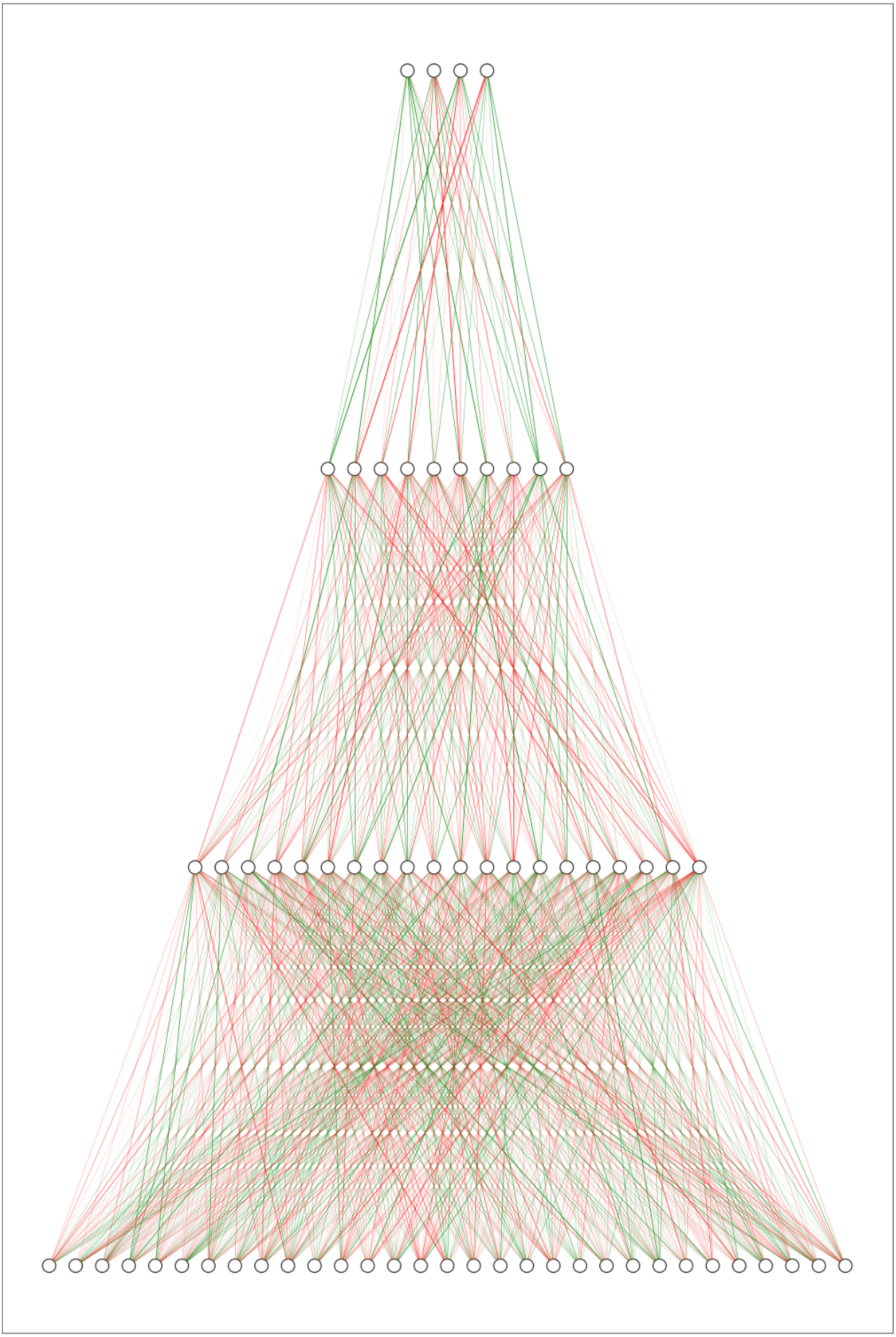
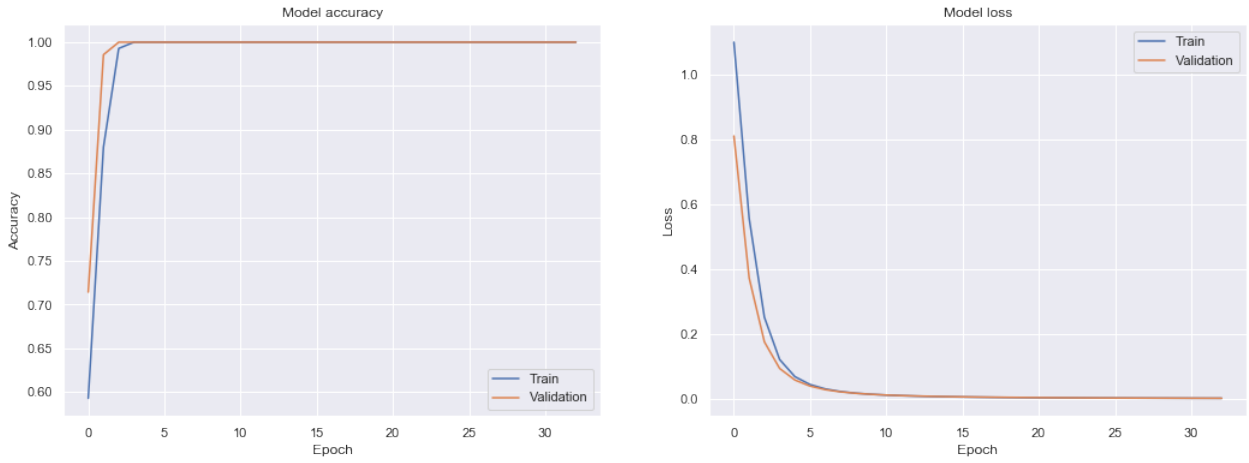


Figure 7: Visualisation of the untrained network topological structure with connection strengths highlighted in red for negative valued weights and green for positive values ones.



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

Figure 8: Accuracy and Loss profiles, clean data set.

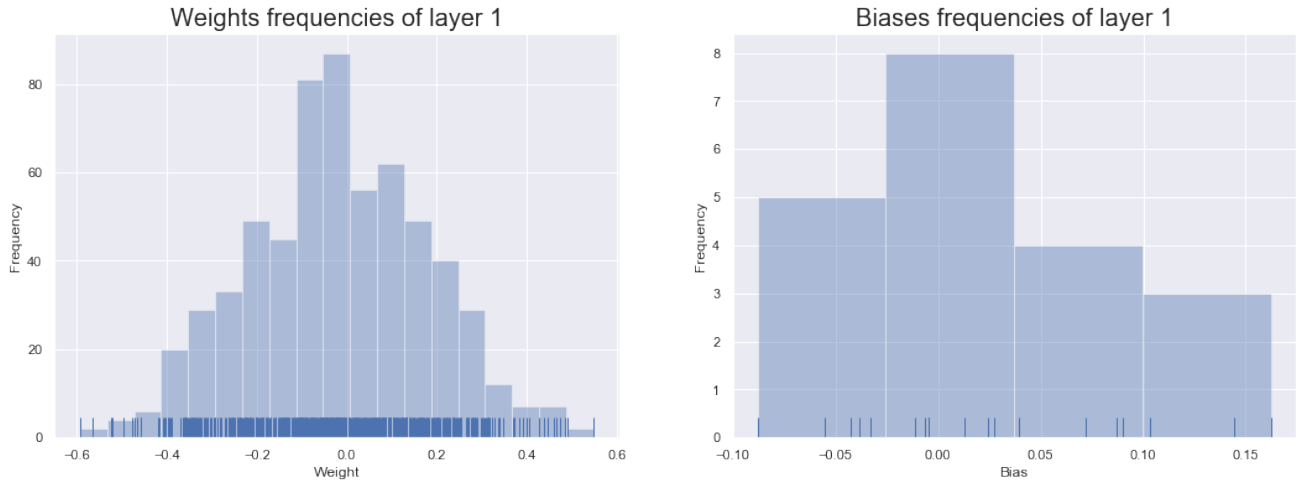
5.2 System performance and final configuration

5.2.1 *Clean data set*

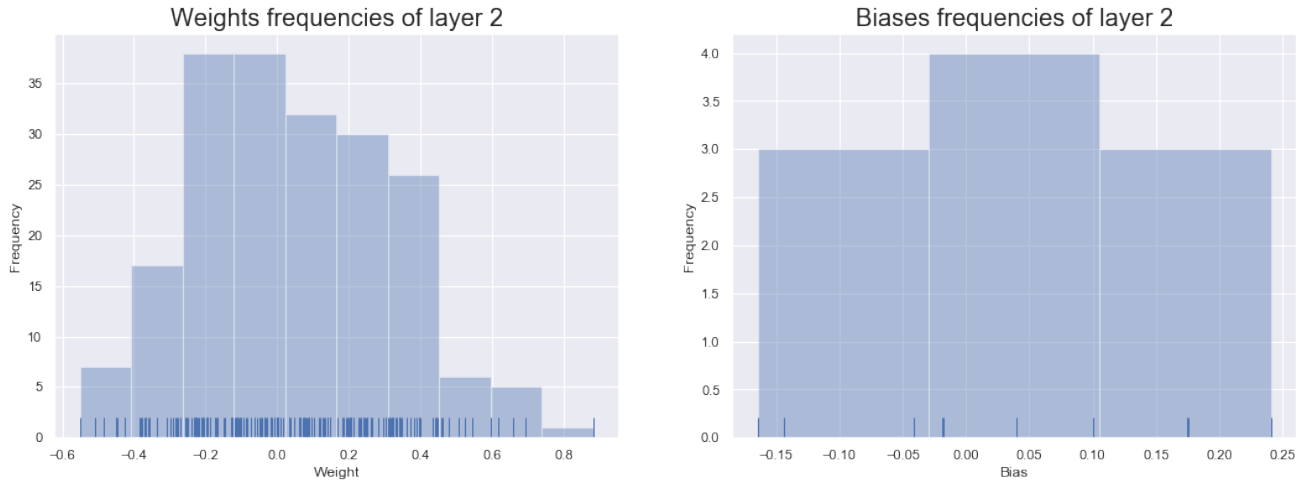
The simple neural network illustrated is trained with the data set. As said, a 0.7 fraction is used for training, a 0.1 of which is used as validation set, and the 0.3 kept apart before is used as test set. Being the data set linearly separable, in few epochs the accuracy metric attains the top value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

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

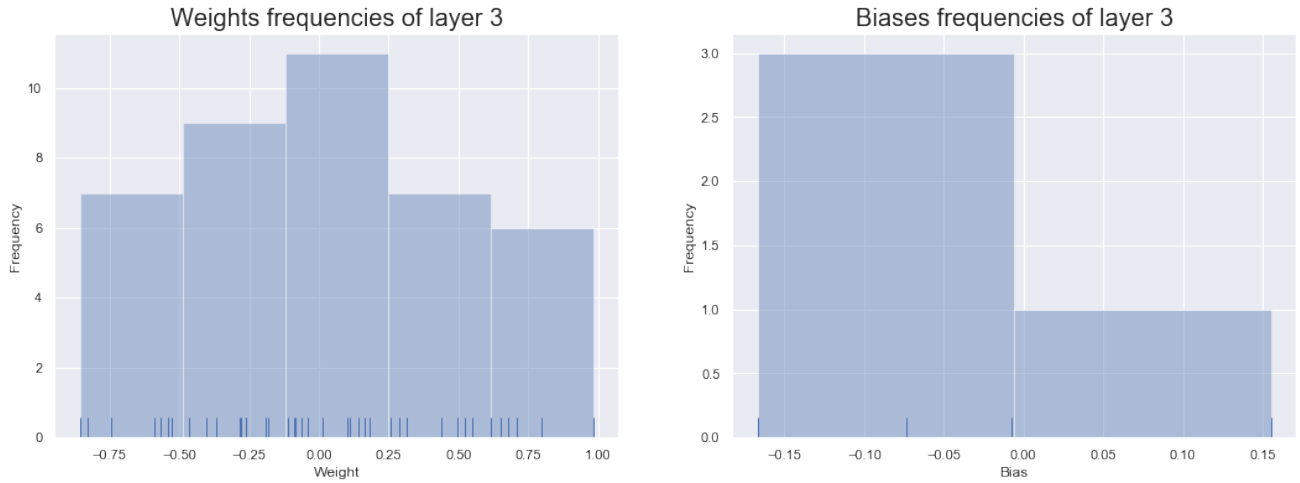
As final result, weights and biases distributions are also reported for the trained network, alongside with the connections strengths visualisation.



(a) First layer

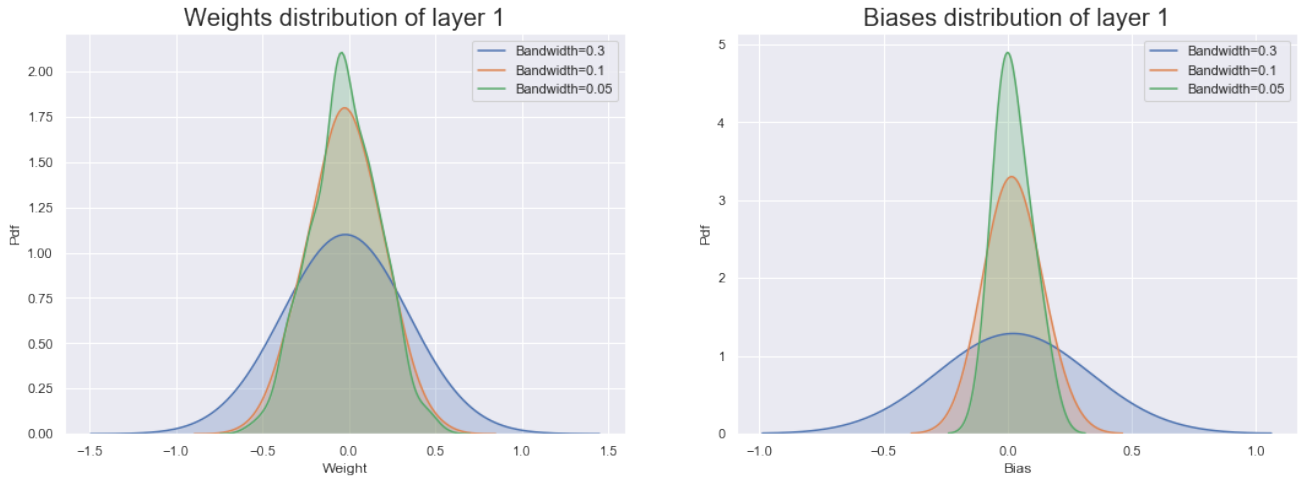


(b) Second layer

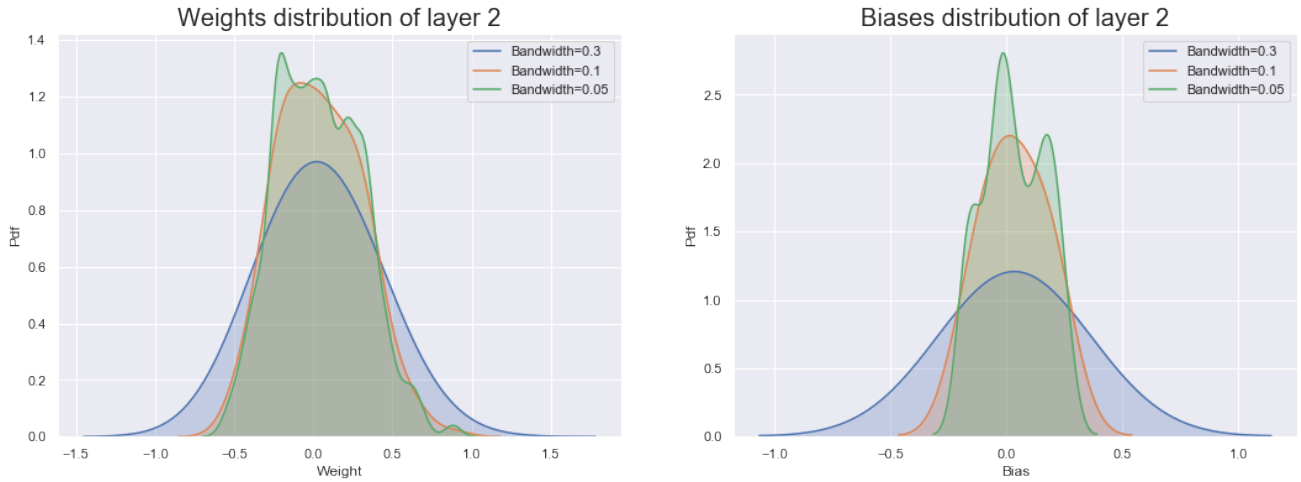


(c) Third layer

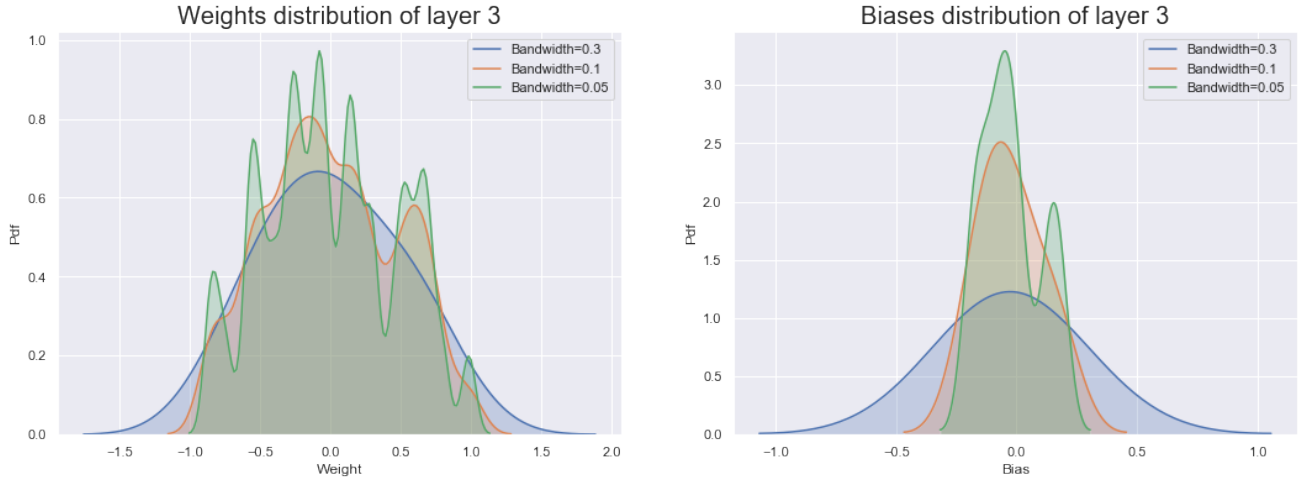
Figure 9: Frequency plots of the parameters once the model is trained on linearly separable data.



(a) First layer



(b) Second layer



(c) Third layer

Figure 10: KDE of the parameters once the model is trained on linearly separable data.

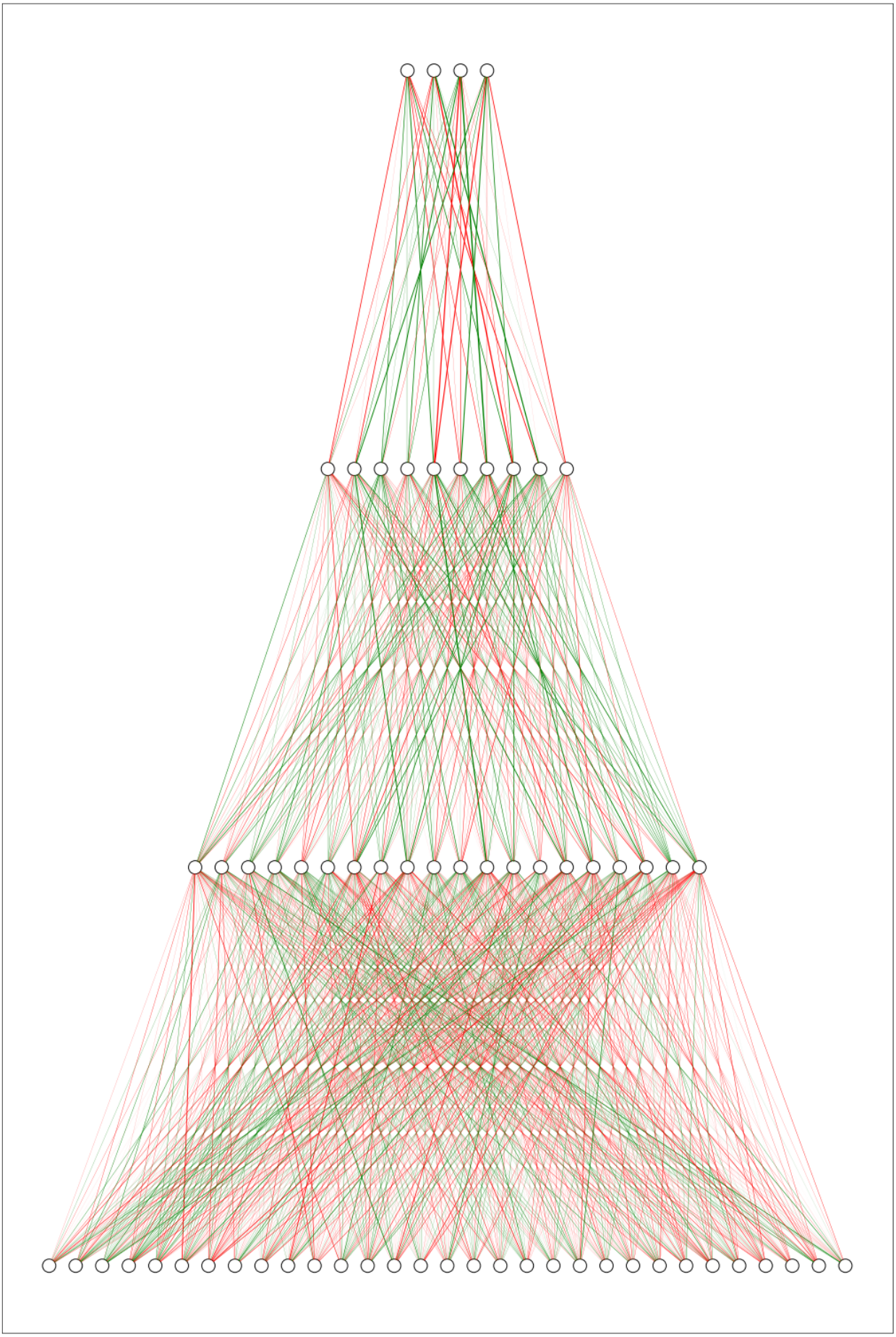
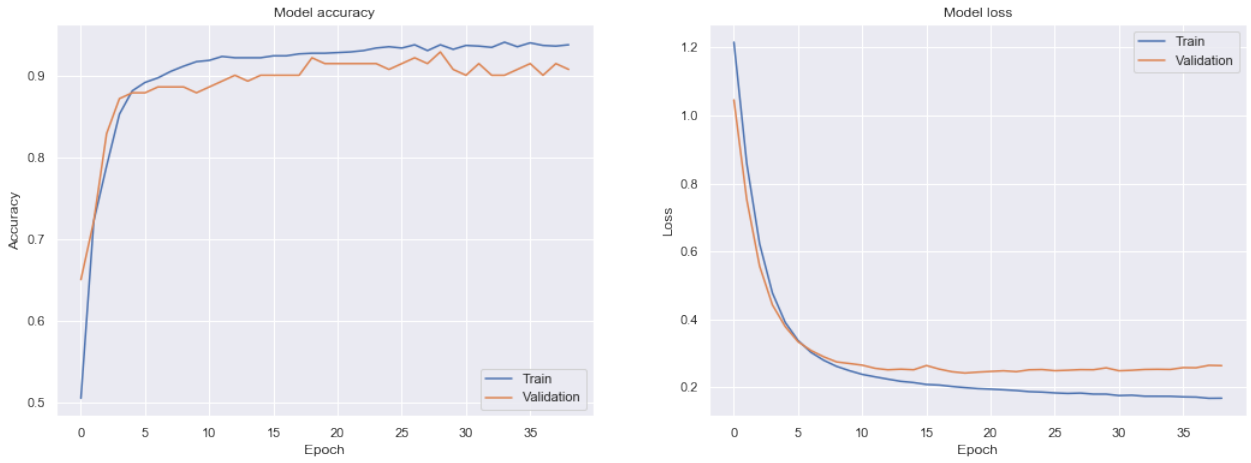


Figure 11: Visualisation of the trained network topological structure with connection strengths highlighted in red for negative valued weights and green for positive values ones. This plot refers to the clean data set.



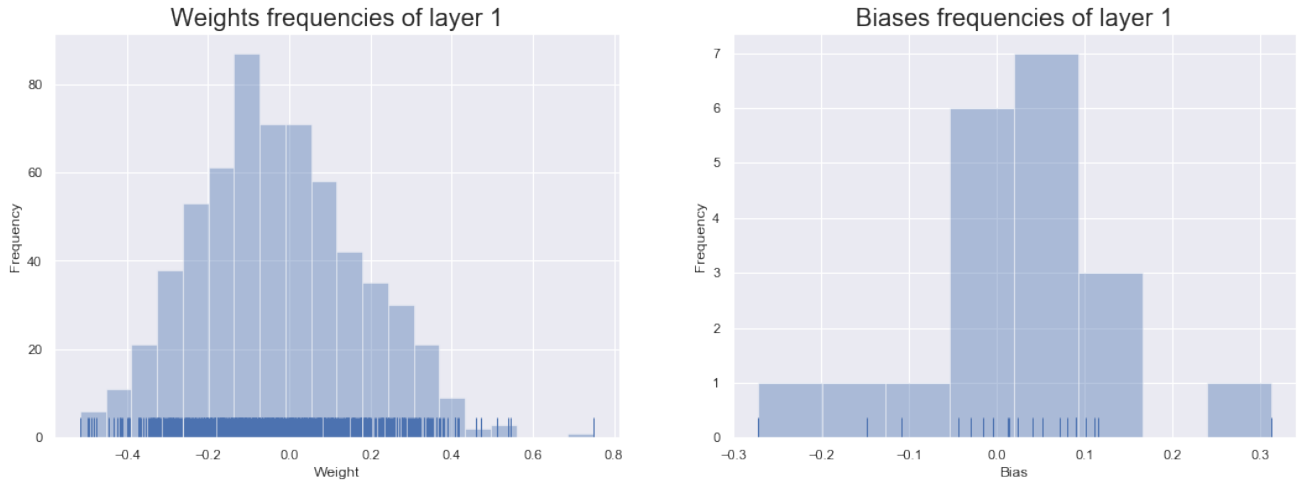
(a) Accuracy **(b)** Loss

Figure 12: Accuracy and Loss profiles, noisy data set.

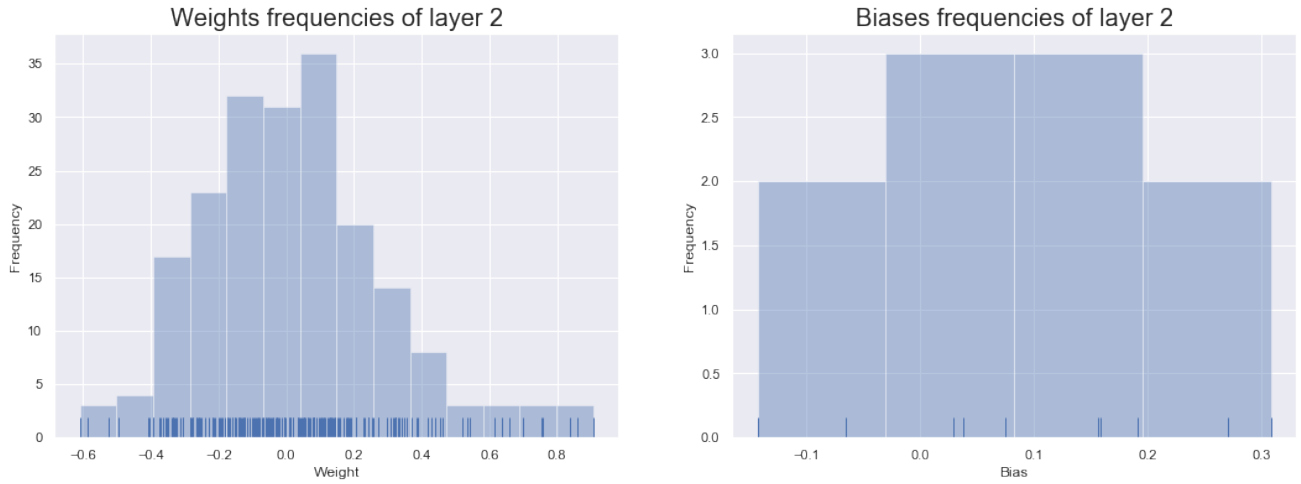
5.2.2 *Noised data set*

As expected, learning process in presence of noisy data is not as fast and accurate as in the former case.

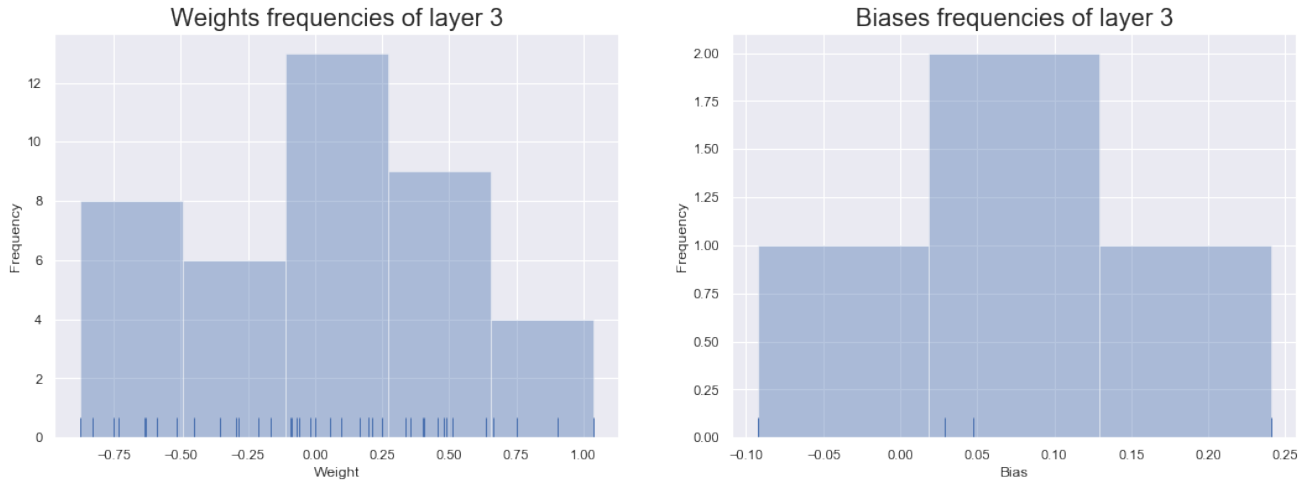
Learning time (epochs)	39 (Early stopped)
Training set accuracy	0.9373
Test set accuracy	0.91667
Test set loss	0.2238



(a) First layer

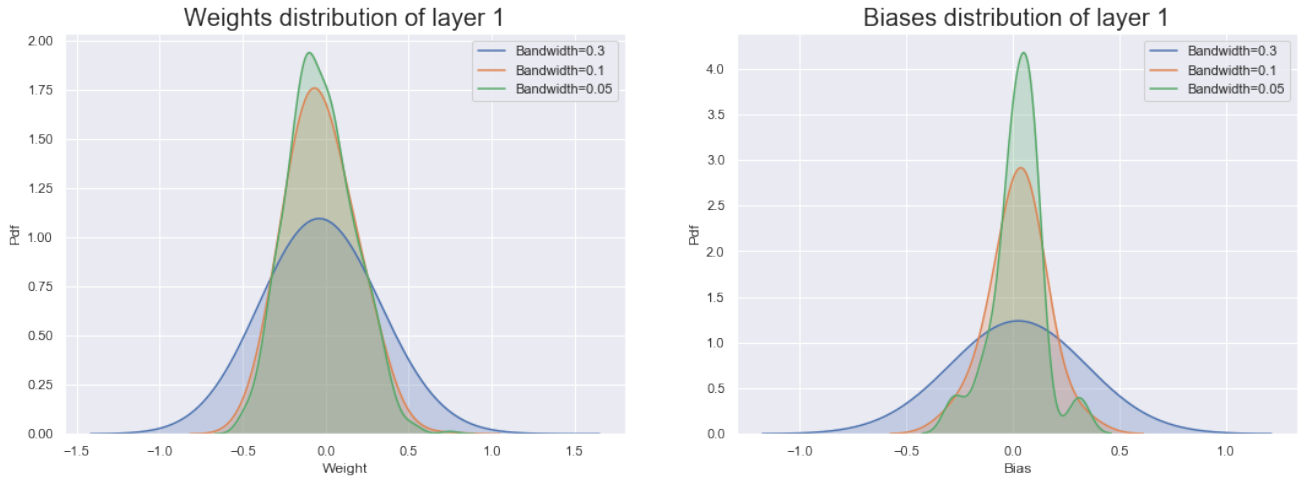


(b) Second layer

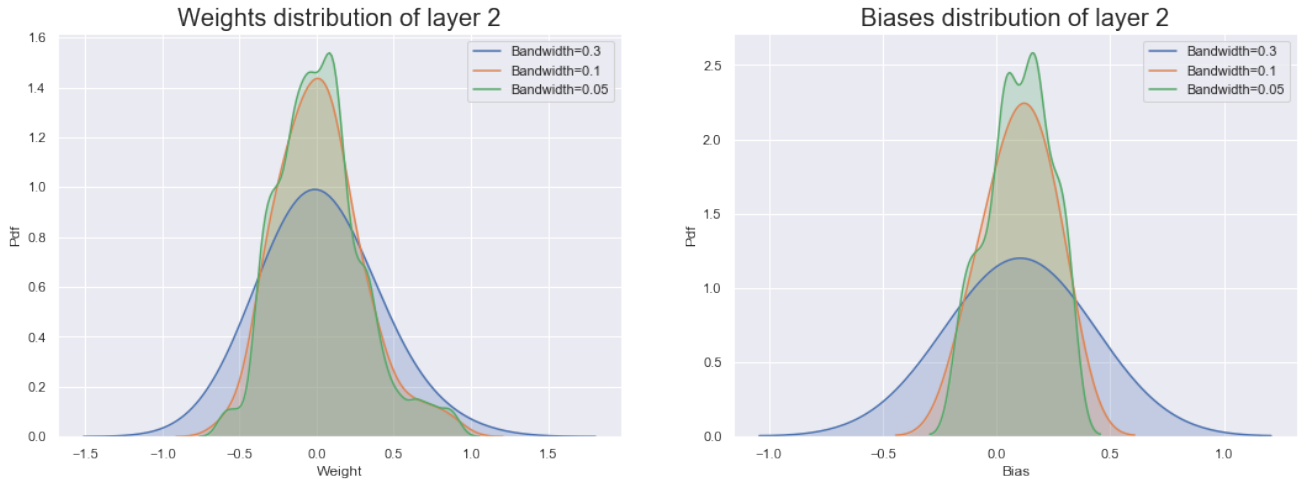


(c) Third layer

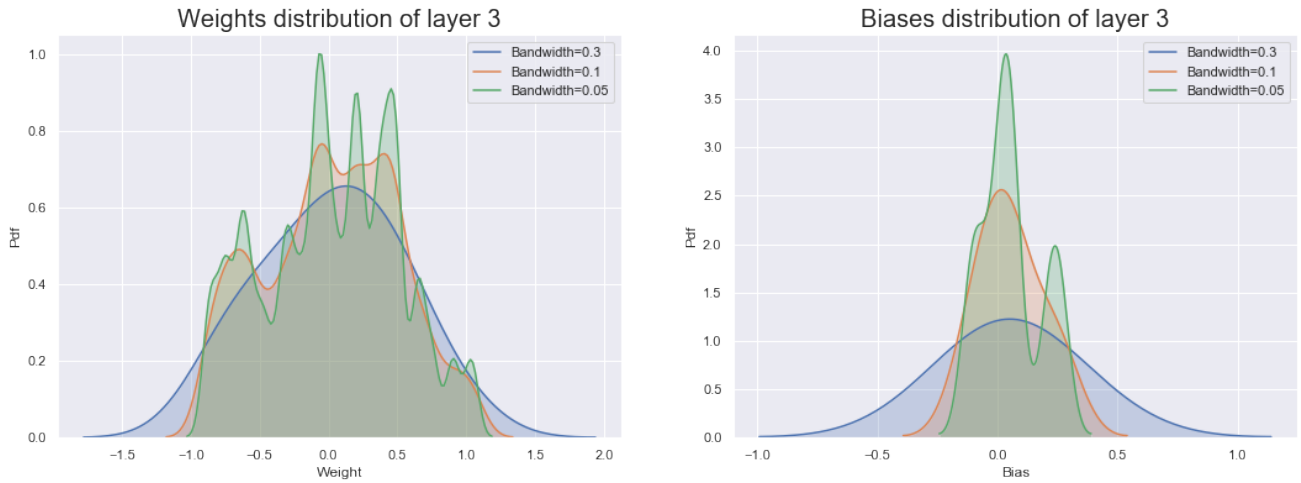
Figure 13: Frequency plots of the parameters once the model is trained on non separable data.



(a) First layer



(b) Second layer



(c) Third layer

Figure 14: KDE of the parameters once the model is trained on noisy data.

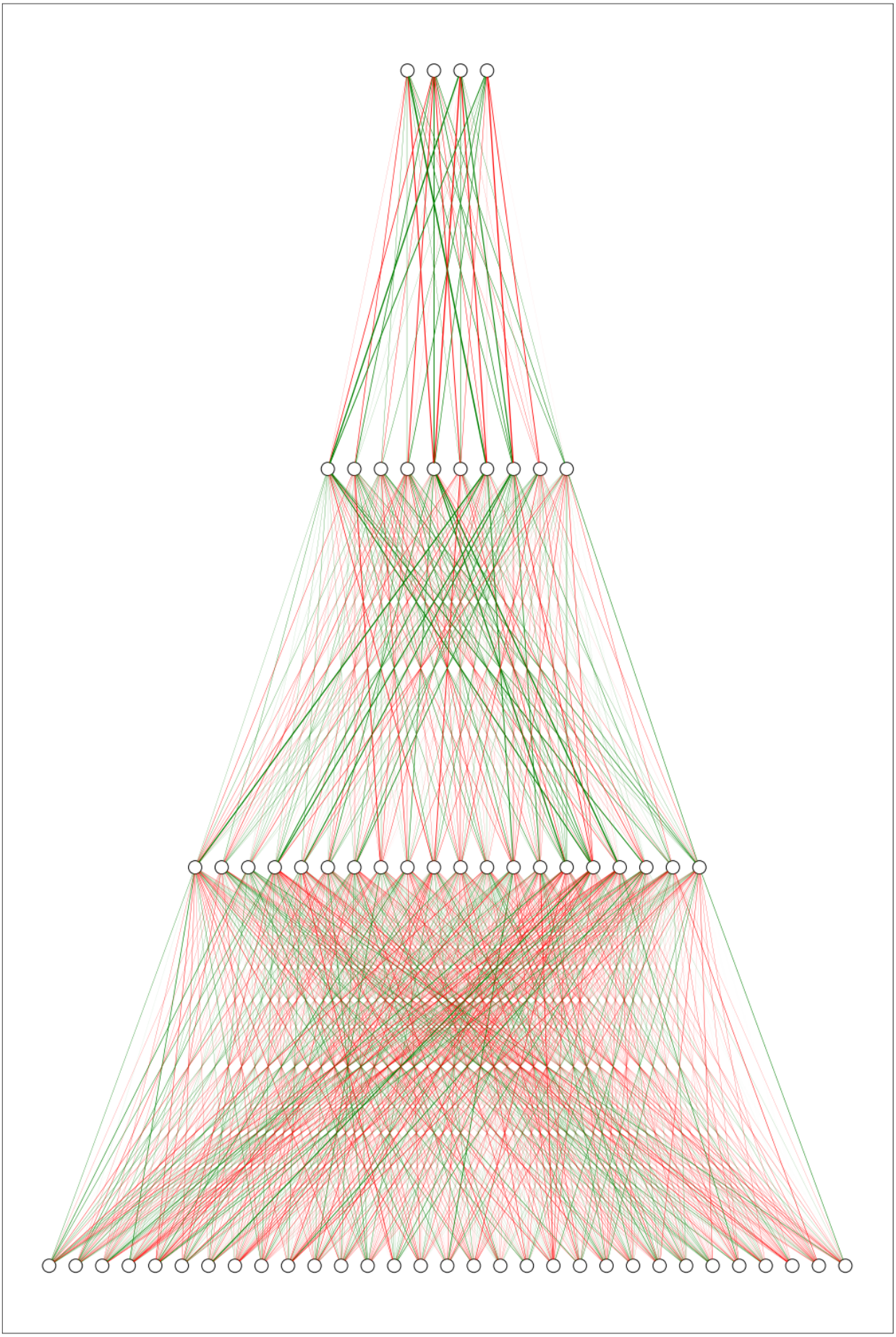


Figure 15: Visualisation of the trained network topological structure with connection strengths highlighted in red for negative valued weights and green for positive values ones. This plot refers to the noisy data set.

6 Conclusions

6.1 Differences between the first version of this report

Situation dramatically changes if the data set is generated in the way here presented. In fact, this allows to dispose of lesser categories and more samples of data not affecting the generation time. Features used (31) is believed to be a good number, but other experiments shows that, even in the case of more features and more categories, learning and performance is not degraded, unless the categories are of the order $\sim O(10^2)$, in this latter case the accuracy on the test set attains the value 0.87.

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 11 and 15 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.

6.2 Further improvements

6.2.1 *Immediate future*

The next steps would be as follows:

- Extrapolate a graph data structure out of the weights, disposable after the calling of `keras.model.get_weights()`, offered by the API Keras, used in the building of the network model.
- Previous works make use of motifs finding algorithms and software. In these, however, networks are not weighted. So, to conform to these, almost preventively, it could be an idea to set the connections weights to 1 if the real weight exceeds a given cut-off threshold, and setting it to 0 (that is, no connection) if on the other hand such a weight is lower than that number.
- Once that done, perform the same motif-search analysis on the resulting (semi-sparse) graph and observe which are the patterns emerging.

However, the landscape of possible motifs is expected to be rather narrow, inasmuch

(i) the architecture of the neural network itself constraints the information to flow one-directionally and

(ii) the neural network as graph is rather simple, since all the connections allowed are established solely between nodes of neighbouring layers.

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?