Effects of Parameter Change on Neural Network and Deep Neural Network

by

Soufia Naseri

Bachelor of Electrical Engineering, Ryerson, 2015

A MRP

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Engineering

in the Program of

Electrical And Computer Engineering

Toronto, Ontario, Canada, 2018

AUTHOR'S DECLARATION FOR ELECTRONIC SUBMISSION OF A MRP

I hereby declare that I am the sole author of this MRP. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this MRP to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this MRP by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my dissertation may be made electronically available to the public.

Effects of Parameter Change on Neural Network and Deep Neural Network

Master of Engineering 2018

Soufia Naseri

Electrical And Computer Engineering

Ryerson University

Abstract

Parameters in neural network such as number of neurons and layers have a direct effect on efficiency and accuracy of the network. On the other hand, it is observed that structure of a network is dependent on data complexity. It is perceived that choosing a complex network for a simple data (e.g. linearly separable data) may not only result in a weak learner but also an increase in computation time. Here, accuracy of the network has been challenged by adding Gaussian Noise to these data to investigate the effect of noise. This additive noise flags overcomplexity thus, decision boundary is generalized which it directs the learner to have better classification as if the number of parameters have been decreased. Throughout these comparisons, other constrains have been minimized in MATLAB to solely observe the effect of each change.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Many researches have been done on Neural network to improve their performances. Introducing new functions and improving the traditional methods are part of these researches. Researching on Neural network topics, it is more apparent how much the structure of a network depends on the data type. To gain a thorough knowledge of this dependency, effect of data type, parameters and hyper-parameters in a NN has been examined. In addition, accuracy of a network with noisy data and clean data has been inspected. There are great optimizations, methods and tests have been done previously though mostly they have been tried with complex networks and not a general one. This may cause additional constrains to our in-depth study. Moreover, by looking at learners' performances and their structures, improving a network can be done more effectively by looking at data types, number of layers and neurons. Effect of noise also has been undermined, since in a real-life examples, data are in fact noisy. In this paper, linearly separable data and non-linearly separable data with and without adding Gaussian Noise have been studied as well as outcome of different number of neurons and layers on performance.

1.1 Background

As technology grows we are lining towards automation in industries and creating methods for machines to decide intelligently. Machine learning comes in to play a key role in intelligent decision-making which unites mathematical modelling and algorithms and ultimately hardware aspects of computers. Over time, machine learning algorithms have been expanded and modified for better and faster results. However, it still divides to two categories, Supervised Learning and Unsupervised Learning. In supervised learning, expected output is given to the algorithm and based on that output the algorithm updates and optimizes itself. However, in unsupervised category there is no known output and learner looks for relation within the given data and therefore, cluster or categorize the data.

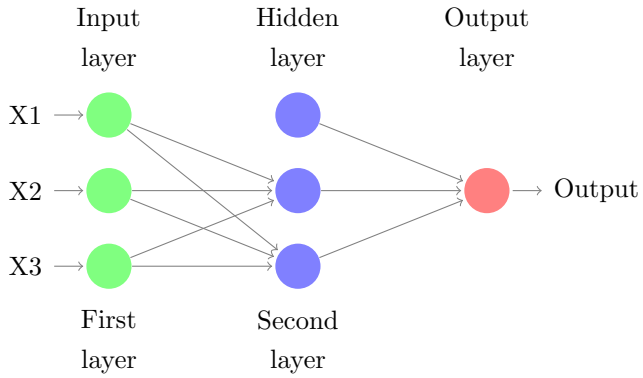
Artificial Neural Network or ANN is mostly used in supervised category though there are useful applications when unsupervised used as well. The ideology of Neural Network is to resemble the way that we humans think and decide. However, creating such a complex network has been a great challenge.

Mathematical modelling is the main idea of ANN where it handles the data processing and learns from the given data to use as examples or experiences. Neural network has different algorithms which contains ANN, Convolutional-NN and Recurrent-NN. Each of these algorithms have their own use of application. For example, CNN is used for images where image segmentation is done first and then sent in to an ANN network. RNNs mostly applicable for time series analysis e.g. speech recognition which is one of the frequent applications of this network.

Choosing a proper structure for these networks are essential. As data types changes e.g. linearly separable or non-linearly separable, structure may vary as well since it affects number of neurons and layers in a network. To make a network more realistic, Gaussian noise has been added to given data to observe the effect on performance. By looking at a general network, ANN which is the core of these mentioned algorithms, a more thorough analysis can be done.

1.1.1 NN Example

In this section, there is an example structure of a two-layer Neural Network. To have a better understanding of how to name layers refer to chapter 2.2.



Neural network consists of two main parts: Forward propagation and backward propagation. In forward pass, inputs get multiplied by weights and give an output. When network is fully connected, each node from previous layer is connected to its next one. X_1 is a bias term where it works as an offset term where it initially assumes to be a value of one. X_2 and X_3 are x and y axis of the examples and as for output if it is one it belongs to one of the dataset and if zero to the the other one. For the sake of simplicity, forward and backward propagation has been evaluated with only one output node.

$$x^{(1)} = \begin{bmatrix} x_1{}^{(1)} & x_2^{(1)} & x_3^{(1)} \end{bmatrix} \tag{1.1}$$

$$w^{(1)} = \begin{bmatrix} w_{10}^{(1)} & w_{11}^{(1)} & w_{12}^{(1)} \\ w_{20}^{(1)} & w_{21}^{(1)} & w_{22}^{(1)} \end{bmatrix} \tag{1.2}$$

$$z^{(2)} = x^{(1)}.w^{(1)T} \tag{1.3}$$

$$x^{(2)} = f(z^{(2)}) = f(x^{(1)}.w^{(1)T}) \tag{1.4}$$

$$x^{(2)} = f\begin{bmatrix} x_1^{(1)}w_{10}^{(1)} + x_2^{(1)}w_{11}^{(1)} + x_3^{(1)}w_{12}^{(1)} & x_1^{(1)}w_{20}^{(1)} + x_2^{(1)}w_{21}^{(1)} + x_3^{(1)}w_{22}^{(1)} \end{bmatrix} \tag{1.5}$$

$$x^{(2)} = \begin{bmatrix} x_1^2 & x_2^2 & x_3^2 \end{bmatrix} \tag{1.6}$$

$$z^{(3)} = x^{(2)}.w^{(2)T} \tag{1.7} \qquad x^{(3)} = f(z^{(3)}) = f(x^{(2)}.w^{(2)T}) \tag{1.8}$$

$$Y = x^{(3)} = f\left[x_1^{(2)}w_{10}^{(2)} + x_2^{(2)}w_{11}^{(2)} + x_3^{(2)}w_{12}^{(2)}\right] \tag{1.9}$$

In backward back propagation, output optimizes itself with respect to expected output where error function is calculated based on a single training set. Then, it updates the weights in a way to decrease Δw and error rate which it forces the NN output to be closer to expected value of that given data set. These steps all are iterative to finding the minimum cost function and it only stops when it reaches the optimal value (any value close to zero).

$$w =: w - \Delta w \tag{1.10}$$ where Δw is: $$\Delta w = \alpha \frac{\partial J}{\partial W} \tag{1.11}$$

There are popular error functions or loss functions available such as Mean Squared error (MSE) and Cross-Entropy. For simplicity, MSE loss function has been used throughout the paper. Cost function shouldn't be confused with error function since it is the average of all loss functions for the entire training set.

$$E^{(3)} = Y_{Expected} - Y \tag{1.12} \qquad E^{(2)} = (w^{(2)})^T E^{(3)} f'(z^{(2)}) \tag{1.13}$$

$$f'(z^{(2)}) = x^{(2)}(1 - x^{(2)}) \tag{1.14} \qquad E^{(2)} = (w^{(2)})^T E^{(3)} f'(z^{(2)}) \tag{1.15}$$

$$E^{(1)} = (w^{1^T})E^{(2)} f'(z^{(1)}) \tag{1.16} \qquad f'(z^{(i)}) = x^{(i)}(1 - x^{(i)}) \tag{1.17}$$

$$\frac{\partial}{\partial w^{(1)}} = x^{(1)} E^{(2)} \tag{1.18} \qquad \frac{\partial}{\partial w^{(2)}} = x^{(2)} E^{(3)} \tag{1.19}$$

$$\frac{\partial}{\partial x_1^{(1)}} = E^{(2)} \tag{1.20} \qquad \frac{\partial}{\partial x_1^{(2)}} = E^{(3)} \tag{1.21}$$

$$x_1 =: x_1 - \Delta x_1 \tag{1.22}$$

1.1.2 Activation Function

Only on hidden neurons, an activation function f is applied where this function has non-linearity properties. Different activation functions may be applied for different layers. These functions are dependent to data, though sigmoid function 1.23, Rectified Linear Unit function (ReLU) 1.24 and Softmax 1.25 are frequently used.

$$f(z) = \frac{1}{1 + e^{-z}} \tag{1.23} \qquad f(z) = max(0, z) \tag{1.24} \qquad f(z) = \frac{e^{z_j}}{\sum_{k=1}^{k} e^{z_k}} \tag{1.25}$$

Softmax function is useful for when presenting the output in terms of probability. ReLU activation function is best for CNNs since all the negative data points become zero and in images pixels have positive value only. Finally, sigmoid function that is among favoured ones is useful the most when output ranges from zero to one. As mentioned, variety of activation functions can be selected for different layers but to minimize the constrains for better observation this function has been kept as sigmoid function

for all the networks 1.1.

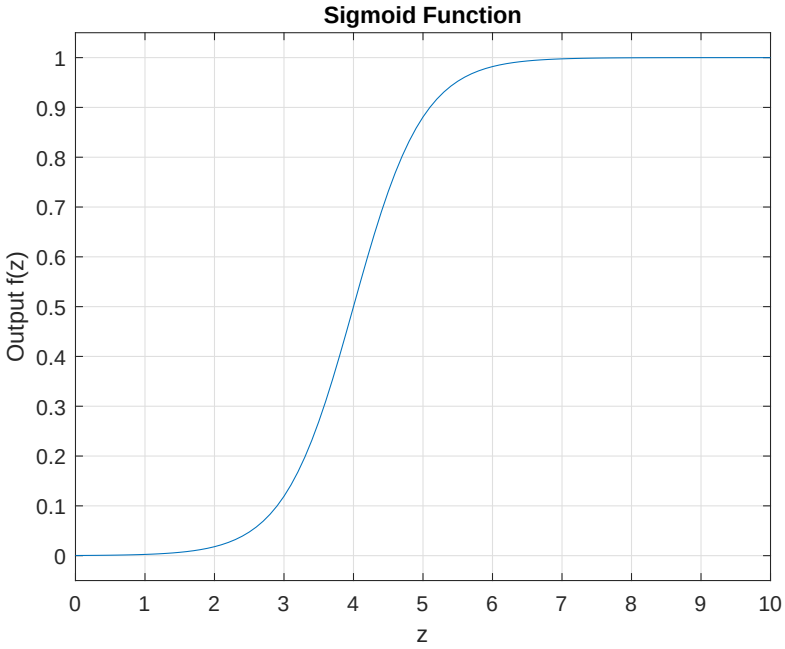


Figure 1.1: Sigmoid Activation function.

Chapter 2

Number of Hidden Neurons and Layers

2.1 Hidden Neurons

Different layers contain different number of neurons. Although choosing number of neurons at input and output layer is not hard, it is very important to be chosen correctly. At input layer, number of categories or features are considered as input nodes e.g. for a two-dimensional input that contains x and y, number of input nodes will be two. Having in mind the bias when creating structure of the NN, number of neurons at the input layer should be increased by one. Here, the network was implemented to classify between two different types of data. This will result in two nodes at output layer where it is called binary classification. When one node is zero the other one is one, meaning only one class is acceptable.

By critically looking at different parameters in NN, a better and more robust neural network may be feasible. Furthermore, different approaches to determine number of hidden neurons are thoroughly investigated. Number of input and output nodes are not difficault to determine since it depends on the selected data. Hidden neurons are deterministic since if the numbers are too few, learner becomes weak and if they are too many their learning process would be too slow [11]. Thus, there is a basic rule of thumb for choosing number of hidden neurons. In [3], few rules are suggested as following:

- Hidden neurons' number should be a value between number of input and output neurons.
- Hidden neurons' number should be $\frac{2}{3}$ of the number of input neurons then added by number of output neurons.
- Hidden neurons' number should be less than twice the size of the input layer.

There are other approaches where actual formula can be used. Existing approaches from 1995 to 2013 papers are analysed and compared in [11] where Mean squared error (MSE), has been used as the error

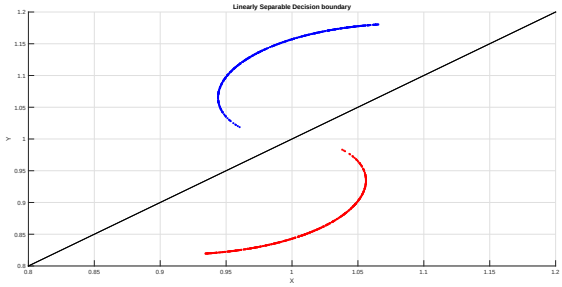


Figure 2.1: Decision boundray of linearly separable data.

function. One of the papers has shown relatively a better result, where it approaches this problem with statistical analysis. This is Sheela and Deepa method where Ni is the number of inputs in the hidden neurons [8].

$$N_h = \frac{4N_i^2 + 3}{N_i^2 - 8} \tag{2.1}$$

This approach was able to reach a better accuracy with a much lower number of iterations and fewer hidden neurons even when compared with rule of thumb. It is worth mentioning that this comparison was done on two different datasets. Unfortunately, both datasets had one output value which may cause a lesser accurate conclusion when working with more than one neuron at the output. However, these methods simply have formulas where it can be tested on the desired network with trial and error to reach the best accuracy.

2.2 Number of Layers

When referring to number of layers there are two terminology used. For instance, when there is no hidden layer, input is fully connected to output with number of weights or a matrix of weights. Since, it only exists one matrix, the network is a one-layer network. Hence for a one hidden layer, there will be two fully connected matrices which then the network will be called a two-layer network. With a one-layer network any linearly separable data is solvable. Once data is non-linearly separable, extra layer is needed. Although, if enough neurons are added even non-linearly separable can be classified with a retentively good error rate, with many linear decision boundaries. This is shown in figure 2.3, where at iteration 1140 the error is 0.06 with computation time of 718.9 second. Number of hidden neurons where 39 using Sheela and Deepa method.

Any network with more than two hidden layers is considered as Deep Neural Network or DNN. When data is simple and linearly separable using more layers than necessary slows down the learner as in table 2.1 and ultimately lower accuracy. Complex datasets follows the same rule; when using one hidden layer with enough number of hidden neurons, process is way slower than with a two layer. In figure 2.3, might

num of H-layers	num of H-neurons	classification error	num of iteration	comp. time(sec)
1	1	0	34	1.907
	3	**0**	**18**	**0.669**
	5	0	21	1.625
Sheela and Deepa	39	0	18	1.545
2	[1,1]	0	46	3.375
	[3,3]	0	20	0.891
	[5,5]	0	27	2.093
Sheela and Deepa	[39,4]	0	17	1.329
3	[1,1,1]	0	70	6.766
	[3,3,3]	0	24	2.218
	[5,5,5]	0	27	2.609
Sheela and Deepa	[39,4,8]	0	19	1.906

Table 2.1: Results for noiseless linearly separable data Ex1.

be able to reach 0 error rate with higher number of neurons but the fact that computation time is too high stays the same since decision boundaries are drawn linearly.

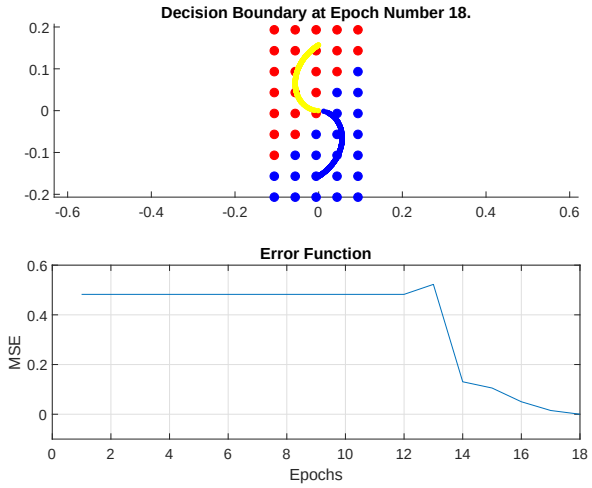


Figure 2.2: Linearly Separable noiseless data with three neurons in a one hidden layer network.

Not using correct number of layers might lead to an event called over-fitting where there is too much data to process and decision boundary loses generalization while classifying. In order to detect over-fitting, error rate on training set is compared with test set. If the error on training set is very low but the error on test set is much higher, then network has lost its general form and thereby over-fitting has happened. There are number of ways suggested to avoid over-fitting so that a designer can build a two or three-layer network for any type of dataset without over-fitting. This topic has been elaborated in

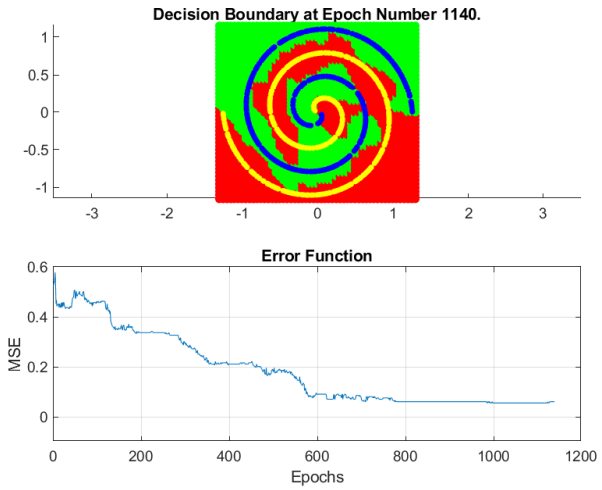


Figure 2.3: Non-linearly Separable noiseless data with one layers.

num of H-layers	num of H-neurons	classification error	num of iteration	comp. time(sec)
1	5	0.246	1860	1426.25
	10	0.035	2340	1798.95
Sheela and Deepa	39	0.06	1140	718.9
2	[5,5]	0	1591	1236.27
	[10,10]	0	1881	1471.83
Sheela and Deepa	**[39,4]**	**0**	**377**	**275.032**
3	[5,5,5]	0	1762	1307.09
	[10,10,10]	0	598	446.078
Sheela and Deepa	[39,4,8]	0	781	690.147

Table 2.2: Results for noiseless non-linearly separable data Ex2.

the next chapter.

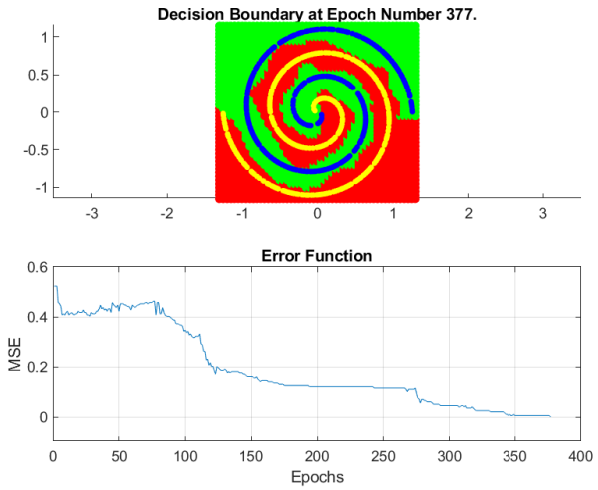


Figure 2.4: Best result for non-linearly separable noiseless data.

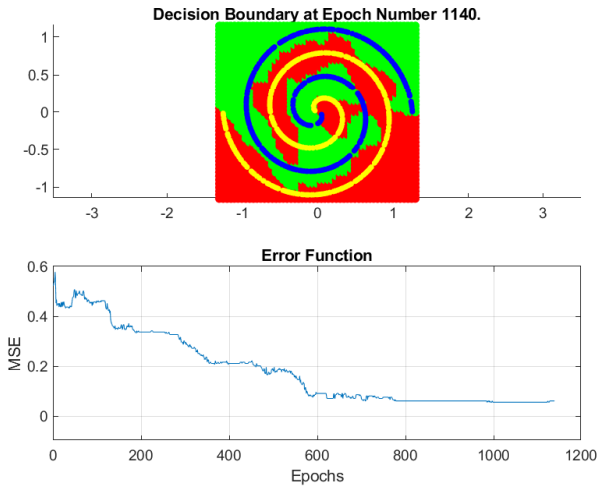


Figure 2.5: Result for non-linearly separable noiseless data for chosen worst case scenario.

Chapter 3

Effect of Noise

ANNs are strong algorithms that are the closest to human power in terms of classifications, but they are not powerful enough to classify correctly when noise in data is high. However, applying noise to network has its own advantages such as generalization and regularization [7] which both helps with robustness. There are many ways to introduce noise to network either multiplicative or additive. In [7], both these methods have been tested. However, this paper is only focused on additive noise. Additive noise can be applied to inputs, weights [1], node activations and outputs [12]. Based on [4], overall result has been improved when applying noise. Though, for an optimal NN, validation set should be noiseless while training set have additive noise and test set is noiseless [4]. Also, it is shown that maximum noise resilience i.g. best solution happens when noise level or standard deviation of training set and test set is the same. Nevertheless, if input is too noisy it decreases the performance rate of noiseless data since learner is also learning noise. Nonetheless, both examples have been tested with noisy validation set since test set never reached zero when validating with clean data. As shown in table 3.1, it has been observed that example 1 (Ex1) test set for both noiseless and noisy data with $\sigma = 0.01$ and $\sigma = 0.1$ reaches zero with less number of iteration than with $\sigma = 0.05$. In addition, as it is shown in 3.1, the higher the level of noise is the slower the learner gets when training. In table 3.2, noisy data in a *worst case scenario* reached zero error much faster comparing to before, showing how noise helped faster convergence.

Table 3.2 reveals how network has been improved with noisy data. This additive noise is most helpful when data has a poor result with clean data. Otherwise, as in table 3.1, additive noise results in additional computation time. However, with a more complex example (Ex2) additional noise on a two hidden layer network with $\sigma = 0.01$ never reached to its previous error (zero value).

	$N(0, \sigma = 0.01)$	$N(0, \sigma = 0.05)$	$N(0, \sigma = 0.1)$
Error rate	0	0	0
Number of iteration	20	26	24
Comp. time(sec)	0.764	1.861	1.873

Table 3.1: Results for noisy data Ex1 for best case scenario.

	$N(0, \sigma = 0.01)$	$N(0, \sigma = 0.05)$	$N(0, \sigma = 0.1)$
Error rate	**0**	0	0
Number of iteration	**49**	45	53
Comp. time(sec)	**4.372**	4.521	5.152

Table 3.2: Results for noisy data Ex1 for worst case scenario.
refer to table 2.1: three layer network [1 1 1]

	$N(0, \sigma = 0.01)$	$N(0, \sigma = 0.05)$	$N(0, \sigma = 0.1)$
Error rate	0.01	0	0.005
Number of iteration	820	540	600
Comp. time(sec)	597.623	404.456	453.213

Table 3.3: Results for noisy data Ex2 for best case scenario.

All things considered, when data is over-fitted e.g. higher order or layers are used, noisy data points assist the classifier with generalization which ultimately causes noise to avoid over-fitting.

The following are another ways to over come over-fitting either due to noise or network structure.

1. Cross-validation: choosing training test, validation test and test set can be done randomly but in order to have a better and more accurate result, there are methods available. First method is dividing the entire dataset to three sets with different proportion. For instance, training test can be 60% of the entire dataset and the rest can split between validation test and test set. Other method is called k-fold cross-validation (CV). K-fold CV is partitioning the data into k subsets where (k-1) of subsets are going to be the training set and one subset for test set. This folding method is done k times on the data where average of the results is taken at the end [14]. Averaging may increase run time or decreases the training time.

2. Data Augmentation or Replication: over-fitting may occur if there exist too many weights. Adding more data to the training set can aid this issue since network has more data to process with the same number of weights. In case of access limitation to data, data can be replicated or augmented.

3. Early stopping: it is done when there is lots of data to be processed ergo higher computation time. By stopping the training process earlier over-fitting can be prevented [13].

	$N(0, \sigma = 0.01)$	$N(0, \sigma = 0.05)$	$N(0, \sigma = 0.1)$
Error rate	**0.01**	0.055	0.05
Number of iteration	**793**	1179	982
Comp. time(sec)	**603.566**	890.086	720.46

Table 3.4: Results for noisy data Ex2 for the chosen worst case scenario.
refer to table 2.2 one layer network 39

	$N(0, \sigma = 0.01)$
Error rate	0
Number of iteration	344
Comp. time(sec)	298.021

Table 3.5: Results for noisy data Ex2 for the chosen worst case scenario.
refer to table 2.2 three hidden layer network with Sheela and Deepa method

4. Regularization

   (a) L1 and L2 regularization: Regularization keeps the weights small doing so for L1 and L2 regularization, a penalty function gets added to squared error. This penalty function smooths down the network with modifying the updated weights. L2 regularizer is also called weight decay (3.2) where the function is sum of squared weights.[2]. To avoid local minima, weight decay and regularization has been found helpful [7]. L1 and L2 regularizations are as follow:

$$J = \sum E + \sum \lambda \left| w_i \right| \tag{3.1}$$

$$J = \sum E + 2\lambda \sum w_i^2 \tag{3.2}$$

   When there is no regularization added to the network, cost function and error function are the same. Therefore, their partial derivatives w.r.t weights are also equal 3.3.

$$\frac{\partial J}{\partial w_i} = \frac{\partial E}{\partial w_i} \tag{3.3}$$

   L2 regularization derivation:

$$\frac{\partial J}{\partial w_i} = \frac{\partial E}{\partial w_i} + \lambda w_i \tag{3.4}$$

   When reaching minima, meaning $\frac{\partial J}{\partial w_i} = 0$, it is apparent that weights are kept small and only change to higher values when derivation of error is large .

$$\frac{\partial E}{\partial w_i} = -\lambda w_i \tag{3.5}$$

$$w_i = -\frac{1}{\lambda} \frac{\partial E}{\partial w_i} \tag{3.6}$$

   (b) Dropout: giving a probability to each neuron to receive a value of zero for its weight. This is only done on hidden layers where chosen probability is a hyper-parameter. In practice, usually probability of 0.5 is considered [9].

On the other hand, it is intresting to mention that there have been researches done where noise is carefully crafted to attack a neural network classifier. However, this paper shows that by only changing one pixel to a faulty one, learner can be fooled. Adversarial images are those images that are created to fool the neural net to misclassify. In [10], these images are very clear to human eye and almost impossible to misclassify since only one pixel is changed, though to machines are not so obvious. In order to have a successful attack on the classifier, choice and colour of that one pixel matter. Familiarizing with different

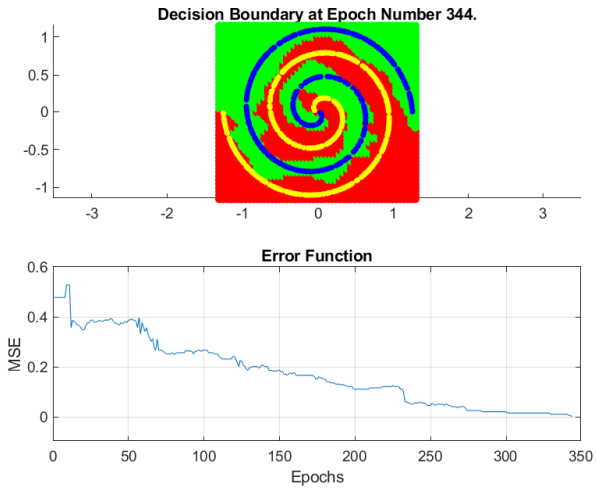


Figure 3.1: Result for noisy Ex2 data for the chosen worst case scenario.
refer to table 3.5

types of attacks, helps the designer to create a robust neural network.

In paper [5], performance of twelve loss functions have been tested with and without noise. In case of non-linearly separable noiseless data $hinge^2$ loss or squared margin loss has been the best for different numbers of layers. Mean squared method or *L2* [5], has shown relatively good results as well.

$$hinge^2 = \sum max(0, \frac{1}{2} - YY_{Expected})^2 \tag{3.7}$$

Thus, effect of noise by choosing different error functions can be handled even better. In [5], is also observed that training accuracy stays the same with σ up to 0.2 for all layers. However, when there exist a high level noise, accuracy stars to fall down since learner is also learning noise as a part of to-be-trained data. In addition, using optimization on gradient improves back propagation and thereby a better learner. Adam optimization or Adaptive moment estimation calculates the first moment (mean), second moment (variance) of the gradients and updates the weight according to those values. This optimization method is one of the newest proposed method where it makes gradient to converge faster while not overshooting the minima [6].

Chapter 4

Conclusion

As it has been shown number of parameters have a direct relationship with results of NN. In case of linearly separable data only one layer is enough and if higher order is chosen, there is an increase in computation time a learner with the same error rate. The same holds true for case of non-linearly separable data both for number of neurons and layers. Over-fitting happens when higher number of layers and neurons are chosen much higher might than necessary for. In order to avoid over-fitting, different methods as well as adding noise were explained.

All in all, adding noise to linearly separable data with only one hidden layer has not been helpful but when implementing with higher order performance is increased. In noisy Ex1, a three hidden layer network performed as good as a two hidden layer network with noiseless data. The same conclusion was observed for non-linearly separable data however, after observing the outcome of a two hidden layer classifier with noise interesting conclusion is drawn.

When classifying with exactly required number of layers, not only additive noise does not improve the process but also slows down the learner. However, when using more hidden layers than necessary with existence of noise, performance increases drastically. Thus, noise flags overcomplexity or over-fitting which directs the learner to have better classification. Noisy data points overcomes the fact that only few data points are fitted in the decision boundary and thereby generalization is introduced.

As in [4], optimal NN has been set to use noiseless dataset for validation but doing so with noisy test set makes the test set to reach zero slower or in few cases not reach zero at all. Since the best solution for NN is to be trained on noisy training and test set, validation set has been also chosen to be noisy where it actually helps to validate over-complexity of the network.

References

[1] Adel M Abunawass and Charles B Owen. Statistical analysis of the effect of noise injection during neural network training. In *Science of Artificial Neural Networks II*, volume 1966, pages 362–372. International Society for Optics and Photonics, 1993.

[2] Stephen José Hanson and Lorien Y Pratt. Comparing biases for minimal network construction with back-propagation. In *Advances in neural information processing systems*, pages 177–185, 1989.

[3] J Heaton. Introduction to neural network for java. heaton research. *Inc, St. Louis*, 2005.

[4] IV Isaev and SA Dolenko. Training with noise as a method to increase noise resilience of neural network solution of inverse problems. *Optical Memory and Neural Networks*, 25(3):142–148, 2016.

[5] Katarzyna Janocha and Wojciech Marian Czarnecki. On loss functions for deep neural networks in classification. *arXiv preprint arXiv:1702.05659*, 2017.

[6] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[7] Yinan Li and Fang Liu. Whiteout: Gaussian adaptive noise regularization in feedforward neural networks. *arXiv preprint arXiv:1612.01490*, 2016.

[8] K Gnana Sheela and Subramaniam N Deepa. Review on methods to fix number of hidden neurons in neural networks. *Mathematical Problems in Engineering*, 2013, 2013.

[9] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[10] Jiawei Su, Danilo Vasconcellos Vargas, and Sakurai Kouichi. One pixel attack for fooling deep neural networks. *arXiv preprint arXiv:1710.08864*, 2017.

[11] Tijana Vujicic, Tripo Matijevic, Jelena Ljucovic, Adis Balota, and Zoran Sevarac. Comparative analysis of methods for determining number of hidden neurons in artificial neural network. In *Central European Conference on Information and Intelligent Systems*, page 219. Faculty of Organization and Informatics Varazdin, 2016.

[12] Chuan Wang and Jose C Principe. Training neural networks with additive noise in the desired signal. *IEEE Transactions on Neural Networks*, 10(6):1511–1517, 1999.

[13] Tong Zhang, Bin Yu, et al. Boosting with early stopping: Convergence and consistency. *The Annals of Statistics*, 33(4):1538–1579, 2005.

[14] Yongli Zhang and Yuhong Yang. Cross-validation for selecting a model selection procedure. *Journal of Econometrics*, 187(1):95–112, 2015.