A Residual Sampling Neural Network

*Note: Implementation can be found on GitHub

Ren Yuan Xue
Computational Mathematics
Univeristy of Waterloo
Waterloo, Ontario

Pierre McWhannel
Computational Mathematics
Univeristy of Waterloo
Waterloo, Ontario

***Abstract*—Avoidance of overfitting for machine learning algorithms has proven to be a paradigm problem over time. There have already been many approaches to this in the context of neural networks. We now propose a novel neural network to address this problem. It uses distributions to model each weight and bias, creating residuals as well. This network will sample from these distributions to make predictions during the feed forward process. The network learns by updating the parameters that are prescribed to each weight and bias, while using the residuals between outputs and targets. The result is a network which can regularize and be more robust than a classical feed forward network, this provides practitioners with another way to combat overfitting. The network also offers a better interpretation of the confidence of its prediction.**

I. Introduction

Neural networks have become one of the most popular machine learning tools for a vast array of predictive applications. One of the most popularized problems with neural networks is overfitting [1], this is where the motivation for our new network arises. Overfitting can be seen as the model or algorithm fitting to the training data as well as the noise in data, this is to say it doesn't generalize well to new unseen data. This can arise when a neural network has been over trained on data, the result of continuously being optimized on the training data. In this case, the model begins to trace the irreducible error present in the data. Our method is a regularization technique that ensures the model generalizes better as well as offers better interpretability.

The neural network (NN) we propose is adapted from a simple feed forward network [2]. Our NN differs by having the weights and biases in each layer represented by a prescribed distribution. The process of learning then differs by needing to learn the parameters of the distributions for the weights and biases rather than point estimates. This requires an MLE for the prescribed distribution parameters for each weight and bias.

Our NN uses these distributions or uncertainty in modelling the weights and biases for the purpose of generalizing better. This leads to a more robust network, promising results have been shown before such as "Weight Uncertainty in Neural Networks" [3]. Our approach differs in that we try to stick more closely with a traditional learning procedure of a neural network. Thereby, we stick with cost functions which are more popularly used in neural networks. These cost functions include but are not restricted to mean squared error, cross entropy, and categorical cross entropy.

Using distributions on weights and biases forces the model to find solutions which are not overly sensitive to perturbations thus being more robust. The distributions can be thought of as small perturbations of the weights and biases. The idea here is that a small perturbation should not dramatically change your prediction yet when models overfit they tend to show this sensitive behaviour [4] and can even fall victim to adversarial attacks [5].

The second novelty introduced in this new network is a threshold parameter. This parameter is used to form a residual between the target and prediction of the algorithm. More formally, we define the residuals as the subset of our outputs that are not within the range of target value $\pm$ threshold as follows:

$$|y - t| \leq threshold : Residuals \tag{1}$$

Where

$$y : model\ prediction,\ t : targets$$

More specifically, in a binary classification problem, we have:

$$y \in [0, 1]\ and\ t \in \{0, 1\}$$

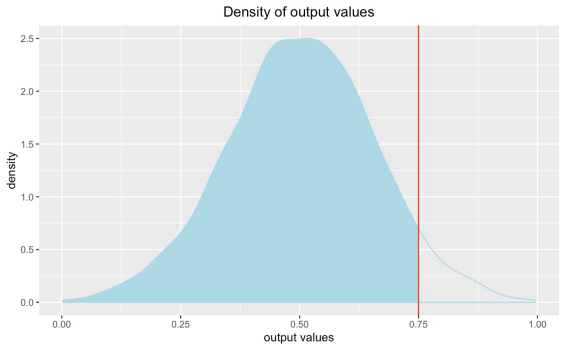

Fig. 1. 0.25 threshold, when t = 1 the shaded area is the residual

The report consists of three sections to follow, methodology, results, and discussion. Methodology is where the algorithm will be presented in detail along with the data sets, and testing

procedures. The results section will display our results from our tests. The discussion contains our analysis and findings, following the results is the conclusion.

II. Methodology

A. Model Description

The new model, Residual Sampling Neural Network (RSNN), is similar to a traditional neural network however has two distinct differences. The first is the use of distributions to model weights and biases. The second is the use of a threshold to form residuals. Below are the three significant steps of the learning for an RSNN.

1) Initialization: Along with the number of neurons in each layer of the network, the learning rate, and epochs, the distributions need to be prescribed for each weight and bias. In this report Gaussian's were solely tested as the prescribed distribution, however, other distributions could be used. A Monte Carlo method will be employed on these distributions when running the feedforward, therefore it is helpful if the distributions have an analytic form. The parameters of Gaussian distributions have been initialized as follows:

$$Weights \sim \mathcal{N}(0,1) \tag{2}$$

$$Biases \sim \mathcal{N}(0,0) \tag{3}$$

2) Feed Forward: The feed forward rather than being deterministic given a set of learnt parameters is now stochastic. This is the case since the weights and biases that are used in a feed forward pass are now modelled by distributions. Each weight and bias is then sampled from the distribution for a fixed amount of times and ultimately completing a fixed amount of feed forwards and thereby predictions for a single observation.

3) Back Propagation: The back propagation now changes from a traditional neural network by having a posterior residual formed with respect to the targets and threshold. This posterior comes from the many resamplings of the weights and bias for a given observation. The learning process which back propagation uses, gradient descent, has to be done by sampling from this residual distribution. Then for each sample taken from the residual, updates for all weights and biases are stored. This process creates a collection of updates for distribution parameters of each weight and bias.

The second important difference to note is the threshold's effect on the gradient $\frac{\partial E}{\partial y}$, here E is the loss function and y is the models output. The threshold forces $\frac{\partial E}{\partial y} = 0$, for any y that is within the threshold of the target as shown in figure 1. This forces a process similar to gradient boosting [6], in which the model only builds on errors which were sufficiently large. For example, in a binary classification the selection of a threshold of 0.5 indicates only predictions which are misclassified will be used to update the network. This differs from a normal network update as a normal network uses any error short of the target to update the network.

4) Parameter Estimation: The weights and biases are initialized as a prescribed distribution, currently Gaussian's are used; however, other distributions could be used. The collection of updates for each weight and bias from the back propagation is used to estimate the parameters of the prescribed distribution. This update is performed using a MLE for the mean parameter.

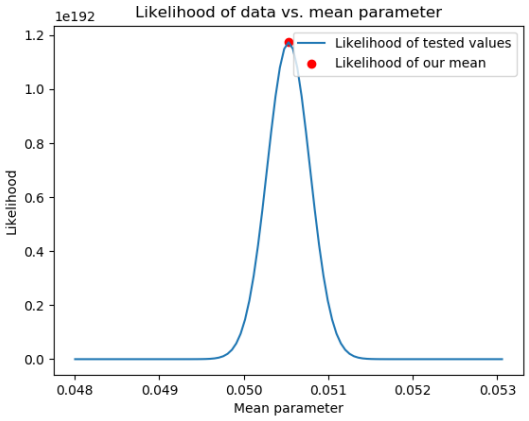

Fig. 2. Example likelihood of our estimate for a weight and neighbouring region of possible mean values

The variance is set in a fashion that keeps the ratio of the mean and standard deviation constant. Thereby, estimating the mean gives the variance as seen in the following equation:

$$\sigma^2 = (c \cdot |\mu|)^2 \tag{4}$$

$$Variance : \sigma^2$$

$$Mean : \mu$$

Here c is a hyper parameter used to select a constant ratio between the mean and standard deviation.
Note: Originally we used sample mean and variance to update distribution mean and variance. However, from tests we did on several data sets, the variance always goes down to zero as number of epochs increases. This is a problem when variance is ignorably small since the predictions we made will be essentially a point estimate. In order to overcome this problem, we decided to update the variance in terms of mean.

B. Algorithm

1) Initialize the parameters of the priors.
2) Sample the weights and biases while performing feed forward.
3) Collect all outputs and identify the residuals.
4) Select samples from residuals and apply backpropagation.
5) Collect a distribution of updates for each weight and bias.
6) Estimate the mean of weights and biases using sample mean and replace old mean.
7) Compute the variance and replace old variance.

8) Repeat steps 2 to 7 for next epoch.

This algorithm causes the solution to differ from a traditional network since as a weight's mean increases the variance will also increase. Forcing this relationship encourages a weight decay like effect. Since a larger variance means the algorithm will sample values which more significantly differ from the mean. This results in the algorithm learning to increase weights only when the increase in variance doesn't cause significantly more wrong predictions when sampling.

C. Loss Functions

This section outlines the various loss functions built into our model and thereby package. All loss functions are desired to be minimized.

1) Binary Cross Entropy/Bernoulli:

$$\frac{-1}{N}\sum_{i=1}^{N} t_i log(y_i) + (1-t_i) log(1-y_i) \tag{5}$$

This is used for classification tasks where there are only 2 exclusive classes.

2) Categorical Cross Entropy:

$$\frac{-1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} t_{i,k} log(y_{i,k}) \tag{6}$$

This is used for classification tasks where there are more than 2 classes and they are exclusive.

3) Mean Squared Error:

$$\frac{1}{N}\sum_{i=1}^{N}(t_i - y_i)^2 \tag{7}$$

This can be used for regression tasks with our model.

D. Data Sets

1) Simulated: The simulated data set was created using the following structure:

$$y = \begin{cases} 1, & \text{if}, a^T x + \eta \geq b \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

$$Label : y \in \{0, 1\}$$

$$Covariates : x \in \mathbb{R}^p$$

$$Coefficients : a \in \mathbb{R}^p$$

$$Bias : b \in \mathbb{R}$$

$$Noise : \eta \sim \mathcal{N}(\mu, \sigma^2) \tag{9}$$

Figure 3 follows which is a plot with $p = 2$, where p is the number of covariates.

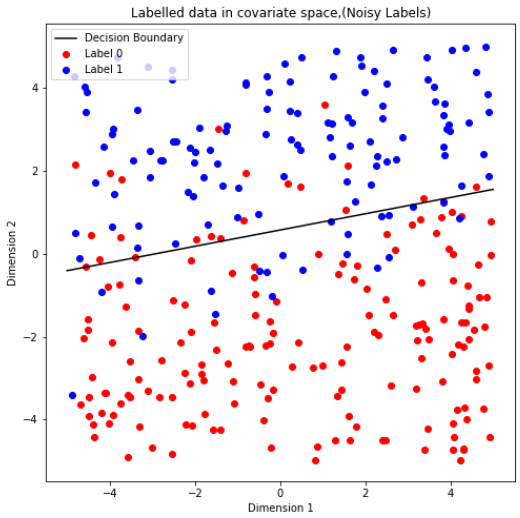

Fig. 3. Plot of 300 simulated points in 2D space

The simulated data is created using a linear boundary as seen in fig 3 and explained in detail above. Having a true linear boundary and 2 classes makes this a good starting point for testing our model.

Loss Function: Bernoulli, (5).

2) Iris: The Iris dataset is multivariate, having 4 covariates. This dataset consists of 150 observations with 3 distinct classes. Each classes has 50 observations forming the total and thereby a balanced dataset. These figures in minds make it a good toy dataset to work with and test our model against. During testing involving the iris dataset the training and test accuracies will be formed using cross validation with K folds, due to only having 150 observations.

Loss Function: Categorical Cross Entropy, (6).

3) MNIST (Hand Written Digits): The MNIST dataset is a multivariate dataset that has 28x28 pixel images of hand written digits. These digits have 10 classes corresponding to 0 through 9. This dataset is popularly used for bench marking new algorithms and models, making it an ideal candidate for our model to be bench marked upon.

Loss Function: Categorical Cross Entropy, (7).

E. Testing Procedures

This section outlines the tests which were performed and presented in this next section. The rationale and procedure of each test is outlined within this section. The tests listed here were performed on all three data sets though only select ones have been chosen to be presented in the results section. Additionally the model was only run using Gaussian priors and with 3 layers.

1) Benchmark Accuracy Test: This test is to benchmark our model versus a normal neural network and one with weight decay. During this test accuracy will be recorded after each epoch for all three models. The results will be presented in

a graph with accuracy vs epochs. The accuracies on training and test sets is presented.

2) Benchmark Cost/Loss Test: This test is to show the convergence of our algorithm compared with a normal neural network and one with weight decay. This offers insight into how the speed of convergence is reached compared to the other models. In the following plots, the test set for the simulated data is noiseless to more easily identify overfitting.

3) Parameter Tests: These tests are used to show the characteristics of our model given different settings for the parameters. This sets a foundation for what is and isn't a hyper parameter and how each affects the models performance. This test is done by comparing the training and test accuracy of the model with various settings for a given parameter with while others are fixed. All plots presented under these tests will be run on the simulated dataset, for ease of experiment controls.

III. Results

A. Benchmark Accuracy Tests

1) Simulated Data: Selected parameters for the following plots are listed below.

- Covariates = 10
- Hidden nodes = 10
- Train size = 200
- Test size = 200
- Learning rate = 1
- MC Samples = 200
- Threshold = 0.3
- Coefficient c = 0.25
- Weight Decay = 0.0025

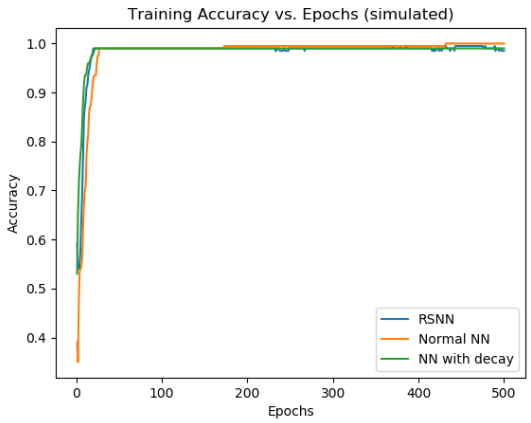

Fig. 4. Training accuracy vs Epochs

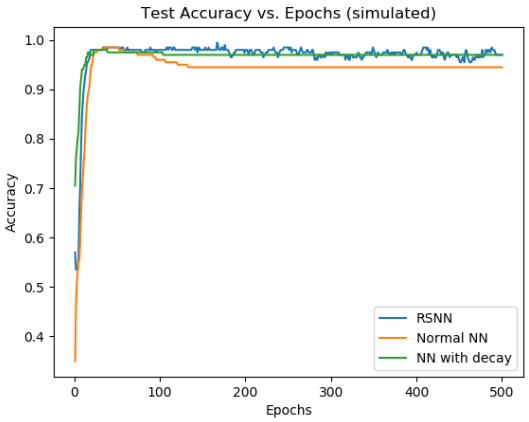

Fig. 5. Testing accuracy vs Epochs

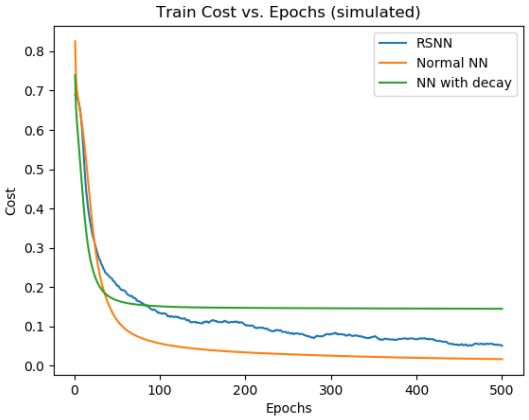

Fig. 6. Training cost vs Epochs

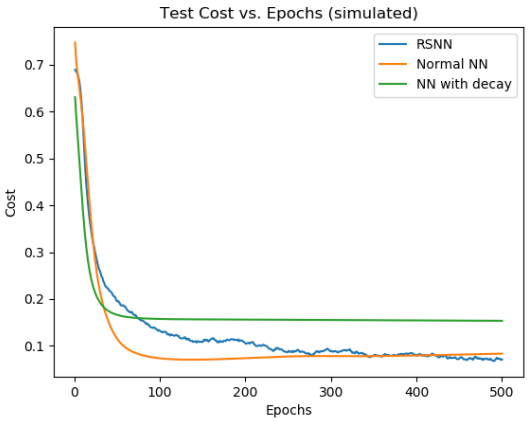

Fig. 7. Testing cost vs Epochs

2) Iris Data: Selected parameters for the following plots are listed below.

- Hidden nodes = 10

- Train size = 100
- Test size = 50
- Learning rate = 0.2
- MC Samples = 200
- Threshold = 0
- Coefficient, c = 0.05
- Weight Decay = 0.01

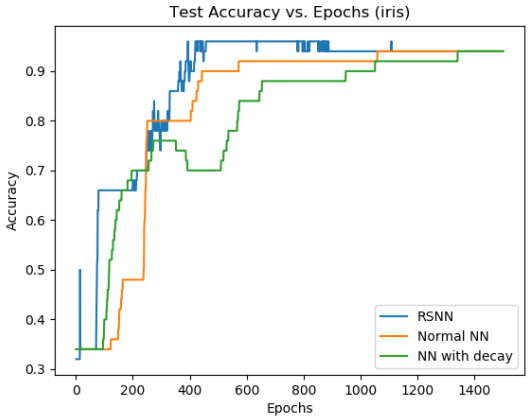

Fig. 8. Testing accuracy vs Epochs

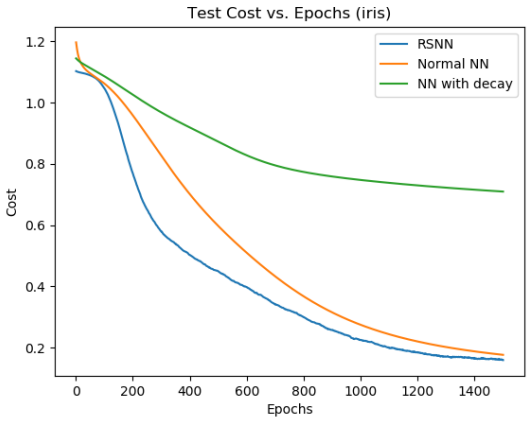

Fig. 9. Testing cost vs Epochs

3) MNIST Data: Selected parameters for the following plots are listed below.

- Hidden nodes = 100
- Train size = 600
- Test size = 200
- Learning rate = 1
- MC Samples = 200
- Threshold = 0
- Coefficient, c = 0.05
- Weight Decay = 0.005

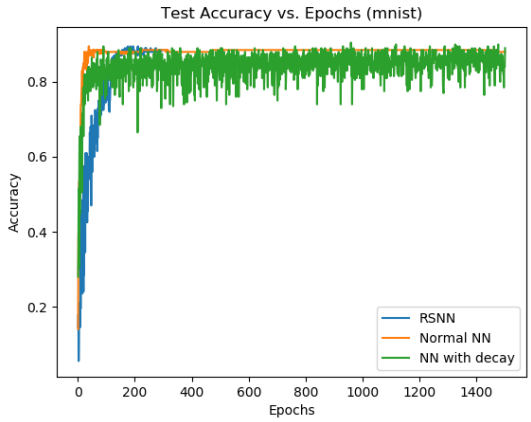

Fig. 10. Testing cost vs Epochs

B. *Parameter Tests*

Parameters will be set to the following default values below, while tested parameters will be varied as indicated by the plots. These tests are all done with the simulated data.

- Covariates = 5
- Hidden nodes = 5
- Train size = 200
- Test size = 200
- Learning rate = 1
- MC Samples = 100
- Threshold = 0
- Coefficient, c = 0.05

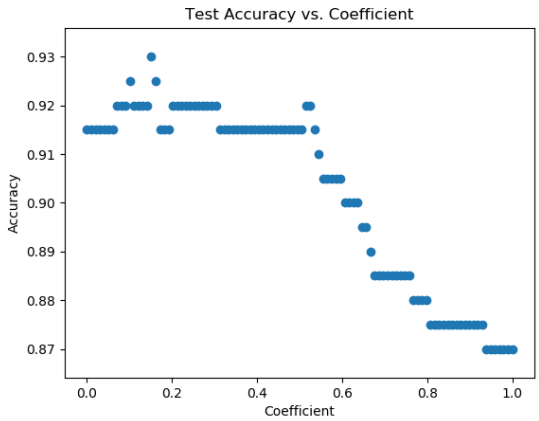

Fig. 11. Test Accuracy vs Coefficient

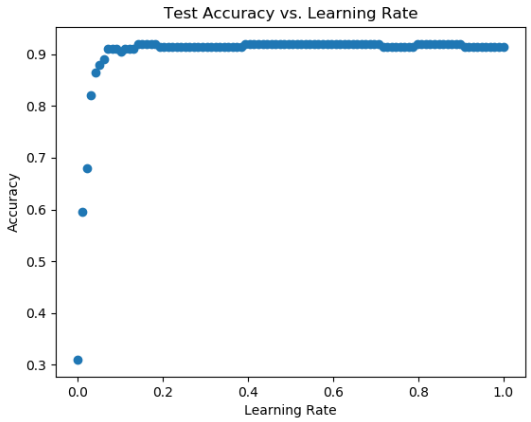

Fig. 12. Test Accuracy vs Learning Rate

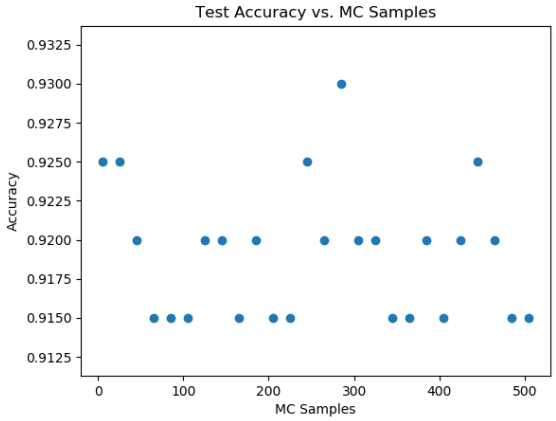

Fig. 13. Test Accuracy vs Monte Carlo Samples

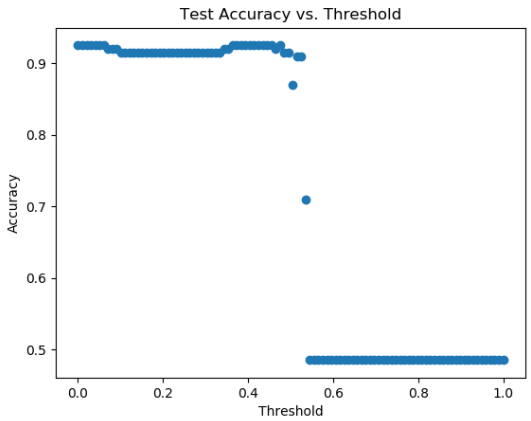

Fig. 14. Test Accuracy vs Threshold

IV. Discussion

A. Findings, Strengths and Limitations

1) Benchmark tests: As can be observed in figures 4, 5 the normal neural network generalized marginally worse than the RSNN and NN with decay. The over fitting can be seen by contrasting figures 6 and 7, here we see the normal neural network overfit when it's cost starts increasing after about 150 epochs. Where as in the RSNN, the general trend is the cost decrementing. These results suggest that the RSNN succeeded in generalizing better than the normal neural network.

The figures 8 and 9 suggest that the performances of all networks are similar with the RSNN performing slightly very better in terms of the cost function. This could be reflective of the dataset or insufficient search of the parameter spaces. However, it does provide evidence of the algorithm performing similarly to another regularization method, weight decay.

The results from the MNIST dataset were similar to those from the iris dataset as can be seen in figure 10. Excluding the varying jumps the NN with decay is having, the accuracies all of 3 are very close. It is interesting to note that although we propose RSNN combats overfitting, it is not jumping around like the NN with decay.

2) Parameter Tests: Figure 11 illustrates an interesting relationship of the accuracy and coefficient. We see here that a coefficients within the interval [0,0.5] perform very similarly than the accuracy linearly drops down as the coefficient increases form 0.5 to 1. This suggests the variance can be significantly smaller than the mean and still produce accurate results. It is important that the peak accuracy does not occur at 0 and then linearly drop. This would of have indicated that our network is behaving like a normal neural network.

Figure 12 suggests increases the learning rate is worth while. Since the accuracy as the learning rate increases is shown to be relatively the same the model isn't very sensitive to changes in this parameter. Since a larger learning rate means learning faster it is advantageous to use a higher learning rate especially in this case where the accuracy does not diminish. It interesting to note that this differs from a regular NN, where we would typically not choose a learning rate as large as 1.

Figure 13 suggests the model is not very sensitive to the number of samplings from the weights and distributions. This is a surprising result as this number increases the amount of Monte Carlo samples does not make a difference on the accuracy.

Figure 14 suggests the threshold should remain below 0.5 which makes sense. A threshold above 0.5 means predictions that are closer to the incorrect target are given errors and error gradients of zero. This leads to the algorithm as seen dramatically dropping in accuracy since misclassifications are not being learnt from. The accuracy does not vary much between 0 and 0.5. We hoped to see that the making the threshold non-zero would combat overfitting, so that there would be an increase in the test accuracy, but that was not the case.

3) Interpretability: One added effect of using distribution on weights and biases is that the output neurons aren't just a single point estimate. This provides with a more visually informative prediction as can be seen below.

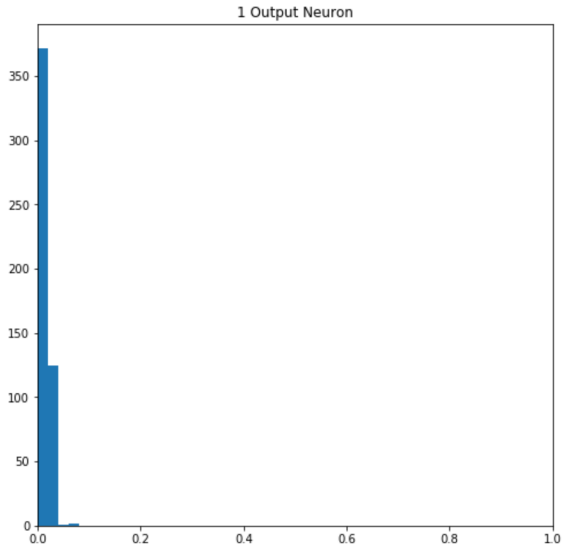

Fig. 15. Histogram of Neuron One's Output

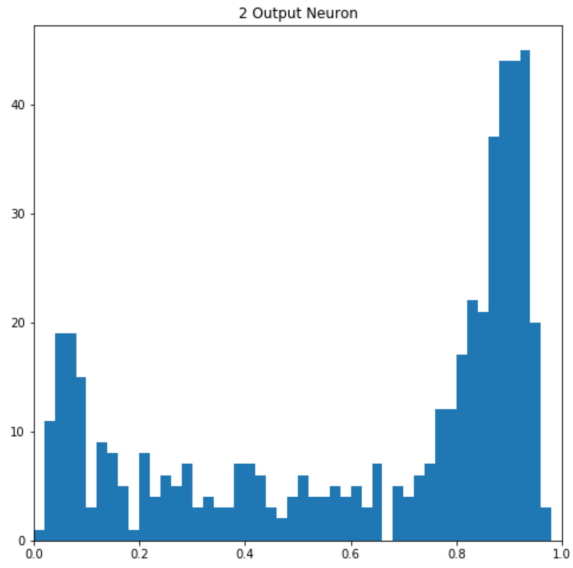

Fig. 16. Histogram of Neuron Two's Output

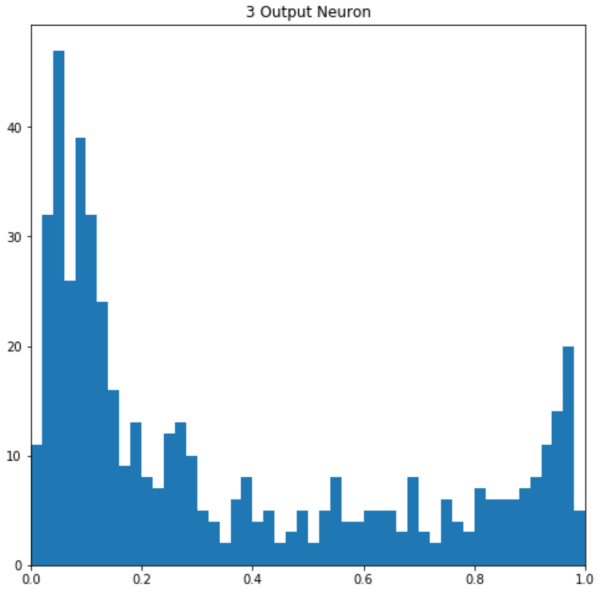

Fig. 17. Histogram of Neuron Three's Output

These histograms were produced on the iris dataset when predicting a test sample of target $[0, 1, 0]$, which we see was correctly predicted here. Though more information than the prediction being correct can be extracted. We also can get an idea to how certain the model was about each output. For example neuron 1 it was very certain it was a zero. Where as the model had a much more difficult time selecting between neurons 2 and 3. This can be used to provide practitioners with a better idea of what the confidence of each output may be.

There are some limitations and exclusions from this report. The regression cost function of MSE has been implemented however has not been thoroughly tested. The ideas behind the threshold parameter haven't been explored for the context of regression as well. Deeper neural network consisting of more than 2 layers also have not been explored within this report. Only Gaussian distributions and bootstrap have been implemented in the model for sampling the weights and biases. The bootstrap sampling of weights and biases are excluded from this report though are implemented in the code. The reason for excluding bootstrapping from the report is due to the way variance was implemented. Since the variance is implemented as a constant relative ratio to the mean, bootstrapping would not work given the updates.

B. Future Work

Some areas we would like to see this model explored more by others or ourselves are the following.

During experimentation deeper neural network with more than three layers were never explored. This would be interesting to see how the computational demand of the RSNN scales up. This report focused on prediction performance though it is also important to explore the computational costs and how they scale for this model.

The threshold seems like a promising area for more experimentation. During this report its implications and use in a regression setting have yet to be explored. In this report the threshold was explored numerically which have shown some interesting and promising results for future usage. A more rigorous and mathematical understanding would help in further developing more impactful applications of the threshold parameter.

An area where further researching is needed, is the selection of which distribution will be used for modelling the weights and biases. It would be interesting to see how using different distributions to model the weights and biases would affect the solutions. The current implementation has been designed to allow for easy integration of other distributions.

How will RSNN defend against adversarial attacks is also a topic we are particularly interested in. Since in theory, our network should be insensitive to the small perturbations on the data.

C. Conclusion

The results show promising evidence for the feasibility of the model and thereby more research of this model. The computational demand of sufficient Monte Carlo sampling was much lower than anticipated, as the number of samples are lower than expected. The performance of the model in terms of accuracy was competitive with a normal neural network and one with weight decay. Though more exploration of datasets and bench marking against a larger variety of model would help in understanding the model better and its place among models. Having the output of the model form a histogram seems like a rich area for improving the interpretability of neural networks. This report consisted of a more computational examination of the RSNN, however, a more mathematically rigorous approach should be explored.

V. Acknowledgements

Professor Jeff Orchard for providing structure of how a traditional neural network is implemented. Vincent Luong for hosting the package and helping setup the GitHub.

References

[1] S. Salman and X. Liu, "Overfitting mechanism and avoidance in deep neural networks," 2019.

[2] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, p. 85–117, Jan 2015.

[3] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural networks," 2015.

[4] H. Shu and H. Zhu, "Sensitivity analysis of deep neural networks," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, p. 4943–4950, Jul 2019.

[5] S. Huang, N. Papernot, I. Goodfellow, Y. Duan, and P. Abbeel, "Adversarial attacks on neural network policies," 2017.

[6] A. Natekin and A. Knoll, "Gradient boosting machines, a tutorial," *Frontiers in neurorobotics*, vol. 7, p. 21, 12 2013.