Michael Hodges
All files can be found here
https://github.com/Michael-Hodges/EECE5644_Machine_Learning.git

Problem 1

For MAP classification for minimium error we can calculate our decision rule is as follows:

$$\begin{aligned}\hat{\Theta}_{MAP} &= \operatorname*{argmax}_{\Theta} P(\Theta|Data)\\ &= \operatorname*{argmax}_{\Theta} \prod_{i=1}^{N} P(\Theta|x_i)\\ &= \operatorname*{argmax}_{\Theta} \prod_{i=1}^{N} P(x_i|\Theta)P(\Theta)\\ &= \operatorname*{argmax}_{\Theta} \ln \prod_{i=1}^{N} P(x_i|\Theta)P(\Theta)\\ &= \operatorname*{argmax}_{\Theta} \sum_{i=1}^{N} \ln P(x_i|\Theta)P(\Theta)\end{aligned}$$

We know plug in for a multivariate Gaussain

$$\begin{aligned}&= \operatorname*{argmax}_{\Theta} \sum_{i=1}^{N} \ln \frac{1}{\sqrt{(2\pi)^k|\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2}(\boldsymbol{x_i}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x_i}-\boldsymbol{\mu})\right)\\ &= \operatorname*{argmax}_{\Theta} \sum_{i=1}^{N} \ln \frac{1}{\sqrt{(2\pi)^k|\boldsymbol{\Sigma}|}} - \frac{1}{2}(\boldsymbol{x_i}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x_i}-\boldsymbol{\mu})\\ &= \operatorname*{argmax}_{\Theta} \sum_{i=1}^{N} \ln 1 - \frac{1}{2}\ln(2\pi)^k - \frac{1}{2}\ln|\boldsymbol{\Sigma}| - \frac{1}{2}(\boldsymbol{x_i}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x_i}-\boldsymbol{\mu})\end{aligned}$$

Now we drop terms that don't depend on Θ which is the paramaters of the gaussian (μ, Σ, and shared constants (-1/2

$$= \operatorname*{argmax}_{\Theta} \sum_{i=1}^{N} -\ln|\boldsymbol{\Sigma}| - (\boldsymbol{x_i}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x_i}-\boldsymbol{\mu})$$

We can then turn this into a minimization problem by dropping the negative

$$\hat{\Theta}_{MAP} = \operatorname*{argmin}_{\Theta} \sum_{i=1}^{N} \ln|\boldsymbol{\Sigma}| + (\boldsymbol{x_i}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x_i}-\boldsymbol{\mu})$$

and there is our MAP decision rule. Note we don't have any specific risk so we just use a 0-1 loss which is just the rule shown above.

Here is our distribution.

Class 1 mean: [3, 3, 3] Covariance: $\begin{bmatrix} 3 & 0.3 & 2 \\ 0.3 & 3 & 0 \\ 2 & 0 & 3 \end{bmatrix}$ Prior: 0.3

Class 2 mean: [-3, 3, 3] Covariance: $\begin{bmatrix} 4 & 0 & 0 \\ 0 & 4 & -3 \\ 0 & -3 & 4 \end{bmatrix}$ Prior: 0.4

Class 3 mean: [-3, -3, 3] Covariance: $\begin{bmatrix} 2 & 0.5 & 1 \\ 0.5 & 2 & 0 \\ 1 & 0 & 2 \end{bmatrix}$ Prior: 0.2

Class 4 mean: [3, -3, 3] Covariance: $\begin{bmatrix} 1 & 0 & 0.2 \\ 0 & 1 & 0.2 \\ 0.2 & 0.2 & 1 \end{bmatrix}$ Prior: 0.1

Actual number of samples per class:

Class 1: 3145

Class 2: 3971

Class 3: 1887

Class 4: 997

MAP Confusion Matrix for Rows as decision and columns as true Labels:

$$\begin{bmatrix} 2998 & 119 & 4 & 24 \\ 141 & 3786 & 43 & 1 \\ 5 & 49 & 1827 & 6 \\ 20 & 1 & 6 & 970 \end{bmatrix}$$

Total Errors: 419/10000
Theoretical minimum probability of error: 95.81%

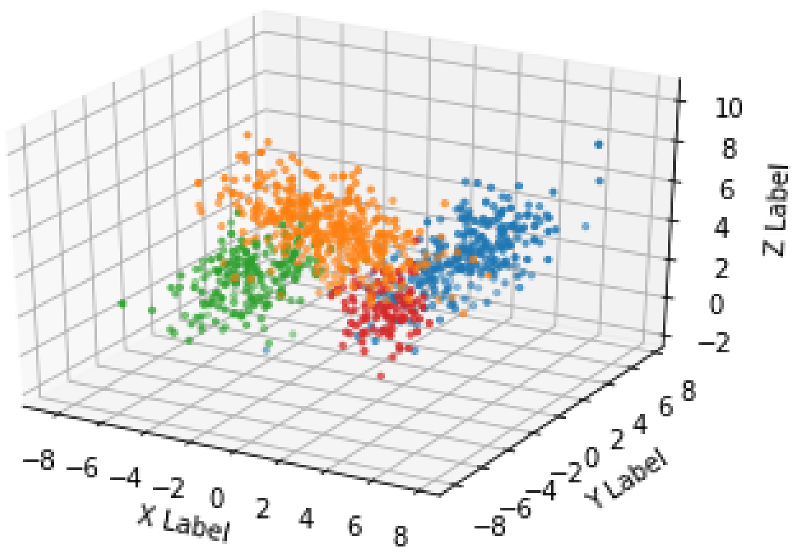

Figure 1: 1000 samples of distribution. (Only for visualization. Data not used for testing or training)

We will know look at the math to determine the optimization problem of the neural net. Given $\mathbf{x} \in \mathbb{R}^{3xN}$ and labels $\ell \in \mathbb{N}^N$ we need to find the maximum likelihood parameter estimation. Here's the math:

$$\begin{aligned}
\hat{\Theta}_{ML} &= \underset{\Theta}{\operatorname{argmax}} \ln \prod_{i=1}^{N} P(x_i, \ell_i|\Theta) \\
&= \underset{\Theta}{\operatorname{argmax}} \sum_{i=1}^{N} \ln P(x_i, \ell_i|\Theta) \\
&= \underset{\Theta}{\operatorname{argmax}} \sum_{i=1}^{N} \ln P(\ell_i|x_i, \Theta) P(x_i|\Theta) \\
&= \underset{\Theta}{\operatorname{argmax}} \sum_{i=1}^{N} \ln P(\ell_i|x_i, \Theta) + \sum_{i=1}^{N} \ln P(x_i|\Theta)
\end{aligned}$$

x is independent from the parameters so we drop the last term. We will also divide by the total number of samples and multiply by negative 1 to get a minimization problem

$$\begin{aligned}
&= \underset{\Theta}{\operatorname{argmin}} -\frac{1}{N} \sum_{i=1}^{N} \ln P(\ell_i|x_i, \Theta) \\
&= \underset{\Theta}{\operatorname{argmin}} -\mathbb{E}_x[\ln P(\ell_i|x_i, \Theta)]
\end{aligned}$$

The above will be our loss function. This function defines the cross entroyp or the Kullback-Leibler divergence as it shows the relative entropy of ℓ with respect to x. The output of our network will approximately be of the form $P(\ell_i|x_i, \Theta)$. Therefore, we will use the cross-entropy loss function in keras and give it our true class labels and the output of the network. We are told to use a MLP with 1 hidden layer. The hidden layer will use a nonlinearity (we will use the sigmoid) and the output will use the softmax function that will map probabilities into each of the 4 classes. The mapping will look something like this: input $x \in \mathbb{R}^3 -> $ Linear transformation using weights plus a bias $->$ non-linear activation using tanh $->$ Linear transformation using weights plus a bias $->$ softmax function. In math notation: $softmax(\mathbf{W}_2^T \sigma(\mathbf{W}_1^T \mathbf{X} + \mathbf{b}_1) + \mathbf{b}_2)$ Note: $\mathbf{W_1}$ transform the input of form $\mathbb{R}^3$ to the shape of the hidden network $\mathbb{R}^{\#nodes}$ and so will be adjusted during cross validation, $\mathbf{W_2}$ transforms the output of the activation function to the 4 classes we are trying to classify and so will take $\mathbb{R}^{\#nodes}$ and map to $\mathbb{R}^4$. Softmax takes the numbers we give it and provides a probability of each class such that they add up to 1. We use categorical cross-entropy in the back-propagation stage to update the weights and biases and this matches the math equation shown above. Furthermore, during training, we will train until the difference in our probability of error is $\Delta P(error) < 0.1$ or 0.001. We will perform 50 epochs between each check of the difference in error. Here are the results we achieved when using separate data sets of size 100, 1000, and 10,000:

Model Selection and results for N = 100

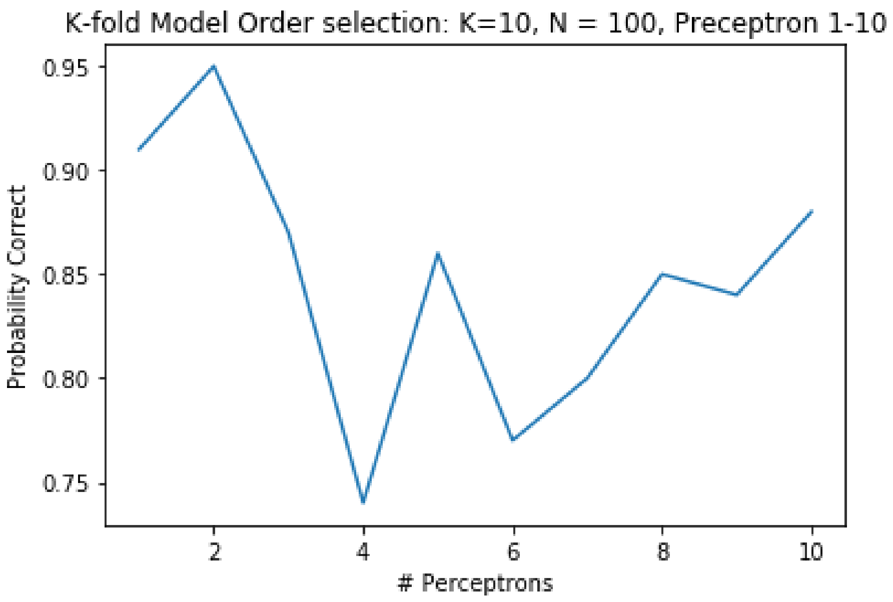

Figure 2: Mean Accuracy (1-P(incorrect)) for k(=10)-fold perceptron model selection. Order selected was 2

NN with training set N=100 Confusion Matrix for Rows as decision and columns as true Labels, Tested on original distribution:

$$\begin{bmatrix} 2751 & 262 & 9 & 123 \\ 106 & 3808 & 56 & 1 \\ 1 & 96 & 1778 & 12 \\ 2 & 1 & 4 & 990 \end{bmatrix}$$

Total Errors: 673/10000
Probability Correct Classification: 93.27%

Model Selection and results for N = 1,000

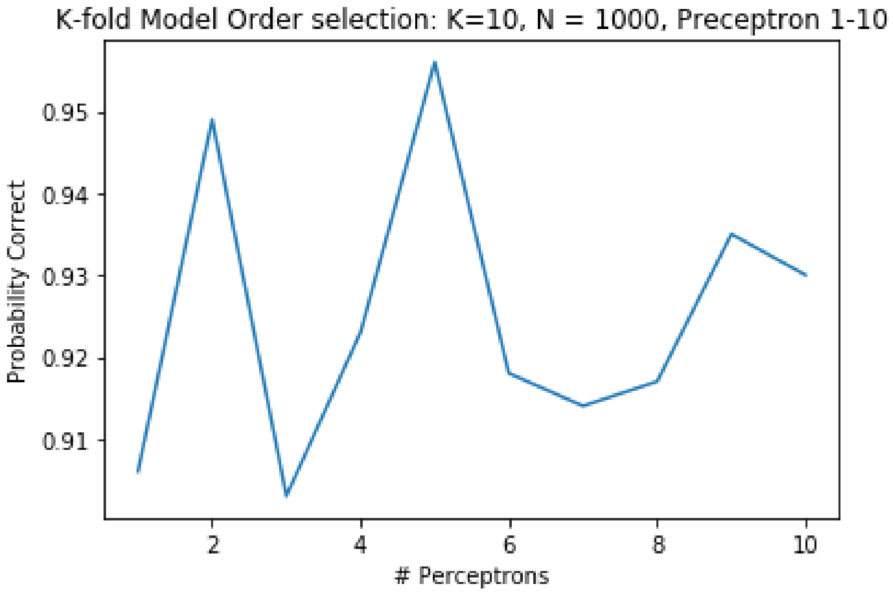

Figure 3: Mean Accuracy (1-P(incorrect)) for k(=10)-fold perceptron model selection. Order selected was 5

NN with training set N=1,000 Confusion Matrix for Rows as decision and columns as true Labels, Tested on original distribution:

$$\begin{bmatrix} 2865 & 235 & 3 & 42 \\ 108 & 3794 & 68 & 1 \\ 1 & 82 & 1788 & 16 \\ 24 & 1 & 4 & 968 \end{bmatrix}$$

Total Errors: 585/10000
Probability Correct Classification: 94.15%

Model Selection and results for N = 10,000

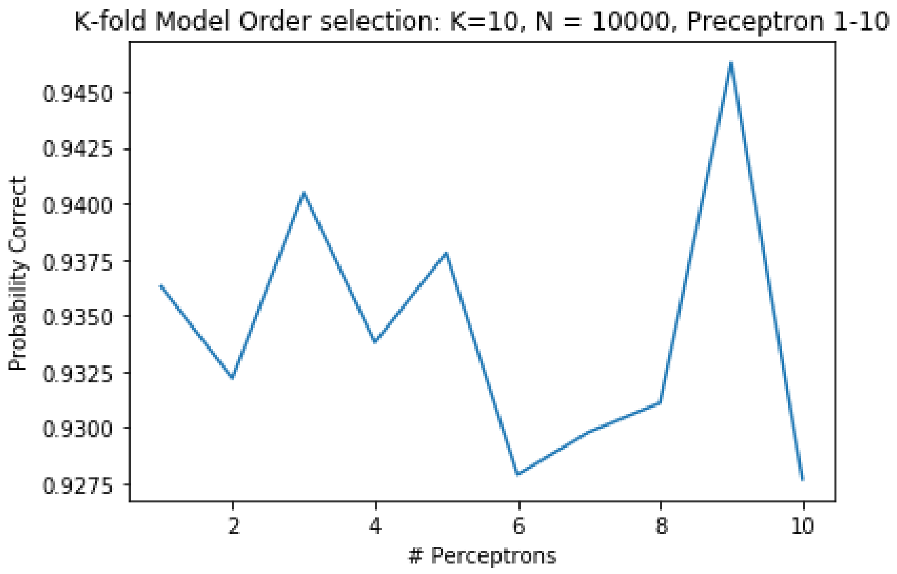

Figure 4: Mean Accuracy (1-P(incorrect)) for k(=10)-fold perceptron model selection. Order selected was 9

NN with training set N=10,000 Confusion Matrix for Rows as decision and columns as true Labels, Tested on original distribution:

$$\begin{bmatrix} 2971 & 144 & 6 & 24 \\ 134 & 3796 & 40 & 1 \\ 2 & 59 & 1819 & 7 \\ 29 & 4 & 5 & 959 \end{bmatrix}$$

Total Errors: 455/10000
Probability Correct Classification: 95.45%

Generally speaking, we see a decrease in the total number of errors as we have more data to train on. Note that we stay below the theoretical maximum probability of correct classification for all of the neural nets we generated.

Problem 2

Here is the ML estimator we are solving:

Given that the function f_{true} generates data $d \approx f_{true}(x) + v$ where $v \sim \mathcal{N}(0, \sigma^2 \mathbf{I})$ we are trying to find Θ such that $d \approx f(x, \Theta)$ we train this using

$$\hat{\Theta}_{ML} = \underset{\Theta}{\text{argmax}} \sum_{i=1}^{N} \ln P(x_i, d_i|\Theta)$$
$$\hat{\Theta}_{ML} = \underset{\Theta}{\text{argmax}} \sum_{i=1}^{N} \ln P(d_i|x_i, \Theta)P(x_i|\Theta)$$

And since we assume our data is generated independent of the model order we can drop that term

$$\hat{\Theta}_{ML} = \underset{\Theta}{\text{argmax}} \sum_{i=1}^{N} \ln P(d_i|x_i, \Theta)$$

We know that a gaussain simplified using the logarithm is as follows:

$$\mathcal{N} = \ln\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) + -\frac{(x-\mu)^2}{2\sigma^2}$$

Dropping constant terms that don't matter and dropping the denominator since it won't affect the minimization.

$$\mathcal{N} = -(x-\mu)^2$$
$$\hat{\Theta}_{ML} = \underset{\Theta}{\text{argmin}} - \sum_{i=1}^{N} (x-\mu)^2$$

When we plug in with our data values and our mean we get the following:

$$\hat{\Theta}_{ML} = \underset{\Theta}{\text{argmin}} - \frac{1}{N} \sum_{i=1}^{N} (d_i - f(x_i, \Theta))^2$$

And this is mean squared error, which we will use for our loss function.

Our cross validation works as follows We will iterate as follows: activation -> model order -> perform cross validation. Essentially, we select our activation and then we go from a model order of 1-10 performing 10-fold cross validation for each model order. Then we select the next activation function and do the same. The result is we end up with average MSEs for both activation functions with model orders 1-10. During training we run in epoch's of 50 and check to see if the MSE has dropped by more than 0.01, as in we are converged if $abs(oldMSE - newMSE) < \epsilon$.

Here is the data we will use to train and test our model respectively:

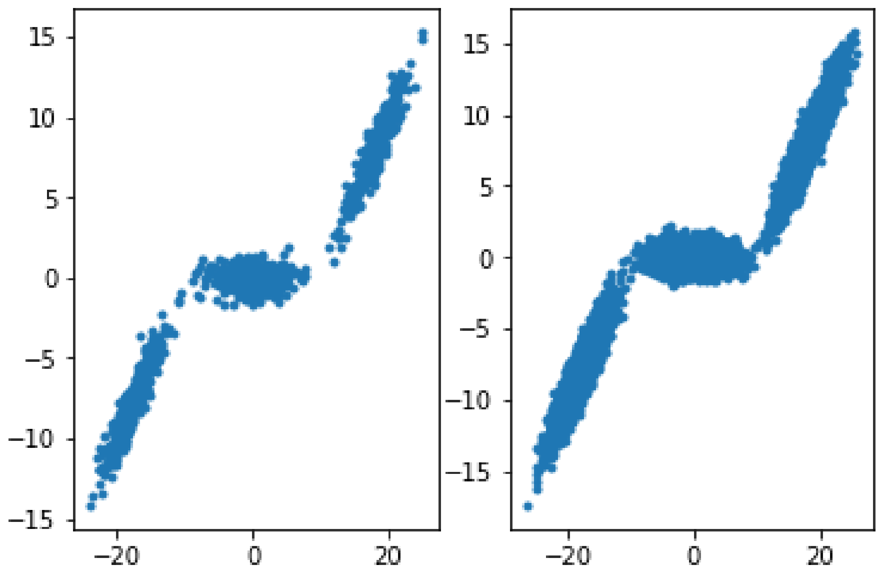

Figure 5: **left** 1,000 samples of training data. **right** 10,000 samples of testing data.

Using the K-fold Cross validation here is what our model order selection looks like:

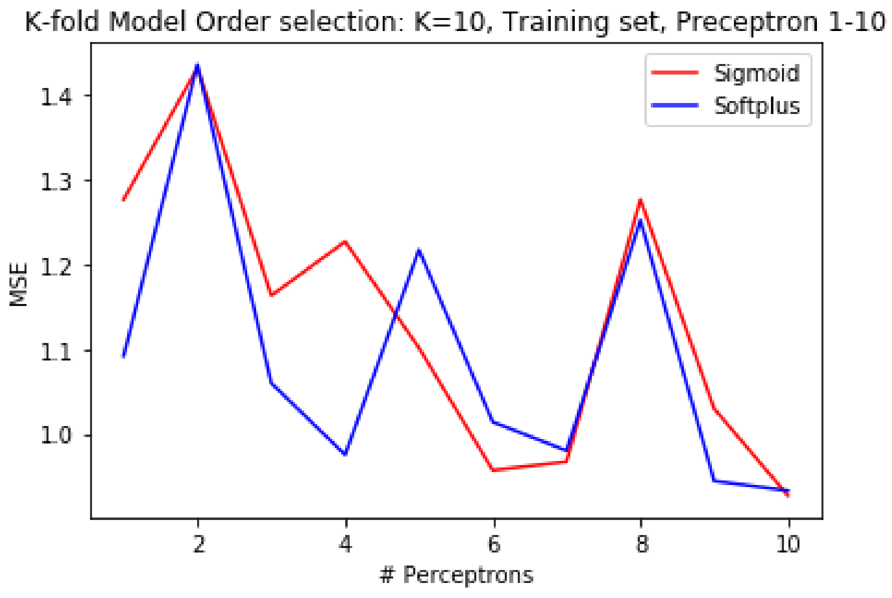

Figure 6: Final Choice Activation = Sigmoid with Perceptrons = 10

The Mean MSE of the selected model on the training data through k-fold cross validation was: 0.92701299

And finally here is the final output of our NN after training on all the training data and testing on the test data:

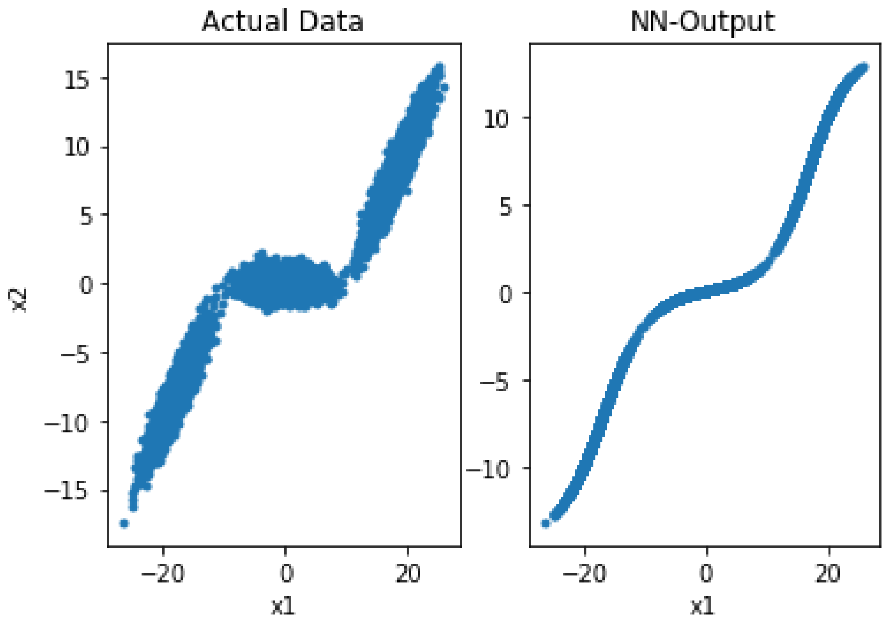

Figure 7: MSE on Testing Data: 0.6696991994857788

Test Performace MSE: 0.6606809679985046

Just a final note: We know that our NN output is going to be a linear combination of the activation function. And since our function has sharp turns we should probably expect it to perform best with a ReLU. So we tested with a ReLU and here is the result:

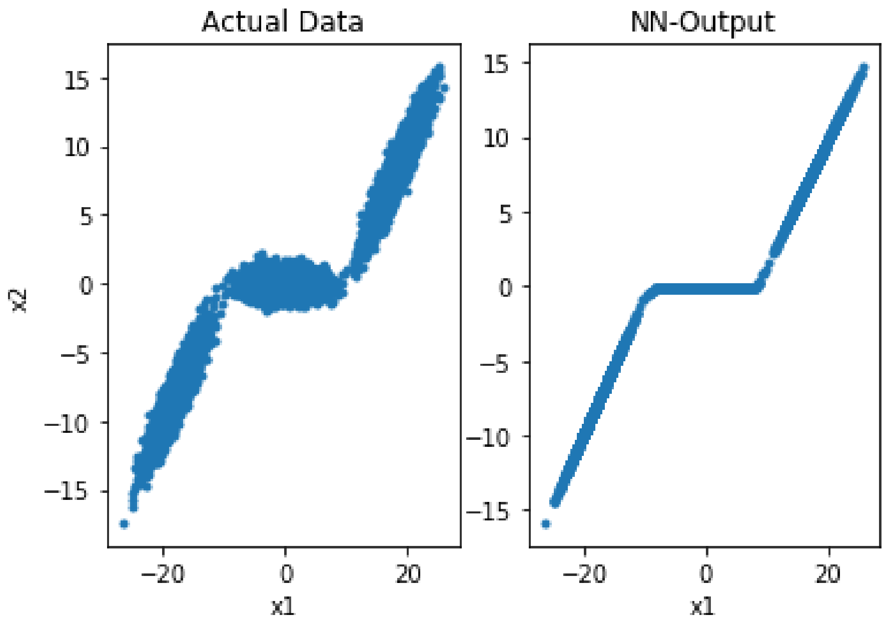

Figure 8: ReLU with model order = 4. MSE on Testing Data: 0.6075891286373138

Appendix