Experiments with Neural Networks using R

Seymour Shlien

December 15, 2016

1 Introduction

Neural networks have been used in many applications, including financial, medical, industrial, scientific, and management operations [1]. For example, a financial institution would like to evaluate an applicant's creditworthiness based on the applicant's personal data, income, expenses, previous credit history, etc.

Mathematically, a neural network with one hidden layer is of the form of

$$y_k = f_k(\alpha_k + \sum_j w_{jk} f_h(\alpha_j + \sum_i w_{ij} x_i)) \tag{1}$$

[2]. $f()$ is called an activation function. Typically it is a logistic function, $f(x) = 1/(1+\exp(-x))$. α are thresholds and w_{jk} are weights connecting the input or hidden nodes to other hidden or output nodes. Neural networks attempt to create a functional approximation to a collection of data by determining the best set of weights and thresholds. In the regression model, the output is a numeric value or vector. It has been proven theoretically that a neural network can approximate a continuous function to any degree, given a sufficient number of hidden nodes. Though knowledge of the existence of such a solution is reassuring, it does not lead to any deterministic method for finding such a solution.

Neural networks are treated as a black box for modeling data. They are an excellent method for classifying data; however, they provide little insight on how it works. It is necessary to use a trial and error process for finding the best configuration of the neural network. There are many free packages for applying neural networks to a problem.

There are many practical issues associated with data analysis and modeling. First, there is the curse of dimensionality; secondly, there is the risk of overfitting. Data is expensive to collect and to clean, so the training set is rarely large enough to estimate the hundreds of free parameters in a model. In real applications one rarely has the benefit of knowing the structure of the data. It is not possible to visualize data beyond 3 dimensions; furthermore, one does not have the luxury of having an unlimited training set. One does not know whether the current model is reasonable for the current data or whether there is another approach that yields better results.

Though there are a lot of free and standard data sets that are available from the internet, for example [3], there are advantages to working with synthetic data. First, the statistical characteristics of the data are known in advance. Secondly, there is no limit to the complexity or the amount of data that can be generated. This allows one to investigate the effect of sample sizes and number of input parameters on the performance of the network.

This note is mainly concerned with the the multilinear perceptron (mlp) or feed forward network. The note, like a laboratory report, describes the performance of the neural network on various forms of synthesized data.

This study was mainly focused on the mlp and adjoining predict function in the RSNNS package [4]. RSNNS refers to the Stuggart Neural Network Simulator which has been converted to an R package. R is a free software environment for statistical analyses and plotting. The software can run under under many operating systems and computers. There are over 9000 packages that can be imported into R, a mature and widely used language. There are numerous books on R and how it can be applied to machine learning, pattern recognition, and data analysis [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17].

There are two main sections in this note, titled Regression and Classification. The next section, titled Regression, is mainly an introduction to neural networks and is relatively short. The rest of the note will concentrate on classification, where we deal with finding an algorithm that correctly categorizes the data based on a labeled training set.

In order to be able to visualize what is occurring, we begin with data in two dimensions. This data still exposes many interesting characteristics and also allows one to see what is happening.

2 Regression

The RSNNS mlp() algorithm is a nondeterministic algorithm for finding the neural network parameters which best describe the data. Once the model is found, one can check its accuracy by running the training set and test set through a predict function which runs the data through the neural network model and returns the model's prediction. The predictions can then be compared with values associated with the two sets. The performance of the model on the test set is the true measure of its accuracy.

Though the predict function is part of the RSNNS package, it was found advantageous to re-implement it in R code. This implementation allowed us to try some additional experiments with the model. A description of the implementation is given in a separate section near the end of this note.

To begin, we shall show how the neural network approximates a numerical function. The radial $sin(x)/x$ function, or $sinc(x)$ function in the x-y plane, Figure 1 was used for studying this behaviour. Training samples uniformly spaced over a rectangle were created using a random number generator. The test data for verifying the model consisted of 900 samples spaced over a 30 by 30 grid.

Poor fits were obtained when the hidden nodes were restricted to a single layer (Figure 2); however, the fit improved significantly when 12 hidden nodes were split evenly between two layers Figure 3. See Figure 5.

The mlp (mulilayer perceptron) algorithm did not converge consistently and several trials were necessary to get a good fit. Figure 4 shows how the SSE (sum of the squared error) falls with each iteration for numerous trials. The algorithm attempts to minimize the SSE, but since there are many local minima, the algorithm frequently gets stuck in one of the minima. It appears that after 1000 iterations one can tell whether the algorithm will reach a good solution.

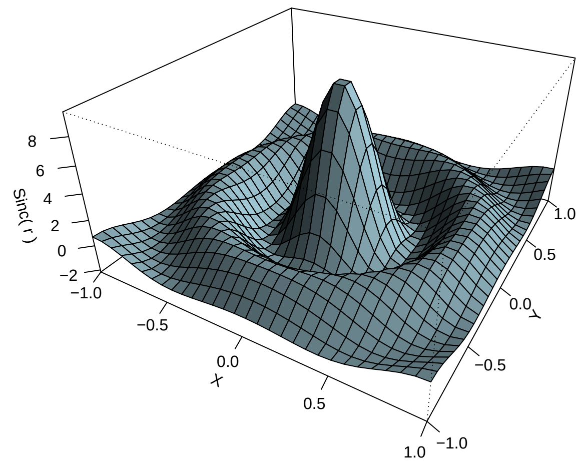


Figure 1: Two dimensional sinc function.

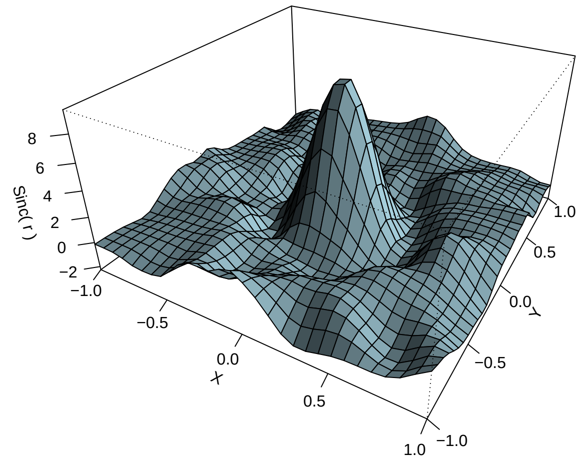


Figure 2: Approximation of the two dimensional sinc function based on the neural network with 20 hidden nodes in one layer and a training set consisting of 3000 samples. There are many obvious problems with this approximation.

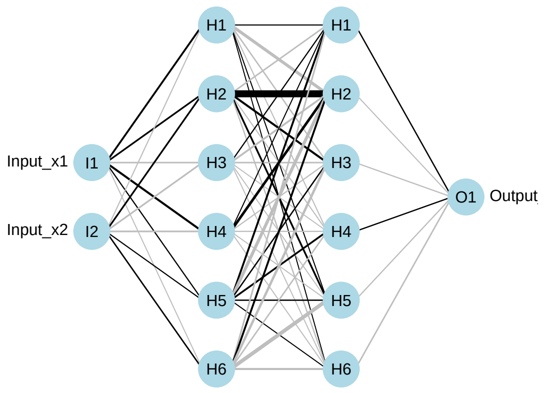


Figure 3: Neural net for sinc function regression problem in 2 dimensions. The best model had two hidden layers, each of which contains 6 nodes. There are 54 connections, each having its own weighting parameter to be estimated. The graph was produced by the neuralnettools package.

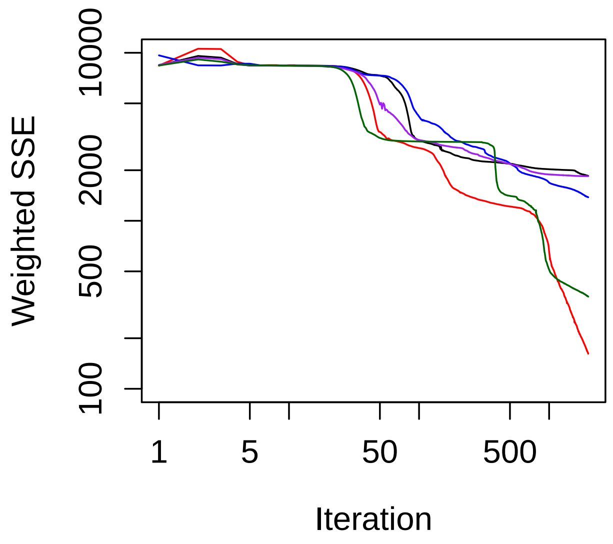


Figure 4: Convergence error for 5 runs of the back propagation algorithm. The weighted sum of the squares of the discrepancy between the fitted and given data is plotted for a training set of 3000 samples. Note that both vertical and horizontal scales are logarithmic in order to show the wide range of the data.

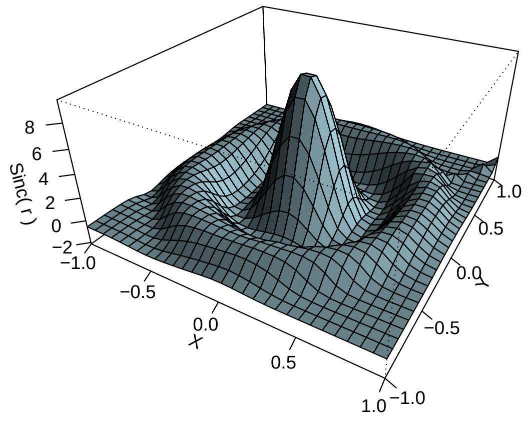


Figure 5: Fitted sinc function using a neural network with two hidden layers – six nodes in each layer. The Rprop algorithm achieved an SSE of 140 on a training set of 3000 samples. To convert SSE to MSE, divide by the number of training samples.

3 Classification

In the classification problem, the range of the function is not a decimal value in real space but instead a fixed set of class labels, usually represented by a small set of positive numbers. For example, Figure 6 shows 500 training samples in a 2-dimensional space that are classified into 5 different colours. The training algorithm attempts to produces a mapping function similar to Figure 7.

We will confine all our experiments to cases where the classes are perfectly separable and arise from uniform distributions. Though this situation rarely occurs in nature, it is ideal for studying the behaviour of classifiers. Based on this assumption, we know that the ideal classifier can achieve perfect accuracy. If the training data accuracy is significantly less than 100 percent, then we know there is bias error and that the architecture of the classifier does not provide sufficient flexibility to match the data. If the training accuracy is close to 100 percent but the test data accuracy is significantly less, this indicates that the model overfits the data or that the amount of training data is insufficient to allow the model to generalize properly. It will be shown that, even for a few perfectly separable classes and in low dimensions, there is a wide range of complexity. We will also consider the situation where the input data are binary or categorical rather than numerical.

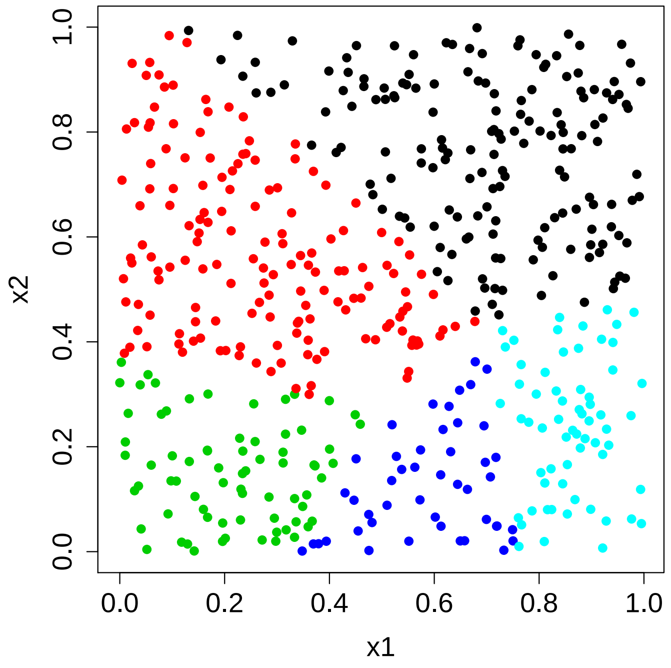


Figure 6: 500 training samples inside a square for a 5 class problem. The training samples were generated by a random number generator and the samples were then mapped to the closest centroid as described in this report.

The neural network requires that each of the classes is represented by a separate output as shown in Figure 8. The training samples are now given vector labels called response vectors. For example, if a sample is associated with class 2 out of a total 5 possibilities, then its response vector would be (0,1,0,0,0). Given a particular test sample, the output nodes of the neural network would be combined to form a predicted vector, for example (-0.2, 0.6, 0.3, 0.2, 0.1). The component number with the highest value, here 2 corresponding to the maximum 0.6, would be the predicted class number (winner gets all).

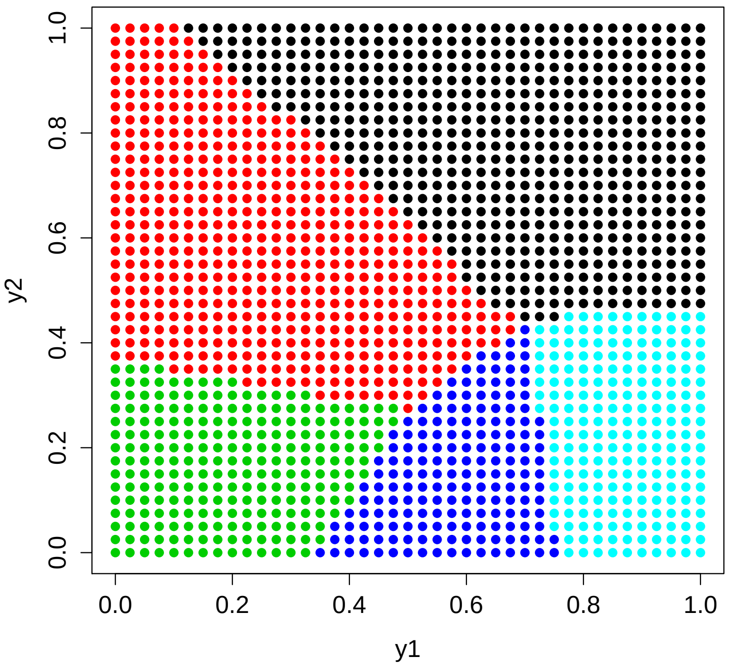


Figure 7: Using a test set spaced on a grid, the inferred mapping function based on the above training set is shown. The test samples were classified based on the model determined by the RSNNS mlp function using the SCG algorithm. The model contained 6 hidden nodes in one layer.

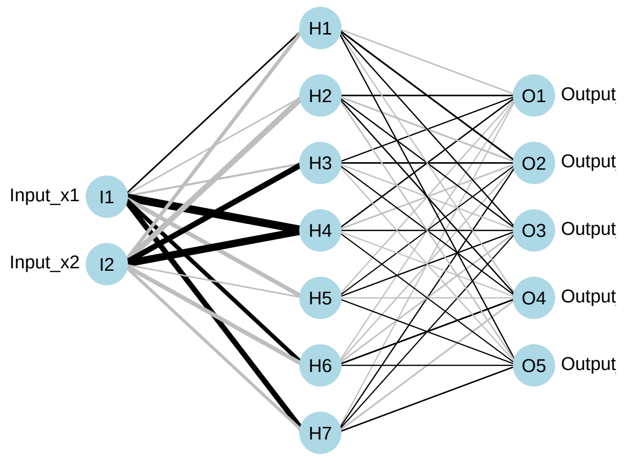


Figure 8: Neural net for 5 class problem in 2D space with one hidden layer containing 7 nodes. There are 49 connections each requiring a weighting parameter. The graph was produced by the neuralnettools package.

3.1 Data Samples

For purposes of this study, I used a set of random samples uniformly distributed inside a unit square (or hypercube in higher dimensions). The space was partitioned into separate regions corresponding to the different classes. The classes are linearly separable only in the sense that the borders separating the classes are straight lines or hyperplane segments in higher dimensions. The ideal classifier should be able to achieve perfect accuracy.

It is fairly easy to set up such a problem. One selects a set of n random centroids inside this space. The random samples in the space are then assigned to the closest centroid. Each of the classes is enclosed by a convex polytope. Figure 9 and Table 1 show the distribution of the 5 centroids used to classify the random samples.

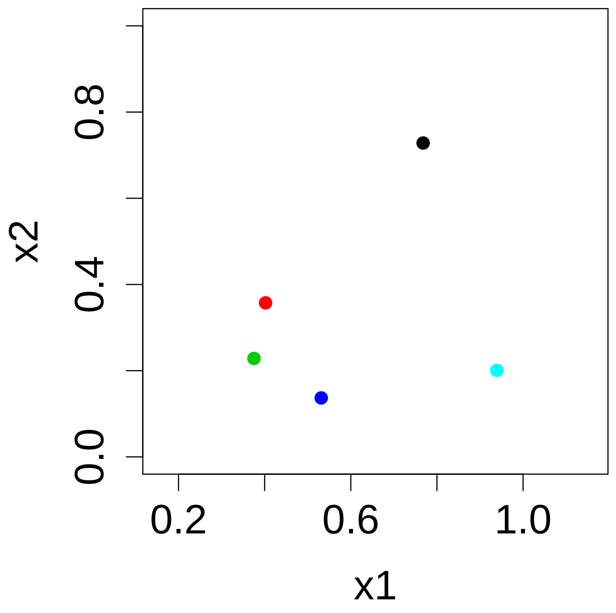


Figure 9: Plot of the centroids used to classify the samples using the nearest neighbour criterion.

Black	0.7680059	0.7282086
Red	0.4021301	0.3574010
Green	0.3751647	0.2285103
Blue	0.5313722	0.1368736
Light Blue	0.9390622	0.2008125

Table 1: Centroids used to classify the samples for most of the experiments described in this paper.

Here is the R code for generating the data samples.

```
Dim <- 2 # Dimension of parameter space
nclasses <- 5 # Number of classes
training.size <- 500 # Number of training samples
```

```
test.size <- 5000 # Number of test samples
synthetic.size <- training.size + test.size

# centers are the centroids of the Voronoi regions
centers <- matrix(runif(Dim * nclasses, 0.05, 0.95), nrow = Dim)
synthetic.samples <- matrix(runif(Dim * synthetic.size, 0, 1), nrow = Dim)

euclid <- function (x1, x2) {
  return (sqrt(sum((x1 - x2) ^ 2)))
}

nearest_center <- function (x) {
  d <- c()
  for (i in 1:nclasses) {
    d <- c(d, euclid (x, centers[, i]))
  }
  return (which.min(d))
}

class_matrix_to_vector <- function (modeled_output) {
  size <- nrow(modeled_output)
  classvalues <- c(rep(0, size))
  for (i in 1:size) {
    class <- which.max(modeled_output[i, ])
    if (modeled_output[i, class] > 0) {
      classvalues[i] <- class
    } else {
      classvalues[i] = NA
    }
  }
  return(classvalues)
```

This model is very flexible, allowing us to investigate the effect of increasing the number of classes or extending the space to 3 or more dimensions (Figure 10). Though it is possible to generate very complicated models, I did not have time to wait for the mlp algorithm to converge for these complex configurations.

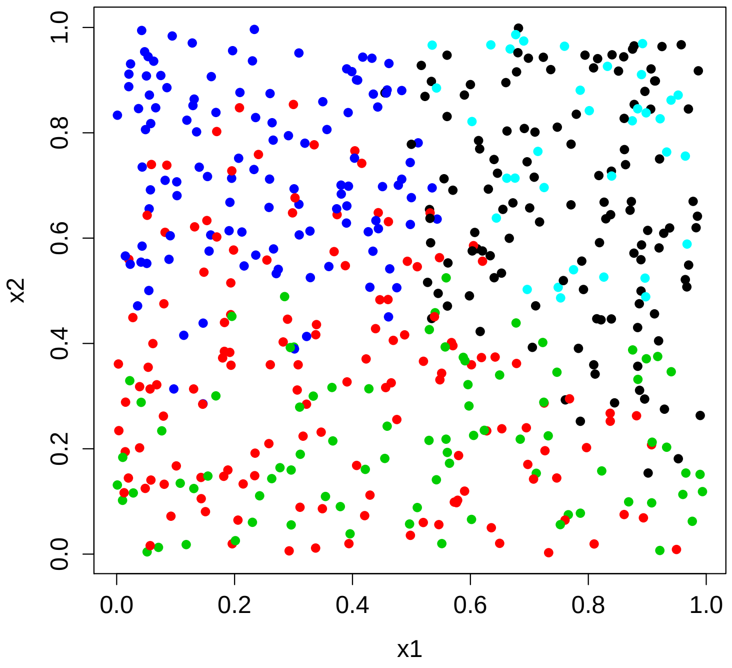


Figure 10: Two dimensional projection of 500 training samples for 5 class problem, 3-dimensional space. Though the classes do not appear to be separable in this projection, they are perfectly separable in 3-dimensional space.

Assuming that one needs a discriminant for every distinct pair of classes, then we would need k(k-1) discriminants for k classes. Each discriminant requires d parameters, where d is the dimension of the space. Therefore the classifier requires a minimum of k(k-1)d free parameters. (In low dimensional spaces not all pairs of classes have common borders, so we could manage with fewer discriminants and parameters.)

3.2 Convergence

How many iterations do you require? This depends upon the number of training samples and the configuration of the neural network. The error rate was computed for both the training set (500 samples) and the independent test set (5000 samples) and plotted for different stopping points in Figure 11.

The mlp algorithm attempts to reduce the sum of the squared error (SSE)by determining the best weights associated with connectors going to the nodes. The minimization is performed on the training set, so the training set error rate is expected to fall along with the SSE. Assuming the neural network generalizes to the test set, the test error rate is expected to fall as well. It appears that after 1000 iterations, there is barely a significant improvement in the test set error rate, though the training set error continues to decline. This suggests that the mlp algorithm is tending to overfit the data beyond 1000 iterations. If the training set is increased to 2000, the training set error decreases from 2 percent to 1 percent. If the training set is limited to 300 samples, the test set error increases to 5 percent.

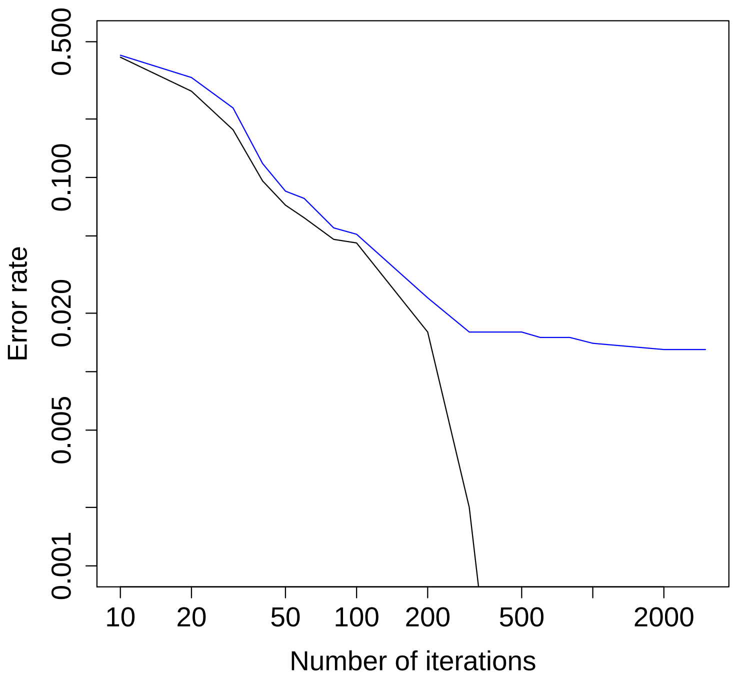


Figure 11: The error rates computed for the training set (black) and for the test set (blue) versus the number of iterations performed by the mlp training algorithm using the SCG algorithm and 7 hidden nodes in a single layer. Training was done on 500 samples for the 5 class problem in 2 dimensions. The neural network achieved 100 percent accuracy on the training data after several hundred iterations.

Besides the input training data and corresponding target values, the RSNNS mlp function has several other important parameters. The number of hidden units and their configuration is specified by the size = argument and the number of iterations to perform specified by the maxit = argument have important bearing on the accuracy of the model. By default, mlp uses the Rprop algorithm; however, for our example the convergence is significantly faster using the SCG (scaled gradient conjugate) algorithm as demonstrated in Figure 14.

The mlp function initializes itself with a random set of weights when it begins and therefore is not a completely determinisitic function. The program then attempts to find the weights which minimize the SSE. Due to the presence of many local minima, the algorithm converges to a different solution each time it is run. (To ensure repeatability, the random generator is initialiazed with a seed 123456 for most of the tests.) This is why some of the graphs for the training and test errors have bumps for some values. Ideally, one should rerun the the learning algorithm numerous times and choose the best solution. Figures 12 and 13 show the distribution of the SSE when the mlp algorithm is rerun 50 times on the same data.

The RSNNS mlp function comes with a choice of many learning functions, learning function parameters, and update functions, not to mention the incorporatation of a cross-validation set and pruning function. An investigation of these features would be a separate study on its own and has been deferred to a future time. According to the 'No Free Lunch Theorem' [18], there does not exist any optimization algorithm which is universally superior. Thus there exists a data set where another optimization algorithm performs better.

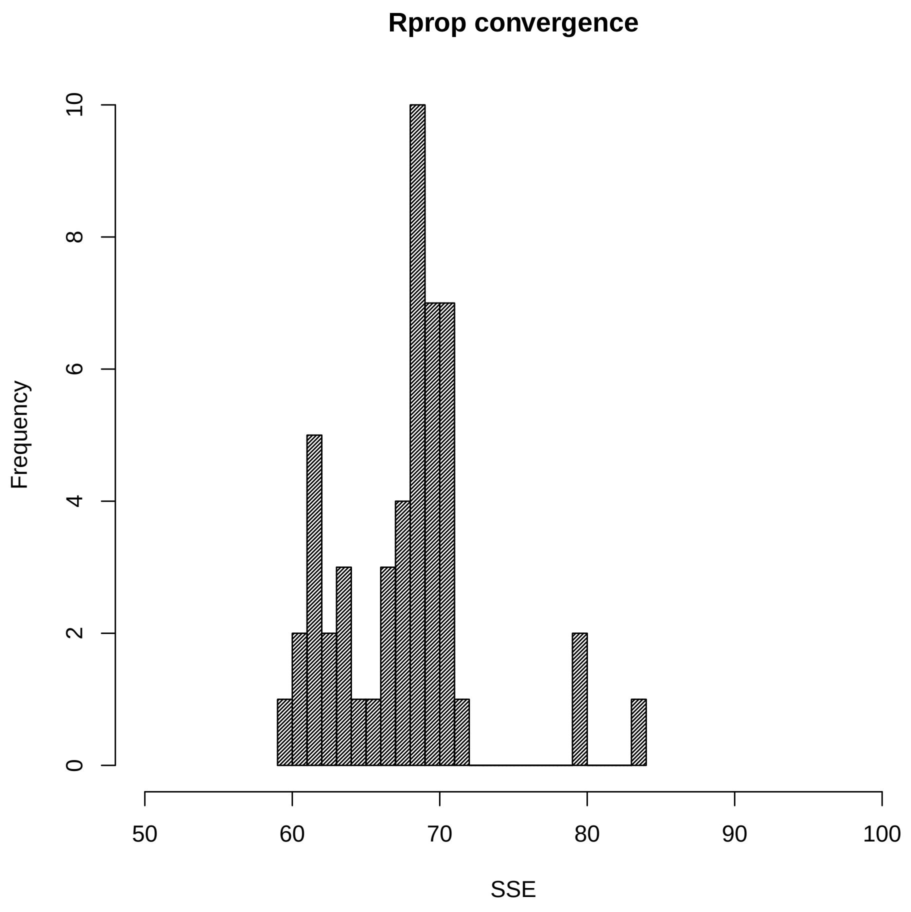


Figure 12: Histogram of SSE for mlp, 500 samples, Rprop, hidden=7. The mlp function was rerun 50 times and the final SSE for each run was plotted.

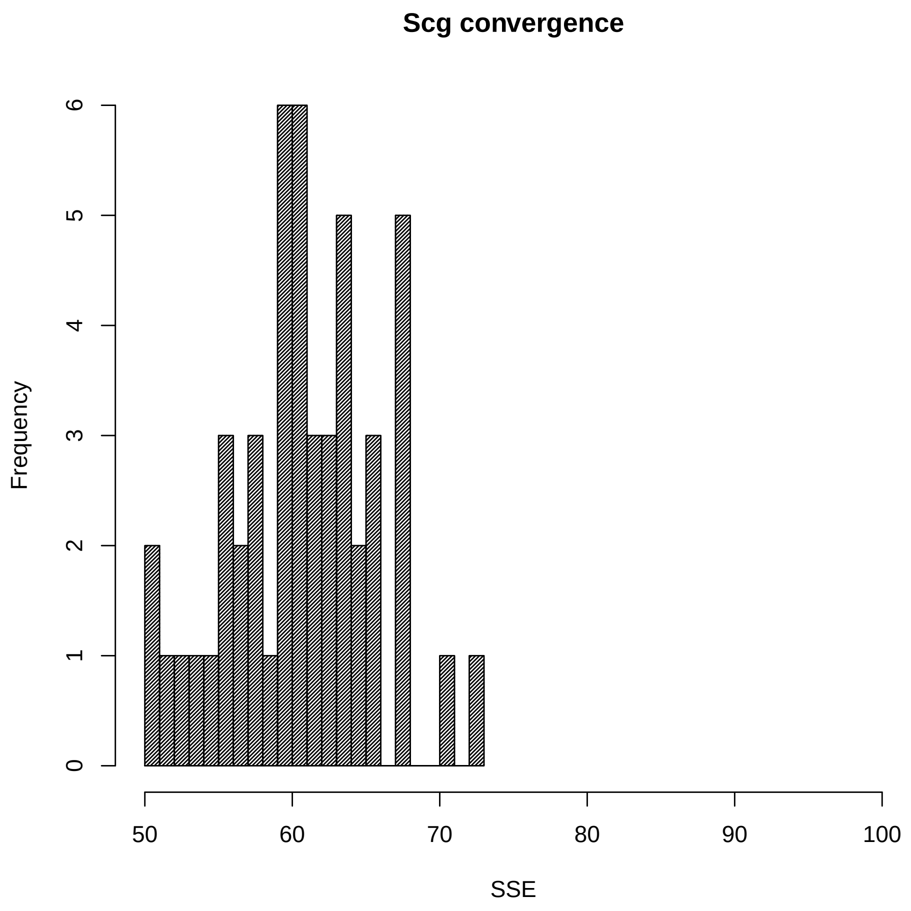


Figure 13: Histogram of SSE for mlp, 500 samples, SCG, hidden=7. The histogram indicates that the SCG algorithm is superior to the Rprop algorithm for this data. However, I did not succeed to get any convergence using the SCG algorithm for the sinc function regression problem described earlier.

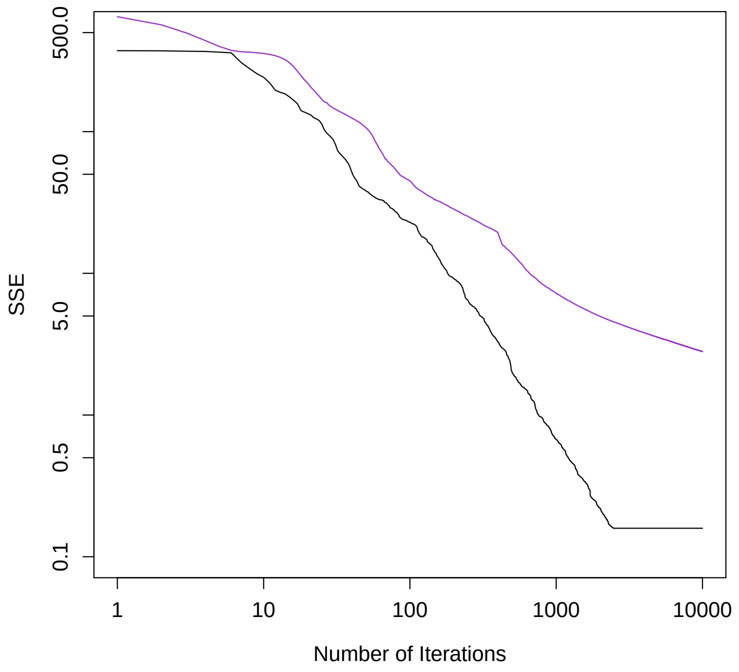


Figure 14: Comparison of the Rprop and SCG learning algorithms. Training was done on 500 samples for the 5 class problem in 2 dimensions. The neural net contained one hidden layer composed of 7 nodes using the learning function parameters (0.05,0.0). Black for SCG and Purple for Rprop.

3.3 Hidden units

How many hidden nodes do we need? On the basis of Figures 6 and 7 6 borders are needed to segment the space into the 5 regions. This provides further support to the selection of 5 nodes. It turns out that we can manage with as few as 3 hidden nodes as seen in Table 2. As seen in Figure 16 some of the borders can be merged together to create a longer separation. For example, the green, red and black regions can be separated from the two blue regions by a wiggly horizontal line.

It is difficult to see what is going on by examining the weight matrix associated with the neural network model; however, one can easily see the purpose of the hidden nodes graphically. In Figure 17, the activation of a hidden node is plotted as a function of the neural network inputs (x,y), also see [19]. The activation values range from 0.0 to 1.0 and are forwarded to the output nodes after applying weighting factors. The activation values of the hidden nodes are affected by the weighting factors that are applied to the links from the (x,y) inputs to the hidden nodes. The boundary between the low and high activation values is reflected in the boundary that separates the classes. This can be verified by disabling one of the three hidden nodes and viewing the output of the neural network model [20]. In Figure 18, a hidden node is disabled by locking its activation to 0.0 for all input (x,y) values. As seen, this prevents the neural network from distinguishing certain pairs of classes in certain regions of the space. Compare these representations with Figure 16 or 15.

Though 3 hidden nodes is the bare minimum, Table 2 indicates that the test set accuracy of 0.974 does not get much better when more hidden nodes are added (for a training set of 500 samples).

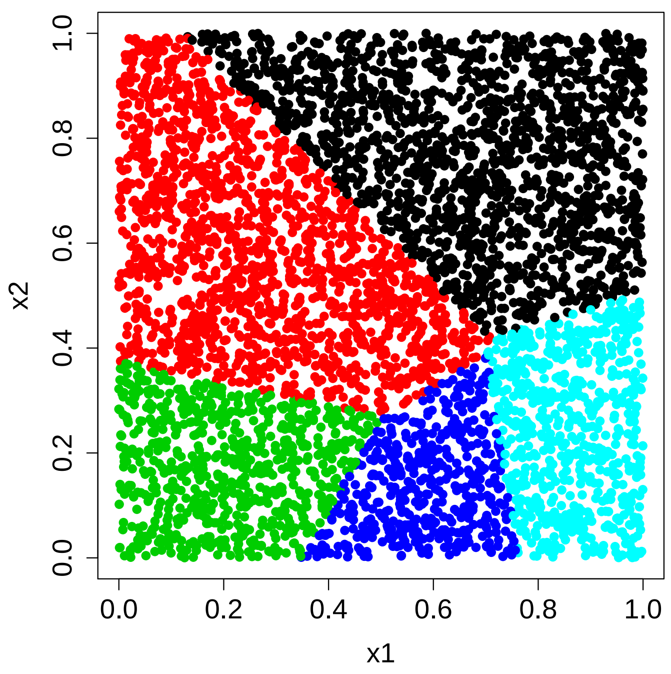


Figure 15: 5000 test samples for evaluating the neural net model with 3 hidden nodes in a single layer. Compare this with the mapping function in the previous figure.

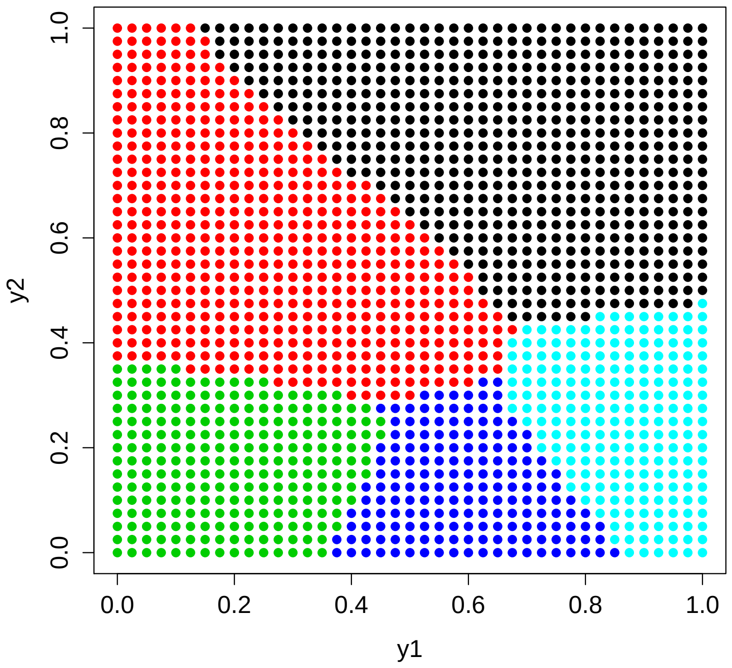


Figure 16: Predicted mapping function determined by the RSNNS mlp SCG algorithm, 500 training samples and 3 hidden nodes in a single layer.

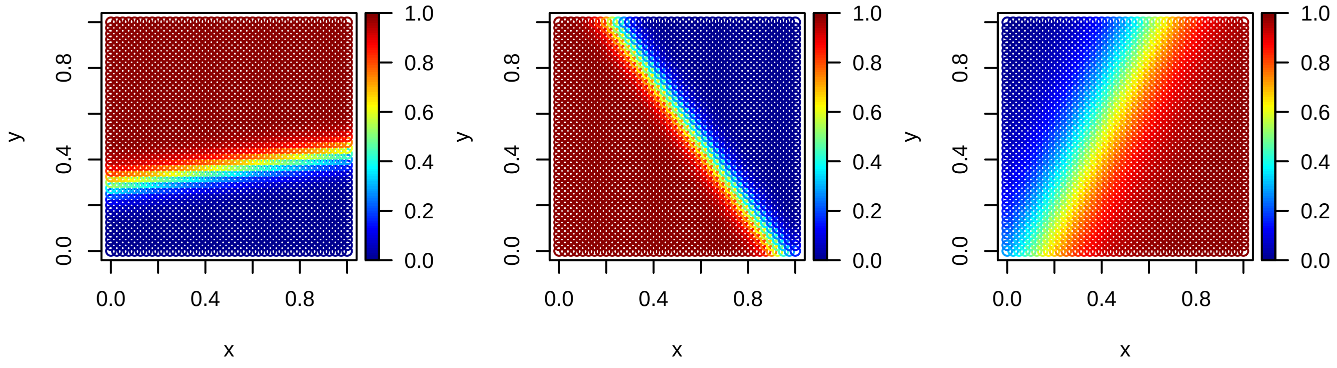


Figure 17: Activations for the 3 hidden nodes as a function of sample coordinates.

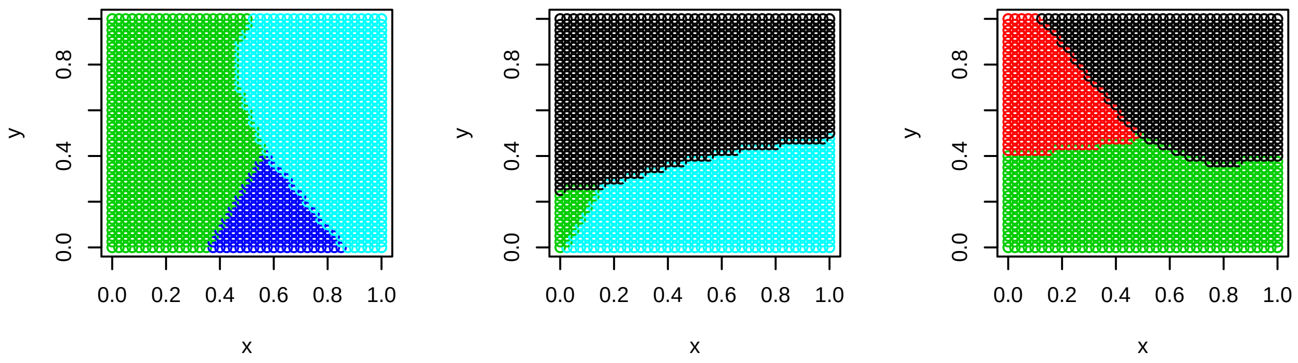


Figure 18: Neural network outputs after disabling hidden node 1, 2, and 3 respectively.

D	hidden	SSE	training set accuracy	test set accuracy
2	2	67.9	0.934	0.922
2	3	21.2	0.980	0.974
2	4	6.0	0.994	0.989
2	5	0.9	1.000	0.987
2	6	0.5	1.000	0.978
3	2	178.3	0.782	0.752
3	3	34.1	0.964	0.951
3	4	21.1	0.984	0.967
3	5	11.4	0.990	0.960
3	6	7.0	0.996	0.966
4	4	31.5	0.968	0.926
4	5	6.6	0.998	0.955
4	6	4.9	0.998	0.957
5	4	23.6	0.972	0.932
5	5	12.2	0.992	0.939
5	6	10.1	0.990	0.935
6	5	2.3	0.996	0.967
6	6	1.0	0.998	0.954
6	7	1.1	0.998	0.946
6	8	4.6	1.000	0.902
6	9	0.0	1.000	0.941

Table 2: Accuracy versus dimensions of sample space and number of nodes in a hidden layer for the 5 class problem. The SSE, training set accuracy and test set accuracy were estimated using 500 training samples, mlp algorithm with SCG with, and 1000 iterations. Best results out of 5 trials are presented. As the dimension of the space increases to 6, we require more hidden nodes in order to achieve perfect training accuracy; however, the test accuracy continues to fall. As the dimension of the space increases, the volume of the space increases exponentially, and we require many more training samples in order to map out the space.

These results apply only to this particular training and test sets. If we apply the same methods to a new set containing 9 classes instead of 5, it is clear that 7 hidden nodes is no longer sufficient. Based on Figure 19 11 hidden nodes would be preferable. Imagine what would happen if we had 50 classes.

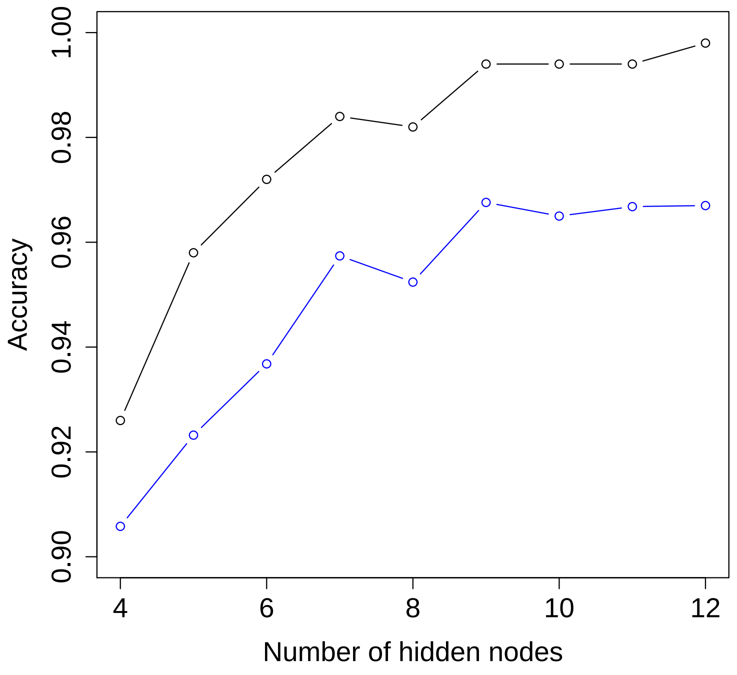
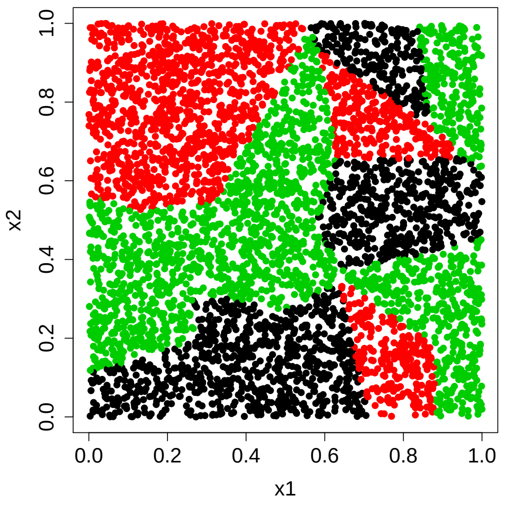


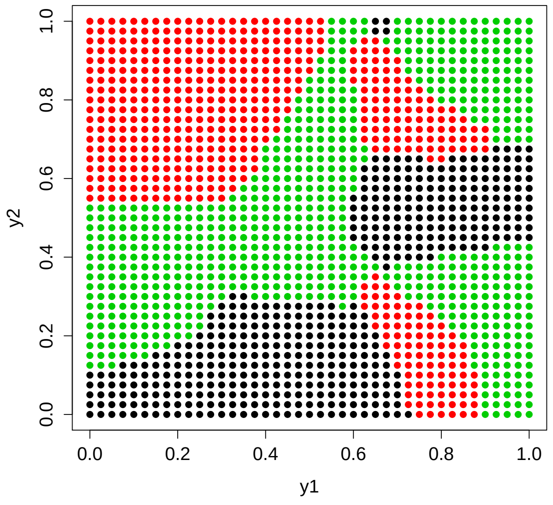
Figure 19: The error rates computed by the SCG learning algorithm for the training set (black) and for the test set (blue) versus the number of nodes in a single hidden layer. Training was done on 500 samples for the 9 class problem in 2 dimensions. Best results were chosen from 5 trials. The test set contained 5000 samples from the same distribution.

3.4 Complexity

It is not necessary to have many classes or high dimensional spaces in order to create a challenge for the artificial neural network. For example, consider the 3 class model in Figure 20a. This data was produced by creating 15 centers and assigning each of the centers to one of 3 classes. The data samples were assigned the class corresponding to the nearest center. The neural network used 1000 training samples and required about 25 hidden nodes in order to get a good fit (see Figure 20b). After 2000 iterations of the SCG algorithm, the model achieved an accuracy of 0.998 on the training set and 0.9668 on the test set.



(a) Distribution of 5000 test samples for a 3 class problem.



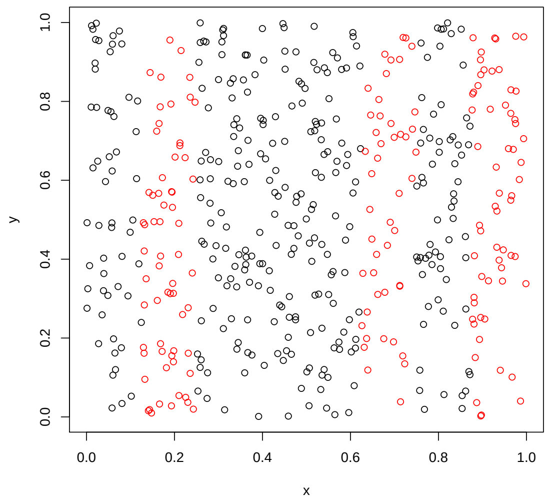
(b) Neural net model computed from 1000 independent training samples for the model in the previous figure.

It is even more revealing if we analyze the 1-dimensional case. Though it is unlikely that one would think of using the multilayer perceptron to classify samples with only one feature, it makes an excellent example for analyzing the way the classifier works. Furthermore, the convergence algorithm is very fast and simple. The input data is restricted to a set of points in a line segment x = [0.0,1.0]. The line segment is split into eighths and each eighth is assigned to one of two classes: black and red. Figure 21a shows the distribution of the samples. For purposes of visual representation, the samples are shown in a 2D plot where the y position is uniformly distributed between 0 and 1.0; however, only the x coordinate is fed into the classifier. The question is how many hidden nodes do we need to handle this data. The right half Figure 21b shows what happens when we do not have enough nodes in the hidden layer. With 5 hidden nodes in a single layer an accuracy of 99 percent was achieved on the test set. A hidden node was required for each boundary between the classes.

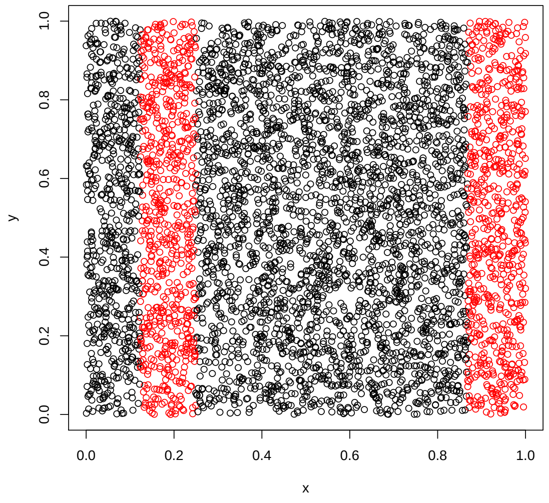
3.5 Beyond Two Dimensions

Most real applications deal with higher dimensional data, so it is appropriate to explore the effect of expanding the data space to more than 2 dimensions. We shall consider several approaches.

In many situations the intrinsic dimension of the data is low but the data is embedded in a



(a) 500 training samples for evaluating the neural net model.



(b) 5000 test samples for evaluating the neural net model with 3 hidden nodes in a single layer.

higher dimensional space. For example, many of the data parameters could be highly correlated. To explore this we introduce new parameters, x3, x4, x5, and x6 which are derived from the x1 and x2 parameters. For example, they can be obtained by applying rotation transformations on x1 and x2. Thus the x3 coordinate could be x3 = cos(theta)*x1 + sin(theta)*x2 where theta is any angle. The net result is that input data lies in a plane tilted in 6-dimensional space and is basicaly still 2-dimensional. It is not necessary to restrict ourselves to a linear transformation. For example, x3 could be x2 squared, x3 could be sqrt(x1), x4 could be sin(x1), etc. The data now lies on a 2-dimensional surface in 6-dimensional space. It is important to ensure that all the new parameters remain in the range of 0 to 1 so that the training algorithm does not give them unequal weights. Results Table 3

D	SSE	training set accuracy	test set accuracy
2	3.45	0.994	0.978
3	1.08	1.000	0.984
4	1.65	1.000	0.982
5	0.46	1.000	0.983
6	0.47	1.000	0.977

Table 3: Accuracy versus dimensions of sample space for the 5 class problem with added correlated input variables, 500 training samples, mlp algorithm with SCG with, 6 hidden nodes and 1000 iterations. Best results out of 3 trials.

We did not need more hidden nodes when we expended the dimension of the space in this manner. The regions are now separated by hyperplanes, but 6 hyperplanes were still sufficient. The training and test data error rates were not affected by this reconfiguration.

In the next case we considered the situation where an input parameter contains noise and no discrimination information. This is also easy to simulate. The x3, x4, etc. were set to random noise spread over 0 to 1. Results are shown in Table 4.

D	SSE	training set accuracy	test set accuracy
2	3.45	0.994	0.978
3	0.83	1.000	0.977
4	0.09	1.000	0.962
5	0.26	1.000	0.953
6	0.04	1.000	0.956

Table 4: Accuracy versus dimensions of sample space for the 5 class problem with added noisy input variables, 500 training samples, mlp algorithm with SCG with 6 hidden nodes and 1000 iterations. Best results out of 3 trials.

Finally, we can consider the case where the data is intrinsically of a higher dimension and all parameters are contributing equal information for classifying the samples. Results are shown in Table 2. The number of hidden nodes needs to be increased somewhat. The test set accuracy also decreases somewhat for the 500 training samples.

4 Categorical Data

Most real applications deal with categorical data rather than numerical. Unfortunately it is not meaningful to restrict the analysis to two dimensions and we therefore lose the ability to visualize the situation. We will still restrict our examples to perfectly separable classes.

Let us assume that we have 16 binary features which assume the value of either 0 or 1. We can string these features together to form a number between 0 and 32767. These numbers are assigned to one of two classes '1' and '2' by following some rule. It turns out that even this model can have all sorts of interesting structures which can be very challenging to a neural network.

In order to understand what is happening, we must remember that these 16 features form a 16-dimensional space where the value along each of the 16 axes can take only one of two values: 0 and 1. It is hard to visualize a 16-dimensional space, so let us start with a 2-dimensional space where there are only two possibilities. This forms a square which is split evenly into 4 sub-squares. Let us number the squares as 1,2,3,4 with 1 and 2 on the top row and 3 and 4 on the bottom row. A feature vector (0,0) will address sub-square 1; a feature vector (1,1) will address sub-square 4 which is on the immediate diagonal. If we go to a 3-dimensional cube divided into 8 sub-cubes, then (1,1,1) will go to the last sub-cube on the diagonal from (0,0,0).

Now sub-cube (0,0,0) shares a common face with sub-cube (1,0,0) and also with (0,1,0) and (0,0,1). Sub-cube (0,0,0) shares common edges with (0,1,1), (1,0,1) and (1,1,0). Finally (0,0,0) shares only a point with (1,1,1). This model extends to higher dimensions but now we call them hypercubes and hyperspaces. Keep this model in mind when we start looking at interesting examples.

The example here is a two class problem with 12 binary features. Thus the feature space is a 12-dimensional hypercube, where each 2-dimensional projection is a 2 by 2 cube. There are 4096 possible distinct feature vectors.

Getting back to our 12 feature example, we shall investigate using various rules for assigning the hyper-sub-cubes to one of two classes and how this impacts the neural network. To get back to our binary number representation, all odd numbers go to class 1 and all even numbers go to class 2. This turns out to be a very easy structure for the neural network. We make a training set by picking 500 random samples (with replacement) from the 4096 possibilities and train the classifier. A test set of 5000 random samples is created in a similar way. The resulting classifier with no hidden layer usually achieves perfect accuracy on both the training set and test set. It is easy to see why. The rule splits the hypercube in half along one hyperplane corresponding to feature 1. Since all the subcubes in each hypercube belong to the same class, the neural net has no difficulty generalizing to the entire test set.

Now let us change the rule so that the binary number is mapped to class 2 when it is divisible by 3 instead of 2. The structure of the data is more complex and some hidden nodes are needed in order to handle the data (see Table 5). Furthermore, 500 training samples is insufficient to cover all the variations and the neural network does not generalize for the 5000 test samples. When the training set is expanded to 2500 samples, the test set accuracy improves.

This model does not scale up very well when the binary vector is extended to 16 bits. The complexity of the data and the number of distinct binary vectors increases to 65536. As seen in Table 6, even 2500 training samples is inadequate. The test set accuracy actually decreases when more hidden nodes are introduced.

Here is the code for creating the binary data.

```
number2binary = function(number, noBits) {
        binary_vector = rev(as.numeric(intToBits(number)))
        if(missing(noBits)) {
           return(binary_vector)
        } else {
           binary_vector[-(1:(length(binary_vector) - noBits))]
        }
     }


# create random binary vectors of length Dim and
# classify them according to this rule. Concatenating
# the Dim bits into an integer s, classify the vector
# as class 2 if s %% jump == 0 otherwise as class 1.
# For example if jump = 2, then the least significant
# bit of s determines whether s is class 1 or 2. If
# jump = 3, the classification is very complex.
make_random_binary <- function(jump) {
for (i in 1:synthetic.size) {
 s <- sample(0:2^Dim,1,replace=TRUE)
 binv <- number2binary(s,Dim)
 if (i == 1) {
   data.samples <<- matrix(t(binv),ncol=Dim)
   } else {
   data.samples <<- rbind(data.samples,binv)
   }
```

```
  if ((s %% jump) == 0) {
    classid <- 2
    } else {
    classid <- 1
    }
  data.class <<- c(data.class, classid)
  }
}
```

Training set size	hidden nodes	SSE	training set accuracy	test set accuracy
500	4	65.5	0.922	0.841
500	8	48.7	0.948	0.619
500	16	4.00	0.996	0.765
500	24	4.00	0.996	0.607
500	32	2.00	0.998	0.679
2500	4	258.	0.944	0.926
2500	6	139.	0.965	0.966
2500	8	0.0	1.000	0.997
2500	16	0.0	1.000	0.996

Table 5: SSE, Training and Test Set Accuracy for neural network as a function of training set size and number of hidden nodes in a single layer. These are the best results for 3 trials using 1000 iterations. 12 binary features were used to classify samples to one of two classes. The samples were generated by assigning them to class 2 if the number formed by concatenating all the binary features to a number is divisible by 3, otherwise to class 1. Observe that the test sample accuracy was low when the training set size was 500 samples.

Training set size	hidden nodes	SSE	training set accuracy	test set accuracy
500	4	109.7	0.872	0.582
500	8	34.3	0.966	0.564
500	16	8.0	0.992	0.552
500	24	6.0	0.994	0.541
500	32	12.0	0.988	0.562
2500	4	514.0	0.847	0.811
2500	6	349.2	0.920	0.898
2500	8	804.5	0.793	0.629
2500	16	459.2	0.908	0.637
2500	24	290.5	0.942	0.627
2500	32	205.3	0.959	0.604

Table 6: SSE, Training and Test Set Accuracy for neural network as a function of training set size and number of hidden nodes in a single layer. These are the best results for 3 trials using 1000 iterations. 16 binary features were used to classify samples to one of two classes. The samples were generated by assigning them to class 2 if the number formed by concatenating all the binary features to a number is divisible by 3, otherwise to class 1. When the dimension of the binary data was increased to 16, there were 65536 possible vectors in the space; the complexity of the data also increased by the same factor; the training set was inadequate; and the test set accuracy did not improve with more hidden nodes.

The above example illustrates what happens when the data is mainly unstructured. In our final example, 4 binary features are used to classify the nodes in a form of a binary decision tree of depth 4. Let us take the sum of the first 4 features and classify the vector to class 1 or 2 based on whether the sum is even or odd. The remaining 12 binary features do not contain any information on how to label the data. How many hidden nodes do we need? Table 7 indicates that exactly 6 hidden nodes are required. Perfect accuracy was achieved for both the training set and test set. Even though the neural network was able to handle this model, it is still quite challenging for the binary decision tree. More advanced methods such as bagging, boosting, and forest methods [21] are necessary to handle data arising from this model. The application of binary decision tree algorithms is a topic for a future study.

hidden	SSE	training error	test error
1	397.7	0.712	0.681
2	91.2	0.947	0.938
3	88.0	0.947	0.938
4	89.8	0.947	0.938
5	60.3	0.947	0.938
6	0.0	1.000	1.000

Table 7: Number of hidden node, SSE, Training Set Accuracy, and Test Set Accuracy for 16 binary features mapped into 2 classes, using a depth 4 decision tree. The neural net was trained on 1000 samples. Best results for 3 trials are given. It was noted that convergence frequently occurs with less than 200 iterations.

5 Implementing mlp predict in R

It is fairly easy to implement the predict function in R using matrix algebra. For illustration, assume a 3 layer neural network where each layer has exactly 2 nodes. Thus there is one hidden layer with two nodes and the input and output also have two nodes each. RSSNS has a function weightMatrix(nn) which returns the weighting matrix for the neural network model nn. For the above model the weighting matrix resembles the following.

$$\mathbf{W} = \begin{bmatrix} 0 & 0 & xx & xx & 0 & 0 \\ 0 & 0 & xx & xx & 0 & 0 \\ 0 & 0 & 0 & 0 & xx & xx \\ 0 & 0 & 0 & 0 & xx & xx \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{2}$$

where xx are the weighting factors. We shall demonstrate the predictor for a single input vector consisting of two components. In order to apply the weighting matrix to the vector, it is necessary to extend it to 6 components by adding zeros. Suppose the input data is (x_1, x_2), we create the vector $\mathbf{x = (x_1, x_2, 0, 0, 0, 0)}$. To simulate the neural network, we compute

$$h = x * W \tag{3}$$

where * denotes matrix multiplication, then add the bias vector and apply the activation function to all the nonzero components of y. This returns a new vector containing the activations in the hidden layer which looks like $h = (0, 0, h_1, h_1, 0, 0)$. Repeating this process on h, the output values

$$y = h * W \tag{4}$$

appear at the tail end of the vector $y = (0, 0, 0, 0, y_1, y_2)$.

It was fairly easy to get the weighting matrix using the weightMatrix(nn) function. Unfortunately getting the bias vector requires more work. Here is the R code that I used.

```
bias <<- c() #biases for activation for all nodes
sObj <<- nn$snnsObject #where nn is the neural net model returned by mlp()
nunits <<- sObj$getNoOfUnits() # number of nodes
for (i in 1:nunits) {
     bias <<- c(bias, sObj$getUnitBias(i))
     }
```

6 Other Classification Methods

6.1 Quadratic Discriminant Analysis

The linear and quadratic discriminant analysis functions are both included in the MASS library. The linear discriminant analysis (lda) was not appropriate for this application because the class statistics did not share a common covariance matrix. The qda function was applied on the training set of 500 samples and was evaluated on 5000 test samples for various intrinsic dimension spaces. Results shown in table 8, indicate that the performance is somewhat inferior to the RSNNS network.

Dimension of space	Training Set Accuracy	Test Set Accuracy
2	0.968	0.949
3	0.944	0.929
6	0.942	0.888

Table 8: Quadratic Discriminant Analysis classifier results for the 5 class problems in various dimensions.

6.2 K Nearest Neighbours

The k nearest neighbours classifier (k-nn) is the simplest of all classifiers. Given an unknown test sample, the classifier searches for the closest k neighbours in the training sample and assigns the test sample the class number which is the majority of these neighbours. Figure 22 demonstrates the classifier for a neighbourhood of 3. The accuracy of the k-nn classifier for this data is shown in table 9. The classifier does not perform as well as the neural network which assumes linear boundaries between the classes.

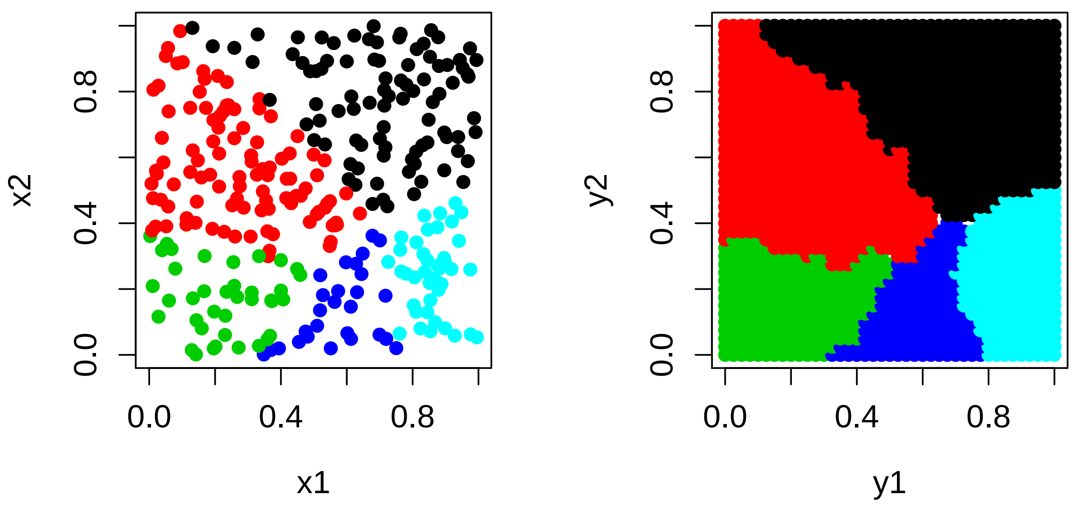


Figure 22: Results of applying the nearest neighbour classifier (right) applied on a training set consisting of 300 samples (left).

Training Set Size	knn 1/1 accuracy	knn 2/3 accuracy
300	0.9512	0.9438
500	0.9630	0.9640
700	0.9676	0.9678
900	0.9710	0.9706
1100	0.9704	0.9754
1500	0.9750	0.9774
2000	0.9764	0.9774
2500	0.9814	0.9792
3000	0.9820	0.9802
4000	0.9850	0.9856
5000	0.9884	0.9872
6000	0.9896	0.9890
7000	0.9894	0.9890

Table 9: k nearest neighbour accuracy versus training set size. First column uses first nearest neighbour. Second column uses majority of 3 nearest neighbours.

6.3 Decision Trees

The binary decision tree [22] is another successful classification scheme. There are several R packages available which can be applied to data. The decision tree is typically applied to nominal data; however, it can be adapted to handle continuous variables. Figure 23 is an example of how the tree would handle such data. The decision tree was applied to the training samples shown in Figure 6 using the rpart package. Decision trees require large training samples in order to build them to a sufficient depth. If one creates a forest of decision trees and use a voting scheme, it is possible to improve their accuracy. This is a subject of a separate reseach note.

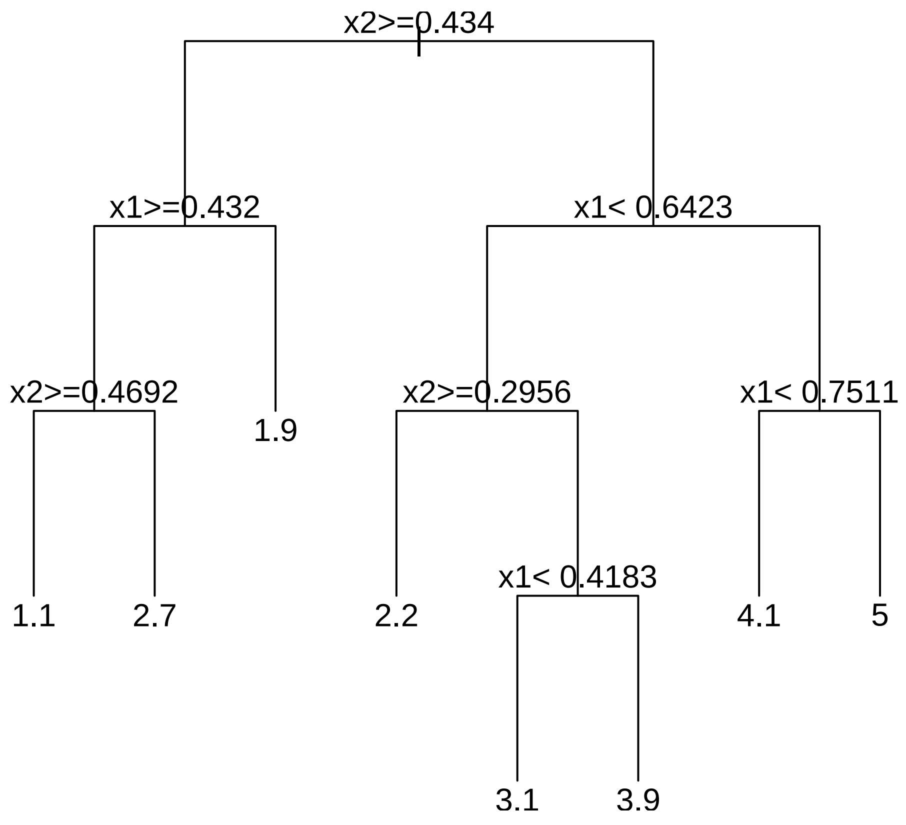


Figure 23: Sample binary tree created by rpart package for 5 class problem shown in Figure 6.

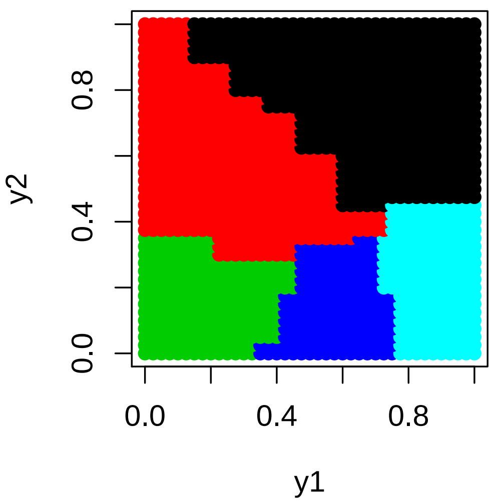


Figure 24: Results of applying the binary decision tree classifier created by the C5.0 package on 1000 sample training set.

Package	Training Set Size	Training accuracy	Test Accuracy
C5.0	200	0.980	0.9134
C5.0	500	0.992	0.9370
C5.0	1000	0.992	0.9464
C5.0	2000	0.998	0.9614
C5.0	5000	0.997	0.9784
party	200	0.890	0.8128
party	500	0.926	0.8738
party	1000	0.964	0.934
party	2000	0.960	0.9336
party	5000	0.974	0.9626
rpart	200	0.875	0.8088
rpart	500	0.892	0.8498
rpart	1000	0.894	0.8732
rpart	2000	0.889	0.8710
rpart	5000	0.881	0.8814

Table 10: Performance of various decision tree algorithms using their default settings. For C5.0 trials = 1.

7 Conclusions

In order to gain this insight, we applied the neural network to synthetic data which is easy to visualize. It is hard to draw specific conclusions, since we cannot generalize from specific examples without doing a rigorous mathematical analysis. Nevertheless, the examples demonstrate many of the interesting characteristics of neural networks and machine learning.

R comes with a choice of several packages for working with neural networks. I have found the RSNNS package to be the most flexible for the analysis described in this text.

Though the SIMBRAIN program [19] is not an R package, it also deserves special notice on account of its excellent user interface and the many YouTube instructional videos available. The program allows one to study the many characteristics of neural networks.

References

[1] Neural network applications. http://www.alyuda.com/products/forecaster/neural-network-applications.htm.

[2] B.D. Ripley. *Pattern Recognition and Neural Networks.* Cambridge University Press, Cambridge, U.K. CB2 2RU, 2005.

[3] Machine learning repository. http://archive.ics.uci.edu/ml/.

[4] Christoph Bergmeir and Jose M Benitez. Neural networks in r using the stuttgart neural network simulator: Rsnns. *Journal Of Statistical Software*, 46(7), 2012. http://jstatsoft.org/issue/view/v046.

[5] Brett Lantz. *Machine Learning with R, Second Edition.* Packt Publishing, Birmingham - Mumbai, 2015.

[6] Chiu Yu-Wei. *Machine Learning with R Cookbook.* Packt Publishing, Birmingham - Mumbai, 2015.

[7] Dr. Mark Gardener. *Beginning R, the Statistical Programming Language.* John Wiley & Sons, Inc, Indianapolis, 2012.

[8] Johannes Ledolter. *Data Mining and Business Analytics with R.* John Wiley & Sons, Inc, Harboken, NJ 07030, 2013.

[9] Ozgur Ergul. *Guide to Programming and Algorithms Using R.* Springer, London, 2013.

[10] Nina Zumel and John Mount. *Practical Data Science with R.* MANNING, Shelter Island, N.Y., 11964, 2014.

[11] Paul Teetor. *R Cookbook.* O'Reilly Media, Inc., Sebastopol, CA 95472, 2011.

[12] Dan Toomey. *R for Data Science.* Packt Publishing, Birmingham - Mumbai, 2014.

[13] Winston Chang. *R Graphics Cookbook.* O'Reilly Media, Inc., Sebastopol, CA 95472, 2012.

[14] Joseph Adler. *R in a Nutshell, 2nd Edition.* O'Reilly Media, Inc., Sebastopol, CA 95472, 2012.

[15] Robert I. Kabacoff. *R in Action, 2nd Edition.* MANNING, Shelter Island, N.Y., 11964, 2015.

[16] Norman Matloff. *The Art of R Programming.* No Starch Press, Inc., San Francisco, CA. 94103, 2011.

[17] Sarah Stowell. *Using R for Statistics.* Apress, 2014.

[18] No free lunch theorems. `http://www.no-free-lunch.org/`.

[19] Jeff Yoshimi. Simbrain. `http://www.simbrain.net/`.

[20] David Kriesel. *A Brief Introduction to Neural Networks.* self. `http://www.dkriesel.com/en/science/neural_networks`.

[21] Trevor Hastie Gareth James, Daniela Witten and Robert Tibshirani. *An Introduction to Statistical Learning with Applications in R.* Springer New York Heidelber Dordrecht London, 2013.

[22] Micheline Kamber Jiwaei Han and Jian Pei. *Data Mining Concepts and Techniques 3rd Edition.* Morgan Kaufmann Publishers, 225 Wyman Street, Waltham, MA 02451, U.S.A., 2012.