Journal of Machine Learning Research Format (2019) 7 Submitted 10/11; Published 10/11

# CSCI 447: Machine Learning – Project 3

**Peter Ottsen** PETER.K.OTTSEN@GMAIL.COM
**Bruce Clark** BRUCEWESTONMT@GMAIL.COM
**Forest Edwards** FORESTJEDWARDS@GMAIL.COM
**Justin McGowen** MCGOWEN.JUSTIN.P@GMAIL.COM

**Editor:** None

## Abstract

We were tasked with implementing two networks, Multilayered Perceptron (D. Montana, 1989) and the Radial Basis Function (Renals, 1989). These networks needed to performed classification and regression on five gold standard datasets: the abalone dataset (Paulo Cortez, 2009), car dataset (Bohanec, 1997), forest fires dataset (Cortez, 2007), machine dataset (Phillip Ein-Dor), image dataset (Vision Group, 1990), wine dataset (Paulo Cortez, 2009). We designed our MLP to take an arbitrary number of inputs, and then have an arbitrary number of hidden layers, with an arbitrary number of hidden nodes in each of those layers, leading to an appropriate output layer where we can compare against the actual data to performed classification and regression. We then perform a feed forward alogorithm and calculated error. After this has occurred, the network backpropagates values for adjusting the weights based off the calculated error. Our RBF network will also take an arbitrary number of inputs and then return the output of a RBF to determine how similar an input is to examples the network has seen. This is used to perform regression and classification to determine the models effectiveness.

In this paper, we tested these networks with different number of hidden layers and different numbers of hidden nodes inside our MLP. The data we trained our RBF network with, which determines the hidden layer size, was based on the outputs of three different KNN reduction/clustering methods.

Our results varied significantly from one datset to another. However, we saw very consistent results within the datasets themselves regardless of the network implemented. Our MLP performed noticeably worse with zero layers and consistently with one and two layers. This leads us believe, that with the proper amount of hidden nodes, the number of hidden layers was less important as long as hidden layers existed. Our RBF network had consistent results across runs using training data generated through Edited KNN (Kazuo Hattori, 2000), k-means (Paul Bradley, 1998), and k-medoids (Hae-Sang Park, 2006).

In this paper, we further discuss the algorithms, experimental approach, results of our implementation of these two algorithms.

## 1. Problem Statement

Finding patterns in data can be a very useful skill, and it can be applied to any field of research. The human mind is well capable of looking at simple datasets and easily recognizing patterns between two features. When presented with datasets with a large number of features, it gets exponentially harder for us to identify patterns. It is often difficult to program a computer to recognize these complex relationships as well. Neural networks are a tool for doing this. It takes in a set of data, and through a long trial-and-error

process known as 'training', it is able to find patterns in complex datasets. This training involves the network making predictions on sets of data and then determining how far off its prediction is so that it can tune the components of the guess to better predict in the next iteration. A trained network should be able to take a data observation and make a classification guess with high accuracy based on the information it was trained on.

## 2. Description of Networks implemented

### 2.1 Multi-Layer Perceptron

The Multi-Layer Perceptron is a feed forward network that uses backpropagation to train. This network takes a number of features and feeds the values of those features forward through an arbitrary number of layers with weights and output node values that are the network's guesses corresponding to what class or value the data point should be. The feed forward equation for calculating layer values is below:

$$Layer_{i+1} = WeightMatrix_i * Layer_i$$

The first $Layer_i$ of our network will be the inputs and the final layer will be the guess of the network (output). The nodes of each layer are then scaled between zero and one before the values are fed forward to the next layer with the sigmoid funtion below:

$$S(x) = \frac{1}{1+e^{-x}}$$

where $x$ is the unscaled node value.

The basis of training a MLP is first to feed forward our data through the array using the process above and then determine the error of the network prediction by comparing to the actual classes. We attempt to reduce the error in the next iteration of our model by feeding back weight adjustments. Gradient descent is used to try and minimize the square of the error (J. Solem, 2005). There is a lot of calculus that goes into this, but in common terms, the gradient of our function would translate to how fast the error of our model is increasing at any given point. To adjust our weight matrices, we move them in the direction of the negative gradient to find the minimum of our cost by iterating through our training data. The gradient, G, at a specific layer in our network is described as:

$$\triangle WM_i = LR * Error * Regularizer * LayerOutput_{i-1}$$

where $WM_i$ is the current weight matrix, $LR$ is the learning rate constant (tuning parameter that scales the size of the $\triangle WM_i$ step), $Error$ is of the current layer, $LayerOutput_{i-1}$ is the output values of the previous layer, and the $Regularizer$ is specified below.

$$Regularizer = LayerOutput_i * (1 - LayerOutput_i)$$

A momentum factor can also be used that takes into account the $\triangle WM_i$ of the previous iteration and scaling it by a "momentum factor" with the below equation:

$$WM_i = WM_i - (\triangle WM_i + MF * \triangle WMPrevious_i)$$

### 2.2 Radial Basis Function Network

The Radial Basis Function Network works essentially the same as the multi-layer perceptron but there is a single hidden layer and instead of the sigmoid function we used a radial basis function to scale the node values. The nodes of the hidden layer in RBN can be thought of as center points, where a distance(cost) is calculated for the input values to each node. This value is then scaled using a radial basis function(McCormick, 2013). For this assignment the below gaussian radial basis function was used:

$$Gaussian_{RBF} = e^{\frac{-||x-c_i||^2}{2*\sigma^2}}$$

where $x$ is the input value(s), $c_i$ is the center value(s) of one node, and $\sigma$ is the standard deviation for the center value(s) of the nodes. This is done for each node in the hidden layer. These node values are then fed forward through a weight matrix to find the output value(s). These output values are then compared to the actual values and the error is used to train to adjust the weight matrix using gradient descent (as described above):

$$\triangle WM_i = LR * Error * RBFLayerOutput$$

where $LR$ is the learning rate constant (tuning parameter), $Error$ of the layer , and $RBFLayerOutput$ is the output values of the hidden layer after the gaussian function. The weight matrices are updated using the $\triangle WM_i$ by the equation below:

$$WM_i = WM_i - \triangle WM_i$$

The training data is iterated through until the $\triangle WM_i$ approaches zero. This means our network is trained and ready to test.

## 3. Experimental Approach

In our experimental approach we focused on a few things:

1. How the number of layers in our MLP would effect performance.
2. How the number of nodes in those hidden layers would effect performance.
3. How will the edited, k-means, and k-medoids condensing algorithms comparatively perform on the MLP network.
4. How our RBF will perform when compared to our MLP given that it only has one hidden layer.

We hypothesize that:

1. The number of layers will affect our performance based on the number of inputs, outputs, noise, and size of dataset. For the MLP, we expect the two-layer model will perform the best. Followed by the single-layer model then the no-layer model. The no-layer model is expected to perform the worst when classes are not linearly separable. The two-layer model is expected to be better than the single-layers as two layers should better capture the non-separable aspect of the attributes. With respect to convergence, it is expected the two-layer and single-layer models will converge the fastest.

2. The number of nodes in the hidden layers would be expected to affect performance if too few or too many nodes are chosen. Too few nodes could cause the network to not accurately capture the different interactions of the attributes, while too many could cause over-saturation and increase runtime. We are choosing to use a number of nodes greater that our number of inputs in the hidden layers by a factor of 1.5.

3. Edited-KNN will perform well. However, edited will take a long time to train since the data may not condense as much as our other condensing algorithms. K-means and K-medoids should perform very similarly because they are functioning in similar ways.

4. Our RBF network will only contain one hidden layer of nodes. However, there will be as many hidden nodes as there are data points (from our condensed training data). This means there is ample opportunity for this network to train and adjust weights for the single layer network. We hypothesis that, despite only having one hidden layer of nodes, the RBF will be just as capable of training and predicting classification and regression problems, as our MLP with multiple layers.

Five fold validation was used and the networks were analyzed using f-score or mean squared error depending on the dataset. For our data sets with categorical classes, we calculate an f-score based on how frequently we made a correct guess on the class. To do this, we created a confusion matrix. To calculate the f-score from our confusion matrix, we do the following:

$$Fscore = 2 * \frac{precision/recall}{precision+recall}$$

$$precision = \frac{TruePositive}{TruePositive+FalsePositive}$$

$$recall = \frac{TruePositive}{TruePositive+FalseNegative}$$

Because confusion matrices are calculated on a per-class basis for multi-class problems, we computed precision, recall, and Fscore for each class and then used a macro - averaged f-score over the folds.

Some of our data sets have non-discrete class values. We used a mean squared error evaluation to measure the accuracy of our algorithms using these data sets. The mean squared error was calculated as such:

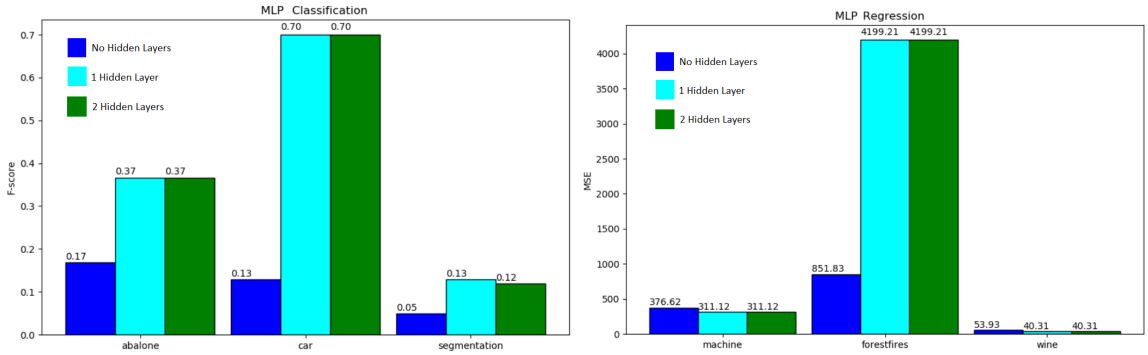
$$MSE = \frac{\sum_{j=1}^{n}(ActualResult_j - PredictedResult_j)^2}{n}$$

Mean-squared-error gives us a positive value that represents the average squared value between the actual and predicted value. Mean-squared-error penalizes more heavily for occasionally guessing values that are further away from the average than consistently guessing values that only vary slightly from the actual value.

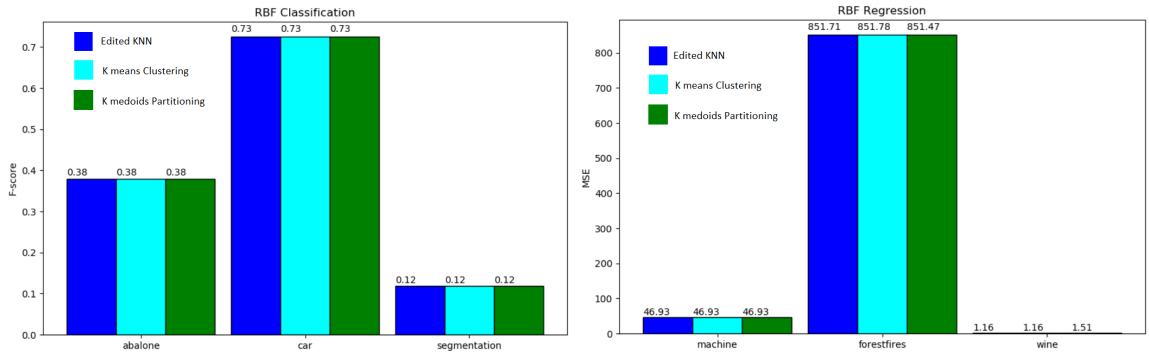
## 4. Results

For each network, we had mixed results. The distributions of our F-score and Mean Squared error were similar to the distributions that we observed in our last project using knn for classification and regression. This might be an observation on how predictive the datasets are since we are getting similar results with multiple algorithms.

For the multi-layered perceptron, we decided to use a number of nodes that were 1.5 times the size of the input layer. We didn't want to under size our hidden layers. So, we decided upon picking a higher number of hidden nodes than the number of inputs we had, since we always have fewer output nodes than that.




In our multi-layered perceptron, our model's hidden layer sizes varied per data set. For abalone, our hidden layers had 13 nodes, for car, they had 10, images(segmentation) had 30, machine(computer) had 15, forest fires had 19, and wine had 18.




We did some tuning on the number of hidden nodes to pick, but we may not have done enough tests to find the optimal level. We tested picking a number of hidden nodes to be between the number of inputs and outputs. In doing this, our evaluation for several of the datasets were worse than picking 1.5 times the input size.

For classification our MLP with no hidden layers performed pretty poorly for classification. We hypothesized this in our design document because the perceptron with no hidden layers cannot solve linearly inseparable problems. For regression, the forest fires data set performed better with zero hidden layers. This could be due to the type of problem we are solving, but we are not sure of why our MSE was so much lower in this case.

Our RBF network performed better than our MLP network. This might have been due to the knn algorithms we used on the front end of this network to pick the hidden nodes. This causes us to have tons of hidden nodes relative to the hidden layer in our single hidden layer MLP. Despite this, we have many vectors to compare a test vector to and will be able to find vectors that are 'close' to it. This may help us classify because now we are not purely basing on backpropogating weights, but also how close our test points are to some of our training data.

## 5. Discussion

The accuracy of the networks varied greatly over the data sets. This could be due to a few different factors:

First, because our nearest neighbor algorithm was classifying points in our datasets incredibly well (with an accuracy of 0.96-1.00), our edited KNN dataset did not discard many points at all. This left us with a huge dataset that was almost as big as the original dataset we put in. Feeding this many points into our RBN probably gave it too many datapoints and might have over fit the model. It was decided to take one fourth of the data points and use those as our nodes. This ended up performing similarly to training on the K-medoids and K-means outputs.

Second, adjusting to batch backpropagation could improve our networks. It was decided to run the backpropagation after every training data point because we hypothesized it would train the network more accurately. Looking back it seems that this may have caused over fitting and actually hindered our performance. In the future, we would like to test batch backpropagation. Batch backpropagation could cause our gradient to be more alike between iterations allowing us to better train the network and not oscillate back and forth between iterations.

Another potential area for optimization could be adjusting the number of nodes in our hidden layers for the MLP. We used a factor of 1.5 times the number of inputs. This could be further optimized as well and using a differing number of nodes between the hidden layers in the networks with two hidden layers.

Finally, tuning the network further could also improve the accuracy. We tried a few different values for our MLP's learning rate and momentum. In our MLP, we saw the best performance from a learning rate of 0.05 and a momentum of 0.1. For our RBN, we saw the best performance from a learning rate of 0.01.

Taking a step back and considering the last assignment with this one, there are very similar trends in accuracy based on the dataset being used. This could indicate a potential issue with the data. For the next assignment, it may be pertinent to process the data further.

## 6. Summary

In conclusion, our results varied significantly from one datset to another. However, we saw very consistent results within the datasets themselves regardless of the network implemented. Our MLP performed noticeably worse with zero layers and consistently with one and two layers. This leads us believe, that with the proper amount of hidden nodes, the

number of hidden layers was less important as long as hidden layers existed. Our RBF/RBN had incredibly consistent results when running through Edited KNN, K-Means Clustering, and K-Medoids Partitioning. Ultimately, we cannot explain why the differences are almost non existent. Our only guess is that our by running through our edited dataset, the radius of points are incredibly small and just happens to make the same predictions as K-Means and K-Partitioning which are similar.

Looking forward, we would love to test with more learning rates and momentums values. We would also like to try introducing bias into our MLP and also testing with more combinations of arbitrary hidden layers and hidden nodes within those layers.

## References

Marko Bohanec. Car evaluation database. 1997.

Paulo Cortez. Forest fires. 2007.

L. Davis D. Montana. Training feedforward neural networks using genetic algorithms. 1989.

Marine Resources Division. Abalone data. 1995.

Chi-Hyuck Jun Hae-Sang Park, Jong-Seok Lee. A k-means-like algorithm for k-medoids clustering and its performance. Technical report, 2006.

N. Overgaard J. Solem. A geometric formulation of gradient descent for variational problems with moving surfaces. 2005.

G. Krishna K. Gowda. The condensed nearest neighbor rule using the concept of mutual nearest neighborhood. Technical report, 1997.

Masahito Takahashi Kazuo Hattori. A new edited k-nearest neighbor rule in the pattern classification problem. Technical report, 2000.

C. McCormick. Radial basis function network. 2013.

Usama Fayyad Paul Bradley. Refining initial points for k-means clustering. Technical report, 1998.

Fernando Almeida Telmo Matos Jose Reis Paulo Cortez, Antonio Cerdeira. Wine quality. 2009.

Jacob Feldmesser Phillip Ein-Dor.

S. Renals. Radial basis function network for speech pattern classification. 1989.

James Givens Stefan Aeberhard, Michael Gray. A fuzzy k-nearest neighbor algorithm. Technical report, 1985.

University of Massachusetts Vision Group. Image segmentation data. 1990.