Learning with Neural Nets

Andrew Kirby andy-kirby@live.com
Kevin Browder browderkevin54@gmail.com
Nathan Stouffer nathanstouffer1999@gmail.com
Eric Kempf erickempf123@gmail.com

Abstract

Multilayer Perceptron and Radial Basis Function neural networks are supervised training methods used for classification or regression. While applicable in many machine learning problems, these networks require extensive tuning of parameters to perform optimally. When tuned appropriately, both networks are capable of equally high performance. As dimensionality of the data increases, the multilayer perceptron network is suspected to outperform the Gaussian radial basis function network in practice as the data becomes harder to cluster into a smaller number of centers.

1. Problem Statement

As a class of learners, Neural Networks gained traction early on in the Machine Learning community. However, research funding was cut short when no one discovered a method for Neural Networks to move out of the realm of linear problems. This all changed when the Backpropagation Rule was published. Since then, Neural Networks have become a pillar of the community and are used in a wide variety of applications from image recognition to sales forecasting. In general, these networks follow a similar format. They are composed of nodes that take in inputs with a corresponding weight and send an output to another node in the network (Svozil et al., 1997).

Of particular interest are Multilayer Perceptron (MLP) and Radial Basis Function (RBF) networks, common networks used for universal function approximation. Each network type will be constructed and tuned to handle the datasets, and their respective performances will be evaluated.

Given datasets, the task is to find a model that predicts a real value or class based upon a feature set. The abalone dataset contains features paired with the age of an abalone (the class). This set has 28 closely related classes, making classification potentially difficult. The car dataset contains attributes corresponding to 4 possible conditions of a car while the segmentation dataset details 6 subjects of outdoor photographs with sets of attributes. The rest of the datasets are regression. The forest fires set contains the area burned by a forest fire. Most of the values are 0 with a few large values which might give the model trouble. The machine dataset predicts the performance of a CPU which has an even distribution across its regression values. Wine quality contains the score of a wine with associated attributes. Several of the attributes are correlated, making regression more difficult (Dua and Graff, 2017).

1.1 Hypothesis

The RBF network is better suited to handle categorical data than a MLP network. This is because a RBF uses a distance metric to take into account how similar a query example is to the center of a cluster. Therefore, the RBF is expected to outperform the MLP network on categorical datasets. However, the MLP network is expected to perform well, with 2 layer MLPs generally underperforming comparatively, especially in the small datasets. This is because a 2 layer network requires more training examples than a network with fewer layers to produce equivalent results.

2. Algorithms

2.1 MLP Networks

The Multilayer Perceptron is a feed forward neural network that can be used for both classification and regression. At its core, the MLP is composed of an input layer, hidden layers, and an output layer. Each hidden layer contains a tunable number of hidden nodes and each node has an activation function. If the nodes in the hidden layer use a non-linear activation function, a MLP becomes a universal function approximator (Svozil et al., 1997).

A classification network uses an output layer with one sigmoidal neuron corresponding to each class and the prediction is chosen to be the class with the largest output value. A regression network has a single neuron in the output layer that uses a linear sum as its activation function. The output of this node is always chosen as the prediction. A MLP is trained via Stochastic Gradient Descent (see Section 2.3).

To use a trained MLP, begin with the input layer where each feature of a query example corresponds to a node. Note that categorical data is encoded into a set of k binary features where k is the number of attribute levels. The nodes are then fed forward to the first hidden layer, where the chosen activation function is applied to a linear sum. Feeding forward continues until the output layer is reached. Here, the appropriate value is selected and returned as the prediction.

2.2 RBF Networks

Radial basis function networks are three layer feed forward networks consisting of an input layer, a single hidden layer, and an output layer. The activation functions in the hidden layer are radial basis functions (RBFs)—functions whose output is determined by a *distance* from some *center*. The non-linear nature of RBFs makes the network a universal approximator, approaching an arbitrary level of performance given enough centers and appropriate standard deviations σ (Wu et al., 2012).

A popular RBF is the Gaussian, whose activation function G given a data point $\vec{x}$ follows the form

$$G(\vec{x}) = exp(\frac{-r^2}{2\sigma^2})$$

where r is the distance between $\vec{x}$ and some center $\vec{c}$, and σ^2 is a parameter treated as variance (Orr et al., 1996). The RBF networks discussed in this paper will use a Gaussian activation function with distance r evaluated using Euclidean distance.

A Gaussian RBF commonly utilizes clustering to find suitable centers. Edited Nearest Neighbor or Condensed Nearest Neighbor, KMeans, and Partitioning Around the Medoids clustering methods were used to determine RBF centers. Variance σ^2 of each center $\vec{c}$ returned by the clustering methods may be computed in a variety of methods, including calculating the variance among the k nearest neighbors of each $\vec{c}$.

A RBF network's output layer follows the same format as a MLP. Once a RBF network has been constructed with the appropriate $\vec{c}$, σ^2, and output nodes, the output layer is trained using gradient descent (see Section 2.3).

2.3 Gradient Descent

Recall that the performance of a Neural Network can be evaluated with an error function. Most commonly, this error function is chosen to be the Mean Squared Error of each of the output nodes (applied appropriately to classification and regression problems).

A given network N performs best when it's error function Err is minimized. At first, this seems like a simple minimization problem: take the derivative of Err and find the weights that produces the smallest output in Err. While that would be a good solution for a problem with small dimensions, the dimension of Err is same as the number of weights in the network, which can be a massive

number. This means that naive optimization methods will be too computationally expensive to find the global minimum of Err. Thus, another technique must be used. The technique used for training Neural Networks is Gradient Descent (GD).

Gradient Descent is an iterative process used to find a local minimum of a function. Note that only a local minimum is found, not a global minimum. This means the performance of a Neural Network trained with GD is not necessarily optimal. However, what is lost in performance is gained in training time, as Gradient Descent's search for a local minimum is far more efficient than the standard search for a global minimum.

For a function f, GD iteratively uses the following rule for finding a local minimum: $\Delta x_a = -\eta \frac{\partial f}{\partial x_a}$ where x_a is value of the Err function in the a^{th} dimension and η is a tuned parameter (also called the learning rate). GD has found a local minimum once all the values of x_a have converged. Applying this to Err, gives

$$\Delta w_{ji} = -\eta \frac{\partial Err}{\partial w_{ji}}$$

where w_{ji} is the weight from the i^{th} to the j^{th} node and η is the tuned learning rate. Using the chain rule, $\frac{\partial Err}{\partial w_{ji}} = \frac{\partial Err}{\partial net_j} * \frac{\partial net_j}{\partial w_{ji}}$. Note that $net_j = \sum w_{ji} * x_{ji}$ where x_{ji} is the i^{th} input to the j^{th} node.

Beginning with the simpler of the partial derivatives, $\frac{\partial net_j}{\partial w_{ji}} = x_{ji}$. The other partial derivative depends on two conditions: the activation function of a node and whether the node is in a hidden layer or an output layer. Let $A_j(net_j)$ denote the activation function of the j^{th} node, whether sigmoidal or linear. The value of the partial derivative also depends on the layer type of node j.

Let δ_j^o denote $\frac{\partial Err}{\partial net_j}$ for the j^{th} node in the output layer and δ_j^h denote $\frac{\partial Err}{\partial net_j}$ for the j^{th} hidden layer. Then,

$$\delta_j^o = -A_j'(net_j) * (d_j - o_j)$$

where d_j is the target of the j^{th} node. And,

$$\delta_j^h = A_j'(net_j) * \sum_{k \in ds(j)} \delta_k * w_{kj}$$

where $ds(j)$ returns the deltas of the nodes that the j^{th} node directly affects (also called the downstream nodes).

This sets up the Backpropagation Rule. This rule begins by computing each δ_j^o for each node in the output layer. Then update the weights that apply to the output layer according to $\Delta w_{ji} = -\eta * \delta_j^o * x_{ji}$. Now the weights going to the output layer have been updated and the network has the necessary values of δ_j to begin a recursive call (from the last to the first hidden layer) that updates weights by the following rule $\Delta w_{ji} = -\eta * \delta_j^h * x_{ji}$ using δ_j^h from above (Rumelhart et al., 1988).

A true Gradient Descent method would average $\frac{\partial Err}{\partial w_{ji}}$ over the entire training set. Stochastic Gradient Descent (SGD) selects a batch of random examples from the training set and averages $\frac{\partial Err}{\partial w_{ji}}$ over that batch. This method, while less stable, runs much faster than true GD (Svozil et al., 1997).

The final step in GD is to repeat batching and propagating the changes backwards until the weights in the network converge. Once the weights have converged, a local minimum has been found and the network is ready to use.

An option when computing GD is to include a momentum term. For a given iteration in GD, denoted $\frac{\partial Err}{\partial w_{ji}}^t$, momentum is $\alpha * \frac{\partial Err}{\partial w_{ji}}^{t-1}$ where $\alpha \in \mathbb{R}$, more specifically, $\alpha \in [0, 0.5]$. This gives the new update rule:

$$\Delta w_{ji} = -\eta \frac{\partial Err}{\partial w_{ji}} + \alpha * \frac{\partial Err}{\partial w_{ji}}^{t-1}$$

A momentum term is added for two reasons. First, momentum typically causes a network to train faster because each Δw_{ji} is a larger value (Rumelhart et al., 1988). The second reason is an attempt to find the best local minimum in a region. By moving in the direction of the previous iteration, GD may pass by a local minimum that does not offer as good of performance as another local minimum. However, as always, there is no free lunch and GD may well pass by a local minimum that performs better than the one it ends up finding.

3. Experiment

3.1 Preprocessing Choices

First, all the examples in the data set are randomly scrambled and then assigned to sets for ten-fold cross validation. All categorical variables are converted to integers and the preprocessor also generates a similarity matrix for each categorical variable that is used for determining distances between categorical variables. All numerical variables are normalized between 0 and 1. The data sets did not contain any missing variables.

3.2 Evaluation Metrics

Classification datasets are evaluated with accuracy and mean squared error (MSE). The MSE metric implemented measures the squared error between the predicted and actual class distributions. This is similar in concept to a Brier score but different in computation. Accuracy indicates how well the algorithm individually classifies examples.

To evaluate regression datasets, mean error (ME) and MSE will be used. MSE takes the distance between real and predicted values and squares it. ME is computed similarly, but will not square the difference. MSE emphasizes the effect of outliers while ME captures whether the learner tends to over- or under-estimate the values in the test set. MSE and ME are computed using z-scores (the number of standard deviations from the mean) so that comparisons can be made between datasets.

3.3 Tuning

3.3.1 RBF

Tuning a RBF network relies on finding the optimal centers, variances, and learning rate for a given problem. The centers were varied using the variety of clustering methods mentioned in Section 2.1. Variances were found by the variance of the k nearest neighbors of each center. The value of k was varied during tests but ultimately set to 25. Tuning found that varying k in this region did not significantly affect the performance of the RBF network as opposed to the clustering method or learning rate. Learning rate was tuned over a range of $1 * 10^{-4}$ to 5. The results of the tuning process are shown in Figure 1.

Learning rates greater than 0.01 are not included for the regression data sets. Saturation in output layer weights caused complications in the linear output's calculations. Thus, it was determined that low learning rates were necessary to prevent network saturation in these cases.

As seen, the best clustering method used to create the RBF network varied slightly for each dataset. The classification networks performed well over a wide range of learning rates, dropping in performance only as the learning rate became too slow. The regression networks appeared to favor

a midband of learning rates. The network did not learn enough at rates lower than $1 * 10^{-4}$ and experienced weight saturation at rates higher than 0.01.

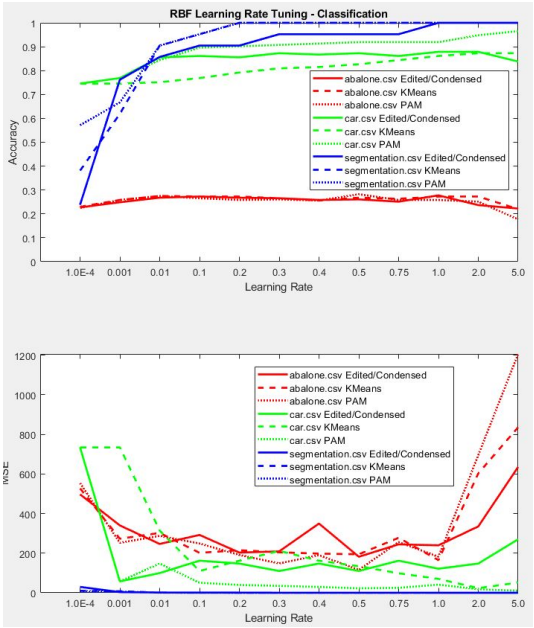

Figure 1: Results of tuning the RBF network on the classification sets. Color indicates dataset and line style indicates the clustering method used to find the centers.

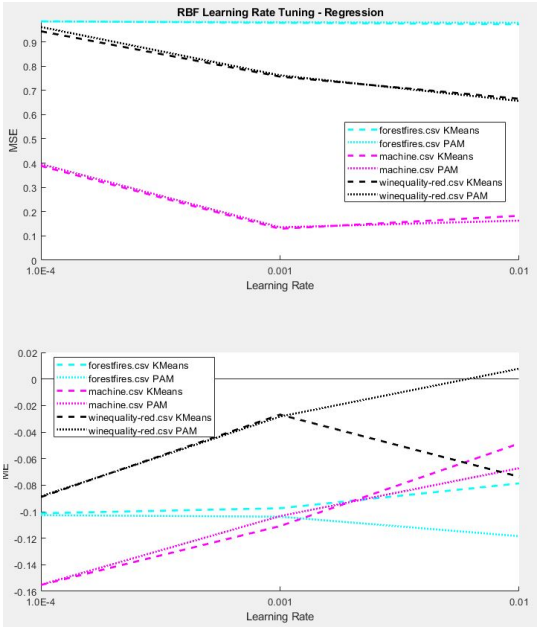

Figure 2: Results of tuning the RBF network on the regression sets. Color indicates dataset and line style indicates the clustering method used to find the centers.

Learning rates greater than 0.01 are not included for the regression data sets. Node saturation for learning rates greater than 0.01 caused complications in the linear output's calculations. Thus, it was determined that low learning rates were necessary to prevent node saturation in these cases.

As seen, the best clustering method used to create the RBF network varied slightly for each dataset. The classification networks performed well over a wide range of learning rates, dropping in performance only as the learning rate became too slow. The regression networks appeared to favor a midband of learning rates. The network did not learn enough at rates lower than $1 * 10^{-4}$ and experienced weight saturation at rates higher than 0.01.

3.3.2 MLP

Tuning began with learning rate, η, which is the step size at each iteration in GD. The following graphs plot the MSE and accuracy for classification as well as MSE and ME for regression. The graphs also show performance on different numbers of hidden layers ranging from 0 to 2. Number of hidden nodes was left at 2 times the number of attributes for the tuning of all other parameters.

Beginning with the classification data sets, the initial value was set to 0.1. The other values tested

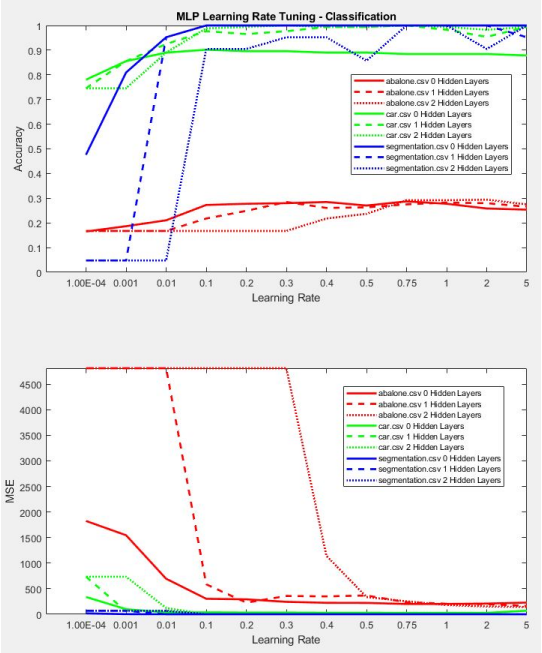

Figure 3: Output when Tuning Learning Rate on the MLP network for Classification. Line color indicates dataset and line style indicates the number of hidden nodes.

were: 0.0001, 0.001, 0.01, 0.2, 0.3, 0.4, 0.5, 1, 2 and 5. Learning rates below 0.1 performed the worst and rates above 0.1 did not see any added performance except on abalone where 0.2 drastically reduced the MSE. This is most likely because the abalone dataset contains significantly more points than the other data set, so, using the same number of iterations, it's learning rate must be larger to find the local minimum. On the regression data sets, the same learning rates were tested. Regression performed best with a learning rate of 0.1 on all of the data sets. Smaller learning rates performed significantly worse and larger learning rates have inconsistent results.

Another parameter must also be tuned: the momentum multiplier α. This value is tested at 0.5, 0.25, and 0.

On the abalone data set, MSE increased significantly when momentum was increased from 0.25 to

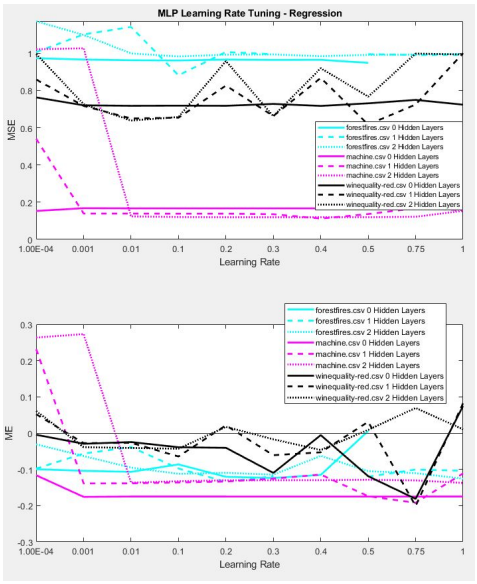

Figure 4: Output when Tuning Learning Rate on the MLP network for Regression. Line color indicates dataset and line style indicates the number of hidden nodes.

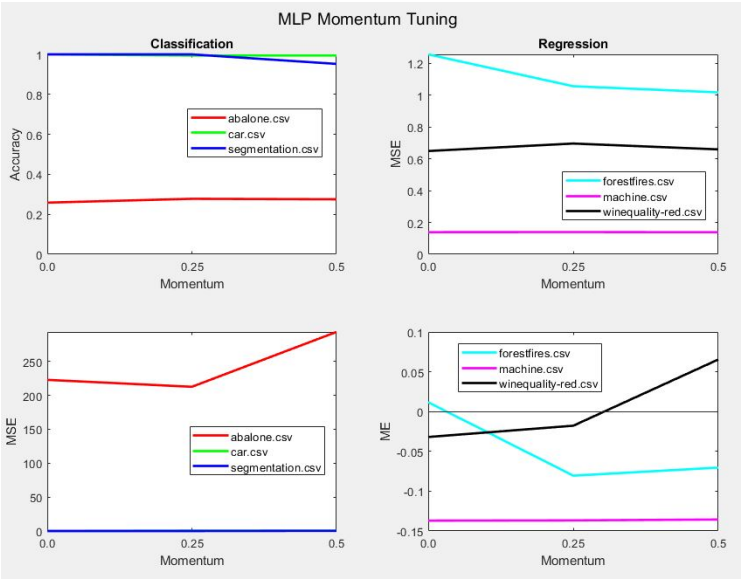

Figure 5: Output when Tuning Momentum on the MLP network. Line color indicates dataset.

0.5 but accuracy had no change. The other two regression data sets (forest fires and machine) had negligible differences in the MSE and accuracy when varying momentum. The regression data sets performed best with a momentum of 0.25. A larger momentum significantly increased ME on the wine quality data set while a smaller momentum significantly increased the MSE on the forest fires data set.

4. Results

The optimal tuning parameters corresponding to each network and dataset were then run for final comparisons. In general, the MLP and RBF networks performed similarly. The MLP network was slightly better at classification whereas the RBF network was slightly better at regression. The overall performance of these networks reinforces their effectiveness as universal approximators.

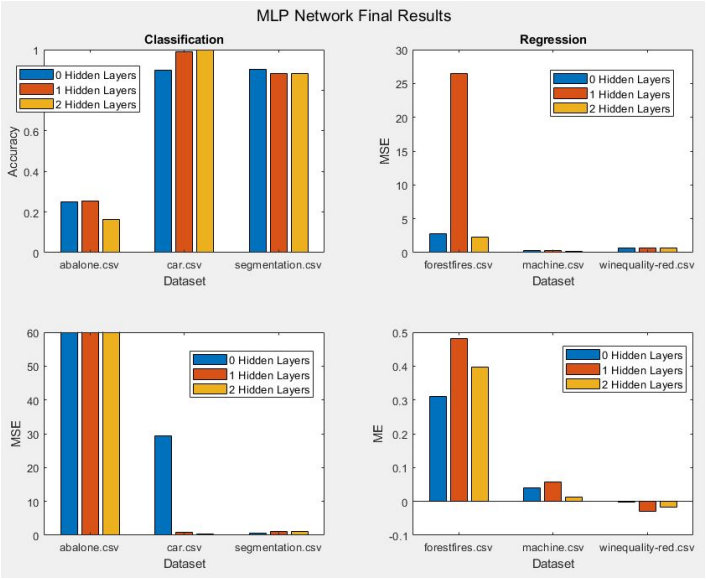

Figure 6: Final Results of the MLP Network. Color indicates number of layers.

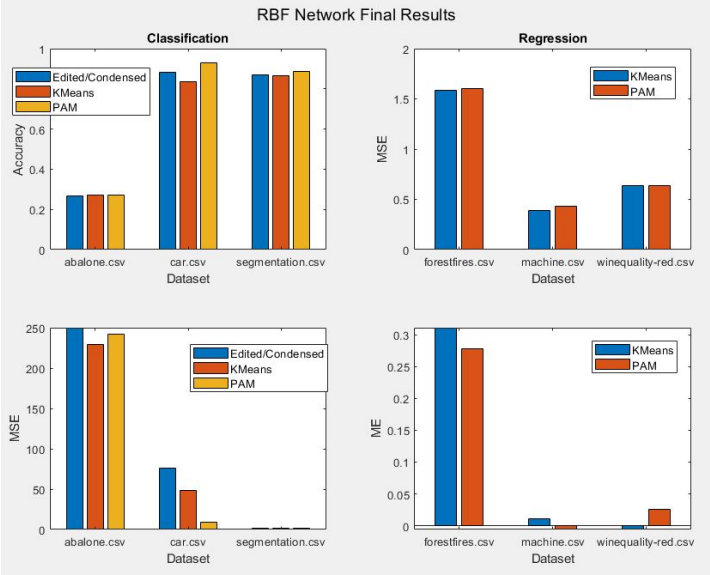

Figure 7: Final Results of the RBF Network. Color indicates number of layers.

The hypotheses that the RBF network would outperform the MLP network and that the 2 layer MLP network would struggle were not reinforced by the results. Possibly, both networks were equally

suited to handle these datasets. Normally, as dimensionality increases, MLP networks require more nodes in their hidden layers and more data to successfully perform gradient descent during training. Gaussian RBF networks are even more susceptible to this trend. They are localized, meaning that the network performs best when the input is near the predefined centers. As the number of dimensions increase, the number of RBF centers needed increase exponentially (Wu et al., 2012). Because the datasets used had relatively few features, the "curse of dimensionality" had yet to take effect on the MLP and RBF networks.

The RBF network had similar results to a K-Nearest Neighbor model, likely because each Gaussian RBF takes distance from a center into account. Additionally, each center was computed from a K-Nearest Neighbor variant.

5. Summary

Feed forward neural networks are powerful models with great versatility. Both MLP and RBF networks are capable of universal approximation. RBF networks train quickly and require less hidden nodes when when representative clusters may be extracted from the underlying distribution easily, whereas MLP networks excel without the need for clustering. These networks are not without flaws. Both networks require sufficient data and computational cost to train. Each network has several vital parameters that must be tuned, increasing the cost of training such models. Finding optimal parameters may be a challenge as dimensionality increases and the network grows in complexity.

References

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

Mark JL Orr et al. Introduction to radial basis function networks, 1996.

David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

Daniel Svozil, Vladimir Kvasnicka, and Jiri Pospichal. Introduction to multi-layer feed-forward neural networks. *Chemometrics and intelligent laboratory systems*, 39(1):43–62, 1997.

Yue Wu, Hui Wang, Biaobiao Zhang, and K.-L Du. Using radial basis function networks for function approximation and classification. *ISRN Applied Mathematics*, 2012, 03 2012. doi: 10.5402/2012/324194.