Forecasting Precipitation in Hanover, NH using Neural Networks

Chris Hoder
Ben Southworth

8 Mar. 2013

Abstract

Even professional precipitation forecasts have a remarkably low level of accuracy. Most of these forecasts are done using numerical modeling, but an alternative method is to train machine learning algorithms on historical weather and precipitation data, and use these results to make predictions based on the current weather data. In this paper we implement a feed-forward back propagation neural network to predict precipitation in Hanover, NH. We then compared these results with other algorithms including a radial basis function neural network, a random forest, k Nearest-Neighbors and Logistic Regression. First we examine the background and scope of our problem, followed by a description of data including the source, format and choice of variables. We then introduce the algorithms that we have implemented and discuss the results they produced. This work aims to improve on previous research into precipitation prediction by considering a more generalized neural network structure as well as a cascade classifier which first aims to predict either precipitation or no precipitation and then run a regression to obtain a precipitation amount. There was no external code used in this project.

1 Background

Meteorologists still lack the ability to produce truly accurate weather forecasts, especially with respect to precipitation. Nature's complex phenomena and the scope of the problem in both time and space leads to predictions that sometimes vary greatly from the actual weather experienced (Santhanam et all, 2011). Figure 1 (HPC,2012) is provided by the National Oceanic and Atmospheric Administration (NOAA) and demonstrates the accuracy of quantitative precipitation forecasts from three different professional prediction models: the North American Mesoscale (NAM), the Global Forecast System (GFS), and the Hydrometeorological Prediction Center (HPC). A threat score is a function that ranges from 0 to 1 and provides a measure of how accurate a prediction was, with 1 being a perfect prediction. The inaccuracy of these predictions provide motivation for developing a better algorithm for predicting precipitation.

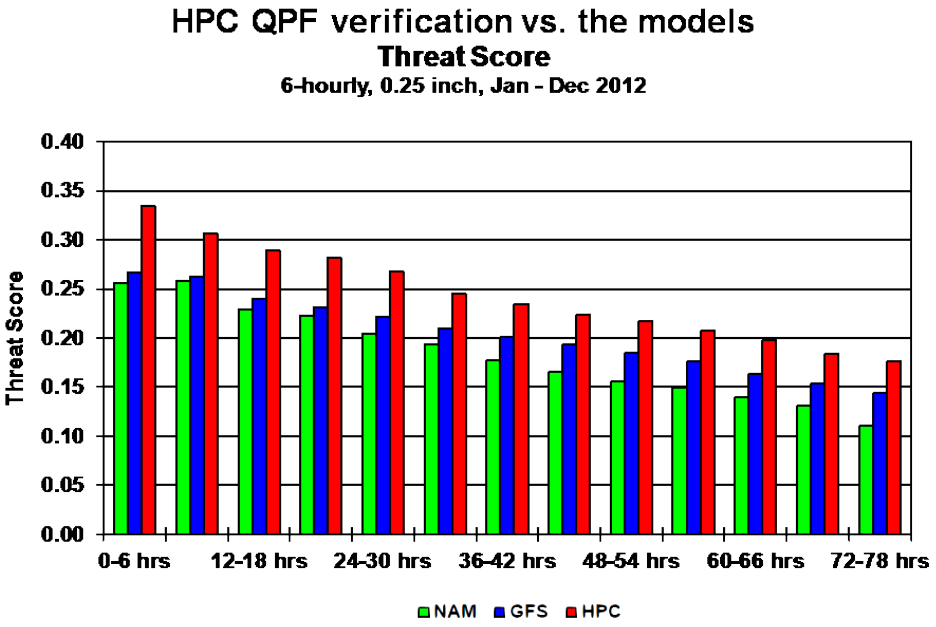


Figure 1.1

2 Problem/Scope

Our original goal was to predict snow in Hanover, NH. The data we found and are using however, turned out to be strictly in terms of hourly precipitation, and not in terms of accumulated snowfall. There was data on total snow coverage in kg/m^2 available, which obviously accounted for falling snow, but it also accounted for melting snow, ground temperature, etc. Because of this, we found it to be an inaccurate measure of how much snow accumulated due to a storm front, and chose to use the precipitation data for our project instead. We can still determine snow from rain based on ambient ground temperature, one of our input variables, but we cannot know the amount of snow which fell due to unknown variations in snow-water density ratio. Thus, while we can still focus on predicting snowfall, the algorithm prediction problem has been entirely focused on precipitation, with the assumption that the forces causing precipitation are the same in both cases. Ground surface temperature can then be used as an indicator of snow or rain. Note, we will still consider all precipitation events, regardless of snow or rain, in our data set during training. Our initial predictions were made over a 6-hour timeframe, after which we increased the timeframe to test our algorithms to forecast precipitation 12, 18 and 24 hours in advance.

3 State of the Art

Traditionally, weather forecasting is done via two different methods. The first and foremost is based on numerically modeling physical simulations with computers (Santhanam et all 2011). Although there has been much work and progress in this field, precipitation forecasts are still far from perfect.

Neural Networks have been used by meteorologists to accurately model many different weather phenomena such as humidity and temperature (Maqsood, et all 2004). However, these features are already fairly accurately predicted via physical simulations. Santhanam

et all, worked to predict rain as a classification problem using back propagation feed-forward neural networks as well as radial basis function networks. They produced a classification error rate of 18% and 11.51% for their feed-forward network and radial basis function respectively but consider only a simple two layer neural network architecture (Santhanam et all 2011). Other work has been done using neural networks to predict precipitation but they have limited themselves to simple network architecture's and very short prediction times (French et all, 1991), (Kuligowski et all, 1998), (Hall et all, 1998)

4 Data

All of our data was obtained from the Physical Sciences Division of the National Oceanic and Atmospheric Administration. Which weather variables are most relevant to snowfall and the physical distance of measurements from Hanover necessary to predict snowfall was determined through an interview with a meteorology Ph.D. candidate at the University of Utah. We concluded that on the earth's surface we should consider wind direction, relative humidity, and mean sea level pressure. At elevations (measured in pressure) 500mb, 700mb, and 850mb we should use wind direction, relative humidity, specific humidity and geopotential height. For these variables, predicting precipitation one day in advance can depend on variables up to and over 1,000 miles away. Thus we selected the location of measurements for our data to approximate a box surrounding Hanover of approximately 2,000 miles in width and height. Figure 2 shows this box, with the blue marker indicating the location of Hanover and red markers representing the location of variable measurements.

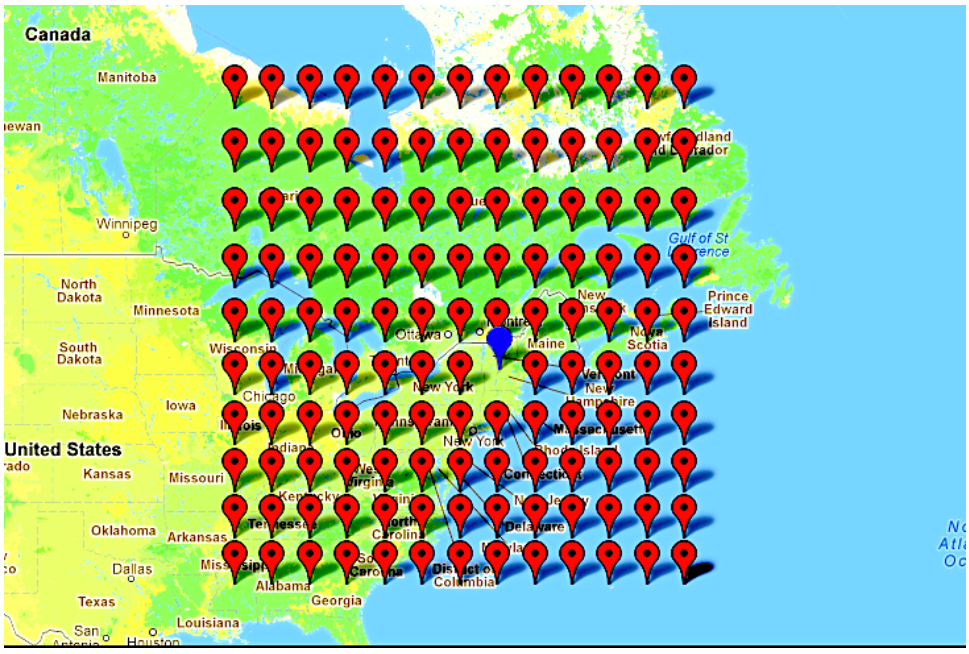


Figure 4.1

Once our source was selected, the first step in obtaining data was to download the files (∼50GB) and reformat them. The data was separated by variable and year, and formatted in .netCdf files, with one file corresponding to one year for one variable. Due to variable measurements spanning 1948-2001 on a 6-hour time interval and precipitation measurements in Hanover spanning 1979-present on hourly intervals, we had to choose

the overlapping period for our training data. This provided us with a total of 23 years of training data separated over 6-hour intervals, which corresponds to 33,600 individual samples. Using a 13×10 grid of measurements shown in Figure 2 for our 19 variables of choice (four on the surface, five at three different pressure levels) gave us 2,470 input features.

After downloading the data, we extracted the 3-dimensional matrix of measurements for each variable over the time frame and geographical spread in which we are interested. Each matrix had dimensions corresponding to latitude, longitude and time, with measurements for one year, or 1,460 (1,464 for a leap year) 6-hour time intervals. However, because we are considering each location of a variable's measurement to be a distinct feature in our algorithm, one matrix had one year of measurements for 130 of our 2,470 different features. For our training data, we then transformed and combined this set of matrices into 2-dimensional matrices with rows corresponding to time intervals, and columns to features, where each 2-dimensional matrix corresponded to one year of training data. When doing this we also reindexed the precipitation for each year so that the ith row of the precipitation vector was the sum of all precipitation in the next $n \cdot 6$ hours, where n is the number of 6-hour intervals ahead of time that we want to make our prediction. We created training matrices for 6, 12, 18 and 24 hour time-intervals.

Once we had training matrices for each year, we normalized each input variable independently to ensure that no single variable's magnitude was disproportionate to its relative importance. Specifically, for each value of a given variable, we subtracted the variable's mean and divided by the standard deviation. These recalculated variable values then have a zero mean and unit standard deviation over the entire data set. In order to attempt both classification and regression algorithms, we also created two different precipitation vectors for training data. One vector has the real-valued amount of precipitation in inches over the given time interval, and the other has boolean values, where y=1 if there was precipitation in the given time interval and y=0 otherwise. In order to have training data and test data to run our algorithms on, we split our data into a set of twenty years of training data and three years of test data. We also used current daily data for a period of several days to test our algorithm, but a larger set is necessary for feedback on the algorithm's error.

5 Algorithms

5.1 Neural Network

We have implemented a feed-forward back propagation neural network. The structure of the network has been generalized to allow for any number of hidden layers with any number of nodes in each layer. This means we have several hyper parameters that need to be tuned, such as the number of hidden layers and the number of nodes in each layer. The structure of the network can be seen in the figure below. Networks with only two hidden layers have been shown to be able to model any continuous function with arbitrary precision. With three or more hidden layers these networks can generate arbitrary decision boundaries (Bishop, 1995). Furthermore, the output of a neural network for a classification problem has been shown to provide a direct estimate of the posterior probabilities (Zhang, 2000). The ability to model arbitrary decision boundaries is of particular interest for us since since weather phenomena has been shown to be highly non-linear and dependent on large number of variables.

The training of the neural network is done by the generalized back propagation al-

gorithm to optimize the sum squared error as a function of the weights of each node output and the bias values. An exact value for the gradient can be computed through application of the chain rule. Our implementation is a variation of the one outlined by K. Ming Leung in his lecture on back-propagation in multilayer perceptrons.(Leung, 2008). The weights are initialized by sampling from a Gaussian distribution with a mean of zero and a variance of the square root of the number of input variables as suggested by Bishop (Bishop, 1995). Below we will discuss the generalized learning rule and methodology for our gradient decent method.

5.1.1 Neural Network Notation

Consider a general Neural Network with L layers which excludes the input layer but includes the output layer, shown in Figure 4.1:

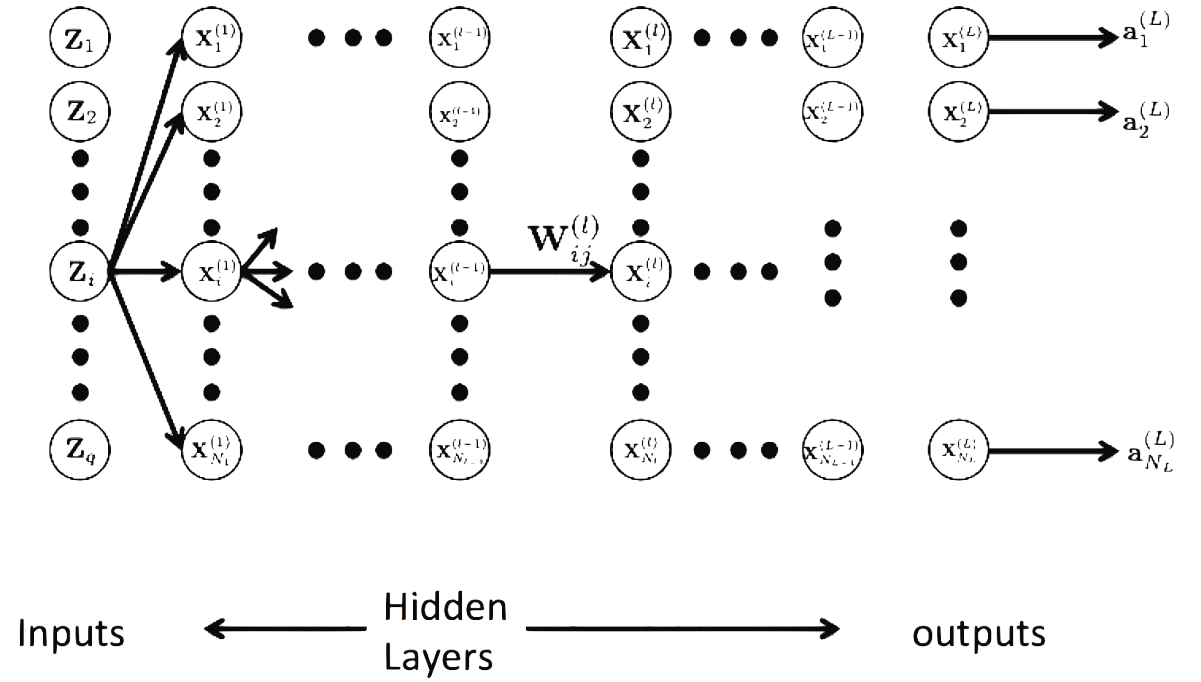


Figure 5.1: A general multilayer Neural Network consisting of L hidden layers and N_L nodes in each layer. The ith node in the lth layer is expressed as $X_i^{(l)}$. The weight from node i in layer $l-1$ to node j in layer l is defined as the i,j element of the $\boldsymbol{W}^{(l)}$ matrix. The output of the activation function at node $X_i^{(l)}$ is given by $a_i^{(l)}$. There is an implied bias input to each node that is not drawn in the image above.

In Figure 5.1 layer l has N_l nodes denoted $X_i^{(l)} \in \{X_1^{(l)}, X_2^{(l)}, \ldots, X_{N_l}^{(l)}\}$. We will consider each node to have the same activation function denoted, f. However, one could easily extend this algorithm to have a different activation function for each node or layer. $w_{ij}^{(l)}$ is the weight factor from $X_i^{(l-1)}$ to node $X_j^{(l)}$. We can also define a N_{l-1} by N_l weight matrix, $\mathbf{W}^{(l)}$, for each layer whose elements are $w_{ij}^{(l)}$. In addition, every node will have a bias term defined as, $b_j^{(l)}$. If we define the output of node $X_j^{(l)}$ as $a_j^{(l)}$ we can write down

the input to a given node $X_j^{(l)}$ as the following equation:

$$n_j^{(l)} = \sum_{i=1}^{N_{l-1}} a_i^{(l-1)} w_{ij}^{(l)} + b_j^{(l)}, \ \forall j \in \{1, 2, \ldots, N_l\},$$

and the output of a node, $X_j^{(l)}$, is given as the following equation:

$$a_j^{(l)} = f\left(n_j^{(l)}\right)$$

5.1.2 Back propagation of a many layer Neural Network

For a given set of labelled training vectors:

$$\mathbf{T} = \{(Z_1, Y_1), (Z_2, Y_2), \ldots, (Z_M, Y_M)\},$$

the goal of training is to minimize the sum squared error:

$$\mathbf{E} = \sum_{i=1}^{M} ||Y_i - t_i||^2$$

where t_i is the output of the neural network when given the input vector Z_i.

For our algorithm we will only be considering one training example at a time, so $\mathbf{E}$ becomes just a single element. We then need to compute the gradient of $\mathbf{E}$ with respect to all of the weights. The steepest decent algorithm then gives us the following update rule for our weights and biases:

$$w_{ij}^{(l)}(time + 1) = w_{ij}^{(l)}(time) - \alpha \frac{\partial \mathbf{E}}{\partial w_{ij}^{(l)}(time)}$$

$$b_j^{(l)}(time + 1) = b_j^{(l)}(time) - \alpha \frac{\partial \mathbf{E}}{\partial b_j^{(l)}(time)}$$

where α is the learning rate or step size and is greater than 0.

The partial derivatives can be directly calculated through application of the chain rule and back propagation through the network to the given weight. While the formal derivation will not be included here we can express the partial derivatives through the following equations. For the output values, which are defined as layer a^L, we can define the s values as follows.

$$\frac{\partial \mathbf{E}}{\partial w_{ij}^{(L)}} = a_i^{L-1} * s_j^{(L)}$$

where $s_n^{(L)}$ is defined as:

$$s_n^{(L)} = 2 * \left(a_n^{(L)} - t_n\right) * \dot{f}^{(L)}(n_n^{(L)}), \ \forall n \in \{1, 2, \ldots, N_L\}$$

and $\dot{f}$ is the derivative of the activation function. For the remaining internal hidden layers we get the following:

$$\frac{\partial \mathbf{E}}{\partial w_{ij}^{(l)}} = a_i^{(l-1)} s_j^{(l)}$$

$$\frac{\partial \mathbf{E}}{\partial b_j^{(l)}} = s_j^{(l)}$$

where $s_h^{(l)}$ can be found recursively through the following equation:

$$s_j^{(l-1)} = \dot{f}\left(n_j^{(l-1)}\right) \sum_{i=1}^{N_l} w_{ji}^{(l)} s_i^{(l)}$$

We can then use these equations to train any arbitrary feed-forward Neural Network via gradient decent and back propagation.

To improve the training time of the algorithm, we randomly choose a single example from the training set without replacement and update the weights based on the sum squared error with respect to this training example instead of batch optimization. There is also an optional momentum parameter which will add a fraction of the previous step direction to the current step direction.

Due to a heavy majority of examples with no precipitation, our initial results provided low classification error rates but classified everything has a non-precipitation event. After experimenting with different solutions to this problem, we settled on modifying our training algorithm to probabilistically sample precipitation examples evenly with non-precipitation examples during the stochastic training.

Then in an effort to optimize our convergence during training we again modified the codebase to perform a simple line search. Three different step sizes are used over a user set number of iterations before the new total sum squared error error is calculated. The step size with the smallest new sum-squared is then taken as our new step size. This modification led to better convergence but much longer training times.

Additionally, we can use this algorithm to perform logistic regression by simplifying our network architecture to one hidden layer and one network node.

5.2 Radial Basis Function Neural Network

We also implemented a specific kind of neural network called a radial basis function (RBF) network, the structure of which can can be seen in Figure 5.2. An RBF network has only one hidden layer composed of an arbitrary number of radial basis functions, functions that are only dependent on the distance of the input from some point. In this case, we call these points the centers, and pick k centers in our sample space to correspond with k unique radial basis functions. Two general forms of radial basis functions that we chose to use are the Gaussian and thin-plate spline, shown below, respectively.

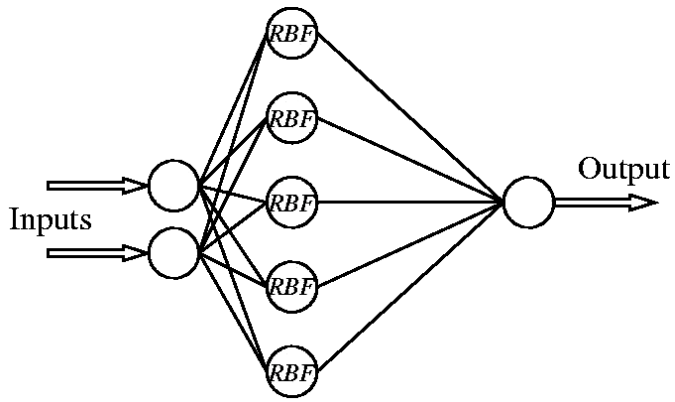


Figure 5.2

$$\phi_j(x) = e^{-\frac{||x-c_j||^2}{-2\sigma_j^2}}$$

$$\phi_j(x) = ||x - c_j||^2 \ln\left(||x - c_j||\right)$$

For each of these equations, x is the input sample being trained on or tested, and c_j the jth center vector. There are various ways to define the scaling parameter σ_j in the

Gaussian function for an RBF network (Schwenker, et al). We tried two of the commonly used definitions, $\sigma_j = \frac{d_{max}}{\sqrt{2m}}$ and $\sigma = 2d_{mean}$, where d_{mean} is the mean distance between all samples and centers, d_{max} the maximum distance of all samples from center j, and m the number of samples. After preliminary testing showing very similar results for each definition, we chose to use $\sigma = 2d_{mean}$ for computational efficiency.

There are also multiple ways to choose center vectors for the k RBFs. We focused on random selection from our sample space in two ways, first selecting random samples from our training set and second selecting a random $k \times n$ matrix of feature values from our training matrix X. Note, for the thin-plate spline RBF we are taking the logarithm of the distance from a sample x to a center c_j, and thus we could not use identical random samples from our training set as the distance would be zero for some x. Other ways to pick center vectors include using a k-means clustering algorithm and using orthogonal least squares to pick k vectors in the sample space. One object of future work would be to implement the choice of centers as a hyper-parameter to determine which methodology gives the optimal results, as well as run cross-validation on this process to improve results.

It is also worth noting that there are two distance metrics used in choosing centers for RBF networks, the standard Euclidean metric, and the Mahalanobis metric. We attempted to implement each of these metrics and choose that which performed best, but ran into numerical problems. The Mahalanobis metric is dependent on the covariance of the centers and samples, which by definition is positive semi-definite. However, numerical problems were giving us arbitrarily small, negative values in our covariance matrix, making the Mahalanobis metric unusable.

After designing our algorithm, we chose to use $k = 100, 500$, and 1000 for the Gaussian as well as the thin-plate spline RBF to get a set of predictions for a wide spread of hidden layer sizes and two different RBFs.

The advantage of RBF networks is that the optimization can be done in two parts, closed form, thus making the computational aspect easier, and allowing for us to consider hidden layers of much larger magnitudes. For a given a weight function W, and a matrix of values calculated by our basis function, Φ, the general of form of our network is

$$y(\mathbf{x}) = W\Phi,$$

where x is some sample. Using a sum-of-squares error, we can then reformulate this as

$$\Phi^T \Phi W^T = \Phi^T Y,$$

where Y is a matrix of outputs for the training set X. We than have that the weight matrix W can be solved for as

$$W^T = \Phi^\dagger T,$$

where $\Phi^\dagger$ is the pseudo-inverse of Φ (Bishop). Thus our training process is to first select centers c_i, for $i = 1, ..., k$ and calculate the each row of our matrix ϕ_j by plugging each training sample x_j into our radial basis function with centers C. We can then use this matrix and our output training set Y to solve for the weight matrix. To test data, we then plug a test sample x into $y(x) = W\Phi$.

5.3 Random Forest

We also implemented a random forest of decision trees. 11,000,000+ possible splits were generated by sorting each variable and taking the midpoint between any two examples where the boolean output value changed between the two output examples. The

forest originally trained each node based on a random subset of 5,000 of the 11,000,000+ possible splits.

5.4 k-Nearest Neighbors

The k nearest neighbors algorithm is a simple model that requires no training but extensive run time computation. The model will simply find the k closest points based on a euclidean distance of the features and then take the majority class as the prediction.

6 Results

6.1 Training and Testing standards

For all of our training and testing we used the following standards. We used data sets from the first 20 years as our training set. The data has been randomly reordered and pre-processed as discussed above. For testing we used the last 3 years of data.

As classification became our primary concern of this project, we needed to convert our output values to boolean values. To do this we considered precipitation greater than .1 inches to be significant, and thus classified outputs greater than .1 as precipitation. Predictions of less than .1 inches were set to zero as non-precipitation.

There is no easy way to rank our classifiers as the "best" since we could tune our model to predict arbitrarily accurate for either of the two classifications. We therefore present the classification errors on both classes as well as the overall classification error to provide more information as to the success of our model.

6.2 Neural Network Classifier

In our final version of the neural network we were able to get informative results training our network. We ran 10-Fold cross validation across 21 different network complexities attempting to train our network to predict precipitation 6 hours in the future. These cases trained different network structures ranging from 2 to 4 hidden layers with either 5, 10 or 20 hidden nodes per layer. The best average validation error for the classification data was 0.1509 with a network structure of 3 hidden layers with five nodes in each of the two non-output layers. Figure 6.1, below, shows an example of the error on the test and validation set as a function of the epoch during training. The network had a two hidden layers with 3 and 1 nodes in layers 1 and 2 respectively.

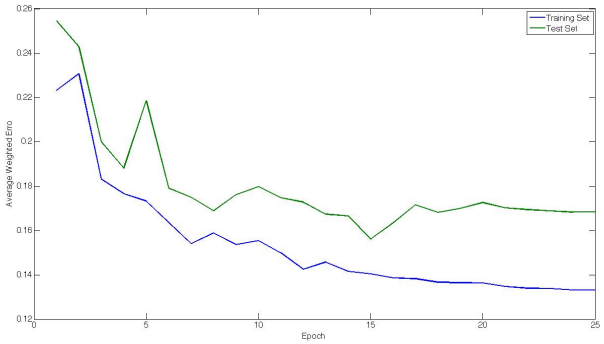
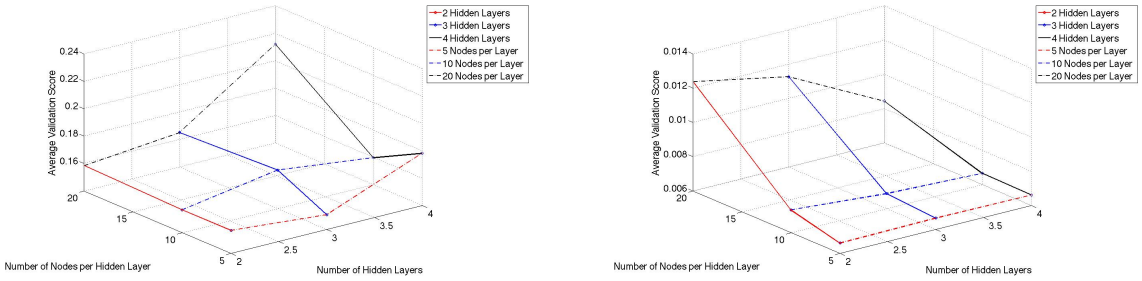


Figure 6.1: This figure shows an example of the error on the training and validation set during the training of a neural network. This network has two hidden layers with 3 and 1 nodes in layers 1 and 2 respectively. Each epoch trained over $\frac{1}{4}$ of the examples in the training set

For visualization purposes Figure 6.2 shows both our classification and regression results for nine of these cases where the number of hidden nodes were all the same at each layer.



(a) (b)

Figure 6.2: Results for both our classification and regression data predicting precipitation 6 hours in the future. 10-Fold cross validation was run across varied network structures with different number of hidden layers and a varied number of nodes in each layer. (a): Average Validation Error Rate for each case when training our classification network. (b): Average Sum Squared Error of the Validation set when training our regression network

Having observed that, in general, the simpler neural network structures produce the most extensible results to unseen data, we re-ran our 10-Fold cross validation considering only 2 layer networks. In this experiment we varied the number of hidden nodes in the first layer from 1 to 10. Figure 6.3 below shows the average validation error and average training error for these 10 cases. However we can see that there was little trend from simple to complex structure.

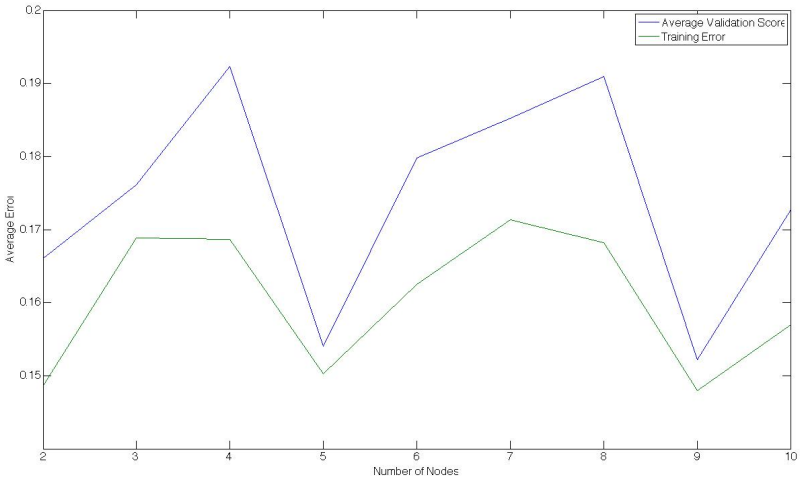
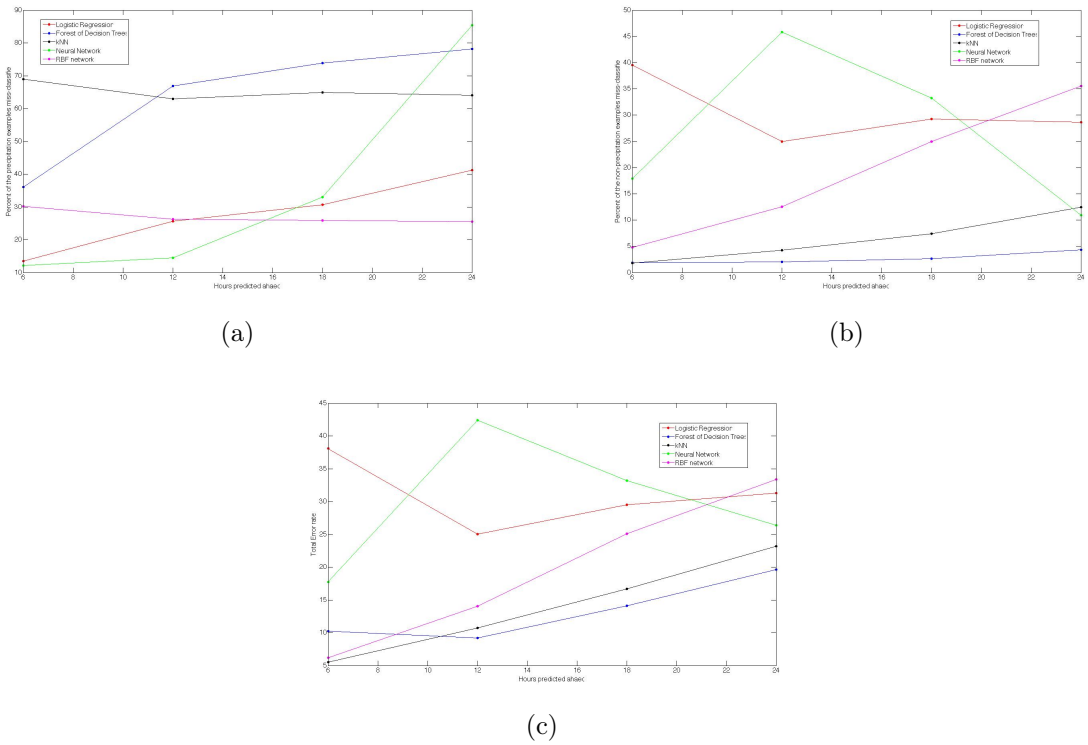


Figure 6.3: This shows the average validation scores of networks trained during 10-fold cross validation as we varied the number of nodes in the layer.

Having achieved fairly good success classifying precipitation 6 hours in advance, we then moved on to predict further into the future, running our training algorithm on prediction data sets of 12, 18 and 24 hours. Due to training times of up to 45 minutes per network we decided to not run 10-Fold cross validation but instead simply train each case one time. Unfortunately, due to the stochastic nature of our training method this

provides a rather inaccurate and noisy look into the prediction potential of our model for the different network structure cases. The best results for these larger prediction times are contained in Table 7.1.

Below Figure 6.4 illustrates the best results as we increased prediction time. We can see that in general our predictions get worse as time increases. However, the decision trees were able to maintain a fairly low overall error but were one of the worst with classifying precipitation examples.



(a) (b)

(c)

Figure 6.4: Figure (a) shows the percent of the precipitation examples miss classified in our models as a function of the prediction time. Figure (b) shows the percent of the non-precipitation examples miss classified in our models as a function of the prediction time. Figure (c) shows the total error rate as a function of prediction time.

6.3 RBF Classifier

The RBF algorithm that we implemented was used as a classification algorithm, although it predicted a given amount of precipitation. This was done in a similar manner to the conversion of our data to boolean values, wherein we considered precipitation greater than .1 inches to be significant, and thus classified outputs greater than .1 as precipitation. Predictions of less than .1 inches were set to zero as non-precipitation.

After training our RBF algorithm on the training data for both the Gaussian and thin-plate spline over time periods of 6, 12, 18 and 24 hours, and for hidden layer sizes of $k = 100, 500$, and 1000, we ran our test data through the resulting classifier. Below is a table showing the error for precipitation events, error for non-precipitation events and overall error for each of the resulting classifiers.

As can be seen in Table 1, the RBF algorithm performed admirably well, especially when extending our forecasts to longer time intervals. Both the Gaussian and thin-plate

RBF	Gaussian				Thin-Plate Spline	
Error	Precip.	Non-Precip.	Total	Precip.	Non-Precip	Total
hr,k = 6,100	.473	.035	.059	.482	.026	.051
hr,k = 6,500	.314	.043	.058	.343	.043	.06
hr,k = 6,1000	.302	.048	.062	.294	.043	.057
hr,k = 12,100	.319	.144	.164	.346	.138	.161
hr,k = 12,500	.288	.132	.149	.302	.15	.167
hr,k = 12,1000	.263	.126	.141	.327	.144	.164
hr,k = 18,100	.291	.272	.275	.281	.285	.284
hr,k = 18,500	.264	.259	.259	.303	.277	.281
hr,k = 18,1000	.259	.25	.251	.303	.291	.293
hr,k = 24,100	.278	.368	.349	.28	.407	.381
hr,k = 24,500	.234	.36	.334	.239	.418	.381
hr,k = 24,1000	.255	.355	.334	.26	.404	.374

Table 1: Precipitation, non precipitation and total error rates on the test set for different RBF network structures.

spline produced good results, with the Gaussian proving to be marginally more accurate. Although the RBF classification of precipitation events six hours in advance could not compete with the standard neural network results, as we increased the prediction timeframe all the way to 24 hours, the RBF network demonstrated a robustness that we saw in no other algorithms. A part of this robustness however did lead to some over-classification of precipitation events, leading to a higher overall error. The RBF error in classifying precipitation events in the training set can be seen in Figure 6.5, the error for non-precipitation events can be seen in Figure 6.6, and the overall error can be seen in Figure 6.7. Note, these figures contain results from both the Gaussian and thin-plate spline basis functions, for k-values of 100, 500 and 1000, and time intervals of 6, 12, 18 and 24 hours.

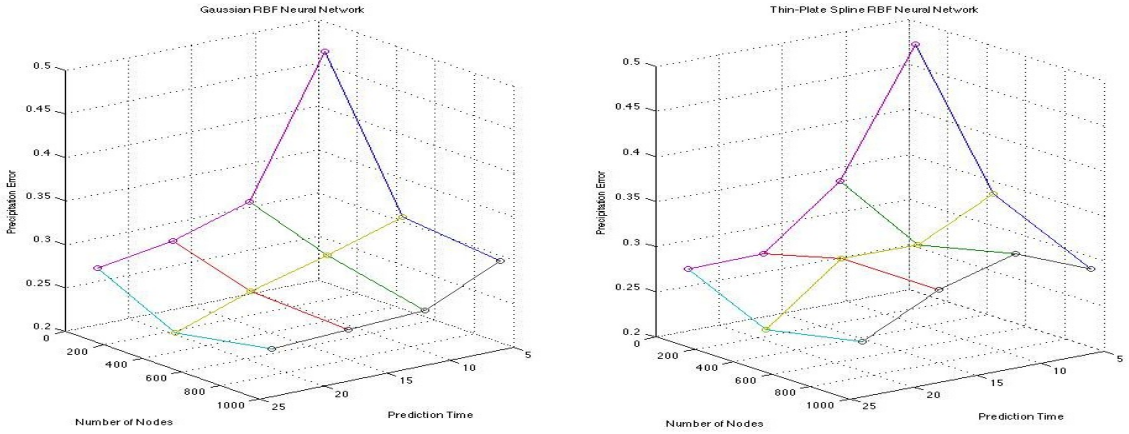


Figure 6.5: The error of the RBF classifying precipitation events for both the Gaussian and thin-plate spline basis functions.

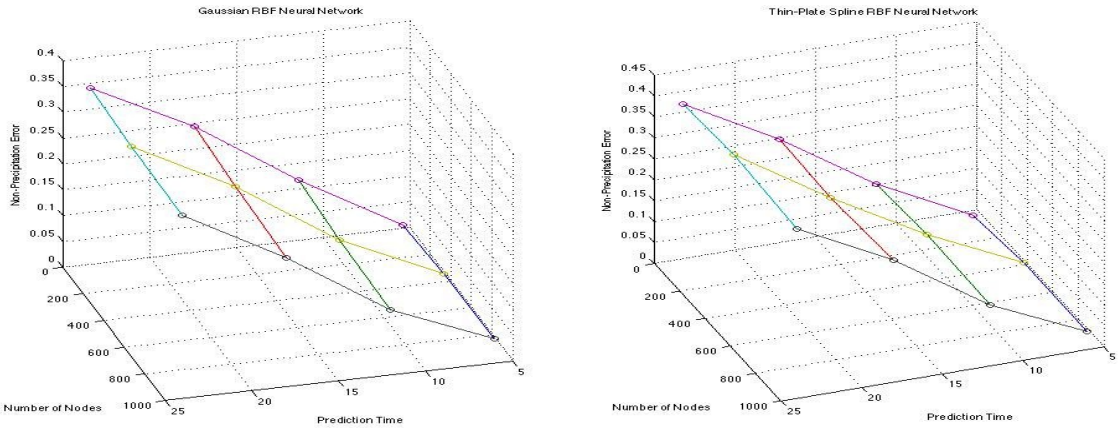


Figure 6.6: The error of the RBF classifying non-precipitation events for both the Gaussian and thin-plate spline basis functions.

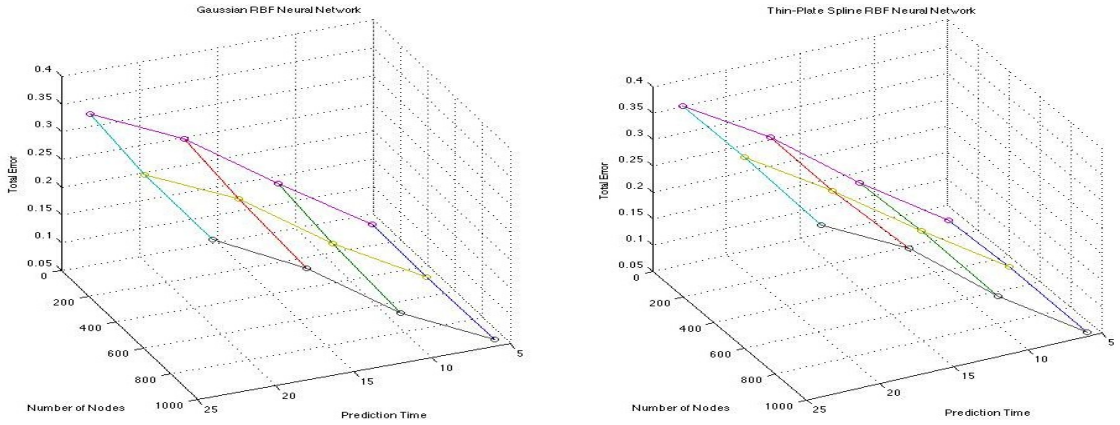


Figure 6.7: The total classification error of the RBF for both the Gaussian and thin-plate spline basis functions.

It is interesting to notice that the plots for the classification error of non-precipitation events and overall classification error are very similar, but in fact it makes sense. There are so many more samples of non-precipitation than precipitation events that the total error rate and non-precipitation error rate should be similar, as non-precipitation events make up such a large proportion of the entire test set. A similar correlation can be seen in other algorithms that produced very low total error rates, which were initially convincing, until it was determined that the algorithm had simply classified everything as non-precipitation.

The over-classification done by the RBF network can be seen in the opposite trends of error in precipitation events and non precipitation events as the time frame increased. To maintain a robust accuracy in predicting precipitation events, the algorithm began classifying more precipitation events over longer timeframes, clearly not all of which were precipitation events. Thus the accuracy of precipitation classification actually increased slightly as the timeframe increased, but the overall accuracy and accuracy of non-precipitation simultaneously declined linearly.

6.4 Comparison:

To compare the results of our networks to a baseline, we also ran our data through a few simpler models for comparison. We chose to run a forest of 20 randomly generated decision trees, k nearest neighbors and logistic regression.

6.4.1 Decision Tree

The forest of decision trees model was run using 20 decision trees randomly choosing their split criteria from a subset of 10,000 of the full 11+ million split cases generated. The classification was performed off of majority vote.

Running our data through our forrest of random decision trees we were able to get reasonably good results for the 6 hour time frame. However, when we extended our predictions out to longer time frames we began to classify more than 70% of the precipitation examples incorrectly. Our best results are shown in Figure 7.1.

6.4.2 k Nearest Neighbors

Predictions were made on our test set using the simple kNN model with k = 5 and achieved moderate results. We were unable to perform optimization of the k hyper parameter due to 6+ hours of computation time to evaluate all predictions on our test set.

One interesting result from this model was that the error rates were very similar for time frames of 6, 12, 18 and 24 hours with about 65-70% of the classification examples miss-classified and 2-4% of the non-precipitation examples miss-classified. The results of our kNN model can be seen in Figure 7.1

6.4.3 Logistic Regression

Our logistic regression model was made by simplifying the neural network to contain only one hidden layer with one hidden node. This simple, linear decision boundary had surprisingly good results, especially at large time frames and arguably performed better than our other models at predicting precipitation over the next 24 hours. The results of our best logistic regression models can be seen in Figure 7.1.

7 Conclusion

A comparison between our best classification results for each model can be seen below in Figure 7.1. We can see that both our neural networks as well as the radial basis functions performed very well against the simpler algorithms. The radial basis function also performed better than other algorithms as we moved out to longer prediction times.

A majority of our work focussed on initially classifying our precipitation data but we also performed regression analysis on a subset of our precipitation data where we only considered examples with non-zero precipitation amounts. The 10-Fold cross validation results from training our neural network over varying network complexity can be seen in Figure 6.4b. The average validation error for the regression data was 0.0066 for a network structure of 10, 5, 5, and 1 nodes for hidden layers 1, 2, 3, and 4 respectively.

Finally, taking our best overall neural network models we were able to test our cascade prediction model. For each test example we first classify the example as either a precipitation event or not. If our classifier predicts no precipitation, the output value is set to 0 inches. Otherwise the example is fed into our regression model. This model will output the normalized rain prediction. Our classification network structure was a 2 layer

Model	Prediction timeframe	% wrong of precip	% wrong of non precip	Overall Error Rate
Logistic Regression	6	13.469	39.536	38.079
Forest of Decision Trees	6	36.031	1.850	10.228
kNN (k=5)	6	68.980	1.788	5.544
Neural Network (best)	6	12.121	17.869	17.742
Radial Basis Function (best)	6	30.204	4.785	6.206
Logistic Regression	12	25.673	24.942	25.023
Forest of Decision Trees	12	66.874	2.053	9.201
kNN (k=5)	12	62.940	4.260	10.731
Neural Network (best)	12	14.493	45.856	42.397
Radial Basis Function (best)	12	26.294	12.574	14.077
Logistic Regression	18	30.682	29.260	29.488
Forest of Decision Trees	18	73.864	2.640	14.093
kNN (k=5)	18	64.915	7.431	16.674
Neural Network (best)	18	32.955	33.234	33.189
Radial Basis Function (best)	18	25.852	24.986	25.100
Logistic Regression	24	41.209	28.650	31.261
Forest of Decision Trees	24	78.132	4.299	19.653
kNN (k=5)	24	64.066	12.435	23.172
Neural Network (best)	24	85.385	10.906	26.394
Radial Basis Function (best)	24	25.494	35.520	33.380

Figure 7.1: This table outlines the best results for the various models tested over the four different prediction time frames.

network with 3 and 1 nodes in layers 1 and 2 respectively. Our regression network was a four layer network with 10, 5, 5, and 1 nodes in layers 1, 2, 3, and 4 respectively. On our test set, predicting six hours in the future, we got a sum squared error of only 0.0012 with a miss-classification of 23 out of 99 precipitation events.

The RBF network could be enhanced by running cross-validation, something computationally intense but favorable to improved results. It is also worth looking into alternative basis functions, as we only considered two of many possible options. The Gaussian performed better of the two, probably due to its inclusion of a scaling factor gauging the general distance between samples and centers, and there are other basis functions with similar scaling parameters. Last, the choice of centers is a critical part of the RBF network, and implementing either clustering algorithms or an orthogonal least squares method could offer significant improvement on results.

Overall, our classification networks performed favorably with those published by Santhanam et all. Furthermore we were able to achieve moderate success looking further into the future with 12, 18 and 24 hour predictions. Our best classification results at predicting precipitation in six hours was a neural network structure, which produced error rates of 12.1% and 17.9% on the precipitation and non-precipitation examples respectively. This presents an improvement over the Santhanam group which achieved only 19.7 % and 16.3% error rates classifying rain and no rain respectively. Furthermore, we were able to achieve relative success taking our algorithms and training them for longer prediction times.

Future work could look into more network structures, especially for regression. Most of our work focussed on classification of precipitation versus non-precipitation and not optimizing the regression networks. Very large and complex networks with large numbers of nodes were not considered due to overwhelming training time but these could provide better modeling for the larger test frames. Additionally, further work could look vary the

activation function or the connectivity of a neural network.

8 References:

Bishop, Christopher M. “Neural networks for pattern recognition." (1995): 5.

French, Mark N., Witold F. Krajewski, and Robert R. Cuykendall. “Rainfall forecasting in space and time using a neural network." Journal of hydrology137.1 (1992): 1-31.

Ghosh, Soumadip, et al. “Weather data mining using artificial neural network." Recent Advances in Intelligent Computational Systems (RAICS), 2011 IEEE. IEEE, 2011.

Hall, Tony, Harold E. Brooks, and Charles A. Doswell III. “Precipitation forecasting using a neural network." Weather and forecasting 14.3 (1999): 338-345.

HPC Verification vs. the models Threat Score. National Oceanic and Atmospheric Administration Hydrometeorological Prediction Center, 2012. Web. 20 Jan. 2013. http://www.hpc.ncep.noaa.gov/images/hpcvrf/HPC6ts25.gif.

Hsieh, William W. “Machine learning methods in the environmental sciences." Cambridge Univ. Pr., Cambridge (2009).

Kuligowski, Robert J., and Ana P. Barros. “Experiments in short-term precipitation forecasting using artificial neural networks." Monthly weather review 126.2 (1998): 470-482.

Kuligowski, Robert J., and Ana P. Barros. “Localized precipitation forecasts from a numerical weather prediction model using artificial neural networks." Weather and Forecasting 13.4 (1998): 1194-1204.

Luk, K. C., J. E. Ball, and A. Sharma. “A study of optimal model lag and spatial inputs to artificial neural network for rainfall forecasting." Journal of Hydrology 227.1 (2000): 56-65.

Manzato, Agostino. “Sounding-derived indices for neural network based short-term thunderstorm and rainfall forecasts." Atmospheric research 83.2 (2007): 349-365.

Maqsood, Imran, Muhammad Riaz Khan, and Ajith Abraham. “An ensemble of neural networks for weather forecasting." Neural Computing & Applications 13.2 (2004): 112-122.

McCann, Donald W. “A neural network short-term forecast of significant thunderstorms." Weather and Forecasting;(United States) 7.3 (1992).

Ming Leung, K. “Backpropagation in Multilayer Perceptrons." Polytechnic University. 3 Mar 2008. Lecture.

PSD Gridded Climate Data Sets: All. National Oceanic and Atmospheric Administration Earth System Research Laboratory, 2012. Web. 20 Jan. 2013. http://www.esrl.noaa.gov/psd/data/gridded/.

Santhanam, Tiruvenkadam, and A. C. Subhajini. “An Efficient Weather Forecasting System using Radial Basis Function Neural Network." Journal of Computer Science 7.

Schwenker, Friedhelm, Hans A. Kestler, and Gunther Palm. "Three learning phases for radial-basis-function networks." Neural Networks 14.4 (2001): 439-458.

Silverman, David, and John A. Dracup. “Artificial neural networks and long-range precipitation prediction in California." Journal of applied meteorology 39.1 (2000): 57-66.

Veisi, H. and M. Jamzad, 2009. sc. Int. J. Sign. Process., 5: 82-92. “A complexity-based approach in image compression using neural networks."
http://www.akademik.unsri.ac.id/download/journal/files/waset/v5-2-11-5.pdf.

Zhang, Guoqiang Peter. “Neural networks for classification: a survey." Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on 30.4 (2000): 451-462.