PSO Trained Neural Networks for Predicting Forest Fire Size: A Comparison of Implementation and Performance

Jeremy Storer
Department of Computer Science
Bowling Green State University
Bowling Green, OH 43403
Email: jstorer@bgsu.edu

Robert Green
Department of Computer Science
Bowling Green State University
Bowling Green, OH 43403
Email: greenr@bgsu.edu

Abstract—**Forest fires are a dangerous and devastating phenomenon. Being able to accurately predict the burned area of a forest fire could potentially limit human casualties as well as better prepare for the ensuing economical and ecological damage. A data set from the Montesinho Natural Park in Portugal provides a difficult regression task regarding the prediction of forest fire burn area due to the limited amount of data entries and the right skew nature of the data set. This paper shows how the use of a novel input structure and representation of the data, along with using Particle Swarm Optimization (PSO) instead of Backpropagation to determine weights of an Artificial Neural Network (ANN), improves error rates significantly.**

I. Introduction

Forest fires are an unpredictable and highly destructive force of nature [1]. They are able to create tremendous amounts of ecological and economic damage, while simultaneously endangering life. If it were possible to peer into the future and see these catastrophic events before they unfold it would greatly assist in mitigating the destructive impact of forest fires.

Artificial Neural Networks (ANN) attempt to predict the outcome of future events by learning from the past. To accomplish this, supervised training of the ANN is used. Supervised training entails a set of input dimensions with an expected output result being fed into an ANN so that it can be trained to identify trends or patterns. Once trained, the ANN can perform regression on a set of data as a means of predicting output values. The type, amount, and overall characteristics of the data fed into an ANN is the most critical aspect to consider when attempting to train an ANN to reliably predict output results. If the amount of data is too small or heavily skewed, ANNs become very difficult to train. If the amount of data is small it is difficult to adequately train for all possible results. If the data is skewed the ANN will be bias towards a result that is overrepresented in the data set. This leads to difficulty converging on an optimum, which can result in inaccurate output predictions.

Data sets available for forest fire burn areas are limited in the number of instances that they contain. This is due, in part, to the fact that each area of the Earth has unique environmental and meteorological conditions. Thus, data from one part of the Earth will not necessarily be helpful for predicting the burn areas of forest fires at different geographical locations. Coupling this with forest fires being a fairly rare event leads to few data set instances. To provide useful predictions of burn areas a method must be devised that is able to give reasonable predictions using a small, and often times skewed, data set.

II. Montesinho Dataset

The data set used in this paper was collected from January 2000 through December 2003 and built using two sources: First, a set of data collected by an inspector that is responsible for documenting Montesinho Park fire occurrences. Every time a forest fire occurred in the park several features were registered by this inspector; the time, date, the spatial location within a 9x9 grid of the park (inputs X, Y), the type of vegetation involved, components from the FWI system, and the total area burned [2]. Second, an organization collected information about wind speed and other weather patterns using a meteorological station at the center of Montesinho Natural Park. These two data sets were combined to form the one that is used in this thesis.

The Fire Weather Index (FWI) is a Canadian system for rating fire danger [2] and contains six elements: the Fine Fuel Moisture Code (FFMC) which denotes the moisture content of surface litter that influences ignition and fire spread, the Duff Moisture Code (DMC) and Drought Code (DC), which represent the moisture content of shallow and deep organic layers, the Initial Spread Index (ISI), which is a score that correlates with fire velocity spread, the Build Up Index (BUI), which represents the amount of available fuel and the final FWI rating, which is derived from the other values.

The complete Monteshino dataset contains twelve input attributes and one output attribute, the burn area, which is measured in hectares (a hectare is equal to 10,000 m^2). The inputs are X, Y, month, day, FFMC, DMC, DC, ISI, temperature, relative humidity, wind speed, and rain. FFMC, DMC, DC, and ISI are taken from the FWI system. The data set supplied for this thesis did not contain information on the

BUI or final FWI rating. The attributes and ranges can be seen in Table I.

Attribute	Range
X,Y coordinates	1 to 9
Month	Jan to Dec
Day	Mon to Sun
FFMC code	18.7 to 96.2
DMC code	1.1 to 291.3
DC code	7.9 to 860.6
ISI index	0 to 56.1
Temp (C)	2.2 to 33.30
Relative Humidity	15.0 to 100
Wind Speed (km/hr)	0.40 to 9.40
Rain (mm/m^2)	0.0 to 6.4
Burned area size (ha)	0.00 to 1090.84

TABLE I
DATA ATTRIBUTES AND RANGES OF THE MONTESINHO DATA SET.

In the case of this paper, data is provided from Montesinho Natural Park, which is in the Trs-os-Montes northeast region of Portugal [3]. The data was collected from January 2000 to December 2003 and built using two sources. The fire inspector responsible for tracking fire occurrences in the park collected the first portion of the data. Every time a forest fire occurred, several features were registered such as the time, date, spatial location within a 9 x 9 grid, the type of vegetation involved, the six components of the FWI [4] system and the total burned area. The second data component was collected by the Braganca Polytechnic Institute and contained information such as wind speed. The two were then merged into a single data set with a total of 517 entries.

The forest fire data from Montesinho has proven difficult to use as a source for training an ANN to predict the burn area of other fires in the area as there are only 517 entries in the set. A large percentage of the burn areas being 0.0 (anything with less than $100m^2$ of burn area is considered statistically insignificant and classified as a 0.0 burn area) compounds the difficulty in ANN training. There are very few entries that have non-zero burn areas, as forest fires larger than $100m^2$ are a relatively rare event. This makes it very challenging to train an ANN to predict burn area since small forest fire events will be heavily overrepresented in the data set giving very limited opportunities for the ANN to learn under which circumstances larger fires occur.

Several researchers have used the traditional Backpropagation algorithm, as well as variants of Particle Swarm Optimization (PSO) in an attempt to create an ANN that can predict the burn areas reliably. These attempts have used ANNs with different architectures and activation functions that have resulted in limited success. Using root mean squared error (RMSE) as a metric for comparison; current attempts of the Backpropagation method have netted results that hover around an RMSE of 60. This is a fairly poor result that would not greatly assist in a reliable prediction of burn area. It is thought that, for the most part, use of the Backpropagation algorithm to solve this problem has been exhausted and new avenues should be pursued.

As an attempt to move away from using Backpropagation to train an ANN, Particle Swarm Optimization (PSO) is used in its place for this paper. PSO provides a different way to determine the weights of an ANN than Backpropagation and thus reacts to data sets in a different manner. In the case of the Montesinho dataset, this paper shows that PSO provides a large decrease in the RMSE value. To accompany the use of PSO in lieu of Backpropagation, the input data is represented in a novel fashion, using 29 inputs instead of 12. This creates an ANN architecture that is unique to this dataset and used far less often than the traditional 12 inputs. Instead of having one input neuron for each attribute, each attribute with discrete levels (e.g. the month attribute has 12 discrete values) was represented by a number of input neurons equivalent to the number of discrete values. Using both this novel input method and PSO to train the weights of the ANN; a drastic decrease of RMSE was produced. This paper will detail these findings along with reproducing some of the poorer results using Backpropagation.

III. RELATED WORK

Multiple works have attempted to regress this data set using a variety of methods. One of the more extensive attempts of predicting forest fire burn areas with the Montesinho data set is from Paulo Cortez and Anibal Morais [5]. They use five different models in an attempt to find one suitable for regressing the data: Multiple Regression (MR), Decision Trees (DT), Random Forests (RF), ANNs, and Support Vector Machines (SVM). Each of the five models were then tested using four different sets of inputs. The four input models were derived to determine if any of the variables had impact on the error rates if they were to be eliminated from the input stage.

Cortez and Morais believe that SVM has a theoretical advantage over an ANN due to the absence of local minima in the model optimization phase. They loosely confirmed their beliefs by showing their strongest testing results were obtained using SVM. Even though the SVM results were the best, with an root means squared error (RMSE) value of 64.7, it was by a fairly insignificant amount as the ANN trained with backpropagation obtained an RMSE of 66.9. These differences could be attributed to a non-optimal ANN architecture. Non-optimal in this case meaning that if different activation functions or different a number of hidden layers or perceptrons were to be used results could be improved.

The use of five different training models and four input methods resulted in 20 different systems tested. Their RMSE rates varied between the range of 64.7 and 68.9. This is a relatively small difference between all of their results. This could mean that taking away input items may have little effect on the RMSE scores. Their methodology did not account for outliers very well so they are able to predict many of the smaller sized fires but struggle with fires of larger size.

Safi and Bouroumi [6] predicted forest fire burn areas of the Montesinho data set as well. They use backpropagation to train the ANN and aim to minimize the global error on the output layer. A variety of architectures are tested with varying levels of perceptrons and hidden layers. Their best results were obtained using a single hidden layer ANN consisting of 35 perceptrons that resulted in a 10% rate of error.

Safi and Bouroumi also found that their results hinge on the number of training iterations performed. There is no obvious pattern to the number of training iterations that should be performed but results vary depending on the the choice. For instance, at 10,000 training iterations they have a roughly 5% variation in the error rate, while at 50,000 training iterations they have roughly a 60% variation in error rate. These numbers are found using identical ANN architectures demonstrating that the number of training iterations could be something important to consider. However, that large amount of training iterations on a data set so small could likely result in over-fitting and the found results should be documented multiple times to verify the number of iterations is indeed what is effecting the error score.

Castelli, Vanneschi, and Popovic [7] proposed an intelligent system based on genetic programming for the prediction of burned areas. They only consider data relating to FWI characteristics and meteorological data in order to build predicting models, employing geometric semantic genetic operators for genetic programming. Strong results are obtained using this method along with testing other methods such as Random Forests, Radial Basis Function Networks, ANNs, and basic Linear Regression. The authors calculate error using mean absolute error (MAE) between predicted and target values. Their ANN system obtains a MAE of 33.8 while their geometric semantic genetic programming method obtains a 12.9 MAE. These findings are very strong but do not contain RMSE values. If RMSE is used as the error metric the numbers would most likely increase due to the outlier nature of the data set. Their work showed that methods other than ANNs are viable for regressing this data set.

Jain and Bhatia [8] use Support Vector Regression to regress the Montesiho data set. The goal of their research is to identify the best parameter settings using a grid-search and pattern search technique as well as to compare the prediction accuracy among the models using different data sorting methods, random sampling and cross validation. They found that E-SVR performs better using various fitness functions and variance analysis. Their best RMSE result being 63.51 with the E-SVR method using 10-fold cross validation.

Wang et al. [9] once again attempted to regress the data set and use a variant of particle swarm optimization (PSO), known as Immune Particle Swarm Optimization (IPSO) with Support Vector Regression (SVR), to execute the task. The immune portion of the algorithm name refers to the Immune Algorithm (IA). IA is a meta-heuristic search algorithm, which is concerned with keeping diversity of the solution group and avoiding particles from becoming stuck in local optimums when used in conjunction with PSO [10]. SVR differs from SVM work [5] as it finds a hyperplane which can accurately predict the distribution of information, but not the plane used to classify the data.

This work uses RMSE as the fitness value measurement of each particle in the PSO algorithm. This was done in an attempt to avoid over fitting and to get a high generalization performance value. A novel approach was also presented by having an affinity for each particle to clone itself. Thus, a particle with a larger affinity would clone more offspring to protect its good genes and accelerate the convergence rate of the algorithm. An aspect of mutation was also introduced into the PSO algorithm to assist in more reliably finding the global optimum. After the clonal copy and mutation process, the best individual mutated particle for each cloned group is selected to compose the next generation. The authors found that there is nearly no difference when using PSO versus IPSO with both achieving an RMSE of approximately 64.1. Thus, using PSO or IPSO alone did not net a very significant decrease in RMSE compared to using a backpropagation trained ANN. Though related to this work, the work presented in this thesis is substantially different because as opposed to using the PSO algorithm alone. PSO is used in conjunction with an ANN to act as the training algorithm instead of backpropagation.

A PSO trained ANN has not been used on the Montesinho fire data set before. However, many trials have been done on various tasks with a PSO trained ANN that demonstrate its validity as a training method. Xue-tong [11] used a PSO trained ANN to predict the prices of petroleum throughout time. He found that using a PSO trained ANN gave better results than that of backpropagation. The primary finding in this work was that convergence occurs at a faster and more consistent rate when using PSO as opposed to backpropagation to train an ANN.

Chau [12] attempts to predict accurate water stages of the Shing Mun River. Accurate water stage prediction would allow the pertinent authority to issue flood advisories or warnings in a more timely manner which would allow earlier planning for evacuations. Two separate data sets are used for trials in Chau's research. Both results confirm that a PSO trained ANN performs better than a backpropagation trained ANN having lower RMSE scores. The backpropagation based ANN had a 0.29 RMSE and the PSO-based ANN a 0.16 RMSE for one trial and a 0.24 versus 0.14 RMSE in the other trial. Both of these results show statistical improvement while using a PSO trained ANN.

Mohammadi and Mirabendini [13] used a data set from the University of California at Irvine (UCI) machine-learning database [14] that contains 21 attributes and 1,000 instances to determine if an instance has good or bad credit. They found that a PSO trained ANN had the lowest mean-square error (MSE) and best accuracy among Levenberg-Marquardt, Gradient Descent, Gradient Descent with Momentum, Gradient Descent with Adaptive Learning Rate, and Gradient Descent with Momentum and Adaptive Learning Rate. This gives PSO winning marks against even more machine-learning methods.

Niu and Li [15] show the merit of using a hybrid PSO-ANN approach as well. They performed tests using the Iris, New-Thyroid, and Glass data sets and then train an ANN using Backpropagation, Genetic Algorithms, and PSO-ANN for each data set. In all cases the hybrid PSO-ANN outperformed the other methods obtaining both the highest accuracy and lowest MSE scores.

IV. Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a population based stochastic optimization technique developed by Eberhart and Kennedy in 1995 [16]. It is modeled after the behavior of flocking birds or schooling fish. PSO is relatively easy to implement and has few parameters that require adjusting for it to work optimally.

PSO is initialized with a group of random particles and then searches for optima by updating generations. In every iteration each particle is updated by following two best values. The first best value being the best solution that particular particle has achieved so far and the second being the best fitness value any particle has obtained so far, these are known as p_{best} and g_{best}, respectively.

After the two best values are found, the particles update their velocity and positions. The velocities are updated via (1) where V_1 represents the newly calculated velocity, w represents a constant called inertia weight, $c1$ and $c2$ are fixed values of 2.05 that were verified in a study by Shi [17], $r1$ and $r2$ represent randomly generated numbers between 0 and 1, Pos_{pbest} is the position of a particles best fitness value, Pos_{gbest} is the position of the global best fitness value of all particles, and $Pos_{current}$ is the current particles position. w is often varied between 1.4 and 0.2 which decreases linearly depending on the training iteration number. The number of particles can also be varied but usually is roughly two or three times the number of input dimensions.

$$\begin{aligned} Q &= \omega v_i + c1 \cdot r1 \cdot (Pos_{pbest,i} - Pos_{current,i}) \\ R &= c2 \cdot r2 \cdot (Pos_{gbest,i} - Pos_{current,i}) \\ V_1 &= Q + R \end{aligned} \tag{1}$$

After the set of velocities is calculated, the new position of the particle is calculated by adding the corresponding velocity and current position point together. The general work-flow of PSO is as follows:

A PSO trained ANN uses PSO to determine the optimal weights of the ANN. Every particle, P, contains several important characteristics: the weights of all of the perceptron connections which represent the position of the particle, the current error measurement of the particle, the current velocity of the particle, and the global best error measurement and position.

On every iteration there is also a chance for each particle to die and then be randomly reinitialized. Kennedy and Eberhart inspired the use of this concept [18]. When a particle dies and is reinitialized it has a chance of reappearing at a location that is more optimal than where other particles currently reside. This prevents particles from converging at a local optima instead of the global optimum.

Algorithm 1: General PSO algorithm.

```
foreach particle do
    initialize particle with random position ;
while exit criteria is not met do
    foreach particle do
        calculate particle's fitness value ;
        if the fitness value is better than the best fitness
          value in particle's history set value as the new
          $p_{best}$ ;
    choose the particle with the best $p_{best}$ of all particles
      as the $g_{best}$ ;
    if the value is better than the best value of global
      particles in history, set the best particle's position as
      the new $g_{best}$ ;
    calculate each particles new velocity and position ;
```

V. Methodology

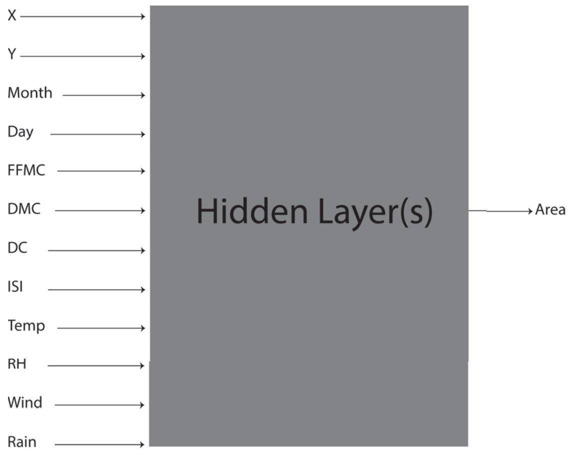
The previous authors' works display similar results when using either an ANN with Backpropagation or PSO to perform the regression task of predicting the burn area of the forest fires. It is thought a new approach needs to be created to continue to improve RMSE results. However, before a new approach is attempted this paper provides an exhaustive approach to Backpropagation to verify that there is little benefit to continue with its use. It is believed that using the structure of an ANN will still provide high flexibility. With this general structure still intact, the Backpropagation algorithm will be replaced with PSO. It is thought that using PSO in place of Backpropagation will net stronger results as it is less likely to get trapped in local optima [13], [15].

A. Data Preprocessing

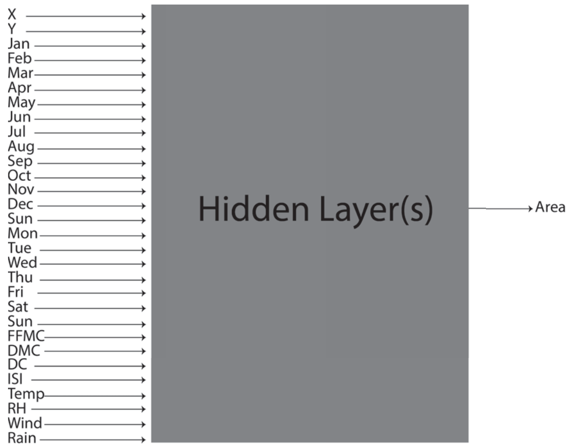
Before beginning to train an ANN with either the Backpropagation or PSO approach, the data must first be preprocessed. The input fields of the Montesinho data set have different scales. If the data fields have different scales some could inadvertently carry more weight than others when training the network. To eliminate this factor, all non-discrete input values are normalized from 0-1. The normalized value, e_i, for variable E in the i^{th} row is calculated as:

$$Normalized(e_i) = \frac{e_i - E_{min}}{E_{max} - E_{min}} \tag{2}$$

where E_{min} is the minimum value for variable E and E_{max} is the maximum value for variable E. Discrete values, such as days of the week or months, have one input node available for each possible value with one hot node, known as 1-of-C encoding [19], [20]. For example, days of the week would be represented by seven nodes with all nodes having a zero value except the intended day, which would have a one value. For example, if the color of a stop light was desired to be expressed in the 1-of-C method it would contain three nodes, each representing a color. Thus, red could be represented as 1

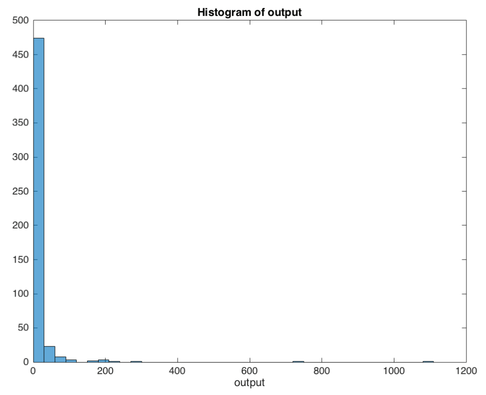


(a) ANN architecture using 12 input method.

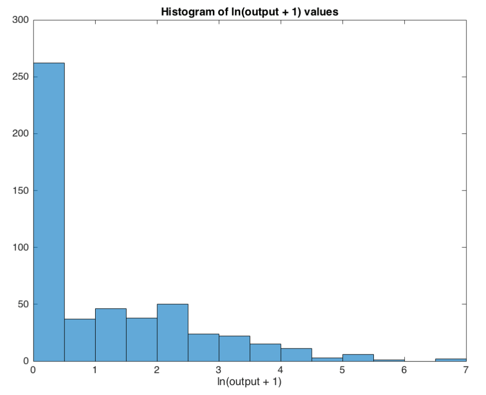


(b) ANN architecture using 29 input method.

Fig. 1. Various ANN architectures as used to classify the Forest Fires Data Set.



(a) Histogram of output values before applying output transformation.



(b) Histogram of output after output transformation applied.

Fig. 2. Histograms of transformed output values for the Forest Fires Data Set.

0 0, yellow 0 1 0, and green 0 0 1. This method resulted in having 29 input attributes instead of the original 12 that were presented as can be seen in Fig. 1.

The output values in the data set had a wide range of values from 0.0 to 1,090.84. Among this wide range of output values many of the values ended up being zero. In fact, 247 of the 517 instances have a 0.0 value for output. The large amount of 0.0 output values result in a right skewed histogram. Taking the natural logarithm of a right skewed histogram will help bring the tail in and assist in training the ANN [21]. Thus, each output value was transformed using

$$output = ln(output + 1) \tag{3}$$

This transformation improves symmetry and reduces the right skew of the histogram as can be seen in Fig. 2. After the output transformation the outputs were also normalized between 0 and 1. With all variables appropriately processed to be fed into an ANN the actual structure of the network can now be considered.

B. Backpropagation Trained ANN

Further exhausting of the Backpropagation algorithm was desired before expanding into other avenues of training for this paper to properly reinforce that they are not the correct method for this regression task. A deep learning library, Keras [22] was used in conjunction with Python to perform a thorough test of architectures containing a single hidden layer of neurons. A key aspect of this paper's ANN construction was using 29 input attributes instead of 12 as discussed in the data preprocessing section.

Keras provides all of the most common and several less often used activation functions along with easy implementation of different ANN architectures. For the purposes of this paper five activation functions were chosen to be used: Sigmoid, Hard Sigmoid, Rectifier, Hyper Tangent, and Linear. With these five activation functions in mind a series of loops were set up in python to account for all activation function permutations within a single hidden layer neural network. The number of hidden nodes was also varied between 1 and 200 for each activation function permutation. This resulted in 5,000 different ANN architectures in total. After the best architecture for each primary activation function was found, the same architecture was used with the standard 12 input method. This was done to verify that the 29 input method is indeed superior to using 12.

Each of the 5,000 different ANN architectures used 5-fold cross-validation of the dataset with 300 epochs of training using Backpropagation for each fold. After each fold completed its training phase it was then tested by taking the inverse of the output transform equation and calculating the RMSE value with the actual output versus the predicted output. These results were then all outputted into a file for analysis.

C. PSO Trained ANN

The PSO trained ANN was customarily constructed in Python with some minor use of the Theano [23], [24] deep learning library to assist in matrix operations. Once again, the PSO trained ANN iterates through different architectures to determine an optimal shape containing one hidden layer that contains between 5 and 60 nodes, as Backpropagation showed small variance in RMSE when the amount of nodes in the hidden layer changed. The total number of particles remained consistent at 250 for each architecture setup. Each training cycle consisted of 50 epochs that would cease iterating if the RMSE of the current epoch was 17 or under. Fewer PSO trained ANN trials were used due the results of the backpropagation trials. It was seen that while different results did occur depending on the combination of activation functions, it was not different enough to warrant the large amount of time it would take to run the same number of trials as backpropagation.

Every particle, P, is a class that contains several important characteristics. The position is stored as an array of size $I * H * O$ where I is the amount of input nodes, which is once again 29 input nodes, H is the amount of hidden nodes, varying between 5 and 150, and O is the amount of output nodes, which is 1. This is because all ANN architectures are fully connected, meaning each node is connected to all other nodes in the proceeding layer. The current RMSE value of the given particle is also stored in this particle class as well as the calculated velocity, which is found via (1).

While velocities, positions and error rates of particles are being calculated there is also a chance for each particle to die. In the case of this paper there is a 0.5% chance of a particle to die on each iteration. Kennedy and Eberhart inspired the use of this concept [18]. When a particle dies it is reinitialized, which means that its position and velocity values are all randomly regenerated so the particle appears at a random location in the answer space. This is to prevent the particles from converging on local optima as the randomly generated particle may have a better RMSE than the current global best. This new global best may not have been found as particles were continuing to converge on local optima instead of the global one. The overall flow of a PSO trained ANN can be broken into several key steps:

Step 1: Treating each weight and bias of a multi-layer ANN as dimensions of a particle in PSO

Step 2: Initialize N particles, in this case N = 250, using a random number generator. In this case the initial positions and velocities are all randomly generated with numbers between 0 and 1.

Step 3: Evaluate each particle's RMSE

Step 4: Modify g_{best} and p_{best} by comparing their RMSE fitness values.

Step 5: Calculate velocities for each particle (1)

Step 6: Move each particle to its new position

Step 7: If a satisfactory RMSE value is met go to step 8, otherwise store the new global best position and error and go back to step 5.

Step 8: Using the best particle, set the weights and bias values for the ANN.

Step 9: Perform test data to evaluate the performance of the PSO trained ANN

As can be seen, a PSO trained ANN does not differ greatly from PSO in general. It is simply using PSO to train the ANN weights instead of Backpropagation. However, integrating a PSO trained ANN allows a user to take advantage of both of their strengths while eliminating weaknesses of using Backpropagation. The results of using this method as well as the Backpropagation trained ANN will be detailed in the next section.

VI. Results and Discussion

All 5,000 trials of Backpropagation and 57 trials of PSO-ANN were run on the same hardware setup using the same programming language. The system that was used is a 2013 MacBook Pro with a 2.3 GHz Intel Core i7 CPU, 16 GB of 1600 MHz DDR3 RAM, and an NVIDIA GeForce GT 750M 2048 MB video card. All tests were conducted by using a combination of Python, Keras, and Theano inside of the PyCharm IDE.

A. Backpropogation

After all of the trials were run as described in the previous section, the results were output into a file that was then analyzed to find the strongest RMSE results. Many of the results were of similar value in the middle 50s as can be seen in Fig. 3. In Fig. 3 it can be seen that the majority of the results are between 51 and 55 RMSE. Two of the primary activation functions were left out of the figure as they had large maximum values of approximately 400. However, their quartile and median values aligned very closely to the other three as well demonstrating how much of the data clustered around the 51 to 55 RMSE results.

The best RMSE values from each variant of activation function in between the input and hidden layer was placed into Table 1 along with the results using the same architecture with the 12 input method. The best of these RMSE values was 47.53. The architecture of this result contained the discussed 29 input method, a Hyper Tangent activation function connecting to the hidden layer, 39 hidden nodes, a Rectifier activation function between the hidden layer and the output, and one output node. This helps further solidify that it is better to

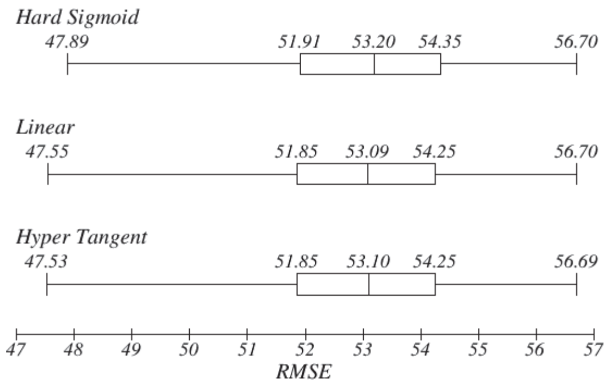


Fig. 3. RMSE values of three of the five primary activation functions with information of locations of the minimum, first quartile, median, third quartile, and maximum.

TABLE II
TOP PERFORMING ARCHITECTURE PER FIRST ACTIVATION FUNCTION. SECOND LISTED ACTIVATION FUNCTION IS THE FUNCTION BETWEEN HIDDEN LAYER AND OUTPUT. H REPRESENTS NUMBER OF HIDDEN LAYER NODES.

Artificial Neural Network Architectures and RMSE Values				
Activation	**H-Nodes**	**Activation**	**RMSE-29**	**RMSE-12**
Hard Sigmoid	142	Hyper Tan	47.88	54.65
Linear	41	Rectifier	47.54	52.03
Rectifier	122	Sigmoid	48.16	55.30
Sigmoid	142	Hyper Tan	47.87	53.13
Hyper Tan	39	Rectifier	47.53	51.33

break discrete categories into their own nodes when designing networks that have smaller datasets with skewed outputs.

The result of an RMSE value of 47.53 is a fairly substantial improvement over previous works that hovered around 60 RMSE. This equates to roughly a 21% improvement in results, which is a substantial amount. There is a possibility that RMSE values could be further lowered with use of multiple hidden layers or more nodes. Although, it is unlikely that results will be substantially better as 5,000 ANN configurations using Backpropagation were tested with peak results not being very distant from the 54.91 average. It is also that the 12 input value tests showed improvement over previous works. This could be due to the optimization algorithm used within the Keras library.

B. PSO

The PSO trained ANN results showed further improvement over the Backpropagation tests. As can be seen in Fig. 3 the lowest RMSE value found using a PSO trained ANN was 16.40. This is a tremendous improvement even from the previously improved Backpropagation results. The method that achieved this result contained 250 particles and 34 nodes inside of its hidden layer. Once again a trial with 12 data inputs was also run on the same architecture and it was found that the RMSE value was 24.7. This aligns with the slight increases seen in the Backpropagation as well.

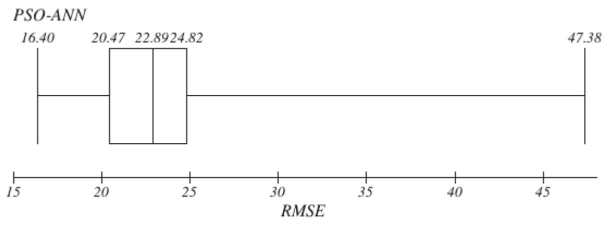


Fig. 4. PSO-ANN 29 input method minimum, first quartile, median, third quartile, and Maximum.

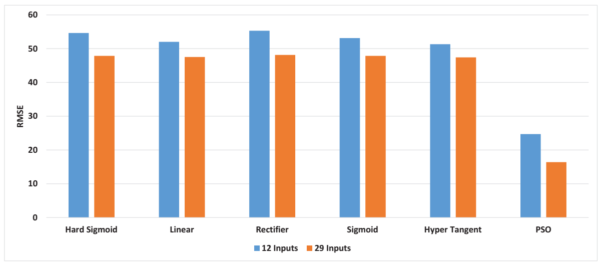


Fig. 5. RMSE values using the 12 and 29 input methods of the best performing architectures found.

The quartile data seen in Fig. 4 demonstrates that once again the data was fairly clustered not being heavily effected by the amount of nodes that were used in the hidden layer with values of 20.47 and 24.82 for the first and third quartiles respectively.

Fig. 5 gives an overview of all of the RMSE values of the notable ANN architectures created in this paper. It can clearly be seen how much more favorably a PSO trained ANN performs versus the traditional Backpropagation using this paper's methodology. In fact, the PSO performs approximately 190 percent better than the Backpropagation in training the ANN for this specific regression task.

VII. CONCLUSION

The Montesinho dataset has proven to be a difficult regression task. It is relatively small as well as heavily skewed. These factors have resulted in fairly high RMSE values in previous research efforts.

Using a novel input method and training structure for the Montesinho dataset has netted much better RMSE results. Using PSO to train an ANN for the regression task of predicting the burned area of a forest fire has netted a roughly 260 percent decrease in RMSE. This is clearly a large leap forward from previous results. The PSO algorithm itself was the primary contributor to these results. However, breaking apart discrete valued variables into their own input nodes further reduced RMSE values. Giving each discrete input value its own node consistently decreased RMSE values throughout all tests.

It is believed that results can be further improved through deep learning methods and parallel metaheuristics. Using larger amounts of particles, hidden layers, and nodes is thought to be able to further decrease RMSE values. As ANNs and the amount of particles grow in size it takes considerable amounts of time to train and calculate RMSE values. If deep learning is to be accomplished in a reasonable amount of time the training of the ANN must be parallelized. Fortunately, parallelizing the PSO algorithm is fairly intuitive as each particle can be executed independently. When all particles on their multiple threads are finished executing, their RMSE values can be compared and this cycle can then continue until an error threshold or iteration maximum is reached.

References

[1] K. Malarz, S. Kaczanowska, and K. Kolakowski, "Are forest fires predictable?" *International Journal of Modern Physics*, vol. 13, no. 8, pp. 1017–1037, 2002.

[2] V. Wagner, "Development and structure of the Canadian forest fire weather index system," Petawawa National Forestry Institute, Tech. Rep., 1987.

[3] J. M. Piçarra, J. C. Gutiérrez-Marco, A. A. Sá, C. Meireles, and E. González-Clavijo, "Silurian graptolite biostratigraphy of the Galicia - Tras-os-Montes Zone (Spain and Portugal)," June 2006.

[4] S. Taylor and M. Alexander, "Science, technology, and human factors in fire danger rating: The canadian experience," *International Journal of Wildland Fire*, vol. 15, pp. 121–135, 2006.

[5] P. Cortez and A. Morais, "A data mining approach to predict forest fires using meteorological data," in *Portuguese Conference on Artificial Intelligence*, Guimares, Portugal, Dec 2007, pp. 512–523.

[6] Y. Safi and A. Bournoumi, "Prediction of forest fires using artificial neural networks," *Applied Mathematical Sciences*, vol. 7, no. 6, pp. 271–286, 2013.

[7] M. Castelli, L. Vanneschi, and A. Popovic, "Predicting burned areas of forest fires: An artificial intelligence approach," *Fire Ecology*, no. 11, pp. 106–118, January 2015.

[8] S. Jain and M. Bhatia, "Performance investigation of support vector regression using meteorological data," *International journal of database theory and application*, vol. 6, pp. 109–118, 2013.

[9] Y. Wang, J. Wang, W. Du, C. Wang, Y. Liang, C. Zhou, and L. Huang, "Immune particle swarm optimization for support vector regression on forest fire prediction."

[10] J. Hunt and D. Cooke, "Learning using an artificial immune system," *Journal of Network and Computer Applications*, vol. 19, pp. 189–212, April 1996.

[11] L. Xue-tong, "Oil price forecasting based on particle swarm neural network," *Seventh International Conference on Measuring Technology and Mechatronics Automation*, pp. 712–715, 2015.

[12] K. Chau, "Particle swarm otpimization training algorithm for ANNs in stage prediction of Shing Mun River," *Journal of Hydrology*, vol. 329, pp. 363–367, 2006.

[13] N. Mohammadi and S. Mirabedini, "Comparison of particle swarm optimization and backproagation algorithms for training feedforward neural network," *Journal of Mathematics and Computer Science*, vol. 12, pp. 113–123, August 2014.

[14] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[15] B. Niu and L. Li, "A hybrid particle swarm optimization for feed-forward neural network training," in *International Conference on Intelligent Computing*, D.-S. H. et al., Ed. Springer–Verlag, 2008, pp. 494–501.

[16] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *IEEE International Conference on Neural Networks*, vol. 4, November 1995, pp. 1942–1948.

[17] Y. Shi and R. C. Eberhart, "Empirical study of particle swarm optimization," in *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, vol. 3, 1999, p. 1950 Vol. 3.

[18] J. Kennedy and R. C. Eberhart, *Swarm Intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001.

[19] P. McCullagh and J. Nelder, *Generalized linear models*, 2nd ed. Chapman & Hall, 1989.

[20] M. Finke and K.-R. Mller, "Estimating A-Posteriori Probabilities Using Stochastic Network Models," in *Summer School on Neural Networks*, 1994, pp. 276–281.

[21] S. Menard, *Applied logistic regression analysis*, 2nd ed. SAGE, 2001.

[22] P. Charles, "Keras," https://github.com/fchollet/keras, 2013.

[23] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. J. Goodfellow, A. Bergeron, N. Bouchard, and Y. Bengio, "Theano: new features and speed improvements," Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[24] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: a CPU and GPU Math Expression Compiler," in *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010, oral Presentation.