Project 4 Design Document

CSCI 447
11/22/2019

Andrew Kirby
Kevin Browder
Nathan Stouffer
Eric Kempf

Project Explanation

Neural Networks can be trained in many ways. In project 3, backpropagation was implemented to train a feed-forward Multi-Layer Perceptron (MLP). In this project, three different population based training algorithms will be used to train a feed-forward network. These training algorithms are the Evolutionary Algorithm (EA), Differential Evolution (DE), and Particle Swarm Optimization (PSO). The goal is to test the performances of population based trainers versus backpropagation. So, a MLP will be trained using each of the algorithms and then compared to the results from project 3.

Class Descriptions

DataReader: This class takes in a file name and reads in the data to instances of Set classes. The data is assumed to be in a standard format that a DataReader object can process.
Set: A Set is composed of a group of examples (data points). This will be used to store examples so that our learning algorithm can be trained and tested.
INeuralNet: This interface defines the methods required to implement a Neural Network. For the purposes of this project, there is only one such class: *Multi-Layer Perceptron*. A MLP can have any number of layers and any number of hidden nodes.
Layer: The Layer class composes networks. Each layer consists of a weight matrix and an activation function. An input vector can be passed into a layer to produce output, which is then fed into the next layer.
IActFunct: This interface defines the methods used in an activation function. There are two types: *Sigmoidal* and *Linear*. Each take in a Vector and compute the activation of each component in the vector.
IPopTrain: The IPopTrain interface defines the methods required to implement a population based method to train a Neural Network. There are three types of population trainers that will be implemented: *Genetic Algorithm*, *Differential Evolution* and *Particle Swarm Optimization.* Each have their own train method that implements the specifics of each type of training.

Algorithm Design

The population based trainers are an implementation of the Strategy Pattern. Each of them implements the IPopTrain Interface and performs the same functionality of training a network, but a different method is used. Each population based algorithm will contain a population consisting of complete feedforward networks parameterized by the weights of their layers. For ease of manipulation, the weights of a feedforward network will be converted into a

single vector of weight values. Each algorithm will be performing successive manipulations on these vectors (or groups of vectors) in search of the optimal feedforward network. Whenever the fitness of a network must be evaluated, the vector will be converted back to a network structure. At the termination of each algorithm, an instantiation of the feedforward network with the best fitness is returned.

The Genetic Algorithm

The Genetic Algorithm relies on a cyclic process of selection, recombination, mutation, and replacement. At the start, individuals with random, small weights are initialized. From the population, individuals are selected for recombination, during which they are crossed in some way with other selected individuals. The resulting individuals are mutated and replace all or some of the members in the current population. Note that each stage in this process has a variety of methods available for implementation. The total possible combinations of these options would be impractical to implement and tune, likely with diminishing returns. Thus, decisions about the algorithm will be made to balance simplicity and expected performance.

Population size is directly related to the algorithm's ability to search the hyperspace for the optimal solution. The N^3 argument states that a population of size N will usefully search a N^3 dimensional hyperplane (Whitley 1994). Given the number of weights in feedforward networks and the range of possible real values each may take on, the population size will need to be large in order to search this high dimensionality effectively. Some sacrifices must be made for the sake of practical implementation, so the population size will start at 64, with opportunity for increase if desired performance is not obtained.

Selection of progenitors will be implemented with rank-based selection. The probability of selecting a particular individual *x* is given by $P(\vec{x}) = \frac{2}{|P|}(\frac{|P| - rank(\vec{x})}{|P|+1})$. This method balances the high selection pressure of Fitness Proportionate Selection or Tournament Selection. Rank-based selection is expected to provide practical convergence times while supporting diversity in the population. A bonus is that this method mitigates the large decrease in selective pressure as the members of the population approach similar fitnesses (Whitley 1994). Additionally, rank-based selection does not require tuning.

Recombination is performed using uniform crossover—a uniform chance, P_c, of swapping weights at any pair. Considered more disruptive than its counterparts, this feature may actually be helpful as a stimulus in low population scenarios (Whitley 1994).

Mutation will also be performed uniformly across the individual with rate P_m. Because the weights are real-valued, a weight x_i will creep according $x_i^{'} = x_i + N(0, \sigma_i)$, where σ_i is determined by the current standard deviation of the corresponding weights in the population. This will facilitate an appropriate resolution to the global search.

Children that are generated via recombination and mutation must be introduced back into the population. In order to balance exploration and exploitation, a steady state replacement

method will be implemented. Half the population size in children will be generated, replacing the individuals in the parent population at random. This will support selection's survival of the fittest concept while not over-pressuring loss in diversity.

Differential Evolution

The Differential Evolution algorithm relies on the same cyclic process of selection, recombination, mutation, and replacement as the Genetic Algorithm.

Because of the small population we want mutation and crossover to be fairly disruptive so that searching is more effective. DE/rand/2/bin will be used because two difference vectors results in better perturbation than a single difference vector. The random mutation strategy has less selection pressure and will therefore perform better on multimodal problems (Qin 2009). Since this algorithm needs to be run on six very different data sets, favoring exploration should result in better performance across all data sets. Binomial crossover is utilized because it is a standard for DE algorithms (Qin 2009). The target vector will be replaced with the trial vector if the trial vector performs better than the target vector.

For the purpose of maintaining uniformity across the three algorithms, a generation will end when the number of children generated is equal to the population size N. Target vectors will be randomly selected from the population.

Particle Swarm Optimization

PSO differs from the GA and DE in that there are no generations. Instead, PSO consists of particles that interact and move around the search space to find the optimum solution. Each particle is influenced by its previous experience as well as the experience of all other particles. Within reason, the number of particles in a population does not affect the performance of PSO. Thus, $N = 60$ particles will be instantiated (Shi 1999). First all particles are initialized at random locations in the search space. Then each particle is updated according to the following rule $x(t+1) = x(t) + v(t)$. Thus the update to the new position of each particle depends entirely on the velocity. This is computed as

$$v(t) = \omega * v(t-1) + c_1 * r_1 * (\text{pBest} - x(t)) + c_2 * r_2 * (\text{gBest} - x(t))$$

where $r_1, r_2 \in [0, 1]$ are generated randomly, $c_1, c_2 > 0$ are tuned (Eberhart 1995). The value of ω can also be a tuned parameter, however, there are already three tuned parameters (number of layers in addition to c_1 and c_2), so ω will start at 1 and linearly decrease to 0.4 over the course of the run. This has been shown empirically to perform better than a constant ω (Shi 1998). The terms pBest and gBest must still be defined. The value of pBest is the position of the current particle with the best performance and the value of gBest is the position of the particle with the best performance of all particles (Eberhart 1995).

Velocity updates will be added to the position vectors until a number of maximum iterations is reached.

Experimental Design

Design Decisions

In order to compare the results of this project to the results of project 3, the topology of our Neural Networks must be consistent between projects. As such, we will test each of the three population based algorithms with 0, 1, and 2 hidden layers in the network. To determine the number of nodes in a hidden layer of a MLP, we will multiply the dimension of the feature space by 2, as tested previously.

Validation

All learning algorithms will be tested on every data set. Regression and Classification will be determined as needed. Each test will involve ten-fold cross validation, which requires separate training and test sets to be allocated during preprocessing. Ten-fold cross validation will also be used while tuning.

Tuning

Tuning the GA will primarily focus on the critical parameters of crossover and mutation rates, P_c and P_m, given the choices made above. P_c will be tuned by testing uniform values between 0 and 0.5, the possible range of probabilities for this uniform crossover. P_m will be tuned uniformly in the range of 0 to 0.02. Mutation rate is typically less than 0.01 for GA algorithms (Whitley 1994). If satisfactory performance is not being obtained with the population size of 64, this value may be increased gradually.

Tuning of the DE is primarily concerned with the crossover and mutation scaling factor parameters. Crossover (P_r) will be tuned between 0.3 and 0.9 (Qin 2009). The mutation scaling factor (□) will be tuned between 0 and 2 non inclusive.

For the PSO, there are two parameters to tune: c_1 and c_2. The community generally accepts $c_1 = c_2 = 2$ (Shi 1999). However, the value of c_1 and c_2 are highly related. So for tuning, perform a grid search of values $\{1, 2, 3\}$ for each of c_1, c_2 and evaluate the performance of the returned network. Choose the best pair of c_1, c_2.

Evaluation Metrics

Mean Squared Error (MSE), and accuracy will be used to evaluate the classification problems. Classes do not have real values associated with them, so MSE is calculated as the square root of the sum of squared differences between total actual examples in a class and total predicted examples in a given class. In this case, MSE is an indicator of how far off the predicted distribution is from the actual distribution. Accuracy indicates how well the algorithm is classifying examples on an individual basis. These two metrics supplement each other because it

is possible to get a MSE of 0 and incorrectly classify every testing example. Using both metrics gives a better evaluation of an algorithms performance.

Mean Error (ME) and MSE will be used to evaluate the regression problems. MSE takes the distance between real and predicted values and squares it. ME is similar but takes the value of the distance between real and predicted values. One issue with MSE is that if the error is less than 1 the performance is underestimated and if the error is greater than 1 the performance is overestimated. This can be addressed by using ME since the sign of metric will show if the MSE is over or underestimating the performance of the algorithm. An issue encountered in previous projects is that both ME and MSE are proportional to the real-valued outputs. This means evaluating these metrics requires that the context (the scale of the outputs) is known. This will be accounted for in this project by computing the MAE and MSE of the z-scores. The mean and standard deviations used to compute the z-score will be computed based on the testing set.

References

Eberhart, R. and Kennedy, J., 1995, October. A new optimizer using particle swarm theory. In *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science* (pp. 39-43). IEEE.

Shi, Y. and Eberhart, R., 1998, May. A modified particle swarm optimizer. In *1998 IEEE international conference on evolutionary computation proceedings. IEEE world congress on computational intelligence (Cat. No. 98TH8360)* (pp. 69-73). IEEE.

Shi, Y. and Eberhart, R.C., 1999, July. Empirical study of particle swarm optimization. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)* (Vol. 3, pp. 1945-1950). IEEE.

Whitley, D. A genetic algorithm tutorial. *Statistics and Computing*, 4(2):65-85, June 1994. ISSN 1573-1375.

Qin A. K., Huang V. L., and Suganthan P. N., "Differential Evolution Algorithm With Strategy Adaptation for Global Numerical Optimization," in *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 2, pp. 398-417, April 2009.