Genetic Algorithms and Higher Order Perceptrons

Tim Andersen and Tony Martinez
Neural Net and Machine Learning Laboratory: http://axon. cs. byu. edu
Computer Science Dept. Brigham Young University

Abstract

Constructive induction, which is defined to be the process of constructing new and useful features from existing ones, has been extensively studied in the literature. Since the number of possible high order featuresfor any given leaming problem is exponential in the number of input attributes (where the order of a feature is defined to be the number of attributes of which it is composed), the main problem faced by constructive induction is in choosing which features to use out of this exponentially large set of potential features. For any feature set chosen the desirable characteristics are minimality and generalization performance. Two reasons for selecting small feature sets are: 1) pruning unneeded inputs allows for a savings in computational complexity during execution, 2) when choosing between two (or more) possible explanations for a given problem, the simplest is the one which will most likely produce the best generalization results. This paper uses a combination of genetic algorithms and linear programming techniques to generate feature sets. The genetic based search minimizes the size of the feature set while at the same time producing feature sets with good generalization acurracy. The features chosen are used as inputs to a high order perceptron network, which is trained with an interior point linear programming method. A critical element of the algorithm is the selection of an appropriate fitness and objective function to train the network. Several fitness/objective functions are tested and compared. The results show that the HOP is capable outperforming a multilayer backpropagation network.

1. Introduction

The problem of constructive induction has been extensively studied in the literature (Elio 91; Pagallo 90; Wnek 94; Quinlam 86; Raedt 92; Donoho 94; Michalski 86; Clark 90). Essentially, constructive induction is an attempt to derive new amd useful concepts from existing concepts. The standard approach to inductive learning is to take a training set T of examples $e_1, e_2, \ldots, e_n$, where each $e_i \in T$ is composed of a vector of input attributes $\boldsymbol{a}$ and an associated output classification z, and present this to a learning algorithm λ which then generates an hypothesis h that represents the classification of the examples in T. The goat is to find an hynothesis that minimizes classification error. With constructive induction the question that is asked is, is it possible to create a new feature from the original features of T which will improve the hypothesis produced by λ?

In other words, given a training set of examples T= $\{e_1, e_2, \ldots e_n)$, where each example is composed of a vector of features $a_1 a_2 \ldots a_m$, and an inductive learning algorithm λ, the goal of constructive induction is to create a new feature a_{m+1} which is some function g of the original features in the training set, such that the new hypothesis h' produced by λ using feature a_{m+1} has less error than the previous hypothesis h. We call the new features constructed by the learning algorithm higher order features, and we define the order of a feature to be the number of original input features of which it is composed. For example, $a_{m+1} = g(a_i, a_k)$ is a 2nd order feature, since it is a function of 2 of the original input attributes. This process of constructing new features continues until sonne acceptable level of error is achieved on the training set or until no further gain can be produced by the addition of new features.

The main problem faced by constructive induction is in finding amd selecting good features. Since the number of possible features for any given learning problem is exponential in the number of input attributes, all of the feature construcuon methods found in the literature use some form of heuristic(s) in order to limit the size of the search space examined by the learning aigorithm. The heuristics which are used to limit the search often make it impossible to guarantee that the aigorithm will converge to a solution.

One area of interest in constructive induction is that of higher order perceptrons (HOPs). A HOP is a network composed of perceptron type node(s) which has tbe set of original 1st order inputs augmented by adding new features which are higher order combinations of the 1st order inputs.

The function which is most often used to combine the inputs is the multiplication function, but any nonlinear function can be used. Theoretically, given the right set of higher order features a HOP is capable of learning any problem. The problem is that the number of higher order features grows exponentially with the number of 1st order inputs, and so to search through all of the features in this space is impractical.

This paper tests the ability of genetic algorithms to search through the exponentially large feature space and find good higher order features for HOPs. Genetic algorithms (GAs) use the same principles foumd in population genetics, such as crossover, mutation, and survival of the fittest, to search through a large space of possibilities in order to find the fittest individual. GAs have been successfully applied to several neural network construction ;and optimization problems (Whitley 89; Harp 89; Varria 91).

In addition to utilizing GAs, this paper also uses an interior point linear programming (LP) method to calculate feature weights for the HOP. By utilizing an LP for calculating the feature weights, this opens up the possibility of tuning the objective function to avoid over learning. Greater detail is given on this in section 2.

Tests are conducted on 4 real world data sets taken from the UCI machine learning database repository. The results show that a GA/LP approach is capable of generating small feature sets for a HOP which outperform a multilayer neural network trained with backpropagation.

Section 2 discusses how to avoid over learning and why this is important. Section 3 gives the basic GA search technique used in this paper. Section 4 discusses the experiments and results. The conclusion is given in section 5.

2. Avoiding Overlearning

With any network construction technique, the avoidance of over learning, often referred to as overfitting, is of critical importance. If the network construction process is allowed to proceed until all training examples are correctly classified, then it is highly likely that the network will have "over learned" the problem and that it will perform poorly at classifying novel examples. By saying that a network has over learned a problem. we mean that the network has memorized a feature which is unique to some (or perhaps a few) example(s) in its training set in order for the network to be able to correctly classify that example. However, since the feature occurred so infrequently in the training data, it is impossible to say that the correlation the network has memorized is statistically valid, and so when it is used to classify novel examples it will be about as likely (from a statistical standpoint) to produce errors as it will be to correctly classify the novel examples.

There are several methods for avoiding over learning, some of which cam be found in (Vapnik 82; Rissanen 86, Craven 79). All of these can be summarized under two basic approaches which are 1. complexity vs accuracy tradeoff and 2. using a holdout set to determine when to stop. For this paper, we employ both the use of a holdout set and a GA fitness function which uses a simple training set accuracy minus complexity tradeoff to prevent over learning. In addition to the GA fitness function, we also incorporate an element into the LP objective function which helps eliminate useless features. All of these methods are explained in greater detail below.

2.1. The GA Fitness Function

The primary goal of the learning algorithm is to produce a system which has the greatest possible generalization accuracy. A secondary consideration is that the system which the learning algorithm produces should be as simple as possible. There are many simplicity measures which could be used, but in terms of a HOP simplicity is generally taken to be the number of features, or perhaps the sum of the order of each feature.

One way of accomplishing these two goals with a GA based approach is through the selection of an appropriate fitness function. The tendency for most network construction techniques is that the algorithm will over learn, producing a solution which is both more complex than what is needed and correspondingly poor at correctly predicting the

outcome of novel examples. In order to combat this tendency, the fitness function of the GA must achieve an appropriate balance between accuracy on the training set and network complexity (number of features). For example, the fitness function could be accuracy minus the number of features. With this fitness function, features which did not increase classification accuracy by more than 1 percent would be selected against. The drawback to this fitness measure occurs when there are higher order or synergistic interactions between several features such that any one of the features taken individually does not give the required increase in classification accuracy, but taken collectively the features bring about a large increase in classification accuracy. This problem can be at least partially compensated for by the genetic algorithm since it is searching for higher order features, and so the algorithm may find a single higher order feature that is a combination of the requisite lower order features and thus subsumes any possible interaction.

2.2. Using a Holdout Set

Another approach to avoid over learning is to use performance of the HOP on a holdout set to attempt to predict how well it will perform on novel examples The drawback to this approach is that it requires that part of the training set be held back during the learning phase, and with fewer training examples to constrain the search space the solution the learning algorithm generates may not be as good as would otherwise be possible. A second problem which is often observed is that performance on the holdout set may not correlate as strongly as desired with performance on novel examples. The approach used in this paper is to use the holdout set to help decide which set of features to use, but after the final set of features is selected the holdout set is returned to the training set for the final calculation of feature weights.

2.3. The LP Objective Function

This paper also uses other approaches to avoid over learning, one of which is based upon linear programming (LP). Instead of using the genetic algorithm to adjust all parameters, an interior point LP method is used to calculate feature weights for the HOP. This shifts the burden of finding weight settings from the GA and allows it to focus on finding good features. In order to calculate feature weights, the objective function for the LP can be designed to either minimize the sum of the error (distance from the separating hyperplane) of all misclassified examples, or to minimize the maximum error of the worst misclassified example.

In addition to finding good weight settings. the objective function can be designed to "zero out" redundant features, or features that are not greatly contributing to training set accuracy. One way to do this is to include the weights as part of the objective function. The balance between weight magnitude and error will then force features which are not contributing to the reduction of the objective function to have a weight of zero, and these can then be removed from the feature set.

3. Basic Algorithm

The basic algorithm uses the GA operator of mutation as the means for creating new individuals in the population. The individuals in the population are prospective sets of features for the HOP. Each individual or set of features is evaluated according to some fitness measure which is essentially an estimate of that feature set's ability to correctly classify examples from the problem domain. The fitness functions used in this paper are discussed in greater detail in section 4.

Pseudo code for the basic algorithm is given in Figure 1. The surviving individuals in the population are first mutated by adding random features to each individual. The new individuals so created are allowed to compete with the current set of individuals, with the top 50 percent of the population surviving. The next step mutates the surviving individuals by removing random features. Again, the children compete with their parents and the top 50 percent survive to the next generation. This continues as long as improvement (as measured by the fitness function) in the performance of the top individual in the population has occurred within the last 5 iterations. If improvement has not occurred in the last 5 iterations a last ditch attempt is made to improve the performance of the top individual by exhaustively looking at each feature to see if it can be removed without causing a decrease in fitness. This is repeated until no more features can be removed from the top individual. If the exhaustive search is unable to remove any features the GA search is terminated

```
while there has been improvement within the last 5 iterations
{
        Create children by adding random features to population.
        Calculate feature weights via the LP.
        Sort the population according to fttness.
        The top 50% of the population survives.
        Create children by removing random features from the population.
        Calculate feature weights via LP.
        Sort the population according to fitness.
        The top 50% of the population survives.
        If no improvement has have occured in the last few iterations
        {
                Exhaustively search for features which can be removed from
                the best individual without hurting fitness amd remove them.
        }
}
Select the most fit individual in the population as the fmal solution.
Calculate feature weights via the LP using all available training examples.
```

Figure 1. GA search technique.

A holdout set is used to help the GA decide when to stop adding new features. Performance on the holdout set is part of the GA fitness function, but the holdout set is

not used by the LP to calculate feature weights during the GA search. In this way, the holdout set helps to provide a reliable estimate of the generalization capability of the feature set while at the same time helping to guide the selection of which features to use in that set. After termination of the GA search, the holdout set is returned to the training set and used by the LP to calculate the final feature weights. In this way the LP has as much information as possible available to it in the final estimation of the relative importance of each feature.

4. Experiments

For the experiments various combinations of fitness and objective functions were tested to determine which combinations worked best. Two different LP objective functions were tried. The first objective function (which we label "d") minimizes the distance of the misclassified examples to the decision boundary, and the other (which we label "w") is similar but includes the feature weights as part of the objective function in an attempt to excise unneeded features by forcing their weights to zero.

Also, several different fitness functions were tested. The most useful of these are:

2 * Holdout + Training (2ht)

2 * Holdout + Training - #features (2htf)

The term in parenthesis is a short hand designation for the given fitness function. In order to distinguish which objective function was used to calculate the feature weights, an underline followed by a_d (for minimizing the distance) or a_w (for including feature weights) is appended to the fitness function. For example, ht_w indicates a fitness function of holdout + training set accuracy and an LP objective function that includes feature weights.

Each combination was tested using 10-fold cross validation on four different real world data sets taken from the UCI machine learning database. These four data sets are sonar, echocardiogram, breast cancer, and liver. The final results are compared against those obtained with a multilayer neural network trained with backpropagation and a single layer network trained via the well known delta rule training method.

Figure 2 shows the results obtained on the test set when running the GA using 2htf_d, and Figure 3 shows the results obtained using 2htf_w. In each of the figures, the results obtained with the multilayer network (bp) and single layer network (per) are included for comparison purposes. The results obtained with 2htf_w are significantly better. This result is likely due to not giving features enough weight in the fitness function, and so the fitness function by itself is unable to properly minimize the number of features for maximum generalization accuracy.

Data sets	2htf_d	bp	per
echo	89.40	89.30	87.00
sonar	71.17	76.40	73.20
liver	67.20	69.00	66.40
breast-cancer	71.71	73.50	66.50
average	**74.87**	**77.05**	**73.28**

Figure 2. 2htf_d test set accuracy

Data sets	2htf_w	bp	per
echo	92.30	89.30	87.00
sonar	76.10	76.40	73.20
liver	69.13	69.00	66.40
breast-cancer	71.70	73.50	66.50
average	**77.31**	**77.05**	**73.28**

Figure 3. 2htf_w test set accuracy.

For example, the average test results which were obtained for 2htf_w (77.3%) are nearly identical to the average test scores for 2ht_w (reported in figure 4), even thought 2htf_w includes the complexity in the GA fitness function and 2ht_w does not.

Data sets	2ht_w	bp	per
echo	91.54	89.30	87.00
sonar	71.17	76.40	73.20
liver	71.29	69.00	66.40
breast-cancer	74.54	73.50	66.50
average	**77.14**	**77.05**	**73.28**

Figure 4. 2ht_w test set accuracy.

Figure 5 gives the average feature set size for each of the methods. From this figure it can be seen that the size of the feature set has a strong correlation with the degree of generalization accuracy, with smaller feature sets almost always producing better generalization results. In fact, if the data in figure 5 is used to select the fitness/objective function combination which produces the smallest feature set for each of the data sets, the average generalization performance can be increased by nearly 1 percent.

Data sets	2ht_w	2htf_w	2htf_d
echo	4.50	3.50	3.80
sonar	18.33	11.90	29.70
liver	7.40	5.80	5.80
breast-cancer	2.90	1.90	1.70
average	**8.28**	**5.78**	**10.25**

Figure 5. Feature set size.

Another interesting observation is that the number of features which are retained by 2htf_w is much less than the original number of features which occur in the respective training sets. For example, echocardiogram has 10 original inputs, sonar has 61, liver has 7, and breast-cancer has 10 original inputs. Almost all of the features retained by 2htf_w

are first order (original inputs), with on average less than 1 higher order (generally 2nd order) feature being retained by the GA. This leads to the conclusion that feature reduction is apparently more critical than feature construction in producing good generalization results, at least with the data sets tested here.

As can be seen from these results, the GA technique is capable of finding good feature sets which allow for a significant improvement in the performance of the single layer network, improving on its average performance by over 4 percent. Also, the HOP performs equivalently to the multilayer network, slightly improving on its average performance by about 0.1 percent when used in conjunction with 2htf_w.

5. Conclusion

The GA search technique which used the fitness/objective function of 2htf_w was able to outperform a multilayer backpropagation network by an average increase of 0.1 percent in classification accuracy on the data sets tested in this paper. Interestingly, it is able to do this while also significantly decreasing the size of the feature set, deleting several of the original 1st order features while adding slightly less than 1 2nd order feature on average.

These results follow those found in (Andersen and Martinez 93) which showed that using features which are greater than 2nd order is unlikely to increase classification accuracy, and will often lead to an overall decrease. The reasoning behind this empirical result is that higher order features typically occur less often in the training set (and test set) than low order features, and so it is difficult to reliably estimate their utility from the training data. In addition, since higher order features tend to be used less often during testing, there is only a small chance that a particular high order feature will actually help improve generalization accuracy.

The results showed that the fitness function by itself was too conservative when it included the number of features as a parameter to reduce the size of the feature set to an optimal level. This is an area where further research could be applied in order to improve the results obtained in this paper. For example, the minimum description length principle could be incorporated into the fitness function to obtain a more elegant solution which might perform better.

Another approach which could improve results would be to allow the GA to determine weight settings, rather than having these calculated by the LP. The LP was used in this paper because of its ability to quickly calculate optimal weight settings for the HOP, and because it allowed the weight calculation to be separate from the GA search enabling the use of the holdout set as part of the fitness function. However. while it calculates the optimal weight setting for the given objective function, it does not do so for the fitness function. Also, it is difficult to determine the interaction between the choice of the objective and fitness functions. So, it may be better to have a single function which is being optimized by the GA

The way that the GA searches for higher order features could also be modified to improve results. Currently, the GA only uses mutation to generate new individuals. Other genetic operators could be tested, such as various forms of crossover. In addition, methods could be employed to increase the genetic diversity of the population to help the GA to search a broader search space.

Future research will focus on

- refinement of the fitness function,
- refinement of the objective function,
- learning the GA calculate feature weights,
- and further refinement of the GA

6. References

Andersen, Timothy and Tony Martinez. Learning and Generalization with Bounded Order Critical Feature Sets. *Proceedings of the AI93 Australian Joint Conference on Artificial Intelligence*, pp 450, 1993.

Clark, P., and T. Niblett. The CN2 Induction Algorithm. *Machine Leaming*, Vol. 45, pp 304-307, 1990.

Craven, P., and G. Wahba. Smoothing Noisy Data with Sphne Functions. *Numer. Math.* vol 31, pp 376-403.

Donoho Steven K. and David C. Wilkins. Using Apprenticeship Techniques to Guide Constructive Induction. *Knowledge Acquisition*, vol 6, pp 295-314, 1994.

Elio, Renee and Larry Watanabe. An Incremental Deducttve Strategy for Controlling Constructive Induction in Leaming from Examples. *Machine Learning*, vol 7, pp 7-44, 1991.

Harp, Steven, Tariq Samad, and Aloke Guha. Towards the Genetic Synthesis of Neural Networks. *Proceedings of the 3rd International Conference on Genetic Algorithms*, pp 360-369, 1989.

Karmarkar, N. A new polynomial-time algorithm for linear programming. *Combinatorica*, vol 4, pp. 373-395,

Michalski, R. S., L. Mozerdc, J. Hong, and N. Lavrac. The Multi-purpose Incremental Leaming System AQ15 and its Testing Application to Three Medical Domains. *Proceedtngs of the Fifth National Conference on Artificial Intelligence*, pp. 1041-1045, Philadelphia: Morgan Kaufmann, 1986.

Pagallo, Giolia and David Haussler. Boolean Feature Discovery in Empirical Leaming. *Machine Leaming* vol 5, pp 71-99, 1990.

Raedt, L. D. and Maurice Bruynooghe. Interactive Concept-Learning and Constructive Induction by Analogy. *Machine Learning*, vol 8, pp 107-150, 1992.

Redding, Nicholas, Adam Kowalczyk and Tom Downs. Constructive Higher-Order Network Algorithm That Is Polynomial Time. *Neural Networks* vol 6, pp 997-1010, 1993.

Rissanen, J. Stochastic Complexity and Modeling. *Ann. Statist.* vol 14, pp 1080-1100, 1986.

Vapnik, V. *Estimation of Dependences Based on Empirical Data*. New York: Springer-Verlag 1982.

Whitley, Darrel and Thomas Hanson. Optimizing Neural Networks Using Faster, More Accurate Genetic Search. *Proceedings of the 3rd Intemational Conference on Genetic Algorithms*, pp 391-396, 1989.

Wnek, Janusz and Ryszard S. Michalski. Hypothesis-Driven Constructive Induction in AQ17-HCI: A Method and Experiments. *Machine Learning*, vol 14, pp 139-168, 1994.