Constructing High Order Perceptrons with Genetic Algrithms

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 selecting 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. This paper uses a combination of genetic algorithms and linear programming techniques to generate feature sets. The genetic algorithm searches for higher order features while at the same time seeking to minimizes the size of the feature set in order to produce a feature set with good generalization accuracy. The features chosen are used as inputs to a high order perceptron network, which is trained with an interior point linear programming method. Performance on a holdout set is used in conjunction with complexity penalization in order to insure that the final feature set generated by the genetic algorithm does not overfit the training data.

1. Introduction

In the standard approach to inductive learning we are given a set of training examples, where each example is composed of a finite vector of input features (or attributes) and an associated output classification. The set of training examples *T* are presented to a learning algorithm which generates an hypothesis that represents the classification of the examples in *T*, with the goal being to find an hypothesis that minimizes classification error. With this approach the vector of input features is fixed, and no attempt is made to improve the performance of the learning algorithm throagh augmentation of the feature vector. This leads to the following question, which encapsulates the motivation behind what has been termed constructive induction. Is it possible to improve the hypothesis generated by the learning algorithm by augmenting or changing the vector of input features in the traimng set through the addition of a new feature (or features), where| the new feature is some function of the original features found in the training set. Essentially, constructive induction is an attempt to construct new and useful concepts from existing ones.

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.

There has been a great deal of research done in this area with some good results [4] [6] |7] [13] [15] [17] [26]. One particular area of interest in constructive induction is that of higher order perceptrons (HOPs) [8] [12] [16] [24]. A HOP is a network composed of perceptron type node(s) which has the 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 multiplication, but any nonlinear function can be used. Theoretically, given the right set of higher order features a HOP is capable of learning any classification problem.

There are two main problems faced by any HOP construction algorithm. The first problem is the process of constructing or searching for new, useful features, and the second is the problem of determining which features out of the potentially many features which are examined will be included in the final set of features. There are several problems which contribute to make this process difficult.

- The number of higher order features is exponential in the number of original input features. This makes it impractical to examine the entire set of potential features.

- Given any set of features, it is difficult to determine the optimum subset for a given learning problem. For example, determining the minimum size set of features (MIN FS) which solves a given learning problem is NP-Complete [1]. The minimum size set of features which solves the leaming problem is generally considered to be the best or most likely set of features for the minimization of classification error.

- Given a set of prospective features, the problem of determining the subset which has the best classificafion accuracy (MAX ACC) on the training set is NP-Complete in the number of features in the prospective set of features [10].

If it were possible to generate and store all possible higher order featares, then the process of coristructing the feature set for a HOP would amount to feature selection, and we could apply any one of the many available feature selection techniques [3] [14] [20]

[21] to generate an optimal or near optimal feature set. Unfortunately, the exponential size of the set of all possible features makes this approach impractical, and so we must turn our attention towards other approaches.

Genetic algorithms (GAs) have been used with some success on the problem of feature selection [20], and they have proven adept at finding near optimal solutions for problems which have extremely large search spaces. The approach which we propose is to use a GA to both generate new features and to select which subset of these features to use in the final solution. This paper tests the ability of a GA to search through the exponentially large feature space and find good higher order features for HOPs. GAs have been successfully applied to several neural network construction and optimization problems [9] [23] [25], including feature selection. This paper is an extension of earlier work [2]. In particular, the algorithm proposed in this paper

- uses a new fitness function to improve results,
- uses 10 fold cross validation with a holdout set to avoid the over fitting problem,
- uses a different objective function to calculate feature weights,
- and the number of data sets tested is increased.

Tests are conducted on 8 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 can 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 Over learning

With any network construction technique, the avoidance of over learning, often referred to as overfitting, is of critical rmportance. If the network construction process is allowed to proceed until all training exarnples 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 leamed a problem, we mean that the network has assigned some level of importance to a feature which is unique to one (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 can be found in [5] [19] [22]. All of these methods can be summarized under two basic approaches which are 1. a complexity vs accuracy tradeoff, and 2. the use of a holdout set to determine at what point to stop adding new features. For this paper, we employ a method which incorporates both of these strategies. The GA is used to generate prospective feature sets, which are then evaluated according to the GA fitness function, which is a simple measure that involves a tradeoff between accuracy on the training and holdout sets and hypothesis complexity.

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 and related consideration is that the system which the leaming algorithm produces should be as simple as possible. There are many simplicity measures which can be used, but in terms of a HOP, simplicity is generally taken to be either the total number of features or the sum of the order of the each feature.

One way of generating simple feature sets that exhibit good predictive accuracy with a GA based approach is through the selection of an appropriate fitness function. As has been said before, the tendency for most network construction techniques is for the algorithm to 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 must achieve an appropriate balance between accuracy on the training set and network complexity (number of features).

For example, the chosen fitness function could be accuracy minus tbe 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

An important method often employed to avoid over learning is to use performance on a holdout set to attempt to predict how well a model will perform on novel examples. The drawback to this approach is that it requires that part of the trainmg set be held back during the training 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 perforrnance on novel examples. This is especially true if the size of the holdout set is small in comparison with the size of the problem domain and the number of models tested. The use of some form of cross validation can help to alleviate this problem to a certain extent, but if the size of the available training set is small it will not completely remove it.

If the confidence that the holdout set results are a good indication of the model goodness is not 100 percent, then additional measures should probably be applied to prevent over learning. One approach which can be tried is to include the training set results as part of the model evaluation criteria. Of course, the reason for using a holdout set in the first place is to get away from the training set results, which are much more likely to lead to a solution which has overfit the data, and so it may seem that incorporating the training set results into the fitness function would be counter productive. Nevertheless, the training set is part of the overall function that the learning algorithm is trying to develop an hypothesis for, and so the results on this set could be pertinent.

The following example serves to illustrate this point. Suppose that there are two feature sets, f_1 and f_2, which are up for consideration. We evaluate f_1 and f_2 on the training and holdout sets and obtain the results shown in figure 1. If confdence in the holdout set result is 100 percent then f_1 will be the feature set of choice, since it has exhibited slightly better performance on the holdout set. However, the small (some would say insignificant) difference between the performance of f_1 and f_2 on the holdout set could shake our confidence in this conclusion. If performance on the holdout set is nearly the same, and f_2's performance on the training is much better, perhaps f_2 should be the preferred feature set.

Feature Set	Holdout	Training	Num Features
f1	88.6	89	3
f2	88.2	98	12

Figure 2. Feature set example.

Admittedly, this may be an extreme example. But this type of situation does sometimes arise when comparing two hypotheses, and it serves to show that there may be cases where looking at something besides the holdout set results will facilitate the selection of a better overall model.

Another element which should probably be included in any model evaluation criteria is model complexity, because over leaming can occur even when using a holdout set. The problem is that the greater the number of models that are being evaluated in comparison with the size of the holdout set, the more likely it is that a complex model will be found that by chance fits the holdout set very well but the overall function not so well. To avoid this, one should penalize the holdout set results in proportion to the model complexity. For example, if in addition to the results shown in figure 1 we knew that the

complexity (number of features) of feature set f_2 was much greater than that of f_1 (in this case 4 times greater) then we probably would not prefer it over f_1.

Since there are several factors which may influence confidence in the holdout set results, such as the degree of difference in the holdout set results, training set results, model complexity, holdout set size, and the number of models tested, it can be diffrcult to say how one should balance all of these factors to optimize the selection of the best model. With the GA approach used in this paper, several different feature sets are generated and the weights for these features are calculated using the training set, the holdout set results are then used to help guide the selection of the best feature set. In this case it is possible that the model can overfit the holdout set data, amd it makes sense to try and avoid this by penalizing models which are overly complex.

2.3. The LP Objective Function

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 is designed to minimize the sum of the error (distance from the separating hyper plane) of all misclassified examples. This insures that all training examples will be correctly classified if they are linearly separable. The error for a particular example is defined to be how far it is (in Euclidean terms) from being correctly classified. The LP objective function is formally defined as follows.

Minimize the sum of the errors.

Minimize (objective function): $z = \sum_{i=1}^{p} \rho_i + \sum_{i=p+1}^{m} \eta_i$

Subject to (constraints):

$$\sum_{j=1}^{n} \left(w_j x_{i,j} \right) + b - \rho_i \le -r \quad \text{for } i = 1,\ldots, p$$

$$\sum_{j=1}^{n} \left(w_j x_{i,j} \right) + b + \eta_i \ge r \quad \text{for } i = p+1,\ldots, m$$

Where:

ρ_i, η_i = error (positive in sign)
w_j = weight assigned to feature$j$
$x_{i,j}$ = value of feature j for example i
b = constant

r = small positive constant
$1,...,p$ = indices of negative examples
$p + 1,...,m$ = indices of positive examples

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.

Pseudo code for the basic algorithm is given in ftgure 2. 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 auempt 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 temove 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 2. 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 fttness function, but the holdout set is not used by the LP to calculate feature weights during the GA search. 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.

The fitness function which is used by the GA to evaluate individuals is a tradeoff between holdout and training set accuracies and complexity. The holdout set results are obtained by using 10-fold cross validation on the training set. The actual fttness function which the GA attempts to minimize is:

$$2*9h+t+ \text{lO}n+o$$

where h = number of errors on the 10 holdout sets,
r = number of errors on the 10 txaining sets,
n = the number of features in the feature set,
o = the sum of the order of each features.

We multiply h by 9 since $9|H| = |\text{T}|$, and so the number of errors in T is likely to be 9 times as great as the number of errors in H. In addition, this is multiplied by 2 to indicate a higher degree of confidence in tbe holdout set results.

In order to account for the complexity of the hypothesis, the number of features and the sum of the order of each feature is included. Since the addibon of a feature should decrease the number of errors encountered in a single training iteration by at least 1 on average, and h and t are obtained by summing errors over 10 separate training/test iterations, we multiply n by 10. It is difficult to say what the complexity of one feature is compared to another since there is nothing in the data sets to indicate the relative complexity of the original features. However, feature order is included as a small part of the fitness function, favoring feature sets which have lower order features, all other things being equal.

4. Experiments

For the experiments the GA_HOP construction technique was tested using 10-fold cross validation on several different real world data sets taken from the UCI machine learning database. These data sets are echocardiogram, breast-cancer-wisconsin, breast-cancer, bupa, credit, pima, and heart. For each of these data sets, the input features were normalized so that all data would fall between 0 and 1. The fnal results are compared against those obtained with a mulilayer neural network trained with backpropagation and

a single layer network trained via the well known delta rule training method. The results for backpropagation and perceptron are taken from [27].

Data Set	GA-HOP	bp	per
echo	92.30	89.30	87.00
b-cancer-w	95.90	96.30	93.00
breast cancer	74.90	73.50	66.50
bupa	70.40	69.00	66.40
credit	84.20	85.10	83.60
pima	76.70	75.80	74.60
heart	83.30	82.60	80.80
average	**82.53**	**81.66**	**78.84**

Figure 3. Test Set Results.

The test set results for these data sets is given in figure 3. The furst column indicates the data set being tested, the second column gives the results for GA HOP, the third gives the results obtained with a mult layer network trained with backpropagation, and the last column gives the results for a single layer perceptron network trained via delta-rule.

As can be seen, the GA HOP technique outperforms the single layer perceptron network on each of the data sets tested (95% confidence). In addition, GA_HOP outperforms a multilayer network on 5 of the 7 data sets (the confidence on the comparison with bp ranges between 70 and 90 percent).

Figure 4 shows (reading from left to right) the average feature set size, the average sum of the order of each feature, the average order for each feature, and the maximum order of features used on the given data set. In addition, the original number of features for the data sets is given in the last column. On average, GA_HOP generates feature sets which have two thirds as many features as the original data sets. The average order of the features generated by GA HOP is 2nd order.

Data set	Features	Sum_order	Avg_order	Max_order	Orig_feat
echo	3.20	3.80	1.19	3.00	10.00
bcw	7.20	9.10	1.26	3.00	10.00
bc	2.70	6.10	2.26	5.00	9.00
bupa	8.50	19.70	2.32	11.00	6.00
credit	4.90	10.10	2.06	12.00	15.00
pima	8.70	25.40	2.92	12.00	8.00
heart	10.30	15.70	1.52	5.00	13.00
average	**6.50**	**12.84**	**1.93**	**7.29**	**10.14**

Figure 4. Average feature set size.

5. Conclusion

The GA search technique was able to outperform a multilayer backpropagation network by an average increase of approximately 1 percent in classification accuracy on the data sets tested in tbis paper. Interestingly, it is able to do this while also decreasing tbe

average size (in numbers of features) of the feature set. Occasionally it was observed that a single very high order feature would be generated by the GA, but on average the GA generated features of relatively low order (2nd order) which produced good generalization results.

These results follow those found in [1] which showed that using features which are greater than 2nd order is unlikely to increase classification accuracy, and can 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 also show that the fitness function by itself was probably too conservative when it included the number of features amd feature order as a parameter to reduce the size of the feature set to an optimal level. This is an area where further research could be done in order to improve the results obtained in this paper. 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, wbile 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 fttness 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 individoals. 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 ftmess function,
- reftnement of the objective function,
- allowing the GA calculate feature weights,
- and further reftnement of the GA.

6. References

[1] 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.

[2] Andersen, Timothy and Tony Martinez. Genetic Algorithms and High Order Perceptrons. *The Second International Worshop on Neural Networks and Neurocontrol*, 1997.

[3] Baim, Paul. A Method for Attribute Selection in Inductive Leaming Systems. *IEEE Transactians on Pattem Analysis and Machine Intelligence*, vol 10, no 6, pp 888-896, November 1988.

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

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

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

[7] Elio, Renee and Larry Watanabe. An Incremental Deducttve Strategy for Controlling Constructive Induction in Leaming from Examples. *Machine Learning*, vol 7, pp 7-44, 199].

[8] Giles, C. Lee and Tom Maxwell. Learning, invariance, and generalization in high-order neural networks. *Applied Optics*, vol 26, no 23, pp 4972-4978, 1987.

[9] 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.

[10] Hoffgen, Klaus, Hans Simon, and Kevin Van Horn. Robust Trainability of Single Neurons.

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

[12] Kowatczyk, Adam and Herman Ferra. Developing Higher Order Networks with Empirically Selected Units. *IEEE Transactions on Neural Networks,* vol 5, no 5, pp 698-711, September 1994.

[13] 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.

[14] Morgera, Salvatore. Toward a Fundamental Theory of Optimal Feature Selection: Part 11 - Implementation and Computational Complexity. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol 9, no 1, pp 29-38, 1987.

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

[16] Perantonis, Stavros and Paulo Lisboa. Translation, rotation, and scale invariamt pattem recogmtion by high-order neural networks amd moment classifiers. *IEEE Transactions on Neural Networks*, vol 3, no 2, pp 241-251, March 1992.

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

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

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

[20] Siedlecki, Wojciech and Jack Sklansky. On Automatic Feature Selection. *International Journal of Pattern Recognition and Artificial Intelligence*, vol 2, no 2, pp 197-200 1988.

[21] Thompson, Mary. Selection of Variables in Multiple Regression: Part 1. A Review and Evaluation. *International Statistical Review*, 46, pp 1-19, 1978.

[22] Vapaik, V. *Estimation of Dependences Based on Empirical Data*. New York: Springer-Verlag 1982.

[23] Vaaria, Jari and Setsuo Ohsuga. Adaptive Neural Architectures through Growth Control. *Intelligent Engineering Systems through Artificial Neural Nerworks*, vol 1, pp 11-16, 1991.

[24] Volper, Dennis amd Steven Hampson. Quadratic Function Nodes: Use, Structure and Training. *Neural Networks*, vol 3, pp 93-107, 1990.

[25] 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.

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

[27] Zamdt, Frederick. Masters Thesis, Computer Science Department, Brigham Young Umversity.