In *WCNN'93 World Congress on Neural Networks*, vol. III, pp. 376-380, 1993.

GS: A Network that Learns Important Features

Cory Barker and Tony Martinez
Computer Science Department, Brigham Young University, Provo, Utah 84602

Abstract

GS is a network for supervised inductive learning from examples that uses ideas from neural networks and symbolic inductive learning to gain benefits of both methods. The network is built of many simple nodes that learn important features in the input space and then monitor the ability of the features to predict output values. The network avoids the exponential nature of the number of features by using information gained by general features to guide the creation of more specific features. Empirical evaluation of the model on real world data has shown that the network provides good generalization performance. Convergence is accomplished within a small number of training passes. The network provides these benefits while automatically allocating and deleting nodes and without requiring user adjustment of any parameters. The network learns incrementally and operates in a parallel fashion.

1. Introduction

This paper describes a network architecture for *supervised learning* that combines techniques used in neural networks (Ackley, Hinton, & Sejnowski, 1985; Rosenblatt, 1958; Rumelhart, Hinton, & Williams, 1986) with symbolic machine learning (Michalski, 1983; Mitchell, 1982, Quinlan, 1986) to gain advantages of both approaches. In supervised learning the network is given a *training set* containing *examples*. Each example gives an *input* pattern along with the corresponding *output* that the network should produce when presented with the input. The job of the network is not only to converge to a *representation* that contains the information given by the training set, but to *generalize* that information so that the network will respond well to inputs that it has not been trained on.

One approach to generalization is to look for important *features* in the input space. A feature is some subset of network inputs along with their associated values. A feature is *matched* when the values on the network inputs that are part of the feature are equal to the values for those inputs as given in the feature. Inputs that are not part of the feature can be any value. A feature that predicts an output with high probability is an important feature. The number of inputs contained in a feature is the *order* of the feature and determines the *generality* of the feature. A feature with few inputs is a *general* feature, while a feature with many inputs is a *specific* feature. It is impractical to monitor all possible input features because the number of features is exponential in the number of inputs.

This paper proposes GS (General to Specific), a network that monitors general input features and then specializes those features by combining the best general features. This section presents an overview of the model while later sections provide detail about the system. The network is made up of many simple nodes. Each node contains the input feature that it monitors. The node gathers statistics during training giving the discrete conditional probability of each possible output value given the input feature.

We use a simplified version of the problem of deciding when to use manual or automatic flight controls when landing an aircraft to illustrate learning in one version of the model. Two examples will be presented to the network shown below which has already processed several examples.

Node 1	Node 2	Node 3
{(Visibility, Clear)}	{(Wind Direction, Head)}	{(Wind Speed, Strong)}
Manual .50 Automatic .50	Manual .50 Automatic .50	Manual .33 Automatic .67

The problem has three inputs (Visibility, Wind Direction, and Wind Speed) and one output that can be set to either Manual or Automatic. Node 1 monitors the input Visibility and matches when the value is Clear. Node 1 predicts both outputs with equal probability. Node 3 matches the feature {(Wind Speed, Strong)} which has a higher probability for output Automatic.

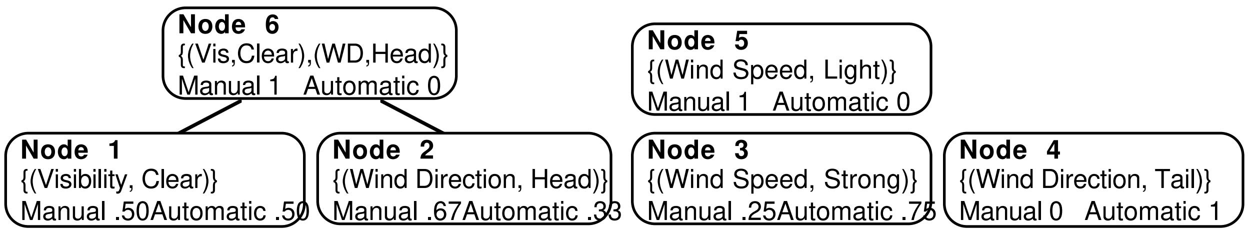
Example 1 (Visibility, Clear) (Wind Direction, Tail) (Wind Speed, Strong) → Automatic

When a new example is presented all possible first-order features in the example are always created. The example above contains the feature {(Wind Direction, Tail)} that does not have an existing node so a new node is created. The network nodes compete according to probability to assert their most probable output value. Node 3 wins the competition and outputs Automatic. The output agrees with the example so no additional nodes are created. The nodes that match the input example update their probabilities giving an updated network.

Node 1	Node 2	Node 3	Node 4
{(Visibility, Clear)}	{(Wind Direction, Head)}	{(Wind Speed, Strong)}	{(Wind Direction, Tail)}
Manual .33 Automatic .67	Manual .50 Automatic .50	Manual .25 Automatic .75	Manual 0 Automatic 1

Example 2 (Visibility, Clear) (Wind Direction, Head) (Wind Speed, Light) → Manual

The next example is presented to the network and node 1 wins the competition and outputs Automatic. The output is incorrect, so not only is the new first-order node with feature {(Wind Speed, Light)} created, but a new second-order node is created. The two nodes with highest probability for the output value Manual (nodes 1 and 2) combine to create a new node with the feature {(Visibility, Clear), (Wind Direction, Head)}. Nodes matching the example update their probabilities giving the new network shown below.



Some nodes created by the network are later found to be unimportant and should be deleted. Nodes determine if they should delete themselves by monitoring their own usage. All nodes keep a flag that is set whenever the node is used. After each pass over the training set, each node checks and resets its use flag. If the flag was not set during the training iteration, the node deletes itself.

Note that first-order nodes have limited power in finding a good representation of the training set, but the addition of nodes that recognize higher-order features allows the network to learn more complex training sets with better accuracy. This is analogous to typical neural networks where a single-layer network is limited to learning features that are linear combinations of the inputs, but multi-layer networks can learn complex higher-order features.

The GS network has a number of potential advantages. Typical neural network training procedures require a large number of iterations over the training set while the GS model converges in a small number of passes. Most networks require the user to specify the number of hidden nodes as well as control many learning parameters, but the GS model automatically creates and deletes nodes and has no parameters to set. Empirical tests of GS have shown promising generalization performance.

2. Definitions

A GS network N has *inputs* $i_1, ..., i_n$, with domains $X_1, ..., X_n$, respectively and *outputs* $o_1, ..., o_m$, with domains $Y_1, ..., Y_m$, respectively. The *input space* of N is $I = X_1 \times ... \times X_n$ (the cross product of the input domains) and the *output space* is $O = Y_1 \times ... \times Y_m$. The network N implements a function f_N from I into O. A training *example* is a pair (i,o) from $I \times O$. A training set is a subset of $I \times O$. The GS model restricts the input and output domains, $X_1, ..., X_n$ and $Y_1, ..., Y_m$, to be finite sets of discrete values.

The aircraft landing problem introduced above has three inputs, Visibility, Wind Direction, and Wind Speed, with domains {Clear, Cloudy}, {Head, Tail}, and {Light, Strong}, respectively. There is a single output, Flight Control, with domain {Automatic, Manual}. The input space has eight members {(Clear, Head, Light), ..., (Cloudy, Tail, Strong)}.

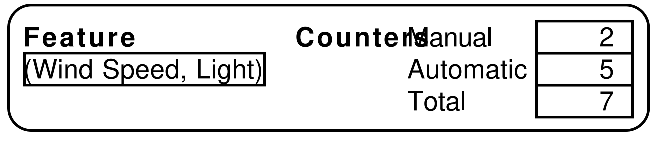
3. Node Architecture

This section describes the structure of individual nodes. Each node in a GS network contains:

- A *feature* which is a set of pairs (i_j, x), where i_j is an input and x is a member of the input domain X_j.
- A vector c_j of *counters* for each output o_j, with a counter for each value of output o_j.
- A total counter T.

The GS model has several variations. In some variations the nodes contain additional items.

A typical node is shown below. The node monitors the feature {(Wind Speed, Light)}. There are two counters for the single output and the total counter.

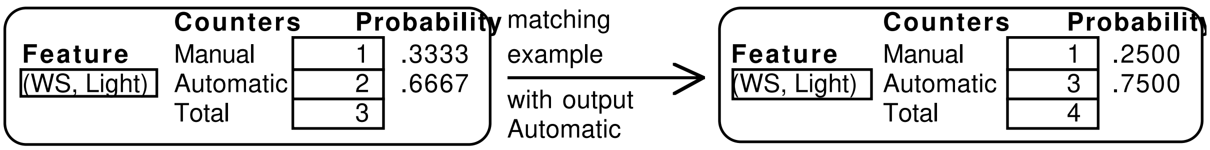


Each pair (i_j, x) in a feature specifies that an input must equal a specific value. A node *matches* the presented input values if for each pair (i_j, x) in its feature, the value of input i_j is equal to x. The *generality* of a node is the number of inputs *not* contained in the feature divided by the total number of inputs n. A node whose feature contains all n inputs is maximally specific; it matches only a single input example. A node whose feature contains no inputs is maximally general; it matches all possible input examples. If an input i_j is added to a feature the feature covers $1/k$ times the input space that it would cover without the input, where k is the size of the input domain X_j or in other words the number of values possible for the input.

The counters in each node allow the node to calculate the discrete probability of each output value based on the input examples that the node has seen. During training with example (*i, o*), all nodes that match *i* do the following:

- Increment the total counter *T*.
- For each output o_j, increment the counter in c_j associated with the value for the output o_j given in the training example.

The *probability* of value *y* for output o_j is $P(o_j = y \mid \text{feature match}) = c_j^y / T$ (where c_j^y is the counter in c_j for the value *y*). Counter update in a typical node is shown below.



4. Execution

When given input values, a network of nodes executes and produces an output as follows. Each node that matches the input values calculates its probability for each output value. The purpose of each node is to approximate the conditional probability of an output value given that the input values match a particular feature. The node with highest probability wins the competition and the best output value of that node is asserted as the output value of the network. This paper intentionally avoids network interconnect issues. However, note that any interconnect that supports broadcast and gather operations (i.e. tree or hypercube) is sufficient to do network execution and learning.

Network Execution

The input values presented to the network inputs are broadcast to all nodes in the network

Each node compares the input values to the feature stored at the node; nodes that match

a) Calculate the probability of each output value

b) Send the maximum probability output value along with its probability out to the network

The network gathers the outputs from each node.

The output value with maximum probability is the network output.

---

5. Learning Features

This section presents how a GS network learns the features that are important in the problem it is solving. The network should approximate the minimum possible set of features if it is to provide the best generalization.

The GS approach to creating features is to start with general nodes and converge toward specific nodes. Nodes are first created for first-order features. Additional inputs are added to the feature to create more specific nodes. First-order nodes are always created. When an example is presented, each individual input of the example is used to create a new node if an equivalent node does not already exist.

If the network outputs differ from the training example, then a new higher-order node is added to the network. The problem of deciding what new higher-order node to create is solved in three different ways giving three unique learning models; *combination*, *specialization*, and *specialization with example memory*. In all three methods existing nodes that match the input values create a new higher-order feature based on the node's current feature. Nodes assign a goodness metric to their new candidate feature based on the generality of the feature as well as the conditional probability with which the feature predicts the output value of the current training example. The nodes compete according to the goodness of their candidate features. The feature that wins the competition may be equal to the feature in an existing node. In this case the competition is repeated without the winning feature to get the next best feature. The best candidate feature that is unique (not equal to a feature in an existing node) is used to create a new node.

Network Learning

The training example (*i, o*) is broadcast to all nodes in the network and the nodes execute giving an output *r*.

For each input value x_j in *i* create a new feature $F = \{(i_j, x_j)\}$. If a node with feature *F* does not already exist, create a new node with feature *F*.

If the network output is not correct ($r \neq o$) then create a higher-order node by either Combination, Specialization, or Specialization with Example Memory.

Each node that matches *i* increments its counters as described above.

5.1. Combination

One method for creating higher-order features is for existing nodes that match the training example and have high generality and high probability of the correct output value to combine. When two nodes combine to create a new node, the feature for the new node is the union of the features of the other nodes. Note that if the two original features contain the same input, that input will have the same value in both features because both features must match the training example. An example of feature combination was shown in the introduction where two nodes with features {(Visibility, Clear)} and {(Wind Direction, Head)} combine to create a node with feature {(Visibility, Clear), (Wind Direction, Head)}.

The two most general and probable nodes combine their features and check to see if the new feature already exists in a node. If the feature already exists, the best node marks itself as not combinable and the next best two nodes attempt to combine. When a unique feature is found a new node is created.

Combination
  All nodes that match the input values mark themselves as available for combination.
  The available nodes compete according to generality and probability. Let *A* be the winning node.
  Node *A* marks itself as unavailable.
  Repeat until either a node is created or all nodes are marked unavailable
    The available nodes compete according to generality and probability. Let *B* be the winning node.
    Node *B* marks itself as unavailable.
    Create a new feature *F* that is the union of the features from nodes *A* and *B*.
    If a node with feature *F* does not already exist, create a new node with feature *F*.
    Set *A* to *B*.
  End.

Note that in the worst case Combination requires $O(n \log n)$ time, where n is the number of nodes. The repeat loop can repeat at most n-1 times. The worst operations inside the loop are competition between nodes and searching for a node with an equal feature. Both operations can be done in log n time on a tree architecture.

5.2. Specialization

Rather than combining nodes to create new nodes, individual existing nodes can specialize themselves by adding an input to their feature. All matching nodes (in parallel) find the best specialization of their feature that is unique. The best specialization adds the highest probability first-order feature to the current feature. The probability information for first-order features can be either stored within each node by adding more counters or it can be obtained from the first-order nodes. The nodes compete according to generality and probability. The winning node creates a new node using its specialized feature.

Specialization
  All nodes that match the input values do the following
    a) Order the inputs that are not contained in the current feature according to probability.
    b) Select the highest probability input that specializes the current feature into a new unique feature.
  The nodes compete according to generality and probability.
  The winning node creates a new node using its new feature.

5.3. Specialization with Example Memory

Combination and specialization both depend on the assumption that higher-order features contain lower-order features that have high predictability. Specialization with example memory allows each node to store a fixed number of examples and then use the stored examples to guide the creation of higher-order features. For every input i_j *not* contained in the node's current feature, a new feature is created by adding the pair (i_j, x_j) to the current feature, where x_j is the value of input i_j from the training example. The probability of the output value given in the training example is calculated for each new feature by counting the output values of the examples stored in the node that match the new feature. The new feature with the highest probability is used in competition with other nodes in the network and the winning node creates a new node using the best feature.

Specialization with Example Memory
  All nodes that match the input values do the following
    a) For each input i_j not in the current feature create a new feature by adding the pair (i_j, x_j) to the current feature, where x_j is the value of input i_j in the training example.
    b) For each new feature, use the examples stored in the node to calculate the probability of the output value given by the training example and select the new feature with highest probability.
  The nodes compete according to generality and probability.
  The winning node creates a new node using its new feature.

---

6. Deletion

Because nodes are created incrementally, many times nodes will be created that are not essential for representing the training set. If non-essential nodes can be identified, they can be deleted from the system. Deletion of unnecessary nodes can improve generalization. Measurement of node usage and comparison of node probabilities can both be used to identify non-essential nodes. Space does not permit a detailed explanation of deletion methods.

7. Results

GS was tested using four real world training sets acquired from the UC Irvine repository of machine learning databases (Murphy & Aha, 1992); iris plant classification (Fisher, 1936), 1984 Congressional voting records (Schlimmer 1987), hepatitis data, and mushroom classification (Schlimmer 1987). Each learning model (combination, specialization, and specialization with example memory) was tested on each problem. The results given here are averages over ten runs. For each run half of the examples were randomly selected for the training set. The remaining examples were used as a test set to test generalization performance. The results shown in the table below are the percent correct on the training set followed by the percent correct on the test set and finally the average number of training set iterations.

	Combination	Specialization	Specialization with Example Memory
Iris	99.20 / 90.93 / 2.2	99.87 / 91.07 / 2.9	99.87 / 92.53 / 2.2
Voting	97.10 / 93.64 / 10.1	98.89 / 94.42 / 14.5	100.0 / 94.42 / 2.6
Hepatitis	95.58 / 82.56 / 7.6	99.35 / 83.33 / 10.5	99.74 / 82.18 / 2.3
Mushroom	99.98 / 99.98 / 3.7	99.94 / 99.96 / 4.5	100.0 / 100.0 / 1.4

Based on these results, GS networks compare favorably to other supervised learning approaches. Specialization typically is more correct than combination but requires more iterations. Using example memory usually improves correctness while at the same time reducing the number of passes over the training set.

8. Conclusions

The GS network has been presented as a system capable of supervised learning with benefits of both neural networks and symbolic inductive learning. Evaluation of the model on four problems has shown that the network provides good generalization performance. Convergence is accomplished within a small number of training passes. The network provides these benefits while automatically allocating and deleting nodes and without requiring user adjustment of any parameters. The network learns incrementally and operates in a parallel fashion.

Future research will focus on finding better methods for creating higher-order nodes, the development of network architectures to support learning with better parallelism, testing on additional data sets, and extensions to real-valued inputs.

References

Ackley, D. H., Hinton, D. E., & Sejnowski, T. J. (1985). A Learning Algorithm for Boltzmann Machines. *Cognitive Science, 9*, 147-169.

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annual Eugenics, 7,* Part II, 179-188.

Michalski, R. S. (1983). A Theory and Methodology of Inductive Learning. *Artificial Intelligence, 20,* 111-161.

Mitchell, T. M. (1982). Generalization as Search. *Artificial Intelligence, 18,* 203-226

Murphy, P. M. & Aha, D. W. (1992). *UCI Repository of machine learning databases*. Irvine, CA: University of California, Department of Information and Computer Science.

Quinlan, J. R. (1986). Induction of Decision Trees. *Machine Learning, 1,* 81-106.

Rosenblatt, F. (1958). The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review, 65,* 386-408.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning Internal Representations by Error Propagation. In D. E. Rumelhart and J. L. McClelland (Eds.) *Parallel Distributed Processing* (Vol. 1). Cambridge, MA: MIT Press.

Schlimmer, J. C. (1987). Concept acquisition through representational adjustment. Doctoral dissertation (Technical Report 87-19), Department of Information and Computer Science, University of California, Irvine, CA.