In *AI'93 Australian Joint Conference on Artificial Intelligence*, pp. 142-149, 1994.

GENERALIZATION BY CONTROLLED INTERSECTION OF EXAMPLES†

Cory Barker and Tony Martinez
Computer Science Department, Brigham Young University, Provo, Utah 84602
cory@axon.cs.byu.edu, martinez@cs.byu.edu

Abstract

SG (Specific to General) 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 creating specific features for each example and then expanding those features; making them more general. Expansion of a feature terminates when it encounters another feature with contradicting outputs. 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[1,7,8] with symbolic machine learning[3,4,6] 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 task 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.

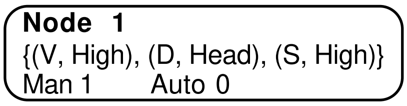
This paper proposes SG (Specific to General), a network that creates specific input features and then generalizes those features. One way SG generalizes is by combining similar specific features. If two features are similar, they are close to each other in the input space. Combining the two features by dropping inputs that are not common between the features creates a new feature that encompasses both of the original features. The new feature is general; it matches points in the input space that have not been defined by any example.

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. During training, the node gathers statistics giving the discrete conditional probability of each possible output value given the input feature.

† This Research Supported in Part by the Air Force Office of Scientific Research

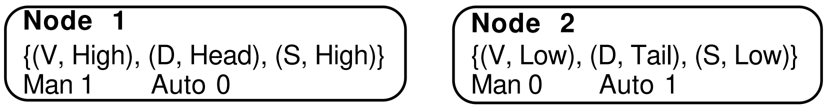
We use a simplified version of the problem of deciding when to use manual or automatic flight controls when landing an aircraft to illustrate learning in one version of the model. The problem has three inputs (Visibility, Wind Direction, and Wind Speed, abbreviated V, D, and S, respectively) and one output that can be set to either Manual or Automatic (abbreviated Man and Auto, respectively). Several examples are presented to an empty network.

Example 1 (V, High) (D, Head) (S, High) → Man



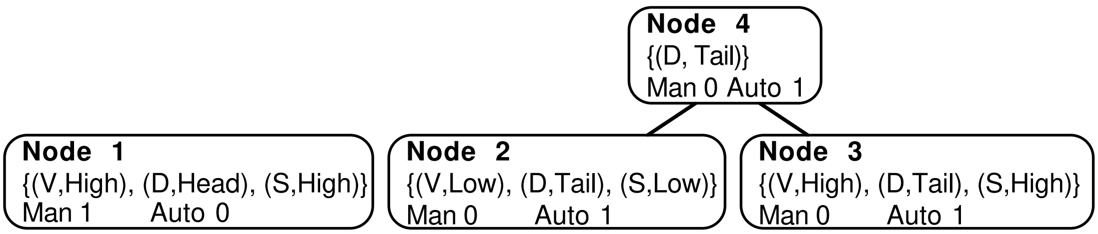
When a new example is presented a node is always created with a specific feature exactly equal to the example unless an equivalent node already exists. The first example causes the creation of Node 1. Node 1 sets its probabilities according to the input example (i.e., Man has probability 1 since the only example seen matching Node 1's feature has output manual). The network has no other nodes to combine with Node 1 so no additional nodes are created.

Example 2 (V, Low) (D, Tail) (S, Low) → Auto



Example 2 causes the creation of Node 2 with the specific feature equal to the example. Node 2 sets its probabilities according to the input example. Node 1 does not match the input example so its probabilities remain unchanged. Node 1 has a different highest probability output than Node 2 so the two nodes cannot combine and no additional nodes are created.

Example 3 (V, High) (D, Tail) (S, High) → Auto



Example 3 causes the creation of the specific Node 3. Node 3 sets its probabilities according to the input example. Nodes 2 and 3 both have the same highest probability output (Automatic) so they create Node 4 with a feature that has all the inputs that are common to both Nodes 2 and 3. Node 4 has a general feature; it matches input examples that have not been presented yet (e.g., (V, Low) (D, Tail) (S, High) → Auto). Node 4 sets its probabilities by combining the probabilities from Nodes 2 and 3. Note that specific nodes can learn the training set perfectly but do not generalize. Generalization has the effect of expanding the area of the input space covered by a node until eventually the entire input space is covered.

The SG network has a number of potential advantages. Typical neural network training procedures require a large number of iterations over the training set while the SG 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 SG model automatically creates and deletes nodes and has no parameters to set. Empirical tests of SG show promising generalization performance.

2. Definitions

An SG 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 SG 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 {Low, High}, {Head, Tail}, and {Low, High}, respectively. There is a single output, Flight Control, with domain {Automatic, Manual}. The input space has eight members {(Low, Head, Low), ..., (High, Tail, High)}.

3. Node Architecture

This section describes the structure of individual nodes. Each node in an SG 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 SG model has several variations. In some variations the nodes contain additional information.

A typical node is shown below. The node monitors the feature {(Wind Speed, Low)}. There is a counter for each of the possible output values and a total counter.

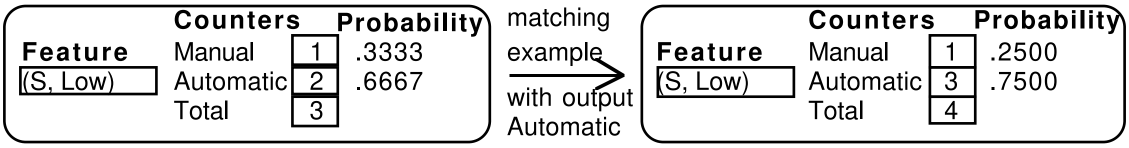
Feature	**Counters**		
(Wind Speed, Low)		Manual	2
		Automatic	5
		Total	7

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 *specificity* of a node is the number of inputs 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 removed from a feature the feature covers k times the input space that it would cover with 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 $\mathrm{P}(o_j = y \mid \text{feature match}) = c_j^y / T$ (where c_j^y is the counter in c_j for the value y). Counter update in a typical node is shown below.



Two nodes or two examples are said to be *concordant* if they both have the same output values. They are *discordant* if they have different output values.

The *strength* of a node is the probability of its highest-probability output value.

Node *A* *covers* node *B* if *A* matches every example matched by *B*, nodes *A* and *B* are concordant, and node *A* is stronger than node *B*. It can be shown that *A* matches every example matched by *B* if and only if *A's* feature is a *subset* of *B's* feature.

Node *A* *fully contradicts* node *B* if *A* matches every example matched by *B*, nodes *A* and *B* are discordant, and node *A* is stronger than node *B*.

Node *A* *partially contradicts* node *B* if *A* matches at least one example matched by *B*, nodes *A* and *B* are discordant, and node *A* is stronger than node *B*.

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 (e.g., tree or hypercube) is sufficient to do network execution and learning.

Network Execution

1. The input values presented to the network inputs are broadcast to all nodes in the network
2. Each node compares the input values to the feature stored at the node.
3. Nodes that match
    4. Calculate the probability of each output value
    5. Send the maximum probability output value along with its probability out to the network
6. End.
7. The network gathers the outputs from each node.
8. The output value with maximum probability is the network output.

End.

5. Learning Features

This section presents how an SG 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 SG approach to creating features is to start with specific nodes and expand around those nodes creating general nodes. Nodes are first created with features that match the input example exactly. Inputs are removed from the feature to create more general nodes until the new nodes conflict with discordant nodes.

When a new example is presented during learning the network executes using only general nodes. Nodes that are completely specific do not participate since the network output would always be correct for the training examples after the first pass. The network creates a node that matches only the new example if such a node does not already exist. If the network outputs differ from the training example, then new general nodes are added to the network. The problem of deciding what general nodes to create is solved in two different ways giving two unique learning models; *combination* and *generalization*. Combination is described in the next section. Generalization is described in elsewhere[2]. In both methods, existing nodes create new candidate features based upon their current feature and the new example.

Network Learning

1. The training example (i, o) is broadcast to all nodes in the network and the general nodes execute giving an output r.
2. Create a new feature $F = \{(i_1, x_1), ..., (i_n, x_n)\}$, where $x_1, ..., x_n$ are the values given in i.
3. If a node with feature F does not already exist, create a new node with feature F.
4. Each node that matches i increments its counters as described above.
5. If the network output is not correct ($r \neq o$) then create general nodes by either Combination or Generalization.

End.

5.1. Combination

One method for creating general features is for existing nodes that are concordant with the training example to combine their current feature with the training example. When the two features combine to create a new node, the feature for the new node is the intersection of the features of the other nodes. An example of feature combination was shown in the introduction where two nodes with features {(V, Low), (D, Tail), (S, Low)} and {(V, High), (D, Tail), (S, High)} combine to create a node with feature {(D, Tail)}.

Once a node has created a new candidate feature by intersection with the training example, the feature must be broadcast to the network to determine if the feature is covered or contradicted by another node. If the new feature is covered or contradicted there is no need to create a node with the feature since the node would not affect the network output. If the new feature is not covered or contradicted then a new node is created with the feature.

Combination

1. All nodes concordant with the training example do the following in parallel
2. Create new candidate feature G by intersection of the node's current feature and the training example
3. If G is not covered and not contradicted then create a new node N with feature G.
4. Initialize counters in N to the sum of counters in fully specific nodes covered by N.

End.

Note that in the worst case Combination requires $O(n \log n)$ steps, where n is the number of nodes. At most n nodes will broadcast a new candidate feature to check for covered and contradicted conditions. Each broadcast is a $\log n$ operation on a tree architecture.

6. Deletion

Because nodes are created incrementally, many times nodes will be created that are not essential for representing the training set. Deletion of unnecessary nodes can improve generalization. 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.

7. Results

SG was tested using four real world training sets acquired from the UC Irvine repository of machine learning databases[5]; iris plant classification, 1984 Congressional voting records, hepatitis data, and mushroom classification. Both learning models (combination and generalization) were 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			Generalization		
Iris	98.13	90.67	2.3	100.0	94.13	2.8
Voting	99.95	94.24	5.3	99.86	91.61	5.9
Hepatitis	99.74	77.69	6.1	99.74	80.77	2.8
Mushroom	100.0	99.97	4.6	100.0	89.33	5.5

Based on these results, SG networks compare favorably to other supervised learning approaches. Combination performs better on voting and mushroom data while generalization performs better on iris and hepatitis data. Both models learn the training set well and require a small number of training iterations.

8. Conclusions

The SG 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 improved methods for creating general nodes, the development of network architectures to support learning with better parallelism, testing on additional data sets, and extensions to real-valued inputs.

References

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

[2] Barker, J. C. (1993). Generalization by Controlled Expansion of Examples. *In preparation.*

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

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

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

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

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

[8] 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.