In Proceedings of the *7th International Symposium on Artificial Intelligence*, pp. 142-149, 1994.

GENERALIZATION BY CONTROLLED EXPANSION OF EXAMPLES

Cory Barker

Computer Science Department
Brigham Young University
Provo, Utah 84602

Tony Martinez

Computer Science Department
Brigham Young University
Provo, Utah 84602

ABSTRACT

SG (Specific to General) is a learning system that derives general rules from specific examples. SG learns incrementally with good speed and generalization. The SG network is built of many simple nodes that adapt to the problem being learned. Learning is done without requiring user adjustment of sensitive parameters and noise is tolerated with graceful degradation in performance. Nodes learn important *features* in the input space and then monitor the ability of the features to predict output values. Learning is O(n log n) for each example, where n is the number of nodes in the network, and the number of inputs and output values are treated as constants. An enhanced network topology reduces time complexity to O(log n). Empirical results show that the model gives good generalization and that learning converges in a small number of training passes.

INTRODUCTION

This paper describes a system for inductive machine learning. The system described in this work uses symbolic representations and generalization rules often found in symbolic machine learning [3, 4, 6]. The system uses a parallel network of simple nodes and the system makes incremental changes to its state as is commonly found in neural networks [1, 7, 8]. The new system described in this work provides:

- Good generalization.
- Good learning speed.
- Incremental learning.
- Graceful degradation in the presence of noise.
- Ease of use.
- A massively parallel architectural model.

Inductive learning systems extract general rules from specific examples. The system is given a *training set* containing *examples*. Each example gives an *input* pattern along with the *output* that the network should produce for that input. The training set is a subset of the complete mapping from input to output. Therefore, the network should not only converge to a *representation* that contains the information given by the training set, but also *generalize* that information, so that the network will respond well to inputs that it has not been trained on. The system finds general rules that agree with the training set. The general rules tend to predict the correct output for inputs that the system has not been trained for.

SG networks learn by monitoring *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 with few inputs is a *general* feature; it matches many points in the input space. A feature with many inputs is a *specific* feature; it matches few points in the input space. It is impractical to monitor all possible input features because the number of features is exponential in the number of inputs.

The network is made up of many simple nodes. Each node monitors the *strength* of an input feature. The strength of a feature is determined by the examples that are matched by the feature. If most examples that match a feature have the same output value, then the feature is a strong feature. SG creates nodes with specific input features and then generalizes those features. Existing nodes generalize their features by dropping input/value pairs.

The nodes in the system operate in parallel. Training examples are broadcast to all nodes in the system. The nodes compete with one another to determine the output of the system. Existing nodes can spawn new nodes by combining the current function of the node with the information given in the training example.

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.

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. 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 application used in this paper to illustrate SG learning is a simplified version of aircraft flight control. The application has three inputs, *Wind Speed, Wind Direction,* and *Visibility,* with domains {*Low, Medium, High*}, {*Head, Tail*}, and {*Clear, Cloudy*}, respectively. The output domain is {*Manual, Automatic*}.

A training *example* is a pair (*i, o)* from $I \times O$. Each example gives the input/output association for one point in the input domain. A *training set* is a subset of $I \times O$.

A *feature* 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 feature *matches* a point in the input space if for each pair (i_j, x), the value of input i_j is equal to x. The set of points in the input space matched by a feature is called the *subdomain* of the feature.

The *generality* of a feature F is defined as $\frac{n-|F|}{n}$, where n is the total number of inputs to the system. Generality is the number of inputs *not* contained in the feature divided by the total number of inputs. The feature containing no inputs is maximally general; it matches all possible input points. A feature containing all n inputs is maximally specific; it matches only a single input point. Each input i_j *not* contained in a feature allows the feature to cover 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.

A feature B obtained by removing one or more input/value pairs from a feature A is a *generalization* of A. It can be shown that the subdomain of B is a superset of the subdomain of A.

Let A be a feature. Let v be a point in the output domain. Let t be the total number of examples matched by A. Let c_v be the number of examples that are matched by A and have output equal to v. The *strength* of output value v for feature A is $\frac{c_v}{t}$. The strength of a feature is a measure of how well the feature can predict an output value. If a feature A matches no examples ($t = 0$), then the strength of all outputs for the feature are defined as 0.

The *output* of a *feature* is its strongest output value. If the two strongest outputs of a feature have the same strength, the output of the feature is *Don't-Know*. The *strength* of a *feature* is the strength of its strongest output value.

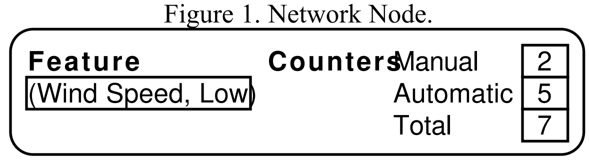
Let A and B be features. Let S_A and S_B be the strength of features A and B, respectively. Let V_A and V_B be the output of features A and B, respectively. Feature A *covers* feature B if A matches every example matched by B, $S_A \geq S_B$, and $V_A = V_B$. Feature A *contradicts* feature B if A matches every example matched by B, $S_A \geq S_B$, and $V_A \neq V_B$.

NODE ARCHITECTURE

This section describes the structure of the individual nodes in an SG network. Each node is responsible for and represents a single feature. Each node contains a set of counters C for measuring feature strength. The set of counters C contains a counter c_o for each point o in the output space. Each node also contains a total counter T.

A typical node is shown in Figure 1. The node monitors the feature {(*Wind Speed, Low*)}. There is a counter for each of the possible output values and a total counter.

Figure 1. Network Node.



The counters in each node allow the node to calculate the strength of each output point based on the input examples that the node has seen. During training with example (*i, o*), all nodes that match *i* update their counters as follows:

- Increment the total counter T.
- Increment the counter c_o, associated with the output point o given in the training example.

The *strength* of output point o is $\frac{c_o}{T}$.

The result of giving the example (*Low*, *Head*, *Clear*) → *Automatic* to the node above is shown in Figure 2. The figure below also shows the strength of each output.

Figure 2. Counter Update.

Feature	Counters		Strength
(Wind Speed, Low)	Manual	2	.25
	Automatic	6	.75
	Total	8	

EXECUTION

When given input values, a network of nodes executes and produces an output as follows. Each node that matches the input state calculates the strength 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 nodes compete according to strength and the strongest node wins the competition. The output value of the winning node is asserted as the output value of the network. Thus, the output of a network of nodes is the output of the highest strength node. If the two strongest nodes have the same strength and different outputs, the output of the network is *Don't-Know*.

The network execution algorithm is shown below. *Broadcast* means that information is given to all nodes. *Gather* means that information is taken from all nodes.

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 where the feature matches the input values:
    4. Calculate the strength of each output value.
    5. Send the strongest output value along with its strength out to the network.
6. End.
7. The network gathers the outputs from each node.
8. The strongest output value is the network output.

End.

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. The system first creates *example nodes* with features that match the training example exactly. Inputs are removed from the feature to create more general nodes until the new nodes conflict with other nodes that have different outputs.

The system executes the learning algorithm shown below for each training example. The system iterates over the training set until a full pass is completed without creating or deleting any nodes. When a new example is presented during learning, the network executes using only general nodes. Example nodes do not participate since the network output would always be correct for the training examples after the first pass. The network creates an example node that matches only the new example, if such a node does not already exist.

If the network output is not equal to the example output, then a general node is created using existing nodes. Otherwise, no additional nodes are added to the system. The problem of deciding what general node to create is solved in two different ways giving two unique learning models; *intersection* and *generalization*. In feature-intersection, nodes create new candidate features by computing the intersection of their current feature and the feature defined by the training example. Details are given in [2]. In feature-generalization, nodes create new candidate features by dropping single input/value pairs from their current feature. Details are given in the next section.

After the training example is learned, nodes in the network that are not needed delete themselves. 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.

Specific to General Learning

1. Broadcast the training example (i, o) to all nodes in the network.
2. Gather the strongest non-example node output giving the network output r.
3. Create a new feature $F = \{(i_1, x_1), ..., (i_n, x_n)\}$,
   where $x_1, ..., x_n$ are the values given in i.
4. If a node with feature F does not already exist, create a new node with feature F.
5. Each node that matches i increments its counters as described above.
6. If the network output is not correct ($r \neq o$) then create a general node by either *feature-intersection* or *feature-generalization*.
7. Delete nodes that are not needed.

End.

Feature Generalization

All nodes (in parallel) that match the training example and have the same output as the example create generalizations of their current feature by removing an input/value pair. For every input i_j contained in the node's current feature, a new feature is created by removing the pair (i_j, x_j) from the current feature.

The strength of each candidate feature is calculated by counting matching example nodes. The candidate feature F with highest strength for the target output that is not covered or contradicted is selected for competition with other nodes. The generalization process is optionally repeated using the feature F.

The nodes compete according to the strength and generality of their candidate features and the winning node is selected to create a new node using its best feature. The counters in the new node are initialized by counting the example nodes that match the new feature.

Feature Generalization

1. For all nodes that match the example and have the same output as the example.
    2. Set F to the existing feature of the node.
    3. For each input i_j in F create a new feature F_j by removing the pair (i_j, x_j) from F.
    4. For each new feature F_j, calculate the strength S_j of F_j using Example Nodes.
    5. If at least one new feature exists that is not covered or contradicted
        6. Set F to the new feature with highest strength.
        7. Optionally repeat lines 3 to 8 using the new feature F.
    8. End.
9. End.
10. Gather the strongest and most general new feature F from the nodes using the metric $Gc + S(1-c)$, where G is the generality of F, S is the strength of F, and c is a constant between 0 and 1.
11. Create a new node N with feature F.
12. Initialize counters in N by counting example nodes that match N.

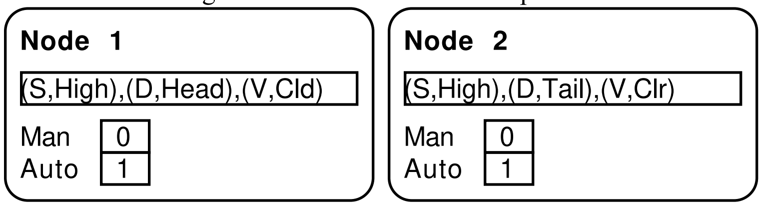
End.

Note that in the worst case feature-generalization 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.

Training Example

We now give an example of SG learning using feature-generalization. For simplicity, repetition is not used in this example, although repetition is typically used with feature-generalization. The current state of the system (having already processed some examples) is shown in Figure 3.

Figure 3. Network before Example 1.



Example 1: (*High, Head, Clear*) → *Manual*

Example 1 is given to the network. The output of the network is *Don't-Know* since no nodes match the example. Node 3 is created as an example node with a feature that matches only the single input point given in the example. The output of the network is different from the output of the example, so Feature-Generalization is executed.

Both Nodes 1 and 2 have outputs that are different from the training example. Generalization is only done with nodes that match the training example and have the same output as the example. Node 3 is the only node that meets this criteria.

Node 3 calculates the strength of the three features resulting from removing each input/value pair from its current feature. The feature {(*Wind Speed, High*), (*Wind Direction, Head*)} matches Example Nodes 1 and 3 so the strength of the feature for *Manual* is 1/2. The strength of feature {(*Wind Speed, High*), (*Visibility, Clear*)} is also 1/2 since it matches Nodes 2 and 3. The strength of feature {(*Wind Direction, Head*), (*Visibility, Clear*)} is 1/1 because it only matches Node 3.

The feature {(*Wind Direction, Head*), (*Visibility, Clear*)} is the strongest feature. The candidate feature is not equal to the feature of any node, nor is it covered or contradicted by any node. Therefore, Node 4 is created using the new feature. The counters in Node 4 are initialized using the only example node that is matched by Node 4, which is Node 3. The resulting network is shown in Figure 4. Note that connections are shown between nodes that have a specific to general relationship.

Example 2: (*Low, Head, Clear*) → *Manual*

Example 2 is given to the network. The output of the network is *Manual* since Node 4 is the only node that matches the example. The strength of output

Figure 4. Network after Example 1.

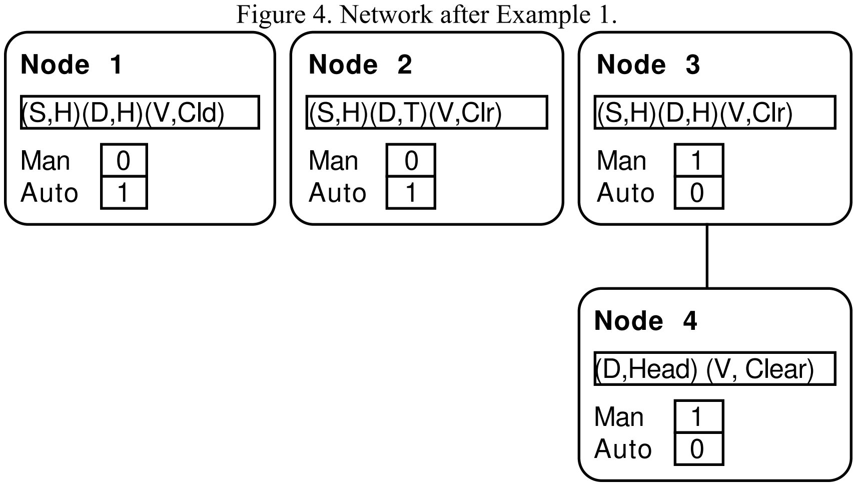
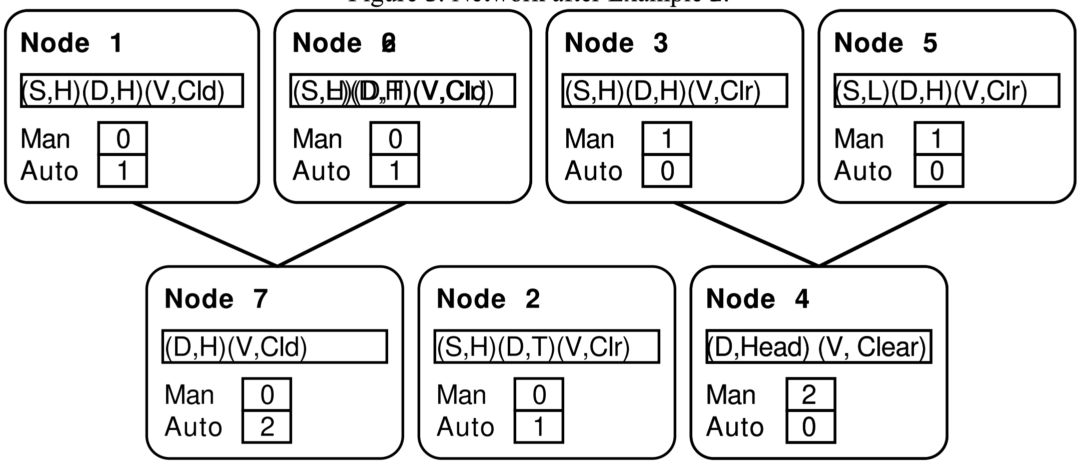


Figure 6. Network after Example 3.



Manual for Node 4 is 1/1 = 1 while the strength of output *Automatic* is 0/1 = 0.

A new example node (Node 5) is created. Node 4 and Node 5 update their counters since they match the example. Since the output of the network is the same as the example, feature-generalization is not done. The resulting network is shown in Figure 5.

Example 3: (*Low, Head, Cloudy*) → *Automatic*

Example 3 is given to the network. The output of the network is *Don't-Know* since no nodes match the example. An example node is created (Node 6). The output of the network is different from the output of the example, so feature-generalization is executed.

Node 6 is the only node that matches the example and has the same output as the example. Node 6 creates three new candidate features with strengths for output *Automatic* as shown in Table 1.

None of the features is equal to any existing feature. Two of the features have equal highest strength, so one of the two is arbitrarily chosen. Suppose that the last feature {(*Wind Direction, Head*), (*Visibility, Cloudy*)} is chosen since it has higher total count.

The new feature is not covered or contradicted by any other node. Node 6 has no competition for creation of a new node. Node 7 is created with counters initialized from Nodes 1 and 6. The final network is shown in Figure 6.

RESULTS

SG was tested using 10 data sets obtained from UC Irvine [5]; breast cancer diagnosis, chess end-game, hepatitis diagnosis, iris plant classification, LED digit recognition, mushroom classification, soybean disease diagnosis, tic-tac-toe end-game, 1984 House voting records, and animal classification. Both learning models (intersection and generalization) were tested on each problem. The results given here are averages over ten runs. One half of the examples in a data set were randomly selected as the training set for each run. The remaining examples were used as the test set. The results shown in the table below are the accuracy on the training set followed by the accuracy on the test set and finally the average number of training set iterations.

Table 1. Strengths for output *Automatic.*

{(*Wind Speed, Low*), (*Wind Direction, Head*)}	1/2 = .50
{(*Wind Speed, Low*), (*Visibility, Cloudy*)}	1/1 = 1.0
{(*Wind Direction, Head*), (*Visibility, Cloudy*)}	2/2 = 1.0

Both SG models have similar good results for Breast Cancer, Chess, LED, and Voting. Feature-generalization performs better than intersection on Iris and Soybean. Feature-intersection performs better than generalization on Mushroom and Tic-Tac-Toe. Both algorithms have low test accuracy on the Zoo data. The SG results compare favorably with previous results on these data sets with the exception of the Zoo data.

CONCLUSION

SG has been presented as a system for learning general rules from specific examples. Results on several data sets suggest that SG gives 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 sensitive 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 new applications, 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. & Martinez, T. R. (1993). Generalization by Controlled Intersection of Examples. *Proceedings of the Sixth Australian Joint Conference on Artificial Intelligence*.

[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.

Table 2. Empirical Results.

Data Set	Feature Generalization			Feature Intersection		
	Train Accuracy	Test Accuracy	Pass Count	Train Accuracy	Test Accuracy	Pass Count
Breast Cancer	99.77%	92.83%	3.3	99.86%	94.71%	2.9
Chess	100.00%	96.65%	3.4	99.98%	97.94%	3.4
Hepatitis	89.87%	81.15%	2.7	98.44%	79.87%	5.1
Iris	100.00%	94.53%	3.1	98.67%	90.93%	2.5
LED	74.36%	71.98%	3.7	65.08%	71.58%	3.3
Mushroom	94.10%	93.86%	4.8	100.00%	99.99%	2.2
Soybean	100.00%	97.92%	2.4	100.00%	79.58%	2.1
Tic Tac Toe	100.00%	81.02%	3.6	100.00%	97.10%	2.9
Voting	96.96%	92.12%	3.8	99.22%	95.16%	5.2
Zoo	100.00%	83.78%	2.8	98.22%	81.33%	2.1