In Proceedings of the *International Symposium on Circuits and Systems*, pp. 706-709, 1990.

CONSISTENCY AND GENERALIZATION IN INCREMENTALLY TRAINED CONNECTIONIST NETWORKS

Tony Martinez
Computer Science Dept., Brigham Young University
Provo, Utah 84602

Abstract

This paper discusses aspects of consistency and generalization in connectionist networks which learn through incremental training by examples or rules. Differences between training set learning and incremental rule or example learning are presented. Generalization, the ability to output reasonable mappings when presented with novel input patterns, is discussed in light of the above learning methods. In particular, the contrast between *hamming distance* generalization and generalizing by high order combinations of *critical variables* is overviewed. Examples of detailed rules for an incremental learning model are presented for both consistency and generalization constraints.

Introduction

A basic component of neural network mechanisms is the ability to adaptively *learn* mappings [2,9]. Learning takes place as information is presented to the network. The system must learn the information such that it can *generalize*. Generalization is the ability for a system, when presented with input not encountered during learning, to still produce an output with good probability of being correct. The class of applications for which neural networks have most promise, are exactly those applications for which generalization is possible [8].

There is no constraint on how information is presented to a connectionist system during learning. Two possible mechanisms are *examples* or *rules* The distinction between rules and examples can be quite fine. Assume a conjunction of boolean inputs with subsequent boolean outputs:

A B' C D => X' Y

where W' is the negation of W. Is this an example or a rule? In fact, it could be considered either. One potential differentiation between rules and examples is that rules may contain a smaller subset of the total possible inputs than an example. Assume an input space of the three boolean variables {A, B, C} and a single output Z. Assume the following examples:

A B C => Z
A B' C => Z
A B C' => Z
A B' C' => Z

Note the obvious correlation between A and Z. In each case Z is high if A is high regardless of the setting of the other variables. A rule representing this knowledge could be A => Z. In this case the variables B and C are considered as *don't care* variables, whereas A could be considered as a *critical* variable. A rule is typically more general than an example because it contains less variables. One mechanism of generalization in learning systems is to manipulate examples into more general rules.

Two basic mechanism for doing generalization are *hamming distance* and *critical variables*. In a hamming distance mechanism, the system seeks to match the input to learned prototypes, matching with the prototype with which it has the least number of total mismatches. This prototype then drives the output. With critical variable generalization, the combination of a few variables drive the output while others are considered as don't cares.

For example, assume the following examples, augmented from the example above, are given to a learning system.

A B C => Z
A B' C => Z
A B C' => Z
A B' C' => Z
A' B' C' => Z'

Let A' B C be the input to the system after learning of the above examples. Since there was no example given of A' B C the system must generalize or output a don't know. If a critical variable scheme is used, the system could use the correlation of A => Z and A' => Z' as a rule with A as a critical variable. Thus, it would output Z'. However, if a hamming distance scheme is used, then A' B C is different by only one variable from A B C => Z and by at least two variables from all other examples. Thus, the output would be Z.

Note that neither of the two options can be said to be correct since we can only guess at the output of an input for which total information has not been given. Which method is most promising will depend on specific applications and research. Most current neural network schemes use hamming distance generalization. There is evidence that natural nervous systems have the ability to extract critical input from a large barrage of total inputs and act, while ignoring currently unimportant inputs.

When differentiating between examples and rules, another potentially important feature is whether order of presentation is important. In the *training set* scheme, used typically with current example driven neural networks, all examples are equally important. The system seeks to average out the information of the many equivalent examples to derive a classification mapping. Rules may also be input in incremental fashion, where the order of input is important. Consider the natural training scheme of learning general rules, followed by refinement through learning exceptions to the general rules. In this case the general rules are still a valid default, but the specific case of the exception rule has higher priority than the general rule. This type of learning can be labeled as *incremental*.

Incremental learning schemes have the advantage of naturally encapsulating the common general to specific learning scheme. It also appears to be advantageous when the input examples (or rules) are more accurate. On the other hand, the training set scheme holds more promise when the input is noisy and has no natural priority of one example to the next. Both techniques have their place and hybrids may be advantageous.

A learning system using incrementally input rules can be maintained *consistent*. By consistent it is meant that no two rules which can be simultaneously matched and which output opposite values should be in the same rule set. This means that rules must be modified in order to maintain consistency. If new rules are given precedence, then old rules which could match with the new rule and which give different output, are deleted or modified such that matching cannot take place.

These rules can also be minimized such that the same information is represented by fewer rules or variables. This is also a type of generalization. For example, the deletion of don't care variables allows only critical variables to remain in rules, thus making critical variable generalization possible.

Consistency and Generalization in Incremental Systems

A class of new connectionist models which uses both incremental learning and critical variable generalization is ASOCS (Adaptive Self-Organizing Concurrent Systems) [3,4,7]. ASOCS is a parallel adaptive system which functions in two modes: processing and learning.

During processing, ASOCS functions like a parallel hardware circuit mapping boolean inputs to boolean outputs. During learning the systems accepts if-then rules in an incremental fashion and reconfigures the network so as to maintain consistency. ASOCS models guarantee learning of arbitrary boolean mappings, and learn any rules in time *O(log(n))* where *n* is the number of nodes or rules in the network. There are a number of different ASOCS learning algorithms and systems and discussion of their mechanisms is found elsewhere [3,5,6].

This paper discusses the basic knowledge input of an ASOCS system and how it is kept consistent at a high level, independent from a specific ASOCS implementation.

The atomic input to the system is called an *instance*. An instance is made up of a vector of boolean inputs and a single boolean output. For example:

A B' => Z'

B C D => C'

D E' => X

An instance specifies what the system should output if the current input matches the instance. So, for the instance D E' => X, the system must output X as high if D is high and E is low, regardless of the setting of any other input variables. This instance says nothing about what X should be when D is not high or D in not low.

The vector of input variables in and instance is called a *variable-list*.

An instance whose output is negated is a *negative* instance. An instance with a non-negated output is a *positive* instance. Thus, an instance can have a positive or negative *polarity*. Two instances with the same polarity are *concordant*, while two instance with opposite polarity are *discordant* with respect to each other.

Instances are input incrementally. The most recent instance is given precedence, although that is not the only possible strategy. The current totality of instances is called the *instance set (IS)*. An instance set is maintained *consistent*. In a consistent set no two discordant instances can simultaneously be matched.

Consistency between any two discordant instances is assured when there exists at least one *discriminant variable* for the two instances. A discriminant variable is an input variable which is negated in one of the instances and not negated in the other. Assume the following three instances.

(1) A B => Z

(2) B' C => Z'

(3) A C => Z'

The first two instance are consistent since they contain the discriminant variable B. Since B can never be simultaneously high and low, these two instance can never simultaneously be matched. Instances 2 and 3 are consistent because they are concordant. However, instances 1 and 3 are inconsistent because they are discordant and contain no discriminant variable.

If we assume that instance 3 is the most recent instance, then the system could have been made consistent by deleting instance 1. However, that is overkill in this case. We would like to keep all the information from old instances except for that which is specifically contradicted. In this case, we need to add a discriminant variable to instance 1. By adding C' to the variable list of instance 1, the instance set becomes consistent, while still retaining all previous information except that specifically contradicted by the new instance. This mechanism of maintaining consistency is called *discriminant variable addition (DVA)*.

We now overview how an instance set is maintained consistent when a new instance (NI) is introduced. All comparisons are pairwise between the NI and each old instance (OI). To do this we must classify how a NI can match with a NI. This is shown by example. Assume the variable list (we currently ignore polarity) of the NI is

A B' D

Assume the following variable lists of OI's.

A	Subset
A B' D	Equal
A B' D E	Superset
B' E'	Overlap
C G	Overlap
A B' D'	Discriminated

An OI is *subset* if its variable-list is a subset of the variables of the NI.

An OI is *equal* if its variable-list is the same as that of the NI.

An OI is *superset* if it has more variables than the NI, but every NI variable occurs in the OI's variable list.

An OI is *overlap* if there is no discriminant variable between the NI and the OI, and it is *not* subset, equal, or superset.

An OI is *discriminated* if it contains at least one discriminant variable (*D* in the example above) relative to the NI.

<u>Consistency: Discordant Instances</u>

Following are the modifications necessary for any OI discordant to the NI in order to maintain a consistent instance set. A NI is *broadcast* to all of the OI. In each case, the view is taken from an OI, how it matches with the NI, and what action should take place. In an actual ASOCS implementation, the logical modification to OI's is done in parallel in a self-organizing network.

I. OI Superset:

NI: A B

OI: A B C

Delete OI

II. OI Equal:

NI: A B

OI: A B

Delete NI

III. OI Subset:

NI: A B C

OI: A B

DVA on OI: (In this case the OI becomes A B C')

IV. OI Overlap:

NI: A B C

OI1: C D

OI2: D E

DVA on OI (Note that DVA can cause creation of multiple modifications. OI1 becomes A' C D *and* B' C D).

V. Discrminated:

NI: A B'

OI: A B C

No Change

<u>Minimization: Concordant Instances (Pairwise)</u>

An instance set is made *minimal* through deletion of redundant instances and variables. Complete minimality is not typically a goal due to its complexity. However, much minimizing can be done through pairwise comparison of the NI to OI's. This attains parsimony or *partial* minimization.

Minimization aids generalization by deleting don't care variables and discovering critical variables. This does one type of generalization. The mechanism of generalization for inputs which do not match the minimized instance set is dependent on system implementation. Both hamming distance or critical variable generalization can then be accomplished at the implementation level. This is discussed elsewhere [1,3,5].

There is one more important matching type between concordant instances for minimization. Assume the NI A B C and the OI's as follows:

A B'	one-difference subset
A B C'	one-difference equal
A B C' D	one-difference superset

Two instances are *one-difference* if they are concordant, contain exactly one discriminant variable, and are otherwise subset, equal, or superset. In this case the variable is called a *one-difference* variable.

Following are types of minimization possible for different matchings of OI to NI.

I. OI Superset:

NI: A B

OI: A B C

Delete OI

II. OI Equal:

NI: A B

OI: A B

Delete NI

III. OI Subset:

NI: A B C

OI: A B

Delete NI

IV. OI One-Difference Subset:

NI: A B' C

OI: A B

Rebroadcast the Modified NI without the one-difference variable

(Note here that it is never *necessary* to rebroadcast an instance to the network. It can aid parsimony, but can increase learning time. In an actual system this is an implementation decision).

V. OI One-Difference Equal:

NI: A B'

OI: A B

Delete OI &

Rebroadcast NI modified by deleting the one-difference variable

VI. OI One-Difference Superset:

NI: A B'

OI: A B C

Delete the one-difference variable from the OI &
Optionally rebroadcast the modified OI (see note above)

Consistency and Minimization in Priority ASOCS

Another scheme for maintaining a consistent instance set is to augment each instance with a priority [1]. Assume each NI is given a priority 1 higher than previous instances. Then if a conflict ever occurs between instances, the instance with the highest priority sets the output. This obviates the need for DVA (discriminant variable addition), thus guaranteeing that the size of the instance set grows by at most one, when any NI is presented.

Following is an overview of how OI's are modified in a priority instance system. Assume that the NI is always added with a higher priority unless specifically noted.

Consistency: Discordant Instances

I. OI Superset:

NI: A B

OI: A B C

Delete OI

II. OI Equal:

NI: A B

OI: A B

Delete OI

III. OI Subset:

NI: A B C

OI: A B

No Change

IV. OI Overlap:

NI: A B C

OI1: C D

OI2: D E

No Change

V. Discrminated:

NI: A B'

OI: A B C

No Change

<u>Minimization: Concordant Instances</u>

Definitions:

PR(I) - returns integer priority of the instance (I) for the current output variable. Higher number signifies higher priority.

Cont-Greater(OI) - returns true
if there exists
{I | (I contradicts NI) & (PR(I) > PR(OI))}
else false

Cont-Greater returns true for a specific OI-NI pair, if there exists a different OI (OI2) such that OI2 matches the NI and has priority greater than OI.

Following are types of minimization possible for different matchings of NI to OI.

I. OI Superset:

NI: A B

OI: A B C

Delete OI

II. OI Equal:

NI: A B

OI: A B

Delete OI

III. OI Subset:

NI: A B C

OI: A B

if Cont_Greater(OI) then Add NI
else Delete NI

NI: A B C D => Z

IS1: ... , A => Z', ... ,AB => Z,...,ABC => Z',...

IS2: ... , A => Z', ... ,AB => Z,...

(In these examples ISn represent different instance sets. The left-most instances have lower priority. Typically the multiple instance sets shown give examples of each possibility when an *if-then-else* strategy is used for modification.)

IV. OI One-Difference Superset:

NI: A B'

OI: A B C

Delete one-difference variable from OI & Add NI

NI: A B' => Z

IS1: ... , A => Z', ... ,ABC => Z,...,AD => Z',...

V. OI One-Difference Equal:

NI: A B'

OI: A B

if Cont_Greater(OI) Add NI & Remove one-difference variable from OI
else Delete OI & Rebroadcast NI modified by deleting
the one-difference variable

NI: A B' => Z

IS1: ... , A => Z', ... ,AB => Z,...,AC => Z',...

IS2: ... , A => Z', ... ,AB => Z,...

VI. OI One-Difference Subset:

NI: A B' C

OI: A B

if Cont_Greater(OI) Add NI
else Rebroadcast the Modified NI without the one-difference
variable and then,
if Cont_Greater(OI) then just add original NI
else continue with modified broadcast

NI: A B' C => Z

IS1: ... , A => Z', ... ,AB => Z,...,AC => Z',...

IS2: ... , A => Z', ... ,AB => Z,...,BC = > Z',...

IS3: ... , A => Z', ... ,AB => Z,...

Simultaneity

The question arises of whether the NI can be simultaneously tested against all OI's or do different actions require an ordering. The answer is they can be done simultaneously. However, for minimization, improved parsimony can be attained if the consistency modifications are done first, followed by minimization. For example:

NI: A B C => Z

IS1: ... , A => Z', ... ,AB => Z,...,ABC => Z',...

If ABC => Z' had not initially been deleted by consistency, the NI ABC => Z could not have been deleted by the OI AB => Z because cont-greater would still return true. Note that consistency is maintained either way.

We also noted that rebroadcast of a modified instance is optional. It can lead to improved parsimony at the cost of greater time complexity. Assume the following NI and IS.

NI: A B C' => Z

IS1: ... , A => Z', ... ,AB => Z,...,BC => Z,...

NI: A B' => Z

IS1: ... , A => Z', ... ,AB => Z,...,AC => Z',...

The NI can be minimized to A => Z by one-difference equal with AB => Z. If the A => Z is then rebroadcast, both Ac => Z' and A => Z will be deleted. Without rebroadcast, the system would have remained consistent, but less parsimonious.

Conclusion

This paper has discussed concepts of learning and generalization in connectionist systems. In particular, it has pointed out that there are a number of mechanisms for fulfilling these goals, each having advantages for specific classes of applications. Potential schemes for maintaining consistency and minimization for incremental systems were presented for two different rule models. Ongoing research seeks to improve speed of learning and accuracy of generalization in connectionist learning systems.

Bibliography

1. Hughes, B. *Prioritized Rule Systems*, M.S. Thesis, C. S. Dept., BYU, 1989.

2. Kohonen, T., *Self-organization and associative memory*, Springer Verlag, New York, (1984).

3. Martinez, T. R., *Adaptive Self-Organizing Logic Networks*, Ph.D. Dissertation, Technical Report - CSD 860093, University of California, Los Angeles, CA (May 1986).

4. Martinez T. R., Models of Parallel Adaptive Logic, *Proceedings of the 1987 IEEE Systems Man and Cybernetics Conference*, pp. 290-296, (October, 1987).

5. Martinez, T. R. and J. J. Vidal, Adaptive Parallel Logic Networks, *Journal of Parallel and Distributed Computing*, Vol. **5**, No. 1, pp. 26-58, (1988).

6. Martinez, T. R., Digital Neural Networks, *Proceedings of the 1988 IEEE Systems Man and Cybernetics Conference*, pp. 681-684, (August, 1988).

7. Martinez, T. R., Adaptive Self-Organizing Concurrent Systems, in *Progress in Neural Networks*, Ablex Publishing, 1989.

8. Martinez, T. R., Neural Network Applicability: Classifying the Problem Space, *Proceedings of the IASTED International Symposium on Expert Systems and Neural Networks*, pp. 41-44, August, 1989.

9. Rumelhart, D. and McClelland, J., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. I, MIT Press, (1986).