Context-Sensitive Weights for a Neural Network

Robert P. Arritt
Roy M. Turner*

Department of Computer Science
University of Maine
Orono, ME 04468–5752 USA
arritt@maine.edu, rmt@umcs.maine.edu

Abstract. This paper presents a technique for making neural networks context-sensitive by using a symbolic context-management system to manage their weights. Instead of having a very large network that itself must take context into account, our approach uses one or more small networks whose weights are associated with symbolic representations of contexts an agent may encounter. When the context-management system determines what the current context is, it sets the networks' weights appropriately for the context. This paper describes the approach and presents the results of experiments that show that our approach greatly reduces the training time of the networks as well as enhancing their performance.

1 Introduction

Neural networks are well known for their ability to separate continuous numerical data into a finite number of discrete classes. This functionality makes them a perfect candidate for converting real-valued data into symbolic values for a symbolic system. We will call such symbolic values "linguistic values", borrowing the term from fuzzy logic (e.g., [14]).

A problem arises though, in real-world situations where linguistic values are highly context-dependent. For example, an underwater agent may convert a depth of 5 meters into *TOO_DEEP* while in a harbor, but later, when it finds itself in the open ocean, 5 meters may be classified as *NOMINAL*. Thus, if we wanted a neural network to convert an agent's depth to a linguistic value we would need to encode context into the network, which would entail adding nodes and connections to the network. It is easy to see that as the number of contextual features and the number of possible contexts increases this would become unmanageable: in order to be context-sensitive, the network's size would make it impractical.

* This work was supported in part by the United States Office of Naval Research through grants N0001-14-96-1-5009 and N0001-14-98-1-0648. The content does not necessarily reflect the position or the policy of the U.S. government, and no official endorsement should be inferred.

This paper appeared in the Proceedings of the 2003 International and Interdisciplinary Conference on Modeling and Using Context (CONTEXT'03).

On the other hand, if a neural network could be constructed and trained for use within a single context, it could be much smaller, and it would be fast to train and highly accurate. A large number of these simpler networks could then do the work of the larger, more complex network. The problem would then become one of somehow identifying which network to use in which context.

We have developed an approach to making neural networks context-sensitive that takes this second approach. We solve the problem of deciding which network to use by making use of prior work on a context-management system that explicitly represents context an agent might reasonably be expected to encounter [13, 12]. With each contextual representation, or *contextual schema*, is stored information useful for the agent in the context. For an agent that uses (or that is) a neural network, contextual schemas contain the weights appropriate for the network to use in the context represented by the schema. Rather than encoding all the features of the context into the neural network's weights, the context-management system handles the problem of diagnosing the current context. This frees the network to encode only those features having to do with categorizing a value within that context. This greatly decreases both the complexity and training time of the neural networks. In addition, the classification error rate is on par with the fuzzy rule-based system currently used by our system for classification.

There has been some prior work aimed at developing context-sensitive neural networks. For example, Henninger *et al.* [7] have developed context-sensitive neural networks that use context to determine which network is to be used. Their work, however, makes use of a very simple form of context, i.e., an agent's distance from a location. In contrast, our approach relies on a much richer notion of context that incorporates numerous environmental factors. This approach not only allows us to better tailor the networks to the situations in which they will be used, but also to remove most of the contextual information from the networks themselves, thus reducing their complexity.

In the remainder of this paper, we discuss this approach in detail. Section 2 discusses our domain, autonomous underwater vehicle (AUV) control, and the agent, Orca, within which our network resides and that provides the context management functionality. Section 3 presents two neural network designs, one using the context-management system and one that does not, that perform one classification task important for AUVs, depth categorization. Section 4 presents the experiments we used to compare these two approaches, including the results of the experiments. Section 5 describes how the neural network is integrated into the context-management system, and Section 6 concludes and discusses future work.

2 AUVs, Orca, and Depth Management

Autonomous underwater vehicles are untethered submersible robots that are capable of carrying out untended underwater missions. They are useful for a variety of tasks in oceanography, ocean engineering, aquaculture, industry, and defense [2]. One of the major advantages of AUVs is their ability to operate

in an area without a human surface presence, which allows them to perform long-term missions without constant human supervision. AUVs can also operate under hazardous conditions, such as within minefields, under ice, and during inclement weather. They can also be combined into multi-AUV systems that can perform tasks such as autonomous oceanographic sampling and surveying [4]. Figure 1 shows the two EAVE (Experimental Autonomous VEhicles) AUVs that were used for developing AUV technology.

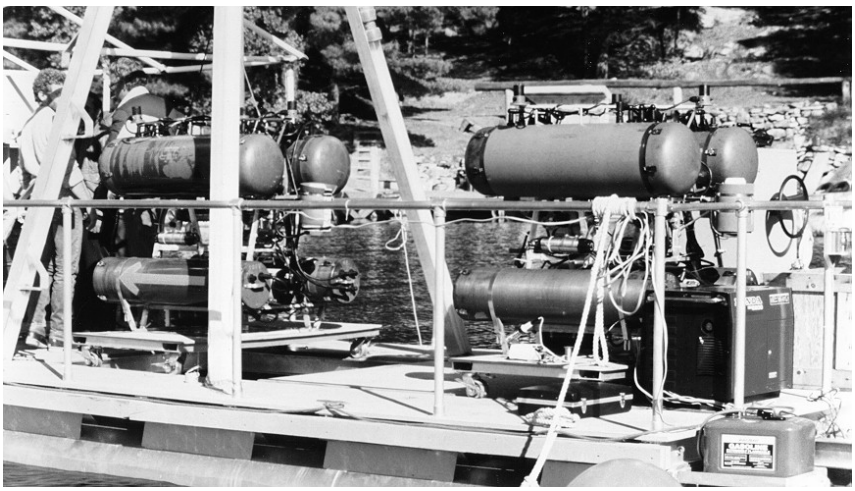

Fig. 1. Two EAVE vehicles on a support barge. Used by permission of the Autonomous Undersea Systems Institute (AUSI), Lee, NH.

The current state-of-the-art of AUV hardware technology is such that competent AUVs can be built and fielded. However, although there has been much recent work on intelligent control of AUVs (e.g., [9, 3]), the competent control software needed to carry out, successfully, a complex, autonomous, possibly long-term mission is largely lacking. For this, an AUV must have sophisticated artificial intelligence (AI) control software that can not only autonomously plan and perform the missions, but that can also respond appropriately to the unexpected events that are sure to arise within the highly dynamic undersea environment.

The Orca project [10] has the goal of creating an intelligent mission controller for long-term, possibly multiagent, ocean science missions. Orca, which is currently in development at the University of Maine, is a context-sensitive agent that will be able to recognize the context it is in and behave appropriately.

Orca's context-sensitivity is conferred by its context-management module, ECHO (Embedded Context Handling Object) [12]. Orca represents all contextual knowledge as *contextual schemas* (c-schemas) [13]. Each c-schema explicitly represents a *context*, that is, a recognizable class of problem-solving situations.[1]

[1] See [13] for a more complete description of our definition of context.

A c-schema contains a wide variety of contextual knowledge useful for the agent as it operates within the context represented. For example, c-schemas contain knowledge about: the expected features of the context; context-specific semantic information (e.g., the meaning of fuzzy linguistic terms [11]); how to cope with unanticipated events (how to recognize them, diagnose them, evaluate their importance, and handle them); goal priorities in the current context; which actions are appropriate for which goals in the context; and the appropriate settings of behavioral and perceptual parameters for the context.

C-schemas are stored in an associative long-term memory [8] that, when presented with features of the current situation, returns the c-schemas that most closely match it. A process of diagnosis then occurs to determine which of these evoked c-schemas actually are germane [1]. These are then merged to form the context object, which is a coherent picture of the current context. The knowledge in this object can be used by the agent to quickly decide how to behave appropriately for the context.

As an example of the usefulness of this approach, consider how an agent might determine the appropriate response to a catastrophic event such as a leak. For such a thing, there will be very little time in which to decide on a response, so complex reasoning after the leak may be impossible. Yet the appropriate response is context-specific. In a harbor, the AUV should probably land on the bottom and release a buoy, since that will avoid collisions with surface traffic. In the open ocean, however, landing might be disastrous: the bottom may be below the crush depth of the vehicle. Instead, since there would be very little likelihood of a collision, the appropriate response would be to surface and radio for help. If the agent always maintains an idea of what its current context is, then it can automatically take the appropriate response.

For the purposes of this paper, we are concerned with context-sensitive perception, that is, appropriate classification of sensory inputs. As an example, consider the AUV's depth envelope, that is, the range of depths that are allowable. This is highly context-sensitive. If the agent finds itself in a harbor, it should tighten its envelope to keep it away from the surface, where it is in danger of being hit by traffic, and above the bottom, where it may encounter debris that could ensnare it. If, on the other hand, the agent finds itself in the open ocean, it can loosen its envelope; the probability of surface traffic is minimal, so the agent only has to worry about staying above its crush depth.

3 Neural Networks for Depth Classification

Elsewhere, we have proposed a solution to the problem of context-sensitive classification of sensory data that was based on a fuzzy rule-based system [11]. However, we are interested in a mechanism that can learn from the agent's own experience or by being presented with training examples, since the AUV's operating conditions may change, and it may be difficult or impossible to obtain the fuzzy rules from human experts. Consequently, we have begun investigating the feasibility of using neural networks for this task.

If the neural network is going to classify (e.g.) depth, context must be taken into account. This can be done in two ways: either a large network with a large number of inputs can be trained with data from instances of all the contexts the AUV may encounter, or numerous smaller networks can each be trained with data from instances of a single context. The former kind of network must not only classify the agent's current depth, but also implicitly determine its current context in the process. To achieve this, the network will have to be provided not only with the current depth, but also with a large amount of environmental data that will help it determine the current context. The idea behind the smaller networks is that they will each specialize in classifying depth within a specific context. It will be up to the context-management system to determine which network is the appropriate one for the current situation. Since these networks are so specialized, they will require a few inputs, chiefly the agent's current depth.

The larger network will be very large indeed, and although it may be adequate for all contexts, it is unlikely it will be especially good for any particular one. Training time will be long for the network. The smaller networks will each be simpler to train, and they will obviously be highly-tailored to depth categorization for the context in which they are used.

Our approach is conceptually the latter. However, instead of using numerous small networks, we have a single network whose weights are set based on the context. The weights for the network that are appropriate in a given context are stored in the c-schema representing that context. When that context is recognized, the weights are retrieved from the c-schema and given to the neural network. During the context, the network then behaves as a highly-specific network. When the context changes, any changes to the weights, e.g., from learning sessions within the context, are then stored in the c-schema for use the next time the agent is in that context. A new c-schema is found for the new context, and its weights are loaded. This approach is particularly useful in a system such as Orca, in which the agent's context is already being recognized and represented.

In the next section, we discuss experiments we performed to determine if a large number of small networks, such as is effectively used in our approach, is as good as or better than a large, multi-context network. In the following section, we discuss how this can be integrated into Orca.

4 Experiments and Analysis

There are several criteria for determining if the smaller networks are better suited for our task. First, the smaller networks would have to perform the classification task at least as well as the larger network. Second, since we would be training numerous smaller networks, they would have to train much faster than the larger network. Finally, we would have to show that the size of the larger network grew at an unacceptable rate as more contexts were added to the system. In this section we will show that the set of small networks classify better than the

single large network. We will then show, in a less formal manner, that the smaller networks train faster and that the larger network grows unmanageably large.

For the sake of the following experiments we made a few decisions regarding our networks. All of our networks were two layer feed-forward networks [5] that had 5 output nodes. Each of these output nodes stood for one of our linguistic values: *TOO_SHALLOW, SHALLOW, NOMINAL, DEEP, TOO_DEEP*. The output node with the highest value was the classification that we chose. Our smaller networks each had five nodes in their hidden layer. The number of hidden nodes was determined by starting with a network with two hidden nodes this network was trained with two-thirds of the training data. Next, the performance of this network was evaluated using the final third of the training data. A node was added and the network was retrained and tested. This process was repeated until an acceptable level of performance was reached. This process is a form of cross-validation[6]. The number of nodes in the hidden layer of the larger network varied with the number of contexts, but was also chosen via cross-validation. Training was done with the Levenberg–Marquardt algorithm[5]. All data was collected via Matlab programs running on an 1.5 GHz Intel Xeon PC under the Linux operating system.

When we set out to test the classification performance of our networks we decided to keep the number of contexts small in order to simplify the training of the larger network. Thus, we chose to test the performance of the networks at classifying depths within a harbor, on a shoal, and in the open ocean. In order to implement the larger network we had to determine the most salient features of the contexts and use them as parameters. We chose the following seven parameters: depth, distance from shore, water column depth, density of fish, density of debris on the bottom, and the density of surface traffic. The training data for all of these parameters was then generated randomly via a normal distribution around the accepted values for each parameter. Since we wanted these networks to perform as well as our current fuzzy system, we used it to generate the correct classifications for our training data.

Next, the large network was trained with the entire training set while the smaller networks were trained with the data from there respective contexts. Upon completion of training, the networks were tested with more randomly generated data. The rate of classification error can be seen in table 1. Performing a t-test with $H_0 : \mu_{large\_net} = \mu_{small\_nets}$ and $H_1 : \mu_{large\_net} > \mu_{small\_nets}$ we get a t statistic of 2.218 and a critical value of 1.649 when $\alpha = 0.05$. Thus, we reject the null hypothesis and we can say that the smaller nets generate, on average, fewer errors.

	Harbor	Shoal	Ocean	μ	σ^2
Large Network	0.0216	0.0197	0.0202	0.0205	0.000008
Small Networks	0.0378	0.0118	0.001	0.0169	0.00008

Table 1. Rate of Classification Errors

Another important aspect of the networks is how long they take to train. Each of the smaller networks take about the same amount of time to train, thus as we increase the number of smaller networks, the training time increases linearly. Consequently, given the one-to-one correspondence between networks and contexts in this approach, training time overall is linear in the number of contexts. It is intuitive that the smaller networks should train faster, but we have to verify that as the number of contexts is increased that the training time for the larger network increases faster than the training time for the set of smaller networks. Table 2 and figure 2 show how the training times for the networks grow as the number of contexts increase from one context to six. It is apparent that the training time for the larger network is increasing much faster than that of the smaller networks. We attribute this to the fact that as the number of contexts increases, the larger network grows both in size and in the number of inputs. It also requires more training data to allow it to distinguish between contexts.

# of Contexts	Large Network	Small Networks
1	12 seconds	11.5 seconds
2	1 minute 9 seconds	24 seconds
3	2 minutes 30 seconds	37 seconds
4	6 minutes 23 seconds	49 seconds
5	10 minutes 16 seconds	62 seconds
6	18 minutes 37 seconds	77 seconds

Table 2. Training times as the number of contexts is increased

Our final test was to find out how fast the number of nodes in our larger network increased as the number of contexts increased. We wanted to know this because as this number increased, we would increase the runtime of the network and the training time of the network. The number of nodes needed was calculated through simple cross-validation[6], as discussed earlier. Table 3 shows the results of this experiment. It appears that the network is growing in an exponential fashion. It should be apparent that this will soon cause the larger network to become both inefficient and impossible to manage.

This set of experiments demonstrates several things. First it shows us that the smaller networks perform better at the classification task than the larger network. Next, we saw that the training time of the set of smaller networks increases linearly as the number of contexts increases and the training time of the larger network appears to increase at a much faster rate. Finally, we saw that the number of nodes in the larger network exploded as more contexts were added. Taking this all into account, the logical choice is to use the set of smaller networks for our classification tasks.

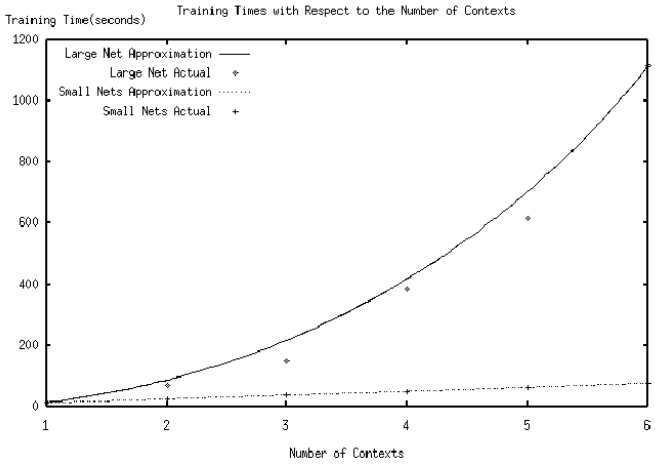

Fig. 2. Graph of training time as contexts are increased

# of Contexts	# of nodes in hidden layer
1	5
2	8
3	20
4	32
5	63
6	112

Table 3. Number of nodes needed to classify depth

5 Embedding the Network within a Schema

As previously discussed, Orca's context-management system explicitly represents contexts in the form of contextual schemas. A c-schema incorporates everything that is deemed important about a given context. This should include information about the operation of neural networks in the context represented. This can be done in one of two ways.

As mentioned earlier in this paper, the only difference between all of the smaller networks was the weights. This static architecture makes it very easy to store the weights in a c-schema. The simplest, although slightly short-sighted, way to make these networks context-sensitive would be to save the weights in a simple list. This makes it very easy for the context manager to manipulate the weights. When the weights are needed they can simply be read in order and slotted into the appropriate spot in the neural network. Likewise, if the context manager deems it necessary to retrain a network the new weights can be easily changed.

The problem with this method is that if in the future we decide to change the structure of the network we would have to rewrite all of our network handling functions. To solve this problem, we could encode the structure of the network into the list. By nesting lists we could generate any feed-forward network structure while maintaining a simple representation. Figure 3 shows how this could work. The idea behind this scheme is that each layer is represented as a list within the master list. Then the set of each node's input weights are stored within the layer list. We intend to use this method in our system, although we will have to be alert for performance degradation from the greater complexity it entails.

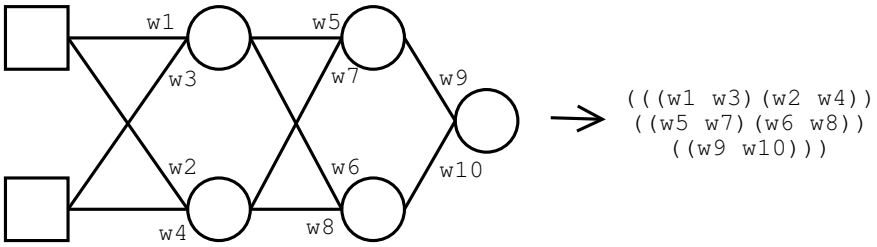

Fig. 3. Converting a network to a list

6 Conclusions

This paper presents a technique that allows a neural network to be context-sensitive without becoming unmanageable. This is achieved by allowing the contextual aspect of the problem to be handled by a symbolic context-management system. By using this technique we are able to to use the efficient classification and learning algorithms associated with neural networks without the explosion in network size that comes with the addition of more contextual data.

While we have only shown this technique to work for our classification task, we believe that it could be applied to many other neural network applications where the network's task and/or domain can be partitioned naturally into contexts in which the network must operate.

This project has raised many questions that we hope to look at in future work. First, we have to implement and test a method for retraining networks during a mission. One possibility is for the context manager to recognize that it is in a training context (by retrieving a corresponding c-schema) and to allow the weights to change while in that context. Another avenue that we are currently exploring, not only with respect to these networks but with our entire notion of context, is how to merge disparate contextual information. For example, an

AUV may be in the context of a harbor and also a rescue mission. The harbor context may tell the AUV that the bottom of the harbor is too deep while the rescue context will want to eliminate the idea of a depth envelope entirely. Finally, another thing that we have to look at is whether we have lost anything by not using the larger network. This paper shows that it does not classify as well, but we may be able to draw other information from it such as when to merge contexts, or we may be able to use it as an indicator for when context is changing.

References

1. Robert P. Arritt and Roy M. Turner. A system for context diagnosis. Technical Report 2003–1, University of Maine Computer Science Department, 2003.
2. D. Richard Blidberg, Roy M. Turner, and Steven G. Chappell. Autonomous underwater vehicles: Current activities and research opportunities. *Robotics and Autonomous Systems*, 7:139–150, 1991.
3. Don Brutzman, Mike Burns, Mike Campbell, Duane Davis, Tony Healey, Mike Holden, Brad Leonhardt, Dave Marco, Dave McClarin, Bob McGhee, and Russ Whales. NPS Phoenix AUV software integration and in-water testing. In *Proceedings of the 1996 IEEE Symposium on Autonomous Underwater Vehicle Technology (AUV'96)*, pages 99–, Monterey, CA, USA, June 1996.
4. T.B. Curtin, J.G. Bellingham, J. Catipovic, and D. Webb. Autonomous oceanographic sampling networks. *Oceanography*, 6(3), 1993.
5. M. T. Hagan and M. Menha. Training feedforward networks with the Marquardt algorithm. *IEEE Transactions on Neural Networks*, 5(6):989–993, 1994.
6. Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice-Hall Inc., 1999.
7. Amy E. Henninger, Avelino J. Gonzalez, Michael Georgiopoulos, and Ronald F. DeMara. A connectionist–symbolic approach to modeling agent behavior: Neural networks grouped by context. In *Proceedings of the Third International and Interdisiplinary Conference on Modeling and Using Context*, pages 198–209, 2001.
8. J.H. Lawton, R.M. Turner, and E.H. Turner. A unified long-term memory system. In *Proceedings of the 1998 IEEE International Conference on Case-Based Reasoning*, 1998.
9. Henrik Schmidt, James G. Bellingham, Mark Johnson, David Herold, David M. Farmer, and Richard Pawlowicz. Real-time frontal mapping with AUVs in a coastal environment. In *Proceedings of the IEEE Oceanic Engineering Society Conference (OCEANS'96 MTS)*, pages 1094–1098, 1996.
10. Roy M. Turner. Intelligent control of autonomous underwater vehicles: The Orca project. In *Proceedings of the 1995 IEEE International Conference on Systems, Man, and Cybernetics*, 1995.
11. Roy M. Turner. Determining the context-dependent meaning of fuzzy subsets. In *Proceedings of the First International and Interdisiplinary Conference on Modeling and Using Context*, 1997.
12. Roy M. Turner. Context-mediated behavior for intelligent agents. *International Journal of Human–Computer Studies*, 48(3):307–330, March 1998.
13. Roy M. Turner. A model of explicit context representation and use for intelligent agents. In *Proceedings of the Second International and Interdisiplinary Conference on Modeling and Using Context*, 1999.

14. L. A. Zadeh. A theory of approximate reasoning. In J. Hayes, D. Michie, and L.I. Mikulich, editors, *Machine Intelligence 9*, pages 149–194. Halstead Press, 1979.