The Governor Architecture: Avoiding Catastrophic Forgetting in Robot Learning

Jeremy Stober & Lisa Meeden
Computer Science
Swarthmore College
Swarthmore, PA 19081

Douglas Blank
Computer Science
Bryn Mawr College
Bryn Mawr, PA 19010

Abstract

The governor architecture is a new method for avoiding catatrophic forgetting in neural networks that is particularly useful in online robot learning. The governor architecture uses a categorizer to identify events and excise long sequences of repetitive data that cause catastrophic forgetting in neural networks trained on robot-based tasks. We examine the performance of several variations of the governor architecture on a number of related localization tasks using a simulated robot. The results show that governed networks perform far better than ungoverned networks. Governored networks are able to reliably and robustly prevent catastrophic forgetting in robot learning tasks.

1. Introduction

The back-propagation algorithm for training neural networks can theoretically approximate any function to an arbitrary degree of precision. However, in practice, the successful application of back-propagation to a particular problem requires careful planning to determine the appropriate network architecture, parameter settings, and training process. In fact, the success of back-propagation is highly dependent on the quality of the training data (Tarassenko, 1998). It is crucial that the training data be comprehensive, including all important categories, as well as balanced, including a relatively equal number of examples of each category interspersed regularly throughout the data. An improper data set can lead to catastrophic interference. That is, the network can completely forget what was previously learned as new patterns are trained.

There are two main methods for attempting to solve the catastrophic forgetting problem: limit the amount of overlap in hidden layer patterns, or continue to rehearse prior input patterns (French, 1999). Our solution, called a neural network governor, provides a method for determining which patterns represent key events that merit rehearsal.

Historically, a governor is a mechanical feedback device used to automatically control an engine. It was probably first used to regulate windmills, but became well-known in Watt's steam engine (Denny, 2002). Briefly, a mechanical governor is a pair of weights attached to a spinning vertical axle of an engine. As the axle spins faster, the weights rise up due to centrifugal force causing a lever to limit the power to the axle, which slows down the engine. As the engine slows down, the weights drop, shifting the lever in the other direction, which supplies more power causing the engine to speed up again. This results in a self-regulating engine that maintains a particular speed.

By analogy, we propose a similar device to automatically regulate the flow of training patterns into a neural network being trained with the back-propagation algorithm. Such a mechanism is necessary in applications of online learning, such as developmental robotics (Blank et al., 2002), where the tasks and the environment may not be known in advance. Instead, the training process must be able to autonomously adapt the training set to new situations as they arise.

2. Catastrophic Forgetting

Training a network online over an input sequence with long subsequences of similar inputs results in poor performance. Catastrophic forgetting provides an explanation for this impediment to learning. Subtasks comprised of long subsequences reallocate connection weights, invalidating previously learned subtasks. Large subsequences of similar inputs corresponding to single subtasks will catastrophically interfere with previously learned subtasks based on previous subsequences.

For example, McCloskey and Cohen conducted an experiment where they trained a neural network on ones addition facts and then twos addition facts. They found that network performance on ones addition facts decreases dramatically very early in the training process on twos addition facts (McCloskey and Cohen, 1989). This work provided strong evidence against the widely held opinion that neural network performance when trained

in successive tasks would degrade gracefully.

In off-line training, interference of this sort can be reduced or eliminated completely by interspersing the subsequences. In the context of developmental robotics, an autonomous solution to reordering the input sequence is desirable.

As mentioned, several solutions have been developed to mitigate catastrophic forgetting in neural networks. Modification of the learning algorithm to promote localist representations in hidden layer activations have proven effective. ALCOVE is perhaps the best example of this kind of algorithm (Krushke, 1992).

Another approach to solving the catastrophic forgetting problem is to continually rehearse previously learned tasks. In the absence of data from previous tasks, pseudo-patterns are continually inserted into new training data. Pseudo-patterns are generated by presenting the network with random input. The output produced along with the random input form a pseudo-pattern (French et al., 2001, French, 1999, Robins, 1998). Combining two networks and pseudo-pattern rehearsal has proven successful at limiting catastrophic forgetting (French et al., 2001). The complimentary memory centers in the neocortex and hippocampus have provided the dual network solution with a biologically probable basis (McClelland et al., 1995, French et al., 2001).

The governor architecture falls under the general category of solutions that utilize rehearsal to overcome catastrophic forgetting. Unlike the dual network models that exploit properties of pseudo-patterns, the governor architecture uses vector quantization to excise repetitive data while identifying and saving important events from the input data for continuous rehearsal.

We also believe that ALCOVE (and other models that attempt to limit hidden layer overlap such as French's activation sharpening) can add substantially to solving the problem of catastrophic forgetting. However, instead of attempting to create 'categories' inside hidden layer representations, we have moved the categorization process outside of the network.

3. Governor Architecture

The governor architecture has three main components, a categorizer, a buffer, and a neural network. The categorizer labels the input stream, consisting of both sensor data and the desired output, according to a set of codebook vectors. These labels are used to determine events. Events are defined as changes in labeling. When an event occurs, the current input is placed in a buffer. The neural network trains on the input vectors stored in the buffer.

The vectors that train the neural network are asynchronous to the vectors of the original input stream. When training is finished, the categorizer and buffer components are removed and the neural network is run synchronously on input data.

The categorizer serves as an event extractor. The constraints on the choice of this algorithm forced by the robotics domain are: *minimal training time*, *dynamic category generation*, and *noise tolerance*. Minimizing training time is always an important requirement in any machine learning task. Since robotics tasks vary and often take place in rich environments, the number of classes of events is not known in advance, so the categorization technique must allow for dynamic category generation. Additionally, since robot sensors are inherently noisy, any categorization method must be noise tolerant to avoid treating small variations in sensor inputs as distinct events.

The categorization component of the governor architecture could be performed by a number of models, such as the self-organizing map (Kohonen, 2001). In this report we have chosen to explore the Resource Allocating Vector Quantizer (described below) because it appears to best meet the above criteria.

3.1 Resource Allocating Vector Quantizer

A formal description of the RAVQ algorithm can be found in (Linåker and Niklasson, 2000a). An extension of the RAVQ, the Adaptive Resource Allocating Vector (ARAVQ), is described in (Linåker and Niklasson, 2000b).

Figure 1 shows the structure of the RAVQ algorithm. For each time step t, a parameter, n, specifies how many of the previous inputs to store in the FIFO buffer $X(t)$. In first stage of the RAVQ algorithm, these n inputs are averaged to generate the moving average vector, $\bar{x}(t)$. Averaging inputs in this way reduces noise. A buffer that is too small will fail to reduce noise. A buffer that is too large will decrease the sensitivity of the RAVQ to event changes. The use of the buffer in the RAVQ algorithm satisfies the *noise tolerance* criterion for event extraction in robotics tasks.

For the second stage, the moving average vector is compared to existing model vectors stored in the model set $M(t)$. If the moving average vector does not fall within a specified distance δ, using the Euclidean metric, of any existing model vector, then a new model vector may be allocated, provided the moving average is a good representation of the current input buffer within a specified threshold of ϵ. The ability to generate new model vectors upon encountering novel situations satisfies the *dynamic codebook vector generation* criterion.

In the third stage, the RAVQ maps the moving average vector to the closest existing model vector. By examining the sequence of mappings produced by the RAVQ algorithm, the governor architecture excises multiple similar input sequences from neural network training.

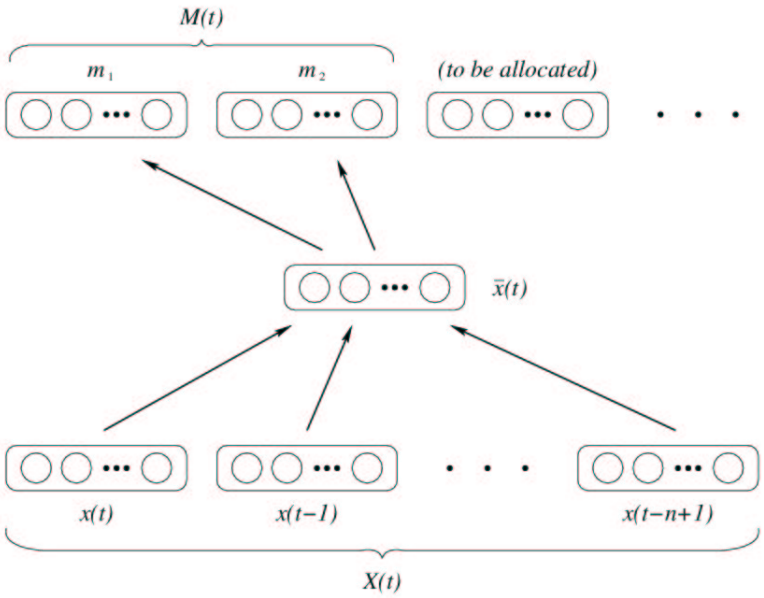

Figure 1: A diagram of the RAVQ architecture from (Linåker, 2003, page 60). $X(t)$ represents a FIFO buffer of n previous input vectors that are averaged to generate $\bar{x}(t)$, the moving average vector. This moving average is then compared to the current set of model vectors $M(t)$. If no good match is found, a new model vector may be allocated.

Since the RAVQ is processing concatenated input and target vectors, masking is employed to adjust the vectors so that input and target components are equally weighted.

Once the buffer $X(t)$ is full, new model vector creation and mapping proceed normally. So initialization of the RAVQ algorithm only requires n steps. Since model vectors are generated dynamically in time, effective event extraction occurs very shortly after initialization, satisfying the *minimal training time* criterion.

3.2 RAVQ Governor

An event is defined as a change in mapping from one model vector to another model vector. Each model vector represents a category. When an event occurs, the current input to the RAVQ at the time of the event is placed into another buffer, termed the training buffer. The neural network trains on the buffer of vectors indicated by the RAVQ as precipitating events. Since events are rare, the buffer of event vectors is circular, allowing the network to train over the buffer multiple times in the absence of new events.

Three different versions of the vector-quantization-based governor architecture are considered. The basic architecture uses the RAVQ algorithm with a circular training buffer size of 50. The RAVQ moving average buffer size is 5, ϵ is 0.2, and δ is 0.8.

3.3 ARAVQ Governor

A modification to the basic architecture employs the ARAVQ algorithm. The ARAVQ algorithm has an additional learning parameter, α, that determines the degree to which existing model vectors can be tuned to better reflect associated vectors in the input stream. This modification uses the same buffer size and parameters as the RAVQ governor with the additional parameter α set to 0.02.

3.4 ARAVQ Balancing Governor

The second modification of the basic design discards the fixed circular buffer size. Instead each model vector maintains an individual FIFO buffer of the last s inputs that mapped to that model vector. For the experiments described below the size of the individual buffers is set to 5. The other parameters are identical to those of the ARAVQ governor.

At each time step the current input is placed in the winning model vector's history buffer, replacing the oldest input in that buffer if the buffer is full, or filling a vacant position in that buffer if the buffer is not full. This scheme has the advantage of dynamically adjusting buffer size to reflect the number of model vectors. Also, the network trains across the buffers as if they were a single flat array. Contiguous vectors in that array be-

long to neighboring model vectors, so all the histories are interspersed, preventing blocks of identical data in the training set.

In this buffering scheme, stored training data is not necessarily connected to events. In the previous version of buffering, new vectors placed into the training buffer were associated directly with changes in the mapping made by the RAVQ algorithm. The data collected in this buffering scheme is more closely associated with the individual model vectors and less with points of change between model vectors.

3.5 Discrete and Random Governors

Two governor designs that did not use vector quantization were also tested. The discrete governor samples at a fixed rate t. Every t time steps this governor architecture samples from the input stream into a circular buffer of size 50. The discrete governor is an approximation of a vector quantization based governor in that it samples from the input stream at a rate much less than real time. However, it is not sensitive to external changes in the state of the environment. For regular repeating environments simply sampling at a rate complimentary to the frequency of events should yield similar results as the governor architectures mentioned above. In more dynamic environments that require robust methods, the discrete governors' lack of attention to the state of the environment is a severe deficiency.

Like the discrete design, the random governor samples independently of changes in the environment. Instead of sampling at a fixed rate, the frequency of sampling is stochastically determined. The wait time between samples is calculated using a matrix of probabilities.

Both the discrete sample rate and random sample rate are determined using data collected from the sampling rates recorded by the ARAVQ governor.

4. Experiments

The localization task explored here required the neural network to identify which of four rooms a simulated robot occupied at each time step. A preset wall following algorithm controlled the robot. The robot followed the outer walls of the world (Figure 2) in a circular clockwise pattern as indicated by the path markers, starting in the lower left corner. The width of the simulated robot was about half the width of the doorways. The robot was slightly longer than it was wide.

The teacher function output a unique four bit basis vector label for each room. For a small distance to either side of a doorway, the teacher function linearly transitioned from one label to the next label.

The neural network architecture was a standard three-layer feed forward neural network. The input layer was size 20. The network inputs included normalized sensor

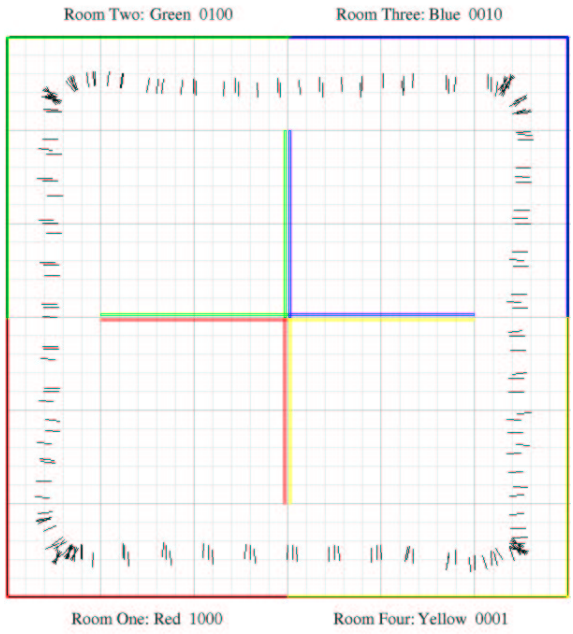

Figure 2: Basic localization environment created in the Stage simulator. The labels indicate the color of the walls and the orthogonal bit vectors associated with each room. The hash marks indicate the position and orientation of the robot at various intervals along its path.

data from each of sixteen sonar sensors and a four bit vector identifying the visible colors in the front field of view. The hidden layer was size 10, and the output layer was size 4. The neural network was trained to identify the room based on the inputs using the standard back-propagation algorithm with momentum. The learning rate of the neural network was 0.2 and the momentum was 0.9. The parameters for back-propagation remained constant over all experiments.

Several different experiments were run to test various aspects of the governor architectures described above. Each of the governor architectures, including an ungoverned network, were run for at least eight trials on each experimental variation. The number of trials depended on the number of available computers at the time of the experiment. For some experiments as many as sixteen trials were completed.

Player/Stage provided the simulation platform for these experiments (Gerkey et al., 2003). The robot definition simulated the capabilities of a ActivMedia Pioneer 2 with 16 sonar sensors and blob vision capability. The Pyro environment was used to develop and execute the controlling code for each experiment(Blank et al., 2003).

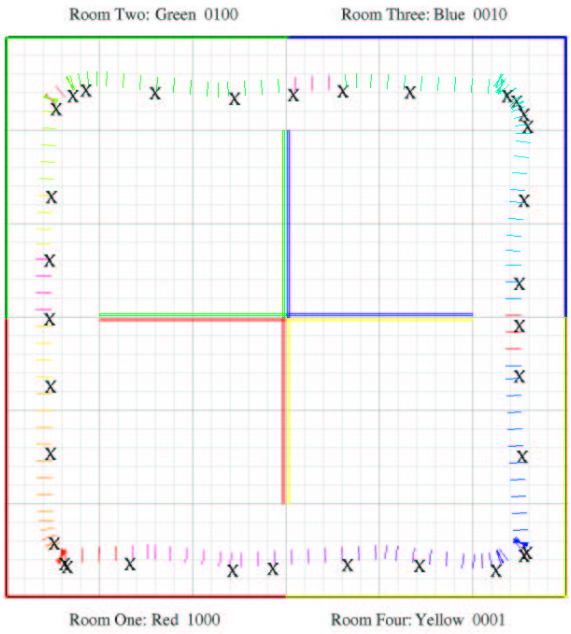

Figure 3: Event extraction. The X's mark event detections by a RAVQ during one complete cycle around the world. The modified experimental world shows a similar pattern of events.

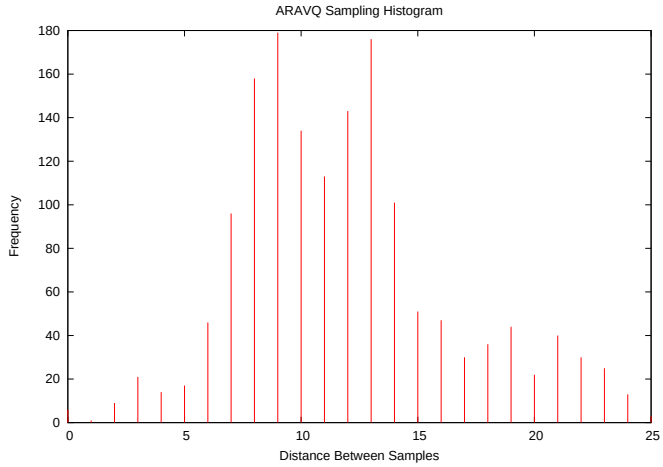

Figure 4: Summary of time step delays between event extractions. This is a typical sampling rate histogram for ARAVQ governor. Each vertical bar represents the frequency over 5,000 training steps of each associated sampling delay. For example, there were 180 times when events were extracted after a delay of nine time steps. The RAVQ based governor produced a similar histogram.

4.1 Basic Localization

The goal of the initial localization experiment was to show that governed neural networks could avoid catastrophic forgetting in the presence of substantial blocks of repetitive training data where ungoverned neural networks would fail. For this experiment, the neural networks were trained for 5,000 time steps.

During training, the RAVQ governor generated an average of 35 model vectors with a range of 33 to 38 model vectors. The ARAVQ governor and balancing governor generated between 31 and 39 model vectors with an average of 34 model vectors. Figure 3 shows a typical case of the distribution of events detected by a RAVQ during one complete circuit around the world. The locations of model vector changes are marked with X's.

Figure 4 shows a typical histogram of sampling frequency for the ARAVQ governor. The vertical bars represent the frequency over 5,000 training steps of each delay between samples. The RAVQ based governor shows similar results. Normalizing this histogram provided the sampling delay probabilities for the random governor. The average sampling frequency from this data served as the fixed sampling rate for the discrete governor.

Following training, the networks were tested for 1,000 time steps against the teacher. The average error over the testing period for each trial was calculated. These average errors for each type of governed network were then used to calculate 98% confidence intervals.

A network that demonstrates catastrophic forgetting will continuously output a single label. The robot will pass through the room labeled by the network approximately one quarter of the testing period. When that occurs the network's error will approach zero. The other three quarters of the time the network's error will approach two. A network that has catastrophically forgotten should have an average error approaching $0.75 \times 2 = 1.5$. Since networks are only trained to within a tolerance of 0.05 of the actual value, any average error in the range 1.35 to 1.5 indicates complete failure. Values approaching this range represent very poor performance.

Figure 5 describes the results of the initial experiment. All the governor architectures learned the localization task. The networks trained on the raw data without governor support failed to learn the localization task.

The Figures 6 and 7 represent typical error plots for networks during the testing period for the basic localization task. The dashed bars represent points of transition between rooms. Due to the gradient nature of the teacher and perceptual aliasing near room boundaries, most error for trained networks occurs during these transitions.

Figure 6 shows the error plot for the ARAVQ balancing governor architecture. Note that most of the error is occurring at room boundaries where it is difficult to determine the robot's location. All other governor architectures including the discrete and random variations showed similar performance.

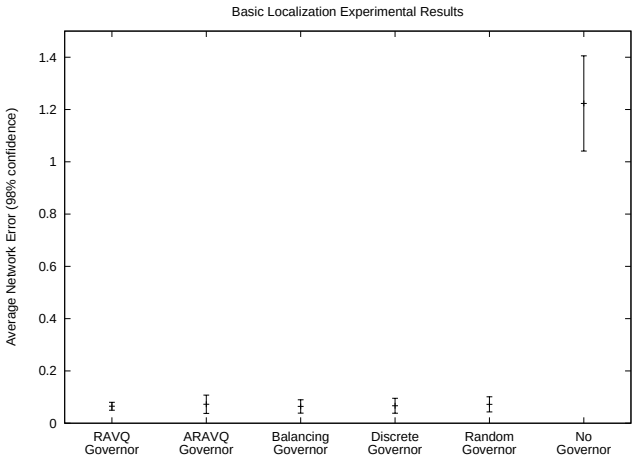

Figure 5: Summary of results for the basic localization experiment. 98% confidence intervals for each network were generated based on average error during testing of each network over multiple trials. All the governed networks performed significantly better than the ungoverned network.

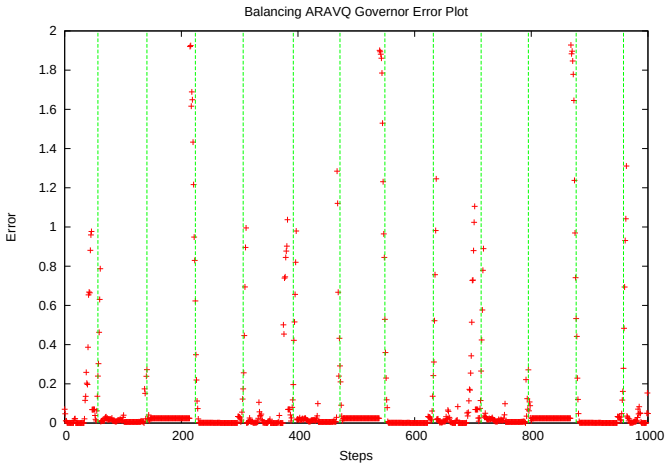

Figure 6: Testing error plot for network successful in basic localization. Room boundaries are delimited by vertical lines. Perceptual aliasing around room boundaries is the most likely cause of the error peaks.

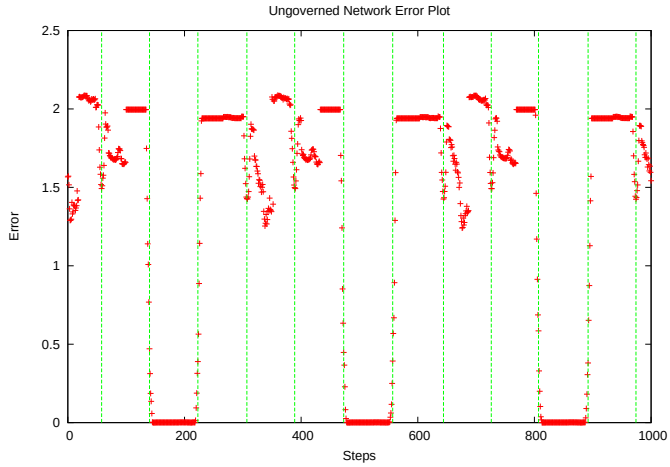

Figure 7: Testing error plot for network unsuccessful in basic localization. Regions of low error correspond to the third room. Analysis of the training set shows that the ungoverned network ended training on data from the third room.

Figure 7 shows the error plot for the ungoverned network architecture. The sections of low error correspond with the robot traveling through the third room. Analysis of the training set reveals that the last data that the ungoverned network trained on was collected in the third room. The ungoverned network clearly reproduces only the aspect of the localization function learned at the end of training.

4.2 Color Aliasing

In the second experiment, two of the rooms shared a single color and could only be distinguished using sonar data. This modified task required that the neural networks make fine distinctions across different sensor modalities. Color blob sensing alone cannot distinguish two rooms of the same color. Only by fusing sensory modalities will a network successfully learn to identify each room. Since governed neural networks train on sparse subsets of the available training data, verifying that this sampling still allows networks to make fine distinctions during training is necessary.

Figure 8 shows the modified world with the robot path indicated by the hashes. Except for the modified world, the experimental setup was identical to the basic experiment.

Figure 9 shows the results of the color aliasing experiment. The various architectures all performed slightly worse in this task than in the first experiment. The increased likelihood of perceptual aliasing may account for the small decrease in performance for the governed networks. The ungoverned network, as expected, failed to learn this modified task as well.

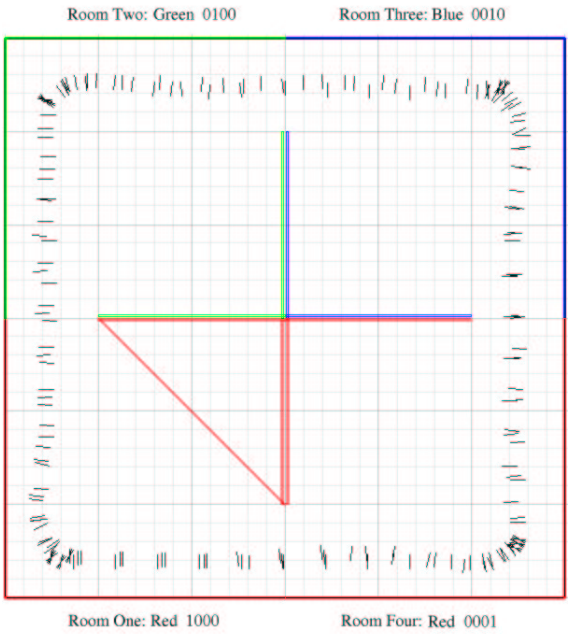

Figure 8: The modified experimental world contains two rooms whose walls are identical in color and are only distinguishable by the shape of the inner walls.

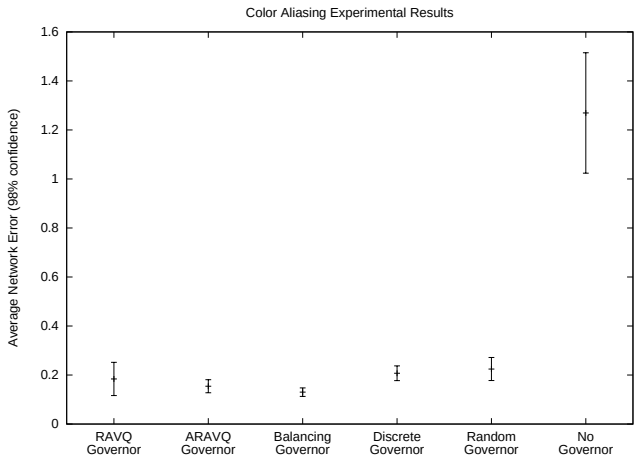

Figure 9: Summary of results for the color aliasing experiment. 98% confidence intervals are shown. The ARAVQ balancing governor performs statistically better than both the discrete and random governors.

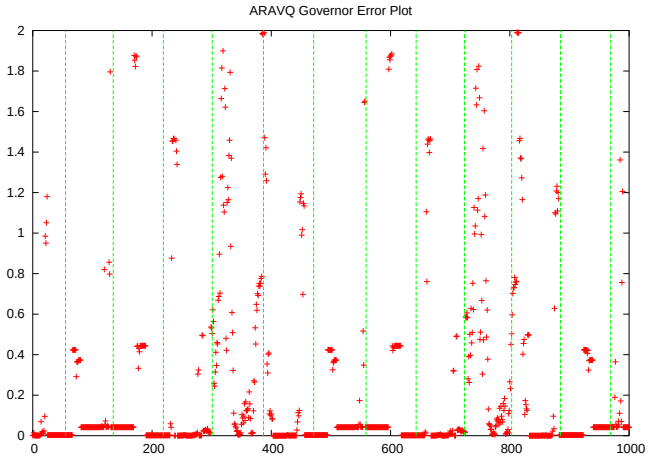

Figure 10: Testing error plot for the ARAVQ governed network. Error peaks tend to occur in the first room, which is one of the two red rooms. This indicates some perceptual aliasing between rooms with identical colored walls.

The ARAVQ balancing governor performs statistically better than both the discrete and random governors. However, all governed networks trained well enough to perform localization.

Figure 10 shows the error plot for the ARAVQ governor architecture. Mislabelings occur when the robot is in the first room, which is one of two red colored rooms. Some variation in the location of perceptual aliasing is evident across trails and governor architectures. However, most of the error occurs in the corner of the first room. The right, slanted wall is the most distant at this point in the room, making the room appear similar to room four. The error plots for the ungoverned networks are predictably similar to the basic experiment.

4.3 Simulated Stall

In the third experiment, a simulated stall was introduced during training using the original world with four distinctly colored rooms. The various architectures trained for 5,000 steps while the robot followed walls. The robot then stopped and the architectures trained on data from the motionless robot for another 5,000 steps.

The simulated stall tests the robust nature of each architecture. The governor architectures employing vector quantization are aware of the status of the environment. In the absence of new events, these architectures will only train the neural network on existing training data already in the buffer.

Since the random and discrete governor architectures are not aware of the environment and are not sensitive to event changes, the training buffers of these architectures will fill with repetitive information, causing degradation

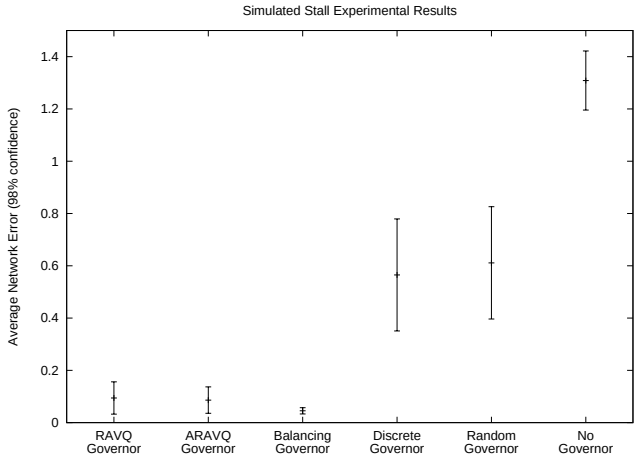

Figure 11: Summary of results for the simulated stall experiment. 98% confidence intervals are shown. All governor architectures employing vector quantization clearly perform better than the discrete and random governor architectures.

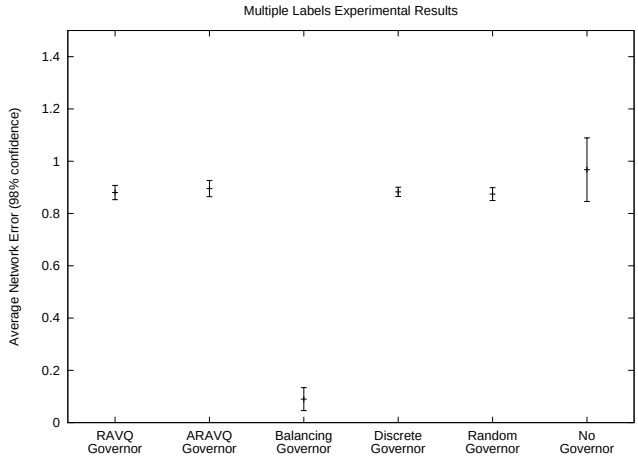

Figure 12: Summary of results for the multiple labels experiments. 98% confidence intervals are shown. The ARAVQ balancing governor performs statistically and significantly better than all other architectures.

in the quality of training.

Testing for the simulated stalling experiment lasted 1,000 steps. Figure 11 shows the results of simulated stalling.

The discrete and random governor architectures, nearly equivalent to the vector quantization based governors in the two previous experiments, are significantly worse at coping with stall situations. Some trials with the discrete governor did show robust behavior despite the long period of over training during the simulated stall. This could be due to the relative position of the stall in combination with variation on how well the network learned prior to the stall.

The ability to cue training to events in the environment confers upon RAVQ and ARAVQ based governors a distinct and robust advantage over both the discrete and random counterparts. The network trained with the discrete governor lost the ability to label all the rooms correctly, showing systematic error towards a reduced labeling. During the stall, the ARAVQ governor did not detect any new events, and so the network trained on previously collected event data in the training buffer, thus preventing degradation in performance.

4.4 Multiple Labels

For the final experiment, the networks had an additional binary input. When the binary input was set to zero, each architecture was trained on the original labeling of the rooms. When the binary input was flipped, each architecture was trained on a reordered labeling of the rooms. The function being learned alternated every 5,000 training steps for a total training time of 20,000 steps.

This modification attempts to induce catastrophic forgetting on multiple scales. The local sequential task of labeling each room is combined with a global sequential task of learning multiple labelings. Switching between global functions occurs on a much longer time scale than the local task of switching between rooms. Modifications to the task at large time-scales will flush circular training buffers of information concerning the original task. Only dynamic buffering, which preserves input data for all model vectors, will train on all the local and global tasks throughout the entire training period.

The networks trained on two separate room labeling arrangements were tested for 1,000 steps on the first labeling and 1,000 steps on the second labeling. The average network errors over the entire testing period of 2,000 steps were then used to generate the confidence intervals shown in Figure 12.

The ARAVQ balancing governor outperforms all other governor architectures. The circular buffer method fails to retain information over long time scales. By changing the labeling every 5,000 steps, circular training buffers are cleared of data from the previous labeling. This allows the network to forget the previous labeling. The balancing governor, with its dynamic buffer, performs much better. The network trains on both labelings throughout the entire training period. Figures 13 and 14 show typical error plots over the 2,000 step testing period for the circular buffer based ARAVQ governor and the ARAVQ balancing governor respectively.

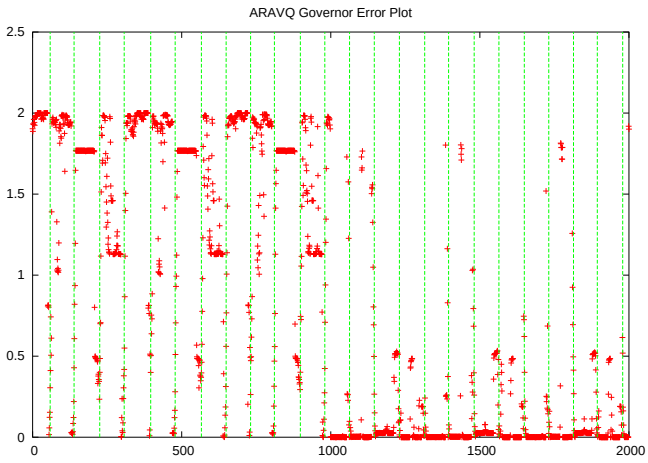

Figure 13: The network only learns the second labeling. This indicates that the circular buffer did not retain data concerning the first labeling during training on the second labeling.

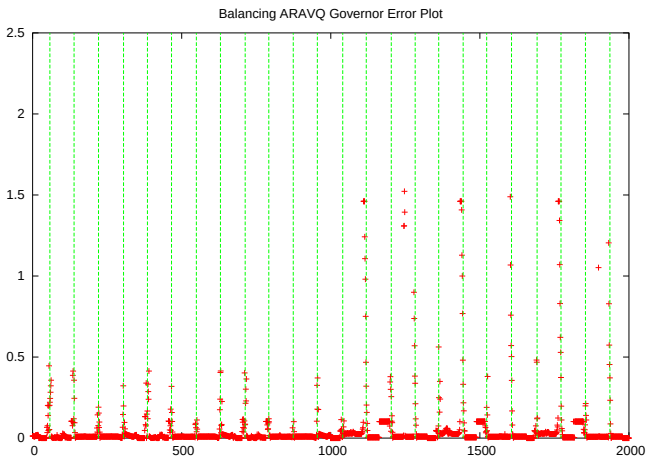

Figure 14: With dynamic buffering the network learns both labelings. The network trains on data from both labelings throughout the training period.

5. Conclusions

The governor assisted neural networks performed significantly better than the ungoverned neural networks. Training based on event abstraction eliminated the sequences of repetitive inputs which caused the occurrence of catastrophic forgetting in the ungoverned networks. Autonomously balancing the data using vector quantization techniques may be applicable more generally as well, eliminating the tedious process of building a balanced data set in many problem domains where neural networks are used.

The color aliasing task demonstrated that certain features in the environment could be vitally important to producing the correct output and yet fail to be isolated and recognized by the RAVQ or ARAVQ algorithms as deserving of event status. The RAVQ and ARAVQ algorithms sample sparsely, making fine correlations, required to learn the color aliasing location task, difficult to amplify in the training set. These subtle correlations may have an impact on a network's sensor fusion ability. Careful attention to the vector quantization parameters may help alleviate some of these difficulties.

The superior performance of the ARAVQ balancing governor in the multiple labeling task is certainly a result of the robust dynamic buffer. The training buffer of the ARAVQ balancing governor contains vectors associated with all model vectors, and so the network continually trains on both functions, whereas a fixed training buffer would eventually lose all vectors associated with the first function once the architecture began training on the second. This would produce exactly the results described above.

The ability to train feed-forward networks effectively with governed back-propagation in robot control problems allows for developmental implementations that otherwise would have had to rely on different learning algorithms. The autonomous nature of event extraction using the RAVQ and ARAVQ algorithms may play other keys roles in developmental systems.

6. Future Work

The governor architecture can be improved upon in many ways. In the ARAVQ balancing governor, the history buffers for previously learned tasks do not change and continued training on these inputs may lead to a reduction in generalization. Populating 'stale' model vector buffers with pseudo-patterns (Robins, 1998) may maintain network generalization during rehearsal of old tasks.

Besides improving the effectiveness of the governor architecture, new methods should be developed to prevent catastrophic forgetting over a larger set of network topologies. The governor architecture does not support recurrence. Excising long sequences of repetitive

data distorts the temporal context of event changes. Sequences of event changes can be predicted but durations of events are removed from consideration by the methods considered. Predicting sequences of event changes requires vector quantization as a permanent feature of the architecture and not merely a training device.

Using event extraction as a permanent part of the architecture may lead to an expanded hierarchical developmental system.

7. Acknowledgements

We would like to thank Matt Fieldler who assisted in the original development of the RAVQ based governor, Evan Moses who continues to work on new applications for governed neural network learning, and Andrew Stout and Yee Lin Tan whose work in localization elucidated the problem of catastrophic forgetting.

References

Blank, D., Meeden, L., and Kumar, D. (2002). Bringing up robot: Fundamental mechanisms for creating a self-motivating, self-organizing architecture. In *Simulation of Adaptive Behavior*.

Blank, D., Meeden, L., and Kumar, D. (2003). Python robotics: An environment for exploring robotics beyond legos. In *ACM Special Interest Group: Computer Science Education Conference (SIGCSE)*.

Denny, M. (2002). Watt steam governor stability. *Institute of Physics Publishing European Journal of Physics*, 23:339–351.

French, R. (1999). Catastrophic forgetting in connectionist networks. *Trends in Cognitive Science*, 3(4):128–135.

French, R., Ans, B., and Rousset, S. (2001). Pseudopatterns and dual-network memory models: advantages and shortcomings. *Connectionist Models of Learning, Development and Evolution*, pages 13–22.

Gerkey, B., Vaughan, R., and Howard, A. (2003). The Player/Stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the 11th International Conference on Advanced Robotics*, pages 317–323, Coimbra, Portugal.

Kohonen, T. (2001). *Self-Organizing Maps*. Springer, third edition edition.

Krushke, J. (1992). ALCOVE: An exemplar-based model of category learning. *Psychological Review*, 99:22–44.

Linåker, F. (2003). *Unsupervised on-line data reduction for memorisation and learning in mobile robotics*. PhD thesis, University of Sheffield.

Linåker, F. and Niklasson, L. (2000a). Sensory flow segmentation using a resource allocating vector quantizer. In *Advances in Pattern Recognition: Joint IAPR International Workshops SSPR2000 and SPR2000*, pages 853–862. Springer.

Linåker, F. and Niklasson, L. (2000b). Time series segmentation using an adaptive resource allocating vector quantizing network based on change detection. In *Proceedings of the International Joint Conference on Neural Networks*, pages 323–328, Como, Italy. IEEE Press.

McClelland, J., McNaughton, B., and O'Reilly, R. (1995). Why there are complimentary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological Review*, 102:419–457.

McCloskey, M. and Cohen, N. (1989). Catastrophic interference in connectionist networks: the sequential learning problem. *The Psychology of Learning and Motivation*, 24:109–165.

Robins, A. (1998). Catastrophic forgetting and pseudorehearsal in neural networks. Connectionists email list. http://www.cs.otago.ac.nz/nnweb/pseudo.html.

Tarassenko, L. (1998). *A Guide to Neural Computing Applications*. John Wiley & Sons Inc., New York, New York.