Context-Driven Predictions

Marc G. Bellemare and **Doina Precup**
McGill University
School of Computer Science
{marcgb,dprecup}@cs.mcgill.ca

Keywords: Prediction learning, associative memories, context-based model

Abstract

Markov models have been a keystone in Artificial Intelligence for many decades. However, they remain unsatisfactory when the environment modelled is partially observable. There are pathological examples where no history of fixed length is sufficient for accurate prediction or decision making. On the other hand, working with a hidden state (like in Hidden Markov Models or Partially Observable Markov Decision Processes) has a high computational cost. In order to circumvent this problem, we suggest the use of a context-based model. Our approach replaces strict transition probabilities by influences on transitions. The method proposed provides a trade-off between a fully and partially observable model. We also discuss the capacity of our framework to model hierarchical knowledge and abstraction. Simple examples are given in order to show the advantages of the algorithm.

1 Introduction

The ability to predict future events is a necessary component of intelligent agents, as it facilitates accurate planning. A standard approach is to predict the future solely through observations, for example using a fixed-order model Markov Chain. Unfortunately, a history of any fixed length might not be sufficient to accurately predict the next observation. Methods using a variable history window (e.g. McCallum, 1995) work well in practice but are largely heuristic. A different approach for making predictions through time is to introduce a notion of latent or hidden state of the environment, as in Hidden Markov Models [Rabiner, 1989] and Partially Observable Markov Decision Processes [Kaelbling *et al.*, 1998]. Such models clearly separate observations from hidden states, and keep track of the current state through a belief vector. However, assuming a hidden state requires knowledge about it, and often the state transition probabilities are assumed to be known. For learning agents, this approach appears restrictive: knowledge is necessarily bounded by the state representation. Furthermore, the true state of the system may involve many elements which are not observable by the agent for long periods of time. One may imagine, for example, an environment divided into regions where certain observations only appear in certain regions. Objects which evolve in the world also involve keeping track of additional state information about them. Recent work on predictive representations [Sutton and Tanner, 2005; Littman *et al.*, 2002] aims to bridge this gap by explicitly constructing a state, which is a sufficient statistic for predicting the future, from the experience of the agent.

The goal of our work is also to build a predictive representation which is not based on a fixed length of history and can be learned incrementally from data. Specifically, our work is geared towards environments where observations are *subjective*. In such systems, we can hope that the only state the agent needs to be concerned about is based on what is directly accessible to it. More formally, this would be observations that lead to better predictions of the immediate future. Our work is based on hetero-associative memories, such as Sparse Distributed Memories [Kanerva, 1993], but puts the emphasis on predicting future events rather than as a way of palliating to noisy inputs. Our model can also handle new observations and can generalize to a certain extent, as there is no necessity for model specification in advance beyond defining the observation features. In this paper we do not explicitly discuss planning and rewards, but assume that predictions can be used to determine actions.

Similar problems have been explored in the connectionist literature. For example, [Elman, 1990] considers the notion of context in a neural network as being closely related to the previous state of the system. Through simple tasks, he shows that a recurrent neural network can learn to base its predictions on past inputs summarized in its hidden units' activation. Research on a symbol prediction task close to ours has been described in [Bose *et al.*, 2005]. However, they take a neural network approach to the problem and focus on a particular kind of network architecture and implementation. Recently, Boltzmann machines have been used for prediction through time as well [Taylor *et al.*, 2007]. Our proposed algorithm is significantly different from those mentioned above: we focus on weighting past observations in a direct fashion rather than through an internal state of the agent. A similar prediction task is addressed in [Gopalratnam and Cook, 2007], in the context of text parsing. However, in their approach all data structures are geared towards dealing with

symbols. Our approach works both for discrete and continuous observations (although our experiments contain discrete observations only).

The paper is organized as follows. In Section 2 we briefly review Sparse Distributed Memories, on which we based our ideas. We also discuss Hopfield networks and Boltzmann Machines and the benefits of Sparse Distributed Memories over them. In Section 3, we formally define our framework. Following this, we discuss a simple algorithm for learning from observations in Section 4. We then give examples showing the predictive power of the algorithm in Section 5. Finally, Section 6 discusses the sort of environments for which our method is suitable, its current failings compared to existing models and how it may be used to obtain abstraction.

2 Background

Sparse Distributed Memories (SDMs) were developed in order to model long-term memory [Kanerva, 1993]. Of interest is the capacity of such memories to both retrieve a noise-free version of an input vector (auto-association) and to retrieve an associated output vector (hetero-association). As opposed to other types of associative memories, SDMs do not require an iterative process in order to converge to the desired answer. In that respect, they resemble feed-forward neural networks, and have been modelled as such. SDMs have been used in prediction, for example for robot navigation [Rao and Fuentes, 1996].

A SDM is divided into two parts: the *hard locations* and the *output words*. A hard location is simply a binary vector of the same length as the input vector. To each hard location corresponds an output word which is composed of an integer-valued vector, possibly of different length.

When a vector is given as input to the memory, its distance d_i to each hard location i is computed; in the original implementation, d_i is simply a Hamming distance. The output of the system is then obtained in the following way: for each hard location with distance $d_i \leq \delta$, where δ is a threshold value, its corresponding word is added to the output sum $\mathbf{s}$. The actual output $\mathbf{o}$ is found by thresholding the sum such that $o_i = 1$ if $s_i > 0$, and $o_i = 0$ otherwise.

Learning in SDMs is fairly straightforward. When we wish to map an input $\mathbf{v}$ to a target word $\mathbf{z}$, we determine which hard locations are activated ($d_i \leq \delta$) by $\mathbf{v}$. For each of these, we add $\mathbf{z}$ to their corresponding output word.

If the hard locations are orthogonal, we obtain something very close to a linear approximator. Such systems have been studied extensively, for example as associative search networks [Barto *et al.*, 1981]. If the hard locations are not orthogonal, however, the memory potentially gains in generalization capacity by producing more complex combinations of mappings. Various works on SDMs have experimented with extending the system to handle real values, computing its capacity, and deciding how to craft the hard locations to better suit a given task. Presented as above, SDMs are interesting as they propose an intuitive coverage of the input space through actual potential inputs (work on this idea in reinforcement learning was done in [Ratitch and Precup, 2004]). Their hetero-associative capacity also makes them promising for prediction purposes.

Other associative memory frameworks have been proposed before SDMs. Among those still used today, we find Hopfield networks [Hopfield, 1982] and Boltzmann Machines [Fahlman *et al.*, 1983]. Both rely on the idea of neuron-like elements that share undirected connections with their neighbors. In the simplest version, each element has two states. With each undirected connection is associated a weight, which influences the corresponding neighbor into being in a certain state, based on the weight sign. Usually, a positive weight indicates that both units should be in the same state, whereas a negative weight indicates they should take different states. The system then attempts to minimize its *energy*. This energy increases when units of the system take states that violate the influence of their connection weight.

Inputs can be given to these algorithms by forcing certain elements to stay in a given state, for example by explicitly defining input units with fixed values. Since both algorithms are undirected by nature, obtaining a minimum energy state (a solution) requires iterating over the unit states until the energy stagnates. Unfortunately, this process can be slow. Learning in such a model is also computationally expensive, although Restricted Boltzmann Machines have recently been used to achieve good results [Hinton *et al.*, 2006].

3 Framework

Unfortunately, SDMs suffer from a big disadvantage. They are, by nature, deterministically noise-correcting and do not allow us to predict events that *may* occur. Before we discuss our proposed framework, we must make a few assumptions regarding the environment that we are interested in modelling and define a few terms. First, let us define a *percept* as a real-valued vector representing an observation. Secondly, we assume that association is not done on a one-to-one basis. Rather, a given percept may be associated to many other percepts, with different strengths of association for each. We formalize this by proposing that an input percept $\mathbf{p}$ maps to a *distribution* over associated percepts.

We define the memory as a set of cells, C. Each of these cells acts as a hard location, and therefore has a single percept associated with it, which we denote C_i through notational abuse. Each cell also has a *saliency* value s_i associated with it and a vector of output weights, W_i. W_i represents directed associations between cells, and so W_i has the same size as C. In general, we denote the weight matrix W and the saliency vector corresponding to C, $\mathbf{s}$.

When the system is presented with an input percept $\mathbf{p}$, it computes an activation vector α similar to the activation in SDMs. Here however, α is a real-valued vector with elements taking values between 0 and 1, where $\alpha_i = 1$ indicates a perfect match between $\mathbf{p}$ and C_i and $\alpha_i = 0$ indicates no match. Usually, we would like $\sum_i \alpha_i = 1$. In our work, we use the simplest activation function, namely $\alpha_i = 1$ if $C_i = \mathbf{p}$, and $\alpha_i = 0$ otherwise. In effect, we are assuming that percepts are actually symbolic and not subject to noise; this need not be the case in general.

By themselves, SDMs do not allow any association based on past observations. To circumvent this, we use the saliency

value of cells as an activation trace. More formally, at every time step we set the saliency vector to

$$\mathbf{s} \leftarrow \gamma \mathbf{s} + \alpha$$

This is similar to the cumulative eligibility trace in reinforcement learning [Sutton, 1988], where γ is a decay factor. A restricted form of this type of encoding has also been used for prediction in [Bose *et al.*, 2005; Furber *et al.*, 2004]. If $\alpha_i = 1$ for exactly one percept and 0 everywhere else, $\mathbf{s}$ represents the past sequence of observations, provided no percept has been observed twice. Note that for the purposes of the system, $\mathbf{s}$ does not need to be an exponentially decaying trace of observations; its goal is to serve as *context* to obtain better predictions. In Section 6 we will discuss one possible way of improving the above equation.

Our algorithm diverges from SDMs as it attempts to predict percepts that match its hard locations, rather than separating them from the output words. We define secondary activation, denoted by β, as a vector representing a prediction weight for each cell. We are interested in predicting which of the percepts (represented by cells) may be observed. We assume here that the hard locations represent actual percepts. We then compute β as

$$\beta = W\mathbf{s}$$

From this equation one can notice that the weight matrix indeed acts as a set of associations; experiencing a percept leads to related percepts being predicted.

At the very least the values of β should be in the correct prediction order and significantly different. Any function of β which preserves this ordering and results in a valid probability distribution may be used to predict the next time step. We chose to use a simple Boltzmann distribution using β and given by:

$$P(C_i|\mathbf{s}) = \frac{e^{\tau\beta_i}}{\sum_i e^{\tau\beta_i}}$$

The distribution's entropy is controlled by τ, a standard temperature parameter between 0 and ∞. In the experiments we used $\tau = 1$ but other values may yield better results.

4 Learning

After having discussed how the algorithm predicts events, we now describe how learning can be accomplished. Logically, since we use the saliency vector $\mathbf{s}$ as contextual hints which allow us to make a better prediction, we should modify weights based on $\mathbf{s}$.

For now, assume that we already have a known set of hard locations. Let p^t be the percept observed at time t, and similarly C^t, the set of cells, W^t the weight matrix and $\mathbf{s}^t$ the saliency vector. We denote the activation due to p^t by $\alpha(p^t)$.

Assuming that we want to predict p^t in the future when $\mathbf{s}$ is present, we should modify W^t to produce a probability distribution similar to $\alpha(p^t)$. Formally, let π be our probability distribution on C; we define the prediction error E as:

$$E = \frac{1}{2}\sum_i (\pi_i - \alpha_i)^2$$

where α_i is the i^{th} component of $\alpha(p^t)$. We then compute the gradient of the error with respect to $W_{i,j}$, the j^{th} weight of cell i. Here $W_{i,j}$ represents how strongly j influences the prediction of i.

1. Initialize hard locations
2. $W \leftarrow 0$
3. For each episode do
4. $\quad s \leftarrow 0$
5. $\quad$ Repeat until done
6. $\quad\quad$ Compute π from s
7. $\quad\quad$ Observe the next percept p
8. $\quad\quad$ Update W based on $\pi - \alpha(p)$
9. $\quad\quad$ $s \leftarrow \gamma s + \alpha(p)$

Table 1: Our context-based prediction algorithm.

Let $\sigma = \sum_i e^{\tau\beta_i}$, and recall that $\beta_k = \sum_i W_{k,i}s_i$. First note that:

$$\begin{aligned}\frac{\partial}{\partial W_{i,j}}\pi_k &= \frac{\partial}{\partial W_{i,j}}\frac{e^{\tau\beta_k}}{\sigma} = \frac{\tau e^{\tau\beta_k}}{\sigma^2}\left(\sigma\frac{\partial}{\partial W_{i,j}}\beta_k - e^{\tau\beta_i}s_j\right) \\ &= \begin{cases} \tau s_j \pi_k(1-\pi_k) & \text{if } k = i \\ -\tau s_j \pi_k \pi_i & \text{otherwise} \end{cases}\end{aligned}$$

Let ϵ be the vector of errors, with $\epsilon_i = \pi_i - \alpha_i$. From the above equation we obtain:

$$\begin{aligned}\frac{\partial}{\partial W_{i,j}}E &= \sum_k(\pi_k - \alpha_k)\frac{\partial}{\partial W_{i,j}}\pi_k \\ &= \tau s_j(\pi_i(1-\pi_i)(\pi_i - \alpha_i) - \sum_{k\neq i}\pi_i\pi_k(\pi_k - \alpha_k)) \\ &= \tau s_j \pi_i((1-\pi_i)\epsilon_i - \sum_{k\neq i}\pi_k\epsilon_k) \\ &= \tau s_j \pi_i(\epsilon_i - \pi\cdot\epsilon)\end{aligned}$$

We can then modify the output weights through a standard update rule with learning rate $c \in (0,1)$:

$$W_{i,j} \leftarrow W_{i,j} - c\frac{\partial}{\partial W_{i,j}}E$$

Usually, probability-producing systems are trained using a likelihood-based gradient. Here, however, there are two reasons why we might prefer to use the sum of squared errors to compute the gradient. First, it explicitly allows us to train the system to output a combination of hard locations, through the α vector. This can be interesting if many percepts activate two or three hard locations due to noise. Also, we are interested in good generalization capabilities. Experiments in which we used maximum likelihood gave worse results. We believe that is might due to the fact that there is no 'ground truth' distribution which we are approximating; instead, we are constructing an appropriate distribution through association.

There is a second learning problem, which we ignore here, but is of interest. The hard locations do not have to be predefined, or fixed. In a way, learning to *recognize* a percept can be just as hard as prediction. We discuss this issue further in Section 6 below. The whole algorithm is presented in Table 1.

Episodes		40	80	400	800
P(1)=0.5	P(1)	0.41	0.43	0.45	0.45
	P(2)	0.46	0.48	0.51	0.51
P(1)=0.75	P(1)	0.69	0.72	0.73	0.73
	P(2)	0.20	0.20	0.23	0.24
P(1)=0.875	P(1)	0.79	0.84	0.86	0.86
	P(2)	0.12	0.10	0.10	0.11

Table 2: Predicted observation frequencies based on the number of training episodes.

5 Examples

In this section we give examples in order to show that our proposed algorithm can indeed perform prediction in a similar fashion to that of a strict Markovian Model, without being restricted by a fixed history length or states. For all of the examples, we use sequences of numbers as observations. Each number is encoded as a n-sized vector with the corresponding bit set to 1 and all others set to 0; n is the maximum number of observations. Note that in a more natural task a percept vector may represent sensory input, and so its structure would be well-defined. To simplify matters, we assume that the agent always receives the percept 0 (first bit set to 1) at the beginning of each episode, and never receives it during the episode. This is similar to defining a special start symbol when learning a k^{th} order Markov Model, and we would expect this to have little impact in a system that learns continuously. Here we are using a learning rate of 0.5, which gave sufficiently stable results. The decay factor γ was also arbitrarily set to 0.5.

The first experiment is the simplest one, and shows that the system can approximate symbol frequencies. We simply produce two separate one-observation episodes with certain frequencies. One of these episodes produces a 1, while the other produces a 2. In order to avoid variance in the results, we chose to use a fixed sequence of episodes. These always start with 1, end in 2 and episodes containing 2's are experienced at regular intervals in-between. Estimated frequencies are given in Table 2, where we show the probability of each event after a certain number of training episodes. Actual frequencies are given on the left-hand side. Note that the given probabilities do not sum to one due to the Boltzmann distribution: it assigns a non-zero probability to all events, and here 0 is predicted as unlikely but not impossible. We can see that with sufficient training, all estimates converge towards their true values. The learning rate here prevents us from obtaining the exact probabilities as the output weights oscillate between episodes.

The second experiment is aimed at showing that the system, despite having no explicit indication of ordering, can in fact discriminate based on recency. The example strings that we use are respectively 123 and 214. The goal here is to show that we can predict with high probability which of the two end symbols will appear based on ordering. Results are given in Table 3.

Clearly as the number of training iterations increases the system becomes able to more precisely predict the true symbol based on context. Again here, the symbols 0, 1 and 2 are also predicted with probabilities that decrease as training increases. If we were only interested in predicting the end symbol, we could increase τ to obtain higher probabilities. With our choice of parameters, the initial symbol (1 or 2) is predicted to occur with roughly 50% chance, as in the first experiment.

Episodes		50	100	200	500
Context 1,2	P(3|12)	0.46	0.61	0.74	0.85
	P(4|12)	0.24	0.19	0.13	0.075
Context 2,1	P(3|21)	0.21	0.17	0.12	0.07
	P(4|21)	0.50	0.64	0.75	0.855

Table 3: Predicted observation based on order of past observations.

Episodes	100	200	500	1000	2000
Sequence pairs					
618	0.77	0.86	0.92	0.95	0.96
719	0.78	0.86	0.92	0.95	0.96
6128	0.59	0.74	0.86	0.90	0.935
7129	0.62	0.75	0.86	0.905	0.935
61238	0.44	0.55	0.71	0.80	0.87
71239	0.50	0.60	0.74	0.82	0.88
612348	*0.40*	*0.44*	0.53	0.63	0.73
712349	0.46	0.51	0.60	0.68	0.75
6123458	*0.39*	*0.42*	*0.45*	0.49	0.55
7123459	0.45	0.48	0.525	0.57	0.62

Table 4: Predicted observation based on long-term context. Values in italic show when the actual observation was not predicted with the highest probability.

The third set of sequences that we looked at shows that the system can learn to use context from any point in the past to differentiate two predictions. More specifically, we have two target symbols (8 and 9) which can only be predicted based on the first symbol that occurs, which is either a 6 or a 7. The training strings and the predicted probability of the correct symbol are reported in Table 4.

As can be seen from this table, the system follows a slow degradation in predicting the correct event as the differentiating observation becomes more remote. The fact that 9 is systematically predicted with higher probability than 8 is an artefact of our experiment, due to the sequence containing 9 being presented after the sequence containing 8. It is interesting to note that for the longest example given here, the saliency of the context percept at the time of prediction is 2^{-5}. Yet it can be seen that as the number of iterations increases, the algorithm learns to correctly distinguish between the two events.

6 Discussion

The framework presented seems to provide us with a way of predicting events which, if not perfectly accurate, is not subject to history length constraints and does not require explicit state knowledge. Of chief interest is the fact that the algorithm, as shown above, can handle temporal ordering, which

can be key to many predictions. However, it can also handle predicting observations from unordered sequences. As a brief example, we can imagine an environment where we obtain clues about the 'state' of the world, which is represented by a separate percept (possibly one that occurs often while in that 'state'). Markov Models relying on a strict order of observations will need many samples to produce clear predictions. Our algorithm, on the other hand, can infer from many weak clues a more general percept.

Of a technical nature, both the saliency vector and the weight updating scheme are flexible and may be modified to fit different situations. For example, we experimented with modifying the saliency vector to keep relevant percepts in context and remove more quickly irrelevant ones. This can be easily done by considering the gradient of the error on the current prediction with respect to the saliency of each cell; the update rule is then very similar to the one used for the output weights. The weight updating scheme may also be improved by preventing negative gradients. Currently, our weight update rule reduces the probability of all the events that did not occur ($\alpha_i = 0$) at the same time as it increases the probability of the actual observation. This might hinder later learning; a purely additive approach, closer to a frequency-based model, might perform better.

Obviously, being able to predict events more or less accurately is not sufficient for an agent; we also need correct control. From a reinforcement learning point of view, we can naturally incorporate reward into our framework in three specific ways. First, we can explicitly attempt to assign a value to each observation, and compute the predicted expected value based on context. A similar idea was developed in [Ratitch and Precup, 2004], but their algorithm did not explicitly model observation prediction, and was done in a fully observable framework. We can also modify the weight update rule to use the magnitude of rewards to focus learning on important parts of the system. More interestingly, though, we can modify the saliency vector based on reward information, so that a percept which is known to be associated with high reward or penalty might be kept in context longer.

In the case of a truly stochastic event, similar to what we presented in the first experiment of Section 5, our prediction will, by definition, never be accurate. This is in some sense a drawback of the algorithm: transitions are expected be fully determined by some past event. There are many ways to address this problem. First, we can assume that no event is truly stochastic in nature, and that stochasticity only appears through lack of contextual information. In such a case, we could try inferring *missing* past events by increasing the saliency of percepts which usually cause the current observation. Another approach would be to explicitly consider the variance of a prediction. Qualitatively, large output weights suggest strong evidence towards a prediction. If more than one event is strongly activated by context, then there is reason to believe that stochasticity will appear in the observations.

One question that has been largely ignored so far in this paper is that of handling percepts which occur more than once in a short interval of time. Since the algorithm has no way of specifying that an observation has occurred twice, we lose the capacity of a k^{th} order model to make a separate prediction when this happens. This may be seen as a failure of the framework, and indeed it is not suited for environments where there are very few observations. However, if we consider that the algorithm builds a causality link from a percept to another, then the problem may be solved in the following way. The repeated occurrence of a percept does not provide additional information; it instead becomes a question of the duration of the observation. Our framework implicitly handles duration if we do not allow non-zero output weights from a cell to itself. Indeed, the output weights to the context should become larger in order to accurately predict an event which occurs for a longer period of time, and therefore such events should be predicted even when the relevant contextual information is further in the past.

We purposefully left out the recognition problem in our presentation of the algorithm. Constructing suitable hard locations might not be simple. However, our framework implicitly proposes a different approach to recognition. A set of temporally or spatially related percepts may all be associated together to form a single, more abstract object. This can be the case in Computer Vision for example, where multiple poses are mapped to be same object.

In a similar fashion, the algorithm is not restricted to a single modality. We have extended it to include multiple modalities which are associated together. Although a separate prediction is made for each modality, context is constituted of all modalities. Such a capacity opens many opportunities, such as building knowledge through associations across modalities. Among the most striking is the possibility to link perceptual knowledge to actions if both are represented in the associative framework proposed.

The discussion above suggests the use of actual percepts as abstraction, ie. strong contextual clues. An abstract percept may be one that activates its instances, for example the idea of a tree may simply be a cartoon tree, which then maps to all other kinds of trees. What is interesting here is that considering an abstract object as a regular percept allows us to manipulate it atomically, without regard for its abstract nature. Having atomic abstractions in turn allows us to abstract them, and gives us the capacity to build hierarchical knowledge without any additional apparatus.

Past research on association and prediction has mainly focused on obtaining averages of noisy observations. The novelty in our approach comes from the fact that we put the emphasis on the importance of associations between incomplete observations. As such, our framework takes a radically different view of perceptions: noise can be overcome through the association of noisy signals. All the associative memories discussed above share the fault that they are geared towards producing an *ideal* output. Rather, we hope that our framework can build many (possibly noisy) associations which, when combined, yield a correct answer. This is something that can only be done if we can produce a probability distribution over associations; we also need to be able to use events from any time in the past. Our algorithm, by achieving both of these, seems promising for predictive and knowledge-building purposes.

7 Conclusion

In this paper we presented a novel framework which uses past observations as contextual clues for predicting future events. At the core of our algorithm lies the concept of saliency, which causes past, important events to be used as context. We described both how to produce predictions using associative links between observations and how to learn such associations. Simple examples show that the algorithm preserves basic properties of Markov Models, such as the capacity to distinguish between differently ordered observations. We also exemplified the potential of the algorithm to use contextual information from anywhere in the past towards current predictions, which is the main failing of history-based models. However, our algorithm does not require knowledge of the true state of the world in order to make predictions, therefore reducing its complexity and making it more applicable to general tasks. Although much remains to be done in order to discover the advantages and flaws of the algorithm, we have sketched out it potential uses. We propose a slightly different view of observations, abstraction and contextual information, which we hope might lead to improved performance of algorithms.

We are currently in the process of testing the algorithm on larger domains which may help understand its strengths and weaknesses. Although it is not suited for environments where very few observations are available and where they repeat often, natural environments provide a large variety of perceptions. The goal in such tasks is not necessarily perfectly accurate prediction, but rather coherent actions. Our framework may help in this respect by using context to drive prediction and action, and by allowing for abstraction of knowledge and the construction of high-level actions through association.

Acknowledgments

This work was supported in part by funding from NSERC and CFI.

References

[Barto *et al.*, 1981] Andrew G. Barto, Richard S. Sutton, and Peter S. Brouwer. Associative search network: A reinforcement learning associative memory. *Biological Cybernetics*, 40(3):201–211, 1981.

[Bose *et al.*, 2005] Joy Bose, Steve B. Furber, and Jonathan L. Shapiro. An associative memory for the on-line recognition and prediction of temporal sequences. In *Proceedings of the International Joint Conference on Neural Networks*, 2005.

[Elman, 1990] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[Fahlman *et al.*, 1983] Scott E. Fahlman, Geoffrey E. Hinton, and Terrence J. Sejnowski. Massively parallel architectures for AI: NETL, Thistle, and Boltzmann machines. In *Proceedings of the National Conference on Artificial Intelligence*, 1983.

[Furber *et al.*, 2004] S. B. Furber, W.J. Bainbridge, J.M. Cumpstey, and S. Temple. Sparse distributed memory using n-of-m codes. *Neural Networks*, 10, 2004.

[Gopalratnam and Cook, 2007] K. Gopalratnam and D. J. Cook. Online sequential prediction via incremental parsing: The active LeZi algorithm. *IEEE Intelligent Systems*, 2007.

[Hinton *et al.*, 2006] Geoffrey E. Hinton, S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.

[Hopfield, 1982] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences USA*, 79(8):2554–2558, 1982.

[Kaelbling *et al.*, 1998] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

[Kanerva, 1993] Pentti Kanerva. Sparse distributed memory and related models. In M. Hassoum, editor, *Associative Neural Memories*, chapter 3. Oxford University Press, 1993.

[Littman *et al.*, 2002] Michael L. Littman, Richard S. Sutton, and Satinder Singh. Predictive representations of state. In *Advances in Neural Information Processing Systems 14*, pages 1555–1561, 2002.

[McCallum, 1995] Andrew K. McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, University of Rochester, 1995.

[Rabiner, 1989] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–287, 1989.

[Rao and Fuentes, 1996] Rajesh P.N. Rao and Olac Fuentes. Learning navigational behaviors using a predictive sparse distributed memory. In MIT Press, editor, *Fourth International Conference on Simulation of Adaptative Behavior*, 1996.

[Ratitch and Precup, 2004] Bohdana Ratitch and Doina Precup. Sparse distributed memories for on-line value-based reinforcement learning. In *Proceedings of the 15th European Conference on Machine Learning*, 2004.

[Sutton and Tanner, 2005] Richard S. Sutton and Brian Tanner. Temporal-difference networks. In *Advances in Neural Information Processing Systems 17*, pages 1377–1384, 2005.

[Sutton, 1988] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

[Sutton, 1995] Richard S. Sutton. TD models: Modeling the world at a mixture of time scales. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 531–539, 1995.

[Taylor *et al.*, 2007] G. Taylor, G. Hinton, and S. Roweis. Modelling human motion using binary latent variables. In *Advances in Neural Information Processing Systems 19*, 2007.