To appear in *Proceedings of the 12th International Conference on Cognitive Modeling*, 2013.

Learning via Gradient Descent in Sigma

Paul S. Rosenbloom[a,b] (rosenbloom@usc.edu), Abram Demski[a,b] (ademski@ict.usc.edu), Teawon Han[c] (teawon.han@gmail.com) & Volkan Ustun[b] (ustun@ict.usc.edu)
[a]Department of Computer Science and [b]Institute for Creative Technologies
12015 Waterfront Dr., Playa Vista, CA 90094 USA
[c]Dongil-highvill apt 309-1503, suji-gu, shinbong-dong, yongin-si, gyeonggi-do, South Korea

Abstract

Integrating a gradient-descent learning mechanism at the core of the graphical models upon which the Sigma cognitive architecture/system is built yields learning behaviors that span important forms of both procedural learning (e.g., action and reinforcement learning) and declarative learning (e.g., supervised and unsupervised concept formation), plus several additional forms of learning (e.g., distribution tracking and map learning) relevant to cognitive systems/modeling. The core result presented here is this breadth of cognitive learning behaviors that is producible in this uniform manner.

Keywords: Cognitive architecture, graphical models, learning, gradient descent.

One of the key hypotheses investigated during Soar's early years was that a simple learning mechanism – *chunking* – when integrated into an appropriate architecture could yield a *general learning mechanism* capable of acquiring the diversity of knowledge required by a cognitive system (Laird, Rosenbloom & Newell 1986). Although this proved to be a bridge too far – with Soar later to incorporate additional reinforcement, episodic and semantic learning mechanisms (Laird, 2012) – much was learned in exploring this hypothesis over the years (Rosenbloom, 2006).

Despite the limitations eventually evident in Soar, the drive towards general learning mechanisms in cognitive architectures/systems remains appealing. To the extent it is feasible, it yields deeper and more elegant theories of intelligent behavior with broader scientific reach (Deutsch, 2012). Here we report results from a somewhat more modest such effort that, like Soar, is based on integrating a simple learning mechanism into an appropriate architecture; however, the particular mechanism here – a local, online variant of *gradient descent* – is quite different from chunking, and is integrated into a *hybrid mixed architecture* called *Sigma* (Σ). The results are interestingly different from those obtained with chunking in Soar.

Sigma is a nascent cognitive system – an integrated computational model of natural and/or artificial intelligent behavior – that is based on a novel cognitive architecture. Its development is driven by three general desiderata: *grand unification* (uniting the requisite cognitive and non-cognitive aspects of embodied intelligent behavior); *functional elegance* (exhibiting a broad set of capabilities while remaining fundamentally simple and theoretically elegant); and *sufficient efficiency* (behaving rapidly enough for anticipated applications). The results presented here predominantly concern functional elegance.

Past work on Sigma, although mostly under the more generic name of a *graphical cognitive architecture*, has spanned memory, problem solving, decisions, mental imagery, perception, localization, and natural language. Learning is a more recent focus, with the first published work covering reinforcement learning (RL) (Rosenbloom, 2012). In compliance with functional elegance, the core results demonstrated RL in a 1D grid task without requiring the addition to the architecture of an explicit RL mechanism. The results instead arose from gradient-descent learning, as investigated further here, plus Sigma's overall generality in representation and processing.

Even at this relatively early stage in the development of learning in Sigma, the RL work showed the acquisition, via a single simple learning mechanism, of reward functions, discounted future utilities, Q values and action models (although to learn action models an additional architectural change was needed to enable simultaneous representation of successive states in working memory). RL and action modeling are both classical forms of procedural learning, but the work also demonstrated forms of distribution tracking, of rewards and utilities. Further investigation has shown that the same mechanism can also yield forms of declarative learning, including supervised and unsupervised concept formation, over both standard datasets and natural-language subtasks such as word sense disambiguation (WSD) and part of speech (POS) tagging. It can also yield other important learning behaviors, such as the intertwining of map learning with localization that is at the heart of simultaneous localization and mapping (SLAM).

The primary claim in this article is not of a general learning mechanism in the sense originally sought with Soar, but that much can be learned about what is possible when a simple learning mechanism – in this case one based on local, online, gradient descent – is integrated deeply into an appropriate architecture (Sigma). The focus is not on the details of the learning mechanism itself, nor on its accuracy – so we have largely eschewed traditional machine learning evaluations in this article – but on the breadth of learning this combination enables in a functionally elegant manner.

After introducing Sigma and its gradient-descent learning mechanism, results will be presented for distribution tracking, including parts of what will ultimately form semantic (declarative) learning, plus action (procedural) learning and map learning. Concept formation (declarative learning) will then be explored in more detail, and finally earlier results on RL (procedural learning) will briefly be

reprised. Although not new, the RL (and action learning) results provide key pieces of the overall breadth of learning.

Sigma

The Sigma cognitive system is constructed in layers, with the two most critical being: (1) a cognitive architecture that provides a language of *predicates* and *conditionals*; and (2) a graphical architecture beneath it that is based on *graphical models* (Koller & Friedman, 2009). A predicate in the cognitive architecture is defined via a name and a set of typed arguments – as in `Location(state:state x:location)`, where the `Location` predicate has a `state` argument whose type is `state` and an `x` argument whose type is `location` – with *working memory* (WM) containing predicate instantiations that embody the state of the system. *Long-term memory* (LTM) is comprised of conditionals, each of which is defined via a set of predicate patterns and an optional function over pattern variables.

Conditionals provide a deep combination of rule systems and probabilistic networks. *Conditions* and *actions* are patterns that behave like the respective parts of rules, providing the forward momentum characteristic of procedural memory. *Condacts* are patterns that support the bidirectional processing that is key to probabilistic reasoning, partial matching, constraint satisfaction and signal processing. *Functions* specify relationships over conditional variables that may be hybrid (discrete and/or continuous) and mixed (probabilistic and/or symbolic).

Figure 1, for example, defines a probabilistic transition function for how a simulated robot's actions affect its location in a 1D grid. It comprises: a condition over `Location` (as defined above) that matches to the distribution in WM over the current location via variables (italicized) for the state and the current location; a condition over `Selected` that matches to the selected operator; a condact over the next location; and a uniform (pre-learning) function over the 8 possible next locations (*nx*) that defines their probabilities given the current location (*x*) and operator (*o*). The stars (*) denote that the constant .125 applies to the entire domain spanned by the three variables.

```
CONDITIONAL Transition
  Conditions: Location(state:s x:x)
              Selected(state:s operator:o)
  Condacts: Location*Next(state:s x:nx)
     Function(x,o,nx): .125<* * *>
```

Figure 1: Conditional for a 1D transition function.

Sigma's core *cognitive cycle* consists of accessing LTM until quiescence and then: (1) *deciding* on changes to WM; and (2) *learning* changes to LTM. Learning at present is limited to modifying the functions in conditionals, with no mechanisms yet incorporated for structure learning. Episodic learning extends temporally organized functions based on changes to WM. Gradient-descent learning improves the alignment of functions to the system's experience. The focus in this article is on gradient descent.

Beneath the cognitive architecture is the graphical architecture, which is built in particular from *factor graphs* – undirected bipartite graphs of factor and variable nodes – and the *summary product* message-passing algorithm (Kschischang, Frey & Loeliger, 2001). At this layer the cognitive cycle maps to a *graph-solution phase* that supports LTM access, followed by a *graph-modification phase* that yields decisions and learning. Predicates and conditionals in the cognitive architecture compile down into factor (square) and variable (circle) nodes, plus links, at this layer (Figure 2). Conditional compilation is based on a generalization of the Rete algorithm (Forgy, 1982) that can handle conditions, actions and condacts – with unidirectional message passing implicated for the former two and bidirectional message passing for the latter – plus additional nodes for functions. The alpha (discrimination) and beta (join) networks in Sigma are generalizations of those in Rete, with the gamma network added for functions.

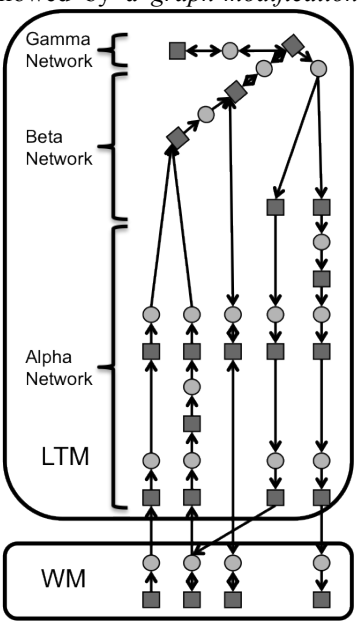

Figure 2: Factor graph for conditional with (left to right) 2 conditions, 1 condact, and 2 actions; plus a function (top).

The contents of the WM and function nodes, plus all of the messages passed around in the graph, are themselves generalizations of Rete's symbolic tokens. In particular, they are *piecewise-linear functions* – rectilinear arrays of multidimensional regions each of which has its own linear function (Figure 3) – that can approximate arbitrary continuous functions and be restricted appropriately to yield both discrete and symbolic functions (Rosenbloom, 2011). Rete's standard notions of constant tests and variables have also been generalized in Sigma. *Filters* are piecewise-linear generalizations of constant tests. *Affines* enable transforms over variables.

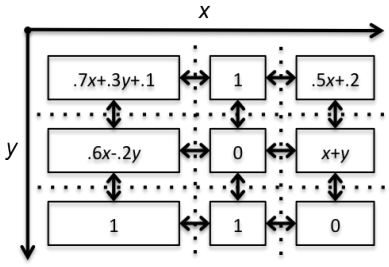

Figure 3: Bivariate function as a 2D array of linear regions.

Sigma's gradient-descent mechanism is implemented at this graphical layer. Based on ideas in (Russell, Binder, Koller & Kanazawa, 1995) for Bayesian networks, a message into a factor node for a conditional

function can be seen as providing feedback to that node from the rest of the graph that induces a local gradient for learning. Although Sigma uses undirected rather than directed graphs, the directionality found in Bayesian networks can be found at factor nodes when one of the variables is distinguished as the *child* – indicated by the underscoring of *nx* in Figure 1 – that is conditionally dependent on the other *parent* variables. We have also modified the original batch algorithm to learn incrementally (online) from each message as it arrives.

Consider the abstracted factor graph in Figure 4, which shows just the key factor nodes for a (naïve Bayes) concept-based semantic memory comprised of six predicates – for the concept and the five features (two of which are Boolean, and one each that is symbolic, discrete and continuous) – plus one conditional for the prior distribution on the concept and five conditionals for the conditional probabilities of the features given the concept (Rosenbloom, 2010). Each predicate yields a WM node (light) and each conditional yields a function node (dark). These are connected via join nodes (medium) in the beta network.

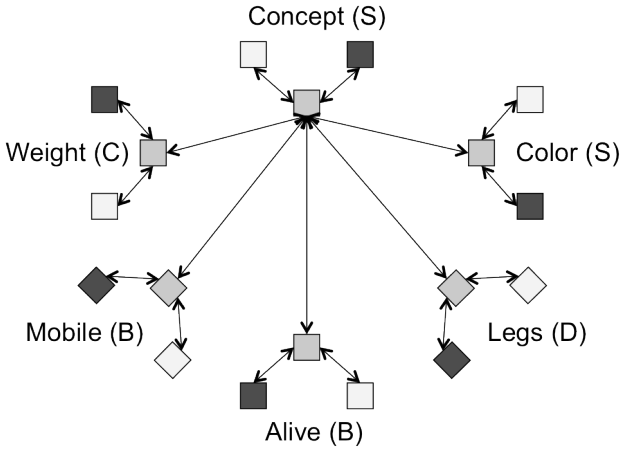

Figure 4: Abstraction of semantic memory graph.

Memory access occurs in this graph by message passing. Suppose evidence is provided, by specifying initial values in the corresponding WM nodes, that an object is alive and has four legs. Messages from these two nodes are combined (via *product*) with the conditional probability distributions sent from the associated function nodes, and the feature variables are then *summarized* out (via integration) to yield messages concerning the concept. At the concept's join node, these messages are multiplied, along with the concept's prior distribution (from its function node), to yield a posterior distribution over the concept. In the case at hand, given the distributions provided, this yields a certainty (probability of 1) that the object is a dog. This result bounces back along the bidirectional links, combining with the conditional probability distributions for the features at the join nodes to yield predicted distributions at the WM nodes for the unspecified features; in this case that the object weighs 67 lb. and is mobile and brown.

Gradient-descent learning occurs after memory access has completed, based on the messages arriving at the function nodes rather than the WM nodes. Consider a fully specified supervised training trial, in which values are provided for the concept and all five of the features. Memory access yields the same kinds of messages just described, albeit with different contents; and the message that ultimately arrives back at a feature's (or the concept's) function node reflects a posterior distribution that is determined by the evidence in its WM node and the combined effect of what is in all of the other WM and function nodes.

How this combined effect is computed depends on the conditionals that define the graph structures that connect the WM and function nodes. The result is similar to that achieved via backpropagation in neural networks – where the structure of LTM both produces behavior and defines a set of dependencies for use in learning – but without a separate backpropagation phase, because the necessary messages are already provided by the bidirectionality of the summary product algorithm. There is also an analogy to chunking in Soar, where the structure of LTM both drives performance and yields dependencies for use in learning, but again without a separate dependency analysis.

Given the incoming message, Sigma first normalizes and smooths it before multiplying it by a learning rate parameter and adding the result to the existing function.

Distribution Tracking

The simplest form of learning supported in this manner is tracking the frequency with which different variable values are experienced over time. Figure 5, for example, shows a conditional defined via one condact and a function (initialized to uniform) that enables learning a distribution over four categories, here drawn from {*walker*, *table*, *dog*, *human*}. Given evidence, such as that the current category is *table*, messages proceed from WM to the function's factor node that yield a bump in the function's value for *table*. Without evidence, a uniform message occurs that leaves the function unchanged. After sufficient evidence, the conditional function should approximate the external distribution. For example, in a sample run with 10K examples, where the desired distribution over the four types was <.1, .2, .3, .4> and the actual distribution was <.1005, .1989, .2972, .4034>, the learned distribution was <.1091 .1985 .3027 .3897>.

```
CONDITIONAL Concept-Prior
  Condacts: Concept(value:c)
  Function(c): .25<*>
```

Figure 5: Conditional for uniform category prior.

The astute reader may have guessed that this conditional is the same as the one already implicitly mentioned in the semantic memory example, and would be right. Similar conditionals produce tracking of conditional probabilities, such as for alive given concept (Figure 6). When applied to NL POS tagging for 425 words, such conditional-probability learning predicts the most common tag for each

word. With a training set of 25,332 instances, this yields 90.3% correct on a test set of 5,138 instances; the same correct percentage as achieved when the baseline conditional probabilities are computed outside of Sigma.

```
CONDITIONAL Alive-Concept
  Condacts: Concept(value:c)
            Alive(value:a)
  Function(c,a): .5<* *>
```

Figure 6: Semantic memory conditional for uniform conditional probability of alive given category.

As demonstrated in (Rosenbloom, 2012), this same form of learning is also capable of acquiring action models for a simulated mobile robot in a 1D grid task, given the more complex conditional in Figure 1. Figure 7 shows another example from RL, for tracking projected future utilities of states. What differs here of interest is that the evidence for learning, rather than coming directly from outside experience, is computed internally from learned functions; in particular, the reward and future utility predicted for the next state, with this all ultimately grounding out in evidence for rewards. This ability to internally compute evidence for variables, and to use one learned function in computing the evidence for another, is one of the benefits of learning within a cognitive system/architecture.

```
CONDITIONAL Future-Utility
  Condacts: Projected(x:x value:u)
  Function(x,u): .1<* *>
```

Figure 7: Grid conditional for uniform future utility.

If the evidence is uncertain, learning must cope with distributions. Because Sigma's function representation is hybrid and mixed it is able to represent both discrete and continuous distributions. The learning algorithm itself is based on the relationship between a factor function and a message, both of which are represented in this manner, so the generality needed for learning from distributions already exists. Sigma can potentially handle anything from a single correct value to a uniform distribution that reflects a complete lack of knowledge, to arbitrary points in between.

This generality is leveraged via the conditional in Figure 7 to learn projected utilities, but it also enables, for example, learning a map that relates perceived objects to their locations in an uncertain world. In earlier work, Sigma was shown capable of localizing a simulated robot in a discrete 1D corridor (Chen *et al*., 2011). But in that work the map was fixed and known ahead of time by the system. In SLAM – where location evidence for map learning is computed from the perceived object and the map while the map is being learned – the map is initially set to a uniform distribution over the four perceptible objects {*wall*, *door1*, *door2*, *none*}, as in Figure 8, and then refined via experience. In an example run in a 1D corridor with objects assigned to six discrete locations as [*wall door1 none none door2 wall*] and noise in both object perception and the transition function – so that the simulated robot can neither

```
CONDITIONAL Object-Location-Map
   Conditions: Object(value:o)
   Condacts:   Location(x:x)
   Function(x,o): .25<* *>
```

Figure 8: Map conditional for uniform probability of object given location in discrete 1D corridor.

be sure of where it is or what it is seeing – it first predicts the correct object in each location after 138 moves (Figure 9, top). However, performance is fragile at this point, with some of the decisions being close calls that may be (incorrectly) reversed by later learning. After 1000 moves, the map is much crisper (Figure 9, bottom).

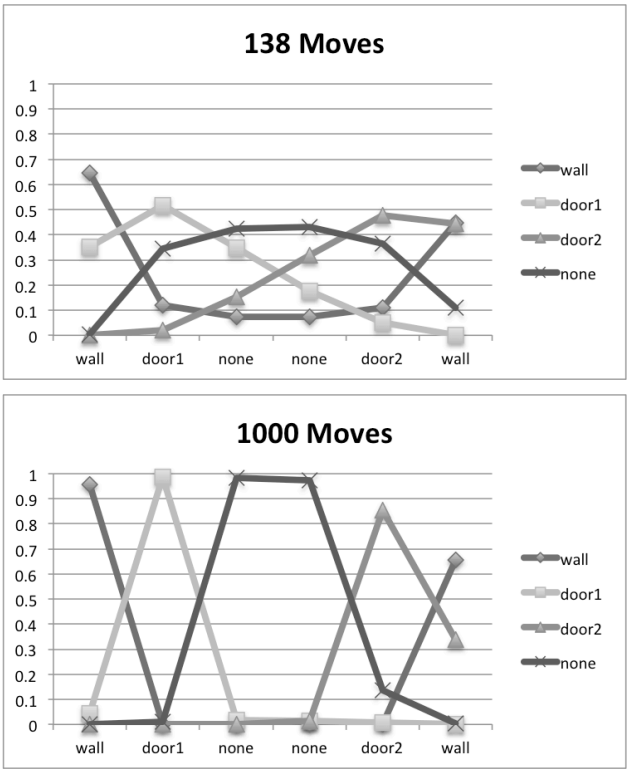

Figure 9: Map learned based on noisy perception and action models after 138 and 1000 moves.

Concept Formation

Concept formation may occur in either a supervised or an unsupervised manner. In supervised concept formation evidence is provided during training for both the features from which prediction is to occur and the result that is to be predicted. The distribution tracking in the previous section demonstrates one form of supervised learning, where learned conditional probabilities enable the child to be predicted when the parent is known. The focus of this section is instead on more general *generative learning* – where a full joint distribution is learned over the data – in the context of a *naïve Bayes classifier*; followed by an extension to unsupervised learning.

The supervised classifier learns a prior distribution for the object category plus conditional distributions for object features given the category. These are all learned in concert by distribution tracking; however, together they yield a classifier that is much more than just a tracked distribution. It can, for example, predict an object's category from evidence for any subset of its features, and flip this back around to predict values for unspecified features. Given that Sigma's semantic memory was earlier defined in terms of just such a naïve Bayes classifier, this work effectively demonstrates how gradient descent yields a mechanism for learning (the nonstructural aspects of) semantic memory.

Learning such naïve Bayes classifiers has so far proven adequate for sample datasets from the UCI ML repository and for basic conjunctive and disjunctive concepts. It has also yielded a classifier for word sense disambiguation. In preliminary experiments over the 30 most common words from the Senseval-3 database, with the same 449 examples used in both training and testing,[1] the baseline classifier – for the most frequent word sense – achieves 71.9% correct. A learned naïve Bayes classifier that includes for each word to be disambiguated features for 50 commonly co-occurring words, and which receives evidence about those that do co-occur in the example sentences, yields 76.8% correct. An extended classifier with an added feature for the correct part of speech achieves 78.0%. Over 746 examples for 6 words from the larger Semcor database, the baseline is 35% correct, a co-occurrence classifier yields 65.5% correct, and an extended classifier yields 70.4% correct.

Less impressively, a learned naïve Bayes classifier for part of speech (POS) tagging, where features correspond to the word to be tagged and the words just before and after it, was no better than the baseline accuracy – 90.3% – of predicting the most common tag. One simple alternative that did show improvement with respect to the baseline eliminated the prior on the tag and reversed the direction of the conditional distributions to be learned, yielding a direct tri-feature approach, and 92.8% correct over the test set.

A bigger issue with the naïve Bayes approach is that, as with one-layer neural networks, it fails on exclusive or (XOR). Gradient descent does support learning over hidden layers, since it applies to arbitrary Bayesian networks. But, rather than explicitly introducing a hidden layer into supervised learning, we'll shift instead to unsupervised learning, which implicitly introduces a hidden layer in the process of removing the overall focus on concept learning.

Unsupervised learning uncovers regularities in data, often via similarity-based clustering of examples. It can yield concept learning if the concept is added as a feature, even when it is not distinguished from the other features. In Sigma, this does not require a different learning algorithm, or even learning a non-naïve-Bayes structure (except that there is no prior). What is different is that the category becomes a feature, while a dummy variable, whose domain size determines how many clusters are used, is added in the category position (Figure 10). Neither a prior nor evidence is provided for this dummy variable. The conditional probabilities are initialized with random rather than uniform distributions, so that the symmetry otherwise inherent across clusters can be broken during learning.

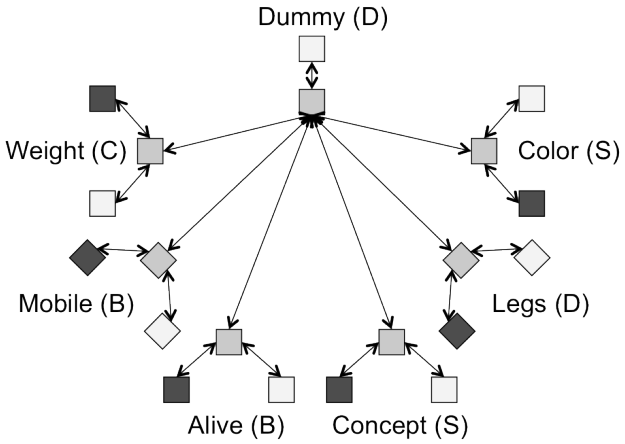

Figure 10: Semantic memory graph for unsupervised learning.

This form of unsupervised learning falls within the same class of mechanisms as expectation maximization, but is not quite identical to it. Evidence for feature values engenders distributions over the available clusters, which bounce back to yield expectations for the feature values. Learning occurs for the conditional probabilities of features given clusters, in service of aligning expected and actual feature values. Although such unsupervised learning requires more training on problems for which supervised learning works, it has the benefit of being able to learn more complex data sets, such as ones that embody XOR. In neural network terms, the dummy variable provides a hidden layer between the features and the category, with one hidden unit per cluster, making it possible to learn these more difficult functions. For XOR, after 30 passes over the four training examples, a correct unsupervised classifier is learned (Figure 11).

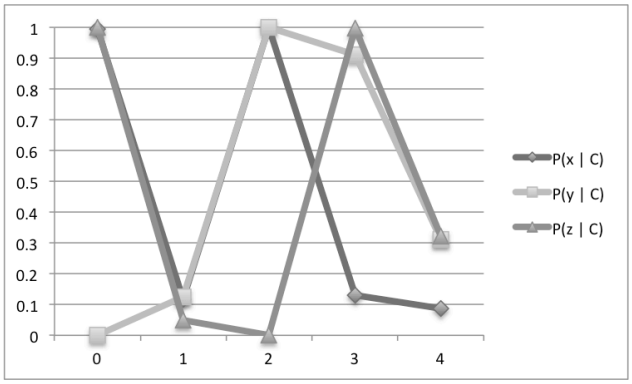

Figure 11: Learned conditional probabilities of x, y and z being *true* given the cluster C (in 0-4) for z = XOR(x, y).

[1] Because the results here focus on the breadth of learning behaviors rather than on learning accuracy per se, reusing training data during testing is not the problem it would be otherwise.

Reinforcement Learning

Reinforcement learning in Sigma is covered in more depth in (Rosenbloom, 2012), but is worth revisiting briefly here because of how it exhibits more complex learning behaviors from gradient-descent over ten conditionals with four learned functions (including action models). The reward is learned from external evidence via distribution tracking. Action model learning is based on the conditional in figure 1. The projected (discounted) future utility and the Q values are also learned in a direct manner, but based on internally computed evidence. The conditional that is key to this consults the current location and operator, plus distributions over the reward and projected utilities for the (predicted) next state – all of which distributions are learned – in providing evidence for the Q values and the projected utility in the current state (Figure 12). Affine transforms are used to add the projected utility to the actual reward (via *translation*) and to discount the result by .95 (via *scaling*).

```
CONDITIONAL Backup
 Conditions: Location(state:s x:x)
             Selected(state:s operator:o)
             Location*Next(state:s x:nx)
             Reward(x:nx value:r)
             Projected(x:nx value:p)
 Actions: Q(x:x operator:o value:.95*(p+r))
          Projected(x:x value:.95*(p+r))
```

Figure 12: Conditional for backing up rewards/utilities.

Conclusion and Future Work

What makes this approach to learning particularly attractive architecturally is that it is: (1) *local*, depending only on the message back into a factor node to determine how to update the node's function; (2) *incremental* – i.e., online – as is appropriate in a cognitive architecture/system; and (3) applicable to any function in the graph with a child variable. A simple learning mechanism can thus be integrated deeply, simply and pervasively into Sigma to yield a range of useful learning behaviors; including forms of procedural (reinforcement and action) and declarative (supervised and unsupervised concept) learning, and map learning (in the context of SLAM). The approach works for both symbolic and numeric data, and for both discrete and continuous distributions; and it works across a range of application domains, from standard toy problems, to larger scale NL classifications, to (simulated) mobile robot problems.

Still, much remains to be done. First, a more sophisticated approach to learning continuous functions is required. Such functions are currently learned in the same manner as discrete functions, without using summary parameters that would enable generalizing across domain elements. Second, learning here involves two parameters – learning rate and smoothing threshold – but an architecture either must not have parameters, or it needs values that are usable everywhere or that can automatically be tuned to new problems. Third, the algorithm needs to handle undirected factor functions; i.e., those without child variables. Fourth, true structure learning is required. Fifth, and finally, follow up is needed on how much further this approach can extend.

Acknowledgements

This work has been sponsored by the U.S. Army, the Air Force Office of Scientific Research, and the Office of Naval Research. Statements and opinions expressed do not necessarily reflect the position or the policy of the United States Government, and no official endorsement should be inferred.

References

Chen, J., Demski, A., Han, T., Morency, L-P., Pynadath, P., Rafidi, N. & Rosenbloom, P. S. (2011). Fusing symbolic and decision-theoretic problem solving + perception in a graphical cognitive architecture. *Proceedings of the 2nd International Conference on Biologically Inspired Cognitive Architectures* (pp. 64-72). Amsterdam: IOS Press.

Deutsch, D. (2011). *The Beginning of Infinity: Explanations that Transform the World*. London, UK: Penguin Books.

Forgy, C. L. (1982). Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. *Artificial Intelligence*, *19*, 17-37.

Koller, D. & Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques*. Cambridge, MA: MIT Press.

Kschischang, F. R., Frey, B. J. & Loeliger, H.-A. (2001). Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, *47*, 498-519.

Laird, J. E. (2012). *The Soar Cognitive Architecture*. Cambridge, MA: MIT Press.

Laird, J. E., Rosenbloom, P. S. & Newell, A. 1986. Chunking in Soar: The anatomy of a general learning mechanism. *Machine Learning*, *1*, 11-46.

Rosenbloom, P. S. (2006). A cognitive odyssey: From the power law of practice to a general learning mechanism and beyond. *Tutorials in Quantitative Methods for Psychology*, *2*, 43-51.

Rosenbloom, P. S. (2010). Combining procedural and declarative knowledge in a graphical architecture. *Proceedings of the 10th International Conference on Cognitive Modeling* (pp. 205-210).

Rosenbloom, P. S. (2011). Bridging dichotomies in cognitive architectures for virtual humans. *Proceedings of the AAAI Fall Symposium on Advances in Cognitive Systems*. Menlo Park, CA: AAAI Press.

Rosenbloom, P. S. (2012). Deconstructing reinforcement learning in Sigma. *Proceedings of the 5th Conference on Artificial General Intelligence* (pp. 262-271). Berlin: Springer.

Russell, S., Binder, J., Koller, D. & Kanazawa, K. (1995). Local learning in probabilistic networks with hidden variables. *Proceedings of the 14th International Joint Conference on AI* (pp. 1146-1152). San Mateo, CA: Morgan Kaufmann.