SAND2011-0710C

An Implemented Architecture for Feature Creation and General Reinforcement Learning

Brandon Rohrer

Sandia National Laboratories,
Intelligent Systems, Robotics, and Cybernetics Group,
Albuquerque, NM, USA
brrohre@sandia.gov
http://www.sandia.gov/~brrohre

Abstract. BECCA is a brain-emulating cognition and control architecture. It was developed in order to perform general reinforcement learning, that is, to enable unmodeled systems operating in unstructured environments to perform unfamiliar tasks. It accomplishes this through complementary feature creation and reinforcement learning (RL) algorithms. Both of these algorithms are novel and have characteristics that have not been demonstrated by previously existing algorithms. This paper contains a description of BECCA and the algorithms that underlie it, illustrated with a sample implementation.

Keywords: feature creation, deep learning, feature extraction, model-based, reinforcement learning, robot, unstructured environment

1 Introduction

Creating a general learning machine has been one of the grand goals of artificial intelligence (AI) since the field was born. Efforts to achieve this goal may be divided into two categories. The first uses a depth-first approach, solving problems that are complex, yet narrow in scope, such as playing chess. The assumption underlying these efforts is that an effective solution to one such problem may eventually be generalized to solve a broad set of problems. The second category emphasizes breadth over depth, solving large classes of simple problems. The assumption underlying these efforts is that a general solution to simple problems may be scaled up to address more complex ones. The work described here falls into the second category, focusing on breadth.

The motivating goal for this work is to find a solution to **natural world interaction**, the problem of navigating, manipulating, and interacting with arbitrary physical environments to achieve arbitrary goals. In this context, *environment* refers both to the physical embodiment of the agent and to its surroundings, which may include humans and other embodied agents.

The agent design presented here is loosely based on the structure and function of the human brain and is referred to optimistically as a brain-emulating cognition and control architecture (BECCA). The remainder of the paper contains an algorithmic description of BECCA, an illustration of its operation on a simple task, and a discussion of its capabilities and limitations with respect to other learning algorithms.

2 Method

A BECCA agent interacts with the world by taking in actions, making observations, and receiving reward. (See Figure 1.) Formulated in this way, natural world interaction is a general RL problem, [15] and BECCA is a potential solution. Specifically, at each discrete time step, BECCA performs three functions:

- It reads in an observation, a vector $\mathbf{o} \in \mathbb{R}^m | 0 \leq o_i \leq 1$.
- It receives a reward, a scalar $r \in \mathbb{R} | -1 \leq r \leq 1$.
- It outputs an action, a vector $\mathbf{a} \in \mathbb{R}^n | 0 \leq a_i \leq 1$.

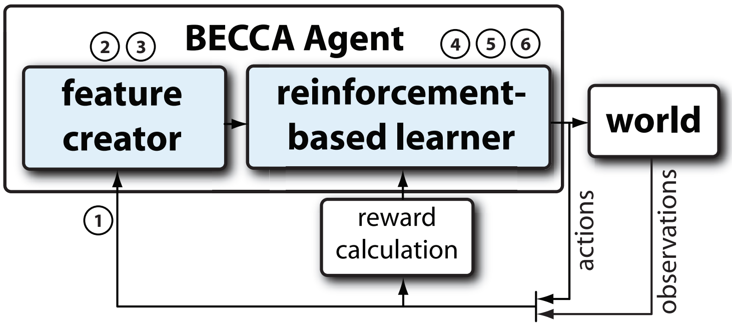

Fig. 1. At each timestep, the BECCA agent completes one iteration of the sensing-learning-planning-acting loop, consisting of six major steps: 1) Reading in observations and reward. 2) Updating its feature set. 3) Expressing observations in terms of features. 4) Predicting likely outcomes based on an internal model. 5) Selecting an action based on the expected reward of likely outcomes. 6) Updating its model.

Because BECCA is intended for use in a wide variety of environments and tasks, it can make very few assumptions about them beforehand. Although it is a model-based learner, it must learn an appropriate model through experience. BECCA uses two key algorithms to do this: an unsupervised feature creation algorithm (See Algorithm 1 and Figure 2.) and a tabular model construction algorithm (See Algorithm 2 and Figure 3).

Algorithm 1 Feature Creator

```
Input: observation vector
Output: feature_activity vector

1:  form input vector by concatenating observation
        and previous modified_feature_activity
2:  update estimate of correlation between inputs
3:  if MAX(correlation) > $threshold_1$ then                creates a new group
4:      add the two input elements achieving the
            maximum correlation to the new group
5:      while NOT(stop_condition_met) do
6:          find mean_correlation between each
                remaining input and group members
7:          if MAX(mean_correlation) > $threshold_2$ then
8:              add the corresponding input element to the group
9:          else stop_condition_met
10: for each group do
11:     if MIN (COSINE DISTANCE (input, features)) > $threshold_3$ then
12:         add normalized input to set of features
13:     for each feature do
14:         feature_activity = feature · input
15:     modified_feature_activity = WINNER-TAKE-ALL(feature_activity)
```

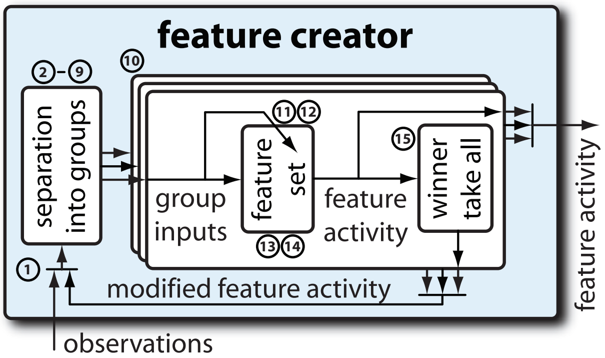

Fig. 2. Block diagram of the feature creator, illustrating its operation. Numbered labels refer to steps in Algorithm 1.

The feature creation algorithm breaks elements of the input into groups, based on the estimates of correlation between them. Each group represents a subspace of the input space. Features, unit vectors in each group subspace, are created based on subsequent inputs. Subsequent inputs are also projected along these features to find the feature activity at each time step, which is in turn passed on to the reinforcement learner. In addition, the most active feature in each group inhibits all others, and the resulting feature activity is fed back and combined with the next observation to form the input for the next time step. The recursive nature of the feature creation algorithm allows more complex features to be created from combinations of simpler ones.

Algorithm 2 REINFORCEMENT LEARNER

```
Input: feature_activity vector
Output: action vector

1:   attended_feature is the feature with MAX(feature_activity)
2:   decay working_memory and add attended_feature
3:   update model, consisting of cause–effect transition pairs
3.1:     cause = NORMALIZE(previous_action + previous_working_memory)
3.2:     effect = NORMALIZE(attended_feature)
3.3:     effect_matches = all transition pairs in model with matching effect
3.4:     if MAX(SIMILARITY(effect_matches, cause) < $threshold_4$)
3.5:         add new cause–effect pair to the model
3.6:     else
3.7:     increment count of nearest cause–effect pair
4:   get predictions from model
4.1:     for each model entry do
4.2:         weighted_effect = effect × LOG(count + 1) ×
                 SIMILARITY(cause, working_memory)
4.3:     expected_effect = NORMALIZE(weighted_effects)
5:   select action
5.1:     expected_reward = expected_effect × reward_map
5.2:     find cause–effect pair associated with MAX(expected_reward)
5.2:     find action associated with cause–effect pair
5.3:     on a fraction, $\alpha$, of time steps,
             generate a random exploratory action
6:   update reward_map using modified_feature_activity and reward
```

The reinforcement learner takes in feature activity and reward and selects an action to execute. An attention filter selects the most salient feature. Working memory is a weighted combination of several recent attended features. The attended feature, previous action, and working memory are used to update the model. Based on the current working memory, the model produces a set of predictions. Predicted outcomes with high expected reward are found, and the

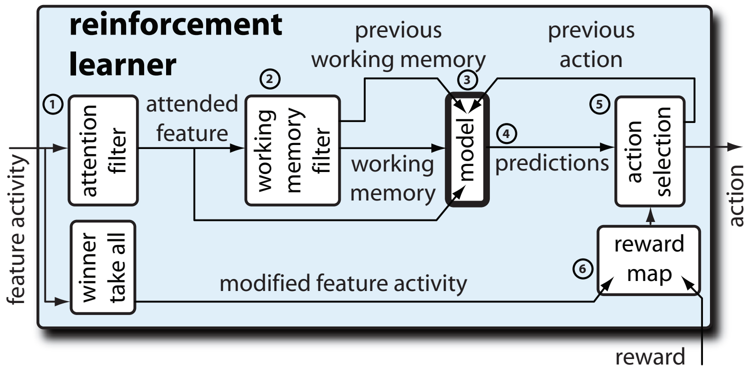

Fig. 3. Block diagram of the reinforcement learner, illustrating its operation. Numbered labels refer to steps in Algorithm 2.

action associated with the highest reward is selected. Occasionally a random exploratory action is substituted for the greedy action.

3 Results

A simple simulation was constructed to demonstrate BECCA in operation. (See Figure 4.) An agent was allowed to pan a virtual camera along a mural. The camera produced a 5 × 5 pixel virtual image of its field of view of the mural, with each pixel taking on a value between 0 (black) and 1 (white). (See Figure 5.) Each observation comprised the 25 pixel values, v, as well as their complement, $1 - v$. The action vector represented four discrete movement steps in each direction. When concatenated, each observation and action resulted in a 58 element vector, which was passed to BECCA at each time step. BECCA was rewarded for directing its "gaze" at a specific region in the center of the mural. It learned to do this reasonably well after approximately 6000 time steps. Complete MATLAB code for the simulation and BECCA implementation can be found at [13].

Over ninety percent of the 6000 time steps was spent learning a feature representation of the inputs. Most of this time was required for the correlation estimate to reach a sufficiently high level to nucleate a group. The correlation estimate was calculated on line and incrementally. In order to reduce noise and avoid spurious groupings, it was incremented very gradually. After a groups was created, features were learned within several time steps, and the system model was learned in several hundred time steps after that. In fact, all aspects of learning (groups, features, and model) were ongoing throughout the simulation, but individual aspects were most prominent at different times, demarcating stages of learning.

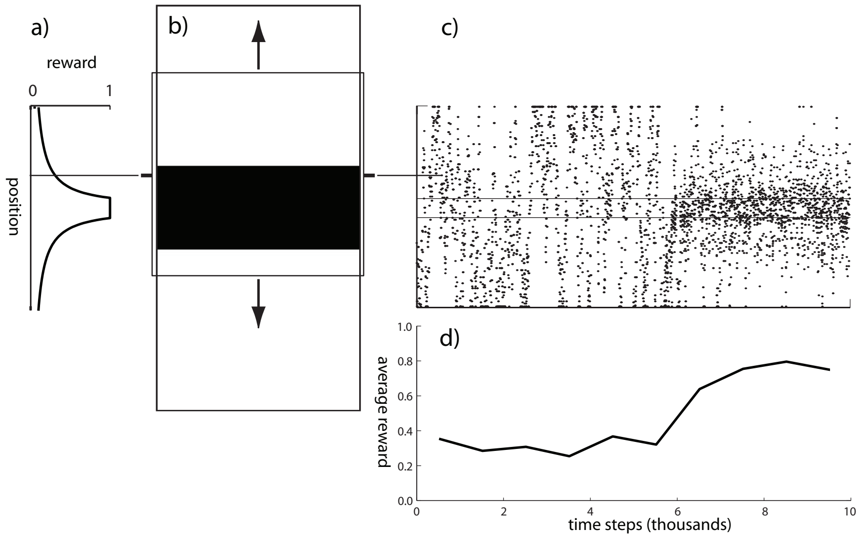

Fig. 4. A one-dimensional visual tracking task. **a)** The reward signal as a function of the agent's gaze direction. **b)** The task environment. The white field with black bar provided a visual world. The frame representing the agent's field of view and the direction of its gaze is also shown. The agent executed panning movements to adjust its gaze direction. **c)** The agent's gaze direction history. Fine lines indicate the region resulting in maximum reward. After approximately 6000 time steps the gaze focused more on the high reward region. **d)** Average reward per time step. After approximately 6000 time steps the average reward roughly doubled.

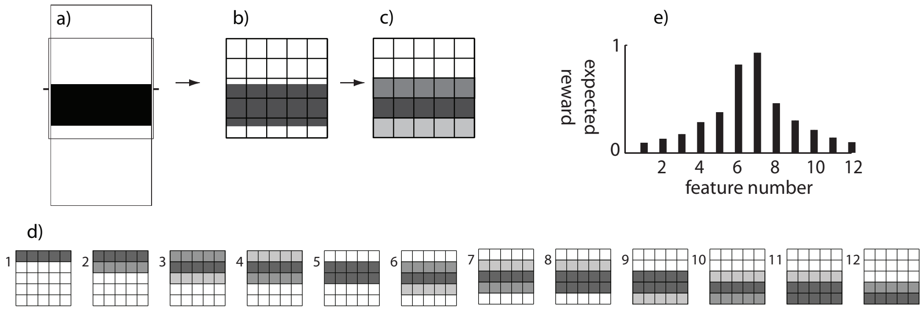

Fig. 5. Feature creation in the visual tracking task. **a)** BECCA's visual field at each time step spanned only a portion of the mural. **b)** Although the visual field was 200 × 200 pixels, it was pixelized into a 5 × 5 pixel array. **c)** Pixelization was achieved by averaging the pixel values within each superpixel. **d)** The visual tracking task resulted in twelve features. **e)** The reward associated with each feature in BECCA's experience corresponded closely to the reward expected, based on the reward-position function in Figure 4a.

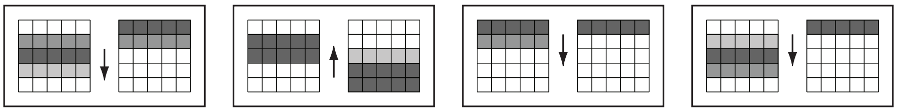

Fig. 6. The most common sequences recorded within the model. Each arrow indicates the direction of gaze shift and results in image motion in the opposite direction.

4 Discussion

In the visual servoing task, BECCA demonstrated its ability to achieve better than random performance on an RL task with a 58-dimensional observation-action space, about which it had no prior knowledge.

The visual servoing task was trivial and has many straightforward solutions that incorporate some knowledge of the task. For instance, after studying the task, a human designer could hand-select useful features and rules for how to behave when those features were observed. The agent custom designed in this way would likely learn faster and perform better than BECCA in a direct comparison, but such a comparison would be misleading. In the case of the task-customized agent, a human performed the feature creation and policy construction functions that BECCA handled itself. Thus BECCA would not be compared to the task-customized agent, but rather to the agent-designer team. While such a comparison might be interesting, it would not accomplish the goal of comparing the learning agents themselves. In order to have a meaningful comparison, each agent should have the same set of domain information available to it beforehand.

BECCA was designed to handle a set of natural interaction tasks that is as broad as possible. The visual servoing task did not adequately illustrate this breadth; it was selected as a means to illustrate BECCA's operation as simply as possible. It should be noted that, simple as it is, a problem with a 58-dimensional state space is considered very challenging or infeasible for many learning methods.

The formulation of natural world interaction as a general RL problem was selected to keep it relevant to physical agents, such as robots, performing a wide variety of tasks in the physical world. Machine leaning methods focused on narrow aspects of learning, such as perception, classification, clustering, and learning production rules, all require a supporting framework in order to be applied to produce goal-directed behavior. An RL framework provides this. Specifically, in the general RL problem formulation used here, only the dimensionality of the observation and action spaces are known beforehand, and the agent seeks to maximize its reward. This framework is roughly descriptive of human and animal agents acting in their environments, suiting it to describe the natural world interaction problem for machine agents as well. Within such a general RL framework, more specialized algorithms may then be used in concert to most effectively seek out reward.

While BECCA has been designed using insights gleaned from experimental psychology and cognitive neuroscience, it is not intended to be a cognitive or neural model of how a human or animal brain operates. BECCA incorporates computational mechansisms, such as the dot product or winner-take-all, that have plausible neuronal implementations, but there is no associated claim, express or implied, that the brain actually performs those calculations. Put simply, BECCA's purpose is not to describe the brain, but to perform like it.

4.1 Related work: Unsupervised learning

BECCA's feature creation algorithm is an example of an unsupervised learning method, in that it learns a structure based only on the observed data. There are many other examples of algorithms that do this automatically, although none with the same properties as BECCA. Feature creation can be described as a state space partitioning problem, where each region of the space corresponds to a feature. Tree-based algorithms are particularly well-suited to this, and several have been proposed. The Parti-Game algorithm [11] uses a greedy controller to crawl through a partitioned state space. The G Algorithm [3], U-Tree [9], Continuous U-Tree [16], AMPS [6], and decision trees of Pyeatt and Howe [12] are all approaches used in conjunction with dynamic programming methods or the popular temporal difference method, Q-learning [17], to estimate the value function across the state space. Whenever a subspace's value estimate is shown to be inadequate, the subspace is divided. The G Algorithm handles binary data, U-Trees handle discrete data, and Continuous U-Trees, AMPS, and Pyeatt and Howe's approach handle continuous data.

Taking a broader view, there are many unsupervised learning methods developed with different sets of assumptions, but BECCA's feature creator provides a novel collection of characteristics. It is *on-line*, meaning that it incorporates data points one at a time and modifies its feature representation incrementally. It is *hierarchical* in that it can use created features to construct still higher level features. It is *stable* in the sense that feature definitions do not change once they are created. In contrast to many unsupervised learning algorithms, BECCA's feature creator does not assume the number of features that exist in the underlying data. Once the dimensionality of the state space is defined, the feature creator always starts from the same initial conditions, so there is no need to carefully pick initial values for cluster parameters. Only three thresholding constants need to be selected. Like other unsupervised learning methods in which the number of clusters is not specified, the validity of the features found depends entirely on the appropriateness of the measure of feature goodness. And, as with other unsupervised learning algorithms of its class, there are no theoretical performance guarantees.

4.2 Related work: Deep learning

The problem of hierachical feature creation is closely related to deep learning. [2] Deep learning approaches seek to discover and exploit the underlying structure of a world by creating higher level, lower-dimensional representations of the system's input space. Deep learning algorithms include Convolutional Neural

Networks (CNN) [7], Deep Belief Networks (DBN) [4], and the Deep SpatioTemporal Inference Network (DeSTIN) [5]. Deep learning algorithms such as these are alternative approaches, worthy of consideration for automatic concept acquisition, although they differ somewhat from BECCA's feature creator. CNNs are designed to work with two-dimensional data, such as images, and they do not apply to arbitrarily structured data, as BECCA does. By using several layers of Restricted Boltzmann Machines, DBNs are capable of generating sophisticated features that allow it to interpret novel inputs. However, they are typically applied to the supervised learning problem of discrimination, and require a substantial amount of labeled data in order to be adequately trained. Whether DBNs can be applied to the unsupervised learning problem of feature creation is unclear. DeSTIN incorporates both unsupervised and supervised learning methods and appears to be fully capable of hierarchical feature creation. It has been published only recently; future papers describing its operation and performance will allow a more detailed comparison with BECCA's feature creator.

4.3 Related work: Reinforcement learning

There are several well known methods for solving the reinforcement learning problem, that is, given state inputs and a reward at each timestep, choose actions that maximize the future reward. Some examples that have been applied in agents include Q-learning [17], the Dyna architecture [14], Associative Memory [8], and neural-network-based techniques including Brain-Based Devices [10] and CMAC [1]. BECCA's reinforcement learner is another such algorithm. It is *on-line* and *model-based*, meaning that as it accumulates experience it creates and refines an internal model of itself and its environment. It differs from most previous methods in two ways. First, its internal model is not a first order Markov model. Instead, by using cause-effect transition pairs in which the cause is a compressed version of the agent's recent state history, it creates a compressed higher order Markov model. This potentially allows BECCA to learn more sophisticated state dynamics and to record distinct sequences more naturally. Second, BECCA's reinforcement learning algorithm can handle a growing state space. This is necessary because it must work in tandem with BECCA's feature creator, which continues to identify new features throughout the life of the agent.

Acknowledgements

This work was supported by the Laboratory Directed Research and Development program at Sandia National Laboratories. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.

References

1. J. Albus. A new approach to manipulator control: Cerebellar model articulation controller (CMAC). *Journal of Dynamic Systems, Measurement and Control*, 97:220–227, 1975.
2. I. Arel, D. C. Rose, and T. P. Karnowski. Deep machine learning—a new frontier in artificial intelligence research. *IEEE Computational Intelligence Magazine*, November 2010.
3. D. Chapman and L. P. Kaelbling. Input generalization in delayed reinforcement learning. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 726–731, 1991.
4. G. E. Hinton, S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527, 2006.
5. D. Rose I. Arel and B. Coop. Destin: A deep learning architecture with application to high-dimensional robust pattern recognition. In *Proc. AAAI Workshop Biologically Inspired Cognitive Architectures (BICA)*, 2009.
6. M. J. Kochenderfer. *Adaptive Modelling and Planning for Learning Intelligent Behaviour.* PhD thesis, University of Edinburgh, School of Informatics, 2006.
7. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998.
8. S. E. Levinson. *Mathematical Models for Speech Technology.* John Wiley and Sons, Chichester, England, 2005. pp. 238-239.
9. A. K. McCallum. *Reinforcement learning with selective perception and hidden state.* PhD thesis, University of Rochester, Computer Science Department, 1995.
10. J. L. McKinstry, G. M. Edelman, and J. L. Krichmar. A cerebellar model for predicitive motor control tested in a brain-based device. *Proceedings of the National Academy of Sciences*, 103(9):3387–3392, 2006.
11. A. W. Moore and C. G. Atkeson. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning*, 21:199–233, 1995.
12. L. D. Pyeatt and A. E. Howe. Decision tree function approximation in reinforcement learning. In *Proceedings of the Third International Symposium on Adaptive Systems: Evolutionary Computationand Probabalistic Graphical Models*, pages 70–77, 2001.
13. B. Rohrer. BECCA code page. http://www.sandia.gov/˜brrohre/code.html, 2010.
14. R. S. Sutton. *Planning by incremental dynamic programming*, chapter Proceedings of the Eighth International Workshop on Machine Learning, pages 353–357. Morgan Kaufmann, 1991.
15. R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, Massachusetts, 1998.
16. W. T. B. Uther and M. M. Veloso. Tree based discretization for continuous state space reinforcement learning. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-98)*, Madison, WI, 1998.
17. C. J. C. H. Watkins and P. Dayan. Technical note: Q-learning. *Machine Learning*, 8(3-4):279–292, May 1992.