Learning behavior hierarchies via high-dimensional sensor projection

Sagar Ratna Bajracharya
International Centre for Integrated Mountain Development
31 PUBLICATIONS **368** CITATIONS
SEE PROFILE

Ross W Gayler
https://www.rossgayler.com
56 PUBLICATIONS **1,046** CITATIONS
SEE PROFILE

Learning Behavior Hierarchies via High-Dimensional Sensor Projection

Simon D. Levy and **Suraj Bajracharya**
Computer Science Department
Washington and Lee University
Lexington Virginia 24450, USA

Ross W. Gayler
La Trobe University
Melbourne VIC 3086, Australia

Abstract

We propose a knowledge-representation architecture allowing a robot to learn arbitrarily complex, hierarchical / symbolic relationships between sensors and actuators. These relationships are encoded in high-dimensional, low-precision vectors that are very robust to noise. Low-dimensional (single-bit) sensor values are projected onto the high-dimensional representation space using low-precision random weights, and the appropriate actions are then computed using elementwise vector multiplication in this space. The high-dimensional action representations are then projected back down to low-dimensional actuator signals via a simple vector operation like dot product. As a proof-of-concept for our architecture, we use it to implement a behavior-based controller for a simulated robot with three sensors (touch sensor, left/right light sensor) and two actuators (wheels). We conclude by discussing the prospects for deriving such representations automatically.

KEYWORDS: *Knowledge representation, Vector Symbolic Architectures, behavior-based robotics*

Sensors and Symbols

How can a robot use its sensory input to reason about the actions it should take? With the exception of Braitenberg's hard-wired vehicles (Braitenberg 1984), answers to this question have usually involved some kind of explicitly symbolic computation. Even the behavior-based robots of Brooks and his students, presented as a radically minimalist alternative to classical planning architectures (Brooks & Flynn 1989) employed a traditional finite-state machine to implement a subsumption hierarchy of sensor/actuator behaviors. For anything but the most trivial kinds of sensor/actuator relationships, it would seem that some sort of discrete language / automata-like behavior is necessary.

In this position paper we will present evidence that non-trivial sensor/actuator relationships can be mediated by an architecture involving only the sensor outputs, actuator inputs, and fixed-size vectors of neural-like weights. Despite their extreme computational simplicity, these architectures can be easily "programmed" to perform subsumption hierarchies and other rule-like behaviors in the service of interesting tasks, in the absence of explicit if/then statements or other traditional symbolic constructs.

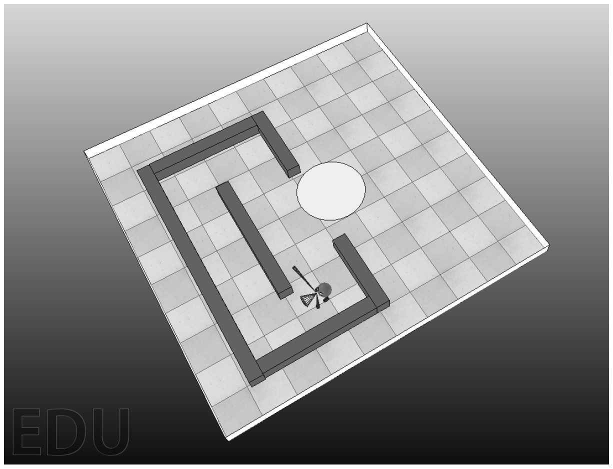

Figure 1: A simple corral-escape testbed for behavior-based robotics

The Task

As a testbed for our architecture, we chose a behavior-based robotics exercise taken from an undergraduate computer science textbook (Kumar 2011). Figure 1 shows an implementation of this exercise in the popular V-REP Simulator from Coppelia Robotics. Our robot has two powered wheels, a single front-facing touch sensor, and two upward-facing light sensors (left and right). Its task is to escape the corral in which it starts, enter the patch of light represented by the disk, and remain there. All three sensors are binary (on/off), as are the wheels (forward or backward rotation).

Classic Behavior-Based Solution

The corral-exiting task in Figure 1 can be solved using a hierarchy of three behaviors: *seekLight*, *avoidObstacles*, and *cruise*. Each behavior takes precedence over the ones after it. In pseudocode:

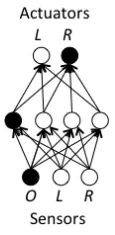

Figure 2: A localist neural network implementing the behavior-based robot controller

```
if senseLightLeft and senseLightRight:
    leftMotor, rightMotor = +1,+1  // stay in light

else if senseLightLeft:
    leftMotor, rightMotor = -1,+1  // turn left

else if senseLightRight:
    leftMotor, rightMotor = +1,-1  // turn right

else if senseObstacle:
    leftMotor, rightMotor = -1,+1  // turn left

else:
    leftMotor, rightMotor = +1,+1  // cruise
```

Here, the first three `if` conditionals support light-seeking, the third implements obstacle-avoidance, and the fourth implements the default cruising behavior. As in its use for an introductory computer science lab, this specification is simple enough to be derivable in a reasonable amount of time, but nontrivial enough to illustrate an important design principle.

Localist Network Implementation

The behavior in the pseudocode above can also be produced in a simple neural network using a "localist" principle. The three binary sensors map to four possible motion patterns (forward, back, left, right), which determine the values of the actuators . Figure 2 shows an implementation of the corral-escape network using such a scheme, with the *avoidObstacles* behavior used as an example.

Distributed Network Implementation

A localist network like the one in Figure 2 is easily implemented in software or hardware. As proponents of distributed representations have argued long argued, however, such networks lack many of the desirable features (content-addressability, robustness to lesioning and noise, generalization) of a network in which representations are distributed among many computational units (Rumelhart & McClelland 1986). [1] Such arguments lead us to wonder whether we can implement our control regime in a network

[1]More recent criticism of localist networks (Stewart & Eliasmith 2009) has described their inability to scale up to realistic-sized problems because of the combinatorial explosion in connections that results when each entity must be represented by a separate unit.

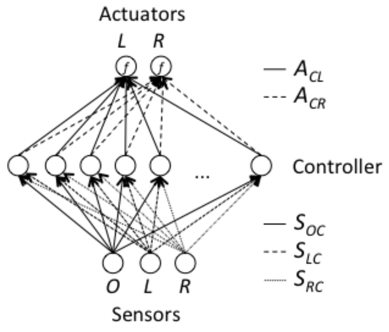

Figure 3: A distributed neural network implementing the behavior-based robot controller. The function f constrains the network output to appropriate sensor values. The ... represents a potentially very large number of units.

that uses distributed representations. A schematic of such a network is shown in Figure 3. In the remainder of this paper we provide details of one such network that we have constructed for our behavioral-control task.

Vector Symbolic Architectures

Vector Symbolic Architecture, or VSA (Gayler 2003) describes a class of connectionist models that use high-dimensional vectors of low-precision numbers to encode structured information as distributed representations. VSAs can represent complex entities such as multiple role/filler relations, trees, and graphs in a way that every entity – no matter how simple or complex – is represented by a pattern of activation distributed over all the elements of the vector.

For our purposes in this paper, we make use of two operations on vectors: an elementwise multiplication operation $\otimes$ that associates or binds vectors, and an elementwise vector-addition operation $+$ that superposes vectors or adds them to a set.[2] For example, given the vector representation of sensor states S_1 and S_2 and associated actions A_1 and A_2, we can represent these associations as the vector $S_1 \otimes A_1 + S_2 \otimes A2$.

If the vector elements are taken from the set $\{-1, +1\}$, then each vector is its own inverse, and binding and unbinding can both be performed by the same operator: $X \otimes (X \otimes Y) = Y$.[3] Because these vector operations are commutative and associative, another interesting

[2]A third operator, permutation, can be used for quoting or protecting vectors from the other operations, but is not needed for the present work.

[3]After applying the addition operation the elements of the vectors are no longer restricted to the set $\{-1, +1\}$. However, the binding and unbinding results still hold approximately because each element can be interpreted as a weighted sign (i.e. an element of the set $\{-1, +1\}$) and the information is mostly carried by the signs. This degree of approximation is acceptable because VSAs are massively resistant to noise.

property holds: the unbinding operation can be applied to a set of associations just as easily as it can to a single association: $Y \otimes (X \otimes Y + W \otimes Z) = X + Y \otimes W \otimes Z$. If the vector elements are chosen randomly, then we can rewrite this equation as $Y \otimes (X \otimes Y + W \otimes Z) = X + noise$, where $noise$ is a vector orthogonal (completely dissimilar) to any of our original vectors W, X, Y, and Z. If we like, the noise can be removed through a "cleanup memory" (e.g. Hopfield Net) that stores the original vectors; however, the important point is that a single association (or set of associations) can be quickly recovered from a set (or larger set) of associations using the same simple operator that creates the associations, in a time that is independent of the number of associations. By using vectors of several thousand dimensions, we obtain a tremendous number of mutually orthogonal vectors (potential symbols) that are highly robust to distortion (Kanerva 2009).

A VSA Robot Controller

With this understanding of Vector Symbolic Architecture, we can now see how to use VSA to build a behavior-based controller for our three-sensor, two-motor robot: from the N-dimensional vector space $\{-1, +1\}^N$ (where typically $N > 1000$) we randomly choose weight vectors S_{OC}, S_{LC}, S_{RC} to represent the weights from the sensors to the control layer, and weight vectors A_{CL} and A_{CR} to represent the weights from the control layer to the actuators. The control layer can then be constructed as the following N-dimensional vector sum (in which we have grouped terms according to the behavior they implement):

$$
\begin{aligned}
&//Seek\ Light\\
&(S_{LC} + S_{RC}) \otimes (A_{CL} + A_{CR}) +\\
&(S_{OC} + S_{LC} + S_{RC}) \otimes (A_{CL} + A_{CR}) +\\
&S_{LC} \otimes A_{CR} +\\
&(S_{OC} + S_{LC}) \otimes A_{CR} +\\
&S_{RC} \otimes A_{CL} +\\
&(S_{OC} + S_{RC}) \otimes A_{CL} +\\
&\\
&S_{OC} \otimes A_{CR} + //Avoidobstacles\\
&\\
&(A_{CL} + A_{CR})//Cruise
\end{aligned}
$$

Each term in this sum corresponds to an if/then rule in the pseudocode. For example, the term $(S_{LC} + S_{RC}) \otimes (A_{CL} + A_{CR})$ associates a reading of (1,1) on light sensors with forward motion of both wheels. To see how this vector implements our behavior-based controller, consider what happens when the input to the sensors is (0, 1, 1); i.e., no obstacle, left light sensing, right light sensing. This input results in the weight vector $S_{LC} + S_{RC}$ being multiplied elementwise by the control vector. This multiplication yields the vector $A_{CL} + A_{CR} + noise$, where $noise$ is a random vector orthogonal to the actuator weights A_{CL} and A_{CR}. If the function f in Figure 3 computes the dot product of this result vector with these actuator weights, the resulting actuator values are (+1,+1), driving the robot forward in the presence of light. Similar reasoning holds for other combinations of sensor inputs.

We have successfully tested our VSA robot controller against the original (if/then/else) controller using the V-REP simulation shown in Figure 1. C++ code for our simulation, along with the V-REP scene file and instructions, can be downloaded from `tinyurl.com/vsarobot`.

Conclusion and Future Work

We have presented a novel knowledge-representation architecture allowing a robot to encode arbitrary hierarchical / symbolic relationships between sensors and actuators. This architecture uses a neural network with distributed representations that exhibit robustness to noise, generalizability, and symbol substitution, which opens the door to rule-like behavior. Because our network weights are hand-coded rather than learned, we have chosen to present them in a position paper for the purposes of this workshop. In the future we hope to be able to obtain the weights through an evolutionary algorithm or similar learning approach.

References

Braitenberg, V. 1984. *Vehicles: Experiments in Synthetic Psychology*. MIT Press.

Brooks, R., and Flynn, A. 1989. Fast, cheap, and out of control: A robot invasion of the solar system. *Journal of the British Interplanetary Society* 42:478–485.

Gayler, R. 2003. Vector symbolic architectures answer Jackendoff's challenges for cognitive neuroscience. In Slezak, P., ed., *ICCS/ASCS International Conference on Cognitive Science*, CogPrints, 133–138. Sydney, Australia: University of New South Wales.

Kanerva, P. 2009. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive Computation* 1:139–159.

Kumar, D. 2011. *Learning Computing With Robots: Python + Scribbler or Scribbler2*. Institute for Personal Robots in Education.

Rumelhart, D., and McClelland, J. 1986. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. MIT Press.

Stewart, T., and Eliasmith, C. 2009. Compositionality and biologically plausible models. In Werning, M.; Hinzen, W.; and Machery, E., eds., *Oxford Handbook of Compositionality*. Oxford, UK: Oxford University Press.