Continuous Learning of Action and State Spaces (CLASS)

Paul Robertson PAUL.ROBERTSON@DOLLABS.COM
Dynamic Object Langauge Labs Inc. Lexington, MA 02421, USA
Olivier L. Georgeon OGEORGEON@UNIV-CATHOLYON.FR
UR CONFLUENCE, Sciences et Humanits - UCLy, LIRIS CNRS UMR5205, Lyon, France

Editors: Minsky, H. and Robertson, P. and Georgeon, O. L. and Minsky, M. and Shaoul, C.

Abstract

We present a novel approach to state space discretization for constructivist and reinforcement learning.

Constructivist and reinforcement learning approaches are often characterized by simple grids. The manner in which the state space is discretized is the source of many problems for both constructivist and reinforcement learning approaches. The problems can roughly be divided into two categories: (1) wiring too much domain information into the solution, and (2) requiring massive storage to represent the state space (such as Q-tables. The problems relate to (1) the non generality arising from wiring domain information into the solution, and (2) non scalability of the approach to useful domains involving high dimensional state spaces. Another important limitation is that high dimensional state spaces require a massive number of learning trials.

We present a new approach that builds upon ideas from place cells and cognitive maps.

Keywords: Cognitive maps, Place cells, Reinforcement learning, Constructivist learning, Contexts

1. Introduction

There are many impressive examples of what various variants of reinforcement learning and constructivist learning can do. Especially with the recent affordable massive parallelism brought by GPU's and the availability of robot or game simulators (see OpenAI.com; gazebosim.org).

Without wishing to diminish the value of the recent triumphs with Deep Neural Networks or Deep Reinforcement Learning, the impressiveness fades when we look closely at how the results are obtained.

In a research blog about Alpha-Go , see Silver and Hassabis (2017), Silver and Hassabis State: "Over the course of millions of AlphaGo vs AlphaGo games, the system progressively learned the game of Go from scratch, accumulating thousands of years of human knowledge during a period of just a few days. AlphaGo Zero also discovered new knowledge, developing unconventional strategies and creative new moves that echoed and surpassed the novel techniques it played in the games against Lee Sedol and Ke Jie."

Whereas there is no denying that this is a landmark achievement worthy of the fanfare that it has received, we note that no human player has ever played 'millions of games' against any opponent, self or otherwise.

We wish for our learning systems to learn rapidly, without baking the rules of the game into the learner, where rapidly, is with respect to a normal physical lifetime. One author of this article became a strong club chess player after playing against other strong club players approximately once a week over a small number of years (hence a few hundred games). He learned other feats like playing tennis over a short period of time and with very limited number of trials. This kind of learning is the goal.

If we were to limit learning to being done in the real world, with real robots, in real-time and before the robot breaks, we would constrain the goal so as to focus on achieving realistic developmental learning. If an AI could learn to play GO well using no more training games than Lee Sedol had played in his life, that would be something that could be learned in a flash in simulation.

There is no doubt that Deep learning, that may require 10^9 training examples, or Deep Reinforcement Learning, that may require millions of episodes, will find useful applications. Carefully limiting the state-space size will yield specialized uses, but for developing a generalized non-parametric developmental learning capability, there remain many obstacles.

1.0.1. Research Assumptions

Natural systems deal with very large state-spaces, learn effectively with a small number of training episodes, and are self-motivated to learn. It is towards this, that this article is motivated. Massive parallelism, such as is available in GPUs is in scope, massive memory capacity is in scope. Massive annotated training corpora are not in scope, nor is baking the solution into the learner, neither is a massive number of episodes that would be infeasible for a physical living agent to perform. State space limitation is not out of scope, but it should be learned rather than backed in to the problem description, the strategy of dimensionality reduction should be learned, if it is necessary, on a problem by problem basis and not determined by human AI experts. There will be parts of the state space that represents things that are happening in the world unrelated to any actions taken by the robot and which can safely be ignored, there will be parts of the state space that are not observable (POMDP not an MDP). A generalized learner must deal with real-world issues in which not all changes are the result of the robot's action. Before these generalized learners can be released into the real world, they must deal with notions like attention and attribution.

1.0.2. Prior Work

Generally, the number of actions is limited, they tend to be discrete, and the state space is limited Q-learning [Watkins (1989)] and constructivist schema type learning [Drescher (1991)] both depend upon representing the state space and the action space. This usually involves discretizing the state space and the action space too. Fine discretization leads to massive data structures and the need for an unreasonably large number of learning trials, whereas a rough discretization lacks the precision to capture important differences between places in the state space that map to a single entry whereas it would be better if it were finer.

Being too rough, has a negative impact on the learning because it groups parts of the state space that should naturally be separate. This adds stochasticity of the problem demain in a bad way. If in state x the action a can lead to two outcomes r_0 or r_1 it can be because

the action is stochastic in its nature or because two distinct states have been captured as a single state and the if the state had been encoded as two separate states, the actions would be deterministic, or at least less stochastic. If r_0 1% of the time leads to success whereas r_1 leads to disaster 99% of the time, the Bellman equation [Bellman (1952)] will learn to avoid this action. While Deep Q-learning [Mnih and Silver (2015)] somewhat alleviates the problem by using deep-learning to estimate Q values, it brings other weaknesses that we discus later. Various approaches to representing continuous continuous state-space models [Raghu (2017)] have been proposed that use dueling double-deep Q-Networks (DDQN's) to represent continuous state spaces.

For a general learning approach, we do not want to encode the details of the learning task into the learning mechanism. We would like to be able to learn both from free exploration (constructivist) and learn given a reward (reinforcement learning).

Reinforcement Learning converges on an optimal policy for an MDP. An MDP policy can be represented as a table. A table lookup will give for every position in the state space the optimal action to perform.

The size of this table grows as the product of the number of discretized values of each dimension and the action dimension (how many actions can be taken at any point?) It would be infinite for continuous dimensions, but this can be discretized to avoid that problem while introducing others. Learning the optimal policy is achieved as a dynamic programming problem driven by the Bellman Equation.

1.0.3. Hippocampal Place Cells

The Hippocampus has long been known to play an important role in learning and episodic memory. More recently, the discovery of place cells has suggested that the primary function of the Hippocampus is to provide a cognitive map. More recently, work described by David Smith and Sheri Mizumori (see Smith and Mizumori (2006)) appears to show a more generalized role of place cells in general context processing. Even if the role was limited to cognitive map functions, the manner in which places are mapped to the Hippocampus remains unknown. How does Hippocampal area get allocated to physical spaces?

Too little is known about the Hippocampus to attempt a simulation of its function, but its role in providing a cognitive map function suggests a very necessary function for learning to navigate in spaces. Furthermore the more generalized function that the hippocampus serves as a generalized context function may be the key to solving one of reinforcement learning and other constructivist learning approaches most serious problems, that of scaling up to interesting sized problems.

The approach described below, for which we have an early prototype, appears to provide not only support for cognitive mapping but also for learning to discretize the state space as is required by what is learned. As the space is navigated, the cognitive map of that space is expanded to accommodate the newly learned structures. The same mechanism provides state-space discretization even without any physical world correspondance. Because the state space grows only as it is needed, a learning system can represent the context space compactly which grows linearly with the number of contexts. High dimensional state spaces, which are normal in animal brains, and which cause state spaces to explode, no longer pose a scaling problem since each visited state is represented as a point in N-Dimensional space,

which clusters with other points to form clusters representing contexts of which are only sufficient in number to support the fidelity required to represent the learning.

1.1. Overview of the approach

Rather than representing the immense table (Q-table) directly we frame the problem as contexts which are learned from values in the state.

Contexts are built from states that have occurred in the learning system. Initially, there is nothing. As points in the state space are observed, they are added to a growing collection of points that are continually being observed. When an action is taken, observations indicate a state transition. $< oldstate, action, newstate >$ Represent a point in a space that are clustered. The clusters grow, divide, and coalesce as new points are found.

When the system observes a state, it maps to the cluster to which it belongs. From this the actions and the new states are can be read, just as they are in a conventional Q-table.

With this approach continuous state spaces map naturally to clusters whose boundaries represent where an action will have a different outcome. The precise boundaries of the important points in a continuous state dimension can thus be learned in parallel with the learning of the policy. A lot of clusters will be formed where necessary to capture the important parts of the state space whereas almost no clusters are generated where nothing interesting happens.

For any state, we can know where we are in a cluster and how close we are to a cluster boundary.

This approach can be used equally well for goal-less exploration for schema learning and reward driven learning for reinforcement learning. The key computational part of CLASS is the incremental clustering algorithm which can be computed, using a GPU, to track a large number of points.

Initially we kept all points, which at some point takes up a lot of memory and takes longer, even for a GPU to compute. To solve this problem, when the number of saved states grows beyond a fixed number, we replace two points that are close together with a single point, set a position between the original two and which is given a count of 2. Later, in general, the points can be collapsed to form higher order points. As such we can place a parametric upper bound on the number of points stored. with some loss of maximum attainable precision.

We are experimenting with this algorithm in the context of undirected skill learning as well as rewarded learning on a Turtlebot3 in a labyrinth with colored walls. Results of these experiments will be published in an upcoming article.

In the following sections, we describe the clustering algorithm and how that allows non-convex inter-tangled clusters to be learned incrementally.

1.2. Contexts

Studies of human perception suggest that we always interpret images within a context that defines our prior expectations about what we expect to see. Psychologists call this "priming". This reaches an extreme form in the case of model-based image analysis, in which programs "hallucinate" (see Clowes (1971)) one of a small set of models onto images.

Typically, the human programmer defines the "context" by providing *a-priori* the (small) set of models that can be matched.

The need for contexts to manage the diversity of the world is no less important outside of image understanding. AI has long understood the importance of contexts. In 1975 Minsky introduced the notion of *frames* (see Minsky (1975)) which was essentially an approach to contexts. Frames have been used extensively in AI research, especially for natural language. Riseman's Schemas (see Draper et al. (1988)) was a similar idea specifically for Computer Vision.

The algorithm described in this paper builds upon previous development of a system called GRAVA (for Grounded Reflective Adaptive Vision Architecture), that segments and labels aerial images in a way that attempts to mimic the competence of a human expert [Robertson (2000)].

2. Principle Component Decomposition

The states achieved by the learning system, such as a robot, provide multiple positive examples of a structure that we wish to model. The structures in question have one or more dimensions, and the available observations provide examples of the structure that enable us to model the location within the appropriate multidimensional space. One way of doing this is to model the structures as a probability distribution function (PDF). The natures of the structures may be very different.

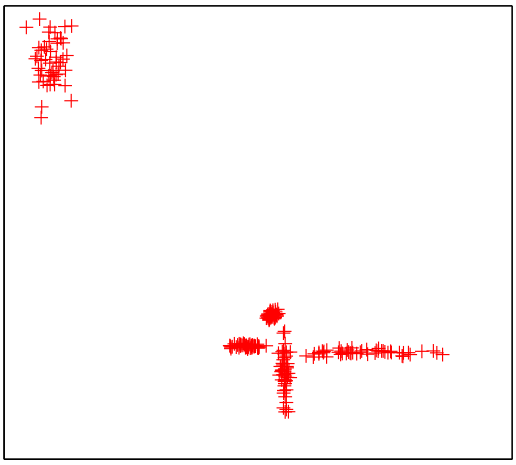

Figure 1: The Need for Decomposition

Consider two dimensional[1] space and the collection of positive examples shown in Figure 1. Given a set of data points it is possible to find a mean point and the principle and secondary eigenvectors. Unfortunately the resulting model is unusably crude since most of

1. We provide two dimensional examples in this paper because they can be graphically illustrated on a two dimensional medium. In practice the number of dimensions is far larger than two.

the points that it generates are not suggested by the data. The problem gets successively worse for higher dimensional spaces.

We experimented with a number of standard clustering approaches such as K-means [R.O.Duda et al. (2001)] and K-medoids but our use requires that the clustering be performed automatically on dynamically collected data and we needed an algorithm that was non-parametric. In particular we should not have to specify the number of clusters or to specify typical values. We need to have the number of clusters and their composition be determined automatically. Furthermore, since natural clusters are often non-convex we need an algorithm that can separate intertwined non-convex clusters. The algorithm described below depends solely on the notion of minimal description length and as such requires no parameters.

Principle component decomposition is the interpretation of a set of data points into the component collections (five in this example) by analyzing the principle components of the interpretation space.

The algorithm builds upon two earlier works. The first is a classification program developed by Wallace. Wallace's [Wallace (1990)] program (SNOB) worked by finding a minimum message length (MML) description of a set of points. The second is the practice of using principle component analysis (see Jackson (1991)) to reduce the dimensionality of high dimensional problems so that the separate populations can be modeled.

Our algorithm applies principle component analysis recursively in order to separate the collection into successively smaller clusters. At each point the criterion for separating a population is that it reduces the global description length of the original population.

Below we present our algorithm for producing such principle component decompositions.

2.1. A Statistical Model for Clusters

Given an n-dimensional space S_n containing m points. we can interpret the points in this space as being:

1. Unrelated points.
2. All members of a single cluster.
3. Grouped into a number of cluster.

A model has a shorter description length if it reduces the amount of uncertainty about the values of features. The best interpretation of the data points that constitute the collected state observations, therefore, is the interpretation that reduces the uncertainty about where the data points appear in the multidimensional space.

The entropy of the collection data points in a data set is given by:

$$H = -\sum_{d \in S_n} P(d) log_2 P(d) \tag{1}$$

The lower bound MDL of a description that represents all of the points in the S_n is given by:

$$DL = -\sum_{d \in S_n} log_2 P(d) \tag{2}$$

In order to compute this theoretical description length, it is necessary to know the PDF for points in S_n. A data set doesn't specify every possible point in the space. It provides a collection of *representative* points in the space. The job of interpreting the data set involves modeling the PDF. There are many choices for modeling a PDF. One model that is simple, predictive, and which often pertains to naturally occurring distributions is the Gaussian.

The description of a Gaussian model consists of a mean and variance of the distribution $< \mu, \sigma^2 >$. For a set of points the Gaussian model can be fitted simply by computing the mean μ and the variance σ^2. Given this characterization, for any point d we can compute the probability $P(d)$ as follows:

$$\begin{aligned} P(d) = erf\left(\frac{(pos(d) - \mu_n + \epsilon/2)}{\sigma_n}\right) \\ -erf\left(\frac{(pos(d) - \mu_n - \epsilon/2)}{\sigma_n}\right) \end{aligned} \tag{3}$$

where $pos(d)$ is the position of the point d, ϵ is the position resolution, μ_n is the n-dimensional mean, σ_n^2 is the n-dimensional variance, and erf() is the error function.

The choice of whether to consider the points in the data set as (1) unrelated individual points, (2) all members of the same model, or (3) divided into groups each of which is modeled, is to select the choice that yields the minimum description length.

The interpretation task can therefore be characterized as dividing the data points in S_n into n proper subsets $C_{i,n}$ such that:

$$S_n = \bigcup_{i=1}^{n} C_i \tag{4}$$

The MDL is

$$\begin{aligned} arg\min_{C_{1,n}} \sum_{i=1}^{n} \Bigg\{ \left(\sum_{d \in C_i} -log_2 P(d|C_i) - log_2 P(d \in C_i) \right) \\ + ddl(C_i) \Bigg\} \end{aligned} \tag{5}$$

where $ddl(C_i)$ is the description length of the distribution used to model C_i. The description of a point is divided into two parts. The first part identifies its position in the space ($-log_2 P(d|C_i)$) and the second part identifies to which collection it belongs ($-log_2 P(d \in C_i)$).

The statistical models chosen for C_i determine the size of the point descriptions. In order to specify the position of a point we choose a resolution ϵ to be used uniformly since otherwise a point can have an arbitrary precision and its representation would be arbitrarily large.

If the representation of a collection includes its mean position μ, the positions of the points in the collection can be described as distances δ from the mean. Figure 2 shows the representation of a point within a collection C_i as an n-dimensional mean ($n = 2$ in this example) and a n-dimensional displacement. So any point d can be described as:

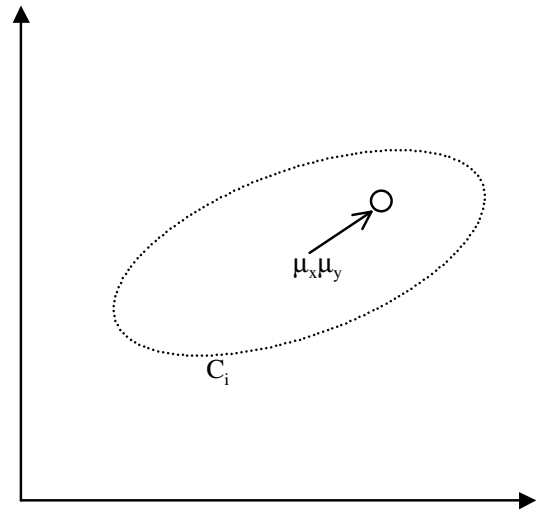

Figure 2: Representation of a Point in a Collection

$$\mu + \delta - \frac{\epsilon}{2} \leq d \leq \mu + \delta + \frac{\epsilon}{2} \tag{6}$$

Given the original set of points it is possible to reconstruct the statistical model that was used to represent it. So to communicate the collections, all that is required is the mean position represented to an accuracy of ϵ. The points are represented as a description of which collection they belong to and the offset from the mean: $< C_i, \delta >$.

If all points are considered to be separate collections, each of a single point, the size of their offset will be 0 since all points reside at the collections mean. Since clusters and points have a one to one relationship, and the point description contains no information other than the collection assignment, the representation only requires the positions of the individual points in the collection. Collections that are represented as individual points in this way have no predictive value.

If all points are considered to be members of a single collection the representation of a point doesn't need to identify to which collection it belongs because there is only one collection.

As the data points in a data set are divided up into smaller collections the description length of the individual points is reduced if the distribution that characterizes the collection is more predictive about the position of its component points than the distribution for the entire data set was. Any suitable statistical distribution can be chosen for a collection.

2.2. Algorithm for Decomposition

Having defined the criteria for an optimal division of the data points into separate models we are left with the task of defining an effective procedure for achieving such a division. To accomplish this we developed an efficient algorithm that approximates a solution to Equation 4.

Our algorithm, which we call "principle component decomposition" (PCD), attempts to divide the data by searching for dividing hyper planes tangental to the eigenvectors of the data. The idea behind the algorithm is that the principle eigenvectors represent the dimensions with the greatest spread. The spread can be caused by a single phenomenon with a large variance, or it can be caused by more than one phenomenon distributed throughout the space. To distinguish these two cases we compute the entropy of the data points as a whole and then we compute the sum of the entropies of the two collections formed by dividing the data points into two collections with a hyper plane perpendicular to the eigenvector[2]. We do this for all possible cut points along the eigenvector. If all sums of divided collections yield a higher description length than the original combined collection the collection is not divided, otherwise the collection is divided at the place that yields the minimum description length.

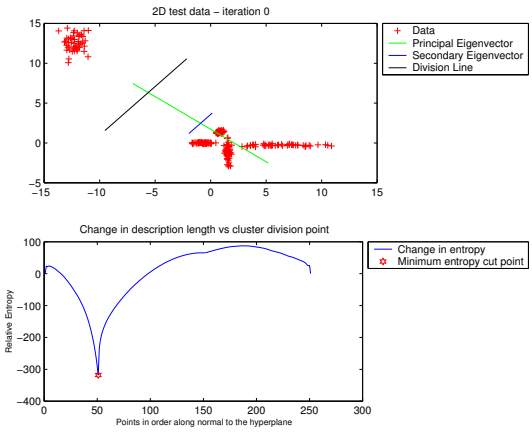

Figure 3: Dividing the Data Points to Reduce Description Length

Figure 3 shows the 2-dimensional data introduced earlier. There are two eigenvectors. The lengths of the principal and secondary lines are the square root of the corresponding eigenvalues[3].

The hyper plane that is used for cutting the data is perpendicular to the principal eigenvector[4]. The graph below shows the change in the total description lengths resulting from cutting the collection at any point along the eigenvector.

When the change is greater than zero, cutting makes the description length larger. In this case the total description length is significantly reduced by cutting the collection at the point where the "division" line is drawn. This point can be seen as the minimum point in the entropy curve.

This procedure is repeated for each eigenvector of the collection starting from the eigenvector that corresponds to the largest eigenvalue until either a division occurs or until all eigenvectors have been tried. Once a collection has been split the algorithm is applied to

2. A 2-dimensional hyper plane is a line.
3. The eigenvectors are computed from a co-variance matrix so the eigenvalues are variances and the square root of the eigenvalues are standard deviations.
4. A 2-dimensional hyper plane is a line.

each of the newly divided collections. Eventually there are no collections of points that split. The algorithm consists of two parts CHOP and MERGE.

CHOP looks for places to divide a collection of data points into two collections by finding a dividing hyper plane. CHOP thus produces two collections that have the property that if collection C_0 is divided into C_1 and C_2, $C_0 = \cup\{C_1C_2\}$, and $DL(C_0) > DL(C_1) + DL(C_2)$. MERGE finds two collections of data points (say C_1 and C_2) that have the property that $DL(\cup\{C_1C_2\}) < DL(C_1) + DL(C_2)$. If the collection of data points is non-convex CHOP can cause some points to become separated from the collection to which they naturally belong. MERGE re-associates points severed in this way with their natural collection. The advantage of this approach is that it is possible to construct non-convex collections of data points.

First we describe the algorithm for $CHOP(S)$ that chops the collection into separate collections.

$CHOP(S)$:

1. S is a set of n-dimensional data points. Let $\bar{m}$ be the mean and C be the co-variance matrix.

2. Let $v_1 \ldots v_n$ and $\lambda_1 \ldots \lambda_n$ be the eigenvectors and corresponding eigenvalues, respectively, sorted into decreasing order of eigenvalue.

3. For each eigenvector v_i starting with v_1 (the one with the largest eigenvalue—the principle eigenvector), search for the best place to cut the data points into two collections as follows:

   (a) Establish the cutting hyper plane. The cutting hyper plane is the plane that is perpendicular to the eigenvector v_i. We arbitrarily choose the hyper plane that passes through the mean $\bar{m}$.

   $$\begin{aligned} n &= \bar{m}^T v_i \\ \bar{r} v_i &= n \end{aligned} \tag{7}$$

   where $\bar{r}$ is a point specified as a row matrix.

   This is the perpendicular form of the equation of a hyper plane. This representation is convenient because it permits fast calculation of the distance of a point from the hyper plane. For any point d the distance from the plane in equation 7 is given by $n - dv_i$.

   (b) Sort the points in S in order of distance from the cutting hyper plane. Since the hyper plane cuts through the mean, approximately half of the points will be on one side of the hyper plane, with the rest on the other side. Approximately half of the points, therefore, will have a negative distance from the plane. The distance is not the absolute distance from the plane, it is how far to move along the normal to the hyperplane to reach the plane in the direction of v_i.

   (c) Let A be the sorted list of data points.

   (d) Let B be an empty list.

(e) Let the $cutPoint = 0$ and $position = 0$

(f) Let $minDL = DL(A)$ the description length of the entire set of data points.

(g) Now we simulate sliding the cutting hyperplane along the eigenvector from one end of the set of data points to the other, by taking points one at a time from A, putting them into B, and computing the description length of the two collections as follows:

For each point d_j in A do:

i. Remove d_j from A.

ii. Add d_j to B.

iii. Increment the position ($position = position + 1$).

iv. Compute the new description length as $newDL = DL(A) + DL(B)$.

v. If $newDL < minDL$ set $minDL = newDL$ and $cutPoint = position$.

(h) If $cutPoint > 0$ divide the data points S into two collections S_1, and S_2 at the position indicated by $cutPoint$. Then recursively apply $CHOP$ to both of the sub-collections to see if further chopping can be performed. Finally return the complete list of chopped collections:
$return\ append(CHOP(S_1), CHOP(S_2))$

4. At this point, all of the eigenvectors of S have been searched for chop points, and none have been found. The data points cannot be represented with a smaller description length by chopping along an eigenvector so return the list of collections as the single collection S:

   $return\ list(S)$

The nature of the way the collections are divided up results in some groups of data points being divided unnecessarily. This can be seen in the second and third iteration for this example data.

In the second iteration (Figure 4 top) one point is chopped off the right most collection. In the third iteration (Figure 4 top) two points are chopped off.

Later, in the fifth and sixth iteration (Figure 5), the one point on the left, and the two points on the right, are completely severed so as to be small disembodied collections of points (one point and two points respectively in this example). These accidentally severed fragments are corrected in phase two of the algorithm—the merge phase.

In phase two of the algorithm (MERGE) pairs of collections of points are checked to see if the description length would be reduced if they were to be merged. If the description length would be reduced by merging them they are merged. This is repeated until no more useful merges can be found.

$MERGE(C)$:

If C is a collection of clusters resulting from the application of $CHOP$ to the original data, the merge phase proceeds as follows:

1. For each pair of clusters $C_i \in C$ and $C_j \in C$ check to see if the description length can be reduced by merging them ($DL(C_i) + DL(C_j) > DL(C_i \cup C_j)$).

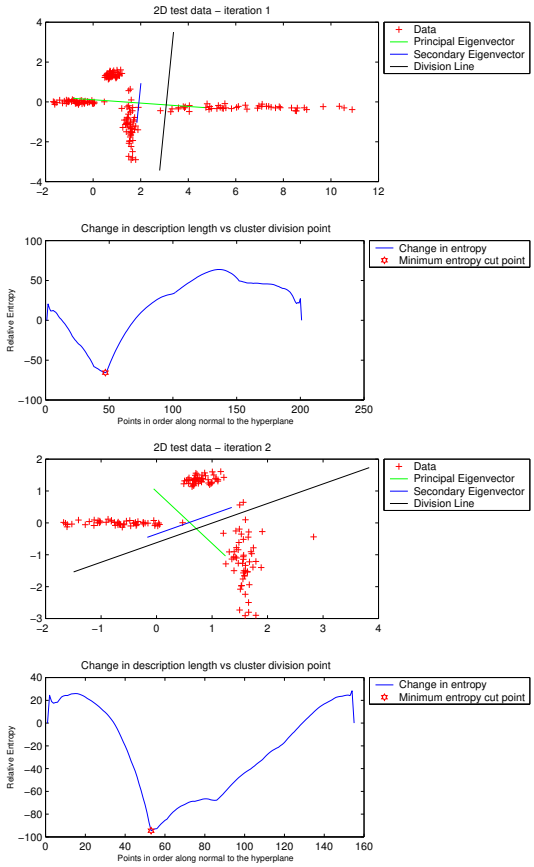

Figure 4: Accidentally Severed Points

2. If no merge candidates are identified in (1) the merge phase is complete.

3. Produce C' by replacing each pair of mergeable clusters with their merged union.

4. Repeat steps 1, 2, and 3 on the new set of clusters.

After 16 iterations the division of the data points in to separate collections is complete. Figure 6 shows the final result of decomposition.
The PCD algorithm described above has a number of interesting characteristics:

1. PCD produces a structural description of the data points that is an approximation to a global MDL description of the points.

2. Each remaining collection of points can be represented efficiently by the statistical model chosen for it since if the collection could not be represented well it would have been divided.

3. Each collection is a good candidate for PCA modeling because of (2).

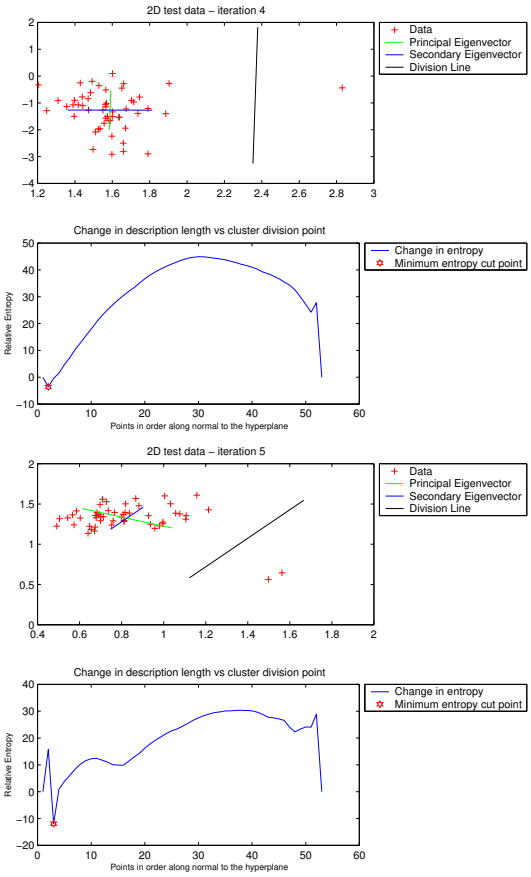

Figure 5: Accidentally Severed Points

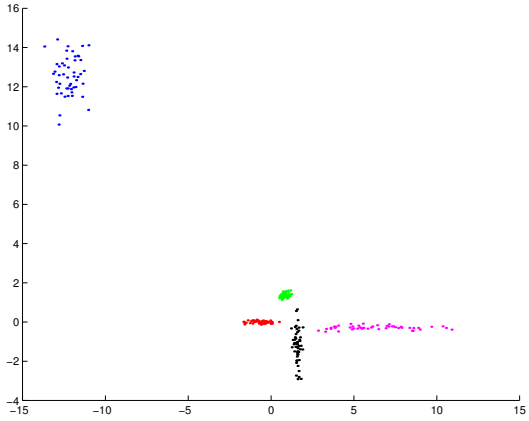

Figure 6: Example Result of Principal Component Decomposition

4. The algorithm can be implemented efficiently and can produce good decompositions very quickly using a GPU implementation. The number of "chop" and "merge" op-

erations that are performed in producing a decomposition is very small compared to the number of points.

5. The algorithm can produce non-convex collections.

The final point (5) is an interesting feature of the algorithm that is not obvious from the example given above. Non-convex collections cannot be disentangled by using the "chop" operation alone but inclusion of the "merge" operation allows two convex collections to be joined so as to produce a non-convex merged collection.

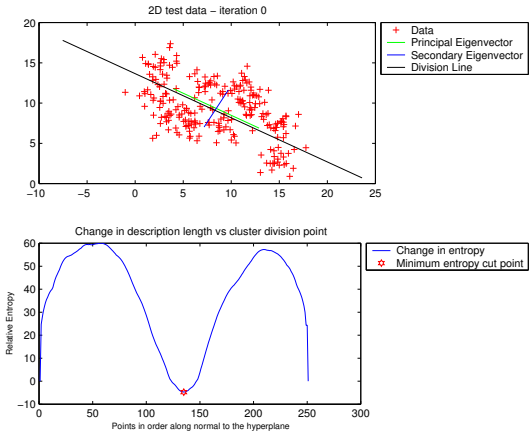

Figure 7: Intertwined Non-Convex Shapes

To demonstrate this capability we generated a set of data points by picking points randomly along two interlocking 'C' shapes in a ying yang configuration. Even though the data is quite dense and intertwined the algorithm manages to "chop" it apart and then "merge" the severed parts back together. Figure 7 shows the first iteration on the data.

The second and third iterations chop the data down further as shown in Figure 8.

Later iterations merge the severed portions back into their rightful places as shown in Figure 9.

The final decomposition of the data is shown in Figure 10.

It should be noted that separating clusters in a 2D space is harder than in a higher dimensional space because higher dimensional spaces are naturally sparser. This is a similar observation to that used to good effect in support vector machines. Sometimes, high dimensionality is a benefit and not a problem.

3. Conclusion

We have developed an approach to dynamically constructing a "cognitive map" by decomposing complex models into collections of simpler models. This forms a backbone mechanism for interpreting learning policies. If these cognitive maps resemble the Hippocampus in function, it would suggest that learning depends upon a cognitive map, not only at for navigation in physical space, but at all cognitive levels, however abstract, and even without physical location aspects, a cognitive map for conceptual spaces.

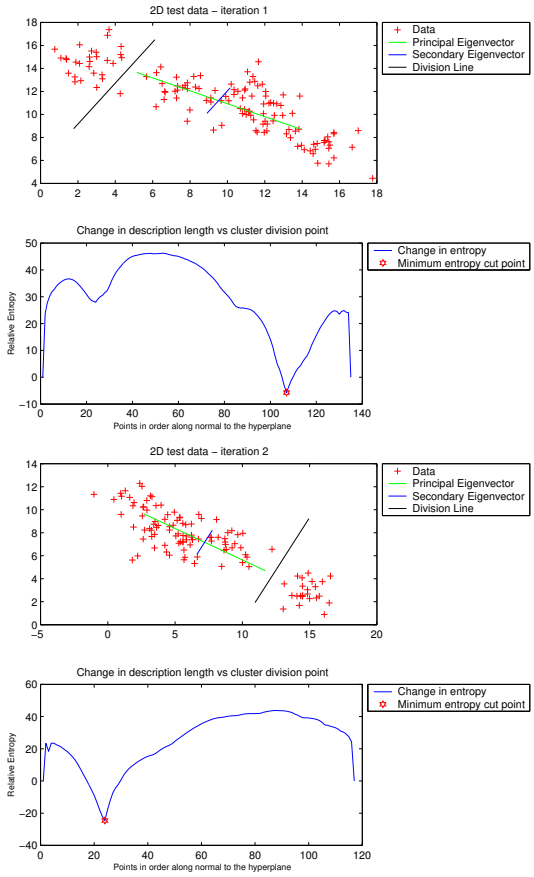

Figure 8: Curved example: CHOP

The algorithm has some important features:

1. The algorithm supports non-convex shapes.
2. The algorithm uses the MDL criteria for interpretation.
3. The algorithm doesn't over fit. The algorithm doesn't try to circumscribe a set of data points. It simply tries to separate collections of points by finding the best place to cut.
4. The algorithm is fast because it only searches for cut points along eigenvector dimensions.
5. The algorithm is non-parametric which removes one more barrier to automation.

4. Acknowledgements

Effort sponsored in part by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Material Command, USAF, under agreement

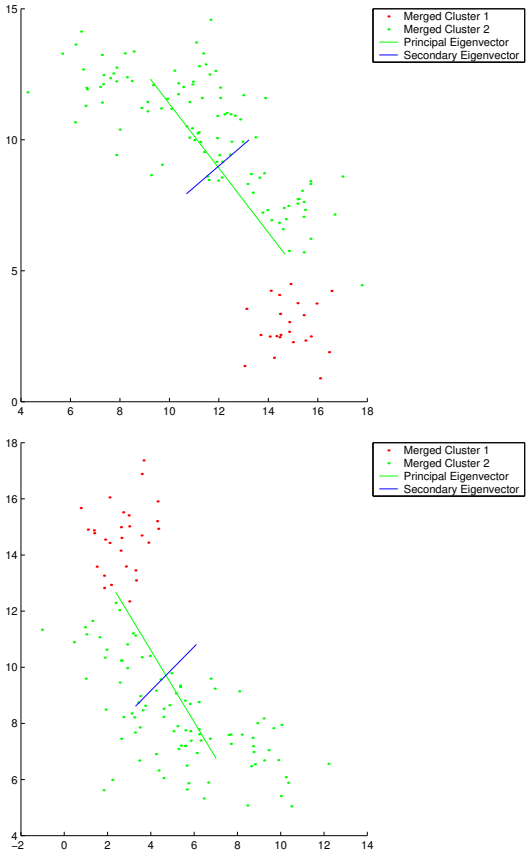

Figure 9: Curved example: MERGE

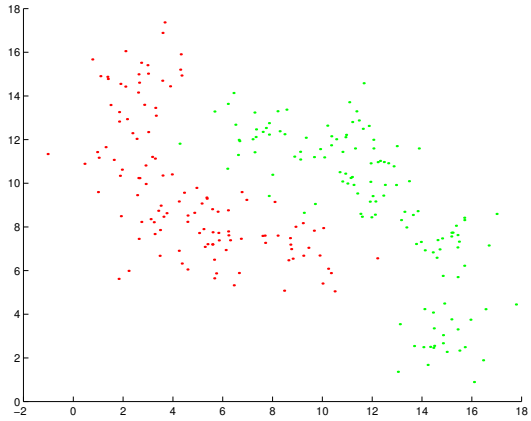

Figure 10: Final Decomposition of Example Non-Convex Data

number F30602-98-0056. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), the Air Force Research Laboratory, or the U.S. Government.

References

Richard Bellman. On the theory of dynamic programming. *Proc Natl Acad Sci USA*, 38(8):716–719, August 1952.

M. Clowes. On seeing things. *Artificial Intelligence*, 2:79–116, 1971.

B. Draper, R. Collins, J. Brolio, A. Hansen, and E. Riseman. The schema system. Technical Report COINS TR88-76, Computer and Information Science, Univ. Massachusetts at Amherst, 1988.

Gary L. Drescher. *Made-Up Minds A Constructivist Approach to Artificial Intelligence*. MIT Press, 1991. ISBN 9780262041201.

gazebosim.org. Gazebo. URL http://gazebosim.org.

J.E. Jackson. *A user's guide to Principal Components*. John Wiley and Sons, New York, 1991.

M. Minsky. A framework for representing knowledge. In P. H. Winston, editor, *The Psychology of Computer Vision*. McGraw-Hill, New York, 1975.

V. Mnih and D. et. al. Silver. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

OpenAI.com. Gym. URL https://gym.openai.com/.

A. et.al. Raghu. Continuous state-space models for optimal sepsis treatment: a deep reinforcement learning approach. In *Proceedings of Machine Learning for Healthcare*, 2017.

P. Robertson. An architecture for self-adaptation and its application to aerial image understanding. In R. Laddaga P. Robertson and H. Shrobe, editors, *Self-Adaptive Software*, pages 199–223. Springer-Verlag, 2000.

R.O.Duda, P.E. Hart, and D.G. Stork. *Pattern Classification*. John Wiley and Sons, 2001.

D. Silver and D. Hassabis. Alphago zero: Starting from scratch. 2017. URL https://deepmind.com/blog/article/alphago-zero-starting-scratch.

D.M. Smith and S.J.Y Mizumori. Hippocampal place cells, context, and episodic memory. *Hippocampus*, 16(9):716–729, August 2006. URL https://doi.org/10.1002/hipo.20208.

C.S. Wallace. Classification by minimum-message-length inference. In G. Goos and J. Hartmanis, editors, *Advances in Computing and Information–ICCI'90*, pages 72–81. Springer-Verlag, 1990.

C.J. Watkins. *Learning From Delayed Rewards*. PhD thesis, University of Cambridge, 1989.