Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence

Learning Integrated Symbolic and Continuous Action Models for Continuous Domains

Joseph Z. Xu and John E. Laird
Computer Science and Engineering, University of Michigan
2260 Hayward Street, Ann Arbor, MI 48109-2121 USA
{jzxu, laird}@umich.edu

Abstract

Long-living autonomous agents must be able to learn to perform competently in novel environments. One important aspect of competence is the ability to plan, which entails the ability to learn models of the agent's own actions and their effects on the environment. In this paper we describe an approach to learn action models of environments with continuous-valued spatial states and realistic physics consisting of multiple interacting rigid objects. In such environments, we hypothesize that objects exhibit multiple qualitatively distinct behaviors we call modes, conditioned on their spatial relationships to each other. We argue that action models that explicitly represent these modes using a combination of symbolic spatial relationships and continuous metric information learn faster, generalize better, and make more accurate predictions than models that only use metric information. We present a method to learn action models with piecewise linear modes conditioned on a combination of first order Horn clauses that test symbolic spatial predicates and continuous classifiers. We empirically demonstrate that our method learns more accurate and more general models of a physics simulation than a method that learns a single function (locally weighted regression).

Introduction

We are interested in the problem of developing long-living, embodied agents that can adapt to a variety of novel environments and tasks. One way an agent can gain competence in a novel environment is to learn to plan in it. This requires that the agent have an internal model of how its actions change the environment. We call such a model an action model.

Many real-world environments can be characterized as collections of discrete, interacting objects with continuous properties embedded in a two or three dimensional space. The action model for such an environment can be described as a continuous function $M: \boldsymbol{s}_t, \boldsymbol{u}_t \rightarrow \boldsymbol{s}_{t+1}$, where $\boldsymbol{s}_t, \boldsymbol{s}_{t+1} \in \mathbb{R}^n$ are the environment state at time steps t and $t+1$, and $\boldsymbol{u}_t \in \mathbb{R}^m$ is the agent's action at time t. Common methods for learning continuous action models include locally weighted regression (Atkeson, Moore, and Schaal 1997), Gaussian processes, and support vector regression (Nguyen-Tuong and Peters 2011). These methods all assume that the model function is smooth over its input space, and rely on this smoothness assumption for generalization, i.e. the behavior of the model is assumed to be similar in the neighborhood of each training example.

However, in environments with multiple interacting objects, the model function often changes abruptly and discontinuously at boundary conditions. Consider a world with a free moving ball and a fixed box. The ball's velocity will change smoothly when it is flying in the air, but it will change direction instantaneously when it bounces against the box. Whether the ball flies or bounces is determined by whether it touches the box and not by the absolute positions of the ball and the box. In the six-dimensional space of the ball's and box's (x, y, z) positions, the points with bouncing behavior forms a set of disjoint hyper-planes instead of a bulbous neighborhood. In these types of environments, the smoothness assumption fails, and generalizations based on that assumption are invalid.

We hypothesize that in environments with multiple interacting objects, it is the relationships between the objects that determine how they behave, rather than their absolute positions in a coordinate system. Furthermore, behaviors tend to vary smoothly when certain relationships hold but change abruptly when relationships change. The action model should therefore be described by a set of individually smooth functions that cover disjoint regions in the relation space instead of a single global smooth function. Call these individual functions *modes*.

In this paper, we present a method to learn action models that are composed of multiple modes. Our system

Copyright © 2013, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

automatically identifies new modes from unlabeled training data using Expectation Maximization (EM) and learns a classifier that predicts which mode the environment exhibits based on spatial relationships between objects. We compare its performance in a simulated environment with realistic physics to that of a global model learning approach (locally weighted regression) and show that our modal approach has better generalization and prediction accuracy.

Related Work

The idea of learning models with multiple modes is not new. Toussaint and Vijayakumar (2005) learned multiple linear models with Expectation Maximization and distinguished between modes with a product-of-sigmoids classifier. Many approaches to a similar problem called "hybrid system identification" have been studied in the control literature (Paoletti et. al. 2007), using a variety of clustering techniques, including EM. There has also been work on learning piecewise linear functions as the leaves of decision trees known as model trees (Potts 2005). These approaches all associate modes with continuous regions of the metric state space, which generalizes poorly when behaviors are conditioned on relationships between objects. Our approach conditions modes on first-order Horn clauses that test spatial relationships and thus performs better in those cases.

Our system uses similar combined symbolic/continuous representations as qualitative and spatial reasoning systems such as FROB (Forbus 1980), CLOCK (Forbus, Nielson, Faltings 1991), and QSIM (Kuipers 1994). The focus of those pieces of work was on symbolic reasoning with hand-coded qualitative representations, whereas we focus on learning and our use of symbolic information is in service of making better numeric predictions.

Troha and Bratko (2011) learn a qualitative model of a robot pushing an object and use it for motion planning. Their model differs from ours in that it only makes predictions about directions of change for individual dimensions of the continuous state, rather than actual numeric values.

Approach

We assume that the environment is deterministic and fully observable, progresses in fixed-length time steps, and is composed of discrete objects. The environment reports its state to the agent as a vector of continuous properties. Each object has a position (x, y, z), rotation $(roll, pitch, yaw)$, and scaling factor (sx, sy, sz), as well as a 3D geometry defined by a convex hull. These properties have a fixed interpretation in the system. Other arbitrary continuous properties can also be included in the state vector, such as the (x, y, z) velocity of the object. Our system has no a priori interpretation for these properties. We also require the environment to provide a type for each object. Our system doesn't have special knowledge about types, but it does assume that objects of the same type behave identically, which aids in generalization.

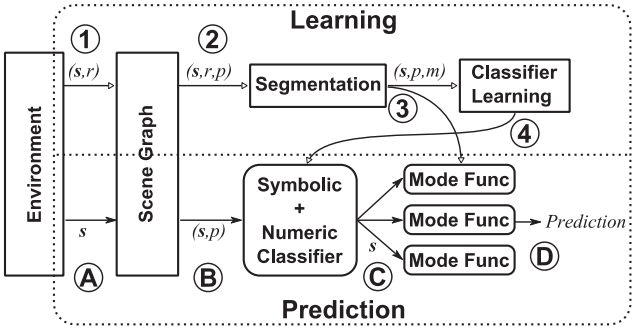


Figure 1. Overview of the learning and prediction algorithms.

We assume for simplicity that the action models for the individual dimensions of the state vector are independent, and decompose the problem of learning the model $M: \boldsymbol{s}_t, \boldsymbol{u}_t \rightarrow \boldsymbol{s}_{t+1}$ into learning individual models $M: \boldsymbol{s}_t, \boldsymbol{u}_t \rightarrow r_t$ for each dimension $r_t \in \boldsymbol{s}_{t+1}$. Furthermore, we treat the agent's output $\boldsymbol{u}$ just like any other dimension in $\boldsymbol{s}$, so the problem further reduces to learning the function $M: \boldsymbol{s} \rightarrow r$.

Figure 1 gives an overview of the learning and prediction algorithms. The learning algorithm (upper half of Figure 1) segments the training examples into modes and learns a classifier that associates modes with initial states. The prediction algorithm (lower half) uses the learned classifier to predict the mode of a state and then uses the mode's function to predict the value of the modeled dimension.

Learning

Given a sequence of observations $(\boldsymbol{s}_1, r_1), (\boldsymbol{s}_2, r_2), \ldots$, (① in Figure 1) our system combines the continuous state vector and object geometries into a 3D scene called the *scene graph*. It is then able to extract spatial relationships from the scene graph, such as if $intersect(A, B)$ is true. The set of spatial relationships tested is hard-coded into the system and invariant across domains. Because most predicates in our system are binary, even a small number of objects results in a large number of possible predicates. Therefore, we only consider the predicates involving the objects closest to the one being modeled. This heuristic is based on the assumption that there is no "action at a distance". Object types are encoded as unary predicates such as $ball(A)$. The true relational and type predicates for each time step are collected in the set p_t and combined with the continuous data to form the input into the model learner $(\boldsymbol{s}_1, r_1, p_1), (\boldsymbol{s}_2, r_2, p_2), \ldots$ (②).

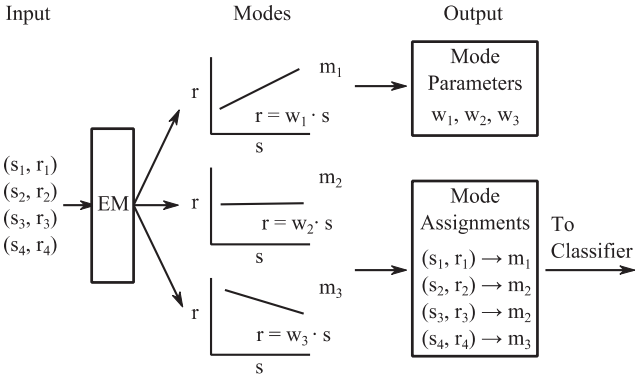


Figure 2. The segmentation algorithm.

Our system must then solve two learning problems, which are described in detail in the following sections. We call the first problem the *segmentation problem*, where the system must identify a set of modes, the parameters of the functions describing those modes, and which modes are responsible for each observation (③). Solving the segmentation problem associates with each input tuple $(\boldsymbol{s_i}, r_i, p_i)$ a corresponding mode index m_i. These augmented observations $(\boldsymbol{s_i}, r_i, p_i, m_i)$ serve as the training input for the *classification problem* (④), where the system must learn a classifier $CLS: \boldsymbol{s}, p \rightarrow m$ that predicts the mode from the initial state of a transition.

The Segmentation Problem

Our system uses Expectation Maximization (EM) to solve the segmentation problem, as shown in Figure 2. Given a set of training examples $(\boldsymbol{s_i}, r_j)$ and a set of modes, EM simultaneously solves for the parameters of each mode and the assignment of examples to modes that results in a locally maximal likelihood. We assume that each mode can be approximated by a linear function, so the parameters for each mode are just a set of weights $\boldsymbol{w}_m$. EM begins with a guess at the parameters of each mode and then iteratively alternates between an expectation (E) step and a maximization (M) step. In the E step, the algorithm calculates the probability that each example i was generated by mode j, assuming that the current parameter estimates for j are correct. We assume the data has Gaussian noise, so the probability that example $(\boldsymbol{s_i}, r_i)$ is generated by mode j with weights $\boldsymbol{w}_m$ follows a Gaussian distribution centered on the dot product $\boldsymbol{w}_m \cdot \boldsymbol{s_i}$ with variance σ^2. In the M step, the parameters for each mode are updated to maximize the likelihood that it generated the examples assigned to it in the E step. This is done with forward stepwise linear regression. We chose forward stepwise regression to avoid overfitting the training data, because the continuous state has high dimensionality even when there are few objects in the environment, but most models only depend on a small number of dimensions. Repeating these two steps guarantees convergence to a local maximum likelihood.

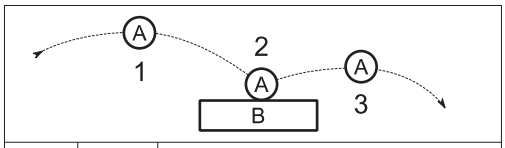
For each new training example, our algorithm runs EM to convergence or until a fixed number of iterations is reached. The algorithm skips the M step when a new example fits an existing mode to within a hand-tuned threshold, allowing it to be responsive enough for online learning. When this is not the case, the M step must rerun the regression for the mode that needs to be updated. While it is difficult to characterize the complexity of forward stepwise regression, it is at least linear in the number of training examples, and hence unsuitable for online learning. Future work may correct this problem, for example by discarding redundant training data.

Textbook EM formulations assume that the number of modes is known, but our system must infer this from training data. We do this by initially assuming that all examples are generated by a single noise mode with a constant low probability. Periodically, the algorithm attempts to find a new linear function that fits a large subset of the noise examples. The system does this by running a second EM loop on the noise data, only assuming that the data was generated by a noise function and a single linear function. If a linear function is found that fits at least 40 examples within the aforementioned accuracy threshold, then a new mode is added containing those examples.

The threshold of 40 is domain-dependent and was chosen to avoid overfitting the noise data and inventing spurious modes, while balancing against the need to discover real modes without requiring too much training data. However, with enough narrow data, the system can still commit to overspecific modes. For example, it may discover two separate constant-valued modes of a ball rolling at two different speeds that can be generalized into a single mode conditioned on the previous speed of the ball. Therefore, when a new mode is discovered, our system will first try to merge it with each existing mode by looking for a function that covers both. Modes will also be removed if they fall below the 40 example threshold. This can occur if the examples in a mode are subsumed by a more general mode.

The Classification Problem

The goal of the classification algorithm is to predict the mode for each transition given the initial state. We hypothesize that many modes can be identified based on common, domain independent spatial relationships, but others are based on the specific numeric properties of the continuous state. For example, the flying mode of a ball can be distinguished from the bouncing mode based on whether the ball is intersecting the ground, but whether the ball is bouncing off a ramp or rolling depends on the numeric value of the ball's y velocity at the beginning of the transition. This leads us to propose that our system actually learns two types of classifiers: a symbolic one



Time	Mode	Predicates
1	I	~intersect(A,B) & left(A,B)
2	II	intersect(A,B) & ~left(A,B)
3	I	~intersect(A,B) & ~left(A,B)
Learned Clause: ~intersect(x,y)		

Figure 3. Simple FOIL classifier learning example.

based on spatial relations and type predicates (p_i), and a numeric one based on continuous properties ($\boldsymbol{s_i}$). These are combined in the final classifier.

For symbolic classification, we use the FOIL (Quilan 1990) algorithm to learn a classifier in the form of a disjunction of Horn clauses that test the spatial predicates of the symbolic state. FOIL is an inductive logic programming (ILP) algorithm, and generalizes over object identities so that the learned clauses describe the concept using variables rather than the actual objects in the training set. We use FOIL because it is simple to implement and sufficient for our experiments, but want to replace it with an incremental algorithm in the future. In our system, each training example consists of all predicates that are true at the beginning of the transition (and implicitly by closed world assumption all predicates that are false), and which mode the transition belongs to.

Consider the example in Figure 3. The FOIL learner is given three observations $(p_1, I), (p_2, II), (p_3, I)$ where I is the flying mode and II is the bouncing mode. It recognizes that $intersect(A,B)$ is the predicate that separates modes I and II, whereas $left(A,B)$ is inconsequential and therefore discarded. The final learned clause, $\sim intersect(x,y)$, is variablized so that it can be used to distinguish between bouncing and flying for any ball and obstacle, not just A and B.

As discussed previously, some modes cannot be distinguished by symbolic information alone. When this is the case, each learned Horn clause may misclassify some negative examples as false positives. Furthermore, true positive examples that cannot be accurately described by Horn clauses will be classified as false negatives. To address this problem, our system learns a numeric classifier that distinguishes between the true and false positives of each clause whose false positive rate is above a hand-tuned threshold. Furthermore, if the false negative rate of the entire disjunction is too high, our system also learns a numeric classifier to distinguish between the true and false negatives. The final decision combines these two types of classifiers as shown in Figure 4. The algorithm has a waterfall model. If any pair of Horn clause/numeric classifier both decide an instance is positive, then it is

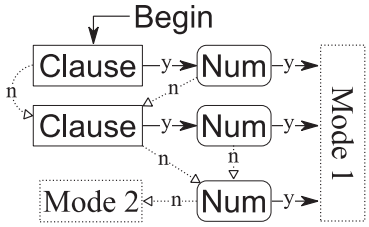


Figure 4. Classification flowchart for how clauses learned with FOIL and numeric classifiers (Num) are combined to make a single binary classification.

labeled as belonging to mode 1. Otherwise it goes on to the next clause/numeric classifier pair. If none of the pairs consider the instance as positive, then a final numeric classifier makes the decision between mode 1 and 2. Note that if the false positive rate of a clause is very low, then the numeric classifier will be null and default to a "yes" answer. The same is true for false negatives.

We currently use Linear Discriminant Analysis (LDA) (Hastie, Tibshirani, Friedman 2001) for learning the numeric classifier. Other methods such as support vector machines can also be used, but we chose LDA due to its simplicity and lack of tunable parameters. As future work, we plan to investigate whether it is possible to learn new spatial predicates from numeric classifiers that prove to be accurate and useful over multiple domains.

A drawback of using numeric classifiers is that they are based on the absolute values of continuous properties rather than the relationships between objects. Therefore, they are more prone to overfitting than the symbolic classifiers, and occasionally will decrease the performance of the overall classification as the system incorrectly second guesses its symbolic classification. We plan to address this problem in future work.

Since FOIL only learns binary classifiers and a model can exhibit more than two modes, we use a one-against-one approach to combine multiple binary classifiers (Tax and Duin 2002). This means that for each pair of modes m and n, we learn a binary classifier using the instances from mode m as positive examples and the instances from mode n as negative examples. During classification, each binary classifier casts a vote for one of its modes, and the mode with the most votes wins. Ties are broken arbitrarily.

Prediction

Having learned a set of linear modes F_m and a classifier $CLS: \boldsymbol{s}, p \rightarrow m$, prediction is straightforward. Given the input state $\boldsymbol{s}$ (**A** in Figure 1), our system first augments the input with predicate information $(\boldsymbol{s}, p)$, just like during learning (**B**). Next, the classifier predicts which mode m governs the transition (**C**). The final prediction is $\boldsymbol{w}_m \cdot \boldsymbol{s}$ where $\boldsymbol{w}_m$ are the linear weights learned for m (**D**).

Ideal	Learned
X-velocity, flying or rolling on flat surface	
vx	vx
X-velocity, rolling or bouncing on ramp	
$k_1 \cdot vx + k_2 \cdot vy + c$	$0.638 \cdot vx - 0.724 \cdot vy + 3.92 \times 10^{-4}$
X-velocity, bouncing against vertical surface	
$-k \cdot vx$	$-0.81 \cdot vx + 3.04 \times 10^{-17}$
Y-velocity, rolling and bouncing on flat surface	
$-k \cdot vy + c$	$-0.81 \cdot vy - 4.87 \times 10^{-19}$
Y-velocity, flying under influence of gravity	
$vy + c$	$vy - 9.8 \times 10^{-4}$
Y-velocity, rolling or bouncing on ramp	
$k_1 \cdot vx + k_2 \cdot vy + c$	$-0.724 \cdot vx - 0.448 \cdot vy - 1.96 \times 10^{-4}$

Table 1. Ideal modes for each model and the learned modes.

Experiments

We test the model learning algorithm's accuracy and generalization in a realistic physics domain. The experimental domain is a 2 dimensional square room containing a ball, a box, and a ramp. The box and ramp are stationary after initial placement. The ball can bounce and slide against the walls, floor, box, and ramp, and is affected by gravity. The domain is implemented with the Chipmunk Physics Engine (Lembcke 2013).

Training occurs in blocks, each consisting of initializing the room in a particular configuration and then running the physics simulation for 200 time steps. The initial positions and sizes of the ball, box, and ramp, and also the ball's initial direction of travel are varied in each block. Furthermore, the exact distances between objects are randomly varied, and the entire room is randomly placed with respect to the origin of the coordinate system. This randomization makes it difficult for algorithms that depend on absolute coordinate values to generalize, but it does not affect generalization using spatial relationships. There are 40 relationally unique initial configurations, and we repeat them three times with different random seeds, for a total of 120 training scenarios. We test the accuracy of the learned models on 120 test scenarios generated in the same way, but with different random seeds. Each test block also runs 200 time steps. Finally, we repeat this training-testing sequence five times, randomizing the presentation order of the training configurations each time, since this effects how the modes are learned.

The algorithm learns two models simultaneously: one for the horizontal or x component of the ball's velocity, and one for the vertical or y component. These two models are qualitatively different because gravity acts on the y axis but not the x axis. The algorithm does not learn models for the ball's position because it can be derived from the velocity predictions. We expect the algorithm to learn individual modes corresponding to the ball rolling on flat surfaces, flying in the air, rolling on the ramp, or bouncing off objects.

Table 1 shows the complete list modes we expected in the environment and those learned by model. vx and vy are the values of the x and y velocities in the initial state of the transition. Except for the very tiny constants introduced by rounding errors, all the learned constants are correct: We used a gravity constant of $9.8ms^{-2}$, and our simulation step size was 10^{-4} seconds. Each object has a restitution constant of 0.9, resulting in the -0.81 constants on the bouncing modes. The system also discovered modes that are more general than we anticipated. We expected separate modes for bouncing and flat rolling in the y-velocity model, as well as for bouncing against a ramp versus rolling on it, but the system combined them with no loss of accuracy in each case. There were also occasional training sequences that resulted in irregular modes, but most of them were pruned after sufficient training.

We compare the accuracy of our model learning method to locally weighted regression (LWR). LWR (Atkeson, Moore, and Schaal 1997) is an instance-based function approximation technique that has been applied successfully to many model learning scenarios in robotics. In LWR, learning involves simply storing each training instance (x_i, y_i) in a table. To make a prediction for instance x, LWR chooses k training instances closest to x and fits a linear function to them using regression. The function is then used to make the prediction for x. This approach has been shown to provide good generalization while fitting arbitrarily complex functions. The most important difference between LWR and our algorithm is that LWR learns a single smooth function conditioned on absolute coordinates, whereas our algorithm learns a set of functions that can change abruptly as spatial relationships change. We use Euclidean distance as the measure of closeness between instances, and set k at 300 and used a d^{-2} kernel, verified empirically to give good results. We center the training and testing data on the location of the ball so that the distance metric measures relative distance between the ball and other objects, which is more robust than using absolute coordinates. Note that we don't perform this centering for our algorithm.

Figure 5 plots the prediction accuracy for both x and y velocities using our model learning method and LWR. The y-axis has a logarithmic scale and marks the ratio of the model's prediction error and the baseline error. The baseline error is the average error of a model that always predicts no change in the modeled dimension. These values are 8.38×10^{-4} for x-velocity and 9.75×10^{-4} for y-velocity. The lines in the plot represent median values. Error bars were not drawn because they obscured the plot. Except for the first two data points, the 5 and 95 percentile ranges of each data point are completely separated. The

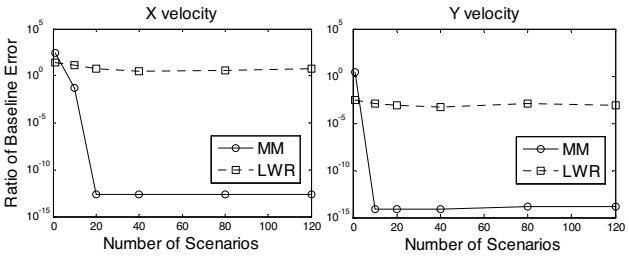


Figure 5. Comparison of prediction accuracy for x and y velocity between our model learning method (MM) and LWR.

results are averaged across 5 different training orders, all 40 initial configurations, and 3 random seeds.

The plot for x-velocity only shows the prediction errors for test points that exhibited either the bouncing or ramp-rolling mode. This is because the third mode – flat rolling/flying – results in no change in x-velocity and is thus easy to predict and uninteresting, but accounts for 94% of the generated test data. Including these test examples would drown out the discrepancy between our approach and LWR on the more interesting transitions, such as rolling on the ramp and bouncing off objects.

The results show that our algorithm outperforms LWR as expected, and that LWR was not able to perform significantly better than the null baseline. The major shortcoming of LWR is that Euclidean distance over the raw input space is a poor measure of the similarity of two transitions, even after centering the data on the ball. Therefore, the learned model doesn't generalize well. We also analyzed nature of the prediction errors made by our model, and found that they all resulted from incorrect mode classifications. For both the x and y velocities, the linear functions for the natural modes of the domain were learned accurately after only a few examples, but the FOIL classifier converged more slowly, and never reached perfect accuracy.

As discussed previously, there exist other multi-modal learning algorithms, but they do not consider spatial relationships in their mode classifiers. We argue that our approach performs better than these other approaches in spatial domains. To show this, we compare the classification accuracy of FOIL with two popular classifiers that only rely on continuous state information – support vector machines and nearest neighbor – on the physics simulation data. For each data point, we use the mode that results in the lowest prediction error as the true mode. For the SVM and NN classifiers, each training and test example has the form $(\boldsymbol{s}, m)$, where $\boldsymbol{s}$ is the vector of continuous state properties and m is the true mode. For these examples, we centered the state vectors around the ball in the same way as for LWR. We use a quadratic kernel for the SVM classifier. The results are shown in Figure 6. Again, there is the problem of the flat-rolling/flying mode dominating the x-velocity test set. Here, we show the accuracy of all three classifiers averaged over all examples ("All" condition), as well as over only the examples from the ramp-rolling and bouncing modes ("Hard" condition). Only the All condition is shown for y-velocity. The plots show the FOIL classifier learns significantly faster and converges at a much higher accuracy than both SVM and NN.

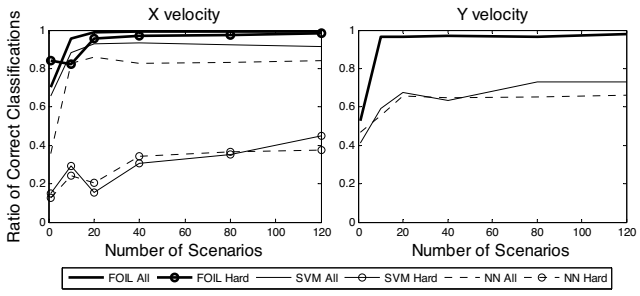


Figure 6. Performance of mode classifiers learned with FOIL, SVM, and NN on x and y velocity modes.

Conclusion

We have presented an algorithm for learning piecewise linear action models conditioned on both symbolic spatial relations and continuous state properties. Our main argument is that in spatial domains with physics-like behavior and multiple interacting objects, knowledge of spatial predicates can lead to more generalization and accuracy in model learning. We have shown that our model learning approach outperforms LWR and our FOIL-based mode classifier outperforms SVM and NN classifiers that only use numeric state information.

One of the major shortcomings of our system is that while it accepts training examples in an online manner, many of its parts are not incremental, and it is not fast enough to run in real-time. The major performance bottlenecks are the algorithm for searching for new modes and FOIL, both taking on the order of seconds for each execution in the domain presented here. This will be addressed in future work.

Another limitation is our assumption that all modes are linear. Our approach should theoretically work with higher-order modes as well, but when the individual modes are capable of modeling complex functions, the system requires more sophisticated ways to balance learning fewer complex modes with learning more simple modes.

Acknowledgment

The authors acknowledge the funding support of the Office of Naval Research under grant number N00014-08-1-0099.

References

Atkeson, C., Moore, A., and Schaal, S. 1997. Locally Weighted Learning. AI Review 11. 11-73.

Forbus, K. D. 1980. Spatial and Qualitative Aspects of Reasoning about Motion. Proceedings of 1st Proceedings of the 1st Annual National Conference on Artificial Intelligence.

Forbus, K. D., Nielsen, P. and Faltings, B. 1991. Qualitative Spatial Reasoning: The CLOCK Project. Artificial Intelligence 51, p 417-471.

Hastie, T., Tibshirani, R. and Friedman, J. 2001. *The Elements of Statistical Learning*. New York: Springer-Verlag.

Kuipers, B. 1994. *Modeling and Simulation With Incomplete Knowledge*. MIT Press.

Lembcke, S. 2013. Chipmunk Physics Engine. Version 6.1.2. Available from http://chipmunk-physics.net.

Nguyen-Tuong, D.; Peters, J. 2011. Model Learning in Robotics: a Survey. Cognitive Processing, 12, 4.

Potts, D. 2005. Incremental Learning of Linear Model Trees. Machine Learning 61.

Quilan, R. 1990. Learning Logical Definitions from Relations. Machine Learning 5, p. 239-266.

Paoletti, S., Juloski, A., Ferrari-Trecate, G. and Vidal, R. 2007. Identification of Hybrid Systems: A Tutorial. European Journal of Control, vol. 13, p. 242-260.

Tax, D. and Duin, R. 2002. Using Two-Class Classifiers for Multiclass Classification. Proceedings of the 16th International Conference on Pattern Recognition.

Toussaint, M., and Vijayakumar, S. 2005. Learning Discontinuities with Products-of-Sigmoids for Switching between Local Models. Proceedings of the 22nd International Conference on Machine Learning, p. 904-911.

Troha, M. and Bratko, I. 2011. Qualitative Learning of Object Pushing by a Robot. Working papers of the 25th International Workshop on Qualitative Reasoning.