Building Characters: A Form of Knowledge Acquisition

Yusuf Pisan

School of Computer Science and Engineering
The University of New South Wales
Sydney, NSW 2052, Australia
ypisan@cse.unsw.edu.au
http://www.cse.unsw.edu.au/~ypisan/

Abstract

While computer games have become more sophisticated, most computer characters follow a limited static script in their interaction with the user. We advocate a knowledge acquisition approach to building characters and use model-based classification techniques as our learning mechanism. Model-based classifiers can be readily integrated with existing rule-based approaches for building computer characters. We demonstrate our idea using a simple simulated game of predator and prey, examine the types of rules created by the classifier and discuss how to develop adaptable characters. This approach facilitates creating and debugging characters and we believe it will lead to more interesting games.

Introduction

Computer games are becoming more sophisticated. We have better graphics, better sound, all sorts of gadgets for control, hundreds of levels and multitude of different characters the user can interact with. Unfortunately, most of the computer characters are dumb. Their conversation is a bore, their activities and abilities limited. Mindless creatures try to kill you without any regard for their survival or any understanding of their weaknesses. Progressing through the game levels increases the number of creatures and their speed, but the creatures remain as mindless as ever.

The challenge is to create characters that learn and adapt as a result of their interaction with the user. To this end, game designers have been trying to incorporate learning methods, often in the form of neural networks and genetic algorithms. Unfortunately, the model-free approach employed in these methods is difficult to combine with the structured model-based approach of using finite state machine and production rules for building characters. As a result, character development becomes a trade-off between automated methods and hand-scripted behaviours. When automated models fail, such as a neural network not performing in a satisfactory way, they cannot be debugged directly and the process usually has to be started from scratch with very little intuition gained in the process. Automatically constructed models should resemble the models built by programmers making model construction a combined effort. Automated character authoring tools that can produce models that can be easily understood and debugged, possibly in the form of concise sets of rules, can improve the quality of computer characters while reducing development time. Among machine learning methods, inductive learning is especially suited for this task. We approach the problem of building a character as a knowledge acquisition task and describe a form classification that can be used to automatically generate production rules that can serve as character models.

How to build a character

To build a character we have to construct a model that captures the knowledge that the character will employ. If the character is supposed to be similar to people in ability, we might have a number of people play the role of this character, allow them to build expertise and then interview them to extract the strategies they have developed. This form of knowledge acquisition is often employed when building expert systems. Unfortunately, articulating strategies in detail, describing the necessary and sufficient conditions for actions, and constructing a detailed knowledge base in this way is difficult and time consuming. Considering that extracting knowledge from practicing doctors, engineers or lawyers with deep knowledge of their respective fields has proved to be difficult, we have good reason to believe that expertise in playing a complex game would be similarly hard to extract.

Although techniques, such as Ripple Down Rules (Compton and Jansen 1989), developed in the knowledge acquisition community can be useful in developing strategy games, for characters that are radically different from humans a different approach is needed. Developing the appropriate expertise by playing the role of a strong but dumb character would be very difficult if not impossible. Our perspective is that the character is an independent entity trying to generate a model of the

world. This perspective allows us to adapt the tools used by the knowledge acquisition community.

Classification

It is possible to view the expert decision about a situation as a classification task. In medical diagnosis, if we have a large database of patient records with their associated diseases, we can inductively construct a model by generalising from specific examples. In a computer game, the character can decide what the next action should be based on what its observations and its historical data. From this perspective, the character is similar to a baby trying to create a model of the world based on the information it has access to. In a deterministic world where the character has access to all the world information, character's model would converge to the world model. A more interesting model emerges when the world is non-deterministic, the character has access to a limited set of features about the world, possibly defined by its sensors and the character as limited memory. In this case, the model created is the set of rules that can fit historical data most accurately.

Given sufficient data that includes all the relevant features necessary to classify the case, an inductive learner will converge to the appropriate classification model. What constitutes sufficient data depends on the complexity of the underlying model and the amount of noise in the data. For computer games, the character has to model the a subset of its world based on its sensors through experimentation. We are interested in the evolution of the models that the characters create based on their limited access to world features.

A Small experiment

We created a toy example where a fox hunts rabbits, squirrels and chipmunks to illustrate the how rules generated by a classifier evolves under different conditions. In this world, when the fox sees a prey, it has to decide whether to hunt or to rest. The current state of the fox is defined by its attributes shown in Table 1. The rules determining whether the fox can successfully hunt is given in Table 2. We ignore the states where there is no visible prey. There are a total of 24,000 possible states the fox can be in, and the fox can successfully hunt in only 10% of them. We are interested in how the quality and the accuracy of the rules fox develops changes with data while making sure that the concepts expressed as rules remain easy to interpret.

Fox's Attributes	Range of Values
speed	1..10
hunger	1..10
distance	1..10
mood	happy, sad
terrain	grassland, forest, swamp, mountain
prey type	rabbit, squirrel, chipmunk

Table 1: State attributes for fox

Rabbit	Squirrel	Chipmunk
speed > 7 hunger > 7 distance < 5 mood = sad terrain = forest, grassland	speed > 5 hunger > 5 distance < 7 mood = any terrain = forest	speed > 3 hunger > 3 distance = any mood = happy terrain = any

Table 2: Conditions for a successful hunt

We use C4.5 (Quinlan 1993) as our classifier. While a number of other classifiers are available, such as CART (Breiman 1984), C4.5 is well-known in the machine learning community and commonly used as a reference program to test new classifiers against. Most importantly for our purposes, C4.5 generates a set of production rules as its model which can be easily examined by programmers.

We randomly generated 10, 100, 1000 and 10,000 cases, classified them according to the rules from Table 2, and examined the models derived by C4.5. The generated cases reflect the distribution of successful hunts in the state space. As a result, approximately 10% of the cases in each data set would be labelled "HUNT" and 90% would be labelled "REST".

For 10 cases, the model generated consists of a single rule with no conditions and "REST" as its result. This is as expected since the number of cases is insufficient to make any generalisation for when to hunt. Since we are only interested in whether a given state is a good candidate for hunting and no penalty is associated with resting, for a randomly chosen state resting is the right choice.

When 100 cases, representing approximately 0.4% of the state space, is used as input the rule for hunting chipmunks, as given in Table 2, is correctly derived. No rule for hunting squirrels or rabbits is found. Based on the rules we have specified, chipmunks are easiest to hunt. Of the 8000 possible states with chipmunks, hunting a chipmunk is successful in 1960 states (or 25% of the time), compared with hunting rabbits which is successful only 0.9% of the time.

For 1000 cases, the rule for hunting chipmunks and squirrels is correctly derived, but the rabbit rule continues to be missing. Of the 294 cases involving rabbits only

one case is labelled to be a good choice for hunting rabbits and is not generalised as a result.

For 10,000 cases, the rule for hunting chipmunks, squirrels and the two additional rules for hunting rabbits is correctly derived. Each rule is made up of conjuncts so two rules are required to specify hunting a rabbit in for grassland or in forest.

When we introduce 5% noise into the data, the rules generated deteriorate slightly with over-constrained and under-constrained conditions for 100 and 1000 cases. For 10,000 cases the correct rules are derived successfully under 5% and 10% noise, but rules with faulty conditions start appearing when the noise level is increased to 20%. A faulty rule derived for 100 cases with noise level at 5% is given below with the correct conditions on the right.

Rule 4:

animal = SQUIRREL
speed > 4 ***X*** *speed > 5*
hunger > 5
X *missing distance < 7*
X *missing terrain = FOREST*
-> class HUNT

As expected, the underlying concepts were discovered with increased accuracy as the number of cases increased. The effect of noise deteriorated the rules produced for smaller data sets, but the larger data set was not effected by noise until the noise level was increased to 20%.

There are two interesting results. First, it was possible to derive the correct rules with only a small sampling of the state space. Second, the rules produced reflected the experience of the character and were easy to understand and modify by programmers.

In this simple example, fox eventually learns the rules for a successful hunt. There are a number of ways in which we can increase the complexity of this game: by making the fox starve after a set number of unsuccessful hunts, by changing what percentage of knowledge is inherited by offspring, by limiting the number past hunts that can be remembered, by changing the rules of the hunt gradually and by allowing the prey to learn as well. In a well-tailored scenario, the fox would never become the perfect hunter, but would have setbacks followed by learning periods where its hunting improves.

We have focused on the behavioural aspects of characters, but a similar approach can also be used for conversational agents. First, we can use the same set of strategies to determine the level of knowledge that the character has. The level of knowledge directly effects the content of the conversation. Next, conversation can be used as a tool to change the attitude of another character where learning takes place in the form of differentiating what is the appropriate attitude or phrase to use with different characters. Complex conversations that require planning are currently beyond the scope of what can be achieved by classifiers.

Although classification is a powerful technique, the number of examples required to develop complex strategies is not feasible for many applications. However, classification techniques are appropriate for our goal to create characters that are developed by a combination of hand-scripting and automated tools. Classifiers that generate easy to modify rules make the learning process visible to the programmer and make it easier to debug and develop complex characters.

Conclusion

The biggest advantage of using classification as a model building tool is its simplicity (hence speed); the disadvantage is that the cases have to be described in the form of an attribute-value description. Learning methods that create explicit models that can be understand by programmers can be more easily integrated and tested. The form of classification technique we have used does not handle varying rewards or time delayed rewards. Other reinforcement learning techniques that address some of these issues, such as Q-learning (Watkins and Dayan 1992) and TD(λ) (Barto, Sutton et al. 1983), lack the explicit models which make them unsuitable for our purposes. Convergence to the right model has become one of the factors in evaluating learning systems. While a classification system, such as C4.5, as well as Q-learning and TD(λ) are guaranteed to converge under the right conditions, the behaviour exhibited and the models created during this convergence is what makes interesting characters rather than their discovery of the optimum strategy. For computer games, this convergence would be intentionally slowed down or crippled to create environments that are continuously changing.

Characters in a computer game have dual roles. Their first role is to react to and learn from their environment. This can be achieved by any learning method or by clever programming. Their second role, entertainment, requires artistic intervention by designers and can be best achieved if the character models remain accessible to programmers.

References

Barto, A. G., Sutton, R. S. and Anderson, C. W. 1983. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics* 13(5):834-846.

Breiman, L. 1984. *Classification And Regression Trees*. Belmont, California: Wadsworth International Group.

Compton, P. and Jansen, R. 1989. A philosophical basis for knowledge acquisition. In *Proceedings of the Third European Knowledge Acquisition for Knowledge-Based Systems Workshop*, 75-89. Paris.

Quinlan, J. R. 1993. *C4.5: Programs for machine learning*. San Mateo, Calif.: Morgan Kaufmann Publishers.

Watkins, C. J. C. H. and Dayan, P. 1992. Q-learning. *Machine Learning* 8(3):279-292.