Fast Learning in an Actor-Critic Architecture with Reward and Punishment

Christian Balkenius, Stefan Winberg
Lund University Cognitive Science, Sweden

Abstract. A reinforcement architecture is introduced that consists of three complementary learning systems with different generalization abilities. The ACTOR learns state-action associations, the CRITIC learns a goal-gradient, and the PUNISH system learns what actions to avoid. The architecture is compared to the standard actor-critic and Q-learning models on a number of maze learning tasks. The novel architecture is shown to be superior on all the test mazes. Moreover, it shows how it is possible to combine several learning systems with different properties in a coherent reinforcement learning framework.

Keywords. Reinforcement learning, reward, punishment, generalization

Introduction

Reward and punishment are often seen as opposite values on the same dimension. This is especially true for reinforcement learning where reward is often represented by positive reinforcement values, while punishment is represented by negative values. Although such a view of reward and punishment may be useful in many cases, it ignores the fundamental difference in how it is appropriate to react to the two types of situation [1].

It is useful to distinguish between passive and active avoidance [1]. Active avoidance is the situation when it is necessary to try to escape, for example when being chased by a predator. Passive avoidance on the other hand does not necessarily require any action. All we need to do is avoid doing something dangerous, such as avoiding running over a cliff.

For appetitive learning, it is useful to be able to generalize to new similar situation. If one situation or action has proved to be rewarding, it is useful to try out similar actions in the future to explore if they too will result in a reward. The appetitive part of a learning system should thus maximally generalize previously learned behaviors to new situations. To make this possible, it is necessary that the coding of the current situation or state contains sufficient information to support generalization. However, a maximally generalizing system will obviously overextend its behavior to situations were they are not appropriate. Context provides a mean to greatly reduce the number of incorrect generalization, making it easier to separate the relevant information about a given situation. Context can be thought of as any information that can be used to characterize the situa-

1Corresponding Author: Lund University Cognitive Science, Kungshuset, Lundagård, SE-222 22 Lund, Sweden; E-mail: christian.balkenius@hacs.lu.se.
tion, such as the task, question, place or even the goal. Studies made on animals suggest that a behavior learned in one context is carried over to other contexts, but learned inhibition of a behavior will be unique to each context where the behavior was extinguished [8,10]. Most reinforcement learning algorithms learn to complete a single task in one context, but animals apply what they learn in one context to other contexts as well.

The solution we propose is to divide the input into one focal part, which can be seen as the attended part of the environment, and a contextual part, which codes for the situation [3]. The focal part is used to control actions by being directly associated with behaviors while the contextual part is used to select between different possible behaviors. Previous studies have shown that it is possible to construct a context sensitive artificial neural network that fulfills these demands [5,15], while simultaneously avoiding catastrophic forgetting [9]. It has been used to model context sensitive categorization [5], task-switching [5] and developmental disorders [6]. We recently tested this type of mechanism within a Q-learning framework [16]. Here, we develop these ideas further and implement context sensitivity in an actor-critic framework [7]. In addition, we investigate how punishment can be included to speed up learning.

1. Overview of the System

The general reinforcement learning framework illustrated in Fig. 1 was used for all the simulations and implemented in the Ikaros system [4]. A simpler version of this framework has been previously described [16], and it is here extended by the addition of an actor and critic [7] and a dedicated punishment system. The extended framework consists of the five main modules ACTOR, CRITIC, PUNISH, RL-CORE, and SELECT.

The module ACTOR is responsible for action selection in each state. It has three inputs and one output. One input-output pair is used to calculate the expected value of each possible action in the current state. The other two inputs are used to train the module on the mapping from a state delayed by two time steps to a target delayed by one time step. Any of a number of algorithms can be used as ACTOR, ranging from tables to different types of function approximators and artificial neural networks. Because of the separate input for training and testing, the module ACTOR can simultaneously work in two different time frames without the need to know about the timing of the different signals. Here we use the context sensitive function approximator we have previously developed [5,15,16].

The module CRITIC is used to estimate the expected value of an action a in each state. Learning is dependent on the current policy of the ACTOR module and the CRITIC module must learn to evaluate the actions of the ACTOR module. Just like the ACTOR module, the CRITIC module has three inputs and one output. The inputs are separate for training and testing but receive the same data. The function is similar to the ordinary implementation of actor-critic architectures [7].

The purpose of the module PUNISH is to learn whether or not any of the surrounding states of the current state is inaccessible. That is, if action a in state s will lead to the return to the same state or not. How this is done is often rather specific to the task at hand but can usually be done by analyzing the state and action vectors of previous time steps to see if the selected action caused a state change or not. If not, a negative association is formed between state and action which will greatly reduce the risk of repeating this
Figure 1. Overview of the general reinforcement learning framework. The different lines indicate different delays ($\Delta$) on the connections. See text for further explanation.

behavior. The learning in the PUNISH module is driven by the collision signal from the environment. This signal is active each time the agent tries to move into a wall. When the PUNISH module receives a collision signal, it will learn to associate the attempted action in the current situation with punishment, which reduces its likelihood in the future. Since the state is coded in a way that allows generalization of punishment to other identical situations (see below), the agent will rapidly learn to not try to move into obstacles.

The module RL-CORE is the reinforcement learning specific component of the system. This module receives information about the current state in the world, the current reinforcement, the action selected at the previous time step and data from both the ACTOR and CRITIC module in the current state. This is used to calculate the training target for the ACTOR and CRITIC modules. The target vectors are both equal to the input vector of the CRITIC from the last time step, but for the ACTOR module’s target vector, the value of the action selected the last time step is replaced with the maximum discounted action value received from the CRITIC in the current time step. For the target vector of the CRITIC module, the incoming reinforcement signal is used.

The module SELECT performs action selection based on its input from RL-CORE. It may also potentially have other inputs that determine what action is selected. This module may for example implement Boltzmann selection or $\epsilon$-greedy selection [13].
may also use different forms of heuristics to select an action. It is even possible that
the action selected is entirely independent of the inputs from RL-CORE. In this case,
RL-CORE will learn the actions performed by some other subsystem.

All communication between the different modules consists of numerical matrices.
The connections may optionally delay the signal by one or several time steps (Fig. 1). One advantage of this framework is that the different modules can be exchanged to build
different forms of reinforcement learning systems. Another advantage is that all timing
is taken care of by the delays on the connections between the different modules. If the
world is slower at producing the reward, the only things that need to be changed are the
different delays in the system.

An important part of the architecture is its ability to handle generalization in different
ways in different parts of the system. The ACTOR, CRITIC and PUNISH modules all
receive input about the current state of the world, but they are coded in different ways to
support different forms of generalizations.

The information about obstacles around the current location in the world consists of
a binary matrix with 1 for obstacles and 0 for open space. This matrix is inverted (INV)
and merged (MERGE) with the original representation to form vector with a dual coding
of the surrounding around the agent. The resulting vector is given as input to the ACTOR.
This coding makes it easy for the actor to generalize actions to new situations since this
input contains information that lets the actor learn about the actions that are possible in
each state. In addition, the ACTOR module also receives location input from the world
that is used for contextual inhibition [16].

The CRITIC does not use the focal information about obstacles around the agent.
Instead, it uses the location code to learn the value for each location in the environment.
This is the most suitable information for learning the shortest path through an environ-
ment.

Finally, the PUNISH module uses another coding of the surrounding around the
agent. The binary pattern with nine elements is ‘decoded’ into one of 512 distinct vectors
with a single 1 at one position and 0 at the others. This decoding is used as a simpli-
fied means of getting distinct categories for environments where the agent has received
punishment. In practice, all the different categories are not used, and it would be possi-
bile to dynamically create the required categories instead, but here we have opted for the
simplest solution.

2. Simulations

A typical reinforcement learning problem has a multidimensional state space. This makes
it difficult to visualize the problem in a way that is easy to comprehend. Therefore, a
navigation task through a two-dimensional maze is often chosen since each state can be
represented by a physical location. When the state space is visualized as a two dimen-
sional surface, the solution can be described as a path from the start state to the goal
state. Initially the agent has no knowledge of the state space. Therefore, the first time
the agent enters the maze it has to search it through to find the goal. It is important to
note that the two-dimensional layout of the state-space is not available to the agent. Our
intuitions about the expected behavior can thus be misleading. Nevertheless, a maze is a
useful visualization of a state-space and we have chosen a set of mazes we call 17F4U
that illustrates different strengths and weaknesses of reinforcement learning algorithms (Figs. 2 and 3) [16]. The mazes superficially looks like the letter in the name of the set. We tested three algorithms in the different mazes: standard tabular Q-learning [14], the novel actor-critic architecture proposed above and the same architecture but with the punishment module disabled [7].

The Q-learning model was initiated in a way to support efficient exploration [11]. For each model and each environment, we recorded the number of steps needed to reach the goal location from the start at each trial. The number of extra steps beyond the shortest path was recorded and added for each trial to give the result graphs which show the average of 100 simulations.

3. Results

Fig. 2 shows the results for the different models on the narrow mazes. On the simple straight corridor, both the actor-critic models learn the optimal policy very quickly, while Q-learning needs about 10 trails before the optimal policy is found. The quick learning of the actor-critic variants depends on the ability to generalize the action of moving straight in a corridor to all positions in the maze.

The merit of the generalization ability is also evident in the 7-maze and the T-maze, but only when it is combined with punishment. The punishment system improves learning speed considerably by suppressing actions that lead into walls. Note that the suppression of these inappropriate actions is learned from experience.

The 4-maze was selected since it is optimally bad for the generalization abilities of the two actor-critic models. The generalized action of moving straight in a corridor will lead it to the dead end. As expected, the actor-critic model does not work well in this maze. However, the model with the punishment system is doing much better and learns faster than both the other models. In the U-maze, the results is similar and the actor-critic with punishment is again outperforming the other models.

Fig. 3 shows the results for the wide mazes. For the models tested here, the results are similar for the wide mazes as for the narrow ones. In all cases, the actor-critic with punishment is best of the three models.

4. Discussion

The results of the simulations show that the combination of a punishment system with a punishment system results in very efficient learning, even for problems designed specifically to be as difficult as possible for this kind of architecture. The separation of the appetitive learning subsystem into an actor and a critic makes it possible to use different forms of generalization for actions and state evaluation. The addition of a punishment system dedicated to aversive learning makes the architecture even more powerful since it allows the system to learn about actions to avoid. Together, the appetitive and aversive parts of the system learns different heuristics that it can later use during exploration.

We described the novel architecture from a maze learning perspective, but the framework is much more general. The division into the three modules ACTOR, CRITIC and PUNISH can be applied also to other reinforcement learning problems. The three sys-
tems learn in three complementary ways. The ACTOR learns associations between the coding of the current state that supports generalization and potential actions in that state. This learning will generalize to other similar states which will give the agents suggestions about what to do in new states it has never seen before based on “perceptual similarity”. When the generalized actions are not successful, they will be inhibited in the current context. There is thus an interplay between generalization and specialization in the ACTOR module [16].

Unlike the ACTOR module, the CRITIC module does not generalize to similar
states. The reason for this is that it attempts to learn a goal-gradient toward the goal [1]. This gradient should be based on the location with respect to the goal and not generalized based on similarity. Our earlier attempts to include contextual inhibition within a Q-learning framework did not distinguish between the role of the ACTOR and the CRITIC, and would sometimes generalize incorrectly which would slow down learning [16]. The actor-critic architecture overcomes this problem. As far as we know, this feature of the actor-critic architecture to allow more flexible generalization has not been investigated before.
The PUNISH module, finally, learns in a way that is different from both the ACTOR and the CRITIC. Like the ACTOR it uses a description of the current state rather than the location. However, unlike the ACTOR that generalizes maximally, the PUNISH module does not generalize at all. It uses different categories for each of the possible local environments around the agent. This reflects a fundamental asymmetry between reward and punishment in that rewarding situations is something that the agent will seek out again and learn more about, while punishing situation will be avoided. The agent will thus not explore potentially punishing actions again and it is important that this suppression is not extended to actions that may be advantageous in other situations.

Acknowledgements

This work was supported by the EU project Mind Races, FP6-511931.

References