A Consolidated Actor-Critic Model with Function Approximation for High-Dimensional POMDPs

Christopher Niedzwiedz and **Itamar Elhanany** and **Zhenzhen Liu** and **Scott Livingston**
Department of Electrical Engineering and Computer Science
University of Tennessee, Knoxville
1508 Middle Drive
Knoxville, TN 37996-2100

Abstract

Practical problems in artificial intelligence often involve both large state and/or action spaces where only partial information is available to the agent. In high-dimensional cases, function approximation methods, such as neural networks, are often used to overcome limitations of traditional tabular schemes. In the context of reinforcement learning, the actor-critic architecture has received much attention in recent years, in which an actor network maps states to actions and a critic produces value function approximation given a state-action pair. This framework involves training two separate networks, thus requiring the critic network to effectively converge before the actor is able to produce a suitable policy, resulting in duplication of effort in modeling the environment. This paper presents a novel approach for consolidating the actor and critic networks into a single network that provides the functionality offered by the two separate networks. We demonstrate the proposed architecture on a partially observable maze learning problem.

Introduction

Reinforcement learning (RL), as a machine learning discipline (Sutton & Barto 1998), has received significant attention from both academia and industry in recent years. What sets RL apart from other machine learning methods is that it aims to solve the *credit assignment problem*, in which an agent is charged with evaluating the long-term impact of each action taken. In doing so, an agent which interacts with an environment attempts to maximize a value function, based only on inputs representing the environment's state and a nonspecific reward signal. The agent constructs an estimated value function that expresses the expected return from taking a specific action at a given state.

Temporal difference (TD) learning is a central idea in reinforcement learning, and is primarily applied to model-free learning problems. As a framework, TD draws from both dynamic programming (DP) and Monte Carlo methods. Similar to DP, TD learning bootstraps in that it updates value estimates based on other value estimates, as such not having to complete an episode before updating its value function representation. Like Monte Carlo methods, TD is heuristic in that it uses experience, obtained by following a given policy (i.e. mapping of states to actions), to predict subsequent value estimates. In that end, the method is essentially updating one guess based on another (Sutton & Barto 1998).

Practical problems in RL, as well as other AI disciplines, are characterized by having high-dimensional state and/or action spaces. Traditional tabular methods for representing state-action pairs suffer from what is commonly known as the curse of dimensionality, since the state space grows exponentially with linear increase in the number of state variables. A common approach to scaling RL systems is to employ function approximation methods, such as neural networks, for value function estimation and policy derivation. Function approximation techniques overcome the curse of dimensionality by constructing a parametrized representation of the state space. Assuming an underlying Markov Decision Process (MDP), a feedfoward neural network consisting of weighted inputs that feed to one or more hidden layers of neurons, can be used for value function approximation. The latter is produced by the output of the network, which is commonly a linear combination of the outputs of the hidden layers. In the more general case of Neurodynamic Programming (NDP), neural networks are used at the core of the TD learning process to approximate the value function, the policy to be followed by an agent, or both.

In many of these problem, only partial information regarding the state is available to the agent. In these instances, it can be assumed that the underlying system is a normal MDP where the true state information is unavailable. The agent utilizes observations of the environment to form its own state estimation. These Partially Observable MDPs (POMDPs) require an agent to infer the true state of the environment when provided with a series of non-unique observations. With respect to neural networks, this requires the use of recurrent elements to provide the system with memory. These memory neurons are used to provide context to new observations and are combined with the input observation to yield state inference. Even simple problems, when posed as a POMDPs, suffer from the fore mentioned curse of dimensionality.

A particular TD learning architecture that has received attention in recent years due to its inherent scalability properties, is the actor-critic model. The latter consists of two neural networks. The first, called a *critic*, is responsible for approximating the value function (i.e. mapping state-action

Copyright © 2008, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

pairs to a value estimate), thus providing a critique signal for the agent. The second (*actor*) network is charged with constructing a policy. Both networks receive observations at their inputs and, therefore, both have to be capable of accurately modeling the environment with which the agent interacts. This is an obvious replication of functionality between the two networks, suggesting that a more compact representation is merited. Before the actor is able to converge to an optimal policy, the critic must first converge its model of the environment to provide an accurate value function estimation (Si, Yang, & Liu 2004).

Reinforcement learning has had great success in diverse application domains. A helicopter controller has been designed using direct neural dynamic programming (DNDP) in (Enns & Si 2003). First simulating the environment using an analytical model and then applying the controller to the real world, this application demonstrated that reinforcement learning methods can be applied to complex control situations and yield positive results. Another example of NDP application has been discussed in (Marbach, Mihatsch, & Tsitsiklis Feb 2000), where call admission control (CAC) and associated routing policies were successfully solved using NDP, despite the complexity of the problem involved.

Recent work has attempted to address the issue of the apparent redundancy in the actor-critic model. A model free actor-critic network has been presented to approximate both the critic signal and action probabilities (Mizutani & Dreyfus 2004). This approach has been applied to low-dimensional stochastic dynamic programming path finding problems. The model is a departure from temporal difference learning methods in that action probabilities are calculated instead of a direct policy. Although a step in the right direction, the scheme as a whole does not appear to scale toward high-dimensional state-action spaces.

This paper presents a Consolidated Actor-Critic Model (CACM) to combine the actor and critic modules into a single network. The CACM estimates the value function and concurrently generates a policy, making it fundamentally different from existing model-free solution methods. Such consolidation of functionality offers improved performance with respect to the traditional actor-critic model, as only one network need model the environment and internal representation is shared. The CACM takes a series of observations and an action to be taken on the inputs and approximates the value function as well as produces a subsequent action.

The paper is structured as follows. First, an outline of the traditional actor-critic framework is presented. This is followed by the introduction of the consolidated actor-critic model, its derivation and relationship to the traditional method. Simulation results are provided as means of demonstrating the key attributes of the CACM in a partially observable environment. Finally, the conclusions are drawn.

The Actor-Critic Model

TD learning methods in reinforcement learning, such as Q-Learning and SARSA, which employ tables to represent the state or state-action values are practical for low-dimensional, fully observable problems. They rapidly lose merit as new state variables are introduced, as each state variable increases the state space exponentially, increasing the amount of system memory and processing power required. Further, the introduction of uncertainty to the environment, where the agent is unable to directly measure the state, but instead has to form state estimations based only on a set of observations make tabular methods impractical. The actor-critic model, depicted in Figure 1, comprises of two neural networks. In the general case, the agent is assumed to have no a-priori knowledge of the environment. Both the actor and critic networks must form their own internal representation of the environment based on interactions and the reward received at each step. As in other reinforcement learning methods, the actor-critic model attempts to maximize the discounted expected return, $R(t)$, given by

$$\begin{aligned} R(t) &= r(t+1) + \gamma r(t+2) + ... \\ &= \sum_{k=1}^{\infty} \gamma^{k-1} r(t+k), \end{aligned} \tag{1}$$

where $r(t)$ denotes the reward received from the environment at time t and γ is the discount rate. The critic network is responsible for approximating this value, represented as $J(t)$. The critic network aims to minimize the overall error defined as

$$E_c(t) = \frac{1}{2} e_c^2(t), \tag{2}$$

where $e_c(t)$ is the standard Bellman error (Si, Yang, & Liu 2004),

$$e_c(t) = [r(t) + \alpha J(t)] - J(t-1). \tag{3}$$

The weight update rule for the critic network is gradient based. Let w_c be the set of weights in the critic network, the value of w_c at time $t+1$ is

$$w_c(t+1) = w_c(t) + \Delta w_c(t). \tag{4}$$

The weights are updated as

$$\Delta w_c(t) = l_c(t) \left[\frac{-\partial E_c(t)}{\partial w_c(t)} \right], \tag{5}$$

$$\frac{\partial E_c(t)}{\partial w_c(t)} = \frac{\partial E_c(t)}{\partial J(t)} \frac{\partial J(t)}{\partial w_c(t)}. \tag{6}$$

Similarly, the goal of the actor network is to minimize the term

$$\begin{aligned} E_a(t) &= \frac{1}{2} e_a^2(t), \\ e_a(t) &= J(t) - R^*. \end{aligned} \tag{7}$$

where R^* denotes the optimal return. Once again, weight updates are based on gradient-descent techniques and, thus, we have

$$\begin{aligned} w_a(t+1) &= w_a(t) + \Delta w_a(t), \\ \Delta w_a(t) &= l_a(t) \left[\frac{-\partial E_a(t)}{\partial w_a(t)} \right], \\ \frac{\partial E_a(t)}{\partial w_a(t)} &= \frac{\partial E_a(t)}{\partial J(t)} \frac{\partial J(t)}{\partial w_a(t)}, \end{aligned} \tag{8}$$

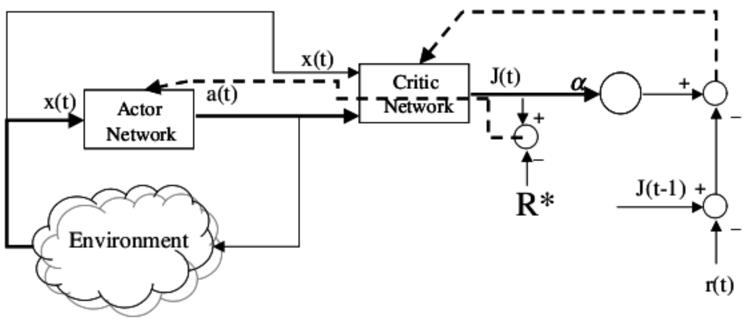

Figure 1: A diagram of an actor-critic model

where $l_a(t)$ is the learning parameter or step size of the actor network update rule. An online learning algorithm can now be derived from the previous equations. Starting with the critic network output, we have

$$J(t) = \sum_{i=1}^{N_{hc}} w_{ci}^{(2)}(t)p_i(t), \tag{9}$$

where N_{hc} is the number of hidden nodes for the critic network, and $p_i(t)$ is the output of node i given as

$$\begin{aligned} p_i(t) &= \frac{1-e^{-q_i(t)}}{1+e^{-q_i(t)}}, i = 1, ..., N_{hc}, &(10)\\ q_i(t) &= \sum_{j=1}^{n} w_{cij}^{(1)}(t)x_j(t), i = 1, ..., N_{hc}, \end{aligned}$$

where $q_i(t)$ is the input to hidden node i at time t. Applying the chain rule to (6) and substituting into (5) yields

$$\Delta w_{ci}^{(2)} = l_c(t)\left[e_c(t)p_i(t)\right] \tag{11}$$

for the output layer to the hidden layer nodes. Another expansion of (6) gives us

$$\begin{aligned} \frac{\partial E_c(t)}{\partial w_c(t)} &= \frac{\partial E_c(t)}{\partial J(t)}\frac{\partial J(t)}{\partial w_c(t)}, \\ &= \frac{\partial E_c(t)}{\partial J(t)}\frac{\partial J(t)}{\partial p_i(t)}\frac{\partial p_i(t)}{\partial q_i(t)}\frac{\partial q_i(t)}{}\partial w_{cij}^{(1)}(t) \\ &= e_c(t)w_{ci}^{(2)}(t)\left[\frac{1}{2}(1-p_i^2(t))\right]x_j(t). \quad (12) \end{aligned}$$

The actor network update rule is calculated similarly, as follows

$$\begin{aligned} a_i(t) &= \frac{1-e^{-v_i(t)}}{1+e^{-v_i(t)}}, i = 1, ..., N_{ha}, \\ v_i(t) &= \sum_{j=1}^{n} w_{aij}^{(1)}(t)g(t), i = 1, ..., N_{ha}, \\ g_i(t) &= \frac{1-e^{-h_i(t)}}{1+e^{-h_i(t)}}, i = 1, ..., N_{ha}, \\ h_i(t) &= \sum_{j=1}^{n} w_{aij}^{(1)}(t)x_j(t), i = 1, ..., N_{ha}, \quad (13) \end{aligned}$$

where v is the input to the actor node, g_i and h_i are the output and input of the hidden nodes of the actor network respectively, and $a_t(t)$ is the action output. Back-propagating from the output to the hidden layer yields

$$\begin{aligned} \Delta w_a^{(2)}(t) &= l_a(t)\left[\frac{-\partial E_a(t)}{\partial w_{a_i}^{(2)}(t)}\right], \\ \frac{\partial E_a(t)}{\partial w_{a_i}^{(2)}(t)} &= \frac{\partial E_c(t)}{\partial J(t)}\frac{\partial J(t)}{\partial w_{a_i}^{(2)}(t)}, \\ &= \frac{\partial E_a(t)}{\partial J(t)}\frac{\partial J(t)}{\partial a_i(t)}\frac{\partial a_i(t)}{\partial v_i(t)}\frac{\partial v_i(t)}{\partial w_{a_i}^{(2)}(t)} \\ &= e_a(t)\sum_{i=1}^{N_{hc}}\left[w_{ci}^{(2)}(t)\frac{1}{2}(1-p_i^2(t))w_{ci,n+1}^{(1)}(t)\right]\cdot \\ &\quad \left[\frac{1}{2}(1-u^2(t))\right]g_i(t). \quad (14) \end{aligned}$$

From the hidden layer to the input,

$$\begin{aligned} \Delta w_{a_{ij}}^{(1)}(t) &= l_a(t)\left[\frac{-\partial E_a(t)}{\partial w_{a_{ij}}^{(1)}(t)}\right], \\ \frac{\partial E_a(t)}{\partial w_{a_{ij}}^{(1)}(t)} &= \frac{\partial E_a(t)}{\partial J(t)}\frac{\partial J(t)}{\partial a_i(t)}\frac{\partial a_i(t)}{\partial v_i(t)}\frac{\partial v_i(t)}{\partial g_i(t)}\frac{\partial g_i(t)}{\partial h_i(t)}\frac{\partial h_i(t)}{\partial w_{a_{ij}}^{(1)}(t)} \\ &= e_a(t)\sum_{i=1}^{N_{hc}}\left[w_{ci}^{(2)}(t)\frac{1}{2}(1-p_i^2(t))w_{ci,n+1}^{(1)}(t)\right]\cdot \\ &\quad \left[\frac{1}{2}(1-u^2(t))\right]w_{a_i}^{(2)}(t)\cdot \\ &\quad \left[\frac{1}{2}(1-g_i^2(t))\right]x_j(t). \quad (15) \end{aligned}$$

It should be noted that the outputs of the actor network are nonlinear, unlike the linear outputs of the critic network. Recurrent elements of the networks can be treated as additional inputs to the network whose values are the hidden layer activations from the previous iteration. With this in mind, this same learning algorithm can be applied to recurrent networks.

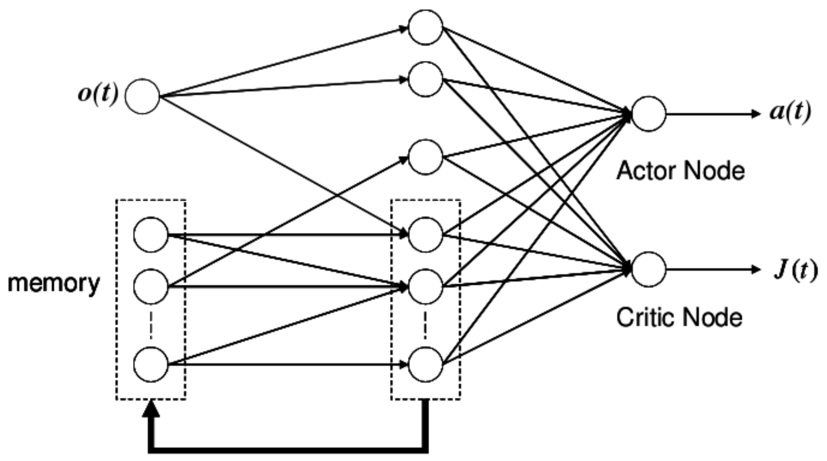

Figure 2: A Consolidated Actor-Critic Model.

The Consolidated Actor-Critic Model

The training of both networks in the traditional actor-critic model results in duplicated effort between the actor and critic, since both have to form internal models of the environment, independently. Combining the networks into a single network would offer the potential to remove such redundancy. Previous attempts to achieve such consolidation were limited to a probabilistic reflection of how an action should be modified in future time (Mizutani & Dreyfus 2004). However, such a scheme does not scale to large action sets, due to the exponential growth in the probabilistic term that would result. Here we proposes a generic, unified framework for true integration between the actor and critic networks in the context of temporal difference learning with function approximation. The consolidated actor-critic network (CACM) produces both state-action value estimates as well as actions. Moreover, the architecture offers improved convergence properties and more efficient utilization of resources.

Figure 2 illustrates the CACM architecture. The network takes a state s_t and an action a_t as inputs at time t, and produces a state-action value estimate J_t and action a_{t+1} to be taken at the next time step. The latter is applied to the environment and fed back to the network at the subsequent time step. The temporal difference error signal is defined by the standard Bellman error, in an identical way to that followed by the regular actor-critic model (3). Additionally, the action error is identical to that given in (7) for the actor network. The weight update algorithm for the CACM is gradient-based given by

$$\begin{aligned} E(t) &= E_c(t) + \frac{\partial E_a(t)}{\partial a(t)} \\ w(t+1) &= w(t) + \Delta w(t) \\ \Delta w(t) &= l(t)\left[-\frac{\partial E(t)}{\partial w(t)}\right] \end{aligned} \tag{16}$$

where $l(t) > 0$ is the learning rate of the network at time t. The output $J(t)$ of the CACM is given by (9) .The action output, $a(t+1)$ is of the form

$$a(t+1) = \sum_{j=1}^{n} w_{ai}^{(2)}(t) y_i(t) \tag{17}$$

where w_{ai} represents the weights between the i^{th} node in the hidden layer and the actor output node. Finally, we obtain $y_i(t)$ in a similar way to that expressed in (10).

To derive the back-propagation through the network expression, we first focus on the action error, which is a linear combination of the hidden layer outputs. Applying the chain rule here yields

$$\frac{\partial E_a(t)}{\partial a(t)} = \sum_{i=1}^{N_h} \frac{\partial E_a(t)}{\partial y_{i(t)}} \frac{\partial y_i(t)}{\partial x_i(t)} \frac{\partial x_i(t)}{\partial a(t)} \tag{18}$$

The weights in the network are updated according to

$$\Delta w_{ia}^{(2)}(t) = l(t)\left[\frac{-\partial E_a(t)}{\partial a(t)} \frac{\partial a(t)}{\partial w_{ia}^{(2)}(t)}\right] \tag{19}$$

for the hidden layer to the actor nodes, where

$$\begin{aligned} \frac{\partial E_a(t)}{\partial a(t)} \frac{\partial a(t)}{\partial w_{ia}^{(2)}(t)} &= \sum_{i=1}^{N_h} \frac{\partial E_a(t)}{\partial y_i(t)} \frac{\partial y_i(t)}{\partial x_i(t)} \frac{\partial x_i(t)}{\partial a(t)} \frac{\partial a(t)}{\partial w_{ia}^{(2)}(t)} \\ &= \sum_{i=1}^{N_h} \frac{\partial E_a(t)}{\partial J(t)} \frac{\partial J(t)}{\partial y_i(t)} \frac{\partial y_i(t)}{\partial x_i(t)} \\ &\quad \times \frac{\partial x_i(t)}{\partial a(t)} \frac{\partial a(t)}{\partial w_{ia}^{(2)}(t)} \\ &= e_a(t)(\sum_{i=1}^{N_h} w_{ci}^{(2)}(t)[\frac{1}{2}(1-y_i^2(t))] \\ &\quad \times w^{(1)}(t) y_i(t)) \end{aligned} \tag{20}$$

and $w_{ia}^{(1)}(t)$ denoting the weight between the action node and the i^{th} hidden node. Moreover, from the hidden layer to

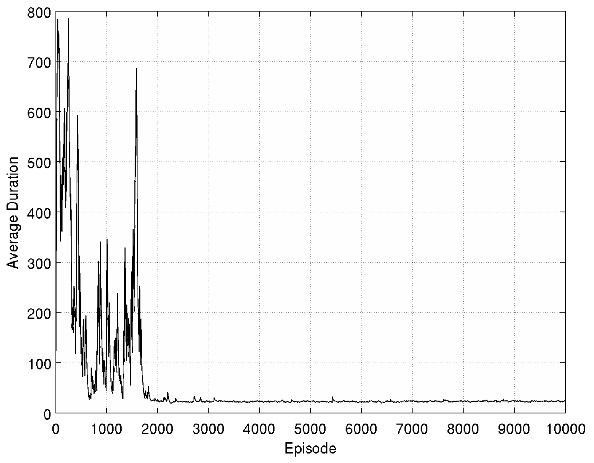

Figure 3: Average Duration vs. Number of Episodes

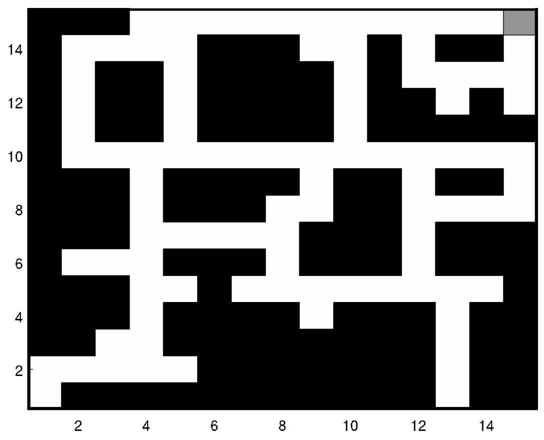

Figure 4: 15x15 Maze Used for Simulation

the critic node, we have

$$\Delta w_c^{(2)}(t) = l(t)[\frac{-\partial E_c(t)}{\partial w_{ic}^{(2)}(t)}], \tag{21}$$

where

$$\begin{aligned}\frac{\partial E_c(t)}{\partial w_{ic}^{(2)}(t)} &= \frac{\partial E_c(t)}{\partial J(t)}\frac{\partial J(t)}{\partial w_{ic}^{(2)}(t)} \\ &= e_c(t) y_i(t).\end{aligned} \tag{22}$$

Finally, for the inputs to the hidden layer, we express the weight update as

$$\Delta w_{ij}^{(1)}(t) = l(t)[\frac{-\partial E(t)}{\partial w_{ij}^{(1)}(t)}] \tag{23}$$

where

$$\frac{\partial E(t)}{\partial w_{ij}^{(1)}} = \frac{\partial E_c(t)}{\partial w_{ij}^{(1)}} + \frac{\partial E_a(t)}{\partial a(t)}\frac{\partial a(t)}{\partial w_{ij}^{(1)}}$$

$$\begin{aligned} = \ & [e_c(t) w_{ic}^{(2)}(t) \\ & + e_a(t)\left(\sum_{i=1}^{N_h} w_{ci}^{(2)}(t)\left[\frac{1}{2}(1 - y_i^2(t))\right] w_{ia}^{(1)}(t)\right) \\ & \times w_{ia}^{(2)}(t)][\frac{1}{2}(1 - y_i^2(t))]u_j(t)\end{aligned} \tag{24}$$

It is noted that the temporal difference nature of the action correction formulation resembles the one employed in the value estimation portion of TD learning. That is, information obtained at time $t+1$ is used to define the error correcting signals pertaining to time t.

Simulation Results

The CACM was simulated using a 15x15 cell maze problem where the objective is to reach the goal state in the minimum number of steps. The agent is randomly relocated within the maze after a successful completion. If the duration of any episode is more than 1000 time steps, it is ended and the agent is randomly relocated in the maze to start a new episode. The duration for each episode is recorded as a running average calculated as

$$D_{avg}(n) = 0.95 D_{avg}(n-1) + 0.05 D_{raw}(n), \tag{25}$$

where $D_{avg}(n)$ is the running average for episode n and $D_{raw}(n)$ is the duration of episode n, with $D_{avg}(0) = 0$.

The CACM was configured with 50 hidden neurons, all of which feedback for recurrency. The input to the network is a 51 feature vector consisting of 49 values for the observation, and two values for the action. Here, the observation consists of a subset of the maze that extends three cells in every direction from the cell currently occupied by the agent. There are four actions permitted at each moment: forward, backward, left and right. These are encoded in a two bit binary vector.

Since the network relies on recurrent connections to form the internal state estimations, training of any of the outputs did not occur for the first 6 steps of an episode. This bootstrapping of the memory neurons permits accurate formation of estimated state early in the episode . When additional memory is needed, extra layers can be added to this model, though extra delay will be needed at the beginning of each run to form complete context. Training was done using stochastic meta-descent with an initial learning rate of 0.005. Action selection was ϵ-greedy where

$$\epsilon(t) = 0.0413 + \frac{\log(t+1)}{22.9361}. \tag{26}$$

and t is the current time step within an episode, reflecting decreasing exploration with increased proficiency of the agent. The reinforcement signal r is +5 for transition into a goal state, -3 for an action that results in no state transition (i.e. moving into a wall), and 0 for all others.

Figure 3 depicts the average episode duration. The initial duration is large, with a hard limit set at 1000 steps due to the duration timeout. Spikes in duration toward the beginning can be attributed to the random relocation. Before the CACM is able to converge on the model on which to base the policy, multiple random placements in well known areas will result in the ”valleys”, and in less well known areas a significant spike. The system was able to identify the optimal policy in approximately 2000 completions, which is roughly 500,000 time steps into the simulation.

Mazes with dead ends take longer to converge. This is attributed to the random walk method being used in the code. The random walk will have a very small probability of getting out of dead ends at any one moment and can hold the agent in a single part of the maze for long periods of time. During this time, the agent is not able to receive multiple observations and is unable to form an accurate state estimation for the area. This leaves room for improvements to the exploration algorithm as future work.

Conclusions

This paper presents a novel and resource-efficient framework for addressing POMDPs in the context of temporal difference learning with function approximation. A Consolidated Actor-Critic Model was introduced in which a single recurrent neural network functionally replaces two independent networks, as commonly used in neuro-dynamic programming. This has been demonstrated through a partially observable maze navigation problem. Future effort will focus on scaling the proposed methodology to address more complex settings, such as robotics applications. Moreover, different methods of balancing exploration and exploitation in the CACM context will be studied.

References

Enns, R., and Si, J. 2003. Helicopter Trimming and Tracking Control Using Direct Neural Dynamic Programming. *Neural Networks, IEEE Transactions on* 14(4):929–939.

Marbach, P.; Mihatsch, O.; and Tsitsiklis, J. Feb 2000. Call admission control and routing in integrated services networks using neuro-dynamic programming. *Selected Areas in Communications, IEEE Journal on* 18(2):197–208.

Mizutani, E., and Dreyfus, S. 2004. Two Stochastic Dynamic Programming Problems by Model-Free Actor-Critic Recurrent-Network Learning in Non-Markovian Settings. *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on* 2:1079–1084 vol.2.

Si, J.; Yang, L.; and Liu, D. 2004. Direct Neural Dynamic Programming. In Si, J.; Barto, A. G.; Powell, W. B.; and II, D. W., eds., *Handbook of Learning and Approximate Dynamic Programming*. Institute of Electrical and Electronics Engineers. 125–151.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning : An Introduction*. The MIT Press.