Navigating a Robot through Big Visual Sensory Data

Altahhan, A.

Original citation:
Altahhan, A. (2015) Navigating a Robot through Big Visual Sensory Data . Procedia Computer Science, volume 53 : 478-485. DOI: 10.1016/j.procs.2015.07.325

http://dx.doi.org/10.1016/j.procs.2015.07.325

Procedia Computer Science

Volume 53, 2015, Pages 478–485

2015 INNS Conference on Big Data

Navigating a Robot through Big Visual Sensory Data

Abdulrahman Altahhan[1]
[1]*Coventry University, Coventry, U.K.*
abdulrahman.altahhan@coventry.ac.uk

Abstract
This paper describes a reinforcement learning architecture that is capable of incorporating deeply learned feature representation of a robot's unknown working environment. An autoencoder is used along with convolutional and pooling layers to deduce the reduced feature representation based on a set of images taken by the agent. This representation is used to discover and learn the best route to navigate to a goal. The features are fed to an actor layer that can learn from a value function calculated by a second output layer. The policy is ε-greedy and the effect is similar to actor-critic architecture where temporal difference error is back propagated from the critic to the actor. This compact architecture helps in reducing the overhead of setting up a desired fully fledged actor-critic architecture that typically needs extra processing time. Hence, the model is ideal for dealing with lots of data coming from visual sensor that needs speedy processing. The processing is accomplished off board due to the limitation of the used robot but latency was compensated by the speedy processing. Adaptability for the different data sizes, critical to big data processing, is realized by the ability to shrink or expand the whole architecture to fit different deeply learned feature dimensions. This added flexibility is crucial for setting up such model since the space dimensionality is not known prior to operating in the environment. Initial experimental results on real robot show that the agent accomplished good level of accuracy and efficacy in reaching the goal.

Keywords: actor-critic, reinforcement learning, big data, visual sensors, deep learning

1 Introduction

Robot visual navigation towards a goal is considered one of the special cases of navigation. This ability is essential and can be used as a building block for achieving general navigation capability by considering multiple goals navigation for example. Using vision is problematic due to various reasons; for example it is difficult often to get the same image quality under different conditions, such as lighting, and it is difficult to repeat the same snapshot with exact positioning. However, the most important difficulty is that visual processing is computationally intensive, and when it is required to be performed in every time step before any simple decision can be made to move left, right or straight, is very demanding. When running the robot in an environment, it can generate huge amount of visual data in short time. It is essential to process this data in order to realize the ability of navigation based on visual sensory input. In any environment, especially indoors, there could be a way to reduce the

Selection and peer-review under responsibility of the Scientific Programme Committee of INNS-BigData2015

doi:10.1016/j.procs.2015.07.325

dimensionality due to the confinement and repetition in the indoor structure. This reduction can be done by deep learning by quickly training the agent on the environment after looking around for short amount of time.

The ability to navigate towards a goal or home pertains to all animals naturally and is a must for most of the commercial and entertaining robotics application. Central to this ability is the skill of orienting towards the home and recognizing it once the agent is around it. Animals do that by wiring the surrounding visual memory somehow to their neural map of the environment. How they do that is yet to be discovered. Traditionally it has been linked to distinctive positions or places in the environment i.e. landmarks. However, the way animals do their navigation and find their home suggests something more subtle than only landmarks (Moller, 2005) (N. Tomatis, 2001). In this paper we propose a model that could be used as proof of concept that animals hard wire the scenes to itself and constantly compare it with the look of the home. It forms a frame of reference which is used to compare all information passed through to obtain internal map of the home as opposed to the different paths to its home.

2 Reinforcement Learning

2.1 Reinforcement Learning as a Framework

Reinforcement Learning performs well when is set up to interact in a novel situation where it is impractical to obtain a model of the working environment. Therefore, it might be suitable to model how an animal react, or plan, once it faces a new situation. RL does not require planning, but planning-like behavior comes as a natural consequence of it. For the learner everything can be summed up in terms of a reward signal (food, shelter etc.) and the sensory data that feed to it, in addition to the actuators that the animal needs to use in order to reach its target and maximize the long term rewards. Returning to its shelter is considered important urge in animals, and could make the difference between life and death in certain situations. Therefore, animals have developed a complex behavior around the shelter need and through evolution it becomes intrinsic in the brains of higher animals. To trace all that back one needs to understand how primitive rewards form skills that governs complex behavior.

2.2 Deep learning and RL

Deep learning has been shown to overcome the bottleneck representation problem that has long set back the success of machine learning applications (G. Hinton, 2006). RL takes relatively long time to converge due to the fact there is no direct answer to the input, the answer is just a signal that indicates how good or bad the current action is, and hence how good or bad the overall behavior is. Supervised learning has been showed to improve significantly when preceded with deep learning. When combined with RL it is also believed to have a good potential for shortening the convergence time. By combining the RL modeless capability with dimension reduction capability of Deep Learning, the overall effect of the model is hoped to have the capability of effectively dealing with big data. For navigation application the velocity and dimension of images the agent needs to process throughout its operations bear the challenges of big data processing. It is estimated that the agent process one image per two seconds hence it can process 2.2 Gigabyte per hour for jpeg files of 3.14 Kbyte.

3 The Model

3.1 Construction of Feature Maps

Our model starts by learning a concise and reduced feature representation. The model obtains a reduced representation in the first episode by randomly choosing a set of small patches form the images that were collected during the first exploratory episode. Then a set of features maps are learned via unsupervised learning to auto encode the patches. Those weights are then convoluted around each new image in order to extract interesting features that are translation invariant. Simple average pooling is then used to further reduce the set of features considered. Then the model shrinks the whole architecture to fit the new reduced number of features, and this concludes the deep learning phase for the feature representation layer. It should be noted that other methods could be better for object detection or recognizing a certain pattern such as a hand-written digits (Y. LeCun, 2004). However, for the purposes of our model we need something fast and low-level and we do not need to obtain features that could be used for recognizing an object. We just need a set feature that is good enough to distinguish the goal from other images.

3.2 Model Architecture and Components

In each consequent episode the model then takes advantage of the learned representation to convolve each timeframe image to deduce a set of convolved features that has more channels but less dimensions for each patch. Those convolved features are pooled together to further reduce the representation. This process is done in each time step in order to obtain a suitable reduced feature representation of the images instead of dealing with raw pixel information of each image. The architecture has been set up in an off board style, where in each time step, the sensory data is sent from the robot for processing in a remote machine, then the resultant action is sent back to the robot to be executed. This situation creates a latency issue that is typical mitigated by a speedy processing in the remote machine side. This scenario is hence ideal for testing whether the model is suitable for big data processing which can be judged by the practicality of running the robot and its efficacy in reaching the goal in a reasonable time. It should be noted that the architecture has the added advantage of adaptability and learning which is not normally available for traditional big data processing systems.

Next is the turn of actor-critic architecture (Tsitsiklis, 2000). After the actor layer takes its input form the feature pooling layer, it then decides to do a certain action, accordingly the critic layer punishes or rewards the actor depending on the reward it receives form the environment. Figure 1 shows an overall view of the model.

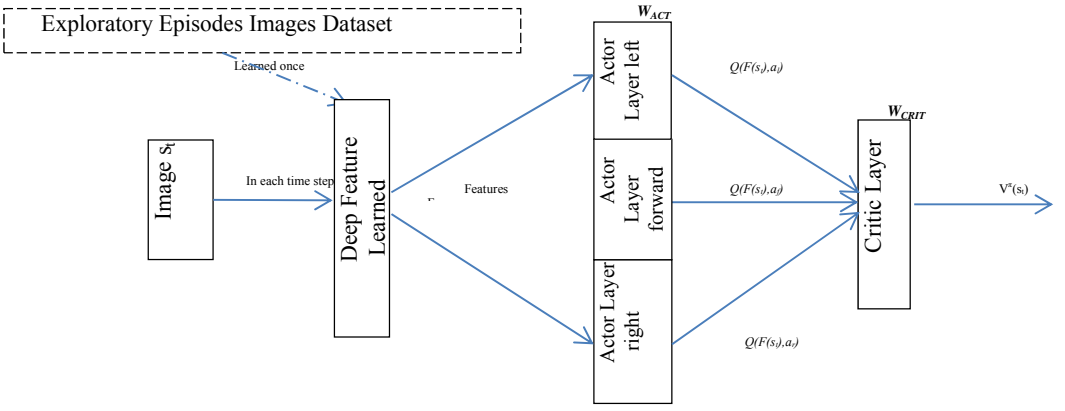

Figure 1. *The Model's Components.*

Formally, the presented model uses the following components/stages. Goal representation: As opposed to many models the goal or the home is represented by a set of snapshots taken for that location with the different orientation towards the goal. In fact the method used to identify the goal is transitionally invariant. Hence the goal location could indeed by identified by the agent from an angle different to the one it has originally taken from as we shall emphasize later by our stopping condition.

The dimension of the features ϕ_k is $d_1 \times d_2 \times 3$ where d_1 and d_2 are the dimensions of the images and 3 coming from having three channels. The dimension of the reduced features F_i is $n<d_1 \times d_2 \times 3$. The model uses an overall similarity measure that specifies the termination of the episode and is given by the mean of the normalized pooled feature

$$NRB(s_t) = \sum_{i=1}^{n} \vec{F}_i(s_t) \Big/ n \tag{1}$$

This measure has been used along with empirical thresholds to set the stopping and approaching conditions for the agent. The reward signal is calculated using the weights of the critic which is constituted of three parameters in accordance with the number of actions allowed in this model. These actions are forward, left and right respectively, where left and right have been set to equal speeds. The reward function is given as a combination of step cost in addition to a reward for going towards (approaching) the goal as well as a reward for reaching the goal itself (which is proportional to how fast the agent reach the goal in terms of number of steps) as in (Altahhan, 2011). Further a higher cost has been associated with turning actions, i.e. when the agent turns it will acquire higher costs than when it goes strait. This had the desired consequence of suppressing unnecessary turns and emphasizing going strait. Also a punishment for taking any action that leads to a reactive behavior has been set. This is also to help reduce the costal behavior and to encourage going directly towards the goal. Figure 1 shows an overall view of the model and its components.

3.3 Deep Blended Actor-Critic Architecture

By setting the second layer to three parameters; one for each action and calculating the error signal for the Actor layer (which is responsible for taking the actions; its decision is the one that is being carried out). And by allowing this second layer to act as a critic that contemplates the consequences of the actions of the actor layer and sends a signal to it to indicate how well is its current policy. And by making the two layers to work as hidden and output layers of a one neural network, we are creating deep blended actor-critic architecture in one sound system that depends on two eligibility traces. The critic layer itself is taking its feedback form the reward function. In that sense the actor-critic layered architecture is deep in terms of learning action representation in the same sense that a deep learning network is learning deep representation structure. Therefore, this model is deep in terms of its feature representation and in terms of its action representation. All training is done using backpropagation.

Formally, the updates are given by:

$$\Delta w_t(i) = \alpha_t \delta_t e_t(i) \tag{2}$$

$$\Delta \theta_t(i,k) = \alpha_t \delta_t e_t(i,k) \tag{3}$$

Where

$$e_t(i) = \gamma \lambda e_{t-1}(i) + \frac{\partial V_t(s_t)}{\partial w(i)}$$
$$\vec{e}_t''(k) = \gamma_t' \lambda_t \vec{e}_{t-1}''(k) + \frac{\partial V_t(s_t)}{\partial \vec{\theta}_t(k)} \tag{4}$$

Since we are dealing with two layered network each using logistic transfer function we have

$$e_t(i) = \gamma\lambda e_{t-1}(i) + Q_t(\vec{F}_t, a_i) \tag{5}$$

$$e_t(i,k) = \gamma\lambda e_{t-1}(i,k) + w_{(i)} Q_t(\vec{F}_t, a_i)\left(1 - Q_t(\vec{F}_t, a_i)\right) F_{t,(k)} \tag{6}$$

Finally the policy has been formulated as a simple greedy policy.

It should be noted that eligibility traces in reinforcement learning framework is similar to the momentum for supervised learning. It constitutes a way to accommodate previous updates into current updates to guide the search for the local optima. In RL it traces blame of current decision back to older decisions that lead to the current situation.

In addition, two regularizes have been multiplied by the two parameter sets to discount the old values of the parameters (hence prevent ovefitting). Also the updates have been committed in an online but in a semi-patch fashion. This means that for every M number of steps the weights is updated. Finally at the end of each episode also the same learning has been done by committing the patch updates of the whole episode. So the model blends patch update, online and offline learning.

4 Experimental Results

The robot was let to run for 50 episodes. Each episode starts by going from any location in the environment to the goal/home location. The size of the robot makes it relatively easier to run it form different locations. Hence, it was allowed to run for a 500 steps before the episode is considered a failure.

4.1 Hardware and Software Settings

Figure 2 shows the used robot and its environment. It is basically an updated version of Lego Mindstorm that has been used with additional camera module and processing unit that was mounted and attached on top of it. This robot has low level of sophistication in terms of the motor commands, balance, senor reading as well as its shape. Yet the results were good, so it is expected to obtain higher performance once a finer robot it used. The Raspberry Pi has been powered by an off-the-shelf 1000AMP chargeable battery that was placed underneath the NXT brick. Matlab have been used throughout the model in the form of a set of library functions that have been written specifically for this model. In addition the RWTH- Mindstorms NXT Toolbox for MATLAB has been used to provide the sensory reading and the actuator commands form the NXT robot to Matlab, only the very basic Direct Motor Command were used which set up the speed of a specific motor without any further calculations.

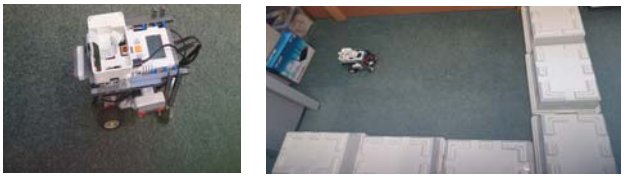

Figure 2. *Left:A snapshot of the built robot with its sensors, actuators, and camera module. Right: The training envirnment.*

4.2 Model Hyper Parameters Settings

It should be noted that the robot abducts itself after each successful/unsuccessful episode by following a rigid set of backwards steps that formed a T-shape so that it is as far and disoriented form the goal as possible. Each episode stops by either reaching the goal (successful) or by exhausting the

allowed budget (number of steps) that the agent is given (unsuccessful). In the case of the presented work it was 500 steps. The settings of the model hyper parameters are shown in Table1

Symbol	Value	Description
Max_e	50	Number of episodes in each run
α_0	0.1	Initial learning rate
ε_0	$0.3\times EP$	Initial exploration rate
ep_0	$0.3\times EP$	Start episode for decreasing α and ε
γ	1	The reward discount factor
m	1	Number of snapshots of the home
b	2	Features histograms bin size
$\psi_{upper}, \psi_{lower}$	0.88 0.87	Goal_at_perspective thresholds
λ	0.8	Eligibility trace discount
Max_steps1	500	steps before agent considered unsuccessful

Table 1: Hyper parameters values and description

Images with resolution of 160x120 were send form the Raspberry PI via wireless network adaptor to an off-board computer for processing where learning is taking place, then the required commands is sent to the actuators of the robot via Bluetooth.

4.3 Convergence and Performance

Figure 3 shows the deep learning stage for 1000 patches that have been taken form various images of the environment and that has been used to train a autoencoder for 30000 epochs. It can be realized that after the first 10000 epochs there was very small improvements. Still these sometimes could make a difference when the high dimensionality of the feature space is considered.

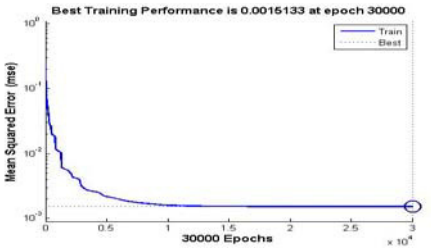

Figure 3. *Deep Learning Performance for 1000 patchs and 30000 epochs*

Figure 4 shows an intermediate stage where the robot was still learning. The number of episodes (upper right corner) is envisaged (as was evident in the simulation in [5]) to show a pattern of convergence towards minimal number of step if the robot where left to run for a very long time. However, due to the time and physical constraints, this was difficult to do. Hence it has been let to run for a limited number of episodes Nonetheless, it should be noted that the number of steps varied between episodes due to the abduction of the robot to a random location after each episode, which resulted near/further position from the goal. Nonetheless, the figure illustrates convergence, which is evident indeed. Figure 4 shows the learned parameters, a tendency towards turning left is developed by the agent through the left bottom critic parameters, which is what is expected when operating in an open plan. It should be noted however that the agent did not just always turned left, the behavior depends on the current image.

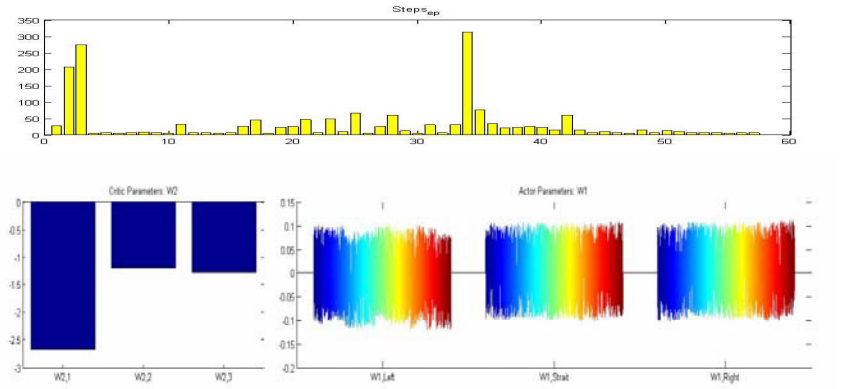

Figure 4. *The model learned parameters and episodes number of steps*

5 Conclusion

Our results show that out of 57 times (50 trainings + 7 testing) it confused the goal 4 times. This has been verified by looking into what the robot has registered as a target in each episode. Hence, accuracy in recognizing and reaching the goal during (training and testing) is 1-4/57 ≈ 93% since the agent missed 4 times. The features are learned through an autoencoder architecture that is set to learn a deep reduced representation of a big number of fixed-size random patches taken from all the snapshots of an exploratory episode. As opposed to many models the goal or the home is represented by a set of six snapshots taken for that location with the desired orientation of the robot. The model uses double deep learning for both feature representation and action learning, which set its distinctive novelty. How practical is it, will be for future work to verify. Also it is intended to show some other interesting properties of the model such as convergence and the relationship between deep feature learning and deep action learning. The processing is accomplished off board due to the limitation of the used robot but latency was compensated by the speedy processing. The model constitutes a proof of concept that it is suitable for processing lots of visual sensory data that is grounded in the agent memory through deep learning.

References

Adam Coates, H. L. (2011). An Analysis of Single-Layer Networks in Unsupervised Feature Learning. *AISTATS* , 14.

Altahhan, A. (2011). A Robot Visual Homing Model that Traverses Conjugate Gradient TD to a Variable λ TD and Uses Radial Basis Features. In A. Mellouk, *Advances in Reinforcement Learning* (pp. 225-254). Vienna: InTech Education and Publishing.

Andrew Ng, J. N. (2010). *Online Tutorial in Deep Learning: Stanford University*. Retrieved 2015, from http://ufldl.stanford.edu/tutorial/

Barto, R. S. (1998). *Reinforcement Learning, an introduction.* Cambridge, Massachusetts: MIT Press.

C. Zhang, S. A. (2008). Efficient multi-agent reinforcement learning through automated supervision. *International Conference on Autonomous Agents.* Estoril, Portugal.

Carlisle, D. (2010, April). *graphicx: Enhanced support for graphics.* Retrieved from http://www.ctan.org/tex-archive/ help/Catalogue/entries/graphicx.html

G. Hinton, S. O. (2006). A fast learning algorithm for deep belief nets. *Neural Computation,18(7)*, 1527–1554.

Moller, V. a. (2005). Biologically plausible visual homing methods based on optical flow techniques. *Connection Science, vol. 17*, 47–89.

N. Tomatis, I. N. (2001). Combining Topological and Metric: a Natural Integration for Simultaneous Localization and Map Building. *Fourth European Workshop on Advanced Mobile Robots (Eurobo)*.

N.Shimkin, O. Z. (2005). Multigrid Methods for Policy Evaluation and Reinforcement Learning. *IEEE International Symposium on Intelligent Control.* Limassol.

P. Vincent, H. L. (2008). Extracting and composing robust features with denoising autoencoders. *ICML*.

S. Bhatnagar, R. S. (2007). Incremental Natural Actor-Critic Algorithms. *Neural Information Processing Systems (NIPS19).*

Tsitsiklis, V. K. (2000). Actor-Critic algorithms. *NIPS 12.*

Voronkov, A. (2004). *EasyChair conference system*. Retrieved from easychair.org

Y. LeCun, F. J. (2004). Learning methods for generic object recognition with invariance to pose and lighting. *CVPR*.

Zeil, J. (April 2012). Visual homing: an insect perspective. *Current Opinion in Neurobiology, Volume 22, Issue 2, ISSN 0959-4388*, 285-293.