**Direct Policy Search Approach for Deep Reinforcement Learning**

*Daniel Elstob*

*Liverpool Hope University*

[*14011862@hope.ac.uk*](mailto:14011862@hope.ac.uk)

**1. Introduction**

Reinforcement learning is one of the most used for reward related learning problems in machine learning and is one of the most promising artificial intelligence paradigms for the future development of autonomous robots. It is “learning what to do – how to map situations to actions – so as to maximize a numerical reward signal” (Sutton and Barto, 1998) and the “two characteristics-trial-and-error search and delayed reward-are the two most important distinguishing features of reinforcement learning” (Sutton and Barto, 1998).

Basically reinforcement learning is the act of learning by interacting with an environment. An agent will make an action and learn from the response rather than from being explicitly taught like in supervised learning. The agent will then choose future actions based on past experiences. The agent goal is to maximize the cumulative reward. This sequential decision making needs to take into account a delayed reward as an action may not receive a reward until much later or even if that reward is positive value or negative value. A future state is dependent on the action taken in the current state. An example has been illustrated in Figure 1.

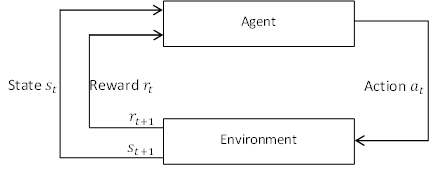


Figure 1 Reinforcement Learning Process

However, according to Peters and Schaal “most of the methods proposed in the reinforcement learning community to date are not applicable to robotics as they do not scale beyond robots with more than one to three degrees of freedom” (Peter and Schaal, 2006), one exception to this is policy gradient methods.

**2. Aims and Objective**

This paper will look into a direct policy search for deep reinforcement learning and apply this to a scenario in Section 5. It will show the process of this designing an algorithm based of Section 4 requirements in order to implement such a system, as well as looking into the limitations to the chosen algorithm and the potential improvements. In Section 6 this paper will discuss other similar research and real world application that deep reinforcement learning can be applied to.

**3. Background**

Direct policy search is not a reactionary function were, based on the state, the agent will pick an action, but instead learns a function that will tell it how, based on observation of the state, which action it should choose, and this process is a policy. And it is learning directly because it will optimise the return. It essentially computes a derivative of the function and follows the gradient that will give it the policy.

**4. Scientific Computing**

When a researcher wants to solve a real-world problem efficiently on computers they will use scientific computing to aid in the development and use numerical methods and mathematical models that they will apply to the problem. Scientific computing allows for models to be built simulating the process, so that when it is finally applied to the problem the computer scientist is better prepared to understand what is being observed. It is useful for simulations to be created for when the situation is impossible such as weather forecasting or too costly to create for example crash tests and aerodynamics.

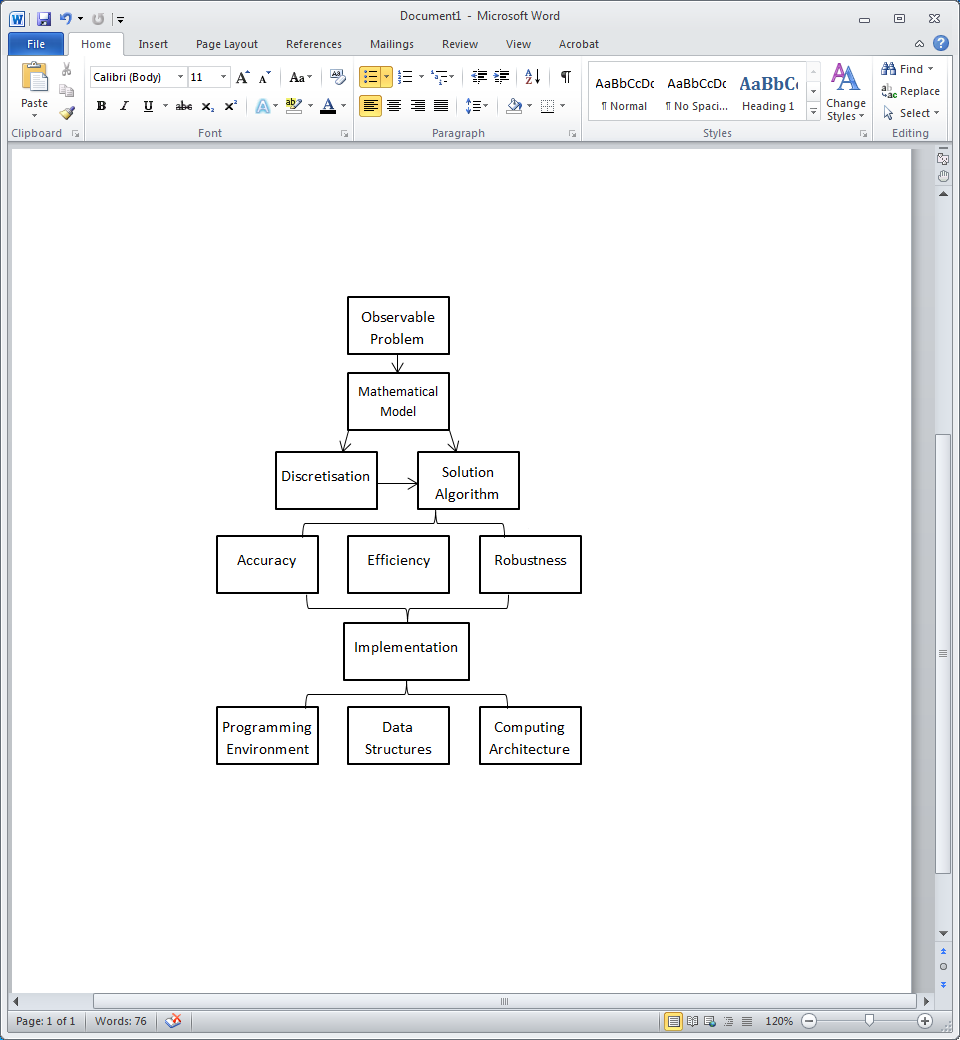


Figure 2 Algorithmic Design Theory

When trying to solve a problem the computer scientist will often follow the algorithmic design theory which is the process followed to create, analyse and implement a solution as illustrated in Figure 2. Starting with the observable problem they will create mathematical models whether that is psuedocode or algorithms that do not yet have full functionality. Issues when deriving a mathematical model starts with what the given task is for example solving a problem or optimisation of a function. The user will need to think about what quantities are of influence and how important are they. There are also issues when analysing the mathematical model, does a solution already exist or is it unique problem.

Depending on the size of the problem they may need to break it down into smaller problems. This is an optional phase called discretisation which is similar to the Divide and Conquer algorithm, in which they break down and individually solve before putting then back together to form a solution.

At this point from discretisation or moving directly from the mathematical model phase there should be a solution algorithm that can be analysed with the three criteria accuracy, efficiency and robustness. Accuracy is about how well the solution algorithm has solved the problem either in a single run or possibly over iterations of testing. Efficiency involves how much time and space is used. Time could be how fast the solution solved the problem. Space could be how much memory is used to run the algorithm. Robustness is how well the solution handles errors during execution or how it handles abnormal data input.

If after these steps the problem is not solved or the solution failed a part of the analysis then it is possible to go back and reiterate the process, however if the solution solved the problem or passed the analysis to a satisfactory level then move onto implementation. Implementation is comprised of three areas, programing environment which is the programming language and EDE, data structures that are used and the platform architecture that the solution will be implemented onto.

**5. Case Study**

**5.1 Statement of the Problem**

On many websites there is a CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) first patented by Lillibridge et al (Lillibridge et al, 1998) and later improved upon when the term was coined by Luis von Ahn et al (Ahn et al, 2003). Through many research papers it was proven that these methods were not sufficient to protect against these computer vision systems (Mori et al., 2003). Alternatives were proposed in the form of image recognition which would be more challenging than character recognition (Chew et al., 2004, Datta et al., 2005, Elson et al., 2007). The popular choice of this is an image, usually of a house number, followed by a character CAPTCHA which can still be solved with computer vison. We will attempt to show how this can be done using reinforcement learning techniques.

Now with reinforcement learning there is an agent and a state. In this example of direct policy search the agent is a recurrent neural network and the state will be the history, but it won’t be all the data just glimpses of the data as it needs to have a finite memory.

**5.2 Theoretical Solution**

To start there is a history “*h*”, which consists of a sequence of observations “*O*”, over time “*t*”, resulting in “”. And then we introduce a policy Eq. (1).

(1)

Equation (1) is a policy that maps the history to the action and given that it has seen all these observations it will choose the next action based on them. For the next step we make the assumption that on the condition of these histories, the actions are independent. The policy of “*T*” decisions is going to factor as a product of the decisions, making them all independent given the history. If history is not given they become coupled, so we need policy “” of action “” and of history “”. Now we need a reward “”, which results in Eq. (2).

(2)

The idea here is to maximise the reward in the future. For example if there are “*T*” steps the reward might be 0, 0, 0, 0, 0, 1 with 1 being the reward signal which in this case is weak. This means we need to design a policy to take that into account, and that there might be immediate rewards that are of value in the present state but won’t be in the long term, so we need it to learn that it’s better to forgo immediate reward in the hope that it will receive a larger reward at the end. Therefore what we want to do is, on expectation, maximise the reward or in other words regardless of what the future actions will be for all possible action sequences we want to come up with the best possible rewards, and that’s why we marginalise “” to get the best variable “*a*”.

**5.3 Practical Implementation**

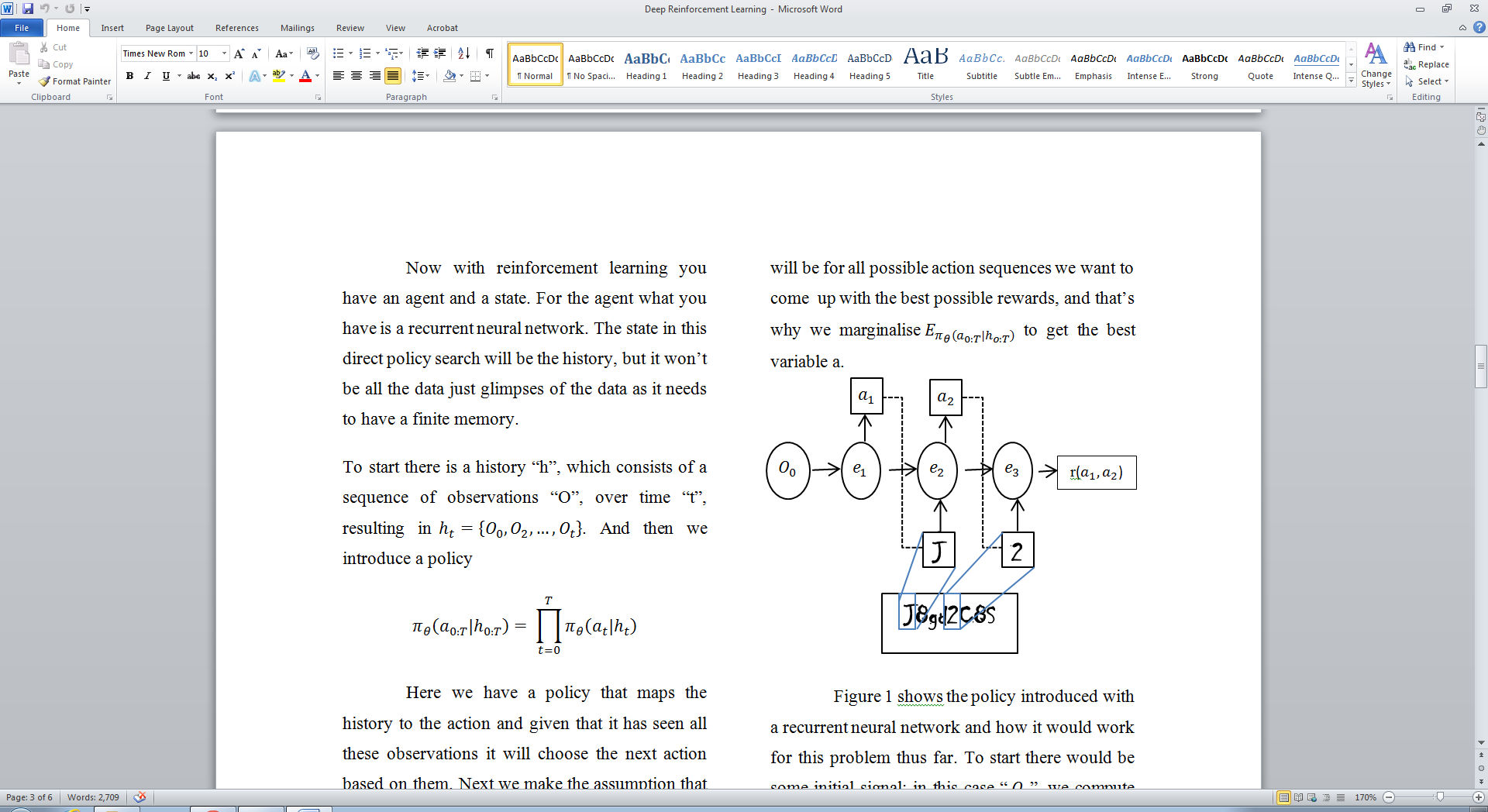


Figure 3 Example Implementation for CAPTCHA Recognition

Figure 3 illustrates an example of how the policy, introduced with a recurrent neural network, would work for a character CAPTCHA problem thus far and would similarly work for an image based CAPTCHA. To start there would be some initial signal; in this case “”, we compute some hidden embedding “”, and from the embedding we would choose to output an action “”. One way of outputting an action could be to output a softmax of probability of a given “e”, in other words “”.

For example if we had a neural network with five outputs, we would sample according to that distribution to choose an action. That “” then deterministically will choose an observation, and this observation is being drawn from the database, perhaps using a foveation model, which will then be fed into the next embedding layer of the recurrent neural network and the process repeats until there is a reward signal, which in this case is of the sequence “ ”.

Another way to think of this is that it is a function that takes “*O*”’s as and input and outputs “*a*” and has some parameter “”. Therefore in order to decide what action to take we need to learn “”. Thus “” is basically, at each time step, a policy that is like a discrete distribution over a number of actions and we need to learn the parameters of the network because the quantities given by the softmax will depend on the neural network parameters.

Now the only thing that’s missing in this setup is we need a way to learn the parameters. And we need it to work with potentially delayed rewards which we can do because we have an objective to maximise rewards when we follow the policy “”. To do this we would maximize the expected returns of “” and were done, that’s a direct policy search.

**5.4 Quality Assurance**

At this point we can test the equation and an issue forms, we need an expectation of all our actions “” and if our actions are binary and there are 10 decision steps then it will have terms in the sum, or if this we even larger such as it is not possible to calculate. We can rewrite this by taking the gradient with respect to “”, and because the reward is not dependant on “” we can move the gradient inside and we will only have to take the derivative with respect to “” which results in Eq. (3).

(3)

In order to calculate the derivative of the cost “”, we can optimise, for example with a greedy stochastic descent, but first let simplify the equation. Because this is a large sum we will use an approximation, in this case sampling, we will act in the world and see what happens. Thus far we have only seen sampling using distribution not derivatives. We can use a trick here to help and that is that. In other words is the derivative of the log which gives us Eq. (4).

(4)

With this knowledge we have put log of distribution in the equation and have the derivative of the log distribution to draw samples from. This creates a sum of each individual action. We are going to do one more simplification and that is instead of having an index for the reward that goes from “0” to “*t*” we are going to have an index from time “*t*” into the future presented in Eq. (5). The idea behind this is that future actions do not affect the past rewards, meaning the next action will only affect what action is done in the future so we only need to look at the rewards that go in the future.

(5)

Now we create the final version which involves the reward, the whole policy and also the derivative of the policy at each time step, Eq. (6), which is essentially the output of the recurrent network which we can compute with backpropagation.

(6)

What we do next is sample a trajectory “”. What we mean by this is that we simulate a recurrent network, we observe an image and take one action, and then we grab some data and take another action. Basically we use the current set of the parameters and let the neural network take action. Then we get a measure of performance, and error, how well we did at classifying the numbers, did we get the numbers right or not. And based on that measure of performance we can put it in “” which will give us a new version of the gradient, Eq. (7).

(7)

Equation (7) is the final version of and to explain this final version, if we drew sample from this distribution “”, then the Monte Carlo estimate involves the rewards that we will accrue along the trajectory “” and the derivatives that we compute by backpropagation “”. And these are the samples we try “N” times “”

**5.5 Conclusion**

**5.5.1 Limitation**

One of the issues with this approach is that it is very high variance and so people will devote most of their effort to coming up with low variance estimates of this gradient. Part of why it is high variance is that we are dealing with derivatives of distribution and often when we use samples to approximate it might have positive samples, some negative samples cancelling them out, causing a lot of waste of samples when trying to compute derivatives

**5.5.2 Improvements**

To improve this is that a group of people create a large class of approximation methods. One of the very useful tricks is to, instead of using distribution, try to break the distribution into the deterministic part and the stochastic part and then it only needs to simulate the noise and then use back propaganda through the deterministic part.

**5.5.3 Relate Work**

A few papers have applied this approach to autonomous helicopters (Bagnell & Schneider, 2001) as well as on using direct policy search to successfully fly a helicopter upside down (Ng et al., 2004). A related paper by a Google team in which they use deep convolution neural networks to recognise number from street view (Goodfellow et al., 2014) to help with giving directions.

**6. Critical Reflection**

This paper has investigated direct policy search for deep reinforcement learning, discussing the key requirements to designing, analysing and implementing such an algorithm to a real world example that required image recognition. There are many methods of image recognition that could have been used in the case study, this paper approached it with the idea that reinforcement learning would be a great way to solve the problem as it wouldn’t require any pre-set data and could simply learn over time what the best actions are with the aid of deep learning which had recent breakthrough in the computer vision field. Direct policy search as its name suggests allows the system find a good policy by searching the policy space directly which causes the problem becomes an instance of stochastic optimisation.

Instead of using direct policy search another approach could have be to use neuro-dynamic programming. With this method replace the dynamic programming with a neural network which works as a function approximate, then choose the action according to the policy and update the parameters. To get the policy, convert the value function using the Bellman equation.

With the new interest of deep learning in the reinforcement community and the combination to create deep reinforcement learning this would allow for a much more powerful system that could solve the problem better. Most recently a group of researchers used “a new deep learning model for reinforcement learning, and demonstrated its ability to master difficult control policies for Atari 2600 computer games, using only raw pixels as input” (mnih et al., 2013). As mentioning in Section 5.5.3 Google has had an interest in this field and its Street View computer vision can be used to beat image based CAPTCHA with “99.8% accuracy on the hardest category of reCAPTCHA.” (Goodfellow et al., 2014).

In conclusion the research opportunities to create powerful systems that are capable of solving complex or computer vision problems with the possibility of being more efficient and having a higher accuracy than that of humans.

**7. References**

Ahn, L., Von, Blum, M., Hopper, N., and Langford, J. (2003). *CAPTCHA: Using hard AI problems for security*. In Proceeding of Eurocyrpt, 2003.

Bagnell, J., and Schneider, J. (2001). *Autonomous helicopter control using reinforcement learning policy search methods*. In Int’l Conf. Robotics and Automation. IEEE, 2001.

Chew, M. and Tygar, J., D. (2004). *Image Recognition CAPTCHAs.* In Proceeing of the 7th International Information Security Conference (ISC 2004), Springer, 2004. pp. 268-279.

Datta, R., Li, J., and Wang, J., Z. (2005). *IMAGINATION: A Robust Image-based CAPTCHA Generation System.* In ACM Multimedia Conf. 2005

Elson, J., Douceur, J., R., Howell, J., and Saul, J. (2007). *Asirra: A CAPTCH that Exploits Interest-Aligned Manual Image Categorization*. In Proceedings of 14th ACM Conference on Computer and Communications Security (CSS). Association for Computing Machinery, Inc.

Goodfellow, I., J., Yaroslav, B., Ibarz, J., Arnoud, S., Shet, V. (2014). *Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks*. arXiv preprint arXiv:1312.6082, 2013.

Lillibridge M., D., Abadi, M., Bharat, K., and Broder, A., Z. (1998). *Method for selectively restricting access to computer systems*.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). *Playing atari with deep reinforcement learning*. *arXiv preprint arXiv:1312.5602*.

Mori, G., and Malik, J. (2004). *Recognizing Objects in Adversarial Clutter: Breaking a Visual CAPTCHA*. IEEE Conf. on Computer Vision and Pattern Recognition, I:184-190, 2005.

Ng, A., Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., and Liang, E. (2004). *Inverted autonomous helicopter flight via reinforcement learning*. In International Symposium on Experimental Robotics, 2004.

Peters, J., & Schaal, S. (2006). *Policy gradient methods for robotics*. In Proceedings of the IEEE/RSJ international conference on intelligent robots and systems. pp. 2219–2225.

Sutton, B., & Barto, A., G. (1998). *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA, USA