Scholars Bulletin ***(Engineering)***
An Official Publication of "Scholars Middle East Publishers"
Dubai, United Arab Emirates
Website: www.saudijournals.com
ISSN 2412-9771 (Print)
ISSN 2412-897X (Online)

Requirement of Python Programming Language for Reinforcement Learning (RL)

Varun Geetha Mohan[1], Mohamed Ariff Ameedeen[1,2]
[1]Faculty of Computer System and Software Engineering, Universiti Malaysia Pahang, Malaysia
[2]IBM Centre of Excellence, Universiti Malaysia Pahang, Malaysia

***Corresponding author**
Mohamed Ariff Ameedeen

Article History
Received: 07.02.2018
Accepted: 19.02.2018
Published: 28.02.2018

DOI:
10.36348/sb.2018.v04i02.010

Abstract: In this article we discussed on why Python require in Reinforcement Learning(RL). Python languages have a unique ecosystem, cultures and have their own philosophies. Python codes are available as open sources and Python community involvement with local, national and international events and it is helpful for new developers. Python have its wide array of open source code libraries, package management and ability to work well on platforms other than Windows OS. Finally, Python is great for deployment automation and web development and many non-developers are first introduced to the language and ecosystem while getting data analysis work done.

Keywords: Machine Learning, Supervised Learning, Unsupervised Learning, Reinforcement Learning, Python, Markov Decision Process.

INTRODUCTION

We are most probably living in the most defining period of human antiquity is the time when computing moved from large mainframes to PC to cloud. But what makes it defining is not what has happened, but what coming our way in years to come. One of the most fundamental questions for scientists across the globe has been – "How to learn a new skill?". The desire to understand the answer is obvious – if we can understand this, we can enable human species to do things we might not have thought before.

Alternately, we can train machines to do more "human" tasks and create true artificial intelligence [1]. While we don't have a complete answer to the above question yet, there are a few things which are clear. Irrespective of the skill, we first learn by interacting with the environment. Whether we are learning to drive a car or whether it an infant learning to walk, the learning is based on the interaction with the environment. Learning from interaction is the foundational underlying concept for all theories of learning and intelligence [2].

Reinforcement Learning (RL) alludes to both a learning issue and a subfield of machine learning. It alludes to figuring out how to control a framework to augment some numerical esteem which speaks to a long-term objective. RL is an extremely natural and complete answer for autonomous decision making. As people utilize comparative approach for day by day life choices demonstrate that the field has a great deal of potential for some applications, and the best of them is Robotics [3, 4]. A typical setting where reinforcement learning works is shown in Figure 1: A controller gets the controlled system's state and a reward related with last state change. It at that point figures an activity which is sent back to the system. Accordingly, the framework makes a change to another state and the cycle is rehashed. The issue is to take in a method for controlling the framework to boost the aggregate reward. The learning issues vary in the points of interest of how the information is gathered and how execution is estimated [5].

The estimations accessible on the framework's state are nitty sufficiently gritty with the goal that the controller can abstain from thinking about how to gather data about the state. Issues with these attributes are best depicted in the structure of Markovian Decision Processes (MDPs). The standard way to deal with 'illuminate' MDPs is to utilize dynamic programming, which changes the issue of finding a decent controller into the issue of finding a decent esteem work. Aside from the easiest situations when the MDP has not very many states and activities, dynamic writing computer programs is infeasible [6, 7]. The Python codes using in RL can be as a way of turning the infeasible dynamic programming methods into practical algorithms so that they can be applied to large-scale problems. Python's expansive library of open source data analysis tools, web frameworks and testing instruments make it ecosystem one of the largest out of any programming community. Python is an accessible language for new programmers because the community provides many introductory resources [8, 9].

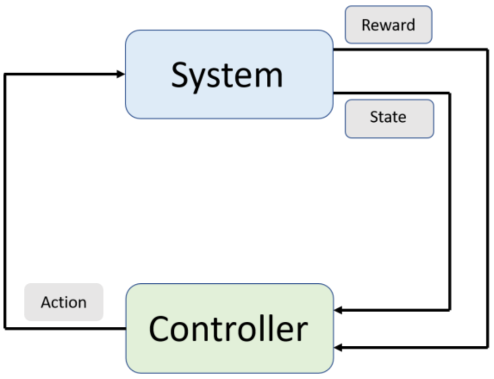

Fig-1: The basic reinforcement learning scenario

Various ML applications include tasks that can be set up as strengthened or reinforced. In the present paper, we have focused on the highlights important and specifically, this work is concerned about grouping and their problems and even their characteristics moreover. We have restricted our references to recent refereed journals, published books and conferences. In option, we have included a few references with respect to the original work that began the specific line of research under exchange. The peruse ought to be advised that a solitary can't be an exhaustive review of all grouping algorithms. Rather, our objective has been to give an agent test of existing lines of research in each learning strategy. In each of our listed regions, there are numerous different papers that more extensively detail important work.

The next section covers comparison of reinforcement learning with supervised and unsupervised learning. The features of Python are described in section 3 and finally, last section concludes this review paper.

Comparison of reinforcement learning with other machine learning methodologies

Reinforcement Learning is of great interest because of the large number of practical applications that it can be used to address, ranging from problems in artificial intelligence to operations research or control engineering. Reinforcement learning methods specify how the agent changes its policy from the result of experience. Roughly, the agent's goal is to get as much reward as it can over the long run. By using this machine learning algorithm, the machine gets trained to make specific decisions. That means, the machine is exposed to an environment where it trains itself continuing trial and error. However, machine learns from the experience and tries to capture the best knowledge to make accurate business decisions [10].

Reinforcement Learning belongs to a bigger class of machine learning algorithm. Let's see a comparison between RL and others.

Supervised vs Reinforcement Learning

The objective of supervised learning is to create an incisive model of the distribution of class labels in terms of forecaster features. Supervised machine learning is the explore for algorithms that basis from externally supplied instances to produce prevalent hypothesis, when then construct predictions about future illustrations. The creating classifier is then used to allot class names to the testing occurrences where the estimations of the forecaster highlights are known, however the estimation of the class mark is obscure. This machine learning algorithm consists of a target or dependent variable which is to be predicted from a given set of predictors or independent variables. Using these set of variables, we originate a function that map inputs to desired outputs. The instruction process continues until the model achieves a desired level of accuracy on the instruction data.

In supervised learning, there's an external "supervisor", which has knowledge of the environment and who shares it with the agent to complete the task. But there are some problems in which there are so many combinations of subtasks that the agent can perform to achieve the objective. So that creating a "supervisor" is almost impractical. For example, in a chess game, there are tens of thousands of moves that can be played and creating a knowledge base that can be played is a tedious task. In these problems, it is more feasible to learn from one's own experiences and gain knowledge from them. This is the main difference that can be said of reinforcement learning and supervised learning. In both supervised and reinforcement learning, there is a mapping between input and output. But in reinforcement learning, there is a reward function which act as a feedback to the agent as opposed to supervised learning [11, 12].

Unsupervised vs Reinforcement Learning

Unsupervised Learning considers the problem of discovering regularities, features or structure in unlabelled data. The unlabelled data distinguishes unsupervised learning from supervised learning and reinforcement learning. It is often easier to obtain large quantities of unlabelled data from databases and sources on the web, for example images of unlabelled objects. For this reason, the idea of using unsupervised learning in combination with supervised learning has attracted interest for sometimes. This machine learning algorithm do not have any target or outcome variable to predict or estimate. It is used for clustering population in different groups, which is widely used for segmenting customers in different groups for specific intervention.

In reinforcement learning, there's a mapping from input to output which is not present in unsupervised learning. In unsupervised learning, the main task is to find the underlying patterns rather than the mapping. For example, if the task is to suggest a news article to a user, an unsupervised learning algorithm will look at similar articles which the person has previously read and suggest anyone from them. Whereas a reinforcement learning algorithm will get constant feedback from the user by suggesting few new articles and then build a "knowledge graph" of which articles will the person like [12, 13].

There is also a fourth type of machine learning methodology called semi-supervised learning, which is essentially a combination of supervised and unsupervised learning. It differs from reinforcement learning as same to supervised and semi-supervised learning has direct mapping whereas reinforcement does not [14].

Python is now appropriate for building enterprise software

From the early 2000s through the today the languages and ecosystems for many dramatically typed languages have greatly improved and often surpassed some aspects of other ecosystems. Python, Ruby and other previously derided languages now have vast, well-maintained open source ecosystems backed by both independent developers and large companies including Microsoft, IBM, Google, Facebook, Dropbox, Twilio and many others. Python's open source libraries, especially for web development and data analysis, are some of the best maintained and fully featured pieces of code for any language.

Meanwhile, some of the traditional enterprise software development languages such as Java have languished due to underinvestment by their major corporate backers [15]. When Oracle purchased Sun Microsystems in 2009 there was a long lag time before Java was enhanced with new language features in Java 7. Oracle also bundles unwanted adware with the Java installation, whereas the Python community would never put up with such a situation because the language is open source and does not have a single corporate controller [16].

Other ecosystems, such as the .NET platform by Microsoft have fared much better. Microsoft continued to invest in moving the .NET platform along throughout the early party of the new millennium[17]. However, Microsoft's enterprise products often have expensive licensing fees for their application servers and associated software. In addition, Microsoft is also a major backer of open source, especially Python, and their Python tools for Visual Studio provide a top-notch development environment [8, 18].

The end of the result is that enterprise software development has changed dramatically over the past couple of decades. CIOs and technical executives can no longer ignore the progress of Python and the great open source community in the enterprise software development landscape if they want to continue delivering business value to their business side customers [19].

Enterprise Python

Enterprise software is built for the requirements of an organization rather than the needs of an individual. Software written for enterprises often needs to integrate with legacy systems, such as existing databases and non-web applications. There are often requirements to integrate with authentication systems such as the Lightweight Directory Access Protocol (LDAP) and Active Directory (AD) [20].

Organizations develop enterprise software with numerous custom requirements to fit the specific needs of their operating model. Therefore, the software development process often becomes far more complicated due to disparate factions within an organization vying for the software to handle their needs at the expense of their factions.

The complexity due to the many stakeholders involved in the building of enterprise software leads to large budgets and extreme scrutiny by non-technical members of an organization. Typically, those non-technical people place irrational emphasis on the choice of programming language and frameworks when otherwise they should not make technical design decisions [21, 22].

Misconceptions about Python in enterprise environments

One of the misconceptions around Python and other dynamically-typed languages is that they cannot be reliably used to build enterprise-grade software. However, almost all commercial and government enterprises already use Python in some capacity, either as glue code between disparate applications or to build the application themselves.

Traditionally large organizations building enterprise software have used statically typed languages such as C++, .NET and Java. Throughout the 1980s and 1990s large companies such as Microsoft, Sun Microsystems and Oracle marketed these languages as “enterprise grade”. The inherent snub to other languages was that they were not appropriate for CIOs’ difficult technical environments. Languages other than Java, C++ and .NET seen as risky and therefore not worthy of investment [23].

In addition, “scripting languages” such as Python, Perl and Ruby were not yet robust enough in the 1990s because their core standard libraries were still being developed[24]. Frameworks such as Django, Flask and Rails (for Ruby) did not yet exist. The web was just a beginning and most enterprise applications were desktop apps built for Windows. Python simply wasn’t made for such environments [25, 26].

CONCLUSION

There are two cue suggestions that allow RL algorithms to achieve the goal. The first suggestion is to use samples to compactly represent the dynamics of the control problem and this is important for two reasons. First reason is that it allows one to deal with learning scenarios when the dynamics is unknown and second, even if the dynamics is available, exact reasoning that uses it might be intractable on its own. The second cue suggestion behind RL algorithms is to use powerful function approximation methods to compactly represent value functions. The significance of this is that it allows dealing with large, high-dimensional state and action-spaces. These two proposals become fit concurrently and tests might be focused around a little subset of the spaces they have a place with, which sharp capacity estimate strategies may misuse. It is the understanding of the interplay between dynamic programming, samples and function approximation that is at the centre of designing, analysing and applying RL algorithms.

Now we can say that Python is the excellent option for the application of RL algorithms as dynamic programming. This programming language have unique ecosystem, cultures and philosophies built around them. Python’s culture values open source software, community involvement with local, national and international events and teaching to new programmers. Developers found that Microsoft’s .NET ecosystem lacking when it came to satisfy their needs and Python filled the gap with its wide array of open source code libraries, package management and ability to work well on platforms other than Windows. Python, Machine Learning and Language Wars compares Python with R, MATLAB and Julia for data science work. While Python is great for deployment automation and web development, many non-developers are first introduced to the language and ecosystem while getting data analysis work done.

REFERENCE

1. Melnikov, A. A., Nautrup, H. P., Krenn, M., Dunjko, V., Tiersch, M., Zeilinger, A., & Briegel, H. J. (2018). Active learning machine learns to create new quantum experiments. *Proceedings of the National Academy of Sciences*, 201714936.
2. Lake, B. M., Ullman, T. D., Tenenbaum, J. B., & Gershman, S. J. (2017). Building machines that learn and think like people. *Behavioral and Brain Sciences*, *40*.
3. Yang, G. H., Sloan, M., & Wang, J. (2016). Dynamic information retrieval modeling. *Synthesis Lectures on Information Concepts, Retrieval, and Services*, *8*(3), 1-144.
4. Aly, A., Griffiths, S., & Stramandinoli, F. (2017). Metrics and benchmarks in human-robot interaction: Recent advances in cognitive robotics. *Cognitive Systems Research*, *43*, 313-323.
5. Portugal, I., Alencar, P., & Cowan, D. (2017). The use of machine learning algorithms in recommender systems: a systematic review. *Expert Systems with Applications*.
6. Xu, X., Zuo, L., & Huang, Z. (2014). Reinforcement learning algorithms with function approximation: Recent advances and applications. *Information Sciences*, *261*, 1-31.
7. Gudivada, V. N., Irfan, M. T., Fathi, E., & Rao, D. L. (2016). Cognitive Analytics: Going Beyond Big Data Analytics and Machine Learning. In *Handbook of Statistics* (Vol. 35, pp. 169-205). Elsevier.
8. Andress, J., & Linn, R. (2016). *Coding for penetration testers: building better tools*. Syngress.
9. Ong, S. P., Richards, W. D., Jain, A., Hautier, G., Kocher, M., Cholia, S., ... & Ceder, G. (2013). Python Materials Genomics (pymatgen): A robust, open-source python library for materials analysis. *Computational Materials Science*, *68*, 314-319.
10. Barto, A. G., & Sutton, R. S. (1997). Reinforcement learning in artificial intelligence. In *Advances in Psychology* (Vol. 121, pp. 358-386). North-Holland.
11. Castelli, M., Vanneschi, L., & Largo, Á. R. (2016). Supervised Learning: Classification.
12. Mendonça, M. R., Bernardino, H. S., & Neto, R. F. (2018). Reinforcement learning with optimized reward function for stealth applications. *Entertainment Computing*, *25*, 37-47.
13. Guyon, I., Dror, G., Lemaire, V., Taylor, G., & Aha, D. W. (2011, July). Unsupervised and transfer learning challenge. In *Neural Networks (IJCNN), The 2011 International Joint Conference on* (pp. 793-800). IEEE.
14. Rhee, P. K., Erdenee, E., Kyun, S. D., Ahmed, M. U., & Jin, S. (2017). Active and semi-

supervised learning for object detection with imperfect data. *Cognitive Systems Research*, *45*, 109-123.

15. Bettini, L., & Damiani, F. (2017). Xtraitj: Traits for the Java platform. *Journal of Systems and Software*, *131*, 419-441.
16. Zhou, Z., Towey, D., Poon, P. L., & Tse, T. H. (2017). Introduction to the Special Issue on Test Oracles.
17. Hardy, C., & Stobart, S. (2003). *Introduction to Web Matrix: ASP. NET Development for Beginners*. Butterworth-Heinemann.
18. Anderson, R. (2002). *Professional ASP. Net 1.0*. Wrox.
19. Nieuwenhuis, L. J., Ehrenhard, M. L., & Prause, L. (2017). The shift to Cloud Computing: The impact of disruptive technology on the enterprise software business ecosystem. *Technological forecasting and social change*.
20. Yeh, Y. S., Lai, W. S., & Cheng, C. J. (2002). Applying lightweight directory access protocol service on session certification authority. *Computer Networks*, *38*(5), 675-692.
21. Fowler, M. (2002). *Patterns of enterprise application architecture*. Addison-Wesley Longman Publishing Co., Inc..
22. Lawton, G. (2005). LAMP lights enterprise development efforts. *Computer*, *38*(9), 18-20.
23. Atkinson, C., Gerbig, R., & Fritzsche, M. (2015). A multi-level approach to modeling language extension in the enterprise systems domain. *Information Systems*, *54*, 289-307.
24. Biggar, P., de Vries, E., & Gregg, D. (2012). A practical solution for achieving language compatibility in scripting language compilers. *Science of Computer Programming*, *77*(9), 971-989.
25. Nitze, A. (2015, January). Evaluation of JavaScript quality issues and solutions for enterprise application development. In *International Conference on Software Quality* (pp. 108-119). Springer, Cham.
26. McMorran, A. W., Lincoln, R. W., Taylor, G. A., & Stewart, E. M. (2011, July). Addressing misconceptions about the common information model (CIM). In *Power and Energy Society General Meeting, 2011 IEEE* (pp. 1-4). IEEE.