*i*Journals: International Journal of Software & Hardware Research in Engineering (IJSHRE)
ISSN-2347-4890
Volume 9 Issue 9 September 2021

Investigating the Humanoid Stand Up Problem using Deep Q-Learning

Author: Wonjun Lee

E-mail: wonjunlee33@gmail.com

10.26821/IJSHRE.9.9.2021.9921

ABSTRACT

Humanoid robots have very much to give; they can be deployed in life-risking situations, as help in the service industry, or even as a companion paired with an AI. The purpose of this paper was to solve the Humanoid Stand Up problem, where a simulated humanoid provided by physics engine MuJoCo was trained by an Artificial Neural Network to stand up after having fallen down. The Reinforcement Learning algorithm Deep Q-Learning was used, since Reinforcement Learning allowed for the lack of a pre-existing dataset. OpenAI Gym was used to compare different Deep Q-Learning training models to find out which was the most efficient method of training the agent in an environment where the action space was immensely large and there were too many state-action pairs. Training models that specifically prevented overfitting underperformed due to the large action space. Instead, models that efficiently explored the state space, such as the Duelling DQN, showed promising results.

1. INTRODUCTION

1.1 Motivation

iJournals

Boston Dynamics recently posted a demonstration of their Humanoid Robot successfully running on various obstacles and even doing somersaults. This quickly went viral on YouTube. These videos showed the rapid progress made in these humanoids: just walking forwards was considered a ground-breaking achievement less than ten years ago. I thus wondered what had allowed such a leap in development to occur; further research made clear that advancements made in Artificial Intelligence (AI), especially in the Deep Learning category, was to be credited. AI is already supporting many aspects of our daily lives. For example, it drives Tesla's autonomous driving features and Netflix's individual content curation amongst many others. Having always loved robots and coding, I used this opportunity to create a Reinforcement Learning program that allowed a humanoid to teach itself to stand up after it had fallen over.

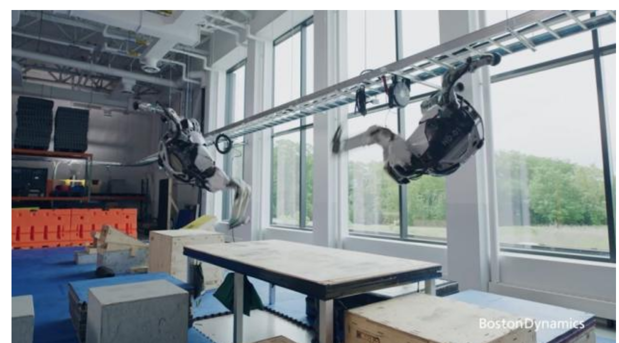


Fig 1: Tesla's autonomous vehicle and Boston Dynamics' Humanoid

1.2 Design of the Paper

This paper broadly consists of three parts. First, I will introduce the concepts of Artificial Neural Networks (ANN) and Reinforcement Learning, which are prerequisites to understanding the experiment. I will also be exploring the concepts of Q-Learning and its derivatives with neural networks. Then, I will present my experimental methodology. Finally, I will be presenting and analysing the results, showing what kind of implications they may make.

2. BACKGROUND KNOWLEDGE

2.1 Artificial Neural Networks

Artificial Neural Networks (ANN) are computing systems designed to simulate the method in which human brains analyse information. Artificial Intelligence (AI) itself relies upon the doctrines of ANN: through these, computers are able to solve problems that would either be impossible or time-consuming to humans. ANN also have self-learning abilities that allow for corrections to be made to their own output as more data becomes available.

2.1.1 Analogy of ANN via Organic Neural Networks

An intelligent animal's brain was the model used to conceive the idea of an ANN. As such, the structure of an ANN is easier to visualise when compared to that of a human brain. A Neural Circuit is defined to be an arrangement of neurons and their interconnections, which perform particular limited functions. Many of these neural circuits come together to form a large scale (brain) network, sending and receiving large amounts of information.

Similarly, an ANN is based on a collection of connected units or nodes called Processing Units (these act as 'artificial neurons'). They are made up of input and output units, much like their organic counterpart. The input units acquire information from other Processing Units based on a weighting system (see below). The output signal given is a real number computed by some non-linear function of the sum of its inputs.

iJournals

Processing Units are usually formed in layers. These different layers often perform different transformations on their inputs. The initial input signal travels from the first layer all the way through to the last layer, where a transformed value is output. The layers in between the first and last layers are known as the hidden layers. Multiple connections are also possible between layers.

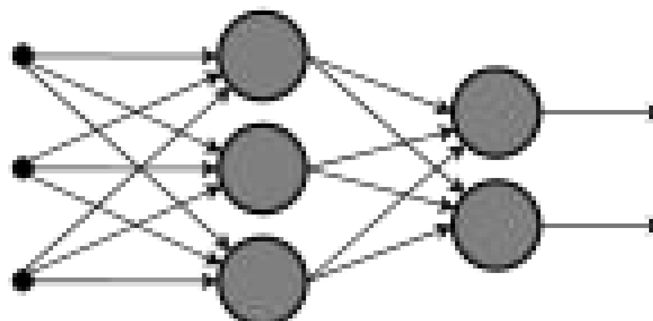


Fig 2: Structure of a basic ANN. The connections are noted as lines, and the Processing Units are denoted as grey circles.

© 2021, *i*Journals All Rights Reserved

2.1.2 Training of ANN

At the conception of an ANN, it goes through a training phase. This is to allow it to recognise patterns in specific data. During this phase, the ANN compares its own output to the actually correct answer (the desired output). This is then corrected using a technique called Backpropagation; a technique entailing the network working backwards from output to input to adjust the weighting of the Processing Units so that the network's output produces the lowest possible error. There are also 3 major learning paradigms: Supervised Learning, Unsupervised Learning, and Reinforcement Learning.

2.2 Reinforcement Learning

2.2.1 Definition

Reinforcement Learning (in programming) is defined to be an area of machine learning in which an agent acts in a given environment in order to maximise the notion of cumulative reward. It differs from the other learning types by eliminating the need for a pre-existing dataset (as in Supervised Learning) and not employing the method of clustering similar data (as in Unsupervised Learning). A more detailed scenario is given with the Bellman Equation.

2.2.2 Fundamentals of Reinforcement Learning

Suppose there exists a being who has just learnt how to park a car. Naturally, they would park their car outside of their house every day, meaning that over time, they would find parking in front of their home straightforward. This is called model-based learning. Suppose now that they must park in an unfamiliar location (i.e. their neighbourhood supermarket). This may cause some degree of trouble, but they may reach into their experience in parking at their home to achieve this. This is called model-free learning.

iJournals

Model-based learning requires the use of prior experiences of the environment the agent is in, whilst Model-free learning makes use of the interactions made previously in other environments. The goal of both methods is similar: to find a 'winning' strategy that maximises the reward gained.

2.2.3 Fundamentals of Reinforcement Learning

The Bellman Equation is used to mathematically model the steps taken in Reinforcement Learning. Following the definition given above, an agent in a Reinforcement Learning algorithm performs a series of steps in a systematic manner so that it can learn the ideal solution by receiving guidance from reward values. Below is a simplified Bellman Equation:

$$\text{Current value} = \text{Immediate Reward} + \text{Future Value}$$

© 2021, *i*Journals All Rights Reserved

2.3 -Learning

2.3.1 Introduction

Q-Learning is a model-free Reinforcement Learning algorithm used to learn the value of an action in a particular state. It finds an optimal strategy to maximise the expected value of the total reward over all states. Like the Bellman Equation, the Q function (derived from Q-learning definition) can be expressed as a mathematical function.

The Q in Q-Learning stands for quality, reflecting its use in calculating how beneficial a given action is to the agent in terms of maximising the reward.

2.3.2 Q-Tables

A Q-Table is created in executing the Q-Learning algorithm. Q-Tables are structured as a matrix in the form of [state, action]; values of zero are initially used. After an episode, the Q-Values (used to denote the values in the Q-Table) are updated and stored in place of the zeroes. This table then becomes the reference table that the agent employs to determine the best action in that particular state.

After the termination of the agent (i.e. the end of an episode), the agent replays the experiences in its saved memory. A certain batch will be chosen from there and training will be performed on it, filling up the Q-Table. This process is called Experience Replay, which will come into handy later.

2.3.3 The Q-Learning Algorithm Equation

As mentioned before, after every episode, the Q-Values are updated. They are updated according to the following equation:

$$Q(state,action) \leftarrow (1-\alpha)Q(state,action)+\alpha(reward+\gamma \max Q(next\ state,all\ actions))$$

where alpha (α) is the learning rate, and gamma (γ) is the discount factor.

The Learning Rate (α) is the extent to which the Q-Values are being updated in every iteration. It can be defined as how much we accept the new Q-Value as opposed to the old Q-Value. The Learning Rate is a real number between 0 and 1, such that $0 < \alpha \leq 1$. As α grows larger, we can see that $(1-\alpha)Q(state,action)$ tends to zero, putting less significance on the old Q-Value.

The Discount Factor (γ) determines how much importance is given to future rewards. Similar to α, the Discount Factor is a real number between 0 and 1 ($0 < \gamma \leq 1$). A high value puts importance onto the long-term reward, whilst a low value makes the agent greedy, that is, prioritise immediate reward.

We also introduce another hyperparameter called epsilon (ϵ). This parameter ensures that the agent does not always take the same route, possibly overfitting. ϵ allows for the agent to sometimes favour exploring the action space further.

2.3.4 The Hyperparameters

The hyperparameters should ideally decrease over time, since:

1. The agent acquires a larger knowledge base (Learning Rate should decrease).

iJournals

2. The agent has no need for a long-term reward, as it will not be around long enough to reach it (Discount Factor decreases).

3. There is no need for further exploration once the agent develops its strategy (Epsilon decreases).

$$R_t + \gamma Q^c(s_{t+1}.\operatorname{argmax} Q(s_{t+1}.a'))$$

Most of the time, the three parameters are tuned according to trial and error. There is, however, a method of fine-tuning the parameters using Genetic Algorithms.

2.4 Deep Q-Learning

We have already touched on Q-Learning several times, but that is only the general encompassing term for a group of different learning models. The ones listed below cannot be ranked in any way: different situations warrant different models. The models can be thought of as the 'brain' or the 'thought process' of the agent. They also handle different aspects of the agent, such as its mind, its experience, and its exploration.

2.4.1 Deep Q-Learning

It can be said that the 'brain' of the agent is the difference between regular Q-Learning and Deep Q-Learning (DQN). If the brain in regular Q-Learning is the Q-Table which the agent refers to when making decisions, the brain in a DQN is a deep neural network. Otherwise, everything remains virtually identical. The input of the neural network is the state or observation, and the number of output neurons would coincide with the number of actions the agent can take.

The targets of the neural network would be the Q-Values of each of the actions and the input would be the state that the agent is in.

2.4.2 Double DQN

As the name implies, Double DQN is an enhanced version of the DQN mentioned above. Double DQN was created to fix a certain problem that DQNs are prone to – overfitting. Double DQN uses two of the same neural network models. One of the networks is one that learns from the experience replay; one that is identical to the neural network in standard DQNs. The second one is actually a copy of the previous episode of the first neural network. The Q-Value is then computed with this second model.

$$Q(s.a) = V(s) + A(s.a) - \frac{1}{n}\sum_{a'} A(s.a')$$

In a DQN, since the Q-Value is computed by adding the reward to the next state maximum Q-Value, the values from consistently high Q-Value states will continue to grow larger. This causes overfitting and proves detrimental to the network's learning efficiency. We cannot have such high differences between the output values. Thus, the second model is used to calculate the Q-Values: since the second model is a copy of the first model from the previous episode, the differences are obviously lower. This can be expressed mathematically as such:

The index of the highest Q-Value from the first model is found and then is correlated to the second model to determine the next action of the agent.

2.4.3 Duelling DQN

Duelling DQN varies from DQN and its derivatives in the structure of the model. It is mathematically expressed as such:

$$Q(s.a) = V(s) + A(s.a)$$

where V(s) is the Value of some arbitrary state s and A is the advantage of doing an arbitrary action a while in state s. In this model, the value of a state is independent of the action. That is, we calculate a score not only for what action to take in a specific state, but for also the state itself.

iJournals

$$Importance = \left(\frac{1}{p_i} * \frac{1}{memory\ size}\right)^b$$

This is especially useful in a scenario where the Q-Values of all the actions in a specific state are identical. It can thus be concluded that there is no one singular action that is superior. We thus divide the Q-Value into two: the Value of the state and the Advantage of the action. We know that if every action has the same result, the Advantage of each action will have the same value. What if the mean of all the Advantages is then subtracted from each advantage? The following mathematical equation is employed to calculate the Q-Value:

This gives a value close to zero, and so the states that are independent of action (i.e. unfavourable states) are not trained on.

2.4.4 Noisy DQN

This is another method of ensuring that overfitting does not happen. Much like the ‘epsilon’ value mentioned above, Noisy DQN involves introducing randomness to allow the exploration of the action space. However, unlike epsilon, Noisy DQN adds noise to the output of the neural network. Therefore, whenever there is noise, the agent explores the environment for a more efficient method.

This is done by defining weightings for the neural network. The mathematical equation for the weighting is such defined as:

W = Mu + Sigma * epsilon

Where Mu is a variable with random initialisation, Sigma is a variable with constant initialisation, and epsilon is the noise with a random value between zero and one. Over time, when exploration is not prioritised in favour of using prior experiences, Sigma drops close to zero, essentially eliminating epsilon, and any noise.

2.4.5 DQN with Prioritised Experience Replay

During the Experience Replay step mentioned above, we know that a certain batch will be chosen to apply the neural network training. However, in DQN with Prioritised Experience Replay, we take into account the credibility of the memory chosen. That is, what if the memory chosen is not so important to learn, and will be rather detrimental to the training process? DQN with Prioritised ER allows for the choosing of important experience memories to put into the batch.

iJournals

The importance of memories is determined by analysing the difference in Q-Values from one state to another. If Q-Values change drastically between states, that must mean that there is significance in this step. This is called Temporal Difference error (or TD error). The formula for TD error is as follows:

This then leads to the probability of an experience memory being chosen to be:

$$TD = |Q(s,a) - Q(s+1,a)|$$

where epsilon prevents division by zero and alpha (between zero and one) controls to what extent the memories are filtered.

But if one experience has high probability and is chosen every time, overfitting would disadvantage the training process. This is thus countered by using these operations:

$$p_i = \frac{(TD_i + \varepsilon)^a}{\sum_k^{memory\ size}(TD_k + \varepsilon)^a}$$

where b starts from zero and slowly increases to 1. The importance is computed from the distribution the experience originated from. This allows high probability memories to not be chosen all the time.

3. EXPERIMENT

3.1 OpenAI Gym

The Gym library is a toolkit provided by OpenAI that facilitates Reinforcement Learning and allows for comparing different reinforcement learning algorithms. Since Reinforcement Learning, by nature, expresses its given environment as a combination of numbers, the computer can progress through its training knowing nothing about what it is doing, provided that every action is tied to a suitable reward. OpenAI Gym makes use of this adaptability and thus provides a variety of different environments.

3.2 HumanoidStandup-v2 environment

The 'HumanoidStandup-v2' environment, used in this experiment, is a humanoid simulator that utilises the MuJoCo physics engine. The goal is to control the angle and speed of all of the humanoid's individual body parts to make it stand upright without falling back down. More specifically, this environment expresses the speed and location vector of the arms, legs, and torso in 17 real numbers. Each action reflects how a certain body part is controlled to move. But as mentioned before, the computer does not have to know what these numbers signify; our ANN considers the given states simply as a combination of numbers. It only concerns itself on the potential reward given to a certain action.

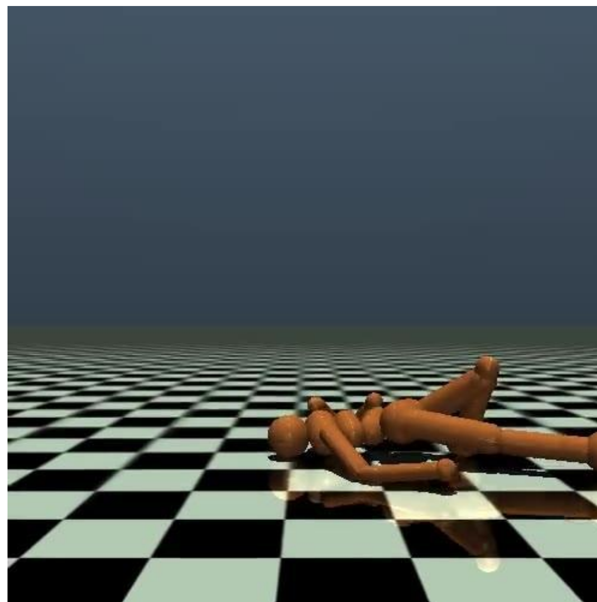
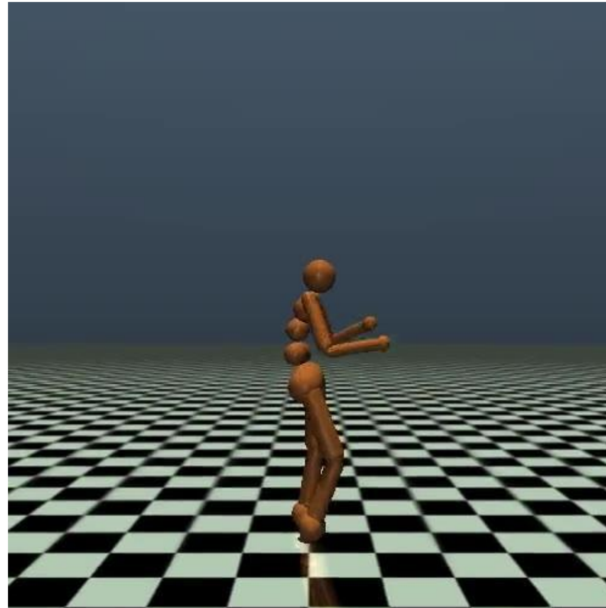


Fig 3: The HumanoidStandup-v2 environment used in this experiment

3.3 Experimental Procedure

In order to compare the performance of different Deep Reinforcement Learning Algorithms, five different versions, all using different Reinforcement Learning models, were made. The Validation Score, which was heavily based on the time spent training, was used as a standardised form of comparison. All learning models were created using Python, and the process of training was conducted using a CentOS Linux machine with an Intel Xeon E5-2680 V3 (@ 2.50GHz) Dual-core CPU, paired with 256GB of memory.

© 2021, *i*Journals All Rights Reserved



4. RESULTS AND ANALYSIS

The Humanoid Stand Up problem has too many states to use regular Q-Learning (there are twelve state variables that have floating point values, so it is impossible to record each and every state), so Deep Q-Learning was used. Typically, it is difficult to create a good algorithm for a certain machine learning problem due to overfitting. However, we postulated that overfitting would hardly be an issue for our problem; the sheer number of states suggested that converging to a valid answer would take a long time in itself.

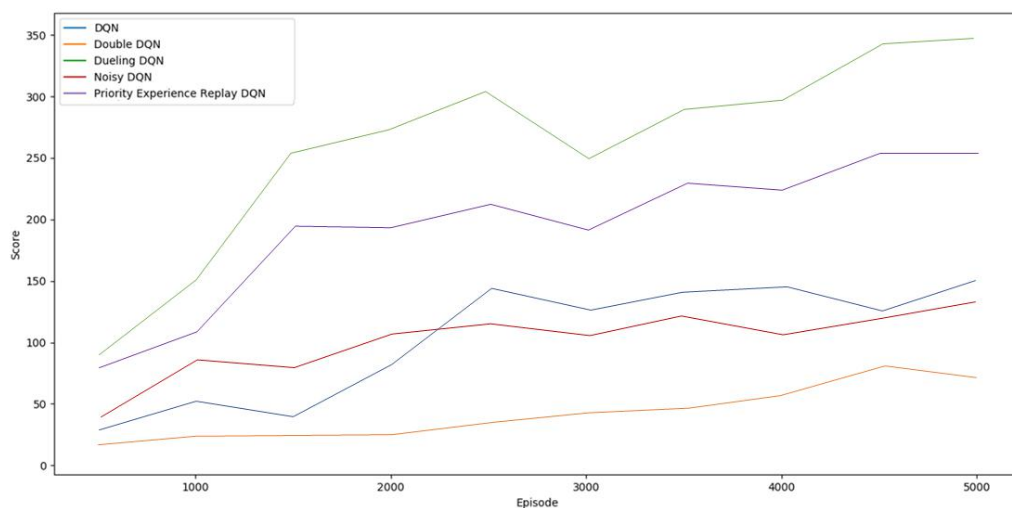


Fig 4: The Score vs Time graph as output by the RL models

Running the experiment with each RL model produces the Validation Score versus Training episode graph above. Looking at the final results it is easy to see that as predicted, the models that employed techniques to reduce overfitting (Double DQN, Noisy DQN) underperformed. The randomness added to combat overfitting

had interfered with the model's ability to converge to an answer. On the other hand, due to the immensely large action space, algorithms that solely focused on converging towards an answer (Duelling DQN, DQN with PER) showed promising performances. In particular, Duelling DQN showed great performance compared to the others. This can be accredited to Duelling DQN's method of giving separate scores to both the current state and the next action. Since our problem focuses on how well the humanoid is able to balance itself to stand up, it can be deduced that the fact that Duelling DQN gave a separate score to posture(state) (i.e. how advantageous it is to be in this state) allowed it to perform much higher than its competitors.

5. CONCLUSION

Through this experiment, we learnt that the Duelling DQN training model was most suited to tackling a problem with a large action space, such as the

Humanoid Stand Up problem – it achieved a Validation Score of 350 in a range of 60 to 350 in 5000 episodes.

Using regular Q-Learning instead of Deep Q-Learning would have resulted in an infinitely longer time due to our problems with excessive state-action pairs. Even using Deep Q-Learning along with a powerful machine took a substantial amount of time.

Not using Reinforcement Learning at all and choosing to randomly (brute-force) come to a convergence would without a doubt have also taken an absurd amount of time.

It would be interesting to consider the practical aspect of these results. Using this model on a real humanoid robot may be the next step – though there may be more effective training models, considering that the training models used in this experiment can be combined to form more advanced models.

6. REFERENCES

[1] Expert System Team. "What is Machine Learning? A Definition." Last modified May 6, 2020. https://www.expert.ai/blog/machine-learning-definition/#:~:text=Machine%20learning%20is%20an%20application,use%20it%20learn%20for%20themselves.

[2] MonkeyLearn. "Everything There Is to Know about Sentiment Analysis".https://monkeylearn.com/sentiment-analysis/#:~:text=Sentiment%20analysis%20(or%20opinion%20mining,feedback%2C%20and%20understand%20customer%20needs.

[3] Stanford CS231n Notes. "A Complete Guide to K-Nearest-Neighbours with Applications in Python and R".

https://cs231n.github.io/classification/#nn

[4] Hawkins, Douglas M. "The Problem of Overfitting." National Library of Medicine. Last modified February 2004.

https://pubmed.ncbi.nlm.nih.gov/14741005/

[5] Stanford. "Large Movie Review Dataset".

https://ai.stanford.edu/~amaas/data/sentiment/

[6] Frankenfield, Jake. "Artificial Neural Networks".

https://www.investopedia.com/terms/a/artificial-neural-networks-ann.asp

[7] American Psychological Association.

https://dictionary.apa.org/neural-circuit

[8] Burns, Ed. "What is a Neural Network?".

iJournals

https://searchenterpriseai.techtarget.com/definition/neural-network

[9] Violante, Andre. "Simple RL: Q-Learning".

https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56

[10] Kansal, Satwik and Martin, Brendan. "Reinforcement Q-Learning".

https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/

[11] Moghadam, Parsa H. "Deep Reinforcement Learning: DQN, Double DQN, Duelling DQN, Noisy DQN and DQN with Prioritised Experience Replay".

https://medium.com/@parsa_h_m/deep-reinforcement-learning-dqn-double-dqn-dueling-dqn-noisy-dqn-and-dqn-with-prioritized-551f621a9823

[12] Yoon, Chris "Duelling Deep Q-Networks".

https://towardsdatascience.com/dueling-deep-q-networks-81ffab672751

iJournals