

University of Puerto Rico - Mayagüez Campus
Programming Languages ICOM 4036
Class Project - Phase 3: Final Report
Prof. Wilson Rivera - Spring Semester 2019

Deep Q-learning Box

Developers

Guillermo Betancourt
Fabiola Badillo
Kathiana Diaz

I. Introduction and motivation:

The broad topic of AI and machine learning is slowly entering mainstream conversation due to its recent strides in algorithm development. For the broad audience machine learning is riddled with mysterious concepts and an inevitable sci-fi background; however, companies such as DeepMind have made strides with Deep Learning techniques that proved successful when their AlphaGo¹ project beat the top Korean Go player back in 2016. The topic of AI is mostly very high level since it is a bleeding edge technology currently involved in many research experiments that are trying develop and improve current techniques that allow a computer or a system to make decisions on its own. Our proposed programing language **DQB** serves to lower the level of abstraction to those interested in learning one of the basic methods of achieving Deep Reinforcement Learning (DRL), one of the many ways we can achieve the behaviour of a “self-thinking” computer.

DQB will encapsulate the main components and concepts of deep reinforcement learning to provide beginners with a subtle introduction to Q-Learning implementations². In order to continue a steady pace of improvements in AI and machine learning in general we must normalize and facilitate the introduction of complex mathematical notation that scares away brilliance. Hence our project is a virtual box with the tools required to create and train simple reinforcement learning agents in multiple environments. The focus would lie on encapsulating convoluted code but maintaining a clear exoskeleton of what is needed to develop a deep reinforcement learning agent.

The scope of DQB lies within 2D-environments that recreate old atari games via the gym python library by OpenAI. Its purpose is to serve as a playground for developing models with the goal of serving as people’s first encounter.

¹ AlphaGo is the first computer program to defeat a professional human Go player, the first program to defeat a Go world champion, and arguably the strongest Go player in history.

² Q-learning is a RL technique that creates an agent and lets it use the environment's rewards to learn, over time, the best action to take in a given state.

II. Language Tutorial

To get started, clone the repository to your computer. Make sure you have Python 3 and the PLY parsing tool installed. Afterwards, you need to install the external libraries required by using the command:

```
pip install -r requirements.txt
```

Now you have everything to write in DQB!

*Ubuntu or Mac only.

Instructions:

1. First, to get started, use the command:

```
python3 DQB.py
```

2. Write the DQB source code on a .txt file called DQB_script.
3. Run the DQB_blackbox which initializes, compiles, and trains the agent.
4. Observe the agent's training process! You have the option to display the environment as it trains and the model's status episode by episode.

To learn how to write the source code go to the example programs and the reference manual below.

Example program 1: Pong Environment

DQB source code on DQB_script.txt file

```
main({  
  ENVIRONMENT:Pong  
  AGENT:AtariPro{  
  
    MODEL_PARAMETERS:{  
  
      Learning_Rate = 0.001  
      Discount_Factor = 0.99  
  
    }  
  
    NETWORK:{  
      add(ConvolutionalLayers)  
      add(PredictiveLayers)  
      showModelSummary()  
  
    }  
  
    TRAINING:{  
      find_probabilities()  
      predict_moves()  
      fit()  
      displayGame()  
  
    }  
  }  
  Execute()  
}
```

Display the environment as it trains and the model's status episode by episode:



```
Episode: 1 - Score: -9.000000.  
Episode: 2 - Score: 5.000000.  
Episode: 3 - Score: -6.000000.  
Episode: 4 - Score: -4.000000.  
Episode: 5 - Score: 4.000000.  
Episode: 6 - Score: -2.000000.  
Episode: 7 - Score: -4.000000.  
Episode: 8 - Score: -7.000000.  
Episode: 9 - Score: 1.000000.  
Episode: 10 - Score: -1.000000.  
Episode: 11 - Score: -2.000000.  
Episode: 12 - Score: 6.000000.  
Episode: 13 - Score: -4.000000.  
Episode: 14 - Score: -2.000000.  
Episode: 15 - Score: -3.000000.  
Episode: 16 - Score: -2.000000.  
Episode: 17 - Score: -4.000000.  
Episode: 18 - Score: -10.000000.
```

Example program 2: Pong Environment

DQB source code on DQB_script.txt file

```
main(){  
  
    ENVIRONMENT:BrickBreaker  
    AGENT:BBA{  
  
        MODEL_PARAMETERS:{  
            Learning_Rate = 0.10  
            Epsilon_Start = 1.0  
            Epsilon_End = 0.10  
            Exploration_Steps = 1000000  
            Batch_Size = 16  
            Discount_Factor = 0.90  
            No_Steps = 30  
            Action_Size = 3  
        }  
        NETWORK:{  
            add(ConvolutionalLayers)  
            add(PredictiveLayers)  
        }  
        TRAINING:{  
            predict_moves()  
            calculateQ_Values()  
            modelCurrentStatus()  
            displayGame()  
        }  
    }  
    Execute()  
}
```

Display the environment as it trains and the model's status episode by episode:



| | | | | | | |
|-------------|------------|---------------------|--------------|-------------------|---------------------------------|-------------------|
| episode: 0 | score: 3.0 | memory length: 245 | epsilon: 1.0 | global_step: 245 | average_q: 0.021846184947965096 | average loss: 0.0 |
| episode: 1 | score: 2.0 | memory length: 444 | epsilon: 1.0 | global_step: 444 | average_q: 0.023213150035870735 | average loss: 0.0 |
| episode: 2 | score: 1.0 | memory length: 590 | epsilon: 1.0 | global_step: 590 | average_q: 0.021619951952095717 | average loss: 0.0 |
| episode: 3 | score: 2.0 | memory length: 767 | epsilon: 1.0 | global_step: 767 | average_q: 0.021686944005600478 | average loss: 0.0 |
| episode: 4 | score: 1.0 | memory length: 924 | epsilon: 1.0 | global_step: 924 | average_q: 0.022075470169163815 | average loss: 0.0 |
| episode: 5 | score: 1.0 | memory length: 1087 | epsilon: 1.0 | global_step: 1087 | average_q: 0.021686089470799714 | average loss: 0.0 |
| episode: 6 | score: 1.0 | memory length: 1232 | epsilon: 1.0 | global_step: 1232 | average_q: 0.023459750863498656 | average loss: 0.0 |
| episode: 7 | score: 2.0 | memory length: 1396 | epsilon: 1.0 | global_step: 1396 | average_q: 0.02182683106738983 | average loss: 0.0 |
| episode: 8 | score: 3.0 | memory length: 1635 | epsilon: 1.0 | global_step: 1635 | average_q: 0.024080044780877344 | average loss: 0.0 |
| episode: 9 | score: 5.0 | memory length: 1941 | epsilon: 1.0 | global_step: 1941 | average_q: 0.021638349356020197 | average loss: 0.0 |
| episode: 10 | score: 1.0 | memory length: 2099 | epsilon: 1.0 | global_step: 2099 | average_q: 0.023378281123181688 | average loss: 0.0 |
| episode: 11 | score: 1.0 | memory length: 2252 | epsilon: 1.0 | global_step: 2252 | average_q: 0.021170890311789668 | average loss: 0.0 |
| episode: 12 | score: 2.0 | memory length: 2436 | epsilon: 1.0 | global_step: 2436 | average_q: 0.02250012611646367 | average loss: 0.0 |
| episode: 13 | score: 0.0 | memory length: 2529 | epsilon: 1.0 | global_step: 2529 | average_q: 0.022584119010516393 | average loss: 0.0 |
| episode: 14 | score: 1.0 | memory length: 2669 | epsilon: 1.0 | global_step: 2669 | average_q: 0.024462294724902935 | average loss: 0.0 |
| episode: 15 | score: 4.0 | memory length: 2949 | epsilon: 1.0 | global_step: 2949 | average_q: 0.023460897460712917 | average loss: 0.0 |
| episode: 16 | score: 1.0 | memory length: 3116 | epsilon: 1.0 | global_step: 3116 | average_q: 0.022065567376906285 | average loss: 0.0 |
| episode: 17 | score: 0.0 | memory length: 3217 | epsilon: 1.0 | global_step: 3217 | average_q: 0.0214828350166283 | average loss: 0.0 |

III. Reference Manual

DQB is a simple high-level reinforcement scripting language, developed on PLY capable of building , training and modeling on top of TensorFlow, Keras and Gym. It is focused on introducing fast experimentation with agents training on environments. It features two training environments, Pong and Brick Breaker.

A. Language Structure

They both implement 4 functions each:

- a. `main(){ ... }` - Initializes the environment, the agent and the operations it performs.
- b. `MODEL_PARAMETERS:{...}`- Initializes the trainable model parameters which are:
 - i. Learning rate: parameter that controls how much the weight of the neural network are updated with respect to the loss gradient. Its value is usually between 0.001 and 0.100.
 - ii. Epsilon start: Initial value of the epsilon parameter. Controls the rate between exploration and exploitation steps. This number decreases as the training progresses.
 - iii. Epsilon End: Final value of the epsilon parameter.
 - iv. Exploration Steps: number of training steps the algorithm performs.
 - v. Batch Size: number of samples per gradient update.
 - vi. Discount Factor: value multiplied by future rewards as discovered by the agent. Future rewards worth less than immediate rewards. It is a value between 0 and 1.
 - vii. Action Size: number of distinct steps the agent is capable performing in each exploration step.
- c. `Network: {...}` - initializes the internal neural network to the agent.
 - i. `Add{...}` - adds layers to the network.
 1. ConvolutionalLayers: process the current environment state
 2. PredictiveLayers: performs optimization and backpropagation.
- d. `Training:{...}` - calls the methods that perform additional training and assigns score to each move.
 - i. PredictMoves: performs calculations of training algorithms
 - ii. CalculateQvalues: calculates the Q-values of each exploration step

- iii. `modelCurrentStatus`: displays in the console the value of each trainable parameter as the training goes on.
- iv. `displayGame`: displays an image of the model's current status.

IV. Language Development

Deep Q-Learning Box is meant to be a beginners introduction to machine learning and reinforcement learning algorithms, and it was kept in mind when building the language's design. It was built using python so it would enable us to use the most popular ML libraries used today such as Keras and tensorflow along with PLY for the lexical and syntactical construction of the language. In order to allow for a simplification of the complicated Reinforcement Learning algorithms and serve as a stepping stone for future machine learning engineering aspirants. The language has uses 2 prebuilt atari environments from the gym libraries that use two different algorithms in which we can modify some of the algorithms most important parameters to modify its performance.

We built the language through a python class generator so one is able to view the full python class that implements and runs the algorithm in naive python. This was achieved through the use of a code generator class named ML code generator that generates strings and attaches them to a file in order to compress the algorithm and break them down to their basic actions to facilitate the learning experience while also recording what was used behind the scenes.

The main class `DQB_blackbox` merely serves as a hub that brings together the lexer and parser built to translate our language. In order to maintain the language lightweight the communication between the developer and the program is established via a .txt file named DQB script held within the projects files this is our "IDE" through which one writes the program. `DQB_blackbox` merely imports the parser and translates the file to generate the classes and executes the native python code that activates.

V. Conclusion

Deep Q-learning Box (DQB) serves to lower the level of abstraction to those interested in learning one of the basic methods of achieving Deep Reinforcement Learning (DRL). It normalizes and facilitates the introduction of complex mathematical notation used in Deep Reinforcement Learning. The focus lies on encapsulating convoluted code but maintaining a clear exoskeleton of what is needed to develop a deep reinforcement learning agent. It is focused on introducing fast experimentation with agents training on environments, giving the user basic knowledge of DRL concepts, like Learning rate, Epsilon Start, Epsilon End, Batch Size, Q-values, and more. DQB serves as a playground for aspiring DRL developers.