Dense vs Focused on DQN for Reinforcement Learning

System Design Document

V 0.1

20.04.2020

Ersin ÇEBİ

Prepared for
COMP4902 Graduation Design Project

IŞIK UNIVERSITY
COMPUTER
SCIENCE AND
ENGINEERING

Table of Contents

SYSTEM DESIGN DOCUMENT

1. Introduction

The purpose of this system is to design a reinforcement learning application and compare Dense and Focusing layers efficiency. This document describes reinforcement learning techniques and the techniques that we are going to use in project, libraries and environments that can be used to train an agent.

1.1. Purpose of the System

The aim of the system is to design an accurate and efficient reinforcement learning DQN model that provides easy to implement, more effective than Dense neural networks and time-efficient. The system allows users to use different optimizers, loss functions as they wish. Also, there is an option for a focussed layer for users so users can compare two-layer efficiency.

1.2. Design Goals

The model should be efficient. Optimized for seamless interaction with the existing system is a must. Based on non-functional requirements the next design targets must be achieved as a way to qualify the system as profitable:

User-Friendly: Creating an interface that users can easily understand is an important tool for understanding our project. We will try to ensure that they can see the parameters they can change properly in the interface.
Stable: One of our most important goals is stability. We want to make the model works on every dataset and want to produce a solution.
End-User: All users can train model if they download the code from Github . Application is for Python developers which have Tensorflow-Keras 1.15 or greater version of Android.
Performance: The Model's accuracy and test results should be better than Dense neural network.

Moreover, the other goals of our design Focusing Recurrent Layer Model should accept upgrades and should be implemented on the python code-based platform.

1.3. Definitions, Acronyms, and Abbreviations

Focused: Focus scales weights and changes the variance of propagated signals.
SDD: System Design Document
Python: Python is an interpreted, object-oriented, high-level programming language with dynamic semantics.
Keras: Kerasis a Python-friendly open-source library for numerical computation that makes machine learning faster and easier.
Colab: Google Colab is a free cloud service and now it supports free GPU! You can; improve your Python programming language coding skills. develop deep learning applications using popular libraries such as Keras, TensorFlow, PyTorch, and OpenCV.
DENSE: A dense layer is just a regular layer of neurons in a neural network. Each neuron receives input from all the neurons in the previous layer, thus densely connected.
Q-learning: A model-free reinforcement learning algorithm to learn a policy telling an agent what action to take under what circumstances.
DQN: The DeepMind system used a deep convolutional neural network.

2. Current Software Architecture

There are plenty of ways of reinforcement learning[1]. One is q learning, this approach uses dynamic programming principles. Firstly we set random actions for states in the environment, this is called creating q-table. Q-table is a nested array that contains, actions on a specific state. Then we run our test on that table and get a Q value on every iteration. This q value calculated with an equation in Figure 2-1:

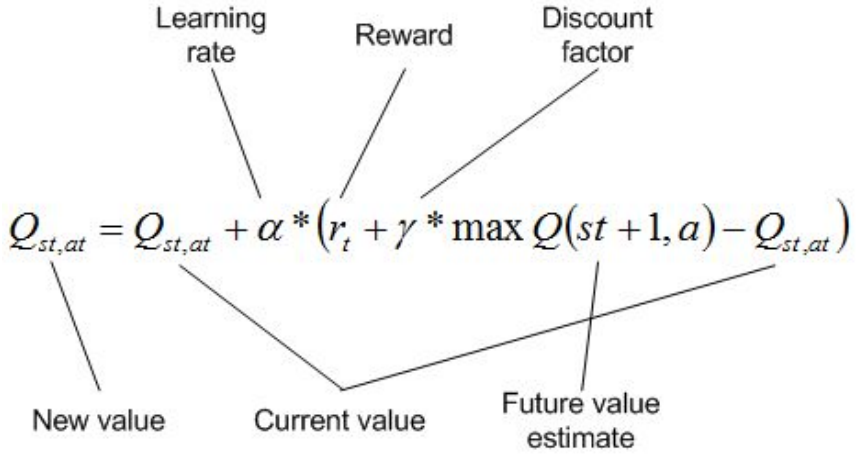

Figure 2-1: Deciding next Q-value on the table, according to the previous action that taken [6]

There is another way of q-learning, it's called Deep Q-learning. In deep learning, each level learns to transform its input data into a slightly more abstract and composite representation. In an agent training application, the raw input may be a matrix of actions; the first representational layer may abstract the action and encode states; the second layer may compose and encode arrangements of states; the third layer may encode an action on a state, and the fourth layer may predict a proper outcome that the action of the current state.

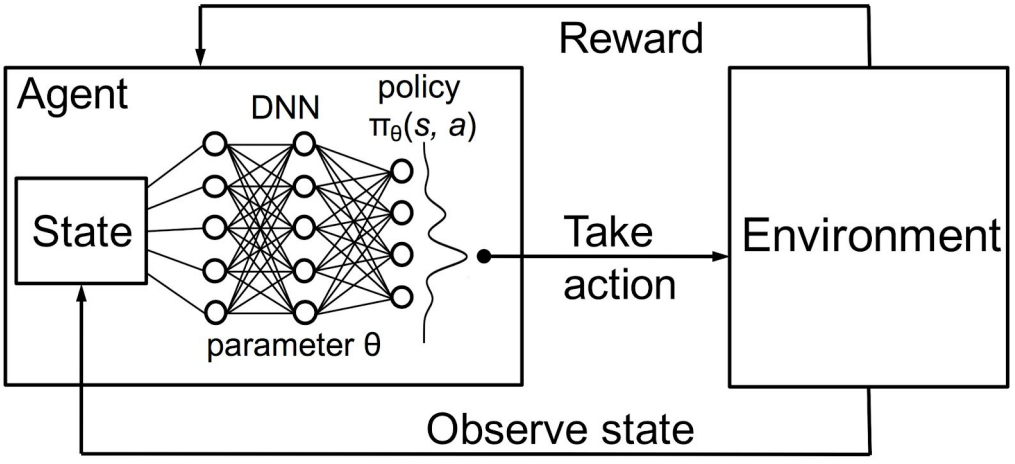

Figure 2-2: Reinforcement Learning using a neural network policy [7]

2.1 Gym Environment

The gym library gives us an agent and environments together. An agent is an artificial intelligence entity that has certain goals, must always remain watchful about things that can come in the way of these goals, and must, at the same time, pursue the things that help in attaining these goals. Because according to the actions of its environment the agent gets either prize or penalty.

According to the problem we give to the agent, the agent takes action. So, what decides this problem and action? The answer is the environment, the environment in the region available for the agent to navigate, and includes all the places the agent can go to, including the obstacles that the agent could crash in to. So the primary task of the agent is to explore the environment, understand how the actions it takes affect its rewards, be cognizant of the obstacles that can cause catastrophic crashes or failures, and then master the art of maximizing the goals and improving its performance over time.

Figure 3.1-1 shows the basic action taking the process of an agent according to its environment.

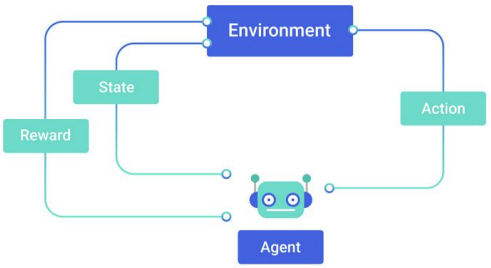

Figure 3.1-1: Agent decision diagram [8]

We said that the agent takes actions according to its environment, by the outcome of the previous action by meaning that the state(observation), reward but how the agent does make these choices? The answer is policy. The policy is the algorithm that determines the action of the agent can use in the environment. we use DQN as a policy, that takes all observation as input and creates output and agent takes those outputs and decides the best action(Figure 2-2).

A policy defines the guidelines for an agent's behavior at a given state. In mathematical terms, a policy is a mapping from a state of the agent to the action to be taken at that state. As a coder of the agent, we expect the best outcome from it, but the policy is not always deterministic, by mean that, it is possible that the policy takes only random actions, there is a chance policy does not observe the environment, it is called a stochastic policy. It is usually denoted as $\pi(a_t|s_t)$ – that is, it is a conditional probability distribution of taking an action a_t in a given state s_t. Policies can be deterministic, wherein the exact value of a_t is known at s_t, or can be stochastic where a_t is sampled from a distribution – typically this is a Gaussian distribution, but it can also be any other probability distribution.

The other way of policies genetic algorithms. In this type of policies we run for example 100 agents in an episode at the same time and choice the best ones, let say there is 20 succeeded, in the next episode you create 4 offspring each of the successful agents for the next generation.

2.2 Q-learning with Neural Networks

Reinforcement learning is an unsupervised learning technique, that is why first we should create data to process, this data is unlabeled and we come up with this data according to the conditions of our environments. An environment limits the agent's actions according to the problem, this problem might be driving a car, a walking person, etc. We could create our own environment by using different libraries but there is OpenAI Gym. The gym gives us plenty of problems with the environment itself, these problems are solved in an environment, each environment has its own action space, observation space, and state variables.

Gym library gives environment type options, which are 3D, 2D, algorithm problems just on the console screen. It is easy to set up, we just need to give how the episode will it run and then set our algorithm that is it. Episode means iteration, on each iteration of the agent there are steps, for each step agent takes an action, end of all this procedures environment returns four basic variables about each step on an episode, these are observation for current state or coordinate of the agent on a grid, an extra parenthesis for coordinate it's for 2D environments, the reward is for if the agent action is correct and successful or fail situation given point, done is a boolean variable if agent completes the task given correctly, info is a dictionary information variable depends on the environment. These variable returns after we call step() function, first we send an action into our environment and this function returns these variables.

In the agent training, we will use the Keras Sequential model with layer Dense and Focused. On the test user can switch between Dense and Focused layers. Focused layer its receptive field in the spatial domain inputs and by using the back-propagation algorithm to learn its focus parameters that control the receptive field locations and apertures, the action space of an environment can be too small. That is why we added a large size Dense layer before switch part of the model, in case of choosing Focused as a layer, so the Focused layer can work more efficient. The figure 3.2-1 represents the DQN working model;

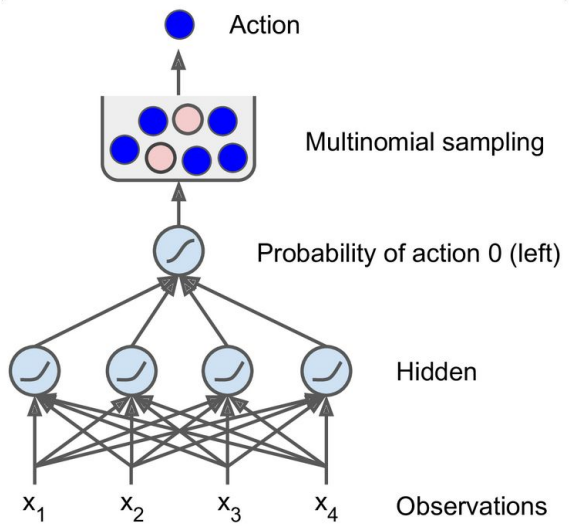

Figure 3.2-1: Neural network policy [1]

As we mentioned, we use DQN, and DQN has some variants, one is fixed q-value targeting. Deep Q Networks take as input the state of the environment and output a Q value for each possible action. The maximum Q value determines, which action the agent will perform. But the same weights apply to both the target and the predicted value. It is like making circles around the same path.

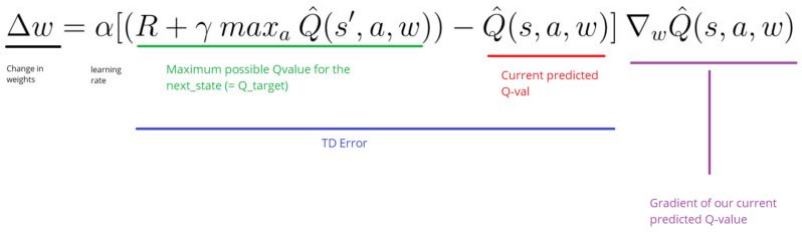

Figure 3.2-2: Calculating weight [9]

We move the output closer to the target, but we also move the target. So, we end up chasing the target and we get a highly oscillated training process. For overcoming this problem we use two models. One is for the agent, the other is for the online working model. Instead of update the weight on each run, for example we calculate them on every 50 runs. This way we updated much less often the target model than the online model, as an outcome our Q-Value targets are more stable.

There is also a method called Double DQN. In Double DQN the solution involves using two separate Q-value estimators, each of which is used to update the other. Using these independent estimators, we can unbiased Q-value estimates of the actions selected using the opposite estimator. We can thus avoid maximization bias by disentangling our updates from biased estimates. On the other hand, the Target network is responsible for the evaluation of that action.

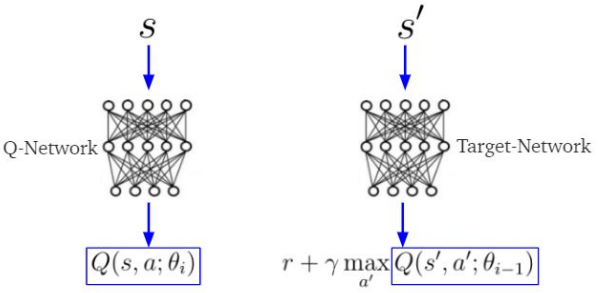

Figure 3.2-3: Double DQN [10]

In this project we will use DQN with dense layers and compare dense loss and accuracy with focusing neuron, this DQN will be applied to OpenAI Gym environments.

3. Proposed Software Architecture

Focusing neurons is better than Dense neurons. Because Focusing neurons can generate unique connection maps for a problem. The new model uses no other tool than the back-propagation algorithm to learn its focus parameters which control the receptive field locations and apertures. So neurons can learn and adapts itself.

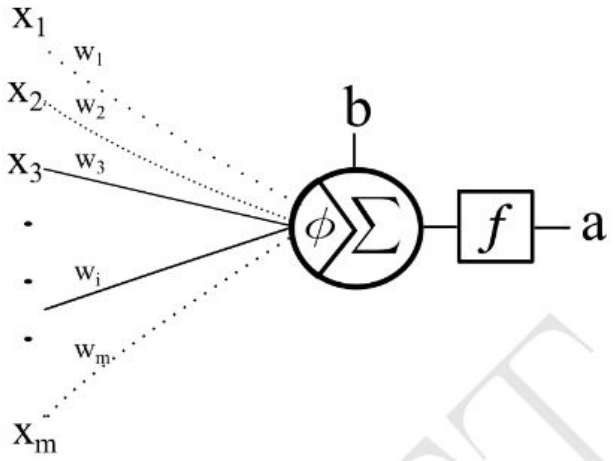

Figure 3-2: The focusing neuron model[3]

Here the focus attachment φ allows the neuron to change its receptive field and adjust the aperture size.

In this project, we replace the dense layers in deep q learning with focused neurons. We will construct deep q-networks of focusing neurons for the following problems:

1) CartPole

   A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The pendulum starts upright, and the goal is to prevent it from falling over by increasing and reducing the cart's velocity.

   There are two actions taken, right and left moves, respect to these observation points is called Box() observation in the OpenAI Gym, and they are not discrete; Cart Position, Cart Velocity, Pole Angle, Pole Velocity At Tip

   Input(4)-Neuron(20)-Output Qlayer(2)

2) Acrobot

   Acrobot is a 2-link pendulum with only the second joint actuated. Initially, both links point downwards. The goal is to swing the end-effector at a height at least the length of one link above the base.

   The observation space is a numpy array in this environment. State calculation, [cos(theta1) sin(theta1) cos(theta2) sin(theta2) thetaDot1 thetaDot2]

   The action is either applying +1, 0 or -1 torque on the joint between the two pendulum links.

3) MountainCar

   A car is on a one-dimensional track, positioned between two mountains. The goal is to drive up the mountain.

The observation space is a numpy space that contains position and velocity. The action space is push left, right, and no push.

4) Own Build Environment: Find The Center

In this environment, there is a red arc that tries to find center of the two-dimensional grid, with constant velocity and speed.

The observation space is the distance between center and arc respect to the X-axis and Y-axis.

The action space is up, right, left, and right.

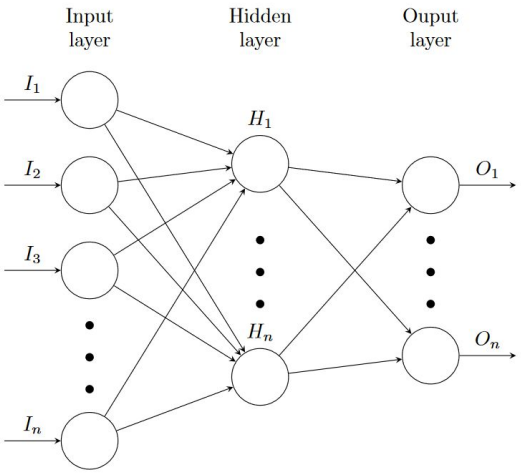

Figure 3-2: Neural network representation

For each environment observation space taken for input size and action space taken as output size.

3.1 Demo Software Design

We will use collab to demonstrate experiments. The system user interface will be made Google Colab .ipynb application pages. In .ipnyb page users can see training and test results, variable changes and model settings. Another thing user can see the backend code on Github. Users will be able to select how many episodes will run, how many iterations will apply on an episode, learning rate, and switching between Dense and Focusing neuron types.

3.2 Planned Experiments

In DQN we train the model while running the environment and create our dataset according to returning data from the environment about the model. After the running tests for Dense and Focussing neurons on an environment, we will use plots to demonstrate some metrics, these are reward taten by the agent on each episode, and loss comparison to show each neuron types efficiency.

4. References

1. Géron, Aurélien. Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems. " O'Reilly Media, Inc.", 2017.
2. Online: https://github.com/btekgit/FocusingNeuron, date accessed: Feb 2020
3. Boray, F. "An Adaptive Locally Connected Neuron Model: Focusing Neuron." ArXiv.org, 31 July 2019, https://arxiv.org/abs/1809.09533.
4. Kuang, Nikki Lijing, Clement H. C. Leung, and Vienne W. K. Sung. "Stochastic Reinforcement Learning." 2018 IEEE First International Conference on Artificial Intelligence and Knowledge Engineering (AIKE) (2018): n. pag. Crossref. https://arxiv.org/abs/1902.04178
5. Hasselt, Hado van, Guez, Arthur, and Silver, David "Deep Reinforcement Learning with Double Q-learning." ArXiv.org, 22 Sep 2015, https://arxiv.org/abs/1509.06461
6. Online:https://www.ibm.com/developerworks/ssa/library/cc-reinforcement-learning-train-software-agent/index.html, date accessed: May 2020
7. Online:https://ithelp.ithome.com.tw/articles/10228127, date accessed: May 2020
8. Online:https://miro.medium.com/max/2000/1*BgOahnFLqAbYl4_eIUBTLg.jpeg, date accessed: May 2020
9. Online:https://www.freecodecamp.org/news/improvements-in-deep-q-learning-dueling-double-dqn-prioritized-experience-replay-and-fixed-58b130cc5682/, date accessed: May 2020
10. Online:https://devhunteryz.wordpress.com/2019/02/10/kendi-kendine-ogrenen-yz-ajanlari-bolum-ii-derin-q-ogrenme/amp/, date accessed: May 2020