Dense vs Focused in DQN for Reinforcement Learning

System Design Document

V 0.1

20.04.2020

Ersin ÇEBİ

Prepared for

COMP4902 Graduation Design Project



**Table of Contents**

1. Introduction 1

1.1. Purpose of the System 1

1.2. Design Goals 1

1.3. Definitions, Acronyms, and Abbreviations 1

2. Current Software Architecture 1

3. Proposed Software Architecture 1

3.1. Overview 1

3.2. System Decomposition 1

3.3. Hardware Software Mapping 2

3.4. Global Software Control 2

3.5. Boundary Conditions 2

4. References 2

SYSTEM DESIGN DOCUMENT

# Introduction

SDD document reports the transformation of the analysis model to system design model. SDD documents contains solution domain which is proposed and specified, design goals, subsystem decomposition, strategies and the definitions of subsystems and interfaces. Mainly, SDD portrays a virtual system that includes all of the specifications and requirements in RAD, and will create a service in boundaries between subsystems and interfaces.

## Purpose of the System

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

## Design Goals

The model should be efficient. Optimized for seamless interaction with existing system is a must. Based on non-functional requirements the next design targets must 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 the our most important goal is stability. We want to make the model works

on 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:** Model’s accuracy and test results are should be better than Dense.

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

## Definitions, Acronyms, and Abbreviations

**Focused:** Focus scales weights and change 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.

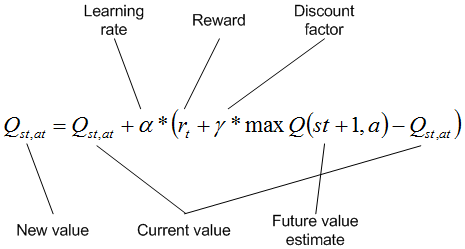
**DENSE**: A dense layer is just a regular layer of neurons in a neural network. Each neuron receives input from all the neurons in the previous layer, thus densely connected.

**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.

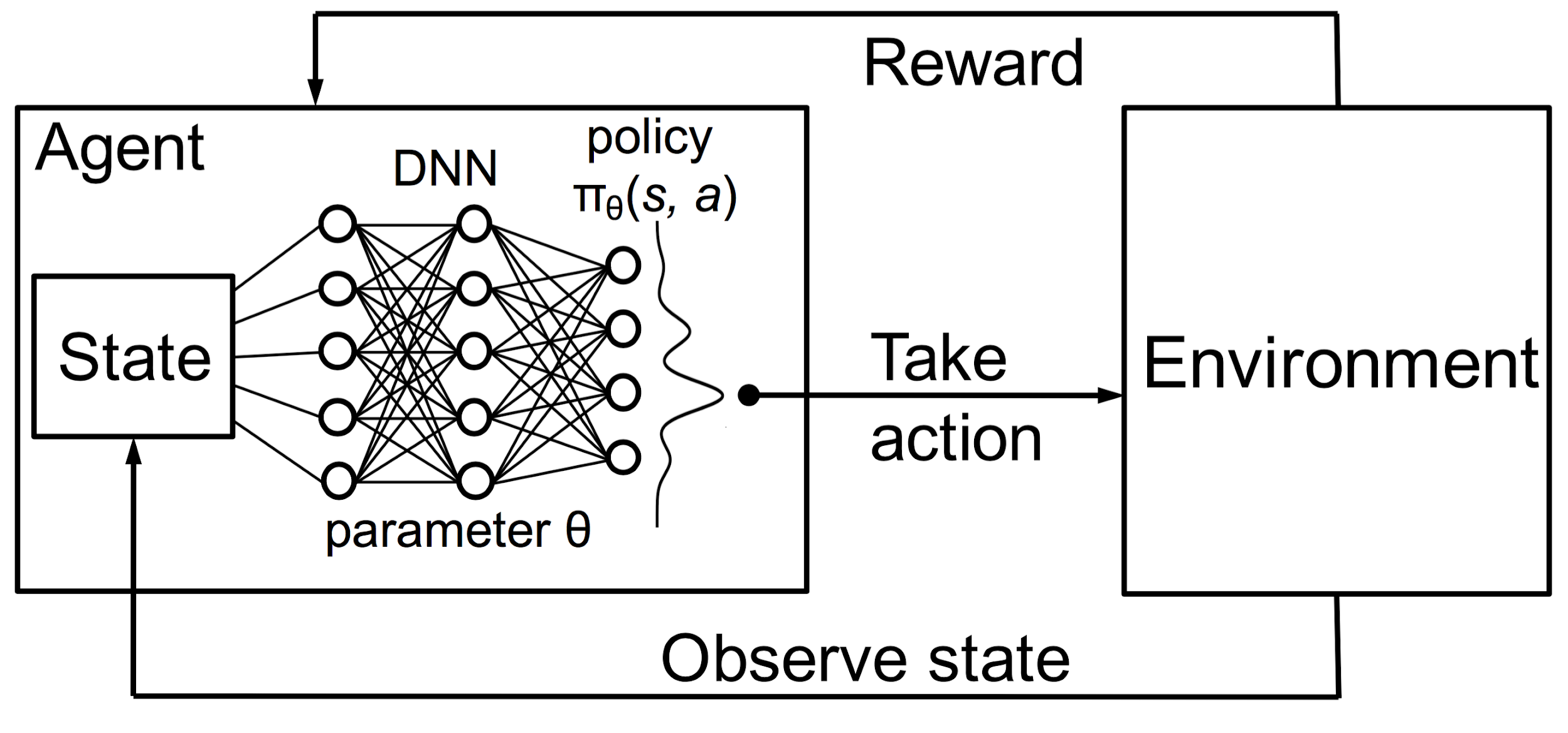
# Current Software Architecture

There is plenty of ways of reinforcement learning. One is q learning, this approach is uses dynamic programing principles. First we set random actions for states in 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 a equation in Figure 2-1:



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

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 a action on a state; and the fourth layer may predict a proper outcome that the current states action.



*Figure 2-2: Reinforcement Learning using a neural network policy*

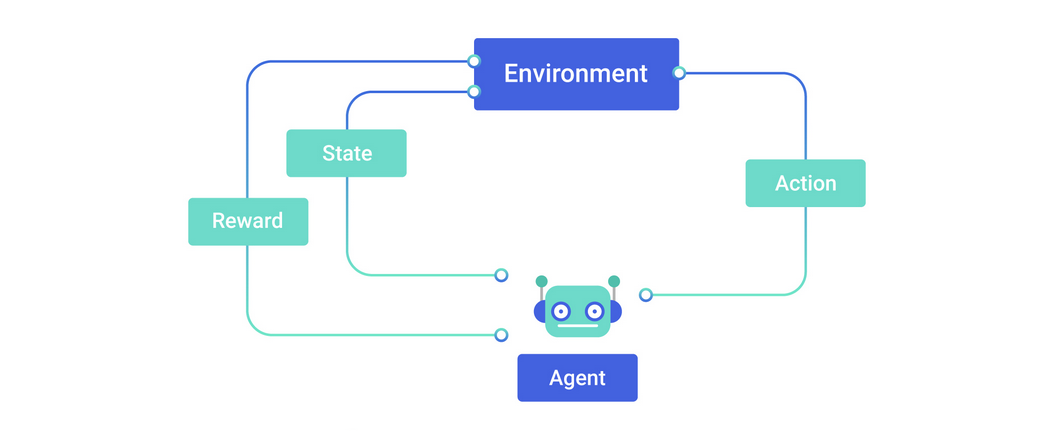
# Proposed Software Architecture

## Overview

Gym library gives us agent and environment 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 the attaining of these goals. Because, according to its actions of its environment the agent gets either prize or penalty.

According to problem we give to agent, agent takes action. So, what decides to this problem and action? The answer is environment, environment is 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 affects 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.

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



*Figure 3.1-1: Agent deciding diagram*

We said that the agent takes actions according to its environment, by the previous actions outcome by meaning that the state(observation), reward but how agent does makes this choices? The answer is policy. Policy is the algorithm that determines the action of agent can use in the environment. I use DQN as policy, that takes all observation as input and creates a 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, I expect 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 *π(at|st) –* that is, it is a conditional probability distribution of taking an action *at* in a given state *st*. Policies can be deterministic, wherein the exact value of *at* is known at *st*, or can be stochastic where *at* 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 a episode at the same time and choice the best ones, let say there is 20 succeeded, in next episode you create 4 offspring each of the successful agents for the next generation.

## System Decomposition

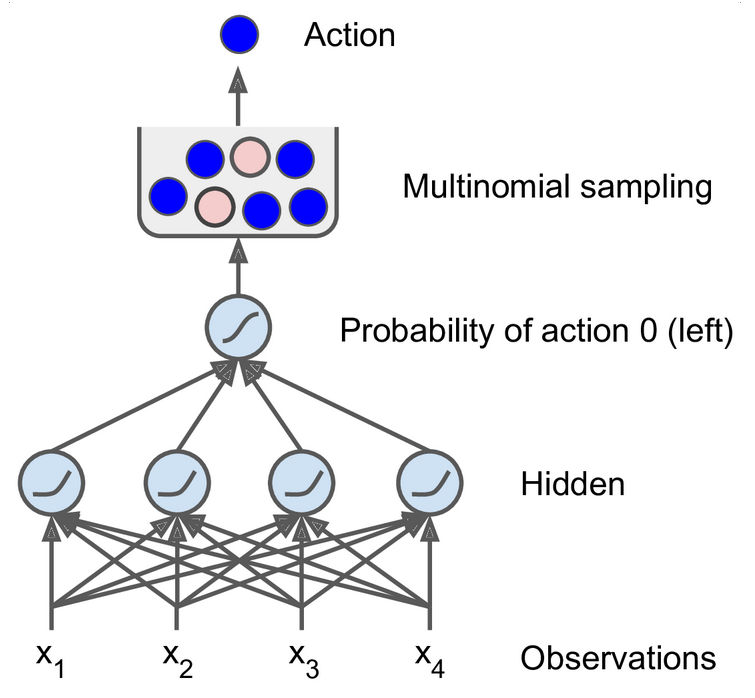
// I yerine We kullan

//passive kullan

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

Gym library gives environment type options, which is 3D, 2D or algorithm problems just on console screen. Also it is easy to set up, we just need to give how episode will it run and then set our algorithm that is it. Episode mean iteration, on each iteration there are steps, for each step agent takes an action, end of all this procedures environment returns four basic variable about each step on an episode, these are observation for current state or coordinate of agent on grid, a extra parenthesis for coordinate its for 2D environments, 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 environment. These variable returns after we call step() function, first we send an action into our environment and this function returns these variables.

In training agent I will use keras Sequential model with layer Dense and Focused. On the test phase user can switch between Dense and Focused layers. Since Focused layer its receptive field in the spatial domain inputs and by using back-propagation algorithm to learn its focus parameters which control the receptive field locations and apertures, action space of a environment can be too small. That is why I added a large size Dense layer before switch part of the model, in case of choosing Focused as 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*

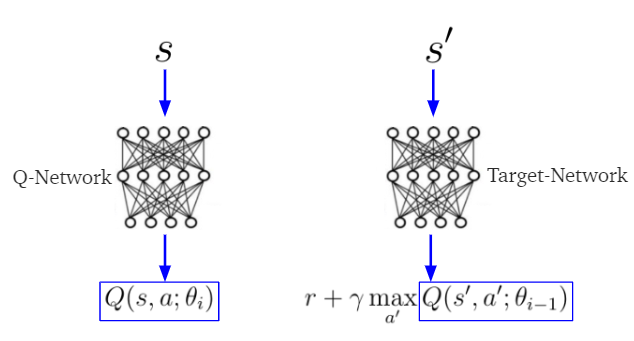
As I mentioned, I use DQN, and DQN has some variants, one is fixed q-value targeting. Deep Q Networks take as input the state of the environment and output a Q value for each possible action. The maximum Q value determines, which action the agent will perform. But the same weights apply to both the target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



*Figure 3.2-2: Calculating weight*

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 model. One is for agent, the other is for the online working model. Instead of update the weight on each run, for example we calculate them on each 50 runs. This way we updated much less often the target model than the online model, as a result 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*

## Software Mapping

## Global Software Control

## Boundary Conditions

Error Conditions:

Sigma :

- Min: 0.01

- Max: 1.0

- Min sigma can affect the performance

Mu :

- Min: 0

- Max: 1.0

- Range can affect the performance and result

# 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 (2019). [link] URL <https://github.com/btekgit/FocusingNeuron>
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>