A
PROJECT REPORT
ON
Q LEARNING FOR 2048

Submitted in partial fulfillment of
the requirements for the award of the degree of

Bachelor of Technology
in
Information Technology

Submitted by
Mr.Akshay Ashok Mehta (Roll No.20100741)
Ms.Prajakta Sidheshvar Mhetre (Roll No.20140731)
Mr.Sagar Prasad Moghe (Roll No.20140732)
Mr.Pranav Prafulla Salvi (Roll No.20140745)

Under the guidance of
Prof.V. J. Kadam
Prof.Dr.S.R.Sutar

DEPARTMENT OF INFORMATION TECHNOLOGY
DR.BABASAHEB AMBEDKAR TECHNLOGICAL UNIVERSITY,
LONERE, RAIGAD-MAHARASHTRA, INDIA-402103

2017-2018

DR.BABASAHEB AMBEDKAR TECHNOLOGICAL UNIVERSITY
LONERE-402103,TAL.MANGAON, DIST. RAIGAD (MS) INDIA

Certificate

This is to certify that the project entitled Q-Learning For 2048 submitted by Mr.Akshay Ashok Mehta (20100741), Miss.Prajakta Sidheshvar Mhetre (20140731), Mr.Sagar Prasad Moghe (20140732), Mr.Pranav Prafulla Salvi (20140745) under Final Year Bachelor of Technology Project for the partial fulfilment of the requirement for the award of a degree Bachelor of Technology in Information Technology to the Dr. Babasaheb Ambedkar Technological University, Lonere, is a bonafide work carried out during academic year 2017-18.

Prof.V.J.Kadam
Department of Information Technology

Prof.Dr.S.R.Sutar
Department of Information Technology

Prof.Dr.S.M.Jadhav
Head of Department,Information Technology

Mr. Swapnil Sawarkar
External Examiner

Acknowledgments

We are pleased to present this final year project report entitled Q-Learning For 2048. It is indeed a great pleasure and a moment of immense satisfaction for us to express our sense of profound gratitude and indebtedness towards our guide Prof. V.J.Kadam and Prof.Dr.S.R.Sutar whose enthusiasm are the source of inspiration for us. We are extremely thankful for the guidance and untiring attention, which they bestowed on us right from the beginning. Their valuable and timely suggestions at crucial stages and above all constant encouragement have made it possible for us to achieve this work. We would also like to give our sincere thanks to Dr. S.M. JADHAV Head of INFORMATION TECHNOLOGY for necessary help and providing us the required facilities for completion of this final year project report. We would like to thank the entire Teaching staffs who are directly or indirectly involved in the various data collection and software assistance to bring forward this final year project report. We express our deep sense of gratitude towards our parents for their sustained cooperation and wishes, which have been a prime source of inspiration to take this final year project work to its end without any hurdles.Last but not the least, we would like to thank all my B.Tech. colleagues for their co-operation and useful suggestion and all those who have directly or indirectly helped us in completion of this final year project work.

Date Place

Abstract

In this project, we explore the possibility of implementing Q Learning algorithm using a Neural Network to play the popular game 2048. Initially, we atudy the game to further build a fullfledged and functioning model of the game. We use a randomizing function to automate the game and set a benchmark. A more sophisticated approach is then adopted using neural network through concepts of Q learning. This model is tested alongside the simple random model to quantify the improvements achieved.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Reinforcement Learning has enjoyed a great increase in popularity over the past decade by controlling how agents can take optimal decisions when facing uncertainty. The first best-known story is probably TD Gammon, a Reinforcement Learning algorithm which achieved a master level of play at backgammon Tesauro[1995]. Recently Q Learning has scaled Reinforcement Learning methods to a new range of problems and thus to media success. Googles Deep Mind developed algorithms that were able to successfully play Atari games Mnih et al. [2013] and defeat the world Go champion Silver et al., 2016. This recent AI accomplishment is considered as a huge leap in Artificial Intelligence since the algorithm should search through an enormous state space before making a decision.

1.2 Project Objective

After reviewing the state of the art in Reinforcement Learning, we chose to work on an average complexity game to obtain results within the project timeline. We have studied the game of 2048 which is played on a 4 * 4 grid with each cell being a power of 2. The objective of the game is to reach the score of 2048 by shifting the grid along any of the four directions up, down, right, left and merging two consecutive cells with same value. Such a game presents an intuitive human strategy that consists of keeping larger elements in a corner. Starting from a random algorithm, we aim at learning such a strategy and test if we can outperform the human player.

1.3 Project Overview

2048 is a recently developed tile sliding game with simple game mechanism but with various mathematical and statistical properties which make it a challenging problem for AI. Dierent types of domain specic heuristics and search algorithms such as minimax, expectiminimax, Flat Monte Carlo Search, can be found in some heated discussions online. However, no formal publication is found about this problem, and Q-Learning as a good candidate for this problem, is not seen in any public-available documentation yet. Therefore, in this work Q-Learning is tested with dierent options of its default policy, including the choice of complete random or some heuristics, or the biasing towards some heuristics with adjustable parameters. The results shows solid performance of Q-Learning, especially when coupled with a carefully designed do main specic heuristic for its default policy.

Chapter 2

2048 - The Game

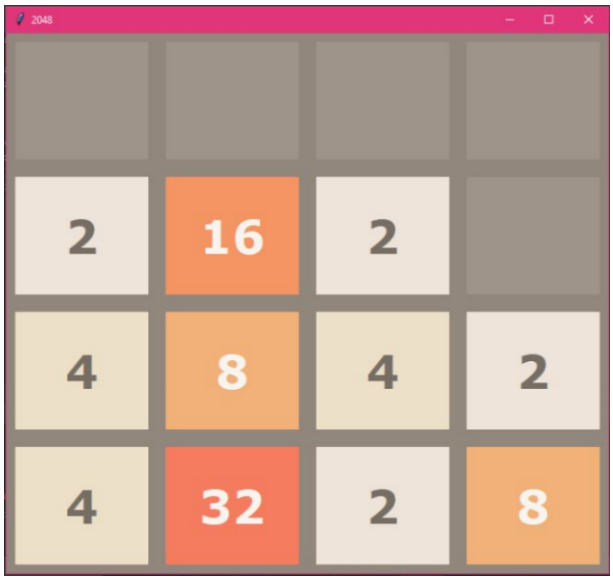

Figure 2.1: A game of 2048 in progress

2048 is a single-player sliding block puzzle game by Italian web developer Gabriele Cirulli. 2048 was originally written in JavaScript and Cascading Style Sheet during a weekend, and released on March 9, 2014, as free and open-source software subject to the MIT license. Clones written in C++ and Vala are available. There is also a version for the Linux terminal.

The game's objective is to slide numbered tiles on a grid to combine them to create a tile with the number 2048; however, you can keep playing the game, creating tiles with larger numbers (such as a 32,768 tile).

2048 has been described as very similar to the Threes! application released a month earlier. Cirulli himself described 2048 as a clone of Veewo Studios' application 1024, who has actually said in the description of the application to be a clone of Threes!

2.1 Development

19-year-old Cirulli created the game in a single weekend as a test to see if he could program a game from scratch; he was surprised when his game received over 4 million visitors in less than a week, especially since it was just a weekend project. "It was a way to pass the time", he said. The game is free to play, Cirulli having said that he was unwilling to make money from something that [he] didn't invent. He released a free app version of the game for iOS and Android in May 2014.

2048 became a viral hit. The game has been described by the Wall Street Journal as almost like Candy Crush for math geeks", and Business Insider called it "Threes on steroids".

As the source code is available, many additions to the original game, including a score leaderboard and improved touchscreen playability have been written by other people and subsequently made available to the public. Spinoffs have been released online, as applications and for the Nintendo 3DS, and include versions with elements from Doge, Doctor Who, Flappy Bird and Tetris; there has also been a 3D version and ones with bigger or smaller grids. Cirulli sees these as "part of the beauty of open source software" and does not object to them "as long as they add new, creative modifications to the game". In 2014, an unofficial clone of the game was published in the iOS app store by Ketchapp, monetized with advertising.

2.2 Rules for 2048

The 2048 game represents a 4 X 4 grid where the value of each cell is a power of 2. An action can be any of the 4 movements: up, down, left and right. When an action is performed, all cells move in the chosen direction. Any two adjacent cells with the same value (power of 2) along this direction merge to form one single cell with value equal to the sum of the two cells (i.e. the next power of 2). After each move the board will randomly generate 0 or 1 new blocks with value 2 or 4 understand given probability in available space.

The score for each move is accumulated. If all spaces on the board are full and the board cannot take any action to move, then the game ends. The objective

of the game is to combine as many cells as possible until reaching 2048 or even a higher value. The performance are measured by five factors, depth of the tree, average movements till game ends, largest tile in the board, average total score, occurrence of large tiles(larger than 128).

2.3 Gameplay

2048 is played on a grey 4 X 4 grid, with numbered tiles that slide smoothly when a player moves them using the four arrow keys. Every turn, a new tile will randomly appear in an empty spot on the board with a value of either 2 or 4. Tiles slide as far as possible in the chosen direction until they are stopped by either another tile or the edge of the grid. If two tiles of the same number collide while moving, they will merge into a tile with the total value of the two tiles that collided. The resulting tile cannot merge with another tile again in the same move. Higher-scoring tiles emit a soft glow.

A scoreboard on the upper-right keeps track of the user's score. The user's score starts at zero, and is incremented whenever two tiles combine, by the value of the new tile. As with many arcade games, the user's best score is shown alongside the current score.

The game is won when a tile with a value of 2048 appears on the board, hence the name of the game. After reaching the 2048 tile, players can continue to play (beyond the 2048 tile) to reach higher scores. When the player has no legal moves (there are no empty spaces and no adjacent tiles with the same value), the game ends.

The simple gameplay mechanics (just four directions) allowed it to be used in a promo video for the Myo gesture control armband, the availability of the code underneath allowed it to be used as a teaching aid for programming, and the second-place winner of a coding contest at Matlab Central Exchange was an AI system that would play 2048 on its own.

There also exists another type of this game, called 177147, which is actually used with exponents of 3 instead of 2.

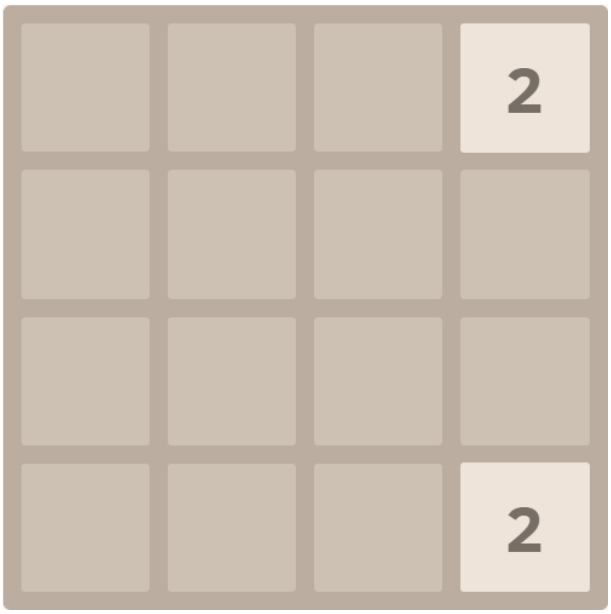

Figure 2.2: New game

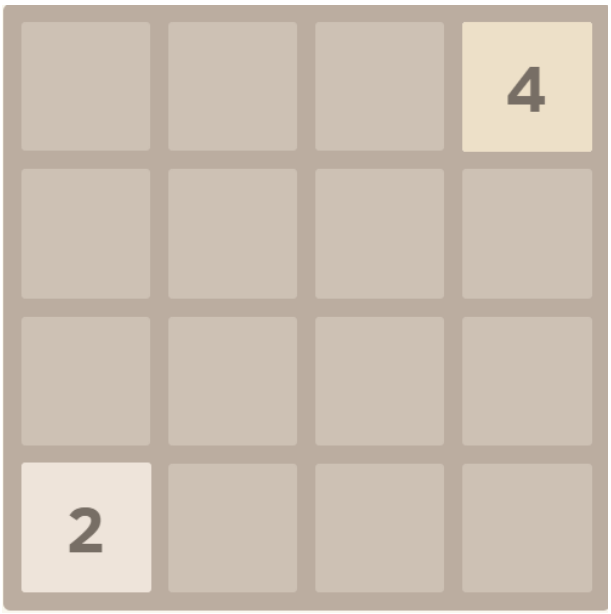

Figure 2.3: UP Move

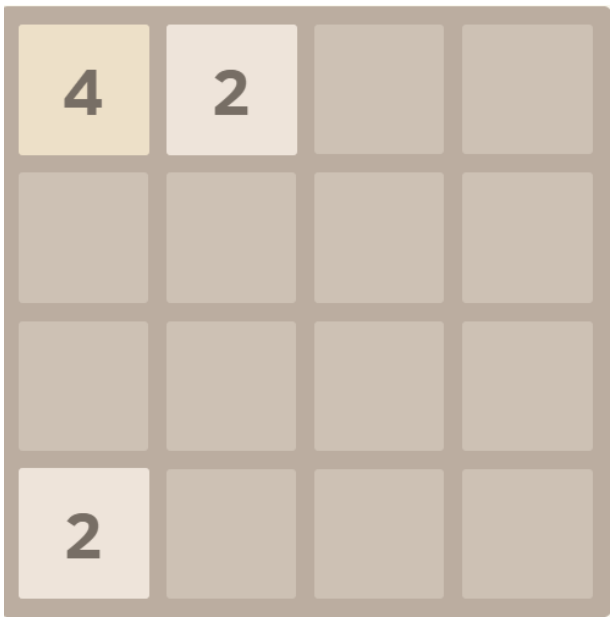

Figure 2.4: LEFT Move

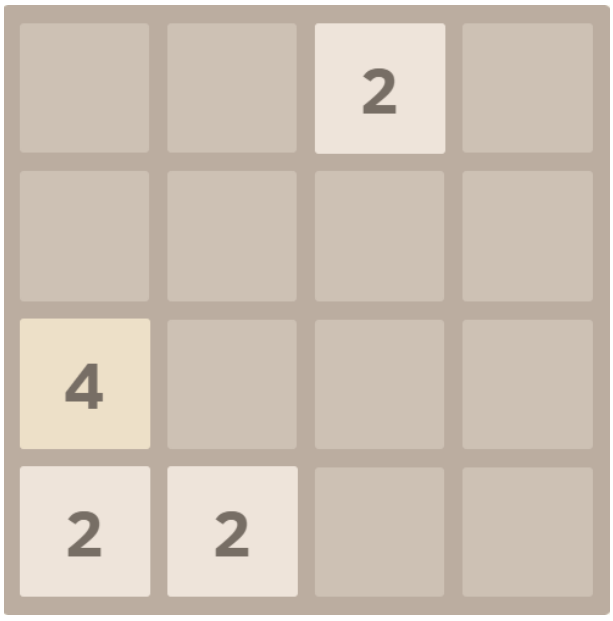

Figure 2.5: DOWN Move

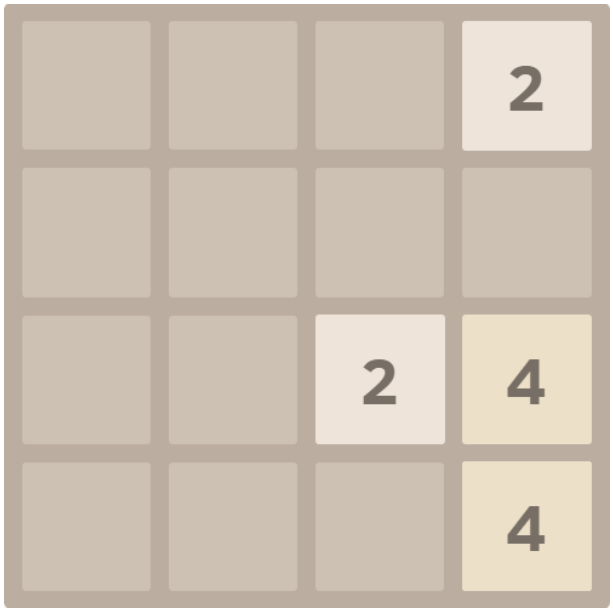

Figure 2.6: RIGHT Move

Chapter 3

Q Learning

3.1 Introduction

Q-learning is a reinforcement learning technique used in machine learning. The technique does not require a model of the environment. Q-learning can handle problems with stochastic transitions and rewards, without requiring adaptations.

For any finite Markov decision process (FMDP), Q-learning eventually finds an optimal policy, in the sense that the expected value of the total reward return over all successive steps, starting from the current state, is the maximum achievable. Q-learning can identify an optimal action-selection policy for any given FMDP.

"Q" names the function or equivalent that returns the reward used to provide the reinforcement and can be said to stand for the "quality" of an action taken in a given state.

3.2 History

Q-learning was introduced by Watkins in 1989. A convergence proof was presented by Watkins and Dayan in 1992. A more detailed mathematical proof was given by Tsitsiklis in 1994, and by Bertsekas and Tsitsiklis in their 1996 Neuro-Dynamic Programming book.

Watkins was addressing Learning from delayed rewards, the title of his PhD Thesis. Eight years earlier in 1981 the same problem under the name of Delayed reinforcement learning was solved by Bozinovsky's Crossbar Adaptive Array (CAA). It was initially published in 1982. The memory matrix W(a,s) is the same as the Q-table of Q-learning. The architecture introduced the term

state evaluation in reinforcement learning. The crossbar learning algorithm, written in mathematical pseudocode in the paper, in each iteration performs the following computation:

- In state s perform action a;
- In state s perform action a;
- Receive consequence state s;
- Compute state evaluation v(s);
- Update crossbar value W(a,s) = W(a,s) + v(s).

The term secondary reinforcement is borrowed from animal learning theory, to model state values via backpropagation: the state value v(s) of the consequence situation is backpropagated to the previously encountered situations. CAA computes state values vertically and actions horizontally (the "crossbar"). Demonstration graphs showing delayed reinforcement learning contained states (desirable, undesirable, and neutral states), which were computed by the state evaluation function. This learning system was a forerunner of the Q-learning algorithm.

A patented application of Q-learning to deep learning, by Google DeepMind, titled "deep reinforcement learning" or "deep Q-learning" that can play Atari 2600 games at expert human levels was presented in 2014.

3.3 Variants of Q Learning

3.3.1 Deep Q-Learning

The DeepMind system used a deep convolutional neural network, with layers of tiled convolutional filters to mimic the effects of receptive fields. Reinforcement learning is unstable or divergent when a nonlinear function approximator such as a neural network is used to represent Q. This instability comes from the correlations present in the sequence of observations, the fact that small updates to Q may significantly change the policy and the data distribution, and the correlations between Q and the target values.

The technique used experience replay, a biologically inspired mechanism that uses a random sample of prior actions instead of the most recent action to proceed. This removes correlations in the observation sequence and smoothing changes in the data distribution. Iterative update adjusts Q towards target

values that are only periodically updated, further reducing correlations with the target.

3.3.2 Double Q-Learning

Because the maximum approximated action value is used in Q-learning, in noisy environments Q-learning can sometimes overestimate the action values, slowing the learning. A variant called Double Q-learning was proposed to correct this. This algorithm was later combined with deep learning, as in the DQN algorithm, resulting in Double DQN, which outperforms the original DQN algorithm.

3.3.3 Delayed Q-Learning

Delayed Q-learning is an alternative implementation of the online Q-learning algorithm, with probably approximately correct (PAC) learning.

3.3.4 Greedy GQ

Greedy GQ is a variant of Q-learning to use in combination with (linear) function approximation. The advantage of Greedy GQ is that convergence is guaranteed even when function approximation is used to estimate the action values.

Chapter 4

System Analysis

System analysis is the process of gathering and interpreting facts, diagnosing problems and using the information to recommend improvements on the system. System analysis is a problem solving activity that requires intensive communication between the system users and system developers.

System analysis or study is an important phase of any system development process. The system is viewed as a whole, the inputs are identified and the system is subjected to close study to identify the problem areas. The solutions are given as a proposal. The proposal is reviewed on user request and suitable changes are made. This loop ends as soon as the user is satisfied with the proposal.

4.1 Software And Hardware Requirement Specifications

4.1.1 Software Requirements

1. Operating System - LINUX

2. IDE - Sublime Text Editor

3. Python 2 or Python 3

4. Libraries :

   (a) Pytorch
   (b) Matplotlib

(c) Numpy

(d) Tkinter

(e) Os

4.1.2 Hardware Requirements

1. Cuda supported GPU

2. RAM - 4 GB

Chapter 5

Design

5.1 Visualization

5.1.1 Tkinter

Tkinter is a Python binding to the Tk GUI toolkit. It is the standard Python interface to the Tk GUI toolkit, and is Python's de facto standard GUI. Tkinter is included with the standard Microsoft Windows and Mac OS X install of Python.

The name Tkinter comes from Tk interface. Tkinter was written by Fredrik Lundh.

As with most other modern Tk bindings, Tkinter is implemented as a Python wrapper around a complete Tcl interpreter embedded in the Python interpreter. Tkinter calls are translated into Tcl commands which are fed to this embedded interpreter, thus making it possible to mix Python and Tcl in a single application.

Python 2.7 and Python 3.1 incorporate the "themed Tk" ("ttk") functionality of Tk 8.5. This allows Tk widgets to be easily themed to look like the native desktop environment in which the application is running, thereby addressing a long-standing criticism of Tk (and hence of Tkinter).

There are several popular GUI library alternatives available, such as wxPython, PyQt (PySide), Pygame, Pyglet, and PyGTK.

Tkinter is free software released under a Python license.

Frame

A frame is rectangular region on the screen. The frame widget is mainly used as a geometry master for other widgets, or to provide padding between other widgets.

When to use the Frame Widget

Frame widgets are used to group other widgets into complex layouts. They are also used for padding, and as a base class when implementing compound widgets.

Grid

The Grid geometry manager puts the widgets in a 2-dimensional table. The master widget is split into a number of rows and columns, and each cell in the resulting table can hold a widget.

When to use the Grid Manager

The grid manager is the most flexible of the geometry managers in Tkinter. If you dont want to learn how and when to use all three managers, you should at least make sure to learn this one.

The grid manager is especially convenient to use when designing dialog boxes. If youre using the packer for that purpose today, youll be surprised how much easier it is to use the grid manager instead. Instead of using lots of extra frames to get the packing to work, you can in most cases simply pour all the widgets into a single container widget, and use the grid manager to get them all where you want them.

Chapter 6

Network

6.1 Pytorch

PyTorch is an open source machine learning library for Python, based on Torch, used for applications such as natural language processing. It is primarily developed by Facebook's artificial-intelligence research group, and Uber's "Pyro" software for probabilistic programming is built on it.

6.1.1 Torch

Torch is an open source machine learning library, a scientific computing framework, and a script language based on the Lua programming language. It provides a wide range of algorithms for deep learning, and uses the scripting language LuaJIT, and an underlying C implementation.

torch

The core package of Torch is torch. It provides a flexible N-dimensional array or Tensor, which supports basic routines for indexing, slicing, transposing, type-casting, resizing, sharing storage and cloning. This object is used by most other packages and thus forms the core object of the library. The Tensor also supports mathematical operations like max, min, sum, statistical distributions like uniform, normal and multinomial, and BLAS operations like dot product, matrix-vector multiplication, matrix-matrix multiplication, matrix-vector product and matrix product.

The torch package also simplifies object oriented programming and serialization by providing various convenience functions which are used throughout its packages. The torch.class(classname, parentclass) function can be used to create object factories (classes). When the constructor is called, torch initializes and sets a Lua table with the user-defined meta table, which makes the table an object.

Objects created with the torch factory can also be serialized, as long as they do not contain references to objects that cannot be serialized, such as Lua coroutines, and Lua user data. However, user data can be serialized if it is wrapped by a table (or meta table) that provides read() and write() methods.

nn

The nn package is used for building neural networks. It is divided into modular objects that share a common Module interface. Modules have a forward() and backward() method that allow them to feedforward and backpropagate, respectively. Modules can be joined together using module composites, like Sequential, Parallel and Concat to create complex task-tailored graphs. Simpler modules like Linear, Tanh and Max make up the basic component modules. This modular interface provides first-order automatic gradient differentiation.

Loss functions are implemented as sub-classes of Criterion, which has a similar interface to Module. It also has forward() and backward() methods for computing the loss and backpropagating gradients, respectively. Criteria are helpful to train neural network on classical tasks. Common criteria are the Mean Squared Error criterion implemented in MSE Criterion and the cross-entropy criterion implemented in Class NLL Criterion.

It also has Stochastic Gradient class for training a neural network using Stochastic gradient descent, although the Optim package provides much more options in this respect, like momentum and weight decay regularization.

Chapter 7

Functions

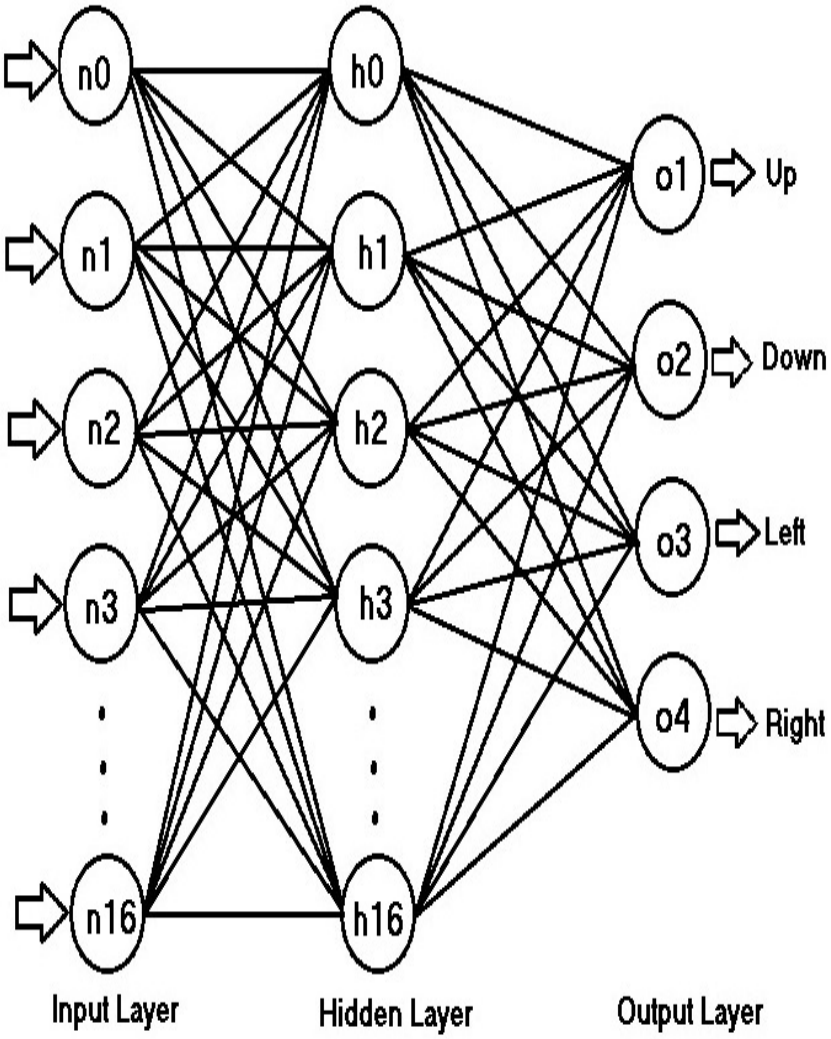

Figure 7.1: Neural Networks

7.1 Reward Function

The reward system is a group of neural structures responsible for incentive salience (i.e., motivation and "wanting", desire, or craving for a reward), associative learning (primarily positive reinforcement and classical conditioning), and positive emotions, particularly ones which involve pleasure as a core component (e.g., joy, euphoria and ecstasy). Reward is the attractive and motivational property of a stimulus that induces appetitive behavior, also known as approach behavior, and consummatory behavior. In its description of a rewarding stimulus (i.e., "a reward"), a review on reward neuroscience noted, "any stimulus, object, event, activity, or situation that has the potential to make us approach and consume it is by definition a reward." In operant conditioning, rewarding stimuli function as positive reinforcers; however, the converse statement also holds true: positive reinforcers are rewarding.

Primary rewards are a class of rewarding stimuli which facilitate the survival of one's self and offspring, and include homeostatic (e.g., palatable food) and reproductive (e.g., sexual contact and parental investment) rewards. Intrinsic rewards are unconditioned rewards that are attractive and motivate behavior because they are inherently pleasurable. Extrinsic rewards (e.g., money or seeing one's favorite sports team winning a game) are conditioned rewards that are attractive and motivate behavior, but are not inherently pleasurable. Extrinsic rewards derive their motivational value as a result of a learned association (i.e., conditioning) with intrinsic rewards. Extrinsic rewards may also elicit pleasure (e.g., euphoria from winning a lot of money in a lottery) after being classically conditioned with intrinsic rewards.

Survival for most animal species depends upon maximizing contact with beneficial stimuli and minimizing contact with harmful stimuli. Reward cognition serves to increase the likelihood of survival and reproduction by causing associative learning, eliciting approach and consummatory behavior, and triggering positive emotions. Thus, reward is a mechanism that evolved to help increase the adaptive fitness of animals.

7.1.1 Bellman-Ford Equation

The BellmanFord algorithm is an algorithm that computes shortest paths from a single source vertex to all of the other vertices in a weighted digraph. It is slower than Dijkstra's algorithm for the same problem, but more versatile, as it is capable of handling graphs in which some of the edge weights are negative numbers. The algorithm was first proposed by Alfonso Shimbel in 1955, but is instead named after Richard Bellman and Lester Ford, Jr., who

published it in 1958 and 1956, respectively. Edward F. Moore also published the same algorithm in 1957, and for this reason it is also sometimes called the BellmanFordMoore algorithm.

Negative edge weights are found in various applications of graphs, hence the usefulness of this algorithm. If a graph contains a "negative cycle" (i.e. a cycle whose edges sum to a negative value) that is reachable from the source, then there is no cheapest path: any path that has a point on the negative cycle can be made cheaper by one more walk around the negative cycle. In such a case, the BellmanFord algorithm can detect negative cycles and report their existence.

$$V(s) = \max_{a}(R(s,a) + \gamma V(s'))$$

Figure 7.2: Bellman-Ford Equation

7.2 Activation Function

In computational networks, the activation function of a node defines the output of that node given an input or set of inputs. A standard computer chip circuit can be seen as a digital network of activation functions that can be "ON" (1) or "OFF" (0), depending on input. This is similar to the behaviour of the linear perceptron in neural networks. However, only nonlinear activation functions allow such networks to compute nontrivial problems using only a small number of nodes. In artificial neural networks this function is also called the transfer function.

7.2.1 Rectified Linear Unit

In the context of artificial neural networks, the rectifier is an activation function defined as the positive part of its argument.

This is also known as a ramp function and is analogous to half-wave rectification in electrical engineering. This activation function was first introduced to a dynamical network by Hahnloser et al. in a 2000 paper in Nature with strong

biological motivations and mathematical justifications. It has been demonstrated for the first time in 2011 to enable better training of deeper networks , compared to the widely used activation functions prior to 2011, i.e., the logistic sigmoid (which is inspired by probability theory; see logistic regression) and its more practical counterpart, the hyperbolic tangent. The rectifier is, as of 2018, the most popular activation function for deep neural networks.

A unit employing the rectifier is also called a rectified linear unit (ReLU).

A smooth approximation to the rectifier is the analytic function which is called the softplus function. The derivative of softplus is , i.e. the logistic function.

Rectified linear units find applications in computer vision and speech recognition using deep neural nets.

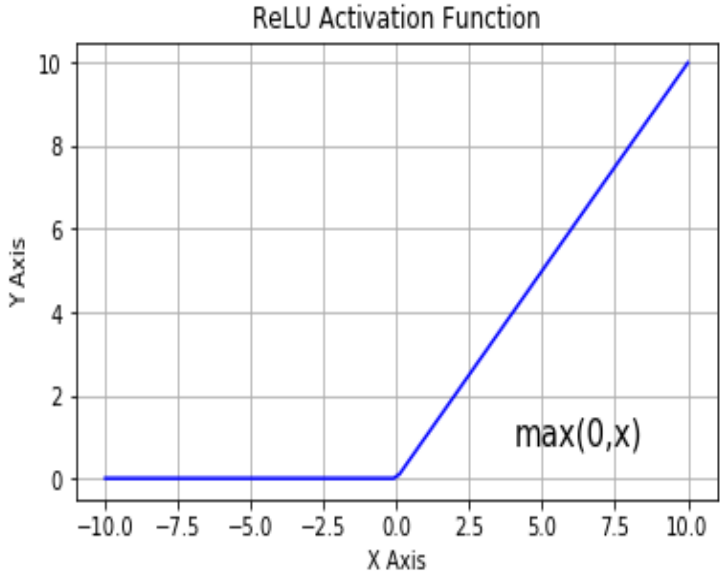

Figure 7.3: ReLU Activation Curve

Advantages of ReLU

- Biological plausibility: One-sided, compared to the antisymmetry of tanh.
- Sparse activation: For example, in a randomly initialized network, only about 50
- Better gradient propagation: Fewer vanishing gradient problems compared to sigmoidal activation functions that saturate in both directions.
- Efficient computation: Only comparison, addition and multiplication.
- Scale-invariant

Rectifying activation functions were used to separate specific excitation and unspecific inhibition in the Neural Abstraction Pyramid, which was trained in a supervised way to learn several computer vision tasks. In 2011, the use of the rectifier as a non-linearity has been shown to enable training deep supervised neural networks without requiring unsupervised pre-training. Rectified linear units, compared to sigmoid function or similar activation functions, allow for faster and effective training of deep neural architectures on large and complex datasets.

7.2.2 Softmax

In probability theory, the output of the softmax function can be used to represent a categorical distribution – that is, a probability distribution over K different possible outcomes. In fact, it is the gradient-log-normalizer of the categorical probability distribution.

The softmax function is used in various multiclass classification methods, such as multinomial logistic regression (also known as softmax regression):206209 , multiclass linear discriminant analysis, naive Bayes classifiers, and artificial neural networks. Specifically, in multinomial logistic regression and linear discriminant analysis, the input to the function is the result of K distinct linear functions

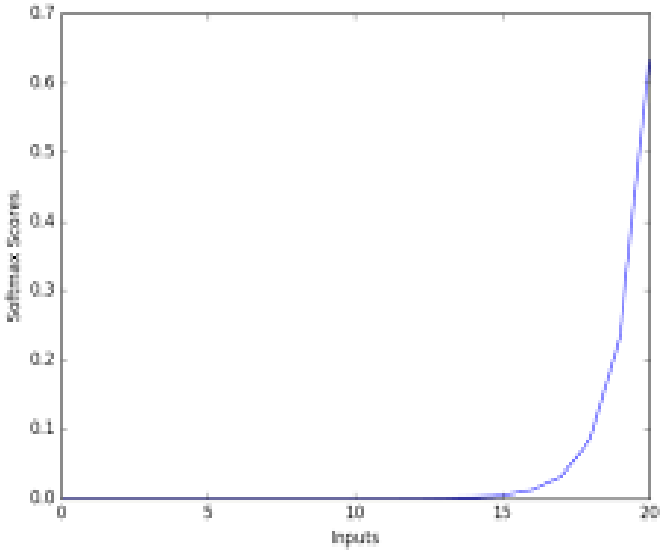

Figure 7.4: Softmax Activation Curve

Chapter 8

Backpropagation

Backpropagation is a method used in artificial neural networks to calculate a gradient that is needed in the calculation of the weights to be used in the network. It is commonly used to train deep neural networks , a term referring to neural networks with more than one hidden layer.

Backpropagation is a special case of an older and more general technique called automatic differentiation. In the context of learning, backpropagation is commonly used by the gradient descent optimization algorithm to adjust the weight of neurons by calculating the gradient of the loss function. This technique is also sometimes called backward propagation of errors, because the error is calculated at the output and distributed back through the network layers.

The backpropagation algorithm has been repeatedly rediscovered and is equivalent to automatic differentiation in reverse accumulation mode. Backpropagation requires a known, desired output for each input valueit is therefore considered to be a supervised learning method (although it is used in some unsupervised networks such as autoencoders). Backpropagation is also a generalization of the delta rule to multi-layered feedforward networks, made possible by using the chain rule to iteratively compute gradients for each layer. It is closely related to the GaussNewton algorithm, and is part of continuing research in neural backpropagation. Backpropagation can be used with any gradient-based optimizer, such as L-BFGS or truncated Newton.

Backpropagation was one of the first methods able to demonstrate that artificial neural networks could learn good internal representations, i.e. their hidden layers learned nontrivial features. Experts examining multilayer feedforward networks trained using backpropagation actually found that many nodes learned features similar to those designed by human experts and those found by neuroscientists investigating biological neural networks in mammalian brains

(e.g. certain nodes learned to detect edges, while others computed Gabor filters). Even more importantly, because of the efficiency of the algorithm and the fact that domain experts were no longer required to discover appropriate features, backpropagation allowed artificial neural networks to be applied to a much wider field of problems that were previously off-limits due to time and cost constraints.

self.learn(BatchsState, BatchNextState, BatchReward, BatchAction)

- Batch-state : The present state of the grid.
- Batch-next-state : The next state which the grid achieves after making the proposed action.
- Batch reward : The reward obtained by playing the proposed action.
- Batch action : The action which is taken at a particular state of the grid.

Chapter 9

Results

Results achieved are as follows :

1. Highest Tile achieved : 512
2. Average number of moves played : 140

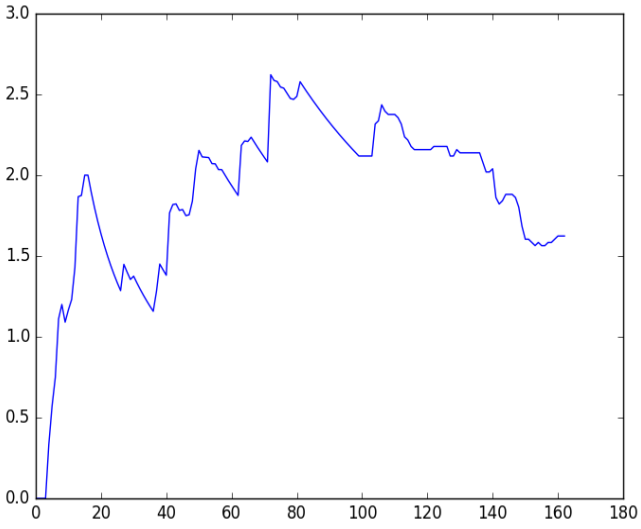

Figure 9.1: Graph on Score obtained

Chapter 10

Conclusion

The puzzle game 2048 constitutes a new interesting challenge for computational intelligence methods. In this study we have shown that for this stochastic game the Q-Learning equipped with a powerful evaluation function. The method of Q-Learning has proved to be rather effective in solving the problem statement. The successful working of the game can be observed through automation.

The highest tile achieved is 512. Thus this method is not ideal for this kind of application. Other previously implemented approaches have proved to be more efficient and reliable. Hence other methods used before, such as Deep Reinforcement Learning, should be given more priority in problems falling in this domain. Therefore, this method is a good benchmark to be compared with other approaches solving similar problems.

Chapter 11

Future Plans

Future work may involve optimizing heuristics for the use of player action in default policy, also optimizing the implementation to reduce the run time, and increase the number of independent runs in each set of tests to get better statistics.Other possibility includes applying the algorithm to a generalized problem with different size of board (this is already supported by current implementation but no systematic experiments are done), or different shape of board (e.g. rectangle instead of square), or board in different dimension (e.g. 3D instead of 2D). With the simple game mechanism and interesting mathematical and statistical properties.

- Addition of negative reward.
- Applying multiple grid games.
- Increasing efficiency of learning.
- Achieving higher scores.

Bibliography

1.https://en.wikipedia.org/wiki/Q-learning
2.https://gabrielecirulli.github.io/2048/
3.https://en.wikipedia.org/wiki/Tkinter
4.https://wiki.python.org/moin/TkInter
5.https://en.wikipedia.org/wiki/PyTorch
6.https://en.wikipedia.org/wiki/Torch(machinelearning)/
7.www.numpy.org/
8.https://wiki.python.org/moin/NumPy 9.https://en.wikipedia.org/wiki/NumPy
10.http://blog.hackerearth.com/explaining-basics-of-machine-learning-algorithms-applications
11.https://sites.google.com/a/umn.edu/fenixshrine/ai-projects-homeworks/ai-for-2048-with-mcts
12.http://ieeexplore.ieee.org/document/6932920/
13.http://www.webpages.ttu.edu/dleverin/neuralnetwork/neuralnetworks.html