

Universitatea POLITEHNICA din Bucureşti
Facultatea de Automatică și Calculatoare,
Departamentul de Calculatoare



LUCRARE DE DIPLOMĂ

Deep Learning în jocuri

Conducător Științific:
As. Drd. Ing. Tudor Berariu

Autor:
Florentina-Ştefania Bratiloveanu

Bucureşti, 2015

University POLITEHNICA of Bucharest
Faculty of Automatic Control and Computers,
Computer Science and Engineering Department



BACHELOR THESIS

Playing games with Deep Learning

Scientific Adviser:
As. Drd. Ing. Tudor Berariu

Author:
Florentina-Ştefania Bratiloveanu

Bucharest, 2015

Abstract

Deep Learning is a new, interesting and a fast-growing field of Machine Learning. It combines the classical model of multilayer perceptrons with layers of feature extraction inspired from visual cortex of the brain. Moreover, dealing with the curse of dimensionality is another advantage when we talk about Convolutional Neural Networks. This paper proposes an alternative to the classical reinforcement learning techniques, such as Q-Learning and SARSA. The question which arises is this: what if we make an agent and allow it to play a game based on information from visual frames and rewards received during the game?

Contents

Abstract	i
1 Introduction	1
1.1 Motivation	1
1.2 Project description	2
1.3 Technologies	3
1.4 Structure of this paper	3
1.5 Copyright infringement	3
2 State of the art	4
2.1 Reinforcement learning	4
2.1.1 History	4
2.1.2 Q-Learning, Sarsa	4
2.2 Neural Networks and Convolutional Neural Networks	6
2.2.1 Data	6
2.2.2 Model	7
2.2.3 Loss functions	13
2.2.4 Optimizations for learning	13
2.2.5 Training and testing	15
2.3 An in-depth look to “Human-level control through deep reinforcement learning” paper	16
2.3.1 Architecture	16
2.3.2 Algorithm	18
2.3.3 Results	19
3 System design and implementation	20
3.1 Q-Learning	20
3.2 Regression with deep-learning	22
3.2.1 First attempt	22
4 Results	26
5 Conclusions	27
6 Future work	28
A Code examples	29
A.1 Model	29
Bibliography	29

Chapter 1

Introduction

1.1 Motivation

Humans have always felt the need to explore and find a way to a better life. Starting with stone tools (2.6 million years ago) and continuing with personal computers are only some proof in human evolution. Lately, we have discovered Machine Learning which have the purpose to allow machines for learning to do different tasks.

Deep learning is a subfield of machine learning area and it is inspired by how the human brain works. Through deep learning, we are trying to solve some of the most pressuring human problems as cancer classification (benign or malignant tumor)[1], self-driving cars based on pedestrian detections[2] or recognizing digits from photos taken with Google Street View[3], all using unsupervised learning of features.

This paper propose an in-dept look of solving reinforcement learning problems, using convolutional neural networks. More exactly, we want to design an agent which is capable of learning to play a game[4] seeing only the pixels from the frames and being rewarded after finishing the game.

This idea was brought to attention in 2013, when Alex Graves et al. from DeepMind¹ came with the idea of creating a convolutional neural network capable of playing different Atari 2600 games, being almost as good as a game tester. We will see later what almost means. The motivation behind this experiment is not only connected with the fact that we have a machine capable of playing games, but the more important problem is that they achieved the generalization of a machine that could learn several types of games, which in fact is the starting of a new revolution in machine learning. If we are capable of making one unique algorithm that can solve multiple tasks, we are capable of simulating the real human brain and capable of reducing complex problems to another ones, much more simple and thus, we can reduce programming burden and solving problems as cancer classification.

Moreover, there are also another types of applications that can suggest the power of convolutional neural networks and one of them is the neural algorithm for creating paintings[5] from ordinary pictures. The inputs is represented by two pictures, one which is the painting (e.g *The Starry Night* of Vincent van Gogh) and the other one which is a normal photo, like a picture with a house. The output of the network is the house painted in van Gogh's technique.

¹<http://deepmind.com/>

1.2 Project description

This paper presents two different algorithms in creating a good agent which can learn playing games. It is worth mentioning that the agent does not know the rules and can not interfere with some rules at the beginning of the game. The whole topic has been splitted in three parts: running Q-Learning on Hanoi Towers game, learning values predicted by Q-Learning using a convolutional neural network and connecting the first two parts together.

First of them is the classical Q-Learning, which is searching the optimal policy for taking actions from each state of the game. We will discuss about exploration vs exploitation problem, how to determine the learning rate suitable for our purposes, how many episodes we have to play until we determine a policy close to the optimal one.

The second topic we approach in this paper is the possibility of predicting continuous output from image. Practically, we take the dataset from Q-Learning which contains images as input and a table of four values corresponding to the four possible actions in Hanoi Tower games: up, down, left and right. Here, we can determine whether we have a good or a bad convolutional neural network. In other words, we will test the capacity of learning samples, the capacity for generalization.

One must pay attention to the data pre-processing, being an important step in data selection. For example, when we learn to classify things, we may not need the color information. Converting data from RGB to YUV gives us the possibility in separating luminance from chrominance.

After that we need to look closer at the model. We need a model that can be proven to converge at the optimal solution and also a model that can learn very fast. Datasets are very big and can contain millions of pictures.

There is also important when we have to stop training to avoid overfitting and our primary target is to minimize the test error. We have to take care how we split dataset in training, testing or validation, how we determine that we can stop training the network and how we can formalize the results.

Another important step is called the loss function, the moment when we determine how far we are from our target. Here, we can find the border between underfitting and overfitting, called the optimum state. If we have high bias, it means that we have underfitting and our classifier is not able to predict well data and if we have high variance, it means that we have learned training samples very well, but can not predict new samples.

The last part is the one where we combine the first two modules. We will try to get a closer look of what DeepMind have done with the Atari paper[4]. The most important part is the connection between the Q-Learning algorithm and the convolutional neural network. We will see how weights are updated according to rewards received during the game.

The main problem is to create an architecture capable of playing any game without adapting initial meta parameters. All the above-mentioned problems will be discussed in this paper and formalized using experimental results, plots or anything that can serve as a proof.

1.3 Technologies

All algorithms described in this paper have been implemented using Torch7¹, a deep learning framework written in Lua², a scripting language based on a C API. For creating the game, generating frames or modifying images LOVE platform³ which is also written in Lua has been used. For interactive computing(image/filter visualization) iTorch⁴ has been prefered. For illustrating different functions Octave⁵ was the best choice open-source.

Why Lua and Torch? They provide a fast environment as opposed to another ones[6], multiple modules with functions already implemented such as transfer functions(tanh, sigmoid), loss functions(Mean Squared Error, Negative Log Likelihood) or convolutional layers(SpatialMaxPooling, SpatialSubSampling)

1.4 Structure of this paper

[Chapter 2 State of the art](#) presents the entire architecture of a network based on a complex research in the machine learning, pattern recognition and reinforcement learning fields. Also, an in-depth look to the “Human-level control through deep reinforcement learning” paper is provided. This paper has been chosen for the fact that it is the primary resource which proves the combination of deep learning and reinforcement learning is possible.

[Chapter 3 System design and implementation](#) describes several architectures used for pursuing the purpose of combining deep learning and reinforcement learning. Every step in choosing the architecture is well-argumentated, either with mathematical reasoning or with empirical reasoning.

[Chapter 4 Results](#) shows the configuration of hyperparameters chosen for tuning every module of learning.

[Chapter 5 Conclusions](#) is dedicated to personal opinions of the paper’s author on this topic and also presents the systematic investigation in gathering information.

[Chapter 6 Future work](#) describes future tunnings of the architectures and also further investigation of one of the follows fields: machine learning, pattern recognition or reinforcement learning.

1.5 Copyright infringement

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page.

¹<http://torch.ch/>

²<http://www.lua.org/about.html>

³https://love2d.org/wiki/Main_Page

⁴<https://github.com/facebook/iTorch>

⁵<https://www.gnu.org/software/octave/>

Chapter 2

State of the art

Being a large domain, deep learning imposes a research in multiple subdomains such as neuroscience, reinforcement learning, neural networks or convolutional neural networks. That being said, this chapter is dedicated to gathering information on each of them.

2.1 Reinforcement learning

2.1.1 History

The history of reinforcement learning starts with the pavlovian experiment[7] in 1903, when Pavlov tried to demonstrate that there is a connection between conditioned and unconditioned stimulus. The unconditioned response of salivation of a dog is fired up by bringing an unconditioned stimulus as food. If we try to make the dog to salivate in the presence of a neutral stimulus(the sound of a bell) we won't succeed, but if we use in the same time the unconditioned and neutral stimulus we can make the dog salivate and transform the bell and salivation in conditioned stimulus and conditioned response.

2.1.2 Q-Learning. Sarsa

Q-Learning

Q-Learning[8] is the algorithm for learning in small steps details about the environment. We know the initial starting state of the game and also a set of possible actions we can take in order to change the current state. At the beginning, we are constrained to choose randomly an action, because we do not know which one could bring us winning the game and this is the exploration phase. We keep randomly picking actions until the game is finished.

A game can have **n** number of states and **a** number of actions. We keep in memory a table **Q** of size **nxa** and we initialize all the corresponding elements with zero. During the game, rewards can be granted. Some of them can be good rewards and some of them can be bad rewards. When we receive rewards, we update the previous state with the score obtained.

Here we have to choose between exploration and exploitation. In the first states of the game, when we know nothing about environment, so we choose to explore new states in order to see if they bring us rewards. After some **e** episodes we learn which actions can bring us closer to the winning of the game. This is the moment when to start to exploit the things we have already learnt in the previous episodes. Assume we are in state **s** and have the next possible actions

$\{a_1, a_2, a_3, a_4\}$. Keep in mind that we recorded the Q-table. At this moment we choose the action which gives us the best utility from states s . If $Q(s, a_3)$ is the maxim value of Q in state s we will pick action a_3 .

Sarsa

Q-Learning and Sarsa look alike except for the fact that Sarsa is updating his policy based on the action it takes and which is not necessary the best action, but it is the action expected to pursue his policy.

When dealing with large negative rewards Q-Learning is dealing better than Sarsa[9] because Q-Learning does not avoid risk situations while Sarsa is avoiding them. Both algorithms have variables that have to be adjusted during the computation.

- **Learning rate** adjusts learning of new information flow. **0** is for not acquiring information and **1** is for considering only the new information.
- **Discount factor** adjust the importance of current rewards(**1**) vs future rewards(**0**)

ϵ -Greedy is the policy for choosing the next action. As we have already mentioned the problem between choosing explorations vs. exploitation, this strategy comes for helping in the first episodes where we don't have too much details about environment. It says that we can choose randomly an action from the set of actions with probability ϵ and choose the best action with probability $1 - \epsilon$.

Figure 2.1 shows behaviour of Q-Learning and Sarsa

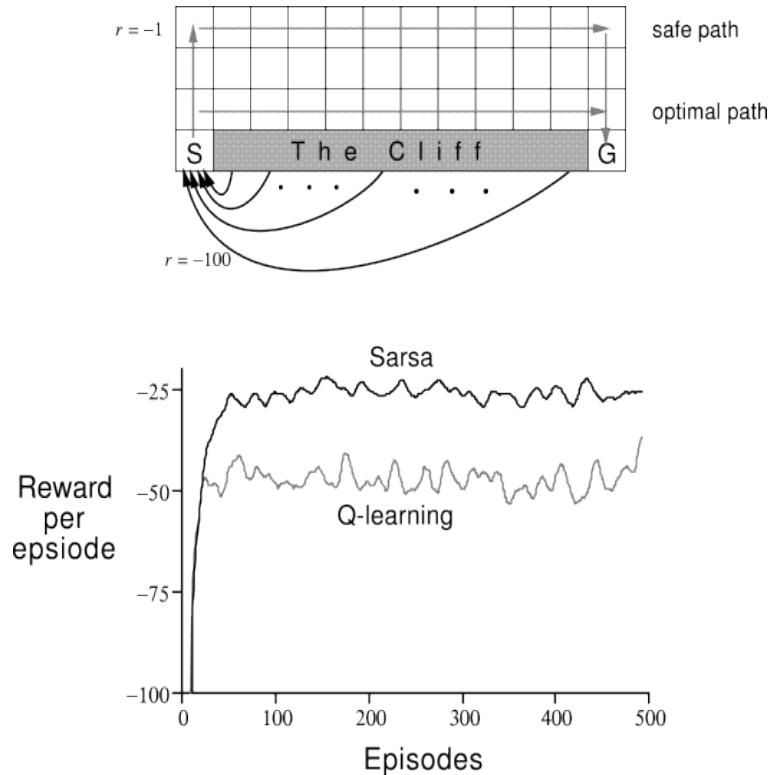


Figure 2.1: Q-Learning vs Sarsa[10]

2.2 Neural Networks and Convolutional Neural Networks

In this chapter we will discuss how networks works, the principles behind them and what optimization techniques we can use. It is worth mentioning that the information is splitted in data, model, loss functions, train and test.

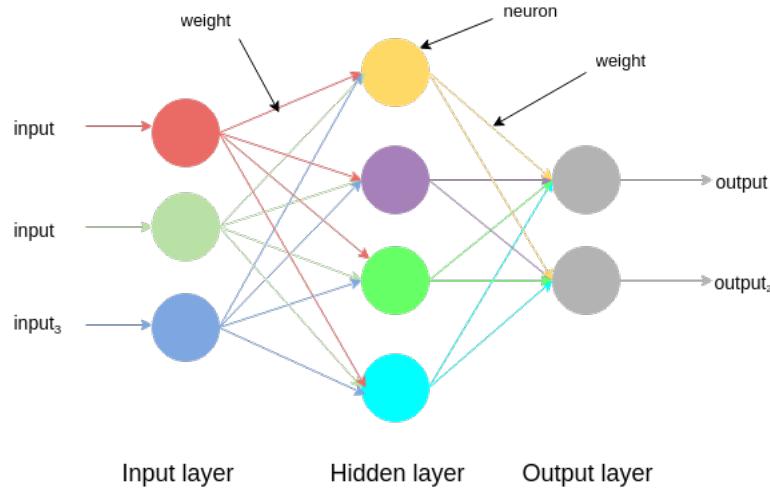


Figure 2.2: example of neural network

2.2.1 Data

The data resulted from pre-processing step is crucial for learning. To differentiate between noisy images and/or bad lighting is a big problem for a network because it may learn the inappropriate features.

One of the first concerns is choosing the color space. We can have RGB(Red Green Blue) or we can have YUV(luminance blue–luminance red–luminance). In many cases RGB is the first choice instead of YUV[11]. If we are interested in color both of them are a good choice, but if we are interested in edges the luminance channel it will be more suitable.



Figure 2.3: Lena: RGB and Y, U, V channels

If values have different ranges of scales, then logarithmic normalization should be used. In distance-based classification, if we have a feature with values between [0, 1] and another feature with values between [0, 2000] we need to normalize the data between [0, 1] or [-1, 1] such that every variation of each feature counts the same as the other feature. We do not want to start with giving more importance to a feature than the others.

Another aspect that should be considered is the importance of illumination conditions. If we are not interested than we remove the mean-value for each samples. If we have thousand of pictures with different illumination condition we remove the average illumination by extracting the mean-value such that all the pictures have the same contrast.

Splitting data is another importan aspect in training networks. We have to have three sets: train sets, test sets and validation sets. 50% of initial data goes to training and the rest fifteen percent is splitted in 25% for testing and 25% for validation. This is used when the dataset is very large. Having a small dataset implies the statretgy to split data into $\frac{2}{3}$ for training and $\frac{1}{3}$ for testing[12].

Cross validation[13] should also be also taken into consideration. Cross validation is the algorithm of splitting data into test dataset and train dataset forcing both sets cross-over in succesive epochs such that every part in the initial dataset validates the model of the neural network. Assume that we have splitted data into $\{\mathbf{A}, \mathbf{B}, \mathbf{C}\}$, each one being $\frac{1}{3}$ of initial dataset. In the first iteration we take the model to train on \mathbf{A} and \mathbf{B} and test it with \mathbf{C} . The second iteration implies the training on \mathbf{A} and \mathbf{C} and testing on \mathbf{B} and the last iteration implies training on \mathbf{B} and \mathbf{C} and testing on \mathbf{A} . In this way we can assure that our model is working well for every case.

2.2.2 Model

Nonlinear activation functions[14, 15, 16]

Activation functions are used to define a connection between input and output.

Hyperbolic tangent function(\tanh)

- outputs values between 0 and 1
- outputs are zero-centred
- if extremely large negative numbers are given as input, the output will be close to -1 and the error will propagate correctly
- generally used in hidden layers
- \tanh layers learn fasten than sigmoid layers because of the size of the gradient

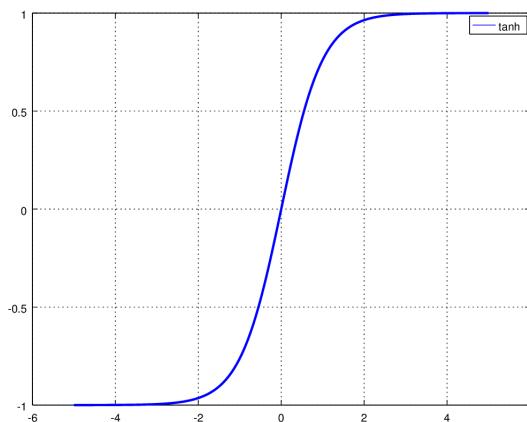


Figure 2.4: \tanh function

Sigmoid function(sigmoid)

- learning values far away from threshold is difficult
- useful for backpropagation algorithms
- easy for calculating derivatives and used in gradient descent methods
- output values between 0 and 1
- can be used in binary classification problems with setting some threshold
- if extremely large negative numbers are given as input, the output will be close to zero and the error will not propagate correctly

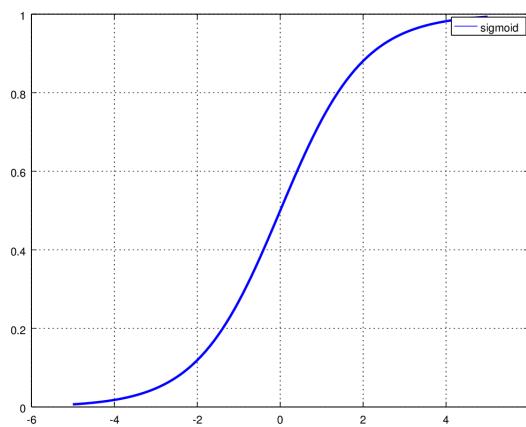


Figure 2.5: sigmoid function

Rectified Linear Units(ReLU)

- better results in networks combined with stochastic gradient descent than tanh/sigmoid[17]
- ReLU is determined to be better than tanh/sigmoid empirically

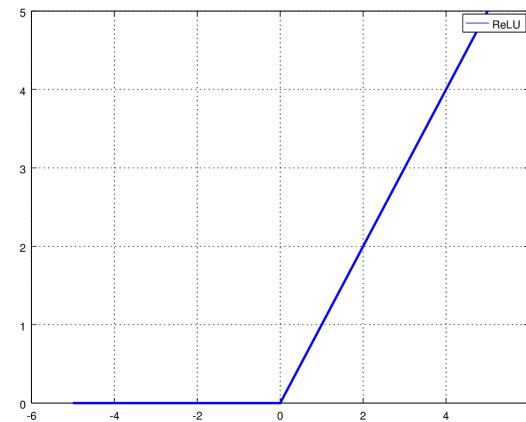


Figure 2.6: ReLU

Convolutional layers

Pooling layers can be used in getting rid of unwanted information and also can be useful in dimensionality reduction.

Spatial Convolution

- used to modify the frequency of neighbourhood pixels
- accentuates edges in combination with Gabor filters
- can be used to obtain gaussian blur effect
- suppose we have an image of **width x height** size, if we apply a convolutional kernel of size **kw x kh** the size of the output will become **(width - kw + 1) x (height - kh + 1)**



Figure 2.7: Lena: Spatial Convolution(2D) with 12 output channels, 5x5 kernel matrix and 2x2 stride

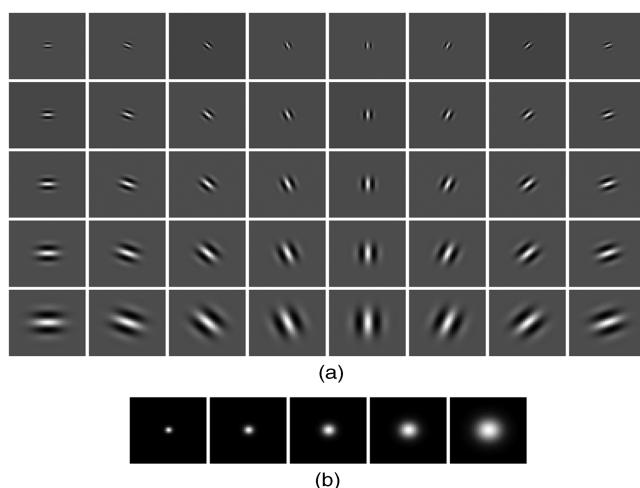


Figure 2.8: Gabor filters[18]

Average pooling

- useful for dimensionality reduction
- acts poorly in some situations as opposed to max-pooling

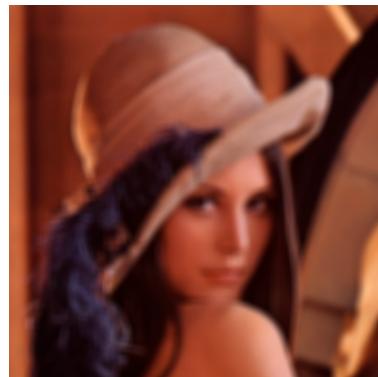


Figure 2.9: Lena: Average pooling with 10x10 kernel matrix and 2x2 stride

Max pooling

- useful for dimensionality reduction
- useful for activating features that have a low probability of being activated[19]
- if a feature appears in a cell it is less probably to appear in the neighbourhood cells

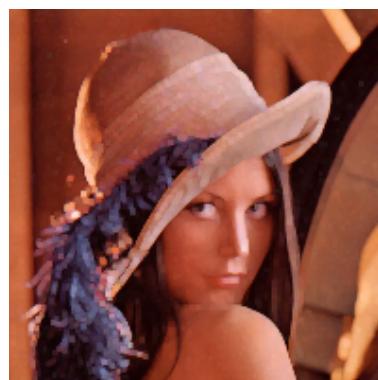


Figure 2.10: Lena: Max pooling with 5x5 kernel matrix and 2x2 stride

LP pooling

- the norm $P=1$ represents gaussian averaging and $P=\infty$ represents max pooling
- computes the information from neighbourhood if $P>1$

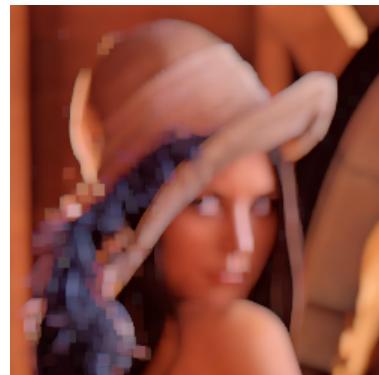


Figure 2.11: Lena: LP pooling with $\text{pnorm} = 10$, 15×15 kernel matrix and 2.5×2.5 stride

Spatial Subtractive Normalization

- the norm $P=1$ represents gaussian averaging and $P=\infty$ represents max pooling
- computes the information from neighbourhood if $P>1$



Figure 2.12: Lena: Spatial Subtractive Normalization pooling with kernel size 11×1

Feature visualization

The technique called Deconvolutional Networks takes a neural network already trained and maps the features to pixels such that information from features learned can be human “readable”.

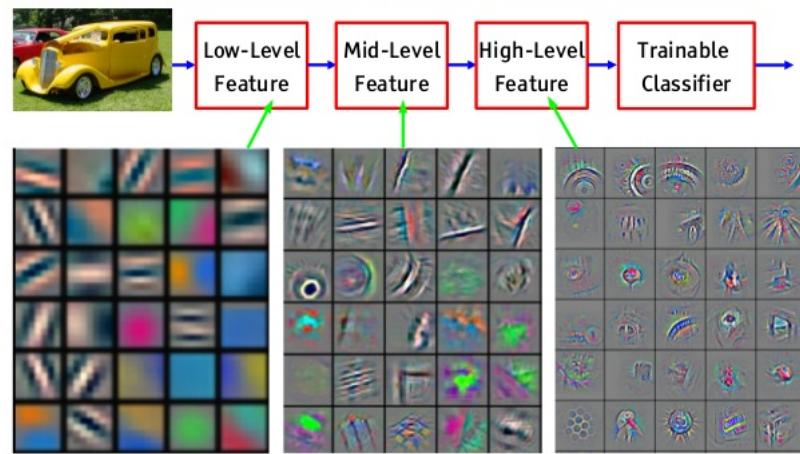


Figure 2.13: Feature visualization[20]

2.2.3 Loss functions

Loss functions are used to approximate how far we are from the solution, the difference between observed values and predicted values.

Mean squared error

- useful for regression
- not used in classification problems

$$J(\text{weights}) = \frac{1}{2 \cdot no_{samples}} \cdot \sum_{i=1}^{no_{samples}} \cdot (nn_{weights}(\text{samples}^{(i)}) - \text{labels}^{(i)})^2 \quad (2.1)$$

Hinge loss (margin criterion)

- useful for binary classifications
- forces labels to be -1 or 1

$$J(\text{weights}) = \sum_{i=1}^{no_{samples}} \max(0, 1 - \text{labels}^{(i)} \cdot nn_{weights}(\text{samples}^{(i)})) \quad (2.2)$$

2.2.4 Optimizations for learning

Optimization algorithm are used in updating the weights in a neural network in accordance with a loss function. Assuming that our cost function is defined as follows[21]:

$$J(\text{weights}_n) = \frac{1}{2no_{samples}} \cdot \sum_{i=1}^{no_{samples}} (nn_{weights}(\text{samples}^{(i)}) - \text{labels}^{(i)})^2 \quad (2.3)$$

we state that derivative of \mathbf{J} w.r.t **weights** is[21]:

$$\frac{\partial J}{\partial \text{weights}} = \frac{1}{no_{samples}} \cdot \sum_{i=1}^{no_{samples}} (nn_{weights}(\text{samples}^{(i)}) - \text{labels}^{(i)}) \quad (2.4)$$

Updating weights[21] is done in accordance with the derivative of cost function.

$$\text{weights}_n = \text{weights}_n - \alpha \cdot \frac{\partial J}{\partial \text{weights}} \quad (2.5)$$

where α represents learning rate.

Gradient descent (GD)

- if α is too small, GD converges slow
- if α is too large, GD can converge to local minimum and not global minimum
- recommended for huge number of samples
- require more iterations than SGD

Algorithm 1 Gradient Descent[21]

```

1: repeat
2:   for  $j = 1, \text{no\_weights}$  do
3:

$$weights_j = weights_j - \alpha \sum_{i=1}^{no\_samples} (nn_{weights}(samples^{(i)}) - labels^{(i)}) \cdot samples_j^{(i)} \quad (2.6)$$

4:   end for
5: until convergence is obtained

```

Stochastic gradient descent (SGD)

- very fast if number of samples is small
- iterations are removed by computing $(X^T \cdot X)^{-1}$
- requires regularization
- needs feature scaling

Algorithm 2 Stochastic Gradient Descent[21]

```

1: repeat
2:   shuffle dataset (randomly)
3:   for  $i = 1, \text{no\_samples}$  do
4:     for  $j = 1, \text{no\_weights}$  do
5:

$$weights_j = weights_j - \alpha \cdot (nn_{weights}(samples^{(i)}) - labels^{(i)}) \cdot samples_j^{(i)} \quad (2.7)$$

6:     end for
7:   end for
8: until convergence is obtained

```

2.2.5 Training and testing

The question now arises: when do we have to stop with training if a (convolutional) neural network? We have to take care of how much we train a network. If we train too much, it will learn the samples and will predict exactly the same values, but there will be a large error for samples it has not seen. If we train too little, the network will not learn how to predict the samples it has seen and will also have a large test error. Mathematically, the moment when we should stop with training is the moment when the test error will increase its value. In 2.14 we can see the optimum moment when we should stop training.

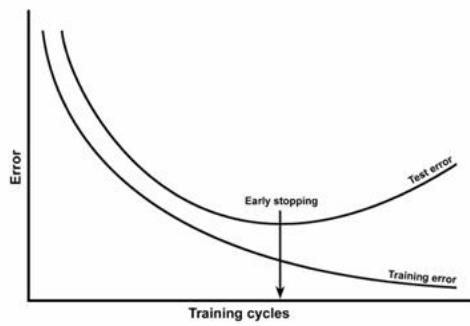


Figure 2.14: train-test error[22]

Underfitting means also high bias and represents the incapacity of the algorithm to recognize the pattern features in samples.

Overfitting means also high variance and represent the incapacity of the algorithm to adapt to new samples.

Both of them are illustrated in the figure 2.15.

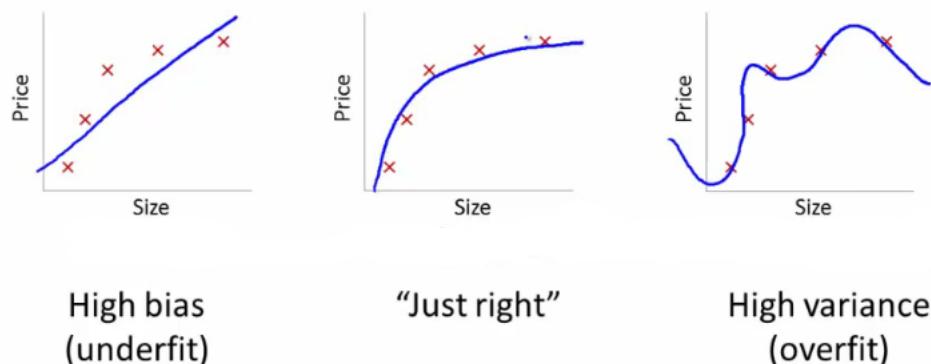


Figure 2.15: underfitting, bias-variance trade-off and overfitting[21]

2.3 An in-depth look to “Human-level control through deep reinforcement learning” paper

This section is dedicated to analyzing the paper in which is proposed the method to solve playing games using deep learning and reinforcement learning. The proposed an agent, called Q-Network capable of playing 43 different games and adapting to each one without being modified.



Figure 2.16: Snapshots with five Atari 2600 Games: Pong, Breakout, Space Invaders, Seaquest, Beam Rider[23]

2.3.1 Architecture

The article[23] proposes a new type of network, Q-network which combines convolutional layers, a representation of human receptive fields and pooling layers used for dimensionality reduction. The input layer is represented by $84 \times 84 \times 4$ neurons for the image produced after preprocessing frames from the game and is followed by three convolutional layers(32 filters of 8×8 and stride 4 with ReLU, 64 filters of 4×4 and stride 2 with ReLU, 64 filters of 3×3 and stride 1 with ReLU), two fully connected layers: a hidden layer with 512 units and the output layer which has a number of neurons equal with the number of actions corresponding to each game. Each hidden layer is followed by a rectified linear unit (ReLU).

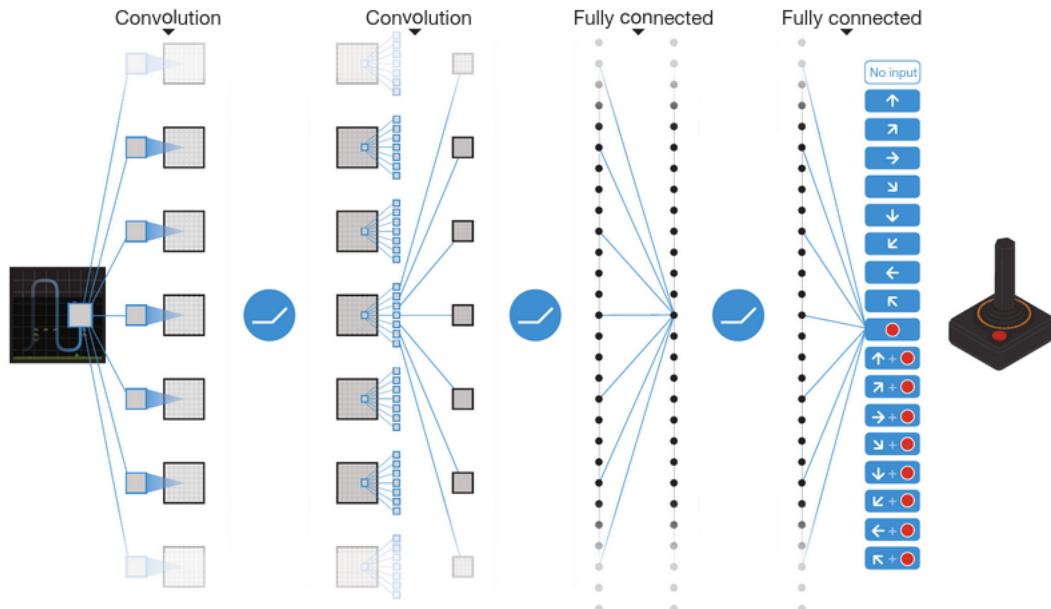


Figure 2.17: Q-Network architecture[23]

The network is used to approximate the optimal action-value function Q^* [23]:

$$Q^*(s, a) = \max_{\pi} E[r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \dots | s_t = s, a_t = a, \pi] \quad (2.8)$$

The function Q is also parameterized with the weights and the agent state $e_t = (s_t, a_t, r_t, s_{t+1})$ is saved at each iteration in a set $D_t = (e_1, e_2, \dots, e_t)$. With all the necessary information here is the computation of the loss function[23]:

$$L_i(\theta_i) = E_{(s, a, r, s')}[(r + \gamma \cdot \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i))^2] \quad (2.9)$$

where γ represents the discount factor and θ_i^- is updated with θ_i every x defined steps and here is the gradient[23]:

$$\nabla_{\theta_i} L(\theta_i) = E_{s, a, r, s^i}[(r + \gamma \cdot \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i)) \nabla_{\theta_i} Q(s, a; \theta_i)] \quad (2.10)$$

The Q-learning action-value function can be updated using $\theta_i^- = \theta_{i-1}$ following the optimal policy with probability $1 - \epsilon$ and selecting a random move with probability ϵ .

For minimizing loss function, Stochastic Gradient Descent is used in combination with experience replay technique for preventing ‘dead’ neurons and also, Q is not updated at each iteration. This avoids the instability of Q caused by nonlinearity of network.

The ϵ -greedy policy was set to 0.05 and the Q-values were scaled because of big range values from rewards.

2.3.2 Algorithm

The algorithm[23] used for training Q-Network is presented as it follows:

Algorithm 3 Q-Network

```

1: create replay memory D for storing N experiences
2: choose  $\epsilon$  between (0,1)
3: init Q model weights with  $\theta$ 
4: save  $Q^-$  model weights to  $\theta^-$ 
5: init variable episode to 0 and choose MAX_EPISODES
6: while episode < MAX_EPISODES do
7:   generate state  $s_1$  from frame:  $s_1 = image_1$ 
8:   while game not over do
9:     r = random number between (0,1)
10:    if r <  $\epsilon$  then
11:       $a_t$  = random action
12:    else
13:       $a_t = argmax_a Q(s_t, a; \theta)$ 
14:    end if
15:     $s_{t+1} = (s_t, a_t, image_{t+1})$ 
16:    D = D +  $\{(s_t, a_t, r_t, s_{t+1})\}$ 
17:    shuffle D
18:    extract  $(s_j, a_j, r_j, s_{j+1})$  from D
19:    if episode is finished at step j+1 then
20:       $y_j = r_j$ 
21:    else
22:       $y_j = r_j + \gamma \cdot \max_{a'} Q^-(s_{j+1}, a'; \theta^-)$ 
23:    end if
24:    propagate error  $(y_j - Q(s_j, a_j; \theta))^2$ 
25:    at each x steps save model weights  $\theta$  to  $\theta^-$ 
26:  end while
27: end while

```

2.3.3 Results

Q-Network is capable of learning the optimal policy. For example, in Breakout is learning to make a tunnel between block for sending the ball in the back. Receiving only the pixel from the frames, rewards and using the same network structure it is capable of playing different games. The Q-Network can achieve 75% of a score of a human test at 49 games[23].

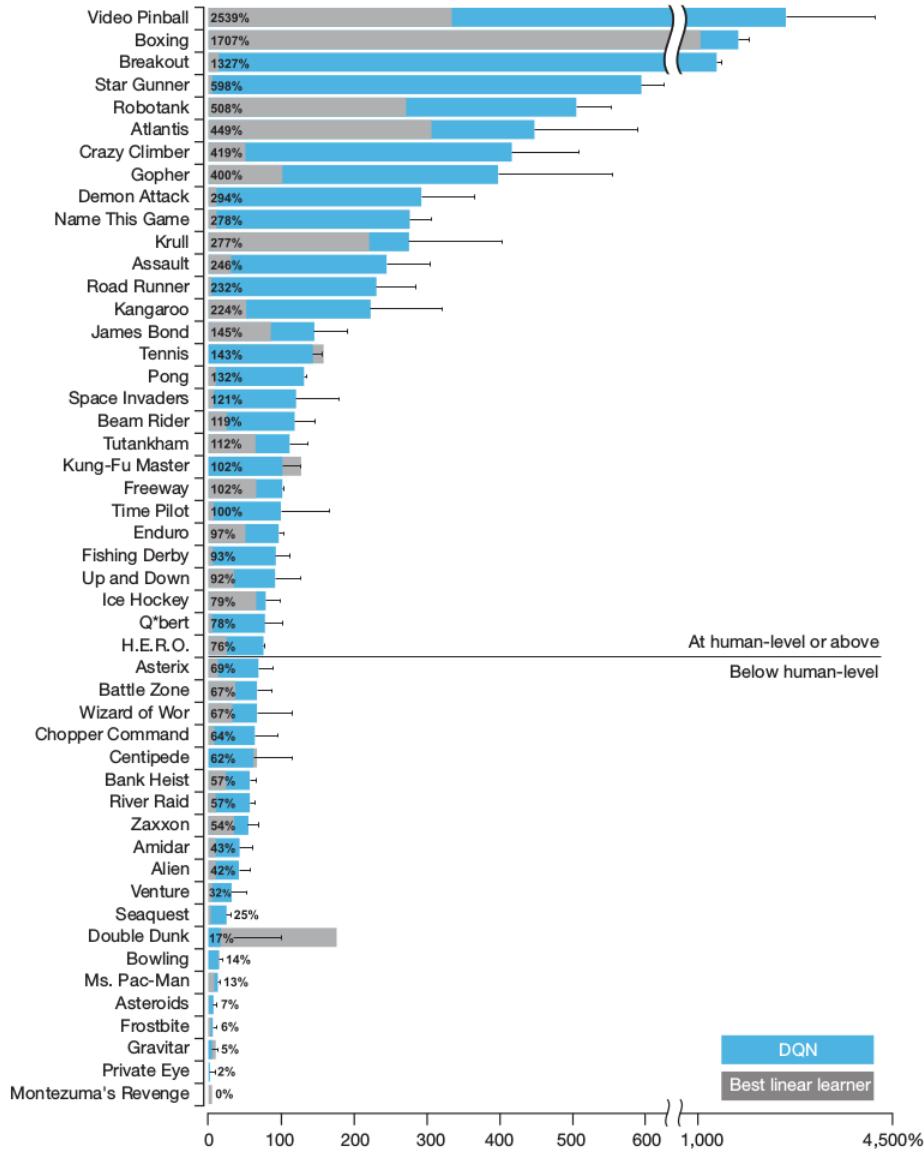


Figure 2.18: Comparision[23]

Chapter 3

System design and implementation

This chapter presents the entire work of the author. The work is splitted in three parts for the attempt to combine deep neural networks with Q-Learning. The game chosen for experimenting was Tower of Hanoi due to its simplicity and also, the static environment. Then, the values obtained by Q-Learning were used to train a deep convolutional network in combination with the generated frames. Finally, we will try to combine Q-learning algorithm with convolutional networks.

3.1 Q-Learning

In this section we try to find the optimal policy for playing Tower of Hanoi game. The algorithm start with a pool of actions initialized with UP, DOWN, LEFT, RIGHT and a table Q which the values for all (state,action) pairs will be stored.

The discount factor γ is set to 0.95, this future rewards are taking into cosideration when updating the action-value function. The learning rate α is chosen to be 0.1, thus the algorithm can converge to an optimal policy.

The exploration vs. exploitation problem is solved by using a variable ϵ which is initialized with 1 (1 is for choosing only random actions and 0 is for choosing the action that can maximize the score). For making the current policy to converge to the optimal policy there was also used a number of iterations setted to 11 and number of episodes per iteration setted to 1000. After each iteration is finished, ϵ is decreased with 0.1 until it will take the 0-value. This method was used to let the agent explore using random action in the first iteration and after exploit the values learned in the last iterations.

After all the iterations are finished, the algorithm generates a dataset. For each value stored in Q -table, it generates the frame from each coded state.

On the next page, a pseudocode of Q-Learning is presented, which is adapted after the algorithm found in Norvig's book[8]. The algorithm tries to estimate an optimal action-value function using the ϵ -greedy policy, more exactly, it chooses a random action with probability ϵ and chooses the optimum action with probability $1-\epsilon$

Algorithm 4 Q-Network

```

1: set size of Q to with no_states x no_action and initialize it
2: choose  $\epsilon$  between (0,1)
3: let  $s$  be the current state of the game
4: let actions be a pool with  $a_1, a_2, \dots, a_n$ 
5: while game not over do
6:    $r$  = random number between (0,1)
7:   if  $r < \epsilon$  then
8:      $a_t$  = random action
9:   else
10:     $a_t = argmax_a Q(s, a)$ 
11:   end if
12:    $s', r = apply\_action(a, s)$ 
13:    $Q(s,a) = Q(s,a) + \alpha \cdot (r + \gamma \cdot \max_a Q(s', a') - Q(s, a))$ 
14: end while

```

In the next figures the values resulted from Q are presented. As it can be seen the optimal policy is followed. The game allows the picker to move LEFT from first stack and to move RIGHT from last stack (e.g. If the picker points to the first stack and chooses to move LEFT, then the picker will point to the last stack). In fig. 3.1 the picker points to the second stack so the optimal action will be to move RIGHT. As we can see the value for RIGHT-action is bigger than the other values. In fig. 3.2 the agent has only one move to finish the game, in optimal conditions. Again, if we look at which value is greater than the others we can see that DOWN appears to be 100 which is exactly the value of the reward received when an episode is finished. Fig. 3.3 represents the initial state of the game, thus as it is expected the game should begin with picking for the first time a disk and that is exactly what Q-Learning is doing.

Another important aspect which should be mentioned is that the values for actions corresponding to a state s are close. Why? This happens because the game has a static environment and there is no negative reward, thus whatever happens, the agent will always loose or it will be forced to play until the game is finished. The agent can never loose.

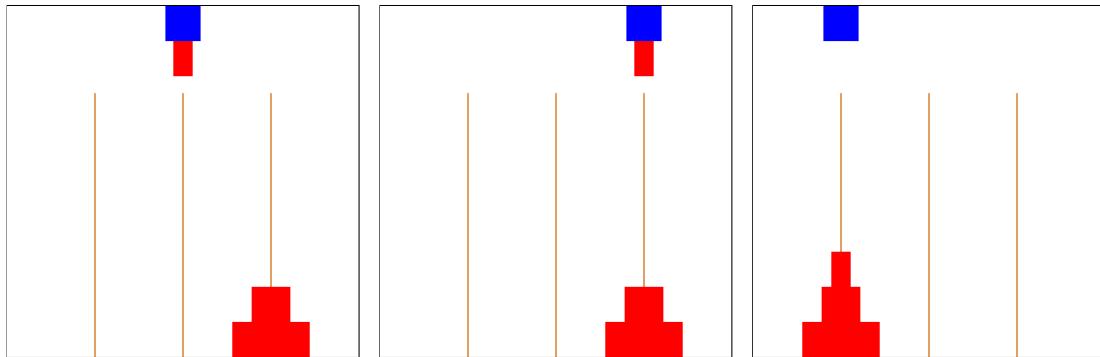


Figure 3.1:
 $UP = 90,6534$
 $DOWN = 86,8787$
 $LEFT = 89,1867$
 $RIGHT = 94,2824$

Figure 3.2:
 $UP = 97,6530$
 $DOWN = 100,0000$
 $LEFT = 93,8538$
 $RIGHT = 92,5261$

Figure 3.3:
 $UP = 26,3520$
 $DOWN = 23,8452$
 $LEFT = 23,8897$
 $RIGHT = 22,8827$

3.2 Regression with deep-learning

This chapter comes with an attempt to discuss various models of architecture that could be suitable for our problem. To see if the network fits the main theme of this paper, one must provide proof that the network model can generalize and is not hitting the “high variance problem”. In order to come with a scientific proof or at least an empirical one, several experiments have been made.

3.2.1 First attempt

The model proposed by Yann Lecun et al.[24] was slightly modified in order to map our problem. For the purpose of keeping things clear and well-separated every discussion about a network should be splitted in the next subjects: preprocessing data, chosen model for the network, loss function, training and testing.

For the first attempt, a model used in face detection and pose estimation has been chosen because good results in learning features relevant for object recognition were promised. Also, this model is capable to be applied over large images.

Data

In the section 3.1, we generated a dataset of 160 images with 4 labels for each one (Tower of Hanoi has 160 states and offers 4 possible actions). In this section we try to build a convolutional neural network capable of predicting those values from images. This is made for the sake of bringing a strong argumentation that our network model is capable of learning features from the dataset and it is not conducting to a “dead” network which is “overlearning” and can not be used on new samples.

First step was to resolve the problem of the small number of samples (160) from dataset. This has been solved by generating 64 another datasets (to achieve approximately 10,000 samples) from the original one with noise added or color changed. In this way we can ensure that the network is capable of learning features dependent of size in this case.

The next figure(3.4) represent a state of the game and the next ones(3.5) represent altered images with noise added and color changed.

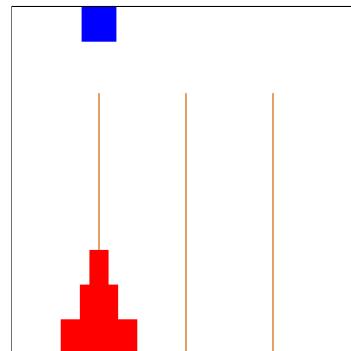


Figure 3.4: Initial state of the game “Tower of Hanoi”

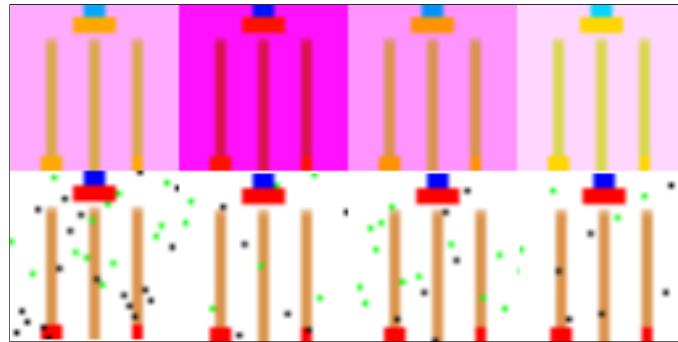


Figure 3.5: Altered dataset

For the preprocessing data various problems have been attempted. First of all, there is a large discussion on the RGB vs. YUV topic. This paper implements the YUV transformation based on the assumption that luminance should be separated from chrominance. If features dependent of color should be learned, then the neurons connected with the chrominance pixels will be activated, otherwise the neurons connected with luminance pixels will be activated. Both type of channels are kept in the attempt to make an algorithm that could generalize very well. Also, the YUV color is more closer to the human vision than RGB.

The decision to not work with 1-channels colors space like grayscale are well-founded. There are games that size should not be a feature which should be learned and in this cases we are interested in learning features dependent of color.

The next step was to normalize the images for aligning the range of pixel intensity values to a normal distribution. With other words, the histogram of the images are stretched and the contrast is improved in order to get sharp edges.

The normalization of the images implies computing the mean and the standard deviation. These values are computed for the training dataset for each channel. Then, we add the mean and divide it with standard deviation. This operation is applied on each channel on both datasets (train and test).

Another way to improve the contrast is using a gaussian matrix which follows a gaussian distribution, as it is clear from its name. The gaussian distribution state that average things should occur frequently and extreme things should occur rare. The gaussian matrix will slide all over the image and will modify the pixel values such that the pixels from the middle of the neighbourhood will be more important than the pixel values from the edge of the gaussian matrix. If the edge pixel values have a zero importance than the result image will be exactly as the initial image.

On the other hand, labels were also needed to be processed. Because of the formula for estimating the action-value function (Q) different ranges of values were generated. For example, if we have to do regression on values as 0,..., 100 there would not be a problem because the delta range is 10^2 , but the values generated were following a delta range of 10^5 . This can cause instability to the network. These being said, the paper proposes to bring the values to the same range using logarithm function and of course, to normalize them to follow the normal distribution.

Data was also shuffled to avoid having consecutive states from the game

Model

The model used for training the convolutional neural network is the model presented in the next paper: “Synergistic Face Detection and Pose Estimation with Energy-Based Models”[24] with some modification such that the model becomes suitable for our network.

The model follows another model called LeNet-5 proposed by Yann LeCun et al. which was used in document recognition[25] for training a network that could recognize noisy handwritten characters. The results obtained by the network state that the error rate is small, under 1% which is extremely satisfying. Trained on noisy handwritten characters draws attention to one important aspect, if the model is able to generalize even in heavy conditions, why should not be used in combination with Q-learning for playing games.

In the next figure(3.6) we can see how the network proposed by this article looks.

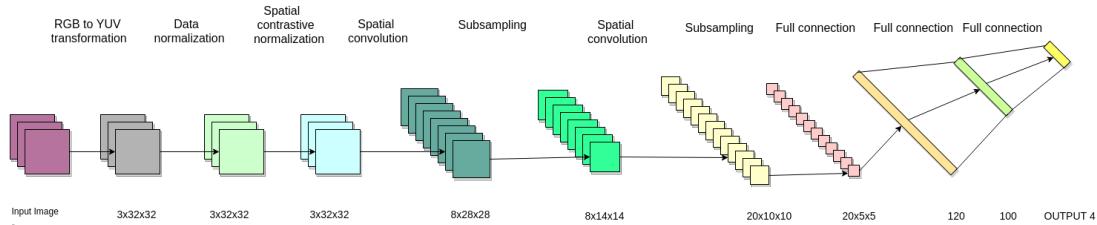


Figure 3.6: ConvNet Architecture

The network is composed of three layers. Each layer applies kernel convolutions and samples on the data. After each layer we use hyperbolic tangent as layer activation function. As discussed in the paper “Efficient Backprop” by Yann LeCun et al., the motivation for using hyperbolic tanh is simmilar to the one to normalize data. The mean of the output data tends to be close to zero, thus the it is more likely less probable to cause instabilities to the network derrivatives when used in combination with backpropagation.

After the finally layer, the sigmoid activation function was used because are labels are normalized by scaling between 0 and 1.

The input layer consists of no_features x no_pixels_width x no_pixels_height neurons and the output layer has 4 neurons because only 4 actions are needed to play Tower of Hanoi.

The last layers are fully connected and for hidden layers Spatial Convolution and Sub Sampling was used. In the appendix A.1 the implementation in Torch7 can be found.

Loss function

For computing the error, Mean Squared Error was used. Being a task of regression, and not a task of classification the MSE is most appropriate to be used.

$$\text{loss}(\text{target}, \text{output}) = \frac{1}{\text{no}_{\text{samples}}} \cdot \sum_{i=1}^{\text{no}_{\text{samples}}} \cdot ((\text{output}^{(i)}) - \text{target}^{(i)})^2 \quad (3.1)$$

Training and testing

In the last sections we discussed the composition of the model. We stated that the last layer is formed by 100 neurons. Sometimes, neural networks are not as easy as we imagine to implement. Even if we have a big dataset, or even if we altered data in a way that we think that could work for the sake of the generalizations, bad things happens. As we can see from the figures, the network is doing very good at learning samples from the training dataset. It is worth mentioning that one epoch is equivalent with learn one time the train dataset and get the error for forwarding over the network the dataset. Also, to make a parallel let's remember that the output values are normalized to 0-1 range. Now, we can return to the graph and see that the training error behaves as we expect to happen and yes, it is decreasing. However, the test error is not behaving mathematically as showed in figure 2.14. This brings into question the validity of the model. One must observe the slope of the train-line between first epoch and the second epoch is steeper, which means that after only one epoch the network overlearned the train-dataset and that is one reason to not predict well on the test-dataset. Anyway, we can state that the model with 50 neurons is an improvement as opposed to the one with 100 neurons. From this we conclude that if the model is too complex, the network could run into overfitting phenomenon.

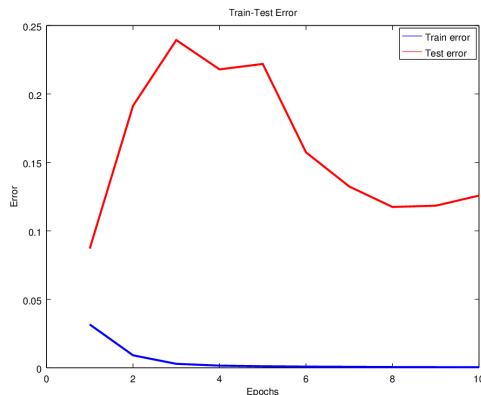


Figure 3.7: Last hidden fully connected layer has 100 neurons

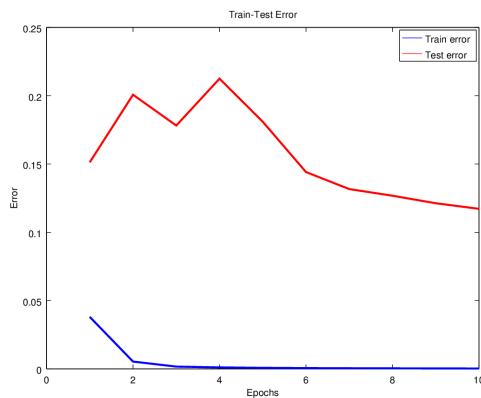


Figure 3.8: Last hidden fully connected layer has 50 neurons

Chapter 4

Results

Results

Chapter 5

Conclusions

Conclusions

Chapter 6

Future work

Future work

Appendix A

Code examples

A.1 Model

```
1 model = nn.Sequential()
2
3 model.add(nn.SpatialConvolutionMM(3, 8, 5, 5))
4 model.add(nn.Tanh())
5 model.add(nn.SpatialSubSampling(8, 2, 2, 2, 2))
6
7 model.add(nn.SpatialConvolutionMM(8, 20, 5, 5))
8 model.add(nn.Tanh())
9 model.add(nn.SpatialSubSampling(20, 2, 2, 2, 2))
10
11 model.add(nn.SpatialConvolutionMM(20, 120, 5, 5))
12 model.add(nn.Reshape(120))
13 model.add(nn.Linear(120, 100))
14 model.add(nn.Tanh())
15 model.add(nn.Linear(100, 4))
16
17 model.add(nn.Sigmoid())
```

Listing A.1: Convolutional Neural Network Model

Bibliography

- [1] DanC. Cireşan, Alessandro Giusti, LucaM. Gambardella, and Jürgen Schmidhuber. Mitosis detection in breast cancer histology images with deep neural networks. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2013*, pages 411–418. Springer Berlin Heidelberg, 2013.
- [2] Yonglong Tian, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Pedestrian detection aided by deep learning semantic tasks. *CoRR*, 2014.
- [3] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.
- [4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, 2013.
- [5] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A Neural Algorithm of Artistic Style, August 2015.
- [6] Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011.
- [7] Ivan Petrovich Pavlov. *Conditioned Reflexes: An Investigation of the Physiological Activity of the Cerebral Cortex*. Oxford University Press: Humphrey Milford, 1927.
- [8] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009.
- [9] Paul Davidsson, Brian Logan, and Keiki Takadama, editors. *Multi-Agent and Multi-Agent-Based Simulation, Joint Workshop MABS 2004, New York, NY, USA, July 19, 2004, Revised Selected Papers*, Lecture Notes in Computer Science, 2005.
- [10] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.
- [11] Pierre Sermanet. A deep learning pipeline for image understanding and acoustic modeling. Technical report, DTIC Document, 2014.
- [12] KK Dobbin and RM Simon. Optimally splitting cases for training and testing high dimensional classifiers.
- [13] Payam Refaeilzadeh, Lei Tang, and Huan Liu. *Cross Validation*. 2009.
- [14] David Kriesel. *A Brief Introduction to Neural Networks*. 2007.
- [15] Włodzisław Duch and Norbert Jankowski. Transfer functions: Hidden possibilities for better neural networks. In *9th European Symposium on Artificial Neural Networks (ESANN), Brugge 2001.*, pages 81–94, 2001.

- [16] Y. LeCun, L. Bottou, G. Orr, and K. Muller. Efficient backprop. In G. Orr and Muller K., editors, *Neural Networks: Tricks of the trade*. Springer, 1998.
- [17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.
- [18] Xudong Xie, Qionghai Dai, Kin-Man Lam, and Hongya Zhao. Efficient rotation- and scale-invariant texture classification method based on gabor wavelets. 2008.
- [19] Y-Lan Boureau, Jean Ponce, and Yann Lecun. A theoretical analysis of feature pooling in visual recognition. In *27TH INTERNATIONAL CONFERENCE ON MACHINE LEARNING, HAIFA, ISRAEL*, 2010.
- [20] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, 2013.
- [21] A. Ng. Lecture notes. CS 229: Machine learning. *Stanford University*, 2003.
- [22] Schematic of a neural network training with early stopping. <http://documentation.statsoft.com/STATISTICAHelp.aspx?path=SANN/Overview/SANNOversviewsNetworkGeneralization>.
- [23] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 2015.
- [24] Margarita Osadchy, Yann Le Cun, and Matthew L. Miller. Synergistic face detection and pose estimation with energy-based models. *J. Mach. Learn. Res.*, May 2007.
- [25] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.