Universitatea POLITEHNICA din București

Facultatea de Automatică și Calculatoare,
Departamentul de Calculatoare

LUCRARE DE DIPLOMĂ

Deep Learning în jocuri

Conducător Științific:
As. Drd. Ing. Tudor Berariu

Autor:
Florentina-Ștefania Bratiloveanu

București, 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-Ștefania 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 recieved during the game?

Contents

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 form photos taken with Google Street View[3], all using unsupervised learning of features.

This paper propese 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 ideea was brought to attention in 2013, when Alex Graves et al. from DeepMind[1] came with the ideea 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 pictues. 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 infere 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 discus 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 continous 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 thing, 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 learning 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 accorded to rewards recieved 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 Torch7[1], a deep learning framework written in Lua[2], a scripting language based on a C API. For creating the game, generating frames or modifying images LOVE platform[3] which is also written in Lua has been used. For interactive computing(image/filter visualization) iTorch[4] has been prefered.

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

Chapter 3

Chapter 4

Chapter 5

Chapter 6

1.5 Copyright infringement

[1] http://torch.ch/
[2] http://www.lua.org/about.html
[3] https://love2d.org/wiki/Main_Page
[4] https://github.com/facebook/iTorch

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 recieve 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 $\mathbf{Q(s,}a_3\mathbf{)}$ is the maxim value of $\mathbf{Q}$ in state $\mathbf{s}$ 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 2.2 shows behaviour of Q-Learning and Sarsa

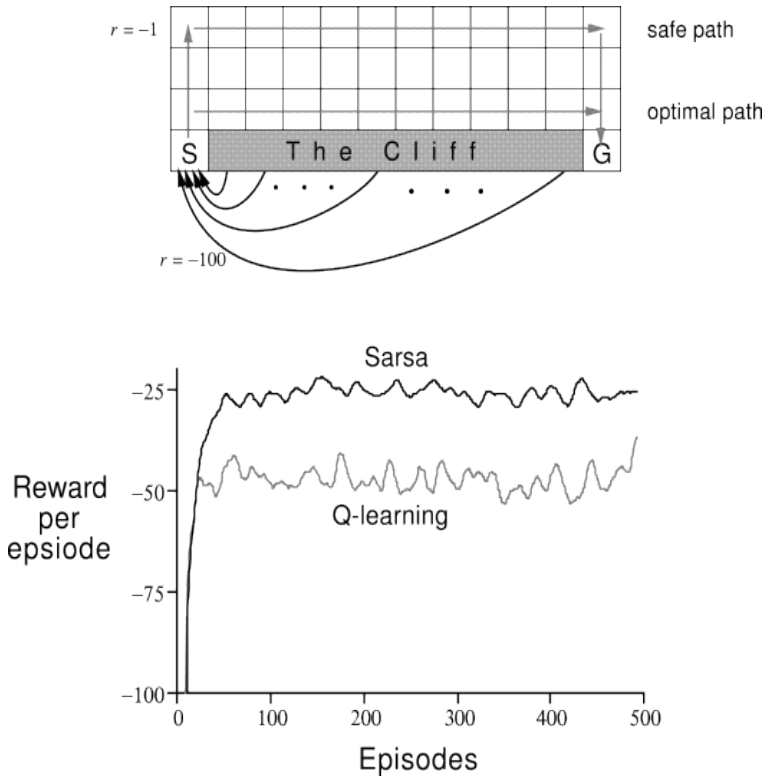

Figure 2.1: Q-Learning vs Sarsa[10]

2.2 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.

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 intrested in edges the luminance channel it will be more suitable.

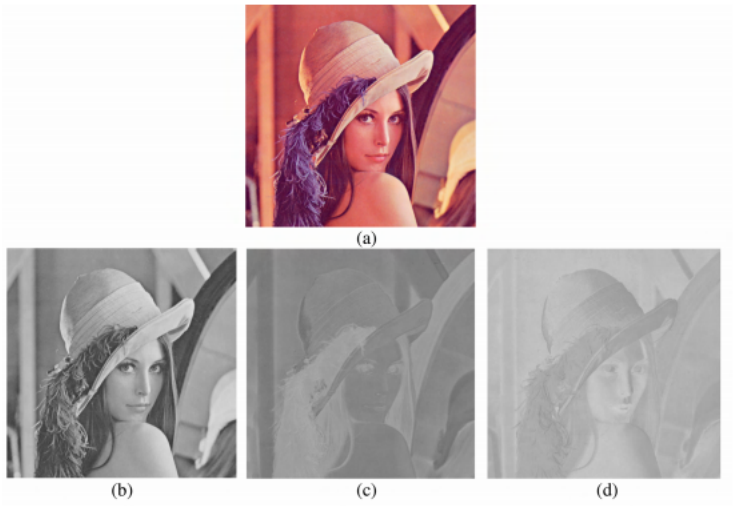

Figure 2.2: Lena: RGB and Y, U, V channels[12]

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. Fifteen percent of initial data goes to training and the rest fifteen percent is splitted in twenty five percent for testing and twenty five percent for validation. This is used when the dataset is very large. Having a small dataset implies the statretgy to split data into 2/3 for training and 1/3 for testing[13].

Cross validation[14] should also be also taken into consideration. Cross validation is the algorithm of splittng 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 {**A**, **B**, **C**}, each one being 1/3 of initial dataset. In the first iteration we take the model to train on **A** and **B** and test it with **C**. The second iteration implies the training on **A** and **C** and testing on **B** and the last iteration implies training on **B** and **C** and testing on **A**. In this way we can assure that our model is working well for every case.

2.2.2 Model

Nonlinear activation functions[15, 16, 17]

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

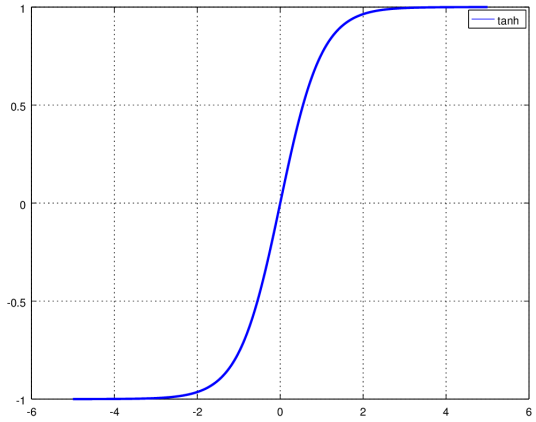

Figure 2.3: tanh function

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

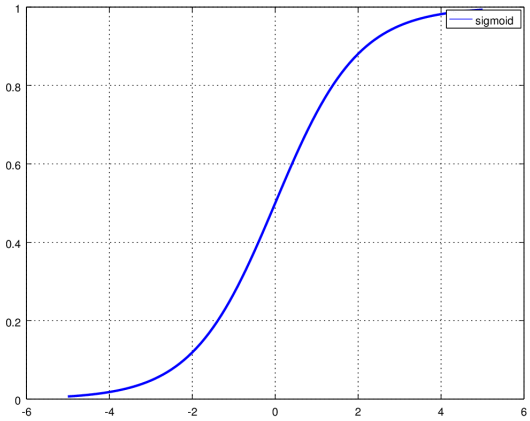

Figure 2.4: sigmoid function

Rectified Linear Units(ReLU)

- better results in networks combined with stochastic gradient descent than tanh/sigmoid[18]
- ReLU is determined to be better than tanh/sigmoid empirical

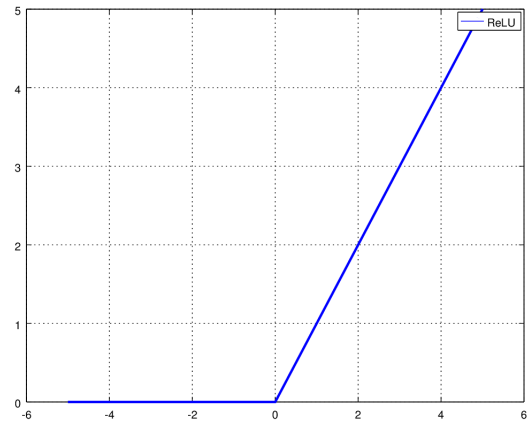

Figure 2.5: ReLU

2.2.3 Loss functions

2.2.4 Train and testing

Chapter 3

System design and implementation

Sys

Chapter 4

Results

Results

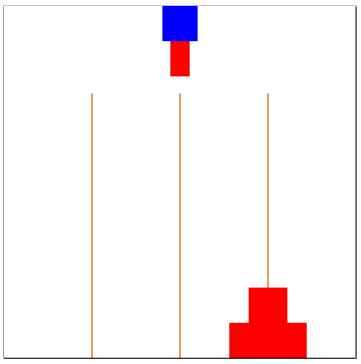

Figure 4.1:
A really Awesome Image

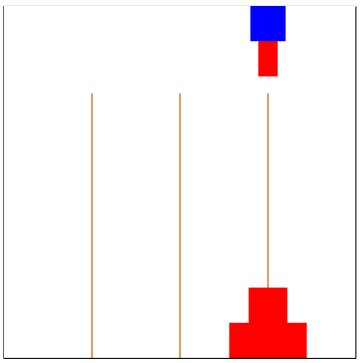

Figure 4.2:
97,6530
100,0000
93,8538
92,5261

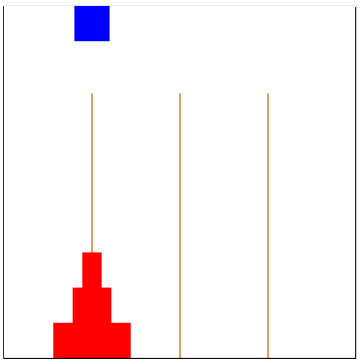

Figure 4.3:
97,6530
100,0000
93,8538
92,5261

Chapter 5

Conclusions

Conclusions

Chapter 6

Future work

Future work

Appendix A

Code examples

A.1 Model

```
model = nn.Sequential()

model:add(nn.SpatialConvolutionMM(3, 8, 5, 5))
model:add(nn.Tanh())
model:add(nn.SpatialSubSampling(8, 2, 2, 2, 2))

model:add(nn.SpatialConvolutionMM(8, 20, 5, 5))
model:add(nn.Tanh())
model:add(nn.SpatialSubSampling(20, 2, 2, 2, 2))

model:add(nn.SpatialConvolutionMM(20, 120, 5, 5))
model:add(nn.Reshape(120))
model:add(nn.Linear(120, 100))
model:add(nn.Tanh())
model:add(nn.Linear(100, 4))
```

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] Yıldıray YALMAN and Ismail Ertürk. A new color image quality measure based on yuv transformation and psnr for human vision system. *Turkish Journal of Electrical Engineering and Computer Sciences*, 2013.

[13] KK Dobbin and RM Simon. Optimally splitting cases for training and testing high dimensional classifiers.

[14] Payam Refaeilzadeh, Lei Tang, and Huan Liu. *Cross Validation*. 2009.

[15] David Kriesel. *A Brief Introduction to Neural Networks*. 2007.

[16] Wlodzislaw 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.

[17] 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.

[18] 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.