# Model Free Deep Learning With Deferred Rewards For Maintenance Of Complex Systems.

***Alan DeRossett[1], Pedro V Marcal[2]** , Inc.,
[1]Boxx Health Inc.., 3538 S. Thousand Oaks Blvd., Thousand Oaks CA. 91362
[2]MPACT Corp.,5297 Oak bend Lane, Suite 105, Oak Park, CA. 91377
*Presenting and Corresponding author : alan@mb1.com

Abstract

This paper reviews progress in Deep Learning and the successful Application of Deep Q Networks to Competitive Games. The basis of the method is Watkins' Deferred Rewards Learning[1]. However its implementation in its current form is due to D. Hassabis et al [2]. In its current form the technology is heavily dependent on image processing. This suggests that the method may be adapted for the establish ment of Optimal Maintenance Policies for Complex systems such as Airliners and/or racing cars with the addition of monitoring of Sound.

Keywords : deep learning, Deferred Rewards Learning, game theory, maintenance of complex systems.

## Introduction

The authors started off by investigating the recent explosive growth of the GPU in numerical processing. We soon discovered that the Deep Learning Community provided the largest growth in using the GPU. In fact it may be said that progress in Deep Learning owes its recent progress to massive computing which was able to achieve the scale necessary to solve problems in imaging. Two programs that enabled this was Theano [3] from the University of Montreal and the recent open source system from Google, Tensor Flow [4]. Theano may be looked upon as a system for code optimization and deployment on GPUs. The program was developed and used for neural network problems. The Tensor Flow program achieves the same means by allowing itd users to make use of a data flow model. Once users cast their algorithms in data flow format. The program will allow the user to bring all the available computing power to bear on the data flow object to be completed. The goal of massive parallelization has been elusive. For a long time the people in this audience have tried to apply it to FEM for example. The existence of multiple core machines have sped up the solution of our problems. However the Artificial Neural Network Community has shown us that concurrency is much simpler when problems are increased by two orders of magnitude. It is interesting to speculate that the FEM community may be able to take advantage of the two programs for FEA. In order to achieve our objective of explaining the technology behind the success of DQN [2] and its spectacular achievement of beating the world Go Champion, we will start introduce the three technologies behind it. We should also note that one of us (ADR) has spent considerable time and effort downloading and installing Theano and Tensor Flow on local computers as well as on the Cloud.

## Theoretical Considerations.

### Deep Learning

Nielsen [5] has pre-released the first Chapter of a book in preparation by Bengio et al [6] that explains the theory of deep learning applies it to decyphering handwritten numbers The exposition is particularly instructive because of the demonstration of the theory in the form of a Python computer program.

Following [5] we start by defining a sigmoid network.

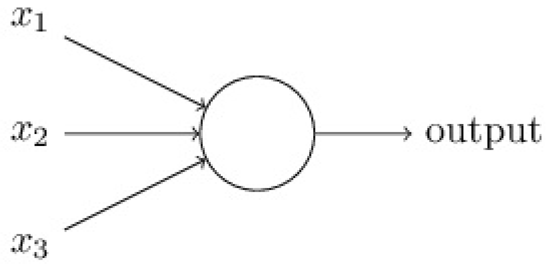

Fig. 1 Sigmoid neuron

The neuron has 3 inputs x and a bias b (scalar) with weights w.
The input to the neuron is w.x+ b. The output is modified by the sigmoid or logistic function written as,

$$\frac{1}{1+\exp(-\sum_j w_j x_j - b)}.$$

Fig.2 Sigmoid function

The sigmoid function σ (z) has the desirable property of small input changes resulting in small and smooth changes in output.
In deep networks, we introduce additional layers between the input and output.

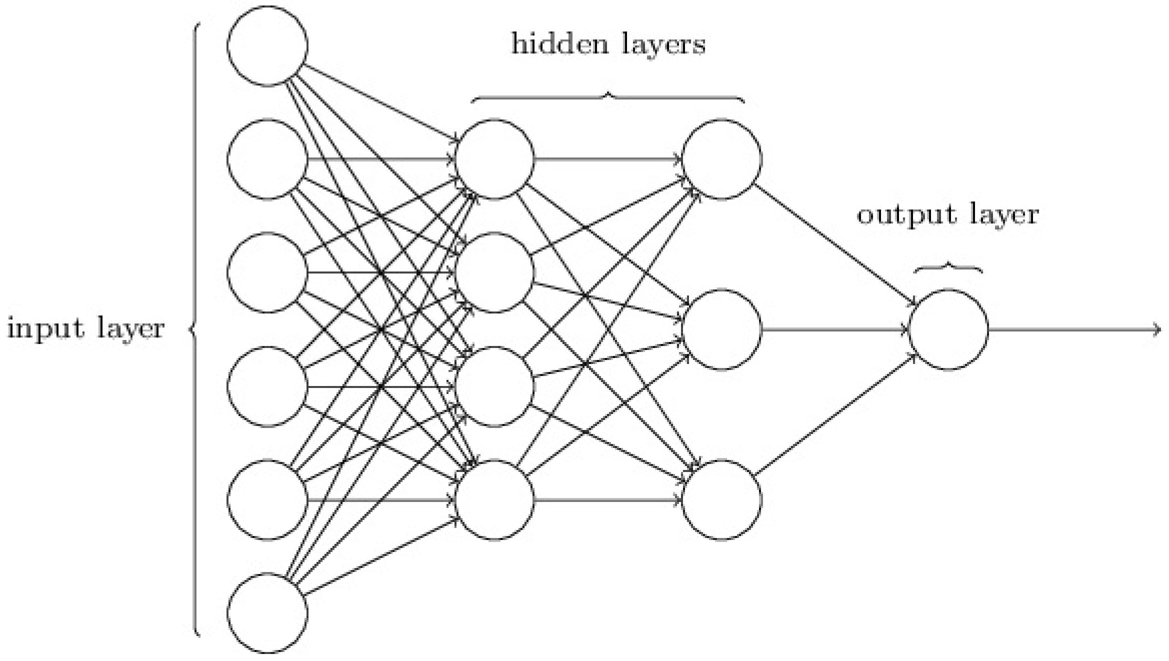

Fig. 3 Deep network with hidden layers (2).
We introduce an additional index to indicate the layer, then the input to a neuron in a hidden layer is $z_{ij} = \Sigma\ x_{ij}.w_{ij} + b_{ij}$ and its output is $\sigma_{ij}(z)$ for j= 0 to d layers where d is the output layer.

Hence an input $x_{i0}$ results in a nonlinearly mapped output $\sigma_{id}(z)$.
The network is defined by a number of training inputs x with n outputs y(x).
We define a cost function $C(w,b)= 1/2\, n\, .\ \Sigma\ (y_{id}-\sigma_{id}(z))^2$
then the objective is to vary the weights w and biases b to reduce the value of C for all the members of the training set. So we see the training problem as the minimization of the quadratic function C. The numerical procedure for solving such problems is well known. However the most used method is that of back propagation of errors. As mentioned earlier, we are at a stage where the combined power of software and hardware can be applied to large problems with many layers (typically up to 10). I think of this process as building a large interpolation function so that any other input x will result in an output that conforms to the output function defined by the input output set x, y respectively. This concludes our brief discussion of deep learning.

## Learning With Delayed Rewards

In his thesis[1], Watkins investigated animal behavior in both the wild and under lab controlled condition. There is a rich set of experiences and theories. One may view an animal's reaction as an intelligent response to a set of stimuli as an immediate as well as a long term reaction with a view to survival, (the ultimate response). Watkins framed the problem as a number of variables $x_i$ with state $a_{it}$ at time t. At any stage the problem was assumed to be a Markov process. The problem is framed in turns of a cycle or epoch in terms of which all the states are changed in sequence according to some set policy. Initially this could even be a random one. The time step within the epoch is assumed to take n steps. At a step t we assume that some action $Q(a_{it})$ results in some reward $r_{it}$ at every time step but depreciated by a factor $\gamma$. Because $\gamma < 1$ the return tends to 0 with n the number of steps being large. Hence we have the n-step truncated return given by a change in $a_i$ at time t.

$$r_t^{[n]} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{n-1} r_{t+n-1}$$

The rewards for actions $Q(a_{it})$ over n steps is given by the shifted n-step rewards

$$R_n = \Sigma\ r_{it}^{[n]t}$$

At this stage the framing of the problem has introduced a further number of unknowns. We note that the state a changes with every action Q. The reward function r is not known and the policy for selecting Q is also not known.The problem is however given specificity by selecting the changes $Q(a_{it})$ by the principle of dynamic programming(DP).[7]. Watkins showed that with DP the action Q can be achieved iteratively and Watkins and Dayan[8] gave further proof governing the iterative procedure. Watkins also assumed that the reward function could also be defined by repeated observation of the actual game over time. The thesis did not go into the application of the theory. One must assume that a large amount of numerical calculations must have been performed for Watkins to be able to speak so authoritatively on the problem. The reader is referred to [9] to see a tutorial on learning with deferred rewards.

## Theory of Games with deep learning and DQN.

It was up to Hassabis and his colleagues at Deep Mind[2] to bring substance to Watkins' methods. Hassabis reformulated the problem in neural network terms. This was demonstrated by applying it to a whole set of Atari Games. In many ways the Atari games were the perfect form as a project. The games already had a scoring system that provided the Reward Function R and the games were built for a user to specify an action Q at any stage. The pixels on the screen were used as input. They were turned into values by applying a convolution mapping to the screen. The input state $a_{i0}$ were defined by the screen image. The Q function was defined as a square matrix giving all the possible combination of the changes in the state a sequence with time. The preprocessing for input to the neural net is shown in Fig. 6. The Q functions were given by the coding on the right while the conversion to convolutions were obtained on the left side. Both results were fed into the neural network.

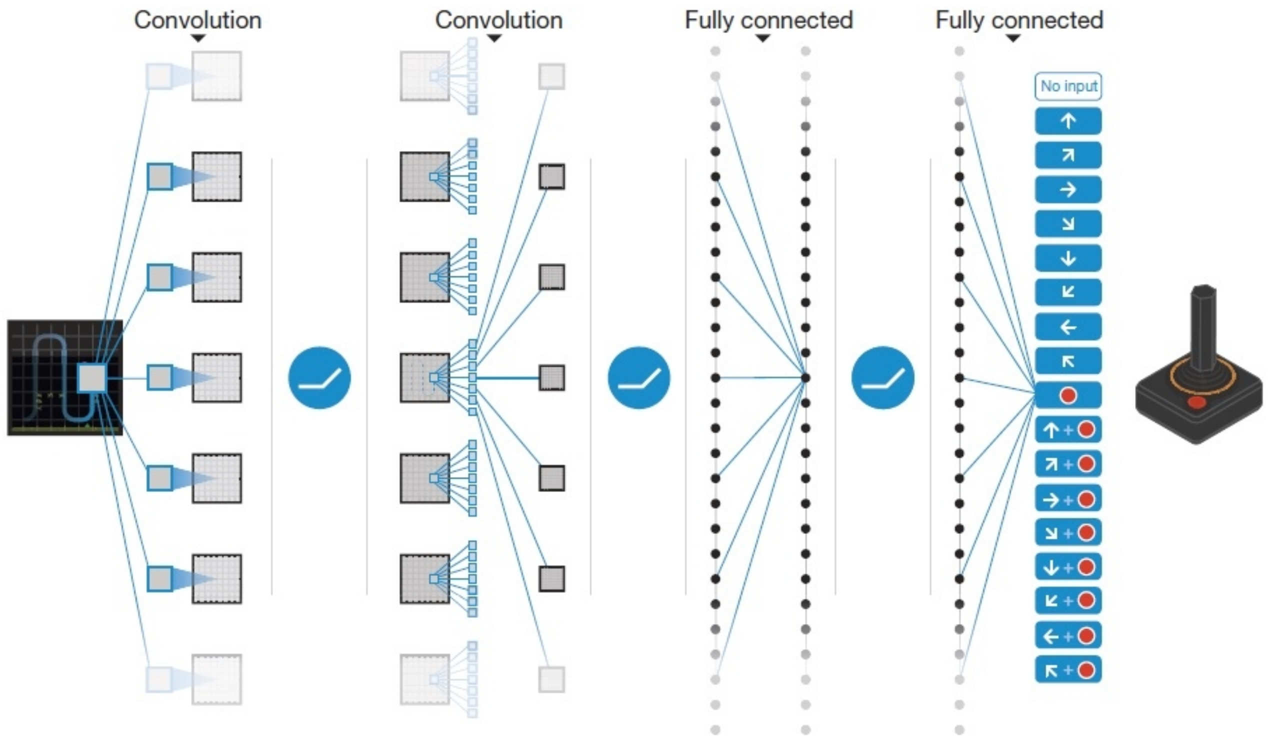

Fig. 6 Schematic for preprocessing raw input for input to the neural net.
The project proceeded in two phases. In the first training phase the program was set to collect a massive amount of data resulting from changes in the Q functions for different starting state values a. The program stores all the experiences in a database $D_e$ where e is the total sum of  recordings for training a particular game. The training of DQN networks is known to be unstable. In the second phase the database D was used to train the neural net in what is known as a experience replay developed in [2] and using a biologically inspired mechanism that randomizes over the data, thereby removing correlations in the observation sequence and smoothing over changes in the data distribution. The second improvement used an iterative update that adjusts Q towards targets that are only periodically updated.
Results
Here we show the results for the Space Invader game using the two useful metrics of average score and averagepredicted action-value Q.

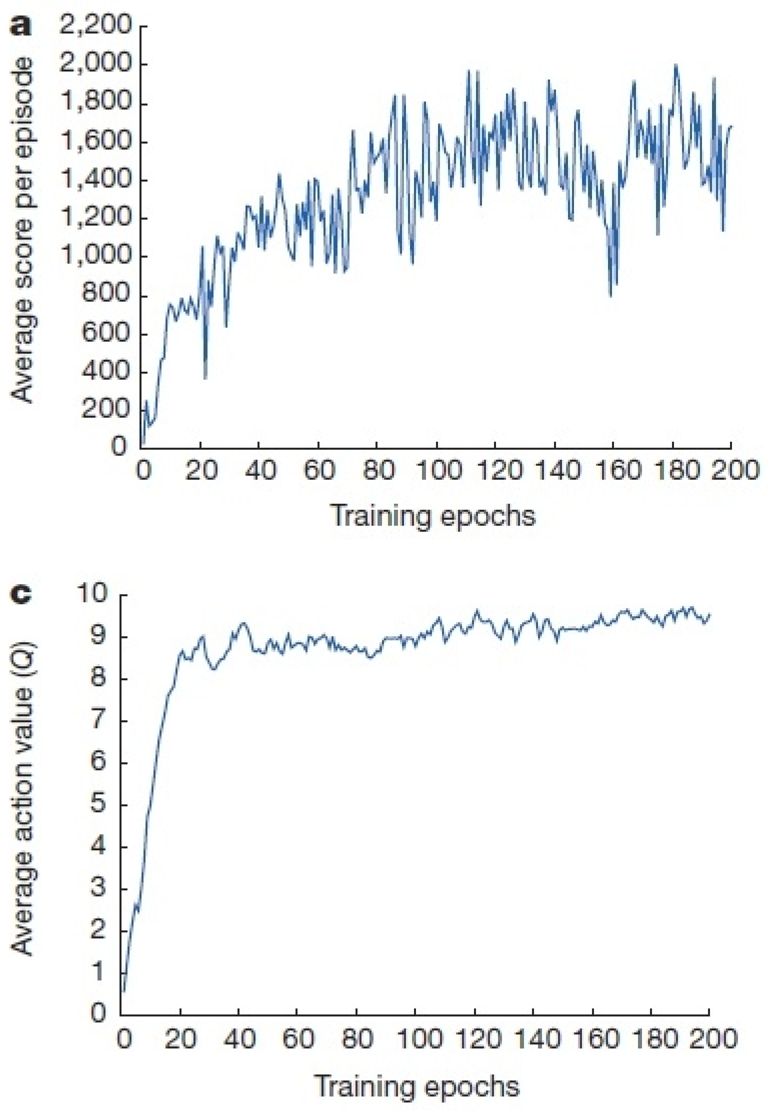

Fig. 7 Scores for space invader.

We conclude by noting that the results for all the Atari Games beat the results obtained by specialized computer playing games (tailored to one game).Hence the project even at this first step, proved the power of the method for implementing the DQN as a deep learning network.

Possible Applications in Complex Engineering Systems.
Such a general approach to applying DQN to Model Free problems has many potential applications. So we should pick the more important problems. One such problem that requires little alteration in the Lua Open Source program provided by the authors could be in the study of the maintenance problem in Airlines. Currently the fleet is overhauled and serviced on a regular basis. At such times parts are examined and sometimes replaced. The maintenance actions have an impact on the performance of an aircraft. Perhaps this improvement can be detected by video cameras that record the visual performanc of the plane during taxiing and parking.  Such records could replace those captured for the Atari Games. In another similar vein, we could probably also record the roar of the racing car engines and add this to figure out the best maintenance policy.

References

[1] CJCH Watkins, 'Learning From Delayed Rewards', Ph. D. Thesis, King's College, Cambridge, 1989
[2] Google Deep Mind[1]
Volodymyr Mnih[1]*, Koray Kavukcuoglu[1]*, David Silver[1]*, Andrei A. Rusu[1], Joel Veness[1], Marc G. Bellemare[1], Alex Graves[1], Martin Riedmiller[1], Andreas K. Fidjeland[1], Georg Ostrovski[1], Stig Petersen[1], Charles Beattie[1], Amir Sadik[1], Ioannis Antonoglou[1], Helen King[1], Dharshan Kumaran[1], Daan Wierstra[1], Shane Legg[1] & Demis Hassabis[1]
'Human-level Control Through Deep Reinforcement Learning', Nature, Vol 518, February, 2015
[3]. I. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley and Y. Bengio. "Theano: A CPU and GPU Math Expression Compiler". *Proceedings of the Python for Scientific Computing Conference (SciPy) 2010. June 30 - July 3, Austin, TX* .
[4] Google Brain Team,'Tensor Flow', Open source, 2016.
[5] Michael Nielsen,'Deep Learning', Chapter 1. of [6], January, 2016
[6] Y. Bengio, I. Goodfellow, A. Courville, 'Deep Learning', MIT Press, (in Press), 2016.
[7] R.E. Bellman and S.E. Dreyfus, 'Appied Dynamic Programming', Rand Corp., 1962.
[8] CJCH. Watkins and P. Dayan, 'Q-Learning', Machine Learning, 8, 272-279, 1992.
[9] 'A Painless Q Learning Tutorial',http://mnemstudio.org/path-finding-q-learning-tutorial.htm, 2016