Deep Learning Methods for Reinforcement Learning

Daniel Luis Simões Marta

***Abstract*— This thesis focuses on the challenge of decoupling state perception and function approximation when applying *Deep Learning Methods* within *Reinforcement Learning*. As a starting point, high-dimensional states were considered, being this the fundamental limitation when applying *Reinforcement Learning* to real world tasks. Addressing the *Curse of Dimensionality* issue, we propose to reduce the dimensionality of data in order to obtain succinct codes (internal representations of the environment), to be used as alternative states in a *Reinforcement Learning* framework.**

Different approaches were made along the last few decades, including *Kernel Machines* with *Hand-crafted features*, where the choice of appropriate filters for each particular task consumed a considerable amount of research. In this work, various *Deep Learning* methods with unsupervised learning mechanisms were considered.

Another key thematic relates to estimating Q-values for large state-spaces, where tabular approaches are no longer feasible. As a mean to perform Q-function approximation, we search for supervised learning methods within *Deep Learning*.

The objectives of this thesis include a detailed exploration and understanding of the proposed methods with the implementation of a neural controller. Several simulations were performed taking into account a variety of optimization procedures and increased parameters to draw several conclusions.

Several architectures were used as a Q-value function approximation. To infer better approaches and hint for higher scale applications, a trial between two similar types of Q-networks were conducted. Implementations regarding state-of-the-art techniques were tested on classic control problems.

I. INTRODUCTION

Deep Learning arose to popularity as a field at the ImageNet Large-Scale Visual Recognition Competition (ILSVRC) of 2012 when a conv-net had victory, with a state-of-the-art top-5 error of 15.3% from 25%, against other methods, as sparse-coding [1], with the latest competitions achieving errors close to 3%.Both traffic signs classification [2], enhanced localization and detection on images used convolutional networks [3]. Probabilistic reasoning through classification had also been applied with success in automated medical diagnosis. Besides image recognition, speech recognition has seen overwhelming improvements with a drop of error rates after a decade of stagnation [4]. Since then, the complexity, scale, and accuracy of deep networks have increased and the complexity of tasks that can be solved has grown. For instance, neural networks can learn outputting chains of letters and numbers taken from images, instead of only proposing one object [5]. Another interesting neural network designed to model sequences, LSTM (Long Short-Term Memory) has arisen in popularity with the application of machine translation [6]. Creative applications of neural networks shown a neural network could potentially learn entire programs, leading to revolutionary [7] (but still in a embryonal stage) self-programming technology. In reinforcement learning, recent research included Deep Learning architectures to obtain succinct state representations in a data efficient manner with Deep Auto-Encoders [8]. Recently Google published in nature a reinforcement learning framework, where the same model was used to learn different ATARI games, achieving human performance in half of the tested games [9]. Research by Google DeepMind, using a Deep Neural Network and tree-search, allowed the creation of an algorithm capable of beating a master at the game GO [10]. Furthermore, asynchronous methods for Deep Reinforcement Learning have been used for neural controllers, namely in tasks of navigation in 3D environments with even greater success than previous methods [11]. In robotics, Deep Learning was successfully applied to solve classical control tasks, such as an inverted pole, directly from raw visual inputs [12]. End-to-End training of deep visuomotor policies using a convolutional network and several fully connected layers, from raw visual inputs to the motors at the joints, demonstrated higher performances when compared to previous methods [13]. At the end of the writing of this thesis, Deep Learning was also applied to *self-driving car* technology [14].

This extended abstract is organized as follows: Section II enumerates the reinforcement learning challenges ahead. Section III describes the framework of the problem and outlines the underlying concepts used. Section IV develops the methodologies applied for state reduction, based around several architectures, supported by simulations. The results from these are presented. Section V considers the exploration of two main Q-value approximations with several tests. Section VI contemplates a partial DQN (multilayer-network at the output layer) similar to our previous Q-networks, to be tested on two MDP classic control problems. Conclusions and ideas for future work are reported in Section VII.

II. PROBLEM STATEMENT

The application of reinforcement learning rises several challenges. Information about the environment, when considering real world or complex tasks, is often overwhelming. Implementations considering raw unprocessed data from perceptions are often infeasible, due to the massive data availability, corresponding to memory usage and processing power. A second concern resides upon entering the realm of

This work was supported by project FCT/MEC(PIDDAC) INCENTIVO/EEI/LA0009/2014

Instituto Superior Técnico, Universidade de Lisboa, Av. Rovisco Pais, 1049-001 Lisboa, Portugal.
daniel.marta@ist.utl.pt

large state-spaces. Identifying and classifying similar states and their properties, thus generalizing the environment with a sufficient model, is quintessential.

Therefore, this thesis approaches two reinforcement learning challenges: encoding raw input into meaningful states; and generalizing the environment from a proper model.

To tackle the previous, deep learning methods were explored as an instrument alongside state-of-the-art reinforcement learning considerations.

The first problem was approached by considering unsupervised learning such as auto-encoders and stacked auto-encoders to perform dimensionality reduction on high-dimensional images from a toy problem. Several comprehensive experiments were performed, adjusting the number of neurons and the optimization procedure. The aim of the former mentioned tests were to verify the impact of choosing different architectures for the specific task.

The second problem was firstly tackled with two similar neural networks, depicting two Q-function approximations. An offline method to train both networks under the same conditions was used. A battery of tests was conducted to achieve a conclusive result on better practices. Furthermore, the best Q-function architecture was used on a much larger state-space. To accomplish this, a more efficient algorithm, was employed along side better procedures. Results are final drawn within two classic control tasks.

III. DEEP LEARNING APPROACH

This thesis aims to explore a solution started with algorithms inspired in the works of recent Deep Learning applications ([8] and [15]) with efficient practices elaborated regarding the genesis of a Q-value function [9]. Deep learning will enhance reinforcement learning, solving (in some degree) the *Curse of Dimensionality* by doing unsupervised learning on input data to obtain state representations and supervised learning on a Q-value function approximation. Starting with a raw data input vector, which could be black and white pixels from an image, a music, or electromagnetic signals of any kind. In sum, information our agent receives from the world:

$$\mathbf{x}_{input} \in \mathbb{R}^D \tag{1}$$

where $\mathbb{R}^D$ is the dimension of the input vector. The analysis proceeds by extracting properties with a unsupervised learning algorithm, where we extract the components with the highest variance from the data.

Auto-Encoders are feed-forward neural networks. A Linear Factor model only specifies a parametric encoder, whereas an auto-encoder specifies a decoder and an encoder.

We will use Neural Networks with non-linear activations, *Stacked Auto-Encoders* and denote each hidden layer as a composite function of other hidden layers, by the following notation:

$$\begin{aligned} \mathbf{z} =& h(g_1(\mathbf{x}_{input})) \in \mathbb{R}^B \\ \widetilde{\mathbf{x}} =& g_2(\mathbf{z}) \in \mathbb{R}^D \end{aligned} \tag{2}$$

and space $\mathbb{R}^B$ might be in a higher dimension than $\mathbb{R}^D$ in the case of an over-complete Stacked Auto-Encoder. In this thesis an under-complete Auto-Encoder is used to compress raw data into more manageable states. This is specially important for reinforcement learning, since data efficient methods to perform a large amount of iterations and deal with learning in real-time are paramount.

Defining the problem as a Markov Decision Process, our states are vectors of features with a finite action space, and actions will be deterministic, hence:

$$s_t = \mathbf{z} \in \mathbb{R}^B \quad \pi(s_t) = a_t \quad \text{with} \quad a_t \in \mathbb{R}^A \tag{3}$$

Starting with the Bellman expectation equation, we may compute a value function Q^π for a certain policy π, thus:

$$\begin{aligned} Q^\pi(s_t,a_t) =& \mathbb{E}_{s_t}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ...|(s_t,a_t)] \\ =& \mathbb{E}_{s_{t+1}}[r_{t+1} + \gamma\ Q^\pi(s_{t+1},a_{t+1})|(s_t,a_t)] \end{aligned} \tag{4}$$

or the Bellman optimality equation, unrolling an optimal value function Q^*:

$$Q^*(s_t,a_t) = \mathbb{E}_{s_{t+1}}[r_{t+1} + \gamma \max_{a_{t+1}} Q^*(s_{t+1},a_{t+1})|s_t,a_t] \tag{5}$$

From here, one could directly search for the optimal policy, on the policy space, solving iteratively the Equation. (4), known as policy iteration. In a deterministic environment the expectation simplifies to:

$$Q_{i+1}(s_t,a_t) = r_{t+1} + \gamma\ Q_i(s_{t+1},a_{t+1}) \tag{6}$$

or directly solving Equation. (5) by value iteration:

$$Q_{i+1}(s_t,a_t) = r_{t+1} + \gamma \max_{a_{t+1}} Q_i(s_{t+1},a_{t+1}) \tag{7}$$

Since our state space might still be very large, an approximation for $Q^\pi(s_t,a_t)$ is considered. We will use a neural network, which might be refereed as a "Deep Value function", in analogy to Deep Neural networks, thus obtaining a non-linear value iteration:

$$Q^*(s_t,a_t;\boldsymbol{\theta}) \approx Q^*(s_t,a_t) \tag{8}$$

the parameters of the neural network, i.e, weights and biases are represented by $\boldsymbol{\theta}$. Finally, after stipulating a neural network to represent our value function, one needs to define an objective function (loss function) to define train. To accomplish this, tuples of experiences $(s_t,a_t,s_{t+1},r_{t+1})$ from the gathered data set (also referenced as replay memory) $\mathcal{D}$ have to be sampled. Since it is a regression problem, we will use mean square error to propagate parameter changes, on the following manner:

$$\begin{aligned} \mathcal{L}_i(\theta_i) = \mathbb{E}_{(s_t,a_t,s_{t+1},r_{t+1})\sim\mathcal{D}}[(r_{t+1} \\ + \gamma \max_{a_{t+1}} Q(s_{t+1},a_{t+1};\boldsymbol{\theta}_{i-1}) - Q(s_t,a_t;\boldsymbol{\theta}_i))^2] \end{aligned} \tag{9}$$

therefore $r + \gamma \max_{a_{t+1}} Q(s_{t+1},a_{t+1};\theta_{i-1})$ is the network target [9] (critic term) and $Q(s_t,a_t;\theta)$ is the current output of the network (actor term) after a forward-pass. Our goal is to lower the expectation of them being different after every iteration. The gradient of Eq. (9) is presented:

$$\begin{aligned} \nabla_{\theta_i}\mathcal{L}_i(\theta) = \mathbb{E}_{(s_t,a_t,s_{t+1},r_{t+1})\sim\mathcal{D}}[(r_{t+1} \\ + \gamma \max_{a_{t+1}} Q(s_{t+1},a_{t+1};\boldsymbol{\theta}_{i-1}) \\ - Q(s_t,a_t;\boldsymbol{\theta}))\nabla_{\boldsymbol{\theta}_i} Q(s_t,a_t,\boldsymbol{\theta}_i)] \end{aligned} \tag{10}$$

The objective function is then optimized end-to-end, according to an optimization procedure, using $\nabla_{\theta_i}\mathcal{L}_i(\theta_i)$.

A Neural Fitted Q-funcition [15] was elaborated and a small variation [9] considered a fully connected network of different output layer dimension (one Q-value per action to increase overall computational performance).

Both Networks (*Stacked Auto-Encoders* and *Feed-Forward Network*) are connected, although they are pre-trained and trained separately. The final architecture is tested on a mini-game.

For the construction of the Neural Fitted Q-iteration we revised [15] with a small variation inspired in [9] fully connected network (one Q-value per action to increase overall computational performance).

Both Networks (*Stacked Auto-Encoders* and *Feed-Forward Network*) are connected, although they are pre-trained and trained separately, the final architecture is tested on a mini-game.

In addition, the fully connected network regarding Q-function approximation is also tested according to [9] (same hyper-paramaters, optimization procedures, non-linearites but different dimensions) in two classic control problems III-B .

A. Purposed Deep Reinforcement Framework

In this thesis, a stacked auto-encoder was selected, as suggested in [8] and for the Q function-aproximation [15] with a small alteration inspired on [9] to lower the computational costs by a factor of the number of actions (each time we access the maximum Q-value for every state in every iteration). In sum, the pre-training of stacked auto-encoder was applied for the state reduction and appling alongside NFQ algorithm. Furthermore, a partial DQN (the fully connected structure) is presented as a less computational expensive alternative to NFQ. Nevertheless, other supervised/unsupervised methods might emerge in the forthcoming years. A general scheme Fig. 1 and an algorithm Alg. 1 are both presented. Moreover, specific implementations are designed and described in Sec. III-B and finally tested at Chap. V.

Algorithm 1 Deep RL Pseudo Algorithm

```
Initialize Neural Network
Initialize Pattern Set P
Initialize Memory Replay D
for each episode step do
    Observe o_t and convert to s_t
    Select a_t according to ε-greedy((nnetQ,s_t,ε))
    Take action a_t, observe r_{t+1}, o_{t+1}
    Store Experience in D
    Use unsupervised learning method with o
    observations from D
    Use encoder F : X → Z to convert
    observations o into s = z
    P.s^i_t ← s
    P.s^i_{t+1} ← s_{t+1}
    P.r^i ← r
    P.a^i ← a
    if s == terminal then
        Break
    end if
    run(SupervisedLearningMethod(nnetQ,P))
    Reset Pattern Set P
end for
```

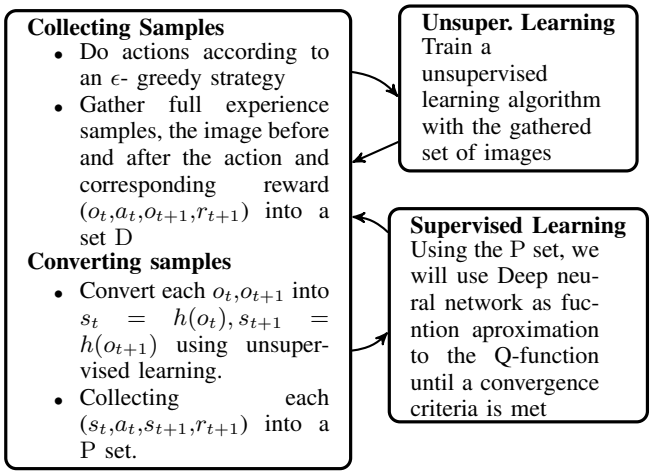


Fig. 1: Deep Reinforcement Learning Scheme.

B. Test Bed Environment

1) Toy Problem:Maze:

- **A finite set of states** S**,** each image (100×100 pixels used, resulting in 10000-dimensional input vectors) produced by the interaction between the maze and our learning agent.
- **A finite set of actions** A**,** four actions, ($a_t = 0 : left$), ($a_t = 1 : right$), ($a_t = 2 : up$) and ($a_t = 3 : down$).
- **A reward function** $r = R(s_t,a_t,s_{t+1})$**,** the immediate reward of performing a given action a_t in the maze: $r = 0$, if the agent moves to an empty space; $r = -1$, if a wall is met; and $r = 1$, if the agent reaches the goal.

2) Mountain Car: This classic problem was first described in [16]. A car starting from the bottom of a two-hill mountain aims to reach the top of the right hill. The car is not able to reach the top applying only one action (right velocity), it has to balance to the left first and gain momentum on the descent.

$$\begin{aligned} x_{t+1} &= x_t + v_{t+1}\tau \quad , x \in [-1.2,0.6](m) \\ v_{t+1} &= v_t + (a-1)0.001 + cos(3x_t)(-0.0025)\tau \\ &\quad , v \in [-0.07,0.07](m/s) \end{aligned} \tag{11}$$

Where $\tau = 1 \quad second/step$ and a varies from (0,1,2). image of the car here

- **Environment dynamics** $\mathbf{s}_t$: ($x = position$,$v =$

$velocity) \in S$, which represents a vector of position and velocity, following the dynamics presented above Eq. 11. The starting position is $x_0 = -0.5(m)$ at the middle of the hill. The goal is met if the cart is able to reach $x > 0.5(m)$.

- **A finite set of actions** A, three actions, apply left thrust ($a_t = 0 : left$), no action ($a_t = 1 : NoAction$) and right thrust($a_t = 2 : right$).
- **A reward function** $r = R(s_t,a_t,s_{t+1})$, the reward is $r = -1$ across all the state-space.

3) Cart Pole: Simplified cart pole problem without friction [17] (between cart/pole and cart/track). The angle between the pole and the verical axis is θ and x is the horizontal position of the cart. The problem dynamics can be converted to by the following equations:

$$\begin{aligned} x_{t+1} &= x_t + \tau \dot{x}_t \\ \dot{x}_{t+1} &= \dot{x}_t + \tau \ddot{x}_t \\ \ddot{x}_t &= \frac{F + ml(\dot{\theta}_t^2 sin(\theta_t) - \ddot{\theta}_t cos(\theta t))}{m + M} \\ \theta_{t+1} &= \theta_t + \tau \dot{\theta}_t \\ \dot{\theta}_{t+1} &= \dot{\theta}_t + \tau \ddot{\theta}_t \\ \ddot{\theta}_t &= \frac{gsin(\theta_t) + cos(\theta_t)(\frac{-F - ml\dot{\theta_t}^2 sin(\theta_t)}{m+M})}{l(\frac{4}{3} - \frac{mcos(\theta_t)^2}{m+M})} \end{aligned} \tag{12}$$

where $\tau = 0.02(seconds/step)$, the length of the pole $l = 0.5(m)$, $F = \pm 10$ is the magnitude of the force applied by our agent. The mass of the pole is $m = 0.1(kg)$, mass of the cart $m = 1.0(kg)$ and gravity $g = 9.8(ms^{-2})$

- **Environment dynamics** $\mathbf{s}_t : (x_t, \dot{x}_t, \theta_t, \dot{\theta}) \in S$, consisting of a state for the agent, following the dynamics presented above Eq. 12. The starting position is s_0, a uniform randomized 4-D vector with boundaries for each variable of $[-0.05, 0.05]$. The goal is to balance the pole as long as possible. Each episode stops when $||x_t|| > 2.4(m)$ or $||\theta_t|| > \frac{24}{360}(rad)$
- **A finite set of actions** A, two actions, apply left thrust ($a = -1 : F = -10(N)$), or right thrust($a = 1 : F = 10(N)$).
- **A reward function** $r = R(s_t,a_t,s_{t+1})$, in the used model, the reward is $r = 1$ in every state.

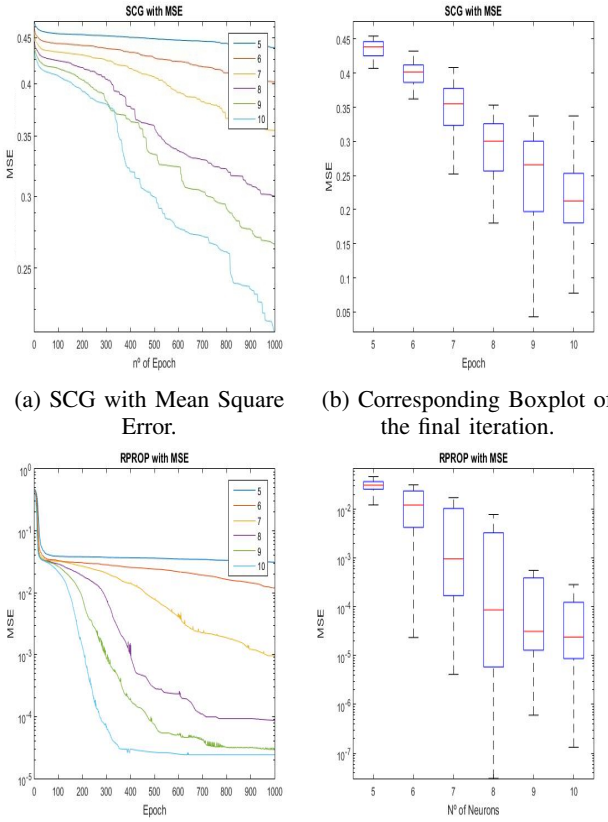
IV. STATE REDUCTION DESIGN

Comprehensive tests varying the number of neurons, number of layers, and testing with two different optimization procedures were performed. A random dataset was provided (simulating data set $\mathcal{D}$) with one hundred images from different frames of the maze shown in Fig. 12. In sum these tests, lead us to the selection of the final architecture. Based on this results, this architecture will be coupled with the NFQ for Deep Reinforcement Learning at Section V.

A. Auto-Encoders varying the number of neurons

The number of neurons at the bottleneck have a strong impact in the reconstruction error. Each neuron increases the complexity of our model by adding more degrees of freedom to the reconstruction. Many neurons could be added to the bottleneck (middle low dimensional hidden layer) as in the case of an over-complete auto-encoder, requiring increased computational and time efforts. Approaching the Curse of Dimensionality [18] problematic, a reduced number of neurons at the bottleneck is desirable. Starting with only five neurons at the first shallow auto-encoder, which will then be increased up to ten neurons on the pre-training phase. In order to have statistic relevance within these experiments, weights are randomly initialized. One hundred full trainings were preformed until convergence (low absolute value of the training criterion) or with a maximum of one thousand epochs, to reveal robustness in the optimization procedures.

Increasing the number of neurons has a strong impact in the performance measured (as seen in Fig 2). Adding additional neurons give our model extra degrees of freedom, capturing the data distribution accurately.



(a) SCG with Mean Square Error.

(b) Corresponding Boxplot of the final iteration.

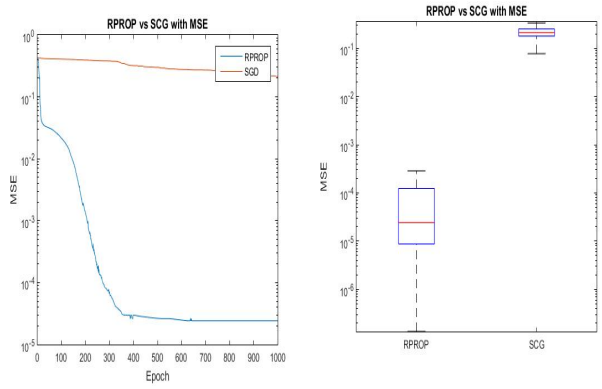
(c) RPROP with Mean Square Error.

(d) Corresponding Boxplot of the final iteration.

Fig. 2: Tests performed on AEs varying the number of neurons from five to ten at the bottleneck using SCG and RPROP. Boxplots show the median and variance of the error at the last Epoch on one hundred tests.

B. Auto-Encoders varying the optimization procedure

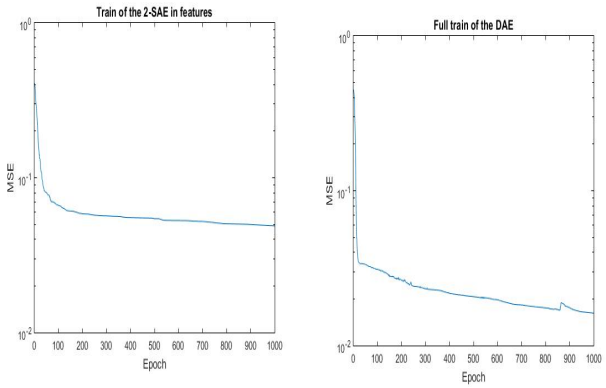
In this section, a comparison between both training procedures used, RPROP and SCG, was elaborated, when applied



(a) SCG vs RPROP with MSE.

(b) Boxplot of the final iteration.

Fig. 3: Comparing different optimization procedures with a AE of ten neurons.



(a) Two neuron Auto-Encoder trained in features.

(b) Stacked Auto-Encoder.

Fig. 4: Pre-Training and Full Training of the SAE. The ten neuron AE is used to convert the observations into features by using the resulted encoder. Afterwards, a two-neuron AE is trained in features and full training is performed by copying the weights of the two-neuron AE into the hidden layer of the SAE. Finally full training is executed and the results shown.

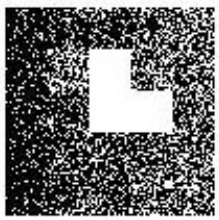
to the ten neuron shallow Auto-Encoder, which had the lowest reconstructions errors on the previous section. As illustrated in Fig 3 and stated at [19], the convergence is accelerated by using RPROP. A limit of one thousand epochs was kept, since most RPROP tests converged below this number.

Furthermore, the results obtained here, corroborate [20], in which was stated that RPROP is superior to SCG (in achieving a small MSE error amongst the training data set), when the objective function contains discontinuities or the number of patterns is small.

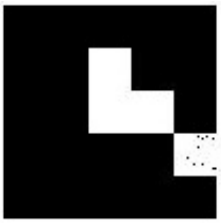
C. Stacked Auto-Encoder Resolution

It was mathematically demonstrated [21] and empirically put in practice with [22], that a properly trained Stacked auto-encoder could compress deeper information. Deeper architectures are indeed exponentially more prolific in linear regions [23] than shallow ones. As stated before, a short internal representation of each state is desirable in order to increase our Q-value approximation performance. Since our test bed environment is rather simple Sec. III-B the hypothesis of a two neuron seems sound (as an analogy of (x,y) coordinates to describe the position of our agent). A stacked auto-encoder is obtained according to by stacking the auto-encoders are re-training end-to-end. A second shallow auto-encoder was trained with the features (forward-pass on the input) generated by the first auto-encoder Fig. 4b chosen in the previous sections and the final train of the Stacked Auto-Encoder was obtained Fig. 4 in order to guarantee a meaningful two-dimensional code.

Both the feature training on the second auto-encoder and the full Stacked Auto-Encoder training revealed satisfying results when comparing Fig. 5 to the shallow auto-encoder . From Fig 4 the necessity of a Stacked Auto-Encoder



(a) Example of an image retrieved from a two-neuron shallow Auto-Encoder after convergence.



(b) Example of the same image, retrieved from a Stacked Auto-Encoder with a two-neuron bottleneck.

Fig. 5: Comparison between a 2-AE and a 2-SAE about the reconstruction of an image from the dataset.

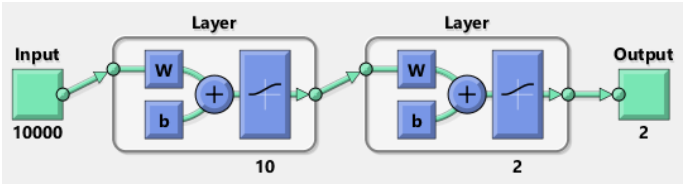


Fig. 6: Deep Encoder from raw images. The input represents a $10000by1$ vector, $\mathbf{W}$ represents a matrix of weights and $\mathbf{b}$ represents the bias of each block. Each number bellow a box represents its quantity within each layer.

was acknowledged in order to achieve a meaningful two dimensional code of every image within dataset $\mathcal{D}$.

V. EXPERIMENTAL RESULTS: NFQ

This sections aims to validate the proposed framework in III by implementing and testing it on our maze. The encoded states produced by the Stacked Auto-Encoder allow for increased computational speed, since we will have to iterate thousands of times during the training of our Q-value neural network, and about a dozen times (meaning 3000×12) more using the fitted Q-iteration to propagate rewards through Q-values. The starting structures were initially thought with [24] in mind.

One of the main concerns when applying Deep Learning as our function approximation is the impact of a bad initialization has on the Q-function itself. However, in DNN's there is an exponential number of good local minima due to the existence of saddle points [25], close to the global minimum.

A concern rises from clipping the rewards between [0,1], turning impactful rewards indistinguishable from average ones. Also, if the back-propagated error by the network is large at some point, the policy diverges from the optimal policy.

Another issue is related to a known Q-value oscillation (thus policy) caused by a biased data set $\mathcal{D}$. An imbalanced set of transitions across the state space will lead to suboptimal policies or overwhelm the impact of some transitions across the state space in the training phase. This is partially solved by a sampling strategy (as for example an sampling heuristic from the goal [24]) or a better exploration algorithm.

The generalization of Q-values for different regions of the state space that were not initially intended, might also be problematic, since the network is fully connected.

Therefore, an architecture flexible enough is required to deal with the above mentioned problems. Such a design has to keep a low error across every iteration in order to converge to an optimal policy. Furthermore, experimental results from the two structures presented are used for comparison and further validation and understanding of different valid options.

A. First NFQ variant (NFQ1, one Q-value at the output)

- **3 inputs, 2 hidden Layers and 1 output,** the inputs correspond to the states and actions and the output corresponds to $Q(s,a;\boldsymbol{\theta})$. Each hidden layer has 80 units, and each hidden unit is composed by $\Phi(a) = tanh(a) = \frac{2}{1+e^{-2a}} - 1$ non-linearity. Our output layer has a sigmoid activation, since the rewards are clipped from [0;1].
- **Loss function,** mean squared error, between $r + \gamma \max_{a_{t+1}} Q(s_{t+1},a_{t+1};\boldsymbol{\theta}_i)$ and the output $Q(s_t,a_t,\boldsymbol{\theta}_i)$.
- **A training criterion and regularizer,** simply the loss function, since the state-space is small.
- **Define an optimization procedure,** RPROP optimization algorithm explained in detail.

One hundred tests were performed by iterating 10 times (needed to solve the maze from the starting point to the goal) a data set of 10 different episodes including the goal on the maze III-B. The median value of each iteration is shown above:

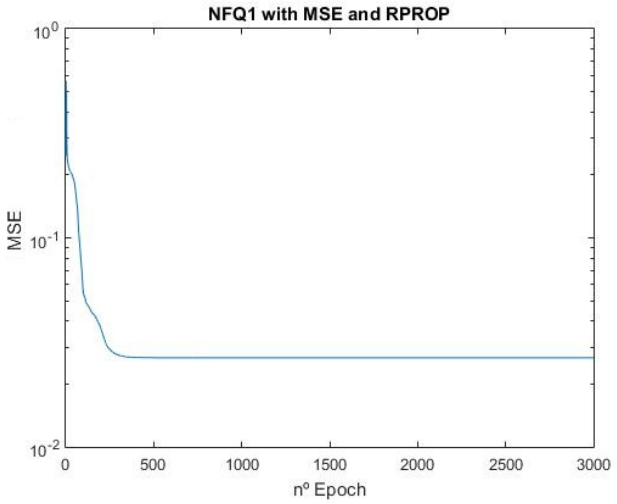


Fig. 8: Training error on the NFQ1 7 architecture, after fifteen FQI iterations (where there are more Q-values different then zero). The median error of each Epoch is presented with a statistic relevance of one hundred tests.

B. Second NFQ variant (NFQ2, one Q-value per action)

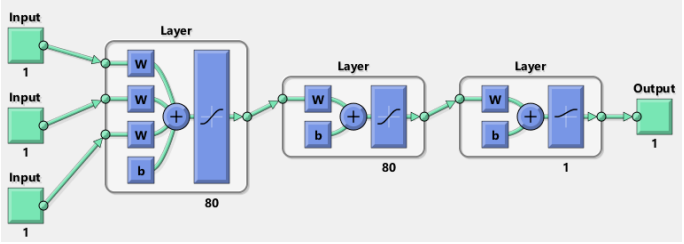


Fig. 7: Neural Fitted Q Network with states and actions as input and one Q-value output (interchangeably named NFQ1).

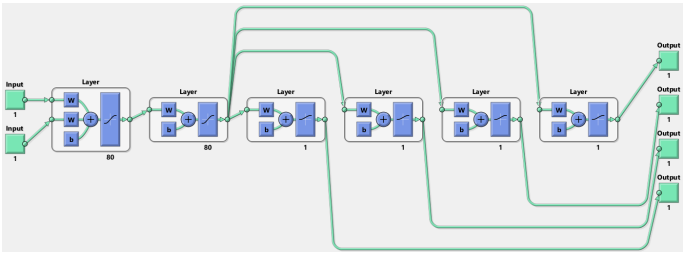


Fig. 9: Neural Fitted Q Network with states and actions as input and one Q-value output (interchangeably named NFQ2).

- **2 inputs, 2 hidden Layers and 4 outputs,** the inputs correspond to the states, and actions correspond to four outputs $Q(s,a;\boldsymbol{\theta})$. Each hidden layer has 80 units, each hidden unit represents a $\Phi(a) = tanh(a) = \frac{2}{1+e^{-2a}} - 1$ non-linearity. Our output layer has four sigmoid activations, since the rewards are clipped from [0;1].
- **Loss function,** mean squared error between $r + \gamma \max_{a_{t+1}} Q(s_{t+1},a_{t+1},\boldsymbol{\theta}_i)$ and our output $Q(s_t,a_t,\boldsymbol{\theta}_i)$.
- **A training criterion and regularizer,** simply the loss function, since the state-space is small.
- **Define an optimization procedure,** RPROP optimization algorithm explained in detail.

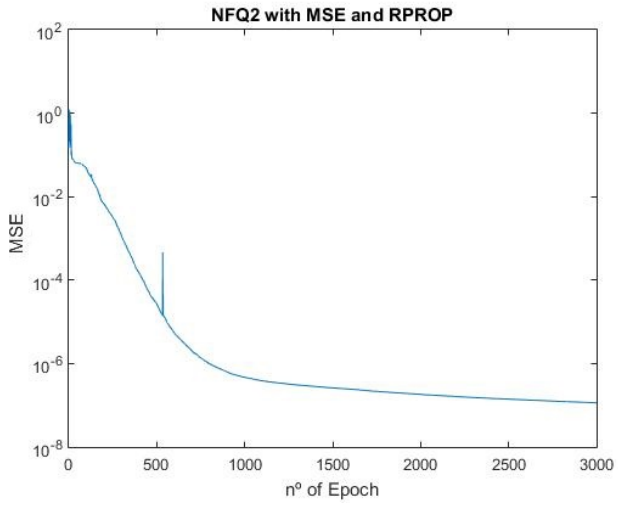


Fig. 10: Median of the training error on the last iteration of the NFQ2 9 architecture, after fifteen FQI iterations (where there are more Q-values different then zero). The median error of each Epoch is presented with a statistic relevance of one hundred tests.

C. Final notes on NFQ

Both architectures converged to an optimal policy, leading from the starting point to the goal. The last structure (NFQ2) clearly surpassed the first one (NFQ2) in a performance perspective, illustrated by Fig 11.

Furthermore, the variance in the first structure NFQ1 is also lower when comparing to the variance of the second structure NFQ2, pointing towards a more stable version of the non-linear Q-value approximation. Since finding $max_a Q(s_{t+1},a_{t+1})$ for every Q-value update requires a linear amount of cycles proportional to the number of actions for the first structure NFQ1, there are far more computational costs to be considered.

Nevertheless, the first Q-approximation is fairly interesting, once it does not limit the number of actions. If the number of actions was unknown or the same structure needed to be used for different problems, the aforementioned architecture would be the only solution. However, the fitted Q-algorithm has an inherent disadvantage: re-learning the Q-function, each time more information is gathered, becomes costly. In the next section, a highly efficient method to perform Deep Reinforcement Learning with batch-learning on an increasing replay memory is addressed.

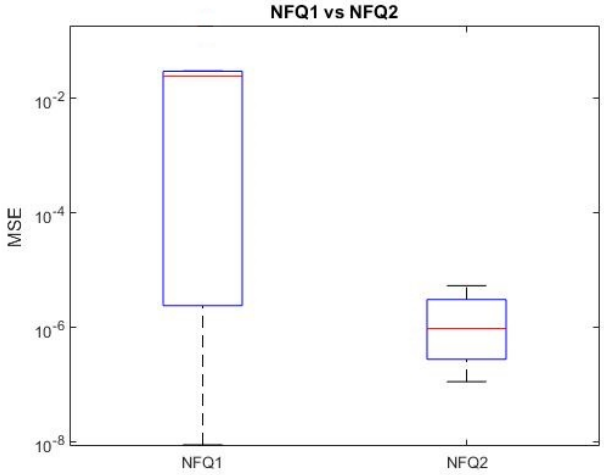


Fig. 11: Boxplot of both NFQ1 and NFQ2 on the last epoch. NFQ presents a larger median and variance error, pointing towards a more stable version.

VI. EXPERIMENTAL RESULTS: PARTIAL DQN

To further test the hypothesis, a deep feed-forward network is able to perform supervised learning on a larger dataset, for Q-learning purposes, several classic control problems Sec. III-B were chosen.

Since a certain precision on Q values has been reached in the former experiments, by both NFQ1 and NFQ2, an architecture of similar dimensions was employed. Moreover, compact states are used (states from the emulator) for this endeavor, instead of using the whole perceptual information (frames).

Although batches of frames could have been used as a higher-dimensional input representation, the Stacked Auto-Encoders tests were considered to be successful and the employment in this setting would rise the computational resources much further. Nevertheless, the effectiveness of

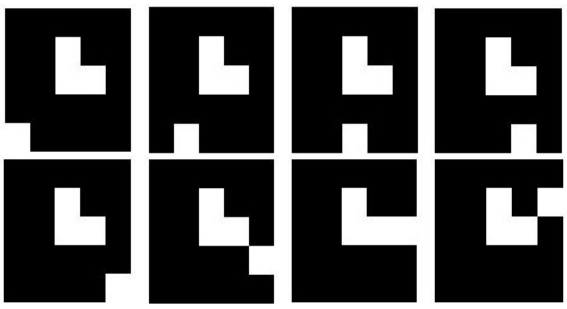


Fig. 12: Sequence of images retrieved from the game, using the Deep NFQ, after convergence.

Stacked Auto-Encoders and variations has been recently researched and employed with a high degree of success [14].

The reasoning behind this experiment is to verify how the *Q-approximation* (fully connected network) stands on a much larger state space. Leveraging the former proposed architecture, a simplified version of DQN was elaborated, approximately corresponding to the final fully connected layers of the Google's DQN [9] (double hidden layer of 256 units). In addition, the amount of hidden units was lowered to be closer to our former architectures (one hundred units each hidden layer). The first convolutional layers (of the DQN architecture) aimed to compress pre-processed frames into meaningful features, serving a similar purpose of a *Stacked Auto-encoder*, but with a much higher efficiency in the case of images.

In this particular set-up, all figures were obtained with the same hyper-parameters, under the same computational tests. The choice of hyper-parameters is partially learning independent, meaning the algorithm is robust to changes, once it will only hinder the speed/efficiency of learning. Nevertheless, the most impactful changes are enumerated as follows:

- **N inputs, 2 hidden Layers and N outputs,** the inputs correspond to the states values for each problem and one output $Q(s,a;\boldsymbol{\theta})$ per action. Each hidden layer has M units, each unit has a $\Phi(a) = ReLU(a)$. Outputs are linear activations, once rewards are no longer clipped.
- **Loss function,** mean squared error for each mini-batch between our output $Q(s_t,a_t,\boldsymbol{\theta}_i)$ and $r + \gamma \max_{a_{t+1}} Q(s_{t+1},a_{t+1};\boldsymbol{\theta}_{i-1})$.
- **Training criterion and regularizer,** coupled with a L2 weight decay $\lambda = 0.001$.
- **Optimization procedure,** ADAM [26] as a faster method than RPOP. The ADAM hyper-parameters used were: $\alpha = 0.001$; $\beta_1 = 0.9$; $\beta_2 = .999$; $\epsilon = 1 \cdot 10^-8$.
- **Fixed Target-Network,** a copy of weights and biases of the current network every N iterations.

A. Problem: Mountain Car

After two thousand episodes, the Q-agent obtained an average rewards of -332 per last 100 episodes Fig. 13. According to [16] the Mountain Car problem is considered solved when an agent achieves -110 rewards per last 100 episodes. Despite the agent's inability to solve the task in two thousand episodes, the result was considered to be a reasonable starting point of a free-model architecture. Nevertheless, further tests and more computational resources could be spent to guarantee resolvability.

The cost at each training step was gathered for further inference of our agents performance Fig. 14. The median and minimum cost within each episode are considerably low (mainly due to the impact ADAM has on accelerating propagation) and the maximum cost was highlighted at red. The red spikes reflect each cost immediately before the transfer of parameters between the current and target-network, when predictions differ the most. A slight decrease in these jumps is expected, since the Q-network parameters should change in a slower rate as learning progresses.

The Q-values at the beginning of each episode reflect the agent's initial policy around the starting state. From Fig. 15 there is no clear choice from our agent, corroborating the belief that an optimal policy was not found (further propagation should be performed to guarantee convergence, since $Q_i = Q^*$ when $i \to \infty$, but the former would also be no guarantee since the approximation error would hinder this convergence). Nevertheless, a mean value of -68 for the three Q-values was reached, forcing our agent to balance from the bottom of the two-sided hill.

An average of the last one hundred Values (max operation of the output after a forward pass) at the end of each episode was recorded for further inspection Fig. 16. At the beginning,

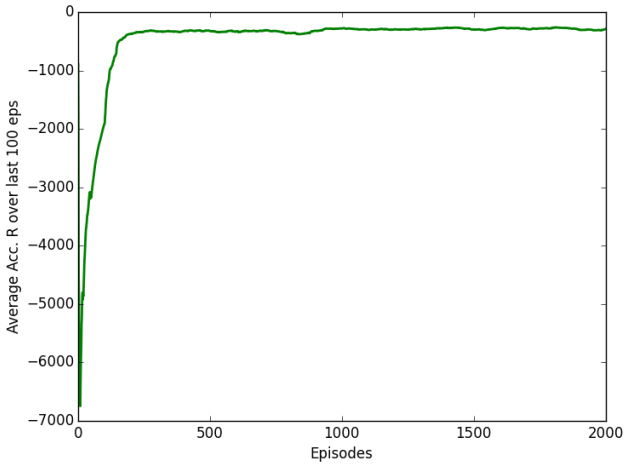


Fig. 13: Average Rewards over every 100 episodes on the MC problem

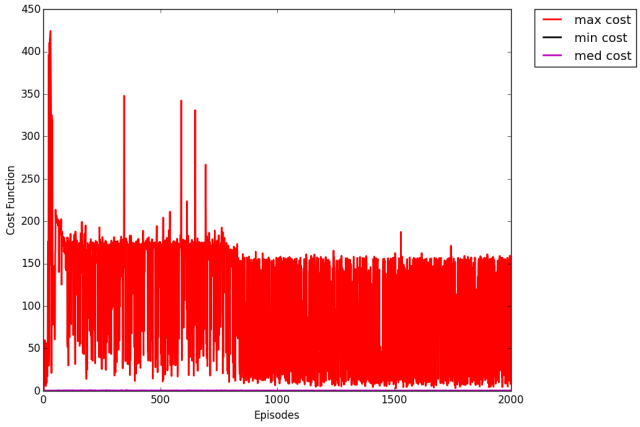


Fig. 14: Cost over each episode on the MC problem

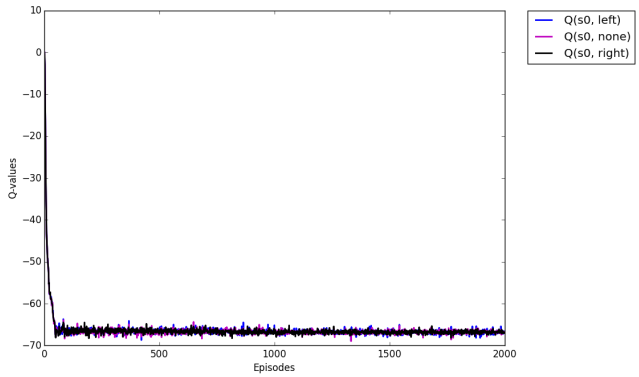


Fig. 15: Q-values at the start of each episode on the MC problem

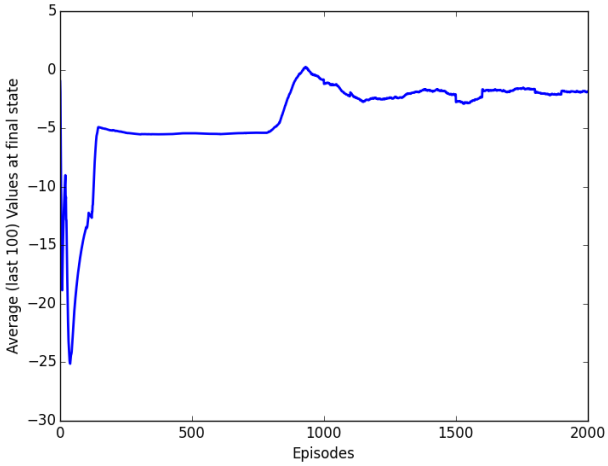


Fig. 16: Value function of the last episode on the MC problem

around the first one hundred episodes, our Q-values are further bellow -1. This correlates with the fact of our agent first 5-20 episodes end without the goal being met. After a certain number of concrete episodes ending with a goal met, our Network prediction approaches to -1. The former leads to the conclusion that the agent learns the optimal policy around the goal and correctly propagates change through the rest of the state-space.

B. Problem: Cart Pole

As a generalization test of the former Q-learning framework, a second task: Cart Pole Problem Sec. III-B was chosen to access the validity of the approach. According to [27], the problem is considered "solved" after achieving an average reward of 195. The code and hyper-parameters are equal to the Mountain Car task and the code is exactly the same, except in the definition of the OpenAI gym environment and the target-network transfer rate. Since each episode has a random starting point (initial angle), early episodes ended with an average of 10-20 steps. A target network rate of one thousand would make for an extremely slow learning. The rate was changed to ten to achieve a higher average reward earlier.

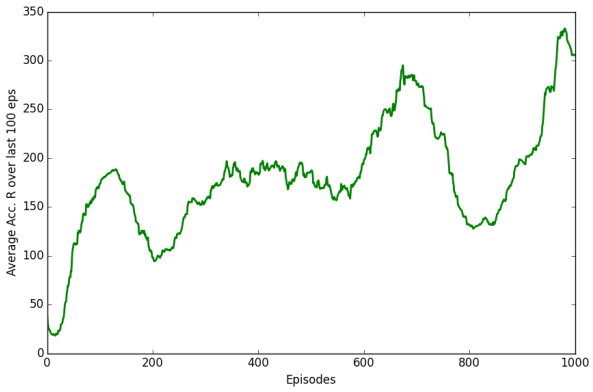


Fig. 17: Average Rewards over every 100 episodes on the CP problem.

One noticeable change when comparing Fig. 17 to the former problem is the increased oscillation of our agent's performance. As mentioned this oscillation is due to our stochastic optimization procedure, considering different state-spaces within each step. Nevertheless, the trend (considering a mean of the average) gathered rewards per episode was increasing, achieving an average of 311 rewards after one thousand episodes. The cost within each episode has the same interpretation of the Mountain Car Problem, but the decrease in maximum cost is more abrupt, this fact is also related with the reduced target-network update. Q-values from both actions are gathered and presented in Fig. 19. The starting state in each episode is not static as in the previous problem. The Cart-Pole spawns on a random position, generating different starting dynamics each time. Nevertheless, our Q-

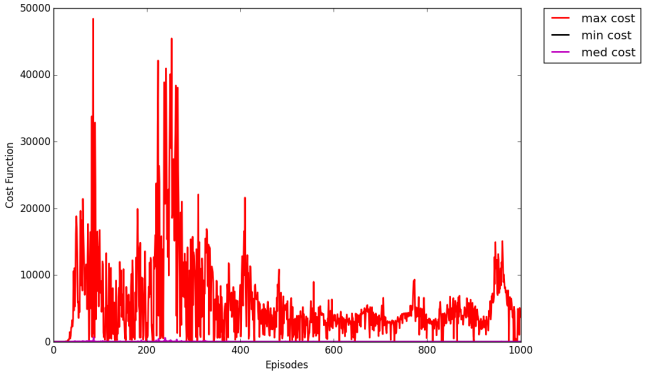


Fig. 18: Cost over each episode on the CP problem.

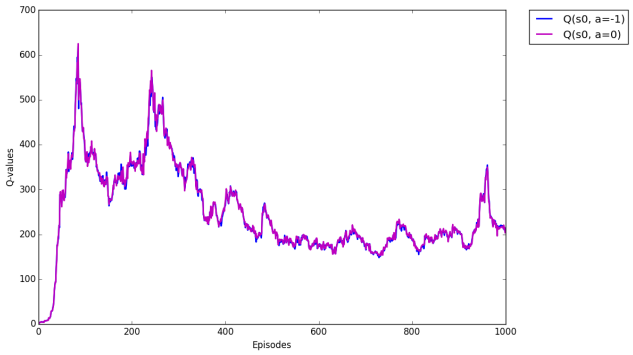


Fig. 19: Q-values at the start of each episode on the CP problem.

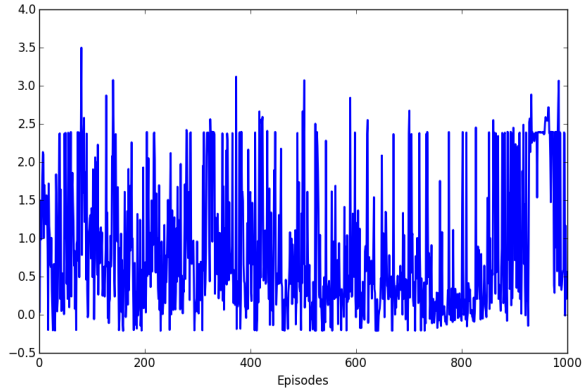


Fig. 20: Value function of the last episode on the CP problem.

network predicts about two hundred steps until failure, which is above the "solved" flag consideration.

From Fig. 20 the value function varies from 0 to 2.5 (the correct value should be around 1). The oscillation is also due to the update-rate and the fact the optimization procedure derives from a stochastic sampling.

VII. CONCLUSIONS

This thesis approached the challenge of designing methods to improve from classic reinforcement learning to "Deep" reinforcement learning by allowing different state representations and function approximations. Unsupervised learning methods were used on high dimensional inputs. Several architectures to approximate Q-function and consequently Q-iteration algorithms were designed, specifically to apply supervised learning alongside a replay memory $\mathcal{D}$. Future work will include mainly the incorporation of memory to deal with long term dependencies in a reinforcement learning framework as a whole. One of the solutions might be introducing a LSTM (Long Short Term Memory neural network) already being very popular in translation. An attention window mechanism could also be incorporated to look at only a part of the high dimensional input and process sequences of actions-states instead of just one at a time. The second solution could be using a LSTM as an approximation of the Q-value function, which could interpret long term dependencies and enhance correlations within sequences of states, creating a deep interpretation (a history rich one).

REFERENCES

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[2] Dan CirecAn, Ueli Meier, Jonathan Masci, and Jurgen Schmidhuber. Multi-column deep neural network for traffic sign classification. *Neural Networks*, 32:333–338, 2012.

[3] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[4] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6):82–97, 2012.

[5] Ian J Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay Shet. Multi-digit number recognition from street view imagery using deep convolutional neural networks. *arXiv preprint arXiv:1312.6082*, 2013.

[6] Minh-Thang Luong, Ilya Sutskever, Quoc V Le, Oriol Vinyals, and Wojciech Zaremba. Addressing the rare word problem in neural machine translation. *arXiv preprint arXiv:1410.8206*, 2014.

[7] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[8] Sascha Lange and Martin Riedmiller. Deep auto-encoder neural networks in reinforcement learning. In *Neural Networks (IJCNN), The 2010 International Joint Conference on*, pages 1–8. IEEE, 2010.

[9] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[10] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[11] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy P Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *arXiv preprint arXiv:1602.01783*, 2016.

[12] Jan Mattner, Sascha Lange, and Martin Riedmiller. Learn to swing up and balance a real pole based on raw visual input data. In *International Conference on Neural Information Processing*, pages 126–133. Springer, 2012.

[13] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

[14] Eder Santana and George Hotz. Learning a driving simulator. *arXiv preprint arXiv:1608.01230*, 2016.

[15] Martin Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *Machine Learning: ECML 2005*, pages 317–328. Springer, 2005.

[16] Andrew William Moore. Efficient memory-based learning for robot control. 1990.

[17] António E Ruano. *Intelligent control systems using computational intelligence techniques*, volume 70. Iet, 2005.

[18] David L Donoho et al. High-dimensional data analysis: The curses and blessings of dimensionality. *AMS Math Challenges Lecture*, pages 1–32, 2000.

[19] Martin Riedmiller and I Rprop. Rprop-description and implementation details. 1994.

[20] Rprop-scg comparison. `http://www.ra.cs.uni-tuebingen.de/SNNS/SNNS-Mail/96/0123.html`.

[21] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[22] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[23] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *Advances in neural information processing systems*, pages 2924–2932, 2014.

[24] Martin Riedmiller. 10 steps and some tricks to set up neural reinforcement controllers. In *Neural Networks: Tricks of the Trade*, pages 735–757. Springer, 2012.

[25] Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2933–2941. Curran Associates, Inc., 2014.

[26] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[27] Andrew G Barto, Richard S Sutton, and journal= Anderson, Charles W. Neuronlike adaptive elements that can solve difficult learning control problems.

[28] Timothy F Cootes, Gareth J Edwards, and Christopher J Taylor. Active appearance models. In *Computer Vision—ECCV'98*, pages 484–498. Springer, 1998.

[29] GE Hinton. Reducing the dimensionality of data with neural. *IEEE Trans. Microw. Theory Tech*, 47:2075, 1999.