I did a 14 week long research internship at Google Deepmind, London, UK, as part of my final project. The internship went well enough for me to be hired fulltime as a Research Engineer starting February 5, 2018. Due to confidentiality I am not able to write about my work there.
A very brief introduction to Deep Reinforcement Learning

Felix Axel Gimeno Gil
Facultat d’Informàtica de Barcelona
Universitat Politècnica de Catalunya

Supervisor

Oriol Vinyals
DeepMind, London, UK

University Supervisor

Jordi Torres Viñals
Computer Architecture Department
Universitat Politècnica de Catalunya

January 15, 2017
Abstract

English
Deep Learning is allowing computers to solve unsolved problems. Reinforcement Learning is a general framework to model the real world from an agent-environment perspective. Deep Reinforcement Learning seems like a possible way to reach Artificial General Intelligence.

Català
Deep Learning està permetent que els ordinadors resolguin problemes no resolts. Reinforcement Learning és un marc general per modelar el món real des d’una perspectiva agent-medi ambient. Deep Reinforcement Learning sembla una forma possible d’arribar a la Intel··ligència General Artificial.

Castellano
Deep Learning permite que las computadoras resuelvan problemas sin resolver. Reinforcement Learning es un marco general para modelar el mundo real desde una perspectiva agente-ambiente. Deep Reinforcement Learning parece una forma posible de alcanzar la Inteligencia General Artificial.
Acknowledgements

Thanks to all the people who collaborated with me, helped me, and guided me.
Contents

List of Figures vi

List of Tables vii

1 Introduction to Machine Learning 1

1.1 What is Machine Learning? . . . . . . . . . . . . . . . . . . 1

1.1.1 As optimization . . . . . . . . . . . . . . . . . . . . . . . 1

1.1.2 As interpolation . . . . . . . . . . . . . . . . . . . . . . . 1

1.2 Current Trend . . . . . . . . . . . . . . . . . . . . . . . . . . 1

1.3 Supervised Machine Learning . . . . . . . . . . . . . . . . . 2

1.4 Unsupervised Machine Learning . . . . . . . . . . . . . . . . 2

1.5 Duality . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

1.6 Some definitions . . . . . . . . . . . . . . . . . . . . . . . . . . 3

1.6.1 Hypothesis space . . . . . . . . . . . . . . . . . . . . . . . 3

1.6.2 Prior . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

1.6.3 Capacity . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

1.6.4 Data . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

1.6.5 Train set . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

1.6.6 Validation set . . . . . . . . . . . . . . . . . . . . . . . . 3

1.6.7 Test set . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

1.6.8 Loss function . . . . . . . . . . . . . . . . . . . . . . . . 3

1.6.9 Train error . . . . . . . . . . . . . . . . . . . . . . . . . . 4

1.6.10 Test error . . . . . . . . . . . . . . . . . . . . . . . . . . 4

1.6.11 Generalization error . . . . . . . . . . . . . . . . . . . . 4

1.6.12 Underfitting . . . . . . . . . . . . . . . . . . . . . . . . . 4

1.6.13 Overfitting . . . . . . . . . . . . . . . . . . . . . . . . . . 4

1.6.14 Dataset bias . . . . . . . . . . . . . . . . . . . . . . . . . 4

1.6.15 Regularization . . . . . . . . . . . . . . . . . . . . . . . . 4

1.7 No Free Lunch Theorem . . . . . . . . . . . . . . . . . . . . . 4
# 2 Introduction to Deep Learning

## 2.1 Neural Network Basic Definitions

- **2.1.1 Neuron**
- **2.1.2 Layer**
- **2.1.3 Network**

## 2.2 Why non-linearities?

## 2.3 Training

## 2.4 Neural Networks Data-Scalability

## 2.5 Neural Networks Priors

# 3 Introduction to Reinforcement Learning

## 3.1 Multi-Arm Bandit

- **3.1.1 Exploration vs Exploitation**
- **3.1.2 $\epsilon$-greedy policies**

## 3.2 Markov Decision Process formalism

- **3.2.1 Why?**

## 3.3 Markov Decision Process

- **3.3.1 States**
- **3.3.2 Actions**
- **3.3.3 Transitions**
- **3.3.4 Rewards**
- **3.3.5 Discount**
- **3.3.6 Policy**
- **3.3.7 Value functions**

# 4 Introduction to Deep Reinforcement Learning

## 4.1 Deep Q Networks (DQN)

## 4.2 Asynchronous advantage actor-critic (A3C)

## 4.3 AlphaZero

- **4.3.1 Monte Carlo Tree Search (MCTS)**
- **4.3.2 Self-Play**

# 5 Project Management

## 5.1 Project Planning

- **5.1.1 Objectives**
- **5.1.2 Tasks**
- **5.1.3 Outline**
- **5.1.4 Author time planning**
- **5.1.5 Human costs**
- **5.1.6 Personal hardware costs**
- **5.1.7 Software costs**
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1.8 Cloud computing costs</td>
<td>16</td>
</tr>
<tr>
<td>5.1.9 Total cost</td>
<td>16</td>
</tr>
<tr>
<td>5.2 Knowledge used</td>
<td>16</td>
</tr>
<tr>
<td>5.3 Legality</td>
<td>16</td>
</tr>
<tr>
<td>6.1 Economic</td>
<td>18</td>
</tr>
<tr>
<td>6.2 Social</td>
<td>18</td>
</tr>
<tr>
<td>6.3 Ecological</td>
<td>18</td>
</tr>
<tr>
<td>6.4 Sustainability Matrix</td>
<td>18</td>
</tr>
<tr>
<td>7.1 Always check libraries are thread-safe</td>
<td>19</td>
</tr>
<tr>
<td>7.2 Always search hyperparameters</td>
<td>19</td>
</tr>
<tr>
<td>7.3 Always do sanity checks</td>
<td>19</td>
</tr>
<tr>
<td>7.4 Always do static analysis and tests</td>
<td>19</td>
</tr>
<tr>
<td>7.5 Always check for aliasing</td>
<td>20</td>
</tr>
<tr>
<td>A Bibliography</td>
<td>21</td>
</tr>
</tbody>
</table>
List of Figures
List of Tables

5.1 Time planning ......................................................... 15
5.2 Human resources costs .............................................. 15
5.3 Hardware costs ....................................................... 16
5.4 Cloud computing costs .............................................. 16
6.1 Quality of estimation of different impacts ..................... 18
Chapter 1

Introduction to Machine Learning

1.1 What is Machine Learning?

According to Tom Mitchell, Machine Learning is defined as:

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E."

1.1.1 As optimization

This operational definition shows links to mathematical optimization, as we want to optimize the performance measure P, and the experience E could be understood as what we know about the tasks T and the previous models’ performance.

1.1.2 As interpolation

Sometimes the task can be understood as a prediction problem, where the experience is multiple input/output pairs.

1.2 Current Trend

Replacing human-designed programs with learned programs, where the learning process is good enough and has sufficient computing resources and data, is
a way of improving current limits or even allowing computers to do things they couldn’t before.

For example, inside some traditional data structures there are human-designed key \( \mapsto \) position functions whose performance is improved when replaced by a learned function [1].

The tradeoff between hard-coding things and learning them is based on the symmetries or constraints of the tasks, and the amount of data and computational resources available (in some regulated industries the uninterpretability of the methods is a constraint).

### 1.3 Supervised Machine Learning

Supervision in machine learning means that the data is such that when given some input, we know the "correct" output, so it is learning a function from input/output pairs.

In a more probabilistic setting, we want the true distribution of the output \( Y \) given the input \( X \)

\[
p(Y|X)
\]

while we only know the empirical distribution

\[
p_{\text{emp}}(Y|X)
\]

Supervision is said to be strong when we know what the output should be, and weak when we only know how good is the current output.

Supervised ML include classification and prediction.

### 1.4 Unsupervised Machine Learning

Trying to learn the true distribution of \( X \)

\[
p(X)
\]

while only knowing the empirical distribution

\[
p_{\text{emp}}(X)
\]

Unsupervised ML includes compression and clustering.
1.5 **Duality**

\[ X_{\text{unsup}} = (X_{\text{sup}}, Y_{\text{sup}}) \]

An unsupervised ML problem can be understood as a supervised one and viceversa.

1.6 **Some definitions**

1.6.1 **Hypothesis space**

The space of all possible learnable models considered by our learning algorithm.

1.6.2 **Prior**

Sometimes learning algorithms have preferences towards some kinds of solutions, in a Bayesian point of view learning means updating a hypothesis distribution.

1.6.3 **Capacity**

The amount of different possible hypotheses that explain the data.

1.6.4 **Data**

The information we have about the tasks and for learning.

1.6.5 **Train set**

The part of the data used for learning parameters of the model.

1.6.6 **Validation set**

The part of the data used for learning hyperparameters of the model.

1.6.7 **Test set**

The part of the data used for evaluating the model.

1.6.8 **Loss function**

The cost function, we want to minimize.
1.6.9 **Train error**
The value of the cost function in the train set.

1.6.10 **Test error**
The value of the cost function in the test set.

1.6.11 **Generalization error**
Some authors call it the same as test error,

1.6.12 **Underfitting**
When train error is too high.

1.6.13 **Overfitting**
When train error is low but test error is high, or equivalently when generalization error is high.

1.6.14 **Dataset bias**
If the data gathered to train and evaluate a learning algorithm and get a model, is biased (isn’t a representative sample of the true target distribution) then the model learnt could do well on the current data but bad on new or differently gathered data [2][3].

1.6.15 **Regularization**
An optional term in the loss function that serves as a prior towards simpler or more generalizable models.

1.7 **No Free Lunch Theorem**
Wolpert’s *no free lunch* theorem says that no machine learning algorithm is universally better than any other when classifying unobserved points, when universally is taken over all possible data generating processes.

This means that to be able to improve our understanding and algorithms we need to constrain the data generating processes we allow, that leads to algorithms that are better in some kind of problems, usually we are interested
in making reasonable assumptions related to "real world" problems so we can study "real world" solutions (knowing that universal solutions do not exist).
Chapter 2
Introduction to Deep Learning

2.1 Neural Network Basic Definitions

2.1.1 Neuron
A neural network cell, as the one in artificial neural networks, is a scalar nonlinear function \( \sigma \) composed with a scalar affine transformation

\[
x \mapsto \sigma(c + a^T x)
\]

where \( c \) is a scalar and \( a \) is a vector.

2.1.2 Layer
A neural network layer \( f_i \) is a vector non-linear function \( \sigma \) composed with a vector affine transformation

\[
x \mapsto \sigma(b + W x)
\]

where \( b \) is a row vector and \( W \) is a matrix.

2.1.3 Network
A neural network is a composition of layers

\[
f_k \circ f_{k-1} \circ \cdots \circ f_2 \circ f_1
\]

and layers are a concatenation of neurons.
2.2 Why non-linearities?

Non-linearities are also called activation functions.
Non-linearity functions are needed because a composition of affine transformations is an affine transformation, and thus the hypothesis space of deep linear networks are linear functions only.

Sometimes we desire the activation functions to be almost-everywhere differentiable to be able to use optimization methods that use gradients (e.g. stochastic gradient descent), for a recent comparison of activation functions see [4].

2.3 Training

Most people use stochastic gradient descent or variants of it (e.g. with momentum, Adam [5], ...), with backpropagation (chain rule of calculus), applied to minimizing the loss function. Higher-order derivatives are too costly to compute and numerically unstable.

There is interest in applying optimization methods without gradients (e.g. synthetic gradients) or that use black-box optimization algorithms or evolutionary algorithms (e.g. NES [6]).

2.4 Neural Networks Data-Scalability

Dataset size is very important, larger dataset size means better performance [7], thus we need models that are able to deal with non-linearities but also scale well with more data.

Stochastic Gradient Descent is an iterative method that uses sampling, thus has a run-time that is linear in the training dataset size, which is difficult to improve on (constants apart) if we only uses algorithms that do at least one full pass over all available data for training.

2.5 Neural Networks Priors

Neural networks perform better when the functions we want to learn are a composition (or a hierarchy) of simpler functions (the functional programming view), or a sequential computer program where the outputs of each step are used in the next steps.

More depth empirically results in better generalization in some tasks.
Translation-invariance (shared-weights) and locality (weight sparsity) are priors in Convolutional Neural Networks that work well with images. See [8] for examples of priors humans have.
Chapter 3

Introduction to Reinforcement Learning

Supervised Machine Learning learns input/output pairs, which allows learning functions given that access to the its outputs, but what if we don’t have such an explicit input/output pairs, for example a deterministic board game has a (possibly non-unique) best agent, so a best agent function of board $\rightarrow$ action exists and thus can be learned, if this functions impractical to compute, how can be iteratively compute better agent functions until we spend all computational resources available? This problem is studied in the field of Reinforcement Learning.

3.1 Multi-Arm Bandit

A multi-arm bandit or slot machine is a problem where an agent chooses repeatedly which random variable $A_1, A_2, \ldots, A_k$ (called arms) to sample rewards from, trying to maximize some discounted cumulative reward.

3.1.1 Exploration vs Exploitation

Choosing which random variable is best requires knowing the true probability distributions, and as empirical probability distributions are approximations, if an agent doesn’t explore it may be getting an opportunity cost by not choosing the best one and if an agent only explores it gets an opportunity cost everytime it explores a non-optimal one.

This is the tradeoff between exploration (trying new things) and exploitation (choosing the current best thing).
The solution to the tradeoff depends on constraints on the reward distributions and how many times can the agent choose arms.

3.1.2 \( \epsilon \)-greedy policies

An agent is taking a greedy policy if it is in exploitation mode, the \( \epsilon \) in \( \epsilon \)-greedy means it is exploring an uniformly chosen non-optimal arm with a probability of \( \epsilon \).

3.2 Markov Decision Process formalism

3.2.1 Why?

A multi-step multi-arm bandit problem, where the rewards could depend on all previous actions, is an extension for making the problem closer to model reality (we add "Time").

Without loss of generality we can say that the environment has the Markov property which means we can completely model the current "state" of the environment.

An environment is the part of reality that isn’t the agent, the agent receives observations and rewards from the environment and takes actions.

3.3 Markov Decision Process

The deterministic time-discrete Markov Decision Process (MDP) framework is a 5-tuple

\[ (S, A, T(\cdot, \cdot), R(\cdot, \cdot), \gamma) \]

3.3.1 States

\( S \) is the set of possible environment states, each state giving the agent all information available to understand it.

3.3.2 Actions

\( A \) is the set of possible actions the agent can take.
3.3.3 Transitions

\[ T : S \times A \rightarrow S \]

\( T(s_i, a_i) \) is the environment’s state at time \( i + 1 \) when the agent received state \( s_i \) at time \( i \) and chose to take action \( a_i \).

3.3.4 Rewards

\( R(s, a) \) is the reward after taking action \( a \) when in the state \( s \)

3.3.5 Discount

\( \gamma < 1 \) is the discount factor, used for computing a discounted cumulative reward:

\[ \gamma^0 R_1 + \gamma^1 R_2 + \gamma^2 R_3 + \ldots \]

3.3.6 Policy

An agent follows a policy \( \pi : S \rightarrow A \) if for every state \( s \in S \) it takes action \( \pi(s) \) when in state \( s \)

3.3.7 Value functions

The state value function \( q_{\pi} : S \rightarrow \mathbb{R} \) is the discounted cumulative reward when in state \( s \in S \) and following policy \( \pi \).

\[ q_{\pi}(s) = R(s, \pi(s)) + \gamma q_{\pi}(T(s, \pi(s))) \quad \forall s \in S \]

The action value function \( q_{\pi} : S \times A \rightarrow \mathbb{R} \) is

\[ q_{\pi}(s, a) = q_{\pi}(T(s, a)) \quad \forall a \in A \quad \forall s \in S \]
Chapter 4

Introduction to Deep Reinforcement Learning

4.1 Deep Q Networks (DQN)

From actions values for some policy $\pi_{old}$ we can compute a policy that is better or equal to $\pi_{old}$:

$$\pi_{new}(s) = \arg\max_a q_{\pi_{old}}(s, a)$$

If from this policy we compute the action values and then the policy improvement, repeatedly, under reasonable assumptions we reach an optimal policy and action values [9]. So if we learn (using neural networks) and improve action values we can learn to solve RL problems like Atari games (Deep Q Network [10] and its extension Rainbow [11]).

4.2 Asynchronous advantage actor-critic (A3C)

Just like DQN works with only learning action values and having an implicit policy, we can use a policy that differentiable depends on parameters and do gradient ascent using policy-gradients (REINFORCE [12]). A3C learns a policy $\pi$ called actor and a state value function $q$ called critic.

4.3 AlphaZero

AlphaZero is a framework for solving deterministic board games [13] without using handcrafted features or domain-specific adaptions.
4.3.1 Monte Carlo Tree Search (MCTS)
States as nodes labeled with win frequency, and actions as directed edges, each player chooses actions depending on the win frequency of the nodes, and each time a player wins or losses it adjusts the frequency of the end node of the action taken.

4.3.2 Self-Play
The same neural network than outputs a predicted outcome and a policy, play again itself, whenever a game ends the parameters are updated to minimize the error between the predicted outcome and the game outcome, and to maximise the similarity of the policy vector to the search probabilities given by MCTS.

Even self-play starting from a random policy can learn \[13, 14, 15\]
Chapter 5

Project Management

5.1 Project Planning

5.1.1 Objectives

Achieve better test error than baselines

5.1.2 Tasks

Design machine learning system
Implement machine learning system
Sanity-check machine learning system
Evaluate progress and performance
Discuss improvements
Repeat

5.1.3 Outline

The rough initial sequence of tasks for the author will be designing a machine learning system, evaluate its test error and try to implement some new or old idea to improve its test score.

For supervised learning classification this could be trying different regularizers and optimizers, label smoothing, penalizing low entropy, curriculum learning; for reinforcement learning this could be reward shaping or replay buffers.

As the specific technique that should be used isn’t known until enough information is obtained, the author will denominate all of them as “Iterations” (applying new techniques/ideas).
The initial setup and each Iteration consist of some design, some implementation and some hyperparameter search, each iteration also has a discussion and idea generation.

For the duration of the project the time constraints the tasks would be the initial setup and four iterations, with a final refactoring and code cleanup if project was successful enough.

5.1.4 Author time planning

<table>
<thead>
<tr>
<th>Stage</th>
<th>Estimated dedication (human hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial setup: Design</td>
<td>50</td>
</tr>
<tr>
<td>Initial setup: Implementation</td>
<td>90</td>
</tr>
<tr>
<td>Initial setup: Hyperparameter tuning</td>
<td>30</td>
</tr>
<tr>
<td>Iterations 1..4: Idea generation</td>
<td>30</td>
</tr>
<tr>
<td>Iterations 1..4: Design</td>
<td>130</td>
</tr>
<tr>
<td>Iterations 1..4: Implementation</td>
<td>200</td>
</tr>
<tr>
<td>Refactoring and improving code quality</td>
<td>50</td>
</tr>
<tr>
<td>Making a research report</td>
<td>20</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>600</strong></td>
</tr>
</tbody>
</table>

Table 5.1: Time planning

The iterations are aggregated so the variance of the estimation is reduced 600 in 14 weeks is 43 hours per week which is in the range of healthy and feasible.

5.1.5 Human costs

This project is going to be developed only by one person but advised by other people like my supervisor, an expert in the specific research area, and an engineering mentor and code reviewer.

<table>
<thead>
<tr>
<th>Role</th>
<th>Estimated Hours</th>
<th>Estimated €/h</th>
<th>Total estimated cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supervisor</td>
<td>2h/w * 14 weeks</td>
<td>40€/h</td>
<td>1120€</td>
</tr>
<tr>
<td>Senior Research Scientist</td>
<td>1h/w 14 weeks</td>
<td>60€/h</td>
<td>840€</td>
</tr>
<tr>
<td>Mentor and Code Reviewer</td>
<td>4h/1kloc * 3kloc</td>
<td>30€/h</td>
<td>360€</td>
</tr>
<tr>
<td>Programmer (me)</td>
<td>40h/w * 14 weeks</td>
<td>15€/h</td>
<td>8400€</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td><strong>10720€</strong></td>
</tr>
</tbody>
</table>

Table 5.2: Human resources costs
5.1.6 Personal hardware costs

<table>
<thead>
<tr>
<th>Product</th>
<th>Units</th>
<th>Useful Life</th>
<th>Price</th>
<th>Amortized Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Laptop</td>
<td>1</td>
<td>2 years</td>
<td>1500€</td>
<td>200€</td>
</tr>
<tr>
<td>Workstation</td>
<td>1</td>
<td>2 years</td>
<td>1000€</td>
<td>135€</td>
</tr>
</tbody>
</table>

Table 5.3: Hardware costs

5.1.7 Software costs

Every software program used was free and open source or developed in-house. So there are no licence costs.

5.1.8 Cloud computing costs

<table>
<thead>
<tr>
<th>Cloud provider</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google Cloud</td>
<td>2000€</td>
</tr>
</tbody>
</table>

Table 5.4: Cloud computing costs

5.1.9 Total cost

The estimation of the total cost is between 10000€ and 15000€.

5.2 Knowledge used

To understand and develop successfully this project the author used the knowledge gained in the following university subjects:

- PE: Probability and Statistics
- IA: Artificial Intelligence
- APA: Machine Learning

5.3 Legality

This project was about pure research in a company setting, where the data can be thought as generated by a random number generator as the only input to some program, with which I wish to say that no human-generated data was used (that means there wasn’t any personally identifiable information
(PII) or sensitive personal information (SPI)). The project was done with a internship contract with a Non Disclosure Agreement (NDA) and as a Eramus Plus traineeship.
Chapter 6

Sustainability

6.1 Economic

Costs were estimated, the project only focuses on pure research (no revenue generation), and the project was allowed by the company so it was better than the opportunity cost, so its economic impact is positive.

6.2 Social

As pure research, there is no social impact except the self-improvement of the author.

6.3 Ecological

The energy and materials cost for this project was evaluated against the resulting quality and a reasonable tradeoff was chosen.

Other ways of reducing costs included expending more time thinking better machine learning systems and experiments, and avoiding duplicating experiments and redundant retraining.

6.4 Sustainability Matrix

<table>
<thead>
<tr>
<th>Economic</th>
<th>Social</th>
<th>Ecologic</th>
</tr>
</thead>
<tbody>
<tr>
<td>7/10</td>
<td>7/10</td>
<td>7/10</td>
</tr>
</tbody>
</table>

Table 6.1: Quality of estimation of different impacts
Chapter 7

Lessons Learned

7.1 Always check libraries are thread-safe

I had a multi-threaded part of a program that was getting Python exceptions that came from calling an external python library, which at first I thought they were from bad inputs, but then I noticed the exceptions dissapeared when only one thread was running, so I concluded it wasn’t the calls’ inputs but the part inside them, so I added reentrant locks and the problem was solved.

7.2 Always search hyperparameters

From Vizier [16] to population-based training [17], there’s a lot of interest of achieving the best models for each task.

7.3 Always do sanity checks

From not having a connected TensorFlow Graph to doing an operation in Python like == that should be a tf.equal, to not understanding why a model underfits some data and not knowing if it also underfits for an easier problem, the value of a systematic approach to debug programs and review research is limitless.

7.4 Always do static analysis and tests

Continuous integration that detects if the code changes isn’t worsening functionality or introducing previously seen bugs is a very useful feature for software
development.

This can be achieved via explicit specific tests [18] or with static analysis e.g. type checking [19].

7.5 Always check for aliasing

In Python, doing [[ ]] *5 aliases, and [[ ]] for _ in range(5)] doesn’t.
Appendix A

Bibliography


