

UNIVERSITY OF FRIBOURG

MASTER THESIS

---

# Exploration of novel Methods in Reinforcement Learning using Black-Box-Optimization

---

*Author:*  
Corina Masanti

*Supervisor:*  
Dr. Giuseppe Cuccu

*Co-Supervisor:*  
Prof. Dr. Philippe  
Cudré-Mauroux

January 01, 1970

eXascale Infolab  
Department of Informatics



# Abstract

Corina Masanti

*Exploration of novel Methods in Reinforcement Learning using  
Black-Box-Optimization*

Neural networks as generic function approximators can solve many challenging problems. However, they can only be applied successfully for a suited problem structure. In specific, neural networks require differentiability. But there are many areas where calculating an accurate gradient is non-trivial, including problems in Reinforcement Learning (RL). In contrast, Black-Box Optimization (BBO) techniques are less limiting. They presume no constraints on the problem structure, the model, or the solution. With this flexibility, we can study alternative models to neural networks that are unexplored in the context of RL. This thesis aims to achieve pleasing results with a function approximator other than neural networks. I analyze promising models optimized with BBO methods.

Problem -> Solution -> Results

**Keywords:** Black-Box Optimization, Reinforcement Learning



# Contents

<b>Abstract</b>	<b>iii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Black-Box Optimization . . . . .	1
1.1.1 Evolution Strategies . . . . .	1
1.1.2 Covariance Matrix Adaptation Evolution Strategy . . . . .	1
1.2 Function Approximators . . . . .	1
1.2.1 Neural Networks . . . . .	1
1.2.2 Polynomial . . . . .	2
1.2.3 Fourier . . . . .	3
1.2.4 Bézier . . . . .	3
1.3 Previous Work . . . . .	3
1.3.1 Benchmarks in Reinforcement Learning . . . . .	3
<b>2 Experiments</b>	<b>7</b>
2.1 Research Questions . . . . .	7
2.2 Experiments . . . . .	7
2.2.1 OpenAI Gym Environments . . . . .	7
2.2.2 Models . . . . .	8
2.3 Results . . . . .	10
2.3.1 Experiment 1 . . . . .	10
<b>3 Conclusion</b>	<b>13</b>
3.1 Conclusion . . . . .	13
3.2 Future Work . . . . .	13
<b>Bibliography</b>	<b>15</b>



# List of Figures

1.1	Sketch of a Neural Network . . . . .	2
1.2	Reproduced Plots . . . . .	5
1.3	Impact of Bias . . . . .	6
2.1	Upper and lower bound . . . . .	8
2.2	Sigmoid function . . . . .	9
2.3	Results of experiment 1 . . . . .	10





## Chapter 1

# Introduction

### 1.1 Black-Box Optimization

#### 1.1.1 Evolution Strategies

Evolution Strategies (ES) ...

#### 1.1.2 Covariance Matrix Adaptation Evolution Strategy

Covariance Matrix Adaptation Evolution Strategy (CMA-ES) ...

### 1.2 Function Approximators

We can represent many problems in machine learning as a function mapping an input space into an output space. This underlying function is the output of the learning process of an algorithm and is often called the *target function*. Optimally, we would derive the formula of the function explicitly. However, even though we suspect that such a function exists, we usually do not have enough information to derive a formula. Thus, we aim to approximate the target function with function approximators by using the available data. In general, we can apply any function approximator. However, each function approximator has its advantages and limitations. For example, some may be prone to local optimum. Depending on the task, this could prevent us from finding a good approximation.

#### 1.2.1 Neural Networks

Neural networks are machine learning algorithms inspired by the functionalities of the brain. They consist of connected neurons or nodes that imitate the biological neurons of the brain sending signals to each other. In a neural network, the neurons are arranged in connected layers. A network has at least an input and an output layer. We can further expand it with one or more hidden layers. Depending on the model, the connectivity between the layers differs. Figure 1.1 shows a sketch of a network with three hidden layers that are fully connected. Each neuron holds an associated weight and threshold. It receives an input and outputs the sum of weighted inputs. If this sum is greater than the associated threshold, the neuron gets activated.

A neural network with one or more hidden layers defines a non-linear function. It is theoretically able to represent any function. Thus, it is a *generic function approximator*. Neural networks are highly expressive and flexible. They are applied successfully in many areas. A large part of the success of neural networks can be

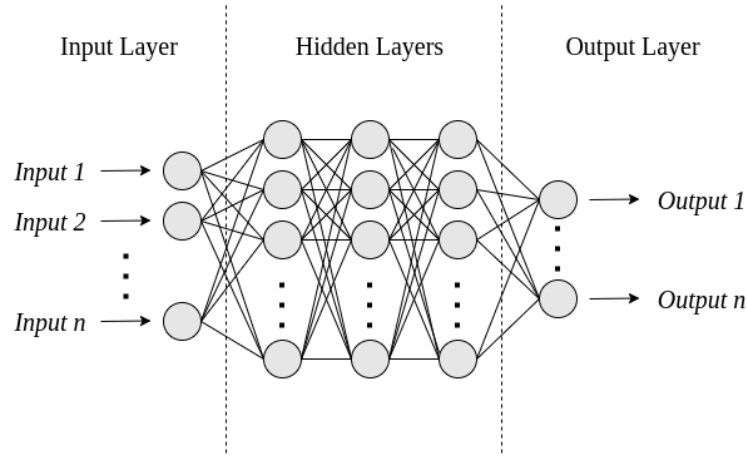


FIGURE 1.1: **Sketch of a neural network.** The image shows a sketch of a neural network with three fully connected hidden layers.

traced to the use of *backpropagation*. Backpropagation uses information from the previous epoch (i.e. iteration) to adjust the weights of the network using the error gradient. However, neural networks also come with a few limitations or disadvantages. The backpropagation algorithm requires the calculation of an accurate gradient. Depending on the problem, this can be a challenging task. Furthermore, there is a lack of understanding. The output of a neural network is often incomprehensible because of its complex structure.

Explain more, e.g. bias, generic function approximator  
Improve NN Figure

### 1.2.2 Polynomial

In mathematics, a polynomial is the sum of powers in one or more variables multiplied by coefficients. It can be used to express linear combinations. In the following, we will mainly work with polynomials consisting of vectors instead of scalars. Thus, a polynomial  $p(x)$  with  $x$  being a vector of degree  $n$  can be represented by

$$p(x) = \sum_{i=0}^n w_i^T (x_k^i)_{k \in I} \in \mathbb{R}, \quad w_i, x \in \mathbb{R}^n, \quad I = \{0, \dots, n\},$$

where  $w_i$  are constant coefficients. Polynomials are relatively simple mathematical expressions and offer some significant advantages: their derivative and indefinite integral are easy to determine and are also polynomial. Due to their simple structure, polynomials can be valuable to analyze more complex functions using polynomial approximations. *Taylor's theorem* tells us that we can locally approximate any  $k$ -times differentiable function by a polynomial of degree  $k$ . We call this approximation *Taylor polynomial*. Furthermore, the *Weierstrass approximation theorem* says that we can uniformly approximate every continuous function defined on a closed interval by a polynomial. Other applications of polynomials are *polynomial interpolation* and *polynomial splines*. Polynomial interpolation describes the problem of constructing a polynomial that passes through all given data points. Polynomial splines are piecewise polynomial functions that can also be used for spline interpolation.

Include theorems

### 1.2.3 Fourier

### 1.2.4 Bézier

## 1.3 Previous Work

### 1.3.1 Benchmarks in Reinforcement Learning

When developing a novel algorithm, it is important to compare our results with existing models. For this evaluation, we need standard benchmark problems. These are a set of standard optimization problems. OpenAI Gym <sup>1</sup> is a toolkit created for exactly this scenario. It contains a collection of benchmark problems with various levels of difficulty. However, not all benchmark problems are meaningful for the evaluation of an algorithm. If a problem is too trivial to solve, the results do not reflect the quality of the model adequately. We do not need to put a large amount of effort into the creation of a complex model for an easy-to-solve task.

In the paper *Analyzing Reinforcement Learning Benchmarks with Random Weight Guessing* (Oller, Glasmachers, and Cuccu (2020)), the authors analyze and visualize the complexity of standard RL benchmarks based on score distribution. They tested their approach on the five Classic Control benchmark problems from the OpenAI Gym interface: CartPole, Acrobot, Pendulum, MountainCar, and MountainCarContinuous. Given an RL environment, the authors conducted a fixed series of experiments. For these experiments, they used three neural network architectures ( $N_{architectures} = 3$ ): a network without any hidden layers (0 HL), a network with a single hidden layer of 4 units (1 HL, 4 HU), and a network with two hidden layers of 4 units each (2 HL, 4 HU). With these, they cover a variety of network models that are suited to solve the given tasks. The evaluation should be as objective as possible and should not include bias in the data. To achieve this, the authors did not include any learning opportunities for the network models. Instead, they chose the network weights i.i.d. from the standard normal distribution  $\mathcal{N}(0, 1)$  with Random Weight Guessing (RWG). This approach assures randomness and no directed learning. The goal of the paper was not to further analyze the network models but to investigate the benchmark problems themselves. With this in mind, they initialized  $10^4$  samples ( $N_{samples} = 10^4$ ) with different random weights. The number of samples would be too large for a reasonable learning strategy. However, the large number of samples serves a different purpose than optimizing the results. Instead, the aim is to draw statistical conclusions. Each of these samples of a neural network represents a controller that maps observations to actions in the environment. Later in this thesis, I will explore function approximators other than neural networks representing the controller. In the paper, the authors tested the controllers for each environment during 20 independent episodes ( $N_{episodes} = 20$ ). For each episode, they saved the score in the score tensor  $S$ . Algorithm 1 illustrates the procedure with pseudocode.

After the authors obtained the scores, they calculated the mean performance over all episodes from a sample and its variance. These statistics are significant insights. They can reveal how stable the network models are in completing a given task. A low mean value suggests that, in general, the network cannot complete the task. The variance gives us further insight into the score distribution. It illustrates how spread out the scores are from their respective mean score. A high value means that we have high variability. A controller is valuable if it can solve a specific task reliably and stable. Therefore, we strive for a high value for the mean and a low value for

---

<sup>1</sup>[gym.openai.com](https://gym.openai.com)

**Algorithm 1** Evaluation process taken from Oller, Glasmachers, and Cuccu (2020)

---

```

1: Initialize environment
2: Create array  $S$  of size  $N_{architectures} \times N_{samples} \times N_{episodes}$ 
3: for  $n = 1, 2, \dots, N_{samples}$  do
4:   Sample NN weights randomly from  $\mathcal{N}(0, 1)$ 
5:   for  $e = 1, 2, \dots, N_{episodes}$  do
6:     Reset the environment
7:     Run episode with NN
8:     Store accrued episode reward in  $S_{a,n,e}$ 

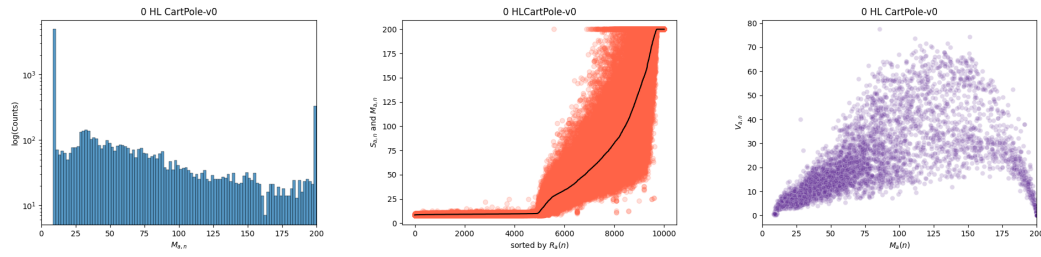
```

---

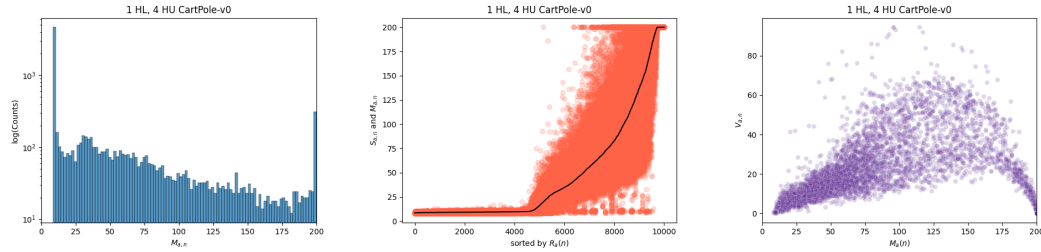
the variance. However, training a network with random weight guessing should generally not result in a stable controller. If this is the case, we can assume that the task to solve was too trivial and is not valuable for evaluation measurements. In the illustrations of the paper, the authors ranked the samples according to their mean scores. They then visualized their results with three plots: a log-scale histogram of the mean scores, a scatter plot of the sample scores over their rank, a scatter plot of score variance over the mean score.

I reproduced the results of the authors following the mentioned methodology. My findings for the environment `CartPole` are displayed in Figure 1.2. The figures illustrate the results for each of the three network architectures. Each row shows the histogram of the mean score values in the left image, the scatter plot of all scores over their rank in the image in the middle, and the scatter plot of the score variance over the mean score in the right image for a specific network architecture. There are few differences, but overall all network architectures deliver similar insights. The histogram plots show that the majority of networks receive a low score. Since the weights of the networks were chosen with RWG, this is rather unsurprising. But there is still a significant amount of networks that were able to achieve a high mean value or even the maximum value of 200. With a score of 200, the network was able to solve the task each episode. Therefore, the network could reliably solve the task without any learning technique involved. This should not be the case for a complex task. Furthermore, in the scatter plot in the middle, we can see that the line plot of the mean scores is a continuous increasing line without any jumps. Thus, a suited RL algorithm should generally be able to learn the task incrementally without converging into a local optimum. At the top of the scatter plot, we can see quite a few data points with a score of 200 that have a relatively low mean score. This indicates that a network that generally performs poorly can still solve the task with the right initialization conditions. Lastly, in the scatter plot on the right, we can see the distribution of the variance according to the mean value. On the left side, we have low scores of variance corresponding with a low mean value. These networks were consistently unable to achieve a high score. Without any training involved, we can expect most networks to be in this area. However, in the middle of the plot, the data points are spread out. For a high variance, the scores of a network differ highly from the mean value. Thus, we might get lucky and receive a high score depending on initialization conditions, but we might as well get a low score. These networks are inconsistent and unstable. On the right side of the scatter plot, we can see that the data points with a high mean value are mostly of low variance. Thus, to achieve a high mean value, the network needs consistency.

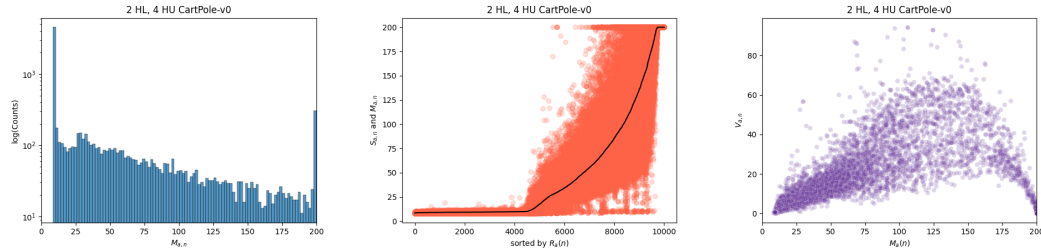
Interestingly, the usage of the bias had a relatively large impact on the performance of the network for the environment `CartPole`. Without bias, the networks seem to achieve overall better scores. The plots in Figure 1.2 illustrate the results



(A) Results of network architecture without hidden layers



(B) Results of network architecture with one hidden layer



(C) Results of network architecture with two hidden layers

**FIGURE 1.2: Results of the benchmark evaluation.** Results of the three network architectures illustrated as: (left images) a log-scale histogram of the mean scores, (middle images) a scatter plot of the sample scores over their rank, (right image) a scatter plot of score variance over the mean score

without bias. For comparison, Figure 1.3 shows the results of a network with two hidden layers with the same configurations as before but this time including bias. In the paper, the authors conducted experiments with and without bias connection for all environments. They discovered that the amount of top performers increases when dropping bias for all five environments.

Add plots from Acrobot

Adjust axis name in bias images, Change title of plots: , instead of with

Make text easier to read with padding

Make captions of figures more meaningful (why is this added? why shown like this?), refer to subplots

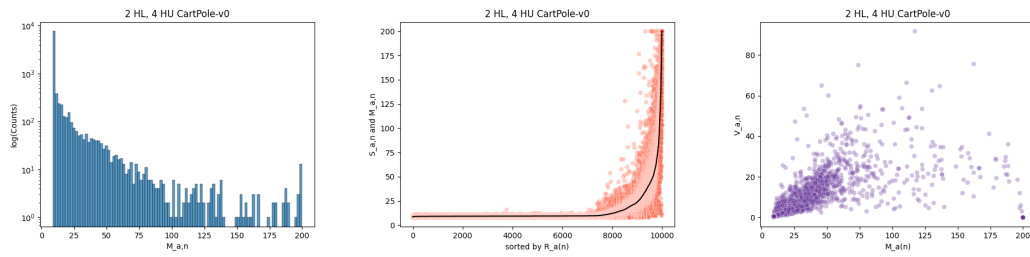


FIGURE 1.3: **Impact of bias.** The figures show the performance of a network with two hidden layers with the same settings as before but here we include bias.

## Chapter 2

# Experiments

## 2.1 Research Questions

To analyze and compare the different models, I formulated the following research questions:

1. How do function approximators other than neural networks compare with the latter?
2. Bias

## 2.2 Experiments

To investigate and answer the formulated research questions, I conducted the following experiments:

1. Similar to Oller, Glasmachers, and Cuccu (2020), I used RWG to draw the weights of the models and tested the performance on a Classic Control environment from OpenAI Gym.
2. Second experiment

In the first experiment, analogous to Oller, Glasmachers, and Cuccu (2020), there is no learning involved. For their paper, they were interested in the complexity of the environment, whereas I aim to find out more about the nature of the models. I selected a few candidates for the models, which are described in Section 2.2.2 and used them for a series of experiments. I used the same procedure for all models, which allows me to compare them with one another. For the experiments, first, I initialized the environment. I used the `CartPole` and `Acrobot` environment for these experiments. These environments are fairly easy to solve, as explained in Section 1.3.1. Therefore, we expect some controllers to solve the task even without any training. Second, I initialized the respective model. Then, I drew the model weights from the standard normal distribution  $\mathcal{N}(0, 1)$ . Each of these instances of the model represents a sample. In total, I used 10'000 samples ( $N_{\text{samples}}$ ). Finally, I ran 20 episodes ( $N_{\text{episodes}}$ ) with each sample for an environment and stored the respective score as an entry of the score tensor  $S$ . Algorithm 2 shows an overview of the described procedure.

### 2.2.1 OpenAI Gym Environments

- `CartPole`
- `Acrobot`

**Algorithm 2** First experiment with RWG

- 
- 1: Initialize environment
  - 2: Initialize model
  - 3: Create array  $S$  of size  $N_{samples} \times N_{episodes}$
  - 4: **for**  $n = 1, 2, \dots, N_{samples}$  **do**
  - 5:   Sample model weights randomly from  $\mathcal{N}(0, 1)$
  - 6:   **for**  $e = 1, 2, \dots, N_{episodes}$  **do**
  - 7:     Reset the environment
  - 8:     Run episode with model
  - 9:     Store accrued episode reward in  $S_{n,e}$
- 

Explain used environments

**2.2.2 Models**

For the polynomial model, I used two architectures  $P_1$  and  $P_2$ . The first model  $P_1$  consists of one polynomial for each possible action in a discrete action space. The input of the model is the observation from the environment. The dimension of the weight vectors is according to the dimension of the input vector. For the environment CartPole with the discrete action space  $\{0, 1\}$  and observation  $x = [x_0, x_1, x_2, x_3]^T$ , this means that  $P_1$  consists of two polynomials:

$$p_0(x) = \sum_{i=0}^n w_i^T (x_k^i)_{k \in I} \in \mathbb{R}, \quad w_i \in \mathbb{R}^4, \quad I = \{0, 1, 2, 3\}$$

$$p_1(x) = \sum_{j=0}^n w_j^T (x_k^j)_{k \in I} \in \mathbb{R}, \quad w_j \in \mathbb{R}^4, \quad I = \{0, 1, 2, 3\}$$

In the formulas,  $n$  denotes the degree of the polynomial. In the experiments, I tested polynomials with degrees 1, 2, and 3. The output of the polynomials has no reasonable upper and lower limit, as illustrated in Figure 2.1. That makes it harder to interpret the results reasonably. So, I scaled the outputs with a sigmoid function. A

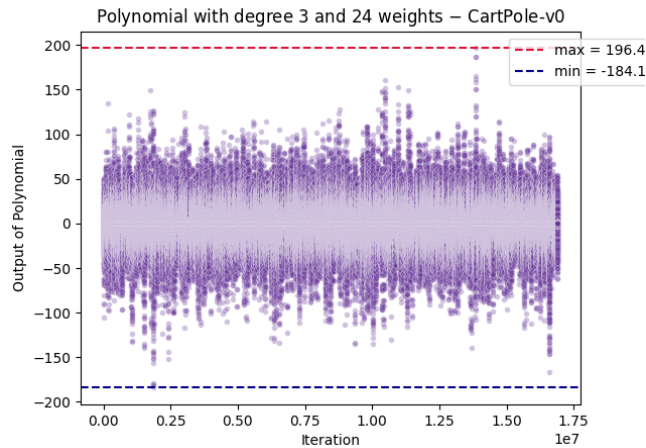


FIGURE 2.1: **Upper and lower bound.** The figure shows each output of the polynomial functions described above. As we can see, the functions are not well bound, and there are quite a few outliers. That makes it hard to interpret the output sensibly.

sigmoid function is a mathematical function that maps an arbitrary input space into



an output space with a small range, for example, 0 and 1. The function has a characteristic S-shaped curve. We can interpret the output space of the sigmoid function as a probability. In this case, we search for the probability that a specific action is the reasonable one given an observation  $x$ . Thus, for our example with the CartPole environment, we can interpret  $\text{sig}(p_0)$  as the probability that action 0 is the correct one and  $\text{sig}(p_1)$  as the probability that action 1 is the correct one. Putting this thought into a formula for  $P_1$  and the CartPole environment, we get:

$$P_1(x) = \begin{cases} 1 & \text{if } \text{sig}(p_1(x)) > \text{sig}(p_0(x)) , \\ 0 & \text{otherwise} . \end{cases}$$

For the sigmoid function, I used the logistic sigmoid function. The formula and a plot of the function are shown in Figure 2.2.

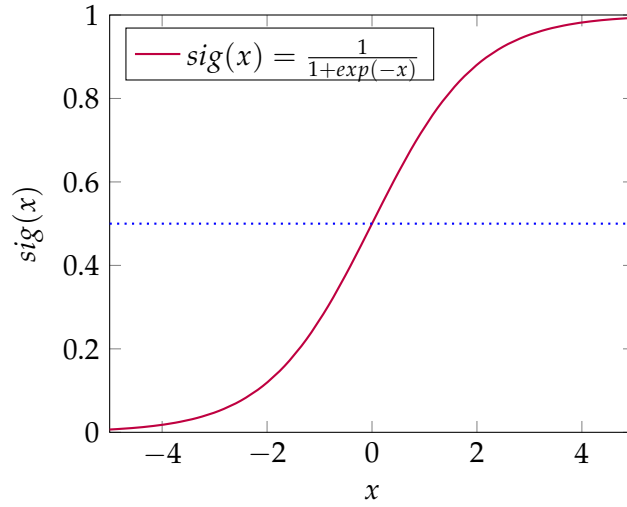


FIGURE 2.2: **Sigmoid function.** The figure shows a plot of the logistic sigmoid function. The function maps an arbitrary input space into the range between 0 and 1. The output of the function can be interpreted as a probability. It is useful to scale data into a meaningful value.

The second model  $P_2$  is constructed similarly to  $P_1$ , but it only consists of one polynomial instead of one for each possible action. For the CartPole environment, this means  $P_2$  consists of:

$$p(x) = \sum_{i=0}^n w_i^T (x_k^i)_{k \in I} \in \mathbb{R}, \quad w_i \in \mathbb{R}^4, \quad I = \{0, 1, 2, 3\}$$

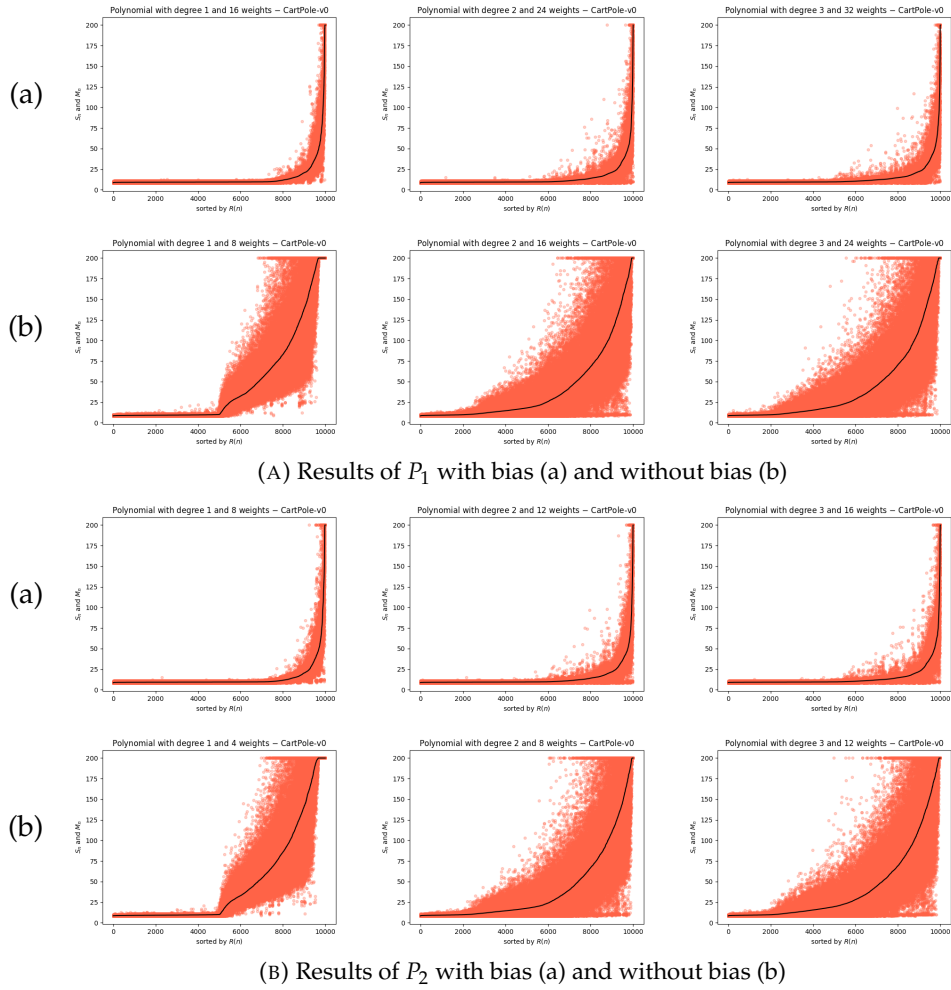
Analogous to  $P_1$ , I tested the polynomial  $p(x)$  with degree 1, 2, and 3. So,  $n \in \{1, 2, 3\}$ . In addition, I again used the logistic sigmoid function to scale the output of the polynomial. However, the output of the  $P_2$  is determined by a fixed threshold instead of comparing multiple polynomials. Putting this into a formula for the CartPole environment, we get:

$$P_2(x) = \begin{cases} 1 & \text{if } \text{sig}(p(x)) > 0.5 , \\ 0 & \text{otherwise} . \end{cases}$$

## 2.3 Results

### 2.3.1 Experiment 1

Figure 2.3 shows the results of the first experiment for the two architectures of the polynomial model  $P_1$  and  $P_2$ . I visualized the results of each model working with a bias in row (a) and without bias in row (b). The structure of the plots is identical for better comparison. The samples are ranked according to their mean score and aligned on the  $x$ -axis according to their rank. The scatter plots show all scores of the samples, whereas the lineplot illustrates the mean of each sample over all episodes. Subfigure 2.3a shows the results of the polynomial model  $P_1$  and Subfigure 2.3b shows the results of the polynomial model  $P_2$ . As we can see in the images, the plots



**FIGURE 2.3: Results of experiment 1.** The figures show the results of the first experiment with the two polynomial models. On the  $x$ -axis, we have the rank of each sample. On the  $y$ -axis, we have the scores of all samples and the mean as a lineplot. Each subplot shows the performance of the model with bias in row (a) and without bias in row (b). As we can see, both models perform equally good or bad. However, there is a huge difference in performance whether we are working with or without bias. In addition, we can see a difference in the slope of the mean values between polynomials with degree 1 and one with a higher degree.

of the two models almost look identical. Both models performed equally good or

bad despite their different architecture and the different number of weights. Because of this similarity, I will not go into each model independently but instead discuss further results for both of them.

Looking at the linear model without bias, we can see a striking resemblance to the performance of the neural network previously shown in Section 1.3.1. Looking further at the linear model, we can see that the curve of the mean score stays low until around 5'000 but then goes up relatively steeply. That means that the linear model fails around 50% miserably. However, after that, we have a high probability to achieve a good score or even solve the task entirely during multiple episodes. There are also quite a few samples that could solve the task each time, indicated by the short straight black line at the top of the plot. Looking at the polynomials with degree 2, we can see that the scores increase already at around 2'000, but the slope is less steep than for the linear model. In addition, there are fewer samples that could solve the task for each episode than there are for the linear model. If we look at the results of the polynomials with degree three, we can see that there is only a small boost compared to the polynomials of degree 2. The scores are overall slightly higher, but the slope is very similar to before. There is only a little difference between the two models even though the weights are doubled for  $P_1$  and a half more for  $P_2$ . The large difference lies between the polynomials with degree 1 and polynomials with degree 2, respectively, degree 3. It seems that the complexity of the model depends more on the architecture of the model than on the number of weights.

In conclusion, with the linear model, we have a 50% chance of failing but the probability of actually solving the task for all episodes is higher than for the polynomials with a higher degree. That means that with the linear model, we have a larger fraction of samples that can solve the environment independent of the initialization conditions. At first glance, we could assume that this is an important aspect for such a model and choose the polynomial with degree 1 over one with a higher degree. However, we should remind ourselves that these experiments are rather unusual for an application since there is no learning involved and the number of samples is huge. In a common application, we would use some kind of training and want the model to sequentially improve its performance. Considering this aspect, when using a learning algorithm, we would not prefer the polynomials with degree 1 as we might get stuck in a fitness plateau when the algorithm has no method of dealing with this behavior.

Another observation we can make from Figure 2.3 is that the bias influences the performance of the model significantly. We already saw this behavior with the neural network in Section 1.3.1. Thus, the influence of the bias is not specific to neural networks but seems to be more of a general factor.

Change title of plots: , instead of with,  $P_1/P_2$  instead of Polynomial



## **Chapter 3**

# **Conclusion**

### **3.1 Conclusion**

In this work we...

### **3.2 Future Work**

The continuation of this work includes...



# Bibliography

Oller, Declan, Tobias Glasmachers, and Giuseppe Cuccu (Apr. 16, 2020). “Analyzing Reinforcement Learning Benchmarks with Random Weight Guessing”. In: *arXiv:2004.07707 [cs, stat]*. arXiv: 2004.07707. URL: <http://arxiv.org/abs/2004.07707> (visited on 12/07/2021).