Estimator complexity in Deep Reinforcement Learning

Carla Crivoi, Andrei Hodoroaga and Florin Radu Gogianu *
University of Bucharest, Romania
*Bitdefender, Romania

crivoicarla02@gmail.com, andreihodoroaga36@gmail.com, florin.gogianu@bitdefender.com

1. Introduction

The project's goal is to determine the effect of different objective functions used in RL on the complexity of the neural networks they are trained on.

Challenges:

1. isolating the possible effects of each algorithm on the neural network
2. determining the proper metrics to evaluate the network on

2. Dataset and Preprocessing

Dataset Description:
The dataset contains 10,000 samples from a **Grid World** environment, a rectangular grid with a start and end state, and (optionally) some walls in between.
The samples come from complete trajectories of an agent following a random policy (having equal probability to move to every possible next state).
One sample of the dataset is a tuple of the form $(\{x, y\}, z)$, where $\{x, y\}$ are the coordinates on the grid and z is the associated label. z differs between the algorithms:

- for the regression task it is the value function for the state $\{x, y\}$
- for Monte Carlo and TD(0) it is the respective objective function at the state $\{x, y\}$

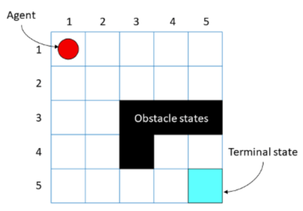


Figure 1: Example of a Grid World

Data preprocessing: The only preprocessing step implemented is normalizing the state coordinates between $(-0.99, 0.99)$.

3. Objectives

1. **Regression to the true value function** - We evaluate a random policy to convergence and then train the neural networks to approximate the true value function.
2. **Monte Carlo prediction** - We train the neural networks using the returns obtained from complete episodes for each state.
3. **Semi-gradient TD(0)** - We train the neural networks using the TD target: $R_{t+1} + \gamma\hat{v}(S_{t+1}, w)$.

4. Models

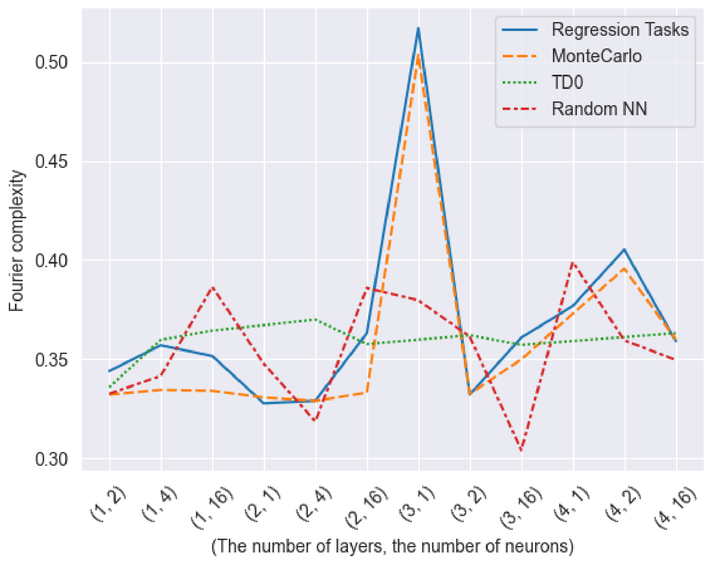
We trained the algorithms on several different architectures: with 1, 2, 3, and 4 hidden layers, and each layer with 2, 4 or 16 neurons.

5. Results

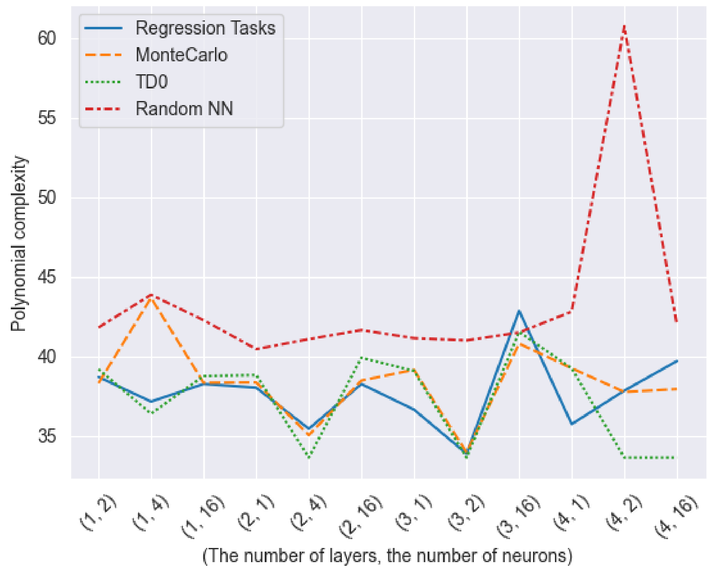
Training the models:

1. **Regression task and Monte Carlo:** We trained the networks until the loss was reduced to less than 0.0001 or until there was no improvement in the loss for 200 consecutive epochs.
2. **TD0 algorithm:** Each network was trained for around 5 epochs.

Fourier complexity:

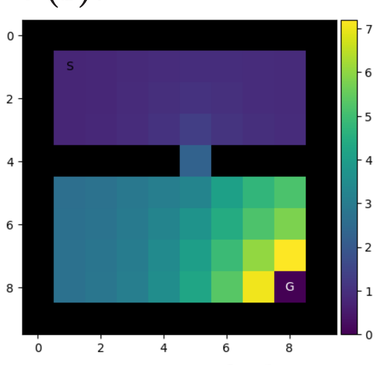


Polynomial complexity:

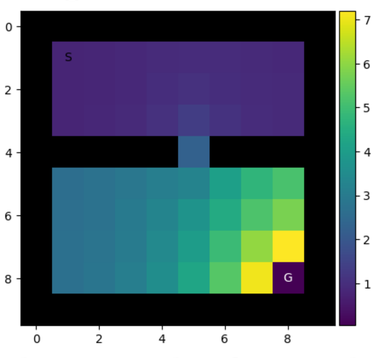


Visual representation of predictions:

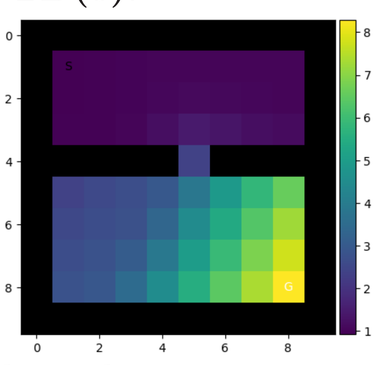
V(s):



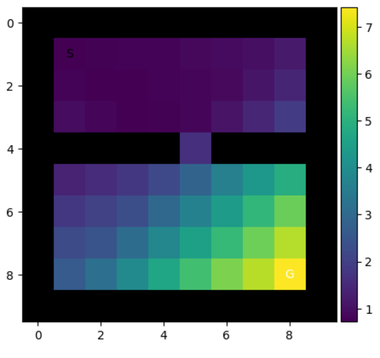
Regression:



TD(0):

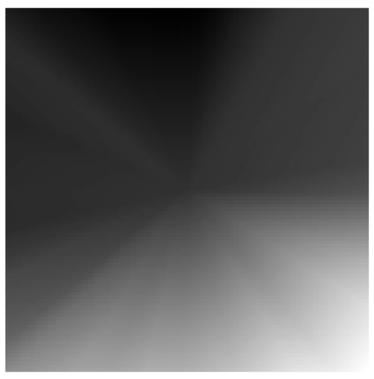


Monte Carlo:



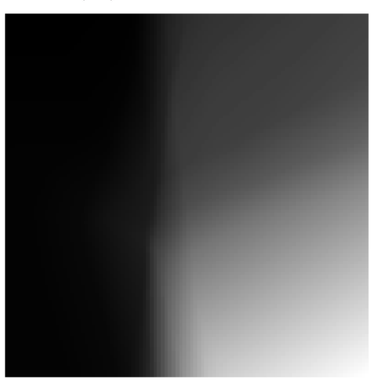
Heatmaps of the neural networks for each algorithm:

Untrained:

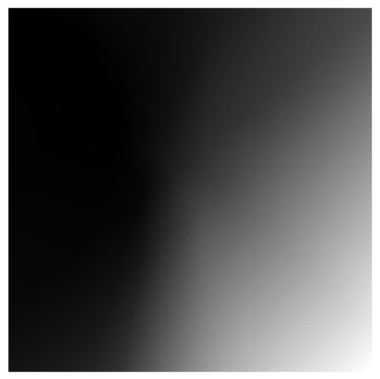


Regression:

TD(0):



Monte Carlo:



6. Network complexity metrics

Fourier complexity: The spectral or frequency complexity of a function on a 2D surface is evaluated, using Fourier transforms. This is done by analyzing how the function values vary on a grid defined in a 2D space.
$\mathrm{f(k)} = \sum_{x \in X_{grid}} \omega_x^{\top k} f(x)$
Polynomial complexity: We evaluate how a function behaves over a 2D domain by approximating and analyzing it with Chebyshev polynomials. This can be useful in mathematical and numerical analysis to understand characteristics such as the degree of variation or smoothness of a function over a given domain.
$\mathrm{CChebyshev(f)} = \sum_{n_1,n_2=0}^{N} |c_{n_1 n_2}| \cdot \|[n_1, n_2]\|^2 \frac{}{\sum_{n_1,n_2=0}^{N} |c_{n_1,n_2}|}$

7. Limitations and future work

- Results based on only one initialization for each model
- Metrics limited to two types of complexity metrics (Chebyshev and Fourier), will add compressibility in the future
- TD(0) experiments should be revisited, we suspect under-training
- Only one MDP was used in the comparison, we should compare different MDP classes