Industrial Benchmark for Fuzzy Particle Swarm Reinforcement Learning

Felix Linker

Leipzig University, 04109 Leipzig, Germany

Abstract. This paper summarizes the results of applying the industrial benchmark to the approach of fuzzy particle swarm reinforcement learning (FPSRL) as proposed in [3].

1 Introduction

The FPSRL approach tries to solve the problem of finding control policies to real world problems by machine learning. We recommend to read the paper for a detailed explanation but in short, the approach takes the following steps:

1. Gather real world data of the e.g. industrial plant you want to generate a control strategy for by just measuring its performance
2. Generalize this data to a world model turning the data gathered into a reward function
3. Use particle swarm optimization to find an optimal policy regarding this generalized world model
4. If the performance in the generalized model of the policy is sufficient, evaluate the policy in the real world

We implemented this approach to benchmark it with the Industrial Benchmark[1] [2]. The Industrial Benchmark tries to give a benchmarking environment for real world industrial plants. It generates output from a high-dimensional and hidden state-space, only a part of which is observable. In this repository we used the Industrial Benchmark to generate data to create a world model, thus simulating real world runs of some industrial plant and evaluate generated policies.

By doing so, we implemented an benchmarking approach that was proposed in [4] for an algorithm that implements the same approach as FPSRL but uses genetic programming instead of particle swarm optimization to search for a policy.

[1] `https://github.com/siemens/industrialbenchmark`

2 Results

2.1 World Model

The industrial benchmark has an observable state of 4 dimensions. When you apply a 3-dimensional action vector to it, the internal state changes which in almost all cases also results in a different observable state. The benchmark also yields two performance indicators: a value for fuel costs and consumption costs, latter of which indicate a metric comparable to wear.

We used two neural networks to predict each of these output variables in time series predicion by implementing an approach as described in [1] and [5] which in turn are referenced by [2]. The networks were trained on a dataset of which 30% have been used for validation. The world models mean absolute errors on a time series were as follows:

Fuel model	Consumption model
6.11449	18.87096

It is to no surprise that the prediction of the consumption costs is worse than the prediction of fuel costs as in [5] the authors noted that it took more past observations of the state space to accurately predict the consumption costs as opposed to predicting fuel costs.

In figure 1 and 2 we appended some figures that illustrate time series predictions.

2.2 Policy

A policy then was generated by a particle swarm in ring topology. In total, our policy had 84 weights thus leading to a 84-dimensional space that was traversed by 25 particles over 250 iterations. We found that quite quickly, after about 100 iterations, the best solution would have been found and further iterations wouldn't increase the solution's quality. We also found that using more particles didn't improve results.

We used the almost the same methods as proposed in [4] to evaluate a policy with the only difference being that our performance was a cost and positive value leading to a minimization problem whereas the original performance was measured in negative rewards leading to a maximization problem. We evaluated our generated policy against the mean performance of 20 randomly generated policies with the following results:

Policy performance	Mean random policy performance
1866.5951	6215.69734

Although the authors in [4] didn't give all parameters to their reward function, they considered a successful policy to have costs lower or equal to 165 which is far from close to our results. An explanation for this would be that the approach of geneticly programming algorithms as a policy could utilize the whole

past observations history of states whereas the FPSRL policy only inspects the last state. Inspecting the whole history of observed states in this case might lead to a policy-state-space one magnitude bigger than the current one - depending on the implementation.

The authors in [4] were able to tackle this problem by limiting the complexity of the algorithms. The algorithms could choose a number of parameters from past oberservations to calculate their output but as the complexity of the algorithms was limited, they couldn't use *all*. This allowed them to use long histories of past oberservations whilst not introducing huge vectors to weigh all those inputs.

In future, it might be worthwhile to investigate how such an optimization could be applied to the FPSRL approach.

3 Implementation

All code was written in python and can be found alongside example results and configuration files here: `https://github.com/felixlinker/IB_FPSRL`. We used keras[2] with the tensorflow[3] backend to implement the world model generation. We used pyswarms [6] to implement the particle swarm optimization.

References

1. Duell, S., Udluft, S., Sterzing, V.: Solving Partially Observable Reinforcement Learning Problems with Recurrent Neural Networks, pp. 709–733. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
2. Hein, D., Depeweg, S., Tokic, M., Udluft, S., Hentschel, A., Runkler, T.A., Sterzing, V.: A benchmark environment motivated by industrial control problems. CoRR **abs/1709.09480** (2017). https://doi.org/10.1109/SSCI.2017.8280935
3. Hein, D., Hentschel, A., Runkler, T., Udluft, S.: Particle swarm optimization for generating interpretable fuzzy reinforcement learning policies. Engineering Applications of Artificial Intelligence **65**, 87 – 98 (2017). https://doi.org/10.1016/j.engappai.2017.07.005
4. Hein, D., Udluft, S., Runkler, T.A.: Interpretable policies for reinforcement learning by genetic programming. Engineering Applications of Artificial Intelligence **76**, 158 – 169 (2018). https://doi.org/10.1016/j.engappai.2018.09.007
5. Hein, D., Udluft, S., Tokic, M., Hentschel, A., Runkler, T.A., Sterzing, V.: Batch reinforcement learning on the industrial benchmark: First experiences. CoRR **abs/1705.07262** (2017). https://doi.org/10.1109/IJCNN.2017.7966389
6. Miranda, L.J.V.: PySwarms, a research-toolkit for Particle Swarm Optimization in Python. Journal of Open Source Software **3** (2018). https://doi.org/10.21105/joss.00433

[2] `https://keras.io/`
[3] `https://www.tensorflow.org/`

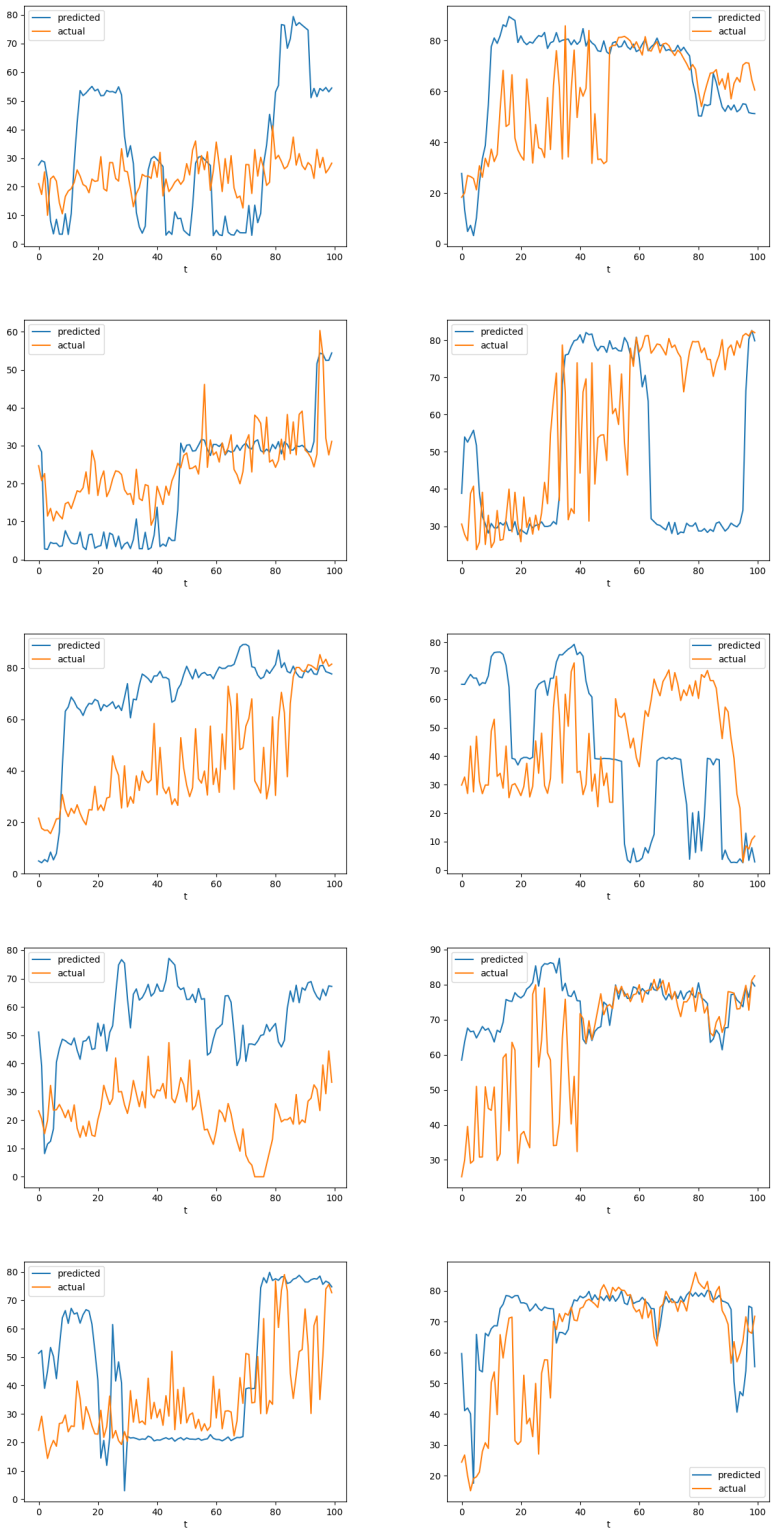

Fig. 1: Example time series predictions of the fuel models

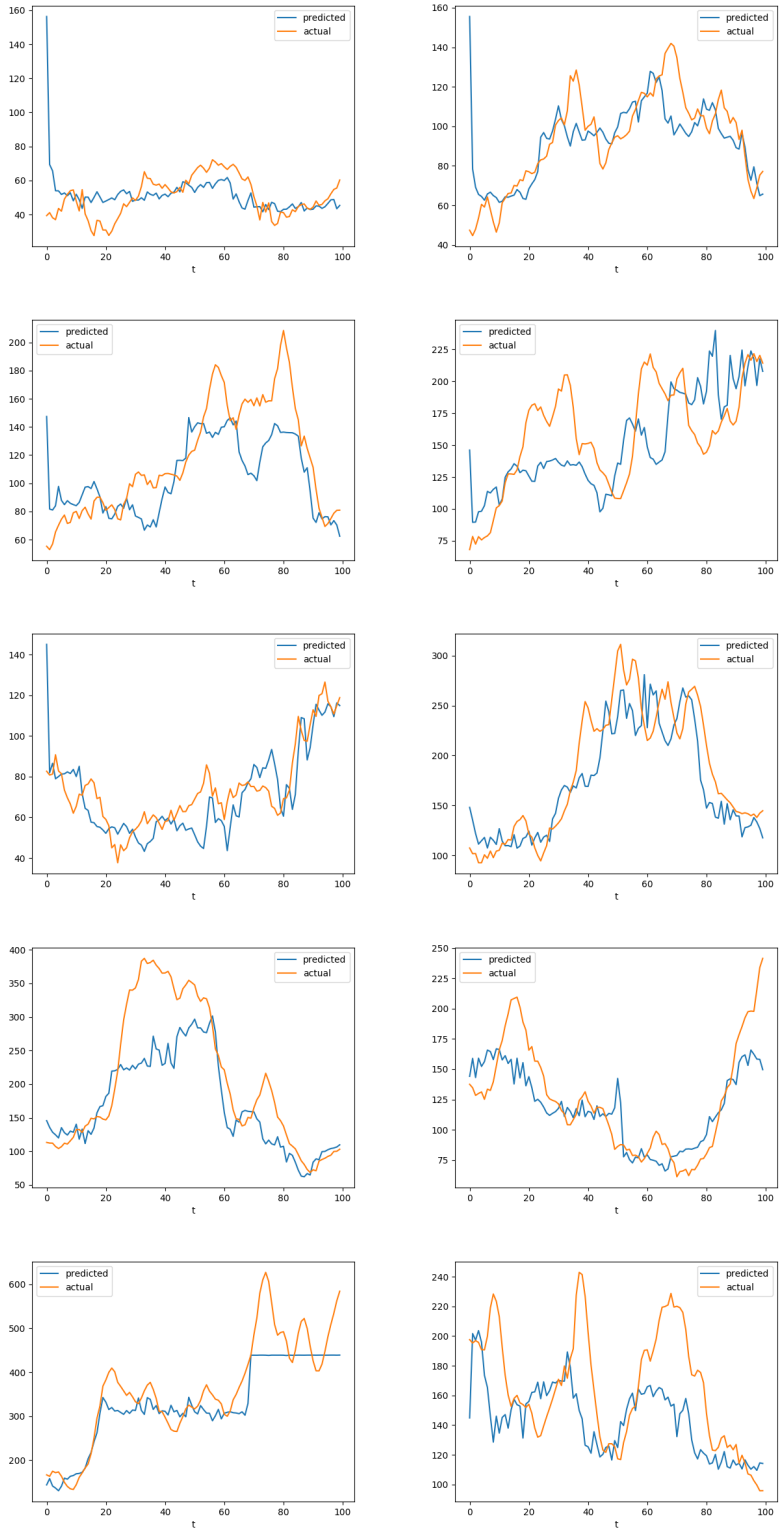

Fig. 2: Example time series predictions of the consumption models