TensorFlow Implementation of Recurrent Control Neural Network for the Mountain Car Problem

Ruoxia Qi ruoxia.qi@campus.lmu.de, 11225370

Jingcheng Wu jingcheng.wu@campus.lmu.de, 12215745

Yiwei Li yiwei.li@campus.lmu.de, 12122057

1. Introduction

Reinforcement Learning (RL) describes the problem, where an agent changes and adapts its action to the environment to maximize the reward. One difficulty in solving the problem is that action and state spaces are always not discrete in the real life (e.g., driverless cars). Schäfer et al. (2007) provide a novel model-based approach, the Recurrent Control Neural Network (RCNN), to deal with high dimensional and continous RL problems. The RCNN model can be seen as a RNN extended by a controller: the RNN ist used to simulate the dynamics, since it is a well-known powerful structure for modeling sequence data. The controller built upon is trained on the task of learning an optimal policy. Moreover, it can be adapted to solve multiple RL problems by adapting output clusters regarding action and reward.

Mountain Car, introduced by Moore (1990), is a classic RL problem. In this problem, an under-powered car is released near the bottom of an U-form terrain with the goal of driving up to the top. However, the engine of the car is not strong enough to cover the gravity and to directly drive out from the environment that the car needs to learn to accelerate reversely to leverage potential energy.

In this report, we present the application of RCNN to the mountain car problem and focus on the the following research questions:

1. How well is the dynamics of the mountain car problem simulated?
2. Is the car able to reach the top of the mountain, following the policy learned by the RCNN?
3. How well does a successful policy perform?
4. How dose the amount of data impact the model performance?

2. Dataset

We use a modified gym environment[1] for our experiments. The car has a continuous state consisting of its position $o \in [-1.2, 0.6]$ and velocity $v \in [-0.07, 0.07]$, and three possible

1. https://gym.openai.com/envs/MountainCar-v0/

Submitted: 14.03.2021

actions $a \in \{0, 1, 2\}$, where 0 indicates accelerating to the left, 1 stopping accelerating and 2 accelerating to the right. The car terminates when $o \geq 0.5 \wedge v \geq 0$, or the length of episode ≥ 200. The reward is 0 for each step and 1 when the car has reached the top of the mountain.

Our dataset is collected by running the modified gym simulation for 200 episodes. In each episode, the car starts with an initial state drawn from an uniform distribution over [-1.2, 0.6] and travels with a random action. Table 1 provides an overview of our dataset. Dominant episodes (190) do not terminate until 200 steps are reached. The rest 10 stop earlier, since the car is located near the goal position initially. Besides, episodes with 200 steps cover the entire range of the car positions (Figure 1). Thus, we conclude that our dataset covers all possible trajectory patterns the model should learn.

We split our dataset 7:1:2 into training, validation and test sets randomly, i.e., keeping the same distribution of initial positions of each episode in all three subsets. The validation set is used for hyperparameter optimization and early stopping, whereas the test set is only used to evaluate the final model performance reported in section 5.1.

steps	# episode	initial position
1	4	0.515 - 0.6
2	1	0.53
4	3	0.521 - 0.579
5	1	0.556
11	1	0.558
200	190	-1.192 - 0.5

Table 1: Dataset statistics

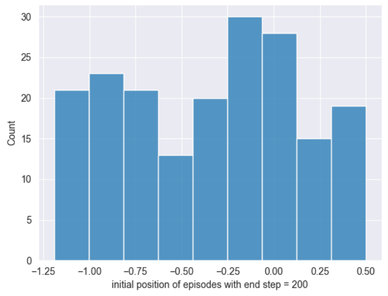


Figure 1: Initial position distribution

3. Methods

The RCNN approach based on Schäfer et al. (2007) and Schäfer (2008) is a 2-stage procedure. The first stage focuses on the dynamic simulation and is implemented with an RNN (blue part in Figure 2). It takes a certain size (window size, w) of activities in the past as input and predicts the next state. The forward computation can be formalized as:

$$
\begin{aligned}
p_t &= As_{t-1} + Bx_t & (1)\\
s_t &= tanh(p_t + Da_t + \theta) & (2)\\
\hat{x}_{t+1} &= Cs_t + \tau, & (3)
\end{aligned}
$$

where $x_t \in \mathbb{R}^2$ consists of o_t and v_t at time step t, $a_t \in \{0, 1, 2\}$ indicates the action carried out at time step t and $\hat{x}_{t+1}$ is the model's prediction, i.e., the state at time step $t + 1$. p_t and s_t indicate hidden states of the RNN, weight matrices A, B, C, D and bias θ and τ are weights to be learned. I stands for the identity matrix. Thus, RNN follows the supervised learning strategy and the objective is to minimize $\|\hat{x}_{t+1} - x_{t+1}\|^2$.

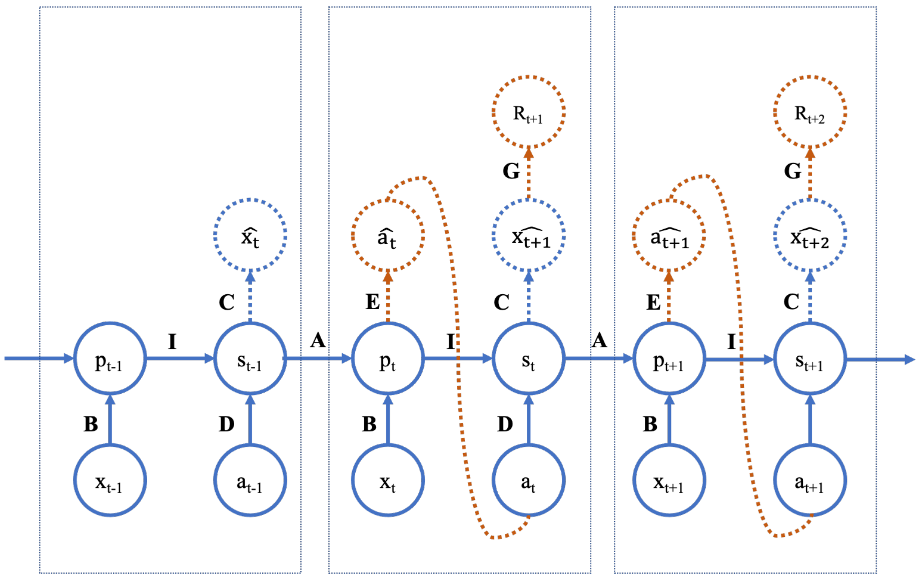


Figure 2: RCNN based on Schäfer et al. (2007). The first stage is marked in blue, where the model is reduced to a simple RNN and trained for simulating dynamics. The orange part is used in the second stage and works as a controller, learning the optimal action to maximize the reward. Dotted connections indicate predictions. The rectangle frame marks components of a single time step. Biases are omitted.

In the second stage, we extend the RNN to RCNN by integrating a controller, which is trained to predict the optimal action at each time step, i.e., the weights learned in the first stage get fixed. The output is the action at time step t calculated by:

$$\hat{a}_t = tanh(Ep_t + b), \tag{4}$$

where p_t is the hidden state computed by the previous RNN. Matrix E and bias b are weights to be trained. The objective of RCNN is to maximize the reward, which depends merely on the position of the car and is defined as

$$R_t = \frac{1}{1 + \exp -10 * (o_t - 0.5)}. \tag{5}$$

We observe that Equation 5 is a steep logistic function and is approximately 1 for $o > 0.5$ and 0 otherwise (see Figure 3), which corresponds to our problem setting. Besides, $G = [1 \quad 0]$ is a known matrix, because R is not impacted by velocity.

4. Training

Our experiments include 2 models, RNN and RCNN, corresponding to the 2 stages introduced in section 3, and are implemented in TensorFlow[2].

The input of the RNN at a single time step is a triple (o_t, v_t, a_t), i.e., for a window size w, the input is a two dimensional array in the shape of $(w, 3)$. We map Equation 1 and 2

2. https://www.tensorflow.org

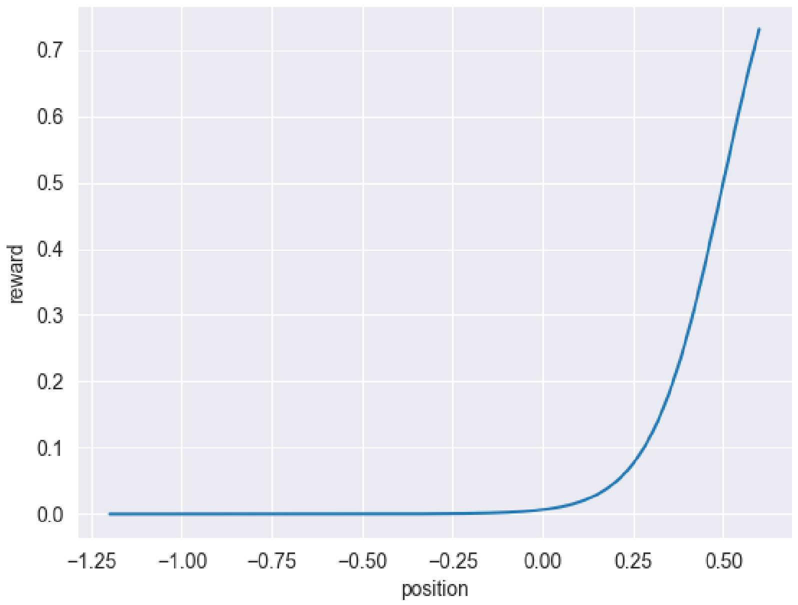


Figure 3: Position and reward relationship based on Equation 5.

to an RNN cell with two hidden states to ensure that $x_t = (o_t, v_t)$ share the weight B and a_t has its own weight matrix D. Two additional layers are built upon the RNN layer with the customized cell: one dropout layer with the dropout rate $= 0.001$ and one dense layer implementing Equation 3.

Since RCNN and RNN share the input weights, their input should have the same shape as well. However, as we want to train a controller in the second stage, the model should not see the action a_t. We obstacle this problem by keeping a_t in the input but unused: since we have copied all the weight matrices from RNN, we can compute Equation 4, replace a_t by the predicted one $\hat{a}_t$ and calculate R. For the convenience of the evaluation, we also let RCNN output $\hat{x}_{t+1}$ and $\hat{a}_{t+1}$, so that the car is able to move forward automatically. This will be detailed in section 5.2.

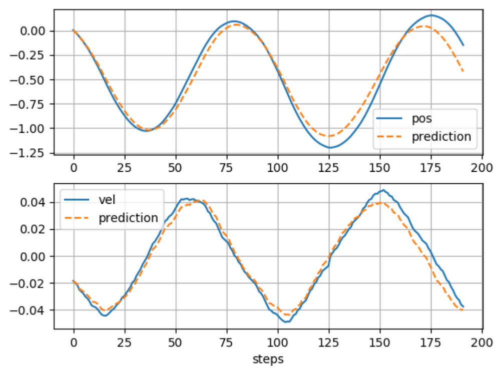
We use the Adam optimizer with an initial learning rate of 10^{-3} and early stopping with a patience of 20 for all models. The learning rate is reduced automatically with a factor of 0.1, if the validation loss does not improve in 10 epochs. We train each model for 10 times.

5. Results

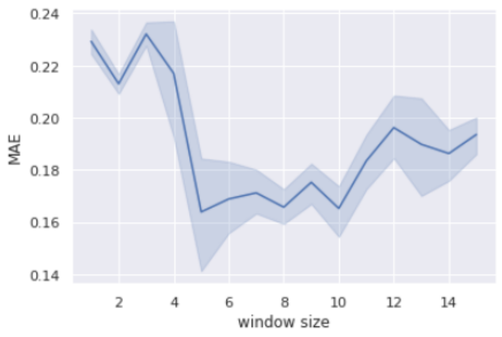
5.1 Dynamics Simulation

We evaluate RNN on episodes in the test set with total steps larger than $w + 10$ in an autoregressive way: for each episode, we start with the first w time steps and feed the model's output back to itself so that the predictions are conditioned on the previous one. We compute the MAE score as an objective measure and plot the true and predicted dynamics for an intuitive understanding. For instance, the orange dashed line in Figure 4a presents the episode with the lowest MAE generated by the best RNN with $w = 8$. The predicted dynamics matches perfectly with the true one.

To decide on an appropriate window size, we analyse the relationship between window size and MAE. Figure 4b presents the results. We observe that models with $w \in [5, 10]$ perform similarly and obviously better than models with other window sizes. Since there's no significant difference among window size in the range of 5 to 10, we further analyse this question based on the performance of RCNN, which will be discussed in the next section.



(a)



(b)

Figure 4: (a) The episode with the lowest MAE generated by the best RNN with $w = 8$. (b) The MAE evaluated on RNN with increasing window size.

5.2 Policy Optimization

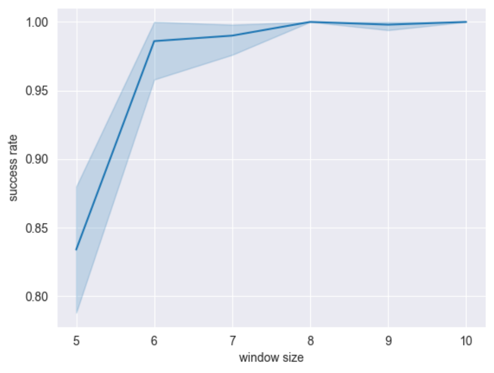
After training RNN, we construct the RCNN with the learned weight matrices A, B, C, D, get them fixed and implement training with the same hyperparameters, in particular the window size and the training set size.

We evaluate RCNN autoregressively as well, but without the test set, since RCNN is trained to predict optimal actions to make the car to the goal. In another word, we merely generate the first window with a default gym initial position and random actions, and the car travels further automatically following the action predicted by the model. For each model, we repeat this evaluation process for 50 episodes. Note that the initial position here is in $[-0.6, -0.4]$, i.e., near the bottom of the mountain, differing from the initial position used in sampling data ($[-1.2, 0.6]$), since we want to ensure that we can observe the car's behaviour over a relatively long period of time. We take two metrics into account:

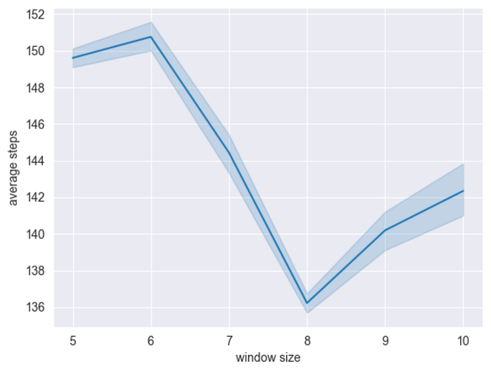
1. success rate: in how many episodes among 50 episodes is the goal position reached?
2. average steps: how many steps are needed in successful cases?

We first try to decide on the window size by training RCNN with $w \in [5, 10]$ based on the best RNN with the same w trained in the previous stage. The results are presented in

Figure 5. Models with $w = 8$ clearly outperform others: they all have the success rate of 1 with the lowest average steps.



(a)



(b)

Figure 5: The success rate (a) and the average steps (b) evaluated on RCNN with increasing window size.

Furthermore, we plot the trajectory of the car generated by the best-performance RCNN in Figure 6. In all the 50 episodes, the car starts with the $o \in [-0.4, -0.6]$ and $v = 0$ and tries to accelerate to the right first. However, it is underpowered to reach the goal in a single pass that has to reverse the direction to build up the momentum. The car stops once it reaches the top of the mountain ($o \geq 0.5$ and $v \geq 0$). Therefore, we conclude that the mountain car problem is simulated and solved by RCNN. Note that we reset the velocity to 0 when the car hits the wall ($o \leq -1.2$) according to the gym environment (break on the left side in Figure 6).

We also explore the trajectories predicted by the same model starting from an arbitrary position from $[-1.2, 0.6]$, i.e., the same distribution as the training set. To avoid the clutter, we divide the entire range equally into 6 subsets and generate 10 episodes each. The results are presented in Figure 7. Again, the car reaches the goal in all episodes. With a start position < -0.73, i.e., from relatively high on the left side, the car is possible to reach the goal in a single pass. Otherwise it needs to drive back to gain the momentum. It implies that the dynamics has been learned sufficiently.

Following Riedmiller (2005), we further explore whether we can obtain a satisfying model with a smaller training set size. We sample observations using the same method as described in 2 with episodes $\in \{50, 100, 150, 200, 250, 300, 350, 400, 450, 500\}$ and implement stage 1 and 2 in section 3 consecutively, i.e., without evaluating and selecting the best RNN. We observe that the results seem unstable after training 10 times (Figure 8a and 8b). Particularly, the success rate of training size = 150 and 200 has a wide confidence interval. Hence, we implement another 10 runs and plot the results across 20 runs in Figure 8c and 8d. Unfortunately, it does not help much. It seems that the model trained on only 50 episodes observations outperforms all other models. It can find the successful policy for all 50 test episodes and also has the lowest average steps. The success rate decreases when the

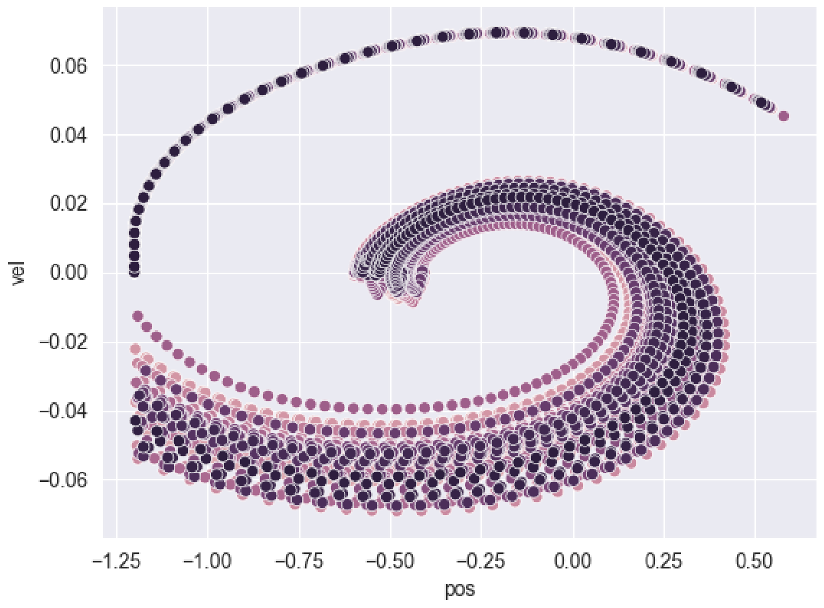


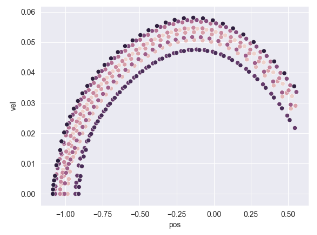
Figure 6: Trajectories of the car generated by RCNN with the lowest average steps.

training set size is $\in [150, 200, 250]$ and increases again when the the training set sampled over 300 episodes.

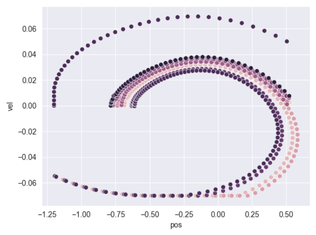
To analyse the reason, we plot the trajectories of 50 episodes for each setting in Figure 9. We find out that the task of simulating dynamics is not sufficiently fulfilled. An typical example is Figure 9f: in some episodes, the car is able to reach the goal from the bottom without driving back. In other episodes, the car reaches the top with a negative velocity and then reverses the direction, which is also impossible. In comparison, trajectories in Figure 9b, 9c and 9d match obviously better to the problem's dynamics. The failed trials are due to the limited iterations of 200. The car has changed its direction for 3 times to build up the momentum and we can expect that it would reverse its direction again in a few steps and drives up the mountain on the right. Therefore, evaluating models merely on success rate and average steps after a consecutive training process might be insufficient. We can confirm this hypothesis by evaluating the intermediate RNN in future work.

6. Conclusion and Future Work

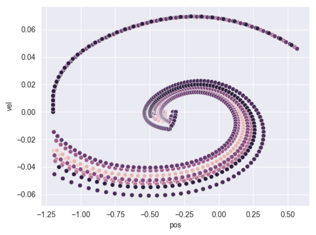
In this report, we presented our experimental setup and the results. Aiming at solving the mountain car problem, we implemented RCNN in TensorFlow. With nearly 4,000 observations, the simulator part of the model, RNN, is able to map the trajectory of the car and the extended RCNN can always predict a successful policy to get the car out of the environment. The relationship between window size and model performance is clear: the simulator has a near-optimal performance and the controller has the best performance with



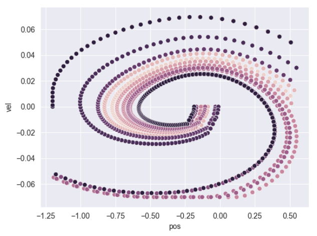
(a)



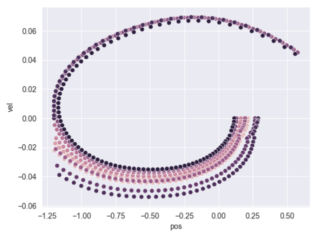
(b)



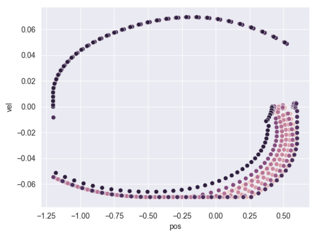
(c)



(d)



(e)



(f)

Figure 7: Trajectories starting from (a) [-1.2, -0.9), (b) [-0.9, -0.6), (c) [-0.6, -0.3), (d) [-0.3, 0), (e) [0, 0.3), (f) [0.3, 0.6).

$w = 8$. It might be due to the vanishing gradient problem of RNN when processing a long sequence.

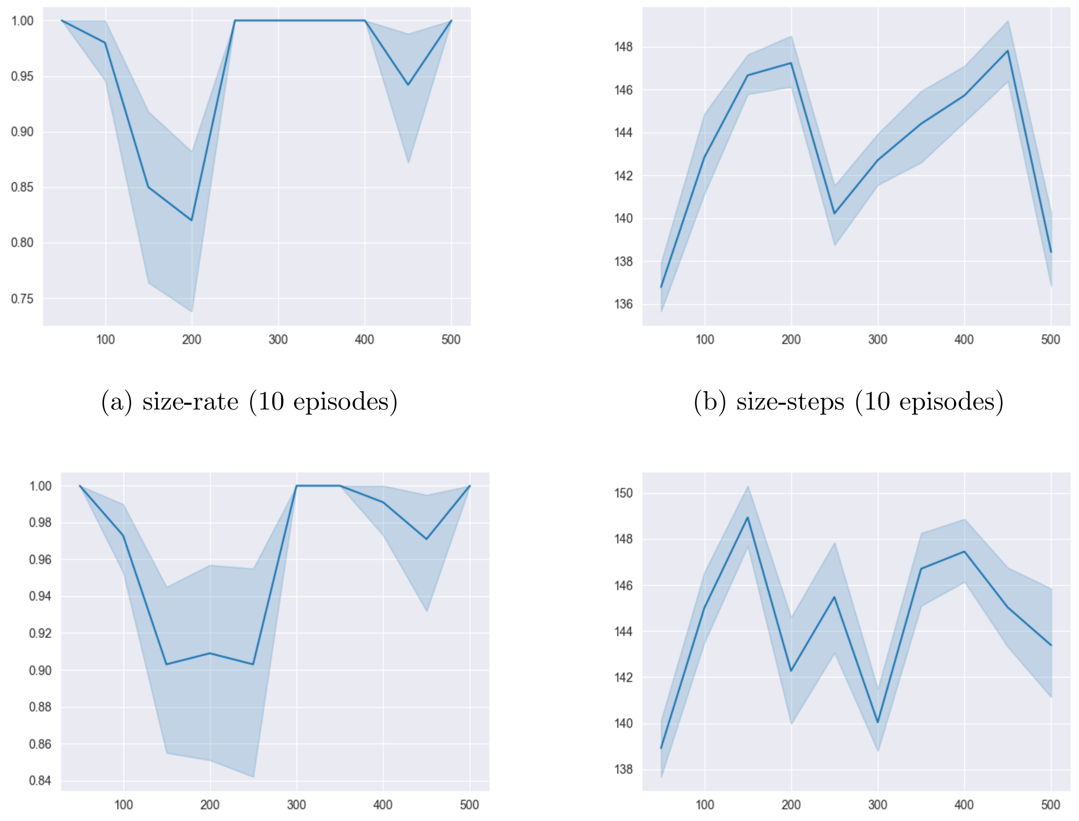
However, we cannot conclude the data efficient of the RCNN, although models trained on 50 episodes are capable of solving the problem. There's lack of evidence explaining the tendency with increasing training set size and the performance of the simulator. In future work, we plan to evaluate the intermediate RNN first to address this problem. Besides, we can compare our results to related works to have a better evaluation. We also intended to replace simple RNN by a LSTM. We could expect a better result, since LSTM can deal with a longer sequence.

References

Andrew William Moore. Efficient memory-based learning for robot control. Technical report, 1990.

Martin Riedmiller. Neural fitted q iteration – first experiences with a data efficient neural reinforcement learning method. In João Gama, Rui Camacho, Pavel B. Brazdil, Alípio Mário Jorge, and Luís Torgo, editors, *Machine Learning: ECML 2005*, pages 317–328, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-31692-3.

Anton Maximilian Schäfer. *Reinforcement learning with recurrent neural networks*. PhD thesis, University of Osnabrück, Germany, 2008. URL `http://elib.ub.uni-osnabrueck.de/publications/diss/html/E-Diss839_HTML.html`.



(a) size-rate (10 episodes)

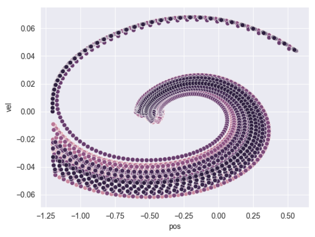
(b) size-steps (10 episodes)

(c) size-rate (20 episodes)

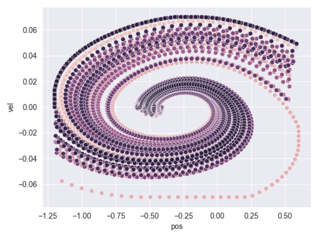
(d) size-steps (20 episodes)

Figure 8: The success rate and the average steps evaluated on RCNN with increasing training set size. The shaded areas indicate confidence intervals across 10 runs (a, b) and 20 runs (c, d).

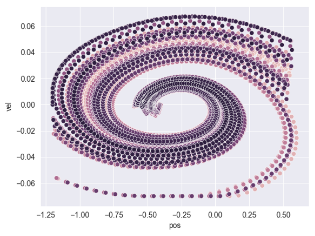
Anton Schäfer, Steffen Udluft, and Hans Zimmermann. The recurrent control neural network. In *ESANN 2007 Proceedings - 15th European Symposium on Artificial Neural Networks*, pages 319–324, 01 2007.



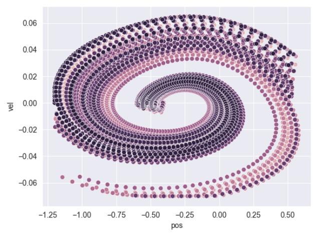
(a) 50



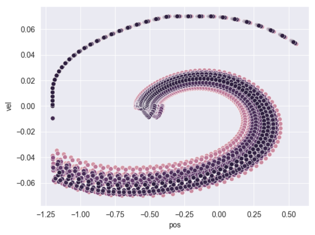
(b) 100



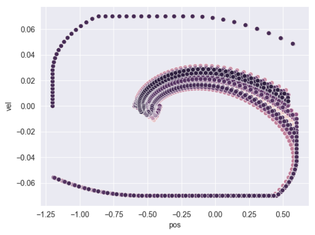
(c) 150



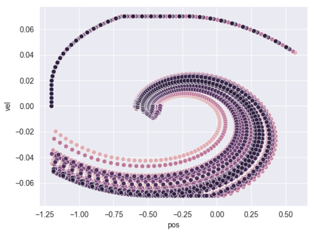
(d) 200



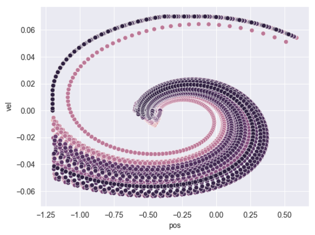
(e) 250



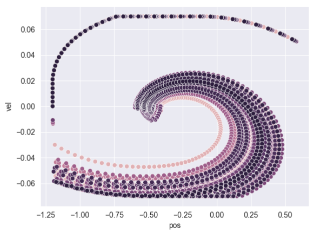
(f) 300



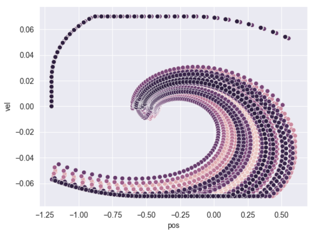
(g) 350



(h) 400



(i) 450



(j) 500

Figure 9: Trajectories generated by RCNN trained on 50 - 500 episodes samples.