Duckietown - Reinforcement Learning based on World Models
Duckietown - megerősítéses tanulás a World Models alapján

conDUCKtors
Máté Büki, Péter Zoltán Sánta, Gábor Tamás
Deep Learning in Practice with Python and LUA
VITMAV45
Budapest University of Technology and Economics
2019/20/1

Abstract

In our homework, we use a reinforcement learning framework to teach an agent to be able to follow the lane in the Duckietown Gym environment. The model our work is based on was named World Models (1) and was published in 2018. The main idea of the model is to use a compressed representation of the observable environment instead of the raw image seen by the agent. The transformed observations are then passed to a recurrent neural network, which is responsible for the prediction of the next observation. Finally, the agent is trained using reinforcement learning. We have modified the model to use the so called SAC policy instead of a simple linear function in the RL phase of the training, which made the RNN part of the model unnecessary.

Házi feladatunk keretein belül egy megerősítéses tanulás alapú modellt használva tanítottunk egy ágenst arra, hogy képes legyen az út követésére a Duckietown Gym szimulációs környezetben. Ehhez munkánk során a 2018-ban publikált ún. World Models (1) modellt használtuk kiindulási alapként. A modell különlegessége, hogy nem az ágens által látott környezet nyers képét használja fel, hanem a környezet egy tömörített reprezentációját. Az így kapott, már transzformált megfigyeléseket továbbítja egy rekurrens neurális hálónak, amelynek feladata, hogy az ágens által a következő lépés során látott környezet transzformált formáját prediktálja. Végül az ágens megerősítéses tanulás segítségével tanul. Az eredeti modellt úgy módosítottuk, hogy az eredeti cikkben szereplő lineáris függvény helyett az ún. SAC policy-t használtuk a megerősítéses tanulás során, így az RNN-nek már nem volt létjogosultsága.

1 Introduction

Many of the algorithms in the field of artifical intelligence are inspired by how humans learn and think. Our brain is able to learn an abstract representation of the information it gets from our environment. Our homework is based on a model called World Models (1), which mimics this behaviour. In their article, the authors present a new framework for facilitating the policy evolution in a reinforcement learning (2) task. The main idea of their work is that they transform the environment seen by the agent into a so called latent vector, and the reinforcement learning algorithm is trained using the already transformed reality. So the agent learns in its own world, hence the name of the model. It is

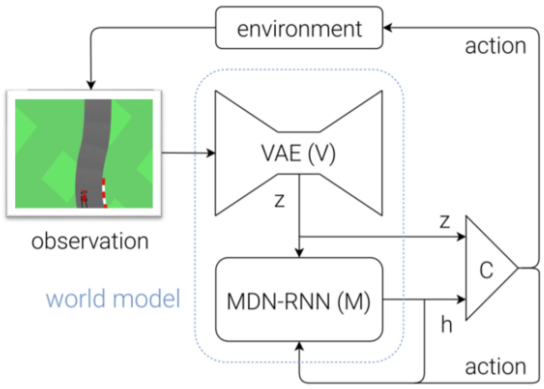


Figure 1: Components of World Models

important to note that the idea of using a VAE network together with an evolutional reinforcement learning algorithm is not unique, there is an article (3) published in 2017 that already using this idea.

2 Agent model

Our task was to implement the World Models(1) (that has been already done for the OpenAI CarRacing environment) for the Duckietown Gym environment. In this chapter we briefly introduce the architecture of the model.

World Models has 3 basic components, depicted in figure 1.

2.1 Vision (V)

The Vision model is a simple Convolutional Variational Autoencoder (VAE) (4), (5) which takes an observation (2D image) from the environment and encodes it into a latent vector z of size 32.

The architecture of the VAE network is shown in figure 2a.

2.2 MDN-RNN (M)

The second building block of the model is an LSTM-based (6) recurrent neural network which is combined with a mixture density network(7). It takes the latent vector z from the vision module, the previous action a chosen by the controller and the previous hidden state h of itself. Similarly to the vision module, its goal is to capture a latent understanding/observation of the environment by predicting what would the next z look like.

The architecture of the Memory RNN network is shown in figure 2b.

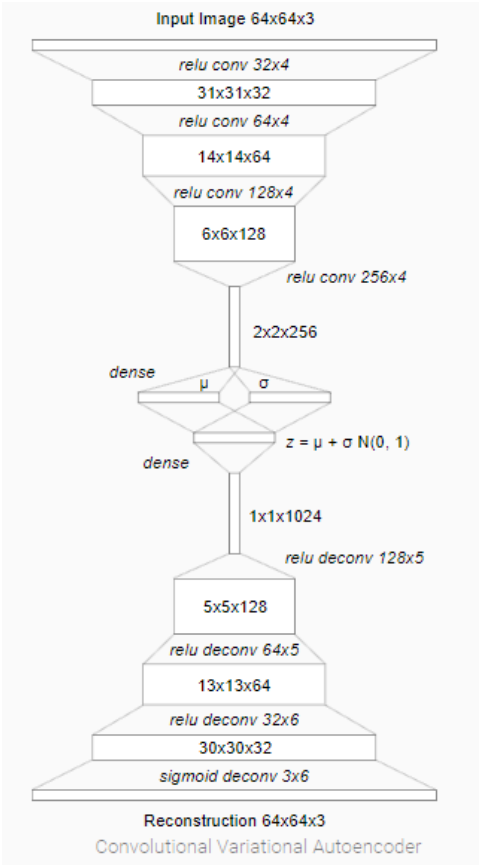
2.3 Controller (C)

In their implementation, the authors used the Covariance-Matrix Adaptation Evolution Strategy (CMA-ES) (8).

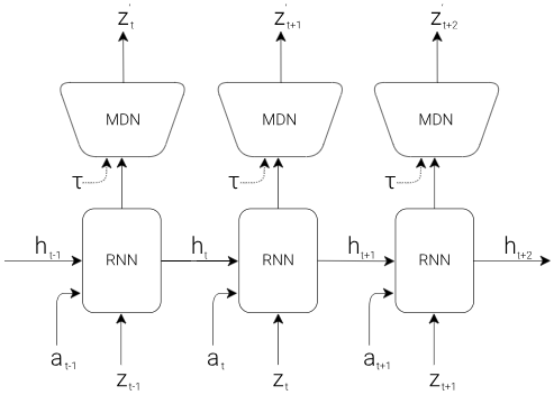
The controller consists of a simple single layer linear model which maps the vision module's observation z and the memory module's hidden state h to an action a at each time step. The output is an action-vector (numpy array) of size 2 containing the quantitative representation of the velocity [-1.0 to 1.0] and the angle of the steering wheel [-1.0 to 1.0].

3 Implementation

In our homework, we used the Duckietown Gym (9) environment. Gym (10) is a toolkit for developing and comparing reinforcement learning algorithms. It supports teaching agents everything from



(a) VAE



(b) Memory RNN

Figure 2: Architecture of the given components

walking to playing games like CarRacing or Pacman. Duckietown is a playful learning experience for state-of-the-art robotics and AI. It makes it possible to develop and test autonomous driving algorithms with Deep Learning in a joyful sandbox environment. To make the development easier, the Duckietown Foundation published the Duckietown Simulator based on the OpenAI Gym environment.

Our goal was to modify an already existing PyTorch implementation of World Models[1] so that we can apply it for Gym-Duckietown.

3.1 Modified agent model

3.1.1 Vision (V)

We had to modify the architecture of the VAE due to the fact that the resolution of the images extracted from the Duckietown Gym environment is different (much larger) compared to the CarRacing environment. Additionally, we changed the size of the latent vector from 32 to 64, since we had to store more information in it.
We performed three different modifications in the following order: firstly we supplemented the VAE network with several convolutional and deconvolutional layers. Thanks to this it was able to encode and decode images with resolution of 480x640. Secondly, we put batchnorm layers between each and every already existing layer. Thirdly, we put in the network several more layers which didn't modify the spatial dimensions of the feature maps being propagated through the network. The number of input channels of these layers equaled the number of output channels.

[1] https://github.com/ctallec/world-models

3.1.2 MDN-RNN (M)

Since we have modified the controller as described below, it was not clear for us how we could combine the MDN-RNN and our new controller. An important difference compared to the CarRacing-based implementation is that they had the current observation and the proposed action, while what we have is the last action and the observation after that. We came to the conclusion that we have to exclude the RNN model from our implementation.

3.1.3 Controller (C)

Instead of the simple linear model, we used the so called SAC reinforcement learning policy (11).

An AC model in general uses a Q-network to estimate the Q function. Given a sample, the neural network predicts a Q value for every possible action. The weights of the Q network are updated using SGD and backpropagation. A stack of consecutive frames is used as a representation of a single input.

SAC uses entropy regularization and is trained to maximize a trade-off between expected return and entropy. Entropy is used to describe the randomness in the policy. A higher value of entropy leads in more exploration, which can accelerate the learning process.

3.2 Data preparation

We have created a dataset of 100 training, 50 testing and 20 validation rollouts. A rollout consists of four arrays (compressed in .npz files) containing the values returned by the step function which is involved in the Gym environment class. The images given by step() are located in "observations".

3.3 Training the model

We made the starting position of our agent fixed, which makes the training easier and faster. We used the so called ”udem1” map.

Since our model has two components, we had to train it sequentially. In the first step, the VAE network was trained independently in the following way:

1. The network takes an image with a size of (640 x 480 x 3) as an input.
2. A convolutional network creates a so called z vector of length 64 as a compressed representation of the input image.
3. A deconvolutional network then takes the z vector as input and generates a (640 x 480 x 3) sized image as an output.
4. Finally, the VAE network computes the loss based on the similarity of the input (original) and the output (generated) images. If the two images are quite similar, that means that the VAE learned to compress images efficiently into a vector of length 64.

Note that the deconvolutional network will only be used during training, after that it will be ”deactivated”. Before going further, we will first fix the parameters of the VAE (we do not want it to train any more).

The controller takes the z vector (length: 64) from the VAE, and using the SAC policy, generates an action a as an output. Based on the action of the controller, the environment generates a reward (a float number) that indicates how good decision the controller has made. The aim of the controller is to maximize the reward it gets.

We use wrappers for modifying the default behaviour of the Duckietown Gym environment. The action wrapper overrides the default shape of the action, so that it tells the SAC that we expect three floating point numbers between 0 and 1. We find the maximum value in this array, and based on its position (1st, 2nd or 3rd) we define the direction of our next discrete step (left, right, straight respectively). The reward wrapper - according the studies carried out earlier - can optimise the way the reward is computed by the environment. The observation wrapper passes the action produced by the SAC (which creates its action using the action wrapper) through the step function of the Gym environment, and transforms the raw image using the VAE into a latent vector.

3.4 Evaluation

3.4.1 VAE

For evaluating the performance of image encoding and decoding we used the quantity called Intersection of Unions (IoU). This is related to the ratio of the number of appropriately decoded pixels and the number of all pixels. We computed the average number of pixels which completely matched with the encoded and decoded ones. We found that there were 23 pixels out of 1201 test images that satisfied this condition the average IoU was accordingly 0,019. We used a channelwise method as well to compute the IoU, this computation gave IoU values for the R, G and B channels 0.6953, 0.8793 and 0.0092 respectively.

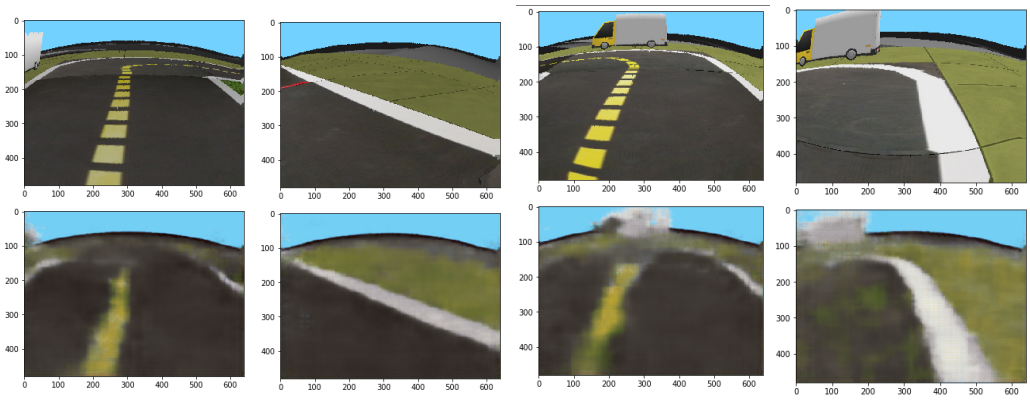


Figure 3: Examples of how our trained VAE is able to encode, and then decode a given observation from the Duckietown Gym environment.

3.4.2 Controller

During the testing phase, we switch of the Critic part of the SAC, and the Actor is in eval mode, which means that it produces actions that we can observe in the Duckietown Gym environment.

The Controller has been trained for 150000 steps. The result is an agent, which is able to follow the road and to stay on the right lane.

4 Summary

All in all, we can say that the policy learnt by the controller is acceptable. However, additional work is needed to optimise the components of our model in order to be able to give a better result.

Acknowledgments

We wish to acknowledge the help provided by Róbert Moni, who has guided us throughout the semester.

Contributions

All of us participated in discovering the literature and reading the papers we required to proceed. In the beginning, we worked together to understand the World Models. This means we have created our first training dataset, and created and trained the first version of our VAE network together. In the early stages of the project, we experimented with the RNN together. As the final deadline came closer, we divided the tasks. The hyperparameter optimization of the VAE was done by Máté Büki. The Controller was implemented and fine-tuned by Péter Zoltán Sánta. Rollout generation was Gábor Tamás's responsibility. In the end, we documented our work together. During the semester, we tried to work as collaboratively as possible to ensure all of us can learn as much as possible.

References

[1] D. Ha and J. Schmidhuber, "World models," *arXiv preprint arXiv:1803.10122*, 2018.

[2] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[3] S. Alvernaz and J. Togelius, "Autoencoder-augmented neuroevolution for visual doom playing," in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 1–8, IEEE, 2017.

[4] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[5] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models," *arXiv preprint arXiv:1401.4082*, 2014.

[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[7] C. M. Bishop, "Mixture density networks." 1994.

[8] N. Hansen, "The cma evolution strategy: A tutorial," *arXiv preprint arXiv:1604.00772*, 2016.

[9] M. Chevalier-Boisvert, F. Golemo, Y. Cao, B. Mehta, and L. Paull, "Duckietown environments for openai gym." `https://github.com/duckietown/gym-duckietown`, 2018.

[10] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[11] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *arXiv preprint arXiv:1801.01290*, 2018.