Matrix/tensor factorization for Reinforcement learning

December 2020

Team members: Maxim Komatovskiy, Nazerke Sandibay, Anastasia Sozykina

Motivation

What and why

Reinforcement learning (RL) algorithms have proven to be successful at various machine learning tasks, but RL requires sampling efficiency and over-parameterization in neural networks used to approximate functions. Therefore, we aim to research ways of making RL techniques computationally and time-efficient.

Hypothesis

To show empirically that in the low-data regime, it is possible to learn online policies less total coefficients, with little or no loss of performance.

Problem statement

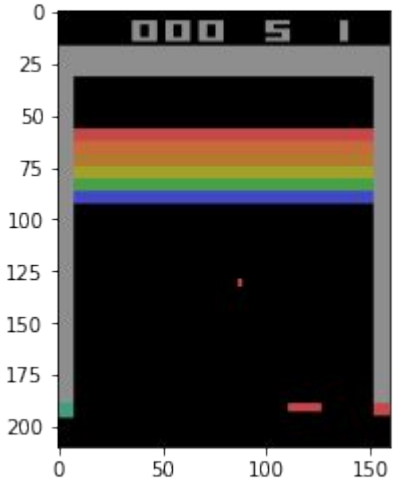
- Environment: Atari, Breakout, v4.0
- States - pictures of the current gamespace
- Actions - moving the platform
- Q-learning based on decaying epsilon-greedy policy
- Approx of the policy function with a network
- Actor-Critic approach*

- What would happen if we somehow interfere some of the layers?

Problem statement

Figure out the impact of different internal changes in linear layers on the system’s trajectory.

The quality measure would be enhanced by the rules of the game - just the score (or number of the bricks hit).



Atari

Methods

Linear Algebra

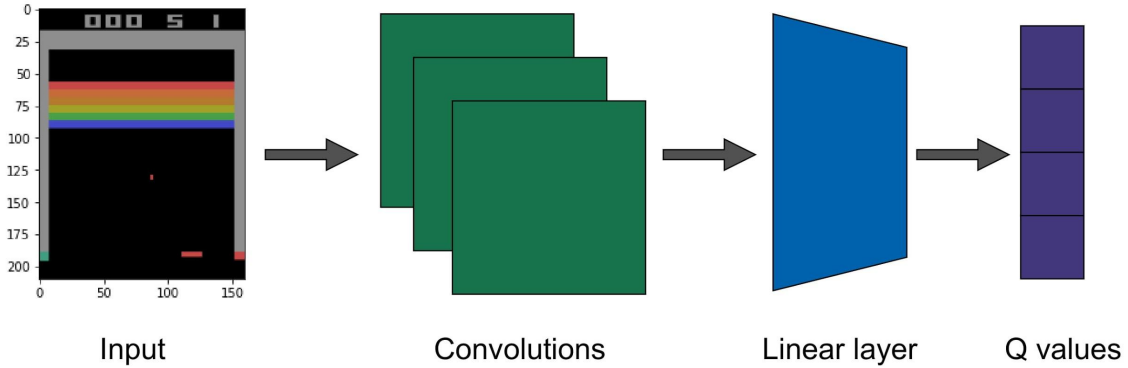
- SVD (Tucker Decomposition for potential Tensor case)

Reinforcement Learning

- Q-learning
- Actor-Critic

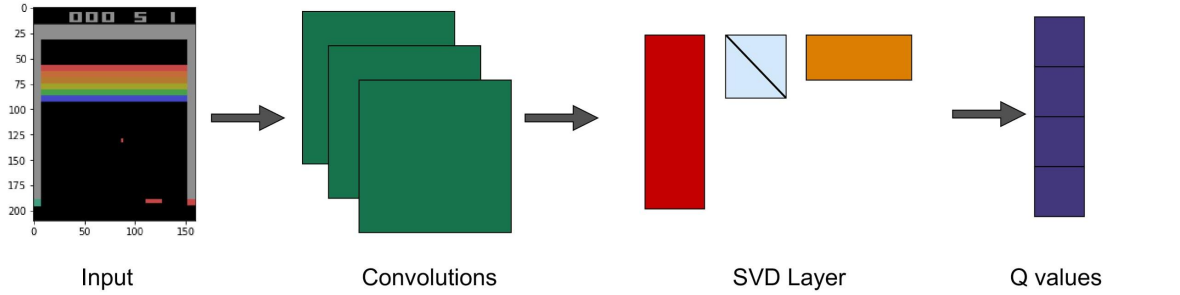
Policy function proxy

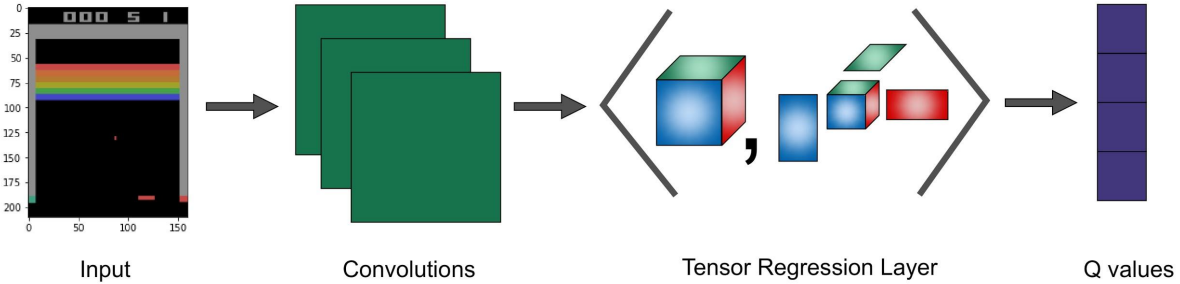
Architecture




```
Net(
  (conv1): Conv2d(4, 32,
kernel_size=(8, 8), stride=(2, 2))
  (r1): ReLU()
  (conv2): Conv2d(32, 64,
kernel_size=(4, 4), stride=(2, 2))
  (r2): ReLU()
  (conv3): Conv2d(64, 64,
kernel_size=(3, 3), stride=(2, 2))
  (r3): ReLU()
  (dense): Linear(in_features=4096,
out_features=512, bias=True)
  (r4): ReLU()
  (out): Linear(in_features=512,
out_features=4, bias=True)
)
```

Decomposition of hidden layer





Main idea is to replace dense, fully-connected linear layers with

- SVD layer
- Tensor regression layer (Tucker decomposition)

SVD on trivial benchmark solution

The initial quality of DQN solution was quite low, as a one-hidden layer network is a basic way to approximate such a complex Q-function.

Attempts to factorize this Hidden layer with SVD led to some crucial losses of the mapping and the results decreased to almost random-playing ones (average of 2.3 bricks per episode).

Also the results became highly dependent on the initial state of the environment, which can be split into the “good” ones and the “bad” ones.

Actor-Critic

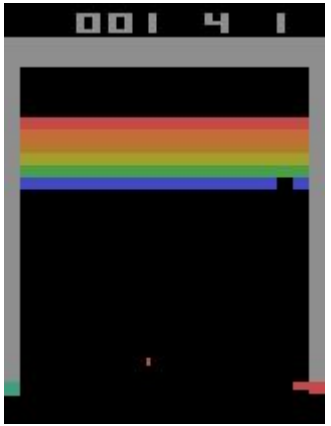
Consequent estimation:

On the first step we update the value (or Q-) function,

based on the last buffered info.

On the second step, the policy part is updated.

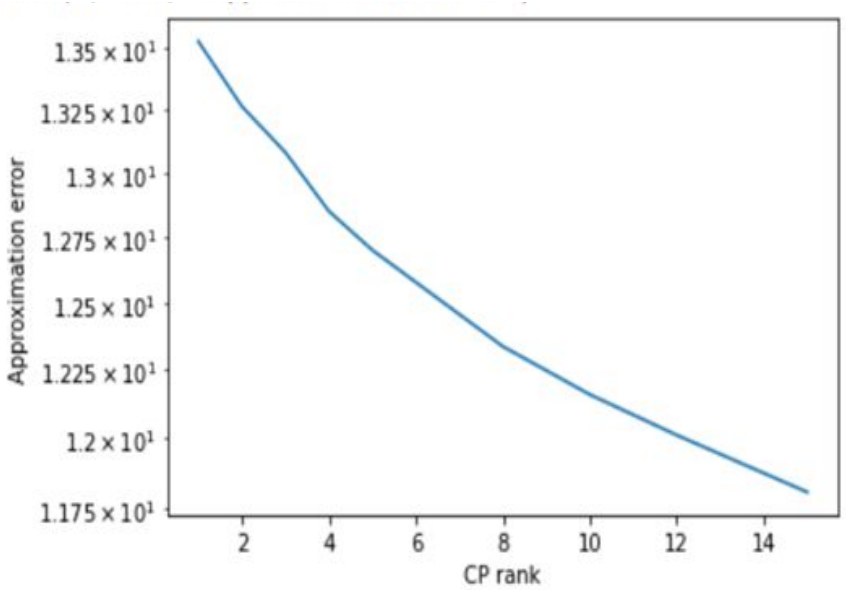
To put it simply, the architecture becomes much heavier.



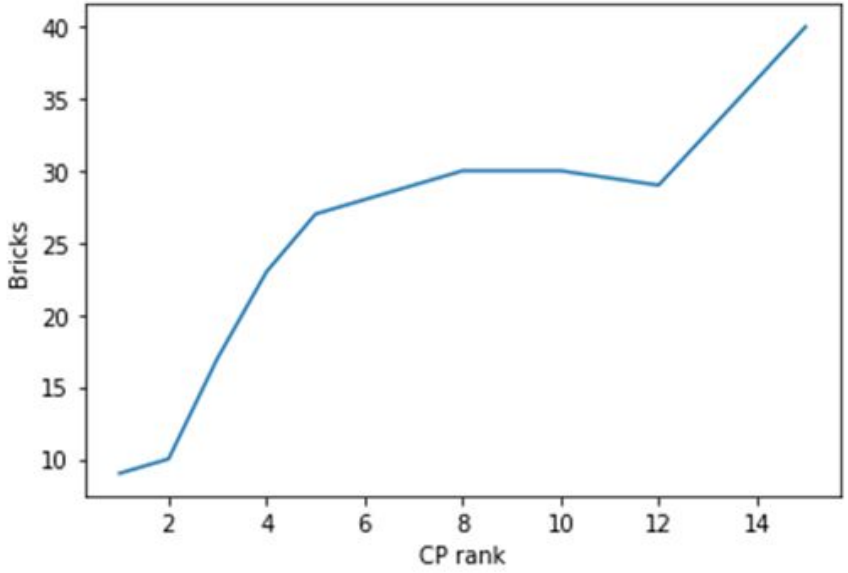
rank=8

3x3x64x64=36864
3x8 + 3x8+ 64x8+ 64x8 =1072

Performance



Approximation error - CP rank dependency



Amount of bricks - CP rank dependency

Summary

- If the initial model is not so complex, there is no sense in compressing it further: some crucial mappings may be lost.

- In case of more complex approximators or approaches, an attempt to decrease the number of parameters may be relevantly beneficial.

References

1. Pierre H. Richemond and Arinbjorn Kolbeinsson and Yike Guo, How many weights are enough : can tensor factorization learn efficient policies?, 2020, in ICLR 2020 openreview, https://openreview.net/forum?id=B1l3M64KwB
2. Jean Kossaifi, Yannis Panagakis, Anima Anandkumar and Maja Pantic, TensorLy: Tensor Learning in Python, Journal of Machine Learning Research, Year: 2019, Volume: 20, Issue: 26, Pages: 1−6.
3. https://openai.com/blog/baselines-acktr-a2c/

Matrix/tensor factorization for Reinforcement learning

December 2020

Team members: Maxim Komatovskiy, Nazerke Sandibay, Anastasia Sozykina