Mnih-Large-Convolutional

January 10, 2020

1 Mnih-Large-Convolutional

In this notebook, we demonstrate how to train the original DQN as described by Mnih et al. [1] on Breakout. Before finishing the thesis, we didn't reach satisfying results and also didn't merge the code for reproducing Mnih et al. with the rest of our code. Therefore, we outline what challenges remain in training, laoding, and converting and how to merge the code in this directory with the main part of our code in the future. The code for training, loading, and converting the convolutional network is provided in this repository in the form of python scripts.

1.1 DQN Training

The script TrainMnihBinary.py trains the DQN agent. Compared to Minh et al., we use the same hyperparameters with two exceptions: First,the RMS-Prop parameters are potentially different as the implementations between tensorflow and PyTorch are different. Second, we use the same preprocessing as Mnih et al. but then binarize the images by setting a low threshold (all pixels with gray values >5 become 1, all others 0). We did this in order to reduce the amount of required working memory.

Training takes a lot of time (several days using one GPU) and information about the progress is printed every 1000 episodes.

To improve the runtime of the code, one could try to remove the binarization which induces a runtime overhead because images are converted to bitarray before saving them in the replay memory and then back to torch when loading them from the replay memory. However, without binarization the memory requirement goes up to around 40GB RAM. One could try to save the replay memory on disk and then load only random lines from the document during optimization. This could potentially speed up the runtime and improve the results (if the binarization causes performance issues) but needs to be tested.

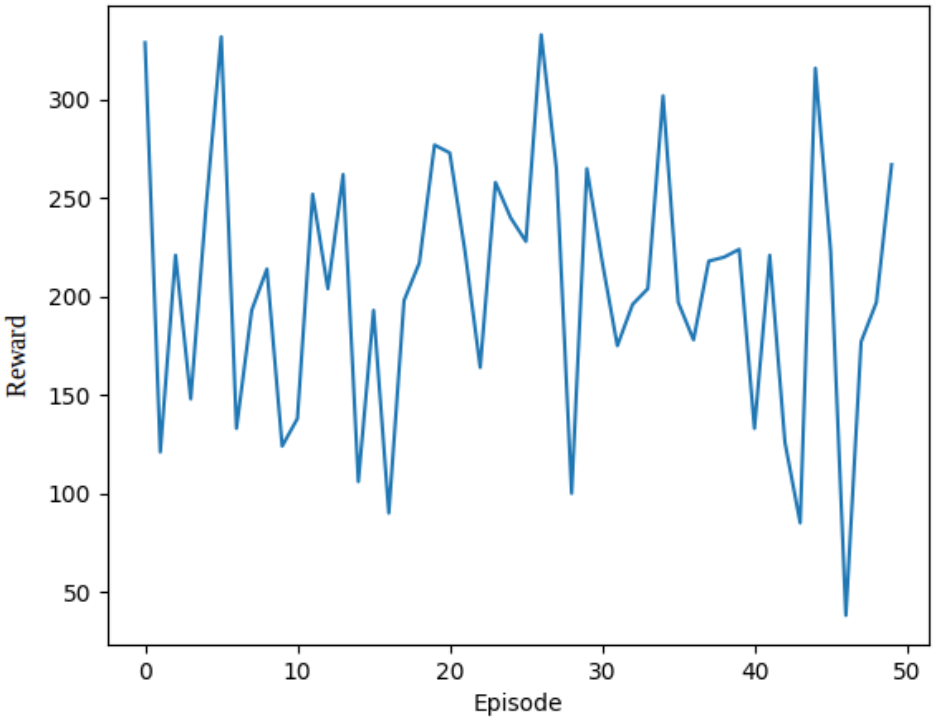
Merging TrainMnihBinary.py with the rest of our code faces one major challenge. This is that when pushing to or pulling from the memory additional preprocessing happens. To merge the code, one would need to pass an encoding and a decoding funcction to the agent which is then applied before pushing and before pulling respectively.

Parts in the code which need improvement or need to be changed in order to merge are marked with TODO.

1.2 Loading

We have trained our agent using the code already but due to a bug in the code which is now fixed, results were suboptimal. The best performing agent is saved in the results directory in the folder

Breakout-Mnih-Preliminary-DQN. By executing the script LoadMnihBinary.py the agent can be loaded. It achieves an average performance of around 200 which is good but much worse than the result reported by Mnih et al. which is around 400. The graphic below shows our agent's performance across 50 episodes.



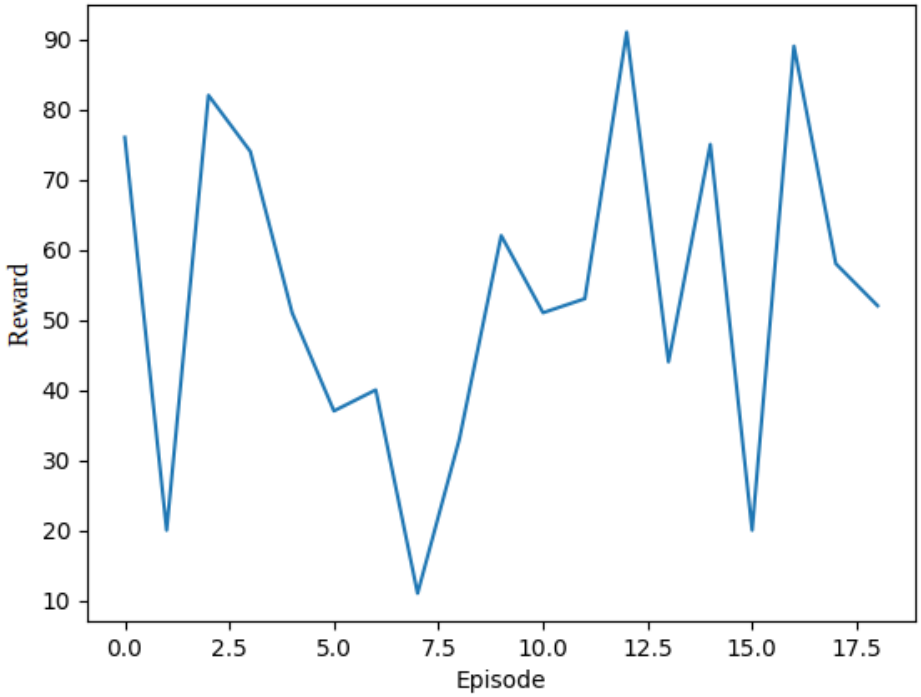
title

If you run LoadMnihBinary.py, similar results can be observed.

Merging the code from loading with our main code should be straightforward as a input preprocessing function can be passed to load_agent.

1.3 Converting

Using the script ConversionMnihBinary.py the saved agent can be converted. If you want to execute the script, make sure you first saved the replay memory using LoadMnihBinary.py. The replay memory is not provided in the repository because it exceeds the maximal allowed file size of 100MB. We simulate the converted network for 18 episodes and obtain the following performance. The simulation time of the spiking neural network is set to 500 time steps as reported by Patel et al. [2].



title

Simulating the agent takes a lot of time, so we only simulated it for 18 episodes. Already, it can be seen that the performance massively drops through the conversion. Also, the conversion accuracy is quite low. It needs to be investigated why converting does not work well in our experiment, while Patel et al. report extremely strong results. One reason could be that through our suboptimal training process (see Loading) the agent becomes harder to convert.

Also, the method we use to simulate a spiking convolutional layer is hacky and should be triple checked for bugs. In the future, our SQN class should be extended to be able to use convolutional layers. This would then also allow to merge the script ConversionMnihBinary.py into the main part of our code.

1.4 Direct Training using Surrogate Gradients

We have not yet investigated direct training with surrogate gradients for the large network. It would be necessary to augment our SQN class in order to support convolutional layers. This is likely the only change that is required to be able to use surrogate gradient training.

1.4.1 Sources

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. Nature, Volume 518, 26 February 2015.

[2] Devdhar Patel, Hananel Hazan, Daniel J. Saunders, Hava T. Siegelmann, and Robert Kozma. Improved robustness of reinforcement learning policies upon conversion to spiking neuronal network platforms applied to Atari Breakout game. Neural Networks, 2019.