Feature Visualization for A2C

Christopher Galias Robin de Heer

1 Introduction

As machine learning techniques become even more pervasive in critical systems, the problem of their interpretability is coming to the forefront. Some people think about artificial neural networks as if they are black boxes which can be trained to perform quite well on certain tasks.

A way to really understand what happens in an artificial neural network is by training it using the algorithm black box, in this case A2C (which we elaborate later on), and allow our agent to 'solve' Atari Breakout. If our agent is able to do so, we can attempt to visualize what happens in specific layers and even in specific neurons. That is, we can generate a picture which will stimulate this layer or neuron maximally. That way, we can see to what a certain part of the network is really sensitive to. This technique is called feature visualization and is outlined in e.g. [Olah et al., 2017]. Using this technique may grant us new insights in the functionality of artificial neural networks.

2 Concepts

2.1 Advantage Actor Critic (A2C)

A2C is the synchronous variant of Asynchronous Advantage Actor Critic (A3C) [Mnih et al., 2016], which is found to perform similarly [Wu et al., 2017a]. A3C features first of all asynchronous updates, but also a way to update after a fixed amount of time steps, which are then used to predict the returns of the advantage function. The value functions and policies are closely connected to one another. A2C does exactly that, but with synchronous updates.

In short, A2C does for some amount of iterations the following: Act for some amount of time steps, while computing both the target value function and the estimated advantage function, use those values to compute a loss gradient and put this loss value in stochastic gradient descent.

2.2 Feature Visualization

Feature visualization is a way to graphically show which pixels with which values stimulate a neuron, a layer of neurons or even multiple layers of neurons most. We planned to visualize the features using the Optimization Approach as described by Elisa Sayrol[Sayrol, 2016]. The first step in this process is to initialize and train an artificial neural network to become proficient at a specific task. Then we generate a random picture. This picture is then forwarded through the network until we reach the layer or neurons we wanted to visualize. Now we forward the image though this layer and check whether or not the activations are maximal. If not, we adjust pixels of the random picture, backpropagate and try again until the activations are maximal.

3 Methods

The A2C agent models are implemented using the Chainer ([Tokui et al., 2015]) package in Python. The environments are OpenAI's ([Brockman et al., 2016]). We used standard hyperparameters, utilizing the Adam optimizer with gradient clipping to a norm of 40 (as in OpenAI's baselines, cf. [Wu et al., 2017b]). The number of steps after which an update is made is 20. The two actor/critic architectures considered are an MLP and a CNN.

4 Results

Unfortunately, as of time of writing, the A2C agent does not learn in a satisfactory manner, even on the toy examples we considered. This might be a result of bugs, but perhaps a more thorough hyperparameter search would lead to better results, as would longer running time (even more if we had access to more compute). We also failed to implement feature visualization in the manner proposed before.

5 Discussion

In this paper we attempted to implement and train an A2C agent[Wu et al., 2017b] to become proficient at the Atari game "Breakout"[Brockman et al., 2016] and tried to visualize neurons and layers using the optimization approach[Sayrol, 2016]. Because the A2C agent and the visualization of the features do not yet function as they should, we are greatly limited in drawing conclusions about the performance of both. Further work should include getting the agent and visualization method to work, but could also include comparing different visualization methods to one another and comparing the A2C algorithm to other reinforcement learning algorithms. Other possible future projects are finding optimal hyperparameters and comparing those hyperparameters to optimal hyperparameters for other problems in the OpenAI Gym.

References

[Brockman et al., 2016] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). OpenAI Gym. `https://arxiv.org/abs/1606.01540`.

[Mnih et al., 2016] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783.

[Olah et al., 2017] Olah, C., Mordvintsev, A., and Schubert, L. (2017). Feature Visualization. *Distill*. `https://distill.pub/2017/feature-visualization`.

[Sayrol, 2016] Sayrol, E. (2016). Visualization. http://imatge-upc.github.io/telecombcn-2016-dlcv/slides/D2L3-visualization.pdf.

[Tokui et al., 2015] Tokui, S., Oono, K., Hido, S., and Clayton, J. (2015). Chainer: a next-generation open source framework for deep learning. In *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Twenty-ninth Annual Conference on Neural Information Processing Systems (NIPS)*.

[Wu et al., 2017a] Wu, Y., Mansimov, E., Liao, S., Grosse, R. B., and Ba, J. (2017a). Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. *CoRR*, abs/1708.05144.

[Wu et al., 2017b] Wu, Y., Mansimov, E., Liao, S., Radford, A., and Schulman, J. (2017b). OpenAI Baselines: ACKTR & A2C. `https://blog.openai.com/baselines-acktr-a2c/`.