Reproducing “Towards Interpretable Reinforcement Learning Using Attention Augmented Agents”

Charles Lovering
Department of Computer Science
Brown University
charles_lovering@brown.edu

Sam Lobel
Department of Computer Science
Brown University
samuel_lobel@brown.edu

Denizalp Goktas
Department of Computer Science
Brown University
denizalp_goktas@brown.edu

Kweku Kwegyir-Aggrey
Department of Computer Science
Brown University
kweku@brown.edu

Albert Webson
Department of Computer Science
Brown University
albert_webson@brown.edu

Abstract

We attempt to reproduce “Towards Interpretable Reinforcement Learning Using Attention Augment Agents” [14][1], a recent work which introduces a novel attention mechanism to understand the reasoning behind the predicted policy for canonical Atari games in a reinforcement learning setting. Like the original paper, our implementation utilizes the Importance Weighted Actor Critic Architecture (IMPALA). Our implementation, despite using considerable resources, was only trained for a fraction of the number of steps done by the original authors. Nonetheless, the results that we obtain from our early-training stage model indicate the validity of the authors’ work.

1 Introduction

Recently, deep reinforcement learning algorithms have demonstrated significant success in playing canonical Atari games [12, 13]. These algorithms combine deep learning and reinforcement learning techniques to process the raw pixel inputs. Often, such algorithms map the pixels either directly to policies, to state-action value estimates, or both. The computation of this optimal behavior typically relies on deep learning function approximation. Unfortunately, despite its well-championed success, deep reinforcement learning also suffers from one of the well-known drawbacks of deep learning - neural networks are difficult to interpret. In this report, we evaluate a proposed approach for handling the opaque nature of deep reinforcement learning agents.

Previous attempts to elucidate deep reinforcement learning have used saliency maps [5, 20] to determine how changes in the input change the learned policy. Although these methods are successful in highlighting regions responsible for a policy’s decision, these approaches offer limited information beyond the pixel level. The paper “Towards Interpretable Reinforcement Learning Using Attention Augmented Agents” by Mott et al. proposes using attention mechanisms to determine regions in

[1]We refer to the authors of this work as the “original authors” and Mott et al.

input images which are salient to some learned policy. By only providing very compressed attention-modulated features to the policy, they ensure that for the agent to increase its score, the model must learn meaningful attention maps. They demonstrate success in this regard, producing videos of an attention mechanism following enemies, looking ahead of the agent, and observing locations of potential future importance. They also find that the trained agent's attention maps often answer "what" and "where" questions about important game elements.

In this report, we attempt to show that the attention maps blended with the Atari game video offers a clear interpretation for what the agent takes into account when forming its policy during gameplay in `SeaQuest` (see Figure 2 in [14]). Due to limitations in computing resources, we were unable to fully reproduce this result, however, using our implementation, we have been able to achieve learning outcomes on-track with those in the paper of interest. We expect that with additional computing power, our reproduction will be able to replicate the full results of the original implementation.

2 Related Work

Attention mechanisms have been used extensively in many deep learning applications: visual question answering [10, 19], machine translation [2], image classification [17, 1], object tracking [3], and deep reinforcement learning[11]. Attention mechanisms are typically separated into two categories: soft attention and hard attention [18]. A visualized set of soft attention weights will show how every region of an image contributed to some classification or generation result, when overlayed on the input pattern. Although soft attention can be computational expensive for large input, it is fully differentiable, which motivates its use for an end-to-end differentiable system. Hard attention however, can only select one portion of an image to direct its attention, or highlight. This attention is computationally less expensive than soft attention, but is not differentiable, and typically requires alternate methods to successfully employ.

2.1 Query-Based Attention

The attention mechanism used in the original paper is described as a "soft key, query, and value type of attention" mechanism. The attention is similar to [15] and [16], with some key differences. In the proposed model, a query-based attention mechanism produces attention queries based on internalized agent state. For any observation at time t, an LSTM processes the agent's output from time $t-1$. The resulting internal LSTM state is then used to guide the attention query. The recurrent nature of this style of mechanism, allows temporal dependencies to be encoded in the network's behavior: an agent can ask, "given the previous observed state, what regions in the next input patterns should attention be placed?" Thus, the attention mechanism does not have immediate access to observed pixels nor a CNN-encoded feature map when forming its queries, and acts as a strong bottleneck during training.

2.2 Asynchronous Advantage Actor-Critic

The original paper uses the IMPALA framework [4] to train the agent. The IMPALA framework is based on the Asynchronous Advantage Actor Critic (A3C) architecture [12], an actor-critic architecture [8] designed to work well in distributed settings. Actor-critic methods learn both a state-action value function and a policy used to determine agent behavior; the value function is learned via value iteration, and the policy is learned though policy gradient on the estimated value function [4, 12]. Additionally, A3C is asynchronous: multiple agents are deployed with a copy of the learning environment and parameters. The agents each navigate their learning environments and contribute their learned contributions back to some global network, which encapsulates the learned global behavior, which is the behavior of interest. The A3C framework requires a high number of iterations to achieve *good* performance on a single task [6], thus the authors looked to the IMPALA [4] framework to address the scalability and tractability issues of A3C.

2.3 Is Attention Actually Interpretable?

Because attention mechanisms often supply intuitive visualizations, attention weights are commonly accepted as a valid interpretation of what a model "considers important" in making predictions. Recently, however, Jain and Wallace [7] show compelling evidence that dispute this assumption. They use a standard bidirectional LSTM model with attention in a variety of common NLP tasks.

They find that the attention weights are only weakly correlated with gradient-base measurements of feature importance. Furthermore, they replaced the learned attention distribution with some randomly permuted distributed or an adversarial distribution, minimally affects prediction accuracy.

It is important to note that Jain and Wallance's study focuses on the language setting, whereas our model uses attention in a visual domain. We also highlight that Mott et al. [14] (the original authors) showed in an ablation study that removing the attention module from the LSTM module results in significantly worse performance. As more complex attention mechanisms are developed, more research is needed in to verify their interpretability.

3 Architecture Overview

Here we provide a brief overview of the architecture for the reader, and point both to the original work [4] and our source code [2] for more details. See Figure 1 for a schematic diagram of the architecture.

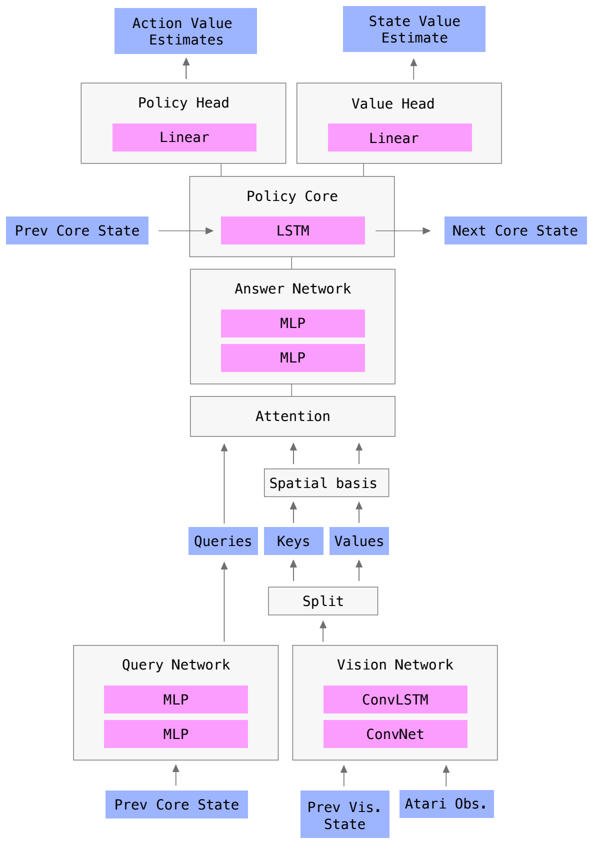

Figure 1: **Attention Network Architecture.** Purple rectangles denote network inputs and outputs (matrices); pink rectangles denote network operations with learnable parameters (layers); grey rectangles denote higher-level network modules (conceptual steps).

As shown in Figure 1, the Vision Network processes an observation from the Atari game, and produces a set of keys and corresponding values. These matrices are then modified to track spatial information using a spatial basis (see Figure 2). In parallel, the Query Network processes the model's hidden state from the previous time step and produces a set of queries (or questions). Next, a soft-attention mechanism akin to [16] is used to summarize an answer for each query; each answer is a linear combination of the values. The queries and answers, along with the previous reward and previous action (not shown in the figure), are concatenated together and processed by the Answer Network. From here, the Policy Core processes the result and its output is linearily projected 1) to policy estimates (a set of state-action value estimates) and 2) to values estimates (the state value estimate).

[2]`https://github.com/cjlovering/torchbeast/blob/master/torchbeast/attention_net.py`

Crucially, the policy and value heads only have access to attention-modulated representations of the vision features. This allows us to use attention visualizations to understand the agent's decision-making. This is an actor-critic setup: the estimates are used to update the agent using VTRACE loss with a RMSProp optimizer.

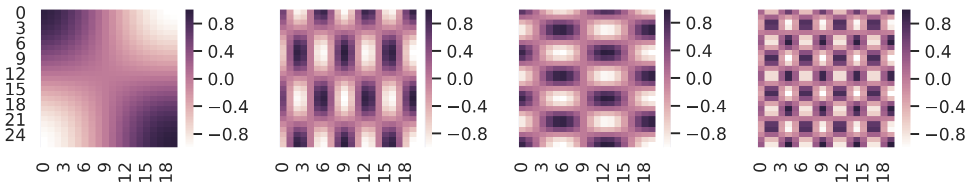

Figure 2: **Four Slices from the Spatial Basis.** Each of the above heatmaps corresponds to one of the 64 static 20×27 spasis basis that the agent uses for learning spatial relations. We want to stress 1) that the basis is static (not learned) and 2) the model does not need to use the spatial basis for unseen dimensions or sizes. Specifically, the spatial basis is specific to the size of model's internal representation, which in this case 20×27 "pixels". Thus, the spatial basis is used by concatenating the 64 dimensions (for each pixel location) to each key and value output by the Vision Network. Note that each of these pixels are CNN-encoded features and do not correspond one-to-one with the pixels observed from the Atari games.

4 Implementation Details

4.1 Model

We implement the proposed model in PyTorch[3], as opposed to TensorFlow[4], with the hopes of showing that the results are both implementation and platform independent. We used Google Cloud Compute Instances to host our experiments, and we used two NVIDIA T4 GPUs to train our agents. All hyperparameters are set to those reported in [14]. The original authors did not publicly release any code and we did not correspond with the original authors.

4.2 Framework

The authors used IMPALA [4], a TensorFlow-based model-training distributed system, to train their models. IMPALA is a successor of the Asynchronous Actor Critic Advantage (A3C): it consists of a set of actor threads which generate offline experience trajectories, which are then dynamically batched and processed by a learner thread. The learner processes the trajectories, then updates both its weights and the actors' weights. We use a recent PyTorch reimplementation of IMPALA called Torchbeast [9]. We worked with the authors of Torchbeast when setting up their framework on Google Cloud.

We trained our model with two NVIDIA T4 GPUs and 48 CPUs. [14] does not mention the computational resources or wall-clock time required for training, so we estimated them using the reported results of IMPALA [4]. We expected a complete training run of *2e9* frames to take roughly 12 hours given a comparable architecture's performance in Table 1 of [4], however training actually executed roughly 20x slower than the reported runtime. We attribute this to three oversights in our estimate:

1. The benchmark runtime reports frames per second executed by the agents, which only calculate an action for one every 4 frames. The reported training axis in [14] is frames-with-actions. This oversight result in a quadruple-time slowdown in expected runtime.
2. Images were downsampled to have roughly 5 times less pixels than our Atari games, further slowing down our model compared to the reported figures.

[3]`https://pytorch.org/docs/master/torch.html`

[4]`https://www.tensorflow.org/`

3. The architectures used to generate the reported FPS in [4], are much simpler than the attention-based architecture proposed by Mott et al. [14]. The increased size of our architecture further exacerbated the tractability of our setup.

The first two differences account for most of the difference between our expected speed and reality. However, taking these into account, our model trains at the rate expected of IMPALA. Thus, these combined factors limited the amount of training we were able to perform with our allotted cloud budget, as we highlight in the results below (Sec. 5).

4.3 Video Generation with Attention Maps

The authors claimed that the attention maps were used by the model to ask (and answer) “where” and “what” questions regarding important game elements. For example, one attention head may track the location of the agent’s sprite location, and others may track enemy sprite locations. To demonstrate their point the authors blend the re-scaled attention maps with the frames that generated them, and create a video by the array of frames obtained.

To produce videos similar to those of the authors, we use a similar blending process. First, we re-normalize the values in the attention map so that the largest value is 1 and then add a positive bias so that the original image is visible. We then re-scale each attention map so that the size of the attention map matches the size of the frames. Finally, we do a pixel-wise weighted average of each RGB channel’s entry with the attention map entries, allowing us to display pixels in the frame with high attention more prominently. We do this process for all the frames in a randomly sampled run, for each attention map [5]. As there are 4 attention maps, this results in 4 videos.

5 Results

We focus on the qualitative results of the original authors’ work, aiming to find if the proposed model is able to play Atari games successfully, and if it uses attention to focus on important game elements. We trained our implementation for significantly fewer steps than the original work, so the model’s score in `SeaQuest`, the Atari game we chose to train on, is lower. However, our learning curve looks qualitatively similar to theirs during this early training regime.

As said in Sec 4.2, the authors do not report the required computing power or training time to achieve these results. Nevertheless, due to the reasoning in Sec 4.2, we believe our model ran at a similar efficiency to their implementation, although likely on different hardware. Training 75 million frames with our architecture (approximately 3% of total training) required 20 hours of wall-clock time and \$75 worth of resources on Google Cloud Platform. Thus, training to completion on a single Atari task on *2e9* frames would cost roughly \$2,000.

As we did not train to completion, we cannot offer conclusive evidence on the replicability of this model. That said, our model replicated the exact neural network architectures and used a framework similar to that of IMPALA used by the authors, leading us to believe that the main issue in not having similar results is the lack of training time. We have re-created figures and videos in an attempt to analyze the attention mechanism that the model learns (Figure 2 (page 4) in [14]). While the attention maps do statically cover certain high-importance areas such as the edges of the screen, they do not display many of the convincing dynamic features of the maps presented in [14], such as tracking enemies or forward planning. As our model is learning, and attention acts as a strong bottleneck on the inputs to the policy, we can conclude that the agent learns some degree of attention. However, at our stage of training this attention is not interpretable. We provide examples of the attention maps in Figure 4. We also provide videos of attention from an untrained model [6] and from our trained model [7]. The untrained model’s attention is diffuse through the whole screen, while the trained model focuses its attention on specific areas. The differences between these, as well as the corresponding score improvement, suggest some amount of learned attention.

[5]We terminate the run after 500 steps.

[6]https://imgur.com/a/vyJ6z8m

[7]https://imgur.com/a/SGLPQvQ

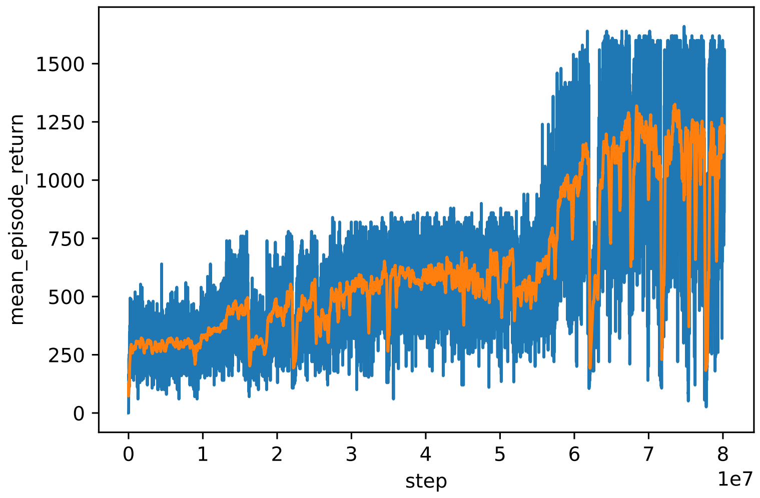

Figure 3: **Model Performance on** `SeaQuest`. The noisy blue line is the model's mean episode return at the given step, and the orange line is the rolling average over the previous 100 observations (which corresponds to 160,000 steps). We can see the model is improving over time, but this score is significantly below that reported by Mott et al. which is approximately 20,000.

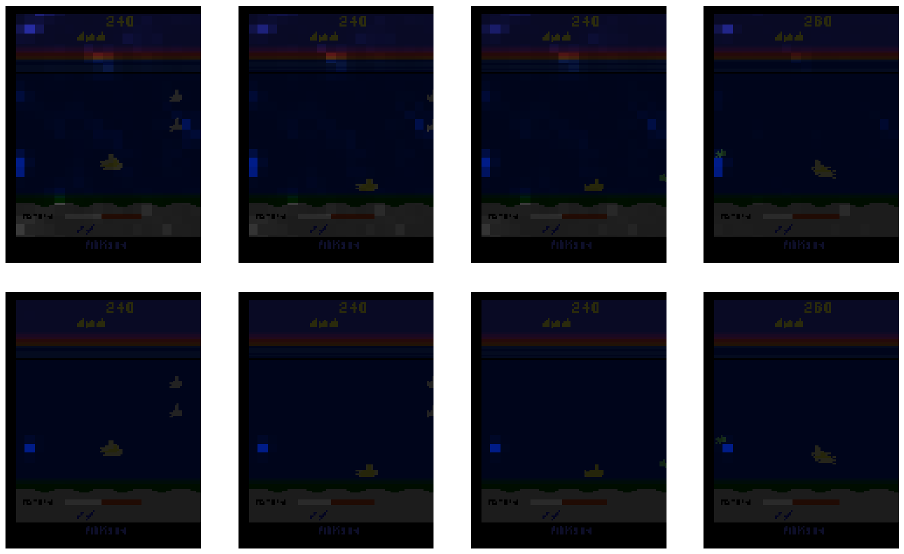

Figure 4: **Snapshots from Sampled Episodes on** `SeaQuest` **with Attention Blending**. Both attention heads shown, appears to remain static, but do attend to the location where new enemies appear. However, the attention heads do not seem to track important moving game elements, like enemies or the agent's sprite.

6 Conclusions

We re-implemented Mott's [14] proposed model, but because of limited compute resources, we were unable to conclusively reproduce either Mott's quantitative or qualitative results. Specifically, while the model does improve over time, the learned attention maps did not lend themselves to clear interpretations. Still, the results that we obtain from our early-training stage model indicate the validity of the authors' work.

References

[1] Peter Anderson, Xiaodong He, Chris Buehler, Damien Teney, Mark Johnson, Stephen Gould, and Lei Zhang. Bottom-up and top-down attention for image captioning and VQA. *CoRR*, abs/1707.07998, 2017.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv*, 2014.

[3] Qi Chu, Wanli Ouyang, Hongsheng Li, Xiaogang Wang, Bin Liu, and Nenghai Yu. Online multi-object tracking using cnn-based single object tracker with spatial-temporal attention mechanism. *CoRR*, abs/1708.02843, 2017.

[4] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*, 2018.

[5] Sam Greydanus, Anurag Koul, Jonathan Dodge, and Alan Fern. Visualizing and understanding atari agents. *CoRR*, abs/1711.00138, 2017.

[6] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z. Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *CoRR*, abs/1611.05397, 2016.

[7] Sarthak Jain and Byron C. Wallace. Attention is not explanation. *CoRR*, abs/1902.10186, 2019.

[8] Vijay R. Konda and John N. Tsitsiklis. Actor-critic algorithms. 2000.

[9] Heinrich Kuttler, Nantas Nardelli, Thibaut Lavril, Marco Selvatici, Viswanath Sivakumar, Tim Rocktaschel, and Edward Grefenstette. Torchbeast: A pytorch platform for distributed rl. *arXiv preprint arXiv:1910.03552*, 2019.

[10] Huayu Li, Martin Renqiang Min, Yong Ge, and Asim Kadav. A context-aware attention network for interactive question answering. *CoRR*, abs/1612.07411, 2016.

[11] Anthony Manchin, Ehsan Abbasnejad, and Anton van den Hengel. Reinforcement learning with attention that works: A self-supervised approach. *CoRR*, abs/1904.03367, 2019.

[12] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.

[13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[14] Alex Mott, Daniel Zoran, Mike Chrzanowski, Daan Wierstra, and Danilo J Rezende. Towards interpretable reinforcement learning using attention augmented agents. *arXiv preprint arXiv:1906.02500*, 2019.

[15] Niki Parmar, Ashish i, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, and Alexander Ku. Image transformer. *CoRR*, abs/1802.05751, 2018.

[16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[17] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *CoRR*, abs/1502.03044, 2015.

[18] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *CoRR*, abs/1502.03044, 2015.

[19] Zichao Yang, Xiaodong He, Jianfeng Gao, Li Deng, and Alexander J. Smola. Stacked attention networks for image question answering. *CoRR*, abs/1511.02274, 2015.

[20] Tom Zahavy, Nir Ben-Zrihem, and Shie Mannor. Graying the black box: Understanding dqns. *CoRR*, abs/1602.02658, 2016.