Deep Reinforcement Learning with Actor-Critic

Christopher M. Lamb
Department of Computer Science
University of California San Diego
San Diego, CA 92122
c2lamb@eng.ucsd.edu

Daniel Reznikov
Department of Computer Science
University of California San Diego
San Diego, CA 92122
drezniko@eng.ucsd.edu

Luke Liem
Department of Computer Science
University of California San Diego
San Diego, CA 92122
lliem@eng.ucsd.edu

Alexander Potapov
Department of Electrical and Computer Engineering
University of California San Diego
San Diego, CA 92122
apotapov@eng.ucsd.edu

Sean Kinzer
Department of Computer Science
University of California San Diego
San Diego, CA 92122
skinzer@eng.ucsd.edu

Christian Koguchi
Department of Electrical and Computer Engineering
University of California San Diego
San Diego, CA 92122
ckoguchi@eng.ucsd.edu

Abstract

Deep Reinforcement Learning (RL) is a powerful approach to agent learning in complex environments. Its applications range from gaming, to robotics to financial-market trading, achieving state-of-the art results across some distinctly challenging problem spaces. The aim of this work is to survey the actor-critic based techniques that have been applied to agent learning in Atari games [1]. We profile baseline approaches and experiment with network topologies and hyperparameter settings. We then benchmark our actor critic agents against an A3C agent implemented by Kostrikov [2] based on the asynchronous multi-agent learning described by Minh et al [3]. Using our own actor critic implementations which utilize LSTM units and temperature annealing we are able to achieve 123% and 487% of human expert level play in Pong and Breakout respectively.

1 Introduction

The algorithms for RL can be categorized into policy-based (policy gradients), value-function based (Q-learning) and a hybrid of the two (actor-critic). The actor-critic paradigm is simple and logically expressive. The actor's role is to engage in some (pre-determined) policy, and the critic is responsible for evaluating this policy. Through interactions with the environment (in our case gameplay) and explicit rewards (points scored) both the actor and critic are able to iteratively update the representations of their tasks.

Our goal is to profile a baseline approach which uses a single-agent advantage actor-critic model (where the actor is learning a policy through policy gradient [4] and the critic is learning a value function similar to classic Q-Learning [5]), and explores ways we can modify the network to achieve better performance. We also try to reimplement A3C [3] for asynchronous multi-agent play what has recently been shown to reduce learning time drastically.

In summary, our efforts represent a survey of the state-of-the art methods, with some novel experimentation do understand more deeply how network topologies (architectures and convolution vs linear approaches) and hyperparameter tunings affect both the ability of agents to learn and the training time.

2 Related Works

One of the early attempts at reinforcement learning is SARSA [5](state-action-rewards-state-action) algorithm that updates a policy based upon an agent's interactions with its environment. The algorithm attempts to learn a Markov Decision Process by updating a Q-value function which depends on the current state s_t, the current action a_t, the current reward r_t for choosing that action, the next state s_{t+1} and the next action a_{t+1}.

Most recent work in Deep RL can be categorized into advancements in value function methods, policy search, or a hybrid mix of the two.

In the value function paradigm of RL, significant progress was made by Minh et al. [5] in their work at addressing the assumptions of deep networks. Specifically, neural networks assume that data is I.I.D. but this assumption is violated in many RL applications. Minh's work for deep Q-Networks, introduces experience replay, a mechanism for sampling previous game states. Since consecutive game frames are highly correlated, this mechanism enables practitioners to sample from a varied distribution, addressing the I.I.D. violation. They show empirically that this allows the agent to learn a value function that generalizes well to unseen game states.

Policy Gradient, specifically Karpathy's REINFORCE [4], is a recent attempt of directly estimating the optimal policy, rather than extracting a policy from a value function. Exclusively an on-policy method, this approach is simple, effective and quite stable. Playing until the end of an episode, this method is able backprop the gradient of the cross-entropy loss of the action the agent selected from its distribution relative to the distribution itself, using the reward at the end of the end of the episode to indicate the direction of the gradient. However, a limitation of this approach is that the method cannot leverage off-policy data, which may be more data efficient.

Actor-Critic (AC) is a hybrid approach defining an actor (agent) which learns through Policy Gradient methods, and a critic, which measures the efficacy the agent's policy through a learned value function. The insight with this approach is that one can measure the advantage of a policy relative to an estimate for the rewards produced by that policy. This learned value function ($V^{\pi}(s_t)$) is then used as a baseline function to reduce the variance of the policy gradient. Specifically, the policy gradient is scaled by an Advantage Function, defined as the advantage of action a_t in state s_t:

$$A(a_t, s_t) = Q(a_t, s_t) - V^{\pi}(s_t)$$

Until recently, the state-of-the art approach for actor-critic method has been the Asynchronous Advantage Actor Critic (A3C) [3] introduced by Minh et al. A3C enables multi-agent game play which results in significantly reduced training time, lower policy variance and high agent performance.

Since the creation of the DQN, there have been many extensions to the fundamental ideas. Rainbow [6] is an approach that combines insights from recent work in DQN, addressing performance and limitations of the original DQN algorithm. They introduce Double DQN to mitigate overestimation bias from the maximization step in the DQN algorithm. They also introduce prioritized replay, so that the replay emphasizes replaying memories where the real reward significantly diverges from the expected reward, allowing the agent to adjust itself in response to incorrect assumptions. A clever trick is to use multi-step learning which that looks n steps ahead to determine the action to take.

Actor Critic using Kronecker-Factored Trust Region (ACKTR) [7] is a scalable trust-region optimization algorithm for actor-critic methods. A second order method, this proposed algorithm allows for efficient inversion of the covariance matrix of the gradient; improving sample efficiency and the final performance of the agent. It applies policy gradient with a trust region on generic A3C agents, allowing them to achieve higher rewards, and a 2 to 3-fold improvement in sample efficiency over both A2C and A3C [7].

3 Methodology

3.1 Games

We are interested in profiling deep RL with Atari Games. OpenAIGym provides emulators for many Atari games. For each game, it exposes two interfaces: RAM provides the memory state of the game, and RGB provides a $210 \times 160 \times 3$ image of the screen, reflecting the pixel values that would be rendered for game-play.

We profile two games Pong and Breakout of varying difficulty. While the size of the action space for each game is the same (3), the major differentiating factor for complexity comes from the delay in reward assignment. We also experimented with other games such as *Space Invaders*, but due to frame-rate difficulties (the frame-rate of the emulator matches the period of the flashing lasers rendering the lasers invisible to our agent), we abandoned this effort.

Game Name	Difficulty Score	Category
Pong	2	Easy
Breakout	3	Easy

Figure 1: Game Difficulties

Pong is a simple game with only three possible actions at any given point in time, move up, move down, and no op. The screen is always based on the same display, so there are no changes from round to round. The score of Pong is the difference in the amount of wins between the user and the computer, where each win is comprised of 21 point rounds. Based on this simplicity, we have assigned Pong a difficulty score of 2.

Breakout is also a relatively simple game with only three possible actions at any given point in time, move left, move right, and no op. In contrast to Pong, there is no built-in opponent for the user and instead the network attempts to destroy blocks on the screen to gain points. Breakout is considered more difficult than Pong, as the game involves a longer round with only positive rewards, that can trick the network into believing that it is doing well, when it is actually not making much progress leading the agent to get stuck in poor local optima.

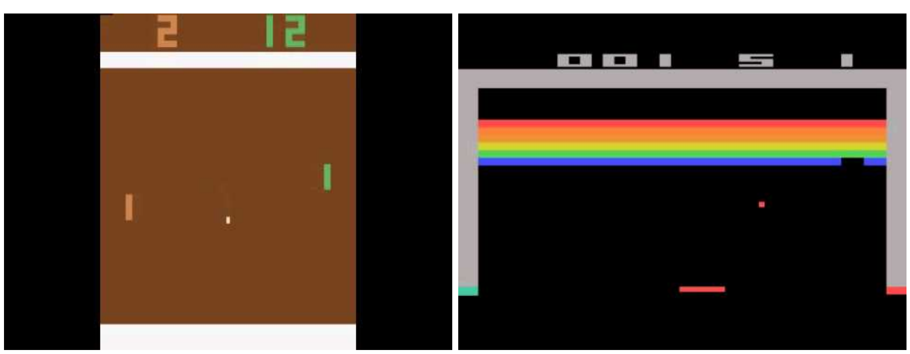

Figure 2: Sample gameplay: Pong (left), Breakout (right)

3.2 Metrics

To compare the performance across experiments, we measure *score_vs_episodes*. We explore both the number of episodes required to reach human level performance and the peak performance achieved after a certain number of episodes specific to each game. For a comparison, we thought it would be interesting to include the performance of random play, SARSA [5], human experts as reported by Minh et al [3], DQN [5], A3C [3], and ACKTOR [7].

4 Models

We implement 3 network topologies:

1. Linear + Actor/Critic (Model 1)
2. CNN + Actor/Critic (Model 2)
3. CNN + LSTM + Actor/Critic (Model 3)

Model 1: Linear + Actor/Critic

The first model, shown in Figure 3, is a feed-forward, fully-connected linear network. The $210 \times 160 \times 3$ image returned by the gym environment at each time step is cropped to 160×160, converted to grayscale, downsampled to 80×80, and stretched to a vector of length 6400. The input to the network is then computed as the difference between the previous frame and the current frame. The network consists of a single fully connected hidden layer with 256 nodes. The hidden layer then connects to two separate heads. The action head is connected through a Softmax to a node for each of the available actions. The value head is a single node representing the value of the current state. The input is connected to the hidden layer through a ReLU activation function. This model is optimized using RMSProp with various learning rates and regularization in the form of weight decay with a coefficient of 0.1.

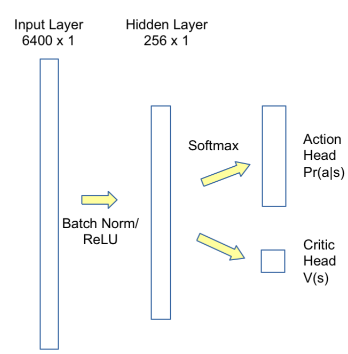

Figure 3: Linear Neural Network Architecture

Model 2: CNN + Actor/Critic

The second model, shown in Figure 4, is a two-layer convolutional neural network. The preprocessing for this model is different from the first. Again the $210 \times 160 \times 3$ image returned by the gym environment at each time step is cropped to 160×160, converted to grayscale, downsampled to 80×80. Instead of vectorizing the frame and taking a difference, this model stacks four consecutive frames as a four channel input to the network. The first convolutional layer has a kernel of size of 8 with stride 4 and padding 2, resulting in the input image being downsized to 20×20. This layer has 16 output channels. The second convolutional layer has a kernel of size 4 with stride 2 and padding 1 which further downsizes the image to 10×10. This layer has 32 output channels. The data is then stretched to a single vector of length 3200 and fully connected to the action and value heads as in the previous model. Between each convolutional layer, batch normalization and ReLU are applied.

This model is optimized using RMSProp with a learning various learning rates and regularization in the form of weight decay with a coefficient of 0.1.

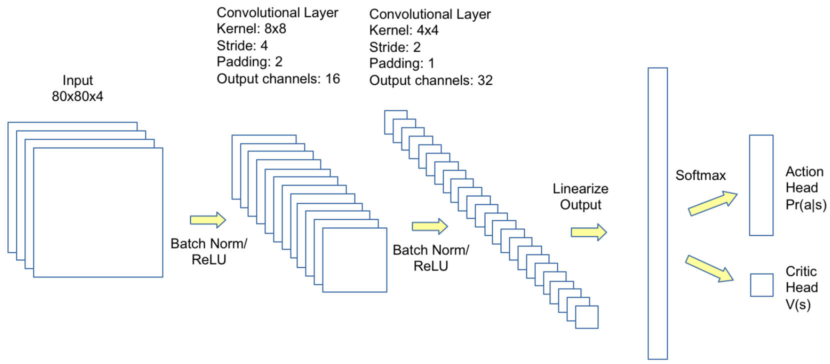

Figure 4: Convolutional Neural Network Architecture

Model 3: CNN + LSTM + Actor/Critic

Traditional algorithms in reinforcement learning make the Markov assumption in which the current state conditioned on all previous states is only dependent on the state from the previous time step. However, this may ignore actions and states from several time steps ago that may have contributed to some future rewards. In the case of Breakout and Pong, the previous states beyond the past four frames may be very important, and it may be the case that a long-run sequence of actions and states could better train a policy over each episode. We draw inspiration from the DQN paper, where "experience replay" is utilized to randomly sample previous transitions to train and backpropagate uncorrelated past behaviors. We hypothesize that using LSTM units with gradient descent can learn which state-action pairs far into the past may have influenced final rewards, resulting in faster and more efficient credit assignment and therefore performance. This hypothesis is supported by the work of Bakker et al [8].

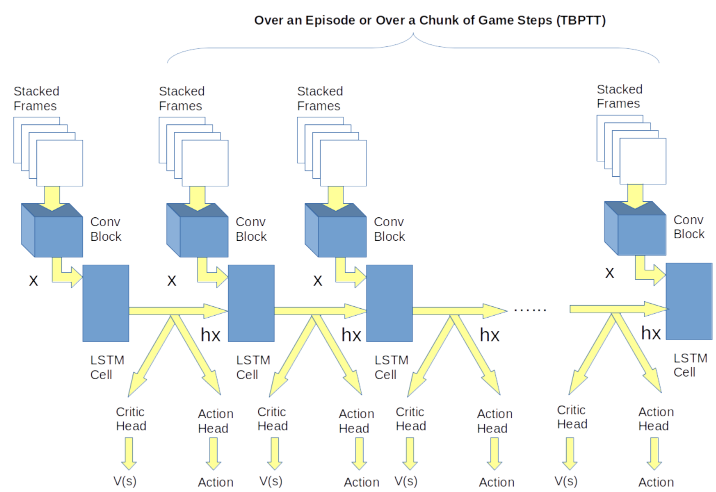

Figure 5: LSTM Network Architecture

The third model, shown in Figure 5, is a three layer convolutional neural network with an LSTM layer. The preprocessing for this model is the same as for the previous convolutional model. The first convolutional layer has a kernel of size of 8 with stride 4 and padding 2 which results in the input image being downsized to 20×20. This layer has 16 output channels. The second convolutional layer has kernels of size 4 with stride 2 and padding 1 which further downsizes the image to 10×10. The third convolutional layer has kernels of size 4 with stride 2 and padding 0 which further downsizes the image to 4×4. Each of these layers has 32 output channels. The data is then stretched to a single vector of length 512 which is fed into an LSTM layer with 256 hidden nodes. The outputs of the LSTM layer are then fully connected to the action and value heads as in the previous model. Between each convolutional layer batch normalization and ReLU are applied. This model is optimized using RMSProp with a learning various learning rates and regularization in the form of weight decay with a coefficient of 0.1.

5 Hyperparameters

In our work, we explore the effects of various hyperparameters on our models. In this section, we report on the various hyperparameters that we experiment with and the motivations therein.

- **Temperature:** We use linearly annealed *temperature* to transition the agent from exploration to exploitation through the state-space. This was inspired by observing that the agent was not exploring much, failing to learn a general strategy effectively. To implement this effect, we take the the output of the network (before *Softmax*) and scale by *temperature*.

$$\text{output-distribution} = \text{softmax}\left(\frac{\text{network-output}}{T}\right)$$

$$T = \max\left(T_{min}, T_{init} - \frac{\Delta_T \cdot \text{episode}}{C}\right)$$

  Where T is temperature, T_{min} is the smallest value we allow the temperature to get, T_{init} is the initial temperature value, $\Delta_T = T_{init} - T_{min}$ and C is the a constant, set for each game ($C_{Pong} = 10000, C_{Breakout} = 50000$).
- **Learning-Rate:** We tune the learning-rate, testing values at $\eta = 1\text{e} - 3$ and $\eta = 1\text{e} - 4$.
- **Gradient Clipping:** While LSTM units inherently mitigate the effects of vanishing gradient, they do not prevent the gradients from exploding during a long training sequence. In order to mitigate this affects of this, we experiment with gradient-clipping at various values for the gradient norm. Specifically we try clipping at $||\nabla|| \in \{1000, 2000, 5000, 80000\}$.
- **Truncated Backprop Through Time:** We found that our experiments with larger (deeper and wider) models had a huge memory footprint to store a full episode, and did not fit in memory. This motivated introducing *TBPTT*. Instead of running multiple forward passes and then one single backward pass though the entire game sequence, we run multiple forward passes and a single backward pass through chunks of the game sequence, while the hidden values of the LSTM hx and cx are carried forward in time between chunks. In experimenting with the *chunk-size* hyperparameter, we tried *chunk-size* $\in \{256, 512, 768\}$.
- **Activation function:** We experiment with replacing the ReLU activation function present in the rest of our models with the SiLU activation function defined as:

$$\text{SiLU}(x) = x \cdot \sigma(x)$$

  We decided to implement the Sigmoid-Weighted Linear Unit (*SiLU*) activation function after having read through the work by Elfwing et al. 2017 [9] and examined properties of the function which show it to be similar to *ReLU*, but instead of having a global minimum of 0, it has a minimum value of -0.28. This global minimum prevents the "dying *ReLU*" issue which is fixed in the leaky *ReLU* function as well. Unlike *ReLU*, at its minimum *SiLU* has a gradient of 0 which behaves like regularization, preventing large negative gradients. An illustration of the *SiLU* activation function which demonstrates these properties can be seen in Figure 6.

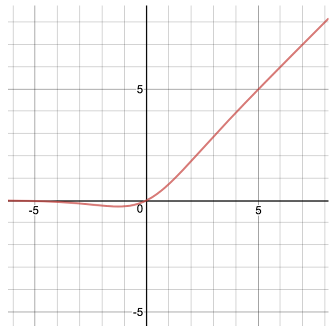

Figure 6: SiLU Activation Function

6 Results

We find that Model 3 (CNN+LSTM+Actor/Critic) is the best network topology for Deep Reinforcement Learning, with a final *running-mean* of 123% and 487% of human-expert level performance on Pong and Breakout respectively. To train the model for optimal performance, it is necessary to optimize two major hyperparameters - *temperature* and *learning-rate*. The *SiLU* activation function is comparable to *ReLU*. We find that it is important to implement Gradient Clipping and that TBPTT is a very promising way to significantly reduce demands on GPU memory.

Agent	**Pong-v0**		**Breakout-v0**	
	Running Mean	% of Human Expert	Running Mean	% of Human Expert
Random Play	-20.7	1%	1.7	5%
SARSA	-17.4	12%	6.1	19%
Human Expert	9.3	100%	31.8	100%
DQN	18.9	132%	401.2	1262%
A2C	19.9	135%	581.6	2411%
ACKTR	20.9	138%	735.7	2314%
A3C (1 agent)*	18.6	131%	61.0	192%
A3C (4 agents)*	15.8	121%	53.6	169%
A3C (16 agents)*	18.8	131%	54.3	171%
Model 2**	**11.2**	**106%**	**49.0**	**154%**
Model 3 (ReLU)**	**16.2**	**123%**	**154.8**	**487%**
Model 3 (SiLU)**	**12.4**	**110%**	**119.2**	**375%**

Figure 7: Comparison of scores for various agents playing Pong and Breakout. *Refers to models implemented by Kostrikov [2] and tested by us. **Refers to models implemented and tested by us.

6.1 Network Topology Experiment Results

We decided abandon further experimentation with Model 1 because the memory demands of the linear layers are extremely high (over 1.3M parameters per game step), and when gradients and intermediary values are accumulated over an episode of several thousand game steps, this results in GPU out-of-memory errors.

In addition, we discover that while the difference of two consecutive images works for Pong (where all of the objects on the screen are constantly in motion). This approach, however, is unsuitable for

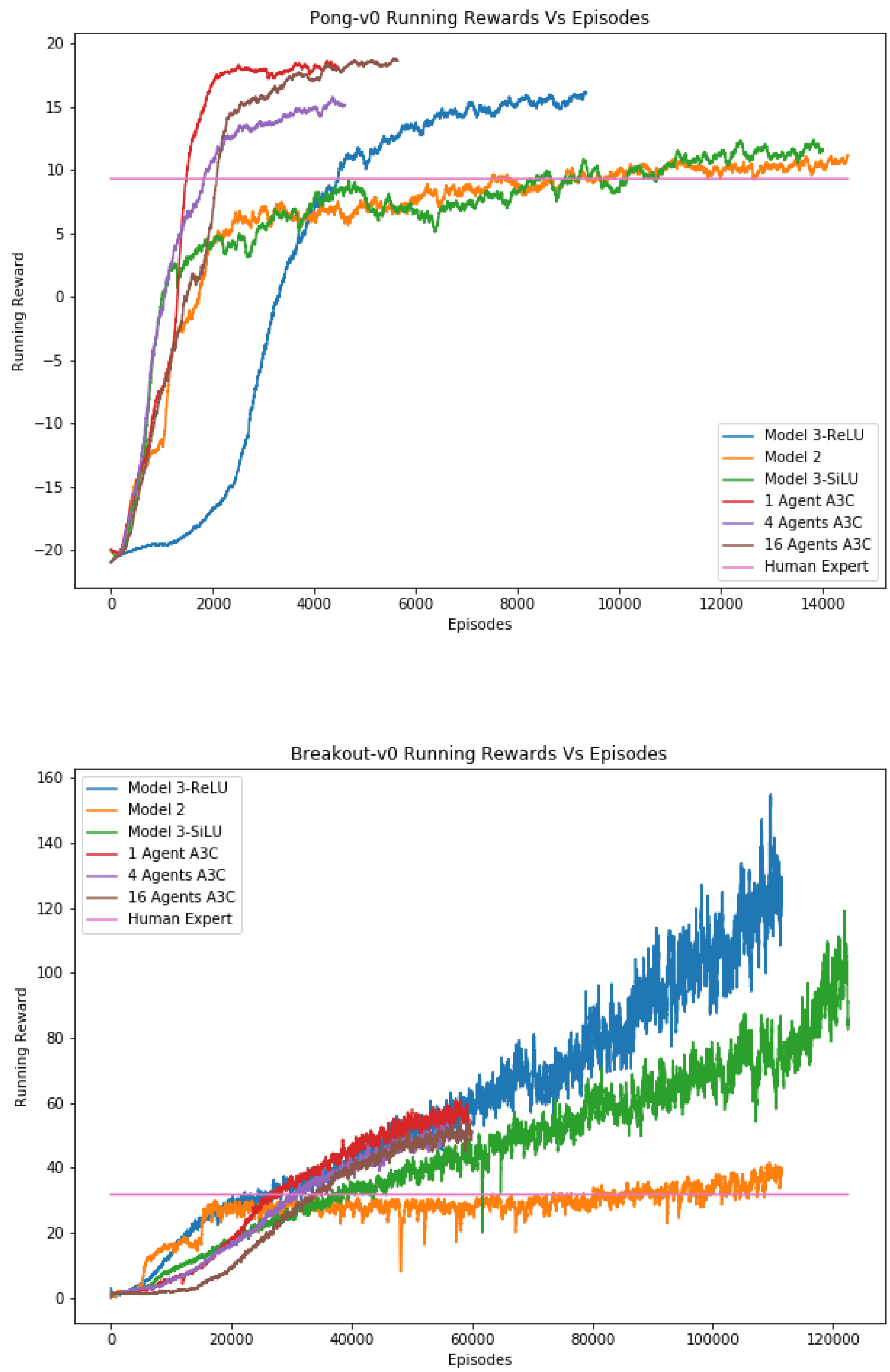

Figure 8: Comparison of results across all models for Pong/Breakout

Breakout because the bricks are not in motion and therefore disappear in the difference image. As a result, we focus our project efforts on Models 2 and 3.

As shown in Figure 8 we can see that Model 3 (CNN+LSTM+Actor/Critic) outperforms Model 2 (CNN+Actor/Critic) in both Breakout and in Pong. This is true in both how quickly it achieves expert-human play, and in its peak performance.

6.2 Hyperparameters

In our exploration of Deep RL, we have learned a tremendous amount about sensitivity of these network setups to hyperparameters, and the trade-off space between learning-time and model generalization and various tricks-of-the-trade that can condition these models to learn better.

- **Temperature:** We used *Temperature* to anneal the state-space exploration rate of the agent. This was inspired by observing that the agent was not exploring much, failing to learn a a general strategy effectively. Specifically, with no temperature annealing, both the Pong and Breakout agents would get stuck at $scores = +1, +3$, respectively. Upon watching the Breakout agent play, we noticed it consistently moved the paddle to the corner of the board where it was able to consistently achieve small positive rewards - and once it learned this strategy, it failed to learn anything else.
- **Learning-Rate:** One may not be surprised to hear that our models are highly sensitive to learning rate. At $\eta = 1\text{e} - 3$, the agent quickly achieves human-level performance, but its progress plateaus and the agent never achieves strong game-play strategy. At $\eta = 1\text{e} - 4$, the agent is slower to reach human-level performance, but continues learning, achieving 487% and 123% on Breakout and Pong respectively.
- **Gradient Clipping:** We implemented gradient clipping to mitigate exploding gradients. We found that with gradients clipped at 1000, and 5000 learning was hindered by the effectively lowered learning rate. With the threshold raised to $80,000$ we did not observe any gradients being clipped.
- **Truncated Backpropagation Through Time:** We found that our experiments with larger (deeper and wider) models had a huge memory footprint to store the activations of a full episode, and did not fit in memory. This motivated introducing *TBPTT*. In exploring the impacts thereof, we find that this speeds up learning because it enables our model to do more frequent policy updates per episode. In tuning the *chunk-size* hyperparameter, we tried *chunk-size* $\in \{256, 512, 768\}$ and found that performance (running reward) improves with higher *chunk-size*. At *chunk-size* $= 768$ our model is able to surpass human-level performance in both games. In general TBPTT agents perform better with larger chunk sizes.
- **Activation function:** We found that using ReLU and SiLU activation functions within Model 3 produced comparable results in Breakout 487% versus 375%. In pong ReLU outperforms SiLU 123% versus 110%, although we tuned hyperparameters for ReLU and carried them over to SiLU hence the comparison is not entirely fair.

It can be seen that it is possible to beat human expert ability in both games across the various methods surveyed in our experiments. Interestingly, actor-critic without LSTM learned very quickly while plateauing around human expert level performance for both games. Meanwhile using LSTM units causes the agent to learn more slowly at first, but performed at much higher levels across both games at around $40,000$ episodes. This suggests that perhaps the network may be exploiting long-run patterns across time for better gameplay performance.

6.3 A3C

Our A3C benchmark uses an existing implementation [2] and beats human experts very easily as shown below in Figures 7 and 8 for both Pong and Breakout.

While using various numbers of agents in the A3C code that we chose to have as a reference point, we discovered a few interesting notes about the benefits of parallelization in neural network architecture design. Even though trials with fewer agents reached higher levels of performance in fewer episodes, as shown in Figure 9, when we look at their performance through the lens of the number of iterations, which are indicative of wall clock training time, the higher agent counts performed markedly better. Therefore, improving and implementing parallelism is conducive to fast and effective networks.

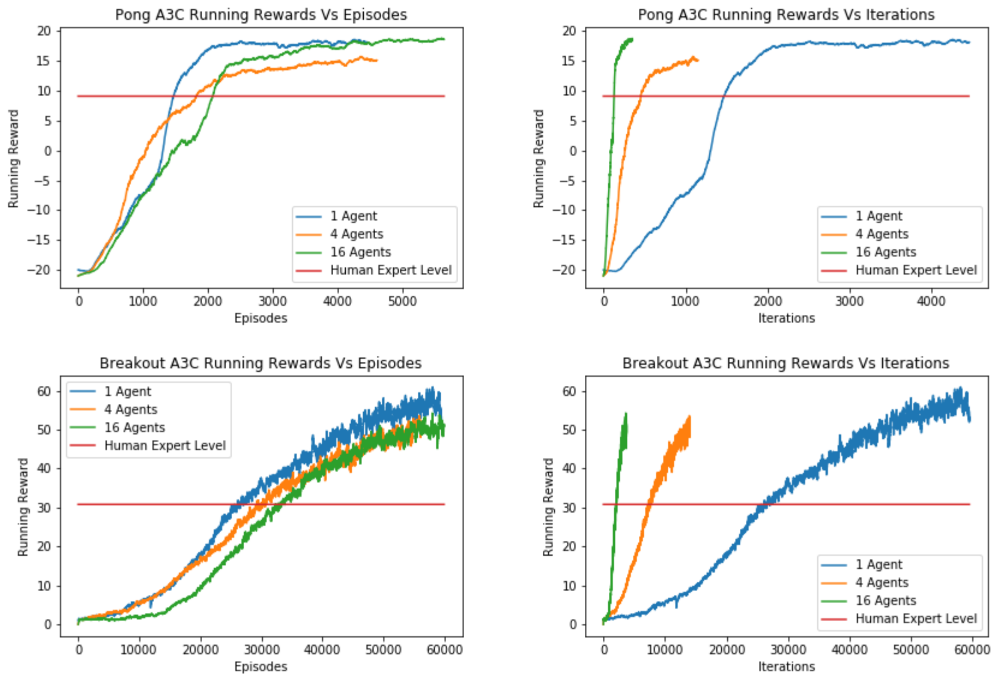

Figure 9: A3C benchmark results rewards versus episodes and rewards versus iterations.

7 Discussion

Replicating and improving upon recently published works in the field of neural networks can be quite challenging due to crucial details being left out of papers. We began with some grand ideas like evaluating our models on many different games and performing meticulous grid searches through all of the hyperparameters we were attempting to tune. We quickly realized that we would have to scale back the scope of our project as each of the models require training time on the order of days even on GPUs. These long training times also limited our ability to iterate on hyperparameter optimization. We also ran into some difficulties comparing our results to published results because the definitions of their metric were not always clear. There is inherent stochasticity in all of these models which made it difficult to isolate the causes of variance in performance. In the future, running each model multiple times with various random seeds and keeping track of which seeds lead to positive outcomes would make our results more consistent and interpretable. Initially we were not saving our Pytorch optimizer along with our models which caused warm starts to lose any momentum that had been accrued.

We were able to successfully implement numerous single-agent actor-critic models which significantly outperform human expert level play. We show that by properly tuning hyperparameters such as temperature, learning rate, gradient clipping threshold, and backpropagation chunk size, this single-agent actor-critic model can, in some situations, outperform multiple-agent A3C models with regard to the common metric of running score versus episodes trained. We discovered that using truncated backpropagation through time, we were able to significantly reduce memory usage while only slightly decreasing performance. Upon deciding to include the SiLU activation function we learned that it is not built in to Pytorch, but we were able to add our own implementation.

In general this project proved to be an interesting introduction to, and survey of, the state of deep reinforcement learning. We look forward to delving into this topic further.

8 Future Works

Given our time constraints, the training duration for each individual model, and the number of tunable parameters there are ample opportunities for future work just systematically exploring the variables we have already considered. One such variable which we would like to delve deeper into is TBPTT chunk size. We saw that as chunk size increases model performance increases but we did not fully explore where these returns begin to diminish. The use of TBPTT greatly reduces the memory required and optimizing TBPTT chunk size would allow us to experiment with larger networks with more parameters. We also noticed that learning rates of $1e-3$ and $1e-4$ performed well during different stages of training so are interesting in exploring learning rate annealing. The A3C [3], A2C [10], and ACKTR [7] papers utilizes policy entropy to encourage exploration, whereas we use temperature annealing, which is worth further exploration as temperature annealing requires knowledge about the rate of convergence in order to set an appropriate schedule.

9 Contributions

Christopher Lamb implemented DQN which did not end up being included in this paper. He also implemented the actor critic baseline driver code which trained all models. He designed the first convolutional model. He implemented the plotting and visualization code used to generate the figures in this paper and the video in the accompanying presentation. He contributed to and edited every section of this paper.

Sean Kinzer did research on activation functions then implemented the SiLU activation function. He was responsible for creating the architecture to combine all the various models into one simple codebase. He also contributed to every section of this paper.

Daniel Reznikov played a key role in the writing of this paper contributing significantly to every section. He analyzed results and pointed other group members in the right direction for potential experiments. He also made the slides for the presentation.

Luke Liem did extensive background research on the state-of-the-art of reinforcement learning. He implemented the LSTM model as well as the TBPTT, and gradient clipping portions of the driver code. He experimented with numerous network topologies and hyperparameter tunings. He also experimented with alternative reinforcement learning methods such as Karpathy's REINFORCE. He contributed extensively to the model description and results sections of this paper.

Christian Koguchi did extensive background research and helped determine which models to pursue. He wrote the related works section of this paper.

Alexander Potapov synchronized the A3C baseline code with our metrics and performed all of the A3C experiments.

References

[1] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[2] I. Kostrikov, "Pytorch implementations of asynchronous advantage actor critic." `https://github.com/ikostrikov/pytorch-a3c`, 2018.

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

[4] A. Karpathy, "(3) deep reinforcement learning: Pong from pixels."

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013.

[6] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. G. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," *CoRR*, vol. abs/1710.02298, 2017.

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

[8] B. Bakker, "Reinforcement learning with long short-term memory," in *In NIPS*, pp. 1475–1482, MIT Press, 2002.

[9] S. Elfwing, E. Uchibe, and K. Doya, "Sigmoid-weighted linear units for neural network function approximation in reinforcement learning," *CoRR*, vol. abs/1702.03118, 2017.

[10] OpenAI, "Openai baselines: Acktr and a2c," Nov 2017.

[11] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.