Vision Enhanced Asynchronous Advantage Actor-Critic on Racing Games

Rahul Palamuttam
Stanford University
450 Serra Mall, Stanford, CA 94305
rpalamut@stanford.edu

William Chen
Stanford University
450 Serra Mall, Stanford, CA 94305
wic006@stanford.edu

Abstract

Modern applications of Deep Reinforcement learning center around game playing. Even though agents like the Asynchronous Actor Critic model from Deepmind generalizes well to playing a variety of games, the performance on individual games is lacking when compared to human level performance. In particular we look at racing flash games like DuskDrive where the state of the art OpenAI universe agent simply learns to go straight to achieve the optimal reward. This behavior is analgous to being stuck in a local minima. The following work demonstrates how deep learning techniques in vision and methods such as immitation learning and visual attention can influence an agent to succesfully perform turns during the DuskDrive game. While our results do not do better than the average reward achieved the OpenAI's agent, our agents are able to either achieve the same convergent behavior faster or learn different behaviors such as avoiding cars or performing turns.

1. Introduction

Since DeepMind's breakthrough in utilizing deep reinforcement learning to learn control policies for various Atari games [13], there has been a lot of interest in developing better models for the task. Learning how to play games is an incredibly interesting problem because games can be thought of as simplified representations of the real world. A model which can learn how to play a racing game may be able to produce insight into the task of getting cars to drive autonomously. Even if the above argument is a stretch, the prospect of getting an autonomous agent to learn how to play various genres of games without any prior information is interesting in and of itself.

For this project, we will explore the effectiveness of image processing techniques when applied to Asynchronous Methods for Deep Reinforcement Learning. Specifically, we will look at using various convolutional neural network models and techniques to improve the asynchronous advantage actor-critic (A3C)[12] method for the purpose of generalizing game play across multiple game instances. We leverage the agent-playable games found in OpenAI Universe. We also use the Gym library which provides a common interface to the variety of environments agents can play in.

Figure 1: The DuskDrive game

2. Problem Statement

The A3C algorithm is effective because it employs parallel actors which each follow a different exploration policy. It essentially aggregates and summarizes the experiences of each actor, which effectively stabilizes training by removing correlations in the data. This differs from a synchronous approach such as experience replay used in Deep-Q-Learning which also removes correlations, but at the cost of more memory and computation. Experience replay requires maintaining a history of state, action, reward, and next state pairs. A batch is then randomly sampled from the history and fed through a Deep Q Network to update the policy gradients.

Thus, our dataset will be comprised of sequences of frames from episodes of the DuskDrive game. Frames will be gathered at each time step by each actor and fed into its corresponding convolutional network and LSTM for feature extraction. The actors will then use these features to predict the next action they should take.

From our initial experiments, we see that the model learns a

suboptimal policy of taking a forward action at every state. Clearly this will lead to issues when the road curves and the agent fails to turn with the road. Our goal will be to try and get the agent to learn to turn when it should.

3. Related Works

Deep Reinforcement learning has taken the Artifical intelligence community by storm. At the crux of these developments is game playing as noted by Mnih et al. [13]. It is important to note that despite recent trends deep reinforcement learning has been used in the past to play games like backgammon [22]. Recent developments leveraged the asynchronous actor critic method [12] which uses inherent agent-level parallelism to train deep reinforcement learning models. We see that these models are limited by their sole reliance on the reward function. We adopt methods from imitation learning [18] to train an inference model which will be used to provide an intermediate reward signal during the A3C game play. One of the motivations behind training game playing agents is achieving human level control as noted by Mnih et al. [14]. While game playing agents can perform in discrete timesteps, deep reinforcement learning has also been shown to be effective in scenarios requiring continuous control [10].

Deep Learning has shown to be effective at a variety of classification tasks, as it leverages representation layers in the form of weights to learn complex models [7]. Of these tasks is video classification, with several works from a variety of domains including geography [11], medical imagery [3], sports [2], and robotics [8]. Game playing is not necessarily a novel application area for Deep Learning, despite its recency. Schuurmans et al. [19] has shown that the Nash equilibria from game theory can be extended to a general neural network architecture. We see that the key differentiator in immitation game playing online is the aspect of vision. Our methods are partly inspired by the work of Yeung et al. [6] [24] in identifying actions in video data. Furthermore, the central component of deep learning vision applications are convolutional neural network. Analysis of CNN architectures have culminated in work by Szegedy et al. [21] whose study of the structural characteristics of the Inception-v4 demonstrated the importance of residual connections and deeper convolutional architectures. Similar work in the study of CNN's is also done by Karpathy et al. [17].

4. Methods

4.1. A3C

The asynchronous advantage actor critic algorithm is a relatively recent innovation by DeepMind introduced in [12]. We leverage the universe-starter-agent codebase [15]

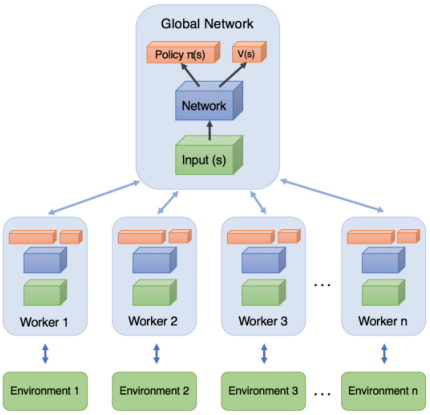

Figure 2: A3C high level architecture [5]

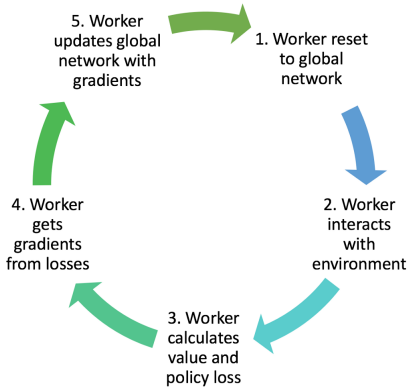

Figure 3: A3C workflow [5]

and build additional functionality around the OpenAI gym framework.

The asynchronous aspect of A3C simply means that the algorithm utilizes worker threads to collect independent experience rollouts while having a global network to aggregate all of the information into its network parameters. The workflow can be broken down into a simple cycle as depicted in Figure 3. At each step, the worker threads get a copy of the global network's parameters and use those to select its agent's next action. Each worker computes its corresponding loss and gradients using the experiences and then the global network's parameters are updated. The benefits of using an asynchronous algorithm are two-fold: there is no explicit need for experience replay as used in the DQN algorithm [13] since rollouts are independent of one another (although it could still be used for data efficiency [23]) and the other is faster data collection.

The A3C algorithm computes advantages for each state-

action pair instead of Q-values. The advantage is defined as the difference between the value of a particular state-action pair and the value of the state. Intuitively, this represents how well taking one action would be compared to taking any other action for some state and allows the agent to know where it should focus its learning. Note that since we do not actually have the Q-values, the algorithm uses the discounted returns from the rollouts to estimate the Q-values.

$$A(s,a) = Q(s,a) - V(s)$$
$$A(s,a) = R - V(s)$$
$$R = r_n + \gamma r_{n-1} + \gamma^2 r_{n-2} + ...$$

With each of the workers, the algorithm keeps estimates of the policy $\pi(a_t|s_t;\theta)$ and value function $V(s_t;\theta_v)$ which are used in the following advantage estimate and losses.

$$A(s_t,a_t) = \Sigma_{i=0}^{k-1}\gamma^i r_{t+i} + \gamma^k V(s_{t+k};\theta_v) - V(s_t;\theta_v)$$

$$\text{Value Loss:} L_v = \Sigma(R - V(s))^2$$

$$\text{Policy Loss:} L_p = \log(\pi(s)) * A(s) + \beta H(\pi)$$

Formulas adapted from [5]

In the policy loss above, an entropy term $H(\pi)$ is used to drive the agent to learn policies which prefer one particular action with high probability over the others.

The entry point to the model will be a convolutional neural network which will take in image frames and output high level spatial features of the game state. The baseline model uses 4 convolutional layers, each with 32 filters of size (3,3) and stride 1. We will experiment with more complex CNN architectures by making use of heterogeneous convolutional layers (varying the filter sizes and counts per layer), pooling techniques to down-sample the data, as well as placing activation functions between the layers.
The output of the CNN is then fed into an LSTM to incorporate the temporal features of the game and the outputs of the LSTM are sent through two separate linear layers to predict the values and policies mentioned above.

4.2. Imitation Learning

4.2.1 Baseline architecture and Improvements

Our imitation learning baseline architecture consisted of a convolutional layer that leveraged an 8x4 kernel and 4x4 stride with 10 filters. This was followed by a two dimensional average pooling layer with an 16x16 kernel and 8x8 stride. The final layer consisted of a simple affine layer, which was then used to calculate the softmax cross entropy. As per convention, the convolution was expected to capture structural feature correlations. We did note that because the input image was not the usual 3 channel volume our results may not match with our expectations. We also did not use relu activation in this architecture, because we wanted to study the full behavior of the gradient for such a simple model. The saliency maps for the baseline model loss can be seen in 4.

4.2.2 Modified Inception Net for Inference

Instead of cascading input through several layers, we decided to apply the Inception Net architecture. Our first inception module consisted of a convolutional layer with an 8x8 kernel, a 4x4 stride, and 10 filters. Instead of utilizing the previous output, we then passed the original image through another convolutional layer with a 4x4 kernel, a 2x2 stride, and 10 filters. Finally, we passed the original input image again through a two dimensional average pooling layer with a 16x16 kernel and 8x8 stride, and a max pooling layer with a 12 x 12 kernel and a 6 x 6 stride. Thus an inception module consists of multiple channel networks each capturing different features and feature granularities of the original input.

The inception architecture then concatenates the results of these layers and passes it through an affine layer, which is then used to calculate the loss. Note that we modified the original inception architecture and removed module internal 1x1 convolutional layers as we though it was unnecessary to reduce channels since we only have one channel to start with. However, we kept the 1x1 convolutional layer after the concatenated result, to reduce the dimensionality. Otherwise, we would run into out of memory errors and could not cascade modules as well.

We then appended two more inception modules to the original one. That is we used the output of one inception module as the input of another inception module. During our exploration we observed that the saliency maps of the loss from the first inception layer were not what we expected. The highlighted regions are faint as noted in 4. As a consequence we decided to change our architecture.

Our optimal model was configured to have an inception module consisting of a 5x5 of the original image, a 3x3 convolution of the original image, the output of a 5x5 convolution followed by a 3x3 convolution and a 1x1 convolution. We also utilize an average pooling layer over the input image. The key difference here is that instead of feeding a max pooling layer over the image, we concatenate the above mentioned layers and then pass the resulting tensor through a max pooling layer. We cascade the output of the module through three more of the same layers. Note that this is very similar to the original inception net module.

When including the models to be used in tandem with the A3C agent, we use the softmax vector from the inference model. We compute an additional loss term using softmax cross entropy. We need the labels to be a distribution since we are comparing with the policy out vector. The final loss

Figure 4: Shows saliency maps from the different loss computations for the inference model and their corresponding images. On the right hand side is the simple baseline architecture. In the middle is the initial inception model. Note that it the model gradients are affected by relatively few points on the input frame. On the far left is the best model. Note that the extra maxpooling layer following the filter concatenations are able to bring to the front features that would otherwise be glossed over in a simple 1x1 convolution.

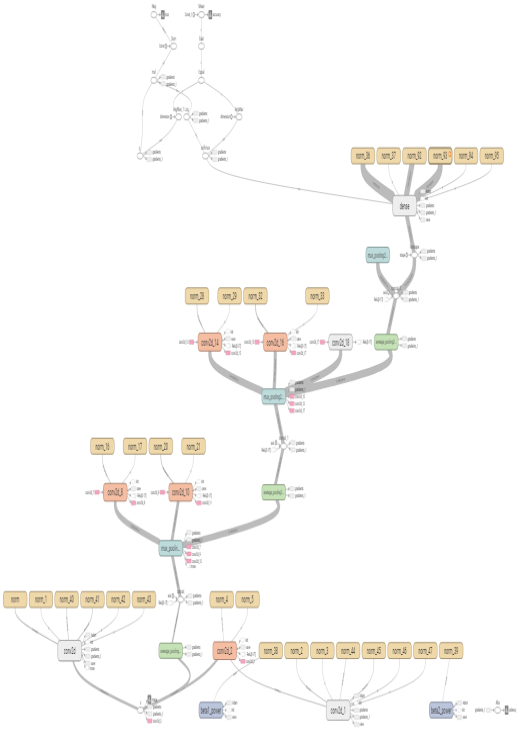

Figure 5: Final inception network architecture

equation is given below.

$$\text{Modified Loss with Inference Signal:}$$
$$L = L_p + 0.5 * L_v - 0.01 * E + L_i$$

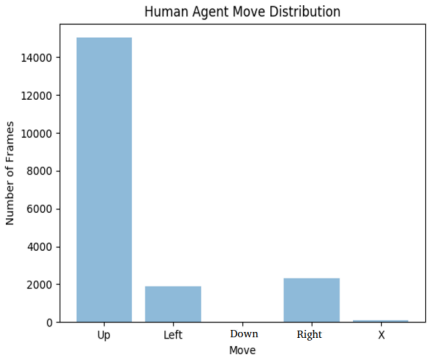

Figure 6: Distribution of moves from human agent playthrough

4.2.3 Data Pre-Processing

In order to generate an appropriate dataset to train our Inception net, we needed to implement additional functionality that was not there in the OpenAI Gym [4]. Internally, the Gym framework leverages the selenium framework to capture and send keys to the browser. However, there is no functionality to retrieve keys pressed by a human agent and pass that data to the python runtime. We realized, the key capture had to occur in the python runtime, before it was sent to the browser.

First, we needed to extract frame and key pairs from several game plays. In order to do this we built a key capture framework that routed keys from Standard Input to the

browser. The framework also recorded the observation data seamlessly. The modified gym framework was then used to manually play through six instances of DuskDrive level 1, which generated approximately 10 gigabytes of video data. The first three instances were used as the training set and the latter three as the test set. Note that each dataset consisted of pairs of frames and their corresponding key vectors.

After noting the distribution of keys pressed, we observed that the key vector distribution was skewed towards the 'up' key 6 This is a consequence of a human agent playing optimally for a game that consists of seven turns. We needed to make sure that keys were equally represented in the distribution of data points. We did so by replicating the underrepresented key-frame pairs. Finally, in order to be compatible with the A3C agent which processes frames as a 2D tensor rather than the normal 3D tensor (including rgb channels) we also reduced the resolution of the image to be 128 x 200 and removed of the extra channels resulting in a black and white image. This reduced our dataset size down from 10 gigabytes to 100 megabytes.

Importing the pre-trained model to be used in A3C proved to be a challenge as well, given the distributed nature of A3C. We needed to reduce the number of worker agents, as we encountered a Tensorflow bug when importing an external model in different session instances at the same time. In short, the pre-processing setup to incorporate raw video-frames was substantial.

4.3. Spatial Softmax

We experimented with a spatial softmax layer as introduced by Levine et al [9] to see if it could improve the baseline A3C convolutional network's ability to represent feature coordinates, thus improving both the value and policy estimates. The layer would aim to achieve this by first computing a distribution over each of the activation maps from the output of the CNN and then performing an expected value computation over each of the distributions. This effectively transforms the feature representation from pixel space into spatial coordinates which may better serve the fully connected layers that follow. Another benefit could be that the softmax operation allows the network to differentiate between strong activations and low ones, giving it the ability to ignore such distractor signals.

Contrary to all the benefits that the authors detailed, the spatial softmax layer had little effect on the A3C model's performance. The augmented model still converged to the same value, but at a slightly slower rate. While this result was discouraging, it led us to explore other modes of measuring attention which we detail in the next section.

4.4. Attention Network

We adapted the soft attention mechanism used in the Deep Attention Recurrent Q-Network model [20] into the A3C baseline with the goal of allowing it to leverage more efficient spatial features of the input frames. The idea is that with attention, the agent could focus on small, yet important sections of the input in order to determine how it should act next.

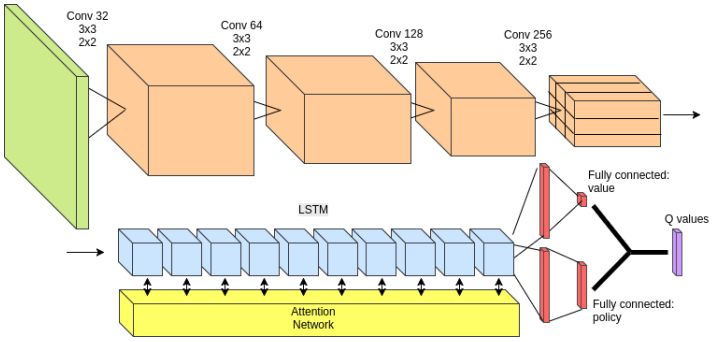

Figure 7: A3C with attention net architecture

Soft attention works by computing a weight for each spatial region of the CNN output. In this case, for each element of the batch there would be $H' * W'$ vectors, each of which has dimension $\mathbb{R}^C$ where the output of the CNN has shape $(batchsize, H', W', C)$. Each of these $v_t^i, 1 \leq i \leq H' * W'$ vectors represents a spatial location within the frame. The context vector z_t is therefore a weighted sum of the v_t^i vectors with weights $g(v_t^i, h_{t-1})$. The weights can be computed as follows:

$$g(v_t^i, h_{t-1}) = exp((tanh(v_t^i A_1 + b_1 + h_{t-1} W)) A_2 + b_2)/Z$$

The spatial vectors can be combined along with the weights to get the context vector:

$$z_t = \sum_{i=1}^{L} g(v_t^i, h_{t-1}) v_t^i$$

At each time step, a context vector is then fed into an LSTM cell and the resulting next hidden state is used in the computation of the weights in the next time step.

One clear benefit of computing the attention at each time step is that we are able to visualize the weights of each spatial location as seen in Figure 8. We take the low resolution weights mapping and upsample it to the original input frame size. Finally, overlaying the attention map onto the input frame gives us some intuition about what our model is learning.

5. Dataset and Features

The OpenAI Universe environment automatically deals with preprocessing of the frames from the game by resizing them and eliminating all channels except one. They are

Figure 8: Attention map overlaid on input frame

Figure 9: Universe DuskDrive environment input frame

modified to have dimensions (128, 200, 1) as shown in Figure 9. For the main A3C model, these processed frames are used directly while we manually preprocessed a dataset for the inference model as described in section 4.2.3.

6. Experiments and Results

The main metric we used for evaluating the A3C model was the average reward per episode that the agent achieved. The inference model uses a different metric while training as detailed below.

6.1. Baseline

The baseline with 8 worker threads was able to converge to an average per episode score of 350000 as shown in Figure 10. The baseline uses an Adam optimizer with learning rate 1e-4 and batch size of 20 experience samples. It learns the policy of taking a forward action for every state.

6.2. Attention Network

The A3C with attention network model converges to the same average reward per episode score as the baseline as well as the same policy, but converges much faster and with fewer workers. A plot of its performance compared to the baseline is given in Figure 11. It uses the same

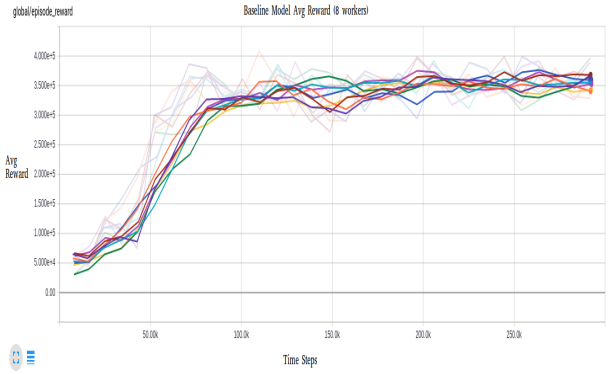

Figure 10: Baseline Model Avg Reward

hyperparameters as the baseline except: the weight of the entropy loss term β is decreased from 0.01 to 0.001 because we initially observed that the entropy term was dominating the loss entirely. We also increased the number of filters in the convolutional layers from 32, 32, 32, 32 to 32, 64, 128, 256 in order to give the attention network a hidden state of 256.
Figure 12 compares the attention network with a baseline model that uses the modified hyperparameters (entropy weight and filter counts). This shows that the hyperparameters are not the cause of the faster convergence observed with the attention model.

Another interesting result worth noting is depicted in the saliency map in Figure 13. Note that the turn arrow signs which appears before and during turns has some impact in the computation of the policy loss. However, why the agent is not able to learn to turn correctly is not completely clear to us. We suspect that it is because the state-action space is too complex for the model to learn the optimal policy directly without any intermediate signals. In implementing the code for this, I used suriyadeepan's rnn-from-scratch code base [16].

6.3. Inception Powered Inference

We now present the results from the three inference models we used to provide intermediate reward signals to the A3C agent. We evaluate our results using three metrics, first the accuracy of the inference model on the test and training dataset generated from the human agent playthrough. Second we evaluate the performance of the inference model as the sole decision maker for playing DuskDrive. We also

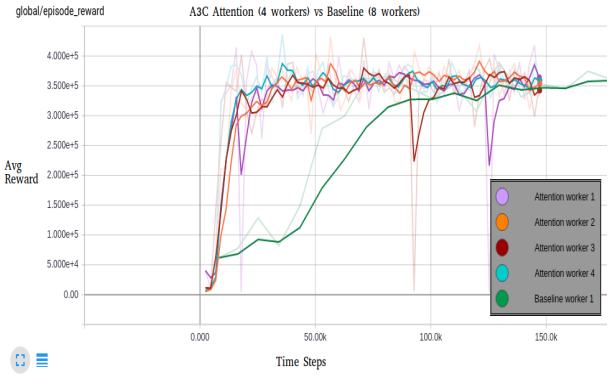

Figure 11: Attention Model: Avg Reward

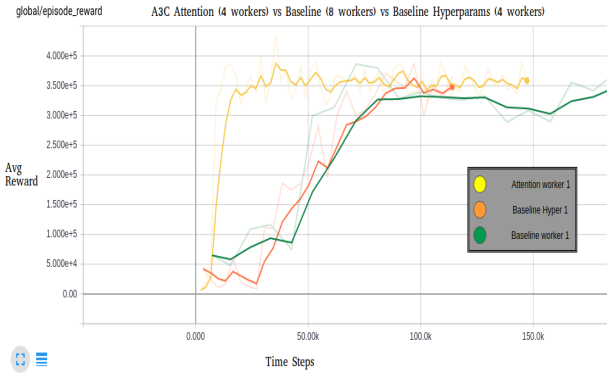

Figure 12: Attention Model: Comparing Hyperparameters

Figure 13: Attention Saliency Map

evaluate the model in tandem with the A3C agent. Our final evaluation consists of observing game-play, as our goal is to ensure that the agent makes turns.An important caveat to note here is that we could not evaluate the A3C agent in multi-agent mode due to inherent challenges with distributed Tensorflow [1] and replicating checkpointed models.

The results of the models on the generated dataset are shown in 14. As expected the inception models are substantially (about 30 percent) more accurate than the baseline model. However, it is interesting to note that the two variants of the inception model converge to the same accuracy.

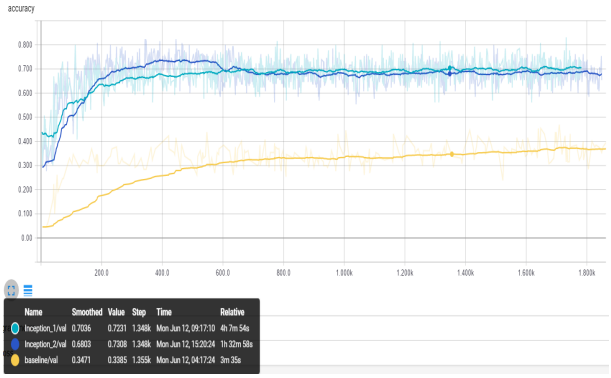

Figure 14: The Inception_1 plot shows the validation accuracy of the initial inception network. The Inception_2 plot shows the validation accuracy of the second inception network plot. Note both networks have a far greater performance than the simple baseline model.

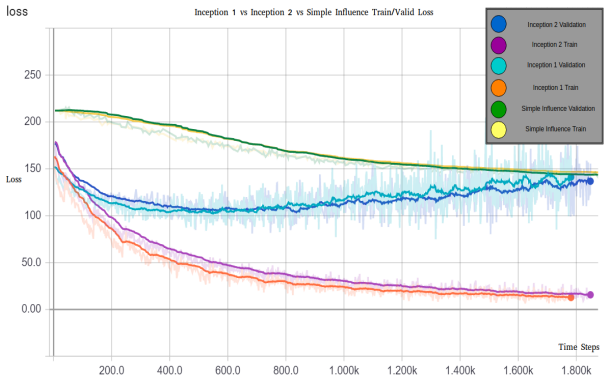

Figure 15: The corresponding training and validation losses

Next we evaluate the trained models as the sole decision maker while playing DuskDrive. We note that these results were quite poor. From simple observations we see that neural network by themselves are not adequate for solving problems in real time asynchronous systems. Furthermore, the number of states the human agent would have needed to explore and have exemplar data points for would have been too many. Thus the experiment demonstrates why we need reinforcement learning for playing games. We did however, test the model in tandem with human agent key presses. The model did well on ideal scenarios where it was in the middle of the road during a turn or avoiding a car that is coming up to it. However, during non-ideal scenarios the model would be stuck and unable to move often pressing 'up-right'. During these scenarios we coaxed the model back into an ideal position by pressing keys. We observed that providing moderate human guidance was enough for the model to finish the race.

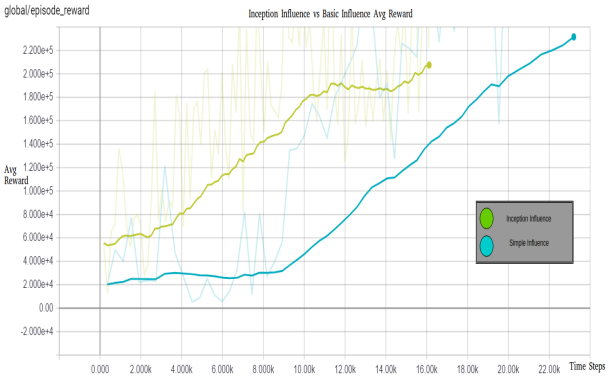

Figure 16: The above figure shows the score of a single AC3 agent instance. The yellow line represents the inception-net powered agent while the pink line represents the baseline powered agent.

For the final evaluation we compare the baseline in tandem with the AC3 model in 16, and the modified Inception network. We do not compare both inception networks as their results are quite similar. We also discovered an interesting property. Initially, we had the keys in the inception net following the same one-hot vector mapping as the A3C agent. However, we noticed that by swapping the mapping representation the model did exponentially better. We observed it avoid cars, press the 'x' key to speed up, and for the most part make turns. The model covered a greater distance than the A3C agent without the additional inference signal, with some instances almost finishing the level. We believe that because we swapped the key-vector embeddings the model was given a greater hypothetical key-vector space to optimize over rather than the simple 6 element vector. An important caveat here is that the model is still about 200,000 points away from the baseline A3C score. One explanation for this is that the 'x' key actually speeds up the game, giving less time for the model to gain points. Other reasons include an incomplete dataset. We believe the inference model could be far more effective, if we played had data from playing hundreds of games rather than twelve.

7. Conclusion and Future Work

In exploring ways to teach a reinforcement learning agent to learn how to play the DuskDrive game, we have touched on many aspects and techniques lying at the intersection between visual image processing as well as deep reinforcement learning. As a baseline model, A3C performs exceptionally well considering the complexity of the task at hand. Our first approaches to improving on the baseline amounted to investigating ways to help the model learn better visual and spatial representations of the state space. When this itself turned out not to be enough, we considered ways to help guide the agent through the learning process by solving a simpler, human guided problem and combining the results in an ensemble-like fashion. Additionally, we looked into several ways to help visualize the learning process in the form of saliency maps and attention weights. While none of the models we built upon were able to strictly beat the baseline model in terms of average reward per episode achieved, one of our models (A3C with attention network) was able to learn much more quickly while the other was able to learn how to utilize turns in ways the baseline was not.

We would like to work further with integrating the influence models with A3C. Currently, we are not able to run the A3C model with more than 1 worker thread when importing the inference model as a meta graph. Running A3C with only 1 worker could lead to instability in the training since the lack of any experience replay or asynchronicity could cause the model to fail to converge. Once there is more support for distributed Tensorflow, we would like to further tune the model to get it to learn how to make turns successfully.

8. Acknowledgements

We would like to thank the teaching staff and Professor Fei-Fei for putting together a great course on convolutional neural networks!We would also like to thank Justin Johnson and Serena Yeung for being amazing lecturers and running the course like a pro. We would also like to especially thank teaching assistant Russell Kaplan who spent time during office hours to explain how he beat the Montezuma game in CS224N. We adopted a similar approach with our inference model, to provide a intermediate signal for our agent to learn.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] M. Baccouche, F. Mamalet, C. Wolf, C. Garcia, and A. Baskurt. Sequential deep learning for human action recognition. In *International Workshop on Human Behavior Understanding*, pages 29–39. Springer, 2011.

[3] Y. Bar, I. Diamant, L. Wolf, and H. Greenspan. Deep learning with non-medical training used for chest pathology identification. In *SPIE Medical Imaging*, pages 94140V–94140V. International Society for Optics and Photonics, 2015.

[4] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[5] A. Juliani. Simple reinforcement learning with tensorflow: Asynchronous actor-critic agents. `https://medium.com/emergent-future/simple-reinforcement-learning-with-\tensorflow-part-8-asynchronous-actor-\critic-agents-a3c-\c88f72a5e9f2`, 2016.

[6] Q. V. Le, W. Y. Zou, S. Y. Yeung, and A. Y. Ng. Learning hierarchical invariant spatio-temporal features for action recognition with independent subspace analysis. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3361–3368. IEEE, 2011.

[7] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[8] I. Lenz, H. Lee, and A. Saxena. Deep learning for detecting robotic grasps. *The International Journal of Robotics Research*, 34(4-5):705–724, 2015.

[9] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *CoRR*, abs/1504.00702, 2015.

[10] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[11] F. P. Luus, B. P. Salmon, F. van den Bergh, and B. T. J. Maharaj. Multiview deep learning for land-use classification. *IEEE Geoscience and Remote Sensing Letters*, 12(12):2448–2452, 2015.

[12] 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*, abs/1602.01783, 2016.

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

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[15] OpenAI. universe-starter-agent. `https://github.com/openai/universe-starter-agent`, 2016.

[16] S. Ram. rnn-from-scratch. `https://github.com/suriyadeepan/rnn-from-scratch`, 2017.

[17] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[18] S. Schaal. Is imitation learning the route to humanoid robots? *Trends in cognitive sciences*, 3(6):233–242, 1999.

[19] D. Schuurmans and M. A. Zinkevich. Deep learning games. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 1678–1686. Curran Associates, Inc., 2016.

[20] I. Sorokin, A. Seleznev, M. Pavlov, A. Fedorov, and A. Ignateva. Deep attention recurrent q-network. *CoRR*, abs/1512.01693, 2015.

[21] C. Szegedy, S. Ioffe, and V. Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, abs/1602.07261, 2016.

[22] G. Tesauro. Neurogammon wins computer olympiad. *Neural Computation*, 1(3):321–323, 1989.

[23] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas. Sample efficient actor-critic with experience replay. *CoRR*, abs/1611.01224, 2016.

[24] S. Yeung, O. Russakovsky, G. Mori, and L. Fei-Fei. End-to-end learning of action detection from frame glimpses in videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2678–2687, 2016.