Comparing Attention Mechanisms on Car-Racing Variants

Amrith Arunachalam
Columbia University
IEOR 4540
aa4053@columbia.edu

Abstract

Implicit and Explicit Attention have in previous works shown to have trade-offs between performance and generalizability. Through this report we propose a new method of attention that draws from both implicit attention and explicit attention to leverage the benefits of both. We test these results on the Car-Racing Simulation Environment and its variants. We demonstate that the theoretical performance gain exists in our empirical testing.

1. Introduction

The Transformer Architecture was devised to help in the language processing space but has since seen application to various other domains including vision-based learning problems. For our uses, we will be looking at the attention mechanism originally proposed by [5]. More specifically, we will be comparing the explicit attention bottleneck approach taken by [4] with the implicit attention for pixels algorithms proposed by [2].

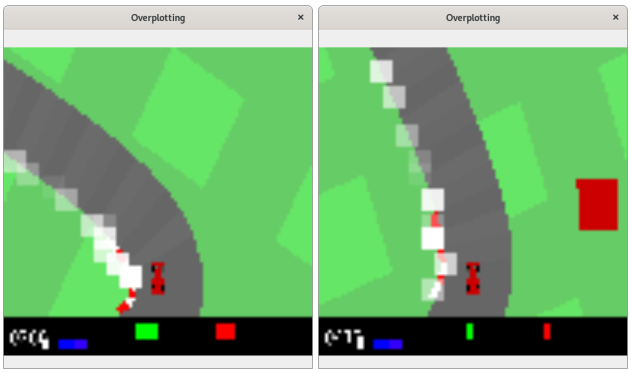


Figure 1. Car-Racing simulation and its variants

1.1. Self-Attention Bottleneck

The purpose of the attention mechanism is to learn how the input interacts with itself(through ES techniques) and help the controller weigh the more important areas using an importance matrix. The controller learns which of the inputs to pay more "attention" to and follows the neurovolution seen in humans. The output of the attention layer is the importance or attention to be placed on a given pixel/patch in a image. The output is sent to a controller module that controls our agent's actions. The use of an attention layer inroduces a performance bottleneck for the Transformer architecture. Self-attention as calculated in [4] is as follows:

$$A = softmax(\frac{1}{\sqrt{d_{in}}}(XW_k)(XW_q)^\top)$$

We note that in the process of calculating this attention matrix we take a softmax of a matrix size $A \epsilon R^{nxn}$ in addition to manually computing and storing the complete matrix. This encourages practitioners to reduce the size of the attention matrix as much as possible. While there are some benefits to using a smaller attention matrix (better generalization), there are certainly cases where being able to use smaller patches and even using pixels in the Attention matrix would be useful.

1.2. Implicit Attention For Pixels

The work performed initally by [1] showed a new technique for performing attention and generalized the concept such that they introduced kernel transformations. This meant we could approximate the softmax used by [4] with the performance gains of using the FAVOR+ algorithm. This next step is shown in the development of the IAP-Rank algorithm which uses the attention bottlenecks found in [4] in combination with the FAVOR+ algorithm to produce faster training and equal if not better performance in tests. As compared to the method used by [4], the attention matrix calculated via FAVOR+ goes as folllows:

$$\hat{Att}(V) = \Xi((P_{1,L}Q')(K')^\intercal, V)$$

Instead of calculating the Attention matrix outright, by disentangling Q' from K' we compute the input in linear(as compared to quadratic) time and space. In this report we not only compare explicit attention and implicit attention but we

also compare two kernels used with IAP-Rank ($softmax$ and $ReLU$).

In this report, we focus on comparing IAP-Rank with [4] as we feel that this comparison is the most natural and will lead to interesting discussion.

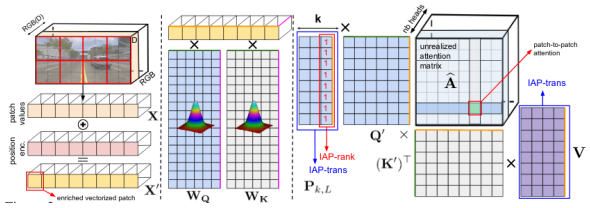


Figure 2. IAP Architecture [2]

2. Method

2.1. Implementation Details

Google Brain Tokyo has kindly provided a github repository containing the code to reproduce the results they achieved in their paper[4]. As a starting point, we initially trained our model locally to get a baseline accuracy and model performance. Interestingly, without changing any hyperparameters and following the same steps as the authors, our model performed slightly worse. We believe the authors have reduced the hyperparameter weights from what they used to train the original models (reduced iterations of ES). From this starting point, we then found an implementation of Performers [1] and refactored the code provided by [3]. Using the provided FastAttention class as a base, we were able to implement IAP-Rank as detailed in [2]. To test the different kernel implementations we changed the attn_fn variable to reflect the appropriate kernel.

2.2. Controller

Table 1. Config Variables

Variable	Values
VisionTaskSolution.patch_size	[1,3,5,7]
VisionTaskSolution.patch_stride	4
utility.create_task.modification	['original','color', 'bar','blob']

We take the output from our attention mechanism and use it to train our controller that determines the movement of the car. For consistency across different experiments, we used the same controller architecture as suggested by [4] with modifications to the input shape to the LSTM layer and the multi-layer perception class. These modifications allow us to experiment with various patch sizes from as granular as a pixel to as general as the whole image. For our different experiments we adjusted the variables found here 1

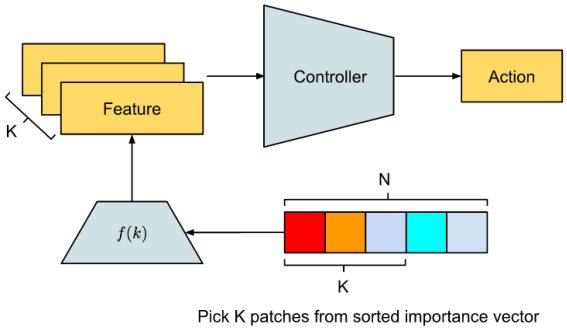


Figure 3. Attention Bottleneck Architecture [4]

2.3. Training Environment

All tests conducted were performed in a GNU/Linux(Debian) VM running on a Windows laptop with an Intel 6-core CPU. This introduces bias in our results as the experiments were not conducted under replicable conditions.
To reproduce the environment please use python 3.9.2 and activate a virtual environment of your choice. From here download the necessary libraries using the following command from the project root directory:

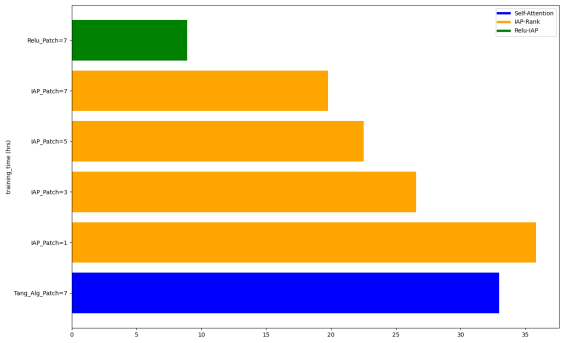
```
$ pip install -r requirements.txt
```

3. Results and Discussion

In this section we compare the performance of explicit attention mechanisms with implicit attention mechanisms from training performance, training scores and testing scores.

3.1. Training Agents

Figure 4. Training Times For Various Experiments



In this section we will discuss the training times for various attention mechanisms. Our testing methodology is as

follows, we used the same Transformer RL model for all models only replacing the attention mechanism for the 3 varieties. We started by using the Self-Attention Bottleneck described by [4] which took the second most amount of time. We then used our implementation of IAP-Rank with the softmax kernel approximation and 25 random orthogonal projections. See 4 for specifics regarding each individual trial, overall we see that as expected the training time decreases as we increase patch size, this makes intuitive sense as the attention matrix decreases quadratically by k^2 as we increase patch size by k. Finally, as a novel new concept trial, we used the FastAttention layer's ability to use different kernels to test using a ReLU kernel approximation and found that the model had very strong training performance but suffered significantly in the test results.

3.2. Testing Agents

In this section, we discuss the results achieved by various agents on OpenAI Gym Car-Racing and it's variants. There are 3 major variants, Self-Attention bottleneck as provided by [4], IAP-Rank Softmax by [2] and IAP-Rank ReLU by [2]. Each model is trained on the original track and tested on variations of the original track 10 times. We record the mean and the variance for every model tested. In the root directory for this project there is a folder called videos that has videos of trained agents performing the vision task. Because we are testing IAP-Rank, we kept the 10 most important patches as described in the paper.

Table 2. Test Scores on Original Track

model	score
Original Tang	914 ± 15
IAP Softmax Patch=1	803 ± 210
IAP Softmax Patch=5	751 ± 51
IAP Softmax Patch=7	723 ± 32
Tang Patch=7	712 ± 53
IAP Softmax Patch=3	703 ± 91
IAP ReLU Patch=7	599 ± 192

The smaller our patch size the better our model fit the training environment. However, it should be noted that there is also increased variability in our model's performance with decreased patch size. We believe this is because the bottleneck is not being adjusted. If we were to test multiple bottleneck sizes, the number of experiments would grow by a factorial and we leave adjusting this parameter as an exercise for interested readers to test out. Because the bottleneck was left at size $l = 10$, we kept the l most important patches as determined by our attention mechanism. Intuitively, as the patch size decreases, the variability in our model increases as the layers following our attention module will have less information to work with than if the patch size was strictly larger. If our goal was to make our model perform optimally in a specific training environment, then removing the attention bottleneck and providing the following layers with the whole image as the importance matrix would perform better than using a bottleneck. This however, is not the goal of this task as we want our agent to be generalizable to variation in environment such as the color, bar or blob variation to Car-Racing.

Table 3. Test Scores on Color Track

model	score
Original Tang	866 ± 115
IAP Softmax Patch=1	783 ± 325
Tang Patch=7	717 ± 142
IAP Softmax Patch=7	705 ± 132
IAP Softmax Patch=3	703 ± 191
IAP Softmax Patch=5	692 ± 147
IAP ReLU Patch=7	509 ± 315

3.3. Car-Racing Variants

3.3.1 Color

We can see from 3 that the results on the color changed variant of the Car-Racing problem follow the results of the original training environment. This follows intuition as we do not expect our model to rely on absolute pixel values for color, but rather the difference between pixels at the boundaries. This can be most clearly seen by looking at 1, we can see that while the car is moving forward, we see that the attention patches are all along the left side of the road, meaning the car's control module is making navigation decisions based on where the left side of the road is. This introduces an interesting variable in the environment that may be worth exploring where the road on the left side goes to the edge of the screen. Additionally, because all of these models are trained on a counter-clockwise track, the attention module makes the agents follow the left side of the road for control decisions. We postulate whether training a model on a counter-clockwise track would make the model have importance for the right side of the track? And with these attached biases, is there a way for us to limit the bias by training on both clockwise and counter-clockwise tracks? We assume that the model would perform worse in training environments but may generalize better and would overall be considered a better agent.

3.3.2 Bar

The bar variation of the Car-Racing problem adds two vertical bars either side of the screen, changing the visual experienced our agents. Because all agents use the top $k = 10$ patches from our attention matrix, we can see that the performance of our models do not get significantly affected by this change. We note that the IAP-Rank algorithm with the

same hyperparameters as Tang trains at a significantly faster rate and performs the same. In future experimentation, it may be interesting to set a time allowance for both models to train in and to see how big the performance gap is. Based on results from these experiments, we are led to postulate that IAP-Rank would be heavily favored.

Table 4. Test Scores on Bar Track

model	score
Original Tang	898 ± 53
IAP Softmax Patch=1	804 ± 205
IAP Softmax Patch=5	758 ± 53
IAP Softmax Patch=7	720 ± 37
Tang Patch=7	708 ± 43
IAP Softmax Patch=3	698 ± 93
IAP ReLU Patch=7	571 ± 186

3.3.3 Blob

The Blob variant of the Car-Racing problem has a floating rectangle in the peripheral vision space seen by our agents. This would have a significant impact on RL agents that do not utilize an Attention Bottleneck, as those agents would have to process those pixels. Because both models ignore information outside of the most important patches, we see that while the performance has decreased, the agent's are still performing well. Our results fall in line with the performance seen by the initial model used in [4].

Table 5. Test Scores on Blob Track

model	score
Original Tang	900 ± 35
IAP Softmax Patch=7	792 ± 46
Tang Patch=7	791 ± 51
IAP Softmax Patch=5	749 ± 49
IAP Softmax Patch=1	717 ± 231
IAP Softmax Patch=3	705 ± 87
IAP ReLU Patch=7	503 ± 201

4. Conclusion

In this report we apply the FAVOR+ Algorithm proposed by [1] using the self-attention bottleneck proposed by [4] with IAP-Rank by [2]. We see that for Car-Racing the application of a attention bottleneck provides better generalization as compared to using smaller patch size. However, when we apply the FAVOR+ approximation for softmax kernel, we achieve better results in both training and testing environments than either paper's results. The ability for the model to be trained quickly and it's generalizability makes it a strong candidate to be applied to other vision tasks like other Atari game simulations.

5. Acknowledgements

I would like to thank Prof. Choromanski for teaching this course, it has been a great experience and while I have been taking the course online, I have enjoyed the material greatly. Additionally, I would like to thank all the contributors who worked on making both the brain-tokyo and performer-pytorch repositories. Without their code to base my work on, this project would not have been possible. Additionally, I used the sample template from the 2017 CVPR Proceedings to generate this LaTeXdocument.

6. Reflection

On a more personal note, through the work conducted on this project, I had the chance to read and understand some sophisticated code. I had to understand what different modules were trying to achieve and then change it to suit my needs. As someone who is still learning python, it was a challenging but rewarding opportunity to learn. This project has been an absolute pleasure to work on and I hope my passion for the material has shown in my presentation of the work this semester. This research is amazing and I am excited to see it change the world! Through working on this project, I have a deeper understanding of the optimization problems that need to be solved for RL to be viable and I had the opportunity to apply one such technique to a new task. As an individual with no prior background in research I found that reading technical papers was quite challenging but a great learning opportunity and I enjoyed my foray into research.

References

[1] K. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Davis, A. Mohiuddin, L. Kaiser, D. Belanger, L. Colwell, and A. Weller. Rethinking attention with performers, 2021.

[2] K. M. Choromanski, D. Jain, W. Yu, X. Song, J. Parker-Holder, T. Zhang, V. Likhosherstov, A. Pacchiano, A. Santara, Y. Tang, J. Tan, and A. Weller. Unlocking pixels for reinforcement learning via implicit attention, 2021.

[3] lucidrains. Performer pytorch. `https://github.com/lucidrains/performer-pytorch`, 2021.

[4] Y. Tang, D. Nguyen, and D. Ha. Neuroevolution of self-interpretable agents. *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, Jun 2020.

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2017.