Graph Based Attention for Reinforcement Learning

Achint Kumar Aggarwal
Indian Insitute of Technology
Delhi, India
ee1180433@iitd.ac.in

Amokh Varma
Indian Insitute of Technology
Delhi, India
mt61805272@iitd.ac.in

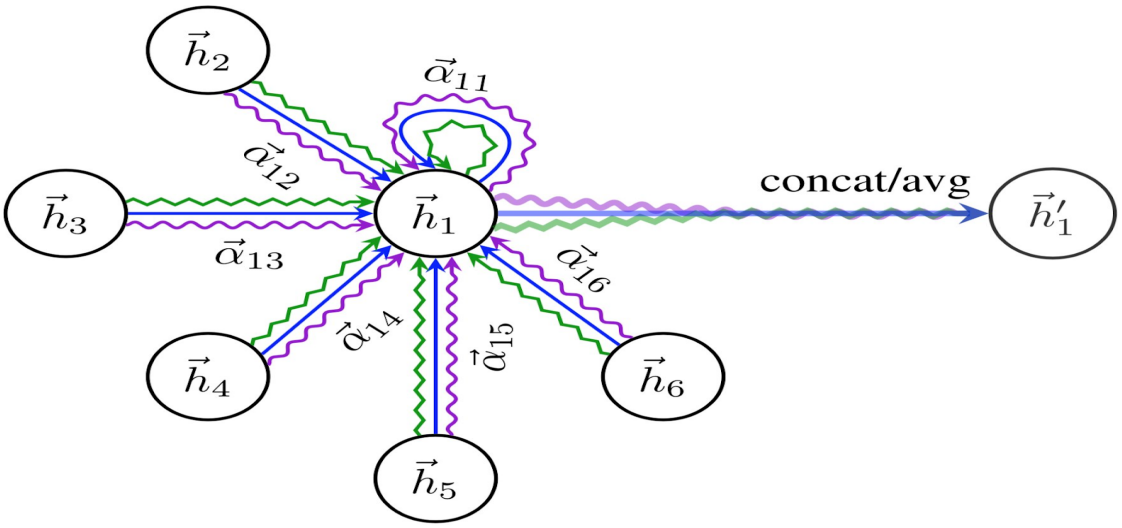

Figure 1. An image of Attention Network from [5]

Abstract

This paper talks about the use of Graph Attention Networks (GATs) in deep Q-learning networks (DQNs). While the aim of network modifications is generally to enhance performance, this paper does not solely aim to do. It instead talks about the insights one can get from the parameters (attention weights, in this particular case) learnt while training. OpenAI's MsPacmanV0 environment has been used for interacting with the model.

Keywords: GNNs, Attention Networks, DQN, Reinforcement Learning

ACM Reference Format:
Achint Kumar Aggarwal and Amokh Varma. 2020. Graph Based Attention for Reinforcement Learning. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 7 pages.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00

1 Introduction

Reinforcement learning is a paradigm of machine learning which allows a given learning agent to interact with the environment and learn a behavior which maximizes the reward it earns while interacting with the environment. Deep Q-learning Networks (DQNs) are very well-known in the reinforcement learning literature. [4] shows how they can even out-perform humans in certain tasks like Atari gaming. There have been further improvements in the field, broadening the array of tasks in which DQNs out-perform existing methods and end up setting new baselines. Despite large strides in the performance ability of the DQNs, only little insights are available in their working mechanism, as is generally the case with many components of deep learning.

Taking motivation from the same, this paper makes an attempt to propose deep learning methods that may help us understand the working of DQNs better. We use one of the games of Atari2006, MsPacmanV0, as our environment. DQNs are known to give extremely good results for the same as has been mentioned above. Graph Attention Networks (GATs) have been used to do the same. The input in our case is an image of the game-screen layout. The same has been treated as a graph to enable the GAT framework we propose to use it. Attention weights will be analyzed in order to understand the relative importance (from the DQN's perspective) of the various local spaces of the input image.

The same has been discussed in more detail in the following sections.

The paper has been structured as follows. Section 2 mentions some of the works which talk about graph-based methods for RL, several others talk about the use of attention mechanism in general for the same. Some previous works on graph-based attention to enhance model performance have also been talked about. Section 3 introduces the reader to the required preliminaries. Section 4 talks about the mechanism(s) we propose in this paper while section 5 talks about the experiments and results obtained from the same. Section 6 concludes the paper and talks about possible future directions of research.

2 Related Works

There have been multiple works which use attention networks for Deep Reinforcement Learning (RL), however, most of them deal with CNN or MLP based bottlenecks. [6] is the main work, which aimed to connect RL with graph based state representation. This paper suggests a simple representation rule where every edge represents a possibility of going from from one state to another . [8] and [7] deal with application of attention networks to RL and planning. [8] proposes 5 different ways of using attention layers in a non-graph setting. Our work partly builds upon these ideas and the representation concept used [6]. [7] works on the hypothesis that these attention layers are a good way to enforce coordination among different agents, in a multi agent setting. Our baseline is taken from [4], which proposes a Convolutional neural networks (CNN) based Deep Q learning network to play Atari. Unlike many of the previous works, our work utilises the flexibility of OpenAI gym [1] , which makes it much easier to build upon and also makes the baselines and results much more general and reproducible. Our implementation can be found here [1]

3 Notations and Preliminaries

3.1 Deep Q Learning (DQNs)

DQNs are a deep learning method used to solve RL problems. The algorithms deals with finding Q-values , which are defined as

$$Q(s_t, a_t) = \mathbf{E}(R_t | S_t = s_t, A_t = a_t)$$

where t refers to time index and R_t is the reward. For the Q learning algorithm, the updates are as follows

$$Q(s,a) \leftarrow Q(s,a) + \alpha * (r + \gamma * \max_{a'} Q()s', a') - Q(s,a))$$

which in DQNs will translate to mapping $Q_\theta(s,a)$ with $r + \gamma * (\max_{a'} Q_\theta(s', a'))$, where θ is the weight. This process is called bootstrapping. We use MSE loss to train it. Our baseline [4],uses a CNN followed by fully connected layers to model $Q_\theta(s,a)$. γ decides the short-sightedness of our method, that it defines whether our aim is maximize short term reward or long term reward.

[1]github link : https://github.com/amokhvarma/GraphAttentionRL.git

3.2 Graph Attention Networks (GATs)

GATs [5] are one of the more well-known Graph Neural Network (GNN) models. GNNs are deep learning based architectures which generally aim to provide the user with embeddings for the constituent nodes of a given graph. Certain models once trained on given graph(s) even have the ability to scale well when run for an unseen graph. This is known as inductive nature of the models, GATs fall under this paradigm. Coming back to GATs, the following equations will help us understand their working.

$$z_i^{(l)} = W^{(l)} h_i^{(l)} \quad (1)$$

$$e_{ij}^{(l)} = \text{LeakyReLu}(a^{(l)^T}(z_i^{(l)} || z_j^{(l)})) \quad (2)$$

$$\alpha_{ij}^{(l)} = \frac{exp(e_{ij}^{(l)})}{\sum_{k \in N(i)} exp(e_{ik}^{(l)})} \quad (3)$$

$$h_i^{(l)} = \sigma(\sum_{j \in N(i)} \alpha_{ij}^{(l)} z_j^{(l)}) \quad (4)$$

GATs can be said to be under the category of l−hop message passing neural networks. By definition, 0-hop embeddings for all nodes are taken as their respective feature vectors. One can then use the above equations to compute the $l+1$ hop embeddings, given the l hop embeddings for all nodes. The l−hop embeddings are denoted by $h_i^{(l)}$ for the i^{th} node. $W^{(l)}$ is a matrix of appropriate dimensions, a characteristic of each layer. $\alpha_{ij}^{(l)}$ are referred to as the attention weights for layer l. They denote the relative 'attention' one must pay to the j^{th} node in the neighborhood of i ($N(i)$) while aggregating the received signals, as done in (4). One can also have multiple attention heads, in which case an extra aggregation over all the heads will have to be performed. $\sigma(.)$ is called the activation function and is subject to the task at hand. The vector a is a learnable parameter which helps us compute the attention weights based on the existing set of embeddings.

3.3 OpenAI gym

OpenAI gym [1] is a python based library which provides environment for lots of tasks and games, which can be used to study and benchmark RL algorithms. We use the Atari2600 environment . More details of the environment are given in Experiments Section.

4 Our Proposal

Motivated by [6] and [7], we decide on 3 types of Graph Representation. For the baseline, we use a CNN based model, as proposed by [4] (Refer to Figure 2 . The 3 representation methods used by us are as follows :

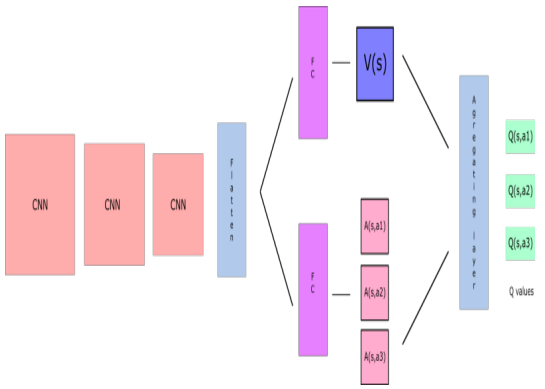

Figure 2. Our baseline. The output models $Q(s, a)$

- **Fully Connected** (see [9]) : The pacman state is in the form of a RGB image (260x130x3) . We downsample it to (50x50x3) or (100x100x3). We use the reduced image to get n^2 nodes, where n is the dimension of the images (50 or 100) and each node represents a pixel. Therefore, each node has a 3 dimensional feature vector.
- **Fully Connected (Partially Observable)** : We only input a smaller sub-image of the whole image, based on where the pacman is. The rest remains as above.
- **Adjacent Connected** : We only make edges between states which can be reached from one another.

These graphs are then fed into Graph Neural Network , like GAT [5], Graph Convolutional Network (GCN) [3] and GraphSage [2] . The embedding dimension be E and number of attention heads be A . So we get a n x E x A output from the GAT (n x E from others) . This is fed into an MLP and the output is taken as a 5 dimension vector. In addition to standard GAT, we also use a model with skip-connections .

5 Experiments and Results

One must expect our modification to not give as optimal results as the CNN based DQN. This is due to decades of experience with the models, wherein CNNs are known to outshine other architectures for almost all visual input based tasks. The purpose of GATs here is not to enhance our model performance but provide us with some insight into the working mechanism of the same. Hence we should not bother much about the aforementioned sub-optimality. However, we will actually be able to have comparable performance, with careful hyperparameter tuning (as we can see from Table 1)

5.1 Given Framework.

We work in the MsPacmanV0 (See 12) environment developed by OpenAI gym [1]. The environment feeds us with an RGB image of the present screen layout (the present state of the environment) and expects us to feedback an action, which renders us some form of reward. This process is repeated over and over until the game ends.

Table 1. All test results for **Experiment 1**

Model	Input	Avg Reward	Avg Episodes
GAT	50x50	74.3264	646.1274
GAT	100x100	81.2314	**682.1241**
GAT-skip	50 x 50	**79.8465**	651.2341
GAT-skip	100 x 100	**81.2641**	679.0165
CNN	100x100	**83.1345**	678.8076
CNN	50x50	**79.2145**	**682.1723**
GCN	50x50	66.1124	623.1085
GraphSage	50x50	75.2342	638.1042
Random	-	52.2314	428.9124

Specifications. Input Image: RGB, 210 x 160 x3; Allowed Actions: Move up, Move down, Move left, Move right, NoOp (Do nothing); Each action is performed for k frames where k is sampled uniformly from {2, 3, 4}; Reward Setting: Retrieving food rewards 10 points; Aim. to maximize the reward by taking the most rewarding action.

Modifications. The input images have been down-sampled to (50 x 50 x 3) and (100 x 100 x 3). The sole purpose of this dimension reduction is to aid our computation which is already expected to be on the higher side for reinforcement learning based tasks. After this dimension reduction, we form a clique with $n \times n$ nodes ($n = 50, 100$). We wish to find out the relative importance of various locations of the image that are taken into consideration while making the decision, hence we use a clique and let the model decide the attention it thinks the nodes/ pixels deserve. The implementation is done using pytorch-geometric. A link to the same can be found in footnote 1 (section 2).

5.2 Experiment 1

In this experiment, we train all our models i.e. GAT (with and without skip), GCNs, GraphSage, and CNN with varied hyperparameters. The hyperparameters giving best test rewards have been reported . We use three metrics to evaluate our training and results :

- **Loss** : This is the MSE loss that we incur during training.
- **Number of Episodes Lasted** : This is the number of steps pacman takes before getting hit by a ghost.
- **Reward :** Reward gained per episode or per game.

We show our training plots for GAT in Figures 3,4 and 5 . Similar plots for CNN can also be seen (6 and 7).
Testing : For testing, we run 20 games (~ 1200-1500 episodes) . (Each game goes on until pacman hits a ghost). We report the average rewards (per game) and average number of episodes survived in 1 . We can notice that CNN marginally outperforms GAT. GrapSage and GATs have very similar performance. We also try GATs with skip connections, which gave much more stable training (in the Appendix) and slightly better results than GATs in some fields.

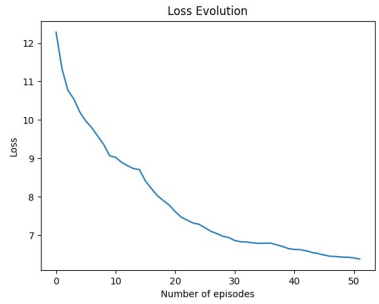

Figure 3. Loss of GAT in **Experiment 1**

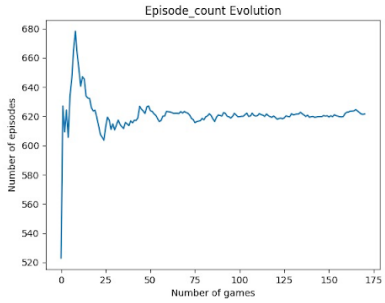

Figure 4. Number of episodes lasted (GAT) in **Experiment 1**

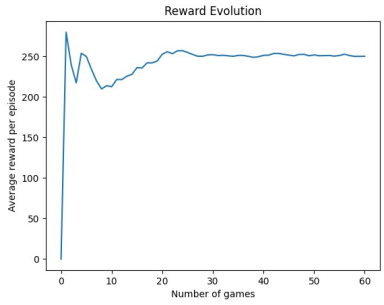

Figure 5. Reward (GAT) in **Experiment 1**

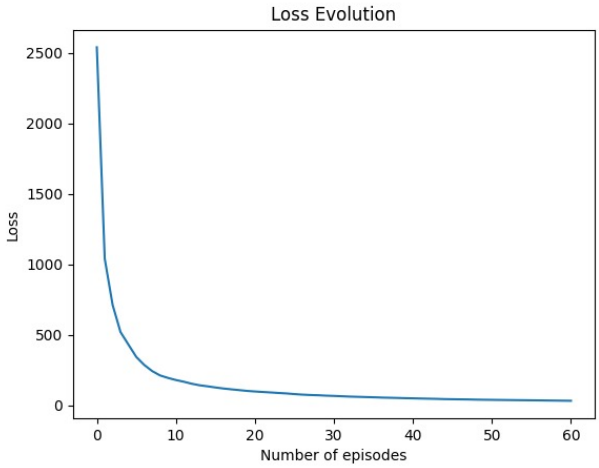

Figure 6. Loss of CNN in **Experiment 1**

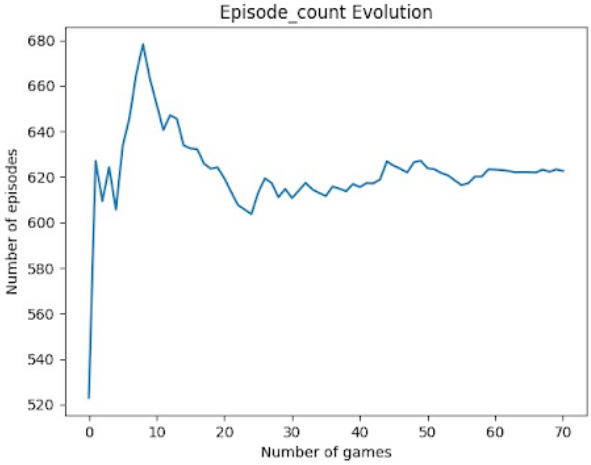

Figure 7. Number of episodes lasted (CNN) in **Experiment 1**

5.3 Experiment 2

In this experiment , we try the setting given in [6], in which they only connect the states if there is an action which allows transition from one to another. However, what we see is that this much information is just not enough for the pacman to make it's decision, especially when ghosts are nearby, in that case, the pacman should see the ghost and try to avoid it. While one may think multi-hop networks would perform better, our 2-hop model suggests otherwise. We plot our training graphs for the 1-hop case (As it gave better results in testing) . But still, we see that the rewards, keep on decreasing (though marginally, since it is a running average), which means that the model is not learning anything. (The loss was still decreasing, but performance did not improve). This suggests that this graph is not enough for the model to learn (at-least using 1 or 2 hops) . Similarly, for second part of experiment 2, we input a sub-image of the full 260x130x3 image. This did not get as good results as our Experiment 1 setting, but it was better than the previous case. The plots for this experiment, can be seen in Fig 8 and 9 for GATs and Fig 10 and 11 for CNN.

5.4 Experiment 3

Now, we try to visualize the attention weights. After training the model, we input the image to it and look at the matrix $\alpha = \alpha_{i,j}$ as described in Equation (3) . The attention layer outputs $z = [z_0, z_1 \dots z_{n^2}]$ for each node, where $n = 50, 100$. We combine them into α, using Equation (3) . Now, we focus on the attention weights given to one particular node (here,

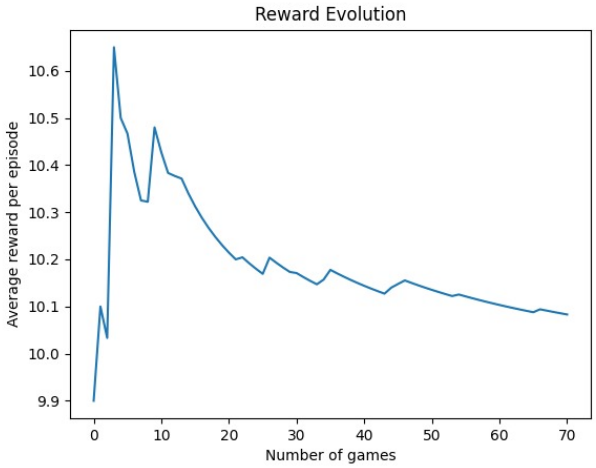

Figure 8. Reward of GAT, with partial observability (**Experiment 2**)

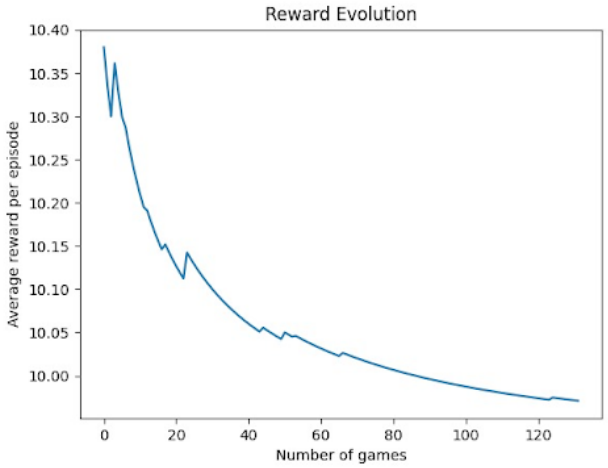

Figure 9. Reward of GAT, with adjacent pixels as edges (**Experiment 2**)

the nodes close to pacman's position) . This will be an n^2 dimensional vector , which we will roll into an $n \times n$ black and white array. The results for 50 x 50 can be seen in Fig 13, 14 an 15 . We can clearly see that the model learns the locations of the walls and some of the food . However it is not able to learn the ghosts, which might be because they are not stationary. Also, some of the foods is not seen in the attention weights. We however notice that the model is able to recognise the empty places. This suggests that it can survive for more time, however the reward will not be high as it does not capture the food position. This is in line with what we observed in the results. The Fig 16 and 17 show the attention weights during and after training.

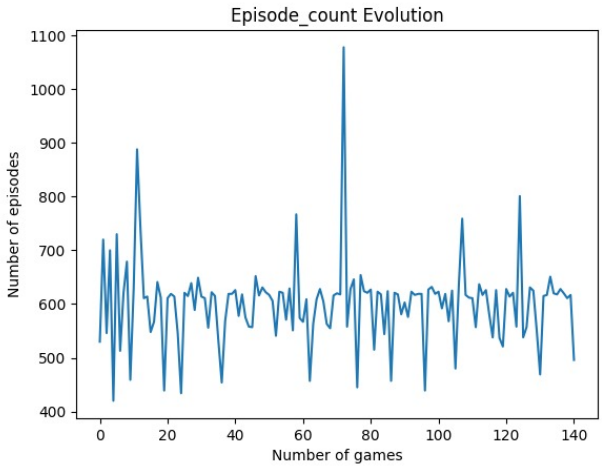

Figure 10. Number of episodes lasted in **Experiment 2**

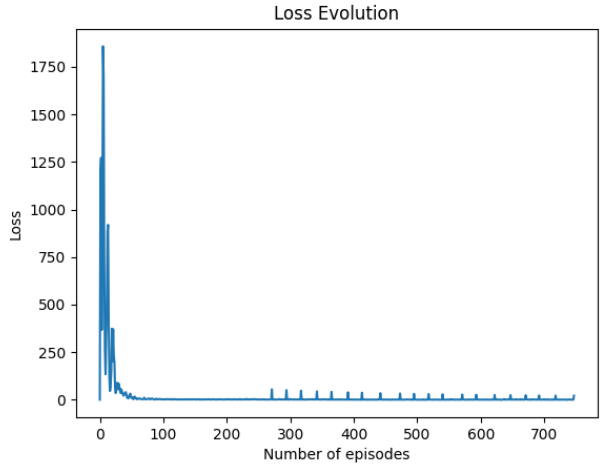

Figure 11. Loss in **Experiment 2** (It is unstable)

Figure 12. An example screen-layout

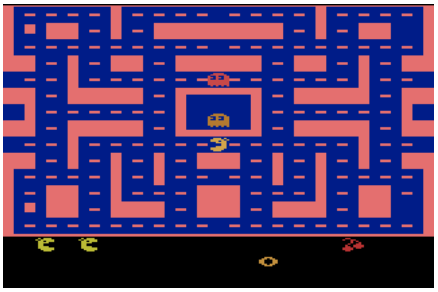

Figure 13. Input Image given

Figure 14. Attention Weights with 50x50 image as graph. Black spot is the node w.r.t to which we find the attention. (because self attention is 0)

Figure 15. Attention Weights with 50x50 image as graph

6 Remarks and Future work

The results, though promising do not ideally meet one's expectations. However, the results are promising enough to not make one give up on the idea of GAT based modification to DQNs, or more generally stating, the idea of learnable parameters which capture varying spatial importance.

Thus, it only makes sense to talk about possible modifications that can be brought about to the above proposal. One possible modification is to replace the attention layer with attention-like layers. The signal/ embeddings can be fed from only one pixel (preferably the pacman, which will help the model identify it as a more 'special' node) rather than concatenating the embeddings from all the nodes. One must note that we do need to find the embeddings for other nodes in such a setting as we would not have a final objective that helps us backpropogate error to learn the same. This framework justifies the weights being called 'attention' in a more convincing manner.

Figure 16. Attention Weights with 100x100 image as graph (during training)

Figure 17. Attention Weights with 100x100 image as graph (after training)

Moreover, one must note that a single attention layer does not embed a lot of positional information into the embeddings. The only sense of positionality that would come in embeddings obtained this way would be from the order of concatenation, since it is a fully connected graph. What we feel may be our next best chance at getting a more explanatory set of attendance weights would be a multi-hop GAT, first layer of which assumes the graph to be fully connected and the second layer takes into account only the local neighborhood. This would surely enhance the significance of positional information in the embeddings. Lastly, the results of many such experiments (including ours) are not anywhere near where we would want them to be. But, one must remember even a bad result is not no result!

Acknowledgments

To Robert, for the bagels and explaining CMYK and color spaces.

References

[1] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. arXiv:1606.01540 [cs.LG]

[2] William L. Hamilton, Rex Ying, and Jure Leskovec. 2018. Inductive Representation Learning on Large Graphs. arXiv:1706.02216 [cs.SI]

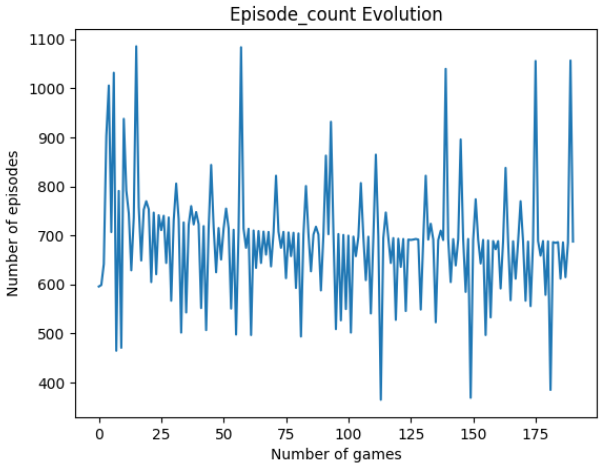

Figure 18. Skip GAT loss in **Experiment 1**

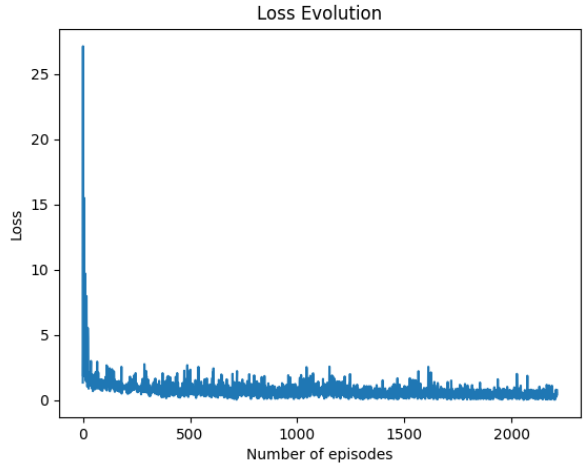

Figure 19. Skip GAT number of episodes lasted, in **Experiment 1**

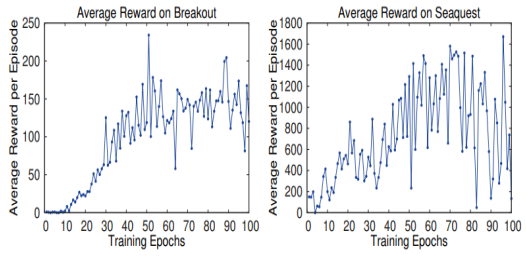

Figure 20. Training Curve taken from [4]. Reward vs Epoch is shown. We can see that unstable rewards are almost unavoidable in RL.

[3] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. arXiv:1609.02907 [cs.LG]
[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. arXiv:1312.5602 [cs.LG]
[5] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. arXiv:1710.10903 [stat.ML]
[6] Daniel Kudenko Vikram Waradpande and Megha Khosla. 2010. Graph-based State Representation for Deep Reinforcement Learning. *arxiv* 41, 4 (April 2010), 185–194.
[7] Huimu Wang, Zhiqiang Pu, Zhen Liu, Jianqiang Yi, and Tenghai Qiu. 2020. A Soft Graph Attention Reinforcement Learning for Multi-Agent Cooperation. In *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*. 1257–1262. https://doi.org/10.1109/CASE48305.2020.9216877
[8] Vikram Waradpande, Daniel Kudenko, and Megha Khosla. 2021. Graph-based State Representation for Deep Reinforcement Learning. arXiv:2004.13965 [cs.LG]
[9] Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S. Du, Ken ichi Kawarabayashi, and Stefanie Jegelka. 2020. What Can Neural Networks Reason About? arXiv:1905.13211 [cs.LG]

A Appendix

A.1 Skip GATs

[5] suggests that GATs with skip connections (across GAT layers) work better. We try that and notice that the training is much more stable and the results are slightly higher. However, the reward and episode plots, do not improve much with time, even though they have higher values than those recorded for GATs. Refer to Fig 18 and 19 .

A.2 Training DQNs

Training RL algorithms is much harder than normal supervised algorithms. This is because RL has no concept of ground truth. The algorithm works on bootstrapping, i.e. using it's own estimates to improve itself. The algorithm depends on the ability to find good rewards while exploring . Once we start getting good rewards, we focus more on exploiting the values that we have. Due to this, it is very hard to get a clean training curve for RL algorithms, they tend to have very high variance. For our case, we keep $\epsilon = 0.995$ as our initial exploration vs exploitation constant (i.e. we explore 995% of the time) and then keep on decreasing it until $\epsilon = 0.01$. To compare the quality of training , we can look at the reward plots shown in [4] (Fig 20), which was one of SOTA models for Atari.