Memory-Based Multi-Tasking Agent

Ganga Meghanath(EE15B025), Sahana Ramnath(EE15B109)

I. Abstract

Reinforcement learning agents have been found to be successful in various tasks and have even been transferred to and used in real-world domains. However, there is still a long way to go before they come close to humans when it comes to multi-tasking and transfer learning over similar and vastly different tasks. The idea for our project emerged from the belief that memory storage in cells is one advantage humans possess over machines that results in a significant difference in performance level, especially in data efficiency and long term dependencies between rewards and actions in a Reinforcement learning setup. In this project, we have done an extensive survey on existing methods for multi-tasking and incorporating memory in RL agents, and have developed and deployed a model that has memory and can learn to do multiple tasks. We have demonstrated the performance of our model on single as well as multiple(multi-tasking setup) Atari games, but in theory, the model can be extended to any reinforcement learning problem.

II. Introduction

Humans have the ability to store patterns in their brain, retrieve them when required as well as the power to make predictions about the future based on what they had perceived in their past as well as the present and make better decisions. A lot of research work has been done in the area of improving the performance of reinforcement learning agents by trying to model the environment, execute rollouts towards the future and several other techniques have been employed in the past.

Humans possess the ability to learn multiple tasks and remember them over long periods of time and also utilize this past knowledge when they come across a new task which enables them to learn faster. Multi-task learning is a famous problem in the domain of reinforcement learning on which people have been working for years. It is a difficult task to work with because the network should ideally learn to perform each of the tasks to the level achieved while learning on those tasks using individual learners, and at the same time, ensure that learning on one task doesn't result in significant amount of catastrophic forgetting on the other task. Catastrophic forgetting is a common phenomenon which has to be dealt with while learning to multi-task using a single network. Another phenomenon to be concerned about is whether positive and negative transfers are occurring while learning to perform multiple tasks. A positive transfer is desirable especially if the tasks are similar and since the network capacity is limited. A positive transfer enables the network to share it's parameters or share the information learned between tasks. Whereas a negative transfer is undesirable because if the network is unable to differentiate between tasks and the skills learned are ones that are beneficial to some of the tasks, then the skills that proved to improve the performance on one task might be detrimental while performing another task. Our end goal is to achieve optimal performance on all the tasks.

Here, we propose a model which is based on one of the many hypotheses of how the human brain works : a query-retrieval system. Our model is inspired from [14] where they maintain a short term as well as a long-term memory in a DQN architecture to improve performance in partially observable environments. Our proposed Multi-tasking model is built over a basic A3C-LSTM network. Since our ultimate aim is to learn multiple tasks using the model, we opted for an on-policy learning algorithm such as A3C over an off-policy update algorithm such as DQN which requires a replay memory, which in our case we would have to maintain replay memories corresponding to each task. Our architecture has multiple long-term memory modules and a single short term memory module, but in our experiments, we'll be making use of only three long term memory modules. The purpose of the long term memories is to acquire and store segments of information about various tasks and situations encountered over long periods of time. Now, when a task/situation is faced, information from all these modules queried the short term memory module to selectively aggregate the task relevant information and then a decision/action is taken by updating the state-action values using information from both the short term as well as long term memory modules.

We believe this architecture could aid in transfer between the tasks and also reduce catastrophic forgetting and negative transfer while learning to do multiple tasks. In an ideal scenario, each long-term module would presumably learn to represent distinct but shared features of the tasks and aid in performing well on each of the task.

III. Preliminaries

In this section, we briefly describe the algorithms and architectures we've mainly used or have taken inspiration from.

A. Asynchronous Advantage Actor-Critic (A3C)

A3C [12] is an algorithm for asynchronous gradient descent optimization in deep reinforcement learning frameworks where parallel actor-critic threads are initialized and executed, asynchronously updating the main network to give a stable and on-policy learning technique without the requirement of a replay memory as used for off-policy updates as observed in DQN [13] based frameworks. An estimate of the policy $\pi(a_t|s_t;\theta)$ and the value function $V(s_t;\theta_v)$ which are updated when

an episode terminates or if t_{max} number of environmental steps are reached, upon which the thread is reinitialized. The algorithm executes multiple threads of the actor and critic networks asynchronously and gathers parameter updates in parallel.

B. Adaptive Active Sampling Method : A5C

The algorithm was introduced in [20] where the sequence of incoming tasks for training the multi-tasking network was controlled by using Adaptive Active Sampling technique. At the end of every episode, the next task to train the network on is sampled from the probability distribution

$$p_i = \frac{e^{\frac{m_i}{\tau}}}{\sum_{c=1}^{k} e^{\frac{m_c}{\tau}}}$$

where k denotes the number of tasks and p_i denotes the probability of sampling task i, $m_i = \frac{ta_i - a_i}{ta_i}$ is called the *evidence* which is an estimate of how well the network can solve task i, ta_i is the target score for task i and a_i is the average score over the past n scores for task i.

C. MQN, RMQN and FRMQN

The paper *Control of Memory, Active Perception, and Action in Minecraft* [14] introduces and describes three memory-incorporated DQN architectures. There are two levels of memory : a short-term memory which is a vectorial representation of the recent history of observations and a long term memory which is a vectorial representation of the entire history of the agent's learning. A context vector representing the task is obtained from retrieving information from the two levels of memory and is used for state-action value estimation; the current time step's context vector becomes the long term memory representation for the next time step. Depending on how the context vector is constructed the three architectures are obtained : Memory Q-Network(MQN), Recurrent Memory Q-Network(RMQN) and Feedback Recurrent Memory Q-Network(FRMQN).

- **MQN :** The context vector is constructed as a projected representation of the short term memory.
- **RMQN :** Here the long term memory is used to query and retrieve relevant material from the short term memory. The short and long term memory are together used to update the long term memory which is used as the context vector.
- **FRMQN :** Here again, the long term memory is used to to query and retrieve relevant material from the short term memory. But there is a feedback loop wherein the retrieved short term memory is used along with the current state and long term memory to update the context vector.

IV. Memory Incorporated Architectures for Single-Task Agents

The following three architectures which incorporate memory into an A3C agent were adapted from [14]. They consist of a short term memory and a long term memory : vectorial representations of the recent history and entire history of observations. The two memories query and refresh themselves before they are used to calculate the state-action values which the agent uses to move in the environment.

A. Memory-A3C Network

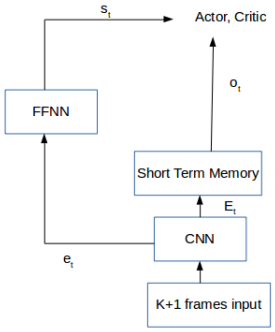


Fig. 1: Diagrammatic description of the architecture of Memory-A3C (M-A3C) network comprising of a short term memory and a feed forward neural network which are used to compute the q values

This model (Figure 1) has been adapted from MQN architecture [14], which comprises of an A3C-Feed Forward network augmented with only a short term memory. This projected short term memory vector along with the feed forward representation of the current state(the context vector in this case) are used to calculate the actor and the critic. The equations can be found in the appendix.

B. Recurrent-Memory-A3C Network

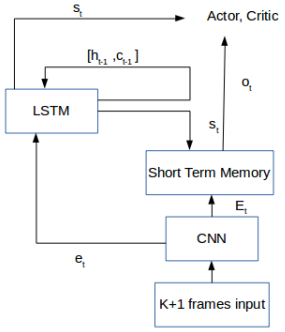


Fig. 2: Diagrammatic description of the architecture of Recurrrent Memory-A3C (RM-A3C) comprising of a short term memory and an LSTM as the long term memory/context vector which are used to update the context vector and to compute the state-action values

This model(Figure 2) has been adapted from the RMQN architecture [14], which comprises of a short term as well as a long term memory module incorporated into an A3C-LSTM network. Here, there is a connection from the long term to the short term memory and so, the context vector is used to view and understand the current short term observations in a better light. The projected short term memory vector and the context vector(calculated from the short and long term memories) are used to calculate the actor and the critic. The equations can be found in the appendix.

C. Feedback-Recurrent-Memory-A3C Network

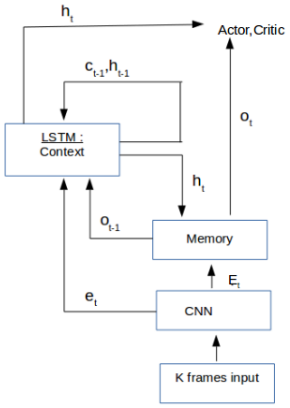


Fig. 3: Diagrammatic description of the architecture of Feedback Recurrrent Memory-A3C (FRM-A3C) comprising of a short term memory and an LSTM as the long term memory/context vector which are used to update the context vector and to compute the state-action values and a feedback connection from the short-term memory to long term memory while updating the long-term memory

This model(Figure 3) has been adapted from the FRMQN architecture [14], which comprises of a short term as well and a long term memory module incorporated into an A3C-LSTM network as well as a feedback from the long term to the short term memory. The projected short term memory vector along with the context vector(calculated from the long term memory, short term memory as well as the projected short term memory vector) are used to calculate the actor and the critic. The equations can be found in the appendix.

V. Architecture of MRM-A3C

Modularised Recurrrent Memory-A3C (Refer Figure 4) is our proposed model which multi-tasks by using memory. The architecture comprises of two layers of attention :

- Lower-level : n attention mechanisms, one corresponding to each long-term memory. This attention layer is used to extract information from the long-term memory modules using the common short-term memory. The short term memory comprising of recently observed states is slid over each long-term memory to extract relevant information corresponding to the sequence of states.
- Higher-level : The information from the long-term memory modules (extracted by the lower-level attention) are selectively aggregated to decide what action to take, based on the current state(inspired from [16]).

The relevant equations for the construction of the architecture (Refer Figure 4) with (current observation) Input frame : x_t
Frame Encoding :

$$e_t = CNN(x_t)$$
$$E_t : [e_{t-1}, e_{t-2}, ..., e_{t-K}]$$

Long-term Memory Module i :

$$[s_{ti}, h_{ti}, c_{ti}] = LSTM(e_t, h_{(t-1)i}, c_{(t-1)i})$$

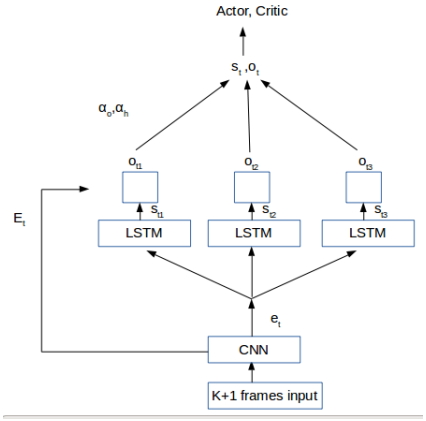


Fig. 4: Diagrammatic description of the architecture of Modularized Recurrent Memory-A3C (MRM-A3C) comprising of a single short term memory and three long term memory modules with two layers of attention mechanism to enable selective aggregation of information

Short-term Memory Storage :

$$M_t^{key} = W^{key} E_t$$
$$M_t^{value} = W^{value} E_t$$

Short-term Memory Output for i^{th} Long-term Memory Module at time step t using Lower-level Attention mechanism:

$$p_{ti} = softmax(s_{ti}^T M_t^{key})$$
$$o_{ti} = M_t^{val} p_{ti}$$

High Attention weights for aggregation of outputs :

$$r_o = V_{attn_o}^T \tanh(U_{attn_o} e_t + W_{attn_o} O_o)$$
$$\alpha_o = softmax(r_o)$$
$$r_s = V_{attn_h}^T \tanh(U_{attn_h} e_t + W_{attn_h} S_o)$$
$$\alpha_s = softmax(r_s)$$

Aggregation of outputs of Sort-term Memory and Long-term Memory at time step t:

$$o_t = O_o^T \alpha_o$$
$$s_t = S_o^T \alpha_s$$

VI. Overview of the Algorithm MRM-A3C

For the current task, the features from the last k input observations are extracted using a Convolutional Neural Network and stored in the short term memory as a key-value representation. The lower layer attention uses the keys and the corresponding long term memory to selectively aggregate values from the short-term memory corresponding to the information stored in the respective long term memory for the recent observations. This information along with the context vector corresponding to each long term memory is used to selectively aggregate the vectors to give an accumulated short-term memory vector and a context vector which is used to compute the state-action values and take actions in the environment. Upon switching to a new task, the short term memory is cleared and re-initialized with the observations from the new-task and the process is repeated.

VII. Project Report

In this project, we aimed to create and implement a working model of a RL agent which can learn to perform multiple tasks with the help of augmented short term and long term memory. Our proposed framework will potentially work for any RL task at hand, but for the purpose of demonstrating our idea, we trained our agent on games from the ALE domain.

The agent is evaluated based on its performance in the games learnt as compared to the baselines upon which it is built : raw rewards per game(or q_{am} scores as in [20]) and number of games learnt by the agent while multi-tasking.

For all models, we implemented our ideas over an A3C[12] and not a DQN [13] as in [14] because the final goal was to use the model for multi-tasking over a large number of games and we wished to avoid maintaining separate replay buffers for each game. We initially proposed three models :

- Model 1 : Agent with memory learning a single task
- Model 2 : Multi-tasking agent with dedicated memories per task
- Model 3 : Multi-tasking agent with memories shared across tasks

We implemented Model 1 for the first and part of the second phase, after which we directly implemented Model 3. All the models were implemented in Tensorflow[1].

A. First Phase

In the first phase of the project we implemented the FRM-A3C architecture (described in Section V) in Tensorflow. We took a little time to understand the original codes of [14] which was implemented in Lua Torch[2]. We started training on three different Atari games : Pong, Frostbite and Breakout. Our main aim at this time before moving on to multi-tasking was to test our hypothesis that addition of memory will result in a significant improvement in the performance level. But by the end of the first phase, the FRM-A3C agent barely showed any learning and was showing an extremely worse performance when compared to a normal A3C agent. With that, for the second phase we moved on to simpler architectures with memory : M-A3C and RM-A3C.

B. Second Phase

In the second phase of the project we first implemented 2 simpler memory-incorporated architectures : M-A3C and RM-A3C(described in Section V). We tested these on individual Atari games : Pong and Frostbite. These models gave promising results and performed much better than standard memory-less A3C agents; they gave better regret optimality and earlier convergence.

On this note, we implemented our multi-tasking model MRM-A3C and started experiments on it.

[1]building on top of https://github.com/miyosuda/async_deep_reinforce

[2]https://github.com/junhyukoh/icml2016-minecraft

C. Third Phase

In the third phase of the project, we evaluated MRM-A3C, RM-A3C and A3C-LSTM on two sets of Atari games from [20] :

- **MT1 :** Space Invaders, Seaquest, Crazy Climber, Demon Attack, Name This Game and Star Gunner
- **MT2 :** Asterix, Assault and Alien

All three models were extended to multitasking setup and Adaptive Active Sampling technique [20] was used for deciding the next task during the training phase.

Our initial experiments resulted in catastrophic forgetting by the agent in all the games. We then tuned the hyperparameters and started another set of experiments; these are running now, and we observed that the model with memory outperforms/performs on par with the baseline memory-less A3C model. This time, there was no catastrophic forgetting observed. At the time of submission of this report, the models have learnt for almost $65M$ steps across all tasks(games); the A5C agent used in [20] takes around $100M - 300M$ (varies across different games) to finally converge, so we're hoping that by that time or even lesser, our model will converge to a better performance.

VIII. Related Work

Researchers have been working on Multi-task learning and the use of memory for enhancing performance of RL agents and neural network models for many years. The paper [14] introduces an architecture (adapted from [23])that performs well in partially observable environments with delayed rewards using high dimensional input observations. The final DQN-based architecture has a separate short-term and long-term memory inter-connected by feedback loops. The architecture generalizes well over temporally extended frames and is able to retrieve memory blocks through time while taking appropriate actions for the current time step. [15] introduces an architecture similar to [14], but uses an adaptive and sparse write operation to memory (reducing frequent overwriting) selective to the agent's current location, storing information over long time periods in 2D and 3D environments. In [8], a short term memory for recent sensory inputs coupled with a long term memory is incorporated into a recurrent framework to enable continuous control in partially observable environments, by an algorithm derived from DPG[22] and SVG[7]. For better autonomous exploration and obstacle avoidance, memory has been incorporated into the agent : *Memory-based Multilayer Q-Network* [3]; learning from scratch using a model-free off-policy method and maintains a linear sort-term as well as a long-term memory. The paper [5] suggests that memory is an integral part of a learning system, enabling data-efficient learning in sparse conditions and long term dependencies between actions and rewards; and stresses on the importance of an episodic memory(relating to events) in an RL agent.

A model called Pathnet [4] uses a single large neural network capable of multi-tasking, continual learning and transfers by evolving pathways (evaluated using fitness function) corresponding to each task that could have shared paths from

other tasks. The learning of tasks is done in a sequential fashion whereas pathway for a single task can be learned sequentially or in parallel using an A3C [12]. Recently [21] introduces algorithms where a single A3C[12] agent is trained *online* on mutliple tasks in a sequential, occasionally periodic fashion by sampling the difficult tasks relatively more frequently using 3 different techniques. AMN [24] uses Policy distillation technique[18] to inherit task-specific policies from expert DQN[13] networks into a single DQN network to perform multiple tasks and enhance transfer (feature regression from experts). Distill and transfer learning [24] is a joint multi-task learning and transfer setup where individual task-specific workers learn in a constrained fashion by a shared policy-distilled to be the centroid of task specific policies. Differential Policy Gradient [2] derived from DDPG[10], is used for joint multi-task learning in the context of robotic systems having continuous action space with shared sets of actions across tasks using an A3C based framework [12]. The Joint Many-Task model [6] was proposed in the context of NLP; the depth of the model is increased with task complexity with short cut connections from higher level to lower level to enable task hierarchy, giving outputs at different layers of the model.

Inorder to reduce catastrophic forgetting in neural networks while learning to do multiple tasks, [9] introduces an algorithm where the weight updates relevant to previous tasks are selectively constrained using *elastic weight consolidation* to allow continual learning. Other relevant works include Attend, Adapt and Transfer[16], Continuous Memory States[25], Progressive Learning[1], Universal Value Function Approximators[19], Deep Relationship Networks[11], Sluice Networks[17], etc.

IX. Experimental Results

In this section, we show the observed experimental results on individual as well as sets of games and compare the performance of different architectures and suggest explanations for the observed results.

We used Arcade Learning Environment and Atari games for the purpose of our experiments to evaluate the performance of the models mainly because :

- We felt that individual games in Atari could benefit from short term memory(we confirmed this in our experiments)
- Multi-tasking :
  - Baselines are available for multi-tasking in Atari games.
  - Wide range of games available for experimentation
  - All games can be accessed and used similarly leading to uniformity in the implementation.
  - Similarity of Atari games : Though each game is sufficiently unique in terms of appearance and in terms of how it's played, the Atari suite of games do share some underlying physics and abstract representations. Hence, they provide easier extension to multi-tasking and transfer learning.

For the evaluation criteria on the performance of the models, we'll be using the same metrics as mentioned in [20] and given below :

$$p_{am} = \frac{1}{k}\sum_{i=1}^{k}\frac{a_i}{ta_i}$$

$$q_{am} = \frac{1}{k}\sum_{i=1}^{k}\min\left(\frac{a_i}{ta_i}, 1\right)$$

$$q_{gm} = \sqrt[k]{\prod_{i=1}^{k}\min\left(\frac{a_i}{ta_i}, 1\right)}$$

$$q_{hm} = \frac{k}{\sum_{i=1}^{k}\max\left(\frac{ta_i}{a_i}, 1\right)}$$

A. Single task Experiments

We did a comparison of the performance of memory-augmented versus memory-less algorithms on individual games Pong and Frostbite.

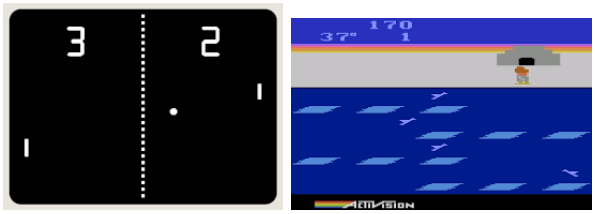


Fig. 5: Pong and Frostbite

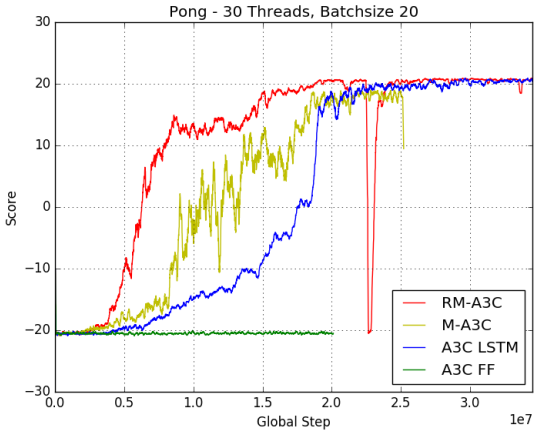


Fig. 6: Performance of 4 algorithms on Pong

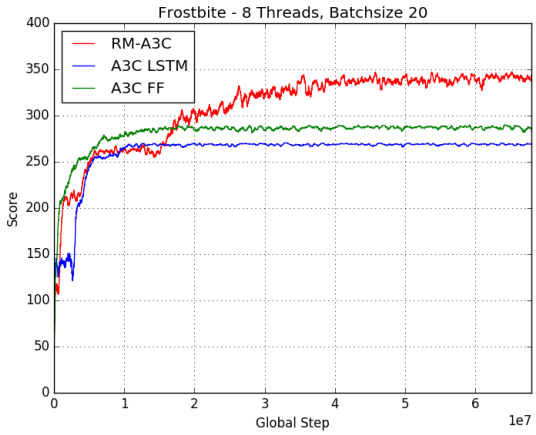


Fig. 7: Performance of 4 algorithms on Frostbite

TABLE I: Table for Comparison of Performance on Single Task

Metric	Game	A3C	M-A3C	RM-A3C
p_{am}	Pong	1	1	1
	Frostbite	0.675	-	0.875
q_{am}	Pong	1	1	1
	Frostbite	0.675	-	0.875
q_{gm}	Pong	1	1	1
	Frostbite	0.675	-	0.875
q_{hm}	Pong	1	1	1
	Frostbite	1	-	1

Experiments on Pong and Frostbite(Figure 5) showed that memory was indeed useful to improve performance(Figures 6 and 7).
Pong is a sports game that simulates table tennis. The player controls a paddle by moving it vertically across the left or right side of the screen. It can compete against another player controlling a second paddle on the opposite side. Players use the paddles to hit a ball back and forth. The goal is for each player to reach eleven points before the opponent; points are earned when one fails to return the ball to the other. In this game, knowing how the opponent has hit the ball and in what direction the ball is coming from will help to improve performance; this is where memory comes of use.
In Frostbite, the bottom two-thirds of the screen is covered by water with four rows of ice blocks moving left or right(alternate rows) constantly. The player moves by jumping from one row to another while trying to avoid various kinds of foes including crabs and birds. Each time it jumps on a piece of ice, a part of its igloo gets built; the goal of each level is to build the igloo.This game is helped by the presence of memory, since when the player has to jump from one row to the next, it has to know in which direction the next row's ice blocks are moving in and on which one(the one to the left or the right) it has to jump.

As can be see in Figure 6 addition of just a short-term memory itself gives better performance than A3C-LSTM and A3C-FF(which hasn't learnt anything). Further addition of long-term memory gives an even better performance.
In Frostbite as well(Figure 7), addition of short and long term memory helps to significantly increase performance.

B. Multi-tasking Experiments

We ran two multitasking experiments on MT1 and MT2 set of games using MRM-A3C and a slight variation of it, RM-A3C and normal A3C-LSTM. All three of these architectures were exposed to a game sequence picked using the A5C algorithm [20].

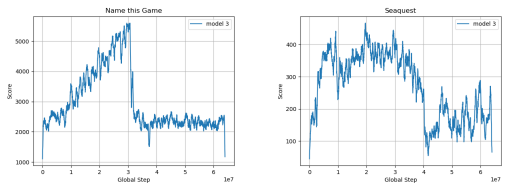


Fig. 8: Catastrophic Forgetting in Name This Game and Seaquest

Our initial set of hyperparameters led to catastrophic forgetting on all the games, most prominently noticed on Name This Game and Seaquest(refer Figure 8).
However, after tuning hyperparameters, the performance improved and MRM-A3C and RM-A3C started showing significantly better performance in some of the games(refer Figures 9 and 14)

1) Experiment 1: Our first experiment was conducted on an architecture similar to MRM-A3C but with one small variation : instead of obtaining higher-level attention weights based on the current state, we aggregated the outputs of the long term memory modules as well as the retrieved memories. Figure 9 shows our results on the same.

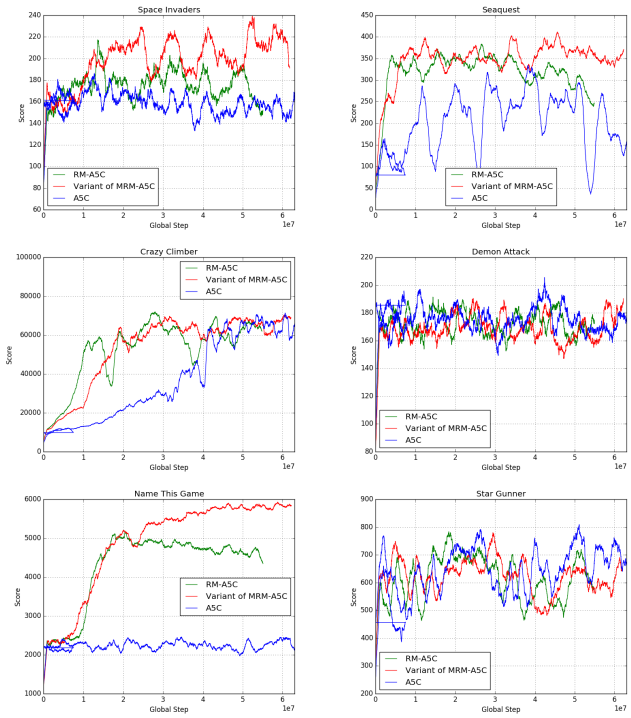


Fig. 9: Performance of the models on MT1 set of Games

Space Invaders is a shooter game where the player shoots aliens which move left/right/down. Memory helps the agent to figure out whether to shoot or move away according to the recent movement of the aliens. In Seaquest, the agent has to shoot the sharks as well as keep track of the oxygen decreasing(and how fast this happens); memory aids the agent to do this. In Crazy Climber, the agent has to climb four skyscrapers while avoiding sudden obstacles; while memory definitely helps this, A5C starts to perform well after some time because some of the obstacles arrive/disappear instantaneously and this can be captured by the 4 frames considered by the A5C. In Demon Attack, the agent has to shoot flying demons which can fly either to the left or to the right, but can also instantaneously change direction while flying; because of this, both A5C and memory-augmented A5C's both have to rely on past say 2-3 frames only and so they both perform similarly. In Name This Game, the agent has to protect a treasure from a giant octopus at the top, while being distracted by a shark; agent

can refresh oxygen by touching a long pole extended from a boat at the top from time to time. This game very clearly benefits from memory since the agent has to keep track of how the shark moves, which tentacle of the octopus is extending and in which direction and where the oxygen pole is; this is clearly seen in the performance difference in the plot(Figure 9 bottom left). In Star Gunner, similar to Demon Attack, the agent has to shoot flying enemies who seem to change their direction rapidly. So, presence or absence of memory leads to around the same performance.
All the above trends(memory being helpful or neutral) can be seen in Figure 9 very clearly.

TABLE II: Table for Comparison of Performance on MT1 set of Games

Metric	A3C	RM-A3C	MRM-A3C
p_{am}	0.367	0.479	**0.507**
q_{am}	0.367	0.479	**0.507**
q_{gm}	0.183	0.229	**0.243**

The above table shows our metrics used to compare the performance of the three architectures. Since the code takes almost triple the time it has currently ran for to converge/cross the baseline, we have used all baselines scores divided by three for the above calculation.

We have visualized the firing rate of the output neurons for each of the memory modules as well as for the aggregated representation from the attention module for the MT1 set of Games.

On comparing the firing rates of the neurons Figures 10, 11 and 12 (note that neurons have been rearranged in the decreasing order of firing rate), we see that patterns are not exactly the same, meaning that the long term memory modules are in fact learning information non-uniformly. Upon close observation, we can see that the fraction of neurons firing for each game is also different for the three context vectors.

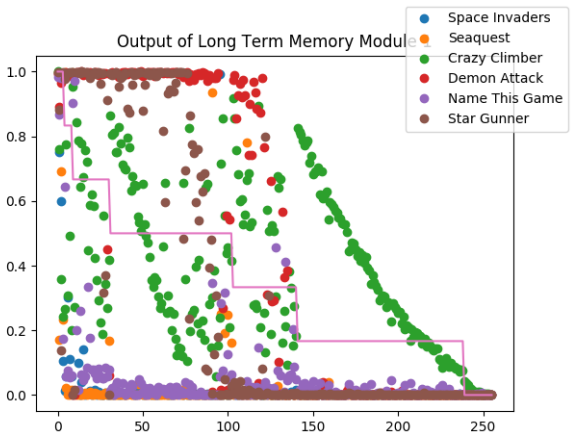


Fig. 10: Visualization of firing patterns of output neurons corresponding to the first long-term memory module

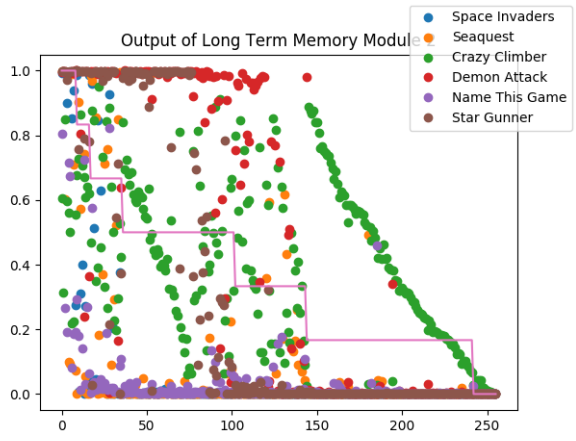


Fig. 11: Visualization of firing patterns of output neurons corresponding to the second long-term memory module

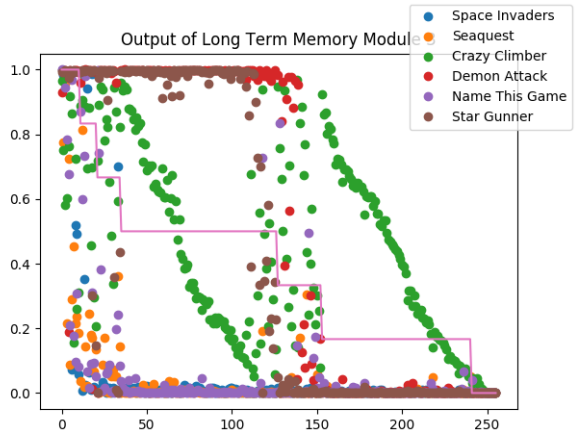


Fig. 12: Visualization of firing patterns of output neurons corresponding to the third long-term memory module

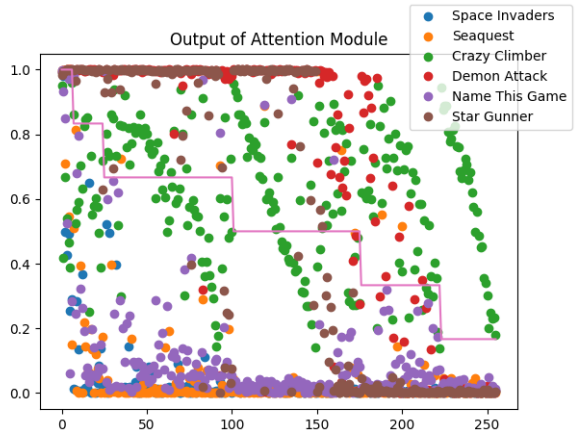


Fig. 13: Visualization of firing patterns of neurons at the output of the higher level attention mechanism which is an aggregation of the outputs from the long term memory modules

For the final context vector(13) we see that maximum information has been extracted out and almost all neurons

are subject to learning relevant information. We also see that Figure 13 represents that the final context vector is a direct aggregation of the three long term context vectors. Here, a larger fraction of neurons are firing with higher firing rates in comparison to the three individual base context vectors. This could be one of the reasons for better performance of MRM-A5C variant over RM-A3C and A3C.

2) Experiment 2: Our second experiment was conducted on the proposed MRM-A3C architecture : we apply higher-level attention weights on the long term memory modules as well as the retrieved memories based on the current state. Figure 14 shows our results on the same for MT1. Results for MT2 can be found in the appendix.

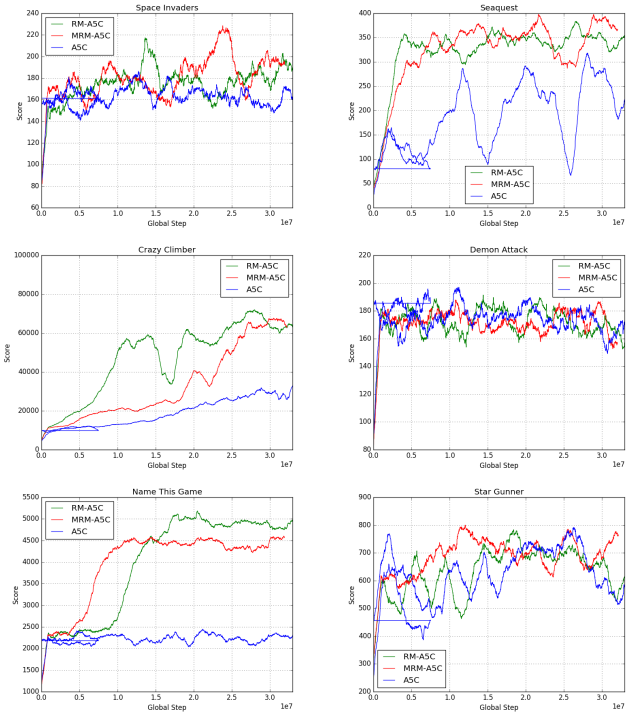


Fig. 14: Performance of the models on MT1 set of Games

Again here, the memory-based agents seem to be performing better than memory-less agents in most cases. The trend observed in the games seems to be the same as in Experiment 1(variant of MRM-A3C). The experiments have run only for 27M steps all games put together; this could account for the slightly lesser performance of the agent in Name This Game and Crazy Climber; however the agent has to train for atleast 100M steps before a clear conclusion can be made owing to the much deeper structure of the network as compared to RM-A3C.

Again, we have visualized the firing rate of the output neurons for each of the memory modules as well as for the aggregated representation from the attention module for the MT1 set of Games.

On comparing the firing rates of the neurons in Figures 15, 16 and 17 (note that neurons have been rearranged in the decreasing order of firing rate), we again see that patterns are not exactly the same, showing that the long term memory modules are learning information non-uniformly and that the fraction of neurons firing for each game is different for the three context vectors.

Fig. 15: Visualization of firing patterns of output neurons corresponding to the first long-term memory module

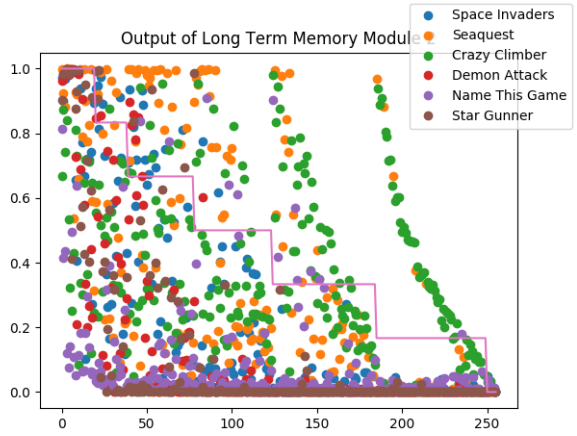


Fig. 16: Visualization of firing patterns of output neurons corresponding to the second long-term memory module

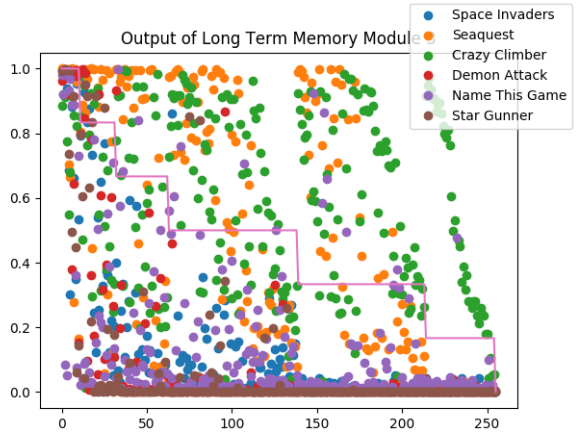


Fig. 17: Visualization of firing patterns of output neurons corresponding to the third long-term memory module

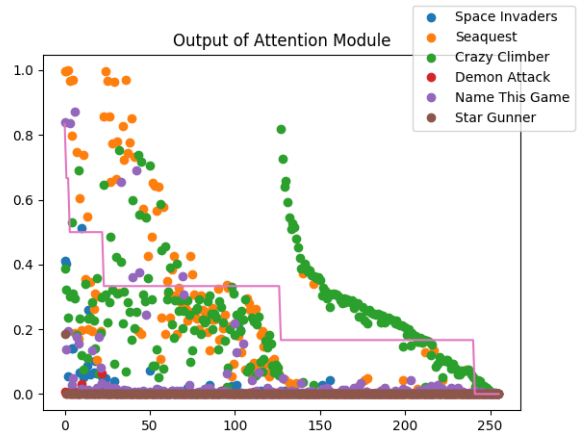
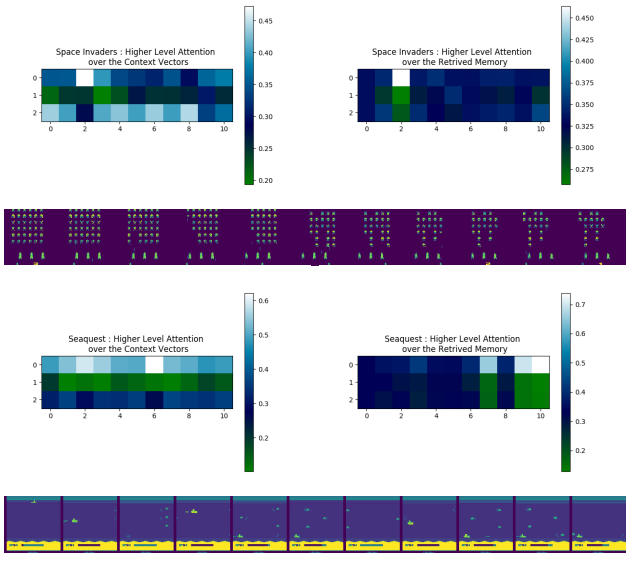


Fig. 18: Visualization of firing patterns of neurons at the output of the higher level attention mechanism which is a **selective** aggregation of the outputs from the long term memory modules depending on the current encoded input observation

For the final context vector(18) we see that maximum information has been extracted out and almost all neurons are subject to learning relevant information. We see on comparison that these figures are extremely different from Figures 10,11,12 and 13, indicating the difference in the architectures. Again here, a larger fraction of neurons are firing with higher firing rates in comparison to the three individual base context vectors. Figure 18 shows that a large fraction of the neurons learn all/most the games confirming our hypothesis that the different long term memory modules learn a shared common representation of the various tasks.

We have also visualized the attention weights learnt for MT1(refer Figure 19). Most of the games seem to have well separated attention weights as intended. However, some of them have almost overlapping attention weights $[0.33, 0.33, 0.33]$. This is probably a consequence of less learning; it also corresponds to poor performance as in Star Gunner(both context vector and retrieved memory weights are not well separated).



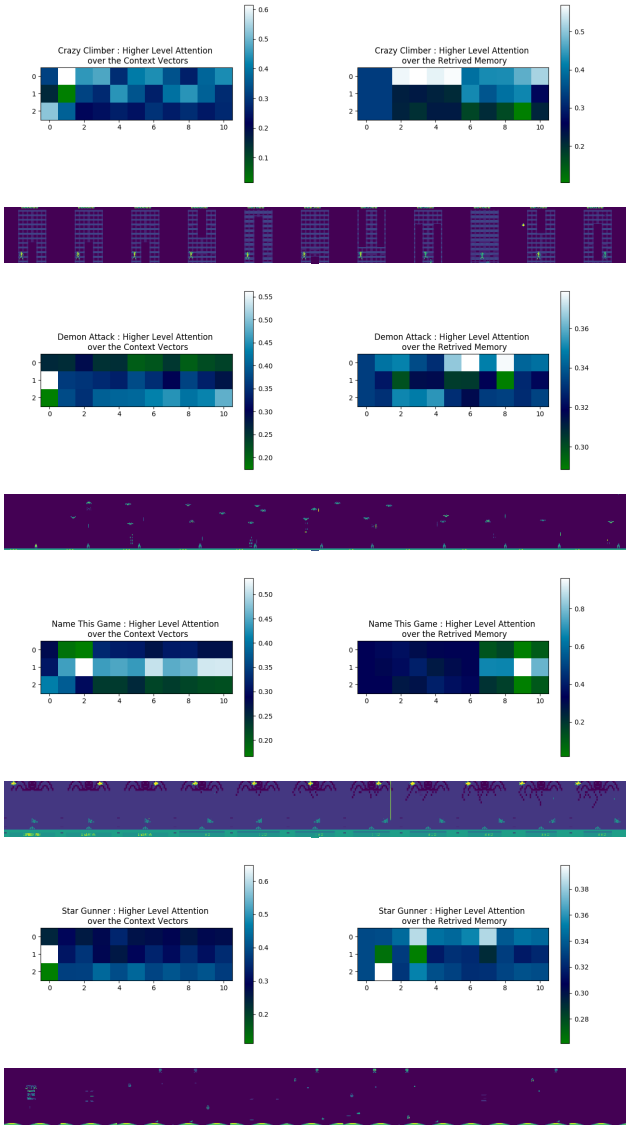


Fig. 19: Visualization of Higher Level Attention Weights for MT1 set of games : Space Invaders, Seaquest, Crazy Climber, Demon Attack, Name This Game and Star Gunner in order

It could also be because there aren't external conditions for constraining weights to be sparse so that the modules learn and store distinct information.

X. Conclusion and Future Work

We started this project trying to prove that memory does help an agent to perform better, whether it's performing a single task or multiple tasks. On that note, we implemented four different agents augmented with memory and compared their performances with memory-less agents. We have achieved promising results towards the same, as seen in the results for Pong and Frostbite(single tasks) and MT1 (multiple tasks).

However, there are a lot more experiments which could be conducted and intuitions gained. We could run experiments on more combinations of games, and observe the attention weights and also evaluate the agent on transfer tasks using the learned skills and check for negative transfers. Making the

domain more challenging can help introduce scenarios where short term memory becomes a requirement. For example one could tweak input frames as used in [16], where parts of the frames are hidden such that there is a need for dependency on nearby frames while taking the decision for the current observation, increasing the need for a short term memory. The model could be evaluated in a multi-tasking setup in such scenarios.

One could also try using pre-trained short term memory representations that capture the essence of all the different tasks and learn a better encoding for the observations. We could also try updating the W_{value} as $\sum_{i=1}^{K} S_i S_i^T$ for storing patterns from different games and then retrieve information and an encoding using the current observation and with the key calculated as before and W_{key} undergoing regular updates. A better initialization method for the LSTM weights could also help in the storage of more distinct information in the three context vectors. One way is to initialize the LSTM parameters to be far apart. Another way is to start off with pre-trained weights from individual games for the LSTM networks. We could also constrain the attention weights to be sparse, thus forcing the long term memory modules to learn distinct representations. One limitation of our current model is that it doen't have any explicit constraints for ensuring that the information stored in the 3 context vectors are distinct from each other.

ACKNOWLEDGEMENTS

First we would like to thank Professor Balaraman Ravindran for guiding us through the course of the project. We would also like to thank our TA Rahul Ramesh for his support and advise, and last but not least, our sincere gratitude to Ashutosh Jha [20] for allowing us to use his Adaptive Active Sampling code as well as the plotting function for the firing rates of the neurons.

CORRECTIONS

We would like to apologize for a previous mistake in delivering the results. The results presented in the final presentation were for Experiment 1, that is aggregation without higher level attention. We have included results with attention and detailed explanations in the report.

REFERENCES

[1] Glen Berseth et al. "Progressive reinforcement learning with distillation for multi-skilled motion control". In: *arXiv preprint arXiv:1802.04765* (2018).

[2] Parijat Dewangan et al. "DiGrad: Multi-Task Reinforcement Learning with Shared Actions". In: *arXiv preprint arXiv:1802.10463* (2018).

[3] Amir Ramezani Dooraki and Deok Jin Lee. "Memory-based reinforcement learning algorithm for autonomous exploration in unknown environment". In: *International Journal of Advanced Robotic Systems* 15.3 (2018), p. 1729881418775849.

[4] Chrisantha Fernando et al. "Pathnet: Evolution channels gradient descent in super neural networks". In: *arXiv preprint arXiv:1701.08734* (2017).

[5] Samuel J Gershman and Nathaniel D Daw. "Reinforcement learning and episodic memory in humans and animals: an integrative framework". In: *Annual review of psychology* 68 (2017), pp. 101–128.

[6] Kazuma Hashimoto et al. "A joint many-task model: Growing a neural network for multiple NLP tasks". In: *arXiv preprint arXiv:1611.01587* (2016).

[7] Nicolas Heess et al. "Learning continuous control policies by stochastic value gradients". In: *Advances in Neural Information Processing Systems*. 2015, pp. 2944–2952.

[8] Nicolas Heess et al. "Memory-based control with recurrent neural networks". In: *arXiv preprint arXiv:1512.04455* (2015).

[9] James Kirkpatrick et al. "Overcoming catastrophic forgetting in neural networks". In: *Proceedings of the national academy of sciences* (2017), p. 201611835.

[10] Timothy P Lillicrap et al. "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (2015).

[11] Mingsheng Long and Jianmin Wang. "Learning multiple tasks with deep relationship networks". In: *CoRR, abs/1506.02117* 3 (2015).

[12] Volodymyr Mnih et al. "Asynchronous methods for deep reinforcement learning". In: *International conference on machine learning*. 2016, pp. 1928–1937.

[13] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), p. 529.

[14] Junhyuk Oh et al. "Control of memory, active perception, and action in minecraft". In: *arXiv preprint arXiv:1605.09128* (2016).

[15] Emilio Parisotto and Ruslan Salakhutdinov. "Neural map: Structured memory for deep reinforcement learning". In: *arXiv preprint arXiv:1702.08360* (2017).

[16] Janarthanan Rajendran et al. "Attend, Adapt and Transfer: Attentive Deep Architecture for Adaptive Transfer from multiple sources in the same domain". In: *arXiv preprint arXiv:1510.02879* (2015).

[17] Sebastian Ruder et al. "Learning what to share between loosely related tasks". In: *arXiv preprint arXiv:1705.08142* (2017).

[18] Andrei A Rusu et al. "Policy distillation". In: *arXiv preprint arXiv:1511.06295* (2015).

[19] Tom Schaul et al. "Universal value function approximators". In: *International Conference on Machine Learning*. 2015, pp. 1312–1320.

[20] Sahil Sharma et al. "Learning to Multi-Task by Active Sampling". In: *arXiv preprint arXiv:1702.06053* (2017).

[21] Sahil Sharma et al. "Learning to Multi-Task by Active Sampling". In: *arXiv preprint arXiv:1702.06053* (2017).

[22] David Silver et al. "Deterministic policy gradient algorithms". In: *ICML*. 2014.

[23] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. "End-to-end memory networks". In: *Advances in neural information processing systems*. 2015, pp. 2440–2448.
[24] Yee Teh et al. "Distral: Robust multitask reinforcement learning". In: *Advances in Neural Information Processing Systems*. 2017, pp. 4496–4506.
[25] Marvin Zhang et al. "Learning deep neural network policies with continuous memory states". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 520–527.

APPENDIX

A. M-A3C Equations

The relevant equations involved in Memory-A3C network has been portrayed below. The storage of information and it's retrieval from the short term memory has been depicted.

Input frame : x_t

$$e_t = CNN(x_t)$$
$$E_t : [e_{t-1}, e_{t-2}, ..., e_{t-K}]$$
$$s_t = W^c e_t$$

Memory Storage :

$$M_t^{key} = W^{key} E_t$$
$$M_t^{value} = W^{value} E_t$$

Memory Output :

$$p_t = softmax(s_t^T M_t^{key})$$
$$o_t = M_t^{val} p_t$$

B. RM-A3C Equations

The relevant equations involved in Recurrent Memory-A3C network has been portrayed below. The storage of information and it's retrieval from the short term memory as well as the long term memory has been depicted.

Input frame : x_t

$$e_t = CNN(x_t)$$
$$E_t : [e_{t-1}, e_{t-2}, ..., e_{t-K}]$$

Long-term Memory :

$$[s_t, h_t, c_t] = LSTM(e_t, h_{t-1}, c_{t-1})$$

Short-term Memory Storage :

$$M_t^{key} = W^{key} E_t$$
$$M_t^{value} = W^{value} E_t$$

Short-term Memory Output :

$$p_t = softmax(s_t^T M_t^{key})$$
$$o_t = M_t^{val} p_t$$

C. FRM-A3C Equations

The relevant equations involved in Feedback Recurrent Memory-A3C network has been portrayed below. The storage of information and it's retrieval from the short term memory as well as the long term memory has been depicted.

Input frame : x_t

$$e_t = CNN(x_t)$$
$$E_t : [e_{t-1}, e_{t-2}, ..., e_{t-K}]$$

Long-term Memory :

$$[s_t, h_t, c_t] = LSTM([e_t, o_{t-1}], h_{t-1}, c_{t-1})$$

Short-term Memory Storage :

$$M_t^{key} = W^{key} E_t$$
$$M_t^{value} = W^{value} E_t$$

Short-term Memory Output :

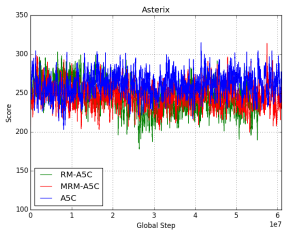
$$p_t = softmax(s_t^T M_t^{key})$$
$$o_t = M_t^{val} p_t$$

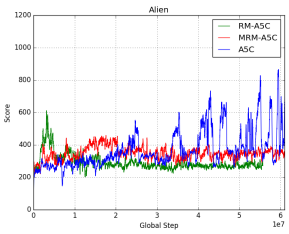
D. A5C : Adaptive Active Sampling

Algorithm 1 Multitasking using Adaptive Active Sampling

for i in range(k) : **do**
  $p_i = \frac{1}{k}$
for train_steps in range(t) : **do**
  if t **then**rain_steps $\geq l$:
    for i in range(k) : **do**
      $a_i \leftarrow s_i.average()$
      $m_i = \leftarrow \frac{ta_i - a_i}{ta_i \times \tau}$
      $p_i \leftarrow \frac{e^{m_i}}{\sum_{q=1}^{k} e_q^m}$
      $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\delta\boldsymbol{e}$
      $S \leftarrow S'$
    if S' is terminal
  $j \approx p$
  $score\_j \Leftarrow Train_one\_episode(T\_j)$
  $s.append(score\_j$
  if then$s\_j.length() > n$
    $s\_j.remove\_oldest()$

E. Results for MT2





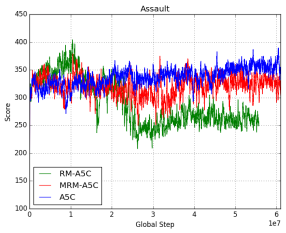


Fig. 20: Performance of the models on MT2 set of Games

For this set of games, we observed that memory-less agents performed better or on par with memory-augmented agents. Though each game could potentially individually benefit from having memory, learning together seems to have reduced the advantage of having memory. Now this issue could be because of slight limitations in the models used; running MT2 after incorporating some of the changes we thought of and have mentioned in Future Works could help in better learning.