Transfer in RL - Mapping RGB to RAM in Atari Games

Nidhi Kadkol **Yixuan Ni**

Abstract

In this project we attempt transfer learning between different input environments for Reinforcement Learning (RL) Agents in Atari Games. By learning a mapping from RGB screen image inputs to the Atari machine RAM input, we show that we take less than half the time to train an RL agent than on the original RGB screen input. We experiment with feedforward neural networks, Convolutional neural networks, as well as LSTMs to learn the mappings and get a best mean squared error of 0.37% of the RAM mappings using a Convolutional LSTM. We apply trained RAM input RL agents under the RGB environments by feeding the mapped inputs to the agents. The best agent accumulates 3.29 average rewards per episode using the CNN mapping, which is 27.5% of the rewards accumulated under the RAM environments.

1 Introduction

Every Atari game is one of two versions: either the input environment is a high dimensional visual input (210 x 160 x 3 RGB image screenshot) or a fixed size RAM state of the Atari Machine (128 bytes). Popular RL approaches focus on training RL agents with the first input kind (Mnih et al., 2013). However, this requires a significantly large amount of memory and training time. Furthermore, training two different RL agents for the same game simply because the input types differ is wasteful. We aim to create a framework wherein a single RL agent can be trained to play an Atari game irrespective of what the input type is. Training an agent on RAM input is much faster and requires less memory since the RAM size is so compact. We leverage deep neural networks to learn a mapping from an RGB input type to the corresponding RAM input type. This allows us to transfer the knowledge of how to play the game on RAM input to work with RGB input as well.

There has been work training RL agents on purely the RAM state of the Atari machine (Sygnowski and Michalewski, 2017) and purely RGB video inputs (Mnih et al., 2013) but no work on trying to resolve these separate methods into a single framework. The policies of the RL agent, that is, the actions it chooses to take based on the current game state should be the same irrespective of the input representation. Thus, there should be a way to use the same RL agent for either kind of input. This motivated us to leverage deep learning to find a mapping between different input representations. This will make big savings on resources, since now we do not need a completely different training setup and agent for each input type for a game.

To test the correctness of our mappings, we compared the mapped RAM state to the original RAM state of the machine at that instant. To learn mappings we experimented with feedforward neural networks, CNNs, as well as LSTMs with different sequence lengths. Using the mean squared error (MSE) and mean absolute error (MAE) metrics, our best model for mapping RGB to RAM gave a validation MSE of 0.37% and validation MAE of 1.68%, which was achieved by a CNN. We also got a best validation MSE of 0.37% and MAE of 2.25% with a Convolutional LSTM (Donahue et al., 2017) of sequence length 3.

Our low error rates for finding mappings show that there is a strong correlation between the information contained in the RGB images and that contained in the memory of the Atari machine. Thus, transferring knowledge between these inputs works well.

To train the RL agents on RAM inputs, we used feed forward neural networks as described in (Sygnowski and Michalewski, 2017). The RL agent with one hidden layer achieves an average of 11.10 rewards per episode of the Atari game, Breakout. The RL agent with three hidden layers achieves average 11.95 rewards. We then transfer these agents to RGB environments, and the best models achieve 3.29 and 1.90 average rewards for the large network and small network, respectively. Due to lack of resources we used CPUs and only trained for a few hours. Considering that the authors of that work had access to GPUs and trained for 4 days, this discrepancy is inevitable, and the results prove the feasibility of our idea that only one agent is needed for the same game with different types of inputs.

Section 2 summarizes related works in Reinforcement Learning and Transfer Learning; section 3 includes the tools and frameworks of our work, the motivation and a detailed description of our approach. In section 4, we detail our model architectures, experiments settings and results. In sections 5 and 6, we list our contributions and the limitations of our works, and identify possible future works for this task.

2 Related Work

Our work lies in the intersection of two major areas: Reinforcement Learning (RL) and Transfer Learning (TL). This section summarizes the development of deep-learning based RL approaches that leads to our baseline architecture:, DQN; and various attempts of applying TL in RL that related to our approach.

Reinforcement Learning One of the major goals of reinforcement learning is to control agents based on high-dimensional sensory inputs like vision and speech (Mnih et al., 2013). Many successful neural-based approaches have been proposed over the past decades. TD-gammon (Tesauro, 1995), for example, is a system playing backgammon that approximates the value function using a multi-layer perception and has achieved human-level performance. The achievements of deep learning approaches in computer vision and speech recognition have inspired Deep Neural Network based RL approaches. Restricted Boltzmann machines have been used to estimate the value function (Sallans and Hinton, 2004) or the policy (Heess et al., 2013); yet it has been difficult to develop nonlinear approximation for the Q-network since it can easily diverge (Tsitsiklis and Van Roy, 1997). The earliest successful nonlinear deep learning approach is the Deep Q-Network (DQN) (Mnih et al., 2013) (Mnih et al., 2015). DQN overcomes the instability of nonlinear approximation by using four techniques: experience replay, target network, clipping rewards and skipping frames. In this project, we use the DQN network as our baseline model, and we extend our approach on top of it.

Transfer learning in RL The work of Torrey and Shavlik (2009) describe the goal of transfer learning as developing methods to transfer knowledge learned in one or more source tasks to use it to improve learning in a related target task. For reinforcement learning, in particular, there are 5 metrics to measure the benefits of transfer (Taylor and Stone, 2009): 1. the initial performance of an agent in the target task; 2. the final learned performance of an agent in the target task; 3. the total reward accumulated by the agent; 4. the ratio of total reward accumulated by transfer learner and non-transfer learner; 5. learning time needed by the agent to achieve certain performance. For our project, we use metrics 3 and 5 to evaluate our approach, concerned with the performance of our model and the speed of the learning process.

Different transfer learning approaches have been proposed to tackle the problem in different ways: (Sutton et al., 2007) suggests solving a single large task as a sequential series of subtasks; some approaches transfer policies (Asadi and Huber, 2007), transition functions (Atkeson and Santamaria, 1997) or learned subroutines (Andre and Russell, 2002) across tasks. Most of the research focuses on transferring between two different tasks, which is inherited from the principle of transfer learning tasks in general. Our work is the first to identify the difference within the same task, and propose to unify the training with deep neural-based transfer learning approach.

3 Approach

In this section, we describe the tools and frameworks we used, the motivation and outline of our approach to transfer information from the RGB screen input to RAM input to train an RL agent, as well as the method of collecting data.

3.1 Atari Games in OpenAI Gym

OpenAI Gym (Brockman et al., 2016) is a toolkit for reinforcement learning research. In supervised learning, progress has been driven by large labeled datasets like ImageNet (Deng et al., 2009). In Reinforcement Learning, most open-source collections of environments do not have enough variety, are not standardized across publications, and are often difficult to set up and even use. OpenAI Gym is an attempt to fix these problems. It is a collection of environments that you can use to develop and compare RL algorithms. For our project we focus on Atari games and make use of the atari game environments provided by OpenAI Gym. For developing our models, we can choose any arbitrary game, we decided to focus on the Atari Game Breakout. We used keras-rl (Plappert, 2016) for developing our models and training our agent.

3.2 Transfer Learning Motivation

For each Atari game there are 2 environments, one in which the observation for the RL agent is the RGB image of the screen, which is an array of shape (210, 160, 3) (down-sampled to (84, 84, 3)); and one in which the observation is the RAM of the atari machine, consisting of 128 bytes. The size of the RGB image is much larger than that of the Atari Machine RAM (a single color channel ranges from 260 to over 300 bytes). Training RL agents on an RGB input environment thus takes much more time and memory space. Moreover a much deeper neural network for the RL agent

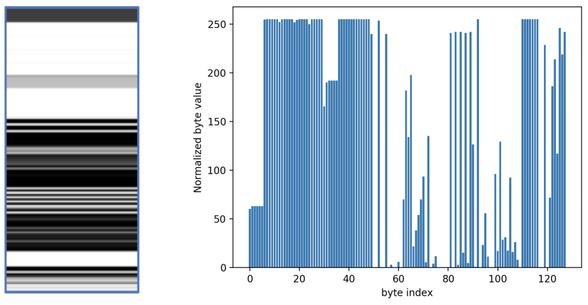

Figure 1: Visualization of RAM state averaged over 1078 samples. Left: Gray scale view, each row is a byte. Right: Graphical view, each column is a byte ranging from 0 to 255.

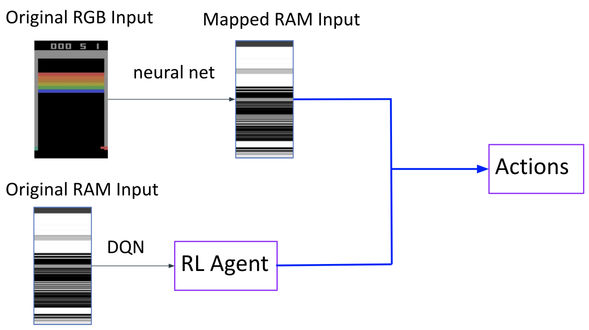

Figure 2: Process outline. When the the input observations are the RGB images we map them to the corresponding RAM state and use an RL agent pre-trained on RAM state inputs to predict the actions.

is required. The RAM environments can be encoded by much smaller networks since the input size shrinks so much. This motivates us to try to find a mapping between the RGB and the RAM input, and thus train an RL agent on the mapped RAM input.

3.3 Getting data

To collect training data for our experiments, we started a game of Breakout in the OpenAI Gym. The observations in each episode were the current environment state, both in RAM and RGB format. We saved every 20th pair of RGB and RAM observations in every 10th episode of the game. Saving every observation of every episode would lead to a much larger dataset with many consecutive observations having little or no difference. We wanted to capture as many different types of input observations as possible without having an unmanageable dataset. We collected 1078 pairs of original RGB images and RAM states which we used for all the experiments of the first half of our project, i.e. learning a mapping from RGB to RAM. The visualization of the mean of these ram states is shown in figure 1.

3.4 Process Outline

We want to used supervised learning to be able to map an RGB screenshot of a game at an instant to its RAM state at that instant. We train a neural network rgb2ram to learn this mapping. We also train an RL agent to play the game given the RAM state as input. For this we use the original RAM instances and train the RL agent using a Deep Q-Network (DQN) (Mnih et al., 2013, 2015).

Now given our trained RL agent and our trained neural network rgb2ram, we have a framework to play the atari game that can handle any kind of input environment. If the input observations are the RAM states of the Atari machine, we can directly use our RL agent trained on that kind of input to interact with these observations and predict actions for playing the game. If the input observations are the RGB screen images of the game, we use rgb2ram to map the images to the corresponding RAM state and have our RL agent interact with the mapped RAM states to predict actions to play the game. Figure 2 describes the process when the input observations are the RGB images.

4 Experiments

In this section we detail the model architectures used to learn a mapping from the RGB screen images to the RAM states of the Atari machine, and the models used to train the RL agent on the RAM input.

4.1 Learning Mappings

We have a set of 1078 pairs of original RGB images and original RAM states. Each value in the RGB and RAM samples are integers ranging from 0 to 255. We scale all the values to floating point numbers between 0 and 1. We divide the examples into an 80/20 train validation split. For all our models, we use a batch size of 8 and train for 60 epochs. During training we use the Adam optimizer (Kingma and Ba, 2014) and mean squared error as the loss function. To evaluate our models, we inspect the mean squared error as well as the mean absolute error.

4.1.1 Feedforward Neural Network

In our first attempt, we used a simple feedforward neural network. We took the original 84 x 84 RGB image, converted it to gray scale and flattened it into a vector of size 7056. Then we added 2 hidden layers of size 32 and 64 respectively, followed by the final layer of size 128 for the RAM output. For all layers we used the ReLU activation function except the last layer which had a linear activation.

Training this network for 60 epochs on 862 training samples gave us a final validation MSE of 0.39% and MAE 2.07%. The train and validation errors over training are shown in figures 3 and 4.

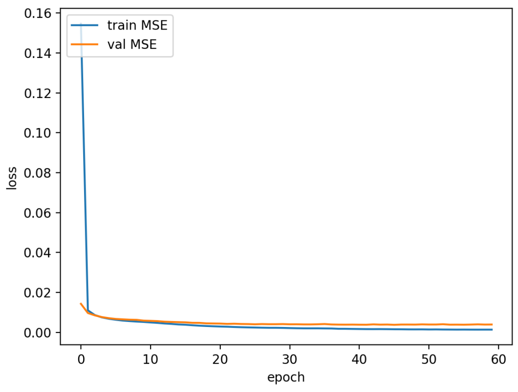

Figure 3: Mean squared error over 60 epochs for feed-forward neural network to learn mappings from RGB images to RAM states.

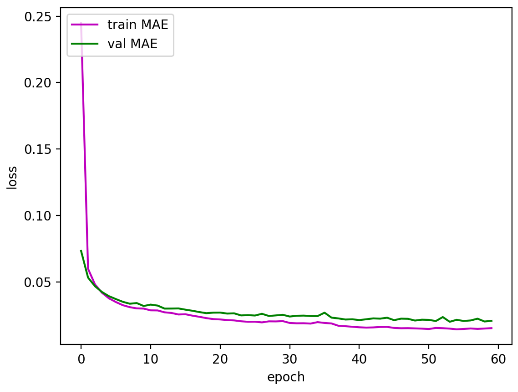

Figure 4: Mean absolute error over 60 epochs for feed-forward neural network to learn mappings from RGB images to RAM states.

4.1.2 CNN Model 1

Since the input is an image, flattening it out and passing that as input will make it lose information that is characteristic to image data types. Naturally, the next type of model that we used was a convolutional neural network. We started out with 2 convolution layers followed by a final fully connected layer of size 128. The architecture is shown in figure 5.

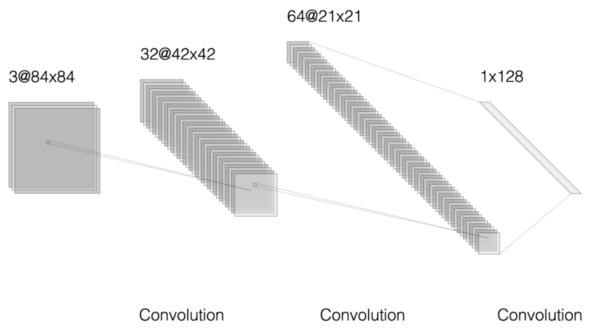

Figure 5: Architecture of CNN Model 1

Training for 60 epochs, we got a final validation MSE of 0.44% and MAE of 2.02%. The train and validation plots of MSE and MAE are shown in figures 6 and 7.

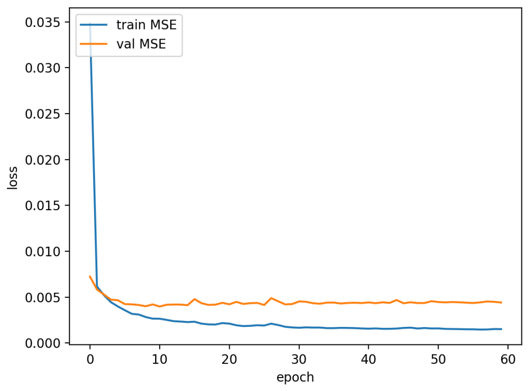

Figure 6: Mean squared error over 60 epochs for CNN Model 1 to learn mappings from RGB images to RAM states.

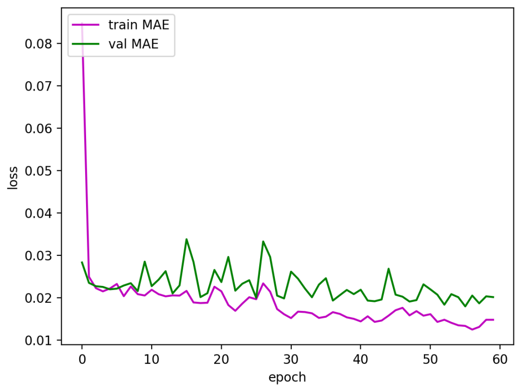

Figure 7: Mean absolute error over 60 epochs for CNN Model 1 to learn mappings from RGB images to RAM states.

4.1.3 CNN Model 2

We now extend our original CNN Model by adding max pooling layers in between the convolution layers, and adding a dense layer of size 1000 before the final layer. The architecture is shown in figure 8.

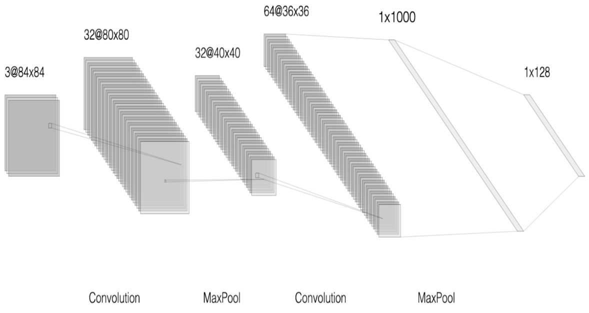

Figure 8: Architecture of CNN Model 2

This is the best performing model so far, with a validation MSE of 0.37% and MAE of 1.68%. The plots are shown in figures 9 and 10.

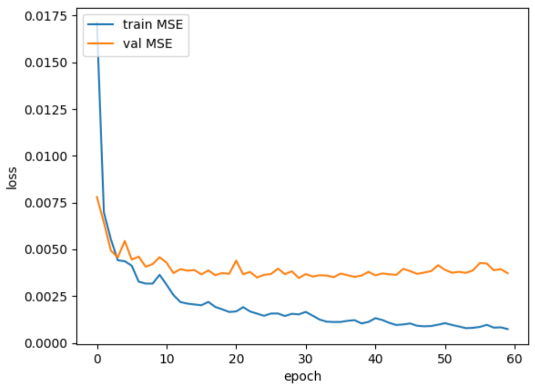

Figure 9: Mean squared error over 60 epochs for CNN Model 2 to learn mappings from RGB images to RAM states.

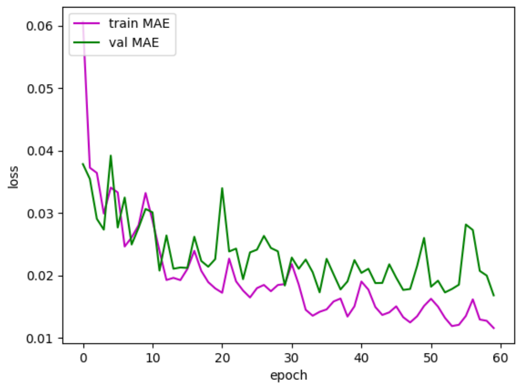

Figure 10: Mean absolute error over 60 epochs for CNN Model 2 to learn mappings from RGB images to RAM states.

4.1.4 LSTM

Our training samples are essentially snapshots in time of the state of the game. Since there is a temporal relation between the training examples, it follows that a recurrent neural network would do well. Since the input data is an image type, we require convolutions in the initial layers of our network. Thus, we use a convolutional LSTM architecture. An LSTM (Hochreiter and Schmidhuber, 1997) predicts the output at a current timestep based on the input at that time step as well as the last n inputs in time. We refer to n as the sequence length. Our main CNN-LSTM architecture is shown in figure 11. The figure depicted has a sequence length of 4. We experimented with different sequence lengths to learn the mapping from RGB images to RAM outputs, the results of which are shown in table 1.

seq len	Train MSE	Train MAE	Val MSE	Val MAE
2	0.11	1.41	0.38	1.96
3	0.18	1.93	0.37	2.25
4	0.21	2.03	0.37	2.35
5	0.35	2.45	0.42	2.66
10	0.46	2.89	0.49	2.92

Table 1: Mean squared error (%) and mean absolute error (%) on train and validation sets for different LSTM sequence lengths.

From table 1, we see that the best performance on the validation set is with a sequence length of 3. The data collection was done not by collecting consecutive frames of the game, it was done by collecting about 20 frames every 10 episodes. Thus, having larger sequence lengths will turn out to be counterproductive for the LSTM, as the sequence may include frames that were too far back in time to have any impact on the current output and may even be from different episodes. Moreover, the initial training set has 862 samples. With a sequence length of 2, this gets reduced to 431 samples with each sample having 2 snapshots. With a sequence length of 3, the training set size becomes one-third the original size. Thus, with an increase in sequence length the number of training examples also reduces. Hence at the end of 60 epochs the training error is higher for greater sequence lengths. Increasing the sequence length of the LSTM would require longer training and collecting more data sampled at a higher frequency for it to give the best results. The MSE and MAE over the course of training for an LSTM of sequence length 3 is shown in figures 12 and 13.

4.2 Training RL Agent

We train each RL agent for 1750,000 steps, which plays for 4000 to 5000 episodes depending on the

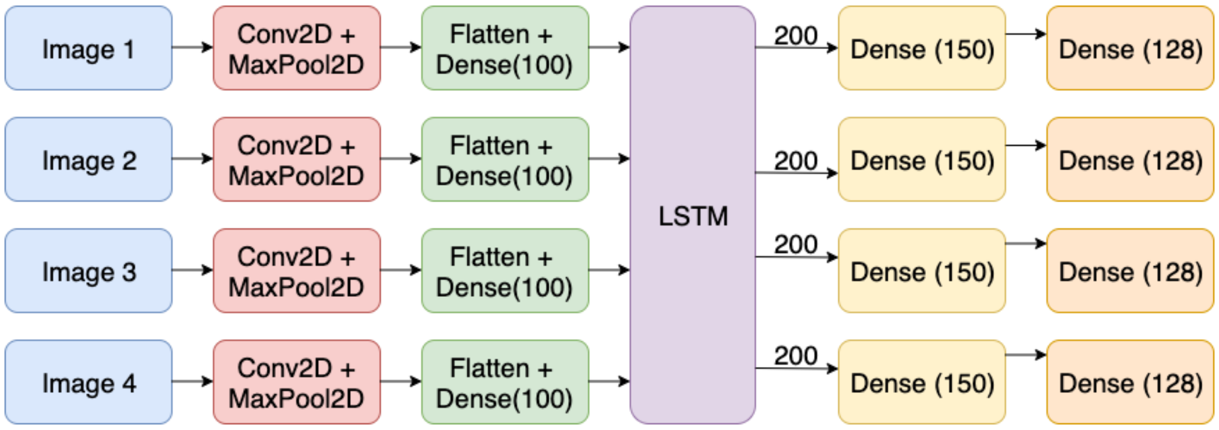

Figure 11: Architecture of Convolutional LSTM network with sequence length 4

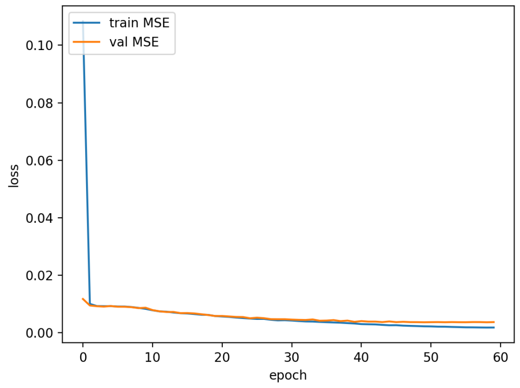

Figure 12: Mean squared error over 60 epochs for a convolutional LSTM of sequence length 3 to learn mappings from RGB images to RAM states.

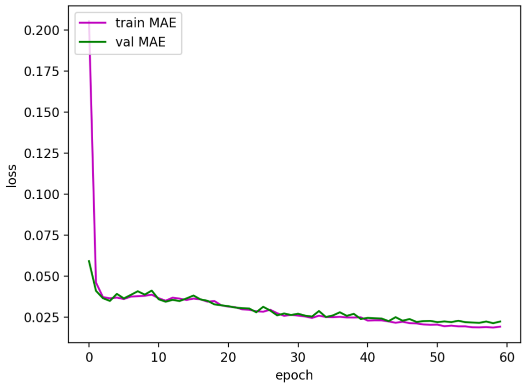

Figure 13: Mean absolute error over 60 epochs for a convolutional LSTM of sequence length 3 to learn mappings from RGB images to RAM states.

agent's performance. We use an ϵ-greedy action selection policy, where a random action is selected with probability ϵ. The higher ϵ is, the more likely the agent will explore its actions instead of following the policy. ϵ is annealed linearly from 1.0 to 0.1 over 1 million steps. As a result, the agent tends to explore the environment more at earlier stages (with high ϵ) and then gradually tends to follow the policy it has learned (with low ϵ). During testing time, we set $\epsilon = 0.05$ so that there is still a chance for random actions. If the agent gets stuck, it will have a chance to recover from the mistake.

4.2.1 Architectures for RL Agents

In this project, we have built three RL agents: one is trained with RGB inputs and serves as the baseline (referred to as the RGB agent), two are trained with RAM inputs (referred to as the RAM agents). For the RGB RL agent, we follow the paper (Mnih et al., 2013) and use a CNN architecture for the Deep Q network. The first convolution layer applies 16 8×8 filters with stride of size 4 on the input image of size 84×84. Then a Rectified Linear Unit (ReLU) (Nair and Hinton, 2010) is applied on top of the convolution layer. The second convolution layer applies 32 4×4 filters with stride of size 2, again followed by a ReLU nonlinearity. Then there is a fully-connected hidden layer with 256 ReLU units. Finally the output layer is a fully-connected linear layer with a single output for each valid action.

For the RAM RL agents, we adapt the architectures used in (Sygnowski and Michalewski, 2016) that achieve the state-of-the-art results for DQN agents playing Atari games. There are two architectures: just-RAM and big-RAM. They are both Feed-Forward Neural Networks with 128 ReLu hidden units in each hidden layer. just-RAM has 1 hidden layer and big-RAM has three hidden layers. On top of the hidden layers is a fully-connected linear layer, again with a single output for each action.

RL Agent	Training Time	#Parameters
RGB-CNN	10h 58m 24s	1,686,180
big-RAM	4h 3m 4s	68,100
just-RAM	4h 56m 3s	35,076
Input Mapping		
FF	$\approx$ 20 s	236,256
CNN	$\approx$ 30 m	20,917,224
LSTM	$\approx$ 6 m	5,411,210

Table 2: Training time and number of parameters of the three RL agents and mapping models. It is much more efficient to train an RL agent in the RAM environment and transfer to RGB environment in terms of time and space.

4.2.2 Experimental Results

Table 2 shows the training time and number of parameters of the three RL agents and the mapping models. As we predicted, since the RAM input is a more compact form compared to the RGB input we require networks that have much fewer parameters and shorter training times. The mapping models take a maximum of 30 minutes to train for 60 epochs. In total, for our transfer learning framework, the training time is about 5 hours, which is only half the time it takes to train an agent under the RGB environment. Since the number of parameters has reduced by an order of 3 it is possible to train our model in a less powerful machine with limited amount of CPU resources. Thus our framework suits applications where limited resources are available.

Table 3 displays the average and standard deviation of the rewards gained by each agent per episode of a game. The RGB RL agent has accumulated the highest average rewards of 40.4, and is able to achieve 50 rewards in the best episode. just-RAM with no transfer accumulated an average of 11.1 rewards per episode, and big-RAM with no transfer accumulated an average of 11.1 rewards per episode. From our results, there is no significant different between using a larger network (big-RAM) and a smaller network (just-RAM) under the RAM environment.

The rewards of feeding mapped RGB inputs to the RAM agents are included in columns 3-5 in Table 3. Using CNN mapping with the big-RAM agent gains the most average rewards per episode: 3.29, and is 27.5% of what the non-transfer model gains. With just-RAM, it gains 17.12% of the rewards of the non-transfer model. As we expected, for FF and CNN, the better the mapping models, the better the transfer results. Lower MSE and MAE of the mapping models indicate less information loss, hence when the RGB inputs of a new episode of game are mapped to RAM the agent can perform based on more accurate inputs.

For LSTM, however, we observe that transfer learning does not work at all. Both agents get 0 reward during episodes of the game, indicating that the agent is no better than acting randomly. By closely examining our methods, we find out this behavior is due to the way we generate our training data for the LSTM mapping model. For LSTM, we want to take advantage of the time relationship between frames: we take four frames sampled at a frequency of $1/20$ frames and feed them into the LSTM model. The LSTM model is able to identify how the frames change in time and is able to accurately predict what RAM state the last frame corresponds to. However, when training the RL agent, we apply the technique called *experience replay* (Mnih et al., 2013) where the agents' experiences at each time-step are stored in memory. When we do Q-learning updates, a minibatch of 4 experiences are randomly sampled from the memory pool to form of a sequence of RGB inputs of size 4. Although the size of the inputs match what we use to train the input mapping we lose the time relationship of the images since they are all randomly selected. In this case, it deviates from the basic principle of LSTM, which depends heavily on the sequential relationship of its inputs.

5 Discussion

In this section, we evaluate our works, analyze the limitations and list our contributions. We also include possible future works extending from our idea.

We have established that it is not necessary to train different RL agents for different input types of Atari games as what has been done in the past (Mnih et al., 2013) (Sygnowski and Michalewski, 2017). We are able to train an RL agent using the RAM state inputs and apply it using transfer learning to an environment where only RGB inputs are available. We show that our approach is more time and space efficient. The main contributions of our work include:

1. We are the first to attempt to unify the training of RL agents for Atari games with two different types of inputs;
2. We demonstrate that RL agents trained with RAM state inputs can be successfully transferred to RGB environments;
3. We show that there is a strong correlation between the information contained in the RGB

RL Agent	No Transfer	Transfer (FF)	Transfer (CNN)	Transfer (LSTM)
RGB-CNN	40.4 ± 9.87	-	-	-
big-RAM	11.95 ± 5.26	0.15 ± 0.37	3.29 ± 1.59	0.00 ± 0.00
just-RAM	11.10 ± 5.19	0.85 ± 0.49	1.90 ± 1.62	0.00 ± 0.00

Table 3: Rewards per episode gained by each agent. The second column denotes the rewards when no transfer is applied and agents are trained on the original RGB or RAM inputs. The rightmost three columns include the rewards when the RGB observations are mapped to RAM states using FF, CNN, and LSTM networks before being fed to the RAM-input RL agent.

images of an Atari game and that of the corresponding RAM state;

4. We propose a convolutional LSTM architecture to map sequences of RGB images to that of RAM states with minimal loss of information.

As we discussed in Section 4, the LSTM mapping model fails to help transferring the RL agent due to the use of experience replay, where the input image sequence is sampled randomly instead of sequentially. One possible way to make use of LSTM is to change the input image sequences we use to train LSTM as well. We can save RGB observations and RAM states as before, but when we train LSTM we randomly sample 4 images from all the training samples to mimic experience replay. Although this way the model will not learn how one image develops to another temporally, it is still able to learn the invariance between images such as the unchanged background, which may help the model to predict RAM states accurately. Another way is to change the way the RL agent is trained. Instead of using experience replay have it sample consecutive image sequences, thus learning as a human would. This would allow LSTMs to work very well as the power of temporal input can be used to make decisions and shape the policy of the RL agent.

Another major limitation of our work is that we have only experimented with one Atari game, Breakout. As we have shown that it is possible to recover the RAM state vector from the RGB screen of Breakout, the same principle can be applied to all of the Atari games. One possible extension of our idea is to build a universal mapping between the RGB inputs and the RAM inputs for any Atari game. Thus for any Atari game, having the universal mapping will render it sufficient to train a single RL agent for either RGB environment or RAM environment. This requires much more training data extracted from various Atari games, possibly a more complex network architecture to do the mapping, and has a lot of potential worth exploring.

6 Conclusion

Transfer learning has a lot of potential in the field of reinforcement learning. Our work shows that the input representation does not matter when trying to train an RL agent. This has implications beyond Atari games, for instance when training an agent to do a task we do not have to rely on difficult to manage high dimensional visual and sensory input, we can convert that to a representation that is far more compact and easier to train the agent with. This can speed up a lot of reinforcement learning as well as neuroevolution tasks, and has great promise for the future of research in these fields.

Acknowledgments

We are grateful to Professor Risto Miikkulainen for his feedback and encouragement throughout the semester. We are also thankful to our fellow classmates who gave us insightful questions and comments during our presentations.

References

David Andre and Stuart J. Russell. 2002. State abstraction for programmable reinforcement learning agents. In *Eighteenth National Conference on Artificial Intelligence*, pages 119–125, Menlo Park, CA, USA. American Association for Artificial Intelligence.

Mehran Asadi and Manfred Huber. 2007. Effective control knowledge transfer through learning skill and representation hierarchies. pages 2054–2059.

Christopher G. Atkeson and Juan Carlos Santamaria. 1997. A comparison of direct and model-based reinforcement learning. In *IN INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION*, pages 3557–3564. IEEE Press.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*.

Jeff Donahue, Lisa Anne Hendricks, Marcus Rohrbach, Subhashini Venugopalan, Sergio Guadarrama, Kate Saenko, and Trevor Darrell. 2017. Long-term recurrent convolutional networks for visual recognition and description. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(4):677–691.

Nicolas Heess, David Silver, and Yee Whye Teh. 2013. Actor-critic reinforcement learning with energy-based policies. In *Proceedings of the Tenth European Workshop on Reinforcement Learning*, volume 24 of *Proceedings of Machine Learning Research*, pages 45–58, Edinburgh, Scotland. PMLR.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9:1735–80.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *International Conference on Learning Representations*.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. 2013. Playing atari with deep reinforcement learning. *ArXiv*, abs/1312.5602.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Vinod Nair and Geoffrey E. Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA. Omnipress.

Matthias Plappert. 2016. keras-rl. `https://github.com/keras-rl/keras-rl`.

Brian Sallans and Geoffrey E. Hinton. 2004. Reinforcement learning with factored states and actions. *J. Mach. Learn. Res.*, 5:1063–1088.

Richard S. Sutton, Anna Koop, and David Silver. 2007. On the role of tracking in stationary environments. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 871–878, New York, NY, USA. ACM.

Jakub Sygnowski and Henryk Michalewski. 2016. Learning from the memory of atari 2600. *CoRR*, abs/1605.01335.

Jakub Sygnowski and Henryk Michalewski. 2017. Learning from the memory of atari 2600. pages 71–85.

Matthew E. Taylor and Peter Stone. 2009. Transfer learning for reinforcement learning domains: A survey. *J. Mach. Learn. Res.*, 10:1633–1685.

Gerald Tesauro. 1995. Temporal difference learning and td-gammon. *Commun. ACM*, 38(3):58–68.

L. Torrey and J. Shavlik. 2009. Transfer learning. *Handbook of Research on Machine Learning Applications*.

J. N. Tsitsiklis and B. Van Roy. 1997. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690.