Deep Reinforcement Learning using Asynchronous Advantage Actor-Critic (A3C) method with Progressive Neural Network

Hengda Shi **Gaohong Liu** **Jintao Jiang**
{hengda.shi, cheimu, jiang1995}@cs.ucla.edu
Department of Computer Science
University of California, Los Angeles
Los Angeles, CA 90095

Abstract

Traditional transfer learning technique has a prolong problem in catastrophic forgetting that prevents it to achieve human-level intelligence. Progressive Neural Networks [12] offers a solution that can both leverage previous knowledge and avoid catastrophic forgetting via lateral connections to previously learned weights. In our project, we plan to re-implement Progressive Neural Networks and deploy it on a set of Atari games to simulate its performance. There are several improvements over the original paper including the use of Adam optimizer, normalization of image data, and slightly changed neural network structure. We were able to achieve an weighted average of 18 in around 0.87 million steps for a single column of a vanilla Pong game, and around 0.42 million steps by transferring from a noisy version of the Pong game to vanilla Pong game.

1 Introduction

Recently, deep reinforcement learning suggests great potential in reinforcement learning field, such as [14, 16]. Transfer learning is thus natural adopted in deep reinforcement learning due to its widely usage in the deep learning field. However, traditional transfer learning techniques suffer from catastrophic forgetting if the frozen layers get unfrozen and retrained on the target domain. How to improve the transfer effectiveness of deep reinforcement learning remains as an important task in the field. In addition, same scenario can be applied to the supervised learning domain because the transfer learning technique is the same. In this re-implementation, we will explain in detail how does Progressive Neural Networks tackle these problems and achieve an impressive performance. We have re-implemented the original progressive neural networks and A3C method which utilizes multi-agents to update one global agent for fast convergence. Generalized advantage estimation is utilized to effectively reduce the variance of the advantage estimation. Adam optimizer is also utilized as a replacement to RMSprop which provides momentum and bias correction. Image pre-processing techniques including normalization of the input image and turning image to grayscale were utilized to improve convergence speed. In our results, we achieved 6%x to 50%x training speed up.

2 Related Work

In recent years, deep learning systems have demonstrated tremendous potential in their ability to perform complex cognitive tasks that were previously thought to be only achievable by humans. These systems can recognize objects [8], locate objects[9], reason about relations[1] and play complex video games[18]. Because of the ability of deep learning on vision tasks, there are researches[21] using deep neural networks such as ResNet[5] to generate the image embedding which could work as both feature extraction and function approximation that could be useful to understand the images and find the relations between objects inside the images rather than simply recognizing the image.

Nowadays, deep Q learning, an off-policy model-free reinforcement learning algorithm performs very well. Deep Q network seeks to find the best action to take given the current state. It seeks to learn a policy that maximizes the total reward. [11, 19, 17, 13, 6] have demonstrated the power of deep Q networks. Deep Q networks successfully learn control policies from raw video data in complex reinforcement learning environment where deep Q networks would be working as a function approximator to give a both quality and value estimation on the video games frames such as Atari 2600 games. In addition to use deep Q networks to approximate Q function, the advantage function proposed by [19] improves the estimation quality by calculating the difference between Q function and V function. It indicates the extra reward that could be obtained by the agent by taking that particular action. In other words, advantages functions not only can tell how well an action performed, but also how well it behaved comparing with other actions.

Policy gradients method [7, 15, 10, 3, 4], targeting at modeling and optimizing the policy directly, tries to find an optimal behavior strategy for the agent to obtain optimal rewards and this method has been widely adapted recently. Specifically, actor-critic method [3, 4, 10, 20] where critic updates the value function parameters by evaluating the value functions, quality function or the advantage function, and actor updates the policy parameters in the direction suggested by the critic makes policy gradients more practical and efficiency. To improve the efficiency of actor-critic method, multi-agent actor-critic is proposed [10], where there will be multiple agents running local-network to evaluate the value functions using multi-process and synchronously updates the weights to the global network. In this way, there will be more emulators running at same time to improve the training efficiency.

3 Model

In this section we will discuss the design of our project, that is a general neural network working as a function approximator to learn the quality function working for multiple environments of same type, an asynchronous multi-agent actor-critic framework for deep reinforcement learning. We will discuss each framework in details but start with an overall architecture.

3.1 Overall Architecture

Our overall architecture is shown in Figure 1 (Left). Where there is a global neural network working as the function approximator for the output policy, and there will be multiple agents running locally and aggregate updates asynchronously to the global neural network. For each agent process, the workload is shown in figure 1(Right). The same advantage actor-critic algorithm and advantage function are calculated using both Q function and value function with Progressive Neural Network. When there is at least one agent that finished emulation in one time, the updates will be sent to the global neural network asynchronously.

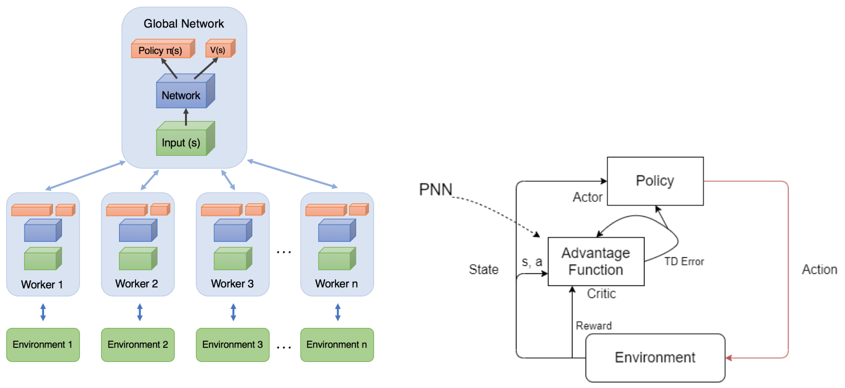

Figure 1: (Left) Overall Architecture. (Right) Architecture of each Agent

3.2 Progressive Neural Networks

Environments can be similar for the same type of environments, such as First-person shooting games, Role-playing games, which means it is possible to find a general deep Q network that shares similar knowledge to speed up the training of deep Q network process. However, it is hard to find a general neural network architecture for all the environments. A common transfer learning technique called Fine-tuning would pretrain the model on massive dataset and replace the output layer with output dimension of the target domain, and unfreeze the layers further fine tuning the neural network. During the retraining on the target domain, information stored in the previous weights would be forgotten. This problem is called catastrophic forgetting.

In order to find a trade-off between fine-tuning and previous knowledge, we utilize the Progressive Neural Networks [12] shown in 2. The intuition of the Progressive Neural Networks is that a new neural network (a **column** in PNN) is instantiated for each task being solved, while transferring is enabled via lateral connections from all previously learned columns. For each column itself, which is a deep neural network for a specific task, entire column can be fine-tune with previous columns' weights frozen, so that prior knowledge would not be forgotten and we could also fine-tune current column's weights.

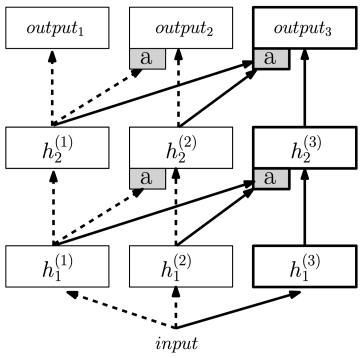

Figure 2: Progressive Neural Network

To train such model, a progressive network starts with a single column: a deep neural network having L layers with hidden activation $h_i^{(1)} \in \mathbb{R}^{n_i}$ with n_i the number of units at layer $i \leq L$, and parameters $\Theta^{(1)}$ trained to convergence.

When switching to a second task, the parameters $\Theta^{(1)}$ are "frozen" and a new column with parameters $\Theta^{(2)}$ is instantiated (with Xavier initialization), where layer $h_i^{(2)}$ receives input from both $h_{i-1}^{(2)}$ and $h_{i-1}^{(1)}$ via lateral connections. This generalizes to K tasks as follows:

$$h_i^{(k)} = f(\mathbf{W}_i^{(k)} h_{i-1}^{(k)} + \sum_{j<k} U_i^{(k:j)} h_{i-1}^{(j)})$$

where $\mathbf{W}_i^{(k)} \in \mathbb{R}^{n_i \times n_{i-1}}$ is the weight matrix of layer i of column k, $U_i^{(k:j)} \in \mathbb{R}^{n_i \times n_{i-1}}$ is lateral connections from layer $i-1$ of column j to layer i of column k, h_0 is the network input and f is the element-wise activation function. This would be the progressive neural network computation flow for a single column. Note that there is no lateral connection on the first column so that the equation would be simplified as:

$$h_i^{(1)} = f(\mathbf{W}_i^{(1)} h_{i-1}^{(1)})$$

A problem with this setting is that previous columns output will be accumulated and added to the current layer output. It can easily dominate the current layer output such that current layer output would contribute little to the final output. Therefore, as proposed by the original paper, the lateral connections is changed to the following:

$$h_i^{(k)} = f(\mathbf{W}_i^{(k)} h_{i-1}^{(k)} + \sum_{j<k} U_i^{(k:j)} f(V_i^{(k:j)} \alpha_{i-1}^{(<k)} h_{i-1}^{(j)}))$$

α would be serving as a learnable parameter that can be tuned to adjust different scales of different input tasks. It can help the current layer output to be still effective on learning. V serves as a dimensionality reduction layer that would perform 1×1 convolutions if the previous layer is a convolutional layer.

After each column has been trained, we need to freeze all previous columns so that their weights would no longer be updated by gradient descent. It can be easily achieved by using PyTorch API. We show an sample code below to demonstrate how to freeze columns:

```python
def freeze(self):
    """freeze the previous columns"""
    for i in range(self.cid):   # self.cid represents the number of
    ↪   current columns in the network
        for params in self.columns[i].parameters():
            params.requires_grad = False
```

3.3 Asynchronous Multi-Agent Advantage Actor-Critic Method

As we mentioned before, for on-policy algorithms where the learning occurs by following a fixed policy, data dependency exists so that neural network can only process data of batch size 1 with a single player. In other word, it is infeasible to only have one agent to emulate the game and policy because it will waste both time and resource. In order to resolve the problem of the data dependency, we utilize the A3C method, which shown in figure 3. Instead of one agent (player) interacting with the environment (playing a game) and playing under the policy while training, we have multiple agents interacting with different version of the environment. For instance, if we play Super Marios game, we could have multiple agents playing at different levels of the game and use the same policy (generated with one neural network weights).

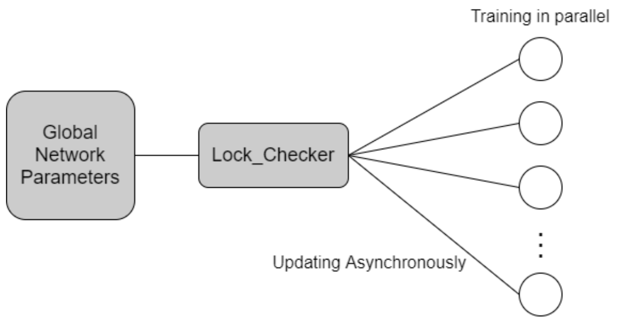

Figure 3: Asynchronous Multi-agent Advantage Actor-Critic architecture

For each agent, there will be one local network for itself which is a local version of progressive neural network in the architecture. To update the policy, once one agent finished its advantage function computation, it will synchronize its gradient with the global neural network . Because of the existence of lock, when there is one agent updating the global neural network, other agents can only wait until the shared global neural network is unlocked, by doing so, we can make sure that there is no data racing between each agent which could totally destroy the training process.

What's more, when the global network can achieve a weighted average exceed the threshold reward of the specific environment, it needs to update its current column to the next column. A shared variable that keeps track of the current column running is included in the global model. Local models would constantly check the shared variable before updating its gradient to global model so that it can properly move on to the next column.

4 Evaluation

4.1 Experiment Setup

Our code in written in PyTorch, which is an open-source framework that can be utilized to build computational graph of neural network. The neural network of a single column consists of 4 convolutional layers, 1 MLP, and 2 layers on the same level for policy and value prediction. The convolutional layers are as follows. They all have 32 feature maps, kernel size of 3, stride size of 2, and padding size of 1. We used ELu as the activation function since it does not entirely zero out the negative output. Hyper-parameter was chosen identically as the original paper. Learning rate is set to 10^{-4}, max gradient norm is set to 40, alpha is randomly picked from $\{10^0, 10^{-1}, 10^{-2}\}$.

The experiment is run on AWS c5.4.xlarge machine which has 16 cores to fully utilize the multi-processing setting. Python has a limitation called Global Interpreter Lock which allows only one thread to run the actual python bytecode. Therefore, instead of using multithreading, we used multi-processing to deploy local agents.

For the OpenAI Gym environment, we used only the deterministic verions of the game because it would constantly skip 4 frames. It would be easy to train comparing with using the non-deterministic version which would randomly skip 2-4 frames because the resulting frame after taking an action would be deterministic.

4.2 Training Results and Analysis

In this section, we present our training results. There are three types of experiment: single column (table 1), double columns (table 2), triple columns (table 3). Single column is as same as training neural network for each game individually. There is no transfer learning and each neural network is fine-tuned. Double columns and triple columns uses the advantages of PNN where after the first column(task/game) is trained, the following column can then take activations from previous column and this is where transfer learning happened.

Game Version	Steps
PongFlipDeterministic-v4	1037062
PongDeterministic-v4	874249

Table 1: Steps Taken before Convergence using Single Column

Game Version	Steps
PongNoisyDeterministic-v4	2032417
PongDeterministic-v4	423078

Table 2: Steps Taken before Convergence using Second Column (Training In Order)

Game Version	Steps
PongNoisyDeterministic-v4	1514485
PongDeterministic-v4	224385
PongFlipDeterministic-v4	717777

Table 3: Steps Taken before Convergence using Triple Column (Training In Order)

Starting by comparing table 1 with table 2, we can see that, the training time of PongDeterminstic-v4 using double columns is only half than that using single column where only train and fine-tune PongDeterminstic-v4 itself. Similarly, by comparing 1 with table 3, we can see that the training PongFlipDeterministic-v4 using single column is also speed up. The transfer effectiveness has shown in figure 4. This figure was drawn using data by comparing table 1 with table 3. It is clearly that the effects of transfer learning enabled by PNN are working.

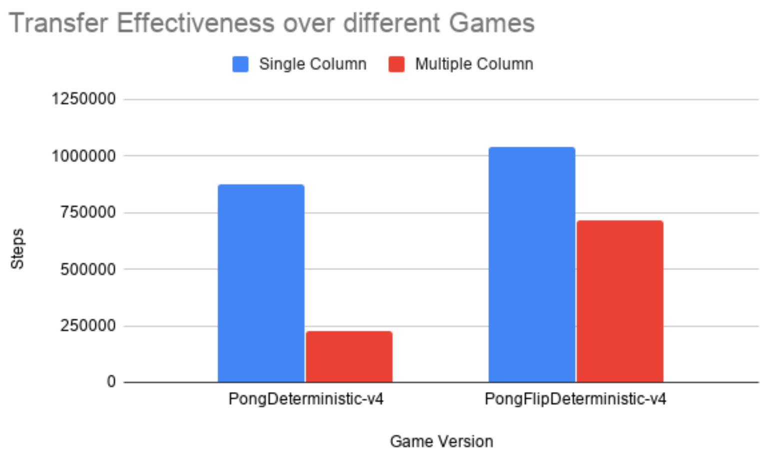

Figure 4: Transfer Effectiveness over different Games

5 Conclusion and Future Work

In this project, we tries to combine multiple level optimizations to improve the performance of deep reinforcement learning. We provide three contributions:

1. A general neural network working as deep Q networks to learn the quality function working for multiple environments in same type.
2. An asynchronous multi-agent advantage actor-critic framework for deep reinforcement learning.

In the future, this projects can be further improved in the following ways.

1. CuLE[2] is an GPU environment support for reinforcement learning provided by NVidia where it can render one environment per thread. Now we have everything running on CPU, so could we run everything on GPU to accelerate the training of A3C method is one of the research direction.
2. An known issue of Progressive Neural Networks is that only a fraction of the previous column's information can be utilized. Can we possible add a pooling layer or perform an online compression to prune part of the information from previous columns (It can also potentially speedup the training since part of the network output is pruned)?

6 Statement of Work

We divide the design of architecture evenly including pair programming. Thanks Hengda for the design of exhaustive experiments.

References

[1] Clark, P., Tafjord, O., and Richardson, K. Transformers as soft reasoners over language. *arXiv preprint arXiv:2002.05867* (2020).

[2] Dalton, S., Frosio, I., and Garland, M. Gpu-accelerated atari emulation for reinforcement learning. *arXiv preprint arXiv:1907.08467* (2019).

[3] Degris, T., White, M., and Sutton, R. S. Off-policy actor-critic. *arXiv preprint arXiv:1205.4839* (2012).

[4] Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905* (2018).

[5] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition, 2015.

[6] Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence* (2018).

[7] Kakade, S. M. A natural policy gradient. In *Advances in neural information processing systems* (2002), pp. 1531–1538.

[8] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (2012), pp. 1097–1105.

[9] Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision* (2017), pp. 2980–2988.

[10] Lowe, R., Wu, Y. I., Tamar, A., Harb, J., Abbeel, O. P., and Mordatch, I. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in neural information processing systems* (2017), pp. 6379–6390.

[11] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[12] Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. Progressive neural networks. *CoRR abs/1606.04671* (2016).

[13] Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. *arXiv preprint arXiv:1511.05952* (2015).

[14] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. Mastering the game of go with deep neural networks and tree search. *nature 529*, 7587 (2016), 484.

[15] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. Deterministic policy gradient algorithms.

[16] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. Mastering the game of go without human knowledge. *Nature 550*, 7676 (2017), 354–359.

[17] Van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence* (2016).

[18] VINYALS, O., BABUSCHKIN, I., CZARNECKI, W. M., MATHIEU, M., DUDZIK, A., CHUNG, J., CHOI, D. H., POWELL, R., EWALDS, T., GEORGIEV, P., ET AL. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature 575*, 7782 (2019), 350–354.

[19] WANG, Z., SCHAUL, T., HESSEL, M., VAN HASSELT, H., LANCTOT, M., AND DE FREITAS, N. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581* (2015).

[20] WENG, L. Policy gradient algorithms. *lilianweng.github.io/lil-log* (2018).

[21] ZHAO, J., WANG, T., YATSKAR, M., ORDONEZ, V., AND CHANG, K.-W. Men also like shopping: Reducing gender bias amplification using corpus-level constraints. *arXiv preprint arXiv:1707.09457* (2017).