Jirat Manpadungkit
Student ID: 20320731

Investigating the Impact of Transfer Learning Direction on Generalisation in Reinforcement Learning

Jirat Manpadungkit

This paper investigates the topic of transfer learning and aims to answer the following question: "Which transfer learning direction produces a more generalised model, simple-to-complex or complex-to-simple?" The environment that will be used for this experiment is VizDoom wrapped in OpenAI Gym framework. Through multiple experiments, this paper shows that transferring knowledge must be done from simple to complex for effective knowledge sharing.

I. INTRODUCTION

Reinforcement Learning (RL) is a machine learning technique, where an agent is incentivised to maximise rewards through making decisions based on past experiences interacting with an environment. While RL models have achieved great success in the specific environments they had been trained on, they often struggle to generalise their learning to new, unseen environments.

Transfer Learning (TL) allows models to leverage their knowledge learned and apply them on a different task. This can possibly lead to the reduction of training time, improved performance, and overall better generalisation capability **[1]**. However, not a lot of research has been done on how the direction of transfer learning affects the generalisation ability of an RL model. In this paper, we train two agents to play Doom, one starts learning in a simple Doom environment then transfers its learning to a complex environment, while the second agent learns in the opposite order. The thinking behind this idea is that the two agents will get to learn and understand all the possible moves but in different orders. The hypothesis is since they both will have learnt all the moves; they should be able to generalise well in other environments that has the same actions and similar objectives. The two agents will then be evaluated on unseen environments to compare their generalisation.

The results of this study shed light on the significance of transfer learning direction and provide insights for efficient transfer learning techniques. These findings are important because they increase the effectiveness of training new models by allowing us to use a model's pre-existing knowledge to cut down on computation time and power.

II. BACKGROUND

In real world situations, environments are complex and uncertain. For an RL model to perform well, it has to be able to adapt and respond well to changing environments. This is why achieving generalisation is an important goal, as a generalised model is equipped to handle uncommon situations. **[1]** But this is extremely difficult due to the fact that models tend to overfit to the specific environment they were trained on.

Though the reinforcement learning field has made tremendous advances in recent years, including AlphaGo **[3]** and OpenAI Five **[2]**, a key challenge of achieving generalisation still remains **[6]**. Transfer learning is one of the approaches to tackle this obstacle.

In their insightful paper **[4]**, they were able to successfully create a model that was pre-trained in the game Puck World, which was able to improve performance and stability in learning to play the game Snake. Keep in mind these two environments have relatively similar objective and same number of actions, therefore they were able to transfer relevant knowledge effectively. Many other papers were able to produce similar results. However not much research has been done on transfer learning in environment with different complexity. Complexity in this context is the number of possible actions and state space.

This is why in this project; we will experiment with different methods of weight transfer while doing transfer learning in environments of varying complexity. We will investigate whether or not the models can effectively transfer knowledge. Finally, we will use unseen environments to assess how well the models generalise.

III. TECHNOLOGIES

This section will be about the different technologies use to implement this project.

A. VizDoom

Doom is a famous first-person shooter game released in 1993. The main character is a space marine that goes through various levels killing monsters. VizDoom provides an interface between the game and reinforcement learning algorithms allowing researchers to develop agents to play the game, moreover it has various scenarios or environment that an agent can interact with. According to **[13]**, transfer learning can only happen effectively if the task of the different environments is sufficiently similar. The scenarios below were then chosen due to their similarity in objective, which is finding enemies and shooting them **[5]**. The details of each scenario are found below, and the details of their reward system can be found in **Appendix 1**.

1. *Basic Scenario*

The basic scenario is one of the simplest environments in VizDoom. A player spawns in a rectangle map and on the other side of the wall there is a monster that the player must shoot. Game finishes when the monster is killed or on timeout. The player has only 3 action spaces: Move_Right, Move_Left and Attack. We used this level as the "simple" environment.

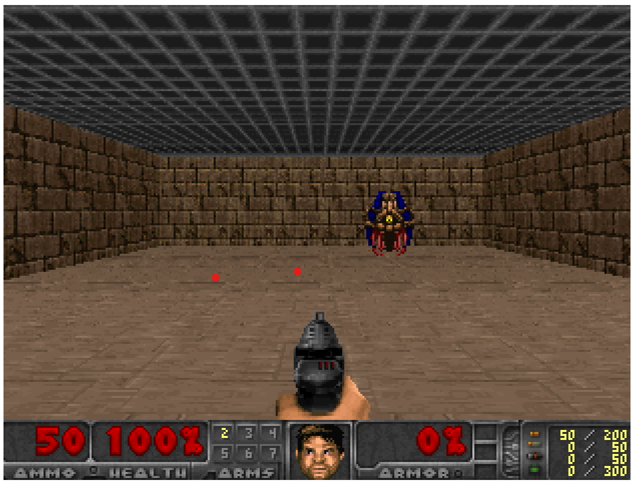


Figure 1 Basic Scenario

2. Deadly Corridor

The deadly corridor is one of the most complex environments in VizDoom. A player spawns in a narrow corridor with attacking enemies on both sides of the wall. To win the game, the player must collect the armour by navigating to the end of the corridor without getting killed. There are a total of 7 action spaces: Move_Left, Move_Right, Attack, Move_Forward, Move_Backward, Turn_Left and Turn_Right. We used this level as the "Complex" environment.

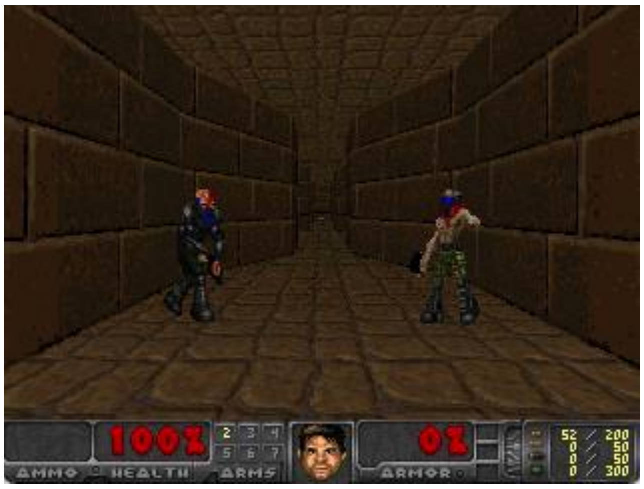


Figure 2 Dead Corridor Scenario

3. Defend the Centre

Defend the centre is a scenario where the player is in a circular map, and enemies are coming from all directions. Player is stationary and can only turn left and right to shoot the incoming monsters. The monsters are melee-only and continuously spawn until the player runs out of ammo and dies. This scenario is used for evaluation.

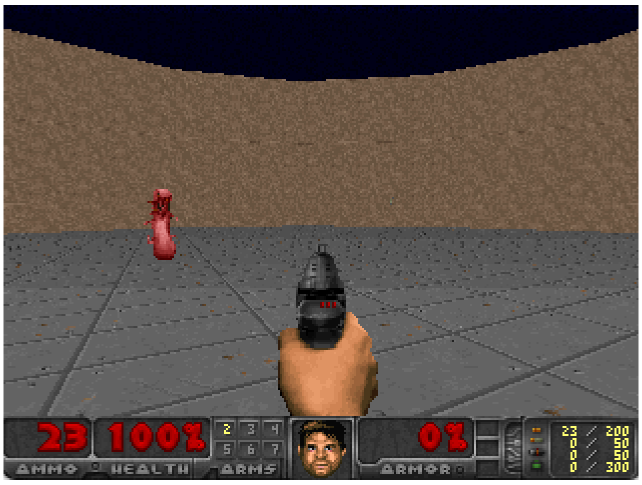


Figure 3 Defend the Center Scenario

4. Defend the Line

This scenario is similar to Defend the circle, where the player is stationary while shooting incoming monsters. But the enemies are not coming from all directions, instead they are all standing in a line, some are long-range attackers some are short-range attackers. This scenario is used for evaluation.

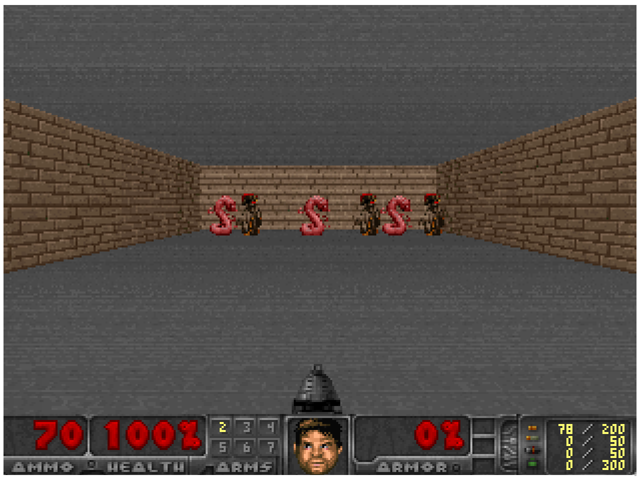


Figure 4 Defend the Line Scenario

5. Predict Position

This scenario teaches our agent to use a missile weapon and try to shoot a moving monster. The monster will be randomly spawned in a rectangle room and will move left and right along the opposite wall. There is significant delay between shooting and hitting, moreover the agent has only one bullet therefore it is necessary to hit the monster on the first try.

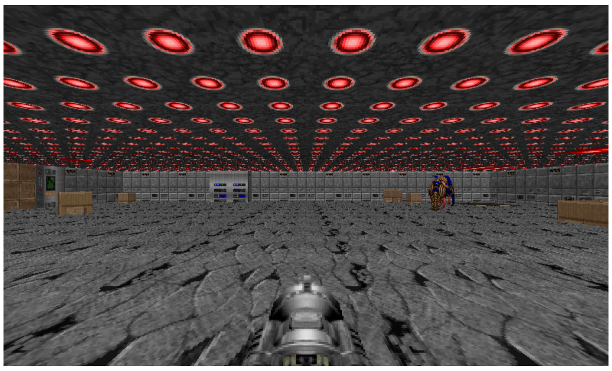


Figure 5 Predict Position Scenario

B. Stable-Baseline3

Stable-Baseline3 is a library in Python, that provides implementations of reinforcement learning built on Pytorch. It provides user-friendly interface for both training and evaluating, with access to various RL algorithms. **[7]**

The algorithm that will be used for training the agent is **Proximal Policy Optimisation (PPO). The reasons why PPO** was chosen is because the algorithm supports the discrete actions of VizDoom, more importantly it has proven to work extremely successfully with VizDoom in **[9].** PPO is a type of policy gradient algorithm, and it works by utilising its experience to optimise a surrogate objective function using stochastic gradient ascent. The goal of the stochastic gradient ascent is to increase the expected reward score by changing the policy network parameters.**[11]**

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t\left[\log \pi_\theta(a_t \mid s_t)\hat{A}_t\right].$$

Figure 6 where L^{pG} is the policy gradient objective function, expectation E_t is the Expectation over finite samples, πθ is a stochastic policy, and A_t is an estimator of the advantage function at timestep t. ***[21]***

C. OpenAI Gym

OpenAI Gym is a toolkit for developing and testing agents. Gym provides API for interacting with environments and is widely used in the RL community making it one of the standardised ways of evaluating the performance of different models. **[16]**

D. Optuna

Certain parameters in deep learning such as learning rate has a significant impact on the model, hence making hyper-parameter tuning extremely important and unfortunately the most difficult part of training RL models. The reason is because these models take extremely long to train, therefore it would be greatly time-consuming to manually tune the model. To tackle this problem, I utilised Optuna which can automatically find the optimal hyperparameter. It works keeping record of trails and using this data, it estimates a promising value to try next. Though not perfect and further manual tuning is needed, Optuna is able to give a very good rough guideline. **[19]**

IV. EXPERIMENTAL SET-UP

Before proceeding into the experimental set-up, it is important to understand the high-level overview of the direction being taken. The basic model will be used to learn the deadly_corridor environment and the deadly_corridor model will be used to learn the basic environment. These two models will represent our simple-to-complex and complex-to-simple models. Finally, they will be evaluated on the three other environments.

A. Wrapping VizDoom in OpenAi Framework

Since VizDoom is not one of the built-in environments of OpenAI, additional work has to be done in order for the environment to have the OpenAI Gym interface. This can be done by creating a new class inheriting from the Gym's ***Env*** class and implementing all the required methods of the class. Most importantly, the observation space and action space have to be defined accordingly to each environment. To make the training faster, the environment is set to black and white to reduce processing time. This part of the code was heavily inspired from **[9].**

B. Training Baseline Models

The first step taken was training each of the environments on its own: Basic, Deadly, Defend_the_Circle, Defend_the_Line and Predict_Position. Optuna and some manual tuning was used to optimise the hyper-parameters. All the models are tuned through trial and error until they can complete their tasks in most trials. The algorithm employed for all trainings is PPO and each environment went through different amount of timesteps as certain environments are simpler therefore requires less data for the reward function to converge. The full detail of hyperparameters can be found in **Appendix 2**.

1. *Training Deadly Corridor*

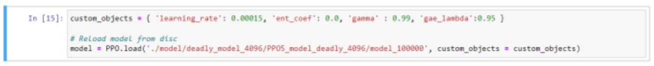
The deadly corridor scenario has 7 total action spaces, making it the most complex environment. Since it is the most complex, training an agent to successfully complete this level is challenging. Additionally, the monsters within the environment are so aggressive that once the agent spawns, it dies almost instantly. To effectively train our model, reward shaping is necessary to provide more guidance, so the agent knows what the desirable actions along the way so that it can finish its real task. Another addition to this environment is, I implemented curriculum training by starting the training from easiest mode and slowly progressing to the hardest mode. This is to give the agent more time to learn the environment and not die instantly.

2. *Training Other Environments*

Since, all the other environments are not as complex as the deadly corridor environment, not much extra has to be done. Only hyperparameter tuning is needed in order to produce a successful model.

C. Transfer Learning

To do transfer learning, a trained model is loaded, then its network has to be modified in order to learn a new environment. But if the model has the same number of outputs as the action spaces in the new environment, nothing has to be done to the network, besides reinitialising the output layer (explained in detail below). A loaded model will still have the old parameters of its last training, therefore, to change the parameters the model has to be loaded in along with a **custom_objects,** containing the information of the desired parameters.



```
In [15]: custom_objects = { 'learning_rate': 0.00015, 'ent_coef': 0.0, 'gamma' : 0.99, 'gae_lambda':0.95 }

         # Reload model from disc
         model = PPO.load('./model/deadly_model_4096/PPO5_model_deadly_4096/model_100000', custom_objects = custom_objects)
```

Figure 7 loading a pre-trained model with custom_objects

1. *Transfer Learning Limitations*

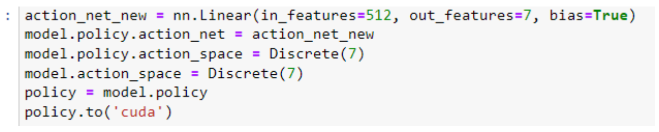
To do transfer learning on environments with different numbers of action spaces is difficult. This is because, the network of the model has already been built and cannot be changed to support a different number of actions/outputs. Furthermore, stable-baseline3 has no dedicated API for transfer learning making it even more challenging. To tackle this problem, I realised that we have to keep the ***n_steps*** and ***batch_size*** parameter constant through out every model so transfer learning can happen. The reason why is the two parameter determines how many timesteps to run before it updates the network. This causes a slight change in the reward function, consequently, transfer learning cannot happen.

2. *Network Customisation*

With ***n_steps*** and ***batch_size*** constant, transfer learning can now be done, but only with further modifications.

- Basic to Deadly_Corridor

  To modify the basic model to fit deadly corridor, the model's policy **action_net** layer, which is the output layer, has change from 3 to have 7 outputs. Furthermore, **action_space** needs to be modified from Discrete (3) to Discrete (7). Finally, use the function **set_env ()** to set the new environment for the model to learn.

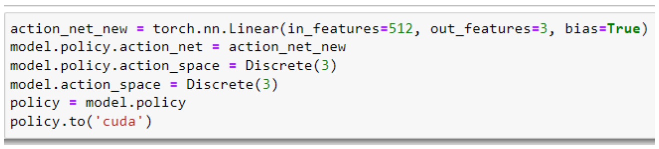


```
action_net_new = nn.Linear(in_features=512, out_features=7, bias=True)
model.policy.action_net = action_net_new
model.policy.action_space = Discrete(7)
model.action_space = Discrete(7)
policy = model.policy
policy.to('cuda')
```

Figure 8 Modifying output layer from 3 to 7.

- Deadly_Corridor to Basic

  To modify the deadly_corridor model to fit the basic environment. Similar process has to be done but in the opposite direction, basically changing from 7 to 3. Then set the environment to the new one.



```
action_net_new = torch.nn.Linear(in_features=512, out_features=3, bias=True)
model.policy.action_net = action_net_new
model.policy.action_space = Discrete(3)
model.action_space = Discrete(3)
policy = model.policy
policy.to('cuda')
```

Figure 9 Modifying output layer from 7 to 3.

Through trial and error, an observation was made that when the output layer gets reinitialised, the models were able to produce better results. This is because an agent usually experience overfitting to their specific environments, therefore reinitialising the output weights allow the model to increase randomness in their output actions while keeping most of its knowledge.

3. *Hyper-Parameters*

To determine hyper-parameters, Optuna was utilised to aid in helping find the optimal parameters for the pre-trained model on a new environment. But in most cases, using the same parameters as the ones used to train the baseline models gave the best results.

D. Logging Call-back

In order to understand and keep record of RL models, a log keeper is needed. The class ***TrainAndLoggingCallback,*** accredited to [9], keeps a log of all the models, and produces a graph in real-time on tensorboard. Furthermore, it also saves a copy of a model every 10000 timesteps.

E. Model Evaluation

To evaluate a model, the built-in function, ***evaluate_policy,*** is used to find the mean reward score and standard deviation of 50 rounds of gameplay. This number was chosen because the minimum acceptable size of a sample is 30 **[20]**. Due to the need for thorough testing of many models, 50 trails were the most reasonable within the given time constraint.

V. Challenges & Addition to Environment

Throughout the project, 3 major challenges were faced, firstly hyperparameter tuning, secondly transfer learning from more complex environment to simpler environment, finally reward shaping on deadly_corridor scenario.

1. *Tuning Hyperparameter*

The game doom is quite a complex game; thus, an agent requires a large amount of data to properly learn and play the game. Consequently, a model has to be trained for a very long time, with the longest ones taking almost up to six hours. This makes tuning hyperparameters extremely time-consuming and a lot of time was taken for trial and error and research on each parameter of the PPO algorithm. Furthermore, since the minimum requirement set for each model is to be able to complete its task, this caused a lot of stress to train. Especially the deadly_corridor scenario where the trainings were extremely unstable and often crashes to a negative reward score. Additionally, it took a while for me to realise a pre-trained model's parameters cannot be changed manually. It has to be loaded alongside with custom_objects that contains the desired parameters.

2. *Complex-to-Simple Transfer Learning*

For context, simple to complex transfer learning was relatively easy to implement. My hypothesis being, it is simpler to add output actions and initialise them as zero compared to removing output nodes that already has complex interconnections. This challenge took the longest time to figure out because there were virtually zero sources on this topic. Even if the output layers and action spaces of the complex model was changed from 7 to 3. The code would still output an error regarding the output size being wrong. I took tremendous amount of trail and error until I figured that the parameter n_steps and batch_size must be kept constant in order for transfer learning to be possible. After figuring this out, all the trained models had to be re-trained using a standardised **n_steps**: 4096 and **batch_size**: 64. Unluckily, due to the change in parameters the models were not able to produce an equally good results, therefore they had to be manually tuned once again, which took extremely long.

3. *Reward Shaping on Deadly Corridor*

Training an agent to effectively kill all the monsters and get the armour at the end of the corridor was strenuous. Both reward shaping and hyper-parameters tuning is necessary to make it robust. Sometimes changing the reward system has very unpredictable consequences given the complexity of the environment. If too much reward is given to killing a monster, the agent will just kill one monster and pause. If too much reward is given to movement, then the player will just sprint towards the armour and die in the process. Finding that sweet spot in the reward system is the hardest part of training and extremely time-consuming because the hyper-parameters also have to be tuned at the same time. An observation made is when the reward received is divided by 100, the model was able to perform better, the reason is unknown.

4. *Addition*

The addition I have contributed to this environment is wrapping the doom environment into OpenAi Gym. Secondly, reward shaping and hyperparameter tuning on the models. Thirdly, utilising curriculum training to train a difficult environment. Fourthly, I found that reinitialising the output layers of a pre-trained model will give better results. Lastly, implementing transfer learning from environments with different action spaces and also being able to change their parameters with custom_objects. As far as I know, no one else on the internet or any papers has attempted this or explored this using stable-baseline3.

VI. Experimental Methods

The first experiment is to see whether reinitialising a pre-trained model's output layer produces better results or not. To conduct this experiment, the basic model is used to learn defend_the_line and predict_position. One training will happen with the output layer reinitialised, the other training their output layer will not be reinitialised. Parameters and timesteps are kept constant and the mean reward of each model will be used for evaluating. To ensure that the direction of transfer learning has no impact on the effect of reinitialisation, we will also test the deadly model learning to play defend_the_line.

Secondly, the basic-to-deadly and deadly-to-basic model is used to learn Defend_the_Center, Defend_the_Line and Predict_Postion. The number of timesteps used and hyperparameters are kept constant to their corresponding baseline models. The only difference is the output layers of the pre-trained models are reinitialised. The models are then evaluated by finding the mean reward score of 50 episodes and their standard deviation.

To further explore the impact of transfer learning direction on the effectiveness of transfer learning. The basic and deadly model is also used to learn Defend_the_Center, Defend_the_Line and Predict_Postion. This allows us to have more data to compare more thoroughly.

VII. Result & Discussion

Firstly, the results are very clear that reinitialising the output layer was able to make the models produce significantly better scores. This happens because models tend to overfit their environments, therefore reinitialisation is needed to randomise the actions for the agent to effectively explore new environments. But as the deadly_corridor model fails to learn any of the environments, reinitialising the output model did not have any significant impact, see **Appendix 4** for more graphs.

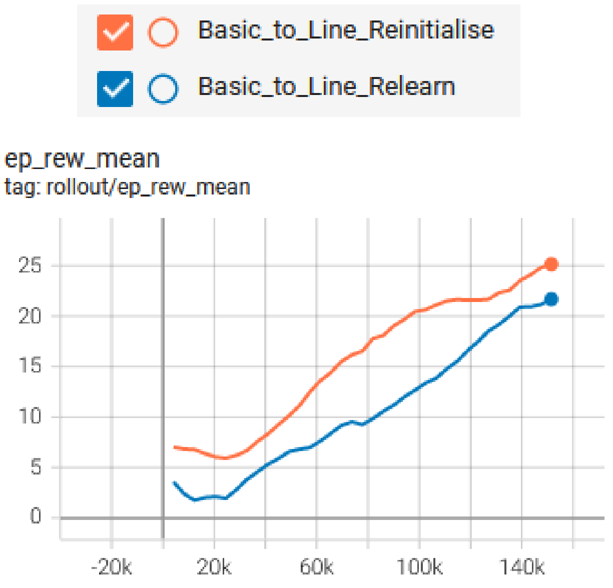


Figure 10 Comparing reinitialising and relearning basic to line model.

Both basic-to-deadly and deadly-to-basic models were not able to generalise well, thus failing to successfully learn any of the environments. Many attempts were made to tune their hyper-parameters in hopes of improving the model but to no avail. Though basic-to-deadly were able to learn much more effectively than deadly-to-basic in certain environments, it still was not as good as transferring from basic directly to the environments.

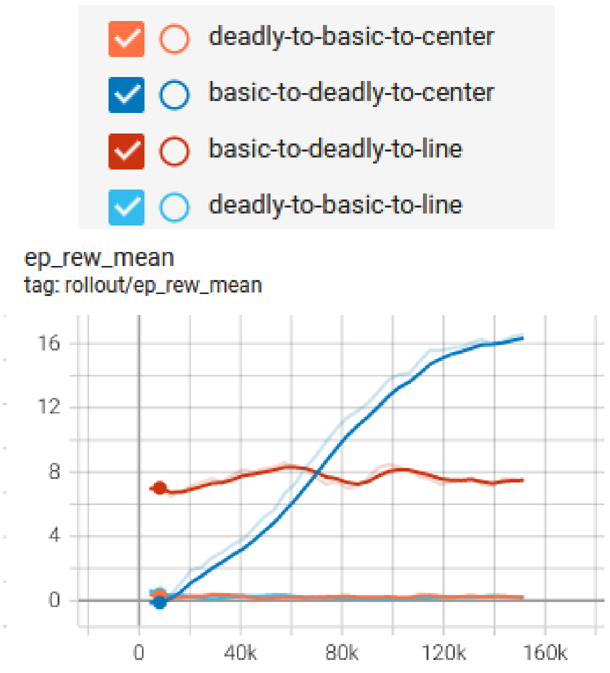


Figure 11 basic-to-deadly model is significantly better at generalisation then deadly-to-basic model.

An explanation for this is a complex environment is not able to transfer learning effectively to simpler environment. The knowledge learned in a complex environment is very specific and may not be applicable to simpler environment. Complex environments usually require advanced techniques that are not relevant to simpler environments, causing detriment to performance. This is proven by the fact that the deadly_corridor model failed to learn effectively on any of the simpler environments. While the basic model is able to complete all of the environments. To summarise, when the basic model learned to play deadly, the model became specialised in playing deadly_corridor. Additionally, the deadly model is not able to transfer learning to any simpler environment anyways, therefore explaining why both basic-to-deadly and deadly-to-basic was not able to generalise well.

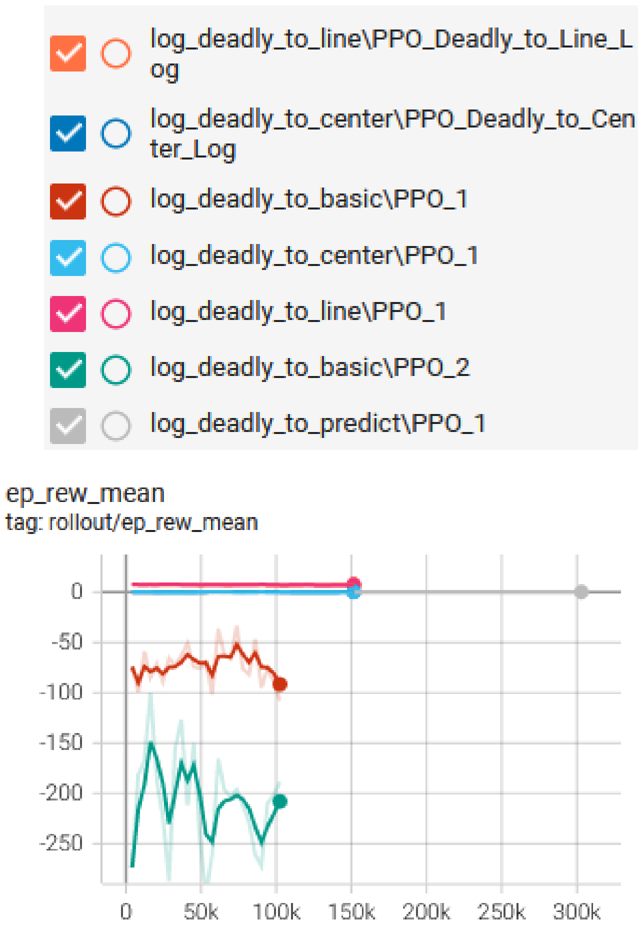


Figure 12 deadly model fails to learn all other environments.

On the other hand, the basic model was able to learn center, line and predict extremely well. It was able to produce better scores than the model specifically trained for those environments. This is most likely due to the fact that the basic environment taught the agent a very broad and useful skill, which is detecting monsters and shoot it. The full list of mean reward of each model can be found in **Appendix 3**.

VIII. Conclusion

The answer to the question is simple-to-complex models are more generalised than complex-to-simple. But the whole point of transfer learning is to reduce training time by using existing knowledge, in hopes of producing a decent model in a shorter amount of time. But that is not the case here, even though the simple-to-complex model was able to play certain environments pretty well. It does not come close to models that were specifically trained for that environment.

Though it is not a significant difference, we cannot deny the effectiveness of transfer learning, as our basic model was able to transfer its learning over to other environments, please refer to **Appendix 5**. Resulting in an increase in performance in a shorter amount of time. A limitation still remains, we are not certain of how much the similarity of environments affect our models. Consequently, to decide whether to do transfer learning or training a model from scratch will still be difficult. As transfer learning in deep learning lacks interpretability, it leaves us with no information on what knowledge is exactly being transferred.

In future work, I will try to better understand how each layer in the network affects transfer learning. This can be done by reinitialising different layers and see the effects it has on the result. This will provide insight to what each layer is responsible for learning, perhaps increasing the interpretability of transfer learning. If given more time, I would have made some new Doom maps, with various action spaces. This will help create a more thorough and fairer experiment. Different combinations of complexities can be tested, which will make the results more reliable.

References

[1] *A comprehensive survey on Transfer Learning - arXiv* (no date). Available at: https://arxiv.org/pdf/1911.02685.pdf (Accessed: May 8, 2023).

[2] openai.com. (n.d.). *OpenAI Five defeats Dota 2 world champions*. [online] Available at: https://openai.com/research/openai-five-defeats-dota-2-world-champions.

[3] Sopov, V. and Makarov, I. (n.d.). *Reward Shaping for Deep Reinforcement Learning in VizDoom*. [online] Available at: https://ceur-ws.org/Vol-3094/paper_17.pdf [Accessed 17 Apr. 2023]. Deepmind (n.d.). *AlphaGo*. [online] www.deepmind.com. Available at: https://www.deepmind.com/research/highlighted-research/alphago.

[4] Asawa, C., Elamri, C. and Pan, D. (n.d.). *Using Transfer Learning Between Games to Improve Deep Reinforcement Learning Performance and Stability*. [online] Available at: https://web.stanford.edu/class/cs234/CS234Win2020/past_projects/2017/2017_Asawa_Elamri_Pan_Transfer_Learning_Paper.pdf [Accessed 17 Apr. 2023].

[5] GitHub. (2023). *Farama-Foundation/ViZDoom*. [online] Available at: https://github.com/Farama-Foundation/ViZDoom.

[6] *(PDF) a study on overfitting in deep reinforcement learning - ResearchGate* (no date). Available at: https://www.researchgate.net/publication/324643719_A_Study_on_Overfitting_in_Deep_Reinforcement_Learning (Accessed: May 8, 2023).

[7] stable-baselines3.readthedocs.io. (n.d.). *Stable-Baselines3 Docs - Reliable Reinforcement Learning Implementations — Stable Baselines3 1.2.0a2 documentation*. [online] Available at: https://stable-baselines3.readthedocs.io/en/master/.

[8] Zhu, Z., Lin, K. and Zhou, J. (n.d.). *Transfer Learning in Deep Reinforcement Learning: A Survey*. [online] Available at: https://arxiv.org/pdf/2009.07888.pdf.

[9] www.youtube.com. (n.d.). *Reinforcement Learning for Gaming | Full Python Course in 9 Hours*. [online] Available at: https://www.youtube.com/watch?v=dWmJ5CXSKdw&t=5082s [Accessed 17 Apr. 2023].

[10] www.youtube.com. (n.d.). *Reinforcement Learning in 3 Hours | Full Course using Python.* [online] Available at: https://www.youtube.com/watch?v=Mut_u40Sqz4 [Accessed 17 Apr. 2023].

[11] www.youtube.com. (n.d.). *Deep Reinforcement Learning for Atari Games Python Tutorial | AI Plays Space Invaders*. [online] Available at: https://www.youtube.com/watch?v=hCeJeq8U0lo&t=1931s [Accessed 17 Apr. 2023].

[12] Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Openai, O. (2017). *Proximal Policy Optimization Algorithms*. [online] Available at: https://arxiv.org/pdf/1707.06347.pdf.

[13] Csefalvay, C.von (2019) *Transfer learning: The DOS and don'ts, Medium*. Starschema Blog. Available at: https://medium.com/starschema-blog/transfer-learning-the-dos-and-donts-165729d66625 (Accessed: May 8, 2023).

[14] Raval, S. (2023). *Coding Challenge*. [online] GitHub. Available at: https://github.com/llSourcell/Unity_ML_Agents/blob/master/docs/best-practices-ppo.md [Accessed 17 Apr. 2023].

[15] Cf.ac.uk. (2023). Available at: https://pats.cs.cf.ac.uk/@archive_file?p=2382&n=final&f=1-report.pdf&SIG=e2bcbaeccb196d7aec4a355cace2e10515c2b31856f0fc717a3efca43304e4f2 [Accessed 17 Apr. 2023].

[16] GitHub. (2022). *openai/gym*. [online] Available at: https://github.com/openai/gym.

[17] Harder, H. de (2022). *Techniques to Improve the Performance of a DQN Agent*. [online] Medium. Available at: https://towardsdatascience.com/techniques-to-improve-the-performance-of-a-dqn-agent-29da8a7a0a7e.

[18] Sopov, V. and Makarov, I. (n.d.). *Reward Shaping for Deep Reinforcement Learning in VizDoom*. [online] Available at: https://ceur-ws.org/Vol-3094/paper_17.pdf [Accessed 17 Apr. 2023].

[19] Preferred Networks Research & Development. (2018). *Optuna: An Automatic Hyperparameter Optimization Framework*. [online] Available at: https://tech.preferred.jp/en/blog/optuna-release/#:~:text=Optuna%20uses%20a%20history%20record [Accessed 8 May 2023].

[20] Ganti, A. (2023). *What Is the Central Limit Theorem (CLT)?* [online] Investopedia. Available at: https://www.investopedia.com/terms/c/central_limit_theorem.asp#:~:text=A%20sample%20size%20of%2030.

[21] Khandelwal, R. (2023) "Proximal Policy Optimization (PPO): Exploring the Algorithm Behind ChatGPT's Powerful Reinforcement Learning Capabilities," *Medium*, 2 March. Available at: https://arshren.medium.com/proximal-policy-optimiation-ppo-exploring-the-algorithm-behind-chatgpts-powerful-1634f6c3c768.

APPENDIX

1. Reward System

Environment	Reward
Basic Scenario	+101 for killing monster. -5 for missing -1 for living
Deadly Corridor Scenario	-100 for dying. +dX for getting closer to armour. -dX for getting further from armour. +1000 for collecting armour. -21 for damage taken. +210 for hitting a monster.
Defend the Centre Scenario	+1 for killing a monster.
Defend the Line Scenario	+1 for killing a monster.
Predict Position Scenario	-0.0001 for living +1 for killing the monster

2.Hyper-parameters Settings

Environment	Hyper-parameters
Basic Scenario	learning_rate: 0.00015 n_steps: 4096 ent_coef: 0.0 gamma: 0.99 gae_lambda: 0.95 time_steps: 100000
Deadly Corridor Scenario	learning_rate: 0.0001 n_steps: 4096 ent_coef: 0.0001 gamma: 0.95 gae_lambda: 1 time_steps: 300000 (on difficulty 1) time_steps: 100000 (for difficulty 2-5)
Defend the Centre Scenario	learning_rate: 0.0001 n_steps: 4096 ent_coef: 0.00001 gamma: 0.99 gae_lambda: 0.95 time_steps: 150000

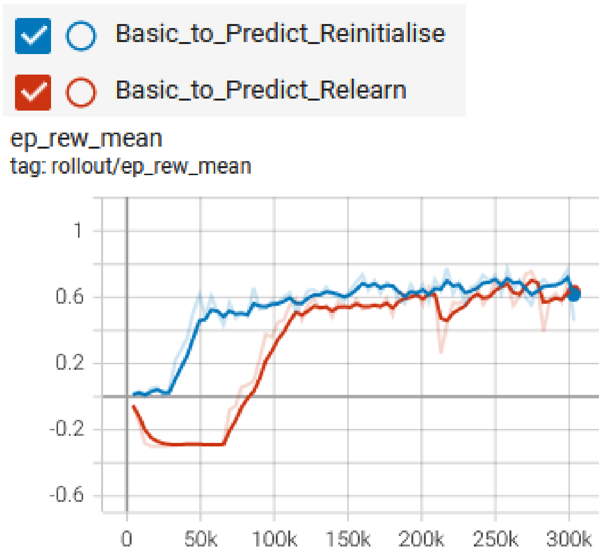
Defend the Line Scenario	learning_rate: 0.0001 n_steps: 4096 ent_coef: 0.00001 gamma: 0.99 gae_lambda: 0.95 time_steps: 150000
Predict Position Scenario	learning_rate: 0.00015 n_steps: 4096 ent_coef: 0.0 gamma: 0.99 gae_lambda: 0.95 time_steps: 300000

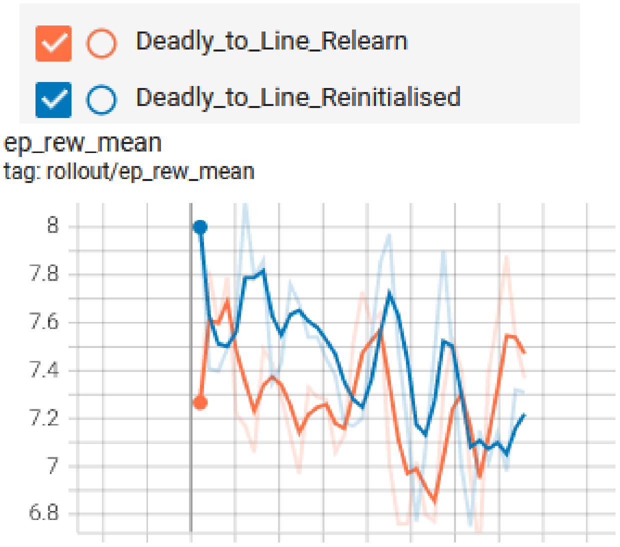
3.Mean Reward and Standard Deviation

Environment	**Mean Reward and Standard Deviation**
Basic	Mean reward: 88.28. Std: 8.483
Deadly to Basic	Mean reward: -300. Std: 0.0
Line	Mean reward: 26.4. Std: 5.115
Deadly to Line	Mean reward: 1.68. Std: 1.406
Basic to Line	Mean reward: 29.74. Std: 6.788
Deadly to Basic to Line	Mean reward: 1.88. Std: 1.862
Basic to Deadly to Line	Mean reward: 2.0. Std: 2.010
Center	Mean reward: 15.26. Std: 2.110
Deadly to Center	Mean reward: -1.0. Std: 0.0
Basic to Center	Mean reward: 20.04. Std: 1.562
Basic to Deadly to Center	Mean reward: 17.00. Std: 2.2
Deadly to Basic to Center	Mean reward: -1.0. Std: 0.0
Predict	Mean reward: 0.77. Std: 0.394
Deadly to Predict	Mean reward: -0.3. Std: 0.0
Basic to Predict	Mean reward: 0.797. Std: 0.367
Deadly to Basic to Predict	Mean reward: -0.3. Std: 0.0
Basic to Deadly to Predict	Mean reward: -0.3. Std: 0.0
Deadly	Mean reward: 13.60. Std: 8.563
Basic to Deadly	Mean reward: 8.81. Std: 11.614

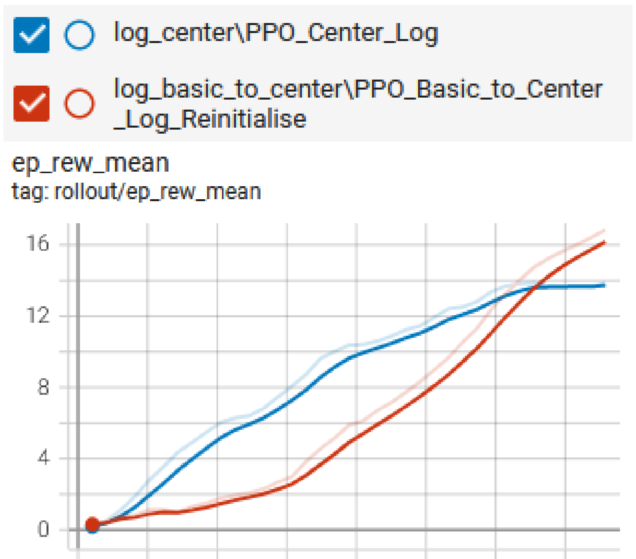
Jirat Manpadungkit
Student ID: 20320731

4. Reinitialising and Relearning Comparison





5.Basic Model successfully transfer learning.



Basic model beating a model that was specifically trained for defending the center scenario.