Component Transfer Learning for Deep RL Based on Abstract Representations

Anonymous Author(s)
Affiliation
Address
email

Abstract

In this work we investigate a specific transfer learning approach for deep reinforcement learning in the context where the internal dynamics between two tasks are the same but the visual representations differ. We learn a low-dimensional encoding of the environment, meant to capture summarizing abstractions, from which the internal dynamics and value functions are learned. Transfer is then obtained by freezing the learned internal dynamics and value functions, thus reusing the shared low-dimensional embedding space. When retraining the encoder for transfer, we make several observations: (i) in some cases, there are local minima that have small losses but a mismatching embedding space, resulting in poor task performance and (ii) in the absence of local minima, the output of the encoder converges in our experiments to the same embedding space, which leads to a fast and efficient transfer as compared to learning from scratch. The local minima are caused by the reduced degree of freedom of the optimization process caused by the frozen models. We also find that the transfer performance is heavily reliant on the base model; some base models often result in a successful transfer, whereas other base models often result in a failing transfer.

1 Introduction

State-of-the-art deep reinforcement learning agents are able to beat the best human players in complex games such as Dota 2 [1] and StarCraft II [2]. However training such agents requires many resources and a long training time. For instance the Dota2 agent has been trained using 770±50 PFlops/s·days of compute for 10 months. Therefore transfer learning yields an interesting avenue: training a model once to then use it for multiple purposes. This is not straightforward because a trained agent usually needs to relearn everything from scratch, without gain in performance, when facing an environment with minor visual perturbations such as changing the color of a couple of pixels [3]. A popular approach to this problem is fine-tuning (see Figure 1a), however some works argue that fine-tuning performs worse than learning from scratch [4, 3, 5, 6].

In this work, we leverage the modularity brought by RL techniques based on an abstract representation that is used for both model-free and model-based Deep Reinforcement Learning algorithms (e.g. [7, 8, 9, 10]). In that context, only some components are fine-tuned for the new target task. In particular, we consider the visual transfer problems where only the representation of the state space changes such that the encoder has to be fine-tuned and all other components can be frozen (illustrated in Figure 1b). We will investigate whether this approach suffers from the same poor fine-tuning performance as reported by the literature. We hypothesize that the frozen models will expect a certain input, and thus enforce an embedding space. More specifically, it will enforce the same embedding space that the agent has found in the source task. We further hypothesize that this method might result in a fast transfer, because 1) only the encoder model weights have to be retrained, and 2) the

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

gradients flowing from the frozen models directly point to an already found global optimum which can help in speeding up the training process.

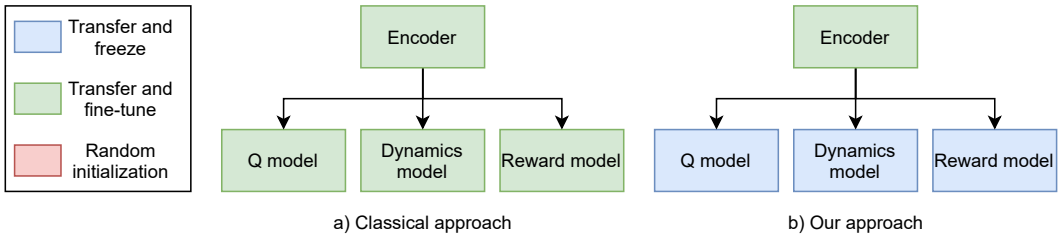

Figure 1: Classical fine-tuning approach (a): transfer and fine-tune all models on target task. In comparison, our approach (b) only fine-tunes the encoder by transferring and partially freezing components of model-based RL agent. Note the shared embedding space.

Our main contributions are the following:

- In the context of an RL agent that learns an abstract representation suited for both planning and a value function evaluation, we explore a specific transfer learning procedure. The experimental[1] results reveal that it works well in many cases but that it can lead into a local minimum other cases.
- A study of the local-minimum phenomena suggest that it is caused by 1) small losses area in the optimization landscape at the place of the local minima, and 2) that it gets stuck due to the reduced degree of freedom of the optimization process caused by the frozen models.

2 Related Work

In the field of transfer learning the most adopted and straightforward approach is fine-tuning. Fine-tuning approaches train an agent on a source task, and then the weights of a number of k layers are used to initialize the weights of a different agent that will be trained on the target task. Besides determining the amount of layers to transfer it can also be decided to freeze the transferred layers. Typically fine-tuning approaches freeze early layers of the convolution network and use a smaller learning rate to fine-tune the rest of the network.

A couple of papers investigated the fine-tuning performance of deep reinforcement learning agents, such as by experimenting with fine-tuning a different amount of layers [6]. They find that this approach scores better than learning from scratch, suggesting that deep neural networks are able to transfer knowledge. They do not find a major difference in the performance between the amount of layers transferred. A similar study [5] finds that fine-tuning all the convolution layers and resetting the fully connected layers has a slightly better performance than learning from scratch. Two similar studies [4, 3] reports different findings for experiments in which they freeze the transferred convolution layers. The main findings are 1) freezing layers performs worse than fine-tuning due to negative transfer, 2) learning from scratch generally outperforms the fine-tuning approach and it is more stable, 3) however, fine-tuning can sometimes reach the same or slightly better performance. This suggests that our approach might also deal with negative transfer due to the frozen layers.

Finally a recent study [11] proposed to 1) transfer fractions of the weights and 2) similarly to our work, transfer components of a learned latent dynamics (Dreamer[9]). So instead of choosing to transfer or not transfer certain model layers, they introduce a third option which is to fractional transfer the weights of these layers by multiplying the weights with a fraction. In their experiment they consider a transfer to a target task that has a different reward and action space. And thus they choose to 1) fully transfer the encoder, decoder and RSSM weights, 2) partially transfer action model and 3) fractionally transfer the last layer of the reward and value model. The main difference compared to our approach (besides the fractional transfer) is that we freeze all components except the encoder to ideally recover the same embedding space.

[1] Source code and plot data available at (anonymised, see attached zip file)

3 Background

Reinforcement learning We define our environment as a Block MDP [12] defined by the tuple $\langle \mathcal{S}, \mathcal{A}, X, p, q, R \rangle$, with a finite unobservable/hidden state space $\mathcal{S}$, finite action space $\mathcal{A}$, finite observation space X, latent transition function $p(s'|s,a)$ for $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}$, emission function q which provides the observation given a latent state $q(h|s)$ for $h \in X$ and $s \in \mathcal{S}$, and finally the reward function $R(s,a,s')$. Then the source and target task have very similar MDP's: M_s and M_t, respectively. Only with different observation space $X_s \neq X_t$, and emission function $q_s \neq q_t$. From any state s, the objective is to maximize the expected sum of discounted rewards: $\mathbb{E}[\sum_{k=0}^{\infty} \gamma^k R(s_{t+k}, a_{t+k}, s'_{t+k+1}) \mid s_t = s]$

Transfer learning We define transfer learning [13] as using the source task M_s to provide prior knowledge, in our case represented by the transferred model (parameters), with the objective of learning the task in target domain M_t better compared to not using prior knowledge.

4 Method

Model based methods learn an internal dynamics function to predict what the reward and next (internal) state will be, given a state and action. These models often have different losses to train on than the value function. When the encoder is shared and exposed to all losses, the shared embedding space captures features relevant to estimating the value function and the internal model (reward included). Thus we consider model-based DRL agents that utilize a shared encoder prior to the internal dynamics and value function. Meaning that our transfer learning method is applicable to algorithms such as DreamerV2 [9] and MuZero [10]. Furthermore it could also be applied to supervised or unsupervised deep learning problems for time series.

4.1 Component Transfer Learning

Given a fully trained model-based DRL agent on a source task M_s (base model), we have access to a solution of the encoder model weights (which outputs an embedding space x) and dynamic model weights, for input space X_s and transition function p that have small losses. Meaning that the dynamics model is able to estimate correctly the environment transitions given the embedding space x. When considering a new MDP with a different emission function, the loss and the gradients will be big again. If we freeze the weights of the dynamics model while retraining only the encoder, one strong solution would be that the encoder learns to map the observations (of the new MDP with the new emission function) to exactly the same representation as previously. Freezing a good internal dynamics model could thus enforce a certain abstract representation for any MDPs, even with different observations (a different emission function).

4.2 Description of the different losses

We combine model-free and model-based, which provides the advantage of both approaches; planning and estimation of the cumulative reward. Both are trained on top of a shared low-dimensional embedding space to capture task reward (model-free) and transition (model-based) related information in the same encoding space, as depicted in Figure 2. Even if the environment has no rewards, the agent is able to create a meaningful embedding space because of a specific loss that prevents the collapse of the representation.

Notation We define the embedding/abstract state space as $x \in \mathcal{X}$ where $\mathcal{X} = \mathbb{R}^{n_m}$ and $n_m \in \mathbb{N}$ is the dimension of the continuous embedding space. Furthermore it defines the following functions: 1) the encoder $\hat{e} : X \to \mathcal{X}$ is parameterized by $\theta_{\hat{e}}$ which maps the observable state X to the abstract state x 2) the internal transition function $\hat{p} : \mathcal{X} \times A \to \mathcal{X}$ is paramterized by $\theta\hat{p}$, 3) the reward function $\hat{r} : \mathcal{X} \times A \to [0, 1)$ is parameterized by $\theta_{\hat{r}}$, and 4) the discount function $\hat{\gamma} : \mathcal{X} \times A \to [0, 1)$ parameterized by $\theta_{\hat{\gamma}}$ which is used to predict terminal states.

We use an off-policy learning algorithm that samples transition tuples (s, a, r, γ, s') from a replay buffer. Given this tuple the current and next state are encoded using the encoder to generate the embeddings $x \leftarrow \hat{e}(s; \theta_{\hat{e}}), x' \leftarrow \hat{e}(s'; \theta_{\hat{e}})$. The Q model is trained using the DDQN algorithm[14]

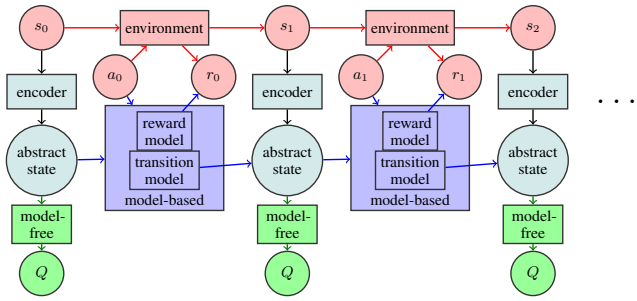

Figure 2: Visualisation of the CRAR architecture [8]. The red components indicate environment attributes. The model-free components are depicted with green, and the model-based with blue. The encoder and thus the abstract state are shared between the two, and depicted with cyan.

with the target value:

$$Y = r + \gamma Q(\hat{e}(s'; \theta_{\hat{e_-}}), \underset{a' \in A}{\operatorname{argmax}}\, Q(x', a'; \theta_Q); \theta_{Q-})$$

Where θ_{Q-} and θ_{e-} are the buffered Q and encoder models. Then the agent minimizes the loss:

$$L_Q(\theta_Q, \theta_{\hat{e}}) = (Q(x, a; \theta_Q) - Y)^2$$

Similar to the CRAR[8] agent, the reward, discount and transition models are trained using:

$$L_R(\theta_{\hat{e}}, \theta_{\hat{p}}) = |r - \hat{r}(x, a; \theta_{\hat{r}})|^2, L_\gamma(\theta_{\hat{e}}, \theta_{\hat{\gamma}}) = |\gamma - \hat{\gamma}(x, a; \theta_{\hat{\gamma}})|^2, L_p(\theta_{\hat{e}}, \theta_{\hat{p}}) = ||[x + \hat{p}(x, a; \theta_{\hat{p}})] - x'||_2^2$$

Finally we have three embedding structure enforcing losses:

$$L_{d1}(\theta_{\hat{e}}) = exp(-C_d||e(s_1; \theta_{\hat{e}}) - e(s_2; \theta_{\hat{e}})||_2)), L_{d2}(\theta_{\hat{e}}) = max(||e(s_1; \theta_{\hat{e}}||_\infty^2) - 1, 0)$$

From the L_{d1} loss (which forces that two random states have as different embedding values as possible) we introduce the third structure enforcing loss: L'_{d1} which is the same but it samples only successive states. The L_{d2} loss is essentially forcing the embedding space to be in a radius of 1.

5 Experiments/Results

The section present two experiments: an experiment with a simple maze environment (Figure 3) and a distribution of mazes (Figure 4).

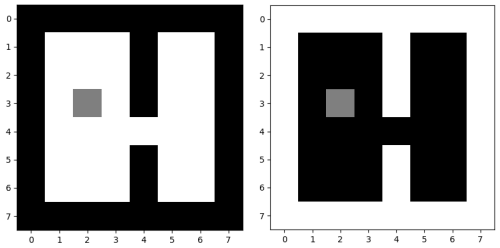

Figure 3: Visualisation of the low-dimensional environment state space, on the right the inverse variant is displayed.

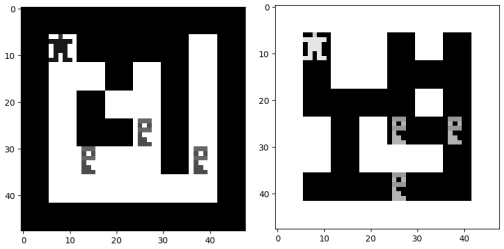

Figure 4: Visualisation of the high dimensional environment state space. Note the random generation of wall, path and reward positions.

5.1 Simple maze

In this experiment, we consider a simple maze environment without rewards. The environment consists of 8 by 8 tiles, see Figure 3. Each tile is either a wall, path or the position of the agent. These are represented by a value between 1 and -1. The agent has four possible actions, namely to move in any orthogonal direction. For this environment, the agent is given an embedding space of two

continuous variables (two neurons) which allows for simple and direct visualisation and interpretation of the learned embedding space.

The agent is first trained on the regular environment. When it has converged, we use our proposed approach to transfer the learning to the target domain. Meaning that the models weights are used to initialise the models of a second CRAR agent. Then all models are frozen, except for the encoder that is fine-tuned in the target domain. In the transfer phase, the agent is trained on the same maze but with the inverse visuals (target domain), see Figure 3.

5.1.1 Training the base model

We use the default parameters (learning rate = 5×10^{-4}) to train the base model. From Figure 5 we can observe the convergence process of the embedding space. We observe that the left side of the maze is mapped to the top side ($x_2 > 0$) of the embedding space, we thus expect the same in the fine-tuned/transferred embedding space.

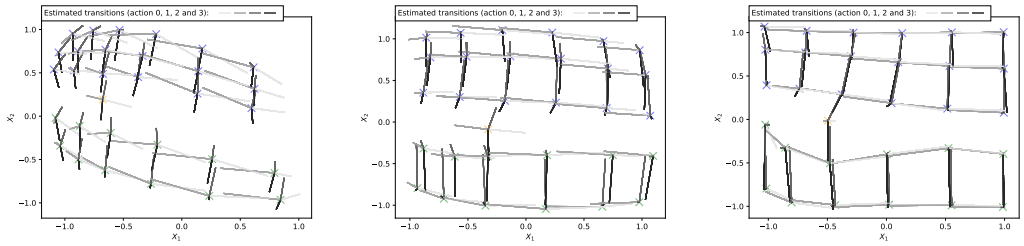

Figure 5: Visualisation of the convergence of the embedding space of a CRAR agent during the training of the simple low-dimensional maze. The crosses indicate the embedded value of a certain state, blue crosses indicate a state of the left side of the maze, and green ones indicate the right side. The lines indicate the corresponding (embedded) transition given an action and a state. From left to right: 5000, 20000 and 160000 (used as base model 1) steps.

5.1.2 Transferring

After the base model has been trained, we have executed 30 transfers by fine-tuning the encoder model on the inverse environment (learning rate = 1×10^{-4}). In this simple environment without reward, we can qualitatively asses the performance of the agent by visually inspecting whether the converged embedding space is the same for the base model as for the fine-tuned model. We find that most transfers successfully converge to the same embedding space, but some fail to converge.

Successful transfer From Figure 6 we can observe that the converged embedding space (step 80000) looks identical to the converged embedding space of the base model (see Figure 5). This result looks promising because it appears that the embedding space of the fine-tuning agent converges efficiently to the same representation obtained previously.

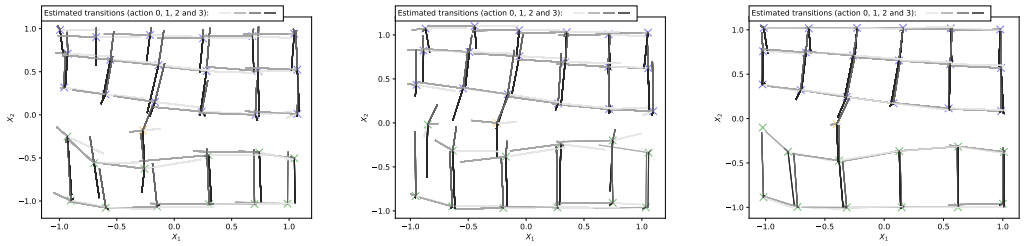

Figure 6: Visualisation of the successful convergence of the embedding space of a CRAR agent during the fine tuning on the inverse low-dimensional maze (base model 1). From left to right: 5000, 20000 and 80000 steps

Failing transfer Interestingly, we also observe that not all transfers are successful. We trained two CRAR agents on the original environment and used those as base models. For base model one, 19 out of 22 transfers failed (90%), and for base model two only 1 out of 22 transfers failed ($\sim 5\%$). This might indicate that the quality of the transfer is determined by the frozen models (p-value < 0.0001).

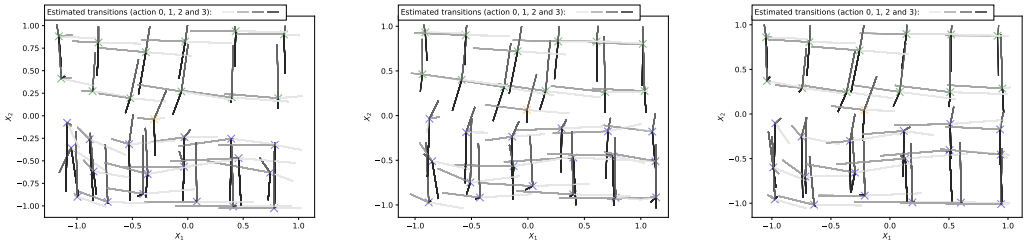

Figure 7: Visualisation of the failing convergence of the embedding space of a CRAR agent during the fine tuning on the inverse low-dimensional maze (base model 1). From left to right: 5000, 20000 and 80000 steps

From the visualisations of the embedding space of a failed transfer (Figure 7), we can observe that indeed the agent is not able to converge to the previous found optimum. Furthermore, the structure of the embedding space looks similar to the base model but, if we look at the colors of the crosses, we see that the embedding space has flipped. In this experiment, most of the failing transfers converge to this same embedding space (flipped sides).

5.1.3 Interpretation

We argue that the convergence to the switched mapping is no coincidence. The main argument we make is that by taking the inverse of the environment, it immediately gets caught in quite a “good" local optimum in terms of the losses but with a very different representation. An in-depth study of this can be found in Appendix A.

The previous argument explains why the failed transfers specifically end up in the swapped-embedding local minimum previously visualised. However it only partially explains why it gets stuck in this local minimum (small losses). We argue that the frozen models restrict the degree of freedom in the optimization process. Specifically, we argue that freezing the last layers with respect to the different losses (instead of the encoder losses) changes the neural network degrees of freedom in the optimization process. This in combination with the existence of area with relatively small losses area in the optimization landscape that leads the agent to getting stuck (the maximum entropy losses might also make things worse). See appendix B for a more detailed explanation.

5.2 High-Dimensional Meta-Labyrinth with rewards

We now consider a distribution of mazes where each element (wall, agent, reward) are represented as blocks of 6 by 6 pixels, which makes the total input space 48×48 pixels. In this case, the agent uses an embedding space of 8x8x3 continuous variables and uses convolutional layers in the encoder. Each maze of the distribution of mazes contains initially three rewards represented by the keys (see Figure 4). After collecting the three rewards (or a maximum of 50 steps), the environment generates a new maze in which the positions of the rewards, walls and paths change. Each action that yields to a key leads to a reward of $+1$ and otherwise a reward of -0.1. Again we have a normal variant and an inverse variant which will be used as the visual transfer scenario.

Using this environment we can quantify the performance of a trained model. In this way we will test 1) whether the transfer is successful, 2) what base model has the best transfers, 3) whether the transfer is faster than the training of the base model, 4) compare performance between approaches (e.g. learning from scratch, our approach, some variants of our approach such as resetting the encoder) 5) hyperparameter tuning (learning rate). See Appendix for the latter.

5.2.1 Impact of base models on transfer performance

First we trained six base models (basemodel1,...,basemodel6). For each base model, we ran six transfers (using six different seeds) by fine-tuning the encoder on the inverse variant of the environment. This results in a total of 36 transferred agents. The base models will be trained for 200 epochs, and the transfers for 100 epochs. Lastly the learning rate used for the fine-tuning is the same as the training of the base model: 5×10^{-4}.

In Figure 8 the performance of the transfers can be observed. From the left image we can make some quick conclusions, 1) the variance in the performance is big among different transfers with different base models, but can also be big among the same base model (see performance of group basemodel3), 2) some base models do not have a single succeeding transfer (basemodel4 and basemodel5). 3) some base models appear to always have a successful transfer (basemodel1 and basemodel2). Following these findings we can confirm the same finding as before: the base model that will be fine-tuned on determines the transfer success rate. Meaning that the frozen models determine whether the flowing gradients steer into a local minimum or not. From the image on the right we find what we expect: approximately the same average and standard deviation for the different seeds.

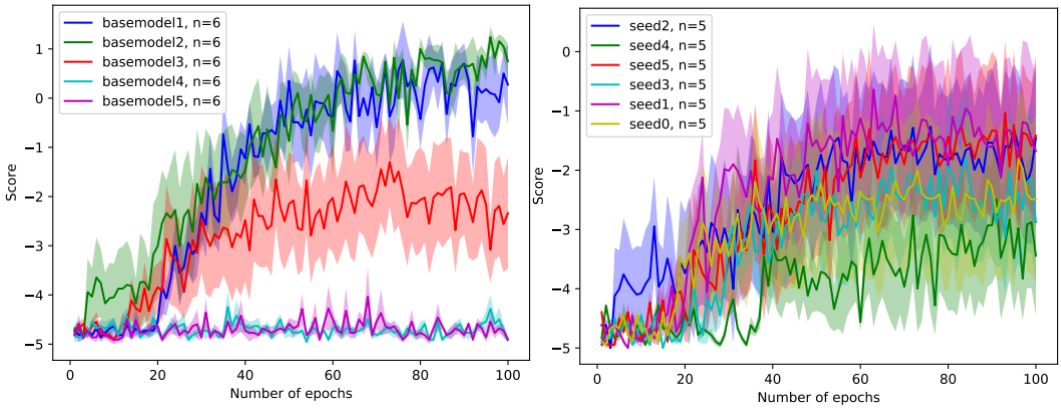

Figure 8: Average and standard error of the performance of the transfers, six transfers per base model (first base model is called basemodel1, etc..). Both plots are generated from the same data but show the average and standard deviation of different groupings, the left plot has the data grouped by the base model, and the right plot has the data grouped by the seed number of the fine tuning process.

5.2.2 Study of high-dimensional embeddings

In this subsection we will look into the difference between the embeddings of successful and failing transfers. This is done 1) quantitatively by calculating the L2 distance between the base model and the transfer and 2) qualitatively by visually inspecting the embedding space. The L2 distance is measured by taking an instance of the environment, and having this same instance rendered in the normal visual variant and the inverse variant. Then the normal visual variant is fed to a base model, and the inverse variant is fed to one of its transferred models. Then the L2 distance is calculated between the embedding of the base model and the embedding of the transferred model.

Measuring the L2 distance reveals that the distance between the transfers and the base model is bimodal, with successful transfers that have a loss around 1.1 while it is around 6.2 for the failing transfers. The embedding space of the successful transfer converge close to the embedding space of the base model. This partially confirms our hypothesis: the gradients flowing from the frozen models can stimulate the encoder to converge to the same output space (embeddings) for visually different inputs but the risk of local minima is also confirmed. An overview of the visualisations of the embeddings of the five base models with an associated transfer is given in Figure 9. We can for instance observe that 1) the embedding of the good base model (basemodel1) is the same as the transfer, and 2) the embedding of the poor base model(basemodel5) has a mismatching embedding with its transfer.

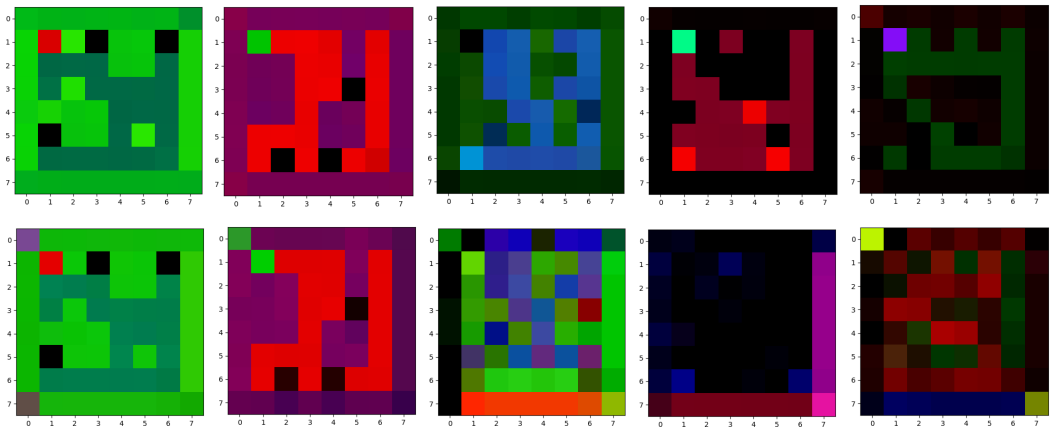

Figure 9: Upper row are embeddings of base models, lower row are embeddings of fine-tuned models (on the inverse environment instance). Each column indicates a different model. So the two figures in the first column, are 1) the original model: basemodel1 and 2) a fine-tuned model on basemodel1. The second column contains the base model basemodel2 and the model fine-tuned on basemodel2, etc.

5.2.3 Performance comparison among approaches

In this subsection, the performance of four variants of our approach are reported. The first variant is the same as our approach in all regards except that it resets the encoder weights instead of initializing them with the encoder weights of the base model (Figure 17). The second variant simply fine-tunes the whole base model on the target domain, so not freezing any model (Figure 1a). The third variant again is the same as our proposed approach with the exception that it omits the Disambiguation1 loss. The fourth variant temporarily (for 15 epochs) omits all structure enforcing losses. We evaluate the performance using two different base models, one which had good transfer performance using our original approach, and one with poor transfer performance. See Figure 10 for an overview of the performance of six approaches: our original approach, the four variants and learning from scratch.

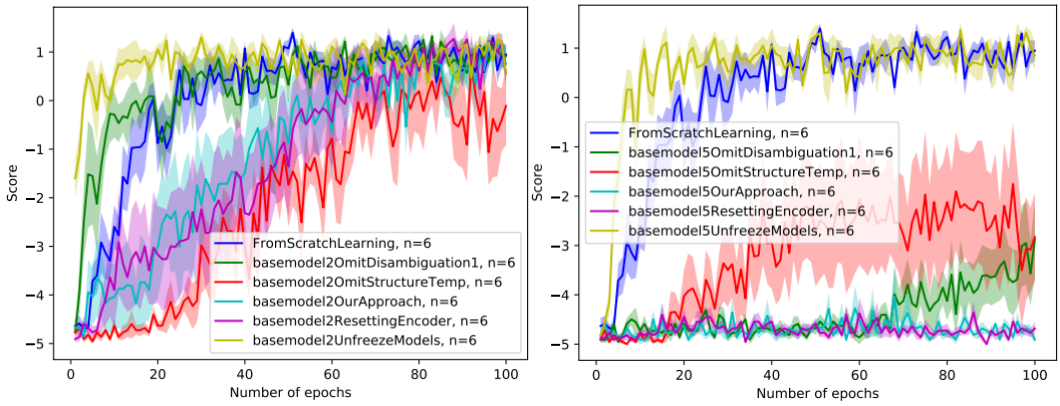

Figure 10: Visualisation of the average and standard error of the score for six approaches. The left depicts the transfer performance of the approaches using the good transfer performing base model (basemodel2), and on the right using the poor transfer performing base model (basemodel5). Note that the spikes in all lines can be explained due to the variance in the difficulty of randomly generated maps; completing the map successfully (collecting three rewards) will yield different scores based on the required amount of steps that have to be taken.

The results of the four variations in the transfer procedure are shown in Figure 10: 1) resetting the encoder (cyan) helps to avoid a local minimum for transfers (but not in all cases), 2) fine-tuning all models (purple) performs the best, 3) omitting the disambiguation1 loss (blue) performs on average better than learning from scratch, 4) temporarily omitting all structure enforcing losses (green)is the

only approach (without fine-tuning) that is able to perform some successful transfers in all cases,5) from scratch learning (yellow) has a low variance and scores second best, and finally 6) our original approach (red) performs the same as resetting the encoder variation. This implies that 1) fine-tuning all models indeed resolves the restricted degree of freedom in the optimization process, and 2) changing the loss function in the fine-tuning steps looks promising (interestingly enough it is still able to converge to the same solution although the loss function changed).

6 Discussion and Conclusion

Our main findings were that successful transfers when only retraining the encoder can converge, in some cases, to the same embedding space as the base model. However we also found that our proposed approach does not always converge to the same (optimal) embedding space as the base model, instead getting stuck in a local minimum. Furthermore we find that the success rate of the transfer is heavily dependant on the base model. A study of the local minima suggested that they are caused by two problems, namely 1) small losses area in the optimization landscape at the place of the local minima, and 2) the reduced degree of freedom of the optimization process caused by the frozen parts in the neural network models.

Quantitative results show that 1) learning from scratch is more stable and performs better than retraining only the encoder, and 2) fine-tuning all parts of the model (complete fine-tuning) outperforms all other approaches.

References

[1] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Christopher Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *CoRR*, abs/1912.06680, 2019.

[2] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575, 2019.

[3] S Gamrian and Y Goldberg. Transfer learning for related reinforcement learning tasks via image-to-image translation. *International Conference on . . .*, 2019. Query date: 2021-05-25 02:34:38.

[4] Jacob Tyo and Zachary Lipton. HOW TRANSFERABLE ARE THE REPRESENTATIONS LEARNED BY DEEP Q AGENTS? Technical report, 2020.

[5] Chaitanya Asawa, Christopher Elamri, and David Pan. Using Transfer Learning Between Games to Improve Deep Reinforcement Learning Performance and Stability. Technical report, 2017.

[6] Yunshu Du, Gabriel V De La Cruz, James Irwin, and Matthew E Taylor. Initial Progress in Transfer for Deep Reinforcement Learning Algorithms. Technical report, 2016.

[7] David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

[8] Vincent François-Lavet, Yoshua Bengio, Doina Precup, and Joelle Pineau. Combined Reinforcement Learning via Abstract Representations. *arXiv*, sep 2018.

[9] Danijar Hafner, Timothy P. Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *CoRR*, abs/2010.02193, 2020.

[10] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy P. Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *CoRR*, abs/1911.08265, 2019.

[11] Remo Sasso, Matthia Sabatelli, and Marco A. Wiering. Fractional transfer learning for deep model-based reinforcement learning. *CoRR*, abs/2108.06526, 2021.

[12] Simon S. Du, Akshay Krishnamurthy, Nan Jiang, Alekh Agarwal, Miroslav Dudík, and John Langford. Provably efficient RL with rich observations via latent state decoding. *CoRR*, abs/1901.09018, 2019.

[13] Zhuangdi Zhu, Kaixiang Lin, and Jiayu Zhou. Transfer Learning in Deep Reinforcement Learning: A Survey. Technical report.

[14] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015.

[15] Julius Berner, Philipp Grohs, Gitta Kutyniok, and Philipp Petersen. The modern mathematics of deep learning. *CoRR*, abs/2105.04026, 2021.