Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Everything is in the github repo, but in light of anonymization I have put it in a zip file
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] All are in the code repo, more details can be found in the CRAR paper, also see appendix E.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Standard deviations and standard error are used.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See appendix E
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] Cited CRAR
   (b) Did you mention the license of the assets? [No] License is in repo
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] The new code and data is in the github repo
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] No external data is used
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Analysis local minimum

In the following sections we provide arguments of 1) why the failing transfers of the low-dimensional maze specifically end up in the swapped embedding space local minimum previously visualised and 2) why it gets stuck in this local minimum.

A.1 Swapped embedding local minimum

When we look at the embedding space of the converged base model in Figure 11 and look at the environment and the position of the agent in Figure 3, we can conclude that the state space with the agent on position (2,3) is mapped to the embedding value: (-0.6, 0.2). For the sake of keeping the example as simple as possible we will round these value to (-0.5, 0). Meaning that the agent has an encoding function that converts the state (which is a 8 by 8 matrix where, for the non-inverse variant,

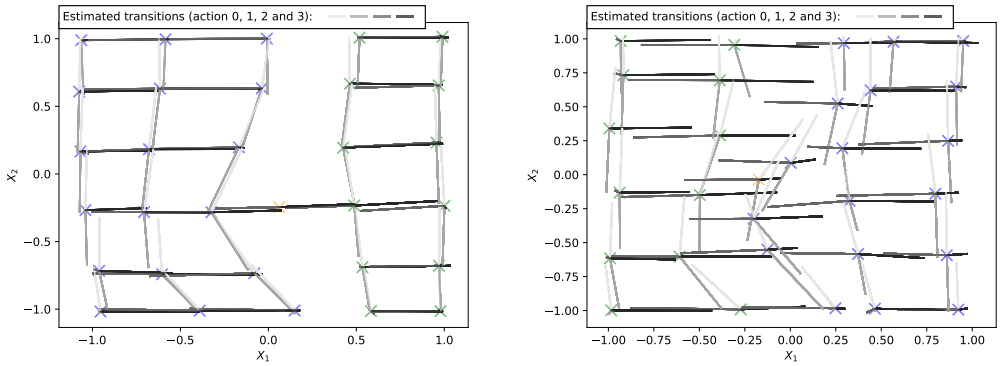

Figure 11: Visualisation of the embedding space learned by the CRAR agent in the low dimensional environment without any rewards (base model 2). The image on the left is the original model learned from scratch, whereas the image on the right is the (failed) fined-tuned model trained in the inverse environment.

an element has the value 1, 0 or 0.5 indicating a wall, path or player position, respectively) with the player position at (2,3), to the embedding value: (-0.5, 0). This is visualised on the left side of Figure 12.

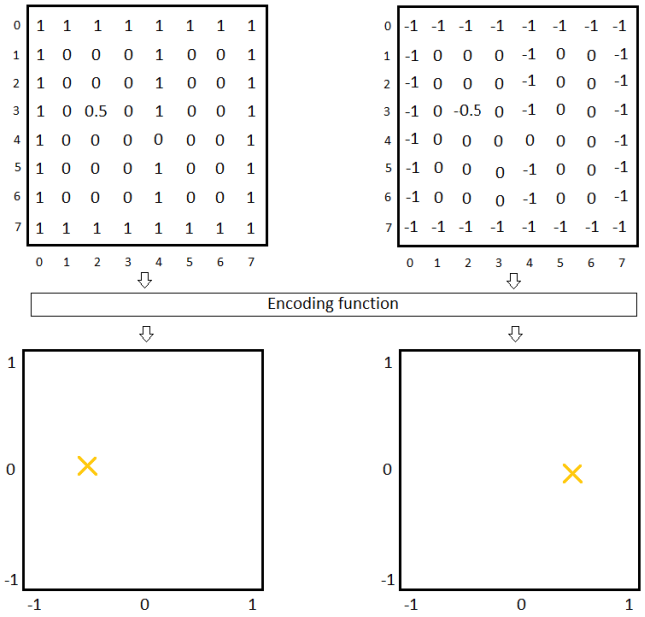

Figure 12: Visualisation of the swapped embedding space phenomena. The first row contains the matrices of the actual input values, for the original (left) and inverse (right) environment (as visualised in Figure 3). The second row contains the resulting embedding associated to that input when passed through the simple encoding function.

A solution of such an encoding function could simply be: embeddingX=state(2,3)*-1, embeddingY=0, which results in: -0.5=0.5*-1, 0=0. Now given this function we can feed the inverse state to this function and see what happens. Remember that the inverse environment simply replaced the 1, 0 and 0.5 tile values by -1, 0, and -0.5. Now if we take the same positions of all tiles but in the inverse fashion, such as depicted by the right image of Figure 3 and feed it to the encoding function just defined. We get the following result 0.5=-0.5*-1, 0=0. Meaning that we previously had an embedding of (-0.5, 0) and now have an embedding of (0.5,0) for the same hidden state namely having a state with the player at position (2,3). Which indeed results in the switched sides in the embeddings

previously seen. This ofcourse assumes that the simple function can be extrapolated to other states with different player locations as well, which is quite trivial.

From Figure 13 we can observe that the losses are quite similar between the failed transfer and the converged base model (only the transition loss is slightly bigger). This confirms that although the embedding space is swapped, the losses are still small. Also note that the reward (R), gamma (discount) and Q (model free) losses are all 0 because there is no reward and no terminal state.

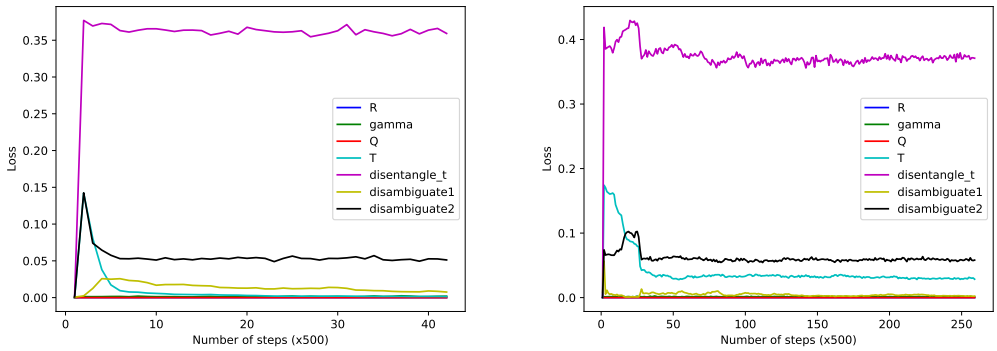

Figure 13: Visualisation of the losses over time. The left image shows the losses of the base agent, and the right image shows the losses of the failed fine-tuned agent in the target domain. Notice the difference in the transition loss

A.2 Stuck local minimum.

Now that it is clear why they often end up in the local minimum with switched sides in the embedding space, we can focus on why they keep stuck in this local optimum. We argue that a combination of two problems cause it to get stuck in the local optimum, namely 1) small losses for the initial weight initialisation in combination with a steep slope between this point and the global optimum and 2) lesser degree of freedom in the optimization process due to the frozen dynamics model. In the following paragraphs these will be explained.

First of all, let's take a more in-depth look why the losses are so small. We can feed the state of the inverse environment through the simple encoding function and then use the resulting embedding and an action to feed it through the transition function. We can then calculate the transition loss based on this new state embedding and the embedding of the actual expected state. Remember that the transition loss is simply the difference between the predicted embedding based on the state and an action, and the embedding of the actual state where you would end up in given the action. In Figure 14 we can observe the end result of this process.

We take the same player location (2,3) as before, thus the base model embedding is (-0.5,0) and the inverse embedding is (0.5,0) (illustrated with the yellow cross). Which if we wouldn't use rounding we find the actual corresponding inverse embedding (0.5, -0.25), as can be seen in the Figure 11. When we apply the transition function (given by the lines in Figure 11) to this embedding and the action 'up', we end up in the embedding value (0.5,0.25). Whereas if we actually get the inverse embedding of the state with a player location of (1,3), we first observe the base embedding: (-0.5, 0.5), then determine the simple encoding function for this state -0.5=0.5*-1, 0.5=0.5, and then finally calculate the inverse embedding by feeding it through the simple encoding function: 0.5=-0.5*-1, -0.5=-0.5, resulting in embedding value: (0.5, -0.5). Now we can approximate the transition loss by determining the distance between (0.5,0.25) and (0.5, -0.5). We can observe that this is quite small, and this is approximately the maximum transition loss that will be encountered, since it will never be more than two edges away (at the sides of the environment its even only one edge away).

We conclude that because of the switched sides the structure in the positions of the embedding values is maintained, and therefore the transition function will predict the embedding value of the opposite neighbour, resulting in a loss of maximum of two edge-lengths away (which for an embedding space with range -1 <> 1 and 6 positions is on average 0.33 * 2). However many embedding states (28 out

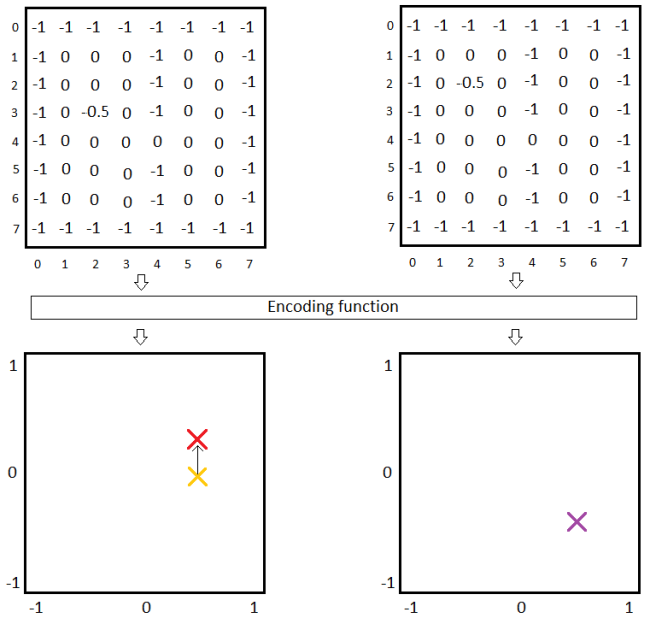

Figure 14: The first row contains the matrices with the actual input values, for the original player location (left) and for the player location when taken the up action (right). The yellow cross indicates the embedding value generated for the state with the original player location, red indicates the embedding value after applying the transition function when given the embedding value of the yellow cross and the up action, and the purple cross indicates the embedding value of the actual expected state. Note that the transition loss is the distance between the purple and red cross.

of 64) reside near a side; which only have a loss of one edge length. Regarding all the other losses: these will be same as the base model since the positions of the states are very similar and the other losses mainly look at the distances between the states.

It is easy to see that this small loss will result in a low point in the loss landscape. However a low point does not directly mean a local optimum, for example there can still lead a path downwards from this low point directly to the global optimum (lowest point). However we informally argue that this is not the case, and that there possible is quite a steep slope between the global minimum and the local minimum. So given 1) the normal environment and the converged base model and 2) the inverse environment and a failed fine-tuned model, we just showed that the embedding values are inversed for the same hidden state, for this we also showed that the losses are quite small because the structure of the values is maintained. Therefore the global optimum is at embedding value x and the (suboptimal inversed) local minimum is at value -x, and thus an embedding value $i \in \mathbb{Z} : -x < i < x$ will result in bigger losses because the structure is no longer maintained. More concretely, if we consider for a hidden state the embedding value of the base model E_b and the embedding of the failed transfer E_t a pair (E_b, E_t), then it is easy to see that for all pairs the middle value of the two values of a pair is always 0 ($E_b = x, E_t = -x, 0 = x - x$). Then if all embedding states need to converge past value 0 (which is the direct path between the stuck local minimum -x and the global optimum x) it is easy to see that for example the Disambiguate1 and the Disentangle loss will be at the highest point.

The second problem arises by the fact that the frozen dynamics model offers less freedom in the optimization process. We argue that the lack of freedom in combination with the first problem of getting caught in the local minimum will cause it to be stuck. We first describe this second problem in an intuitive manner, after which we provide it in a formal proof format. If we consider that the encoder and the dynamics weights have two separate loss landscapes. Then when the dynamics model weights change (e.g. get updated), then for a given input the loss changes (and its gradients w.r.t. the weights), which means that the loss landscape of the encoder model changes since the loss gets backpropagated through the dynamics model to the encoder(and even if the loss doesn't change but the weights do, the backpropagated gradients can change), see Theorem and Proof 2. Now if we consider a scenario where the encoder weights are stuck in a local minimum (which can happen

because the loss function is non-convex), we can imagine that the dynamics model will keep trying to converge to an optimum solution which keeps changing the loss landscape of the encoder model which might help in escaping the local minimum, either by 1) making the local minimum of the encoder model a global minimum by changing the dynamics so that the model as a whole performs optimal, or 2) that by chance a random change in the encoder loss landscape opens a path out of the local minima towards the global optimum. However, when we freeze the dynamics model we lose this degree of freedom, and this in combination with the poor encoder weight initialisation, described before, with small but suboptimal losses causes it to get stuck.

theorem Given a model $\hat{y}$ which consist of an encoder and a dynamics model where the loss of the output of this model will be backpropagated through the dynamics model into the encoder. Then the changes in the dynamics model will change the loss landscape of the encoder model (this of course could be generalized to any model architecture with two sequential models with a shared loss).

proof Given our model $\hat{y_1} = w_1 x + b_1$ where w_1 and b_1 contain all our weights and biases of our encoder and dynamics model. To simplify the proof we use a simple squared error as our loss function (instead of the CRAR losses): $L_1 = (y - \hat{y_1})^2$ which is the same as $L_1 = (w_1 x + b_1 - y)^2$. Where y is the target value. Then given a small change in the weights (e.g. after an update step) of our dynamics model which gives a different model $\hat{y_2} = w_2 x + b_2$, where $\hat{y_1} \neq \hat{y_2}$ because $w_1 \neq w_2$. Then given that the output of the models is different, it follows that the loss for a given input (x) is different between the two models, because: $(\hat{y_1} - y)^2 \neq (\hat{y_2} - y)^2$, thus $L_1 \neq L_2$. Then it also follows that the gradients of the losses with respect to the weights($\frac{\partial L_1}{\partial w_1} = 2x(w_1 x + b_1 - y)$) are different: $\frac{\partial L_1}{\partial w_1} \neq \frac{\partial L_2}{\partial w_2}$ because $2x(w_1 x + b_1 - y) \neq 2x(w_2 x + b_2 - y)$ since $w_1 \neq w_2$. Given that the encoder is part of the model and that the losses and gradients are different between the models, we can conclude that the loss landscape of the encoder model changes as the dynamics model changes its weights. Which implies that if we freeze the dynamics model weights, the encoder loss landscape will not change by means of changes in the dynamics model.

To future illustrate this reduced freedom in the optimization process, consider the loss landscape of Figure 15, and take for x the set of weights of the encoder model, for y the the set of weights of the dynamics model, and for z the loss function (w.r.t. the weights of the encoder and dynamics model). Which of course is an oversimplification since the plot is displaying the y- and x-axis as an numerical variable, instead of a set of numerical values, but for this point it suits the needs as we can assume that both models only contain a single weight. Using this visualisation it is easy to see that if we freeze the dynamics model we can remove the y-axis since this is now a single constant value instead of a possible range of values, meaning that we reduce the degree of freedom of the optimization process by one dimension.

We conclude that the combination of the two problems cause it to get stuck in a local optimum: the weight initialisation causes it to immediately find the swapped-sides-embedding solution with small losses and a steep slope between this solution and the global optimum, thus finding it self in a local minimum (problem 1), and the dynamics model is not able to alleviate this problem by changing the loss landscape because it is frozen (problem 2). Further concluding that freezing the dynamics model can result in a sub optimal solution, worse than the pre-trained model or training from scratch. The sub optimal performance is caused by the incompatible embedding space and thus poor task performance. We argue that due to randomness in the learning process sometimes the transfer escapes the local optimum and thus succeeds, and sometimes it gets stuck and thus fails.

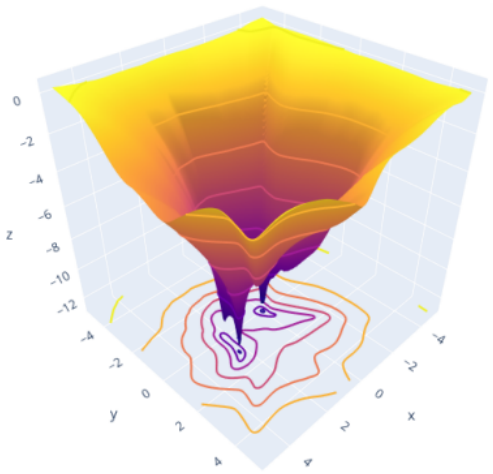

Figure 15: Visualisation of a loss landscape of a neural network[15].

B Losses of high-dimensional transfers

As before we find that the losses are quite small for the failing transfers (Figure 16). However, it can be noted that the reward, transition and q losses remain slightly higher. This indicates that the model did not converge to the same global optimum, and thus did not converge to the same embedding space. Remarkable here is that the losses are very small, but the agent is not able to perform the task at all (score of -5). This phenomena is caused by that the agent doesn't explicitly optimize the reward, it optimizes the implicit losses. This means that even if the agent is able to accurately estimate the reward for a state action pair, then the model could still have trouble collecting the rewards: for instance with the transition model being stuck in the swapped sides local minimum.

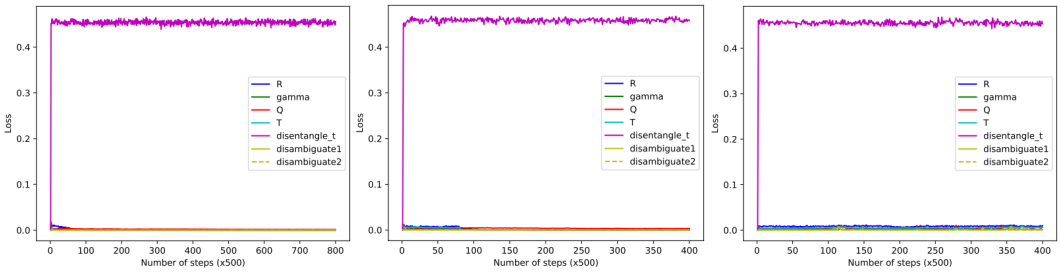

Figure 16: Visualisation of the losses associated to (from left to right): a base model, a successful transfer, and a failing transfer.

C Study of Approach Variations

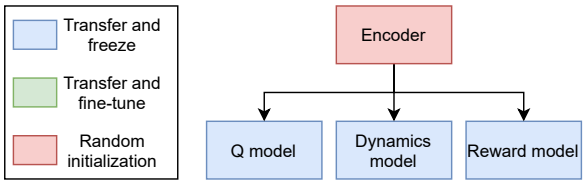

Figure 17: Variation 1: re-learn the encoder from scratch

D Study of learning rate

In the final experiment we experiment with different learning rates for the transfers (using the original proposed approach). We use the following learning rates: 0.00001, 0.00005, 0.0001, 0.0005, 0.001 and 0.005. To exclude the effect of different starting models, we use a single base model: basemodel2. Using this single base model, we will execute six transfers per learning rate, so a total of 36 transfers.

From Figure 18 we can observe that 0.001, 0.0005 and 0.00005 mostly have successful transfers, with 0.00005 having the biggest variance. We find that 0.005, 0.0001 and 0.00001 mostly fail, but 0.0001 and 0.00001 have quite a big variance. We conclude that the main trend we can observe from this experiment is that in the middle of the learning rates they perform quite well, but to the more extremes: 0.005 and 0.00001 they fail, which is as expected. The only exception being 0.0001, which appears to perform worse than both its neighbouring learning rates, even after increasing its sample size. We attribute this strange occurrence to the small sample sizes (e.g. 5e-05 might appear to perform much better than it actually should). Therefore the experiment results provide a general understanding of the behaviour of the different learning rates, but learning rate specific performance should be taken with a grain of salt.

E Hyperparameters and compute resources per experiment

For each experiment the used computing resources and the total usage time of the compute resource for the experiment are given:

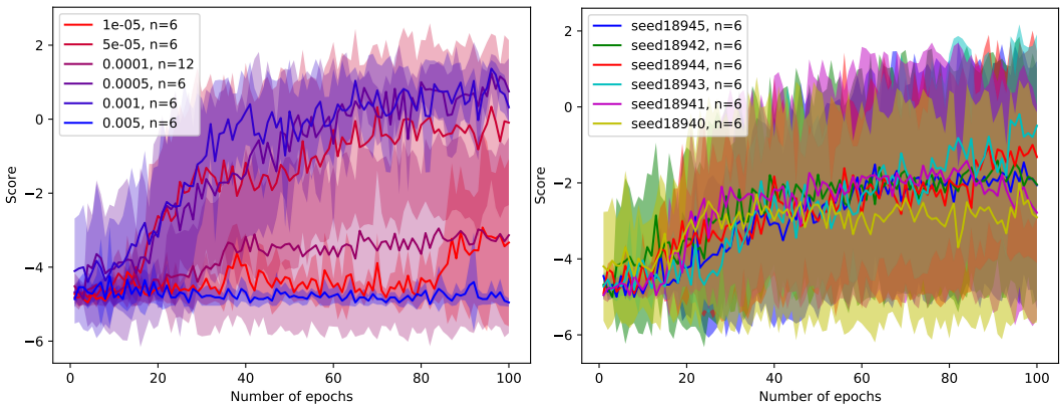

Figure 18: Visualisation of the average and the standard deviation of the score of 36 transfers, with the same base model but six different learning rates with six seeds each. In the left plot the data is grouped by the learning rate, and in the right group the data is grouped by the seed. The six extra samples for learning rate 0.0001 have been obtained using six different seeds, these have been omitted from the right plot due to irrelevance.

- Low dimensional maze: Nvidia GTX 1050Ti for 10 days.
- High dimensional maze:
  - Original approach with 5 different base models (section 5.2.1): 4 nodes with an Intel® Xeon® Bronze 3104 with 256GB memory and 4xNvidia 1080Ti for 2 days
  - Visualisation of embedding per base model (section 5.2.2): Nvidia GTX 1050Ti for 30 minutes
  - Performance of different approach (section 5.2.3): 4 nodes with an Intel® Xeon® Bronze 3104 with 256GB memory and 4xNvidia 1080Ti for 2 days