A. Supplementary Materials

A.1. The Reinforcement Learning Paradigm

The reinforcement learning (RL) paradigm consists of an agent receiving a sequence of observations s^o_t which are some function of environment states $s_t \in \mathcal{S}$ and may be accompanied by rewards $r_{t+1} \in R$ conditional on the actions $a_t \in \mathcal{A}$, chosen at each time step t (Sutton & Barto, 1998). We assume that these interactions can be modelled as a Markov Decision Process (MDP) (Puterman, 1994) defined as a tuple $D \equiv (\mathcal{S}, \mathcal{A}, \mathcal{T}, R, \gamma)$. $\mathcal{T} = p(s|s_t, a_t)$ is a transition function that models the distribution of all possible next states given action a_t is taken in state s_t for all $s_t \in \mathcal{S}$ and $a_t \in \mathcal{A}$. Each transition $s_t \overset{a_t}{\rightarrow} s_{t+1}$ may be accompanied by a reward signal $r_{t+1}(s_t, a_t, s_{t+1})$. The goal of the agent is to learn a policy $\pi(a_t|s_t)$, a probability distribution over actions $a_t \in \mathcal{A}$, that maximises the expected return i.e. the discounted sum of future rewards $R_t = E[\sum_{\tau=1}^{T-t} \gamma^{\tau-1} r_{t+\tau}]$. T is the time step at which each episode ends, and $\gamma \in [0, 1)$ is the discount factor that progressively down-weights future rewards. Given a policy $\pi(a|s)$, one can define the value function $V_\pi(s) = E[R_t|s_t = s, \pi]$, which is the expected return from state s following policy π. The action-value function $Q_\pi(s, a) = E[R_t|s_t = s, a_t = a, \pi]$ is the expected return for taking action a in state s at time t, and then following policy π from time $t + 1$ onward.

A.2. Further task details

A.2.1. DeepMind Lab

As described in Sec 3.1, in each source episode of DeepMind Lab the agent was presented with one of three possible room/object type conjunctions, chosen at random. These are marked D_S in Fig 2. The setup was a seek-avoid style task, where one of the two object types in the room gave a reward of +1 and the other gave a reward of -1. The agent was allowed to pick up objects for 60 seconds after which the episode would terminate and a new one would begin; if the agent was able to pick up all the 'good' objects in less than 60 seconds, a new episode was begun immediately. The agent was spawned in a random location in the room at the start of each new episode.

During transfer, the agent was placed into the held out conjunction of object types and room background; see D_T in Fig 2.

Visual pre-training was performed in other conjunctions of object type and room background denoted D_U in Fig 2.

The observation size of frames in the DeepMind Lab task was 84x84x3 (HxWxC).

A.2.2. MuJoCo/Jaco Arm Experiments

As described in Sec 3.2, the source task consisted of an agent learning to control a simulated arm in order to reach toward an object. A shaping reward was used, with a maximum value of 1 when the centre of the object fell between the pinch and grip sites of the end effector, or within a 10cm distance of the two. Distances on the x and y dimensions counted double compared to distances on the z dimension.

During each episode the object was placed at a random drop point within a 40x40cm area, and the arm was set to a random initial start position high above the work-space, independent of the object's position. Each episode lasted for 150 steps, or 7.5 seconds, with a control step of 50ms. Observations s^o_U were sampled randomly across episodes. Overall, 4 million frames of dimensions 64x64x3 (HxWxC) were used for this stage of the curriculum. For each episode the camera position and orientation were randomly sampled from an isotropic normal distribution centred around the approximate position and orientation of the real camera, with standard deviation 0.01. No precise measurements were used to match the two. Work-space table colour was sampled uniformly between -5% and $+5\%$ around the midpoint, independently for each RGB channel; object colours were sampled uniformly at random in RGB space, rejecting colours which fell within a ball around 10 held-out intensities (radius 10% of range); the latter were only used for simulated transfer experiments, i.e. in D_T in the sim2sim experiments. Additionally, Gaussian noise with standard deviation 0.01 was added to the observations s^o_T in the sim2sim task.

For the real Jaco arm and its MuJoCo simulation counterpart, each of the nine joints could independently take 11 different actions (a linear discretisation of the continuous velocity action space). In simulation Gaussian noise with standard deviation 0.1 was added to each discrete velocity output; delays in the real setup between observations and action execution were simulated by randomly mixing velocity outputs from two previous steps instead of emitting the last output directly. Speed ranges were between -50% and 50% of the Jaco arm's top speed on joints 1 through 6 starting at the base, while the fingers could use a full range. For safety reasons the speed ranges have been reduced by a factor of 0.3 while evaluating agents on the Jaco arm, without significant performance degradation.

A.3. Vision model details

A.3.1. Denoising Autoencoder for β-VAE

A denoising autoencoder (DAE) was used as a model to provide the feature space for the β-VAE reconstruction loss to be computed over (for motivation, see Sec. 2.3.1). The DAE was trained with occlusion-style masking noise in the vein of (Pathak et al., 2016), with the aim for the DAE to learn a semantic representation of the input frames. Concretely, two values were independently sampled from $U[0, W]$ and two from $U[0, H]$ where W and H were the width and height of the input frames. These four values determined the corners of the rectangular mask applied; all pixels that fell within the mask were set to zero.

The DAE architecture consisted of four convolutional layers, each with kernel size 4 and stride 2 in both the height and width dimensions. The number of filters learnt for each layer was {32, 32, 64, 64} respectively. The bottleneck layer consisted of a fully connected layer of size 100 neurons. This was followed by four deconvolutional layers, again with kernel sizes 4, strides 2, and {64, 64, 32, 32} filters. The padding algorithm used was 'SAME' in TensorFlow (Abadi et al., 2015). ReLU non-linearities were used throughout.

The model was trained with loss given by the L2 distance of the outputs from the original, un-noised inputs. The optimiser used was Adam (Kingma & Ba, 2014) with a learning rate of 1e-3.

A.3.2. β-VAE with Perceptual Similarity Loss

After training a DAE, as detailed in the previous section[6], a β-VAE$_{DAE}$ was trained with perceptual similarity loss given by

[6]In principle, the β-VAE$_{DAE}$ could also have been trained end-to-end in one pass, but we did not experiment with this.

Eq. 2, repeated here:

$$\mathcal{L}(\theta, \phi; \mathbf{x}, \mathbf{z}, \beta) = E_{q_\phi(\mathbf{z}|\mathbf{x})} \, \|J(\hat{\mathbf{x}}) - J(\mathbf{x})\|_2^2 - \beta \, D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) || p(\mathbf{z})) \tag{3}$$

Specifically, the input was passed through the β-VAE and a sampled[7] reconstruction was passed through the pre-trained DAE up to a designated layer. The L2 distance of this representation from the representation of the original input passed through the same layers of the DAE was then computed, and this formed the training loss for the β-VAE part of the β-VAE$_{DAE}$ [8]. The DAE weights remained frozen throughout.

The β-VAE architecture consisted of an encoder of four convolutional layers, each with kernel size 4, and stride 2 in the height and width dimensions. The number of filters learnt for each layer was $\{32, 32, 64, 64\}$ respectively. This was followed by a fully connected layer of size 256 neurons. The latent layer comprised 64 neurons parametrising 32 (marginally) independent Gaussian distributions. The decoder architecture was simply the reverse of the encoder, utilising deconvolutional layers. The decoder used was Gaussian, so that the number of output channels was $2C$, where C was the number of channels that the input frames had. The padding algorithm used was 'SAME' in TensorFlow. ReLU non-linearities were used throughout.

The model was trained with the loss given by Eq. 3. Specifically, the disentangled model used for DARLA was trained with a β hyperparameter value of 1 and the layer of the DAE used to compute the perceptual similarity loss was the last deconvolutional layer. The entangled model used for DARLA$_{\text{ENT}}$ was trained with a β hyperparameter value of 0.1 with the last deconvolutional layer of the DAE was used to compute the perceptual similarity loss.

The optimiser used was Adam with a learning rate of 1e-4.

A.3.3. β-VAE

For the MuJoCo/Jaco tasks, a standard β-VAE was used rather than the β-VAE$_{DAE}$ used for DeepMind Lab. The architecture of the VAE encoder, decoder and the latent size were exactly as described in the previous section A.3.2. β for the the disentangled β-VAE in DARLA was 175. β for the entangled model DARLA$_{\text{ENT}}$ was 1, corresponding to the standard VAE of (Kingma & Welling, 2014).

The optimizer used was Adam with a learning rate of 1e-4.

A.3.4. Denoising Autoencoder for baseline

For the baseline model DARLA$_{\text{DAE}}$, we trained a denoising autoencoder with occlusion-style masking noise as described in Appendix Section A.3.1. The architecture used matched that exactly of the β-VAE described in Appendix Section A.3.2 - however, all stochastic nodes were replaced with deterministic neurons.

The optimizer used was Adam with a learning rate of 1e-4.

A.4. Reinforcement Learning Algorithm Details

A.4.1. DeepMind Lab

The action space in the DeepMind Lab task consisted of 8 discrete actions.

DQN: in DQN, the convolutional (or 'vision') part of the Q-net was replaced with the encoder of the β-VAE$_{DAE}$ from stage 1 and frozen. DQN takes four consecutive frames as input in order to capture some aspect of environment dynamics in the agent's state. In order to match this in our setup with a pre-trained vision stack $\mathcal{F}_U$, we passed each observation frame $s^o_{\{1..4\}}$ through the pre-trained model $s^z_{\{1..4\}} = \mathcal{F}_U(s^o_{\{1..4\}})$ and then concatenated the outputs together to form the k-dimensional (where $k = 4|s^z|$) input to the policy network. In this case the size of s^z was 64 for DARLA as well as DARLA$_{\text{ENT}}$, DARLA$_{\text{DAE}}$ and DARLA$_{\text{FT}}$.

On top of the frozen convolutional stack, two 'policy' layers of 512 neurons each were used, with a final linear layer of 8 neurons corresponding to the size of the action space in the DeepMind Lab task. ReLU non-linearities were used throughout. All other hyperparameters were as reported in (Mnih et al., 2015).

A3C: in A3C, as with DQN, the convolutional part of the network that is shared between the policy net and the value net was replaced with the encoder of the β-VAE$_{DAE}$ in DeepMind Lab tasks. All other hyperparameters were as reported in (Mnih et al., 2016).

Episodic Control: for the Episodic Controller-based DARLA we used mostly the same hyperparameters as in the original paper by (Blundell et al., 2016). We explored the following hyperparameter settings: number of nearest neighbours $\in \{10, 50\}$, return horizon $\in \{100, 400, 800, 1800, 500000\}$, kernel type $\in$ {inverse, gaussian}, kernel width $\in \{1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 0.5, 0.99\}$ and we tried training EC with and without Peng's $Q(\lambda)$ (Peng, 1993). In practice we found that none of the explored hyperparameter choices significantly influenced the results of our experiments. The final hyperparameters used for all experiments reported in the paper were the following: number of nearest neighbours: 10, return horizon: 400, kernel type: inverse, kernel width: 1e-6 and no Peng's $Q(\lambda)$ (Peng, 1993).

UNREAL: We used a vanilla version of UNREAL, with parameters as reported in (Jaderberg et al., 2017).

A.4.2. MuJoCo/Jaco Arm Experiments

For the real Jaco arm and its MuJoCo simulation, each of the nine joints could independently take 11 different actions (a linear discretisation of the continuous velocity action space). Therefore the action space size was 99.

DARLA for MuJoCo/Jaco was based on feedforward A3C (Mnih et al., 2016). We closely followed the simulation training setup of (Rusu et al., 2016) for feed-forward networks using raw visual-input only. In place of the usual conv-stack, however, we used the encoder of the β-VAE as described in Appendix A.3.3. This was followed by a linear layer with 512 units, a ReLU non-linearity and a collection of 9 linear and softmax layers for the 9 independent policy outputs, as well as a single value output layer that outputted the value function.

[7] It is more typical to use the mean of the reconstruction distribution, but this does not induce any pressure on the Gaussians parametrising the decoder to reduce their variances. Hence full samples were used instead.

[8] The representations were taken after passing through the layer but before passing through the following non-linearity. We also briefly experimented with taking the L2 loss post-activation but did not find a significant difference.

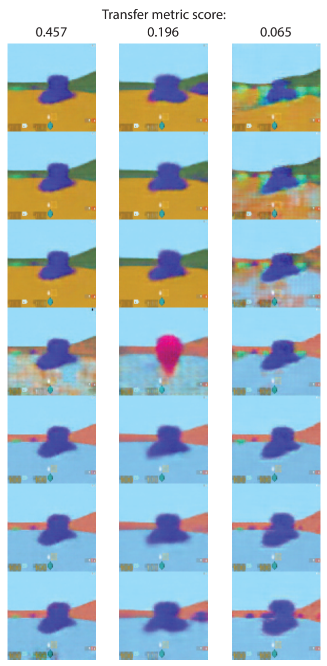

Figure 6. Traversals of the latent corresponding to room background for models with different transfer metric scores (shown top). Note that in the entangled model, many other objects appear and blue hat changes shape in addition to the background changing. For the model with middling transfer score, both the object type and background alter; whereas for the disentangled model, very little apart from the background changes.

A.5. Disentanglement Evaluation

A.5.1. Visual Heuristic Details

In order to choose the optimal value of β for the β-VAE -DAE models and evaluate the fitness of the representations s^z_U learnt in stage 1 of our pipeline (in terms of disentanglement achieved), we used the visual inspection heuristic described in (Higgins et al., 2017). The heuristic involved clustering trained β-VAE based models based on the number of informative latents (estimated as the number of latents z_i with average inferred standard deviation below 0.75). For each cluster we examined the degree of learnt disentanglement by running inference on a number of seed images, then traversing each latent unit $z_{\{i\}}$ one at a time over three standard deviations away from its average inferred mean while keeping all other latents $z_{\{\backslash i\}}$ fixed to their inferred values. This allowed us to visually examine whether each individual latent unit z_i learnt to control a single interpretable factor of variation in the data. A similar heuristic has been the *de rigueur* method for exhibiting disentanglement in the disentanglement literature (Chen et al., 2016; Kulkarni et al., 2015).

A.5.2. Transfer Metric Details

In the case of DeepMind Lab, we were able to use the ground truth labels corresponding to the two factors of variation of the object type and the background to design a proxy to the disentanglement metric proposed in (Higgins et al., 2017). The procedure used consisted of the following steps:

1) Train the model under consideration on observations s^o_U to learn $\mathcal{F}_{\mathcal{U}}$, as described in stage 1 of the DARLA pipeline.

2) Learn a linear model $\mathcal{L} : S^z_V \to M \times N$ from the representations $s^z_V = \mathcal{F}_V(s^o_V)$, where $M \in \{0, 1\}$ corresponds to the set of possible rooms and $N \in \{0, 1, 2, 3\}$ corresponds to the set of possible objects[9]. Therefore we are learning a low-VC dimension classifier to predict the room and the object class from the latent representation of the model. Crucially, the linear model $\mathcal{L}$ is trained on only a subset of the Cartesian product $M \times N$ e.g. on $\{\{0, 0\}, \{0, 3\}, \{1, 1\}, \{1, 2\}\}$. In practice, we utilised a softmax classifier each for M and N and trained this using backpropagation with a cross-entropy loss, keeping the unsupervised model (and therefore $\mathcal{F}_{\mathcal{U}}$) fixed.

3) The trained linear model $\mathcal{L}$'s accuracy is evaluated on the held out subset of the Cartesian product $M \times N$.

Although the above procedure only measures disentangling up to linearity, and only does so for the latents of object type and room background, we nevertheless found that the metric was highly correlated with disentanglement as determined via visual inspection (see Fig. 6).

A.6. Background on RL Algorithms

In this Appendix, we provide background on the different RL algorithms that the DARLA framework was tested on in this paper.

A.6.1. DQN

(DQN) (Mnih et al., 2015) is a variant of the Q-learning algorithm (Watkins, 1989) that utilises deep learning. It uses a neural network to parametrise an approximation for the action-value function $Q(s, a; \theta)$ using parameters θ. These parameters are updated by minimising the mean-squared error of a 1-step lookahead loss $\mathcal{L}_Q = E\left[(r_t + \gamma max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta))^2\right]$, where θ^- are parameters corresponding to a frozen network and optimisation is performed with respect to θ, with θ^- being synced to θ at regular intervals.

A.6.2. A3C

Asynchronous Advantage Actor-Critic (A3C) (Mnih et al., 2016) is an asynchronous implementation of the advantage actor-critic paradigm (Sutton & Barto, 1998; Degris & Sutton, 2012), where separate threads run in parallel and perform updates to shared parameters. The different threads each hold their own instance of the environment and have different exploration policies, thereby decorrelating parameter updates without the need for experience replay.

A3C uses neural networks to approximate both policy $\pi(a|s; \theta)$ and value $V_\pi(s; \theta)$ functions using parameters θ using n-step look-ahead loss (Peng & Williams, 1996). The algorithm is trained using an advantage actor-critic loss function with an entropy regularisation penalty: $\mathcal{L}_{A3C} \approx \mathcal{L}_{VR} + \mathcal{L}_\pi - E_{s\sim\pi}\left[\alpha H(\pi(a|s; \theta))\right]$, where H is entropy. The parameter updates are performed after every t_{max} actions or when a terminal state is reached. $\mathcal{L}_{VR} =$

[9]For the purposes of this metric, we utilised rooms with only single objects, which we denote by the subscript V e.g. the observation set S^o_V.

$E_{s\sim\pi}\left[(R_{t:t+n}+\gamma^n V(s_{t+n+1};\theta)-V(s_t;\theta))^2\right]$ and $\mathcal{L}_\pi = E_{s\sim\pi}\left[log\,\pi(a|s;\theta)(Q^\pi(s,a;\theta)-V^\pi(s;\theta))\right]$. Unlike DQN, A3C uses an LSTM core to encode its history and therefore has a longer term memory permitting it to perform better in partially observed environments. In the version of A3C used in this paper for the DeepMind Lab task, the policy net additionally takes the last action a_{t-1} and last reward r_{t-1} as inputs along with the observation s^o_t, as introduced in (Jaderberg et al., 2017).

A.6.3. UNREAL

The *UNREAL* agent (Jaderberg et al., 2017) takes as a base an LSTM A3C agent (Mnih et al., 2016) and augments it with a number of unsupervised auxiliary tasks that make use of the rich perceptual data available to the agent besides the (sometimes very sparse) extrinsic reward signals. This auxiliary learning tends to improve the representation learnt by the agent. While training the base agent, its observations, rewards, and actions are stored in a replay buffer, which is used by the auxiliary learning tasks. The tasks include: 1) pixel control the agent learns how to control the environment by training auxiliary policies to maximally change pixel intensities in different parts of the input; 2) reward prediction - given a replay buffer of observations within a short time period of an extrinsic reward, the agent has to predict the reward obtained during the next unobserved timestep using a sequence of three preceding steps; 3) value function replay - extra training of the value function to promote faster value iteration.

A.6.4. Episodic Control

In its simplest form EC is a lookup table of states and actions denoted as $Q^{EC}(s,a)$. In each state EC picks the action with the highest Q_{EC} value. At the end of each episode $Q^{EC}(s,a)$ is set to R_t if $(s_t,a_t)\notin Q^{EC}$, where R_t is the discounted return. Otherwise $Q^{EC}(s,a)=max\left\{Q^{EC}(s,a),R_t\right\}$. In order to generalise its policy to novel states that are not in Q^{EC}, EC uses a non-parametric nearest neighbours search $\widehat{Q^{EC}}(s,a)=\frac{1}{k}\sum_{i=1}^{k}Q^{EC}(s^i,a)$, where $s^i, i=1,...,k$ are k states with the smallest distance to the novel state s. Like DQN, EC takes a concatenation of four frames as input.

The EC algorithm is proposed as a model of fast hippocampal instance-based learning in the brain (Marr, 1971; Sutherland & Rudy, 1989), while the deep RL algorithms described above are more analogous to slow cortical learning that relies on generalised statistical summaries of the input distribution (McClelland et al., 1995; Norman & O'Reilly, 2003; Tulving et al., 1991).

A.7. Source Task Performance Results

The focus of this paper is primarily on zero-shot domain adaptation performance. However, it is also interesting to analyse the effect of the DARLA approach on source domain policy performance. In order to compare the models' behaviour on the source task, we examined the training curves (see Figures 7-10) and noted in particular their:

1. Asymptotic task performance, i.e. the rewards per episode at the point where π_S has converged for the agent under consideration.
2. Data efficiency, i.e. how quickly the training curve was able to achieve convergence.

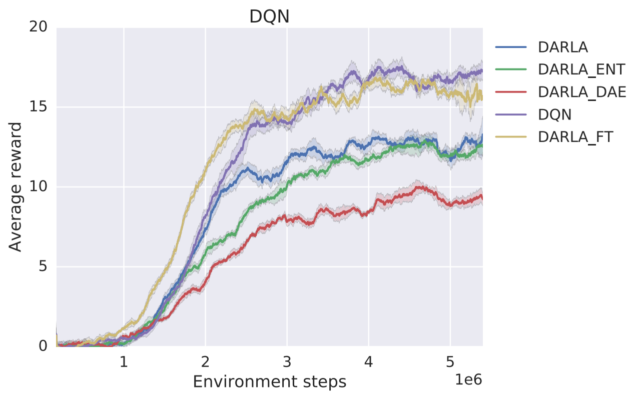

Figure 7. Source task training curves for DQN. Curves show average and standard deviation over 20 random seeds.

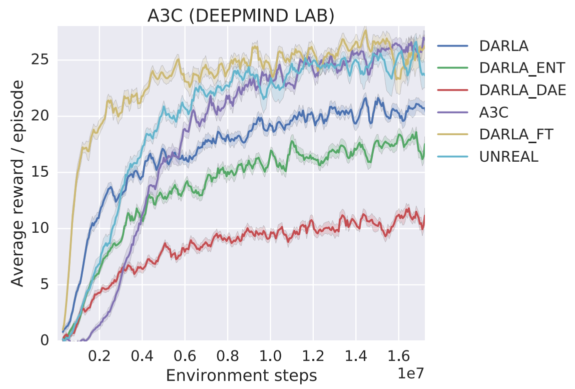

Figure 8. Source task performance training curves for A3C and UNREAL. DARLA shows accelerated learning of the task compared to other architectures. Results show average and standard deviation over 20 random seeds, each using 16 workers.

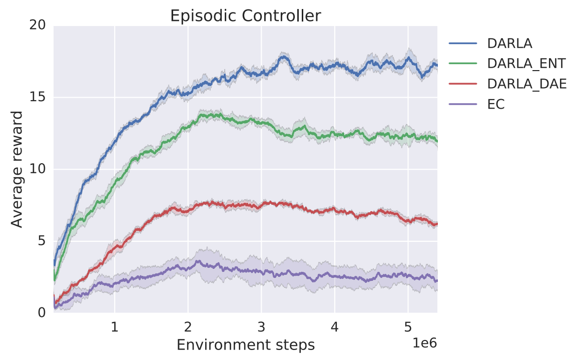

Figure 9. Source task training curves for EC. Results show average and standard deviation over 20 random seeds.

We note the following consistent trends across the results:

1. Using DARLA provided an initial boost in learning performance, which depended on the degree of disentanglement of the representation. This was particularly observable in A3C,

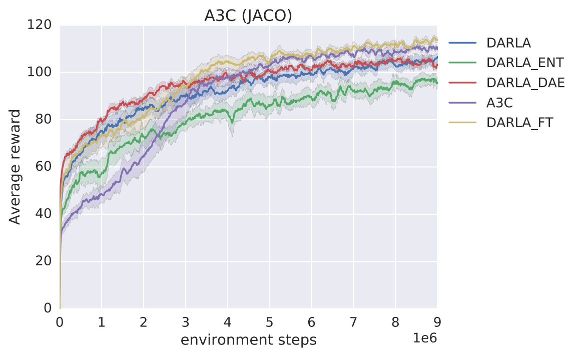

Figure 10. Training curves for various baselines on the source MuJoCo reaching task

see Fig. 8.

2. Baseline algorithms where $\mathcal{F}$ could be fine-tuned to the source task were able to achieve higher asymptotic performance. This was particularly notable on DQN and A3C (see Figs. 7 and 8) in DeepMind Lab. However, in both those cases, DARLA was able to learn very reasonable policies on the source task which were on the order of 20% lower than the fine-tuned models – arguably a worthwhile sacrifice for a subsequent median 270% improvement in target domain performance noted in the main text.

3. Allowing DARLA to fine-tune its vision module (DARLA_{FT}) boosted its source task learning speed, and allowed the agent to asymptote at the same level as the baseline algorithms. As discussed in the main text, this comes at the cost of significantly reduced domain transfer performance on A3C. For DQN, however, finetuning appears to offer the best of both worlds.

4. Perhaps most relevantly for this paper, even if solely examining source task performance, DARLA outperforms both $\text{DARLA}_{\text{ENT}}$ and $\text{DARLA}_{\text{DAE}}$ on both asymptotic performance and data efficiency – suggesting that disentangled representations have wider applicability in RL beyond the zero-shot domain adaptation that is the focus of this paper.