A. Ground Truth Evaluation

Whenever ground truth values are used in the supervised setups, we compute these by building an explicit MDP for each R_g, converted to tabular representation, and then solving for V_g or Q_g. We use these same MDPs also for evaluating policy quality: we compute the expected discounted return when the induced stochastic policy from $Q(s, a, g)$ is run on the MDP (with temperature $\tau = 0.05$), and average these values over all possible start states. Of course, this ground-truth based procedure is only applicable to small-scale problems.

B. Neural Network Details

All neural networks mentioned use two hidden layers of size 128, rectified linear units, all of them implemented using the Torch7 library (Collobert et al., 2011a). All optimisation uses the SGD-variant Adam (Kingma & Ba, 2014) with its recommended default hyper-parameters, a mini-batch size of 20 and learning-rate $\alpha = 0.005$. The only exception to this setup is for the larger-scale Ms Pacman experiment described below.

C. Ms Pacman Experiment Details

150 subgoals were defined, for collecting particular pellets in the game. Specifically, for $i \in \{1, ..., 150\}$, pseudo-reward and pseudo-discount functions for each goal g_i were given by:

$$R_{g_i}(s, a, s') = \begin{cases} 1 & \neg\text{pellet}_i(s') \wedge \text{pellet}_i(s) \\ & \wedge \gamma_{ext}(s) \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\gamma_{g_i}(s) = \begin{cases} 0 & \neg\text{pellet}_i(s) \\ \gamma_{ext}(s), & \text{otherwise} \end{cases}$$

where γ_{ext} is the external discount function and pellet(s') is true if and only if the corresponding pellet is still in the game.

For reasons of expedience, a few simplifications are made to the environment during training. These should not dramatically affect the performance of the method, however. For each of the above-defined subgoals, we construct a modified version of the Ms Pacman Atari environment (Bellemare et al., 2012) as follows. When the subgoal is achieved, the environment is reset, Pacman's position is set to a random location, and a new episode begins. At the beginning of each episode we skip over a start-up time of 260 frames where nothing happens in the game.

Each demon uses a variant of Deep Q-Learning with Experience Replay (Mnih et al., 2013) to learn the value functions with respect to its subgoal. The external discount factor is fixed at 0.95. An action repeat of 4 is used, and frames are preprocessed by conversion to grayscale and downsampling to 84x84 pixels. The input to the network is the concatenation of the last 4 frames, i.e. an 84x84x4 tensor. The first hidden layer convolves this with 16 8x8 filters, with stride 4, and then a rectifier. The second layer is another convolution, with 32 4x4 filters, with stride 2, and a further rectifier. This is followed by a fully-connected layer of 256 rectifier units. The output layer is linear, followed by a Log-SoftMax. The action set is restricted to the up, down, left and right actions.

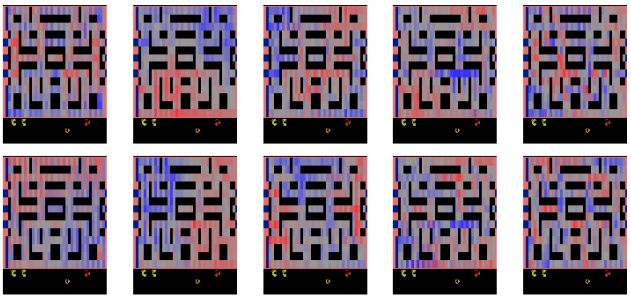

Figure 14. The factors discovered when training a UVFA of rank $n = 10$ on all data (150 pellet goals). They visibly capture spatial regions, but of course the full value functions are much more complex, and involve power-pills and ghosts as well – those are just more difficult to visualise. However, when running a Pacman agent that follows the UVFA policy for one of the pellet goals, it consistently reaches it, while avoiding ghosts on the way.

We use the value estimates learned in this manner to train a UVFA, by performing a rank 10 SVD decomposition to obtain desirable embedding vectors. We then train the goal half of the UVFA network to map each pellet's (x, y) position (normalized to the interval $[0, 1]$) to the corresponding vector in the embedding space, using supervised targets. The goal half of the UVFA is a multi-layer perceptron with rectifier units, and 4 hidden layers of size 100. This is trained on 10000 minibatches of size 20, using Adam with learning rate 0.001. The state half of the network is tabular.

In order to visualize the values learned by the network, we define a set of mapping states, which are generated by taking the initial state of the game, and teleporting pacman to each point in turn on a 37x14 grid, by modifying the appropriate memory location in the emulator state. If the resulting position is not a wall, the corresponding observation image is duplicated 4 times and fed into the network. The maximal output is taken as an estimate of the value of that state.

Figure 14 illustrates the matrix factorization in the Pacman environment. Each of the ten plots illustrates a single component of the state embedding $\phi(s)$, sweeping over a set of states s in which Pacman's position has been set to the corresponding location.

D. State Embeddings

The same embedding techniques that underlie our two-stream UVFA architectures can be used for another,

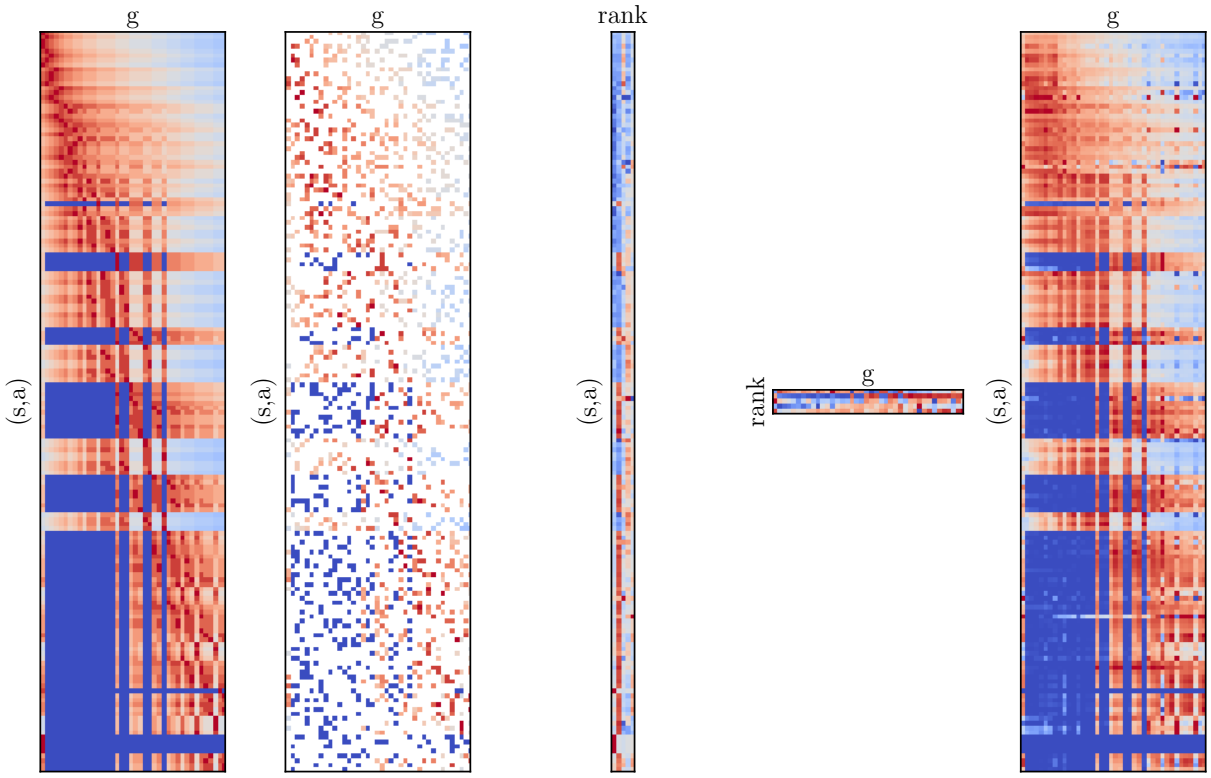

Figure 13. Illustration of sparse matrix decomposition for a 2-room LavaWorld. From left to right: ground truth Q-values, sparse matrix with subset of visible ones $\mathbf{M}$, $\hat{\phi}$, $\hat{\psi}$, reconstructed Q-values for all entries.

straightforward purpose, namely for learning an embedding of states that induces a useful, reachability-based metric. In particular, we can use $\mathcal{G} = \mathcal{S}$ and a very simple parameter-free combination function of the form $h(\cdot,\cdot) = \gamma^{D(\cdot,\cdot)}$, where D is a distance function (as in section 5.3).

The motivation for this form comes from natural language processing literature. Several works, for example (Collobert et al., 2011b; Mikolov et al., 2013), define an embedding vector for every word. The training objective is to predict a given word from words around it in a sentence. The probability of such prediction is calculated as essentially an exponential of the negative distance between the word vector and a vector obtained from words around it. After training, words with similar meaning will have their vectors close in the embedding space. In our case states of the environment are related to one another. Specifically it is easy to move from one state to some states but hard to move to other states (it takes a longer time for example). What we would like to achieve is that states that are easily reachable from one another are close in the embedding space and those which are not are far. This is what we indeed observe in experiments. Initially random embedding vectors organize to reflect relations between states. For example in an open two dimensional grid world we find that the vectors roughly lie on a two dimensional plane, thus recovering the underlying structure of the environment.

We test the ability of embeddings to generalize in three settings: 1) V learning with shared state and goal embeddings $\phi = \psi$, 2) V learning without sharing and 3) Q learning (no sharing possible). In many environments it might be easy to go from state s to state s' but not vice-versa. In this case a symmetric distance function is not appropriate, so we propose both a symmetric and an asymmetric D:

$$\begin{aligned} D_S(s,g) &= \|\phi(s) - \psi(g)\|_2 \\ D_A(s,g) &= \|\sigma(\psi_1(g))(\phi(s) - \psi_2(g)))\|_2 \end{aligned}$$

where σ is a the logistic function and ψ_1 and ψ_2 are two halves of the embedding vector of g. When learning Q, the ϕ embeddings are for pairs (s,a).

If a transition model is available we can instead do bootstrap learning with only V, by replacing $Q(s_{t+1}, a', g)$ in the $\max$ of Equation 1 by $V(T_{a'}(s_{t+1}), g)$, where $T_a(s)$ denotes the state reached after executing action a in state s.

We look at the training time and generalization in the settings of Section 4.1 where only a fraction of state goal pairs is present. We use 4 room environment of size 8. The training time is shown in the Figure 15. We see that sharing the

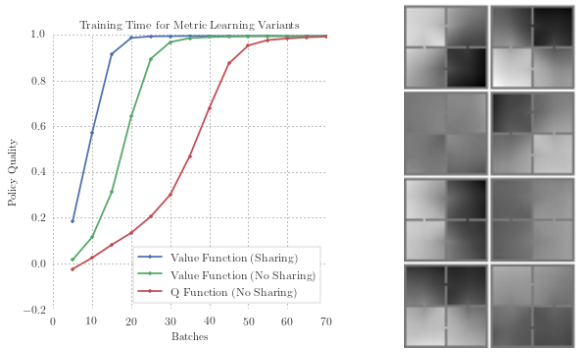

Figure 15. **Left**: Policy quality as a function of batches of bootstrapping updates, using V learning with shared and not shared representations for goals and using Q learning, in a 8x8 4-rooms environment. Learning is easier with a shared architecture, but sharing is only applicable when learning V: the embeddings needed for Q depend on (s, a) jointly in one stream, but only on g in the other. Given that the Q case must learn 4 times as many values (differentiating actions in every state), its learning performance is very much in line with the expected 4x slowdown. **Right**: overlaid mapping of 8 embedding factors for a 32×32 environment. Note how they capture smooth changes within rooms, while capturing topology and spilling through doorways.

state and goal representation gives the best performance.

Unlike the 4-room environment, the dynamics in LavaWorld are not reversible. There we compared the two distance function D_S and D_A, finding that in a two room, 10×10 LavaWorld environment D_S obtains a policy quality of 0.95, while D_A obtained 1.0. Surprisingly, the symmetric function captured the environment quite well, but not as well as the asymmetric functions that captured it perfectly.