High-level Features for Resource Economy and Fast Learning in Skill Transfer

Alper Ahmetoglu[a*], Emre Ugur[a], Minoru Asada[b], Erhan Oztop[b,c]

[a]*Department of Computer Engineering, Bogazici University, Turkey*
[b]*OTRI/SISREC, Osaka University, Osaka, Japan*
[c]*Department of Computer Science, Ozyegin University, Turkey*

ARTICLE HISTORY
Compiled March 6, 2025

ABSTRACT
Abstraction is an important aspect of intelligence which enables agents to construct robust representations for effective and efficient decision making. In the last decade, deep neural networks are proven to be effective learning systems that are applicable to a wide range of application domains, in particular, due to their ability to form increasingly complex abstractions at successive layers in an end-to-end fashion. However, these abstractions, unless enforced, are mostly distributed over many neurons, making the re-use of a learned skill costly and blind to the insights that can be obtained on the emergent representations. Previous work either enforced formation of abstractions creating a designer bias, or used a large population of neural units without investigating how to obtain high-level low-dimensional features that may more effectively capture the source task. For avoiding designer bias and unsparing resource use, we propose to exploit neural response dynamics to form compact representations to use in skill transfer. For this, we consider two competing methods based on (1) information loss/maximum information compression principle and (2) the notion that abstract events tend to generate slowly changing signals, and apply them to the neural signals generated during task execution. To be concrete, in our simulation experiments, we either apply principal component analysis (PCA) or slow feature analysis (SFA) on the signals collected from the last hidden layer of a deep neural network while it performs a source task, and use these features for skill transfer in a new, target, task. We then compare the generalization and learning performance of these alternatives with the baselines of skill transfer with full layer output and no-transfer settings. Our experimental results on a simulated tabletop robot arm navigation task show that units that are created with SFA are the most successful for skill transfer. SFA as well as PCA, incur less resources compared to usual skill transfer where full layer outputs are used in the new task learning, whereby many units formed show a localized response reflecting end-effector-obstacle-goal relations. Finally, SFA units with the lowest eigenvalues resembles symbolic representations that highly correlate with high-level features such as joint angles and end-effector position which might be thought of precursors for fully symbolic systems.

KEYWORDS
transfer learning, symbol emergence, reinforcement learning

1. Introduction

It can be argued that intelligent behavior, both artificial and biological agents, require the use of abstractions for high competency in the real world [1]. Abstraction allows an agent to filter out irrelevant aspects of incoming stimuli, and thus, equips it with a robust representation. From a biological perspective, it is not clear how concept and symbol representations are formed by the central nervous system [2]. Yet, behavioral markers of such representations can be seen even in

*Corresponding author. Email: alper.ahmetoglu@boun.edu.tr

animals at the lower levels of cognitive hierarchy. For example, toads have abstract prey and predator feature detectors that respond according to the high level features present in the visual stimuli related to the size and shape relative to the direction of motion [3]. From the robotics point of view, classically, a set of concepts or symbols can be defined apriori and later grounded to the sensorimotor stimuli of the agent acting in its environment [4]. More recent approaches aim to abstract representations directly from the sensorimotor experience of the agent [5–8].

In fact, the deep learning approaches form complex feature representations by combining simpler ones in successive layers [9]. As deep neural networks have been very successful in many real world problems such as image recognition [10], machine translation [11], playing the game of Go [12], and protein folding problem [13], most of the focus has been put on improving performance and finding newer applications of deep learning. One of the bottlenecks of learning in deep networks is the large amount of data required for learning tasks from scratch. To attack this problem, knowledge or skill transfer methods allowing the reuse of learned representations have started to appear [14,15], although similar ideas were present earlier in the robot learning domain [16]. One straightforward way to achieve knowledge transfer is to create a new network with access to all the layers of original network that was used to learned the initial task (e.g. [17,18]). This ensures that any representation discovered will have a chance of exploitation during new task learning. However, in these approaches all the units in the layers are linked to the new network through adaptable weights, and thus, which of those units need to be exploited for the new task is left for the learning algorithm to find. Considering resource economy both in terms of memory and energy, this seems wasteful. In this sense, a principled task-agnostic way of selecting *which* units to use for the next task learning is highly desirable. This study aims to make a step forward in this direction.

Earlier robotic work indicates that units that capture high level features are potential candidates to facilitate effective knowledge transfer [19]. From a neural computation point of view, this also makes sense as compact high-level representations would be more economical to process and pass around in the neural circuits of biological systems. Thus, the brain might adopt a strategy to represent sensorimotor information compactly with minimum information loss, inline with the information compression ideas [20,21]. Another hypothesis can be obtained by generalizing the idea that slowly changing features in sensory data tend to correspond to more high-level concepts [22] to neural responses. According to this view, the brain might seek to exploit those neurons that are more stable over the others that show frequent changes. For example, when a hand waving action is observed, the earlier sites in the visual processing pathway (e.g. areas V1, V2) would show temporally changing activity where as at the end of the processing pipeline (i.e. in area IT), the recognition of the hand waving action would be represented with a few neurons, which would show stable activity during the most of the observation period over a variety hand waving actions.

Motivated from these hypotheses, in this paper, we analyze different ways of transferring previously learned representations for a new task in an economical way. More specifically, we consider two methods: (1) principal component analysis (PCA) for the minimum information loss, and (2) slow feature analysis (SFA) [22] for creating signals that change slowly. PCA reduces the dimensionality of the data while preserving the information maximally which is a suitable candidate for transferring compact features. On the other hand, SFA, which is not very well explored in the transfer learning context, creates slowly changing representations which are arguably more robust because objects of interest in the real world do not make abrupt changes considering the notion of object permanency. In our experiments, we first train a deep Q-network [23] on a task, where the goal is to move the robot arm to a desired target position in the presence of a rectangular obstacle which may appear at different locations. After training, we create separate skill transfer scenarios in which either PCA or SFA transformations are applied on the last hidden layer activations of the network and used in new task learning by augmenting the features found to the last hidden layer of the new network. We compare these scenarios with the baselines of skill transfer with full layer output and no-transfer scenarios. Our experimental results show that:

- Using features that are constructed with SFA is not only more economical in terms of the number of units, but also better for skill transfer than the naive approach of using the full set of layer activations.
- PCA is also helpful for resource economy in skill transfer, but not as good as SFA in terms of success rate of the new task.
- SFA and PCA capture interpretable high-level features such as joint angles, tip locations, and the distance from the tip position to the goal position, solely from the activation history of the network.

In the rest of this paper, we detail our methodology in Section 2, define the experimental setup in Section 3, give the results in Section 4, and conclude in Section 5.

1.1. Related Work

There has been an increased attention on learning high-level abstractions from the experience of the robot. Konidaris et al. proved that learning symbols for the pre- and post-conditions of executed actions are necessary and sufficient for determining the feasability of a plan [6]. Based on this, they constructed symbols for pre-conditions and effects of executed actions in a 2d game environment. In a follow-up work, they moved this framework into the probabilistic setting and used the learned symbols in a real-world robot to make plans [24]. Ugur and Piater followed an object-centric strategy and created symbols from clusters of object features and effects for each action [19]. A more recent follow-up work uses deep neural networks with a binary bottleneck layer to predict the effect of executed actions in an end-to-end fashion [8]. Activations in the bottleneck layer are then treated as symbols and used for planning. A similar work also employs deep neural networks with a binary layer but symbols are learned independently from the executed actions and observed effects. All of these works force symbols to be discrete and this effectively limits the capability of the system. A more desirable property would be to allow the system to use continuous representations and observe discrete symbols only as a by-product of some learning process. To this end, in this paper, we explore different methods of constructing features in a generic, task-agnostic way to investigate whether they exhibit a symbol-like meaning.

PCA is a frequently used method for dimensionality reduction and data analysis due to its ability to create features that capture the data with a desired level of information loss. In particular, it allows the tuning of the amount of information loss by choosing the principal vectors to represent data by simply examining the eigenvalues corresponding to them. Santello et al. used PCA to analyze the intrinsic dimensionality of hand posture for grasping different objects and found that the first two dimensions cover more than 80% of the variance [25]. Tripathi and Wagatsuma applied PCA to different joint sets and time segments to impose a prior on the PCA algorithm based on the control approach of the central nervous system [26]. Promsri and Federolf analyzed the effect of using PCA on the acceleration of postural changes as opposed using PCA only on the amplitude [27]. As a non-linear approach, Chen et al. combined sparse autoencoders with dynamic movement primitives [28] for 50-dimensional human movement data, and showed that new data can be generated using only a single neuron of the trained system [29].

SFA [22] can be loosely thought as the application of PCA to the first time-derivative of the data. Therefore, it is particularly useful for extracting slowly changing signals when the low eigenvalue vectors are chosen. Dawood and Loo used an incremental version of SFA [30] for action segmentation from high-dimensional video input. A similar work to ours studied the use of hierarchical SFA features extracted from raw image data in the context of reinforcement learning [31]. Our study differs from this work as we are interested in extracting high-level features from an already formed network, be it either prewired or learned, and use them in subsequent tasks.

2. Methods

Suppose that an agent has developed a neural system to solve a specific task. When the agent encounters a similar task, neurons in this system will respond to the sensorimotor input albeit possibly in a different way since the input will be different. Even though the system does not directly provide the appropriate motor control output, it would be economically viable to use a previously learned network instead of re-learning everything from scratch. We focus on extracting features from this network in an economical way. Note that it is not important how this network has been formed; it might have been learned via stochastic gradient descent, pre-wired, or obtained through an evolutionary algorithm.

Firstly, we train a deep Q-learning agent [23] on a primary task, so to have a network which we can extract features from. In the primary task, a robot arm tries to move its arm to the goal position while avoiding a rectangular obstacle. After training, we extract features from this network in three different ways and use them in a secondary task to assess the bootstrapping effect induced for the new task learning. As a baseline we also learn the new task from scratch without any transfer. So, overall, we have these cases:

(1) **Learning from scratch (`transfer:none`).** The new task is trained without any transfer to form a baseline for the next three transfer scenarios.

(2) **The last hidden layer activations (`transfer:full`).** This is one of the most frequently used methods for transferring visual features from networks that are trained on large-scale data sets [32] and serves as a reference for the next two methods.

(3) **Transforming the last hidden layer with PCA (`transfer:PCA`).** While the last layer activations provide useful information, it is mostly distributed in many units. PCA is especially useful for concentrating the distributed information into fewer units, effectively reducing the number of neurons, thus reducing the number of weights that needs to be optimized for learning the new task. Let us denote our dataset as $X = \{H_i\}_{i=1}^{N}$ where N is the number of robot movement trajectories arising from executing the primary task of the robot, and H_i is an $T_i \times D$ matrix where T_i is the number of timesteps in ith trajectory and D is the dimensionality of the last hidden layer. We concatenate each H_i from the first dimension so that X is a $(T_1 + T_2 + \cdots + T_N) \times D$ matrix. We want to find a projection $z = Xw$ such that $\mathrm{Var}(z)$ is maximized. This leads to the following objective:

$$\arg\max_{w} \quad \frac{w^T X^T X w}{w^T w} \tag{1}$$

which is the Rayleigh quotient, and the expression takes its maximum value when w is equal to the eigenvector with the largest eigenvalue of $X^T X$ [33].

(4) **Transforming the last hidden layer with SFA (`transfer:SFA`).** PCA puts an emphasis on the maximum information preserving units. On the other hand, SFA tries to minimize the time-derivative of the output signals. More specifically, SFA creates output features $z = Xw$ with the following objective and restrictions [22]:

$$\min \quad \mathbb{E}[\dot{z}_i^2] \tag{2}$$

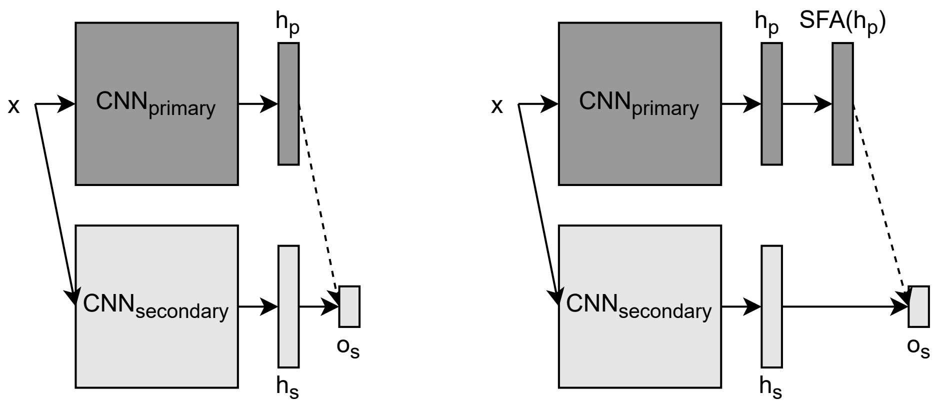
subject to

$$\mathbb{E}[z_i] = 0 \tag{3}$$

$$\mathbb{E}[z_i^2] = 1 \tag{4}$$

$$\mathbb{E}[z_i z_j] = 0 \quad \forall j < i \tag{5}$$

Here, Equations 3 and 4 prevents the trivial solution of a constant feature and also forces the output to be normalized. Equation 5 forces features to be orthogonal to each other. The solution can be found by first whitening the data, and then applying PCA to the time derivative of X. This method is shown to be useful for extracting the independent factors of an input signal.



(a) `transfer:full` (b) `transfer:sfa`

Figure 1. Different transfer methods are depicted. `transfer:pca` is identical to 1b except that the transformation is done with PCA, instead of SFA.

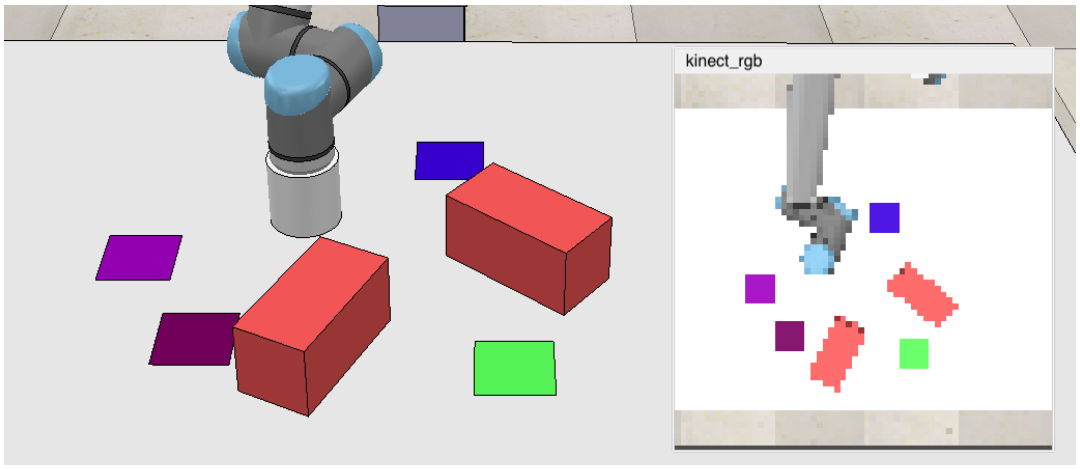
We transfer the information from a network with these three methods to a new task by concatenating the features to the input of the last layer as shown if Figure 1. In our experiments, we define the secondary task to be a reinforcement learning setup as well. The environment is similar to the primary task except that there is a different type of obstacle to avoid.

Note that both PCA and SFA defines an affine transformation of the last layer, and thus, does not bring any advantage in terms of function complexity. However, re-alignment of the feature space can greatly accelerate the speed of convergence with stochastic gradient descent [34]. Likewise, we expect that agents that use these transformations will exhibit a better performance with fewer samples.

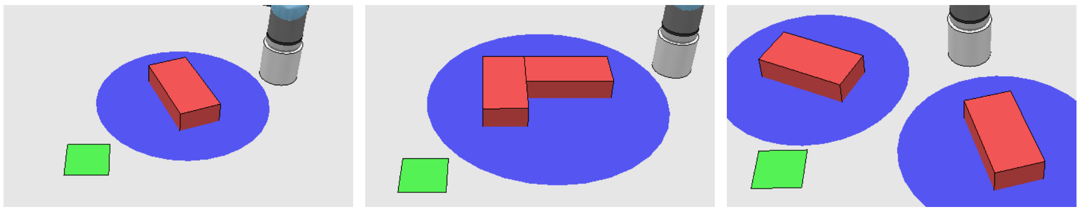
3. Experiment Setup

We perform our experiments on CoppeliaSim 4.2.0 simulator [35]. The experiment setup consists of a UR10 [36] robot arm with a solid cylinder attached to its end effector and a camera placed at 180cm over the table for a top-down visual perception. The robot arm can make planar movements in 8 different uniform directions in a 80cm $\times$ 100cm rectangular workspace. The camera input provides a 64 $\times$ 64 pixels colored image at each timestep (Figure 2). There are rectangular shaped and L-shaped obstacles, a goal marker, and distractor markers in the environment. The general overview of the setup is shown in Figure 2.

In this environment, we define two types of tasks: a primary task and a secondary task. We train a network to solve the primary task and then try to transfer the knowledge inside the network to the secondary task using different methods. In the primary task, the robot tries to move its end-effector to the goal position while avoiding a rectangular obstacle. There are two secondary tasks that differ by their obstacle types. In the first one, there is an L-shaped obstacle instead of a rectangular one, and in the second one, there are two rectangular obstacles. Secondary tasks are deliberately selected to be a linear increment from the first one to see if there are high-level features in the network such as an ‘obstacle detector’. If there are such high-level units, then the superposition of two obstacles would correspond to a new one, thus, a transfer would increase the speed dramatically. In both primary and secondary tasks, we included distractors that change colors and move in random directions with a constant speed to increase the difficulty of the tasks.



(a) The experiment setup.



(b) A rectangular obstacle (primary environment). (c) L-shaped obstacle (secondary environment). (d) Two rectangular obstacles (secondary environment).

Figure 2. The obstacles are red-colored, and the goal marker is green-colored. Distractors dynamically change their colors in the RGB range of [(0, 0, 0)-(255, 0, 255)]. The corresponding input from the camera is shown on the top-right inset. Right: The regions of penalty around obstacles are visualized in blue (see text).

The reward function for both environments is defined as follows:

$$R(t) = \begin{cases} 10 & \text{if} \quad \|x_{\text{tip}}(t) - x_{\text{goal}}\|_2 < 5\text{cm} \\ 10\Delta_t(x_{\text{tip}}, x_{\text{goal}}) & \text{else if} \quad \|x_{\text{tip}}(t) - x_{\text{obstacle}}(t)\|_2 > \tau \\ 10(\Delta_t(x_{\text{tip}}, x_{\text{goal}}) - \text{relu}(\Delta_t(x_{\text{tip}}, x_{\text{obstacle}}))) & \text{otherwise} \end{cases} \tag{6}$$

where τ is a threshold for obstacle proximity penalty and Δ_t denotes the change in the distance between robot end-effector and the target over consecutive time steps:

$$\Delta_t(x, y) = \|x(t-1) - y(t-1)\|_2 - \|x(t) - y(t)\|_2 \tag{7}$$

We set τ to 21cm for the primary environment and 28cm for secondary environments. The robot gets a penalty whenever it goes towards the obstacle when it is in the blue region visualized in Figure 2.

We use a convolutional neural network (CNN) architecture, where three convolutional layers followed by two fully-connected layers are employed in each architecture. Convolutional layers have 16, 32, and 64 channels successively. Each layer has a kernel size of 3×3 with a stride of 2 and a padding of 1. Fully-connected layers have 512 and 9 number of units. We concatenate the transferred features to the last hidden layer except the learning from scratch case. For example, if we transfer 100 slow features, the dimensionality of the last hidden layer becomes $512 + 100 = 612$. The outline of the architecture is shown in Figure B1. For PCA, we selected the first 100 components which cover more than 99% of the variation in the data. Likewise, we selected the first 100 slow features for a fair comparison.

We train each model as a deep Q-network [23] for 2000 episodes with a replay buffer of size 50,000. Each episode lasts for at most 200 time-steps, after then, the episode terminates. We used the reward function defined in Equation 6. The last fully-connected layer is used for the Q-value estimation. There are 8 different actions for 8 different directions and an additional action for no operation. For the optimization, we use Adam optimizer [37] with a learning rate of 0.001 with no learning rate decay.

4. Results

In this section, we first test the generalization performance attained by the transfer of features acquired with different methods to a new task in 4.1. Next, we analyze correlations between the proposed features and high-level task related features in Section 4.2. Lastly, in Section 4.3, we visualize the responses of neurons for varying inputs to get an insight about their functionalities.

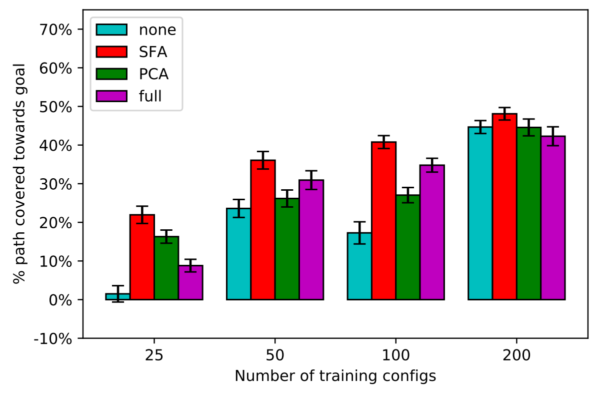
4.1. Transfer Performance

The aim of this experiment is to test the bootstrapping effect of using a previously learned representation on learning a similar task from scratch. To this end, we concatenate the activations of a previously trained network (described in the previous section) to the last hidden layer. This method is one of the basic transfer learning methods [32]. We compare the transfer of plain activations, PCA features, and SFA features. In addition, we train a network with no transfer to observe the bootstrapping effect.

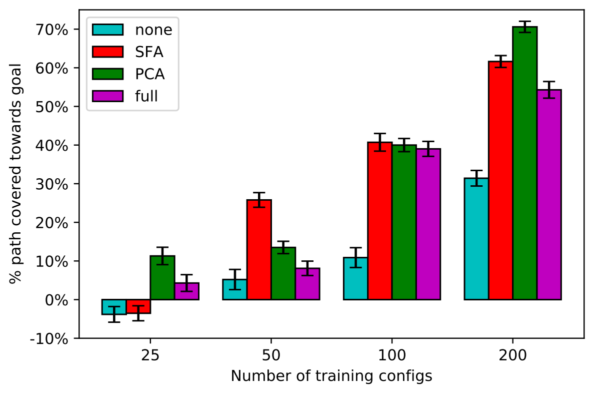
Each model is trained with 25, 50, 100, and 200 number of training configurations for 2000 episodes. Here, a training configuration refers to an environment setting (i.e. initial position of the objects). Each episode is initialized with a configuration sampled from this set. After convergence, we test each model at each 500 episodes on the previously unseen environment settings (i.e. configurations) to observe the generalization performance. For testing, we collect 100 runs with each model and calculate the average percentage of path covered towards the goal, and treat this estimate as one test result. The percentage of path covered towards the goal is calculated as follows:

$$\frac{\text{initial distance} - \text{final distance}}{\text{initial distance}} \times 100 \tag{8}$$

Here, the distance is the Euclidean distance from the robot's tip position to the goal position. The average of 15 different test results at best performing episodes (validated with 5 results) for each model is reported in Figure 3.



(a) Secondary environment with L-shaped obstacle

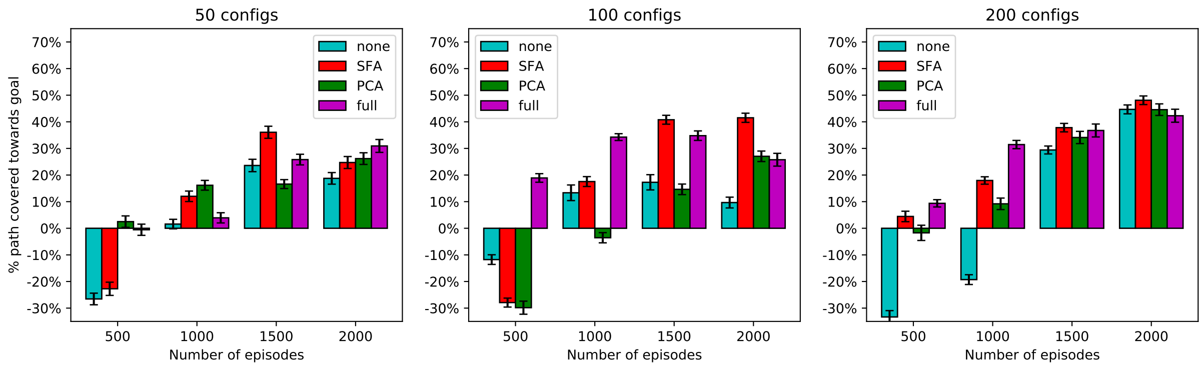


(b) Secondary environment with 2 rectangular obstacles

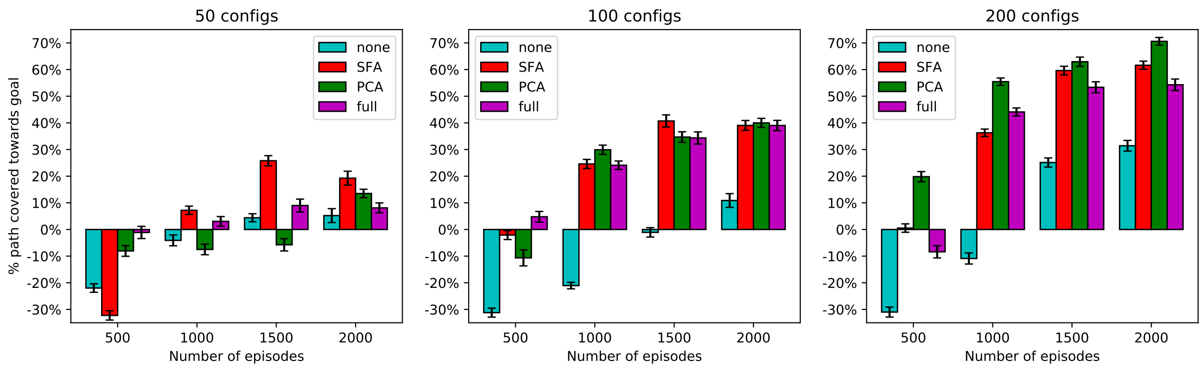
Figure 3. Generalization performance measured as distance covered towards the goal vs. number of configurations experienced during learning in the new task.

We first see that all transfer methods increase the performance even though they are more susceptible to overfitting with more parameters and fewer training configurations. This result suggests that there might be indeed high-level features in previously learned networks that help learn new skills from a small number of examples.

We observe that using `transfer:SFA` gives better performance compared to `transfer:PCA`, `transfer:full` and `transfer:none` except for 25 and 200 configurations in the secondary environment with 2 rectangular obstacles (Figure 3b). Welch's t-test is used to understand the significance of the results [38]. We use $p \leq 0.05$ as our threshold for significance. In Figure 3a for the environment with L-shaped obstacle, Welch's t-test shows a significant difference between `transfer:SFA` and `transfer:full` for 25, 100, and 200 configurations ($p = 0.0008$, $p = 0.022$, and $p = 0.041$, respectively), and almost significant difference for 50 configurations ($p = 0.082$). For 2 rectangular obstacles in Figure 3b, `transfer:SFA` is significantly better than `transfer:full` for 50 and 200 configurations ($p = 0.0001$ and $p = 0.0125$, respectively). The `transfer:PCA` method also has a competitive performance with less number of units when compared to `transfer:full`. Note that SFA and PCA features are 100 dimensional vectors while plain activations are 512 dimensional vectors. These results show that SFA indeed creates more condensed features that are appropriate for skill transfer. The bootstrapping effect fades away when we increase the number of training configurations to 200 configurations. This is an expected result as every model gets better when we increase the variation in the training set. However, the usage of low training configurations is desirable in many real world settings. On the other hand, there is still a performance gain even for 200 configurations for two rectangular obstacles. This might be due to the complexity of the task.



(a) Secondary environment with L-shaped obstacle



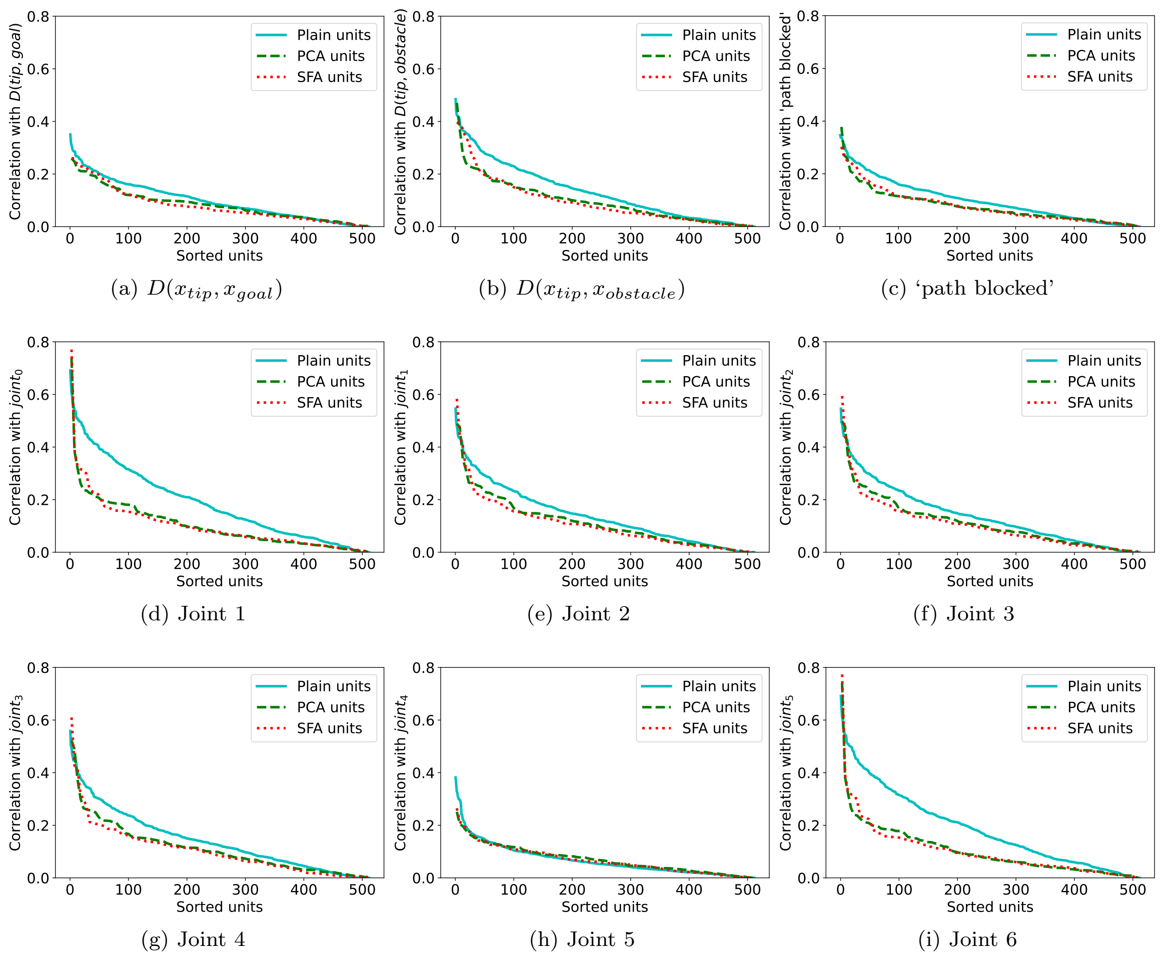
(b) Secondary environment with 2 rectangular obstacles

Figure 4. Test performance vs. number of training episodes for different methods and configuration numbers.

We also report the test results at different episodes in Figure 4. The figure suggests that using an additional set of features from a previously learned task helps generalization, especially when there are fewer number of configurations. We see that `transfer:sfa` performs on par with `transfer:none` in the initial stages of the training, then achieves the peak of its performance after 1500 episodes which is generally less than other methods. On the other hand, the contribution of `transfer:full` steadily increases. This might be due to the distribution of information in the neurons. Only a few SFA units contain useful information for the task, and when these neurons are discovered with gradient descent in later steps, the performance increases rapidly. Moreover, `transfer:sfa` learns faster compared to other methods for less number of configurations (reaching the peak around 1500 episodes for 50 and 100 configs in Figure 4), and also requires less parameter update compared to `transfer:full` as there are less number of units, which is an essential property for small, autonomous systems with no access to a graphics processing unit (GPU). See Appendix A for an empirical analysis regarding the computational needs for SFA.

4.2. Correlation with High-Level Task Properties

In this experiment, we investigate correlations between various units and high-level features. To this end, we freeze the network and run the policy for 100 episodes to collect hidden layer activations for PCA and SFA calculations. Then, we apply PCA and SFA with 100 components to the last hidden layer activations. To understand whether the transformed features capture any high-level features such as 'distance from the tip to the goal' or joint angles that are helpful to solve the task, we calculate the correlation between those high-level features and PCA/SFA features. We also compare the correlations between the last layer activations to see the effect of the these transformations on the correlations.



(a) $D(x_{tip}, x_{goal})$ (b) $D(x_{tip}, x_{obstacle})$ (c) 'path blocked'

(d) Joint 1 (e) Joint 2 (f) Joint 3

(g) Joint 4 (h) Joint 5 (i) Joint 6

Figure 5. Units are sorted by their correlation. SFA and PCA units are highlighted in red and green, respectively.

In Figure 5, each subplot shows the distribution of correlations for each different high-level feature. Units are sorted by their correlation value for better visualization. After the transformation with PCA and SFA, most of the units become less correlated while only a few of them become highly correlated (e.g. Figures 5d and 5e). This is probably due to PCA and SFA objectives force components to be decorrelated. From Figure 5, we can say that PCA and SFA capture the general neural response; they eliminate redundant neural responses and focus on the important ones by keeping the general response profile of the hidden layer. Out of the two, SFA unit distribution has more units with high correlation, especially for joint angles (Figures 5d-5g). This might be one of the reasons that the learning performance of `transfer:sfa` is comparable to `transfer:full`, and even better for the case of less number of configurations since redundant and possibly noisy variations are filtered out by SFA.

Table 1. Units that correlate the maximum with high-level features are reported. The numbers on the left and the right denotes the correlation and the index of the unit, respectively.

High-level feature	Full activations	PCA features	SFA features
Joint 1	0.69 / 297	0.74 / 2	**0.77** / 1
Joint 2	0.55 / 249	0.49 / 5	**0.58** / 3
Joint 3	0.55 / 249	0.50 / 5	**0.59** / 3
Joint 4	0.56 / 249	0.52 / 5	**0.61** / 3
Joint 5	**0.38** / 471	0.20 / 3	0.19 / 2
Joint 6	0.69 / 297	0.74 / 2	**0.77** / 1
$D(x_{\text{tip}}, x_{\text{goal}})$	**0.35** / 303	0.26 / 2	0.26 / 2
$D(x_{\text{tip}}, x_{\text{obstacle}})$	**0.48** / 147	0.47 / 3	0.40 / 3
‘path blocked’	0.35 / 147	**0.38** / 3	0.25 / 2

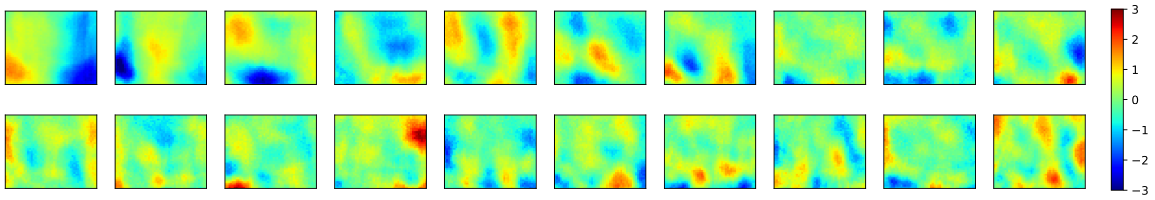
In Table 1, we select the most correlated units from each method and report their correlations. We see that all methods have high correlation with joint angles and slightly lower correlation with high-level features that relates the tip position to other objects. The high correlation with joint angles is an expected result as the agent should know about the location of the arm in order to navigate it to the goal position.

4.3. Visualizing Features

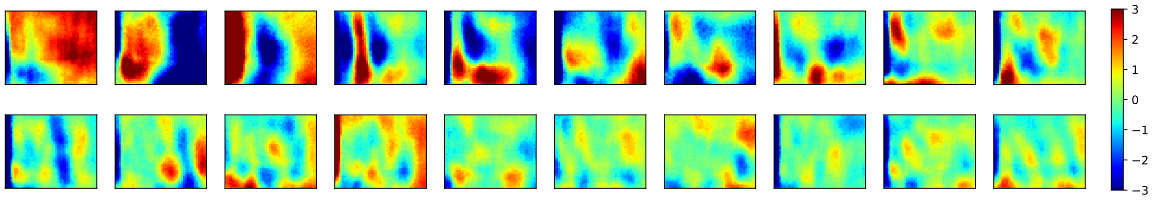
We created heatmaps for the responses of different SFA and PCA units to varying inputs. In Figure 6, each rectangle represents a unit’s average response to different tip locations. For example, for the top left unit in Figure 6a, the response is high when the tip is located around the bottom left corner of the table, and it is low around the bottom right corner. To create these figures, we averaged out over other variables (i.e. the goal position and the obstacle position). We see that both SFA and PCA units response to blob-like regions of the table. The first few units are very compact and can be partially treated as symbolic representations (e.g. in Figure 6a, the first and the third units detect the position in x-axis and y-axis, respectively.) As the unit number increases, these regions start to become scattered.

While the tip position is a useful information to solve the task, the agent should also know the relative position of the goal and the obstacle with respect to its tip to successfully navigate in the environment. To understand the responses for relative distance to the goal and the obstacle, we created heatmaps with the similar procedure in Figures 7 and 8. The center point of each subplot represents the zero distance (i.e. $x_{\text{goal}} - x_{\text{tip}} = (0, 0)$). In Figure 7, most of the units are scattered except the first few ones. This figure suggests that both SFA and PCA are not very successful at covering the relative goal position compactly; the information is distributed into many units as in plain activations. However, in Figure 8, we see that both methods generates neuron responses that are inactive when the tip is close to the obstacle (e.g. the second SFA unit and the first PCA unit).

To understand whether these two high-level information are present in the network prior to transformation, we visualized the neurons that correlate the most with the relative goal position and the relative obstacle position in Figure 9. For the bottom half in Figure 9a, we see that the most correlating neuron partially responds to x-axis location of the relative goal position. On the other hand, in Figure 9b, the response is similar to SFA and PCA units; it becomes inactive when distance is below some threshold.

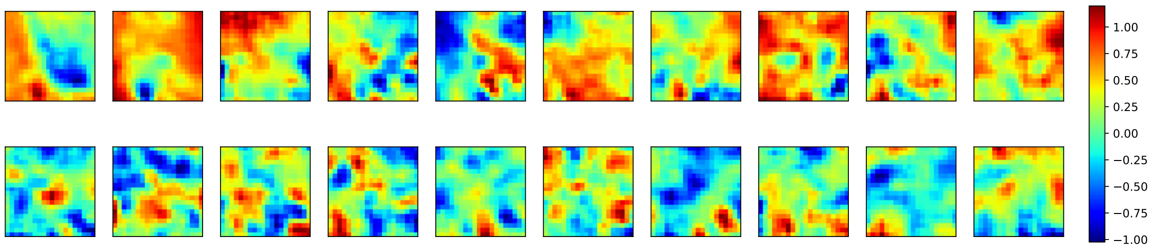


(a) SFA responses to the tip position.

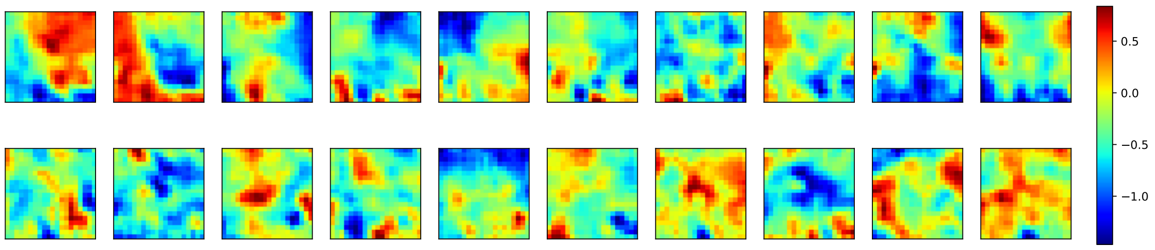


(b) PCA responses to the tip position.

Figure 6. First 20 SFA and PCA units' responses to different tip positions.

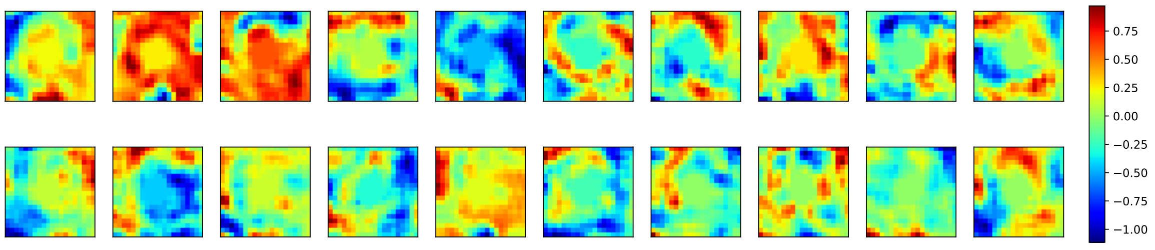


(a) SFA responses to the relative goal position.



(b) PCA responses to the relative goal position.

Figure 7. First 20 SFA and PCA features' responses to the relative goal position. The center point represents (0, 0), i.e. zero distance between the tip and the goal.


(a) SFA responses to the relative obstacle position.

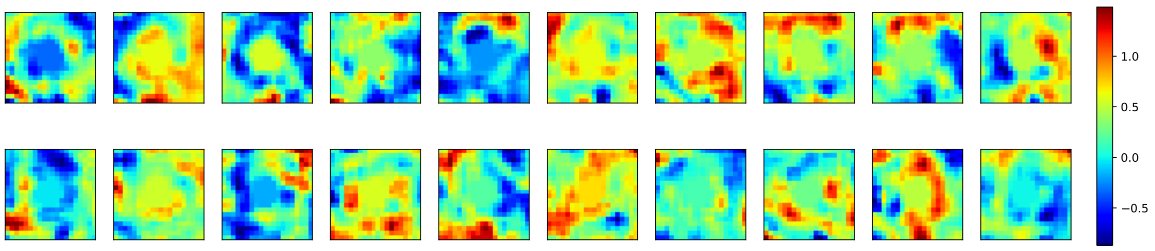
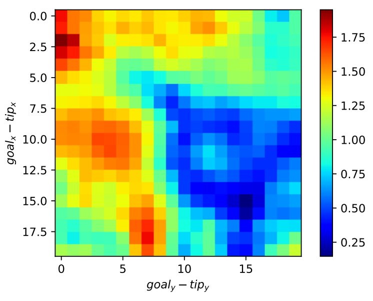
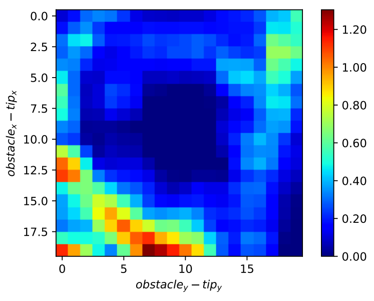

(b) PCA responses to the relative obstacle position.

Figure 8. First 20 SFA and PCA features' responses to the relative obstacle position. The center point represents (0, 0), i.e. zero distance between the tip and the obstacle.



(a) 303rd neuron



(b) 147th neuron

Figure 9. Neurons that correlate the most with 9a the relative goal position and 9b the relative obstacle position.

We conclude that even though the plain activations carry a distributed representation in general, they might represent some high-level information as a by-product which can be further processed and refined. We see that SFA and PCA transformations create high-level features to detect the tip position, however, this is not as compact as for the relative positions. Note that both SFA and PCA are unsupervised methods, not specifically tailored for creating more symbolic information. However, since they automatically create low-complexity signals in a principled way, they are of good candidates of a more complex system.

5. Conclusion

As the literature expands rapidly, the integration of architectures that are optimized for a specific scenario will gain more importance. In this work, we provide a principled way for transferring the existing knowledge in a network to new problems. Our experimental results show that applying SFA, an unsupervised method, to the last hidden layer of a trained network generates features that are useful for skill transfer. Transfer with SFA performs better with less number of units compared to transfer with the full layer. This is a desirable property for life-long learning systems because of its resource economy. We see that features that are generated from SFA are more interpretable and can be treated as quasi-symbolic information. Although not fully symbolic, they are of low complexity, i.e. the response of the units do not change abruptly as we change the input (especially the first few units). These might be precursors for fully symbolic systems because of their low complexity response to input. Moreover, due to its formulation, the components with the lowest eigenvalue will contain the most useful and symbol-like information. Therefore, one can always tune the number of units by simply picking the first k components with the lowest eigenvalues, a procedure familiar to PCA.

As a future work, we plan to extend the experimental setup to a more continual, open-ended, and general environment so that we can learn representations that are of progressive complexity to see the limits of the method. One interesting setup is to have a moving camera instead of a static one. Moving the camera would create a more realistic setup in which we see the world from a mobile embodiment. This in turn might enable the formation of more abstract representations that better describe the environment via measures such as distance between objects. Another possible future work is to integrate the slowness prior to the objective of learning to make the whole system an end-to-end architecture. The effectiveness of other dimensionality reduction methods such as independent component analysis for skill transfer can be also explored as a future work.

Acknowledgements

This research was supported by the International Joint Research Promotion Program of Osaka University under the project "Developmentally and biologically realistic modeling of perspective invariant action understanding" and TÜBİTAK (The Scientific and Technological Research Council of Turkey) ARDEB 1001 program (project number: 120E274).

References

[1] Konidaris G. On the necessity of abstraction. Current Opinion in Behavioral Sciences. 2019;29:1–7.
[2] Taniguchi T, Ugur E, Hoffmann M, et al. Symbol emergence in cognitive developmental systems: A survey. IEEE transactions on Cognitive and Developmental Systems. 2018;11(4):494–516.
[3] Ewert JP. Motion perception shapes the visual world of amphibians; 2004. p. 117–160.
[4] Petrick R, Kraft D, Mourao K, et al. Representation and integration: Combining robot control, high-level planning, and action learning. In: Proceedings of the 6th International Cognitive Robotics Workshop; 2008. p. 32–41.

[5] Sun R. Symbol grounding: A new look at an old idea. Philosophical Psychology. 2000;13(149–172).

[6] Konidaris G, Kaelbling LP, Lozano-Perez T. Constructing symbolic representations for high-level planning. In: 28th AAAI Conf. on AI; 2014.

[7] Asai M, Fukunaga A. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. In: Proceedings of the AAAI Conference on Artificial Intelligence; Vol. 32; 2018.

[8] Ahmetoglu A, Seker MY, Piater J, et al. DeepSym: Deep symbol generation and rule learning from unsupervised continuous robot interaction for planning. arXiv preprint arXiv:201202532. 2020; abs/2012.02532.

[9] Bengio Y. Learning deep architectures for AI. Now Publishers Inc; 2009.

[10] Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems. 2012;25:1097–1105.

[11] Sutskever I, Vinyals O, Le QV. Sequence to sequence learning with neural networks. arXiv preprint arXiv:14093215. 2014;abs/1409.3215.

[12] Silver D, Huang A, Maddison CJ, et al. Mastering the game of Go with deep neural networks and tree search. Nature. 2016;529(7587):484–489.

[13] Senior AW, Evans R, Jumper J, et al. Improved protein structure prediction using potentials from deep learning. Nature. 2020;577(7792):706–710.

[14] Weiss K, Khoshgoftaar TM, Wang D. A survey of transfer learning. Journal of Big data. 2016; 3(1):1–40.

[15] Vanschoren J. Meta-learning: A survey. arXiv preprint arXiv:181003548. 2018;abs/1810.03548.

[16] Thrun S, Mitchell TM. Lifelong robot learning. Robotics and Autonomous Systems. 1995;15(1):25–46. Available from: `http://www.sciencedirect.com/science/article/pii/092188909500004Y`.

[17] Rusu AA, Rabinowitz NC, Desjardins G, et al. Progressive neural networks. arXiv preprint arXiv:160604671. 2016;abs/1606.04671.

[18] Li Z, Hoiem D. Learning without forgetting. In: Leibe B, Matas J, Sebe N, et al., editors. Computer Vision – ECCV 2016. Springer International Publishing; 2016. p. 614–629.

[19] Ugur E, Piater J. Bottom-up learning of object categories, action effects and logical rules: From continuous manipulative exploration to symbolic planning. In: 2015 IEEE International Conference on Robotics and Automation (ICRA); IEEE; 2015. p. 2627–2633.

[20] Schmidhuber J. Driven by compression progress: A simple principle explains essential aspects of subjective beauty, novelty, surprise, interestingness, attention, curiosity, creativity, art, science, music, jokes. In: Pezzulo G, Butz MV, Sigaud O, et al., editors. Anticipatory Behavior in Adaptive Learning Systems. Springer Berlin Heidelberg; 2008. p. 48–76.

[21] Wolff JG. Information compression, multiple alignment, and the representation and processing of knowledge in the brain. Frontiers in Psychology. 2016;7:1584–1584. Available from: `https://pubmed.ncbi.nlm.nih.gov/27857695https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5093141/`.

[22] Wiskott L, Sejnowski TJ. Slow feature analysis: Unsupervised learning of invariances. Neural computation. 2002;14(4):715–770.

[23] Mnih V, Kavukcuoglu K, Silver D, et al. Playing atari with deep reinforcement learning. arXiv preprint arXiv:13125602. 2013;abs/1312.5602.

[24] Konidaris G, Kaelbling LP, Lozano-Perez T. From skills to symbols: Learning symbolic representations for abstract high-level planning. Journal of Artificial Intelligence Research. 2018;61:215–289.

[25] Santello M, Flanders M, Soechting JF. Postural hand synergies for tool use. Journal of Neuroscience. 1998;18(23):10105–10115.

[26] Tripathi GN, Wagatsuma H. PCA-based algorithms to find synergies for humanoid robot motion behavior. International Journal of Humanoid Robotics. 2016;13(02):1550037.

[27] Promsri A, Federolf P. Analysis of postural control using principal component analysis: The relevance of postural accelerations and of their frequency dependency for selecting the number of movement components. Frontiers in Bioengineering and Biotechnology. 2020;8:480.

[28] Schaal S. Dynamic movement primitives-a framework for motor control in humans and humanoid robotics. In: Adaptive Motion of Animals and Machines. Springer; 2006. p. 261–280.

[29] Chen N, Bayer J, Urban S, et al. Efficient movement representation by embedding dynamic movement primitives in deep autoencoders. In: 2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids); IEEE; 2015. p. 434–440.

[30] Kompella VR, Luciw M, Schmidhuber J. Incremental slow feature analysis. In: Twenty-Second International Joint Conference on Artificial Intelligence; Citeseer; 2011.

[31] Legenstein R, Wilbert N, Wiskott L. Reinforcement learning on slow features of high-dimensional

input streams. PLoS Comput Biol. 2010;6(8):e1000894.

[32] Goodfellow I, Bengio Y, Courville A, et al. Deep learning. Vol. 1. MIT press Cambridge; 2016.

[33] Trefethen LN, Bau III D. Numerical linear algebra. Vol. 50. Siam; 1997.

[34] LeCun YA, Bottou L, Orr GB, et al. Efficient backprop. In: Neural networks: Tricks of the trade. Springer; 2012. p. 9–48.

[35] Rohmer E, Singh SPN, Freese M. CoppeliaSim (formerly V-REP): a versatile and scalable robot simulation framework. In: Proc. of The International Conference on Intelligent Robots and Systems (IROS); 2013. www.coppeliarobotics.com.

[36] Universal Robots. The UR10 collaborative industrial robot [https://www.universal-robots.com/products/ur10-robot]; 2012. Online; accessed 10 September 2020.

[37] Kingma DP, Ba J. Adam: A method for stochastic optimization. arXiv preprint arXiv:14126980. 2014;abs/1412.6980.

[38] Welch BL. The generalization of student's' problem when several different population variances are involved. Biometrika. 1947;34(1/2):28–35.

Appendix A. Computation Time of Slow Features

The computation of slow features is a one-time process that depends on the singular value decomposition of the trajectory array. For 200 trajectories with a total of 20961 timesteps, the computation of slow features takes 0.36 seconds on an Intel i7-8700K CPU. The only computational overhead after extracting slow features is a matrix multiplication and addition. For comparison, we analyzed the inference time of `transfer:sfa` and `transfer:full`. For `transfer:sfa`, 10,000 forward iterations took 6.409 ± 0.022 seconds on average out of 20 runs. If we convert these numbers to frequencies, then the 99% confidence interval is 1544-1576Hz. On the other hand, the same experiment took 6.223 ± 0.019 seconds for `transfer:full` which translates to 1592-1621Hz. These experiments are done on an Intel i7-8700K CPU. On a comparably older CPU (i5-5257U), `transfer:sfa` has a confidence interval of 713-738Hz, and `transfer:full` has 741-766Hz. The frequency drop due to the additional computation is tolerable since the frequency is still higher than the maximum control frequency of the robot (500Hz for UR10). Moreover, SFA only creates an affine transformation, which can be integrated with the learned last layer after the training to remove the additional layer. We did not follow such optimizations in our current implementation for the sake of easy experimentation.

Appendix B. Network Architecture

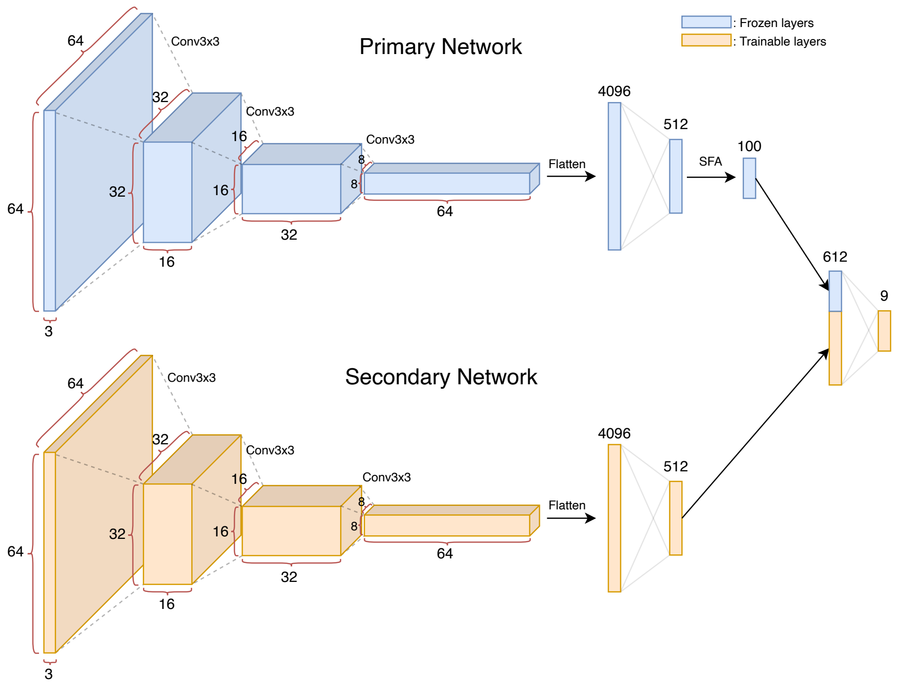


Figure B1. The transfer network architecture for the `transfer:sfa` method. `transfer:pca` is the same except there is a PCA transformation instead of SFA.