Multi-Task Reinforcement Learning with Context-based Representations

Shagun Sodhani [1] **Amy Zhang** [1 2 3] **Joelle Pineau** [1 2 3]

Abstract

The benefit of multi-task learning over single-task learning relies on the ability to use relations across tasks to improve performance on any single task. While sharing representations is an important mechanism to share information across tasks, its success depends on how well the structure underlying the tasks is captured. In some real-world situations, we have access to *metadata*, or additional information about a task, that may not provide any new insight in the context of a single task setup alone but inform relations across multiple tasks. While this metadata can be useful for improving multi-task learning performance, effectively incorporating it can be an additional challenge. We posit that an efficient approach to knowledge transfer is through the use of multiple context-dependent, composable representations shared across a family of tasks. In this framework, metadata can help to learn interpretable representations and provide the context to inform which representations to compose and how to compose them. We use the proposed approach to obtain state-of-the-art results in Meta-World, a challenging multi-task benchmark consisting of 50 distinct robotic manipulation tasks.

1. Introduction

Reinforcement learning (RL) has made large strides over the last several years (Mnih et al., 2013; Silver et al., 2017; Radford et al., 2019). While these improvements are significant, much of this success has been restricted to the single task setting (Teh et al., 2017; Yu et al., 2020b). In contrast, humans are adept at multi-tasking by acquiring new skills and composing known skills to solve complex tasks (Parascandolo et al., 2018; Allen et al., 2020). For autonomous agents to adapt effectively in the real world, they need to master multiple tasks in a sample efficient manner. Multi-task reinforcement learning (MTRL) is a promising approach to train effective real-world agents (Tanaka & Yamamura, 2003; Rusu et al., 2016; Borsa et al., 2016; Rajeswaran et al., 2016; El Bsat et al., 2017; Andreas et al., 2017; Igl et al., 2020; D'Eramo et al., 2020; Yu et al., 2020a).

One limitation of existing MTRL methods is the inability to leverage side information (or *metadata*), like the description of a task, to learn generalizable skills and transfer common knowledge across tasks. Such metadata is often available in real-world tasks but is not leveraged in the typical MTRL setting. This metadata can take the form of natural language task descriptions or instructions, which are often incorporated only for human usage to communicate information about tasks. These natural language descriptions have been utilized in single-task RL setups like goal-oriented RL (Chevalier-Boisvert et al., 2019; Luketina et al., 2019; Jiang et al., 2019). We show that this information can also be used to learn context-dependent, composable representations shared across a family of tasks, with the metadata acting as the context. Specifically, we show that the metadata can be used to learn a prior over a collection of encoders and can be leveraged to select the encoder(s) for any given task. The learned encoders can specialize to different aspects of the tasks, allowing for more efficient sharing of knowledge across the tasks. An important additional benefit is that the learned representations are interpretable.

The default formulation of MTRL environments is as a family of Markov Decision Processes (MDPs) (Bellman, 1957; Puterman, 1995). This framework does not provide a natural way to incorporate metadata. We therefore turn to the

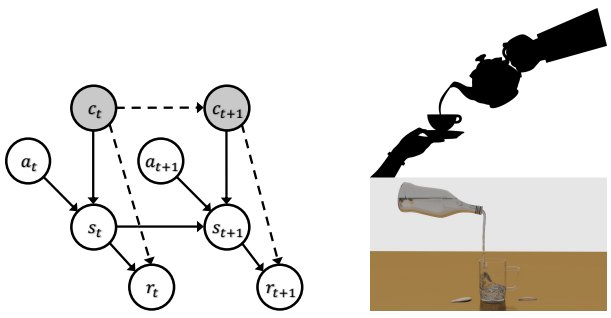


Figure 1. Graphical model of the Contextual MDP setting (left). Example of different contexts that can affect the task of pouring (right).

[1]Facebook AI Research [2]Mila [3]McGill University. Correspondence to: Shagun Sodhani <sodhani@fb.com>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

Contextual Markov Decision Process formulation (Hallak et al., 2015) to define our setting (Figure 1). We assume access to a *context* (the metadata) that contains additional task-specific information which is easily available and can be used to improve per-task performance. This metadata is more useful in a multi-task setting where it can be used to infer the relationship between different tasks coming from a specified family. As an example, given two contexts, "pour tea into my cup" and "pour water into the mug kept on the table", the agent can infer the relations between the two tasks and identify the common contexts (and subtasks) like "pour" (Figure 1, right). Such context is less likely to be useful to accelerate learning in the single task setup.

Sharing representations across tasks can be an effective way for RL agents to transfer knowledge across tasks. However, not all knowledge transfers are *positive*. Some aspects of a given task could be meaningful only for that task and irrelevant (or even detrimental) for the other tasks. This effect is commonly known as *negative interference* (Parisotto et al., 2015; Teh et al., 2017). In general, choosing which information/knowledge to transfer across the tasks, or even deciding which tasks should be learned together is an open problem (Standley et al., 2019). We posit that in the CMDP setup, the context can be used to inform which representations and information should be shared across the tasks.

In this work, we propose a novel approach for the contextual multi-task RL setting where we encode an input observation into multiple representations (corresponding to different skills or objects) using a *mixture of encoders*. The learning agent can use the context to decide which representation(s) it uses for any given task, giving the agent a fine-grained control over what information is shared across tasks, thus alleviating *negative interference*. We call our method **C**ontextual **A**ttention-based **RE**presentation learning, or CARE for short.

Key contributions of this work are 1) a simple, yet effective, way to incorporate task metadata, or contextual information, to improve sample efficiency and asymptotic performance, 2) a new representation learning algorithm for MTRL that leverages a mixture of interpretable encoders which encodes task and object-specific information about each state space, and 3) state-of-the-art results on a challenging multi-task RL benchmark, Meta-World (Yu et al., 2020b). For example videos see `https://sites.google.com/view/mtrl-care`. The implementation of the algorithms is available at `https://github.com/facebookresearch/mtrl`.

2. Preliminaries

A **Markov Decision Process** (MDP) (Bellman, 1957; Puterman, 1995) is defined by a tuple $\langle \mathcal{S}, \mathcal{A}, R, T, \gamma \rangle$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, $T : \mathcal{S} \times \mathcal{A} \to Dist(\mathcal{S})$ is the environment transition probability function, and $\gamma \in [0, 1)$ is the discount factor. At each time step, the learning agent perceives a state $s_t \in \mathcal{S}$, takes an action $a_t \in \mathcal{A}$ drawn from a policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, and with probability $T(s_{t+1}|s_t, a_t)$ enters next state s_{t+1}, receiving a numerical reward R_{t+1} from the environment. The value function of policy π is defined as: $V_\pi(s) = E_\pi[\sum_{t=0}^{\infty} \gamma^t R_{t+1} | S_0 = s]$. The optimal value function V^* is the maximum value function over the class of stationary policies.

Contextual Markov Decision Processes were first proposed by Hallak et al. (2015) as an augmented form of Markov Decision Processes that utilize *side information* as a form of context, similar to in contextual bandits.

Definition 1 (Contextual Markov Decision Process)**.** *A contextual Markov decision process (CMDP) is defined by tuple $\langle \mathcal{C}, \mathcal{S}, \mathcal{A}, \mathcal{M} \rangle$ where $\mathcal{C}$ is the context space, $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space. $\mathcal{M}$ is a function which maps a context $c \in \mathcal{C}$ to MDP parameters $\mathcal{M}(c) = \{R^c, T^c\}$.*

Contexts can be applied in the multi-task setting, where we define a *family* of MDPs where each MDP has a shared state space $\mathcal{S}$. However, the agent only has access to a partial state space $\mathcal{S}^c$ (either low-dimensional or rich, like pixels) that is a subspace of the original state space $\mathcal{S}$, focusing only on objects relevant to the task at hand. Different MDPs can involve different combinations of objects and skills, hence the state space $\mathcal{S}^c$ and reward function R^c can differ across MDPs. However, the objects are *shared* across tasks, i.e., the object-specific dynamics remain consistent across tasks[1]. In this work, we focus on the low-dimensional setting where $\mathcal{S}^c$ is a strict subset of the dimensions in $\mathcal{S}$.

We attach this additional relaxation of the original CMDP definition to define a new setting, a Block Contextual MDP (BC-MDP)[2] (Du et al., 2019; Zhang et al., 2020):

Definition 2 (Block Contextual Markov Decision Process)**.** *A block contextual Markov decision process (BC-MDP) is defined by tuple $\langle \mathcal{C}, \mathcal{S}, \mathcal{A}, \mathcal{M}' \rangle$ where $\mathcal{C}$ is the context space, $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space. $\mathcal{M}'$ is a function which maps a context $c \in \mathcal{C}$ to MDP parameters and observation space $\mathcal{M}(c) = \{R^c, T^c, \mathcal{S}^c\}$.*

This brings us to our setting for evaluation, **Meta-World** (Yu et al., 2020b)[3], as a natural instantiation of a BC-MDP. Meta-World proposes a benchmark for meta-RL

[1] Note that the dynamics T^c for each MDP are still different because the state spaces are different.

[2] This is not the partial observability setting because we have access to a task id or description that uniquely identifies the task, and therefore what objects are referred to by the task-specific state space.

[3] We use the following commit from MetaWorld for our experiments: af8417bfc82a3e249b4b02156518d775f29eb289

and multi-task RL, consisting of 50 distinct robotics manipulation tasks, with some example tasks shown in Figure 2. The state space across all tasks is of the same dimensionality, but those dimensions have different semantics across tasks. For example, the same subset of dimensions can refer to a goal position in one task and some object's position in the other task. We assume access to a family of N MDPs consisting of potentially different reward functions and state spaces of consistent dimensionality, but not necessarily with the same semantic meaning. Unlike previous works, which generally focus on very narrow task distributions, Meta-World provides a diverse task distribution with 50 different tasks involving objects like doors, cups, windows, drawers, etc. and skills like push, pull, open, close, etc. while still providing a shared state and action space. Evaluating on a broad task distribution provides a better estimate of the generalization capabilities of MTRL algorithms.

3. A Method for Learning Contextual Attention-based Representations

In the multi-task reinforcement learning setting described in Section 2, we propose to factorize the state representation into sub-components that are common across the MDPs within a defined BC-MDP family. While each task has its own state space, there are commonalities across tasks. For example, this can take the form of objects like "drawer" or "door", and skills like "open" or "close". In this example, the goal would be to disentangle the state into object-specific and skill-specific representations. We train a universal policy (i.e. one shared policy for all the tasks) that uses the task-specific metadata (or context) to choose a functional representation (e.g. of objects and skills) for any given task. We introduce this compositionality by incorporating a mixture of encoders where different encoders specialize to different aspects for the given family of tasks. In our example, given three tasks "open a door", "open a drawer", and "close a drawer", the encoders could specialize to "open" and "close" skills and "door" and "drawer" objects.

In this section, we describe how we use the metadata to train the different components in CARE. It is important to note that the role of CARE is to learn a representation that enables the incorporation of metadata and functional abstraction. For end-to-end reinforcement learning, it must be paired with a policy optimization algorithm. In the scope of this work, we use Soft Actor-Critic (SAC, (Haarnoja et al., 2018)), but CARE can be paired with any policy optimization method.

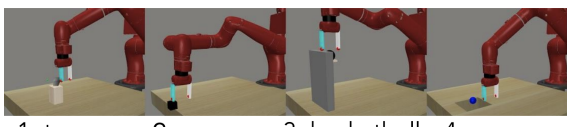


Figure 2. Some tasks from Meta-World. Image taken from Yu et al. (2020b).

3.1. Incorporating Information from Metadata

In the BC-MDP setting, the different tasks share objects and skills across the family of MDPs, but the state spaces across tasks are context-dependent, and therefore not the same, $\mathcal{S}^c$. Our goal is to reconstruct the universal state space $\mathcal{S}$. Of course, we do not have access to the true state space, but a useful inductive bias for the learning agent would be to learn composable representations for objects and attend over these representations for different tasks. One major challenge to this approach is that knowing which objects are relevant for each task requires object-level supervision, which is not commonly available. We propose to sidestep this problem by conditioning the attention on the task context, which is modelled using the easily available task metadata or description. Note that this metadata can be high-level, under-specified, and unstructured. It does not have to explain "how to perform the task"; it can simply describe the task. Even in cases where this metadata is not readily available, it can be easily constructed. An example of a task description from Meta-World could be "Reach a goal position". Another example is MuJoCo tasks from Deepmind Control (Tassa et al., 2018) and OpenAI Gym (Brockman et al., 2016), where humans identify the tasks by descriptive names like "HalfCheetah Run" and "Maze Solver Ant", as opposed to task ids $\{1, 2, ..., n\}$.

Given such a high-level task description, we focus on the case where the task context is captured using pre-trained language models. Specifically, we use the Roberta model (Liu et al., 2019b) to obtain a 768-dimensional representation of the task description. This representation is projected to a lower-dimensional space using feedforward layers, and the resulting representation is denoted as the context $z_{context}$. The context is used to condition the different components of the policy, as described below.

3.2. Contextual Attention based Representations

We posit that a useful inductive bias for the training agent is to learn contextual representations for different tasks by learning multiple representations and attending over those representations using the task context. Given N tasks, we use a mixture of k encoders to learn k state-representations. Here, k is a hyperparameter and, in practice, it is much smaller than the number of tasks in the family of MDPs. Note that unlike some work on object-oriented learning, we do not assume access to *privileged* information in terms of which objects are present in the input observation or useful to encode in a task. Moreover, while our design

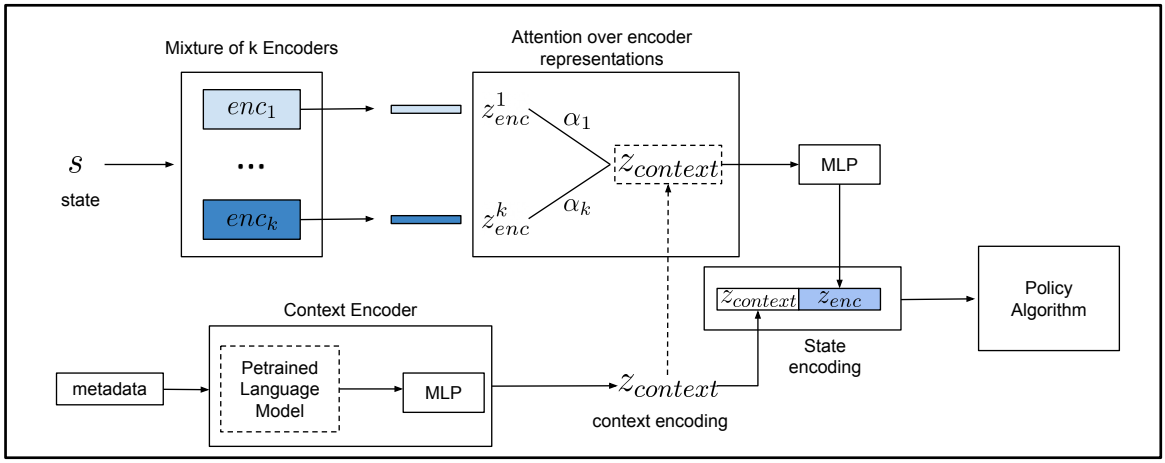


Figure 3. Architecture of the CARE model: Given an input state s, an mixture of k encoders is used to compute k encodings, denoted as $z^i_{enc} \forall i \in \{1, \cdots, k\}$. Given the metadata, a pretrained language model (followed by a feedforward network) is used to encode the task description as a real valued *context* vector $z_{context}$. This context is used to compute an attention score between z^i_{enc} and $z_{context}$. The attention scores are normalized (to sum to 1) and are denoted as $\alpha_i \forall i \in \{1, \cdots, k\}$. The attention scores are used to compute a weighted sum of the encoders' representations and is denoted as z_{enc}. It is then concatenated with the context to obtain the state encoding z_s that is given as input to the policy network. Dashed lines indicate that gradient does not flow through those components or computations.

Algorithm 1 CARE algorithm for the multi-task RL setting.

Require: SAC Components
Require: Context Encoder Network C
Require: k Encoders $E_1, \ldots, E_k$
for each timestep $n = 1..N$ **do**
  for each task $\mathcal{T}_i$ **do**
    $z^i_{context} = C(metadata_i)$
    $z^j_{enc} = E_j(s^i_n), \quad \forall j \in \{1, \ldots, k\}$
    $\bar{z}^i_{context} = stopgrad(z^i_{context})$
    $\alpha_j = softmax(z^j_{enc} \cdot \bar{z}^i_{context}) \quad \forall j \in \{1, ..., k\}$
    $z^i_{enc} = MLP(\sum_{j \in \{1,\ldots,k\}} z^j_{enc} \times \alpha_j)$
    $z^i = z^i_{context} :: z^i_{enc}$ (input to the SAC algorithm)
    UPDATESAC($\mathcal{D}, z^i$). Refer to Algorithm 2.
    UPDATECONTEXTENCODER($\mathcal{D}, z^i$). Refer to Algorithm 3.
    UPDATEMIXTUREOFENCODERS($\mathcal{D}, z^i$). Refer to Algorithm 4.
  end for
end for

encourages the specialization of different encoders to combinations of different objects and skills, we note that this setup is incorporating a softer inductive bias than object-oriented learning (Greff et al., 2019; Locatello et al., 2020). Specifically, while we are conditioning the policy on object representations, we do not explicitly model the interactions between the encoders/objects, as is done in recent works like Goyal et al. (2019). We factorize the representation in terms of reusable components, unlike methods that first perform object detection and then model higher-order interactions between the objects using attention-based mechanisms or graph neural networks (Kipf et al., 2018; Pathak et al., 2019; Li et al., 2020). We see our design choice as a trade-off between imposing more useful structure on the algorithm versus requiring less access to privileged information.

Given the k encoders, we compute the k representations $z^i_{enc} \forall i \in \{1, \cdots, k\}$. Given the context, $z_{context}$, we compute the normalized soft-attention weights for the encoder representations (denoted as $\alpha_i \forall i \in \{1, \cdots, k\}$). We pool the k encoder representations into a fixed-size representation by performing a weighted sum using the **soft-attention** weights. The resulting encoder representation (z_{enc}) is computed as $z_{enc} = \sum_i^k \alpha_i \times z^i_{enc}$. We concatenate the encoder representation (z_{enc}) with the context representation ($z_{context}$) to obtain the state encoding (z_s). This state encoding is used as an input to the policy network, and the entire setup is trained end-to-end. Note that the language model is not updated during training and the context representation $z_{context}$ is detached from the computation graph before computing the attention weights. $z_{context}$ is updated using the policy loss directly, as it is a part of the state encoding.

3.3. Downstream Evaluation

We use **Soft Actor-Critic** (SAC,(Haarnoja et al. (2018) for downstream evaluation of the learned representations. SAC is an off-policy actor-critic method that uses the maximum entropy framework for soft policy iteration. At each iteration, SAC performs soft policy evaluation and improvement steps. For more details on SAC, refer Haarnoja et al. (2018).

The overall algorithm is described in Algorithm 1, with the sub-function details available in the Appendix (Algorithms 2, 3, 4). We note that steps 3 to 11 can be run

concurrently for multiple tasks (as is done in our implementation). The architecture diagram is shown in Figure 3 and the Appendix contains additional implementation details (Appendix A) and hyper-parameters (Appendix B).

4. Experiments

We now empirically evaluate the effectiveness of the proposed CARE model on Meta-World (Yu et al., 2020b) – a multi-task RL benchmark with 50 tasks. We design our experiments to answer the following questions: **i)** Is learning contextual attention-based representations an effective mechanism for knowledge transfer in multi-task RL? Does it perform better than methods that do not utilize this context? **ii)** Is the metadata useful only when learning compositional representations? **iii)** Does the metadata help to learn factored, specialized representations? **iv)** Does the metadata help in zero-shot generalization to unseen environments?

4.1. How CARE compares to existing MTRL baselines

Existing works in multi-task RL come in two flavours: **i)** Extend a single task RL baseline for multi-task by using task-specific parameters (like one policy-head-per-task or per-task entropy regularization). These approaches are generally algorithm-agnostic. **ii)** Specialized multi-task algorithms. In this second category, we compare against the PCGrad (Yu et al., 2020a) algorithm which is specifically proposed for the Meta-World benchmark and achieves state-of-the-art results, outperforming multi-task algorithms like GradNorm (Chen et al., 2018) and Orthogonal Gradients (CosReg) (Suteu & Guo, 2019). We also compare with the Soft Modularization approach (Yang et al., 2020) that performs routing in a shared policy network to learn different policies for different tasks and provides state-of-the-art results for Meta-World benchmark. Additionally, we compare with a popular and general-purpose conditioning method called FiLM (Perez et al., 2018). We use FiLM layers to condition the encoder on the context (generated using a context encoder, just like in CARE). While FiLM is not a multi-task RL algorithm, it is used effectively in the language-conditioned RL setups (Chevalier-Boisvert et al., 2019; Zhong et al., 2020).

The evaluation performance of the agent is computed as follows: At regular intervals, the agent is evaluated 5 times on each test environment, and the mean of the 5 runs is taken as the success rate for the corresponding environment. These success rates are averaged across the environments to obtain the mean success rate at every interval. A time-series of the mean success rates is obtained by evaluating the agent at regular intervals. The agent is trained for multiple seeds (10 in our case) resulting in 10 different time-series (one per seed) of mean success rates. These 10 time-series are averaged to compute the mean of the success rates (mean over the seeds). The best mean (across the time-series) is reported as the evaluation performance.

Table 1. Evaluation performance on the **MT10** test environments, after training for **2 million** steps (for each environment). Results are averaged over 10 seeds. ME = Mixture of Encoders. CARE's improvement is statistically significant for baselines with *.

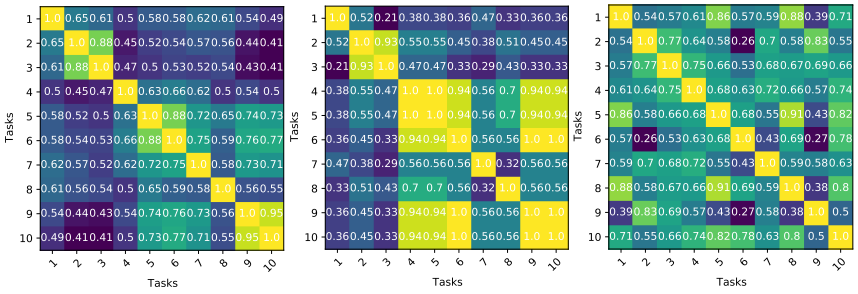
Agent	success (mean $\pm$ stderr)
Multi-task SAC (Yu et al., 2020b) *	0.49 ± 0.073
Multi-task SAC + Task Encoder *	0.54 ± 0.047
Multi-headed SAC (Yu et al., 2020b) *	0.61 ± 0.036
PCGrad (Yu et al., 2020a) *	0.72 ± 0.022
Soft Modularization (Yang et al., 2020)	0.73 ± 0.043
SAC + FiLM (Perez et al., 2018)	0.75 ± 0.037
SAC + Metadata + ME (**CARE**)	$\mathbf{0.84 \pm 0.051}$
One SAC agent per task (upper bound)	0.90 ± 0.032

We highlight some challenges in comparing the performance of different models on the Meta-World suite. Generally, RL agents are evaluated for continuous-valued episodic rewards (Brockman et al., 2016; Tassa et al., 2018) while Meta-World uses a binary-valued success signal. While the use of success signals is not entirely unheard of (Chevalier-Boisvert et al., 2019), we find that with Meta-World, we can improve the agent's performance just by increasing the frequency of evaluation (as shown in Table 19 in Appendix). Evaluating the agent more often makes it more likely for the agent to solve a given task, making it harder to compare results across different works. For example, consider the extreme case where the agent is evaluated after every single update. Since we report the best (max) of the mean(success), evaluating more frequently could improve the performance since we are computing the max over a larger set. Thus, evaluation frequency acts like an implicit hyperparameter. We control for this issue in our setup by evaluating every agent at a fixed frequency (once every 10K environment steps, per task). Second, we find that the number of seeds (for evaluation) plays a big role in a model's reported performance. For example, we report that for MT10, the standard Multi-headed SAC achieves a mean success of 61% (10 seeds), whereas Yu et al. (2020b) reports a success rate of 88% (1 seed), leading to a 44% change in performance. Some other models (Yang et al., 2020) use just 3 seeds. We account for the stochasticity of the evaluation process and ensure a fair comparison of all the models by running all the experiments with 10 seeds. We also report if our model's improvements are statistically significant or not. Additional details about testing for statistical significance can be found in Appendix D.

We use the same setup as other works. At regular interval, agent is evaluated on all envs

1. Reach a goal position.
2. Push the puck to a goal.
3. Pick and place a puck to a goal.
4. Open a door with a revolving joint.
5. Open a drawer.
6. Push and close a drawer.
7. Press a button from the top.
8. Insert a peg sideways.
9. Push and open a window.
10. Push and close a window.

(a)



(b) (c) (d)

Figure 4. (a): Task descriptions for MT10. (b) Cosine similarity (for MT10) between the pretrained task embeddings. (c) Cosine similarity (for MT10) between the context representations using CARE model with $k = 6$ encoders. (d) Cosine similarity (for MT10) between the context representations without using metadata with $k = 6$ encoders. Structure across tasks is clearly exhibited in (b) and (c), but not in (d), which has no access to metadata.

Table 2. Evaluation performance on the **MT10** test environments, after training for **100 thousand** steps (for each environment). Results averaged over 10 seeds. ME = Mixture of Encoders. CARE's improvement is statistically significant for baselines with *.

Agent	success (mean $\pm$ stderr)
Multi-task SAC (Yu et al., 2020b) *	0.13 ± 0.022
Multi-task SAC + Task Encoder *	0.14 ± 0.012
Multi-headed SAC (Yu et al., 2020b) *	0.17 ± 0.033
PCGrad (Yu et al., 2020a) *	0.20 ± 0.032
Soft Modularization (Yang et al., 2020)	0.33 ± 0.036
SAC + FiLM (Perez et al., 2018) *	0.27 ± 0.037
SAC + Metadata + ME (**CARE**)	$\mathbf{0.36 \pm 0.035}$

Table 3. Evaluation performance on the **MT50** test environments, after training for **2 million** steps (for each environment). Results are averaged over 10 seeds. ME = Mixture of Encoders. CARE's improvement is statistically significant for baselines with *.

Agent	success (mean $\pm$ stderr)
Multi-task SAC (Yu et al., 2020b) *	0.36 ± 0.013
Multi-task SAC + Task Encoder *	0.40 ± 0.024
Multi-headed SAC (Yu et al., 2020b)	0.45 ± 0.064
PCGrad (Yu et al., 2020a)	0.5 ± 0.017
Soft Modularization (Yang et al., 2020)	0.5 ± 0.035
SAC + FiLM (Perez et al., 2018) *	0.40 ± 0.012
SAC + Metadata + ME (**CARE**)	$\mathbf{0.54 \pm 0.031}$
One SAC agent per task (upper bound)	0.74 ± 0.041

We note that these challenges are not inherent limitations of Meta-World and issues related to seeds affect the evaluation of RL algorithms in general (Henderson et al., 2017). However, we believe that some of these challenges can be alleviated by standardizing the evaluation protocol. Given the usefulness of Meta-World as a multi-task RL benchmark, these challenges should be highlighted to ensure they are considered in the subsequent works.

Meta-World benchmark provides two setups - **MT10**: a suite of 10 tasks and **MT50**: a suite of 50 tasks (a superset of MT10). We use both setups for evaluation. In Table 1, we compare the performance of the CARE model for MT10 with the different baselines and report the performance after 2M steps. Following the setup in Kaiser et al. (2019); Srinivas et al. (2020), we also compare the performance in the low-sample regime with 100K steps per task in Table 2 and 500K steps per task in Table 16 (in the Appendix). We note that the proposed CARE model consistently outperforms the other models for the MT10 task. We also note that among specialized multi-task algorithms, PCGrad (Yu et al., 2020a) performs quite poorly in the low-sample regime.

Similarly, in Table 3 and Table 4, we compare the performance of the models for MT50 after 2M and 100K steps respectively. The corresponding table for 500K steps is in the Appendix (Table 17). We note that not only does CARE outperform the other baselines, it is much more sample efficient than other baselines in the low-sample regime.

One additional benefit of the CARE model is that it can be easily combined with more powerful policies and learning algorithms. For example, we find that using multi-headed SAC with the CARE model significantly improves the performance on the MT50 setup (mean success of 0.61, with a standard error of 0.0287). Since the CARE model focuses on learning representations, it can benefit from the improvements in policy optimisation for multi-task RL.

In Appendix C.1 we consider some ablations with the CARE model. First, we vary the number of encoders, showing that increasing the number of encoders hurts performance when shared information is no longer leveraged. We also try using only top-k encoders (i.e. encoders with top-k highest attention scores) with hard attention, confirming the robustness of our method to different aggregation techniques. Finally, we design an experiment where we hardcode the mapping between the tasks and the encoders, showing that mapping

Table 4. Evaluation performance on the **MT50** test environments, after training for **100 thousand** steps (for each environment). Results averaged over 10 seeds. ME = Mixture of Encoders. CARE's improvement is statistically significant for baselines with *.

Agent	success (mean ± stderr)
Multi-task SAC (Yu et al., 2020b) *	0.13 ± 0.0061
Multi-task SAC + Task Encoder *	0.28 ± 0.015
Multi-headed SAC (Yu et al., 2020b) *	0.19 ± 0.0071
PCGrad (Yu et al., 2020a) *	0.21 ± 0.0068
Soft Modularization (Yang et al., 2020) *	0.20 ± 0.023
SAC + FiLM (Perez et al., 2018) *	0.16 ± 0.006
SAC + Metadata + ME (**CARE**)	$\mathbf{0.40 \pm 0.015}$

Table 5. Effect of using the metadata or the ensemble of encoders or both (proposed CARE model). Evaluation performance on the MT10 test environments, after training for **2 million** steps (for each environment). ME = Mixture of Encoders. CARE's improvement is statistically significant for the baselines marked with *.

Agent	success (mean ± stderr)
SAC + ME	0.74 ± 0.043
SAC + Metadata	0.79 ± 0.041
SAC + Metadata + ME (**CARE**)	$\mathbf{0.84 \pm 0.051}$

encoders to specific objects and skills helps performance.

Table 6. Effect of using the metadata or the ensemble of encoders or both (proposed CARE model). Evaluation performance on the MT50 test environments, after training for **2 million** steps (for each environment). ME = Mixture of Encoders. CARE's improvement is statistically significant for the baselines marked with *.

Agent	success (mean ± stderr)
SAC + Mixture of Encoders *	0.44 ± 0.012
SAC + Metadata	0.48 ± 0.025
SAC + Metadata + ME (**CARE**)	$\mathbf{0.54 \pm 0.031}$

4.2. Is the metadata useful only when learning compositional representations?

In Section 4.1, we showed that the metadata is useful for learning compositional representations in the CARE model. A follow-up question is whether the metadata is also useful when using a single encoder. We address this question in Tables 5 and 6 where we compare the performance of the CARE model in the absence of metadata or in the absence of mixture of encoders. We note that removing the metadata (first row) hurts the models more than removing the mixture of encoders (second row). While using metadata with just a single encoder (second row) provides comparable performance to the other baselines, using it with a mixture of encoders leads to even better performance.

Table 7. Evaluation performance on held-out environments from MT10, after training on remaining 8 environments for **2 million** steps for each environment. Results averaged over 10 seeds. ME = Mixture of Encoders. CARE's improvement is statistically significant for baselines marked with *.

Agent	success (mean ± stderr)
PCGrad (Yu et al., 2020a) *	0.05 ± 0.076
Soft Modularization (Yang et al., 2020)	0.1 ± 0.089
SAC + FiLM (Perez et al., 2018)	0.2 ± 0.073
SAC + Metadata + ME (**CARE**)	$\mathbf{0.3 \pm 0.077}$

4.3. Interpreting the specialized representations

We perform a visual investigation of what information is being shared across tasks and the role of metadata in the information sharing. Specifically, in Figure 4 we show cosine similarity between the context representations under the 10 tasks of MT10. For each task, we feed the context through the pretrained language model (RoBERTa) and refer to the resulting representation as the pretrained task embedding. In Figure 4b, we show the cosine similarity between just these pretrained task embeddings, which are the input to the MLP of the context encoder (Figure 3). In Figure 4c, we show the cosine similarity between the context representations from the proposed CARE model (with 6 encoders) after training. We observe a similar structure in both similarity matrices, that tasks with semantically similar descriptions also have similar context encodings. For example, task 2 and 3 require the agent to interact with the same object. Similarly tasks 4, 5, and 6 are related by the skill "open" and object "drawer". We now want to observe if these similarities can be found from just the tasks, without access to metadata. In Figure 4d, we plot the cosine similarity between the context representations from the CARE model (with 6 encoders) without the use of task metadata. Some similarities are present, but nothing as strongly correlated as seen in Figure 4c. This result further supports our hypothesis that task metadata plays an important role in inferring similarity between tasks, and shows the interpretability of the CARE representation.

4.4. Zero-shot generalization to unseen environments

Given the compositionality present in language, we want to evaluate if CARE can be used for zero-shot generalization to unseen environments. We train the agents on 8 environments from MT10 and evaluate on two held-out environments, "drawer-open-v1" and "window-open-v1". There are three training environments that directly relate to these test environments, "drawer-close-v1", "window-close-v1", "door-open-v1", thus allowing for the possibility of zero-shot generalization. In Table 7, we observe that CARE

exhibits some promising performance. We note that the comparison is unfair to PCGrad and Soft Modularization as they do not have any means for generalizing to the unseen task, but our motivation is to highlight the potential benefits of using metadata for zero-shot generalization to unseen environments. We further note that CARE generalizes better than FiLM, which also leverages metadata.

5. Related Work

Multi-task learning holds the promise of accelerating learning across multiple tasks by sharing useful information (Caruana, 1997; Zhang et al., 2014; Kokkinos, 2017; Radford et al., 2019; Rajeswaran et al., 2016; Ruder, 2017; Liu et al., 2019a; Mott et al., 2019; Vithayathil Varghese & Mahmoud, 2020). **Multi-task reinforcement learning** (MTRL) has been extensively studied with the focus on assumptions around shared properties and structures of different tasks. (Calandriello et al., 2014; Borsa et al., 2016; Maurer et al., 2016) assume that in the multi-task settings, tasks share a common, low dimensional representation, and therefore advocate for learning a shared representation space across all the tasks. (Bräm et al., 2019) considered the setup when the action space between different tasks is not aligned. Zhang et al. (2021) describes multi-task learning algorithms where the tasks have different dynamics but a shared reward structure and makes the assumption of a universal dynamics model. All of these methods for the MTRL setting only utilize a simplistic context in the form of an ordinal task id. Richer contexts in the form of task embeddings are learned online from the differences in reward and dynamics across tasks. We instead focus on the setting where side information is available and can be utilized as a richer context than task ids.

Several works have focused on the problem of **negative interference** (Du et al., 2018; Suteu & Guo, 2019; Yu et al., 2020a) where the gradients corresponding to the different tasks interfere negatively with each other. Along with slowing down training, conflicting gradients could cause the agent to *unlearn* knowledge of one task to learn another task. Despite some successes, the proposed approaches are unsatisfactory, either because they increase the computational/memory overhead (for example, Yu et al. (2020a) introduces an $\mathcal{O}(n^2)$ complexity, where n is the number of tasks) or because they require the training model to ignore some components of the gradients, thus slowing down learning and deteriorating sample efficiency. We propose to use the context for deciding which information should be shared across tasks, thus alleviating negative interference.

Contextual MDPs have been previously defined and analyzed as a setting where side information is exploited for transfer across tasks (Hallak et al., 2015; Modi et al., 2017). Hallak et al. (2015) first defined the contextual MDP setting, drawing connections to contextual multi-arm bandits (Lai & Robbins, 1985; Langford & Zhang, 2008). However, they assume that the state spaces across contexts are the same. Modi et al. (2017) requires an assumption of smoothness in the MDP parameters with respect to the context and examine the online learning scenario, providing an extension of the Rmax algorithm with PAC bounds. They also assume that the context is given. Klink et al. (2019) adopt the contextual MDP setting but with an additional assumption that the agent can control the context, and therefore the task distribution. They propose an algorithm to generate a curriculum that allows the agent to gradually progress to a target context distribution. In our work, we relax several of the assumptions made by these prior works and extend the type of context explored.

Multi-task learning with metadata has been used in You et al. (2016); Zheng et al. (2019) where the focus is on Task Relation Discovery in the context of supervised learning. In Reinforcement Learning, our work has close ties to **language-conditioned RL**, where natural language phrases have been used as part of task descriptions in the context of several single task RL setups like goal-oriented RL (Chao et al., 2011; Chevalier-Boisvert et al., 2019), grounded language acquisition (Hermann et al., 2017; Chaplot et al., 2017), and instruction following (Tellex et al., 2011; Chen & Mooney, 2011; Williams et al., 2018). Similar to our BC-MDP setting, recent work (Zhong et al., 2020) also defines meta-environments specific to the language-conditioned setting. In all these works, the language description is a part of the problem specification and not just extra information, while our work proposes a way to incorporate auxiliary side information to improve sample efficiency and generalization in multi-task RL. Many of these works make additional assumptions about the language description. For example, (Shu et al., 2017) uses "a two-word tuple template consisting of a skill and an item" to describe the task while the metadata for CARE can be high-level, under-specified, and unstructured. Further, we note that the contextual setting we propose includes information beyond just text, and can also include images or features.

Several works have also focused on **learning compositional models** for multi-task learning. Liu et al. (2019a) trains a network with task-specific soft-attention modules. Devin et al. (2017) decompose the policy into two components – "task-specific" (shared across all robots) and "robot-specific" (shared across all tasks). Chang et al. (2018) consider a family of algorithmic tasks, with different levels of complexity. Yang et al. (2020) performs routing in a base policy network to generate different policies for different tasks. Unlike our work, these works do not leverage metadata to learn a context on which the module decomposition can be conditioned on.

6. Discussion

In this work, we highlight an under-explored setting of using contextual information to improve performance in multi-task reinforcement learning. We show that metadata, which is often present in multi-task settings, can be leveraged within the MDP framework to improve performance by enabling more efficient sharing of information across tasks. Further, we define a new setting, the block contextual Markov decision process (BC-MDP), to handle settings where the state space can differ across tasks. Finally, we show that a mixture of encoders can effectively learn a context-dependent representation that can be used with a single policy to solve a family of tasks. We showcase our method, CARE, on Meta-World and achieve new state-of-the-art results.

There are two dimensions along which this work can be extended. Here, we only explore a specific type of context in the form of text descriptions for each task. Other forms of context can also be considered, such as people or places. As examples, in personalized medicine or recommendation systems, the user can be used as a form of context to adapt and share knowledge from other users to transfer the policy. Similarly, in household robotics, the location and layouts of homes will differ although the task is the same.

The relevance of this work also carries over to rich observation settings, or the high-dimensional version of the block contextual MDP setting we examine here (Mozifian et al., 2020). This setting is relevant in robotics, where the agent typically has a first person view of the world. The entire state space (the world) cannot be captured in each task. Instead, the agent only focuses on a relevant subspace of the entire state space at a time. This extension significantly enriches the set of multi-task problems we can tackle and brings us closer to the intractable partial observability (POMDP) setting (Kaelbling et al., 1998; Zhang et al., 2019), but offers more flexibility that allows for additional tractability by limiting the partial observability settings we consider.

Acknowledgements

We thank Edward Grefenstette, Tim Rocktäschel, Danielle Rothermel and Olivier Delalleau for feedback that improved this paper.

References

Allen, K. R., Smith, K. A., and Tenenbaum, J. B. Rapid trial-and-error learning with simulation supports flexible tool use and physical reasoning. *Proceedings of the National Academy of Sciences*, 117(47):29302–29310, 2020. ISSN 0027-8424. doi: 10.1073/pnas.1912341117. URL https://www.pnas.org/content/117/47/29302.

Andreas, J., Klein, D., and Levine, S. Modular multitask reinforcement learning with policy sketches. In *International Conference on Machine Learning*, pp. 166–175, 2017.

Bellman, R. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

Borsa, D., Graepel, T., and Shawe-Taylor, J. Learning shared representations in multi-task reinforcement learning. *CoRR*, abs/1603.02041, 2016. URL http://arxiv.org/abs/1603.02041.

Bräm, T., Brunner, G., Richter, O., and Wattenhofer, R. Attentive multi-task deep reinforcement learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 134–149. Springer, 2019.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Calandriello, D., Lazaric, A., and Restelli, M. Sparse multi-task reinforcement learning. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q. (eds.), *Advances in neural information processing systems 27*, pp. 819–827. Curran Associates, Inc., 2014.

Caruana, R. Multitask learning. *Machine learning*, 28(1): 41–75, 1997.

Chang, M. B., Gupta, A., Levine, S., and Griffiths, T. L. Automatically composing representation transformations as a means for generalization. *arXiv preprint arXiv:1807.04640*, 2018.

Chao, C., Cakmak, M., and Thomaz, A. L. Towards grounding concepts for transfer in goal learning from demonstration. In *2011 IEEE International Conference on Development and Learning (ICDL)*, volume 2, pp. 1–6. IEEE, 2011.

Chaplot, D. S., Sathyendra, K. M., Pasumarthi, R. K., Rajagopal, D., and Salakhutdinov, R. Gated-attention architectures for task-oriented language grounding. *arXiv preprint arXiv:1706.07230*, 2017.

Chen, D. L. and Mooney, R. J. Learning to interpret natural language navigation instructions from observations. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

Chen, Z., Badrinarayanan, V., Lee, C.-Y., and Rabinovich, A. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *International Conference on Machine Learning*, pp. 794–803. PMLR, 2018.

Chevalier-Boisvert, M., Bahdanau, D., Lahlou, S., Willems, L., Saharia, C., Nguyen, T. H., and Bengio, Y. BabyAI: First steps towards grounded language learning with a human in the loop. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJeXCo0cYX.

D'Eramo, C., Tateo, D., Bonarini, A., Restelli, M., and Peters, J. Sharing knowledge in multi-task deep reinforcement learning. In *International Conference on Learning Representations*, 2020.

Devin, C., Gupta, A., Darrell, T., Abbeel, P., and Levine, S. Learning modular neural network policies for multi-task and multi-robot transfer. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2169–2176. IEEE, 2017.

Du, S. S., Krishnamurthy, A., Jiang, N., Agarwal, A., Dudík, M., and Langford, J. Provably efficient RL with rich observations via latent state decoding. *CoRR*, abs/1901.09018, 2019.

Du, Y., Czarnecki, W. M., Jayakumar, S. M., Pascanu, R., and Lakshminarayanan, B. Adapting auxiliary losses using gradient similarity. *arXiv preprint arXiv:1812.02224*, 2018.

El Bsat, S., Bou-Ammar, H., and Taylor, M. E. Scalable multitask policy gradient reinforcement learning. In *AAAI*, pp. 1847–1853, 2017.

Goyal, A., Lamb, A., Hoffmann, J., Sodhani, S., Levine, S., Bengio, Y., and Schölkopf, B. Recurrent independent mechanisms. *arXiv preprint arXiv:1909.10893*, 2019.

Greff, K., Kaufman, R. L., Kabra, R., Watters, N., Burgess, C., Zoran, D., Matthey, L., Botvinick, M., and Lerchner, A. Multi-object representation learning with iterative variational inference. *arXiv preprint arXiv:1903.00450*, 2019.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pp. 1861–1870. PMLR, 2018.

Hallak, A., Castro, D. D., and Mannor, S. Contextual markov decision processes, 2015.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del R'ıo, J. F., Wiebe, M., Peterson, P., G'erard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL https://doi.org/10.1038/s41586-020-2649-2.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. Deep reinforcement learning that matters. *arXiv preprint arXiv:1709.06560*, 2017.

Hermann, K. M., Hill, F., Green, S., Wang, F., Faulkner, R., Soyer, H., Szepesvari, D., Czarnecki, W. M., Jaderberg, M., Teplyashin, D., et al. Grounded language learning in a simulated 3d world. *arXiv preprint arXiv:1706.06551*, 2017.

Igl, M., Gambardella, A., He, J., Nardelli, N., Siddharth, N., Böhmer, W., and Whiteson, S. Multitask soft option learning, 2020. URL https://openreview.net/forum?id=BkeDGJBKvB.

Jiang, Y., Gu, S. S., Murphy, K. P., and Finn, C. Language as an abstraction for hierarchical deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 9419–9431, 2019.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. Planning and acting in partially observable stochastic domains. *Artif. Intell.*, 101(1–2):99–134, May 1998. ISSN 0004-3702.

Kaiser, L., Babaeizadeh, M., Milos, P., Osinski, B., Campbell, R. H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., et al. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019.

Kipf, T. N., Fetaya, E., Wang, K., Welling, M., and Zemel, R. S. Neural relational inference for interacting systems. In Dy, J. G. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2693–2702. PMLR, 2018. URL http://proceedings.mlr.press/v80/kipf18a.html.

Klink, P., Abdulsamad, H., Belousov, B., and Peters, J. Self-paced contextual reinforcement learning. In Kaelbling,

L. P., Kragic, D., and Sugiura, K. (eds.), *3rd Annual Conference on Robot Learning, CoRL 2019, Osaka, Japan, October 30 - November 1, 2019, Proceedings*, volume 100 of *Proceedings of Machine Learning Research*, pp. 513–529. PMLR, 2019. URL http://proceedings.mlr.press/v100/klink20a.html.

Kokkinos, I. Ubernet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6129–6138, 2017.

Lai, T. and Robbins, H. Asymptotically efficient adaptive allocation rules. *Adv. Appl. Math.*, 6(1):4–22, March 1985. ISSN 0196-8858. doi: 10.1016/0196-8858(85)90002-8. URL https://doi.org/10.1016/0196-8858(85)90002-8.

Langford, J. and Zhang, T. The epoch-greedy algorithm for multi-armed bandits with side information. In Platt, J., Koller, D., Singer, Y., and Roweis, S. (eds.), *Advances in Neural Information Processing Systems*, volume 20, pp. 817–824. Curran Associates, Inc., 2008. URL https://proceedings.neurips.cc/paper/2007/file/4b04a686b0ad13dce35fa99fa4161c65-Paper.pdf.

Li, Y., Torralba, A., Anandkumar, A., Fox, D., and Garg, A. Causal discovery in physical systems from videos, 2020.

Liu, S., Johns, E., and Davison, A. J. End-to-end multi-task learning with attention. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1871–1880, 2019a.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019b.

Locatello, F., Weissenborn, D., Unterthiner, T., Mahendran, A., Heigold, G., Uszkoreit, J., Dosovitskiy, A., and Kipf, T. Object-centric learning with slot attention. *arXiv preprint arXiv:2006.15055*, 2020.

Luketina, J., Nardelli, N., Farquhar, G., Foerster, J., Andreas, J., Grefenstette, E., Whiteson, S., and Rocktäschel, T. A survey of reinforcement learning informed by natural language. *arXiv preprint arXiv:1906.03926*, 2019.

Maurer, A., Pontil, M., and Romera-Paredes, B. The benefit of multitask representation learning. *J. Mach. Learn. Res.*, 17(1):2853–2884, January 2016. ISSN 1532-4435.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning, 2013.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Nature*, 518 (7540):529–533, February 2015. ISSN 00280836. URL http://dx.doi.org/10.1038/nature14236.

Modi, A., Jiang, N., Singh, S., and Tewari, A. Markov Decision Processes with Continuous Side Information. *arXiv:1711.05726 [cs, stat]*, November 2017. URL http://arxiv.org/abs/1711.05726. arXiv: 1711.05726.

Mott, A., Zoran, D., Chrzanowski, M., Wierstra, D., and Rezende, D. J. Towards interpretable reinforcement learning using attention augmented agents. *arXiv preprint arXiv:1906.02500*, 2019.

Mozifian, M., Zhang, A., Pineau, J., and Meger, D. Intervention design for effective sim2real transfer. *arXiv preprint arXiv:2012.02055*, 2020.

pandas development team, T. pandas-dev/pandas: Pandas, February 2020. URL https://doi.org/10.5281/zenodo.3509134.

Parascandolo, G., Kilbertus, N., Rojas-Carulla, M., and Schölkopf, B. Learning independent causal mechanisms. In *International Conference on Machine Learning*, pp. 4036–4044. PMLR, 2018.

Parisotto, E., Ba, J. L., and Salakhutdinov, R. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*, 2015.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

Pathak, D., Lu, C., Darrell, T., Isola, P., and Efros, A. A. Learning to Control Self-Assembling Morphologies: A Study of Generalization via Modularity. In Wallach, H., Larochelle, H., Beygelzimer, A., Alch-Buc, F. d., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 2295–2305. Curran Associates, Inc., 2019.

Perez, E., Strub, F., De Vries, H., Dumoulin, V., and Courville, A. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Puterman, M. L. Markov decision processes: Discrete stochastic dynamic programming. *Journal of the Operational Research Society*, 1995.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.

Rajeswaran, A., Ghotra, S., Ravindran, B., and Levine, S. Epopt: Learning robust neural network policies using model ensembles. *arXiv preprint arXiv:1610.01283*, 2016.

Ruder, S. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017.

Rusu, A. A., Colmenarejo, S. G., Çaglar Gülçehre, Desjardins, G., Kirkpatrick, J., Pascanu, R., Mnih, V., Kavukcuoglu, K., and Hadsell, R. Policy distillation. In *ICLR (Poster)*, 2016. URL http://arxiv.org/abs/1511.06295.

Shu, T., Xiong, C., and Socher, R. Hierarchical and interpretable skill acquisition in multi-task reinforcement learning. *arXiv preprint arXiv:1712.07294*, 2017.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., and Hassabis, D. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, October 2017. ISSN 0028-0836, 1476-4687. doi: 10.1038/nature24270.

Sodhani, S. and Zhang, A. Mtrl - multi task rl algorithms. Github, 2021. URL https://github.com/facebookresearch/mtrl.

Sodhani, S., Denoyer, L., Kamienny, P.-A., and Delalleau, O. Mtenv - environment interface for mulit-task reinforcement learning. Github, 2021. URL https://github.com/facebookresearch/mtenv.

Srinivas, A., Laskin, M., and Abbeel, P. Curl: Contrastive unsupervised representations for reinforcement learning. *arXiv preprint arXiv:2004.04136*, 2020.

Standley, T., Zamir, A. R., Chen, D., Guibas, L., Malik, J., and Savarese, S. Which tasks should be learned together in multi-task learning? *arXiv preprint arXiv:1905.07553*, 2019.

Student. The probable error of a mean. *Biometrika*, pp. 1–25, 1908.

Suteu, M. and Guo, Y. Regularizing deep multi-task networks using orthogonal gradients. *arXiv preprint arXiv:1912.06844*, 2019.

Tanaka, F. and Yamamura, M. Multitask reinforcement learning on the distribution of MDPs. In *Proceedings 2003 IEEE International Symposium on Computational Intelligence in Robotics and Automation. Computational Intelligence in Robotics and Automation for the New Millennium (Cat. No. 03EX694)*, volume 3, pp. 1108–1113. IEEE, 2003.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., de Las Casas, D., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., Lillicrap, T. P., and Riedmiller, M. A. Deepmind control suite. *CoRR*, abs/1801.00690, 2018. URL http://arxiv.org/abs/1801.00690.

Teh, Y., Bapst, V., Czarnecki, W. M., Quan, J., Kirkpatrick, J., Hadsell, R., Heess, N., and Pascanu, R. Distral: Robust multitask reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 4496–4506, 2017.

Tellex, S., Kollar, T., Dickerson, S., Walter, M., Banerjee, A., Teller, S., and Roy, N. Understanding natural language commands for robotic navigation and mobile manipulation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 25, 2011.

Vithayathil Varghese, N. and Mahmoud, Q. H. A survey of multi-task deep reinforcement learning. *Electronics*, 9(9), 2020. ISSN 2079-9292. doi: 10.3390/electronics9091363. URL https://www.mdpi.com/2079-9292/9/9/1363.

Williams, E. C., Gopalan, N., Rhee, M., and Tellex, S. Learning to parse natural language to grounded reward functions with weak supervision. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–7. IEEE, 2018.

Yadan, O. Hydra - a framework for elegantly configuring complex applications. Github, 2019. URL https://github.com/facebookresearch/hydra.

Yang, R., Xu, H., Wu, Y., and Wang, X. Multi-task reinforcement learning with soft modularization. In *Advances in Neural Information Processing Systems*, 2020.

You, Q., Wu, O., Luo, G., and Hu, W. Metadata-based clustered multi-task learning for thread mining in web communities. In *International Conference on Machine Learning and Data Mining in Pattern Recognition*, pp. 421–434. Springer, 2016.

Yu, T., Kumar, S., Gupta, A., Levine, S., Hausman, K., and Finn, C. Gradient surgery for multi-task learning. *arXiv preprint arXiv:2001.06782*, 2020a.

Yu, T., Quillen, D., He, Z., Julian, R., Hausman, K., Finn, C., and Levine, S. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning*, pp. 1094–1100, 2020b.

Zhang, A., Lipton, Z. C., Pineda, L., Azizzadenesheli, K., Anandkumar, A., Itti, L., Pineau, J., and Furlanello, T. Learning causal state representations of partially observable environments, 2019.

Zhang, A., Lyle, C., Sodhani, S., Filos, A., Kwiatkowska, M., Pineau, J., Gal, Y., and Precup, D. Invariant causal prediction for block MDPs. In *International Conference on Machine Learning (ICML)*, 2020.

Zhang, A., Sodhani, S., Khetarpal, K., and Pineau, J. Learning robust state abstractions for hidden-parameter block MDPs. In *International Conference on Learning Representations*, 2021.

Zhang, Z., Luo, P., Loy, C. C., and Tang, X. Facial landmark detection by deep multi-task learning. In *European conference on computer vision*, pp. 94–108. Springer, 2014.

Zheng, Z., Wang, Y., Dai, Q., Zheng, H., and Wang, D. Metadata-driven task relation discovery for multi-task learning. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pp. 4426–4432. International Joint Conferences on Artificial Intelligence Organization, 7 2019. doi: 10.24963/ijcai.2019/615. URL https://doi.org/10.24963/ijcai.2019/615.

Zhong, V., Rocktäschel, T., and Grefenstette, E. Rtfm: Generalising to new environment dynamics via reading. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SJgob6NKvH.

A. Additional Implementation Details

A.1. Libraries

We use the following open-source libraries: PyTorch (Paszke et al., 2019)[4], Hydra (Yadan, 2019)[5], MetaWorld (Yu et al., 2020b)[6], MTEnv (Sodhani et al., 2021)[7], MTRL (Sodhani & Zhang, 2021)[8], Numpy (Harris et al., 2020)[9] and Pandas (pandas development team, 2020)[10]. For MetaWorld, we use the following commit-id to run our experiments: `https://github.com/rlworkgroup/metaworld/commit/af8417bfc82a3e249b4b02156518d775f29eb289`

A.2. SAC Algorithm

We use the SAC policy algorithm (Haarnoja et al., 2018) to learn representation for the CARE model. We provide the pseduo-code for SAC, along with the key equations. For more details, refer to Haarnoja et al. (2018).

Algorithm 2 UPDATESAC($\mathcal{D}, z_t^i$)

Require: SAC components i.e. Policy parameters (ϕ), Value function parameters (ψ), Target value function parameters ($\bar{\psi}$) and Q-function parameters (θ)

1: $a_t \sim \pi(\cdot|z_t^i)$
2: Execute a_t in the environment (for task $\mathcal{T}_i$) and get the next state s_{t+1}, reward r_t and *done* signal d_t.
3: $\mathcal{D} \leftarrow \mathcal{D} \cup (s_t, a_t, s_{t+1}, r_t, d_t, i)$
4: $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$ (Equation (2))
5: $\bar{\psi} \leftarrow \tau\psi + (1-\tau)\bar{\psi}$
6: $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$ (Equation (5))
7: $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$ (Equation (6))

The soft value function is trained to minimize the squared residual error

$$J_V(\psi) = \mathbb{E}_{\mathbf{s}_t \sim \mathcal{D}} \left[\frac{1}{2} \left(V_\psi(\mathbf{s}_t) - \mathbb{E}_{\mathbf{a}_t \sim \pi_\phi} \left[Q_\theta(\mathbf{s}_t, \mathbf{a}_t) - \log \pi_\phi(\mathbf{a}_t|\mathbf{s}_t) \right] \right)^2 \right]. \tag{1}$$

The gradient of Equation (1) is estimated with an unbiased estimator

$$\hat{\nabla}_\psi J_V(\psi) = \nabla_\psi V_\psi(\mathbf{s}_t) \left(V_\psi(\mathbf{s}_t) - Q_\theta(\mathbf{s}_t, \mathbf{a}_t) + \log \pi_\phi(\mathbf{a}_t|\mathbf{s}_t) \right), \tag{2}$$

where the actions are sampled according to the current policy, instead of the replay buffer. The update uses a target value network $V_{\bar{\psi}}$, where $\bar{\psi}$ is an exponentially moving average of the value network weights, which has been shown to stabilize training (Mnih et al., 2015).

The soft Q-function parameters are trained to minimize the soft Bellman residual

$$J_Q(\theta) = \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \mathcal{D}} \left[\frac{1}{2} \left(Q_\theta(\mathbf{s}_t, \mathbf{a}_t) - \hat{Q}(\mathbf{s}_t, \mathbf{a}_t) \right)^2 \right], \tag{3}$$

with

$$\hat{Q}(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \, \mathbb{E}_{\mathbf{s}_{t+1} \sim p} \left[V_{\bar{\psi}}(\mathbf{s}_{t+1}) \right], \tag{4}$$

which is optimized with stochastic gradients

$$\hat{\nabla}_\theta J_Q(\theta) = \nabla_\theta Q_\theta(\mathbf{a}_t, \mathbf{s}_t) \left(Q_\theta(\mathbf{s}_t, \mathbf{a}_t) - r(\mathbf{s}_t, \mathbf{a}_t) - \gamma V_{\bar{\psi}}(\mathbf{s}_{t+1}) \right). \tag{5}$$

The policy parameters are learned by minimizing:

$$J_\pi(\phi) = \mathbb{E}_{\mathbf{s}_t \sim \mathcal{D}} \left[\mathrm{D}_{\mathrm{KL}} \left(\pi_\phi(\cdot|\mathbf{s}_t) \,\middle\|\, \frac{\exp\left(Q_\theta(\mathbf{s}_t, \cdot)\right)}{Z_\theta(\mathbf{s}_t)} \right) \right]. \tag{6}$$

A.3. CARE Components

The CARE algorithm uses two components: (i) A context encoder network and (ii) mixture of k encoders (shown in Figure 3). Both components are trained using the policy loss as described in Algorithm 3 and Algorithm 4 respectively. We note that for computing the encoder representation z_{enc}, the context encoding $z_{context}$ is detached from the computational graph.

Algorithm 3 UPDATECONTEXTENCODER($\mathcal{D}, z_t^i$)

Require: Context Encoder Network C with ω as the parameters, SAC components

1: $a_t \sim \pi(\cdot|z_t^i)$
2: Execute a_t in the environment (for task $\mathcal{T}_i$) and get the next state s_{t+1}, reward r_t and *done* signal d_t.
3: $\mathcal{D} \leftarrow \mathcal{D} \cup (s_t, a_t, s_{t+1}, r_t, d_t, i)$
4: $J_C(\omega) = J_V + J_Q + J_\pi$
5: $\omega \leftarrow \omega - \lambda_\omega \hat{\nabla}_\omega J_C(\omega)$

B. Hyperparameter Details

In this section, we provide hyper-parameter values for each of the methods in our experimental evaluation. In Table 8, we provide the hyperparameter values that are common acrss all the methods.

[4] `https://pytorch.org/`
[5] `https://github.com/facebookresearch/hydra`
[6] `https://github.com/rlworkgroup/metaworld`
[7] `https://github.com/facebookresearch/mtenv`
[8] `https://github.com/facebookresearch/mtrl`
[9] `https://numpy.org/`
[10] `https://pandas.pydata.org/`

Algorithm 4 UPDATEMIXTUREOFENCODERS($\mathcal{D}, z_t^i$)

Require: k Encoders $E_1, \ldots, E_k$ with ζ_k as the parameters, SAC components
$a_t \sim \pi(\cdot|z_t^i)$
Execute a_t in the environment (for task $\mathcal{T}_i$) and get the next state s_{t+1}, reward r_t and *done* signal d_t.
for each encoder in $k = 1..K$ **do**
  $\mathcal{D} \leftarrow \mathcal{D} \cup (s_t, a_t, s_{t+1}, r_t, d_t, i)$
  $J_{E_k}(\zeta_k) = J_V + J_Q + J_\pi$
  $\zeta_k \leftarrow \zeta_k - \lambda_{\zeta_k} \hat{\nabla}_{\zeta_k} J_{E_k}(\zeta_k)$
end for

C. Additional Results

C.1. Ablations

In Table 18, we consider some ablations with the CARE model for MT10. First, we vary the number of encoders (k). We note that having too many encoders can hurt the performance. Second, we consider the case where we use initialise m encoders but use only top-k encoders at each timestep. Top-k encoders are the ones which have the top-k highest attention scores. The remaining $m - k$ are not used and are considered to be *inactive*. The attention scores α, corresponding to the selected k encoders, are re-normalized to sum to 1. In this case,

$$z_{enc} = \frac{\sum_i^m \alpha_i \times z_{enc}^i \times \mathbb{1}_{top-k}(i)}{\sum_i^m \alpha_i \times \mathbb{1}_{top-k}(i)},$$

where $\mathbb{1}_{top-k}(i)$ indicates if the α_i is among the top-k attention weights. We find that while this form of *hard-attention* does not work as well as the *soft-attention* mechanism used by CARE, the performance is still comparable to the performance of other baselines. Finally, we consider a hand-coded assignment of tasks to encoders. We read the task descriptions, identify common skills and objects, and manually assign these skills and objects to encoders. The tasks that mention these skills/objects are then mapped to the corresponding encoders. The resulting task-encoder mappings are shown in Table 20. We note that this manual mapping of encoders works very well in practice and the mapping is interpretable by design. However, the process of creating such mappings is time consuming and error-prone. On the other hand, CARE can learn the mapping between tasks and encoders while learning the representations. This experiment validates the hypothesis that explicitly assigning object and skill concepts to the encoders improves multi-task performance, and lends credibility to the additional hypothesis that CARE learns interpretable representations, i.e. ones that represent objects and skills.

C.2. Effect of frequency of evaluation

We find that we can improve the agent's performance just by increasing the frequency of evaluation as shown in Table 19.

Table 8. Hyperparameter values that are common across all the methods.

Hyperparameter	Hyperparameter values
batch size	128 $\times$ number of tasks
network architecture	feedforward network
actor/critic size	three fully connected layers with 400 units
non-linearity	ReLU
policy initialization	standard Gaussian
exploration parameters	run a uniform exploration policy 1500 steps
# of samples / # of train steps per iteration	1 env step / 1 training step
policy learning rate	3e-4
Q function learning rate	3e-4
optimizer	Adam
policy learning rate	3e-4
beta for Adam optimizer for policy	(0.9, 0.999)
Q function learning rate	3e-4
beta for Adam optimizer for Q function	(0.9, 0.999)
discount	.99
Episode length (horizon)	150
reward scale	1.0

Table 9. Hyperparameter values for Multi-task SAC

Hyperparameter	Hyperparameter values
temperature	learned and disentangled with tasks

Table 10. Hyperparameter values for Multi-task SAC + Task Encoder

Hyperparameter	Hyperparameter values
task encoder size	embedding layer + two fully connected layers. Embedding/hidden/output dims = 50
temperature	learned and disentangled with tasks

Table 11. Hyperparameter values for Multi-headed SAC

Hyperparameter	Hyperparameter values
network architecture	multi-head (one head per task)
temperature	learned and disentangled with tasks

Table 12. Hyperparameter values for PCGrad

Hyperparameter	Hyperparameter values
actor/critic size	five fully connected layers with 400 units
temperature	learned and disentangled with tasks

Table 13. Hyperparameter values for Soft Modularization

Hyperparameter	Hyperparameter values
task encoder size	two layer feedforward network. Hidden/output dims = 50
routing network size	4 layers and 4 modules per layer.
temperature	learned and disentangled with tasks

Table 14. Hyperparameter values for FiLM

Hyperparameter	Hyperparameter values
task encoder size	two layer feedforward network. Hidden/output dims = 50
temperature	learned and disentangled with tasks

Table 15. Hyperparameter values for CARE

Hyperparameter	Hyperparameter values
task encoder size	two layer feedforward network. Hidden/output dims = 50
number of encoders	6 for MT10, 10 for MT50
temperature	learned and disentangled with tasks

In Meta-World, the agent is evaluated for a binary-valued success signal and evaluating the agent more often makes it more likely for the agent to solve a given task. This effect makes it harder to compare the performance of the algorithms from different works. We control for this issue in our setup by evaluating every agent at a fixed frequency (once every 10K environment steps, per task).

D. Testing for statistical significance

We perform a two-tailed, Student's t-distribution test (Student, 1908) under *equal sample sizes, unequal variance* setup (also called Welch's t-test). The null hypothesis is: the mean performance of the two models (CARE and any baseline) are equal. The significance level (p) is set to 0.05.

Table 16. Evaluation performance on the MT10 test environments, after training for **500 thousand** steps (for each environment). The results are averaged over 10 seeds. ME = Mixture of Encoders. CARE's improvement is statistically significant for the baselines marked with *.

Agent	success (mean $\pm$ stderr)
Multi-task SAC (Yu et al., 2020b) *	0.38 ± 0.041
Multi-task SAC + Task Encoder *	0.42 ± 0.031
Multi-headed SAC (Yu et al., 2020b) *	0.44 ± 0.045
PCGrad (Yu et al., 2020a) *	0.53 ± 0.030
Soft Modularization (Yang et al., 2020)	0.64 ± 0.052
SAC + FiLM (Perez et al., 2018) *	0.57 ± 0.035
SAC + Metadata + ME (**CARE**)	$\mathbf{0.66 \pm 0.028}$

Table 17. Evaluation performance on the MT50 test environments, after training for **500 thousand** steps (for each environment). The results are averaged over 10 seeds. The proposed method, CARE, outperforms the other baselines. ME = Mixture of Encoders. CARE's improvement is statistically significant for the baselines marked with *.

Agent	success (mean $\pm$ stderr)
Multi-task SAC * (Yu et al., 2020b)	0.28 ± 0.017
Multi-task SAC + Task Encoder *	0.37 ± 0.016
Multi-headed SAC (Yu et al., 2020b)	0.45 ± 0.064
PCGrad (Yu et al., 2020a)	0.47 ± 0.016
Soft Modularization (Yang et al., 2020)	0.47 ± 0.012
SAC + FiLM (Perez et al., 2018) *	0.30 ± 0.012
SAC + Metadata + ME (**CARE**)	$\mathbf{0.51 \pm 0.036}$

Table 18. Evaluation performance on the MT10 test environments, after training for **2 million** steps (for each environment). The results are averaged over 10 seeds. ME = Mixture of Encoders. CARE's improvement is statistically significant for the baselines marked with *.

Agent	success (mean $\pm$ stderr)
CARE with 2 encoders	0.76 ± 0.043
CARE with 10 encoders *	0.71 ± 0.029
CARE with 10 encoders with 2 active *	0.55 ± 0.051
CARE with 10 encoders with 4 active *	0.69 ± 0.040
CARE with 10 encoders with 6 active *	0.71 ± 0.051
CARE with 10 encoders with 8 active *	0.67 ± 0.043
Manually mapping tasks to encoders	0.80 ± 0.037
CARE with 6 encoders	$\mathbf{0.84 \pm 0.051}$

Table 19. Evaluation performance on the MT10 test environments, after training for **2 million** steps (for each environment). The agent is evaluated after 1K and 10K steps. The results are averaged over 10 seeds. ME = Mixture of Encoders. We note that evaluating the agent more frequently improves the performance across the baselines.

Agent	success (1K) (mean $\pm$ stderr)	success (10K) (mean $\pm$ stderr)
PCGrad (Yu et al., 2020a)	0.75 ± 0.033	0.72 ± 0.022
Soft Modularization (Yang et al., 2020)	0.78 ± 0.049	0.73 ± 0.043
SAC + Metadata + ME (**CARE**)	$\mathbf{0.87 \pm 0.054}$	$\mathbf{0.84 \pm 0.051}$

Table 20. Mapping between the encoders and skills/objects and tasks for MT10 environments.

Encoder Index	Skills or objects that map to the encoder	Tasks that map to the encoder
1	skill: close	drawer-close-v1, window-close-v1
2	skill: open	door-open-v1, drawer-open-v1, window-open-v1
3	skill: push	push-v1, door-open-v1, drawer-open-v1, window-open-v1
4	skill: any remaining skills	reach-v1, pick-place-v1, button-press-topdown-v1, peg-insert-side-v1
5	object: drawer	drawer-open-v1, drawer-close-v1
6	object: goal	reach-v1, push-v1, pick-place-v1, peg-insert-side-v1
7	object: puck	push-v1, pick-place-v1
8	object: window	window-open-v1, window-close-v1
9	object: any remaining objects	door-open-v1, button-press-topdown-v1, peg-insert-side-v1