Task-Embedded Control Networks for Few-Shot Imitation Learning

Stephen James
Dyson Robotics Lab
Imperial College London
slj12@imperial.ac.uk

Michael Bloesch
Dyson Robotics Lab
Imperial College London
m.bloesch@imperial.ac.uk

Andrew J. Davison
Dyson Robotics Lab
Imperial College London
a.davison@imperial.ac.uk

Abstract: Much like humans, robots should have the ability to leverage knowledge from previously learned tasks in order to learn new tasks quickly in new and unfamiliar environments. Despite this, most robot learning approaches have focused on learning a single task, from scratch, with a limited notion of generalisation, and no way of leveraging the knowledge to learn other tasks more efficiently. One possible solution is meta-learning, but many of the related approaches are limited in their ability to scale to a large number of tasks and to learn further tasks without forgetting previously learned ones. With this in mind, we introduce Task-Embedded Control Networks, which employ ideas from metric learning in order to create a task embedding that can be used by a robot to learn new tasks from one or more demonstrations. In the area of visually-guided manipulation, we present simulation results in which we surpass the performance of a state-of-the-art method when using only visual information from each demonstration. Additionally, we demonstrate that our approach can also be used in conjunction with domain randomisation to train our few-shot learning ability in simulation and then deploy in the real world without any additional training. Once deployed, the robot can learn new tasks from a single real-world demonstration.

Keywords: Manipulation, Few-shot Learning, Sim-to-Real

1 Introduction

Humans and animals are capable of learning new information rapidly from very few examples, and apparently improve their ability to 'learn how to learn' throughout their lives [1]. Endowing robots with a similar ability would allow for a large range of skills to be acquired efficiently, and for existing knowledge to be adapted to new environments and tasks. An emerging trend in robotics is to learn control directly from raw sensor data in an end-to-end manner. Such approaches have the potential to be general enough to learn a wide range of tasks, and they have been shown to be capable of performing tasks that traditional methods in robotics have found difficult, such as when close and complicated coordination is required between vision and control [2], or in tasks with dynamic environments [3]. However, these solutions often learn their skills from scratch and need a large amount of training data [3, 4, 5]. A significant goal in the community is to develop methods that can reuse past experiences in order to improve the data efficiency of these methods.

To that end, one significant approach is Meta-Imitation Learning (MIL) [6], in which a policy is learned that can be quickly adapted, via one or few gradient steps at test time, in order to solve a new task given one or more demonstrations. The under-

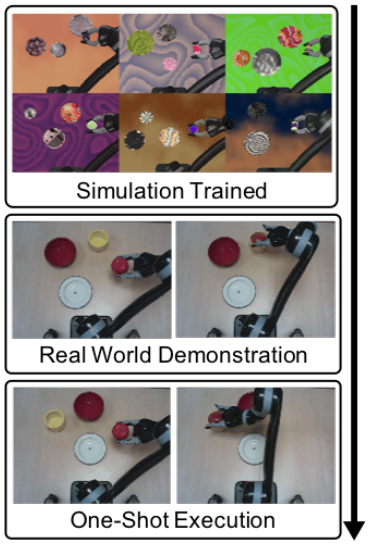

Figure 1: The robot gains its few-shot learning ability in simulation, and can then learn a new task from a single demonstration.

2nd Conference on Robot Learning (CoRL 2018), Zürich, Switzerland.

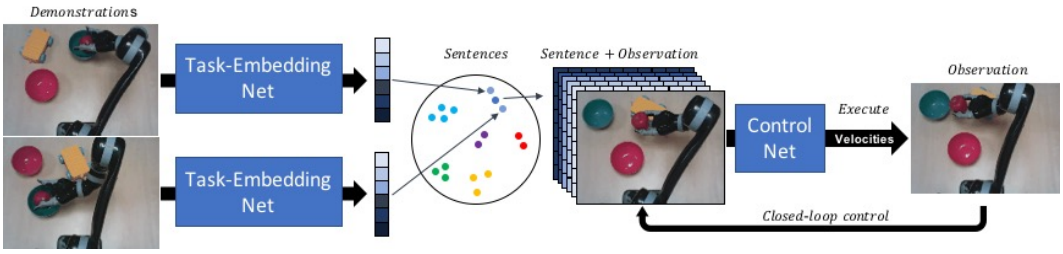

Figure 2: Task-Embedded Control Networks (TecNets) allow tasks to be learned from single or multiple demonstrations. Images of demonstrations are embedded into a compact representation of a task, which can be combined to create a *sentence*. This sentence is then expanded and concatenated (channel-wise) to the most recent observation from a new configuration of that task before being sent through the control network in a closed-loop manner. Both the task-embedding net and control net are jointly optimised to produce a rich embedding.

lying algorithm, Model-Agnostic Meta-Learning (MAML) [7] can be very general, but lacks some of the properties that we might hope for in a robotic system. For one, once the policy is trained, it cannot accomplish any of the tasks seen during training unless it is given an example again at test time. Also, once a specific task is learned, the method can lose its ability to meta-learn and be stuck with a set of weights that can only be used for that one task. One way around this is to make a copy of the weights needed for each task, but this raises scalability concerns.

Our new approach, Task-Embedded Control Networks (TecNets), is centred around the idea that there is an embedding of tasks, where tasks that are similar (in terms of visual appearance) should be close together in space, whilst ones that are different should be far away from one another. Having such an expressive space would not only allow for few-shot learning, but also opens the possibility of inferring information from new and unfamiliar tasks in a zero-shot fashion, such as how similar a new task may be to a previously seen one.

TecNets, which are summarised in Figure 2, are composed of a *task-embedding network* and a *control network* that are jointly trained to output actions (e.g. motor velocities) for a new variation of an unseen task, given a single or multiple demonstrations. The task-embedding network has the responsibility of learning a compact representation of a task, which we call a *sentence*. The control network then takes this (static) sentence along with current observations of the world to output actions. TecNets do not have a strict restriction on the number of tasks that can be learned, and do not easily forget previously learned tasks during training, or after. The setup only expects the observations (e.g. visual) from the demonstrator during test time, which makes it very applicable for learning from human demonstrations.

To evaluate our approach, we present simulation results from two experimental domains proposed in MIL [6], and demonstrate that we can train our meta-learning ability in simulation and then deploy in the real world without any additional training. We believe this to be a desirable property given that large amounts of data are needed to end-to-end solutions. Despite being trained to meta-learn in simulation, the robot can learn new tasks from a single demonstration in the real world.

Our contributions in this work are threefold. We demonstrate the ability to one-shot and few-shot learn visuomotor control through the use of TecNets in a selection of visually-guided manipulation tasks. Secondly, we show that TecNets are able to achieve higher success rates compared to MIL [6] when using only visual information from each demonstration. Finally, we demonstrate the first successful method of a few-shot learning approach trained in simulation and transferred to the real world, which we believe is an important direction for allowing large-scale generalisation.

2 Related Work

Our work lies at the intersection of imitation learning [8, 9] and meta-learning [10, 11]. Imitation learning aims to learn tasks by observing a demonstrator. One focus within imitation learning is *behavioural cloning*, in which the agent learns a mapping from observations to actions given demonstrations, in a supervised learning manner [12, 13]. Another focus is *inverse reinforcement*

learning [14], where an agent attempts to estimate a reward function that describes the given demonstrations [15, 16]. In our work, we focus on behavioural cloning in the context of learning motor control directly from pixels. A common issue in behavioural cloning is the large amount of data needed to train such systems [3], as well as the fact that tasks are often learned independently, where learning one task does not accelerate the learning of another. Recently, there has been encouraging work to address this problem [6], and our approach provides a further advance.

One-shot and few-shot learning is the paradigm of learning from a small number of examples at test time, and has been widely studied in the image recognition community [17, 18, 19, 20, 21, 22]. Many one-shot and few-shot learning methods in image recognition are a form of meta-learning, where the algorithms are tested on their ability to learn new tasks, rather than the usual machine learning paradigm of training on a single task and testing on held out examples of that task. Common forms of meta-learning include recurrence [19], learning an optimiser [20], and more recently Model Agnostic Meta-Learning (MAML) [7]. Many works in metric learning, including ours, can be seen as forms of meta-learning [17, 22], in the sense that they produce embeddings dynamically from new examples during test time; the difference to other more common meta-learning approaches is that the embedding generation is fixed after training.

The success of our new approach comes from learning a metric space, and there has been an abundance of work in metric learning for image classification [23, 24], from which we will summarise the most relevant. Matching Networks [17] use an attention mechanism over a learned embedding space which produces a weighted nearest neighbour classifier given labelled examples (support set) and unlabelled examples (query set). Prototypical Networks [22] are similar, but differ in that they represent each class by the mean of its examples (the prototype) and use a squared Euclidean distance rather than the cosine distance. In the case of one-shot learning, matching networks and prototypical networks become equivalent. Our approach is similar in that our sentence (prototype) is created by averaging over the support set, but differs in the way we couple the learned embedding space with the control network. These metric learning methods have all been developed for image classification, and in the visuomotor control domain of our method, we do not explicitly classify sentences, but instead jointly optimise them with a control network.

Recently, Hausman et al. [25] proposed learning a skill embedding space via reinforcement learning that led to speed-ups during training time. Although impressive, that method does not focus on few-shot learning, and the experiments are run within simulation with low dimensional state spaces. Another piece of work that uses embedding spaces is [26], where a multimodal embedding is learned for point-clouds, language and trajectories. This work involves pre-training the parts of the network, and also relies on accurate models of the world. Our approach has the benefit that we map directly from images to motor actions and train jointly embedding and control networks, with no pre-training.

In terms of setup, the closest related work to ours is MIL [6], where they apply MAML [7] and behaviour cloning to learn new tasks, end-to-end from one visual demonstration. The underlying algorithm, MAML, learns a set of weights that can be quickly adapted to new tasks. If we were to use this approach to retain information we had previously learnt, we would need to hold copies of weights for each task. In comparison, our method relies on storing a compact sentence for every task we want to remember.

3 Task-Embedded Control Networks

We now formally summarise the notation for our method. A policy π for task $\mathfrak{T}$ maps observations $\boldsymbol{o}$ to actions $\boldsymbol{a}$, and we assume to have expert policies π^* for multiple different tasks. Corresponding example trajectories consist of a series of observations and actions: $\tau = [(\boldsymbol{o}_1, \boldsymbol{a}_1), \ldots, (\boldsymbol{o}_T, \boldsymbol{a}_T)]$ and we define each task to be a set of such examples, $\mathfrak{T} = \{\tau_1, \cdots, \tau_K\}$. TecNets aim to learn a universal policy $\pi(\boldsymbol{o}, \boldsymbol{s})$ that can be modulated by a sentence $\boldsymbol{s}$, where $\boldsymbol{s}$ is a learned description of a task $\mathfrak{T}$. The resulting universal policy $\pi(\boldsymbol{o}, \boldsymbol{s})$ should emulate the expert policy π^* for task $\mathfrak{T}$.

3.1 Task Embedding

We now introduce our task embedding, which can be used independently in other fields, such as image classification, and so we keep this section general. Assume we are given a small set of K examples of a task $\mathfrak{T}^j$. Our task embedding network $f_\theta : \mathbb{R}^D \rightarrow \mathbb{R}^N$ computes a normalised N-

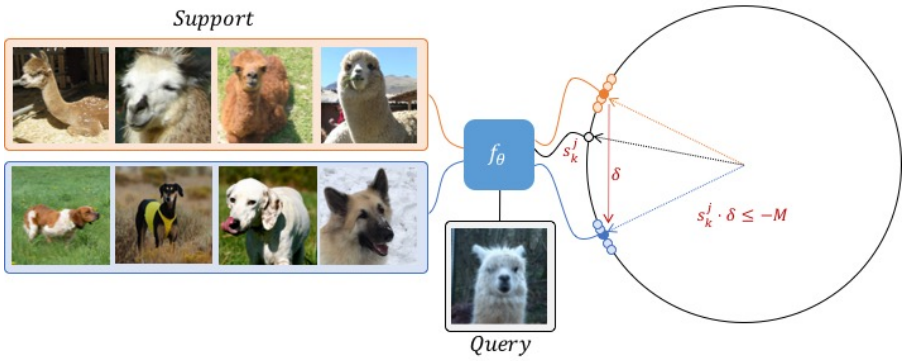

Figure 3: A visualisation of how the embedding is learned. Imagine a simple case where we have 2 tasks (or classes): llamas and dogs. We have a support set of 4 examples, which are then embedded and averaged in order to get a sentence for each task. The hinge rank loss drives the dot product of the query image ($\boldsymbol{s}_k^j$) with the difference between the actual sentence and the negative sentences (δ) to be at least a factor of margin away (M in the Figure above). In other words, $\boldsymbol{s}_k^j$ should point in the opposite direction to δ by at least a factor of margin.

dimensional vector $\boldsymbol{s}_k^j \in \mathbb{R}^N$ for each example $\tau_k^j \in \mathfrak{T}^j$. A combined sentence $\boldsymbol{s}^j \in \mathbb{R}^N$ is then computed for that task by taking the normalised mean of the example vectors:

$$\boldsymbol{s}^j = \left[\frac{1}{K} \sum_{\tau_k^j \in \mathfrak{T}^j} f_\theta(\tau_k^j) \right]^\wedge , \tag{1}$$

where $\boldsymbol{v}^\wedge = \frac{\boldsymbol{v}}{\|\boldsymbol{v}\|}$. We then need to define a loss function that can be used to learn an ideal embedding. We use a combination of the cosine distance between points and the hinge rank loss (inspired by [27]). The loss for a task $\mathfrak{T}^j$ is defined as:

$$\mathcal{L}_{emb} = \sum_{\tau_k^j \in \mathfrak{T}^j} \sum_{\mathfrak{T}^i \neq \mathfrak{T}^j} max[0, margin - \boldsymbol{s}_k^j \cdot \boldsymbol{s}^j + \boldsymbol{s}_k^j \cdot \boldsymbol{s}^i] , \tag{2}$$

which trains the model to produce a higher dot-product similarity between a task's example vectors $\boldsymbol{s}_k^j$ and its sentence $\boldsymbol{s}^j$ than to sentences from other tasks $\boldsymbol{s}^i$. We illustrate the intuition behind this loss in Figure 3.

Additionally, we pick two disjoint sets of examples for every task $\mathfrak{T}^j$: a support set $\mathfrak{T}_U^j$ and a query set $\mathfrak{T}_Q^j$. In the above embedding loss, the support set is used to compute the task sentences, $\boldsymbol{s}^i$ and $\boldsymbol{s}^j$, and only the examples picked from the query set are used as example vectors, $\boldsymbol{s}_k^j$. Given that each of the sampled tasks in a training batch are unique, the negatives $\mathfrak{T}^i$ can be chosen to be all the other tasks in the batch. Therefore, for each task within a batch, we also compare to every other task. Further details are given in Algorithm 1.

In all of our experiments, we set $margin = 0.1$, though in practice we found a wide range of values between $0.01 \leq margin \leq 1.0$ that would work. Although not used, we can treat this embedding as a classification of tasks, whose accuracy we can estimate by computing the sentence $\boldsymbol{s}_k$ of an example and then performing a nearest neighbour search in the embedding space over all task sentences $\boldsymbol{s}^j$. In addition to the dot-product similarity and hinge rank loss, we also tried other distances and losses. One such distance and loss was the squared Euclidean distance used in [22], but we found that this did not work as well for our case.

3.2 Control

In contrast to metric learning systems for classification, which would use some sort of nearest neighbour test to find the matching class, here the embedding is relayed to the control network and both networks are trained jointly. Given a sentence $\boldsymbol{s}_U^j$, computed from the support set $\mathfrak{T}_U^j$, as well as

Algorithm 1 Training loss computation for one batch. B is the batch size, K_U and K_Q are the number of examples from the support and query set respectively, and $RandomSample(S, N)$ selects N elements uniformly at random from the set S.

procedure TRAINING ITERATION
  $\mathcal{B} = RandomSample(\{\mathfrak{T}_1, \cdots, \mathfrak{T}_N\}, \text{B})$
  for $\mathfrak{T}^j \in \mathcal{B}$ **do**
    $\mathfrak{T}_U^j = RandomSample(\mathfrak{T}^j, K_U)$
    $\mathfrak{T}_Q^j = RandomSample(\mathfrak{T}^j \backslash \mathfrak{T}_U^j, K_Q)$
    $\boldsymbol{s}_U^j = \left[\frac{1}{K_U} \sum_{\tau \in \mathfrak{T}_U^j} f_\theta(\tau)\right]^\wedge$
    $\boldsymbol{s}_q^j = f_\theta(\tau_q) \quad \forall \tau_q \in \mathfrak{T}_Q^j$
  $\mathcal{L}_{emb} = \mathcal{L}_{ctr}^U = \mathcal{L}_{ctr}^Q = 0$
  for $\mathfrak{T}^j \in \mathcal{B}$ **do**
    $\mathcal{L}_{emb} \mathrel{+}= \sum_q \sum_{i \neq j} max[0, margin - \boldsymbol{s}_q^j \cdot \boldsymbol{s}_U^j + \boldsymbol{s}_q^j \cdot \boldsymbol{s}_U^i]$
    $\mathcal{L}_{ctr}^U \mathrel{+}= \sum_{\tau \in \mathfrak{T}_U^j} \sum_{(\boldsymbol{o}, \boldsymbol{a}) \in \tau} \|\pi(\boldsymbol{o}, \boldsymbol{s}_U^j) - \boldsymbol{a}\|_2^2$
    $\mathcal{L}_{ctr}^Q \mathrel{+}= \sum_{\tau \in \mathfrak{T}_Q^j} \sum_{(\boldsymbol{o}, \boldsymbol{a}) \in \tau} \|\pi(\boldsymbol{o}, \boldsymbol{s}_U^j) - \boldsymbol{a}\|_2^2$
  $\mathcal{L}_{tec} = \lambda_{emb}\mathcal{L}_{emb} + \lambda_{ctr}^U \mathcal{L}_{ctr}^U + \lambda_{ctr}^Q \mathcal{L}_{ctr}^Q$
  return $\mathcal{L}_{tec}$

Algorithm 2 How TecNets operate during test time. D is the set of demonstrations for a task, Env is the environment in which to act.

procedure TEST(D, Env)
  $\boldsymbol{s} = \left[\frac{1}{|D|} \sum_{\tau \in D} f_\theta(\tau)\right]^\wedge$
  while task not complete **do**
    $\boldsymbol{o} = Env.GetObservation()$
    $\boldsymbol{a} = \pi(\boldsymbol{o}, \boldsymbol{s})$
    $Env.Act(\boldsymbol{a})$

examples from the query set $\mathfrak{T}_Q^j$ we can compute the following loss for the policy π:

$$\mathcal{L}_{ctr} = \sum_{\tau_q^j \in \mathfrak{T}_Q^j} \sum_{(\boldsymbol{o}, \boldsymbol{a}) \in \tau_q^j} \|\pi(\boldsymbol{o}, \boldsymbol{s}_U^j) - \boldsymbol{a}\|_2^2 \,. \tag{3}$$

This allows the embedding not only to be learned from the embedding loss $\mathcal{L}_{emb}$, but also from the control loss, which can lead to a more meaningful embedding for the control network than if they were trained independently. Though appropriate weightings must be selected, as the control network needs the embedding in order to know which task to perform, but the embedding network may have to wait for a reasonable control output before being able to enrich its structure.

We found it helpful for the control network to also predict the action for the examples in the support set $\mathfrak{T}_U^j$. This has the advantage that it makes the task of learning $\mathcal{L}_{ctr}^Q$ easier, as learning $\mathcal{L}_{ctr}^U$ can be seen as an easier version of minimising the former (since example dependent information can be passed through the embedding space). Thus, the final loss is:

$$\mathcal{L}_{Tec} = \sum_{\mathfrak{T}} \lambda_{emb}\mathcal{L}_{emb} + \lambda_{ctr}^U \mathcal{L}_{ctr}^U + \lambda_{ctr}^Q \mathcal{L}_{ctr}^Q \tag{4}$$

Input to the task-embedding network consists of $(width, height, 3 \times |\tau|)$, where 3 represents the RGB channels. For all of our experiments, we found that we only need to take the first and last frame of an example trajectory τ for computing the task embedding and so discarded intermediate frames, resulting in an input of $(width, height, 6)$. The sentence from the task-embedding network is then tiled and concatenated channel-wise to the input of the control network (as shown in Figure 2), resulting in an input image of $(width, height, 3{+}\text{N})$, where N represents the length of the vector. Pseudocode for both the training and testing is provided in Algorithms 1 and 2 respectively.

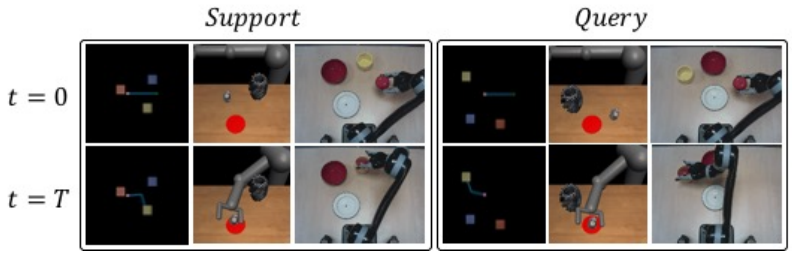

Figure 4: Here we show the first and last timestep of a single example (i.e. one-shot) from the support and query set for each of the 3 experimental domains. The support examples are used to describe the task, whilst the query set examples test the networks ability to perform a modified version of the task. We now highlight each of the tasks in the support set. *Left:* the simulated reaching experiment, where the robot must reach a specified colour. *Centre:* the simulated pushing experiment, where the robot must push a specified object to the red target. *Right:* the real world placing experiment, where the robot must place an item into a specified container.

4 Experiments

In this section, we aim to answer the following: (1) Is it possible to learn a task embedding that can directly be used for visuomotor control? (2) Does our metric loss lead to a better embedding rather than allowing the control network to have free rein on the embedding? (3) How do we compare to a state-of-the-art one-shot imitation learning method? (4) How is our performance affected as we go from one-shot to many-shot? (5) Does this method apply to sim-to-real?

We begin by presenting results from two simulated experimental domains that were put forward for MIL [6]. We then continue to present results for our own experiment where we perform a placing task using a real-world robot arm, similar to that of MIL's third experimental domain. All 3 experiments are shown in Figure 4. For all experiments, we ensure that our control network follows a similar architecture to MIL [6] in order to allow fair comparison. All networks are trained using the ADAM [28] optimiser with a learning rate of 5×10^{-4}, and a batch-size of 64. Further network architecture details are defined in Appendix A. Our approach only uses visual information for the demonstrations whilst MIL reports results where the input demonstrations are given with and without actions and robot arm state. For completeness, we have reported all of MIL's results, but our aim is to compare against the results where only visual information is used for input demonstrations. Qualitative results for our approach can be seen in the video[1].

4.1 Simulated Reaching

The aim of this first experimental domain is to reach a target of a particular colour in the presence of two distractors with different colours. Input to the control network consist of the (current) arm joint angles, end-effector position, and the 80×64 RGB image, whilst the task-embedding network receives only images (first and last). For details regarding data collection, we point the reader to the Appendix of [6]. Our results (presented in Table 1) show that we outperform MIL by a large margin, as well as other variations of our approach. The results show that the embedding loss is vital for the high success rate, with the exclusion leading to a drop in success of over 70%. In addition to the embedding loss, the inclusion of the support loss heavily assists the network in learning the task.

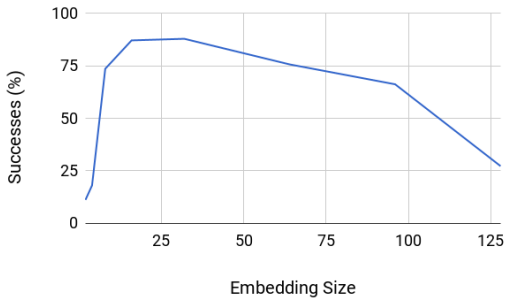

Figure 5: How the percentage of success changes as the size of the embedding varies for the simulated reaching domain.

[1] https://sites.google.com/view/task-embedded-control

	Method	Success (%)
1-Shot	MIL (vision+state+action)	93.00
	MIL (vision)	**29.36***
	Ours (vision)	**86.31**
	Ours ($\lambda^{U}_{ctr} = 0$)	25.68
	Ours ($\lambda_{emb} = 0$)	10.48
	Ours ($\boldsymbol{s} = \vec{0}$)	20.30
	Ours (contextual)	19.17

(a) Simulated Reaching Results

	Method	Success (%)
1-Shot	MIL (vision+state+action)	85.81
	MIL (vision+state)	72.52
	MIL (vision)	**66.44**
	Ours (vision)	**77.25**
	Ours ($\lambda^{U}_{ctr} = 0$)	70.72
	Ours ($\lambda_{emb} = 0$)	58.56
	Ours ($\boldsymbol{s} = \vec{0}$)	02.49
	Ours (contextual)	37.61
5-Shot	MIL (vision+state+action)	88.75
	MIL (vision+state)	78.15
	MIL (vision)	**70.50**
	Ours (vision)	**80.86**
	Ours ($\lambda^{U}_{ctr} = 0$)	72.07
	Ours($\lambda_{emb} = 0$)	67.12
	Ours ($\boldsymbol{s} = \vec{0}$)	-
	Ours (contextual)	-

(b) Simulated Pushing Results

Table 1: The result for both simulated reaching (a) and simulated pushing (b), for both our full solution, and a series of ablations. In the tables, $\lambda^{U}_{ctr} = 0$ refers to excluding the support loss, $\lambda_{emb} = 0$ refers to excluding the embedding loss, $\boldsymbol{s} = \vec{0}$ refers to ignoring the output of the embedding, and instead passing in a zero sentence, and '*contextual*' refers to ignoring the output of the embedding, and passing in one of the support example's images directly to the control network. There is no entry for the final 2 rows of *Table (b)* as these are equivalent to their 1-shot counterpart. *Note that in *Table (a)*, the results reported here for MIL were not reported in the paper, and so the results here are reported from running their publicly available code.

Note that it is possible to achieve 33% on this task by randomly choosing one target to reach for. We believe this is an important note, as it appears that MIL is not capable of learning from one visual demonstration alone on this task, resulting in a policy that randomly selects a colour to reach for. As with MIL, only 1-shot capabilities were explored for this domain.

We also use this experimental domain to see how the embedding size effects the performance, and we show the results in Figure 5. By increasing the embedding size, we are increasing the dimensionality of our vector space, allowing a greater number of tasks to be learned. But as Figure 5 shows, increasing the dimensionality can lead to poor performance. We hypothesise that increasing the embedding size too much can lead to a trivial embedding that does not look for similarities and will thus generalise poorly when encountering new tasks. A balance must be struck between the capacity of the embedding and the risk of overfitting. Although this is an extra hyperparameter to optimise for, Figure 5 encouragingly suggest that this can take on a wide range of values.

4.2 Simulated Pushing

The second experimental domain from [6] involves a simulated robot in a 3D environment, where the action space is 7-DoF torque control. The goal is to push a randomly positioned object to the red target in the presence of another randomly positioned distractor, where the objects have a range of shapes, sizes, textures, frictions, and masses. The control network input consists of a 125×125 RGB image and the robot joint angles, joint velocities, and end-effector pose, whilst the task-embedding network again receives images only. For details regarding data collection, we point the reader to the Appendix of [6]. In both the 1-shot and 5-shot case, our method surpasses MIL when using its few-shot ability on visual data alone. Unlike the previous experiment, excluding the support loss is less detrimental and leads to better results than MIL in both the 1-shot and 5-shot case.

4.3 Real-world Placing via Sim-to-Real

The final experiment tests how well our method works when applied to a real robot. Not only that, but we also look at the potential of our method to be used in a sim-to-real context; where the goal is to learn policies within simulation and then transfer these to the real world with little or no additional training (we focus on the latter). This is an attractive idea, as data collection in the real world is often cumbersome and time consuming.

We run a robotic placing experiment much like the one proposed in MIL, where a robot arm is given the task of placing a held object into a specified container whilst avoiding 2 distractors. The key difference is that our data is collected in simulation rather than real world. As summarised in Figure 1, our TecNet is trained in simulation with a dataset of 1000 tasks with 12 examples per task. Our containers consist of a selection of 178 bowls from the ShapeNet database [29]. To enable transfer, we use domain randomisation; a method that is increasingly being used transfer learned polices from simulation to the real world [3, 30, 31]. We record RGB images of size 160×140 from an external camera positioned above the robot arm, joint angles, and velocities along a planned linear path for each example. During domain randomisation, we vary the lighting location, camera position, table texture, target object textures and target object sizes, and create procedurally generated holding objects. An example of the randomisation can be seen in Figure 1.

Figure 6: The real world test set for the placing domain. Holding objects on the left and placing objects (consisting of bowls, plates, cups, and pots) on the right.

Once the network has been trained, we randomly select one holding object and 3 placing targets from our test set of real-world objects (shown in Figure 6); these objects have not been seen before in either simulation or real world. The robot is shown a single demonstration via human teleoperation using the HTC Vive controller. During demo collection, only RGB images and joint angles are collected. A trial is successful if the held object lands in or on the target container after the gripper ihas opened.

One-shot success rates in the real-world is **72.97%**, and is based on 18 tasks with 4 examples each (72 trials total), showing that we are able to successfully cross the reality-gap and perform one-shot imitation learning. The majority of our failure cases appeared when the target objects were cups or plates, rather than bowls. We imagine this is due to the fact that our training set only consisted of bowls. Results for the real world evaluation can be seen in the video[2], and a visualisation of the learnt embedding can be seen in Appendix B.

5 Conclusion

We have presented TecNets, a powerful few-shot learning approach for end-to-end few-shot imitation learning. The method works by learning a compact description of a task via an embedding network, that can be used to condition a control network to predict action for a different example of the same task. Our approach is able to surpass the performance of MIL [6] for few-shot imitation learning in two experimental domains when only visual information is available. Unlike many other meta-learning approaches, our method is capable of continually learning new tasks without forgetting old ones, and without losing its few-shot ability. Moreover, we demonstrate that the few-shot ability can be trained in simulation and then deployed in the real world. Once deployed, the robot can continue to learn new tasks from single or multiple demonstrations.

Similar to other meta-learning approaches, we expect the approach to perform poorly when the new task to learn is drastically different from the training domain; for example, a TecNet trained to place items in containers would not be expected to learn few-shot pushing. Having said that, if the training set were to include a wide range of tasks, generalising to a broad range of tasks may be possible, and so this is something that should be looked at further. Parallel work has shown extensions to MIL that infer policies from human demonstration [32]. As our method inherently only uses visual information, we are also keen to investigate the inclusion of human demonstrations to our approach.

[2] https://sites.google.com/view/task-embedded-control

Acknowledgments

Research presented in this paper has been supported by Dyson Technology Ltd. We thank the reviewers for their valuable feedback.

References

[1] H. F. Harlow. The formation of learning sets. *Psychological review*, 1949.

[2] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 2016.

[3] S. James, A. J. Davison, and E. Johns. Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task. *Conference on Robot Learning*, 2017.

[4] T. Zhang, Z. McCarthy, O. Jow, D. Lee, K. Goldberg, and P. Abbeel. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. *International Conference on Robotics and Automation*, 2018.

[5] S. James and E. Johns. 3d simulation for robot arm control with deep q-learning. *NIPS Workshop (Deep Learning for Action and Interaction)*, 2016.

[6] C. Finn, T. Yu, T. Zhang, P. Abbeel, and S. Levine. One-shot visual imitation learning via meta-learning. *Conference on Robot Learning*, 2017.

[7] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *International Conference on Machine Learning*, 2017.

[8] S. Schaal. Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences*, 1999.

[9] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 2009.

[10] S. Thrun and L. Pratt. *Learning to learn*. Springer Science & Business Media, 2012.

[11] C. Lemke, M. Budka, and B. Gabrys. Metalearning: a survey of trends and technologies. *Artificial Intelligence Review*, 2015.

[12] D. A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. *Advances in Neural Information Processing Systems*, 1989.

[13] S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. *International Conference on Artificial Intelligence and Statistics*, 2011.

[14] A. Y. Ng, S. J. Russell, et al. Algorithms for inverse reinforcement learning. *International Conference on Machine Learning*, 2000.

[15] P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. *International Conference on Machine learning*, 2004.

[16] C. Finn, S. Levine, and P. Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. *International Conference on Machine Learning*, 2016.

[17] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra, et al. Matching networks for one shot learning. *Advances in Neural Information Processing Systems*, 2016.

[18] G. Koch, R. Zemel, and R. Salakhutdinov. Siamese neural networks for one-shot image recognition. *ICML Deep Learning Workshop*, 2015.

[19] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap. Meta-learning with memory-augmented neural networks. *International Conference on Machine Learning*, 2016.

[20] S. Ravi and H. Larochelle. Optimization as a model for few-shot learning. *International Conference on Learning Representations*, 2017.

[21] E. Triantafillou, R. Zemel, and R. Urtasun. Few-shot learning through an information retrieval lens. *Advances in Neural Information Processing Systems*, 2017.

[22] J. Snell, K. Swersky, and R. Zemel. Prototypical networks for few-shot learning. *Advances in Neural Information Processing Systems*, 2017.

[23] B. Kulis et al. Metric learning: A survey. *Foundations and Trends in Machine Learning*, 2012.

[24] A. Bellet, A. Habrard, and M. Sebban. A survey on metric learning for feature vectors and structured data. *arXiv preprint arXiv:1306.6709*, 2013.

[25] K. Hausman, J. T. Springenberg, Z. Wang, N. Heess, and M. Riedmiller. Learning an embedding space for transferable robot skills. *International Conference on Learning Representations*, 2018.

[26] J. Sung, I. Lenz, and A. Saxena. Deep multimodal embedding: Manipulating novel objects with point-clouds, language and trajectories. 2017.

[27] A. Frome, G. S. Corrado, J. Shlens, S. Bengio, J. Dean, T. Mikolov, et al. Devise: A deep visual-semantic embedding model. *Advances in Neural Information Processing Systems*, 2013.

[28] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representation*, 2015.

[29] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

[30] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, 2017.

[31] J. Matas, S. James, and A. J. Davison. Sim-to-real reinforcement learning for deformable object manipulation. *Conference on Robot Learning*, 2018.

[32] T. Yu, C. Finn, A. Xie, S. Dasari, T. Zhang, P. Abbeel, and S. Levine. One-shot imitation from observing humans via domain-adaptive meta-learning. *Robotics: Science and Systems*, 2018.

[33] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[34] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *International Conference on Learning Representation*, 2016.

[35] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. *International Conference on Medical image computing and computer-assisted intervention*, 2015.

[36] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 2008.

A Experimental Details

In this section we provide additional experiment details, network architecture, and hyperparameters. In all cases the task-embedding network and control network use a convolutional neural network (CNN), where each layer is followed by layer normalisation [33] and an elu activation function [34], except for the final layer, where the output is linear for both the task-embedding and control network. Optimisation was performed with Adam [28] with a learning rate of 5×10^{-4}, and lambdas were set as follows: $\lambda_{emb} = 1.0$, $\lambda^{U}_{ctr} = 0.1$, $\lambda^{Q}_{ctr} = 0.1$.

A.1 Simulated Reaching

The CNN consists of 3 strided convolution layers, each with 40 (3×3) filters, followed by 4 fully-connected layers consisting of 200 neurons. Input consists of a 80×64 RGB image and the robot proprioceptive data, including the arm joint angles and the end-effector position. The proprioceptive is concatenated to the features extracted from the CNN layers of the control network, before being sent through the fully-connected layers. The output of the embedding network is a vector of length 20. The output corresponds to torques applied to the two joints of the arm. The task is considered a success if the end-effector comes within 0.05 meters of the goal within the last 10 timesteps. Further information regarding this task can be accessed from Finn et al. [6].

A.2 Simulated Pushing

The CNN consists of 4 strided convolution layers, each with 16 (5×5) filters, followed by 3 fully-connected layers consisting of 200 neurons. Input consists of a 125×125 RGB image and the robot proprioceptive data, including the joint angles, joint velocities, and end-effector pose. The proprioceptive is concatenated to the features extracted from the CNN layers of the control network, before being sent through the fully-connected layers. The output of the embedding network is a vector of length 20. The output of the control network corresponds to torques applied to the 7 joints of the arm. The task is considered a success if the robot pushes the centre of the target object into the red target circle for at least 10 timesteps within 100-timestep episode. Further information regarding this task can be accessed from Finn et al. [6].

A.3 Real-world Placing

The CNN consists of 4 strided convolution layers, each with 16 (5×5) filters, followed by 4 fully-connected layers consisting of 100 neurons. Input consists of a 125×125 RGB image and the robot proprioceptive data, including just the joint angles. The proprioceptive is concatenated to the features extracted from the CNN layers of the control network, before being sent through the fully-connected layers. The output of the embedding network is a vector of length 20. The output of the control network corresponds to torques applied to the 7 joints of a Kinova Mico 7-DoF arm. There is also an additional auxiliary end-effector position output that is learned via an $L2$ distance between the prediction and the ground truth during simulation training. The task is considered a success if the robot drops the held object into the correct target container.

As a note, we also experimented with using a U-Net architecture [35] for the control network, where the sentence is concatenated to the image features at the bottleneck, but our experiments showed that channel-wise concatenation at the input layer of the control network worked just as well.

B Sim-to-Real Embedding Visualisation

In this section we show t-SNE [36] visualisation of the learnt embedding of the real-world placing task of Section 4.3. Note that the TecNet was trained entirely in simulation without having seen any real-world data. In order to visualise how the embedding looks on real-world data, we collect a dataset of 164 tasks, each consisting of 5 demonstrations. Each demonstration consists of a series of RGB images that were collected via human teleoperation using the HTC Vive controller. Each demonstration in these visualisations are represented via the final frame of that demonstration.

Figure 7: A t-SNE visualisation of the individual sentences of each of the demonstrations learnt by the task-embedding network. We embed 5 demonstrations (without averaging) across each of the 164 tasks. The aim of the visualisation is to illustrate how examples of the same task relate with each other. The result shows that the task-embedding network does indeed learn to place examples of the same tasks next to each other, whilst also placing other, visually similar, tasks nearby.

Figure 8: A t-SNE visualisation of the combined sentences learnt by the task-embedding network. We embed 5 demonstrations and average to get the task sentence for each of the 164 tasks. Given that we are only plotting the combined sentences, this can be seen as a more legible version of Figure 7, focusing on how tasks relate to other tasks, rather than how examples of the same task relate with each other.