DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics

Learning to be Taught

Alexander Koch

DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics

Learning to be Taught

Lernen unterrichtet zu werden

Author:	Alexander Koch
Supervisor:	Prof. Dr.-Ing. habil. Alois C. Knoll
Advisor:	Dr. rer. nat. Zhenshan Bing
Submission Date:	March 15th, 2021

I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, March 11th, 2021 Alexander Koch

Abstract

Deep reinforcement learning (RL) has recently been very successful at learning complex behaviors. But it requires a lot of interactions with the environment to learn a new task. This makes it difficult to apply it to real-world robotic tasks. Meta-reinforcement learning (meta-RL) aims to learn an efficient RL algorithm that can quickly learn a new task. However, in meta-RL the new task is communicated to the agent only with rewards. We think it might be advantageous to give the agent some additional information about the task. In this thesis, we show that meta-RL can be improved by inserting task instructions directly into the observations of the agent. We evaluate our algorithm on the challenging Meta-World benchmarks for robotic manipulation tasks. We demonstrate similar results for two different kinds of task instructions: task demonstrations and language instructions.

Contents

1. Introduction

In recent years, deep reinforcement learning (RL) has been applied very successfully to hard control tasks like playing video games (Berner et al. 2019; Mnih et al. 2015; Silver et al. 2017; Vinyals et al. 2019), acquiring locomotion skills (Barth-Maron et al. 2018; Haarnoja et al. 2018a; Schulman et al. 2015) and robotic manipulation tasks (Akkaya et al. 2019; Andrychowicz et al. 2017; Levine et al. 2016). However, learning these tasks often requires enormous amounts of environment interactions. For example, learning to manipulate a Rubik's cube required 14,000 years of simulated interactions (Akkaya et al. 2019). This makes it impractical for many applications. Humans, on the other hand, are often capable of learning new tasks in a few trials. They have learned during their lifetime to learn new tasks (Wang et al. 2018).

Meta-reinforcement learning (meta-RL) aims to mimic this ability by learning an efficient reinforcement learning algorithm that is able to learn new tasks quickly (Botvinick et al. 2019; Hussein et al. 2017; Wang et al. 2016). Context-based meta-RL achieves this by conditioning the policy on past experience (Rakelly et al. 2019). If the past experience contains the received rewards, the agent can use this information to optimize its actions.

Unfortunately, recent meta-RL algorithms perform very poorly on a diverse set of tasks (Yu et al. 2019). We think, the problem is that they rely solely on rewards to communicate the new task to the agent. This is especially problematic when the rewards are sparse. The agent can only receive rewards after it has already solved the task. If the space of possible tasks is large, this becomes unlikely. When a human worker is given a new task, they are usually told what to do by using language. They do not have to try every possible action sequence to figure out what they are supposed to achieve.

In Natural Language Processing (NLP) multi-task learning has been achieved by inserting a task description directly into the input of the neural network (Brown et al. 2020; Raffel et al. 2019). For example, if the model should do translation, the input to the model would be: "translate English to German: That is good". The supervised learning target for this input would be "Das ist gut" (Brown et al. 2020). This enables the training of a single language model that can do multiple tasks such as text summarization, translation, question answering or text classification.

Inspired by this, we demonstrate that meta-reinforcement learning can be improved by inserting task instructions directly into the observations of the agent. To the best of our knowledge this has not been done before. We demonstrate the effectiveness with two kinds of task instructions: language instructions and task demonstrations. The language instructions are short sentences that describe the task. The words in the sentences can be encoded into real-valued vectors (Pennington et al. 2014). The task demonstrations are generated by using a scripted policy. These instructions are then used as observations for the agent at the beginning of an episode. After that the agent is given multiple trials to solve the task.

Since the agent has to remember the task instructions, we use on-policy Maximum a Posteriori Policy Optimization (V-MPO) (Song et al. 2019) with a Gated Transformer-XL (GTrXL) Model (Parisotto et al. 2019). This combination has shown strong empirical performance on memory tasks. In order to improve sample efficiency and decrease the training time, we modify the V-MPO algorithm slightly to use sampled environment interactions multiple times for gradient updates. We demonstrate the effectiveness on a simple locomotion task.

We evaluate our algorithm on the Meta-World meta learning benchmarks ML10 and ML45 (Yu et al. 2019). These are collections of diverse robotic manipulation tasks. Previous work on these benchmarks has achieved less than 50% success rate on the training tasks and less than 40% on the test tasks. We show that our algorithm achieves almost perfect performance on the training tasks and can solve about half of the test tasks. However, we applied two small changes to the environments to make training possible with our limited computational resources. First, we repeat the actions of the agent twice so that the episodes become shorter. Second, we add the remaining trial time to the observations (Pardo et al. 2018). Our results indicate that a larger variety of training tasks might be necessary to generalize to new tasks. Furthermore, we show that the task performance declines slightly over the three trials. This shows that remembering the instruction over long time scales might still be an issue for the agent.

2. Background

2.1. Reinforcement Learning

Reinforcement learning aims to learn a policy π for a Markov decision process (MDP). The MDP is defined by a tuple (S, A, p, r, γ, T) with a state space S, an action space A, the transition dynamics p, a reward function r, a discount factor γ and a set of terminal states T. The agent starts in an initial state $s_0 \sim p(s_0)$. The transition dynamics $p(s_{t+1}|s_t, a_t)$ determines the probability of transitioning from state s_t to state s_{t+1} if the agent chooses action a_t. The reward function $r : S \times A \rightarrow \mathbb{R}$ determines the reward that the agent receives when taking action a in state s. If there is a transition to a terminal state $s_t \in T$ the sequence ends.

The policy $\pi(a|s)$ defines the probability that the agent takes action a in state s. In deep reinforcement learning the policy $\pi_\theta(a|s)$ is implemented by a Deep Neural Network (DNN) with weights θ. The resulting sequence of environment interactions can be formalized as a trajectory $\tau_{0:n} = [(s_0, a_0, r_1), (s_{t+1}, a_{t+1}, r_{t+2}), \dots, (s_{t+n-1}, a_{t+n-1}, r_{t+n})]$. The probability of the trajectory τ under the policy π is:

$$p(\tau|\pi) = p(s_0) \prod_{t=0} p_\theta(a_t|s_t) r(r_t|s_t, a_t) p(s_{t+1}|s_t, a_t). \tag{2.1}$$

The goal of reinforcement learning is to learn an optimal policy $\pi^\star$ that maximizes the expected sum of discounted rewards:

$$\pi^\star = arg \max_\pi \mathbb{E}_{\tau \sim p(\tau|\pi)} \sum_{t \geq 0} \gamma^t r_t. \tag{2.2}$$

The discount factor $\gamma \in [0, 1]$ ensures that the sum is finite. Many RL algorithms also learn a state-value function $V^\pi : S \rightarrow \mathbb{R}$ that defines the expected cumulative discounted reward when following the policy π:

$$V^{\pi}(s) = \mathbb{E}_{\tau \sim p(\tau|\theta)} \left[\sum_{t \geq 0} \gamma^t r_t | s_0 = s \right]. \tag{2.3}$$

2.2. On-Policy Maximum a Posteriori Policy Optimization (V-MPO)

V-MPO (Song et al. 2019) is a recent on-policy reinforcement learning algorithm that has shown strong asymptotic performance on a variety of tasks. It simultaneously learns a policy $\pi_\theta(a|s)$ and a state-value function V^{π}_{ϕ}. The policy $\pi_{\theta_{old}}$ is used to sample trajectories from the environment. These trajectories are used to form a batch of data D that contains a number of trajectories length n with $|D|$ total state-action samples. This batch is then used to update the policy π_θ. After T_{target} learning steps the sampling weights θ_{old} are updated with the new weights θ. This enables an efficient parallel implementation of the sampling and learning processes. Since the online policy π_θ is updated with trajectories from $\pi_{\theta_{old}}$, the data is slightly off-policy. There are algorithms to correct for this (Espeholt et al. 2018), but the authors decided that this is not necessary for V-MPO.

The complete loss function for the policy network weights θ, the value network weights ϕ and the Lagrange multipliers η, α_μ and α_Σ is the following:

$$\mathcal{L}(\phi, \theta, \eta, \alpha_\mu, \alpha_\Sigma) = \mathcal{L}_V(\phi) + \mathcal{L}_{V-MPO}(\theta, \eta, \alpha_\mu, \alpha_\Sigma). \tag{2.4}$$

Usually, the policy network and the value network share some of their parameters and are optimized together. The policy loss can be further divided:

$$\mathcal{L}_{V-MPO}(\theta, \eta, \alpha_\mu, \alpha_\Sigma) = \mathcal{L}_\pi(\theta) + \mathcal{L}_\eta(\eta) + \mathcal{L}_\alpha(\theta, \alpha). \tag{2.5}$$

The policy loss $\mathcal{L}_\pi$ is a weighted maximum likelihood loss

$$\mathcal{L}_\pi(\theta) = - \sum_{s,a \sim \tilde{D}} \psi(s,a) log \pi_\theta(a|s), \tag{2.6}$$

$$\text{where } \psi(s,a) = \frac{exp(\frac{A^{target}(s,a)}{\eta})}{\sum_{s,a\sim\tilde{D}} exp(\frac{A^{target}(s,a)}{\eta})}. \tag{2.7}$$

The computation of the advantages $A^{target}(s,a)$ is described in the section for the value function loss. The temperature loss $\mathcal{L}_\eta(\eta)$ is:

$$\mathcal{L}_\eta(\eta) = \eta\epsilon_\eta + \eta log\left(\frac{1}{|\tilde{D}|}\sum_{s,a\sim\tilde{D}} exp(\frac{A^{target}(s,a)}{\eta})\right). \tag{2.8}$$

$\tilde{D}$ is the half of the batch with the highest advantages. The computation of the advantages $A^{target}(s,a)$ is described below. The KL constraint $\mathcal{L}_\alpha(\theta,\alpha)$ ensures that the online policy does not change too quickly. This has empirically shown to improve the performance drastically. For continuous action spaces the KL divergence is separated into a mean and a covariance component: $\mathcal{L}_\alpha(\theta,\alpha) = \mathcal{L}^\mu_{\alpha_\mu}(\theta,\alpha_\mu) + \mathcal{L}^\Sigma_{\alpha_\Sigma}(\theta,\alpha_\Sigma)$. This allows us to set different target values for the KL components. The mean and covariance components for a d-dimensional multivariate normal distribution can be computed as follows:

$$D^\mu_{KL}(\pi_{\theta_{old}}||\pi_\theta) = 0.5(\mu_\theta - \mu_{\theta_{old}})^T\Sigma^{-1}_{\theta_{old}}(\mu_\theta - \mu_{\theta_{old}}), \tag{2.9}$$

$$D^\Sigma_{KL}(\pi_{\theta_{old}}||\pi_\theta) = \frac{1}{2}\left[Tr(\Sigma^{-1}_\theta\Sigma_{\theta_{old}}) - d + log\frac{|\Sigma_\theta|}{|\Sigma_{\theta_{old}}|}\right]. \tag{2.10}$$

With these KL divergences the losses $\mathcal{L}^\mu_{\alpha_\mu}(\theta,\alpha_\mu)$ and $\mathcal{L}^\Sigma_{\alpha_\Sigma}(\theta,\alpha_\Sigma)$ can be computed as

$$\begin{aligned}\mathcal{L}^C_{\alpha_C}(\theta,\alpha_C) = \frac{1}{|D|}\sum_{s\in D}\Big[&\alpha_C(\epsilon_{\alpha_C} - sg[[D^C_{KL}(\pi_{\theta_{old}}(a|s)||\pi_\theta(a|s))]]) \\ &+ sg[[\alpha_C]]D^C_{KL}(\pi_{\theta_{old}}(a|s)||\pi_\theta(a|s))\Big],\end{aligned} \tag{2.11}$$

for $C = \mu, \Sigma$. The symbol $sg[[\cdot]]$ denotes a stop gradient. That means that the gradient is not propagated into the enclosed term. $V^\pi_\phi(s)$ is optimized to minimize the squared difference to the standard n-step return from the current time step $G_t^{(n)}$, where

$$\mathcal{L}_V(\phi) = \frac{1}{2|D|}\sum_{s_t \sim D}(V_\phi^\pi(s_t) - G_t^{(n)})^2, \tag{2.12}$$

$$G_l^{(n)} = \sum_{k=l}^{t+n-1} \gamma^{k-l} r_k + \gamma^{t+n-l} V_\phi^\pi(s_{t+n}). \tag{2.13}$$

The advantages for action a_t in state s_t can then be computed as $A^\pi(s_t, a_t) = G_t^{(n)} - V_\phi^\pi(s_t)$.

Algorithm 1 V-MPO

```
1: policy $\pi_\theta(a|s)$, state-value function $V_\phi^\pi(s)$
2: while not converged do
3:     Update $\pi_{\theta_{old}} \leftarrow \pi_\theta$
4:     for learning step $l = 1..T_{target}$ do
5:         Use $\pi_{\theta_{old}}$ to generate a batch $D$ with $B$ trajectories of length n
6:         Compute loss $\mathcal{L}(\phi, \theta, \eta, \alpha_\mu, \alpha_\Sigma)$ from batch $D$
7:         Update $\phi, \theta, \eta, \alpha_\mu, \alpha_\Sigma$ with gradient step
```

2.3. Transformer

The Transformer (Vaswani et al. 2017) is a neural network architecture that was developed for natural language processing. It performs better on sequence to sequence learning tasks than recurrent architectures like the Long Short-Term Memory (Hochreiter et al. 1997). The improvement is especially large on sequences with long-term dependencies. Furthermore, it is much easier to parallelize. This has enabled much faster training on modern GPUs.

It uses a self-attention operation called scaled dot-product attention. The inputs consist of a matrix of queries Q, keys K with dimension $T \times d_k$ and values V with dimension $T \times d_v$ where T is the sequence length. The attention operation can then be written as

$$Attention(Q, K, V) = MaskedSoftmax(\frac{QK^T}{\sqrt{d_k}})V. \tag{2.14}$$

The Multi Head Attention Block uses h of these self-attention operations and concatenates their outputs.

$$MultiHeadAttention(Q,K,V) = Concat(head_1, head_2, .., head_h)W^O, \tag{2.15}$$

$$\text{where } head_i = Attention(QW_i^Q, KW_i^K, VW_i^V). \tag{2.16}$$

The full Transformer layer adds a position-wise feed-forward network and Layer Normalization (Ba et al. 2016). $E^0 \in \mathbb{R}^{T \times D}$ is the input sequence with size D and length T.

TransformerLayer(E^{l-1}):

$$\begin{aligned} \bar{Y}^l &= LayerNorm(E^{l-1} + MultiHeadAttention(E^{l-1}, E^{l-1}, E^{l-1}) \\ E^l &= LayerNorm(\bar{Y}^l, ReLU(W_1\bar{Y}^l + b_1)W_2 + b_2))). \end{aligned} \tag{2.17}$$

The learnable weight matrices are $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{model}}$. Usually the dimensions are chosen to be $d_k = d_v = d_{model}/h$.

2.3.1. Transformer-XL

The time complexity of the original Transformer grows quadratically with the input sequence length. The Transformer-XL (Dai et al. 2020) was developed to allow the model to capture longer term dependencies without increasing the sequence length. It allows every layer to attend to the output of the last layer for the current segment and from the last segment. That means layer l receives the output of the previous layer on the current segment E^{l-1} and the memory M^{l-1} which is E^{l-1} from the last segment. E^0 is the input segment. Thus a Transformer-XL with N layers can look back at $T \times N$ time steps.

One complete layer of the Transformer-XL can be written as follows (Parisotto et al. 2019):

TransformerXlLayer(M^{l-1}, E^{l-1}):

$$\begin{aligned}
\tilde{E}^{l-1} &= [sg[[M^{l-1}]], E^{l-1}] \\
Q^l, K^l, V^l &= W_Q^l E^{l-1}, W_K^l \tilde{E}^{l-1}, W_V^l \tilde{E}^{l-1} \\
R &= W_R^l \Phi \\
A_{htm}^l &= Q_{htd} K_{hmd} + Q_{htd} R_{hmd} + u_{h*d} K_{htm} + v_{h*d} R_{hmd} \\
W_{htm}^l &= MaskedSoftmax(\frac{A^l}{\sqrt{d_k}}) \\
\hat{Y}^l &= Concat(W_{htm}, V_{hmd}, axis = h) W^O \\
\bar{Y}^l &= LayerNorm(E^{l-1} + \hat{Y}^l) \\
E^l &= LayerNorm(\bar{Y}^l, ReLU(W_1 \bar{Y}^l + b_1) W_2 + b_2))).
\end{aligned} \tag{2.18}$$

Einstein summation notation is used here for the tensor products. u and v are trainable vectors and Φ is a sinusoid encoding matrix (Vaswani et al. 2017). This allows the model to use positional information.

2.3.2. Gated Transformer XL

The original Transformer architecture does not work well for reinforcement learning (Mishra et al. 2017). That is why the Gated Transformer-XL (GTrXL) has been developed (Parisotto et al. 2019). It is a Transformer-XL with two critical changes. First, it applies Layer Normalization only on the input to the Attention module. The residual connections form a path without any Layer Normalization layers. The second change is an added gating layer that combines the residual connections with the output of the Attention module. In this work we will use the Gated-Recurrent-Unit-type gating function that was inspired by the Gated Recurrent Unit (GRU) (Cho et al. 2014). The gating output is computed as $g^l(x, y) = (1 - z) \odot x + z \odot \hat{h},$ where

$$r = \sigma(W_R^l y + U_r^l x),\ z = \sigma(W_z^l y + U_z^l x - b_g^l),\ \hat{h} = tanh(W_g^l y + U_g^l (r \odot x)). \tag{2.19}$$

The bias b_g is usually initialized > 0. This causes the gating function to increase the weight of the residual connection initially. Thus the agent's policy depends mostly on the observation of the current time step. This makes it easier to learn a good policy. In this thesis, we will initialize it with $b_g = 2$ as it was done it the initial work. Here is the algorithm for a full GTrXL layer:

$$\text{GTrXL}_{\text{Layer}}(M^{l-1}, E^{l-1}):$$

$$\begin{aligned}
\tilde{E}^{l-1} &= LayerNorm([sg[[M^{l-1}]], E^{l-1}]) \\
Q^l, K^l, V^l &= W_Q^l E^{l-1}, W_K^l \tilde{E}^{l-1}, W_V^l \tilde{E}^{l-1} \\
R &= W_R^l \Phi \\
A_{htm}^l &= Q_{htd} K_{hmd} + Q_{htd} R_{hmd} + u_{h*d} K_{htm} + v_{h*d} R_{hmd} \\
W_{htm}^l &= MaskedSoftmax(\frac{A^l}{\sqrt{d_k}}) \\
\hat{Y}^l &= Concat(W_{htm}, V_{hmd}, axis = h) W^O \\
\bar{Y}^l &= g_{MHA}^l (E^{l-1}, ReLU(\hat{Y}^l)) \\
\bar{E}^l &= ReLU(W_1 LayerNorm(\bar{Y}^l) + b_1) W_2 + b_2) \\
E^l &= g_{MLP}^l (\bar{Y}^l, \bar{E}^l).
\end{aligned} \tag{2.20}$$

2.4. Pop-Art Reward Normalization

The scale of rewards can vary greatly between environments. This usually makes the learning of a state-value function much more difficult. Preserving outputs precisely, while adaptively rescaling targets (Pop-Art) (Van Hasselt et al. 2016) can be used to normalize the learning targets for the value function for every task individually.

The unnormalized value function for a single task can be written as

$$f_{\theta,\Sigma,\mu}(x) = \sigma n_\theta(x) + \mu = \sigma(W h_{\theta/W,b} + b) + \mu, \tag{2.21}$$

where h is the neural network with the weights θ. W, b, σ and μ are parameters that are learned for every task individually. μ and σ can then be updated to track the mean and standard deviation of the returns G_t for one task:

$$\mu_t = (1 - \beta)\mu_{t-1} + \beta G_t \qquad \sigma_t = \sqrt{\nu_t - \mu_t^2}, \qquad \nu_t = (1 - \beta)\nu_{t-1} + \beta (G_t)^2. \tag{2.22}$$

Now we also have to update $w \rightarrow w'$ and $b \rightarrow b'$ in order to keep the learning problem stationary:

$$w' = \frac{\sigma}{\sigma'} w, \qquad \sigma' = \frac{\sigma b + \mu - \mu'}{\sigma'}. \tag{2.23}$$

It has been demonstrated that this kind of reward normalization is very important for multi-task learning (Hessel et al. 2019). If the agent receives rewards for one task with much larger magnitudes, it will neglect tasks with smaller rewards. Pop-Art can normalize the rewards for each task individually. Thus the agent learns every task with the same priority.

2.5. Meta-Reinforcement Learning

The goal of meta-reinforcement learning (meta-RL) is to learn from a set of training tasks $\mathcal{D}_{\mathcal{T}}^{train}$ to quickly learn a good policy for tasks from a test set $\mathcal{D}_{\mathcal{T}}^{test}$. Tasks in both sets are drawn from a distribution over MDPs $p(\mathcal{T})$. The learning phase on $\mathcal{D}_{\mathcal{T}}^{train}$ is called meta training. The test phase on $\mathcal{D}_{\mathcal{T}}^{test}$ is called meta testing. The formal objective is to learn an optimal policy $\pi_{\theta^\star}$ that maximizes the expected sum of rewards on the test tasks

$$\theta^\star = arg\max_{\theta} \left[\mathbb{E}_{\mathcal{T}\sim\mathcal{D}_{\mathcal{T}}^{test}} \left[\sum_{t\geq 0} \gamma^t r_t \right] \right]. \tag{2.24}$$

3. Related Work

There are two popular types of meta-reinforcement learning. Context-based meta-RL learns a policy that is conditioned on the past trajectory. Gradient-based meta-RL uses gradients to change the weights to learn a new task.

3.1. Context-based meta-RL

In Context-based meta-RL the policy is conditioned on the past trajectory $\pi(a|\tau_{0:t})$. Since the trajectory contains the rewards, the agent can learn to maximize the rewards within an episode.

One approach is to use classical RL algorithms with a recurrent neural network (RNN) (Hussein et al. 2017; Wang et al. 2016). The recurrent connections allow the agent to memorize past observations.

Simple Neural Attentive Meta Learner (Mishra et al. 2017) uses temporal convolutions and Attention layers instead of a recurrent model. They state that they were unable to train a model using only attention layers. However, later it has been shown that reinforcement learning using only attention layers can achieve state of the art results on several environments (Parisotto et al. 2019).

PEARL (Rakelly et al. 2019) is another approach that uses a variational autoencoder to learn a latent representation of the task. It then uses the Soft-Actor Critic RL algorithm (SAC) (Haarnoja et al. 2018a,b) to learn a policy conditioned on this latent representation.

3.2. Gradient-based meta-RL

Gradient-based meta-RL algorithms use gradients to adapt quickly to new tasks. Model Agnostic Meta Learning (MAML) (Finn et al. 2017a) has pioneered this type of meta-RL. It learns weights θ so that after one gradient step it finds good task specific parameters θ'. The objective is to learn:

$$min_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i}), \tag{3.1}$$

$$\text{where } \theta'_i = \theta - \alpha \nabla \theta \mathcal{L}_{\mathcal{T}_i}(f_{\theta}). \tag{3.2}$$

The task specific loss is just a classical RL loss. In the original work (Finn et al. 2017a) the REINFORCE loss is used in the inner loop (Williams 1992) and Trust-Region Policy Optimization (TRPO) (Schulman et al. 2015) is used in the meta optimization.

3.3. Learning from Demonstrations

Learning from Demonstrations (LfD) aims to use task demonstrations to support learning. Behavioral Cloning (BC) is a simple algorithm that trains the policy using supervised learning with the demonstration actions (Pomerleau 1991; Ross et al. 2011).

Meta-imitation learning (Finn et al. 2017b) uses MAML to imitate a demonstration of a robotic manipulation task with visual observations. It uses the demonstration actions as supervised learning targets in the inner loss. This work has also been extended to use human demonstrations (Yu et al. 2018). One-Shot imitation learning (Duan et al. 2017) uses a specialized attention model to learn to imitate a block stacking policy.

Watch-Try-Learn (Zhou et al. 2019) first learns a policy conditioned on a demonstration. After that, this policy is used to interact with the environment and generate a trajectory that contains the received rewards. After that, a second policy is trained that uses the demonstration and the trial to improve over the first trial.

Generative Adversarial Imitation Learning (GAIL) (Ho et al. 2016) learns a policy and a discriminator. The discriminator is trained to distinguish between the demonstrations and the learned policy. The output of the discriminator is then used as a reward signal to train the policy. Goal-conditioned GAIL (Ding et al. 2019) extends GAIL by conditioning the policy and the discriminator on the goal states. Furthermore, it gives the agent an additional reward for reaching the goal.

3.4. Language Instructions for robotic manipulation tasks

Concept2Robot (Shao et al. 2020) first trains an action recognition model. The action classification prediction is used as a reward signal for the agent. After that, multiple single-task policies are trained on a set of tasks. The policy model combines a Transformer model for the language instructions with a convolutional neural network for visual observations to output the control actions. After all single-task policies have been learned, a multi-task model is trained to imitate all single-task policies. Abramson et al. 2020 have investigated combinations of Behavioral Cloning, GAIL and other auxiliary losses to learn to execute language instructions.

4. Methodology

4.1. Problem Statement

In this thesis we focus on meta-reinforcement learning on continuous action spaces with additional task information and a variable number of trials. We define a trial as the environment interactions between resets of the task MDP where the agent tries to solve the task. The additional task information should help the agent to figure out the task. That is why we will also call it an instruction. It can be, for example, a demonstration of the task or a language instruction. The task information should be able to be encoded into a sequence of real-valued vectors $[x_1, x_2, \dots, x_n]$ where $x_i \in \mathbb{R}^d, d \in \mathbb{N}$.

Since the agent has access to the task instruction, rewards are not necessary during meta testing. This is especially helpful in a sim-to-real setting where the robot is trained in a simulation and tested in the real world. Usually, it is much easier to provide rewards in a simulation than in the real world.

Our problem statement is very general and special cases have been researched extensively. For example, if the additional task information is a demonstration and the agent has one trial, this is similar to many imitation learning algorithms (Duan et al. 2017; Finn et al. 2017b; Pathak et al. 2018). If the agent gets two trials, this resembles Watch-Try-Learn (Zhou et al. 2019). In the case of multiple demonstrations and one trial, our problem setting is similar to the one used for Task-Embedded Control Networks (James et al. 2018).

In contrast to most imitation learning algorithms, we do not assume access to the robot actions for the demonstrations. Many imitation learning algorithms use the robot actions during the demonstration for supervised learning. For our algorithm we only need some kind of demonstration to communicate the task. It could be, for example, the sequence of joint positions of the robot during the demonstration. This can be formalized as $[d_1, d_2, \dots, d_T]$ for a demonstration of length T, where $d_t \in \mathbb{R}^n$ are the joint positions at time t and n is the number of joints. If the additional task information is a language instruction and the agent has one trial, it is similar to Concept2Robot (Shao et al. 2020). However, we do not assume access to an action classifier.

4.2. Algorithm

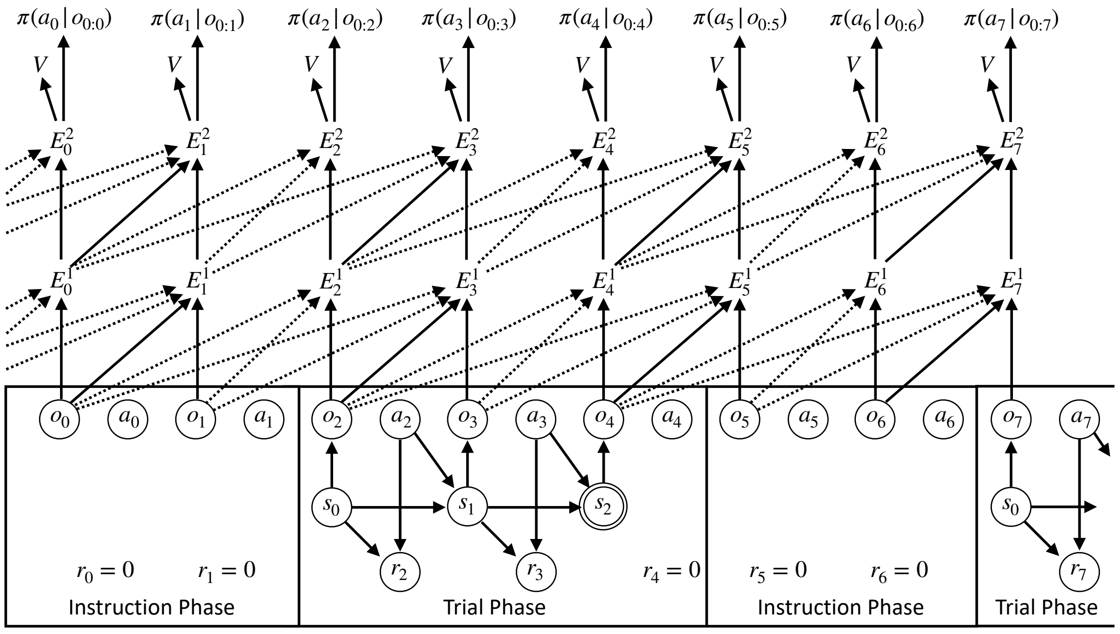
Prior work on learning from demonstrations and learning from language instructions uses specialized algorithms. Concept2Robot, for example, only works with language instructions. Meta-imitation learning cannot use multiple trials. Watch-Try-Learn was developed to work with exactly two trials but cannot easily be extended to more trials. Moreover, meta-imitation learning and Watch-Try-Learn cannot be used with language instructions.

We propose a simple and general algorithm that can use different types of task information and is able to use a variable number of trials. The basic idea is to encode additional task information into the observations of the agent. The agent is trained on the training environments through episodes. At the beginning of an episode, a new training task environment is sampled $\tau \sim \mathcal{D}_{\tau}^{train}$. The episode starts with an instruction phase. During this phase the agent observations contain the additional task information. The actions from the agent are simply ignored and the rewards are set to zero in this phase. We describe in the next two sections how the observations can be generated during this phase.

After the instruction phase, a trial phase is started. During this phase the agent interacts with the task MDP until a terminal state is reached. If the agent solved the task successfully in the trial, another trial phase is started after the task environment is reset.

In the case of an unsuccessful trial another instruction phase starts directly after the trial phase ends. This resembles a real world scenario where a human operator might try a slightly different instruction to communicate the task to the agent. After a fixed number of finished trial phases the episode ends. Then a new task environment is sampled and the procedure repeats. This algorithm is visualized in Figure 4.1.

Since the agent might need the instructions at a much later time step during the trial phase, we have to use a model architecture that works well on long sequences. That is why we use V-MPO with a Gated Transformer-XL model to train the agent. This combination has shown strong performance on difficult memory tasks (Parisotto et al. 2019). In order to improve the training time, we slightly modify the V-MPO algorithm



$\pi(a_0 | o_{0:0})$
$\pi(a_1 | o_{0:1})$
$\pi(a_2 | o_{0:2})$
$\pi(a_3 | o_{0:3})$
$\pi(a_4 | o_{0:4})$
$\pi(a_5 | o_{0:5})$
$\pi(a_6 | o_{0:6})$
$\pi(a_7 | o_{0:7})$
V
E_0^2 E_1^2 E_2^2 E_3^2 E_4^2 E_5^2 E_6^2 E_7^2
E_0^1 E_1^1 E_2^1 E_3^1 E_4^1 E_5^1 E_6^1 E_7^1
o_0 a_0 o_1 a_1 o_2 a_2 o_3 a_3 o_4 a_4 o_5 a_5 o_6 a_6 o_7 a_7
s_0 s_1 s_2 s_0
$r_0 = 0$ $r_1 = 0$ r_2 r_3 $r_4 = 0$ $r_5 = 0$ $r_6 = 0$ r_7
Instruction Phase
Trial Phase
Instruction Phase
Trial Phase

Figure 4.1.: Overview of our algorithm. Actions a_t are sampled from the distribution $\pi(a_t|o_{0:t})$. The dotted lines indicate how the memory segment in the GTrXL influences the hidden states. The memory segment before the first observation of the episode is initialized as as a sequence of zero vectors. States with a double circle are terminal states. In our experiments we use a larger size of the GTrXL such that the agent can still use the observations from the first instruction phase to compute the last few actions of an episode.

to reuse sampled environment interactions more often. The modifications are described in section 4.5.

For all experiments, we also concatenate a binary indicator to the trial observations that shows whether the task was solved successfully. If the size of the observations during the instruction phase differ from the observation size during the trial phase, we concatenate an appropriately sized zero vector to the observations during one phase to ensure an equal observation shape during both phases.

4.3. Language Instructions

For every task we create a set of several different simple language instructions. These instructions are short sentences that describe the task. At the beginning of the instruction phase, a new language instruction will be sampled for the current task. Then we use the GloVe algorithm (Pennington et al. 2014) to encode the sentence words into a sequence of vectors $[w_0, \dots, w_T]$ with $w_i \in \mathbb{R}^{50}$, where the sentence is T words long. During the instruction phase, the observations will then contain the word encodings. This means that at time step t of the instruction phase the observation will be w_t. After T time steps the trial phase will start.

4.4. Demonstrations

Our experiments with task demonstrations as additional information are very similar to our algorithm with language instructions. During the instruction phase, a scripted policy is used to control the robot so that it solves the task. The observations during this phase contain the same type of information as during the trial phase. For example, if the normal environment observations contain the robot joint positions, the instruction phase observations will contain the joint observations during the demonstration. Since this might make it difficult for the agent to determine the current phase, we add a binary flag to the observations of both phases that indicates the current phase.

4.5. V-MPO with Improved Sample Efficiency

V-MPO is very sample inefficient. This means that it requires a lot of environment interactions during training. This usually causes the environment interactions to be the bottleneck during training. Thus, if we can improve the sample efficiency, we could train the agent faster. We improve the sample efficiency by modifying the V-MPO algorithm slightly to reuse sampled environment interactions more often.

The original V-MPO algorithm uses every environment trajectory only for one gradient update. We change this by keeping a small FIFO buffer with the last $T_{target} \times B$ trajectories, where B is the batch size for the gradient updates. Then we randomly sample batches from this buffer. The ratio of trajectories sampled from this buffer over the new trajectories added to the buffer will be called the replay ratio. As in the original V-MPO algorithm, we update the neural network that interacts with the environment after T_{Target} gradient updates.

Algorithm 2 Sample Efficient V-MPO

```
1: policy $p_\theta(a|s)$, state-value function $V^\pi_\phi(s)$
2: initialize FIFO buffer B with capacity $B * T_{target}$
3: while not converged do
4:     Update $\pi_{\theta_{old}} \leftarrow \pi_\theta$
5:     for learning step $l = 1..T_{target}$ do
6:         Use $\pi_{\theta_{old}}$ to generate $B$ trajectories of length n and add to B
7:         Sample B trajectories from B
8:         Compute loss $\mathcal{L}(\phi, \theta, \eta, \alpha_\mu, \alpha_\Sigma)$ from batch $B$
9:         Update $\phi, \theta, \eta, \alpha_\mu, \alpha_\Sigma$ with gradient step
```

5. Experiments

5.1. Sample Efficient V-MPO with increased Replay Ratio

In this experiment, we show that V-MPO with an increased replay ratio is more sample efficient. We show the results on the Ant-v3 environment from the OpenAI gym benchmark collection (Brockman et al. 2016). The task is to make a four-legged robot run as fast as possible. Further training details are described in Appendix B.1.

The results (see Figure 5.1) show that a higher replay ratio results in a better sample efficiency. But there are no significant efficiency gains for replay ratios higher than four. For all following experiments, we will use a replay ratio of four.

5.2. Meta-World

Meta-World is a collection of 50 diverse robotic manipulation tasks (Yu et al. 2019) build on the MuJoCo physics simulator (Todorov et al. 2012). It contains the ML45 and ML10 meta-RL benchmarks. The ML45 consists of 45 training tasks and 5 test tasks. The tasks are visualized in Figure 5.2. The ML10 contains a subset of the ML45 training tasks which are split into 10 training tasks and 5 test tasks.

Most tasks contain some kind of object that has to be manipulated with the robot arm. Within every task the starting position of the object can vary. Every task has a function that evaluates whether the task has been solved successfully. After a certain number of time steps the trial ends. This is independent of whether the task was solved.

The observations consist of a vector that contains the 3D position of the end-effector and the position of the object if it exists. The actions are the desired 3D end-effector positions and commands for the gripper. For each task a well shaped reward function is provided with a similar structure for all tasks. This reduces the exploration problem

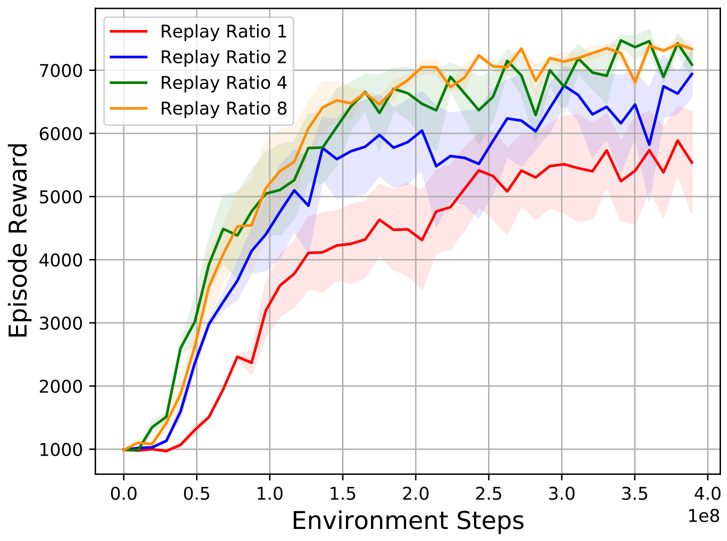


Figure 5.1.: V-MPO tested with different replay ratios on Ant-v3. Every configuration was trained with three random seeds and the shaded regions indicate one standard deviation.

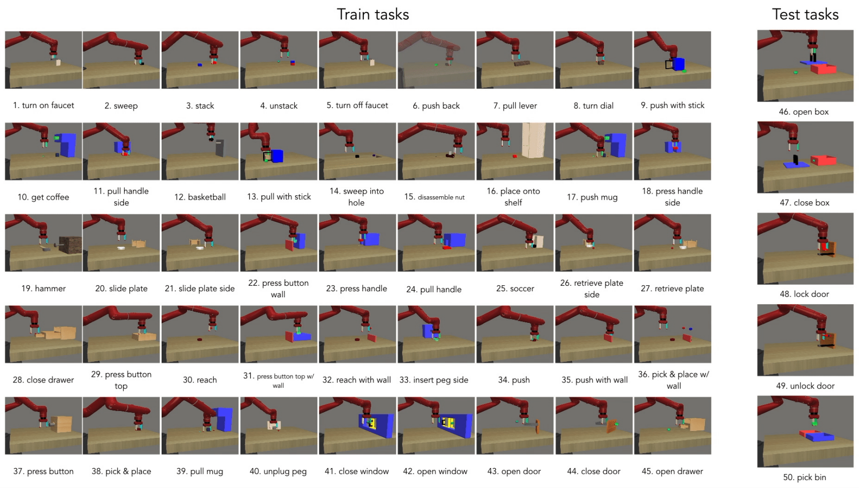


Figure 5.2.: Meta-World ML45 Benchmark (Yu et al. 2019)

and makes the tasks individually solvable for recent RL algorithms.

We make two small changes to the Meta-World benchmarks to reduce the training cost. In all experiments we repeat actions twice during the trial phase to reduce the trial length (Bellemare et al. 2012). This enables a shorter sequence length for the Transformer model which reduces the compute requirements significantly. The reported number of environment steps in our results corresponds to the number of observations the agent has seen. Furthermore, we add a scalar to the observations during the trial phase that indicates the remaining time in the trial. During the instruction phase, a zero is concatenated to the observation instead. This helps the agent to learn a value function because the Meta-World environments have a time dependent termination condition (Pardo et al. 2018). The time observation is computed as (step in trial) / (maximum steps per trial). We will also show the training progress with a different time observation in an ablation study.

5.2.1. ML10 Benchmark

We use the Meta-World ML10 benchmark to show that our algorithm performs better when it is shown a task demonstration or a language instruction instead of only receiving rewards. The task demonstrations are generated with a scripted policy that performs the task. The agent then receives the observations generated by the Meta-World environment. The language instructions are short sentences that describe the task. For every task we designed multiple simple language instructions. The exact language instructions are listed in appendix D.

We compare this to a version of our algorithm that only observes rewards and no instructions. This means an episode consists only of three trials and no instruction phase. The rewards are simply concatenated to the observations. This is similar to many other recent context-based meta-RL algorithms. Our algorithm will only observe whether the task was solved successfully. Further details are listed in Appendix C.

The results show that the agents that receive additional task information learn significantly faster and are able to solve almost all training tasks in all three trials (see Figure 5.3). On the test tasks, the agents solve only just over half of the trials with demonstrations and even less with language instructions. The agent that observes only rewards is able to solve only about 70% of the training trials and about 30% of the test trials.

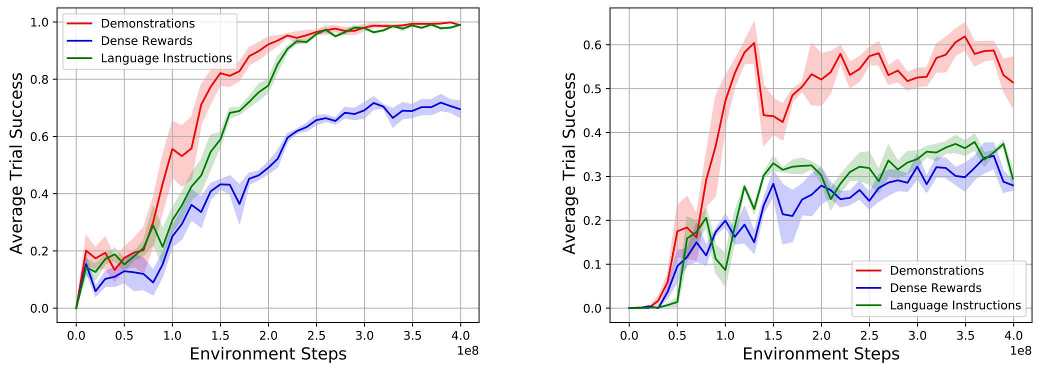


Figure 5.3.: Meta-World ML10 training progress with demonstrations and with dense rewards. The shaded regions indicate one standard deviation over three training runs. The left plot shows the results on the training tasks. On the right plot the results on the test tasks are visualized.

We also show the performance differences of the agent over the three trials. To measure this we use the number of time steps in a trial until the task is solved successfully (see Figure 5.4). We expected to see that the agent gets better with every successive trial. However, we can not detect any improvement after more trials. This indicates that the agent has trouble using information from previous trials.

Furthermore, the results show that on the training tasks the agent solves the task faster than the demonstration shown to it. This indicates that the agent did not just copy the demonstration.

5.2.2. ML10 Ablation

We make several ablations to our algorithm on ML10 with demonstrations to emphasize the importance of some design decisions.

First, we demonstrate the importance of the Pop-Art reward normalization. This normalizes the rewards for every task individually. Thus the advantages are at a similar magnitude for all tasks and the agent is encouraged to solve all tasks. If the reward magnitudes were different between tasks, the agent might focus only on tasks with high reward. Furthermore, the advantages are normalized which reduces the variance in the gradient magnitudes. The results in Figure 5.5 demonstrate that Pop-Art is very

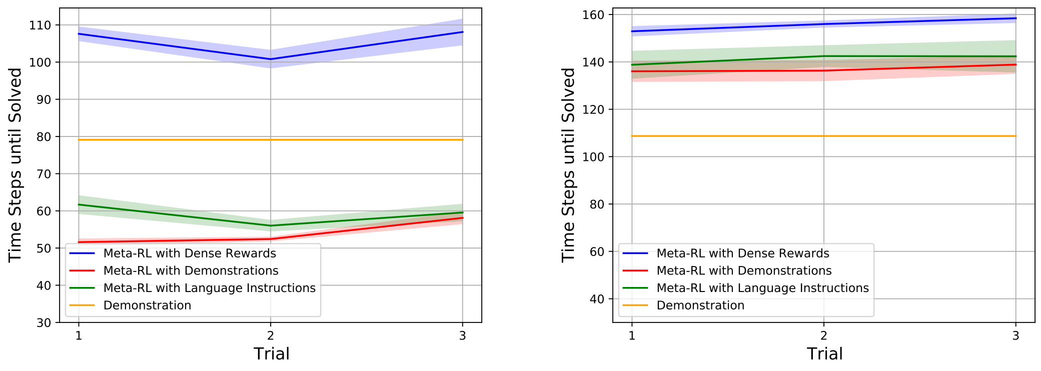


Figure 5.4.: Meta-World ML10 success rate over test task trials. The vertical axis shows the average number of time steps the agent took until the task is solved successfully. The left plot shows the results on the training tasks. On the right plot the performance on the test tasks is visualized.

important for our algorithm. Without this the agent does not learn to solve more than half of the training tasks.

Another ablation is to use a different time observation. Our algorithm observes the remaining time in the current trial. Here we evaluate our algorithm when it observes the remaining time in the full episode. This was originally proposed by Pardo et al. 2018. The results show that this version learns the training tasks much slower and solves less test tasks successfully. Our hypothesis is that the discount factor causes the value function during a trial to be relatively independent of the next trial rewards. This means that the remaining time in the trial is much more important for the value function than the remaining time in the episode. The rewards at the beginning of a trial are often very small in Meta-World environments and during an instruction phase the agent does not receive any rewards at all.

We also show the learning progress when the agent observes its previous action which is common in other context-based meta-RL algorithms (Hussein et al. 2017; Wang et al. 2016). The previous action is concatenated to the observation during the trial phase. During the instruction phase this part is set to zero. The results show that the training progress is much slower in this case. We are not sure what the reason for this worsening is. However, the performance on the test tasks is almost as good as without previous action observation.

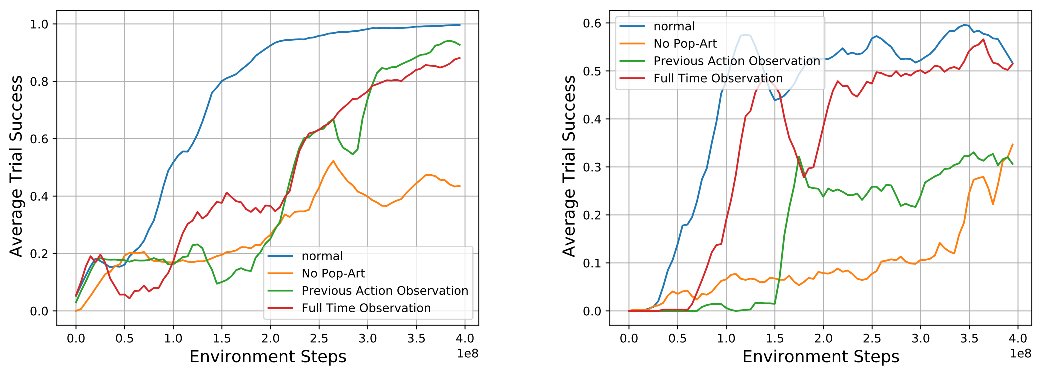


Figure 5.5.: Meta-World ML10 ablations task success rate. The policy is evaluated every 5 million environment steps and averaged over 5 consecutive evaluations. The left plot shows the results on the training tasks. On the right plot the training progress on the test tasks is visualized.

5.2.3. ML45 Benchmark

The Meta-World ML45 benchmark consists of 45 training tasks and 5 test tasks. We compare our algorithm with task demonstrations and language instructions. We use the same hyperparameters and the same number of trials as for the ML10 benchmark. However, we train the agent for over 1.5 billion time steps instead of just 400 million because the benchmark contains more diverse tasks. Since the training takes much longer than on the ML10 benchmark, we were only able to test both configurations with one random seed. The results (see Figure 5.6) show that both versions are able to learn almost all training tasks and about 40% of the test tasks.

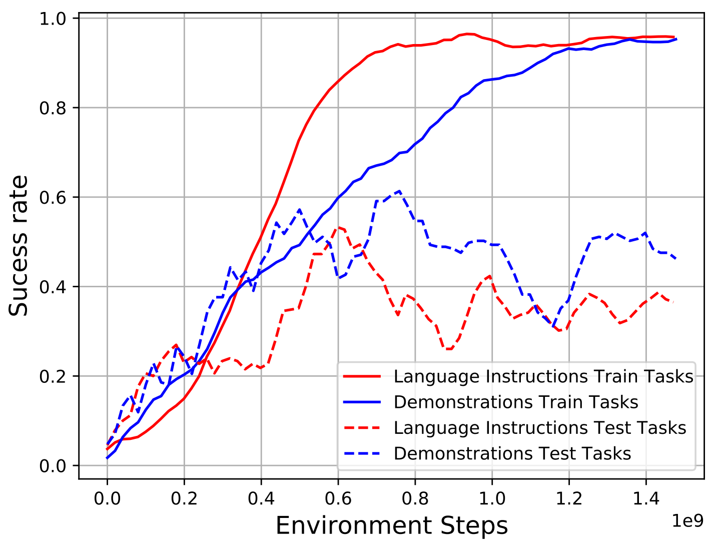


Figure 5.6.: Average trial success rate on ML45 training and test tasks. The policy is evaluated every 5 million environment steps and averaged over 5 consecutive evaluations.

6. Discussion

6.1. Applicability and Advantages

Our algorithm is very general and can be applied to a wide variety of Meta-Reinforcement problems. We have shown that it works with demonstrations and with language instructions. Moreover, it can also be used with other types of task instructions. Anything that helps the agent to identify the task is useful. For example, a human might demonstrate only the most important part of the task and the agent infers the complete task. It is also possible to use combinations of different types of instructions. The robot could be first shown a demonstration of the task and then receive feedback after a trial by using language.

Another advantage is that it does not require dense rewards during meta-testing. We have demonstrated this in our experiments with the Meta-World meta learning benchmarks. This allows it to be applied to real-world robotic tasks where dense rewards might be difficult to provide. For example, the dense rewards in Meta-World (Yu et al. 2019) require the position of movable objects which are difficult to compute in the real world.

Furthermore, task instructions often contain information on how to solve the task in contrast to only rewards as task information. This makes it much easier for the agent to learn a new task. We have shown in section 5.2.1 that our algorithm learns to solve more tasks when it receives a demonstration instead of dense rewards.

6.2. Limitations and Drawbacks

A major problem is that the agent has to remember the task information for a long time. If the agent receives an instruction at the beginning of an episode it usually has to memorize this information for all trials. In section 5.2.1 we show that the agent gets worse with every trial.

Another problem is the required amount of compute resources. The Transformer models are very large in comparison to other neural networks used for reinforcement learning. And they are very inefficient during sampling. The Transformer architecture was optimized for learning a full sequence in parallel and not sampling only one output during evaluation. Recurrent Neural Networks are much more efficient for this part.

Furthermore, V-MPO is an on-policy RL algorithm that is very sample inefficient. That means, it needs a large amount of environment interactions during training. Training in the real world is thus very impractical. We have shown in section 5.1 that the sample efficiency of V-MPO can be improved by reusing recent samples for learning. However, it still needs multiple orders of magnitude more samples than recent off-policy RL algorithms like Soft Actor-Critic (Haarnoja et al. 2018b) or Twin Delayed Deep Deterministic Policy Gradient (Fujimoto et al. 2018).

7. Conclusion & Future Work

7.1. Conclusion

In this thesis, we showed that meta-reinforcement learning can be improved by providing the agent with additional task information. By encoding the information in the observations, we designed a very simple and general algorithm. This makes it unnecessary to give the agent rewards during meta testing. Other task information, like language instructions, are often much easier to provide than dense rewards. This simplifies the application to real-world robotic tasks. Furthermore, we demonstrated that this enables a recent on-policy RL algorithm combined with a Transformer model to solve a set of very diverse robotic manipulation tasks.

7.2. Future Work

A straightforward extension of our work is to use visual observations to solve the Meta-World environments. An advantage of this is that visual observations also contain information about the shape and orientation of objects. The low dimensional observations used in this thesis only contain the position of one object. However, the increase in observation size would probably enlarge the learning difficulty and expand the training time. But it has been shown that V-MPO can learn to control a humanoid with a high dimensional action space just from visual observations (Parisotto et al. 2019).

Additionally, using visual observations would make it easier to apply it on a real-world robot since the object position does not have to be computed. However, the poor sample efficiency of V-MPO makes it impractical to do meta training in the real world. Domain randomization (Akkaya et al. 2019; Tobin et al. 2017) aims to randomize the simulation settings so that the agent can generalize to the real world. This might make it possible to train the agent in the simulation and test it in the real world.

Another solution might be to use a more sample efficient RL algorithm. Model-based reinforcement learning aims to learn a model of the environment and use this during action selection. This makes the algorithms usually several orders of magnitude more sample efficient than on-policy RL. This is especially helpful for meta-RL since the model is usually the same for all tasks. Recent work has also shown comparable performance to model-free RL (Hafner et al. 2019, 2020; Kozakowski et al. 2020; Nagabandi et al. 2019).

Our experiments have shown that the generalization to the test tasks is still poor. A simple solution might be to develop more training tasks. However, creating new tasks in a simulation is very difficult. But there are algorithms that can automatically generate new tasks (Plappert et al. 2021).

The algorithm we designed is very general and can use any additional information that might help the agent. Thus it could be trained using human-like feedback for every trial or even during the trials. For example, a human might watch the trial and tell the agent how to improve. They could, for example, demonstrate only the specific part of the task where the robot failed. However, this will probably be very expensive because it requires human supervision during training.

A. Implementation Details

We use PyTorch (Paszke et al. 2019) for training Deep Neural Networks. We implement V-MPO within the rlpyt reinforcement learning library (Stooke et al. 2019). The sampler and learner are executed in parallel to efficiently use CPUs and GPUs. The sampler instantiates multiple environments for every CPU hardware thread and sends the observations to a central action selection process. All the observations are collected into a large batch and then fed into the DNN on a GPU. In order to optimally use this GPU, two sampler groups are created that alternate environment steps and action selection. All samplers use mixed precision for inference (Narang et al. 2017). The learner uses full precision for training because it is not the limiting factor during training. We use a RTX 3080 and GTX 1080TI GPUs and a CPU with 24 hardware threads. On this hardware, we achieve about 7000 environment steps per second with the GTrXL architecture. This results in a training time of a little over three days for one of our ML45 experiments.

B. V-MPO Common Hyperparameters

Here are the hyperparameters for V-MPO that were common for all our experiments. They are mostly the same as in Parisotto et al. 2019.

Setting	
Learning rate	1×10^{-4}
initial η	1.0
initial α_μ	1.0
initial α_Σ	1.0
ϵ_η	0.01
ϵ_{α_μ}	0.0075
ϵ_{α_Σ}	1×10^{-5}
T_{target}	100
Agent discount	0.99

B.1. Sample Efficient V-MPO OpenAI Gym Experiments

We use the same hyperparameters and network architecture for the Ant-v3 environment as Song et al. 2019 used for the OpenAI gym environments. The neural network architecture consists of a shared feed-forward neural network with hidden layer sizes of 512 and 256. The output is then fed into two feed-forward networks for the value and policy output. Both networks have a hidden layer size of 256. The output of the policy network indicates the mean and standard deviation for the action distribution. The standard deviation is additionally transformed by a softmax function to ensure positive values.

Setting	
Batch Size	64
Unroll length	40

C. Meta-World Training Details

For all experiments on Meta-World we use the same model architecture. The core consists of a Gated Transformer-XL. The output of the Transformer model is used as input for value and policy networks. These two are implemented as a feed-forward neural network with a single hidden layer of size 256. The policy network outputs the mean and standard deviation for a normal distribution with diagonal covariance. The mean is transformed by a tanh function and the standard deviation is fed through a softmax function to ensure positive values.

The segment length of the transformer is chosen to be equal to the length of the sampled trajectories. This enables the computation of all losses in exactly one forward pass. During sampling the memory is updated periodically after the last action from the current segment was computed. The training samples then contain the trajectories between the memory segment updates. At the beginning of an episode, the memory segment is initialized as zero vectors.

On the ML10 experiments we apply Layer Normalization (Ba et al. 2016) to the observations before feeding them into the model. This is necessary because the dense rewards which are encoded in the observations of one configuration can become huge which results in numerical instabilities.

The task demonstrations are generated with scripted policies from the official Meta-World repository at: http://github.com/rlworkgroup/metaworld. We let the scripted policy interact with the environment at every time step during the instruction phase. But we keep only every fifth observation for the demonstration sequence. Thus the demonstration is 40 time steps long. During the trial we repeat the actions twice as in the other Meta-World experiments.

Three tasks had to be removed from the benchmarks because the scripted policy was not working. This includes peg-insert-side-v1, lever-pull-v1 and bin-picking-v1. Another task, sweep-v1, had to be removed because the reward function did not encourage the agent to solve the task. These problems are now fixed in the version 2 environments of

Meta-World. Unfortunately, they were not yet available at the time of our experiments.

	Dense Rewards	Language Instructions	Demonstrations
Transformer Layers	.	6	.
Transformer Hidden Dim	.	64	.
Transformer Num Heads	.	4	.
Replay Ratio	.	4	.
Batch Size	.	128	.
Segment Length	64	64	75
Memory length	64	64	75

D. Language Instructions for Meta-World

Here are all the language instructions listed that we used. Every word except "goal_pos" is encoded into a 50-dimensional real-valued vector using the GloVe algorithm (Pennington et al. 2014). We use the pre-trained word vectors from http://nlp.stanford.edu/projects/glove/ that were trained on 6 billion tokens. "goal_pos" is encoded by concatenating the 3D goal position in the current task with zeros to obtain a 50-dimensional vector.

Task	Instructions
reach-v1	reach to goal_pos, reach goal_pos
push-v1	push goal_pos, push to goal_pos, push object to goal_pos, push block to goal_pos
pick-place-v1	pick and place at goal_pos, pick object and place at goal_pos
door-open-v1	pull goal_pos, open door, pull to goal_pos
drawer-open-v1	pull goal_pos, pull to goal_pos, pull back to goal_pos
drawer-close-v1	push goal_pos, push to goal_pos, push forward to goal_pos
button-press-topdown-v1	push object down to goal_pos, press button, press down, press button down
window-open-v1	push right to goal_pos, push object right, slide object left, open window
window-close-v1	push left to goal_pos, push object left, slide object right, close window
door-close-v1	push to goal_pos, close door, push from left
reach-wall-v1	reach over obstacle to goal_pos, reach goal_pos, reach to goal_pos
pick-place-wall-v1	pick object and place at goal_pos, pick and place at goal_pos
push-wall-v1	push object around obstacle to goal_pos, push to goal_pos, push object to goal_pos
button-press-v1	press forward, press button forward, push to goal_pos
button-press-topdown-wall-v1	press behind obstacle, press down to goal_pos, press down
button-press-wall-v1	press button behind obstacle, press forward, press forward to goal_pos
peg-unplug-side-v1	pull object to right, pull object to goal_pos, pick object and pull to right
disassemble-v1	pick and pull up, pick and place at goal_pos, pick and put down at goal_pos
hammer-v1	push to goal_pos with object, push to goal_pos with hammer, use object to push to goal_pos
plate-slide-v1	push to goal_pos, push plate to goal_pos, push forward to goal_pos
plate-slide-side-v1	push left to goal_pos, push plate left to goal_pos, slide left to goal_pos
plate-slide-back-v1	push back to goal_pos, push plate back to goal_pos, slide back to goal_pos
plate-slide-back-side-v1	push right to goal_pos, push plate right to goal_pos, slide right to goal_pos
handle-press-v1	push down, press down, push down to goal_pos, press down to goal_pos
handle-pull-v1	pull up, push up, push up to goal_pos, pull up to goal_pos
handle-press-side-v1	push down, press down, push down to goal_pos, press down to goal_pos
handle-pull-side-v1	pull up, push up, push up to goal_pos, pull up to goal_pos
stick-push-v1	push with object to goal_pos, use stick to push to goal_pos, push with stick to goal_pos
stick-pull-v1	push with object to goal_pos, use stick to push to goal_pos, push with stick to goal_pos
basketball-v1	pick and place at goal_pos, pick ball and place at goal_pos, pick ball and place at goal_pos from the top
soccer-v1	push to goal_pos, push ball to goal_pos, place object at goal_pos, place ball at goal_pos
faucet-open-v1	open faucet, push to goal_pos, push right, push from left to right, turn object from left to right, turn to goal_pos
faucet-close-v1	close faucet, push to goal_pos, push left, push from right to left, turn object from right to left, turn to goal_pos
coffee-push-v1	push object to goal_pos, push cup to goal_pos, push object forward to goal_pos
coffee-pull-v1	place object away, pick and place at goal_pos, place cup away
coffee-button-v1	push forward, press button, press coffee button
sweep-into-v1	sweep into hole, push to goal_pos, push block to goal_pos, push block into hole
pick-out-of-hole-v1	pick object, pick object out of hole, pick and place at goal_pos
assembly-v1	pick and place at goal_pos, assemble, pick object and place down at goal_pos
shelf-place-v1	pick and place at goal_pos, place on shelf at goal_pos, pick object and place on shelf at goal_pos
push-back-v1	push back to goal_pos, push to goal_pos, puch block to goal_pos, push block back to goal_pos
dial-turn-v1	turn object, dial turn, rotate object
box-close-v1	pick object and place at goal_pos, pick and place at goal_pos
hand-insert-v1	sweep object to goal_pos, pick object and place at goal_pos, push object to goal_pos, push block to goal_pos
door-lock-v1	push object down to goal_pos, turn object down to goal_pos
door-unlock-v1	push object to goal_pos, turn object to goal_pos

List of Figures

Bibliography

Abramson, J., A. Ahuja, A. Brussee, F. Carnevale, M. Cassin, S. Clark, A. Dudzik, P. Georgiev, A. Guy, T. Harley, F. Hill, A. Hung, Z. Kenton, J. Landon, T. Lillicrap, K. Mathewson, A. Muldal, A. Santoro, N. Savinov, V. Varma, G. Wayne, N. Wong, C. Yan, and R. Zhu (2020). "Imitating Interactive Intelligence." In: pp. 1–96. arXiv: 2012.05672.

Akkaya, I., M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, and N. Tezak (2019). "SOLVING RUBIK'S CUBE WITH A ROBOT HAND." In: pp. 1–51. arXiv: arXiv:1910.07113v1.

Andrychowicz, M., F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba (2017). "Hindsight experience replay." In: *Advances in Neural Information Processing Systems* 2017-Decem.Nips, pp. 5049–5059. ISSN: 10495258. arXiv: 1707.01495.

Ba, J. L., J. R. Kiros, and G. E. Hinton (2016). "Layer Normalization." In: arXiv: 1607.06450.

Barth-Maron, G., M. W. Hoffman, D. Budden, W. Dabney, D. Horgan, D. TB, A. Muldal, N. Heess, and T. Lillicrap (2018). "Distributed Distributional Deterministic Policy Gradients." In: arXiv: 1804.08617.

Bellemare, M. G., J. Veness, and M. Bowling (2012). "Investigating contingency awareness using Atari 2600 games." In: *Proceedings of the National Conference on Artificial Intelligence* 2, pp. 864–871.

Berner, C., G. Brockman, B. Chan, V. Cheung, P. P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. De Oliveira Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang (2019). "Dota 2 with large scale deep reinforcement learning." In: *arXiv*. arXiv: 1912.06680.

Botvinick, M., S. Ritter, J. X. Wang, Z. Kurth-Nelson, C. Blundell, and D. Hassabis (2019). *Reinforcement Learning, Fast and Slow*. DOI: 10.1016/j.tics.2019.02.006.

Brockman, G., V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba (2016). "OpenAI Gym." In: pp. 1–4. arXiv: 1606.01540.

Brown, T. B., B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan,

R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei (2020). “Language Models are Few-Shot Learners.” In: arXiv: `2005.14165`.

Cho, K., B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio (2014). “Learning phrase representations using RNN encoder-decoder for statistical machine translation.” In: *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, pp. 1724–1734. DOI: `10.3115/v1/d14-1179`. arXiv: `1406.1078`.

Dai, Z., Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov (2020). “Transformer-XL: Attentive language models beyond a fixed-length context.” In: *ACL 2019 - 57th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, pp. 2978–2988. DOI: `10.18653/v1/p19-1285`. arXiv: `1901.02860`.

Ding, Y., C. Florensa, M. Phielipp, and P. Abbeel (2019). “Goal-conditioned Imitation Learning.” In: NeurIPS. arXiv: `1906.05838`.

Duan, Y., M. Andrychowicz, B. Stadie, J. Ho, J. Schneider, I. Sutskever, P. Abbeel, W. Zaremba, and B. A. Research Lab (2017). “One-Shot Imitation Learning.” In:

Espeholt, L., H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, B. Yotam, F. Vlad, H. Tim, I. Dunning, S. Legg, and K. Kavukcuoglu (2018). “IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures.” In: *35th International Conference on Machine Learning, ICML 2018*.

Finn, C., P. Abbeel, and S. Levine (2017a). “Model-agnostic meta-learning for fast adaptation of deep networks.” In: *34th International Conference on Machine Learning, ICML 2017* 3, pp. 1856–1868. arXiv: `arXiv:1703.03400v3`.

Finn, C., T. Yu, T. Zhang, P. Abbeel, and S. Levine (2017b). “One-Shot Visual Imitation Learning via Meta-Learning.” In: arXiv: `1709.04905`.

Fujimoto, S., H. Van Hoof, and D. Meger (2018). “Addressing Function Approximation Error in Actor-Critic Methods.” In: *35th International Conference on Machine Learning, ICML 2018* 4, pp. 2587–2601. ISSN: 1938-7228. arXiv: `1802.09477`.

Haarnoja, T., A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine (2018a). “Soft Actor-Critic Algorithms and Applications.” In: arXiv: `1812.05905`.

Haarnoja, T., A. Zhou, P. Abbeel, and S. Levine (2018b). “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor.” In: arXiv: `1801.01290`.

Hafner, D., T. Lillicrap, J. Ba, and M. Norouzi (2019). “Dream to Control: Learning Behaviors by Latent Imagination.” In: arXiv: `1912.01603`.

Hafner, D., T. Lillicrap, M. Norouzi, and J. Ba (2020). “Mastering atari with discrete world models.” In: *arXiv*, pp. 1–24. ISSN: 23318422. arXiv: `2010.02193`.

Hessel, M., H. Soyer, L. Espeholt, W. Czarnecki, S. Schmitt, and H. van Hasselt (2019). "Multi-task deep reinforcement learning with PopArt." In: *33rd AAAI Conference on Artificial Intelligence, AAAI 2019, 31st Innovative Applications of Artificial Intelligence Conference, IAAI 2019 and the 9th AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019*, pp. 3796–3803. ISSN: 2159-5399. DOI: `10.1609/aaai.v33i01.33013796`. arXiv: `1809.04474`.

Ho, J. and S. Ermon (2016). "Generative adversarial imitation learning." In: *Advances in Neural Information Processing Systems*, pp. 4572–4580. ISSN: 10495258. arXiv: `1606.03476`.

Hochreiter, S. and J. Schmidhuber (1997). "Long Short-Term Memory." In: *Neural Computation* 9.8, pp. 1735–1780. ISSN: 08997667. DOI: `10.1162/neco.1997.9.8.1735`.

Hussein, A., M. M. Gaber, E. Elyan, and C. Jayne (2017). *Imitation Learning*. Tech. rep. 2, pp. 1–35. DOI: `10.1145/3054912`.

James, S., M. Bloesch, and A. J. Davison (2018). "Task-embedded control networks for few-shot imitation learning." In: *arXiv* CoRL. ISSN: 23318422. arXiv: `1810.03237`.

Kozakowski, P., S. Levine, A. Mohiuddin, and R. Sepassi (2020). "Model Based Reinforcement Learning For Atari." In: *ICLR*. arXiv: `arXiv:1903.00374v4`.

Levine, S., C. Finn, T. Darrell, and P. Abbeel (2016). "End-to-end training of deep visuomotor policies." In: *Journal of Machine Learning Research* 17, pp. 1–40. ISSN: 15337928. arXiv: `1504.00702`.

Mishra, N., M. Rohaninejad, X. Chen, and P. Abbeel (2017). "A Simple Neural Attentive Meta-Learner." In: arXiv: `1707.03141`.

Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis (2015). "Human-level control through deep reinforcement learning." In: *Nature* 518.7540, pp. 529–533. ISSN: 14764687. DOI: `10.1038/nature14236`.

Nagabandi, A., K. Konoglie, S. Levine, and V. Kumar (2019). "Deep dynamics models for learning dexterous manipulation." In: *arXiv*, pp. 1–12. ISSN: 23318422. arXiv: `1909.11652`.

Narang, S., G. Diamos, E. Elsen, P. Micikevicius, J. Alben, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu (2017). "Mixed precision training." In: *arXiv*, pp. 1–12. ISSN: 23318422. arXiv: `1710.03740`.

Pardo, F., A. Tavakoli, V. Levdik, and P. Kormushev (2018). "Time limits in reinforcement learning." In: *35th International Conference on Machine Learning, ICML 2018* 9, pp. 6443–6452. arXiv: `1712.00378`.

Parisotto, E., H. F. Song, J. W. Rae, R. Pascanu, C. Gulcehre, S. M. Jayakumar, M. Jaderberg, R. L. Kaufman, A. Clark, S. Noury, M. M. Botvinick, N. Heess, and R.

Hadsell (2019). “Stabilizing Transformers for Reinforcement Learning.” In: pp. 1–20. arXiv: 1910.06764.

Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala (2019). “PyTorch: An Imperative Style, High-Performance Deep Learning Library.” In: NeurIPS. arXiv: 1912.01703.

Pathak, D., P. Mahmoudieh, G. Luo, P. Agrawal, D. Chen, F. Shentu, E. Shelhamer, J. Malik, A. A. Efros, and T. Darrell (2018). “Zero-shot visual imitation.” In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops* 2018-June, pp. 2131–2134. ISSN: 21607516. DOI: 10.1109/CVPRW.2018.00278. arXiv: arXiv:1804.08606v1.

Pennington, J., R. Socher, and C. D. Manning (2014). “GloVe: Global Vectors for Word Representation.” In: *Conference on Empirical Methods in Natural Language Processing (EMNLP)* 19.5, pp. 417–425. ISSN: 00047554.

Plappert, M., R. Sampedro, T. Xu, I. Akkaya, V. Kosaraju, P. Welinder, R. D’Sa, A. Petron, H. P.d. O. Pinto, A. Paino, H. Noh, L. Weng, Q. Yuan, C. Chu, and W. Zaremba (2021). “Asymmetric self-play for automatic goal discovery in robotic manipulation.” In: arXiv: 2101.04882.

Pomerleau, D. A. (1991). “Efficient Training of Artificial Neural Networks for Autonomous Navigation.” In: *Neural Computation* 3.1, pp. 88–97. ISSN: 0899-7667. DOI: 10.1162/neco.1991.3.1.88.

Raffel, C., N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. L. Peter, and J. Liu (2019). “Exploring the limits of transfer learning with a unified text-to-text transformer.” In: *arXiv* 21, pp. 1–67. arXiv: 1910.10683.

Rakelly, K., A. Zhou, D. Quiilen, C. Finn, and S. Levine (2019). “Efficient off-policy meta-reinforcement learning via probabilistic context variables.” In: *36th International Conference on Machine Learning, ICML 2019* 2019-June, pp. 9291–9301. arXiv: 1903.08254.

Ross, S., G. J. Gordon, and J. A. Bagnell (2011). “A reduction of imitation learning and structured prediction to no-regret online learning.” In: *Journal of Machine Learning Research* 15, pp. 627–635. ISSN: 15324435. arXiv: 1011.0686.

Schulman, J., S. Levine, P. Moritz, M. Jordan, and P. Abbeel (2015). “Trust region policy optimization.” In: *32nd International Conference on Machine Learning, ICML 2015* 3, pp. 1889–1897. arXiv: 1502.05477.

Shao, L., T. Migimatsu, Q. Zhang, K. Yang, and J. Bohg (2020). “Concept2Robot: Learning Manipulation Concepts from Instructions and Human Demonstrations.” In: *Robotics: Science and Systems (RSS)*.

Silver, D., T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis (2017). "Mastering chess and shogi by self-play with a general reinforcement learning algorithm." In: *arXiv*, pp. 1–19. arXiv: `1712.01815`.

Song, H. F., A. Abdolmaleki, J. T. Springenberg, A. Clark, H. Soyer, J. W. Rae, S. Noury, A. Ahuja, S. Liu, D. Tirumala, N. Heess, D. Belov, M. Riedmiller, and M. M. Botvinick (2019). "V-MPO: On-Policy Maximum a Posteriori Policy Optimization for Discrete and Continuous Control." In: pp. 1–19. arXiv: `1909.12238`.

Stooke, A. and P. Abbeel (2019). "rlpyt: A Research Code Base for Deep Reinforcement Learning in PyTorch." In: pp. 1–12. arXiv: `1909.01500`.

Tobin, J., R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel (2017). "Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World." In: arXiv: `arXiv:1703.06907v1`.

Todorov, E., T. Erez, and Y. Tassa (2012). "MuJoCo: A physics engine for model-based control." In: *IEEE International Conference on Intelligent Robots and Systems*, pp. 5026–5033. ISSN: 21530858. DOI: `10.1109/IROS.2012.6386109`.

Van Hasselt, H., A. Guez, M. Hessel, V. Mnih, and D. Silver (2016). "Learning values across many orders of magnitude." In: *Advances in Neural Information Processing Systems* Nips, pp. 4294–4302. ISSN: 10495258. arXiv: `1602.07714`.

Vaswani, A., G. Brain, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin (2017). *Attention Is All You Need*. Tech. rep. arXiv: `1706.03762v5`.

Vinyals, O., I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver (2019). "Grandmaster level in StarCraft II using multi-agent reinforcement learning." In: *Nature* 575.7782, pp. 350–354. ISSN: 14764687. DOI: `10.1038/s41586-019-1724-z`.

Wang, J. X., Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick (2016). "Learning to reinforcement learn." In: arXiv: `1611.05763`.

Wang, J. X., Z. Kurth-Nelson, D. Kumaran, D. Tirumala, H. Soyer, J. Z. Leibo, D. Hassabis, and M. Botvinick (2018). "Prefrontal cortex as a meta-reinforcement learning system." In: *Nature Neuroscience* 21.6, pp. 860–868. ISSN: 15461726. DOI: `10.1038/s41593-018-0147-8`.

Williams, R. J. (1992). "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning." In: *Machine Learning* 8.3, pp. 229–256. ISSN: 15730565. DOI: `10.1023/A:1022672621406`.

Yu, T., C. Finn, A. Xie, S. Dasari, T. Zhang, P. Abbeel, and S. Levine (2018). "One-shot imitation from observing humans via domain-adaptive meta-learning." In: *6th International Conference on Learning Representations, ICLR 2018 - Workshop Track Proceedings*. arXiv: `arXiv:1802.01557v1`.

Yu, T., D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine (2019). "Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning." In: CoRL, pp. 1–18. arXiv: `1910.10897`.

Zhou, A., E. Jang, D. Kappler, A. Herzog, M. Khansari, P. Wohlhart, Y. Bai, M. Kalakrishnan, S. Levine, and C. Finn (2019). "Watch, Try, Learn: Meta-Learning from Demonstrations and Reward." In: arXiv: `1906.03352`.