CoMic: Complementary Task Learning & Mimicry for Reusable Skills Supplementary Material

Leonard Hasenclever [1] **Fabio Pardo** [2] **Raia Hadsell** [1] **Nicolas Heess** [1] **Josh Merel** [1]

1. Tasks

All tasks use the MuJoCo physics simulator (Todorov et al., 2012) and are simulated with a timestep of 5ms. We use a humanoid body adapted from the "CMU humanoid" available at dm_control/locomotion (Merel et al., 2019a). We adjusted limb lengths, masses, and dynamic properties of the body to make it more consistent with an average human. The humanoid is controlled with a control timestep of 30ms.

1.1. Motion Capture Tracking Task

In this section, we give additional details on the multi-clip motion capture tracking task used in this work. The task is broadly similar to others used in prior work on motion capture tracking (Peng et al., 2018; Merel et al., 2019a; Chentanez et al., 2018; Peng et al., 2019). Our task is available in the dm_control/locomotion package.

Associated with an instance of this task is an underlying set of reference motion capture clips. The important aspects of the tasks are initialization, reward function, termination conditions and observations.

Initialization At the start of each episode we randomly select a starting frame from all frames in the underlying set of clips (excluding the last 10 frames from each clip). At the beginning of each episode the humanoid is initialized to the target pose in the selected frame.

Observations The agent receives both proprioceptive observations as well as information about the target pose in the active mocap clip. The proprioceptive observations are the joint angles, joint angular velocities, a velocimeter observation, a gyro observation, the end effector positions, the 'up' direction in the frame of the humanoid, the actuator state, as well as touch sensors on the hands, fingers and feet and torque sensors in both shoulders.

The observations about the target poses consist of the relative target positions and orientations of different body parts in the local frame. We provide the agent with target poses $\{\hat{s}_{t+1}, \ldots \hat{s}_{t+5}\}$ for the next 5 timesteps (similarly to Chentanez et al., 2018; Merel et al., 2019b).

Reward function and terminations. The reward function captures how close the pose of the simulated humanoid is to the respective target pose. Similar to previous work (Peng et al., 2018; Merel et al., 2019a) we found results to be relatively sensitive to the reward function. Our reward function contains five different terms:

$$r = \frac{1}{2} r_{\text{trunc}} + \frac{1}{2}\left(0.1 r_{\text{com}} + r_{vel} + 0.15 r_{\text{app}} + 0.65 r_{\text{quat}}\right)$$

The first reward term r_{trunc} penalizes large deviations from the reference in joint angles and the euclidean position of a set of 13 different body parts:

$$r_{\text{trunc}} = 1 - \frac{1}{\tau}(\overbrace{\|b_{\text{pos}} - b^{\text{ref}}_{\text{pos}}\|_1 + \|q_{\text{pos}} - q^{\text{ref}}_{\text{pos}}\|_1}^{:=\varepsilon}),$$

where b_{pos} and $b^{\text{ref}}_{\text{pos}}$ correspond to the body positions of the simulated character and the mocap reference and q_{pos} and $q^{\text{ref}}_{\text{pos}}$ correspond to the joint angles. This reward term is linked to the termination condition of our tracking task. Given a termination threshold τ, we terminate an episode if $\varepsilon > \tau$. Note that this ensures that $r_{\text{trunc}} \in [0, 1]$. We found that including this termination condition and the coupled reward speeds up training on larger clip sets but does not by itself lead to visually appealing tracking behavior. We used $\tau = 0.3$

The second reward term is similar to the objective proposed in Peng et al. (2018) with terms penalizing deviations in terms of the center of mass, the joint angle velocities, the end effector positions and the joint orientations but uses slightly different weights. The first term r_{com} penalizes deviations from the centre of mass:

$$r_{\text{com}} = \exp\left(-10\|p_{\text{com}} - p^{\text{ref}}_{\text{com}}\|^2\right), \qquad (1)$$

where p_{com} and $p^{\text{ref}}_{\text{com}}$ are the positions of the centre of mass of the simulated character and the mocap reference, respectively. The second term r_{vel} penalizes deviations from the

[1]DeepMind, London [2]Imperial College, London, work done during an internship at DeepMind. Correspondence to: Leonard Hasenclever <leonardh@google.com>.

Proceedings of the 37th International Conference on Machine Learning, Online, PMLR 119, 2020. Copyright 2020 by the author(s).

reference joint angle velocities:

$$r_{\text{vel}} = \exp\left(-0.1\|q_{\text{vel}} - q_{\text{vel}}^{\text{ref}}\|^2\right), \quad (2)$$

where q_{vel} and $q_{\text{vel}}^{\text{ref}}$ are the joint angle velocities of the simulated character and the mocap reference, respectively. The third term r_{app} penalizes deviations from the reference end effector positions:

$$r_{\text{app}} = \exp\left(-40\|p_{\text{app}} - p_{\text{app}}^{\text{ref}}\|^2\right), \quad (3)$$

where p_{app} and $p_{\text{app}}^{\text{ref}}$ are the end effector positions of the simulated character and the mocap reference, respectively. Finally, r_{quat} penalizes deviations from the reference in terms of the quaternions describing the joint orientations:

$$r_{\text{quat}} = \exp\left(-2\|q_{\text{quat}} \ominus q_{\text{quat}}^{\text{ref}}\|^2\right), \quad (4)$$

where $\ominus$ denotes quaternion differences and q_{quat} and $q_{\text{quat}}^{\text{ref}}$ are the joint quaternions the simulated character and the mocap reference, respectively.

1.2. Locomotion Transfer Tasks

We consider three locomotion tasks from the DM Control locomotion task library (dm_control/locomotion) (Tassa et al., 2018; Merel et al., 2019a). In all tasks we initialize the humanoid using suitable motion capture pose.

Go-To-Target We consider a sparse go-to-target task. The simulated humanoid is randomly initialized in an arena. The task requires locomoting to a random target location. The agent is rewarded for being within 1 meter of the target location for 10 steps after which the target randomly moves to a new location. The task terminates after 25 seconds.

Gaps and Walls Obstacle Courses We consider two challenging obstacle courses to be solved from first-person visual observations. In both tasks the agent is rewarded for achieving a target root velocity of 3m/s in the direction of the corridor. We use the default reward function provided by the task – the reward linearly increases from 0 at 0m/s to 1 at 3m/s and linearly decreases to 0 at 6m/s. In both cases the task terminates after 45 seconds and we use corridor length of 100m and a corridor width of 10m.

For the gaps task, the length of the gaps is randomized between 0.75m and 1.25m and the length of the platforms between gaps is randomized between 0.3m and 2.5m.

For the walls task, the distance between walls is 5m. The walls are randomly placed on either side. The length of the walls is randomized between 1m and 7m. The height of the walls is randomized between 2.5m and 4m. We also randomize the wall colors.

1.3. Complementary Tasks

To demonstrate our joint training approach we use two tasks that require skills not well covered by the motion capture data.

Get up and Stand Task The first task we consider is a get up and stand task. The humanoid is initialized in a large variety of poses (about 5% lying on the ground and 95% floating slighly above the ground in a standing pose which induces falling in a variety of different ways). The reward function is $\exp(-(h - h_{\text{target}})^2)$, where h is the head height of the humanoid and h_{target} is the target height corresponding to standing. The task terminates after 8 seconds, implying a maximum reward of about 267. The perfect task performance the humanoid has to be able to get up from the ground when initialized on the ground and avoid falling for all other intializations.

Ball catching A second task involves catching a ball thrown towards the humanoid. This task is similar to the one considered by Merel et al. (2020). In this task the humanoid is initialized in a standing pose. The ball is initialized in mid-air with a random velocity components propelling it upwards and towards the humanoid. The size and mass of the ball, as well as its angular velocity are also randomly sampled across episodes. Episodes terminate after 6 seconds or if the ball makes contact with the ground or hits the humanoid's head. In the case of failure terminations the agent receives a reward of -1. Furthermore, the agent receives a reward of $+0.01$ for each hand touching the ball with the front and a penalty of -0.01 for touching the ball with the back. In addition, we use a small shaping reward to encourage standing up. The reward is identical to the one used in the get up and stand task but scaled to a maximum of 0.01 per timestep. Upon catching the ball, there is a small shaping reward incentivizing the humanoid to walk forwards towards a target.

2. Training Details and Network Architectures

We use V-MPO (Song et al., 2020), an on-policy variant of MPO in all of our experiments (Abdolmaleki et al., 2018). In the case of mixture networks we use an additional KL constraint on the distribution of mixture components similar to Wulfmeier et al. (2019). For all experiments we use a batchsize of 128 and an unroll length of 32. We use n-step returns to train the critics. In the V-MPO E-Step we use the top 50% of advantages as suggested by Song et al. (2020). All Lagrange multiplier in V-MPO were initialized by default to 1. We used Adam with a learning rate of 10^{-4}. We used a discount factor of 0.95 in all experiments.

We use a distributed infrastructure with a learner with access

	Hyperparameter	Value
V-MPO (E-Step)	ϵ	0.1
V-MPO (M-Step)	ϵ_μ	0.1
	ϵ_Σ	10^{-5}

Table 1. Hyperparameter choices for all MLP low-level tracking experiments.

to a 1x1 TPUv2 chip (with 2 cores) (Google, 2018) and use 4000 actor processes to produce environment interactions for the learner. A single experiment takes about 2-3 days.

2.1. Default Architectures

We tried to keep most architectural choices similar across our experiments. In this section we describe the default architectural choices which were used unless otherwise stated.

Reference Encoder For the reference encoder, we concatenate the reference observations for the next 5 timesteps, proprioceptive observations and the sampled latent z_{t-1} from the previous timestep. This combined observation is fed into an MLP with 2 hidden layers with 1024 hidden units each. We use layer norm but found that this makes little difference empirically. In the case of MLP low-level controller we also use an additional linear pathway from the input to the output of the MLP. We found that this change slightly improved results but made no difference for other low-level architectures. We use linear layers on top of the resulting representations to produce mean and log standard deviation for the stochastic latent embedding.

Value function For the value function we concatenate the reference observations for the next 5 timesteps, proprioceptive observations as well as a 30-dimensional learned clip embedding. This is fed into an MLP with 3 hidden layers with 1024 hidden units each. We use layer norm. We predict a value for each reward component separately on top of this representation.

2.2. KL regularization and Embedding Size Experiments

For this experiment, we use an MLP low-level policy with 2 hidden layers with 1024 hidden units each followed by linear layers to produce mean and log standard deviation. The inputs are a concatenation of the latent embedding z_t and the proprioceptive observations. Hyperparameters for this experiment are shown in table 1.

2.3. Architecture Comparison

In this experiment we compared a number of different low-level architectures across a range of data regimes. For all architectures, we optimized the V-MPO hyperparameters separately on the locomotion clip set (40min) based on downstream transfer performance and kept them the same across all other data regimes. We chose low-level architecture with a similar number of parameters as the MLP low-level.

MLP For the MLP low-level we used the same architecture as in the KL regularization and embedding size experiment with an embedding size of 60 and a KL regularization strength of 5×10^{-4}. The hyperparameters are the same as in table 1.

LSTM For the LSTM experiments we use two stacked LSTM with 384 and 256 units. In addition we use a recurrent value function instead of the feedforward value function described above. We used an embedding size of 60 and a KL regularization strength of 5×10^{-4}. The V-MPO hyperparameters are the same as in table 1.

Mixture and Product For the mixture architecture and the product architecture we use a shared torso with two hidden layers with 512 hidden units each followed by separate MLPs with 256 hidden units per primitive and 5 primitives.[1] The V-MPO hyperparameters for the mixture architecture and the product architecture can be found in tables 2 and 3, respectively. For mixture and product architectures, directly regularizing the space of mixture weights or exponents with a KL penalty does not seem sensible. Thus, for both mixture and product low-level architectures, we experimented with different ways to include regularization in the architecture. In addition to the default implementation without a latent bottleneck or KL regularization we also experimented with a latent bottleneck of dimensionality 60. In this case the latent produced by the reference encoder is concatenated with the proprioceptive information and fed through a two-layer MLP with 400 hidden units each to produce the mixture weights or exponents, respectively. For each data regime we compare the default implementation without KL regularization and version with a latent bottleneck and different regularization strengths. We also tried explicitly initializing the different components far away from each other to encourage use of different primitives by using a fixed bias in the mean of each component. Specifically, we explored biases of -0.66, -0.33, ..., 0.66 for the 5 primitives. We report the best performing architectures in table 4. We note that we found training a product architecture without additional regularization very unstable.

2.4. Joint Training Experiments

In our joint training experiments we used an MLP low-level architecture and per task high level controllers and value

[1]We note that in preliminary experiments we also explored using more primitives without substantially different results.

	Hyperparameter	Value
V-MPO (E-Step)	ϵ	0.1
V-MPO (M-Step)	ϵ_μ	0.1
	ϵ_Σ	10^{-3}
	$\epsilon_{\text{discrete}}$	10^{-3}

Table 2. V-MPO hyperparameter choices for all mixture low-level experiments.

	Hyperparameter	Value
V-MPO (E-Step)	ϵ	0.1
V-MPO (M-Step)	ϵ_μ	0.1
	ϵ_Σ	10^{-3}

Table 3. V-MPO hyperparameter choices for all product low-level experiments.

functions. For the mocap tracking task we used the same default architecture as above. We used the same architectures for high-level controllers and value functions in both the tracking task and the complementary tasks. We did not learn separate value function per reward term in these experiments. In the experiments involving the get up and stand task we sampled the get up and stand task in 10% of all episodes. In the experiments involving the ball catching task we sampled the ball catching task in 20% of all episodes.

2.5. Locomotion and Joint Training Transfer Experiments

We used the same high-level controller architecture, algorithm and hyperparameters for all transfer experiments on a given task. We used SVG(0) (Heess et al., 2015) with the Retrace off-policy correction (Munos et al., 2016) in all of our experiments. In all of our experiments we used shared observation encoders. These are small MLPs for the proprioceptive observations and the non-visual task observations in the go-to-target, get up and ball catching tasks. For the image observations in the walls and gaps tasks we use a small ResNet. The observation encodings are concatenated and processed by an LSTM with 128 units that is shared between policy and action-value function. On top of this LSTM we have a one-layer MLP with 256 hidden units for the action-value function function and another LSTM with 128 units followed by linear layers to produce the high-level policy.

Data regime	Best mixture architecture
Walking (2min)	No bottleneck, bias init
Running (2min)	No bottleneck, bias init
Locomotion (40min)	No bottleneck, bias init
Large (3.5h)	Bottleneck $\beta = 5 \times 10^{-4}$, bias init
	Best product architecture
Walking (2min)	Bottleneck $\beta = 5 \times 10^{-4}$
Running (2min)	Bottleneck $\beta = 5 \times 10^{-4}$
Locomotion (40min)	Bottleneck $\beta = 5 \times 10^{-4}$
Large (3.5h)	Bottleneck $\beta = 1 \times 10^{-4}$

Table 4. Best performing modular architectures for each data regime.

References

Abdolmaleki, A., Springenberg, J. T., Tassa, Y., Munos, R., Heess, N., and Riedmiller, M. Maximum a posteriori policy optimisation. In *International Conference on Learning Representations (ICLR)*, 2018.

Chentanez, N., Müller, M., Macklin, M., Makoviychuk, V., and Jeschke, S. Physics-based motion capture imitation with deep reinforcement learning. In *International Conference on Motion, Interaction, and Games (MIG)*. ACM, 2018.

Google. Cloud TPU. 2018. URL https://cloud.google.com/tpu/.

Heess, N., Wayne, G., Silver, D., Lillicrap, T., Erez, T., and Tassa, Y. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, pp. 2944–2952, 2015.

Merel, J., Ahuja, A., Pham, V., Tunyasuvunakool, S., Liu, S., Tirumala, D., Heess, N., and Wayne, G. Hierarchical visuomotor control of humanoids. In *International Conference on Learning Representations (ICLR)*, 2019a.

Merel, J., Hasenclever, L., Galashov, A., Ahuja, A., Pham, V., Wayne, G., Teh, Y. W., and Heess, N. Neural probabilistic motor primitives for humanoid control. In *International Conference on Learning Representations (ICLR)*, 2019b.

Merel, J., Tunyasuvunakool, S., Ahuja, A., Tassa, Y., Hasenclever, L., Pham, V., Erez, T., Wayne, G., and Heess, N. Catch & carry: Reusable neural controllers for vision-guided whole-body tasks. In *SIGGRAPH 2020*, 2020.

Munos, R., Stepleton, T., Harutyunyan, A., and Bellemare, M. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 1054–1062, 2016.

Peng, X. B., Abbeel, P., Levine, S., and van de Panne, M. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *Transactions on Graphics (TOG)*, 2018.

Peng, X. B., Chang, M., Zhang, G., Abbeel, P., and Levine, S. Mcp: Learning composable hierarchical control with multiplicative compositional policies. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Song, H. F., Abdolmaleki, A., Springenberg, J. T., Clark, A., Soyer, H., Rae, J. W., Noury, S., Ahuja, A., Liu, S., Tirumala, D., Heess, N., Belov, D., Riedmiller, M., and Botvinick, M. M. V-MPO: On-policy maximum a posteriori policy optimization for discrete and continuous control. In *International Conference on Learning Representations (ICLR)*, 2020.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., Casas, D. d. L., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., Lillicrap, T., and Riedmiller, M. DeepMind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2012.

Wulfmeier, M., Abdolmaleki, A., Hafner, R., Springenberg, J. T., Neunert, M., Hertweck, T., Lampe, T., Siegel, N., Heess, N., and Riedmiller, M. A. Regularized hierarchical policies for compositional transfer in robotics. *arXiv preprint arXiv:1906.11228*, 2019.