
CompILE: Compositional Imitation Learning and Execution

Thomas Kipf^{1†} Yujia Li² Hanjun Dai^{3†} Vinicius Zambaldi² Alvaro Sanchez-Gonzalez²
Edward Grefenstette^{4#} Pushmeet Kohli² Peter Battaglia²

Abstract

We introduce Compositional Imitation Learning and Execution (CompILE): a framework for learning reusable, variable-length segments of hierarchically-structured behavior from demonstration data. CompILE uses a novel unsupervised, fully-differentiable sequence segmentation module to learn latent encodings of sequential data that can be re-composed and executed to perform new tasks. Once trained, our model generalizes to sequences of longer length and from environment instances not seen during training. We evaluate CompILE in a challenging 2D multi-task environment and a continuous control task, and show that it can find correct task boundaries and event encodings in an unsupervised manner. Latent codes and associated behavior policies discovered by CompILE can be used by a hierarchical agent, where the high-level policy selects actions in the latent code space, and the low-level, task-specific policies are simply the learned decoders. We found that our CompILE-based agent could learn given only sparse rewards, where agents without task-specific policies struggle.

1. Introduction

Discovering compositional structure in sequential data, without supervision, is an important ability in human and machine learning. For example, when a cook prepares a meal, they re-use similar behavioral sub-sequences (e.g., slicing, dicing, chopping) and compose the components hierarchically (e.g., stirring together eggs and milk, pouring the mix-

[†]Work done during an internship at DeepMind. [#]Work done while employed at DeepMind. ¹Informatics Institute, University of Amsterdam, Amsterdam, The Netherlands ²DeepMind, London, UK ³School of Computational Science and Engineering, Georgia Institute of Technology, Atlanta, Georgia, USA ⁴Facebook AI Research, London, UK. Correspondence to: Thomas Kipf <t.n.kipf@uva.nl>.

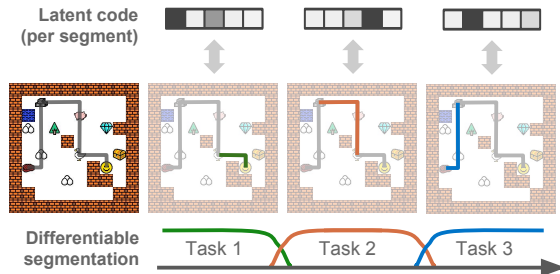


Figure 1: Joint unsupervised learning of task segmentation and encoding in CompILE. CompILE auto-encodes sequential demonstration data by 1) softly breaking an input sequence into segments of variable length, and 2) mapping each such segment into a latent code, which can be executed to reconstruct the input sequence. At test time, the latent code can be re-composed to produce novel behavior.

ture into a hot pan and stirring it to form scrambled eggs). Humans are adept at inferring event structure by hierarchically segmenting continuous sensory experience (Zacks et al., 2001; Baldassano et al., 2017; Radvansky & Zacks, 2017), which may support building efficient event representations in episodic memory (Ezzyat & Davachi, 2011) and constructing abstract plans (Richmond & Zacks, 2017).

An important benefit of compositional sub-sequence representations is combinatorial generalization to never-before-seen conjunctions (Davidson, 1984; Denil et al., 2017). Behavioral sub-components can also be used as high-level actions in hierarchical decision-making, offering improved credit assignment and efficient planning. To reap these benefits in machines, however, the event structure and composable representations must be discovered in an unsupervised manner, as sub-sequence labels are rarely available.

In this work, we focus on the problem of jointly learning to segment, explain, and imitate agent behavior (from demonstrations) via an unsupervised auto-encoding objective. The encoder learns to jointly infer event boundaries and high-level abstractions (latent encodings) of activity within each event segment, while the task of the decoder is to reconstruct or imitate the original behavior by executing the inferred sequence of latent codes.

We introduce a *fully differentiable*, unsupervised segmentation model that we term CompILE (Compositional Imitation Learning and Execution) that addresses the segmentation problem by predicting soft *segment masks*. During training, the model makes multiple passes over the input sequence, explaining one segment of activity at a time. Segments explained by earlier passes are softly masked out and thereby ignored by the model. Our approach to masking is related to soft self-attention (Parikh et al., 2016; Vaswani et al., 2017), where each mask predicted by our model is localized in time (see Figure 1 for an example). At test time, these soft masks can be replaced with discrete, consecutive masks that mark the beginning and end of a segment. This allows us to process sequences of arbitrary length by 1) identifying the next segment, 2) explaining this segment with a latent variable, and 3) cutting/removing this segment from the sequence and continue the process on the remainder of the input.

Formally, our model takes the form of a conditional variational auto-encoder (VAE) (Kingma & Welling, 2014; Rezende et al., 2014; Sohn et al., 2015). We introduce a method for modeling segment boundaries as softly relaxed discrete latent variables (Jang et al., 2017; Maddison et al., 2017) which allows for efficient, low-variance training.

We demonstrate the efficacy of our approach in a multi-task, multiple instruction-following domain similar to Oh et al. (2017) and a continuous control environment. Our model can reliably discover event boundaries and find effective event (sub-task) encodings. In a number of experiments, we found that CompILE generalizes to unseen environment configurations and to task sequences which were longer than those seen during training.

Once trained, the latent codes and associated behavior discovered by CompILE can be reused and recomposed to solve new, unseen tasks. We demonstrate this ability in a set of experiments using a hierarchical agent, with a *meta controller* that learns to operate over discovered policies and associated latent codes to solve difficult sparse reward tasks, where non-hierarchical, non-compositional baselines struggle to learn.

2. Model overview

We consider the task of auto-encoding sequential data by 1) breaking an input sequence into disjoint segments of variable length, and 2) mapping each segment individually into some higher-level code, from which the input sequence can be reconstructed.

More specifically, we focus on modeling state-action trajectories of the form $\rho = ((s_1, a_1), (s_2, a_2), \dots, (s_T, a_T))$ with states $s_t \in \mathcal{S}$ and actions $a_t \in \mathcal{A}$ for time steps $t = 1, \dots, T$, e.g. obtained from a dataset $\mathcal{D} = \{\rho_1, \rho_2, \dots, \rho_N\}$ of N expert demonstrations of variable length for a set of tasks.

2.1. Behavioral cloning

Our basic setup follows that of behavioral cloning (BC), i.e., we want to find an imitation policy π_θ , parameterized by θ , by solving the following optimization problem:

$$\theta^* = \arg \max_{\theta} \mathbb{E}_{\rho \in \mathcal{D}} [p_\theta(a_{1:T} | s_{1:T})]. \quad (1)$$

In BC we have $p_\theta(a_{1:T} | s_{1:T}) = \prod_{t=1:T} \pi_\theta(a_t | s_t)$, where $\pi_\theta(a | s)$ denotes the probability of taking action a in state s under the imitation policy π_θ .

2.2. Sub-task identification and imitation

Differently from the default BC setup, our model breaks trajectories ρ into M disjoint segments (c_1, c_2, \dots, c_M) :

$$c_i = ((s_{b_{i'}}, a_{b_{i'}}), (s_{b_{i'}+1}, a_{b_{i'}+1}), \dots, (s_{b_i-1}, a_{b_i-1})), \quad (2)$$

where M is a hyperparameter, and $i' = i - 1$. Here, $b_i \in [1, T + 1]$ are discrete (latent) boundary indicator variables with $b_0 = 1$, $b_M = T + 1$, and $b_i \geq b_{i'}$. We allow segments c_i to be empty if $b_i = b_{i'}$. We model each part independently with a sub-task policy $\pi_\theta(a | s, z)$, where z is a latent variable summarizing the segment. Framing BC as a joint segmentation and auto-encoding problem allows us to obtain imitation policies that are specific to different inferred sub-tasks, and which can be re-combined for easier generalization to new settings. Each sub-task policy is responsible for explaining a variable-length segment of the demonstration trajectory.

We take the segment (sub-task) encoding z to be discrete in the following, but we note that other choices are possible and require only minor modifications to our framework. The probability of an action sequence $a_{1:T}$ given a sequence of states $s_{1:T}$ then takes the following form*:

$$\begin{aligned} p_\theta(a_{1:T} | s_{1:T}) &= \quad (3) \\ &\sum_{b_{1:M}} \sum_{z_{1:M}} p_\theta(a_{1:T} | s_{1:T}, b_{1:M}, z_{1:M}) p(b_{1:M}, z_{1:M}) = \\ &\sum_{b_{1:M}} \prod_{z_{1:M}} p_\theta(a_{b_{i'}:b_i-1} | s_{b_{i'}:b_i-1}, z_i) p(b_i | b_{i'}) p(z_i) = \\ &\sum_{b_{1:M}} \prod_{z_{1:M}} \left[\prod_{j=b_{i'}:b_i-1} \pi_\theta(a_j | s_j, z_i) \right] p(b_i | b_{i'}) p(z_i), \end{aligned}$$

where the double summation marginalizes over all allowed configurations of the discrete latent variables $z_{1:M}$ and $b_{1:M}$. We omit $p(b_0)$ since we set $b_0 = 1$. Note that our framework supports both discrete and continuous latent variables $z_{1:M}$ —for the latter case, the summation sign in Eq.(3) is replaced with an integral. Our (conditional) generative model $p_\theta(a_{1:T} | s_{1:T}, b_{1:M}, z_{1:M})$ factorizes across time steps if we

*We again use the shorthand notation $i' = i - 1$ for clarity.

choose a non-recurrent policy $\pi_\theta(a|s, z)$. Using recurrent policies is necessary, e.g., for partially observable environments and is left for future work.

For simplicity, we assume independent priors over b and z as follows: $p(b_i, z_i | b_{1:i'}, z_{1:i'}) := p(b_i | b_{i'})p(z_i)$. If more complex dependencies are present in the data, this assumption can be replaced with some mechanism for implementing conditional probabilities between segments. We choose a uniform categorical prior $p(z_i)$ and the following empirical categorical prior for the boundary latent variables:

$$p(b_i | b_{i'}) \propto \text{Poisson}(b_i - b_{i'}, \lambda) = e^{-\lambda} \frac{\lambda^{b_i - b_{i'}}}{(b_i - b_{i'})!}, \quad (4)$$

proportional to a Poisson distribution with rate λ , but truncated to the interval $[b_{i'}, T + 1]$ and renormalized, as we are dealing with sequences of finite length. This prior encourages segments to be close to λ in length and helps avoid two failure modes: 1) collapse of segments to unit length, and 2) a single segment covering the full sequence length.

2.2.1. RECOGNITION MODEL

Following the standard VAE (Kingma & Welling, 2014; Rezende et al., 2014) framework, we introduce a recognition model $q_\phi(b_{1:M}, z_{1:M} | a_{1:T}, s_{1:T})$ that allows us to infer a task decomposition via boundary variables $b_{1:M}$ and task encodings $z_{1:M}$ for a given trajectory ρ . We would like our recognition model to be able to generalize to new compositions of the underlying latent code. We can encourage this by dropping the dependence of q_ϕ on any time steps before the previous boundary position. In practice, this means that once a segment (sub-task) has been identified and explained by a latent variable z , the corresponding part of the input trajectory will be masked out and the recognition model proceeds on the remainder of the trajectory, until the end is reached. This will further facilitate generalization to sequences of longer length (and with more segments) than those seen during training.

Formally, we structure the recognition model as follows:

$$q_\phi(b_{1:M}, z_{1:M} | x_{1:T}) = \prod_{i=1:M} q_{\phi_z}(z_i | x_{b_{i'}:b_i-1}) q_{\phi_b}(b_i | x_{b_{i'}:T}), \quad (5)$$

where we have used $x_t = (a_t, s_t)$ and $i' = i - 1$ to simplify notation. Expressed in other words, we re-use the same recognition model with shared parameters for each segment while masking out already explained segments. The core modules are the *encoding network* $q_{\phi_z}(z|x)$ and the *boundary prediction network* $q_{\phi_b}(b|x)$, both are modeled as categorical distributions. We use recurrent neural networks (RNN)—specifically, a uni-directional LSTM (Hochreiter & Schmidhuber, 1997)—with shared parameters, but with

different output heads: one head for predicting the logits h_{b_i} for the boundary latent variable b_i at every time step, and one head for predicting the logits h_{z_i} for the sub-task encoding z_i at the last time step in the current segment C_i .

We use multi-layer perceptrons (MLPs) to implement the output heads:

$$h_{z_i} = \text{MLP}_z(\text{LSTM}_{b_{i-1}}(\tilde{x}_{b_{i'}:b_{i-1}})), \quad (6)$$

$$h_{b_i}^t = \text{MLP}_b(\text{LSTM}_t(\tilde{x}_{b_{i'}:T})), \quad (7)$$

where the MLPs have parameters specific to b or z (i.e., not shared between the output heads). The subscript t on LSTM_t denotes the time step at which the output is read. Note that h_{z_i} is a K -dimensional vector where K is the number of latent categories, whereas $h_{b_i}^t$ is a scalar specific to time step t . \tilde{x}_t denotes a learned embedding of the input x_t at time step t . In practice, we implement this embedding using a convolutional neural network (CNN), i.e., $\tilde{x}_t = \text{CNN}(x_t)$, with layer normalization (Ba et al., 2016) for pixel-based inputs and using an MLP otherwise. Note that the CNN is only applied to the state, but not on the action component of x_t .

2.2.2. CONTINUOUS RELAXATION

We can jointly train the recognition and the generative model by using the usual ELBO as an objective for learning (see supplementary material). To obtain low-variance gradient estimates for learning, we can use the reparameterization trick for VAEs (Kingma & Welling, 2014). Our current model formulation, however, does not allow for reparameterization as both b and z are discrete latent variables. To circumvent this issue, we make use of a continuous relaxation, i.e., we replace the respective categorical distributions with Gumbel softmax / concrete (Maddison et al., 2017; Jang et al., 2017) distributions. While this is straightforward for the sub-task latent variables z , some extra consideration is required to translate the constraint $b_i \geq b_{i'}$ and the conditioning on trajectory segments of the form $x_{b_{i'}:b_i-1}$ to the continuous case. Note that we again summarize pairs of states s_t and actions a_t in a single variable $x_t = (a_t, s_t)$ for ease of notation. The continuous relaxation is only necessary at training time, during testing we can fall back to the discrete version explained in the previous section.

Soft segment masks In the relaxed/continuous case at training time we cannot enforce a strict ordering $b_i \geq b_{i'}$ on the boundaries directly as we are now dealing with “soft” distributions and don’t have access to discrete samples at training time. It is still possible, however, to evaluate segment probabilities of the form $P(t \in C_i)$, i.e., the probability that a certain time step t in the trajectory ρ belongs to the i -th segment $C_i = [\max_{0 \leq j \leq i-1} b_j, b_i)$. The lower boundary of the segment is now given by the maximum

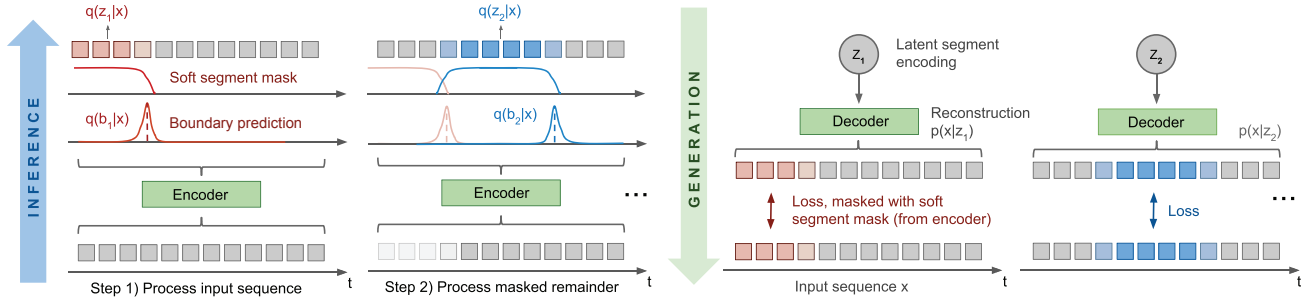


Figure 2: Differentiable segmentation of an input trajectory x composed of a sequence of sub-tasks. The recognition model (encoder, marked as *inference*) predicts relaxed categorical (Gumbel softmax) boundary distributions $q(b_i|x)$ from which we can obtain soft segment masks $P(t \in C_i)$. Each segment C_i is encoded via $q(z_i|x)$. The generative model $p(x|z_i)$ is executed once for every latent variable z_i . The reconstruction loss is masked with $P(t \in C_i)$, so that only the reconstructed part corresponding to the i -th segment receives a training signal. For imitation learning, the generative model (decoder, marked as *generation*) takes the form of a policy $\pi_\theta(a_t|s_t, z_i)$.

value of all previous boundary variables, as the ordering $b_i \geq b_{i'}$ is no longer guaranteed to hold. C_i is assumed to be empty if any $b_j \geq b_i$ with $j < i$. We can evaluate segment probabilities as follows:

$$P(t \in C_i) = P\left(\max_{0 \leq j \leq i-1} b_j \leq t < b_i\right) = \quad (8)$$

$$[1 - \text{cumsum}(q_{\phi_b}(b_i|x), t)] \prod_{j=0:i-1} \text{cumsum}(q_{\phi_b}(b_j|x), t),$$

where $\text{cumsum}(q_{\phi_b}(b_j|x), t) = \sum_{k \leq t} q_{\phi_b}(b_j = k|x)$ is a shorthand for the *inclusive* cumulative sum of the posterior $q_{\phi_b}(b_j|x)$, evaluated at time step t , i.e., it is equivalent to the CDF of $q_{\phi_b}(b_j|x)$. We further have $\text{cumsum}(q_{\phi_b}(b_0|x), t) = 1$ and $\text{cumsum}(q_{\phi_b}(b_M|x), t) = 0$. It is easy to verify that $\sum_{i=1:M} P(t \in C_i) = 1$ for all t . These segment probabilities can be seen as soft segment masks. See Figure 2 for an example.

RNN state masking We softly mask out parts of the input sequence explained by earlier segments. Using a soft masking mechanism allows us to find suitable segment boundaries via backpropagation, without the need to perform explicit and potentially expensive/intractable marginalization over latent variables. Specifically, we mask out the *hidden states*[†] of the encoding and boundary prediction networks’ RNNs. Thus, inputs belonging to earlier segments are effectively hidden from the model while still allowing gradients to be passed through. The hidden state mask for the i -th segment takes the following form:

$$\text{mask}_i(t) = P\left(t \geq \max_{0 \leq j \leq i-1} b_j\right) = \quad (9)$$

$$\prod_{j=0:i-1} P(t \geq b_j) = \prod_{j=0:i-1} \text{cumsum}(q_{\phi_b}(b_j|x), t),$$

[†]Including the cell state in the LSTM architecture.

where we set $\text{mask}_1 = 1$. In other words, it is given by the probability for a given time step to *not* belong to a previous segment. Masking is performed by multiplying the RNN’s hidden state with mask_i (after the RNN update of the current time step). For every segment $i \in [1, M]$ we thus need to run the RNN over the full input sequence, while multiplying the hidden states with a segment-specific mask. Nonetheless, the parameters of the RNN are shared over all segments.

Soft RNN readout In addition to softly masking the RNN hidden states in both $q_{\phi_b}(b_i|x)$ and $q_{\phi_z}(z_i|x)$, we mask out illegal boundary positions by setting the respective logits to a large negative value. Specifically, we mask out the first time step (as any boundary placed on the first time step would result in an empty segment) and any time steps corresponding to padding values when training on mini-batches of sequences with different length. We allow boundaries (as they are exclusive) to be placed at time step $T + 1$. Further, to obtain $q_{\phi_z}(z_i|x)$ from the z -specific output head h_z^t —where t denotes the time step at which we are reading from the RNN—we perform the following weighted average:

$$q_{\phi_z}(z_i|x) = \text{concrete}_\tau \left(\sum_{t=1:T} q_{\phi_b}(b_i = t+1|x) h_z^t \right), \quad (10)$$

which can be understood as the “soft” equivalent of reading the output head h_z^t for the last time step within the corresponding segment. concrete_τ is a concrete / Gumbel softmax distribution (Jang et al., 2017; Maddison et al., 2017) with temperature τ . Note the necessary shift of the boundary distribution by 1 time step, as $q_{\phi_b}(b_i|x)$ points to the first time step of the *following* segment.

Loss masking The reconstruction loss part of the ELBO $\mathcal{L} = -\mathbb{E}_{q_{\phi}(b,z|a,s)}[\log p_{\theta}(a|s,b,z)]$ decomposes into independent loss terms for each segment, i.e., $\mathcal{L} = \sum_{i=1:M} \mathcal{L}_i$, due to the structure of our generative model, Eq. (3). To retain this property in the relaxed/continuous case, we softly mask out irrelevant parts of the action trajectory when evaluating the loss term for a single segment:

$$\mathcal{L}_i = \mathbb{E}_{q_{\phi}(b,z|a,s)}[\text{seg}_i \cdot \log p_{\theta}(a|s, z_i)], \quad (11)$$

where the segment mask for time step t is given by $\text{seg}_i(t) = P(t \in C_i)$, i.e. the probability of time step t being explained by the i -th segment. The operator “ \cdot ” denotes element-wise multiplication. In practice, we use a single sample of the (reparameterized) posterior to evaluate Eq. (11).

Number of segments At training time, we need to specify the maximum number of segments M that the model is allowed to use when auto-encoding a particular sequence of length T . For efficient mini-batch training, we choose a single, fixed M for all training examples. Providing the correct number of segments can further be utilized as a form of weak supervision.

Complexity Evaluating the model components $q_{\phi_b}(b_i|x)$, $q_{\phi_z}(z_i|x)$, and $p_{\theta}(x|z_i)$ is $\mathcal{O}(T)$ for a single $i = 1, \dots, M$. The overall forward pass of the CompILE model for a single demonstration trajectory in terms of its length T and the number of segments M is therefore $\mathcal{O}(TM)$.

3. Related work

Our framework is closely related to option discovery (Niekum et al., 2013; Kroemer et al., 2015; Fox et al., 2017; Hausman et al., 2017; Krishnan et al., 2017; Fox et al., 2018), with the main difference being that our inference algorithm is agnostic to what type of option (sub-task) encoding is used. Our framework allows for inference of continuous, discrete or mixed continuous-discrete latent variables. Fox et al. (2017) introduce an EM-based inference algorithm for option discovery in settings similar to ours, however limited to discrete latent variables and to inference networks that are independent of the position of task boundaries: in their case without recurrency and only dependent on the current state/action pair. Their framework was later applied to continuous control tasks (Krishnan et al., 2017) and neural program modeling (Fox et al., 2018).

Option discovery has also been addressed in the context of inverse reinforcement learning (IRL) using generative adversarial networks (GANs) (Goodfellow et al., 2014) to find structured policies that are close to demonstration sequences (Hausman et al., 2017; Sharma et al., 2018). This approach requires being able to interact with the environment for imitation learning, whereas our model is based on BC and works on offline demonstration data.

Various solutions for supervised sequence segmentation or task decomposition exist which require varying degrees of supervision (Graves, 2012; Escorcia et al., 2016; Krishna et al., 2017; Shiarlis et al., 2018). In terms of two recent examples, Krishna et al. (2017) assume fully-annotated event boundaries and event descriptions at training time whereas TACO (Shiarlis et al., 2018) only requires *task sketches* (i.e., supervision on sub-task encodings but not on task boundaries) and solves an alignment problem to find a suitable segmentation. A related recent approach decomposes demonstration sequences into underlying programs (Sun et al., 2018) in a fully-supervised setting, based on a seq2seq (Sutskever et al., 2014; Vinyals et al., 2015) model without explicitly modeling segmentation.

Outside of the area of *learning from demonstration*, hierarchical reinforcement learning (Sutton et al., 1999; Kulkarni et al., 2016; Bacon et al., 2017; Florensa et al., 2017; Vezhnevets et al., 2017; Riemer et al., 2018) and the options framework (Sutton et al., 1999; Kulkarni et al., 2016; Bacon et al., 2017; Riemer et al., 2018) similarly deal with learning segmentations and representations of behavior, but in a purely generative way. Learning with task sketches (Andreas et al., 2017) and learning of transition policies (Lee et al., 2019) has also been addressed in this context.

Unsupervised segmentation and encoding of sequential data has also received considerable attention in natural language and speech processing (Blei & Moreno, 2001; Goldwater et al., 2009; Chan et al., 2017; Wang et al., 2017; Tang et al., 2018), and in the analysis of sequential activity data (Johnson et al., 2016; Dai et al., 2017). In concurrent work, Pertsch et al. (2019) introduced a differentiable model for keyframe discovery in sequence data, which is related to our setting. Sequence prediction models with adaptive step size (Neitz et al., 2018; Jayaraman et al., 2018) can provide segment boundaries as well, but do not directly learn a policy or latent encodings.

4. Experiments

The goals of this experimental section are as follows: 1) we would like to investigate whether our model is effective at both learning to find task boundaries and task encodings while being able to reconstruct and imitate unseen behavior, 2) test whether our modular approach to task decomposition allows our model to generalize to longer sequences with more sub-tasks at test time, and 3) investigate whether an agent can learn to control the discovered sub-task policies to quickly learn new tasks in sparse reward settings.

4.1. Multi-task environments

We evaluate our model in a fully-observable 2D multi-task grid world, similar to the one introduced in Oh et al. (2017)

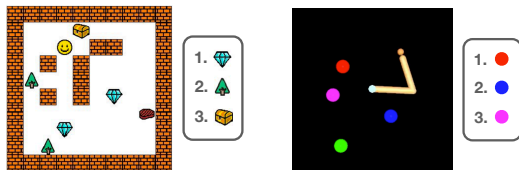


Figure 3: Example instances of multi-task, instruction-following environments used in our experiments. *Left*: Grid world with walls. An agent has to pick up or visit certain objects. *Right*: Continuous control reacher task with multiple targets. The tip of the reacher arm has to touch multiple colored spheres in a pre-specified order.

and a continuous control task, where a reacher arm has to reach certain target locations. An example instance for each environment is shown in Figure 3. See supplementary material for additional implementation and evaluation details.

Grid world The environment is a 10x10 grid world with a single agent, impassable walls, and multiple objects scattered throughout the scene. We generate scenes with 6 objects selected uniformly at random from 10 different object types (excl. walls and player) jointly with task lists of 3-5 *visit* and *pick up* tasks. A single visit task can be solved by moving the agent to the location of an object of the correct type. For example, if the instruction is *visit tree*, the task is completed if any tree in the scene is visited. Similarly, a pick up task can be solved by picking up an object of the correct type (moving to a field adjacent to the object and executing a directional pick up action, e.g. *pick up north*). We generate a demonstration trajectory for each environment instance and task list by running a shortest path algorithm on the 2D environment grid (while marking walls as impassable).

Continuous control In this environment, a two-link planar *reacher* arm has to be controlled to reach towards pre-specified target locations. The environment is an adaptation of the single-target reacher task from the DeepMind Control Suite (Tassa et al., 2018). We simultaneously place up to 6 targets drawn without replacement from 10 different target types (spheres of different color) in a single environment instance, distributed uniformly at random within reach of the reacher arm. The number of targets in an environment is drawn uniformly in range [number of tasks, 6]. For each such instance, we generate a task list by selecting 3-5 of the target object types in the environment. The current target is marked as reached and removed from the scene if the end effector—a small sphere at the tip of the reacher arm—touches the target sphere. The observations to the agent are the positions of the all targets, and the position of the reacher arm. We generate demonstration trajectories using a hand-coded control policy, which opens or closes the arm based on the distance of the target to the center, and rotates the shoulder based on the direction to the target.

4.2. Imitation learning

In this set of experiments, we fit our CompILE model to demonstration trajectories generated for random instances of the multi-task environments (incl. randomly generated task lists). We train our model with discrete latent variables (as the target types are discrete) on demonstration trajectories with three consecutive tasks, either 3x visit instructions or 3x pick up instructions in the grid world, and 3x reaching instructions in the continuous control environment. Training is carried out on a single GPU with a fixed learning rate of 10^{-4} using the Adam (Kingma & Ba, 2015) optimizer, with a batch size of 256 and for a total of 50k training iterations (500k for reacher task). We further train a causal termination policy that shares the same architecture as the encoder of CompILE to mimic the boundary prediction module in an online setting, i.e., without *seeing the future*.

We evaluate our model on 1024 newly generated instances of the environment. We again generate demonstration trajectories with random task lists of either 3 consecutive tasks (same number as during training) or 5 consecutive tasks, to test for generalization to longer sequences, and we evaluate both boundary prediction performance and accuracy of action sequence reconstruction from the inferred latent code. We provide weak supervision by setting the number of segments to $M = 3$ and $M = 5$, respectively. We find that results slightly degrade with non-optimal choice of M (see additional experiments in the supplementary material).

Baselines We compare against two baselines that are based on behavioral cloning (BC): an autoregressive baseline for evaluating segmentation performance, termed *LSTM surprisal*, where we find segment boundaries by thresholding the state-conditional likelihood of an action. In the grid world domain, we further compare against a VAE-based *BC baseline* that corresponds to a variant of our model without inferred task boundaries, i.e. with only a single segment. This baseline allows us to evaluate task reconstruction performance from an expert trajectory that is encoded in a *single* latent variable. We choose a 32-dim. Gaussian latent variable z (i.e., with significantly higher capacity) and a unit-variance, zero-mean Gaussian prior for this baseline. We further show results for two model variants: z - and b -CompILE, where we provide supervision on the latent variables z or b during training. z -CompILE is comparable to TACO (Shiarlis et al., 2018), where task sketches (z in our case) are provided both during training and testing (we only provide z during training), whereas b -CompILE is related to imitation learning of annotated, individual tasks.

Grid world results Results for the grid world tasks are summarized in Figure 4. For the pick up task, we see that our model reliably finds the correct boundary positions, i.e., it discovers the correct segments of behavior both in the 3-task



Figure 4: Imitation learning results in grid world domain. We report accuracy of segmentation boundary recovery, reconstruction accuracy (average over sequence vs. percentage of exact full-sequence matches) and *online evaluation*: average reward obtained when deploying the generative model (with termination policy) using the inferred latent code from the demonstration sequence in the environment, without re-training. See main text for additional details.

setting (same as training) and in the longer 5-task setting. Reconstructions from the latent code sequence are almost perfect and only degrade slightly in the generalization setting to longer sequences, whereas the BC baseline without segmentation mechanism completely fails to generalize to longer sequences (see *exact match* score). In the visit task setting, ground truth boundary positions can be ambiguous (the agent can walk over an object unintentionally on its way somewhere else) which is reflected in the sometimes lower online evaluation score, as the termination policy can be sensitive to ambiguous termination conditions (e.g., unintentionally walked-over objects). Nonetheless, CompILE is often able to generalize to longer sequences whereas the baseline model without task segmentation consistently fails. In both tasks, our model beats a surprisal-driven segmentation baseline by a large margin.

Continuous control results Results for unsupervised segmentation boundary recovery for the reacher task are summarized in Table 1. We find that CompILE can (almost) perfectly recover segmentation boundaries when trained with partial supervision on z (z-CompILE), matching the performance of b-CompILE that receives supervision on boundary position. Note that different from TACO (Shiarlis et al., 2018), no supervision is provided at test time. The fully unsupervised model (CompILE) outperforms an autoregressive baseline (LSTM surprisal) by a large margin, but often does not recover the exact segmentation that generated the trajectory. The F1 score with tolerance for misplaced boundaries by 1 time step (tol=1) shows that in some cases the error can be explained by a minor prediction offset. We omit reconstruction performance results in the continuous domain, as a fair evaluation would require addressing the *covariate shift* problem in BC to allow the policy to recover from small errors, e.g., using a technique such as DART (Laskey et al., 2017) to inject noise in the training process. We leave this for future work.

Model	Accuracy	F1 (tol=0)	F1 (tol=1)
3 tasks			
LSTM surprisal	24.8 ± 0.6	39.0 ± 0.3	47.1 ± 0.4
CompILE	62.0 ± 4.5	74.3 ± 3.3	78.9 ± 2.5
z-CompILE	99.5 ± 0.2	99.7 ± 0.2	99.8 ± 0.1
b-CompILE	99.8 ± 0.1	99.9 ± 0.1	100 ± 0.0
5 tasks – generalization			
LSTM surprisal	21.6 ± 0.5	44.9 ± 0.5	54.4 ± 0.5
CompILE	41.7 ± 8.0	69.3 ± 4.7	74.0 ± 4.6
z-CompILE	98.4 ± 0.5	99.3 ± 0.2	99.8 ± 0.1
b-CompILE	98.8 ± 0.3	99.5 ± 0.1	99.8 ± 0.1

Table 1: Segmentation results in continuous control domain. We report accuracy (mean and standard deviation over 5 runs) of exact segmentation boundary recovery and two F1 scores (in %), which measure the harmonic mean between precision and recall for boundary prediction, with (tol=1) and without (tol=0) tolerance for boundaries that are misplaced by 1 time step in either direction.

4.3. Hierarchical reinforcement learning

In this set of experiments, we pre-train a CompILE model under the same setting as in Section 4.2 in the grid world environment and only keep the discovered sub-task policies and the termination policy. We provide these policies to a hierarchical agent that can either call a low-level action (such as move or pick up) directly in the environment, or call a *meta action*, that executes a particular sub-task policy incl. termination policy, until a termination criterion is met (termination probability larger than 0.5 or end of episode).

We generate tasks and environments at random as in the imitation learning setting, but deploy agents in the environment where they either receive a reward of 1 for every completed sub-task (*dense reward* setting) or a single reward of 1 at the end of the episode if all tasks are completed and no termination criterion (e.g., wrong object was picked up, or

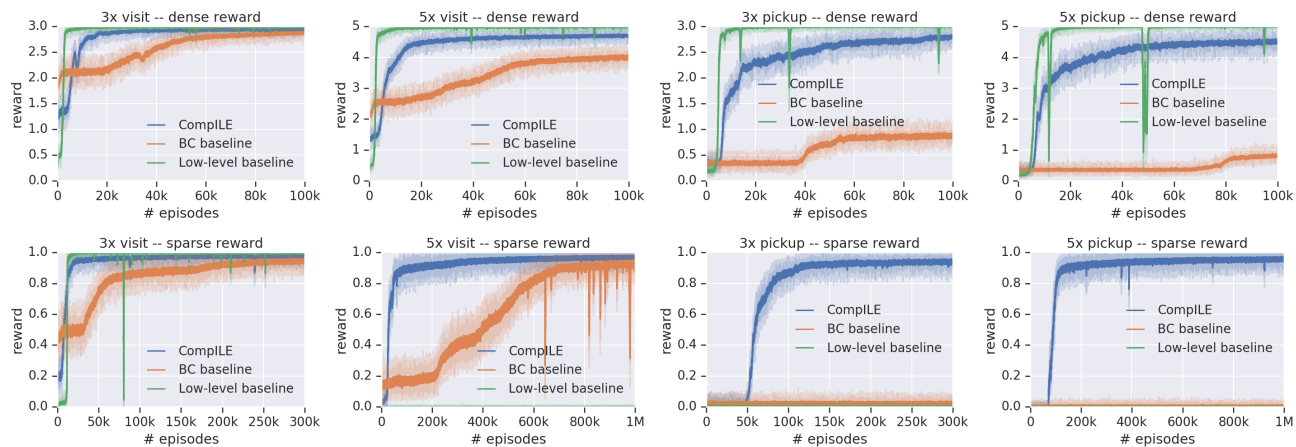


Figure 5: Learning curves for agents trained in the multi-task grid world environment for a single representative seed. We found that the qualitative behavior was consistent across seeds. Original learning curve (reward at every episode) plotted as shaded line; overlaid with solid line using exponential smoothing for easier visibility. BC denotes a VAE-based behavioral cloning baseline that was exposed to the same number of task demonstrations as our CompILE model. The low-level baseline is an agent without internal hierarchy. The CompILE-based hierarchical agent benefits from significantly improved exploration and is the only agent that succeeds at all sparse reward tasks. Best viewed in color.

reached maximum number of 50 steps) was met (*sparse reward* setting). The sparse reward setting poses a very challenging exploration problem: the agent only receives a learning signal if it has completed all tasks from the task list in the correct order, without mistakes (i.e., without picking up a wrong object which could render the episode unsolvable). We compare against a low-level baseline agent that only has access to low-level actions and a VAE-based, pre-trained BC baseline that receives the same pre-training as our CompILE agent, but does not learn a task segmentation (it also has access to low-level actions). All agents use the same CNN-based architecture (see supplementary material for details) and are trained using the distributed policy-gradient algorithm IMPALA (Espeholt et al., 2018). Results are summarized in Figure 5.

The hierarchical agent with sub-task policies from the CompILE model achieves consistent results across all settings and generalizes well to the 5 task setup, even though it has only seen demonstrations of 3 tasks during pre-training. It is the only agent that learns to solve the pick up task setting with sparse reward. The visit task is significantly easier to solve as the episode does not end if a wrong object is visited. Nonetheless, the low-level baseline (without pre-training) fails to learn under the sparse reward setting for all but the *3x visit* task. Only if reward for every individual sub-task is provided, the low-level baseline learns to solve the task in the fewest number of episodes.

4.4. Limitations and future work

As our training procedure is completely unsupervised, the model is free to choose any type of semantics for its la-

tent code. For example, in the grid world environment we found that the model learns a location-specific latent code (with only a small degree of object specificity), whereas the ground truth task list is specific to object type. See supplementary material for an example. It remains to be seen to what degree the latent code can be grounded in a particular manner with only weak supervision, e.g. in a semi-supervised setting or using pairs of demonstrations with the same underlying task list. Furthermore, we have currently only explored fully-observable, Markovian settings. An extension to partially-observable environments will likely introduce further challenges, as the generative model will require some form of recurrency or memory, and the model might learn to ignore the latent code altogether.

5. Conclusions

Here we introduced CompILE, a model for discovering and imitating sub-components of behavior in sequential demonstration data. Our results showed that CompILE can successfully discover sub-tasks and their boundaries in an imitation learning setting, and the latent sub-task encodings can then be used as sub-policies in a hierarchical RL agent to solve challenging sparse reward tasks. While here we explored imitation learning, where inputs to the model are state-action sequences, in principle our method can be applied to any sequential data, and an interesting future direction is to apply our differentiable segmentation and auto-encoding mechanism to other data domains. Future work will investigate extensions for partially-observable environments, its applicability as an episodic memory module, and a hierarchical extension for abstract, high-level planning.

Acknowledgements

We would like to thank Junhyuk Oh, Nicolas Heess, Ziyu Wang, Razvan Pascanu, Caglar Gulcehre, Klaus Greff, Neil Rabinowitz, Andrea Tacchetti, Daniel Mankowitz, Chris Burgess, Irina Higgins, Murray Shanahan, Matthew Willson, Matt Botvinick, and Jessica Hamrick for helpful discussions.

References

- Andreas, J., Klein, D., and Levine, S. Modular multitask reinforcement learning with policy sketches. In *International Conference on Machine Learning*, 2017.
- Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.
- Bacon, P.-L., Harb, J., and Precup, D. The option-critic architecture. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.
- Baldassano, C., Chen, J., Zadbood, A., Pillow, J. W., Hasson, U., and Norman, K. A. Discovering event structure in continuous narrative perception and memory. *Neuron*, 95(3):709–721, 2017.
- Blei, D. M. and Moreno, P. J. Topic segmentation with an aspect hidden markov model. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 343–348. ACM, 2001.
- Chan, W., Zhang, Y., Le, Q., and Jaitly, N. Latent sequence decompositions. In *International Conference on Learning Representations*, 2017.
- Dai, H., Dai, B., Zhang, Y.-M., Li, S., and Song, L. Recurrent hidden semi-markov model. In *International Conference on Learning Representations*, 2017.
- Davidson, D. *Inquiries into Truth and Interpretation*. Clarendon Press, Oxford, 1984.
- Denil, M., Colmenarejo, S. G., Cabi, S., Saxton, D., and de Freitas, N. Programmable agents. *arXiv preprint arXiv:1706.06383*, 2017.
- Escorcia, V., Heilbron, F. C., Niebles, J. C., and Ghanem, B. Daps: Deep action proposals for action understanding. In *European Conference on Computer Vision*, pp. 768–784. Springer, 2016.
- Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, 2018.
- Ezzyat, Y. and Davachi, L. What constitutes an episode in episodic memory? *Psychological Science*, 22(2):243–252, 2011.
- Florensa, C., Duan, Y., and Abbeel, P. Stochastic neural networks for hierarchical reinforcement learning. In *International Conference on Learning Representations*, 2017.
- Fox, R., Krishnan, S., Stoica, I., and Goldberg, K. Multi-level discovery of deep options. *arXiv preprint arXiv:1703.08294*, 2017.
- Fox, R., Shin, R., Krishnan, S., Goldberg, K., Song, D., and Stoica, I. Parametrized hierarchical procedures for neural programming. In *International Conference on Learning Representations*, 2018.
- Goldwater, S., Griffiths, T. L., and Johnson, M. A bayesian framework for word segmentation: Exploring the effects of context. *Cognition*, 112(1):21–54, 2009.
- Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pp. 2672–2680, 2014.
- Graves, A. Supervised sequence labelling. In *Supervised sequence labelling with recurrent neural networks*, pp. 5–13. Springer, 2012.
- Hausman, K., Chebotar, Y., Schaal, S., Sukhatme, G., and Lim, J. J. Multi-modal imitation learning from unstructured demonstrations using generative adversarial nets. In *Advances in Neural Information Processing Systems*, pp. 1235–1245, 2017.
- Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- Jang, E., Gu, S., and Poole, B. Categorical reparameterization with Gumbel-softmax. In *International Conference on Learning Representations*, 2017.
- Jayaraman, D., Ebert, F., Efros, A. A., and Levine, S. Time-agnostic prediction: Predicting predictable video frames. In *International Conference on Learning Representations*, 2018.
- Johnson, M., Duvenaud, D. K., Wiltchko, A., Adams, R. P., and Datta, S. R. Composing graphical models with neural networks for structured representations and fast inference. In *Advances in Neural Information Processing Systems*, pp. 2946–2954, 2016.
- Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

- Kingma, D. P. and Welling, M. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2014.
- Krishna, R., Hata, K., Ren, F., Fei-Fei, L., and Niebles, J. C. Dense-captioning events in videos. In *International Conference on Computer Vision*, pp. 706–715, 2017.
- Krishnan, S., Fox, R., Stoica, I., and Goldberg, K. Ddco: Discovery of deep continuous options for robot learning from demonstrations. In *Conference on Robot Learning (CoRL)*, 2017.
- Kroemer, O., Daniel, C., Neumann, G., Van Hoof, H., and Peters, J. Towards learning hierarchical skills for multi-phase manipulation tasks. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pp. 1503–1510. IEEE, 2015.
- Kulkarni, T. D., Narasimhan, K., Saeedi, A., and Tenenbaum, J. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pp. 3675–3683, 2016.
- Laskey, M., Lee, J., Fox, R., Dragan, A., and Goldberg, K. Dart: Noise injection for robust imitation learning. In *Conference on Robot Learning (CoRL)*, 2017.
- Lee, Y., Sun, S.-H., Somasundaram, S., Hu, E., and Lim, J. J. Composing complex skills by learning transition policies with proximity reward induction. In *International Conference on Learning Representations*, 2019.
- Maddison, C. J., Mnih, A., and Teh, Y. W. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations*, 2017.
- Neitz, A., Parascandolo, G., Bauer, S., and Schölkopf, B. Adaptive skip intervals: Temporal abstraction for recurrent dynamical models. In *Advances in Neural Information Processing Systems*, pp. 9816–9826, 2018.
- Niekum, S., Chitta, S., Barto, A. G., Marthi, B., and Osentoski, S. Incremental semantically grounded learning from demonstration. In *Robotics: Science and Systems*, volume 9. Berlin, Germany, 2013.
- Oh, J., Singh, S., Lee, H., and Kohli, P. Zero-shot task generalization with multi-task deep reinforcement learning. In *International Conference on Machine Learning*, 2017.
- Parikh, A. P., Täckström, O., Das, D., and Uszkoreit, J. A decomposable attention model for natural language inference. In *Conference on Empirical Methods in Natural Language Processing*, 2016.
- Pertsch, K., Rybkin, O., Yang, J., Derpanis, K., Lim, J., Daniilidis, K., and Jaegle, A. Keyin: Discovering subgoal structure with keyframe-based video prediction. *arXiv preprint arXiv:1904.05869*, 2019.
- Radvansky, G. A. and Zacks, J. M. Event boundaries in memory and cognition. *Current opinion in behavioral sciences*, 17:133–140, 2017.
- Rezende, D. J., Mohamed, S., and Wierstra, D. Stochastic backpropagation and approximate inference in deep generative models. In *International Conference on Machine Learning*, 2014.
- Richmond, L. L. and Zacks, J. M. Constructing experience: event models from perception to action. *Trends in cognitive sciences*, 2017.
- Riemer, M., Liu, M., and Tesauro, G. Learning abstract options. In *Advances in Neural Information Processing Systems*, pp. 10445–10455, 2018.
- Sharma, A., Sharma, M., Rhinehart, N., and Kitani, K. M. Directed-info gail: Learning hierarchical policies from unsegmented demonstrations using directed information. In *International Conference on Learning Representations*, 2018.
- Shiarlis, K., Wulfmeier, M., Salter, S., Whiteson, S., and Posner, I. TACO: Learning task decomposition via temporal alignment for control. In *International Conference on Machine Learning*, 2018.
- Sohn, K., Lee, H., and Yan, X. Learning structured output representation using deep conditional generative models. In *Advances in Neural Information Processing Systems*, pp. 3483–3491, 2015.
- Sun, S.-H., Noh, H., Somasundaram, S., and Lim, J. Neural program synthesis from diverse demonstration videos. In *International Conference on Machine Learning*, pp. 4797–4806, 2018.
- Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pp. 3104–3112, 2014.
- Sutton, R. S., Precup, D., and Singh, S. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2): 181–211, 1999.
- Tang, D., Li, X., Gao, J., Wang, C., Li, L., and Jebara, T. Subgoal discovery for hierarchical dialogue policy learning. In *Conference on Empirical Methods in Natural Language Processing*, 2018.

- Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., Casas, D. d. L., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.
- Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., and Kavukcuoglu, K. FeUdal networks for hierarchical reinforcement learning. In *International Conference on Machine Learning*, 2017.
- Vinyals, O., Toshev, A., Bengio, S., and Erhan, D. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3156–3164, 2015.
- Wang, C., Wang, Y., Huang, P.-S., Mohamed, A., Zhou, D., and Deng, L. Sequence modeling via segmentations. In *International Conference on Machine Learning*, 2017.
- Zacks, J. M., Tversky, B., and Iyer, G. Perceiving, remembering, and communicating structure in events. *Journal of Experimental Psychology: General*, 130(1):29, 2001.