



CompLE: Compositional Imitation Learning and Execution

Thomas Kipf, Yujia Li, Hanjun Dai, Vinicius Zambaldi, Alvaro Sanchez-Gonzalez,
Edward Grefenstette, Pushmeet Kohli, Peter Battaglia - ICML (2019)

Overview



- What is CompILE?
- Baseline behavioral cloning approach
- Compositional Imitation Learning
 - ◆ Introduction to CompILE approach
 - ◆ Background info on Gumbel-Softmax and VAEs
 - ◆ Methodology
- Experiments
- Results
- Conclusion

What is a CompILE?

CompILE is an imitation learning framework that aims to identify, learn and execute **sub-skills** from demonstrations instead of learning them as a whole.

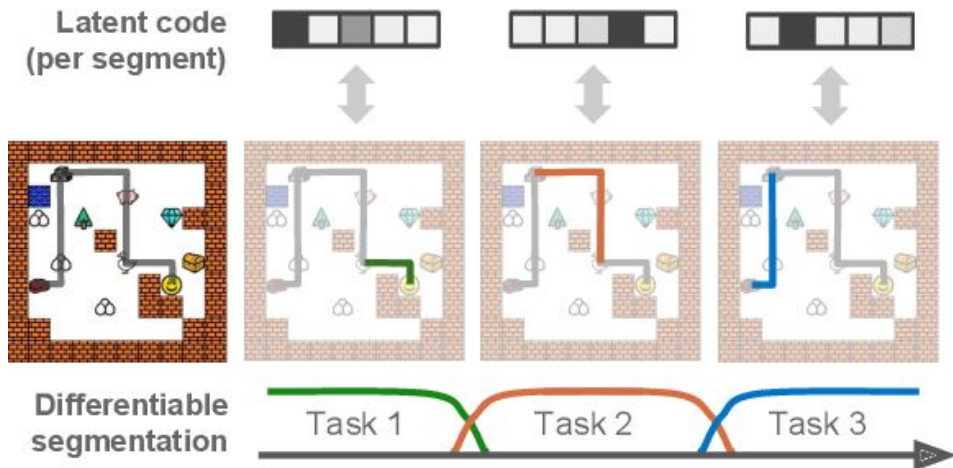
Allows:

- generalization
- hierarchical planning
- episodic memory / compression



What is a CompILE?

- Jointly learn to:
 - Identify segments of behaviour
 - Auto-encode each segment into latent code
- **Unsupervised** training
- **Fully differentiable**
- **VAE** + additional structure on latent code
- Inductive bias for **generalization to longer sequences**



Imitation Learning



Demonstrations: $\rho = ((s_1, a_1), (s_2, a_2), \dots, (s_T, a_T))$

Behavioural cloning: Simple imitation learning approach, not compositional.

Learn imitation policy $\pi_{\theta}(a | s)$ by maximum likelihood:

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

with: $p_{\theta}(a_{1:T} | s_{1:T}) = \prod_{t=1:T} \pi_{\theta}(a_t | s_t)$

* Used as the baseline method in ComPILE experiments

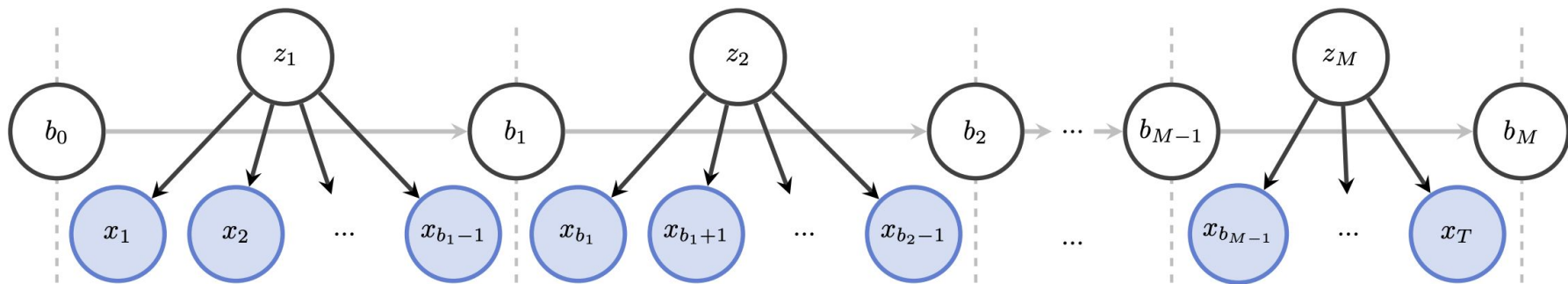
Compositional Imitation Learning



- Discover reusable chunks of action
- Reuse discovered chunks to solve other tasks
- Introduce 2 sets of latent variables: Chunk boundaries $\mathbf{b}_{1:M}$ and latent codes $\mathbf{z}_{1:M}$

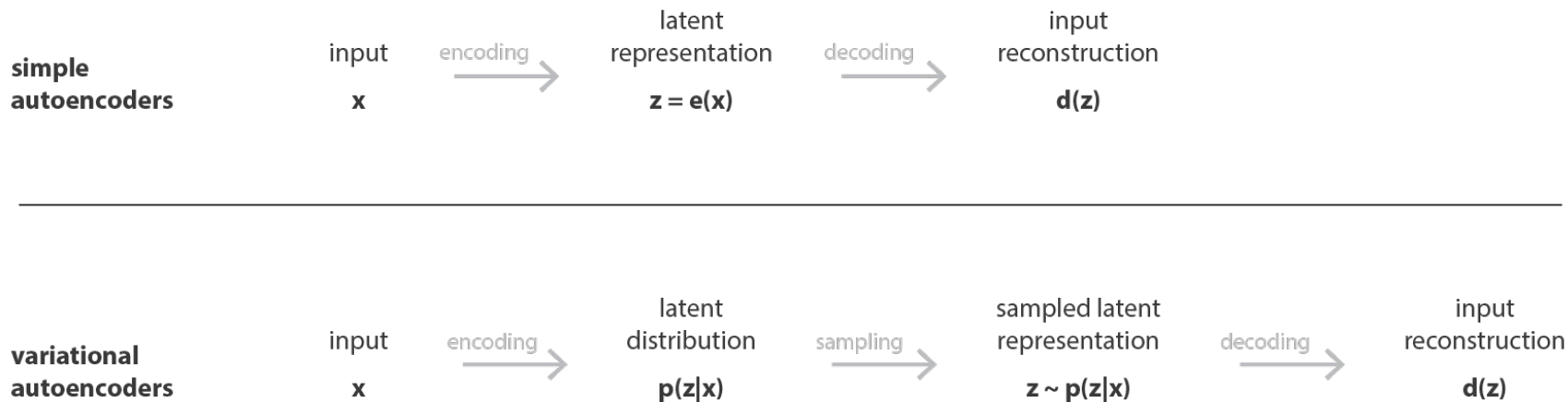
Compositional Imitation Learning

- Discover reusable chunks of action
- Reuse discovered chunks to solve other tasks
- Introduce 2 sets of latent variables: Chunk boundaries $\mathbf{b}_{1:M}$ and latent codes $\mathbf{z}_{1:M}$

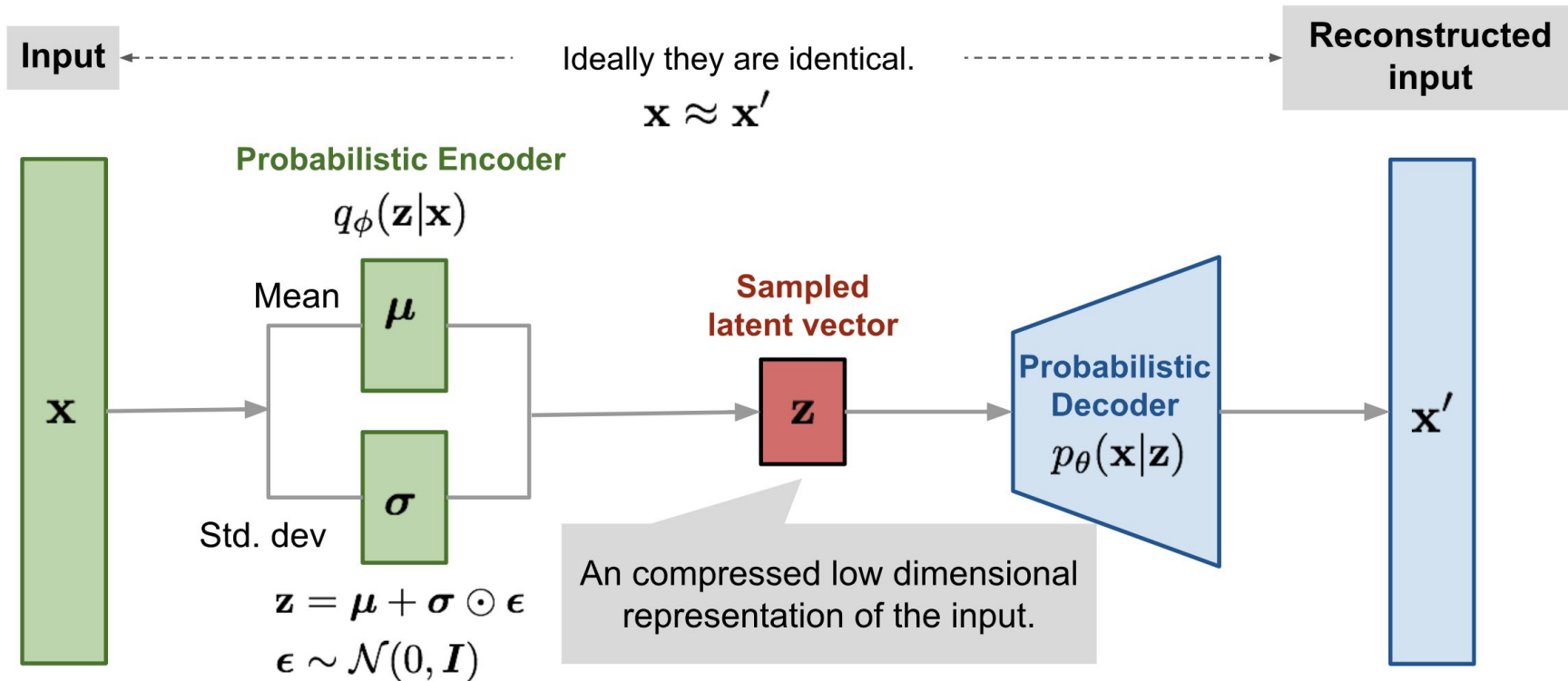


Variational Auto-Encoder (VAE)

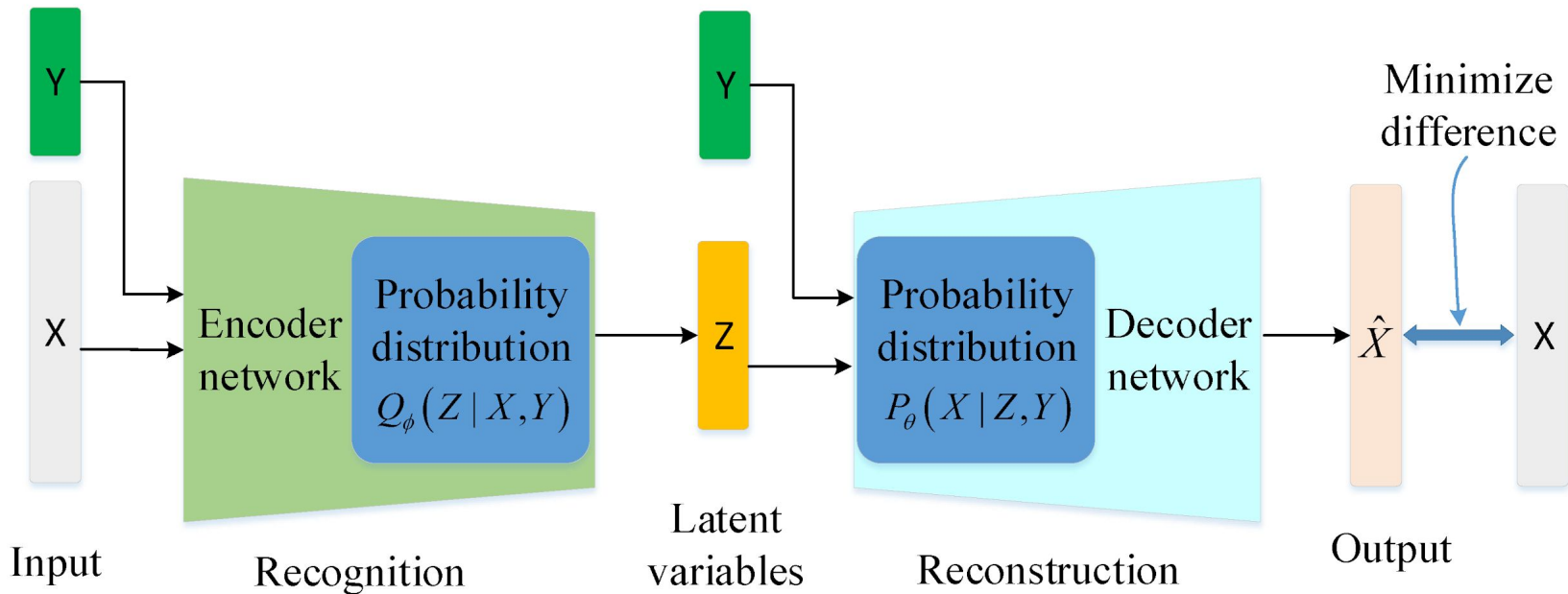
Instead of encoding an input as a single point, VAEs encode the input as a **distribution over the latent space**.



Variational Auto-Encoder (VAE)



Conditional Variational Auto-Encoder (VAE)



Compositional Imitation Learning



Working with discrete variables - Gumbel-Softmax Distribution:

Convert categorical distributions into continuous distributions to allow for optimization via backpropagation.

How: The Gumbel-Max trick provides a simple a simple way to draw samples (one-hot vector) from a categorical distribution with class probabilities π :

$$\mathbf{z} = \text{one-hot}(\operatorname{argmax}_i [g_i + \log \pi_i]) \quad \text{where} \quad \mathbf{g} = -\log(-\log(\mathbf{u})), \mathbf{u} \sim \text{Uniform}(0, 1)$$

As the operator *argmax* is not differentiable we use the *softmax* function as a continuous, differentiable approximation and generate the (approximated) samples :

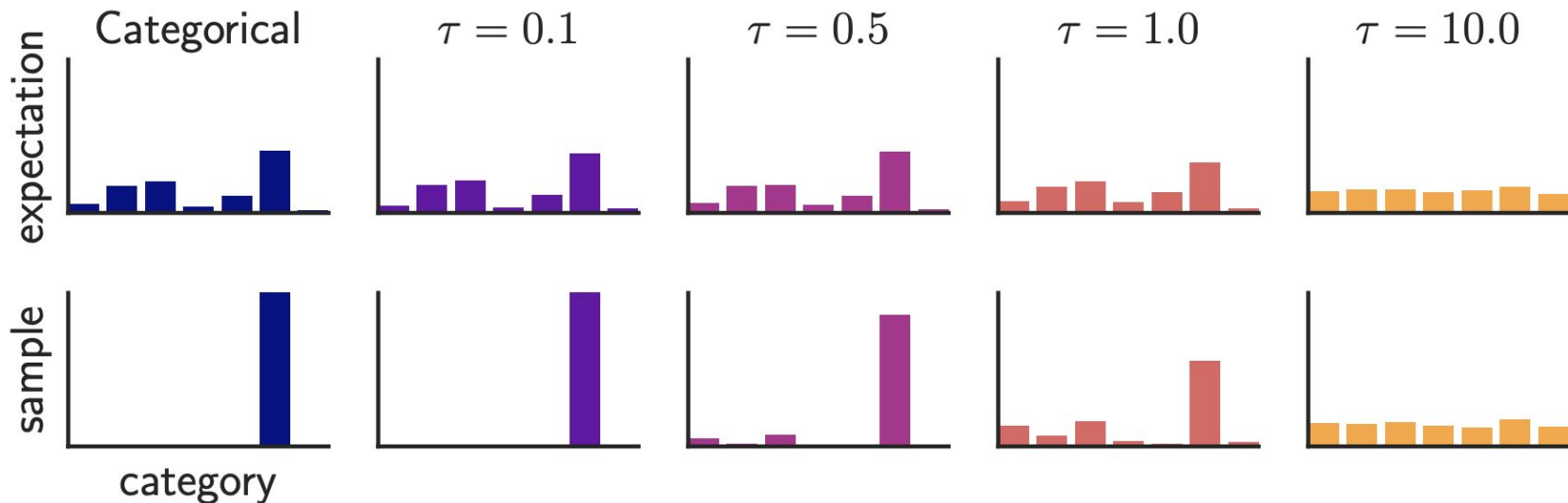
$$\mathbf{z} = \text{softmax}((\log(\pi) + \mathbf{g})/\tau) \quad \text{where } \tau \text{ is a constant referred as temperature.}$$

Compositional Imitation Learning



Working with discrete variables - Gumbel-Softmax Distribution:

The Gumbel-Softmax distribution interpolates between discrete one-hot-encoded categorical distributions and continuous categorical densities.



Methodology



Learning with a conditional VAE approach:

- Encoder / inference: $q_{\phi}(b_{1:M}, z_{1:M} | a_{1:T}, s_{1:T})$

Infer chunk boundaries and latent code from demonstrations

- Decoder / execution: $\pi_{\Theta}(a | s, z)$ compute actions given state and latent code
- Prior: $p_{\Theta}(b_{1:M}, z_{1:M})$ make sure segments are not too long and all codes / sub-behaviours are represented

Problem: Boundary variables are discrete.

→ Relax discrete segmentation via **soft segment masks**

Methodology

→ Break demonstration trajectories p into M disjoint segments (c_1, c_2, \dots, c_M):

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

* M is a hyperparameter, $i' = i - 1$, b_i denote the **artificial boundary** timesteps.

E.g. for $M = 3$, we can segment a demonstration sequence as follows:

b_0 b_1 b_2 b_M
| (s1, a1), (s2, a2) | (s3, a3), (s4, a4) | (s5, a5), (s6, a6) |

$c_1 = (s_1, a_1), (s_2, a_2), (s_3, a_3), (s_4, a_4)$

$c_2 = (s_3, a_3), (s_4, a_4), (s_5, a_5), (s_6, a_6)$

$c_3 = (s_5, a_5), (s_6, a_6), \text{padding}, \text{padding}$

Methodology



We are trying to get the latent distribution of \mathbf{b}_i and \mathbf{z}_i given input sequence of states and actions \mathbf{x} : $\mathbf{p}(\mathbf{b}_i | \mathbf{x})$ and $\mathbf{p}(\mathbf{z}_i | \mathbf{x})$ for $i = 0, \dots, M$

→ Assume independent priors over \mathbf{b} and \mathbf{z} :

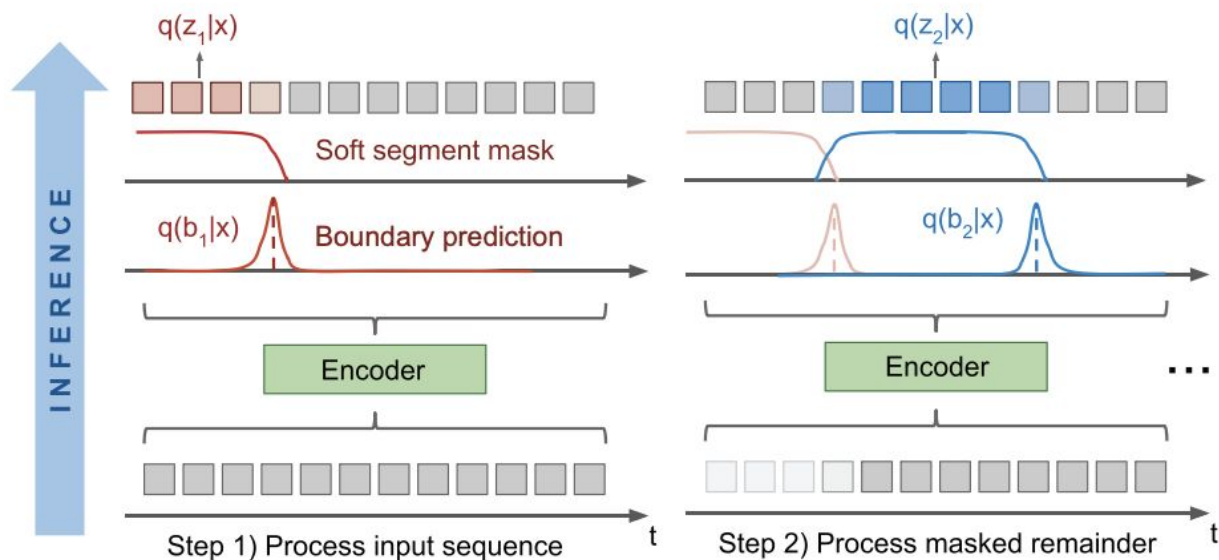
$$p(b_i, z_i | b_{1:i'}, z_{1:i'}) := p(b_i | b_{i'}) p(z_i)$$

A uniform categorical prior $\mathbf{p}(\mathbf{z}_i)$ and a categorical prior for the boundary latent variables $\mathbf{p}(\mathbf{b}_i)$:

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

Methodology

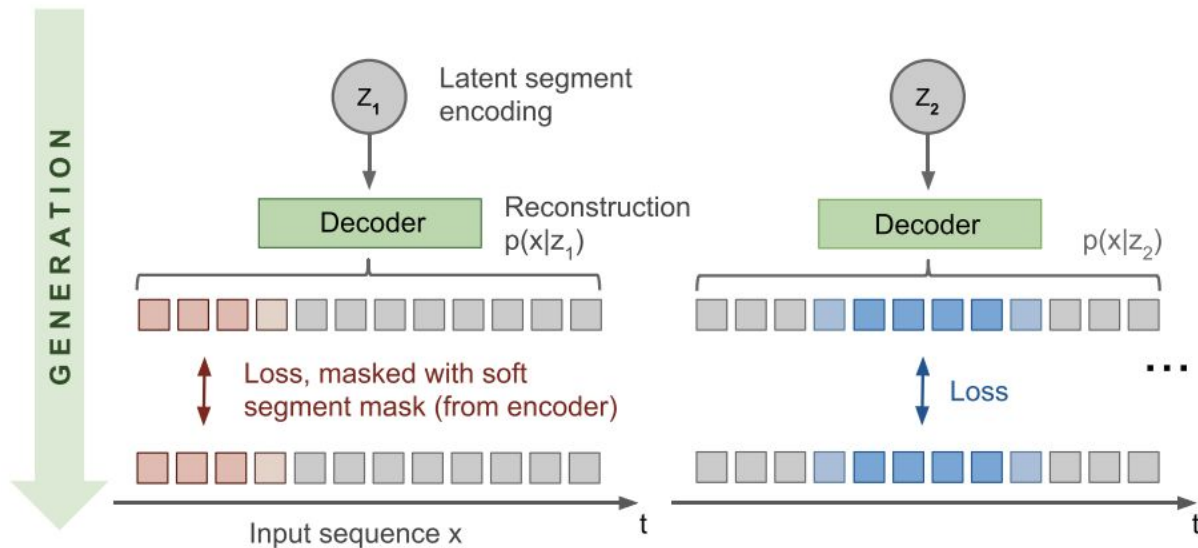
- Prior distributions $p(b_i)$ and $p(z_i)$
- Train VAE to get the posterior distributions $q(b_i | x)$ and $q(z_i | x)$
- Threshold the boundary distributions $q(b_i | x)$ to mask the input sequences and apply a soft mask to $q(z_i | x)$



Methodology

Given a list of tasks (real boundaries), for each task:

- Find corresponding boundaries and latent code with the highest likelihood $p(\mathbf{b}_i | \mathbf{b}_i) p(\mathbf{z}_i)$
- Reconstruct the state-action sequence
- Execute the sequence

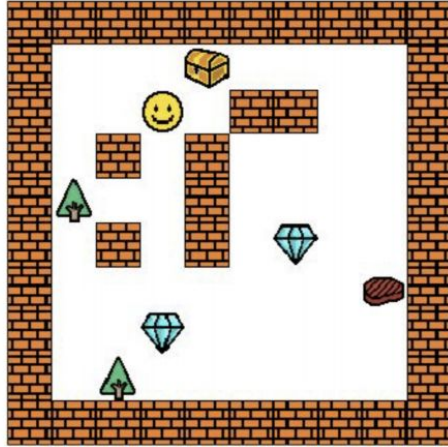


Experiments - Evaluation Criteria



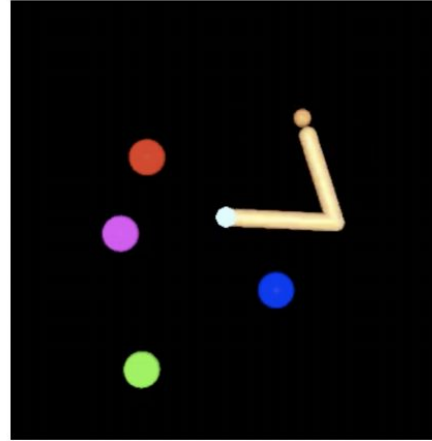
1. Learning to find task boundaries and task encodings while being able to reconstruct and imitate unseen behavior
2. Testing whether task decomposition allows ComplLE to generalize to longer sequences with more sub-tasks at test time.
3. Investigating whether an agent can learn to control the discovered sub-task policies to quickly learn new tasks in sparse reward settings.




Experiments - Test Environments



1. 
2. 
3. 

Grid world with walls. An agent has to pick up or visit certain objects.



1. 
2. 
3. 

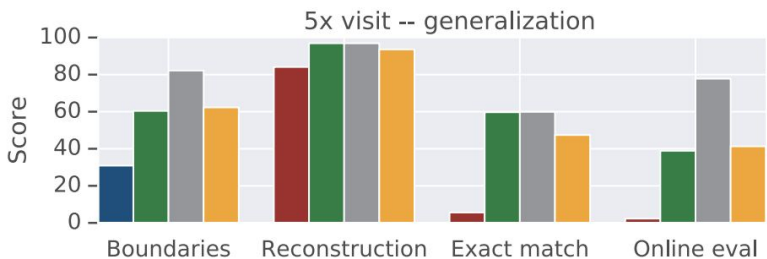
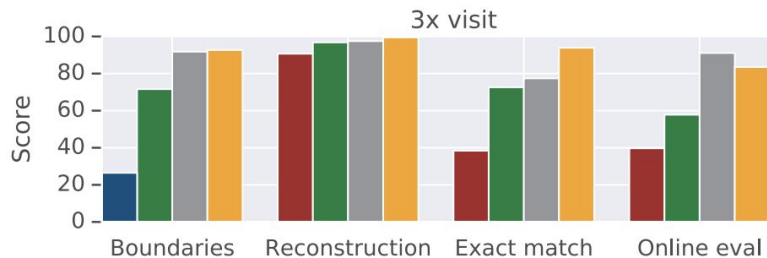
Continuous control reacher task with multiple targets. The tip of the reacher arm has to touch multiple colored spheres in a pre-specified order.

Experiments - Test Environments



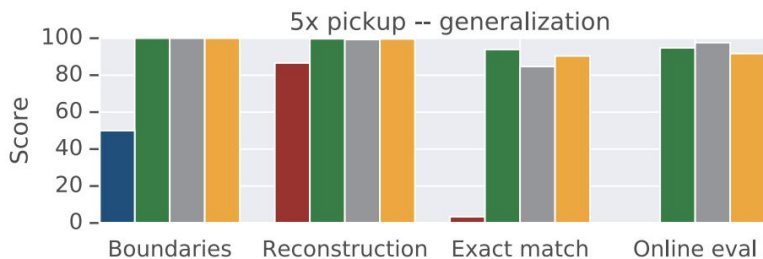
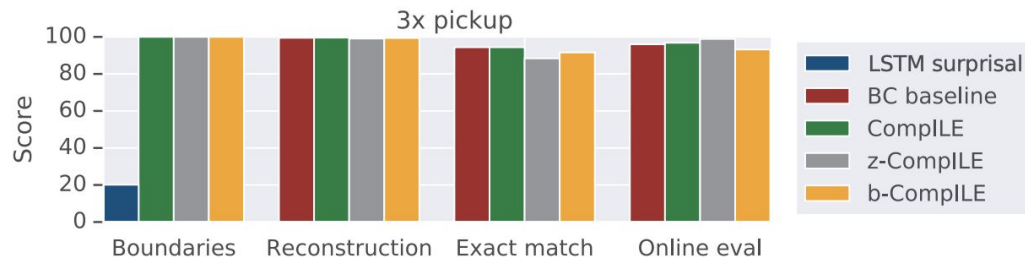
- There are 10 different object types in each test environment
- To generate a demonstration:
 - ◆ Place up to 6 targets drawn without replacement from 10 different target types
 - ◆ Generate a task list of 3-5 objects to visit / pick up
(sub-tasks with discrete boundaries)
- Use a shortest path algorithm (2D grid) and a hand-coded control policy (continuous control reacher) to generate expert demonstrations.

Results - 2D Grid World



Boundaries: percentage of correctly predicted boundaries

Reconstruction: average reconstruction accuracy of the original action sequence



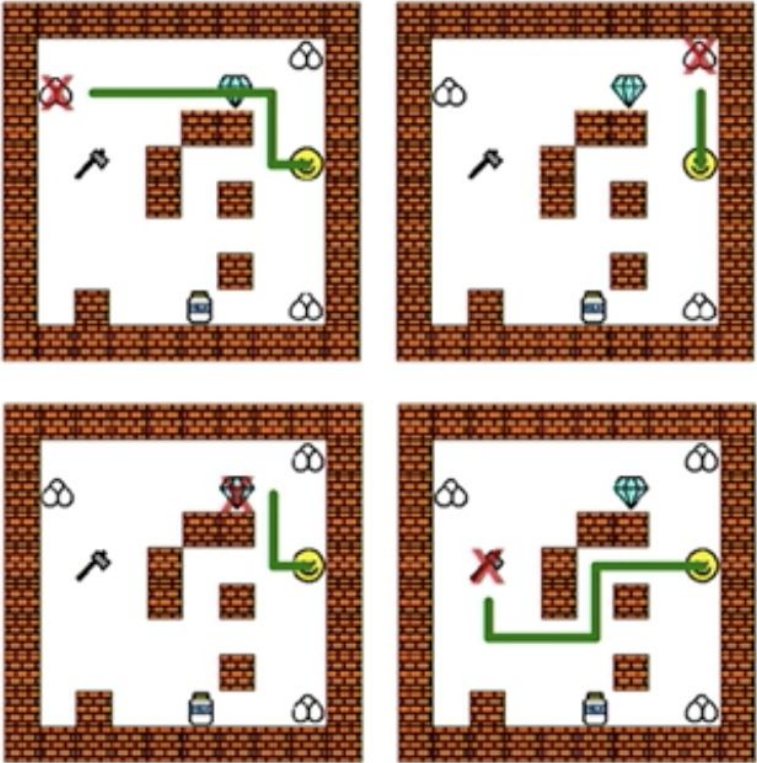
Exact match: percentage of exact full-sequence reconstruction matches

Online evaluation: percentage of avg. reward obtained / maximum reward

Results - 2D Grid World



Demonstration



Results - Continuous Control Reach



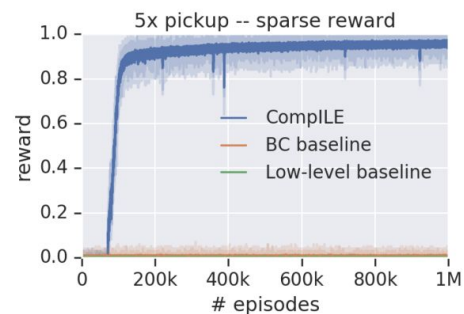
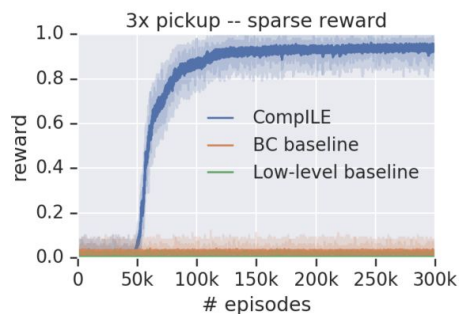
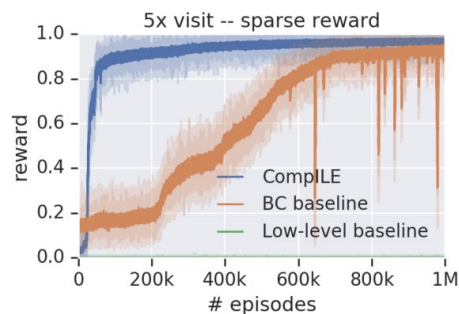
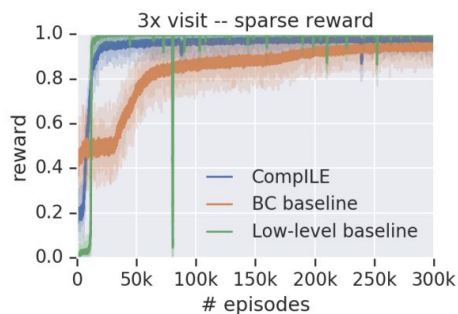
Model	Accuracy	F1 (tol=0)	F1 (tol=1)
3 tasks			
LSTM surprisal	24.8 ± 0.6	39.0 ± 0.3	47.1 ± 0.4
CompILE	62.0 ± 4.5	74.3 ± 3.3	78.9 ± 2.5
z-CompILE	99.5 ± 0.2	99.7 ± 0.2	99.8 ± 0.1
b-CompILE	99.8 ± 0.1	99.9 ± 0.1	100 ± 0.0
5 tasks – generalization			
LSTM surprisal	21.6 ± 0.5	44.9 ± 0.5	54.4 ± 0.5
CompILE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

Accuracy: percentage of correctly executed tasks

F1: correct boundary step predictions

Results - Continuous Control Reach

Teach agent to follow a list of tasks - reward only at the end of the episode



Conclusion and Future Work



- ComplLE can successfully discover sub-tasks and their boundaries in an imitation learning setting
- While inputs to the model are state-action sequences, ComplLE can be applied to any sequential data
- ComplLE learns location-specific latent code whereas the ground truth task lists are often specific to object type
- Future work will investigate extensions for:
 - partially-observable environments
 - ComplLE's applicability as an episodic memory module
 - a hierarchical extension for abstract, high-level planning

References



- Kipf, Thomas et al. “CompILE: Compositional Imitation Learning and Execution.” ICML (2019).
- Sohn, Kihyuk et al. “Learning Structured Output Representation using Deep Conditional Generative Models.” NIPS (2015).