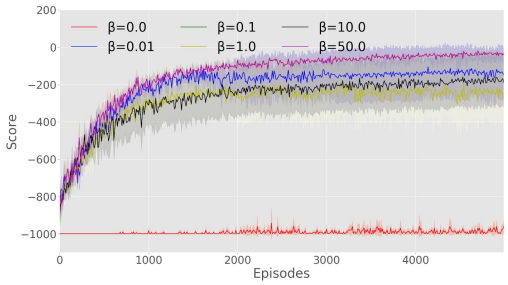
Supplementary Materials for Reinforcement Learning in Latent Action Sequence Space

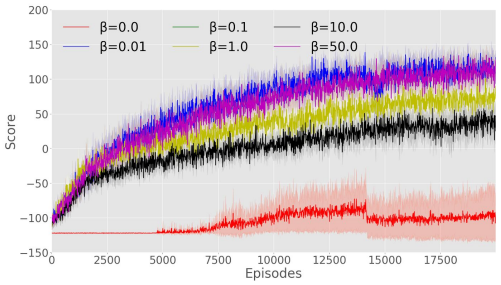
1 Additional figures

1.1 Comparison along different β on RobotHand

We present additional comparisons along different $\boldsymbol{\beta}$ on RobotHand Reaching and BallPlacing.



Comparison between different β on Reaching



Comparison between different β on BallPlacing

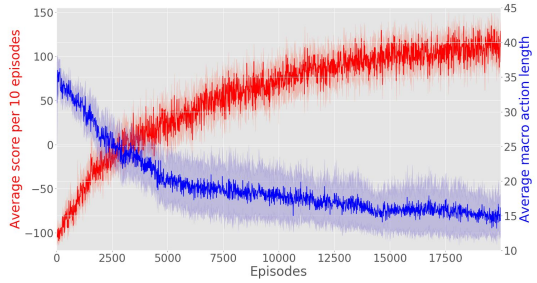
We found that $\beta=0.1$ reached highest score on RobotHand.

1.2 Length of macro actions on BallPlacing

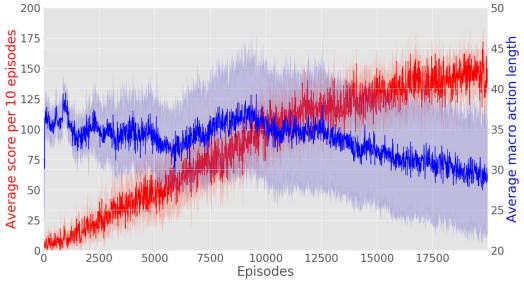
The reward function of BallPlacing is defined by $\boldsymbol{-|b-g|}$ where $\boldsymbol{b}$ is the position of the ball and $\boldsymbol{g}$ is the position of the goal. And additional $\boldsymbol{+200}$ is given when the ball reaches the goal.

We plotted length of macro actions of **1)** BallPlacing with time penalty ($\boldsymbol{-|b-g|+200}$ when the ball reaches the goal) and **2)** BallPlacing without time penalty ($\boldsymbol{+200}$ only when the ball reaches the goal).

Similar to the result of Reaching, the length of macro actions diminishes only when there is time penalty.



BallPlacing with time penalty



BallPlacing without time penalty

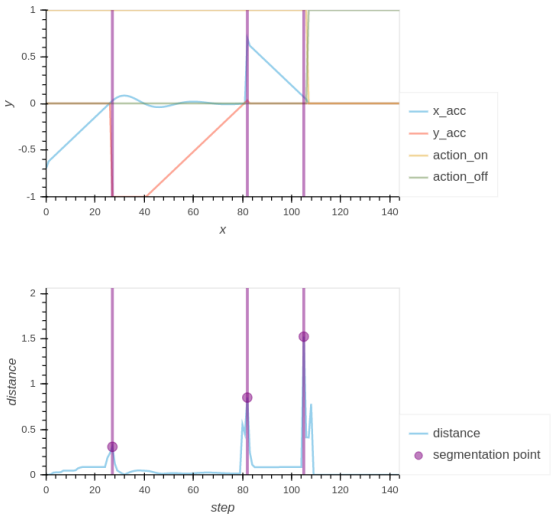
2 Expert actions and segmentation result

In this section, we exhibit examples of expert actions used in the paper, and its segmentation. Following plot shows dataset (top) and distance between adjacent features (bottom) and segmentation point (purple). All dataset can be visualized by running ./visualization.ipynb

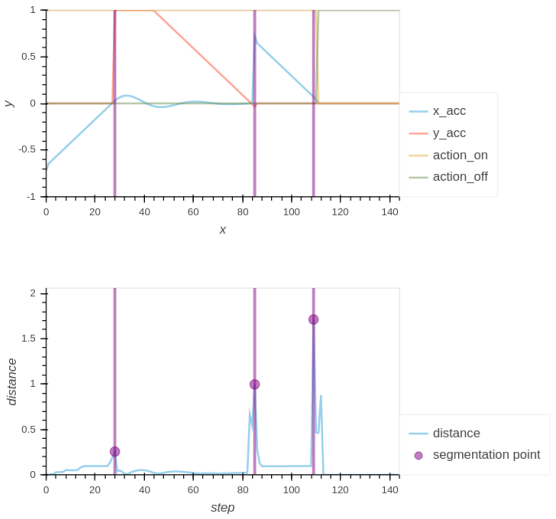
2.1 ContinuousWorld

2.1.1 Down&Up

In Down&Up, the agent and goal are initialized at the end of each corner; top or bottom respectively. Thus, actions contain acceleration toward **1)** up, **2)** down, **3)** right, **4)** left. The actions were segmented at where there were large changes in actions.



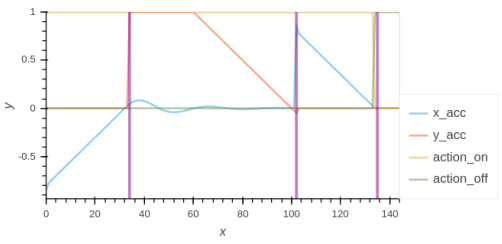
example 1

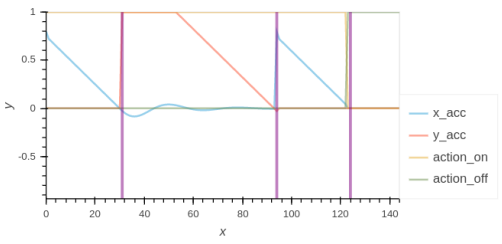


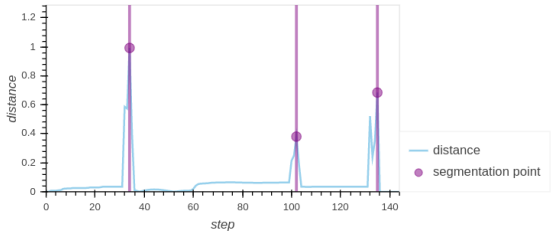
example 2

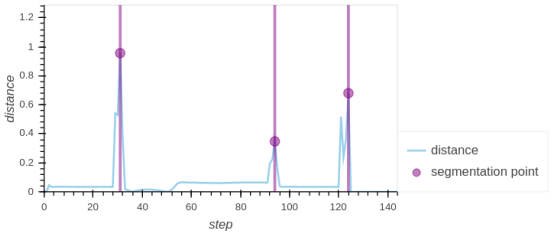
2.1.2 DownOnly

In DownOnly, the agent is initialized at top corners and the goal is initialized at bottom corners. Actions contain acceleration toward **1)** down, **2)** right, **3)** left. It does not contain acceleration toward upward.





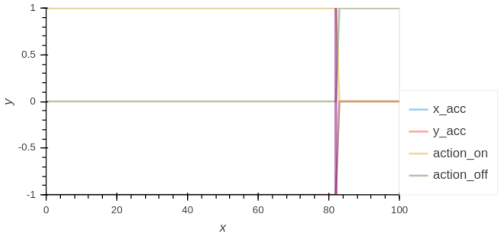


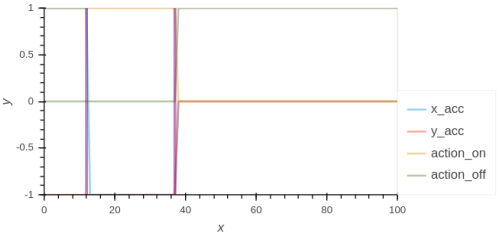


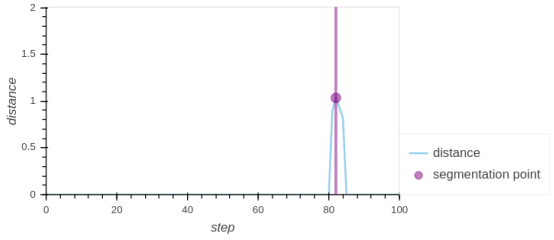
example 1 **example 2**

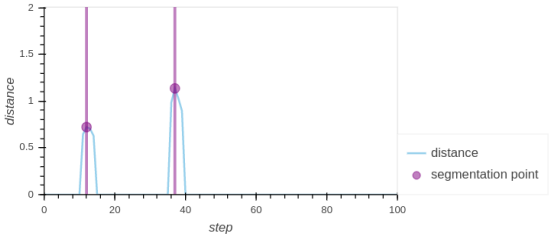
2.1.3 PushedDown&Up

In PushedDown&Up, the initialization of the agent and goal is identical to Down&Up. However, the agent always accelerates to upward or downward (***y_acc=1*** or ***-1***). Thus, actions contain acceleration toward **1)** up and right, **2)** up and left, **3)** down and right, **4)** down and left.





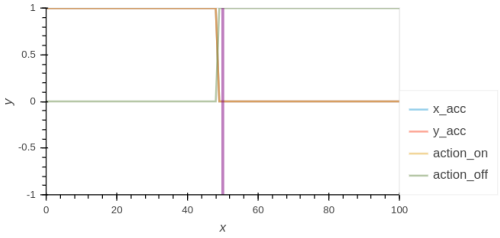


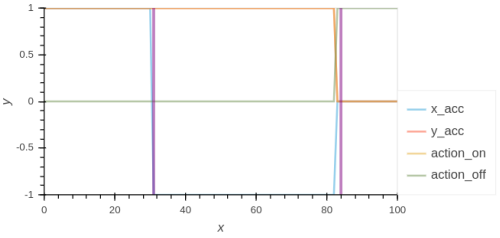


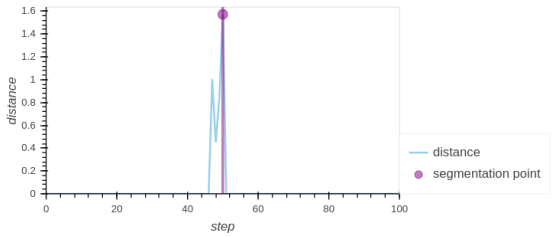
example 1 **example 2**

2.1.4 PushedDownOnly

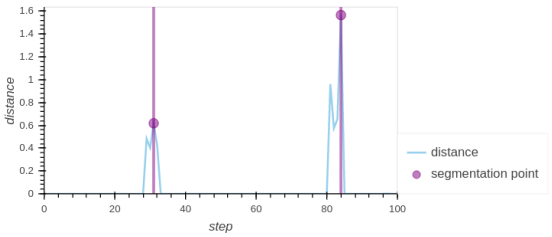
In PushedDownOnly, the initialization of the agent and the goal is identical to DownOnly. However, the agent always accelerates to downward (***y_acc=-1***). Thus, actions contain acceleration toward **1)** down and right, **2)** down and left.







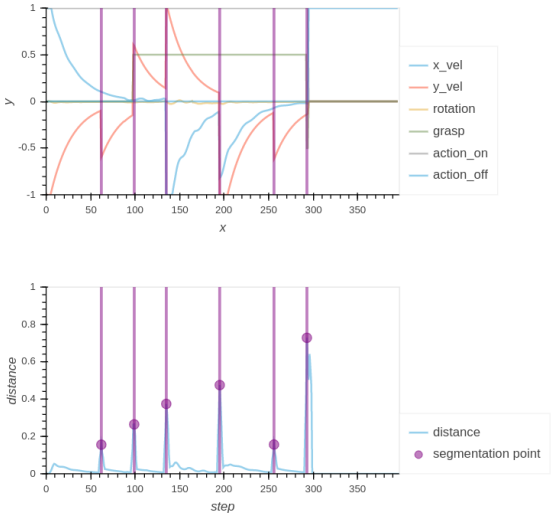
example 1



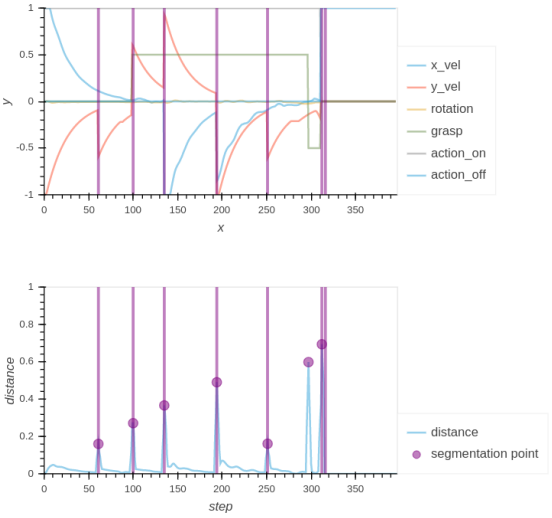
example 2

2.2 RobotHand

In RobotHand, the expert demonstration is generated at Base task, operated by programmed script. The robot hand **1)** moves toward the rod, **2)** grasp the rodm, **3)** get back to the starting position, while grasping the rod on its hand, **4)** moves toward the target basket, **5)** release.



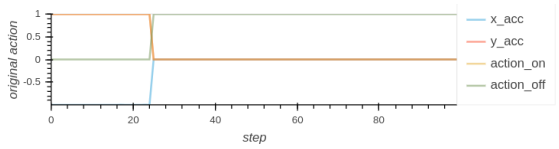
example 1



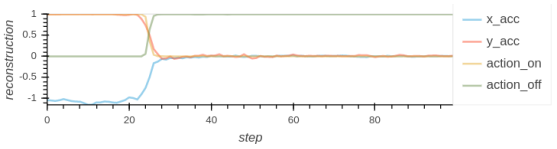
example 2

3 Disentangled latent variables (PushedDownOnly)

In this section, we visually exhibit that FAVAE learned disentangled latent variables. FAVAE learns reconstruction from the original data. You can check what factor that a latent variable (z) learned by fixing all other latent variables and only changing that latent variable. This technique is called latent traversal. We exhibit an example of latent traversal on PushedDownOnly.



Original actions

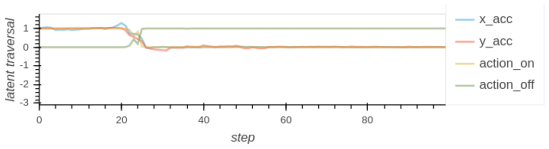


Reconstruction by FAVAE

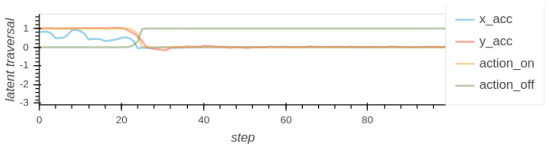
3.1 1st z in 1st ladder

All other latent variables are fixed and only 1st latent variable (z) in 1st ladder was changed from ***-2.0*** to ***+2.0***. This exclusively changed acceleration of x axis and did not change any of other factors.

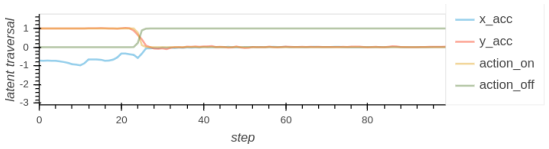
z=-2.0



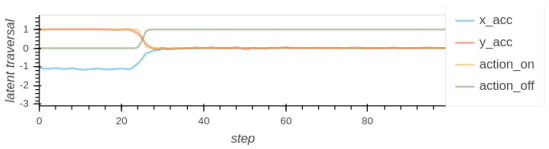
z=-1.0



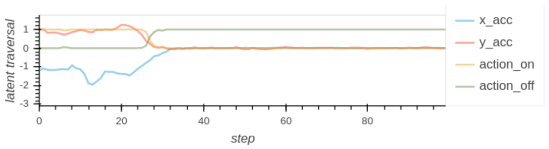
z=0.0



z=+1.0



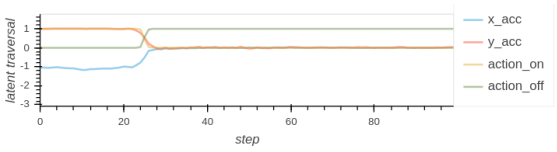
z=+2.0



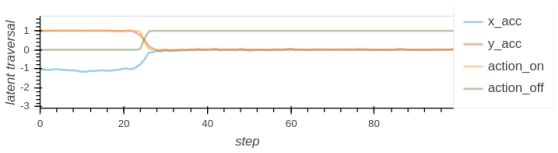
3.2 3rd z in 1st ladder

Latent traversal of 3rd z in 1st ladder did not change anything, because this latent variable did not learn any factor. This happens when number of latent variables exceed number of factors and latent variables are well disentangled.

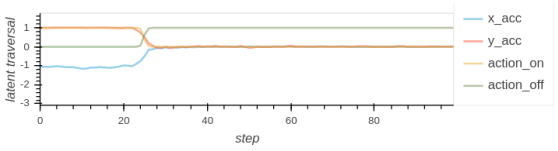
z=-2.0



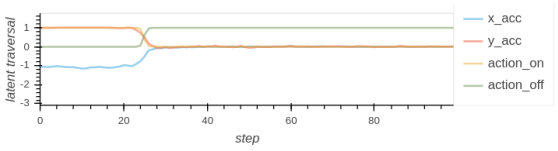
z=-1.0



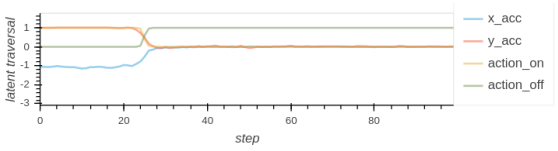
z=0.0



z=+1.0



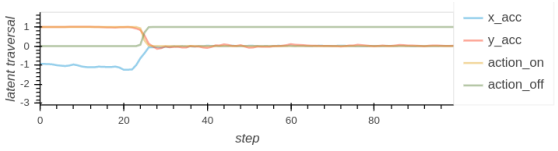
z=+2.0



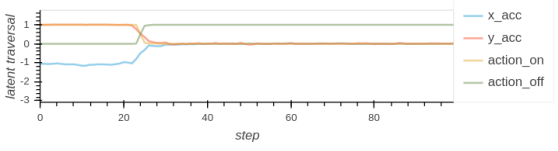
3.3 1st z in 3rd ladder

Latent traversal on 1st z in 3rd ladder changes lengths of actions. The timing of ***action off*** is selected is postponed with higher value of the latent variable.

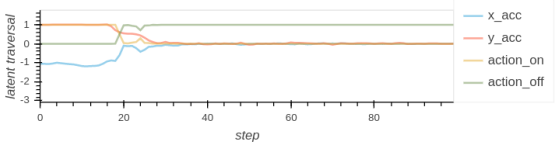
z=-2.0



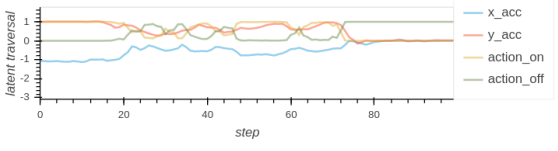
z=-1.0



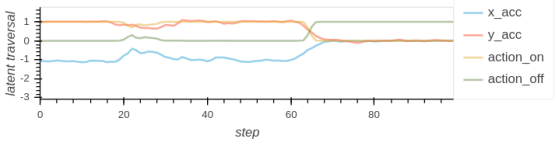
z=0.0



z=+1.0



z=+2.0



4 Experimental setups

4.1 Hyper-parameters setup

Action Segmentation

Hyper-parameters\Environments	ContinuousWorld	RobotHand
random seed	190312	190312
number of latent variables	4	4
window size	4	4
learning rate	1e-3	1e-3
epochs	500	500
batch size	32	32

We found that only a few epochs can learn enough features to compute distance between adjacent windows. Thus we used epochs of 500 to train autoencoder. Also, window size can be small (4 for this time), because the entire length of demonstration is relatively short (<400).

FAVAE

Hyper-parameters\Environments	ContinuousWorld	RobotHand
random seed	190312	190312
number of latent variables	[4, 2, 1]	[12, 8, 4]
learning rate	1e-3	1e-3
epochs	10000	10000
batch size	128	128
β	0.0, 0.01, 0.1, 1.0, 10.0, 50.0	0.0, 0.01, 0.1, 1.0, 10.0, 50.0
c	DownOnly =[12.0, 5.5, 3.1], Down&Up =[11.0, 5.3, 2.0], PushedDownOnly =[12.1, 5.8, 3.7], PushedDown&Up =[13.0, 5.6, 3.4]	[29.3, 8.0, 5.7]

Number of latent variables indicates [1st ladder, 2nd ladder, 3rd ladder]. We have squeezed number of parameters until reconstruction starts to collapse. Also, number of parameters on lower ladder is bigger than higher ladder.

C is applied to each ladder respectively. C is decided based on following steps:

1. Train FAVAE with small β. We used β=0.1.
2. Check last kl divergence loss on each ladder. This kl losses are set to c.
3. Train FAVE with decided c on targeted β.

We have searched β on [0.0, 0.01, 0.1, 1.0, 10.0, 50.0]. Comparison between β has been performed at Figure 5.a in paper and 1.1 in this supplement.

Reinforcement learning

We used PPO implementation at https://github.com/Anjum48/rl-examples. All hyper-parameters are identical to this github repository's, except for learning rate which is changed from 1e-3 -> 1e-4. We searched learning rate among [1e-2, 1e-3, 1e-4, 1e-5] with primitive action, and found that learning rate=1e-4 is best for our environment.

To prevent cherry picking of good results, we repeated each experiment 10 times with random seeds [894492, 101745, 828267, 782861, 412907, 219180, 489237, 584400, 716014, 734948] across all experiments.

4.2 Computing infrastructures

On Action Segmentation and FAVAE, we used following setup:

CPU	Intel(R) Core(™) iu-4720HQ CPU @ 2.60GHz
GPU	None
Memory	7894MB

On learning policy with PPO, we used:

CPU	Intel(R) Xeon(R) CPU E5-2623 v3 @ 3.00GHz
GPU	4 GeForce GTX TITAN X
Memory	125GB

4.3 Software infrastructures

We used Python 3.5.2 with Pytorch 0.4.1 across all experiments.

PPO implementation we used can be found at: https://github.com/Anjum48/rl-examples