Meta-Learning for Automatic Skill Extraction through Imitation Learning

Ahmed M. Ahmed
Stanford University, Stanford, CA
ahmedah@stanford.edu

1 Introduction

A highly desirable property of autonomous robots is to solve long-horizon, compositional tasks. This is because such tasks are often realistic and solving them could allow for the reuse of individually learned skills as necessary. For example, consider a robot operating in a kitchen which can execute multiple skills such as gathering utensils, operating a stove or preparing ingredients where each skill is useful by itself, and we can compose skills for useful long-horizon behaviors such as making a full meal. However, learning such skills is a difficult problem because it is prohibitively slow to learn from scratch for each skill , it is difficult to leverage past experience from data for different skills (5), and it is often easier to provide demonstrations than designing a reward function for complex tasks limiting the viability of RL (31). Additionally, prior work has shown that learning hierarchical policies in general is quite difficult (28; 32). Finally, safety is of critical concern in imitation learning because of covariate shift (27) and recent work has shown that bayesian methods are best suited to mitigating this risk due to calibrated uncertainty estimates (10) adding additional constraints to an ideal approach.

One promising approach to tackling this multifaceted problem is to learn from demonstrations through *imitation learning*, which requires no reward function and can lead to exponential decreases in sample complexity (22; 30). One other promising tool is *meta-learning*, which allows few-shot learning to quickly generalize across a variety of tasks (5). In this work, we combine both classes of algorithms to address the hierarchical imitation learning problem (18), which can be seen as a formalization of the desiderata described above. Recent methods (23; 11; 24; 17) address this problem formulation by leveraging a variational auto-encoder (16) to pretrain on a dataset of expert trajectories for unsupervised skill learning before few-shot learning on target demos to be imitated during test time. However, these works either 1) assume a fixed horizon for each skill, which is unrealistic for all skills and could make execution take longer if some skills are shorter than the choice of horizon, 2) rely on downstream fine-tuning schemes to guide the agent towards executing the correct order of skills presented at test time, which is undesirable as we'd like our agent to automatically infer which skills to execute directly from test-time demonstrations or 3) require multiple passed over trajectories to segment latent skills which can be computationally expensive.

In this work we tackle this problem by leveraging a variational inference procedure through the lens of meta-learning where we alternate between computing the expectation of the run-lengths in a given trajectory, and maximizing the ELBO assuming the run-lengths are fixed which we refer to as meta-learning for automatic skill extraction through imitation learning (MASEIL). This allows us to learn skills of variable-length and automatically infer the skills to be executed during test-time assuming only that MASEIL knows the number of skills to be extracted, and MASEIL is comprised of bayesian meta-learning algorithms in both steps that explicitly reason over uncertainty, allow for efficient reuse of data for learning all skills, and optimize for few-shot adaptation. We demonstrate promising initial experimental results, and sketch some preliminary proofs that MASEIL can recover the optimal run-lengths of our latent skills as well as improved theoretical optimization property over approaches that forgo meta-learning, giving it credence as a principled method of tackling the hierarchical imitation learning problem.

Preprint. Under review.

2 Preliminaries

2.1 Meta-Learning

We first introduce meta-learning (33; 34), whose core idea is to optimize the few-shot learning capacity of a given model over a distribution of tasks such that the model can quickly adapt to new tasks from this distribution when given a small amount of data.

Formally, we seek to learn a function $\boldsymbol{f_\theta}$ that takes in a given dataset composed of k input-output pairs $\mathcal{D} = \{(\boldsymbol{x}, \boldsymbol{y})_{1:k}\}$ and a set of test inputs $\boldsymbol{x}_{\text{test}}$ to output test predictions $\boldsymbol{y}_{\text{test}}$ (5). This function is learned by training over a meta-training set composed of tasks $\mathcal{T}_i \sim p(\mathcal{T})$ where each task samples a training dataset, and test input-output pairs $\{\mathcal{D}_t, \boldsymbol{x}_{\text{test}}, \boldsymbol{y}_{\text{test}}\} \sim \mathcal{T}_i$. During meta-testing we are given a new task $\mathcal{T}_{\text{test}} \sim p(\mathcal{T})$ for which we want to evaluate the performance of $\boldsymbol{f_\theta}$ in predicting the test outputs given the test inputs and k samples for task-specific adaptation.

Furthermore, we adopt a Bayesian perspective of meta-learning (8; 6). Assuming that $\boldsymbol{\theta}$ are the prior parameters learned during meta-training, we can view meta-testing as using the context data $\mathcal{D}_{\text{test}}$ to compute statistics $\boldsymbol{\eta} = \boldsymbol{f_\theta}(\mathcal{D}_{\text{test}})$ which can be thought of as task-adapted parameters. We leverage these statistics within a posterior predictive distribution $p_\theta(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{\eta})$, and our test-time loss is

$$\mathcal{L}(\mathcal{D}_{\text{test}}, \boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{y} \sim p(\boldsymbol{y}_{\text{test}} \mid \mathcal{T}_{\text{test}}), \boldsymbol{x} \sim p(\boldsymbol{x}_{\text{test}} \mid \mathcal{T}_{\text{test}})}[-\log p_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{f_\theta}(\mathcal{D}_{\text{test}}))] \tag{1}$$

2.2 Continuous Meta-Learning via Online Changepoint Analysis

One drawback of the traditional meta-learning setting is the assumption that the tasks $\mathcal{T}_i$ which generated our dataset $\mathcal{D}$ are known, which is not a realistic assumption in settings where task segmentation may be unavailable. To address this issue, we leverage a framework known as Meta-Learning via Online Changepoint Analysis (MOCA) (13) which relaxes the assumption that tasks are known and instead assumes we are given access to a dataset $\mathcal{D}_{\text{train}} = \{(\boldsymbol{x}, \boldsymbol{y})_{1:N}\}$ where the data is drawn according to an unobserved task $\mathcal{T}_z$ and new tasks are sampled with a fixed probability λ, which we call the hazard rate. This can also be called the *continuous* meta-learning problem, as it can be viewed as meta-learning over continuously incoming data points in an online fashion.

MOCA addresses the uncertainty over the latent tasks by leveraging Bayesian Online Changepoint Detection (BOCPD) (1), a filtering approach for detecting discrete changes in the underlying generative process of a data stream (i.e. task switches). This is done by maintaining a belief distribution over the *run length* of the current task, which tracks how many prior datapoints were generated under the given task. The run length r_t can only take on a value of 0 (the task has switched) or $r_{t-1}+1$ (the task continues) and we assume it has finite discrete support $\{0, ...T\}$, which allows a very simple update rule for our run length. Formally, MOCA maintains a belief $b_t(r_t)$ which is initialized with $b_1(r_1 = 0) = 1$ and $\boldsymbol{\eta}_0[r = 0]$ according to our prior parameters θ. Then, as our meta-learning observes datapoints $\boldsymbol{x}_t, \boldsymbol{y}_t$, we update our belief distribution as $b_t(r_t \mid \boldsymbol{x}_t, \boldsymbol{y}_t) \propto p_\theta(\boldsymbol{y}_t \mid \boldsymbol{x}_t, \boldsymbol{\eta}_{t-1}[r_t])b_t(r_t)$ which can be normalized by summing over the finite support of $b_t(r_t)$. Thus our updated belief is

$$b_{t+1}(r_{t+1} = k) = \begin{cases} \lambda & \textbf{if } k = 0 \\ (1-\lambda)b_t(r_t \mid \boldsymbol{x}_t, \boldsymbol{y}_t) & \textbf{if } k > 0 \end{cases} \tag{2}$$

MOCA then uses the resulting belief function to allow a predictive model that performs meta-learning on time-series data without explicit task knowledge. Note that this framework also assumes that we can compute task-specific statistics as $\boldsymbol{\eta}_t[r] = g(\boldsymbol{x_t}, \boldsymbol{y_t}, \boldsymbol{\eta}_{t-1}[r-1])$ where g is a differentiable recursive update rule. While recent work has demonstrated it is also feasible to model the run lengths *until* the next changepoint for improved predictive performance (2), we opt to forgo such an approach as it would lead to additional memory overhead from modeling both types of run lengths and because since we are not in a formal online setting the run-lengths from MOCA can be used to retrospectively identify changepoints.

2.3 Imitation Learning

We finally formulate our problem as a Markov control process (MCP) (36) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \boldsymbol{p})$ where $\mathcal{S}, \mathcal{A}$ are state and action spaces, $\boldsymbol{p} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is a stochastic dynamics function. Note that unlike an . During training, our agent is given access to demonstrations $\mathcal{D}_{\text{train}} = \{\sum_{i=1}^{M} \tau_i\}$ of the form

$\tau_i = (\boldsymbol{s}_0, \boldsymbol{a}_0, \ldots, \boldsymbol{s}_T, \boldsymbol{a}_T)$ where each trajectory acts in accordance with some latent skill which is unobserved, i.e. $\tau_i \sim p(\tau \mid z_i), z_i \sim p(z)$. At test time we receive a small amount of additional demonstrations $\mathcal{D}_{\text{test}} = \{\sum_{j=1}^{L} \tau_i\}$ where $L \ll M$ and is finally evaluated on an optimal trajectory τ^*. Ultimately, we want our approach to learns a *primitive* policy mapping skills and states to distributions over actions $\boldsymbol{\pi}_a(\boldsymbol{a}_{t+1} \mid \boldsymbol{s}_t, \boldsymbol{z}_t)$, as well as a *hierarchical* policy over skills $\boldsymbol{\pi}_z(\boldsymbol{z}_t \mid \boldsymbol{h}(\cdot))$ where $\boldsymbol{h}$ is some recurrent function that over previous states. During test time, we assume our low-level policy $\boldsymbol{\pi}_a(\boldsymbol{a}_{t+1} \mid \boldsymbol{s}_t, \boldsymbol{z}_t)$ remains fixed and that our agent must select a sequence of skills $\boldsymbol{z}_1 \ldots \boldsymbol{z}_N$ from an optimal demonstration τ^* to be executed. Due to it's generality, the continuous meta-learning setting is thus applicable to our MCP to accelerate the imitation and identification of novel skills.

3 Approach: Hierarchical Imitation Learning

At a high-level, our suggested approach alternatives between (1) determining the segmentation of skills in our dataset by leveraging MOCA to learn a belief function over run-lengths of latent skill switches and (2) learning the hierarchical and primitive policies over a fixed choice of segmentation. We will first discuss the prerequisite meta-learning algorithms necessary for learning the functions in both steps of our overall variational inference scheme, and then explain each step in detail.

3.1 Bayesian Meta-Learning

3.1.1 ALPaCA

We will first discuss ALPaCA (Adaptive Learning for Probabilistic Connectionist Architectures) (12), which is the core meta-learning algorithm in our procedure that is used both during the expectation step as the underlying predictive model for estimating run-lengths through MOCA, and as the skill-conditioned policy during the maximization step. In standard bayesian linear regression (BLR) (21), we consider a probabilistic regression model $\boldsymbol{y} \sim \mathcal{N}(W\boldsymbol{x}, \boldsymbol{\sigma}^2)$ where we omit the bias term for clarity and σ^2 is typically fixed, about where we also represent the weights of our model with a prior $W \sim p(W)$. ALPaCA can be viewed as extension of to meta-learning, where we perform BLR in a linear feature space such that $\boldsymbol{y} \sim \mathcal{N}(W^T\phi(\boldsymbol{s}_t), \Sigma_\epsilon)$. We refer to our encoder as the "backbone" and the matrix at the last layer as the "head" which is initialized with a matrix-normal prior (3).

Concretely, we sample our initial prior head as $W \sim \mathcal{MN}(W_0, \Sigma_\epsilon, \Lambda_0)$, where $W_0 \in \mathbb{R}^{n_\phi \times n_y}$ is the prior mean, Λ_0 is the prior precision (inverse of covariance) matrix, and Σ_ϵ is the scale matrix. n_ϕ is the dimensionality of the backbone and n_y is the dimensionality of $\boldsymbol{y}$. We initialize each term with a matrix-normal $\mathcal{MN}(\mathbf{0}, I, I)$ where I is the identity matrix and $\mathbf{0}$ is a matrix with all zero entries. Given these these priors, as we observe new datapoints $\boldsymbol{x}_t, \boldsymbol{y}_t$ we can recursively update our posterior as follows:

$$\Lambda_{t+1}^{-1} = \Lambda_t^{-1} - \frac{(\Lambda_t^{-1}\boldsymbol{\phi}(\boldsymbol{x}))(\Lambda_t^{-1}\boldsymbol{\phi}(\boldsymbol{x}))^T}{1 + \boldsymbol{\phi}(\boldsymbol{x})^T\Lambda_t^{-1}\boldsymbol{\phi}(\boldsymbol{x})} \tag{3}$$

$$Q_{t+1} = \boldsymbol{y}_{t+1}\boldsymbol{\phi}(\boldsymbol{x})_{t+1}^T + Q_t \tag{4}$$

Where $Q_0 = \Lambda_0^{-1}W_0$. Finally, for inference we can reformulate our matrix-normal distribution as a multivariate normal (MVN) distribution. For example, our prior formulated as an MVN is:

$$W \sim \mathcal{N}(\text{vec}(W_0), \Sigma_\epsilon \otimes \Lambda_0^{-1})$$

Where $\otimes$ is the kronecker product and vec is the standard vectorization operation on a matrix to a column vector. Thus, our final posterior predictive is

$$\begin{aligned}
\boldsymbol{\mu}(\boldsymbol{x}_{t+1}) &= (\Lambda_t^{-1}Q_t)^T\boldsymbol{\phi}(\boldsymbol{x})_{t+1} \\
\Sigma(\boldsymbol{x}_{t+1}) &= (1 + \boldsymbol{\phi}(\boldsymbol{x})^T\Lambda_t^{-1}\boldsymbol{\phi}(\boldsymbol{x}))\Sigma_\epsilon \\
\hat{\boldsymbol{y}} &\sim \mathcal{N}(\boldsymbol{\mu}(\boldsymbol{x}_{t+1}), \Sigma(\boldsymbol{x}_{t+1}))
\end{aligned}$$

Following out earlier notation the posterior task-specific statistics for ALPaCA are $\boldsymbol{\eta} = \{Q_t, \Lambda_t^{-1}\}$, and the meta-learned priors are $\boldsymbol{\theta} = \{W_0, \Lambda_0, \boldsymbol{w}, \Sigma_\epsilon\}$ where $\boldsymbol{w}$ are the weights of the backbone. Note the slight abuse of notation here since the prior parameters for our matrix normal are different

at the end of training because we back-propagate through time. Also note that during test-time our neural network weights are fixed and we thus only have to do linear algebraic updates to adapt $\boldsymbol{\eta}$ due to conjugacy. ALPaCA can be thought of as a method that learns the mean and kernel function for a Gaussian Process from offline data (26).

3.1.2 PCOC

We will now discuss PCOC (probabilistic clustering for online classification) (13), which is an extension of ALPaCA to the classification setting. For PCOC, we use the backbone to send our input data to an embedding space over which we perform we estimate the density for each class by conditioning on the embedding means and a Dirichlet prior to impose a Categorical-Gaussian generative model in the embedding space. More concretely, our priors are

$$p_1, ..., p_{n_y} \sim Dir(\alpha_0) \tag{5}$$
$$y_t \sim Cat(p_1, ..., p_{n_y}) \tag{6}$$
$$\boldsymbol{\phi}_t \mid y_t \sim \mathcal{N}(\bar{\boldsymbol{\phi}}[y_t], \Sigma[y_t]) \tag{7}$$
$$\bar{\boldsymbol{\phi}}[y_t] \sim \mathcal{N}(\boldsymbol{\mu}_0[y_t], \Lambda_0^{-1}[y_t]) \tag{8}$$

Where $\boldsymbol{\mu}_0[y_t] \in \mathbb{R}^{n_\phi}$ and $\Lambda_0^{-1}[y_t] \in \mathbb{R}^{n_\phi \times n_\phi}$ are the prior mean and covariance for $\bar{\boldsymbol{\phi}}[y_t]$, which is the embedding mean for each class. The bracket notation is to index the statistics for each class which are separate and thus we have $\boldsymbol{\mu}_0[k], \Sigma[y_t]$ and $\Lambda_0^{-1}[k]$ for $k = 1, ..., n_y$ and α_0 are parameters for our Dirichlet prior (assumed to be all ones) which we index by class similarly. Thus given new datapoints $\boldsymbol{x}_t, \boldsymbol{y}_t$ and defining $\boldsymbol{q}_t[k] = \Lambda_t^{-1}[k]\boldsymbol{\mu}_t[k]$ we can again recursively update our posterior:

$$\boldsymbol{\alpha}_{t+1}[y_{t+1}] = \boldsymbol{\alpha}_t[y_{t+1}] + 1 \tag{9}$$
$$\boldsymbol{q}_{t+1}[y_{t+1}] = \boldsymbol{q}_{t+1}[y_{t+1}] + \Sigma[y_t]^{-1}\boldsymbol{\phi}(x_{t+1}) \tag{10}$$
$$\Lambda_{t+1}^{-1}[y_{t+1}] = \Lambda_t^{-1}[y_{t+1}] + \Sigma[y_t]^{-1} \tag{11}$$

Defining $\boldsymbol{\eta}_{t+1} = \{\boldsymbol{\alpha}_{t+1}[1:n_y], \boldsymbol{q}_{t+1}[1:n_y], \Lambda_{t+1}^{-1}[1:n_y]\}$ the final posterior predictive is defined as:

$$p(y \mid \boldsymbol{\eta}_{t+1}) = \frac{\boldsymbol{\alpha}_{t+1}[y]}{\sum_{y'=1}^{n_y} \boldsymbol{\alpha}_{t+1}[y']}$$
$$p(y, \bar{\boldsymbol{\phi}}[y]) \mid \boldsymbol{\eta}_{t+1}) = p(y \mid \boldsymbol{\eta}_{t+1})\mathcal{N}(\Lambda_{t+1}^{-1}[y_{t+1}]\boldsymbol{q}_{t+1}[y_{t+1}], \Lambda_{t+1}^{-1}[y_{t+1}] + \Sigma[y_t]^{-1})$$
$$\hat{\boldsymbol{y}} \sim \frac{p(y, \bar{\boldsymbol{\phi}}[y]) \mid \boldsymbol{\eta}_{t+1})}{\sum_{y'=1}^{n_y} p(y, \bar{\boldsymbol{\phi}}[y]) \mid \boldsymbol{\eta}_{t+1})}$$

Where task specific parameters are $\boldsymbol{\eta}_{t+1}$ and our meta-learned prior is $\boldsymbol{\theta} = \{\boldsymbol{\mu}_0[1:n_y], \Lambda_0[1:n_y], \boldsymbol{w}, \Sigma_\epsilon[1:n_y]\}$. Note that like in ALPaCA we can adapt $\boldsymbol{\eta}$ without gradient updates during test time due to conjugacy. PCOC can be thought of as the bayesian version of prototypical networks (29)

3.2 Expectation over run-lengths

First, we wish to learn a belief function over the run-length of the latent skill switches such that we can leverage this distribution to compute boundaries for our segmented trajectories in the expectation step. For our choice of the underlying predictive model we leverage ALPaCA as previously described, which is desirable because it allows use of a posterior predictive density over actions, and because during meta-test time the task specific statistics $\boldsymbol{\eta}$ can be computed recursively, satisfying the requirements for MOCA. Note that as in BOCDP, MOCA maintains a belief over all possible run-lengths r_t where b_t is the belief distribution before observing data at timestep t, with discrete support over $r_t \in \{0, ..., t-1\}$. Practically, this translates into maintaining statistics η for each possible value of r_t, so we denote $\boldsymbol{\eta}[r]$ as the posterior statistics after adapting to the past r datapoints and we also set a max run-length horizon of R.

More concretely, we wish to learn an updated belief function $b_t(r_t \mid \boldsymbol{s}_t, \boldsymbol{a}_t)$ after observing $\boldsymbol{s}_t, \boldsymbol{a}_t$. While in principle we could condition our belief function on both $\boldsymbol{s}$ and $\boldsymbol{a}$ separately, this requires a maintaining a generative model of $p(\boldsymbol{s})$ which ALPaCA does not provide since we are in a regression

Algorithm 1 MASEIL

Require: Training data $\boldsymbol{s}_{1:M}, \boldsymbol{a}_{1:M}$, number of training iterations M, sub-trajectory length T, Expectation ALPaCA model $p_{\boldsymbol{\theta}}(\hat{\boldsymbol{a}} \mid \boldsymbol{s}, \boldsymbol{\eta})$, ELBO weight δ, threshold ζ, PCOC LSTM m, Multi-head primitive ALPaCA $\boldsymbol{\pi}_a(\boldsymbol{a} \mid \boldsymbol{s}, \boldsymbol{z}, \eta_{\boldsymbol{a}})$, hierarchical PCOC $\boldsymbol{\pi}_z(\boldsymbol{z} \mid \boldsymbol{h}, \eta_{\boldsymbol{z}})$

Initialize $\text{NELBO}_\delta = 0$

for $i = 1$ to N **do**

Sample continuous training batch $\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T}$ from full training data.

▷ Expectation Step

Initialize run length belief $b_1(r_1 = 0) = 1$, $\eta_0[r = 0]$ according to $\boldsymbol{\theta}$

for $t = 1$ to T **do**

Observe $\boldsymbol{s}_t, \boldsymbol{a}_t$

Incure NLL loss $l_t = -\log p_\theta(\boldsymbol{a}_t \mid \boldsymbol{s}_t, \boldsymbol{\eta}_t)$

Update $\boldsymbol{\eta}_t[r_t]$ for all r_t according to equations $(3), (4)$

Compute $b_t(r_t \mid \boldsymbol{a}_t, \boldsymbol{s}_t)$ according to equations $(12, 13, 14)$

Compute updated belief b_{t+1} according to equation 15

end for

Compute $\nabla_{\boldsymbol{\theta}} \delta \cdot \text{NELBO}_\delta + \sum_{t=0}^{T} l_t$ and take gradient descent step to update $\boldsymbol{\theta}$

Segment $\tau^* = \tau_1, ..., \tau_h$ according to ζ

for $\tau_j \in \tau^*$ **do**

Encode $\boldsymbol{h_j} = m(\tau_j)$

Sample $z \sim \boldsymbol{\pi}_z(\boldsymbol{z} \mid \boldsymbol{h_j})$

Update $\boldsymbol{\eta_z}$ according to equations $(9 - 11)$

Compute $\text{NELBO}_{\text{KL}j} = (\log \boldsymbol{p}(z) - \log \boldsymbol{\pi}_z(\boldsymbol{z} \mid \boldsymbol{m}(\tau))$

for $t = 1 \in \text{length}(\tau_j)$ **do**

▷ Add to running Rec loss total for current trajectory

Compute $\text{NELBO}_{\text{Rec}j} -= \log \boldsymbol{\pi}_a(t \mid \boldsymbol{s}_t, \boldsymbol{z}_i)$

end for

Update $\boldsymbol{\eta_a}$ according to equations $(3), (4)$

end for

$\text{NELBO}_\delta = \sum_{j=0}^{\text{length}(\tau^*)} \text{NELBO}_{\text{Rec}j} + \text{NELBO}_{\text{KL}j}$

Compute $\nabla_{\boldsymbol{\pi}_a + \boldsymbol{\pi}_z} \text{NELBO}_\delta$

end for

setting. However, we still note our updated belief function with the notation above. Concretely, our update rule is

$$b_t(r_t \mid \boldsymbol{s}_t, \boldsymbol{a}_t) = p(r_t \mid \boldsymbol{s}_{1:t}, \boldsymbol{a}_{1:t}) \tag{12}$$

$$= Z^{-1} p(r_t, \boldsymbol{a}_t \mid \boldsymbol{s}_{1:t}, \boldsymbol{a}_{1:t}) \tag{13}$$

$$= Z^{-1} p(\boldsymbol{a}_t \mid \boldsymbol{s}_t, \boldsymbol{\eta}_{t-1}[r_t]) b_t(r_t) \tag{14}$$

Where Z is a normalizing constant which can be computed exactly over the finite support of b_t. We then push our belief forward in time through a hazard rate λ which can be seen as our prior probability of switching skills at any timestep. This yields the update

$$b_{t+1}(r_{t+1} = k) = \begin{cases} \lambda \textbf{ if } k = 0 \\ (1 - \lambda) b_t(r_t = k - 1 \mid \boldsymbol{s}_t, \boldsymbol{a}_t) \textbf{ if } k > 0 \end{cases} \tag{15}$$

After computing our belief function, we must then segment the training batch into trajectories $\tau_1, ..., \tau_H$ according to some threshold of the MAP run-length ζ.

3.3 Maximization given run-lengths

Given segmented trajectories $\{\sum_{i=1}^{H} \tau_i\}$, we are now left with a fairly straightforward variational inference problem. we take a Bayesian approach to learning the primitive policies which assumes a skill conditioned set of Gaussian distributions $\boldsymbol{\pi}_a(\boldsymbol{a} \mid \boldsymbol{s}, \boldsymbol{z} = i) = \mathcal{N}(\boldsymbol{\mu}_i(\boldsymbol{s}), \Sigma_\epsilon[i])$, and we thus utilize a multi-head version of ALPaCA that maintains separate statistics $\boldsymbol{\eta}_i$ for each skill $\boldsymbol{z}_i$ such that $\boldsymbol{a}_t \mid \boldsymbol{z}_i \sim \mathcal{N}(K^T[i]\phi(\boldsymbol{s}_t), \Sigma_\epsilon[i])$ (19). Note that we use a separate backbone for the multi-head

model. While in principle we could use some weight sharing scheme between the backbone of the ALPaCA model from the expectation step and the multi-head ALPaCA model, we leave this for future work.

For our hierarchical policy $\boldsymbol{\pi}_z(\boldsymbol{z} \mid \boldsymbol{h})$ we leverage PCOC as described earlier. We transform our trajectories to get our histories as $\boldsymbol{h}_i = \boldsymbol{m}(\tau_i)$ where $\boldsymbol{m}(\cdot)$ is an LSTM (14). Both policies optimize a negative log likelihood loss. The combined optimization procedure simply reduces to maximizing the evidence lower bound (ELBO):

$$\log p(\boldsymbol{a}_t \mid \boldsymbol{s}_t) \geq \mathbb{E}_{\tau \sim \mathcal{D}, \boldsymbol{z} \sim \boldsymbol{\pi}_z(\boldsymbol{z} \mid \boldsymbol{h}) \boldsymbol{a} \sim \boldsymbol{\pi}_a(\boldsymbol{a} \mid \boldsymbol{s}, \boldsymbol{z})}[\log \boldsymbol{\pi}_a(\boldsymbol{a} \mid \boldsymbol{s}, \boldsymbol{z}) + (\log \boldsymbol{p}(z) - \log \boldsymbol{\pi}_z(\boldsymbol{z} \mid \boldsymbol{m}(\tau)))]$$

Where $\boldsymbol{p}(z)$ is a categorical with uniform probabilities across all classes. While other methods have optimized the above objective to solve the hierarchical imitation learning problem (11; 23). We note that these approaches differ from ours in that they randomly sampled trajectories of a fixed length from the dataset, which is an unrealistic assumption as different skills likely have different lengths. They also assume a Gaussian distribution for their posterior over skills which is undesirable as it makes it much harder to interpret the choice of $\boldsymbol{z}$ as they are effectively learning an infinite number of skills corresponding to randomly sampled continuous trajectories. Our approach learns a discrete encoder allowing for clear interpretablility between skills and avoids the use of the reparamterization trick when computing the expectation over skills. Note that throughout this procedure we assumed that there exist a fixed number of skills N which is a hyperparameter.

Note that in practice we optimize the Negative ELBO since we are minimizing with gradient descent. We also need some sort of way to send feedback to the expectation step about the quality of segmented trajectories and we do so by down-weighting the ELBO computed at each iteration of the maximization step with some factor δ and adding it to the loss in MASEIL's expectation step.

3.4 Evaluation

During test-time once we are given a new dataset of state action transitions $(s_{1:P}, a_{1:P})$ where P is the number of timesteps in the expert test time demo, we simply run our expectation step from algorithm 1 once again but without the need to backprop. We then compute the segmented trajectories $\tau_1, ..., \tau_L$, and use the learned primitive and hierarchical policies to imitate the resulting trajectories. We rollout our primitive policies for r_{max} timesteps each where r_{max} is the length of the longest segmented trajectory.

4 Experiments

4.1 Quantatitive Results

For the ALPaCA models we use 3 layers MLPs with architecture with hidden unit sizes $[128, 128, 64]$ for the backbones and for the PCOC LSTM we will use 128 hidden units with two recurrent layers. For all models we used $n_\phi = 32$

For our preliminary experiments, we benchmark the Franka Kitchen simulated environment (9), where the task is to use a 7-DoF robotic arm to manipulate different objects in a kitchen environment in a specified order, such as opening a microwave door, or opening a cabinet. The agent is given access to the joint positions and velocities of the arm as well as a goal state corresponding to the object to be manipulated. This is a well suited test environment as there is a clear hierarchical structure which arises from the distinct tasks the robot must accomplish which can be tackled by learning skill-conditioned policies. This environment is also suitable for imitation learning as the original implementation is a sparse reward setting, where the agent would receive a reward of 1 upon completing a task and 0 otherwise. We train on 100,000 transitions which include all the data collected for all tasks and at test time fine-tune on 5,000 transitions which demonstrate the desired Task order we wish to imitate. The maximum score in the metric below is 4

4.1.1 Behavioral Cloning (BC)

For our first baseline, we use behavioral cloning (27; 25), which directly learns a distribution over actions conditional on states $\pi_\theta(a \mid s)$. As expected, naive behavioral cloning is unable to fully imitate the desired skills for any task order, and does not successfully imitate even 1 skill for the

last task order. This shows there is room for improvement, which will likely come in the form of explicitly reasoning and the latent skill-conditioned behavior present within these demonstrations.

4.1.2 Skill-Prior RL (SPIRL)

This method trains a Beta-VAE over the same dataset optimizing the ELBO objective as in our maximization step, but the trajectories they encode are randomly sampled continuous trajectories of a fixed length H, and they use a normal distribution as the posterior for the decoder (23). The objective function optimized is formally

$$\log p(a_i) \geq \mathbb{E}_q[\log p(a_i \mid z) - \beta(\log q(z \mid a_i) - \log p(z))]$$

Where $p(z)$ is a standard normal distribution

4.1.3 Few-Shot Imitation Learning with Skill Transition Models (FIST)

This method (11) optimizes the same objective as SPIRL, but with an added skill transition decoder which is trained with an augmented objective to the ELBO defined as

$$L(\psi) = \mathbb{E}_{\tau \sim \mathcal{D}}[\log(q_\phi(z \mid \tau) - q_\phi(z \mid s_t, s_{t+H-1}))]$$

Where during test-time they select the s_{t+H-1} by finding the state in the expert demo that minimizes the InfoNCE (35) objective with the current state s_t. This can be seen as learning an implicit energy based model (37)

Task Order	**BC**	**SPiRL**	**FIST**	**Ours**
Microwave, Bottom Burner, Light Switch, Slide Cabinet	2.2 ± 0.28	2.3 ± 0.5	2.3 ± 0.16	2.2 ± 0.54
Microwave, Kettle, Slide Cabinet, Hinge Cabinet	1.3 ± 0.47	0 ± 0	$\mathbf{3.5} \pm 0.3$	2.4 ± 0.1
Microwave, Kettle, Top Burner, Light Switch	0.0 ± 0.0	2.1 ± 0.48	$\mathbf{3.6} \pm 0.16$	2.8 ± 0.14

Table 1: Reward on multiple test-time expert demonstrations. Max score is 4. **Higher** is better

While we outperform BC and match SPIRL we have not outperformed FIST which is the current state of the art. We hypothesize this is because of bottlenecks in the implementation of the expectation over run-lengths. We also note that while in principle we should be able to learn both δ and ζ, we did not have time to implement the necessary procedures to backprop through these weights and instead fine-tuned them through HP tuning to values of 0.1 and 5 respectively. We assume a maximize run-length of 100 steps, and a subtrajectory length of size $T = 250$. However we note our method gives explicitly calibrated bayesian estimates, satisfying our earlier criteria for safety. We also note that while our method takes longer to train than fist (around 12 hours until convergence vs 4 for FIST), our test time inference is much faster and does not require gradient updates so it can be efficiently done on CPUs which is more important for deployment in robotics. The github link for my code can be found here: `https://github.com/ahmeda14960/CS_236_final_project`

Model	**Test time inference (lower is better)**
Our Approach	≈ 3 **minutes**
FIST	≈ 15 minutes

Table 2: Test time inference on RTX 2080 Ti

4.2 Qualitative results

To add some qualitative analysis to our project, we visualized the trained run-length prediction meta-learning algorithm from the expectation step of MASEIL over the rollout of a complete expert demo. In Figure 2, you can see this run-length distribution visualization where the x-axis is the time of the rollout and the y-axis corresponds to the run-length. The darkness of the grey shows us how much probability mass the model is putting on a specific run-length value, and the sharp drops show times when the model thinks a switch as likely occurred. The red dark lines show when the ground truth expert transitions happen, which are shown in figure one as the end of the skills where the robot

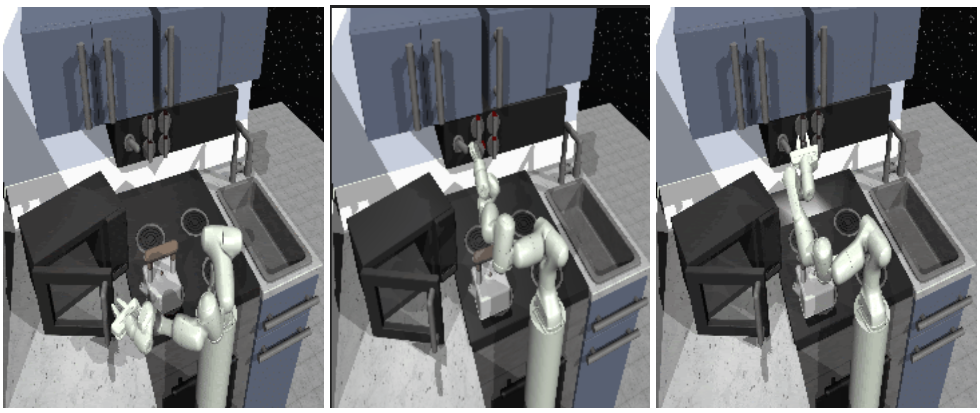

Figure 1: Expert trajectory snapshots during ground truth changepoints. These show the changepoints for the microwave on the **left**, the bottomknob on the **middle** and the light switch on the **right**

arm opens the microwave door, turns the bottomknob and the lightswitch respectively. Each of these show the three possible cases of our run-length prediction, as our model almost perfectly computed the skill switch for the microwave, but was too early for predicting the switch for the bottomknob and too late for the lightswitch. Our hypothesis is that because the microwave is relatively far away from other objects in the state-space it was easier for a skill switch to be detected. However, since the bottomknob and lightswitch are much closer this leads to a much harder problem. You can see in the middle plot of figure 2 that when the run-length predicted an early switch it still maintainted some mass on run-length continuing as the dark line in between the more solidly grey rectangle underneath where most of the mass was placed. This likely contributed to the late skill switch for the lightswitch as this confusion lead to increased uncertainty about whether or not to switch.

5 Theoretical Analysis of MASEIL

We will now analyze some theoretical properties of MASEIL to better understand why it is desirable for the hierarchical imitation learning problem. We will highlight the ability of MASEIL to recover the optimal run-lengths for any trajectories where such a run-length can be learned. This is an informal outline but will hopefully give some intuition as to our algorithm's performance.

We fill first note that the standard meta-learning objective in equation (1) can be posed as a variational inference problem in and of itself, of the form

$$\max \mathbb{E}_{q(\psi)}[\log p(\mathcal{D}_{\text{train}} \mid \psi) - KL(q(\psi)||p(\psi)] \tag{16}$$

Where ψ is the latent ground truth task specific parameter that we do not observe and attempt to infer. While this connection has been explore in the literature (6; 7), we explicitly note it here to connect our over procedure MASEIL as an implicit meta-learning algorithm.

We now also note a universality common in meta learning that (4) for any function g which can be learned, there exists a meta-learning algorithm which leverages a neural network with optimal prior parameters $\boldsymbol{\theta}^*$ such that $g(x) = f(x, \boldsymbol{\theta}^*) - \alpha \nabla l(\boldsymbol{\theta}^*, \mathcal{D}_{adapt})$ with one-step of gradient descent where $\mathcal{D}_{adapt}$ is some finite number of adaptation input-output pairs, i.e. $\mathcal{D}_{adapt} = \{(\boldsymbol{x}, \boldsymbol{y})_{1:k}\}$. Note that the loss function here cannot be any loss function, but the paper proves the above holds for the negative log likelihood which is the only loss function we consider in our work.

Now, we note that in general a geometric random variable is defined with parameter $Geom(\lambda)$ is the number of trials until a success with expected number of trials $\frac{1}{\lambda}$. Note that the prior hazard term in MOCA can simply be seen as such a geometric random variable, and thus MASEIL can be seen as a meta-learning algorithm that learns a prior which can quickly adapt to segmented trajectories according to some expected run-length. Now assuming that there exists a learnable function $h(\tau) = \frac{1}{r^*}$ which leverages the optimal run-length r^* for an trajectory τ which is unsegmented, which equivalently defines a geometric random variable $Geom(h(\tau))$. Now note that our learned function $b_t(r_t \mid \boldsymbol{s}_t, \boldsymbol{a}_t)$ can be seen as a Geometric random variable where the hazard rate $\lambda_{\textbf{prior}}$ can be adapted

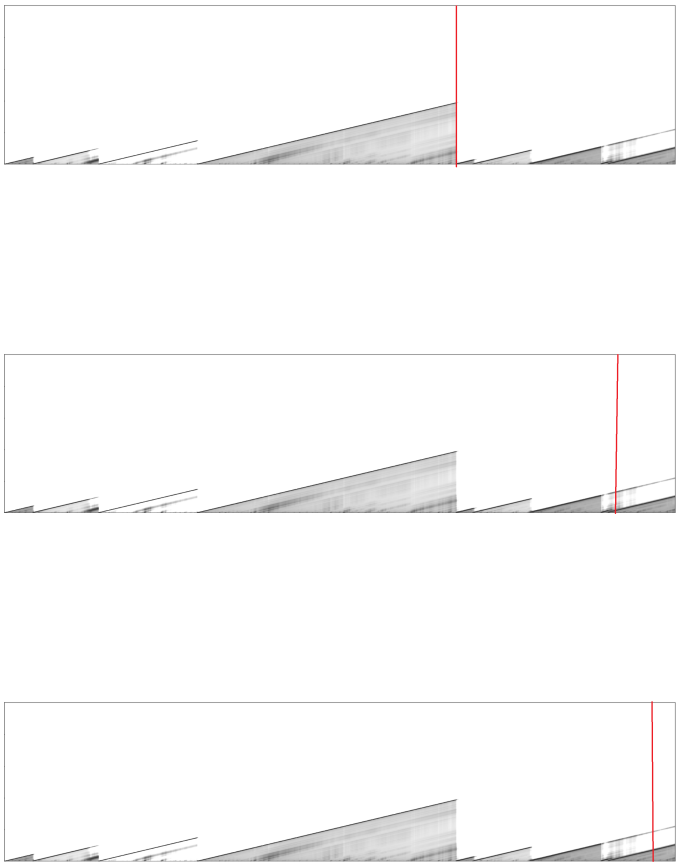

Figure 2: Ground truth task switch over run-lengths for **microwave (top)**, **bottomknob (middle)** and **lightswitch (bottom)**

through gradient descent via the ALPaCA meta-learning algorithm. Now we can see that there exists optimal parameters $\boldsymbol{\theta}^*$ for MASEIL such that

$$h(\tau) = b_t(r_t \mid \boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{\theta}^*) - \alpha \nabla l(\boldsymbol{\theta}^*, \mathcal{D}_{adapt})$$

For some number of finite adaptation data from the target trajectory $\mathcal{D}_{adapt}$. Assuming we adapt to $h(\tau)$ our posterior Geometric distribution is now defined as

$$\begin{aligned} Geom(\lambda_{\textbf{prior}|\mathcal{D}_{adapt}}) = \\ Geom(b_t(r_t \mid \boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{\theta}^*) - \alpha \nabla l(\boldsymbol{\theta}^*, \mathcal{D}_{adapt})) = \\ Geom(h(\tau)) = \\ Geom(\frac{1}{r^*}) = \\ r^* \end{aligned}$$

This "proof" is quite informal, but hopefully gives some theoretical intuition as to why MASEIL is a good approach and how in the limit, we can eventually learn meta-learning parameters that allow us to quickly adapt to the optimal geometric distribution that describes the run-length of our unsegmented trajectories.

6 Conclusion and Future Work

Ultimately, we see that while MASEIL does not beat the state of the art in terms of performance and incurs high costs of computing during training, it still benefits from fast test-time inference, explicit Bayesian calibrated uncertainties and some theoretically nice properties that make it suitable for the hierarchical imitation learning setting in it's ability to recover optimal run-lengths within a single step of gradient descent. As for future work, instead of assuming a discrete encoding of skills we could use the Gumbel-Softmax distribution for training our encoder (15; 20), which is a continuous relaxation of the categorical distribution. In principle, MOCA works with any meta-learning algorithm that admits a recursive update rule so we could use LSTM models in place of both ALPaCA and PCOC, which is worth investigating as a baseline. We can also take next steps towards learning the number of optimal skills N by exploring using meta-learning combined with a Dirichlet process.

References

[1] Ryan Prescott Adams and David JC MacKay. Bayesian online changepoint detection. *arXiv:0710.3742*, 2007.

[2] Diego Agudelo-España, Sebastián Gómez-González, Stefan Bauer, Bernhard Schölkopf, and Jan Peters. Bayesian online prediction of change points. In *UAI*, 2020.

[3] A. Philip Dawid. Some matrix-variate distribution theory: Notational considerations and a bayesian application. *Biometrika*, 68:265–274, 1981.

[4] Chelsea Finn and Sergey Levine. Meta-learning and universality: Deep representations and gradient descent can approximate any learning algorithm, 2018.

[5] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *International Conference on Machine Learning (ICML)*, 2017.

[6] Chelsea Finn, Kelvin Xu, and Sergey Levine. Probabilistic model-agnostic meta-learning, 2019.

[7] Jonathan Gordon, John Bronskill, Matthias Bauer, Sebastian Nowozin, and Richard E. Turner. Meta-learning probabilistic inference for prediction, 2019.

[8] Erin Grant, Chelsea Finn, Sergey Levine, Trevor Darrell, and Thomas Griffiths. Recasting gradient-based meta-learning as hierarchical Bayes. *International Conference on Learning Representations (ICLR)*, 2018.

[9] Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning, 2019.

[10] Dylan Hadfield-Menell, Stuart J Russell, Pieter Abbeel, and Anca Dragan. Cooperative inverse reinforcement learning, 2016.

[11] Kourosh Hakhamaneshi, Ruihan Zhao, Albert Zhan, Pieter Abbeel, and Michael Laskin. Hierarchical few-shot imitation with skill transition models, 2021.

[12] James Harrison, Apoorva Sharma, and Marco Pavone. Meta-learning priors for efficient online Bayesian regression. *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2018.

[13] James Harrison, Apoorva Sharma, Chelsea Finn, and Marco Pavone. Continuous meta-learning without tasks. *Neural Information Processing Systems (NeurIPS)*, 2020.

[14] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9: 1735–80, 12 1997. doi: 10.1162/neco.1997.9.8.1735.

[15] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax, 2017.

[16] Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. *International Conference on Learning Representations (ICLR)*, 2014.

[17] Thomas Kipf, Yujia Li, Hanjun Dai, Vinicius Zambaldi, Alvaro Sanchez-Gonzalez, Edward Grefenstette, Pushmeet Kohli, and Peter Battaglia. Compile: Compositional imitation learning and execution, 2019.

[18] Hoang M. Le, Nan Jiang, Alekh Agarwal, Miroslav Dudík, Yisong Yue, and Hal Daumé III au2. Hierarchical imitation and reinforcement learning, 2018.

[19] T. Lew, A. Sharma, J. Harrison, A. Bylard, and M. Pavone. Safe active dynamics learning and control: A sequential exploration-exploitation framework. March 2021. URL `https://arxiv.org/pdf/2008.11700.pdf`. Submitted.

[20] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables, 2017.

[21] Kevin P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022. URL `probml.ai`.

[22] Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J. Andrew Bagnell, Pieter Abbeel, and Jan Peters. An algorithmic perspective on imitation learning. *Foundations and Trends in Robotics*, 7(1-2):1–179, 2018. ISSN 1935-8261. doi: 10.1561/2300000053. URL `http://dx.doi.org/10.1561/2300000053`.

[23] Karl Pertsch, Youngwoon Lee, and Joseph J. Lim. Accelerating reinforcement learning with learned skill priors, 2020.

[24] Karl Pertsch, Youngwoon Lee, Yue Wu, and Joseph J. Lim. Demonstration-guided reinforcement learning with learned skills, 2021.

[25] Dean A. Pomerleau. ALVINN: an autonomous land vehicle in a neural network. In David S. Touretzky, editor, *Advances in Neural Information Processing Systems 1*, pages 305–313. San Francisco, CA: Morgan Kaufmann, 1989.

[26] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005. ISBN 026218253X.

[27] Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning, 2011.

[28] Stefan Schaal, Jan Peters, Jun Nakanishi, and Auke Ijspeert. Learning movement primitives. *International Symposium on Robotics Research (ISRR)*, 2005.

[29] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. *Neural Information Processing Systems (NeurIPS)*, 2017.

[30] Wen Sun, Arun Venkatraman, Geoffrey J. Gordon, Byron Boots, and J. Andrew Bagnell. Deeply aggrevated: Differentiable imitation learning for sequential prediction. *CoRR*, abs/1703.01030, 2017. URL `http://arxiv.org/abs/1703.01030`.

[31] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL `http://incompleteideas.net/book/the-book-2nd.html`.

[32] Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1): 181–211, 1999. ISSN 0004-3702. doi: https://doi.org/10.1016/S0004-3702(99)00052-1. URL `https://www.sciencedirect.com/science/article/pii/S0004370299000521`.

[33] Sebastian Thrun. Is learning the n-th thing any easier than learning the first? *Neural Information Processing Systems (NeurIPS)*, 1996.

[34] Sebastian Thrun and Lorien Pratt. *Learning to Learn: Introduction and Overview*, pages 3–17. Springer US, Boston, MA, 1998. ISBN 978-1-4615-5529-2. doi: 10.1007/978-1-4615-5529-2_1. URL `https://doi.org/10.1007/978-1-4615-5529-2_1`.

[35] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding, 2019.

[36] Chelsea C. White. A markov quality control process subject to partial observation. *Management Science*, 1977.

[37] Roland S. Zimmermann, Yash Sharma, Steffen Schneider, Matthias Bethge, and Wieland Brendel. Contrastive learning inverts the data generating process, 2021.