Hierarchical Expert Networks for Meta-Learning

Heinke Hihn 1  Daniel A. Braun 1

Abstract

The goal of meta-learning is to train a model on a variety of learning tasks, such that it can adapt to new problems within only a few iterations. Here we propose a principled information-theoretic model that optimally partitions the underlying problem space such that specialized expert decision-makers solve the resulting sub-problems. To drive this specialization we impose the same kind of information processing constraints both on the partitioning and the expert decision-makers. We argue that this specialization leads to efficient adaptation to new tasks. To demonstrate the generality of our approach we evaluate three meta-learning domains: image classification, regression, and reinforcement learning.

1. Introduction

Recent machine learning research has shown impressive results on incredibly diverse tasks from problem classes such as pattern recognition, reinforcement learning, and generative model learning (Devlin et al., 2018; Mnih et al., 2015; Schmidhuber, 2015). These success stories typically have two computational luxuries in common: a large database with thousands or even millions of training samples and a long and extensive training period. Applying these pre-trained models to new tasks naively usually leads to poor performance, as with each new incoming batch of data, expensive and slow re-learning. In contrast to this, humans can learn from few examples and excel at adapting quickly (Jankowski et al., 2011), for example in motor tasks (Braun et al., 2009) or at learning new visual concepts (Lake et al., 2015).

Sample-efficient adaptation to new tasks can be a form of meta-learning or “learning to learn” (Thrun & Pratt, 2012; Schmidhuber et al., 1997; Caruana, 1997) and is an ongoing and active field of research—see e.g. (Koch et al., 2015).

Sample-efficient adaptation to new tasks can be a form of meta-learning or “learning to learn” (Thrun & Pratt, 2012; Schmidhuber et al., 1997; Caruana, 1997) and is an ongoing and active field of research—see e.g. (Koch et al., 2015).

The outline of this paper is as follows: first, we introduce bounded rationality and meta-learning, next we introduce our novel approach and derive applications to classification, regression, and reinforcement learning. We evaluate our method against current meta-learning algorithms and perform additional ablation studies. Finally, we conclude.

2. Information-Processing Constraints in Hierarchical Learning Systems

An important concept in decision making is the notion of utility (Von Neumann & Morgenstern, 2007), where an agent picks an action $a^*_s \in A$ such that it maximizes their utility in some context $s \in S$, i.e., $a^*_s = \arg \max_a U(x, a)$, where the utility is a function $U(x, a)$ and the states distribution $p(s)$ is known and fixed. Trying to solve this optimization problem naively leads to an exhaustive search over all possible $(a, x)$ pairs, which is in general a pro-
Hierarchical Expert Networks for Meta-Learning

**Figure 1. Left:** The selector assigns the new input encoding to one of the three experts $\theta_0$, $\theta_1$, or $\theta_2$, depending on the similarity of the input to previous inputs seen by the experts. **Right:** Our proposed method consists of three main stages. First, we feed the training dataset $D_{\text{train}}$ through a convolutional autoencoder to find a latent representation $z(d_i)$ for each $d_i \in D_{\text{train}}$, which we get by flattening the preceding convolutional layer (“flattening layer”). We apply a pooling function to the resulting set of image embeddings which serves as input to the selection network.

Habitual strategy. Instead of finding an optimal strategy, a bounded-rational decision-maker optimally trades off expected utility and the processing costs required to adapt. In this study we consider the information-theoretic free-energy principle (Ortega & Braun, 2013) of bounded rationality, where an upper bound on the Kullback-Leibler divergence $D_{\text{KL}}(p(a|x)||p(a)) = \sum_a p(a|x) \log \frac{p(a|x)}{p(a)}$ between the agent’s prior distribution $p(a)$ and the posterior policy $p(a|x)$ model the decision-maker’s resources, resulting in the following constrained optimization problem:

$$\max_{p(a|x)} \sum_{x,a} p(x)p(a|x)U(x,a)$$

subject to $\mathbb{E}_{p(x)} \left[D_{\text{KL}}(p(a|x)||p(a)) \right] \leq B$. 

This constraint defines a regularization on $p(a|x)$. We can transform this into an unconstrained variational problem by introducing a Lagrange multiplier $\beta \in \mathbb{R}^+$:

$$\max_{p(a|x)} \mathbb{E}_{p(x)} \left[U(x,a) - \frac{1}{\beta} D_{\text{KL}}(p(a|x)||p(a)) \right].$$

For $\beta \rightarrow \infty$ we recover the maximum utility solution and for $\beta \rightarrow 0$ the agent can only act according to the prior. The optimal prior in this case is the marginal $p(a) = \sum_{x \in X} p(x)p(a|x)$ (Ortega & Braun, 2013).

Aggregating bounded-rational agents by a selection policy allows for solving optimization problems that exceed the capabilities of the individual decision-makers (Genewein et al., 2015). To achieve this, the search space is split into partitions such that each partition can be solved by a decision-maker. A two-stage mechanism is introduced: The first stage is an expert selection policy $p(m|x)$ that chooses an expert $m$ given a state $x$ and the second stage chooses an action according to the expert’s posterior policy $p(a|x,m)$. The optimization problem given by Equation (3) can be extended to incorporate a trade-off between computational costs and utility in both stages:

$$\max_{p(a|x,m),p(m|x)} \mathbb{E}_{p(x, a)} \left[U(x, a) - \frac{1}{\beta_1} I(X; M) - \frac{1}{\beta_2} I(A; X | M) \right]$$

where $\beta_1$ is the resource parameter for the expert selection stage and $\beta_2$ for the experts. $I(\cdot; \cdot)$ is the mutual information between the two random variables. To measure a decision-maker’s performance on terms of the utility vs. cost trade off we use the free-energy difference defined as

$$\Delta F_{\text{par}} = \mathbb{E}_{p}[U] - \frac{1}{\beta} D_{\text{KL}}(p\|q),$$

where $p$ is the posterior and $q$ the decision-maker’s prior policy. The marginal distribution $p(a|x)$ defines a mixture-of-experts (Jordan & Jacobs, 1994; Jacobs et al., 1991) policy given by the posterior distributions $p(a|m)$ weighted by the responsibilities determined by the Bayesian posterior $p(m|x)$. Note that $p(x|m)$ is not determined by a given likelihood model, but is the result of the optimization process (4).

3. Meta Learning

We can divide Meta-learning algorithms roughly into Metric Learning (Koch et al., 2015; Vinyals et al., 2016; Snell et al., 2017), Optimizer Learning (Ravi & Larochelle, 2017; Finn et al., 2017; Zintgraf et al., 2018; Rothfuss et al., 2018), and Task Decomposition Models (Lan et al., 2019; Vezhnevets et al., 2019). Our approach depicted in Figure 1 can be seen as a member of the latter group.

3.1. Meta Supervised Learning

In a supervised learning task we are usually interested in a dataset consisting of multiple input and output pairs $D = \{(x_i, y_i)\}_{i=1}^N$ and the learner’s task is to find a function $f(x)$ that maps from input to output, for example through a deep neural network. To do this, we split the dataset into training and test sets and fit a set of parameters $\theta$ on the training data.
and evaluate on test data using the learned function \( f_\theta(x) \).

In meta-learning, we are instead working with meta-datasets \( D \), each containing regular datasets split into training and test sets. We thus have different sets for meta-training, meta-validation, and meta-test, i.e., \( D = \{ D_{\text{train}}, D_{\text{val}}, D_{\text{test}} \} \).

The goal is to train a learning procedure (the meta-learner) that can take as input one of its training sets \( D_{\text{train}} \) and produce a classifier (the learner) that achieves low prediction error on its corresponding test set \( D_{\text{test}} \). The meta-learning is then updated using performance measured based on the learner’s performance on \( D_{\text{val}} \). This may not always be the case, but our work (among others, e.g., (Finn et al., 2017)) follow this paradigm. The rationale being that the meta-learner is trained such that it implicitly optimizes the base learner’s generalization capabilities.

### 3.2. Meta Reinforcement Learning

First, we give a short introduction to reinforcement learning in general, and then we show how this definition can be extended to meta reinforcement learning problems. We model sequential decision problems by defining a Markov Decision Process as a tuple \((S, A, P, r)\), where \( S \) is the set of states, \( A \) the set of actions, \( P : S \times A \times S \rightarrow [0, 1] \) is the transition probability, and \( r : S \times A \rightarrow \mathbb{R} \) is a reward function. The aim is to find the parameter \( \theta \) of a policy \( \pi_\theta \) that maximizes the expected reward:

\[
\theta^* = \arg \max_\theta \mathbb{E}_{\pi_\theta} \left[ \sum_{t=0}^{\infty} r(s_t, a_t) \right].
\]

We define \( r(\tau) = \sum_{t=0}^{\infty} r(s_t, a_t) \) as the cumulative reward of trajectory \( \tau = \{(s_t, a_t)\}_{t=0}^{\infty} \), which is sampled by acting according to the policy \( \pi \), i.e., \( (s, a) \sim \pi(\cdot|s) \), and \( s_{t+1} \sim P(\cdot|s_t, a_t) \). We model learning in this environment as reinforcement learning (Sutton & Barto 2018), where an agent interacts with an environment over some (discrete) time steps \( t \). At each time step \( t \), the agent finds itself in a state \( s_t \) and selects an action \( a_t \) according to the policy \( \pi(a_t|s_t) \). In return, the environment transitions to the next state \( s_{t+1} \) and generates a scalar reward \( r_t \). This process continues until the agent reaches a terminal state after which the process restarts. The goal of the agent is to maximize the expected return from each state \( s_t \), which is typically defined as the infinite horizon discounted sum of the rewards.

In meta-reinforcement learning the problem is given by a set of tasks \( t_i \in T \), where MDP \( t_i = (S, A, P_i, r_i) \) define tasks \( t_i \) as described earlier. We are now interested in finding a set of policies \( \Theta \) that maximizes the average cumulative reward across all tasks in \( T \) and generalizes well to new tasks sampled from a different set of tasks \( T' \).

### 4. Expert Networks for Meta-Learning

Information-theoretic bounded rationality postulates that hierarchies and abstractions emerge when agents have only limited access to computational resources (Genewein et al., 2015), e.g., limited sampling complexity (Hihn et al., 2018) or limited representational power (Hihn et al., 2019). We will show that forming such abstractions equips an agent with the ability of learning the underlying problem structure and thus enables learning of unseen but similar concepts. The method we propose comes out of a unified optimization principle and has the following important features:

1. A regularization mechanism to enforce the emergence of expert policies.
2. A task compression mechanism to extract relevant task information.
3. A selection mechanism to find the most efficient expert for a given task.
4. A regularization mechanism to improve generalization capabilities.

#### 4.1. Latent Task Embeddings

Note that the selector assigns a complete dataset to an expert and that this can be seen as a meta-learning task, as described in (Ravi & Larochelle 2017). To do so, we must find a feature vector \( z(d) \) of the dataset \( d \). This feature vector must fulfill the following desiderata: 1) invariance
against permutation of data points in \( d \), 2) high representation capacity, 3) efficient computability, and 4) constant dimensionality regardless of sample size \( K \). In the following we propose such features for image classification, regression, and reinforcement learning problems.

For image classification we propose to pass the positive images in the dataset through a convolutional autoencoder and use the outputs of the bottleneck layer. Convolutional Autoencoders are generative models that learn to reconstruct their inputs by minimizing the Mean-Squared-Error between the input and the reconstructed image (see e.g., [Chen et al. 2019]). In this way we get similar embeddings \( z(d) \) for similar inputs belonging to the same class. The latent representation is computed for each positive sample in \( d \) and then passed through a pooling function \( h(z(d)) \) to find a single embedding for the complete dataset—see figure 1 for an overview of our proposed model. While in principle functions such as mean, max, and min can be used, we found that max-pooling yields the best results. The authors of [Yao et al. 2019] propose a similar feature set.

For regression we define a similar feature vector. We transform the \( K \) training data points into a feature vector \( z(d) \) by binning the points into \( N \) bins according to their \( x \) value and collecting the \( y \) value. If more than one point falls into the same bin, we average the \( y \) values, thus providing invariance against the order of the data points in \( D_{\text{train}} \). We use this feature vector to assign each data set to an expert according to \( p_\theta(x|z(d)) \).

In the reinforcement learning setting we use a dynamic recurrent neural network (RNN) with LSTM units (Hochreiter & Schmidhuber 1997) to classify trajectories. We feed the RNN with \((s_t, a_t, r_t, t)\) tuples to describe the underlying Markov Decision Process describing the task. At \( t = 0 \) we sample the expert \( x \) according to the learned prior distribution \( p(x) \), as there is no information available so far. The authors of [Lan et al. 2019] propose a similar feature set.

4.2. Specialization in Supervised Learning

Combining multiple experts can often be beneficial (Kuncheva 2004), e.g. in Mixture-of-Experts (Yuksel et al. 2012) or Multiple Classifier Systems (Bellmann et al., 2018). Our method can be interpreted as a member of this family of algorithms.

We define the utility as the negative prediction loss, i.e. \( U(f_x(d), y) = -\mathcal{L}(f_x(d), y) \), where \( f_x(d) \) is the prediction of the expert \( x \) given the input data point \( d \) (in the following we will use the shorthand \( \hat{y}_x \) and \( y \) is the ground truth. We define the cross-entropy loss \( \mathcal{L}(\hat{y}_x, y) = -\sum_i y_i \log \hat{y}_i \) as a performance measure for classification and the mean squared error \( \mathcal{L}(\hat{y}_x, y) = \sum_i (\hat{y}_i - y_i)^2 \) for regression.

**Algorithm 2** Expert Networks for Meta-Reinforcement Learning.

1. **Input:** Environment Distributions \( p(T) \) and \( p(T') \), number of roll-outs \( K \), batch-size \( M \), training episodes \( N \), number of tuples \( L \) used for expert selection
2. **Hyper-parameters:** resource parameters \( \beta_1, \beta_2 \), learning rates \( \eta_x, \eta_y \) for selector and experts
3. **Initialize parameters** \( \theta, \vartheta \)
4. **for** \( i = 1, 2, 3, \ldots, N \) **do**
5. Sample batch of \( M \) environments \( E_i^{\text{train}} \sim p(T) \) and \( E_i^{\text{val}} \sim p(T') \)
6. **for** \( E \in E_i^{\text{train}} \) **do**
7. **for** \( k = 1, 2, 3, \ldots, K \) **do**
8. Collect \( \tau = \{(x_t, a_t, r_t, t)\}_{t=1}^L \) tuples following random expert
9. Select expert \( m \sim p_\theta(m|\tau) \) with RNN policy
10. Collect trajectory \( \tau_k = \{(x_t, a_t, r_t, t)\}_{t=1}^T \) by following \( p_\vartheta(a|x, m) \)
11. **end for**
12. Compute \( F_i = \sum_{t=0}^{T} \gamma^t f(x_{t+1}, m_{t+1}, a_{t+1}) \) for trajectories \( \tau \)
13. where \( f(x, m, a) = r_{\text{train}}(x, a) - \frac{1}{\beta_2} \log p_\vartheta(a|x, m) - \frac{1}{\beta_1} \log p_\theta(x|m) \)
14. **end for**
15. Compute \( \hat{F}_i = \sum_{t=0}^{T} \gamma^t \hat{f}(x_{t+1}, m_{t+1}) \) with \( \hat{f}(x, m) = \mathbb{E}_{p_\theta(a|x, m)}[r_{\text{val}}(x, a) - \frac{1}{\beta_2} \log p_\vartheta(a|x, m)] \)
16. Update selection parameters \( \theta \) with \( \hat{F} \) collected in batch \( i \)
17. Update experts \( m \) with roll-outs collected in batch \( i \)
18. **end for**
19. **end for**
20. **return** \( \theta, \vartheta \)

The objective for expert selection thus is given by

\[
\max_\theta \mathbb{E}_{p_\theta(x|d)} \left[ \hat{f} - \frac{1}{\beta_1} \log \frac{p_\theta(x|d)}{p(x)} \right],
\]

where \( \hat{f} = \mathbb{E}_{p_\theta(\hat{y}_x|x, s)} \left[ -\mathcal{L}(\hat{y}_x, y) - \frac{1}{\beta_2} \log \frac{p_\theta(\hat{y}_x|x, s)}{p(\hat{y}_x|x)} \right] \), i.e. the free energy of the expert and \( \theta, \vartheta \) are the parameters of the selection policy and the expert policies, respectively. Analogously, the action selection objective for each expert \( x \) is defined by

\[
\max_\vartheta \mathbb{E}_{p_\vartheta(\hat{y}_x|x, s)} \left[ -\mathcal{L}(\hat{y}_x, y) - \frac{1}{\beta_2} \log \frac{p_\vartheta(\hat{y}_x|x, s)}{p(\hat{y}_x|x)} \right].
\]

This regularization technique can be seen as a form of output regularization [Pereyra et al., 2017].

4.3. Specialization in Reinforcement Learning

In the following we will derive our algorithm for specialization in hierarchical reinforcement learning agents. Note
that in the reinforcement learning setup the reward function \( r(x, a) \) defines the utility \( U(x, a) \). In maximum entropy RL (see e.g., Haarnoja et al. (2017) (Haarnoja et al. [2017]) the regularization penalizes deviation from a fixed uniformly distributed prior, but in a more general setting we can discourage deviation from an arbitrary prior policy by optimizing for:

\[
\max_p \mathbb{E}_p \left[ \sum_{t=0}^{\infty} \gamma^t \left( r(x_t, a_t) - \frac{1}{\beta} \log \frac{p(a_t | x_t)}{p(a)} \right) \right], \tag{9}
\]

where \( \beta \) trades off between reward and entropy, such that \( \beta \to \infty \) recovers the standard RL value function and \( \beta \to 0 \) recovers the value function under a random policy.

To optimize the objective (9) we define two separate kinds of value function, \( V \) for the selector and one value function \( V_a \) for each expert. Thus, each expert is an actor-critic with separate actor and critic networks. Similarly, the selector has an actor-critic architecture, where the network selects experts and the critic learns to predict the expected free energy of the experts depending on a state variable. The selector’s policy is represented by \( p_\theta \), while each expert’s policy is represented by a distribution \( p_\theta \).

In standard reinforcement learning a common technique to update a parametric policy representation \( p_\omega(a|x) \) with parameters \( \omega \), is to use policy gradients that optimize the cumulative reward \( J(\omega) = \mathbb{E}[p_\omega(a|x)V_\phi(x)] \) expected under the critic’s prediction \( V_\phi(x) \), by following the gradient \( \nabla_\omega J(\omega) = \mathbb{E} \nabla_\omega \log p_\omega(a|x)V_\phi(x) \). This policy gradient formulation (Sutton et al. [2000]) is prone to producing high variance gradients. A common technique to reduce the variance is to formulate the updates using the advantage function instead of the reward (Arulkumaran et al. [2017]). The advantage function \( A(a, x) \) is a measure of how well a certain action \( a \) performs in a state \( x \) compared to the average performance in that state, i.e., \( A(a, x) = Q(x, a) - V(x) \). Here, \( V(x) \) is the value function and is a measure of how well the agent performs in state \( x \), and \( Q(x, a) \) is an estimate of the cumulative reward achieved in state \( x \) when the agent executes action \( a \). Instead of learning the value and the Q function, we can approximate the advantage function solely based on the critic’s estimate \( V_\phi(x) \) by noting that \( Q(x_t, a_t) \approx r(x_t, a_t) + \gamma V_\phi(x_{t+1}) \). Similar to the standard policy update based on the advantage function, the expert selection stage can be formulated by optimizing the expected advantage \( \mathbb{E}_{p_\phi(a|x, m)}[A_m(x, a)] \) for expert \( m \) with

\[
A_m(x_t, a_t) = f(x_t, m, a_t) + \gamma V_\phi(x_{t+1}) - V_\phi(x_t). \quad \tag{10}
\]

Accordingly, we can define an expected advantage function \( \mathbb{E}_p[p_\phi(m|x)A(x, m)] \) for the selector with

\[
\hat{A}(x, m) = \mathbb{E}_{p_\phi(a|x, m)}[A_m(x, a)], \quad \tag{11}
\]

where \( m_t \) denotes the expert \( m \) selected at time \( t \). We estimate the double expectation by Monte Carlo sampling.

\[
F_t = \sum_{l=0}^{T} \gamma^l \left( r(x_{t+l}, a_{t+l}) - \frac{1}{\beta_2} \log \frac{p_\theta(a_{t+l} | x_{t+l}, m_{t+l})}{p(a_{t+l} | m_{t+l})} \right), \quad \tag{12}
\]

which we learn through a value function \( V_\phi \) for each expert. We define the selector’s discounted free energy as

\[
\hat{F_t} = \sum_{l=0}^{T} \gamma^l \hat{f}(x_{t+l}, m_{t+l}), \quad \tag{13}
\]

with \( \hat{f}(x, m) = \mathbb{E}_{p_\phi(a|x, m)}[r(x, a) - \frac{1}{\beta_2} \log \frac{p_\theta(a|x, m)}{p(a|m)}] \) that is learned through the selector’s value function \( V_\phi \). Now let \( p_\theta(m|x) \) be the selection policy and \( p_\phi(a|x, m) \) the expert policy. The expert selection stage is optimizing the expected advantage \( \mathbb{E}_p[p_\phi(a|x, m)A_m(x, a)] \) for expert \( m \) with

\[
A_m(x_t, a_t) = f(x_t, m, a_t) + \gamma V_\phi(x_{t+1}) - V_\phi(x_t). \quad \tag{14}
\]

Accordingly, we define an expected advantage function
We adopt this task from (Finn et al., 2017). In this $K$-shot problem, each task consists of learning to predict a function of the form $y = a \cdot \sin(x + b)$, with both $a \in [0, 1, 5]$ and $b \in [0, 2\pi]$ chosen uniformly, and the goal of the learner is to find $y$ given $x$ based on only $K$ pairs of $(x, y)$. Given that the underlying function changes in each iteration it is impossible to solve this problem with a single learner. Our results show that by combining expert networks, we are able to reduce the generalization error iteratively as we add more experts to our system—see Figure 2. where we also show how the system is able to capture the underlying problem structure as we add more experts. In Figure 3 we visualize how the selector’s partition of the problem space looks like. The shape of the emerged clusters indicates that the selection is mainly based on the amplitude $a$ of the current sine function, indicating that from an adaptation point-of-view it is more efficient to group sine functions based on amplitude $a$ instead of phase $b$. We can also see that an expert specializes on the low values for $b$ as it covers the upper region of the $a \times b$ space. The selection network splits this region among multiple experts if we increase the set of experts to 8 or more.

5.2. Few-Shot Classification

The Omniglot dataset (Lake et al., 2011) consists of over 1600 characters from 50 alphabets. As each character has only 20 samples each drawn by a different person, this forms a challenging meta-learning benchmark.

The selection policy now selects experts based on their free-energy that is computed over datasets $D_{val}$ and the selection policy depends on the training datasets $D_{train}$:

$$
\max_\theta \mathbb{E}_{p_{os}(m|D_{train})} \left[ \hat{f}(m, D_{val}) - \frac{1}{\beta_1} \log \frac{p_{os}(m|D_{train})}{p(m)} \right],
$$

where $\hat{f}(m, D_{val}) := \mathbb{E}_{p_{os}(\hat{y}|m, x)} \left[ -\mathcal{L}(\hat{y}, y) - \frac{1}{\beta_2} \log \frac{p_{os}(\hat{y}|m)}{p(\hat{y}|m)} \right]$ is the free energy of expert $m$ on dataset $D_{val}$, $\mathcal{L}(\hat{y}, y)$ is loss function, and $(x, y) \in D_{val}$. The experts optimize their free energy objective on the training dataset $D_{train}$ defined by

$$
\max_\theta \mathbb{E}_{p_{os}(\hat{y}|m, x)} \left[ -\mathcal{L}(\hat{y}, y) - \frac{1}{\beta_2} \log \frac{p_{os}(\hat{y}|m, x)}{p(\hat{y}|m)} \right],
$$

where $(x, y) \in D_{train}$.

We train the learner on a subset of the dataset ($\approx 80\%$, i.e., $\approx 1300$ classes) and evaluate the remaining $\approx 300$ classes, thus investigating the generalization. In each training episode we build the datasets $D_{train}$ and $D_{val}$ by selecting a target class $c_t$ and sampling $K$ positive and $K$ negative samples. To generate negative samples, we draw $K$ images randomly out of the remaining $N - 1$ classes. We present the selection network with the feature presentation of the $K$ positive training samples (see Figure 1), but evaluate the experts’ performance on the $2K$ test samples in $D_{val}$. In this way, the free energy of the experts becomes a generalization measure. Using this optimization scheme, we train the networks to become experts in recognizing a subset of classes. After a proper expert is selected we train that expert using the $2K$ samples from the training dataset.

We consider three experimental setups: 1) how does a learner with only a single hidden layer perform when trained nively compared to with sophisticated methods such as MAML (Finn et al., 2017) and Matching Nets (Vinyals et al., 2016) as a baseline? 2) does the system benefit from adding more experts and if so, at what rate? and 3) how does our method compare to the aforementioned algorithms? Regarding 1) we note that introducing constraints by reducing
We create a set of RL tasks by sampling the parameters for the Inverted Double Pendulum problem (Sutton, 1996) implemented in OpenAI Gym (Brockman et al., 2016), where the task is to balance a two-link pendulum in an upward position. We modify inertia, motor torques, reward function, goal position and invert the control signal – see Table 1 for details. We build the meta task set $T'$ on the same environment, but change the parameter distribution and range, providing new but similar reinforcement learning problems. During training, we sample $M$ environments in each episode and perform updates as described earlier. During evaluation, we measure the system’s performance on tasks sampled from $T'$ – see results in Figure 3, where we can see improving performance as more experts are added and the increasing mutual information in the selection stage indicates precise partitioning.

The expert policies are trained on the meta-training environment policies, but evaluated on unseen but similar validation environments. In this setting we define the discounted free energy $\tilde{F}_t$ with

$$\tilde{f}(x, m) = E_{{p}_\psi}(a|x, m) \left[ r_{\text{val}}(x, a) - \frac{1}{\beta_2} \log \frac{p(a|x, m)}{p(a|m)} \right],$$

where $r_{\text{val}}$ is a reward function defined by a validation environment (see Figure 3 for details).

6. Related Work

The hierarchical structure we employ is related to Mixture of Experts (MoE) models. (Jacobs et al., 1991; Jordan & Jacobs, 1994) introduced MoE as tree structured models for complex classification and regression problems, where the underlying approach is a divide and conquer paradigm. As in our approach, three main building blocks define MoEs: gates, experts, and a probabilistic weighting to combine expert predictions. Learning proceeds by finding a soft partitioning of the input space and assigning partitions to experts performing well on the partition. In this setting, the model response is then a sum of the experts’ outputs, weighted by how confident the gate is in the expert’s opinion (see (Yuksel et al., 2012) for an overview). This paradigm has seen recent interest in the field of machine learning, see e.g., (Aljundi et al., 2017; Rosenbaum et al., 2019; Jerfel et al., 2019; Shazeer et al., 2017). The approach we
Hierarchical Expert Networks for Meta-Learning

propose allows learning such models, but also has applications to more general decision-making settings such as reinforcement learning.

Our approach belongs to a wider class of models that use information constraints for regularization to deal more efficiently with learning and decision-making problems (Mar-tius et al., 2013; Leibfried et al., 2017; Grau-Moya et al., 2017; Achiam et al., 2017; Hihn et al., 2018; Grau-Moya et al., 2019). One such prominent approach is Trust Region Policy Optimization (TRPO) (Schulman et al., 2015). The main idea is to constrain each update step to a trust region around the current state of the system. This region is defined by $D_{KL}(\pi_{\text{new}} || \pi_{\text{old}})$ between the old policy and the new policy, providing a theoretcic monotonic policy improvement guarantee. In our approach we define this region by $D_{KL}$ between the agent’s posterior and prior policy, thus allowing to learn this region and to adapt it over time. This basic idea has been extend to meta-learning by (Rothfuss et al., 2018), which we use to compare our method against in meta-rl experiments.

Most other methods for meta-learning such as the work of (Finn et al., 2017) and (Ravi & Larochelle, 2017) find an initial parametrization of a single learner, such that the agent can adapt quickly to new problems. This initialization represents prior knowledge and can be regarded as an abstraction over related tasks and our method takes this idea one step further by finding a possibly disjunct set of such compressed task properties. Another way of thinking of such abstractions by lossy compression is to go from a task-specific posterior to a task-agnostic prior strategy. By having a set of priors the task specific information is available more locally then with a single prior, as in MAML (Finn et al., 2017) and the work of (Ravi & Larochelle, 2017). In principle, this can help to adapt within fewer iterations. Thus our method can be seen as the general case of such monolithic meta-learning algorithms. Instead of learning similarities within a problem, we can also try to learn similarities between different problems (e.g., different classification datasets), as is described in the work of (Yao et al., 2019). In this way, the partitioning is governed by different tasks, where our study however focuses on discovering meta-information within the same task family, where the meta-partitioning is determined solely by the optimization process and can thus potentially discover unknown dynamics and relations within a task family.

7. Discussion

We have introduced and evaluated a novel hierarchical information-theoretic approach to meta-learning. In particular, we leveraged an information-theoretic approach to bounded rationality. Although our method is widely applicable, it suffers from low sample efficiency in the RL domain.

A research direction would be to combine our system with model-based RL, which is known to improve sample efficiency. Another research direction would be to investigate the performance of our system in continual adaption tasks, such as in (Yao et al., 2019). Another limitation is the restriction to binary meta classification tasks, which we leave for feature work.

The results show that our method can identify sub-regions of the problem set and solve them efficiently with expert networks. In effect, this equips the system with initializations covering the problem space and thus enables it to adapt quickly to new but similar tasks. To reliably identify such tasks, we have proposed feature extraction methods for classification, regression, and reinforcement learning, that could be simply be replaced and improved in future work. The strength of our model is that it follows from simple principles that can be applied to a large range of problems. Moreover, the system performance can be interpreted in terms of the information processing of the selection stage and the expert decision-makers. We have shown empirically that our method achieves results comparable to recent meta-learning algorithms, such as MAML (Finn et al., 2017), Matching Nets (Vinyals et al., 2016), and Proximal Meta-Policy Search (Rothfuss et al., 2018).

ACKNOWLEDGMENTS

This work was supported by the European Research Council Starting Grant BRISC, ERC-STG-2015, Project ID 678082.

References


Hierarchical Expert Networks for Meta-Learning


Appendix

Few-Shot Classification Experimental details: We followed the design of (Vinyals et al., 2016) but reduce the number of blocks to one. We used a single convolutional block consisting of 32 3×3 filters with strided convolutions followed by a batch normalization layer and a ReLu non-linearity. During training we used a meta-batch size of 16. The convolutional autoencoder is a 3 layer network consisting of 16, 16, and 4 filters each with size 3×3 with strided convolutions followed by a leaky ReLu non-linearity. The layers are mirrored by de-convolotional layers to reconstruct the image. This results in an image embedding with dimensionality 64. The selection network is a two layer network with 32 units each, followed by a ReLu non-linearity, a dropout layer (Srivastava et al., 2014) per layer. We augment the dataset by rotating each image in 90, 180, and 270 degrees resulting in 80 images per class. We also normalize the images two be in (0,1) range. We evaluate our method by resetting the system to the state after training and allow for 10 gradient updates and report the final accuracy. To evaluate MAML on 2-way N-shot omniglot dataset we used a inner learning rate of $\alpha = 0.05$ and one inner update step per iteration for all settings. We used a single convolutional block followed by a fully connected layer with 64 units and a ReLU non-linearity. Note, that we reduce the number of layers to make the tests comparable. Using the suggested architectures by (Finn et al., 2017) we achieve classification accuracy $\geq 95\%$. To generate this figure, we ran 10-fold cross-validation on the whole dataset and show the averaged performance metric and the standard-deviation across the folds. In both settings, “0 bits” corresponds to a single expert, i.e., a single neural network trained on the task.

Meta-RL Experimental details: The selector’s actor and critic net are build of RNNs with 200 hidden units each. The critic is trained to minimize the Huber loss between the prediction and the cumulative reward. The experts are two layer networks with 64 units each followed by ReLu non-linearities and used to learn the parameters of a Gaussian distribution. The critics have the same architecture (except for the output dimensionality). The control signal $a$ is continuous in the interval [-1,1] and is generated by neural network that outputs $\mu$ and $\log(\sigma)$ of a gaussian. The action is then sampled by re-parameterizing the distribution to $p(a) = \mu + \exp(\sigma)\epsilon$, where $\epsilon \sim \mathcal{N}(0, 1)$, so that the distribution is differentiable w.r.t to the network outputs. We average the results over 10 random seeds and trained for 1000 episodes each with a batch of 64 environments.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Task Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>$T'$</td>
</tr>
<tr>
<td>Distance Penalty</td>
<td>$[10^{-3}, 10^{-1}]$</td>
</tr>
<tr>
<td>Goal Position</td>
<td>$[0.3, 0.4]$</td>
</tr>
<tr>
<td>Start Position</td>
<td>$[-0.15, 0.15]$</td>
</tr>
<tr>
<td>Motor Torques</td>
<td>$[0, 5]$</td>
</tr>
<tr>
<td>Inverted Control</td>
<td>$p = 0.5$</td>
</tr>
<tr>
<td>Gravity</td>
<td>$[0.01, 4.9]$</td>
</tr>
<tr>
<td>Motor Actuation</td>
<td>$[185, 215]$</td>
</tr>
</tbody>
</table>

Table 2. Environment Parameters for the Meta-RL Setting.