Appendix A. Supplementary Materials

Appendix A.1. iTAML vs Other Meta Algorithms

Lemma 1. Given a set of feature space parameters $\theta$ and task classification parameters $\phi = \{\phi_1, \phi_2, \ldots, \phi_T\}$, after $r$ inner loop updates, iTAML’s meta update gradient for task $i$ is given by,

$$g_{\text{itaml}}(i) = g_{i,0} + \cdots + g_{i,r-1},$$

where, $g_{i,j}$ is the $j^{th}$ gradient update with respect to $\{\theta, \phi_i\}$ on a single micro-batch.

Proof. Let $\Phi_i = \{\theta, \phi_i\}$ is the set of feature space parameters and task-specific parameters of the task $i$, $L_i(\Phi_i)$ is the loss calculated on a specific micro-batch $B_{i,r}$ for task $i$ using $\Phi_i$, and $\alpha$ is the inner loop learning rate. The parameters update is given by,

$$\Phi_{i,r} = \Phi_{i,r-1} - \alpha \nabla L_i(\Phi_{i,r-1}), \text{ where } \Phi_{i,0} = \Phi_i.$$

Let $g_{i,j} = \nabla L_i(\Phi_{i,j})$,

$$\Phi_{i,r} = \Phi_{i,r-1} - \alpha g_{i,r-1}.$$  

Using the meta gradient update rule defined in Reptile [19] i.e., $(\theta_{i,0} - \theta_{i,r})/\alpha$, we have,

$$g_{\text{itaml}}(i) = \frac{\theta_{i,0} - \theta_{i,r}}{\alpha} = \frac{\theta_{i,0} - (\theta_{i,r-1} - \alpha g_{i,r-1})}{\alpha} = \cdots = \frac{\theta_{i,0} - (\theta_{i,0} - \alpha g_{i,0} - \cdots - \alpha g_{i,r-1})}{\alpha} = g_{i,0} + g_{i,1} + \cdots + g_{i,r-1}$$

Lemma 2. Given a set of feature space parameters $\theta$ and task classification parameters $\phi = \{\phi_1, \phi_2, \ldots, \phi_T\}$, iTAML allows to keep the number of inner loop updates $r \geq 1$.

Proof. For a given task $t$, there will be $t$ gradients available for meta update,

$$g_{\text{itaml}} = \eta \frac{1}{t} \sum_{i=1}^{t} g_{\text{itaml}}(i) = \exp \left(-\beta \frac{t}{T} \right) \cdot \frac{1}{t} \sum_{i=1}^{t} \sum_{j=1}^{r-1} g_{i,j}.$$  

Reptile algorithm requires $r > 1$ since, $r = 1$ would result in joint training in Reptile algorithm. Reptile updates the parameters with respect to $\{\theta, \phi\}$ in the inner loop, while iTAML updates the parameters with respect to $\{\theta, \phi_i\}$ in the inner loop of task $i$. When $r = 1$,

$$g_{\text{itaml}} = \exp \left(-\beta \frac{t}{T} \right) \cdot \frac{1}{t} \sum_{i=1}^{t} g_{i,0} = \exp \left(-\beta \frac{t}{T} \right) \cdot \frac{1}{t} \sum_{i=1}^{t} \nabla L_i(\Phi_{i,0}) = \exp \left(-\beta \frac{t}{T} \right) \cdot \frac{1}{t} \sum_{i=1}^{t} \nabla L_i(\{\theta, \phi_i\})$$

$$\neq \frac{1}{t} \sum_{i=1}^{t} \nabla L_i(\{\theta, \phi\}) = g_{\text{joint}}$$

Appendix A.2. Additional Results

Figure 8: Classification accuracy on CIFAR100, with 2 tasks. Exemplar memory is set to 2000 samples and ResNet-18(1/3) is used for training. We keep $p = 20$ for experiments on data continuum.

Variation on $b$: iTAML uses a low $b$ value i.e., $b=1$. Parameter $b$ denotes the number of epochs for model update during adaptation. We observed that higher $b$ values do not have a significant impact on performance, but the time complexity increases linearly with $b$. Below, we report experimental results by changing $b$ from 1 to 5 and note that the accuracies does not improve significantly.

<table>
<thead>
<tr>
<th>$b$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>78.24%</td>
<td>78.48%</td>
<td>78.48%</td>
<td>78.53%</td>
<td>78.50%</td>
</tr>
</tbody>
</table>

Note on SVHN: For SVHN dataset, we keep $r = 4$ for the last task. This is due to the fact that, SVHN has a lower
variance in the data distribution and which forces the model to stuck at the early stages of local minima.

**Backends and Optimizers:** We evaluate our method with various architectural backends. Even with a very small model having (0.49M) parameters, iTAML can achieve 69.94% accuracy, with a gain of 13.46% over second-best (RPS-net 77.5M) method. ResNet-18 full model gives 80.27%. Further, iTAML is a modular algorithm, we can plug any optimizer into it. We evaluate iTAML with SGD, Adam [12] and RAdam [16], and respectively achieve a classification accuracy of 70.34%, 74.83% and 76.63% with these optimizers.