Deep Meta-Learning

AutoML Assignment 2

Filippo Boni & Joël During

1 Theory questions

1. Splitting the data

For a given dataset with classes $\mathcal{Y}$, we can sample training, validation, and test tasks in an N-way k-shot manner by splitting the classes into training, validation, and test sets without overlap. The tasks for each set can then be obtained by randomly sampling groups of N from the corresponding sets of classes. For each class, k examples are randomly sampled to form the support set, and the remainder will form the query set.

2. ProtoNet prediction

Given the support set $\mathbf{x}$, the embeddings $g_\theta(\mathbf{x})$, and the class labels as stated in the assignment, the ProtoNet makes predictions for a new query image $\hat{\mathbf{x}}$ with embedding $g_\theta(\hat{\mathbf{x}})$ in the following steps:

1. Compute the centroids for each class: $\mathbf{m}_1 = \frac{1}{2}(\mathbf{x}_1 + \mathbf{x}_2) = [1, 2.5]$ and $\mathbf{m}_2 = \frac{1}{2}(\mathbf{x}_3 + \mathbf{x}_4) = [-1, -2.5]$
2. Compute the squared Euclidian distances between the query embedding and the class centroids: $d_1 = ||g_\theta(\hat{\mathbf{x}}) - \mathbf{m}_1||_2^2 = 2^2 + 2^2 = 8$ and $d_2 = ||g_\theta(\hat{\mathbf{x}}) - \mathbf{m}_2||_2^2 = 0^2 + 3^2 = 9$
3. Compute the predictions for each class by taking the softmax over the negative squared Euclidian distances: $p_1 = \frac{\exp(-d_1)}{\exp(-d_1)+\exp(-d_2)} \approx 0.731$ and $p_2 = \frac{\exp(-d_2)}{\exp(-d_1)+\exp(-d_2)} \approx 0.269$

The class to which $\hat{\mathbf{x}}$ belongs is decided by considering the greater prediction of the two classes. In this case, the final prediction for $\hat{\mathbf{x}}$ would be class 1.

3. Meta-learning

A prototypical network is a meta-learning algorithm because it uses a set of tasks to learn a meaningful feature space that puts embeddings of examples from the same class close to one another while putting embeddings from distinct classes further away. It does this by updating the network weights that produce the embeddings through backpropagation. This feature space can be used for new tasks. In this way, ProtoNet uses the knowledge gained from previous tasks on new tasks.

4. Meta-gradient

The meta-gradient of MAML is $\nabla_\theta \mathcal{L}_{D^{te}_{\mathcal{T}_j}}(\theta_j^{(T)})$, where $\mathcal{T}_j$ represents the j^{th} task and $\theta_j^{(T)}$ represents the model parameters for task j after T inner gradient descent steps. This meta-gradient can be expanded, as shown below. Here, the chain rule is used, as well as the update rule for the MAML inner gradient descent step: $\theta_j^{(i)} = \theta_j^{(i-1)} - \alpha \nabla_{\theta_j^{(i-1)}} \mathcal{L}_{D^{tr}_{T_j}}(\theta_j^{(i-1)})$, where α is the inner learning rate.

$$
\begin{aligned}
\nabla_\theta \mathcal{L}_{D^{te}_{\mathcal{T}_j}}(\theta_j^{(T)}) &= \nabla_{\theta_j^{(T)}} \mathcal{L}_{D^{te}_{Tj}}(\theta_j^{(T)}) \cdot (\nabla_{\theta_j^{(T-1)}} \theta_j^{(T)}) \dots (\nabla_\theta \theta_j^{(1)}) \cdot (\nabla_\theta \theta) && \text{[chain rule]} \\
&= \nabla_{\theta_j^{(T)}} \mathcal{L}_{D^{te}_{Tj}}(\theta_j^{(T)}) \cdot (\nabla_{\theta_j^{(T-1)}} \theta_j^{(T)}) \dots (\nabla_\theta \theta_j^{(1)}) && [\nabla_\theta \theta = I] \\
&= \nabla_{\theta_j^{(T)}} \mathcal{L}_{D^{te}_{Tj}}(\theta_j^{(T)}) \cdot \prod_{t=1}^{T} \nabla_{\theta_j^{(t-1)}} \theta_j^{(t)} && [\theta = \theta_j^{(0)}] \\
&= \nabla_{\theta_j^{(T)}} \mathcal{L}_{D^{te}_{Tj}}(\theta_j^{(T)}) \cdot \prod_{t=1}^{T} \nabla_{\theta_j^{(t-1)}} (\theta_j^{(t-1)} - \alpha \nabla_{\theta_j^{(t-1)}} \mathcal{L}_{D^{tr}_{\mathcal{T}_j}}(\theta_j^{(t-1)})) && \text{[substitute update rule]} \\
&= \nabla_{\theta_j^{(T)}} \mathcal{L}_{D^{te}_{Tj}}(\theta_j^{(T)}) \cdot \prod_{t=1}^{T} (I - \alpha \nabla_{\theta_j^{(t-1)}} (\nabla_{\theta_j^{(t-1)}} \mathcal{L}_{D^{tr}_{\mathcal{T}_j}}(\theta_j^{(t-1)})) && \text{[simplify using identity matrix]}
\end{aligned}
$$

From this equation, we can see that the meta-gradient is equal to zero when $\nabla_{\theta_j^{(T)}} \mathcal{L}_{D^{te}_{Tj}}(\theta_j^{(T)}) = 0$. That is, the meta-gradient is equal to zero when the gradient over the loss function with respect to the parameters after T inner gradient steps is zero. This occurs when the fine-tuned model parameters have reached a (local) minimum in the loss function over the query set $D^{te}_{\mathcal{T}_j}$.

The outer gradient descent step of MAML updates the initialization parameters based on the sum of gradients over all tasks in the training set. Therefore, θ is not updated when $\nabla_{\theta_j^{(T)}} \mathcal{L}_{D^{te}_{\mathcal{T}_j}}(\theta_j^{(T)}) = 0$ for all tasks $\mathcal{T}_j$. This occurs when the initialization parameters θ are positioned such that a local optimum over the loss function for each task with respect to its query set can be reached within T inner gradient descent steps.

5. Distant tasks

For a new task that is distant from the tasks we have trained ProtoNet and MAML on, we expect MAML to perform better. This is because ProtoNet is **nonparametric**, meaning it does not adjust any parameters for a new task. Therefore, it requires that the learned features are informative for the new task, which they might not be if the task is very different. On the other hand, MAML is **parametric**; for a new task, it adjusts its parameters using one or several gradient descent steps. This can help the algorithm to adapt to new tasks.

6. ProtoNet as a linear model

In order to answer, we analyze: $y_\theta(\hat{\mathbf{x}})_n$ and expand $-\|g_\theta(\hat{\mathbf{x}}) - \mathbf{m}_n\|_2^2$.

$$
-\|g_\theta(\hat{\mathbf{x}}) - \mathbf{m}_n\|_2^2 = -g^T(\hat{\mathbf{x}}) g(\hat{\mathbf{x}}) + 2\mathbf{m}_n^T g(\hat{\mathbf{x}}) - \mathbf{m}_n^T \mathbf{m}_n
$$

Since the first term is constant w.r.t. to the class 'n', it can be ignored. The result is:
$2\mathbf{m}_n^T g(\hat{\mathbf{x}}) - \mathbf{m}_n^T \mathbf{m}_n$

Considering the vector $y_\theta(\hat{\mathbf{x}})$:

$$y_\theta(\hat{\mathbf{x}}) = \begin{pmatrix} y_\theta(\hat{\mathbf{x}})_1 \\ \cdot \\ \cdot \\ y_\theta(\hat{\mathbf{x}})_n \end{pmatrix} \propto \begin{pmatrix} -\|g_\theta(\hat{\mathbf{x}}) - \mathbf{m}_1\|_2^2 \\ \cdot \\ \cdot \\ -\|g_\theta(\hat{\mathbf{x}}) - \mathbf{m}_n\|_2^2 \end{pmatrix} \propto \begin{pmatrix} 2\mathbf{m}_1^T g(\hat{\mathbf{x}}) \\ \cdot \\ \cdot \\ 2\mathbf{m}_n^T g(\hat{\mathbf{x}}) \end{pmatrix} + \begin{pmatrix} -\mathbf{m}_1^T\mathbf{m}_1 \\ \cdot \\ \cdot \\ -\mathbf{m}_n^T\mathbf{m}_n \end{pmatrix}$$

The result of this expression is:

$$= 2 \begin{pmatrix} \cdot\ \mathbf{m}_1\ \cdot \\ \cdot \\ \cdot \\ \cdot\ \mathbf{m}_n\ \cdot \end{pmatrix} g(\hat{\mathbf{x}}) + \begin{pmatrix} -\mathbf{m}_1^T\mathbf{m}_1 \\ \cdot \\ \cdot \\ -\mathbf{m}_n^T\mathbf{m}_n \end{pmatrix} = \mathbf{W}g(\hat{\mathbf{x}}) + \mathbf{b} \qquad \mathbf{W} = 2 \begin{pmatrix} \cdot\ \mathbf{m}_1\ \cdot \\ \cdot \\ \cdot \\ \cdot\ \mathbf{m}_n\ \cdot \end{pmatrix}, \mathbf{b} = \begin{pmatrix} -\mathbf{m}_1^T\mathbf{m}_1 \\ \cdot \\ \cdot \\ -\mathbf{m}_n^T\mathbf{m}_n \end{pmatrix}$$

To conclude, we mimic the nearest-centroid type of behavior using the linear classifier $f(\hat{\mathbf{x}}) = \mathbf{W}g(\hat{\mathbf{x}}) + \mathbf{b}$ with $2\mathbf{m}_n$ as n-th row of $\mathbf{W}$ and $-\mathbf{m}_n^T\mathbf{m}_n$ as n-th entry of $\mathbf{b}$. In this way, the predictions of the linear model are proportional to the one of the Prototypical Network.

2 Empirical questions

We set up several experiments to evaluate the performance of these meta-learning algorithms and the effect of some of their parameters. In each experiment, the algorithm makes 40,000 meta-updates during training and is evaluated on the validation set after every 500 meta-updates. Unless otherwise specified, the experiments are in the 5-way, 1-shot setting with 15 query samples per class, 1 inner gradient descent step, a meta-batch size of 1, an inner learning rate of 0.4, an outer learning rate of 0.001, and the Adam optimizer [3]. In order to deal with randomness, we performed each experiment 3 times with three different random seeds, and considered the averaged results.

1. Algorithm comparison

In Figure 1, we can see the results of the ProtoNet [2], MAML[1], and FOMAML[1] algorithms on the meta-training and meta-validation tasks. The ProtoNet algorithm has the lowest loss at every epoch on the meta-training and the meta-validation sets. It also performs consistently, with a low standard deviation. MAML performs slightly worse than ProtoNet, with a slightly higher loss at every epoch. The standard deviation on the training set is also bigger. However, MAML seems to not have fully converged after 40,000 iterations. Therefore, it might outperform ProtoNet, given more training steps. FOMAML performs the worst, with the highest loss and standard deviation. These results carry over to the accuracy of these algorithms, which can also be seen in Figure 1. Again, ProtoNet performs best, reaching an accuracy of over 95% on the validation set within 10,000 iterations. FOMAML achieved the lowest accuracy and has the largest standard deviation.

2. Number of inner gradient update steps

We set up an experiment to investigate the effect of the number of inner gradient update steps T on the performance of FOMAML and MAML. The results are presented in Figure 2. We can see that the FOMAML algorithm performs better when T is larger than 1, achieving a lower loss and standard deviation on both the meta-training and meta-validation sets. However, there is no clear effect of increasing the number of inner gradient update steps beyond 3. For the MAML algorithm, the final losses after 40,000 meta-update steps are comparable for all the values of T. The only difference is

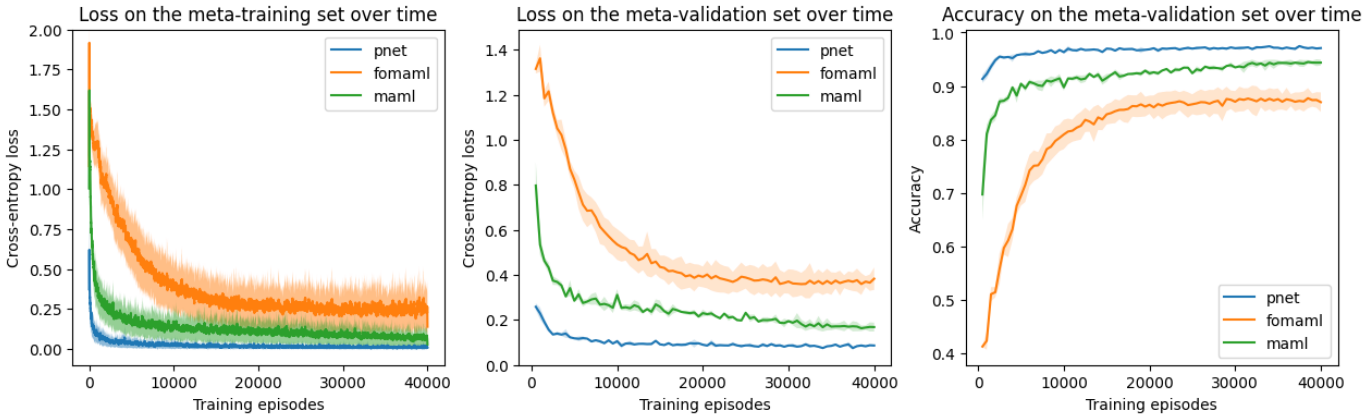

Figure 1: Loss and accuracy of ProtoNet, FOMAML, and MAML on the meta-training and -validation sets. Results are averaged over three runs, and the standard deviation is shown. For the training set, the running average over 50 iterations is shown.

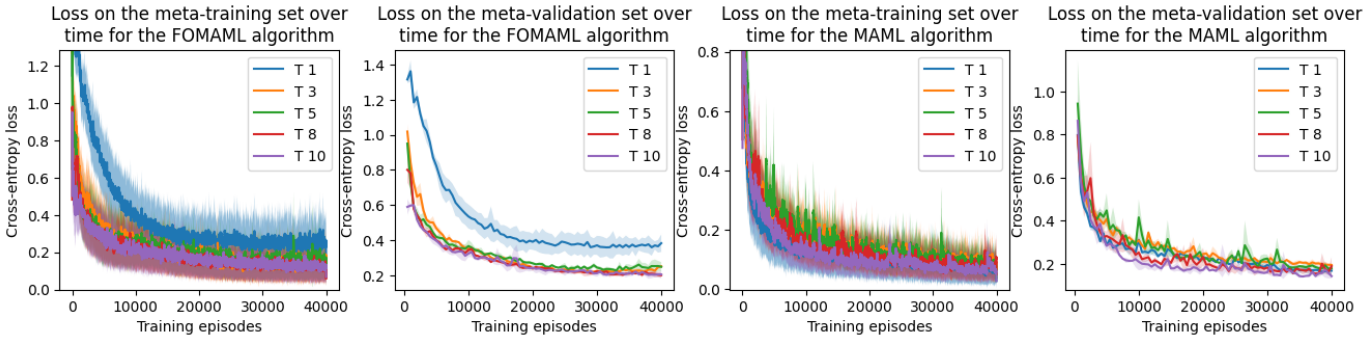

Figure 2: Loss and accuracy of FOMAML and MAML on the meta-training and -validation sets for different numbers of inner gradients descent steps T. Results are averaged over three runs, and the standard deviation is shown. For the training set, the running average over 50 iterations is shown.

that the algorithm learns slightly faster when using many inner gradient update steps like 8 or 10. An interesting observation is that the loss of FOMAML is close to that of MAML for $T \geq 3$. This shows that FOMAML is a good approximation of MAML when using several inner gradient update steps.

3. Number of classes per task

We set up an experiment to investigate the effect of the number of classes N per task on the performance of ProtoNet, FOMAML, and MAML. The results are presented in Figure 3. For all three algorithms, the performances degrade when the number of classes increases. Indeed the task becomes harder to solve with a greater number of classes because variation is added to the meta-training and meta-validation sets. The ProtoNet algorithm is the least affected by the degradation. We hypothesize that this is because it can still learn meaningful embeddings because the feature space has a high dimensionality (64). Whereas for (FO)MAML, the classification performed by the CNN is more sensitive to the extra variation in the meta-training and meta-validation sets. FOMAML is the most influenced, resulting in a very low accuracy for higher numbers of classes. This is likely because the 1^{st}-order approximation of the meta-gradient becomes less accurate for a higher number of classes. Considering the results of point

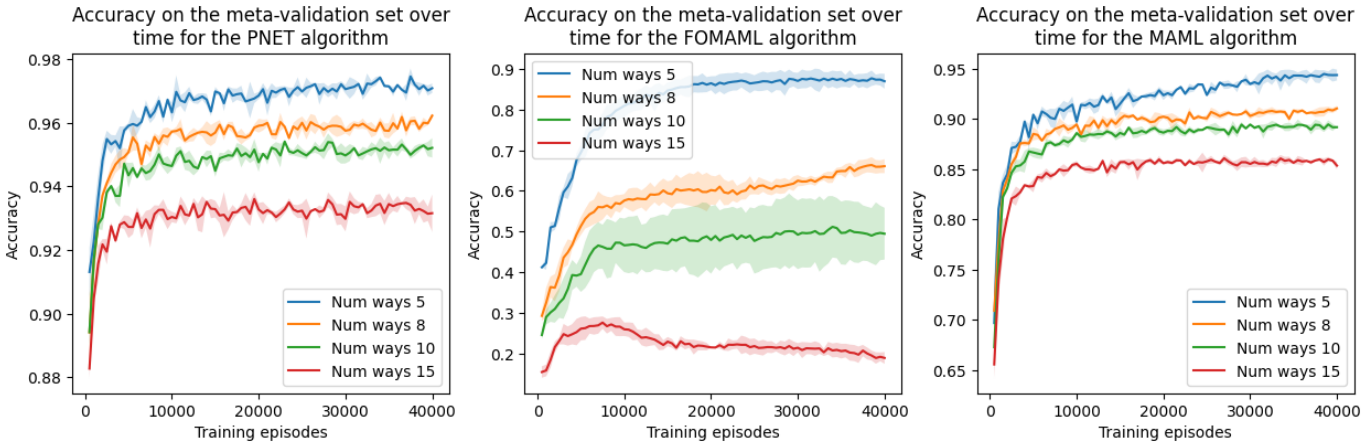

Figure 3: Loss and accuracy of ProtoNet, FOMAML, and MAML on the meta-validation sets for different numbers of classes per task. Results are averaged over three runs, and the standard deviation is shown.

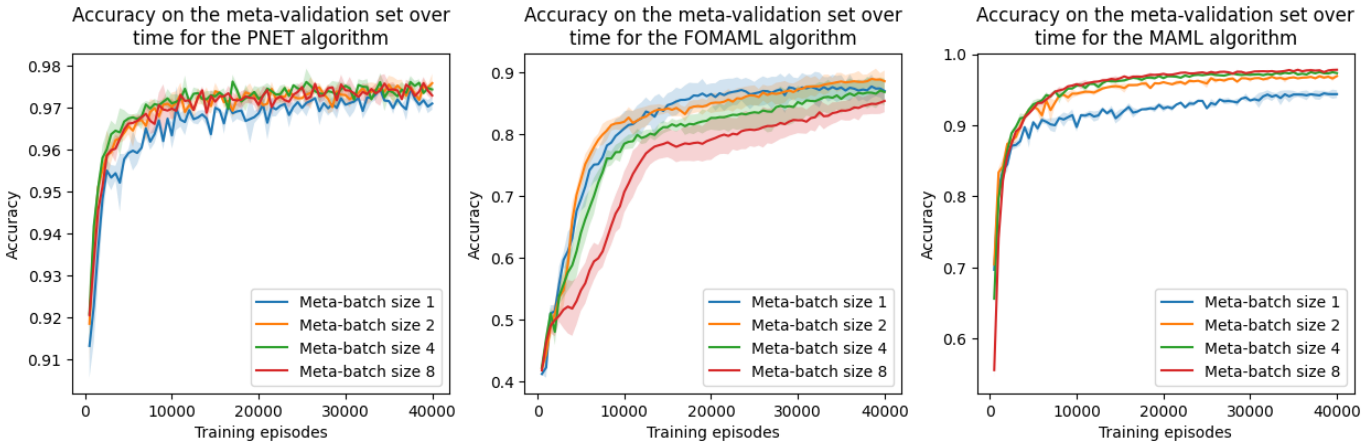

Figure 4: Loss and accuracy of ProtoNet, FOMAML, and MAML on the meta-validation sets for different meta-batch sizes. Results are averaged over three runs, and the standard deviation is shown.

2 we think this effect might be reduced by increasing the number of inner gradient update steps.

3 Bonus (to be eligible for a 10)

We perform an additional experiment to investigate the effect of the meta-batch size on the ProtoNet, FOMAML, and MAML algorithms. We compare the default size of 1 to meta-batch sizes of 2, 4, and 8. We see that for ProtoNet and MAML, the accuracy increases slightly when using a meta-batch size larger than 1. For MAML, this increase is larger, resulting in an accuracy that is similar to that of ProtoNet. This increase is likely because, even though the number of meta-update steps remains the same, the algorithm is trained on more tasks for larger meta-batch sizes. For FOMAML, we see that the performance decreases for meta-batch sizes larger than 2. This is likely because the gradients of FOMAML are approximations, and therefore grouping several in a batch creates an even worse approximation of the actual gradient.

4 Work distribution

For this assignment, we worked together on the code, experiments, and report. Therefore, we can not acknowledge any part of the final result to be an individual's contribution. Rather, the whole work should be seen as a team effort.

References

[1] Sergey Levine Chelsea Finn, Pieter Abbeel. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400v3*, 2017.

[2] Richard Zemel Jake Snell, Kevin Swersky. Prototypical networks for few-shot learning. *31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.*, 2017.

[3] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.