A Short-Path Gradient Descent

Optimizing Eq. 3 naively by Algorithm 1 would be costly and ineffective. It is costly because in the case of domain adaptation (unlike for example, few-shot learning [12], the inner loop requires many iterations). So back-propagating through the whole optimization path to update the initial Θ in the outer loop will produce multiple high-order gradients. For example, if the inner loop applies j iterations, we will have

$$\begin{aligned} \Theta^{(1)} &= \Theta - \alpha \nabla_{\Theta^{(0)}} \mathcal{L}_{\text{uda}}(.) \\ &\dots \\ \Theta^{(j)} &= \Theta^{(j-1)} - \alpha \nabla_{\Theta^{(j-1)}} \mathcal{L}_{\text{uda}}(.) \end{aligned} \tag{13}$$

then the outer loop will update the initial condition as

$$\Theta^* = \Theta - \alpha \overbrace{\nabla_{\Theta} \mathcal{L}_{\text{sup}}(\Theta^{(j)}, \mathcal{D}_{\text{val}})}^{\text{Meta Gradient}} \tag{14}$$

where higher-order gradient will be required for all items $\nabla_{\Theta^{(0)}} \mathcal{L}_{\text{uda}}(.), \dots, \nabla_{\Theta^{(j-1)}} \mathcal{L}_{\text{uda}}(.)$ in the update of Eq. 14.

One intuitive way of eliminating higher-order gradients for computing Eq. 14 is making $\nabla_{\Theta^{(0)}} \mathcal{L}_{\text{uda}}(.), \dots, \nabla_{\Theta^{(j-1)}} \mathcal{L}_{\text{uda}}(.)$ constant during the optimization. Then, Eq. 14 is equivalent to

$$\Theta^* = \Theta - \alpha \overbrace{\nabla_{\Theta^{(j)}} \mathcal{L}_{\text{sup}}(\Theta^{(j)}, \mathcal{D}_{\text{val}})}^{\text{First-order Meta Gradient}} \tag{15}$$

However, in order to compute Eq. 15, one still needs to store the optimization path of Eq. 13 in memory and back-propagate through it to optimize Θ, which requires high computational load. Therefore, we propose a practical solution an iterative meta-learning algorithm to iteratively optimize the model parameters during training.

Shortest Path Optimization To obtain the meta gradient in Eq. 15 in a more efficient way, we propose a more scalable and efficient meta-learning method using shortest-path gradient (S-P.G.) [36]. Before the optimization of Eq. 13, we copy the parameters Θ as $\tilde{\Theta}^{(0)}$ and use $\tilde{\Theta}^{(0)}$ in the inner-level algorithm.

$$\tilde{\Theta}^{(j)} = \begin{cases} \tilde{\Theta}^{(0)} - \alpha \nabla_{\Theta^{(0)}} \mathcal{L}_{\text{uda}}(\tilde{\Theta}^{(0)}, \mathcal{D}_{\text{tr}}), \\ \dots \\ \tilde{\Theta}^{(j-1)} - \alpha \nabla_{\Theta^{(0)}} \mathcal{L}_{\text{uda}}(\tilde{\Theta}^{(j-1)}, \mathcal{D}_{\text{tr}}) \end{cases} \tag{16}$$

then, after finishing the optimization in Eq. 16, we can get the shortest-path gradient between two items $\tilde{\Theta}_i^{(j)}$ and Θ_i.

$$\nabla_{\Theta}^{\text{short}} = \Theta - \tilde{\Theta}^{(j)} \tag{17}$$

Different from Eq. 15, we use this shortest-path gradient $\nabla_{\Theta}^{\text{short}}$ and initial parameter Θ to compute $\mathcal{L}_{\text{sup}}(.)$ as,

$$\mathcal{L}_{\text{sup}}(\Theta_i - \nabla_{\Theta_i}^{\text{short}}, \mathcal{D}_{\text{val}}) \tag{18}$$

Then, one-step meta update of Eq. 18 will be,

$$\begin{aligned} \Theta_i^* &= \Theta_i - \alpha \nabla_{\Theta_i} \mathcal{L}_{\text{sup}}(\Theta_i - \nabla_{\Theta_i}^{\text{short}}, \mathcal{D}_{\text{val}}) \\ &= \Theta_i - \alpha \nabla_{\Theta_i - \nabla_{\Theta_i}^{\text{short}}} \mathcal{L}_{\text{sup}}(\Theta_i - \nabla_{\Theta_i}^{\text{short}}, \mathcal{D}_{\text{val}}) \\ &= \Theta_i - \alpha \nabla_{\tilde{\Theta_i}^{(j)}} \mathcal{L}_{\text{sup}}(\tilde{\Theta_i}^{(j)}, \mathcal{D}_{\text{val}}) \end{aligned} \tag{19}$$

Effectiveness: We can see that one update of Eq. 19 corresponds to that of Eq. 15, which proves that using shortest-path optimization has the equivalent effectiveness to the first-order meta optimization. **Scalability/Efficiency:** The computation memory of the first-order meta-learning increases linearly with the inner-loop update steps, which is constrained by the total GPU memory. However, for the shortest-path optimization, storing the optimization graph is no longer necessary, which makes it scalable and efficient. We also experimentally evaluate that one step shortest-path optimization is 7x faster than one-step first-order meta optimization in our setting. The overall algorithm flow is shown in Algorithm 2.

B Additional Illustrative Schematics

To better explain the contrast between our online meta-learning domain adaptation approach with the sequential meta-learning approach, we add a schematic illustration in Figure 4. The main difference between sequential and online meta-learning approaches is how do we distribute the meta and DA updates. Sequential meta-learning approach performs meta updates and DA updates sequentially. And online meta-learning conducts the alternative meta and DA updates throughout the whole training procedure.

C Additional Experiments

Visualization of the Learned Features We visualize the learned features of MCD and Meta-MCD on PACS when sketch is the target domain as shown in Figure 5. We can see that both MCD and Meta-MCD can learn discriminative features. However, the features learned by Meta-MCD is more separable than vanilla MCD. This explains why our Meta-MCD performs better than the vanilla MCD method.

Effect of varying S Our online meta-learning method has iteration hyperparameters S and J. We fix $J = 1$ throughout, and analyze the effect of varying S here using the DomainNet MSDA experiment with ResNet-18. The result in Table 7 shows that MetaDA is rather insensitive to this hyperparameter.

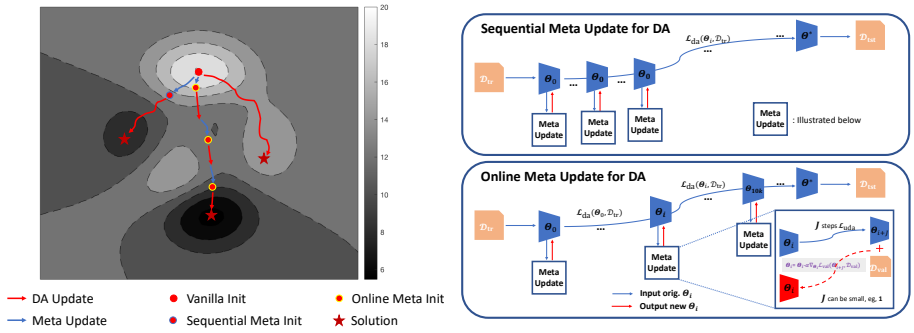

Fig. 4: Illustrative schematics of sequential and online meta domain adaptation. Left: Optimization paths of different approaches on domain adaptation loss (shading). (Solid line) Vanilla gradient descent on a DA objective from a fixed start point. (Multi-segment line) Online meta-learning iterates meta and gradient descent updates. (Two segment line) Sequential meta-learning provides an alternative approximation: update initial condition, then perform gradient descent. Right: (Top) Sequential meta-learning performs meta updates and DA updates sequentially. (Bottom) Online meta-learning alternates between meta-optimization and domain adaptation.

Method	Meta-MCD (S=3)	Meta-MCD (S=5)	Meta-MCD (S=10)
DomainNet (ave.)	41.02	40.98	40.93

Table 7: MetaDA is insensitive to the update ratio hyperparameter S – Results for MSDA ResNet-18 performance on DomainNet.

Varying the Number of Source Domains in MSDA For multi-source DA, the performance of both Meta-DA and the baselines is expected to drop with fewer sources (same for SSDA if fewer labeled target domain points). To disentangle the impact of the number of sources for baseline vs Meta-DA we compare MSDA by Meta-MCD on PACS with 2 vs 3 sources. The results for Meta-MCD vs vanilla MCD are 82.30% vs 80.07% (two source, gap 2.23%) and 87.24% vs 84.79% (three source, gap 2.45%). Meta-DA margin is similar with reduction of domains. Most difference is accounted for by the impact on the base DA algorithm.

Other base DA methods Besides the base DA methods evaluated in the main paper (DANN, MCD and MME), our method is applicable to any base domain adaptation method. We use the published code of JiGen[7] and M^{3}SDA[8], and further apply our Meta-DA on the existing code. The results are shown in Table 8 and 9. From the results, we can see that our Meta-JiGen and Meta-M^{3}SDA-β improves over the base methods by 3.42% and 1.2% accuracy respectively,

[7] https://github.com/fmcarlucci/JigenDG
[8] https://github.com/VisionLearningGroup/VisionLearningGroup.github.io

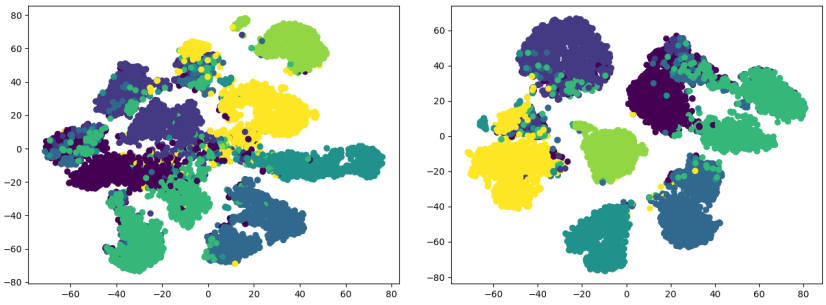

Fig. 5: t-SNE [32] visualization of learned MCD (left) and Meta-MCD (right) features on PACS (sketch as target domain). Different colors indicate different categories.

Method	C,P,S→A	A,P,S→C	A,C,S→P	A,C,P→S	Ave.
JiGen [7]	84.88	81.07	97.96	79.05	85.74
JiGen*	81.54	85.88	97.25	68.21	83.22
Meta-JiGen	85.21	86.13	97.31	77.91	86.64 (+3.42)

Table 8: Test accuracy on PACS. * our run.

which confirms our Meta-DA's generality. The reason we excluded these from the main results is that: (i) Re-running JiGen's published code on our compute environment failed to replicate their published numbers. (ii) M^{3}SDA as a base algorithm is very slow to run comprehensive experiments on. Nevertheless, these results provide further evidence that Meta-DA can be a useful module going forward to plug in and improve future new base DA methods as well as those evaluated here.

Initialization Dependence of Domain Adaptation One may not think of domain adaptation as being sensitive to initial condition, but given the lack of target domain supervision to guide learning, different initialization can lead to a significant difference in accuracy. To illustrate this we re-ran MCD-based DA on PACS with sketch target using different initializations. From the results in Tab 10, we can see that both different classic initialization heuristics, and simple perturbation of a given initial condition with noise can lead to significant differences in final performance. This confirms that studying methods for tuning initialization provide a valid research direction for advancing DA performance.

Method	mt,up,sv,sy →mm	mm,up,sv,sy →mt	mt,mm,sv,sy →up	mt,mm,up,sy →sv	mt,mm,up,sv →sy	Ave.
M^{3}SDA-β [38]	72.82	98.43	96.14	81.32	89.58	87.65
Meta-M^{3}SDA-β	71.73	98.79	97.80	84.81	91.12	88.85 (+1.2)

Table 9: Test accuracy on Digit-Five.

Classifier Init	Kaiming $\mathcal{U}$	Xavier $\mathcal{U}$	Kaiming $\mathcal{N}$	Xavier $\mathcal{N}$
	74.49	73.02	64.27	73.66
Feat. Extr. Init	No perturb	$+\ \epsilon \in \mathcal{N}(0, 0.01)$	$+\ \epsilon \in \mathcal{N}(0, 0.02)$	$+\ \epsilon \in \mathcal{N}(0, 0.03)$
	74.49	71.85	59.99	52.18

Table 10: Test accuracy of MCD on PACS (sketch) with different initialization.