

Supplementary Materials: Exploring Stable Meta-optimization Patterns via Differentiable Reinforcement Learning for Few-shot Classification

Anonymous Authors

1 DETAILS OF CLASSIFIERS

We leverage three popular non-parametric classifiers in our approach: ProtoNet[7], DSN[6], and MetaOptNet[4]. In the following, we describe the details of each classifier. We omit time step t for simplicity.

ProtoNet[7]¹ predicts the probability distribution directly on the instance embeddings. Class prototypes are first computed via averaging support samples in each class:

$$\mathbf{p}_c = \frac{1}{K} \sum_{(\mathbf{x}_i, y_i) \in S_c} \mathbf{e}_{\mathbf{x}_i}, \quad (1)$$

where S_c denotes samples from class c . Then the probability distribution of a query sample \mathbf{x}_q is computed as:

$$p(y_q = c \mid \mathbf{x}_q) = \frac{\exp(-d(\mathbf{e}_{\mathbf{x}_q}, \mathbf{p}_c))}{\sum_{c'} \exp(-d(\mathbf{e}_{\mathbf{x}_q}, \mathbf{p}_{c'}))}, \quad (2)$$

where $d(\cdot)$ denotes a distance function (e.g., Euclidean distance or cosine distance) between the query sample and the prototype. We use cosine distance in the model RMO-PN. The reward r is defined as:

$$r = \frac{1}{NM} \sum_c \log p(y_q = c \mid \mathbf{x}_q). \quad (3)$$

DSN[6]² predicts the probability distribution with the subspace projection distance. For class c , a new set of samples are first computed as:

$$\tilde{S}_c = [\mathbf{e}_{\mathbf{x}_{c,1}} - \mathbf{p}_c, \dots, \mathbf{e}_{\mathbf{x}_{c,K}} - \mathbf{p}_c], \quad (4)$$

where \mathbf{p}_c denotes the prototype of class c calculated with Equation 1. Then a class-specific projection matrix \mathbf{P}_c is calculated via truncated singular value decomposition on \tilde{S}_c . The distance from a query sample \mathbf{x}_q to its projection onto \mathbf{P}_c is calculated as:

$$d_c(\mathbf{x}_q) = -\| (I - \mathbf{P}_c \mathbf{P}_c^\top) (\mathbf{e}_{\mathbf{x}_q} - \mathbf{p}_c) \|^2, \quad (5)$$

where I denotes the identity matrix. The probability distribution of the query \mathbf{x}_q is defined as:

$$p(y_q = c \mid \mathbf{x}_q) = \frac{\exp(d_c(\mathbf{x}_q))}{\sum_{c'} \exp(d_{c'}(\mathbf{x}_q))}. \quad (6)$$

The distance between two subspaces \mathbf{P}_i and \mathbf{P}_j is defined as:

$$\begin{aligned} \delta^2(\mathbf{P}_i, \mathbf{P}_j) &= \|\mathbf{P}_i \mathbf{P}_i^\top - \mathbf{P}_j \mathbf{P}_j^\top\|_F^2 \\ &= 2n - 2 \|\mathbf{P}_i^\top \mathbf{P}_j\|_F^2. \end{aligned} \quad (7)$$

To obtain more discriminative subspaces, we need to maximize $\delta^2(\mathbf{P}_i, \mathbf{P}_j)$, which is equal to minimize $\|\mathbf{P}_i^\top \mathbf{P}_j\|_F^2$. The reward r is defined as:

$$r = \frac{1}{NM} \sum_c \log p(y_q = c \mid \mathbf{x}_q) - \lambda \sum_{i \neq j} \|\mathbf{P}_i^\top \mathbf{P}_j\|_F^2. \quad (8)$$

The value of λ is set to 0.03. For 1-shot case, we generate a support image for each class by data augmentation via flipping the support image following Simon *et al.*[6].

MetaOptNet[4]³ predicts the probability distribution with convex base learners. Lee *et al.*[4] utilizes a multi-class Support Vector Machine (SVM) as the learner. A N -class SVM can be expressed as $\theta = \{\mathbf{w}_c\}_{c=1}^N$. The optimization problem of θ can be formulated as:

$$\theta = \underset{\{\mathbf{w}_c\}}{\operatorname{argmin}} \min_{\{\epsilon_i\}} \frac{1}{2} \sum_c \|\mathbf{w}_c\|^2 + C \sum_i \epsilon_i,$$

s.t.

$$\mathbf{w}_{y_i} \cdot \mathbf{e}_{\mathbf{x}_i} - \mathbf{w}_c \cdot \mathbf{e}_{\mathbf{x}_i} \geq 1 - \delta(y_i, c) - \epsilon_i, \quad \forall i, c \quad (9)$$

where $i \in \{1, \dots, NK\}$ and $c \in \{1, \dots, N\}$, C is the regularization item and $\delta(\cdot)$ denotes the Kronecker delta function. Denote

$$\mathbf{w}_c(\mathbf{a}^c) = \sum_i a_i^c \mathbf{e}_{\mathbf{x}_i}, \quad \forall c \quad (10)$$

the dual formulation of the objective in Equation 9 can be expressed as:

$$\max_{\{a_i^c\}} -\frac{1}{2} \sum_c \|\mathbf{w}_c(\mathbf{a}^c)\|^2 + \sum_i a_i^{y_i},$$

s.t.

$$\begin{aligned} a_i^{y_i} &\leq C, \quad a_i^c \leq 0, \quad \forall c \neq y_i, \\ \sum_c a_i^c &= 0, \quad \forall i. \end{aligned} \quad (11)$$

This is a Quadratic Program (QP) over the dual variables $\{\mathbf{a}^c\}_{c=1}^N$. Following Lee *et al.*[4], we use the GPU-based differentiable QP solver QPTH [1] to solve Equation 11. The gradients of θ_G , θ_p , and θ_g can be computed via applying the implicit function theorem on the KKT conditions, provided by Barratt [2]. The reward r is defined as:

$$r = \mathbf{w}_{y_q} \cdot \mathbf{e}_{\mathbf{x}_q} - \log \sum_c \exp(\mathbf{w}_c \cdot \mathbf{e}_{\mathbf{x}_q}). \quad (12)$$

The value of C is set to 0.1. The maximum iteration number of the QP solver is set to 5. For RMO-MON, we use the feature maps before the last global pooling layer of the backbone to calculate the reward, following Lee *et al.*[4]. The dimension of the feature maps used for classification is 1600 and 16000 for Conv-4 and ResNet-12, respectively. The feature map is globally averaging pooled before encoding the task state.

¹<https://github.com/orobix/Prototypical-Networks-for-Few-shot-Learning-PyTorch>

²https://github.com/chrysts/dsn_fewshot

³<https://github.com/kjunelee/MetaOptNet>

Table 1: Classification accuracy and 95% confidence interval on different values of T .

Backbone	Model	Number of time step T						
		0	1	2	3	4	5	6
Conv-4	PN	71.25±0.16	73.18±0.16	74.33±0.16	74.84±0.16	74.97±0.16	75.11±0.16	74.94±0.16
	DSN	71.96±0.18	73.16±0.17	74.43±0.17	75.39±0.17	75.98±0.17	76.32±0.16	76.91±0.16
	MON	69.43±0.17	71.66±0.17	72.91±0.17	73.62±0.17	74.21±0.17	74.64±0.18	74.49±0.18
ResNet-12	PN	80.81±0.14	83.48±0.14	84.54±0.14	85.59±0.15	86.01±0.15	86.03±0.15	86.67±0.15
	DSN	81.36±0.19	84.17±0.18	85.26±0.19	85.84±0.18	86.33±0.19	86.69±0.15	86.41±0.15
	MON	80.46±0.17	82.72±0.17	84.19±0.18	84.72±0.18	85.26±0.18	85.52±0.15	85.65±0.15
ResNet-18	PN	82.19±0.14	84.52±0.15	85.73±0.15	85.61±0.15	85.92±0.15	86.32±0.15	86.12±0.15
	DSN	82.60±0.17	84.89±0.15	86.19±0.15	86.84±0.15	87.35±0.15	87.54±0.16	87.13±0.15

Table 2: Classification accuracy on different values of γ .

Backbone	Discount factor γ				
	0.1	0.5	0.8	0.9	0.99
Conv-4	73.79	74.45	74.76	75.11	74.83
ResNet-12	85.01	85.25	85.94	86.03	85.21
ResNet-18	85.05	85.84	86.26	86.32	85.67

Table 3: Classification accuracy on different values of D .

Backbone	Length of DND D				
	100	500	1000	2000	3000
Conv-4	74.07	74.60	74.79	75.11	74.78
ResNet-12	85.47	85.62	85.91	86.03	85.82
ResNet-18	85.65	85.74	85.89	86.32	86.10

2 IMPLEMENTATION DETAILS

2.1 Backbone Architecture

For Conv-4, the network G_1 contains two convolution blocks, where each block is composed of {3*3 conv with 64 filters, BN, ReLU, 2*2 maxpool}, and the network G_2 contains two convolution blocks, where each block is composed of {3*3 conv with 64 filters, BN with FiLM, ReLU, 2*2 maxpool}, following a global max-pooling layer. For ResNet-12, denote R_k as a residual block consisted of three {3*3 conv with k filters, BN, LeakyReLU(0.1)}, R_{kA} as R_k with a FiLM layer inserted after the last BN layer, MP as a 2*2 max-pooling layer, and $DB(k, b)$ as a DropBlock layer with $keep_rate=k$ and $block_size=b$. The architecture of G_1 is: R64-MP-DB(0.9, 1)-R160-MP-DB(0.9, 1). The architecture of G_2 is: R320A-MP-DB(0.9, 5)-R640A-MP-DB(0.9, 5), with a global average pooling layer. For ResNet-18, denote R_k as a residual block consisted of two {3*3 conv with k filters, BN, ReLU}, R_{kA} as R_k with a FiLM layer inserted after the last BN layer. The architecture of G_1 is: {3*3 conv with k filters, BN, ReLU}-R64-R64-R128-R128. The architecture of G_2 is: R256A-R256A-R512A-R512A, with a global average pooling layer. For WRN-28-10, denote R_k as a residual block consisted of two {BN, ReLU, 3*3 conv with k filters} with a dropout layer inserted between the two blocks, R_{kA} as R_k with a FiLM layer inserted after the last BN layer. The architecture of G_1 is: {3*3 conv with k filters}-R16-R16-R16-R16-R160-R160-R160. The architecture of G_2 is: R320-R320A-R320-R320A-R640-R640A-R640-R640A-BN, with a global average pooling layer.

2.2 Pre-training Strategy of Main Experiments

Following Ye *et al.* [9], we utilize the same pre-training strategy to obtain initial weights of the backbones. The backbone is first appended with a *softmax* layer and then trained to correctly classify all the classes in training class split of the dataset using cross-entropy

loss. In this stage, we utilize random crop, color jittering, and random flip to augment the training images. After training for 300 epochs, we validate the few-shot classification performance of the model on the validation split of dataset after each epoch. In validation, we randomly sample 200 1-shot N -way few-shot tasks, where each task contains one support sample and 15 queries per class for evaluation. Here N is the number of classes in validation class split of dataset. A nearest neighbor classifier is utilized to measure the accuracy. We choose the weights of backbone with the best few-shot classification performance on validation set. Then the weights are used to initialize the backbone G_1 and G_2 . In meta-training, the weights are optimized together with the whole model.

2.3 Details of Ablation on DenseNet Backbone

We use the same network architecture of DenseNet-121 as in SimpleShot [8]. We train the backbone with data augmentation for 90 epochs from scratch using the SGD optimizer to minimize the cross-entropy loss on base classes. The learning rate is set to 0.1 and the batch size is set to 256. On *MiniImageNet*, we scale the learning rate by 0.1 at epoch 45 and 66 respectively. On *TieredImageNet*, we scale the learning rate by 0.1 after every 30 epochs. In meta-training, the whole model is meta-trained for 200 epochs, with each epoch including 100 tasks. We use 600 randomly sampled tasks on validation set to choose the best model. During evaluation, the model is tested over 10000 randomly sampled tasks on testing set. The values of hyper-parameters are set as the same in the main experiments.

2.4 Details of Ablation with CNAPS and Simple-CNAPS

Following [3, 5], we use ResNet-18 pre-trained on ImageNet as backbone. In meta-training, we train the whole model for totally 110000 randomly sampled tasks, with 16 tasks per batch. We use

Adam optimizer with learning rate of 0.0005. We validate the model every 1000 tasks with 200 randomly sampled tasks to choose the best model. During evaluation, we use 600 tasks to measure the classification accuracy, following Bateni *et al.* [3].

3 ADDITIONAL EXPERIMENTS

3.1 Selection of Hyper-parameters

We explore the impacts of hyper-parameter T , D , and γ on classification performance. The experiments are conducted under 5-shot 5-way on *MiniImageNet*. Table 1 lists the classification accuracy with different values of T . It is observed that increasing T can lead to performance gains on all three models. Our intuition is that increasing the value of T can take stronger impact to model via accumulated long-term reward. Table 3 lists the classification accuracy with different values of D . It can be seen that with the increase of D , the model obtains better performance. While the value of D reaches 3000, the classification performance degrades. Our intuition is that larger values of D make DND store past associations on a longer time span to help the model learn generalizable patterns. But too larger values of D can introduce much redundancy of associations, leading to performance degradation. Table 2 lists the classification accuracy with different values of γ . It can be seen that the best value of γ is around 0.9. A proper value of γ is able to put an appropriate penalty on uncertainty of future rewards, but too much or less values can weaken the impact. According to the results, we set the value of T , D , and γ as 5, 2000, and 0.9, respectively.

3.2 Visualization

We plot the curves of validation accuracy among 2-step RMO-PN and 5-step RMO-PN (denoted as “2-step PN” and “5-step PN”, respectively) with Conv-4 backbone across 2000 training epochs in Figure 1. The experiments are conducted under 5-shot 5-way with each epoch including 10 meta-tasks. For a fair comparison, we augment ProtoNet with averaging to obtain task state and a linear layer to learn the parameters γ and β (denoted as “Baseline PN”), which has similar quantity of learnable parameters with ours. It can be seen that the variance of the curve of “5-step PN” is observably lower comparing to “2-step PN” and “Baseline PN”. “2-step PN” and “5-step PN” also have a faster convergence rate than “Baseline PN”. This suggests that our strategy does contribute on meta-training stability via exploring stable meta-optimization patterns with differentiable RL.

REFERENCES

- [1] Brandon Amos and J Zico Kolter. 2017. Optnet: Differentiable optimization as a layer in neural networks. In *ICML*. 136–145.
- [2] Shane Barratt. 2018. On the differentiability of the solution to convex optimization problems. (2018). arXiv preprint arXiv:1804.05098.
- [3] Peyman Bateni, Raghav Goyal, Vaden Masrani, Frank Wood, and Leonid Sigal. 2020. Improved few-shot visual classification. In *CVPR*. 14493–14502.
- [4] Kwonjoon Lee, Subhansu Maji, Avinash Ravichandran, and Stefano Soatto. 2019. Meta-learning with differentiable convex optimization. In *CVPR*. 10657–10665.
- [5] James Requeima, Jonathan Gordon, John Bronskill, Sebastian Nowozin, and Richard E Turner. 2019. Fast and flexible multi-task classification using conditional neural adaptive processes. In *NeurIPS*, Vol. 32.
- [6] Christian Simon, Piotr Koniusz, Richard Nock, and Mehrtash Harandi. 2020. Adaptive subspaces for few-shot learning. In *CVPR*. 4136–4145.
- [7] Jake Snell, Kevin Swersky, and Richard Zemel. 2017. Prototypical networks for few-shot learning. In *NeurIPS*, Vol. 30.

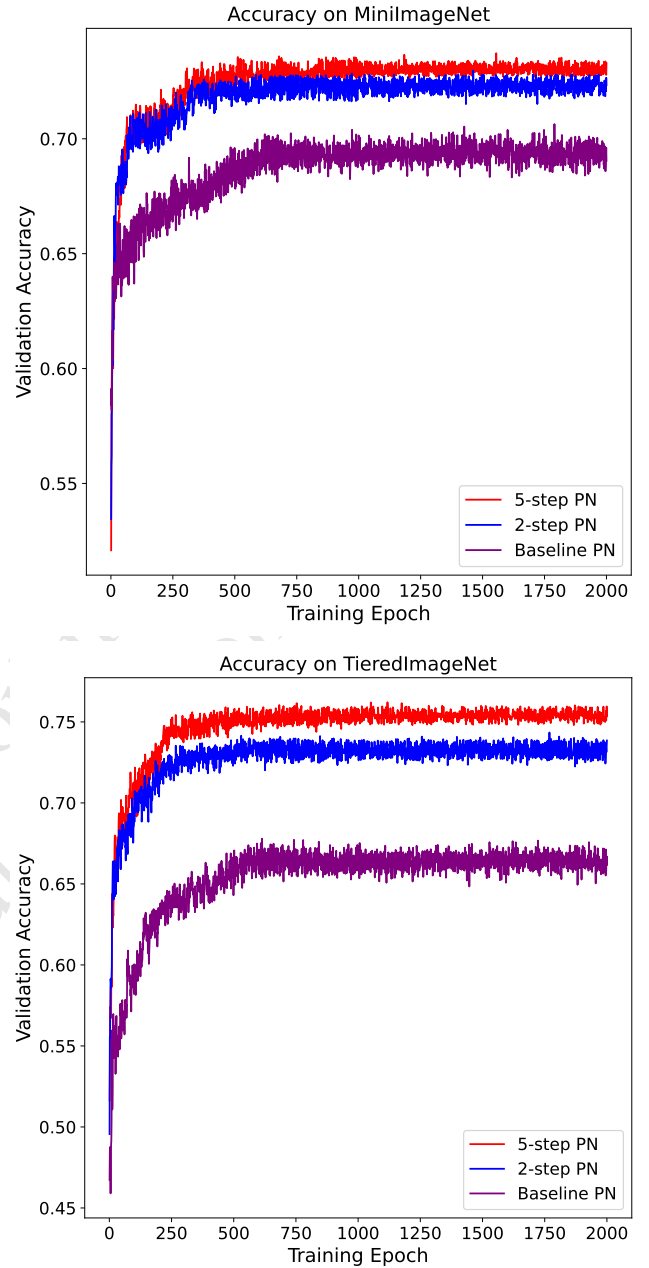


Figure 1: The classification accuracies of RMO-PN on validation set against ProtoNet.

- [8] Yan Wang, Wei-Lun Chao, Kilian Q Weinberger, and Laurens van der Maaten. 2019. Smpshot: Revisiting nearest-neighbor classification for few-shot learning. (2019). arXiv preprint arXiv:1911.04623.
- [9] Han-Jia Ye, Hexiang Hu, De-Chuan Zhan, and Fei Sha. 2020. Few-shot learning via embedding adaptation with set-to-set functions. In *CVPR*. 8808–8817.