Supplementary for GEMS: Generating Efficient Meta-Subnets

Varad Pimpalkhute
TCS Research,
Mumbai, India
varad.p@tcs.com

Shruti Kunde
TCS Research,
Mumbai, India
shruti.kunde@tcs.com

Rekha Singhal
TCS Research,
Mumbai, India
rekha.singhal@tcs.com

A. Experimental Setup

All experiments are conducted in isolation on a dedicated MIG A100 GPU setup, with 30 GB RAM, 8vCPUS and 10GB GPU memory. For each experiment we record the model accuracy and time required for training. Each experiment has been repeated 3 times with different seeds to ensure sufficient randomness.

A.1. Architecture Details

The model architecture is similar to the original MAML [2] paper, i.e., 4 modules with 3 x 3 convolutions and 64 filters with a stride of 2, followed by batch normalization, a ReLU nonlinearity and 2 × 2 max-pooling. The MLP is a fully connected network, where each layer consists of 2N hidden units, where N is the number of layers of the base learner network. The ReLU activation function is placed between the MLP layers.

A.2. Dataset Details

We have used 4 quasi-benchmark datasets (CuBirds, VGGFlowers, Aircraft, Fungi) from the field of meta-learning for our experimentation purpose. The CU-Birds dataset contains 11,788 images of 200 bird species. The data is split into 200 classes that are divided into 100, 50 and 50 for meta-training, meta-validation and meta-testing respectively. VGGFlower is a dataset consisting of 102 flower categories. Each class consists between 40-258 images. The aircraft dataset contains 10,200 images. There are 100 images for each of the 102 aircraft model variants. The data is divided into 3 equally-sized training, validation and test subsets. The Fungi dataset contains 1394 species of fungi. It has 85578 training images, 4182 validation images and 9758 testing images. The table below provides a detailed list of hyper-parameters that were used when setting up the experiments.

A.3. Training Details

The default setting for adaption steps is 5 and fast LR is 0.01. We use 0.5 as the random initialization for sparsity in the MMSUP experiments and the value for sparsity is learned during training. These settings are used in our experiment setup for comparing performance of MMSUP and BMP to MAML (base line) and Multi-MAML. The varying sparsity, inner loop learning rate and adaptation steps values depicted in Table 1 are used for setting up experiments conducted as a part of ablation studies outline in the main paper.

Table 1. Experiment setup parameters

Parameter	Value
Iterations	30000
Meta-batch size	5
Shots	1
Ways	5
Adaptation steps	1,2,3,4,5
Fast LR	0.1, 0.5, 0.05, 0.001, 0.005
Meta-LR	0.0001
Random seeds	21, 42, 56
Sparsity%	10, 20, ..., 90

B. Additional Results

Table 2 depicts a snapshot of additional combinations of datasets on which we tested performance of BMP and MMSUP.

C. Analysis of GEMS

C.1. Analysis of convergence of MMSUP

In this section, we will study whether there is any positive learning in the network after each training iteration. We want to prove that

$$\mathcal{L}_{n+1} < \mathcal{L}_n$$

where, n is the current training iteration.

Let us consider a neuron whose weights are getting masked or unmasked after every iteration. Masking depends on the sparsity value of the layer, i.e., if sparsity is k, then top k weights will be fed to the neuron. Consider two weights – W_i and W_j. In n^{th} iteration, W_j is fed as input to the neuron whereas W_i is masked. While, in the $n+1^{th}$ iteration, W_i is fed as input to the neuron whereas W_j is masked. This signifies that,

Table 2. 5ways, 1 shot

Train Dist	Test Dist	MAML Accuracy	MAML Time	Multi-MAML Accuracy	Multi-MAML Time	BMP Accuracy	BMP Time	MMSUP Accuracy	MMSUP Time
CUB200 + Aircraft	CUB	0.468	7.07	0.533	15.51	0.528	12.13	0.545	8.76
	Aircraft	0.309		0.399		0.428		0.397	
VGG102 + Aircraft	VGG102	0.656	7.05	0.728	14.92	0.738	12.09	0.722	8.2
	Aircraft	0.342		0.399		0.448		0.414	
Fungi + Aircraft	Fungi	0.372	8.6	0.424	16.92	0.437	13.7	0.393	14.31
	Aircraft	0.324		0.399		0.438		0.427	

(Meta-learning Architectures: MAML, Multi-MAML, BMP, MMSUP)

$$\bar{W}_i - W_i > \bar{W}_j - W_j$$

Thus, during backpropagation, we can say that the value of gradients of W_i is greater than that of W_j, which bring us to the following equation:

$$-\frac{\delta \mathcal{L}}{\delta I_p} W_i Z_i > -\frac{\delta \mathcal{L}}{\delta I_p} W_j Z_j \tag{1}$$

where, I_p is the input and Z_i is the activation function for the i^{th} weight. Hence, after swapping the weights / edges of the neural network, the equation of the input I_p changes from

$$I_p = \sum W_k Z_k + W_i Zi$$

to

$$\bar{I}_p = \sum W_k Z_k + W_j Zj$$

Hence, change in the input is given by:

$$\bar{I}_p - I_p = W_j Zj - W_i Zi \tag{2}$$

We now compute the change in the loss from iteration n to iteration $n + 1$.

$$\mathcal{L}(\bar{I}_p) = \mathcal{L}(I_p + (\bar{I}_p - I_p))$$

Expanding the Taylor's series and approximating it, we get,

$$\mathcal{L}(\bar{I}_p) = \mathcal{L}(I_p) + \frac{\delta \mathcal{L}}{\delta I_p}(\bar{I}_p - I_p)$$

Thus, substituting Equation 2 in the above equation, we get,

$$\mathcal{L}(\bar{I}_p) = \mathcal{L}(I_p) + \frac{\delta \mathcal{L}}{\delta I_p}(W_j Zj - W_i Zi) \tag{3}$$

From Equation 1 and Equation 3, we conclude that, $\mathcal{L}(\bar{I}_p) < \mathcal{L}(I_p)$. We can henceforth conclude that as the network learns on the subnetwork, it also converges towards the global minima.

D. Resources

The results reported in the paper are produced using open source and free software. We build up on learn2learn [1] [1] to make custom modules for BMP and MMSUP keeping in spirits with the PyTorch [9] framework. We also made use of Numpy [4] and Pandas [7] libraries for building custom few-shot learning dataset modules and sparsity in the network using lottery ticket hypothesis. All plots were generated using Matplotlib [5].

All the final experiments were run in a Linux environment. For producing and debugging, we made use of Google Colab Pro that also ran on a Linux environment. The datasets were hand-picked from public domain datasets allowing usage for research purposes.cub200 We made use of CUBirds [10], VGGFlowers [8], FGVC Aircraft [6] and FGVC Fungi [3] datasets.

[1] learn2learn is an open-source PyTorch library for meta-learning research https://github.com/learnables/learn2learn.

E. PyTorch Code Snippets

In this section, we walkthrough some of the important modules required for implementation of BMP and MMSUP approaches. We present pseudo codes on how to reproduce the algorithms in PyTorch.

```
class BinaryMask(torch.autograd.Function):
    def __init__(self):
        super(BinaryMask, self).__init__()

    @staticmethod
    def forward(ctx, input):
        # Compute Binary mask --
        # 1: Values greater than zero
        # 0: Values less than or equal to zero

    @staticmethod
    def backward(ctx, grad):
        # Return grad without computing gradients
        # Straight Through Estimator (STE)
```

Code Listing 1. Binary Mask

```
# Number of adaptation steps
for step in range(adaptation_steps):
    # Compute loss on input batch
    train_error = loss(learner(data_batch),
    true_labels)
    param_dict = # Get backbone parameters
    # Compute gradients using training error and
    param dict
    support_loss_grad = grad(train_error,
    param_dict.values(), retain_graph=True)
    task_embedding = # Stack mean of weights and
    gradient per layer
```

```python
    layer_pred = regularizer(torch.mul(task_embedding, inp_embedding))
    # Use previously learned knowledge
    # to compute binary mask
    mask = BinaryMask.apply(layer_pred)
    # Update weights of the backbone
    learner.adapt(train_error, mask)
```

Code Listing 2. Inner Loop Optimization

```python
class ComputeMask(torch.autograd.Function):
    @staticmethod
    def forward(ctx, params, sparsity):
        # k is the sparsity of the layer
        # 1: For top k parameters
        # 0: Rest of the parameters

    @staticmethod
    def backward(ctx, g):
        # Return gradient as it is
```

Code Listing 3. Generate Sparsity Mask

```python
class SupermaskConv(nn.Conv2d):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        # Set an initialization (e.g., kaiming normal)

    def forward(self, x, sparsity):
        self.scores = self.weight.detach()
        # Compute a subnetwork from backbone
        subnet = ComputeMask.apply(self.scores, sparsity)
        # Apply the mask on the network
        w = self.weight * subnet
        return F.conv2d(x, w, self.bias, self.stride, self.padding, self.dilation, self.groups)
```

Code Listing 4. Compute Sparse Network

References

[1] Sébastien M R Arnold, Praateek Mahajan, Debajyoti Datta, Ian Bunner, and Konstantinos Saitas Zarkias. learn2learn: A library for Meta-Learning research. Aug. 2020.

[2] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *CoRR*, abs/1703.03400, 2017.

[3] Tobias Guldberg Frøslev, Jacob Heilmann-Clausen, Christian Lange, Thomas Læssøe, Jens Henrik Petersen, Ulrik Søchting, Thomas Stjernegaard Jeppesen, and Jan Vesterholt. Danish mycological society, fungal records database, 2022.

[4] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, Sept. 2020.

[5] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[6] S. Maji, J. Kannala, E. Rahtu, M. Blaschko, and A. Vedaldi. Fine-grained visual classification of aircraft. Technical report, University of Massachusetts Amherst, 2013.

[7] Wes McKinney et al. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. Austin, TX, 2010.

[8] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *2008 Sixth Indian Conference on Computer Vision, Graphics Image Processing*, pages 722–729, 2008.

[9] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[10] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge J. Belongie. The caltech-ucsd birds-200-2011 dataset. In *Caltech-UCSD*, 2011.