EvoGrad: Efficient Gradient-Based Meta-Learning and Hyperparameter Optimization

Ondrej Bohdal, Yongxin Yang, Timothy Hospedales
{ondrej.bohdal, yongxin.yang, t.hospedales}@ed.ac.uk

EvoGrad overview

- Efficient approach for gradient-based meta-learning
  - Existing approaches: expensive second-order derivatives and a longer computational graph
  - EvoGrad: first-order derivatives only
- Inspired by evolutionary techniques to efficiently compute hypergradients
- Significantly lower runtime and memory usage, with similar performance as existing methods
  - Can scale meta-learning to larger network architectures
- Evaluated on several recent meta-learning applications
  - Cross-domain few-shot learning with feature-wise transformations
  - Noisy label learning with MetaWeightNet
  - Low-resource cross-lingual learning with MetaXL

Background

- Goal: estimate hyperparameters $\boldsymbol{\lambda}$ that minimize the validation loss ℓ_V of the model parameterized by θ and trained with loss ℓ_T and $\boldsymbol{\lambda}$

$$\boldsymbol{\lambda}^* = \arg\min_{\boldsymbol{\lambda}} \ell_V^*(\boldsymbol{\lambda}), \text{ where } \ell_V^*(\boldsymbol{\lambda}) = \ell_V(\boldsymbol{\lambda}, \boldsymbol{\theta}^*(\boldsymbol{\lambda})) \text{ and } \boldsymbol{\theta}^*(\boldsymbol{\lambda}) = \arg\min_{\boldsymbol{\theta}} \ell_T(\boldsymbol{\lambda}, \boldsymbol{\theta})$$

- Necessary to calculate hypergradient $\frac{\partial \ell_V^*(\boldsymbol{\lambda})}{\partial \boldsymbol{\lambda}} = \frac{\partial \ell_V(\boldsymbol{\lambda}, \boldsymbol{\theta}^*(\boldsymbol{\lambda}))}{\partial \boldsymbol{\lambda}} + \frac{\partial \ell_V(\boldsymbol{\lambda}, \boldsymbol{\theta}^*(\boldsymbol{\lambda}))}{\partial \boldsymbol{\theta}^*(\boldsymbol{\lambda})} \frac{\partial \boldsymbol{\theta}^*(\boldsymbol{\lambda})}{\partial \boldsymbol{\lambda}}$

- Direct term $\frac{\partial \ell_V(\boldsymbol{\lambda}, \boldsymbol{\theta}^*(\boldsymbol{\lambda}))}{\partial \boldsymbol{\lambda}}$ is typically zero, so simple first-order approximation is not available

EvoGrad update

- Evolutionary inner step
  - Sample *K* random perturbations $\boldsymbol{\epsilon}$ and apply them to model parameters θ as $\boldsymbol{\theta}_k = \boldsymbol{\theta} + \boldsymbol{\epsilon}_k$
  - Compute training losses $\ell_k = f(\mathcal{D}_T | \boldsymbol{\theta}_k, \boldsymbol{\lambda})$ for *K* models on the current minibatch $\mathcal{D}_T$
  - Calculate weights for each model as

$$w_1, w_2, \ldots, w_K = \mathrm{softmax}([-\ell_1, -\ell_2, \ldots, -\ell_K]/\tau)$$

  - Update model parameters via the affine combination

$$\boldsymbol{\theta}^* = w_1 \boldsymbol{\theta}_1 + w_2 \boldsymbol{\theta}_2 + \cdots + w_K \boldsymbol{\theta}_K$$

- Compute hypergradient using validation data $\frac{\partial \ell_V}{\partial \boldsymbol{\lambda}} = \frac{\partial f(\mathcal{D}_V | \boldsymbol{\theta}^*)}{\partial \boldsymbol{\lambda}}$
  - Inner loop does not use gradients, so overall the method is first-order
- Do standard update of the base model afterwards

Illustration

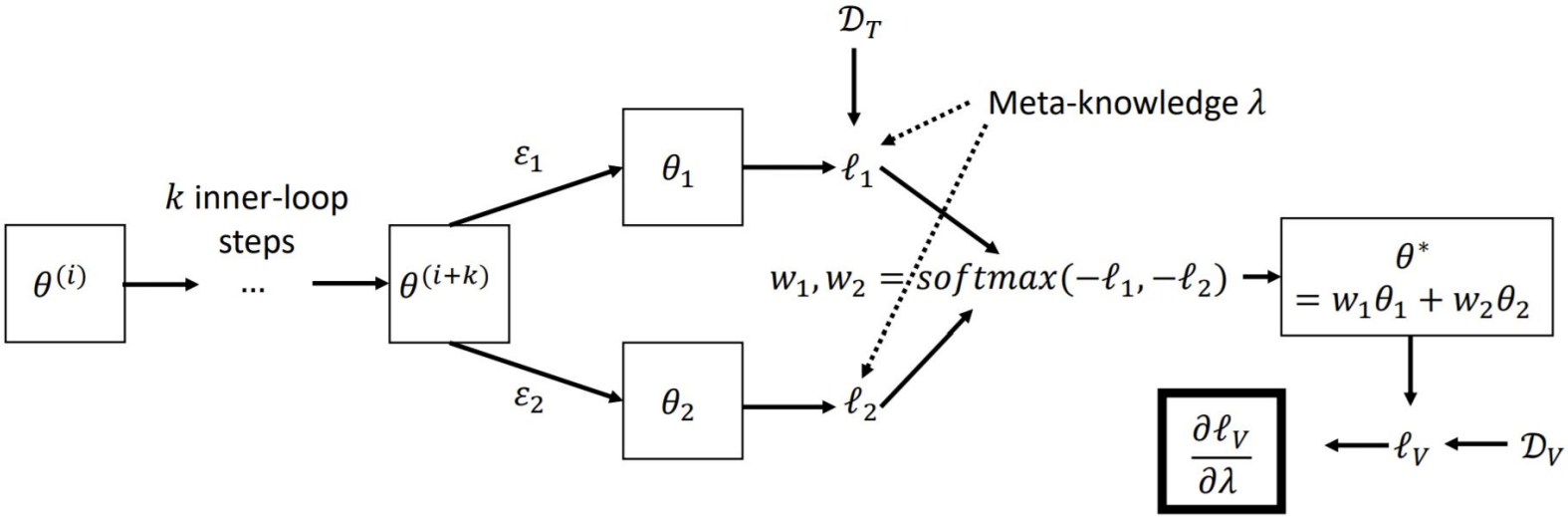

Figure 1: Graphical illustration of a single EvoGrad update using $K = 2$ model copies.

Comparison to existing methods

Table 1: Comparison of hypergradient approximations of $T_1 - T_2$ and EvoGrad.

Method	Hypergradient approximation
$T_1 - T_2$	$\frac{\partial \ell_V}{\partial \boldsymbol{\lambda}} - \frac{\partial \ell_V}{\partial \boldsymbol{\theta}} \times \boldsymbol{I} \frac{\partial^2 \ell_T}{\partial \boldsymbol{\theta} \partial \boldsymbol{\lambda}^T}$
EvoGrad (ours)	$\frac{\partial \ell_V}{\partial \boldsymbol{\lambda}} + \frac{\partial \ell_V}{\partial \boldsymbol{\theta}} \times \mathcal{E} \frac{\partial \boldsymbol{w}}{\partial \boldsymbol{\ell}} \frac{\partial \boldsymbol{\ell}}{\partial \boldsymbol{\lambda}} = \frac{\partial \ell_V}{\partial \boldsymbol{\lambda}} + \frac{\partial \ell_V}{\partial \boldsymbol{\theta}} \times \mathcal{E} \frac{\partial \operatorname{softmax}(-\boldsymbol{\ell})}{\partial \boldsymbol{\lambda}}$

Table 2: Comparison of asymptotic memory and operation requirements of EvoGrad and $T_1 - T_2$ meta-learning strategies. P is the number of model parameters, H is the number of hyperparameters. $K \ll H$ is the number of model copies in EvoGrad. Note this is a first-principles analysis, so the time requirements are different when using e.g. reverse-mode backpropagation that uses parallelization.

Method	Time requirements	Memory requirements
$T_1 - T_2$	$\mathcal{O}(PH)$	$\mathcal{O}(P + H)$
EvoGrad (ours)	$\mathcal{O}(KP + H)$	$\mathcal{O}(P + H)$

Experiments

- Simple problems
  - 1-dimensional problem where we try to find the minimum of a function
  - Meta-learning rotation transformation
- Recent meta-learning applications
  - Cross-domain few-shot learning with feature-wise transformations
  - Noisy label learning with MetaWeightNet
  - Low-resource cross-lingual learning with MetaXL

Illustration using a 1-dimensional problem

- Minimize $f_V(x) = (x - 0.5)^2$ where parameter x is optimized using SGD with $f_T(x) = (x - 1)^2 + \lambda \|x\|_2^2$ that includes a meta-parameter λ
- Closed-form solution for the hypergradient is $g(\lambda) = (\lambda - 1)/(\lambda + 1)^3$

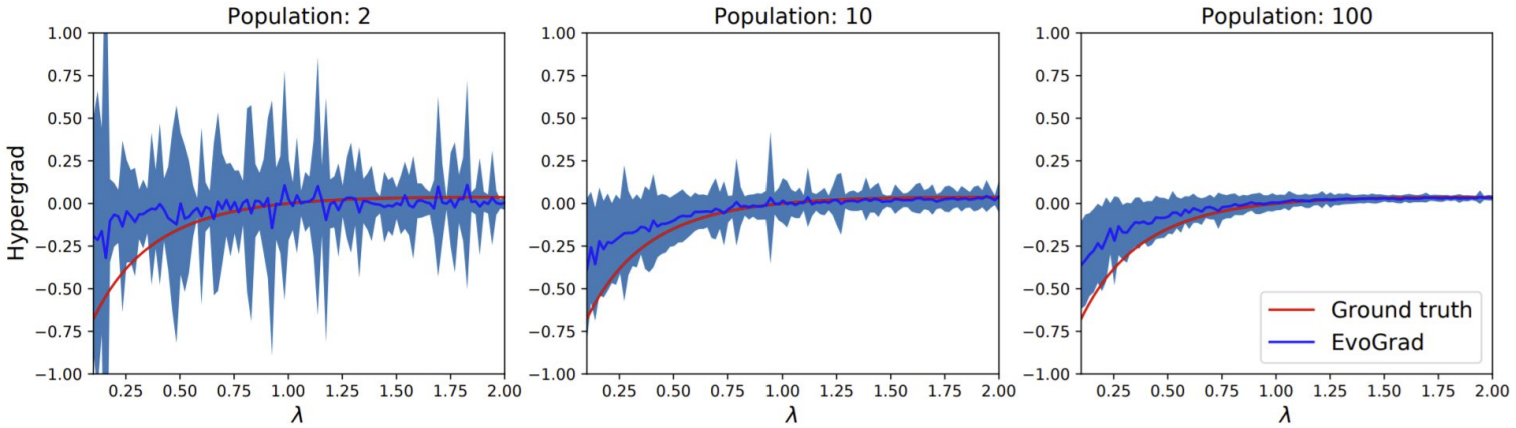

Figure 2: Comparison of the hypergradient $\partial f_V / \partial \lambda$ estimated by EvoGrad vs the ground-truth.

Illustration using a 1-dimensional problem

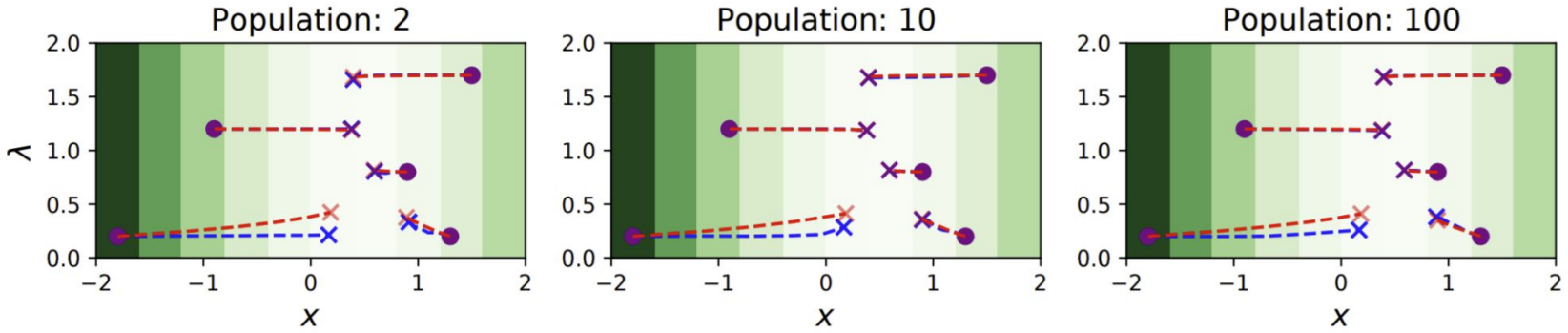

Figure 3: Trajectories of parameters x, λ when following $\partial f_T / \partial x$ and $\partial f_V / \partial \lambda$ using SGD for 5 random starting positions. Comparison of trajectories using EvoGrad estimated (blue) or ground-truth (red) hypergradient. The initial position is marked with a circle, and the final position after 5 steps is marked with a cross. The shading is validation loss $f_V(x)$.

Rotation transformation

- Goal: train a model that correctly classifies rotated images
  - Training data is originally unrotated
- How: meta-learn rotation angle

Table 3: Rotation transformation learning. The goal is to accurately classify MNIST test images rotated by 30° degrees compared to the training set orientation. Test accuracies (%) of a baseline model, and one whose training set has been rotated by the EvoGrad's meta-learned rotation, and associated EvoGrad rotation estimate ($^\circ$). Accuracy for rotation matched train/test sets is 98.40%.

True Rotation	Baseline Acc.	EvoGrad Acc.	EvoGrad Rotation Est.
30°	81.79 ± 0.64	98.11 ± 0.32	$28.47^\circ \pm 5.23^\circ$

Cross-domain few-shot classification via learned feature-wise transformation

- Approach from Tseng et al., ICLR'20
- Goal: improve few-shot learning generalisation in cross-domain conditions
- How: meta-learn stochastic feature-wise transformation layers that regularize metric-based few-shot learners
- Key steps:
  - 1) Update the model with the meta-parameters on a pseudo-seen domain
  - 2) Update the meta-parameters by evaluating the model on a pseudo-unseen domain by backpropagating through the first step
- EvoGrad efficiency improvements allow scaling from ResNet10 to ResNet34 within standard 12GB GPU!

Cross-domain few-shot classification via learned feature-wise transformation

Table 4: RelationNet test accuracies (%) and 95% confidence intervals across test tasks on various unseen datasets. LFT EvoGrad can scale to ResNet34 on all tasks within 12GB GPU memory, while vanilla second-order LFT $T_1 - T_2$ cannot. We also report the results of our own rerun of the LFT approach using the official code – denoted as *our run*. EvoGrad can clearly match the accuracies obtained by the original approach that uses $T_1 - T_2$.

	Model	Approach	CUB	Cars	Places	Plantae
5-way 1-shot	ResNet10	-	44.33 ± 0.59	29.53 ± 0.45	47.76 ± 0.63	33.76 ± 0.52
		FT	44.67 ± 0.58	30.38 ± 0.47	48.40 ± 0.64	35.40 ± 0.53
		LFT $T_1 - T_2$	48.38 ± 0.63	32.21 ± 0.51	50.74 ± 0.66	35.00 ± 0.52
		LFT $T_1 - T_2$ (our run)	46.03 ± 0.60	31.50 ± 0.49	49.29 ± 0.65	36.34 ± 0.59
		LFT EvoGrad	47.39 ± 0.61	32.51 ± 0.56	50.70 ± 0.66	36.00 ± 0.56
	ResNet34	-	45.61 ± 0.59	29.54 ± 0.46	48.87 ± 0.65	35.03 ± 0.54
		FT	45.15 ± 0.59	30.28 ± 0.44	49.96 ± 0.66	35.69 ± 0.54
		LFT EvoGrad	45.97 ± 0.60	33.21 ± 0.54	50.76 ± 0.67	38.23 ± 0.58
5-way 5-shot	ResNet10	-	62.13 ± 0.74	40.64 ± 0.54	64.34 ± 0.57	46.29 ± 0.56
		FT	63.64 ± 0.77	42.24 ± 0.57	65.42 ± 0.58	47.81 ± 0.51
		LFT $T_1 - T_2$	64.99 ± 0.54	43.44 ± 0.59	67.35 ± 0.54	50.39 ± 0.52
		LFT $T_1 - T_2$ (our run)	65.94 ± 0.56	43.88 ± 0.56	65.57 ± 0.57	51.43 ± 0.55
		LFT EvoGrad	64.63 ± 0.56	42.64 ± 0.58	66.54 ± 0.57	52.92 ± 0.57
	ResNet34	-	63.33 ± 0.59	40.50 ± 0.55	64.94 ± 0.56	50.20 ± 0.55
		FT	62.48 ± 0.56	41.06 ± 0.52	64.39 ± 0.57	50.08 ± 0.55
		LFT EvoGrad	66.40 ± 0.56	44.25 ± 0.55	67.23 ± 0.56	52.47 ± 0.56

Cross-domain few-shot classification via learned feature-wise transformation

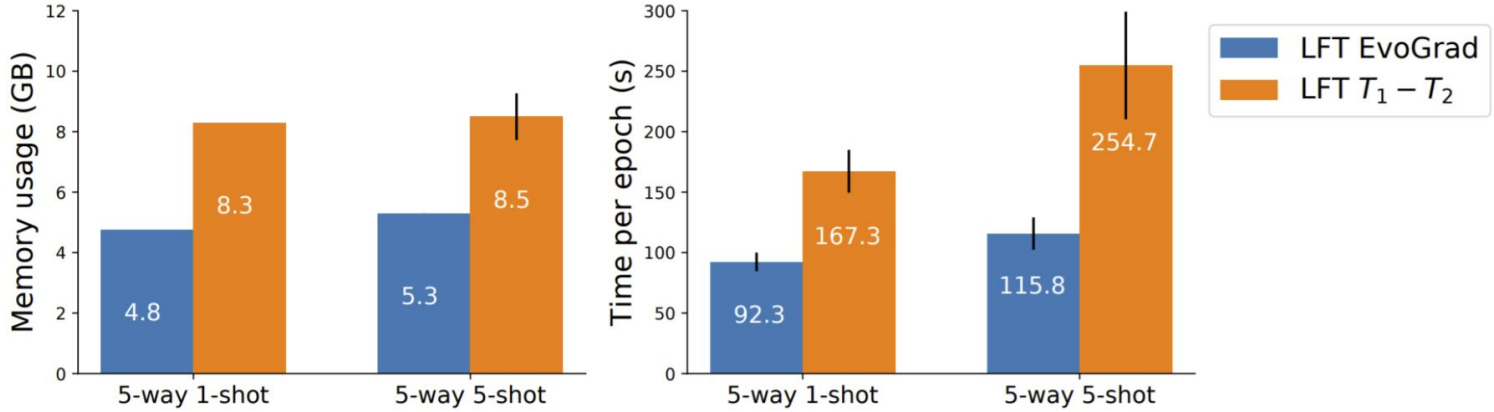

Figure 4: Cross-domain few-shot learning with LFT: analysis of memory and time efficiency of EvoGrad vs standard second-order $T_1 - T_2$ approach. Mean and standard deviation reported across experiments with different test datasets. EvoGrad is significantly more efficient in terms of both memory usage and time per epoch.

Noisy label learning with MetaWeightNet

- Approach from Shu et al., NeurIPS'19
- Goal: improve robustness to training with noisy labels
- How: train an auxiliary neural network that performs instance-wise loss re-weighting on the training set

Noisy label learning with MetaWeightNet

Table 5: Test accuracies (%) for Meta-Weight-Net label noise experiments with ResNet-32 – means and standard deviations across 5 repetitions for the original second-order algorithm vs EvoGrad. EvoGrad is able to match or even exceed the accuracies obtained by the original MWN approach.

Dataset	Noise rate	Baseline	MWN $T_1 - T_2$	MWN $T_1 - T_2$ (our run)	MWN EvoGrad
CIFAR-10	0%	92.89 ± 0.32	92.04 ± 0.15	91.10 ± 0.19	92.02 ± 0.31
	20%	76.83 ± 2.30	90.33 ± 0.61	89.31 ± 0.40	89.86 ± 0.64
	40%	70.77 ± 2.31	87.54 ± 0.23	85.90 ± 0.45	87.74 ± 0.54
CIFAR-100	0%	70.50 ± 0.12	70.11 ± 0.33	68.42 ± 0.36	69.16 ± 0.49
	20%	50.86 ± 0.27	64.22 ± 0.28	63.43 ± 0.43	64.05 ± 0.63
	40%	43.01 ± 1.16	58.64 ± 0.47	56.54 ± 0.90	57.44 ± 1.25

Noisy label learning with MetaWeightNet

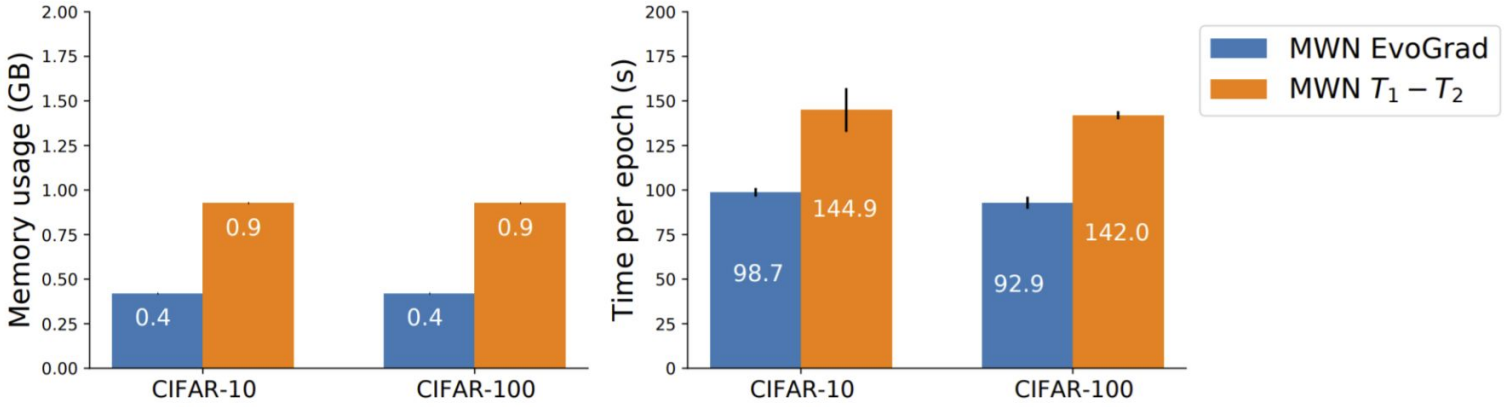

Figure 5: Analysis of memory and time cost of MWN EvoGrad vs the original second-order MWN, showing significant efficiency improvements by EvoGrad. Mean and standard deviation is reported across 5 repetitions of 40% label noise problem.

Scalability analysis

- Modify the number of filters in the base model (MetaWeightNet application)

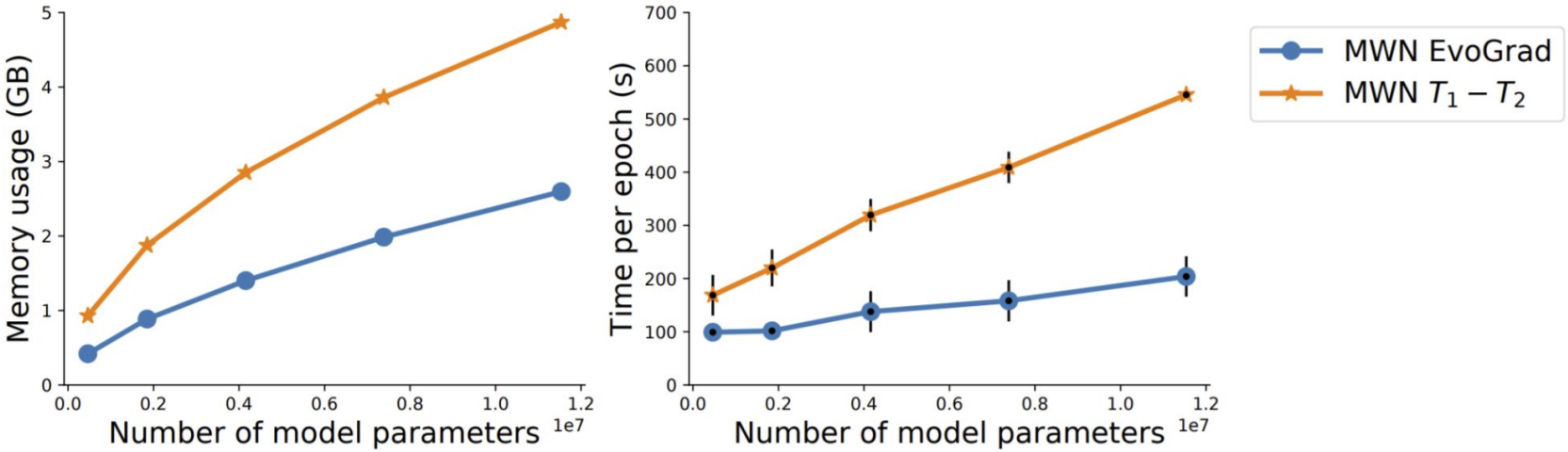

Figure 6: Memory and time scaling of MWN EvoGrad vs original second-order Meta-Weight-Net. Efficiency margins of EvoGrad are larger for larger models.

Low-resource cross-lingual learning with MetaXL

- Approach from Xia et al., NAACL'21
- Goal: more efficient transfer from source language to low-resource target language
- How: meta-learn how to transform representations using a representation transformation network
- Selected task: named entity recognition (NER) with English source language

MetaXL performance

Table 6: Test F1 score in % for named entity recognition task. English source language. The first two rows are taken from the MetaXL paper, while our own runs are in the following rows. EvoGrad clearly matches and even surpasses the performance of $T_1 - T_2$ baseline. Joint-training (JT) represents a simple non-meta-learning baseline approach.

Method	qu	cdo	ilo	xmf	mhr	mi	tk	gn	Average
JT	66.10	55.83	80.77	69.32	71.11	82.29	61.61	65.44	69.06
MetaXL $T_1 - T_2$	68.67	55.97	77.57	73.73	68.16	88.56	66.99	69.37	71.13
JT (our run)	59.75	49.19	79.43	68.85	68.42	89.94	61.90	69.44	68.37
MetaXL $T_1 - T_2$ (our run)	65.29	56.33	76.50	67.24	71.17	89.41	66.67	64.11	69.59
MetaXL EvoGrad	71.00	57.02	85.99	70.40	65.45	88.12	66.97	70.91	71.98

MetaXL Efficiency

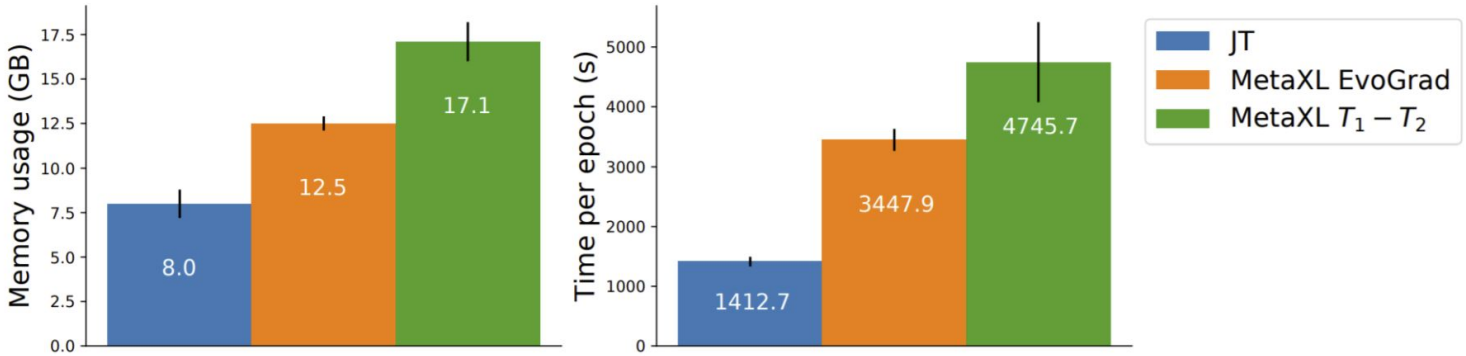

Figure 7: Analysis of memory and time cost of MetaXL EvoGrad vs the original second-order MetaXL, in the context of a simple joint-training (JT) baseline. EvoGrad consumes significantly less memory than $T_1 - T_2$ and is faster. Mean and standard deviation is calculated over the 8 different target languages.

Summary

- Efficient first-order method for gradient-based meta-learning and hyperparameter optimization
- Significant improvements in runtime and memory, while achieving similar performance as existing methods
- Practical impact shown on recent meta-learning applications from both computer vision and natural language processing

Thanks!

Code: https://github.com/ondrejbohdal/evograd