LEARNING TO LEARN VIA GRADIENT COMPONENT CORRECTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Gradient-based meta-learning algorithms require several steps of gradient descent to adapt to newly incoming tasks. This process becomes more costly as the number of samples increases. Moreover, the gradient updates suffer from several sources of noise leading to a degraded performance. In this work, we propose a meta-learning algorithm equipped with the GradiEnt Component COrrections, a GECCO cell for short, which generates a multiplicative corrective low-rank matrix which (after vectorization) corrects the estimated gradients. GECCO contains a simple decoder-like network with learnable parameters, an attention module and a so-called context input parameter. The context parameter of GECCO is updated to generate a low-rank corrective term for the network gradients. As a result, meta-learning requires only a few of gradient updates to absorb new task (often, a single update is sufficient in the few-shot scenario). While previous approaches address this problem by altering the learning rates, factorising network parameters or directly learning feature corrections from features and/or gradients, GECCO is an off-the-shelf generator-like unit that performs element-wise gradient corrections without the need to 'observe' the features and/or the gradients directly. We show that our GECCO (i) accelerates learning, (ii) performs robust corrections of the gradients corrupted by a noise, and (iii) leads to notable improvements over existing gradient-based meta-learning algorithms.

1 INTRODUCTION

Approaches (Finn et al., 2017; Zintgraf et al., 2019; Rusu et al., 2019; Ravi & Larochelle, 2017; Antoniou et al., 2019; Rajeswaran et al., 2019) belong to the family of so-called gradient-based meta-learning algorithms which are popular due to their ability to update a generic model to specific incoming tasks. In the absence of abundant data (*e.g.*, low-shot learning) the gradient information is often noisy which degrades the performance of meta-learners. Due to low sample nature of this problem, the noise further intensifies if higher-order solvers are are required *e.g.*, Hessian.

In this work, we introduce a new meta-learning algorithm that benefits from the GradiEnt Component COrrections (a GECCO cell for short) to overcome the impurity of the gradient information in low-shot and/or dynamic regimes.

In the literature, a large family of meta-learning solutions can be understood as models that, in one way or another, correct the gradient information. Algorithms such as Meta-SGD (Li et al., 2017), MAML++ (Antoniou et al., 2019)) and LEO (Rusu et al., 2019) adaptively alter the step-size of the gradient updates (by adjusting the learning rate) to attenuate the effect of the noise.

However, Wu et al. (2018) have noted that in some cases, so-called *short-horizon bias* may exist in the real setting so that the models trained on short horizons will fail to generalize to longer horizons. Moreover, numerous prior works show that the performance and training stability depend on learning rates. This problem is highly related to so-called gradient steps for which a long unrolling of the gradient updates requires tuning its learning rates for the best performance.

Gradient steps. When adapting to specific tasks, MAML employs early stopping after a few of gradient steps to implicitly regularize the solution. However, MAML also suffers from practical problems concerning gradient updates for large neural networks in meta-learning setting especially in the few-shot scenario, as discussed in (Rusu et al., 2019). Specifically, updates of high-dimensional pa-

rameters suffer from overfitting due to a few samples being available for training. Our experiments confirm this observation as discussed in Section 4. Another problem emerges when a meta-learning approach is employed to deal with the multi-shot scenario as described in (Flennerhag et al., 2019). The more data samples are provided, the more parameter updates are required for a network to converge to a good solution. Thus, a rapid adaptation of neural networks to a new task is desirable but difficult to achieve. Similarly, it is also expensive to calculate the Hessian matrix (which typically helps to converge to better solutions) with long steps for every task as is discussed in Rajeswaran et al. (2019).

When a meta-learner is employed to learn in dynamic environments, the existence of corrupted data such as noisy labels or noisy signals makes the meta-learner vulnerable to such a noise. For the second-order solvers, the noise further affects the calculation of the Hessian matrix thus interfering with the curvature of the objective. In turns, this leads to a performance degradation of the model. In this paper, we demonstrate that existing algorithms are susceptible to noisy gradients which affects their performance. To address this issue, we propose a GECCO cell which corrects such noisy gradients. Our evaluations shows that the performance of meta-learning algorithms drops heavily when gradients are affected by the noise or corrupted. Therefore, a model that is stable to the gradient noise is also expected to exploit the meta-knowledge from the incoming tasks better.

In this paper, we develop a meta-learning algorithm with a GECCO cell which corrects the gradients during training so that the parameters of a neural network are adapted rapidly and robustly to new tasks. While prior works estimate transformations applied to gradients implicitly in the form of additional layers or so-called modulation of deep neural networks (Zintgraf et al., 2019; Lee & Choi, 2018; Rusu et al., 2019), our algorithm transforms the gradients explicitly by the multiplicative gradient corrections. The GECCO corrector has the ability to learn by adapting itself to the loss by adapting so-called context vector, it imposes low-rank correction patterns, and does not require direct gradient and/or feature vector inputs, making it easily applicable to many existing meta-learners. Our method converges fast and outperforms other meta-learners. Our contributions are threefold:

i. We propose a novel meta-learning algorithm with GECCO cells (gradient corrections) to tackle the classification, regression, and reinforcement learning.

ii. We perform numerous experiments to show that our meta-learner achieves a good performance with a mere 1-step in a few-shot scenario, *etc*. Meta-learning with GECCO also achieves a good performance for deeper networks (easier to overfit) and higher-shot scenarios.

iii. We provide an additional study by injecting various levels of the Gaussian noise to corrupt gradients. We show that meta-learning with GECCO is able to recover from such a noise while other methods have a low tolerance to such corruptions.

2 Related Work

For the deep learning regime, meta-learning has progressed significantly and provided a framework that can adapt to new tasks rapidly. One of the solutions in meta-learning is to optimize the model parameters and hyperparameters in a meta level through gradient updates. Recent meta-learning methods include (Balcan et al., 2019; Grant et al., 2018). For deep neural networks regime, several gradient-based methods learn the update functions (Ravi & Larochelle, 2017; Andrychowicz et al., 2016; Chen et al., 2017). These methods have additional parameters to learn the update functions, while Finn et al. (2017) avoids additional model parameters. Our method lies in between these two ends in that GECCO learns the update functions of the gradients with two levels (meta and task objectives) as in MAML (Finn et al., 2017).

Currently, a wide variety of meta-learners are available which are an extension of a meta-learning concept in MAML (Finn et al., 2017). As a first-order version of MAML, Reptile (Nichol et al., 2018) has a direct update from initial parameters to the updated parameters in the last iteration. CAVIA (Zintgraf et al., 2019) and LEO (Rusu et al. (2019)) deliver cheaper solutions that generate the modulation and additional layers for the main networks. To boost the performance, MAML++ (Antoniou et al., 2019) appleis weights on the loss functions in the inner-loop for meta-optimization. Furthermore, Meta-SGD (Li et al., 2017) is analogous to MAML++ but it reweights the gradients in the inner-loop.

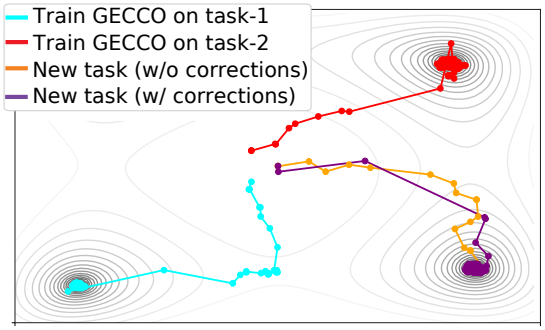


Figure 1: An illustration of GECCO corrections. Both axes represent two parameters. GECCO not only corrects the gradients but also accelerates the optimization process.

Another perspective on meta-learning is that neural networks can self-modify according to a given task. MAML (Finn et al., 2017), Reptile (Nichol et al., 2018), LEAP (Flennerhag et al., 2019) update the whole networks based on the gradients from a given task. However, some approaches update the networks partially such as fast and slow networks (Munkhdalai & Yu, 2017). This work is also aligned with (M)T-Nets (Lee & Choi, 2018) where only several layers are updated and the rest is fixed. These approaches have a lesser computational cost than updating whole model parameters.

Our work is also close to neural networks that can generate other networks with larger size. One seminal work is HyperNetworks (Ha et al., 2017) that generates bigger neural networks from small neural networks. Another work is the optimization technique that generates the gradients such that neural networks do not have to wait for backpropagation from following layers. Jaderberg et al. (2017) proposes synthetic gradients generated from small networks to produce gradients. These methods can be viewed as a decoding process that produces big output vectors from small input vectors.

3 Proposed Method

In this section, we introduce our meta-learner (*ie*., GECCO) to accelerate learning process for few- and multi-shot classification, regression, and reinforcement learning. Before delving into details, we recall that the objective in meta-learning is to **1.** achieve rapid convergence for new tasks (task-level) and **2.** to generalize beyond seen tasks (meta-level). To achieve the meta-learning capability, a common approach is to design models that can learn from limited data using the concept of episodic training (Vinyals et al., 2016; Santoro et al., 2016). There, a model is presented with a set of tasks (*e.g*., image classification), where for each task limited data is available.

GECCO is a gradient-based meta-learner. The core idea of gradient-based meta-learners is to adapt the model in hand by performing a few steps of the gradient update. Assuming that the model is initialized properly, one can expect that a few iterations of the gradient descent, even with limited data, to lead to a well-adapted model. Our GECCO algorithm learns a good initialization point along a gradient correction term to expedite gradient-based meta-learning.

To put the discussion into context, we first provide a brief overview of the MAML (Finn et al., 2017) algorithm. Let $\mathcal{D}_\tau^{trn}$ and $\mathcal{D}_\tau^{tst}$ be the training and the validation sets of a given task $\tau \sim p(\mathcal{T})$, respectively. We assume that $\mathcal{D} \coloneqq \{\boldsymbol{x}_i, y_i\}_{i=1}^{|\mathcal{D}|}, \boldsymbol{x}_i \in \mathcal{X},\ y_i \in \mathcal{Y}$ for some small $|\mathcal{D}|$. Furthermore, let $h : \mathcal{X} \times \mathbb{R}^n \to \mathcal{Y}$ be the functionality of the model of interest, parameterized by $\theta \in \mathbb{R}^n$. The MAML seeks a universal initialization θ^* by minimizing:

$$\mathcal{L}^{\text{MAML}}(\theta^*) \coloneqq \sum_{\tau \sim P(\mathcal{T})} \mathcal{L}\left(D_\tau^{\text{tst}}, \theta^* - \alpha \sum_{k=0}^{K-1} \nabla \mathcal{L}\big(D_\tau^{\text{trn}}, \theta_\tau^k\big)\right). \tag{1}$$

Here, $\theta_\tau^k = \theta_\tau^{k-1} - \alpha \nabla \mathcal{L}\big(D_\tau^{\text{trn}}, \theta_\tau^{k-1}\big)$ with $\theta_\tau^0 = \theta^*$. The loss terms are:

$$\mathcal{L}\big(D_\tau^{\text{trn}}, \theta\big) \coloneqq \mathbb{E}_{\boldsymbol{x}, y \sim \mathcal{D}_\tau^{\text{trn}}}\big[\ell(h(\boldsymbol{x}, \theta), y)\big],$$
$$\mathcal{L}\big(D_\tau^{\text{tst}}, \theta\big) \coloneqq \mathbb{E}_{\boldsymbol{x}, y \sim \mathcal{D}_\tau^{\text{tst}}}\big[\ell(h(\boldsymbol{x}, \theta), y)\big],$$

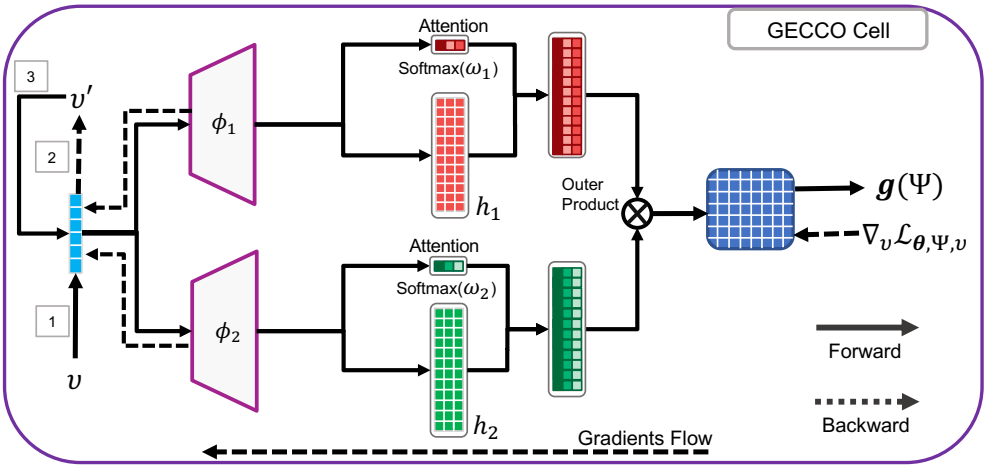


Figure 2: A GECCO cell.

where, $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ is the loss of the model h. Intuitively, given a task τ, the MAML starts from θ^* and performs K gradient updates on D_τ^{trn} to attain the adapted parameters θ_τ^K (this is called the inner-loop updates). Then it uses D_τ^{tst} and θ_τ^K (which is dependent on θ^*) to improve the universal initialization point θ^* (this is called the outer-loop update).

Remark 1. *We stress that the outer-loop in MAML requires higher-order derivatives to update the initial parameters. This, in principle, increases the computational complexity and memory footprint of MAML, limiting its deployment alongside large or very deep models. The first-order MAML (FOMAML)(Finn et al., 2017) is a possible remedy as FOMAML only uses the gradient of the last update in its computations. GECCO, while not entirely resolves the dependency on the higher order gradients, requires less gradient updates and hence partially addresses the compute/memory issues.*

We generalize the MAML model according to the following loss:

$$\mathcal{L}^{\text{GECCO}}(\theta^*, \Psi) := \sum_{\tau \sim P(\mathcal{T})} \mathcal{L}\Big(D_\tau^{\text{tst}}, \theta^* - \alpha \sum_{k=0}^{K-1} \boldsymbol{g}_\tau^k(\Psi) \odot \nabla \mathcal{L}\big(D_\tau^{\text{trn}}, \theta_\tau^k\big) \Big). \tag{2}$$

Remark 2. *The GECCO update can be viewed as an element-wise correction to the gradient vector. This is analogous to a large list of quasi-Newton optimizers such as ADAM (Kingma & Ba, 2015) or even the natural gradient descent method of Amari (Amari, 1998). Compared to the SGD optimizer, quasi-Newton methods often enjoy rapid convergence. Replacing SGD with quasi-Newton techniques in the inner-loop of MAML may be beneficial. However, this adds another layer of difficulty to the computation of gradient in the outer-loop. One can also understand the GECCO updating scheme as a generalization to meta-learners that adaptively alter the learning rate of the SGD* (e.g., *(Antoniou et al., 2019; Li et al., 2017; Rusu et al., 2019)).*

In essence, what we would like to achieve by minimizing Eq. 2 is to jointly learn the universal initialization point θ^* and the gradient corrections $\boldsymbol{g}_\tau^k(\Psi)$ to enrich adaptability of the meta-learner. In what follows and without loss of generality, we assume $n = n_1 \times n_2$ and elaborate on how the gradient correction function $\boldsymbol{g}_\tau^k(\Psi) \in \mathbb{R}^n$ for a given task τ is obtained by a GECCO cell.

For reasons become clear shortly, a GECCO cell makes use of a context vector $\nu \in \mathbb{R}^d$ to generate the correction vector $\boldsymbol{g}_\tau^k(\Psi)$. In doing so, the context vector ν is first processed by two sister modules ϕ_1 and ϕ_2. This generates,

$$(\boldsymbol{\omega}_1, \boldsymbol{h}_1) = \phi_1(\boldsymbol{\nu}), \quad \boldsymbol{\omega}_1 \in \mathbb{R}^u, \boldsymbol{h}_1 \in \mathbb{R}^{n_1 \times u},$$
$$(\boldsymbol{\omega}_2, \boldsymbol{h}_2) = \phi_2(\boldsymbol{\nu}), \quad \boldsymbol{\omega}_2 \in \mathbb{R}^u, \boldsymbol{h}_2 \in \mathbb{R}^{n_2 \times u}.$$

The output of the GECCO cell is then obtained as[1]

$$\boldsymbol{g}(\Psi) = \text{Vec}\big(\big(\boldsymbol{h}_1 \text{Softmax}(\boldsymbol{\omega}_1)\big)\big(\boldsymbol{h}_2 \text{Softmax}(\boldsymbol{\omega}_2)\big)^\top\big). \tag{3}$$

[1] We have slightly abused the notation here in the sense that $\boldsymbol{h}\text{Softmax}(\boldsymbol{\omega})$ means that the softmax output is applied element-wise over the columns of h.

The form in Eq. equation 3 uses a low-rank approximation to generate the correction term. This essentially enables us to scale up GECCO cells to very large networks and also acts as a regularization (see more details in Appendix B).

With the above, the only remaining piece of the GECCO algorithm is the way the context vector is created. In doing so, we first reset $\nu = \vec{0}$ and generate an intermediate $\boldsymbol{g}_\tau^k(\Psi)$. We then update ν as

$$\boldsymbol{\nu}' \leftarrow -\nabla_{\boldsymbol{\nu}} \mathcal{L}\big(\theta^i - \alpha \boldsymbol{g}_\tau(\Psi) \odot \nabla_{\theta^i} \mathcal{L}(\theta^i)\big)|_{\nu=\mathbf{0}}. \tag{4}$$

Remark 3. *GECCO uses a context vector $\boldsymbol{\nu}$ to generate the corrections. This enables us to lower the computational complexity even further. One can also expect the gradient field to comply with a low-dimensional structure (as a result of smoothness of the gradient updates). As such, enforcing the context vector to be low-dimensional may implicitly contribute to capturing the geometry of the gradient field.*

Algorithm 1 provides details on how the parameters of the GECCO Ψ and the main network θ should be updated. This algorithm is chiefly designed for classification and regression tasks. For the reinforcement learning, the form of GECCO is provided in Appendix D.

Algorithm 1 Train GECCO

Require: θ, Ψ, α, $p(\tau)$
θ, $\Psi \leftarrow$ Random initialization
while not done **do**
  Sample $\tau_1 \dots \tau_B$ from $p(\mathcal{T})$
  for b in $\{1, ..., B\}$ **do**
    $\theta^0 \leftarrow \theta$
    $\mathcal{D}_\tau^{trn}, \mathcal{D}_\tau^{tst}$ from $\mathcal{T}_b$
    for i in $\{0, ..., K-1\}$ **do**
      Reset $\boldsymbol{\nu}$
      Compute $\nabla_{\boldsymbol{\theta}^i} \mathcal{L}(\boldsymbol{\theta}^i)$
      Compute $\boldsymbol{\nu}'$ using Eq. 4
      $\theta^{i+1} \leftarrow \theta^i - \alpha \boldsymbol{g}_\tau^k(\Psi, \boldsymbol{\nu}') \nabla_{\theta^i} \mathcal{L}(\theta^i)$
    end for
  end for
  $\theta \leftarrow$ OptimizerStep($\mathcal{D}_\tau^{tst}, \theta^K$)
  $\Psi \leftarrow$ OptimizerStep($\mathcal{D}_\tau^{tst}, \Psi$)
end while

Remark 4. *In contrast to (M)T-Nets (Lee & Choi, 2018) and natural neural networks (Desjardins et al., 2015) that insert additional layers for gradient projection, GECCO directly produces the prediction for a preconditioning matrix on the gradients. The main benefit in a meta-learning setup is that we avoid altering the main network to perform adaptation.*

Remark 5. *In practice and to lower the computational complexity, one can make use of a set of m distinct GECCO cells $\Psi = \{\psi_1 \dots \psi_m\}$, each acting on and optimizing a layer of the network (see Fig. 2 for a conceptual diagram). A GECCO cell can be attached to any structure of neural networks including fully-connected and convolutional layers. This design requires no changes to the main networks and brings more flexibility such that the corrections are applied only to selected layers. A detail description on how to implement to both structures is provided in Appendix A.*

4 Experiments

4.1 Classification

Multi-shot classification. The Omniglot dataset (Lake et al., 2015) contains 1623 characters from 50 different alphabets. This experiment follows the setting in (Flennerhag et al., 2019) where there are multi-shot and multi-tasks to evaluate the capacity of learning rapidly. The splits are 40 alphabets

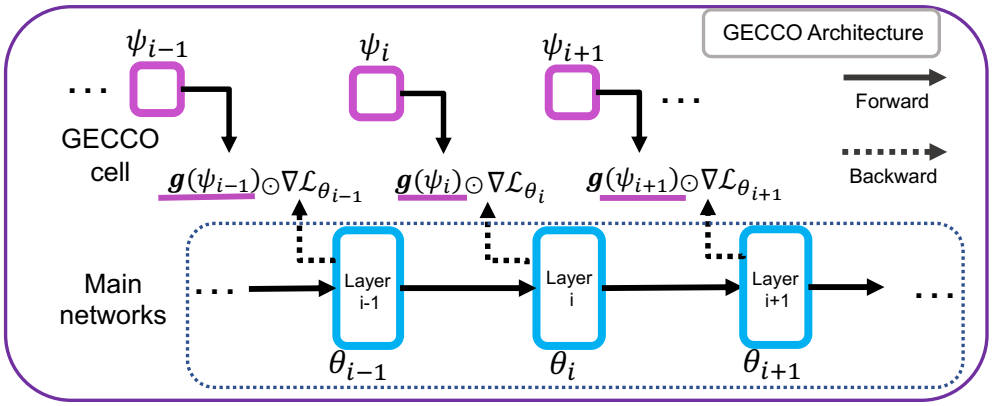
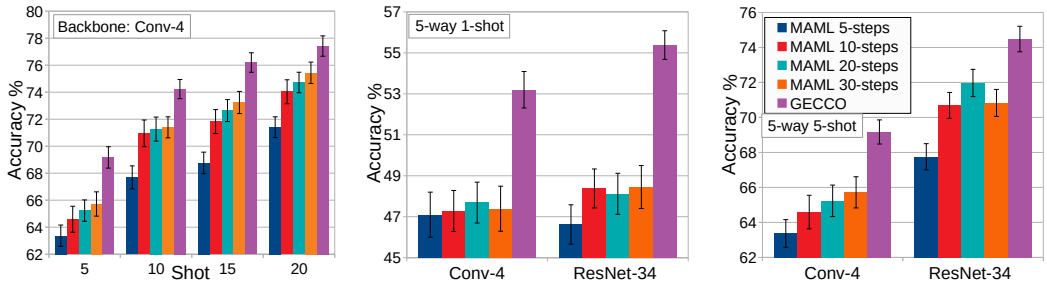


Figure 3: Every layer is equipped with a GECCO cell to correct incoming gradients from the main networks.



(a) (b) (c)

Figure 4: The performance of MAML with 5, 10, 20, and 30 steps in the inner-loop and GECCO with various shots (a) and deeper networks (b), (c). GECCO only applies 1-step to achieve superior performances for various shots and backbones.

for training and the rest are for evaluation. Here, we compare the performance and the rate of convergence on the Omniglot dataset (Lake et al., 2015) with 20-way, 100 samples per batch, and 25 tasks in total. There are two evaluations in this setting: the learning capability and the performance on the evaluation set. These empirical results show that our method needs a less number of steps to achieve low training loss compared to LEAP (Flennerhag et al., 2019), Reptile (Nichol et al., 2018), and FOMAML (Finn et al., 2017) as shown in Fig. 6. For more details, see Appendix A.

Few-shot classification. We evaluate GECCO on the few-shot classification benchmark with the *mini*-ImageNet dataset by Ravi & Larochelle (2017). The *mini*-ImageNet is the subset of the ImageNet dataset (Krizhevsky et al., 2012) with 64, 16, and 20 classes for training, validation, and testing, respectively. We follow a common protocol for 5-way 5-shot and 1-shot with 600 tasks for testing. The image size is downsampled to 84×84. All training is performed with episodic training without any augmentation. We evaluate on two convolutional neural network (CNN) backbones: 4-convolutional layers (Conv-4) with the same structure as in (Snell et al., 2017; Finn et al., 2017) and WideResNet 28-10 (WRN-28-10) (Zagoruyko & Komodakis, 2016) as stated in (Qiao et al., 2018; Rusu et al., 2019). For WRN-28-10, we follow the strategy to use pre-trained networks as mentioned in (Qiao et al., 2018; Rusu et al., 2019) then GECCO is trained with episodic training. We use these two backbones and training protocols to fairly compare the performance to existing meta-learning methods. In our implementation, GECCO cells are applied only on the last two convolutional layers of the main networks. The model parameters of the main networks and GECCO are optimized with an Adam optimizer (Kingma & Ba, 2015) and the learning rate is set to 10^{-3}. The learning rate is cut to half for every 10K episodes. The size of $\boldsymbol{\nu}$ and α are 300 and 0.1, respectively for all experiments on the *mini*-ImageNet.

Model	Backbone	1-shot	5-shot
Meta-Learner LSTM (Ravi & Larochelle, 2017)	Conv-4	43.44 ± 0.77	60.60 ± 0.71
MAML (64)$^{\#}$ (Finn et al., 2017)	Conv-4	47.89 ± 1.20	64.59 ± 0.88
Reptile (Nichol et al., 2018)	Conv-4	49.97 ± 0.32	65.99 ± 0.58
Meta-SGD ((Li et al., 2017)	Conv-4	50.50 ± 1.90	64.00 ± 0.90
R2-D2 (Bertinetto et al., 2019)	Conv-4	48.70 ± 0.60	65.50 ± 0.60
MT-Nets (Lee & Choi, 2018)	Conv-4	51.70 ± 1.84	–
CAVIA (512) (Zintgraf et al., 2019)	Conv-4	51.82 ± 0.65	65.85 ± 0.55
GECCO (64 / 1-step)	Conv-4	$\mathbf{53.20 \pm 0.86}$	$\mathbf{69.17 \pm 0.69}$
Qiao et al. (Qiao et al., 2018)	WRN-28-10	59.60 ± 0.41	73.74 ± 0.19
LEO (Rusu et al., 2019) (fine-tuning)	WRN-28-10	61.76 ± 0.08	77.59 ± 0.12
GECCO (no fine-tuning / 1-step)	WRN-28-10	$\mathbf{62.58 \pm 0.45}$	$\mathbf{78.16 \pm 0.33}$

Table 1: Comparison with existing methods for few-shot classification with various backbones. The reported results are evaluated for 5-way with 1 and 5 shot classification on the *mini*-ImageNet dataset. $^{\#}$ is our reimplementation.

On this benchmark, GECCO can outerperform existing few-shot methods on various backbones: Conv-4 and WRN-28-10 as presented in Table 1. Using Conv-4, GECCO only needs 1-step and 64 filters per layer to outperform CAVIA that employs 5-steps and 512 filters by around 1.4% and 3.3% for 5-way 1-shot and 5-shot, respectively. Furthermore, GECCO with WRN-28-10 can also perform better than the works by Qiao et al. (2018) and Rusu et al. (2019). GECCO needs only 1-step in the inner-loop compared to other meta-learning methods that need more than 1-step to achieve good performances.

Number of steps. We also investigate on deeper networks using ResNet (He et al., 2016) and higher number of shots to observe the relationship between the number of step and the performance. Here, we reimplement MAML and use the first-order method because the memory load for second-order method is enormous for very deep networks. Note that, the number of step is applied for both training and testing stages for a 5-way classification. Training for Conv-4 and ResNet-34 is performed over 50K and 100K episodes, respectively. For ResNet-34, data augmentation and image size of 224×224 are applied following the settings in (Chen et al., 2019). Fig 4 shows that MAML (Finn et al., 2017) needs more steps to achieve better performances for higher shots and deeper networks.

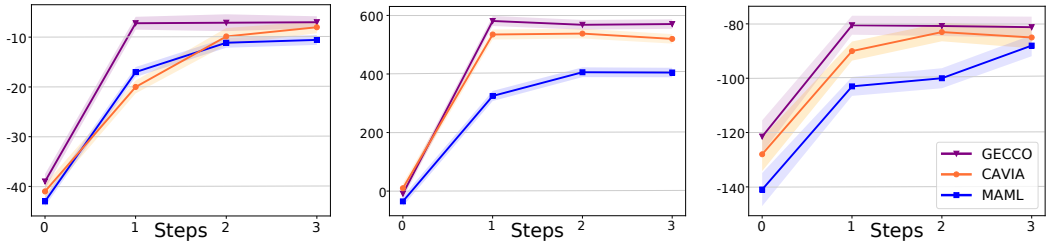
For deeper networks, we can observe that the performance gap for 5-steps and 30-steps (the highest accuracy) in Conv-4 (64 filters) is 2.5% but the performance gap reaches 4% on ResNet-34. In deeper networks, GECCO outperforms MAML by $\sim$6.5% and $\sim$2.5% for 5-way 1-shot and 5-shot, respectively. For higher shots, MAML with more additional steps also shows the improvement. In 5-shot, the performance gap is about 2.5% between 5-steps and 30-steps but the performance gap increases up to 4% in 20-shot. Furthermore, GECCO can outperform MAML 30-steps by 2% in the 20-shot classification. We conjecture that GECCO can achieve a good performance in 1-step for both cases because the corrections scale the gradients to reach a minimum rapidly and impose a regularization implicitly.

4.2 Regression

Image completion. Meta-learning is a general algorithm that can also be applied for regression tasks. The task for image regression in this experiment is adopted from (Garnelo et al., 2018) using the CelebA dataset (Liu et al., 2015). The task is to complete the whole image pixels given only some pixels of images (random and ordered). The inputs are pixel locations and the models have to perform regression to approximate pixel values with 10, 100, and 1000 provided pixels. The results in Table 2 show that GECCO has lower errors for image regression tasks. We use the same setup as stated in (Zintgraf et al., 2019) with five 128 hidden layers and a 128-dimensional input vector for Ψ. For this regression task, GECCO applies to only one fully-connected layer (before the last layer). Note that, our results are only 1-step while CAVIA (Zintgraf et al., 2019) and MAML (Finn et al., 2017) use 5 gradient steps to train from provided pixels.

Model	Random Pixels			Ordered Pixels		
	10	**100**	**1000**	**10**	**100**	**1000**
Cond. Neural Process (Garnelo et al. (2018))	0.039	0.016	0.009	0.057	0.047	0.021
MAML (Finn et al. (2017))	0.040	0.017	0.006	0.055	0.047	0.007
CAVIA (Zintgraf et al. (2019))	0.037	0.014	0.006	0.053	0.047	0.006
GECCO (1-step)	**0.034**	**0.012**	**0.005**	**0.048**	**0.043**	**0.005**

Table 2: Error rate for image completion tasks on 10, 100, and 1000 pixels on the CelebA dataset.



(a) 2D Navigation (b) Half-Cheetah Dir (c) Half-Cheetah Vel

Figure 5: Reinforcement learning results on 2D navigation, half-cheetah direction, and velocity.

4.3 Reinforcement Learning

2D Navigation. In this experiment, we evaluate GECCO on 2D-Navigation tasks from Finn et al. (2017). Every task contains a randomly chosen goal position where an agent has to move to this position. The goal of this task is to adapt the policy of an agent quickly such that it can maximize the (negative) rewards from the given tasks. The goals of this navigation are within the range $[-0.5, 0.5]$ and the actions are clipped within $[-0.1, 0.1]$. In total, 20 trajectories are used for one gradient update. We use the same networks as in (Zintgraf et al., 2019) with two-layer networks, 100 hidden units, and a ReLU activation function. In 1-step, GECCO can achieve rewards around -8 while CAVIA and MAML are far below -15.

Locomotion. We evaluate our method with half-cheetah locomotion tasks from the MuJoCo simulator (Todorov et al., 2012). The tasks consist of predicting the direction and the velocity. The velocity ranges between 0.0 and 2.0. Each rollout length is 200 and 20 rollouts are used per gradient step during training. GECCO reaches rewards around 590 with only 1-step but CAVIA and MAML obtain rewards below 550 for half-cheetah direction tasks. Furthermore, GECCO reaches around -80 for half-cheetah velocity tasks with 1-step but CAVIA and MAML only reach around -90 and -100, respectively.

In all reinforcement learning tasks, it is shown in Fig. 5 that GECCO needs a fewer update to achieve better rewards. This shows that our method is also beneficial for non-differentiable and dynamic environments. Further details in these experiments are provided in Appendix D.

4.4 How robust is GECCO to noisy gradients?

Several methods *e.g.*, CAVIA (Zintgraf et al., 2019), and T-Net (Lee & Choi, 2018) use a modulation to the parameters or an additional layer as a transformation. These methods receives the gradients directly from the main networks. However, we show empirically that these approaches have a significant drawback when corrupted gradients exist. To evaluate the robustness of the method, we conduct experiments where true gradients $\nabla\mathcal{L}(\boldsymbol{\theta})$ is corrupted by additive Gaussian noise $\boldsymbol{\eta}$ in the inner-loop. It is empirically shown in Fig. 7 that a direct task adaptation approach to the model parameters fails miserably when additive noise exists on the gradients. Our method only degrades about 10% but T-Net, CAVIA, and MAML drop by 30% and 40% for 5-way 1-shot and 5-shot, respectively. These results (see Appendix A) prove that GECCO is more robust and reduces the effect of noise.

5 CONCLUSIONS

In conclusion, this work presents a meta-learner via gradient component corrections so-called GECCO. The method has general adaptations to address a wide range of problems including classification, regression, and reinforcement learning. Empirical results show that GECCO is competitive against other existing meta-learners. Furthermore, GECCO is designed to be modular for every layer in deep neural networks, thus, it can be utilized for more interesting applications that require no structural changes to the main networks.

In practice, the usability of gradient corrections is to alleviate the problems of learning rate, gradient step, and noise. Our method is robust towards corrupted gradients while other existing methods have a low tolerance. Another benefit is that gradient corrections can accelerate the learning process of the model. As a result, GECCO can learn deeper networks and more shots with less number of steps compared to MAML.

From a formal standpoint, our future work will be investigating and analysing GECCO properties in the light of conformal divergences (Nock et al., 2016). Indeed, Eq. 2 can be seen as the gradient of a loss integrating a geometric structure as defined in (Amari, 2012; 2013; Zhang, 2004), which would make the gradient correction equivalently *tuning* the loss to the task at hand. This is an interesting avenue for future research.

REFERENCES

Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

Shun-ichi Amari. New developments of information geometry (17): Tsallis q-entropy, escort geometry, conformal geometry. In *Mathematical Sciences (suurikagaku)*, number 592, pp. 73–82. Science Company, October 2012. in japanese.

Shun-ichi Amari. New developments of information geometry (26): Information geometry of convex programming and game theory. In *Mathematical Sciences (suurikagaku)*, number 605, pp. 65–74. Science Company, November 2013. in japanese.

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in neural information processing systems*, 2016.

Antreas Antoniou, Harrison Edwards, and Amos Storkey. How to train your maml. In *International Conference on Learning Representations (ICLR)*, 2019.

Maria-Florina Balcan, Mikhail Khodak, and Ameet Talwalkar. Provable guarantees for gradient-based meta-learning. In *International Conference on Machine Learning*, pp. 424–433, 2019.

Luca Bertinetto, Joao F. Henriques, Philip Torr, and Andrea Vedaldi. Meta-learning with differentiable closed-form solvers. In *International Conference on Learning Representations*, 2019.

Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Frank Wang, and Jia-Bin Huang. A closer look at few-shot classification. In *International Conference on Learning Representations*, 2019.

Yutian Chen, Matthew W Hoffman, Sergio Gómez Colmenarejo, Misha Denil, Timothy P Lillicrap, Matt Botvinick, and Nando de Freitas. Learning to learn without gradient descent by gradient descent. In *Proceedings of the International Conference on Machine Learning*, 2017.

Guillaume Desjardins, Karen Simonyan, Razvan Pascanu, et al. Natural neural networks. In *Advances in Neural Information Processing Systems*, pp. 2071–2079, 2015.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning(ICML)*, 2017.

Sebastian Flennerhag, Pablo Garcia Moreno, Neil Lawrence, and Andreas Damianou. Transferring knowledge across learning processes. In *International Conference on Learning Representations*, 2019.

Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J Rezende, SM Eslami, and Yee Whye Teh. Neural processes. *arXiv preprint arXiv:1807.01622*, 2018.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

Erin Grant, Chelsea Finn, Sergey Levine, Trevor Darrell, and Thomas Griffiths. Recasting gradient-based meta-learning as hierarchical bayes. In *International Conference on Learning Representations*, 2018.

David Ha, Andrew Dai, and Quoc V. Le. Hypernetworks. In *International Conference on Learning Representations*, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, David Silver, and Koray Kavukcuoglu. Decoupled neural interfaces using synthetic gradients. In *International Conference on Machine Learning*, pp. 1627–1635, 2017.

Diederik P. Kingma and Jimmy L. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations(ICLR)*, 2015.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems(NIPS)*, 2012.

Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

Yoonho Lee and Seungjin Choi. Gradient-based meta-learning with learned layerwise metric and subspace. In *International Conference on Machine Learning*, pp. 2933–2942, 2018.

Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. Meta-sgd: Learning to learn quickly for few shot learning. *arXiv preprint arXiv:1707.09835*, 2017.

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

Tsendsuren Munkhdalai and Hong Yu. Meta networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2554–2563. JMLR. org, 2017.

Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

Richard Nock, Frank Nielsen, and Shun-ichi Amari. On conformal divergences and their population minimizers. *IEEE Trans. Information Theory*, 62(1):527–538, 2016.

Siyuan Qiao, Chenxi Liu, Wei Shen, and Alan L. Yuille. Few-shot image recognition by predicting parameters from activations. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Aravind Rajeswaran, Chelsea Finn, Sham Kakade, and Sergey Levine. Meta-learning with implicit gradients. In *Advances in Neural Information Processing Systems*, 2019.

Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *International Conference on Learning Representations(ICLR)*, 2017.

Andrei A. Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. Meta-learning with latent embedding optimization. In *International Conference on Learning Representations*, 2019.

Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pp. 1842–1850, 2016.

Jake Snell, Kevin Swersky, and Zemel Richard. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems(NIPS)*, 2017.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012.

Oriol Vinyals, Charles Blundell, Tim Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems(NIPS)*, 2016.

Yuhuai Wu, Mengye Ren, Renjie Liao, and Roger Grosse. Understanding short-horizon bias in stochastic meta-optimization. In *International Conference on Learning Representations*, 2018.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *Proceedings of the British Machine Vision Conference (BMVC)*, pp. 87.1–87.12, 2016.

Jun Zhang. Divergence function, duality, and convex analysis. 16:159–195, 2004.

Luisa Zintgraf, Kyriacos Shiarli, Vitaly Kurin, Katja Hofmann, and Shimon Whiteson. Fast context adaptation via meta-learning. In *International Conference on Machine Learning*, pp. 7693–7702, 2019.

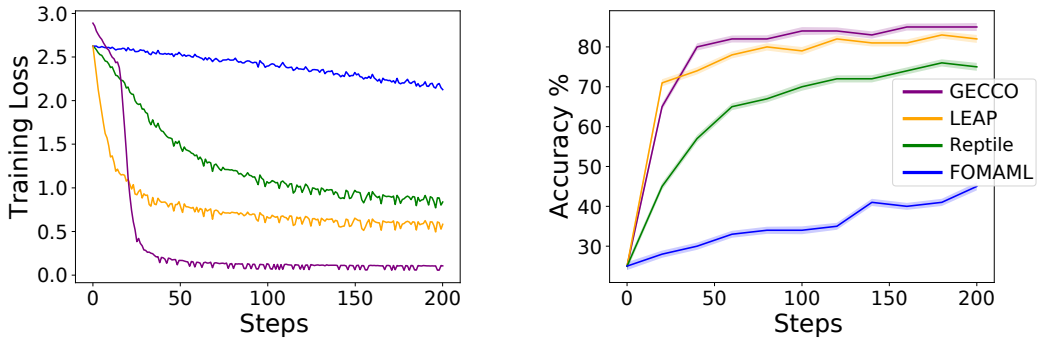
Appendices

A Additional Results

In this section, we provide more additional results and details on the Omniglot dataset and noisy gradients on the *mini*-ImageNet.

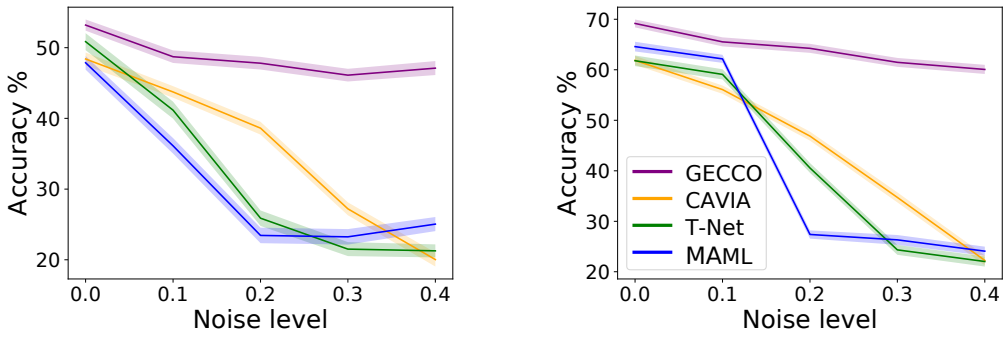
The Omniglot. This experiment is to show how fast our method can converge in multi-shot setting by Flennerhag et al. (2019). The CNN backbone used in this experiment is 4-convolutional layers with 64 filters as in Vinyals et al. (2016). Data augmentation is applied as described in Flennerhag et al. (2019) with random sampling, rotation, and cropping. For GECCO, the inner-loop learning rate (α) is set to 0.1. All of the images are downsampled to 28×28. Fig. 6 (a) shows that GECCO has a faster convergence in comparison to other methods, namely first-order MAML (FOMAML) (Finn et al. (2017)), Reptile (Nichol et al., 2018), and LEAP (Flennerhag et al., 2019). In 25 steps, GECCO can outperform LEAP (Flennerhag et al., 2019) in term of training loss and test accuracy.

Noisy Gradients. Noisy gradients is performed on the *mini*-ImageNet dataset with 5-way 5-shot and 1-shot. This experiment follows a few-shot classification setting as described in Section 4. The noise is assumed Gaussian with various levels ($\mu = 0, \sigma = \{0.1, 0.2, 0.3, 0.4\}$). As shown in Fig. 7, GECCO has a high tolerance to noisy gradients compared to the existing meta-learning methods. This capability occurs because GECCO performs corrections to reduce the effect of noise on gradients.



(a) Training loss (b) Testing accuracy

Figure 6: Training loss and testing accuracy on the Omniglot dataset in multi-shot setting.



(a) 5-way 1-shot (b) 5-way 5-shot

Figure 7: Performance comparison on the *mini*-ImageNet dataset with different noise level for 5-way with 1 and 5 samples in every episode.

B GECCO DESIGN AND IMPLEMENTATION DETAILS

In this section, we explain in more details about our proposed method for a fully-connected (FC) layer and a convolutional layer of the main networks.

FC-layer. In a FC-layer, there are a weight matrix ($\boldsymbol{W} \in \mathbb{R}^{D_1 \times D_2}$) and an additional bias ($\boldsymbol{b} \in \mathbb{R}^{D_2}$). A GECCO cell ($\psi_i$) consists of several small networks to produce a weight and a bias. A Correction term is generated from these networks ($\phi_1 \dots \phi_m$). For instance, if there is only a weight matrix then we need only two networks (ϕ_1, ϕ_2) to generate two long matrices as described in Section 3 and their outer product becomes the corrections for the gradients. Technically speaking, $\boldsymbol{\phi}_j$ generates a vector and then it is reshaped to be $\boldsymbol{\omega}_j \in \mathbb{R}^u$ and $\boldsymbol{h}_j \in \mathbb{R}^{u \times D_j}$. If a bias is also counted then there is $\boldsymbol{\phi}_3$ producing $\boldsymbol{\omega}_3 \in \mathbb{R}^u$ and $\boldsymbol{h}_3 \in \mathbb{R}^{u \times D_2}$.

Convolutional layer. Convolutional neural networks have a weight matrix ($\boldsymbol{W} \in \mathbb{R}^{D_1 \times D_2 \times c \times c}$) with $c \times c$ as a kernel size. To generate the corrections for $\boldsymbol{W}$, we also use two networks (ϕ_1, ϕ_2) as in FC-layer but we consider c for every ϕ yielding $\boldsymbol{\omega}_j \in \mathbb{R}^u$ and $\boldsymbol{h}_j \in \mathbb{R}^{(D_j c) \times u}$. Thus, the outer product can be reshaped to the dimension a weight matrix and a kernel size. For a bias term, the same approach like in FC-layer is applied.

Implementation details. Every small network ϕ_j consists of two FC-layers with sizes of $\mathbb{R}^{d \times D_j}$ and $\mathbb{R}^{D_j \times (u + uD_j)}$, respectively. As shown in Fig. 8, a ReLU activation function is inserted in between of these two layers. In all of our experiments, we use only two small networks (ϕ_1, ϕ_2) to generate the gradient corrections for a weight parameter. If a bias exists then ϕ_3 is added to a GECCO cell (ψ_i). We initialize all layers in GECCO with Xavier initialization (Glorot & Bengio, 2010).

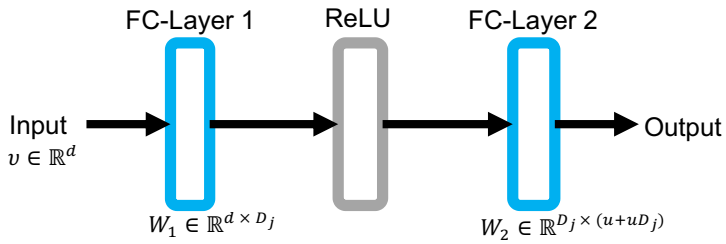


Figure 8: An architecture of a small network ϕ.

C ABLATION STUDIES

In this section, we provide ablation studies of our proposed method. The experiments are conducted to give the study on how the number of steps and the column dimension of a long matrix (u) for outer product operation may effect the performance.

Number of steps. We run over 5-steps in the inner-loop to check the performance of 5-way 5-shot and 1-shot on the *mini*-ImageNet. It is observed from Table 3 that GECCO in 1-step can achieve a good performance and running for some steps may vary the performance $\pm 1\%$ for 1-shot and 5-shot. Thus, it implies that our method is more robust to the number of step selection in the few-shot setting.

Step	1	2	3	4	5
5-way 5-shot	69.17 ± 0.7	68.80 ± 0.7	68.42 ± 0.7	68.34 ± 0.7	68.24 ± 0.7
5-way 1-shot	53.20 ± 0.9	52.66 ± 0.9	53.54 ± 0.9	52.87 ± 0.9	52.76 ± 0.9

Table 3: The results with various number of steps on the mini-ImageNet for 5-way 5-shot and 1-shot.

Dimensions of column matrices. This experiment is to show the selection of column dimension u. Table 4 shows that the performance degrades when a higher number of dimensions is applied to create gradient corrections by around 1.5% and 0.5% for 5way 5-shot and 1-shot, respectively. Empirically, 5 column dimension yields the best results. We keep the dimension of u low to avoid a large memory used to create a large matrix. In all of our experiments for classification, regression, and reinforcement learning, we use $u = 5$.

Dim. of u	1	5	10
5-way 5-shot	68.43 ± 0.7	69.16 ± 0.6	67.62 ± 0.7
5-way 1-shot	53.13 ± 0.9	53.20 ± 0.9	52.53 ± 0.9

Table 4: The results with various column dimensions of a long matrix (u) on the *mini*-ImageNet.

D REINFORCEMENT LEARNING

In this section, we provide the details how to employ GECCO for reinforcement learning (RL). We denote a state $\boldsymbol{x}$, an action $\boldsymbol{a}$, a task τ, a task distribution $p(\mathcal{T})$, a GECCO module Ψ, and a policy π_θ. We also define for sample trajectories and evaluation for H horizon as $\mathcal{D}_\tau^{trn}$ and $\mathcal{D}_\tau^{tst}$, respectively. In RL, several trajectories N are sampled to learn the policy and evaluation is done to other trajectories following the setting in Finn et al. (2017). The goal is to reach the best rewards $\mathcal{R}$ given actions and states. The policy is randomly initialized and the loss is defined as:

$$\mathcal{J}_{\tau_i}(\pi_{\boldsymbol{\theta}}) = -\mathrm{E}_{\boldsymbol{x}_t, \boldsymbol{a}_t \sim \pi_{\boldsymbol{\theta}}, \tau_i} \left[\sum_{t=1}^{H} \mathcal{R}_i(\boldsymbol{x}_t, \boldsymbol{a}_t) \right]. \tag{5}$$

Basically, the algorithm 2 has similar a similar structure to GECCO for classification and regression. The gradient correction from GECCO (Ψ) is also applied to the gradient of policy π_θ.

For all RL experiments, we run 500 meta-iterations to update the policy π_θ and GECCO parameters Ψ with $\alpha = 0.1$ in the inner-loop. The dimension of $\boldsymbol{\nu}$ is set to 5 and 50 for 2D Navigation and half-cheetah, respectively. GECCO module per layer is designed as two-layer networks with 100 hidden units and a ReLU activation function in the middle of both layers.

Algorithm 2 Train GECCO for RL

Require: π_θ, Ψ, $p(\mathcal{T})$
θ, $\Psi \leftarrow$ Random initialization
while not done **do**
  Sample $\tau_1 \dots \tau_B$ from $p(\mathcal{T})$
  for i in $\{1, ..., B\}$ **do**
    Reset $\boldsymbol{\nu}$
    Sample N trajectories $\mathcal{D}_{\tau_i}^{trn}$ using $\pi_{\boldsymbol{\theta}}$
    Compute $\nabla \mathcal{J}_{\tau_i}(\pi_{\boldsymbol{\theta}})$
    Compute $\boldsymbol{\nu}'$ on N sampled trajectories (rerun) using $\Psi, \pi_\theta, \boldsymbol{\nu}$
    Update θ' from $\nabla \mathcal{J}_{\tau_i}(\pi_\theta), \Psi, \boldsymbol{\nu}'$
    Sample trajectories $\mathcal{D}_{\tau_i}^{tst}$ using $\pi_{\theta'}$
  end for
  Update Ψ, θ using $\nabla \sum_{\tau_i \sim p(\mathcal{T})} \mathcal{J}_{\tau_i}(\pi_{\theta'}), \mathcal{D}_{\tau_i}^{tst}$
end while

E THE CELEBA QUALITATIVE RESULTS

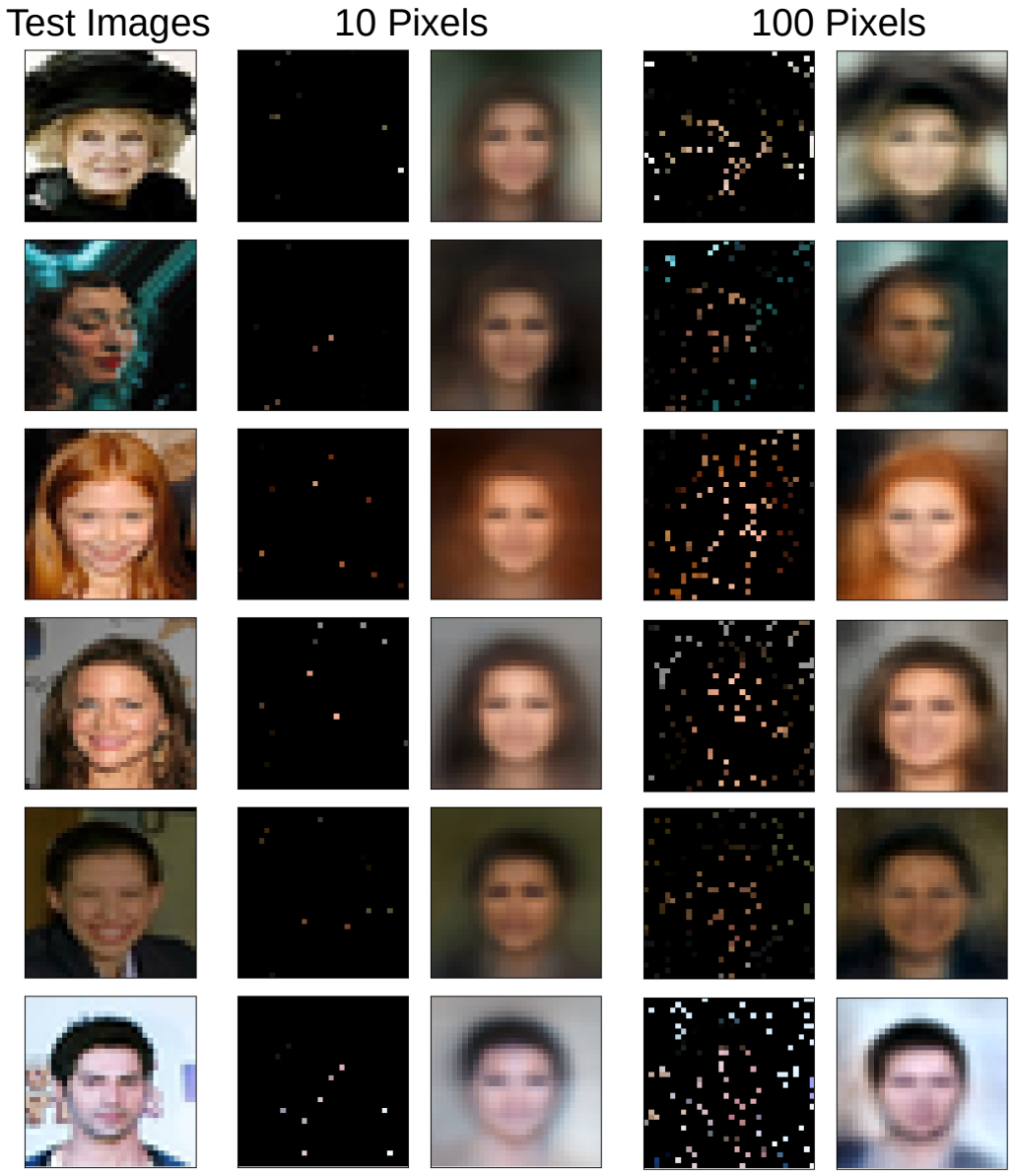


Figure 9: Image regression results with 10 and 100 pixels.