M^{2}SGD: Learning to Learn Important Weights

Nicholas I-Hsien Kuo[1], Mehrtash Harandi[2], Nicolas Fourrier[4],
Christian Walder[1,3], Gabriela Ferraro[1,3], and Hanna Suominen[1,3,5]
[1]RSCS, The Australian National University and [2]ECSE, Monash University, Australia
[3]Data61, CSIRO, Australia, [4]Pôle Universitaire Léonard de Vinci, Paris La Défense, France
[5]Department of Future Technologies, University of Turku, Turku, Finland
`u6424547@anu.edu.au, mehrtash.harandi@monash.edu, nfourrier@gmail.com`
`{christian.walder, gabriela.ferraro}@data61.csiro.au, hanna.suominen@anu.edu.au`

Abstract

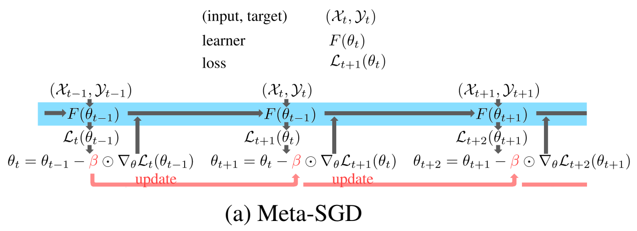
Meta-learning concerns rapid knowledge acquisition. One popular approach cast optimisation as a learning problem and it has been shown that learnt neural optimisers updated base learners more quickly than their handcrafted counterparts. In this paper, we learn an optimisation rule that sparsely updates the learner parameters and removes redundant weights. We present ***Masked Meta-SGD (M^{2}SGD)****, a neural optimiser which is not only capable of updating learners quickly, but also capable of removing 83.71% weights for ResNet20s.*
We release our codes at `https://github.com/Nic5472K/CLVISION2020_CVPR_M2SGD`.

1. Introduction

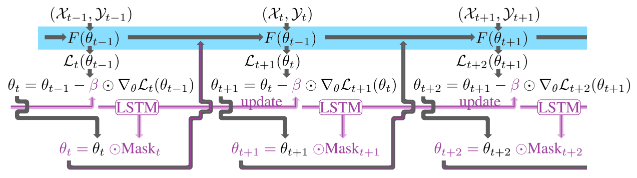
Meta-learning concerns the idea of learning quickly. This is a field that has recently seen substantial amount of advancements – there are many different styles of question formulations, and a diverse range of approaches in solving them. In this paper, we follow the *learning to learn* experimental setup as described in Andrychowicz *et al.* [1], and focus on optimisation-based meta-learning approaches.

Learning to learn (L2L) involves two networks, a base learner and an auxiliary meta-learner. The learner functions as a classifier for a task such as image recognition, and the meta-learner is trained to assist the learner to classify better. The goal is to design a meta-learner that converges the learner as quickly as possible.

In their paper, Andrychowicz *et al.* [1] explored learning to optimise. Unlike the handcrafted optimisation methods of SGD [21], momentum [18], ADAM [12], and RMSprop [24], they propose to cast the optimisation algorithm as a learning task for the meta-learner. The aim for the neural optimiser meta-learner is to directly learn the rules for updating learner parameters θ_t of time t. The neural optimiser replaces gradient descent which updates θ_t via


(input, target)
$(\mathcal{X}_t, \mathcal{Y}_t)$
learner
$F(\theta_t)$
loss
$\mathcal{L}_{t+1}(\theta_t)$
$(\mathcal{X}_{t-1}, \mathcal{Y}_{t-1})$
$(\mathcal{X}_{t}, \mathcal{Y}_{t})$
$(\mathcal{X}_{t+1}, \mathcal{Y}_{t+1})$
$F(\theta_{t-1})$
$F(\theta_{t})$
$F(\theta_{t+1})$
$\mathcal{L}_t(\theta_{t-1})$
$\mathcal{L}_{t+1}(\theta_t)$
$\mathcal{L}_{t+2}(\theta_{t+1})$
$\theta_t = \theta_{t-1} - \beta \odot \nabla_\theta \mathcal{L}_t(\theta_{t-1})$
$\theta_{t+1} = \theta_t - \beta \odot \nabla_\theta \mathcal{L}_{t+1}(\theta_t)$
$\theta_{t+2} = \theta_{t+1} - \beta \odot \nabla_\theta \mathcal{L}_{t+2}(\theta_{t+1})$
update
update

(a) Meta-SGD


$(\mathcal{X}_{t-1}, \mathcal{Y}_{t-1})$
$(\mathcal{X}_{t}, \mathcal{Y}_{t})$
$(\mathcal{X}_{t+1}, \mathcal{Y}_{t+1})$
$F(\theta_{t-1})$
$F(\theta_{t})$
$F(\theta_{t+1})$
$\mathcal{L}_t(\theta_{t-1})$
$\mathcal{L}_{t+1}(\theta_t)$
$\mathcal{L}_{t+2}(\theta_{t+1})$
$\theta_t = \theta_{t-1} - \beta \odot \nabla_\theta \mathcal{L}_t(\theta_{t-1})$
$\theta_{t+1} = \theta_t - \beta \odot \nabla_\theta \mathcal{L}_{t+1}(\theta_t)$
$\theta_{t+2} = \theta_{t+1} - \beta \odot \nabla_\theta \mathcal{L}_{t+2}(\theta_{t+1})$
LSTM
update
LSTM
update
LSTM
$\theta_t = \theta_t \odot \text{Mask}_t$
$\theta_{t+1} = \theta_{t+1} \odot \text{Mask}_{t+1}$
$\theta_{t+2} = \theta_{t+2} \odot \text{Mask}_{t+2}$

(b) M^{2}SGD

Figure 1. Meta-SGD vs M^{2}SGD

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta \mathcal{L}_{t+1}(\theta_t) \tag{1}$$

with learning rate α and gradients of the learner loss with respect to the parameters $\nabla_\theta \mathcal{L}_{t+1}(\theta_t)$. Their experiments showed that neural optimisers updated learner parameters more rapidly and yielded learners with lower losses.

One particular simple yet successful neural optimiser is Meta-SGD [15]. The meta-learning target of Meta-SGD is the learning rates used for updating learner parameters. Meta-SGD learns a unique learning rate for each parameter of the learner network. Despite its simplicity, Meta-SGD has shown to be a successful neural optimiser.

In this present work, we present the **Masked Meta-SGD (M^{2}SGD)** as an extension to Meta-SGD. Two new features are added, and we depict the conceptual differences between updating a learner F with Meta-SGD and M^{2}SGD in **Figure 1**. First, we propose to learn the learning rates dynamically; and second, to learn a mask to remove redundant weights of the learner. Both features are learnt with a *long short-term memory* recurrent neural network (LSTM RNN) [10]. That is, our proposed M^{2}SGD neural optimiser employs an RNN to cast optimisation as a function of its hidden iterative variables. Together, these two features remove unimportant connections of the learner and ensure that only the important weights are being updated.

Current state-of-the-art neural networks possess a large amount of parameters. This makes them both memory intensive and computationally demanding. To address these limitations, sparse learning can be used to only learn the important connections. Han *et al*. [8] introduced an influential paradigm which removed redundant weights via a cyclic training which (1) trains, (2) prunes weights, and (3) retrains a dense network. This method is capable of reducing the total number of VGG-16 [23] parameters by 13 times while showing no loss in accuracy. However, cyclic training is itself a computationally expensive operation – it requires a dense network to be repeatedly pruned and retrained from the beginning of the training phase.

The masking system of M^{2}SGD removes unimportant weights as a part of its learnt optimisation algorithm. That is, M^{2}SGD is capable of finding a small size sub-network within a dense learner architecture *on the fly* as it updates the parameters. M^{2}SGD thus removes the need to retrain the dense network thereby hastening the weight removal procedure. Our experiments showed that M^{2}SGDs not only updated learners quickly, but also removed 83.71% weights for ResNet20s [9].

2. Preliminaries

Our method proposes a learnt optimisation algorithm that removes redundant learner weights. This section will first discuss the seminal paper of Andrychowicz *et al*. [1] which introduced the idea to train an RNN as an optimiser. We will also provide a detail description on the L2L experimental setup. Second, we will introduce its successful simplified method of Meta-SGD [15]. Then, we will discuss more on sparse learning.

2.1. The LSTM-optimiser

Andrychowicz *et al*. [1] proposed to conduct meta-learning via learning to optimise. They trained an RNN $\mathcal{V}(\phi)$ as a neural optimiser to update another independent learner $F(\theta_t)$. Below, we explain the L2L experiment as described in Andrychowicz *et al*. with **Algorithm 1**.

The neural optimiser meta-learner $\mathcal{V}$ employs an RNN to prepare updates g_t for the learner parameters θ_t via

$$\theta_{t+1} = \theta_t + g_{t+1} \quad \text{and} \tag{2}$$

$$[g_{t+1}, h_{t+1}] = \mathcal{V}\big(\mathscr{X}_t^{(\mathcal{V})}, h_t|\phi\big), \tag{3}$$

Algorithm 1 Training the LSTM-optimiser of [1]

Input:
Learner F with parameter θ,
LSTM-optimiser $\mathcal{V}$ with parameter ϕ,
and training dataset $\mathcal{D}_{\text{Train}}$.

Hyper-parameters:
Training-trial $\mathcal{Q}$, training-step $\mathcal{S}$,
unrolling-instance Ξ, and learning rate ω for $\mathcal{V}$.

```
1:  $\tau = 0$
2:  $\mathcal{D} \leftarrow$ set up dataset
3:  $\phi \leftarrow$ random initialisation                         ▷ For $\mathcal{V}$
4:  for $q = 1, \mathcal{Q}$ do                                     ▷ Updating $\mathcal{Q}$ learners
5:      $\theta_0 \leftarrow$ random initialisation                 ▷ For the $q$th $F$
6:      for $t = 1, \mathcal{S}$ do                                 ▷ Update for $\mathcal{S}$ steps
7:          $\mathcal{X}_t, \mathcal{Y}_t \leftarrow$ sample batch from $\mathcal{D}_{\text{Train}}$
8:          $\mathcal{L}_{t+1}(\theta_t) \leftarrow \mathcal{L}\big(F(\mathcal{X}_t|\theta_t), \mathcal{Y}_t\big)$   ▷ Acquire $\mathcal{L}$
9:          $\tau \leftarrow \tau + 1$
10:         $\mathcal{L}_{\tau+1}(\theta_\tau) \leftarrow \mathcal{L}_{t+1}(\theta_t)$        ▷ Record $\mathcal{L}$
11:         if $\tau = \Xi$ then                                    ▷ Unroll $\mathcal{V}$
12:             $\mathscr{L}(\phi) = \mathbb{E}\left[\sum_{\tau=1}^{\Xi} \mathcal{L}_{\tau+1}(\theta_\tau)\right]$   ▷ Acquire $\mathscr{L}$
13:             $\phi \leftarrow \phi - \omega \nabla_\phi \mathscr{L}(\phi)$            ▷ Update $\phi$
14:             $\tau = 0$                                          ▷ Reset records
15:         end if
16:         $\mathscr{X}_t^{(\mathcal{V})} \leftarrow \nabla_\theta \mathcal{L}_{t+1}(\theta_t)$   ▷ Prepare the input of $\mathcal{V}$
17:         $g_{t+1} \leftarrow \mathcal{V}(\mathscr{X}_t^{(\mathcal{V})}|\phi)$   ▷ Compute the update to $\theta_t$
18:         $\theta_{t+1} \leftarrow \theta_t + g_t$                    ▷ Update $\theta$
19:     end for
20: end for
```

where $\mathscr{X}_t^{(\mathcal{V})}$ is the input to $\mathcal{V}$ and that h_t is the hidden state of the RNN. Parameters of the neural optimiser are updated along learners parameters but only once every "unrolling". Below, the unroll procedure will be explained in detail, and we will discuss $\mathscr{X}_t^{(\mathcal{V})}$ afterwards.

The neural optimiser learns through updating $\mathcal{Q}$ learners for $\mathcal{S}$ steps. The objective loss $\mathscr{L}$ of $\mathcal{V}$ is dependent on the objective loss $\mathcal{L}$ of F. (Note the differences between the two Ls.) For every Ξ steps, the neural optimiser "unrolls" and parameters ϕ of $\mathcal{V}$ are updated based on their respective gradients to $\mathscr{L}$, which is defined as

$$\mathscr{L}(\phi) = \mathbb{E}\left[\sum_{\tau=1}^{\Xi} \mathcal{L}_{\tau+1}(\theta_\tau)\right]. \tag{4}$$

While the learner loss $\mathcal{L}_{t+1}(\theta_t)$ is dependent on the parameters, Equation (4) shows that the objective of $\mathcal{V}$ depends on the trajectory of the optimisation with Ξ as the maximum memory bandwidth of the RNN.

In their paper, the RNNs were 2-layer LSTMs [10] and their inputs $\mathscr{X}_t^{(\mathcal{V})}$ were the learner gradients $\nabla_\theta \mathcal{L}_{t+1}(\theta_t)$.

The *LSTM-optimiser* provides coordinate-wise updates to θ_t. Each entry is updated only with information of its own corresponding gradient. In practice, Andrychowicz *et al.* [1] pre-processed gradients as two vectors according to **Appendix A**. Input $\mathscr{X}_t^{(\mathcal{V})}$ is thus treated as a data of batch size $|\theta_t|$ and feature size 2. The hidden dimensions of the LSTMs were 20, and Ξ was dependent on the task at hand.

However, updating large learners with LSTMs incur high computational complexity. Moreover, it is difficult to interpret the learnt update rules. Fortunately, Li *et al.* [15] demonstrated that neural optimisers can be made simple.

2.2. Meta-SGD

Li *et al.* [15] proposed *Meta-SGD* to update learner parameters via

$$\theta_{t+1} = \theta_t - \beta \odot \nabla_\theta \mathcal{L}_{t+1}(\theta_t), \tag{5}$$

where $\beta \in \mathbb{R}^{|\theta_t|}$ and that symbol $\odot$ represents the element-wise product. Learning rates β are the meta-learning target of Meta-SGD; and they are a constant vector after training. Meta-SGD hence learns a unique learning rate for every feature. Compared to the popular LSTM-metalearner [20], it was shown that this simple method achieved better results than its complicated counterpart.

There is an important difference between Meta-SGD and the LSTM-optimiser. Unlike the LSTM-optimiser which learns to directly prepare for updates g_t, Meta-SGD explicitly utilise the native gradient $\nabla_\theta \mathcal{L}_{t+1}(\theta_t)$. That is, the LSTM-optimiser learns to precondition the gradients, whereas Meta-SGD learns auxiliary components that are compatible with the gradient descent paradigm.

2.3. Sparse learning

Sparse learning can be used to find small and sparse sub-networks within a dense architecture. Sparse networks enjoys multiple benefits from the removal of unimportant weights. For instance, the reduced size and computational cost makes it possible for small but powerful networks to be built on low-resource devices such as mobile phones.

In the early works, LeCun *et al.* [14] proposed to prune weights via second-order derivatives, and Ishikawa [11] modified the loss function to selectively prune weights with small magnitudes. More recently, Han *et al.* [8] showed that the number of parameters in a dense network could be reduce by an order of magnitude via the cyclic training of (1) train, (2) prune smallest weights, and (3) retrain.

However, in the "*Lottery Ticket Hypothesis*", Frankle & Carbin [5] noted that dense networks trained easier because there were more possible sub-networks which training might optimise to. Hence instead of pruning the weights, they masked the smallest weights of the learner and held them constant during training. They demonstrated that dense networks could be trained successfully even with up to 80% of their weights held constant.

As noted in Frankle & Carbin [5], cyclic training trains a dense learner for multiple times over multiple trials and is computationally costly. Our proposed M^{2}SGD is a trainable SGD-like optimiser which learns a mask to nullify unnecessary connections in the learner architecture. Hence after training, the M^{2}SGD simultaneously updates the learner parameters and applies a mask to remove redundant weights from the learner. This therefore allows the M^{2}SGD to find a lightweight sub-network within the dense architecture without the need of retraining, thereby hastening the weight removal procedure.

3. Extensions to Meta-SGD

Meta-SGD [15]
$\theta_{t+1} = \theta_t - \beta \odot \nabla_\theta \mathcal{L}_{t+1}(\theta_t)$
Dynamic Meta-SGD (DMSGD) [Ours]
$\theta_{t+1} = \theta_t + \beta_{t+1} \odot (-\nabla_\theta \mathcal{L}_{t+1}(\theta_t)), \quad (6)$ $\beta_{t+1} = \text{ReLU}(\beta^{sta} + \gamma \odot \beta^{dyn}_{t+1}), \quad (7)$ $\beta^{dyn}_{t+1} = \tanh(U_\beta h_{t+1}), \text{ and} \quad (8)$ $[h_{t+1}, c_{t+1}] = \text{LSTM}(\nabla_\theta L(\theta_t), h_t, c_t \mid \phi). \quad (9)$
Masked Meta-SGD (M^{2}SGD) [Ours]
After Dynamic Meta-SGD (6) - (9), apply $\theta_{t+1} = \theta_{t+1} \odot \text{Mask}_{t+1} \text{ where} \quad (10)$ $\text{Mask}^{(j)}_{t+1} = \max(0, \text{sgn}(\beta^{(j)}_{t+1} > \xi)) \text{ for} \quad (11)$ $j = 1, \ldots, |\theta_t|.$

Table 1. Extending the Meta-SGD neural optimiser

In this section, we introduce the **Dynamic Meta-SGD (DMSGD)** and the **Masked Meta-SGD (M^{2}SGD)** as extensions to the existing Meta-SGD [15] neural optimiser meta-learner. We compare the algorithms side-by-side in **Table 1** and discuss our newly proposed features below.

3.1. The Dynamic Meta-SGD (DMSGD)

The DMSGD neural optimiser is described in Equations (6) to (9) in **Table 1**. Similar to Meta-SGD, the sole meta-learning target is the learning rates β_t. As described in **Section 2.2**, the Meta-SGD learning rates are constants after training. Here, we develop DMSGD as an optimisation algorithm which fine-tunes its own learning rates given the gradients of the learner parameters. In addition to the existing static learning rates β^{sta}, DMSGD infers for an additional set of dynamic learning rates β_t^{dyn} with an LSTM. Variables h_t and c_t are the hidden states and the cell states of the LSTM respectively. Vector $\gamma \in \mathbb{R}^{|\theta_t|}$ provides optional weights for β^{dyn}; and we default γ as a constant vector $\vec{\mathbf{1}}$ unless stated otherwise. We denote tanh as the hyperbolic tangent function, and ReLU as the rectified linear unit [7].

There are two important notes for the DMSGD. First, like the LSTM-optimiser introduced in **Section 2.1**, the DMSGD employs a coordinate-wise LSTM for inferring β_t^{dyn}. Second, the DMSGD provides no update to the jth element of parameters θ_t when $|\beta^{sta^{(j)}}| < |\gamma^{(j)}\beta_t^{dyn^{(j)}}|$ and $\beta^{sta^{(j)}}\gamma^{(j)}\beta_t^{dyn^{(j)}} < 0$. Hence the DMSGD is trained to find and sparsely update the important weights of the learner while keeping the less important weights as constants.

3.2. The Masked Meta-SGD (M^{2}SGD)

The M^{2}SGD neural optimiser is built on the DMSGD and further extends Meta-SGD with a masking system as described in Equations (10) and (11) in **Table 1**. The mask is applied after parameters θ_t are updated; and its purpose is to nullify weights that are secondary in importance.

In this work, we consider less important weights as those that are updated with small learning rates. Our conjecture is based on two reasons. First, learner networks are encouraged to have their parameters randomly initialised with small magnitudes in order to prevent saturation in layer outputs [6]. Second, randomly initialised learners are non-optimal and have low accuracy. Hence following reasons 1 and 2, weights that are not updated with large learning rates are of low priority to the successful inferential process of a converged learner. Due to these reasons, we nullify the weights base on the magnitude of their respective learning rates β_t as shown in Equation (11).

In Equation (11), ξ is a non-negative lower bound for the learning rates. It represents a threshold for the importance of the weights. We denote sgn as the sign function. When $\xi = 0$, M^{2}SGD nullifies weights that receive no update during training; and when $\xi > 0$, we nullify both the non-updated weights and a portion of weights with less significant learning rates.

4. L2L image recognition

In this section, we followed Andrychowicz *et al.* [1] and employed neural optimisers to update learners for image recognition. In their paper, Andrychowicz *et al.* employed neural optimisers to update small size learners. They updated multiple layer perceptron (MLP) learners for the first 100 iterations, and updated small convolutional learners for the first 1000 iterations. They demonstrated that neural optimisers updated the small learners more rapidly than the handcrafted optimisation methods of SGD and ADAM.

Andrychowicz *et al.* focused on introducing the concept of L2L, but in this present work we focus on performance verification. Hence, we adapted three changes. First, we updated large learners instead of MLPs. Second, Andrychowicz *et al.* updated MLPs to classify the small size 28×28 greyscale MNIST [13] images, and we selected a more difficult dataset. Third, instead of updating learners for the first 1000 steps, we updated them until they converged.

4.1. Dataset, learner, and experimental setups

We trained neural optimisers to update ResNet20 [9] learners for STL10 [3]. The low resource STL10 dataset consists 5,000 training and 8,000 test RGB colour images in 10 classes on 96×96 pixels; whereas the ResNet20 learner follows the identical architecture as described in [22] and has 270K weights.

We trained DMSGDs, M^{2}SGDs, and Meta-SGDs to update ResNet20s in a similar fashion to that in **Algorithm 1**. Lines 16 to 18 of the algorithm vary for each neural optimiser, and these lines should be substituted with the correct formulation as documented in **Table 1**. The minimisation for the neural optimiser loss $\mathscr{L}$ of lines 12 and 13 was performed using ADAM with learning rate 0.001.

Additionally, we set unroll $\Xi = 5$, learner-trials $\mathscr{Q} = 1$, and learner-steps $\mathscr{S} = 100$ (, see **Algorithm 1**). The combination of these hyper-parameters meant that, each neural optimiser was trained to update 1 learner for 100 steps; and that each neural optimiser memorised 5 continual steps of the depth of the optimisation trajectory.

After the neural optimisers were trained, we employed them to update the learners until they converged. Despite having the neural optimisers trained only to update the learners for $\mathscr{S} = 100$ steps, we updated ResNet20s for 4000 iterations. We emphasise that this number (of 4000 steps) was the least amount of iterations for Meta-SGDs to converge the learners. That is, the setups of this section was purposely selected to favour the optimisation results for our rival Meta-SGD neural optimiser.

We also compared our neural optimisers to the classic SGD. The classic SGD updated the ResNet20 learners with learning rate 0.001 and with momentum 0.9. We also updated learners with the classic SGD until they converged (in 7850 steps). These results were included to verify that neural optimisers converged learners with comparable performance to those that were updated by the classic SGD.

4.2. Results and analysis

After training the neural optimisers, we employed them to update twenty randomly initialised learners with different θ_0, and our results are summarised in **Figures 2** and **3**, and in **Table 2**. Both figures illustrate the learning curves of the learner networks while being updated with different optimisers. **Figure 2** shows the performances of the learners from initialisation to convergence; whereas **Figure 3** shows the leaners being lightly updated for the first 1000 iterations. **Table 2** records various statistics, including the average accuracy of the updated learners, the amount of iterations applied to the learners, the total amount of weights of the yielded learners, and the amount of weight reduction from the initial dense architecture. We use these results to address four main findings below.

First, as shown in **Figure 2**, all neural optimisers updated learners more efficiently than SGDs. We found that M^{2}SGDs performed comparably to Meta-SGD, while DMSGDs outperformed all rival optimisers. DMSGDs updated the learners more rapidly, and converged learners to the least amount of loss.

Second, as shown in **Table 2**, DMSGDs converged ResNets20s with the highest accuracy 65.52%. This is significantly higher than Meta-SGDs' converged accuracy 60.39% and marginally higher than the classic SGDs' 63.64%. In addition, DMSGDs were able to converge the learners much faster than SGDs. DMSGDs managed to converge ResNet20s in 4000 steps of update, while SGDs required 7850 steps. Furthermore, SGDs were only able to yield ResNet20s with 59.54% after 4000 steps of updates. Thus we showed that the learning rates could be learnt to yield competitive learners via only updating the important weights.

Third, **Table 2** shows that M^{2}SGDs yielded competitive ResNet20s and also removed a significant portion of weights from the learners. M^{2}SGDs with $\xi = 0.2825$ removed 83.71% weights for ResNet20s and lowered the total amount of weights from 270K to 40K. In addition, M^{2}SGDs converged these learners to 63.53% in 4000 steps, and matched those that were converged by SGDs to 63.64% in 7850 steps. This showed that unnecessary learner connections can be successfully detected by the magnitude of their corresponding learning rates. Furthermore, these positive results showed that sparse sub-networks can be found without the need to retrain a dense network. Hence, our M^{2}SGD neural optimisers updated learners quickly and also provided efficient weight removal.

Forth, that the final accuracy of the learners are equivalently important to the convergence rate of neural optimisers. **Figure 2** shows that Meta-SGDs updated ResNet20s more rapidly than the classic SGDs; however, **Table 2** reveals that the final accuracy of their learners were not as competitive (60.39%) as those that were updated with the classic SGDs (63.64%). Hence a neural optimiser should not only be judged by the rate which they converged the learners, but also with their learners' converged accuracy.

4.3. Open questions in the original L2L experiment

Here we address three open questions that we have identified in Andrychowicz *et al.* [1]'s L2L experiments. Our aim is to use our results to complement their previous work.

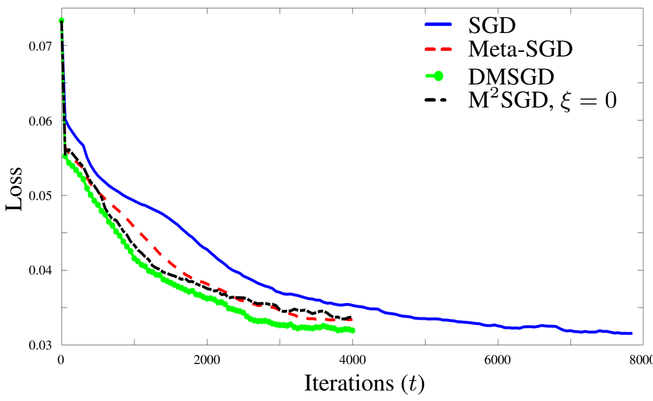


Figure 2. Loss curve for updating ResNet20s (until convergence)

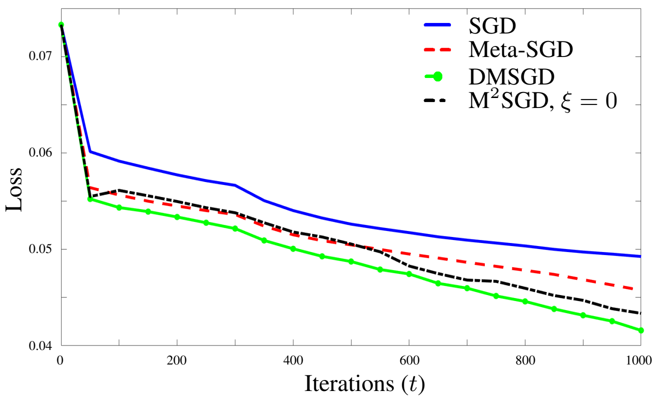


Figure 3. Loss curve for updating ResNet20s (first 1000 steps)

Optimiser	% Accuracy	# Iterations	# Weights (% ↓)
SGD	59.54 ± 0.23	4000	270K (- -)
SGD	63.64 ± 0.73	7850	270K (- -)
Meta-SGD [15]	60.39 ± 2.27	4000	270K (- -)
DMSGD [Ours]	$\mathbf{65.52 \pm 1.22}$	4000	270K (- -)
M^{2}SGD [Ours]			
$\xi = 0$	65.10 ± 1.17	4000	140K (46.89%)
$\xi = 0.1$	64.63 ± 1.50	4000	100K (61.31%)
$\xi = 0.2825$	63.53 ± 0.38	4000	**40K** (83.71%)

Table 2. Performance details of ResNet20 learners

As mentioned in the **opening remarks**, Andrychowicz *et al.* updated small learners for a short duration. Similar to **Figures 2** and **3**, they provided learning curves of different optimisers, and showed that neural optimisers updated learners more rapidly than the hand-designed counterparts. However, they did not report the accuracy of their learners.

The first open question concerns the efficiency of updating learners with neural optimisers for a long duration. This is an important question for understanding whether neural optimisers converged learners to undesirable suboptimal local minima. To ensure convergence, we extended the amount of updates from 1000 iterations to 4000 iterations. As we show in **Figure 2**, neural optimisers updated the learners nicely for the prolonged duration. There were no dramatic changes in the slopes and all final losses were similar. There were thus no obvious training difficulties.

The second open question regards the converged accuracy of learners. In their work, Andrychowicz *et al*. demonstrated the rapid knowledge acquisition of their neural optimisers, but they did not report the accuracy of their learner networks. As we previously discussed in result 4 of **Section 4.2**, the converged accuracy is as important as the rate of update; and the two desirable qualities do not necessarily exist simultaneously.

The last open question regards the compatibility of updating large size learners with neural optimisers. In the original work, Andrychowicz *et al*. updated a single layer MLP with 20 hidden units. Our ResNet20 learner is an order of magnitude larger than their MLP, and our results in **Table 2** show no difficulties in updating the larger ResNet20.

4.4. More results

We also found a similar pattern of success in updating MLPs with DMSGDs and M^{2}SGDs. The MLPs were selected to classify Fashion-MNIST [26]. We summarised the extra results in **Appendix B**.

5. Related works

Meta-learning

Meta-learning is a large field of study that addresses the general idea of learning quickly. The concept can be formulated in many different ways. Our work focused on L2L [1]; and alternatively, one can consider the task of few-shot learning as described in Vinyals *et al*. [25]. That task trains a network which is capable of mapping unlabelled data to their labels through inferring with a small labelled support set.

There are three common approaches in meta-learning. They are the (1) metric-based [25], (2) model-based [16], and (3) optimisation-based [1] learning. Metric-based learning aims to learn a network which encodes inputs as a kernel function for measuring the similarity between sampled data. Model-based learning aims to learn a meta-learner which produces weights for the learner based on sampled data. Our work is most aligned with optimisation-based learning, which we discuss separately below.

There are three sub-classes of optimisation-based meta-learners. First, Finn *et al*. [4] proposed the double-loop MAML optimisation setup. The inner loop finds κ copies of candidate fast-weights and losses; then the outer loop performs a classic gradient descent which aggregates over the κ losses. Second, there are neural optimisers which precondition the gradient like the LSTM-optimiser [1] (see **Section 2.1**). Park & Oliva [19] showed that the first two classes can be combined to yield better results. Third, there are neural optimisers which meta-learn auxiliary components to update learner parameters with the native gradient. This includes the LSTM-metalearner [20], Meta-SGD [15], and our proposed methods of DMSGD and M^{2}SGD.

Sparse learning

As mentioned in **Section 2**, early techniques prune weights through second-order derivatives [14] and heuristics [11]. Whereas recent works, including network compression [2, 17], follow the paradigmatic prune-and-retrain cyclic training of Han *et al*. [8]. The work of Frankle & Carbin [5] found that dense networks could also be updated while holding redundant weights as constants. However, most of these approaches are computationally costly due to the need to repeatedly retrain dense networks.

Comparison

In this paper, we showed that it was possible to conduct weight removal via meta-learning. Our proposed M^{2}SGD neural optimiser learns a direct rule to sparsely update learner parameters while simultaneously removing redundant weights with a masking system. M^{2}SGDs do not require the learners to be retrained and thus hastens the weight removal procedure.

The DMSGD method shares some similarities to the Lottery Ticket Hypothesis of Frankle & Carbin [5]. Instead of removing the less important weights, DMSGDs learn to provide them with no updates and thus indirectly keeping them as constants.

The M^{2}SGD is more closely related to the early sparse learning works of [14] and does not remove weights base on the weights' magnitudes. Instead, we influenced the weights through the magnitudes of the learning rates. We considered weights that were associated with small learning rates as parameters that were secondary in importance. This is because that they were not required to be configured quickly for successful network inferences. The masking system of M^{2}SGD is designed to nullify weights with small learning rates.

6. Discussion

This paper introduced **DMSGD** and **M2SGD** as extensions to the optimisation-based Meta-SGD [15] metalearner. Our proposed methods added two new features. First we inferred for dynamic learning rates with an LSTM [10]; and second, we added a masking system to nullify weights that were considered secondary in importance. On the L2L [1] task for classifying STL10 [3] images, our neural optimisers updated ResNet20s [9] more rapidly than Meta-SGDs, converged ResNet20s to lower losses, updated ResNet20s with better accuracy, and removed 83.71% weights from the dense ResNet20 architecture.

Acknowledgement

This research was supported by the Australian Government Research Training Program (AGRTP) Scholarship.

References

[1] Marcin Andrychowicz, Misha Denil, Sergio Gómez, 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*, pages 3981–3989. 2016. 1, 2, 3, 4, 5, 6, 7, 8

[2] Miguel A Carreira-Perpinán and Yerlan Idelbayev. **"Learning-Compression" Algorithms for Neural Net Pruning**. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 8532–8541, 2018. 6

[3] Adam Coates, Andrew Ng, and Honglak Lee. **An Analysis of Single-Layer Networks in Unsupervised Feature Learning**. In *International Conference on Artificial Intelligence and Statistics*, pages 215–223, 2011. 4, 6

[4] Chelsea Finn, Pieter Abbeel, and Sergey Levine. **Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks**. In *International Conference on Machine Learning*, pages 1126–1135, 2017. 6

[5] Jonathan Frankle and Michael Carbin. **The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks**. In *International Conference on Learning Representations*, 2019. 3, 6

[6] Xavier Glorot and Yoshua Bengio. **Understanding the Difficulty of Training Deep Feedforward Neural Networks**. In *International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010. 4

[7] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. **Deep Sparse Rectifier Neural Networks**. In *International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011. 4

[8] Song Han, Jeff Pool, John Tran, and William Dally. **Learning both Weights and Connections for Efficient Neural Network**. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015. 2, 3, 6

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. **Deep Residual Learning for Image Recognition**. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016. 2, 4, 6

[10] Sepp Hochreiter and Jürgen Schmidhuber. **Long Short-Term Memory**. *Neural Computation*, 9(8):1735–1780, 1997. 2, 6

[11] Masumi Ishikawa. **Structural Learning with Forgetting**. *Neural Networks*, 9(3):509–521, 1996. 3, 6

[12] Diederik P Kingma and Jimmy Ba. **Adam: A Method for Stochastic Optimization**. *International Conference on Learning Representations*, 2014. 1

[13] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. **Gradient-Based Learning Applied to Document Recognition**. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 4, 8

[14] Yann LeCun, John S Denker, and Sara A Solla. **Optimal Brain Damage**. In *Advances in Neural Information Processing Systems*, pages 598–605, 1990. 3, 6

[15] 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. 1, 2, 3, 5, 6, 8

[16] Tsendsuren Munkhdalai and Hong Yu. **Meta Networks**. In *International Conference on Machine Learning*, pages 2554–2563, 2017. 6

[17] Sharan Narang, Erich Elsen, Gregory Diamos, and Shubho Sengupta. **Exploring Sparsity in Recurrent Neural Networks**. In *International Conference on Learning Representations*, 2017. 6

[18] Yurii E Nesterov. **A Method for Solving the Convex Programming Problem with Convergence Rate $O(\frac{1}{k^2})$**. In *Dokl. Akad. Nauk SSSR (Proceedings of the USSR Academy of Sciences)*, volume 269, pages 543–547, 1983. 1

[19] Eunbyung Park and Junier B Oliva. **Meta-Curvature**. In *Advances in Neural Information Processing Systems*, pages 3309–3319, 2019. 6

[20] Sachin Ravi and Hugo Larochelle. **Optimization as a Model for Few-Shot Learning**. In *International Conference on Learning Representations*, 2017. 3, 6

[21] Herbert Robbins and Sutton Monro. **A Stochastic Approximation Method**. *The annals of mathematical statistics*, pages 400–407, 1951. 1

[22] Shawn Shan, Emily Willson, Bolun Wang, Bo Li, Haitao Zheng, and Ben Y Zhao. **Gotta Catch'Em All: Using Concealed Trapdoors to Detect Adversarial Attacks on Neural Networks**. *arXiv preprint arXiv:1904.08554*, 2019. 4

[23] Karen Simonyan and Andrew Zisserman. **Very Deep Convolutional Networks for Large-Scale Image Recognition**. In *International Conference on Learning Representations*, 2015. 2

[24] Tijmen Tieleman and Geoffrey Hinton. **Lecture 6.5-RMSProp: Divide the Gradient by a Running Average of Its Recent Magnitude**. *COURSERA: Neural Networks for Machine Learning*, 4(2):26–31, 2012. 1

[25] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. **Matching Networks for One Shot Learning**. In *Advances in Neural Information Processing Systems*, pages 3630–3638, 2016. 6

[26] Han Xiao, Kashif Rasul, and Roland Vollgraf. **Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms**. *arXiv preprint arXiv:1708.07747*, 2017. 6, 8

A. Gradient pre-processing

This appendix documents the gradient pre-processing used in Andrychowicz *et al.* [1]. We reiterate the following content for the completeness of this paper, and to highlight the size of the input dimension of the LSTM-optimiser in **Section 2**. The original content can be found on page 11 of [1] under "A Gradient preprocessing".

Andrychowicz *et al.* proposed to pre-process the neural optimiser's input gradient $\nabla_\theta \mathcal{L}_{t+1}(\theta_t)$. We will simplify $\nabla_\theta \mathcal{L}_{t+1}(\theta_t)$ as ∇ in this appendix. The pre-processing scheme is given as

$$\nabla^{(j)} \rightarrow \begin{cases} \left(\frac{\log(|\nabla|)}{p}, \text{sgn}(\nabla)\right) & \text{if } |\nabla| \geq e^{-p}, \\ (-1, e^p \nabla) & \text{otherwise} \end{cases}$$

and pre-processes each element of the gradient $\nabla^{(j)}$ for $j = 1 \ldots |\theta_t|$ as a pair of values. The hyper-parameter p is defaulted as 10 in all of their and in all of our experiments.

Andrychowicz *et al.* devised this scheme to ensure that the magnitudes of every dimension is on the same order. They mentioned that this is necessary because neural networks, and including neural optimisers "*naturally disregard small variations in input signals and concentrate on bigger input values*". In addition, hyper-parameter $p > 0$ controls how small gradients are disregarded.

B. More L2L experiments for MLPs

This appendix addresses additional L2L experiments for MLPs. Similar to **Section 4**, we employed classic SGDs, Meta-SGDs, DMSGDs, and M^{2}SGDs to update the MLPs. The MLP learners were tasked to classifying images of the Fashion-MNIST [26] dataset.

As noted in **Section 4.1**, Andrychowicz *et al.* [1] updated MLPs on the MNIST dataset. Here, we employ neural optimisers to update MLPs, but on a slightly more difficult dataset. First, the MNIST dataset [13] contains 60,000 training and 10,000 test images in 10 classes of 28×28 greyscale handwritten digits. On the other hand, Fashion-MNIST is a slightly more difficult dataset crafted similarly to MNIST. FashionMNIST consists 60,000 training and 10,000 test grayscale images of 10 classes of fashion items on 28×28 pixels.

Our MLP learners were simple. They consisted one linear layer followed by a ReLU activation, and one softmax layer. The linear layer received all Fashion-MNIST images as vectors of pixels. Thus, the input dimension was $784 (= 28 \times 28)$, and we set the hidden size to 32. The MLP leaner had 25.4K parameters.

All hyper-parameteric setup follow that in **Section 4.1**; hence we will not reiterate them here. After training the neural optimisers, we employed them to update the MLP learners until convergence (for 10000 iterations); and also updated the MLP learners with classic SGDs until convergence (for 37500 iterations). We summarised the results in **Figures 4** and **5**, and in **Table 3**.

Figures 4 and **5** plot the first 5000 and 1000 steps of the learner loss curves respectively. We found that all neural optimisers updated the MLPs more rapidly than SGDs. Furthermore, we found that both DMSGDs and M^{2}SGDs outperformed Meta-SGDs.

We found a similar pattern of successes to **Section 4** in **Table 3**. Again, DMSGDs converged the learners with the highest accuracy (85.37%). Those converged MLPs that were updated by M^{2}SGDs with $\xi = 0.2$ enjoyed a similar level of accuracy (83.81%) to those that were updated by the classic SGDs (82.72%), but also had 68.94% less parameters.

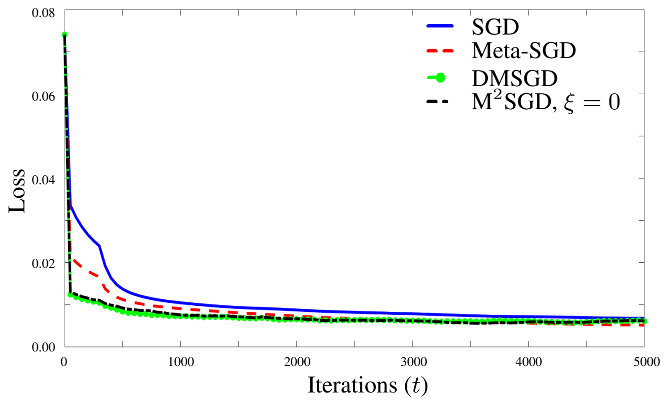


Figure 4. Loss curve (first 5000 steps) for Fashion-MNIST

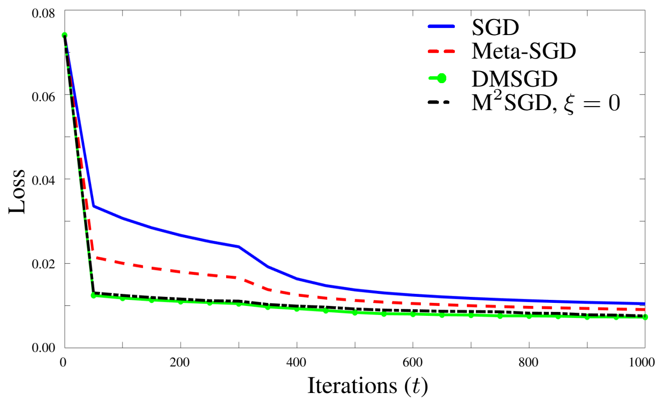


Figure 5. Loss curve (first 1000 steps) for Fashion-MNIST

Optimiser	**% Accuracy**	**# Iterations**	**# Weights (% ↓)**
SGD	82.72 ± 0.55	37500	25.4K (- -)
Meta-SGD [15]	85.06 ± 0.72	10000	25.4K (- -)
DMSGD [Ours]	$\mathbf{85.37 \pm 0.55}$	10000	25.4K (- -)
M^{2}SGD [Ours]			
$\xi = 0$	84.80 ± 0.35	10000	13.0K (49.00%)
$\xi = 0.1$	84.46 ± 0.43	10000	9.9K (60.68%)
$\xi = 0.2$	83.81 ± 0.76	10000	**7.8**K (68.94%)

Table 3. Performance details of MLP learners