MTL2L: A Context Aware Neural Optimiser

Nicholas I-Hsien Kuo[1]   U6424547@ANU.EDU.AU
Mehrtash Harandi[2]   MEHRTASH.HARANDI@MONASH.EDU
Nicolas Fourrier[3]   NFOURRIER@GMAIL.COM
Christian Walder[1,4]   CHRISTIAN.WALDER@DATA61.CSIRO.AU
Gabriela Ferraro[1,4]   GABRIELA.FERRARO@DATA61.CSIRO.AU
Hanna Suominen[1,4,5]   HANNA.SUOMINEN@ANU.EDU.AU

[1] : *RSCS, The Australian National University, Canberra, ACT, Australia*

[2] : *ECSE, Monash University, Melbourne, Victoria, Australia*

[3] : *Research Center of Léonard de Vinci Pôle Universitaire, Paris La Défense, France*

[4] : *Data61, CSIRO, Canberra, ACT, Australia*

[5] : *Department of Future Technologies, University of Turku, Turku, Finland*

Abstract

Learning to learn (L2L) trains a meta-learner to assist the learning of a task-specific base learner. Previously, it was shown that a meta-learner could learn the direct rules to update learner parameters; and that the learnt neural optimiser updated learners more rapidly than handcrafted gradient-descent methods. However, we demonstrate that previous neural optimisers were limited to update learners on *one* designated dataset. In order to address input-domain heterogeneity, we introduce **Multi-Task Learning to Learn (MTL2L)**, a context aware neural optimiser which self-modifies its optimisation rules based on input data. We show that MTL2L is capable of updating learners to classify on data of an unseen input-domain at the meta-testing phase.

We release our codes at https://github.com/Nic5472K/AutoML2020_ICML_MTL2L.

1. Introduction

Deep learning has firmly established itself in image recognition (He et al., 2016), speech recognition (Oord et al., 2016), and text processing (Vaswani et al., 2017) through training models with high-capacity on a large amount of labelled data. However, it remains difficult to learn from a low resource environment. This challenging task is carefully studied in meta-learning, the branch of machine learning that addresses rapid knowledge acquisition from only a few examples. In this paper, we address extensions to the neural optimiser approach for the meta-learning task of *Learning to learn* (L2L) (Andrychowicz et al., 2016).

L2L formulates the concept of learning quickly as the task to rapidly converge a network with the least amount of iterations. This task involves training a meta-learner to assist the learning of a base learner. The learner performs a specific task, such as image classification; whereas the meta-learner captures how task structures vary within the domain (Rendell et al., 1987). This paper extends on meta-learners which directly update learner parameters.

Andrychowicz et al. (2016) cast optimisation as the learning target for a recurrent neural network (RNN). The RNN was *meta-trained* as a neural optimiser meta-learner; then *meta-tested* to update a base learner. They showed that the *RNN neural optimiser* converged
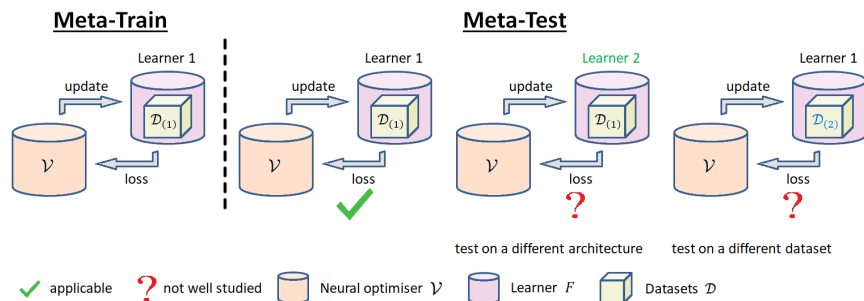
Figure 1: Neural optimiser applicability

learners more rapidly than hand-designed optimisations such as SGD (Robbins and Monro, 1951) and ADAM (Kingma and Ba, 2014). Learning to optimise has since become a common approach in meta-learning, and has been extended by the LSTM-metalearner (Ravi and Larochelle, 2017) and Meta-SGD (Li et al., 2017).

However, neural optimisers have the two limitations[1] illustrated in **Figure 1**. First, neural optimisers are specific to architectures. If they were meta-trained to update a multiple layer perceptron (MLP), it cannot be guaranteed to work on ResNet (He et al., 2016). Second, neural optimisers are specific to input-domain (dataset). When naively applied, they cannot update the same learner on two distinct datasets as the underlying task structures may vary significantly in the two input-domains. Neural optimisers are thus not as generally applicable as SGD.
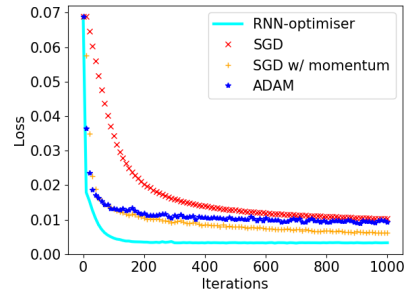
We prepare **Figure 2** to demonstrate and to convince our readers of the consequences of naively applying neural optimisers on an unseen dataset. The solid cyan lines represent the RNN-optimiser of Andrychowicz et al. (2016). **Figure 2(a)** illustrates meta-learning under the common assumption:

(a) Meta-trained on MNIST and Meta-tested on MNIST

(b) Meta-trained on MNIST and Meta-tested on Modified Cifar10

Figure 2: Domain variance

to meta-test on the single identical input-domain used in meta-training; and **Figure 2(b)** depicts deteriorated performance in meta-testing on an unseen dataset. In this paper, we introduce **Multi-Task Learning to Learn (MTL2L)** – a novel context aware neural optimiser which supports *multi-task learning* (Caruana, 1997). MTL2L can capture shared task structures across multiple input-domains to update learners on a dataset unseen during meta-training. The experiments will be revisited in **Section 4**.
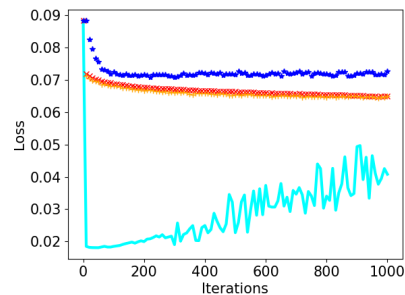
2. Preliminaries

The classic gradient descent updates parameters θ_t of learner F at time t via

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

---

1. See more descriptions in **Appendix A**.
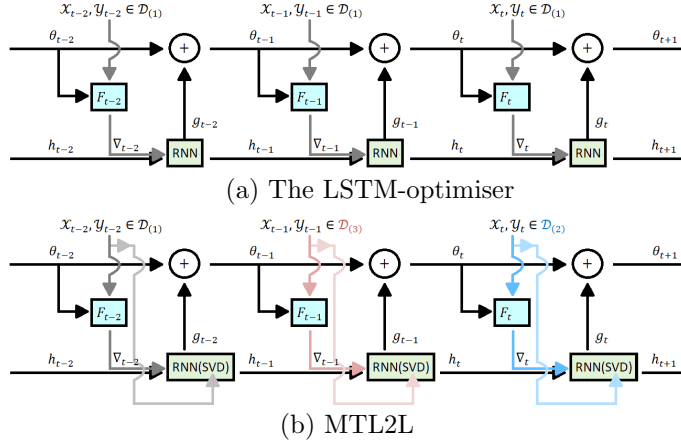
(a) The LSTM-optimiser

(b) MTL2L

Figure 3: A comparison between the LSTM-optimiser and MTL2L

with learning rate α and gradients of the learner loss with respect to the parameters $\nabla_\theta \mathcal{L}_{t+1}(\theta_t)$. Andrychowicz et al. (2016) proposed to replace Equation (1) with an RNN $\mathcal{V}$ with parameters ϕ to learn the direct rules g_t to update the learner parameters

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

$$[g_{t+1}, h_{t+1}] = \mathcal{V}\big(\nabla_\theta \mathcal{L}_{t+1}(\theta_t), h_t | \phi\big), \tag{3}$$

where h_t is the hidden states of $\mathcal{V}$. This concept is illustrated in **Figure 3(a)**.

We now describe the training of the neural optimiser with **Algorithm 1**. Neural optimiser $\mathcal{V}$ is trained via updating $\mathcal{Q}$ learners for $\mathcal{S}$ steps on the training data $\mathcal{D}_{\text{Train}}$. They are updated along with the learner but on a different time scale – only once every "unroll". The objective loss $\mathscr{L}$ of $\mathcal{V}$ is dependent on the objective loss $\mathcal{L}$ of F over Ξ time steps, and is defined as

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

The neural optimiser is updated via classic gradient descent with learning rate ω; and $\mathscr{L}$ shows that $\mathcal{V}$ is set to learn the trajectory for optimising learner F with Ξ units of memory bandwidth for its RNN.

$\tau = 0$
$\phi \leftarrow$ random initialisation
for $q = 1, \mathcal{Q}$ **do**
    $\theta_0 \leftarrow$ random initialisation
    for $t = 1, \mathcal{S}$ **do**
        $\mathcal{X}_t, \mathcal{Y}_t \leftarrow$ sample from $\mathcal{D}_{\text{Train}}$
        $\mathcal{L}_{t+1}(\theta_t) \leftarrow \mathcal{L}\big(F(\mathcal{X}_t|\theta_t), \mathcal{Y}_t\big)$
        $\tau \leftarrow \tau + 1$
        $\mathcal{L}_{\tau+1}(\theta_\tau) \leftarrow \mathcal{L}_{t+1}(\theta_t)$
        if $\tau = \Xi$ **then**
            $\mathscr{L}(\phi) = \mathbb{E}\left[\sum_{\tau=1}^{\Xi} \mathcal{L}_{\tau+1}(\theta_\tau)\right]$
            Update meta-learner
            $\phi \leftarrow \phi - \omega \nabla_\phi \mathscr{L}(\phi)$
            $\tau = 0$
        end
        $g_{t+1} \leftarrow \mathcal{V}\left(\nabla_\theta \mathcal{L}_{t+1}(\theta_t) | \phi\right)$
        Update learner
        $\theta_{t+1} \leftarrow \theta_t + g_t$
    end
end

Algorithm 1: Meta-training

3. Context awareness through HyperNetwork

Andrychowicz et al. (2016) employed the *long short-term memories* (LSTMs) (Hochreiter and Schmidhuber, 1997; Gers et al., 1999) presented in Equations (5) to (8) in **Table 1** as RNNs for their neural optimiser. Symbols σ, tanh, and $\odot$ are the sigmoid function, hyperbolic tangent function, and element-wise dot product, respectively; with synapses $\mathbf{W}$ and $\mathbf{U}$, and bias $\mathbf{b}$. The forget gate $\mathbf{f}_t$ and the input gate $\mathbf{i}_t$ modulate cell $\mathbf{c}_t$ with new

Traditional LSTM	**MTL2L (Ours)**

$$\begin{bmatrix}\mathbf{f}_t\\\mathbf{i}_t\\\mathbf{o}_t\end{bmatrix}=\sigma\begin{pmatrix}\mathbf{W}_{\mathcal{F}}\mathbf{x}_t+\mathbf{U}_{\mathcal{F}}\mathbf{h}_{t-1}+\mathbf{b}_{\mathcal{F}}\\\mathbf{W}_{\mathcal{I}}\mathbf{x}_t+\mathbf{U}_{\mathcal{I}}\mathbf{h}_{t-1}+\mathbf{b}_{\mathcal{I}}\\\mathbf{W}_{\mathcal{O}}\mathbf{x}_t+\mathbf{U}_{\mathcal{O}}\mathbf{h}_{t-1}+\mathbf{b}_{\mathcal{O}}\end{pmatrix}, \quad (5)$$

$$\begin{bmatrix}\mathbf{f}_t\\\mathbf{i}_t\\\mathbf{o}_t\end{bmatrix}=\sigma\begin{pmatrix}\mathcal{W}_t^{(\mathcal{F})}\mathbf{x}_t+\mathcal{U}_t^{(\mathcal{F})}\mathbf{h}_{t-1}+\mathbf{b}_{\mathcal{F}}\\\mathcal{W}_t^{(\mathcal{I})}\mathbf{x}_t+\mathcal{U}_t^{(\mathcal{I})}\mathbf{h}_{t-1}+\mathbf{b}_{\mathcal{I}}\\\mathcal{W}_t^{(\mathcal{O})}\mathbf{x}_t+\mathcal{U}_t^{(\mathcal{O})}\mathbf{h}_{t-1}+\mathbf{b}_{\mathcal{O}}\end{pmatrix}, \quad (9)$$

$$\mathbf{a}_t=\tanh(\mathbf{W}_{\mathcal{A}}\mathbf{x}_t+\mathbf{U}_{\mathcal{A}}\mathbf{h}_{t-1}+\mathbf{b}_{\mathcal{A}}), \quad (6)$$
$$\mathbf{a}_t=\tanh(\mathcal{W}_t^{(\mathcal{A})}\mathbf{x}_t+\mathcal{U}_t^{(\mathcal{A})}\mathbf{h}_{t-1}+\mathbf{b}_{\mathcal{A}}), \quad (10)$$

$$\mathbf{c}_t=\mathbf{f}_t\odot\mathbf{c}_{t-1}+\mathbf{i}_t\odot\mathbf{a}_t, \text{ and} \quad (7)$$
$$\mathbf{c}_t=\mathbf{f}_t\odot\mathbf{c}_{t-1}+\mathbf{i}_t\odot\mathbf{a}_t, \text{ and} \quad (11)$$

$$\mathbf{h}_t=\mathbf{o}_t\odot\tanh(\mathbf{c}_t). \quad (8)$$
$$\mathbf{h}_t=\mathbf{o}_t\odot\tanh(\mathbf{c}_t); \quad \text{where} \quad (12)$$

Note:
The modified components are coloured in brown. The components in purple are set as <u>constants</u> prior to training; whereas those in orange are <u>learnt</u> from training and dependent on input x_t.

$$\mathcal{W}_t=Q\,\Gamma_t\,P^{\mathrm{T}}, \quad (13)$$
$$\mathcal{U}_t=S\,\Omega_t\,J^{\mathrm{T}}, \quad \text{such that} \quad (14)$$
$$\Gamma_t=\mathrm{diag}\big(\mathcal{N}^{(1)}(\mathbf{x}_t)\big), \text{ and} \quad (15)$$
$$\Omega_t=\mathrm{diag}\big(\mathcal{N}^{(2)}(\mathbf{x}_t)\big). \quad (16)$$

Table 1: A comparison between traditional LSTM against MTL2L

memory $\mathbf{a}_t$. The LSTM output is hidden state $\mathbf{h}_t$, $\mathbf{c}_t$ transformed with controlled exposure from the output gate $\mathbf{o}_t$. Our novel **Multi-task Learning to Learn (MTL2L)** neural optimiser is introduced in Equations (9) to (16), and replace LSTMs as $\mathcal{V}$ in Equation (3).

We took a *HyperNetwork* (Ha et al., 2016) approach towards dataset-agnostic meta-learning. HyperNetworks are small auxiliary networks that re-implement weights for a larger network based on input data. Equations (13) and (14) describe MTL2L synapses as variables via *singular value decomposition* (SVD). SVD splits each synapse as fixed purple terms and orange components that require learning. The former are constant orthonormal matrices, and the latter are diagonal matrices with eigenvalues as non-zero entries.

The eigenvalues of the orange matrices are inferred with hypernetworks $\mathcal{N}=\mathcal{N}(\mathbf{x}_t)$. Since $\mathcal{N}$ yield different combinations of eigenvalues for different datasets, MTL2L becomes context aware and self-modifies its optimisation rules based on input data. It is natural for MTL2L to support multi-task learning. During meta-training, we expose MTL2L to different datasets as shown in **Figure 3(b)** – the aim is to represent each task structure as different combinations of eigenvalues. During meta-testing, the eigenvalues enable MTL2L to compare the task structure of the input data to those learnt during meta-training. This enables MTL2L to update base learners on an unseen dataset at the meta-testing phase.

4. Experiments – Learning to learn image classification

We followed Andrychowicz et al. (2016) and employed neural optimisers to update MLPs. The neural optimisers were meta-trained to update the MLPs for 100 steps, but during meta-testing, they updated MLPs for the much longer 1000 steps. Base on this setting, we present two experimental scenarios as below.

Scenario 1 : <u>To test under the common meta-training assumption</u>
First meta-train neural optimisers to update MLP learners to classify images on one single dataset; and then meta-test by employing the learnt neural optimisers to update MLP learners to classify *the identical dataset* used in meta-training.

Scenario 2 : <u>To test for input-domain heterogeneity</u>
Unlike **Scenario 1**, meta-test to update learners to classify *an unseen dataset*.

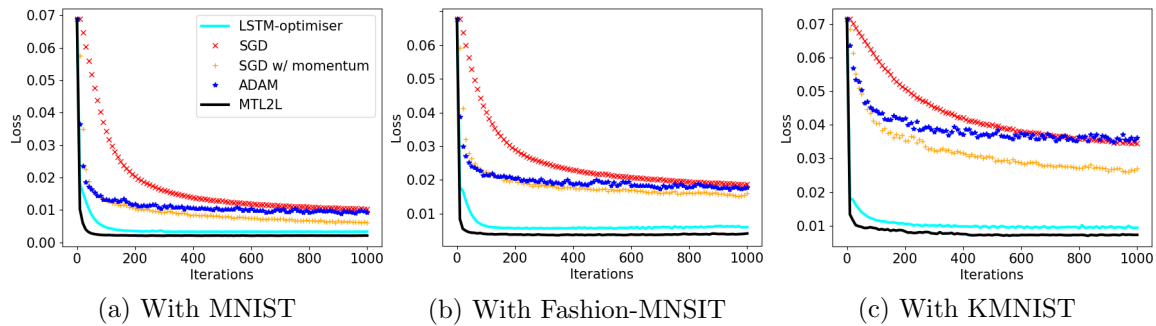(a) With MNIST   (b) With Fashion-MNSIT   (c) With KMNIST

Figure 4: Meta-testing on the same single domain as meta-training

We tested MTL2Ls against LSTM-optimisers, and against the hand-designed optimisation algorithms of vanilla SGD, SGD with momentum 0.9, and ADAM; all with learning rates 0.01. We presented our results as the learning curves of the learners. The results were colour-coded. We use black solid lines for MTL2Ls, cyan solid lines for LSTM-optimisers, red '×' for vanilla SGD, yellow '+' for SGD with momentum, and blue '⋆' for ADAM.

We upated the MLPs on MNIST (LeCun et al., 1998), Fashion-MNIST (Xiao et al., 2017), KMNIST (Clanuwat et al., 2018), and Cifar10 (Krizhevsky, 2009). Due to space constraint, we refer the readers to **Appendix B** for descriptions on the base learner, descriptions on the datasets, hyper-parameters for the neural optimisers, and choices of hypernetworks $\mathcal{N}$s (see **Section 3**) for MTL2Ls.

The results for **Scenario 1** are shown in **Figure 4**. For all datasets, the neural optimisers updated learners more rapidly than their handcrafted counterparts. MTL2Ls also converged the learners more efficiently and yielded lower losses than LSTM-optimisers.

The results for **Scenario 2** are shown in **Figure 5**. Since LSTM-optimisers only supported single-task learning, they were meta-trained to update MLPs for KMNIST. In contrast and as MTL2Ls naturally supports multi-task learning, we meta-trained them to update MLPs on both Fashion-MNIST and KMNIST. During meta-testing, all optimisers updated MLPs on a modified re-sized greyscale Cifar10 dataset (see the synthesis procedure in **Appendix B.2**). As shown in the figure, LSTM-optimisers exhibited instability with deteriorating performances whereas MTL2Ls displayed stability.
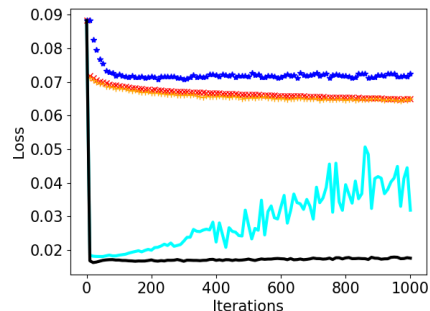

Figure 5: Meta-tested on an unseen Modified Cifar10 dataset

It was also interesting to observe the *timing* with which the LSTM-optimisers exhibited instability. As shown in **Figure 5**, the LSTM-optimiser instability was less severe in the first 400 iterations – the loss curve upraised but with no volatile oscillations. The volatility was only amplified after the first 400 iterations. This showed that there existed some underlying similarities between the task structures for updating learners on KMNIST and on the modified Cifar10. Hence, we conjectured that LSTM-optimisers were actually able to capture the "overall trends" in updating learners for an unseen dataset, but with imprecision. Furthermore, such imprecision introduced errors which reiterated in the LSTM and resulted in the eventual deteriorating performances.

5

5. Related Work

In this present work, we extended on the seminal paper of Andrychowicz et al. (2016) which explored meta-learning via learning to optimise. As stated in Equation (2), Andrychowicz et al. (2016) proposed to learn alternative rules to replace the traditional gradient descent methods. Their work has also been extended by the *LSTM-metalearner* (Ravi and Larochelle, 2017) and *Meta-SGD* Li et al. (2017) to learn auxiliary components compatible with the gradient descent regime[2].

Learning to optimise is one approach to meta-learning. Alternatively, there are metric-based approaches (Vinyals et al., 2016) which learn a kernel function to estimate the similarity between two sampled data. There are also model-based approaches (Munkhdalai and Yu, 2017) that combine weights updated at different timescales to synchronise different levels of abstracted features.

Besides the task of L2L, there are different ways to formulate the concept of learning quickly. For instance, *few-shot learning* (Vinyals et al., 2016) trains a network to map unlabelled data to their labels based on a small labelled support set.

Model-agnostic meta-learning (MAML) (Finn et al., 2017) is a meta-learning technique that addresses both input-domain heterogeneity and learner architecture heterogeneity (see **Figure** 1). Instead of training a neural optimiser from the scratch, MAML adopts a double-loop optimisation setup with hand-designed optimisation algorithms. The inner loop finds new candidate weights for the learner, and the outer loop aggregates all candidates to find an optimal update. Hence MAML optimises the parametric initialisation for the least amount of updates to converge a learner.

The *hierarchically structured meta-learning* (HSML) (Yao et al., 2019) extended on the idea of MAML. HSML jointly trains a task clustering network to learn features specific to datasets as parametric initialisation for the base learner.

6. Conclusion

In this paper, we showed that neural optimisers are able to perform beyond the common meta-learning assumption: *to meta-test on the single identical input-domain used during meta-training*. We introduced our novel context aware neural optimiser **Multi-task Learning to Learn (MTL2L)** with synaptic connections described with SVD and hypernetworks. MTL2Ls encoded input features as eigenvalues for their synapses to self-modify its optimisation rules. In our experiments, we meta-trained our MTL2L on Fashion-MNIST and KMNIST, then meta-tested it on an unseen modified Cifar10 dataset. We demonstrated that MTL2Ls successfully converged learners on an unseen input-domain, thereby making neural optimisers more generally applicable.

Acknowledgments

---

2. An extended Related Work section can be found in **Appendix** A.

References

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.

Rich Caruana. **Multitask Learning**. *Machine learning*, 28(1):41–75, 1997.

Yutian Chen, Matthew W Hoffman, Sergio Gómez Colmenarejo, Misha Denil, Timothy P Lillicrap, Matt Botvinick, and Nando Freitas. **Learning to Learn Without Gradient Descent by Gradient Descent**. In *International Conference on Machine Learning*, pages 748–756, 2017.

Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. **Deep Learning for Classical Japanese Literature**. In *NeurIPS Workshop on Machine Learning for Creativity and Design*, 2018.

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.

Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. **Learning to Forget: Continual Prediction with LSTM**. In *International Conference on Artificial Neural Networks*, pages 850–855, 1999.

David Ha, Andrew Dai, and Quoc V Le. **Hypernetworks**. In *International Conference on Learning Representations*, 2016.

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.

Sepp Hochreiter and Jürgen Schmidhuber. **Long Short-Term Memory**. *Neural computation*, 9(8):1735–1780, 1997.

Diederik P Kingma and Jimmy Ba. **ADAM: A Method for Stochastic Optimization**. In *International Conference on Learning Representations*, 2014.

Alex Krizhevsky. **Learning Multiple Layers of Features from Tiny Images**, Technical Report, 2009.

Nicholas I-Hsien Kuo, Mehrtash Harandi, Nicolas Fourrier, Christian Walder, Gabriela Ferraro, and Hanna Suominen. **M2SGD: Learning to Learn Important Weights**. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2020.

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.

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.

Yuan Luo. **Recurrent Neural Networks for Classifying Relations in Clinical Notes**. *Journal of biomedical informatics*, 72:85–95, 2017.

Kaifeng Lv, Shunhua Jiang, and Jian Li. **Learning Gradient Descent: Better Generalisation and Longer Horizons**. In *International Conference on Machine Learning*, 2017.

Luke Metz, Niru Maheswaranathan, Jeremy Nixon, Daniel Freeman, and Jascha Sohl-Dickstein. **Understanding and Correcting Pathologies in the Training of Learned Optimizers**. In *International Conference on Machine Learning*, pages 4556–4565, 2019.

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

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.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. **Wavenet: A Generative Model for Raw Audio**. *The Speech Synthesis Workshop in INTERSPEECH*, 2016.

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

Larry A Rendell, Raj Sheshu, and David K Tcheng. **Layered Concept-Learning and Dynamically Variable Bias Management**. In *International Joint Conferences on Artificial Intelligence*, pages 308–314, 1987.

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

G. van Rossum. **Python Tutorial**. Technical Report CS-R9526, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, May 1995.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. **Attention is All You Need**. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. **Matching Networks for One Shot Learning**. In *Advances in Neural Information Processing Systems*, pages 3630–3638, 2016.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake Vand erPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1. 0 Contributors. **SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python**. *Nature Methods*, 17:261–272, 2020. doi: https://doi.org/10.1038/s41592-019-0686-2.

Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. **Learning to Reinforcement Learn**. *CogSci*, 2016.

Olga Wichrowska, Niru Maheswaranathan, Matthew W Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Nando de Freitas, and Jascha Sohl-Dickstein. **Learned Optimisers that Scale and Generalise**. In *International Conference on Machine Learning*, 2017.

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

Huaxiu Yao, Ying Wei, Junzhou Huang, and Zhenhui Li. **Hierarchically Structured Meta-learning**. In *International Conference on Machine Learning*, pages 7045–7054, 2019.

Barret Zoph and Quoc V Le. **Neural architecture search with reinforcement learning**. In *International Conference on Learning Representations*, 2016.

Appendix A. Related Work (part 2)

Our focus in **Section 5** was to highlight different approaches in meta-learning. Here and as requested by our reviewers, we present an additional literature review on studies that extended the original work of Andrychowicz et al. (2016).

Remedies to Low Adaptability in Learnt Optimisers   As mentioned in the **Introduction**, it is not well studied whether neural optimisers are capable of updating base learners of a different architecture during the meta-testing phase.

   As demonstrated in their own work, Andrychowicz et al. (2016) showed (in Section 3.2 page 5 of that paper) that a learnt LSTM-optimiser was not generally applicable to changes made in activation functions. They changed the activation function of a MLP base learner from `tanh` to `ReLU` and showed that the modified MLP was updated unsuccessfully. This negative result was again brought up by the original panel of authors in Wichrowska et al. (2017) and was also independently studied by Lv et al. (2017). The two mentioned papers addressed different solutions to overcome this problem.

In Lv et al. (2017), the authors solved this issue by taking inspiration from the hand-crafted adaptive optimisation algorithms. They explicitly modified the input of their RNN-optimiser similarly to ADAM (Kingma and Ba, 2014) to allow current derivatives to be reduced slightly through dependencies on past derivatives. Whereas in Wichrowska et al. (2017), the authors presented a solution similar to processes of relation classification in text (Luo, 2017). They presented a hierarchical multi-layered RNN-optimiser capable of abstracting information of base learner gradients at different levels.

Unlike RNN-optimisers, low adaptability is much harder to be remedied in non-RNN-based learnt optimisers. For instance, Meta-SGD (Li et al., 2017) trains a unique learning rate per parameter for the base learner. Hence, the amount of parameters of the base learner becomes a hyperparameter of the neural optimiser. It is thus inapplicable to meta-test Meta-SGD on ResNet20s if it was meta-trained to update MLPs.

Unable to Handle Large Amount of Descent Steps As mentioned in **Section 4**, Andrychowicz et al. (2016) only employed LSTM-optimisers to update MLP base learners for 100 steps. Thus whether LSTM-optimisers are capable to handle large amount of descent steps has been a research topic with much attention in the community. Again, both Lv et al. (2017) and Wichrowska et al. (2017) have shown that the vanilla LSTM-optimiser was incapable of updating the base learner for a prolonged duration, and this could be solved by Lv et al. (2017) through the adaptive gradient modification, and by Wichrowska et al. (2017) through the hierarchical abstraction architecture.

A similar study was also conducted for Meta-SGD. Metz et al. (2019) analysed the maximum eigenvalue of the objective loss with respect to the meta-learner parameters. They took a variational bound approach to prevent gradient explosion in the training of the meta-learner and remedied biased introduced by truncated backpropagation through time. However, Kuo et al. (2020) also showed that, without modification, naive Meta-SGD was able to update base learners up to 5000 steps.

Other Types of Learnt Optimisers In this paper, we have mainly focused on learnt supervised learning optimisation algorithms. However, a large amount of research has also been devoted to learn reinforcement learning optimisation algorithms.

For instance, the original panel of authors of Andrychowicz et al. (2016) have also extended the LSTM-optimiser under the reinforcement learning (RL) setting in Chen et al. (2017). That is, they trained an RNN to learn RL. Similarly and as shown in Wang et al. (2016), the learning of a RL algorithm could also be learnt by another independent RL algorithm. Their key concept was to use a standard RL to train an RNN which implements its own free-standing RL procedure. The latter idea has also been implemented in an adjacent field of meta-learning. In neural architecture search, Zoph and Le (2016) have shown that an RNN could be trained with RL to maximised the expected accuracy of the generated architectures on a validation set.

Appendix B. Experimental Details for Section 4

This appendix documents the experimental details for **Section 4**. Information will be provided, in the order of

- **(1)** : The MLP Learner,
- **(2)** : Dataset description,
- **(3)** : Gradient pre-processing,
- **(4)** : The meta-training hyper-parametric setups,
- **(5)** : Choices of orthonormal SVD matrices, hypernetwork $\mathscr{N}$s, and the MTL2L multi-task setup.

B.1 The MLP Learner

Our MLP learners were simple. They consisted one linear layer followed by a ReLU activation, and one softmax layer. The linear layer has input dimension 784, the softmax layer has output dimension 10, and the hidden dimension connecting the linear layer and softmax layer was set to 32. The specific reasons for numbers 784 and 10 will be made clear in the **next section** for data description; and the MLP learners had 25.4K parameters.

B.2 Dataset description

We upated the MLP learners on MNIST (LeCun et al., 1998), Fashion-MNIST (Xiao et al., 2017), KMNIST (Clanuwat et al., 2018), and Cifar10 (Krizhevsky, 2009). We will first discuss the three MNIST datasets, and then separately address Cifar10.

The MNIST dataset contains 60,000 training and 10,000 test images in 10 classes of $28{\times}28$ greyscale handwritten digits. This is a popular dataset among the machine learning community and commonly used as a benchmark to validate their algorithms. The subsequent datasets of Fashion-MNIST and KMNIST were crafted in a similar style to that of the original MNIST dataset. Both Fashion-MNIST and KMNIST has 60,000 training and 10,000 test greyscale images with 10 classes of items on $28{\times}28$ pixels. While Fashion-MNIST consists iamges of fashion items, KMNIST consists kuzushiji (崩し字) versions of hiragana (平仮名) characters from classical Japanese literature.

We choose MNIST, Fashion-MNIST, and KMNIST for the similarities in their dataset constructions. All images were presented as a vector of $784(= 28 \times 28)$ pixels to the MLP learners. Hence the input dimension of the MLPs was 784, and the output dimension was 10 due to the amount of labels.

The CIFAR-10 dataset is slightly more challanging. It consists $32{\times}32$ RGB-colour images in 10 classes, and there are 50,000 training images and 10,000 test images. In order to fit these images in the MLP learner of **Appendix B.1**, we created a synthesised Cifar10 dataset consisting of greyscale $28{\times}28$ pixel images. The images were made greyscale by combining the coloured filters in the ratio of 0.30R + 0.59B + 0.11G. Furthermore, the images were made $28{\times}28$ by only selecting the first 28 rows and the first 28 columns.

B.3 Gradient pre-processing

This subsection reiterates the following content from Andrychowicz et al. (2016) for the completeness of this paper. The original content can be found on page 11 of their paper under "A Gradient preprocessing".

Andrychowicz et al. (2016) proposed to pre-process the neural optimiser's input gradient $\nabla_\theta \mathcal{L}_{t+1}(\theta_t)$ (which we simplify as ∇ in this subsection) with

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

Each element of the native gradient $\nabla^{(j)}$ for $j = 1 \ldots |\theta_t|$ is pre-processed as a pair of values. The hyper-parameter p controls how small gradients are disregarded, and defaults to 10 in all of their and in all of our experiments. This scheme was devised to ensure that the magnitudes of every dimension is on the same order. Andrychowicz et al. (2016) 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*".

B.4 The meta-training hyper-parametric setups

The RNNs of the LSTM-optimisers and MTL2Ls were formulated as documented in **Table 1**; and both were trained with the ADAM optimiser with learning rate 0.001. Following Andrychowicz et al. (2016), we set the RNNs as 2-layer deep with hidden dimension 20. The inputs to the RNNs were the pre-processed gradients of **Appendix B.3**; they were treated as a data of batch size $|\theta_t|$ with feature size (input dimension) 2. Due to this reason, the input dimension of the RNNs of both the LSTM-optimisers and MTL2Ls were of size 2. Below, we applied the additional hyper-parametric setups to **Algorithm 1** for meta-training.

For all LSTM-optimisers, we set $\Xi = 5$, $\mathcal{Q} = 20$, and $\mathcal{S} = 100$. The combination of these hyper-parameters meant that, LSTM-optimisers were trained to update 20 MLP learners for 100 steps; and that the LSTMs were unrolled every 5 steps for updates. Thus they memorised 5 continual steps of the optimisation trajectory for the MLP learners.

MTL2Ls of experimental **Scenario 1** were setup identically as the LSTM-optimisers. MT2Ls of experimental **Scenario 2** were setup differently to enable multi-task learning. For **Scenario 2**, we set $\Xi = 5$, $\mathcal{Q} = 30$, and $\mathcal{S} = 100$. It will be made clear in **Appendix B.5** for the reason of the prolonged learner-trial $\mathcal{Q}$.

B.5 Choices of orthonormal SVD matrices, hypernetwork $\mathcal{N}$s, and the MTL2L multi-task setup

The main novelty in our MTL2L neural optimiser are the self-modifying synapses in Equations (13) and (14). In this subsection, we addresses how to configure the MTL2L SVD synapses and setup the MTL2L multi-task learning. We will discuss in the order of, the orthonormal SVD matrix selections, the hypernetwork $\mathcal{N}$s for experimental **Scenarios 1** and **2**, and then the MTL2L multi-task learning setups for **Scenario 2**.

B.5.1 Orthonormal matrix selection for SVD synapses

Equations (13) and (14) describe the MTL2L synapses with SVD. Abiding to the SVD formulation, the purple components are orthonormal matrices, whereas the orange components are diagonal matrices with eigenvalues (or singular values, to be more precise). We emphasise again that the purple matrices are fixed prior to training, and only the orange components are trained with hypernetworks. More on the orange components will be addressed in the following subsubsections.

Any means of orthonormal matrix generation is viable. The authors of this manuscript coded in Python (van Rossum, 1995), and we choose to use the SciPy library (Virtanen et al., 2020). Refer to https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ortho_group.html for guidelines.

B.5.2 For Scenario 1

The hypernetwork $\mathcal{N}$s for **Scenario 1** are MLPs. These hypernetwork-MLPs consisted two linear layers – the first linear layer was followed by a ReLU activation, then followed by the second linear layer. The hypernetwork-MLPs received the images of as vectors of 784 pixels. Hence, the input dimension of the first linear layer was set to 784. In addition, the output dimension of the second linear layer was 20. This is because the hypernetwork-MLPs were employed to infer for the eigenvalues of the SVD synapses of MTL2Ls. Hence, the output dimension of the second linear layer was required to be identical to the hidden dimension of the MTL2Ls. Last, the hidden dimension which connects the first and the second linear layer of the hypernetwork-MLPs were set to 32.

B.5.3 For Scenario 2

As mentioned in **Section 4**, we set MTL2Ls to perform multi-task learning on Fashion-MNIST and on KMNIST for the meta-training phase of experimental **Scenario 2**. As discussed in **Appendix B.4**, the learner-trial $\mathcal{Q}$ was set as the slightly longer 30 – 15 trials were set to update the MLP learners for the Fashion-MNIST dataset, and the remaining 15 were reserved to update the MLP learners on KMNIST.

During meta-training, we alternated between updating the MLP learners for Fashion-MNIST and for KMNIST as illustrated in **Figure 3(b)**. That is, when $\mathcal{Q} = 1, 3, \ldots, 29$, MTL2Ls updated MLP learners to classify on Fashion-MNIST; while for even-number $\mathcal{Q}$s, MTL2Ls updated MLP learners to classify on KMNIST.

The hypernetwork $\mathcal{N}$s for **Scenario 2** were similar to those in **Scenario 1** but with a slight modification. We formulated them as

$$\mathcal{N}_t = 0.9\mathcal{N}_{t-1} + 0.1\text{MLP}(\mathbf{x}_t), \text{with} \tag{17}$$

$\mathcal{N}_0 = \vec{\mathbf{0}} \in \mathbb{R}^{20}$. The hypernetwork-MLPs employed in Equation (17) were identical to those employed in **Scenario 1**. However in **Scenario 2**, we treated the propagation of the eigenvalues of the SVD synapses of MTL2Ls like momentum (Nesterov, 1983). This was done to increase training stability for MTL2L.

As a final and less significant note, we injected small-sized perturbations to the parameters of the MLP learners during the meta-training phase. This is done to incorporate more noise during the training of MTL2Ls.