Meta-Learner LSTM for Few Shot Learning

Stanislas Furrer*, Yiğit Efe Erginbaş† and Mert Kayaalp‡
School of Engineering (STI)
École polytechnique fédérale de Lausanne (EPFL)
Lausanne, Switzerland
Email: *stanislas.furrer@epfl.ch, †yigit.erginbas@epfl.ch, ‡mert.kayaalp@epfl.ch

Abstract—**Recently, large-scale few-shot learning (FSL) has become topical. It is an important challenge in deep learning as large number of training instances may not be available in many real-world settings. Learning to accurately classify objects from a few training examples is often unfeasible due to over-fitting effects. To overcome this challenge, meta-learning based few-shot learning has been suggested to acquire quick knowledge from few examples. Ravi & Larochelle [1] has addressed the weakness of neutral networks trained with gradient-based optimization on the few-shot learning problem by framing the problem within a meta-learning setting. They use an LSTM-based meta-learner optimizer to learn the exact optimization algorithm used to train another learner neural network classifier in the few-shot regime. In this paper, we expand the performance analysis of their algorithm. By comparing with other state of the art techniques, we demonstrate the strengths and weaknesses. Additionally, we replace the LSTM form with other recurrent cell structures used in the literature and compare their performance. Finally, we illustrate the cross domain performance of the algorithm.**

I. Introduction

The availability of large quantities of labelled data has enabled deep learning methods to achieve impressive breakthroughs in several tasks related to artificial intelligence, such as speech recognition [2], object recognition [3] and machine translation [4]. These systems, however, usually require a large amount of labelled data, which can be impractical or expensive to acquire. Limited labelled data lead to over-fitting and generalization issues in classical deep learning approaches. On the other hand, existing evidence suggests that human visual system is capable of effectively operating in small data regime: humans can learn new concepts from a few samples, by leveraging prior knowledge and context. The problem of learning new concepts with small number of labelled data points is usually referred as few-shot learning [5]–[7].

Vinyals et al. [8] proposed matching networks that combine both embedding and classification to form an end to end differentiable nearest neighbors classifier. A much simpler and scalable technique is proposed by Snell et al. [9] and is called Prototypical Network. The algorithm learns a metric space in which classification can be performed by computing distances to prototype representations.

In this paper we study the performance of the meta learning approach proposed by Ravi & Larochelle [1]. They addressed the few shot learning problem with a meta-learning approach which works by learning a parameterized function that embeds a variety of learning tasks and can generalize to new ones. They proposed an LSTM based meta-learner model to learn the exact optimization algorithm used to train another learner neural network classifier in the few-shot regime. Their meta-learning technique leads to learning the appropriate parameter updates together with a general initialization of the learner, which allows quick convergence in the training.

II. Meta-Learner LSTM for Few Shot Learning

Traditional gradient based optimization methods fail in few-shot learning settings because the algorithms such as AdaGrad, AdaDelta, Adam might need thousands of iterations in order to converge if the step-sizes are not well-chosen. In order to resolve this issue, Ravi & Larochelle has proposed an LSTM-based meta-learner that learns the best gradient-based update rule for the learner parameters.

A. Model Description

Traditional optimization algorithms are variants of gradient descent of the form

$$\theta_t = \theta_{t-1} - \alpha_t \nabla_{\theta_{t-1}} L_t \tag{1}$$

The main observation of this paper is that this gradient descent update is similar to cell state update in an LSTM.

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c_t} \tag{2}$$

If $f_t = 1, c_{t-1} = \theta_{t-1}, i_t = \alpha_t, \tilde{c_t} = -\nabla_{\theta_{t-1}} L_t$, one recovers the standard gradient descent update. However, for faster and better convergence, one can freely choose f_t and i_t while keeping $c_{t-1} = \theta_{t-1}$ and $\tilde{c_t} = -\nabla_{\theta_{t-1}} L_t$ intact. Therefore, Ravi & Larochelle propose training a meta-learner LSTM for learning the update rules for i_t (the input gate) and f_t, (the forget gate):

$$\begin{cases} i_t = \sigma(W_I[\nabla_{\theta_{t-1}} L_t, L_t, \theta_{t-1}, i_{t-1}] + b_I) \\ f_t = \sigma(W_F[\nabla_{\theta_{t-1}} L_t, L_t, \theta_{t-1}, f_{t-1}] + b_F) \end{cases} \tag{3}$$

where σ is the default sigmoid function. In addition to parameters $\{W_I, b_I, W_F, b_F\}$, the initial weights for the learner, c_o, are also meta-learned. Note that if there was no budget for adaptation to tasks (no gradient step for specification to task), c_o would correspond to the optimum weight that minimizes the expected risk of multiple tasks.

B. Parameter sharing and Pre-processing

Since the number of weight parameters can be in the order of millions, employing different meta learner parameters for each of them can be computationally expensive. Therefore, LSTM parameters are shared and same update rule has been applied to all learner parameters. Furthermore, meta-learner is fed with a pre-processed version of the gradients and losses in the form proposed in [10] so that the magnitude and sign information would be separated. Pre-processing is formulated as in Equation 4 and $p = 10$ has been used in the experiments, as proposed in [1].

$$x = \begin{cases} (\log(|x|)/p, \text{sgn}(x)) & |x| \geq e^{-p} \\ (-1, e^p x) & o.w. \end{cases} \tag{4}$$

III. Experimental Results

A. Experimental Setup

The implementation has been done in Python using the Tensorflow API as it enables us to speed up the training process besides simplifying the design of the pipeline architecture. Training with the default settings takes nearly 2.5 hours on a single Tesla V100 while occupying 2GB GPU memory. We comply with the implementation details and hyper-parameter selections given in the paper by Ravi & Larochelle. Although the authors have also published their code with the paper, we have only used it as a reference while implementing our code since their code is in Lua [11].

Algorithm 1: Meta-Training

```
Inputs: Learner M with parameters $\theta$,
        Meta-Learner $R$ with parameters $\Theta$
$\Theta_0 \leftarrow$ random initialization
for n=1,N do
    $D_{train}, D_{test} \leftarrow$ random sets from $D_{meta-train}$
    $\theta_0 \leftarrow c_0$
    for t=1,T do
        $X_t, Y_t \leftarrow$ random batch from $D_{train}$
        $\mathcal{L}_t \leftarrow \mathcal{L}(M(X_t, \theta_{t-1}), Y_t)$
        $\theta_t \leftarrow R(\nabla_{\theta_{t-1}} \mathcal{L}_t, \mathcal{L}_t)$
    end
    $X, Y \leftarrow D_{test}$
    $\mathcal{L}_{test} \leftarrow \mathcal{L}(M(X, \theta_T), Y)$
    Update $\Theta_n$ by $\nabla_{\Theta_{n-1}} \mathcal{L}_{test}$ using Adam optimizer
end
```

All of the data sets are split into three subsets that contain images from different classes and they are referred as *training*, *validation* and *testing* data. The adaptation process of the meta-learner LSTM consists of 3 different stages: *meta-training*, *meta-validation* and *meta-testing*. We start with the meta-training stage and and train the meta-learner for 100,000 episodes. In each episode, we randomly sample 5 classes from the *meta-training* data, and in each class, we pick 1 or 5 (depending on #-of-shots) instances for the *episode-training* set and 15 for the *episode-evaluation* set. The learner applies a gradient descent on the *episode-training* set using the rule determined by the meta-learner and then the meta-learner is evaluated according to the performance of the learner on the *episode-evaluation* set. In order to update the parameters of the meta-learner, a back-propagation is applied on the evaluation loss. After each 500 episodes of *meta-training*, we switch to a *meta-validation* stage and apply the same episode procedure using the *meta-validation* data without updating the parameters of the meta-learner. After all meta-training process is completed, we test the final performance of the meta-learner on *meta-testing* data for 600 episodes and use the results as the performance measure.

B. Accuracy Comparison for Different Data Sets

Ravi & Larochelle test their proposed algorithm using the mini-ImageNet data set, which is a smaller version of the original Imagenet data set and designed to be used in evaluating the performance of few-shot learning algorithms. They provide the results that they have obtained in both 1-shot and 5-shot learning settings and compare it with the state-of-art algorithms at the time. To validate the correctness of our implementation, we start by comparing our results to the reported numbers for the mini-ImageNet data set.

TABLE I: Accuracy obtained on mini-Imagenet with different architectures. We report the mean of 600 randomly generated test episodes as well as the 95% confidence intervals.

	5-shot	1-shot
Meta-Learner LSTM (Ravi & Larochelle)	60.6 $\pm$0.71	43.4 $\pm$0.77
Meta-Learner LSTM (Our implementation)	61.1 $\pm$0.69	44.0 $\pm$0.73
MatchingNet [8]	55.3 $\pm$0.73	43.6 $\pm$0.84
MAML [12]	62.7 $\pm$0.71	46.5 $\pm$0.82
ProtoNet [9]	66.7 $\pm$0.68	47.7 $\pm$0.84

Although the authors only preferred testing on the mini-ImageNet data, there are also other data sets widely used for bench-marking few-shot learners and we have chosen to use the following two: The CUB data set consisting of colored images of different bird species and the Omniglot data set consisting of gray-scale images of various letters. Since images in the Omniglot data set are easier to be classified compared to mini-ImageNet and CUB, the convention in the literature is to work with 1-shot classification tasks.

TABLE II: Accuracy obtained on CUB & Omniglot with different architectures. We report the mean of 600 randomly generated test episodes and the 95% confidence intervals.

	CUB	Omniglot
Meta-Learner LSTM (Our implementation)	51.6 $\pm$0.72	94.7 $\pm$0.44
MatchingNet [8]	59.3 $\pm$0.75	no data
MAML [12]	75.8 $\pm$0.76	98.7 $\pm$0.40
ProtoNet [9]	76.4 $\pm$0.64	98.8 $\pm$0.34

As the results show, the proposed algorithm is successful in achieving a performance similar to state-of-art approaches for the mini-ImageNet data set. However, the results do not generalize to other data sets we test on. The reason might be that the image classes in both CUB and Omniglot data sets are similar, while the mini-ImageNet classes have quite different attributes. This suggests that Meta-Learner LSTM technique is not suitable for fine-grained classification problems that require the extraction of small details.

C. Different Recurrent Cell Structures

***Adaptive SGD*:** In order to have a baseline to compare the results obtained using recurrent cell structures, we start by implementing a basic non-recurrent adaptive SGD optimizer. This structure can only learn the initial parameters of the learner and the SGD step-size that will be used to update the parameters of the learner during its training.

***Complete-LSTM*:** Ravi & Larochelle present their algorithm as an LSTM meta-learner since its structure resembles the LSTM cell previously proposed in [13]. However, the standard LSTM also has an output gate that produces the next hidden state, besides the input and forget gates described in a previous section. Therefore, we have implemented a modified LSTM meta-learner by including an additional output gate and referred this new structure as *Complete-LSTM*.

***GRU*:** There is also another recurrent approach in the literature called *GRU cells* which is used to reduce the computational complexity of an LSTM cell while mostly preserving the performance [14]. Hence, we have also implemented a *GRU*-like structure using similar ideas.

TABLE III: Accuracy obtained on mini-ImageNet using meta-learners with different recurrent cell architectures. We report the results of 600 randomly generated test episodes.

Recurrent Cell	Accuracy (5-class, 5-shot)
Adaptive SGD	55.5 $\pm$0.70
LSTM	61.1 $\pm$0.69
Complete-LSTM	57.5 $\pm$0.70
GRU	59.7 $\pm$0.67

The results show that all of the recurrent structures perform better than a non-recurrent strategy. The reason for the *Complete-LSTM* structure to perform worse than the proposed version is the excessive complexity which causes over-fitting that will be discussed in the following section. The *GRU* also performs suboptimally, however this is due to its less complex structure and hence it can be preferred to the proposed structure for its reduced computational costs.

D. Over-fitting and Reducing the Model Complexity

When we implement the meta-learner structure as described in the paper, the test accuracy of the models starts to decrease after reaching a maximum. Although we have tried decreasing the learning rate or including gradient clippings as proposed in [15] for recurrent networks, the issue was still persistent. At the end, we have managed to solve this problem by reducing the hidden size of the meta-learner cell structure, and hence reducing the meta-learner model complexity. Since the issue only exists in the test results and has been solved by decreasing model complexity, we can relate it to over-fitting. The learning curves obtained for different hidden sizes can be found in Appendix.

E. Data Augmentation

In order to cope with the over-fitting problem, we have also tried implementing a data pre-processing stage before feeding the training data to the meta-learner/learner pair. This process mainly consists of random color jitters, crops and changes in the aspect ratios as proposed in [16]. By adding more noise and increasing the complexity of the data, we aim increasing the robustness of the trained meta-learner. When we compare the results obtained with and without a pre-processing stage, we observe that we have achieved this goal.

TABLE IV: Accuracy of default meta-learner LSTM model on default and augmented mini-ImageNet data. Augmentation is done with random crops and color jitters.

Data augmentation	Accuracy (5-class, 5-shot)
True	61.1 $\pm$0.69
False	57.7 $\pm$0.67

F. Cross-Domain Scenario

To further dig into the issue of domain differences, we have designed a scenario that provides domain shifts. After training the meta-learner model on either CUB or Omniglot data set, we apply tests over the mini-ImageNet data. Although the meta-learner LSTM learns to learn from the training set during the meta-training stage, the results show that meta-learner is not able to adapt to classes data are much different. We can further observe that the accuracy becomes lower as the domain difference gets larger. Since both CUB and mini-ImageNet data sets consist of colored images of physical entities, they include similar features and a transfer between them becomes possible up to some degree. However, the Omniglot data set consists of single-channel images of a different nature and the model transfer becomes harder.

TABLE V: Accuracy obtained by training meta-learners on CUB or Omniglot and testing on mini-ImageNet data. We report the results of 600 randomly generated test episodes.

Transfer	Accuracy 5-class, 5-shot
Omniglot $\rightarrow$ mini-ImageNet	32.3 $\pm$0.54
CUB $\rightarrow$ mini-ImageNet	44.0 $\pm$0.67

IV. Conclusion

In this project, we have investigated the meta-learner LSTM structure [1] in detail. In particular, besides reproducing the results that have been reported by authors, we have expanded the experiments to various data sets and recurrent cell structures. Moreover, we have solved the problem of decreasing accuracy after reaching a maximum, examined the effect of data augmentation in the accuracy and explored the scenario of cross-domain training-testing.

References

[1] S. Ravi and H. Larochelle, "Optimization as a model for few-shot learning," in *ICLR*, 2017.

[2] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *CoRR*, vol. abs/1609.03499, 2016.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[4] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, L. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, "Google's neural machine translation system: Bridging the gap between human and machine translation," *CoRR*, vol. abs/1609.08144, 2016.

[5] E. Bart and S. Ullman, "Cross-generalization: Learning novel classes from a single example by feature replacement," vol. 1, pp. 672– 679 vol. 1, 07 2005.

[6] M. Fink, "Object classification from a single example utilizing class relevance metrics," in *Advances in Neural Information Processing Systems 17* (L. K. Saul, Y. Weiss, and L. Bottou, eds.), pp. 449–456, MIT Press, 2005.

[7] Li Fei-Fei, R. Fergus, and P. Perona, "One-shot learning of object categories," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 4, pp. 594–611, 2006.

[8] O. Vinyals, C. Blundell, T. P. Lillicrap, K. Kavukcuoglu, and D. Wierstra, "Matching networks for one shot learning," *CoRR*, vol. abs/1606.04080, 2016.

[9] J. Snell, K. Swersky, and R. S. Zemel, "Prototypical networks for few-shot learning," *CoRR*, vol. abs/1703.05175, 2017.

[10] M. Andrychowicz, M. Denil, S. G. Colmenarejo, M. W. Hoffman, D. Pfau, T. Schaul, and N. de Freitas, "Learning to learn by gradient descent by gradient descent," *CoRR*, vol. abs/1606.04474, 2016.

[11] M. Dong, "meta-learning-lstm-pytorch." https://github.com/markdtw/meta-learning-lstm-pytorch, 2018.

[12] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," *CoRR*, vol. abs/1703.03400, 2017.

[13] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[14] M. Ravanelli, P. Brakel, M. Omologo, and Y. Bengio, "Light gated recurrent units for speech recognition," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, p. 92–102, Apr 2018.

[15] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *Proceedings of the 30th International Conference on Machine Learning* (S. Dasgupta and D. McAllester, eds.), vol. 28 of *Proceedings of Machine Learning Research*, (Atlanta, Georgia, USA), pp. 1310–1318, PMLR, 17–19 Jun 2013.

[16] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9, June 2015.

Appendix A
Learning Curves for Some of the Experiments

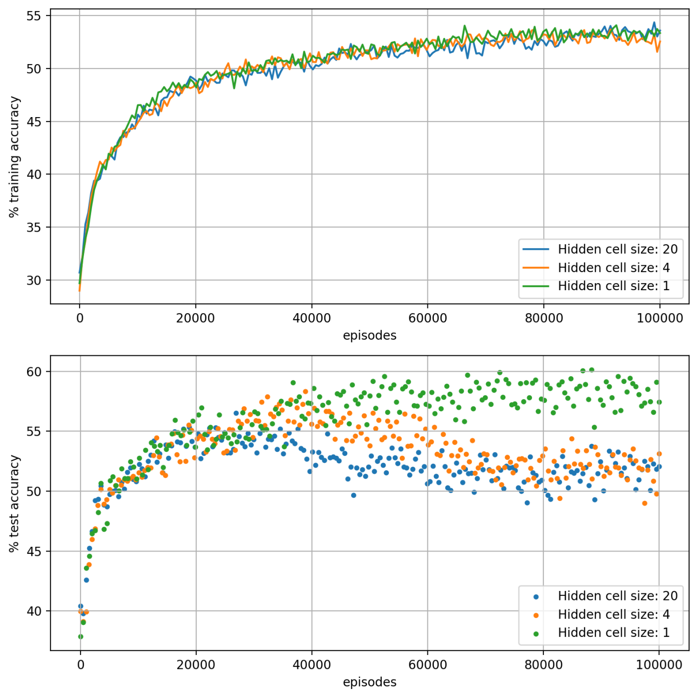

Fig. 1: The training and testing accuracy obtained by using different hidden sizes for the meta-learner architecture (proposed value: 20). The test accuracy starts to reduce after reaching a maximum when we use a large hidden size, and hence a too complex architecture. However, the training accuracy continues increasing with similar rates for all structures. This suggests that complex architectures are vulnerable to over-fitting.

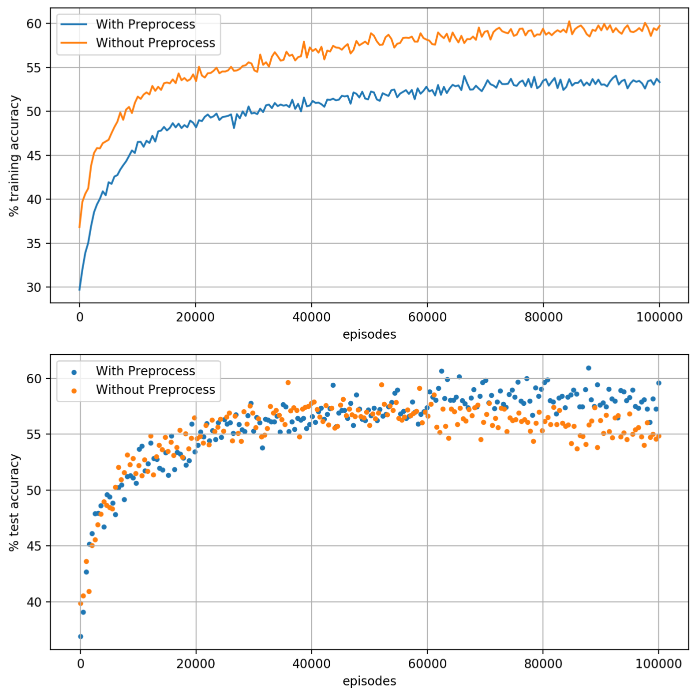

Fig. 2: The training and testing accuracy obtained by augmenting data with random crop and random jitter. When we do not augment data, the model fits too much to the training samples and the model cannot be generalized well enough on the test.

APPENDIX B
PARAMETER UPDATES FOR META-LEARNER GATES

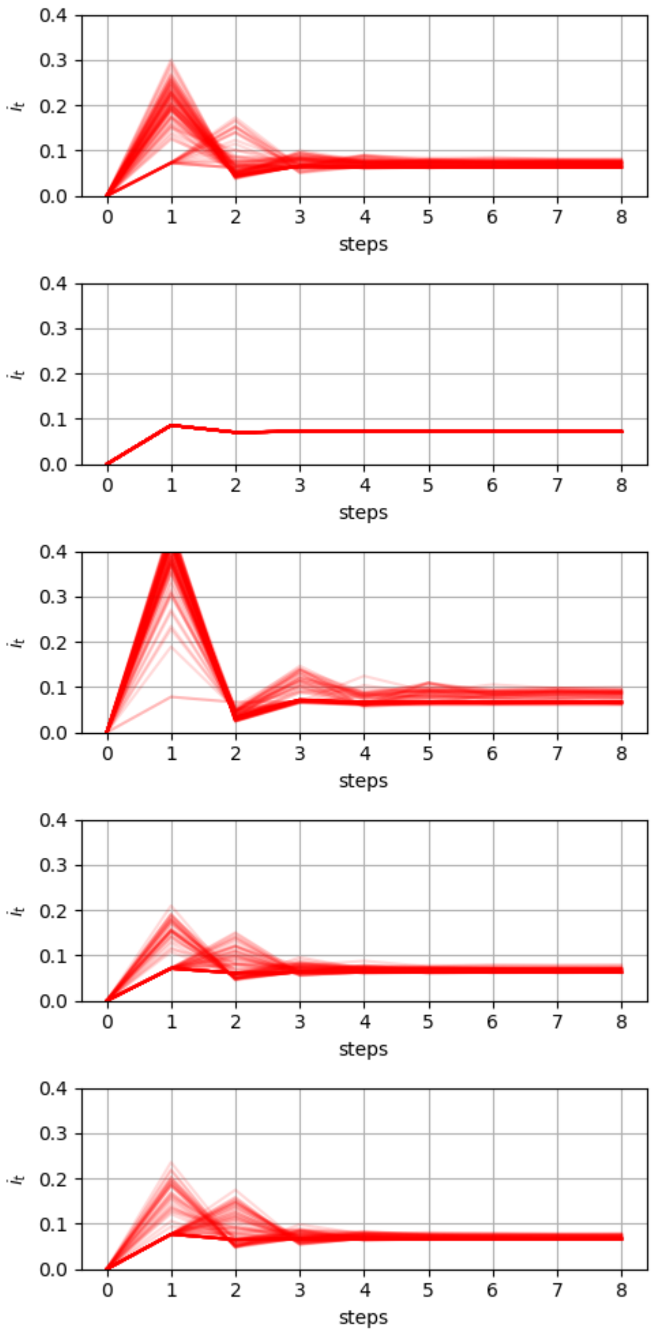

Fig. 3: The evolution of input gate values for randomly selected parameters. The vertical axis is the value of the parameter. The horizontal axis is the learning steps of the learner in each episode. Each plot corresponds to a different parameter. Each curve corresponds to a different test episode.

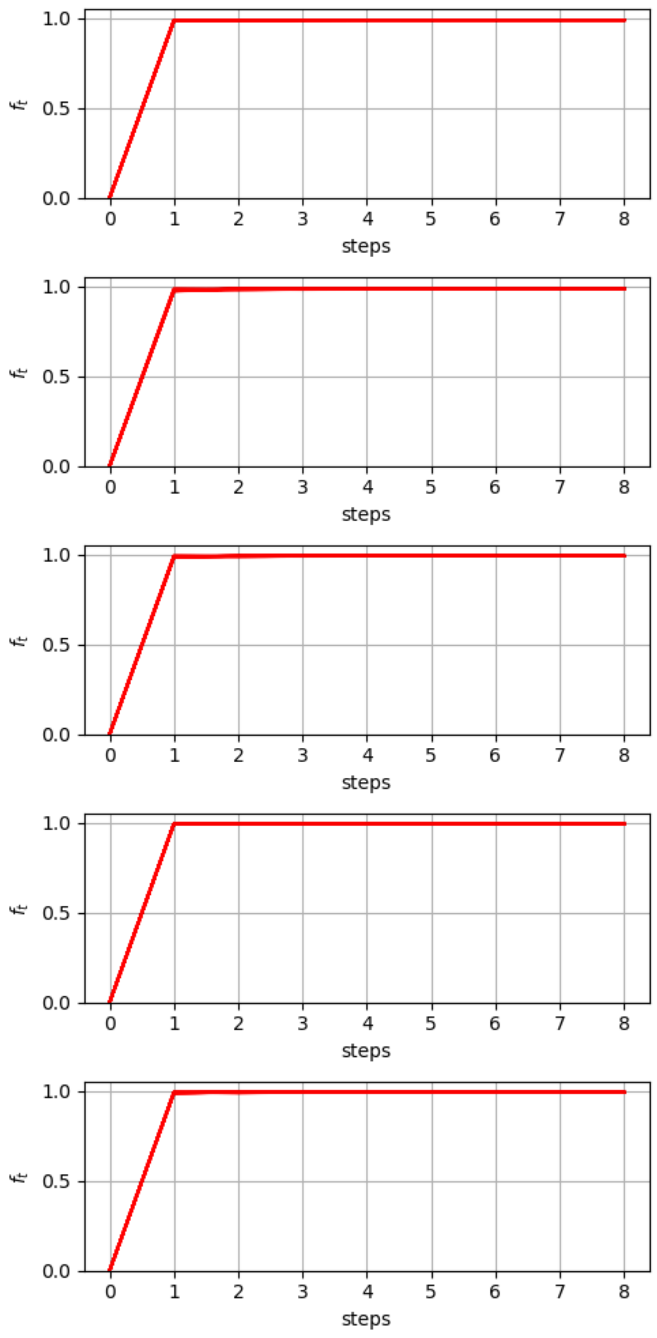

Fig. 4: The evolution of forget gate values for randomly selected parameters. The vertical axis is the value of the parameter. The horizontal axis is the learning steps of the learner in each episode. Each plot corresponds to a different parameter. Each curve corresponds to a different test episode.