REPRODUCIBILITY CHALLENGE: LA OPTIMIZER

John Joyce
jvjj1u19@soton.ac.uk
31417027

Ilias Kazantzidis
ik3n19@soton.ac.uk
31447538

Connah Romano-Scott
crs1u19@soton.ac.uk
28802284

1 INTRODUCTION

In optimisation, stochastic gradient descent (SGD) is a popular method for training neural networks, using 'minibatches' of data to update the network's weights. Improvements are often made upon SGD by either using acceleration/momentum or by altering the learning rate over time. In this paper, we discuss and attempt to reimplement a different form of improvement, applied directly on the weight space put forth by Zhang et al. (2019) which they dub the Lookahead (LA) optimiser.

LA uses two sets of weights for the network in question — one for 'fast weights' and the other for 'slow weights' — along with two unique hyper-parameters, the number of fast weight updates (steps) $k \in \mathbb{N}$ and slow weights learning rate $\alpha \in (0, 1]$. In each iteration, the slow weights are cached, then the fast weights are updated k times through SGD or with other 'inner-loop' optimisers, such as Adam. A vector from the slow weights to the resulting fast weights is then calculated, multiplied by α, and added slow weights. The fast weights are then reset to match the new values of the slow weights before the next iteration begins.

Zhang et al. in all their deep learning experiments claimed to improve convergence and performance over the inner optimiser, and additionally considered their optimiser robust to changes in the inner-loop optimiser hyper-parameters, the number of fast weight steps and the slow weights learning rate. We aim to reconstruct these experiments to verify these claims. However, there are two experiments which we will not reimplement due to computational resource constraints. In Section 2, we will compare the implementation of LA to its presentation in the original paper. In Section 3, we will attempt to reproduce experiments performed by Zhang et al., including classification on CIFAR and language modelling on Penn Treebank. Finally, we will conclude with Section 4.

2 IMPLEMENTATION OF LOOKAHEAD ALGORITHM

The LA algorithm is given in pseudocode in the original paper by Zhang et al., and an implementation using PyTorch is publicly available at `https://github.com/michaelrzhang/lookahead`. The code below is taken from this link and performs a single step of LA, which we will verify matches its given description and pseudocode.

```
1   def step(self, closure=None):
2       """Performs a single Lookahead optimization step.
3       Arguments: closure (callable, optional): A closure that reevaluates the model and returns the loss."""
4       loss = self.optimizer.step(closure)
5       self._la_step += 1
6       if self._la_step >= self._total_la_steps:
7           self._la_step = 0
8           # Lookahead and cache the current optimizer parameters
9           for group in self.optimizer.param_groups:
10              for p in group['params']:
11                  param_state = self.state[p]
12                  p.data.mul_(self.la_alpha).add_(1.0 - self.la_alpha, param_state['cached_params'])
13                  param_state['cached_params'].copy_(p.data)
14                  if self.pullback_momentum == "pullback":
15                      internal_momentum = self.optimizer.state[p]["momentum_buffer"]
16                      self.optimizer.state[p]["momentum_buffer"] = internal_momentum.mul_(self.la_alpha).add_(
17                          1.0 - self.la_alpha, param_state["cached_mom"])
18                      param_state["cached_mom"] = self.optimizer.state[p]["momentum_buffer"]
19                  elif self.pullback_momentum == "reset":
20                      self.optimizer.state[p]["momentum_buffer"] = torch.zeros_like(p.data)
21      return loss
```

Lines 4 and 5 perform a step on the fast weights using the inner optimiser. Line 6 checks whether it has been k steps since the last time the fast weights have been updated. If so, lines 9 and 10 iterate over each slow weight. On line 12, given a single slow weight p_i and fast weight q_i for some index i, the slow weight is updated as $p_i \leftarrow \alpha q_i + (1 - \alpha)p_i$. This update rule matches

the rule described in the pseudocode and in the original paper, since given a vector $\mathbf{r}$ from the full vector of slow weights $\mathbf{p}$ to the full vector of fast weights $\mathbf{q}$, the slow weights are updated as $\mathbf{p} + \alpha\mathbf{r} = \mathbf{p} + \alpha(\mathbf{q} - \mathbf{p}) \equiv \mathbf{p} + \alpha\mathbf{q} - \alpha\mathbf{p} \equiv \alpha\mathbf{q} + (1 - \alpha)\mathbf{p}$. The remainder of the code resets momentum where applicable, so we can see that the implementation matches the presentation. It is interesting to note that this code clarifies that a 'step' of LA is simply a step with the fast weights (and an update of the slow weights if applicable), rather than k fast weight steps and a single slow weight step. This is not specified in the original paper, but is relatively unimportant.

3 EXPERIMENTS

We will now attempt to re-implement experiments by Zhang et al. using their LA implementation. Section 3.1 discusses classification experiments on CIFAR-10 and CIFAR-100 and Section 3.2 discusses language modelling experiments on Penn Treebank. Our code is available at **`https://github.com/COMP6248-Reproducability-Challenge/LookaheadOptimizer`**.

3.1 CLASSIFICATION ON CIFAR-10 AND CIFAR-100

The CIFAR-10 and CIFAR-100 datasets (Krizhevsky, 2012) contain 10 and 100 classes respectively, each with a total of 60,000 32×32 colour images and train/test ratios of 5:1. Zhang et al. demonstrated the performance of the LA optimiser with various ResNet architectures over the CIFAR datasets. The primary results for these experiments can be found in Section 5.1 of their paper. With a training loss curve and a table summarising final validation accuracies, they show that LA consistently converges faster than SGD, AdamW and Polyak (averaged SGD) and that final validation accuracy is consistently better.

We aim to test their claims by re-implementing the training procedure for each optimiser according to the detailed models, parameters and augmentations (Appendix C.1 of their paper). They used a ResNet-18 implementation produced by Devries & Taylor (2017) (available at `https://github.com/uoguelph-mlrg/Cutout`) which has wider channels and more parameters than the original. For data augmentations, they followed the procedure by Zagoruyko & Komodakis (2016) where images are zero-padded with 4 pixels on each side, random 32×32 crops are extracted and images are flipped 50% of the time. The input data was normalised with per-channel means and standard deviations (of the train set).

Zhang et al. used grid searches to find optimal hyper-parameters. The experiments in this section use what they found to work best. These are listed and referred to as follows: SGD with momentum=0.9, learning rate=0.001, weight decay=0.001. Polyak with learning rate=0.3, weight decay=0.001. LA with α=0.8, k=5 wrapped around SGD with learning rate=0.1. AdamW with learning rate=3e-4, weight decay=1. In addition, the learning rate schedule was set to decay by a factor of 0.2 at the 60th, 120th, and 160th epochs. Two additional optimisers were included in these experiments, namely Adam and LA(Adam) which were setup with default hyper-parameters to act as baselines.

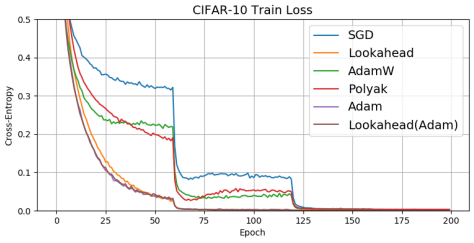

Figure 1: CIFAR-10 Training losses

Optimizer	Accuracy		Loss	
	CIFAR-10	CIFAR-100	CIFAR-10	CIFAR-100
SGD	**95.35**	**78.16**	**0.173**	**0.882**
Polyak	94.98	77.86	0.192	0.904
AdamW	94.90	77.43	0.260	1.096
LA	**93.19**	**73.08**	**0.389**	**1.513**
Adam	**93.83**	71.69	0.554	2.311
LA(Adam)	93.12	**71.96**	**0.500**	**2.216**

Table 1: CIFAR final validation metrics

All experiments were run for 200 epochs with a batch size of 128. Figure 1 shows training loss (cross-entropy) curves for each optimiser on CIFAR-10. We can see that LA converged faster than SGD, AdamW and Polyak, however the baselines Adam and LA(Adam) performed better (in this regard) than the 'optimal' hyper-parameters as found by Zhang et al. (2019). The plot also shows little difference between Adam and LA(Adam), hinting that LH does not significantly improve the performance of Adam on CIFAR-10 with default hyper-parameters (as claimed in the abstract of

their original paper). Table 1 shows final validation losses and accuracies for each optimiser. We can see that SGD consistently outperformed all other optimisers and that LA(Adam) marginally outperformed Adam. In addition to these results, the average time per epoch for Adam and LA(Adam) was $\approx 52s$ and $\approx 57s$ respectively (using a single Nvidia RTX 2070).

Overall these results differ significantly to those obtained by Zhang et al. (2019) (Figure 5 of their paper). It is possible that they did not mention some details of their implementation (suspected due to the differences with 'optimal' hyper-parameters). While LA **marginally** outperformed Adam with default parameters, this came with a slight additional time complexity. These experiments have illustrated that LA **does not** consistently/significantly outperform other optimisers.

3.2 Language modelling on the Penn Treebank dataset

Zhang et al. aimed to show the efficiency of LA in a word-level language modelling task (model prediction of next word given previous words) on the Penn Treebank (PTB) dataset - a highly pre-processed dataset of 10,000 unique words without capital letters, numbers, or punctuation. They followed the available code of Merity et al. The latter proposed a wide range of regularising and optimising strategies wrapped in a long short-term memory (LSTM) network. We skip the optimising strategies since they are substituted by LA or the plain optimiser. We start by giving all the default settings proposed by Merity et al., used by both Zhang et al. and us [1]: 3 LSTM layers of 1150 hidden units each, embedding layer size equal to 400, gradient clipping of maximum norm 0.25, batch size equal to 80, weight decay of 1.2e-6, L2 regularisation on RNN activations equal to 2 (alpha) and temporal activation regularisation equal to 1 (beta). Furthermore, dropout probability on the word vectors (input embedding layer) equal to 0.65 (*dropouti*), dropout probability to remove words from the embedding layer equal to 0.1 (*dropoute*), dropout probability on RNN layers (within each hidden layer) equal to 0.3 (*dropouth*) and dropout probability for the weight-drop method (applied to the RNN hidden to hidden matrix) used by Merity et al. equal to 0.5 (*wdrop*).

However, there were other hyper-parameters/strategies not mentioned by Zhang et al., so we proceeded with the default parameters on those: Dropout probability on the output of the final LSTM layer equal to 0.4 (*dropout*), variable sequence length according to the normal distribution $\mathcal{N}(70, 5)$ with probability 0.95 and $\mathcal{N}$(35,5) with probability 0.05, use of Weight tying (sharing the weights between the embedding and softmax layer) and not use of pointer augmentation - continuous cache pointer (*i.e. the results were extracted directly from main.py*)

From this point, we followed the different than default options set by Zhang et al., i.e. not including the fine-tuning stage proposed by Merity et al., batch sizes of 80, reducing the learning rate by half if validation loss has not decreased for 15 epochs (even though we believe using less than 15 epochs would converge faster to the same values, we left the same number to be consistent with Zhang et al.), not using momentum on SGD and using the default momentum values (β_1=0.9, β_2=0.999) on Adam. However, Zhang et al. mention only that for SGD they searched learning rates in the range $\{$50, 30, 10, 5, 2.5., 1, 0.1$\}$, for Adam in $\{$0.1, 0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001$\}$ and picked the best models based on the best validation performance, but without indicating which were these models. Similarly, after picking these best baseline models, they tuned the LA optimiser, but again they did not provide specific information about the choice of the number of inner-loop updates k and interpolation coefficients α, writing: *'For this task,* $\alpha = 0.5$ *or* $\alpha = 0.8$ *and* $k = 5$ *or* $k = 10$ *worked best'*. They reported (in Figure 7(a) from Zhang et al. (2019)) the learning curves where we observed in terms of convergence and performance a big advantage of LA(Adam) over Adam and no advantage of LA (SGD) over SGD which prompted our curiosity.

With much uncertainty over the settings used, we aimed to replicate all the experiments following the same procedure as Zhang et al., using fewer epochs (mostly 350 instead of 750 used by Zhang et al.), but enough to understand the behavior of each optimiser and keep time in reasonable bounds for our computational resources (epochs averaged 40s on an RTX 2070 GPU). Initially, we found that for the baseline optimisers, the optimal learning rate for SGD was 30 and for Adam 0.005. For Adam, 0.01 gave worse but acceptable performance, in contrast to other learning rates which would take too long to converge (or not converge at all). For the best choices, we tested the 4 combinations of α and k mentioned above and found that $\alpha = 0.8$ and $k = 10$ yielded the best results but with

[1] Original code available at `https://github.com/salesforce/awd-lstm-lm`. Italics denote the name of hyper-parameters in code. Explaining the regularising techniques is out of the scope of this paper.

subtle differences. The latter observation can confirm in some extent the statement of Zhang et al. regarding the robustness of LA to changes in the number of α and k. Moreover, in the beginning of training when we usually want to explore more, high values of α and k would be more suitable, although later smaller values seemed sometimes more beneficial. Based on this observation, future work could be focused on building an appropriate learning scheduler for LA.

Figure 2 illustrates the training perplexity curves [2] of the LSTM models we trained using the best hyper-parameters. We saw a lower average perplexity than that of Zhang et al., which most likely derives from some of their unmentioned settings, such as the continuous cache pointer or the weight tying strategy. Nevertheless, that does not confute the following analysis and we did not focus on that, since our purpose was to compare LA with the inner-loop optimiser, rather aim for the best possible performance. On the one hand, a contradiction to the claims of Zhang et al. is that for Adam, LA does not perform better in terms of convergence and perplexity than the inner optimiser if we use the optimal learning rate for the latter, i.e. 0.005. On the other hand, if we use the Adam default learning rate of 0.01, then indeed LA(Adam) outperforms Adam. This confirms also the authors' claim that LA is robust to less optimal inner-loop hyper-parameters. A 'malicious' hypothesis would be that Zhang et al. reported the default (but sub-optimal in this case) Adam learning rate of 0.01, which is misleading and not in accordance to their methodology of picking the best models. But since we likely had some different setting from Zhang et al., we trust that their optimal Adam learning rate happened to yield a case where LA(Adam) clearly outperformed Adam. Finally, regarding LA(SGD) and SGD our curves more or less agree with those of Zhang et al., i.e. depict slower convergence of both optimisers. For completeness, Table 2 shows the perplexity values derived at the best validation performance epoch.

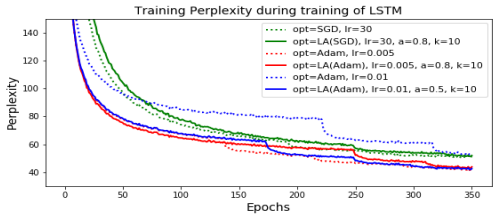

Figure 2: Training perplexities of LSTM models

Optimiser	lr	α	k	Train	Val	Test
SGD	30	-	-	50.61	66.85	63.96
LA (SGD)	30	0.8	10	51.45	67.87	64.59
ADAM	0.005	-	-	41.72	63.88	61.14
LA(Adam)	0.005	0.8	10	43.26	63.87	60.63
ADAM	0.01	-	-	52.95	66.26	63.44
LA(Adam)	0.01	0.5	10	41.92	63.75	60.92

Table 2: LSTM perplexities on best validation epoch

4 Conclusion

We have attempted to reimplement results put forth by Zhang et al. The implementation of LA matches its presentation in the original paper and its additional computational cost was subtle in both tasks. In CIFAR experiments, we illustrated that LA does not always improve performance. With 'optimised' hyper-parameters, SGD outperformed LA and with default parameters, LA marginally outperformed Adam. However, from the language modelling task we concluded that LA is indeed robust to the changes of its own hyper-parameters, and at least as robust as its inner optimiser to the changes of the latter. Even though other factors like the correct learning scheduling seemed to be more critical for the final outcome, LA rarely showed much worst results than its plain optimiser and even outperfomed Adam on its default settings. Thus, overall we consider LA as a competitive cautious ('1 step back') optimiser, useful when there is not much time for tuning.

References

Terrance Devries and Graham W. Taylor. Improved regularization of convolutional neural networks with cutout. *CoRR*, abs/1708.04552, 2017. URL `http://arxiv.org/abs/1708.04552`.

Alex Krizhevsky. Learning multiple layers of features from tiny images. *University of Toronto*, 05 2012.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *CoRR*, abs/1605.07146, 2016. URL `http://arxiv.org/abs/1605.07146`.

Michael Zhang, James Lucas, Jimmy Ba, and Geoffrey E Hinton. Lookahead optimizer: k steps forward, 1 step back. In *Advances in Neural Information Processing Systems*, pp. 9593–9604, 2019.

[2] On original code it was $Perplexity = e^{loss}$, making 'small changes in loss seem significant in perplexity'