Reproducibility Report
SAdam: A Variant of Adam for Strongly Convex Functions

Anonymous Author(s)
Affiliation
Address
email

Reproducibility Summary

Scope of Reproducibility

The central claim of the original paper is that a modification of Adam to utilize strong convexity yields faster convergence owing to a data dependent logarithmic regret bound, much akin to the recently proposed SC-RMSProp. Additionally the authors also claim that the higher performance is sustained even in non-convex optimization problems. We run the same experiments as the ones listed in the original paper, trying to reproduce the observations using a different framework (PyTorch) as compared to the authors (TensorFlow) [1].

Methodology

We did not use any of the source code provided, preferring to implement everything from the scratch, however we did run the source code to verify the results, using the code provided itself. Although we did vary the hyperparameters to see the difference on the results obtained, the values mentioned in the original paper, proved to work the best. We used the NVIDIA Tesla K80 GPU provided on the Google Colab platform to perform all our experiments.

Results

All the claims of the paper, are in line with our empirical observations as well, and our results are within a range of 1-2 % of accuracy. For one of the experiment listed in the appendix of the original paper involving training of ResNet-18 with all of the optimizers, our results deviate. However, the same does not discredit any of the claims made by the original paper, as that additional experiment merely was shown to prove an additional benefit of using SAdam in the original paper.

What was easy

All experiments were easy to setup, from procuring all the 3 datasets to the selection of the hyperparameters. In addition, none of the experiments had a high runtime requirement, hence no additional hardware was required to conduct the experiments and training on Google Colab was sufficient.

What was difficult

Implementing the optimizers against which SAdam was to be compared took up a lot of time and compounded the work at hand. Besides that a few contradictions in the proposed values of the mini-batch size by the authors of SC-RMSProp and SAdam caused a delay in the completion of the reproducibility study.

Communication with original authors

The authors shared their codebase with us in our very first correspondence. We reached out to them on multiple occassions for clarifications, and they were always prompt in answering our doubts.

Submitted to ML Reproducibility Challenge 2020. Do not distribute.

1 Introduction

Any variation with respect to Adam with a solid mathematical backing deserves a serious look and hence SAdam [2] got our attention. SAdam is an online convex optimizer that enhances the Adam algorithm by utilizing strong convexity of functions wherever possible. Although the motivation behind making these modifications are to improve performance in only convex cases, they prove to be effective even in non-convex cases. The Online Convex Optimization (OCO) framework involves iterative improvement of the model over a sequence of rounds. The OCO framework models the decision set as a convex set in Euclidean space denoted $K \subset R^n$. In each round t the player (our model) chooses a decision x_t from K. At the same time, the adversary reveals a cost function $f_t(.)$ and the player incurs a cost of $f_t(x_t)$. The difference between the cost incurred by the player and the cost incurred for taking the best decision is called Regret, and the minimization of this value entails improvement of the model.

$$R(T) := \sum_{t=1}^{T} f_t(x_t) - \min_{x \in K} \sum_{t=1}^{T} f_t(x) \tag{1}$$

Intuitively, an algorithm performs well if its regret is sublinear as a function of T where T is the time horizon i.e. $R_{Algo}(T) = o(T)$, and as the complexity of the regret decision decreases, the performance of the optimization increases.

Adam dynamically adjusts the step size and the update direction by exponential average of the past gradients. Although Adam has been extremely successful in many applications, it suffers from the non-convergence issue. AMSgrad and AdamNC were introduced to address these issues. These variants boast a data-dependant regret bounds that are $O(\sqrt{T})$ in the worst case.

SAdam is the first variant of Adam adapted to strongly convex functions. It follows the general framework of Adam, yet keeping a faster decaying rate step size controlled by time-variant hyperparameters to exploit strong convexity. Although it must be mentioned that similar ideas have successfully been applied in the past to the frameworks of Adagrad and RMSProp to get the variants called SC-Adagrad and SC-RMSProp [3]. Theoretical analysis of SAdam show a data-dependent $O(logT)$ regret bound for strongly convex functions, which means that it converges faster than AMSgrad and AdamNC in such cases.

2 Scope of reproducibility

We extrapolate the following claims from the original paper -

- Modification of Adam (henceforth referred to as SAdam) to utilize strong convexity yields faster convergence owing to a data dependent logarithmic regret bound, much akin to the recently proposed SC-RMSProp
- SAdam ends up outperforming all the existing optimization techniques in the context of deeper network training as well, which is inherently a non-convex optimization problem.
- A specific configuration of hyperparameters in SAdam generates a data-dependant logarithmic regret bound for SC-RMSprop algorithm.

We seek to verify the first 2 claims made by the authors empirically as done by the authors themselves, as the other claim is theoretical, and the base paper provides sufficient proofs to uphold the veritibility of the same.

3 Methodology

The authors made their source code available to us, upon request (now available on the OpenReview Portal as well). Their experimental setup made use of the Tensorflow framework, whilst utilizing the publicly available implementations [4] of the other optimizers against which comparisons were made. However we implemented the codebase from scratch using the PyTorch 1.7 framework, which meant reimplementing the other optimizers in question along with SAdam, due to a lack of a publicly available implementation of the same in PyTorch. All experiments were carried out on Google Colab with the NVIDIA Tesla T4. The following experiments were conducted -

- Calculation of Regret for L2 Regularized Logistic Regression on MNIST, CIFAR10, CIFAR100 for our pool of optimizers, to check the validity of claim 1.
- Computing test accuracy and training loss for a 4-layer CNN and ResNet18 on MNIST, CIFAR10, CIFAR100. for our pool of optimizers to check the validity of claim 2.

3.1 Model Description & Experimental Setup

All of the optimizer algorithms try to learn the parameter vectors $\{w_i, b_i\}_{i=1}^{K}$, by minimizing the loss accumulated over each mini-batch of training samples $\{(x_i, y_i)\}_{i=1}^{m}$, where $y_i \in [K], \forall i \in [m]$, in each round t. This loss function also called the Cross Entropy Loss [5] has been used across all our experiments.

$$J(w,b) = -\frac{1}{m}\sum_{i=1}^{m} log\left(\frac{e^{w_{y_i}^T + b_{y_i}}}{\sum_{j=1}^{K} e^{w_j^T + b_j}}\right) + \lambda_1 \sum_{k=1}^{K} \|w_k\|^2 + \lambda_2 \sum_{k=1}^{K} b_k^2 \tag{2}$$

3.1.1 L2 Regularised Softmax Regression

A sequential layer is added and the outputs of the same are directly fed into the loss function. It must be noted that the PyTorch implementation of Categorical Cross Entropy Loss inherently applies Softmax, while computing the loss, hence to leverage their implementation we remove the last softmax function from our model, and directly feed in the fully connected layers output to the Loss function. We report the regret for this machine learning model across all our datasets and optimizers, considering the expert model of choice, to be the one trained by the Adam optimizer. This choice is completely trivial, and a different choice of an expert model would lead to a quantitative change of results, but would have no effect on the inferences obtained from the experiments.

3.1.2 4 Layer CNN

To verify the second claim, pertaining to the efficacy of SAdam in non-convex cases, we test the performance of the optimizers on a custom 4 layer CNN, consisting of two convolutional layers (each with 32 filters of size $3X3$), one max-pooling layer (with a $2X2$ window and 0.25 dropout), and one fully connected layer (with 128 hidden units and 0.5 dropout). We employ $ReLU$ function as the activation function for convolutional layers and softmax function as the activation function for the fully connected ones. We train this network on all of our datasets, using all of the optimizers.

3.1.3 ResNet 18

To further substantiate the claim of the authors, we choose a deeper usage of architecture to see, whether the higher levels of performance are sustained as the number of layers increase. The popular ResNet-18 architecture was used for this purpose, the details of which can be found in the base paper [6]. To verify the claims made by the authors, we conduct the same experiment as them, running our models 10 times and finally taking a mean of the aggregate results on the CIFAR 10 dataset.

3.2 Datasets

The following datasets were used across experiments -

- MNIST : Dataset containing 60000 28X28 training images and 10000 test images depicting handwritten digits spread evenly across each digit (0-9). [7]
- CIFAR10 : A subset of the 80 million tiny images dataset and consists of 60,000 32x32 color images containing one of 10 object classes, with 6000 images per class. It was collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton.[8]
- CIFAR100 : Subset of the same dataset as CIFAR10, containing 600 images (500 for training and 100 for testing) per class for 100 classes. The 100 classes in the CIFAR-100 are grouped into 20 superclasses.[8]

3.3 Hyperparameters

An exhaustive list of hyperparameter settings used across optimizers and experiments have been surmised in table 1. In the table α refers to the learning rate to be used, t refers to the step number, and thus in all convex cases, the learning rate is replaced by a degrading learning rate to account for the convergence. The β_1 and β_2 parameters appears in Adam and all of its variants controls the gradient moment decay rates as mentioned in [9]. The λ_1 and λ_2 parameters control the regularization in the cross entropy loss function, and is used only in convex cases. δ refers to a scalar value, that helps in avoiding divide by zero errors in optimizer updates. However, in the SC variants of RMSProp and Adagrad a time variant $\delta_{t,i}$ is used with the hyperparameters ξ_1 and ξ_2 controlling the variance.

Optimizer	Type	α	$\delta_{t,i}$	β_1	β_2	λ_1	λ_2	ξ_1	ξ_2
OGD	Convex	0.01	10^{-8}	$-$	$-$	0.01	0.01	-	-
	Non Convex	$0.01/t$	10^{-8}	-	-	0	0	-	-
Adam	Convex	0.001	10^{-8}	0.9	0.999	0.01	0.01	-	-
	Non Convex	$0.001/t$	10^{-8}	0.9	0.999	0	0	-	-
SAdam	Convex	0.01	10^{-2}	0.9	$1-0.9/t$	0.01	0.01	-	-
	Non Convex	$0.01/t$	10^{-2}	0.9	$1-0.9/t$	0	0	-	-
AmsGrad	Convex	0.001	10^{-8}	0.9	0.999	0.01	0.01	-	-
	Non Convex	0.001	10^{-8}	0.9	0.999	0	0	-	-
SC RMSProp	Convex	0.01	$\xi_1 e^{\xi_2 t V_{t,i}}$	$1-0.9/t$	-	0.01	0.01	0.1	1
	Non Convex	$0.01/t$	$\xi_1 e^{\xi_2 t V_{t,i}}$	$1-0.9/t$	-	0	0	0.1	0.1
SC Adagrad	Convex	0.01	$\xi_1 e^{\xi_2 t V_{t,i}}$	-	-	0.01	0.01	0.1	1
	Non Convex	$0.01/t$	$\xi_1 e^{\xi_2 t V_{t,i}}$	-	-	0	0	0.1	0.1

Table 1: Value of hyperparameters used across experiments and optimizers

3.4 Computational requirements

All the experiments listed in the paper were carried out on the Google Colab platform, with the hardware acceleration provided by NVIDIA Tesla K80 GPU. As none of the experiments took more than 3 hours to completion, no cost had to be incurred on our part for conducting any of the experiments. The list of average runtimes across experiments NVIDIA Tesla K80 GPU can be seen in the table below -

Model	**Dataset**	**Batch Size**	**Epochs**	**Average Runtime** (mins)
Softmax Regression	All	60	-	15
4 layer CNN	MNIST	64	50	13
4 layer CNN	CIFAR10	64	100	27
4 layer CNN	CIFAR100	512	100	52
ResNet 18	CIFAR10	64	100	128

Table 2: Comparison of run times across experiments

The calculation of regret in the case of softmax regression warrants the availability of an expert model. Thus, the majority of the runtime is spent in obtaining this model, which once ready enables quicker completion of the regret calculations, since in that case, all of the other optimizers will only have to be trained over 1 epoch. After the trained model is avaiable, calculation of regret on a CPU takes an average of 3 - 4 minutes per experiment, and if the same is done over a GPU, the experiment would take around 40 - 50 seconds on an average.

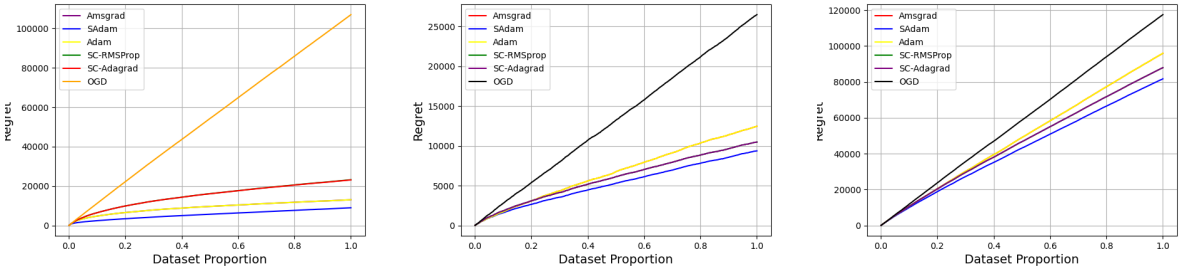
4 Results

All the claims of the paper, are in line with our empirical observations as well. The original paper also compared the performance of SAdam to that of AdamNC, however we skip that particular comparison, as AMSGrad already serves as an extension for the Adam algorithm to improve upon the convergence of its optimization process, and the inclusion of AdamNC does not enrich the comparisons with any new information, with its performance almost being a replica of AMSGrad and Adam. Aside from that fact, we were able to reproduce all of the experiments listed in the paper within a range of 1-2 % of accuracy in PyTorch.

4.1 Regret Calculation for L2-regularized Logistic Regression

This particular experiment seeks to establish the first claim made by the authors, pertaining to the utilisation of the convexity of the problem statement, to yield faster convergence. Across all the datasets in question, SAdam does converge fastest, and hence has the lowest overall regret. SC-RMSProp and SC-Adagrad have almost identical performances with very minor quantitative differences only in the decimal places. It must be noted that the final values of regret here are higher than the values reported in the actual paper due to the values in the paper being normalized by the batch size. Adam and AMSgrad also have identical performances, however their overall regret is much higher

compared to the algorithms that take advantage of the convexity. OGD due to its simplistic algorithmic setup has the highest overall regret as expected in all of the datasets.

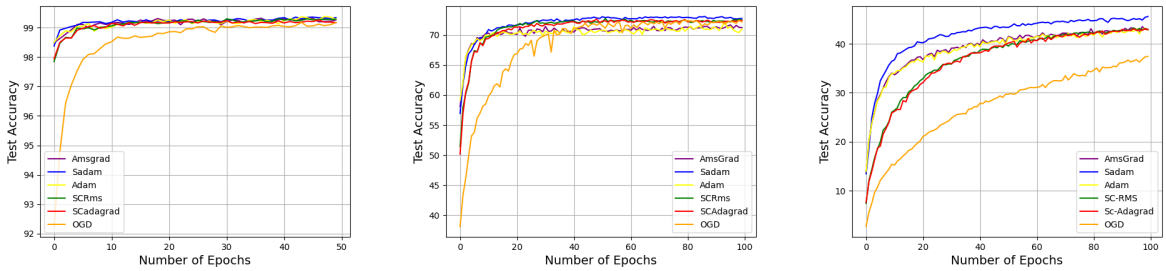


(a) MNIST (b) CIFAR 10 (c) CIFAR 100

Figure 1: Regret vs Data Proportion for $l2$-Regularized Softmax Regression

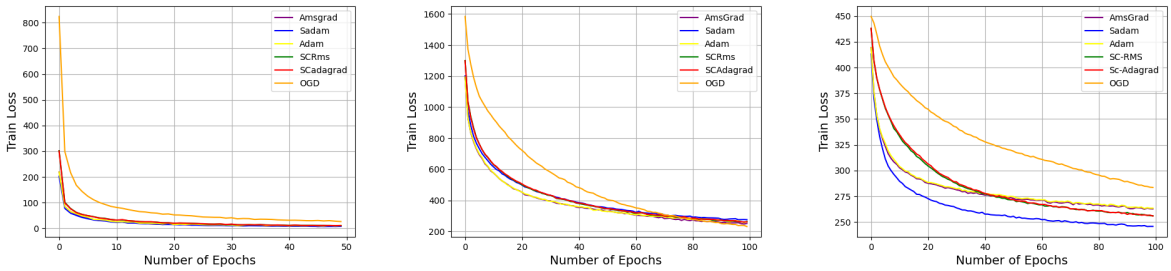
4.2 Performance on 4 Layer CNN

The 4 Layer CNN is trained using different optimizers to substantiate the second claim according to which, the performance of the algorithms is sustained even in case of non convex problems. The test accuracy and the training losses are reported against all of the datasets. The performance of SAdam is again overall the highest if only test accuracy is taken into account. The performance on CIFAR 10 and MNIST is only marginally better than the next best option which is SC-RMSProp. However the test accuracy achieved after training for a 100 epochs on the CIFAR 100 dataset exceeds the values reported in the actual paper by 2% and is significantly higher than all of the other alternatives. In case only training losses are compared, SAdam achieves the lowest values in case of MNIST and CIFAR 100, but for CIFAR 10, OGD has the lowest loss value after 100 epochs. Amongst the other optimizers, SC-RMSProp consistently scores better metrics, with both Adam and AMSGrad again having comparable values following closely behind.



(a) MNIST (b) CIFAR 10 (c) CIFAR 100

Figure 2: Test Accuracy vs Number of Epochs for 4 layer CNN

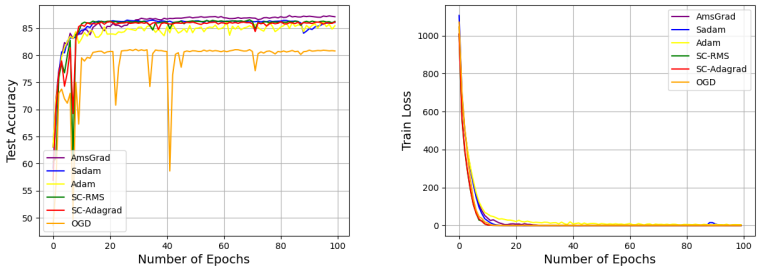


(a) MNIST (b) CIFAR 10 (c) CIFAR 100

Figure 3: Training Loss vs Number of Epochs for 4 layer CNN

4.3 Performance on ResNet-18

For this experiment, like the authors, we run the experiment 10 times, and report the mean aggregate values across all runs. In our results, AmsGrad gives the best results on the test set after 100 epochs, although SAdam does appear to give the best performance after around 30 epochs, before stagnating at the same accuracy levels, since it converges to a 0 at that point, and keeps taking random jumps from that point onwards. OGD suffers from this phenomenon the most, and thus has uneven jumps sporadically throughout its learning phase. However, it must be noted that all the optimizers that take advantage of the convexity, retain their faster convergence property while training on ResNet as well.



(a) CIFAR 10 Test Accuracy (b) CIFAR 10 Training Loss

Figure 4: Peformance on Optimizers on CIFAR-10 trained on the ResNet-18 architecture

5 Discussion

SAdam as shown in all of the experiments above, has the property of converging faster specially so in convex functions, but also to a certain extent in non-convex environments. Even though, the power of the algorithm diminishes with an increase in the complexity of the non-convex functions, SAdam still performs better that Adam and AmsGrad only falling short behind SC RMSProp in those cases. Although Adam has been the primary choice in optimizers for a long time for most deep learning applications, the faster convergence rates of SAdam makes it a viable choice for any convex setting, and at the very least enables quicker baseline evaluation checks of deeper networks, leading to reduced usage of hardware acceleration, while choosing amongst many model architectures.

5.1 Strength

Whilst writing our code, we chose to build everything from scratch in a different framework, taking the algorithms mentioned in the base papers without letting any bias creep in from the source code. We also in addition, conducted the experiments with various seeds, and the overall results remained the same. We would like to point out that, our code also serves as an open source contribution for all of the optimizers against which comparisons of SAdam were drawn, none of which except for Adam and AMSGrad had a publicly available PyTorch implementation, at the time of writing this report.

5.2 Weakness

It must be noted that although SAdam performs the best in deeper networks in most of the aforementioned experiments, there is no theoretical foundation for this empirical finding. Any attempt to do so warrants a sophisticated level of understanding of the calculus involved in the optimization mathematics and consequently our ability to address this concern was limited.

5.3 What was easy

The source code provided by the authors worked in one go, without modifications of any kind required. Even, when the base framework was changed to PyTorch, all the empirical experiments made in the original paper, were relatively easy to setup, right from procuring all the 3 datasets to the selection of the hyperparameters, owing to clear mention of all such variables in the paper. The algorithms itself, for each of the optimizers were clearly explained, and the translation of the algorithm to code was seamless. In addition, the low runtime for each individual run meant that, no additional hardware was required, and training on Google Colab was sufficient for each experiment given in the paper.

5.4 What was difficult

As mentioned before, implementing the additional optimizers in PyTorch compunded the work at hand, since to prove the efficacy of SAdam itself, a comparison with the state of the art optimizers with the same environmental variables at play was necessary. Besides this, the regret calculation experiment for L2-regularized Logistic Regression took much more time than expected owing to the contradictions in the proposed values of mini-batch size for each of the datasets, by the authors of SC-RMSProp and SAdam. Although SAdam works better for a far wider range of possible mini-batch sizes, the performance of SC-RMSProp increases, and eventually even surpasses that of SAdam if the mini-batch size is decreased substantially. The authors of SC-RMSProp themselves suggest a mini-batch size of 1, which therefore delivers the best of results for their alogrithms itself. For most real world applications, a per example based approach to learning is more likely to cause unstable optimization, and a sufficiently large mini-batch size leads to a smoother convergence, therefore holistically SAdam should be a preferred choice in a convex setting.

5.5 Communication with original authors

At the time of the launch of the Reproducibility challenge source code for the paper had not been shared on the open review portal. However with our first email correspondence, they shared their codebase promptly. The discrepancies in the recommended value of the mini-batch sizes proposed for the different algorithms were discussed with the authors, and they clarified their reasoning behind choosing of a different batch size. Besides this, the authors answered our minor queries with regards to the source code as well, and were supportive throughout the entire reproducibility process.

References

[1] Yash Raj Sarrof and Narayanan Elavathur Ranganatha. SAdam Reproducibility Code Repository. `https://github.com/naruarjun/SADAM-reproducibility`.

[2] Guanghui Wang, Shiyin Lu, Weiwei Tu, and Lijun Zhang. SAdam: A variant of Adam for strongly convex functions. *arXiv preprint arXiv:1905.02957*, 2019.

[3] Mahesh Chandra Mukkamala and Matthias Hein. Variants of RMSProp and Adagrad with logarithmic regret bounds. In *International Conference on Machine Learning*, pages 2545–2553. PMLR, 2017.

[4] Guanghui ang, Shiyin Lu, Weiwei Tu, and Lijun Zhang. SAdam Original Code Repository. `https://github.com/SAdam-ICLR2020/codes`.

[5] Zhilu Zhang and Mert R Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. *arXiv preprint arXiv:1805.07836*, 2018.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[7] Christopher J.C. Burges Yann LeCun, Corinna Cortes. Mnist dataset download link. `http://yann.lecun.com/exdb/mnist/`.

[8] Vinod Nair Alex Krizhevsky and Geoffrey Hinton. Cifar 10 and cifar 100 dataset download link. `https://www.cs.toronto.edu/~kriz/cifar.html`.

[9] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.