Benchmarking Optimizers for Sign Language detection

Loran Knol (s3182541) Jeroen Overschie (s2995697)

University of Groningen, Nijenborgh 9 9747AG Groningen

Abstract

A benchmark on both established and relatively new Neural Network optimizers was conducted, on a sign language dataset. Although performance levels for all optimizers were similar, Adamax and Adagrad scored well on both computational time and generalization performance. Accompanied with the benchmark results comes a small demo[1] - demonstrating the prediction capabilities of the sign language detection model, and also the program source code[2].

1 Introduction

Deep Neural Networks are in increasing popularity. Ever more applications are found to benefit from an increased amount of layers in the network architecture - making the networks bigger than before. But adding more neurons means a larger computational cost, causing the need for more efficient techniques. Among innovations such as back-propagation [10], proper network initialization [5] and GPU utilization [8] are also: better **optimizers**.

Optimizers aim to minimize the loss function - a metric used to assess the performance of a Neural Network. The optimization process is performed by step-wise descending the loss function landscape; by computing the gradient w.r.t. the network parameters. Due to the computational complexity involved, the optimizer choice is an important one.

In this project, we aim to benchmark a line-up of optimizers, based on their effectiveness in terms of achieved accuracy, generalization and speed. To benchmark such optimizers, we posed ourselves a classification problem, which is to detect the alphabetic sign language letters [1] using a Neural Network.

The report will proceed as follows. First, we will discuss the set-up of the study in Section 2: the dataset and prediction goal, the network architecture and the line-up of optimizers. Next, the results will be presented in Section 3. Lastly, the whole will be concluded in Section 4.

Optimizer	β_1	β_2	Decay	ϵ	Learning rate	ρ
AdaBelief	0.9	0.999	0	10^{-7}	10^{-3}	-
AdaDelta	0.9	0.999	0	10^{-7}	10^{-3}	0.95
AdaGrad	0.9	0.999	0	10^{-7}	10^{-3}	-
Adam	0.9	0.999	0	10^{-7}	10^{-3}	-
AdaMax	0.9	0.999	0	10^{-7}	10^{-3}	-
Nadam	0.9	0.999	0	10^{-7}	10^{-3}	-
RAdam	0.9	0.999	0	10^{-7}	10^{-3}	-
RMSProp	0.9	0.999	0	10^{-7}	10^{-3}	0.9
Yogi	0.9	0.999	10^{-8}	10^{-7}	10^{-3}	-

Table 1: Optimizers under consideration

2 Methods

This study considers ten optimizers in total, which have been listed in Table 1. These optimizers include widely known

[1] https://dunnkers.com/optimizers-sign-language

[2] https://github.com/dunnkers/optimizers-sign-language

ones that have been used by the field for some time now, such as Root Mean Square Propagation (RMSProp) [3], but also some that have only been invented much more recently, like AdaBelief [12], RAdam [6] and Yogi [9].

The used dataset used is found on Kaggle [4] and has 87,000 samples depicting 29 classes: 26 for the letters A through Z and another three for 'delete', 'nothing' and 'space'. The data has been shuffled and split into a training and validation set, where the training set is constituted by 90% of the data and the validation set is formed by the remaining 10%. Every model was trained for 10 epochs with a batch size of 32 and the categorical cross-entropy loss function. The experiment was run on a Tesla P100 GPU with the Keras API [2] on a TensorFlow backend. All models were compared on their top-1 training and validation accuracy.

Type / Stride	Filter shape	Input shape
Conv / 1	4x4x3x64	64x64x3
Conv / 2	4x4x64x64	61x61x64
Dropout / -	-	29x29x64
Conv / 1	4x4x64x128	29x29x64
Conv / 2	4x4x128x128	26x26x128
Dropout / -	-	12x12x128
Conv / 1	4x4x128x256	12x12x128
Conv / 2	4x4x256x256	9x9x256
Flatten / -	-	3x3x256
Dropout / -	-	2304
FC / -	2304x512	2304
FC / -	512x29	512

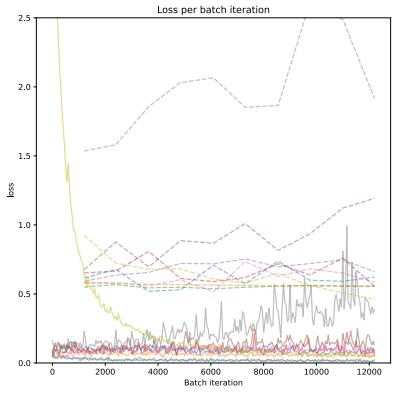
Table 2: Network architecture.

A small network architecture was chosen [7] in the hopes that this would limit the capacity of the model enough to allow it to generalize from the training to the validation set. The network architecture is given in Table 2. It consists of a number of convolutional layers that project every image to progressively smaller dimensions while increasing the number of kernels. Every two convolutional layers are followed by a dropout layer (dropout rate 0.5). The last convolutional layer is also followed by a fully connected (FC) layer of 512 units, which connects to an FC layer of 29 output units. All layers use the ReLU activation function (when applicable), except the last layer, which uses the softmax activation function.

3 Results

The top-1 training and validation accuracy is depicted in Figure 1a for every optimizer per batch iteration. The figure shows that the training accuracy is already quite high (above 95%) for most optimizers in the starting batches. The Yogi optimizer starts off somewhat lower with its training accuracy, however, but quickly climbs to above 95% as well. The optimizers that end up with a near-100% training accuracy are AdaGrad and AdaMax.

(a) Top-1 training and validation accuracy for all optimizers.



(b) Training- and validation loss per optimizer.

Figure 1: Results of comparing optimizers in a sign language detection benchmark. Note that performance metrics are computed: every batch for training set and every epoch for the validation set.

The pattern in validation accuracies is similar to the one in the training accuracies. All optimizers start with a high ($> 80\%$) validation accuracy, except the Yogi optimizer, which climbs from $< 70\%$ to accuracies roughly similar to the other optimizers. The two optimizers with the highest validation accuracy are again AdaGrad and AdaMax.

Also for Figure 1b, this pattern largely returns: All optimizers have low loss values from the start of training, with the Yogi optimizer starting with high loss values but steadily decreasing as training progresses. A difference is that RMSProp starts of with small training loss, which actually increases during training, and has a very high validation loss over the entire training procedure.

The total runtime to test all optimizers was 5 hours and 27 minutes, whilst one individual training run on an optimizer took about 200 seconds per epoch, as can be seen in Figure 2. The resolution and scale of our experiment does not allow to tell significant differences in the runtimes of the optimizers. The only outlier that exists is Yogi: it spends an unusually long amount of time on the first epoch, but acquires similar results afterwards.

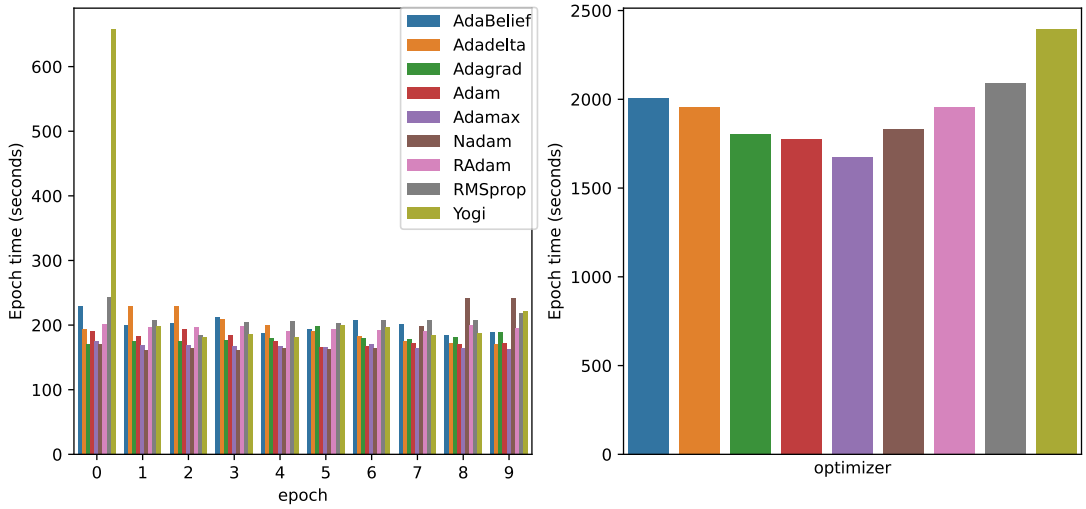


Figure 2: Execution time over epochs for all optimizers (left) and average execution time for all optimizers (right).

4 Discussion

The realm of optimizers has been a much studied space - which should not come as a surprise: choosing the right optimizer for the task at hand can save many computational resources and yield better results at the same time. The choice of hyperparameters remains a difficult one, and no general consensus exists on generically applicable good configurations. It has been quoted to sometimes be rather an 'art' than a science [3].

However, we are still able to distill general characteristics about optimizers, such as runtime, accuracy and generalization capability (i.e. inverse of validation error) based on the conducted experiment. Good performers in our benchmark in terms of both validation performance and runtime are AdaGrad and Adamax.

Many of the optimizers tested show remarkably high training and validation accuracies from the very first epoch. This might be a consequence of the fact that the dataset contains a relatively large number of images (87,000) for just 29 classes. In addition, all images seem to have been captured in the same room, with the same hand doing all the sign language gestures, which might make it easier for the network to learn the filters relevant for removing background information very early on.

An improvement for future work would be to achieve better generalization performance by using a more varied, larger dataset. An idea for future work would be to conduct a more comprehensive benchmark, e.g. by including larger datasets and testing multiple hyperparameter settings for each optimizer. There are also many more optimizers [11] around; the line-up could have also been more extensive.

References

[1] American Sign Language. https://www.nidcd.nih.gov/health/american-sign-language. Accessed: 2021-02-22.

[2] François Chollet et al. Keras. https://keras.io, 2015.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[4] Grassknoted dataset. https://www.kaggle.com/grassknoted/asl-alphabet. Accessed: 2021-02-22.

[5] G. E. Hinton and R. R. Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. *Science*, 313(5786):504–507, 2006.

[6] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *CoRR*, abs/1908.03265, 2019.

[7] Network architecture. https://www.kaggle.com/dansbecker/running-kaggle-kernels-with-a-gpu. Accessed: 2021-02-25.

[8] Kyoung-Su Oh and Keechul Jung. Gpu implementation of neural networks. *Pattern Recognition*, 37(6):1311–1314, 2004.

[9] S Reddi, Manzil Zaheer, Devendra Sachan, Satyen Kale, and Sanjiv Kumar. Adaptive methods for nonconvex optimization. In *Proceeding of 32nd Conference on Neural Information Processing Systems (NIPS 2018)*, 2018.

[10] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[11] Robin M Schmidt, Frank Schneider, and Philipp Hennig. Descending through a crowded valley–benchmarking deep learning optimizers. *arXiv preprint arXiv:2007.01547*, 2020.

[12] Juntang Zhuang, Tommy Tang, Yifan Ding, Sekhar C Tatikonda, Nicha Dvornek, Xenophon Papademetris, and James Duncan. Adabelief optimizer: Adapting stepsizes by the belief in observed gradients. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 18795–18806. Curran Associates, Inc., 2020.