A summary of novel optimization techniques

Raja Haseeb Ur Rehman
Department of Aerospace
KAIST
South Korea
rajahaseeb147@kaist.ac.kr

Abstract

Since the evolution of deep networks, optimization has always been a key process. Reducing training time while maintaining the same level of accuracies has always been a challenging task. In this summary paper, some recent novel optimization concepts have been presented.

1 Introduction

In recent years, deep learning has become immensely popular and has been used in a large number of applications including image processing, natural language processing, object detection, text mining, social media recommendations, and so forth. Compared to traditional models, deep learning architectures extract these features in an automated way, making it both efficient and easier for the user to implement modern machine learning tasks. With huge success in many fields, deep learning algorithms are now the state of the art standard for all kinds of machine learning tasks. However, such accurate results and performance also requires some careful training procedure.

Due to its importance in the training and performance of the deep learning architecture, optimization has been the area of interest for many researchers in recent years, and many techniques have been developed to make the optimization process as smooth and convergent as possible. Smooth optimization process depends on many factors such as data preprocessing and careful weight initialization, Batch Normalization [4], network architecture, selecting reasonable loss function, choice of the optimizer, learning rate and so on. This paper summarizes some of the recent advancements in the optimization of deep neural networks.

2 Deep Learning Optimization

Deep learning is a highly iterative process. We have to try out various permutations of the hyperparameters to figure out which network design works best. Therefore our deep learning model must train in a shorter time without penalizing the cost. Researchers from all over the world have been trying to reduce the computational cost in deep networks as much as possible. Some of the recent such attempts are discussed and summarized in the sections below.

2.1 Don't decay the learning rate, increase the batch size

Nowadays, during the optimization of deep neural networks, it is common practice to decay the learning rate during the training for better results. When gradient descent is near a minima point, the values of the parameters can oscillate back and forth around the minima. This effect can be reduced by decaying the learning rate manually when the validation accuracy plateau. Alternatively, the use of the learning rate schedule has been proposed [8] to automatically reduce the learning rate, based on the number of epochs. However, the authors of the [10] argue that one can usually get the

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

same learning curve for training and test process by using a larger batch size during training. This implementation reaches the same accuracy levels as of learning rate decay in the same number of epochs, but in fewer parameter updates which leads to a very short training period. According to the recent work by Smith & Le (2017) the noise scale in the SGD is given by the equation

$$g = \epsilon(\frac{N}{B} - 1) \tag{1}$$

where ϵ is the learning rate, N is the size of the training set and B is the batch size. This noise scale usually controls the magnitude of the oscillation of the training dynamics. When the learning rate is dropped, the gradient noise scale also falls, which then enables the algorithm to converge to the minima. However, according to the equation (1), increasing the batch size can also provide the same effect with even fewer parameter updates. Furthermore, we can combine both methods i.e. after reaching a certain batch size limit say *B∼N/10*, we can switch to the learning rate decay method.

2.1.1 Experiments

Experimental evidence has been provided by the authors to support their claim that decreasing the learning rate and increasing the batch size during training are equivalent. Increasing batch size reduces the number of parameter updates and hence training time. We can then further reduce the number of parameters by increasing the effective learning rate and scaling $B \propto \frac{\epsilon}{(1-m)}$, where m is the momentum coefficient [10].

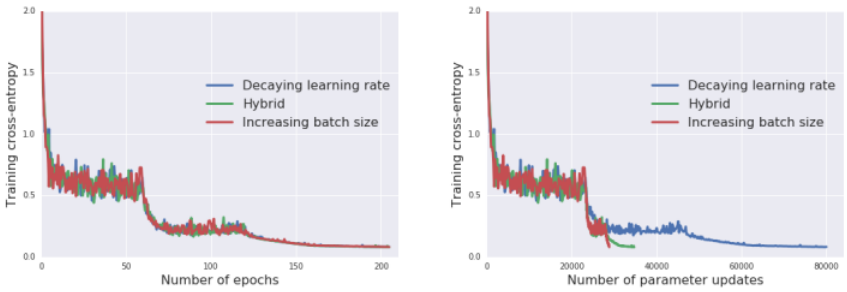

Figure 1: Wide ResNet on CIFAR-10. Training set cross-entropy, evaluated as a function of the number of training epochs and the number of parameter updates [10].

The three learning curves in figure 1 are almost identical, however, increasing the batch size significantly reduces the number of parameter updates required. As for the test accuracy during the training, all three schedules track each other very closely.

2.1.2 Remarks

As promising as it seems, there remains a question about the practicality of the proposed methodology as there is a limit to which batch size can be increased. Even with the best GPUs available today, normal people and researchers do not have access to such huge batch sizes as mentioned in the paper unless they are working in a very big organization. So the extent to which this proposed procedure is feasible compared to the learning weight decay is still a question that needs to be answered.

2.2 On the variance of the adaptive learning rate and beyond

Recently, many efforts have been made to accelerate optimization by applying the adaptive learning rate. In particular, Adagrad [2] and its variants, e.g., RMSprop (Hinton et al., 2012), Adam [5], Adadelta [14] and Nadam [1], stand out due to their fast convergence, and have been considered as the optimizer of choice in many applications.

However, it has been observed that such optimization algorithms might converge to a bad optima, and may need a warmup heuristic [13][7]. However, there is no proper justification nor guidance on how to conduct warmup. The root cause for such bad convergence properties is the large variance of the adaptive learning rate in the early stage of model training, particularly due to the small number of

training examples being used. hence using smaller learning rates in the first few steps can reduce variance, which justifies the warmup heuristic. Besides, a new variant of Adam *i.e. Rectified Adam (RAdam)*, is also proposed which rectifies the variance and compares favorably with the heuristic warmup [6].

RAdam deactivates the adaptive learning rate in extreme cases where the variance is divergent which avoids the undesired high variance behavior in the first few epochs, without any additional hyperparameter. Comparisons have been made between the heuristic warmup and proposed RAdam on different datasets including ImageNet and CIFAR-10. Both methods show similar performance, representing the effectiveness of proposed rectification. Compared to heuristic warmup, RAdam also shows more robustness to the change of learning rates.

2.2.1 Remarks

Compared to heuristic warmup, RAdam textit(Rectified Adam) does not require any additional parameter and is also more sound to the changes in the learning rate. However, the quantitative extent up to which it outperforms the heuristic approach is still quite unclear in the paper. Further justification can build a stronger case and hence can replace the heuristic approach completely. Another important point to notice is that RAdam fails to outperform SGD in terms of test accuracy, which a big question mark.

2.3 MobileNetV2: Inverted Residuals and Linear Bottlenecks

The main contribution of this paper [9] is the implementation of a new mobile architecture i.e. textit-MobilenNetV2), which is specifically developed for resource-constrained environments. Compared to the previous SOTA models, the proposed architecture significantly reduces the number of operations required and memory required, while maintaining the same accuracy levels. The key idea is to combine the depthwise separable convolutions and inverted residual bottlenecks. Bottlenecks layers are used in a deep network to reduce the number of feature maps in the network, which otherwise tend to increase in each layer [12].

As usual, each layer has batch normalization and the activation function is ReLU6. However, the output of the projection layer does not have an activation function applied to it. Since this layer produces low-dimensional data, the authors of the paper found that using a non-linearity after this layer destroyed useful information.

The motivation for inserting the residual connection is similar to the original attempt, to improve the gradient flow in the network as we go deeper. However, the inverted design appears to be considerably more memory efficient [9]. The new result reported in this paper is that the use of shortcut connections between the bottleneck layers instead of expansion layers yields better performance. This paper also introduces a new variant to SSD i.e. SSDLite, in which all the regular convolutions are replaced with separable convolutions.

Network	mAP	Params	MAdd	CPU
SSD300[34]	23.2	36.1M	35.2B	-
SSD512[34]	26.8	36.1M	99.5B	-
YOLOv2[35]	21.6	50.7M	17.5B	-
MNet V1 + SSDLite	22.2	5.1M	1.3B	270ms
MNet V2 + SSDLite	22.1	**4.3M**	**0.8B**	200ms

Figure 2: Performance comparison of real time detectors on COCO dataset object detection task [9].

The results indicate that the MobileNetV2 SSDLite is not only the most efficient model but also outperforms many other mobile architectures. The proposed architecture improves the state of the art performance on the ImageNet dataset. It also outperforms the SOTA real-time detectors on the COCO dataset, and that too with less model complexity. Notably, when combined with the SSDLite detection module, it has 20× less computation and 10× less parameters than YOLOv2.

2.3.1 Remarks

As usual for this kind of model, the number of channels is increased over time. But overall, the tensors remain relatively small, thanks to the bottleneck layers that make up the connections between the blocks. Compared to this, V1 lets its tensors become much larger (up to 7×7×1024). Using low-dimension tensors is the key to reducing the number of computations. After all, the smaller the tensor, the fewer multiplications the convolutional layers have to do.

However, applying a convolutional layer to filter a low-dimensional tensor won't be able to extract a whole lot of information. So to filter the data we ideally want to work with large tensors. MobileNet V2's block design gives us the best of both worlds. The trick that makes this all work, of course, is that the expansions and projections are done using convolutional layers with learnable parameters, and so the model can learn how to best (de)compress the data at each stage in the network.

2.4 Four thing everyone should know to improve Batch Normalization

Despite its extensive use in the optimization process for deep architectures, it has always been a challenging task to improve upon Batch Normalization [4]. In this paper [11], authors present four improvements upon the Batch Normalization while without any need for additional computation during training. The first one is the inference example weighing. The output range of a network with Batch Normalization is wider during training than inference, creating a discrepancy. The proposed inference example weighing resolves this disparity between training and inference while using Batch Normalization.

Ghost Batch Normalization [3] can also be used for medium-sized batches and is easy to tune. A combination of Ghost Batch Normalization with the inference example weighing can have a significant effect on the optimization process. According to the hypothesis, and the intermediate batch value would offer the best tradeoff of regularization strength. Surprisingly this simple technique was capable of improving the performance by 5.8% on Caltech-256 and 0.84% on CIFAR-100.

According to the evaluation, using weight decay on γ and β instead of weights can help improve performance. On CIFAR-10, it was found that incorporating it improved the accuracy by significant 0.3%. Another important discovery is that network architecture plays a crucial role here. In Inception-v3, this technique hurts the performance while in ResNet-50, it improved the performance by 0.91%.

While Batch Normalization seems to be very helpful and effective in medium to large batch sizes, it suffers when there are not enough examples available. However combining Group (Wu & He, 2018) and Batch Normalization i.e. normalization statistics are not only calculated over channels only but also over examples. This leads to more accurate models in the setting of smaller batch size B >1, and also provides regularization effect.

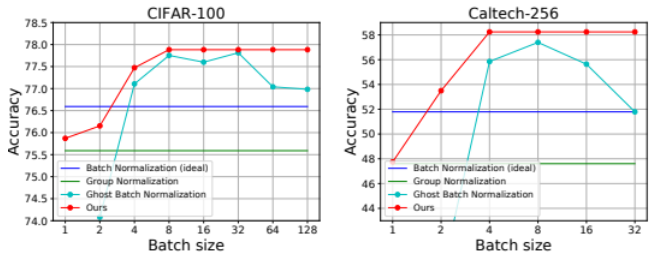

Figure 3: Total performance changes across batch sizes for CIFAR-100 and Caltech-256 [11].

2.4.1 Remarks

The authors have done their best to demonstrate the effectiveness of proposed methodologies, and also guided the cases in which it might be applicable. Performance improvements are quite significant, and it provides a new direction for research and expansion of the proposed methods to various other architectures and cases as well.

References

[1] Timothy Dozat. Incorporating nesterov momentum into adam. 2016.

[2] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011.

[3] Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 1731–1741. Curran Associates, Inc., 2017.

[4] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[5] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[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, 2019.

[7] Martin Popel and Ondřej Bojar. Training tips for the transformer model. *The Prague Bulletin of Mathematical Linguistics*, 110(1):43–70, Apr 2018.

[8] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

[9] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks, 2018.

[10] Samuel L. Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V. Le. Don't decay the learning rate, increase the batch size, 2017.

[11] Cecilia Summers and Michael J. Dinneen. Four things everyone should know to improve batch normalization, 2019.

[12] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[13] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017.

[14] Matthew D. Zeiler. Adadelta: An adaptive learning rate method, 2012.