Speeding up ResNet training

Konstantin Solomatov (06246217), Denis Stepanov (06246218)
Project mentor: Daniel Kang

December 2017

Abstract

Time required for model training is an important limiting factor for faster pace of progress in the field of deep learning. The faster the model training is, the more options researchers are able to try in the same amount of time, and the higher the quality of their results. In this work we stacked a set of techniques to optimize training time of the ResNet model with 20 layers and achieved a substantial speed up relative to the baseline. Our best stacked model trains about 5 times faster than the baseline.

1 Introduction

High accuracy in image recognition and classification needs deeper networks architectures, which leads to slower learning speed. So, training speed is a bottleneck in deep learning research and applications preventing fast iteration times, and limiting a number of experiments per unit of time. In [12], authors managed to train Imagenet [2] on 256GPU cluster in just 1 hour. In this project we strive to achieve substantial speedup on one machine.

In this work we tried stacking many different methods. The best combination we achieved used multi-GPU training, pre-activation, and stochastic depth to achieve more than 5 times reduction in training time on CIFAR-10 dataset [1].

2 Related Work

We haven't found any other works where authors specifically tried stacking multiple techniques to achieve better accuracy. However, during the last two years many approaches were used by researchers to increase learning speed and led to different significant speedups:

- BoostResNet [7] (~3x speedup)
- Stochastic depth [8] (~2x speedup)

- YellowFin auto-tuning optimizer [9] (~1.5x speedup)
- Half-precision [3] (~2x speedup)

In BoostResNet approach authors applied boosting theory and proved that ResNet can be interpreted as a telescoping sum of weak classifiers, which can be trained sequentially and the error exponentially decays as the number of residual blocks grows.

In Stochastic depth approach, the authors addressed the problem of vanishing gradients in deep neural networks and solved the problem by bypassing some randomly chosen layers during the training phase with the identity function. This approach made possible to use even deeper networks and train algorithm in less time.

Using lower precision floats allows to achieve a better training time than on full precision floats without sacrificing model accuracy.

3 Methods

In this work we tried stacking multiple methods. We used CIFAR10 dataset of small 32x32 images due to its small size and wide use among ML researchers. As a baseline we choose the vanilla version of ResNet [11] on one GPU.

We succeeded with the following:

- Multi GPU training. We implemented techniques described in [12]: To make mutli GPU training worthwhile we scaled learning rate proportionally to the batch size, and increased learning rate gradually during the warmup period. We had to use batch size much larger than in the base line model, due to communication overhead between GPUs.
- Stochastic depths. We implemented it according to [8]. The idea is to associate each layer in the ResNet with a probability. During training, in each batch we throw away the layer with that probability. Since, the model which we train becomes smaller, we achieve speedup.
- Pre-activation. Pre-activation is a slight improvement of the ResNet [11] architecture described in [10]. It rearranges layers in a more natural fashion and achieves better performance than the vanilla ResNet. We implemented it due to the fact that BoostResNets are more naturally formulated with it than with the original ResNet model.

We tried the following methods, were able to implement them, but decided not to include them into the final model:

- Half precision. Operations with half precision floats in theory are faster than full precision float operations. Unfortunately speed of our model's training didn't improve when we used half precision API, though, we saved some GPU memory but it wasn't essential for our model.

We tried the following methods but weren't able to reproduce them correctly, and thus didn't include them into the final model:

- BoostResNet. In this approach proposed in [7] boosting theory is used to show, that ResNet can be interpreted as an telescoping weighted sum of weak classifiers, which can be trained sequentially and achieve any specified accuracy level on the training set subject to weak learning condition is fulfilled.

4 Experiments

We used the following setup for experiments. CIFAR10 is split into 2 parts: test and train sets. We used 50000 images from the train set. We used the first 1000 images from the test as the dev set, and the last 9000 images as the test set.

Our code was written in Python. We used PyTorch [4] as our deep learning library. For visualizations and monitoring we used the visdom package [5]. We also used the code from Yellowfin [9] github repo.

For experiments we had 2 machines. One with 1x1080 Ti GPUs and the other one with 4x1080 Ti GPUs, both generously provided for use in this project by our employer JetBrains, Inc.

We tuned the following parameters

- Multi-GPU training:
  - Batch size. We used values from 128 to 3072. With increasing batch size the accuracy was slowly decreasing. From our experiments we found out that the best batch size was 1024.
- Stochastic depth:
  - We compared 2 variants. In one we used range of probability from 1.0 to some p. I.e. first layers had lower probability of being thrown away. In the other variant we had the uniform probability of throwing away layers. We found out that the one with uniform probability performed better, but the other one had better accuracy. Since we are optimizing for speed we chose the uniform variant.
  - We were choosing the probability p of throwing away layer. The higher probability, the slower the model is. On the other hand, lower probability was leading to the loss in accuracy. We found that probability of 0.5 was the best if we consider trade off between speed and accuracy. The stacked model with this probability achieved the same accuracy as the baseline while being relatively fast.
- Stacked model. We tuned the following hyperparameters:
  - Duration of training. We found that the original schedule was the best.

Model	Time	Train	Test
BaseLine	2:42	1.00	0.93
Half precision	2:42	1.00	0.928
Pre activation	2:42	1.00	0.935
x4 GPU	0:49	1.00	0.92
Stochastic Depth	1:30	0.99	0.931
Best Model	0:32	0.991	0.934

Table 1: Result of running experiments

– Learning rate. We found that the original learning rate from the ResNet paper was the best. (It was scaled by Multi-GPU training proportionally to the batch size).

5 Results

For results of running experiments see the Table 1.

The best model included multi GPU training with the batch size of 1024 using a uniform version of the stochastic depth with the probability of 0.5 and the same learning schedule as in the original ResNet paper (though scaled to the larger batch size).

6 Discussions

It's surprising that half precision didn't lead to any improvement. However, there're mentions on the internet that many people weren't able to get expected FP16 performance on 1080Ti.

As has been shown in the [10] ResNet with pre-activated units leads to a slightly better performance, than with baseline units. Pre-activated unit is just a unit with ReLU and BN activation moved before the convolution layer. This form of residual unit is also preferable for the interpretation of ResNet as a telescoping sum of weak classifiers used in [7] in terms of boosting theory approach.

In addition to hypothesis blocks in BoostResNet [7] we introduced additional batch normalization layers to avoid gradient explosion problem. We used exponentiated cross-entropy function as a loss function as was offered by the authors (personal communication). We also performed weak learner condition check to detect the moment, when the next layer can be trained. Unfortunately, our implementation fails to achieve accuracy higher than 0.69 - 0.85 (depending on the loss function and optimizer) on the training set. The average γ_t, characterizing the normalized improvement of the correlation between true labels and hypothesis module's outputs on the layer t was near the value 0.004. Taking the value of error p to be 0.07, according to Theorem E.3. [7], we'll need

$-\frac{2 \ln p}{\gamma^2} \approx 332402$ residual blocks to train in order to achieve accuracy 0.93 having such a small value of γ. We suppose, that the problem is that training deep blocks needs well tuned optimization procedures to be used, because automatic optimizers like Adam [6] and YellowFin [9] failed to produce better results that simple SGD with training schedule.

Distributing on 4GPUs was the most straightforward programatically, but we were surprised by the fact that if the batch size is too small, GPUs aren't completely loaded and there're no improvement in train time. Most likely, this is caused by the fact that we transfer data between GPUs and RAM and this data transfer has a cost which eats all gains achieved by higher parallelization.

An interesting fact about stochastic depth is that it not only achieved faster train time but also acted as a regularizer. The models which contained it in their stack, had lower overfitting compared to other models. We run a number of experiments which measured accuracy and found that with the higher batch size, the accuracy of the stochastic depth was getting lower and lower.

The last thing which we wanted to mention is the importance of having a good infrastructure code. Without it, experiments would be harder to run, and analyze.

7 Conclusion

To summarize, we were able to achieve substantial improvement in training speed without sacrificing accuracy by utilizing relatively straightforward and easy to introduce methods.

8 Future Work

Due to limited amount of time we were able to devote to the project, we weren't able to completely reproduce the following techniques:

- Boosted ResNets [7]
- Yellowfin auto-tuning optimizer [9]

We didn't have time to try the following techniques:

- MeProp [13]

In future work, it would be nice to finish reproducing the methods described, try methods which we didn't have time to try and to see how they stack together. It would be also interesting to see how our approach scales to larger datasets such as ImageNet [2].

9 Contributions

Here's a breakdown by participants:

- Konstantin: infrastructure, baseline, multiple GPUs, Lower Precision, Stochastic Depth.
- Denis: Pre-activated ResNets, Boosted ResNets.

References

[1] URL https://www.cs.toronto.edu/ kriz/cifar.html.

[2] URL http://www.image-net.org/.

[3] URL https://devblogs.nvidia.com/parallelforall/mixed-precision-training-deep-neural-

[4] URL http://pytorch.org/.

[5] URL https://github.com/facebookresearch/visdom.

[6] J. Ba D. P. Kingma. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://arxiv.org/abs/1412.6980.

[7] J. Langford R. Schapire F. Huang, J. Ash. Learning deep resnet blocks sequentially using boosting theory. URL https://arxiv.org/pdf/1706.04964.pdf.

[8] Z. Liu D. Sedra K.Q. Weinberger G. Huang, Y. Sun. Deep networks with stochastic depth. URL https://arxiv.org/pdf/1603.09382.pdf.

[9] C. Ré J. Zhan, I. Mitliagkas. Yellowfin and the art of momentum tuning. *arXiv preprint arXiv:1706.03471*, 2017.

[10] Sh. Ren J. Sun K. He, X. Zhang. Identity mappings in deep residual networks, . URL https://arxiv.org/abs/1603.05027.

[11] Sh. Ren J. Sun K. He, X. Zhang. Deep residual learning for image recognition, . URL https://arxiv.org/abs/1512.03385.

[12] P. Dollar et al. P. Goyal. Accurate, large minibatch sgd: Training imagenet in 1 hour. URL https://arxiv.org/pdf/1706.02677.pdf.

[13] Sh. Ma H. Wang X. Sun, X. Ren. meprop: Sparsified back propagation for accelerated deep learning with reduced overfitting. URL https://arxiv.org/abs/1706.06197.