Weight Initialization, Batch Normalization, and Architecture Choices: Stabilizers for Training Deep Convolutional Networks?

Trevor McInroe

Northwestern University

July 27, 2020

Abstract

Deep convolutional neural networks are difficult to train due to various instabilities, such as vanishing gradients and internal covariate shift. Overcoming these issues is of great importance, as deeper networks have greater representational capacity. Researchers have explored several avenues for introducing stability such as weight initialization, batch normalization, and architecture choices. In this work, we explore various combinations of these factors and record the progress and stability during the training of deep convolutional networks on an image classification task. On our given data and task, we find that the strongest factor can be network architecture, that batch normalization adds stability, and that weight initialization is not influential enough on its own.

Introduction and Problem Statement

As the complexity of the problem grows, so does the required depth of the neural network. However, we cannot simply make networks deeper without consequence. Deep neural networks have been shown to be unstable and difficult to train for a variety of reasons. As such, a significant amount of research effort has gone into identifying the root cause of the instability and suggesting remedies. In this work, we experiment with a variety of proposed solutions implemented on deep convolutional networks trained for the task of image classification. The goal of this study is to determine the relative merits of proposed solutions to training instability on our given problem.

Background and Related Works

Training deep convolutional neural networks (CNNs) is a difficult problem. One of the most prevalent issues occurs during the backpropagation step in the weight updating process. Gradients can become unstable or vanish, which leads to network weights changing in an undesirable way or not changing at all. Much of this instability has been attributed to nonlinearities that tend to saturate their outputs near flat portions of their gradients, such as the sigmoid or tanh nonlinearities (Glorot and Bengio 2010). However, even networks that use nonlinearities without this property (such as the rectifier class of activations) can still show undesirable behavior during training. Lately, a significant amount of research has resulted in various methods for stabilizing the training trajectories of deep neural networks. The approaches pinpoint a variety of network aspects, such as weight initialization, batch normalization, and architecture choices.

Deep CNNs that are initialized with weights drawn from vanilla Normal distributions and fixed standard deviations have been shown to be particularly tough to train (Simonyan and Zisserman 2014). Methods to overcome these issues have been explored, such as pre-training portions of the larger network before training the network as a whole, or adding auxiliary classifiers within the networks themselves (Lee et al. 2014; Szegedy et al. 2014). However, these methods are not ideal as they require additional computational efforts. Instead, a significant amount of research has been focused around intelligent ways to initialize the learnable parameters of deep neural networks (Qiao, Li, and Li 2016; Kathirvalavakumar and Subavathi 2011; Deshpange and Somani 2015; Habibi, Trisilowati, and Wibowo 2015). Most of the successful work in weight initialization focuses on adjusting the variance

of the distributions from which the weights are drawn. Two of the most popular initialization approaches today are He initialization from He et al. (2015b) and Glorot initialization from Glorot and Bengio (2010). He initialization was specifically designed to work well with the rectifier class of nonlinearities and uses layer-dependent distributions where the variance is scaled relative to the number of inputs into the layer. They have concluded that this strategy avoids both reducing and magnifying the magnitudes of each layer's input signals and therefore leads to more stable training (He et al. 2015b). Glorot initialization targets a similar problem but without the specific focus on the rectifier class of nonlinearities. This variance scaling takes into account both the number of inputs into and outputs out of the given layer.

The training of deep neural networks is further complicated by the problem of *internal covariate shift*. This issue can be characterized as the continually shifting distribution of layer inputs. This phenomenon is caused by the learning process of the network itself. As $layer_n$ updates its weights, the distribution of inputs to $layer_{n+1}$ changes. This shifting behavior is extremely prevalent in deep networks, where small shifts in weights early in the network can grow to large changes in input distributions for layers deep in the network. Internal covariate shift can pose problems for network convergence through causing activations to saturate or for gradients to explode (Geron 2017). This issue motivated the invention of batch normalization, a method for stabilizing outputs of network layers (Ioffe and Szegedy 2015). The key contribution of this work is the implementation of a pair of parameters added to each of the network's activation functions that zero-centers and normalizes their input on a batch-by-batch basis. The authors conclude that the addition of batch normalization allows for more aggressive learning schedules and training that is more stable.

Further work has targeted the design of the networks themselves. Work from He et al. (2015a) introduced the *residual block*, a key component in most state of the art deep CNNs today. A residual block allows its input to flow through a series of convolutions, as well as around the convolutions. These two paths are concatenated together by addition before being passed through an activation. Several versions of a residual block exist. As an example, imagine a residual block consisting of three convolutions, (C_1, C_2, C_3), where the input passes both through C_1 and C_2. The result from C_2 is then passed through C_3. Finally, the output from C_1 is added to the output from C_3 and passed through a nonlinearity. A common misconception of residual blocks is that they alleviate the vanishing gradient problem. While a complete explanation of the success of residual blocks is not clear, it is theorized that the residual connections make learning the identity mapping between a residual block's input and output easy. This identity mapping allows for deeper networks to theoretically perform no worse than a shallower version (He et al. 2015a).

Methods and Data

Data

For this study, we have used the Food-101 dataset introduced by Lee et al. (2018). The dataset contains 101,100 full-color images of food categorized into 101 mutually exclusive classes. We have rescaled the images to 128x128 before training our networks.

Initialization Types

For both the Glorot and He initializations, we have used the Gaussian-adaptation:

$$\boldsymbol{\theta}_{He}^{(\ell)} \sim \mathcal{N}^{\mathcal{T}}(0, \frac{2}{\hat{n}^{(\ell)}})$$
$$\boldsymbol{\theta}_{Glorot}^{(\ell)} \sim \mathcal{N}^{\mathcal{T}}(0, \frac{2}{\hat{n}^{(\ell)} + \hat{n}^{(\ell+1)}})$$

where $\mathcal{N}^{\mathcal{T}}$ is a truncated-Normal distribution, $\boldsymbol{\theta}^{(\ell)}$ are the learnable weights and $\hat{n}^{(\ell)}$ is the number of inputs for layer ℓ. In addition, biases are initialized to zero. The distribution is truncated by re-drawing every value that lies outside of $\pm 2\sigma$ from the mean [1].

Models

We have tested two different deep convolutional architecture schemes, one with and one without residual blocks. We have trained a version of each architecture type with and without batch normalization as well as using both weight initialization schemes outlined in the preceding section. All eight network versions were trained with the Adam optimizer from Kingma and Ba (2014) and use cross-entropy loss. To help observe the stability of the learning process, each network combination has been trained three times. In our graphs, we have reported on the average as well as the minimum and maximum classification accuracy over 50 epochs of training. To aid in computation, GPU acceleration via TensorFlow was used as well as asynchronous mini-batch fetching by the CPU [2].

---

1. For an in-depth look, see the TensorFlow source code:
https://github.com/tensorflow/tensorflow/blob/v2.2.0/tensorflow/python/ops/init_ops_v2.pyL392

2. GPU: GTX 1080ti, CPU: AMD Ryzen Threadripper 1900x

YOLO

For our network without residual blocks, we implemented a modified version of the "You Only Look Once" (YOLO) model introduced by Redmon et al. (2016) called "Tiny YOLO". This modification earned its namesake by being a version of YOLO that contains a smaller number of layers. YOLO is originally an architecture for object detection that outputs region proposal bounding boxes and class predictions. For our exercise, we removed the "head" of YOLO and replaced it with a sequence of two dense layers with 1,470 and 101 nodes. For a full view of the convolutional stacks preceding these layers, see Table 1. The version of the network with batch normalization has 10,311,833 learnable parameters, and the version without batch normalization has 10,307,257 learnable parameters. For our nonlinearities, we used Leaky ReLU as introduced by Mass, Hannun, and Ng (2013) and the softmax function after the output of the network.

ResNet

For our network with residual blocks, we used the ResNet architecture developed by He et al. (2015a). For our study, we implemented the ResNet26 version of the architecture, which utilizes the ReLU nonlinearity within the network and the softmax activation at the end of the network. The residual blocks contain two convolutions in the feed-forward section and one convolution in the skip-connection. The network ends with a global pooling layer, followed by a single dense layer with 101 units. For a full picture of the convolutional stacks preceding these two layers, see Table 2. The version of the network that uses batch normalization has 11,237,413 learnable parameters, and the version that does not use batch normalization has 11,227,813 learnable parameters.

Analysis of Results

Observing Figure 1, we note three results. First, across both architectures and both initialization types, batch normalization allowed for extremely consistent training trajectories that amounted to near-perfect accuracy. This suggests that batch normalization may be a sound technique for introducing stability.

Second, when no batch normalization is used, the architecture without explicit design considerations for stability (Tiny-YOLO) exhibits a poor training trajectory in terms of consistency and accuracy for both initialization types. This suggests that weight initialization, alone, may not be enough to ensure success when training deep convolutional networks.

Thirdly, the architecture with purposeful stability-giving design (ResNet26) exhibits training stability and accuracy *nearly* as good without batch normalization as it does with batch normalization. This suggests that network architecture plays a significant role in the training stability of deep convolutional networks.

Conclusions

In this work, we examined the effects of various network choices, such as weight initialization, batch normalization, and architectures on the training stability of deep convolutional networks for image classification. We found that weight initialization, alone, is not enough to guarantee non-divergent behavior. We showed that batch normalization can grant stability in networks that would otherwise be unstable. However, from our results, we concluded that architecture choice may be the strongest determinant. Specifically, we observed that the inclusion of residual blocks in a deep convolutional network provides a significantly higher likelihood of success.

Future Work

Image classification is just one type of task in one specific domain of deep learning. What remains to be seen is if these results are consistent across other areas of computer vision or even other deep learning tasks.

Appendix

Table 1. Convolutional stacks of Tiny-YOLO. All strides are 1, all activations are Leaky ReLU.

Layers	**Kernel size**	**Filterss**
Conv2d	3x3	16
BatchNorm		
MaxPool	2x2	
Conv2d	3x3	32
BatchNorm		
MaxPool	2x2	
Conv2d	3x3	64
BatchNorm		
MaxPool	2x2	
Conv2d	3x3	128
BatchNorm		
MaxPool	2x2	
Conv2d	3x3	256
BatchNorm		
MaxPool	2x2	
Conv2d	3x3	512
BatchNorm		
MaxPool	2x2	
Conv2d	3x3	1024
Conv2d	3x3	256

Table 2. Convolutional stack of ResNet28. Layers involved in skip-operations highlighted in blue.

Layers-strides	Kernel size	Filters
Conv2d-2	7x7	64
BatchNorm		
MaxPool-2	2x2	
Conv2d-2	3x3	64
BatchNorm		
Conv2d-1	3x3	64
BatchNorm		
Conv2d-1	3x3	64
BatchNorm		
Conv2d-1	3x3	64
BatchNorm		
Conv2d-1	3x3	64
BatchNorm		
Conv2d-1	3x3	64
BatchNorm		
Conv2d-2	3x3	128
BatchNorm		
Conv2d-1	3x3	128
BatchNorm		
Conv2d-1	3x3	128
BatchNorm		
Conv2d-1	3x3	128
BatchNorm		
Conv2d-1	3x3	128
BatchNorm		
Conv2d-1	3x3	128
BatchNorm		
⋮	⋮	⋮

Layers-strides	Kernel size	Filters
⋮	⋮	⋮
Conv2d-1	3x3	256
BatchNorm		
Conv2d-1	3x3	256
BatchNorm		
Conv2d-1	3x3	256
BatchNorm		
Conv2d-1	3x3	256
BatchNorm		
Conv2d-1	3x3	256
BatchNorm		
Conv2d-1	3x3	256
BatchNorm		
Conv2d-2	3x3	512
BatchNorm		
Conv2d-1	3x3	512
BatchNorm		
Conv2d-1	3x3	512
BatchNorm		
Conv2d-1	3x3	512
BatchNorm		
Conv2d-1	3x3	512
BatchNorm		
Conv2d-1	3x3	512
BatchNorm		

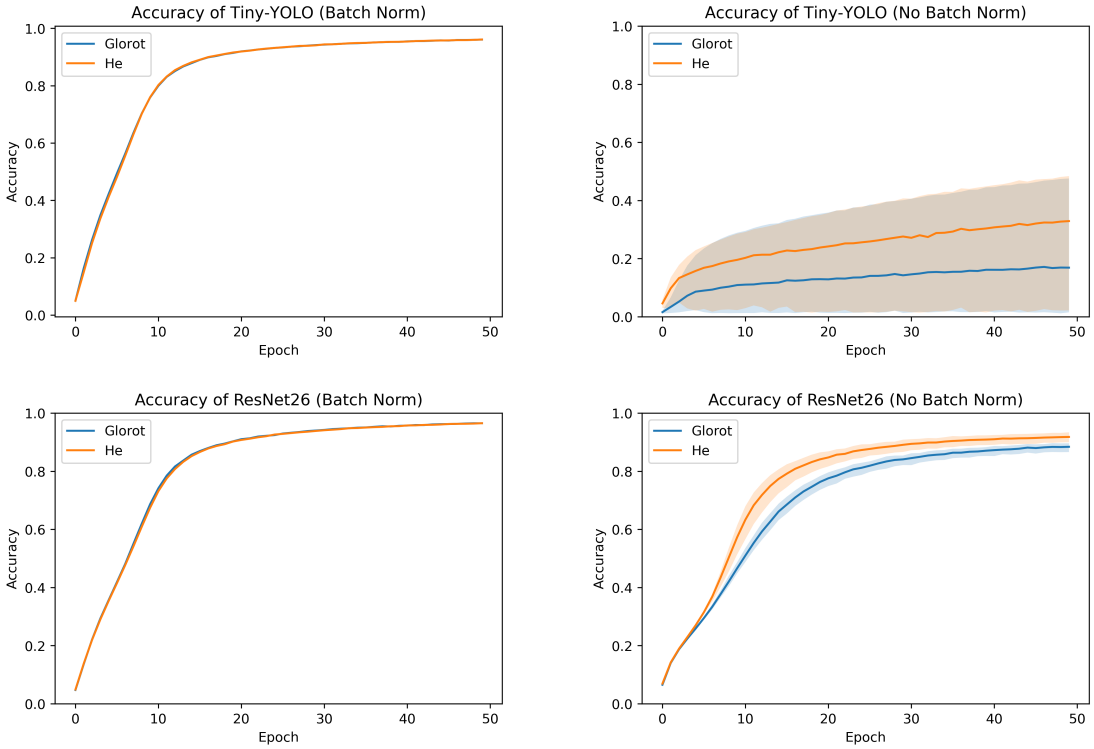

Figure 1. Training accuracy and stability. Rows differentiate model architecture. Columns differentiate use of batch normalization.

Bibliography

Deshpange, Ameet, and Vedant Somani. 2015. "Weight Initialization in Neural Language Models". arXiv: 1805.06503[cs.CL].

Geron, Aurelien. 2017. *Hands-on Machine Learning with Scikit-Learn and TensorFlow*. Sebastopol, CA: O'Reilly.

Glorot, Xavier, and Yoshua Bengio. 2010. "Understanding the difficulty of training deep feedforward neural networks". In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, ed. by Yee Whye Teh and Mike Titterington, 9:249–256. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR. http://proceedings.mlr.press/v9/glorot10a.html.

Habibi, A.R., R.B.E. Trisilowati, and R.B.E. Wibowo. 2015. "Modification of neural network algorithm using conjugate gradient with addition of weight initialization". *Journal of Theoretical and Applied Information Technology* 77 (1): 1–7. ISSN: 19928645.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015a. "Deep Residual Learning for Image Recognition". arXiv: 1512.03385[cs.CV].

— . 2015b. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". arXiv: 1502.01852[cs.CV].

Ioffe, Sergey, and Christian Szegedy. 2015. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". arXiv: 1502.03167v3[cs.LG].

Kathirvalavakumar, Thangairulappan, and Subramanian Jeyaseeli Subavathi. 2011. "A new weight initialization method using Cauchy's inequality based on sensitivity analysis.(Report)". *Journal of Intelligent Learning Systems and Applications* 3 (4): 242. ISSN: 2150-8402.

Kingma, Diederik P., and Jimmy Ba. 2014. "Adam: A Method for Stochastic Optimization". arXiv: 1412.6980[cs.LG].

Lee, Chen-Yu, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. 2014. "Deeply-Supervised Nets". arXiv: 1409.5185[stat.ML].

Lee, Kuang-Huei, Xiaodong He, Lei Zhang, and Linjub Yang. 2018. "CleanNet: Transfer Learning for Scalable Image Classifier Training with Label Noise". arXiv: 1711.07131[cs.CV].

Mass, Andrew, Awni Hannun, and Andrew Ng. 2013. "Rectifier Nonlinearities Improve Neural Network Acoustic Models". In *International Conference of Machine Learning*, vol. 30.

Qiao, Junfei, Sanyi Li, and Wenjing Li. 2016. "Mutual information based weight initialization method for sigmoidal feedforward neural networks". *Neurocomputing* 207:676–683. ISSN: 0925-2312.

Redmon, Joseph, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. "You Only Look Once: Unified, Real-Time Object Detection". arXiv: `1506.02640[cs.CV]`.

Simonyan, Karen, and Andrew Zisserman. 2014. "Very Deep Convolutional Networks for Large-Scale Image Recognition". arXiv: `1409.1556[cs.CV]`.

Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragonmir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2014. "Going Deeper with Convolutions". arXiv: `1409.4842[cs.CV]`.