Batch Normalization for Improved DNN Performance, My Ass

Joshua A. Wise
joshua@joshuawise.com
Emarhavil Heavy Industries

Abstract

Batch normalization is an extremely popular technique to enable faster training, and higher network performance after training. We apply batch normalization to a relatively small network, and find it to be completely ineffective, and indeed, to reduce network convergence and overall network performance.

1. Introduction

Batch normalization [4] is a strategy used to accelerate learning in deep neural networks. Theorized to work by reducing "internal covariate shift", it dynamically computes normalization coefficients at each channel internal to a convolutional network while training, and then during validation and operation, hopes that they generalize. Although the effect of batch normalization can, in theory, be baked into weights at each neuron, the batch normalization coefficients are not learned through gradient descent, and only their second-order effects are. Through a convoluted process, this means that adding more parameters somehow makes the network converge more readily, and so everybody does it.

Batch normalization has been used in many networks from deep to shallow: recent DCGAN architectures (for instance, `pix2pix` [5]) have used batch normalization between layers when training regression, and Google's Inception network has used it when training classification. Batch normalization is said to be tolerant to hyperparameters; for instance, the `decay` hyperparameter is said to reasonably range from 0.999 through 0.99 all the way down to 0.9 and "etc.", which is apparently one nine fewer than 0.9. It also has a configurable value of `epsilon` [2], which is likely to be valuable during times of shortage [9].

In this work, we sprinkle batch normalization pixie dust onto an existing neural network to improve its performance, and analyze the performance gained.

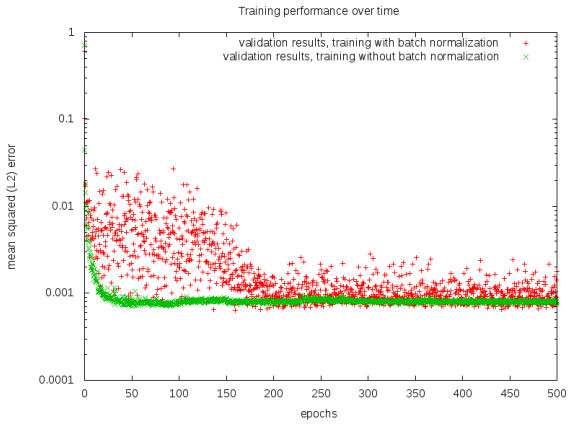

Figure 1. Batch normalization performance vs. classical training.

2. Related Work

Everybody who does work on deep neural networks cites the founding paper on the subject that was written long before anyone had ever heard of a GPU. So we do so here too [7]. But let's be real here, this whole lab report is actually a take-off of Kovar and Hall [6], who did this way better than I did.

3. Experimental Procedure

We took an existing neural network of a few layers, a corruption of the work in [3]. It already did not work very well, but the batch normalization pixie dust was expected to substantially improve it, and make everything all better. We inserted batch normalization layers in all but the final convolutional layer, since adding a normalizing layer before the output seemed obviously stupid and likely to produce absurd nonlinearities.

The batch normalization layer was built using TensorFlow's `tf.contrib.layers.batch_norm` [10] function. (The `contrib` in the Python module path means that the routine is extra-well-tested.) We experimented with multiple sets of hyperparameters, primarily because the first set of hyperparameters were no good. The initial set of hyperparameters used a value of 0.9 for `decay` and a value of 10^{-5} for `epsilon`, because

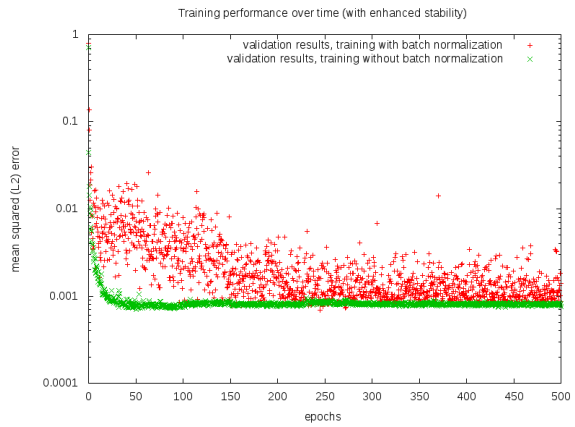

Figure 2. Batch normalization performance, with stability enhancements.

that's what pix2pix did. The results were, hey, wait, this is the wrong section for that.

The second set of hyperparameters used increased the `decay` coefficient to 0.999, and enabled `zero_debias_moving_mean`, because it is said that one should do that if one's results are unstable.

Training on both runs took place overnight using TensorFlow on 8 NVIDIA Really Big GPUs in parallel. On the new power-efficient Pascal architecture, training consumed approximately 1.5 kW, for 12 hours, or 18 kWh of total power, or enough for my coworker to boil 540 cups of tea [1].

4. Results

The results were utter crap. The first run was dramatically unstable (see Figure 1). When measures were taken to make the system more stable, it responded in the opposite fashion (see Figure 2). Convergence did not happen faster than without batch normalization, inasmuch as anything that the batch normalization runs did could be at all described as converging.

Visual quality of the output batch-normalized runs was not verified, because, let's face it, it's going to be noisy crap. Also, I didn't finish writing the support to load and save the batch-normalization coefficients into checkpoint files, so that's another strike against that.

5. Future Work

Maybe someone can get this crap to work. Like, everyone else who sprinkles batch normalization pixie dust on their CNNs gets them to train right quick, and the Google folks say that you don't even need L_2 normalization with them, let alone any other kind of normalization. Work should be done to investigate whether the Google folks just got a really lucky RNG seed each time they did their batch-norm runs, or maybe a really bad one for their control runs, because clearly this stuff ain't working for me.

Other experiments could be run with other normalization schemes, like Dropout [8]. Initial experiments are under way that indicate that all of the literature about Dropout is also a lie.

6. Conclusion

I still don't know anything about how neural networks work, and as far as I can tell, neither does anyone else.

References

[1] Personal correspondence.

[2] Tom 7. What, if anything, is epsilon? *SIGBOVIK*, you know, like the only one that year, 2014.

[3] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *CoRR*, abs/1501.00092, 2015.

[4] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[5] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. *CoRR*, abs/1611.07004, 2016.

[6] Lucas Kovar. Electron band structure in germanium, my ass. Online, `http://pages.cs.wisc.edu/~kovar/hall.html`, 2007.

[7] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Comput.*, 1(4):541–551, December 1989.

[8] Infected Mushroom. Drop out. From the album, *Converting Vegetarians, Disc 2*, 2003.

[9] Chris Tuffley. The great epsilon shortage. *Mathematical Intelligencer*, 21(4):37, 1999.

[10] Someone who didn't proofread their code sample. Tensorflow API documentation. Online, `https://www.tensorflow.org/api_docs/python/tf/contrib/layers/batch_norm`, 2017.