FASTNORM: IMPROVING NUMERICAL STABILITY OF NEURAL NETWORK TRAINING WITH EFFICIENT NORMALIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose a modification to weight normalization techniques that provides the same convergence benefits but requires fewer computational operations. The proposed method, FastNorm, exploits the low-rank properties of weight updates and infers the norms without explicitly calculating them, replacing an $O(n^2)$ computation with an $O(n)$ one for a fully-connected layer. It improves numerical stability and reduces accuracy variance enabling higher learning rate and offering better convergence. We report experimental results that illustrate the advantage of the proposed method.

1 INTRODUCTION

Achieving efficient convergence while training deep networks is a known challenge. One of the reasons for this difficulty is the change in the distributions of layer inputs caused by the updates in the parameter values in the course of training. This phenomenon, known as the internal covariate shift (Shimodaira, 2000), skews activation distributions over time. This distortion gets amplified along the depth of the network may in turn lead to the vanishing gradient problem (Hochreiter et al., 2001) impeding or preventing convergence.

Normalization is a common approach to limit the internal covariate shift and to keep parameters from reaching extreme values. In particular, batch normalization introduced by Ioffe & Szegedy (2015) is very successful in stabilizing and accelerating the learning process whitening layer inputs for each training mini-batch.

While very successful, batch normalization relies on sufficiently large batches to work effectively. More recent techniques adapt the approach of batch normalization to smaller batches or to online learning tasks. They include Batch Renormalization (Ioffe, 2017) and Layer Normalziation (Ba et al., 2016).

Another class of algorithms normalize the weights rather than layer inputs. Weight Normallization (Salimans & Kingma, 2016) and Normalization Propagation (Arpit et al., 2016) have convergence properties comparable to batch normalization without relying on mini-batches to work. However, normalizing a weight matrix directly is computationally expensive, as it requires processing each entry of the matrix, taking the square root of the sum of squares, and finally updating each entry of the matrix.

We propose FastNorm, a purely mathematical acceleration of weight normalization techniques. For the fully connected case we derive a method to track the norm of the weights without computing it and we apply those norms efficiently to avoid updating the weights explicitly. Because total number of operations performed is reduced but the internal covariate shift is still mitigated, training proceeds with more stability, and a higher learning rate can be used. In the convolutional case, we can generalize the method to compute the norms in the standard manner but continue to apply them more efficiently.

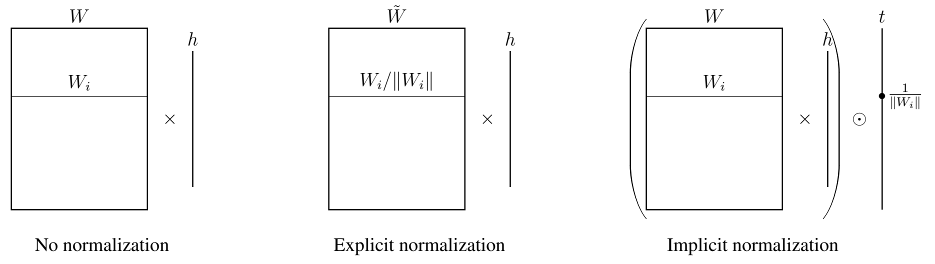

Figure 1: Weight normalization: forward pass.

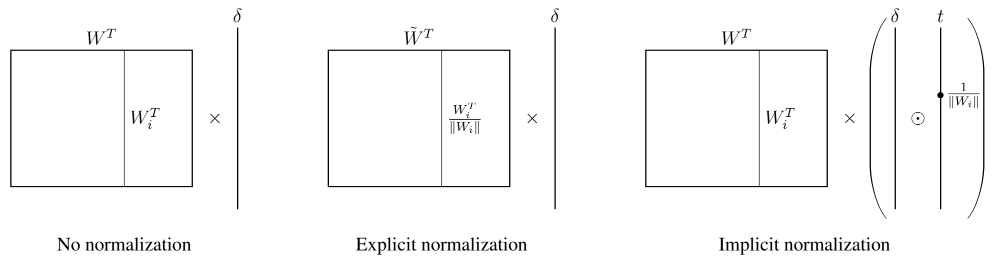

Figure 2: Weight normalization: delta pass.

2 IMPLICIT WEIGHT NORMALIZATION

On each update weight normalization methods adjust the weight matrix to normalize its rows. For a fully connected layer the application of the weight matrix W to the input h on the forward pass becomes

$$\begin{aligned} z_i &= \gamma_i \frac{\sum_k (w_{ik} h_k)}{\sqrt{\sum_m w_{im}^2}} + \beta_i \\ &= \gamma_i \frac{W_{i\cdot} h}{\|W_{i\cdot}\|} + \beta_i \\ &= \gamma_i \tilde{W}_{i\cdot} h + \beta_i \ , \end{aligned} \tag{1}$$

where $W_{i\cdot}$ and $\tilde{W}_{i\cdot}$ are the i-th row of the original and normalized weight matrices and β and γ are trainable rescaling parameters (Arpit et al., 2016). The gradient with respect to weight matrix entries is

$$\frac{\partial z_i}{\partial w_{ij}} = \gamma_i \frac{\|W_{i\cdot}\| h_j - \frac{w_{ij}}{\|W_{i\cdot}\|} W_i h}{\|W_{i\cdot}\|^2} \tag{2}$$

and once the corresponding error term δ is computed, the update for a row of the weight matrix with learning rate α can be written as

$$W_i' = W_i - \alpha \delta_i \gamma_i \frac{||W_i|| h^T - \frac{W_i}{||W_i||} W_i h}{||W_i||^2} \ . \tag{3}$$

As expected, the normalization procedure forces the gradient to be orthogonal to the matrix row W_i. We can use it to compute the norm of the updated W_i'.

$$\begin{aligned} ||W_i'||^2 &= W_i' W_i'^T \\ &= ||W_i||^2 + \alpha^2 \delta_i^2 \gamma_i^2 \frac{||W_i||^2 h^T h - 2(W_i h)^2 + (W_i h)^2}{||W_i||^4} \\ &= ||W_i||^2 + \frac{\alpha^2 \delta_i^2 \gamma_i^2}{||W_i||^2} \left(h^T h - \frac{(W_i h)^2}{||W_i||^2} \right) \ . \end{aligned} \tag{4}$$

Note, that unlike explicit normalization of the weight matrix, computation of row norms using the update expression (4) doesn't require $O(n^2)$ operations. Instead, it requires to compute a single

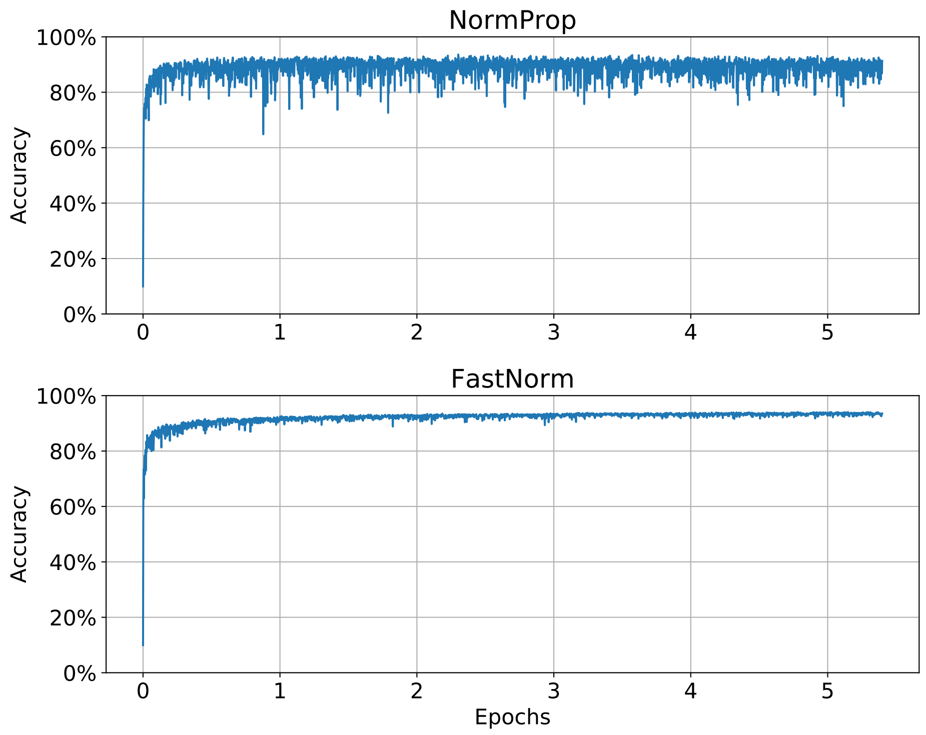

Figure 3: Convergence of a 2-layer fully-connected network with MNIST data set. Validation accuracy was sampled every 100 iterations.

vector norm $h^T h$ amortized over all values of i and reuses already computed values $W_i h$ brining the total number of operations to $O(n)$.

Instead of explicitly normalizing W at each step we can keep the matrix unnormalized but define $t_i = 1/\|W_i\|$ and maintain a vector t with these scalars. In this notation the forward pass (1) becomes

$$z_i = \gamma_i t_i (W_i h) + \beta_i \ . \tag{5}$$

Figure 1 shows a schematic comparison of explicit and implicit normalization (we removed the term γ for simplicity). The update for the norm-tracking vector expressed from (4) can be written as

$$t'_i = \left(\frac{1}{t_i^2} + \alpha^2 \gamma_i^2 \delta_i^2 t_i^2 \left(||h||^2 - (W_i h)^2 t_i^2 \right) \right)^{-\frac{1}{2}} \ . \tag{6}$$

Finally, we need to adjust the back propagation pass to include the effect of normalization. When propagating the error using the unmodified weight matrix we need to apply normalization factors t_i to mimic the effect of multiplying by the scaled matrix.

We use δ' to denote the δ passed to the next layer. Ordinarily, $\delta' = W^T(\gamma \odot \delta)$, where we use $\odot$ for component-wise multiplication. Explicit normalization will follow the same patter with normalized matrix $\tilde{W}$

$$\tilde{W}^T(\gamma \odot \delta)$$

As in the forward pass, for implicit normalization we use the original weight matrix W and absorb normalization step in component-wise multiplication by t (Figure 2).

$$\delta' = W'^T (t \odot \gamma \odot \delta) \tag{7}$$

Equations (5,6,7) define FastNorm algorithm. We assume that the weight matrix is normalized after initialization and subsequent implicit updates are tracked by scalars t_i. In practice we can perform infrequent explicit normalizations (are reset t_i values to 1) to make sure that tracking scalars are fully representable especially if the computation is performed in low precision. Our experiments show that explicit normalization can be done approximately once per epoch for 16-bit computations to keep tracking scalars within the representable range. Extra computational cost of such infrequent normalizations is negligible.

3 Computational properties

3.1 Theoretical savings

For a fully connected layer with m inputs and n outputs both explicit weight normalization and proposed method require m square roots and m inverse operations. In addition explicit normalization requires $2n$ operations to compute the norm of each of m rows. It also requires mn operations to rescale the weight values resulting in normalization cost of $3mn$ plus m divisions and m square roots.

A normalization step performed by the proposed algorithm requires $2n$ operations to compute $||h||^2$, then for each row it can reuse already computed value $(W_i h)t_i$ to compute the difference term in (6) in two operations with additional 8 operations to compute the rest of the expression in (6) plus a division and a square root. The back propagation step adds another m operations to scale the result with t. The total operation count for that FastNorm implicit normalization adds is $n + 11m$ plus m divisions and m square roots.

The computational complexity reduction is possible because at each step the weight matrix undergoes a simple low rank update: without normalization it's just the outer product of activation and error vectors. Because of that we can recalculate updated norms of rows of the weight matrix without explicit normalization.

3.2 Numerical stability

Computational savings have another positive effect: the reduction in the number of operations from $O(n^2)$ to $O(n)$ also reduces the accumulated rounding error leads to a more stable training process.

We used a two-layer fully connected network with the MNIST data set to compare NormProp and FastNorm. Figure 3 compares the convergence behavior for this network for both techniques. Just observing the shape of the curves we can conclude that applying FastNorm results in a more steady increase in accuracy.

To quantify the variability over time, we compare the standard deviations of accuracy levels for both methods (Figure 4). We observe close to five-fold reduction in variance when changing the explicit normalization to the implicit scheme of FastNorm.

Because of the gained stability, we can use FastNorm with a slightly higher learning rate. We observed that NormProp diverges with a learning rate of 0.07, but FastNorm can handle a learning rate of 0.075. In one epoch, with a learning rate of 0.06, NormProp achieved 86.47% accuracy while FastNorm achieved 90.97% accuracy with a learning rate of 0.075. Even when both networks were run with the same learning rate of 0.03, FastNorm slightly outperformed NormProp in resulting accuracy.

4 Generalizations of the method

The weight matrix update (3) uses the gradient computed from a single sample. More generally, the Stochastic Gradient Descent method applies the update averaged over a mini-batch of samples. For small batches that results in a low-rank update to the weight matrix and can be handled similarly to the approach outlined in Section 2. As the batch size grows the method leads to diminishing returns and ultimately becomes less efficient that explicit normalization.

Similarly, convolution layers require special handling. The derivation presented in Section 2 doesn't directly apply to convolutions because of the specific structure of corresponding computations.

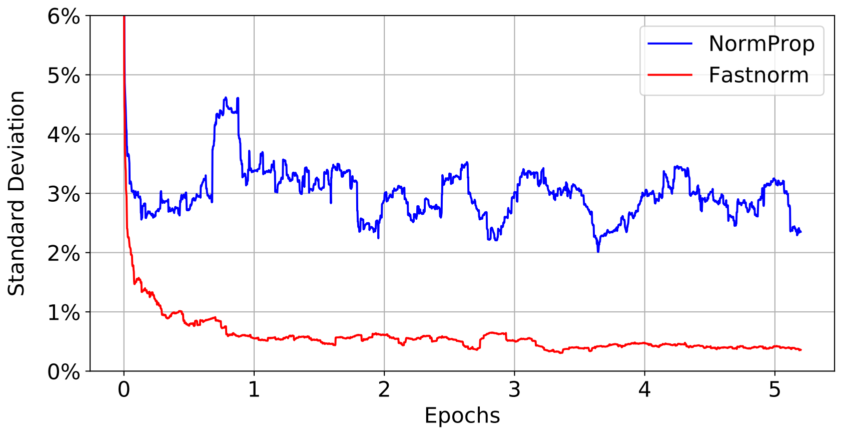

Figure 4: Standard deviations of accuracies (measured every 100 iterations) using a sliding window of size 100 samples for two-layer network normalized with NormProp, and FastNorm.

An extension to the proposed method is a hybrid approach in which we compute the norms explicitly but don't apply them to the weights and use scalars t_i to track the effects of normalization. This still results in numerical savings: an $O(n^2)$ operation of weight matrix rescaling gets replaced with an $O(n)$ scheme of implicit normalization.

5 Conclusions

FastNorm provides a normalization method that is mathematically equivalent to explicit weight normalization in infinite precision but provides computational savings and increased stability in the network by implicitly normalizing weights. Because of that stability, FastNorm tolerates a higher learning rate and as the result reaches convergence faster.

The approach or applying implicit or lightweight normalization can be extended to cases that include batching and convolutions.

References

Devansh Arpit, Yingbo Zhou, Bhargava Urala Kota, and Venu Govindaraju. Normalization propagation: A parametric technique for removing internal covariate shift in deep networks. *CoRR*, abs/1603.01431, 2016.

Lei Jimmy Ba, Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016.

Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jrgen Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

Sergey Ioffe. Batch renormalization: Towards reducing minibatch dependence in batch-normalized models. *CoRR*, abs/1702.03275, 2017.

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

Tim Salimans and Diederik P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 901, 2016.

Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference*, 90(2):227–244, October 2000.