Weight initialization strategies for Binarized Neural Networks

Aidan.B.Clark [1 2] **Vinay Uday Prabhu** [1] **John Whaley** [1]

Abstract

To train a Binarized Network Network (BNN), one needs a set of full precision weights whose initial value forms the basis for the first iteration of binary weights. The BinaryConnect paper (Courbariaux et al., 2015) and those that followed do not mention their method of weight initialization, but open-source implementations found online point to uniform distributions in $[-1, 1]$ or random matrices of $\{1, -1\}$ to be the most common. From parallel research in full precision neural networks, there is no reason to assume this is the best choice. We discuss several alternate approaches towards weight initialization in binary networks.

1. Introduction

Binarized Network Networks (BNNs) constitute an important deep-net compression strategy that has been shown to be extremely promising in terms of both inference time as well as memory footprint while still achieving state-of-the-art accuracy. As shown in (Courbariaux et al., 2016), deploying BNNs results in up to $32\times$ network compression and a dramatic reduction in inference time given that the 32-bit floating point multiply accumulations can now be replaced by 1-bit `xnor-popcnt` operations. In this extended abstract we look at the problem of optimal weight initialization to be adopted during the training of these BNNs and propose a *Hadamard initialization* strategy that outperforms the other standard approaches prevalent in current literature.

1.1. {1, -1} Initialization

The first and most directly binary method of initialization one might consider is to initialize all weights as random matrices of $\{1, -1\}$. In practice, this means that a large number of gradient updates must occur for any change to be reflected into the binary weights, so a more practical choice is to initialize weights to be a random matrix of $\{\alpha, -\alpha\}$, where α is a small value near 0.

1.2. Skewed Binary Initialization

An alteration that can be made is to once again set the matrices to lie in $\{1, -1\}$, but to specify the ratio between the number of -1s and 1s. This can be done either throughout the whole matrix or by row or by column. If we let p be the probability of assigning 1 to a given weight, we can notice that forcing a given ratio of 1 to -1 in a column will have the effect of increasing the correlation of neuron firings as p increases. Interpreting the result of forcing a $1s$ and -1s ratio of p in each row (or across the whole matrix) is less obvious.

1.3. Standard Initialization

A perhaps-obvious method of weight initialization would be to initialize full precision weights normally, with no regard to the binary nature of the network. In practice, for us this meant initializing the weights via the He method from (He et al., 2015). This works better than one might expect.

1.4. Orthogonal and Hadamard Initializations

There is evidence that initializing weights to be orthogonal or orthonormal leads to increased performance (Saxe et al., 2013). Such weight matrices are easy to create using the Singular Value Decomposition in full precision. One method of pseudo-orthogonal initialization in binary networks would be to initialize full precision weight matrices to be orthogonal via canonical methods, then directly binarize. With high likelihood, however, the resulting binary matrix will not be orthogonal. Fortunately, there is a method to directly create orthogonal binary matrices.

A Hadamard Matrix is a square matrix W whose values lie in $\{1, -1\}$, whose rows and columns are pairwise orthogonal, and which has the property that $W^T W = I$. Hadamard Matrices are believed to exist for all shapes $(4n, 4n)$ where $n \in \mathbb{N}$, but are easily constructable only for shapes of the form $(2^n, 2^n)$. Luckily, due to hardware optimalities, many weight matrices are of this shape.

Furthermore, it is easy to produce $\{1, -1\}$ matrices with row orthogonality of shape $(m, 2^n)$ for any m by randomly duplicating or deleting rows from a Hadamard Matrix[1]. Matrices "close" to orthogonal of *any* shape can also be

[1]Even after tie-breaking from the bias vector, this might cause duplicate rows in the weight matrix. This is an issue already discussed in (Courbariaux & Bengio, 2016), with the result affirming the utility of such duplicates.

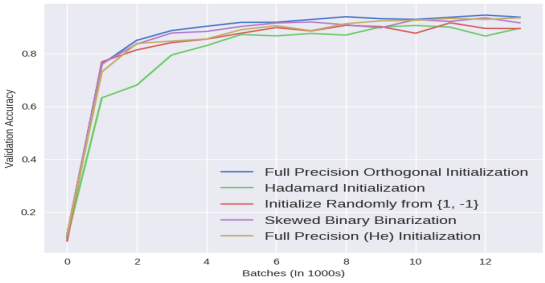
constructed by duplicating or deleting columns from the closest Hadamard matrix. We will call Hadamard Initialization the process of initializing weight matrices to be Hadamard or "close" to Hadamard via this process. Like matrices randomly sampled from $\{1, -1\}$, it is beneficial to introduce a scaling factor to speed up initial training. We multiply each Hadamard matrix element-wise by the absolute value of a random matrix sampled via the He method, which empirically provides an improvement over a normal scaling factor.

2. Experiments and Results

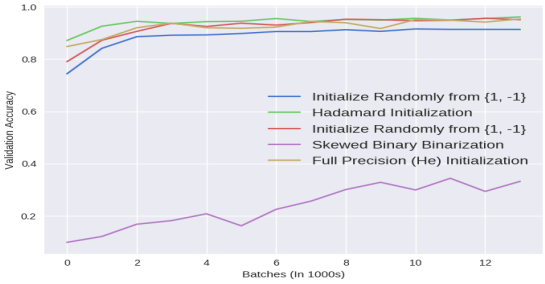
We wish to compare strategies for weight initialization. We test results on two networks: a standard feed-foward binary network and on a small CNN, both on MNIST. Our CNN involved three convolutional layers with 128, 256 and 512 filters respectively, interwoven with max pooling, and followed by two fully connected layers of 1024 neurons each. Our feed-forward network was a 20-layer deep network with 1024 neurons in the first layer and 512 neurons in every layer that followed. In both cases we use the Binomial Estimator as previously described. For each, we test the following initializations:

1. Initializing weights to random matrices of $\{1, -1\}$.
2. Using standard (He) initialization of full precision weights, and directly binarizing these.
3. Random matrices with high neuron correlation, such that each column of a weight matrix is expected to contain 75% of one weight (either 1 or -1) and 25% of the other.
4. Initializing weights as binarizations of orthogonal full precision matrices.
5. Initializing weights using the Hadamard Initialization described above.

Results are shown in Fig 1. We see that for (shallow) convolutional networks, the weight initializations are all roughly equal, with He and (full-precision) orthogonal initialization performing marginally better. For a deeper feedforward network, we see that the Hadamard initialization performs best, with the others grouped with slightly less accuracy. Interestingly, initializing weights with high correlation causes extremely slow learning behavior on the deep binary feed-forward network.



(a) Accuracy curves on MNIST for a Binary Convolutional Neural Network with three binary convolutions, interspersed with max pooling, followed by two fully connected layers.



(b) Accuracy curves on MNIST for a 20-layer deep feedforward network.

Figure 1. Testing weight initializations on convolutional and feedforward networks with MNIST. On the left is a Binary Convolutional Network, on the right is a 20-layer feedforward binary network.

References

Courbariaux, Matthieu and Bengio, Yoshua. Binarynet: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

Courbariaux, Matthieu, Bengio, Yoshua, and David, Jean-Pierre. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*, pp. 3123–3131, 2015.

Courbariaux, Matthieu, Hubara, Itay, Soudry, Daniel, El-Yaniv, Ran, and Bengio, Yoshua. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

Saxe, Andrew M, McClelland, James L, and Ganguli, Surya. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.