THE HIGH-DIMENSIONAL GEOMETRY OF BINARY NEURAL NETWORKS

Alexander G. Anderson[*] **& Cory P. Berg**
Redwood Center for Theoretical Neuroscience
University of California, Berkeley
{aga, cberg500}@berkeley.edu

ABSTRACT

Traditionally, researchers thought that high-precision weights were crucial for training neural networks with gradient descent. However, recent research has obtained a finer understanding of the role of precision in neural network weights. One can train a NN with binary weights and activations at train time by augmenting the weights with a high-precision continuous latent variable that accumulates small changes from stochastic gradient descent. However, there is a dearth of theoretical analysis to explain why we can effectively capture the features in our data with binary weights and activations. Our main result is that the neural networks with binary weights and activations trained using the Courbariaux, Hubara et al. (2016) method work because of the high-dimensional geometry of binary vectors. In particular, the continuous vectors that extract out features in these BNNs are well-approximated by binary vectors in the sense that dot products are approximately preserved. Compared to previous research that demonstrated the viability of such BNNs, our work explains why these BNNs work in terms of the geometry of high-dimensional binary vectors. Our theory serves as a foundation for understanding not only BNNs but networks that make use of low precision weights and activations. Furthermore, a better understanding of multilayer binary neural networks serves as a starting point for generalizing BNNs to other neural network architectures such as recurrent neural networks.

1 INTRODUCTION

While neural networks have traditionally used relatively costly 32-bit floating point multiplications, recent work by Courbariaux et al. (2016) and Hubara et al. (2016) has shown that one can effectively train neural networks that have binary weights and activations that have similar performance to their continuous counterparts. There are two key ideas in this paper summarized in Fig. 1. First, associated with each binary weight, w^b, is a continuous weight, w^c that allows us to accumulate gradients. Second, we replace the non-differentiable binarize function ($bin(x) = 1$ if $x > 0$ and -1 otherwise) with a continuous one for the backward pass. This work is closely related to Rastegari et al. (2016) which also gives an algorithm for training BNNs and an extensive review of previous research. While recent work has shown how to train such networks, the existence of neural networks with binary weights and activations needs to be reconciled with previous work that has sought to understand weight matrices as extracting out features in the data (e.g. Zeiler & Fergus (2014)). Summary of contributions:

1. To build our intuition, we show that binarizing a high-dimensional vector with entries originally chosen uniformly in the interval $[-1, 1]$ changes the direction of that vector a small amount relative to the angle between two random vectors chosen from the same distribution. Thus binary vectors form a highly expressive set of feature extractors, contrary to intuition based on low dimensional geometry.
2. We show empirically that the binarization of such vectors approximately preserves angles in the neural networks trained using the Courbariaux et al. (2016) method.

[*]AGA is a Ph.D. Candidate in the U.C. Berkeley Physics Department

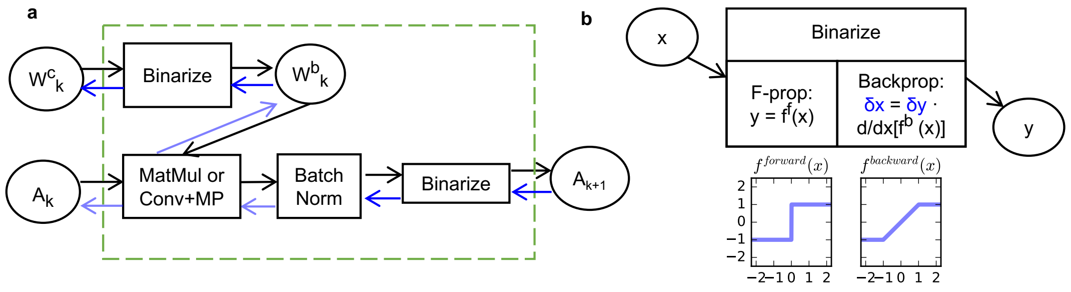

Figure 1: Review of the Courbariaux et al. (2016) BNN Training Algorithm a. A binary neural network is composed of this binary convolution transformer (green). Ovals correspond to a tensor and the derivative of the cost with respect to that tensor and rectangles correspond to transformers that specify forward and backward propagation functions. Associated with each binary weight, w^b, there is a continuous weight, w^c, that is used to accumulate gradients. b. Each binarize transformer has a forward function and a backward function. The forward function simply binarizes the inputs. In the backward propagation step, we compute the derivative of the cost with respect to the input in terms of the Jacobian of the forward function and the derivative of the cost with respect to the output ($\delta u \equiv dC/du$ where C is the cost function used to train the network). Since the binarize function is non-differentiable, the backward function is modified.

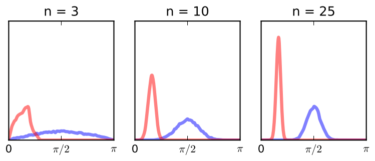

Figure 2: Binarization of Random Vectors Approximately Preserves their Direction: In high dimensions, the red and blue curves have almost no overlap. The blue curves show the angle between two random vectors of dimension n whose entries are chosen uniformly from the interval $[-1, 1]$ (we choose this distribution because the learning algorithm clips the weights w^c to be in this interval). The red curves show the angle between such a random vector and the binarized version of that vector. We see that for low dimensions, the angle between a vector and its binarized version has a high probability of being similar to a pair of two random vectors. However, when we get to a moderately high dimension, we see that the red and blue curves are well separated.

2 Theory

In the hyperdimensional computing theory of Kanerva (2009), the key idea is that two random, high-dimensional vectors whose entries are in the set $\{-1, 1\}$ are approximately orthogonal. This is a consequence of the central limit theorem, where the cosine angle between two random vectors is normally distributed with mean zero and standard deviation proportional to $n^{-0.5}$. Taking the inverse cosine of these dot-products gives us angles that are close to $\frac{\pi}{2}$. We apply this approach of analyzing the geometry of high-dimensional vectors to binary vectors. When we are in a moderately high dimension, binarizing a vector changes its direction by a small amount relative to the angle between two random vectors (Fig. 2).

This theory leads to a concrete prediction for our neural networks. Suppose that in our data or some intermediate representation of the input that there is an informative direction, w. The weight matrices are trying to extract out this informative direction. We hypothesize that $w \cdot x$ is preserved where w is a weight matrix and x are the activations. In particular, we predict that: $x \cdot w^b \sim x \cdot w^c$ where $w^b = bin(w^c)$ and $\sim$ denotes 'proportional to'[1] in our neural networks. It is worth noting that this isn't going to be true in general, but is a consequence of the conditions set up by the

[1]This proportionality constant is normalized out because the dot products are followed by a batch normalization layer as in Fig. 1.

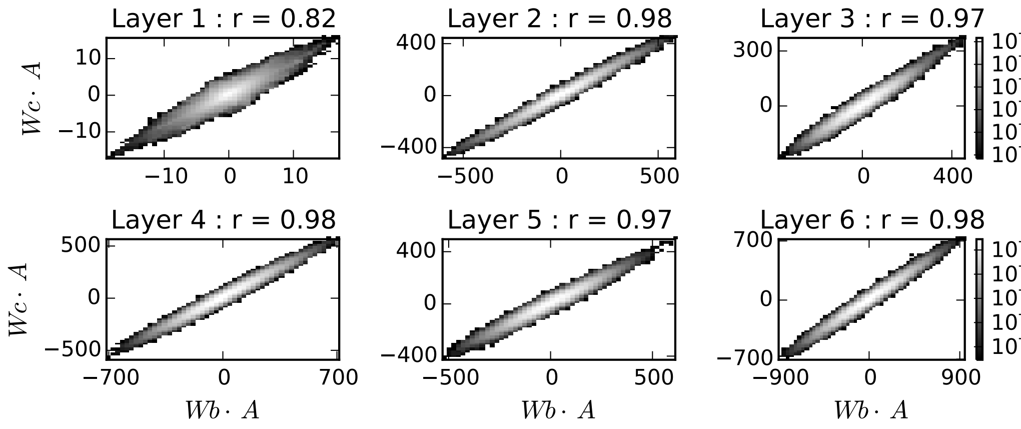

Figure 3: Binarization Preserves Dot Products: In this figure, we verify our hypothesis that binarization approximately preserves the direction of a vector in high dimension. We train a convolutional neural network on CIFAR-10. Each figure shows a 2d histogram of the binarized weights times the activations (horizontal axis) and the continuous weights times the activations (vertical axis). We see that these dot products are highly correlated, verifying our theory. Note they may differ up to a scaling constant due to the subsequent batch norm layer. The first figure (labeled as Layer 1) corresponds to the input and the first weight matrix. Note that the correlation is weaker in lower dimensions, which is consistent with our theory.

learning algorithm. See 5.1 for a more detailed argument. While we exculde this figure due to space constraints, we also verify that a similar relationship holds for the binarization of the activations E.g. $w^b \cdot x^c \sim w^b \cdot x^b$ where x^c denotes the pre-binarized (post-batch norm) activations.

3 Experiments

In this section, we verify our predictions experimentally. We train a binary neural network on CIFAR-10 (same as in Courbariaux et al. (2016)). This convolutional neural network has 3 by 3 convolutional filters with numbers of channels: 128, 128, MP, 256, 256, MP, 512, 512 MP (MP is max pool). Then two fully connected layers with 1024 units each. Each layer has a batch norm layer in between. At each layer, the dimension of the weight vector in consideration is the patch size ($= 3 * 3 = 9$) times the number of channels. We see that the dot products pre and post binarization are highly correlated (Fig. 3). We also get similar results when training a MLP on MNIST.

4 Conclusion

We provide a theory and experimental verification for understanding how we can represent features in binary neural networks. In particular, we show that the informative directions in our data and in the intermediate representations formed by the neural network for classification purposes can be well-approximated by binary vectors when we are in a sufficiently high dimensional space.

Acknowledgments

The authors would like to thank Bruno Olshausen and other members from the Redwood Center for useful feedback on this work. Further, we thank the anonymous ICLR reviewers for useful feedback. This material is based upon work supported by the National Science Foundation under Grant No. DGE 1106400 (AGA). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. This work was supported in part by Systems on Nanoscale Information fabriCs (SONIC), one of the six SRC STARnet Centers, sponsored by MARCO and DARPA (AGA).

REFERENCES

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

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *arXiv preprint arXiv:1609.07061*, 2016.

Pentti Kanerva. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive Computation*, 1(2):139–159, 2009.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. *arXiv preprint arXiv:1603.05279*, 2016.

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pp. 818–833. Springer, 2014.

5 SUPPLEMENTARY INFORMATION

5.1 LEARNING DYNAMICS

Here, we discuss in more detail the conditions that the learning algorithm sets up that gives rise to the preservation of dot products (up to a constant). From a theoretical perspective, the key idea is as follows. Suppose we have a neural network where there is some transformer $v = f(u)$ (u, v are vectors). On the forward pass, we use f, but on the backwards pass, we replace f by g. How does this impact the learning dynamics? In our case, u is the continuous weight, w^c, $f(u)$ is the pointwise binarize function and $g(u)$ is the identity function as in part (b) of Fig. 1. In particular, suppose that the loss function as a function of the output of f is $L(v)$. Then our updates are $-\delta u_i \sim \sum_j \partial_j L(f(u)) * \partial_i g_j(u)$ where $\partial_j L(v)$ denotes the partial derivative of L with respect to the jth component of v and $\partial_i g_j(u)$ denotes the partial derivative of the jth component of $g(u)$ with respect to u_i. Note that the typical gradient would be $\sum_j \partial_j L(f(u)) * \partial_i f_j(u)$. In our particular situation, g is the identity function, so the Jacobian of g is the identity matrix. E.g. $\partial_i g_j(u) = \delta_{i,j}$ So our learning dynamics are just $-\delta u_i \sim \partial_i L(f(u))$. The steady state of these equations is when $\partial_j L(v) = 0$ for all j.

Suppose that $v^* \in R^n$ is a minima of $L(v)$. Then the dynamics are stable when there exists a u^* such that $f(u^*) \approx v^*$. This is where the high dimensionality of the problem comes in. The high dimensionality makes it relatively easy to find an approximate solution to this equation (i.e. we can approximate the direction of a continuous HD vector, v^* relatively well with a binary vector $f(u^*)$). We also note that the optimization only needs to estimate the direction of v^* because $L(v) = L(kv)$ where k is a constant and v is a vector. This relation is true because we use batch norm without subtracting the mean and then binarize the output of the dot product of the weight vector with the activations.

Now where do the dot products come in? We can think of the above equations as $-\delta u_i = \partial_i L(u+\epsilon)$ where ϵ is noise that models the fact that $f(u)$ corrupts u. In this case, we can see that these learning dynamics seek to send u to the minimum of $L(v)$. Thus $f(u)$ and $g(u) = u$ seek to approximate the informative direction, v^*, which give us the approximate dot product relation.

5.2 COMPARISON TO BINARIZING CONTINUOUSLY TRAINED NETWORKS

Given the set-up in this paper, an important question remains: why does this theory not directly apply to binarizing pretrained networks? The first thing to note is that in our networks, the magnitude of a weight vector (and correspondingly the dot products) does not matter because it is divided out by a batch norm layer and then binarized. So when we train such a BNN, the algorithm is forced to find a solution where the magnitude of the weight vectors are not used. This is in contrast to a typical neural network where the network will be trained with the possibility of the magnitude of the dot products playing an important role. The approach of approximating a continuous weight matrix with a binary matrix and a scale factor is described in Rastegari et al. (2016).