White Paper

Improving distributed mesh computing with Hadamard Binary Neural Networks

Yash Akhauri
Birla Institute of Technology and Science
akhauri.yash@gmail.com

September 2, 2018

Deep neural networks are an important tool in modern applications. It has become a major challenge to accelerate their training. As the complexity of our training tasks increase, the computation does too. For sustainable machine learning at scale, we need distributed systems that can leverage the available hardware effectively. This research hopes to exceed the current state of the art performance of neural networks by introducing a new architecture optimized for distributability. The scope of this work is not just limited to optimizing neural network training for large servers, but also to bringing training to heterogeneous environments; paving way for a distributed peer to peer mesh computing platform that can harness the wasted resources of idle computers in a workplace for AI.

Contents

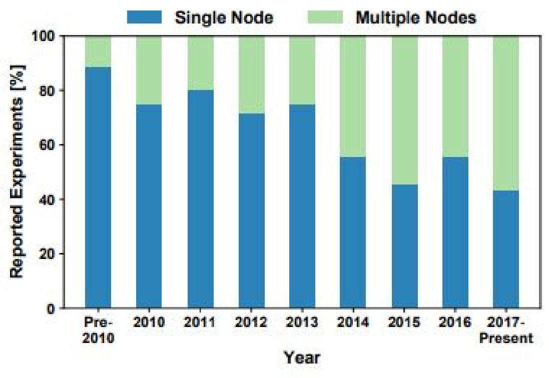
1 Background and Significance

This whitepaper will firstly highlight the shortcomings of current solutions in distributed training of neural networks. We will then delve into binary neural networks and its limitations. In the research section, I shall introduce *Hadamard Binary Neural Network*, how it deals with the problems at hand, and the resources that may be needed.

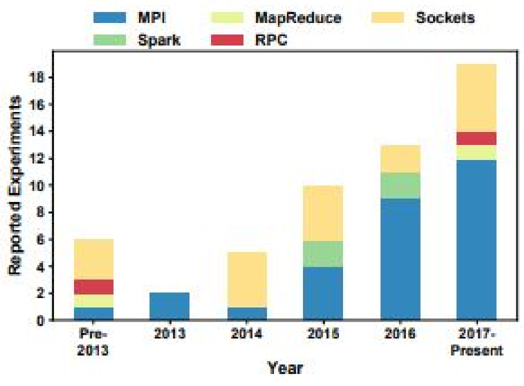
1.1 Contributions:

1) Introduce a new architecture (*HBNNs*) that increases the performance of binary neural networks significantly with ~10x lesser memory requirements than conventional methods.
2) Study the high dimensional geometry of HBNNs.
3) Solve the scalability deadlock of InnerProduct.
4) Use HBNNs as regularizers instead of dropout, improving *scalability* significantly.
5) Reduce the *communication bottleneck* in distributed training tasks ~30 times.
6) Develop a distributed training platform that can scale in heterogeneous compute environments.

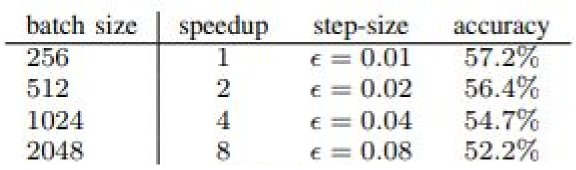
As indicated by Fig. 1(a), we are shifting to a distributed methodology for training neural networks. We owe that to not only the increasing complexity of problems but also the benefits reaped by harnessing multiple CPUs/GPUs for training. The most common approach to distributed machine learning -- Data parallelized stochastic gradient descent is a vastly communication bound problem.[2]



(a)



(b)



batch size	speedup	step-size	accuracy
256	1	$\epsilon = 0.01$	57.2%
512	2	$\epsilon = 0.02$	56.4%
1024	4	$\epsilon = 0.04$	54.7%
2048	8	$\epsilon = 0.08$	52.2%

(c)

Larger batch sizes can not compensate for the loss in accuracy[2].

Fig. 1.

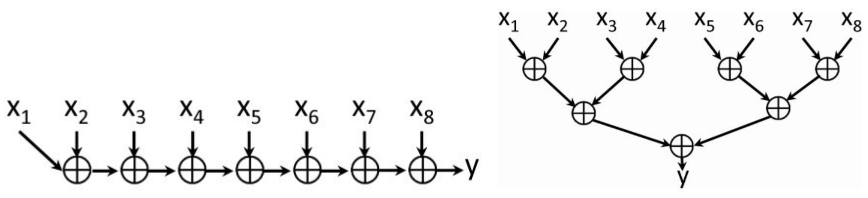
Due to the sequential nature of the training algorithm (1), we can speed up training in two ways -- Compute updates faster or use larger steps.

$$\omega_{t+1} \leftarrow \omega_t - \varepsilon \delta_j a_i \quad (1)$$

While we continually push the boundaries of how fast we can do inference, we are limited by topologies of non-convex problems with regards to our step sizes. Inappropriate step sizes can cause the *optimizer to diverge*.

In data parallel models, the global batch size (**S**) is split among **n** worker nodes with size **s** (**s** = **S**/**n**). The workers can use several strategies to update the weights of the parameter server → Bulk synchronous parallel, asynchronous parallel, stale synchronous parallel etc. In a heterogeneous environment, synchronous updates can cripple the training algorithm massively, whereas an asynchronous update methodology leads to significantly slower convergence[3] due to *stale gradients*.

Aside from the limitations of data parallel models, the communication bottleneck must also be taken into consideration. The accumulated gradients to update the model must also be shared to the parameter server. The reduction tree method (Fig. 2(b).) is a commonly used methodology, however, typical ImageNet training problem became communication bound after only *4-8 nodes.* [2]



(a) Linear-depth reduction (b) Reduction tree

Fig. 2.

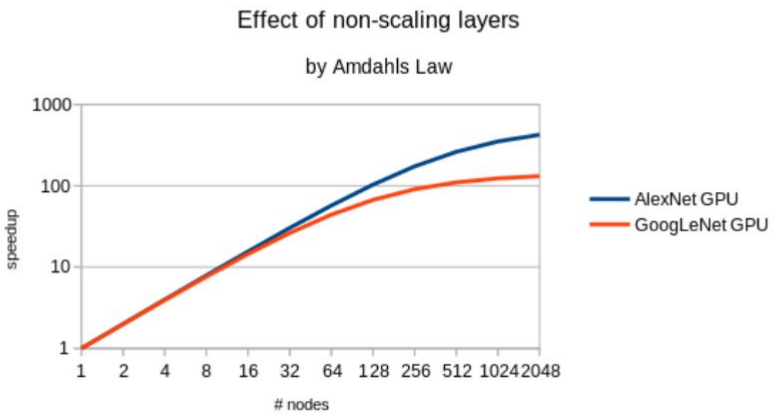
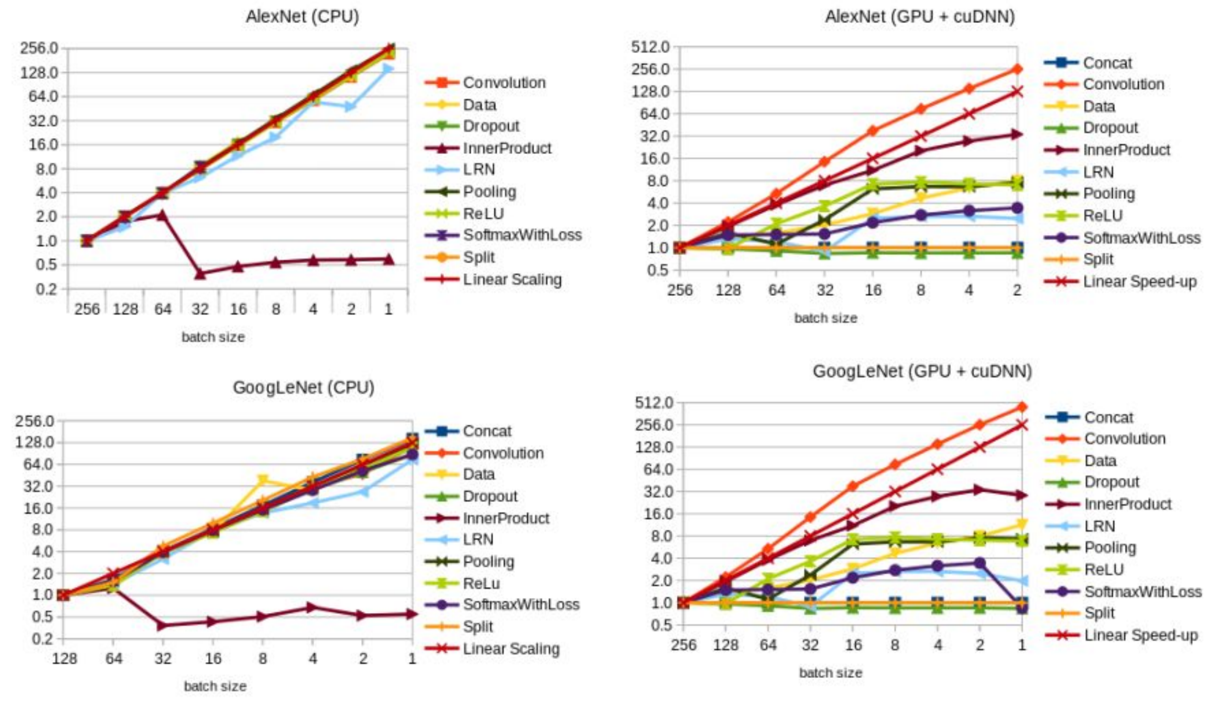


Fig. 3.

We now come to the scaling of certain modules of a neural network. An observation made by [2] stresses on how the portion of the compute time spent on *InnerProduct* (Fully-Connected) layer increases sizably on both CPUs and GPUs. The InnerProduct layer scales poorly for batch sizes smaller than 64. This problem is very grave, when we look at Fig. 4. While **CPUs scale most layers remarkably** in both directions, we see a general problem with the InnerProduct.
From Fig. 4. It is also evident that there are some 'non-scaling' layers. Layers such as Dropout, pooling etc. As nodes increase, we see the overhead generated by these layers too.

The remarkable advantage of CPUs here is that they scale such ‘non-scaling’ layers exceedingly well too.

To sum it up, the problems with current approaches to distributed training of models range from the computational complexity of neural networks, to scalability of layers and communication bottlenecks in networks.



Layer by layer analysis of speedup achieved by reducing the batch size.
Fig. 4.

1.2 Binary neural networks

Binary weights are weights that are constrained to only two possible values (-1/+1). Constraining a neural network to have full precision scalars tied to a weight matrix which only consists of +1 or -1s enables us to *replace multiply-accumulate operations with bit-manipulations*. The *BinaryConnect* implementation of neural networks consists of training a DNN with binary weights during the forward and backward propagations, while retaining precision of the stored weights in which gradients are accumulated. Fig.5. details the binarization scheme of such a neural network.

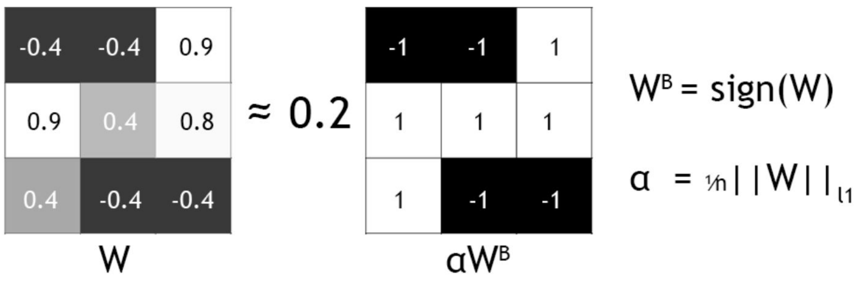
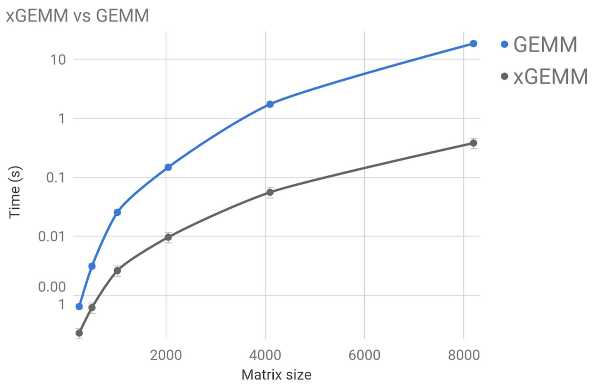
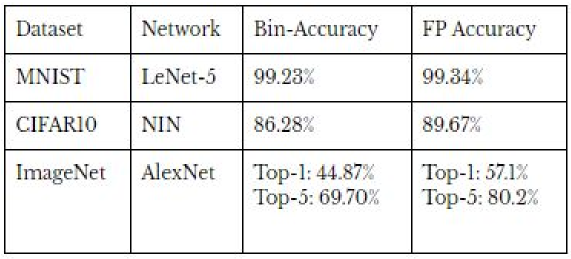


Fig. 5.

Conducting benchmarks (Fig. 6(a)) on a home-made xGEMM kernel (using OpenMP), which is the binarized scheme equivalent to GEMM, shows significant improvement in inference speeds. However, we see a noticeable drop in the accuracy (Fig. 6(b)) of such a binarization approach on several datasets and architectures.



(a)



Dataset	Network	Bin-Accuracy	FP Accuracy
MNIST	LeNet-5	99.23%	99.34%
CIFAR10	NIN	86.28%	89.67%
ImageNet	AlexNet	Top-1: 44.87% Top-5: 69.70%	Top-1: 57.1% Top-5: 80.2%

(b)

Fig. 6.

But why do binarized neural networks work at all?

This is a very valid question, the answer lies in the high dimensional geometry of BNNs (Binary neural networks).

We will delve into two properties of BNNs that give an intuition of why this works. (Mostly rephrasing [4])

1. **Angle preservation property.**

Binarization approximately preserves the *direction of high dimensional vectors*[4]. The study [4] demonstrates that the angle between a random vector (from a standard normal distribution) and its binarized version converges to ~ *37 degrees* as the dimension of the vector goes to infinity. This angle is exceedingly small in high dimensions. Fig. 7a shows the distribution of angles between two random vectors (blue) and between a vector and its binarized version (red). Fig. 7b shows the angle distribution between continuous and binary weight vectors by layer for a binary CNN trained on CIFAR-10.

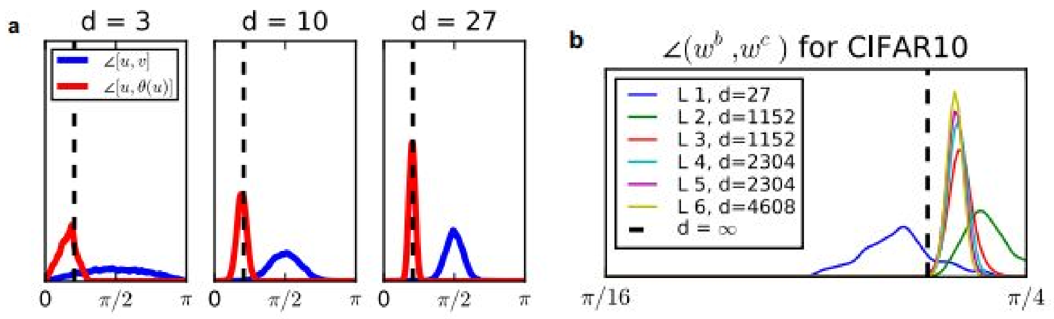


Fig. 7

2. **Dot product preservation**

Research [4] in Fig. 8. verifies the hypothesis that binarization approximately preserves the dot-products that the network uses for computation. Fig. 8. shows a 2D histogram of dot products between the binarized weights and activations (x-axis), and dot product between the continuous weights and the activations (y-axis). *r* is the Pearson correlation coefficient. We see however that this isn't true for the first layer. Thus it is wise to keep the first layer full precision.

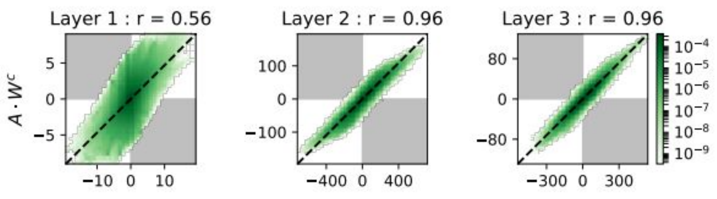
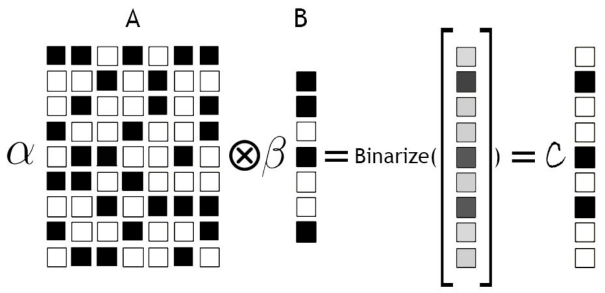


Fig. 8.

2 Research

Now that we have introduced the limitations of current distributed training methodologies and developed insight into the binarization scheme for a BNN, we can delve into the new proposed architecture and its requirement.
The XNOR-Net architecture (Binary neural network studied above (Fig. 5.)) takes the average of an *entire* weight matrix and ties it to a scalar full precision value. While this vanilla binarization constraint might work for a simple MNIST MLP, such an approach might leave the network heavily under-parameterized.
To increase the capacity of the network, we can increase the precision of the weights. However, with this we lose out on the advantage of using *bit-manipulations for forward passes*. We must thus think of a new way to increase the degree of freedom of such a weight matrix.
For this purpose, I propose a new methodology of binarizing the network, which allows us to increase the capacity of the network while maintaining the bit-manipulation advantage.

Let us first look at how the original BNN works in Fig. 9. Note that A and B are binary matrices. **f** is a function that depends upon the dimensions of the matrix (form is f(x) = ax + b).



$\otimes$ = f(pcnt(xnor(BP(A[i, :]), BP(B[:]))))

Fig. 9.

2.1 The model (Hadamard Binary Neural Networks (HBNNs)

$\widetilde{W}$ = Binarized weights	W = Full precision weights	α_r = FP matrix tied to $\widetilde{W}$
C = Cost function	r = Number of rows	c = Number of columns

In this model, we do row wise binarization, where the row-wise mean is stored in α_r. It is *extremely* important to note that α_r is *broadcasted to the same dimensions* as the weight matrix. This allows us to have many degrees of freedom with very little computation. We can imagine this as Fig. 10. This is a **Hadamard product**, which is differentiable. We can segment the row binarization scheme as well, based on the data type we bit-pack into for efficiency.

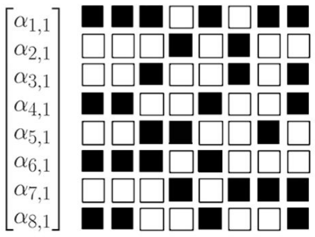


Fig. 10.

$$W = \begin{bmatrix} W_{1,1}....W_{1,8} \\ ...... \\ ...... \\ W_{r,1}....W_{r,c} \end{bmatrix} \qquad \widetilde{W} = \begin{bmatrix} \widetilde{W}_{1,1}....\widetilde{W}_{1,8} \\ ...... \\ ...... \\ \widetilde{W}_{r,1}....\widetilde{W}_{r,c} \end{bmatrix}$$

$$\widetilde{W} = \alpha_r \odot sign(W) \qquad \alpha_r = (1/c) * (||W_{r,:}||_{l1})$$

$$\widetilde{W}_{r,c} = \alpha_r \odot sign(W_{r,c})$$

$\odot \rightarrow$ Hadamard product.

2.2 Backpropagation in model

$$\frac{\partial C}{\partial W_{i,j}} = \sum_{k=1}^{r}\sum_{l=1}^{c}\left(\frac{\partial C}{\partial \widetilde{W}_{k,l}}\frac{\partial \widetilde{W}_{k,l}}{\partial W_{i,j}}\right)$$

$$\frac{\partial C}{\partial W_{i,j}} = \sum_{k=1}^{r}\sum_{l=1}^{c}\left(\frac{\partial C}{\partial \widetilde{W}_{k,l}}\frac{\partial(\alpha_k \odot sign(W_{k,l}))}{\partial W_{i,j}}\right)$$

$$\frac{\partial C}{\partial W_{i,j}} = \sum_{k=1}^{r}\sum_{l=1}^{c}\left(\frac{\partial C}{\partial \widetilde{W}_{k,l}}\frac{\partial(\alpha_k)}{\partial W_{i,j}} \odot sign(W_{k,l})\right) + \sum_{k=1}^{r}\sum_{l=1}^{c}\left(\frac{\partial C}{\partial \widetilde{W}_{k,l}}\frac{\partial(sign(W_{k,l}))}{\partial W_{i,j}} \odot \alpha_k\right)$$

$$\frac{\partial C}{\partial W_{i,j}} = \sum_{k=1}^{r}\sum_{l=1}^{c}\left(\frac{\partial C}{\partial \widetilde{W}_{k,l}}\frac{\partial(\alpha_k)}{\partial W_{i,j}} \odot sign(W_{k,l})\right) + \left(\frac{\partial C}{\partial \widetilde{W}_{i,j}}\frac{\partial(sign(W_{i,j}))}{\partial W_{i,j}} \odot \alpha_i\right)$$

As we know,

$$\alpha_k = \frac{||W_{k,:}||_{l1}}{c} \qquad \frac{\partial \alpha_k}{\partial W_{i,j}} = \frac{sign(W_{i,j})}{c}$$

Thus,

$$\frac{\partial C}{\partial W_{i,j}} = \sum_{k=1}^{r}\sum_{l=1}^{c}\left(\frac{\partial C}{\partial \widetilde{W}_{k,l}}\frac{sign(W_{i,j})}{c}\odot sign(W_{k,l})\right)+\left(\frac{\partial C}{\partial \widetilde{W}_{i,j}}\frac{\partial(sign(W_{i,j}))}{\partial W_{i,j}}\odot \alpha_i\right)$$

Note that this is incomplete, in the sense that only 1D α_r arrays have been accounted for. However, this *easily generalizes to higher dimensions for* α_r.

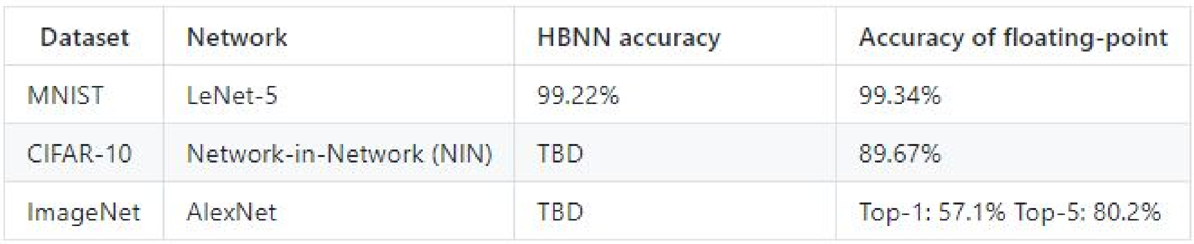
Here, we can say that the *aggressiveness of binarization* is decided by the scheme detailed below:

$$\alpha_{s,z} = \frac{1}{c}||W_{k,(\lfloor\frac{c}{n}\rfloor*z):(\lfloor\frac{c}{n}\rfloor*(z+1))}||_{l1} \qquad where\ z\ varies\ \in (0\ ,\lfloor\frac{c}{n}\rfloor)$$

As we increase n, the aggressiveness increases. Typically, n will be decided by available data types and their bit-widths. (n → 4-bit, 8-bit, 16-bit, 32-bit, 64-bit and so on.)

Now that we have developed an intuition for the model and its dynamics, let us discuss the advantages that it offers us.
Angle preservation property: In the previous models, the binarization scheme led to a deviation of 37 degrees between a random normal initialized vector and its binarized version. However, we may be able to reduce that difference using the HBNNs by giving the model more freedom.
The same applies to the *preservation of dot product*. A network will have to be trained with this constraint to truly investigate its characteristics.
A strategy that could be effective is to progressively increase the *aggressiveness of binarization* as the depth of the network increases. This could give massive speed-up compared to current solutions, but at the same time have good accuracy. This hypothesis is directly supported by the fact that binarizing higher level features don't make a significant difference. (Fig. 8.)

2.3 Accuracy benchmarks on standard datasets



Dataset	Network	HBNN accuracy	Accuracy of floating-point
MNIST	LeNet-5	99.22%	99.34%
CIFAR-10	Network-in-Network (NIN)	TBD	89.67%
ImageNet	AlexNet	TBD	Top-1: 57.1% Top-5: 80.2%

Fig. 11.

Implementation and training logs can be found here.

In the above experiments, the binAgg hyperparameter was taken to be 16. This has an angle preservation property where the angle between the randomly initialized matrix and the Hadamard binarized matrix converges to ~ 33 degrees.

2.4 xHBNN benchmark

Generating minimally optimized CMMA and HBNN OpenMP kernels in C and comparing it to the state of the art Intel® Math Kernel Library (Intel® MKL) demonstrates that there is a lot of potential in the xHBNN algorithm for speed up in inference and training.
X Axis: Matrix Size
Y Axis: Time (seconds)

CMMA vs xHBNN

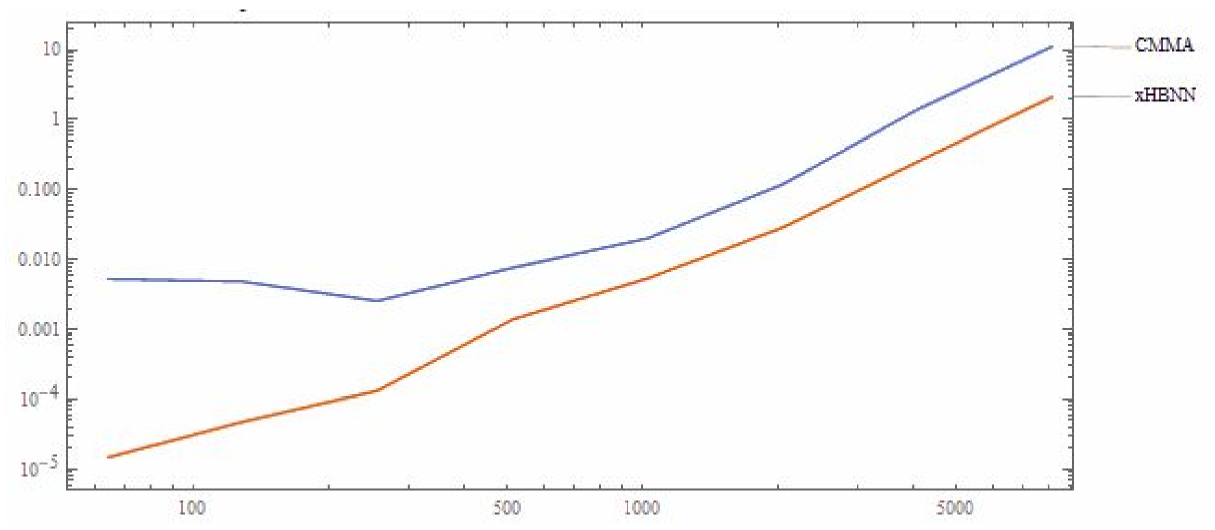


Fig. 12.

MKL vs xHBNN

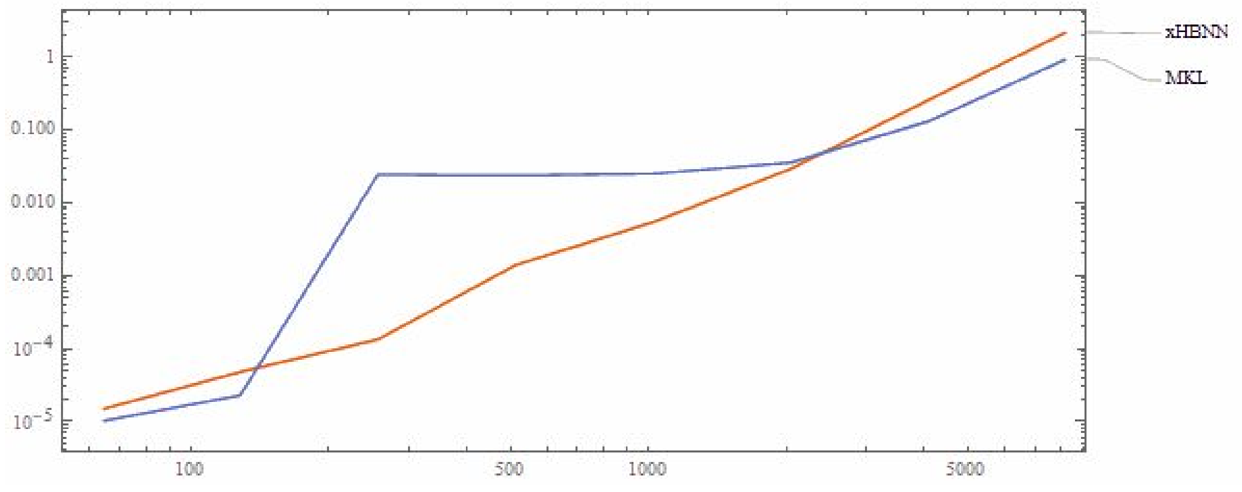


Fig. 13.

The current stock processor configuration for the Intel® AI DevCloud, used for training and testing this architecture, features the Intel® Xeon® Gold 6128 processor clocked at 3.40 GHz, a 19.25 MB cache, 24 cores with 2-way hyper-threading.

2.5 Angle preservation property

Binarization approximately preserves the direction of high dimensional vectors. The figure above demonstrates that the angle between a random vector (from a standard normal distribution) and its binarized version converges to ~ 37 degrees as the dimension of the vector goes to infinity. This angle is exceedingly small in high dimensions.

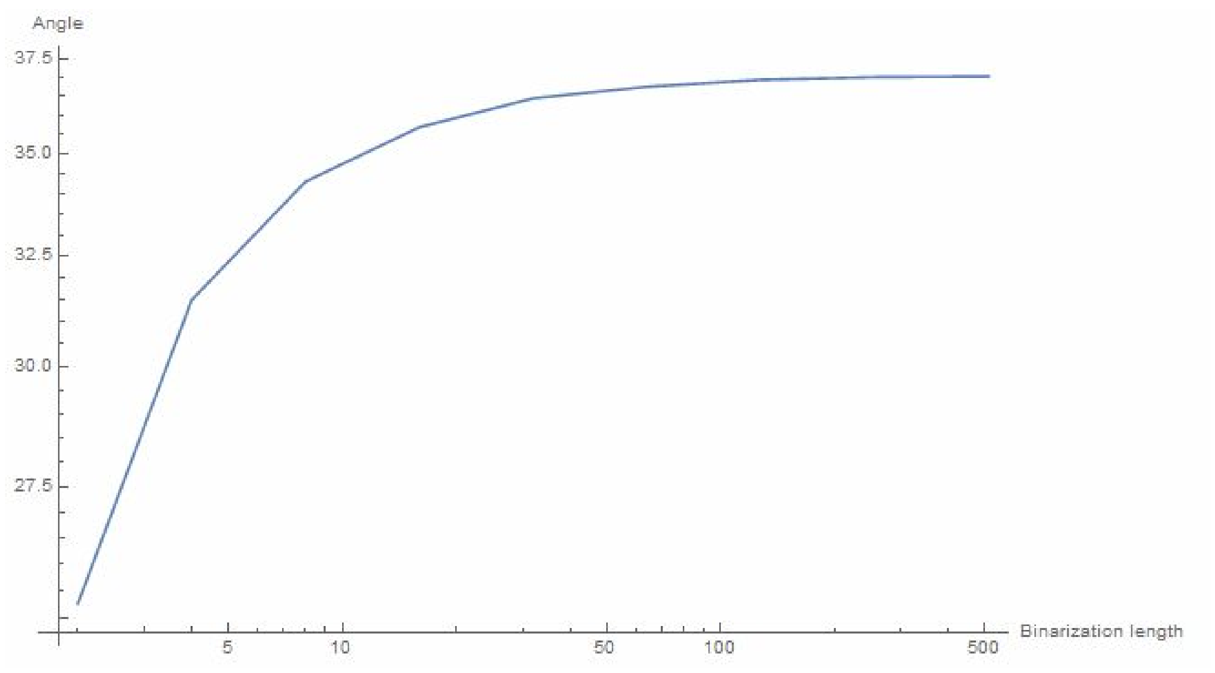


Fig. 14.

2.6 Weight histograms

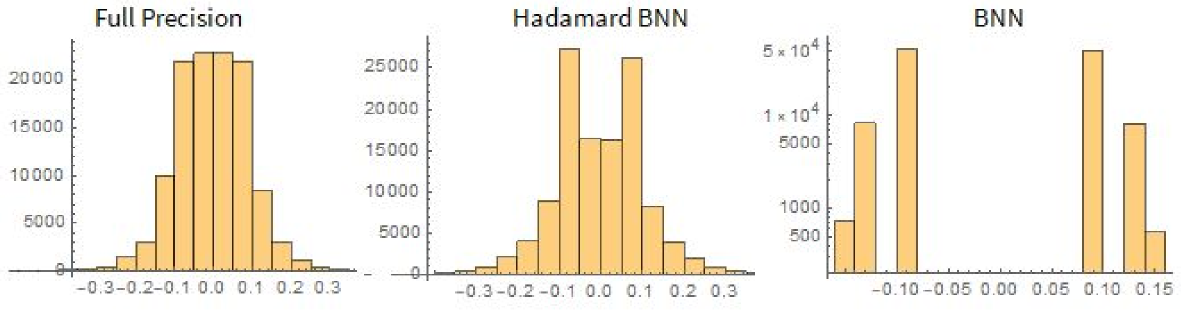


Fig. 15.

It is evident that the Hadamard BNN preserves the distribution of the weights much better. Note that the BNN graph has a logarithmic vertical axis, for representation purposes.
While this does not stress upon any inherent property of Hadamard BNNs specifically, it does demonstrate that the granularity of the weights does increase notable when compared to the sparse BNN histogram distribution.

3 Conclusion

3.1 Communication costs

We have successfully managed to theoretically increase the capacity of the network while roughly maintaining the massive speed up offered by binarization of networks. Gauging the performance of this network in terms of accuracy remains.
Let us return to the key issues of current distributed training methodologies and utilize this algorithm to streamline the process.
We know from the background of this paper that inherently, there is a communication bottleneck in current distributed systems for training neural networks. We owe this to hardware limitations, but also one belonging to the nature of neural networks.
Massive sizes of neural networks and *sequential training algorithm* requires us to communicate with a parameter server for every update. The distribution load of AlexNet, assuming 100 epochs till convergence, results in **450000*250*2(n-1) MB** of traffic in gradient and update communications. (Assuming 450k iterations and n workers.) [2]. This would place significant stress on the network.
The overhead generated by data transfer is inherent in training every network and cannot be dealt with architecture here. Moreover, as [2] suggests, the distribution load of training data can be neglectable, e.g. for AlexNet we have 100 epochs till convergence, resulting in **100 * 150GB** = **15TB** of data, which pales in comparison to the **900 TB** of gradient updates generated for 4 workers and 1 parameter server.
With our current algorithm, and a hybrid HBNN (with some full precision layers) we can expect anywhere from **10-30x** lesser traffic generated in updates. This is evidenced by the compression of a model update size upon binarization of a full precision neural network in Fig. 16.

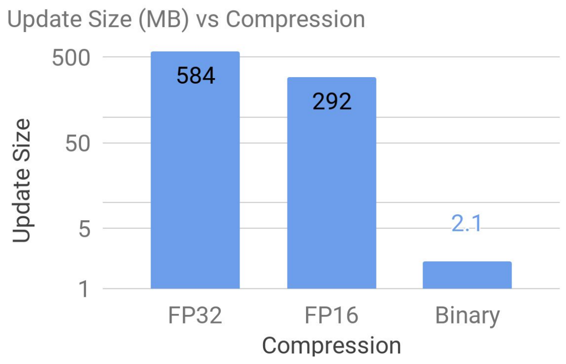


Fig. 16.

3.2 The InnerProduct

As detailed in the background, we know that compute time for InnerProduct (Fully connected layers) increases as batch size decreases. Batch sizes for a large distributed system will be very less (for a global batch size of 1024 distributed over 32 nodes, we have a batch size of 32).

We cannot accommodate this deadlock in scalability. However, scaling fully connected layers is hard, this deadlock is due to “degenerated” non square matrices being produced, which sGEMM cannot handle well.
There seems to be no *principal* solution to this issue. We see from Fig. 6a. that the xGEMM operation outperforms a home-made GEMM kernel optimized similarly significantly. Thus I propose a *non-principal* solution, using Hadamard BNNs could scale exceedingly well to a suitable number of nodes.

3.3 Non-scaling layers

We know of several *non-scaling layers* like dropout. These require very little computation by itself, but when scaling to over 1000 nodes (Fig. 3.), we see a performance degradation.

4 References

[1] Tal Ben-Nun and T. Hoefler (2018). Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency. *CoRR, abs/1802.09941*, .

[2] Janis Keuper and Franz-Josef Preundt. 2016. Distributed training of deep neural networks: theoretical and practical limits of parallel scalability. In Proceedings of the Workshop on Machine Learning in High Performance Computing Environments (MLHPC '16). IEEE Press, Piscataway, NJ, USA, 19-26. DOI: https://doi.org/10.1109/MLHPC.2016.6

[3] Jiawei Jiang, Bin Cui, Ce Zhang, and Lele Yu. 2017. Heterogeneity-aware Distributed Parameter Servers. In Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD '17). ACM, New York, NY, USA, 463-478. DOI: https://doi.org/10.1145/3035918.3035933

[4] Alexander G. Anderson and (2017). The High-Dimensional Geometry of Binary Neural Networks. *CoRR, abs/1705.07199,* .