Look-ups are not (yet) all you need for deep learning inference

Calvin McCarter[*]
calmcc@amazon.com

Nicholas Dronen[*]
ndronen@amazon.com

Abstract

Fast approximations to matrix multiplication have the potential to dramatically reduce the cost of neural network inference. Recent work on approximate matrix multiplication proposed to replace costly multiplications with table-lookups by fitting a fast hash function from training data. In this work, we propose improvements to this previous work, targeted to the deep learning inference setting, where one has access to both training data and fixed (already learned) model weight matrices. We further propose a fine-tuning procedure for accelerating entire neural networks while minimizing loss in accuracy. Finally, we analyze the proposed method on a simple image classification task. While we show improvements to prior work, overall classification accuracy remains substantially diminished compared to exact matrix multiplication. Our work, despite this negative result, points the way towards future efforts to accelerate inner products with fast nonlinear hashing methods.

1 Introduction

To reduce the computational cost of dense matrix multiplications (or, more generally, inner products) in neural network (NN) inference, recent research efforts have sought to develop cheap approximations with minimal sacrifice of classification accuracy. Methods based on distillation [1, 2] and pruning [3, 4] have proposed to reduce the number of stored parameters and the number of multiply-accumulate (MAC) operations, while methods based on scalar quantization [5] have proposed to reduce the size of stored parameters and the cost of each MAC. A recent method, MADDNESS [6], instead proposed to approximate MACs by replacing them entirely with lookup-accumulate (LAC) operations. This reduced computational cost by involving fewer LACs (and less memory) than required by MACs, and by utilizing hardware multiplexers, which require fewer transistors than hardware multipliers.

MADDNESS breaks down a matrix multiply $\boldsymbol{AB}$ (where $\boldsymbol{A}$ contains inputs / activations and $\boldsymbol{B}$ contains weights) into an assemblage of $\boldsymbol{a}^\top \boldsymbol{b}$ inner products, and approximates each of them by building on product quantization (PQ) [7, 8]. PQ enables fast inner products of the form $\boldsymbol{a}^\top \boldsymbol{b}$ when a training set of sampled $\boldsymbol{a}$s is given in advance. For paired vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ of dimension D, the PQ algorithm partitions their dimensions to form $C \leq D$ pairs of subvectors (and subspaces). The inner product is then computed as the sum of approximate partial inner products over subspaces. To do this, the K-Means algorithm is run over each subspace of the training set of $\boldsymbol{a}s$ to obtain C subspace-specific sets of prototypes. Since all $\boldsymbol{b}$s (corresponding to columns of NN weight matrices) are also known in advance, all possible inner products between subspace prototypes and $\boldsymbol{b}$ subvectors are precomputed offline and stored in C lookup tables. (Each subspace-specific lookup table is referred to as a codebook, so C is the number of codebooks.) To compute (approximate) inner products for a newly-seen $\boldsymbol{a}$ at inference time, PQ identifies, for each subspace, the nearest subspace prototype (represented by its index) for each subvector of $\boldsymbol{a}$. This mapping from subvectors to categorical variables, represented as $\log_2(K)$-bit integer indices where K is the number of prototypes

[*]Work done prior to employment at Amazon

Preprint. Under review.

per subspace, is called the *encoding function*. Then, the computed inner product is simply the sum of partial inner products obtained from indexing into the appropriate rows in the precomputed lookup tables.

The primary contribution of MADDNESS addressed the computational bottleneck of PQ, with a fast new encoding function. MADDNESS proposed a new hashing-based encoder, based on balanced binary regression trees, that runs efficiently on modern processors when $K = 16$; this encoder was learned from the training set of $\boldsymbol{a}s$ to minimize reconstruction error. Also, rather than choosing prototypes via K-Means, MADDNESS optimized a least-squares objective, which also minimized reconstruction error; this objective conditioned on the training set of $\boldsymbol{a}$s, as well as the (already learned) encoder's resulting indices from the training samples. Finally, lookup tables were quantized to 8-bit block floating point format, with fast rounded 8-bit summation over subspaces.

While proposed to accelerate entire NNs, MADDNESS was validated for a large variety of models and settings but only on the final classification layers. In each of these settings, the authors varied the number of codebooks, the key tuning parameter which trades off accuracy versus efficiency. For a NN feedforward layer with M output dimensions and FLOAT32 weights, the weight matrix requires $4DM$ bytes of storage. With MADDNESS, the lookup table requires $KCM = 16CM$ bytes of storage.

In this work, we propose a new method which builds on MADDNESS, and analyze its applicability to whole NN inference. We dub our proposed method, Inference Targeted LookUp-based Matrix Multiplication (ITLUMM) [2]. First, we intelligently partition the inner dimension rather than naively partitioning it into contiguous subspaces. Second, we directly optimize the lookup table rather than prototypes, taking into account the subsequent nonlinear activation function (or classification loss function). Furthermore, for accelerating entire neural networks, we propose incremental fine-tuning of linear layers.

We apply our approach to feedforward-only image classification on MNIST [9] and CIFAR-10 [10]. We found that, while ITLUMM improves upon MADDNESS, the accuracy-efficiency tradeoff is not yet good enough for practical use. Our analysis of results suggests that an improved encoding function should be a key focus of future work.

2 Method

Here we describe ITLUMM, which comprises intelligent subspace partitioning (Section 2.1), model-aware optimization of the lookup tables (Section 2.2), and fine-tuning of full networks (Section 2.3).

2.1 Intelligent Subspace Partitioning

For each subvector, MADDNESS performs exactly 4 comparisons in order to map the subvector into one of 16 lookup-table rows. We would therefore prefer a subvector with maximal mutual information among its elements, such that this small number of comparisons provides maximal information about the entire subvector. Thus, we should partition the vector $\boldsymbol{a}$ such that mutually-informative dimensions are placed together in the same subvector. To do this, we find a permutation of the D dimensions of $\boldsymbol{a}$, then slice the permuted dimensions into C contiguous subspaces.

This problem is closely related to Optimized Product Quantization (OPQ) [11] which rotates vectors before partitioning them, but in our case our rotation matrix must be a permutation matrix. Directly optimizing for the optimal permutation is challenging, so we try two different approximate solutions for this problem.

2.1.1 OPQ-based Partitioning

We first run OPQ to obtain a fully-dense rotation matrix $\boldsymbol{R}$. Then, we find a permutation matrix $\boldsymbol{Q}$ that approximates the effect of $\boldsymbol{R}$, by solving the following maximum weight matching problem:

$$\max_{\boldsymbol{Q}} \sum_i \sum_j \boldsymbol{R}_{ij} \boldsymbol{Q}_{ij}. \tag{1}$$

[2] Spelled backward, this is MMULTI.

We solve this using the Hungarian algorithm [12]. The resulting bipartite graph matches dimensions of the original vector to their "closest matching" dimensions in the OPQ-rotated vector. Once we have a permuted ordering of the dimensions, we split the reordered dimensions into C contiguous equally-sized chunks.

2.1.2 Squared-correlation Hierarchical Clustering

Here, we instead try to directly identify a permutation of the dimensions that places correlated dimensions closely together. We first compute the squared correlation matrix R^2, then perform hierarchical/agglomerative clustering on it to produce a dendrogram of the dimensions [13]. Next, we convert the dendrogram into a permutation by finding the ordering of the leaf nodes such that the distance between successive leaves is minimal. Finally, we split the reordered dimensions into C contiguous equally-sized chunks.

2.2 Model-Aware Lookup-Table Optimization

The prototype-optimization objective function of MADDNESS sought to maximize the ability to reconstruct $\boldsymbol{A}$, given the prototypes and the encoder's output (i.e. indices into the lookup-tables). In other words, given a row vector $\boldsymbol{a}^\top$ and the corresponding encoder output $\boldsymbol{g}^\top$ (the concatenation of C one-hot 16-dimensional vectors), the prototype matrix $\boldsymbol{P}$ was optimized so that $\boldsymbol{a}^\top \approx \boldsymbol{g}^\top \boldsymbol{P}$. To do this, given a training matrix $\boldsymbol{A}$, its corresponding encoder output matrix $\boldsymbol{G}$, and the original K-means prototype matrix $\boldsymbol{P}_0$, MADDNESS optimized the following:

$$\arg\min_{\boldsymbol{P}} \|\boldsymbol{A} - \boldsymbol{G}\boldsymbol{P}\|^2 + \lambda \|\boldsymbol{P} - \boldsymbol{P}_0\|^2, \tag{2}$$

where λ is a regularizer. Finally, MADDNESS computed the lookup-table via $\boldsymbol{T} = \boldsymbol{P}\boldsymbol{B}$.

We note that this procedure ignores the fact that for deep learning inference, we also know the already-learned weight matrix $\boldsymbol{B}$. We also observe that it aims at reconstructing the layer inputs $\boldsymbol{A}$, rather than the matrix product $\boldsymbol{A}\boldsymbol{B}$. In fact, in deep learning we actually care about not the matrix product $\boldsymbol{A}\boldsymbol{B}$, but rather the layer output $\sigma(\boldsymbol{A}\boldsymbol{B})$. The function $\sigma(\cdot)$ is the composition of the bias term with a nonlinearity (either an elementwise activation function such as ReLU or a row-wise nonlinearity for classification such as softmax). Optimizing quantization for downstream performance has shown great success in previous works [14, 15].

Therefore, in ITLUMM we take advantage of our knowledge of both the model weights and the subsequent nonlinearity. Furthermore, for computationally efficiency at training time, we directly optimize the lookup table, rather than the prototypes. This leads to the following objective:

$$\arg\min_{\boldsymbol{T}} \|\sigma(\boldsymbol{A}\boldsymbol{B}) - \sigma(\boldsymbol{G}\boldsymbol{T})\|^2 + \lambda \|\boldsymbol{T} - \boldsymbol{P}_0\boldsymbol{B}\|^2. \tag{3}$$

2.3 Fine-tuning for Acceleration of Full Neural Networks

Replacing a neural network layer with approximate hashing-based lookups may degrade accuracy in two ways. Layer replacement may alter the distribution of the layer's outputs without destroying information; inference accuracy would still be degraded since subsequent layers would undergo domain shift. In this case, it would in principle be possible to retrain subsequent layers to adjust to this shift and regain lost accuracy, assuming subsequent layers are sufficiently high-capacity. Alternatively, layer replacement may actually destroy information contained in the layer's inputs, and subsequent layers would be unable to recover original accuracy.

To accelerate full networks, while ameliorating the first cause of degraded accuracy, we propose incremental layer replacement. For an L-layer NN with layers indexed as $l = 1, \ldots, L$, we start with the input layer 1 and proceed towards the classification layer L. At each step l, we first replace the layer by collecting its inputs for the training data. We then freeze layers $1, \ldots, l$ (now using our non-differentiable lookups) and fine-tune the weights of layers $l+1, \ldots, L$ (still using exact matrix multiplication) on the training data.

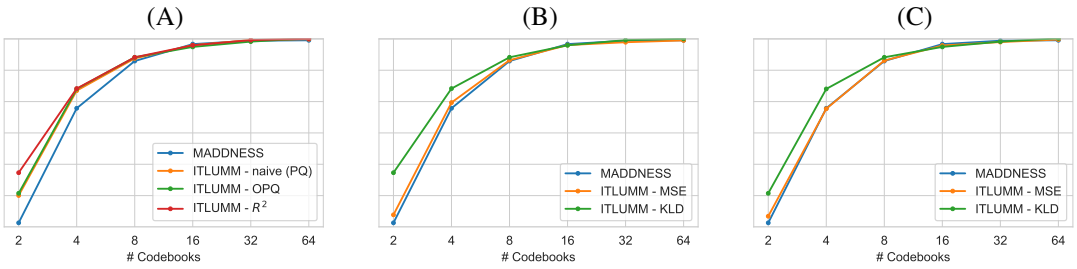

Figure 1: Comparison of MADDNESS and ITLUMM accuracy vs efficiency tradeoff for CIFAR-100 classifier layer. (A) Effect of ITLUMM partitioning strategy.(B) Effect of nonlinearity function when ITLUMM uses R^2-based partitioning. (C) Effect of nonlinearity function when ITLUMM uses OPQ-based partitioning.

3 Experiments

3.1 Approximating Softmax Classifiers

We first repeat the softmax classifier experiments of [6] using the final dense classifier layers of VGG-like neural networks trained on CIFAR-100. In both these networks, the final activations are 512-dimensional, so the $\boldsymbol{A}$ matrices are $10{,}000 \times 512$ in both cases, while the $\boldsymbol{B}$ matrices are 512×10 and 512×100, respectively. Because MADDNESS was Pareto optimal compared to its prior works, we compare only it to our improved methods. Also, because the inference runtimes and memory usage are identical between MADDNESS and ITLUMM, depending only on the number of codebooks, we show results in terms of number of codebooks rather than the runtime speedup (which relies on assumptions regarding hardware support and software implementation). For each experimental setting, we plot the relative accuracy, the classification accuracy of a given setting divided by the accuracy obtained from using exact matrix multiplication.

Our results are shown in Figure 1. Overall, we see that all settings of ITLUMM (depicted in orange, green, and red) are superior to MADDNESS (depicted in blue). We also see the effect of the partitioning heuristic in Figure 1(A). For ITLUMM we fix the prototype-optimization objective to be the KL-divergence; we also plot the accuracy of MADDNESS as a reference. We see that the OPQ heuristic provides essentially no improvement over naive (PQ) partitioning. Meanwhile, R^2-clustering partitioning provides a modest improvement. Meanwhile, we see the effect of the prototype-optimization objective in Figure 1(B & C). We see that, regardless of the choice of partitioning heuristic, minimizing the KL-divergence (KLD) is better than minimizing mean-squared error (MSE).

3.2 Accelerating Full Networks

Our goal is to replace all of a network's matrix multiplications with lookup tables. To that end, we first evaluated our proposed approach on a simple multi-layer perceptron (MLP) trained on the MNIST dataset. The network has 4 layers, with 30 neurons per layer. We began with an ablation study to measure the effect that substituting lookup for multiplication has on accuracy in each layer, then evaluated the effect of said substitution on every layer.

Figure 2 shows the accuracy of a network as a function of the number of codebooks when a given layer's matrix multiplication has been replaced by a lookup table. Higher layers in the network, those closer to the output, are more robust to the error introduced by a lookup tables. However, the performance breakeven point is between 2 and 3 codebooks, and the degradation of accuracy is quite severe. Because of the compounding nature of error in neural networks, it's likely that replacing all of the layers in the network with ITLUMM will result in even more severe degradation.

Indeed, Table 1 shows that replacing all of the matrix multiplications in this network with ITLUMM results in accuracy worse than that due to replacing any single layer. Since the performance breakeven point is 2 codebooks, accelerating the entire network ITLUMM requires a loss of more than 60% accuracy.

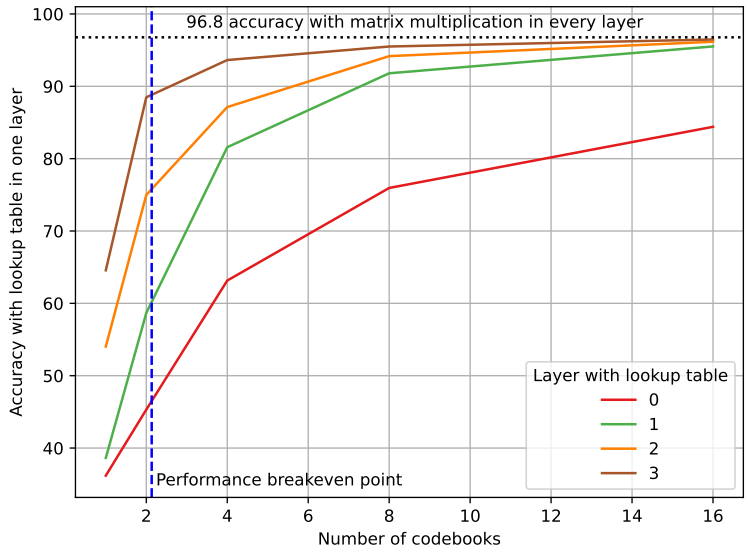

Figure 2: Effect of ITLUMM on MNIST accuracy when converting a single layer to use ITLUMM instead of matrix multiplication. The original network is a 4-layer MLP. The accuracy of the original network, with matrix multiplication in every layer, is the upper bound (black dotted line). Any configuration with fewer codebooks than the performance breakeven point is faster than matrix multiplication (blue dashed line).

Number of codebooks	Accuracy	Faster
1	36.1	YES
2	36.3	YES
4	52.0	NO
8	70.5	NO
16	84.9	NO

Table 1: Effect of ITLUMM on MNIST accuracy when converting every layer to use ITLUMM instead of matrix multiplication. A configuration in the table is faster than matrix multiplication when its number of codebooks is less than the breakeven point in Figure 2.

4 Discussion

As expected, ITLUMM's model-aware lookup-table optimization improves upon MADDNESS. Meanwhile, optimized partitioning provides little improvement, surprisingly so given the substantial benefits of OPQ in maximum inner-product search (MIPS) [16, 11]. However, we are not able to obtain the full benefits of OPQ because the applied rotation must be a permutation. Furthermore, in the MIPS setting, input data are expected to have redundancy/correlation and a wide range of variances, while NN activations are regularized (e.g. with dropout) [17] to avoid this.

We suggest that future research focus on improving the MADDNESS hash function, which appears to be an accuracy bottleneck. One could exploit knowledge of weight matrix $\boldsymbol{B}$ while learning hash function parameters, as done for NN scalar quantization [18]. More substantially, a differentiable hash function could improve inference accuracy and speed up conversion of full networks.

5 Conclusion

We proposed ITLUMM, an approach for accelerating matrix multiplication by replacing multiplications with look-ups, which improves upon previous work. In the context of deep learning inference, where model weights are known, our approach advanced classification accuracy beyond prior work in MADDNESS [6]. Our approach and analysis on full neural networks informs the community on the current state of hashing-based acceleration, and points the way to potential future advances.

References

[1] Cristian Bucilua, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–541, 2006. 1

[2] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015. 1

[3] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. 1

[4] Steven A Janowsky. Pruning versus clipping in neural networks. *Physical Review A*, 39(12):6600, 1989. 1

[5] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4820–4828, 2016. 1

[6] Davis Blalock and John Guttag. Multiplying matrices without multiplying. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 992–1004. PMLR, 18–24 Jul 2021. 1, 4, 5

[7] Robert Gray. Vector quantization. *IEEE Assp Magazine*, 1(2):4–29, 1984. 1

[8] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2010. 1

[9] Yann LeCun, Corinna Cortes, and Christopher J Burges. Mnist handwritten digit database. 2010. *URL http://yann. lecun. com/exdb/mnist*, 7(23):6, 2010. 2

[10] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 2

[11] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. Optimized product quantization. *IEEE transactions on pattern analysis and machine intelligence*, 36(4):744–755, 2013. 2, 5

[12] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955. 3

[13] Daniel Müllner. fastcluster: Fast hierarchical, agglomerative clustering routines for r and python. *Journal of Statistical Software*, 53:1–18, 2013. 3

[14] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. Accelerating large-scale inference with anisotropic vector quantization. In *International Conference on Machine Learning*, pages 3887–3896. PMLR, 2020. 3

[15] Avner May, Jian Zhang, Tri Dao, and Christopher Ré. On the downstream performance of compressed word embeddings. *Advances in neural information processing systems*, 32, 2019. 3

[16] Mohammad Norouzi and David J Fleet. Cartesian k-means. In *Proceedings of the IEEE Conference on computer Vision and Pattern Recognition*, pages 3017–3024, 2013. 5

[17] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014. 5

[18] Pierre Stock, Armand Joulin, Rémi Gribonval, Benjamin Graham, and Hervé Jégou. And the bit goes down: Revisiting the quantization of neural networks. In *ICLR 2020-Eighth International Conference on Learning Representations*, pages 1–11, 2020. 5