# Neural Network Weight Compression with NNW-BDI

Andrei Bersatti*
Georgia Institute of Technology
Atlanta, Georgia
abersatti@gatech.edu

Nima Shoghi*
Georgia Institute of Technology
Atlanta, Georgia
nimash@gatech.edu

Hyesoon Kim
Georgia Institute of Technology
Atlanta, Georgia
hyesoon@cc.gatech.edu

## ABSTRACT

Memory is a scarce resource and increasingly so in the age of deep neural networks. Memory compression is a solution to the memory scarcity problem. This work proposes NNW-BDI, a scheme for compressing pretrained neural network weights. NNW-BDI is a variation to standard Base-Delta-Immediate [13] compression technique to make it a better fit for neural network weights, using techniques such as quantization, downscaling, randomized base selection, and base-delta-configuration adjustment. We evaluate our algorithm by compressing the weights of a MNIST classification network. Our evaluation shows that NNW-BDI reduces memory usage by up to 85% percent without any reduction in inference accuracy.

## CCS CONCEPTS

• **Computer systems organization → Neural networks**; • **Information systems → Data compression**.

## KEYWORDS

memory, compression, neural networks

**ACM Reference Format:**
Andrei Bersatti, Nima Shoghi, and Hyesoon Kim. 2020. Neural Network Weight Compression with NNW-BDI. In *The International Symposium on Memory Systems (MEMSYS 2020), September 28-October 1, 2020, Washington, DC, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3422575.3422805

## 1 INTRODUCTION

The demand for memory has accelerated with the explosion of data and of data-intensive applications and particularly with the increasing depths of neural networks. Weights represent a significant proportion of used memory in DNNs. Memory compression is a proposed solution to this problem. Compressed data can be represented with a reduced fraction of its original size, thus saving both memory and energy.

Several solutions have been proposed to compress data on a computer system. We observe that, among the different compression alternatives, delta-based encoding is one that does not require pre-processing for building a table ahead of time, as Huffman-based schemes do, and does not require a value lookup, as frequent value or frequent pattern representations do. Therefore, BDI has been widely adopted in many hardware solutions [1, 8, 9, 17]. We thus find that Base-Delta-Immediate (BDI) [13] is a good fit for our purposes. However, the characteristics of weights on a neural network are such that, by their nature, the values' dynamic ranges are higher than the ones that work well with existing delta-based compression solutions. We observe that the characteristics of weight values and the ensuing spread of data are such that existing delta-based solutions fail to compress the weights. As a solution to this problem we propose a neural network weight compression solution scheme with a modified delta-based compression that extends the delta values with a multiplier implemented as a scale factor bit shift.

After we ran simulations on MNIST, we observed that we can reduce the memory footprint by up to 85%.

Our contribution will result in:

- Proposing a delta-based compression modification that is suitable to Neural Network weight values that contain a greater dynamic range and that incorporates a scale factor component to accommodate greater value ranges.
- Analyzing the trade off of the scale factor versus the accuracy of the network.
- Proposing a design that incorporates and makes use of the delta-based compression scale factor for weights compression suitable for neural networks.

### 1.1 Comparison to Prior Work

There are many different ways of dealing with memory scarcity for neural networks. Neural network pruning compresses a network by removing parameters. Most pruning methods involve computationally expensive compression steps [2], making them unfavorable in many use cases. Compression methods like GIST [10] and JPEG-ACT [4] are meant for training and function on convolutional neural networks only. Our method does not decrease the depth of the network, is not computationally hard to execute, and was designed to work on all kinds of neural network inference tasks.

## 2 BACKGROUND

This paper presents a solution that is based on two different compression methods: BDI, which is lossless compression, and Quantization, which is lossy compression.

### 2.1 Base-Delta-Immediate

Delta-based encoding is based on the observation that values that are spatially close in memory tend to be close in value to each other. In other words, it takes advantage of what its authors call 'low dynamic range'. Base-Delta-Immediate Compression [13], proposed

*Both authors contributed equally to this research.

*MEMSYS 2020, September 28-October 1, 2020, Washington, DC, USA*
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-8899-3/20/09…$15.00
https://doi.org/10.1145/3422575.3422805

by Pekhimenko et. al., is a computationally simple compression method that works by establishing a base value for the first seen value on the cache line, and only the delta, or the difference from the base value, for all the other values on the cache line. The authors observed, through experimentation, that using two base values provided the optimum compression for standard BDI's compression of caches. They figured out that, since determining the optimum base value is expensive, and since a lot of values are narrow values close to 0, that the best bases to use were, for a fixed delta size: 1) the value 0, which requires no calculation (the immediate value), and 2) the first value in the message which is not compressible by base 0. The decompression process for any value involves 1) finding the base (or immediate) used to encode that value, 2) finding the memory offset of that value's delta, and 3) adding the base and the delta at the offset.

## 2.2 Quantization

Sometimes information loss can be tolerable. Quantization involves a reduction in the number of bits to represent numbers by approximating values. This is usually done by converting floating point numbers into fixed-range integers (e.g., 32 bit integer). There are many different types of quantization, including uniform quantization, JPEG quantization, and logarithmic quantization. Uniform quantization divides every element by a scale factor and truncates the results to the nearest integers. JPEG quantization [16] happens after the application of the discrete cosine transform and is very similar to uniform transformation, but the scale factor is picked from the JPEG quantization matrix entry corresponding to value being quantized. Finally, logarithmic quantization [5, 6, 14] applies uniform quantization on the base-2 logarithm of the input. At the extreme, Courbariaux et.al [3] proposed a form of quantization, BinaryConnect, that quantizes weights to one bit: 1's and 0's.

## 3 NEURAL NETWORK WEIGHT - BDI (NNW-BDI)

We next discuss the implementation of our proposed solution by first presenting an analysis, then a study of the characteristics of neural network weights as constrained by the result of such analysis, and finally describing the high-level design.

## 3.1 Limitations of Using BDI for Neural Network Weight Compression

To conserve energy, the compression technique chosen for neural network weights during inference should minimize the amount of pre-processing required and its use of lookup tables. Base-Delta-Immediate is a solution that addresses these concerns. BDI exploits the property of low dynamic range by taking advantage of the observation that the differences between values tend to be smaller than the values themselves. However, standard BDI's limitations when implemented as originally designed are: 1) BDI is not well-suited for compressing floating point values, and 2) it is not optimally suitable for neural network weight values since weight values tend to have a greater dynamic range than suitable for standard, unmodified BDI.

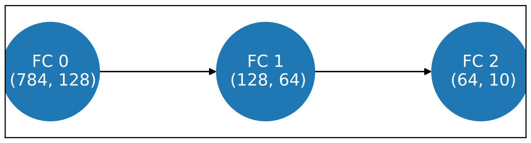

**Figure 1: Our evaluation neural network's architecture consists of a 784 neuron fully connected input layer, a 128 neuron fully connected hidden layer, and a 64 neuron fully connected output layer.**

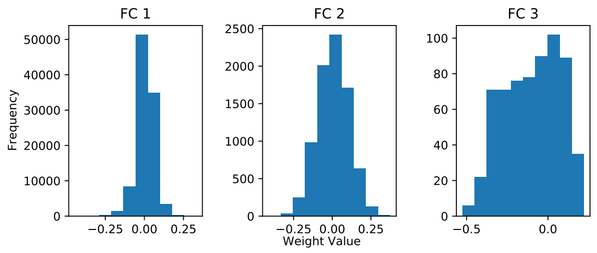

**Figure 2: The weight distribution of each layer of our neural network (see fig. 1 for the architecture of this network).**

## 3.2 Characteristics of Neural Network Weights

Neural network weights are usually represented as continuous arrays of 32-bit floating point values, usually represented as tensors. During training, these weights are initialized randomly and get optimized. Once trained, different layers and network architectures exhibit different weight distributions. Figure 2 shows histograms of weight distributions for a basic MNIST classifier (fig. 1 shows the network's architecture). Studying the weight distribution results in two important takeaways:

(1) The 32-bit floating point granularity of these weights gives them very high dynamic range when used at the highest precision.
(2) Layer weights exhibit unimodal or bimodal weight distributions. Therefore, if we can increase the range of values that deltas can cover without significant accuracy loss, we can use a simple method like BDI to compress these weights.

## 3.3 High-Level Design

We now describe the high-level design of our neural network compression solution in this subsection. This work is built on top of BDI[13]. Standard BDI is modified to address some of its inherent problems dealing with neural network weight distributions. We focus on the relevant changes to make the technique suitable for compression on neural network weights. As we have analyzed in the previous section, the main challenge with this task is the high dynamic range of the weight distributions. To tackle this, we employ the following techniques:

- We make the compression **lossy** by **quantizing** the layer weights from 32 bit floating point values to 32 bit integers and

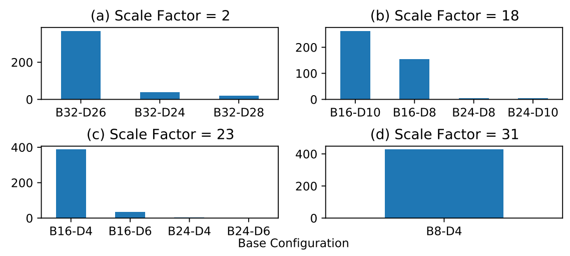

**Figure 3: Base configuration usages when compressing our neural network with different scale factors.**

applying a constant **scale factor** that decreases the precision of the quantized value.

- We use a **randomized** base selection algorithm, increasing the chances of picking more optimal bases.
- We increase the **cache line size** and **number of bases** and adjust the **base configurations**.

### 3.3.1 *Quantization & Scale Factor.*

BDI was originally intended for cache line compression and thus has a strict requirement of being lossless. Machine learning inference workloads, however, do not have the same precision requirements, and inference has been shown to work with as low as 3, 2, or 1 bits [3, 15]. We introduce a quantization and truncation mechanism for storing a lower-precision version of the network's weights. We first use a variant of uniform quantization to quantize the network's weights into 32 bit integer. Our quantization scheme uses stochastic rounding [7] in order to remediate the reduced numerical precision in the network introduced by the quantization and scaling of values. We then reduce the precision of the quantized values by applying the scale factor hyperparameter. This hyperparameter is constant for the network (stored as metadata) and does not incur any additional memory overhead. The scale factor is applied by use of a shift unit:

$$W_{quantized} = stochastic\_round\left(\frac{W}{|max(W)|} \cdot 2^{31}\right)$$
$$W_{scaled} = W_{quantized} >> scale\_factor$$

NNW-BDI is then executed on $W_{scaled}$. By doing this, we increase the dynamic range of our quantized weights. When we want to decompress, we undo the downscaling operation by left shifting the decompressed values by the scale factor. This operation truncates the lower-bit information (the least significant bits) of our quantized values while maintaining the most significant bits' information.

### 3.3.2 *Cache Line Size, Base Count, & Base Configurations.*

Standard BDI works on 32 byte cache lines, but we are working with much larger tensors (e.g., a basic $64x32x3x3$ 2D convolution layer consists of 18432 unique 32-bit floating point weights). When working with memory compression, lower cache line sizes lead to higher fragmentation, increasing the compression overhead. Additionally, when working with more uniform distributions that have a common base, lower cache line sizes lead to redundant storage of similar bases for contiguous cache lines. Therefore, we increase the cache line size. We found 1024 bytes to be a good size for simple inference workloads.

This increased cache line size, when coupled with the high dynamic range of our data, makes compression with 1 base and 1 immediate (the immediate being '0' as outlined in BDI [13]) nearly impossible. To compensate for this, we increase the base count to allow for base counts up to 24 (see average base counts in fig. 5). Increasing the base count increases the size of the metadata needed for each element. To keep this size increase manageable, we only consider base count increases in steps of 4 (e.g., 4, 8, 12, ...). For any element, the base and delta memory positions are calculated as shown below (base_count, base_size, and delta_size are determined per cache line by the current encoding and are known statically):

```
base  = cache_line
        + base_index * base_size
delta = cache_line
        + base_count * base_size
        + delta_index * delta_size
```

BDI always tries the same static set of configurations (e.g., base8-delta4, base8-delta4, etc). This method does not work for our data. To deal with the high dynamic range of our data, we increase the granularity of base sizes and delta sizes (e.g., allowing for 16 bit base size and 10 bit delta size). As our scale factor increases, we also adjust the configurations used. For example, when using a scale factor of 24 bits, a configuration like base32-delta24 would never be used and wastes memory (as we have to account for it in metadata if it is being considered). Our analysis suggests that at every scale factor, only a few configurations are actually being used. Figure 3 shows examples of the base configurations and base counts for different scale factors. As a result, NNW-BDI is designed to use different configurations at different scale factors to minimize the total number of configurations. For example, if the scale factor is 18, NNW-BDI only uses the base16-delta10, base16-delta8, base24-delta8, and base24-delta10 configurations. This process massively decreases the metadata and computation overhead while maintaining the high granularity.

### 3.3.3 *Randomized Base Selection.*

BDI uses a greedy base selection mechanism. When a value cannot be compressed by the existing bases, the algorithm picks that value as the next base. While this maximizes the simplicity and efficiency of BDI, it also leads to inefficient base selection for unstructured data layouts. NNW-BDI uses a randomized base selection algorithm instead. This algorithm works as follows:

```
function FindNewBase($cache\_line$, $uncompressed\_size$)
    $i \leftarrow 0$
    for all $value \in cache\_line$ do
        if $value$ is not compressed then
            if $rand(0,1) <= \frac{i}{|uncompressed\_size|}$ then
                return $value$
            else
                $i \leftarrow i + 1$
            end if
        end if
    end for
end function
```

The base selection efficiencies of compressing our neural network with and without randomized base selection are shown on

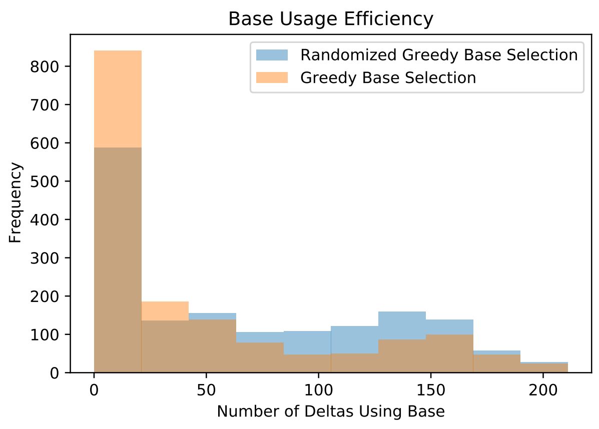

**Figure 4: The number of deltas using each base after compressing our neural network's weights using a scale factor of 27 (note: other bases exhibit similar behaviors). Randomized base selection picks more efficient bases, decreasing the number of bases that do not have many deltas associated with them.**

fig. 4. The run with randomized base selection exhibits a much more efficient base selection, having much less inefficient bases (i.e., those with a few deltas).

## 3.4 Algorithm Summary

In this section, we explain the entire BDI-NNW algorithm through a worked example. For this example, our scale factor is 18 bits, and the configurations supported are Base14-Delta14, Base14-Delta8, and Base24-Delta10. The number of bases supported is 20.

*3.4.1 Compression.* Compression takes a 1024-byte contiguous 32-bit floating point chunk of memory as input. This input is then quantized to 32 bit integers. The quantized 32 bit integers are then right shifted by our scale factor, 24 bits, resulting in a 12 bit representation. The cache line is then compressed with all 3 configurations in parallel, and the configuration that produces the best compression ratio is selected. This compression is done by picking a base using the randomized base selection algorithm and calculating the deltas for the selected base. This operation is repeated until the entire cache line is compressed or the number of bases reaches 20. For every element, the configuration used and the index are stored in the tag.

*3.4.2 Decompression.* To decompress an element of the cache line, we extract the configuration used and the base index from the element's tag. The base and delta are retrieved using the mechanism in section 3.3.2. The quantized output value is calculated by adding the base and delta. We can convert this back to a floating point number by undoing the quantization process.

## 3.5 Hardware & Software Implementation

While the current demonstration of NNW-BDI is carried out in software, one of the biggest appeals of NNW-BDI is its highly parallel nature, making it very ripe for implementation in hardware. The decompression process can be easily parallelized. The decompression of any memory position can be done independently and in parallel to any other memory position using the mechanism described in section 3.3.2.

Compression is also highly parallelizable. When compressing an arbitrary sized region of memory (e.g., a PyTorch tensor representing the quantized weights of a layer), the memory gets chunked into constant-sized batches (e.g., 1024 bytes in our evaluation). Each batch is then compressed independently and in parallel. When compressing a single batch, we attempt to compress the batch with each configuration (similar to BDI); this process can also be done in parallel (i.e., every unique configuration processes the batch independently and the best compression is used). The only sequential component of compression is base selection, as the necessity of a new base depends on the effectiveness of previous bases. For this reason, limiting the number of bases leads to higher performance. Also, the compression step only happens once during inference, and thus sacrificing some performance is acceptable if it results in higher compression ratios.

## 4 EVALUATION

To get proper accuracy and compression ratio results, we have created a neural network inference pipeline that simulates the compression/decompression process. For inference, we compress and subsequently decompress the pretrained model into a fully fledged PyTorch [11] model. We then perform inference using these decompressed weights. This allows us to measure the test accuracy loss from the lossy quantization and scale factor step. During this process, we measure **inference accuracy**, **compression ratio** compared to the uncompressed 32 bit floating point weights, average **compression time** per cache line, and average **base count** per cache line. Compression ratio is calculated without metadata overhead, but NNW-BDI attempts to minimize the metadata size (see section 3.3.2 for more details). The compression time is the average time it takes for our software implementation to compress a single block. Because of the unreliability of software performance timing, this metric is only used relative to other data points metric (e.g., Greedy Base Selection vs. Random Base Selection).

All experiments are ran on a pretrained MNIST classification network. The architecture of this network is shown in figure fig. 1. This network architecture is used because of its simplicity, giving us the ability to reason about and visualize the effect of our lossy compression on the network's performance. During our experiment, we vary the scale factor, running our compression algorithm under different configurations, scale factors, and base selection mechanisms. For the baseline, we perform inference with the lossy quantization and scale factor application only, without using any base-delta encoding. This allows us to focus the effectiveness of the base-delta compression by discounting the effect of downscaling (i.e., lossy compression due to quantization). Our results are shown in fig. 5.

## 4.1 Analysis

*4.1.1 Benefits Over Quantization.* NNW-BDI employs quantization, described in section 3.3.1, as part of its implementation in

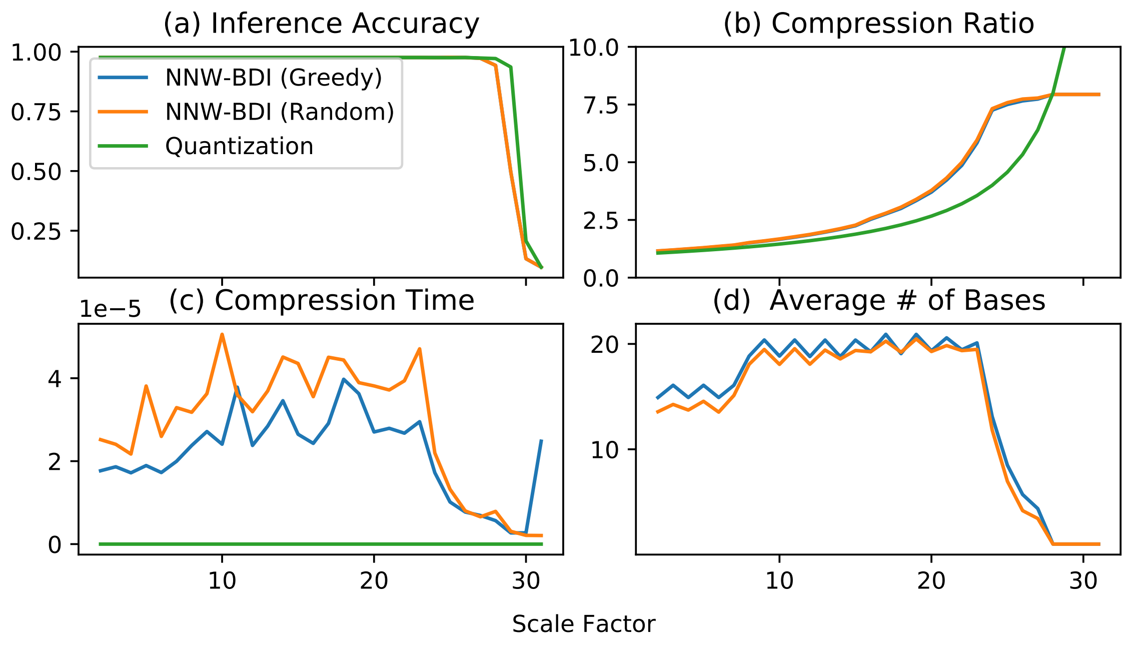

**Figure 5: Inference accuracy, compression ratio (compared to uncompressed 32 bit value), compression time (in seconds), and average number of bases when compressing and running our network under greedy NNW-BDI, randomized NNW-BDI, and quantization. The X-axis represents the scale factor and is shared for all graphs.**

order to make floating point weight values representable by integer values which are better compressible under BDI. fig. 5b shows that for most scale factors we achieve higher compression ratios than quantization alone without any accuracy loss. However, once our scale factor reaches very high values (> 28), quantization becomes the clear choice, exhibiting higher accuracy and higher compression ratio. On average, the NNW-BDI compression ratio was 37% higher than the compression ratio from quantization alone, with the NNW-BDI compression ratio being 3.32 higher than the quantization compression ratio when the scale factor is 24. As the inference tasks become more difficult, we suspect that the benefit of our scheme will only increase in terms of memory saved and accuracy when compared to quantization alone.

*4.1.2 Performance Impact of Different Scale Factors.* The compression time is used to get an understanding of the relative performance of the algorithm as we change the scale factor. With this in mind, it is evident that increasing the number of bases increases the total compression time. This is primarily due to the heavily sequential nature of the base selection mechanism (see section 3.5 for more details).

*4.1.3 Base-Selection Algorithm.* Finally, the data shows that while our randomized base-selection algorithm does not change accuracy or compression ratio, it decreases the average base count. This validates our theory that randomized base selection leads to more optimal bases. This is ideal because lower base counts means less computation during decompression. This improvement, however, comes at a clear trade-off, as the randomized base selection algorithm is, on average, 28% slower in compression. However, the decompression time, which is the critical path, does not change.

## 5 CONCLUSION

This study introduced NNW-BDI, a modified delta-based compression algorithm that is well-suited for compressing neural network weights. We studied the effectiveness of compressing weights of deep neural networks using the Base-Delta-Immediate algorithm. We found that these weights have a very high dynamic range at 32-bit floating point precision, making standard BDI impractical. We tackled these challenges by introducing NNW-BDI, a modified delta-based compression algorithm that is well-suited for compressing neural network weights. Our solution uses techniques such as quantization, downscaling, and randomized base selection, adjusts the cache-line size, number of bases, and base configurations for optimal compression rate and minimal accuracy loss. We evaluated this model with different configurations and compared it to using quantization only. Our results show that NNW-BDI has the potential of achieving a higher compression ratio than quantization only, while maintaining a high level of test accuracy. This work, similarly to standard BDI, can be adapted to work with memory via Linearly Compressed Pages[12].

## ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers.

Andrei Bersatti would like to thank his employer, Plus Technologies Consulting, Inc. and its parent company, Plus Holding

International Limited, for the moral support and for the flexibility granted that allowed for Andrei to carry out this research.

## REFERENCES

[1] A. Arunkumar, S. Lee, V. Soundararajan, and C. Wu. 2018. LATTE-CC: Latency Tolerance Aware Adaptive Cache Compression Management for Energy Efficient GPUs. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 221–234.

[2] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. 2020. What is the state of neural network pruning? *arXiv preprint arXiv:2003.03033* (2020).

[3] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. BinaryConnect: Training Deep Neural Networks with Binary Weights during Propagations. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2* (Montreal, Canada) *(NIPS'15)*. MIT Press, Cambridge, MA, USA, 3123–3131.

[4] R. David Evans, Lufei Liu, and Tor M. Aamodt. 2020. JPEG-ACT: Accelerating Deep Learning via Transform-based Lossy Compression. In *Proceedings of the 47th Annual International Symposium on Computer Architecture (ISCA 47)*. https://doi.org/10.1109/ISCA45697.2020.00075

[5] Y. Fang, P. Chou, B. Chen, T. Lin, and J. Wang. 2017. An all-n-type dynamic adder for ultra-low-leakage IoT devices. In *2017 IEEE 12th International Conference on ASIC (ASICON)*. 68–71.

[6] M. Gautschi, M. Schaffner, F. K. Gürkaynak, and L. Benini. 2016. 4.6 A 65nm CMOS 6.4-to-29.2pJ/FLOP@0.8V shared logarithmic floating point unit for acceleration of nonlinear function kernels in a tightly coupled processor cluster. In *2016 IEEE International Solid-State Circuits Conference (ISSCC)*. 82–83.

[7] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep Learning with Limited Numerical Precision. *CoRR* abs/1502.02551 (2015). arXiv:1502.02551 http://arxiv.org/abs/1502.02551

[8] M. Hemmat, T. Shah, Y. Chen, and J. S. Miguel. 2020. CRANIA: Unlocking Data and Value Reuse in Iterative Neural Network Architectures. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 295–300.

[9] Seokin Hong, Prashant J. Nair, Bulent Abali, Alper Buyuktosunoglu, Kyu-Hyoun Kim, and Michael B. Healy. 2018. Attaché: Towards Ideal Memory Compression by Mitigating Metadata Bandwidth Overheads. In *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture* (Fukuoka, Japan) *(MICRO-51)*. IEEE Press, 326–338. https://doi.org/10.1109/MICRO.2018.00034

[10] A. Jain, A. Phanishayee, J. Mars, L. Tang, and G. Pekhimenko. 2018. Gist: Efficient Data Encoding for Deep Neural Network Training. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. 776–789.

[11] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[12] Gennady Pekhimenko, Vivek Seshadri, Yoongu Kim, Hongyi Xin, Onur Mutlu, Phillip B. Gibbons, Michael A. Kozuch, and Todd C. Mowry. 2013. Linearly Compressed Pages: A Low-Complexity, Low-Latency Main Memory Compression Framework. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture* (Davis, California) *(MICRO-46)*. Association for Computing Machinery, New York, NY, USA, 172–184. https://doi.org/10.1145/2540708.2540724

[13] G. Pekhimenko, V. Seshadri, O. Mutlu, M. A. Kozuch, P. B. Gibbons, and T. C. Mowry. 2012. Base-delta-immediate compression: Practical data compression for on-chip caches. In *2012 21st International Conference on Parallel Architectures and Compilation Techniques (PACT)*. 377–388.

[14] M. Schaffner, M. Gautschi, F. K. Gürkaynak, and L. Benini. 2016. Accuracy and Performance Trade-Offs of Logarithmic Number Units in Multi-Core Clusters. In *2016 IEEE 23nd Symposium on Computer Arithmetic (ARITH)*. 95–103.

[15] Wonyong Sung, Sungho Shin, and Kyuyeon Hwang. 2015. Resiliency of Deep Neural Networks under Quantization. *ArXiv* abs/1511.06488 (2015).

[16] G. K. Wallace. 1992. The JPEG still picture compression standard. *IEEE Transactions on Consumer Electronics* 38, 1 (1992), xviii–xxxiv.

[17] V. Young, S. Kariyappa, and M. K. Qureshi. 2019. Enabling Transparent Memory-Compression for Commodity Memory Systems. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 570–581.