Binary Graph Neural Networks - Supplementary Material

Mehdi Bahri[1] Gaétan Bahl[2,3] Stefanos Zafeiriou[1]
[1]Imperial College London, UK [2]Université Côte d'Azur - Inria [3]IRT Saint-Exupéry
{m.bahri, s.zafeiriou}@imperial.ac.uk, gaetan.bahl@inria.fr

A. DGCNN and ModelNet40

In this appendix, we provide details of the DGCNN model and of the ModelNet40 dataset ommitted from the main text for brevity.

ModelNet40 classification The ModelNet40 dataset [4] contains 12311 shapes representing 3D CAD models of man-made objects pertaining to 40 categories. We follow the experimental setting of [3] and [1]. We keep 9843 shapes for training and 2468 for testing. We uniformly sample 1024 points on mesh faces weighted by surface area and normalize the resulting point clouds in the unit sphere. The original meshes are discarded. Only the 3D cartesian coordinates (x, y, z) of the points are used as input. We use the same data augmentation techniques (random scaling and perturbations) as [3] and base our implementation on the author's public code[1]. We report the overall accuracy as the model score.

Model architecture All DGCNN models use 4 EdgeConv (or BinEdgeConv or XorEdgeConv) layers with 64, 64, 128, and 256 output channels and no spatial transformer networks. According to the architecture of [3], the output of the four graph convolution layers are concatenated and transformed to node embeddings of dimension 1024. We use both global average pooling and global max pooling to obtain graph embeddings from all node embeddings; the resulting features are concatenated and fed to a three layer MLP classifier with output dimensions 512, 256, and 40 (the number of classes in the dataset). We use dropout with probability $p = 0.5$.

B. Low-level implementation

This appendix provides further details on the low-level implementation and memory cost of our models.

B.1. Parameter counts

We report the counts of binary and floating-point parameters for the baseline DGCNN and our binary models (stage 3) in Table 1.

[1] https://github.com/WangYueFt/dgcnn/tree/master/pytorch

Model	FP32 Param.	Bin. param.	Total param.
Baseline	1,812,648	0	1,812,648
BF1	11,064	1,804,672	1,815,736
BF2	11,064	1,804,672	1,815,736
RF	15,243	1,804,672	1,819,915

Table 1. Number of parameters given by torchsummaryX. Separated into FP and binary operations. 99.39% of the parameters are binary for **BF1** and **BF2**, 99.16% of the parameters are binary for **RF**.

As can be seen in Table 1, our binarization procedure introduces a few extra parameters, but over 99% of the network parameters are binary.

B.2. Profiling and optimization of DGCNN

In order to obtain the data from Section 5.1 of the main paper, we convert our models with the LARQ converter and benchmark them using the LCE benchmark utility.

The pairwise Hamming distance is naively implemented as a matrix multiplication operation (Eq. 21 of the main text), and we obtain the profiler data in Table 2, where we have highlighted the nodes used by that operation. However, not all nodes of these types belong to the three pairwise distances calculations. We thus provide in Table 3 the complete profiler output for only one distance calculation in binary space, of which there are three in the DGCNN models.

These operations account for 24% of the network's run time. Thus, a speed-up of 32x of these operations would reduce them to around 1% of the network's run time, which is negligible.

While we did not have an optimized version integrated with the LARQ runtime at the time of writing, optimizing the pairwise Hamming distance computation in binary space with ARM NEON (SIMD) operations is quite simple, since it can be implemented as popcount($x$xor$y$). On bit-packed 64-bit data (conversion handled by LCE), with feature vectors of dimension 64, this can be written as:

```
#include "arm_neon.h"

// input data in feats
int8_t n_outs = npoints*(npoints-1)/2
int8_t* out = malloc(n_outs*sizeof(int8_t));
for(int i = 0; i < npoints; i++) {
```

Node Type	Avg. ms	Avg %	Times called
TOPK_V2	488.007	22.18%	4
CONCATENATION	384.707	17.485%	6
FULLY_CONNECTED	171.175	7.77994%	32
PRELU	143.086	6.50329%	7
TILE	136.443	6.20137%	4
LceBconv2d	127.371	5.78904%	6
MAX_POOL_2D	122.743	5.5787%	5
MUL	105.993	4.81741%	11
SUB	92.382	4.19878%	4
LceQuantize	91.168	4.14361%	10
NEG	78.453	3.56571%	4
PACK	56.301	2.55889%	4
GATHER	55.989	2.54471%	4
CONV_2D	39.096	1.77692%	2
RESHAPE	35.091	1.59489%	82
ADD	28.557	1.29792%	6
TRANSPOSE	23.829	1.08303%	36
AVERAGE_POOL_2D	8.071	0.366829%	1
SLICE	5.278	0.239886%	64
LceDequantize	5.174	0.235159%	4
SUM	1.132	0.0514497%	1
SQUARE	0.153	0.00695389%	1
SOFTMAX	0.01	0.000454502%	1

Table 2. LCE Profiler data for "BDGCNN BF H", summary by node types. In red: nodes that appear in Matmul op which can be rewritten as NEON operations for Hamming distance.

```
    // load first feature
    uint32x2_t a = vld1_u32(feats + 8*i);

    for(int j = i; j < npoints; j++) {

        //load second feature
        uint32x2_t b = vld1_u32(feats + 8*j);

        b = veor_u32(a, b); // XOR op

        // popcount op
        int8x8_t c = vreinterpret_u32_s8(b);
        c = vcnt_s8(c);

        // reduce to single number
        // by adding as a tree
        int64x1_t res;
        res = vpaddl_s32(vpaddl_s16(vpaddl_s8(c)));

        //store the output (last 8 bits)
        int8x8_t res8 = vreinterpret_s64_s8(res);
        out[j + npoints*j] = vget_lane_s8(res8, 7);
    }
}
```

Listing 1. Implementation of pairwise Hamming distance in ARM NEON instrinsics (for readability). Note that this code actually treats 64 features at a time and could thus provide a 64x speedup (or more by grouping loads and writes with vld4). We use 32x as a conservative estimate since we couldn't account for LCE's bit-packed conversion.

"TopK" operations account for 22% of the runtime and we view them as incompressible in our simulation (Table 2). It is possible that they could be written in NEON as well, however, this optimization is not as trivial as the Hamming distance one. Remaining operations, such as "Concatenation", cannot be optimized further.

Contrary to simpler GNNs such as GCN, DGCNN is quite computationally intensive and involves a variety of operations on top of simple dot products, which makes it an interesting challenge for binarization, and illustrate that for complex graph neural networks more efforts are required, such as redefining suitable edge messages for binary graph features, or speeding-up pairwise distances computations, as done in this work. The inherent complexity also limits the attainable speedups from binarization, as shown by the large portion of the runtime taken by memory operations (concatenation) and top-k.

C. Details regarding GraphSAGE

In all experiments, the architecture used is identical to that used as a baseline by the OGB team. We report the accuracy following verbatim the experimental procedure of the OGB benchmark, using the suitable provided evaluators and dataset splits. Due to the very large number of edges in the dataset, we were unable to implement LSP in a sufficiently scalable manner (although the forward pass of the similarity computation can be implemented efficiently, the gradient of the similarity with respect to the node features is a tensor of size $|\mathcal{E}| \times |\mathcal{V}| \times D$ where $|\mathcal{E}|$ is the number of edges in the graph, $|\mathcal{V}|$ the number of nodes, and D the dimension of the features. Although the tensor is sparse, Pytorch currently did not have sufficient support of sparse tensors for gradients. We therefore chose not to include the results in the main text. We report the results of our binary GraphSAGE models, against two floating-point baselines: GraphSAGE and GCN.

D. Balance functions

For completeness, we also report the results at stage 2 of the multi-stage distillation scheme in Table 4. It is apparent that the additional operations degraded the performance not only for the full-binary models of stage 3, but also for the models for which all inputs are binary but weights are real.

E. Table of mathematical operators

References

[1] R Qi Charles, Hao Su, Mo Kaichun, and Leonidas J Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *CVPR*, 2017. 1

[2] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-Net : ImageNet Classification Using Binary. *ECCV*, 2016. 4

[3] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic Graph CNN for Learning on Point Clouds. *ACM Trans. on Graphics*, 38(5), 2019. 1

[4] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3D ShapeNets: A deep representation for volumetric shapes. In *CVPR*, 2015. 1

Node type	Avg. time	Avg? %	Operation name
TRANSPOSE	5.91618	0.268874%	[bin_dgcnn_b_f1_h/MatMul_315]:208
SLICE	0.45752	0.020793%	[bin_dgcnn_b_f1_h/MatMul_316]:209
RESHAPE	0.43792	0.0199023%	[bin_dgcnn_b_f1_h/MatMul_317]:210
SLICE	0.14872	0.00675892%	[bin_dgcnn_b_f1_h/MatMul_318]:211
RESHAPE	0.22018	0.0100066%	[bin_dgcnn_b_f1_h/MatMul_319]:212
SLICE	0.16006	0.0072743%	[bin_dgcnn_b_f1_h/MatMul_320]:213
RESHAPE	0.22242	0.0101084%	[bin_dgcnn_b_f1_h/MatMul_321]:214
SLICE	0.16268	0.00739337%	[bin_dgcnn_b_f1_h/MatMul_322]:215
RESHAPE	0.21126	0.0096012%	[bin_dgcnn_b_f1_h/MatMul_323]:216
SLICE	0.15406	0.00700161%	[bin_dgcnn_b_f1_h/MatMul_324]:217
RESHAPE	0.20686	0.00940123%	[bin_dgcnn_b_f1_h/MatMul_325]:218
SLICE	0.1494	0.00678983%	[bin_dgcnn_b_f1_h/MatMul_326]:219
RESHAPE	0.21348	0.00970209%	[bin_dgcnn_b_f1_h/MatMul_327]:220
SLICE	0.15514	0.00705069%	[bin_dgcnn_b_f1_h/MatMul_328]:221
RESHAPE	0.2117	0.00962119%	[bin_dgcnn_b_f1_h/MatMul_329]:222
SLICE	0.15154	0.00688708%	[bin_dgcnn_b_f1_h/MatMul_330]:223
RESHAPE	0.17318	0.00787056%	[bin_dgcnn_b_f1_h/MatMul_331]:224
SLICE	0.15244	0.00692799%	[bin_dgcnn_b_f1_h/MatMul_332]:225
RESHAPE	0.16864	0.00766423%	[bin_dgcnn_b_f1_h/MatMul_333]:226
SLICE	0.15614	0.00709614%	[bin_dgcnn_b_f1_h/MatMul_334]:227
RESHAPE	0.1736	0.00788965%	[bin_dgcnn_b_f1_h/MatMul_335]:228
SLICE	0.15118	0.00687072%	[bin_dgcnn_b_f1_h/MatMul_336]:229
RESHAPE	0.17086	0.00776513%	[bin_dgcnn_b_f1_h/MatMul_337]:230
SLICE	0.149	0.00677165%	[bin_dgcnn_b_f1_h/MatMul_338]:231
RESHAPE	0.16924	0.0076915%	[bin_dgcnn_b_f1_h/MatMul_339]:232
SLICE	0.1505	0.00683982%	[bin_dgcnn_b_f1_h/MatMul_340]:233
RESHAPE	0.16894	0.00767787%	[bin_dgcnn_b_f1_h/MatMul_341]:234
SLICE	0.14994	0.00681437%	[bin_dgcnn_b_f1_h/MatMul_342]:235
RESHAPE	0.17058	0.0077524%	[bin_dgcnn_b_f1_h/MatMul_343]:236
SLICE	0.15018	0.00682528%	[bin_dgcnn_b_f1_h/MatMul_344]:237
RESHAPE	0.16996	0.00772422%	[bin_dgcnn_b_f1_h/MatMul_345]:238
SLICE	0.1496	0.00679892%	[bin_dgcnn_b_f1_h/MatMul_346]:239
RESHAPE	0.17112	0.00777694%	[bin_dgcnn_b_f1_h/MatMul_347]:240
TRANSPOSE	0.41792	0.0189933%	[bin_dgcnn_b_f1_h/MatMul_348]:241
FULLY_CONNECTED	8.78396	0.399207%	[bin_dgcnn_b_f1_h/MatMul_349]:242
TRANSPOSE	0.72016	0.0327293%	[bin_dgcnn_b_f1_h/MatMul_350]:243
FULLY_CONNECTED	8.64452	0.39287%	[bin_dgcnn_b_f1_h/MatMul_351]:244
TRANSPOSE	0.71804	0.032633%	[bin_dgcnn_b_f1_h/MatMul_352]:245
FULLY_CONNECTED	8.63224	0.392312%	[bin_dgcnn_b_f1_h/MatMul_353]:246
TRANSPOSE	0.72162	0.0327957%	[bin_dgcnn_b_f1_h/MatMul_354]:247
FULLY_CONNECTED	8.62624	0.392039%	[bin_dgcnn_b_f1_h/MatMul_355]:248
TRANSPOSE	0.68654	0.0312014%	[bin_dgcnn_b_f1_h/MatMul_356]:249
FULLY_CONNECTED	8.6722	0.394128%	[bin_dgcnn_b_f1_h/MatMul_357]:250
TRANSPOSE	0.69886	0.0317613%	[bin_dgcnn_b_f1_h/MatMul_358]:251
FULLY_CONNECTED	8.6892	0.394901%	[bin_dgcnn_b_f1_h/MatMul_359]:252
TRANSPOSE	0.71076	0.0323021%	[bin_dgcnn_b_f1_h/MatMul_360]:253
FULLY_CONNECTED	8.70248	0.395504%	[bin_dgcnn_b_f1_h/MatMul_361]:254
TRANSPOSE	0.71256	0.0323839%	[bin_dgcnn_b_f1_h/MatMul_362]:255
FULLY_CONNECTED	8.76456	0.398326%	[bin_dgcnn_b_f1_h/MatMul_363]:256
PACK	13.822	0.628173%	[bin_dgcnn_b_f1_h/MatMul_364]:257
SUB	29.9335	1.3604%	[bin_dgcnn_b_f1_h/sub_3;bin_dgcnn_b_f1_h/MatMul_3;b]:258

Table 3. LCE Profiler data for a single Hamming distance computation as a matrix multiplication, in "BDGCNN BF H".

Model	Stage	KNN	LSP	Global balance	Edge balance	Acc
BF2	2	H	-	-	Median	90.07
BF2	2	H	-	-	Mean	83.87
BF2	2	H	ℓ_2	Median		87.60
BF2	2	H	ℓ_2	Mean		89.47
Baseline BF2	2	H	ℓ_2	None		**91.57**

Table 4. Effect of additional balance functions on models with binary activations but floating-point weights. The performance of the baseline model suffers with the introduction of either mean or median centering prior to quantization.

Symbol	Name	Description
$\|\|.\|\|_H$	Hamming norm	Number of non-zero (or not -1) bits in a binary vector
$d(.,.)_H$	Hamming distance	Number of bits that differ between two binary vectors, equivalent to `popcount(xor())`
$\oplus$	Exclusive OR (XOR)	$1 \oplus 1 = -1 \oplus -1 = -1, \quad -1 \oplus 1 = 1 \oplus -1 = 1$
$\odot$	Hadamard product	Element-wise product between tensors
$\circledast$	Binary-real or Binary-binary dot product or convolution	Equivalent to `popcount(xnor())` (*i.e.* no multiplications) for binary tensors
$\otimes$	Outer product	
$\star$	Dot product or convolution	Denoted by $*$ in [2]
$\|\mathcal{X}\|$	Cardinal of a set $\mathcal{X}$	Number of elements in the set
$\mathbf{x}^{(l)}$	Feature maps at layer l	
$.\|\|.$	Concatenation	
$:=$	Definition	
$\mathbf{x}^{(l)}$	Element $\mathbf{x}$ at layer l	

Table 5. Table of the mathematical operators used in the manuscript.