Efficient Computation of Quantized Neural Networks by $\{-1, +1\}$ Encoding Decomposition

Anonymous Author(s)
Affiliation
Address
email

Abstract

Deep neural networks require extensive computing resources, and can not be efficiently applied to embedded devices such as mobile phones, which seriously limits their applicability. To address this problem, we propose a novel encoding scheme by using $\{-1, +1\}$ to decompose quantized neural networks (QNNs) into multi-branch binary networks, which can be efficiently implemented by bitwise operations (xnor and bitcount) to achieve model compression, computational acceleration and resource saving. Our method can achieve at most $59 \times$ speedup and $32 \times$ memory saving over its full-precision counterparts. Therefore, users can easily achieve different encoding precisions arbitrarily according to their requirements and hardware resources. Our mechanism is very suitable for the use of FPGA and ASIC in terms of data storage and computation, which provides a feasible idea for smart chips. We validate the effectiveness of our method on both large-scale image classification (e.g., ImageNet) and object detection tasks.

1 Introduction

Deep Neural Networks (DNNs) have been successfully applied in many fields, especially in image classification, object detection and natural language processing. Because of numerous parameters and complex model architectures, huge storage space and considerable power consumption are needed. However, for mobile phones and embedded platforms, whose resources are limited, it’s hard to achieve satisfactory performance for industrial applications. With the rapid development of DNNs, more and more computing resources are needed, and the requirements for hardware are becoming higher and higher.

In order to improve the energy efficiency of hardware, achieve model compression or computational acceleration, many solutions have been proposed, such as network sparse and pruning [7, 22, 12, 25, 21], low-rank approximation [5, 11, 23], architecture design [20, 10, 8, 19, 16], model quantization [1, 13, 3, 9, 18, 14], and so on. [17, 6, 28] constrain their weights to $\{-1, +1\}$ or $\{-1, 0, 1\}$ and achieve limited acceleration by using simple accumulation instead of complicated multiplication-accumulations. In particular, [2, 18, 4, 27, 24] quantize activation values and weights to bits and use bitwise logic operations to achieve extreme acceleration ratio in inference process but they are suffering from significant performance degradation. However, most models are proposed for fixed precision, and can not extend to other precision models. They easily fall into local optimal solutions and face slow convergence speed in training process. In order to bridge the gap between low-bit and full-precision and be applied to many cases, we propose a novel encoding scheme of using $\{-1, +1\}$ to easily decompose trained QNNs into multi-branch binary networks. Therefore, the inference process can be efficiently implemented by bitwise operations to achieve model compression, computational acceleration and resource saving.

2 Model Decomposition

As the basic computation in most neural network layers, matrix multiplication costs lots of resources and also is the most time consuming operation. Modern computers store and process data in binary format, thus non-negative integers can be directly encoded by \{0, 1\}. We propose a novel decomposition method to accelerate matrix multiplication as follows: Let \( x = [x_1, x_2, ..., x_N]^T \) and \( w = [w_1, w_2, ..., w_N]^T \) be two vectors of non-negative integers, where \( x_i, w_i \in \{0, 1, 2, ..., N\} \) for \( i = 1, 2, ..., N \). The dot product of these two vectors can be represented as follows:

\[
x^T \cdot w = [x_1, x_2, ..., x_N][w_1, w_2, ..., w_N]^T = \sum_{n=1}^{N} x_n \cdot w_n.
\]

All of the above operations consist of \( N \) multiplications and \( (N-1) \) additions. Based on the above encoding scheme, the vector \( x \) can be encoded to binary form using \( M \) bits, i.e.,

\[
x = [x_M^1, x_M^2, ..., x_M^N][x_M^1, x_M^2, ..., x_M^N]^T.
\]

Then we convert the right-hand side of (2) into the following form:

\[
\begin{bmatrix}
x_M^1 & x_M^2 & \cdots & x_M^N \\
x_{M-1}^1 & x_{M-1}^2 & \cdots & x_{M-1}^N \\
\vdots & \vdots & \ddots & \vdots \\
x_1^1 & x_1^2 & \cdots & x_1^N
\end{bmatrix}
= \begin{bmatrix} x_M^1 \\
x_{M-1}^1 \\
\vdots \\
x_1^1
\end{bmatrix}.
\]

where \( x_i = \sum_{m=1}^{M} 2^{m-1} \cdot x_i^m, x_i^m \in \{0, 1\} \), \( x_i = [x_1^i, x_2^i, ..., x_N^i] \).

In such an encoding scheme, the number of represented states is not greater than \( 2^M \). In addition, we encode another vector \( w \) with \( K \)-bit numbers in the same way. Therefore, the dot product of the two vectors can be computed as follows:

\[
x^T \cdot w = \sum_{n=1}^{N} x_n^1 \cdot w_n^1 = \sum_{n=1}^{N} \left( \sum_{m=1}^{M} 2^{m-1} \cdot x_n^m \right) \cdot \left( \sum_{k=1}^{K} 2^{k-1} \cdot w_k^n \right)
= \sum_{m=1}^{M} \sum_{k=1}^{K} 2^{m+k-2} \cdot x_m^1 \cdot w_k^1.
\]

From the above formulas, the dot product is decomposed into \( M \times K \) sub-operations, in which each element is 0 or 1. Because of the restriction of encoding and without using the sign bit, the above representation can only be used to encode non-negative integers. However, it’s impossible to limit the weights and the values of the activation functions to non-negative integers. In order to encode both positive and negative integers, we propose a novel encoding scheme, which uses \{-1, +1\} as the basic elements rather than \{0, 1\}. Then we can use multiple bitwise operations (i.e., \( \text{nor} \) and \( \text{bitcount} \)) to effectively achieve the above vector multiplications. Our operation mechanism can be suitable for all vector/matrix multiplications. Besides fully connected layers, our mechanism is also suitable for convolution and deconvolution layers in deep neural networks.

3 M-bit Encoding Functions

As an important part in neural networks, activation function can enhance the nonlinear characterization of the networks. In our proposed model decomposition method, encoding function plays a critical role and can encode input data to multi bits (-1 or +1). Those numbers represent the encoding of input data. Therefore, the dot product can be computed by the formula (6). Without other judgment and mapping calculation, we use trigonometric functions as the basic encoding functions. In the end, we use the sign function to hard divide to -1 or +1. The mathematical expression can be formulated as follows:

\[
M \text{BitEncoder}(x) = \begin{cases} 
\varphi_M^m(x) : \text{sign}(\sin(\frac{2^{m-1}}{2^m-1} \pi \cdot x)), & m \in \{1, 2, ..., M-1\}, \\
\varphi_M^M(x) : \text{sign}(\sin(\frac{2^{M-1}}{2^M-1} \pi \cdot x)), & \text{otherwise,}
\end{cases}
\]

where \( \varphi_M^M(x) \) is the encoding function of the highest bit of \( M \text{BitEncoder} \) (i.e., \( m = M \)). The periodicity is obviously different from others because it needs to denote more states.
4 Experiments

In this section, we use the same network architecture described in [17, 2] for CIFAR-10 and choose ResNet-18 as the basic network for ImageNet. It is very hard to train on large-scale training sets (e.g., ImageNet), and thus parameter initialization is particularly important. In particular, the well-trained full-precision model parameters activated by ReLU can be directly used as initialization parameters for our 8-bit quantized network. After fine-tuning dozens of epochs, 8-bit quantized networks can be well-trained. Similarly, we use the 8-bit model parameters as the initialization parameters to train 7-bit quantized networks, and so on. We use the loss computed by quantized parameters to update full precision parameters described as the straight-through estimator [26]. Table 1 lists the performance (e.g., accuracy, speedup ratio, memory saving ratio) of our method and several typical models mentioned above. The accuracies were achieved after dozens of times fine-tuning. If continue training those networks, we can reach slightly better performance. We also use the trained ResNet-18 with the Single Shot MultiBox Detector (SSD) framework [15] to validate object detection tasks. We also use the trained model parameters in ImageNet classification to initialize SSD, and report the experimental results in Table 1 after dozens of times fine-tuning.

We analyze the theoretical performance of our encoding scheme. The theoretical speedup and model compression ratios are given in the following table. Thus, our method can obtain at most $59 \times$ speedup and $32 \times$ memory saving over its full-precision counterparts. It can achieve $59/M K \times$ speedup and $32/K \times$ memory saving by constraining activation values to $M$-bit and the values of weights to $K$-bit, where $M, K \in \{1, 2, ..., 8\}$. In fact, our method can provide 64 available encoding choices, and hence our encoded network with different encoding precisions has different calculation speed, memory requirements and experimental precisions. Here, we use 64-bit binary operation in one clock cycle. If those decompositions are implemented in the FPGA or ASIC platform, the speedup ratios can be much higher.

<table>
<thead>
<tr>
<th>Method</th>
<th>CIFAR-10 (Top-1)</th>
<th>ImageNet (Top-5)</th>
<th>ImageNet (mAP)</th>
<th>VOC</th>
<th>Speedup</th>
<th>MemorySave</th>
</tr>
</thead>
<tbody>
<tr>
<td>BWN [17]</td>
<td>90.10%</td>
<td>60.80%</td>
<td>83.00%</td>
<td>-</td>
<td>~2x</td>
<td>~32x</td>
</tr>
<tr>
<td>BNN [2]</td>
<td>88.60%</td>
<td>42.20%</td>
<td>67.10%</td>
<td>-</td>
<td>~64x</td>
<td>~32x</td>
</tr>
<tr>
<td>TWN [6]</td>
<td>92.56%</td>
<td>61.80%</td>
<td>84.20%</td>
<td>-</td>
<td>~2x</td>
<td>~16x</td>
</tr>
<tr>
<td>XNOR-Net [18]</td>
<td></td>
<td>51.20%</td>
<td>73.20%</td>
<td>-</td>
<td>~58x</td>
<td>~32x</td>
</tr>
<tr>
<td>ABC-Net [14]</td>
<td></td>
<td>65.00%</td>
<td>85.90%</td>
<td>-</td>
<td>-</td>
<td>~6.4x</td>
</tr>
<tr>
<td>Full-Precision</td>
<td>91.40%</td>
<td>68.60%</td>
<td>88.70%</td>
<td>0.6392</td>
<td>1x</td>
<td>1x</td>
</tr>
</tbody>
</table>

**Encoded activations and weights**

| M=K=1      | 90.39% | 47.10% | 71.70% | -     | ~59.00x | ~32x     |
| M=K=2      | 91.06% | 56.30% | 79.48% | -     | ~14.75x | ~16x     |
| M=K=3      | 91.27% | 58.69% | 81.84% | -     | ~6.56x  | ~10.7x   |
| M=K=4      | 91.15% | 59.57% | 82.35% | -     | ~3.69x  | ~8x      |
| M=K=5      | 90.92% | 65.09% | 86.42% | 0.5423 | ~2.36x  | ~6.4x    |
| M=K=6      | 91.01% | 67.04% | 87.69% | 0.6131 | ~1.64x  | ~5.3x    |
| M=K=7      | 90.20% | 68.37% | 88.47% | -     | ~1.20x  | ~4.6x    |
| M=K=8      | 90.43% | 68.63% | 88.70% | 0.6351 | ~0.92x  | ~4x      |

5 Conclusions

In this paper, we proposed a novel encoding scheme of using $\{-1, +1\}$ to decompose QNNs into multi-branch binary networks, in which we used bitwise operations ($\text{xnor}$ and $\text{bitcount}$) to achieve model compression, computational acceleration and resource saving. In particular, we can use the high-bit model parameters to initialize a low-bit model and achieve good results in various applications. Thus, users can easily achieve different encoding precisions arbitrarily according to their requirements (e.g., accuracy and speed) and hardware resources (e.g., memory). This special mechanism of data storage and calculation can yield great performance in FPGA and ASIC, and thus our mechanism is a feasible idea for smart chips. Future works will focus on improving the hardware implementation and chip technology, and exploring some ways to automatically select proper bits for various network architectures (e.g., VGG and ResNet).
References


