Winograd Algorithm for AdderNet

Wenshuo Li [1] **Hanting Chen** [1 2] **Mingqiang Huang** [3] **Xinghao Chen** [1] **Chunjing Xu** [1] **Yunhe Wang** [1]

Abstract

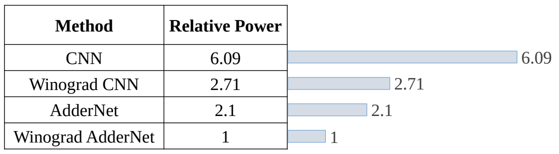
Adder neural network (AdderNet) is a new kind of deep model that replaces the original massive multiplications in convolutions by additions while preserving the high performance. Since the hardware complexity of additions is much lower than that of multiplications, the overall energy consumption is thus reduced significantly. To further optimize the hardware overhead of using AdderNet, this paper studies the winograd algorithm, which is a widely used fast algorithm for accelerating convolution and saving the computational costs. Unfortunately, the conventional Winograd algorithm cannot be directly applied to AdderNets since the distributive law in multiplication is not valid for the ℓ_1-norm. Therefore, we replace the element-wise multiplication in the Winograd equation by additions and then develop a new set of transform matrixes that can enhance the representation ability of output features to maintain the performance. Moreover, we propose the ℓ_2-to-ℓ_1 training strategy to mitigate the negative impacts caused by formal inconsistency. Experimental results on both FPGA and benchmarks show that the new method can further reduce the energy consumption without affecting the accuracy of the original AdderNet.

1. Introduction

The effectiveness of deep neural networks has been well demonstrated in a large variety of machine learning problems. With the rapid development of the accessible datasets, learning theory and algorithms and the computing hardware, the performance of considerable computer vision tasks has been improved by these neural networks, especially convolutional neural networks (CNNs). (Krizhevsky et al., 2012) first applies the deep CNN on the large-scale image classification and a series of subsequent network architectures are explored for boosting the accuracy such as ResNet (He et al., 2016), EfficientNet (Tan & Le, 2019), and GhostNet (Han et al., 2020). In addition, there is a great number of networks presented for addressing different task including object detection (Tan et al., 2020), segmentation (Tao et al., 2020), and low-level computer vision tasks (Guo et al., 2019; Zhang & Patel, 2018; Ren et al., 2019). Although these models can obtain state-of-the-art performance, most of them require massive computations and cannot be easily used on portable devices such as microphones, robots and self-driving cars.

To reduce the computational costs of pre-trained deep neural networks without affecting their performance is also a very important problem, a series of works have been explored for removing the network redundancy such as pruning (Han et al., 2015), distillation (Hinton et al., 2015), and neural architecture search (Liu et al., 2018a). Besides, according to the investigation in (Dally, 2015), the energy consuming varies largely with different operations and different numeric precision (e.g., the energy consumption of an 32-bit multiplication is about 100$\times$ larger than that of an 8-bit addition). Therefore, quantization is now becoming the most common scheme for deploying deep neural networks on resource limited devices. (Qiu et al., 2016) finds that 8-bit fixed-point number is sufficient for CNN to achieve a promising accuracy, and soon 8-bit fixed-point number becomes a common practice.

Furthermore, (Courbariaux et al., 2016) proposes binary networks to quantize neural network to binary values (i.e., +1 and -1) to have an extreme simplification of deep networks. (Rastegari et al., 2016) inserts a scale factor after each binarized layer to enhance the representational capability. (Lin et al., 2017) proposes ABC-Net to use the linear



Method	Relative Power
CNN	6.09
Winograd CNN	2.71
AdderNet	2.1
Winograd AdderNet	1

Figure 1. Comparison of relative power consuming between CNN, Winograd CNN, AdderNet and Winograd AdderNet. All data is achieved under 8-bit fixed-point number. *: The relative power of Winograd CNN is estimated by theoretical analysis.

[1]Noah's Ark Lab, Huawei Technologies. [2]Peking University. [3]Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences. Correspondence to: Wenshuo Li <liwenshuo@huawei.com>, Yunhe Wang <yunhe.wang@huawei.com>.

Proceedings of the 38th International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).

combination of binary bases to approximate floating-number weights and activations. (Liu et al., 2020) proposes RSign and RPReLU to learn the distribution reshape and shift for enhancing the performance. However, the main disadvantage of binary quantization is still the great loss of accuracy, e.g., the performance of the recent binary net is about 5% lower than that of the baseline CNN with the same architecture on the ImageNet benchmark. Recently, (Chen et al., 2020) proposes the adder neural network (AdderNet), which uses the conventional ℓ_1-norm to calculate the output features. Since the cost of addition is much cheaper than that of multiplication (e.g., 8-bit addition is 7 times cheaper than 8-bit multiplication), AdderNet can significantly reduce the energy consumption of a given CNN model with comparable performance (Xu et al., 2020).

Additionally, the fast calculation algorithms, including FFT (Mathieu et al., 2013) and Winograd algorithm (Winograd, 1980), are widely used for improving the efficiency and reducing the computational complexity of deep neural networks. The Winograd algorithm is the most popular and effective method in acclerating CNNs (Lavin & Gray, 2016), since it has the best performance on accelerating 3×3 layers, which is most commonly used in the modern neural architectures. Some following work focuses on the further optimization of the applications of Winograd algorithm for CNNs. To combine Winograd algorithm with neural network pruning, training in Winograd domain is proposed and the results show little loss of accuracy (Liu et al., 2018b).

Although the AdderNet can significantly reduce the overall energy cost of the resulting neural network, the energy consumption of convolutional layers could be obviously optimized by the Winograd algorithm as shown in Figure 1. Thus, we are motivated to explore the fast calculation algorithm for adder layers to further reduce the energy costs of using deep neural networks. However, due to distributive law is not applicable to the operation (i.e., sum of absolute values) in AdderNet, the conventional Winograd algorithm cannot be directly used. Therefore, we first thoroughly analyze the difficulties of applying the Winograd algorithm to AdderNet and explore a new paradigm for optimizing the inference of adder layers. The main contributions of this paper are summarized as follows:

- We propose to inherit the original framework of the Winograd algorithm for optimizing AdderNet, and replace the original element-wise multiplication by adder operation, i.e., the ℓ_1-norm for using additions.
- We then analyze the unbalance of feature maps in the Winograd for AdderNet, and investigate the optimal transform matrix for maximally enhancing the feature representation ability of the new output features. In addition, we present a ℓ_2-to-ℓ_1 training strategy to adapt the Winograd AdderNet paradigm and avoid the decline on the network performance.
- Experiments conducted on benchmark datasets show that the performance of Winograd AdderNet is comparable to that of the baseline model, while achieving an about $2.1\times$ lower energy consumption on Field-Programmable Gate Array (FPGA).

2. Preliminaries

We briefly review the AdderNets and Winograd algorithm.

2.1. AdderNet

Different from convolutional neural networks, AdderNet (Chen et al., 2020) proposes to use ℓ_1-norm to conduct the feed-forward process for extracting features. This method replaces multiplications with additions, which brings benefits for energy consumption and circuits area. The inference process is formulated as

$$Y(m,n,t) = -\sum_{i,j,k} |F(i,j,k,t) - X(m+i,n+j,k)|, \tag{1}$$

where Y represents the output features, F represents weights and X represents input features. The backward process of weights F and feature maps X is approximated with ℓ_2-norm and HardTanh instead of sign function, respectively.

$$\frac{\partial Y(m,n,t)}{\partial F(i,j,k,t)} = X(m+i,n+j,k) - F(i,j,k,t), \tag{2}$$

$$\frac{\partial Y(m,n,t)}{\partial X(m+i,n+j,k)} = HT(F(i,j,k,t) - X(m+i,n+j,k)), \tag{3}$$

where HT$(\cdot)$ is short for HardTanh function

$$HT(\cdot) = \begin{cases} x, & -1 < x < 1, \\ -1, & x < -1, \\ 1, & x > 1. \end{cases}$$

Since the norms of gradients in AdderNet are smaller than that in CNNs, the authors propose an adaptive learning rate for different layers in AdderNets. The learning rate for each layer l could be formulated by:

$$\Delta F_l = \gamma \times \alpha_l \times \Delta L(F_l), \tag{4}$$

$$\alpha_l = \frac{\eta\sqrt{k}}{||\Delta L(F_l)||_2}. \tag{5}$$

The following work expands the application scope, such as super-resolution (Song et al., 2020). AdderNet has shown its potential to replace CNN in many computer vision tasks and attracted a lot of attention.

2.2. Winograd algorithm

Winograd algorithm (Winograd, 1980) is a widely used fast calculation method, which can accelerate the convolution calculation in the signal processing area. (Lavin & Gray, 2016) applies Winograd algorithm in convolutional neural networks and largely reduce the computation cost of CNNs. Denote the length of filter as r, the output length as m and the corresponding Winograd algorithm as $F(m, r)$.

The Winograd algorithm of $F(2, 3)$ can be formulated as

$$Y = A^T[[GgG^T] \odot [B^T dB]]A, \tag{6}$$

$$A = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & -1 \\ 0 & -1 \end{bmatrix}, G = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \end{bmatrix},$$

$$B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 1 \\ -1 & 1 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}. \tag{7}$$

where g represents the 3×3 convolution filter, d represents a 4×4 tile of input feature map and $\odot$ respresents the element-wise multiplication. A, G and B are the transform matrix for output, weights and input, respectively. When forward process is performed, $\hat{g} = GgG^T$ could be calculated at advance to reduce the calculation overhead, since g would not be changed. Therefore, Equation (6) can be reformulated as:

$$Y = A^T[\hat{g} \odot [B^T dB]]A. \tag{8}$$

Considering the complexity of transformation and the numeric precision issues, $F(2\times2, 3\times3)$ is the most commonly used form in practice (Liu et al., 2018b; Yan et al., 2020). Moreover, with $m > 2$ or $r > 3$, the transformation matrix could not be binary, making it harder to be applied to AdderNets. Therefore, we only focus on $F(2 \times 2, 3 \times 3)$ in the following sections.

3. Method

3.1. Winograd Algorithm for AdderNet

As AdderNet and Winograd algorithm can all improve the efficiency of neual networks, we explore to combine the two techniques together to further reduce the computation cost.

In this section, we will introduce the vanilla form of Winograd algorithm on AdderNet. As shown in Equation (8), the Winograd algorithm consists of several parts of calculations, including pre-transformations for filters GgG^T, pre-transformations for inputs $B^T dB$, element-wise multiplications, and output transformations using matrix A. Since the calculations in input pre-transformations and output transformations only contain additions, we do not need to modify them. Therefore, we only replace the element-wise multiplications with ℓ_1-distance, the calculations can be reformulated as:

$$Y = A^T[-|\hat{g} \ominus [B^T dB]|]A. \tag{9}$$

A, G, and B are the same as those we introduced in Section 2.2. $\ominus$ represents element-wise minus operation. (Additions and minus operation are actually the same, since minus operation could be implemented by additions of complement.) And $|\cdot|$ represents the absolute operation for each element in the matrix.

Here we give a brief analysis of the complexity of Winograd algorithm for AdderNet. The calculation of Winograd algorithm consists of four parts: weight pre-transformations, input pre-transformations, element-wise minus and absolute operation of weights and output transformations. The transformation of weights could be calculated before deployment, so we do not take this part into account. We denote the shape of input features as (N, C_{in}, X_h, X_w), and the shape of weights as $(C_{out}, C_{in}, K_h, K_w)$. The input features could be divided into $N \times C_{in} \times \frac{X_h}{2} \times \frac{X_w}{2}$ groups for applying Winograd algorithm. Each group requires 3 additions since each column and row of matrix B has two non-zero values (1 or -1), which means the final $B^T dB$ results are the sum of four values. For the element-wise minus and absolute operation of weights and features, each group requires 16 additions, and there are $N \times C_{out} \times C_{in} \times \frac{X_h}{2} \times \frac{X_w}{2}$ groups in total. Since the results of addition and absolute operation need to be accumulated, the times of additions should be doubled, which results in $N \times C_{out} \times C_{in} \times \frac{X_h}{2} \times \frac{X_w}{2} \times 16 \times 2$ additions. The output features could be divided into $C_{out} \times X_h \times X_w$ groups and each group needs 8 additions since the matrix A has 3 non-zero values each column. The total additions of three parts is

$$N \times \frac{X_h}{2} \times \frac{X_w}{2} \times (C_{out} \times C_{in} \times 16 \times 2 + C_{in} \times 3 + C_{out} \times 8). \tag{10}$$

Since the values of C_{in} and C_{out} are generally dozens or hundreds, last two items can be ignored. Then the formula (10) becomes

$$N \times X_h \times X_w \times C_{out} \times C_{in} \times 8. \tag{11}$$

The total additions of original AdderNet are

$$N \times X_h \times X_w \times C_{in} \times C_{out} \times 9 \times 2. \tag{12}$$

Thus, the Winograd algorithm for AdderNet only requires about $\frac{4}{9}$ additions of original AdderNet.

However, since the absolute value is used in the calculation of AdderNet, the distributive law is not valid. So the Winograd form in Equation (9) is not equal to the original addition operation in Equation (1). Moreover, the accumulative absolute values put forward higher requirements for output transform matrix A. Let $X = -|\hat{g} \ominus [B^T dB]|$, and denote elements in X and Y as

$$X = \begin{bmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{bmatrix}, Y = \begin{bmatrix} y_0 & y_1 \\ y_2 & y_3 \end{bmatrix}.$$

We expand the equation $Y = A^T X A$ and get

$$\begin{aligned}
y_0 &= x_0 + x_1 + x_2 + x_4 + x_5 + x_6 + x_8 + x_9 + x_{10},\\
y_1 &= x_1 - x_2 - x_3 + x_5 - x_6 - x_7 + x_9 - x_{10} - x_{11},\\
y_2 &= x_4 + x_5 + x_6 - x_8 - x_9 - x_{10} - x_{12} - x_{13} - x_{14},\\
y_3 &= x_5 - x_6 - x_7 - x_9 + x_{10} + x_{11} - x_{13} + x_{14} + x_{15}.
\end{aligned}$$

We can find that the number of additions and that of minus operations in each equation is not the same. The magnitude of each x is usually similar. Since all elements in X are negative, the magnitude of output features Y is not consistent for each y_i. The unbalance of different positions obviously affect the performance of the network. In the next section, we will introduce our method to mitigate the influence of this two problem.

3.2. Optimal Transform Matrix

As discussed above, if we apply the Winograd algorithm for AdderNets, the overall computational complexity during the inference can be reduced. In this section, we explore new transform matrixes to solve the feature unbalanced problem. There are two requirements of the transform matrixes.

- The modified transform matrixes could balance the magnitude of all positions of output features Y.
- The output of Winograd algorithm transformed by modified matrixes should be equal to that of the original form in CNN.

The first requirement is to ensure the output of Winograd AdderNet have a similar magnitude which can be properly handled by the following layers (Batchnorm and ReLU). The second requirement is to remain the basic characteristic of conventional Winograd algorithm and make it universal to CNNs.

Theorem 1 *The general solution of the Winograd form $F(2,3)$ is*

$$Y = A^T[[GgG^T] \odot [B^T dB]]A.$$

and the matrixes are

$$A = \begin{bmatrix} \alpha_0 & -\alpha_0 c_0 \\ \beta_0 & -\beta_0 c_1 \\ \gamma_0 & -\gamma_0 c_2 \\ 0 & \delta_0 \end{bmatrix},$$

$$G = \begin{bmatrix} \frac{\alpha_1}{(c_1-c_0)(c_2-c_0)} & -\frac{\alpha_1 c_0}{(c_1-c_0)(c_2-c_0)} & \frac{\alpha_1 c_0^2}{(c_1-c_0)(c_2-c_0)} \\ \frac{\beta_1}{(c_0-c_1)(c_2-c_1)} & -\frac{\beta_1 c_1}{(c_0-c_1)(c_2-c_1)} & \frac{\beta_1 c_1^2}{(c_0-c_1)(c_2-c_1)} \\ \frac{\gamma_1}{(c_0-c_2)(c_1-c_2)} & -\frac{\gamma_1 c_2}{(c_0-c_2)(c_1-c_2)} & \frac{\gamma_1 c_2^2}{(c_0-c_2)(c_1-c_2)} \\ 0 & 0 & \delta_1 \end{bmatrix},$$

$$B = \begin{bmatrix} \frac{c_1c_2}{\alpha_0\alpha_1} & \frac{c_0c_2}{\beta_0\beta_1} & \frac{c_0c_1}{\gamma_0\gamma_1} & \frac{c_0c_1c_2}{\delta_0\delta_1} \\ \frac{c_1+c_2}{\alpha_0\alpha_1} & \frac{c_0+c_2}{\beta_0\beta_1} & \frac{c_0+c_1}{\gamma_0\gamma_1} & \frac{c_0c_1+c_0c_2+c_1c_2}{\delta_0\delta_1} \\ \frac{1}{\alpha_0\alpha_1} & \frac{1}{\beta_0\beta_1} & \frac{1}{\gamma_0\gamma_1} & \frac{c_0+c_1+c_2}{\delta_0\delta_1} \\ 0 & 0 & 0 & \frac{1}{\delta_0\delta_1} \end{bmatrix}.$$

in which c_0, c_1 and c_2 are arbitrary rational numbers, and α_i, β_i, γ_i and δ_i, $i = 0, 1$ are arbitrary real numbers.

Proof For the Winograd form F(2, 3) , denote input sequence y and filter sequence g with length two and three as $[y_0, y_1]$ and $[g_0, g_1, g_2]$, then the results of convolution operation $[d_0, d_1, d_2, d_3]$ would be

$$(y * g)(\tau) = \sum_{t, \tau - t > 0}^{3} y_t g_{\tau - t},$$

$$d_0 = (y*g)(0) = y_0 g_0, d_1 = (y*g)(1) = y_0 g_1 + y_1 g_0,$$
$$d_2 = (y*g)(2) = y_1 g_1 + y_2 g_0, d_3 = (y*g)(3) = y_1 g_2.$$

The results of the convolution can be derived from the product of two discrete sequence polynomials $y(n)$ and $g(n)$.

$$\begin{aligned}
&y(n) = y_0 + y_1 n, g(n) = g_0 + g_1 n + g_2 n^2,\\
&d(n) = y(n)g(n)\\
&= y_0 g_0 + (y_0 g_1 + y_1 g_0)n + (y_0 g_2 + y_1 g_1)n^2 + y_1 g_2 n^3\\
&= (y*g)(0) + (y*g)(1)n + (y*g)(2)n^2 + (y*g)(3)n^3.
\end{aligned}$$

The coefficients of $n^i, i = 0...3$ term in polynomials $d(n)$ are actually the results of the ith term in convolution, so we can get the results of convolution by calculating the polynomials. In order to solve the polynomial coefficients, we need to construct an equivalent transformation for $d(n)$. We divide $d(n)$ into two parts, mutual prime polynomial $M(n)$ and remainder $d'(n)$. The order of $M(n)$ is the same as that of $d(n)$ so that the coefficient of $M(n)$ is Then the problem of solving polynomial coefficients is converted into the problem of solving remainders. Denote three relatively prime polynomials as $m_0(n) = a_0 n + b_0$, $m_1(n) = a_1 n + b_1$, $m_2(n) = a_2 n + b_2$, in which a_0, a_1, a_2 are arbitrary non-zero integers and b_0, b_1, b_2 are arbitrary integers. Then we get

$$M(n) = (a_0 n + b_0)(a_1 n + b_1)(a_2 n + b_2) \quad (13)$$
$$= a_0 a_1 a_2 (n + c_0)(n + c_1)(n + c_2) \quad (14)$$
$$c_0 = b_0/a_0, c_1 = b_1/a_1, c_2 = b_2/a_2. \quad (15)$$

where c_0, c_1 and c_2 are arbitrary rational numbers, and

$$d(n) = tM(n) + d'(n), t = y_1 g_2. \quad (16)$$

We use Extended Euclidean algorithm to solve the inverse elements $[(\frac{M(n)}{m_i(n)})^{-1}]_{m_i(n)}$. The results are $\frac{1}{(c_1-c_0)(c_2-c_0)}$, $\frac{1}{(c_0-c_1)(c_2-c_1)}$, $\frac{1}{(c_0-c_2)(c_1-c_2)}$ for $m_0(n)$, $m_1(n)$, $m_2(n)$, respectively. According to Chinese remainder theorem,

$$d'(n) = \sum_{i=0}^{2} d'_i(n) \frac{M(n)}{m_i(n)} [(\frac{M(n)}{m_i(n)})^{-1}]_{m_i(n)}. \quad (17)$$

Substituted into Equation (16) and then we get

$$d'_0(n) = \frac{(y_0 - c_0y_1)(c_0^2g_2 - c_0g_1 + g_0)}{(c_1 - c_0)(c_2 - c_0)}, \quad (18)$$

$$d'_1(n) = \frac{(y_0 - c_1y_1)(c_1^2g_2 - c_1g_1 + g_0)}{(c_0 - c_1)(c_2 - c_1)}, \quad (19)$$

$$d'_2(n) = \frac{(y_0 - c_2y_1)(c_2^2g_2 - c_2g_1 + g_0)}{(c_1 - c_2)(c_0 - c_2)}, \quad (20)$$

$$t = y_1g_2. \quad (21)$$

and the input transformation

$$\begin{aligned}d(n) &= tn^3 + [d'_0(n) + d'_1(n) + d'_2(n) + (c_0 + c_1 + c_2)t]n^2 \\ &+ [(c_1 + c_2)d'_0(n) + (c_0 + c_2)d'_1(n) + (c_0 + c_1)d'_2(n) \\ &+ (c_0c_1 + c_0c_2 + c_1c_2)t]n \\ &+ (c_1c_2d'_0(n) + c_0c_2d'_1(n) + c_0c_1d'_2(n) + c_0c_1c_2t].\end{aligned}$$

From Equation (18)-(21), we have

$$\begin{bmatrix}\alpha_0\alpha_1d'_0(n)\\ \beta_0\beta_1d'_1(n)\\ \gamma_0\gamma_1d'_2(n)\\ \delta_0\delta_1t\end{bmatrix} = A\cdot\begin{bmatrix}y_0\\y_1\end{bmatrix}\odot G\cdot\begin{bmatrix}g_0\\g_1\\g_2\end{bmatrix}, A = \begin{bmatrix}\alpha_0 & -\alpha_0c_0\\ \beta_0 & -\beta_0c_1\\ \gamma_0 & -\gamma_0c_2\\ 0 & \delta_0\end{bmatrix}.$$

Since the division of $d'_0(n)$, $d'_1(n)$, $d'_2(n)$and t is arbitrary, we add coefficients α_i, β_i, γ_i and δ_i, $i = 0, 1$ to maintain the generability of the solution. Moreover, the order of rows in matrix A and matrix G can be swapped simultaneously, and the corresponding column in matrix B should also be swapped. Then the coefficients of polynomial $s(x)$ can be represented with$[\alpha_0\alpha_1d'_0(n), \beta_0\beta_1d'_1(n), \gamma_0\gamma_1d'_2(n), \delta_0\delta_1t]$

$$\begin{bmatrix}d_0\\d_1\\d_2\\d_3\end{bmatrix} = B\cdot\begin{bmatrix}\alpha_0\alpha_1d'_0(n)\\ \beta_0\beta_1d'_1(n)\\ \gamma_0\gamma_1d'_2(n)\\ \delta_0\delta_1t\end{bmatrix}.$$

Now we get the convolution result $x = B[Gg\odot Ay]$. For the correlation operation we need in CNN, s would be the input and h would be the output, so we have $y = A^T[Gg\odot B^Td]$. Then the 1-D result is nested to itself to obtain the 2-D result $Y = A^T[GgG^T\odot B^TdB]A$. □

In order to reduce the amount of calculation during the inference process, the elements in matrix A should be chosen from 0, 1, -1 to avoid shift or multiplication operations. So c_0, c_1 and c_2 could only be chosen from 0, -1 and 1, one of each. Without losing generality, we set $\alpha_1 = -1$, $\delta_0 = -1$ and other coefficients to 1, then we get the standard Winograd algorithm for convolution like Equation (6)-(7).

Denote that the number of $+1$ and -1 in matrix A of column i is p_i and $k - p_i$, in which k represents the total number of non-zero elements. According to the previous results, we have $k = 3$ in all columns of matrix A.

Theorem 2 *$\forall i, j, m, n$, the number of additions and minus operations in the calculations of output feature $Y_{i,j}$ and $Y_{m,n}$ would be equal respectively if and only if $p_i = p_j = p_m = p_n$.*

Proof $\forall i, j, m, n$, if the number of additions of $Y_{i,j}$ and $Y_{m,n}$ is the same, then we have

$$p_ip_j + (k - p_i)(k - p_j) = p_mp_n + (k - p_m)(k - p_n)$$

$$k[(p_i + p_j) - (p_m + p_n)] = 2(p_ip_j - p_mp_n)$$

Since i, j, m, n are all arbitrary, to ensure the equation always established, we have $(p_i + p_j) - (p_m + p_n) \equiv 0$ and $p_ip_j - p_mp_n \equiv 0$. That is to say, $p_i = p_m, p_j = p_n$ or $p_i = p_n, p_j = p_m$. Since i, j, m, n are arbitrary, actually we have $p_i = p_j = p_m = p_n$. □

Based on the conclusions we deduce above, we can modify the coefficients α_i, β_i, γ_i and δ_i, $i = 0, 1$ to let the number of $+1$ and -1 in every column of matrix A keep the same. It is easy to draw the conclusion that there are only four matrixes $A_i, i = 0...3$ which meets our requirements.

$$A_0^T = \begin{bmatrix}-1 & 1 & 1 & 0\\ 0 & 1 & -1 & 1\end{bmatrix}, A_1^T = \begin{bmatrix}-1 & -1 & 1 & 0\\ 0 & -1 & -1 & 1\end{bmatrix},$$

$$A_2^T = \begin{bmatrix}1 & -1 & -1 & 0\\ 0 & -1 & 1 & -1\end{bmatrix}, A_3^T = \begin{bmatrix}1 & 1 & -1 & 0\\ 0 & 1 & 1 & -1\end{bmatrix}.$$

Correspondingly, we can get the matrixes $G_i, i = 0...3$.

3.3. Training with L2-to-L1 Distance

Although we can solve the feature unbalanced problem by optimal transform matrix proposed in the above section, the winograd form of AdderNet is still not equal to its original form. Here we introduce a training method to mitigate the gap of Winograd AdderNet and original AdderNet. According to the training strategy described in AdderNet (Chen et al., 2020), the output of ℓ_2-AdderNet can be regarded as a linear transformation of that in the conventional CNN. For Winograd algorithm, ℓ_2-norm also means a better appoximation of multiplication, since ℓ_2 is composed of squares and multications, i.e.,

$$\begin{aligned}Y &= A^T[-([GgG^T]\ominus[B^TdB])^2]A\\ &= A^T[(2[GgG^T]\odot[B^TdB]\ominus[GgG^T]^2\ominus[B^TdB]^2)]A\end{aligned}$$

However, ℓ_1-norm is more hardware friendly than ℓ_2-norm since it requires no multiplications. To improve the network accuracy and maintain the hardware efficiency, we propose to train AdderNet after applying the Winograd algorithm in an ℓ_2-to-ℓ_1 distance paradigm.

In the inference process, the adder layer is formulated as

$$t = F(i, j, k, t) - X(m + i, n + j, k). \quad (22)$$

Table 1. Results on CIFAR-10 and CIFAR-100 datasets

Model	Method	#Mul	#Add	CIFAR-10 Accuracy	CIFAR-100 Accuracy
ResNet-20	Winograd CNN	19.40M	19.84M	92.25%	68.14%
	AdderNet	-	80.74M	91.84%	67.60%
	Winograd AdderNet	-	39.24M	91.56%	67.96%
ResNet-32	Winograd CNN	31.98M	32.74M	93.29%	69.74%
	AdderNet	-	137.36M	93.01%	69.02%
	Winograd AdderNet	-	64.72M	92.34%	69.87%

$$Y(m,n,t) = -\sum_{i,j,k}(|t|)^p. \tag{23}$$

The backward process is formulated as

$$\frac{\partial Y(m,n,t)}{\partial X(m+i,n+j,k)} = p \cdot t^{p-1} \cdot sign(t). \tag{24}$$

$$\frac{\partial Y(m,n,t)}{\partial F(i,j,k,t)} = p \cdot (-t)^{p-1} \cdot sign(-t). \tag{25}$$

During the training process, we gradually reduce the exponent p from 2 to 1. Then the forward and backward process finally calculated as followed,

$$Y(m,n,t) = -\sum_{i,j,k}|t| \tag{26}$$

$$\frac{\partial Y(m,n,t)}{\partial X(m+i,n+j,k)} = sign(t). \tag{27}$$

$$\frac{\partial Y(m,n,t)}{\partial F(i,j,k,t)} = sign(-t). \tag{28}$$

To ensure the continuity of approximation, we do not apply the ℓ_2 gradients for F and HardTanh gradients for X as AdderNet. The adaptive learning rate in Equation (5) proposed in AdderNet is adapted to stabilize the training process.

There are several strategies to reduce the exponent p. We denote the step of reduction as s.

- **Training until converge and then reducing p.** Train network with cosine annealing learning rate until the learning rate close to 0. Then reduce p with a certain step s and restart the training process.
- **Reducing p during the converge process.** Reduce p every k epoch of the training process and the step s is set to $\frac{k}{epochs}$.

We will give detailed analysis of different strategies in the experimental results.

Since the kernel transformation $\hat{g} = GgG^T$ in Winograd algorithm is not equivalent for AdderNet, we do not perform weight transformation during the training process. Instead, we directly train the weights in the Winograd domain. We also compare different ways to deal with weights in the ablation study part.

4. Experiments

Here we conduct experiments to show the effectiveness of our proposed Winograd algorithm for AdderNet. The experiments are done on several commonly used datasets, including MNIST, CIFAR and ImageNet. We also make some ablation studies and give visualization of features to provide insights of our methods. All experiments are made via PyTorch on NVIDIA Tesla V100 GPU. For all experiments, we use the transform matrix A_0 and G_0, and other A_i and G_i matrixes can achieve the similar results.

4.1. Classification

Experiments on MNIST First we evaluate our method on the MNIST dataset. To facilitate Winograd algorithm, we replace 5×5 layers with 3×3 layers in the original LeNet-5-BN (LeCun et al., 1998). The detailed network structure is shown in the supplemental material. The learning rate is set to 0.1 at the beginning and decay with the cosine function in the following 100 epochs. We use SGD optimizer with momentum as 0.9, and the batch size is set as 256.

The original AdderNet achieves a 99.28% accuracy while the Winograd AdderNet achieves 99.19%. Meanwhile, the Winograd AdderNet requires only 401.1M additions instead of 746.8M additions required by original AdderNet.

Experiments on CIFAR We also evaluate Winograd AdderNet on the CIFAR dataset, including CIFAR-10 and CIFAR-100. The data settings are the same as (He et al., 2016). The initial learning rate is set to 0.1 and then decays with a cosine learning rate schedule. The model is trained for 800 epochs and the training batch size is 256. The hyper-parameter η in Equation (5) is set to 0.1.

To make fair comparison, we follow the settings in (Chen et al., 2020) to set the first and last layers as full-precision convolutional layers. The results are shown in table 1 and the results of CNN and AdderNet are from (Chen et al., 2020). We only count the additions of adder part instead of the whole neural network. At the cost of little accuracy loss, the number of additions is reduced by more than 50%.

Experiments on ImageNet ImageNet is a large scale vision dataset which consists of 224×224 pixel RGB images. We train ResNet-18 follow the original data settings in (He et al., 2016). We train Winograd AdderNet on 8 GPUs with

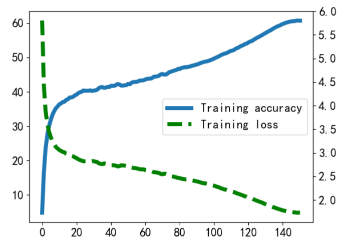


Figure 2. Training accuracy and training loss of Winograd AdderNet ResNet-18 on ImageNet

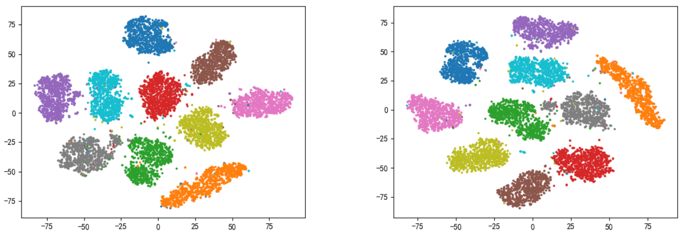
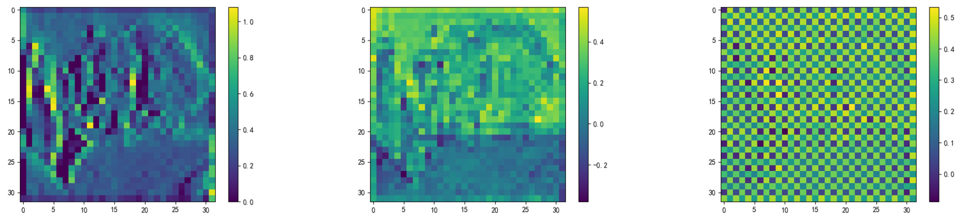


Figure 3. Dimension reduction results of features in Winograd for AdderNet (left) and original AdderNet (right). The visualization results are very close, which means that Winograd for AdderNet attracts the similar features to original AdderNet



(a) Input features (b) Output features with modified A (c) Output features with original A

Figure 4. Comparison of the feature heatmaps under different matrix A. Without the modified matrix A, there is a obvious grid in the heatmap shown in figure 4(c).

batch size 512, and the total training epochs are 150. The weight decay is set as 0.0001 and the momentum is 0.9. The hyper-parameter η is set to 0.05 for Winograd AdderNet. Experimental results are shown in Figure 2 and we use the AdderNet baseline from their paper (Chen et al., 2020). Winograd AdderNet achieves a 66.2% top-1 accuracy and an 86.8% top-5 accuracy in ResNet-18, which is slightly less than AdderNet (67.0% top-1/87.6% top-5). If we extend the training epochs to 250, Winograd AdderNet achieved 66.5% top-1 accuracy while AdderNet got no improvement. Besides, Winograd AdderNet uses only 1.72G adder operations compared with 3.39G in AdderNet.

4.2. Ablation Study

In this part, we evaluate the effectiveness of our proposed transform matrix and ℓ_2-to-ℓ_1 training method. All experiments in this section is done with ResNet-18 network on the CIFAR-10 dataset.

First we compare different methods to reduce parameter p in the ℓ_2-to-ℓ_1 training strategy. The total training epochs are set to 800 to make fair comparison. The evaluation results are shown in Table 3. We can find that reducing p during the converge process with $p = 35$ is the best. We provide the curves of training loss (green lines) and accuracy (blue lines) in the upper figure of Figure 5. When p=140, the training process is unstable with an accuracy drop (-0.12% as shown in Table 3). In addition, the lower figure of Figure 5 shows the values of weights using norm reduction with different settings. It is obvious that when p=35, the curve of weight norm is the most closed to that of using only ℓ_2-norm, i.e., the network trained using our method can successfully approximate the ℓ_2-norm wino AdderNet. Thus, we set p=35 in our experiments and obtain better performance.

We also show the comparison of three ways to deal with kernel transformation. The first way is the same as Winograd algorithm for convolution layers. We apply kernel transformation (KT) to weights during every inference process and update the origin 3×3 kernel. The second way and the third way are to train the network in the Winograd domain. For these two ways, we initialize weights with normal distribution initialization for the 4×4 Winograd kernel and for the 3×3 original Adder kernel, respectively. The results are shown in Table 4. Training with kernel transform has the worst performance, since the inconsistent transform makes the training harder. Other two ways have similar results, so we recommend to directly initialize Winograd kernel due to its convenience.

Next we evaluate the effectiveness of our proposed methods. The results are shown in Table 5. Without any modification, the original Winograd algorithm only achieves 83.87%. Modified transform matrix and our ℓ_2-to-ℓ_1 training strategy brings 4.38% and 5.38% accuracy improvement, respectively. Finally the combination achieves 91.56%, which is comparable with that of AdderNet.

4.3. Visualization

To intuitively perceive the effectiveness of Winograd for AdderNet, we visualize the features. We acquire the features

Table 2. FPGA Simulation Results of original AdderNet and Winograd AdderNet

Method	Module	#cycle	Hardware Resource	Total Energy Consuming (Equivalent)[1]
original AdderNet	total	7062	7130	50.4M
Winograd AdderNet	padding	900	31	0.03M
	input transform	3136	433	1.36M
	calculation	3140	6900	21.7M
	output transform	3136	309	0.97M
	total	-	7673	**24.0M**

[1] Since the ratio of hardware resource usage is close to 100%, we use the hardware resource overhead to approximate equivalent power consumption.

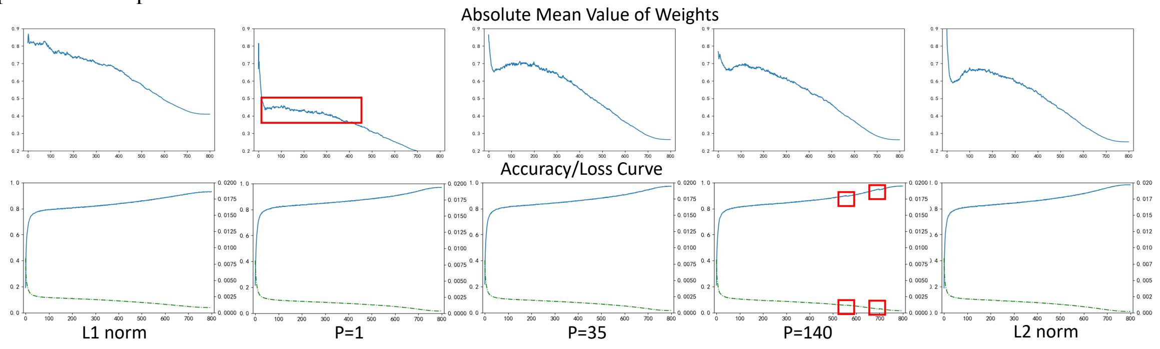


Figure 5. Upper: Trending of bsolute mean value of weights during training process. Lower: Training loss and accuracy.

Table 3. Ablation Study on the Reduction Method of p

Method	Accuracy
Training until converge	89.24
Reducing during converge with $p = 1$	90.94
Reducing during converge with $p = 35$	**91.56**
Reducing during converge with $p = 140$	91.44

Table 4. Ablation Study on the Kernel Transformation

Method		Accuracy
Training w/ KT		89.19
Training w/o KT	Init Winograd kernel	**91.56**
	Init adder kernel and transform	91.28

of the last adder layer in our modified LeNet-5-BN, and then use t-SNE method to reduce the dimension to 2. The result of original AdderNet is shown in the left of Figure 3 and the result of Winograd for AdderNet is shown in the right. From the results, we can find that the two are extremely close, which means that Winograd for AdderNet attracts the similar features to original AdderNet.

Besides, we visualize the output features of Winograd Adder and original Adder layer, to intuitively display the effect of modifying matrix A. From Figure 4, we can find that output features with original A show a distinct grid while output features with modified A do not show this phenomenon.

4.4. FPGA simulation

To evaluation the energy efficiency of our method in the runtime, we implement the Winograd algorithm for AdderNet and original AdderNet on FPGA.The designed parallelism of calculation is 256, which means that 16 input channels and 16 output channels are calculated simultaneously.

Table 5. Ablation Study on Our Proposed Methods

Mod A	ℓ_2-to-ℓ_1-norm	CIFAR-10	CIFAR-100
		83.87%	54.72%
	$\surd$	88.25%	62.00%
$\surd$		89.25%	62.83%
$\surd$	$\surd$	**91.56%**	**67.96%**

We take a single layer with input shape $(N, C_{in}, X_h, X_w) = (1, 16, 28, 28)$ and kernel shape $(C_{out}, C_{in}, K_w, K_h) = (16, 16, 3, 3)$ as an example. The comparison of Winograd AdderNet and original AdderNet is shown in Table 2. We can find that Winograd AdderNet requires only $24.0/50.4 \approx 47.6\%$ energy consuming of original AdderNet. As we analysed in Section 3.1, the theoretical cost of Winograd AdderNet is 45.4% of that of original AdderNet with $C_{in} = 16$ and $C_{out} = 16$. So our implementation validates this theoretical result. Moreover, with the pipeline technique, Winograd AdderNet may achieve about 50% latency reduction (estimated).

5. Conclusion

In this paper, we propose the Winograd algorithm for AdderNet. We replace the element-wise multiplications in the Winograd equation with additions to further reduce the energy costs of CNN. To mitigate the accuracy loss brought by the replacement, we develop a set of new transform matrixes to balance output features and introduce an ℓ_2-to-ℓ_1 training method for the Winograd AdderNet paradigm. As a result, the proposed method reduces about 52.4% energy consumption on our FPGA simulation while achieving similar performance with the original AdderNet, which would have a excellent prospects in future hardware design.

References

Chen, H., Wang, Y., Xu, C., Shi, B., Xu, C., Tian, Q., and Xu, C. Addernet: Do we really need multiplications in deep learning? In *CVPR*, pp. 1468–1477, 2020.

Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., and Bengio, Y. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

Dally, W. High-performance hardware for machine learning. *NIPS Tutorial*, 2, 2015.

Guo, S., Yan, Z., Zhang, K., Zuo, W., and Zhang, L. Toward convolutional blind denoising of real photographs. In *CVPR*, pp. 1712–1722, 2019.

Han, K., Wang, Y., Tian, Q., Guo, J., Xu, C., and Xu, C. Ghostnet: More features from cheap operations. In *CVPR*, pp. 1580–1589, 2020.

Han, S., Pool, J., Tran, J., and Dally, W. J. Learning both weights and connections for efficient neural networks. *arXiv preprint arXiv:1506.02626*, 2015.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25: 1097–1105, 2012.

Lavin, A. and Gray, S. Fast algorithms for convolutional neural networks. In *CVPR*, pp. 4013–4021, 2016.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Lin, X., Zhao, C., and Pan, W. Towards accurate binary convolutional neural network. In *Advances in neural information processing systems*, pp. 345–353, 2017.

Liu, H., Simonyan, K., and Yang, Y. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018a.

Liu, X., Pool, J., Han, S., and Dally, W. J. Efficient sparse-winograd convolutional neural networks. *arXiv preprint arXiv:1802.06367*, 2018b.

Liu, Z., Shen, Z., Savvides, M., and Cheng, K.-T. Reactnet: Towards precise binary neural network with generalized activation functions. *arXiv preprint arXiv:2003.03488*, 2020.

Mathieu, M., Henaff, M., and LeCun, Y. Fast training of convolutional networks through ffts. *arXiv preprint arXiv:1312.5851*, 2013.

Qiu, J., Wang, J., Yao, S., Guo, K., Li, B., Zhou, E., Yu, J., Tang, T., Xu, N., Song, S., et al. Going deeper with embedded fpga platform for convolutional neural network. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 26–35, 2016.

Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pp. 525–542. Springer, 2016.

Ren, D., Zuo, W., Hu, Q., Zhu, P., and Meng, D. Progressive image deraining networks: A better and simpler baseline. In *CVPR*, pp. 3937–3946, 2019.

Song, D., Wang, Y., Chen, H., Xu, C., Xu, C., and Tao, D. Addersr: Towards energy efficient image super-resolution. *arXiv preprint arXiv:2009.08891*, 2020.

Tan, M. and Le, Q. V. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

Tan, M., Pang, R., and Le, Q. V. Efficientdet: Scalable and efficient object detection. In *CVPR*, pp. 10781–10790, 2020.

Tao, A., Sapra, K., and Catanzaro, B. Hierarchical multi-scale attention for semantic segmentation. *arXiv preprint arXiv:2005.10821*, 2020.

Winograd, S. *Arithmetic complexity of computations*, volume 33. Siam, 1980.

Xu, Y., Xu, C., Chen, X., Zhang, W., Xu, C., and Wang, Y. Kernel based progressive distillation for adder neural networks. *arXiv preprint arXiv:2009.13044*, 2020.

Yan, D., Wang, W., and Chu, X. Optimizing batched winograd convolution on gpus. In *Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 32–44, 2020.

Zhang, H. and Patel, V. M. Densely connected pyramid dehazing network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3194–3203, 2018.