Universal Adder Neural Networks

Hanting Chen, Yunhe Wang, Member, IEEE, Chang Xu, Member, IEEE, Chao Xu, Chunjing Xu, and Tong Zhang, Fellow, IEEE

Abstract—Compared with cheap addition operation, multiplication operation is of much higher computation complexity. The widely-used convolutions in deep neural networks are exactly cross-correlation to measure the similarity between input feature and convolution filters, which involves massive multiplications between float values. In this paper, we present adder networks (AdderNets) to trade these massive multiplications in deep neural networks, especially convolutional neural networks (CNNs), for much cheaper additions to reduce computation costs. In AdderNets, we take the $l_1$-norm distance between filters and input feature as the output response. We first develop a theoretical foundation for AdderNets, by showing that both the single hidden layer AdderNet and the width-bounded deep AdderNet with ReLU activation functions are universal function approximators. An approximation bound for AdderNets with a single hidden layer is also presented. We further analyze the influence of this new similarity measure on the optimization of neural network and develop a special training scheme for AdderNets. Based on the gradient magnitude, an adaptive learning rate strategy is proposed to enhance the training procedure of AdderNets. AdderNets can achieve a 75.7% Top-1 accuracy and a 92.3% Top-5 accuracy using ResNet-50 on the ImageNet dataset without any multiplication in the convolutional layer.

Index Terms—Efficient network, deep learning, computer vision, energy consumption.

1 INTRODUCTION

Given the advent of Graphics Processing Units (GPUs), deep convolutional neural networks (CNNs) with billions of floating number multiplications could receive speed-ups and make important strides in a large variety of computer vision tasks, e.g., image classification [1], segmentation [4], and human face verification [5]. However, the high-power consumption of these high-end GPU cards (e.g., 250W+ for GeForce RTX 2080 Ti) has blocked modern deep learning systems from being deployed on mobile devices, e.g., smartphone, camera, and watch. Existing GPU cards are far from svelte and cannot be easily mounted on mobile devices. Though the GPU itself only takes up a small part of the card, we need many other hardware for supports, e.g., memory chips, power circuitry, voltage regulators and other controller chips. It is therefore necessary to study efficient deep neural networks that can run with affordable computation resources on mobile devices.

Addition, subtraction, multiplication and division are the four most basic operations in mathematics. It is widely known that multiplication is slower than addition, but most of the computations in deep neural networks are multiplications between float-valued weights and float-valued activations during the forward inference. There are thus many papers on how to trade multiplications for additions, to speed up deep learning. The seminal work [6] proposed BinaryConnect to force the network weights to be binary (e.g., -1 or 1), so that many multiply-accumulate operations can be replaced by simple accumulations. After that, Hubara et al. [7] proposed BNNs, which binarized not only weights but also activations in convolutional neural networks at run-time. Moreover, Rastegari et al. [8] introduced scale factors to approximate convolutions using binary operations and outperform [7], [8] by large margins. Zhou et al. [9] utilized low bit-width gradient to accelerate the training of binarized networks. Cai et al. [10] proposed an half-wave Gaussian quantizer for forward approximation, which achieved much closer performance to full precision networks.

Though binarizing filters of deep neural networks significantly reduces the computation cost, the original recognition accuracy often cannot be preserved. In addition, the training procedure of binary networks is not stable and usually requests a slower convergence speed with a small learning rate. Convolutions in classical CNNs are actually cross-correlation to measure the similarity of two inputs. Researchers and developers are used to taking convolution as a default operation to extract features from visual data, and introduce various methods to accelerate the convolution, even if there is a risk of sacrificing network capability. But there is hardly no attempt to replace convolution with another more efficient similarity measure that is better to only involve additions. In fact, additions are of much lower computational complexities than multiplications. Thus, we are motivated to investigate the feasibility of replacing mul-
Elliptical by additions in convolutional neural networks.

In this paper, we propose adder networks that maximize the use of addition while abandoning convolution operations. Given a series of small template as "filters" in the neural network, $\ell_1$-distance could be an efficient measure to summarize absolute differences between the input signal and the template as shown in Figure 1. Since subtraction can be easily implemented through addition by using its complement code, $\ell_1$-distance could be a hardware-friendly measure that only has additions, and naturally becomes an efficient alternative of the convolution to construct neural networks. To give a theoretical guarantee for AdderNets, we prove that both the single hidden layer AdderNet and the width-bounded AdderNet can approximate any Lebesgue integrable function in a compact set. This result is comparable to the universal approximation results for traditional neural networks. We also present a approximation bound for AdderNets with a single hidden layer. To optimize the AdderNet effectively, we design the $\ell_2$ to $\ell_1$ training scheme and the adaptive learning rate scaling to ensure sufficient updates of the templates and a better network convergence. The proposed AdderNets are deployed on several benchmarks, and experimental results demonstrate that AdderNets can achieve comparable recognition accuracy to conventional CNNs.

A preliminary version of this work has been presented earlier [11]. The present work improves the initial version significantly. First, we propose the universal approximation theorem of two-layer AdderNet and width-bounded deep AdderNet, which lays the foundation for AdderNets in various deep learning-based applications. Second, we introduce a general $l_p$-AdderNet, where $1 \leq p \leq 2$. By gradually reducing $p$ from 2 to 1 during the iterations, the training difficulty of $l_1$-AdderNet could be reduced. In addition, extended experiments are conducted to show the approximation capacity of AdderNets and the improvement of the new training scheme.

This paper is organized as follows. Section 2 revisits the related works on neural network compression. Section 3 proposes the AdderNets and discusses the universal approximation theorem of AdderNets. Section 4 introduces the optimization scheme of AdderNets using $\ell_2$ to $\ell_1$ training and adaptive learning rate scaling. Section 5 conducts experiments to show the approximation capacity and effectiveness of the proposed AdderNets on toy examples and classification tasks, respectively. Section 6 concludes this paper.

## 2 Related works

To reduce the computational complexity of convolutional neural networks, a number of works have been proposed for eliminating useless calculations.

Pruning based methods aims to remove redundant weights to compress and accelerate the original network. Denton et.al [12] decomposed weight matrices of fully-connected layers into simple calculations by exploiting singular value decomposition (SVD). Han et.al [13] proposed discarding subtle weights in pre-trained deep networks to omit their original calculations without affecting the performance. Wang et.al [14] further converted convolution filters into the DCT frequency domain and eliminated more floating number multiplications. In addition, Hu et.al [15] discarded redundant filters with less impacts to directly reduce the computations brought by these filters. Luo et.al [16] discarded redundant filters according to the reconstruction error. Hu et.al [17] proposed dubbed Robust Dynamic Inference Networks (RDI-Nets), which allows for each input to adaptively choose one of the multiple output layers to output its prediction. Wang et.al [18] proposed a E2-Training method, which can train deep neural networks with over 80% energy savings.

Instead of directly reducing the computational complexity of a pre-trained heavy neural network, lot of works focused on designing novel blocks or operations to replace the conventional convolution filters. Howard et.al [19] designed MobileNet, which decompose the conventional convolution filters into the point-wise and depth-wise convolution filters with much fewer FLOPs. Zhang et.al [20] combined group convolutions [21] and a channel shuffle operation to build efficient neural networks with fewer computations. Wu et.al [22] presented a parameter-free "shift" operation with zero flop and zero parameter to replace conventional filters and largely reduce the computational and storage cost of CNNs. Wang et.al [23] developed versatile convolution filters to generate more useful features utilizing fewer calculations and parameters. Xu et.al [24] proposed perturbative neural networks to replace convolution and instead computes its response as a weighted linear combination of non-linearly activated additive noise perturbed inputs. Han et.al [25] proposed GhostNet to generate more features from cheap operations and achieve the state-of-the-art performance on lightweight architectures.

Besides eliminating redundant weights or filters in deep convolutional neural networks, Hinton et.al [26] proposed the knowledge distillation (KD) scheme, which transfer useful information from a heavy teacher network to a portable student network by minimizing the Kullback-Leibler divergence between their outputs. Besides mimic the final outputs of the teacher networks, Romero et.al [27] exploit the hint layer to distill the information in features of the teacher network to the student network. You et.al [28] utilized multiple teachers to guide the training of the student network and achieve better performance. Yin et.al [29] regarded the relationship between features from two layers in the teacher network as a novel knowledge and introduced the FSP (Flow of Solution Procedure) matrix to transfer this kind of information to the student network.

Nevertheless, the compressed networks using these algorithms still contain massive multiplications, which costs enormous computation resources. As a result, subtractions or additions are of much lower computational complexities when compared with multiplications. However, they have not been widely investigated in deep neural networks, especially in the widely used convolutional networks. Therefore, we propose to minimize the numbers of multiplications in deep neural networks by replacing them with subtractions or additions.
there are some differences in their outputs. The output of the filters and features can be efficiently computed.

Addition is the major operation in $\ell_1$ distance, as a weighted summation of values in the input feature map, can be positive or negative, but the output of an adder filter is always negative. Hence, we resort to batch normalization for help, and the output of adder layers will be normalized to an appropriate range and all the activation functions used in conventional CNNs can then be used in the proposed AdderNets. Although the batch normalization layer involves multiplications, its computational cost is significantly lower than that of the convolutional layers and can be omitted. Considering a convolutional layer with a filter $F \in \mathbb{R}^{d \times d \times c_{in} \times c_{out}}$, an input $X \in \mathbb{R}^{H \times W \times c_{in}}$, and an output $Y \in \mathbb{R}^{H' \times W' \times c_{out}}$, the computation complexity of convolution and batch normalization is $O(d^2 c_{in} c_{out} H W)$ and $O(c_{out} H' W')$, respectively. In practice, given an input channel number $c_{in} = 512$ and a kernel size $d = 3$ in ResNet [30], we have $\frac{d^2 c_{in} c_{out} H W}{W' c_{out}} \approx 4068$. Since batch normalization layer has been widely used in the state-of-the-art convolutional neural networks, we can simply upgrade these networks into AddNets by replacing their convolutional layers into adder layers to speed up the inference and reduces the energy cost.

Intuitively, Eq. (1) has a connection with template matching [31] in computer vision, which aims to find the parts of an image that match the template. $F$ in Eq. (1) actually works as a template, and we calculate its matching scores with different regions of the input feature $X$. Since various metrics can be utilized in template matching, it is natural that $\ell_1$ distance can be utilized to replace the cross-correlation in Eq. (1). Note that there are few works also focus on utilizing different metrics in deep networks. Wang et al. [32] aims to achieve high performance by employing complex metrics. Zhang et al. [33] proposed to use $\ell_{inf}$ distance as basic operation and directly provides a theoretical guarantee of the certified robustness. In contrast, we focus on the $\ell_1$ distance to minimize the energy consumption.

3 Adder Networks

Consider a filter $F \in \mathbb{R}^{d \times d \times c_{in} \times c_{out}}$ in an intermediate layer of the deep neural network, where kernel size is $d$, input channel is $c_{in}$ and output channel is $c_{out}$. The input feature is defined as $X \in \mathbb{R}^{H \times W \times c_{in}}$, where $H$ and $W$ are the height and width of the feature, respectively. The output feature $Y$ indicates the similarity between the filter and the input feature, 

$$Y(m, n, t) = \sum_{i=0}^{d} \sum_{j=0}^{d} S(X(m+i, n+j, k), F(i, j, k, t)), \quad (1)$$

where $S(\cdot, \cdot)$ is a pre-defined similarity measure. If cross-correlation is taken as the metric of distance, i.e., $S(x, y) = x \times y$, Eq. (1) becomes the convolution operation. Eq. (1) can also implies the calculation of a fully-connected layer when $d = 1$. In fact, there are many other metrics to measure the distance between the filter and the input feature. However, most of these metrics involve multiplications, which bring in more computational cost than additions.

3.1 Adder Networks

We are therefore interested in deploying distance metrics that maximize the use of additions. $\ell_1$ distance calculates the sum of the absolute differences of two points’ vector representations, which contains no multiplication. Hence, by calculating $\ell_1$ distance between the filter and the input feature, Eq. (1) can be reformulated as:

$$Y(m, n, t) = -\sum_{i=0}^{d} \sum_{j=0}^{d} c_{in} |X(m+i, n+j, k) - F(i, j, k, t)|. \quad (2)$$

Addition is the major operation in $\ell_1$ distance measure, since subtraction can be easily reduced to addition by using complement code. With the help of $\ell_1$ distance, similarity between the filters and features can be efficiently computed.

Although both $\ell_1$ distance Eq. (2) and cross-correlation in Eq. (1) can measure the similarity between filters and inputs, there are some differences in their outputs. The output of a convolution filter, as a weighted summation of values in the input feature map, can be positive or negative, but the output of an adder filter is always negative. Hence, we resort to batch normalization for help, and the output of adder layers will be normalized to an appropriate range and all the activation functions used in conventional CNNs can then be used in the proposed AdderNets. Although the batch normalization layer involves multiplications, its computational cost is significantly lower than that of the convolutional layers and can be omitted. Considering a convolutional layer with a filter $F \in \mathbb{R}^{d \times d \times c_{in} \times c_{out}}$, an input $X \in \mathbb{R}^{H \times W \times c_{in}}$, and an output $Y \in \mathbb{R}^{H' \times W' \times c_{out}}$, the computation complexity of convolution and batch normalization is $O(d^2 c_{in} c_{out} H W)$ and $O(c_{out} H' W')$, respectively. In practice, given an input channel number $c_{in} = 512$ and a kernel size $d = 3$ in ResNet [30], we have $\frac{d^2 c_{in} c_{out} H W}{W' c_{out}} \approx 4068$. Since batch normalization layer has been widely used in the state-of-the-art convolutional neural networks, we can simply upgrade these networks into AddNets by replacing their convolutional layers into adder layers to speed up the inference and reduces the energy cost.

Intuitively, Eq. (1) has a connection with template matching [31] in computer vision, which aims to find the parts of an image that match the template. $F$ in Eq. (1) actually works as a template, and we calculate its matching scores with different regions of the input feature $X$. Since various metrics can be utilized in template matching, it is natural that $\ell_1$ distance can be utilized to replace the cross-correlation in Eq. (1). Note that there are few works also focus on utilizing different metrics in deep networks. Wang et al. [32] aims to achieve high performance by employing complex metrics. Zhang et al. [33] proposed to use $\ell_{inf}$ distance as basic operation and directly provides a theoretical guarantee of the certified robustness. In contrast, we focus on the $\ell_1$ distance to minimize the energy consumption.
3.2 Universal Approximation by AdderNets

There have been many studies on the approximation capacity of neural networks. Hornik et al. [34] first proved that feedforward networks with one hidden layer using an arbitrary squashing function as the activation function are capable of approximating any Borel measurable function from a finite dimensional space to another up to any desired accuracy. This property is referred to as universal approximation. Leshno et al. [35] further extended the results to feedforward networks with non-polynomial activation functions. Besides the traditional feedforward networks, Schauer et al. [36] proved that the universal approximation property of RNNs. Yun et al. [37] demonstrated that transformers are universal approximators, then the performance of using AdderNets on any applications employed in the traditional neural networks. If AdderNets has not been established. In fact, the AdderNet proves that the universal approximation property of RNNs. This property is referred to as universal approximation.

Lemma 2. The set of functions $g(X) = \sum a_i \text{ReLU}(\|W_i - X\|_1 + b_i)$ can be represented by two-layer AdderNets, where $X$ belongs to a compact set $K$ of $\mathbb{R}^d$ is the input vector, $W_i \in \mathbb{R}^d, a_i \in \mathbb{R}, b_i \in \mathbb{R}$.

Proof. We set the first layer of AdderNet as:

$$L^1 : \mathbb{R}^d \rightarrow \mathbb{R}^t : L^1(X)_i = \|W_i - X\|_1 + b_i,$$

where $X$ is compact and $L^1$ is continuous, there exists $M \in \mathbb{R}$ such that: $|L^1(X)_i| < M, i = 1, 2, \ldots, t$. We then set the second layer as:

$$L^2 : \mathbb{R}^t \rightarrow \mathbb{R} : L^2(x) = \sum_{a_i > 0} |X_i + M| + \sum_{a_i \leq 0} |X_i - M| + (p_1 + p_2)M,$$

where $p_1, p_2$ is the number of $i$ such that $a_i \geq 0$ or $a_i \leq 0$, respectively.

Therefore, the two-layer AdderNet can be denote as:

$$A(x) = \sum_{a_i > 0} |\text{ReLU}(\|W_i - X\|_1 + b_i) + M| + \sum_{a_i \leq 0} |\text{ReLU}(\|W_i - X\|_1 + b_i) - M| - (p_1 + p_2)M$$

$$= \sum_{a_i > 0} |\text{ReLU}(\|W_i - X\|_1 + b_i) - M| \leq M,$$

where $K$ is a compact set of $\mathbb{R}^d$. In other words, for any $\epsilon > 0$, there is an two-layer AdderNet $A$ such that:

$$\int_K |f(X) - A(X)|dX \leq \epsilon,$$

where $X \in K$.

In order to prove Theorem 1 [1] we construct a set of function $g(X) = \sum a_i \text{ReLU}(\|W_i - X\|_1 + b_i)$, and show that it can be represented by a two-layer AdderNet. Thus, it is sufficient to prove that $g(X)$ is a universal approximator.

Theorem 3 (38). The radial basis function networks:

$$\sum_i a_i K(\frac{X - Z_i}{\sigma})$$

is dense in $\ell_1(\mathbb{R}^d)$, where $X, Z_i \in \mathbb{R}^d$, if $K$ is integrable bounded function such that $K$ is continuous almost everywhere and $\int K(X)dX \neq 0$.

Theorem 3 is the universal approximation theorem for the RBF networks, which is proved by Park and Sandberg [38]. We are now ready to present the proof of Theorem 1.

Proof. (of Theorem 1) From Lemma 2 two-layer AdderNets can present the set of functions $g(X) = \sum a_i \text{ReLU}(\|W_i - X\|_1 + b_i)$, while $g(X)$ is dense in $\ell_1(K)$ according to the
universal approximation theorem of radio basis function (RBF) networks (theorem 3). We conclude that the class of two-layer AdderNets is dense in $\ell_1(K)$, where $K$ is an arbitrary compact set in $\mathbb{R}^d$.

Though we show that AdderNets are universal approximators, the expressive power of AdderNets is still unknown, i.e., how many hidden units should we use to achieve a required approximation error. Therefore, we present a approximation bound for AdderNets with a single hidden layer.

**Theorem 4.** Assume that $f$ is Lipschitz: $\exists L_1 > 0$ such that $|f(X) - f(X')| \leq L_1 \|X - X'\|_1$ for all $X$ and $X'$, and assume $\|f\|_1 = \|f(0)\|_d < \infty$. Given any probability measure $\mu$, and define $\|f\|_\mu = \int |f(X)|d\mu(X)$. There exists a single hidden layer AdderNet $A$ with ReLU activation function of width no more than

$$\frac{\|f\|_1}{\min(\epsilon^{d+2}L_1^2, \epsilon^{d+1}L_1\|f\|_\mu)}$$

such that

$$\int (f(X) - A(X))^2d\mu(X) = O(\epsilon^2 L_1^2),$$

where $O(\cdot)$ contains a $d$-dependent constant.

**Proof.** By lemma 2 we can use $g(X)$ to approximate $f$, where $g(X)$ contains:

$$\phi_N(X) = \frac{1}{N} \sum_{i=1}^n a_i r_c(\|X - W_i\|),$$

where $r_c(x) = \frac{r(x)}{n}$ and $r(x) = \max(0, x+1) + \max(0, x-1) - 2\max(0, x)$.

We want to use $\phi_N(X)$ to approximate $f$. To do so, we first approximate $f$ by continuous approximation below:

$$\psi_c(X) = \int f(Z) r_c(\|X - Z\|_1)c_0^{-1}dZ,$$

where $c_0 = \int g(\|X\|_1)dX$.

Since $f$ is Lipschitz, we have

$$\|f(X) - \psi_c(X)\| \leq \int |f(X) - f(\epsilon Z)|r(\|Z\|_1)c_0^{-1}dZ$$

$$\leq \epsilon L_1 c_0^{-1} c_1,$$

where $c_1 = \|f\|_1 r(\|X\|_1)dX$.

Let $q(x) = \frac{f(x)}{\|f\|_\mu}$. We draw $Z_1, \ldots, Z_N \sim q(Z)$, and set $a_i = \text{sgn}(f(Z_i))\|f\|_1 c_0^{-1}$ in (10). This implies

$$E_{Z_1, \ldots, Z_n} \phi_N(X) = \psi_c(X).$$

It follows that

$$E_{Z_1, \ldots, Z_n} (\phi_c(X) - \phi_N(X))^2$$

$$\leq \frac{\|f\|_1^2}{N} E_{Z_1, \ldots, Z_n} r_c(\|X - Z\|_1)^2$$

$$= \frac{\|f\|_1^2}{N} \int |f(Z)| r_c(\|X - Z\|_1)^2dZ$$

$$= \frac{\|f\|_1^2}{N} \int \|f(X - \epsilon Z)| c_0^{-2}r(\|Z\|_1)^2dZ$$

$$\leq \frac{\|f\|_1^2}{N} \epsilon^{d+2} c_0^{-2} \|f\|_1^2 \int \|f(X)\| + G \|Z\|_1 c_0^{-2}r(\|Z\|_1)^2dZ$$

$$= O(N^{-1}\epsilon^{-d}\|f\|_1\|f(X)\| + L_1\epsilon),$$

where $O(\cdot)$ a $d$-dependent constant. From (13), we have:

$$E(\phi_N(X) - f(X))^2$$

$$= E(\phi_N(X) - \psi_c(X))^2 + E(\psi_c(X) - f(X))^2$$

$$= O(N^{-1}\epsilon^{-d}\|f\|_1\|f(X)\|_\mu + L_1\epsilon + \epsilon^2 L_1^2),$$

where $O(\cdot)$ contains a $d$-dependent constant. Taking expectation over $\mu$, we obtain

$$E \int (\phi_N(X) - f(X))^2d\mu(X)$$

$$= O(N^{-1}\epsilon^{-d}\|f\|_1\|f(X)\|_\mu + L_1\epsilon + \epsilon^2 L_1^2).$$

We may pick optimal $N$ with $\|f\|_1 N^{-1}\epsilon^{-d} = c \min(\epsilon^2, \|f\|_\mu^{-1} L_1^2, \epsilon L_1)$, for a constant $c$. Note that with this choice, the desired bound holds for the expectation over the choices of $z_1, \ldots, z_N$ of (10). Therefore there exists a choice of $z_1, \ldots, z_N$ such that the desired bound holds, which implies the theorem.

This theorem shows that two-layer AdderNet achieve the error less than $\epsilon$ by using $O(\epsilon^{-1})$ hidden units (choosing $d = 0$ or $d = -1$). In other words, with $n$ hidden units, AdderNets achieve a integrated squared error of order $O(1/n)$. This result is same as the bound in traditional networks [39], indicating that the two different networks have similar expressive power.

Next, we begin the universal approximation theorem for deep AdderNets with a bounded width, which is similar with the important results in traditional networks [40]. We first present a lemma to represent a set of fully-connect layers by two-later AdderNets.

**Lemma 5.** A fully-connected layer $C_1$ with $m$ outputs whose weights are same or zero along each dimension:

$$C_1 : \mathbb{R}^d \rightarrow \mathbb{R}^m : C_1(X)_i = A_i \sum_{j=1}^d B_{ij} X_j,$$

where $A \in \mathbb{R}^m$, $B_{ij}$ is 0 or 1, and $X \in \mathbb{R}^d$, can be represented by two-layer AdderNets with $2n + 2$ hidden units.

**Proof.** We first show that an adder layer can be constructed to a fully-connected layers $C_2$ with only $+a$ or $-a$ weights in each dimension whose weights are same along each dimension:

$$L_A(X)_i = a_i \sum_j |W_{ij} - X_j| - \|W\|_1 + b_i$$

$$= a_i \sum_j \text{sgn}(W_{ij})(W_{ij} - X_j) - \sum_j \text{sgn}(W_{ij})W_{ij} + b_i$$

$$= a_i \sum_j \text{sgn}(-W_{ij})X_j + b_i,$$

where we set $W_{ij} > \max_k |X_k|$ (since the function is defined in a compact set, $X$ in each layer is bounded).

Now we begin to construct two-layer AdderNet to $C_1$. For the fist layer $L_1$, we set the $i$-th in $L_1$ as: $\text{sgn}(-\tilde{W}_{ij})$ is 1 or -1 if $B_{ij}$ is 1 or 0. The $2i$-th neuron as: $\text{sgn}(-\tilde{W}_{ij})$ is 1 or -1 if $B_{ij}$ is 1 or 0. Then the weight of the second last and last neuron (2$m + 1$ and 2$m + 2$) are all 1 and -1, respectively. For the second layer, the $i$-th neuron is set as:
\[ \text{sgn}(-W^2_{i,j}) = -1 \text{ for the } j = 2i \text{ and } j = 2m + 2 \text{ and } 1 \text{ for others. Thus, the output of } j\text{-th neuron in layer two is:} \\
L_2(L_1(X))_{i} = a_i \sum_j \text{sgn}(-W^2_{i,j})L_1(X)_{j} \\
= a_i (L_1(X)_i - L_1(X)_{2i} + L_1(X)_{2m+1} - L_1(X)_{2m+2}) \\
= 2a_i \sum_j 1_{B_{i,j} > 0}X_j - \sum_j 1_{B_{i,j} = 0}X_j + \sum_j X_j \\
= 4a_i \sum_j B_{i,j}X_j. \tag{19} \]

Note that the ReLU activation between two layer can be easily ignored by selecting large enough bias for the first layer and then the bias can be cut back by the weights in the second layer.

We are now ready to proof universal approximation of width-bounded AdderNet.

**Theorem 6 (Universal Approximation of Width-Bounded AdderNet).** For any \( d \in \mathbb{N} \), the family of width-bounded AdderNet with ReLU as activation function can universally approximate any \( f \in \ell_1(K) \), where \( K \) is a compact set of \( \mathbb{R}^d \). More precisely, for any \( \epsilon > 0 \), there is a deep AdderNet \( A \) with maximum width \( w \leq 2(d + 5) \) such that:

\[ \int_K |f(X) - A(X)|dX \leq \epsilon, \tag{20} \]

where \( X \in K \).

**Proof.** According to Lu et.al. [40], any Lebesgue-integrable function can be approximated by a fully-connected ReLU network with width \( w \leq d + 4 \), which is proved by construction. We can find that the construction use fully-connected layers whose weights are same or zero along each dimension. According to lemma 5, we can use two adder layers to replace each fully-connected layers this construction, and the width for adder layers is less than \( 2(d+5) \).

This theorem can be regarded as a dual version of Theorem 1, which proves that the two-layer AdderNets (i.e., depth-bounded AdderNets) are universal approximator. In fact, the width-bounded traditional neural networks have also been proved to be universal approximators [40], and the bound of width is \( d + 4 \). Our bound for AdderNets (2\(d+5\)) has the same magnitude with that of traditional networks, i.e., \( O(d) \), which implies that the AdderNets has similar approximation capacity with traditional networks.

## 4 OPTIMIZATION

In the above section, we present that AdderNets are universal approximator, which provide a strong theoretical guarantee. We then turn to analyze how to train AdderNets to achieve good performance.

### 4.1 Gradient in AdderNets

Neural networks utilize back-propagation to compute the gradients of filters and stochastic gradient descent to update the parameters. In CNNs, the partial derivative of output features \( Y \) with respect to the filters \( F \) is calculated as:

\[ \frac{\partial Y(m, n, t)}{\partial F(i, j, k, t)} = X(m + i, n + j, k), \tag{21} \]

where \( i \in [m, m + d] \) and \( j \in [n, n + d] \). To achieve a better update of the parameters, it is necessary to derive informative gradients for SGD. In AdderNets, the partial derivative of \( Y \) with respect to the filters \( F \) is:

\[ \frac{\partial Y(m, n, t)}{\partial F(i, j, k, t)} = \text{sgn}(X(m + i, n + j, k) - F(i, j, k, t)), \tag{22} \]

where \( \text{sgn}(\cdot) \) denotes the sign function and the value of the gradient can only take \( +1, 0, \) or \(-1\). Considering the derivative of \( \ell_2 \)-norm

\[ \frac{\partial Y(m, n, t)}{\partial F(i, j, k, t)} = X(m + i, n + j, k) - F(i, j, k, t), \tag{23} \]

Eq. (22) can therefore lead to a signSGD [41] update of \( \ell_2 \)-norm. However, signSGD almost never takes the direction of steepest descent and the direction only gets worse as dimensionality grows [42]. It is unsuitable to optimize the neural networks of a huge number of parameters using signSGD. Therefore, we propose using Eq. (23) to update the gradients in our AdderNets.

### 4.2 \( \ell_2 \) to \( \ell_1 \) Training

It is common in deep learning to lean a “weak” model with the help of a “strong” model. For example, to train a low bit-width neural network [33], [43]. Motivated by the full-precision gradient, we further introduce the \( \ell_2 \)-AdderNets, which calculate \( \ell_2 \) distance between the filter and the input feature, the filters in \( \ell_2 \)-AdderNets can be reformulated as

\[ Y(m, n, t) = -\sum_{i=0}^{d} \sum_{j=0}^{d} \sum_{k=0}^{c_m} [X(m+i, n+j, k) - F(i, j, k, t)]^2. \tag{24} \]

In fact, the output of the \( \ell_2 \)-AdderNets can be calculated as

\[ Y_{\ell_2}(m, n, t) = -\sum_{i=0}^{d} \sum_{j=0}^{d} \sum_{k=0}^{c_m} [X(m+i, n+j, k) - F(i, j, k, t)]^2 \]

\[ = \sum_{i=0}^{d} \sum_{j=0}^{d} \sum_{k=0}^{c_m} [2X(m+i, n+j, k) \times F(i, j, k, t) \]

\[ - X(m+i, n+j, k)^2 - F(i, j, k, t)^2] \]

\[ = 2Y_{\text{CNN}}(m, n, t) - \sum_{i=0}^{d} \sum_{j=0}^{d} \sum_{k=0}^{c_m} [X(m+i, n+j, k)^2 \]

\[ + F(i, j, k, t)^2]. \tag{25} \]

\[ \sum_{i=0}^{d} \sum_{j=0}^{d} \sum_{k=0}^{c_m} F(i, j, k, t)^2 \] is same for each channel (i.e., each fixed \( t \)), \[ \sum_{i=0}^{d} \sum_{j=0}^{d} \sum_{k=0}^{c_m} X(m+i, n+j, k)^2 \] is the \( \ell_2 \)-norm of each input patch. If this term is same for each patch, the output of \( \ell_2 \)-AdderNet can be roughly seen as a linear transformation of the output of CNN. Although the \( \ell_2 \)-AdderNet can achieve comparable performance with CNNs, its calculation contain square operations, which introduce multiplications and would bring large energy consumption compared with the \( \ell_1 \)-AdderNet.
To this end, we propose an $\ell_2$ to $\ell_1$ training strategy to utilize the ability of $\ell_2$ norm to guide the training of $\ell_1$-AdderNet. We introduce the $\ell_p$-AdderNets ($1 \leq p \leq 2$), which can be formulated as:

$$Y(m, n, t) = -\sum_{i=0}^{d} \sum_{j=0}^{d} \sum_{k=0}^{d} |X(m+i, n+j, k) - F(i, j, k, t)|^p.$$  

(26)

Since it is difficult to directly training the $\ell_1$-AdderNet, we train the $\ell_2$-AdderNets at the beginning of training. During the training procedure, $p$ is linearly reduced from 2 to 1. Therefore, the $\ell_p$-AdderNet becomes $\ell_1$-AdderNet at the end of training. Since it is easy to see that the Proposition 2 is also right for $\ell_p$-AdderNet when $1 < p < 2$, the partial derivative of output features $Y$ in $\ell_p$-AdderNet with respect to the filters $F$ is calculated by Eq. (23).

Note that we do not use the full precision gradient like Eq. (23) since the derivative of $X$ would influence the gradient in not only current layer but also layers before the current layer according to the gradient chain rule, and the change of gradients will make the training unstable. The partial derivative of output features $Y$ in $\ell_p$-AdderNet with respect to the input features $X$ is calculated as:

$$\frac{\partial Y(i, j)}{\partial X(i, j)} = \left[F(i, j, k) - X(i, j, k)\right]^{p-1}.$$  

(27)

### 4.3 Adaptive Learning Rate Scaling

In conventional CNNs, assuming that the weights and the input features are independent and identically distributed following normal distribution, the variance of the output can be roughly estimated as:

$$Var[Y_{CNN}] = \sum_{i=0}^{d} \sum_{j=0}^{d} \sum_{k=0}^{d} Var[X \times F]$$

$$= d^2 c_{in} Var[X] Var[F].$$  

(28)

If variance of the weight is $Var[F] = \frac{1}{n^2}$, the variance of output would be consistent with that of the input, which will be beneficial for the information flow in the neural network. In contrast, for AdderNets, the variance of the output can be approximated as:

$$Var[Y_{AdderNet}] = \sum_{i=0}^{d} \sum_{j=0}^{d} \sum_{k=0}^{d} Var[|X - F|]$$

$$= \sqrt{\frac{2}{d^2 c_{in}}(Var[X] + Var[F])},$$

when $F$ and $X$ follow normal distributions. In practice, the variance of weights $Var[F]$ is usually very small [44], e.g., .10^{-3} or 10^{-4} in an ordinary CNN. Hence, compared with multiplying $Var[X]$ with a small value in Eq. (28), the addition operation in Eq. (29) tends to bring in a much larger variance of outputs in AdderNets.

We next proceed to show the influence of this larger variance of outputs on the update of AdderNets. To promote the effectiveness of activation functions, we introduce batch normalization after each adder layer. Given input $x$ over a mini-batch $B = \{x_1, \cdots, x_m\}$, the batch normalization layer can be denoted as:

$$y = \gamma \frac{x - \mu_B}{\sigma_B} + \beta,$$  

(30)

where $\gamma$ and $\beta$ are parameters to be learned, and $\mu_B = \frac{1}{m} \sum_{i=x_i}$ and $\sigma_B^2 = \frac{1}{m} \sum_{i=x_i} - \mu_B^2$ are the mean and variance over the mini-batch, respectively. The gradient of loss $\ell$ with respect to $x$ is then calculated as:

$$\frac{\partial \ell}{\partial x_i} = \sum_{j=1}^{m} \frac{\gamma}{m \sigma_B^2} \left\{ \frac{\partial \ell}{\partial y_i} \frac{\partial y_i}{\partial y_j} \left[1 + \frac{(x_i - x_j)(x_j - \mu_B)}{\sigma_B}\right] \right\}.$$  

(31)

Given a much larger variance $Var[Y] = \sigma_B^2$ in Eq. (29), the magnitude of the gradient w.r.t $X$ in AdderNets would be much smaller than that in CNNs according to Eq. (31), and then the magnitude of the gradient w.r.t the filters in AdderNets would be decreased as a result of gradient chain rule.

**Table 1** The $\ell_2$-norm of gradient of weight in each layer using different networks at 1st iteration.

| Model     | Layer 1 | Layer 2 | Layer 3 |
|-----------|---------|---------|---------|
| AdderNet  | 0.0009  | 0.0012  | 0.0146  |
| CNN       | 0.2261  | 0.2990  | 0.4646  |

Table 1 reports the $\ell_2$-norm of gradients of filters $\|F\|_2$ in LeNet-5-BN using CNNs and AdderNets on the MNIST dataset during the 1st iteration. LeNet-5-BN denotes the LeNet-5 [45] adding a batch normalization layer after each convolutional layer. As shown in this table, the norms of gradients of filters in AdderNets are much smaller than that in CNNs, which could slow down the update of filters in AdderNets.

A straightforward idea is to directly adopt a larger learning rate for filters in AdderNets. However, it is worth noticing that the norm of gradient differs much in different layers of AdderNets as shown in Table 1 which requests special consideration of filters in different layers. To this end, we propose an adaptive learning rate for different layers in AdderNets. Specifically, the update for each adder layer $l$ is
calculated by:

\[ \Delta F_l = \gamma \times \alpha_l \times \Delta L(F_l), \]

where \( \gamma \) is a global learning rate of the whole neural network (e.g., for adder and BN layers), \( \Delta L(F_l) \) is the gradient of the filter in layer \( l \) and \( \alpha_l \) is its corresponding local learning rate. As filters in AdderNets act subtraction with the inputs, the magnitude of filters and inputs are better to be similar to extract meaningful information from inputs. Because of the batch normalization layer, the magnitudes of inputs in different layers have been normalized, which then suggests a normalization for the magnitudes of filters in different layers. The local learning rate can therefore be defined as:

\[ \alpha_l = \frac{\eta \sqrt{k}}{\| \Delta L(F_l) \|_2}, \]

where \( k \) denotes the number of elements in \( F_l \), and \( \eta \) is a hyper-parameter to control the learning rate of adder filters. By using the proposed adaptive learning rate scaling, the adder filters in different layers can be updated with nearly the same step. The training procedure of the proposed AdderNet is summarized in Algorithm 1.

5 EXPERIMENT

In this section, we first conduct toy experiments to show the approximation capacity of AdderNets. We further implement experiments to validate the effectiveness of the proposed AdderNets on several benchmark datasets, including MNIST, CIFAR and ImageNet. Ablation study and visualization of features are provided to further investigate the proposed method. The experiments are conducted on NVIDIA Tesla V100 GPU in PyTorch.

5.1 Toy Experiments of Approximation Capacity

In the above subsections, we have proved that a two-layer adder neural network with a single hidden layer can be regarded as a universal approximator. Here we will further verify the efficiency of AdderNet based universal approximation using some classical toy classification datasets.

Comparisons of AdderNets and traditional neural networks. To achieve a fair comparison, we initialize a two-layer AdderNet and a two-layer feedforward neural networks with \( n \) hidden units. These networks are optimized using SGD with Nesterov’s Accelerated Gradient (NAG). Weight decay and momentum are set as \( 5 \times 10^{-4} \) and 0.9, respectively. Then, we train the two networks for 10,000 iterations using cosine learning rate schedule with an initial learning rate of 0.1. In addition, we use the binary cross entropy loss with sigmoid function for the binary classification task. For classification, the output is classified by whether it is larger than 0.5.

Unit Ball. The training set consists of random samples \( \{(x_i, y_i)\} \) generated from a two-dimensional normal distribution with mean of 0 and variance of 10. The label \( z_i \) for the input sample \((x_i, y_i)\) is calculated as

\[ z_i = \begin{cases} 1, & \sqrt{x_i^2 + y_i^2} < 10, \\ 0, & \sqrt{x_i^2 + y_i^2} > 15. \end{cases} \]

where \( \sqrt{x_i^2 + y_i^2} \) is the Euclidean distance and the points are divided by centrals. So, AdderNet can well distinguish all input samples even when \( n = 1 \).

Multiple Unit Balls. We further construct the classification tasks on the multiple unit ball dataset, where the label \( z_i \) for the input sample \((x_i, y_i)\) is calculated as

\[ z_i = \begin{cases} 1, & \sqrt{(x_i - 10)^2 + (y_i - 10)^2} < 10, \\ 1, & \sqrt{(x_i + 10)^2 + (y_i + 10)^2} < 10, \\ 0, & \text{otherwise,} \end{cases} \]

where \( \{(x_i, y_i)\} \) generated from a two-dimensional normal distribution with mean of 0 and variance of 15.

Since the classification task with multiple unit balls is more complex than that with a single ball, we use \( n = 3, 5, 10 \) as the numbers of hidden units. As shown in figure 3, the traditional networks make some mistakes when \( n = 3 \) and successfully classify most data points when \( n = 10 \). For AdderNet, few data points are misclassified when \( n = 3 \), and all data points are correctly classified when \( n = 5, 10 \). This shows that for certain problems where classes are centered, AdderNets can be superior to traditional neural networks.
Linear Classification. In this example, we consider a linear classification task using traditional networks and AdderNets to verify their capacity. The label $z_i$ for the input sample $(x_i, y_i)$ is calculated as

$$z_i = \begin{cases} 1, & x_i \times y_i \geq 0, \\ 0, & x_i \times y_i < 0, \end{cases} \quad (36)$$

where $\{(x_i, y_i)\}$ generated from a two-dimensional normal distribution with mean of 0 and variance of 10.

Training Data $n = 1$ $n = 2$ $n = 3$

![Fig. 4. Decision boundaries of classifying linear classification obtained by training fully connected networks (top row) and AdderNets (bottom row) with different number of hidden units.](image)

5.3 Experiments on CIFAR

We then evaluate our method on the CIFAR dataset, which consist of $32 \times 32$ pixel RGB color images. Since the binary networks [9] can use the XNOR operations to replace multiplications, we also compare the results of binary neural networks (BNNs). We use the same data augmentation and pre-processing in He et al. [30] for training and testing. Following Zhou et al. [9], the learning rate is set to 0.1 in the beginning and then follows a polynomial learning rate schedule. The models are trained for 800 epochs with a 256 batch size. We follow the general setting in binary networks to set the first and last layers as full-precision convolutional layers. In AdderNets, we use the same setting for a fair comparison. The hyper-parameter $\eta$ is set to 0.1 following the experiments on the MNIST dataset.

The classification results are reported in Table 2. Since computational cost in batch normalization layer, the first layer and the last layer are significantly less than other layers, we omit these layers when counting FLOPs. We first evaluate the VGG-small model [10] in the CIFAR-10 dataset.

### Table 2: Classification results on the CIFAR-10 and CIFAR-100 datasets.

| Model   | Method     | #Mul. | #Add. | XNOR | CIFAR-10 | CIFAR-100 |
|---------|------------|-------|-------|------|----------|-----------|
| VGG-small | BNN       | 0     | 0.65G | 0.65G | 89.80%   | 67.24%    |
|          | $\ell_1$-AddNN | 0     | 1.3G  | 0     | 93.72%   | 74.63%    |
|          | $\ell_2$-AddNN | 0.6G  | 1.3G  | 0     | 94.20%   | 76.04%    |
|          | $\ell_1$-AddNN v2 | 0     | 1.3G  | 0     | 93.75%   | 74.85%    |
|          | CNN       | 0.65G | 0.65G | 0     | 94.25%   | 75.96%    |
| ResNet-20 | BNN       | 0     | 41.17M | 41.17M | 84.87%   | 86.74%    |
|          | $\ell_1$-AddNN | 0     | 82.34M | 0     | 92.90%   | 68.71%    |
|          | $\ell_2$-AddNN | 41.17M | 82.34M | 0     | 92.93%   | 68.81%    |
|          | $\ell_1$-AddNN v2 | 0     | 82.34M | 0     | 92.93%   | 68.75%    |
|          | CNN       | 41.17M | 41.17M | 0     | 92.93%   | 68.75%    |
| ResNet-32 | BNN       | 0     | 69.12M | 69.12M | 87.01%   | 56.21%    |
|          | $\ell_1$-AddNN | 0     | 138.24M | 0     | 93.01%   | 69.02%    |
|          | $\ell_2$-AddNN | 69.12M | 138.24M | 0     | 93.33%   | 69.38%    |
|          | $\ell_1$-AddNN v2 | 0     | 138.24M | 0     | 93.10%   | 69.32%    |
|          | CNN       | 69.12M | 69.12M | 0     | 93.59%   | 70.46%    |

1. www.agner.org/optimize/instruction_tables.pdf
and CIFAR-100 dataset. As a result, the AdderNets achieve nearly the same results with CNNs with no multiplication. Although the model size of BNN is much smaller than those of AdderNet and CNN, its accuracies are much lower. We then turn to the widely used ResNet models (ResNet-20 and ResNet-32) to further investigate the performance of different networks. As for the ResNet-20, The convolutional neural networks achieve the highest accuracy but with a large number of multiplications (41.17M). The proposed AdderNets achieve a 92.31% accuracy in CIFAR-10 and a 67.81% accuracy in CIFAR-100 without multiplications, which is comparable with CNNs. In contrast, the BNNs only achieve 84.87% and 54.14% accuracies in CIFAR-10 and CIFAR-100. The results in ResNet-32 also suggest that the proposed AdderNets can achieve similar results with conventional CNNs.

5.4 Experiments on ImageNet

We next conduct experiments on the ImageNet dataset [2], which consist of 224 × 224 pixel RGB color images. We use ResNet-18 model to evaluate the proposed AdderNets follow the same data augmentation and pre-processing in He et al. [30]. We train the AdderNets for 300 epochs utilizing the cosine learning rate decay [46]. These networks are optimized using Nesterov Accelerated Gradient (NAG), and the weight decay and the momentum are set as 10^{-4} and 0.9, respectively. The batch size is set as 256 and the hyper-parameter in AdderNets is the same as that in CIFAR experiments.

Table 3 shows the classification results on the ImageNet dataset by exploiting different neural networks. The convolutional neural network achieves a 69.8% top-1 accuracy and an 89.1% top-5 accuracy in ResNet-18. However, there are 1.8G multiplications in this model, which bring enormous computational complexity. Since the addition operation has smaller computational cost than multiplication, we propose AdderNets to replace the multiplications in CNNs with subtractions. As a result, our AdderNet achieve a 69.1% top-1 accuracy and an 88.4% top-5 accuracy in ResNet-18, which demonstrate the adder filters can extract useful information from images. Rastegar et al. [8] proposed the XNOR-net to replace the multiplications in neural networks with XNOR operations. Although the BNN can achieve high speed-up and compression ratio, it achieves only a 51.2% top-1 accuracy and a 73.2% top-5 accuracy in ResNet-18, which is much lower than the proposed AdderNet. We then conduct experiments on a deeper architecture (ResNet-50).

The BNN could only achieve a 55.8% top-1 accuracy and a 78.4% top-5 accuracy using ResNet-50. In contrast, the proposed AdderNets can achieve a 75.7% top-1 accuracy and a 92.3% top-5 accuracy, which is closed to that of CNN (76.2% top-1 accuracy and 92.9% top-5 accuracy).

5.5 Visualization Results

Fig. 5. Visualization of filters in the first layer of LeNet-5-BN on the MNIST dataset. Both of them can extract useful features for image classification.

Visualization on filters. We visualize the filters of the LeNet-5-BN network in Figure 5. Although the AdderNets and CNNs utilize different distance metrics, filters of the proposed adder networks (see Figure 5(a)) still share some similar patterns with convolution filters (see Figure 5(b)). The visualization experiments further demonstrate that the filters of AdderNets can effectively extract useful information from the input images and features.

Visualization on features. The AdderNets utilize the L1-distance to measure the relationship between filters and input features instead of cross correlation in CNNs. Therefore, it is important to further investigate the difference of the feature space in AdderNets and CNNs. We train a LeNet++ on the MNIST dataset following [47], which has six convolutional layers and a fully-connected layer for extracting powerful 3D features. Numbers of neurons in each convolutional layer are 32, 32, 64, 64, 128, 128, and 2, respectively. For the proposed AdderNets, the last fully connected layers are replaced with the proposed add filters.

The visualization results are shown in Figure 6. The convolutional neural network calculates the cross correlation between filters and inputs. If filters and inputs are approximately normalized, convolution operation is then equivalent to calculate cosine distance between two vectors. That
is probably the reason that features in different classes are divided by their angles in Figure 1. In contrast, AdderNets utilize the $\ell_1$-norm to distinguish different classes. Thus, features tend to be clustered towards different class centers. The visualization results demonstrate that the proposed AdderNets could have the similar discrimination ability to classify images as CNNs.

![Image](image-url)

**Table 4.** Effectiveness of different training strategy in AdderNets.

| Strategy               | $\ell_2$ to $\ell_1$ training | Adaptive learning rate | Top 1 accuracy |
|------------------------|--------------------------------|------------------------|----------------|
| full-precision gradient| ✓                              | ✓                      | 80.77%         |
| $\ell_2$ to $\ell_1$ training | ✓                            | ✓                      | 85.34%         |
| Adaptive learning rate  | ✓                              | ✓                      | 86.21%         |
| Top 1 accuracy          | ✓                              | ✓                      | 90.14%         |

We further investigate the influence of the $\ell_2$ to $\ell_1$ training strategy in Figure 6. We find that AdderNet v2 can be trained more easily with the help of the $\ell_2$ norm.

**Table 5.** The impact of parameter $\eta$ using LeNet-5-BN on the MNIST dataset.

| $\eta$ | 1   | 0.5 | 0.2 | 0.1 | 0.05 |
|--------|-----|-----|-----|-----|------|
| Acc. (%)| 99.28 | 99.35 | 99.40 | 99.35 | 99.30 |

**Impact of parameters.** As discussed above, the proposed adaptive learning rate scaling has a hyper-parameter: $\eta$. We then test its impact on the accuracy of the student network by conducting the experiments on the MNIST dataset. We use LeNet-5-BN as the backbone of AdderNet. Other experimental settings are same as mentioned in Sec. 5.2. It can be seen from Table 5 that the AdderNets trained utilizing the adaptive learning rate scaling achieves the highest accuracy (99.40%) when $\eta = 0.2$. Based on the above analysis, we keep the setting of hyper-parameters for the proposed method.

**Table 6.** The impact of number of epoch when $p = 1$.

| epoch | 300 | 400 | 500 | 600 | 700 |
|-------|-----|-----|-----|-----|-----|
| Acc. (%)| 91.91 | 92.00 | 92.11 | 92.31 | 91.76 |

**Decay of $p$.** We use the $\ell_p$ norm to training AdderNet, where the $p$ is linearly decayed from 2 to 1. We analyze the impact of number of epoch when $p$ is decayed to 1 on the CIFAR-10 dataset. The network is trained for 800 epochs. As shown in Table 6. If the number of decayed

**5.6 Ablation Study**

We propose to use an $\ell_2$ to $\ell_1$ training strategy to update the filters in our adder filters and design an adaptive learning rate scaling for deal with different layers in AdderNets. It is essential to evaluate the effectiveness of these components. As shown in Table 4, applying the proposed techniques can successfully improve the performance of AdderNets.
epoch is too large (e.g., 700 epoch), the network achieves only 91.76% accuracy, since the AdderNet is not fully trained in ℓ₁ norm. If the number of decayed epoch is too small (e.g., 300 epoch), the network is rarely trained with ℓₚ norm (p > 1), therefore the ℓₚ cannot help the training of ℓ₁ norm. The network achieves the highest performance with 600 decayed epochs, and we keep this setting of hyper-parameters in other experiments.

6 Conclusion

The role of classical convolutions used in deep CNNs is to measure the similarity between features and filters, and we are motivated to replace convolutions with more efficient similarity measure. We investigate the feasibility of replacing multiplications by additions in this work. An AdderNet is explored to effectively use addition to build deep neural networks with low computational costs. This kind of networks calculate the ℓ₁-norm distance between features and filters. We proved that AdderNets are universal approximators, which is analogous to the universal approximation results for two-layer traditional network networks [34] and width-bounded deep networks [49], etc. The universal approximation theorem gives us assurance that AdderNets can solve an arbitrary problem. We further provided empirical results showing that AdderNets can indeed approximate a complex decision boundary with a sufficient number of hidden units or depths. Corresponding optimization method is developed by using ℓₚ-norm. Experiments conducted on benchmark datasets show that AdderNets can well approximate the performance of CNNs with the same architectures, which could have a huge impact on future hardware design. Visualization results also demonstrate that the adder filters are promising to replace original convolution filters for computer vision tasks.

References

[1] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in ICLR, 2015.
[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in NeurIPS, 2012, pp. 1097–1105.
[3] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” in NeurIPS, 2015, pp. 91–99.
[4] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” in CVPR, 2015, pp. 3431–3440.
[5] Y. Ren, K. Zhang, Z. Li, and Y. Qiao, “A discriminative feature learning approach for deep face recognition,” in ECCV. Springer, 2016, pp. 499–515.
[6] M. Courbariaux, Y. Bengio, and J.-P. David, “Binaryconnect: Training deep neural networks with binary weights during propagations,” in NeurIPS, 2015, pp. 3123–3131.
[7] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, “Binarized neural networks,” in NeurIPS, 2016, pp. 4107–4115.
[8] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, “Xnor-net: Imagenet classification using binary convolutional neural networks,” in ECCV. Springer, 2016, pp. 525–542.
[9] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, “Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients,” arXiv preprint arXiv:1606.06160, 2016.
[10] Z. Cai, X. He, J. Sun, and N. Vasconcelos, “Deep learning with low precision by half-wave gaussian quantization,” in CVPR, 2017, pp. 5918–5926.
[37] C. Yun, S. Bhojanapalli, A. S. Rawat, S. J. Reddi, and S. Kumar, “Are transformers universal approximators of sequence-to-sequence functions?” arXiv preprint arXiv:1912.10077, 2019.

[38] J. Park and I. W. Sandberg, “Universal approximation using radial-basis-function networks,” Neural computation, vol. 3, no. 2, pp. 246–257, 1991.

[39] A. R. Barron, “Universal approximation bounds for superpositions of a sigmoidal function,” IEEE Transactions on Information theory, vol. 39, no. 3, pp. 930–945, 1993.

[40] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang, “The expressive power of neural networks: A view from the width,” in Advances in neural information processing systems, 2017, pp. 6231–6239.

[41] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar, “signsgd: Compressed optimisation for non-convex problems,” arXiv preprint arXiv:1802.04434, 2018.

[42] J. Bernstein, K. Azizzadenesheli, Y.-X. Wang, and A. Anandkumar, “Convergence rate of sign stochastic gradient descent for non-convex functions,” 2018.

[43] J. Yang, X. Shen, J. Xing, X. Tian, H. Li, B. Deng, J. Huang, and X.-s. Hua, “Quantization networks,” in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 7308–7316.

[44] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in Proceedings of the thirteenth international conference on artificial intelligence and statistics, 2010, pp. 249–256.

[45] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner et al., “Gradient-based learning applied to document recognition,” Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, 1998.

[46] I. Loshchilov and F. Hutter, “Sgdr: Stochastic gradient descent with warm restarts,” arXiv preprint arXiv:1608.03983, 2016.

[47] Y. Wen, K. Zhang, Z. Li, and Y. Qiao, “A discriminative feature learning approach for deep face recognition,” in ECCV, 2016.

[48] S. M. Stigler, The history of statistics: The measurement of uncertainty before 1900. Harvard University Press, 1986.

[49] J. Rennie, “On l2-norm regularization and the gaussian prior,” 2003.

Hanting Chen received a B.E degree from Tongji University, China. Currently, he is a Ph.D. candidate with the Key Laboratory of Machine Perception (Ministry of Education) at Peking University. His research interests lie primarily in machine learning and computer vision.

Chang Xu is Senior Lecturer and ARC DECRA Fellow at the School of Computer Science, University of Sydney. He received the Ph.D. degree from Peking University, China. His research interests lie in machine learning algorithms and related applications in computer vision. He has published over 100 papers in prestigious journals and top tier conferences. He has received several paper awards, including Distinguished Paper Award in IJCAI 2018. He regularly serves as the PC member or senior PC member for many conferences, e.g., NeurIPS, ICML, ICLR, CVPR, ICCV, IJCAI and AAAI. He has been recognized as Top Ten Distinguished Senior PC Member in IJCAI 2017.

Yunhe Wang is currently a Senior Researcher at Noah’s Ark Lab, Huawei Technologies. He received a Ph.D. degree from Peking University, China. His research interests mainly include machine learning, computer vision, and efficient deep learning. He has published over 50 papers in prestigious journals and conferences. Many of them are widely applied in industrial products, and received important awards. He leads the “Adder Neural Networks” project, which presents a new computing paradigm for artificial intelligence with significantly lower hardware costs. He regularly serves as PC/senior PC member for top conferences, e.g., NeurIPS, ICML, ICLR, CVPR, ICCV, IJCAI, and AAAI.

Chao Xu received the B.E. degree from Tsinghua University in 1988, the M.S. degree from University of Science and Technology of China in 1991 and the Ph.D degree from Institute of Electronics, Chinese Academy of Sciences in 1997. Between 1991 and 1994 he was employed as an assistant professor by University of Science and Technology of China. Since 1997 Dr. Xu has been with School of EECS at Peking University where he is currently a Professor. His research interests are in image and video coding, processing and understanding. He has authored or co-authored more than 80 publications and 5 patents in these fields.

Chunjing Xu received the B.E. degree from Wuhan University, China, the M.S. degree from Peking University, China, and the Ph.D. degree from the Chinese University of Hong Kong, Hong Kong, China. He is currently a principal researcher with Huawei Noah’s Ark Lab. His research interests lie primarily in deep learning, and computer vision.

Tong Zhang is a professor of Computer Science and Mathematics at the Hong Kong University of Science and Technology. Previously, he was a professor at Rutgers university, and worked at IBM, Yahoo, Baidu, and Tencent. He received a B.A. in mathematics and computer science from Cornell University and a Ph.D. in Computer Science from Stanford University. His research interests include machine learning algorithms and theory, statistical methods for big data and their applications. He is a fellow of ASA and IMS, and he has been in the editorial boards of leading machine learning journals and program committees of top machine learning conferences.