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. The influence of this new similarity measure on the optimization of neural network have been thoroughly analyzed. To achieve a better performance, we develop a special training approach for AdderNets by investigating the ℓ_p-norm. We then propose an adaptive learning rate strategy to enhance the training procedure of AdderNets according to the magnitude of each neuron's gradient. As a result, the proposed AdderNets can achieve 75.7% Top-1 accuracy 92.3% Top-5 accuracy using ResNet-50 on the ImageNet dataset without any multiplication in convolutional layer. Moreover, we 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. These results match those of the traditional neural networks using the more complex multiplication units. An approximation bound for AdderNets with a single hidden layer is also presented.

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], [2], object detection [3], 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.* ,smart phone, 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

- *Hanting Chen and Chao Xu are with the Key Laboratory of Machine Perception (Ministry of Education) and Coopertative Medianet Innovation Center, School of EECS, Peking University, Beijing 100871, P.R. China. E-mail:htchen@pku.edu.cn, xuchao@cis.pku.edu.cn*
- *Hanting Chen, Yunhe Wang and Chunjing Xu are with the Noah's Ark Laboratory, Huawei Technologies Co., Ltd, HuaWei Building, No.3 Xinxi Road, Shang-Di Information Industri Base, Hai-Dian District, Beijing 100085, P.R. China. E-mail:htchen@pku.edu.cn, yunhe.wang@huawei.com*
- *Chang Xu is with the School of Computer Science in the Faculty of Engineering and Information Technologies at The University of Sydney, J12 Cleveland St, Darlington NSW 2008, Australia. E-mail: c.xu@sydney.edu.au, dacheng.tao@sydney.edu.au.*
- *Tong Zhang is with the School of Computer Science and Engineering, and Mathematics at the Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong. E-mail: tongzhang@tongzhang-ml.org.*
- *Hanting Chen and Yunhe Wang contributed equally to this manuscript.*
- *Correspondence to Chang Xu.*

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 multiplications 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, ℓ_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, ℓ_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. An improved ℓ_2 to ℓ_1 training scheme is designed 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. 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. Moreover, we present a approximation bound for AdderNets with a single hidden layer.

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.* [11] decomposed weight matrices of fully-connected layers into simple calculations by exploiting singular value decomposition (SVD). Han *et.al.* [12] proposed discarding subtle weights in pre-trained deep networks to omit their original calculations without affecting the performance. Wang *et.al.* [13] further converted convolution filters into the DCT frequency domain and eliminated more floating number multiplications. In addition, Hu *et.al.* [14] discarded redundant filters with less impacts to directly reduce the computations brought by these filters. Luo *et.al.* [15] discarded redundant filters according to the reconstruction error. Hu *et.al.* [16] 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.* [17] 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.* [18] designed MobileNet, which decompose the conventional convolution filters into the point-wise and depth-wise convolution filters with much fewer FLOPs. Zhang *et.al.* [19] combined group convolutions [20] and a channel shuffle operation to build efficient neural networks with fewer computations. Wu *et.al.* [21] 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.* [22] developed versatile convolution filters to generate more useful features utilizing fewer calculations and parameters. Xu *et.al.* [23] 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.* [24] 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.* [25] 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.* [26] exploit the hint layer to distill the information in features of the teacher network to the student network. You *et.al.* [27] utilized multiple teachers to guide the training of the student network and achieve better performance. Yim *et.al.* [28] 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.

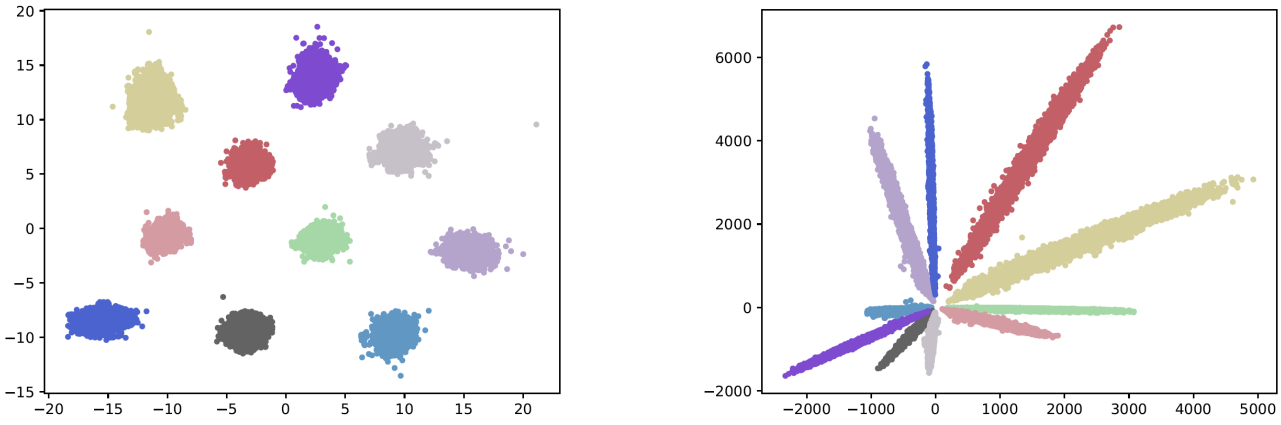
Nevertheless, the compressed networks using these algorithms still contain massive multiplications, which costs enormous computation resources. As a result, subtractions or additions are of much lower computational complexities when compared with multiplications. However, they have not been widely investigated 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.

3 Networks without Multiplication

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,

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

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.



(a) Visualization of features in AdderNets

(b) Visualization of features in CNNs

Fig. 1. Visualization of features in AdderNets and CNNs. Features of CNNs in different classes are divided by their angles. In contrast, features of AdderNets tend to be clustered towards different class centers, since AdderNets use the ℓ_1-norm to distinguish different classes. The visualization results suggest that ℓ_1-distance can serve as a similarity measure the distance between the filter and the input feature in deep neural networks

3.1 Adder Networks

We are therefore interested in deploying distance metrics that maximize the use of additions. ℓ_1 distance calculates the sum of the absolute differences of two points' vector representations, which contains no multiplication. Hence, by calculating ℓ_1 distance between the filter and the input feature, Eq. (1) can be reformulated as:

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

Addition is the major operation in ℓ_1 distance measure, since subtraction can be easily reduced to addition by using complement code. With the help of ℓ_1 distance, similarity between the filters and features can be efficiently computed.

Although both ℓ_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 $\mathcal{O}(d^2 c_{in} c_{out} HW)$ and $\mathcal{O}(c_{out} H'W')$, respectively. In practice, given an input channel number $c_{in} = 512$ and a kernel size $d = 3$ in ResNet [29], we have $\frac{d^2 c_{in} c_{out} HW}{c_{out} H'W'} \approx 4068$. 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 [30] 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 ℓ_1 distance can be utilized to replace the cross-correlation in Eq. (1). Note that Wang *et.al.* [31] also discussed different metrics in deep networks. However, they focused on achieve high performance by employing complex metrics while we focus on the ℓ_1 distance to minimize the energy consumption.

3.2 Optimization

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:

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

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:

$$\frac{\partial Y(m,n,t)}{\partial F(i,j,k,t)} = \mathrm{sgn}(X(m+i,n+j,k) - F(i,j,k,t)), \quad (4)$$

where $\mathrm{sgn}(\cdot)$ denotes the sign function and the value of the gradient can only take +1, 0, or -1.

Eq. (4) can therefore lead to a signSGD [32] update of ℓ_2-norm. However, signSGD almost never takes the direction of steepest descent and the direction only gets worse as dimensionality grows [33]. It is unsuitable to optimize the neural networks of a huge number of parameters using signSGD. To this end, we propose to introduce the ℓ_2 norm to help the training of AdderNets.

Considering the derivative of ℓ_2-norm:

$$\frac{\partial Y(m,n,t)}{\partial F(i,j,k,t)} = X(m+i,n+j,k) - F(i,j,k,t), \quad (5)$$

Eq. (4) can therefore lead to a signSGD [32] update of ℓ_2-norm. However, signSGD almost never takes the direction of steepest descent and the direction only gets worse as dimensionality grows [33]. It is unsuitable to optimize the neural networks of a huge number of parameters using signSGD. Therefore, we propose using Eq. (5) to update the gradients in our AdderNets. We further investigate the convergence of taking these two kinds of gradients.

Proposition 1. *Denote an input patch as $x \in \mathbb{R}^n$ and a filter as $f \in \mathbb{R}^n$, the optimization problem is:*

$$arg\min_f ||x-f|-y|, \quad (6)$$

where $y \in \mathbb{R}$ is the desired output. Given the learning rate α, this problem basically cannot converge to the optimal value using sign grad (Eq. (4)) via gradient descent.

Proof. The optimization problem 6 can be rewritten as:

$$\arg\min_{f_1,...,f_n} |\sum_{i=1}^{n} |x_i - f_i| - y|, \quad (7)$$

where $x = \{x_1, ..., x_n\}, f = \{f_1, ..., f_n\}$. When $y < 0$, the optimization problem becomes:

$$\arg\min_{f_1,...,f_n} \sum_{i=1}^{n} |x_i - f_i|, \quad (8)$$

The update of f_i using gradient descent is:

$$f_i^{j+1} = f_i^j - \alpha \text{sgn}(f_i^j - x_i), \quad (9)$$

where f_i^j denotes the f_i in jth iteration. Without loss of generality, we assume that $f_i^0 < x_i$. So we have:

$$f_i^{j+1} = f_i^j + \alpha = f_i^{j-1} + 2\alpha = ... = f_i^0 + (j+1)\alpha, \quad (10)$$

when $f_i^j < x_i$. Denote $t = \arg\max_j f_i^j < x_i$, we have $f_i^{t+1} >= x_i$. If $f_i^{t+1} = f_i^0 + (t+1)\alpha = x_i$ (*i.e.* , $\frac{(x_i - f_i^0)}{\alpha} = t+1$), $|f_i - x_i|$ can converge to the optimal value 0. However, if $f_i^{t+1} > x_i$, we have

$$f_i^{t+2} = f_i^{t+1} - \alpha \text{sgn}(f_i^{t+1} - x_i) = f_i^0 + (t+1)\alpha - \alpha = f_i^t \quad (11)$$

Similarly, we have $f_i^{t+3} = f_i^{t+1}$. Therefore, the inequality holds:

$$f_i^{t+2k} = f_i^t < x_i < f_i^{t+2k+1}, k \in \mathbb{N}^+ \quad (12)$$

which demonstrate that the f_i cannot converge and have an error of $x_i - f_i^t$ or $x_i - f_i^t$. The f_i^j can converge to x_i if and only if $\frac{(x_i - f_i^0)}{\alpha} \in \mathbb{Z}$, which is a strict constraint since $x_i, f_i, \alpha \in \mathbb{R}$. Moerover, the f can converge to x if and only if $\frac{(x_i - f_i^0)}{\alpha} \in \mathbb{Z}$ for each $f_i \in f$. The difficulty of converge increases when the number n grows. In neural networks, the dimension of filters is can be very large. Therefore, problem 6 basically cannot converge to its optimal value. For $y > 0$, the result is obviously the same. □

According to the Proposition 1, if we use the sign gradient, the AdderNets will achieve a poor performance.

Proposition 2. *For the optimization peoblem 6, f can converge to the optimal value using full-precision gradient (Eq. (5)) with the learning rate α via gradient descent when $\alpha < 1$.*

Proof. The optimization problem 6 can be rewritten as:

$$\arg\min_{f_1,...,f_n} |\sum_{i=1}^{n} |x_i - f_i| - y|, \quad (13)$$

where $x = \{x_1, ..., x_n\}, f = \{f_1, ..., f_n\}$. When $y < 0$, the optimization problem becomes:

$$\arg\min_{f_1,...,f_n} \sum_{i=1}^{n} |x_i - f_i|, \quad (14)$$

The update of f_i using gradient descent is:

$$f_i^{j+1} = f_i^j - \alpha(f_i^j - x_i), \quad (15)$$

where f_i^j denotes the f_i in jth iteration. If $f_i^j < x_i$, then we have the inequality:

$$f_i^{j+1} = f_i^j - \alpha(f_i^j - x_i) = (1-\alpha)f_i^j + \alpha x_i < x_i, \quad (16)$$

and $f_i^{j+1} < f_i^j$. Without loss of generality, we assume that $f_i^0 < x_i$. Then f_i^j is monotone and bounded with respect to j, so the limit of f_i^j exists and $\lim_{j\to+\infty} f_i^j \leq x_i$. Assume that $\lim_{j\to+\infty} f_i^j = l < x_i$. For $\epsilon = \alpha(x_i - l)$, there exists k subject to $l - f_i^k < \epsilon$. Then we have:

$$f_i^{k+1} = f_i^k + \alpha(x_i - f_i^k) \geq f_i^k + \alpha(x_i - l) > l - \epsilon + \alpha(x_i - l) = l, \quad (17)$$

which is a contradiction. Therefore, $\lim_{j\to+\infty} f_i^j \geq x_i$. Finally, we have $\lim_{j\to+\infty} f_i^j = x_i$, *i.e.* ,$f$ can converge to the optimal value. For $y > 0$, the result is obviously the same. □

Therefore, by utilizing the full-precision gradient, the filters can be updated precisely.

Motivated by th full-precision gradient, we further introduce the ℓ_2-AdderNets, which calculate ℓ_2 distance between the filter and the input feature, the filters in ℓ_2-AdderNets can be reformulated as

$$Y(m,n,t) = -\sum_{i=0}^{d}\sum_{j=0}^{d}\sum_{k=0}^{c_{in}} \left[X(m+i,n+j,k) - F(i,j,k,t)\right]^2. \quad (18)$$

In fact, the output of the ℓ_2-AdderNets can be calculated as

$$\begin{aligned} Y_{\ell_2}(m,n,t) =& -\sum_{i=0}^{d}\sum_{j=0}^{d}\sum_{k=0}^{c_{in}} \left[X(m+i,n+j,k) - F(i,j,k,t)\right]^2 \\ =& \sum_{i=0}^{d}\sum_{j=0}^{d}\sum_{k=0}^{c_{in}} \big[2X(m+i,n+j,k) \times F(i,j,k,t) \\ & - X(m+i,n+j,k)^2 - F(i,j,k,t)^2\big] \\ =& 2Y_{CNN}(m,n,t) - \sum_{i=0}^{d}\sum_{j=0}^{d}\sum_{k=0}^{c_{in}} \big[X(m+i,n+j,k)^2 \\ & + F(i,j,k,t)^2\big]. \end{aligned} \quad (19)$$

$\sum_{i=0}^{d}\sum_{j=0}^{d}\sum_{k=0}^{c_{in}} F(i,j,k,t)^2$ is same for each channel (*i.e.* ,each fixed t). $\sum_{i=0}^{d}\sum_{j=0}^{d}\sum_{k=0}^{c_{in}} X(m+i,n+j,k)^2$ is the ℓ_2-norm of each input patch. If this term is same for each patch, the output of ℓ_2-AdderNet can be roughly seen as

a linear transformation of the output of CNN. Although the ℓ_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 ℓ_1-AdderNet.

To this end, we propose an ℓ_2 to ℓ_1 training strategy to utilize the ability of ℓ_2 norm to guide the training of ℓ_1-AdderNet. We introduce the ℓ_p-AdderNets ($1 \leq p \leq 2$), which can be formulated as:

$$Y(m,n,t) = -\sum_{i=0}^{d}\sum_{j=0}^{d}\sum_{k=0}^{c_{in}} |X(m+i,n+j,k) - F(i,j,k,t)|^p. \tag{20}$$

Since it is difficult to directly training the ℓ_1-AdderNet, we train the ℓ_2-AdderNets at the beginning of training. During the training procedure, p is linearly reduced from 2 to 1. Therefore, the ℓ_p-AdderNet becomes ℓ_1-AdderNet at the end of training. Since it is easy to find that the Proposition 2 is also right for ℓ_p-AdderNet when $1 < p < 2$, the partial derivative of output features Y in ℓ_p-AdderNet with respect to the filters F is calculated by Equ. (5). The partial derivative of output features Y in ℓ_p-AdderNet with respect to the input features X is calculated as:

$$\frac{\partial Y(m,n,t)}{\partial X(i,j,k,t)} = \big[F(i,j,k,t) - X(m+i,n+j,k)\big]^{p-1}. \tag{21}$$

Note that we do not use the full precision gradient like Equ. (5) 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.

3.3 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:

$$\begin{aligned} Var[Y_{CNN}] &= \sum_{i=0}^{d}\sum_{j=0}^{d}\sum_{k=0}^{c_{in}} Var[X \times F] \\ &= d^2 c_{in} Var[X] Var[F]. \end{aligned} \tag{22}$$

If variance of the weight is $Var[F] = \frac{1}{d^2 c_{in}}$, 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:

$$\begin{aligned} Var[Y_{AdderNet}] &= \sum_{i=0}^{d}\sum_{j=0}^{d}\sum_{k=0}^{c_{in}} Var[|X - F|] \\ &= \sqrt{\frac{\pi}{2}} d^2 c_{in} (Var[X] + Var[F]), \end{aligned} \tag{23}$$

when F and X follow normal distributions. In practice, the variance of weights $Var[F]$ is usually very small [34], *e.g.* ,10^{-3} or 10^{-4} in an ordinary CNN. Hence, compared with multiplying $Var[X]$ with a small value in Eq. (22), the addition operation in Eq. (23) 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 $\mathcal{B} = \{x_1, \cdots, x_m\}$, the batch normalization layer can be denoted as:

$$y = \gamma \frac{x - \mu_{\mathcal{B}}}{\sigma_{\mathcal{B}}} + \beta, \tag{24}$$

where γ and β are parameters to be learned, and $\mu_{\mathcal{B}} = \frac{1}{m}\sum_i x_i$ and $\sigma^2_{\mathcal{B}} = \frac{1}{m}\sum_i (x_i - \mu_{\mathcal{B}})^2$ are the mean and variance over the mini-batch, respectively. The gradient of loss ℓ with respect to x is then calculated as:

$$\frac{\partial \ell}{\partial x_i} = \sum_{j=1}^{m} \frac{\gamma}{m^2 \sigma_{\mathcal{B}}} \left\{ \frac{\partial \ell}{\partial y_i} - \frac{\partial \ell}{\partial y_j}[1 + \frac{(x_i - x_j)(x_j - \mu_{\mathcal{B}})}{\sigma_{\mathcal{B}}}] \right\}. \tag{25}$$

Given a much larger variance $Var[Y] = \sigma_{\mathcal{B}}$ in Eq. (23), the magnitude of the gradient w.r.t X in AdderNets would be much smaller than that in CNNs according to Eq. (25), and then the magnitude of the gradient w.r.t the filters in AdderNets would be decreased as a result of gradient chain rule.

Algorithm 1 The feed forward and back propagation of adder neural networks.

Require: An initialized ℓ_p-adder network $\mathcal{N}$ and its training set $\mathcal{X}$ and the corresponding labels $\mathcal{Y}$, the global learning rate γ, $p = 2$ and the hyper-parameter η.

1: **repeat**
2: Randomly select a batch $\{(\mathrm{x}, \mathrm{y})\}$ from $\mathcal{X}$ and $\mathcal{Y}$;
3: Employ the ℓ_p-AdderNet $\mathcal{N}$ on the mini-batch: $\mathrm{x} \rightarrow \mathcal{N}(\mathrm{x})$;
4: Calculate the derivative $\frac{\partial Y}{\partial F}$ and $\frac{\partial Y}{\partial X}$ for adder filters using Eq. (5) and Eq. (21);
5: Exploit the chain rule to generate the gradient of parameters in $\mathcal{N}$;
6: Calculate the adaptive learning rate α_l for each adder layer according to Eq. (27).
7: Update the parameters in $\mathcal{N}$ using stochastic gradient descent.
8: Decrease p linearly if $p \geq 1$.
9: **until** convergence

Ensure: A well-trained ℓ_1-adder network $\mathcal{N}$ with almost no multiplications.

TABLE 1
The ℓ_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 ℓ_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 [35] adding an 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

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%
	ℓ_1-AddNN	0	1.30G	0	93.72%	74.63%
	ℓ_2-AddNN	0.65G	1.30G	0	94.20%	76.01%
	ℓ_1-AddNN v2	0	1.30G	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%	54.14%
	ℓ_1-AddNN	0	82.34M	0	91.84%	67.60%
	ℓ_2-AddNN	41.17M	82.34M	0	92.90%	68.71%
	ℓ_1-AddNN v2	0	82.34M	0	92.31%	67.81%
	CNN	41.17M	41.17M	0	92.93%	68.75%
ResNet-32	BNN	0	69.12M	69.12M	86.74%	56.21%
	ℓ_1-AddNN	0	138.24M	0	93.01%	69.02%
	ℓ_2-AddNN	69.12M	138.24M	0	93.55%	70.48%
	ℓ_1-AddNN v2	0	138.24M	0	93.10%	69.32%
	CNN	69.12M	69.12M	0	93.59%	70.46%

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:

$$\Delta F_l = \gamma \times \alpha_l \times \Delta L(F_l), \tag{26}$$

where γ 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 α_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:

$$\alpha_l = \frac{\eta\sqrt{k}}{\|\Delta L(F_l)\|_2}, \tag{27}$$

where k denotes the number of elements in F_l, and η 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.

4 Experiment

In this section, we 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.

4.1 Experiments on MNIST

To illustrate the effectiveness of the proposed AdderNets, we first train a LeNet-5-BN [35] on the MNIST dataset. The images are resized to 32×32 and are pro-precessed following [35]. The networks are optimized using Nesterov Accelerated Gradient (NAG), and the weight decay and the momentum were set as 5×10^{-4} and 0.9, respectively. We train the networks for 50 epochs using the cosine learning rate decay [36] with an initial learning rate 0.1. The batch size is set as 256. For the proposed AdderNets, we replace the convolutional filters in LeNet-5-BN with our adder filters. Note that the fully connected layer can be regarded as a convolutional layer, we also replace the multiplications in the fully connect layers with subtractions.

The convolutional neural network achieves a 99.4% accuracy with ∼435K multiplications and ∼435K additions. By replacing the multiplications in convolution with additions, the proposed AdderNet achieves a 99.4% accuracy, which is the same as that of CNNs, with ∼870K additions and almost no multiplication.In fact, the theoretical latency of multiplications in CPUs is also larger than that of additions and subtractions. There is an instruction table [1] which lists the instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD and VIA CPUs. For example, in VIA Nano 2000 series, the latency of float multiplication and addition is 4 and 2, respectively. The AdderNet using LeNet-5 model will have ∼1.7M latency while CNN will have ∼2.6M latency in this CPU. In conclusion, the AdderNet can achieve similar accuracy with CNN but have fewer computational cost and latency. Noted that CUDA and cuDNN optimized adder convolutions are not yet available, we do not compare the actual inference time.

4.2 Experiments on CIFAR

We then evaluate our method on the CIFAR dataset, which consist of 32×32 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 pro-precessing in He *et.al.* [29] 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

1. www.agner.org/optimize/instruction_tables.pdf

TABLE 3
Classification results on the ImageNet datasets.

Model	Method	#Mul.	#Add.	XNOR	Top-1 Acc.	Top-5 Acc.
ResNet-18	BNN	0	1.8G	1.8G	51.2%	73.2%
	ℓ_1-AddNN	0	3.6G	0	67.0%	87.6%
	ℓ_2-AddNN	1.8G	3.6G	0	69.9%	89.1%
	ℓ_1-AddNN v2	0	3.6G	0	69.1%	88.4%
	CNN	1.8G	1.8G	0	69.8%	89.1%
ResNet-50	BNN	0	3.9G	3.9G	55.8%	78.4%
	ℓ_1-AddNN	0	7.7G	0	74.9%	91.7%
	ℓ_2-AddNN	3.9G	7.7G	0	76.1%	92.8%
	ℓ_1-AddNN v2	0	7.7G	0	75.7%	92.3%
	CNN	3.9G	3.9G	0	76.2%	92.9%

comparison. The hyper-parameter η is set to 0.1 following the experiments on the MNIST dataset.

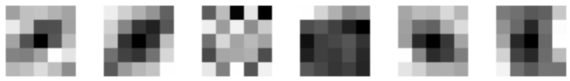
The classification results are reported in Table 2. Since computational cost in batch normalization layer, the first layer and the last layer are significantly less than other layers, we omit these layers when counting FLOPs. We first evaluate the VGG-small model [10] in the CIFAR-10 and CIFAR-100 dataset. 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.

4.3 Experiments on ImageNet

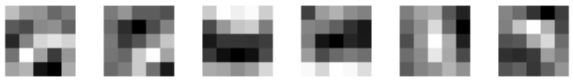
We next conduct experiments on the ImageNet dataset [2], which consist of 224×224 pixel RGB color images. We use ResNet-18 model to evaluate the proposed AdderNets follow the same data augmentation and pro-precessing in He *et.al.* [29]. We train the AdderNets for 300 epochs utilizing the cosine learning rate decay [36]. 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 nerual 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. Rastegari *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).

4.4 Visualization Results



(a) Visualization of filters of AdderNets



(b) Visualization of filters of CNNs

Fig. 2. 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 2. Although the AdderNets and CNNs utilize different distance metrics, filters of the proposed adder networks (see Figure 2 (a)) still share some similar patterns with convolution filters (see Figure 2 (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 ℓ_1-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 [37], 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

TABLE 4
Effectiveness of different training strategy in AdderNets.

full-precision gradient		✓			✓	✓		✓
ℓ_2 to ℓ_1 training			✓		✓		✓	✓
Adaptive learning rate				✓		✓	✓	✓
Top 1 accuracy	80.77%	85.34%	86.21%	90.14%	87.61%	91.83%	91.80%	92.32%

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 1. 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 ℓ_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.

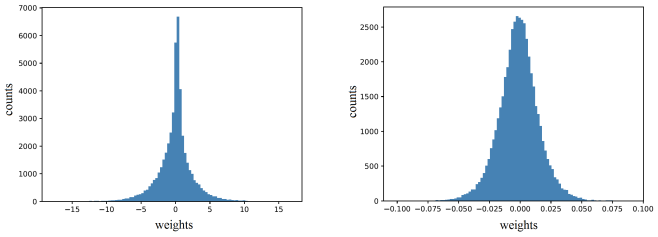


Fig. 3. Histograms over the weights with AdderNet (left) and CNN (right). The weights of AdderNets follow Laplace distribution while those of CNNs follow Gaussian distribution.

Visualization on distribution of weights. We then visualize the distribution of weights for the 3rd convolution layer on LeNet-5-BN. As shown in Figure 3, the distribution of weights with AdderNets is close to a Laplace distribution while that with CNNs looks more like a Gaussian distribution. In fact, the prior distribution of ℓ_1-norm is Laplace distribution [38] and that of ℓ_2-norm is Gaussian distribution [39].

4.5 Ablation Study

We propose to use an ℓ_2 to ℓ_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.

TABLE 5
The impact of parameter η using LeNet-5-BN on the MNIST dataset.

η	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: η. 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. 4.1. 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 ℓ_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 epoch is too large (*e.g.* ,700 epoch), the network achieves only 91.76% accuracy, since the AdderNet is not fully trained in ℓ_1 norm. If the number of decayed epoch is too small (*e.g.* ,300 epoch), the network is rarely trained with ℓ_p norm ($p > 1$), therefore the ℓ_p cannot help the training of ℓ_1 norm. The network achieves the highest performance with 600 decayed epochs, and we keep this setting of hyper-parameters in other experiments.

5 Universal Approximation by AdderNets

There have been many studies on the approximation capacity of neural networks. Hornik *et.al.* [40] 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.* [41] further extended the results to feedforward networks with non-polynomial activation functions. Besides the traditional feedforward networks, Schafer *et.al.* [42] proved that the universal approximation property of RNNs. Yun *et.al.* [43] demonstrated that transformers are universal approximators of sequence-to-sequence functions.

Nevertheless, the universal approximation property of AdderNets has not been established. In fact, the AdderNet calculates the Manhattan distance between the input features and filters, which is fundamentally different to traditional feedforward networks using cross correlation. Thus, most operations in AdderNets are cheaper additions which are significantly more energy efficient than massive multiplications employed in the traditional neural networks. If

we can utilize adder units to construct universal approximators, then the performance of using AdderNets on any deep learning based applications can be guaranteed.

5.1 Proof of Universal Approximation Theorem

Here we briefly review the annotation of each adder layer along with the batch normalization layer and present two universal approximation theorems for AdderNets, one for shallow networks with a single hidden layer and the other for deep networks with bounded width.

Definition 1. For the arbitrary input data $\mathbf{X} \in \mathbb{R}^{d_{in}}$ the weight parameters $\mathbf{W} \in \mathbb{R}^{d_{in} \times d_{out}}$ where $d_{in}, d_{out} \in \mathbb{N}$, a one-layer adder net L using batch normalization for calculate the outputs $Y \in \mathbb{R}^{d_{out}}$ can be formulated as:

$$L : \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{out}} : L(\mathbf{X})_i = a_i \|\mathbf{W}_i - \mathbf{X}\|_1 + b_i, \tag{28}$$

where $\mathbf{W}_i \in \mathbb{R}^{d_{in}}$ is the i-th column in $\mathbf{W}$, $a_i, b_i \in \mathbb{R}$ are the scale factor and bias for the i-th output channel, respectively. Based on the above definition for a single adder layer, we then define the multi-layer adder network.

Definition 2. For an arbitrary activation function $G(\cdot)$ for mapping $\mathbb{R}$ to $\mathbb{R}$ and the d-dimentional input data $\mathbf{X} \in \mathbb{R}^d$ a n layers AdderNet A for calculating a scalar can be formulated as

$$A : \mathbb{R}^d \to \mathbb{R} : A(\mathbf{X}) = L_n(G(L_{n-1}(\dots G(L_1(\mathbf{X}))\dots))). \tag{29}$$

Definition 3. A width-bounded AdderNet means the maximum number of neuron in each layer of this network is bounded by a given constant $w \in \mathbb{N}$.

Then, we aim to prove that both a two-layer AdderNet and a width-bounded AdderNet with the widely used nonlinear activation function $G(x) = \text{ReLU}(x) = \max(x, 0)$ is a universal approximator. Note that the above definition does not contain any convolutional layers, max pooling and others operations, because we investigate the AdderNet itself for constructing the universal approximator.

Theorem 1 (Universal Approximation of Two-Layer AdderNet)**.** *For any $n \in \mathbb{N}$, the family of AdderNet with ReLU as activation function can universally approximate any $f \in \ell_1(K)$, where K is a compact set of $\mathbb{R}^d$. In other words, for any $\epsilon > 0$, there is an two-layer AdderNet A such that:*

$$\int_K |f(\mathbf{X}) - A(\mathbf{X})| d\mathbf{X} \leq \epsilon, \tag{30}$$

where $\mathbf{X} \in K$.

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

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

Proof. We set the first layer of AdderNet as:

$$L^1 : \mathbb{R}^d \to \mathbb{R}^t : L^1(\mathbf{X})_i = |a_i| \|\mathbf{W}_i - \mathbf{X}\|_1 + b_i, \tag{31}$$

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

$$\begin{aligned} L^2 : \mathbb{R}^t \to \mathbb{R} : L^2(x) = &\sum_{a_i > 0} |\mathbf{X}_i + M| \\ &+ \sum_{a_i \leq 0} |\mathbf{X}_i - M| + (p_1 + p_2)M, \end{aligned} \tag{32}$$

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:

$$\begin{aligned} A(x) = &\sum_{a_i > 0} |\text{ReLU}(|a_i| \|\mathbf{W}_i - \mathbf{X}\|_1 + b_i + M)| \\ &+ \sum_{a_i \leq 0} |\text{ReLU}(|a_i| \|\mathbf{W}_i - \mathbf{X}\|_1 + b_i) - M| - (p_1 + p_2)M \\ = &\sum_{a_i > 0} \text{ReLU}(|a_i| \|\mathbf{W}_i - \mathbf{X}\|_1 + b_i) \\ &- \sum_{a_i \leq 0} \text{ReLU}(|a_i| \|\mathbf{W}_i - \mathbf{X}\|_1 + b_i) \\ = &\sum_i a_i \text{ReLU}(\|\mathbf{W}_i - \mathbf{X}\|_1 + b_i). \end{aligned} \tag{33}$$

□

Theorem 3 ([44])**.** *The radio basis function networks:*

$$\sum_i a_i K(\frac{\mathbf{X} - \mathbf{Z}_i}{\sigma}) \tag{34}$$

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

Theorem 3 is the universal approximation theorem for the RBF networks, which is proved by Park and Sandberg [44]. 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 $g(\mathbf{X}) = \sum_i a_i \text{ReLU}(\|\mathbf{W}_i - \mathbf{X}\|_1 + b_i)$, while $g(\mathbf{X})$ 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 $\mathbb{R}^d$. □

Next, we begin to the universal approximation theorem for deep AdderNets with a bounded width. We first present a lemma to represent a set of fully-connect layers by two-layer AdderNets.

Lemma 4. *A fully-connected layer C_1 with m outputs whose weights are same or zero along each dimension:*

$$C_1 : \mathbb{R}^d \to \mathbb{R}^m : C(\mathbf{X})_i = \mathbf{A}_i \sum_{j=1}^{d} \mathbf{B}_{ij} \mathbf{X}_j, \tag{35}$$

where $A \in \mathbb{R}^m$, $\mathbf{B}_{ij}$ is 0 or 1, and $\mathbf{X} \in \mathbb{R}^d$, can be represented by two-layer AdderNets with $2m + 2$ 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:

$$\begin{aligned} L_A(\mathbf{X})_i &= a_i \sum_j |\mathbf{W}_{ij} - \mathbf{X}_j| - \|\mathbf{W}_i\|_1 + b_i \\ &= a_i \sum_j \text{sgn}(\mathbf{W}_{ij})(\mathbf{W}_{ij} - \mathbf{X}_j) - \sum_j \text{sgn}(\mathbf{W}_{ij})\mathbf{W}_{ij} + b_i \\ &= a_i \sum_j \text{sgn}(-\mathbf{W}_{ij})\mathbf{X}_j + b_i, \end{aligned} \tag{36}$$

where we set $\mathbf{W}_{ij} > \max_k |\mathbf{X}_k|$ (since the function is defined in a compact set, $\mathbf{X}$ 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: $\text{sgn}(-\mathbf{W}^1_{ij})$ is 1 or -1 if $\mathbf{B}_{ij}$ is 1 or 0. The $2i$-th neuron as: $\text{sgn}(-\mathbf{W}^1_{(2i)j})$ is -1 or 1 if $\mathbf{B}_{ij}$ is 1 or 0. Then the weight of the second last and last neuron ($2m+1$ and $2m+2$) are all 1 and -1, respectively. For the second layer, the i-th neuron is set as: $\text{sgn}(-\mathbf{W}^2_{ij}) = -1$ for the $j = 2i$ and $j = 2m+2$ and 1 for others. Thus, the output of j-th neuron in layer two is:

$$\begin{aligned} L_2(L_1(\mathbf{X}))_i =& a_i \sum_j \text{sgn}(-\mathbf{W}^2_{ij}) L_1(\mathbf{X})_j \\ =& a_i (L_1(\mathbf{X})_i - L_1(\mathbf{X})_{2i} + L_1(\mathbf{X})_{2m+1} \\ & - L_1(\mathbf{X})_{2m+2}) \\ =& 2a_i (\sum_j \mathbf{1}_{\mathbf{B}_{ij}>0}\mathbf{X}_j - \sum_j \mathbf{1}_{\mathbf{B}_{ij}=0}\mathbf{X}_j + \sum_j \mathbf{X}_j) \\ =& 4a_i \sum_{j=1} \mathbf{B}_{ij}\mathbf{X}_j. \end{aligned} \tag{37}$$

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 5 (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:*

$$\int_K |f(\mathbf{X}) - A(\mathbf{X})| d\mathbf{X} \leq \epsilon, \tag{38}$$

where $\mathbf{X} \in K$.

Proof. According to Lu *et.al.* [45], 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 4, we can use two adder layers to replace each fully-connected layers this construction, and the width for adder layers is less than $2(d+5)$. □

5.2 Approximation bound

In this subsection, we present a approximation bound for AdderNets with a single hidden layer.

Theorem 6. *Assume that f is Lipschitz: $\exists L_1 > 0$ such that $|f(\mathbf{X}) - f(\mathbf{X}')| \leq L_1 \|\mathbf{X} - \mathbf{X}'\|_1$ for all $\mathbf{X}$ and $\mathbf{X}'$, and assume $\|f\|_1 = \int |f(\mathbf{X})| d\mathbf{X} < \infty$. Given any probability measure μ, and define $\|f\|_\mu = \int |f(\mathbf{X})| d\mu(\mathbf{X})$. There exists a single hidden layer AdderNet A with ReLU activation function of width no more than*

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

such that

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

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

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

$$\phi_N(\mathbf{X}) = \frac{1}{N} \sum_{i=1}^n a_i r_\epsilon(\|\mathbf{X} - \mathbf{W}_i\|), \tag{39}$$

where $r_\epsilon(x) = \frac{r(\frac{x}{\epsilon})}{\epsilon^d}$ and $r(x) = \max(0, x+1) + \max(0, x-1) - 2\max(0, x)$.

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

$$\psi_\epsilon(\mathbf{X}) = \int f(\mathbf{Z}) r_\epsilon(\|\mathbf{X} - \mathbf{Z}\|_1) c_0^{-1} d\mathbf{X}, \tag{40}$$

where $c_0 = \int g(\|\mathbf{X}\|_1) d\mathbf{X}$.

Since f is Lipschitz, we have

$$\begin{aligned} |f(\mathbf{X}) - \psi_\epsilon(\mathbf{X})| &\leq \int |f(\mathbf{X}) - f(\mathbf{X} - \epsilon\mathbf{Z})| r(\|\mathbf{Z}\|_1) c_0^{-1} d\mathbf{Z} \\ &\leq \epsilon L_1 c_0^{-1} c_1, \end{aligned} \tag{41}$$

where $c_1 = \int \|\mathbf{X}\|_1 r(\|\mathbf{X}\|_1) d\mathbf{X}$

Let $q(x) = \frac{|f(\mathbf{X})|}{\|f\|_1}$. We draw $\mathbf{Z}_1, \ldots, \mathbf{Z}_N \sim q(\mathbf{Z})$, and set $a_i = \text{sgn}(f(\mathbf{Z}_i)) \|f\|_1 c_0^{-1}$ in (39). This implies

$$E_{\mathbf{Z}_1,\ldots,\mathbf{Z}_n} \phi_N(\mathbf{X}) = \psi_\epsilon(\mathbf{X}). \tag{42}$$

It follows that

$$\begin{aligned} & E_{\mathbf{Z}_1,\ldots,\mathbf{Z}_n} (\psi_\epsilon(\mathbf{X}) - \phi_N(\mathbf{X}))^2 \\ \leq & \frac{\|f\|_1^2}{N} E_{\mathbf{Z}_1 \sim q(\mathbf{Z})} c_0^{-2} r_\epsilon(\|\mathbf{X} - \mathbf{Z}_1\|_1)^2 \\ = & \frac{\|f\|_1}{N} \int |f(\mathbf{Z})| c_0^{-2} r_\epsilon(\|\mathbf{X} - \mathbf{Z}\|_1)^2 d\mathbf{Z} \\ = & \frac{\|f\|_1}{N\epsilon^d c_0^2} \int |f(\mathbf{X} - \epsilon\mathbf{Z})| c_0^{-2} r(\|\mathbf{Z}\|_1)^2 d\mathbf{Z} \\ \leq & \frac{\|f\|_1}{N\epsilon^d c_0^2} \int (|f(\mathbf{X})| + G\epsilon\|\mathbf{Z}\|_1) c_0^{-2} r(\|\mathbf{Z}\|_1)^2 d\mathbf{Z} \\ = & O(N^{-1}\epsilon^{-d} \|f\|_1 [|f(\mathbf{X})| + L_1\epsilon]), \end{aligned} \tag{43}$$

where $O(\cdot)$ a d-dependent constant. From (42), we have:

$$\begin{aligned} & E(\phi_N(\mathbf{X}) - f(\mathbf{X}))^2 \\ = & E(\phi_N(\mathbf{X}) - \psi_\epsilon(\mathbf{X}))^2 + E(\psi_\epsilon(\mathbf{X}) - f(\mathbf{X}))^2 \\ = & O(N^{-1}\epsilon^{-d} \|f\|_1 [|f(\mathbf{X})| + L_1\epsilon] + \epsilon^2 L_1^2), \end{aligned} \tag{44}$$

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

$$\begin{aligned} & E \int (\phi_N(\mathbf{X}) - f(\mathbf{X}))^2 d\mu(\mathbf{X}) \\ = & O(N^{-1}\epsilon^{-d} \|f\|_1 [\|f(\mathbf{X})\|_\mu + L_1\epsilon] + \epsilon^2 L_1^2). \end{aligned} \tag{45}$$

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 (39). Therefore there exists a choice of $z_1, \ldots, z_N$ such that the desired bound holds, which implies the theorem. □

5.3 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.

Comparisions 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 5×10^{-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

$$z_i = \begin{cases} 1, & \sqrt{x_i^2 + y_i^2} < 10, \\ 0, & \sqrt{x_i^2 + y_i^2} > 15. \end{cases} \tag{46}$$

where a margin is created between positive and negative samples to make classification easier.

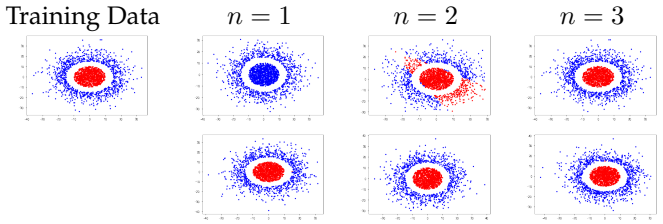


Fig. 4. Decision boundaries of classifying unit balls obtained by training fully connected networks (top row) and AdderNets (bottom row) with different number of hidden units.

Figure 4 shows the decision boundaries of classifying unit balls obtained by training fully connected traditional neural networks and AdderNets with two layers, n hidden units and ReLU activation functions. As the figure shows, the traditional network using multiplications fails to classify these points when $n = 1$ and $n = 2$ and obtains good results when $n = 3$. In fact, the traditional network utilizes the cross-correlation between input data and weight parameters (*i.e.*, the classifier) to calculate the output. Thus the classification results of the given data shown in figrue 4 are divided by lines. In contrast, AdderNet utilizes the ℓ_1-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

$$z_i = \begin{cases} 1, & \sqrt{(x_i - 10)^2 + (y_i - 10)^2} < 10, \\ 1, & \sqrt{(x_i + 10)^2 + (y_i + 10)^2} < 10, \\ 0, & \text{otherwise}, \end{cases} \tag{47}$$

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

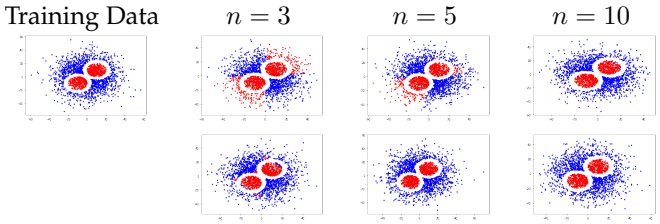


Fig. 5. Decision boundaries of classifying multiple unit balls obtained by training fully connected networks (top row) and AdderNets (bottom row) with different number of hidden units.

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 5, the traditional networks make some mistakes when $n = 3, 5$ 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

$$z_i = \begin{cases} 1, & x_i \times y_i \geq 0, \\ 0, & x_i \times y_i < 0, \end{cases} \tag{48}$$

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

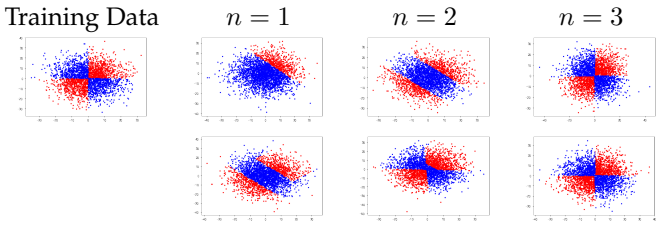


Fig. 6. Decision boundaries of classifying linear classification obtained by training fully connected networks (top row) and AdderNets (bottom row) with different number of hidden units.

Figure 6 shows the classification results. When $n = 1$ and $n = 2$, neither AdderNets nor traditional networks performs well on this problem. When $n = 3$, they can successfully classify all data points. The results empirically indicate that the adder neural network can also be regarded as a universal approximator and the approximation power increases with more hidden units.

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 ℓ_1-norm distance between features and filters. Corresponding optimization method is developed by using ℓ_p-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. Moreover, we proved that AdderNets are universal approximators, which is analogous to the universal approximation results for two-layer traditional network networks [40] and width-bounded deep networks [45], 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.

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. Wen, 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.

[11] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *NeuriPS*, 2014.

[12] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[13] Y. Wang, C. Xu, S. You, D. Tao, and C. Xu, "Cnnpack: Packing convolutional neural networks in the frequency domain," in *NeuriPS*, 2016, pp. 253–261.

[14] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," *arXiv preprint arXiv:1607.03250*, 2016.

[15] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *ICCV*, 2017, pp. 5058–5066.

[16] T.-K. Hu, T. Chen, H. Wang, and Z. Wang, "Triple wins: Boosting accuracy, robustness and efficiency together by enabling input-adaptive inference," *arXiv preprint arXiv:2002.10025*, 2020.

[17] Y. Wang, Z. Jiang, X. Chen, P. Xu, Y. Zhao, Y. Lin, and Z. Wang, "E2-train: Training state-of-the-art cnns with over 80% energy savings," in *NeuriPS*, 2019, pp. 5139–5151.

[18] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[19] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *CVPR*, 2018, pp. 6848–6856.

[20] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *CVPR*, 2017, pp. 1492–1500.

[21] B. Wu, A. Wan, X. Yue, P. Jin, S. Zhao, N. Golmant, A. Gholaminejad, J. Gonzalez, and K. Keutzer, "Shift: A zero flop, zero parameter alternative to spatial convolutions," in *CVPR*, 2018, pp. 9127–9135.

[22] Y. Wang, C. Xu, C. Xu, C. Xu, and D. Tao, "Learning versatile filters for efficient convolutional neural networks," in *NeuriPS*, 2018, pp. 1608–1618.

[23] F. Juefei-Xu, V. Naresh Boddeti, and M. Savvides, "Perturbative neural networks," in *CVPR*, 2018, pp. 3310–3318.

[24] K. Han, Y. Wang, Q. Tian, J. Guo, C. Xu, and C. Xu, "Ghostnet: More features from cheap operations," *arXiv preprint arXiv:1911.11907*, 2019.

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

[26] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," *arXiv preprint arXiv:1412.6550*, 2014.

[27] S. You, C. Xu, C. Xu, and D. Tao, "Learning from multiple teacher networks," in *SIGKDD*. ACM, 2017, pp. 1285–1294.

[28] J. Yim, D. Joo, J. Bae, and J. Kim, "A gift from knowledge distillation: Fast optimization, network minimization and transfer learning," in *CVPR*, 2017, pp. 4133–4141.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[30] R. Brunelli, *Template matching techniques in computer vision: theory and practice*. John Wiley & Sons, 2009.

[31] C. Wang, J. Yang, L. Xie, and J. Yuan, "Kervolutional neural networks," in *CVPR*, 2019, pp. 31–40.

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

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

[34] 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.

[35] 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.

[36] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," *arXiv preprint arXiv:1608.03983*, 2016.

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

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

[39] J. Rennie, "On l2-norm regularization and the gaussian prior," 2003.

[40] K. Hornik, M. Stinchcombe, H. White *et al.*, "Multilayer feedforward networks are universal approximators." *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[41] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural networks*, vol. 6, no. 6, pp. 861–867, 1993.

[42] A. M. Schäfer and H. G. Zimmermann, "Recurrent neural networks are universal approximators," in *International Conference on Artificial Neural Networks*. Springer, 2006, pp. 632–640.

[43] 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.

[44] J. Park and I. W. Sandberg, "Universal approximation using radial-basis-function networks," *Neural computation*, vol. 3, no. 2, pp. 246–257, 1991.
[45] 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.