NON-EUCLIDEAN VECTOR PRODUCT FOR NEURAL NETWORKS

Arman Afrasiyabi[†*] *Diaa Badawi*[§] *Barış Nasır*[*] *Ozan Yıldız*[*] *Fatoş T. Yarman Vural*[*] *A. Enis Çetin*[§¶]

[*]Department of Computer Engineering, Middle East Technical University, Ankara, Turkey
[§]Depatment of Electrical and Electronics Engineering, Bilkent University, Ankara, Turkey
[†]Electrical and Computer Engineering Department, Université Laval, Quebec City, Canada
[¶]Department of ECE, University of Illinois at Chicago, Chicago, Illinois
{bnasir, vural, oyildiz}@ceng.metu.edu.tr, badawi@ee.bilkent.edu.tr,
arman.afrasiyabi.1@ulaval.ca, aecyy@uic.edu

ABSTRACT

We present a non-Euclidean vector product for artificial neural networks. The vector product operator does not require any multiplications while providing correlation information between two vectors. Ordinary neurons require inner product of two vectors. We propose a class of neural networks with the universal approximation property over the space of Lebesgue integrable functions based on the proposed non-Euclidean vector product. In this new network, the "product" of two real numbers is defined as the sum of their absolute values, with the sign determined by the sign of the product of the numbers. This "product" is used to construct a vector product in R^N. The vector product induces the l_1 norm. The additive neural network successfully solves the XOR problem. Experiments on MNIST and CIFAR datasets show that the classification performance of the proposed additive neural network is comparable to the corresponding multi-layer perceptron and convolutional neural networks.

***Index Terms*—** non-Euclidean operator, additive neural networks, multiplication-free operator

1. INTRODUCTION

Deep Neural Networks (DNN) and Convolutional Neural Network (CNN) architectures achieve human performance in many computer vision problems [15], [16, 14, 24]. However, the number of parameters in these high-performance networks ranges from millions to billions which require computers capable of handling high computational complexity, high energy and memory size. Recent developments in VLSI industry create powerful mobile devices which can be used in many practical recognition applications. ANNs are already being used in drones and unmanned aerial vehicles for flight control, path estimation [5],[10].

However, the current structure of the ANNs, especially, deep networks, cannot be implemented effectively on mobile devices due to high energy requirements. A typical neuron needs to perform three main tasks to produce an output: (i) an inner product operation involving multiplication of inputs by weights, (ii) addition, and (iii) pass the result of the inner product through an activation function. The multiplication operation is the most energy consuming operation [11].

In this paper, we propose an l_1 norm based energy efficient neural network, called *additive neural network*, that replaces the multiplication operation with a new energy efficient operator, called *mf-operator*. Instead of multiplications, we use sign multiplications and addition operations in a typical neuron. The sign multiplication of two real numbers is a simple bit operation.

Related Work: The l_1 norm based vector product is first introduced in 2009 [26, 25, 4, 8] and used in some image processing applications. A multiplication free neural network structure is proposed in 2015 [3]. However, the recognition rate was below 10%of a regular neural network. In this article, we are able to match the performance of regular neural networks by introducing a scaling factor to the l_1 norm based vector product and new training methods. For example, we are only 0.034% below the recognition rate of a regular neural network in MNIST dataset.

Other solutions to energy efficient neural networks include dedicated software for a specific hardware, i.e. neuromorphic devices [9, 18, 20, 17, 19], [11], [2]. Although such approaches reduce energy consumption and memory usage, they require special hardware. Our neural network framework can be implemented in ordinary microprocessors and digital signal processors.

Rastegari et al. proposes two methods to provide efficiency on CNNs. The first method, Binary-Weight-Networks (BWN), approximates all the weight values to binary values [21]. As a result the network needs less memory (nearly $\times 32$). Convolutions can be estimated by only addition and subtraction, which eliminates the power draining multiplications. Although BWN catches the state-of-the-art model on ImageNet dataset, the error rate on MNIST dataset is 9.88%. The second method proposed by them is called XNOR-Networks where both weights and inputs to the convolutional and fully connected layers are approximated by binary values.This method offers $\times 58$ faster computation on CPU on average and results 12% loss in accuracy on the average. However, the recognition accuracy is lower than the ordinary neural networks. In [6], BinaryConnect is proposed, which is a DNN where weights are binarized during feedforwarding pass. In the backpropagation pass, the sensitivities are calculated for the binary weights but real weights values are retained for parameters update. The weight binarization can be either deterministic or stochastic. In [12], binarized neural networks are proposed, where weights and activations are binarized during forward pass. The gradient is altered to make it possible for back-propagating the error. Furthermore, with GPU optimization, they could achieve faster forward passes (nearly $\times 7$).

A. Enis Çetin and Diaa Badawi's work was funded by an NSF grant with grant number 1739396

2. NEW VECTOR PRODUCT OPERATOR AND ANN

Let $\mathbf{x}$ and $\mathbf{y}$ be two vectors in $\mathbb{R}^d$. We define a new vector product based on a new operator as follows:

$$\mathbf{x} \oplus \mathbf{y} := \sum_{i=1}^{d} \text{sign}(x_i \times y_i)(|x_i| + |y_i|), \tag{1}$$

where $\mathbf{x} = [x_1, \ldots, x_d]^T, \mathbf{y} = [y_1, \ldots, y_d]^T \in \mathbb{R}^d$. We call the new operator the multiplication-free (mf) operator which can also be represented as follows: $\mathbf{x} \oplus \mathbf{y} := \sum_{i=1}^{d} \text{sign}(x_i) y_i + \text{sign}(y_i) x_i$. The new vector product, $\oplus$, operation does not require any multiplications. The operation $(x_i \times y_i)(|x_i| + |y_i|)$ uses the sign of the ordinary multiplication, but it computes the sum of absolute values of x_i and y_i. It requires summation, unary minus operation and if statements which are all energy efficient operations. Ordinary inner product of two vectors induces the ℓ_2 norm. Similarly, the vector product defined in (1) induces a scaled version of the ℓ_1 norm: $\mathbf{x} \oplus \mathbf{x} = 2||x||_1$.
We introduce the following notation for a compact representation of the mf-operation of a vector by a matrix. Let $\mathbf{x} \in \mathbb{R}^d$ and $\mathbf{W} \in \mathbb{R}^{d \times M}$ be two matrices, then the mf-operation between $\mathbf{W}$ and $\mathbf{x}$ is defined as follows;

$$\mathbf{x} \oplus \mathbf{W} := \begin{bmatrix} \mathbf{x} \oplus \mathbf{w}_1 & \ldots & \mathbf{x} \oplus \mathbf{w}_M \end{bmatrix}^T \in \mathbb{R}^M, \tag{2}$$

where $\mathbf{w}_j$ is jth column of $\mathbf{W}$ for $j = 1, 2, \ldots, M$.

3. ADDITIVE NEURAL NETWORK (ADDNET) WITH MF-OPERATOR

We define a new neuron by replacing inner-product of a classical neural network by the vector product defined in (1). A neuron in a classical neural network is represented by the activation function $f(\mathbf{x}\mathbf{W} + \mathbf{b})$, where $\mathbf{W} \in \mathbb{R}^{d \times M}$, $\mathbf{b} \in \mathbb{R}^M$ are weights and biases, respectively, and $\mathbf{x} \in \mathbb{R}^d$ is the input vector. A neuron in AddNet is represented by the activation function, the affine transform is modified by using the mf-operator as follows;

$$f(\mathbf{a} \odot (\mathbf{x} \oplus \mathbf{W}) + \mathbf{b}), \tag{3}$$

where $\odot$ is element-wise multiplication operator, $\mathbf{W} \in \mathbb{R}^{d \times M}$, $a, b \in \mathbb{R}^M$ are weights, scaling coefficients and biases, respectively, and $\mathbf{x} \in \mathbb{R}^d$ is the input vector. The neural network, where each neuron is represented by the activation function defined in (3), is called Additive Neural Network (AddNet).

In AddNet, we replace the affine scoring function (**xW+b**) of a classical neural network by the scoring function defined over the mf-operator, $(\mathbf{a} \odot (\mathbf{x} \oplus \mathbf{W}) + \mathbf{b})$. Therefore, most of the neural networks can easily be converted into the additive network by just representing the neurons with the activation functions defined over mf-operator, without modification of the topology and the general structure of the optimization algorithms of the network.

Training the AddNet: Standard back-propagation algorithm is applicable to the AddNET with some approximations. Back-propagation algorithm computes derivatives with respect to current values of parameters of a differentiable function to update its parameters. The activation function, f, can be excluded during these computations for simplicity as its derivation depends on the specific activation function and choice of activation function does not affect the remaining computations. Hence, the only difference in the AddNet training is the computation of the derivatives of the argument, $(\mathbf{a} \odot (\mathbf{x} \oplus \mathbf{W}) + \mathbf{b})$, of the activation function with respect to the parameters, $\mathbf{W}, \mathbf{a}, \mathbf{b}$, and input, $\mathbf{x}$, as given below:

$$\frac{\partial(\mathbf{a} \odot (\mathbf{x} \oplus \mathbf{W}) + \mathbf{b})}{\partial \mathbf{a}} = \mathbf{x} \oplus \mathbf{W}, \tag{4}$$

$$\frac{\partial(\mathbf{a} \odot (\mathbf{x} \oplus \mathbf{W}) + \mathbf{b})}{\partial \mathbf{b}} = \mathbf{1}_M, \tag{5}$$

$$\begin{aligned} \frac{\partial(\mathbf{a} \odot (\mathbf{x} \oplus \mathbf{W}) + \mathbf{b})}{\partial \mathbf{x}_i} &= \begin{bmatrix} \mathbf{a}_1(sign(\mathbf{W}_{i,1}) + 2\mathbf{W}_{i,1}\delta(\mathbf{x}_i)) \\ \vdots \\ \mathbf{a}_M(sign(\mathbf{W}_{i,M}) + 2\mathbf{W}_{i,M}\delta(\mathbf{x}_i)) \end{bmatrix} \\ &\approx \mathbf{a} \odot \mathbf{sign}(\mathbf{w}_i), \end{aligned} \tag{6}$$

$$\begin{aligned} \frac{\partial(\mathbf{a} \odot (\mathbf{x} \oplus \mathbf{W}) + \mathbf{b})}{\partial \mathbf{W}_{i,j}} &= (a_j(sign(\mathbf{x}_i) + 2\mathbf{x}_i\delta(\mathbf{W}_{i,j})))\mathbf{e}_j \\ &\approx a_j \mathbf{x}_i \mathbf{e}_j, \end{aligned} \tag{7}$$

where $\mathbf{a}, \mathbf{b} \in \mathbb{R}^M$, and $\mathbf{W} \in \mathbb{R}^{d \times M}$ are the parameters of the hidden layer, $\mathbf{x} \in \mathbb{R}^d$ is the input of the hidden layer, $\mathbf{e}_i \in \mathbb{R}^M$ is the ith element of standard basis of $\mathbb{R}^M$, $\mathbf{w}_i$ is the ith column of $\mathbf{W}$, $\mathbf{sign}(\mathbf{w}_i) = \sum_{j=1}^{M} sign(\mathbf{W}_{i,j})\mathbf{e}_j$ for $i = 1, \ldots, M$, $\mathbf{1}_M = \sum_{j=1}^{M} \mathbf{e}_j$, and δ is the Dirac-delta function which is due to $\frac{d}{dx} sign(x) = 2\delta(x)$. We approximate the derivative operations in above equations by ignoring the case of $x = 0$.

Existence and Convergence of the Solution in AddNet: We show that AddNet satisfies the universal approximation property of [7], over the space of Lebesgue integrable functions. Then, we make a brief analysis for the convergence properties of the back propagation algorithm when the inner product is replaced by the mf-operators.

Universal Approximation Property: In the following proposition, we show that AddNet satisfies the universal approximation theorem for linear and ReLU activation functions.

Proposition 3.1. *The additive neural network (AddNet) defined using the mf-operator with the identity activation function or ReLU activation function*

$$f(\mathbf{a} \odot (\mathbf{x} \oplus \mathbf{W}) + \mathbf{b}) = ReLU(\mathbf{a} \odot (\mathbf{x} \oplus \mathbf{W}) + \mathbf{b}), \tag{8}$$

is dense in $L^1(I_n)$.

In order to prove the above proposition, the following two lemmas should be proved first:

Lemma 3.2. *Let the activation function f be the identity operator. There exist an AddNet, defined using the mf-operator, which can compute $g(x) = sign(\mathbf{y}^T\mathbf{x} + b)$, for any $\mathbf{y} \in \mathbb{R}^d$ and $b \in \mathbb{R}$.*

Lemma 3.3. *If the function $g(x)$ can be computable with the identity activation function then there exist an AddNet architecture with a ReLU activation function, which can also compute $g(x)$.*

We omit the proofs due to the lack of space.

Proof of Proposition 3.1 This can be shown by the universal approximation theorem for bounded measurable sigmoidal functions [7]. This theorem states that finite sums of the form

$$G(\mathbf{x}; \{\alpha_i\}_{i=1}^N, \{\mathbf{y}_i\}_{i=1}^N, \{\theta_i\}_{i=1}^N) = \sum_{i=1}^{N} \alpha_i \sigma(\mathbf{y}_i^T \mathbf{x} + \theta_i), \tag{9}$$

are dense in $L^1(I_n)$, where $\alpha_i, \theta_i \in \mathbb{R}$ and $\mathbf{x}, \mathbf{y}_i \in \mathbb{R}^d$ for $i = 1, 2, \ldots, N$. It can be easily shown that $sign$ function is a bounded sigmoidal function. Lemma 3.2 shows that, if the activation function is taken as identity, then there exist networks which compute $sign(\mathbf{y}_i^T\mathbf{x} + \theta_i)$ for $i = 1, 2 \ldots, N$. **Lemma** 3.3 shows that there are equivalent networks using ReLU as the activation function which compute the same functions. These networks can be combined with concatenation of layers of the additive neural networks to a single network. Also, proposed architecture contains fully connected linear layer at the output, and this layer can compute superposition of the computed $sign$ functions yielding $G(x)$. Since $G(\mathbf{x})$ can be computable by the additive neural networks, and $G(\mathbf{x})$ functions are dense in $L^1(I_n)$, then functions computed by the additive neural networks are also dense in $L^1(I_n)$.

Computational efficiency: AddNet contains more parameters then the classical neuron representation in MLP architectures. However, each hidden layer can be computed using considerably less number of multiplications. A classical neural network, represented by the activation function $f(\mathbf{xW}+\mathbf{b})$, containing M neurons with d dimensional input, requires $d \times M$ many multiplications to compute $\mathbf{xW}+\mathbf{b}$. On the other hand, the additive neural network, represented by the activation function, $f(\mathbf{a}\odot(\mathbf{x}\diamond\mathbf{W})+\mathbf{b})$ with the same number of neurons and input space requires M multiplications to compute $\mathbf{a}\odot(\mathbf{x}\diamond\mathbf{W})+\mathbf{b}$. This reduction on number of multiplications is especially important when input size is large or hidden layer contains large number of neurons. If activation function is taken as either identity or ReLU, then output of this layer can be computed without any complex operations, and efficiency of the network can be substantially increased. Multiplications can be removed entirely, if scaling coefficients, $\mathbf{a}$ are taken as 1 or integer power of 2.

4. EXPERIMENTAL RESULTS

We used MLP with hidden layers, as well as a CNN model in three different classification problems, namely XOR problem, MNIST and CIFAR datasets. In the MLP models the hidden layer(s) also contains neurons constructed using the mf-operator. Each neuron consists of a scoring function and an activation function. In this study, we call the widely used classic scoring function $(\mathbf{xW}+\mathbf{b})$ as c-operator.

Tensorflow [1] is used to train $\mathbf{W}$ and $\mathbf{b}$ using backpropagation [22]. In the first experiment, we examine the ability of additive neural network to partition a simple nonlinear space, solving the XOR problem. In this regard, we could solve the XOR problem with a single hidden layer with 100% accuracy. We compare the classical MLP with affine scoring function and additive neural network with mf-operator. Since a single hidden layer MLP with classical affine operator (c-operator) can solve XOR problem, we used one hidden layer in both classical and the proposed architectures. Mean squared error is used as cost function to measure the amount of loss in training phase of the network, and we fixed the number of neurons in the hidden layer to 10. AddNet with mf-operator can successfully solve the XOR problem and reached to 100% accuracy. We also investigate the rate of changes in loss changes at each epoch. We note that some of the runs do not reach to minimum values in 1000 epochs. This shows that more epochs are needed in some runs.

Left and right sides of Fig. 1 show the change of loss in the MLP using c-operator and mf-operator, respectively, with ReLU as the activation function. We rerun the network for 200 times in 1000 epochs, and used k-fold cross validation to specify the learning-rate parameter of SGD. Each curve shows the variations in loss or cost value (x-axis) across the epochs (y-axis) in one specific run of the network. The cost value of the network with the mf-operator

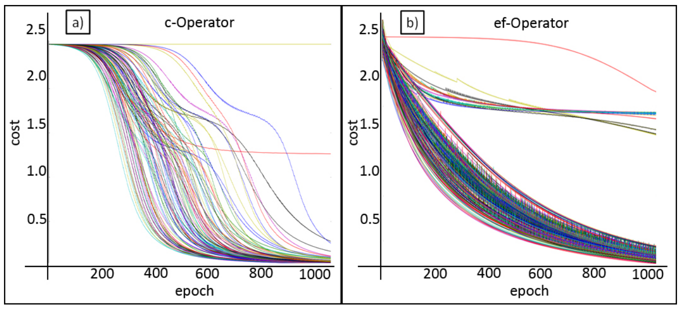

Fig. 1. The plots of loss changes in the stochastic gradient descent (SGD) algorithm in the training phase of XOR problem while using single hidden layer MLP. While the Figure (2.a) shows the the changes of loss in the network by using classical score function **(c-operator)**, Figure (2.b) (right) shows the loss changes in the same network with the proposed **(mf-operator)**. Results were obtained by training the network 200 times in 1000 epochs.

decreases along the epochs and acts similar to classical affine operator, called c-operator.

In the second experiment, we classified the digits of MNIST dataset of [16]. We used the cross-entropy based cost function and stochastic gradient descent (SGD) to train the network.we used a batch size equal to 150. AddNet MLP with $(a = 1)$ reaches to the performance of classic MLP with c-operator in MNIST. Best results were obtained using ReLU activation function. AddNet with two (three) layers achieves 98.09 (97.95) percent recognition accuracy slightly below the ordinary MLP with rates of 98.43 (two layers) and 98.22 (three layers) percent recognition accuracy, respectively.

We also tested two CNN models over MNIST and CIFAR-10 datasets by applying mf-operator in the convolutional units and c-operator in the fully connected units. In this regard, let $\mathbf{W}^l \in \mathbb{R}^{H\times W\times D\times K}$ be the filter tensor of the l^{th} layer, where H, W and D are the filter height, width and depth, respectively and K is the number of filters in the filter bank. We associate a scaling parameter $a \in \mathbb{R}^K$ with this tensor and the corresponding feature map. i,e, each filter w_k^l and its $2D$ output will be associated with a scalar a_k^l.
In our experimentation, we adopted the MNIST model available in "Tensorflow models"[1]. The model has 2 convolutional layers as well as two dense (fully connected) layers. The filter banks are $\mathbf{W^1_{conv}} \in \mathbb{R}^{\mathbf{5\times5\times1\times32}}$ and $\mathbf{W^2_{conv}} \in \mathbb{R}^{\mathbf{5\times5\times32\times64}}$, with each convolutional layer followed by 2×2 max pooling. The sizes of the dense layers are 512 and 10 neurons. This model achieves classification accuracy of 99.2% over test data after training for 10 epochs. The cost function used is mean-square error and the activation function used is $ReLU$.

In a feed-forward pass, the number of add-multiply operations in the convolutional layers is roughly 10.6 Million (mostly in the second layer), whereas the number of add-multiply operations in the dense connections is roughly 1.6 Millions. In this case, since most of the computation takes place in the convolutional units, we opted to replace the c-operator in the add-multiply operations in the convolutional units with our mf-operator and keep the dense units intact.

Table 1 summarizes our experimental results in "convolutional" neural networks. In this case, we replace the multiplication operation in convolutions with the mf-operator. The model does not perform

Table 1. Summary of classification accuracy results over MNIST test dataset for the hybrid CNN model. All the results are obtained after training the models for 10 epochs

Case No.	scaling value	trainable	test accuracy
1	$a^l = \frac{1}{H \times W \times D}$	no	86 %
2	$a^l_k = \sigma(w^l_k)$	no	97.6 %
3	$a^l_k = \frac{\|\|w^l_k\|\|_2^2}{H \times W \times D}$	no	98.5 %
4	$a^l_k = \frac{\|\|w^l_k\|\|_1}{H \times W \times D}$	no	99.0%
5	init: $a^l_k = \sigma(w^l_k)$	yes	98.4%

Table 2. Summary of classification accuracy results over CIFAR-10 test dataset for the hybrid CNN model. All the results are obtained using RMSProp optimizer

scaling value	test accuracy
$a^l_k = \frac{1}{H \times W \times D}$	$\approx 70\%$
a^l trainable (size $X \times Y \times K$)	76.7%
$a^l_k = \sigma(w^l_k)$	80.1%
$a^l_k = \frac{\|\|w^l_k\|\|_1}{H \times W \times D}$	79.5%
a^l_k trainable (size K)	80.8%
equation 10: $f(w^l_k) = \sigma(w^l_k)$	80.7 %
equation 10: $f(w^l_k) = \frac{\|\|w^l_k\|\|_1}{H \times W \times D}$	81.0 %

well when the scaling parameter a is fixed in a convolutional setting. But, the model achieved 99.0% accuracy over MNIST datasets, when the scaling parameter is set to $\frac{||w^l_k||_1}{H \times W \times D}$, where $||w^l_k||_1$ is the ℓ_1 norm of the k^{th} filter w^l_k in the l^{th} layer (norm of a $3D$ tensor). We also studied other scaling methods such as ℓ_2 norm (case 4) and the standard deviation estimate σ for filter w^l_k (case 3). These methods could also achieve decent accuracy rates for a rather simple model. When a is set a trainable parameter for each filter, we were able to achieve accuracy of 98.4%. It is important to note that calculating a^l_k is only done during training. Since weights will be learned and fixed afterwards, we can store a^l_k values as constants and use their values later during inference without having to find them again from the readily learned weights. Furthermore, it may be possible to approximate a^l_k to multiples (or fractions) of the form 2^n where $n \in \mathbb{Z}$. In this way, there will be no need to perform any floating point multiplication.

In a similar fashion, we implemented a hybrid model to be trained over CIFAR-10 dataset [13]. CIFAR-10 dataset consists of 60000 32×32 RGB natural images of 10 categories. In this model, we replaced the multiplication in the convolutional units with mf-operator. The model is a direct adaptation of the CIFAR-10 model in "Tensorflow models", where it comprises two convolutional layers, 2 dense layers and a softmax layer. The input images are randomly cropped into $28 \times 28 \times 3$ subsamples. Furthermore, they are randomly flipped horizontally and contrast is perturbed randomly as a means to acheive data augmentation online [23]. The model consists of two convolutional layers with $\mathbf{W}^1_{conv} \in \mathbb{R}^{5\times5\times3\times64}$ and $\mathbf{W}^2_{conv} \in \mathbb{R}^{5\times5\times64\times64}$ and two dense layers with 384 and 192 neurons, followed by a softmax layer of 10 neurons. This model achieves 86 % accuracy over testing data. The cost function used is cross entropy with softmax layer.

It is important to note that the original model uses local response normalization after each pooling layer. However, based on our early investigation, we could achieve better without local response normalization as it could not exceed a test accuracy of 76%. We could achieve an accuracy of 80.1% over the testing dataset using scaling $a^l_k = \sigma(w^l_k)$ where σ is the standard deviation of the weights of each 3D kernel in the convolutional layers after 250k iterations (640 epochs) using a batch size of 128 and RMSProp optimizer. Contrary to the results over MNIST dataset, the scaling using ℓ_1 norm achieved slightly worse results with accuracy 79.5 %. When a^l_k is set trainable and initialized by the standard deviation of the weight, it could achieve 80.8% after 180k epoch using the same optimizer.

Furthermore, we used hybrid scaling as follows:

$$a^l_k = \alpha^l_k f(w^l_k) + \beta^l_k \qquad (10)$$

where α and β are set trainable and f is function dependent upon w^l_k such as: standard deviation σ and $\ell 1$ norm. The trainable α and β are regularized such that α stays close to unity and β stays small enough by adding two regularization terms to the general loss expression as follows:

$$loss = CE + \lambda_1 \sum ||\alpha^l_k - 1||_2^2 + \lambda_2 \sum ||\beta^l_k||_2^2 \qquad (11)$$

where CE is the cross entropy. Using this method, we could achieve an accuracy of 80.7% over testing data with $f(w^l_k) = \sigma(w^l_k)$ after 250k iterations (640 epochs). Furthermore, we achieved the highest accuracy of 81.0 % with $f(w^l_k) = \frac{||w^l_k||_1}{H \times W \times D}$. Our choices for λ_1 and λ_2 were $10e-3$ and $10e-4$, respectively. Table 2 summarizes our results.

5. CONCLUSION

In this paper, we propose the non-Euclidean AddNet structure. The key operator of AddNet is the mf-operator that eliminates the energy-consuming multiplications in the conventional neural network architectures. As a result, AddNet can be used in mobile devices. AddNet can perform as good as ordinary neural networks in MNIST and CIFAR datasets. We will study the performance of AddNet in ImageNet and other widely used data sets.

6. REFERENCES

[1] M. Abadi and et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] A. M. Abdelsalam, J. Langlois, and F. Cheriet. Accurate and efficient hyperbolic tangent activation function on fpga using the dct interpolation filter. *arXiv:1609.07750*, 2016.

[3] C. E. Akbaş, A. Bozkurt, A. E. Çetin, R. Çetin-Atalay, and A. Üner. Multiplication-free neural networks. In *2015 23th Signal Proc. and Communications Applications Conference (SIU)*, pages 2416–2418. IEEE, 2015.

[4] C. E. Akbaş, O. Günay, K. Taşdemir, and A. E. Çetin. Energy efficient cosine similarity measures according to a convex cost function. *Signal, Image and Video Processing*, pages 1–8.

[5] A. J. Calise and R. T. Rysdyk. Nonlinear adaptive flight control using neural networks. *IEEE control systems*, 18(6):14–25, 1998.

[6] M. Courbariaux, Y. Bengio, and J.-P. David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015.

[7] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[8] H. S. Demir and A. E. Cetin. Co-difference based object tracking algorithm for infrared videos. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 434–438. IEEE, 2016.

[9] S. K. Esser, P. A. Merolla, J. V. Arthur, A. S. Cassidy, R. Appuswamy, A. Andreopoulos, D. J. Berg, J. L. McKinstry, T. Melano, D. R. Barch, et al. Convolutional networks for fast, energy-efficient neuromorphic computing. *arXiv:1603.08270*, 2016.

[10] A. Giusti, J. Guzzi, D. C. Cireşan, F.-L. He, J. P. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. Di Caro, et al. A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robotics and Automation Letters*, 1(2):661–667, 2016.

[11] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015.

[12] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks. In *Advances in neural information processing systems*, pages 4107–4115, 2016.

[13] A. Krizhevsky, V. Nair, and G. Hinton. Cifar-10 (canadian institute for advanced research).

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[15] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

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

[17] S. Moradi and G. Indiveri. An event-based neural network architecture with an asynchronous programmable synaptic memory. *IEEE transactions on biomedical circuits and systems*, 8(1):98–107, 2014.

[18] E. Painkras, L. A. Plana, J. Garside, S. Temple, F. Galluppi, C. Patterson, D. R. Lester, A. D. Brown, and S. B. Furber. Spinnaker: A 1-w 18-core system-on-chip for massively-parallel neural network simulation. *IEEE Journal of Solid-State Circuits*, 48(8):1943–1953, 2013.

[19] J. Park, S. Ha, T. Yu, E. Neftci, and G. Cauwenberghs. A 65k-neuron 73-mevents/s 22-pj/event asynchronous micro-pipelined integrate-and-fire array transceiver. In *2014 IEEE Biomedical Circuits and Systems Conference (BioCAS) Proceedings*, pages 675–678. IEEE, 2014.

[20] T. Pfeil, A. Grübl, S. Jeltsch, E. Müller, P. Müller, M. A. Petrovici, M. Schmuker, D. Brüderle, J. Schemmel, and K. Meier. Six networks on a universal neuromorphic computing substrate. *arXiv:1210.7083*, 2012.

[21] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. *arXiv:1603.05279*, 2016.

[22] H. G. E. Rumelhart, D. E. and R. J. Williams. Learning representations by back-propagating errors. *Nature*, pages 323, 533–536, 1986.

[23] P. Y. Simard, D. Steinkraus, J. C. Platt, et al. Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR*, volume 3, pages 958–962, 2003.

[24] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[25] A. Suhre, F. Keskin, T. Ersahin, R. Cetin-Atalay, R. Ansari, and A. E. Cetin. A multiplication-free framework for signal processing and applications in biomedical image analysis. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1123–1127. IEEE, 2013.

[26] H. Tuna, I. Onaran, and A. E. Cetin. Image description using a multiplier-less operator. *IEEE Signal Processing Letters*, 16(9):751–753, 2009.