It may be time to improve the neuron of artificial neural network

Gang Liu

E-mail: gangliu.6677@gmail.com

Abstract

Artificial neural networks (ANNs) have won numerous contests in pattern recognition, machine learning, or artificial intelligence in recent years. The neuron of ANNs was designed by the stereotypical knowledge of biological neurons 70 years ago. Artificial Neuron is expressed as $f(wx + b)$ or $f(WX)$. This design does not consider dendrites' information processing capacity. However, some recent studies show that biological dendrites participate in the pre-calculation of input data. Concretely, biological dendrites play a role in extracting the interaction information among inputs (features). Therefore, it may be time to improve the neuron of ANNs. According to our previous studies (DD), this paper adds the dendrites' function to artificial Neuron. The dendrite function can be expressed as $W^{i,i-1}A^{i-1} \circ A^{0|1|2|\dots|i-1}$. The generalized new neuron can be expressed as $f(W(W^{i,i-1}A^{i-1} \circ A^{0|1|2|\dots|i-1}))$. The simplified new neuron be expressed as $f(\sum(WA \circ X))$. After improving the neuron, there are so many networks to try. This paper shows some basic architecture for reference in the future.

Interesting things: (1) The computational complexity of dendrite modules ($W^{i,i-1}A^{i-1} \circ A^{i-1}$) connected in series is far lower than Horner's method. Will this speed up the calculation of basic functions in computers? (2) The range of sight of animals has a gradient, but the convolution layer does not have this characteristic. This paper proposes receptive fields with a gradient. (3) The networks using Gang neurons can delete traditional networks' Fully-connected Layer. In other words, the Fully-connected Layers' parameters are assigned to a single neuron, which reduces the parameters of a network for the same mapping capacity.

One important thing: ResDD can replace the current all ANNs' Neurons (ResDD modules+ One Linear module)! ResDD has controllable precision for better generalization capability!

Index Terms

Neural network, neuron, DD, Gang transform, Gang neuron, Receptive fields with gradient, Acceleration algorithm

CONTENTS

I	Introduction	2
I-A	Artificial neural networks	3
I-B	Biological neural network	3
II	Artificial Neuron	4
II-A	Traditional artificial neuron	4
II-B	Novel Gang artificial neuron	4
II-B1	Dendrite processing section ($\circ X$)	4
II-B2	Generalized Dendrite module ($\circ A^i$)	5
II-B3	Generalized Residual Dendrite module ($\circ A^i$)	5
II-B4	ResDD-X or Residual Dendrite module ($\circ X$)	6
II-B5	ResDD-A or Residual Dendrite module ($\circ A^{i-1}$)	6

Gang Liu is at the Institute of Robotics and Intelligent Systems, iHarbour Academy of Frontier Equipment, Xi'an Jiaotong University at present.

1. The study will not be submitted to a journal. Thus, I retain copyright.

2. You can cite this paper and apply Gang neuron to your study (CC BY-NC-SA 4.0. doi:10.36227/techrxiv.12477266).

3. The study will continue to be updated in the future. I'm going to add a section: "Development and application of Gang neurons". If your research using Gang neuron interests me, I will cite yours and update the version.

4. Welcome to follow me. Some information can be found on my GitHub (later updated).

<https://github.com/liugang1234567/Gang-neuron>

Explanation video(I Keep updating it): <https://space.bilibili.com/176499425/channel/detail?cid=137176>

Regardless of the application form and the location of neuron replacement (such as reinforcement learning, etc.), all variants of Dendrite+Cell body applications about the Corresponding module of Gang neuron should be named Gang neuron (Named: "DD of Gang neuron", "ResDD of Gang neuron", "Gang neuron", etc.) and cite this paper. According to the LICENCE: CC BY-NC-SA 4.0, Gang retains the copyright. If there is any infringement that does not meet the specifications, Gang has the right to request the magazine to withdraw the tortious manuscript.

The article is challenging to typeset due to too many Figures. Readers can find "Fig." through hyperlinks.

Citation: Gang neuron (the combination of dendrite in this paper and function $f(\cdot)$) was published on June 16, 2020. All studies using dendrites or the combination of dendrites and function f should cite this article. The combination of dendrites and function $f(\cdot)$ is Gang neuron. The version of this paper is kept updated. The citation format can choose whether to include the current version number. e.g., Liu, Gang (2020): It may be time to improve the neuron of artificial neural network. TechRxiv. Preprint. <https://doi.org/10.36227/techrxiv.12477266> or Liu, Gang (2020): It may be time to improve the neuron of artificial neural network. TechRxiv. Preprint. <https://doi.org/10.36227/techrxiv.12477266.v10>

II-B6	Shared weight of Dendrite module	6
II-B7	Architecture-Gang neuron with one module	8
II-B8	Architecture-Gang “neuron” of unicellular organism	9
II-B9	Architecture-simplified Gang neuron	9
III	ANNs using Gang artificial neuron	9
III-A	Single-layer network	9
III-B	Multi-layer perceptron (MLP) using Gang artificial neuron	11
III-C	Information fusion network	11
IV	Some typical ANNs architecture using Gang neuron	13
IV-A	ResGangneuron (RGN)	13
IV-B	DenseGangneuron (DGN)	13
IV-B1	Dense Dendrite module (DenseDD)	14
IV-B2	DenseGangneuron	14
IV-C	Convolution layer using Gang neuron	14
IV-D	Schematic diagram of RNN using Gang neuron	14
V	Biomimetic architecture using Gang neuron	17
V-A	Module 1: Retina,attention(focus),and convolution-like scan	17
V-B	Module 2: Population coding in brain	17
V-C	Application methods	17
VI	Development and application of Gang neurons	19
VI-A	Development	19
VI-B	Application	19
VII	Tips of Gang neuron	20
VII-A	Avoiding curse of dimensionality and improving generalization ability	20
VII-A1	Adjusting the range of inputs	20
VII-A2	Regularization	20
VIII	Some theories for understanding Gang neurons	20
VIII-A	Polynomial, Convolution, and Dendrite in this paper	20
IX	Discussion and conclusion	21
X	Outlook	22
	References	22
	Biographies	24
	Gang Liu	24

I. INTRODUCTION

AN artificial neural network is an algorithmic architecture that imitates the biological brain [1], [2]. Today, neural networks are gradually changing our lives and making the world better [3]. First of all, I want to thank those who contribute to the present state of the art, e.g., Geoffrey Hinton, YannLeCun, Yoshua Bengio, and so on.

Recently, artificial neural networks, particularly “deep learning” [4], have made some impressive advances in many fields, such as machine vision, speech recognition, autonomous vehicles, and machine translation. Thus, in the tech world today, optimism is high. Some people think ANNs are approaching human intelligence gradually, even it is still far away [2], [3].

Neurons are the fundamental units of the biological brain. Accordingly, the fundamental units of ANNs are artificial neurons. With the development of technology, people have a deeper understanding of the biological brain and neurons. However, the artificial neuron has maintained its original architecture. It may be time to improve the neuron of the artificial neural network.

A brief introduction about the necessary knowledge and latest developments in artificial neural networks and biological neural networks is the following.

Fig. 1. Traditional neuron. (a) Traditional artificial neuron. (b) Traditional imitation to biological neuron.

A. Artificial neural networks

An ANN consists of many interconnected functional units or neurons and can learn the relationship of input space and output space. Thus, they are usually used to solve classification and regression problems [5]. Over the years, based on the application and data's characteristics, different ANN architectures have been developed, such as Convolutional Neural Networks (CNN) [6], [7] in computer vision, Recurrent Neural Networks (RNN) [8] or Long Short Term Memory Network (LSTM) [9] in sequence and time series modeling, and Generative Adversarial Network (GANs) [10].

In the last decade, some newer architectures are developed to endure the need to develop human-like efficient machines in different application areas. From the perspective of information, they can be divided into a network focusing on spatial information [CNN, the representative field is computer vision (CV).] and a network focusing on temporal information [RNN, the representative field is natural language processing (NLP).]. There are several newer CNN architectures and efficient mechanisms: Alexnet [7], VGG [11], Googlenet [12], Inception-V3 [13], ResNet [14], ResNeXt [15], Convolutional Block Attention Module (CBAM) introduced by Woo et al. [16], and competitive squeeze-excitation (SE) mechanism introduced by Hu et al. [17]. There are several newer RNN architectures and efficient mechanisms: Deep Recurrent Attentive Writer (DRAW) [18], Grid Long Short-Term Memory [19], gating mechanism introduced by Jing et al. [20], and factorized recurrent network architecture introduced by Belletti et al. [21].

Besides, the learning of ANNs—by changing the distribution of weights to approximate the relationship between input and output space—has been studied. Lately, success was achieved in many techniques, such as L1 and L2 regularization [22], batch normalization [23], a good collection of weight initialization techniques [24], [25], and a good set of activation functions [5].

B. Biological neural network

The human brain has approximately 100 billion biological neurons, and neurons are connected via specialized structures known as synapses to sense stimulations and pass signals to other neurons [26]. A neuron is the fundamental structural and functional unit of the neural information network and comprises a cell body (soma), dendritic trees, and an axon [27], [28]. The most extended parts of many neurons are dendrites [29], and the active electrical properties of dendrites shape neuronal input.

In the field of biology, researchers have been studying the mechanism of the element of neurons over the years [27], [30]–[32]. Recently, a study discovered a class of calcium-mediated dendritic action potentials (dCaAPs) [32], [33]. Here, we quote the original words in the literature: “These dCaAPs enabled the dendrites of individual human neocortical pyramidal neurons to classify linearly nonseparable inputs—a computation conventionally thought to require multilayered networks.” [32] **Computing exists in the intersection of dendrites. This means that biological dendrites participate in pre-calculation in a biological neural network.**

Reviewing the previous ANNs, I found few studies highlight improving the neuron of ANNs. Fortunately, it happens that Gang et al. [34] propose a module with low computational complexity named DD or Gang transform. The module can solve linearly inseparable problems by Hadamard product. Therefore, it can simulate the dendrites and improve the neuron of ANNs.

Fig. 2. Novel neuron. (a) Gang artificial neuron. (b) Gang imitation to biological neuron.

II. ARTIFICIAL NEURON

A. Traditional artificial neuron

In 1943, McCulloch and Pitts proposed ANN models with adjustable weights [35]. More than a decade later, Rosenblatt put forward the Perceptron Learning Algorithm [36]. In 1986, Rumelhart et al. proposed learning representations by back-propagating errors [37]. By then, a typical artificial neuron was established.

Figure 1 shows the traditional artificial neuron. This architecture ignores the pre-calculation of dendrites. If X is the input and f is the nonlinear activation function, the output y can be represented by,

$$y = f(W^{21}X) \quad (1)$$

Where W^{21} is the weight matrix, $\{1, x_1, x_2, \dots, x_n\} = X$. In the back-propagation procedure, the weight matrix can be updated with,

$$W_{new}^{21} = W^{21} - \eta \frac{\partial E}{\partial W^{21}} \quad (2)$$

Where W_{new}^{21} is the updated weight matrix for W^{21} , E is the cost function or errors from the later layer in back-propagation, and η is the learning rate.

B. Novel Gang artificial neuron

For the biological neurons, the XOR operation is performed in the dendrites with dCaAPs [32], [33], and AND/OR operations are performed at the soma and at tuft and basal dendrites with sodium and NMDA spikes [38]–[40]. The XOR/AND/OR operation means the pre-processing to the interaction information of input data. In the task of understanding a picture, it refers to the relationship between parts of an input-picture. In the task of understanding an article or a speech, it refers to the relationship between words of sentences. However, traditional artificial neurons only simulate the soma.

1) *Dendrite processing section ($\circ X$)*: In previous studies, we presented dendrite modules and introduced logical relationship of inputs [34], [41]. The module is represented as follows.

$$A^i = W^{i,i-1}A^{i-1} \circ X \quad (3)$$

Where A^{i-1} and A^i are the input and output of the module, respectively, and X is the original input. $\circ$ denotes Hadamard product.

In figure 2a, the dendrite is simulated by one module. The “dendrite” contains the interaction item between both inputs and can be represented as follows.

$$A = W^{10}X \circ X \quad (4)$$

Where A is the output of dendrite or the input of the cell body. $\circ$ denotes Hadamard product.

Additionally, the dendrite can also be simulated by using more modules. The “dendrite” in Fig.3 contains the interaction items between multiple inputs. The number of modules plus 1 is the number of interactive inputs. For example, if we use two modules to simulate the dendrite, the “dendrite” contains the interaction items among three inputs and can be represented as follows.

$$\begin{cases} A^1 = W^{10}X \circ X \\ A^2 = W^{21}A^1 \circ X \end{cases} \quad (5)$$

Fig. 3. Gang artificial neuron with multiple dendrite modules.

Fig. 4. Generalized Dendrite module. ‘|’ denotes ‘or’. The dotted line represents any of them.

Where A^2 is the output of dendrite or the input of the cell body. $\circ$ denotes Hadamard product.

The operation of the “dendrite” only consists of matrix multiplication and Hadamard product. Thus, the computational complexity is lower [34], [41]. Additionally, the mapping capacity of Gang neuron can be adjusted by the number of dendrite modules.

2) *Generalized Dendrite module ($\circ A^i$)*: Figure 4 shows a generalized Dendrite module. The module is represented as follows.

$$A^i = W^{i,i-1} A^{i-1} \circ A^{0|1|2|\dots|i-1} \quad (6)$$

Where A^{i-1} and A^i are the inputs and outputs of the module, respectively. $A^{0|1|2|\dots|i-1}$ is any of A . $\circ$ denotes Hadamard product.

We can adjust the promotion degree or the number of interactive variables added for the single module by $A^{0|1|2|\dots|i-1}$. When we add a module, the degree of network can increase by at least one and by i at most.

When $A^{0|1|2|\dots|i-1} = A^0$, the degree increases by 1 for one module, which is similar to Horner’s method. The degree increases with the number of modules linearly. For a certain degree of network, the computational complexity is $n - 1$, where n denotes the number of modules.

When $A^{0|1|2|\dots|i-1} = A^{i-1}$, the degree increases by i for one module. The degree increases with the number of modules exponentially. For a certain degree of network, the computational complexity is approximate $\log_2^n$, where n denotes the number of modules. This is similar to exponentiation by squaring. However, exponentiation by squaring is only used for calculation, and the modules in this paper are used for approximation. The weight matrix in Fig. 4 can be solved for by error backpropagation.

Here, I give a off-topic special USES. when this module is connected in series, it is used for polynomial calculation (Fig. 4). When $A^{0|1|2|\dots|i-1} = A^{i-1}$, there is no doubt that its computational complexity is far lower than Horner’s method. This can change many things.

The typical functions (e.g., $\sin$) are stored in the form of polynomial coefficients in the computer. The value of the function is calculated by polynomial when we invoke these functions.

We can solve for the polynomial coefficients of my new module ($A^{0|1|2|\dots|i-1} = A^{i-1}$) and store these coefficients in the computer. The value of the function is calculated by the network in Fig. 4 when we invoke these functions.

Note: For deeper levels, we can use ResDD or DenseDD (see the ResDD and DenseDD section).

3) *Generalized Residual Dendrite module ($\circ A^i$)*: In order to make DD deeper, a residual strategy can be used (see Fig. 5) [42]. The module is represented as follows.

$$A^i = W^{i,i-1} A^{i-1} \circ A^{0|1|2|\dots|i-1} + W^{i,i-1} A^{i-1} \quad (7)$$

Fig. 5. Generalized Residual Dendrite module. ‘|’ denotes ‘or’. The dotted line represents any of them.

Where A^{i-1} and A^i are the inputs and outputs of the module, respectively. $A^{0|1|2|\dots|i-1}$ is any of A . $W^{i,i-1}$ is a weight matrix. $\circ$ denotes Hadamard product.

4) *ResDD-X or Residual Dendrite module ($\circ X$)*: Figure 7 shows the Residual Dendrite module-X(ResDD-X). The module is represented as follows.

$$A^i = W^{i,i-1} A^{i-1} \circ X + W^{i,i-1} A^{i-1} \quad (8)$$

Where A^{i-1} and A^i are the inputs and outputs of the module, respectively. X is the inputs of DD. $W^{i,i-1}$ is a weight matrix. $\circ$ denotes Hadamard product.

Figure 6 (a) shows ResDD-X code for MNIST dataset in references [34]. The number of modules can be adjusted according to the task.

In addition, we can also add a weight matrix to the short-circuit route. Figure 8 shows the Residual Dendrite module-XW(ResDD-XW). The module is represented as follows.

$$A^i = W^{i,i-1} A^{i-1} \circ X + W_R^{i,i-1} A^{i-1} \quad (9)$$

Where A^{i-1} and A^i are the inputs and outputs of the module, respectively. X is the inputs of DD. $W^{i,i-1}$ and $W_R^{i,i-1}$ are two different weight matrixes. $\circ$ denotes Hadamard product.

Figure 6 (c) shows ResDD-XW code for MNIST dataset in references [34]. The number of modules can be adjusted according to the task.

5) *ResDD-A or Residual Dendrite module ($\circ A^{i-1}$)*: Figure 9 shows the Residual Dendrite module-A(ResDD-A). The module is represented as follows.

$$A^i = W^{i,i-1} A^{i-1} \circ A^{i-1} + W^{i,i-1} A^{i-1} \quad (10)$$

Where A^{i-1} and A^i are the inputs and outputs of the module, respectively. $W^{i,i-1}$ is a weight matrix. $\circ$ denotes Hadamard product.

Figure 6 (b) shows ResDD-A code for MNIST dataset in references [34]. The number of modules can be adjusted according to the task.

In addition, we can also add a weight matrix to the short-circuit route. Figure 10 shows the Residual Dendrite module-AW(ResDD-AW). The module is represented as follows.

$$A^i = W^{i,i-1} A^{i-1} \circ A^{i-1} + W_R^{i,i-1} A^{i-1} \quad (11)$$

Where A^{i-1} and A^i are the inputs and outputs of the module, respectively. $W^{i,i-1}$ and $W_R^{i,i-1}$ are two different weight matrixes. $\circ$ denotes Hadamard product.

Figure 6 (d) shows ResDD-AW code for MNIST dataset in references [34]. The number of modules can be adjusted according to the task.

6) *Shared weight of Dendrite module*: If convolution is the shared weight of the “plane”, then this section denotes the shared weight of depth.

This section is described in the form of *Python* code (example).

Fig. 11 shows the DD module code for MNIST (example).

Fig. 12 shows the Shared ResDD module code for MNIST (example).

For ease of description, this paper describes the Gang neuron with the best-represented architecture ($\circ X$). In use, you can replace ($\circ X$) by ($\circ A^{0|1|2|\dots|i-1}$) or using ResDD, DenseDD, ResGangneuron, or DenseGangneuron (see the ResGangneuron section and the DenseGangneuron section).

```
class ResddNet_X(nn.Module):
```

(a)

```
def __init__(self):
    super(ResddNet_X, self).__init__()

    # xw+b
    self.fc0 = nn.Linear(28*28, 256, bias=False)
    self.dd = nn.Linear(256, 256, bias=False)
    self.dd2 = nn.Linear(256, 256, bias=False)
    self.dd3 = nn.Linear(256, 256, bias=False)
    self.dd4 = nn.Linear(256, 256, bias=False)

    self.fc2 = nn.Linear(256, 10, bias=False)

def forward(self, x):
    # x: [b, 1, 28, 28]
    x = self.fc0(x)
    c = x
    # h1 = x@w1*x

    g=self.dd(x)
    x=g*c+g

    g2=self.dd2(x)
    x=g2*c+g2

    g3=self.dd3(x)
    x=g3*c+g3

    g4=self.dd4(x)
    x=g4*c+g4

    x = self.fc2(x)
    return x
```

```
class ResddNet_A(nn.Module):
```

(b)

```
def __init__(self):
    super(ResddNet_A, self).__init__()

    # xw+b
    self.fc0 = nn.Linear(28*28, 256, bias=False)
    self.dd = nn.Linear(256, 256, bias=False)
    self.dd2 = nn.Linear(256, 256, bias=False)
    self.dd3 = nn.Linear(256, 256, bias=False)
    self.dd4 = nn.Linear(256, 256, bias=False)

    self.fc2 = nn.Linear(256, 10, bias=False)

def forward(self, x):
    # x: [b, 1, 28, 28]
    x = self.fc0(x)
    #c = x
    # h1 = x@w1*x

    g=self.dd(x)
    x=g*x+g

    g2=self.dd2(x)
    x=g2*x+g2

    g3=self.dd3(x)
    x=g3*x+g3

    g4=self.dd4(x)
    x=g4*x+g4

    x = self.fc2(x)
    return x
```

```
class ResddNet_XW(nn.Module):
```

(c)

```
def __init__(self):
    super(ResddNet_XW, self).__init__()

    # xw+b
    self.fc0 = nn.Linear(28*28, 256, bias=False)
    self.dd = nn.Linear(256, 256, bias=False)
    self.dd_1 = nn.Linear(256, 256, bias=False)
    self.dd2 = nn.Linear(256, 256, bias=False)
    self.dd2_1 = nn.Linear(256, 256, bias=False)
    self.dd3 = nn.Linear(256, 256, bias=False)
    self.dd3_1 = nn.Linear(256, 256, bias=False)
    self.dd4 = nn.Linear(256, 256, bias=False)
    self.dd4_1 = nn.Linear(256, 256, bias=False)

    self.fc2 = nn.Linear(256, 10, bias=False)

def forward(self, x):
    # x: [b, 1, 28, 28]
    x = self.fc0(x)
    c = x
    # h1 = x@w1*x

    x=self.dd(x)*c+self.dd_1(x)

    x=self.dd2(x)*c+self.dd2_1(x)

    x=self.dd3(x)*c+self.dd3_1(x)

    x=self.dd4(x)*c+self.dd4_1(x)

    x = self.fc2(x)
    return x
```

```
class ResddNet_AW(nn.Module):
```

(d)

```
def __init__(self):
    super(ResddNet_AW, self).__init__()

    # xw+b
    self.fc0 = nn.Linear(28*28, 256, bias=False)
    self.dd = nn.Linear(256, 256, bias=False)
    self.dd_1 = nn.Linear(256, 256, bias=False)
    self.dd2 = nn.Linear(256, 256, bias=False)
    self.dd2_1 = nn.Linear(256, 256, bias=False)
    self.dd3 = nn.Linear(256, 256, bias=False)
    self.dd3_1 = nn.Linear(256, 256, bias=False)
    self.dd4 = nn.Linear(256, 256, bias=False)
    self.dd4_1 = nn.Linear(256, 256, bias=False)

    self.fc2 = nn.Linear(256, 10, bias=False)

def forward(self, x):
    # x: [b, 1, 28, 28]
    x = self.fc0(x)
    #c = x
    # h1 = x@w1*x

    x=self.dd(x)*x+self.dd_1(x)

    x=self.dd2(x)*x+self.dd2_1(x)

    x=self.dd3(x)*x+self.dd3_1(x)

    x=self.dd4(x)*x+self.dd4_1(x)

    x = self.fc2(x)
    return x
```

Fig. 6. ResDD code for MNIST dataset in references [34]. Figure(a) shows $A^i = W^{i,i-1}A^{i-1} \circ X + W^{i,i-1}A^{i-1}$. Figure(b) shows $A^i = W^{i,i-1}A^{i-1} \circ A^{i-1} + W^{i,i-1}A^{i-1}$. Figure(c) shows $A^i = W^{i,i-1}A^{i-1} \circ X + W_R^{i,i-1}A^{i-1}$. Figure(d) shows $A^i = W^{i,i-1}A^{i-1} \circ A^{i-1} + W_R^{i,i-1}A^{i-1}$.

Fig. 7. ResDD-X code for MNIST dataset in references [34].

Fig. 8. ResDD-XW code for MNIST dataset in references [34].

7) *Architecture-Gang neuron with one module*: Figure 2 shows the Gang artificial neuron with one module. The architecture of Gang artificial neuron can be represented as

$$\begin{cases} A = W^{10} X \circ X \\ y = f(W^{21} A) \end{cases} \quad (12)$$

Where W^{10} and W^{21} are the weight matrix, $\{1, x_1, x_2, \dots, x_n\} = X$. $\circ$ denotes Hadamard product. In the back-propagation procedure, the weight matrix can be updated with,

$$W_{new}^{21} = W^{21} - \eta^{21} \frac{\partial E}{\partial W^{21}} \quad (13)$$

$$W_{new}^{10} = W^{10} - \eta^{10} \frac{\partial A}{\partial W^{10}} \frac{\partial E}{\partial A} \quad (14)$$

Where W_{new}^{21} and W_{new}^{10} are the updated weight matrix for W^{21} and W^{10} respectively, and E is the cost function or errors from the later layer in back-propagation. η^{21} and η^{10} are the learning rates. The learning rates of “dendrite” and cell body can be different.

Fig. 9. ResDD-A code for MNIST dataset in references [34].

Fig. 10. ResDD-AW code for MNIST dataset in references [34].

Fig. 11. DD module code for MNIST dataset. (a) Unshared DD module.(b) Shared DD module.

8) *Architecture-Gang “neuron” of unicellular organism:* In nature, there are unicellular organisms, such as bacteria and fungus. The first organisms to appear on Earth were presumably single-celled [43]. Thus, figure 3 shows the Gang artificial “neuron” with multiple interactions to imitate unicellular organisms. The architecture can realize excellent nonlinear expression [34], [41] and can be represented as follows.

$$\begin{cases} A^1 = W^{10} X \circ X \\ A^2 = W^{21} A^1 \circ X \\ \vdots \\ A^l = W^{l,l-1} A^{l-1} \circ X \\ y = f(W^{l+1,l} A^l) \end{cases} \quad (15)$$

Where l represents the number of the dendrite modules. $\circ$ denotes Hadamard product.

9) *Architecture-simplified Gang neuron:* The weight matrix of the cell body W^{21} in Fig.2 or $W^{l+1,l}$ in Fig. 3 is defined as a constant matrix where all its elements are 1. [Considering batch training, we describes it as a matrix rather than a vector.]

Thus, the architecture in figure 2 can be represented as

$$y = f(W_{ones}(W^{10} X \circ X)) \quad (16)$$

And the architecture in figure 3 can be represented as

$$\begin{cases} A^1 = W^{10} X \circ X \\ A^2 = W^{21} A^1 \circ X \\ \vdots \\ A^l = W^{l,l-1} A^{l-1} \circ X \\ y = f(W_{ones} A^l) \end{cases} \quad (17)$$

III. ANNs USING GANG ARTIFICIAL NEURON

A. Single-layer network

Figure 13 shows the single-layer network. This architecture imitates the intricate dendrite of multiple neurons. Multiple neurons may obtain the same signal [32], [33], [43]. “ $f(\dots)$ ” in Fig. 13 can be set as the function that you plan to use for application. For example, when “ $f(\dots)$ ” is set as *softmax*, it can be used for classification.

```
class ResddNet_X(nn.Module):
```

```
    def __init__(self):
        super(ResddNet_X, self).__init__()
```

```
        # xw+b
        self.fc0 = nn.Linear(28*28, 256, bias=False)
        self.dd = nn.Linear(256, 256, bias=False)
```

```
        self.fc2 = nn.Linear(256, 10, bias=False)
```

```
    def forward(self, x):
```

```
        # x: [b, 1, 28, 28]
```

```
        x = self.fc0(x)
```

```
        c = x
```

```
        # h1 = x@w1*x
```

```
        for i in range(4):
```

```
            g=self.dd(x)
```

```
            x=g*c+g
```

```
        x = self.fc2(x)
```

```
        return x
```

```
class ResddNet_A(nn.Module):
```

```
    def __init__(self):
        super(ResddNet_A, self).__init__()
```

```
        # xw+b
```

```
        self.fc0 = nn.Linear(28*28, 256, bias=False)
```

```
        self.dd = nn.Linear(256, 256, bias=False)
```

```
        self.fc2 = nn.Linear(256, 10, bias=False)
```

```
    def forward(self, x):
```

```
        # x: [b, 1, 28, 28]
```

```
        x = self.fc0(x)
```

```
        #c = x
```

```
        # h1 = x@w1*x
```

```
        for i in range(4):
```

```
            g=self.dd(x)
```

```
            x=g*x+g
```

```
        x = self.fc2(x)
```

```
        return x
```

```
class ResddNet_XW(nn.Module):
```

```
    def __init__(self):
        super(ResddNet_XW, self).__init__()
```

```
        # xw+b
```

```
        self.fc0 = nn.Linear(28*28, 256, bias=False)
```

```
        self.dd = nn.Linear(256, 256, bias=False)
```

```
        self.dd_1 = nn.Linear(256, 256, bias=False)
```

```
        self.fc2 = nn.Linear(256, 10, bias=False)
```

```
    def forward(self, x):
```

```
        # x: [b, 1, 28, 28]
```

```
        x = self.fc0(x)
```

```
        c = x
```

```
        # h1 = x@w1*x
```

```
        for i in range(4):
```

```
            x=self.dd(x)*c+self.dd_1(x)
```

```
        x = self.fc2(x)
```

```
        return x
```

```
class ResddNet_AW(nn.Module):
```

```
    def __init__(self):
        super(ResddNet_AW, self).__init__()
```

```
        # xw+b
```

```
        self.fc0 = nn.Linear(28*28, 256, bias=False)
```

```
        self.dd = nn.Linear(256, 256, bias=False)
```

```
        self.dd_1 = nn.Linear(256, 256, bias=False)
```

```
        self.fc2 = nn.Linear(256, 10, bias=False)
```

```
    def forward(self, x):
```

```
        # x: [b, 1, 28, 28]
```

```
        x = self.fc0(x)
```

```
        #c = x
```

```
        # h1 = x@w1*x
```

```
        for i in range(4):
```

```
            x=self.dd(x)*x+self.dd_1(x)
```

```
        x = self.fc2(x)
```

```
        return x
```

Fig. 12. Shared ResDD module code for MNIST dataset.

Fig. 13. Single-layer network.

Fig. 14. MLP using Gang artificial neuron.

B. Multi-layer perceptron (MLP) using Gang artificial neuron

MLP is the most typical neural network. Here, Fig. 14 shows a MLP using Gang artificial neuron with one module. The forward propagation can be represented as follows.

$$\begin{cases} A_h = W_h^{10} X \circ X \\ H_h = f(W_h^{21} A_h) \end{cases} \quad (18)$$

$$\begin{cases} A_o = W_o^{10} H_h \circ H_h \\ Y_o = f(W_o^{21} A_o) \end{cases} \quad (19)$$

Where $H = [1; H_h]$, W_h^{10} , W_h^{21} , W_o^{10} , and W_o^{21} are the weight matrix, $\{1, x_1, x_2, \dots, x_n\} = X$. $\circ$ denotes Hadamard product. The backpropagation can be represented according to formula 13 and 14.

Additionally, when the simplest form of Gang artificial neuron is used, MLP can be represented as follows.

$$\begin{cases} H_h = f(\sum (W_h^{10} X \circ X)) \\ Y_o = f(\sum (W_o^{10} H_h \circ H_h)) \end{cases} \quad (20)$$

C. Information fusion network

Figure 15 shows the information fusion of different neurons. This information interaction is common in the biological brain [38]–[40]. The architecture in Fig. 15 can be expressed as follows.

Fig. 15. Information fusion network.

$$\left\{ \begin{array}{l} A_x^1 = W_x^{10} X \circ X \\ A_x^2 = W_x^{21} A_x^1 \circ X \\ \vdots \\ A_x^l = W_x^{l,l-1} A_x^{l-1} \circ X \\ h_1 = f(W_x^{l+1,l} A_x^l) \\ \vdots \end{array} \right. \quad (21)$$

$$\left\{ \begin{array}{l} A_z^1 = W_z^{10} Z \circ Z \\ A_z^2 = W_z^{21} A_z^1 \circ Z \\ \vdots \\ A_z^l = W_z^{l,l-1} A_z^{l-1} \circ Z \\ h_e = f(W_z^{l+1,l} A_z^l) \end{array} \right. \quad (22)$$

Where $\{1, x_1, x_2, \dots, x_n\} = X$, $\{1, z_1, z_2, \dots, z_k\} = Z$, and $\{1, h_1, h_2, \dots, h_e\} = H$. $\circ$ denotes Hadamard product. The network architecture can be trained by error backpropagation according to formula 13 and 14.

In fact, these network architectures using Gang neuron can be trained by *PyTorch* expediently. [Note that the inputs and outputs are normalized to $[-1, 1]$.] The architectures presented in this paper are like some plugins and can be used to other larger architectures where you want to design to improve networks' mapping capacity.

Fig. 16. ResGangneuron module. ‘|’ denotes ‘or’. The dotted line represents any of them.

```
class ResGangneuron_A(nn.Module):
    def __init__(self):
        super(ResGangneuron_A, self).__init__()

        # kwab
        self.fc0 = nn.Linear(3*32*32, 256, bias=False)
        self.dd = nn.Linear(256, 256, bias=False)
        self.dd2 = nn.Linear(256, 256, bias=False)
        self.dd3 = nn.Linear(256, 256, bias=False)
        self.dd4 = nn.Linear(256, 256, bias=False)

        self.c1 = nn.Linear(256, 256, bias=False)
        self.c2 = nn.Linear(256, 256, bias=False)
        self.c3 = nn.Linear(256, 256, bias=False)
        self.c4 = nn.Linear(256, 256, bias=False)

        self.fc2 = nn.Linear(256, 10, bias=False)

    def forward(self, x):
        # x: [b, 1, 28, 28]
        x = self.fc0(x)
        # c = x
        # h1 = x@w1*x

        g=self.dd(x)
        x=g*x+g
        x= F.relu(self.c1(x))+x

        g2=self.dd2(x)
        x=g2*x+g2
        x= F.relu(self.c2(x))+x

        g3=self.dd3(x)
        x=g3*x+g3
        x= F.relu(self.c3(x))+x

        g4=self.dd4(x)
        x=g4*x+g4
        x= F.relu(self.c4(x))+x

        x = self.fc2(x)
        return x
```

Fig. 17. Approximation using ResGangneuron (RGN-A) for CIFAR-10 dataset (example). Training loss minimum: 2.117×10^{-4} . Note that this is just a simple example without tuning.

IV. SOME TYPICAL ANNs ARCHITECTURE USING GANG NEURON

A. ResGangneuron (RGN)

ResGangneuron module is the combination of ResDD and residual cell body (see Fig. 16). The formula can be expressed as follows.

$$\begin{cases} A^1 = W^{10} A^0 \circ A^0 + W^{10} A^0 \\ \vdots \\ A^i = W^{i,i-1} A^{i-1} \circ A^{0|1|2|\dots|i-1} + W^{i,i-1} A^{i-1} \\ Y = f(W^{i+1,i} A^i) + A^i \end{cases} \quad (23)$$

Where A^{i-1} and A^i are the inputs and outputs of the module, respectively. $A^{0|1|2|\dots|i-1}$ is any of A . $W^{i,i-1}$ is a weight matrix. i represents the number of the dendrite modules. $\circ$ denotes Hadamard product.

Fig. 17 shows a simple example of using four ResGangneuron.

B. DenseGangneuron (DGN)

Fig. 18. Generalized Dense Dendrite module. ‘|’ denotes ‘or’. The dotted line represents any of them.

1) *Dense Dendrite module (DenseDD)* : In order to make ResDD denser, Generalized Dense Dendrite module can be designed (see Fig. 18). The module is represented as follows.

$$A^i = W^{i,i-1} A^{i-1} \circ A^{0|1|2|\dots|i-1} + X \quad (24)$$

Where A^{i-1} and A^i are the inputs and outputs of the module, respectively. $A^{0|1|2|\dots|i-1}$ is any of A . $W^{i,i-1}$ is a weight matrix. X denotes the DD’s inputs. $\circ$ denotes Hadamard product.

Additionally, similar to ResDD module, the code for the special case is shown in Fig. 19.

2) *DenseGangneuron*: DenseGangneuron module is the combination of DenseDD and Dense cell body. Here, I show two special examples.

Example 1 (DenseGangneuron-X) can be expressed as follows.(see Fig. 20)

$$\begin{cases} A^1 = W^{10} A^0 \circ A^0 + X \\ \vdots \\ A^i = W^{i,i-1} A^{i-1} \circ A^{0|1|2|\dots|i-1} + X \\ Y = f(W^{i+1,i} A^i) + X \end{cases} \quad (25)$$

Where A^{i-1} and A^i are the inputs and outputs of the module, respectively. $A^{0|1|2|\dots|i-1}$ is any of A . $W^{i,i-1}$ is a weight matrix. i represents the number of the dendrite modules. X denotes the DD’s inputs. $\circ$ denotes Hadamard product.

Example 2 (DenseGangneuron-A) can be expressed as follows.(see Fig. 21)

$$\begin{cases} A^1 = W^{10} A^0 \circ A^0 + X \\ \vdots \\ A^i = W^{i,i-1} A^{i-1} \circ A^{0|1|2|\dots|i-1} + X \\ Y = f(W^{i+1,i} A^i) + A^i \end{cases} \quad (26)$$

Where A^{i-1} and A^i are the inputs and outputs of the module, respectively. $A^{0|1|2|\dots|i-1}$ is any of A . $W^{i,i-1}$ is a weight matrix. i represents the number of the dendrite modules. X denotes the DD’s inputs. $\circ$ denotes Hadamard product.

As an example, I show the code when $\circ A^{i-1}$ is used in the dendrite module.(see Fig. 22)

C. Convolution layer using Gang neuron

Figure 23 compares the architecture of using a Gang neuron with a traditional neuron in the convolution layer (an example). The Gang neuron extracts information about the interactions among inputs.

The network using Gang neurons can delete the Fully-connected Layers of traditional networks. In other words, the parameters of the Fully-connected Layer are assigned to a single neuron, which reduces parameters of network for the same mapping capacity. **The Fully-connected Layers do not exist in human brain.**

For example, the number of parameters in the Fully-connected Layer is 10000×100 . When the parameters are assigned to a single neuron, the number of parameters can be $100 \times 10 \times 10$. [100 denotes the number of Gang neurons.]

D. Schematic diagram of RNN using Gang neuron

Figure 24 compares the architecture of using a Gang neuron with a traditional neuron in the schematic diagram of RNN. The Gang neuron extracts information on the interactions between inputs about time. Where x represents the currents input and h represents the outputs of the previous time.

Fig. 19. Dense Dendrite module code (example). (a) and (b) are with the form of the unshared weights. (c) and (d) are with the form of the shared weights.

Fig. 20. ResGangneuron-X module. ‘|’ denotes ‘or’. The dotted line represents any of them.

Fig. 21. ResGangneuron-A module. ‘|’ denotes ‘or’. The dotted line represents any of them.

```
class DenseGangneuron_AX(nn.Module):

    def __init__(self):
        super(DenseGangneuron_AX, self).__init__()

        # xw+b
        self.fc0 = nn.Linear(3*32*32, 256, bias=False)
        self.dd = nn.Linear(256, 256, bias=False)
        self.dd2 = nn.Linear(256, 256, bias=False)
        self.dd3 = nn.Linear(256, 256, bias=False)
        self.dd4 = nn.Linear(256, 256, bias=False)

        self.c1 = nn.Linear(256, 256, bias=False)
        self.c2 = nn.Linear(256, 256, bias=False)
        self.c3 = nn.Linear(256, 256, bias=False)
        self.c4 = nn.Linear(256, 256, bias=False)

        self.fc2 = nn.Linear(256, 10, bias=False)

    def forward(self, x):
        # x: [b, 1, 28, 28]
        x = self.fc0(x)
        c = x
        # h1 = x@w1*x

        g=self.dd(x)
        x=g*x+c

        x= F.relu(self.c1(x))+c

        g2=self.dd2(x)
        x=g2*x+c

        x= F.relu(self.c2(x))+c

        g3=self.dd3(x)
        x=g3*x+c

        x= F.relu(self.c3(x))+c

        g4=self.dd4(x)
        x=g4*x+c

        x= F.relu(self.c4(x))+c

        x = self.fc2(x)
        return x
```

```
class DenseGangneuron_AA(nn.Module):

    def __init__(self):
        super(DenseGangneuron_AA, self).__init__()

        # xw+b
        self.fc0 = nn.Linear(3*32*32, 256, bias=False)
        self.dd = nn.Linear(256, 256, bias=False)
        self.dd2 = nn.Linear(256, 256, bias=False)
        self.dd3 = nn.Linear(256, 256, bias=False)
        self.dd4 = nn.Linear(256, 256, bias=False)

        self.c1 = nn.Linear(256, 256, bias=False)
        self.c2 = nn.Linear(256, 256, bias=False)
        self.c3 = nn.Linear(256, 256, bias=False)
        self.c4 = nn.Linear(256, 256, bias=False)

        self.fc2 = nn.Linear(256, 10, bias=False)

    def forward(self, x):
        # x: [b, 1, 28, 28]
        x = self.fc0(x)
        c = x
        # h1 = x@w1*x

        g=self.dd(x)
        x=g*x+c

        x= F.relu(self.c1(x))+x

        g2=self.dd2(x)
        x=g2*x+c

        x= F.relu(self.c2(x))+x

        g3=self.dd3(x)
        x=g3*x+c

        x= F.relu(self.c3(x))+x

        g4=self.dd4(x)
        x=g4*x+c

        x= F.relu(self.c4(x))+x

        x = self.fc2(x)
        return x
```

Fig. 22. DenseGangneuron code (example). The left figure is DenseGangneuron-X. The right figure is DenseGangneuron-A.

Fig. 23. Convolution layer using Gang neuron or traditional neuron.

V. BIOMIMETIC ARCHITECTURE USING GANG NEURON

A. Module 1: Retina, attention(focus), and convolution-like scan

Fig. 25 shows a retina module. This module imitates the focus mechanism of the human eye when we observe things. At the focus, our eye extracts the multiple relationships of the picture, and then, the extracted texture is more and more rough along the direction of focus diffusion. Additionally, **Attention** in Fig. 25 can scan the entire picture, which happens to be similar to convolution. However, my architecture contains gradients related to focus, and the maximum receptive field can be set as the entire picture (Of course, you can also set a small range of receptive field with gradient). I show a 4th-order gradient in Fig. 25. The number of gradients can be adjusted by dendrite modules appropriately. For specific details, please see the specific formula in the Fig. 25.

B. Module 2: Population coding in brain

The dendrites of neurons are crisscrossed. In population coding of biological brain, the primary afferent fibre whose receptive field centre is closest to the point of stimulation will produce more action potentials than those on the periphery (Fig. 26, the green fibre has a greater action potential frequency than the blue fibres) [44].

Fig. 26 shows a biomimetic population coding module. Two points are important to emphasize.

1) Biomimetic dendrite is staggered; 2) When we design a network by multilayer Module 2 in Fig. 26, the dimension of output decreases as the layer.

C. Application methods

This paper shows three usage.

- 1) In the existing networks, the convolutional layers can be replaced by module 1. (Compared with the existing convolution, module 1 has a larger receptive field and can extract the position information.)
- 2) Multi-layer modules 2 are interconnected to build a new neural network.
- 3) Module 1 and Module 2 are used together. There is a special architecture imitating the biological brain. The first layer is designed by module 1, and all the later layers are designed by module 2.

Fig. 24. Schematic diagram of RNN using Gang neuron.

Fig. 25. Module 1: Retina, attention, and convolution-like scan. The gap between the 'squares' only aims to distinguish the 'squares' in drawing. The calculation of each color block consists of two steps (see Fig. 23): 1) Transform X into a 1-dimensional input; 2) use the dendrite modules. In terms of the activation function, you can use what you want, such as *relu*. Besides, linear activation function (e.g., $Y = WA$) also can be used because the dendrite module has the nonlinear mapping capacity. Additionally, as an example, the figure shows the calculation process of one channel. For more channels, each channel can be calculated separately. Then, the outputs of channels are averaged in the same position. Note that the above formulas express one operation like a convolution kernel. The later layer is generated by a convolution-like scan.

Fig. 26. Module 2: Population coding in brain. Refer to Fig. 15 for the calculation of neurons. Besides, as an example, the figure shows the calculation process of one channel. For more channels, each channel can be calculated separately. Then, the outputs of channels are averaged in the same position.

VI. DEVELOPMENT AND APPLICATION OF GANG NEURONS

A. Development

No new development has emerged yet. Therefore, this part is temporarily empty.

B. Application

(a) *The model using Gang neurons detects myocardial infarction based on ECG signal measured by the flexible sensor.*

On July 5, 2021, I found a conference paper that uses Gang neurons to detect myocardial infarction based on ECG signal measured by the flexible sensor [45]. [*This paper [45] used Gang neuron, but the paper did not cite Gang neuron; I have been contacted the authors. Considering that this paper is a conference paper and clearly cited DD [34], I did not request to amend their paper. The author has promised that subsequent related papers will cited Gang neuron in accordance with License: CC BY-NC-SA 4.0.*] Next, let's interpret this paper.

The paper showed the module $X = (WX) \circ (WX)$. This module is actually a linear module and DD module(I proposed it in "Fig. 11. Classification experiments" and "Fig. 15. Additional figure. Python code." of DD [45] and Fig. 11 Gang neuron (this paper)). It's just that they wrote it together. See the following formula (a linear module and DD module in Fig. 27).

A linear module:

$$Y = W^{10}X \quad (27)$$

One DD module:

$$X' = W^{21}Y \circ Y \quad (28)$$

Let's combine Eq 27 and Eq 28. The following equation is obtained.

$$X' = (W^{21}W^{10}X) \circ (W^{10}X) \quad (29)$$

Eq 29 is the module in paper [45]. *Gang neuron (the combination of dendrite in this paper and function $f(\cdot)$) was published on June 16, 2020. I re-emphasized the citation on the first page of the paper.*

In addition, I don't recommend considering one linear module and one DD module as a whole (see Fig. 28, like the paper [45]). In this way, many linear modules are redundant after being connected in series. I think the use of DD should be more flexible. When the user wants to reduce or increase the dimension, just insert a linear module. We can use one linear module and two DD modules, one linear module and three DD modules, or one linear module and three DD modules.

Regarding the fusion of two different dimensions, we can use the strategy used in Fig. 24 and Fig. 15. For example, $x_1 = [0.1 \ 0.2 \ 0.3]$, $x_2 = [0.1 \ 0.2]$. We can concatenate x_1 and x_2 to get $X = [x_1 \ x_2] = [0.1 \ 0.2 \ 0.3 \ 0.1 \ 0.2]$, and then we can use a linear module to reduce the dimensionality and use Gang neurons(Choose the most suitable form in this paper).

Fig. 27. A linear module and DD module. I proposed it in “Fig. 11. Classification experiments” and “Fig. 15. Additional figure. Python code.” of DD [45] and Fig. 11 Gang neuron (this paper).

Fig. 28. I don't recommend considering one linear module and one DD module as a whole (like the paper [45]). In this way, many linear modules are redundant after being connected in series. I think the use of DD should be more flexible. When the user wants to reduce or increase the dimension, just insert a linear module.

VII. TIPS OF GANG NEURON

A. Avoiding curse of dimensionality and improving generalization ability

1) *Adjusting the range of inputs:* When the absolute value of the dendrites' inputs is less than 1, there is an inhibitory effect on higher-order terms.

$$\lim_{n \rightarrow +\infty} x^n = 0 \quad (30)$$

Where $|x| < 1$, $n \in \mathbb{N}^+$. Simultaneously, $i, j \in \mathbb{N}^+$, when $i < j$, $x^i > x^j$. The simulation example is shown in Fig. 29. The degree of suppression of higher-order terms can be changed by adjusting the range of x (see Fig. 29).

2) *Regularization:* Regularization (e.g., L1-regularization [46]) is used to balance the weight distribution, which makes Gang neuron more inclined to approximate the data set by using low-order terms. Gang neuron (e.g., ResDD) uses lower-order terms as much as possible, and then higher-order terms are used to approximate details, which is consistent with what we expect. Here, I show an example of L1-regularization (see Fig. 30).

VIII. SOME THEORIES FOR UNDERSTANDING GANG NEURONS

The following content is only to help readers understand Gang neurons and does not contain strict proof.

A. Polynomial, Convolution, and Dendrite in this paper

Understanding dendrites for feature extraction from the perspective of convolution

Convolution:

The convolution of f and g is written $f * g$, denoting the operator with the symbol $*$. It is defined as the integral of the product of the two functions after one is reversed and shifted. As such, it is a particular kind of integral transform:

$$\int_{-\infty}^{\infty} f(\tau)g(t - \tau)d\tau \quad (31)$$

We can express it in discrete form.

$$\sum f[k]g[x - k] \quad (32)$$

Polynomial multiplication:

E.g: $(x \times x + 3 \times x + 2)(2 \times x + 5)$

The calculation process is as follows:

$$\begin{aligned}
 & (x \times x + 3 \times x + 2)(2 \times x + 5) \\
 &= (x \times x + 3 \times x + 2) \times 2 \times x + (x \times x + 3 \times x + 2) \times 5 \\
 &= 2 \times x \times x \times x + 3 \times 2 \times x \times x + 2 \times 2 \times x + 5 \times x \times x + 3 \times 5 \times x + 10
 \end{aligned} \quad (33)$$

Then, merge the same terms:

$$\begin{array}{r}
2x \times x \times x \\
3 \times 2 + 1 \times 5x \times x \\
2 \times 2 + 3 \times 5x \\
2 \times 5 \\
- - - - \\
2 \times x \times x \times x + 11 \times x \times x + 19 \times x + 10
\end{array} \tag{34}$$

The polynomial can be regarded as a vector space, and its basis can be selected as $\{1, x, x \times x, x \times x \times x, \dots\}$. Then any polynomial can correspond to a coordinate vector in an infinite-dimensional space.

$(x \times x + 3 \times x + 2)$ corresponds to $(1 \ 3 \ 2)$; $(2 \times x + 5)$ corresponds to $(2 \ 5)$.

We perform coordinate calculation according to the definition of convolution:

$(1 \ 3 \ 2) * (2 \ 5)$ is expressed as follows.

$$\begin{array}{r}
2 \quad 3 \quad 1 \\
- \quad - \quad 2 \quad 5 \\
- - - - - \\
 2
\end{array} \tag{35}$$

$$\begin{array}{r}
2 \quad 3 \quad 1 \\
- \quad 2 \quad 5 \\
- - - - - \\
 6 + 5 = 11
\end{array} \tag{36}$$

$$\begin{array}{r}
2 \quad 3 \quad 1 \\
2 \quad 5 \\
- - - - - \\
4 + 15 = 19
\end{array} \tag{37}$$

$$\begin{array}{r}
- \quad 2 \quad 3 \quad 1 \\
2 \quad 5 \\
- - - - - \\
10
\end{array} \tag{38}$$

e.i., $(1 \ 3 \ 2) * (2 \ 5) = (2 \ 11 \ 19 \ 10)$

Now we look at Eq.34: $(x \times x + 3 \times x + 2)(2 \times x + 5) = 2 \times x \times x \times x + 11 \times x \times x + 19 \times x + 10$. Polynomial multiplication is equivalent to convolution of coefficient vectors. Therefore, we can also understand the calculation of Dendrite modules from the perspective of convolution.

The above example comes from the Internet.

IX. DISCUSSION AND CONCLUSION

In this paper, I compared the development of artificial neural networks with biological neural networks. As expected, the direction of their research fields is different. Researchers in ANNs pay attention to the architectures for different applications, while researchers in biology focus on exploring the neural mechanisms. Both directions achieve success. It may be time to learn from each other.

As we know, artificial neural networks are inspired by biological neural networks. Seventy years ago, people designed artificial neurons by the knowledge about biological neurons at that time. Today, we have a better understanding of how the work of neurons, especially dendrites.

The traditional artificial neurons ignore a fact that dendrites participate in pre-calculation in a biological neuron or biological neural network. More specifically, biological dendrites play a role in the pre-processing to the interaction information of input data. Two examples can be mentioned in this regard. In the task of understanding a picture, biological dendrites play a role in extracting the relationship between parts of an input-picture. In the task of understanding an article, biological dendrites play a role in extracting the relationship between parts of an article.

Therefore, this paper improved the neuron of ANNs by introducing the dendrite module and presented a novel neuron, named Gang neuron.

Fig. 29. $\lim_{n \rightarrow +\infty} x^n = 0$. (a) $x \in [0, 0.99]$. (b) $x \in [0, 0.5]$.

X. OUTLOOK

This paper shows some basis architecture using Gang neuron. In fact, these network architectures using Gang neuron can be trained by *PyTorch* expediently. [Note that the inputs and outputs are normalized to $[-1, 1]$.] These architectures in this paper are like some plugins. They can be used to other larger architectures where you want to design to improve the mapping capacity of networks.

There is an interesting phenomenon. Some data augmentation methods, such as flipping and zooming in or out pictures, improve networks' performance. I believe this shows that the existing networks can not fully extract some simple information, such as flipping and zooming in or out pictures. However, our brains do this well.

```
# Regularization L1
Regularization_loss=0

for param in net.parameters():
    Regularization_loss += torch.sum(torch.abs(param))

loss = F.mse_loss(out, y_onehot)+a*Regularization_loss
```

Fig. 30. L1-regularization (Example, *PyTorch* code).

REFERENCES

- [1] T. P. Lillicrap, A. Santoro, L. Marris, C. J. Akerman, and G. Hinton, "Backpropagation and the brain," *Nature Reviews Neuroscience*, vol. 21, no. 6, pp. 335–346, 2020.
- [2] A. M. Zador, "A critique of pure learning and what artificial neural networks can learn from animal brains," *Nature Communications*, vol. 10, p. 7, 2019.
- [3] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.
- [4] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
- [5] S. Sengupta, S. Basak, P. Saikia, S. Paul, V. Tsalavoutis, F. Atiah, V. Ravi, and A. Peters, "A review of deep learning with special emphasis on architectures, applications and recent trends," *Knowledge-Based Systems*, vol. 194, p. 33, 2020.
- [6] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the Ieee*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the Acm*, vol. 60, no. 6, pp. 84–90, 2017.
- [8] T. Mikolov, M. Karafiat, L. Burget, J. Cernocký, and S. Khudanpur, "Recurrent neural network based language model," in *conference of the international speech communication association*, pp. 1045–1048, 2010.
- [9] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [10] I. Goodfellow, J. Pougetabadi, M. Mirza, B. Xu, D. Wardefarley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *neural information processing systems*, pp. 2672–2680, 2014.
- [11] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *computer vision and pattern recognition*, 2014.
- [12] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *computer vision and pattern recognition*, pp. 1–9, 2015.
- [13] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *computer vision and pattern recognition*, pp. 2818–2826, 2016.
- [14] Y. Bengio, "Learning deep architectures for ai," *Foundations and Trends in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.
- [15] S. Xie, R. Girshick, P. Dollar, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *computer vision and pattern recognition*, pp. 5987–5995, 2017.
- [16] S. Woo, J. Park, J. Lee, and I. S. Kweon, "Cbam: Convolutional block attention module," in *european conference on computer vision*, pp. 3–19, 2018.
- [17] Y. Hu, G. Wen, M. Luo, D. Dai, and J. Ma, "Competitive inner-imaging squeeze and excitation for residual network," *arXiv: Computer Vision and Pattern Recognition*, 2018.
- [18] K. Gregor, I. Danihelka, A. Graves, D. J. Rezende, and D. Wierstra, "Draw: A recurrent neural network for image generation," *arXiv: Computer Vision and Pattern Recognition*, 2015.
- [19] N. Kalchbrenner, I. Danihelka, and A. Graves, "Grid long short-term memory," *arXiv: Neural and Evolutionary Computing*, 2015.
- [20] L. Jing, C. Gulcehre, P. Peurifoy, Y. Shen, M. Tegmark, M. Soljacic, and Y. Bengio, "Gated orthogonal recurrent units: On learning to forget," *Neural Computation*, vol. 31, no. 4, pp. 765–783, 2019.
- [21] F. Belletti, A. Beutel, S. Jain, and E. H. Chi, "Factorized recurrent neural architectures for longer range dependence," in *international conference on artificial intelligence and statistics*, pp. 1522–1530, 2018.
- [22] Y. Bengio, N. Boulangerlewandowski, and R. Pascanu, "Advances in optimizing recurrent networks," in *international conference on acoustics, speech, and signal processing*, pp. 8624–8628, 2013.
- [23] G. E. Dahl, T. N. Sainath, G. E. Hinton, and Ieee, *IMPROVING DEEP NEURAL NETWORKS FOR LVCSR USING RECTIFIED LINEAR UNITS AND DROPOUT*, pp. 8609–8613. International Conference on Acoustics Speech and Signal Processing ICASSP, New York: Ieee, 2013.
- [24] D. Mishkin and J. Matas, "All you need is a good init," in *international conference on learning representations*, 2015.
- [25] D. Sussillo and L. F. Abbott, "Random walk initialization for training very deep feedforward networks," *arXiv: Neural and Evolutionary Computing*, 2014.
- [26] Z. Shi, S. Zhang, J. Yuan, B. Zhu, Y. Jiang, X. Shen, and Y. Wang, "Spatiotemporal summation and correlation mimicked in a four-emitter light-induced artificial synapse," *Scientific Reports*, vol. 8, no. 1, pp. 2159–2159, 2018.
- [27] J. Defelipe, L. Alonsonanclares, and J. I. Arellano, "Microstructure of the neocortex: Comparative aspects," *Journal of Neurocytology*, vol. 31, no. 3, pp. 299–316, 2002.

- [28] S. Sardi, R. Vardi, A. Sheinin, A. Goldental, and I. Kanter, "New types of experiments reveal that a neuron functions as multiple independent threshold units," *Scientific Reports*, vol. 7, no. 1, pp. 18036–18036, 2017.
- [29] Y. Timofeeva, S. Coombes, and D. Michieletto, "Gap junctions, dendrites and resonances: A recipe for tuning network dynamics," *Journal of Mathematical Neuroscience*, vol. 3, no. 1, pp. 15–15, 2013.
- [30] H. Mohan, M. B. Verhoog, K. K. Doreswamy, G. Eyal, R. Aardse, B. Lodder, N. A. Goriounova, B. Asamoah, A. B. C. B. Brakspear, and C. Groot, "Dendritic and axonal architecture of individual pyramidal neurons across layers of adult human neocortex," *Cerebral Cortex*, vol. 25, no. 12, pp. 4839–4853, 2015.
- [31] Y. Deitcher, G. Eyal, L. Kanari, M. B. Verhoog, G. A. A. Kahou, H. D. Mansvelder, C. P. J. De Kock, and I. Segev, "Comprehensive morpho-electrotonic analysis shows 2 distinct classes of I2 and I3 pyramidal neurons in human temporal cortex," *Cerebral Cortex*, vol. 27, no. 11, pp. 5398–5414, 2017.
- [32] A. Gidon, T. A. Zolnik, P. Fidyanski, F. Bolduan, A. Papoutsi, P. Poirazi, M. Holtkamp, I. Vida, and M. E. Larkum, "Dendritic action potentials and computation in human layer 2/3 cortical neurons," *Science*, vol. 367, no. 6473, pp. 83–, 2020.
- [33] P. Poirazi and A. Papoutsi, "Illuminating dendritic function with computational models," *Nature Reviews Neuroscience*, vol. 21, no. 6, pp. 303–321, 2020.
- [34] G. Liu and J. Wang, "Dendrite Net: A White-Box Module for Classification, Regression, and System Identification," *arXiv e-prints*, p. arXiv:2004.03955, Apr. 2020.
- [35] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.
- [36] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.
- [37] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.
- [38] M. E. Larkum, T. Nevian, M. Sandler, A. Polsky, and J. Schiller, "Synaptic integration in tuft dendrites of layer 5 pyramidal neurons: A new unifying principle," *Science*, vol. 325, no. 5941, pp. 756–760, 2009.
- [39] A. Polsky, B. W. Mel, and J. Schiller, "Computational subunits in thin dendrites of pyramidal cells," *Nature Neuroscience*, vol. 7, no. 6, pp. 621–627, 2004.
- [40] P. Poirazi, T. Brannon, and B. W. Mel, "Pyramidal neuron as two-layer neural network," *Neuron*, vol. 37, no. 6, pp. 989–999, 2003.
- [41] G. Liu and J. Wang, "A Polynomial Neural network with Controllable Precision and Human-Readable Topology II: Accelerated Approach Based on Expanded Layer," *arXiv e-prints*, p. arXiv:2006.02901, June 2020.
- [42] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.
- [43] A. V. Ritchie, S. Van Es, C. Fouquet, and P. Schaap, "From drought sensing to developmental control: Evolution of cyclic amp signaling in social amoebas," *Molecular Biology and Evolution*, vol. 25, no. 10, pp. 2109–2118, 2008.
- [44] J. Fitzakerley, "Sensory physiology," <https://www.d.umn.edu/~jfitzake/Lectures/DMED/SensoryPhysiology/GeneralPrinciples/LateralInhibition.html>, 2014.
- [45] X. Ma, X. Fu, Y. Sun, N. Wang, X. Ning, and Y. Gao, "Convolutional dendrite net detects myocardial infarction based on ecg signal measured by flexible sensor," in *2021 IEEE International Conference on Flexible and Printable Sensors and Systems (FLEPS)*, pp. 1–4, 2021.
- [46] C. C. Aggarwal *et al.*, *Neural networks and deep learning*. Springer, 2018.

Gang Liu :

Gang Liu is currently at the Institute of Robotics and Intelligent Systems, iHarbour Academy of Frontier Equipment, Xi'an Jiaotong University, China. Gang Liu is the Advisor to IOP Publishing and reviews for IEEE Transactions on Signal Processing(TSP), IEEE Transactions on Neural Systems and Rehabilitation Engineering(TNSRE), Journal of Neural Engineering(JNE), etc.

He presented Dendrite Net, Relation Spectrum, and Gang neuron. His current research interests include machine learning, deep learning, computer vision, natural language processing, and brain-computer interface. He is striving to improve the current ANNs in AI (CV or NLP) from roots.