**Is matrix neural network the alternative of convolutional neural network?**

Loc Nguyen

Loc Nguyen’s Academic Network, Vietnam

Email: ng\_phloc@yahoo.com

Homepage: www.locnguyen.net

**Abstract**

**Keywords:**

|  |  |
| --- | --- |
|  | () |

**1. Introduction**

Machine learning is classified into three subjects such as supervised learning, unsupervised learning, and reinforcement learning, in which self-supervised learning is the intermediate one between supervised learning and unsupervised learning. Artificial intelligence (AI), which is the sub-domain of machine learning, can be considered as the core of machine learning, where *artificial neural network* (*ANN*) is the preeminent approach built in AI. Fortunately, ANN supports fully the four subjects such as supervised learning, self-supervised learning, unsupervised learning, and reinforcement learning. ANN which simulates human neural network consists of one input layers, many enough hidden layers, and one output layer so that the information is entered input layer, then is propagated through hidden layers, and finally evaluated at output layer which is the result of mentioned machine learning approaches, according to a so-called *propagation rule*. For instance, given layer *k* specified by vector variable ***x****k* = (*xk*1, *xk*2,…, *xkn*)*T* is evaluated by the previous layer ***x****k*–1 by propagation rule as follows:

Or shortly,

Note, *Wk* and Θ*k* are weight parameter and bias parameter at layer *k*, where *Wk* is weight matrix and is Θ*k* bias vector as usual. The function *f*(.) is called activation function which is often squash function like sigmoid function. Training ANN is to estimate the parameters *Wk* and Θ*k* and a popular training method is the association of stochastic gradient descent (SGD) algorithm and backpropagation algorithm, based on predefined likelihood function or predefined error function. When the number of hidden layers is large enough, ANN is called deep neural network (DNN) which is the basic of deep learning. The common representation of artificial neural network (ANN) is *feedforward network* (FFN) in which the input layer is fed forwards so as to evaluate the output layer and so, ANN mentioned in this research is FNN if there is no additional explanation.Given FFN with vector layers represented by a series of layers ***x***0, ***x***1, ***x***2,…, ***x****K* is called *K*-layer FFN which is represented as follows:

Classification is the most popular method belonging to supervised learning, which is implemented perfectly by artificial neural network (ANN). As usual, training FFN classifier (ANN classifier) aims to minimize the so-called *cross-entropy loss function* loss(***x***) given variable vector ***x*** = (*x*1, *x*2,…, *xn*)*T* being the output layer ***x****K*=***x***. For instance, given the last output vector ***x****K* = (*xK*1, *xK*2,…, *xKn*)*T* computed from ANN and the real class probability ***p*** = (*p*1, *p*2,…, *pn*)*T* of such vector ***x****K*, the cross-entropy loss function loss(***x****K*) is specified as follows:

Where,

So that:

The parameters *WK* and Θ*K* at the last layer *K* is estimated by stochastic gradient descent (SGD) algorithm as follows:

Where γ (0 < γ ≤ 1) is learning rate. Note, and are gradients of loss(***x****K*) with respect to *WK* and Θ*K*, respectively, which are determined based on the cross-entropy gradient ∇loss(***x***) that is calculated as following row vector:

Where,

So that:

Note, the superscript “*T*” denotes transposition operator of vector and matrix. By association of SGD and backpropagation algorithm, all gradients and of all layers from *k*=1 to *k*=*K* are determined as follows:

|  |  |
| --- | --- |
|  | () |

So that ANN classifier are totally trained by estimating their parameters *Wk* and Θ*k* from *k*=1 to *k*=*K* as follows:

Data classification become extremely hazard when the data is image which is 2-dimension data, which raises the problem of *image classification* consisting of two issues: 1) context meaning of image is too difficult to be caught and 2) high-resolution image causes the so-called boom problem of a huge number of parameters so that training ANN consumes a lot of time and resources. Convolution neural network (CNN) is invented to solve such boom problem, in which an image is filtered via a so-called filter so as to extract the feature of such image, then such feature is fed as input of ANN for classification. The feature extraction has two goals: 1) dimension of image data space is reduced significantly by CNN filter so as to make ANN classifier to be feasible in training and 2) feature which is the essential aspect of image such as edge and pattern makes the image classification more accurate. In general, CNN is an excellent approach to image classification and so, it is necessary to sketch its methodology so that it is possible to compare matrix neural network and CNN. Essentially, convolutional neural network (CNN) has two connected parts: 1) the first part called *convolutional network* consists of a set of sequential convolutional layers which of them is filtered by a *k*x*k* filter and 2) the second part called fully connected network or dense network takes output of convolutional network as its input feature and then performs classification task on such feature with note that the output of convolutional network is image feature, essentially. Such classification task has just described but please pay attention that the main point of CNN is convolutional network (not dense network). Shortly, the input and output of convolutional network are image and image feature, respectively whereas the input and output of dense network are image feature and image class, respectively. Dense network is feedforward network (FFN) as usual whereas each convolutional layer is a 2-dimension data/feature that was filtered by a filter. Given an *m*x*n* image as *m*x*n* matrix and given *k*x*k* filter then the corresponding convolutional layer is reduced as (*m*/*k*)x(*n*/*k*) matrix after filtering, which means that image size is decreased *k*2 times with *k*x*k* filter. For instance, given an image as 2-dimension *m*x*n* matrix denoted ***X*** = (*xij*)*m*x*n*, a *filter* also called kernel is a *k*x*k* square matric denoted *W* = (*wij*)*k*x*k*, which is often 3x3 or 5x5 matrix. Given pixel ***X***[*i*][*j*] = *xij* = *xi*,*j* at the *i*th row and *j*th column of matrix ***X***, the respective value *y* which is a cell of convolutional layer ***Y***, which is called *convolutional value* or convolutional cell, is calculated by a so-called *filter operator* which is the sum of multiplications of neighbor pixels and *W* = (*wij*)*k*x*k*.

Both *W* and *θ* are parameters of convolutional layer where *W* is called filter weight matrix and *θ* is called filter bias. Activation function *f*(.) can be applied into filter operator like ANN does:

But the common activation function for CNN is Rectified Linear Unit (ReLU) which limits its input in the interval [*a*, *b*].

The effectiveness of a CNN depends on how to specify the set of sequential convolutional layers associated with their filters so that there are many scientific researches that defines very appropriate filters to extract (image) features as much as possible. However, this research focuses on specifying general filters and how to train such filters in order to generalize CNN in comparison with matrix neural network mentioned later. Therefore, it is necessary to skim over convolutional filter training (Gemini 2025). Given *K*-layer CNN whose layers are ***X***0, ***X***1, ***X***2,…, ***X****K* where each convolutional layer ***X****k* has parametric filter weight matrix and parametric filter bias , let be the *likelihood function* which takes as its input at the last layer ***X****K*, as usual, is the negative of error function:

Let *l*(***X****K*) be the entire likelihood function which is the mean of over all points (s) belonging to ***X****K*.

Where the notation |***X****K*| denotes the size of layer ***X****K*, for instance, if ***X****K* is *m*x*n* matrix, then its size |***X****K*| is *m*\**n*. The parameters and of the last layer ***X****K* are estimated firstly:

Training CNN within association of stochastic gradient descent (SGD) algorithm and backpropagation algorithm is to estimate parameters and of all convolutional layers, which is summarized from *k*=1 to *k*=*K* as follows:

|  |  |
| --- | --- |
|  | () |

Such that:

Note,

The *last convolutional error* at the output layer ***X****K* is propagated backward from dense network. Let denote the last convolutional error (matrix) including all at the last convolutional layer ***X****K*. Suppose the *N*-layer dense network has *N* layers such as ***y***0, ***y***1, ***y***2,…, ***y****N*, the last convolutional error is translated as the bias of the dense network at its first layer ***y***0, propagated from the *core last bias* as follows:

Where vec(.) represents vectorization technique which makes a matrix flatten as column vector, *g*(.) is activation function of dense network, and *Wn* is parametric weight matrix of dense layer *n*. If dense network is trained based on minimizing squared norm function, the core last bias is easily determined as follows:

Where ***y***’*N* is the real output of dense network from environment.

Although it is no doubt about the preeminence of convolutional neural network (CNN) in training image data, it is not meaningful enough to expend CNN into other high-dimension data different from image because CNN filter is appropriate to extract image feature. Moreover, data is changed or distorted after it goes through CNN so that original data with complete or full meaning is not kept intact within CNN methodology. Therefore, this research focuses on evaluating the hypothesis “whether matrix neural network is the alternative of convolutional neural network”. The next section describes matrix neural network.

**2. Matrix neural network**

The common representation of artificial neural network (ANN) is feedforward network (FFN) in which the input layer is fed forwards so as to evaluate the output layer. If FFN has many hidden layers between input layer and output layer, it will be called deep neural network (DNN). As usual, every DNN layer is coded as a vector, which raises the so-called *boom problem* of a huge number of parameters, for example, given two successive layers are coded as *m*\**n*-element vector and *p*\**q*-element vector, then there are *m*\**n*\**p*\**q* parametric weights, which is the problem of quadratic complexity, indeed. *Matrix neural network* (MNN) aims to solve the boom problem by coding layers as matrix and coding parameters as matrix too. The MNN mentioned here is FNN with matrix layers, which is represented by a series of matrix layers ***X***0, ***X***1, ***X***2,…, ***X****K* is called *K*-layer FFN as follows:

Each matrix layer ***X****k* represented by a matrix ***X****k* has three parameters such as parametric weight matrix *Uk*, parametric weight matrix *Vk*, and parametric bias matrix Θ*k*, which aims to solve the boom problem of a huge number of parameters because weigh matrices are always keep intact their size in squared number. Training MNN is to estimate parameters *Uk*, *Vk*, and Θ*k* by maximizing the likelihood function *l*(***X****K*) at the last output layer ***X****K* which is defined as the negative of squared Frobenius norm given the real output ***XK***’.

So that:

The optimization problem is solved iteratively by association of stochastic gradient descent (SGD) algorithm and backpropagation algorithm, as follows:

|  |  |
| --- | --- |
|  | () |

Such that:

Where , , and are gradients of likelihood function *l*(***X****K*) with respect to *Uk*, *Vk*, and Θ*k*, respectively. Note, the notation vec(.) denotes vectorization operator that convert a matrix into single column vector without information loss and the notation denotes Kronecker product. The superscript “*T*” denotes transposition operator of vector and matrix. Please pay attention that functions *r*(*A*) and *c*(*A*) returns the number of rows and the number of columns for matrix *A*. Therefore, is identity matrix whose dimension is *r*(*Uk*)x*r*(*Uk*) and is identity matrix whose dimension is *c*(*Vk*)x*c*(*Vk*). If *Uk* is ignored, it is identity matrix . If *Vk* is ignored, it is identity matrix .

In practical calculation, the estimation equations are not significantly simpler than traditional ANN training because of vectorization operator and Kronecker product. Fortunately, wise-multiplication is applied into reducing computational cost as follows:

Such that:

Where notation denotes wise-multiplication and notation vec–1(.) denoted the inverse vectorization operator that converts reversely a single column vector into original matrix. Note, given two vectors (matrices) whose have the same dimension, the wise-multiplication operator between them produces a new vector (matrix) whose every element is result of multiplication of one element from a vector (matrix) and one element from another vector (matrix) with constraint that the two elements have the same index in their own vectors (matrices). Please pay attention that is not the derivative of the activation function *f*(.) with respective to , indeed, which is the matrix of taking derivatives of all elements of , which means that:

Moreover, Kronecker product is easily optimized in calculation by programming technique.

The gradient of the bias Θ*k* at the layer ***X****k* is called the *k*th bias is extended as follows:

Where the quantity is called the core last bias which was calculated by the likelihood based on squared Frobenius norm.

If MNN is applied into aforementioned classification task in order to derive MNN classifier, it is simpler to modify the core last bias to minimize the cross-entropy loss function as follows:

Where,

Note, the vector ***p*** = (*p*1, *p*2,…, *pn*)*T* is the real class probability of the last layer ***X****K* = ***x****K* such that

Please pay attention that it is totally to remove complexity of Kronecker product and vectorization technique by association of programming technique and independent aspect of activation function on matrix.

**References**