IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Volume-3, Issue-5 E-ISSN: 2347-2693

Using Convolutional Neural Network to Recognize Handwritten Digits

Loc Thanh Huynh[1*], Hung Thang Phung[2] andToai QuangTon[3th]

[1,2,3]*Department of Information Technology,*
Ho Chi Minh City University of Foreign Languages – Information Technology (HUFLIT), Vietnam

www.ijcseonline.org

Received: May/05/2015 Revised: May/10/2015 Accepted: May/21/2015 Published: May/30/2015

***Abstract*—**An artificial neural network (ANN) or simply "neural net" is a data processing system consisting of a large number of simple, highly interconnected processing elements in an architecture inspired by the structure of the human brain. Hence, neural networks are often capable of doing things which humans or animals do well but which conventional computers often do poorly. This paper presents a brief introduction to convolutional neural network(CNN) – a neural network with a special structure and describes how it works to recognize handwritten digits.After a network was trained by training dataset from MNIST database, it can classify 10,000 examples from MNIST testing dataset within 35 secondsand achieve3.25% error rate.

***Keywords*—**Neural Network, Convolutional Neural Network, Feed forward, Back propagation, Classification

I. INTRODUCTION

There are problem categories that cannot be formulated as an algorithm. Recognition problems as an example. With the algorithmic approach, the computer follows a set of instructions in order to solve a problem. Unless the specific steps that the computer needs to follow are known the computer cannot solve the problem. That restricts the solving capability of conventional computers to problems that we already understand and know how to solve.In fact, computers would be so much more useful if they could do things that we don't exactly know how to do.

Neural networks take a different approach to solving than that of conventional computers. It processes information in a similar way the human brain does.The network is composed of a large number of highly interconnected processing elements (neurons) working in parallel to solve a specific problem. Neural networks learn by example. They cannot be programmed to perform a specific task. The examples must be selected carefully otherwise useful time is wasted or even worse the network might be functioning incorrectly. The disadvantage is that because the network finds out how to solve the problem by itself, its operation can be unpredictable. On the other hand, conventional computers use a cognitive approach to problem solving; the way the problem is to solved must be known and stated in small unambiguous instructions. These instructions are thenconverted to a high level language program and then into machine code that the computer can understand. These machines are totally predictable; if anything goes wrong is due to a software or hardware fault. Neural networks and conventional algorithmic computers are not in competition but complement each other. There are tasks are more suited to an algorithmic approach like arithmetic operations and tasks that are more suited to neural networks. Even more, a large number of tasks, require systems that use a combination of the two approaches (normally a conventional computer is used to supervise the neural network) in order to perform at maximum efficiency.

Corresponding Author:*Loc Huynh Thanh, thanhloc.huflit@gmail.com*
Department of Information Technology,Ho Chi Minh City University of Foreign Languages – Information Technology,Vietnam

II. METHODOLOGY

There are many neural network models can be used to solve recognition problems. In this paper, we research the convolutional neural network and backpropagation algorithm to training the network.

The back propagation algorithm consists of two paths: forward and backward path.

Forward path contains creating a feed forward network, initializing weights, simulation and training the network. Before training a feed forward network, the weights and biases must be initialized. Once the network weights and biases have been initialized, the network is ready for training.

The training process requires a set of proper inputs and targets as outputs. During training, the weights and biases of the network are iteratively adjusted to minimize the network performance function.The network weights and biases are updated in backward path.

A convolutional neural network (CNN) is comprised of one or more convolution layers (often with a subsampling step) and then followed by one or more fully connected layers as in a standard multilayer neural network. In this paper, a convolutional neural network with 5 layers is developed. An input vector and the corresponding desired output are considerfirst. Thus, we choose the MNIST database of handwritten digits to training the network ([3]). It has a training set of 60,000 examples and a test set of 10,000 examples. Pixel intensities of the original gray-scale images range from 0 (background) to 255 (maximum foreground),

28 x 28 = 784 pixels per image get mapped to real values $\frac{pixel\ intensity}{127.5} - 1.0$ in [-1.0, 1.0] and are fed into the neural network input layer.

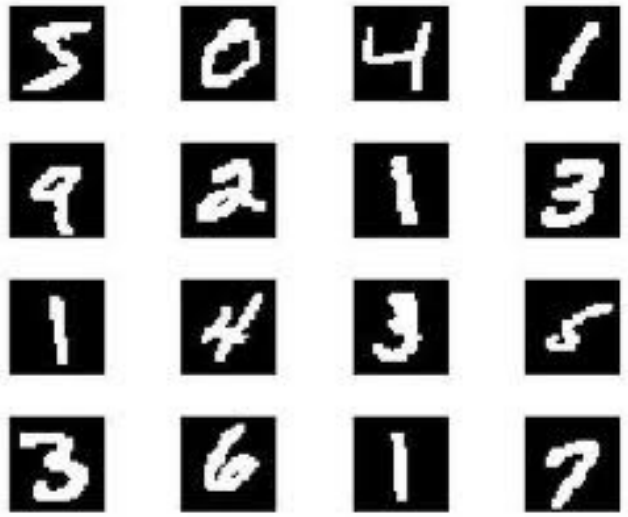

Figure 1. MNIST pixel image

The input is propagated forward through the network to compute the output vector, and the errors are determined. The errors are then propagated back through the network from output to input layer. The process is repeated until the errors being minimized.

Now, we'll look at them in detail.The overall architecture of the convolutional neural network we used for MNIST digit recognition is depicted in Figure 1.

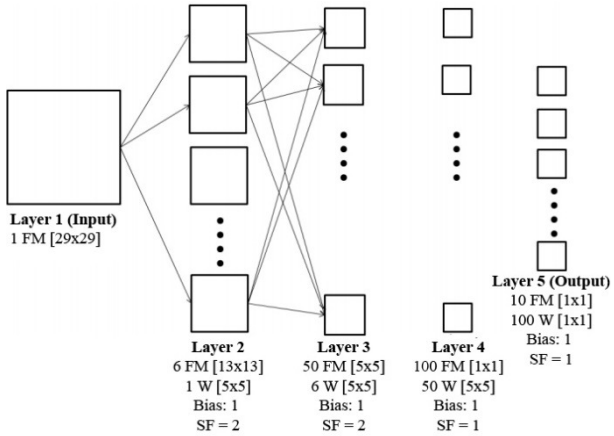

Figure 2. Convolutional architecture for handwritten recognition

Legend:
FM: feature map
W: kernel window
SF: sampling factor

The general strategy of a convolution network is to extract simple features at a higher resolution, and then convert them into more complex features at a coarser resolution. The simplest was to generate coarser resolution is to sub-sample a layer by a factor of 2. This, in turn, is a clue to the convolutional kernel's size. The size of the kernel give rise to the locally connected structure which are each convolved with the image to produce feature maps size.The width of the kernel is chosen be centered on a unit (odd size), to have sufficient overlap to not lose information (3 would be too small with only one unit overlap), but yet to not have redundant computation (7 would be too large, with 5 units or over 70% overlap). So a convolution kernel size of 5 is chosen.

As we can see from the figure 1, *Layer 1* (input layer) has one feature map which consists of $29x29 = 841$ neurons. The MNIST image database has images whose size is 28x28 pixels each, but because of the considerations described by Dr. Simard in his article "Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis", the image size is padded to $29x29$ pixels.

Layer 2 is a convolutional layer with 6 feature maps. Each feature map is sized to $13x13$ pixels (neurons). Each neuron in each feature map is a 5×5 convolutional kernel of the input layer, but every other pixel of the input layer is skipped (as described in Dr. Simard's article). As a consequence, there are 13 positions where the $5x5$ kernel will fit in each row of the input layer (which is 29 neurons wide), and 13 positions where the $5x5$ kernel will fit in each column of the input layer (which is 29 neurons high). There are therefore $13x13x6 = 1014$ neurons in Layer 1, and $(5x5 + 1)x6 = 156$ weights (the "+1" is for the bias).

On the other hand, since each of the 1014 has 26 connections, there are $1014x26 = 26364$ connections from this layer to the prior layer. At this point, one of the benefits of a convolution "shared weight" neural network should become more clear: because the weights are shared, even though there are 26364 connections, only 156 weights are needed to control those connections. As a consequence, only weights need training. In comparison, a traditional "fully connected" neural network would have needed a unique weight for each connection, and would therefore have required training for 26364 different weights. None of that excess training is needed here.

Layer 3 is also a convolutional layer, but with 50 feature maps. Each feature map is $5x5$, and each unit in the feature maps is a 5x5 convolutional kernel of corresponding areas of all 6 of the feature maps of the previous layers, each of which is a $13x13$ feature map. There are therefore $5x5x50 = 1250$ neurons in Layer 2, $(6x5x5 + 1)x50 = 7550$ weights, and $1250x26 = 32500$ connections.

Turning to *Layer 4*, *Layer 4*is a fully-connected layer with 100 units. Since it is fully-connected, each of the 100 neurons in the layer is connected to all 1250 neurons in the previous layer. There are therefore 100 neurons in Layer 4, $(50x5x5 + 1)x100 = 125100$weights, and $100x1251 = 125100$ connections.

Layer 5 is the final, output layer.This layer is a fully-connected layer with 10 units, corresponding the expected result (represent the number from 0 to 9). Since it is fully-connected, each of the 10 neurons in the layer is connected to all 100 neurons in the previous layer. There are therefore 10 neurons in Layer 5, $(100x1x1 + 1)x10 = 1010$ weights, and $10x101 = 1010$ connections.

Altogether, adding the above numbers, there are a total of 3215 neurons in the neural network, 133816 weights, and 184974 connections.

Layer	Neuron (pixel)	Weight	Connection
1	841	-	-
2	1014	156	26364
3	1250	7550	32500
4	100	125100	125100
5	10	1010	1010
Sum	3215	133816	184974

Table 1. The number of neurons, weights and connections in CNN

Feed-Forward and activation function

Before the training all weights and biases must be initialized. That are initialized with a uniform random distribution in [-0.05, 0.05].Neurons in layers is computed by sum of a convolution of its kernels and the previous layer, to which a bias is added. This weighted sum, denoteda_i for neuron i, is then passed through a activation function to produce the state of neuron i, denoted by x_i:

$$x_i = f(a_i)$$

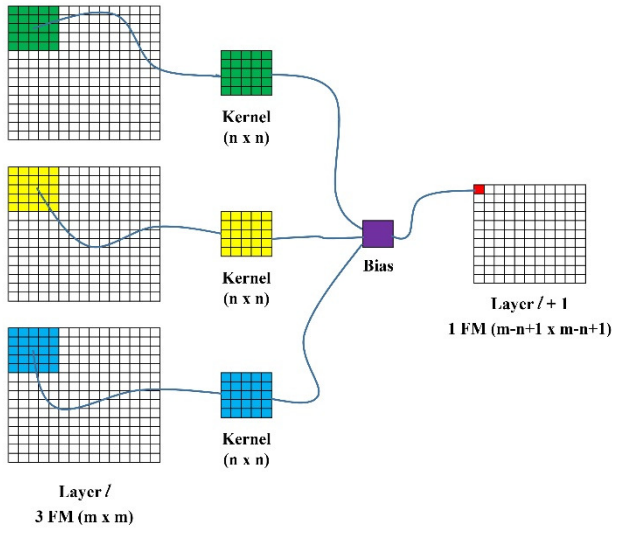

Figure 3. Feed-forward process

Selection of a good activation function is an important part of the design of neural network. The activation function should be symmetric and the neural network should be trained to a value that is lower than the limits of the function. One function which is recommended by many articles on the web is the classical sigmoid function (or "logistic" function), defined as $F(x) = \frac{1}{1+e^{-x}}$, should not be used since it is not symmetric: its value approaches +1 for increasing x, but for decreasing x its value approaches zero (it does not approach -1 which it should for symmetry).

One good selection for the activation function is the hyperbolic tangent, or $F(x) = \tanh(x)$. This function is a good choice because it's completely symmetric, as show in the following graph. If used, then do not train the neural network to ± 1. Instead, choose an intermedia value, like ± 0.8.

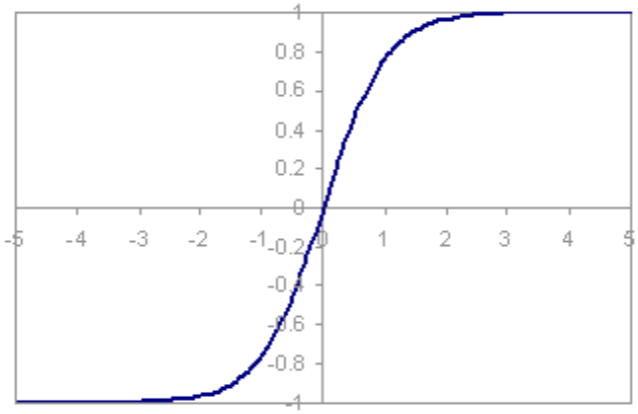

Figure 4. Tanh function graph

Another reason why hyperbolic tangent is a good choice is that it's easy to obtain its derivative. Not only it is easy to obtain its derivative, but also the value of derivative can be expressed in terms of the output value. More specifically, given that:

$$a = \partial(x) = tanh(x) = \frac{sinh(x)}{cosh(x)} \quad (1)$$

where $x$is the input to the function (corresponding to the activation value of a neuron) and $\partial(x)$ is the output of the neuron.

Then

$$\frac{d\partial}{dx} = \frac{d}{dx}\left(\frac{sinh(x)}{cosh(x)}\right) = \frac{cosh^2(x) - sinh^2(x)}{cosh^2(x)} = 1 - tanh^2(x) \quad (2)$$

Since $\partial(x) = tanh(x)$, so the result is $\frac{d\partial}{dx} = 1 - \partial^2(x)$(3)

This result means that we can calculate the value of the derivative of $\partial(x)$ given only the output of the function, without any knowledge of the input.

To train the network to values of ± 1, we multiply x with $\frac{2}{3}$ and this result with 1.7159 ([4]). Thus, the activation used in this paper is a scaled version of the hyperbolic tangent:

$$\partial(x) = 1.7159tanh\left(\frac{2}{3}x\right) \quad (4)$$

and

$$\frac{\partial(x)}{dx} = \frac{2}{3x1.7159}(1.7159 - x)(1.7159 + x) \quad (5)$$

Backpropagation

Backpropagation gives us a way to determine the error in the output of a prior layer given the output of the current layer. The process is therefore iterative: start at the last layer

cad calculate the change in the weights for the last layer. Then calculate the error in the output of the prior layer. Repeat.

The backpropagation equations are given below. Start the process off by computing the partial derivative of the error due to a single input image pattern with respect to the outputs of the neurons on the last layer. The error due to a single pattern is calculated as follows:

$$C_P^L = \frac{1}{2}\sum\left(a_i^L - y_i^L\right)^2 \quad (6)$$

where:

C_x^L is the error due to a single pattern P at the last layer L;

y_i^L is the target output at the last layer (the desired output at the last layer);

a_i^L is the actual of the value output at the last layer.

Given (6), then taking the partial derivative yields:

$$\frac{\partial C_P^L}{\partial a_i^L} = a_i^L - y_i^L \quad (7)$$

Equation (7) gives us a starting value for the backpropagation process. We use the numeric value for the quantities on the right side of the (7) in order to calculate numeric values for the derivative. Using the numeric values of the derivative, we calculate the numeric values for the changes in the weights, by applying the following two equation (8) and (9):

$$\frac{\partial C_P^L}{\partial x_i^L} = \frac{\partial C_P^L}{\partial a_i^L}\partial'(a_i^L) = (a_i^L - y_i^L)\partial'(a_i^L) \quad (8)$$

where $\partial'(a_i^L)$ is the derivative of the activation function.

$$\frac{\partial C_P^L}{\partial w_{ij}^L} = a_j^{L-1}.\frac{\partial C_P^L}{\partial x_i^L} \quad (9)$$

Then, using (7) again and also (8), we calculate the error for the previous layer, using the following equation (10):

$$\frac{\partial C_P^{L-1}}{\partial a_k^{L-1}} = \sum_i w_{ik}^L . \frac{\partial C_P^L}{\partial x_i^L} \quad (10)$$

The values we obtain from (10) are used as starting values for the calculations on the immediately preceding layer. In other words, we take the numeric values we obtained from (10), and use them in a repetition of (8), (9) and (10) for the immediately preceding layer.

Meanwhile, the values from (9) tell us how much to change the weights in the current layer L. In particular, we update the value of each weight according to the formula:

$$\left(w_{ij}^n\right)_{new} = \left(w_{ij}^n\right)_{old} - eta.\frac{\partial C_P^L}{\partial w_{ij}^L} \quad (11)$$

Where eta is the "learning rate", typically a small number like 0.00005 that is gradually decrease during training.

III. EXPERIMENTAL RESULTS

Most remarkable, the best network has an error rate of only 0.35% (35 out of 10,000 digits). In this network, we used the first 50,000 patterns of the MNIST training set for training, and the remaining 10,000 for validation and parameter adjustment. The results are reported in the table below:

Classifier	Preprocessing	Error rate	Preferences
2-layer NN, 300 hidden units, mean square error	Không	4.7%	LeCun et al. 1998
2-layer NN, 1000 hidden units	Không	4.5%	LeCun et al. 1998
6-layer NN 784-2500-2000-1500-1000-500-10 (trên GPU) [elastic distortions]	Không	0.35%	Ciresan et al. Neural Computation 10, 2010 and arXiv 1003.0358, 2010

Table 2. Comparison between various algorithms

ACKNOWLEDGMENT *(HEADING 5)*

We would like to thank Toai Ton MSc who is the vice dean of Information Technology at Ho Chi Minh City University of Foreign Languages – Information Technology (HUFLIT), Vietnam for his guidance and support when we initially looked for references and sources of information on this research project. We look forward to his guidance in the years to come.

REFERENCES

[1] Michael Nielsen, "Neural networks and deep learning", Sep. 2014, http://neuralnetworkanddeeplearning.com.

[2] Patrice Y. Simard, Dave Steinkraus, John C. Platt, "Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis", Microsoft Research, 2003

[3] MNIST Database of Handwritten digits, MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges, 28 Jan, 2015

[4] Y. LeCun, "Generalization and Network Design Stategies", Technical Report, 1989.

[5] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, "Gradient-Based Learning Applied to Document Recognition", Proceedings of the IEEE, Vol-86, Issue-11, Page no (**2278-2324), 1998**