A neural network solver for differential equations

Qianyi WANG*, Tomoo AOYAMA**

The faculty of engineering, Miyazaki University
Gakuen Kibanadai-nishi 1-1, Miyazaki city, 889-2192, Japan.
* tgb247u@student.miyazaki-u.ac.jp
** t0b217u@cc.miyazaki-u.ac.jp

Abstract

We propose a solver for differential equations, which uses only a neural network. The network is built of multi-layer-structure and can be learned. The learning method is defined as an equation that resembles the BP. The techniques are based on the analogue type of neural network, its derivative expression, and iterations similar to the BP algorithm. Precision of the solution depends on learning-error of the analogue type of neural network. The structure of the solver be equivalent to the multi-layer neural network; therefore, a parallel processing can be done.

1. Introduction

The differential equations are the most important means that describe the natural phenomena. The analytical derivative expressions of the multi-layer neural networks are already published [1,2,3]. Here, there is a possibility; that is a connecter between neural networks and differential equations. We have considered a means, the differential equation solver, which is executed on the neural networks. If the method were defined, the differential equation can be solved by actions in the sea of neurons, which proves that collective neurons have another kind of ability in addition to the pattern recognition. The neural networks are not equal to the brain; however, they are a part of the structure. Therefore, we are interested in the connection means.

2. Derivation of learning equation

2.1 Analogue type of neural network and its derivative

The BP-learning equation gives a converter that transforms a vector to another hose element number is different. When the learning is finished, the converter and the derivatives are both defined. The derivatives describe relations between inputs and outputs of the neural network. The BP-learning is a function generator, where values of the object function are given. Then, if derivatives of the object function are given, we may derive another type of generator. For the generator, the input data are voluntary, then, we can select the data as arithmetical progression; however, it would be required that they should be similar to samplings for the object function.

On original BP-learning equation, the energy “E” is defined as,

$$E=\sum_j (O_j-T_j)^2.$$  (eq1)

where $O_j$ and $T_j$ are output of j-th neuron on output-layer and expectation value for the j-th neuron respectively. $\sum_j$ is to accumulate over all indexes “j”.
The (eq1) is defined at the learning points. The differential is,
\[ \frac{dE}{dW} = 0, \]  
(eq2)
where \( W \) is a connection-weight between neurons on different layers. BP-learning equation is derived from the (eq2).

Here, we describe a notation for the analogue type of neural network (ANN)[4], which is a function transformer.

For the processing between first and second layers:
\[ Y_k = \sum_i V_{ik} X_i, \quad Y'_k = \frac{1}{1+\exp(-Y_k)} , \]  
(eq3)
Where \( V \) is a connection-weight matrix between the first-layer and the second-one, and \( \{X_i\} \) is input data.

For the processing between the second and third layers:
\[ O = \sum_k W_k Y'_k, \]  
(eq4)
Where \( W \) is a connection-weight matrix between the second-layer and the third-one, and \( O \) is the output of the neural network. The \( W \) is redefined here. The numbers of neurons from the first to third layers are any, any, and 1, respectively.

When you consider the connection of (eq3) and (eq4), you will find that ANN is the sigmoid-function expansion. In the notation, we defined the derivative as followings,
\[ \frac{dO}{dX} = \sum_k W_k V_{ik} \frac{1}{1+\exp(-Y_k)}. \]  
(eq5)

### 2.2 Modification of the energy term

Here, we assume the differential equation for one variable; therefore, the number of neurons on the first layer is two, where the bias neuron is included. So, (eq5) is reduced as,
\[ \frac{dO}{dX} = \sum_k W_k V_{ik} \frac{1}{1+\exp(-Y_k)}. \]  
(eq6)
Where index “\( i \)” does not include the bias path, and the number of the path is one.

The index “\( j \)” means neurons on the third layer. However, on the case, as the neuron number is 1. Here, we rewrite the both index “\( i \)” to “iptn” that is for the individual input data. So, we get the followings,
\[ \frac{dO}{dX} \text{iptn} = [\sum_k W_k V_{ik} \frac{1}{1+\exp(-Y_k)}] \text{iptn} = \text{Diptn}, \]  
(eq7)
The (eq1) is rewritten as for the index “iptn”,
\[ E = (\text{Diptn} - \text{Uiptn})^2. \]  
(eq8)
\( \text{Uiptn} \) is a sampling datum for first-order-derivative of the object function. The (eq8) is equivalent to the next expression,
\[ \frac{df}{dt} = f(t), \quad i.e., \quad (\frac{df}{dt} - f(t))^2 = 0. \]  
(eq9)
Here, we get a method to solve the differential equations.

### 3. Numerical calculations

#### 3.1 Definition of neural network

The (eq8) is not (eq1); but, in 0th order approximation, we can use the same calculation schemes derived by the original BP-algorithm. The convergence probability will be less; it is natural. We used network parameters on table 1.

**Table 1. Network parameters of ANN**

<table>
<thead>
<tr>
<th>parameters</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neuron# on 1st.</td>
<td>2</td>
</tr>
<tr>
<td>Neuron# on 2nd.</td>
<td>12</td>
</tr>
<tr>
<td>Neuron# on 3rd.</td>
<td>1</td>
</tr>
<tr>
<td>Epsn. of 2nd.</td>
<td>0.1-0.2</td>
</tr>
<tr>
<td>Epsn. of 3rd.</td>
<td>0.05-0.1</td>
</tr>
</tbody>
</table>

#### 3.2 Precision of differential of object function

To examine the (eq6/7), as for input/teaching data are sampled at 21 points of \( f(x) = x, f(x) = x^2 \) respectively; we calculated the derivatives. The number of BP-learning iterations was 50K, and the E-term was converged under 0.0039. When coded by java, the CPU time was within 10 seconds on 300MHz-Celeron. The results are listed in figure 1.
As it is clear in the figure 1, the calculations are reasonable.

**Figure 1. Error-curves of the differential and output of calculated ANN:**
The output values are multiplied by 10. The errors are differences between the true values and calculated ones.

### 3.3 Solution of (dx/dt)=x^n(t), n={-1,1,2}

Next, we calculated solutions for differential equations: (dx/dt)=x^n(t), n={-1,1,2}. Input data were all arithmetic progression, and the sampling number for the teaching data was 21. The numbers of iterations were 3001, 116, 6354, for n={-1,1,2}. These numbers were fixed when decreasing of E-term in (eq8) was stopped and they were increasing. At the time, the E-terms were under 0.0017, 0.017, and 0.0045 respectively. The calculated results are listed in figure 2. We believe that the calculated error curves are expected; however, in case of n=1, the iteration number is rather small.

**Figure 2. The error curves for the differential equations: (dx/dt)=x^n(t), n={-1,1,2}.**
The description numbers {1,2,3} correspond with that of n={-1,1,2}.

### 3.4 Solution of (dx/dt)=sin {x(t)}

We calculated solutions for differential equations: (dx/dt)=sin{x(t)}, 0<t<π. The term (dx/dt) is increasing at first, and gradually it is decreasing; therefore, the network integrations would be difficult. This is a reason for selecting the test.

Input data were all arithmetic progression, and the sampling number for the teaching data was 21. The number of iterations was 9500 that was fixed when decreasing of E-term in (eq8) was stopped and it was increasing. At the time, the E-term was under 0.069. The calculated result is listed in figure 3. We believe that the calculated error curve is reasonable, and the solution (curve 2) is acceptable.

**Figure 3. The true and solution values, and the differences for the differential equations: (dx/dt)=sin x(t), 0<x<π.** The description numbers {1,2,3} are the true, solution, and differences, respectively. The amplitude of curve 3 is multiplied by 10.

### 4. Conclusion

We proposed a solver for differential equations, which used only a multi-layer neural network. The learning equation is (eq8) that is almost equivalent to the BP-learning equation, which is derived in case of analogue type of neural networks. We used the solver, and got some results for elementary differential equations, which are acceptable.

The solver is only constructed of the neural networks; therefore, the solver can be processed in parallel. It does not accumulate numerical errors like as the direct solvers; therefore, we believe that the solver is a useful tool. However, since there is an
improvement in (eq8/9), we are deriving new equations.

5. Reference