Deep Learning in Solving Mathematical Equations

Bassma A. Awad Abdrazg  Hanan G.A. Atetalla  Sumaia Masoud Abdalla

Department of Mathematics, Faculty of Science, Omar AL-Mokhtar University, Albaïda – Libya

bassma.abraheem@omu.edu.ly
Deep Learning in Solving Mathematical Equations

Abstract
Solving a mathematical equation can be a tough task. There is a need for employing deep learning in this field. Hence, it can be easier to solve mathematical equations. Deep networks have been applied into various fields and showed a great performance in terms of accurate generalization. In this paper, a deep network, named stacked auto-encoder (SAE) with two hidden layers is trained to learn solving mathematical equations. The network is trained and tested using 200 different equations, where 100 are used for training and 100 for testing. Experimentally, the network showed good generalization accuracy in predicting the answers of solving the mathematical equations which were not used during the training of the network.

Keywords: Mathematical equations, Deep learning, Neural networks, Generalization, Stacked auto-encoder.
1. Introduction

Deep learning is considered as a new method of machine learning which is closer to artificial intelligence than other previous and conventional neural networks models [1]. In contrast to the traditional neural networks which are linear models, deep networks are hierarchical representations of data to extract complexity and abstractions of data. In other words, deep learning networks are neural networks of many hidden layers, that is why they are called deep, which allows them to extract different levels features from input data. The training and working principles vary from one deep network to another [2]. For example, stacked auto-encoder which is a deep network has its own learning principles in which it uses Greedy-layer wise training, however, a convolutional neural network has a different architecture and training algorithm than stacked auto-encoder.

Recently, deep learning has encountered progress in the field of mathematics. Neural networks have been applied first in solving many differential equations by Dissanayake and Phan-Thien [3]. Furthermore, the concept of generality in a network layer was elaborated upon by Yosinski et al. [4] as its capacity to exhibit effective performance across a spectrum of tasks. This notion stemmed from the observation that initial layers within image-oriented convolutional neural networks (CNNs) converged towards comparable features, spanning multiple network architectures and applications. We assert that these concepts share a correlation: when a particular representation fosters exemplary performance across diverse tasks, networks that adapt to any of these tasks are likely to uncover analogous representations.

To illustrate, Li et al. [5] introduced a methodology for gauging layer similarity through the identification of neuron alterations that enhance the interrelation between networks. In contrast, while preserving a logarithmic efficiency, the Singular Value Canonical Correlation Analysis (SVCCA) stands out as a more computationally streamlined alternative to change-based techniques. Additionally, Berg and Nyström [6] concurrently endeavored to unravel the composition of Deep Evolutionary Neural Networks (DENNs) by scrutinizing weights and biases directly. However, their findings were impeded by the sensitivity of the outcomes to local minima, into which their networks converged.

Hence, in this work we employ stacked auto-encoder which is a deep network consisting of two hidden layers which are learned by this network, trained on solving mathematical equation tasks.
2. Neural Networks

The artificial neuron model constitutes a mathematical representation of a biological neuron, characterized by multiple inputs ($x_1, x_j$) and a solitary output ($y$). This conception aligns with the foundational framework presented by McCulloch and Pitts, often denoted as a rudimentary neuronal paradigm. This construct assimilates input patterns and configures them as input parameters through the corresponding weights. To elucidate, the linear threshold system, an archetype of this neuron, adeptly processes input from numerous units, culminating in tangible values, contingent on the activation function. This procedural progression adheres to the transfer function, orchestrating the translation from input (actual values) to output (within a specified interval); this conversion can manifest as either linear or nonlinear in nature. McCulloch and Pitts' model employs the sigmoidal function, recognized as the hard limiter, as their chosen transfer function, denoted by ($\Theta$).

Within the artificial neuron framework, the synaptic connections are termed as weights ($w$), signifying the conduits that interlink inputs and the neuron. In the context of McCulloch and Pitts' model, the weight ($w$) and threshold ($\theta$) values remain invariant. Notably, the artificial neuron model adeptly segregates input sets into dichotomous classes, thereby generating a binary output (Figure 1). The ultimate output ($y$) in both the artificial neuron and McCulloch and Pitts' model is explicated through the summation of the dot product, encapsulating weights and input parameters ($w_i, x_i$), while being governed by the activation function $\Theta$ [7,8].

$$N = \sum_{i=1}^{J} w_i x_i - \theta = w^T x - \theta$$  \hspace{1cm} (1)

$$y = \Theta(N)$$ \hspace{1cm} (2)

**Figure 1:** Architecture of artificial-neuron model [8]
In the context of this study, the symbol $N$ is indicative of an artificial neuron network model, while the variable $x$ encapsulates the input parameters under consideration. The weight or connection lines between inputs and the transfer function are denoted by $w$, signifying their role in transmitting information within the network. The activation function, governed by the symbol $\phi$, is characterized as sigmoidal in nature. Moreover, $\theta$ represents the threshold value, a parameter employed to displace the decision boundary, thereby enabling the network to discern patterns that extend beyond the origin.

In the year 1957, the inception of the initial perceptron, a construct characterized by a singular layer, was credited to Rosenblatt. This pioneering development drew inspiration from the McCulloch & Pitts model and the tenets of Hebbian learning. Notably, Rosenblatt's Perceptron model transcends the constraints of its precursor, the artificial neuron (McCulloch & Pitts) model, in that it demonstrates the capacity to classify input sets into more than two categories. This stands in contrast to the latter, which exclusively accommodates the bifurcation of input sets (Figure 2).

Within the realm of the single-layer perceptron model, the selection of diverse activation functions (denoted as $\phi$) has been explored, encompassing variants such as the bipolar function. The determination of weights ($w$) and thresholds or biases ($\theta$) is achieved either through analytical computation or by means of a learning algorithm. Notably, the output ($'y'$) of the single-layer perceptron is mathematically articulated as follows [7,8,9,10].

$$N = w^T x - \theta$$

$$'y = \phi (N)$$

Figure 2: Architecture of Rosenblatt’s Perceptron [7]
The single-layer perceptron exhibits competence solely in resolving issues that are amenable to linear separation. The modulation of inter-neuron weights is facilitated by the employment of a learning algorithm, a principle underscored by Rosenblatt's perceptron convergence theorem. This mechanism operates by iteratively adapting the weight values, an iterative process often guided by the error equation \( E_{t,j} \). Furthermore, the operational framework of the perceptron's learning algorithm can be articulated as presented below:

\[
N_{t,j} = \sum_{i=1}^{j} x_{t,i} w_{ij} (t) - \theta_j = w_j^T x_t - \theta_j
\]  

\[
'y_{t,j} = \begin{cases} 
1 & N_{t,j} > 0 \\
0 & \text{otherwise} 
\end{cases}
\]

\[
E_{t,j} = y_{t,j} - 'y_{t,j}
\]

\[
w_{ij} (t + 1) = w_{ij}(t) - n x_{t,i} E_{t,j}
\]

Let \( N \) represent a network comprising single-layer perceptrons. In this context, \( x_{t,i} \) pertains to the \( i \)th input of the \( t \)th example. The weight connecting neurons, specifically \( w_{ij} \), pertains to the weight associated with the \( i \)th node at the \( t \)th instance.

Furthermore, the symbol \( \theta \) denotes the bias or threshold applied to the neuron. Concomitantly, \( \phi \) is indicative of the transfer or activation function characterizing the neuron's behavior.

The term \( E_{t,j} \) is utilized to represent the error encountered within the system. In a similar vein, \( y_{t,i} \) signifies the actual output, aspired for and contingent upon the network's performance. Conversely, \( 'y_{t,i} \) denotes the effective or actual output, an outcome derived from the network's predictive capabilities, as delineated by Du and Swamy in 2013.

3. **Proposed Stacked Auto-Encoder for solving mathematical equations.**

This paper introduces a novel approach employing deep learning techniques to address mathematical equation solving. The method leverages a stacked auto-encoder (SAE) as its core mechanism for this purpose. The choice of utilizing an SAE was influenced by extensive research within the mathematical domain, wherein existing networks predominantly comprised backpropagation neural networks or convolutional neural networks. Consequently,
the adoption of a stacked auto-encoder was prompted by the need to explore and assess its efficacy in handling mathematical equations, a task characterized by its inherent complexity. Given the intricate nature of the prediction task, a preprocessing step involving data normalization was incorporated prior to inputting data into the network. This normalization step aims to discern pivotal and unique data features, thereby facilitating the learning phase of the stacked auto-encoder. The anticipated outcome of this preprocessing is enhanced prediction performance and heightened accuracy.

The architecture of the proposed network is delineated in Figure 1, encapsulating the concepts.

The core objective of system identification pertains to establishing a model relationship, determining system orders, and approximating an elusive function through a neural network model. This procedure relies on a dataset comprising input and output data pairs. The neural identifiers utilized in this context are Multi-Layer Feed-Forward artificial neural networks (MLFF), characterized by an input layer, one or more hidden layers with biases, and a linear or non-linear output layer. The proposed Black-Box (BB) system specifically employs an MLFF with a single hidden layer. This configuration can be further elaborated as follows:

a) Input Layer:
Comprising (n+1) neurons, this layer encapsulates the equation coefficients within a linear algebraic system. Each neuron corresponds to the coefficients of an equation, with an additional neuron dedicated to representing the equation results.

b) Hidden Layer:
Encompassing 48 neurons, this layer serves as a conduit for individual bits from the Expansion Permutation process.

c) Output Layer:
Featuring n neurons, this layer corresponds to the desired variable values within the equations under consideration.

The architectural depiction of the Black-Box Neural Network (BBNN) system pertaining to three equations with three variables is depicted in Figure 3, outlining the structural components and their interconnections.
Figure 3: Topology of the proposed stacked auto-encoder for solving mathematical equations.

3.1 SAE Training

The discussion of the training of the proposed SAE and its performance in this stage is presented in this section. It is important to mention that the data were all normalized to range between 0 and 1 before being fed into the network, which may help in arranging all data to range of 0 to 1. Note that SAE is trained on 100 samples out of 200, which include different examples of equations.

Since the stacked auto-encoder is a deep network, it means it needs pre-training. Pre-training is achieved by training the stacked auto-encoder layer by layer, which is called greedy layer-wise training. During pre-training, outputs are not labeled as in this phase; the network is being trained to learn how to extract important features from the input data. This may help the network to perform accurately in the fine-tuning where it is trained to predict different answers of mathematical equations.

When the network finishes pre-training, it becomes ready to be fine-tuned. Fine-tuning is done using the conventional backpropagation algorithm that traditional neural networks use.
Table 1: Dataset and data division

<table>
<thead>
<tr>
<th>Image sets</th>
<th>Number of data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training set</td>
<td>100</td>
</tr>
<tr>
<td>Testing set</td>
<td>100</td>
</tr>
<tr>
<td>Total number of data</td>
<td>200</td>
</tr>
</tbody>
</table>

Table 1 presents the data samples number in the dataset and its data division for training and testing, respectively.

The training parameters of the network during pre-training and fine-tuning are shown in Table 2.

Table 2: Learning parameters of networks during pre-training and fine-tuning

<table>
<thead>
<tr>
<th>Learning parameters</th>
<th>Values (Pre-training)</th>
<th>Values (Fine-training)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of training images</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Number of output neurons</td>
<td>100</td>
<td>3</td>
</tr>
<tr>
<td>Number of layers of the network</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Number of hidden layers</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Learning rate</td>
<td>0.47</td>
<td>0.18</td>
</tr>
<tr>
<td>Maximum number of iterations</td>
<td>500/500</td>
<td>323/400</td>
</tr>
<tr>
<td>Transfer function</td>
<td>Sigmoid</td>
<td>Sigmoid</td>
</tr>
</tbody>
</table>

Figure 4 illustrates the learning trajectory of the network throughout its pre-training phase. The discernible trend showcases a reduction in the network error as the iterations progress. However, it is noteworthy that the error fails to converge to an exceedingly diminutive value (0.04), indicative of a residual level of imprecision in the network's predictive outcomes.
The learning of the network can usually be seen by looking at its learning curve. Learning curve is a graph that shows the variation of error versus the increase of iterations required for the network to finish training. As can be seen from Figure 4, the developed network seems to learn well as the error is decreasing sharply with increase of number of iterations. The network was capable of reaching an error of 0.040218 with maximum iterations of 500.

**Figure 4: Learning curve of SAE during pre-training**

**Figure 4: Learning curve of SAE1 during fine-tuning**
Table 3: Training network performance

<table>
<thead>
<tr>
<th>Learning results</th>
<th>Pre-training</th>
<th>Fine-tuning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of training images</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Training recognition rate</td>
<td>87%</td>
<td>100%</td>
</tr>
<tr>
<td>Minimum square error achieved (MSE)</td>
<td>0.0402</td>
<td>0.0009</td>
</tr>
<tr>
<td>Iterations required</td>
<td>500</td>
<td>323</td>
</tr>
<tr>
<td>Training time</td>
<td>250 secs</td>
<td>38 secs</td>
</tr>
</tbody>
</table>

The performance of the designed SAE for solving mathematical equations can be summarized in Table 3. As can be seen, the network was trained on 470 samples during pre-training and fine-tuning. However, its performance was not the same. It is seen that the accuracy of network in predicting different mathematical equations was 100% during fine-tuning which is than that obtained during pre-training. Note that this accuracy was achieved in a shorter time (38 secs) than that of pre-training (250 secs). Moreover, it is important to mention that the network required a smaller number of iterations during fine-tuning to achieve an error of 0.0009, which is smaller than that reached during pre-training (0.0402).

This can be considered as a major problem as the performance of the network is measured by its performance during fine-tuning as it is learning to predict data. However, during pre-training the network is just trying to warm up and get some power in extracting and identifying features in data.

3.2 Deep Models Testing

Once trained, the stacked auto-encoder network (SAE) is tested using 100 other samples. Note that these samples are other than those used for training the network. In addition, these data include samples of all 3 classes. Table 8 shows the performance of the network during testing, in terms of accuracy.

Table 4 shows that the trained and tested SAE was capable of achieving a good generalization ability when tested on data that are unseen before, where it reached an accuracy of 91.2%.
4. Results Discussion

A deep stacked auto-encoder based deep learning is developed in this study from the purpose of solving mathematical equations. The designed SAE is first built using two auto-encoders and then it is pre-trained in order to learn the extraction of useful features from input data which helps then in predicting the answers of different mathematical equations more accurately. These kinds of networks are trained using Greedy-layer wise training. This is an algorithm that allows the training of an auto-encoder layer by layer, in an unsupervised manner. The importance of this training technique (pre-training) is that it provides the network with good features extracting abilities and learned weights that are used in fine-tuning the network.

In this study, the data is normalized before being fed into network and then the pre-training starts. Then, the same data are fed into a network which is also fine-tuned to predict answers of mathematical equations of different 3 classes. Table 5 shows the deep networks’ performances.

<table>
<thead>
<tr>
<th>Performance parameters</th>
<th>SAE1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of training images</td>
<td>100</td>
</tr>
<tr>
<td>Number of testing images</td>
<td>100</td>
</tr>
<tr>
<td>Training recognition rates</td>
<td>100%</td>
</tr>
<tr>
<td>Testing recognition rates</td>
<td>91.2%</td>
</tr>
<tr>
<td>Minimum square error achieved (MSE)</td>
<td>0.0009</td>
</tr>
<tr>
<td>Iterations required</td>
<td>323</td>
</tr>
<tr>
<td>Training time</td>
<td>288 secs</td>
</tr>
</tbody>
</table>
Table 5 shows the evaluation of the network’s performance during both training and testing. It is seen that the network was able to achieve a 100% training accuracy; however, it required a relatively long training time of 288 secs and 323 iterations to achieve such accuracy. Note that this network also achieved a very small error of 0.0009. Moreover, the network was tested on 152 different samples, and it achieved an accuracy of 91.2% which can be considered as promising results from such mathematical application.

5. Conclusion

The current investigation centers around the utilization of a deep network, specifically a stacked auto-encoder, as a tool for the resolution of mathematical equations. The application of a deep network, especially a stacked auto-encoder, within this domain is inherently intricate, given the task's demand for precise equation solution based on their intrinsic data and attributes. This prerequisite for high accuracy is driven by the field's intolerance for even minor error margins. Thus, the deployment of a sophisticated predictive model like the deep network is instrumental in expediting and facilitating the work of mathematicians. The chosen approach employs a stacked auto-encoder model, which is subjected to training and testing using datasets comprising 100 samples each. The model's initial training involves the utilization of normalized data derived directly from the equations. The primary aim of data normalization is the enhancement and extraction of pertinent, distinctive attributes, enabling the model to discern unique characteristics within each sample. This feature extraction process is crucial, contributing to the model's proficiency in accurately categorizing the data. Consequently, this enhancement in classification accuracy serves to expedite the network's learning process and bolster its convergence. Subsequently, the model is assessed through testing on a distinct set of 100 samples, with its performance evaluated in terms of accuracy, training time, and the extent of error incurred.

In summation, the assessment of the stacked auto-encoder's performance demonstrates its adeptness at extracting pertinent features and delivering accurate predictions for previously unseen equations. This underscores the model's potential as an effective predictor for mathematical equations, given a preliminary preprocessing stage aimed at refining classification, all while adhering to a stringent error margin.
References


