Easy implementation of Classification of MNIST Handwritten using keras

Kundan Sharma
AItechsystems
http://www.ai-techsystems.com

Kundansharma6161@gmsil.com

Abstract. This paper presents an implementation using Back-propagation Neural Network to achieve the classification of the MNIST handwritten digit database. Here, we use plot of loss during training and classification accuracy to determine the performance of the neural network. The experimental results show that to a certain extent, the back-propagation neural network can be used to solve classification problem in the real world. Besides, we try to achieve image compression with Autoencoder and determine its effects. It does help to reduce the network size and thus increase the speed of the neural network, improving the performance. However, the accuracy rate cannot be guaranteed and potential reasons are discussed. Then we try to modify the structure of the original neural network into a Convolutional Neural Network (CNN). The results indicate that CNN can be used to improve the performance during solving image recognition problem. Besides, we combine CNN with Autoencoder into a Conv_Autoencoder structure and some tests are conducted. At last, the results generated by our network system are compared with those in another research paper tested the same MNIST handwritten digit database. It seems that the results of our experiment are a little bit worse, which means further work is necessary.

Keywords: MNIST handwritten digit database; Neural Network; Back-propagation; Classification; Autoencoder; Convolutional Neural Network.

1 Introduction

Handwritten digit recognition is an important problem in optical character recognition, and it can be used as a test case for theories of pattern recognition and machine learning algorithms. To promote research of machine learning and pattern recognition, several standard databases have emerged. The handwritten digits are preprocessed, including segmentation and normalization, so that researchers can compare recognition results of their techniques on a common basis as well as reduce the workload (Deng, 2012).

This paper uses MNIST handwritten digit database on Artificial Neural Network (ANN). MNIST handwritten digit database can be taken from the page of Yann LeCun (Yann.lecun.com, n.d.). It has become a standard for fast-testing theories of pattern recognition and machine learning algorithms. The MNIST database was constructed out of the original NIST database; hence, modified NIST or MNIST. It contains 60,000 handwritten digit images for the classifier training and 10,000 handwritten digit images for the classifier testing, both drawn from the same distribution. All these black and white digits are size normalized, and centered in a fixed-size image where the center of the intensity lies at the center of the image with 28 × 28 pixels. The dimensionality of each image sample vector is 28 * 28 = 784, where each element is binary (Deng, 2012).

The reasons why we choose to use this MNIST handwritten digit database are various. Firstly, as mentioned above it is a standard which is a relatively simple database for fast- testing theories and algorithms. And we want to test neural networks applied to the practical problems in the real world, the handwritten digits in MNIST database have already been preprocessed including segmentation and normalization, so that it could be a good start for us spending minimal efforts on preprocessing and formatting. Besides, there are lots of researchers evaluating their theories and algorithms using MNIST, which means we can compare our results with the results from a rather comprehensive set of literature.

The purpose of this paper is to define how neural networks are used to resolve a problem of MNIST handwritten digit classification. Our work is to design a neural network model and then implement it to solve the classification problem. Besides, some extra experiments have been done to test different methods that may consequently influence the performance of our model.

As we know, Artificial neural networks (ANNs) are inspired by biological central systems in brains and have been utilized in a variety of applications ranging from modeling, classification, pattern recognition, and multivariate data analysis (Basheer and Hajmeer, 2000). And in practical ANN application, Back-propagation neural network is widely applied. Back-propagation algorithm is put forward by D. Rumelhart and J. McCelland in 1986 and they presented the neural networks using BP methods are called backpropagation neural networks (BPNN) (Rumelhart, Hinton, and McClelland, 1986). In this paper, we use Back-propagation neural network to achieve the classification problem. Back-propagation neural networks are supervised multi- layer feed forward neural networks, commonly consist of an input layer, an output layer, and one or several hidden layers.

Here is a basic model of the three- layer neural network shown in Figure 1. There are three layers in the network called input layer, hidden layer, and output layer respectively. Layers are fully interconnected with each other but there are no interconnections in the units of the same layer. The nodes represent computational units and need to obtain inputs which should be processed in neurons to produce the outputs. The functions to process inputs might be summing the figures. A basic neuron contains several elements: input, weights, activation function, threshold, and output. Weights are multiplied with inputs and then added in summing function and the sum is processed in activation function and finally, the model gives an output.

2 Method

As mentioned above, in this paper we use MNIST handwritten digit database in which the handwritten digits have been preprocessed including segmentation and normalization. There are 60,000 training images and 10,000 test images, and the dimensionality of each image sample vector is 28 * 28 = 784, where each element is binary.

2.1 Back-propagation neural network setting up

As discussed in the section above, Back-propagation neural networks are supervised multi-layer feed forward neural networks, commonly consist of an input layer, an output layer, and one or several hidden layers. Here we build a neural network with only one hidden layer. Input layer contains 28 * 28 = 784 neurons, representing the features; Hidden layer contains 300 neurons, using Sigmoid as activation function; And output layer contains 10 neurons representing the digits from 0 to 9.

Besides, we use cross-entropy error function as network loss function. It seems that cross-entropy error function is more suitable for classification problems than mean squared error (MSE) function. And it is convenient to use CrossEntropyLoss () to achieve it. As for the optimizer, here we choose to use Adam optimizer implementing the Adam algorithm which can be achieved by using Adam ().

After defining the neural network, we train the model by batch. In order to set the appropriate parameters such as batch_size, number_of_epochs and learning rate, we conducted several tests to evaluate. Honestly, it is really difficult to decide. There should be a balance among these parameters, contributing to the performance of the neural network together. Finally, the system parameters are given as number_of_epochs = 6, learning rate = 0.005 and batch_size = 100.

2.2 Evaluation methods

In order to determine the performance of the neural network and predictions and report the results produced by our neural network, we choose to calculate the training accuracy and testing accuracy. Plotting historical loss from "all losses" during network learning is also a visualized representation.

2.3 Modify original model with Autoencoder

T.D. Gedeon and D. Harris put forward that the major disadvantages of the back-propagation method are that it can be slow to train networks. If the new units end up duplicating the functionality of existing units, the speed of the network by increasing its size have decreased (Gedeon and Harris, 1992). They try to progressively reduce the size of the compression layer for the desired level of image quality.

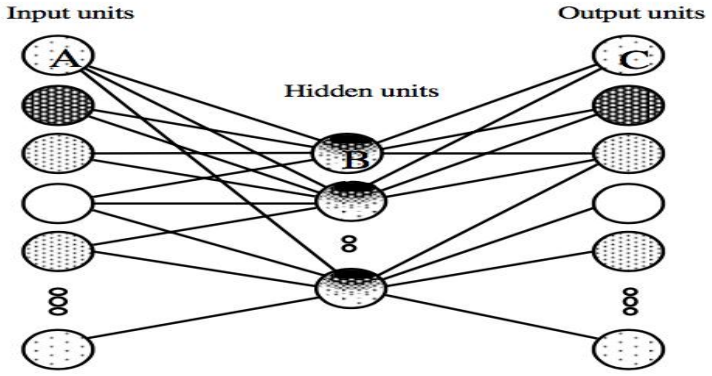

Figure 1: Auto-associative network topology [6]

Here is a standard auto-associative network put forward by T.D. Gedeon and D. Harris (Gedeon et al). The architecture is a n input, m unit hidden layer, and n output network, each of the pixels of the image being a single input, and output. The input value to A is the same as the output value at C, and so on. The first layer of weights from the inputs to the hidden units can be seen as implementing a compression function on the input pattern. The second layer of weights from the hidden units to the outputs implement a decompression function on the compressed data. That is, the hidden unit activations are mapped to recreate an approximation of the original data. The significance of training an auto -associative network for image compression is that the hidden neurons learn a compressed representation for the input patterns. Ideally, the data produced from the hidden layer can be regarded as a substitution representing the main information of the original input data. It can be used to reduce the network size and thus increase the speed of the neural network, improving the performance.

This technique inspires us that we can also use autoencoder to compress the image dataset. An autoencoder is a neural network that is designed to learn a representation of its input by attempting to copy its input to output. It can be thought of as two separate functions, an encoder, and a decoder.

We try to modify and test autoencoder on MNIST handwritten digit database, looking for whether it helps or not. The architecture we used is a 28 * 28 input, 6 unit hidden layer, and 28 * 28 output network, each of the pixels in the 28 by 28 parts of the image being a single input, and output. Actually, we add some layers between the input layer and hidden layer as well as between hidden layer and output layer for better effects.

3 Results and Discussion

For analysis purpose, we present the results comparison by tables and figures. Due to the random bias that happens on a single result, testing accuracy that we use to compare is the average among 10 runs.

3.1 Results of the original model, Autoencoder model, and CNN model

Here show the results after several tests on the original model, Autoencoder model, and CNN model. All the system parameters are given as number_of_ epochs = 6, learning rate = 0.005. And there is a certain degree of randomness in the results

The results in Table 1 shows that as for the MNIST handwritten digit database, our original neural network model performs well achieving the accuracy rate 97.65%. The modified model using Autoencoder doesn't perform well since its accuracy rate is only 74.38% while the modified model using CNN achieves 98.84%, performing a little better than our original neural network.

Besides, historical loss from "all losses" during network learning are plotted for each model and they are shown above. Figure 5 shows the loss plot of the original model, Figure 6 is the loss plot of the Autoencoder model, and Figure 7 indicates the loss of the CNN model. It can be easily observed that the loss plot of the CNN model is smoother than others. And we can also see the great amount of loss during the training process of the Autoencoder model since its plot is more twisted.

The reason why the Autoencoder model doesn't work well may be that in the Autoencoder network topology, the data produced from the hidden layer may lose information related to the categories. Regarding it as a substitution of the original input data is possible to cause an error.

The reason why the CNN model performs better may be as follow: firstly, the CNN structure is inspired by the visual pathway which could help to perform optical recognition tasks; secondly, CNNs make use of convolutional layers instead of densely connected neurons; thirdly, convolutional layers perform convolutions rather than general matrix multiplication.

Figure 2: mnist digit samples

The results of the original Autoencoder, as well as the Conv_Autoencoder, are presented above in Figure 8 and Figure 9. And the results in Table 2 shows that the loss of the original model as well as the Conv_ Autoencoder model. The loss of the Conv_Autoencoder is a little less than the loss of the original Autoencoder model so that we can conclude that the Conv_Autoencoder performs better than the original Autoencoder as for image compression.

3.3 Results compared with other results in a published research paper

Besides, we search in the Google Scholar to find other researchers who also research on the MNIST handwritten digit database. And we compare our results with their results.

The paper written by D. Ciresan et al suggests that the performance, about 0.27% error rate or 27 errors in the full 10,000 test set, is achieved by a committee of convolutional nets (with elastic distortion in augmenting the training set) (Ciresan et al, 2011). The accuracy of the MNIST handwritten digit database in their experiment is a little bit better than the results in our CNN model. However, the modified model using Autoencoder in our experiment gains the relatively worse results, which need to study and improve further.

4 Conclusion and Future Work

In this paper, we conduct an experiment implementing Back- propagation Neural Network to achieve the classification of the MNIST handwritten digit database. In the experimental model, 28 * 28 = 784 pixels are regarded as input and 10 different classes of digits from 0 to 9 as output. Besides, we use classification accuracy and loss plot to determine the performance of the neural network. After testing on various parameters of the model, we set the system parameters as follow: number_of_ epochs = 6, learning rate = 0.005 and batch_size = 100. The experimental results show that to a certain extent, the back-propagation neural network can be used to solve classification problem in the real world.

Besides, we try to achieve image compression with Autoencoder and determine its effects. It does help to reduce the network size and thus increase the speed of the neural network. However, the accuracy rate cannot be guaranteed. Then we try to modify the structure of the original neural network into a Convolutional Neural Network (CNN). The results indicate that CNN can be used to improve the performance during solving image recognition problem. Besides, we combine CNN with Autoencoder into a Conv _Autoencoder structure and tests conducted suggest that he Conv_Autoencoder performs better than the original Autoencoder during image compression.

It seems that the results of our Autoencoder model are a little bit worse, which means further work is necessary. In the future, we will go further with our modified model using Autoencoder and try to improve its performance. Besides, our CNN model should be studied and improved as well since there can be better performance from the related literature.

References

1. Basheer, I.A., and Hajmeer, M., 2000. Artificial neural networks: fundamentals, computing, design, and application. *Journal of microbiological methods*, *43*(1), pp.3-31.
2. Ciresan, D.C., Meier, U., Gambardella, L.M. and Schmidhuber, J., 2011, September. Convolutional neural network committees for handwritten character classification. In *Document Analysis and Recognition (ICDAR), 2011 International Conference on* (pp. 1135-1139). IEEE.
3. Deng, L., 2012. The MNIST database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, *29*(6), pp.141-142.
4. Gedeon, T.D. and Harris, D., 1992, June. Progressive image compression. In *Neural Networks, 1992. IJCNN., International Joint Conference on* (Vol. 4, pp. 403-407). IEEE.
5. Gedeon, T.D., Catalan, J.A. and Jin, J., Image Compression using Shared Weights and Bidirectional Networks. In *Proceedings 2nd International ICSC Symposium on Soft Computing (SOCO'97)* (pp. 374-381).
6. LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P., 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), pp.2278-2324.
7. O'Reilly | Safari., 2018. *TensorFlow for Deep Learning*. [online] Available at: https://www.safaribooksonline.com/library/view/tensorflow-for-deep/9781491980446/ch01.html [Accessed 30 May 2018].
8. Rumelhart, D.E., Hinton, G.E. and McClelland, J.L., 1986. A general framework for parallel distributed processing. *Parallel distributed processing: Explorations in the microstructure of cognition*, *1*, pp.45-76.
9. Yann.lecun.com., n.d. *MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges*. [online] Available at: http://yann.lecun.com/exdb/mnist/ [Accessed 30 May 2018].
10. Yann.lecun.com., n.d. *MNIST Demos on Yann LeCun's website*. [online] Available at: http://yann.lecun.com/exdb/lenet/ [Accessed 30 May 2018].