Neural Networks based Handwritten Digit Recognition

Mr. Yogesh Sharma¹, Mr. Jaskirat Singh Bindra², Mr. Kushagr Aggarwal³, Mr. Mayur Garg⁴
¹Assistant Professor, Maharaja Agrasen Institute of Technology, New Delhi
²,³,⁴Department of Computer Science and Engineering, Maharaja Agrasen Institute of Technology, New Delhi

ABSTRACT - In the field of Artificial Intelligence, scientists have made many enhancements that helped a lot in the development of millions of smart devices. On the other hand, scientists brought a revolutionary change in the field of image processing and one of the biggest challenges in it is to identify data in both printed as well as hand-written formats. One of the most widely used techniques for the validity of these types of document is NEURAL NETWORKS. Neural networks currently provide the best solutions to many problems in image recognition, speech recognition, and natural language processing. Handwritten Digit Recognition is an extensively employed method to transform the data of handwritten form into digital format. This data can be used anywhere, in any field, like database, data analysis, etc. There are multitude of techniques introduced now that can be used to recognize handwriting of any form. In the suggested system, we will handle the issue of machine reading numerical digits using the technique of NEURAL NETWORKS. We aim to learn the basic functioning of the neural network and expect to find the correlation between all the parameters i.e. No. of layers, Layer Size, Learning Rate, Size of the Training sets and the associated accuracy achieved by the neural net in identifying the handwritten digits, by comparing the accuracy achieved on different sets of the four variable parameters varying in the suitable range.

I. INTRODUCTION

In the field of Artificial Intelligence, scientists have made many enhancements that helped a lot in the development of millions of smart devices. On the other hand, scientists brought a revolutionary change in the field of image processing and one of the biggest challenges in it is to identify documents in both printed as well as hand-written formats. One of the most widely used techniques for the validity of these types of document is NEURAL NETWORKS. Neural networks currently provide the best solutions to many problems in image recognition, speech recognition, and natural language processing. Handwritten Digit Recognition is an extensively employed method to transform the data of handwritten form into digital format. This data can be used anywhere, in any field, like database, data analysis, etc. There are millions of techniques introduced now that can be used to recognize handwriting of any form. In the suggested system, we will handle the issue of machine reading numerical digits using the technique of NEURAL NETWORKS. We aim to learn the basic functioning of the neural network and expect to find the correlation between all the parameters i.e. No. of layers, Layer Size, Learning Rate and the associated accuracy achieved by the neural net in identifying the handwritten digits, by comparing the accuracy achieved on different sets of the three variable parameters varying in the suitable range. We also plan to find the relation between the size of the training set used to train the neural network and find the accuracy it achieves in recognising the handwritten digits accordingly.

We have used a convolutional neural network in this case which are already recognised for showing outstanding result for its use in image and speech recognition. Then number of hidden layers used are varied from one to three with size of each hidden layer varying from 10-50. The accuracy of the neural net in recognising the handwritten digits is also observed at different learning rates ranging...
from 0.1 - 2.0. Independent observations for the accuracy achieved are also observed for the neural net being trained on datasets of size 5000 to 50000 image sets.

Objectives
- The objective is to make a system that can classify a given input correctly.
- Correctly train a neural network to recognize handwritten digits.
- Analyze the accuracy and of the neural network with respect to the size of Training Sets and number of Hidden Layers.
- Analyze the accuracy and of the neural network with respect to the size of Hidden Layers
- Analyze the accuracy and of the neural network with respect to the learning rate.

II. METHODOLOGIES

What is a neural net?
An Artificial neural network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurons). ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true of ANNs as well.

Simply a Neural Network can be defined as a mathematical function that maps a given input to a desired output.

Neural Networks consist of the following components
- An input layer, x
- An arbitrary amount of hidden layers
- An output layer, y
- A set of weights and biases between each layer, W and b
- A choice of activation function for each hidden layer, σ.

The diagram below shows the architecture of a 2-layer Neural Network (the input layer is typically excluded when counting the number of layers in a Neural Network)

Technologies Used
Python 3.6 - Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales.
Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including by metaprogramming and metaobjects (magic methods)). Many other paradigms are supported via extensions, including design by contract and logic programming.

Python uses dynamic typing, and a combination of reference counting and a cycle-detecting garbage collector for memory management. It also features dynamic name resolution (late binding), which binds method and variable names during program execution.

Python’s design offers some support for functional programming in the Lisp tradition. It has filter(), map(), and reduce() functions; list comprehensions, dictionaries, and sets; and generator expressions. The standard library has two modules (itertools and functools) that implement functional tools borrowed from Haskell and Standard ML.

NumPy 1.15 - NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities
- Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data.
- Arbitrary data-types can be defined using Numpy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

IDLE - IDLE (short for integrated development and learning environment) is an integrated development environment for Python, which has been bundled with the default implementation of the language since 1.5.2b1. It is packaged as an optional part of the Python packaging with many Linux distributions. It is completely written in Python.

IDLE is intended to be a simple IDE and suitable for beginners, especially in an educational environment. To that end, it is cross-platform, and avoids feature clutter. According to the included README, its main features are:

- Multi-window text editor with syntax highlighting, autocompletion, smart indent and other.
- Python shell with syntax highlighting.
- Integrated debugger with stepping, persistent breakpoints, and call stack visibility.

MNIST Dataset - The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems. The database is also widely used for training and testing in the field of machine learning. It was created by "re-mixing" the samples from NIST's original datasets. The creators felt that since NIST's training dataset was taken from American Census Bureau employees, while the testing dataset was taken from American high school students, it was not well-suited for machine learning experiments. Furthermore, the black and white images from NIST were normalized to fit into a 28x28 pixel bounding box and anti-aliased, which introduced grayscale levels.

The MNIST database contains 60,000 training images and 10,000 testing images. Half of the training
set and half of the test set were taken from NIST’s training dataset, while the other half of the training set and the other half of the test set were taken from NIST’s testing dataset.

Matplotlib - Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+. There is also a procedural “pylab” interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB, though its use is discouraged. SciPy makes use of matplotlib. Matplotlib was originally written by John D. Hunter, has an active development community, and is distributed under a BSD-style license. As of 23 June 2017, matplotlib 2.0.x supports Python versions 2.7 through 3.6. Matplotlib 1.2 is the first version of matplotlib to support Python 3.x. Matplotlib 1.4 is the last version of matplotlib to support Python 2.6. Matplotlib has pledged to not support Python 2 past 2020 by signing the Python 3 Statement.

Variable Parameters Used

1. Number of hidden layers

A single hidden layer neural networks is capable of [universal approximation]. The universal approximation theorem states that a feed-forward network, with a single hidden layer, containing a finite number of neurons, can approximate continuous functions with mild assumptions on the activation function. The first version of this theorem was proposed by Cybenko (1989) for sigmoid activation functions. Hornik (1991) expanded upon this by showing that it is not the specific choice of the activation function, but rather the multilayer feedforward architecture itself which allows neural networks the potential of being universal approximators.

Due to this theorem you will see considerable literature that suggests the use of a single hidden layer. This all changed with Hinton, Osindero, & Teh (2006). If a single hidden layer can learn any problem, why did Hinton et. al. invest so heavily in deep learning? Why do we need deep learning at all? While the universal approximation theorem states/proves that a single layer neural network can learn anything, it does not specify how easy it will be for that neural network to actually learn something. Every since the multilayer perceptron, we’ve had the ability to create deep neural networks.

Traditionally, neural networks only had three types of layers: hidden, input and output. These are all really the same type of layer if you just consider that input layers are fed from external data (not a previous layer) and output feed data to an external destination (not the next layer). These three layers are now commonly referred to as dense layers. This is because every neuron in this layer is fully connected to the next layer. In the case of the output layer the neurons are just holders, there are no forward connections. Modern neural networks have many additional layer types to deal with. In addition to the classic dense layers, we now also have dropout, convolutional, pooling, and recurrent layers. Dense layers are often intermixed with these other layer types.

Problems that require more than two hidden layers were rare prior to deep learning. Two or fewer layers will often suffice with simple data sets. However, with complex datasets involving time-series or computer vision, additional layers can be helpful.
The following table summarizes the capabilities of several common layer architectures.

**Table: Determining the Number of Hidden Layers**

<table>
<thead>
<tr>
<th>Num Hidden Layers</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>Only capable of representing linear separable functions or decisions.</td>
</tr>
<tr>
<td>1</td>
<td>Can approximate any function that contains a continuous mapping from one finite space to another.</td>
</tr>
<tr>
<td>2</td>
<td>Can represent an arbitrary decision boundary to arbitrary accuracy with rational activation functions and can approximate any smooth mapping to any accuracy.</td>
</tr>
<tr>
<td>&gt;2</td>
<td>Additional layers can learn complex representations (sort of automatic feature engineering) for layer layers.</td>
</tr>
</tbody>
</table>

Here we have observed the functioning of the neural nets for three cases -

- With a single hidden layer
- With two hidden layers
- With three hidden layers

2. Size of hidden layers

Deciding the number of neurons in the hidden layers is a very important part of deciding your overall neural network architecture. Though these layers do not directly interact with the external environment, they have a tremendous influence on the final output. Both the number of hidden layers and the number of neurons in each of these hidden layers must be carefully considered.

Using too few neurons in the hidden layers will result in something called underfitting. Underfitting occurs when there are too few neurons in the hidden layers to adequately detect the signals in a complicated data set.

Using too many neurons in the hidden layers can result in several problems. First, too many neurons in the hidden layers may result in overfitting. Overfitting occurs when the neural network has so much information processing capacity that the limited amount of information contained in the training set is not enough to train all of the neurons in the hidden layers. A second problem can occur even when the training data is sufficient. An inordinately large number of neurons in the hidden layers can increase the time it takes to train the network. The amount of training time can increase to
the point that it is impossible to adequately train the neural network. Obviously, some compromise must be reached between too many and too few neurons in the hidden layers. There are many rule-of-thumb methods for determining an acceptable number of neurons to use in the hidden layers, such as the following:

- The number of hidden neurons should be between the size of the input layer and the size of the output layer.
- The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer.
- The number of hidden neurons should be less than twice the size of the input layer.

Here the size of the hidden layers has been varied from 10 neurons to 50 neurons for each layer.

3. Learning rate

The learning rate is one of the most important hyper-parameters to tune for training deep neural networks.

Deep learning models are typically trained by a stochastic gradient descent optimizer. There are many variations of stochastic gradient descent: Adam, RMSProp, Adagrad, etc. All of them let you set the learning rate. This parameter tells the optimizer how far to move the weights in the direction opposite of the gradient for a mini-batch.

If the learning rate is low, then training is more reliable, but optimization will take a lot of time because steps towards the minimum of the loss function are tiny. If the learning rate is high, then training may not converge or even diverge. Weight changes can be so big that the optimizer overshoots the minimum and makes the loss worse.

There are multiple ways to select a good starting point for the learning rate. A naive approach is to try a few different values and see which one gives you the best loss without sacrificing speed of training. The trick is to train a network starting from a low learning rate and increase the learning rate exponentially for every batch.

Selecting a starting value for the learning rate is just one part of the problem. Another thing to optimize is the learning schedule: how to change the learning rate during training. The conventional wisdom is that the learning rate should decrease over time, and there are multiple ways to set this up: step-wise learning rate annealing when the loss stops improving, exponential learning rate decay, cosine annealing, etc.

For this project we have varied the learning rate from 0.1 to 2.0.

4. Training Dataset

In this particular project size of the training data set has been varied in the range 5000 - 50000.

5. Weights

In an artificial neuron, a collection of weighted inputs is the vehicle through which the neuron engages in an activation function and produces a decision (either firing or not firing). Typical artificial neural networks have various layers including an input layer, hidden layers and an output layer. At each layer, the individual neuron is taking in these inputs and weighting them accordingly. This simulates the biological activity of individual neurons, sending signals with a given synaptic weight from the axon of a neuron to the dendrites of another neuron.
We can utilize specific mathematical equations and visual modeling functions to show how synaptic weights are used in an artificial neural network. In a system called backpropagation, input weights can be altered according to the output functions as the system learns how to correctly apply them. All of this is foundational to how neural networks function in sophisticated machine learning projects. We have assigned the weights using following techniques:
- Using standard distribution to randomly assign values to weights.
- Using the value 0 initially assignment.

III. EXPERIMENTAL RESULTS

Comparison of accuracy
1. Single Hidden layer
Specifications -
- The results have been averaged for 7 datasets to remove the effect of random initialisation of weights.
- The learning rate was varied from 0.1 to 2.0 in increments of 0.1
- Only 1 hidden layer was used.
- The size of hidden layer was varied from 10 to 50 in increments of 5.
- 1260 ANN were trained and tested in the process

Results -

![Figure 1 - Accuracy vs Learning Rate for one hidden layer](image1)

![Figure 2 - Accuracy vs Size of one Hidden Layer](image2)

Observations made -
- Maximum average accuracy (over 7 datasets) = 94.59 was achieved with Learning Rate = 0.3 and Size of Hidden Layer = 50.
- Keeping all other parameters similar, Maximum average accuracy = 93.08 was achieved with Learning Rate = 0.2.
• Keeping all other parameters similar, Maximum average accuracy = 92.49 was achieved with Size of Hidden Layer = 50.
• It was observed that accuracy peaked for the ANN for Learning rate in the range 0.2 to 0.3 and followed a gradual (almost linear i.e. constant slope) descent as the Learning rate increased further.
• It was observed that accuracy peaked for the ANN for Size of Hidden Layer = 50 and that accuracy rose with the increase in the size of the Hidden Layer but the ascent was not linear and seemed to slow down (decreasing slope) as the size of Hidden Layer kept increasing further.

2. Single Hidden layer with initial weight zero
Specifications -
• The learning rate was varied from 0.1 to 2.0 in increments of 0.1
• Only 1 hidden layer was used.
• The size of hidden layers was varied from 10 to 50 in increments of 5.
• 180 ANN were trained and tested in the process.

Results -

*Figure 3 - Accuracy vs Learning Rate when initial weights are zero*

*Figure 4 - Accuracy vs Size of Hidden layer when initial weights are zero*

Observations made -
• Maximum accuracy =36.41 was achieved with Learning Rate = 0.5 and Size of Hidden Layer = 10.
• Keeping all other parameters similar, Maximum average accuracy = 33.95 was achieved with Learning Rate = 0.4.
• Keeping all other parameters similar, Maximum average accuracy = 31.23 was achieved with Size of Hidden Layer = 35.
• It was observed that accuracy peaked for the ANN for Learning rate in the range 0.2 to 0.3 (similar to the results where weights in ANN were initialised using standard distribution) and followed a descent as the Learning rate increased further but no clearly visible trend was observed.
• It was observed that accuracy peaked for the ANN for Size of Hidden Layer = 35 but no clearly visible trend was observed relating the size of Hidden Layer and accuracy of the ANNs.
• It was observed that retraining and retesting an ANN with the same specifications had no effect on accuracy since no random factor was involved.

3. Two Hidden layers
Specifications -
• The results have been averaged for 3 datasets to remove the effect of random initialisation of weights.
• The learning rate was varied from 0.1 to 2.0 in increments of 0.1
• 2 Hidden Layers were used
• The size of each hidden layer was varied from 10 to 50 in increments of 5.
• 4860 ANN were trained and tested in the process

Results -

![Accuracy vs Learning Rate for two hidden layers](image1)

Figure 5 - Accuracy vs Learning Rate for two hidden layers

![Accuracy vs Size of Individual Hidden Layers for two hidden layers](image2)

Figure 6 - Accuracy vs Size of Individual Hidden Layers for two hidden layers
Observations made -

- Maximum average accuracy (over 3 datasets) = 94.26 was achieved with Learning Rate = 0.3 and Size of Hidden Layer = (50, 30) in that order.
- Keeping all other parameters similar, Maximum average accuracy = 92.71 was achieved with Learning Rate = 0.2.
- Keeping all other parameters similar, Maximum average accuracy = 91.37 was achieved with Size of Hidden Layer = 50 and that accuracy rose with the increase in the size of the each Hidden Layer.

It was observed that accuracy peaked for the ANN for Learning rate in the range 0.2 to 0.3 and followed a gradual (almost linear i.e. constant slope) descent as the Learning rate increased further. The descent was almost identical to the one observed in ANNs with only one hidden layer.

It was observed that accuracy peaked for the ANN for Size of each Hidden Layer = 50 and that accuracy rose with the increase in the size of the each Hidden Layer.
The ascent of accuracy with respect to increase in size of each Hidden Layer was not linear and seemed to slow down (decreasing slope) as the size of each Hidden Layer kept increasing further. Moreover, accuracy rose much quicker (slope decreased slowly) with increase in size of the first Hidden Layer than with the increase in size of the second Hidden Layer.

The difference in accuracy in when order of the size of hidden layer is reversed was more pronounced when the difference between the size of each layer was higher and vice versa. The general rule observed was that order of the size of the hidden layer (in case of unequal hidden layers) must be so chosen as to maximise the number of weights in the ANN.

4. Three Hidden layers
Specifications -
- The results have been averaged for 3 datasets to remove the effect of random initialisation of weights.
- The learning rate was varied from 0.1 to 1.0 in increments of 0.1
- 3 Hidden Layers were used
- The size of each hidden layer was varied from 30 to 50 in increments of 5.
- 3750 ANN were trained and tested in the process

Results -

- Figure 9 - Accuracy vs Learning Rate for three hidden layers
- Figure 10 - Accuracy vs Size of Individual Hidden Layers for three hidden layers
Observations made -

- Maximum average accuracy (over 3 datasets) = 93.19 was achieved with Learning Rate = 0.3 and Size of Hidden Layer = (40, 50, 40) in that order.
- Keeping all other parameters similar, Maximum average accuracy = 92.27 was achieved with Learning Rate = 0.3.
- Keeping all other parameters similar, Maximum average accuracy = 91.67 was achieved with Size of Hidden Layer = (50, 45, 50) in that order.
- It was observed that accuracy peaked for the ANN for Learning rate in the range 0.2 to 0.3 and followed a gradual (almost linear i.e. constant slope) descent as the Learning rate increased further. The descent was almost identical to the one observed in ANNs with one and two hidden layer.
- It was observed that accuracy peaked for the ANN for Size of each Hidden Layer = 50 and that accuracy rose with the increase in the size of the each Hidden Layer.
- The ascent of accuracy with respect to increase in size of each Hidden Layer was not linear and seemed to slow down (decreasing slope) as the size of each Hidden Layer kept increasing further. Moreover, accuracy rose much quicker (slope decreased slowly) with increase in size of the first Hidden Layer whereas accuracy showed only a nominal rise with increase in size of second and third hidden layer.

5. Single Hidden Layer with varying training sets

Specifications -

- The size of training set was varied from 5,000 to 50,000 in increments of 5,000.
- Only 1 hidden layer was used.
- 1100 ANN were trained and tested in the process and the results were averaged across various learning rates and sizes of the hidden layer.
Observations made -

- Keeping all other parameters similar, Maximum average accuracy = 93.98 was achieved using the training set of size = 50,000.
- It was observed that accuracy peaked for the ANN when it was trained using the training set of size = 50,000 and that accuracy rose with the increase in the size of the training set but the ascent was not linear and seemed to slow down (decreasing slope) as the size of the training set kept increasing further.

IV. CONCLUSION

Summary - The artificial neural network to classify handwritten digits was successfully created and by tuning the various parameters (no. of hidden layers, size of hidden layers, learning rate and size of the training set used), accuracy of upto 95% was achieved. It was observed that accuracy peaked for the ANN for Learning rate in the range 0.2 to 0.3. Size of Hidden Layer = 50 and Size of Training set = 50,000 in the range tested. Accuracy fell almost linearly with the learning rate when the learning rate was moved out of its optimal range. Accuracy of the ANN increased with increase in size of the hidden layers and the size of the training set used but ascent was not linear and had a decreasing slope as they kept increasing further. On increasing the number of hidden layers, higher accuracy was observed even if the the size of hidden layer and the learning rate was not in the optimal range.

Future scope -

- Handwritten digit recognition can be used in numerous fields to convert data in handwritten format to digital format which can then be easily stored, transferred, processed and analysed for further usage.
- The effects of various parameters on accuracy of the ANN can be used to efficiently train any ANN to gain the most accurate output. The various parameters can be tuned in accordance to the quality of the results desired.

REFERENCES


