MANTECH Publications
Journal of Artificial Intelligence, Machine Learning and Soft Computing
Volume 4, Issue 2, July-December 2019

Handwritten Digit Recognition by Neural Networks

Shivagami R., Ramesh P.
Department of Computer Engineering
Sona College of Engineering & Technology, Tamil Nadu
Email ID: *shivagamir657@rediffmail.com*

Abstract

This paper chronicles the development of an artificial neural network designed to recognize handwritten digits. The neural network described here is not a general-purpose neural network, and it's not some kind of a neural network workbench. Rather, we will focus on one very specific neural network (a five-layer convolutional neural network) built for one very specific purpose (to recognize handwritten digits). One of our goals here was to reproduce the accuracy achieved by Dr. LeCun, who was able to train his neural network to achieve 99.18% accuracy (i.e., an error rate of only 0.82%). This error rate served as a type of "benchmark", guiding our work.

Keywords: *Digit Recognition, Neural Networks, Handwritten Digit Recognition, Artificial Neural Networks*

INTRODUCTION

The idea of using neural networks for the purpose of recognizing handwritten digits is not a new one. The inspiration for the architecture described here comes from articles written by two separate authors.

The first is Dr. Yann LeCun, who was an independent discoverer of the basic back propagation algorithm. Dr. LeCun hosts an excellent site on his research into neural networks. In particular, you should view his "Learning and Visual Perception" section, which uses animated GIFs to show results of his research.

The MNIST database (which provides the database of handwritten digits) was developed by him. We used two of his publications as primary source materials

Page 53-62 © MANTECH PUBLICATIONS 2019. All Rights Reserved

for much of our work, and WE highly recommend reading his other publications too (they're posted at his site). Unlike many other publications on neural networks, Dr. LeCun's publications are not inordinately theoretical and math-intensive; rather, they are extremely readable, and provide practical insights and explanations.

One of our goals here was to reproduce the accuracy achieved by Dr. LeCun, who was able to train his neural network to achieve 99.18% accuracy (i.e., an error rate of only 0.82%). This error rate served as a type of "benchmark", guiding our work.

As a final introductory note, I'm not overly proud of the source code, which is most definitely an engineering work-in-progress. We started out with good intentions, to make source code that was flexible and easy to understand and to change. As things progressed, the code started to turn ugly.

WE began to write code simply to get the job done, sometimes at the expense of clean code and comprehensibility. To add to the mix, w,e were also experimenting with different ideas, some of which worked and some of which did not. As we removed the failed ideas, WE did not always back out all the changes and there are therefore some dangling stubs and dead ends. WE contemplated the possibility of not releasing the code. But that was one of our criticisms of the articles WE read: none of them included code. So, with trepidation and the recognition that the code is easy to criticize and could really use a re-write, here it is.

SOME NEURAL NETWORK THEORY This is not a neural network tutorial, but to understand the code and the names of the variables used in it, it helps to see some neural networks basics. The following discussion is not completely general. It considers only feed-forward neural networks, that is, neural networks composed of multiple layers, in which each layer of neurons feeds only the very next layer of neurons, and receives input only from the immediately preceding layer of neurons. In other words, the neurons don't skip layers.

Consider a neural network that is composed of multiple layers, with multiple neurons in each layer. Focus on one neuron in layer n, namely the i-th neuron. This neuron gets its inputs from the outputs of neurons in the previous

Page 53-62 © MANTECH PUBLICATIONS 2019. All Rights Reserved

layer, plus a bias whose valued is one ("1"). WE use the variable "x" to refer to outputs of neurons. The i-th neuron applies a weight to each of its inputs, and then adds the weighted inputs together so as to obtain something called the "activation value". WE use the variable "y" to refer to activation values. The i-th neuron then calculates its output value "x" by applying an "activation function" to the activation value. WE use the letter "F()" to refer to the activation function.

The activation function is sometimes referred to as a "Sigmoid" function, a "Squashing" function and other names, since its primary purpose is to limit the output of the neuron to some reasonable range like a range of -1 to +1, and thereby inject some degree of non-linearity into the network. Here's a diagram of a small part of the neural network; remember to focus on the i-th neuron in level n:

This is what each variable means:

is the output of the i-th neuron in layer n

is the output of the j-th neuron in layer n-1

is the output of the k-th neuron in layer n-1

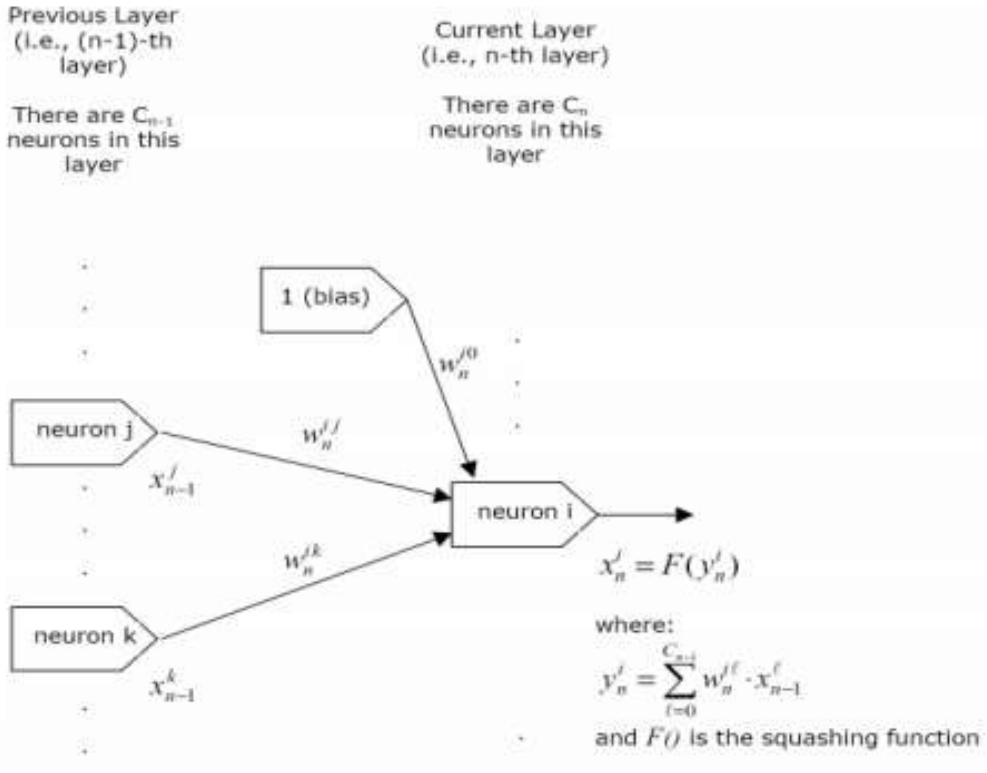


Figure: 1

Page 53-62 © MANTECH PUBLICATIONS 2019. All Rights Reserved

is the weight that the i-th neuron in layer n applies to the output of the j-th neuron from layer n-1 (i.e., the previous layer). In other words, it's the weight from the output of the j-th neuron in the previous layer to the i-th neuron in the current (n-th) layer.

is the weight that the i-th neuron in layer n applies to the output of the k-th neuron in layer n-1
is the general feed-forward equation, where F() is the activation function. We will discuss the activation function in more detail in a moment.

How does this translate into code and C++ classes? The way WE saw it, the above diagram suggested that a neural network is composed of objects of four different classes: layers, neurons in the layers, connections from neurons in one layer to those in another layer, and weights that are applied to connections. Those four classes are reflected in the code, together with a fifth class -- the neural network itself -- which acts as a container for all other objects and which serves as the main interface with the outside world. Here's a simplified view of the classes. Note that the code makes heavy use of std::vector, particularly std::vector< double >:

// simplified view: some members have been omitted,

// and some signatures have been altered

// helpful typedef's typedef std::vector< NNLayer* >

VectorLayers; typedef std::vector< NNWeight* >

VectorWeights; typedef std::vector< NNNeuron* >

VectorNeurons; typedef std::vector< NNConnection >

VectorConnections; // Neural Network class

class NeuralNetwork

{public: NeuralNetwork();

virtual ~NeuralNetwork();

void Calculate(double* inputVector, UINT iCount,

double* outputVector = NULL, UINT oCount = 0);

Page 53-62 © MANTECH PUBLICATIONS 2019. All Rights Reserved

```
void Backpropagate( double *actualOutput,
double *desiredOutput, UINT count );
VectorLayers m_Layers;
};
// Layer class
class NNLayer
{
public:
NNLayer( LPCTSTR str, NNLayer* pPrev = NULL );
 virtual ~NNLayer();
void Calculate();
void Backpropagate( std::vector< double >& dErr_wrt_dXn /* in */,
 std::vector< double >& dErr_wrt_dXnm1 /* out */,
double etaLearningRate );
 NNLayer* m_pPrevLayer;
VectorNeurons m_Neurons;
VectorWeights m_Weights;
 };
 // Neuron class
ClassNNNeuron
 {
 public:
NNNeuron( LPCTSTR str );
virtual ~NNNeuron();
void AddConnection( UINT iNeuron, UINT iWeight );
 void AddConnection( NNConnection const & conn );
double output;
VectorConnections m_Connections;
};
// Connection class
 class NNConnection
 {
public: NNCo nnection(UINT neuron = ULONG_MAX, UINT weight = ULONG_MAX);
```


Page 53-62 © MANTECH PUBLICATIONS 2019. All Rights Reserved

```
virtual ~NNConnection();

 UINT NeuronIndex;

UINT WeightIndex; };
 // Weight class class NNWeight
 {
 public:

NNWeight( LPCTSTR str,

double val = 0.0 );

virtual ~NNWeight();

double value;

};
```

The human visual system is one of the wonders of the world. Consider the following sequence of handwritten digits:



Most people effortlessly recognize those digits as 504192. That ease is deceptive. In each hemisphere of our brain, humans have a primary visual cortex, also known as V1, containing 140 million neurons, with tens of billions of connections between them. And yet human vision involves not just V1, but an entire series of visual cortices - V2, V3, V4, and V5 - doing progressively more complex image processing.

We carry in our heads a supercomputer, tuned by evolution over hundreds of millions of years, and superbly adapted to understand the visual world. Recognizing handwritten digits isn't easy. Rather, we humans are stupendously, astoundingly good at making sense of what our eyes show us. But nearly all that work is done unconsciously. And so we don't usually appreciate how tough a problem our visual systems solve.

The difficulty of visual pattern recognition becomes apparent if you attempt to write a computer program to recognize digits like those above. What seems easy when we do it ourselves suddenly becomes extremely difficult. Simple intuitions about how we recognize shapes - "a 9 has a loop at the top, and a vertical stroke in the bottom right" - turn out to be not so simple to express algorithmically. When you try to make such rules precise, you quickly get lost in a morass of exceptions and caveats and special cases. It seems hopeless.

Neural networks approach the problem in a different way. The idea is to take a large

Page 53-62 © MANTECH PUBLICATIONS 2019. All Rights Reserved

number of handwritten digits, known as training examples, and then develop a system which can learn from those training examples. In other words, the neural network uses the examples to automatically infer rules for recognizing handwritten digits. Furthermore, by increasing the number of training examples, the network can learn more about handwriting, and so improve its accuracy. So while I've shown just 100 training digits above, perhaps we could build a better handwriting recognizer by using thousands or even millions or billions of training examples.

In this paper we'll write a computer program implementing a neural network that learns to recognize handwritten digits. The program is just 74 lines long, and uses no special neural network libraries. But this short program can recognize digits with an accuracy over 96 percent, without human intervention. Furthermore, in later papers we'll develop ideas which can improve accuracy to over 99 percent. In fact, the best commercial neural networks are now so good that they are used by banks to process cheques, and by post offices to recognize addresses.

We're focusing on handwriting recognition because it's an excellent prototype problem for learning about neural networks in general. As a prototype it hits a sweet spot: it's challenging - it's no small feat to recognize handwritten digits - but it's not so difficult as to require an extremely complicated solution, or tremendous computational power. Furthermore, it's a great way to develop more advanced techniques, such as deep learning. And so throughout the paper we'll return repeatedly to the problem of handwriting recognition. We'll also discuss how these ideas may be applied to other problems in computer vision, and also in speech, natural language processing, and other domains. Of course, if the point of the paper was only to write a computer program to recognize handwritten digits, then the paper would be much shorter! But along the way we'll develop many key ideas about neural networks, including two important types of artificial neuron (the perceptron and the sigmoid neuron), and the standard learning algorithm for neural networks, known as stochastic gradient descent.

Throughout, WE focus on explaining why things are done the way they are, and on building your neural networks intuition. That requires a lengthier discussion than if

Page 53-62 © MANTECH PUBLICATIONS 2019. All Rights Reserved

WE just presented the basic mechanics of what's going on, but it's worth it for the deeper understanding you'll attain. Amongst the payoffs, by the end of the paper we'll be in position to understand what deep learning is, and why it matters.

RESULTS AND CONCLUSION

All 74 errors made by the neural network are shown in the following collage. For each pattern, the sequence number is given, together with an indication of the error made by the network. For example, "4176 2=>7" means that for pattern number 4176, the network misrecognized a 2, and thought it was a 7:

For some of these patterns, it's easy to see why the neural network was wrong: the patterns are genuinely confusing. For example, pattern number 5937, which is purported to be a 5, looks to me like a 3, which is what the neural network thought too. Other patterns appear to have artifacts from pre-processing by the MNIST authors, and it's the presence of these artifacts that confused the network.

For example, pattern number 5457, which is clearly a 1, also has artifacts on the right-hand side, possibly caused when the pattern was separated from an adjacent pattern. Other patterns, primarily in the 8's, have too much missing from the bottom of the patterns.



Figure 1.

Page 53-62 © MANTECH PUBLICATIONS 2019. All Rights Reserved



Figure: 2

But for the vast majority, the neural network simply got it wrong. The patterns aren't especially confusing. Each of the 7's looks like a 7, and there's no good reason why the neural network failed to recognize them as 7's.

Perhaps more training, with a larger training set, would help. I'm inclined to believe, however, that the network is at a point of diminishing returns with respect to training. In other words, more training might help some, but probably not much. Instead, WE believe that a different architecture is needed. For example, WE

Page 53-62 © MANTECH PUBLICATIONS 2019. All Rights Reserved

think that the input layer should be designed with due recognition for the fact that the human visual system is very sensitive to spatial frequency. As a result, WE believe that better performance could be obtained by somehow converting the input pattern to a frequency domain, which could be used to supplement the purely spatial input now being given as input.

REFERENCES

1. Patrice Y. Simard, Dave Steinkraus, John Platt, "Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis," International Conference on Document Analysis and Recognition (ICDAR), IEEE Computer Society, Los Alamitos, pp. 958-962, 2003.

2. Fabien Lauer, Ching Y. Suen and Gerard Bloch, "A Trainable Feature Extractor for Handwritten Digit Recognition", Elsevier Science, February 2006. Pp. 341-353

3. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-Based Learning Applied to Document Recognition,"Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998. [46 pages]

4. Y. LeCun, L. Bottou, G. Orr, and K. Muller, "Efficient BackProp," in Neural Networks: Tricks of the trade, (G. Orr and Muller K., eds.), 1998. [44 pages]

5. Handwritten Digit Recognition using Convolutional Neural Networks in Python with Keras, by Jason Brownlee. Pp 67- 83

6. Neural Network for Recognition of Handwritten Digits, by Mike O'Neill. Pp 134-165.

Page 53-62 © MANTECH PUBLICATIONS 2019. All Rights Reserved