Handwritten Digit Recognition using Shallow ANN

Ethan Shealey
High Point University Department of Computer Science
High Point University
High Point, North Carolina
eshealey@highpoint.edu

***Abstract*—This work attempts to showcase the use of ANN neural networks in identifying hand written digits.**

I. Introduction

Artificial Neural Networks (ANNs) are a breakthrough in modern computing. ANNs are inspired by the early models of sensory processing by the brain. An artificial neural network can be created by simulating a network of model neurons in a computer. By applying algorithms that mimic the processes of real neurons, we can make the network 'learn' to solve many types of problems. [1] ANNs are trained using large data sets in order to properly teach the network what is what. In this experiment we are using a collection of 10,000 28x28 pixel images to attempt to train the ANN to recognize the handwritten digits.

II. Using MATLAB's patternnet to train and test an Artificial Neural Network

In MATLAB, we are provided with built in in functions to help us create, train, and test ANNs. The function we will focus on in this experiment is `net = patternnet(hiddenSizes, trainFcn)` where `hiddenSizes` are the sizes of the hidden layers in the network, specified as a row vector, like `[100 100]` for two hidden layers of the size 100.

`trainFcn` is the training function to be used in the neural network, set to Scaled Conjugate Gradient (trainscg) as default. The Scaled Conjugate Gradient algorithm is based on conjugate directions, but this algorithm does not perform a line search at each iteration unlike other conjugate gradient algorithms which require a line search at each iteration. Making the system computationally expensive. [3] Scaled Conjugate Gradient algorithm was designed to avoid the time-consuming line search. [3] In MATLAB we can use this to train any network as long as its weight, net input, and transfer functions have derivative functions. In Scaled Conjugate Gradient algorithm, the step size is a function of quadratic approximation of the error function which makes it more robust and independent of user defined parameters. The step size is estimating using different approach. The second order term is calculated as

$$\bar{s_k} = \frac{E'(\bar{w_k} + \sigma_k \bar{p_k}) - E'(\bar{w_k})}{\sigma_k} + \lambda_k \bar{p_k} \tag{1}$$

where λ_k is a scaler and is adjusted each time according to the sign on δ_k.
The step size

$$\alpha_k = \frac{\mu_k}{\delta_k} = \frac{-\bar{p_j}^T E'_q w(\bar{y}_1)}{\bar{p_j}^T E"(\bar{w})\bar{p_j}} \tag{2}$$

where $\bar{w}$ is weight vector in space R^n,
$E(\bar{w})$ is the global error function,
$E'(\bar{w})$ is the gradient of error,
$E'_{qw}(\bar{y_1})$ is the quadratic approximation of error function,
$\bar{p_1}$, $\bar{p_2}...\bar{p_k}$ be the set of non-zero weight vectors. [3]
λ_k is to be updated such that

$$\bar{\lambda_k} = 2(\lambda_k - \frac{\delta_k}{|\bar{p_k}|^2}) \tag{3}$$

If $\Delta_k > .75$, then $\lambda_k = \lambda_k/4$
If $\Delta_k < .25$, then $\lambda_k = \lambda_k + \dfrac{\delta_k(1-\Delta_k)}{|\bar{p_k}|^2}$
Where, Δ_k is comparison parameter and is given by

$$\Delta_k = 2\delta_k[E(\bar{w_k}) - E(\bar{w_k} + \alpha_k \bar{p_k})]/\mu_k^2 \tag{4}$$

Initially the values are set as, $0 < \sigma \leq 10^{-4}, 0 < \lambda_l \leq 10^{-6}$ and $\lambda_l = 0$. [3]
Training stops when any of the following occurs: [3]

- The maximum number of epochs is reached
- The maximum amount of time is exceeded
- Performance is minimized to the goal
- The performance gradient falls below min-grad
- validation performance has increased more than max-fail times since the last time it decreased (when using validation)

There are a number of training algorithms you can pick from when using the `patternnet` function, like Levenberg-Marquardt Algorithm, Bayesian Regularization Algorithm, BFGS Quasi-Newton Algorithm, Resilient Backpropagation Algorithm, Conjugate Gradient with Powell/Beale Restarts Algorithm, Fletcher-Powell Conjugate Gradient Algorithm, Polak-Ribiére Conjugate Gradient Algorithm, One Step Secant Algorithm, Variable Learning Rate Gradient Descent Algorithm, Gradient Descent Algorithm, and the Gradient Descent With Momentum Algorithm. All these various algorithms present a different way to train our neural networks in different ways, resulting in various outcomes. `net`, the output of the `patternnet` function, is the pattern recognition neural network object. We can then train this newly developed network to be able to reliably recognize our set of handwritten values.

III. Implementation

Implementing this experiment in MATLAB was our best course of action thanks to its built in functions to easily build, train, and test the neural network. To begin, we need our data sets. For this experiment we are going to be training and testing with the MNIST database of handwritten digits. Our training set is made up of 60,000 28x28 pixel images, and our testing set is 10,000 images of the same size. The first step in this process is loading in the data itself by using custom made functions `loadMNISTImages` and `loadMNISTLabels`. Once loaded we need to reshape our data from row wise to column wise so we can use it in the `patternnet` function. Once reshaped we can then create a matrix of size ten by the training label column length to indicate which label each of the training images belong to. Once these variables are prepared we can begin creating and training the ANN. To create the ANN we will use the aforementioned `patternnet` function as shown below.

$$net = patternnet([100\ 100]) \tag{5}$$

This line will initiate our variable `net` to be out neural network object with two hidden layers of size 100. Next we need to train the neural network to be able to recognize the input, we can do that with the segment below.

$$net = train(net, training\_set, tt) \tag{6}$$

`net` has now been reassigned to a trained neural network. In the built in function `train` we pass in `net`, the untrained neural network, `training_set`, the data set of 60,000 28x28 images to train the network with, and `tt`, the ten by the training label size matrix, filled with zeros other then the indices for each image marking the correct label.

We now have a fully trained ANN. With this newly trained neural network we can test it using a data set of our choice, and once tested we can then calculate the accuracy of our network. To test our new network we need to do the following.

$$output = net(testing\_set) \tag{7}$$

`output` is a matrix acting as the results of testing the ANN, filled with the percent that each image belongs to each label. `testing_set` is the 10,000 28x28 images set aside to test the network we loaded in earlier.

With the results now stored in *output*, we need to figure out which label the neural network chose for each of the images we just tested. Doing this is simple. We just need to get the column size of *output*, and initialize an matrix of zeros of size 1 by column size of *output*. With this we can iterate through each column of *output* and get the index of the maximum value, resulting in the label the neural net decided was most likely the input.

Once we create this matrix of guesses, we can find the accuracy of the neural net by comparing the results to the correct labels like so

$$accuracy = \frac{sum(guess == testing\_labels)}{numel(testing\_labels)} \tag{8}$$

Where `accuracy` now holds the determined accuracy in decimal format, `guess` is the new matrix of the nerual networks guesses, and `testing_labels` is the loaded set of labels for the 10,000 test images. With this now calculated we can simply return the value and view how accurate our ANN is.

IV. Experiment Results and Discussion

On running this MATLAB experiment with the same 60,000 28x28 pixel training images and the 10,000 28x28 pixel testing images, our ANN was able to produce a 96.85% accuracy rate. These results were extremely satisfactory and proved that our ANN worked as intended. The applications of this concept are endless and can vastly improve the human experience. This ideas can be applied to amazing real world uses like self driving cars, where an artificial neural network could learn and produce accurate results in real time. Self driving cars are the future as seen with Tesla's autopilot vehicles are proving. The issue with things like this is that they need to be extremely precise, much more precise then our 96.85%. No one would want a 3.15% chance their car does not know what a wall is and drive directly into it, as that would be devastating and possibly life ending. Luckily, large companies like Tesla have access to unimaginably large sets of test data to train their artificial networks on, resulting in higher probability of guessing correctly.

Meanwhile, in the medical field, the use of Artificial Neural Networks can help identify diseases like breast cancer, hepatocellular carcinoma, and other malignancies in humans. [4] ANNs are often incorporated into machinery such as magnetic resonance imaging systems which are useful for studying segments of the brain. [4] One way in which ANNs are used in neurology was demonstrated by Andre Gabor and Masud Seyal as they conducted a study that showed the effectiveness of using artificial neural networks in the identification of electroencephalographic (EEG) patterns. [4] By recording epileptiform transients in individuals, they measured two different levels of certainty of recognition to detect spikes and sharp waves. [4] These spikes and waves are essential in recording EEG activities. [4] The results of this study showed that using ANNs is an effective way of recording brain activity with precision and accuracy. [4] ANNs may be used in radiology to identify different brain structures. Vincent Magnotta and peers used ANNs to identify various structures of the brain. They described in their article "Measurement of Brain Structures with Artificial Neural Networks: Two- and Three-dimensional Applications" the potential of artificial neural networks to identify those structures when applied to magnetic resonance images. They found that ANNs are able to identify brain structures as accurately as human technicians could, however quicker. [4] These applications in the medical field span much more then discussed in this paper, but the work has been life changing and will hopefully change the way we view health and medicine as this technology becomes better and better.

Another amazing use case of a neural network like the one we created in this experiment is their use in predicting the stock market. Ramon Lawrence at the University of Manitoba conducted research on this topic and he believes the ability of neural networks to discover nonlinear relationships in input data makes them ideal for modeling nonlinear dynamic systems such as the stock market. [5] The uses of this kind of technology is truly endless and could pave the way for ground breaking research and discoveries in the coming decades. I for one am extremely excited to see where this goes and will hopefully be able to reap the benefits of such discoveries.

V. Conclusion

In conclusion, our expirement to find if Artificial Neural Networks could successfully identify handwritten values based on a set of training images and labels was a success. With a 96.85% recognition rate I believe we have sufficiently been able to identify the test images.

References

[1] Krogh, A. (2008). What are artificial neural networks? Nature Biotechnology, 26(2), 195–197. doi:10.1038/nbt1386

[2] Chowdhury, A. M. S., & Rahman, M. S. (2016). Towards optimal shallow ANN for recognizing isolated handwritten Bengali numerals. 2016 9th International Conference on Electrical and Computer Engineering (ICECE). doi:10.1109/icece.2016.7853889

[3] Lochan Babani, Sadhana Jadhav, Bhalchandra Chaudhari. Scaled Conjugate Gradient Based Adap- tive ANN Control for SVM-DTC Induction Motor Drive. 12th IFIP International Conference on Artificial Intelligence Applications and Innovations (AIAI), Sep 2016, Thessaloniki, Greece. pp.384- 395, 10.1007/978-3-319-44944-9_33. hal-01557626

[4] Haglin, J.M., Jimenez, G. Eltorai, A.E.M. Artificial neural networks in medicine. Health Technol. 9, 1–6 (2019). https://doi.org/10.1007/s12553-018-0244-4

[5] Lawrence, Ramon. (1998). Using neural networks to forecast stock market prices.