Leonardo Ferrisi
CSC320 - Artificial Intelligence
Final Project

Classifying Handwritten Digits from the MNIST Dataset using an Artificial Neural Network

Abstract:

Artificial Neural Networks are designed to function based on concepts in learning inspired by what we know about how neurons in the human brain learn. Whereas a regular neuron communicates using ion gradients and a combination of charge and neurotransmitters between neurons, artificial neurons, better known as perceptrons take an input and apply it to an activation function defining what gets output.

Depending on the correctness of the output value as determined by simply propagating forward, the value is back propagated through and weights from the activation function to the output are altered until the value output matches the target.

Some (most) problems however are far too complex for a single neuron (perceptron) to solve, so by wiring them up (and a ton of tweaking) complex problems can become solvable.

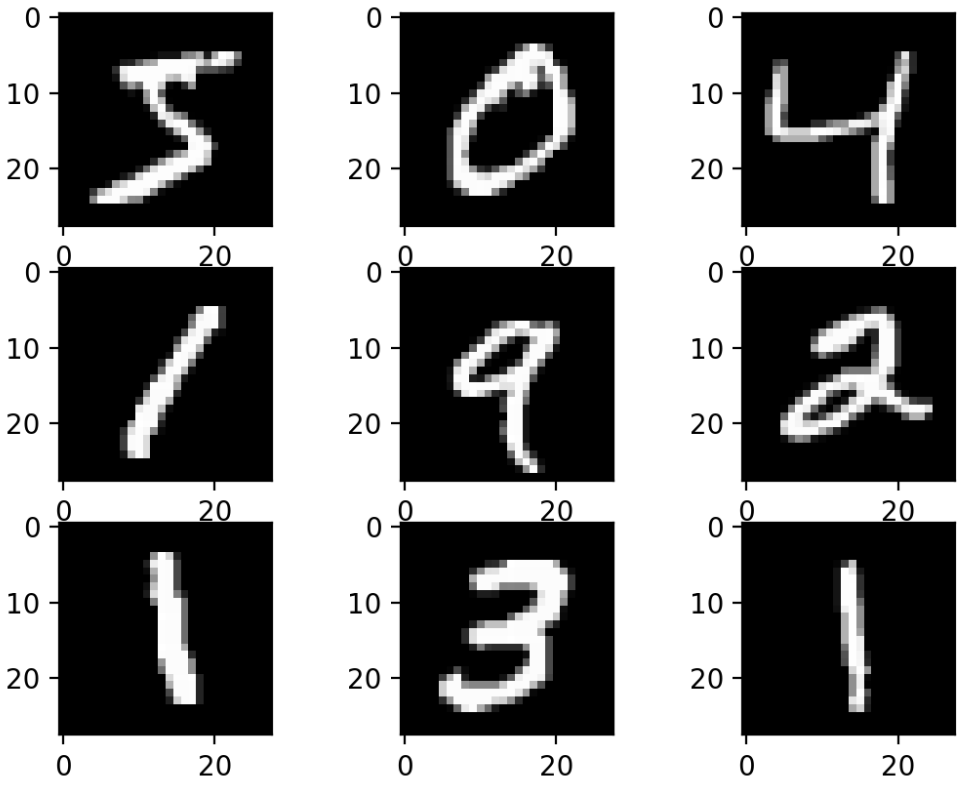
There are several variations on the Artificial Neural Network, which typically involves a small number of layers, an input, hidden and output layer. One such variation, the Deep Neural Network for example, involves a great many hidden layers allowing for more neurons and weights to adjust to targets.

This final project will involve attempting to classify MNIST data using an Artificial Neural Network. The ANN in question will be directly adapted from the code written to solve project-6 for this course using ANNs to solve the XOR problem and Auto Association.

Introduction:

Handwriting classification is one of the most popular starting tasks for applied machine learning, more specifically applications for Neural Networks. Usually, researchers use either Deep or Convolutional Neural Networks on images, which allow for the usage of more complex images with more distinct features being considered.

One of the projects covered for the Spring 2022 CSC320 course was applying Artificial Neural Networks to some basic functions such as Auto Association and the XOR problem. These Artificial Neural Networks differed from Convolutional Neural Networks (CNNs) or Deep Neural Networks (DNNs) in their simplicity. Whereas a CNN processes images and other data by running a kernel over the data to alter the data into a digestible format such as applying a gaussian blur or normalizing the brightness of an image, or a deep neural network uses a large quantity of hidden layers to tackle all kinds of problems, an ANN at least as the initial assignment specified involves a much smaller (countable) amount of neurons without using any convolution. [1], [2]



The first 9 images in the MNIST dataset as loaded from the **keras** package

The MNIST dataset is a massive dataset of labeled images of handwritten digits. I am using the mnist dataset accessible through the keras package. Each image is 28 by 28 pixels in size and can be represented with arrays of values between 0 - 255. I'm going to make an artificial neural network that (attempts) to classify with some degree of accuracy handwritten digits from the MNIST dataset.

Methods:

The ANN being used for this is a modified version of the project-6 three layer ANN. This ANN has 4 layers, 1 input, 2 hidden, and 1 output. While it doesn't appreciate XOR, it works quickly on the Auto Association finishing training in less than 20 seconds.

Getting this ANN to recognize MNIST data is not going to be simple - we can't just feed the image into the network, it needs to be processed and encoded. Whereas convolution applies a kernel over the dataset to simplify the data while keeping some distinction with features, instead my ANN breaks up the 28 x 28 arrays representing each handwritten digit into sixteen 28/n x 28/n dimensional arrays. These subarrays then have all of their values normalized as initially all the numbers are represented with values between 0 - 255, we want these to become values between 0 - 1, so we simply divide every item by 255. [3] After the subarray has been normalized we then have an average calculated for each sub-array and store it in an array of averages resulting in $28/n^2$ average values or *N* average values being generated representing each of the handwritten digits. These N item long arrays are fed in as the inputs to N input neurons with 10 output neurons giving a value corresponding to the number it thinks the data represents.

Results:

Testing demonstrated that having only 16 sub-sets (28/7 x 28/7 where n = 7 and N = 16) is not enough to accurately distinguish MNIST images from each other. That being considered, I moved for the next best thing, a 7 x 7 matrix of 4 x 4 subsets for a total of 49 subsets being passed into 49 input neurons (n=4, N = 49). Before being loaded in, every image was converted into a set of 49 averages from the normalized data from each of the 49 subsets generated by the **Encoder** class written for this assignment.

The process of encoding to reiterate, involves:

- breaking an image of 28 x 28 pixels into an array of smaller subsets of pixels
- Each subset is normalized by dividing every pixel by 255 such that all values are between 0 and 1
- An average is then calculated for each subset and stored in an array with a length of the quantity of subsets
- The array of 49 averages is then passed in as the inputs for the input layer.

For the outputs an array such as:

[0, 0, 0, 0, 1, 0, 0, 0, 0, 0]

Would represent the number 5, given the 1 is in the 5th spot or the forth index. A zero would be an empty array of just zeros.
Since the handwritten digits are 0 - 9, these are the only kinds of values the ANN will be concerned with classifying.

For a multilayer ANN, in this case one with 2 hidden layers - the network was able to solve basic problems like auto association in less than 5000 epochs in under 20 seconds (4591 epochs) with a learning rate of 0.4 and a tolerance of 0.01 for squared error. In practice what works best is a hidden layer of the same size or greater size than the output layer. I went with 28 artificial neurons for each hidden layer which was a little over half of the input layer of 49 neurons.

In practice, because this ANN did not implement the full extent of numpy for representing its data, even though the MNIST dataset contains well over 60,000 images of training and test data - I opted to use a smaller subset of the data so as to save time.

I started by attempting to train the network on a set of 100 images but this proved too difficult for the neural network and in addition still ran incredibly slowly. I began starting up for 10 images to see what this current Neural Network's limits were.

Amount of images	Epochs Taken	Time Taken (s)	Tolerance	TSS Error
10	848	49.54	0.3	0.2932
20	855	117.55	0.3	0.2621
20	844	124.61	0.2	0.1733
20	994	150.96	0.1	0.09432
30	2170	416.41	0.1	0.0975
40	882	252.33	0.1	0.0728
50	479	168.05	0.1	0.0964

With 49 input neurons, 28 hidden1, 28 hidden2, and 10 output neurons. Learning Rate=0.6

As shown above the Artificial Neural Network successfully was able to classify up to 50 images of MNIST data at a time in a reasonable timeframe

The neural net is capable of classifying as of now from a set of 10 images in a dataset with a tolerance of 0.2 and a learning rate of 0.6. I stopped at 50 as attempts with 60 took upwards of 10 minutes at a learning rate of 0.6 and hit a local minimum of 24/60 and stayed there for a few thousand epochs before I opted to end the training and simply demonstrate what this network *was* able to do in a timely manner.

Discussion:

With a higher learning rate, the Artificial Neural Network was observed to spend the majority of the time with a low correctness until at some point the right weights finally adjust causing several values to be correctly classified which then quickly would lead the neural network to find the solution which classifies the entire set of images with a TSS Error below the tolerance.

Since I was unsure whether or not it would even be possible to classify MNIST using only an ANN, I kept the tolerance at 0.1.

In practice, this ANN worked for small amounts of MNIST data but once those sets became too large, the time alone for training took way too long to actually allow time to properly evaluate larger datasets. It is also possible that splitting each image into 49 smaller squares and then averaging their values is not enough to properly distinguish all handwritten digits from each other. Typically, MNIST classifiers use convolution, or instead use a deep neural network with one input neuron per pixel in an image (28 x 28 results in 784 input neurons for such a Neural Network!)

Packages such as Keras are incredibly optimized for creating Neural Networks so can process large amounts of data efficiently.

My Artificial Neural Net, though functional enough to demonstrate the concept of classifying handwritten digits off of images, is still incredibly inefficient. Some future work for this project would be to implement everything representing the data moved around with Numpy classes such as arrays and matrices. From there, it might actually be possible to attempt expanding this ANN to have 784 input neurons to wire to each pixel. Such an action would remove any need for an encoder class, though preferable all data would be normalized.

Conclusion:

All in all, this project demonstrated that even a simple and inefficient Neural Network is capable of classifying MNIST data without need for Convolution or incredibly deep neural nets needing to be implemented. With the current arrangement, the best classified 50 images with a tolerance of 0.1 and a learning rate of 0.6 in less than 3 minutes and in only 479 epochs.

ScoreSheet:

- **Scope / Challenge: (10/10)**
  - When I started working through this problem, it occurred to me that it might not be possible with a Neural Network this simple. A considerable amount of time was spent simply ensuring that the multilayer ANN was working as desired and furthermore, that all the values were propagating and backpropagating properly.
  - Timing was an issue as well, with massive datasets, printing anything out adds a significant amount of time to the overall runtime so that needed to be done away with
- **Completion: (9/10)**
  - The end goal would have been to be able to train this ANN on all 60,000 samples of the MNIST training dataset. Unfortunately for this current multilayer ANN, the amount of time that would have taken would have been far too long and the model would still be training by the time the term ends. Instead I opted in to test this ANN on smaller subsets of the larger MNIST dataset.
  - Because of the current existing inefficiencies in this ANN, as more data is passed in as an input, the time taken to train increases. In addition, it is possible that the 49 averaged normalized values for each image are not sufficient to properly classify any handwritten digit.
  - The ANN as it currently is could not classify in a sufficient amount of time datasets of or greater than 60 MNIST images.
- **Implementation: (9/10)**
  - Implementing the ANN as written for project-6 and adding on additional layers was conceptually an easy feat but required quite a bit of tweaking and debugging to get right. Luckily when I did project-6 I anticipated I might be adding extra layers and I already had some functionality written in to iterate through additional layers in the network for forward and backward propagation.
  - The ANN would have been substantially more efficient had I rewritten everything using large arrays using numpy subclasses like array or matrix. Numpy has some under the hood logic for optimizing massive datasets and this may have drastically improved runtimes.
  - The implementation of an Encoder class was helpful for an ANN suffering from efficiency constraints, however the best way for doing this with an ANN likely would have been to wire up a neuron to each pixel in the 28 x 28 MNIST images such that a total of 784 input neurons would exist. This may have resulted in issues for overfitting, or it could have enabled the model to work much quicker and more smoothly / efficiently
- **Assessment/Analysis (10/10):**
  - I measured how well this ANN performed by timing each run and taking into account the amount of epochs taken, the tolerances used and the learning rates as well as quantity of input neurons.
  - This display of data shows the performance of the network on datasets of several different (small) sizes.

- **Utility: (8/10)**
    - Implementing a multilayer ANN would not be a terribly difficult thing to have as a project in later AI classes. Although Keras is the best way of loading the MNIST dataset, it would give students an excuse to at least try using some aspect of Keras even if it is just to load data. In addition, using something such as the Encoder class is an excellent opportunity to segway into why convolution is helpful and how it works. The Encoder does not perform convolution, but it does simplify the dataset in an attempt to make it more digestible for a neural network where we care about whether or not it takes all day to run. By using an encoder to divide MNIST images into chunks and then averaging the values in each chunk to be used as a means of classifying each image is an excellent way to introduce how this can help, and then with Convolution, explaining how it can be done better and how the industry currently tackles the problem.

[1] D. C. Ciresan, U. Meier, L. M. Gambardella and J. Schmidhuber, "Convolutional Neural Network Committees for Handwritten Character Classification," *2011 International Conference on Document Analysis and Recognition*, 2011, pp. 1135-1139, doi: 10.1109/ICDAR.2011.229.

[2] Ahlawat, Savita, et al. "Improved handwritten digit recognition using convolutional neural networks (CNN)." *Sensors* 20.12 (2020): 3344

[3] Saravanan, Abishek. “MNIST Handwritten Digits Recognition with Keras.” *Medium*, Medium, 30 Dec. 2020, https://abishuriya932.medium.com/mnist-handwritten-digits-recognition-with-keras-1a4e5712b508.
.