The Nintendo Artificial Neural Network System*

Carmine Guida[1], Lauren Demaio[1]

[1] *Pace University, One Pace Plaza, New York, NY 10038*

Abstract

Artificial Neural Networks (ANN) are used in a variety of machine learning tasks such as image classification and pattern recognition. Training ANN models on large datasets can be time consuming and require specialized hardware and processing power. Often, the training is performed on powerful systems, and the resultant final trained network can be utilized on small computers and mobile devices to perform tasks instantly. The Nintendo Entertainment System (NES) was released in 1985 and featured a 1.79 MHz CPU and 2 KB of RAM. The original program code for games for the NES fit within 32 KB of ROM on cartridges. In this work, we create an ANN and train it using the EMNIST hand-drawn digit dataset on a desktop computer. This ANN is then ported to the NES utilizing assembly language. The program code and pretrained weights are stored to a game cartridge fitting within the bounds of 32 KB. Additionally, a user interface is provided for drawing and loading test samples and executing the ANN classification of hand drawn digits on a physical NES device from 40 years ago.

Keywords

artificial neural network, machine learning, image processing, retro hardware

1. Introduction

Artificial Neural Networks (ANN) are a part of Machine Learning (ML) and are used for several different kinds of tasks such as classification and segmentation [1] as well as prediction [2]. The applications of ANN are numerous and are often used to solve problems in a wide range of fields including science, finance, engineering, agriculture, education, and energy [3].

A simple architecture for an ANN may involve three layers known as the input, hidden, and output layers [4], though more advanced ANNs like those in Deep Learning (DL) will have several different layers [5]. These layers contain nodes with connections from previous layers, a bias and weights. Data will feed forward through the network producing an output. The learning process involves updating these weights. One of the most common algorithms for this is known as backpropagation [6].

ANNs typically use the IEEE-754 single or double-precision floating-point format [7]. To perform the math required, devices must have a floating-point unit (FPU) along with the usual CPU. Originally used for 2D and 3D graphics acceleration for games, Graphics Processing Units (GPU) are now commonly used for machine learning [8]. In 2015 Google introduced the Tensor Processing Unit (TPU) to increase performance for neural networks and use less power than GPUs [9].

The Nintendo Entertainment System (NES) was originally released in North America in 1985 [10]. The NES featured a Ricoh 2A03 chip that was based on the 8-bit 6502 processor operating at 1.79 MHz [11]. The NES also had only 2 KB of RAM available. The program (game) code is stored within ROM chips on the game cartridges. The original Super Mario Bros. fit within an original restriction of 32 KB of program ROM space [12] before the later creation of memory mapper chips.

In this experimental work, we demonstrate that an ANN is capable of running even with severe hardware constraints. We created a simple ANN in the C programming language and trained it on a dataset of small images on a modern computer. The program code for the ANN is ported to 6502 assembly language and includes the pretrained weights from the ANN model. By reducing dimensionality of the images and using a small number of hidden nodes we were able to fit within

* *Joint AIIDE Workshop on Experimental Artificial Intelligence in Games and Intelligent Narrative Technologies, November 10-11, 2025, Edmonton, Canada.*

[1] cguida@pace.edu (C. Guida); ldemaio@pace.edu (L. DeMaio)

the 32 KB limit. The program code is placed onto a specialized game cartridge and loaded into the physical NES hardware. We also developed a user interface (UI) which allows the user to draw images as well as load sample dataset images using the NES gamepad. Pressing START executes the ANN and displays its classification of the image.

2. Materials and Methods

2.1. Dataset

The MNIST dataset [13] consists of hand drawn digit (0 – 9) images and is often referred to as the "hello world" of neural networks and deep learning [14]. The dimensions of the images are 28x28 and contain grayscale values ranging from 0 - 255. Examples of the dataset images can be seen in Figure 1. The training set contains 60,000 images with the testing set having 10,000. These sets are balanced with 6,000 of each digit in the training set and 1,000 of each digit in the testing set. Given we are pretraining the neural network on a modern computer, we opted to use the larger EMNIST dataset [15] which is an extension of MNIST containing four times the number of images of MNIST. This dataset has 240,000 images in the training set and 40,000 in the testing set. These sets are also balanced.

Figure 1: Example digit images.

2.2. Preparing the Data

We took multiple steps to reduce the dimensionality of the images to work within memory constraints. First, we changed pixel values from a range of 0 - 255 to be just binary values of 0 or 1. Any pixel value less than 128 was changed to 0 with 128 or greater becoming a 1. Figure 2 shows an image with grayscale values and then changed to binary values. Reducing the values to bits allowed for 8 pixels to be packed into a single byte. This reduced storage requirements by almost ⅛ for our input weight data and loadable test samples.

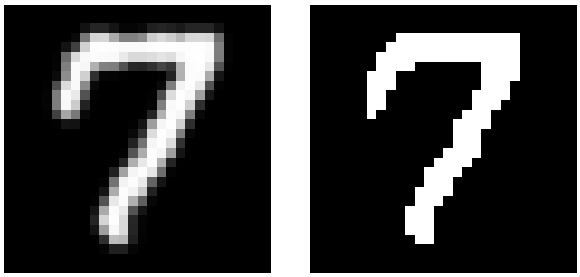

Figure 2: Left: Original 256 grayscale image. Right: 1-bit binary image.

Our goal was to reduce the image resolution to 20×20 with minimal scaling required. Given that the images in the dataset are centered, columns of pixels on the left and right, as well as rows on the top and bottom of the image, can be discarded as they contain no information about the digit. This can be seen in Figure 3. We discarded rows and columns with 2 or fewer pixels of a value of 1.

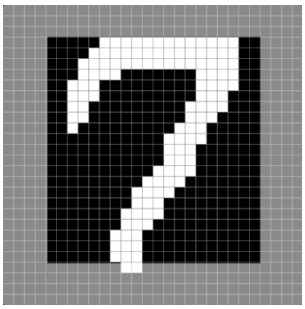

Figure 3: Pixel grid indicating candidate rows/columns to be removed.

2.3. Neural Network

Figure 4 shows the configuration of our ANN. The input layer contains the pixel information of the current image. With our input images having dimensions 20×20, there will be 400 input nodes. As in most neural networks, every input node is connected to every hidden node [16]. Our model uses 16 hidden nodes. Here, we introduce an optimization. Given our pixel input values are either 0 or 1, and these values are multiplied by the weights, input values of 0 can simply be skipped. Additionally, with an input value of 1, we can skip the multiplication and just add the weight to the node's total. Every hidden node connects to every output node. There are 10 output nodes, one for each digit (0 – 9). All of our nodes also have a bias. Originally, the Sigmoid activation function [17] was popular for neural networks however we opted to use the Leaky ReLU function [18] as it provided better performance.

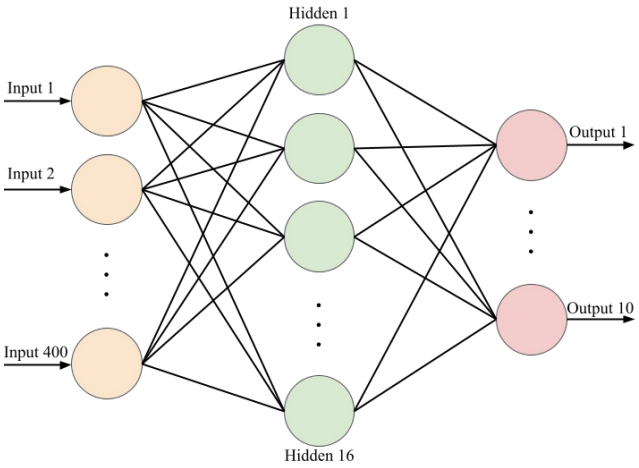

Figure 4: Diagram of our ANN.

With 20×20 input nodes each connected to 16 hidden nodes at 4-bytes per floating-point weight, this would use (20×20×16×4) 25,600 bytes of space. Additionally, the 16 hidden nodes connected to the 10 output nodes is (16×10×4) 640 bytes. Each of the 16 hidden nodes and 10 nodes have a bias input for (16×4 + 10×4) 104 bytes. The stored pretrained weights occupy 26,344 of our total 32,768 program ROM space leaving 6,424 bytes for our code that runs the user interface and executes the ANN to perform the classification.

2.4. Hardware and Tools

While modern machine learning research is typically developed in the Python programming language using libraries such as PyTorch, Keras, and TensorFlow [19], our ANN is coded from scratch in the C programming language without any of these types of libraries. This allowed for a more direct port to the 6502 assembly language required for developing for the NES. After training,

the weights were exported to a format that could be included by a 6502 assembly compiler. Given the NES is an 8-bit processor with no floating point support, in order to port our neural network, we used a floating point library developed by Roy Rankin and Steve Wozniak [20]. To run our program on a physical NES device, we loaded our program onto an SD card and placed it into a Krikzz Everdrive N8 Pro cartridge which can be seen in the left of Figure 5. This cartridge was then placed into an original NES from 1985 seen on the right in Figure 5.

Figure 5: Left: Everdrive N8 Pro cartridge. Right: Original 1985 NES.

3. Experiments and Discussion

We trained our ANN on the EMNIST dataset on a desktop computer. We used a learning rate of 0.01, an alpha value of 0.01 for LeakyReLU and 5 epochs. The training completed in 21 seconds. Our simple ANN achieved a 93.11% accuracy on the EMNIST test set. This number could be higher with a larger neural network however for this experiment we needed to stay within the 32 KB limit. The pretrained weights along with our program code for the UI and executing the ANN were loaded onto the cartridge and NES seen in Figure 5. Our program begins with a basic startup/menu screen seen in the left of Figure 6. Pressing START enters the main drawing screen seen on the right in Figure 6.

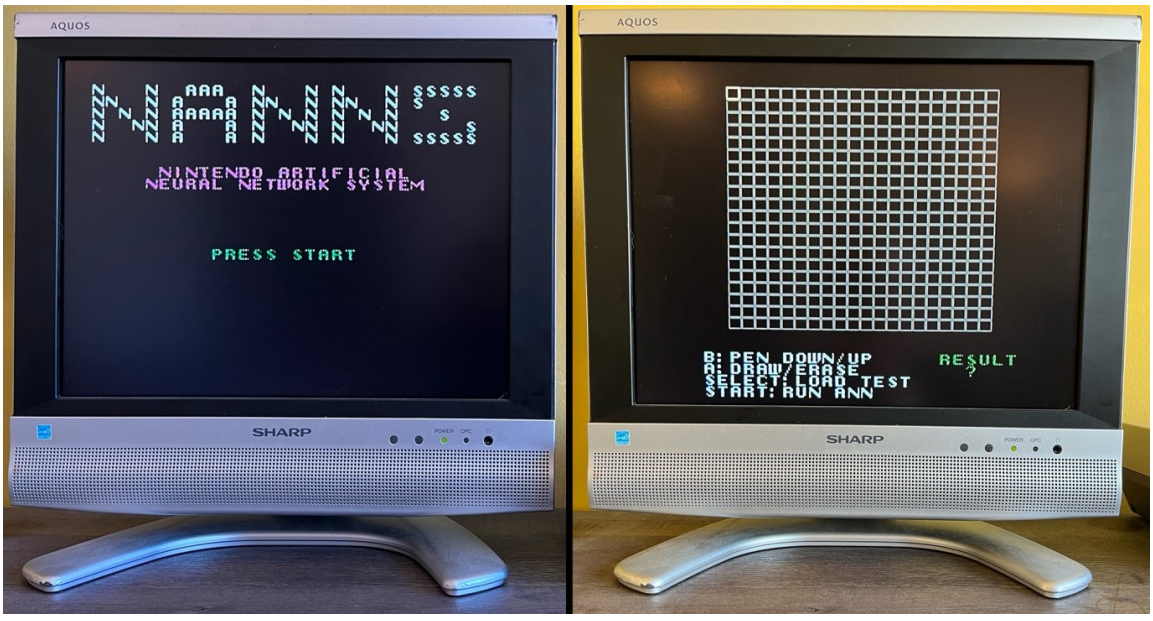

Figure 6: Left: Startup screen. Right: Main drawing screen.

The drawing screen provides a 20×20 pixel grid along with instructions. Using the D-PAD will move the cursor around the grid. Pressing the B button will toggle the pen (cursor) being down (drawing) or up (movement only). Pressing the A button will toggle drawing or erasing mode.

Along with drawing, the user can press the SELECT button to load 10 images of digits (0 – 9) from the EMNIST test set. Pressing the START button will execute the ANN and show the classification result in under 3 seconds. Figure 7 shows various test images that were loaded and the results after running the ANN.

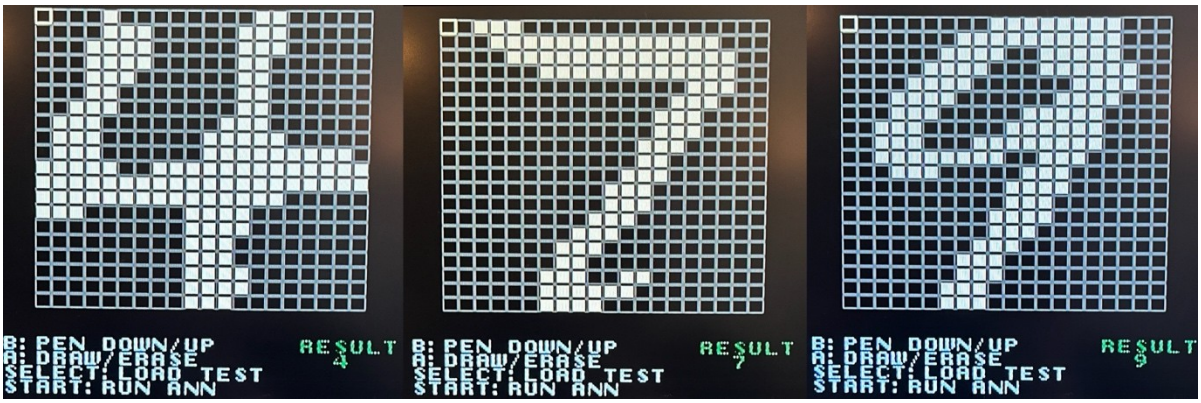

Figure 7: Loaded testing samples from the EMNIST dataset and result from the ANN on the NES.

Even though the ANN was trained on hand drawn style digits, our on-device ANN is also capable of recognizing simpler pixel art style images. Figure 8 shows example digits drawn by ourselves which the ANN was able to classify.

Figure 8: Simple pixel images drawn using our UI and the result from the ANN.

4. Conclusion

In this experimental work, we created a basic ANN image classifier which was trained on a modern computer and then exported the weights from the various layers. We greatly reduced the dimensionality of the input images and used a small set of hidden nodes in order to fit within a 32 KB limit. The ANN was ported to assembly language and using the UI, a user can draw or load testing images and execute the ANN. This demonstrates the 1985 NES is capable of using a basic ANN with pretrained weights to successfully classify digit images.

Future work includes using memory mapper (bank swapping) chips to enable more ROM and therefore larger models to be stored. Additional future work includes building a Reinforcement Learning model that could be used for game AI on the NES.

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] Reddick, W. E., Glass, J. O., Cook, E. N., Elkin, T. D., & Deaton, R. J. (1997). Automated segmentation and classification of multispectral magnetic resonance images of brain using artificial neural networks. *IEEE Transactions on medical imaging*, *16*(6), 911-918.

[2] Grossi, E., & Buscema, M. (2007). Introduction to artificial neural networks. *European journal of gastroenterology & hepatology*, *19*(12), 1046-1054.

[3] Abiodun, O. I., Jantan, A., Omolara, A. E., Dada, K. V., Mohamed, N. A., & Arshad, H. (2018). State-of-the-art in artificial neural network applications: A survey. *Heliyon*, *4*(11).

[4] Abraham, A. (2005). Artificial neural networks. *Handbook of measuring system design*.

[5] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, *521*(7553), 436-444.

[6] Rojas, R., & Rojas, R. (1996). The backpropagation algorithm. *Neural networks: a systematic introduction*, 149-182.

[7] Popescu, V., Nassar, M., Wang, X., Tumer, E., & Webb, T. (2018, June). Flexpoint: Predictive numerics for deep learning. In *2018 IEEE 25th Symposium on Computer Arithmetic (ARITH)* (pp. 1-4). IEEE.

[8] Aamodt, T. M., Fung, W. W. L., Rogers, T. G., & Martonosi, M. (2018). *General-purpose graphics processor architectures*. Morgan & Claypool Publishers.

[9] Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., ... & Yoon, D. H. (2017, June). In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture* (pp. 1-12).

[10] Kohler, C. (2010). Oct. 18, 1985: Nintendo Entertainment System launches. https://www.wired.com/2010/10/1018nintendo-nes-launches

[11] Diskin, P. (2004). Nintendo Entertainment System Documentation. *Tokyo: Nin-tendo*.

[12] Cook, K. M. (2023). 8-Bit Affordances: Jun Chikuma's Soundtrack to Faxanadu. *Music Theory Online*, *29*(3).

[13] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278-2324.

[14] Teh, V., Loong, J. C. Y., Wai, E. D. H., Yang, L. J., Cheng, C. J., & bin Abdul Salam, Z. A. (2024). Enhancing neural network models for mnist digit recognition. *Journal of Applied Technology and Innovation (e-ISSN: 2600-7304)*, *8*(1), 15.

[15] Cohen, G., Afshar, S., Tapson, J., & Van Schaik, A. (2017, May). EMNIST: Extending MNIST to handwritten letters. In *2017 international joint conference on neural networks (IJCNN)* (pp. 2921-2926). IEEE.

[16] Maind, S. B., & Wankar, P. (2014). Research paper on basic of artificial neural network. *International Journal on Recent and Innovation Trends in Computing and Communication*, *2*(1), 96-100.

[17] Basterretxea, K., Tarela, J. M., & del Campo, I. (2004). Approximation of sigmoid function and the derivative for hardware implementation of artificial neurons. *IEE Proceedings-Circuits, Devices and Systems*, *151*(1), 18-24.

[18] Maas, A. L., Hannun, A. Y., & Ng, A. Y. (2013, June). Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml* (Vol. 30, No. 1, p. 3).

[19] Kim, S., Wimmer, H., & Kim, J. (2022, May). Analysis of deep learning libraries: Keras, pytorch, and MXnet. In *2022 IEEE/ACIS 20th International Conference on Software Engineering Research, Management and Applications (SERA)* (pp. 54-62). IEEE.

[20] Rankin, R., & Wozniak, S. (1976, August 1). Floating point routines for the 6502. *Dr. Dobb's Journal of Computer Calisthenics & Orthodontia*, *11*(7), 17. https://doi.org/10.1016/0165-6074(83)90086-8