**A Fast Deployable Model at Minimal Cost for Handwritten Digit Recognition**

**Jayson Mercurio1, Alexander Choi1, Jose L. Guzman1, Ayesha Iqbal1, Kevin Yamada2, and Hao Jiang2**

**1Cañada College, Redwood City, CA**

**2San Francisco State University, San Francisco, CA**

**Abstract**

Image recognition is a growing technology with deep learning as its framework to process a large pool of images. Deep learning is a method in computer science that allows a computer to learn and adapt to different kinds of data. One specific architecture of deep learning is the multilayer perceptron (MLP), which takes in input, feeds it forward through multiple hidden layers, then produces an output. For the purposes of the present study, the two main components of this research include using an MLP model/ algorithm along with the Modified National Institute of Standards and Technology (MNIST) dataset to create a model that is both fast and yields high accuracy. In order to implement the model, an open-source library such as TensorFlow is used to program and validate. Model validation will be evaluated by the runtime and accuracy output using the MNIST dataset. When the validated model is complete, it can be deployed by the NVIDIA Jetson TX1 supercomputer for mobile support. The NVIDIA Jetson TX1 contains 256 CUDA cores that can utilize parallel processing which is important when using TensorFlow. Because TensorFlow implements the use of GPUs parallel with the CPU, the Jetson is a good choice to not only train the model, but also take in live images to recognize.

1. **Introduction**

With recent interest in deep learning networks come greater demand for specialized hardware capable of training networks.[1] The NVIDIA Jetson TX1 embedded supercomputer was developed with the sole purpose of accelerating deep learning computations in a mobile environment. Utilizing 256 CUDA (Compute Unified Device Architecture) cores in its Maxwell-based GPU, the Jetson TX1 can train neural networks many times faster compared to traditional CPUs. Another advantage the Jetson holds is lower power consumption over traditional GPUs, making the module ideal for mobile purposes such as drones, vehicles, and other autonomous devices. Due to the advantages of the Jetson, the purpose of this internship project was the construction of a fast-training neural network model with high accuracy for live digit recognition.

Entering this internship, many members were unfamiliar with the concepts and mathematics behind the project. Therefore, a primary goal of this project was to familiarize ourselves with the subject, alongside completing a functional model for the Jetson, to gain research experience from our laboratory. The internship group consisted of one full-time member and three half-time members alongside a graduate student mentor. The full time-scale of this project was ten weeks of research, including weekly planning meetings as well as presentations on the previous week’s work.

1. **MLP Network Overview**

For the purposes of this study, a multilayer perceptron (MLP) neural network was prepared for implementation into the Jetson. A MLP neural network is a type of deep learning network capable of probabilistically classifying objects into categories, e.g. recognizing a cat versus a dog. A neural network trains in two stages: the training stage and the test stage. Using a type of simple cross validation, the neural network begins in the training stage where the network modifies itself to recognize patterns through weights, then the network’s capabilities are then tested in the test stage, often referred to as the inference stage. During this stage, a network is introduced with new data and is compared to the previous stage to evaluate. This network is defined as a feed-forward network with backpropagation due to the network’s nature of processing a dataset forward through the network of perceptrons and updating the network with training optimizations.

During the training phase of our network, all inputted data must pass through the network of neurons in a feed-forward manner, as shown in Fig. 1. A full forward and backward pass of the training data is described as an epoch. For a large majority of models, multiple epochs are needed for high accuracy. Within each epoch the data is split even further into batches as the network is under the memory constraints of the platform.

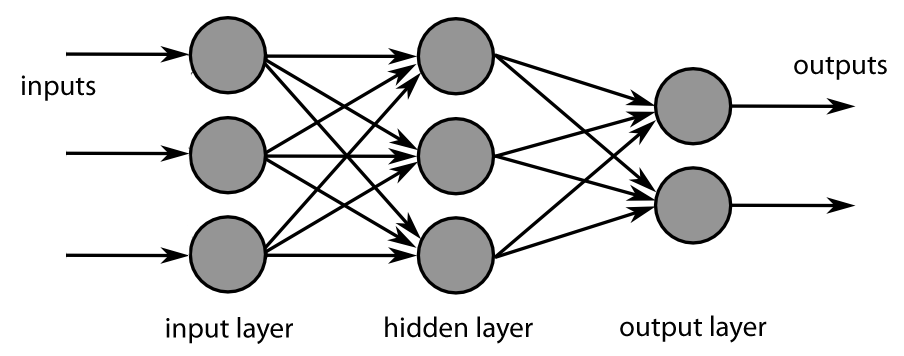


Figure 1: Example feed-forward neural network

The input is defined as the data transformed into a one-dimensional vector, while the output layer activates in response of the probability distribution resulting from the hidden layers and contains the neurons needed to classify the dataset, which in this network’s case are integers from 0 to 9. The network’s hidden layers contain the neurons, or perceptrons, of the network that evaluate the incoming data with associated weights. These perceptrons, or more colloquially called hidden units, contain an activation function as well as the summations of a product of the weights and input values:

|  |  |  |
| --- | --- | --- |
|  |  | (1) |

where defines the weights associated with the inputs, the inputs, and the number of elements from the input layer. Once the inputs are evaluated in the perceptron and the relevant activation functions triggered, the results are passed into a classifier to create a probability distribution and “guess” the correct label for the data. The weights are then adjusted for future inputs after obtaining results using the backpropagation methods.

Training through backpropagation involves the adjustment of the training weights by calculating the gradient of the loss function (or cost function) from the classifier. The gradient can be conceptualized as the cost of the function, or how accurate the network was at evaluating the input in comparison to the actual value. The classifier used in our constructed network is a softmax classifier with cross-entropy loss and logits, defined as follows:

|  |  |  |
| --- | --- | --- |
|  | for | (2) |
|  |  | (3) |

where the softmax classifier takes in a -dimensional vector and outputs a similarly sized vector for any th neuron containing real values between 0 and 1. Taking the gradient of the loss function, defined as above, the gradient is computed to find the cost function. Due to the robustness of this classifier in related research, it was selected as our primary activation layer.[1] Because a cost function becomes more inaccurate as the function increases in value, an optimizer algorithm is used to find the local minima of the cost function. Many optimizer algorithms exist and are a constant source of research in neural networks, but all exist as some form of gradient descent to find minimal cost and update the model’s parameters accordingly.[2][4][5] For the purposes of this research, standard TensorFlow optimizers will be used.

1. **MNIST Dataset and TensorFlow**

The MNIST dataset is one of the most widely recognized datasets available to the public. It is a balanced set (an equivalent number of elements in each classification) of images of handwritten digits ranging from 0 to 9 and contains a total of 60,000 training images and 10,000 test images. Each image is sized at 28 x 28 pixels, totaling 784 mapped values to input and evaluate. These mapped values are assigned to a one dimensional vector, in expression (1). Due to the size of the dataset, training time may take up to 10 minutes for most modern computers. For specialized equipment such as a NVIDIA Graphics Processing Unit (GPU) or Google Tensor Processing Unit (TPU), the same network may only take a minute or seconds as these devices are far more capable performing matrix computations.[6] As such, our primary testing platforms utilized a NVIDIA GTX 1070 GPU, which contains 1920 CUDA cores found in the Jetson, for our initial validation tests.

As the Jetson TX1 natively supports several deep learning libraries, the library we implemented for this study is the TensorFlow open-source, Python-based library created by Google.[7] The library contains common neural network elements that were utilized in the creation of our model such as predefined optimizers and activation functions. As such, TensorFlow was chosen over other alternative neural network libraries such as Keras and Caffe.

1. **Validation Tests**

To prepare an ideal, lightweight neural network model for import into the Jetson, a series of validation tests were performed to tune the parameters of the model. A network is validated by passing through the training and inference stages. During these tests, the main metric indicating a network’s performance is its accuracy; however, runtime was also considered in later tests in consideration of the Jetson’s needs.

|  |  |
| --- | --- |
| **Baseline Network** | |
| Batch Size | 256 |
| Training Epochs | 100 |
| Learning Rate | 0.001 |
| Optimizer | Adam |
| Hidden Units | 100 |
| Activation Function | ReLU |
| Number of Hidden Layers | 1 |

Table 1: Initial baseline network utilized during validation tests

Seven influential model parameters were selected for tuning, as shown in Table 1. Each parameter was assigned a default value to construct a baseline network.[4][2] Parameters were then varied individually and trained several times to produce acceptable averaged results.

|  |  |
| --- | --- |
| Figure 2: Number of Epochs vs. Accuracy[%] | Figure 3: Number of Hidden Units vs. Accuracy[%] |

For the number of training epochs and number of hidden units, with results represented in Fig. 2 and Fig. 3, accuracy reached a peak value then stabilized as the values of each parameter increased. This strongly suggests that additional accuracy is no longer possible to gain with the baseline network in a practical manner. As these tests were a series of independent parameter tuning tests, a range of values was taken to construct a more refined set of parameters in further tests.

|  |
| --- |
| Figure 4: Activation Function vs. Accuracy[%] |

Figure 5: Optimizer vs. Accuracy[%]

The activation functions tested were chosen primarily due to their differentiable nature. This is a required characteristic of the perceptron for an optimizer to be utilized. The functions that were ultimately chosen were the Rectified Linear Unit (ReLU) and the sigmoid functions due to their common usage in published neural networks as well as their obvious performance advantage over other functions as seen in Fig. 4.[2] The optimizers shown in Fig. 5 were chosen in a similar fashion as the Adaptive Moment Estimation (Adam) optimizer is commonly used for its fast convergence rate and RMSProp for the management of the learning rate over large training periods.[2][5]

|  |  |
| --- | --- |
| Figure 6: Learning Rate vs. Accuracy[%] | Figure 7: Number of Hidden Layers vs. Accuracy[%] |

The learning rate parameter refers to the scale of the optimizer’s adjustment of the loss function. Depending on the optimizer in question, the learning rate may affect the optimizer’s overall performance accordingly, therefore, a conventional value of 0.01 was taken as our parameter value for the next stage of tests.[8] The number of hidden layers was left as its default value due to suffering performance with greater values as seen in Fig. 7, and our choice is supported by established literature.[9]

Figure. 8: Batch Size vs. Accuracy[%]

Lastly, a batch size was arbitrarily chosen at a value of 64 due to the linear performance (Explain more or better) seen in Fig. 8. This choice was given a caveat to be variable when utilizing the network on the Jetson in consideration of the mobile platform’s hardware constraints.

Once the initial tuning of the parameters was completed, a second series of validation tests were run to tune the model further. These tests were done on a single platform containing a GTX 1070 utilizing CUDA architecture, 16GB of RAM, and an Intel i7 processor. The models utilized were constructed from the range of parameters taken from the independent validation tests and dependently varied. Added to these dependent validation tests was the consideration of the runtime as the chosen model resulting from the tests would be the model imported into the Jetson.

Figure. 9: Runtime[s] vs. Accuracy[%]

Intel i7 920 @ 3.8Ghz, NVIDIA GTX 1070, 16GB RAM

Since high accuracy and low runtime was prioritized, the cluster of models highlighted in the plot in Fig. 9 was of great interest. Due to the overarching trend of low epochs and large batch sizes in this grouping, a low epoch count as well as a batch size of 1024 was selected. The activation function and the optimizer were found to be the same as the baseline network. The optimal number of hidden units found was found to be 800 hidden units, which aligned with frequent suggestions that the number of hidden units should align with the number of input units.[9] Also critical to note is the runtime of this group of models: each model took less than 60 seconds to train while achieving 95% - 97% accuracy, thus achieving our goal of constructing an ideal model to implement onto the Jetson.

|  |  |
| --- | --- |
| **Baseline Network** | |
| Batch Size | 256 |
| Training Epochs | 100 |
| Learning Rate | 0.001 |
| Optimizer | Adam |
| Hidden Units | 100 |
| Activation Function | ReLU |
| Number of Hidden Layers | 1 |

Table 2: Finalized Model

1. **Conclusion**

Using a multilayer perceptron type network for MNIST training isn’t anything groundbreaking; it’s much like printing “Hello World” into the terminal in the eyes of many neural network researchers. However, while most other research projects on MNIST and MLP tried to maximize the total accuracy output, this study also took account of runtime for the purposes of mobility. Hundreds of different validation tests were done while other work required extensive porting work done for the model to function on the Jetson. During this program, valuable skills were gained from the challenges and complications that arose to meet our in-experienced research members, such as basic teamwork in a research group, proper compartmentalization of work to produce results during a limited timeframe, and creating proper presentations that clearly explain the work done in a persuasive manner. Furthermore, the data reveals that our resultant model achieved low runtime and maintained reasonable accuracy while also successfully executing on the Jetson to take in live digits from a webcam, completing all our primary goals for this internship.

**Acknowledgment**

This project is supported by the US Department of Education through the Minority Science and Engineering Improvement Program (MSEIP, Award No. P120A150014); and through the Hispanic-Serving Institution Science, Technology, Engineering, and Mathematics (HSI STEM) Program, Award No. P031C110159.

**Bibliography**

1. Bridle, J. S. (1989). Training Stochastic Model Recognition Algorithms as Networks can lead to Maximum Mutual Information Estimation of Parameters. *Advances in Neural Information Processing Systems 2*.
2. Ruder, S. (2017). An Overview of Gradient Descent Algorithms. *[1609.04747] An Overview of Gradient Descent Algorithms.*
3. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM,* *60*(6), 84-90. doi:10.1145/3065386
4. Simard, P., Steinkraus, D., & Platt, J. (n.d.). Best practices for convolutional neural networks applied to visual document analysis. *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.* doi:10.1109/icdar.2003.1227801
5. Schaul, Tom, Ioannis Antonoglou, and David Silver. (2014). Unit Tests for Stochastic Optimization. *[1312.6055] Unit Tests for Stochastic Optimization*.
6. Ian Goodfellow, Yoshua Bengio & Aaron Courville (2016). Deep Learning*. MIT Press*. doi:10.1038/nature14539
7. M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin et al. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *[1603.04467]*
8. Lecun, Y. A., Bottou, L., Orr, G. B., & Müller, K. (2012). Efficient BackProp. Lecture Notes in Computer Science *Neural Networks: Tricks of the Trade*, 9-48. doi:10.1007/978-3-642-35289-8\_3
9. Stathakis, D. (2009). How many hidden layers and nodes? *International Journal of Remote Sensing*, 30(8), 2133-2147. doi:10.1080/01431160802549278