Jetson TX1: The Next Step in Deep Learning?

George Matelich, Alec Singer, Rob Jones, Harrison DeKnight

Washington and Lee University
Lexington, VA 24450

Abstract:
A constant need in deep learning is computing power. NVIDIA recently released the Jetson TX1 Graphics Processing Unit; a 256 CUDA core processor that supposedly speeds up training time on deep-layer networks. Unfortunately, like many other products of its kind, the TX1 was rushed to market with little documentation. Furthermore, there is no clear evidence that the new TX1 is worth the $600 investment. To determine this, we compared the training speed of the TX1 to other processors, such as those found in an Apple laptop, HP desktop, and old NVIDIA Quadro K620 GPU. Our experiment was to train and test a multilayer perceptron with the MNIST data set to classify digits. We recorded the time it took to train on 100 epochs and found that the Quadro K620 was fastest, followed by the Apple, HP, and then TX1. Our findings pose the following questions: is the TX1 a waste of money, is NVIDIA releasing products too early, why is the documentation so poor, what is its unique advantage, and is there something we are not exploiting in this processor? Deep learning is a sect of computing, currently held back by processing speed and the Jetson TX1 may not be the logical next step forward.

Data:

Device Specifications for standard CPUs:

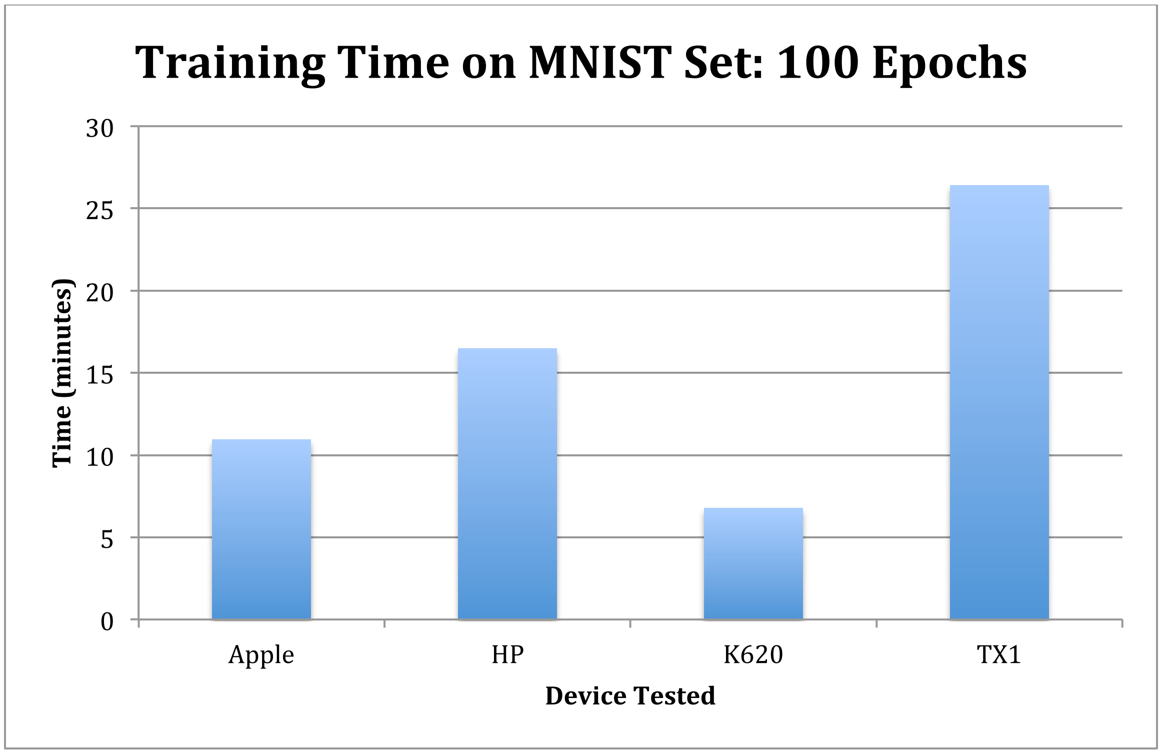
Maker	Name	Speed (GHz)	Threads	Cores
Apple	Intel Core i7	3.1	4	2
HP	Intel i7-4790	3.6	8	4

Device Specifications for NVIDIA GPUs:

Maker	Name	GPU Memory (GB)	CUDA Cores	Max Power Consumption
NVIDIA	Quadro K620	2	384	41 Watts
NVIDIA	Jetson TX1	4	256	5.7 Watts

Training time on MNIST set for 100 epochs:

Apple	HP	K620	TX1
10.93 minutes	16.48 minutes	6.77 minutes	26.41 minutes

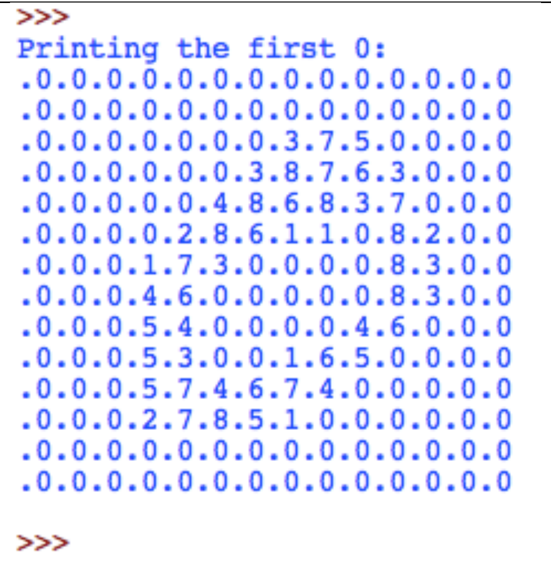
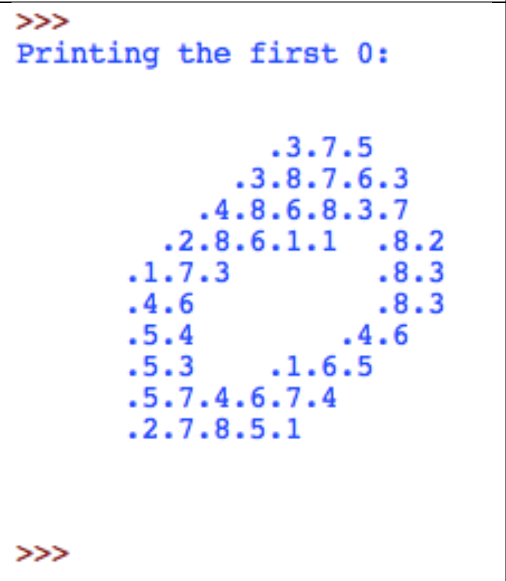


The Data Set Used:

The data set we trained on was the MNIST originally provided by Yann LeCun. We used 2,500 digits from the larger set for training, and another 2,500 for the testing. The digits (samples pictured below) were scanned and converted to a 14x14 array of values from 0.0 to 1.0, where a higher value corresponds to a darker pixel.

Samples from the MNIST Data set

Converted Data

Raw data	Non-zero values
	

```
>>>
Printing the first 0:
.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0
.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0
.0.0.0.0.0.0.0.3.7.5.0.0.0.0.0
.0.0.0.0.0.0.3.8.7.6.3.0.0.0.0
.0.0.0.0.0.4.8.6.8.3.7.0.0.0.0
.0.0.0.0.2.8.6.1.1.0.8.2.0.0.0
.0.0.0.1.7.3.0.0.0.0.8.3.0.0.0
.0.0.0.4.6.0.0.0.0.0.8.3.0.0.0
.0.0.0.5.4.0.0.0.0.4.6.0.0.0.0
.0.0.0.5.3.0.0.1.6.5.0.0.0.0.0
.0.0.0.5.7.4.6.7.4.0.0.0.0.0.0
.0.0.0.2.7.8.5.1.0.0.0.0.0.0.0
.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0
.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0

>>>
```

```
>>>
Printing the first 0:


        .3.7.5
      .3.8.7.6.3
    .4.8.6.8.3.7
  .2.8.6.1.1  .8.2
.1.7.3        .8.3
.4.6          .8.3
.5.4        .4.6
.5.3    .1.6.5
.5.7.4.6.7.4
.2.7.8.5.1



>>>
```

The Code:

The code used for this test was borrowed from deeplearning.net and creates a Multi-Layer-Perceptron (MLP). An MLP is the connection of inputs to hidden units, and hidden units to a softmax layer, through different weights that are changed by using back propagation to calculate their respective error. This MLP would train on small mini-batches of data while consistently updating the error. We chose 100 epochs for this experiment. An epoch is essentially one full round of training on a data set. To speed up the training, the code implements the C compiled libraries NumPy and Theano, which allows for a significant speed up in many computations. They are useful for everything from the dot products to more complex calculus applications.

```
while (epoch < n_epochs) and (not done_looping):
    epoch = epoch + 1
    for minibatch_index in range(n_train_batches):

        minibatch_avg_cost = train_model(minibatch_index)
        # iteration number
        iter = (epoch - 1) * n_train_batches + minibatch_index

        if (iter + 1) % validation_frequency == 0:
            # compute zero-one loss on validation set
            validation_losses = [validate_model(i) for i
                                 in range(n_valid_batches)]
            this_validation_loss = numpy.mean(validation_losses)

            print(
                'epoch %i, minibatch %i/%i, validation error %f %%' %
                (
                    epoch,
                    minibatch_index + 1,
                    n_train_batches,
                    this_validation_loss * 100.
                )
            )
```

The code above trains the network for a predetermined number of epochs. On each epoch, it loops through a mini-batch and trains the model. After each epoch it tests the model against a validation set. If the error is not decreasing against the validation set then training is stopped to prevent over fitting. The epoch number, mini-batch index, and error on the validation set are then returned.

A Niche:

The TX1 does serve one, very unique purpose due to its low power consumption. It is very well suited to use on a drone because it is the size of a credit card and draws only 5.7 watts. The module is 50x87mm, "encased in a heat sink that brings the volume to about the same size as a pack of cigarettes" (Benchoff). It weighs approximately 45 grams with the heat sink (Stone). This could lead to improved autonomous vehicles that can process massive amounts of data while airborne. The TX1 is a small, light, low consuming GPU that would not drastically alter the payload. Figure 1 below, from Hack-A-Day illustrates this point by comparing the TX1 processing speed per watt of power consumed versus a similar model of Intel processor.

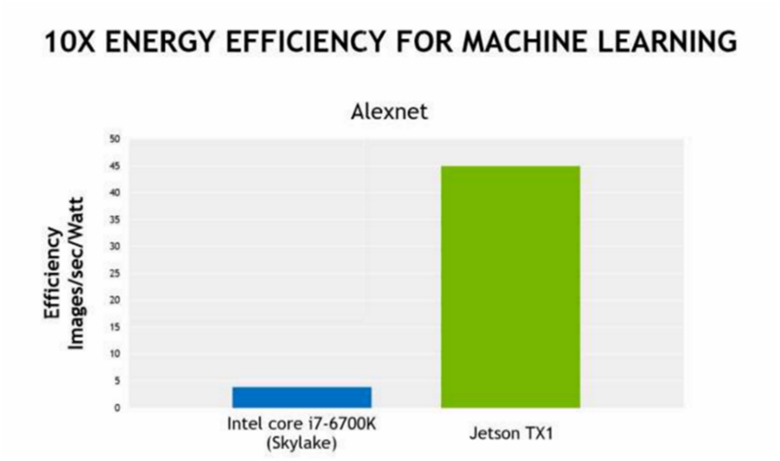


Figure 1: Energy Efficiency Comparison (Hack-A-Day)

According to Hack-A-Day, "this credit card-sized module is a 'supercomputer' advertised as having more processing power than the latest Intel Core i7s, while running at under 10 Watts" (Benchoff). Our findings, however, contradicted NVIDIA's advertisement. Still, the Jetson TX1 does excel in one niche: it is incredibly efficient.

Figure 2: Drone with Ubuntu-powered computer (Verma)

Figure 2 is an example of a drone that utilizes a powerful, “Ubuntu-powered computer” (Verma). In this case, the computer is used with on onboard camera and an “NVIDIA GPU” to process images (Verma). Though it is not the TX1, it proves insight into the practical and ideal use for a board of this type.

Summary:

The TX1 board has a way to go. With sparse documentation and no proof that the TX1 is faster than any of the other platforms we tested, the TX1 is not worth the $600 price tag. It may be useful to those envisioning drone applications, as its computation/power ratio is stunning, but the lack of raw computing power makes it a bad investment for terrestrial deep learning applications. Furthermore, we conclude that the headache of setting up the board is not to be underestimated. It is difficult to operate the device as root, install packages, and navigate the directories of the machine. It also took a long time to install Theano because we did not change the flags to expect a 32-bit word size instead of 64. Though NVIDIA’s product may appeal to some, the TX1 disappointed us in the end.

Bibliography

Benchoff, B. (2015, November 24). The Nvidia Jetson TX1: It's Not For Everybody, But It Is Very Cool. Retrieved March 28, 2016.

Jetson TX1 Embedded Systems Module from NVIDIA Jetson. (n.d.). Retrieved March 28, 2016.

Nielson, M. (n.d.). Neural networks and deep learning. Retrieved March 28, 2016.

NVIDIA Quadro K620NVIDIA Quadro K620 Part No. VCQK620-PB. (n.d.). Retrieved March 28, 2016.

Stone, T. (n.d.). Jetson TK1 Promo Offer. Retrieved April 03, 2016

Verma, A. (2015). This Ubuntu-Powered Drone Is Actually A Powerful Flying Computer. Retrieved April 03, 2016

6th Generation Intel® Core™ Processors. (n.d.). Retrieved March 28, 2016.