CSC Deutschland GmbH

GPU Computing Pilot for DB VXR

Runtime tests with TensorFlow and Theano in Python

Autor: Zilong Zhao
✉: zzhao3@csc.com
April 19, 2016

Contents

1 Introduction

GPU-accelerated computing is the use of a graphics processing unit (GPU) together with a CPU to accelerate scientific, analytics, engineering, consumer, and enterprise applications. Pioneered in 2007 by NVIDIA, GPU accelerators now power energy-efficient datacenters in government labs, universities, enterprises, and small-and-medium businesses around the world. GPU-accelerated computing offers unprecedented application performance by offloading compute-intensive portions of the application to the GPU, while the remainder of the code still runs on the CPU. From a user's perspective, applications simply run significantly faster. A simple way to

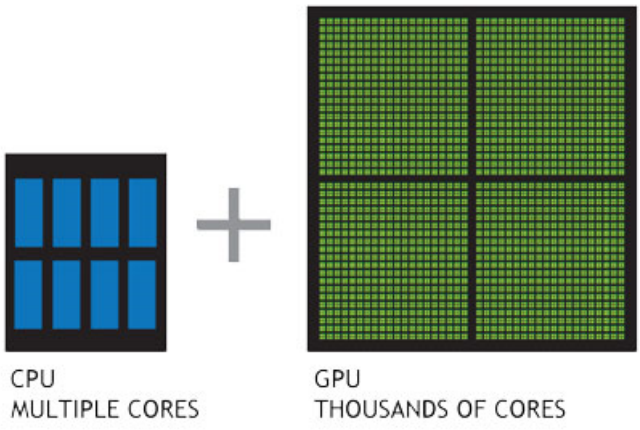

Figure 1: CPU vs. GPU[1]

understand the difference between a CPU and GPU is to compare how they process tasks. A CPU consists of a few cores optimized for sequential serial processing while a GPU has a massively parallel architecture consisting of thousands of smaller, more efficient cores designed for handling multiple tasks simultaneously.[1]

The aim of this report is to compare the runtime of CPU and GPU by using the DB data `HUNL_to_HSUD_v3.csv`.

The computing task will be performed in Python on two platforms:

[1]See [1]

- The ThinkPad W541 from CSC, which has Intel i7-4810MQ CPU, and a graphics chip NVIDIA Quadro K1100M with 2G VRAM,
- A desktop PC with Intel i7-6700K CPU, and a graphics chip NVIDIA Geforce GTX980Ti with 6G VRAM.

The following table shows the specifications of the two graphics chips:

GRAPHICS FEATURES	Quadro K1100M	GTX 980 Ti
CUDA Cores	384	2816
Base Clock (MHz)	705	1000
Standard Memory Config	2GB	6GB
Compute Capability	3.0	5.2

Table 1: Quadro K1100M vs. GTX 980 Ti

Not only the higher compute capability and more CUDA cores, but the larger memory of GTX 980 Ti ensures a faster data transfer directly on graphics card as well.

Also, we check the specifications of the two CPUs:

CPU FEATURES	i7-4810MQ	i7-6700K
Number of cores / threads	4/8	4/8
Operating frequency (GHz)	3.8	OC@4.5
L1 cache (KB)	256	256
L2 cache (KB)	1024	1024
L3 cache (KB)	6144	8192

Table 2: Quadro K1100M vs. GTX 980 Ti

2 The dataset

The csv data HUNL_to_HSUD_v3.csv has 126525 rows and 42 parameters. It represents the train data from the station Unterlüß to Suderburg[1] over three

[1] http://fahrweg.dbnetze.com/fahrweg-de/medien/veroeffentlichungen/betriebsstellen.html

years (start: Jan. 1st 2012, end: Apr. 4th 2015). We randomly choose 15% of the dataset as test data for validation.

3 Implementation

We use **Keras** for the neural network modelling. **Keras** is a minimalist, highly modular neural networks library for Python, capable of running on top of either TensorFlow or Theano.

TensorFlow was developed originally by the Google Brain Team for conducting research in machine learning and deep neural networks. The framework's name is derived from the fact that it uses data flow graphs, where nodes represent a computation and edges represent the flow of information – in Tensor form – from one node to another. Unlike any other framework, TensorFlow has the ability to do partial subgraph computation, which involves taking a subsample of the total neural network and then training it, apart from the rest of the network. This is also called Model Parallelization, and allows for distributed training.

Theano is an open source project, primarily developed by a machine learning group at the Université de Montréal. Theano computes the gradient when determining the backpropagation error by deriving an analytical expression. This eliminates accumulation of error during successive derivative calculations using the chain rule, which other frameworks accrue due to their use of numerical methods. Backpropagation with neural networks always involves some amount of stochastic component, in order to avoid getting trapped in a local minima. A noisy or numerical computation might not have too detrimental of an effect on results, at least during the start of the backpropagation, where the stochastic component might be largest, depending on which gradient search method is being used. For search methods which require high accuracy for the gradient in the latter stages of the global minima search, Theano would be expected to produce better results.

Using **Keras** we build a neural network with two hidden layers, each layer has 30 neurons. To deal with overfitting, we introduce a technique named dropout. The key idea is to randomly drop units from the neural network during training. Such neural network can be easily built by following Python code:

```
model = Sequential()
model.add(Dense(30, input_shape=(16,)))
model.add(Activation('tanh'))
model.add(Dropout(0.001))
model.add(Dense(30))
model.add(Activation('tanh'))
model.add(Dropout(0.01))
model.add(Dense(nb_classes))
model.add(Activation('softmax'))
```

After topology modelling, we train the neural network with the RMSProp optimizer[1] and cross entropy as our cost function.

```
rms = RMSprop()
model.compile(loss='categorical_crossentropy',
  ↪ optimizer=rms)

model.fit(X_train, Y_train,
          batch_size=batch_size,
  ↪ nb_epoch=nb_epoch,
          show_accuracy=True, verbose=2,
          validation_data=(X_test, Y_test))
```

Besides, the Python package `timeit` is used for runtime logging. Now we show some results for the training epoch 2000

FRAMEWORK PLATFORM	ThinkPad W541	Desktop
Theano (CPU)	2294.6760 s	1683.2134 s
Theano (GPU)	2120.5801 s	918.6653 s
TensorFlow (CPU)	5045.6896 s	4474.0128 s
TensorFlow (GPU)	8709.7587 s	4695.4436 s

Table 3: The runtime contest of Theano and TensorFlow

[1]For more information, see `http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`

In terms of speed, TensorFlow is slower than Theano, but it's still in development. We expect the performance of future releases will be similar to Theano. In order to get a better view of the runtime, we record it for each step with 50 epochs in interval $[50, 2000]$ and get a plot as following. The statement *Tensor-*

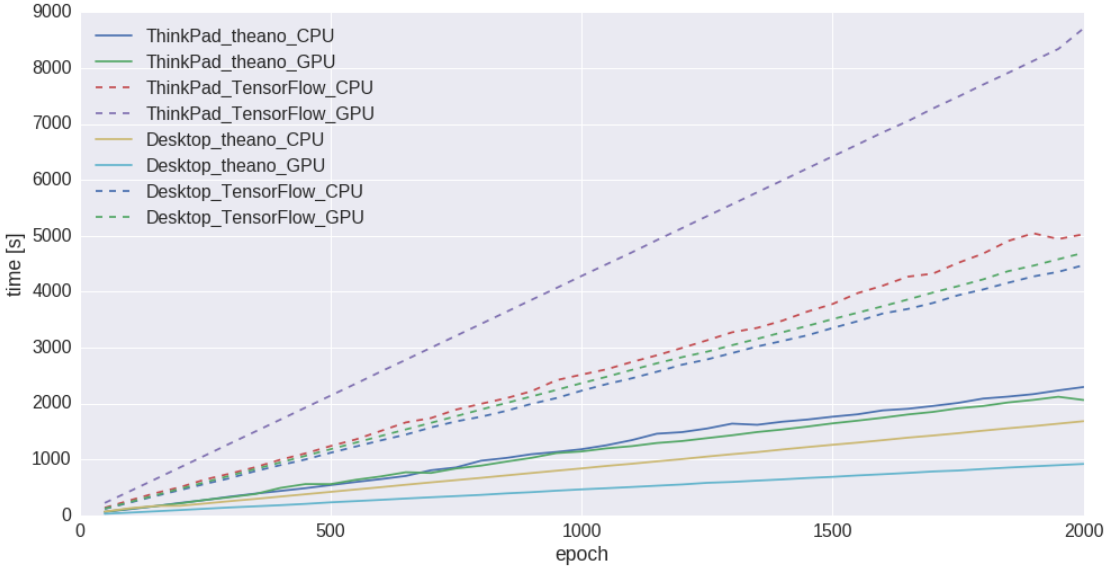

Figure 2: Runtime curves on two platforms based on CPU and GPU

Flow is slower than Theano is shown here during the whole computation. By using the Theano framework, we see some speed-up effect on GPU of both platforms, especially on the GTX `980 Ti` graphic card. As we see the similar CPU specifications in table 2, the runtime with CPU on the two platforms is not dramatically improved as GPU, which we expect. If we look closer at figure 2, we see the runtime curve on ThinkPad W541's CPU has significantly more knickpoints. Oppositely, not for the Desktop. Such phenomenon may be caused by the comparatively poor cooling effect of Laptop, the CPU automatically slowed down during the computation, in order to prevent overheating (The temperature was at 95 ℃ maximum).

After checking the runtime, we may ask how well the GPU performs. To answer this question, we use the **cross-entropy** loss as the cost function, aka. Log-loss,

which is defined by:

$$L(y_t, y_p) = -\Big(y_t \log(y_p) + (1 - y_t) \log(1 - y_p)\Big),$$

where y_t is true value and y_p is prediction. First, let's just concentrate on the Theano framework, here the four curves overlap almost perfectly. This rep-

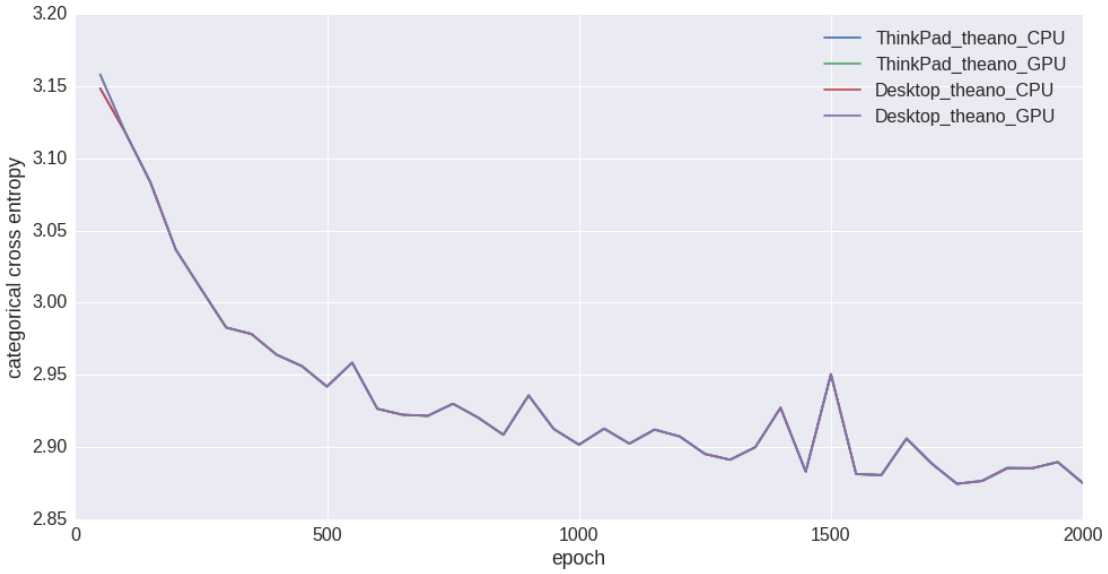

Figure 3: Cross entropy curves with Theano on two platforms

resents the result of GPU computing should be as well as CPU with Theano, independent on different platforms. Besides, the cross-entropy error converges to its minimum and it reaches already at about 1750 epochs.

Adding the curves for TensorFlow, we get some interesting observations in figure 4. Generally, TensorFlow does a better job with respect to the smaller cross-entropy error, except the computation on desktop with CPU, slightly above Theano at the end. We summarize in tabular 4 for the two frameworks in Python.

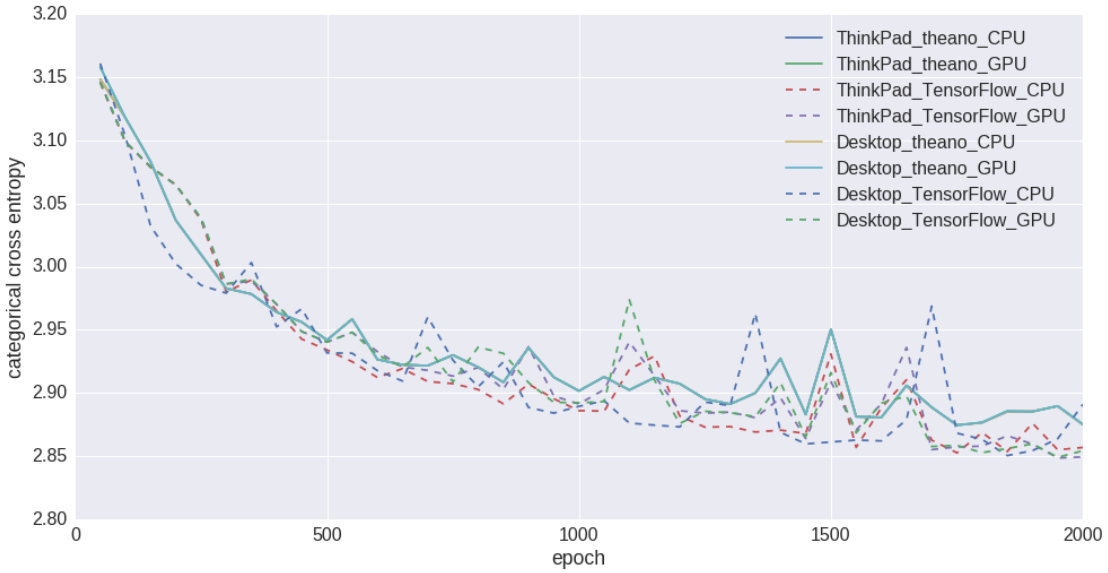

Figure 4: Cross entropy curves with Theano and TensorFlow

FRAMEWORK	Theano	TensorFlow
base language	Python	Python and C++
multi-GPU?	by default, no.	yes
execution speed	fast	slower than Theano

Table 4: Theano vs. TensorFlow

In fact, nearly one week ago, Google announced the new version 0.8 of TensorFlow with distributed computing support[1]. TensorFlow allows the training process on different kinds of devices, such as multi-core CPUs, GPUs, or even mobile processors.

4 Summary and outlook

In this report, applying the data from DB, we take a first look at the GPU computing and see how it accelerates the computation process. The next step is

[1]For more information, see `http://googleresearch.blogspot.de/2016/04/announcing-tensorflow-08-now-with.html`

to see the parallelizability on multiple GPUs, with that the cluster size can be scaled back for the reason of reducing cost.

We end this report with a distribution plot of time difference between prediction (by TensorFlow) and real value. The zero centered Gaussian-like distribution curve is tall and narrow, meaning the standard deviation is small. Such observation suggests the prediction from neural network with TensorFlow works well.

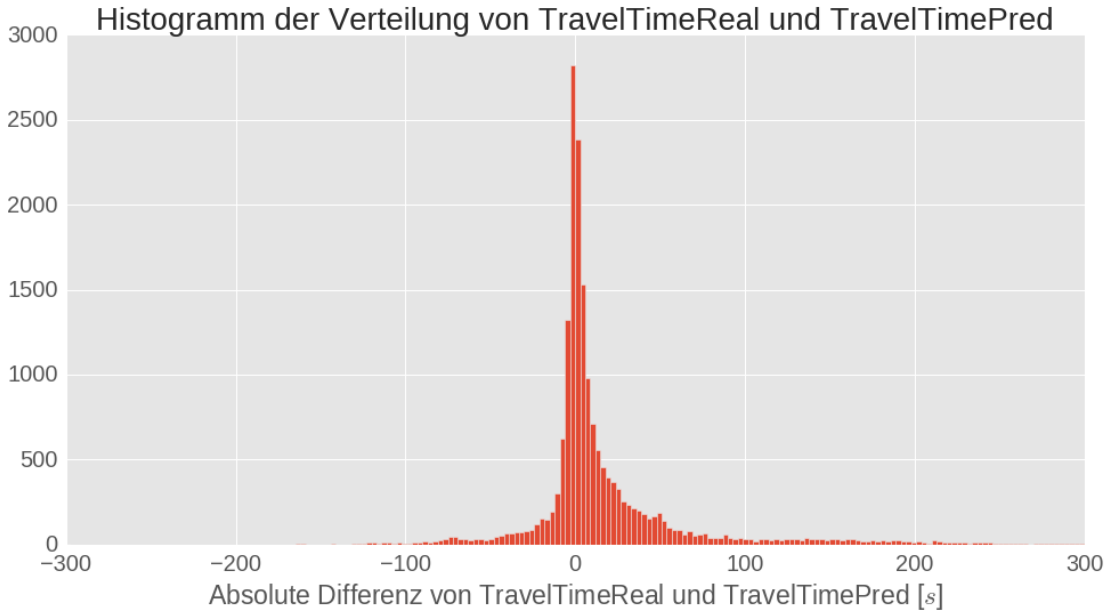

Figure 5: Distribution of time difference between prediction and real value

Bibliography

[1] www.nvidia.com *WHAT IS GPU ACCELERATED COMPUTING?* http://www.nvidia.com/object/what-is-gpu-computing.html

[2] https://developer.nvidia.com/ *CUDA GPUs* https://developer.nvidia.com/cuda-gpus

[3] *Deep Learning Frameworks: A Survey of TensorFlow, Torch, Theano, Neon, and the IBM Machine Learning Stack* https://www.microway.com/hpc-tech-tips/deep-learning-frameworks-survey-tensorflow-torch-theano-ca