NeuralTalk on Embedded System and GPU-accelerated RNN

Subhasis Das
CVA group, Stanford University
Stanford, CA, 94305
subhasis@stanford.edu

Song Han
CVA group, Stanford University
Stanford, CA, 94305
songhan@stanford.edu

Abstract

We hope to let the blind hear the description of what's happening in the world. We implemented the NeuralTalk [4] on Nvidia Jetson TK1 embedded system to evaluate the hardware efficiency of CNN and RNN. The system consists of a camera, a TK1 development board, and a speaker. We show that the NeuralTalk model not only work well on dataset images which has been carefully categorized, but also generalize to images captured randomly in real world. The system captures image and process it through CNN and RNN to convert to text caption, and finally convert it to voice. It can act as a wearable device to help the blind.

The other part of our work is implemented the GPU-accelerated RNN training and testing for NeuralTalk. We accelerated RNN feedforward by $\mathbf{9}\times$ and training by $\mathbf{18}\times$ over the provided CPU implementation on a Jetson TK1. The CNN+RNN works end to end almost in real time at very low power consumption: it captures an image converting it to text at 1 image per second, consuming only 8 watt power. We provided the profiling of the time spent on each pipeline stage, and analyzed the bottleneck.

1 Introduction

Neural networks have become ubiquitous in applications ranging from computer vision [5] to speech recognition [2, 3] and natural language processing [1, 6]. Those algorithms outperform traditional algorithms but require a lot of computation power, thus requires high-end GPUs to do the computation. A modern GPU typically consumes 250W when running at full speed [8], which makes it hard to apply to mobile applications. Thus the demand of implementing deep learning algorithms in real time on low cost embedded system has been increasing.

Nvidia Jetson TK1 development board is the first embedded supercomputer [9] that has high computation power with low power consumption. It features a Kepler GPU with 192 cores, an NVIDIA 4-plus-1 quad-core ARM Cortex-A15 CPU, and can run AlexNet with Caffe in 34ms per image. We are going to port the CNN-RNN model of neural talk on this board, install a camera and speaker, let TK1 generates a description of the neighboring world and convert it to speech.

The original NeuralTalk doesn't have GPU acceleration. To fully utilize the powerful CUDA cores on the TK1 board, we implemented the GPU-accelerated NeuralTalk, which accelerated the feedforward LSTM by $8.9\times$. The implementation is based on PyCUDA and scikits.cuda. We made a util libraray that wrapped the common operations used in RNN, which also facilitated us building a GPU-accelerated training framework for RNN. Specifically, we implemented a GPU-acclerated GRU, which is more complicated than RNN but less complicated than LSTM, and compared the training speed on a modern CPU (Intel Core i7 Haswell 5930k) and modern GPU (NVIDIA TitanX), and received $18\times$ speed up.

The contribution of this work are:

Figure 1: Jetson board with Tegra-K1 hooked up to a webcam and an earphone

1. Implemented NeuralTalk on embedded system and studied the hardware efficiency
2. Accelerated NeuralTalk with GPU
3. Made a GPU util library which accelerated the training of RNN, GRU and LSTM.
4. Measured speed up on both embedded CUDA core v.s. ARM core and Titan X GPU v.s. Haswell CPU

2 Implementation

The end-to-end processing pipeline consists of using v4l2capture to capture image with a webcam, using Caffe with VGG16 CNN model to extract the 4096 dimension image feature, using RNN to generate sentences, and finally using pyttsx to convert it to speech. Figure 1 shows a picture of our testbed.

2.1 Image capture

We are using Microsoft LifeCam VX-5000 webcam that's connected with Jetson TK1 through USB 2.0 interface. v4l2capture and Image packages are used to capture image in python and return the numpy array for processing. The camera is intended to capture neighboring scenes in real time, instead of using just the images from the dataset.

2.2 Image feature extraction

Caffe is used to process the image and extract features. We used the VGG16 caffemodel to do feed forward till layer fc7 and take out the feature after the ReLU of fc7. The feature is 4096 dimension. VGG16 caffemodel is very deep and has 130 million parameters, making the total model size more than 528MB, plus the memory needed to store activations, the total amount of memory required is 732MB, which fits the 1GB memory of Jetson TK1.

Caffe has both CPU and GPU implementation, we evaluated both. The GPU implementation takes only 0.9 seconds to run VGG16 to process each image after caching all parameter in the GPU memory. For the cold memory, however, it takes 3s to complete the whole feed forward, since transferring the parameters from CPU to GPU memory takes time.

2.3 Image to sentence

We used the flickr8k dataset to train the RNN. After 8 hours it completed, but the model is too small (has 16 million parameters) and the result was not satisfying. Then we tried the trained COCO

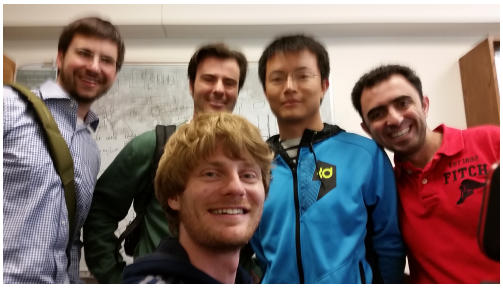
(a) a cat sitting on top of a refrigerator

(b) a man holding a nintendo wii game controller

(c) a cat sitting on top of a laptop computer

(d) a close up of a stuffed animal on a table

Figure 2: Example images and generated captions



(a) a couple of men standing next to each other

(b) a man holding a cellphone to his ear

model from the NeuralTalk model zoo, this model has 90 million parameters and has higher capacity, it worked pretty well. However, MKL doesn't work on ARM platform thus the CPU implementation is very slow. The baseline doesn't have GPU implementation yet.

2.4 Sentence to speech

The final step is to convert the image to voice. This is a simple step and pretty fast. We used the pyttsx package to do the conversion. We hope to let the blind people wear this machine to give them real time description what's happening in the world.

3 Experimental Results

3.1 Algorithm performance

Figure 2 shows the examples we apply our system to the images captured in the real world. It can be seen that the first caption doesn't make sense at all, the second and third captions make some sense, while the last caption is fairly accurate. Thus, we conclude that the baseline system is not accurate enough in our evaluation scenario. Whether this is an artifact of the algorithm or of the low quality images captured by the webcam remains to be seen.

In the next few sections, we discuss the performance of the Tegra-K1 SoC on the NeuralTalk workload.

3.2 Hardware performance

3.2.1 CNN on GPU and RNN on CPU

The default NeuralTalk model is implemented to get image features and generate captions on the CPU. The image features can be obtained by the Python API to Caffe. With this pipeline of image features via Caffe, and caption generation via NeuralTalk CPU code. Table 1 shows the time break down of the image caption pipeline. Although the TK1 features a decently powerful GPU, its CPU is pretty under-provisioned, and this results in very slow linear algebra operations on the CPU. Thus, the first thing to do would be to speed up the RNN step by doing the RNN computations on GPU. Also, it can be seen that the beam search degrades the CPU performance by a significant amount.

Operation	Time
VGG16 Feature Generation (CPU)	8.92s
VGG16 Feature Generation (GPU)	0.74s
RNN forward pass beam size 32 (CPU)	7.45s
RNN forward pass beam size 1 (CPU)	0.73s

Table 1: Speed of various operations in NeuralTalk pipeline on Jetson-TK1

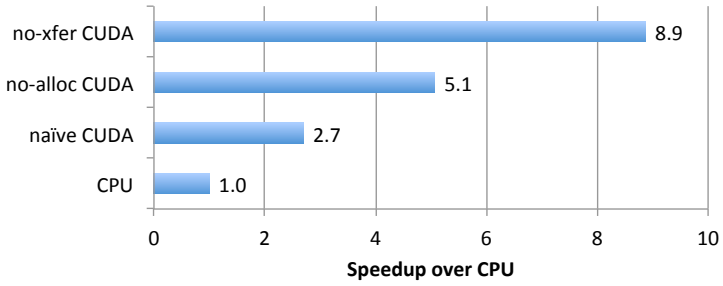


Figure 4: Speedups of various GPU codes over CPU

Thus, for near-real time operation, it is desirable to create algorithms which can generate realistic captions even without a beam search.

3.2.2 CNN and RNN on GPU

In this next step, we experimented with porting the RNN code to GPU. We used the PyCUDA and scikits.cuda libraries to create an equivalent LSTM layer as the NeuralTalk model on GPU. The performance of this model is shown as the bar labeled "naive CUDA" in Figure 4. As we can see, this achieves only a $\mathbf{2.7}\times$ speedup over the CPU model.

3.2.3 Removing GPU allocations

The performance of the naive GPU port was quite underwhelming. Thus, we looked into the reasons behind the low performance of the GPU code. One of the reasons that we found was that GPU allocations were being made in almost every function call in the naively ported GPU code. As a result, we strove to eliminate all memory allocations during the forward pass of the LSTM. To achieve this goal, we pre-allocated GPU buffers during the LSTM layer initialization. The results of this optimization can be found in the bar labeled "no-alloc CUDA". We can see that this optimization alone results in a $1.9\times$ speedup over the naive CUDA porting, and a $\mathbf{5.1}\times$ speedup over the CPU code overall. CUDA programming manuals such as [7] also advocate against excessive usage of `cudaMalloc` and `cudaFree` API calls.

To perform this optimization, we had to deviate significantly from the scikits.cuda and PyCUDA libraries, since many of the API functions present in those libraries can not exploit the presence of a pre-allocated memory pool. To circumvent this problem, we wrote our own versions of several elementwise functions and matrix vector product functions.

3.2.4 Removing CPU-GPU transfers

The second bottleneck we observed was that any CPU to GPU transfers had a high overhead, but were necessary in certain parts of the forward pass, e.g., for transferring the maximum scoring word index at each timestep. In order to eliminate this overhead, we pre-allocated a `GPUArray` for holding the word indices at each timestep, and also performed a texture memory based word vector loading at each step. These optimizations ensured that the word index did not need to be transferred to the CPU at each step. The performance with this approach is labeled as the "no-xfer CUDA". It

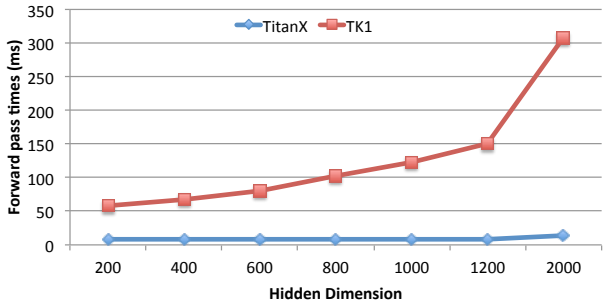


Figure 5: Forward pass times for Jetson TK1 and TitanX

can be seen that this version is $\mathbf{8.9}\times$ faster than the CPU version, and $1.8\times$ faster than the no-alloc version.

One problem with this approach is that it is not possible to tell in the GPU itself when the "STOP" token has been found. One way to remedy this problem would be to perform the forward pass in "batches", i.e., a few forward steps (somewhere around 4-5) can be made on the GPU without transferring anything to the CPU, and then a bulk transfer of these tokens can be made to the CPU to amortize the transfer overhead. We have not experimented with this technique.

3.2.5 GPU Training

Impressed with the success of our GPU forward pass, we also wrote the corresponding backward passes for GPU training. Writing the GPU code for the NeuralTalk LSTM code proved to be fairly difficult, so we decided to write a training code for a GRU layer instead. We found that GPU training over a TitanX GPU was almost $18\times$ faster than the CPU training for NeuralTalk code, with similar code complexity using the PyCUDA and scikits.cuda libraries. For a hidden dimension of 1024 neurons, an input dimension of 1024, and 10 timesteps, the GRU code on TitanX takes only **11ms/sample**, which means the entire COCO dataset can be trained in **880s**.

3.2.6 Forward pass timings with different hidden dimensions

We were also curious about how the performance of the CPU-accelerated LSTM changes with different hidden dimensions. The below figure shows that above $hdim = 600$ the time grows linearly with the hidden size, while below $hdim = 600$ the grow is proportional to hidden size. This is due to with small hidden size the overhead of memory allocation and data transfer could not be amortized. We also ran the same code over a TitanX GPU, and observed that the forward passes on the TitanX GPU were $22\times$ faster than the TK1 GPU, thus underlining its suitability for training.

3.2.7 Power Consumption

Since mobile systems are battery constraint, low power budget is crucial to deploy CNN and RNN. We measured the power consumption of NeuralTalk by attaching a power meter to the power supply. When the system is idle it consume only 2.8 watts, which is dissipated by the operating system. When running NeuralTalk, the peak power is 8.0 watts, and the average power is around 6 7 watts. Compared with a desk top GPU which has a peak power of 250 watts, this system is $30\times$ more power efficient.

4 Future Plan

Equipped with the powerful GPU acceleration tool for both training and testing, there's plenty of room to improve the model. First, there is opportunity to make improvements by incorporating attention point and segmentation information, since every word in the sentence corresponds to some of the segmentation bounding boxes. There's also opportunity to feed the image to every RNN step, instead of only at the beginning, by adding a separate gate controlling whether the image need to be

(a) Idle power consumption of NVIDIA Jetson TK1

(b) Peak power consumption running NeuralTalk on NVIDIA Jetson TK1

fed into that time stamp. If time permits we'll add in reinforcement learning and interact with users to improve the model.

Acknowledgments

We thank NVIDIA for the gifting of a Jetson Tegra K1 development board at GTC 2015

References

[1] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *The Journal of Machine Learning Research*, vol. 12, pp. 2493–2537, 2011.

[2] A. Graves, D. Eck, N. Beringer, and J. Schmidhuber, "Biologically Plausible Speech Recognition with LSTM Neural Nets," in *in Proc. of Bio-ADIT*, 2004, pp. 127–136.

[3] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech Recognition with Deep Recurrent Neural Networks," *arXiv:1303.5778 [cs]*, Mar. 2013, arXiv: 1303.5778. [Online]. Available: http://arxiv.org/abs/1303.5778

[4] A. Karpathy and L. Fei-Fei, "Deep visual-semantic alignments for generating image descriptions," *arXiv preprint arXiv:1412.2306*, 2014.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[6] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur, "Recurrent neural network based language model." in *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010*, 2010, pp. 1045–1048.

[7] Nvidia. Cuda c best practices guide. [Online]. Available: http://docs.nvidia.com/cuda/cuda-c-best-practices-guide/#allocation

[8] ——. Geforce gtx titan specifications. [Online]. Available: http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-titan/specifications

[9] ——. The world's first embedded supercomputer. [Online]. Available: http://www.nvidia.com/object/jetson-tk1-embedded-dev-kit.html