Efficient Theano 1D Convolutions for Variable Length Sequences

Final Report

Darshan Hegde
Center for Data Science
New York University
e-mail: darshan.hegde@cims.nyu.edu

May 19, 2015

Supervisor Prof. Georg Stadler
Course MATH-GA 2012.003 High Performance Computing

Abstract

In this repost, we motivate the need for efficient 1D convolution kernels that support variable length sequences for Theano. Then we provide details of implementation and optimizations. At last, we compare our implementation with Theano's default implementation with zero padding.

Motivation

Convolution Networks are used heavily used by Computer Vision Researcher and has outperformed other conventional techniques on many important benchmark tasks [5]. Theano [1] [2] is mathematical expressions compiler library which can efficiently compile both CPU and GPU codes. Many Computer Vision researchers have been using Theano [1] [2] for building and experimenting with Convolution Networks and has very efficient implementation of 2D convolutions on GPU. Since one needs to evaluate (and take gradients of weights w.r.t cost function) Convolution Networks on a large number of training images (usually color with 3 channels) during training of the network, 2D convolution is implemented as 4D tensor operation with input represented as (images-per-batch, channels, width, height) and output represented as 4D tensor (images-per-batch, num-conv-kernels, out-width, out-height). On a typical GPU implementation, training images are processed in mini-batches and each image in a given mini-batch is processed in parallel. All images in a given training dataset are of same size (height and width) and hence amount of computation done per image are uniform.

Recently, Natural Language Processing (NLP) researchers have applied Convolution Networks for tasks such as Sentiment Analysis [4] and Document Summarization [3] with state of the art performance. Convolution operation used in NLP are applied on 1D variable length sequences (Sentence is a sequence of words / Document is sequence of sentences). One can use efficient 2D convolutions in Theano to compute 1D convolutions by setting one of the dimensions (height or width) to 1. However, to compute 1D convolutions for sentences in a mini-batch (just like we do for images) one needs to use zero-padding for shorter sentences. Using this strategy, width of a given mini-batch is decided by longest sentence and for shorter sentence we are wasting many computation cycles where there is zero-padding. This effect of wasting computations due to zero-padding is quite significant in NLP because sentence are of varying lengths. Fig 1 shows distribution of sentence lengths in Trip Advisor Hotel Reviews Dataset. This motivates us to implement a separate 1D convolution kernel for Theano which avoids zero-padding for variable length sequences.

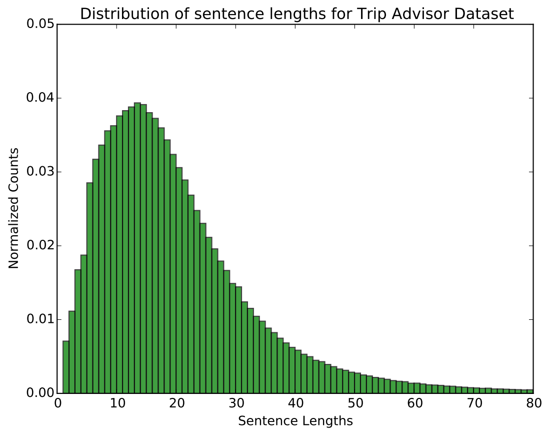

Figure 1: Distribution of sentence lengths in Trip Advisor Dataset

Implementation Details

Fig 2 shows Convolution Network used for modeling a document. One typically applied multiple convolution filters in each layer to extract different features. Convolution operation in each layer is called full convolution and is given by following formula,

$$OUT^n[i,k] = \sum_{j=max(0,i-w_k+1)}^{min(i,l_n-1)} \sum_{d} IN^n_{(j,d)} \circ KERN^k_{(w-1-i+j,d)}$$

Where n is the sentence index, i is the output location index, k is the kernel index, j is the running index of where convolution is applied, d is the index for dimension of the input. $OUT^n[i,k]$ indicates output value for n^{th} sentence at output location i and kernel k, $IN^n_{(j,d)}$ is the input value for n^{th} sentence at input location j and dimension d and $KERN^k_{(w-1-i+j,d)}$ is kernel value for k^{th} kernel at location $w-1-i+j$ and dimension d. w_k is width of k^{th} convolution kernel.

So the formula suggest that serial computation involves 5 loops one for each index n sentence index, i output index, k kernel index, j input index and d dimension index.

We choose to implement 1D convolutions using CUDA because Theano library uses CUDA as it's backed for 2D convolutions and many other operations and already has interfaces for integrating CUDA code. From here on we assume that the reader is familiar with CUDA API and it's terminology. An important design decision to be made before writing CUDA kernels is to decide what each block computes and what each thread computes.

Since n is the index for sentence and is the largest index, it's natural to assign it to a block. So each block computes convolutions on a given sentence. But now, each block has variable length of convolutions to compute which calls for some for load balancing in a mini-batch. CUDA run-time takes care of scheduling blocks such that resource requirements are met and each multi-processor is utilized to maximum extent. Hence, by assigning each variable length sequence to a block we have mitigated the load balancing to CUDA runtime. Within each block, for high performance CUDA requires that each thread performs same computation on different data points. Since in each dimension we are performing element-wise multiplication and summation, we assign each dimension to a thread.

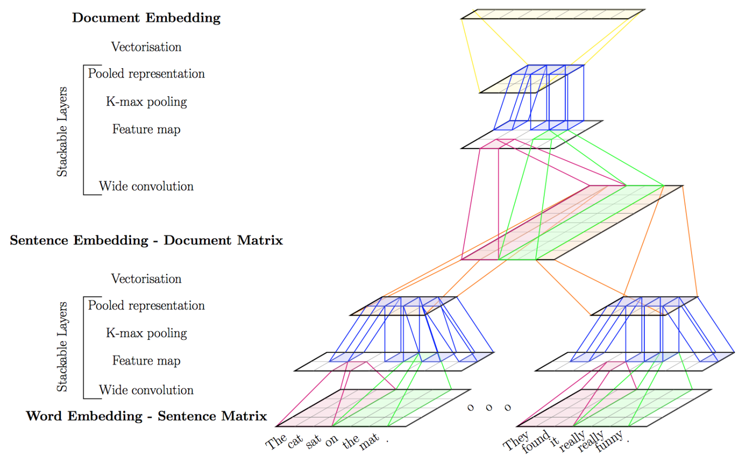

Figure 2: Convolution Network Architecture for Document Modeling. Picture courtesy: Denil et. al. [3]

Optimizations

1. **Minimum branch divergence**: Each thread in a given block computes the same element-wise product between input and kernels, the CUDA kernel has no branch divergence there. But computing the summation across dimensions ($\sum_d$) in the formula has some branch divergence but that is very minimal too.

2. **Shared memory summation**: For computing the summation across dimensions ($\sum_d$) in the formula, we store each individual element-wise products in a shared memory to save global memory bandwidth. So for each output location i and kernel k there is just 1 write operation to global memory.

3. **Global memory coalescing**: Inputs and Kernels are stored in dimension major order. Within the CUDA kernel, each thread in a block accesses consecutive dimensions and hence all the accesses to Inputs and Kernels are global memory coalesced.

Experimental Results

We ran all our run-time comparisons on NVIDIA GRID K520, which has CUDA compute capability of 3. We compare our implementation to Theano's implementation with zero-padding. Obviously, if all the sentences in a given mini-batch are of the same length then Theano's implementation outperforms our implementation because Theano's GPU code has been optimized for over 7 years now.

For comparing the performance we use the similar set of parameters used in the 1^{st} layer of Convolution Network used by Denil et. al. [3]. Firstly, we compare the performance for 1d convolution where dimension=100, convolution kernel width=7, mini-batch size=200, number of kernels=64 and batch size=100 with all sentence having the same length=50. We found that Theano's implementation was about **1.2x** faster than our implementation.

We use sentence lengths from Trip Advisor Dataset and initialize inputs and kernels randomly for all experiments. We use dimension=100, convolution kernel width=7, mini-batch size=200, number of kernels=64 and batch size=100 held fixed and change the parameter of interest. Fig 3 shows the performance of our implementation compared to Theano's implementation with zero padding as we vary mini-batch size parameter. As seen from Fig 3 as we increase the size of the mini-batch, speed up due to our implementation goes up as number of zero-padding increases and thus the amount of not useful computation in Theano's implementation. We want to use larger mini-batches so that convergence of training is smoother. Researcher typically use a mini-batch size of $[100, 500]$, in case of larger mini-batch size 500 we get upto **2.6x** speedup. As Fig 4 and Fig 5 performance is quite robust to number of kernels and width of kernels. An example of optimization done by Theano is shown in Fig 5 for width of the kernel = 9. For larger kernel size's Theano's implementation does some optimization to make it run faster. However, typical size of the kernels used by Denil et. al. [3] is $\{5, 7\}$ and Computer Vision Researchers have found that smaller kernel sizes and deeper networks work much better.

Finally, The kernel achieves CGMA (Compute to Global Memory Access ratio) of ≈ 30.12.

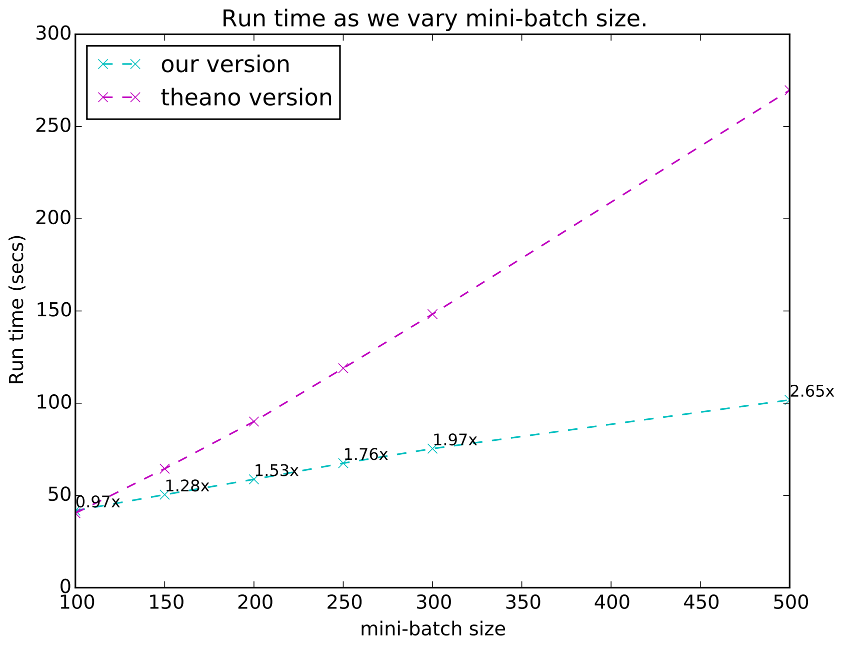

Figure 3: Performance of our implementation as we vary mini-batch size

Future Extensions

We plan to optimize our implementation even further so that we match the performance of Theano implementation on sentences of uniform lengths. Also, we need to integrate our implementation with Theano python API and release it for the benefit of larger research community.

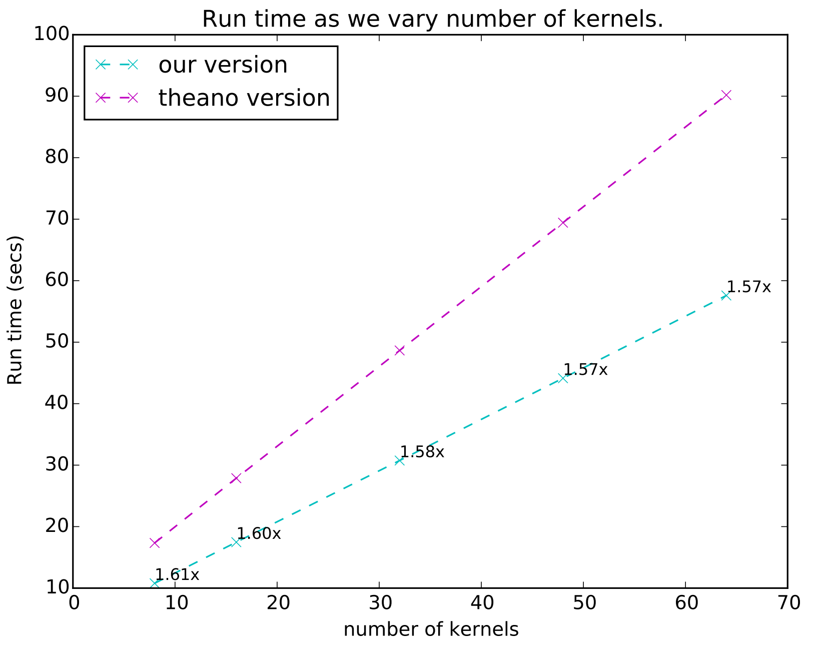

Figure 4: Performance of our implementation as we vary number of kernels

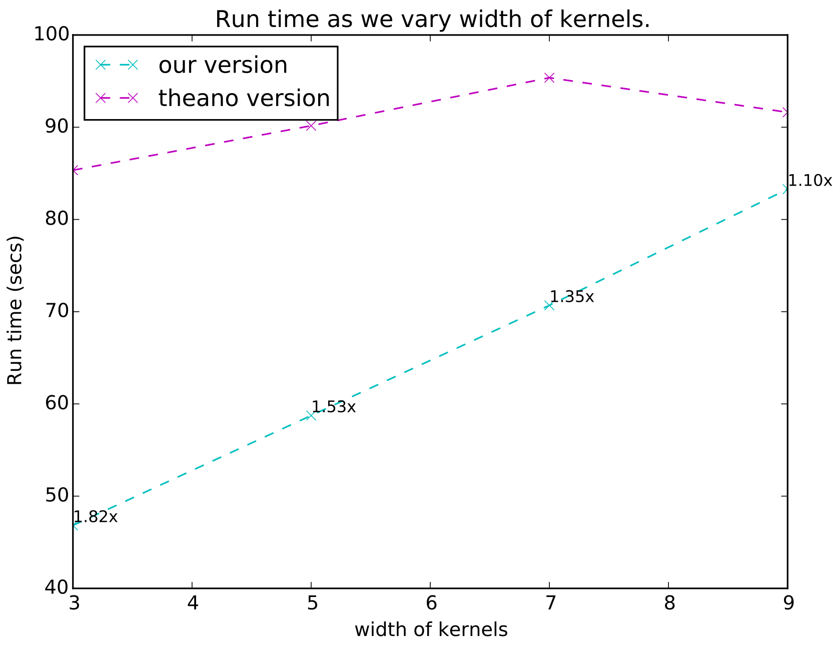

Figure 5: Performance of our implementation as we vary width of kernels

References

[1] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[2] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.

[3] Misha Denil, Alban Demiraj, and Nando de Freitas. Extraction of salient sentences from labelled documents. Technical report, University of Oxford, 2014.

[4] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, June 2014.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.