Improving the Performance of Fully Connected Neural Networks by Out-of-Place Matrix Transpose

Shaohuai Shi, Pengfei Xu, Xiaowen Chu
Department of Computer Science, Hong Kong Baptist University
\{csshshi, pengfeixu, chxw\}@comp.hkbu.edu.hk

Abstract—Fully connected network has been widely used in deep learning, and its computation efficiency is highly benefited from the matrix multiplication algorithm with cuBLAS on GPU. However, We found that, there exist some drawbacks of cuBLAS in calculating matrix A multiplies the transpose of matrix B (i.e., NT operation). To reduce the impact of NT operation by cuBLAS, we exploit the out-of-place transpose of matrix B to avoid using NT operation, and then we apply our method to Caffe, which is a popular deep learning tool. Our contribution is two-fold. First, we propose a naive method (TNN) and model-based method (MTNN) to increase the performance in calculating $A \times B^T$, and it achieves about 4.7 times performance enhancement in our tested cases on GTX1080 card. Second, we integrate MTNN method into Caffe to enhance the efficiency in training fully connected networks, which achieves about 70% speedup compared to the original Caffe in our configured fully connected networks on GTX1080 card.

Index Terms—Deep Neural Networks; Linear Algebra; Matrix Multiplication; Transpose; GPU;

1. Introduction

It is common to see fully connected network has been widely applied in deep learning \cite{1,2}. And the matrix multiplication operation plays an important role of a key mathematic tool in solving deep learning computations. The general form of matrix multiplication for two matrices $A$ and $B$, which are both row-major matrices, can be represented as follows:

\[ C = A \times B \]  

where $A \in R^{m \times k}$, $B \in R^{k \times n}$ and $C \in R^{m \times n}$. Assume that the element in row $i$ and column $j$ of a matrix $M$ is $M_{ij}$, we have:

\[ C_{ij} = \sum_{k=1}^{k} A_{ik} \times B_{kj} \]  

There is another form of matrix multiplication, which is $A$ times the transpose of $B$, i.e.,

\[ C = A \times B^T \]  

where $B^T$ is the transpose of $B$, $B^T_{ji} = B_{ij}$ and $B \in R^{n \times k}$.

From Equation \[2\] we can find that the complexity of matrix multiplication is $O(m \times k \times n)$, which is a very time-consuming computation on a serial algorithm. To do the computation of Equation \[3\] there are two general methods. The first one is that not to change the positions of elements of $B$, when it needs to access the column elements, it changes to access the corresponding row elements instead. The second is to transpose $B$ to $B^T$, and then do the matrix-matrix multiplication. In this paper, we focus on accelerating Equation \[3\].

Nowadays, there exists optimized algorithms to speedup the computation, including ATLAS, LAPACK, OpenBLAS, GotoBLAS, Intel MKL, Eigen, cuBLAS, clBLAS, etc., which focus on different types of hardware platforms. It is well known that GPUs play an important role in parallel computing. The cuBLAS running on NVIDIA GPU cards, achieves about 3000GFLOPS, which is up to 17x faster than that of MKL on Intel IvyBridge single socket 12-core E5-2697 v2 @ 2.70GHz, with single float matrix multiplication on NVIDIA K40M card \cite{3}.

However, there are some drawbacks of cuBLAS when do computation of Equation \[3\] which we call NT operation, compared to Equation \[2\] named NN operation.

In this paper, we test the performance GEMM of cuBLAS on both NN and NT operations, and then we figure out the drawbacks on some cases of cuBLAS. After that, we propose a solution to avoid these poor cases and achieve a better performance to work out Equation \[3\]. Lastly, we exploit our solution to a real-world application Caffe \cite{4} which is a popular deep learning framework and uses cuBLAS to accelerate its operations of matrix-matrix multiplication\footnote{Our source codes can be found here: http://github.com/caffe-optimized}. Our experiments show that our solution achieves up to 70% faster than original Caffe in some cases on the GPU of GTX1080.

The rest of the paper is organized as follows. We introduce the motivation in Section 2 and followed our methodology in Section 3. Experimental results are presented in Section 4. We conclude the paper and discuss our future work in Section 5.
2. Motivation

On deep neural networks, especially the fully connected networks [5], matrix-matrix multiplication is one of the main operations to do the training of networks. According to findings on fully connected networks in [6], we benchmark the performance of GEMM in NN and NT operations with a set of matrix configurations on the newest generation of NVIDIA GPU card (i.e., GTX 1080 with Pascal architecture) with CUDA-8.0. Table 1 shows our hardware setup for the experiments.

Table 1. The experimental GPU hardware with CUDA-8.0.

| GPU Model | Cores | Memory | OS       | Core frequency |
|-----------|-------|--------|----------|----------------|
| GTX1080   | 2560  | 8 GB   | Ubuntu 14.04 | 1607 MHz       |

The benchmark result is shown in Fig. 1. It can be noted that, in most cases, the performance of NN operation is much better than NT operation. For example, in the case of $C = A \times B^T$, where $A \in R^{1096 \times 53504}$ and $B \in R^{2048 \times 53504}$, the performance of NN operation is up to four times better than the NT operation on both GTX1080 card. Therefore, there exists some optimization space to speedup the calculation of Equation 3 Since the performance of NN operation is much better than NT operation, we first propose a naive method (TNN) which does the transpose of matrix $B$ first and then exploits NN operation to finish the calculation of Equation 3 However, the time used on transpose operation may be larger than the difference between NT and NN, which results in the performance of TNN operation could be worse than NT operation. To solve this problem, we propose an SVM-based model to decide to choose the faster operation in calculating Equation 3.

3. Methods

3.1. Naive Method

To calculate Equation 3, we replace the one-step NT operation by two-step TNN operation (i.e., transpose $B$ and do NN operation). This method requires extra GPU memory to store $B^T$ to perform out-of-place matrix transpose, and the size of memory required is the same with matrix $B$. The pseudo-code of TNN method is shown in Algorithm 1. Since TNN operation requires the additional transpose on GPU, the time used by transpose operation ($T_{\text{transpose}}(n,k)$) should not be larger than differential between NT ($T_{\text{NT}}(m,n,k)$) and NN ($T_{\text{NN}}(m,n,k)$). In other words, to guarantee $T_{\text{TN}}(m,n,k)$ is smaller than $T_{\text{NT}}(m,n,k)$, we have:

$$T_{\text{transpose}}(n,k) < T_{\text{NT}}(m,n,k) - T_{\text{NN}}(m,n,k) \tag{4}$$

However, the performance of transpose operation is decided by the hardware platform and the size of matrix. It is difficult to guarantee Equation 4 in practical if the difference between $T_{\text{NT}}(m,n,k)$ and $T_{\text{NN}}(m,n,k)$ is small enough.

Therefore, to perform faster calculation of Equation 3, we do not always use TNN instead of NT, which inspires the next method with SVM model.

Algorithm 1 TNN

1: procedure TNN(A, B, C, m, n, k)
2: // 1. Allocate GPU memory for transpose of B
3: BT = cudaMemAlloc(n*k*sizeof(float))
4: // 2. Tranpose B on GPU and store to BT
5: transposeOnGPU(B, n, k, BT)
6: // 3. Call gemm of cuBLAS with NN parameters
7: cublasSgemm(
8: cublasHandler,
9: CUBLAS_OP_N,
10: CUBLAS_OP_N,
11: A, lda,
12: BT, ldb,
13: C, ldc, ...)
14: // 4. Free GPU memory of BT
15: cudaFree(BT)

3.2. Model-based Method

According to the performance difference analysis between NT and TNN, we design a model-based optimized method (MTNN) to calculate Equation 3 From Fig. 1 there exist some cases that the performance of NT is extremely close to or even better than that of NN operation, the TNN method could not achieve better performance than NT, so we should not simply replace NT operation to TNN operation in real application. We propose a model to decide which one to choose to achieve better performance between NT and TNN by the size of matrices. Our method is straightforward, and it consists of two steps. First, we collect history data by testing the results of NT and TNN. Second, machine learning method is applied to train a decision model.

3.2.1. Data Collection

NT operation of cuBLAS is the API: “cublasSgemm” with the second and third parameters set to be “CUBLAS_OP_T” and “CUBLAS_OP_N” respectively, while TNN operation is doing the out-of-place transpose first by using “cublasSgemm”, and then calling “cublasSgemm” with both the second and third parameters set to be “CUBLAS_OP_N”.

We choose a range of matrices with sizes in $S = \{2^i | i = 7, 8, \ldots, 16\}$. In other words, for all $m, n$ and $k$ $(m \in S, n \in S, k \in S)$, which has 1000 combinations, we test the performances of NT and TNN in calculating Equation 3. Let $P_{\text{NT}}(m,n,k)$ and $P_{\text{TN}}(m,n,k)$ denote the performance of NT and TNN respectively with two matrices $A$ and $B$, where $A \in R^{m \times k}$ and $B \in R^{n \times k}$. The difference value between $P_{\text{NT}}(m,n,k)$ and $P_{\text{TN}}(m,n,k)$ is denoted by $D(m,n,k)$.

The collected data have 1000 records, and each record is with the following format:
\[ m, n, k, P_{NT}(m, n, k), P_{TNN}(m, n, k), D(m, n, k) \]

The cases that run out of memory are excluded, so the number of total samples is 891, and there are 655 samples with \( D < 0 \), and 236 samples with \( D > 0 \).

### 3.2.2. Decision Model

Given two matrix sizes (i.e., \( m, n, k \)) and two methods (i.e., NT and TNN) to calculate Equation 3, we should choose the one with higher performance operation. In formal, given input variables: \( (m, n, k) \), there is a decision function: \( f \), where

\[
    f(m, n, k) = \begin{cases} 
        -1, & P_{NT}(m, n, k) < P_{TNN}(m, n, k) \\
        +1, & P_{NT}(m, n, k) \geq P_{TNN}(m, n, k) 
    \end{cases}
\]

If \( f(m, n, k) = -1 \), then we choose TNN, otherwise we choose NT. Our target is to learn the decision function: \( f \) with the collected data, which is generalized as a classification problem. The training samples are denoted by \( X = \{(\log_2(m), \log_2(n), \log_2(k)) | (m, n, k) \in S\} \), and the corresponding labels are \( Y = \{f(m, n, k) | (m, n, k) \in S\} \). The SVM algorithm [2] is applied to solve this decision model. We use libsvm [8] with the kernel of radial basis function. Since the number of positive samples (236) is much smaller than that of negative one (655), the \(-wi\) parameter of libsvm is set to be 1.4 to solve the unbalanced-data problem. A 10-fold cross validation is used to test the performance of the model, 90 percentages of samples are used to be the training set, and the others are the testing set.

### 3.3. Integration with Caffe

To accelerate the training of deep learning by using MTNN method, the decision model is integrated into Caffe, and fully connected networks, the main operations of which are NT and NN, are tested on the integrated Caffe. When there is an operation of Equation 3 needs to be computed, the predictor can decide a faster version to perform the operation instead of calling NT of cuBLAS directly.

### 4. Results

We first present the results of matrix multiplication which compares the performances between original NT method by cuBLAS and TNN method, and then results of model-based method are also included for comparison. After that, the performance of our method in Caffe is shown.

#### 4.1. Results of TNN Method

Results of the original NT method and TNN method are shown in Fig. 2. It can be noticed that there are some cases that NT method still outperforms TNN method, especially when the value of \( K \) is small (e.g., there are up to half of the cases that NT method is better than TNN method when \( K = 128 \)). Among the tested cases, the maximum speedup of TNN method is up to 4.7 times compared to NT method. In the cases with better results of NT, the performance of NT is 1.67 times better than TNN at maximum. On average, TNN is 1.6 times better than NT.

#### 4.2. Results of MTNN

The data collected is splitted into ten parts on average, and 10-fold cross validation method is used to check the model. So among the 10 parts, each part is used to be a test set and the others to be a training set. The average precision of 10-fold cross validation is 95.1\%, which means that the decision model can make the calculation of Equation 4 fast enough in 95.1\% cases. After 10-fold cross validation, we put all the samples as the training set to generate the final decision model. The result compared to the original method is shown in Fig. 3. Compared to Fig. 2 most of red rectangles have been replaced by blue dashes. The statistic result is shown in Table 2 and Table 3. From Table 2 it is noted that the ratio of the cases that NT is better TNN decreases from 0.262 to 0.02 by MTNN method. In other words, our method can improve the performance in 98\% of the cases.
Figure 2. The performance comparison between NT method by cuBLAS and our TNN method in calculating Equation 3, where M is the height of matrix A, N is the height of matrix B, and K is the width of A and B. The rectangle symbol in the legend represents the performance of NT is better than TNN, and the cycle symbol green color represents the performance of NT is worse than TNN.

Figure 3. The performance comparison between NT method by cuBLAS and our model-based method in calculating Equation 5, where M is the height of matrix A, N is the height of matrix B, and K is the width of A and B. The rectangle symbol in the legend represents the performance of NT is better than TNN, and the cycle symbol green color represents the performance of NT is worse than TNN.

Table 2. Compare performance of NT to TNN and MTNN.

|                | Larger than | Smaller than | Equal to |
|----------------|-------------|--------------|----------|
| # of Cases     | 233         | 18           | 655      |
| Ratio          | 0.262       | 0.02         | 0.735    |
|                | 0.698       | 0.003        | 0.282    |

Note: the ratio is equal to the number of cases divided by the total number of valid samples.

Table 3. Performance speedup of TNN and MTNN compared to NT.

|         | Maximum speedup | Minimum speedup | Average speedup |
|---------|-----------------|-----------------|-----------------|
| TNN     | 4.7             | -1.66           | 1.59            |
| MTNN    | -1.14           | 1.66            |

Note: the negative speedup means that the performance of NT is better than TNN or MTNN.

4.3. FCN Results in Caffe

To test the performance of integrated Caffe, we choose two fully connected layers, which has 4096 neurons each layer, in AlexNet [2]. In addition, a smaller FCN network is tested to do the comparison. The mini-batch size varies from 128 to 2048. The performance comparison of FCN model running between three versions of Caffe, say original Caffe (CaffeNT), Caffe with TNN method (CaffeTNN) and Caffe with MTNN method (CaffeMTNN), is displayed in Fig. 4.

By integrating our method to Caffe, the performance of optimized Caffe can accomplish up to 70% speedup compared to the original Caffe on GTX1080 card. Compared to CaffeTNN, CaffeMTNN has slightly better performance, which is contributed by the decision model.

5. Conclusion and Future Work

Our method to multiply matrix A and the transpose of matrix B is much better than that of cuBLAS API. The original results achieve about 4.7x speedup compared to...
using cuBLAS directly on GTX1080 card. Furthermore, the method is applied to Caffe, and the optimized Caffe performs about 70% speedup on GTX1080 card.

The transposition algorithm we used is out-of-place method, which results in double memory of a matrix and it cannot run normally if there is no enough memory. Therefore, we plan to exploit in-place matrix transposition algorithm and to find a good trade-off between memory overhead and throughput.

References

[1] Y. LeCun et al., “Lenet-5, convolutional neural networks,” URL: http://yann.lecun.com/exdb/lenet, 2015.

[2] 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.

[3] NVIDIA, “cublas — nvidia,” https://developer.nvidia.com/cublas, 2016, accessed: 2016-09-6.

[4] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, “Caffe: Convolutional architecture for fast feature embedding,” in Proceedings of the 22nd ACM international conference on Multimedia, 2014, pp. 675–678.

[5] R. Hecht-Nielsen, “Theory of the backpropagation neural network,” in Neural Networks, 1989. IJCNN., International Joint Conference on. IEEE, 1989, pp. 593–605.

[6] S. Shi, Q. Wang, P. Xu, and X. Chu, “Benchmarking state-of-the-art deep learning software tools,” arXiv preprint arXiv:1608.07249, 2016.

[7] J. A. Suykens and J. Vandewalle, “Least squares support vector machine classifiers,” Neural processing letters, vol. 9, no. 3, pp. 293–300, 1999.

[8] C.-C. Chang and C.-J. Lin, “LIBSVM: A library for support vector machines,” ACM Transactions on Intelligent Systems and Technology, vol. 2, pp. 27:1–27:27, 2011, software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm