Weight Matrices Sharing in High Utilization Pipeline Architecture Design for Neuromorphic Computing

Dong Yang, Group 20

I. Introduction

Machine learning proves to be a huge success in the last few years. Neural networks plays an important part in the development of the machine learning. In general, a deeper neural network can have a better performance while it costs much more computation resource both in training and evaluation. Someone has proposed the pipeline architecture design for neuromorphic computing by using resistive random access memory. In this project, we focus mainly on the evaluation of the neural networks and introduce a simple method to improve the utilization by sharing the weight matrices in the neural networks.

II. Background

A. Convolution Neural Network

In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery. Just like ordinary deep neural network, a CNN consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of convolutional layers (use RELU as the activation function), pooling layers, fully connected layers and normalization layers.

The performance of some CNNs is unprecedented in image recognition. However those networks are memory-intensive and computation-intensive.

B. Resistive Random Access Memory(ReRAM)

Resistive random-access memory (ReRAM) is a type of non-volatile (NV) random-access (RAM) computer memory that works by changing the resistance across a dielectric solid-state material, often referred to as a memristor. It can combine storage and computation. In other words, it is able to do computation while storing the data in the memroy. However it takes a lot to write the ReRAM compared with reading.

III. Related Works

Someone has developed the method to accelarated CNNs by using the ReRAM. Because the feature of ReRAM mentioned in the above section is just corresponding to the challenge of CNNs that they are memory-intensive and computation-intensive.

IV. Motivation

Because the convolutional neural networks include several sets of convolutional layers and pooling layers. And there would be bubbles in the pipline architecture. A weight matrix in the neural networks can be regarded to be associated to a block of hardware. As mentioned in the related works section, we know that it costs a lot to write the hardware. Therefore, if one block of hardware can be used among layers of neural networks, the utilization of the hardware can be definitely increased.

V. Problem Formulation

Given a CNN with several layers of weight matrices. Make the similar weight matrices from different layers share the excatly the same value such that a block of hardware can be utilized among different layers rather than only once.

In this project, we mainly focus on the change of the accuracy of the neural network after the modulation. The utilization is definitely increased if we dispatch the hardware properly.

VI. Proposed Method

In this project, we mainly focus on the convolution layers.

A. Find Similar Weight Matrices

Firstly, we consider how to define the similarity of two weight matrices. We just use hamming distance to measure the difference between two matrices. If two matrices are identical, the hamming distance is 0 without any doubt. However, it's nearly impossible to find two identical matrices in a neural network. We set a threshold according to the size of the kernel filter to determine whether two matrices are similar.

There is no problem if we compare the matrices directly. But intuitively, we want to just compare the feature of the matrices rather than every value in the matrices to reduce complexity. One idea is to calculate some feature of the matrix, which can be scalar or vector. But feature with too low dimension cannot represent the original matrix effectively. Or some features such as diagonal of SVD or eigenvalues costs too much to calculate. We decide to just use part of values in the matrix as the feature and round them for convenience. For example, if the kernle filter size is 3 by 3, we take out the values in four corner and the center. This idea comes from the fact that CNN is a simulation of human visual recogniton. A kernel filter is used to extract the feature of some part of the object. Two similar kernel filter means they extract the feature in a similar way. As the distribution of the values in the kernel filter is uniform, if more than half values in the two kernel filters are the same, it is more likely they extract information in a similar way. Just image how our eyes capture the information of a object. Even if we lose some pixels (uniformly distributed in the view), we can still recognize the object in all probability.

Secondly, we consider how to find the similar weight matrices. Finding similar ones globally can be time consuming. So we try to find similar ones between two adjacent layers.

One solution is greedy, for each weight matrix in $layer_i$, find the most similar one in $layer_{i+1}$.

The other one is that for each weight matrix in $layer_i$, find all similar candidates in $layer_{i+1}$ with loss as cost. Regard each involved weight matrix as a node to build a graph. Add s, t nodes before $layer_i$ and after $layer_{i+1}$ respectively such that s connects all nodes in $layer_i$ with capacity 1, cost 0, all available connects between two layers have capacity 1 and t connects all nodes in $layer_{i+1}$. Then applay mincost max flow algorithm to find a proper scheme.

B. Construct Shared Weight Matrices

A large portion of kernel filters have the same size. We can use a set of basis to represent the original kernel. Those basis weight matrices can be used among differrent layers. And coefficient matrices (a scalar in fact) can be compared more conveniently such that we can find similar coefficient matrices more easily.

For instance, the input feature is a $H \times W \times C$. Take $C = 1$ for convenience. We have a set of kernel filters $N \times k \times k \times C$ i.e. N kernel filters whose shape is k by k. Generate a set of basis $\{B_i | 1 \leq i \leq m\}$. And calculate coefficients $\{a_i | 1 \leq i \leq N, dim(a_i) = m\}$ for $\{W_i | 1 \leq i \leq N\}$.The weight matrix

$$W_i = \sum_{j=1}^{m} a_{i,j} \cdot B_j$$

Then the output should be

$$Output_i = Input * W_i = \sum_{j=1}^{m} a_{i,j} \cdot (Input * B_j)$$

If we want the output be accurate, then let $m = k^2$. However, the original time complexity is $\mathcal{O}(Nk^2HW)$, afterwards it becomes $\mathcal{O}(mk^2HW + NmHW)$, as one layer is separated into two layers. If we don't want to increase the time complexity, we should consider the value of m. At least, m should be less than k^2. But in this way, we can only use the linear combination of the basis to approximately represents the orignal kernel filter.

Intuitively, we use gradient descent method to find the coefficients rather than just use the inverse of matrix. Firstly, we can define the loss function

$$loss_i = \|\sum_{j=1}^{m} a_{i,j} \cdot B_j - W_i\|_2^2$$

And the derivative is

$$\frac{\partial loss_i}{\partial a_{i,j}} = \|2(\sum_{j=1}^{m} a_{i,j} \cdot B_j - W_i) * B_j\|_1$$

VII. Experiments

The experiments are conducted in a small CNN for image recognition. The model consists of two convulutional layers, a maxpooling layer, two dropout layers and two dense layers. The kernel filters' size is 3 by 3. The dataset is standard preprocessed mnist dataset. The output is the class of the input image. For the pretrained neural network, the accuracy can obtain about 99.20%.

Every small experiments has been conducted several times and get an average result to exclude accidents.

A. Find Similar Weight Matrices

We had a test of greedy solution among these two convolution layers. The application of mincost maxflow algorithm can definitely increase the performance compared with this greedy one.

For each weight matrix in the first layer we find the most similar one in the second layer and do the replacement. But for those who cannot find a similar one, we didn't touch them. The following shows some results of various selection.

- Select the diagonal of the weight matrix. The accuracy drops by more than 50%.
- Select four adjacent position of the weight matrix ([[0,0], [0,1], [0,2], [1,0]]). The accuracy drops by 0.39% on average.
- Select the four corner of the weight matrix. The accuracy drops by 0.12% on average.
- Select five adjacent position of the weight matrix ([[0,0], [0,1], [0,2], [1,0], [2,0]). The accuracy drops 0.09% on average.
- Select the four corner and the center of the weight matrix. The accuracy drops 0.02% on average. And sometimes the accuracy could even increase.

From above results, we can find the selection of the part of the weight matrix is critical to the replacement. The more positions we select, the less the accuracy drops, but the harder it is to find a similar weight matrix as well. Even if the number of values we select is the same, the position distribution makes a difference. Uniform distribution of the position is helpful to maintain the accuracy.

B. Construct Shared Weight Matrices

We had a test on the first convolution layer. And the basis is randomly generated. So for each kernel filter, we have a set of coefficients to

Firstly, we test basis with 9 base matrices. And the results shows that the accuracy is just the same as before.

Secondly, we tried different number of the basis and got the following results

- 8 basis. The accuracy drops 0.28% on average.
- 7 basis. The accuracy drops 2.94% on average. And the drop fluctuate dramatically raging from less than 1% to around 9%.
- 6 basis. The accuracy drops 23.7% on average. And the drop fluctuate more dramatically raging from 5% to more than 50%.

From the above, we can infer that the number of basis to represent the original should be enough. For a 3 by 3 kernel filter, 8 basis is acceptable and 7 is a candidate, but 6 is far away from the goal. Let aside those with less than 6 basis.

VIII. CONCLUSIONS

Neural networks are tolerant with some modulation. Because there is so much redundent information in the neural network that it allows us to do some incorrect or improper change. But we should try our best to optimize the neural networks without bringing defect.

When we try to find the similar matrix in the neural networks, although we can find the most similar ones by comparing the whole weight matrix, the number of pair we find could be small. And the utilization shall improves only slightly. If we relax some requirements, just compare partial of weight matrix, we can find more pairs of similar weight matrix while keeping the accuracy in a reasonable range.

Constructing some basis as shared weight matrices is also acceptable. But it requires sufficient basis if we want to maintain the accuracy. The utilization of the hardware is improved. But the time complexity of computation is decreased only when $Nk^2 \leq mk^2 + mN$. Luckily, in most cases, N is far larger than k^2 and k^2 is larger than m. So $m = k^2 - 1$ or $m = k^2 - 2$ is enough. And these m's can keep the accuracy in a reasonable range.

In the second method, we just introduce the reuse of the basis. If we want to make the utilization rate higher, similar coefficients could be set the same. As for this part, we can turn to the first method mentioned above. Futhermore, the coefficient matrix is of 1 by 1 size. We can find similar ones more easily. That can improve the utilization further.