Estimation of convolutional neural net forward pass time on GPU

A.V. Dushatskiy, D.Y. Buryak

Moscow State University
Faculty of Computational Mathematics and Cybernetics
Department of Supercomputers and Quantum Informatics
Problem statement

- It is given a fixed GPU device (in the experiments Nvidia Tesla K40c is used)
- A convolutional neural net architecture description is given in special format (JSON)
- Goal: estimate convolutional neural network running time without implementing the neural net
Formal problem statement

Input of estimation model:

- Neural net architecture $A$
- Input data dimensions $D$ (e.g. 32x32 pixels, in RGB)
- Input data batch size $B$ (batch of images is processed in parallel)

Neural net running time is $T$

Task: model/learn dependency $f(A,D,B)\rightarrow T$ for particular GPU device
Neural net running time is understood as:

- Inference time (forward pass)
- Time for transferring data from host to device and from device to host is not taken into account
Solution usage scenario

1. A researcher / engineer runs the estimating model building script (once) for a target GPU.

2. Then he/she can estimate running time of different neural nets without explicit implementation, debugging and time measurement.

3. This technique can significantly speed up the search for optimal neural net architecture as the neural nets with estimated time more than task limit can be dropped without implementation.
Approach novelty

• The majority of current solutions for parallel programs running time on GPU estimation are based on analytical models and are not suitable for such complex calculations as deep convolutional neural nets

• Several works devoted to parallel programs running time estimation using machine learning are focused on applications to estimating running time of numerical methods (for solving differential equations) on supercomputers
Supervised machine learning is used to estimate the running time of single layer. Target value of models: layer running time. The overall running time estimation is sum of each layer running time estimations.

- Separate models are built for convolutional, pooling, activation layers.
- Fully-connected layers are considered as special case of convolutional ones
- Used machine learning models: decision tree, multilayer perceptron
- Models features: calculated values based on layer parameters and input dimensions, for convolutional layers: multiplied matrices sizes
Approach details

• The target metric is Mean Absolute Percentage Error (MAPE):

\[ MAPE = \frac{100}{N} \sum_{i=1}^{N} \left| \frac{y - \hat{y}}{y} \right|, \text{y is true value, } \hat{y} \text{ is predicted value} \]

• Minimized loss for machine learning models training is Mean Squared Error (MSE):

\[ MSE = \frac{1}{N} \sum_{i=1}^{N} (y - \hat{y})^2, \text{y is true value, } \hat{y} \text{ is predicted value} \]

• For accuracy improving the training dataset is modified: the objects are multiplied number of times, inversely proportional to the value of object’s target value; thus, increasing the impact of objects with small target values to MSE
• Neural nets operations are implemented with cuDNN library (C++) to avoid overhead effects which are induced by Python frameworks

• Machine learning methods and results visualization are implemented with Python (Scikit-Learn, Numpy, Matplotlib)
Neural net architectures

- Considered neural nets architectures: convolutional neural nets applicable for image classification tasks, similar to AlexNet architecture
- Possible layers are convolutional, pooling, fully-connected, possible activation functions in experiments are ReLU and Softmax
  (but generally it’s not a limitation as activations consume little time)
- Neural net structure is feed-forward
- Number of color channels of images and images dimensions are not restricted
Experiments

The proposed approach was tested on 100 generated neural net architectures, similar to AlexNet:

• There is a block of convolution layers with insertions of max-pooling layers
• Activation functions after convolution layers are ReLU
• The final two layers of the neural net are fully-connected, the last one has SoftMax activation

All tests are performed for cifar-10 dataset: classification task into 10 classes, 32x32 RGB images; GPU device is Tesla K40c
Comparison of ML models

In the comparison of decision trees and multilayer perceptron for separate layers time estimation the decision trees demonstrated better performance on validation datasets:
## Results

Results of testing on 100 generated neural net architectures:

<table>
<thead>
<tr>
<th>Batch size</th>
<th>Convolution, MAPE</th>
<th>Pooling, MAPE</th>
<th>Activation, MAPE</th>
<th>Full net, MAPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.9</td>
<td>12.2</td>
<td>12.1</td>
<td>6.2</td>
</tr>
<tr>
<td>4</td>
<td>4.2</td>
<td>11.0</td>
<td>12.5</td>
<td>6.4</td>
</tr>
<tr>
<td>16</td>
<td>6.8</td>
<td>16.3</td>
<td>10.9</td>
<td>9.5</td>
</tr>
</tbody>
</table>
Results

- Overall estimation accuracy (MAPE < 10%) provides an opportunity for practical usage.

- Convolution running time estimation accuracy is significantly lower than for other layers types and provides the overall good accuracy. Probably it can be explained by obvious convolution representation as matrix-matrix multiplication.

- Estimation accuracy for batch size 16 is significantly lower than for smaller batch sizes. Probably because of sophisticated optimizations in cuDNN for large batches.
Conclusion

• A novel approach for estimating neural nets running time on GPU is proposed

• Experiments on generated neural nets architectures applicable to image classification tasks are conducted

• Approach accuracy gives an opportunity for practical usage
Thank you for your attention!