Deep Learning Workshop II 2018

Organizing Committee:
Maggi Zhu
Aly El Gamal
Stanley Chan
Charles Bouman
Greg Buzzard
Dong Hye Ye
Amir Ziabari
Sri Yarlagada
Diyu Yang
Part 1:
GPU Implementation of Deep Neural Networks Using Tensorflow and Keras

Amirkoushyar Ziabari

aziabari@purdue.edu

Aug 08 2018
Outline

- Deep Learning Frameworks
- Tensors
- What is Tensorflow (TF)?
- Tensorflow Structure
- CNN to classify MNIST
- GPU implementation
- Keras
# Deep Learning Frameworks

<table>
<thead>
<tr>
<th>Framework</th>
<th>Distributed Execution</th>
<th>Architecture Optimization</th>
<th>Visualization</th>
<th>Community Support</th>
<th>Portability</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tensorflow</td>
<td>✔ ✔ ✔</td>
<td>✔ ✔ ✔</td>
<td>✔ ✔ ✔</td>
<td>✔ ✔ ✔</td>
<td>✔ ✔</td>
<td>• Google, • wide usage, ecosystem and community support • Visualization is superior</td>
</tr>
<tr>
<td>Pytorch</td>
<td>✔ ✔ ✔</td>
<td>✔ ✔ ✔</td>
<td>✔ ✔ ✔</td>
<td>✔ ✔ ✔</td>
<td>✔ ✔</td>
<td>• Facebook • Easy to use &amp; technically impressive</td>
</tr>
<tr>
<td>CNTK</td>
<td>✔ ✔ ✔</td>
<td>✔ ✔ ✔</td>
<td>✔ ✔ ✔</td>
<td>-</td>
<td>✔ ✔</td>
<td>• Microsoft • Licensig issues</td>
</tr>
<tr>
<td>MXNet</td>
<td>✔ ✔ ✔</td>
<td>✔ ✔ ✔</td>
<td>✔ ✔ ✔</td>
<td>-</td>
<td>✔ ✔</td>
<td>• Behind other frameworks in DE, and lacks details</td>
</tr>
<tr>
<td>Caffe2</td>
<td>✔ ✔ ✔</td>
<td>✔ ✔ ✔</td>
<td>✔ ✔ ✔</td>
<td>-</td>
<td>✔ ✔</td>
<td>• Facebook • Still moving</td>
</tr>
<tr>
<td>Caffe</td>
<td>-</td>
<td>✔ ✔ ✔</td>
<td>✔ ✔ ✔</td>
<td>✔ ✔</td>
<td>✔ ✔</td>
<td>• Facebook • Lacks future community support</td>
</tr>
<tr>
<td>Theano</td>
<td>✔ ✔ ✔</td>
<td>✔ ✔ ✔</td>
<td>✔ ✔ ✔</td>
<td>✔ ✔</td>
<td>✔ ✔</td>
<td>• University of Montreal • Lacks future community support</td>
</tr>
</tbody>
</table>

- Tensors are the standard way of representing data in Tensorflow (deep learning)
- Tensors are multidimensional arrays, an extension of matrices to data with higher dimensions
## Tensor Rank

<table>
<thead>
<tr>
<th>Rank</th>
<th>Math Entity</th>
<th>Python Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Scalar (magnitude only)</td>
<td><code>s = 483</code></td>
</tr>
<tr>
<td>1</td>
<td>Vector (magnitude and direction)</td>
<td><code>v = [1.1, 2.2, 3.3]</code></td>
</tr>
<tr>
<td>2</td>
<td>Matrix (table of numbers)</td>
<td><code>m = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]</code></td>
</tr>
<tr>
<td>3</td>
<td>3-Tensor (cube of numbers)</td>
<td><code>t = [[[2], [4], [6]], [[8], [10], [12]], [[14], [16], [18]]]</code></td>
</tr>
<tr>
<td>n</td>
<td>n-Tensor (you get the idea)</td>
<td><code>...</code></td>
</tr>
</tbody>
</table>
In addition to dimensionality, Tensors have different data types:

<table>
<thead>
<tr>
<th>Data type</th>
<th>Python type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DT_FLOAT</td>
<td>tf.float32</td>
<td>32 bits floating point.</td>
</tr>
<tr>
<td>DT_DOUBLE</td>
<td>tf.float64</td>
<td>64 bits floating point.</td>
</tr>
<tr>
<td>DT_INT8</td>
<td>tf.int8</td>
<td>8 bits signed integer.</td>
</tr>
<tr>
<td>DT_INT16</td>
<td>tf.int16</td>
<td>16 bits signed integer.</td>
</tr>
<tr>
<td>DT_INT32</td>
<td>tf.int32</td>
<td>32 bits signed integer.</td>
</tr>
<tr>
<td>DT_INT64</td>
<td>tf.int64</td>
<td>64 bits signed integer.</td>
</tr>
<tr>
<td>DT_UINT8</td>
<td>tf.uint8</td>
<td>8 bits unsigned integer.</td>
</tr>
<tr>
<td>DT_STRING</td>
<td>tf.string</td>
<td>Variable length byte arrays. Each element of a tensor is a byte array.</td>
</tr>
<tr>
<td>DT_BOOL</td>
<td>tf.bool</td>
<td>Boolean.</td>
</tr>
</tbody>
</table>
What is Tensorflow?

- Tensorflow (TF): a python library to implement deep networks
  - Very simple to install on all operating systems (tensorflow.org/install)
  - Pycharm, ipython, etc can be used to run TF on windows
  - In Tensorflow, computation is approached as a dataflow graph

\[
\begin{array}{cccc}
2.9 & 1.3 & -2.3 & \ldots \\
-1.0 & 4.1 & 3 & \ldots \\
\ldots & \ldots & \ldots & \ldots \\
\ldots & \ldots & \ldots & \ldots \\
\end{array}
\]

\[
X, W, b: \text{Tensors} \\
\text{Relu, Add, Matmul: functions}
\]
TensorFlow Structure

TensorFlow core programs consist of two discrete sections:

- Building a computational graph
- Running a computational graph

A computational graph is a series of TensorFlow operations arranged into a graph of nodes.

```python
import tensorflow as tf

a = tf.constant(5.0, tf.float32)
b = tf.constant(6.0)
c = a*b
```

Build a computational graph

Run the computational graph
Goal

- Develop a Convolutional Neural Network To Classify MNIST DATA
Training Process

1. **Start**
2. **Read the Dataset**
3. **Define features and labels**
4. **Encode The Dependent variable**
   - **Pre-processing of dataset**
5. **Divide the dataset into two parts for training and testing**
   - **TensorFlow data structure for holding features, labels etc..**
6. **Train the model**
7. **Implement the model**
8. **Repeat the process to decrease the loss**
   - **Reduce MSE (actual output - desired output)**
9. **Make prediction on the test data**
10. **End**
Code Structure

0. Import the necessary libraries
1. Read the input data; define parameters, constants..
2. Define input/target size, type
   Assign space for input/target
3. Define weights and biases
4. Define and construct the model
   (e.g. Convolutional Neural Net)
5. Define loss function
6. Choose optimization technique
7. Define Training operation
8. Define Initialization operation
9. Define events logs and saving operations
10. Define Session and run initialization
11. Train the model (run the training op.)
   Print outputs, Save (or restore) model and events logs
MNIST Dataset Overview

This example is using MNIST handwritten digits. The dataset contains 60,000 examples for training and 10,000 examples for testing. The digits have been size-normalized and centered in a fixed-size image (28x28 pixels) with values from 0 to 1. For simplicity, each image has been flattened and converted to a 1-D numpy array of 784 features (28^2).

CNN Network

Convolution
Kernel = 5x5
Stride = (1,1)
Filters = 32

Max Pooling
Kernel = 2x2
Stride = (2,2)

Conv1 feature maps
(28x28) x 32

pool1 feature maps
(14x14) x 32

Convolution
Kernel = 5x5
Stride = (1,1)
Filters = 64

Max Pooling
Kernel = 2x2
Stride = (2,2)

Conv2 feature maps
(14x14) x 64

pool2 feature maps
(7x7) x 64

FC1:1024

Full connection

Dropout
And Fully Connected

OUT: 10
Review the code
Supported devices

On a typical system, there are multiple computing devices. In TensorFlow, the supported device types are CPU and GPU. They are represented as strings. For example:

- "/cpu:0": The CPU of your machine.
- "/device:GPU:0": The GPU of your machine, if you have one.
- "/device:GPU:1": The second GPU of your machine, etc.

If a TensorFlow operation has both CPU and GPU implementations, the GPU devices will be given priority when the operation is assigned to a device. For example, matmul has both CPU and GPU kernels. On a system with devices cpu:0 and gpu:0, gpu:0 will be selected to run matmul.
Manual Device Placement

1. To perform particular operation on a device of your choice

2. with tf.device ()
   1. Creates a context such that all operations with the context will have the same device assignment
Allowing GPU memory growth

- By default, TF maps nearly all of the GPU memory of all GPUs visible to the process.

- In some cases it is desirable for the process to only allocate a subset of the available memory, or to only grow the memory usage as it is needed by the process.

```python
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
session = tf.Session(config=config, ...)
```
If you would like TensorFlow to automatically choose an existing and supported device to run the operations in case the specified one doesn't exist, you can set `allow_soft_placement` to `True` in the configuration option when creating the session.
Websites

- https://www.tensorflow.org/api_docs/python/tf/ConfigProto
- https://www.tensorflow.org/guide/using_gpu
- https://www.tensorflow.org/api_docs/python/tf/device
Assignment 1

- Change code 1 to run on a gpu
  - Set allow growth and soft placement to True
  - Set the graph to be on the cpu, while optimization and loss calculation to be on the gpu
  - If you confront memory error on google colab when calculating the test accuracy, then please write the code to input the test data in batches
What is Keras?

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.

Why Keras?

- User friendly
- Modularity
- Easy Extensibility
- Work with python
Keras Structure

0. Import the necessary libraries

1. Read the input data; define parameters, constants..

2. Define model structure. (**Sequential** model: linear stack of layers, or more complicated **Keras functional API** user defined model)

3. Add layers of the model

4. Compile the model

5. Fit the model

6. Evaluate the model
https://keras.io/
Let’s convert our TF code to Keras
Show how to run Keras on GPU and Multi-GPUs
Assignment 2

- Convert the tensorflow code for AlexNet into Keras
Backup
• **training**: Either a Python boolean, or a TensorFlow boolean scalar tensor (e.g. a placeholder). Whether to return the output in training mode (normalized with statistics of the current batch) or in inference mode (normalized with moving statistics). **NOTE**: make sure to set this parameter correctly, or else your training/inference will not work properly.
Batch Norm at test time

- **Training (mini-batch)**
  \[ m \sum_i z^{(i)} \]
  \[ m \sum_i (z^{(i)})^2 \]

- **Testing (one-example)**
  \[ z_{\text{norm}}^{(i)} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}} \]
  \[ \tilde{z}^{(i)} = \gamma z_{\text{norm}}^{(i)} + \beta \]
Batch Normalization

**Input:** Values of $x$ over a mini-batch: $\mathcal{B} = \{x_1...m\}$; Parameters to be learned: $\gamma$, $\beta$

**Output:** $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

$$
\mu_\mathcal{B} \leftarrow \frac{1}{m} \sum_{i=1}^{m} x_i \quad \text{// mini-batch mean}
$$

$$
\sigma^2_\mathcal{B} \leftarrow \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2 \quad \text{// mini-batch variance}
$$

$$
\hat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma^2_\mathcal{B} + \epsilon}} \quad \text{// normalize}
$$

$$
y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \quad \text{// scale and shift}
$$

**Algorithm 1:** Batch Normalizing Transform, applied to activation $x$ over a mini-batch.