Chainer: a Next-Generation Open Source Framework for Deep Learning

CSE 5449

Tu Tran
Outline

1. Introduction
2. Existing deep learning frameworks
3. Chainer
4. Benchmark
5. Supporting Chainer running on Intel Xeon Phi
Introduction (1)

• Around 2015
  – Deep learning became very successful in computer vision and speech recognition
  – Existing frameworks designed for such purposes
    • Feed-forward networks like CNNs, which is good for data samples of fixed length, such as images.
• Recently, many novel types of models other than CNNs received interests from the communities
  – For example: recurrent neural networks (RNNs) shows promising results on variable-length data such as natural language text
Introduction (2)

• Problem: Can new deep learning models be efficiently implemented using existing frameworks?
  – Implementation of new models is difficult
  – They were designed for image processing using CNNs
  – Most of them use a domain specific language.
    • Debugging will be difficult

• These motivate the creation of Chainer,
  – open source framework
  – provides simple and efficient methods for implementing complex algorithms, training models, and tuning model parameters.
Existing deep learning frameworks

- Use a paradigm called **Define-and-Run**
- Define phase:
  - The model defined a specific markup language, such as Protobuf for Caffe or YAML for PyLearn2
  - The framework will interpret the model, build a graph and load it on memory
- Run phase:
  - receives inputs (data),
  - processes these inputs (forward/backward computation)
  - changes the models internal state (updating)
  - and outputs the results (predictions).
Drawbacks (1)

• Although Define-and-run works well for CNNs, there are three major problems when implementing other types of NN models.
  1. Inefficient memory usage
     – Before training, the entire graph must be loaded into memory
     – Some layers needed only at the beginning or end of the training process
     – Example: Recurrent Neural Networks (RNNs)
     – Even if it is a CNN, this paradigm is still problematic
Drawbacks (2)

2. Limited extensibility
   – Theses framework are model-specific
   – It is difficult to maintain backward compatibility and extend the Define-and-Run scheme to accommodate more complex models.
   – For example, many forked versions of Caffe exist for implementing specific algorithms, but these versions are mutually incompatible and cannot be merged

3. The network is a black box
   – The inner workings of the network are not accessible to the user
   – What if you want to debug and tune a model effectively? A user needs to be able to see what is happening inside the model.
Chainer (1)

- A second-generation deep learning framework
- Based on a novel paradigm called **Define-by-Run**
  - Implement more complex deep learning architectures easily
  - Written in Python due to the simplicity, popularity, and benefits of using the existing multi-dimensional array library (NumPy)
- Support running on GPU with CuPy
Chainer (2)

# (1) Function Set definition
model = FunctionSet(
    l1=F.Linear(784, 100),
    l2=F.Linear(100, 100),
    l3=F.Linear(100, 10)).to_gpu()

# (2) Optimizer Setup
opt = optimizers.SGD() opt.setup(model)

# (3) Forward computation
def forward(x, t):
    h1 = F.relu(model.l1(x))
    h2 = F.relu(model.l2(h1))
    y = model.l3(h2)
    return F.softmax_cross_entropy(y, t)

# (4) Training loop
for epoch in xrange(n_epoch):
    for i in xrange(0, N, b_size):
        x = Variable(to_gpu(...))
        t = Variable(to_gpu(...))
        opt.zero_grads()
        loss = forward(x, t)
        loss.backward()
        opt.update()

Example Multi-layer perceptron
**Benchmark**

- Chainer is compared with Caffe using various CNNs

Table 1: Mean time for computations on various networks (msec).

<table>
<thead>
<tr>
<th></th>
<th>Basic convolutional nets</th>
<th>ImageNet</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>conv1</td>
<td>conv2</td>
</tr>
<tr>
<td><strong>batchsize</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>forward</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chainer</td>
<td>1st</td>
<td>245.62</td>
</tr>
<tr>
<td>Chainer</td>
<td>2-11th</td>
<td>72.31</td>
</tr>
<tr>
<td>Caffe</td>
<td>1-10th</td>
<td>47.14</td>
</tr>
<tr>
<td><strong>backward</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chainer</td>
<td>1st</td>
<td>103.01</td>
</tr>
<tr>
<td>Chainer</td>
<td>2-11th</td>
<td>85.48</td>
</tr>
<tr>
<td>Caffe</td>
<td>1-10th</td>
<td>101.44</td>
</tr>
</tbody>
</table>
Supporting Chainer running on Intel Xeon Phi
pyMIC-DL: A Library for Deep Learning Frameworks Run on the Intel Xeon Phi Coprocessor

Tu Tran, Huu-Phu Nguyen, Minh Tri Nguyen, Thanh-Dang Diep, Nguyen Quang-Hung and Nam Thoai
SUPERCOMPUTER

- Milky Way 2
- Thunder
- cascade
- SuperMUC
- …

Equipping IXPKC
61 cores, 512-bit vector registers

XPKL: Xeon Phi Knights Landing (2013)
XPKC: Xeon Phi Knights Corner (2012)
- Facilitate deep learning applications run on Intel Xeon Phi Knights Corner.
- Exploit the computing power of legacy systems containing Intel Xeon Phi Knights Corner such as Tianhe-2 (Milky Way 2), Thunder, cascade, SuperMUC, SuperNode-XP (HCMUT),… for Deep Learning computation.
In Deep Learning, all computations are performed on ARRAY
pyMIC-DL: NumPy-like Library
Main Features:

- An open source Python module.
- Help to connect C/C++ code and Python code so that Python applications can run on Xeon Phi.
- Its API is compatible to NumPy. Its data structures can work with NumPy without causing any conflict.
pyMIC ARCHITECTURE

pyMIC [Python]

_pyMICimpl [C/C++]

Intel LEO runtime

offload_array (Kernels) [C]
Host (CPU)

Python → Cython → C/C++

Offload directive

Xeon Phi

Shared Object (C/C++ kernel)
Deep Learning Applications

Deep Learning Frameworks

pyMIC Functions

Type Conversion

Broadcasting

Hardware (CPU + Xeon Phi Coprocessors)

pyMIC-DL
Chung et. al. (*) has proved that applications run on the Docker and physical machine have the same performance. Therefore, we perform entire evaluation on Docker in order to easily customize the constraints of system resources.

Hardware specifications described below:

<table>
<thead>
<tr>
<th>Codename</th>
<th>CPU</th>
<th>Coprocessor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>Intel Xeon E5-2680V3</td>
<td>Intel Xeon Phi 7120P</td>
</tr>
<tr>
<td>Microarchitecture</td>
<td>Sandy Bridge EP</td>
<td>Intel Many Integrated Core</td>
</tr>
<tr>
<td>Clock frequency</td>
<td>2.50/3.30 GHz</td>
<td>1.24/1.33 GHz</td>
</tr>
<tr>
<td>Memory Size</td>
<td>128 GB</td>
<td>16 GB</td>
</tr>
<tr>
<td>Cache</td>
<td>30.0 MB SmartCache</td>
<td>30.5 MB L2</td>
</tr>
<tr>
<td>Max Memory Bandwidth</td>
<td>68 GB/s</td>
<td>352 GB/s</td>
</tr>
<tr>
<td>Core/Threads</td>
<td>12/24</td>
<td>61/244</td>
</tr>
</tbody>
</table>

Figure 1.1: Function $eq$

Figure 1.2: Function $maximum$
Figure 2: Compare arange, argmax, exp, log of pyMIC-DL and Numpy
Figure 4: dot()
Figure 6: Run time of ANN-XP and ANN-CPU
Propose and develop pyMIC-DL which is a NumPy-like library for deep learning frameworks run on IXPKC coprocessor.

pyMIC-DL can currently be run on at most one IXPKC coprocessor.

- Exploit fully systems containing more than one IXPKC coprocessor.
- Share and balance workload to boost the performance.
Thank you for listening