Karlsruhe Institute of Technology

High Performance Humanoid Technologies

Introducing TensorFlow to H2T

Jonas Rothfuss, Fabio Ferreira | November 6, 2017

HIGH PERFORMANCE HUMANOID TECHNOLOGIES (H2T)

Outline

1. Overview
2. Symbolic Programming
3. Example
4. Conclusion

Overview

TensorFlow[1] is a symbolic open-source software library for numerical computation using data flow graphs (Abadi et al. 2016)

- suitable for both research & production
- currently available for Python, C/C++ and Java
- embedded applications: Mobile TensorFlow (Raspberry Pi, Android, iOS)

[1] TensorFlow, the TensorFlow logo and any related marks are trademarks of Google Inc.

Overview

TensorFlow[1] is a symbolic open-source software library for numerical computation using data flow graphs (Abadi et al. 2016)

- suitable for both research & production
- currently available for Python, C/C++ and Java
- embedded applications: Mobile TensorFlow (Raspberry Pi, Android, iOS)

[1] TensorFlow, the TensorFlow logo and any related marks are trademarks of Google Inc.

Overview

TensorFlow[1] is a symbolic open-source software library for numerical computation using data flow graphs (Abadi et al. 2016)

- suitable for both research & production
- currently available for Python, C/C++ and Java
- embedded applications: Mobile TensorFlow (Raspberry Pi, Android, iOS)

[1] TensorFlow, the TensorFlow logo and any related marks are trademarks of Google Inc.

Overview

- CPU support
- GPU support for Nvidia GPUs (requires CUDA and cuDNN)
- for installation see: [tensorflow manual]
- for workstations@H2T see: [H2T Deep Learning Wiki]

Overview

- CPU support
- GPU support for Nvidia GPUs (requires CUDA and cuDNN)
- for installation see: [tensorflow manual]
- for workstations@H2T see: [H2T Deep Learning Wiki]

Preliminary

Best Practices

- use Python's *virtualenv* along with pip
- if your environment allows it: use *Docker* images (not at H2T)
- use *tcmalloc* (memory allocator for high concurrency situations) for a tremendously more efficient resource allocation during training

- for more Best Practices, see: H2T Deep Learning Wiki

Preliminary

Best Practices

- use Python's *virtualenv* along with pip
- if your environment allows it: use *Docker* images (not at H2T)
- use *tcmalloc* (memory allocator for high concurrency situations) for a tremendously more efficient resource allocation during training

- for more Best Practices, see: H2T Deep Learning Wiki

Preliminary

Best Practices

- use Python's *virtualenv* along with pip
- if your environment allows it: use *Docker* images (not at H2T)
- use *tcmalloc* (memory allocator for high concurrency situations) for a tremendously more efficient resource allocation during training

- for more Best Practices, see: H2T Deep Learning Wiki

TensorFlow's Symbolic Programming

Two distinctive phases during development:

symbolic programming paradigm

- **phase 1**: build a computation graph of operations
- **phase 2**: convert the graph into a function (compilation) and execute it in a session

(Bad) consequences:

- computation happens as the last step in the code
- debugging code is usually difficult
- native python statements must be provided in TensorFlow language
- special statements for typical control flow e.g. *tf.while_loop*

TensorFlow's Symbolic Programming

Karlsruhe Institute of Technology

Two distinctive phases during development:

symbolic programming paradigm

- **phase 1**: build a computation graph of operations
- **phase 2**: convert the graph into a function (compilation) and execute it in a session

(Bad) consequences:

- computation happens as the last step in the code
- debugging code is usually difficult
- native python statements must be provided in TensorFlow language
- special statements for typical control flow e.g. *tf.while_loop*

TensorFlow's Symbolic Programming

Karlsruhe Institute of Technology

Two distinctive phases during development:

symbolic programming paradigm

- **phase 1**: build a computation graph of operations
- **phase 2**: convert the graph into a function (compilation) and execute it in a session

(Bad) consequences:

- computation happens as the last step in the code
- debugging code is usually difficult
- native python statements must be provided in TensorFlow language
- special statements for typical control flow e.g. *tf.while_loop*

TensorFlow’s Symbolic Programming

Two distinctive phases during development:

symbolic programming paradigm

- **phase 1**: build a computation graph of operations
- **phase 2**: convert the graph into a function (compilation) and execute it in a session

(Bad) consequences:

- computation happens as the last step in the code
- debugging code is usually difficult
- native python statements must be provided in TensorFlow language
- special statements for typical control flow e.g. *tf.while_loop*

TensorFlow's Symbolic Programming

Two distinctive phases during development:

symbolic programming paradigm

- **phase 1**: build a computation graph of operations
- **phase 2**: convert the graph into a function (compilation) and execute it in a session

(Bad) consequences:

- computation happens as the last step in the code
- debugging code is usually difficult
- native python statements must be provided in TensorFlow language
- special statements for typical control flow e.g. *tf.while_loop*

TensorFlow's Symbolic Programming

Karlsruhe Institute of Technology

Two distinctive phases during development:

symbolic programming paradigm

- **phase 1**: build a computation graph of operations
- **phase 2**: convert the graph into a function (compilation) and execute it in a session

(Bad) consequences:

- computation happens as the last step in the code
- debugging code is usually difficult
- native python statements must be provided in TensorFlow language
- special statements for typical control flow e.g. *tf.while_loop*

TensorFlow's Symbolic Programming

Two distinctive phases during development:

symbolic programming paradigm

- **phase 1**: build a computation graph of operations
- **phase 2**: convert the graph into a function (compilation) and execute it in a session

(Good) consequences :

- execution is efficient (memory, in-place computation)
- preprocessing and data loading is simply done by adding operations to graph
- distribute computation among different resources (multi-gpu, clusters)

TensorFlow's Symbolic Programming

Two distinctive phases during development:

symbolic programming paradigm

- **phase 1**: build a computation graph of operations
- **phase 2**: convert the graph into a function (compilation) and execute it in a session

(Good) consequences :

- execution is efficient (memory, in-place computation)
- preprocessing and data loading is simply done by adding operations to graph
- distribute computation among different resources (multi-gpu, clusters)

TensorFlow's Symbolic Programming

Karlsruhe Institute of Technology

Two distinctive phases during development:

symbolic programming paradigm

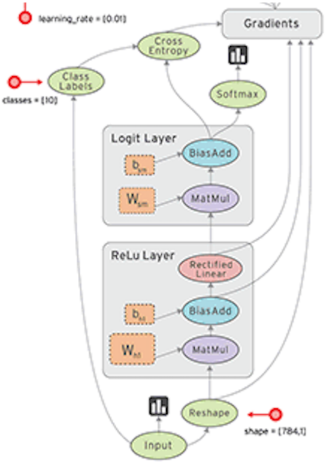
- **phase 1**: build a computation graph of operations
- **phase 2**: convert the graph into a function (compilation) and execute it in a session

(Good) consequences :

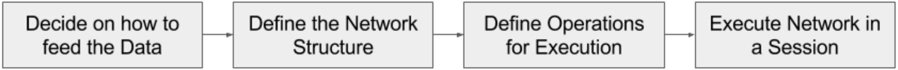
- execution is efficient (memory, in-place computation)
- preprocessing and data loading is simply done by adding operations to graph
- distribute computation among different resources (multi-gpu, clusters)

TensorFlow’s Symbolic Programming

An exemplary graph with input, neurons (weights, biases), softmax and cross entropy function:



Workflow of training a NN



Try out the enclosed example code by running the script *start_tutorial* from the command line.

Concluding

Things we considered most challenging during Deep Episodic Memory

Challenges

- setting-up the training pipeline with tfrecords
- keeping track of changes and decisions that affected the design of the model
- finding bugs (or assess the effect of code changes) in symbolic programming

Concluding

Karlsruhe Institute of Technology

Things we considered most challenging during Deep Episodic Memory

Challenges

- setting-up the training pipeline with tfrecords
- keeping track of changes and decisions that affected the design of the model
- finding bugs (or assess the effect of code changes) in symbolic programming

Concluding

Things we considered most challenging during Deep Episodic Memory

Challenges

- setting-up the training pipeline with tfrecords
- keeping track of changes and decisions that affected the design of the model
- finding bugs (or assess the effect of code changes) in symbolic programming

References I

Martín Abadi et al. “TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems”. In: *CoRR* abs/1603.04467 (2016). arXiv: 1603.04467. URL: http://arxiv.org/abs/1603.04467.