TensorFlow in CMSSW

Pablo de Castro, Christopher Jones, Jan Kieseler, Slava Krutelyov, Marcel Rieger, Markus Stoye, Mauro Verzetti

IML Meeting
28 February 2018
Jet-flavour tagger using multi-class neural networks

Input data structure:
- Global jet variables
- Secondary vertices
- Neutral & charged PF candidates

Multiple applications:
- AK4 tagger: b, c, udsg
- AK8 tagger: t, W, Z, H, QCD
- Jet momentum regression

AK4 tagger
### Successful integration in CMSSW: PR #19893

- Contains both TF interface and DeepFlavour tagger
- Most important “issues”: multi-threading + memory consumption
  - Changed interface language and model format 2 times (CPython → C → C++)
  - Knowledge on TF internals evolved during review process
- PR finally merged on 25 Jan (after 6 months) → PhysicsTools/TensorFlow
Choosing the backend: C vs. C++ (vs. Python)

- Choice of backend depends on software environment and TF’s requirements:
  - From-source installation requires **bazel** (google’s build tool, JDK 8)
  - CMSSW external packages iteratively accommodated to needs as TF interface evolved

1. Interface based on TF Python API ([link](#)):
   - Used Python C API and pre-built TF python package
   - Overhead & no handle on threading / memory (only used by user analyses)

2. Interface based on TF C API ([link](#)):
   - C API very low level and not necessarily convenient
   - Requires custom tensor implementation and graph, IO, etc. handling

3. Interface based on TF C++ API ([PhysicsTools/TensorFlow](#), [link](#))
   - Access to features like separation of “session ↔ graph”, thread models, ...
   - Shallow convenience layer to avoid boilerplate code and to connect to CMSSW internals (exceptions, logging, ...) ([TensorFlow.h](#))
Multi-threading in TF

- Upon session startup, TensorFlow starts *lots* of threads
  
  ```python
  import os
  import psutil
  import tensorflow as tf
  
  p = psutil.Process(os.getpid())
  print(p.num_threads())  # 2
  
  sess = tf.Session()
  print(p.num_threads())  # 10
  ```

- Reserved by dedicated thread pool for
  - Queued data loading
  - Parallelization *between* operations (inter_op_parallelism_threads)
  - Parallelization *within* operations (intra_op_parallelism_threads)

- Overhead acceptable for end-user, but experiment software typically implements its own threading model (e.g. via Intel’s Thread Building Blocks, TBB)

→ Critical impact on memory consumption, esp. in parallel production jobs
Custom session implementations

- In TF, threading behavior is bound to session implementation
  - Default session: `direct_session` (target: “”)
  - Actual session impl. configurable at runtime:

```
import tensorflow as tf
sess = tf.Session(target='<target_name>')
```

→ Created two custom session implementations (based on `direct_session`):
  - `NTSession`: session *without* threading at all (target: “no_threads”)
  - `TBBSession`: session *with* threading based on Intel’s TBB (target: “tbb”)
  - Conveniently configurable in CMSSW via:

```
#include "PhysicsTools/TensorFlow/interface/TensorFlow.h"
...
tensorflow::SessionOptions opts;
tensorflow::setThreading(opts, 1, "no_threads");
tensorflow::Session* session = tensorflow::createSession(opts);
```
Memory optimization: graphs formats

- TensorFlow graphs can become large
  - ~150 MB in memory for DeepFlavour tagger
  - Trained weights are saved as variables, need more memory than constants
  - Many operations and tensors used for training are saved & loaded

- Format 1: *SavedModel*
  - Supports multiple graphs
  - Can include assets
  - Loading via MetaGraphDef

- Format 2: Constant graphs
  - Converts variable to constant tensors
  - Can skip training ops
  - Loading via GraphDef
  - Reduces memory footprint by O(10-100)

- Format 3: Ahead-of-time compiled graphs (*tfcompile*), under investigation
TF uses two objects to store computational model

- **Graph**: contains the computational graph, representing the network
- **Session**: bounds operations to devices (CPU, GPU) & handles data transfer between them → Tensor’s

In CMSSW, we are mostly interested in model evaluation

- Graph’s are supposed to contain only *constant* data
- Session’s hold dynamic data, depending on device placement, current event, ...
Memory optimization: graph & session (2)

- Multi-threading in CMSSW allows for separation of graph’s and session’s
  - Load graph once and store in a global cache
  - Create new session per thread (in constructor of stream module in CMSSW) and “mount” the graph (example: `DeepFlavourTFJetTagsProducer`)

- Supported by TF interface for `GraphDef`’s and `MetaGraphDef`’s

```cpp
// load the graph definition
tensorflow::GraphDef* graphDef = tensorflow::loadGraphDef("/path/to/constantgraph.pb");

// create a session
tensorflow::Session* session = tensorflow::createSession(graphDef);
```

GraphDef:

```cpp
// load the meta graph definition
tensorflow::MetaGraphDef* metaGraph = tensorflow::loadMetaGraph("/path/to/simplegraph");

// create a session
tensorflow::Session* session = tensorflow::createSession(metaGraph, "/path/to/simplegraph");
```

MetaGraphDef:
```
// load the graph definition
tensorflow::GraphDef* graphDef = tensorflow::loadGraphDef("constantgraph.pb");

// create a session
tensorflow::Session* session = tensorflow::createSession(graphDef);

// create an input tensor (single batch of dimension 10)
tensorflow::Tensor input(tensorflow::DT_FLOAT, { 1, 10 });

// example: fill a single batch of the input tensor with consecutive numbers
// -> [[0, 1, 2, ...]]
for (float i = 0.; i < 10.; i++) input.matrix<float>()(0, i) = i;

// run call
std::vector<tensorflow::Tensor> outputs;
tensorflow::run(session, {{"input", input}}, {{"output"}}, &outputs);

// process outputs
std::cout << outputs[0].matrix<float>()(0, 0) << std::endl;
// -> 46.

tensorflow::closeSession(session);
delete graphDef;
```

(createsconstantgraph.py, Tensor API)
Summary

- Deep learning already proved enormous potential in HEP applications
- Integration within experiment frameworks is key to push developments

- TensorFlow models easy to build, but production deployment challenging, esp.:
  - Multi-threading compatibility
  - Memory footprint

- Measures taken to ensure proper integration into CMSSW
  - Hopefully eases future integrations of deep learning approaches
  - Thanks to everyone involved in the review process of #19893!