Batch Generator for training Machine Learning models from ROOT datasets

Dante Niewenhuis
University of Amsterdam
d.niewenhuis@hotmail.com

Lorenzo Moneta
CERN
lorenzo.moneta@cern.ch

Introduction

Generating batches from data is a vital part of many Machine Learning processes. However, ROOT doesn't have an easy way to get batches from a ROOT file. In this work we propose RBatchGenerator, a BatchGenerator build on top of the RDataFrame data structure.

Goals:

- ***Performance*** should be similar to popular AI tools.
- The BatchGenerator should be able to ***scale*** to large file sizes.
- It should be ***easy to use.***

Common Approach

Most batch generators follow the following steps:

1. Define a method to get data from event *i*.
2. Create batches of data by traversing the indices randomly.

RDataFrame provides extensive tools such as easy data filtering and defining of new columns. However, because in ROOT events are read sequentially, the classic approach is unviable.

Our Approach

RBatchGenerator consists of two steps:

1. *Chunking:* Load the next *chunkSize* rows from the data file into the RTensor.
2. *Batching:* Create batches of *BatchSize* from the Chunk of data. The batches consist of random entries from the Chunk. The batches can be returned in different types.

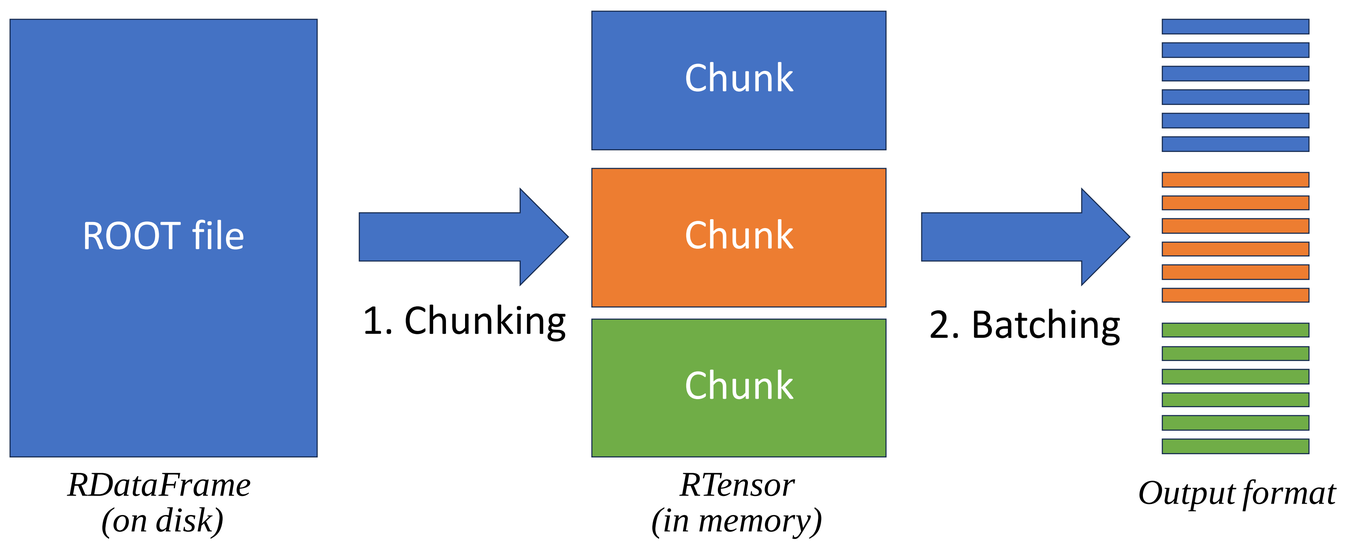

Fig1: Implementation of the RBatchGenerator

Parallel

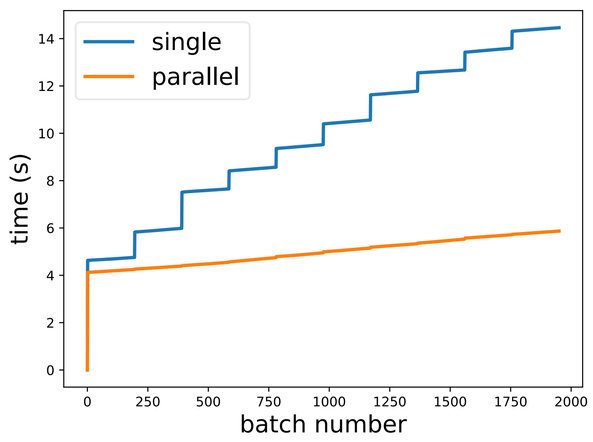

Fig2: Batch Loading time for parallel and non-parallel implementation

- Loading Chunks takes significantly longer than loading batches.
- This results in irregular loading times (see fig 1)
- Loading chunks can be done while processing batches

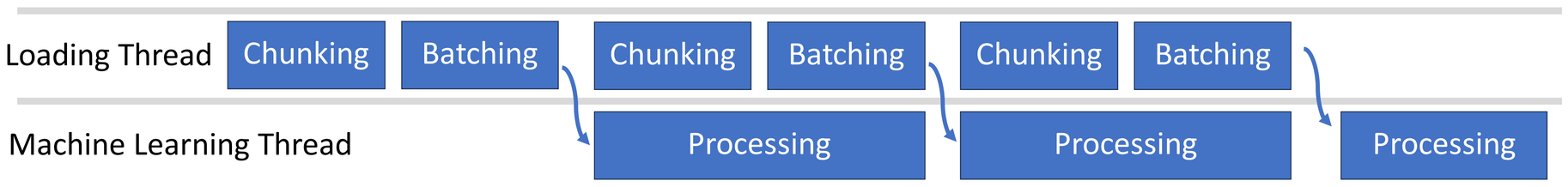

Fig3: Parallel implementation of RBatchGenerator

Performance

The RBatchGenerator performs similarly to TensorFlow on data that can fit into memory

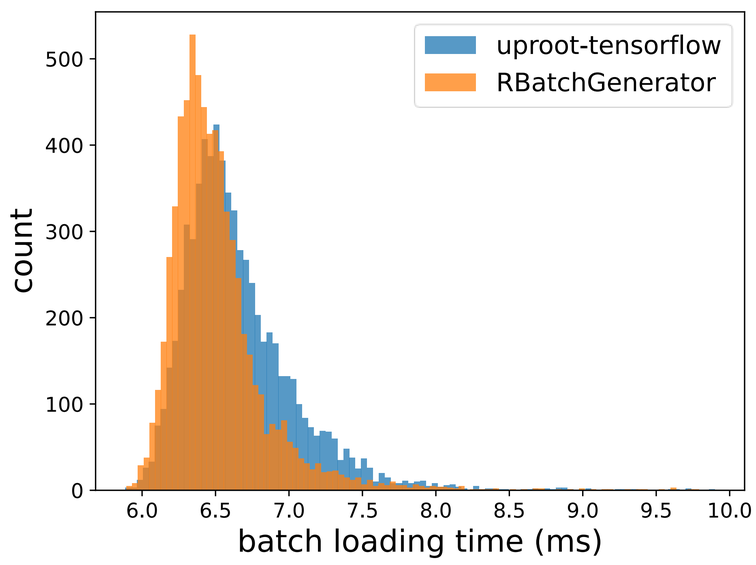

Fig4: Histogram of the batch loading time for RBatchGenerator and TensorFlow

Scaling

Increasing file size has little effect on the memory usage of the RBatchGenerator.

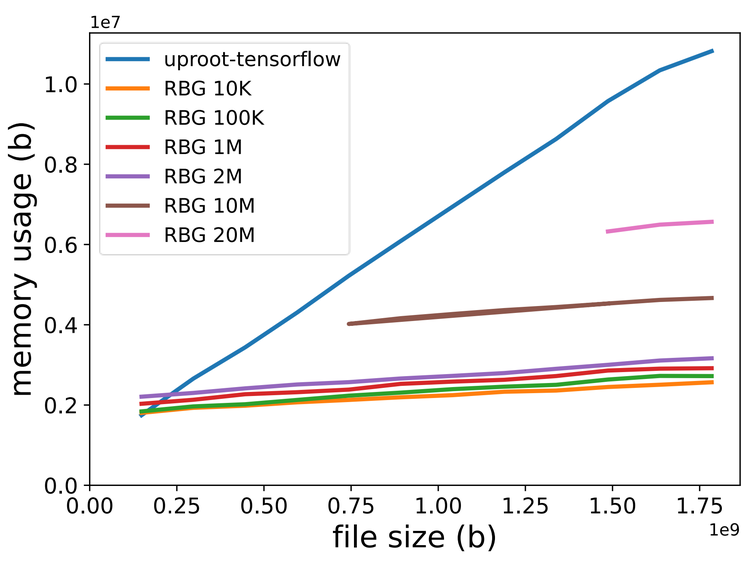

Fig5: Memory usage by RBatchGenerator at different chunk sizes and TensorFlow at increasing file size.

How to use

- RBatchGenerator is invoked using a single line of code
- Depending on the need, three types of output can be returned

```
# Returns a generator that returns NumPy arrays
gen_train, gen_validation = GetBatchGenerators(file_name, tree_name, chunk_rows,
                                batch_rows, target="Type", validation_split=0.3)

# Returns a TensorFlow Dataset
ds_train, ds_validation = GetTFDatasets(file_name, tree_name, chunk_rows,
                                batch_rows, target="Type", validation_split=0.3)

# Returns a generator that returns PyTorch Tensors
gen_train, gen_validation = GetPyTorchDataLoaders(file_name, tree_name, chunk_rows,
                                batch_rows, target="Type", validation_split=0.3)
```

- RBatchGenerator can be used directly in the Fit function:

```
model.fit(ds_train, validation_data=ds_validation, epochs=2)
```

Future Work

- Load directly from disk to GPU
- Loading more complex data
- Using multiple RDataFrames
- More complex RDataFrame interaction

Github

https://github.com/DanteNiewenhuis/root/tree/BatchGenerator_C++