

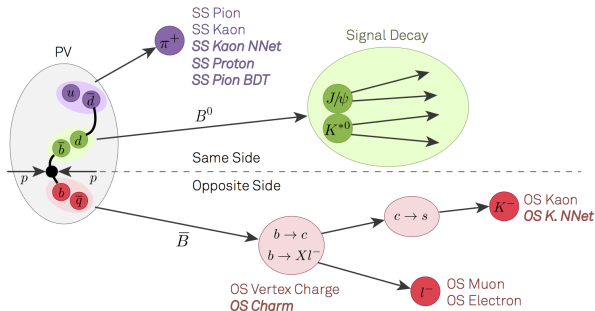
simpleNN

Manuel Schiller

University of Glasgow

April 20th, 2020

recap: Flavour Tagging at LHCb

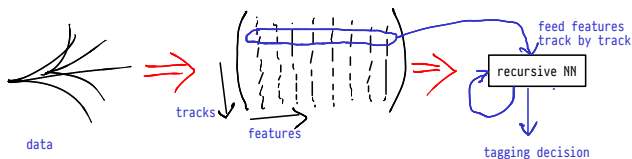


sketch: Julian Wishahi

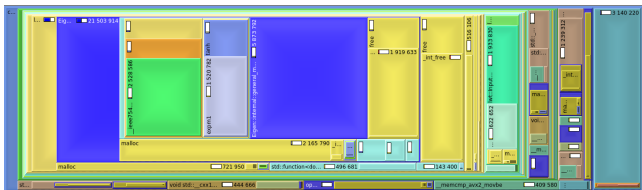
- for many LHCb analyses, need to know flavour of B at production
- Flavour tagging algorithms try to determine production flavour using MVA solutions
- new development: inclusive Flavour Tagging (give “whole event” to MVA) to improve tagging




inclusive Flavour Tagging



- idea: give full information to tagging (all tracks)
- about 20 features per track (momentum, projection on B flight vector, PID, ...)
- production prototype is being developed:
 - development and training happens with keras
 - needs to be fast: code will have to run in LHCb software trigger
 - evaluating NN quickly therefore crucial
 - first stop: [LWTNN](#)
(can evaluate NN in compiled C++)



- simple recursive NN test case (GRU):
 - 5 “tracks” of 3 features each
 - 4 outputs
- method: try evaluating the NN 1024 times for all 5 sets of features
- use valgrind to benchmark
- LWTNN: spends about half its time in dynamic memory management (malloc/free)
- also big chunk in std::exp

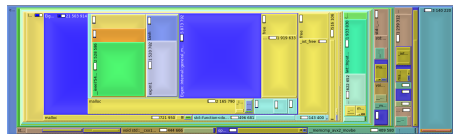


can we do better?

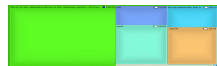
- by the time we compile code, we know structure of NN
 - run-time loadable NN structures not needed
 - allocate feature vectors statically or on stack
 - convert keras NN structure to C++ with code generator
 - thus avoid many run-time checks, virtual function calls, etc.
- no need for dynamic behaviour!
- start implementing [simpleNN](#)



simpleNN: first benchmark



LWTNN

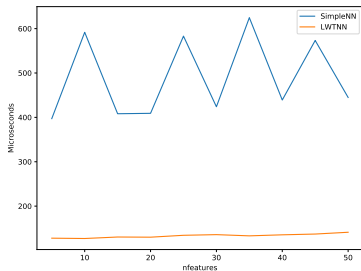


simpleNN

- LWTNN: initialisation 3 Mcycles, 1024 evaluations 37.9 Mcycles
- simpleNN: initialisation 16.1 kcycles, 1024 evaluations 8.5 Mcycles



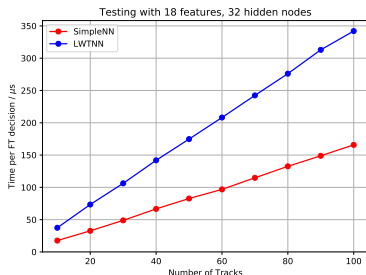
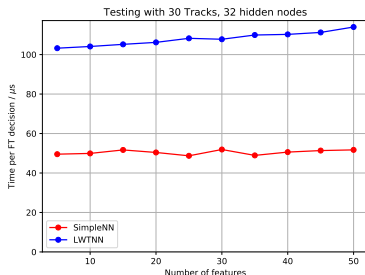
simpleNN: more realistic benchmark



plot: V. Jevtic

- 30 internal nodes, 18 input features, 2 dense layers followed by 2 GRU layers
- compile on Core i5-5257U CPU, compiler flags `-O2 -march=native`
- clearly, LWTNN does something much better!
- after a bit of debugging: LWTNN uses Eigen, which uses vectorised matrix-vector products in the dense layers

simpleNN: more realistic benchmark, fixed



plot: V. Jevtic

- fix simpleNN's matrix-vector multiplication in the dense layers: force compiler to autovectorise the matrix-vector product
- simpleNN now factor 2 faster (malloc/free, remember?)
- interesting for use elsewhere?



conclusion, next steps

- can do better than LWTNN
- faster code through
 - code generation
 - (auto-)vectorisation (by the compiler) of matrix-vector multiplication for normal matrix sizes, about as fast as Eigen (used by LWTNN)!
 - links:
 - the competition: [LWTNN](#)
 - [simpleNN](#)
 - code generator to convert keras NN structure/weights to C++: [rnngenerator](#) (V. Jevtic)
- next steps
 - make benchmarking code part of simpleNN
 - allow vectorized interface (i.e. vectors of features produce vector of outputs)
 - can we do something faster for `std::exp/std::tanh/...` that vectorises well?