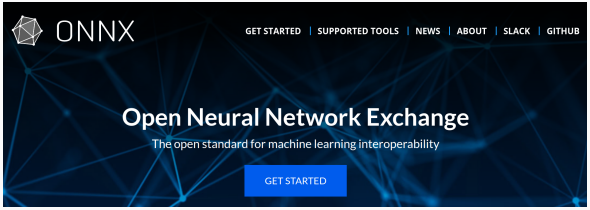
ONNX and its usage in Geant4

Thomas Madlener

Feb 04, 2022

Open Neural Network Exchange - ONNX



onnx/onnx
onnx.ai

- Open ecosystem providing an open source format for AI models
  - Originated from Facebook (as Toffee), now also supported by e.g. Microsoft and Amazon (and many others)
- Provides the definition of an extensible computation graph model, built-in operators and standard data types
  - A common intermediate representation (IR) of the computation graphs employed by the different ML frameworks
  - IR spec is versioned to allow for evolution
- Allows to transfer models between different ML frameworks
  - Choose the one that is best suited for the task at hand

ONNX Runtime - ORT

microsoft/onnxruntime
onnxruntime.ai

- Open source engine with the ability to run ONNX models
  - Training and inference
  - Supported on multiple platforms for different hardware
- Bindings for several languages available
  - Python, C++, C#, Java, JavaScript, Julia,...
- Deeply integrated with the whole Microsoft (Azure) stack[1]
- Shared library with C++ API available in LCG software stack
  - And used for inference in `extended/parameterisations/Par04` of geant4

[1] at least if you believe their website ;)

geant4 Par04 example

- Cylindrical calorimter setup with the possibility to use an ML model for fast simulation energy deposits
  - Support for ORT and LWTNN
  - See source for some more details
- Comes with pre-trained model
  - VAE with latent space $\in \mathbb{R}^{10}$, with i.i.d. $\sim \mathrm{No}(0, 1)$ and 4 condition variables (energy, angle + one-hot encoding of two different geometries)
  - Energy and angle labels are normalized to training range
  - Outputs a 1D vector of energy deposits on a 3D mesh (mesh coincides with mesh used for full simulation training samples)
- Rotates 3D mesh into place depending on incident particle momentum
- `src/Par04InferenceSetup.cc` for details

Running the example - environment setup

- Use latest (and now only) HEAD release: `HEAD-2022-01-30`
- Needs some additional environment setup to pull in ORT from `LCG_101` release
  - I couldn't get it to run with the setup that is used in the README!
  - Use `setup_env.sh` from *onnx-integration* branch of MarlinML that does a complete setup (including iLCSoft HEAD release)

```
git clone -b onnx-integration https://gitlab.desy.de/ilcsoft/MarlinML
source MarlinML/setup_env.sh
```

Running the example - compiling & running

- Copy example sources and compile

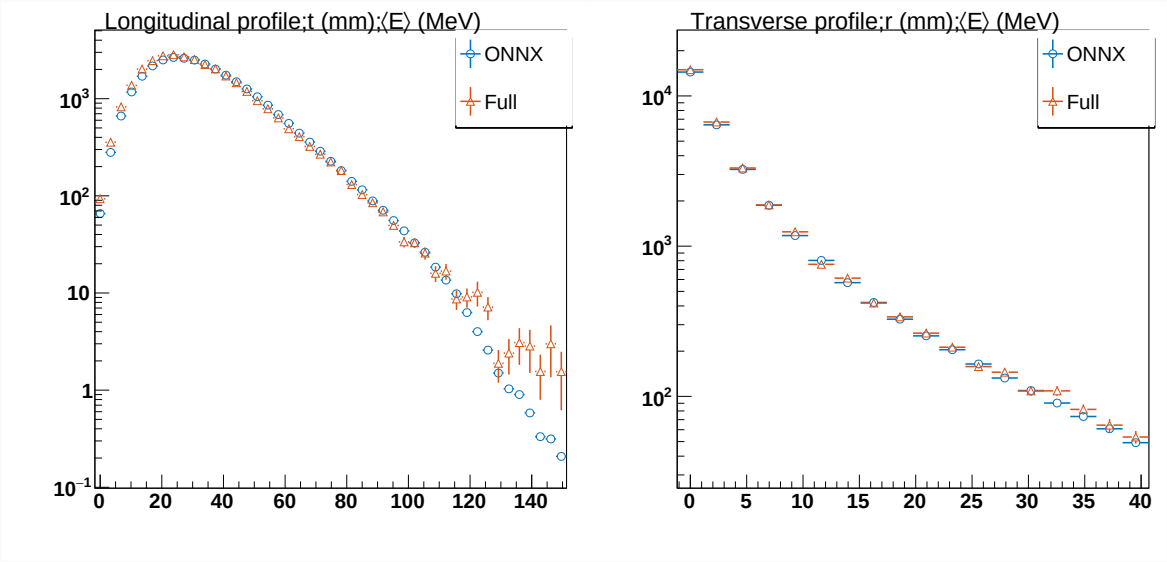
```
cp -r ${Geant4_DIR}/../../examples/extended/parameterisations/Par04 .
cd Par04
mkdir build && cd build
cmake ..
make -j4
```

  - There should be a message `ONNX Runtime inference library found.`
  - Potentially also a warning that can be ignored

- Run (according to README) full simulation and ONNX

```
./examplePar04 -m examplePar04.in       # full simulation
./examplePar04 -m examplePar04_onnx.in  # fast simulation w/ ONNX
```

Compare Full and FastSim



- Default settings shoot 10 GeV electron along *y*-axis

Plotting script

```python
import ROOT as r
from utils.plot_helpers import mkplot
from utils.setup_plot_style import set_basic_style
set_basic_style()
r.gStyle.SetStatStyle(0) # remove stats box

r.gROOT.SetBatch() # no display, just plots

f_onnx = r.TFile.Open('10GeV_100events_fastsim_onnx.root')
f_full = r.TFile.Open('10GeV_100events_fullsim.root')

keys = [k.GetName() for k in f_onnx.GetListOfKeys()]
hnames = filter(lambda k: f_onnx.Get(k).InheritsFrom('TH1D'), keys)

for name in hnames:
    can = mkplot([f_onnx.Get(name), f_full.Get(name)],
                 legEntries=['ONNX', 'Full'], legPos=(0.78, 0.78, 0.98, 0.95))
    can.SaveAs(f'{name}.pdf')

    can = mkplot([f_onnx.Get(name), f_full.Get(name)],
                 legEntries=['ONNX', 'Full'], legPos=(0.78, 0.78, 0.98, 0.95),
                 logy=True)
    can.SaveAs(f'{name}_log.pdf')
```

- `utils` are from tmadlener/chib_chic_polFW
  Get via `svn co https://github.com/tmadlener/chib_chic_polFW/trunk/python/utils utils`

ORT in MarlinML

- Added an example processor using ORT to MarlinML
  - *onnx-integration* branch
- Example of how to export PyTorch model to ONNX and use it in inference in a Marlin processor

```
17  class ExampleModule(torch.nn.Module):
18      """Very simple example module that doesn't do anything useful"""
19
20      def __init__(self):
21          super(ExampleModule, self).__init__()
22          self.linear = torch.nn.Linear(4, 4)
23
24      def forward(self, x, h):
25          vals = torch.tanh(self.linear(x) + h)
26          return vals
27
28      @staticmethod
29      def rand_inputs():
30          """Get random inputs for the module for cases where the values are
31          but the shape information is important (e.g. ONNX export)"""
32          return (torch.rand(3, 4), torch.rand(3, 4))
33
34
35  def _store_model_torch(model, filename):
36      """Store the model as torch script file"""
37      script_model = torch.jit.script(model)
38      script_model.save(filename)
39
40
41  def _store_model_onnx(model, filename):
42      """Store the model in the onnx format"""
43      inputs = model.rand_inputs()
44      # Very basic here. The export method has many more params!
45      torch.onnx.export(model, inputs, filename,
46                        # For testing some C++ API details
47                        input_names=('x', 'h'),
48                        output_names=('o',)
49                        )
```

ORT in MarlinML

```
  for (size_t i = 0; i < nTimes; ++i) {
    // Setup the input tensors
    //
    // NOTE: most of these should be const, but the API doesn't like that too
    // much for some reason
    /*const*/ auto x =
        m_linearInput ? linspace(0.f, 1.f, 12) : randomVals(0.f, 1.f, 12);
    /*const*/ auto h =
        m_linearInput ? linspace(1.f, 0.f, 12) : randomVals(0.f, 1.f, 12);

    const std::vector<int64_t> inputDims = {dim1, dim2};
    // Here we have to go in a slightly roundabout way as well by first
    // creating the tensors and then pushing them into the vector as it seems
    // to not be possible to directly initialize the vector for some reason...
    /*const*/ auto xTensor = Ort::Value::CreateTensor<float>(
        m_info, x.data(), dim1 * dim2, inputDims.data(), inputDims.size());
    /*const*/ auto hTensor = Ort::Value::CreateTensor<float>(
        m_info, h.data(), dim1 * dim2, inputDims.data(), inputDims.size());

    std::vector<Ort::Value> inputs;
    inputs.push_back(std::move(xTensor));
    inputs.push_back(std::move(hTensor));

    auto outputs = m_session->Run(Ort::RunOptions{nullptr}, m_inputNames.data(),
                                  inputs.data(), inputs.size(),
                                  m_outputNames.data(), m_outputNames.size());

    // Here we know that we only have one output variable from the model, so
    // we can directly use that
    const auto &result = outputs.front();
    // printTypeInfo("result:", result.GetTypeInfo(), "outputs[0]", 42);

    const auto values =
        std::vector<float>(result.GetTensorData<float>(),
                           result.GetTensorData<float>() + dim1 * dim2);
```

InferenceExamples/examples

```
Marlin run_example_onnx_model.xml
```