Reusing Neural Networks:
Lessons learned and Suggestions for the future

(Or: a long and oddly public note to self)

Tomasz Procter

INTRODUCTION

The problem

- Neural Nets are becoming more and more central features of many collider analyses.

- Use a wide variety of frameworks - `tensorflow`, `scikit-learn`, `pytorch`, `ROOT TMVA`...
- Implies:
  - Wide variety of dependencies -> heavy codes.
  - Wide variety of output formats (not all human readable).
  - ML in industry is less interested in reproducibility - scary differences between version numbers.
- And anyway, it's rare that an analysis actually publishes their NN data...

Two possible approaches

LWTNN

- Designed to take tf/sk-learn trained neural nets and run them in C++.
- Originally developed for ATLAS trigger.
- Really lightweight: depends on Eigen, Boost only.
- Only officially supports tf or sk-learn nets (though you can do more if you get creative)
- Human readable `.json` files.
- Currently in use “behind the scenes” in several ATLAS analyses (none yet public?)

ONNX (used via ONNXRunTime)

- Designed to allow neural nets trained in one context (e.g pytorch on a GPU) to be run in a completely different one (e.g. on customers’ mobiles).
- Developed by Facebook and Microsoft (though completely open source).
- Supports tf, pytorch, sklearn,++
- Non-human readable `.onnx` files.
- >= 1 analysis has published ONNX files.

An LWTNN Case-Study - EXOTICS

An LW[illegible]NN Case-Stud[illegible]

An ONNX Case Study: SUSY-2019-04

ATLAS SUSY-2019-04

- “Search for R-parity violating supersymmetry in a final state containing leptons and many jets”
- Uses a NN for one of their signal regions (and four control regions).
- Published ONNX files on hepdata (thankyou!)
- Also provided a relatively complete simpleAnalysis file.

The Neural Network(s)

- One network for each case 4jets-8jets
- 65 input variables - mix of event information (H_T, similar), and specific jet/lepton information (e.g. p_T, η, ϕ, btag for lead 10 jets)
- Includes pseudo-continuous b-score for jets?!
  - Detector level.
  - simpleAnalysis suggests using 5, 1 or 0 for truth level data.
  - Paper notes this was the second most significant variable?!
- Paper describes three layer DNN:
  - But interrogating the file it seems a lot more complex - ONNX bloat? Advanced loss?

Rivet ONNX Implementation:

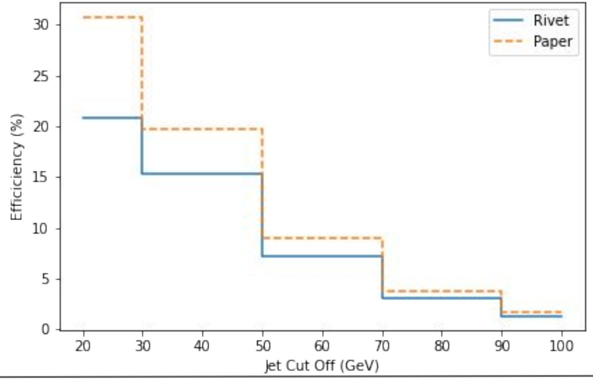
- Minimal RivetORT class that hides the boilerplate from users.
  - For now ORT (and LWTNN) still needs to be explicitly linked during analysis compilation

```
void init(...){
...
    for (size_t i = 4; i < 9; ++i)
        _ORTs[i] = make_unique<RivetORT>(RivetORT(analysisDataPath(std::to_string(i)+"jets.onnx")));
...
}
void analyze(...){
...
    _ORTs[jets.size()]->compute(nn_input_vector, nn_output);
...
}
```

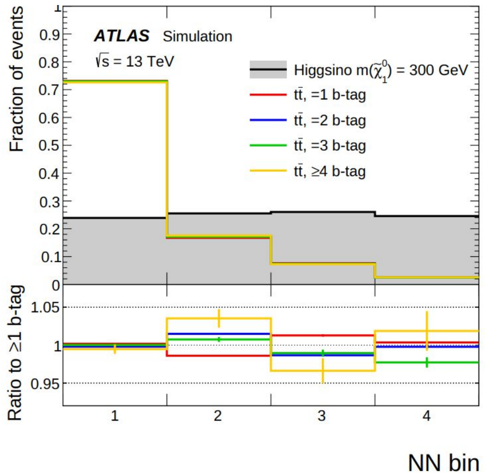
- Implementation follows simpleAnalysis very closely
  - With a couple of exceptions.
- NN bin cuts assumed from simpleAnalysis – but these are approximations!

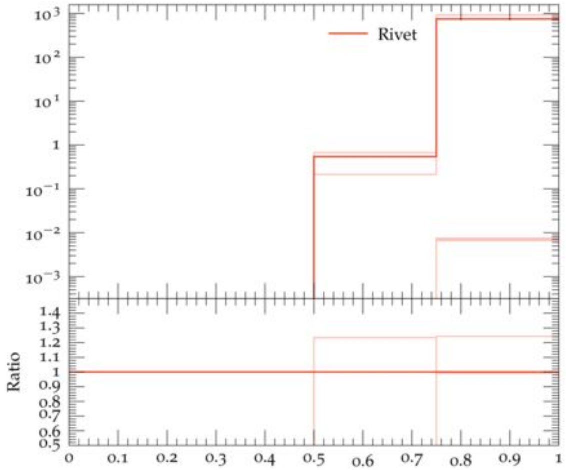
Rivet Implementation – Validation

- Cutflows:
  - Not enough leptons - 22% vs 37% of events pass 1 lep > 27GeV.
  - Too many events passing NN cut.
  - But shapes consistent once you adjust for the leptons.



- Reproduction of Figure 2:





Converting ONNX to LWTNN: SUSY-2019-04

Converting ONNX to LWTNN

- `onnx2keras` python module
- Use lwtnn script to convert keras -> lwtnn.
- Simple…?

Converting ONNX to LWTNN

- `onnx2keras` python module
- Use lwtnn script to convert keras -> lwtnn.
- Simple...?

- Not quite:
  - Keras add layer was not supported (is/will be now!)
  - Slicing layer implemented as a lambda (at least after `onnx2keras`)
- But got it working eventually - so we also have a version using lwtnn!

- N.b. possible future direct conversion lwtnn script?

Rivet LWTNN Implementation:

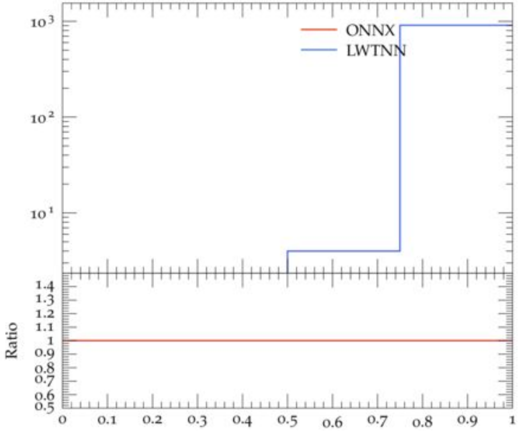
- Minimal RivetLWTNN header (not even a class!) that hides boilerplate.
    - For now LWTNN still needs to be explicitly linked during analysis compilation

```
void init(...){
...
    for (size_t i = 4; i < 9; ++i)
        _lwgs[i] = mkGraphLWTNN(analysisDataPath(std::to_string(i)+"j.json"));
...
}
void analyze(...){
...
    map<string, double> nn_output = _lwgs[jets.size()]->compute(nn_input);
...
}
```

- Already released in Rivet 3.1.7
    - See also example analysis.
    - And already used internally by an ATLAS W+Jets analysis.
- Analysis implementation otherwise identical to ONNX.

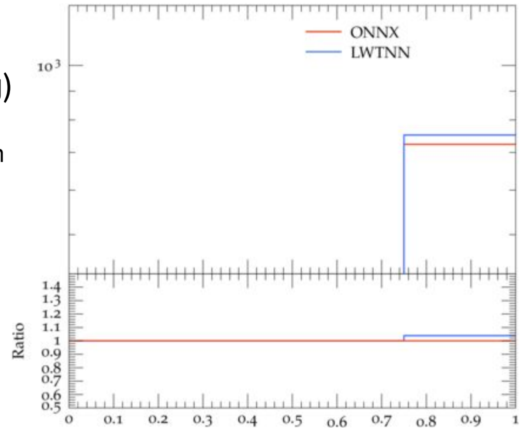
LWTNN-ONNX results comaprison

- Results effectively identical
  - Over 100k hepmc events tested, variety of models, only floating point differences (n.b. `lwtnn` uses `double`, `onnx` uses `float`)
- Performance: LWTNN slightly faster (but negligible compared to analysis time)
- Both are thread safe.



1l_nn_7j_4b
(no-smearing)

1l_nn_5j_4b
(no-smearing)
Note ONNX deficit ends up in overflow



Conclusions - For experiments and re-interpreters.

Final comparison

LWTNN

- Already used internally by some ATLAS analyses - zero extra effort to publish ✓
- Ultra-lightweight, but doesn't cover all conceivable cases - No pytorch support*/some weird layers
- No support for ROOT TMVA ✗
- Human readable files - could reconstruct network by hand if you needed too. ✓
- Only has a C++ interface* ✗

ONNX

- ✓ Relatively easy to convert models to onnx
- Heavier and more complex, but should cover just about every network conceivable.
- ✗ Limited experimental support for ROOT TMVA
- ✗ Non human readable files - are we confident these are truly preserved?
- ✓ Interfaces to any language reasonably used in science (C/C++, Python, Julia?,...)

(* if you're willing to get hacky, this is very circumventable)

Final Notes for Analyses:

- Above all, please publish your nets! - ideally, on HEPData in a preservable format.
- Please avoid variables which aren't accessible at truth (e.g, continuous b-score!)
  - Or if essential, please provide a detailed efficiency map.
- Cuts based on network score – please publish cut-values too!
- Ultra-complex network structures?
  - If essential, describe exactly what it does in detail.
- We'd like as much validation material as possible – more can go wrong.
- Are they valid to reinterpret at all (cf. CMS talk this morning):
  - Not asking for detector level networks (e.g. b-tagger).
  - Let us try!
- Rivet can support both formats – please use in your internal routines (and let me know if the interface works/can be improved)

Final Notes for Reinterpreters:

- Where networks are available, they can be worked with.
- I personally have a slight preference for lwtnn…
  - Format I can investigate easily.
  - More confident the results will be the same forever.
  - Personally, only need C++, and all the extra dependencies/boilerplate from ONNX will probably be a pain.
  - Particularly in ATLAS, there are quite a few of these just lying about.
- …but I'll take whatever I can get
  - I'm confident I'll be able to convert most networks into lwtnn from onnx.
  - Rivet should be able to deal with both (though may require you to link against external libraries yourself)
- Happy (and hope to) to discuss further!

BONUS

What the files look like inside:

LWTNN

```
{
  "input_sequences": [],
  "inputs": [
    {
      "name": "node_0",
      "variables": [
        {
          "name": "n_jet",
          "offset": 0,
          "scale": 1
        },
        {
          "name": "n_bcat",
          "offset": 0,
          "scale": 1
        },
        ……
"layers": [
    {
      "activation": "rectified",
      "architecture": "dense",
      "bias": [
        -0.1086258739233017,
        0.10020996630191803,
        0.04119415581226349,
        ……
```

ONNX

```
^H^F^R^Gpytorch^Z^C1.7:<80>^3^D
F
^Ginput.1^R^B11^Z^GSlice_0"^ESlice*^K
^Daxes@^A ^A^G*^K
^Dends@^B ^A^G*^M
^Fstarts@^@ ^A^G
>^R^B12^Z
Constant_1"^HConstant*"
^Evalue*^V^H^B^P^GJ^Pÿÿÿÿÿÿÿÿ^B^@^@^@^@^@^@^@ ^A^D


^B11^B12^R^B13^Z    Reshape_2"^GReshape
<95>^A
^Ginput.1
$_model.deep.sequence.0.linear.weight
"_model.deep.sequence.0.linear.bias^R^B14^Z^FGemm_3"^DGemm*^O
^Ealpha^U^@^@<80>? ^A^A*^N
^Dbeta^U^@^@<80>? ^A^A*^M
^FtransB^X^A ^A^B
^V
^B14^R^B15^Z^FRelu_4"^DRelu
<90>^A
^B15
$_model.deep.sequence.1.linear.weight
"_model.deep.sequence.1.linear.bias^R^B16^Z^FGemm_5"^DGemm*^O
……
```

Full Cutflows

Rivet

```
>=1 baseline lep: 29.1599% (46.08%)
>=1 siglep: 29.1599% (38.18%)
>=1 lead lep >= 27GeV: 22.8088% (37.36%)
-------------- 1 lepton category ------------
            20*GeV     40*GeV    60*Gev    80*GeV   100*GeV
>=4jets:    20.8156%, 15.3704%, 7.24739%, 3.05076%, 1.36089%
==4jets:    2.13609%, 6.13112%, 4.28612%, 2.14606%, 1.05078%
==5jets:    4.32596%, 5.02235%, 2.05078%, 0.708924%, 0.253644%
==6jets:    5.37082%, 2.75001%, 0.67488%, 0.158628%, 0.0439966%
==7jets:    4.62578%, 1.05087%, 0.178005%, 0.0278783%, 0.00975821%
==8jets:    2.52944%, 0.318079%, 0.0515322%, 0.00466212%, 0.00271047%
---------------------------------------------
==4jets,>=4btags: 0.0390215% (0.05%)
==5jets,>=4btags: 0.166727% (0.20%)
==6jets,>=4btags: 0.39751% (0.39%)
==7jets,>=4btags: 0.498463% (0.42%)
==8jets,>=4btags: 0.36171% (0.27%)
---------------------------------------------
==4jets,>=4btags,NN4jbin4: 0.0390215% (0.02%)
==5jets,>=4btags,NN5jbin4: 0.166727% (0.06%)
==6jets,>=4btags,NN6jbin4: 0.396259% (0.12%)
==7jets,>=4btags,NN7jbin4: 0.495651% (0.13%)
==8jets,>=4btags,NN8jbin4: 0% (0.10%)
>=4jets, 4b:    20.8156%, 15.3704%, 7.24739%, 3.05076%, 1.36089%
```

Paper

$\tilde{\chi}^0_{1,2} \to tbs$, $(m_{\tilde{\chi}^0_{1,2}} = 250$ GeV$)$	N_{raw}		N_{events}		Total Eff
All Events	269512		14491.03		100%
Lepton trigger	139169		7467.78		51.53%
$\geq$ 1 baseline lepton	124548		6677.48		46.08%
$\geq$ 1 signal lepton	103256		5533.07		38.18%
Leading lep $p_T \geq 27$ GeV	101018		5413.64		37.36%
Signal lepton is leading lepton	99585		5336.54		36.83%
Jet p_T threshold	20 GeV	40 GeV	60 GeV	80 GeV	100 GeV
1ℓ category	Total Eff.	Total Eff.	Total Eff.	Total Eff.	Total Eff.
$\geq$ 4 jets	30.8%	19.7%	9.0%	3.8%	1.7%
== 4 jets	4.7%	9.2%	5.8%	2.8%	1.3%
== 5 jets	7.6%	6.4%	2.4%	0.8%	0.3%
== 6 jets	8.2%	2.9%	0.7%	0.1%	0.0%
== 7 jets	5.8%	0.9%	0.1%	0.0%	0.0%
== 8 jets	2.9%	0.2%	0.0%	0.0%	0.0%
== 4 jets, $\geq$ 4 b-tags	0.05%	0.08%	0.05%	0.02%	0.01%
== 5 jets, $\geq$ 4 b-tags	0.20%	0.17%	0.06%	0.02%	0.01%
== 6 jets, $\geq$ 4 b-tags	0.39%	0.15%	0.03%	0.01%	0.00%
== 7 jets, $\geq$ 4 b-tags	0.42%	0.06%	0.01%	0.01%	0.00%
== 8 jets, $\geq$ 4 b-tags	0.27%	0.02%	0.00%	0.00%	0.00%
== 4 jets, $\geq$ 4 b-tags, NN_{4j} bin 4	0.02%	–	–	–	–
== 5 jets, $\geq$ 4 b-tags, NN_{5j} bin 4	0.06%	–	–	–	–
== 6 jets, $\geq$ 4 b-tags, NN_{6j} bin 4	0.12%	–	–	–	–
== 7 jets, $\geq$ 4 b-tags, NN_{7j} bin 4	0.13%	–	–	–	–
== 8 jets, $\geq$ 4 b-tags, NN_{8j} bin 4	0.10%	–	–	–	–

Full Cutflows (lepton adjusted)

Rivet

```
-------------- 1 lepton category ------------
            20*GeV    40*GeV    60*Gev    80*GeV    100*GeV
>=4jets:    34.0953%, 25.1762%, 11.871%, 4.99704%, 2.2291%
            (30.8%)     (19.7%)    (9.0%)    (3.8%)    (1.7%)
==4jets:    3.49884%, 10.0426%, 7.02052%, 3.51518%, 1.72115%
==5jets:    7.08577%, 8.22645%, 3.3591%, 1.16119%, 0.41546%
==6jets:    8.79723%, 4.50442%, 1.10543%, 0.259827%, 0.072065%
==7jets:    7.57687%, 1.72128%, 0.291567%, 0.0456638%, 0.0159836%
==8jets:    4.14314%, 0.521002%, 0.084408%, 0.00763639%, 0.00443965%
---------------------------------------------
==4jets,>=4btags: 0.0639159% (0.05%)
==5jets,>=4btags: 0.273092% (0.20%)
==6jets,>=4btags: 0.651108% (0.39%)
==7jets,>=4btags: 0.816466% (0.42%)
==8jets,>=4btags: 0.592469% (0.27%)
---------------------------------------------
==4jets,>=4btags,NN4jbin4: 0.0639159% (0.02%)
==5jets,>=4btags,NN5jbin4: 0.273092% (0.06%)
==6jets,>=4btags,NN6jbin4: 0.649059% (0.12%)
==7jets,>=4btags,NN7jbin4: 0.81186% (0.13%)
==8jets,>=4btags,NN8jbin4: 0% (0.10%)
```

Paper

$\tilde{\chi}^0_{1,2} \to tbs$, $(m_{\tilde{\chi}^0_{1,2}} = 250$ GeV)	N_{raw}		N_{events}		Total Eff
All Events	269512		14491.03		100%
Lepton trigger	139169		7467.78		51.53%
$\geq$ 1 baseline lepton	124548		6677.48		46.08%
$\geq$ 1 signal lepton	103256		5533.07		38.18%
Leading lep $p_T \geq 27$ GeV	101018		5413.64		37.36%
Signal lepton is leading lepton	99585		5336.54		36.83%
Jet p_T threshold	20 GeV	40 GeV	60 GeV	80 GeV	100 GeV
1ℓ category	Total Eff.	Total Eff.	Total Eff.	Total Eff.	Total Eff.
$\geq$ 4 jets	30.8%	19.7%	9.0%	3.8%	1.7%
== 4 jets	4.7%	9.2%	5.8%	2.8%	1.3%
== 5 jets	7.6%	6.4%	2.4%	0.8%	0.3%
== 6 jets	8.2%	2.9%	0.7%	0.1%	0.0%
== 7 jets	5.8%	0.9%	0.1%	0.0%	0.0%
== 8 jets	2.9%	0.2%	0.0%	0.0%	0.0%
== 4 jets, $\geq$ 4 *b*-tags	0.05%	0.08%	0.05%	0.02%	0.01%
== 5 jets, $\geq$ 4 *b*-tags	0.20%	0.17%	0.06%	0.02%	0.01%
== 6 jets, $\geq$ 4 *b*-tags	0.39%	0.15%	0.03%	0.01%	0.00%
== 7 jets, $\geq$ 4 *b*-tags	0.42%	0.06%	0.01%	0.01%	0.00%
== 8 jets, $\geq$ 4 *b*-tags	0.27%	0.02%	0.00%	0.00%	0.00%
== 4 jets, $\geq$ 4 *b*-tags, NN_{4j} bin 4	0.02%	–	–	–	–
== 5 jets, $\geq$ 4 *b*-tags, NN_{5j} bin 4	0.06%	–	–	–	–
== 6 jets, $\geq$ 4 *b*-tags, NN_{6j} bin 4	0.12%	–	–	–	–
== 7 jets, $\geq$ 4 *b*-tags, NN_{7j} bin 4	0.13%	–	–	–	–
== 8 jets, $\geq$ 4 *b*-tags, NN_{8j} bin 4	0.10%	–	–	–	–

NN binning

discriminate the higgsino signal from the $t\bar{t}$ background. The full distribution of the NN output, binned in four even-width bins with approximately equal signal fraction, is fitted in each of the regions with at

GAMBIT implementation (via LWTNN) SUSY-2019-04

GAMBIT IMPLEMENTATION

- Backending LWTNN for GAMBIT actually quite easy:
  - Advantage of small, simple code with minimal dependencies.
- Example analysis seems to run ok....
- But this is at a very early stage.