AutoEncoder for NANOAOD & update on Event Classification

Valentin Kuznetsov, Cornell University

CMS R&D meeting, Feb 2018

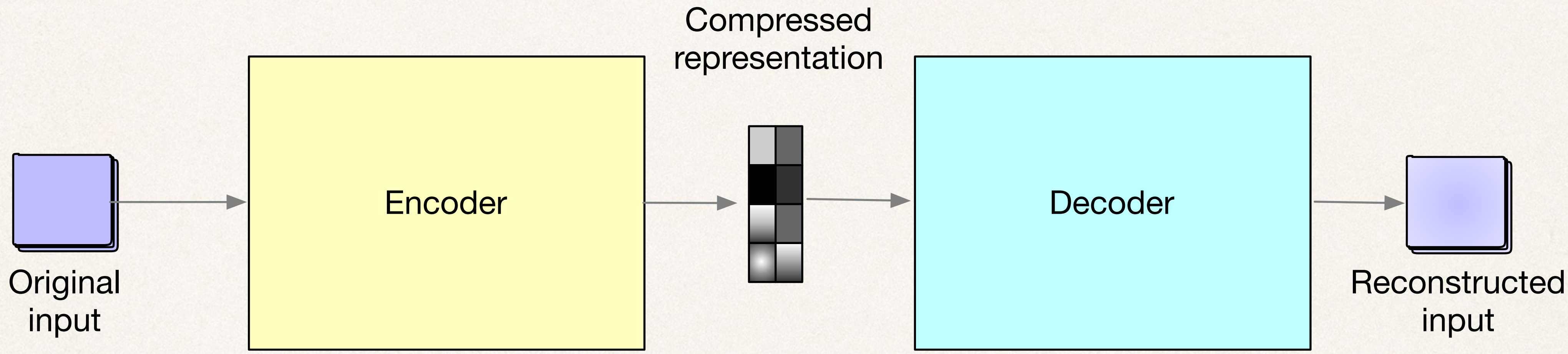
CMS NANOAOD

- CMS NANOAOD is a data-format that:
  - centrally produced (running at 10-20 Hz rate)
  - designed for analysis with bare ROOT
  - relatively small size, few KB per event
  - store basic analysis object to cover analysis use cases
- It is flat ROOT tuple with basic data-types
  - nMuons, Muon_pt[nMuons]
  - simple values, vector of values, single objects (met), collection of objects (jets)

Reference: http://bit.ly/2Bx27pA

Explore usage of AutoEncoder for data compression

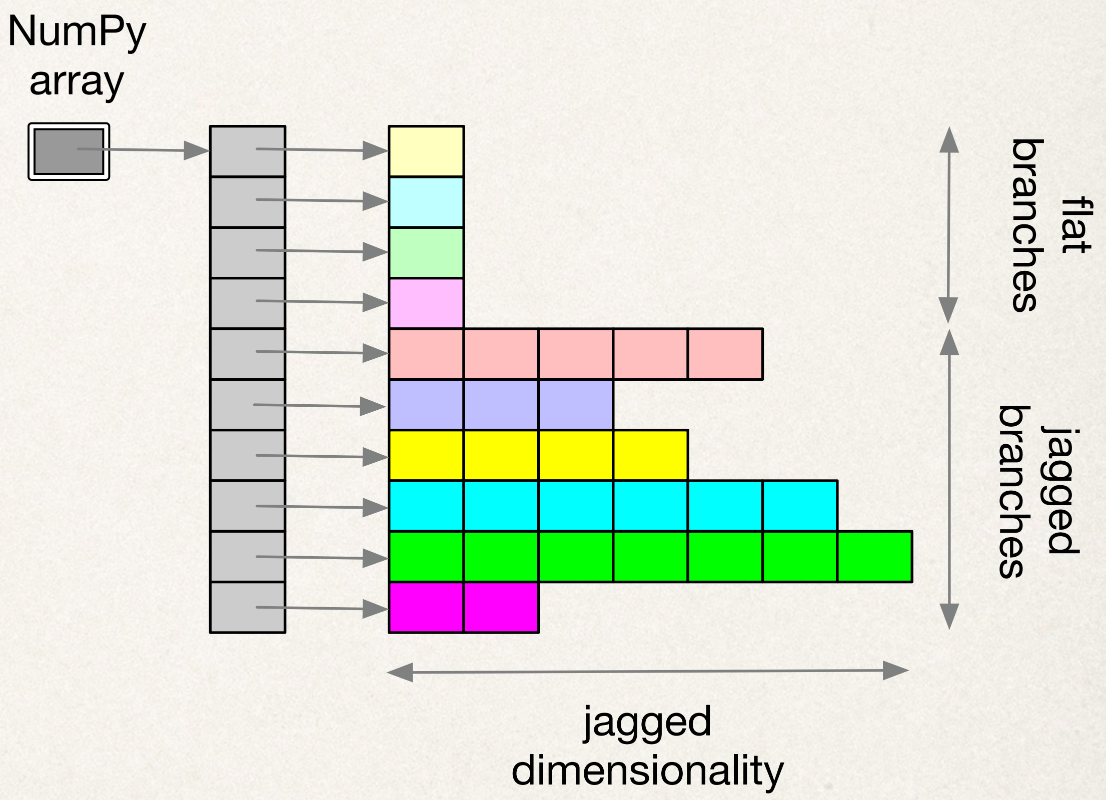
AutoEncoder



- AutoEncoder is an artificial network used for unsupervised learning
- AutoEncoder learns data representation (encoding) for the purpose of dimensionality reduction
- It projects given input to compressed representation and decode it back, usually encoder and decoder parts are symmetrical in terms of representation

Data representations

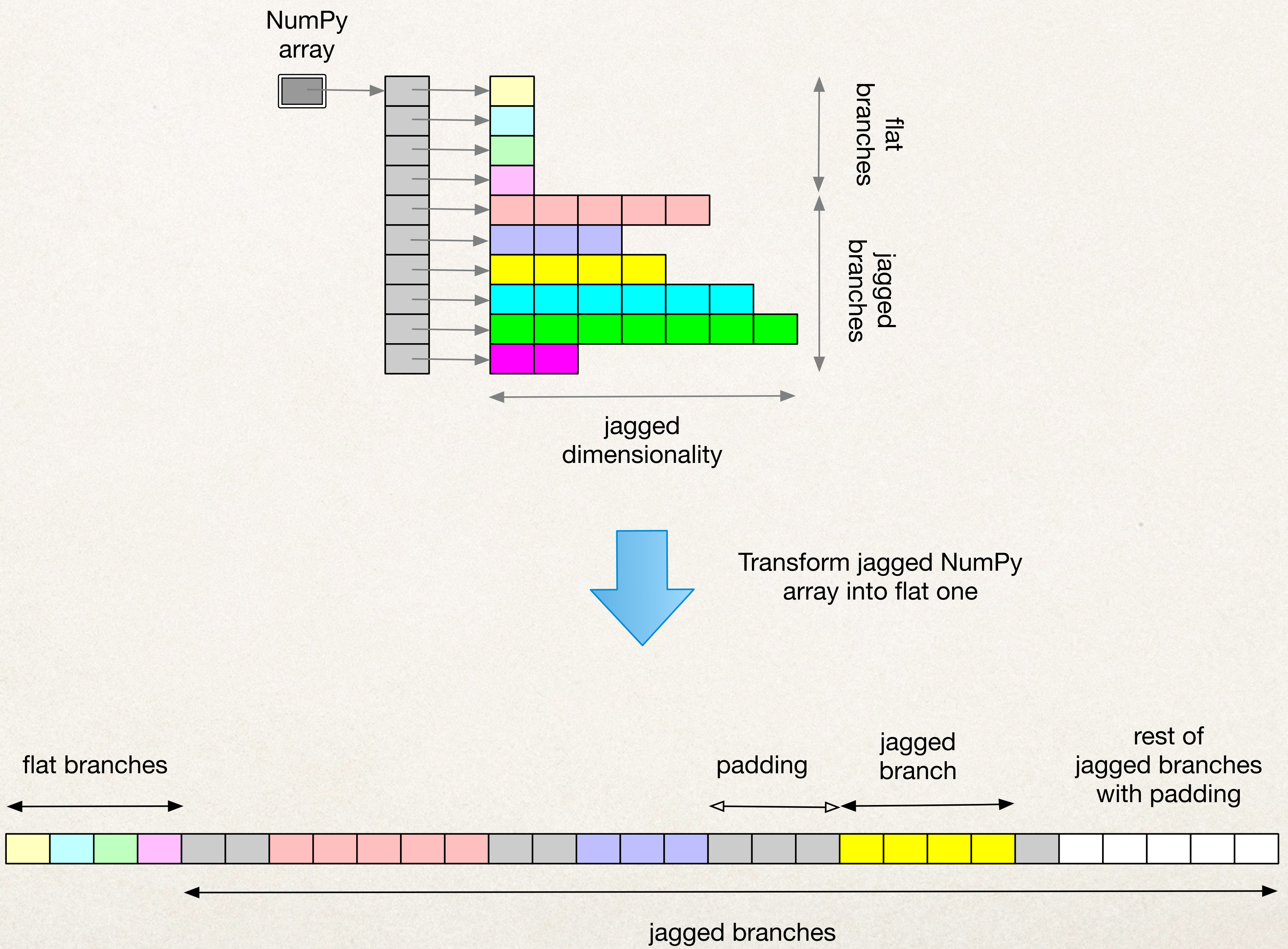
- Data are stored in flat ROOT files
- We can read ROOT files via uproot
- Each event is a composition of flat and jagged arrays
- Usually flat arrays size is less then jagged ones
- Such data representation is not directly suitable for ML (dynamic dimension of jagged arrays across events) and should be flatten to fixed size inputs
- For ML tasks we need to normalize/standardize data into some range, therefore two-step processing is required



Setup

- RelVal TTbar NANOAOD sample
  - 9000 events, 560 flat and 121 jagged branches, jagged dimensionality varies from 4 till 165
  - data are transformed to 0-1 range with padded values for jagged branches assigned to some value, e.g. 0, -1
- Tried 3 different models:
  - simple AE with different hidden layers, inputs -> hidden layer -> compressed -> hidden layer -> outputs, MAE/MSE/RMSE/MSLE losses
  - deep AE with 2 hidden layers in encoder/decoder with additional dropouts layers in between, MAE/MSE/RMSE/MSLE losses
  - Variational AE with latent variable space for input data and custom loss function (reconstruction loss + Kullback-Leibler divergence)
  - training was done on Tesla K40, 70% for training, 15% for test and 15% for validation sets

Data transformation



Benchmarks

- The uproot package can be successfully used to read ROOT files. I used two-step procedure: obtain min/max/dim values; used them in second pass

```
# 1000 entries, 684 branches, 5.4686050415 MB,  0.250020027161 sec,  21.8726679763 MB/sec, 35.9971163199 kHz
# 1000 entries, 684 branches, 5.84591293335 MB, 0.0748720169067 sec, 78.0787425645 MB/sec, 120.20512298 kHz
# 1000 entries, 684 branches, 5.96862983704 MB, 0.0763759613037 sec, 78.1480158829 MB/sec, 117.838124017 kHz
# 1000 entries, 684 branches, 6.13416290283 MB, 0.0802531242371 sec, 76.4351912919 MB/sec, 112.145166753 kHz
# 1000 entries, 684 branches, 5.94776153564 MB, 0.0763578414917 sec, 77.8932643911 MB/sec, 117.866087152 kHz
```

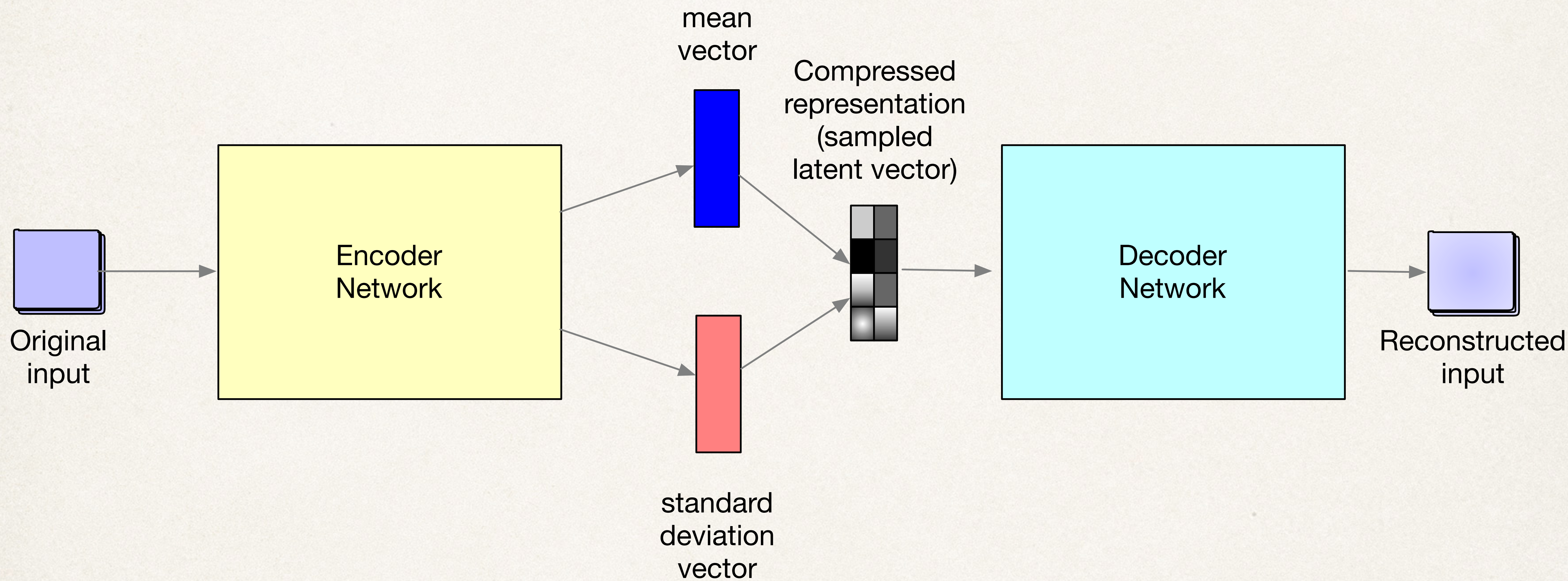
- Large compression is possible, e.g. using 2650 vector with 560 flat and 121 jagged branches (size per event: 4.5KB flat, 16.8KB jagged, ratio: 0.27)

```
test data    : 28620037 bytes (28.6MB)
encoded data: 2764837 bytes (2.8MB), 90.3395058504% reduction
decoded data: 14310037 bytes (14.3MB), 49.99993536% reduction
```

- Training time on GPU takes about 5-10min for 9K events

```
VAE with 5 layers (2048-1024-512-1024-2048) with Dropouts in intermediate layers
Trainable params: 27,488,436
```

Variational AutoEncoder



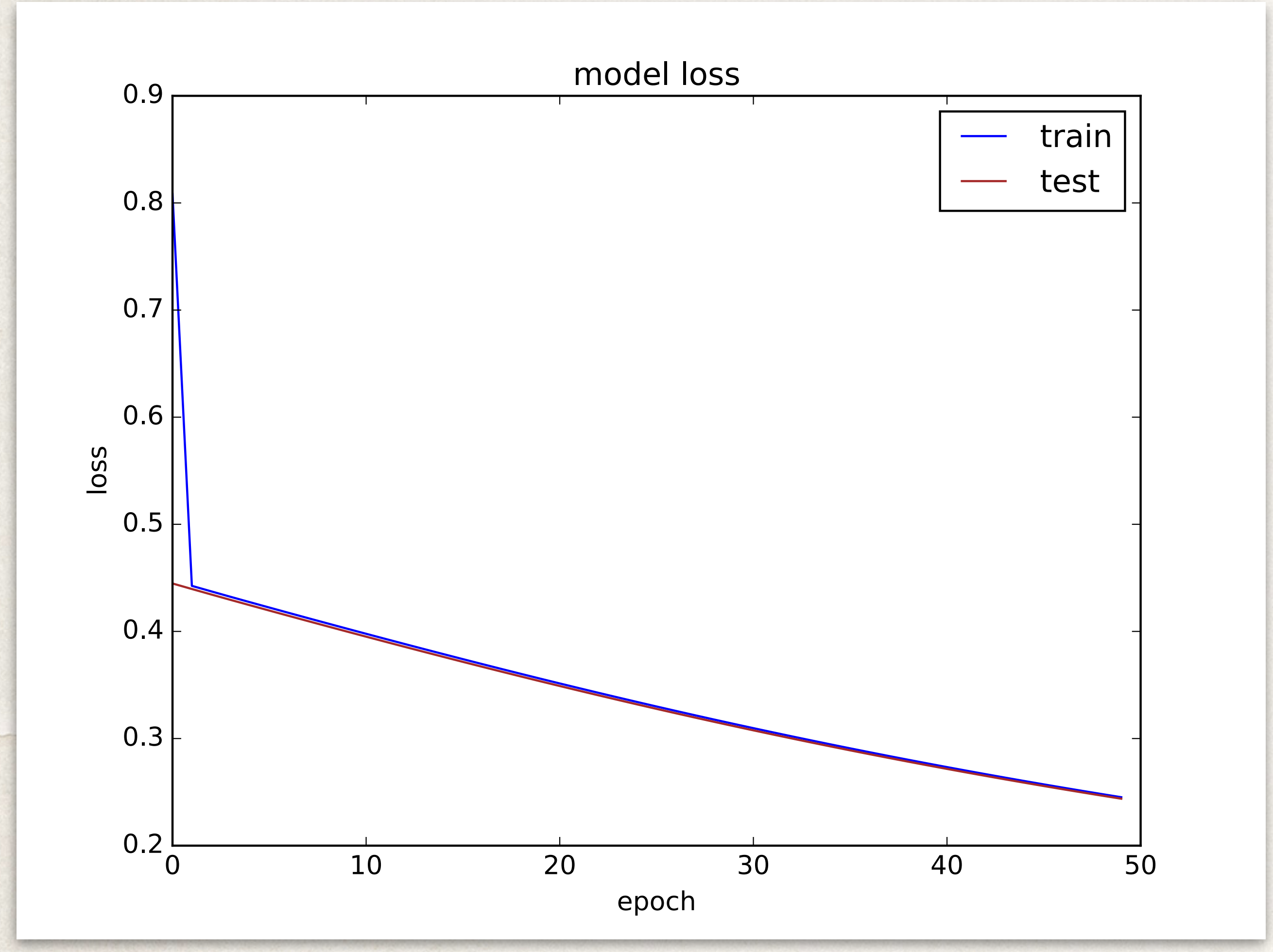
VAE learns a latent variable model for its input data as probability model of data X and latent variable z:

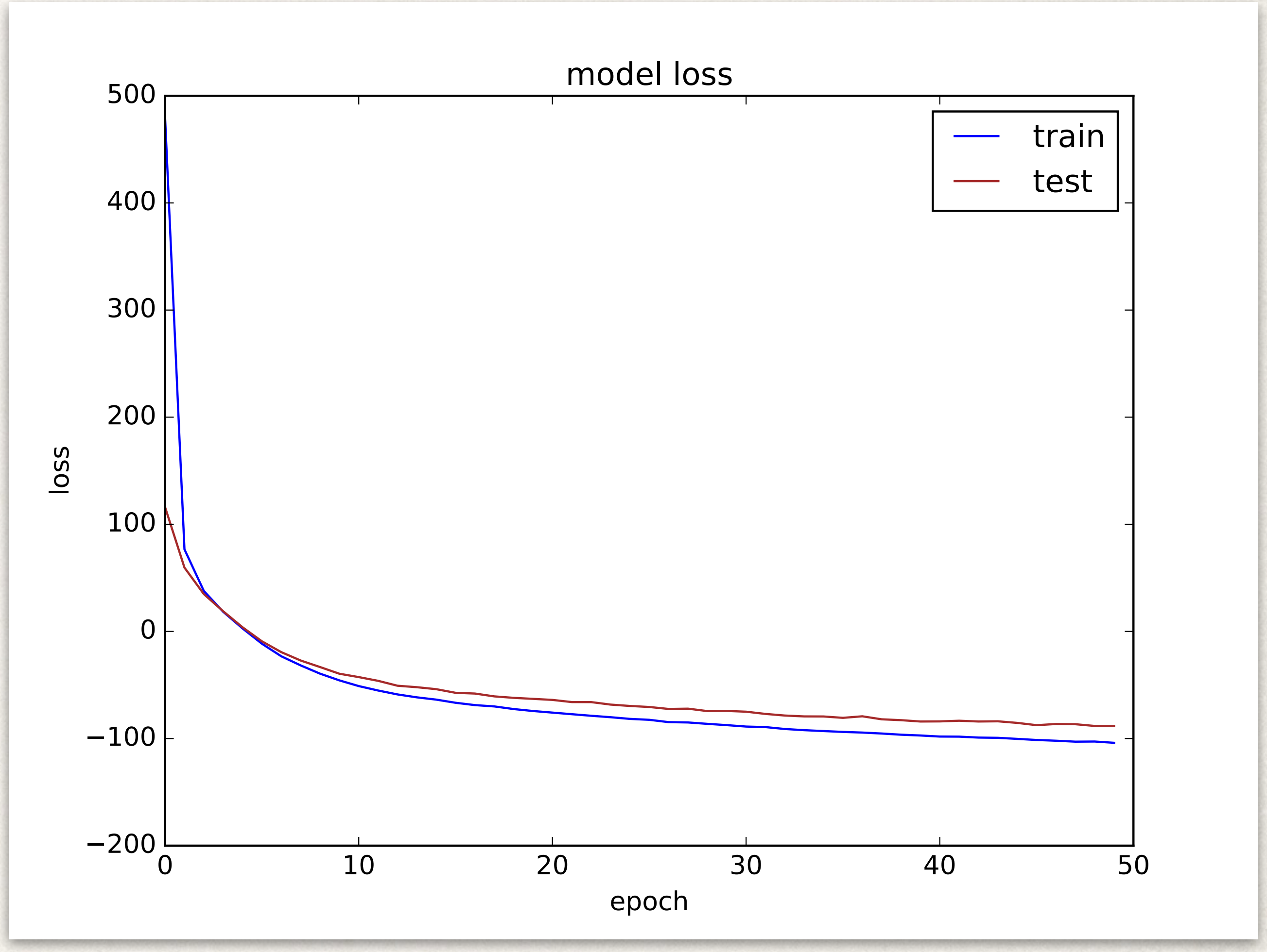
https://arxiv.org/abs/1312.6114

http://kvfrans.com/variational-autoencoders-explained/

https://jaan.io/what-is-variational-autoencoder-vae-tutorial/

Losses

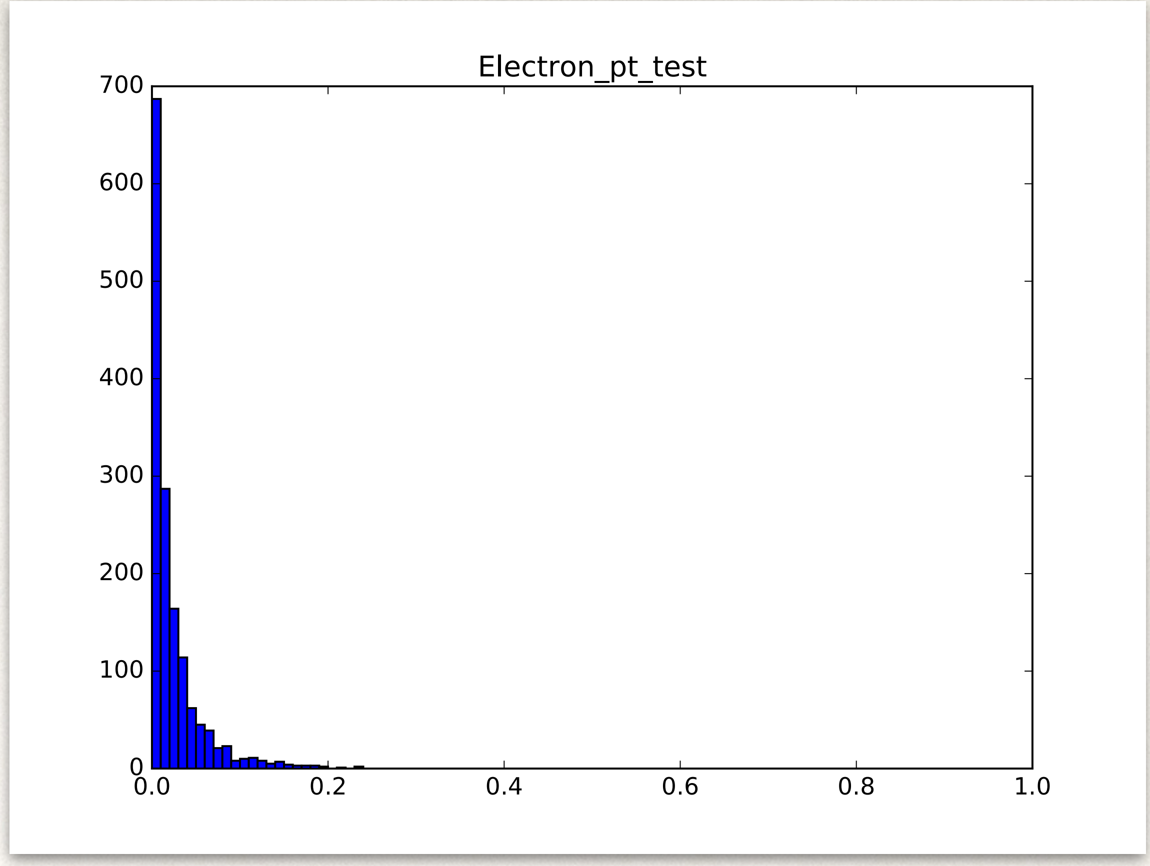




Simple AE: 2650-521-2650
RMSE: 0.006

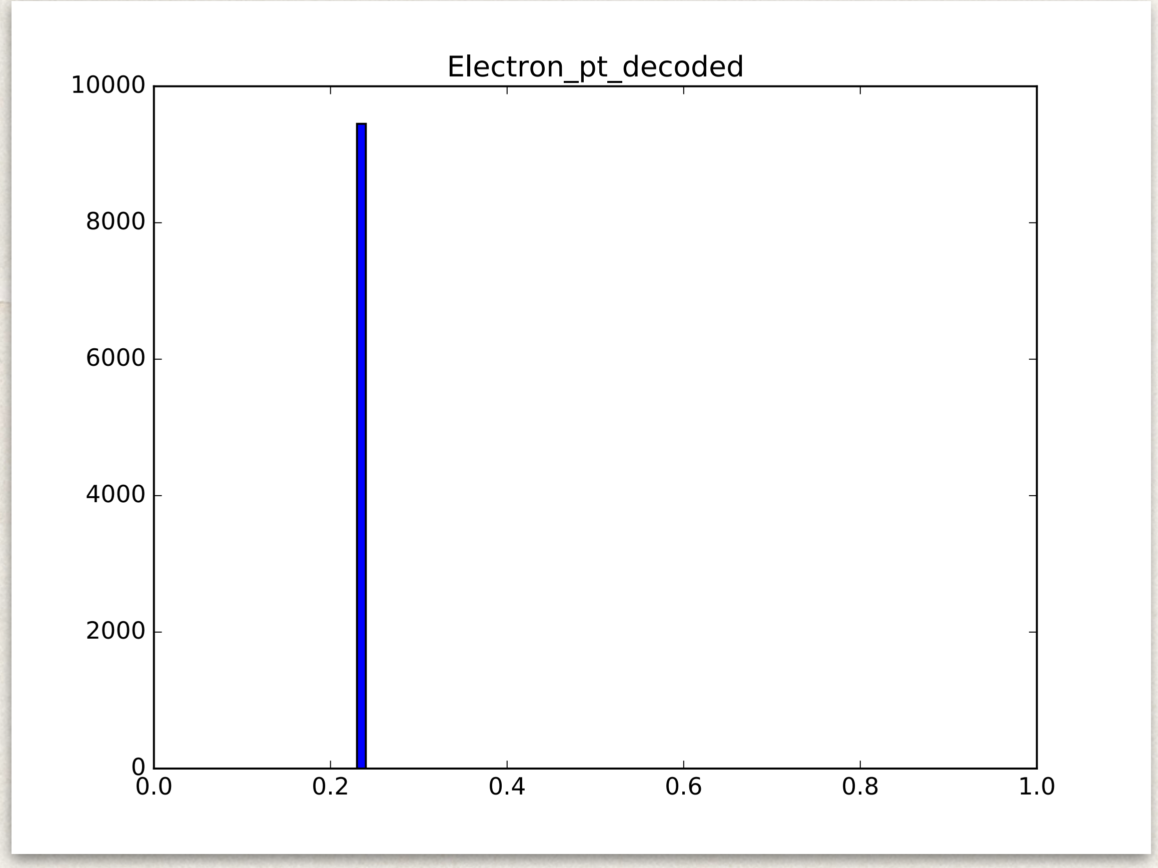
VAE: 2650-1024-512-1024-2650
RMSE: 4e-6

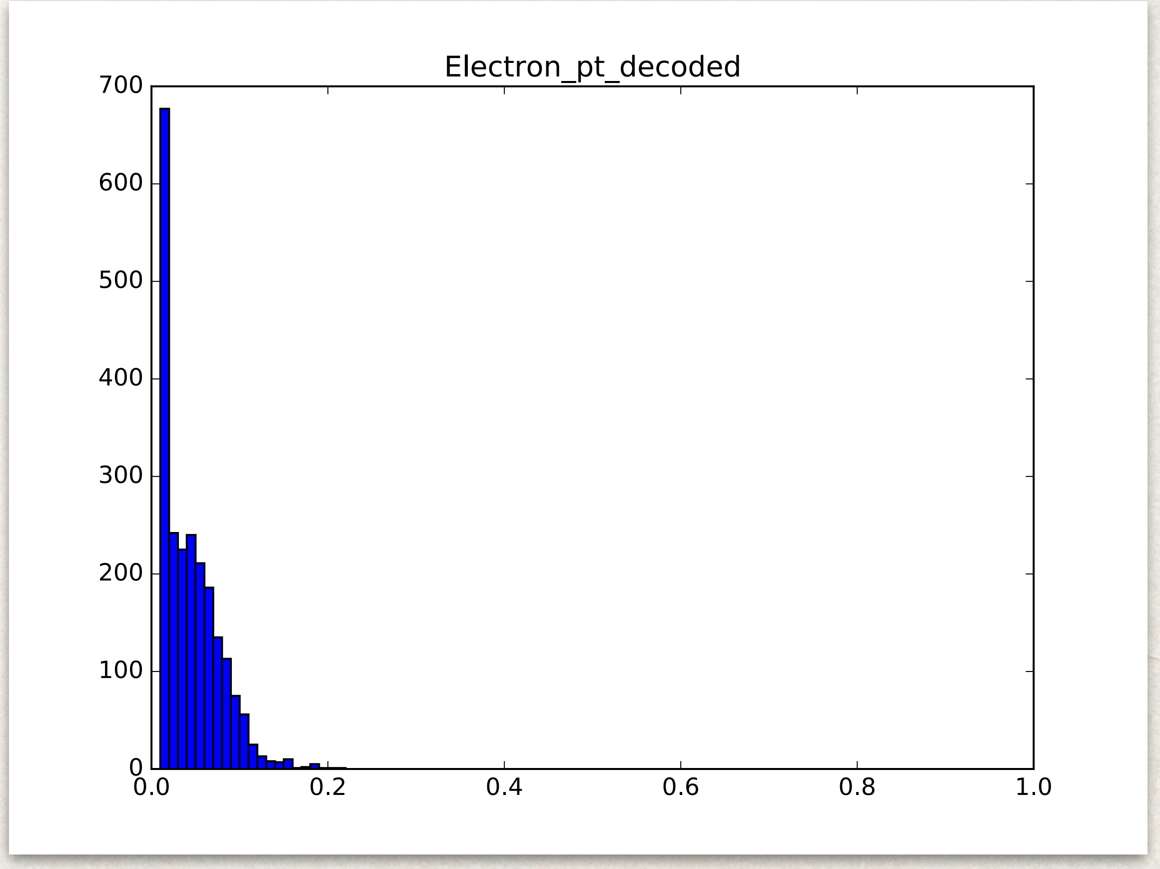
Can we trust the results?

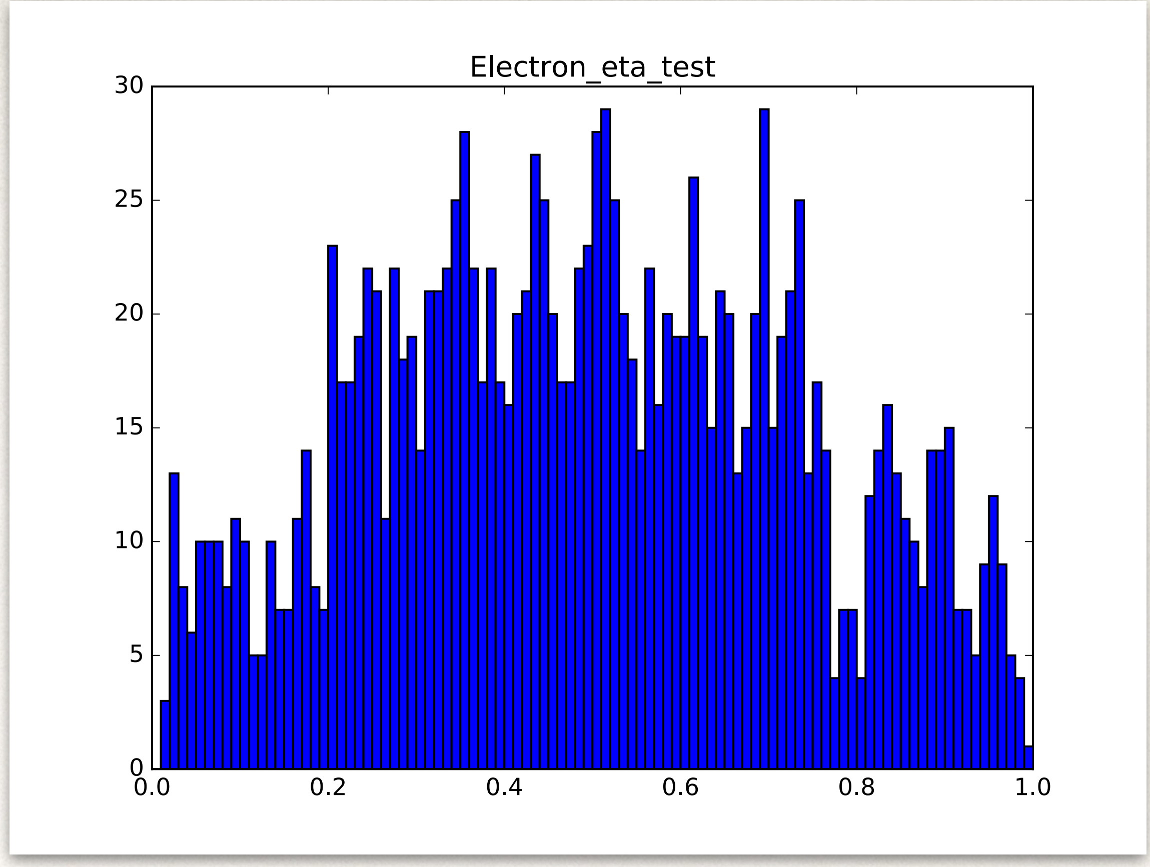


Simple AE

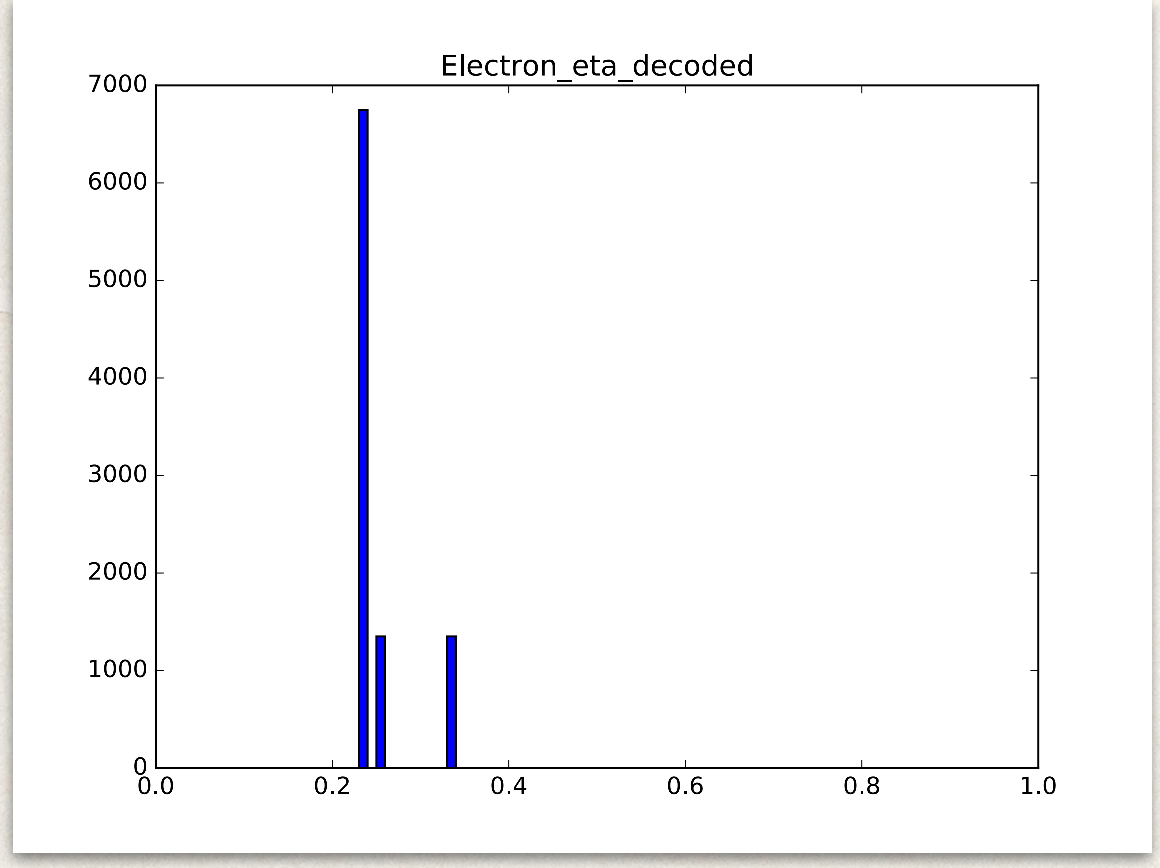
VAE



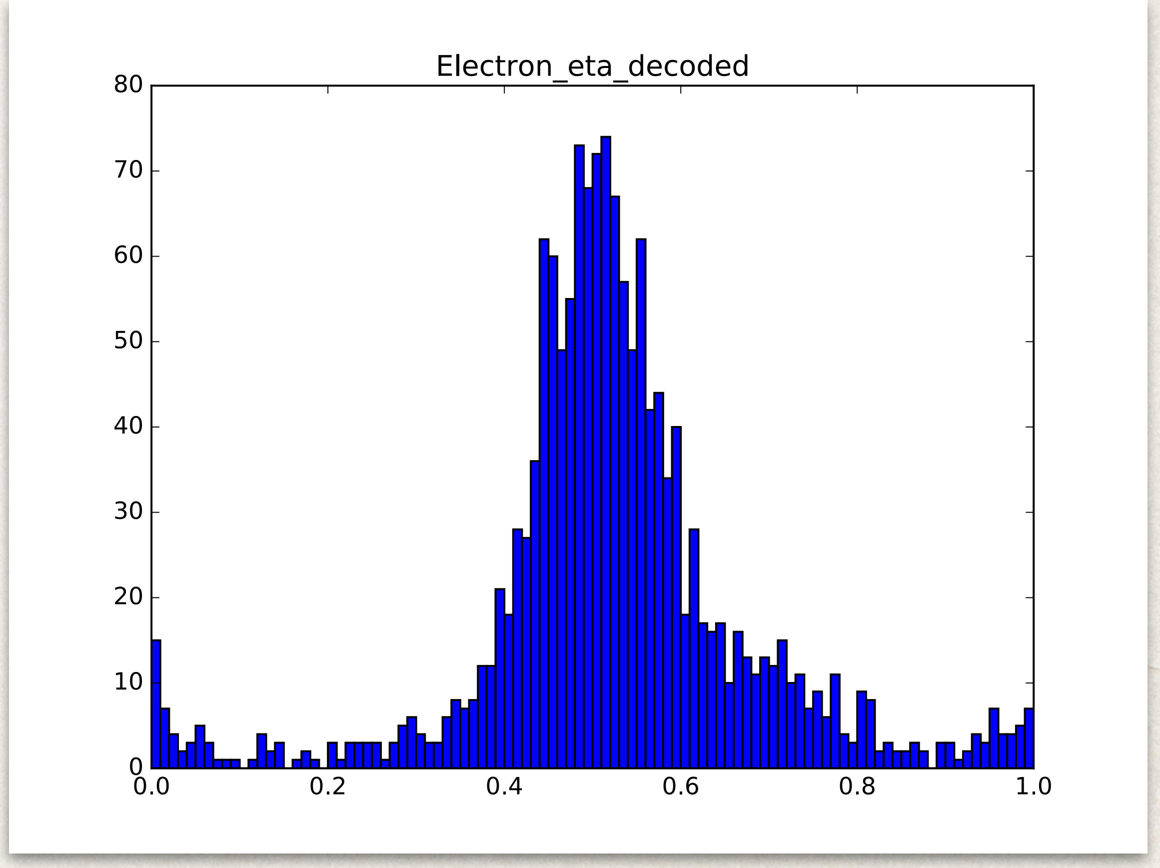


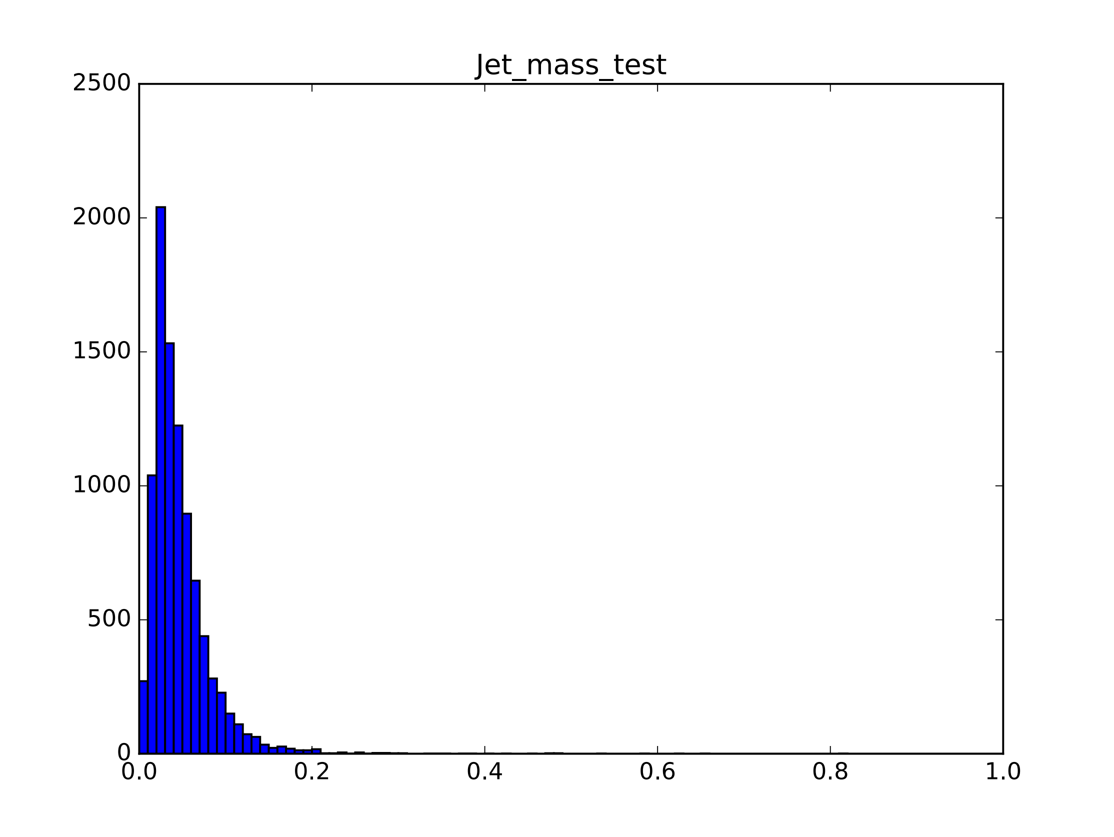


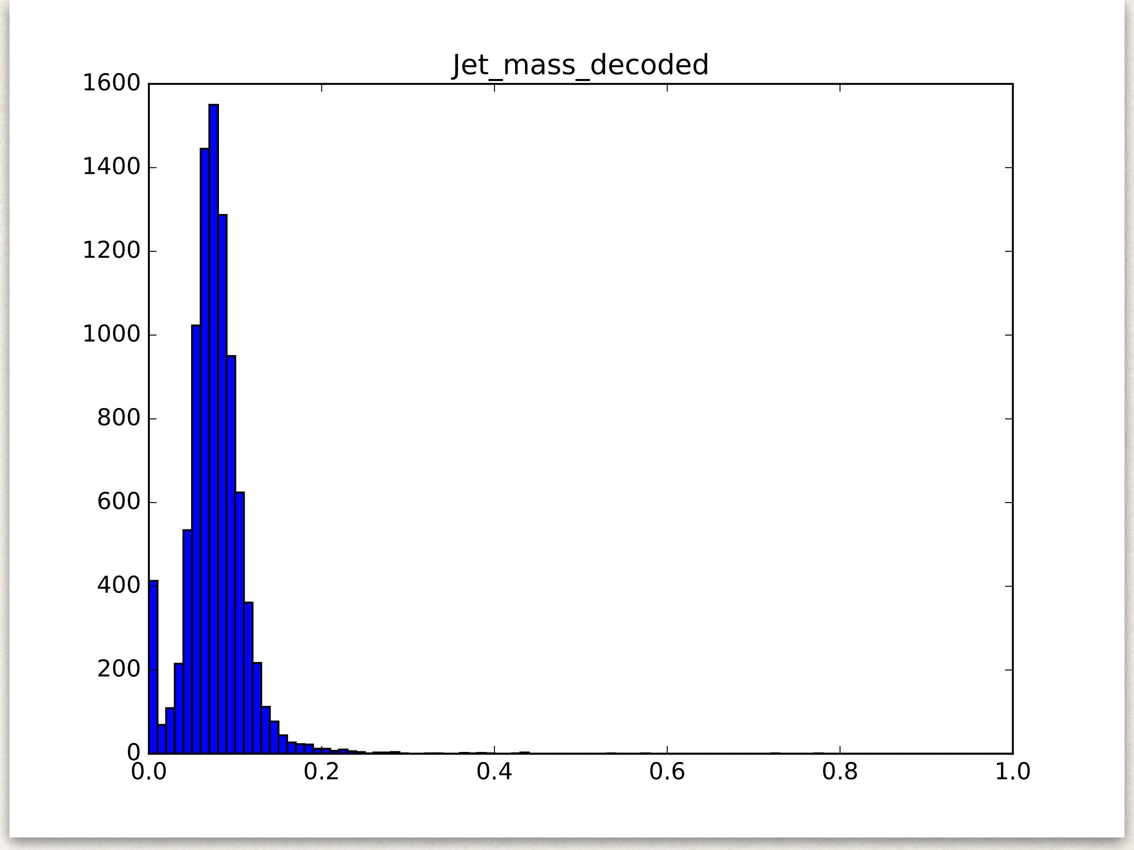
Simple AE

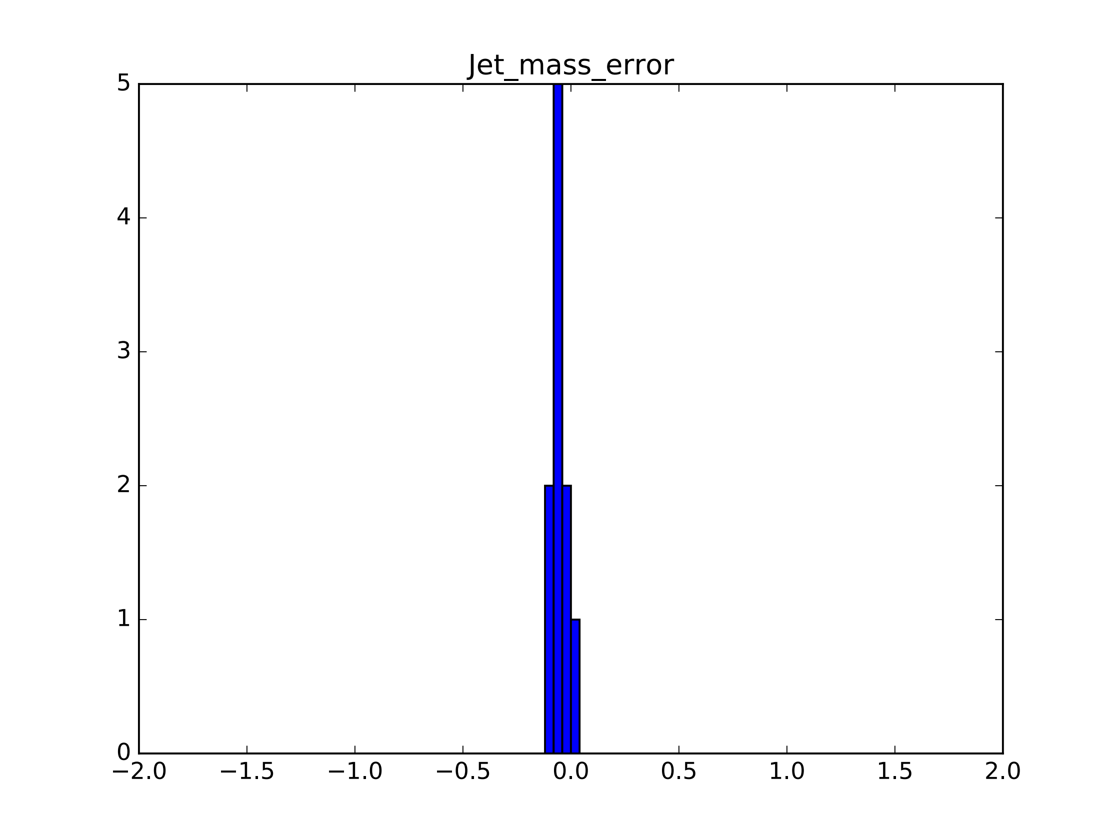


VAE

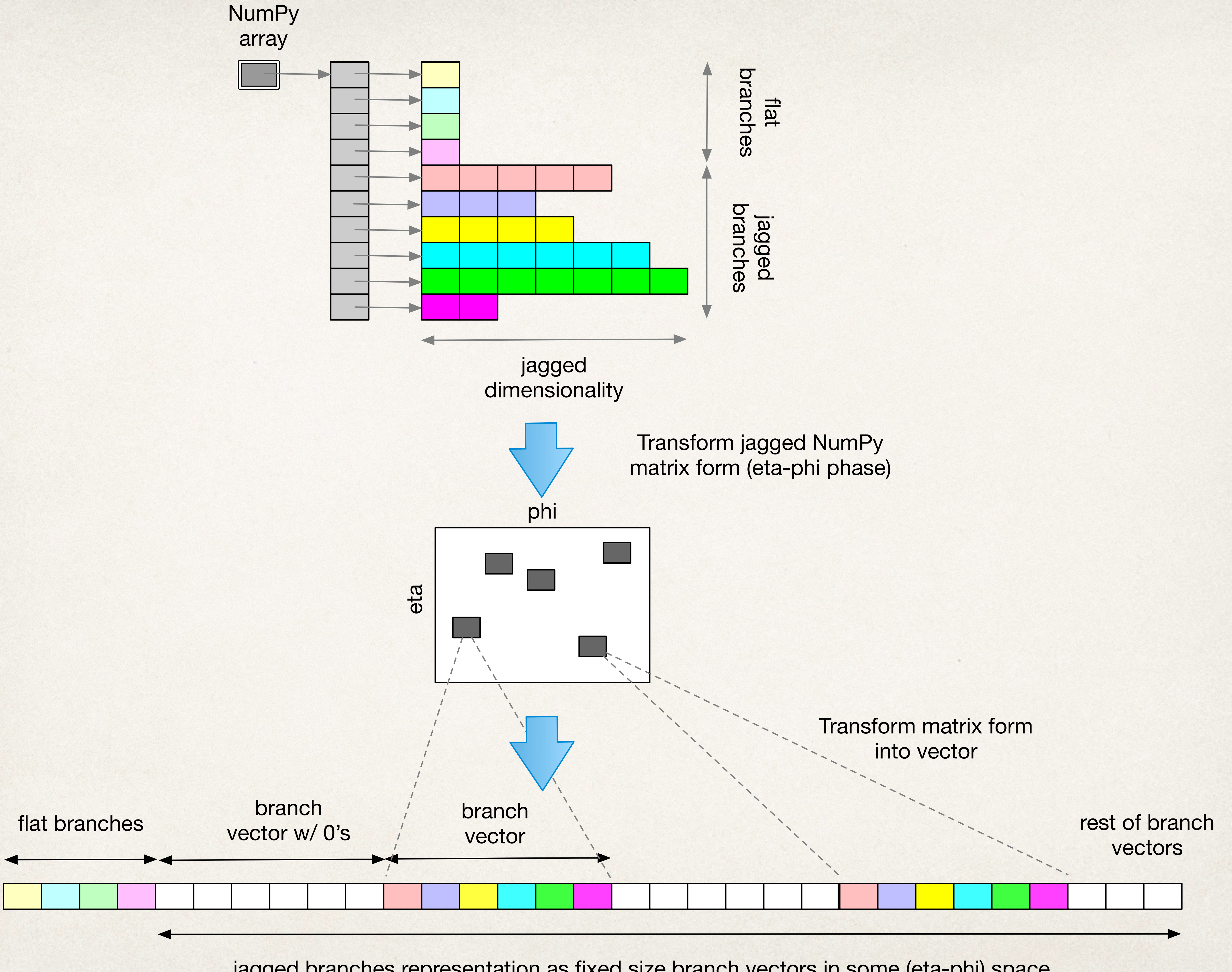








Some distributions can be compressed nicely


NumPy
array
flat
branches
jagged
branches
jagged
dimensionality
Transform jagged NumPy
matrix form (eta-phi phase)
phi
eta
Transform matrix form
into vector
flat branches
branch
vector w/ 0's
branch
vector
rest of branch
vectors
jagged branches representation as fixed size branch vectors in some (eta-phi) space

Challenges

- Data transformation, padding vectors:
  - what to use for padding values (0, -1 or ???) since any real/int numbers are possible
  - padded values introduce skewness in distribution which affect learning and become event dependent
  - what is a largest dimension of jagged arrays across events and/or physics samples, i.e. can't apply one set of parameters to another input data
- Data transformation, matrix representations:
  - we may project jagged values in some (eta-phi) space where each "pixel" represent values of jagged vectors
  - what's an optimal granularity level for matrix, e.g. 100x100 space with 100 jagged attributes represent 1M attributes per event (huge sparse matrix) which significantly affects training time
  - what to do with overlap usage of matrix cells, e.g. different object may be projected into the same cell
- Training is not an issue once we know "correct" distribution of a given attributes (two-pass approach)

Summary

- Initial attempt to apply AE to NANOAOD sample outlines a very challenging problem
  - major size factor in our data is concentrated in jagged arrays (should we compress whole data or its individual parts)
  - proper data transformation is a key to success (VAE shows some promise), looking into GAN (Generative Adversarial Networks)
  - need more time to investigate matrix approach (training time significantly increases to due to large input sparse matrix)
- More generally we need DynamicLayer for NN
  - inputs dimension changes from event to event
  - DynamicLayer should accommodate that and map inputs dimension to fixed layer structure (similar as dropouts work)
  - NN should remember DynamicLayer for internal mapping, e.g. encoder should start with DynamicLayer to map inputs into fixed Layer and decoder should perform reverse operation

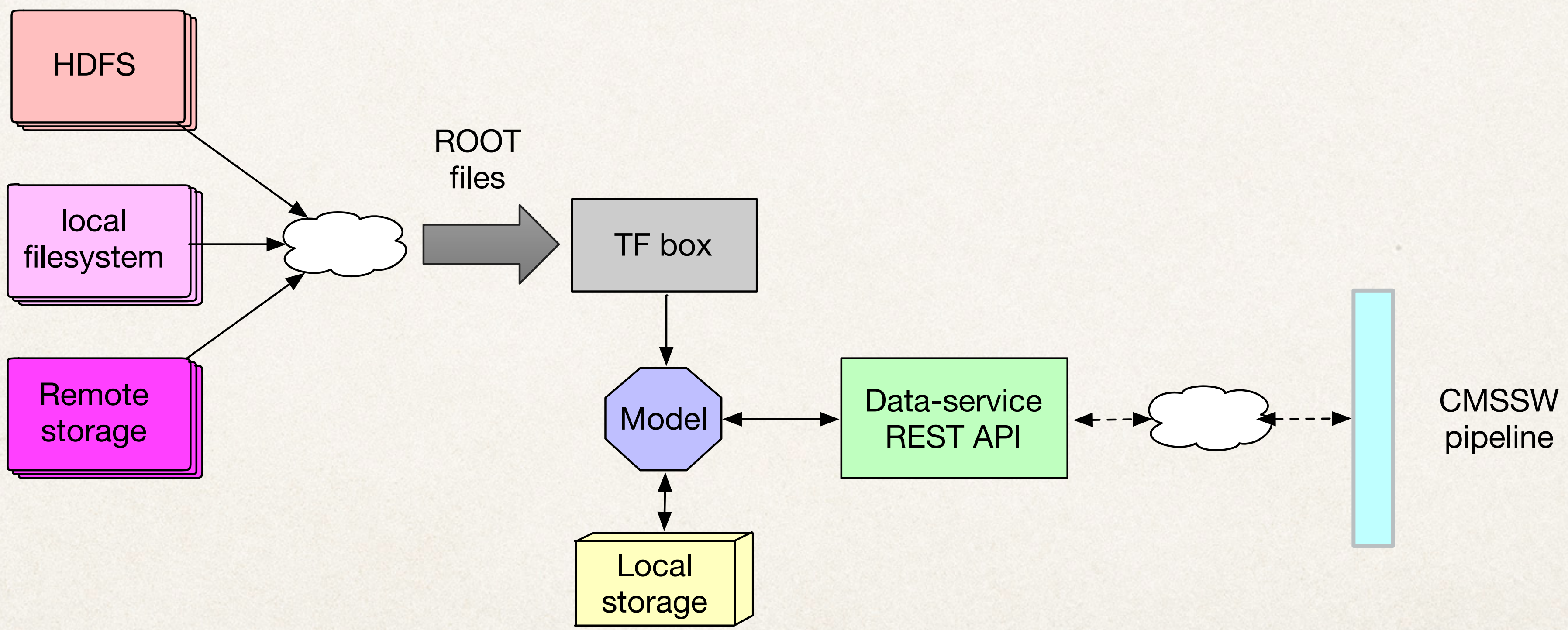
Event Classification update

Project scope

- Aim: Event Classification as unsupervised problem
  - start with simple supervised use case to understand data formats, data representation, workflow, etc.
- Learn how to deal with large data volume (TB/PB scale) and continuous training
- Learn data representation, e.g. AE studies can be applicable to modeling
- Organize workflow for DL training and inference
  - provide demonstrator for building a model and serving prediction (not necessary Event Classification per-se)

DL as a Service

- Serve ML/DL via REST API via high efficient data transport layer (e.g. uproot for reading input data and protobuf for client-server communication
- Read HEP data remotely (via uproot+XRootD), train the model and allow clients (e.g. CMSSW) to communicate with service via HTTP/gRPC to get predictions
- Separate training, model serving and processing pipelines



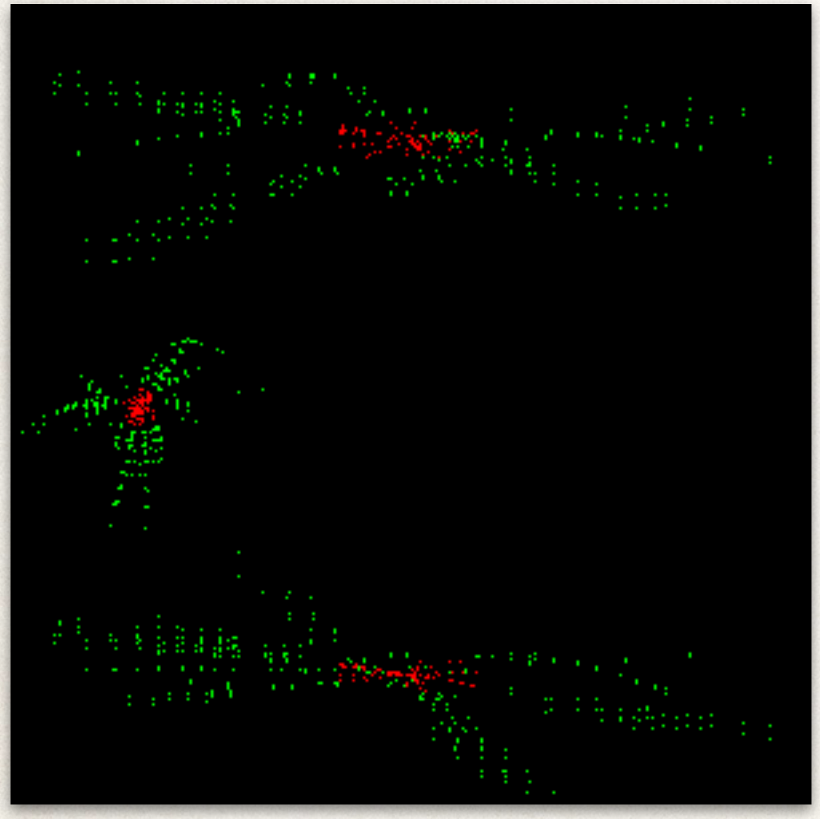
https://github.com/vkuznet/TFaaS

Approach

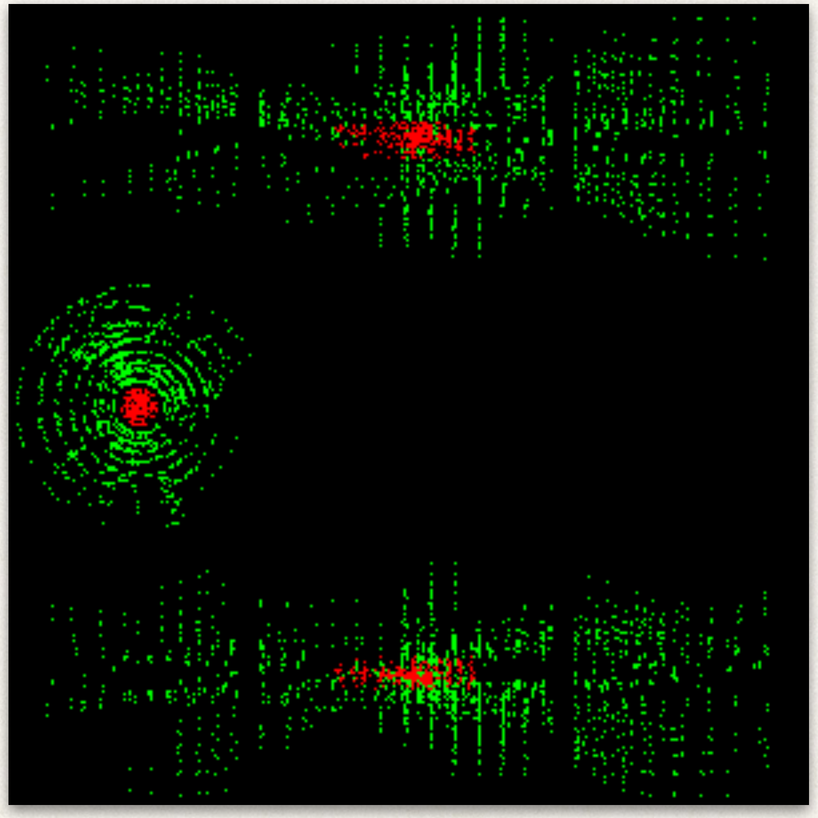
- Extract Pixel/Silicon hits in global coordinate frame
  - use GEN-SIM-RECO files and custom EDAnalyzer to loop over tracks and get their hits (ultimately do not need tracking, use detector hits/clusters)
    - use RelVal samples Jpsi, QCD, Higgs, etc.
  - convert x,y,z hits into PNG image (yz, xy, xz projection in a single image)
- Apply CNN to perform event classification
  - use http://www.fast.ai framework (PyTorch) for CNN; train on Tesla K40c
  - use world's best CNN model, e.g. ResNet, VGG
- Pipeline automation:
  - generate images, train CNN model, serve prediction for remote clients

Events example

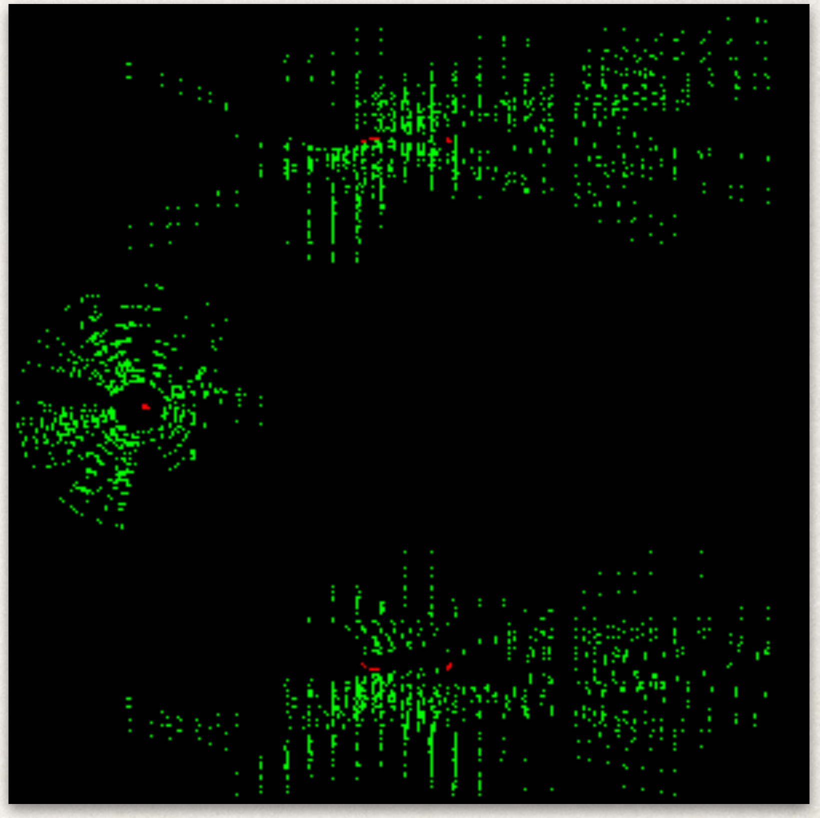
JpsiMuMu



QCD



Higgs200Taus



Extract hits from tracking and build 3D images representing physics events in yz, xy, xz planes

Model results

Sample	Accuracy	Confusion matrix	Accuracy	Confusion matrix	Accuracy	Confusion matrix
Jpsi vs QCD	95%	[[1815 72] [89 1700]]	90%	[[1696 191] [163 1626]]	99%	[[1879 8] [14 1775]]
Higgs vs Jpsi	93%	[[1749 74] [172 1681]]	91%	[[1666 157] [178 1675]]	95%	[[1720 103] [68 1785]]
Higgs vs QCD	72%	[[1390 429] [584 1197]]	76%	[[1415 404] [448 1333]]	88%	[[1614 205] [224 1557]]
Higgs vs Jpsi vs QCD	74%	[[1099 214 494] [179 1718 14] [483 43 1232]]	79%	[[1240 128 439] [172 1735 4] [411 11 1336]]	88%	[[1467 62 278] [125 1786 0] [150 0 1608]]

ResNet34 **ResNet50** **ResNet50 modified**

Summary

- Original results on HEP image classification are encouraging
  - generate HEP images (events) via CMSSW EDAnalyzer plugin (part of event reconstruction)
  - can start studies of major trigger lines: single muon, double muon, jets, etc.
- Have implemented core of TFaaS project
  - data serialization, server and client
  - working on TensorFlow model integration
  - work with Bologna student who is working on S/B hadronic top analysis to test pipeline automation