Adding distributed training and scalability to UCluster

A particle clustering method

UPPSALA UNIVERSITET

Olga Sunneborn Gudnadottir, Daniel Gedon, Colin Desmarais, Karl Bengtsson Bernander, Raazesh Sainudiin, Rebeca Gonzalez Suarez

Uppsala University

CHEP 20-05-21

Distributed training and scalability

Scope of the project

Goal: Make UCluster model fully scalable with feasible training times

- UCluster: a general Unsupervised Clustering algorithm consisting of a Deep Neural Network and clustering
- Fully scalable: usable with any amount of input data
  - Important for the large datasets of the LHC
  - Accomplished with distributed deep learning and selective data loading

UCluster

Vinicius Mikuni and Florencia Canelli
git | arXiv:2010.0710

- Unsupervised Clustering of particle physics data
  - Graph neural network for classification (ABC net) creates latent space in which particles with similar properties are close to each other
  - Each event/particle assigned to a cluster (K-means)
  - Combined classification and clustering loss allows for training the DNN parameters as well as the cluster centroids
- General method, can be used as part of any task for which unsupervised clustering is useful
  - Original paper: multiclass classification of high-p_T jets and anomaly detection for new physics searches

Multiclass classification

Data and training

Task: match high p_T jets to the particle they originate from (top quark, W boson, Z boson)

- UCluster:
  - Classification task: mass classification
    - Match list of jet constituents with the invariant mass (interval) of the jet
    - Creates latent space in which jets from same particle are close

Data

Access it here

Simulated LHC data at $\sqrt{s} = 13$ TeV

High pT jets from
W bosons, Z bosons,
top quarks

Anti-kT with R=0.8

For each jet: a list of up to
100 constituent particles

300 000 jets in train sample

140 000 jets in validation sample

UCluster result

81% Clustering accuracy (Hungarian method)

UCluster

Potential and limitations on the way

Potential

- Training on full LHC data sets
  - UCluster could be used directly in data
    - Multiclass classification for data driven background estimation
    - Anomaly detection for new physics searches

Limitations

- Data samples can be 100s of TB
  - Full data doesn't fit in single GPU memory
- Training time could become prohibitively long

- Distribute the training over multiple GPUs
  - Speeds up training by effectively increasing batch size
- In addition, utilise a file format that allows for reading only parts of the data at once
  - Only need to fit a part of the data in each GPU memory at a time.

Distributed UCluster

Preparations

- All development on Databricks Lakehouse platform
    - Code developed in notebooks
    - Easily create GPU clusters
        - Apache Spark to distribute computing tasks between nodes (GPUs in our case)
        - Fault tolerant distributed file system DBFS
- Three different incarnations of the code:

1. Original code (TensorFlow v1) but run in a Databricks notebook
2. Original code migrated to TensorFlow v2 and run in a Databricks notebook
3. TensorFlow v2 code with distributed training on a GPU cluster with 2 GPUs

Distributed UCluster

Implementation

TensorFlow v2 code with distributed training

- Use Horovod (open source) framework to distribute the deep learning
    - Data copied to each GPU
        - Distributed optimisation by accumulating gradients across all the GPUs
- Run the training on n GPUs
    - Increase learning rate by factor n
- Once the Horovod framework has been set up, it can be used for any number of GPUs (including one)

Distributed UCluster

Training

- All models
  - Pre-trained for 20 epochs
  - Trained for total of 100 epochs
  - Learning rate γ started at 0.001 and decreased by a factor 2 every three steps until it reached 10^{-5}
  - Batch size 1024

GPU cluster

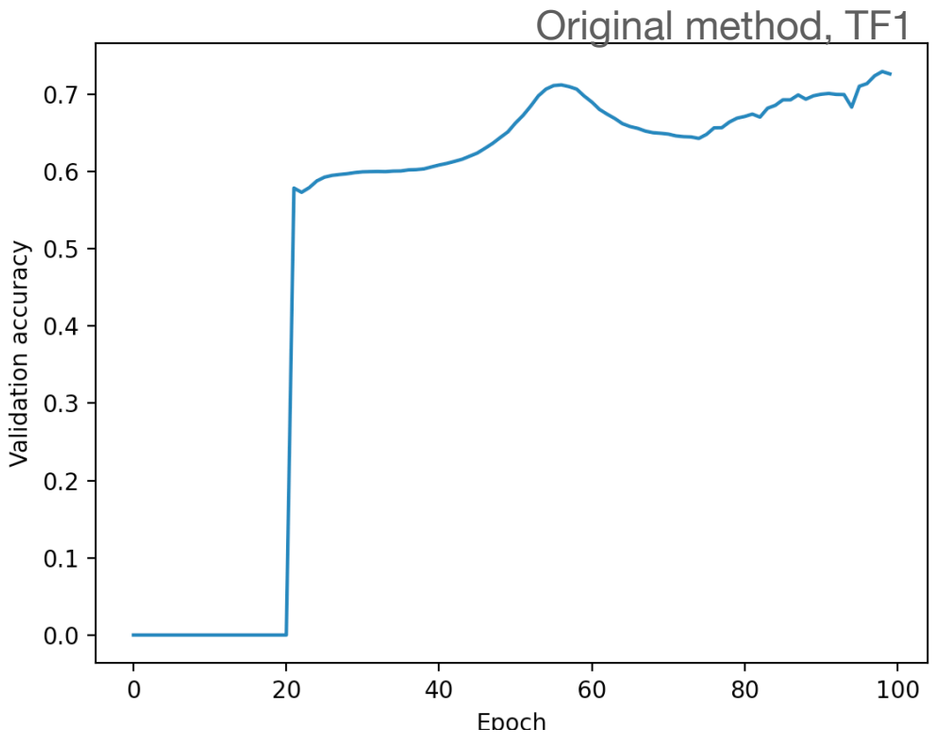
g4dn.xlarge
2 NVIDIA T4 GPUs
16 GB each
Apache Spark 3.0.1

	Original code	TensorFlow 2 version	Distributed training, 2 GPUs
Training time	4 mins/epoch	4 mins/epoch	2 mins/epoch
Learning rate	γ	γ	2γ

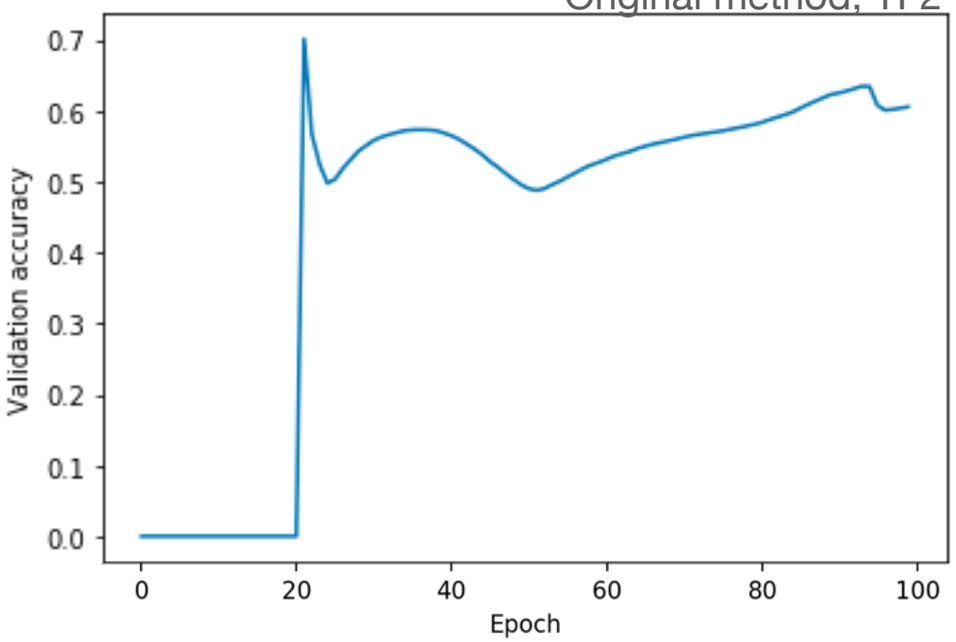
Results

Comparing distributed UCluster to original

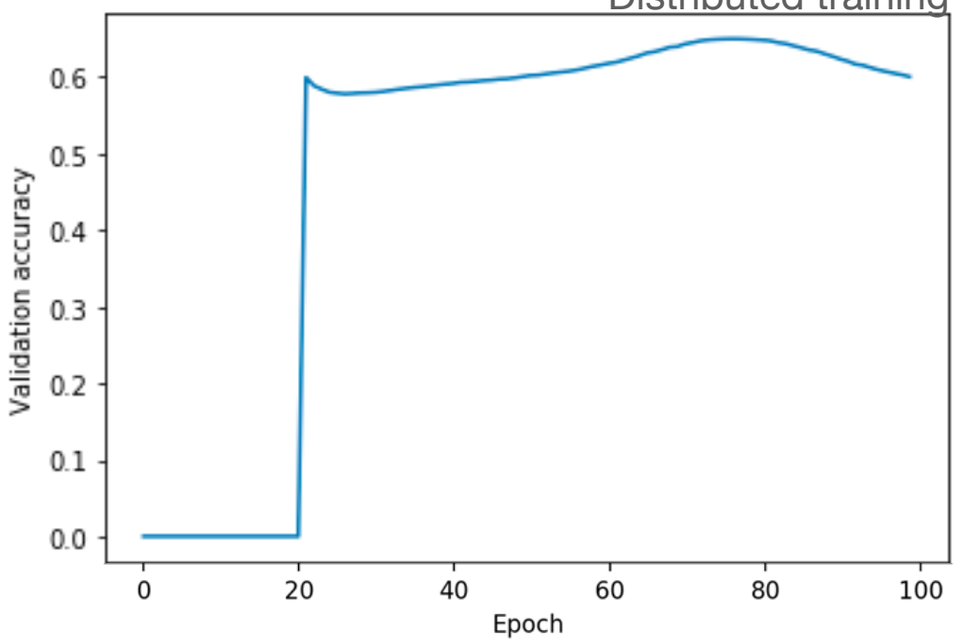
Original method, TF1



Original method, TF2



Distributed training



Accuracy calculated with Hungarian method

Accuracy

- Same training behaviour for all three models
  - Original code
  - TensorFlow v2
  - Horovod (distributed training)

Conclusions and outlook

Path to a fully scalable method

- Scalability
  - Distributed learning seems to work fine
  - Parquet files for selective data loading
    - Idea: load only the data needed for the current batch onto GPUs
    - Not straightforward:
      - Currently all data loaded onto all GPUs (Horovod default)

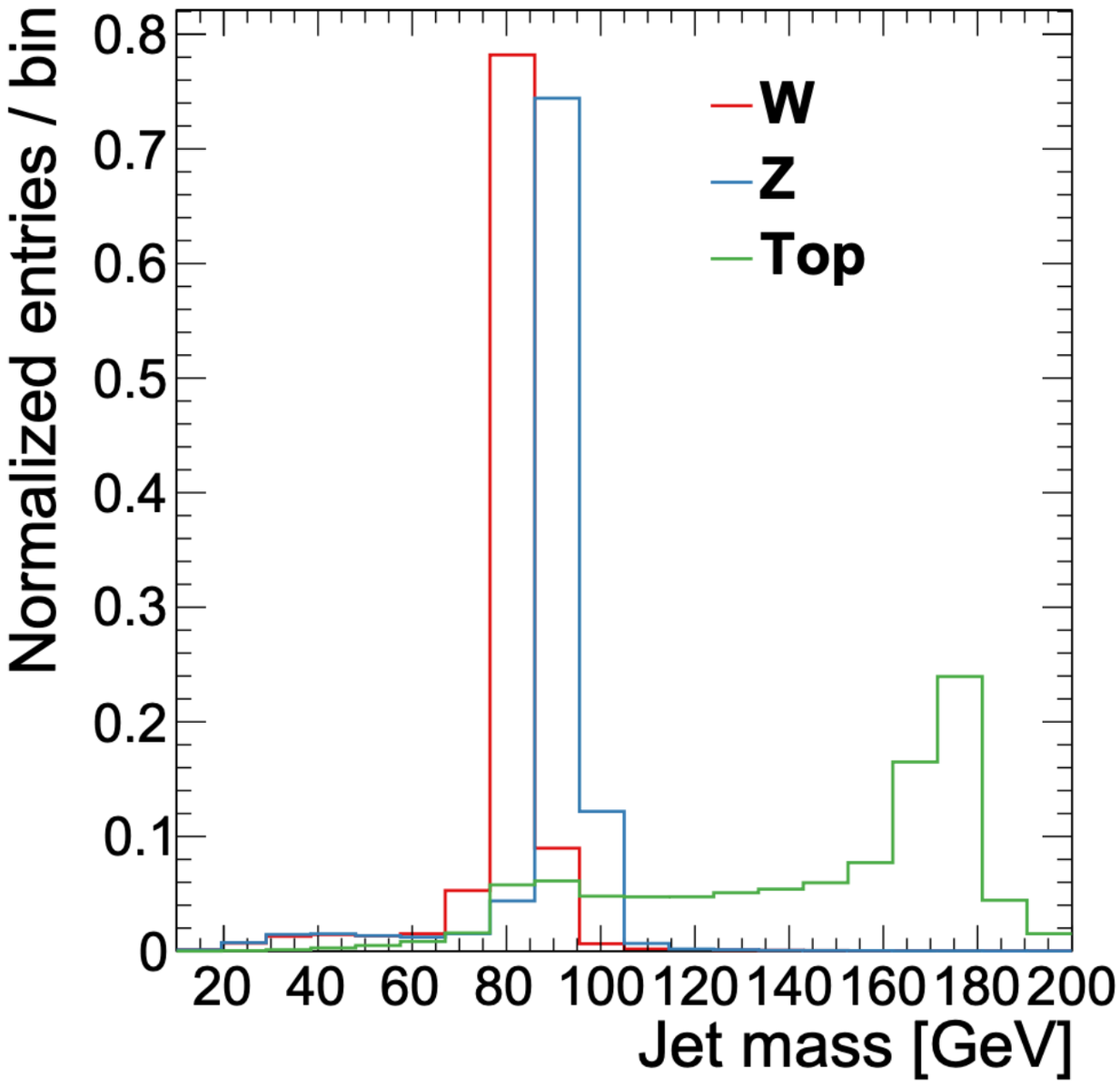
- Accuracy
  - Hyperparameter tuning (systematic search?)
    - Combination of learning rate, relative importance of loss function contributions and the “inverse temperature” allowing the cluster centroids to move?
    - Reproduce original paper accuracy
- Accurate and scalable
  - Once the model is able to ingest 100s of TBs of data, will it still be as useful?

Acknowledgements

This work was partially supported by:

- The Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation
- Combient Mix AB through an Industrial Data Engineering Science Mentorship
- Databricks University Alliance and AWS through distributed computing and storage infrastructure

Jet mass distributions



Simulation parameters

```
parserdict = {'max_dim': 3, #help='Dimension of the encoding layer [Default: 3]')
              'n_clusters': 3, #help='Number of clusters [Default: 3]')
              'gpu': 0, #help='GPU to use [default: GPU 0]')
              'model': 'gapnet_clasify', #help='Model name [default: gapnet_classify]')
              'log_dir': 'log', #help='Log dir [default: log]')
              'num_point': 100, #help='Point Number [default: 100]')
              'max_epoch': 100, #help='Epoch to run [default: 200]')
              'epochs_pretrain': 20, #help='Epochs to for pretraining [default: 10]')
              'batch_size': 1024, #help='Batch Size during training [default: 512]')
              'learning_rate': 0.001, #help='Initial learning rate [default: 0.01]')
              'momentum': 0.9, #help='Initial momentum [default: 0.9]')
              'optimizer': 'adam', #help='adam or momentum [default: adam]')
              'decay_step': 500000, #help='Decay step for lr decay [default: 500000]')
              'wd': 0.0, #help='Weight Decay [Default: 0.0]')
              'decay_rate': 0.5, #help='Decay rate for lr decay [default: 0.5]')
              'output_dir': 'train_results', #help='Directory that stores all training logs and trained models')
              'data_dir': os.path.join(os.getcwd(),'06_LHC_TF1', 'h5'), # '../h5', #help='directory with data [default: hdf5_data]')
              'nfeat': 8, #help='Number of features [default: 8]')
              'ncat': 20, #help='Number of categories [default: 20]')
             }
```

- Inverse temperature α starts at 1 and increases linearly by 2 every epoch after pretraining.

- Proportionality constant between clustering loss and classification loss, β =10

Features

Table 1: Data features used as input in the UCluster method.

Variable	Description
$\Delta\eta$	Difference in pseudorapidity η between jet constituent and jet.
$\Delta\phi$	Difference in azimuthal angle ϕ between jet constituent and jet.
$\log(p_T)$	Logarithm of the constituent transverse momentum p_T.
log(E)	Logarithm of the constituent energy E.
$\log(\frac{p_T}{p_T(\text{jet})})$	Logarithm of the constituent transverse momentum relative to the jet transverse momentum.
$\log(\frac{E}{E(\text{jet})})$	Logarithm of the constituent energy relative to the jet energy.
ΔR	Distance defined as $\sqrt{\Delta\eta^2 + \Delta\phi^2}$
PID	Particle ID [16]