

Deployment of ML in Changing Environments

4th October 2022

Fast Machine Learning for Science Workshop 2022
Southern Methodist University

M. Barbone¹, C. Brown¹, B. Radburn-Smith¹, S. Summers², A. Tapper¹

¹ Imperial College London

² CERN

Outline

Changing Environments

Retraining Old Networks

Uncertainty Quantification

Continual Learning

Embedded Continual Learning

Summary



- We will use the CMS Level 1 Trigger as an example throughout
- And in particular the E2E Vertex Finding: see [Chris' talk yesterday](#)

Changing Environments

There are many situations in which a **Machine Learning (ML) system** is **deployed** in an **evolving environment**

- This can be with **changing conditions** which leads to a **target dataset varying significantly** from the **training dataset**

In **particle physics experiments** we tend to **train ML systems** on **large Monte Carlo simulation datasets** and then **deploy the systems** on **real data** which can have **variable conditions**

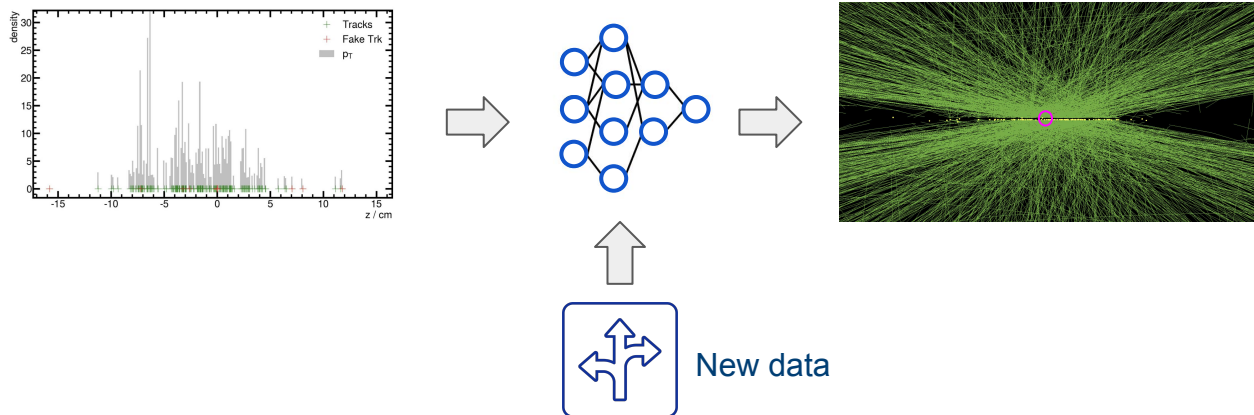
- The **detector itself** might **not respond** in the **desired manner**, alignment and calibrations may need to be performed on the data
- The **Level 1 Trigger (L1T)**, which selects 100k collisions for further selection at a rate of 40 MHz, also follows this paradigm
 - For the **Primary Vertex reconstruction** we rely on the **Silicon Tracker performance** and subsequent **L1 Tracker Tracks** created

Possible Solutions

There are a various **different options** in order to **cope** with the **changing environments**, for example:

- Offline retraining and redeployment
- Uncertainty quantification
- Continual learning

Each option has its own **advantages** and **disadvantages**



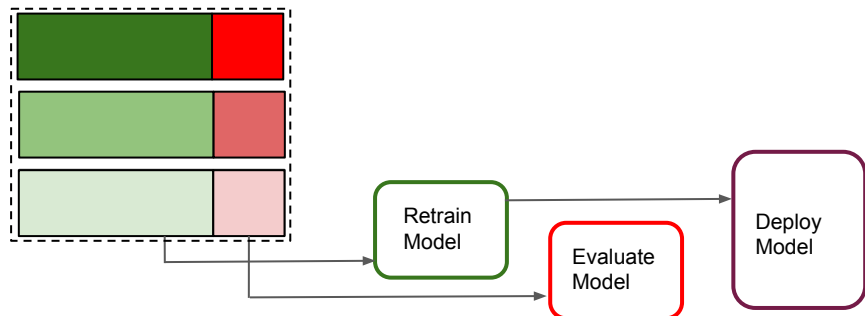
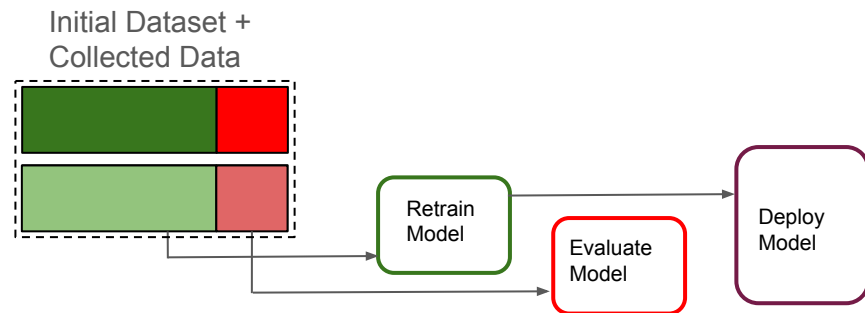
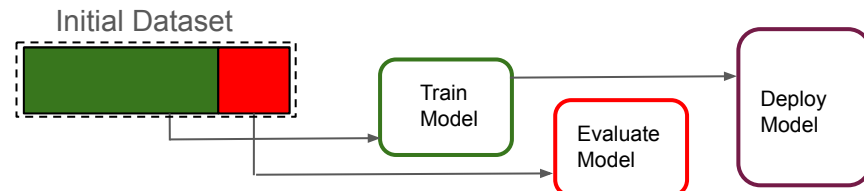
Retraining Old Models

Regularly retrain and redeploy models on large datasets

Can **automate** the **process**, based on some **metric of performance** of the deployed models or **after set time**

Advantages:

- Always **aware** of what the **models** are being **trained on**
- Can **tune datasets** based on **understanding** the **changing environment**
- Can **store previously trained models**, easy to roll-back to known models if performance drops or change in environment is rectified

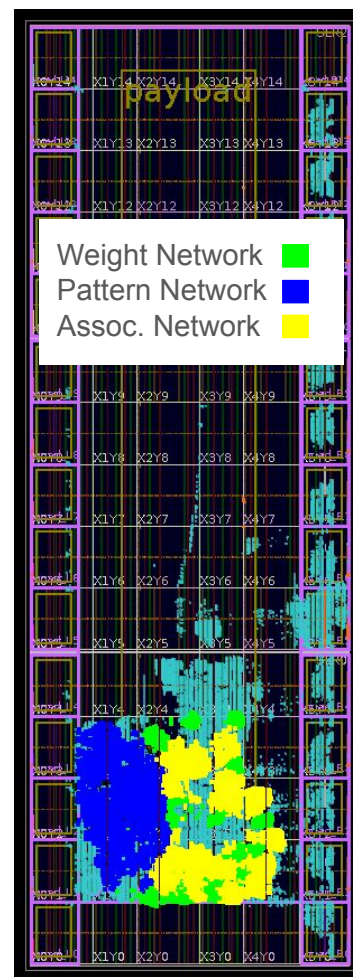


Retraining Old Models

Disadvantages:

- Retraining is **computationally** and **time costly**
- Retrained model might have **different footprint** to original model (pruning or quantisation might change)
- **Catastrophic forgetting** can occur where model performance on previously learned task reduces when new data is given
- Need to **store large amounts of data** for retraining
- **Old data** could become **unavailable**

- For the **L1T** we must fit the model onto **limited resources** on the Field Programmable Gate Array (**FPGA**) and therefore need the **footprint to be stable**
- We would need to **store training data** from L1 Trigger inputs and label in offline analysis
- Model could “**forget**” and **performance** on **previous detector configurations** would be **dramatically reduced**



From [Chris' talk](#)

Uncertainty Quantification

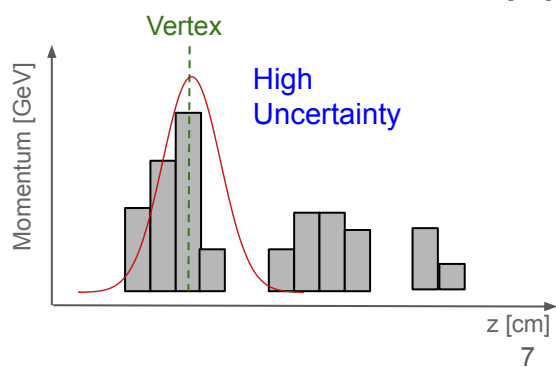
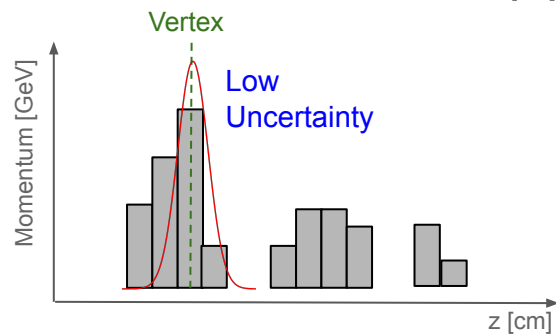
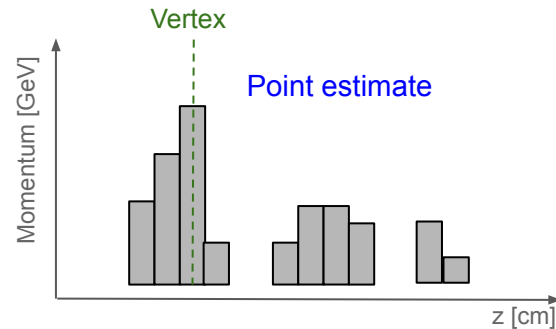
Learn (or quantify) the **uncertainty** of a **model** and use this as a **handle** for our **confidence** in the **model**

This **uncertainty** can be either used by **downstream users** or as a **flag** to **retrain** the model

Can only inform us on how confident we should be in our model, **doesn't give insight into why**

Same issues as retraining models, **core model still needs** to be **retrained** but have an **understanding of when**

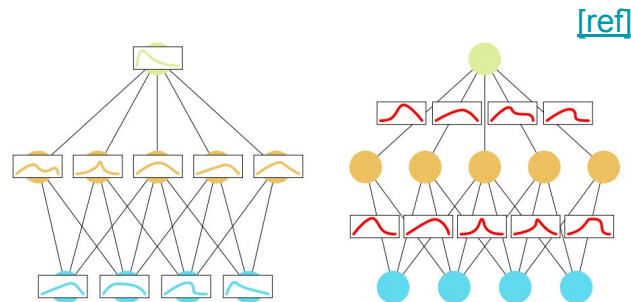
- L1T we could **assign** an **uncertainty** to the **Primary Vertex location**
- This uncertainty could be used to **tune** the **size of the association window** and the **region of interest for tracking** at the **High Level Trigger**



Uncertainty Quantification

Bayesian Neural Networks

- **Network parameters** are a **gaussian distribution** with **mean** and **standard deviation**, at **inference time**: **sample** from these distributions to give a mean and standard deviation on our result
- Most **robust** → **can't do this quickly** [\[ref\]](#)

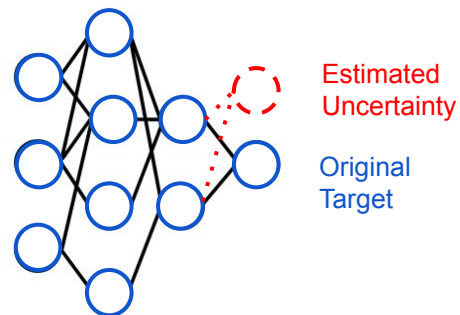


Auxiliary Models

- **Separate model trained** on **intermediate model outputs** in order to **predict uncertainty** of the model → **difficult to train** (what is our target) but can be a **small model**, **quick** at inference time

Additional Target Feature

- **Train model** with **another output** to give **predicted uncertainty** in the model → can **impact original model training** but **cheap** in **inference time**, **no additional latency**



Uncertainty Quantification

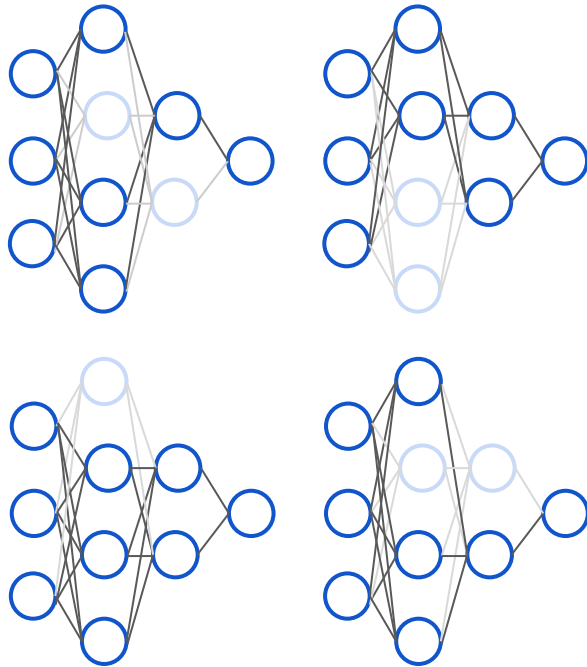
Dropout

- Perform multiple inferences of the a network with different dropout in each layer in each inference and take mean and standard deviation of the output
- Very costly at runtime, multiple inferences needed

Ensemble techniques

- Use multiple models trained on same data with same target and use mean and standard deviation
- Again very costly at runtime, need multiple models implemented and inferences

➤ Further investigations are needed to see which technique (or collection of techniques) would be most beneficial for the L1T



Continual Learning (CL)

Concept: Train a model with a continuous stream of data

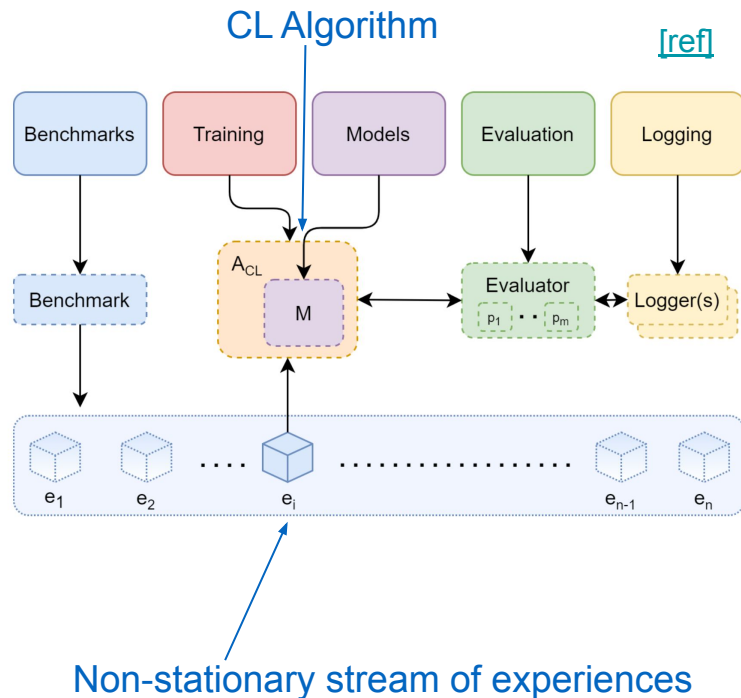
Learns from a sequence of partial experiences rather than all the data at once

- We don't have access to all this data and don't know how it will change

CL may have to deal with scarce data, catastrophic forgetting, or data distribution shifts

Also called Lifelong Learning, Incremental Learning, Never Ending Learning

- Online Learning is a special subset of CL with model updates performed on single data points, instantaneously



Continual Learning

Advantages [\[ref\]](#)

- Avoids catastrophic forgetting → initial training is not disregarded
 - Adapts to a changing data stream → don't need to quantify how the environment changes
 - Don't need to store previous training data
 - Reduces computational costs: don't need to retrain on all the previous data → good for embedded applications
- With the L1T we could have the initial training on a large MC and then have subsequent further learning on smaller datasets taking into consideration the changes in detector performance
- The CL algorithm can explore changing the network size within some footprint limit: switching on/off pruning or quantisation
- Continuous integration style: automatic training and model deployment

Continual Learning

Disadvantages

- Some methods see the **network change dynamically** → in a fast ML setting this makes **deployment in HW difficult**
- For **supervised learning** need a ~continual stream of **labelled data** which **might not be accessible**
- We have to be aware that the **network size can change**

- For the **L1T** there are only **certain times** that the **FPGAs** can be **updated** with a new network; for example in between LHC proton fills
- We have to make sure that there is **space** on the **FPGA** for the **new network footprints**

Continual Learning Techniques [\[ref\]](#)

Replay

- Interleave previous data with new data as model is trained → can lead to large storage of old data
- Alternatives include the use of a second generative model to generate data based on the original data distribution

Regularisation

- Add an additional regularisation term on existing model weights which prevents them drastically changing as new information is provided → which could also limit performance if applied on the network (else could be metadata outside of the network)
- Prevents catastrophic forgetting → can understand task relevant weights within a model

Dynamic Architectures

- Add additional parts of the model to accommodate new tasks or information
- Very flexible but difficult in embedded applications unless larger model footprint is pre-defined

➤ Large flexibility in CL paradigm choice at L1T

Embedded Continual Learning [[ref](#), [ref](#)]

Possible extension is to run the CL on **embedded devices** themselves, e.g. on Firmware

Gives **additional memory** and **time constraints**:

- Each example is seen only once unless explicitly stored
- Limited number of examples stored for replay
- Training time limited by **processing power** and applications constraints
- No control on the ordering of the examples

Literature experimental results show that CL is **viable** on **embedded systems**

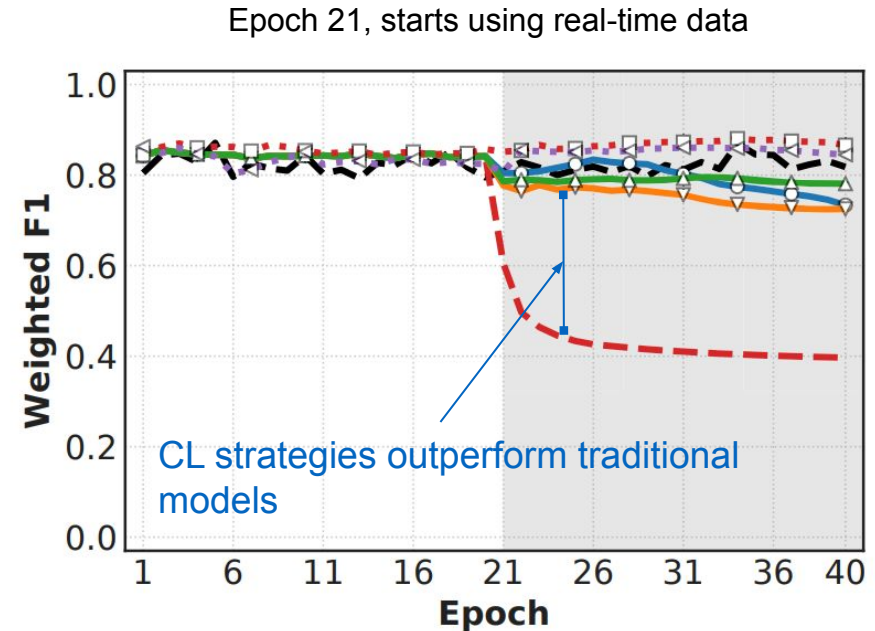
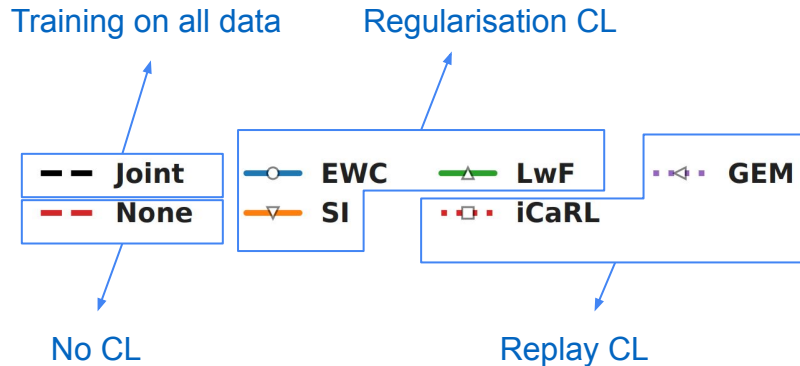
- Performs significantly better than traditional networks
- More expensive strategies (replay) usually give better accuracy
- Less expensive strategies **outperform traditional models** by a great margin

- L1T: **Model** can be **updated** with **continual stream** of **L1 data**
- **No data storage**, truly continual learning algorithm
- Possible to not label every experience in the continual stream

Embedded Continual Learning [ref]

Example Result:

- Using **mobile/embedded sensing data** measuring users physical activities to **determine the Human Activity Recognition (HAR)**
- Various **CL strategies** showed **improved performance** compared to not using CL



Summary

One of the **key issues** for **real-time ML** in **any area of science** is in the **differences** between the **training data** and the “**real-world**” **data** which can lead to **performance degradation**

We have described a few of the **various techniques** which can be used to **mitigate** this **problem**:

- Retraining and redeploying models
- Uncertainty quantification
- Continual Learning

- We face this issue in many areas of HEP including at the **L1T**
- The **detector state can change** over time which leads to **different responses** at the L1T
- **Further studies** will now be performed to see if **these techniques** are **viable** for on-detector inference at the L1T