Project: Neural-Network Tagger for Hadronic $V \to q\bar{q}$ Jets

ROC, purity, and applying a trained model to data

Summary. You will build a small neural-network (NN) classifier to distinguish hadronically decaying electroweak bosons ($V = W/Z$) from generic QCD jets using a lepton + large-R jet topology. You will: (i) train a tiny Multi-Layer Perceptron (MLP) on Monte Carlo (MC) truth-labeled jets, (ii) compare the performance of your MLP with a simple cut based selection using ROC and purity curves, and (iii) apply the trained model to data from the Large Hadron Collider to study the large-R jet mass before/after selection.

This is an **applied** exercise: you will leverage standard libraries to *use* NNs responsibly in a physics context. If you are curious about proofs and deeper theory, see e.g. `http://neuralnetworksanddeeplearning.com/` by Nielsen.

Learning goals. By the end you can:

- Engineer basic input features to a neural network and understand what an NN *sees*.
- Train and validate a small MLP.
- Quantify performance with ROC (TPR vs. FPR), AUC, and purity $S/(S+B)$.
- Apply a trained model to data and interpret mass spectra changes.

Data & files provided. All provided material can be found in the course gitlab.

- `jets.csv` — *Data from the ATLAS experiment* (flattened): per-particle rows with columns `#event_id, pid, pt [GeV], eta, phi, E [GeV], mass [GeV]`.
  The data is taken from the CERN Open Data repository, see `https://opendata.atlas.cern/`.
- `pythia.csv` — *MC* (generator-level truth available): same columns plus `v_true` $\in \{0, 1\}$ for large-R jets (1 if matched to a hadronic V).
  The simulation is carried out with the PYTHIA Monte Carlo event generator, see `https://www.pythia.org/`.
- `data-exercise-template.py` Template for the analysis, containing data classes and a cut-based analysis for you to use as a baseline.
- `requirements.txt` A template file for setting up the neccesary Python environment.

The repo also contains the following auxiliary files (in the `aux`-folder) needed if one wanted to reproduce the data file and the simulation file from scratch. They are there for someone interested to get inspiration. They are not needed for you to complete the project, and you can thus safely ignore them:

- `aux/main213.cc` PYTHIA program to generate the Monte Carlo truth file `pythia.csv`.
- `aux/skimevents.py` The CERN Open Data is delivered in a more complicated format, which depends on the CERN ROOT (see `https://www.root.cern`) package. This program reads the relevant datafile and saves the data in the format of `jets.csv`.

Physics summary of data and simulation files Events contain at least one large-R jet (anti-k_T, R=1.0, trimmed) and one charged lepton (e or μ). You should *not* use jet mass or energy as NN inputs to avoid sculpting the mass spectrum; mass is used only for evaluation plots. You do not need to run a jet finding algorithm on data or MC, this has already been done.

Each row in the data and MC files contains the following information:

`event_id`: A unique identifier (integer) of each event. You can use this to split simulation between training and testing samples.

`pid`: Short for "particle identification", tells you what type of object is in this line. It is an integer equal to 90 for a jet, 11 for an electron and 13 for a muon, roughly following the standard convention of the field. You do not need to worry about this, if you use the `Particle` and `Event` classes provided in the template.

`pt [GeV]`: The particle or jet transverse momentum $p_T = \sqrt{p_x^2 + p_y^2}$. In units of GeV.

`eta`: The particle or jet pseudorapidity $\eta = -\log(\tan(\theta/2))$.

`phi`: The particle or jet azimuthal angle. In radians, typically in $[-\pi, \pi)$

`E [GeV]`: The particle or jet total energy. In units of GeV.

`mass [GeV]`: The particle or jet mass. In units of GeV.

`v_true`: Truth information about the origin of the jet in Monte Carlo. It is 1 if the jet came from a vector boson, 0 otherwise.

Reminder on ROC and purity measure Let $y \in \{0, 1\}$ denote the true label of the *leading* large-R jet in an event (1 for $V \to q\bar{q}$-like, 0 otherwise). For a selection (cuts or NN threshold) define:

$$\text{purity} = \frac{S}{S + B}$$

where S (B) is the number of selected signal (background) jets among MC test events, and S_0 (B_0) are totals before selection. Thus one can also compute a *baseline purity* on the MC events before applying any cuts:

$$\text{purity}_0 = \frac{S_0}{S_0 + B_0}$$

The ROC uses

$$\text{TPR} = \varepsilon_S, \qquad \text{FPR} = \varepsilon_B,$$

where:

$$\varepsilon_S = \frac{S}{S_0}, \qquad \varepsilon_B = \frac{B}{B_0},$$

swept by varying a threshold t on the NN score $\in (0, 1)$.

Tasks

A. Cut-based baseline (start here)

A1. Study the cut-based selection. You are supplied with a baseline in `data-exercise-template.py`. It implements a simple, physics-driven "cut-based" selection where the *leading* large-R jet j and *leading* lepton ℓ:

$$p_T(\ell) \geq 50\,\text{GeV}, \qquad p_T(j) \geq 250\,\text{GeV}, \qquad \Delta\phi(j, \ell) \geq 2.4.$$

The goal of this task is to **do better than this baseline** in terms of purity and ROC using a neural network, and plot them against each other. You may want to try and optimize the baseline. This can be done, and you can obtain a better purity from a cut-based analysis only. If you do that, keep the above as the baseline and show the impact.

A2. Evaluate on MC (truth known). Report purity $S/(S+B)$ of the cut-based analysis and any improvements. Compare to the no-cuts purity $S_0/(S_0+B_0)$. You may also study ε_S, ε_B.

A3. Apply to data (truth unknown). Plot the large-R jet mass distribution (leading jet) *before* and *after* the cuts, density-normalized. Comment on the change in the W/Z mass region.

B. Train a tiny MLP

B1. Feature engineering. For each event, build a feature vector *without* mass or energy, consisting of for example (10D)

$$x = (p_T^j,\ \eta^j,\ \phi^j,\ p_T^\ell,\ \eta^\ell,\ \phi^\ell,\ \Delta\phi(j,\ell),\ \Delta R(j,\ell),\\ p_T^j/p_T^\ell,\ (p_T^j - p_T^\ell)/(p_T^j+p_T^\ell)).$$

B2. Split MC by event id. Make disjoint train/validation/test sets by grouping on `event_id`. This avoids leakage from multiple objects in the same event.

B3. Network and loss. Use a small MLP. You can start with one similar to the one we used in `two-moons.ipynb` and evolve to e.g.

$$\text{Linear}(10\to32) \to \text{ReLU} \to \text{Dropout}(p=0.1) \to \text{Linear}(32\to16) \to \text{ReLU} \to \text{Linear}(16\to1),$$

trained with binary cross-entropy on logits (`BCEWithLogitsLoss`).

B4. Early stopping. Monitor validation loss; keep the best epoch. Report loss per epoch and discuss the curve.

B5. Test performance. On the *test* split:

- Plot the ROC (TPR vs. FPR) and quote AUC.
- Plot *purity vs threshold* t (precision curve) against the class prevalence.
- Choose a working point ($t^\star$) by maximizing purity subject to $\varepsilon_S \geq 30\%$. Report your $t^\star$.

C. Apply the NN to data

C1. Score data. Apply the trained MLP to `jets.csv` to obtain a score per event (leading jet). Do *not* standardize using data; reuse train-set mean/std from MC.

C2. Select. Keep events with $s \geq t^\star$ and plot the leading large-R jet mass *before* and *after* the NN selection on data (density-normalized).

C3. Compare with cuts. On data, overlay three mass spectra: (i) all events, (ii) cut-based, (iii) NN ($t^\star$). On MC *test* split, print the purity for (ii) and (iii) for a direct comparison.

What to submit

Readable, documented code in a merge request to the exercise project. You should clearly describe your results and the methods used. It must at least include:

- Code:
  - Loading of csv file and feature building.

  - MLP training with a convergence criterion (eg. early stopping).
  - Code producing the ROC and purity curves.
  - Application of the selection to both data and MC.
  - Code producing the final mass plots.

- Documentation. This can be delivered either as a pdf report or markdown documentation in your repository.
  - A clear description of features, splits, and NN architecture.
  - ROC (with AUC) and purity-vs-threshold on the MC test split.
  - The chosen working point $t^\star$ and the rationale (purity/efficiency trade-off).
  - Data plots: jet mass before/after cuts and NN; brief interpretation.
  - MC purity numbers for cut-based and NN selections.
  - Figure captions for all figures, explaining content and features.

Document also the environment requirements for running your code, eg. `numpy`, `torch`, `matplotlib`, ... with instructions on how to obtain them and run at least the first part of your code. The file `requirements.txt` can be reused and extended in your merge request.

You are *not* requested to provide mathematical proofs relating to NNs. This is an applied exercise.

Finished early? Stretch goals

If you complete the core tasks ahead of time, pick one or more of the extensions below. Document what you changed and, if relevant, how it affected ROC, AUC, purity, and the data mass spectra.

1. Extend the MC sample (physics, C++)

Add *one additional process* to the MC training sample using `aux/main213.cc` as a template. You will need a local PYTHIA8 + FastJet installation (see `https://www.pythia.org` and `https://www.fastjet.fr` for installation instructions).

- **Suggested backgrounds:**
  - QCD dijets (hard scattering): `HardQCD:all = on` with `PhaseSpace:pTHatMin = 150.` (rich non-V jets).
  - Single top (Wt or t-channel): e.g. `SingleTop:all = on` (leptons from W in top decay can fake the topology).
- **Hints:** The code in `aux/main213.cc` is already set up so you can easily add another subprocess. Reading that code and extending it will be much, much easier than writing your own, since it ensures that you reproduce the relevant jet selection etc..
- **Deliverables:** regenerate `pythia.csv` (or a separate `pythia-extended.csv`), retrain the NN, and compare ROC/purity to the baseline sample.

2. Try a different neural network (ML)

Build a second classifier and compare against your baseline MLP.

- **Change at least one of:**
  - **Architecture:** deeper/wider MLP (e.g. $10 \to 64 \to 32 \to 1$), or remove a hidden layer.
  - **Activation:** replace ReLU with `Tanh`, `LeakyReLU`, or `GELU`.
  - **Loss:** use `BCELoss` with explicit sigmoid, or (for pedagogy) MSE on probabilities; discuss pros/cons vs `BCEWithLogitsLoss`.
  - **Optimizer:** try `SGD` with momentum or `AdamW`.
- **Compare:** ROC (AUC), purity-vs-threshold, and chosen working point $t^\star$. Comment on calibration (do scores look like probabilities?).

3. Add tests and continuous integration (software development, devops)

Propose and implement lightweight tests; run them in CI.

- **Suggested tests:**
  - **Determinism:** fixed seeds $\Rightarrow$ same split and same ROC/AUC on a small frozen subset.
  - **No leakage:** `event_id` sets for train/val/test are disjoint.
  - **Standardizer:** train-set mean ≈ 0, std ≈ 1 after transform; no NaNs/Infs.
  - **Features:** features never uses mass/energy; $\Delta\phi \in [0, \pi]$.
  - **Purity utility:** for a toy label/score vector, computed purity matches a hand calculation.
  - **ROC monotonicity:** FPR/TPR are non-decreasing when sweeping threshold.
- Build a gitlab CI pipline by implementing an appropriate `gitlab-ci.yml`, document your tests and the results.

4. Accuracy vs. compute (sustainability, computing and society)

- **Measure runtime:** record wall–clock time for (i) one training epoch and total to early–stopping, (ii) inference per 10^5 events, and (iii) the cut–based selection. If you can, also estimate energy used.
- **Report accuracy:** on the same test split, give AUC and purity $S/(S+B)$ at your $t^\star$ (and the cut–based purity). Optionally compute simple "benefit per cost": $\Delta\text{AUC}/t_{\text{train}}$, $\Delta\text{purity}/t_{\text{train}}$ (or per Joule if measured).
- **Short note (half a page):** discuss trade–offs of NN vs. cuts (performance, compute/energy, interpretability, maintenance). Say when the NN is "worth it". Propose two footprint mitigations (e.g. smaller model, stronger early stopping, pruning).

Happy Hacking!