Contents

1 Neural network triggers

[to do: write introductory shit here?]

1.1 Motivation

The station-level triggers in the previous chapter have been shown to perform well for the science case of the Pierre Auger Observatory. However, it has also been concluded that a lot of potential data, especially at low energies is ignored. This is by intention in order to keep DAQ readout at feasible levels.

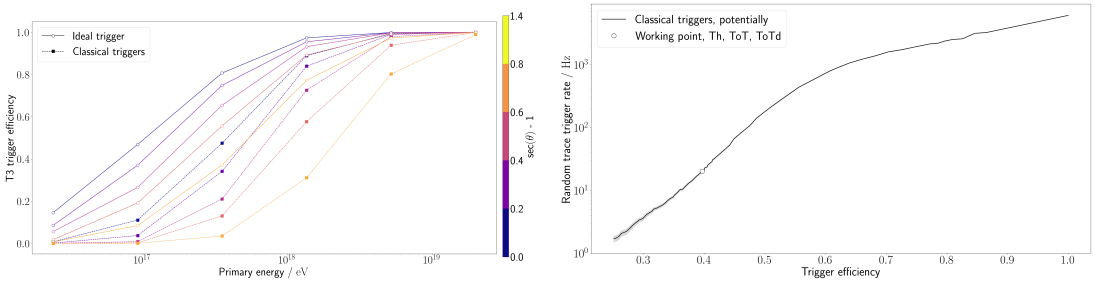
Attempts at improving the overal efficiency of the SD triggers can be made. This is only possible to a certain level. At lowest energies the particle cascade is not big enough to warrant coincident triggers in at least 3 WCD stations. As per **??**, the lateral trigger probability a given classification algorithm can maximally achieve is given by the LPP (c.f. **??**). The T3 detection probability of such an ideal trigger, and consequently the maximal efficiency for an array with 1.5 km spacing is compared to the efficiency of classical triggers in Figure 1.1a.

Of course, efficiency can be improved simply by adjusting trigger thresholds of the algorithms in **??**. However, the more lenient these thresholds are, the more background events will be detected. This quickly results in trigger rates that are unmanagable for the infrastructure at the Pierre Auger observatory. The probability with which time traces correctly raise a T2 is shown alongside the resulting random-trace trigger rate for different thresholds of classical algorithms in Figure 1.1b.

Ideally, neural network architectures developed in this chapter should undercut the random-trace trigger rate of classical triggers, while retaining an overall higher accuracy. That is, they lay below and right of the operating point in Figure 1.1b. For any algorithm that achieves this, the corresponding LTP will be greater than that of classical triggers, resulting in higher event detection efficiency, while not exceeding the bandwidth limitations of the underlaying hardware.

1.2 Implementation

The python library TensorFlow [1] is used as a backend to implement the individual classifiers. All discussed architectures are built and trained with the release version 2.8.4 [2]. Adjustments to the trainable parameters are calculated according to a momentum-based stochastic gradient descent (Adam [3]) on a batch level. In this context, a single



(a) T3 efficiency **(b) Classical trigger potential**

Figure 1.1: (a) Comparison of an ideal trigger sensitive to any shower signal from primary energies $E \geq 10\,\text{PeV}$ to classical triggers. **(b)** The noise level over calculated efficiency for classical triggers. The tail ends of the potential curve are calculated by adjusting the trigger thresholds from +250% to −95% of the nominal values.

batch is made up of all traces that are recorded from a single air shower event. Since batch size grows quickly with increasing energy, a generative approach, where traces are created (c.f. **??**) at runtime upon requirement, is used in building training data in order to make the process as RAM-efficient as possible. This has important implications. As trace building relies heavily on randomization, the actual training data will not be the same if the random number generators are not seeded beforehand. This has been taken into account. All networks are - unless specifically stated otherwise - trained and validated using the same signal input data.

1.3 Design consideration

1.3.1 Architecture

The hardware specifications at the FPGA level, where trigger conditions are currently checked, are limited. For this reason, NN architectures should be kept as simple as possible. Most importantly, the number of weights, biases and other trainable parameters will need to be hardcoded into station software. Because of minimal available storage space, this number needs to be kept low.

This immediately disqualifies powerful candidates like autoencoders or transformers (compare **??**) from consideration, due to their size. Only simple dense-, convolutional-, and recurrent neural networks are viable contenders that might be implemented in the SD electronics.

1.3.2 Choice of input data

As stated in the previous chapters, neural networks learn by example. It is imperative to construct

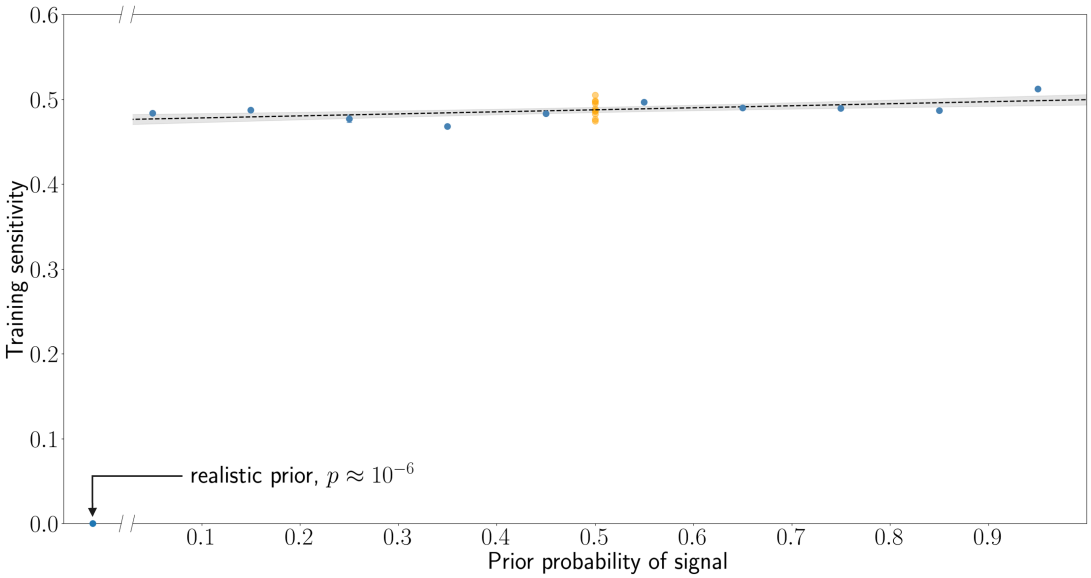


Figure 1.2: A very faint positive slope ($m = 0.02 \pm 0.01$) is observed when relating prior probability to trigger sensitivity (blue dots) of an example convolutional neural network. This could however be attributed to statistical fluctuations in the training fit. An ensemble of networks with the same architecture trained on the same data and with a single prior shows a comparable spread (orange dots).

Prior probability

The flux of cosmic rays with energies exceeding the proton knee is tiny ($O(1\,\mathrm{m}^{-1}\,\mathrm{yr}^{-1})$ [4]). While the size of the SD guarantees decent exposure over the entire array, an individual station will mostly measure background. In fact, the prior probability p of encountering such events in a given random time trace is roughly one in one million. Of course, if this were reflected in the training daatset, neural networks would show very poor performance. On the notion of a broken clock being correct twice a day, a naive classifier can have near perfect accuracy by labelling every input as background. Such behaviour is not desired. The prior probability must be artificially inflated. The influence of prior probability on subsequent trigger sensitivity is shown in Figure 1.2. No strong correlation between prior and network performance is found in the range $0.05 \leq p \leq 0.95$. As long as the fraction of signal over entire training set is statistically relevant, the network learns to discern between signal and background events. For all further analysis a conservative prior of $p = 0.5$ is picked.

Filtering & Downsampling

Loss function weighting

1.3.3 Further hyperparameters

1.4 Performance

1. Have problem
2. Throw math at problem
3. ???
4. Profit

[to do: write this]