A particle track reconstruction algorithm for the Kaggle TrackML challenge

Gerhard Hejc
Deep Water Explorer
Stuttgart, Germany
gerhard.hejc@deepwaterexplorer.com

Abstract

The implemented algorithm starts from the joint probability distribution $p(x_1, x_2, x_3, \ldots x_n|q)$ that the n hits x_1, x_2, x_3, $\ldots x_n$ of an event are part of a charged particle track. This multivariate probability density can be decomposed into n univariate distributions and each term can be approximated by a Gaussian Mixture Density Network, which is fitted individually to the training data.

1 Introduction

The goal of the TrackML data science competition initiated by CERN and hosted by Kaggle is the reconstruction of particle tracks from 3D points (or hits) in a detector. A single event consists of approximately 10000 tracks of particles propagating from the collision center through various detector elements and leaving a trace of hits, where the particle interacts with the environment.

The data consists of 223 GB of uncompressed csv files, which is divided into 8850 training events and 125 test events.

1.1 Data File Format

The event hits file contains the following values for each hit/entry:

- *hit_id*: numerical identifier of the hit inside the event.
- *x, y, z*: measured x, y, z position (in millimeter) of the hit in global coordinates.
- *volume_id*: numerical identifier of the detector group.
- *layer_id*: numerical identifier of the detector layer inside the group.
- *module_id*: numerical identifier of the detector module inside the layer.

The *volume/layer/module id* could in principle be deduced from *x, y, z*.

The event truth file contains the mapping between hits and generating particles and the true particle state at each measured hit. Each entry maps one hit to one particle.

- *hit_id*: numerical identifier of the hit as defined in the hits file.
- *particle_id*: numerical identifier of the generating particle as defined in the particles file. A value of 0 means that the hit did not originate from a reconstructible particle, but e.g. from detector noise.
- *tx, ty, tz*: true intersection point in global coordinates (in millimeters) between the particle trajectory and the sensitive surface.

Preprint. Work in progress.

The event particles files contains the following values for each particle/entry:

- *particle_id*: numerical identifier of the particle inside the event.
- *vx, vy, vz*: initial position or vertex (in millimeters) in global coordinates.
- *px, py, pz*: initial momentum (in GeV/c) along each global axis.
- *q*: particle charge (as multiple of the absolute electron charge).
- *nhits*: number of hits generated by this particle.

All entries contain the generated information or ground truth.

1.2 Mixture Density Networks

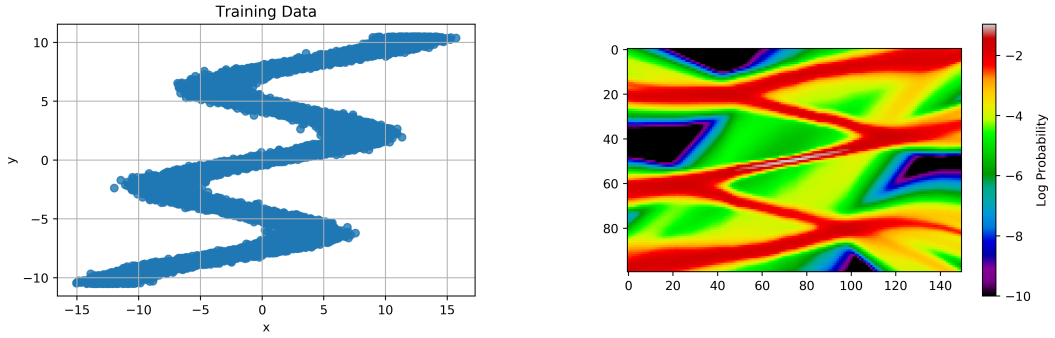
A mixture density network is a fusion of a neural network and a mixture distribution for modelling arbitrary conditional probability densities $p(y|x)$. The mixture distribution is represented by a linear combination of kernel functions with certain parameters and mixing coefficients α, which are calculated using a neural network with a sufficient number of hidden units from the input data x. A nature choice for the kernel function is a Gaussian distribution, which has two parameters μ and σ^2.

$$p\left(y|x\right) = \sum_{i=1}^{N} \alpha_i\left(x\right) \mathcal{N}\left(y|\mu_i\left(x\right), \sigma_i\left(x\right)^2\right) \tag{1}$$

with the constraint

$$\sum_{i=1}^{N} \alpha_i = 1 \tag{2}$$

The probability density can then be used for prediction by selecting the most likely y, but in the case of a multi-modal distribution this would return only a local maxima, as in the two-dimensional toy example shown below. On the rhs side the training data (x_i, y_i) is shown, which is used as input for a 3-layer neural network with 64 hidden units in each layer, and the lhs side shows the result of the approximation of $p\left(y|x\right)$ with 20 Gaussian kernels.



2 Algorithm

Starting point for this algorithm is the joint probability density $p\left(x_1, x_2, \ldots, x_n|q\right)$, which describes the probability that the data points $x_1, x_2, \ldots, x_n$ from the set of all hits $\mathbb{X}$ for a given event are part of a track of a particle with charge q. A input data point x_i has the following components

$$x_i = (x, y, z, volume\_id, layer\_id) \tag{3}$$

The $module\_id$ also present in the hits file is not used here.

An output data point y_i only contains

$$y_i = (x, y, z) \tag{4}$$

because the information about $volume\_id$ and $layer\_id$ can be derived from the x, y and z.

The joint probability distribution can be decomposed into n distributions using Bayes rule

$$p\left(x_1, x_2, \ldots, x_n|q\right) = p\left(x_n|x_1, x_2, \ldots, x_{n-1}, q\right) \ldots p\left(x_2|x_1, q\right) p\left(x_1|q\right) \tag{5}$$

For numerical stability it is favorable to work with the logarithm of the probability densities instead.

$$\log\left(p\left(x_1, x_2, \ldots, x_n|q\right)\right) = \log\left(p\left(x_n|x_1, x_2, \ldots, x_{n-1}, q\right)\right) + \ldots + \log\left(p\left(x_1|q\right)\right) \tag{6}$$

The univariate distributions are approximated by

$$p\left(x_n|x_1, x_2, \ldots, x_{n-1}, q\right) = p\left(y_n|x_1, x_2, \ldots, x_{n-1}, q\right) \tag{7}$$

and a mixture of Gaussian distributions

$$p\left(y_n|x_1, x_2, \ldots, x_{n-1}, q\right) = \sum_{i=1}^{N} \alpha_i\left(x_1, \ldots, q\right) \mathcal{N}\left(y_n|\mu_i\left(x_1, \ldots, q\right), \sigma_i\left(x_1, \ldots, q\right)^2\right) \tag{8}$$

The functions $\alpha_i\left(x_1, \ldots, x_{n-1}, q\right)$, $\mu_i\left(x_1, \ldots, x_{n-1}, q\right)$ and $\sigma_i\left(x_1, \ldots, x_{n-1}, q\right)$ for $n = 1 \ldots N$ are the outputs of a deep neural network. $n + 1$ neural networks are needed for this algorithm.

A prediction of the next hit position $\vec{y}$ can be done using the following expression

$$\vec{y} = \frac{\sum_{i=1}^{N} \alpha_i \frac{\vec{\mu}_i}{\sigma_i^2} \frac{1}{\left(\sqrt{2\pi\sigma_i^2}\right)^d} \exp\left(-\frac{(\vec{y}-\vec{\mu}_i)^2}{2\sigma_i^2}\right)}{\sum_{i=1}^{N} \alpha_i \frac{1}{\sigma_i^2} \frac{1}{\left(\sqrt{2\pi\sigma_i^2}\right)^d} \exp\left(-\frac{(\vec{y}-\vec{\mu}_i)^2}{2\sigma_i^2}\right)} \tag{9}$$

where d is the output dimension (in this case 3). This non-linear equation can be solved iteratively by taking the last track position as a start value and inserting it in the rhs of eq. 9 (it is sufficient for this algorithm to make only one iteration). The predicated value of $\vec{y}$ does typically not coincide with an existing hit value, therefore a search for a close hit near $\vec{y}$ within a distance r is conducted and the log-probability of this data point is checked to decide if the hit is accepted or discarded.

$$x = argmax_{x \in \mathbb{X} \cap |x-y|<r} \log\left(p\left(x|x_1, x_2, \ldots x_{n-1}, q\right)\right) \tag{10}$$

If the hit with the largest log-probability is below a certain threshold or no hit is found in the vicinity of the prediction, the track finding is stopped.

The log-probability threshold and the search distance r are determined in the validation phase, not in the training phase.

The track finding algorithm consists of the following steps for each event:

Algorithm 1 Track finding algorithm

1: Select a q value from the set $(-1, +1)$ and x_1 from all available hits of the event so that $log(p\left(x_1|q\right))$ is a maximum
2: Predict y_2 from the distribution $p\left(x_2|x_1, q\right)$ and select x_2 from all available hits, which are within a certain distance r of y_2
3: If no hits are found or the largest log-probability of the found hits is smaller than a certain threshold, stop the track finding
4: If the hit with largest log-probability is larger than a certain threshold, continue with step 2 for y_3 and distribution $p\left(x_3|x_1, x_2, q\right)$
5: Continue until x_n is reached. If the tracking finding is not stopped at x_n, continue with step 6
6: Use the previous probability density $p\left(x_n|x_1, x_2, \ldots, x_{n-1}, q\right)$ to predict y_{n+1} from $p\left(x_{n+1}|x_2, x_3, \ldots, x_n, q\right)$ and select x_{n+1} from all available hits, which are within a certain distance r of y_{n+1}
7: Continue with step 6 until the stopping criterion is fulfilled.
8: Remove the hits $x_1, x_2, \ldots x_n, x_{n+1}, \ldots x_{n+m}$ (assuming that the track finding stopped at x_{n+m+1}), which are part of the track, from the event and continue with step 1 until all hits are assigned to a track

3 Results

The algorithm was written in Python using Keras 2.0 with Tensorflow as backend. The source code can be found on Github https://github.com/ghejc/trackml.

References

[1] Christopher M. Bishop, Mixture Density Networks, 1994

[2] Axel Brando Guillaumes (2017) Mixture density networks for distribution and uncertainty estimation. Master's Thesis, Facultat de Matemàtiques i Informàtica, Universitat de Barcelona.

[3] David Rousseau, Sabrina Amrouche, Paolo Calafiura, Steven Farrell, Cécile Germain, et al.. The TrackML challenge. NIPS 2018 - 32nd Annual Conference on Neural Information Processing Systems, Dec 2018, Montreal, Canada. pp.1-23, 2018. <hal-01745714>