universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Using Recurrent Neural Networks for Particle Tracking at the CERN Large Hadron Collider“

verfasst von / submitted by

Claus Hofmann, BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2020 / Vienna 2020

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 066 921
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Informatik UG2002
Betreut von / Supervisor:	Univ.-Prof. Dipl.-Inform.Univ. Dr. Claudia Plant

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The CERN Large Hadron Collider (LHC) is the biggest of eight particle accelerators operated by the European Organization for Nuclear Research (CERN). In the Large Hadron Collider, quantum particles are accelerated and subsequently brought to collision. When these particles collide, many new particles are created, which then spread out through space. Around the point where the particles collide, the engineers of CERN have constructed the ATLAS particle detector that records some of the particles' positions along their trajectory. The measurements from the detectors are then used to reconstruct the particles' trajectories through space. This data can then help physicists to uncover previously unknown phenomena of our universe or to confirm theories in particle physics like it has been done with the Higgs-Boson. [10, 32]

The LHC is regularly upgraded. For example, CERN is currently working on the High-Luminosity LHC [9], which is scheduled to be operational by the end of 2027 and which should increase the luminosity (the number of particle collisions per second) of the LHC, making tracking the particles' trajectories more challenging. Therefore, the CERN team is currently also working on further improving the software used for particle tracking. Furthermore, they also want to leverage machine learning experts and enthusiasts from outside CERN to develop ideas on how one could deal with the tracking problem. One part of this effort was that CERN was hosting the

TrackML Kaggle competition [8] that challenged the Kaggle community with the particle tracking problem.

This thesis aims to contribute to this effort by developing a tracker entirely based on machine learning using deep neural networks. More specifically, I will investigate existing approaches to the tracking problem in other areas of research that use neural networks and then adapt one approach to the particle tracking problem.

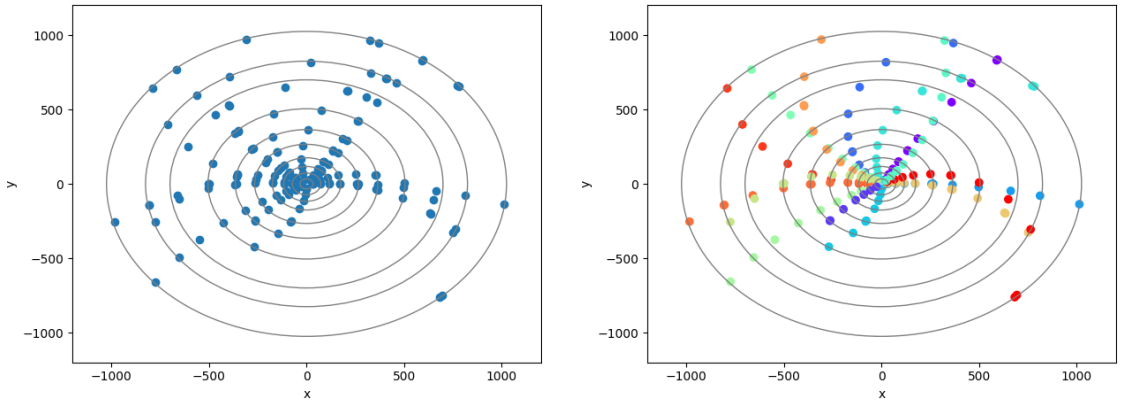
Neural networks have previously been successfully applied to a wide variety of tasks, including face recognition [37], stock price prediction [33], and have even been trained to learn how to play video games [28]. What should be shown with this thesis is that deep neural networks are also capable of learning how to track particles. I do not aim to develop a tracker that can outperform state-of-the-art trackers using physical models, like the one used at CERN [12]. Rather, I would like to show that neural networks can capture physical phenomena.

1.2 The Task of Particle Tracking

When the accelerated matter particles collide, new particles are created. These are then scattered through space, and along their trajectories, they intercept multiple detector layers. Each of the layers records the intercepting particles' position. This means that for each particle, multiple positions that lie on its trajectory (with some inaccuracies because of the measurement process) are measured at certain, predefined locations. The places where the detector records the particles are called hits.

Broadly speaking, there are two types of detector layers: 1) The so-called barrel detector layers, cylinder-shaped detector layers that concentrically wrap around the center, and 2) the negative and positive EC-Rings, which can be described as disks located on both sides of the central barrel detector layers. [32]

The particle tracking task now involves connecting all the hits that originate from the same particle traversing the detector and thereby reconstructing the particles' trajectory through the detector. This is a challenging task because it is not sufficient for a tracker to estimate each particle's trajectory. The tracker also has to distinguish



(a) Without particle assignments (b) With particle assignments

Figure 1-1: 2D projection of the innermost detector with hits from particles traversing it

hits that might originate from different particles on similar trajectories and assign them to the correct track.

Figure 1-1 gives an illustration of the challenges of the particle tracking task. You are probably able to see tracks from some particles by looking at the left figure, but with bare eyes, it is very hard to group all hits that were created by each individual of the 15 particles, especially in places where many hits lie very close to each other. Furthermore, actually, the detector is traversed by approximately 10,000 particles instead of the 15 shown in the figure.

An existing approach that is used to deal with the particle tracking problem will be discussed in section 2.2.

1.3 Detector Structure

The ATLAS detector is constructed around the point in the LHC accelerator, where the particles collide. The detector's structure is described in [32]: It is composed of three sub-detectors: The pixel detector, the short strip detector, and the long strip detector. Each of the sub-detectors consists of three volumes: The barrel volumes, whose layers wrap around the area of impact in a cylindrical fashion, and the negative

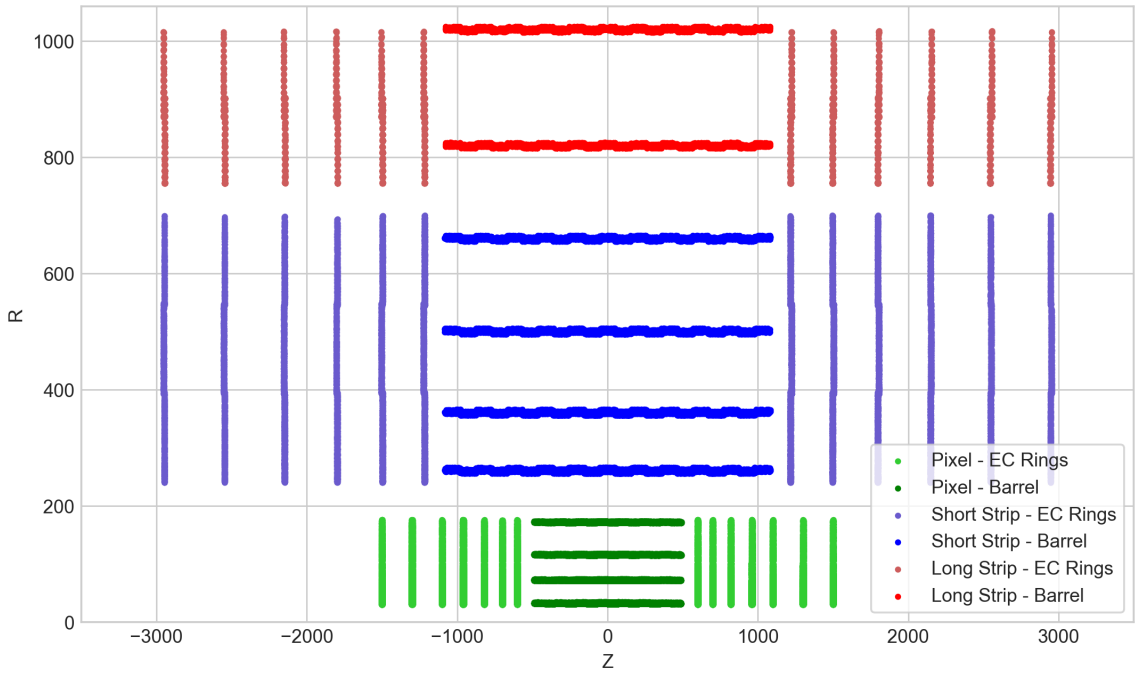


Figure 1-2: (r,z) - Projection of detector structure with central barrel volumes and EC-rings on both sides. Adapted from [32]

and positive end-cap (EC) discs, which are rings located at both ends of the barrel volume cylinders. Note that Figure 1-1 only shows a two-dimensional projection of the barrel volumes; the hits from the end-cap discs are not displayed in this figure.

Figure 1-2 shows another projection of the particle detector, here with the radius (r) and the spatial dimension z as coordinates. This projection displays the cylindrical barrel volumes in the center as horizontal lines and the end-cap discs as vertical lines on both sides.

1.4 Challenges in the Context of Machine Learning Research

The main challenge posed by the TrackML data set to machine learning research is that the particle tracking task cannot easily be posed as a common supervised classification problem because the labels assigned to particles are interchangeable. To illustrate this, let me give an example: Two algorithms, A and B, should track an

event containing two particles. Algorithm A assigns the label X to all hits of particle 1 and the label Y to all hits of particle 2. Algorithm B, however, gives all hits of particle 1 the label Y and the hits of particle 2 the label X. In this example, both algorithms would achieve perfect accuracy, as both of them have grouped the hits correctly. This stands in stark contrast to common classification problems, like distinguishing between images of cats and dogs. An algorithm that classifies cats' images as dogs and dogs' images as cats would score an accuracy of 0, as every image in this example had been misclassified. Furthermore, unlike supervised classification problems, which commonly have a fixed, predetermined number of classes, the number of particles in one event can vary: In the case of the TrackML data set, it can range from 7,000 to 12,000 particles in one event. These two properties of the particle tracking task, namely the interchangeability of labels and the variability in the number of particles, can also be found in unsupervised clustering approaches. Labels assigned to clusters generated by these unsupervised methods can also be interchanged, as for clustering, it is only relevant how the data points are grouped and not how the individual clusters are called. Furthermore, there are clustering algorithms like DBSCAN [13], which do not require to specify the number of clusters beforehand and, therefore, can group the data into a variable number of clusters. While having these two properties in common with unsupervised learning algorithms, the particle tracking task is still a supervised learning problem, as the ground truth is provided in the TrackML data set. However, as already mentioned, the particle tracking task cannot be easily modeled using a common supervised learning-based classifier.

1.5 Notation

In this thesis, care has been taken to use a common notation convention in all parts. Vectors of values are denoted as lower case bold symbols, e.g., $\boldsymbol{v}$. To reference the entry at position i in $\boldsymbol{v}$, the notation $v[i]$ is used. If $\boldsymbol{v}$ is a data vector, i.e., $\boldsymbol{v}$ contains random samples from a probability distribution, a single value in $\boldsymbol{v}$ can also be referenced as the non-bold v, if the specific index of this value is not important in

the current context. Analogous, bold upper case symbols denote matrices, e.g., $\boldsymbol{M}$. A specific entry at row i and column j in the matrix is denoted by $M[i, j]$. If $\boldsymbol{M}$ is a data matrix (i.e., rows represent samples and columns features), the vector $\boldsymbol{m}$ denotes a single sample from this matrix (a single row). To specifically refer to the sample in row i of the matrix, the expression $\boldsymbol{M}[\boldsymbol{i}]$ is used.

Chapter 2

Related Work

2.1 Tracking Multiple Objects

The task of particle tracking is closely related to problems that arise in other areas of research in terms of how the problem is posed and how it can be solved. An example that aims to solve a similar problem as particle tracking is the research area of multiple object tracking (MOT) in computer vision. MOT tackles the problem of how to track multiple objects (for example, persons) in a video sequence. More specifically, MOT aims to identify certain objects in a video, maintain their identities in a frame-by-frame manner, and yield their trajectories. In contrast to single object tracking (SOT), MOT aims at handling video sequences with a large number of objects. This includes the task of modeling the objects' motion and distinguishing different objects in the video. [24]

In MOT, one can divide the way how the objects are tracked in two groups:

- The first group, tracking-by-detection, as discussed in [7, 22, 27, 36] divides the process of tracking multiple objects into two distinct steps: The first step detects all objects that have to be tracked in each frame of the video sequence. It does not connect the detections of the individual frames or recreate trajectories. Rather, it uses a detection model to find all instances of the object in each frame. To give an example, when tracking multiple persons in a video sequence,

the first step should identify those parts of each frame that look like a person. The second step, often referred to as data association, involves tracking the objects by finding all detections that correspond to a single object across the frames. In the literature, this linking step can be done by either only using the motion dynamics of the objects [27] or by also using the objects' appearance as an additional means of discrimination between objects [20]. The tracking-by-detection approach has gained increasing popularity over the last years [22].

- The second group of tracking approaches, which in the literature is also referred to as model-free tracking [42] or category-free tracking [24], rely on an initialization step, which has to be performed manually. Then, they track the manually initialized objects in the video sequence. Examples of this group of approaches include [5,30,42]. The advantage of these approaches over tracking-by-detection is that model-free approaches can track objects with generic appearance, for example, objects with little to no annotated examples available, which would enable one to train a model for object detection. Furthermore, model-free approaches can track multiple different objects in a single video sequence (for example, a person and an animal).

Arguably, the tracking-by-detection approach shares the most similarities with the particle tracking problem. To be more specific, the second step, the data association step, which connects the detections in the individual frames to form tracks, aims to solve a problem that is very similar to the particle tracking problem, as they both create tracks from individual detections. Moreover, the concept of a frame in a video in MOT can be reinterpreted as a layer in a particle detector in particle tracking.

A major difference is that with particle tracking, one does, of course, not have appearance information for the individual detections. Therefore, one has to rely solely on the motion dynamics of the particles. Another challenge that makes particle tracking harder is that there is commonly no strict succession of the detector layers as we have it with the frames in a video sequence. For example, in the structure of the LHC's detector, depending on the particle's trajectory, it can either only hit the

cylindrical barrel layers, but it can also first intercept the barrel layers and then the EC disks.

2.1.1 Other Applications

It turns out that the problem of tracking multiple objects not only is dealt with in computer vision (MOT) or particle physics (particle tracking) but can also be found in biomedicine research [29, 40] and military applications, for example, for tracking objects such as ballistic missiles, aircraft, and military ground vehicles [6, 31]

As a matter of fact, many approaches for solving the problem of tracking multiple objects in these applications use similar algorithms, like the Kalman Filter [14, 18, 23, 31] and min-cost flow [29, 41].

2.2 The Kalman Filter as an Existing Approach to Particle Tracking

Most of the particle trackers used at the ATLAS detector [12] are based on the Kalman Filter. The original version of the Kalman Filter [18] was intended to model dynamic systems in the realm of (tele-)communication and was proposed for solving problems such as the prediction of random signals and the separation of signals from random noise. Subsequently, the Kalman Filter was adapted for solving the problem of tracking multiple targets [31] and also more specifically to the task of particle tracking [14]. In [14], the underlying mathematical model for particle tracking is described as follows:

The state vector $\tilde{\boldsymbol{x}}$ captures the true state of a particle (e.g., the position in space) at each point of its trajectory. The state vector is defined as a function

$$\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{x}}(z)$$

. which describes the entire trajectory of a particle. As the particle intercepts the detector only at certain, discrete locations, it is sufficient not to model the entire

trajectory but rather to only consider the state vector of each particle at these distinct intersection points z_k

$$\tilde{\boldsymbol{x}}(z_k) = \tilde{\boldsymbol{x}}_k = \boldsymbol{f}_{k-1}(\tilde{\boldsymbol{x}}_{k-1}) + \boldsymbol{w}_{k-1}$$

The function $\boldsymbol{f}$ describes the propagation of the particle from layer $k-1$ to layer k, and the random variable $\boldsymbol{w}_{k-1}$ accounts for a possible random disturbance of the particle's track. The states measured by the detector at layer k are described as

$$\boldsymbol{m}_k = \boldsymbol{g}(\tilde{\boldsymbol{x}}_k) + \boldsymbol{\epsilon}_k$$

Note that the measurement at layer k, $\boldsymbol{m}_k$, is influenced by multiple random variables, which in the model are assumed to be independent: The measurement noise $\boldsymbol{\epsilon}_k$, the track disturbance $\boldsymbol{w}_{k-1}$, as well as all prior track disturbances.

Based on this model, the Kalman Filter is performed in three steps:

1. **Filtering** estimates the current state of a particle, based on the past measurements

2. **Prediction** gives an estimate of the state vector in the future time steps

3. **Smoothing** corrects the state vectors in the past by accounting for all measurements up until the current state

Going into detail on how exactly this estimation is performed mathematically would go beyond the scope of this document, but the interested reader will find further details on this topic in [14].

In the particle trackers used at CERN, the Kalman Filter is used in track fitting, the process of estimating the curve describing the particle's trajectory. A Kalman Filter-like formalism is also used for estimating the particles' states in the track finding process, which is the process of finding all measurements originating from a single particle. [12]

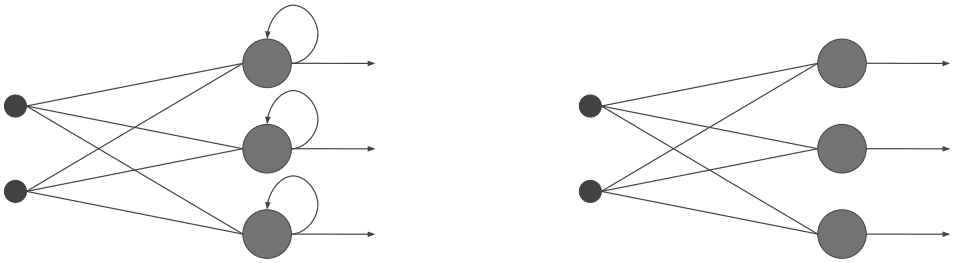
2.2.1 The Combinatorial Kalman Filter

For track finding, which is the process of finding all measurements originating from a single particle, while possible, usage of the classical Kalman Filter is often not sufficient, especially in scenarios with high fluctuation in the particle density, as it is the case in the LHC. To mitigate this issue, [25] proposed a technique called the Combinatorial Kalman Filter (CKF). In the classical approach, a track is formed by applying the Kalman Filter to a seed (an initial guess of a track candidate), subsequently collecting the hits along its predicted path. In this case, a hit is assigned to the closest track prediction. In contrast, the CKF branches a particle track for each hit contained within a certain range $\pm\delta_u^{max}$, giving birth to a new set of track candidates. To limit the number of track candidates in the process, the algorithm discards tracks where the most recently added hit exceeds a limit on this hits' contribution to the filtered track ($\delta_{\chi^2_{max}}$). Moreover, after each evolution step, a quality measure is calculated for each track candidate, and candidates with insufficient quality are discarded. The CKF has been implemented for track finding at the LHC [12].

2.3 Recurrent Neural Networks

Recurrent neural networks (RNNs) have in the past been successfully applied to learning problems dealing with time series [11, 27, 38]. In contrast to feed-forward neural networks, recurrent neural networks contain cycles. More specifically, the input to a recurrent neural network also contains outputs from a previous time step. These inputs that result from a previous time step, which form the so-called hidden state vector ($\boldsymbol{h}$), could also be described as a means of memory that allows the RNN to store and retrieve information from previous inputs. An example of a simple RNN is a neural network with a feedback loop at each neuron (see Figure 2-1).

It seems natural that RNNs could also help with the particle tracking problem, as a particle's position from previous layers evidently heavily influences the particle's position at subsequent layers. The RNN's state could, in this application, be used as a means of transferring information about the particle's movement on previous



(a) Recurrent neural network (b) Feed-forward neural network

Figure 2-1: Comparison: Simple RNN versus feed-forward NN; Adapted from [2]

detector layers to the particle state prediction for the next layer.

2.3.1 Long Short-Term Memory

A common problem with RNNs is that they tend to have a hard time recognizing dependencies between inputs that lie many time steps apart. The reason for this is that with a randomly initialized recurrent neural network, the input of a time step long ago has very little influence on the current output of the RNN. If, however, in reality, this input contains critical information for predicting the current output, the RNN will hardly or only in a very long training process learn this dependency. [16]

To cope with this problem, [17] suggested a neural architecture called long short-term memory (LSTM), which the author claims can bridge time intervals of over 1000 steps. This is done by enforcing constant error flow through each recurrent unit and introducing gates that reduce conflicts in weight update signals. Furthermore, the LSTM uses two state vectors: The hidden state ($\boldsymbol{h}$) and the current state ($\boldsymbol{c}$).

2.4 Tracking Approaches using Neural Networks

2.4.1 Tracking-by-Detection with End-To-End Learning

The authors of [27] propose a tracking-by-detection-based MOT approach that relies entirely on recurrent neural networks. They demonstrate their algorithm on the task of tracking persons in a video sequence. Their tracker can be subdivided into

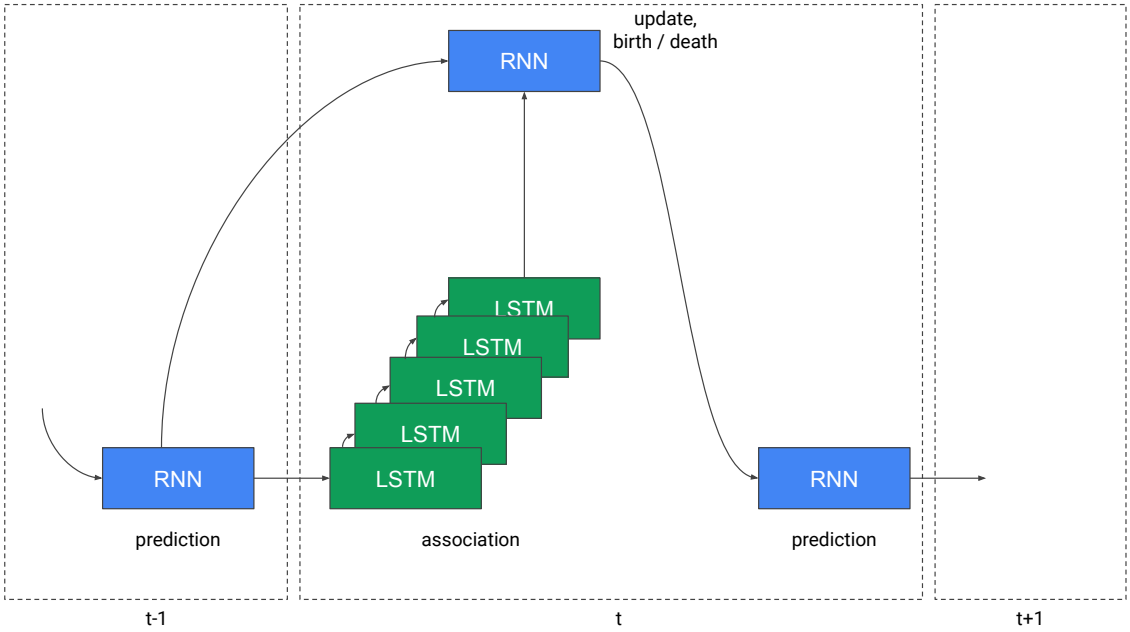


Figure 2-2: Architecture proposed by [27]. Figure adapted from paper

two main parts: 1) A model for predicting the target motion and 2) a model for associating the predicted target states to the respective detections. According to the authors, the approach they propose is the first method for tracking objects that relies on end-to-end learning, i.e., the entire task of tracking is learned entirely from data.

Figure 2-2 gives an overview of the architecture of the tracker. An object's state (i.e., the object's position, not to be confused with the recurrent hidden state used in RNNs) in the next frame is predicted using a custom RNN architecture. The authors then use an LSTM to associate the predicted states to the actual detections from the video sequence. This is done by exploiting the LSTM's temporal step-by-step functionality to predict the assignment for each target, one target at a time. Finally, based on the association information obtained from the LSTM, the authors use another RNN to update the target's state in order to correct the predictions using the detections from the frame.

Motion model

The model to predict and update the target's state (subsequently referred to as the **motion model**) is divided into three main components: The first component is the

prediction step, which deals with predicting the targets' state in the current frame based on the states from the last frames. The second component, the update step, updates the targets' state based on the detections and their estimated association scores to the predicted target states. Finally, the birth/death step accounts for objects entering or disappearing from the scene by assigning an existence score for the current frame to each of the tracked objects.

The prediction step predicts an object's position $\boldsymbol{x}^*_{\boldsymbol{t+1}}$ solely based on the state vector of the last frame $\boldsymbol{x_t}$ and a recurrent hidden state $\boldsymbol{h_t}$, which is fed into the network from the last time step. Thus, the predicted state $\boldsymbol{x}^*_{\boldsymbol{t+1}}$ can be described as a function

$$\boldsymbol{x}^*_{\boldsymbol{t+1}} = pred(\boldsymbol{x_t}, \boldsymbol{h_t}, \boldsymbol{W_p}) \tag{2.1}$$

where $\boldsymbol{W_p}$ represents trainable model parameters.

The update step's inputs comprise of the predicted position $\boldsymbol{x}^*_{\boldsymbol{t+1}}$, the recurrent hidden state from the prediction step $\boldsymbol{h_{t+1}}$ the probability of the object's existence ε_t, the detections $\boldsymbol{M_{t+1}}$ found in frame $t+1$ and the prediction-to-detection association probabilities $\boldsymbol{A_{t+1}}$, latter of which are obtained from the association model. Using these values, the update model estimates the object's position $\boldsymbol{x_{t+1}}$

$$\boldsymbol{x_{t+1}} = upd(\boldsymbol{x}^*_{\boldsymbol{t+1}}, \boldsymbol{h_{t+1}}, \varepsilon_t, \boldsymbol{M_{t+1}}, \boldsymbol{A_{t+1}}, \boldsymbol{W_u}) \tag{2.2}$$

Finally, the birth/death step in the model the authors proposed uses an intermediate result from the update step and outputs the estimated track existence probability ε_{t+1} (the probability of the given object existing in the frame at time t). It is important to note here that the authors propose to keep the number of tracked objects static throughout the tracking process. I.e., the model in each frame predicts the state of n objects, even though there may not be this many objects in the frame. The existence probability should now give the information which of these n objects really exist at the current time step.

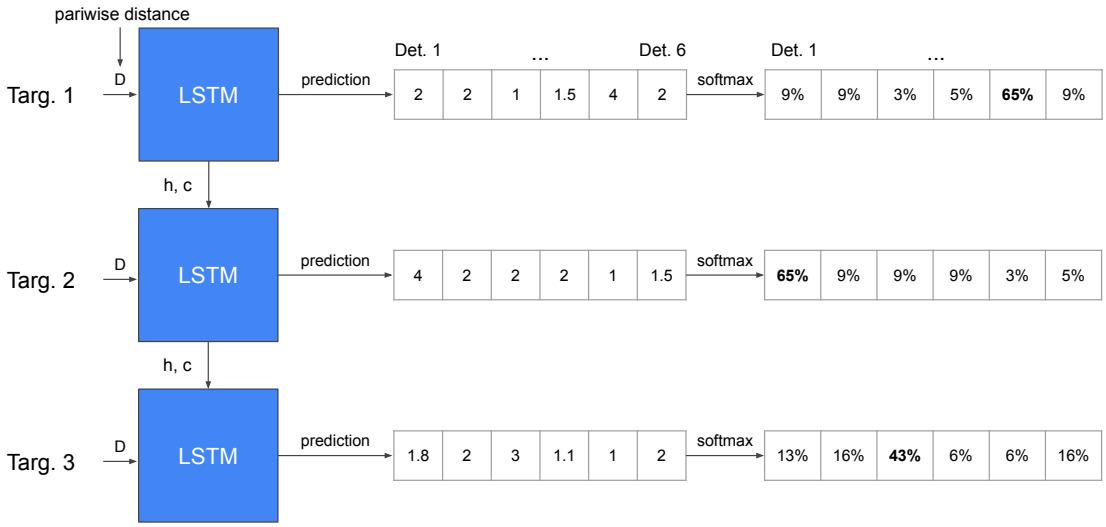


Figure 2-3: LSTM architecture for association model proposed by [27]

Association model

The association model fulfills the task of assigning the individual objects' predicted states $\boldsymbol{x_t}$ to the detections $\boldsymbol{m_t}$ in each frame t. For this, the authors first compute the pairwise distances $D_t[i, j]$ between each object's predicted position i and all detections j. The resulting matrix $\boldsymbol{D_t}$ is of dimension $n \times m$ where n refers to the total number of objects and m to the number of detections. Then, the authors use an LSTM to predict association probabilities $A_t[i, j]$ ($\boldsymbol{A_t}$ is also an $n \times m$ matrix) using the LSTM's time step mechanic to predict association scores for all the objects, one object at a time. It is important to note here that the recurrent time step mechanic in the association model serves a different purpose than in the motion model. While in the motion model, each frame in the input video sequence represents a single time step, in the association model, the time step mechanic is exploited to predict the object's assignments to the detections within a single frame. In other words, in the motion model, a frame requires a single time step, while the association model has to predict a full "time" series for each frame. An overview of how the LSTM architecture proposed by the authors works can be viewed in Figure 2-3. The visualization shows

the process of assigning three objects to six detections. Note that in each of the association model's internal time steps, the full pairwise distance matrix $\boldsymbol{D_t}$ is used.

Losses

The authors employ different losses for each of the two models. For the motion model, the loss is computed as

$$\mathcal{L}(\boldsymbol{x^*}, \boldsymbol{x}, \varepsilon, \tilde{\boldsymbol{x}}, \tilde{\varepsilon}) = \lambda \frac{1}{n} \sum \|\boldsymbol{x^*} - \tilde{\boldsymbol{x}}\|^2 + \kappa \frac{1}{n} \sum \|\boldsymbol{x} - \tilde{\boldsymbol{x}}\|^2 + \nu \mathcal{L}_\varepsilon + \xi \varepsilon^* \tag{2.3}$$

This loss is a sum of in total four components which are weighted using the hyper parameters λ, κ, ν and ξ. The first two components represent losses corresponding to the predicted states $\boldsymbol{x^*}$ and updated states $\boldsymbol{x}$, respectively. More specifically, these two components are calculated using the mean squared error (MSE) between the estimated values and the true states $\tilde{\boldsymbol{x}}$. The loss term $\mathcal{L}_\varepsilon$ represents the binary cross entropy loss between the estimated existence probabilities ε and the true existence labels $\tilde{\varepsilon}$:

$$\mathcal{L}_\varepsilon = -\frac{1}{n} \sum \tilde{\varepsilon} \log(\varepsilon) + (1 - \tilde{\varepsilon}) \log(1 - \varepsilon) \tag{2.4}$$

The last term of the motion model's loss (ε^*) is a regularization term which, according to the authors, should minimize the difference between two consecutive values of ε_t in order to force the model into making less hard decisions, for example in the case where a detection is missing. ε_t^* for an individual time step is computed as

$$\varepsilon_t^* = |\varepsilon_t - \varepsilon_{t-1}| \tag{2.5}$$

For the association model, the authors use the categorical cross entropy loss:

$$\mathcal{L}(\boldsymbol{A}, \tilde{\boldsymbol{A}}) = -\sum_{i=1}^{n} \sum_{j=1}^{m} \tilde{A}[i,j] \log(A[i,j]) \tag{2.6}$$

where $\boldsymbol{A}$ represents the matrix of predicted object-detection association probabilities and $\tilde{\boldsymbol{A}}$ a binary matrix representing the true object-detection assignments.

Evaluation

The authors tested their approach on the MOTChallenge dataset. The approach the authors propose does not quite reach the accuracy of other state-of-the-art trackers, but unlike them, it does not use appearance information but only the geometric locations of the objects. However, according to the authors, the proposed method is two orders of magnitude faster than other state-of-the-art approaches for multiple object tracking.

2.4.2 An Approach from Biomedicine

The authors of [40] propose a method for tracking intra-cellular particles (not to be confused with the particles we want to track in a particle detector) in time-lapse microscopy. To do this, they use a temporal sliding window for solving the association problem from frame to frame. For each frame, first, they extract features from the so far established tracks using an LSTM operating on the last S states of the track. They select a number of possible detections by performing a range query on the track's last state. Finally, they use a feed-forward neural network with dropout to 1) associate the detection candidates to the tracklet by formulating the association problem as a classification problem and 2) predict the tracklet's state in the next frame by formulating a regression problem.

The authors evaluated their approach by comparing its performance to 11 alternative data association methods used in the realm of tracking intra-cellular particles and performed slightly worse than the best state-of-the-art methods.

2.4.3 The HEP. TrkX Project

This is the only scientific source I found that applies neural networks to the particle tracking problem. The authors propose a method that uses LSTMs and convolutional neural networks (CNNs) to find the track of a given particle within all particle tracks. To achieve this goal, they use an image-like representation of the detector layers, where a single pixel is activated when a particle hits the detector layer at the pixel's

position. Then, based on a seed of a particle track (i.e., the particle's hits from the first 2-3 layers) and all "images" of the other layers, CNNs or LSTMs predict the pixels where this particle will hit the detector on the other layers.

According to what they have written in their work, the authors have only tried their algorithm on toy data with a very limited amount of particles and without considering the end-cap discs. [38]

2.4.4 Deep Learning-Based Appearance Model

The work by [39] presents an algorithm for tracking objects in a video sequence using the widespread approach of state predictions using the Kalman Filter [18] and solving the problem of which prediction corresponds to which detection using the Hungarian algorithm [21]. The authors' work mainly focuses on dealing with occlusions. These occlusions occur in visual tracking when a tracked object is not visible in the video sequence, for example, because the tracked object has moved behind another object. Once the tracked object reappears in a video frame, the tracker should still correctly associate the object. For this, the authors employ a convolutional neural network (CNN) to compute an embedding using the objects' appearance. Then, they perform a nearest neighbor query on the embeddings to match the detection to the correct tracked object.

Chapter 3

Tracker Design

3.1 Selection of Tracking Approach to be Adapted for Particle Tracking

To select which of the approaches discussed in section 2.4 to adapt and implement for the particle tracking problem, the proposed algorithms are compared according to adaptability, tracking performance outlook, and run time performance outlook. Unfortunately, the approaches discussed use different evaluation data sets and metrics (except for [27] and [39], which both use the MOTChallenge data set and similar metrics), which makes a comparison according to pure performance measures infeasible. Thus, the comparison has to give an estimate on how the approaches may perform when adapted the particle tracking task.

3.1.1 Adaptability

Milan et al.

The authors of [27] have stated in their publication that they have designed their tracker to be easily adaptable to other applications by replacing the features of the state vector with the features that are relevant to the specific application. A special problem that will arise when adapting this approach to particle tracking is how to

track the particles throughout the different detector volumes, as in contrast to frames in a video sequence, the layers in the detector cannot be ordered in strict succession, as any given particle might or it might not hit a certain detector layer.

Yao et al.

In [40], it is not explicitly mentioned how one could adapt their approach from the realm of molecular biology to other problems, but similarly to the last approach, replacing the state vector features with relevant ones should work. Here also, one has to deal with the problem of the different detector volumes.

HEP. TrkX Project

The approach described in [38] deals with particle tracking, but only on a toy data set and without the problem of orthogonal detector volumes. Thus, some adaption is likely needed, but it might be easier than the two approaches discussed before, as the tracker has been designed specifically for particle tracking.

Wojke et al.

In [39], an approach for visual MOT is discussed, which applies neural networks for appearance modeling. As there is no appearance for the individual objects in particle tracking, this approach cannot be adapted. Thus, it will not be further discussed in this comparison.

3.1.2 Tracking Performance Outlook

Milan et al.

While [27] does hardly reach the state-of-the-art in MOT trackers in terms of accuracy, one must also mention that most of the trackers the authors compared their approach to not only use motion dynamics but additionally appearance information. The authors' work does not use appearance information. As appearance information

is also not available for the particle tracking problem, the outlook on this approach's performance can still be described as promising.

Yao et al.

The results from [40] are comparable to the state-of-the-art in the authors' research domain. As for their application, appearance features cannot be used, one could argue that state-of-the-art trackers from this domain will likely perform worse than the trackers from the research area of MOT because MOT has access to more features. Therefore, one cannot really say if the tracking performance will be better or worse than for the approach by Milan et al.

HEP. TrkX Project

The authors of [38] have tested their approach on a toy data set of up to 22 tracks. The full data set contains around 10,000 tracks. While the median accuracy (the fraction of hits assigned to the correct particle) up until 21 tracks is above 0.8, the median accuracy dramatically drops to 0.5 at 22 tracks. Thus, it can be argued that scaling this tracker to the full data set w.r.t. the accuracy will be quite hard. It could be possible with additional adaptations, though.

3.1.3 Run Time Performance Outlook

Milan et al.

In [27], it is reported that their tracker performed at a frame rate of 165 frames per second. In contrast, the best tracker the authors compared their approach to achieved a frame rate of 32.6 frames per second. The other MOT approaches that were compared were two orders of magnitude slower than the presented approach. This gives a very promising outlook for this approach with respect to run time.

Paper	Adaptability	Tracking	Run time
Milan et al.	●	●	●
Yao et al.	●	●	●
HEP. TrkX	●	●	●
Wojke et al.	●		

●	Very promising
●	Promising
●	Decent
●	Not considerable

Table 3.1: Comparison summary of proposed trackers according to estimated outlooks on adaptability, tracking and run time

Yao et al.

The authors of [40] unfortunately did not compare their approach to other trackers from their research domain. Thus, the proposed algorithm has to be examined closer to assess the run time performance outlook. While the deep learning model seems quite promising in terms of run time, they use exact methods for solving the linear assignment problem when assigning the predicted states to the detections. Using the Hungarian algorithm [21] for solving this task takes $O(n^3)$, which could negatively impact the total run time.

HEP. TrkX Project

There is no run time comparison provided in [38]. As already mentioned in section 3.1.2, the authors tested their approach on up to 22 tracks, with the full data set containing 10,000 tracks. This might indicate that additional work has to be done to scale this approach to the full data set.

3.1.4 Decision

Table 3.1 summarizes the findings from the sections 3.1.1 - 3.1.3. The algorithm by Milan et al. [27] has the best outlook when it comes to run-time performance and is also promising regarding adaptability and tracking performance. As run time might be a very important criterion for scaling a tracker to the full data set of about 10,000 particles, the author of this thesis decided to adapt and implement this approach for

particle tracking. The outlook for Yao et al. [40] is promising concerning all criteria. Thus it might also be interesting to adapt this approach for the LHC data set. As the run time outlook seems slightly less promising than the one of Milan et al., Milan et al. is selected over Yao et al. While the HEP. TrkX Project [38] is estimated to have the best outlook with respect to adaptability, adapting this approach could require additional research regarding scaling concerning tracking performance and run time. Finally, the approach by Wojke et al. cannot be considered for adaption, as the approach is focused on appearance modeling.

3.2 Tracker Adaptations

3.2.1 Tracked Features

The tracking approach by Milan et al. [27] uses a state vector with four components: The object's position in the frame (x and y) and the size of the bounding box around the object (w and h). This vector is a dynamic representation of the objects' states within the different frames. The state vector's components have to be adapted for particle tracking, as evidently there is no such thing as a bounding box in this application. The main components used in this thesis are the particle's three spatial coordinates x, y, and z. Furthermore, pre-computed non-linear projections of these components are added to the state vector, specifically the azimuthal angle ϕ and the polar angle θ. These were added because unlike the spacial components x, y, and z, which increase in absolute magnitude with increasing detector layer count, the angles ϕ and θ of a given particle can be expected to change to a far lower extent with increasing layer count and are therefore arguably better suited as proxies for the positions in a video frame. The azimuthal angle ϕ is computed as

$$\phi = atan2(y, x)$$

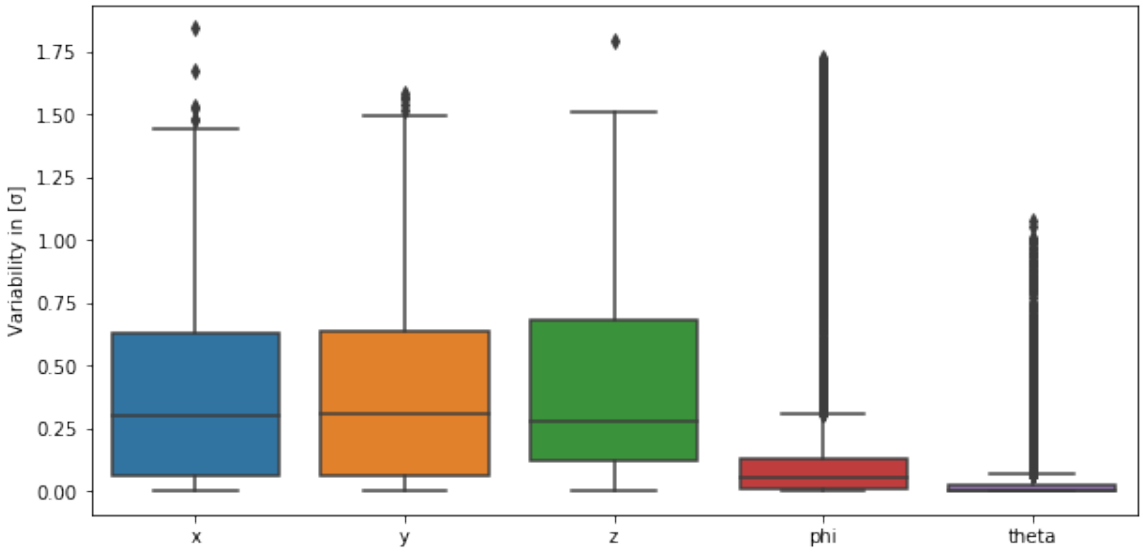


Figure 3-1: Variability of features for individual particles in one event

and the polar angle θ is given by

$$\theta = atan2(\sqrt{x^2 + y^2}, z)$$

Figure 3-1 displays a box plot of how the state vector's features for individual particles vary across different detector layers. For this, the event data was standardized using the z-transformation, and then for each particle individually, the standard deviations of the features were computed. Thus, the unit of the vertical axis is σ, the standard deviation of the entire data set. As one can see, the variabilities of the angles ϕ and θ are far smaller than the ones of the spatial dimensions, although there are some outliers. For the angle ϕ, these outliers could be explained by some particles' tracks wrapping around from 2π to 0 (or vice versa).

For tracking, all state vector components are scaled to be in the range [-0.5, 0.5]. For further information on the scaling of the three spacial components x, y, and z when tracking particles throughout multiple detector volumes, please see section 3.2.5.

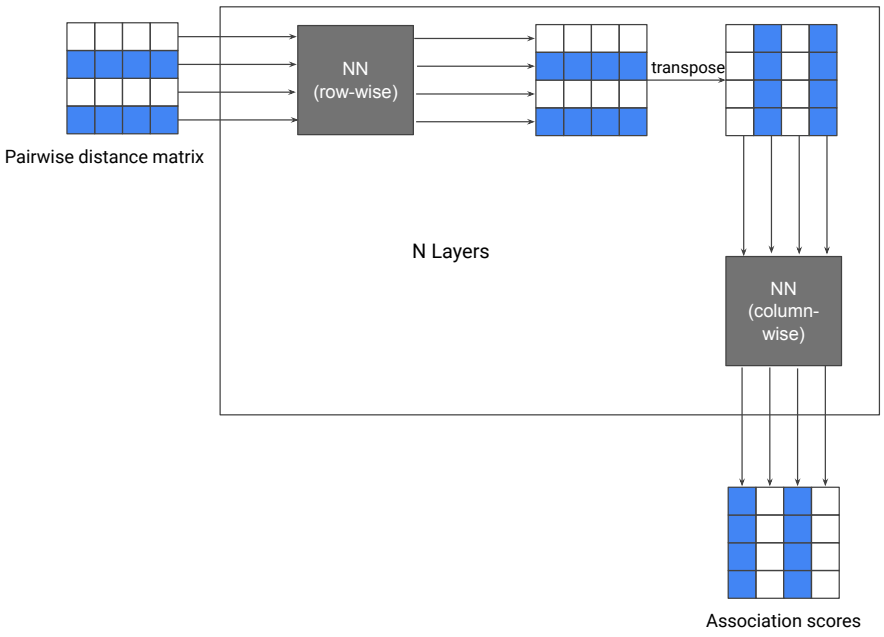


Figure 3-2: Association model using feedforward neural networks

3.2.2 Association Model

Architecture

The model proposed by [27] for assigning the predictions from the motion model to the detections uses an LSTM, which exploits the LSTM's temporal steps to predict association scores of each particle, one particle at a time. For this, the authors state they use the full pairwise distance matrix $\boldsymbol{D_t}$ between predictions and detections as input for each time step. I could not reproduce the author's results with this architecture when using the full pairwise distance matrix, but I achieved good results when I only fed the row $\boldsymbol{D_t[i]}$ instead of the entire matrix to the LSTM when predicting the association scores for particle i.

Furthermore, a novel neural network architecture for the association task has been developed for this thesis, which only uses common feedforward NNs to solve the task. Figure 3-2 illustrates the architecture. The pairwise distance matrix is fed into a dense layer with m units where m refers to the number of measurements. In this step, the dimension of the particles is used as a batch dimension. The matrix is

then transposed and fed into another dense layer with n (the number of particles) units. This process can subsequently be repeated on multiple layers, but experiments showed that a single layer is sufficient.

Evaluation

To compare the performance of the individual architectures for data association, similarly to [27], they have been tested on the problem of solving the linear assignment problem:

$$\arg\min_{\boldsymbol{W}} \sum_i \sum_j W[i,j] \cdot C[i,j] \tag{3.1}$$

such that

$$\sum_j W[i,j] = 1 \text{ for all i} \tag{3.2}$$

$$\sum_i W[i,j] = 1 \text{ for all j} \tag{3.3}$$

$$\boldsymbol{W} \in \{0,1\}^{N \times M} \tag{3.4}$$

Given a cost matrix $\boldsymbol{C}$ the task is to find an assignment $\boldsymbol{W}$ which minimizes the total cost and which ensures that each row and each column is assigned once and only once. Note that none of the architectures discussed earlier can by design guarantee that both requirements (3.2) and (3.3) are fulfilled simultaneously. The idea is rather to give an approximation to the correct solution. To compare the approaches, cost matrices $\boldsymbol{C_i}$ have been randomly generated and the respective linear assignment problems have been solved to optimality using the `scipy` package. Then, the models have been trained on the optimal solutions. Each model was trained on 800,000 random problem instances. Finally, the models are evaluated according to their accuracy (the number of correct assignments) and the cross entropy loss. Furthermore, the run time of the individual model architectures was measured on an Intel Core i5-7267U dual-core processor with 3.1GHz. As depicted in figure 3-3, the row-wise LSTM model performs best with respect to accuracy. The dense model gives slightly worse but still

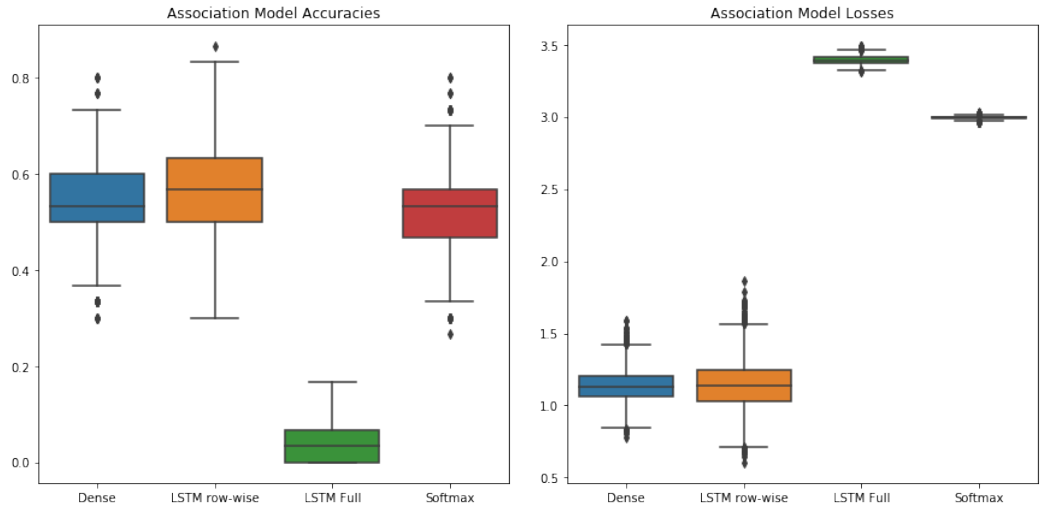


Figure 3-3: Accuracy and loss of three NN architectures for data association compared with a simple approach just assigning each row to the column with the lowest cost

competitive results. The LSTM model using the full pairwise distance matrix at each time step in my experiments gives hardly a better result than random assignment. As a reference, the architectures are compared to a very simple approach where only the softmax function is applied to the entries in the cost matrix $\boldsymbol{C}$, resulting in each row being assigned to the column with the lowest cost. The accuracy of this simple approach is comparable to the one of the row-wise LSTM and the dense architecture, which indicates that when just considering hard assignments, the NNs are hardly better than the naive approach. However, when considering the loss, and therefore soft assignments, it turns out that the two NNs in fact perform better than just applying the softmax function to the cost matrix.

The graphs depicted in Figure 3-4 show that both LSTM architectures perform significantly worse than the dense model with respect to run time. The former show a very large increase in run time with increasing problem size - average training time taking about 6-8 seconds for a 50×50 cost matrix - while the latter does not show an increase in run time for the problem sizes tested, which could be caused by the overhead of the deep learning framework used for implementing the models being larger than the actual run time of the model. Data association on the dense model for a 50×50 matrix takes on average only about 0.07 seconds - a speedup of 85

compared to the LSTM-based architectures.

One-to-many assignment

Unlike the task of tracking objects in a video, where each object in each frame is assigned to at most one detection, a single particle can produce multiple hits on a single detector layer. This is caused by the structure of the particle detector: The smallest units of detection, the so-called modules, overlap in certain regions, which can cause a particle to intercept two or more modules on a single layer. As a result of this, the detector records multiple hits for the particle on this layer. [32] The model proposed by [27] enforces that one object can be assigned to only one detection by applying the softmax function on the axis of the objects (particles in the case of particle tracking). To make the fact that one particle can result in multiple hits reflected in the model for particle tracking, the softmax function is instead applied on the axis of measurements (detections in the case of MOT). This ensures the possibility of a single particle being assigned to multiple measurements and enforces that each measurement can be assigned to one particle and one particle only.

3.2.3 Loss

While [27] state they used separate losses for training the association model and the motion model and at first trained both models independently, it is not quite clear from their paper which loss they used when training the models together, but presumably they also used the motion model loss

$$\mathcal{L}(\boldsymbol{x}^*, \boldsymbol{x}, \varepsilon, \tilde{\boldsymbol{x}}, \tilde{\varepsilon}) = \lambda \frac{1}{n} \sum \|\boldsymbol{x}^* - \tilde{\boldsymbol{x}}\|^2 + \kappa \frac{1}{n} \sum \|\boldsymbol{x} - \tilde{\boldsymbol{x}}\|^2 + \nu \mathcal{L}_\varepsilon + \xi \varepsilon^* \tag{3.5}$$

for training the combined model, because the MOTChallenge [26] they used for evaluating their results only relies on the position and size of the bounding boxes as input. The bounding boxes can be optimized using only the loss formulation (3.5), because using this loss minimizes the errors of the state vector estimators $\boldsymbol{x}$ and $\boldsymbol{x}^*$, which contain the bounding box information.

However, for the particle tracking challenge on Kaggle [8] the benchmark that is used for evaluation is purely based on the particle-to-measurement assignments. The particle's estimated positions are not evaluated. Thus, arguably the loss (3.5) is not sufficient, as it does not contain the matrix of assignments $\boldsymbol{A}$. Therefore, the following loss is used, which combines the loss from the motion and association models:

$$\begin{aligned}\mathcal{L}(\boldsymbol{x}^*, \boldsymbol{x}, \varepsilon, \boldsymbol{A}, \tilde{\boldsymbol{x}}, \tilde{\boldsymbol{A}}, \tilde{\varepsilon}) =& \lambda \frac{1}{n} \sum \|\boldsymbol{x}^* - \tilde{\boldsymbol{x}}\|^2 + \kappa \frac{1}{n} \sum \|\boldsymbol{x} - \tilde{\boldsymbol{x}}\|^2 + \\ & \nu \mathcal{L}_\varepsilon + \xi \varepsilon^* - \psi \sum_{i=1}^{n} \sum_{j=1}^{m} \tilde{A}[i,j] \log(A[i,j])\end{aligned} \tag{3.6}$$

3.2.4 Motion Model

While the original tracker from [27] uses a regular RNN architecture for modeling target motion, the particle tracker presented in this thesis uses an LSTM, as LSTM is known to perform better at learning long-term relationships between time steps and does not suffer from the problem of vanishing gradient to the same extent as regular RNNs do [17].

Figure 3-5 shows a detailed graph of the architecture of the motion model used by the particle tracker. There are two LSTM cells with a distinct set of trainable weights: The cell in the prediction step deals with predicting particles' positions at detector layer $t + 1$ based on the particle state vector and the recurrent hidden and current states from detector layer t. In the update step, another LSTM cell uses the hidden and current states from the prediction step at detector layer $t + 1$ and a weighted mean of the state vector prediction and the measurements at detector layer $t + 1$.

This weighted mean expression requires some additional explanation. A similar process is used in the model from [27], though it is unfortunately not discussed in the respective paper. However, the author explained it in a post on their code repository [4]. The formulation there relies on a one-to-one assignment between objects and detections for each frame. Because in the particle tracking case, a particle can trigger multiple measurements on a single detector layer, the weighted mean for par-

ticle tracking has to be adapted: Namely, as a first step, the assignment scores are standardized such that the scores for each particle sum up to 1:

$$A_{t+1}^{(R)}[i,j] = \frac{A_{t+1}[i,j]}{\sum_{k=1}^{m} A_{t+1}[i,k]} \tag{3.7}$$

As a next step, the individual measurements are weighted with their assignment score according to $\boldsymbol{A_t^{(R)}}$

$$\boldsymbol{X_{t+1}^{(W)}[i]} = \sum_{j=1}^{m} A_{t+1}^{(R)}[i,j] \cdot \boldsymbol{Z_{t+1}[j]} \tag{3.8}$$

note that $\boldsymbol{X_{t+1}^{(W)}[i]}$ is a weighted mean of the measurements using the association scores from particle i, $\boldsymbol{X_{t+1}^{(W)}[i]}$ can thus be interpreted as a (likely biased) estimator of $\boldsymbol{\tilde{X}_{t+1}[i]}$. The re-weighting step denoted in (3.7) is necessary as in the data set, multiple measurements can be assigned to a single particle. Weighting the measurements according to $\boldsymbol{A_{t+1}}$, would introduce an additional bias to $\boldsymbol{X_{t+1}^{(W)}}$ where the entries of $\boldsymbol{X_{t+1}^{(W)}[i]}$ would roughly be proportional to the number of measurements assigned to particle i. This is mitigated by the standardization in (3.7), as $\boldsymbol{X_{t+1}^{(W)}}$ should not be influenced by the number of measurements assigned to the individual particles.

As a next step, $\boldsymbol{X_{t+1}^{(W)}}$ is weighted by the existence scores $\boldsymbol{\varepsilon_{t+1}}$. To explain the reasoning behind this, remember that in the model, throughout the entire time series process, the states for n particles are predicted, even if a lower number of particles actually intercept the current detector layer. As the association scores in $\boldsymbol{A_{t+1}^{(R)}}$ only really contain sensible information if the associated particles actually exist, it could make sense to weigh $\boldsymbol{X_{t+1}^{(W)}}$ according to the existence probability.

$$\boldsymbol{X_{t+1}^{(\varepsilon)}[i]} = \varepsilon_t[i] \cdot \boldsymbol{X_{t+1}^{(W)}[i]} + (1 - \varepsilon_t[i]) \cdot \boldsymbol{X_{t+1}[i]} \tag{3.9}$$

As the existence probabilities $\boldsymbol{\varepsilon_{t+1}}$ of the detector layer $t+1$ are at this moment not yet available, the weighting has to be done using $\boldsymbol{\varepsilon_t}$, the existence scores from detector layer t.

3.2.5 Tracking Particles throughout Different Volumes

An additional requirement that sets a particle tracker apart from an MOT-based tracker is the ability to track particles throughout different volumes. To explain why this is a problem, let's reexamine the detector structure (see also section 1.3. for more information). The barrel volumes in the center are oriented orthogonally to the EC discs on both sides. When not considering the EC discs, the barrel layers form a sequence where one layer is followed by exactly one other layer, much like the frames in a video. However, when adding the EC discs, it is impossible to build such a frame-like succession while also maintaining the particles' natural order intercepting the individual layers.

To solve this problem, each volume uses a separate set of both motion and association model, with an additional recurrent neural network (in the following called the **transition layer**) managing the transition between the individual volumes. Figure 3-6 displays the architecture of the transition layer at the example of transitioning from the pixel barrel volume to the short strip barrel volume. It uses the current and hidden states and the updated states and the existence scores from the last layer t of the pixel barrel volume to estimate the predicted state and the existence score on the first layer 0 of the short strip barrel volume.

Figure 3-7 depicts how the individual volumes have been connected using the transition layer. From each of the three barrel volumes, the particles are transitioned to the respective positive and negative EC volumes and to the succeeding barrel volume. For each of the volumes, states for the full set of n particles are computed. The fact that certain particles never intercept some detector volumes is accounted for by the existence scores $\boldsymbol{\varepsilon}$. This is why the transition layer estimates not only the state prediction for the first layer of the volume transitioned to but also an initial existence score.

When tracking particles on multiple volumes, the spatial coordinates x, y, and z are normalized to be in the range [-0.5, 0.5] for each volume individually. In other words, if a hit's normalized x coordinate in a volume is 0.5, this means that this hit

is at the highest possible x coordinate of this volume, not of the entire detector.

3.3 Scaling Ideas

The data set from the TrackML Kaggle competition [8] contains about 7,000 - 12,000 particles per event, with a maximum of 5,000 - 7,000 particles present on a single layer. Scaling to such a large number of objects to track simultaneously poses a special challenge to the tracker, as for each layer, the tracker has to compute the full pairwise distance matrix between the predicted particle positions and the measurements on this layer. Because the expected number of hits per particle is constant, this operation scales quadratically with the number of particles. Thus, the tracker's run time is bounded from below by $\Omega(p^2 \cdot l)$ where p is the number of particles to track simultaneously and l the total number of layers in the detector. Furthermore, the memory requirements to store the full pairwise distance matrix also scales quadratically with the number of particles and therefore the tracker needs at least $\Omega(p^2)$ units of memory (the number of layers can be omitted here as the memory required to store the matrix could be reused for each layer).

To reduce the problem of quadratically scaling run time and memory requirements, the detector can be divided into buckets where each bucket holds an on average constant amount of particles. Then, the particles in each bucket can be tracked independently of each other. The tracker's run time on a single bucket is then $\Theta(l)$ and memory requirements are $\Theta(1)$. As the number of particles per bucket should be constant, the number of buckets has to scale linearly in p, resulting in the total number of buckets being of order $\Theta(p)$. Thus, the run time of this bucketed tracking approach is $\Theta(p \cdot l)$. It is important to mention here that this improved run time complexity is only possible at the expense of tracking accuracy, as the bucketed tracking cannot account for particles that move between buckets. Thus, it is essential to find a bucketing strategy that minimizes the number of particles moving between buckets - or in other words, maximizes the average track length (i.e. the number of detector layers intercepted by a given particle) within each bucket.

3.3.1 θ Buckets

The first approach tested was to sort the hits into buckets by using the θ angle, because as discussed in section 3.2.1, θ is the feature that varies the least for individual particles. Given the number of desired buckets b, the b-quantiles with regards to the feature θ are extracted from the event data. Then, the hits are distributed to the buckets by assigning a hit that lies between the b-quantiles q_i and q_{i+1} to the bucket with the ID i (with the smallest ID being 0).

3.3.2 Approximate-Nearest-Neighbour Buckets

A second approach that was tested was splitting the hits into buckets using approximate-nearest-neighbors with the library Annoy [34]. Approximate-nearest-neighbours has in the past been successfully applied for finding buckets containing long particle tracks [3]. The process for assigning the hits to the buckets is as follows: As a first step, an index of the event data is created using Annoy. Then, a hit is selected uniformly at random from the event data, and its k approximate-nearest-neighbors are queried using the index. As a similarity measure, the cosine distance is used. All found hits are assigned to a common bucket. Then, a random hit not yet assigned to any bucket is randomly selected, and the process is continued until all hits are assigned to a bucket. Note that using this process, the buckets are not mutually exclusive: A single hit can be assigned to multiple buckets because it can be contained in the k-approximate-nearest-neighborhoods of multiple hits. To reduce complexity, mutual exclusivity is enforced on the buckets by always assigning a hit to the last bucket it was discovered to be part of.

3.3.3 θ/ϕ Buckets

To improve the bucketing strategy covered in section 3.3.1, the angle ϕ is added as a second criterion for the bucketing. The angle ϕ is suited for this, as it is the feature that varies the second least for a single particle, as discussed in section 3.2.1. For this bucketing approach, the hits are first sorted into b_θ buckets, and these buckets are

then subdivided into b_ϕ buckets each. The total number of buckets b is thus $b_\phi \cdot b_\theta$. Compared to the approach from 3.3.1, this allows the number of θ-buckets b_θ to be reduced, which is remedied by additional ϕ-buckets. For example, if we set $b = 56$, in the approach from 3.3.1 the the θ dimension is divided into 56 buckets. If we add $b_\phi = 4$ buckets in the ϕ dimension, it suffices that $b_\theta = 14$ for the same number of total buckets $b = b_\phi \cdot b_\theta = 56$.

3.3.4 Comparison

To compare the bucketing strategies, the particle track lengths within the individual buckets were examined. Figure 3-8 shows the respective distributions. As one can see, all bucketing strategies cause a large increase in the number of particle tracks with only one hit. One of the reasons for this might be that a particle traversing multiple buckets is counted multiple times in the distribution. For example, if a single particle traverses four buckets and creates one hit in each bucket, this particle is counted as four tracks with a single hit. The visualization indicates that the θ / ϕ bucketing strategy works best, as the number of one-hit-tracks is the lowest, and it also contains the most tracks with lengths from 11 to 13 among the three bucketing strategies. Approximate-nearest-neighbour and θ-buckets perform worse, with the ANN-approach seeming slightly superior to the θ-approach.

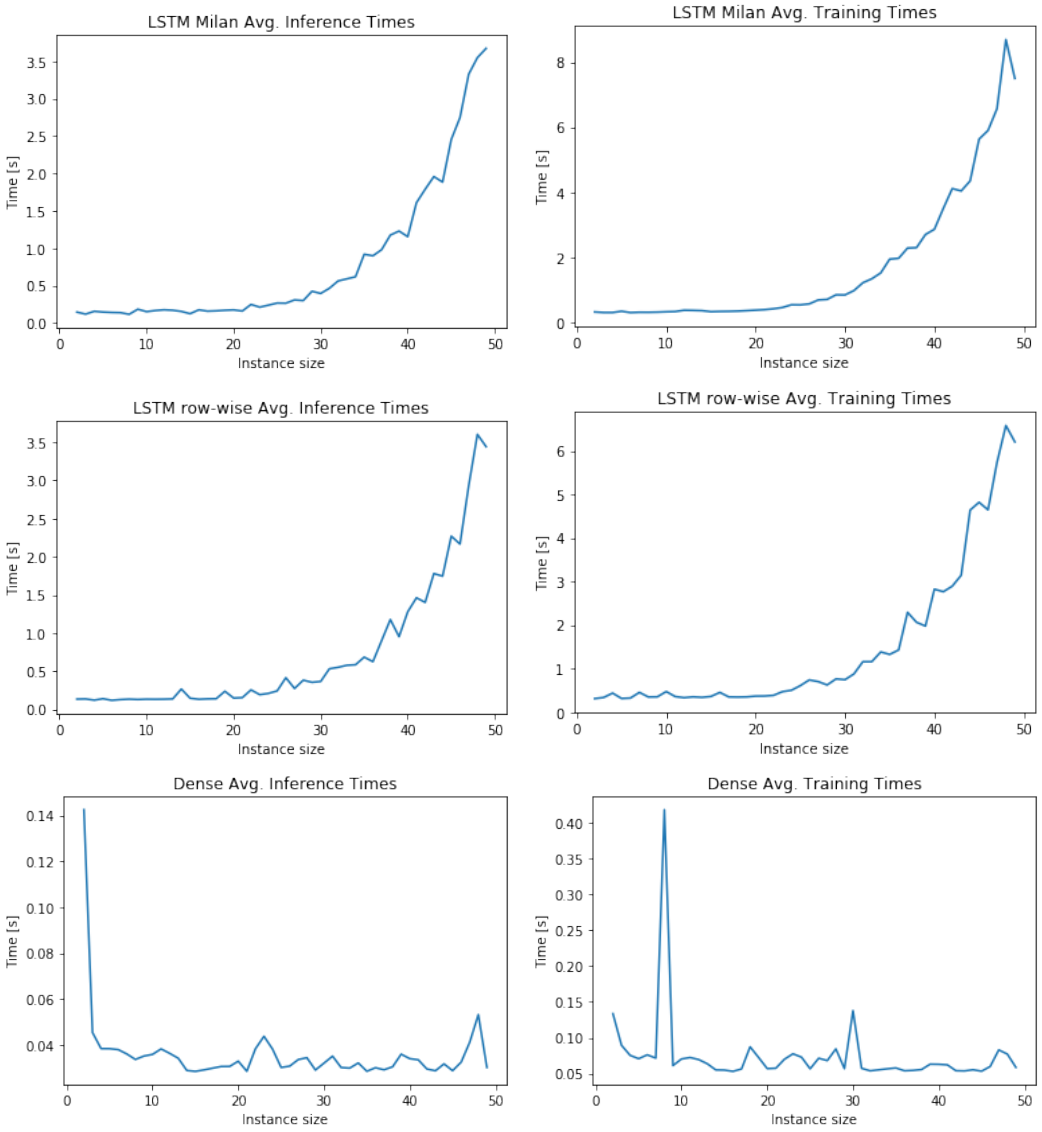


Figure 3-4: Run time comparison of architectures for data association. Both inference and training times are depicted. Average run times over 10 instances

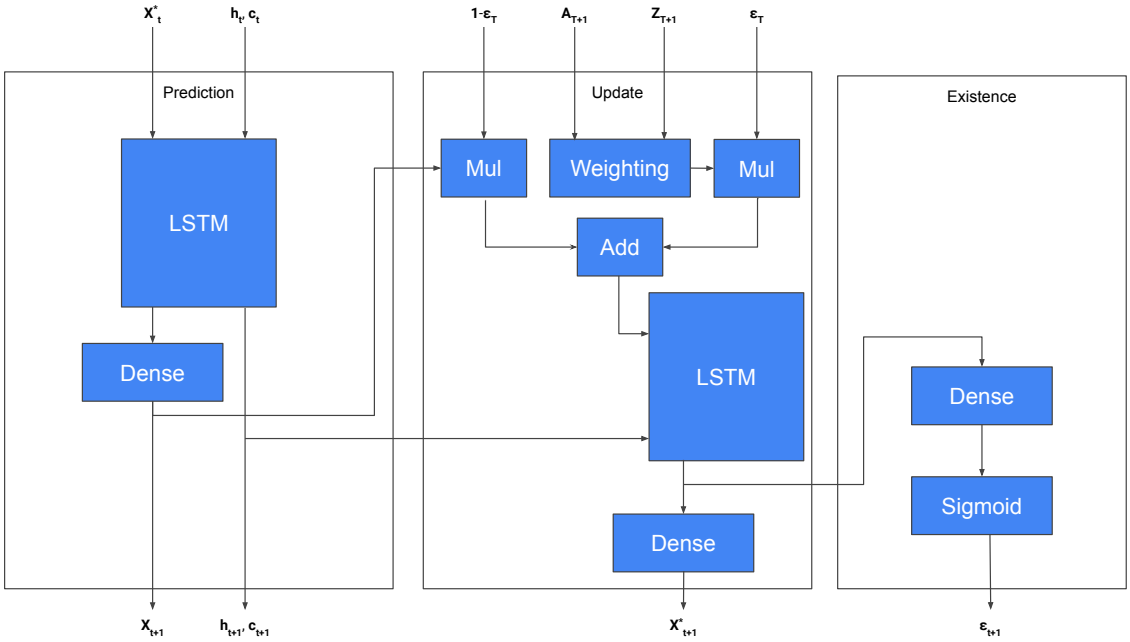


Figure 3-5: Motion model of particle tracker. Model and graphic adapted from [27]

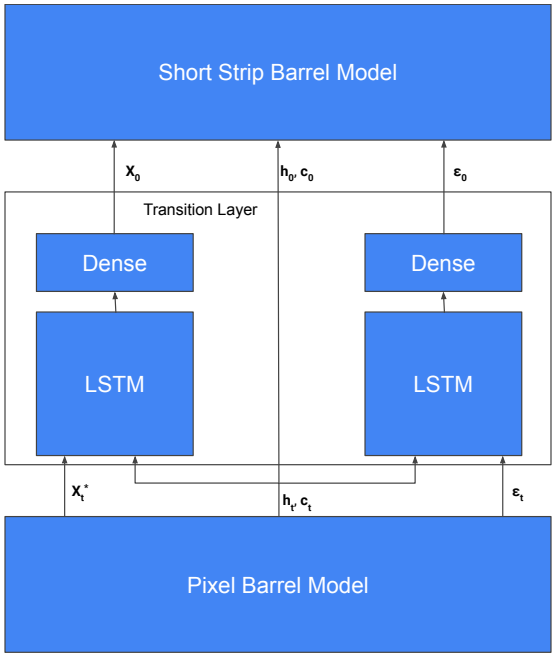


Figure 3-6: The transition layer manages a particle's transition between two volumes. Here, the pixel barrel volume and the short strip barrel volume are shown as an example

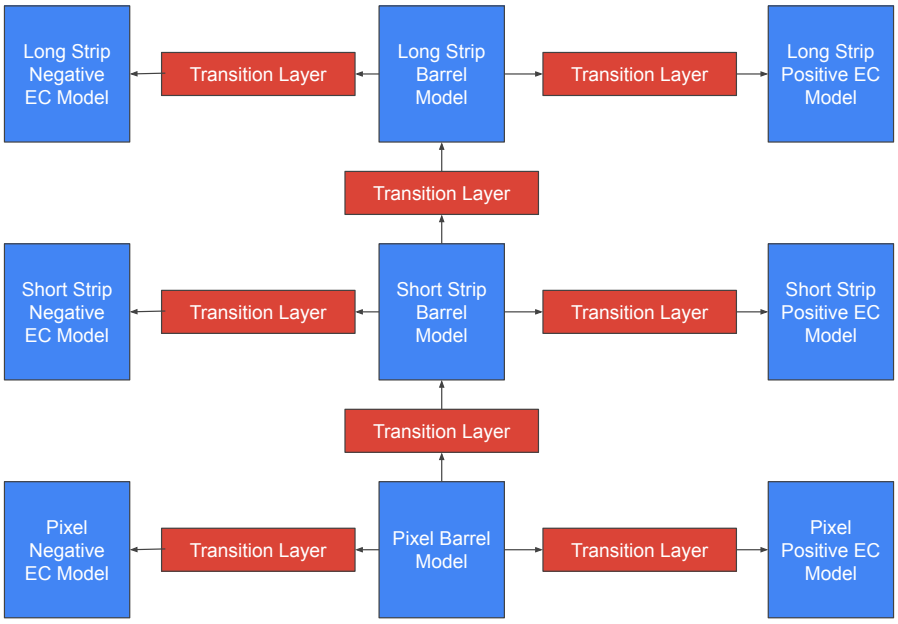
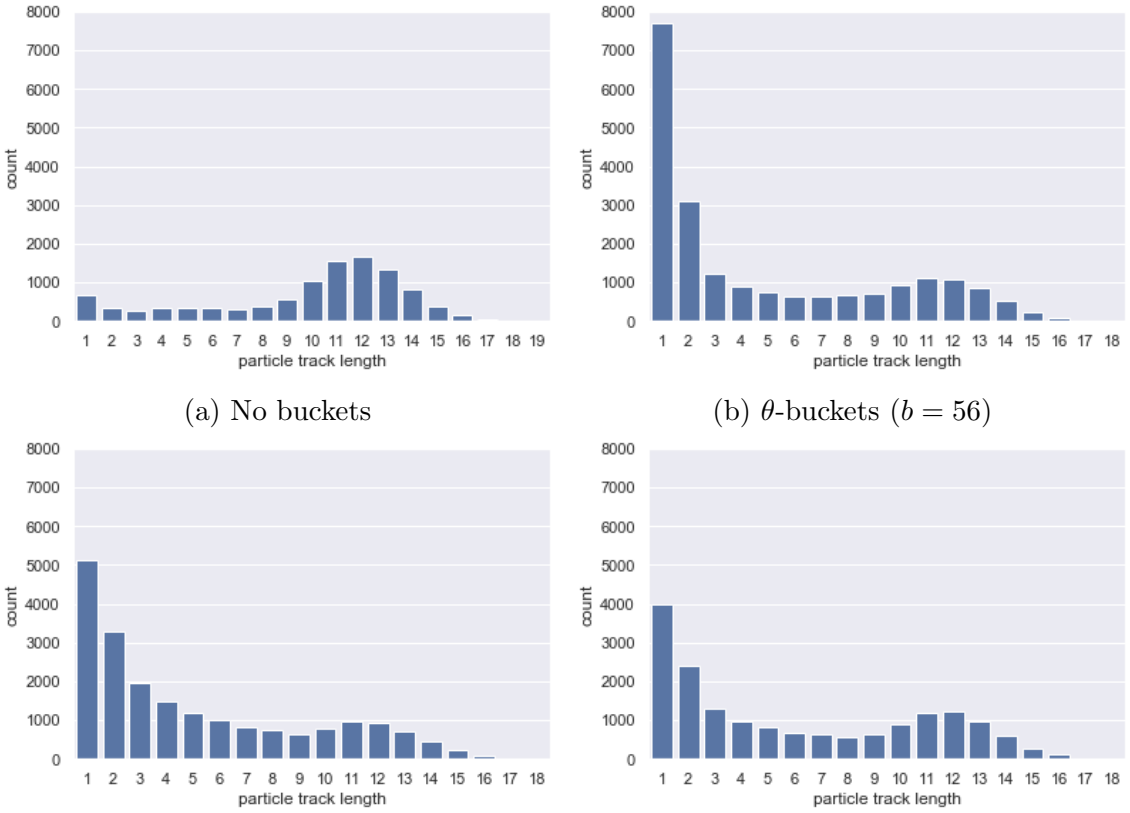


Figure 3-7: Graph depicting how the volumes are transitioned to for tracking the full detector space



(a) No buckets

(b) θ-buckets ($b = 56$)

(c) ANN-buckets ($k = 3500$)

(d) θ / ϕ-buckets ($b_\theta = 14$, $b_\phi = 4$)

Figure 3-8: Distribution of track length per particle per bucket with different bucketing strategies

Chapter 4

Tracker Implementation

4.1 The TrackML Dataset

The collision event data was obtained from the TrackML Kaggle competition [8], which also provides a library for loading the data and scoring tracker outputs. The data for each event is split into four CSV files: `hits`, `cells`, `particles` and `truth`. The `hits` file contains information on all measurements: Their spatial position $\boldsymbol{m_t}$, the volume and layer they were recorded on, and a unique identifier for each hit. The `truth` file includes the true positions of where the particles intercepted the individual detector layers $\tilde{\boldsymbol{x}}_{\boldsymbol{t}}$ and the particle's assignment to the hits. The `cells` file contains more specific information on each hit, namely the exact pixels where the hits were recorded. Finally, the `particles` file gives further information on each particle, like its initial position and velocity. On average, there are about 100,000 hits created from 9,300 particles contained in each event. For the purposes of this tracker, only the `hits` and `truth` files were used.

Before feeding the data to the model, additionally to offline pre-processing (see section 4.3), the data is prepared in an online-manner to bring it into the format expected by the particle tracker: The measurements and true particle positions are stored in tensors of constant size, with the number of particles/measurements set to the maximum expected number of particles/measurements. Empty spots are filled with a row of zeros. To indicate which spots are filled and which are empty, boolean matri-

ces are fed to the model. Furthermore, the tensor of true particle-to-measurement-associations $\tilde{\boldsymbol{A}}$ is created.

4.2 Model Implementation

The tracker [1] was implemented using Python with the deep learning libraries Tensorflow [1] and Keras [19]. The Keras API was mostly used for higher-level code, such as creating and compiling models, organizing the model into self-contained layers, managing the models' inputs and outputs, and the losses. Tensorflow, on the other hand, was applied for lower level custom calculations, such as computing custom losses, pairwise distances, and implementing the weighting procedure in the update step of the motion model. To reduce code complexity and support software design best practices, such as low coupling between components, individual logical units of the tracker were implemented as self-contained subclasses of `tf.keras.layers.Layer` or `tf.keras.Model`, depending on the context and requirements. For example, the motion model's prediction step was implemented as a subclass of `tf.keras.layers.Layer` and the full motion model as a `tf.keras.Model`. A major difference between `Model` and `Layer` in Keras is that a `Model` can be trained on its own while a `Layer` can be described as a building block of a `Model` and therefore does not provide training functionality to the user. However, just like a `Layer`, a `Model` can also be used as a component of another `Model`. For the tracker presented in this thesis, this has the advantage that while both the motion model and the association model can be trained independently, the motion model can also directly call the association model from right within its model definition. If the motion model should be trained independently of the association model, the motion model uses the true particle-to-measurement-assignments instead of calling the association model. Another example where `Model`s are used as part of another `Model` is for implementing the approach to track particles throughout multiple volumes (as discussed in section 3.2.5), where the models for each volume represent distinct instances of `Model` and

[1] Source code available at `https://gitlab.com/hofmann-master-thesis/trackml`

are contained in another `Model` which arranges the individual volumes' models in the correct structure. This modularity also allows for easy swapping of individual components within the models, such as changing the association model used from LSTM-based association to the faster dense association.

4.3 Data Pre-Processing Pipeline

Before the tracker can use the data, several pre-processing steps are required, including the computation of the angles ϕ and θ, the assignment of the hits to buckets, and the normalization of the spatial dimensions. These mentioned steps are done offline, i.e., the entire data set is transformed before any data is fed to the model. Once a transformation of a single event is complete, it is stored on disk to free up the memory. This also means that when a model is trained the next time, the transformations are already available on disk and do not have to be recomputed. To facilitate offline pre-processing, the library Luigi [35] is employed, which allows the creation of data pipelines that can be executed in parallel. For each transformation, a separate `luigi.Task` is implemented, and these tasks can be combined and configured to operate on multiple events at once:

```
import load_data_luigi as ld
events = ld.RootRangeDetectorFiles(start_range=1000,
                                   end_range=1100)
events = ld.DerivedRangeDetectorFiles(create_from=events,
                                      derive_task=ld.CreateAngles)
events = ld.DerivedRangeDetectorFiles(create_from=events,
                                      derive_task=ld.CreateNormalized)

for event in events.load():
    ...
```

In the code example above, the events with the IDs 1000 to 1100 are transformed to include the angles θ and ϕ and normalized spatial dimensions. Upon the first call

of the method `events.load()`, Luigi is invoked to build the necessary files: First, it checks if the files corresponding to the transformations specified are already saved on disk. If they are not, Luigi creates the files containing the transformations for all events in parallel. Finally, the transformed events are loaded one by one and supplied to the loop using a Python generator.

4.4 Model Management and Persistence

The management of the model structure poses special challenges because while the model's modules can be swapped easily, a wide variety of different model architectures and combinations of different modules can be configured. The goal was to make it possible to specify all model parameters and components for the entire tracker from a single place in the code to prevent having to make changes in large parts of the code base when re-configuring the model architecture. This was achieved by implementing a class `TrackerConfiguration`, which contains the information needed for building the model, like, for example, which architecture to use for the association model or the number of neural units in specific components. In total, 34 distinct parameters can be configured using this class. When creating a new model, an instance of the `TrackerConfiguration` class is passed to the tracker, which is then passed on to the individual components when they are initialized. The components then extract the parameters required for their own configuration and, if necessary, pass the `TrackerConfiguration` object to their sub-components.

A second problem that is solved using the `TrackerConfiguration` class is persistence: When saving and loading a model, it is necessary to take care of the architecture of the model, which is saved/loaded. If model weights from a model with mismatching architecture than the one constructed in memory are loaded, the model will likely not perform as expected (or the weights will not load at all because the deep learning framework recognizes the architectural mismatch). Usually, Keras offers functionality to save a model's architecture together with the weights, but for my tracker, the framework cannot persist the model structure, likely because of the

high complexity of the model. Thus, only the trainable weights are saved, and when attempting to load a model, the appropriate architecture is re-constructed first, and then the weights are loaded into the model. This is facilitated by binding the persisted weights of a given architecture to the `TrackerConfiguration` of this architecture: The parameters in the `TrackerConfiguration` are hashed, and the resulting identifier is saved in the name of the file where the persisted weights of the respective architecture are stored.

4.5 Model Training

The training of the tracker was performed on an Nvidia GeForce GTX 1080 Ti GPU. For training and evaluation of the models, a train-validation-test approach was employed: As training data, the events 1000-1399 were used. The models were validated on the events 1400-1431 and tested on 1432-1479. For evaluation purposes, trackers were trained both on particle subsets with no bucketing and full events using the θ/ϕ bucketing strategy.

When training a model, the motion model was trained independently from the association model as a first step. This was done by using the matrix of true particle-to-measurement-associations $\tilde{\boldsymbol{A}}_{\boldsymbol{t}}$ instead of estimated association scores $\boldsymbol{A}_{\boldsymbol{t}}$. Afterwards, the resulting motion model was combined with an association model, and the two were trained together. This allows for faster convergence of the model. [27]

Regarding the network parameters, the number of hidden units for the recurrent cells in the motion model was set to 200. It is important to note here that the motion model operates on each particle independently (i.e., the particles are contained in the batch-dimension). Thus, the number of hidden units can be set independently of the total number of particles to track. For the dense association model, one layer of row-wise and column-wise dense elements was employed, with the number of neural units for rows and columns set to the number of particles and measurements, respectively. When using LSTM association, two layers were used, and the number of hidden units was set proportional to the number of particles. As optimizer, RMSProp [15] was

used, and the learning rate is set similarly as [27] does it: Initially, it is set to 0.003, and each 20,000 training steps, it is decreased by 5% (this can be achieved in Keras using `tf.keras.optimizers.schedules.ExponentialDecay`). The ρ parameter of RMSProp is set to 0.9 and momentum to 0.0.

The hyperparameters of the loss function

$$\begin{aligned}\mathcal{L}(\boldsymbol{x}^*, \boldsymbol{x}, \varepsilon, \boldsymbol{A}, \tilde{\boldsymbol{x}}, \tilde{\boldsymbol{A}}, \tilde{\varepsilon}) =& \lambda \frac{1}{n} \sum \|\boldsymbol{x}^* - \tilde{\boldsymbol{x}}\|^2 + \kappa \frac{1}{n} \sum \|\boldsymbol{x} - \tilde{\boldsymbol{x}}\|^2 + \\ & \nu \mathcal{L}_\varepsilon + \xi \varepsilon^* - \psi \sum_{i=1}^{n} \sum_{j=1}^{m} \tilde{A}[i,j] \log(A[i,j])\end{aligned} \tag{4.1}$$

were configured as follows:

- $\lambda = 10$
- $\kappa = 10$
- $\nu = 0.2$
- $\xi = 0.2$ (if regularization was performed)
- $\psi = 0.1$

The data is fed to the model in batches containing 12 instances each. In the case of bucketed tracking, each bucket represents a single instance.

Chapter 5

Evaluation & Discussion

5.1 Tracking Particle Samples

To get an idea of which model architecture and combination of components perform well on the data set, five different architectures were trained on particle samples and evaluated according to the measure used for scoring submissions at the TrackML Kaggle competition [8]. This measure uses only the predicted assignment of the hits to tracks and the actual assignment of hits to particles as input. It does not use any other predicted information on the particles' positions or their trajectories. The measure is computed as follows: First, the predicted tracks are each matched to a so-called leading particle. Such a matching happens if more than 50% of the hits in this track belong to a single particle. If no particle can be matched using this approach, the track is discarded and scored with 0. The track is also scored with 0 if a track contains less than 50% of the leading particle's hits. To compute the score, the sum of the weights of all hits that were assigned to the correct particle and are part of a matched track is divided by the total weights of all hits. Thus, the score reaches 1 for a perfect assignment and is 0 for a random assignment.

The predicted assignments of the particles to the hits were computed with two different strategies: With existence correction and without existence correction. The assignments without existence correction are taken directly from the matrix of predicted assignments $\boldsymbol{A}$. More precisely, each hit is assigned to the particle with the

highest score according to $\boldsymbol{A}$. For finding the assignments with existence correction, an assignment matrix $\boldsymbol{A}^{(\boldsymbol{\varepsilon})}$ which also considers the existence scores $\boldsymbol{\varepsilon}$ is computed first:

$$A^{(\varepsilon)}[i,j] = A[i,j] \cdot \varepsilon[i] \tag{5.1}$$

The assignment is then determined by finding the maximum entry in $\boldsymbol{A}^{(\boldsymbol{\varepsilon})}$ for each measurement. The weighting by the existence scores $\boldsymbol{\varepsilon}$ makes an assignment to particle tracks that do not exist on a given detector layer less likely and more likely for tracks that do exist.

For sampling particles from the event data, a random set of n particles was selected uniformly at random from the full set of particles contained in a single event. Then, their hits were extracted, and this data was then fed to the tracker. This allows the tracker to be trained and evaluated at a lower particle count and particle density, which should, on the one hand, demonstrate that the tracker can track a subset of the full data set, but also allow one to tune the model, as the training time for a subset is lower than the one for a full event.

5.1.1 Sample of 20 Particles

Different versions of the tracker were first evaluated on samples of 20 particles. The results are shown in table 5.1. The tracker architectures are listed by which recurrent structure is used in the motion model (RNN or LSTM), which approach is used in the association model (LSTM vs. Dense), whether or not existence re-weighting is performed in the update model (see also equation (3.9)) and whether or not the existence regularization ε^* is used in the loss (see equation (3.6)).

The first model tested is a re-implementation of the model proposed by [27], adapted for particle tracking: It uses simple RNNs in the motion model, an LSTM for association, it performs an existence re-weighting step and uses the existence regularization term. As one can see, when calculating the average score based on the association matrix with existence correction $\boldsymbol{A}^{(\boldsymbol{\varepsilon})}$ (Ex. Corr.) the score is higher than when using the regular association matrix $\boldsymbol{A}$ (No Ex. Corr.). If the existence

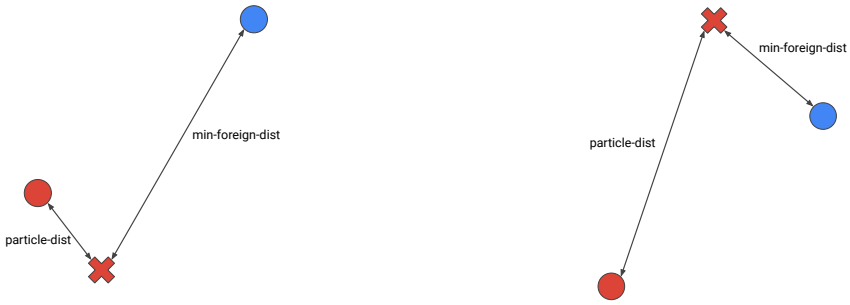
	Model				Avg. Tracking Score	
ID	**Motion**	**Assoc.**	**Ex. Weigh.**	**Ex. Reg.**	**No Ex. Corr.**	**Ex. Corr.**
1	RNN	LSTM	Yes	Yes	0.467	0.513
2	RNN	LSTM	Yes	No	0.463	0.525
3	LSTM	LSTM	Yes	No	0.642	**0.669**
4	LSTM	Dense	Yes	No	0.638	0.664
5	LSTM	Dense	No	No	0.613	0.660

Table 5.1: Tracking scores of different model combinations for samples of 20 particles

regularization term is omitted in the loss, as it is done in model #2, the model gets slightly better. However, the largest gain in tracking performance comes when replacing the RNN in the motion model with a more sophisticated LSTM, as with model #3. It makes tracking accuracy using the assignment with existence correction increase by about 0.14, and the one without existence correction even increases by approximately 0.18. Model #3 has the scores maxed out. When replacing the LSTM in the association model with the dense model or not doing existence re-weighting (models #4 and #5), the scores without existence correction decrease, while the scores with existence correction stay roughly the same. However, it should be noted here that as discovered in section 3.2.2, the dense association model is significantly faster than the one using an LSTM.

5.1.2 Sample of 200 Particles

To closer investigate the proposed algorithm's scaling behavior, a model similar to architecture #5 was trained on samples containing 200 particles. This is a more challenging problem than a sample of 20 particles, not only because the tracks of more particles have to be predicted, but also because the density of hits on each detector layer is higher, making it harder to distinguish which hit belongs to which particle. The results for 200 particles show a significant decrease in tracking performance to 0.202 without existence correction or 0.243 with existence correction.



(a) Desired case (b) Undesired case

Figure 5-1: Visualization of prediction quality. Predictions are marked as crosses, hits as circles

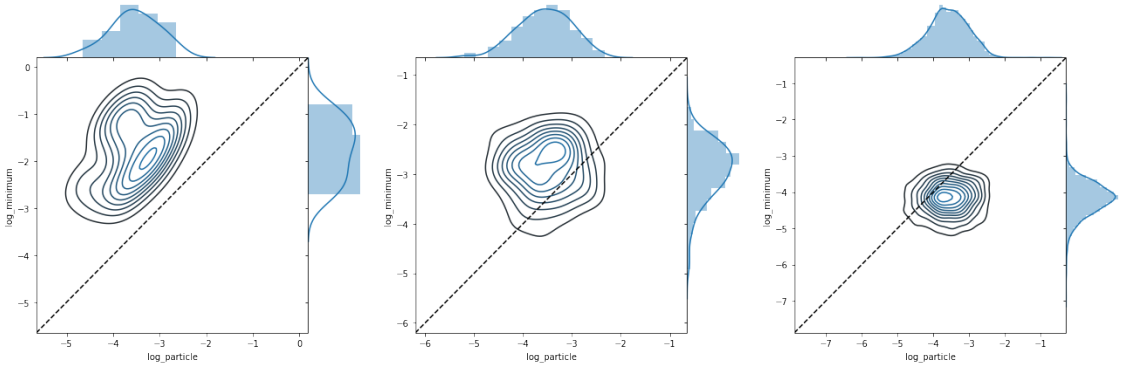
5.2 Tracking Full Events of About 10,000 Particles

The entire set of particles contained in one event was attempted to be tracked using a tracker with an architecture similar to model #5 from Table 5.1 running on θ/ϕ buckets with $b_\theta = 14$ and $b_\phi = 4$. Unfortunately, the tracker was not able to handle this high number of particles. The average tracking score did not go significantly above 0.0. In comparison, the best possible theoretical tracking score, which would result when tracking all particles perfectly within the buckets for the bucketing approach used, would be approximately 0.816. A possible reason why the tracker performs this way on the full set of particles is given in the following section.

5.3 Accuracy Considerations

5.3.1 Performance on the Full Data Set

To explain why running the tracker on the full data set yields this unsatisfying result, a closer evaluation of the quality of the particle state predictions was performed. To explain the main criterion of this evaluation, let us first consider Figure 5-1. In these graphs, predictions are marked as crosses and hits as circles. On the left figure, the desired case, the particle state prediction is closer to the respective particle's hit on this detector layer than to the next hit of a foreign particle. Thus, the distance to the particle's own hit (particle-dist) is smaller than the distance to the closest foreign hit

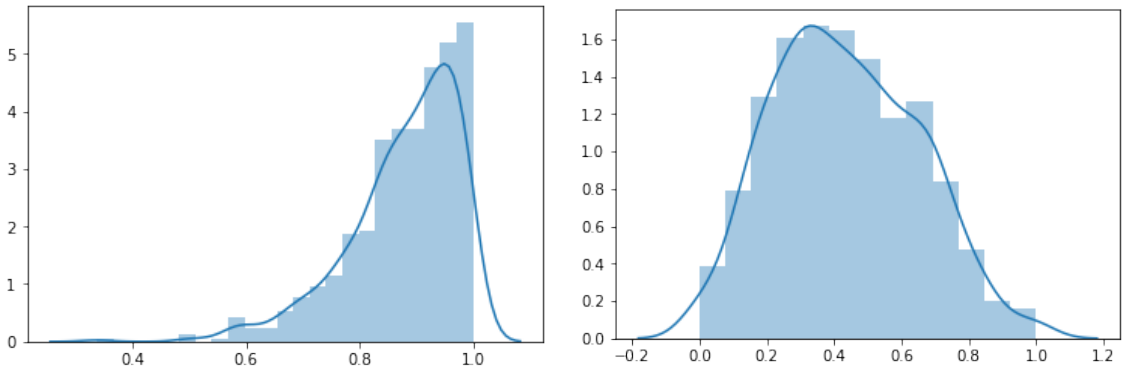


(a) 20 particles (b) 200 particles (c) Full set of particles

Figure 5-2: Minimum distances vs. particle distances for three event sizes (log scale)

(min-foreign-distance). In the context of tracking, it is desired that particle-dist is smaller than min-foreign-dist, because this means that it is straightforward to assign the particle to the correct hit, namely by choosing that hit which is closest to the particle's prediction. If, however, the particle-dist is larger than the min-foreign-dist, one can expect that, while not always impossible, it is much harder to assign the particle to the correct hit.

To reduce complexity, the evaluation of the quality of an LSTM's prediction of the particle states was not performed on the full tracker, but on a simpler architecture consisting only of an LSTM which receives the particle's states on the first three layers of the pixel barrel volume as input and should then predict the states on the fourth layer. To map the LSTM's internal state to the particle state vector, a dense neural layer is employed. Figure 5-2 displays the relations between particle-dist on the x-axis and min-foreign-dist on the y-axis for three different particle samples as a contour plot. If a particle prediction is located in the upper left half of the graph, the particle-dist for this prediction is lower than the min-foreign-dist, i.e., this particle represents an example of the desired case. If, however, it is in the lower right half, the min-foreign-dist is lower than the particle-dist, and therefore this particle represents the undesired case. For 20 particles, all predictions are found in the upper left, i.e., represent the desired case. For 200 particles, while some particles are in the lower right, i.e., the undesired case area, the majority still lies in the upper left area. This can also explain our results from tracking particle samples: The model was able



(a) Tracks starting from innermost layer

(b) Tracks starting from other layers

Figure 5-3: Tracking score only considering tracks starting from specific areas of the detector (sample of 20 particles)

to track 20 particles well and could still track a sample of 200 particles partially. However, when looking at the full set of particles, it becomes evident why the tracker has such a hard time tracking this massive amount of particles: For the largest part, the predictions from the LSTM are actually closer to a foreign hit than to the own hit of the particle. For common RNNs, similar results as the ones from LSTM were obtained. Arguably, this makes it very hard, if not impossible, for the association model to assign each prediction to the correct hit. It must be stressed that this result is produced by an LSTM that only looks at the time series of the true particle states on the first three layers, without any added complexity from bucketing approaches or association models. It can be argued that LSTMs, in general, seem to have a hard time predicting the particles' states at a level of accuracy that is necessary at these high particle densities. Therefore, it can be argued that this result represents an upper bound of what the full tracker with an LSTM can achieve, as the shortened tracks resulting from bucketing or the soft assignments produced by the association model would introduce additional uncertainty to the tracker.

5.3.2 Recognizing New Tracks

Another aspect that was evaluated is how the tracker performs on tracks that do not start in the detector's innermost layer, but which originate from some other

layer in the detector. These newly created tracks are caused by so-called hadronic interactions, where charged hadrons interact with the detector material, such that the initial particle is destroyed, and a spray of new particles is created [32]. The authors of [27] claim that their tracker, which was adapted for particle tracking in this thesis, can recognize new tracks using the birth/death step. This indicates that the tracker could also be able to identify the tracks created from these hadronic interactions. To evaluate this, Figure 5-3 compares the tracking score for 20 particles when only considering tracks starting in the innermost layer to tracks starting at some other place in the detector. The figure shows that the model can track particles originating from the center way better than those created within the detector layers. The average accuracy (with existence correction) for former particles is 0.877, while for the latter, it is only 0.433. On the positive side, this indicates that the model can track particles originating from the center quite well; on the negative, it seems that the model is not that good at recognizing new tracks, at least for the application of particle tracking.

5.4 Discussion

The goal of this thesis was to find out if it is possible to learn the task of tracking particles entirely from data. The tracker proposed in this thesis was able to track event-subsets from the data set of the TrackML competition [8], which demonstrates that with enough data, it is indeed possible for a machine to learn physical models on its own. Furthermore, it could be shown that tracking approaches from visual MOT, which aim to track objects in a video sequence, can also be adapted and applied to the particle tracking problem. The neural architecture of the algorithm selected for adaptation has been further developed, which improved both accuracy and run time for the particle tracking task. Besides, this thesis contributes a bucketing strategy, which allows particles to be tracked in significantly less than quadratic time, and which penalizes accuracy instead of run time with increased particle count. While the algorithm was able to scale to larger particle counts of about 10,000 particles in terms of run time, it could not do so with respect to accuracy. To explain this result,

it was demonstrated that particle state predictions performed by LSTMs seem to be too inaccurate for a tracker to distinguish between particle tracks at such high particle densities correctly. To the best of the author's knowledge, there is no scientifically published deep learning-based algorithm that achieves good accuracy on these high particle densities. The only other DL-algorithm for particle tracking published [38] is evaluated on a maximum of 22 particles, while the algorithm presented in this paper was able to scale to up to 200 particles with some accuracy. While not being able to scale to a particle count of 10,000, it could still be shown that a model from computer vision can be adapted to track particles. Thus, it was demonstrated that deep neural networks can capture physical phenomena; in this case, the trajectories of particles through space. It would be interesting to see if and how machine learning and deep neural networks can model further physical interactions other than particles' trajectories. Being able to learn physical models from data could arguably be a very useful tool for physicists.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, I have explained the task of particle tracking, have discussed similar problems to the particle tracking problem that reside in other scientific realms, like the area of multiple object tracking (MOT) in computer vision, and have compared them to particle tracking. I have given an overview of how particle tracking is currently done at the CERN ATLAS detector. I have reviewed how recurrent neural networks are built, given a glimpse of LSTM as a version of RNN that is better at learning long-time dependencies, and explained why the usage of RNNs can be beneficial to the particle tracking task. I have described promising approaches for tracking multiple objects using recurrent neural networks from the research areas computer vision, bio-medicine, and particle physics and selected one approach that seemed the most promising for the particle tracking task. After some adaptations, like the tracked features, the ability to track throughout orthogonal detector volumes, using an LSTM for target motion, and an adapted model for data association, the tracker was implemented in Python with Tensorflow and Keras. The model's evaluation showed promising results for smaller sets of particles. Hence the model was adapted for tracking events with a larger number of particles (7,000 - 12,000), as they are provided in the Kaggle TrackML challenge [8] using a bucketed scaling approach. Unfortunately, the model was not able to accurately scale to this large data set sizes,

likely because RNNs and LSTMs were not able to predict a particle's position to a level of accuracy that is required for these high particle densities. Nonetheless, the thesis shows that, up to a certain extent, neural networks can capture the physics of particle trajectories only by learning from data.

6.2 Future Work

An obvious possibility for future work is to further investigate the scaling to the full data set. As the LSTM's state predictions seem to lack accuracy for this application, a possibility could be to perform the state predictions using other, possibly non-recurrent neural networks. Another possibility that could be investigated is using other distance functions for computing the pairwise distance between particle state predictions and hits. The tracker presented in this thesis employs the widely used Euclidean distance for this task. Instead, one could try to use the pairwise distance on learned embeddings of the predicted states and the hits, which could reduce the distance between a predicted particle state and its hit. Finally, one could also think of an approach similar to the Combinatorial Kalman Filter (see section 2.2.1), in which the neural network does not create a single track for one particle, but is also able to give birth to multiple track candidates if it encounters multiple hits which could likely be part of the particle's track. The best track candidates could then be selected using some - possibly learned - quality measure that estimates how likely a track candidate is produced by a particle.

Bibliography

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] AI United Redaktion. Recurrent neural networks und LSTM — künstliche intelligenz - ki und machine learning by ai-united. `http://www.ai-united.de/recurrent-neural-networks-und-lstm/`, n.d. [Online, Accessed 29-August-2020].

[3] Amrouche, S., Golling, T., Kiehn, M., Plant, C., and Salzburger, A. Similarity hashing for charged particle tracking. In *2019 IEEE International Conference on Big Data (Big Data)* (2019), pp. 1595–1600.

[4] Anton Milan. Dimensional mismatch in dot-product. `https://bitbucket.org/amilan/rnntracking/issues/26/dimensional-mismatch-in-dot-product`, 2018. [Online, Accessed 17-August-2020].

[5] Babenko, B., Yang, M., and Belongie, S. Robust object tracking with online multiple instance learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence 33*, 8 (2011), 1619–1632.

[6] Blackman, S. S. Multiple hypothesis tracking for multiple target tracking. *IEEE Aerospace and Electronic Systems Magazine 19*, 1 II (2004), 5–18.

[7] Breitenstein, M. D., Reichlin, F., Leibe, B., Koller-Meier, E., and Gool, L. V. Robust tracking-by-detection using a detector confidence particle filter. In *2009 IEEE 12th International Conference on Computer Vision* (2009), pp. 1515–1522.

[8] CERN. TrackML Particle Tracking Challenge. `https://www.kaggle.com/c/trackml-particle-identification`, 2018. [Online, Accessed 08-May-2020].

[9] CERN. High Luminosity LHC. `https://home.cern/science/accelerators/high-luminosity-lhc`, 2020. [Online, Accessed 08-May-2020].

[10] CERN Education Communication and Outreach Group. The LHC FAQ Guide. `http://cds.cern.ch/record/2255762/files/CERN-Brochure-2017-002-Eng.pdf`, 2017. [Online, Accessed 08-May-2020].

[11] Connor, J. T., Martin, R. D., and Atlas, L. E. Recurrent Neural Networks and Robust Time Series Prediction. *IEEE Transactions on Neural Networks 5*, 2 (1994), 240–254.

[12] Cornelissen, T., Elsing, M., Fleischmann, S., Liebig, W., Moyse, E., and Salzburger, A. Concepts, Design and Implementation of the ATLAS New Tracking (NEWT). Tech. Rep. ATL-SOFT-PUB-2007-007. ATL-COM-SOFT-2007-002, CERN, Geneva, Mar 2007.

[13] Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining* (1996), KDD'96, AAAI Press, p. 226–231.

[14] Frühwirth, R. Application of Kalman filtering to track and vertex fitting. *Nuclear Inst. and Methods in Physics Research, A 262*, 2-3 (1987), 444–450.

[15] Hinton, G., Srivastava, N., and Swersky, K. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent, 2012.

[16] Hochreiter, S. Untersuchungen zu dynamischen neuronalen netzen. Master's thesis, Institut fur Informatik, Technische Universitat, München, 1991.

[17] Hochreiter, S., and Schmidhuber, J. J. Long short-term memory. *Neural Computation 9*, 8 (1997), 1–32.

[18] Kalman, R. E. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering 82*, 1 (1960), 35.

[19] Keras Team. Keras: the python deep learning API. `https://keras.io`, 2016-2020. [Online, Accessed 26-August-2020].

[20] Kim, C., Li, F., Ciptadi, A., and Rehg, J. M. Multiple hypothesis tracking revisited. In *2015 IEEE International Conference on Computer Vision (ICCV)* (2015), pp. 4696–4704.

[21] Kuhn, H. W. The hungarian method for the assignment problem. *Naval research logistics quarterly 2*, 1-2 (1955), 83–97.

[22] Leal-Taixé, L., and Canton-Ferrer, C. Learning by tracking: Siamese cnn for robust target association. In *2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (2016), pp. 418–425.

[23] Li, X., Wang, K., Wang, W., and Li, Y. A multiple object tracking method using kalman filter. In *The 2010 IEEE International Conference on Information and Automation* (2010), pp. 1862–1866.

[24] Luo, W., Xing, J., Milan, A., Zhang, X., Liu, W., Zhao, X., and Kim, T.-K. Multiple Object Tracking: A Literature Review. *ArXiv eprints* (2017).

[25] Mankel, R. A concurrent track evolution algorithm for pattern recognition in the HERA-B main tracking system. *Nuclear Instruments and Methods in Physics Research, Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 395*, 2 (1997), 169–184.

[26] Milan, A., Leal-Taixe, L., Reid, I., Roth, S., and Schindler, K. Mot16: A benchmark for multi-object tracking. *ArXiv eprints* (2016).

[27] Milan, A., Rezatofighi, H., Dick, A., and Reid, I. Online multi-target tracking using recurrent neural networks. *ArXiv eprints* (2016).

[28] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *ArXiv eprints* (2013).

[29] Padfield, D., Rittscher, J., and Roysam, B. Coupled minimum-cost flow cell tracking for high-throughput quantitative analysis. *Medical Image Analysis 15*, 4 (2011), 650–668.

[30] Possegger, H., Mauthner, T., and Bischof, H. In defense of color-based model-free tracking. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015).

[31] Reid, D. B. An algorithm for tracking multiple targets. *Automatic Control, IEEE Transactions on 24*, 6 (1979), 843–854.

[32] Rousseau, D., Amrouche, S., Calafiura, P., Farrell, S., Germain, C., Gligorov, V. V., Golling, T., Gray, H., Guyon, I., Hushchyn, M., Hrdinka, J., Innocente, V., Kiehn, M., Salzburger, A., Ustyuzhanin, A., Vlimant, J.-R., and Yilmaz, Y. Particle Tracking Machine Learning Challenge: Detector and Dataset, 2018.

[33] Selvin, S., Vinayakumar, R., Gopalakrishnan, E. A., Menon, V. K., and Soman, K. P. Stock price prediction using lstm, rnn and cnn-sliding

window model. In *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)* (2017), pp. 1643–1647.

[34] Spotify. GitHub - spotify/annoy: Approximate nearest neighbors in c++/python optimized for memory usage and loading/saving to disk. `https://github.com/spotify/annoy`, 2013-2020. [Online, Accessed 25-August-2020].

[35] Spotify. GitHub - spotify/luigi: Luigi is a python module that helps you build complex pipelines of batch jobs. it handles dependency resolution, workflow management, visualization etc. it also comes with hadoop support built in. `https://github.com/spotify/luigi`, 2014-2020. [Online, Accessed 26-August-2020].

[36] Stalder, S., Grabner, H., and Van Gool, L. Cascaded confidence filtering for improved tracking-by-detection. In *Proceedings of the 11th European conference on Computer vision: Part I* (2010), vol. 6311, pp. 369–382.

[37] Sun, Y., Liang, D., Wang, X., and Tang, X. DeepID3: Face Recognition with Very Deep Neural Networks. *ArXiv eprints* (2015).

[38] Tsaris, A., Anderson, D., Bendavid, J., Calafiura, P., Cerati, G., Esseiva, J., Farrell, S., Gray, L., Kapoor, K., Kowalkowski, J., Mudigonda, M., Spentzouris, P., Spiropoulou, M., Vlimant, J.-r., Zheng, S., and Zurawski, D. The HEP. TrkX Project : Deep Learning for Particle Tracking. In *Proceedings, Connecting The Dots / Intelligent Tracker (CTD/WIT 2017)* (2018).

[39] Wojke, N., Bewley, A., and Paulus, D. Simple online and realtime tracking with a deep association metric. In *2017 IEEE International Conference on Image Processing (ICIP)* (2017), pp. 3645–3649.

[40] Yao, Y., Smal, I., and Meijering, E. Deep neural networks for data association in particle tracking. In *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)* (2018), pp. 458–461.

[41] Zhang, L., Li, Y., and Nevatia, R. Global data association for multi-object tracking using network flows. In *26th IEEE Conference on Computer Vision and Pattern Recognition, CVPR* (2008).

[42] Zhang, L., and van der Maaten, L. Structure preserving object tracking. In *2013 IEEE Conference on Computer Vision and Pattern Recognition* (2013), pp. 1838–1845.

Appendices

Appendix A

Abstract

A.1 Deutsch

Diese Arbeit behandelt einen neuen, auf Deep Learning basierenden Algorithmus für Particle Tracking, der nur auf Basis von Daten gelernt werden kann. Particle Tracking ist ein Problem aus der Hochenergiephysik, wobei Materieteilchen in großen Teilchenbeschleunigern, wie dem LHC am CERN, innerhalb eines Teilchendetektors zur Kollision gebracht werden. Durch diese Kollisionen werden neue Teilchen geschaffen, welche vom Detektor an mehreren Orten gemessen und aufgezeichnet werden. Diese Aufzeichnungen werden auch "Hits" genannt. Ein Particle Tracker hat nun die Aufgabe, alle Hits, die von einem einzelnen Teilchen stammen, zu verbinden und so die Flugbahn des Teilchens zu rekonstruieren. Der Tracker, der in dieser Arbeit vorgestellt wird, basiert auf einem existierenden Ansatz aus dem Bereich der Computer Vision, mit dem Objekte, wie zum Beispiel Personen, innerhalb eines Videos getrackt werden können. Der Tracker kann eine kleinere Anzahl an Teilchen gleichzeitig verfolgen. Dabei verwendet er End-To-End Learning, also das Lernen alleine auf Basis von Daten und ohne auf explizit implementierte physikalische Modelle zurückzugreifen. Es wird erörtert, was den Tracker daran hindert, auf sehr hohe Teilchendichten zu skalieren, und mögliche Lösungen werden vorgeschlagen.

A.2 English

This thesis presents a novel, deep learning-based algorithm for particle tracking, which can be learned entirely from data. Particle tracking is a problem in high energy physics, where matter particles are accelerated in large particle accelerators, like the LHC at CERN, and then brought to collision within a particle detector. Upon these collisions, new particles are created, which are recorded by the detector at multiple locations. These recordings are called hits. A particle tracker fulfills the task of connecting all hits originating from a single particle to form the particle's trajectory. The tracker presented in this thesis is based on an approach from computer vision, where objects, like persons, are tracked in a video sequence. The presented tracker can track smaller amounts of particles simultaneously using end-to-end learning, i.e., the task of particle tracking is learned only from data and without any explicitly implemented physical models. Possible reasons hindering the tracker from scaling to very large particle densities are explored, and possible solutions suggested.

Appendix B

Notation and Symbols

$\boldsymbol{A_t}$ Estimated probability matrix of object-measurement associations at time t.

$\tilde{\boldsymbol{A}}_{\boldsymbol{t}}$ Binary matrix representing the true object-measurement associations.

$\boldsymbol{D_t}$ Pairwise distance matrix between predicted object / particle states and detections / measurements at time t

ε_t Estimated probability of an object / a particle existing at time t

$\tilde{\varepsilon}_t$ $\in \{0, 1\}$: 1, if an object / a particle exists at time t, 0 otherwise

$\boldsymbol{\varepsilon_t}$ Vector of ε_t for all tracked objects / particles

$\tilde{\boldsymbol{\varepsilon}}_{\boldsymbol{t}}$ Vector of $\tilde{\varepsilon}_t$ for all tracked objects / particle

ε^* Existence regularization term: $\varepsilon_t^* = |\varepsilon_t - \varepsilon_{t-1}|$

$\boldsymbol{M_t}$ State vector matrix of detections / measurements at time t

$\boldsymbol{m_t}$ State vector of a single detection / measurement at time t

$\boldsymbol{X_t}$ Matrix of all objects' / particles' estimated state vectors at time t given the measurements

$\boldsymbol{x_t}$ A single object's / particle's estimated state vector at time t given the measurements

$\boldsymbol{x}_t^*$ A single object's / particle's estimated state vector at time t not given the measurements

$\boldsymbol{X}_t^*$ Matrix of all objects' / particles' estimated state vectors at time t not given the measurements

$\tilde{\boldsymbol{x}}_t$ A single object's / particle's true state vector at time t

Appendix C

List of Abbreviations

ANN Approximate nearest neighbour

API Application programming interface

CERN European Organization for Nuclear Research

CKF Combinatorial Kalman Filter

CNN Convolutional neural network

DL Deep Learning

EC End-cap (end-cap discs in the particle detector)

LHC Large Hadron Collider (particle collider located at CERN)

LSTM Long short-term memory

RNN Recurrent neural network

MOT Multiple object tracking, used in this thesis for referring to visual tracking

NN Neural network