arXiv:1604.03635v1 [cs.CV] 13 Apr 2016

Online Multi-target Tracking using Recurrent Neural Networks

Anton Milan[1] Seyed Hamid Rezatofighi[1] Anthony Dick[1]
Konrad Schindler[2] Ian Reid[1]

[1]School of Computer Science, The University of Adelaide, Australia
[2]Photogrammetry and Remote Sensing Group, ETH Zürich

Abstract. We present a novel approach to online multi-target tracking based on recurrent neural networks (RNNs). Tracking multiple objects in real-world scenes involves many challenges, including *a)* an a-priori unknown and time-varying number of targets, *b)* a continuous state estimation of all present targets, and *c)* a discrete combinatorial problem of data association. Most previous methods involve complex models that require tedious tuning of parameters. Here, we propose for the first time, a full end-to-end learning approach for online multi-target tracking based on deep learning. Existing deep learning methods are not designed for the above challenges and cannot be trivially applied to the task. Our solution addresses all of the above points in a principled way. Experiments on both synthetic and real data show competitive results obtained at ≈300 Hz on a standard CPU, and pave the way towards future research in this direction.

1 Introduction

Tracking multiple targets in unconstrained environments is an extremely challenging task. Even after several decades of research [1–6], it is still far from reaching the accuracy of human labelling. (*cf*. [7]). The task itself constitutes locating all targets of interest in a video sequence and maintaining their identity over time. One of the obvious questions that arises immediately is how to model the vast variety of data present in arbitrary videos that may include different view points or camera motion, various lighting conditions or levels of occlusion, a varying number of targets, *etc*.

Tracking-by-detection has emerged as one of the most successful strategies to tackle this challenge. Here, all "unused" data that is available in a video sequence is discarded and reduced to just a few single measurements per frame. This can be achieved by background subtraction [8] for static cameras, or object detections [9, 10] in a more general case. The task is then to associate each measurement to a corresponding target, *i.e*. to address the problem of data association. Moreover, due to clutter and an unknown number of targets, the option to discard a measurement as a false alarm and a strategy to initiate new targets as well as terminate exiting ones must be addressed.

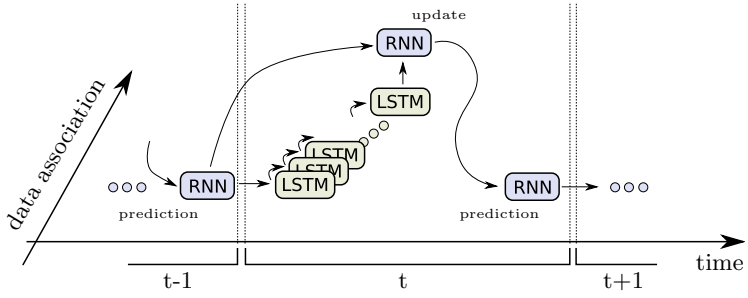

Fig. 1. A schematic illustration of our architecture. We use RNNs for temporal prediction and update as well as track management. The combinatorial problem of data association is solved via LSTMs for each frame.

With the recent rise of deep learning, there has been surprisingly little work related to multi-target tracking. We presume that this is due to several reasons. First, when dealing with a huge number of parameters, deep models require huge amounts of training data, which is not yet available in the case of multi-target tracking. Second, both the data and the desired solution can be quite versatile. One is faced with both discrete and continuous variables, unknown cardinality for input and output, and variable lengths of video sequences. One interesting exception in this direction is the recent work of Ondrúška and Posner [11] that introduces deep recurrent neural networks to the task of state estimation. Although this work shows promising results, it only demonstrates its efficacy on simulated data with near-perfect sensor measurements, a known number of targets, and smooth, linear motion. Its applicability to real-world data is therefore rather limited, because the difficulty of tracking an unknown number of targets in presence of measurement failure and clutter is bypassed entirely.

With this paper, we make an important step towards fully end-to-end model learning for online tracking of multiple targets in realistic scenarios. Our main contributions are as follows:

1. Inspired by the well-studied Bayesian filtering idea, we present a recurrent neural network capable of performing all multi-target tracking tasks including prediction, data association, state update as well as initiation and termination of targets within a unified network structure (Fig. 1). One of the main advantages of this approach is that it is completely model-free, *i.e.* it does not require any prior knowledge about target dynamics, clutter distributions, *etc.* It can therefore capture linear (*cf.* Kalman filter), non-linear (*cf.* particle filter), and higher-order dependencies.
2. We further show, that a model for the challenging combinatorial problem of data association including birth and death of targets can be learned entirely from data. This time-varying cardinality component demonstrates that it is possible to utilise RNNs not only to predict sequences with fixed-sized input and output vectors, but in fact to infer unordered sets with unknown cardinality.

3. Permutation of real data, sampling from generative models, as well as a physically-based system all serve to generate arbitrary amounts of synthetic data, which are used for training.
4. Qualitative and quantitative results on simulated and real data show very promising results, confirming the potential of this approach. We firmly believe that it will inspire other researchers to extend this idea and to further advance its performance.

2 Related Work

Multi-object tracking. A multitude of sophisticated models have been developed in the past to capture the complexity of the problem at hand. Early work includes the multiple hypothesis tracker (MHT) [1] and joint probabilistic data association (JPDA) [2]. Both were developed in the realm of radar and sonar tracking but were considered too slow for computer vision applications for a long time. With the advances in computational power, they have found their way back and have recently been re-introduced in conjunction with novel appearance models [12], or suitable approximation methods [13]. A large amount of work focused on simplified models that could be solved to (near) global optimality [3,4,14–19]. Here, the problem is cast as a linear program and solved via relaxation [3,18], shortest-path [14,16] or max-flow algorithms [4]. Conversely, more complex cost functions have been considered in [20–24], but without any theoretical bounds on optimality. The optimization techniques range from quadratic boolean programming [20], over customized alpha-expansion [22] to greedy constraint propagation [24].

Deep learning. Early ideas of biologically inspired learning systems date back many decades [25]. Later, convolutional neural networks (also known as CNNs) and the back propagation algorithm were developed and mainly applied to handwritten digit recognition [26]. However, despite their effectiveness on certain tasks, they could hardly compete with other well-established approaches. This was mainly due to their major limitation of requiring huge amounts of training data in order not to overfit the high number of parameters. With faster multiprocessor hardware and with a sudden increase in labelled data, CNNs have become increasingly popular, initiated by a recent breakthrough on the task of image classification [27]. CNNs achieve state-of-the-art results in many applications [28–30] but are restrictive in their output format. Conversely, *recurrent* neural networks (RNNs) [31] include a loop between the input and the output. This not only enables to simulate a memory effect, but also allows for mapping input sequences to arbitrary output sequences, as long as the sequence alignment and the input and output dimensions are known in advance.

Our work is inspired by the recent success of recurrent neural nets (RNNs) and their application to language modeling [32–34]. However, it is not straightforward to apply the same strategies to the problem of multi-target tracking for numerous reasons. First, the state space is multi-dimensional. Instead of predicting one character or one word, at each time step the state of all targets has to

be considered at once. Second, the state consists of both continuous and discrete variables. The former represents the actual location (and possibly further properties such as velocities) of targets, while a discrete representation is required to resolve data association. Further discrete indicator variables may also be used to infer certain target states like the track state, the occlusion level, *etc*. Third, the desired number of outputs (*e.g.* targets) varies over time. In this paper, we introduce a method for addressing all these issues and demonstrate how RNNs can be used for end-to-end learning of multi-target tracking systems.

3 Background

3.1 Recurrent Neural Networks

Recurrent neural nets (RNNs) are a particular type of deep architectures that have been applied to numerous tasks including machine translation [35], handwriting recognition and synthesis [36], speech recognition [37], image caption generation [32, 33], object detection [38] and many more. Their distinct power lies in their ability to capture sequential dependencies, providing a mechanism that can be interpreted as a memory capability.

Broadly speaking, RNNs work in a sequential manner, where a prediction is made at each time step, given the previous state and possibly an additional input. The core of an RNN is its hidden state $h \in \mathbb{R}^n$ of size n that acts as the main control mechanism for predicting the output, one step at a time. In general, RNNs may have multiple layers $l = 1, ..., L$. We will denote h_t^l as the hidden state at time t on layer l. h^0 can be thought of as the input layer, holding the input vector, while h^L holds the final embedded representation used to produce the desired output y_t. The hidden state for a particular layer l and time t is computed as

$$h_t^l = \tanh W^l \left(h_t^{l-1}, h_{t-1}^l\right)^\top, \tag{1}$$

where W is a matrix of learnable parameters. Note that $W^l \in \mathbb{R}^{n \times 2n}$ varies on each layer of the network, but does not depend on t.

RNNs of this form have several drawback. First, the simple coupling of the hidden state in Eq. (1) to its previous neighbours in space and time is not strong enough to capture long-term dependences. Second, one is faced with the problem of so-called vanishing gradients, where individual activations become saturated such that the gradient does not propagate sufficiently through time. To remedy this, two types of extensions have been proposed: the long short-term memory (LSTM) [39] and the gated recurrent unit (GRU) [40]. In this work, we only consider the former for our purpose. Next to the hidden state, the LSTM unit also keeps an embedded representation of the state c that acts as a memory. A gated mechanism controls how much of the previous state should be "forgotten" or replaced by the new input (see Fig. 2, right, for an illustration). More formally, the hidden representations are computed as

$$h_t^l = o \odot \tanh\left(c_t^l\right) \tag{2}$$

and

$$c_t^l = f \odot c_{t-1}^l + i \odot g, \tag{3}$$

respectively, where $\odot$ represents element-wise multiplication. The input, output and forget gates are all vectors of size n and model the memory update in a binary fashion using a sigmoid function:

$$i, o, f = \sigma \left[W^l \left(h_t^{l-1}, h_{t-1}^l \right)^\top \right], \tag{4}$$

with a separate weight matrix W^l for each gate.

$$g = \tanh \left[W^l \left(h_t^{l-1}, h_{t-1}^l \right)^\top \right] \tag{5}$$

is the new candidate to replace the current memory, similar to the original RNN formulation from Eq. (1). The complete weight matrix for each layer of an LSTM is $W^l \in \mathbb{R}^{4n \times 2n}$.

3.2 Bayesian Filtering

In this section, we will briefly review the classical Bayesian filtering approach for target tracking. Here, the goal is to estimate the true state x from noisy measurements z. Under the Markov assumption, the state distribution at time t given all past measurements is estimated recursively as

$$p(x_t|z_{1:t}) \propto p(z_t|x_t) \int p(x_t|x_{t-1}) p(x_{t-1}|z_{1:t-1}) dx_{t-1}, \tag{6}$$

where $p(z_t|x_t)$ is the last observation likelihood and $p(x_t|x_{t-1})$ the state transition probability. Typically, Eq. (6) is evaluated in two steps: a *prediction* step that evaluates the state dynamics, and an *update* step that corrects the belief about the state based on the current measurements. Two of the most widely used techniques for solving the above equation are Kalman filter [41] and particle filter [42]. The former performs exact state estimation under linear and Gaussian assumptions for the state and measurements models, while the latter approximates arbitrary distributions using sequential importance sampling.

When dealing with multiple targets, one is faced with two additional challenges. 1) Before the state update can be performed, it is crucial to determine, which measurements are associated with which targets. A number of algorithms have been proposed to address this problem of data association including simple nearest neighbours or similar greedy techniques, and sophisticated probabilistic approaches like JPDA (see [43] for an overview). 2) To allow for a time-varying number of targets, it is necessary to provide a mechanism to spawn new targets that enter the scene, and remove existing ones that disappear indefinitely. Like data association, this task is non-trivial, since each unassigned measurement can potentially be either the start of a new trajectory or a false alarm. Conversely, a missing measurement for a certain target could mean that the target has disappeared, or that the detector has failed. To address this challenge, online tracking approaches typically base their decisions about births and deaths of tracks on heuristics that consider the number of consecutive measurement errors.

4 Our Approach

We will now describe our approach to cast the classical Bayesian state estimation, data association as well as track initiation and termination tasks as a recurrent neural net, allowing for full end-to-end learning of the model.

4.1 Preliminaries and Notation

We begin by defining $x_t \in \mathbb{R}^{N \cdot D}$ as the vector containing the states for all targets at one time instance. In our setting, the targets are represented by their bounding box coordinates (x, y, w, h), such that $D = 4$. Note that it is conceptually straight-forward to extend the state to an arbitrary dimension, *e.g.* to incorporate velocity, acceleration or appearance model. N is the *maximum* number of targets that can be represented (or tracked) simultaneously in one particular frame and x_t^i refers to the state of the i^{th} target. N is what we call the network's *order* and captures the spatial dependencies between targets. In its most general form with $N = 1$, all targets are assumed to move independently. Similar to the state vector above, $z_t \in \mathbb{R}^{M \cdot D}$ is the vector of all measurements in one frame, where M is maximum number of detections per frame. It is important to point out, that there is no limit on the overall number of targets that our model can handle.

The assignment probability matrix $\mathcal{A} \in [0, 1]^{N \times (M+1)}$ represents for each target (row) the distribution of assigning individual measurements to that target, *i.e.* $\mathcal{A}_{ij} = p(i \text{ assigned to } j)$ and $\forall i : \sum_j \mathcal{A}_{ij} = 1$. Note that an extra column in $\mathcal{A}$ is needed to incorporate the case that a measurement is missing. Finally, $\mathcal{E} \in [0, 1]^N$ is an indicator vector that represents the existence probability of a target and is necessary to deal with an unknown and time-varying number of targets. We will use $(\sim)$ to explicitly denote the ground truth variables.

4.2 Multi-target Tracking with RNNs

As motivated above, we decompose the problem at hand into two major blocks: state prediction and update as well as track management on one side, and data association on the other. This strategy has several advantages. First, one can isolate and debug individual components effectively. Second, the framework becomes modular, making it easy to replace each module or to add new ones. Third, it enables one to (pre)train every block separately, which not only significantly speeds up the learning process but can in fact be necessary for joint training. We will now describe both building blocks in detail.

4.3 Target Motion

Let us first turn to state prediction and update. We rely on a temporal RNN depicted in Fig. 2 (left) to learn the temporal dynamic model of N targets jointly as well as an indicator to determine births and deaths of targets (see

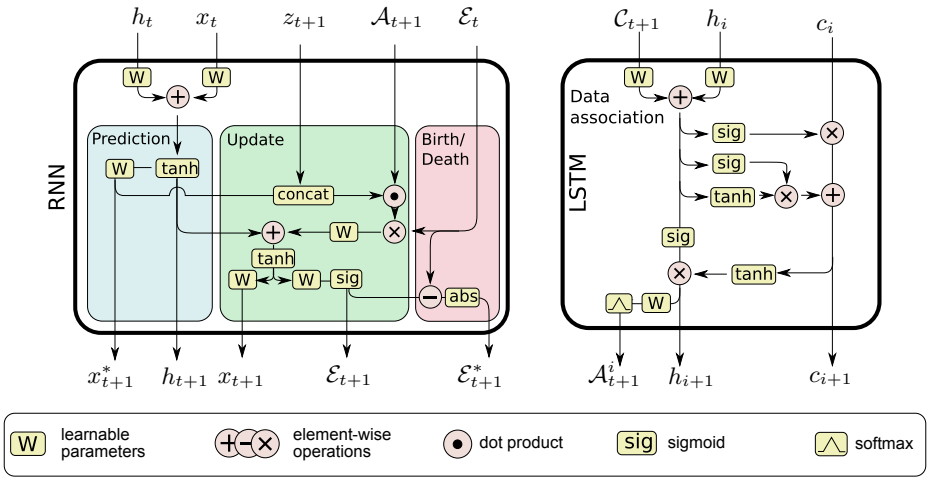

Fig. 2. Left: An RNN-based architecture for state prediction, state update, and target existence probability estimation. Right: An LSTM-based model for data association.

next section). At time t, the RNN outputs four values[1] for the next time step: A vector $x^*_{t+1} \in \mathbb{R}^{N \cdot D}$ of predicted states for all targets, a vector $x_{t+1} \in \mathbb{R}^{N \cdot D}$ of all updated states, a vector $\mathcal{E}_{t+1} \in (0, 1)^N$ of probabilities indicating for each target how likely it is a real trajectory, and $\mathcal{E}^*_{t+1}$, which is the absolute difference to $\mathcal{E}_t$. This decision is computed based on the current state x_t and existence probabilities $\mathcal{E}_t$ as well as the measurements z_{t+1} and data association $\mathcal{A}_{t+1}$ in the following frame. This building block has three primary objectives:

1. **Prediction**: Learn a complex dynamic model for a pre-defined number of targets.
2. **Update**: Learn to correct the state distribution, given target-to-measurement assignments.
3. **Birth / death**: Learn to identify track initiation and termination based on the state, the measurements and the data association.

The prediction x^*_{t+1} for the next frame depends solely on the current state x_t and the network's hidden state h_t. Once the data association $\mathcal{A}_{t+1}$ for the following frame is available (see Sec. 4.5), the state is updated according to assignment probabilities. To that end, all measurements and the predicted state are concatenated to form $\hat{x} = [z_{t+1}; x^*_{t+1}]$ weighted by the assignment probabilities $\mathcal{A}_{t+1}$. At the same time, the track existence probability $\mathcal{E}_{t+1}$ is computed.

Loss. A loss or objective is required by any machine learning algorithm to compute the goodness-of-fit of the model, *i.e.* how close the prediction corresponds to the true solution. It is a continuous function, typically chosen such that minimising the loss maximises the performance of the given task. In our case, we

[1] We omit the RNN's hidden state at this point for clarity.

are therefore interested in a loss that correlates with the tracking performance. This poses at least two challenges. First, measuring the performance of multi-target tracking is far from trivial [22] and moreover highly dependent on the particular application. For example, in vehicle assistance systems it is absolutely crucial to maintain the highest precision and recall to avoid accidents and to maintain robustness to false positives. On the other hand, in sports analysis it becomes more important to avoid ID switches between different players. One of the most widely accepted metrics is the multi-object tracking accuracy (MOTA) that combines these three error types and gives a reasonable assessment of the overall performance. Ideally, one would train an algorithm directly on the desired performance measure. This, however, poses a second challenge. The MOTA computation involves a complex algorithm with non-differentiable zero-gradient components, that cannot easily be incorporated into an analytical loss function. Hence, we propose the following loss that satisfies our needs:

$$\mathcal{L}(x^*, x, \mathcal{E}, \widetilde{x}, \widetilde{\mathcal{E}}) = \underbrace{\frac{\lambda}{ND}\sum \|x^* - \widetilde{x}\|^2}_{\text{prediction}} + \underbrace{\frac{\kappa}{ND}\|x - \widetilde{x}\|^2}_{\text{update}} + \underbrace{\nu\mathcal{L}_{\mathcal{E}} + \xi\mathcal{E}^*}_{\text{birth/death + reg.}}, \tag{7}$$

where x^*, x, and $\mathcal{E}$ are the predicted values, and $\widetilde{x}$ and $\widetilde{\mathcal{E}}$ are the true values, respectively. Note that we omit the time index here for better readability. In practice the loss for one training sample is averaged over all frames in the sequence.

The loss consists of four components. Let us first concentrate on the first two, assuming for now that the number of targets is fixed. Intuitively, we aim to learn a network that predicts trajectories that are close to the ground truth tracks. This should hold for both, predicting the target's motion in the absence of any measurements, as well as correcting the track in light of new measurements. To that end, we minimise the mean squared error (MSE) between state predictions and state update and the ground truth.

4.4 Initiation and Termination

Tracking multiple targets in real-world situations is challenged by the fact that targets can appear and disappear in the area of interest. This aspect must not be ignored. We propose to capture the time-varying number of targets by an additional variable $\mathcal{E} \in (0, 1)^N$ that mimics the probability that a target exists ($\mathcal{E} = 1$) or not ($\mathcal{E} = 0$) at one particular time instance.

Loss. The last two terms of the loss in Eq. (7) guide the learning to predict the existence of each target at any given time. This is necessary to allow for target initiation and termination. Here, we employ the widely used binary cross entropy (BCE) loss

$$\mathcal{L}_{\mathcal{E}}(\mathcal{E}, \widetilde{\mathcal{E}}) = \widetilde{\mathcal{E}} \log \mathcal{E} + (1 - \widetilde{\mathcal{E}}) \log(1 - \mathcal{E}) \tag{8}$$

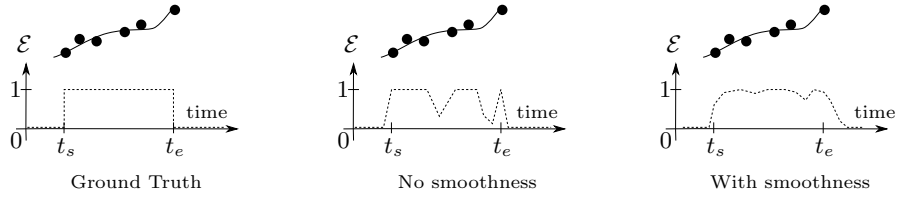

Fig. 3. The effect of the pairwise smoothness prior on the existence probability. See Sec. 4.4 for details.

that approximates the probability of the existence for each target. Note that the true values $\widetilde{\mathcal{E}}$ here correspond to a box function over time (*cf*. Fig. 3, left). When using the BCE loss alone, the RNN learns to make rather hard decisions, which results in track termination at each frame when a measurement is missing. To remedy this, we propose to add a smoothness prior $\mathcal{E}^*$ that essentially minimises the absolute difference between two consecutive values for $\mathcal{E}$.

4.5 Data Association

Arguably, the data association, *i.e*. the task to uniquely classify the corresponding measurement for each target, is the most challenging component of tracking multiple targets. Greedy solutions are efficient, but do not yield good results in general, especially in crowded scenes with clutter and occlusions. Approaches like JPDA are on the other side of the spectrum. They consider *all* possible assignment hypotheses jointly, which results in an NP-hard combinatorial problem. Hence, in practice, efficient approximations must be used.

In this section, we describe an LSTM-based architecture that is able to learn to solve this task entirely from training data. This is somewhat surprising for multiple reasons. First, joint data association is in general a highly complex, discrete combinatorial problem. Second, most solutions in the output space are merely permutations of each other w.r.t. the input features. Finally, any possible assignment should meet the one-to-one constraint to prevent the same measurement to be assigned to multiple targets. We believe that the LSTM's non-linear transformations and its strong memory component are the main driving force that allows for all these challenges to be learned effectively.

To support this claim, we demonstrate the capability of LSTM-based data association on the example of replicating the linear assignment problem. Our model is illustrated in Figures 1 and 2 (right). The main idea is to exploit the LSTM's temporal step-by-step functionality to predict the assignment for each target one target at a time. The input at each step, next to the hidden state h and the cell state c, is the *entire* feature vector. For our purpose, we use the pairwise-distance matrix $\mathcal{C} \in \mathbb{R}^{N \times M}$, where $\mathcal{C}_{ij} = \|x^i - z^j\|_2$ is the Euclidean distance between the predicted state of target i and measurement j. Note that it is straight-forward to extend the feature vector to incorporate appearance or any other similarity information. The output that we are interested in is then a

vector of probabilities $\mathcal{A}^i$ for one target and all available measurements, obtained by applying a softmax layer with normalisation to the predicted values. Here, $\mathcal{A}^i$ denotes the i^{th} row of $\mathcal{A}$.

Loss. To measure the misassignment cost, we employ the widely used negative log-likelihood loss

$$\mathcal{L}(\mathcal{A}^i, \tilde{a}) = -\log(\mathcal{A}_{i\tilde{a}}), \tag{9}$$

where $\tilde{a}$ is the correct assignment and $\mathcal{A}_{ij}$ is the target i to measurement j assignment probability, as described in Sec. 4.1.

4.6 Training Data

It is well known that deep architectures require vast amounts of training data to avoid overfitting the model. Huge labelled datasets like ImageNET [44] or Microsoft COCO [45] have enabled deep learning methods to unfold their potential on tasks like image classification or pixel labelling. Unfortunately, mainly due to the very tedious and time-consuming task of video annotation, only very limited amount of labelled data for pedestrian tracking is publicly available today. We therefore resort to synthetic data augmentation, which we use alongside real data. To that end, we propose three different methods that enable us to generate unlimited amounts of unique training samples: 1) by randomly perturbing real data; 2) by sampling from a simple generative trajectory model learned from real data; and 3) by generating physically motivated 3D-world projections.

Perturbation of real data. As the first concept, we adapt previously known techniques (*cf*. *e.g*. [9]) for data augmentation. Here, the data is altered in the following ways:

- Mirroring. Annotated trajectories are flipped horizontally and in time with probability 0.5. This corresponds to mirroring the image and running the video backwards, respectively. Note that we refrain from vertical flipping as it may lead to unrealistic data.
- Translation. The data is translated randomly by $[0 - 20]\%$ of the image size in both x and y dimensions.
- Rotation. The trajectory are rotated randomly by $[-20, 20]$ degrees around the image centre. Both translation and rotation simulate small pan and roll movements of the sensor.

Any trajectory parts that appear outside the image after the transformation is applied are discarded.

A generative model. The second way to obtain training data is to sample from a generative model. To that end, we first learn a trajectory model from each training sequence. For simplicity, we only estimate the mean and the variance of two features: the start location x_1 and the average velocity $\bar{v}$ from all annotated trajectories in that sequence. For each training sample we then generate up to N tracks by sampling from a normal distribution with the learned parameters.

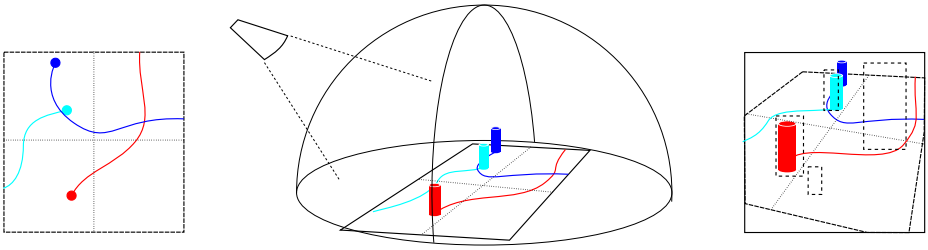

Fig. 4. An overview of our physically-based framework for synthetic data generation. Tracks are generated based on physical priors in a 3D scene and then projected to a synthetic view, where a realistic set of detections (dashed boxes) is obtained.

Physically-based trajectory generation. In the third method, the data is generated by simulating real-world motion and cameras in the following way. All tracks are first generated on a virtual ground plane, as seen from the bird's eye view (*cf*. Fig. 4). Human motion patterns typically do not involve complex manoeuvring so that reasonable motion can be synthesised by combining simple parametric dynamic models such as constant velocity, constant turn and Brownian motion. For this experiment, we generate the motion of each person recursively over time using the constant velocity dynamics with acceleration noise model. The height of each person is randomly sampled from a normal distribution with $\mu = 170\ cm$ and $\sigma = 20\ cm$.

To generate realistic sequences, we use a perspective camera model with random radial location and rotational angles with respect to polar coordinate in 3D world. The camera parameters and rotation and translation matrices are estimated such that the camera is facing toward scene and the centres of the scene and the image plane are coincident whilst covering the entire tracking area. The trajectories of the targets leaving the scene are terminated and we allow new targets to be added into the scene during the sequence.

For each target, we generate a noisy detection according to a random probability of detection. To simulate the performance of a detector, locations and heights of targets are perturbed with multivariate Gaussian noise whose covariance is adjusted to be proportional to the target's height in the image. In addition, a target is deemed missed if more than fifty percent of its bounding box is occluded by other targets that are closer to the camera. Clutter is also required to be added into the detection list. To this end, a number of false detections per frame are generated according to a Poisson distribution with a predefined mean. Then their locations and heights are sampled from a uniform distribution in the image domain.

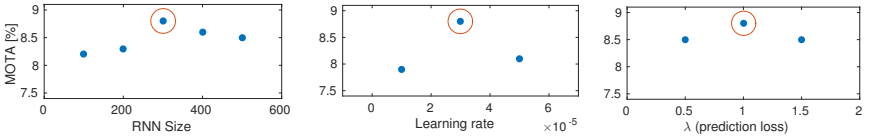

Fig. 5. Influence of three exemplar hyper-parameters on the overall performance on the MOTChallenge benchmark, measured by MOTA. The optimal parameter is marked with a red circle. Note that this graph shows the performance of our prediction/update RNN block for only one target ($N = 1$), which explains the relatively low MOTA.

5 Implementation Details

We implemented our framework in Lua and Torch7. Both our entire code base as well as pre-trained models are publicly available.[2]

Finding correct hyper-parameters for deep architectures still remains a non-trivial task [46]. We follow some of the best practices found in the literature [34, 46], such as setting the initial weights for the forget gates higher (1 in our case), and also employ a standard grid search to find a best setting for the present task (see Fig. 5 for three examples). In this section we will point out some of the most important parameters and implementation details. For a more in-depth coverage, we refer the reader to the supplemental material and to the source code.

Network size. The RNN for state estimation and track management is trained with one layer and 300 hidden units. The data association is a more complex task, requiring more representation power. To that end, the LSTM module employed to learn the data association consists of two layers and 500 hidden units.

Optimisation. We use the RMSprop [47] to minimise the loss. The learning rate is set initially to 0.0003 and is decreased by 5% every $20,000$ iterations. We set the maximum number of iterations to $200,000$, which is enough to reach convergence. The training of both modules takes approximately 30 hours on a CPU. With a more accurate implementation and the use of GPUs we believe that training can be sped up significantly.

Data. The RNN is trained with about $100,000$ 20-frame long sequences. The data is divided into mini-batches of 10 samples per batch and normalised to the range $[-0.5, 0.5]$, w.r.t. the image dimensions. We experimented with the more popular zero-mean and unit-variance data normalisation but found that the fixed one based on the image size yields superior performance.

6 Experiments

To demonstrate the functionality of our approach, we first perform experiments on simulated data. Fig. 6 shows an example of the tracking results on synthetic

[2] `https://bitbucket.org/amilan/rnntracking`

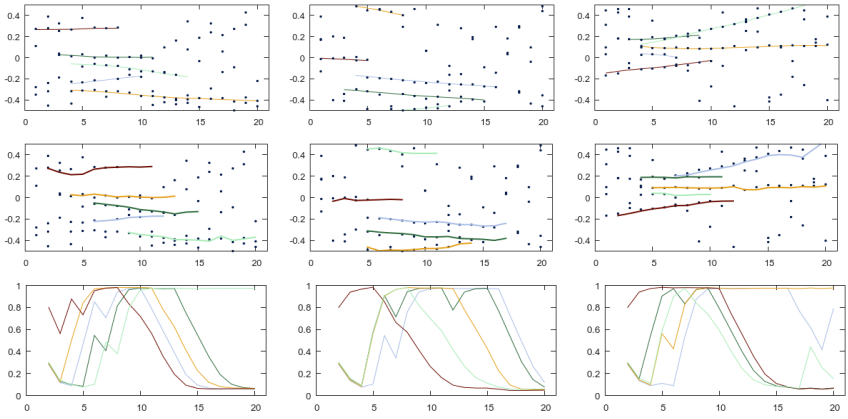

Fig. 6. Results of our tracking method on three 20-frame long synthetic sequences in cluttered environments. Top: Ground truth (x-coordinate vs. frame number). Middle: Our reconstructed trajectories. Bottom: The existence probability $\mathcal{E}$ for each target. Note the delayed initiation and termination, *e.g.* for the top-most track (beige) in the second example. This an inherent limitation of any purely online approach that cannot be avoided.

data. Here, five targets with random birth and death times are generated in a rather cluttered environment. The initiation / termination indicators are illustrated in the bottom row.

We further test our approach on real-world data, using the MOTChallenge benchmark [7]. This pedestrian tracking dataset is a collection of 22 video sequences (11/11 for training and testing, respectively), with a relatively high variation in target motion, camera motion, viewing angle and person density. The evaluation is performed on a server using unpublished ground truth.

Baseline comparison. We first compare the proposed end-to-end learning approach to three baselines. The results on the training set are reported in Tab. 1. The first baseline (Kalman-HA) employs a combination of a Kalman filter with bipartite matching solved via the Hungarian algorithm. Tracks are initiated at each unassigned measurement and terminated as soon as a measurement is missed. This baseline is the only one that fully fulfils the online state estimation without any heuristics, time delay or post-processing. The second baseline (Kalman-HA2) uses the same tracking and data association approach, but employs a set of heuristics to remove false tracks in an additional post-processing step. Finally, JPDA$_m$ is the full joint probabilistic data association approach, recently proposed in [13], including post-processing. We show the results of two variants of our method. One with learned motion model and Hungarian data association, and one with full end-to-end learning for all components, using RNNs and LSTMs. Our learned model performs favourably compared to the purely on-

Table 1. Tracking results on the MOTChallenge training dataset. *Denotes offline post-processing.

Method	Rcll	Prcn	MT	ML	FP	FN	IDs	FM	MOTA	MOTP
Kalman-HA	28.5	79.0	32	334	3,031	28,520	685	837	19.2	69.9
Kalman-HA2*	28.3	83.4	39	354	2,245	28,626	105	342	22.4	69.4
JPDA$_m$*	30.6	81.7	38	348	2,728	27,707	109	380	23.5	69.0
RNN_HA	37.8	75.2	50	267	4,984	24,832	518	963	24.0	68.7
RNN_LSTM	37.1	73.5	50	260	5,327	25,094	572	983	22.3	69.0

Table 2. Tracking results on the MOTChallenge test dataset. *Denotes an offline (or delayed) method.

Method	MOTA	MOTP	FAR	MT%	ML%	FP	FN	IDs	Frag.	FPS
MDP [48]	30.3%	71.3%	1.7	13.0	38.4	9,717	32,422	680	1,500	1.1
JPDA$_m$* [13]	23.8%	68.2%	1.1	5.0	58.1	6,373	40,084	365	869	32.6
TC_ODAL [49]	15.1%	70.5%	2.2	3.2	55.8	12,970	38,538	637	1,716	1.7
RNN_LSTM	19.0%	71.0%	2.0	5.5	45.6	11,578	36,706	1,490	2,081	165.2

line solution (Kalman-HA) and is even able to keep up with similar approaches but without any heuristic or delay in the output.

Benchmark results. Next, we show our results on the benchmark test set in Tab. 2 next to three *online* methods. The current leaderboard lists over 50 different trackers, with the top ones reaching almost 50% MOTA. Even though the evaluation is performed by the benchmark organisers, there are still considerable differences between various submissions, that are worthy pointing out. First, all top-ranked submissions use their own set of detections and not those provided in the dataset. While a better detector typically improves the tracking result, the direct comparison of the tracking method becomes rather meaningless. Therefore, we prefer to use the provided detections to guarantee a fair setting. Second, most methods perform so-called *offline* tracking, *i.e.* the solution is inferred either using the entire video sequence, or by peeking a few frames into the future, thus returning the tracking solution with a certain time delay. This is in contrast to our method, which aims to strictly compute and fix the solution with each incoming frame, before moving to the next one. Finally, it is important to note that many current methods use target appearance or other image features like optic flow [6] to improve the data association. Our method does not utilise any visual features and solely relies on geometric locations provided by the detector. We acknowledge the usefulness of such features for pedestrian tracking, but these are often not available in other application, such as *e.g.* cell or animal tracking. We therefore retain from including them at this point and leave it for future work.

Overall, our approach does not quite reach the accuracy of the state of the art in online tracking [48], but is two orders of magnitude faster. Fig. 7 shows some example frames from the test set.

Fig. 7. Our RNN tracking results on selected MOTChallenge sequences including ADL-Rundle-3 (first row), TUD-Crossing (second row) and PETS S2.L2 (bottom).

7 Discussion and Future Work

We presented an approach to address the challenging problem of data association and trajectory estimation within a neural network setting. To the best of our knowledge, this is the first approach that employs end-to-end training for multi-target tracking. We showed that an RNN-based network can be utilised to learn complex motion models for multiple targets jointly. The second, somewhat surprising finding is that an LSTM network is able to learn a globally optimal assignment, which is a non-trivial task for such an architecture. We firmly believe, that by incorporating appearance and by learning a more robust association strategy, the results can be improved significantly.

References

1. Reid, D.B.: An algorithm for tracking multiple targets. IEEE Transactions on Automatic Control **24**(6) (December 1979) 843–854
2. Fortmann, T.E., Bar-Shalom, Y., Scheffe, M.: Multi-target tracking using joint probabilistic data association. In: 19th IEEE Conference on Decision and Control including the Symposium on Adaptive Processes. Volume 19. (December 1980) 807–812
3. Jiang, H., Fels, S., Little, J.J.: A linear programming approach for multiple object tracking. In: CVPR 2007
4. Zhang, L., Li, Y., Nevatia, R.: Global data association for multi-object tracking using network flows. In: CVPR 2008
5. Andriyenko, A., Schindler, K., Roth, S.: Discrete-continuous optimization for multi-target tracking. In: CVPR 2012. 1926–1933
6. Choi, W.: Near-online multi-target tracking with aggregated local flow descriptor. In: ICCV 2015

7. Leal-Taixé, L., Milan, A., Reid, I., Roth, S., Schindler, K.: MOTChallenge 2015: Towards a benchmark for multi-target tracking. arXiv:1504.01942 [cs] (April 2015) arXiv: 1504.01942.
8. Stauffer, C., Grimson, W.E.L.: Adaptive background mixture models for real-time tracking. In: CVPR 1999. (1999) 2246–2252
9. Dalal, N., Triggs, B.: Histograms of oriented gradients for human detection. In: CVPR 2005. 886–893
10. Dollár, P., Appel, R., Belongie, S., Perona, P.: Fast feature pyramids for object detection. IEEE T. Pattern Anal. Mach. Intell. **36**(8) (2014) 1532–1545
11. Ondruska, P., Posner, I.: Deep tracking: Seeing beyond seeing using recurrent neural networks. In: The Thirtieth AAAI Conference on Artificial Intelligence (AAAI), Phoenix, Arizona USA (February 2016)
12. Kim, C., Li, F., Ciptadi, A., Rehg, J.M.: Multiple hypothesis tracking revisited: Blending in modern appearance model. In: ICCV 2015
13. Rezatofighi, H.S., Milan, A., Zhang, Z., Shi, Q., Dick, A., Reid, I.: Joint probabilistic data association revisited. In: ICCV 2015
14. Berclaz, J., Fleuret, F., Türetken, E., Fua, P.: Multiple object tracking using k-shortest paths optimization. IEEE T. Pattern Anal. Mach. Intell. **33**(9) (September 2011) 1806–1819
15. Ben Shitrit, H., Berclaz, J., Fleuret, F., Fua, P.: Tracking multiple people under global appearance constraints. In: ICCV 2011
16. Pirsiavash, H., Ramanan, D., Fowlkes, C.C.: Globally-optimal greedy algorithms for tracking a variable number of objects. In: CVPR 2011
17. Henriques, J.a., Caseiro, R., Batista, J.: Globally optimal solution to multi-object tracking with merged measurements. In: ICCV 2011
18. Andriyenko, A., Schindler, K.: Globally optimal multi-target tracking on a hexagonal lattice. In: ECCV 2010. Volume 1. 466–479
19. Butt, A.A., Collins, R.T.: Multi-target tracking by Lagrangian relaxation to min-cost network flow. In: CVPR 2013
20. Leibe, B., Schindler, K., Van Gool, L.: Coupled detection and trajectory estimation for multi-object tracking. In: ICCV 2007
21. Milan, A., Roth, S., Schindler, K.: Continuous energy minimization for multitarget tracking. IEEE T. Pattern Anal. Mach. Intell. **36**(1) (2014) 58–72
22. Milan, A., Schindler, K., Roth, S.: Detection- and trajectory-level exclusion in multiple object tracking. In: CVPR 2013
23. Brendel, W., Amer, M.R., Todorovic, S.: Multiobject tracking as maximum weight independent set. In: CVPR 2011
24. Chen, S., Fern, A., Todorovic, S.: Multi-object tracking via constrained sequential labeling. In: CVPR 2014
25. Ivakhnenko, A.G., Lapa, Valentin, G.: Cybernetic predicting devices. (1966) 250
26. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11) (November 1998) 2278–2324
27. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: NIPS*2012, Curran Associates, Inc. 1097–1105
28. Wang, T., Wu, D., Coates, A., Ng, A.: End-to-end text recognition with convolutional neural networks. In: 2012 21st International Conference on Pattern Recognition (ICPR). (November 2012) 3304–3308
29. Eigen, D., Fergus, R.: Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. arXiv

30. Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., Fei-Fei, L.: Large-scale video classification with convolutional neural networks. In: CVPR 2014
31. Goller, C., Küchler, A.: Learning task-dependent distributed representations by backpropagation through structure. In: ICNN, IEEE (1996) 347352
32. Vinyals, O., Toshev, A., Bengio, S., Erhan, D.: Show and tell: A neural image caption generator. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR). (June 2015)
33. Karpathy, A., Li, F.F.: Deep visual-semantic alignments for generating image descriptions. In: CVPR 2015. Volume abs/1412.2306.
34. Karpathy, A., Johnson, J., Li, F.F.: Visualizing and understanding recurrent networks. arXiv:1506.02078 [cs] (June 2015) arXiv: 1506.02078.
35. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: nips-2014. (2014) 3104–3112
36. Graves, A.: Generating sequences with recurrent neural networks. arXiv:1308.0850 [cs] (August 2013) arXiv: 1308.0850.
37. Graves, A., Mohamed, A.r., Hinton, G.E.: Speech recognition with deep recurrent neural networks. In: IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013. (2013) 6645–6649
38. Stewart, R., Andriluka, M.: End-to-end people detection in crowded scenes. arXiv:1506.04878 [cs] (June 2015) arXiv: 1506.04878.
39. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8) (November 1997) 17351780
40. Cho, K., van Merrienboer, B., Bahdanau, D., Bengio, Y.: On the properties of neural machine translation: Encoder-decoder approaches. arXiv:1409.1259 [cs, stat] (September 2014) arXiv: 1409.1259.
41. Kalman, R.E.: A new approach to linear filtering and prediction problems. Transactions of the ASME–Journal of Basic Engineering **82**(Series D) (1960) 35–45
42. Doucet, A., Godsill, S., Andrieu, C.: On sequential monte carlo sampling methods for bayesian filtering. Statistics and Computing **10**(3) (2000) 197–208
43. Bar-Shalom, Y., Fortmann, T.E.: Tracking and Data Association. Academic Press (1988)
44. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ImageNet Large Scale Visual Recognition Challenge. (2014)
45. Lin, T.Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C.L., Dollár, P.: Microsoft coco: Common objects in context. arXiv:1405.0312 [cs] (May 2014) arXiv: 1405.0312.
46. Greff, K., Srivastava, R.K., Koutnk, J., Steunebrink, B.R., Schmidhuber, J.: Lstm: A search space odyssey. arXiv:1503.04069 [cs] (March 2015) arXiv: 1503.04069.
47. Tieleman, T., Hinton, G.: Rmsprop: Divide the gradient by a running average of its recent magnitude. Coursera: Neural networks for machine learning (2012)
48. Xiang, Y., Alahi, A., Savarese, S.: Learning to track: Online multi-object tracking by decision making. In: International Conference on Computer Vision (ICCV). (2015) 4705–4713
49. Bae, S.H., Yoon, K.J.: Robust online multi-object tracking based on tracklet confidence and online discriminative appearance learning. In: CVPR 2014