Deep Learning Based Motion Models for Single Object Tracking

Momir Adžemović
University of Belgrade, Faculty of Mathematics
Belgrade, Serbia
momir.adzemovic@gmail.com

Predrag Tadić
University of Belgrade, School of Electrical Engineering
Belgrade, Serbia
ptadic@etf.rs, ORCID 0000-0002-8845-997X

Mladen Nikolić
University of Belgrade, Faculty of Mathematics
Belgrade, Serbia
mladen.nikolic@matf.bg.ac.rs

Andrija Petrovic
University of Belgrade, Faculty of Organizational sciences
Belgrade, Serbia
andrija.petrovic@fon.bg.ac.rs

***Abstract*—Tracking-by-detection algorithms in visual tracking applied to videos use Kalman filters (KFs) for trajectory estimation in order to localize tracked objects. While the KF is a strong baseline, it cannot model non-linear motion or adapt to specific domains. In this work, we empirically show that more accurate predictions of an object's position at the next time step can be obtained by using a Neural Ordinary Differential Equation (NODE) model. On the MOT20 benchmark dataset, NODE outperforms a linear model used by the KF, and a non-linear model learned by a recurant neural network, by 7% and 4% as measured by tracking accuracy, respectivaly.**

I. Introduction

The goal of object tracking is to estimate an object's track (or trajectory) over a series of frames. Tracking consists of two sub-tasks: object detection and object re-identification. In the case of tracking objects in *videos*, which is the focus of our work, the detection step consists of placing precise bounding boxes around all objects of interest in each video frame. For object re-identification, it is required to keep track of the objects' identity over time. Object tracking is one of the key components in many applications like autonomous driving, robotics and theft detection.

Tracking can be separated into two similar and fairly connected tasks: single- object tracking (SOT) and multi-object tracking (MOT). In the case of SOT, one target is specified in the first frame, and it is required to track that target in the following frames of the video. Ideally, SOT tracker should be able to track any arbitrary object that does not need to be in any specific class. Multi-object tracking additionally requires *association* (re-identification) between tracked objects from previous frame and new detections in the current frame [1].

In this work we propose a new trajectory prediction model based on ODE-RNN [2] architecture (NODE variant [3]). The model is compared to standard trajectory (motion) estimation models: Kalman Filter (KF) [4] and Recurrent Neural Network (RNN) [5]. The results demonstrate that ODE-RNN outperforms KF and RNN in tracking accuracy.

II. Related work

Tracking-by-detection is currently one of the most effective paradigms in both SOT and MOT. Many of the recent works in this field are based on SORT [6], DeepSort [7] and JDE [8]. Tracking-by-detection assumes that there already exists a very accurate detection algorithm. Algorithms based on SORT usually use KF [4] with the constant-velocity model assumption for object bounding box position estimation. KF estimates the joint distribution of an object's bounding box coordinates in the current frame, conditioned on all information observed thus far. The predicted bounding box position is used for association with new detections given by the detection model. The main drawback of using a KF is its inability to model non-linear motion which occurs when the camera is not static, or even when objects have non-linear movement.

Non-linear variations of KF already exist, such as extended Kalman Filter (EKF) [9] and the unscented Kalman Filter (UKF) [10]. One alternative for non-linear, non-Gaussian state modeling is the family of particle filters [11], which are based on Monte-Carlo sampling. These methods work well for the non-linear motion modeling but they can't directly learn from the data like the deep learning based motion models.

Another alternative approach to model an object's motion is to learn from past data. Recurrent Neural Networks (RNN) [5] have been successfully used for time-series related tasks and are capable of non-linear motion modeling. RNNs allow cycle connections between neurons, thus enabling it to store information

from data over time intervals of arbitrary lengths. The most popular architectures are LSTM (Long-Short Term Memory) [12] and GRU (Gated Recurrent Unit) [13] which partly resolve some of the main issues with RNN like vanishing gradients during training. One example of such successful work is TrajE [14], which improve the performance of existing state-of-the-art trackers by combining them with recurrent mixture density networks. NODE is a new architecture that already surpasses RNNs in model performance in many time-series tasks [3, 2]. In this paper we show that it can also surpass RNNs as motion models.

III. Neural ordinary differential equations

Neural Ordinary Differential Equations (NODE) [3] are a family of deep neural network models that can be understood as a continuous version of Residual Networks (ResNets) [15]. ResNet transforms hidden state from discrete time point t to $t+1$:

$$h_{t+1} = h_t + f_t(h_t) \tag{1}$$

where $h_t \in R^n$ is hidden state after layer $t-1$ and f_t is a residual layer which can be any neural network architecture. If we subtract h_t from both sides we get:

$$\frac{h_{t+\Delta t} - h_t}{\Delta t} = f_t(h_t) \tag{2}$$

where $\Delta t = 1$. In the limit $\Delta t \to 0$ we get the derivative $h'(t) = \frac{dh(t)}{dt}$ at the point t:

$$\frac{dh(t)}{dt} = f_t(h_t) = f(h(t), t) \tag{3}$$

From (3) we can see that the hidden state can be modeled using an ODE with the initial condition $h(0) = x$ where x is the input data (e.g. image). Model output is $h(T)$ which can be computed using any black-box differential equation solver (e.g. Euler method).

Some of the key benefits of using Neural ODEs are memory efficiency, adaptive computation and ability to naturally work with irregularly-sampled time series. These models use numerical ODE solvers without back-propagating through operations inside the solver. Instead, gradients can be computed using the *adjoint sensitivity method* [16] and there is no need to store intermediate values during the forward pass through solver operations. Resulting effect is constant memory cost as function of depth while training a neural network.

Time required for ODE solvers to perform numerical approximation can be reduced by lowering solver accuracy (i.e. increasing error tolerance). NODE speed can be increased during inference after the model is trained.

NODE outperforms RNN when used for extrapolation and interpolation for irregularly-sampled time series or time series with high amount of missing data at arbitrary time points. This is one of the main motivations for using NODE for trajectory estimation in tracking applications where bounding box detections can be missing in case of occlusions or any detector errors.

IV. Trajectory Estimation

A trajectory prediction model inputs a sequence of bounding box detections from times $t-s, \ldots, t-1, t$, where t is the time of the last observed frame, and outputs one or more future trajectory points, starting from time $t+1$. One bounding box detection is defined by 4 values $[x, y, w, h]$ where x and y are coordinates of the top-left corner of the bounding box, w is the box width, and h is the box height. Bounding box coordinates at time t are $[x, y, w, h]_t = [x_t, y_t, w_t, h_t]$. For more readable notation we are assuming that we deal with only one object.

A. Data preprocessing

In our experiments preprocessing is used for all trained neural network models (NODE and RNN). It is not used for the Kalman filter. These transformations are cheap and do not add any relevant overhead to computation during inference. Inverse transformations are applied to the model outputs, as shown in Fig. 1.

First order differences are computed for the trajectory coordinates of each object followed by standardization transformation. First order difference is a simple transformation $\Delta X_t = X_{t+1} - X_t$ where X_t is bounding box at time t with values $[x, y, w, h]_t$. Values of $\Delta X_t = [\Delta x, \Delta y, \Delta w, \Delta h]_t$ can be related to a very rough approximations of velocity at time point t. This transformation makes learning easier for the model since it is easier to learn trajectory dynamics (e.g. which way the object is moving) from differences than from raw coordinates.

All bounding box values are expressed in terms relative to the image size in the interval $[0, 1]$. After applying first order difference transformation on the trajectory, all values become close to zero. This makes model learning slower unless network weights initialization is set to appropriately small values. Thus, all differences are standardized to have mean equal to 0 and variance equal to 1.

B. ODE-RNN

The authors of ODE-RNN introduced a hybrid model of NODE and RNN. Main advantage of ODE-RNN is its ability to use additional input features for which dynamics are not modeled with NODE [2].

Architecture overview of the our custom ODE-RNN can be seen in Fig. 1. Transformed trajectory $[\Delta x, \Delta y, \Delta w, \Delta h]_t$ is mapped to a latent vector z_t using an ODE-RNN encoder. The latent vector z_t summarizes relevant input trajectory information into a vector of

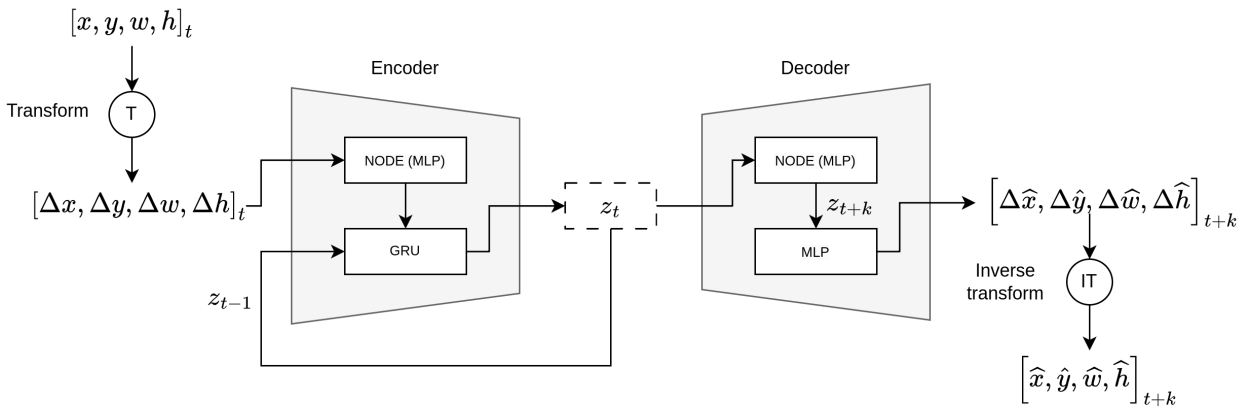

Fig. 1. Overview of our custom ODE-RNN architecture implementation for trajectory estimation

smaller dimension. Using the latent vector z_t as the initial ODE condition, a latent trajectory is extrapolated from z_{t+1} to z_{t+k} using a NODE decoder with an ODE solver. Lastly, extrapolated difference values are mapped using a Multi Layer Perceptron (MLP) [17] to the data space, which is then transformed to the final trajectory prediction. For the encoder ODE-RNN architecture, a combination of NODE with MLP and GRU is used. All MLPs consist of two layers. Each layer is made up of a linear mapping followed by layer normalization [18] and a leaky ReLU activation function [19]. We use Runge-Kutta fourth order method [20] ODE solver with fixed step size equal to 0.25 as the default setup for inference and training.

C. RNN

Architecture used for the RNN model is similar to previously introduced custom ODE-RNN architecture, but without NODE layer for encoder and with NODE layer replaced by GRU in decoder.

V. Experiments

Experiments are performed on the MOTChallenge (MOT20) dataset (pedestrians) [21]. This dataset is mainly used as a multi-object tracking (MOT) benchmark. The public MOT20 dataset consists of 2215 pedestrian tracks over 4 scenes with static camera views. Three scenes: MOT20-01, MOT20-02 and MOT20-03 (1046 tracks in total) are used for training, while scene MOT20-05 (1169 tracks) is used for the evaluation of trained models.

A. Training

During training, each track (object bounding box trajectory) is split into windows of consecutive time points. For each object trajectory all windows are sampled using a sliding window with step 1. For an object trajectory of length n, a total of $n - w + 1$ windows are sampled, where w is window size. For each window trajectory, values from time $t-s$ to t (time of last observed values) are used to predict the trajectory from $t+1$ to $t+f$. We use $s = 9$ (last 10 observations) and $f = 1$ for model training, but we also tested shorter input trajectories. In our experiments we concluded that s can be lowered to 2 while keeping good model performance. For evaluation we use $f = 5$ and report results for all time points in between. The goal is to train a model which can learn dynamics from trajectory history and generalize well. If the motion model performs well for $f = 1$ but not for $f = 5$ then it does not generalize well enough.

All ODE-RNN models are trained for 10 epochs with Adam optimization algorithm [22]. Starting learning rate is 0.001 and it is multiplied by 0.1 every third epoch. Each batch consists of 32 randomly sampled trajectory window samples. ODE-RNN-S has 8-dimensional latent space, while ODE-RNN-L has 32-dimensional latent space. We used weight decay with value 0.01 for ODE-RNN-S and 0.005 for ODE-RNN-L.

B. Evaluation

For model evaluation and comparison, we use mean squared error (MSE) and tracking accuracy. Tracking accuracy is defined as the average IOU (intersection over union— measure of overlap between two bounding boxes) between the predicted bounding box and the ground truth bounding box. Models with lower MSE tend to have higher tracking accuracy. Model trajectory estimation is evaluated from time $t + 1$ to time $t + 5$ where t is the last observed time point at the time of inference. We use $\mathrm{MSE}[a..b]$ and $\mathrm{Acc}[a..b]$ notation for average metric scores from time point $t + a$ to $t + b$.

We compare ODE-RNN with Kalman filter (default linear model for trajectory estimation) and RNN (standard model for time-series extrapolation). For Kalman filter we use implementation and parameters from BoT-SORT [23] which is the state-of-the-art method that uses a modified version of the constant velocity model. For

TABLE I
COMPARISON BETWEEN TRAJECTORY ESTIMATION MODELS IN TERMS OF MSE AND ACCURACY.

Model	MSE[1..5] ↓	Acc[1..5] ↑
KF	7.17e-06	88.53%
RNN (GRU)	4.20e-06	91.78%
ODE-RNN-L	**1.22e-06**	**95.91%**
ODE-RNN-S	2.56e-06	93.37%

multi-step predictions, we employ the Kalman filter auto-regressivelly: predictions from previous steps are used as inputs for the next step. Prediction for one step ahead is $\hat{X}_{t+1} = A \cdot X_t$ where A is the constant velocity motion model and $X_t = [x, y, w, h]_t$ is the bounding box at time t. Predictions for the following steps are $\hat{X}_{t+k+1} = A \cdot \hat{X}_{t+k} = A^k \cdot X_t$. Update step is skipped since measurements are not yet observed. For ODE-RNN we just need to specify multiple moments at which we are interested in the bounding boxes. More precisely, during the training we use time sequence $[t+1]$ and during inference we use $[t+1, ...t+5]$. For RNN we recursively predict the sequence of coordinates by using output of the previous step $t+k$ as input to the current step $t+k+1$.

We trained two ODE-RNNs with the same configuration except for the number of parameters, defined by the latent vector dimension. ODE-RNN-L is the best estimator with 7.38% increase in accuracy compared to Kalman filter and 4.13% increase in accuracy compared to RNN. This model has $12.6K$ parameters while RNN has $47.3K$ parameters which makes ODE-RNN almost 4 times more memory efficient during inference. The smaller version ODE-RNN-S has only 1k parameters. It still manages to outperform RNN by 1.59% in accuracy with 47 times less parameters. Comparison between all models can be seen in table I and comparison between each model's accuracy at each step from $t+1$ to $t+5$ can be seen in Fig. 2.

The main drawback of using NODE is slower training and inference. On the other side, NODE have adaptive computation and speed can be increased during inference by decreasing ODE solver accuracy and thus decreasing model accuracy.

C. Importance of regularization

During experiments we observed that adding regularization is hugely beneficial. Without it ODE-RNN model fails to generalize for multi-step prediction when it is trained as a single-step prediction model. Even if the model is not overfitting on the single-step prediction

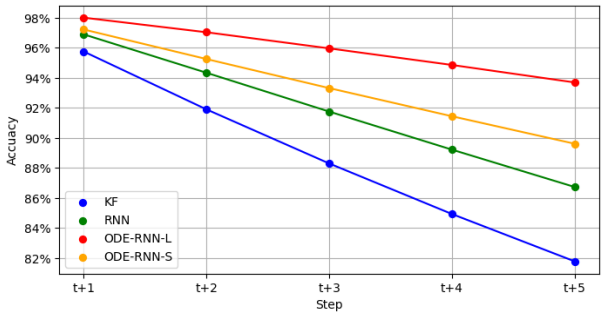

Fig. 2. Comparison between trajectory estimation models in terms of accuracy for first 5 steps.

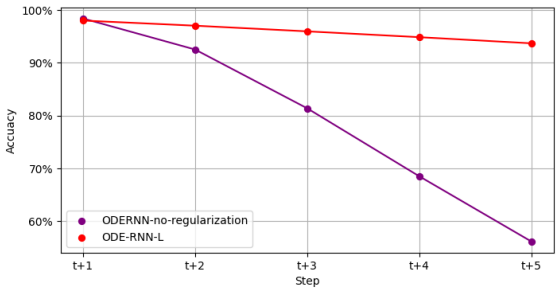

Fig. 3. Comparison between trajectory estimation models in terms of accuracy for first 5 steps for model with and without weight decay.

during training, the accuracy on multi-step prediction degrades drastically for each step forward, as seen from Fig. 3. This model performs worse than the KF and the RNN.

With further analysis we observed that trajectories in the latent space are diverging. Unregularized NODE tend to generate irregular trajectories. This issue has been already observed by other researchers [24, 25]. We resolved the problem of divergent latent space trajectories via weight decay regularization [26].

We also trained another ODE-RNN model to predict several future steps instead of a single step, without using weight decay and achieve accuracy 96.85% which outperforms ODE-RNN-L. However, these are not directly comparable, since in this experiment ODE-RNN is provided with five-step supervision.

VI. CONCLUSION

We have used a custom ODE-RNN architecture for trajectory estimation and gained a significant accuracy improvement over the Kalman filter and recurrent neural network models. This architecture drastically reduces the number of parameters compared to RNN, which makes it suitable for low memory machines. Still, switching from RNN to ODE-RNN may increase average time required for training and inference.

VII. Future work

Most tracking algorithms require an estimate of prediction uncertainty. This is especially true for MOT tasks, where we must make associations between predicted and detected bounding boxes for multiple simultaneous tracks. The cost of some detection-prediction pairing depends on the *statistical* distance between them, which is affected by the uncertainty. However, the main focus of this paper are SOT tasks, where we know in advance that there is only one object in the scene, and the association problem is less pronounced. Adding estimation uncertainty modeling to ODE-RNN would make it a valid alternative to the Kalman Filter in MOT.

Trajectory estimation models should also be able to learn specific motion patterns in case of scenes with reccuring behaviour (e.g. factory monitoring). This approach should be tested on such datasets.

References

[1] Z. Soleimanitaleb and M. A. Keyvanrad. "Single Object Tracking: A Survey of Methods, Datasets, and Evaluation Metrics". In: *CVPR*. Arxiv: 2201.13066. 2022.

[2] Y. Rubanova, R. T. Q. Chen, and D. Duvenaud. "Latent ODEs for Irregularly-Sampled Time Series". In: *NeurIPS*. Arxiv: 1907.03907. 2019.

[3] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud. "Neural Ordinary Differential Equations". In: *NeurIPS*. Arxiv: 1806.07366. 2018.

[4] R. G. Brown and P. Y. C. Hwang. *Introduction to Random Signals and Applied Kalman Filtering with MATLAB Exercises*. 2021.

[5] R. M. Schmidt. "Recurrent Neural Networks (RNNs): A gentle Introduction and Overview". In: *stat.ML*. Arxiv: 1912.05911. 2019.

[6] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft. "Single Object Tracking: A Survey of Methods, Datasets, and Evaluation Metrics". In: *CVPR*. Arxiv: 1602.00763. 2016.

[7] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft. "Single Object Tracking: A Survey of Methods, Datasets, and Evaluation Metrics". In: *CVPR*. Arxiv: 1602.00763. 2016.

[8] Z. Wang, L. Zheng, Y. Liu, Y. Li, and S. Wang. "Towards Real-Time Multi-Object Tracking". In: *CVPR*. Arxiv: 1909.12605. 2020.

[9] M. Gruber. "An approach to target tracking". In: *MIT Lexington Lincoln Lab*. 1967.

[10] S. J. Julier and J. K. Uhlmann. "New extension of the Kalman filter to nonlinear systems". In: *International Society for Optics and Photonics*. 1997.

[11] N. Gordon, D. Salmond, and A. Smith. "Novel approach to nonlinear/non-Gaussian Bayesian state estimation". In: *International Society for Optics and Photonics*. 1993.

[12] S. Hochreiter and J. Schmidhuber. "Long Short Term Memory". In: *Neural Computation*. 1997.

[13] K. C. abd Bart van Merrienboer, D. Bahdanau, and Y. Bengio. "On the Properties of Neural Machine Translation: Encoder-Decoder Approaches". In: Arxiv: 1409.1259. 2014.

[14] A. Girbau, X. Giró-i-Nieto, I. Rius, and F. Marqués. "Multiple Object Tracking with Mixture Density Networks for Trajectory Estimation". In: *CVPR*. 2021.

[15] K. He, X. Zhang, S. Ren, and J. Sun. "Deep Residual Learning for Image Recognition". In: *CVPR*. Arxiv: 1512.03385. 2016.

[16] L. S. Pontryagin, E. Mishchenko, V. Boltyanskii, and R. Gamkrelidze. *The mathematical theory of optimal processes*. 1962.

[17] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. 2009.

[18] J. L. Ba, J. R. Kiros, and G. E. Hinton. "Layer Normalization". In: *stat.ML*. Arxiv: 1607.06450. 2016.

[19] B. Xu, N. Wang, T. Chen, and M. Li. "Empirical Evaluation of Rectified Activations in Convolution Network". In: *stat.ML*. Arxiv: 1505.00853. 2015.

[20] C. Runge. *Über die numerische Auflösung von Differentialgleichungen*. 1895.

[21] P. Dendorfer, H. Rezatofighi, A. Milan, J. Shi, D. Cremers, I. Reid, S. Roth, K. Schindler, and L. Leal-Taixe. "MOT20: A benchmark for multi object tracking in crowded scenes". In: *CVPR*. Arxiv: 2003.09003. 2020.

[22] J. B. Diederik P. Kingma. "Adam: A Method for Stochastic Optimization". In: *3rd International Conference for Learning Representations*. 2014.

[23] N. Aharon, R. Orfaig, and B.-Z. Bobrovsky. "BoT-SORT: Robust Associations Multi-Pedestrian Tracking". In: *CVPR*. Arxiv: 2206.14651. 2022.

[24] C. Finlay, J.-H. Jacobsen, L. Nurbekyan, and A. M. Oberman. "How to train your neural ODE: the world of Jacobian and kinetic regularization". In: *stat.ML*. 2020.

[25] A. Ghosh, H. S. Behl, E. Dupont, P. H. S. Torr, and V. Namboodiri. "STEER: Simple Temporal Regularization For Neural ODEs". In: *stat.ML*. 2020.

[26] A. Ghosh, H. S. Behl, E. Dupont, P. H. S. Torr, and V. Namboodiri. "A Simple Weight Decay Can Improve Generalization". In: *NeurIPS*. 1991.