THESIS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

Deep Learning for Model-Based Multi-Object Tracking

An investigation on the use of deep learning methods for addressing shortcomings in current model-based multi-object trackers/smoothers.

Juliano T. A. L. Pinto

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden, 2023

Deep Learning for Model-Based Multi-Object Tracking

An investigation on the use of deep learning methods for addressing shortcomings in current model-based multi-object trackers/smoothers.

Juliano T. A. L. Pinto
ISBN 978-91-7905-924-8

Doktorsavhandlingar vid Chalmers tekniska högskola
Ny serie nr 5390
ISSN 0346-718X

Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg, Sweden
Phone: +46 (0)31 772 1000

Cover:
Photo by Neida Zarate on Unsplash, adapted by the author.

Printed by Chalmers Reproservice
Gothenburg, Sweden, September 2023

Deep Learning for Model-Based Multi-Object Tracking

An investigation on the use of deep learning methods for addressing shortcomings in current model-based multi-object trackers/smoothers.

Juliano T. A. L. Pinto
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Multi-object tracking (MOT) is the task of estimating the state of multiple objects based on noisy sensor measurements. MOT is essential in various applications, such as pedestrian monitoring, vehicle tracking, animal behavior analysis, and others. It can be broadly divided into two categories: model-free and model-based, depending on whether accurate and tractable models of the measurement sensor and objects' dynamics are available for methods to use.

In model-based MOT, closed-form, Bayes-optimal solutions can be derived for certain model families. These solutions achieve the best possible performance in expectation, but become intractable as the time-horizon increases due to an exponential growth in the number of terms. Approximations are necessary to make these methods feasible, but they result in performance degradation for challenging tracking tasks.

The main objective of this thesis is to use deep learning (DL) to address this limitation, pursued in papers A, C, and D. The approach taken is to treat MOT as a sequence-to-sequence learning task, devising methods that learn to map measurement sequences to state estimates directly. This perspective frees methods from the need to explicitly consider all possible associations between objects and measurements, thereby side-stepping the intractability of traditional approaches. Furthermore, the available models of the environment are leveraged to generate unlimited synthetic data. This is used to train modern DL architectures that excel in the regime of big data, unlocking their power to reason about complicated and long-term temporal interactions in their inputs.

When developing the aforementioned methods, it became necessary to compare their predictions and estimated uncertainties to the state-of-the-art trackers for the model-based setting. To allow for this, another contribution of this thesis is in paper B, which proposes the first uncertainty-aware, hyperparameter-free, mathematically principled performance measure for MOT.

Keywords: Deep learning, multi-object tracking, multi-object smoothing, multi-object tracking performance measures.

List of Publications

This thesis is based on the following publications:

[A] **Juliano Pinto**, Georg Hess, William Ljungberh, Yuxuan Xia, Lennart Svensson, Henk Wymeersch, "Next Generation Multitarget Trackers: Random Finite Set Methods vs Transformer-based Deep Learning". Proceedings of FUSION 2021.

[B] **Juliano Pinto**, Yuxuan Xia, Lennart Svensson, Henk Wymeersch, "An Uncertainty-Aware Performance Measure for Multi-Object Tracking". Published in IEEE Signal Processing Letters, 2021, vol. 28, no. 1689–1693.

[C] **Juliano Pinto**, Georg Hess, William Ljungberh, Yuxuan Xia, Henk Wymeersch, Lennart Svensson, "Deep Learning for Model-Based Multi-Object Tracking". Accepted for publication in IEEE Transactions on Aerospace and Electronic Systems, 2023.

[D] **Juliano Pinto**, Georg Hess, Yuxuan Xia, Henk Wymeersch, Lennart Svensson, "Transformer-Based Multi-Object Smoothing with Decoupled Data Association and Smoothing". In review.

Other publications by the author, not included in this thesis, are:

[E] Viktor Olsson, Wilhelm Tranheden, **Juliano Pinto**, Lennart Svensson, "ClassMix: Segmentation-Based Data Augmentation for Semi-Supervised Learning". Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, 2021.

[F] Wilhelm Tranheden, Viktor Olsson, **Juliano Pinto**, Lennart Svensson, "DACS: Domain Adaptation via Cross-domain Mixed Sampling". Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, 2021.

Acknowledgments

I am very grateful to my supervisor, Lennart, for giving me the opportunity to pursue this PhD. Thank you for believing in my potential, ever since the beginning, sometimes even more than I did. Thank you for being more than just a supervisor. Thank you Henk, my co-supervisor, for having been such a good counterpoint and for caring so much (and being amazing at reviewing our papers). Thank you both for this, and for a hundred other things.

A big thanks to all my colleagues, in special to Yuxuan, Georg, and William, for always being so helpful and kind with me, for providing such useful and stimulating discussions, and for the great collaborations. Thanks also to my former office mates Erik and Anders. You have been amazingly friendly and receptive, and created an awesome office environment. Thanks to the Modulai team, in special Josef and Emil, for the flexibility and support you provided me to help finish this thesis.

I would also like to express my gratitude to my friends. Thank you, Claudia, for all the lunches where we would vent together about the hardships of pursuing a PhD, and for all the good advice. Thank you, Olof, for the great advice on how to juggle writing my thesis with working. Thank you, Filipe and Teddy, for your patience and kindness in listening to me talk about the thousand challenges I was dealing with. Thank you, Tob, Johan, Elin, and Emeric, for all the fun and distractions that I most definitely needed to maintain my sanity. Thank you to all my friends, close and distant, new and old. Your friendship brought balance to my PhD journey.

Thanks to my family. Thanks, mom, for believing in me when others did not, and reminding me that you have to fight for what you want in life. Thanks, grandma, for all the love and support, and the lessons in patience and consistency. Thanks, dad, for reminding me of the need for balance. Thanks tio Beto and tia Sandra for the amazing support you provided to my education. Thanks to all of my family for the continued support and love, even from ten thousand kilometers away.

I am also immensely grateful to my wife. Thank you, love, for being so strong and patient and kind. Thank you for all the happiness and love you bring into my life. Thank you for being so supportive and encouraging in this amazing PhD journey I had... and thank you for being there and help pick up the pieces when it was not so amazing. This thesis would not be what it is without you.

Lastly, thanks to my daughter. Sina, you are too young to read at this moment, but old enough to have taught me some of the biggest lessons in my life. Thank you for showing me I am much stronger than I thought, and for the constant reminders of what is really important in life.

Acronyms

MOT multi-object tracking

DL deep learning

SL supervised learning

PSL probabilistic supervised learning

KLD Kullback-Leibler divergence

SGD stochastic gradient descent

SSM state-space model

RFS random finite sets

MB multi-Bernoulli

MBM multi-Bernoulli mixture

PMBM Poisson multi-Bernoulli mixture

Contents

Part I

Overview

CHAPTER 1

Introduction

Multi-object tracking (MOT) is the problem of recursively estimating the state of an unknown number of objects, based on a sequence of noisy sensor measurements. The number of objects might vary over time, as objects can enter and leave the field of view of the sensor. Furthermore, the sensors are not perfect: there might be missing measurements, and the obtained measurements might contain false entries due to, e.g., sensor noise or clutter.

This task is illustrated in Fig. 1.1, which shows an example of tracking for three objects with 1-dimensional states. On the left, the ground-truth trajectories for each of the three objects are shown, color-coded to distinguish objects. The middle plot shows a depiction of the measurements generated at each time step. Note how each measurement does not contain information about its originating object (nor if it is a false measurement), requiring trackers to reason probabilistically about all possible associations. Lastly, the plot on the right shows a visualization of a possible estimate for the trajectories of such objects, where dark lines represent the mean for that state, and the shaded regions the corresponding 95% confidence intervals.

Methods capable of solving this problem are of high importance to a diverse set of applications. For example, MOT can be used for monitoring the

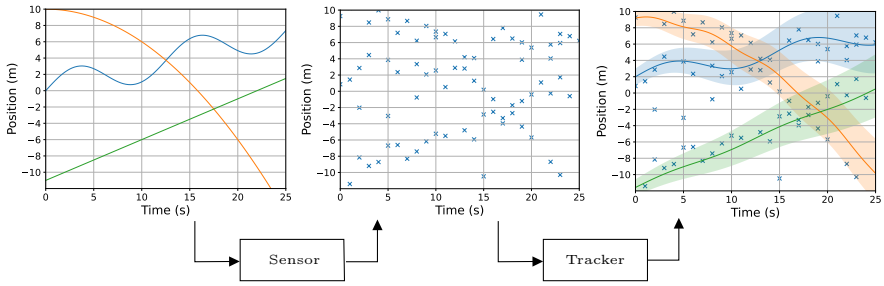


Figure 1.1: Common tracking pipeline. On the left, the trajectories for three objects are plotted over time. At each time step, a measurement sensor produces a set of measurements, generating the plot in the middle of the figure. The task of a multi-object tracker is to process this information into estimates of the states/trajectories of each of the object.

activities of pedestrians in an urban setting [1], tracking the positions and orientations of cars and cyclists on a road [2], [3], studying the interactions and dynamics in groups of animals [4], [5], predicting missile trajectories in military applications [6], analyzing the strategies and performance of players in team sports [7], and much more.

MOT problems can be broadly divided into two categories: model-free and model-based. The major topic of this thesis is model-based MOT, which deals with situations in which accurate and tractable models of the environment are available for methods to use. Such models encode prior knowledge about how objects move, interact, appear, and disappear from the field of view. They also encode information about the sensor being used to generate measurements. Model-free MOT, on the other hand, is concerned with settings where such models are not available.

For example, predicting missile trajectories using radar can be tackled using model-based MOT, as accurate models for the missile's dynamics and the measurement sensor are likely available. On the other hand, tasks such as pedestrian tracking using video cameras are more approachable from the model-free MOT perspective, since obtaining accurate and tractable models of how pedestrians move and interact, and how images are formed in a video camera is a challenging endeavor.

In the model-based MOT setting, particularly for certain families of models, it is possible to derive closed-form, Bayes-optimal solutions to the MOT problem [8]–[11]. Such solutions achieve the lowest possible expected error for a given task and are therefore impossible to outperform in theory. However, the situation is different in practice. Due to the unknown correspondence between measurements and objects, the number of terms in these solutions grows super-exponentially over the time horizon being considered, making such approaches intractable in all but simple tracking tasks [12]. To be applicable in practice, such methods require approximations such as pruning unlikely hypotheses or merging similar ones, which inevitably lead to performance deterioration in challenging tracking tasks.

The main objective of this thesis is to use deep learning (DL) to address this obstacle. We do so by casting MOT as a sequence-to-sequence learning task, where methods learn to directly map sequences of measurements to state estimates, thereby sidestepping the need to explicitly reason about all possible data associations[1]. Furthermore, papers A, C, and D share an insight on how to achieve this: to utilize the available models to generate synthetic training data. This approach, illustrated in Fig. 1.2, allows for generating virtually unlimited amounts of data, allowing for modern DL architectures that are very high-performing but also notoriously data-hungry, such as the Transformer [13] architecture.

This approach not only sidesteps the need to explicitly reason about all possible data associations, but also allows for the use of a much broader family of MOT models, which would be challenging or impossible to explicitly derive closed-form solutions for using the random finite sets (RFS) formalism. In particular, we can use nonlinear models directly, models that capture non-Markovian dynamics, models for non-independent object movement (where the state of one object affects the state of other objects), models with non-Gaussian noise, and many more: if we can sample from the model, we can use it to train a DL-based tracker.

[1] Paper D explores a slight variation of this approach, but still does not explicitly represent all possible data associations.

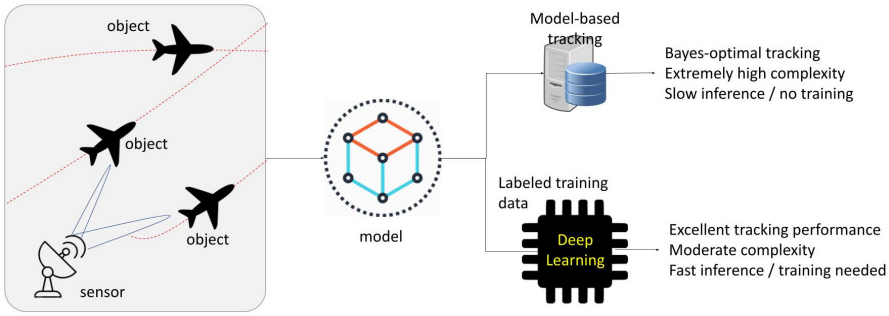


Figure 1.2: The models of the environment are traditionally used to derive Bayes-optimal trackers which suffer from extremely high complexity in challenging tracking scenarios. This thesis proposes to instead use the available models for generating synthetic training data for DL methods that tackle MOT as a sequence-to-sequence prediction task.

1.1 Contributions

In paper A we perform a preliminary analysis into the feasibility of training modern deep learning models for model-based MOT. We design a DL-based tracker (MT3) based on the Transformer encoder-decoder network, supervised using a combination of the loss proposed in [14] and a novel auxiliary loss designed specifically for MOT. This paper provides the first thorough comparison between a tracker using a modern deep learning architecture and state-of-the-art RFS-based trackers. We show that DL-based trackers can approximate the Bayes-optimal performance of the benchmarks in simple tracking tasks while outperforming them when the data association complexity increases.

Although paper A provides strong evidence of the applicability of DL to the model-based MOT setting, MT3 has an important limitation compared to RFS-based approaches: it does not provide uncertainty estimates for its state predictions. When investigating ways to address this, one challenge that had to be overcome was the lack of principled uncertainty-aware performance measures for MOT. Although uncertainty-aware performance measures do exist [15], [16], these do not account for all the uncertainty information available, or require access to ground-truth uncertainties. This motivated the writing of paper B, which introduces the first principled,

hyperparameter-free, uncertainty-aware performance measure for MOT. The proposed performance measure is based on the negative log-likelihood of the predicted MOT posterior, and is used in all subsequent papers in this thesis to compare the DL-based uncertainty predictions with the ones from RFS trackers.

Paper C then expands the preliminary analysis made in paper A. This paper proposes MT3v2, an improved version of MT3 with redesigned architecture for better performance, and to allow it to predict the entire kinematic state of objects (not just its observable part) together with uncertainty predictions. Similar to our preliminary analysis, we also provide a comparison to state-of-the-art trackers, but this time in much more depth. More complicated tasks with nonlinear measurement models are used, more performance measures are compared, and an error analysis of the trackers considered is provided.

Lastly, this thesis concludes with paper D, which expands the use of deep learning in model-based MOT to the smoothing task. Smoothing, in contrast to filtering (the task for papers A and C), takes as input a sequence of measurements from some time t in the past until now, and is tasked with predicting the full trajectory for all objects that appeared within that time window (including objects that disappeared before the end of it). This is a considerably more challenging task than filtering because both the number of trajectories and their length are not known beforehand. To deal with this, paper D proposes a new loss formulation to take this into account, and explores a decoupling of the data association subtask from the trajectory estimation.

1.2 Related Work

Deep learning has been widely applied to multi-object tracking, resulting in multiple breakthroughs to the state-of-the-art performance [17]–[19]. However, most of the field is focused on the *model-free* MOT setting, specifically in video-based tracking. Several studies have been made that use DL methods as aid for solving one or several of the MOT subtasks, such as object detection [20], [21], extracting high-level features from input data [22], [23], associating new measurements to existing tracks [24]–[26], managing track initialization/termination [27], predicting motion models [28], [29], and others. There has also been progress into tackling the entire tracking task using DL, using architectures based on extensions of object detectors [30],

convolutional neural networks [31], [32], and graph neural networks [33], [34], to cite a few.

Closer to our work, Transformer-based architectures for tracking such as [35]–[39] have lately become important, obtaining excellent results in popular video-based MOT benchmarks. These approaches rely on encoder-decoder architectures with the concept of track queries, vectors predicted by the decoder and trained to summarize an object's history. The queries are used by TrackFormer [35] and TransTrack [37] to select which information from the current measurements will be used to generate predictions for the current time step and to link trajectories over time. MOTR [36] extends this method by introducing a temporal aggregation network, allowing the decoder to access measurements from previous time steps. Alternatively, MeMOT [38] maintains an explicit memory bank of prior states to aid with long-term associations, and uses track queries only to summarize an object's history, but not to connect predictions through time (data association is addressed in a different module). SegDQ [39] includes semantic segmentation as an auxiliary task to help MOT performance, demonstrating the advantages of multi-task learning.

Despite the considerable success achieved by DL approaches, their direct applicability to the model-free setting is inherently limited due to several key factors. Firstly, there exists a substantial disparity in dimensionality between model-free and model-based MOT measurements. Model-free MOT often involves measurements comprising entire images, necessitating DL trackers to employ architectures with specific inductive biases tailored to this task, such as convolutional layers and max pooling. Conversely, model-based MOT typically deals with low-dimensional measurements, thus requiring significant architectural modifications, if not complete replacement, for the adaptation of a model-free DL tracker.

Secondly, the two settings have divergent performance requirements concerning temporal associations. In model-free MOT, processing measurement data from multiple time steps jointly is often too computationally expensive, and long-term temporal associations may not always be crucial. Vision-based MOT, for instance, commonly obtains sufficient information for accurately estimating object positions from just one or two frames. As a consequence, most existing model-free MOT trackers [35], [37]–[42] are unable to process more than a few frames per time step. On the other

hand, model-based MOT entails significantly higher uncertainties in data association: in most tasks in papers A, C, and D, even after considering all measurements, considerable uncertainty regarding the correct associations remains. Consequently, achieving accurate temporal associations becomes crucial for achieving good performance. Extending model-free methods to handle long-range temporal associations effectively thus becomes a significant and challenging research endeavor.

Lastly, many of the top-performing DL trackers in popular benchmarks such as MOT20 rely heavily on the image structure of the input measurements [40], [41] (e.g., learning appearance cues to handle occlusion properly). However, such image-based methodologies lack a straightforward counterpart in the model-based setting, rendering some of the key advancements in the model-free state-of-the-art inapplicable.

1.3 Thesis Outline

The remainder of part I of this thesis is organized as follows. Chapter 2 provides a review on machine learning, specifically probabilistic supervised learning. Chapter 3 reviews the most important points about Bayesian inference on state-space models, including the recursive solutions to Bayesian filtering and smoothing. The random finite set framework is then introduced in Chapter 4, with examples of important operations on RFSs and descriptions of relevant distributions. Chapter 5 provides a review of the assumptions, algorithms, and performance measures of model-based multi-object tracking, followed by chapter 6 with a summary of the included papers, and finally chapter 7 with concluding remarks. Part II then contains the included papers that comprise this thesis.

1.4 Notation

This section describes the notations used in Part I. For clarity, we use different typefaces for different mathematical objects, and make a distinction between random and deterministic variables, as shown in Table 1.1. Sequences are denoted with subscripts indicating their index sets, e.g., $x_{1:n}$ or $\mathbf{A}_{1:m}$, and the indexes can be used to identify individual elements (x_3 is the third element in $x_{1:10}$). The sets $\mathbb{R}^+$ and $\mathbb{R}_0^+$ denote respectively the set of all positive and

Table 1.1: Notation used in this thesis for different types of variables and whether or not they are random.

Type	Deterministic	Random
Scalar/Vector	x	$\mathbf{x}$
Matrix	X	$\mathbf{X}$
Set	$\mathbb{X}$	$\mathbb{X}$
Topological Space	$\mathscr{X}$	–

non-negative real numbers, and we use the shorthand $\mathbb{N}_a \doteq \{1, \cdots, a\}$. The distribution for a random variable $\mathbf{x}$ is denoted $f_{\mathbf{x}}(\cdot)$, or just $f_{\mathbf{x}}$, whereas the evaluation of such distribution at a number x is $f_{\mathbf{x}}(x)$. When possible we try to use the same letter for both the random variable and the value for which its distribution is being evaluated, e.g., $f_{\mathbf{x}}(x)$ and $f_{\mathbf{y}}(y)$ instead of $f_{\mathbf{x}}(y)$ and $f_{\mathbf{y}}(x)$.

Joint distributions are denoted using commas in the subscript, as $f_{\mathbf{x},\mathbf{y}}$, and conditional distributions with the | symbol in the subscript, $f_{\mathbf{y}|\mathbf{x}}$. When a distribution is parameterized, we separate the arguments from the parameters with a ' ; ', e.g., $f_{\mathbb{X}}(\mathbb{X}; \theta)$ denotes the evaluation of $\mathbb{X}$'s distribution at $\mathbb{X}$, with parameters θ. The distribution of a Gaussian random variable with mean μ and variance σ^2 is denoted $\mathcal{N}(\cdot, \mu, \sigma^2)$. To assist the reader in interpreting the notations for this thesis, we provide examples illustrating their usage along with explanations.

Example 1.4.1. $f_{\mathbb{X}}(\cdot)$ denotes the distribution for the random set $\mathbb{X}$ (see Table 1.1), and is a function mapping sets to $\mathbb{R}_0^+$. On the other hand, $f_{\mathbb{X}}(\mathbb{X})$ denotes the evaluation of $\mathbb{X}$'s distribution for a specific set $\mathbb{X}$, and is a non-negative real number.

Example 1.4.2. $f_{\mathbf{x}|\mathbf{y}}(\cdot|y)$ is the distribution of the random scalar (or vector, see Table 1.1) $\mathbf{x}$ conditioned on the random scalar $\mathbf{y}$ taking the specific value y; it is a function mapping scalars to $\mathbb{R}_0^+$. On the other hand, $f_{\mathbf{x}|\mathbf{y}}(x|\cdot)$ is not a distribution, but the likelihood function for $\mathbf{y}$. $f_{\mathbf{x}|\mathbf{y}}(x|y)$ is the evaluation of $\mathbf{x}$'s distribution (or $\mathbf{y}$'s likelihood) at $\mathbf{x} = x$ and $\mathbf{y} = y$.

Example 1.4.3. $f_{\mathbf{z}_{1:t}|\mathbb{K}}(z_{1:t}|\mathbb{K}) \doteq \mathcal{N}(\max \mathbb{K}; z_1, z_3)$ defines the distribution of the sequence of random vectors $\mathbf{z}_{1:t}$ conditioned on the random set $\mathbb{K}$ taking

the value $\mathbb{K}$ to be equal to a normal distribution with mean z_1 and variance z_3, evaluated at the point $\max \mathbb{K}$.

CHAPTER 2

Machine Learning

This chapter presents an introduction to the field of machine learning, specifically probabilistic supervised learning, which is a central topic for the research presented in this thesis. It starts by formulating the optimization problem in this setting and then describing the techniques and algorithms that are commonly used to tackle it. After that, a section with remarks about generalization in machine learning is provided, along with considerations on its relevance to the contents of this thesis. This chapter finally concludes with a review on the Transformer [13] architecture, used in papers A, C, and D.

2.1 Introduction

Machine learning is the branch of research concerned with understanding and developing methods that learn how to solve tasks without explicit instructions about *how* to do so, instead being provided a description of *what* is to be done, and data. Such methods leverage data from the task of interest to autonomously learn the specific steps to achieve the desired goal. They are especially helpful in tasks where explicitly listing all the steps required to perform the task of interest is either too laborious or error-prone.

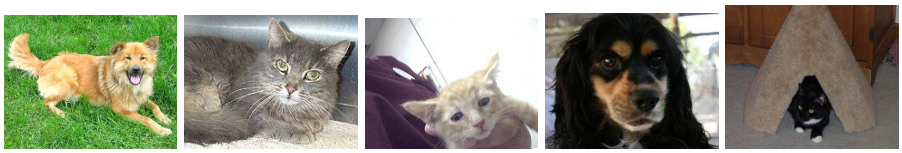


Figure 2.1: Images of cats and dogs.

For example, consider the task of writing an algorithm for classifying natural images such as the ones shown in Fig. 2.1 into one of two classes: "cat" or "dog". The presence of certain visual cues (say, whiskers, or pointy ears) guides the inference, but their location and relation to all other cues are also important. And so are their colors, texture, relative sizes, etc. There is also plenty of irrelevant information present in the image, such as the background the subject is in, other objects, whether any text is present, etc. Explicitly listing all rules and exceptions for correctly classifying such images would be a daunting task.

The field of machine learning provides a different approach. Using its tools, we can sidestep the need for explicit instructions on *how* to classify these images, and instead require only a dataset of already-classified examples: an explanation of *what* is expected of a functioning algorithm. Such a dataset can be created by asking human annotators to look at thousands of images and decide if they contain cats or dogs, for example. Machine learning methods then use this dataset to automatically learn the required steps for reproducing (and under some conditions, generalizing) the annotations present in the training dataset.

In fact, this new paradigm provided by machine learning is responsible for unlocking a level of performance in certain skills that was hitherto inaccessible to computers (and in some cases, even humans). Tasks such as image classification [43], object detection [44], translation [45], playing boardgames [46], and many others, saw a drastic increase in performance, and now computers are used in a myriad of applications requiring these capabilities.

The field has many paradigms on what kinds of tasks are tackled and how they are tackled, and how annotations are to be done (if at all): supervised learning, unsupervised learning, reinforcement learning, to cite a few [47]. This thesis is primarily concerned with *supervised learning*, and the rest of

this chapter will detail this paradigm in more depth. The reader interested in other paradigms is referred to [47]–[49].

2.2 Supervised Learning

Supervised learning (SL) is the paradigm of ML that tackles the task of learning the relationship between a set $\mathbb{Y}$ of classes or targets, and a set $\mathbb{X}$ of measurable properties or characteristics, often referred to as *features*. We assume the existence of a dataset $\mathcal{D} = \{x_i, y_i\}_{i=1}^{N}$, $x_i \in \mathbb{X}, y_i \in \mathbb{Y}$, $\forall i \in \mathbb{N}_N$, exemplifying their relation.

For instance, the task of classifying cat and dog images from Section 2.1 can be defined as learning the relationship between a set $\mathbb{X}$ of all possible natural images (like the ones shown in Fig. 2.1) and a set $\mathbb{Y} = \{\text{'cat'}, \text{'dog'}\}$. The set $\mathcal{D}$ is then the annotated dataset mentioned previously, possibly constructed by human annotators looking at images $x_i \in \mathbb{X}$ and deciding which class $y_i \in \mathbb{Y}$ the images belong to. As another example, the task of predicting the temperature for a given day could be formulated with a set $\mathbb{X}$ of all possible values of relevant historical meteorological conditions and the date of the day for which to predict temperature, and $\mathbb{Y} = \mathbb{R}$.

This thesis characterizes the relationship between $\mathbb{X}$ and $\mathbb{Y}$ probabilistically, by assuming the existence of a joint probability density $f_{\mathbf{x},\mathbf{y}}$ from which the elements of $\mathcal{D}$ were sampled from. This paradigm, denoted probabilistic supervised learning (PSL) [47, Chapter 5.7], has the task of using $\mathcal{D}$ to approximate the conditional distribution of the targets given the features: $f_{\mathbf{y}|\mathbf{x}}$. This type of characterization includes and generalizes simple mappings $f : \mathbb{X} \to \mathbb{Y}$.

Moreover, the papers in this thesis make use of parametric models for approximating $f_{\mathbf{y}|\mathbf{x}}$. Denoting the set of all parameters of a model as θ, the PSL task is then: learn a model $f_\theta : \mathbb{X} \to \text{dist}(\mathbb{Y})$ that approximates $f_{\mathbf{y}|\mathbf{x}}$, where $\text{dist}(\mathbb{Y})$ denotes the set of all valid probability distributions over $\mathbb{Y}$.

2.3 Defining the Optimization Problem

To find a model f_θ that approximates well the conditional distribution $f_{\mathbf{y}|\mathbf{x}}$, we first need to formalize what is meant by "approximate well". One way that this can be done is by defining a function $D : \text{dist}(\mathbb{Y}) \times \text{dist}(\mathbb{Y}) \to \mathbb{R}$

which encodes a notion of distance between distributions. The choice of this function depends on what we care about in the task and defines what it means to be "closer" or "further away" from the target distribution[1]. Using D, we can formalize the optimization problem as

$$f^* = \arg\min_{f \in \mathcal{F}} \mathbb{E}_{\mathbf{x} \sim f_{\mathbf{x}}} \left[D(f(\cdot|\mathbf{x}), f_{\mathbf{y}|\mathbf{x}}(\cdot|\mathbf{x})) \right], \tag{2.1}$$

where $f_{\mathbf{x}}$ is the marginal distribution of $\mathbf{x}$

$$f_{\mathbf{x}}(x) = \int f_{\mathbf{x},\mathbf{y}}(x, y) dy \,, \tag{2.2}$$

and $\mathcal{F}$ is the set of all possible models considered for the optimization and needs not (usually, does not) contain $f_{\mathbf{y}|\mathbf{x}}$. The expectation over $\mathbf{x} \sim f_{\mathbf{x}}$ is done so that the optimal model f^* is not a good approximation only for a specific outcome of $\mathbf{x}$, but a good approximation *on average*. That is, if a certain outcome x of $\mathbf{x}$ is more common in $f_{\mathbf{x},\mathbf{y}}$, (2.1) will weigh that specific $D(f(\cdot|x), f_{\mathbf{y}|\mathbf{x}}(\cdot, x))$ more when defining what is meant by the "best model".

Figure 2.2 is a helpful illustration of the optimization over $\mathcal{F}$. Three models f_1, f_2, and f^* in $\mathcal{F}$ are shown in the figure. Only f^* is optimal in this case, being the closest (in the sense defined by the choice of D) to $f_{\mathbf{y}|\mathbf{x}}$. The models f_1 and f_2 are sub-optimal, as there is a model (f^*) closer to $f_{\mathbf{y}|\mathbf{x}}$ than them.

Since we have a parametrized model, the optimization can be done over the parameter θ instead, transforming (2.1) into

$$\theta^* = \arg\min_{\theta \in \Theta} \mathbb{E}_{\mathbf{x} \sim f_{\mathbf{x}}} \left[D(f_\theta(\,\cdot\,|\mathbf{x}), f_{\mathbf{y}|\mathbf{x}}(\,\cdot\,|\mathbf{x})) \right]. \tag{2.3}$$

This equation specifies the exact conditions for a model (its parameter vector) to be optimal, in terms of D and the distribution $f_{\mathbf{x},\mathbf{y}}$. However, we do not have access to $f_{\mathbf{x},\mathbf{y}}$ in general, and can only obtain samples from it (present in the dataset $\mathcal{D}$). Fortunately, for some choices of D we can approximate the expression above using samples.

2.4 Kullback-Leibler Divergence

A common choice for which it is possible to approximate (2.3) using samples is the Kullback-Leibler divergence (KLD) [50]. Although the KLD is not a

[1] D needs not be a metric in the strict mathematical sense (in many cases, it is not).

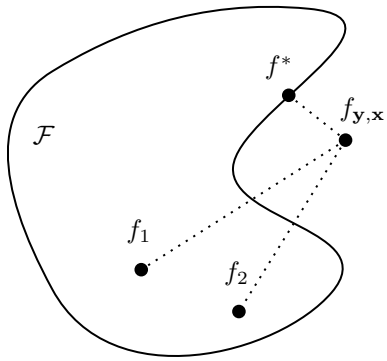


Figure 2.2: Illustration of the optimization landscape in probabilistic supervised learning. Three models within the model family $\mathcal{F}$ are shown; only one is optimal.

metric in the space of distributions (it is a divergence), using it results in a special form of equation (2.3). The KLD between two distributions f_1 and f_2 is defined as

$$KLD(f_1 \mid f_2) = \int f_1(y) \log \left(\frac{f_1(y)}{f_2(y)} \right) dy \,. \tag{2.4}$$

Using the KLD as our choice of D transforms (2.3) into

$$\theta^* = \arg\min_{\theta \in \Theta} E_{\mathbf{x} \sim f_{\mathbf{x}}} \left[\int f_{\mathbf{y}|\mathbf{x}}(y|\mathbf{x}) \log \left(\frac{f_{\mathbf{y}|\mathbf{x}}(y|\mathbf{x})}{f_\theta(y|\mathbf{x})} \right) dy \right] , \tag{2.5}$$

which is equivalent to

$$\theta^* = \arg\max_{\theta \in \Theta} E_{\mathbf{x} \sim f_{\mathbf{x}}} \left[\int f_{\mathbf{y}|\mathbf{x}}(y|\mathbf{x}) \log f_\theta(y|\mathbf{x}) dy \right] \tag{2.6}$$

$$= \arg\max_{\theta \in \Theta} E_{(\mathbf{x},\mathbf{y}) \sim f_{\mathbf{x},\mathbf{y}}} \left[\log f_\theta(\mathbf{y}|\mathbf{x}) \right] . \tag{2.7}$$

We can now use the law of large numbers to write (2.7) in terms of samples from $f_{\mathbf{x},\mathbf{y}}$

$$\theta^* = \arg\max_{\theta \in \Theta} \lim_{n \to \infty} \frac{1}{n} \sum_{i=1}^{n} \log f_\theta(y_i|x_i) \,, \tag{2.8}$$

where x_i and y_i are samples from $f_{\mathbf{x}|\mathbf{y}}$. Although (2.8) is just a specialization of (2.3), it has one fundamental difference: we can compute it without direct access to $f_{\mathbf{y}|\mathbf{x}}$.

This specific choice of D is also special for another reason. Approximating (2.8) with the finite number of samples in $\mathcal{D}$ yields

$$\theta^* = \arg\max_{\theta\in\Theta} \frac{1}{N} \sum_{i=1}^{N} \log f_\theta(y_i|x_i)\,, \tag{2.9}$$

which can be rewritten as

$$\theta^* = \arg\max_{\theta\in\Theta} \frac{1}{N} \log \prod_{i=1}^{N} f_\theta(y_i|x_i) \tag{2.10}$$

$$= \arg\max_{\theta\in\Theta} \prod_{i=1}^{N} f_\theta(y_i|x_i)\,, \tag{2.11}$$

showing that the model f_{θ^*} obtained by minimizing the KLD with respect to (w.r.t.) $f_{\mathbf{y}|\mathbf{x}}$ is also the maximum likelihood model of y_i given x_i.

2.5 Solving the Optimization

Our choice of D determines the optimization problem, defining θ^* in terms of $\{x_i, y_i\}_{i=1}^N$. In general, the optimization can be written as

$$\theta^* = \arg\min_{\theta\in\Theta} \mathcal{L}\Big(\theta, \{x_i, y_i\}_{i=1}^N\Big)\,, \tag{2.12}$$

where $\mathcal{L}$ is referred to the *loss function* for the problem at hand. For example, using the KLD for our choice of D (Section 2.4) results in the following loss function

$$\mathcal{L}\big(\theta, \{x_i, y_i\}_{i=1}^N\big) = -\frac{1}{N} \sum_{i=1}^{N} \log f_\theta(y_i|x_i)\,. \tag{2.13}$$

In some applications, the model f_θ and loss $\mathcal{L}$ are simple enough that $\mathcal{L}$ is convex in terms of θ. Finding θ^* in that scenario is then straightforward; several efficient algorithms exist for finding the global optimum of such a function. However, in most cases, such simple models are not sufficient

for obtaining good performance, and $\mathcal{L}$ is typically a complicated, high-dimensional function of the samples, with multiple local minima, plateaus, and saddle points.

This thesis considers model families comprised of different types of neural networks. Such models are powerful parametrized functions formed by stacking multiple nonlinear layers, in an overall structure for f_θ which is usually differentiable w.r.t the parameter vector θ. When this is the case, we can use gradient-based optimization for tackling the loss optimization. Although gradient-based optimization is not guaranteed to find the global optimum of (2.12), practitioners in the field have empirically discovered that this approach works surprisingly well for most tasks. The reasons for this are still partially unknown, and a great deal of research is currently devoted to better understanding this conundrum [51]–[55].

Perhaps the simplest and most well-known gradient-based optimization is gradient descent. This method starts with some initial parameter vector θ_0 (say, initialized at random) and updates it iteratively according to

$$\theta_k = \theta_{k-1} - \eta \cdot \nabla_\theta \left(\mathcal{L}\big(\theta, \{x_i, y_i\}_{i=1}^N\big) \right), \tag{2.14}$$

where ∇_θ is the gradient operator w.r.t. θ, and $\eta \in \mathbb{R}^+$ is a hyperparameter called learning rate, which has to be tuned properly for each task. That is, the parameter vector is always updated in the direction of steepest descent. Gradient descent has appealing convergence properties, such as guaranteed optimality for smooth, convex functions [56], but it has a severe drawback. At each iteration k, one must compute the loss function over *all* samples in the dataset, which becomes prohibitively expensive for modern deep learning datasets with thousands if not millions of samples.

To address this, practitioners opt instead for a modification of this algorithm called stochastic gradient descent (SGD) [57]. In it, the direction of descent is estimated using a single sample $(\mathbf{x}', \mathbf{y}')$ from the dataset

$$\theta_k = \theta_{k-1} - \eta_k \cdot \nabla_\theta \left(\mathcal{L}\big(\theta, (\mathbf{x}', \mathbf{y}')\big) \right), \tag{2.15}$$

where $\mathbf{x}'$, and $\mathbf{y}'$ are sampled uniformly at random from $\mathcal{D}$. Note that this estimate is stochastic, but its expectation equals the true descent direction. SGD has similar convergence properties to regular gradient descent, provided

that the learning rate η can be changed during training to satisfy three conditions [57]:

$$\eta_k \geq 0, \quad \forall k \geq 0\,, \tag{2.16}$$

$$\sum_{k=0}^{\infty} \eta_k = \infty\,, \tag{2.17}$$

$$\sum_{k=0}^{\infty} \eta_k^2 < \infty\,. \tag{2.18}$$

Despite its convergence properties, estimating the descent direction using a single sample can result in high variance estimates, which in practice can negatively impact training. A compromise between estimating it with a single sample and estimating it with the entire dataset is to use a subset of the dataset, usually called a "mini-batch", or just "batch".

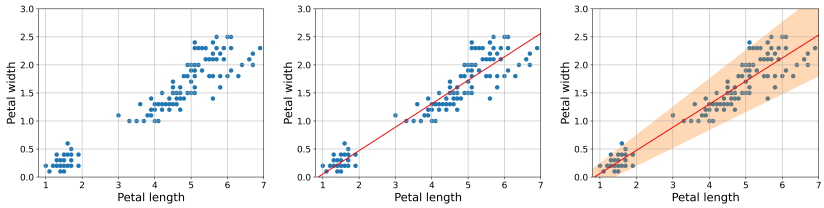
Several other extensions of SGD have been proposed, such as RMSProp [58], Adam [59], AdamW [60], and many others [61]. Comparing optimizers for a task is complicated [62], and the current consensus on the field is that there is not a single optimizer that is best for all tasks. Instead, the choice of optimizer is often either replicated from previous related work or treated as another hyperparameter to be optimized.

2.6 Examples of Supervised Learning Tasks

To further illustrate the concepts introduced in the previous sections, this section presents some examples of tasks that can be tackled with the probabilistic supervised learning framework, along with their corresponding inputs, outputs, model definitions, and illustrations of trained models.

Linear Regression

In a regression task, we are interested in learning the relationship between a *response* variable $\mathbf{y}$ which is real-valued, and an *explanatory* variable $\mathbf{x}$, and we have a dataset $\mathcal{D} = \{x_i, y_i\}_{i=1}^n$ exemplifying their relation. Both variables can be either scalars or vectors. If the response and explanatory variables are scalars, we refer to the task as univariate regression, and multivariate regression otherwise. In some cases, the relationship between the response



(a) Relationship between petal length and width. **(b)** Linear model with constant σ. **(c)** Linear model with feature-dependent σ.

Figure 2.3: Linear regression example in the IRIS flower dataset.

and explanatory variables is modeled as a linear (or affine) combination of the explanatory variables, and denoted *linear regression*.

One illustrative example of a univariate linear regression task can be made on the IRIS flower dataset [63]. This dataset comprises 150 records of three species of the Iris flower. Each record contains morphological attributes from a flower, two of which are its petal's length and width. Figure 2.3a shows the relationship between these two features in a scatter plot.

As one can see from the plot, the relationship between these variables seems mostly linear, and hence a linear model is justified. Following the probabilistic supervised learning paradigm explained in this chapter, we can tackle the linear regression task by treating $\mathbf{y}$ and $\mathbf{x}$ as random, and assuming that our dataset was sampled from their joint distribution $\mathcal{D} \sim f_{\mathbf{x},\mathbf{y}}(\cdot,\cdot)$. An interesting model f_θ for this setting is a Gaussian density with fixed standard deviation and mean which is a linear function of the explanatory variable

$$f_\theta(y|x) = \mathcal{N}(y; \theta_1^\top x + \theta_2, \sigma^2)\,. \tag{2.19}$$

In this single-variable regression task $\theta = [\theta_1, \theta_2] \in \mathbb{R}^2$ and $\sigma \in \mathbb{R}$, but σ is *not* a parameter of the network, and will not be learned (its value does not impact θ^*, as we shall see). Training this model by minimizing the loss

function in (2.13) results in the following optimization problem

$$\theta^* = \arg\max_\theta \frac{1}{N}\sum_{i=1}^{N} \log \mathcal{N}(y_i; \theta_1^\top x_i + \theta_2, \sigma^2) \tag{2.20}$$

$$= \arg\max_\theta \frac{1}{N}\sum_{i=1}^{N} -\frac{1}{2}\left(\frac{y_i - \theta_1^\top x_i - \theta_2}{\sigma}\right)^2 - \ln(\sigma\sqrt{2\pi}) \tag{2.21}$$

$$= \arg\min_\theta \frac{1}{N}\sum_{i=1}^{N} \left(y_i - \theta_1^\top x_i - \theta_2\right)^2, \tag{2.22}$$

which shows (1) that the optimum is independent of σ, and (2) that finding θ^* for this model is equivalent to minimizing the expected squared error between ground-truth (y_i) and predictions ($\theta_1^\top x_i + \theta_2$), a common approach in regular (non-probabilistic) supervised learning[2]. Figure 2.3b illustrates the result of solving for θ^*, by plotting its mean as a function of the input feature.

The advantage with PSL is that we can now go further. Instead of having a model with a fixed σ, we can use (2.9) to *train* it. In fact, it can even be a function of the features (instead of being fixed). To exemplify, consider a model of the form

$$f_\theta(y|x) = \mathcal{N}(y_i; \theta_1^\top x_i + \theta_2, \theta_3^\top x_i + \theta_4), \tag{2.23}$$

where both its mean and standard deviation are affine functions of the input feature (with independent parameters). Training it by using SGD to minimize (2.13) results in an uncertainty-aware model, illustrated in Fig. 2.3c. The figure illustrates both the mean and the $\pm 2\sigma$ region, and we can see that the model learned that there is more uncertainty in the petal's width when its length is larger. Being uncertainty-aware makes the resulting model much more useful to any downstream systems, which can now use the uncertainty quantification to decide how to best handle the model's predictions in each case.

Classification

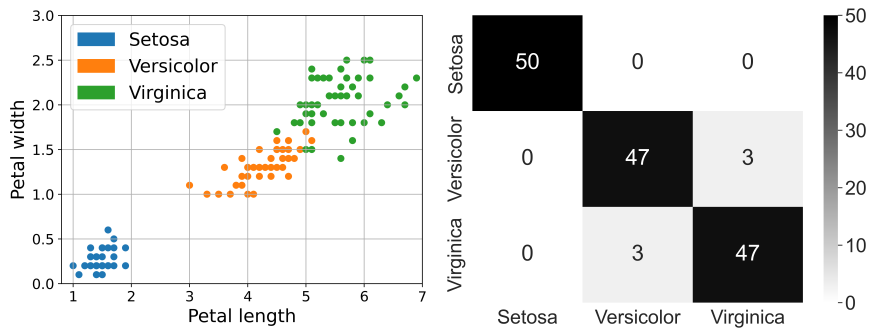
Classification is a fundamental task in machine learning that aims to learn the relationship between input features $\mathbf{x}$ and an output variable $\mathbf{y}$. In contrast to

[2]This equivalence also holds for nonlinear regression, when the model f_θ mean is a nonlinear function of the features.

regression, where the output variable is a continuous value, the output variable in classification is categorical (can only take one of a predefined number of values), and is usually referred to as the *class* of the input.

The primary goal of a classification model is to accurately predict the class of its input based on the learned patterns from the labeled training data. The input features, represented as a feature vector, can be either single or multi-dimensional, depending on the nature of the data being analyzed. These features are used to train the model to recognize patterns that are associated with different classes.

As an illustration of a possible classification task, we will revisit the IRIS dataset. This time we will attempt to use the petal length and width features to predict the class of the Iris flower. Each of the 150 examples in the dataset is annotated with the class of the flower, being either "Setosa", "Virginica", or "Versicolor". The classes of each example are illustrated in Fig. 2.4a, along with the two features studied earlier. It is clear from the graph that the two features hold significant importance in classifying the flowers accurately, even if not perfectly. For brevity, in the equations we will represent each class as



(a) Classes for all 150 records in the dataset, plotted against petal length and width.

(b) Confusion matrix for trained model.

Figure 2.4: Classification example in the Iris flower dataset.

an integer, with Setosa, Virginica, and Versicolor corresponding to 0, 1, and 2, respectively. The first step for using PSL to address this task is to select a

model. For simplicity, we train a model of the form

$$f_\theta(\mathbf{y} = i|x) = \frac{e^{z_i}}{\sum_{j=2}^{2} e^{z_j}}\,, \tag{2.24}$$

where each z_i is computed as

$$z_i = \theta_{i,1}^\top x + \theta_{i,2}\,. \tag{2.25}$$

That is, the predicted probability for $\mathbf{y}$ to have the value i (i.e., that the input flower has class i) is the softmax of affine functions of the input features. Note that now, in contrast to the regression example, x is 2-dimensional. Hence, since we have three different classes, our parameter vector θ is 9-dimensional: 6 parameters for the weights $\theta_{i,1} \in \mathbb{R}^2$ multiplying the features, and 3 parameters for the constant offsets $\theta_{i,2} \in \mathbb{R}$.

Similarly to the regression example, this model is trained by minimizing (2.13) with the help of SGD. Doing so results in a model that takes a feature vector x as input and outputs a probability mass function (pmf) over the three possible classes for y. For example, according to this trained model, the class pmf for the input vector $x = (1.5, 0.5)$ (i.e., petal length of 1.5 and petal width of 0.5) is $(0.99, 0.01, 0.00)$, showing that the model is very certain that the class for this input is Setosa (as one can see in Fig. 2.4a, all flowers have that class in that region of the feature space). On the other hand, the predicted class pmf for the input vector $x = (5.0, 1.7)$ is $(0.0, 0.51, 0.49)$, indicating that the model is uncertain between classes Versicolor and Virginica.

To get a sense of how well the model performs on the entire dataset, we can compare the most likely predicted class by the model ($\arg\max_i f_\theta(\mathbf{y} = i|x)$) with the ground-truth label, for each of the 150 records in the dataset. One way to illustrate the results obtained is by using a confusion matrix, as depicted in Fig. 2.4b. This matrix presents a comparison between the classes predicted by the model (shown on the x-axis) and the actual classes of each record (shown on the y-axis). The number inside grid cell xy represents how many dataset samples with class y were classified as x (e.g., 3 Versicolor flowers were misclassified as Virginica). The figure demonstrates that all the flowers from the Setosa species were correctly classified by the model, which is not surprising considering that this class is easily distinguishable from the others based on the two available features. Additionally, the Virginica and Versicolor classes were mostly accurately predicted, with only a few exceptions that were misclassified, corresponding to the points near a petal length of 5.

Judging from how the classes are separated in Fig. 2.4a, and the fact that we only used two of the four available features of the dataset, the model seems to perform satisfactorily. However, if the objective behind developing this model is to classify new, previously unseen flowers into one of the three categories that have been trained, one should exercise caution. The problem is that we have optimized the model only to be performant on the 150 samples of the dataset. This does not guarantee that the model will be equally performant on flowers *outside the dataset*, especially if they are from a very different distribution than the one used to sample $\mathcal{D}$. This challenge, referred to as generalization, is a significant issue for practitioners, and various tools and techniques have been developed to tackle it.

2.7 Generalization

The previous sections have explained how to train models f_θ by minimizing a distance function to a true conditional distribution $f_{\mathbf{y}|\mathbf{x}}$. To do so, we rely on a Monte-Carlo approximation to (2.7), so that we can use samples from a finite dataset $\mathcal{D}$ to train the model. Doing so raises a problem: what guarantees that minimizing the Monte-Carlo approximation will also minimize (2.7)?

This question highlights the primary difference between the field of optimization and machine learning. In machine learning, practitioners are typically interested in the performance of the model on new, previously unseen samples (denoted test performance), not simply in its performance on $\mathcal{D}$ (denoted training performance). In fact, training performance is only useful as a proxy, because it is often impossible to directly optimize test performance.

Evidently, the larger the number of samples N used, the better the Monte-Carlo approximation will be, and the smaller the gap between training and test performance. However, the field of statistical learning theory has studied this problem for decades, and one of its most important conclusions is that there is another important factor at play: the model's capacity.

Informally speaking, a model's capacity is a measure of how flexible the model is, and hence how able it is to represent a wide variety of different functions. For example, for binary classifier models, one way to quantify their model capacity is via the Vapnik-Chervonenkis dimension [64]. This dimension is defined as the largest possible number of points in feature space

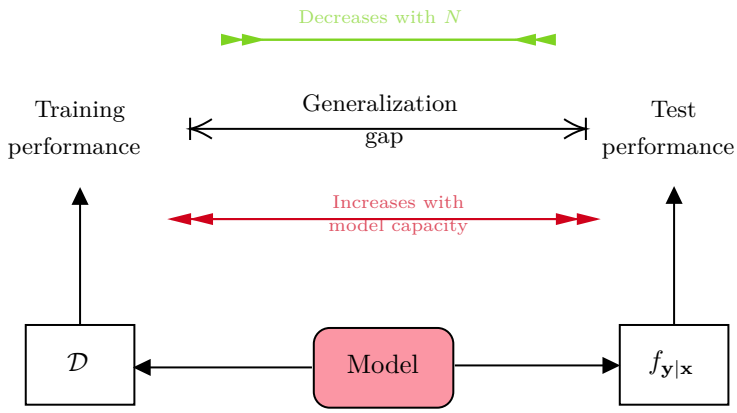


Figure 2.5: Illustration of training (model applied to samples from the dataset $\mathcal{D}$) and test performance (model applied to new samples from $f_{\mathbf{y}|\mathbf{x}}$). The difference between these performances is the generalization gap, which decreases with the number N of samples in $\mathcal{D}$, and increases with the model's capacity.

that the classifier can label *arbitrarily*. That is, no matter the labels assigned to each of the points, there is a parameter vector θ^* that allows the model to correctly classify *all of them*.

For all but trivial models, it is infeasible to compute their exact capacity. Fortunately, an exact computation is often not needed, as there are simpler takeaways from this concept that can be applied in practice. For example, any model A that *generalizes* a model B (i.e., can represent any function representable by B and more), will necessarily have higher capacity. Therefore, increasing the dimensionality of θ usually increases the model's capacity.

The most important results from statistical learning theory show that there is an upper bound on the generalization gap (the gap between training and test performance), and it depends both on the number of samples in the training set and the model's capacity [65]–[68]. The exact details are outside the scope of this thesis, but the main results are: (1) increasing the number of samples decreases the generalization gap (as mentioned previously), but also that (2) increasing the model's capacity *increases* the generalization gap. Fig. 2.5 illustrates the performance gap and its dependencies.

This brings to light an important tradeoff. Models with too low capacity for

a given task will not be able to learn all the existing patterns in the training dataset, and will therefore not be able to obtain good training performance, a phenomenon referred to as *underfitting*. On the other hand, models with too high capacity can achieve excellent training performance but might have terrible testing performance, a phenomenon referred to as *overfitting*. This balancing between under- and overfitting a model is a complex optimization problem, as one very rarely knows beforehand the optimal model capacity for a given task.

There are still missing pieces in the theory to completely explain what is seen in practice, however. Modern deep learning provides ample evidence of extremely high-capacity models being able to generalize well to the testing set, which seems to go against theory [69], [70]. Much work is currently being devoted to try and explain this behavior with new theories of learning, see [71]–[74] for examples.

Fortunately, all the papers in this thesis tackle the task of multi-object tracking in the *model-based setting*. In it, we have accurate and tractable models of the object's dynamics and measurement functions available, from which we can sample a virtually unlimited amount of training samples. By increasing the size of the dataset, we can keep the generalization gap small even when training modern deep-learning models with enormous capacity (millions of parameters), such as models based on the Transformer [13] architecture.

2.8 Review on the Transformer architecture

Due to the recent success of the Transformer architecture proposed in [13] for solving complex sequence-to-sequence learning tasks [75]–[77], parts of this architecture are used in papers A, C, and D included in this thesis. To familiarize the reader with this deep learning architecture, this section provides a background review of its most important parts.

Overall Architecture

The Transformer architecture proposed in [13] is comprised of two main components: an encoder and a decoder, as illustrated in Fig. 2.6. The encoder is in charge of processing an input sequence $z_{1:n}$ so that each element

is transformed into a new representation that encodes both its value and its relationship to other elements of the sequence. This is accomplished primarily by the use of a special type of layer called *self-attention*, and the new sequence is referred to as the embeddings $e_{1:n}$ of the input sequence, with $e_i \in \mathbb{R}^d$, $i \in \mathbb{N}_n$.

The decoder part of the architecture then processes these embeddings (using slightly modified self-attention layers) into an output sequence $y_{1:k}$, either autoregressively [13] or using learned input queries $o_{1:k}$ [14]. These components together make for a powerful learnable mapping between an input sequence $z_{1:n}$ and an output sequence $y_{1:k}$, typically trained using stochastic gradient descent on a loss function $\mathcal{L}(y_{1:k}, x_{1:k})$ that compares the network predictions with a ground-truth sequence $x_{1:k}$.

Multi-head Self-attention Layer

Before describing the encoder and decoder modules, we start with a description of their main building block: the *self-attention layer*. This layer processes an input sequence $a_{1:n}$ into an output $b_{1:n}$, starting by computing three different linear transformations of the input:

$$Q = W_Q A,\ K = W_K A,\ V = W_V A\,, \tag{2.26}$$

where $A = \big[a_1, \ldots, a_n\big] \in \mathbb{R}^{d\times n}$ is the matrix containing each of the a_i vectors as columns, and the matrices Q, K, V are referred to as queries, keys, and values, respectively. The matrices $W_Q, W_K, W_V \in \mathbb{R}^{d\times d}$ are the learnable parameters of the self-attention layer. The output sequence $b_{1:n}$ is then computed as

$$B = V \cdot \text{Softmax-c}\left(\frac{K}{\sqrt{d}}\right)\,, \tag{2.27}$$

where $B = \big[b_1, \ldots, b_n\big] \in \mathbb{R}^{d\times n}$ and Softmax-c is the column-wise application of the Softmax function, defined as

$$[\text{Softmax-c}(Z)]_{i,j} = \frac{e^{z_{i,j}}}{\sum_{k=1}^{d} e^{z_{k,j}}};\quad i,j \in \mathbb{N}_n\,,$$

for $Z \in \mathbb{R}^{n\times n}$, where $z_{i,j}$ is the element of Z on row i, column j. Because of the structure of the self-attention layer, each output b_i directly depends on all inner products of the type $a_i^\top W a_j$, for $j \in \mathbb{N}_n$, with learnable W, between

the elements of the input sequence. This allows for the potential to learn an improved representation of each a_i that takes into account its relationship to all the other elements of the sequence. Compound applications of these layers can then result in complex representations of the input sequence that take into account more complicated relationships between all the elements. This property is very important in MOT, allowing the model to learn and exploit complicated, long-range temporal patterns in the sequence of measurements.

In practice, most DL architectures use several self-attention layers in parallel and then combine the outputs, referring to the entire computation as a multi-head self-attention layer (shown in green in Fig. 2.6). Specifically, A is fed to n_h different self-attention layers (with separate learnable parameters) in parallel, generating n_h different outputs $B_1, \dots, B_{n_h}$, all $\in \mathbb{R}^{d \times n}$. The final output B is then computed by vertically stacking the results and applying a linear transformation to reduce the dimensionality back to $\mathbb{R}^{d \times n}$:

$$B = W^0 \begin{bmatrix} B_1 \\ \vdots \\ B_{n_h} \end{bmatrix}, \tag{2.28}$$

where $W^0 \in \mathbb{R}^{d \times dn_n}$ is a learnable parameter of the multi-head self-attention layer. Finally, B is converted back to a sequence $\mathbf{b}_{1:n} = \text{MultiHeadAttention}(\mathbf{a}_{1:n})$.

Transformer Encoder

The Transformer encoder is the module in charge of processing the input sequence $z_{1:n}$ into embeddings $e_{1:n}$ containing contextual information about all input elements. This module is built from N "encoder blocks" in series, as shown in the left of Fig. 2.6. The output for encoder block $l \in \mathbb{N}_N$ is computed as

$$\tilde{z}^{(l-1)}_{1:n} = z^{(l-1)}_{1:n} + q^e_{1:n}\,, \tag{2.29}$$

$$t^{(l)}_{1:n} = \text{MultiHeadAttention}(\tilde{z}^{(l-1)}_{1:n})\,, \tag{2.30}$$

$$\tilde{t}^{(l)}_{1:n} = \text{LayerNorm}(\tilde{z}^{(l-1)}_{1:n} + t^{(l)}_{1:n})\,, \tag{2.31}$$

$$z^{(l)}_{1:n} = \text{LayerNorm}(\tilde{t}^{(l)}_{1:n} + \text{FFN}(\tilde{t}^{(l)}_{1:n}))\,, \tag{2.32}$$

where MultiHeadAttention is a multi-head self-attention layer, FFN is a fully-connected feedforward neural network applied to each element of the input

sequence separately, LayerNorm is a Layer Normalization layer (introduced in [78]), and $z^{(l)}_{1:n}$ is the input sequence after being processed by l encoder blocks. For instance, $z^{(0)}_{1:n}$ is the original input sequence $z_{1:n}$, and $z^{(N)}_{1:n}$ is the output of the encoder module, also denoted $e_{1:n}$. Note that both multi-head self-attention and layer normalization preserve the size of the input, so $z^{(l)}_{1:n} \in \mathbb{R}^{d_z}$, for $l \in \mathbb{N}_N$

Importantly, $q^e_{1:n}$ in (2.29), referred to as the positional encoding for the input sequence, is added to the input of every encoder block (as done in [14]). This is computed as $q^e_i = f^e_p(i)$, where $f : \mathbb{Z} \to \mathbb{R}^{d_z}$, and f^e_p can either be fixed [13] or learnable [14]. Without it, the encoder module becomes permutation-equivariant[3], which is undesirable in many contexts. For instance, when processing text with Transformer architectures the order of the elements in the input sequence is related to their location in the sentence, essential information for correctly solving language tasks.

Transformer Decoder

The decoder module is in charge of using the embeddings $e_{1:n}$ computed by the encoder to predict the output sequence $y_{1:k}$. Several variants of this module have been proposed [13], [79], [80], each with different pros and cons. The decoder used for papers A and C is based on [14] (illustrated on the right part of Fig. 2.6), due to its speed and capacity to generate outputs in parallel, instead of autoregressively. Similarly to the encoder module, the decoder is comprised of M "decoder blocks", where the output for decoder block $l \in \mathbb{N}_M$ is computed as

$$\tilde{o}^{(l-1)}_{1:k} = o^{(l-1)}_{1:k} + q^d_{1:k}\,, \tag{2.33}$$

$$r^{(l)}_{1:k} = \text{MultiHeadAttention}(\tilde{o}^{(l-1)}_{1:k})\,, \tag{2.34}$$

$$\tilde{r}^{(l)}_{1:k} = \text{LayerNorm}(\tilde{o}^{(l-1)}_{1:k} + r^{(l)}_{1:k})\,, \tag{2.35}$$

$$\tilde{e}^{(l)}_{1:k} = \text{MultiHeadCrossAttention}(\tilde{r}^{(l)}_{1:k}, e_{1:n})\,, \tag{2.36}$$

$$\bar{e}^{(l)}_{1:k} = \text{LayerNorm}(\tilde{r}^{(l)}_{1:k} + \tilde{e}^{(l)}_{1:k}))\,, \tag{2.37}$$

$$o^{(l)}_{1:k} = \text{LayerNorm}(\bar{e}^{(l)}_{1:k} + \text{FFN}(\bar{e}^{(l)}_{1:k})\,, \tag{2.38}$$

where MultiHeadCrossAttention is a regular multi-head self-attention layer as described in Section 2.8, with the difference that the matrices K, Q, and V in

[3] A function f is equivariant to a transformation g iff $f(g(x)) = g(f(x))$.

(2.26) are respectively computed as $W_K e_{1:n}$, $W_Q \tilde{r}^{(l)}_{1:k}$, and $W_V e_{1:n}$ (all of the subsequent self-attention computations are the same).

The first encoder block receives the object queries $o_{1:k}$, which are a sequence of learnable vectors trained alongside other model parameters. After training, each $o_i \in \mathbb{R}^{d_o}$, $i \in \mathbb{N}_k$, can potentially learn to focus on specific parts of the embeddings $e_{1:n}$ that aid in predicting y_i. Similar to the encoder module, $o^{(l)}_{1:k}$ represents the object queries after passing through l decoder blocks. Here, $o^{(M)}_{1:k}$ refers to the output of the decoder module $y_{1:k}$. To ensure that the decoder module is not permutation-equivariant, a positional encoding $q^d_{1:k}$ is added to the input of each layer, computed as $q^d_i = f^d_p(i)$.

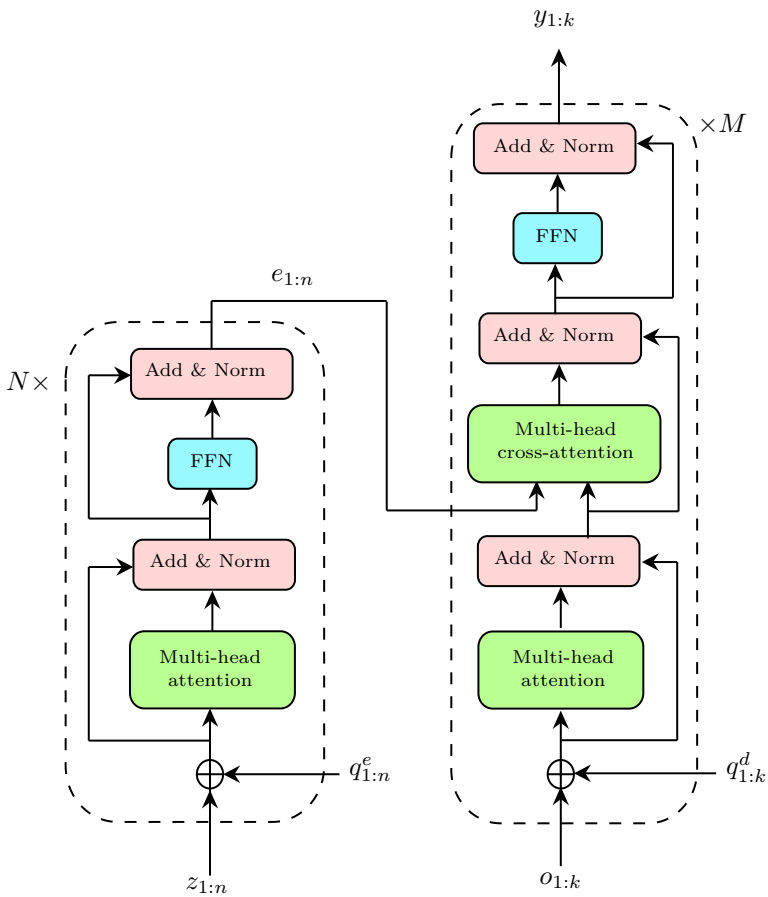


Figure 2.6: Simplified diagram illustrating the Transformer architecture. Encoder on the left, containing N encoder blocks, processes the input sequence $z_{1:n}$ into embeddings $e_{1:n}$. Decoder on the right, containing M decoder blocks, uses the embeddings $e_{1:n}$ produced by the encoder together with the object queries $o_{1:k}$ to predict the output sequence $y_{1:k}$. FFN stands for fully-connected feedforward neural network.

CHAPTER 3

Bayesian Inference

This chapter provides a background review of Bayesian inference methods. In specific, the general equation for Bayesian inference in state-space models is presented in Section 3.2, along with considerations about its tractability. Sections 3.3 and 3.4 then respectively describe the sub-problems of filtering and smoothing.

3.1 Introduction

Bayesian inference is a class of methods that reason about how to update the probability of a hypothesis H given evidence E. In specific, all algorithms in this class update the hypothesis probability according to Bayes' theorem:

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)}\,, \tag{3.1}$$

where $P(E|H)$ is the probability of observing evidence E assuming hypothesis H is true (also denoted likelihood), $P(E)$ is the probability of the evidence (regardless of whether or not H is true), and $P(H)$ and $P(H|E)$ are respectively the prior (before observing E) and posterior (after observing E)

probabilities of the hypothesis.

Due to their generality and principled way of dealing with uncertainties, Bayesian inference methods have numerous applications, ranging from localization systems such as GPS [81], to estimation of hidden parameters in biological processes [82], machine learning [83], and multi-object tracking[8]–[10].

Although all methods in this class use Bayes' theorem for updating beliefs, they can be quite distinct depending on the structure of the hypotheses and evidence considered. For example, consider a Bayesian inference algorithm for estimating how many people are currently infected by a virus given data about hospitalizations, and one for estimating the position of a car given a GPS signal. The first algorithm could use a compartmental epidemiological model such as SIR [84] together with a model of how likely it is for an infected person to be hospitalized to reason about the likelihood of a hypothesis and the probability of observing a certain evidence. On the other hand, the second algorithm would likely use a model of how GPS communication works together with a model for the car dynamics to do the same. Although both methods use Bayes' theorem to update their hypothesis probabilities, they rely on very different models and tools to do so and therefore differ substantially in their implementation.

3.2 Bayesian Inference in State-Space Models

The papers in this thesis focus on the problem of performing Bayesian inference on state-space models. A state-space model (SSM) is a stochastic process composed of a sequence of random *states* $\mathbf{x}_{1:T} \in \mathbb{R}^{d_x \times T}$ and *measurements* $\mathbf{y}_{1:T} \in \mathbb{R}^{d_y \times T}$ where:

1. The sequence $\mathbf{x}_{1:T}$ satisfies the Markovian property:

$$f_{\mathbf{x}_{t+1}|\mathbf{x}_{1:t}}(x_{t+1}|x_{1:t}) = f_{\mathbf{x}_{t+1}|\mathbf{x}_t}(x_{t+1}|x_t), \quad \forall t \in \mathbb{N}_{T-1}\,. \tag{3.2}$$

2. The measurements are conditionally independent on all previous states given knowledge about the current state:

$$f_{\mathbf{y}_t|\mathbf{x}_{1:t}}(y_t|x_{1:t}) = f_{\mathbf{y}_t|\mathbf{x}_t}(y_t|x_t), \quad \forall t \in \mathbb{N}_T\,. \tag{3.3}$$

In such a model, the states $\mathbf{x}_{1:T}$ are the quantity of interest, but they cannot be directly observed. Instead, one has to use the information from the

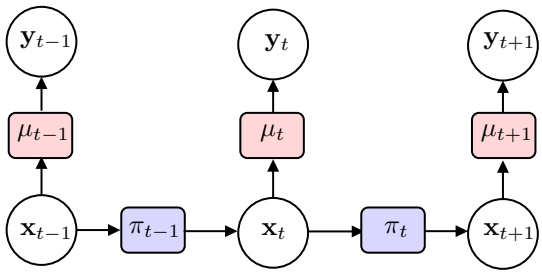


Figure 3.1: Graph depicting a state-space model and the conditional independencies between states and measurements.

measurements $\mathbf{y}_{1:T}$ to infer the conditional distribution of $\mathbf{x}_{1:T}$. A visual depiction of an SSM model is shown in Fig. 3.1. The symbols π_t and μ_t denote respectively the *motion model* and the *measurement model* of the SSM for time step t, defined as the following conditional distributions

$$\pi_t(x_{t+1}|x_t) \doteq f_{\mathbf{x}_{t+1}|\mathbf{x}_t}(x_{t+1}|x_t)\,, \tag{3.4}$$

$$\mu_t(y_t|x_t) \doteq f_{\mathbf{y}_t|\mathbf{x}_t}(y_t|x_t)\,. \tag{3.5}$$

State-space models are a common representation of systems for which we cannot directly observe the quantity of interest. For instance, in an autonomous driving setting, we are often interested in the kinematic state of the car (position, velocity, heading, etc), but we cannot directly measure these variables. Instead, we rely on measurement devices that can give us partial, noisy information about the kinematic state (such as the car's odometer, a GPS signal, etc). An SSM can be used to model this scenario by defining the kinematic state of the car at time t as the state $\mathbf{x}_t$, and the signals from the measurement devices as $\mathbf{y}_t$.

Given only $\mathbf{y}_{1:T}$, the most complete description of $\mathbf{x}_{1:T}$ is the conditional distribution $f_{\mathbf{x}_{1:T}|\mathbf{y}_{1:T}}$. It is straightforward to express this distribution for any sequence $x_{1:T}$ and $y_{1:T}$ using Bayes' theorem

$$f_{\mathbf{x}_{1:T}|\mathbf{y}_{1:T}}(x_{1:T}|y_{1:T}) = \frac{f_{\mathbf{y}_{1:T}|\mathbf{x}_{1:T}}(y_{1:T}|x_{1:T}) f_{\mathbf{x}_{1:T}}(x_{1:T})}{f_{\mathbf{y}_{1:T}}(y_{1:T})}\,. \tag{3.6}$$

Although the Markovian properties stated earlier can simplify this equation, it has two unavoidable problems. First, each time a new measurement $\mathbf{y}_t$

is obtained, the entire posterior has to be recomputed. This is problematic for online applications which receive new measurements and must provide state estimates at each time step. Second, the dimensionality of the posterior $f_{\mathbf{x}_{1:T}|\mathbf{y}_{1:T}}$ increases with the number of time steps, and consequently so does the cost of computing it. Hence, no matter how much computational power is available, (3.6) will eventually become intractable. Unless additional restrictions or information are available, there is no way to sidestep this problem, and approximations are required.

Fortunately, there are simpler variations of the problem posed above which have efficient closed-form solutions and are useful for many practical applications. This thesis is concerned with two such variations: *Bayesian filtering* and *Bayesian smoothing*. For both of these variations, we let go of the requirement of computing the joint posterior over all states $\mathbf{x}_{1:T}$ and instead settle for computing the marginals at each time step. Because of this, we can derive algorithms that require only a constant number of computations per step.

3.3 Bayesian Filtering

Bayesian filtering is the task of estimating the distribution $f_{\mathbf{x}_t|\mathbf{y}_{1:t}}$, $t \in \mathbb{N}_T$. Compared to the joint posterior over all states defined in (3.6), this distribution does not contain all the information about the sequence $\mathbf{x}_{1:T}$. For example, it does not contain enough information to compute probabilities about relations between different states, such as $P(\mathbf{x}_t > \mathbf{x}_{t-1})$. However, such considerations are not of practical interest in many applications, and this form avoids one of the challenges discussed previously in computing (3.6).

The estimation of $f_{\mathbf{x}_t|\mathbf{y}_{1:t}}$ can be efficiently solved by using a recursion with two steps, denoted *prediction* and *update*. These steps alternate between computing the *prior* and *posterior* state distributions at each time step. The prior and posterior states distributions at time t, denoted g_t^p and g_t^u respectively, are defined as

$$g_t^p(x_t|y_{1:t-1}) \doteq f_{\mathbf{x}_t|\mathbf{y}_{1:t-1}}(x_t|y_{1:t-1}), \tag{3.7}$$

$$g_t^u(x_t|y_{1:t}) \doteq f_{\mathbf{x}_t|\mathbf{y}_{1:t}}(x_t|y_{1:t}). \tag{3.8}$$

In the prediction step, we compute the prior distribution of $\mathbf{x}_t$ assuming we have access to the posterior distribution of $\mathbf{x}_{t-1}$. This is done by using the

Chapman-Kolmogorov equation

$$\underbrace{g_t^p(x_t|y_{1:t-1})}_{\text{Prior of } \mathbf{x}_t} = \int \pi_{t-1}(x_t|x_{t-1}) \underbrace{g_{t-1}^u(x_{t-1}|y_{1:t-1})}_{\text{Posterior of } \mathbf{x}_{t-1}} dx_{t-1} \,. \tag{3.9}$$

The update step then uses Bayes' theorem to compute the posterior distribution of $\mathbf{x}_t$ from its prior distribution

$$\underbrace{g_t^p(x_t|y_{1:t})}_{\text{Posterior of } \mathbf{x}_t} = \frac{\mu_t(y_t|x_t) \overbrace{g_t^p(x_t|y_{1:t-1})}^{\text{Prior of } \mathbf{x}_t}}{\int \mu_t(y_t|x') \underbrace{g_t^p(x'|y_{1:t-1})}_{\text{Prior of } \mathbf{x}_t} dx'} \,. \tag{3.10}$$

Starting from a prior $f_{\mathbf{x}_1}$, these two steps provide a closed-form solution to the marginal $f_{\mathbf{x}_t|\mathbf{y}_{1:t}}$ for any t. More importantly, this form does not have the first problem encountered when computing (3.6): the need to recompute the entire posterior whenever a new measurement arrives. By using eqs. (3.9) and (3.10), when a new measurement $\mathbf{y}_{t+1}$ arrives we only need to compute one new prediction and update step; the ones for estimating the marginal of $\mathbf{x}_{t+1}$. The marginals for all other states need no recomputing.

However, the second problem, intractability, is still present in this formulation. For nontrivial choices of the distribution families of the prior $f_{\mathbf{x}_1}$, the measurement model μ_t, and the motion model π_t, the number of parameters necessary to represent $f_{\mathbf{x}_t|\mathbf{y}_{1:t}}$ increases unboundedly with t, eventually making it intractable to compute the posterior (as the complexity of the prediction/update steps is linked to the number of parameters in the posterior for the previous time-step). One way to deal with this is by adding certain restrictions on the distribution families used in the SSM.

Linear and Gaussian Models

One such restriction is considering only SSMs where the prior for $\mathbf{x}_1$ is Gaussian, and both the motion process and measurement process are Gaussian densities with parameters that are linear functions of the state $\mathbf{x}_t$:

$$f_{\mathbf{x}_1}(x_1) = \mathcal{N}(x_1; \bar{x}_{1|1}, P_{1|1}) \,, \tag{3.11}$$

$$\pi_t(x_{t+1}|x_t) = \mathcal{N}(x_{t+1}; Fx_t, Q) \,, \tag{3.12}$$

$$\mu_t(y_t|x_t) = \mathcal{N}(y_t; Hx_t; R) \,, \tag{3.13}$$

where $\bar{x}_{1|1}$ and $P_{1|1}$ are known, $F \in \mathbb{R}^{d_x \times d_x}$, $H \in \mathbb{R}^{d_y \times d_x}$ are respectively referred to as the transition and observation matrices, and $Q \in \mathbb{R}^{d_x \times d_x}$, $R \in \mathbb{R}^{d_y \times d_y}$ are the covariance matrices of the process and measurement noises. Under these assumptions, the prediction and update steps simplify considerably, yielding a closed-form solution referred to as the Kalman Filter [85].

In the Kalman Filter, both the prior and the posterior densities of $\mathbf{x}_t$ are Gaussian

$$f_{\mathbf{x}_t|\mathbf{y}_{1:t-1}}(x_t|y_{1:t-1}) = \mathcal{N}(x_t; \bar{x}_{t|t-1}, P_{t|t-1})\,, \tag{3.14}$$

$$f_{\mathbf{x}_t|\mathbf{y}_{1:t}}(x_t|y_{1:t}) = \mathcal{N}(x_t; \bar{x}_{t|t}, P_{t|t})\,, \tag{3.15}$$

where $\bar{x}_{t|t-1} \in \mathbb{R}^{d_x}$, $P_{t|t-1} \in \mathbb{R}^{d_x \times d_x}$ are the mean and covariance parameters of the prior for $\mathbf{x}_t$, and $\bar{x}_{t|t} \in \mathbb{R}^{d_x}$, $P_{t|t} \in \mathbb{R}^{d_x \times d_x}$ the parameters for its posterior. The prediction step then amounts to computing

$$\bar{x}_{t|t-1} = F\bar{x}_{t-1|t-1}\,, \tag{3.16}$$

$$P_{t|t-1} = FP_{t-1|t-1}F^T + Q\,, \tag{3.17}$$

and the update step to computing

$$\bar{x}_{t|t} = \bar{x}_{t|t-1} + K_t(y_t - \bar{y}_t)\,, \tag{3.18}$$

$$P_{t|t} = (I - K_tH_t)P_{t|t-1}\,, \tag{3.19}$$

where $\bar{y}_t$ is the predicted measurement

$$\bar{y}_t = H\bar{x}_{t|t-1}\,, \tag{3.20}$$

K_t is the kalmain gain

$$K_t = P_{t|t-1}H^T S_t^{-1}\,, \tag{3.21}$$

and S_t is the innovation

$$S_t = HP_{t|t-1}H^T + R\,. \tag{3.22}$$

A big advantage of this form over equations (3.9) and (3.10) is that the number of parameters required to specify the posterior at time-step t is always constant. The posterior only has two parameters, of fixed size: $\bar{x}_{t|t}$ and $P_{t|t}$. This provides an efficient way to compute the exact posterior $f_{\mathbf{x}_t|\mathbf{y}_{1:t}}$ even for long sequences $y_{1:t}$.

Nonlinear Models

In certain applications, it so happens that linear motion and measurement models are not sufficient for accurately describing the phenomenon of interest, and nonlinear models must be used. However, it is often the case that the distributions of these nonlinear models can be reasonably well approximated with Gaussian distributions. When such approximations are accurate, a common approach for handling the nonlinearities is to first obtain linear models from the nonlinear ones, and then apply the Kalman filtering equations for linear Gaussian systems. There are many approaches for obtaining a linearized version of the models, based on tools such as the unscented transform, sigma-point methods, statistical linearization, and more [85].

One of the simplest and most commonly used methods is the *extended Kalman filter* (EKF). The EKF uses a 1st-order Taylor expansion for linearizing the motion and measurement models. For SSMs with additive Gaussian noise of the form

$$\pi_t(x_{t+1}|x_t) = \mathcal{N}(x_{t+1}; f(x_t), Q)\,, \tag{3.23}$$
$$\mu_t(y_t|x_t) = \mathcal{N}(y_t; h(x_t), R)\,, \tag{3.24}$$

where f and h are differentiable, the prediction step for the EKF is obtained by linearizing f around the posterior mean for $\mathbf{x}_{t-1}$, resulting in the equations

$$\bar{x}_{t|t-1} = f(\bar{x}_{t-1|t-1})\,, \tag{3.25}$$
$$P_{t|t-1} = F(\bar{x}_{t-1|t-1})P_{t-1|t-1}F(\bar{x}_{t-1|t-1})^T + Q\,, \tag{3.26}$$

where $F(\bar{x}_{t-1|t-1})$ is the jacobian of f, evaluated at the posterior mean of $\mathbf{x}_{t-1}$

$$F(\bar{x}_{t-1|t-1}) = \left.\frac{\partial f}{\partial x}\right|_{\bar{x}_{t-1|t-1}}. \tag{3.27}$$

The update equations are similarly derived by linearizing h around the prior

mean for $\mathbf{x}_t$

$$\bar{x}_{t|t} = \bar{x}_{t|t-1} + K_t(y_t - \bar{y}_t)\,, \tag{3.28}$$
$$p_{t|t} = (I - K_t H(\bar{x}_{t|t-1}))P_{t|t-1}\,, \tag{3.29}$$
$$\bar{y}_t = h(\bar{x}_{t|t-1})\,, \tag{3.30}$$
$$K_t = P_{t|t-1}H(\bar{x}_{t|t-1})^T S_t^{-1}\,, \tag{3.31}$$
$$S_t = H(\bar{x}_{t|t-1})P_{t|t-1}H(\bar{x}_{t|t-1})^T + R\,, \tag{3.32}$$

where $H(\bar{x}_{t|t-1})$ is the jacobian of h, evaluated at the prior mean of $\mathbf{x}_t$

$$H(\bar{x}_{t|t-1}) = \left.\frac{\partial h}{\partial x}\right|_{\bar{x}_{t|t-1}}. \tag{3.33}$$

These equations can then be iteratively applied to approximate $f_{\mathbf{x}_t|\mathbf{y}_{1:t}}$ for all t. As before, the arrival of new measurements only requires one additional prediction and update step to be computed, and the computational complexity for each step is the same regardless of t. Compared to alternatives for obtaining linearized models, the EKF is easier to understand and implement, and often delivers acceptable performance, making it the de-facto method for handling nonlinear SSMs.

3.4 Bayesian Smoothing

Bayesian filtering is a powerful tool when we are interested in the marginals for each $\mathbf{x}_t$ and only have the $\mathbf{y}_{1:t}$ measurements available. This is the common scenario for online applications, which at each time step receive a new measurement and should compute a new updated belief about the current state of the system. In some applications, however, knowledge about measurements further in time than $\mathbf{x}_t$ is available. *Bayesian smoothing* is a different variation of the joint estimation problem of (3.6) where we use measurements both up until and after $\mathbf{x}_t$ to update our belief about it.

Concretely, Bayesian smoothing is the task of computing $f_{\mathbf{x}_t|\mathbf{y}_{1:T}}$ where $T > t$. That is, the distribution of $\mathbf{x}_t$ conditioned on all the measurements from the first time-step up until the T-th time-step, which occurs after the time-step we are estimating the state for. This can be solved by using the

backward recursive equation

$$f_{\mathbf{x}_t|\mathbf{y}_{1:T}}(x_t|y_{1:T}) = f_{\mathbf{x}_t|\mathbf{y}_{1:t}}(x_t|y_{1:t}) \int \frac{\pi_t(x'|x_t) f_{\mathbf{x}_{t+1}|\mathbf{y}_{1:T}}(x'|y_{1:T})}{f_{\mathbf{x}_{t+1}|\mathbf{y}_{1:t}}(x'|y_{1:t})} dx' \,. \quad (3.34)$$

Before starting the recursion, Bayesian filtering is performed from time-step 1 to[1] $T+1$, computing $f_{\mathbf{x}_t|\mathbf{y}_{1:t}}$ (in the update step for time t) and $f_{\mathbf{x}_{t+1}|\mathbf{y}_{1:t}}$ (in the prediction step of time-step $t+1$). Then the equation (3.34) is iteratively computed backwards in time, from time-step T to time-step t.

Linear and Gaussian Models

As for Bayesian filtering, the Bayesian smoothing equation also simplifies considerably when dealing with linear and Gaussian SSMs with the structure as in (3.11) to (3.13). The recursive equation (3.34) simplifies to a closed-form solution named the Rauch-Tung-Striebel smoother (RTSS)

$$G_t = P_{t|t} F^T P_{t+1|t}^{-1} \,, \quad (3.35)$$

$$\bar{x}_{t|T} = \bar{x}_{t|t} + G_t(\bar{x}_{t+1|T} - \bar{x}_{t+1|t}) \,, \quad (3.36)$$

$$P_{t|T} = P_{t|t} + G_t(P_{t+1|T} - P_{t+1|t}))G_t^T \,, \quad (3.37)$$

where the marginal at each time-step is Gaussian, $f_{\mathbf{x}_t|\mathbf{y}_{1:T}} = \mathcal{N}(x_t; \bar{x}_{t|T}, P_{t|T})$, and $G_t \in \mathbb{R}^{d_x \times d_x}$ is the smoothing gain.

Nonlinear Models

Similarly to Bayesian filtering, although there are efficient closed-form solutions for the linear and Gaussian case, certain applications require the use of nonlinear models. The usual approach here is also to first obtain linear models from the nonlinear ones, via any of the techniques mentioned in Section 3.3.

To exemplify, consider again the nonlinear SSM defined in (3.23)-(3.24). One approach for performing Bayesian smoothing in this case is to linearize the motion model f around the posterior marginal for $\mathbf{x}_t$. The resulting equations define the extended RTS smoother (ERTSS). The only difference between it and equations (3.35)-(3.37) is in the computation of the smoothing

[1] The last update step is not needed for solving the recursion.

gain, which becomes

$$G_t = P_{t|t} F(\bar{x}_{t|t})^T P_{t+1|t}^{-1}, \tag{3.38}$$

where $F(\bar{x}_{t|t})$ is the Jacobian of f evaluated at the posterior mean of $\mathbf{x}_t$

$$F(\bar{x}_{t|t}) = \left.\frac{\partial f}{\partial x}\right|_{\bar{x}_{t|t}}. \tag{3.39}$$

CHAPTER 4

Random Finite Sets

This chapter provides an introduction to the framework of random finite sets. It starts with an introduction to give readers intuition about what RFSs are and why they are useful, and then details certain important operations and useful types of RFSs that will be used in this thesis.

4.1 Introduction

In some applications, it is necessary to reason about sets of entities, where there are uncertainties about both their cardinalities and the elements inside them. To give an example, consider the task of object detection in computer vision. Given an image, the algorithm is tasked with predicting bounding boxes that locate objects of interest. An example of such a task is shown in Fig. 4.1, where the goal is to locate all people in the image.

We are not given any information about: (1) how many people are present in the image, and (2) where they are located. The figure shows predictions from two fictional algorithms, in blue and in red. Note how both the number of bounding boxes and their locations/extent are different for each algorithm. Additionally, the task requires no ordering of the bounding boxes. Hence, the

Figure 4.1: An object detection task. Algorithms are presented such an image and must predict bounding boxes determining the location and extent of objects of interest, in this case people. In here, possible (imperfect) predictions for two algorithms are shown in blue and red. Note how different predictions might vary in the number of bounding boxes but also their locations/extent.

most natural way to model the required output for this task is with a set $\mathbb{B}$. If we are to develop algorithms that reason probabilistically about $\mathbb{B}$, we need a principled manner for dealing with random variables with outcomes that are sets. The framework of random finite sets accomplishes exactly this.

4.2 Definition

Random finite sets are a mathematical framework that extends the notion of scalar/vector random variables to sets. These objects obey most of the same laws of probability that regular random variables do, with the difference that their outcomes are sets, instead of scalar values or vectors. More formally, a random finite set $\mathbf{X}$ is a random variable on the set $\mathcal{F}(\mathcal{Y})$ (the set of all finite subsets of $\mathcal{Y}$), where $\mathcal{Y}$ is any Hausdorff, locally compact, and completely separable topological space.

For example, if $\mathcal{Y} = \mathbb{R}$, the resulting RFS has outcomes which are sets of real numbers, such as $\{1, \pi, -3.1\}$, $\{7, 11, 13, 17\}$, or $\{\}$. Note that there is uncertainty in both the *cardinality* of the set and its *constituents*. If $\mathcal{Y} = \mathbb{R}^3$ instead, possible outcomes could be $\{[0, 0, 0], [8.8, -3, 0]\}$, $\{[0, \pi, \pi^2]\}$, or $\{\}$.

This framework is very useful for, among others, the multi-object tracking field, as it provides a convenient and concise way to describe entities with uncertain cardinality and elements, e.g., the set of objects alive at the current time, or the set of measurements obtained in the previous time-step.

4.3 Random Finite Set Statistics

As mentioned previously, RFSs are similar to regular scalar/vector random variables, but there are important differences. This section reviews some of the most important differences and also provides definitions that will be used in the rest of the thesis.

Set Integral

Given a function $f : \mathcal{F}(\mathbb{R}^n) \to \mathbb{R}$, its *set integral* is defined as

$$\int f(\mathbb{X})\delta\mathbb{X} = \sum_{i=0}^{\infty} \frac{1}{i!} \int \cdots \int f\big(\{x_1, \ldots, x_i\}\big) dx_1 \cdots dx_i \,. \tag{4.1}$$

Intuitively, the set integral computes the area under $f(\{x_1, \ldots, x_i\})$ for all possible values of $x_1, \ldots, x_i$ (similar to a regular integral for $f : \mathbb{R}^n \to \mathbb{R}$). However, it does so not only for a single cardinality i, but instead computes the area under f for all possible cardinalities $i \in \mathbb{N}$. The sum of all these areas, each divided by $i!$, is then the final value of the set integral.

The division by $i!$ is needed due to the order-invariance of sets, to avoid summing duplicate areas. For instance, for $i = 2$, the integral

$$\iint f(\{x_1, x_2\}) dx_1 dx_2 \,, \tag{4.2}$$

evaluates f both for $(x_1 = a, x_2 = b)$ *and* $(x_1 = b, x_2 = a)$, for $a, b \in \mathbb{R}$. However, both possibilities result in the same set $\{a, b\}$, so we should divide the computed area by 2 to avoid counting duplicates like these. In general, for i variables, we can permute them in $i!$ different ways that all result in the same set, which justifies the division by $i!$ in the set integral.

Multi-Object Densities

An RFS density, often denoted *multi-object density* in the MOT literature, is a function $f : \mathcal{F}(\mathfrak{Y}) \to \mathbb{R}^+$, such that

$$\int f(\mathbb{X}) \delta\mathbb{X} = 1 \,. \tag{4.3}$$

We will denote the multi-object density for an RFS $\mathbf{X}$ using subscripts: $f_{\mathbf{X}}$. Such densities, similar to the case of regular random variables, can be used to completely specify an RFS. For instance, the density

$$f_{\mathbf{X}}(\mathbb{X}) = \begin{cases} 0.3 & \text{if } \mathbb{X} = \{\pi\} \,, \\ 0.7 & \text{if } \mathbb{X} = \{-1, 0, 1\} \,, \\ 0 & \text{otherwise.} \end{cases} \tag{4.4}$$

specifies an RFS $\mathbf{X}$ on $\mathcal{F}(\mathbb{R})$ which has only two possible outcomes: $\{\pi\}$ (30% of the time) and $\{-1, 0, 1\}$ (70% of the time). More flexible definitions of $f_{\mathbf{X}}$ will give rise to more useful RFS types. Several examples are given in Section 4.4.

Cardinality Distribution

The cardinality of a set $\mathbb{X}$, denoted $|\mathbb{X}|$ is defined as the number of elements inside the set. For example, $|\{1, 0, \pi\}| = 3$. For an RFS $\mathbf{X}$, its cardinality is a random variable on $\mathbb{N}$, with distribution

$$P\big(|\mathbf{X}| = n\big) = \frac{1}{n!} \int \cdots \int f_{\mathbf{X}}\big(\{x_1, \ldots, x_n\}\big) dx_1 \cdots dx_n \,, \tag{4.5}$$

where $f_{\mathbf{X}}$ is the multi-object density of $\mathbf{X}$. Intuitively, summing up all possible cardinalities results in

$$\sum_{n=0}^{\infty} P(|\mathbb{X}| = n) = \sum_{n=0}^{\infty} \frac{1}{n!} \int \cdots \int f_{\mathbf{X}}\big(\{x_1, \ldots, x_n\}\big) dx_1 \cdots dx_n \tag{4.6}$$

$$= \int f_{\mathbf{X}}(\mathbb{X}) \delta\mathbb{X} \,, \tag{4.7}$$

which equals 1, since f is a multi-object density.

Union of RFSs

The RFS density $f_{\mathbf{Y}}$ of an RFS $\mathbf{Y}$, defined as $\mathbf{Y} \doteq \mathbf{X}_1 \bigcup \cdots \bigcup \mathbf{X}_n$, can be computed as

$$f_{\mathbf{Y}}(\mathbb{Y}) = \sum_{\mathbb{Y} = \mathbb{X}_1 \uplus \cdots \uplus \mathbb{X}_n} f_{\mathbf{X}_1, \cdots, \mathbf{X}_n}(\mathbb{X}_1, \cdots, \mathbb{X}_n) \,, \tag{4.8}$$

where $f_{\mathbf{X}_1, \cdots, \mathbf{X}_n}$ is the joint RFS density of $\mathbf{X}_1, \ldots, \mathbf{X}_n$. The symbol $\uplus$ denotes disjoint union: the summation is over all mutually disjoint (including empty) subsets $\mathbb{X}_1, \ldots, \mathbb{X}_n$ whose union forms $\mathbb{Y}$. In the case where all $\mathbf{X}_i$ are independent, this simplifies to

$$f_{\mathbf{Y}}(\mathbb{Y}) = \sum_{\mathbb{Y} = \mathbb{X}_1 \uplus \cdots \uplus \mathbb{X}_n} \prod_{i=1}^{n} f_{\mathbf{X}_i}(\mathbb{X}_i) \,. \tag{4.9}$$

4.4 Important Examples

This section reviews certain important types of RFS that are used throughout this thesis.

Bernoulli RFS

A Bernoulli RFS is an RFS $\mathbf{B}$ on $\mathcal{F}(\mathfrak{Y})$ with multi-object density given by $f_{\mathbf{B}}(\mathbb{B}) = \mathcal{B}(\mathbb{B};\theta)$, where $\mathcal{B}(\mathbb{B};\theta)$ is shorthand for

$$\mathcal{B}(\mathbb{B};\theta) \doteq \begin{cases} 1-r & \text{if } \mathbb{B} = \emptyset\,, \\ rp(b) & \text{if } \mathbb{B} = \{b\} \text{ and } b \in \mathfrak{Y}\,, \\ 0 & \text{otherwise,} \end{cases} \tag{4.10}$$

and $\theta = (r, p)$ is the tuple containing the parameters to completely specify $\mathbf{B}$. The parameter $r \in [0,1]$ is the *existence probability* of $\mathbf{B}$, and $p(\cdot)$ is a probability density with support on $\mathfrak{Y}$. To illustrate, a Bernoulli RFS with density

$$\mathcal{B}(\,\cdot\,;(0.5,\mathcal{N}(\,\cdot\,;0,1)))$$

has possible outcomes such as $\{\}$, $\{3\}$, and $\{-0.2\}$ (and its cardinality is always at most 1).

Multi-Bernoulli RFS

A multi-Bernoulli (MB) RFS $\mathbf{Y}$ on $\mathcal{F}(\mathfrak{Y})$ is the union of n independent Bernoulli RFSs $\mathbf{X}_i$ on $\mathcal{F}(\mathfrak{Y})$, $i \in \mathbb{N}_n$. Its multi-object density $f_{\mathbf{Y}}$ can be trivially computed by applying (4.9)

$$f_{\mathbf{Y}}(\mathbb{Y}) = \sum_{\mathbb{Y}=\mathbb{X}_1 \uplus \cdots \uplus \mathbb{X}_n} \prod_{i=1}^{n} \mathcal{B}(\mathbb{X}_i;\theta_i)\,, \tag{4.11}$$

which will be abbreviated as

$$f_{\mathbf{Y}}(\mathbb{Y}) = \mathcal{MB}(\mathbb{Y};\theta)\,, \tag{4.12}$$

where $\theta \doteq \{\theta_i\}_{i=1}^{n}$ is the set containing the parameters that completely specify $\mathbf{Y}$, and θ_i denotes the parameters for the i-th Bernoulli component.

In contrast to Bernoulli RFSs, multi-Bernoulli RFSs have a more flexible cardinality distribution: an outcome $\mathbb{Y}$ from an RFS $\mathbf{Y}$ with n components can have any cardinality from zero to n. To exemplify, an RFS with density $\mathcal{MB}\big(\,\cdot\,;\{\theta_1,\theta_2\}\big)$, where

$$\theta_1 = (0.5,\mathcal{N}(\,\cdot\,;0,1))\,,$$
$$\theta_2 = (0.9,\mathcal{N}(\,\cdot\,;10,3))\,,$$

has outcomes such as $\{\}, \{-0.8\}, \{1.1, 10.9\}$, and $\{9\}$.

Multi-Bernoulli Mixture RFS

A multi-Bernoulli mixture (MBM) RFS $\mathbb{M}$ on $\mathcal{F}(\mathfrak{Y})$ is a weighted sum of m MB RFSs $\{\mathbb{Y}_1, \ldots, \mathbb{Y}_m\}$ on $\mathcal{F}(\mathfrak{Y})$. Its multi-object density $f_{\mathbb{M}}$ is directly computed by weighing the densities for each MB component

$$f_{\mathbb{M}}(\mathbb{M}) = \sum_{i=1}^{m} w_i \cdot \mathcal{MB}(\mathbb{M}; \theta_i)\,, \tag{4.13}$$

which will be abbreviated as

$$f_{\mathbb{M}}(\mathbb{M}) = \mathcal{MBM}(\mathbb{M}; \theta)\,, \tag{4.14}$$

where $\theta \doteq \{(w_i, \theta_i)\}_{i=1}^{m}$ are the parameters that completely specify the multi-Bernoulli mixture. The parameters $\{w_i\}_{i=1}^{m}$ are the non-negative weights for each mixture component and sum to 1. Additionally, θ_i denotes the parameters (existence probability and density for each Bernoulli component) for the i-th multi-Bernoulli mixture component.

Poisson RFS

A Poisson RFS $\mathbb{P}$ on $\mathcal{F}(\mathfrak{Y})$ is an RFS with multi-object density of the form

$$f_{\mathbb{P}}(\mathbb{P}) = e^{-\lambda} \lambda^{|\mathbb{P}|} \prod_{x \in \mathbb{P}} p(x)\,, \tag{4.15}$$

which will be abbreviated as

$$f_{\mathbb{P}}(\mathbb{P}) = \mathcal{P}(\mathbb{P}; \theta)\,, \tag{4.16}$$

where $\theta \doteq (\lambda, p)$ are the parameters that fully specify the Poisson RFS $\mathbb{P}$, where $\lambda \in \mathbb{R}_0^+$ and p is a density on $\mathfrak{Y}$. The name for this type of RFS comes from its cardinality distribution

$$P(|\mathbb{P}| = n) = \frac{1}{n!} \int \cdots \int f_{\mathbb{P}}\big(\{x_1, \ldots, x_n\}\big) dx_1 \cdots dx_n \tag{4.17}$$

$$= \frac{e^{-\lambda} \lambda^n}{n!}\,, \tag{4.18}$$

which is Poisson distributed with parameter λ. This gives an intuitive interpretation to a Poisson RFS: a random set with Poisson-distributed cardinality, where each element is independently sampled from $p(\cdot)$.

Poisson Multi-Bernoulli Mixture RFS

The last RFS density to be defined in this thesis is the Poisson multi-Bernoulli mixture (PMBM). A PMBM RFS $\mathbf{X}$ on $\mathcal{F}(\mathfrak{Y})$ is the union of a Poisson RFS $\mathbf{P}$ and a multi-Bernoulli mixture RFS $\mathbf{M}$. Its RFS density can be computed directly from (4.9), resulting in

$$f_{\mathbf{X}}(\mathbb{X}) = \sum_{\mathbb{X}=\mathbb{P}\uplus\mathbb{M}} \mathcal{P}(\mathbb{P};\theta_{\mathrm{P}}) \cdot \mathcal{MBM}(\mathbb{M};\theta_{\mathrm{MBM}}) \,, \tag{4.19}$$

which will be abbreviated as

$$f_{\mathbf{X}}(\mathbb{X}) = \mathcal{PMBM}(\mathbb{X};\theta) \,, \tag{4.20}$$

where $\theta \doteq (\theta_{\mathrm{P}}, \theta_{\mathrm{MBM}})$ are the parameters that fully specify the PMBM RFS $\mathbf{X}$: θ_{P}, the Poisson component parameters, and θ_{MBM}, the parameters for the multi-Bernoulli mixture component. The main importance of this RFS family is its multi-object conjugacy properties, discussed in detail in Section 5.3.

CHAPTER 5

Model-Based Multi-Object Tracking

Multi-object tracking (MOT) is the problem of recursively estimating the state of an unknown number of objects, based on a sequence of noisy sensor measurements. Methods capable of solving this problem are of high importance to a diverse set of applications, such as autonomous driving [2], [3], pedestrian tracking [1], tracking animal behavior [4], [5], military applications [6], sports players tracking [7], to cite a few.

Obtaining good performance in multi-object tracking tasks is challenging because of multiple reasons. First, the number of objects is unknown and may vary over time, as objects can enter and leave the field of view of the sensor. Second, objects may be missed by the sensor at any given time, and false measurements may be generated due to sensor noise or extraneous objects appearing in the field of view. Third, the correspondence between objects and measurements is unknown, as measurements do not contain any information about which was their originating object.

The MOT domain can be broadly classified into two categories: model-free and model-based. The primary emphasis of this thesis centers around model-based MOT, which specifically pertains to situations where accurate and tractable environment models are available for methods to leverage.

The following chapter provides a review of model-based MOT. It starts by defining what problem is to be solved, and under which conditions, in Sections 5.1 and 5.2. This is followed by descriptions of two important conjugate priors and corresponding practical implementations in Section 5.3. It then provides information on how to extend MOT to handle trajectories in 5.4, and finishes with a brief description of two important MOT metrics for comparing performance in Section 5.5.

5.1 Problem Statement

The set of states of the objects present at time-step t is denoted $\mathbf{X}_t = \{\mathbf{x}^1, \dots, \mathbf{x}^{n_t}\}$, where $\mathbf{x}^i \in \mathbb{R}^{d_x}$, $i \in \mathbb{N}_{n_t}$. The set of measurements obtained in this same time-step is denoted $\mathbf{Z}_t = \{\mathbf{z}^1, \dots, \mathbf{z}^{m_t}\}$, where $\mathbf{z}^i \in \mathbb{R}^{d_z}, i \in \mathbb{N}_{m_t}$. Since the constituents and cardinality for both of these sets are random, they are modeled as RFSs. The task of multi-object tracking is to estimate the multi-object density $f_{\mathbf{X}_t|\mathbf{Z}_{1:t}}(\cdot|\mathbb{Z}_{1:t})$ given a sequence of measurements sets $\mathbb{Z}_{1:t}$.

For conciseness, we define shorthands for the predicted multi-object posterior

$$g_t^p(\mathbb{A}|\mathbb{B}_{1:t-1}) \doteq f_{\mathbf{X}_t|\mathbf{Z}_{1:t-1}}(\mathbb{A}|\mathbb{B}_{1:t-1})\,, \tag{5.1}$$

and the updated multi-object posterior

$$g_t^u(\mathbb{A}|\mathbb{B}_{1:t}) \doteq f_{\mathbf{X}_t|\mathbf{Z}_{1:t}}(\mathbb{A}|\mathbb{B}_{1:t})\,. \tag{5.2}$$

Similarly to Chapter 3, the estimation of the multi-object density $g_t^u(\mathbb{X}_t|\mathbb{Z}_{1:t})$ for any set $\mathbb{X}_t$ can be performed by recursively computing alternating steps of prediction and update:

$$g_t^p(\mathbb{X}_t|\mathbb{Z}_{1:t-1}) = \int \boldsymbol{\pi}_t(\mathbb{X}_t|\mathbb{X}_{t-1}) g_{t-1}^u(\mathbb{X}_{t-1}|\mathbb{Z}_{1:t-1}) \delta\mathbb{X}_{t-1}\,, \tag{5.3}$$

$$g_t^u(\mathbb{X}_t|\mathbb{Z}_{1:t}) = \frac{\boldsymbol{\mu}_t(\mathbb{Z}_t|\mathbb{X}_t) g_t^p(\mathbb{X}_t|\mathbb{Z}_{1:t-1})}{\int \boldsymbol{\mu}_t(\mathbb{Z}_t|\mathbb{X}') g_t^p(\mathbb{X}'|\mathbb{Z}_{1:t-1}) \delta\mathbb{X}'}\,, \tag{5.4}$$

where

$$\boldsymbol{\pi}_t(\mathbb{X}_t|\mathbb{X}_{t-1}) \doteq f_{\mathbf{X}_t|\mathbf{X}_{t-1}}(\mathbb{X}_t|\mathbb{X}_{t-1}) \tag{5.5}$$

is referred to as the *multi-object dynamic model* for the task, governing object birth/death and state evolution. Moreover,

$$\boldsymbol{\mu}_t(\mathbb{Z}_t|\mathbb{X}_t) \doteq f_{\mathbf{Z}_t|\mathbf{X}_t}(\mathbb{Z}_t|\mathbb{X}_t) \tag{5.6}$$

is the *multi-object measurement model* for the task, characterizing how both true and false measurements are generated. Common choices for such MOT models are described in the next section.

5.2 Multi-Object Models

This section describes common choices for the multi-object measurement and dynamics models, along with their corresponding assumptions.

Multi-Object Measurement Model

We use the standard model for $\mathbb{Z}_t$ as the union of two RFS:

$$\mathbb{Z}_t = \mathbb{O}_t \cup \mathbb{C}_t \,. \tag{5.7}$$

The set $\mathbb{O}_t$ corresponds to *object measurements* from time step t: measurements originated from the objects present at that time step. The set $\mathbb{C}_t$ corresponds to *false measurements* (interchangeably referred to as clutter measurements): extraneous measurements due to sensor noise or irrelevant objects in the sensor field of view.

For defining the model that characterizes the set $\mathbb{O}_t$, we make the usual point-object assumptions that: (1) each object can generate at most one measurement, and (2) each measurement originated from at most one object. Furthermore, we assume that the probability of an object generating a measurement is conditionally independent of all other objects and measurements, given the originating object's state. Concretely, an object with state x is detected with probability $p^d(x)$. If detected, it generates a measurement $\mathbf{z} \sim \mu(\cdot|x)$, where $\mu : \mathcal{F}(\mathbb{R}^{d_z}) \to \mathbb{R}^+$ is the single-object measurement model (not to be confused with $\boldsymbol{\mu}_t$, the multi-object model).

With these assumptions, for a set of object states $\mathbb{X}_t = \{x_t^1, \ldots, x_t^{n_t}\}$, the set of object measurements $\mathbb{O}_t$ can be expressed as

$$\mathbb{O}_t = \mathbb{O}_t(x_t^1) \cup \cdots \cup \mathbb{O}_t(x_t^{n_t}) \,, \tag{5.8}$$

where $\mathbb{O}_t(x)$ is the RFS representing the set of object measurements from an

object with state x

$$f_{\mathbf{O}_t(x)}(\mathbb{O}) = \begin{cases} 1 - P^D(x) & \text{if } \mathbb{O} = \emptyset\,, \\ P^D(x)\mu(z|x) & \text{if } \mathbb{O} = \{z\},\ z \in \mathbb{R}^{d_z}\,, \\ 0 & \text{if } |\mathbb{O}| > 1\,. \end{cases} \tag{5.9}$$

Therefore, the full set of object measurements $\mathbf{O}_t$ is a multi-Bernoulli RFS with distribution

$$f_{\mathbf{O}_t}(\mathbb{O}|\mathbb{X}_t) = \sum_{\mathbb{O}=\mathbb{O}^1 \uplus \cdots \uplus \mathbb{O}^{n_t}} \prod_{i=1}^{n_t} f_{\mathbf{O}_t(x_t^i)}(\mathbb{O}^i)\,. \tag{5.10}$$

For the set of false measurements $\mathbf{C}_t$, we assume that they are distributed according to a Poisson RFS with distribution

$$f_{\mathbf{C}_t}(\mathbb{C}) = e^{-\lambda}\lambda^{|\mathbb{C}|} \prod_{z \in \mathbb{C}} p_c(z)\,, \tag{5.11}$$

where λ is a model parameter controlling the average number of false measurements per time step, and $p_c(\cdot)$ is the clutter measurement density over the measurement space, $p_c : \mathbb{R}^{d_z} \to \mathbb{R}^+$.

Putting these together, we arrive at the complete multi-object measurement model

$$f_{\mathbf{Z}_t|\mathbf{X}}(\mathbb{Z}|\{x^1, \ldots, x^{n_t}\}) = \sum_{\mathbb{Z}=\mathbb{C} \uplus \mathbb{O}^1 \uplus \cdots \uplus \mathbb{O}^{n_t}} f_{\mathbf{C}_t}(\mathbb{C}) \prod_{i=1}^{n_t} f_{\mathbf{O}_t(x_t^i)}(\mathbb{O}^i)\,. \tag{5.12}$$

Multi-Object Dynamic Model

At each time-step t, the set of object states evolves according to

$$\mathbf{X}_{t+1} = \mathbf{S}_t \cup \mathbf{B}_t\,, \tag{5.13}$$

where $\mathbf{S}_t$ and $\mathbf{B}_t$ are respectively the set of objects which have survived from and born since time step t.

Object survival is modeled as independent for each object: for a set of object states $\mathbb{X}_{t-1} = \{x_{t-1}^1, \ldots, x_{t-1}^{n_t}\}$, the set $\mathbf{S}_t$ can be expressed as the following union

$$\mathbf{S}_t = \mathbf{S}_t(x_{t-1}^1) \cup \cdots \cup \mathbf{S}_t(x_{t-1}^{n_k})\,, \tag{5.14}$$

where $\mathbb{S}_t(x)$ is an RFS which contains the updated state for an object with state x if it survived, or is empty otherwise:

$$f_{\mathbb{S}_t(x)}(\mathbb{S}) = \begin{cases} 1 - P^S(x) & \text{if } \mathbb{S} = \emptyset\,, \\ P^S(x)\pi(s|x) & \text{if } \mathbb{S} = \{s\}\,,\ s \in \mathbb{R}^{d_x}\,, \\ 0 & \text{otherwise,} \end{cases} \tag{5.15}$$

where $P^S(x)$ denotes the survival probability for an object with state x, and π is the single-object dynamic model (not to be confused with $\boldsymbol{\pi}_t$, the multi-object dynamic model). This model governs how individual object states evolve over time; the interested reader is referred to [86] for several examples of possible single-object dynamic models. Given (5.15), the set of surviving states $\mathbb{S}_t$ is a multi-Bernoulli RFS with distribution

$$f_{\mathbb{S}_t}(\mathbb{S}) = \sum_{\mathbb{S}=\mathbb{S}_1 \uplus \cdots \uplus \mathbb{S}_n} \prod_{i=1}^{n_t} f_{\mathbb{S}_t(x)}(\mathbb{S}_i)\,. \tag{5.16}$$

Regarding the set $\mathbb{B}_t$ of objects that were born since time step t, models for it usually allow for a specification of the distribution of the number of objects being born at each time-step, and the distribution of the states of these newly-born objects. Two major types of models are used in most literature: the Poisson and the multi-Bernoulli birth models.

The Poisson birth model characterizes $\mathbb{B}_t$ as an RFS with density

$$f_{\mathbb{B}_t}(\mathbb{B}) = e^{-\lambda}\lambda^{|\mathbb{B}|} \prod_{b \in \mathbb{B}} p_b(b)\,, \tag{5.17}$$

where $\lambda \in \mathbb{R}$ is a model parameter, and $p_b(\cdot)$ is the birth state density over the state space, $p_b : \mathbb{R}^{d_x} \to \mathbb{R}$. That is, at each time step the number of objects born is Poisson-distributed with parameter λ, and the states for the newborn objects are independently sampled from $p_b(\cdot)$.

Alternatively, a multi-Bernoulli birth model with N components characterizes $\mathbb{B}_t$ instead with the density

$$f_{\mathbb{B}_t}(\mathbb{B}) = \sum_{\mathbb{B}=\mathbb{B}_1 \uplus \cdots \uplus \mathbb{B}_N} \prod_{i=1}^{N} f_{\mathbb{B}_t^i}(\mathbb{B}_i)\,, \tag{5.18}$$

where

$$f_{\mathbf{B}_t^i}(\mathbb{B}) = \begin{cases} 1 - r_i & \text{if } \mathbb{B} = \emptyset\,, \\ r_i \; p_b^i(b) & \text{if } \mathbb{B} = \{b\}\,, \\ 0 & \text{otherwise.} \end{cases} \tag{5.19}$$

and the parameters for its i-th Bernoulli components are the existence probability, r_i, and the state density for a newborn object, p_b^i.

5.3 Multi-Object Conjugate Priors

Although equations (5.3) and (5.4) completely specify the MOT posterior, directly computing it is often intractable. One of the main problems is that the set integrals in these equations can become very complicated, and for many choices of dynamic and measurement models they do not have closed-form expressions.

This challenge can be sidestepped in certain classes of model families, referred to as MOT conjugate priors, which maintain the structure of the posterior after prediction and update. This section describes two useful MOT conjugate priors and their corresponding practical implementations: the MBM conjugate prior and the Poisson MBM conjugate prior.

Multi-Bernoulli Mixture Conjugate Prior

If the birth process is modeled as a multi-Bernoulli (mixture), as in (5.18), then the MBM density is a multi-object conjugate prior to the multi-object dynamic and measurement models. That is, if the multi-object posterior at time $t-1$ has the form

$$f_{\mathbf{X}_{t-1}|\mathbf{Z}_{1:t-1}}(\mathbb{X}|\mathbb{Z}_{1:t-1}) = \mathcal{MBM}(\mathbb{X}; \theta_{t-1|t-1})\,, \tag{5.20}$$

where

$$\theta_{t-1|t-1} = \left\{ \left(w_h, \left\{ r_h^i, p_h^i \right\}_{i=1}^N \right) \right\}_{h=1}^H\,, \tag{5.21}$$

then it can be shown that

$$\begin{aligned} f_{\mathbf{X}_t|\mathbf{Z}_{1:t-1}}(\mathbb{X}|\mathbb{Z}_{1:t-1}) &= \int \boldsymbol{\pi}_t(\mathbb{X}|\mathbb{X}_{t-1}) \mathcal{MBM}(\mathbb{X}_{t-1}; \theta_{t-1|t-1}) \delta\mathbb{X}_{t-1} \\ &= \mathcal{MBM}(\mathbb{X}; \theta_{t|t-1})\,, \end{aligned}$$

and that

$$f_{\mathbf{X}_t|\mathbf{Z}_{1:t}}(\mathbb{X}|\mathbb{Z}_{1:t}) = \frac{\boldsymbol{\mu}_t(\mathbb{Z}_t|\mathbb{X})\mathcal{MBM}(\mathbb{X};\theta_{t|t-1})}{\int \boldsymbol{\mu}_t(\mathbb{Z}_t|\mathbb{Y})\mathcal{MBM}(\mathbb{Y};\theta_{t|t-1})\delta\mathbb{Y}} = \mathcal{MBM}(\mathbb{X};\theta_{t|t}),$$

i.e., both the predicted and the updated posterior densities at time-step t are also multi-Bernoulli mixture densities. Therefore, the prediction and update steps of an MBM filter are simplified to the computation of the updated parameters $\theta_{t|t-1}$ and $\theta_{t|t}$, for which the interested reader is referred to [87] for explicit equations.

Besides yielding simpler prediction and update steps, the multi-Bernoulli mixture density also provides an intuitive interpretation of its parameters in the MOT context. Each of the H multi-Bernoulli components in the mixture corresponds to one possible sequence of data associations, with probabilities $\{w_h\}_{h=1}^{H}$. Within one multi-Bernoulli h, its Bernoulli components correspond to potential objects present in the FOV given a data association, each with a state density $p_h^i(\cdot)$ and existence probability r_h^i.

Although this conjugate prior allows for an intuitive interpretation and simpler prediction and update steps, the number of parameters necessary to exactly describe the multi-object posterior density grows rapidly over time. In the prediction step the number N of Bernoulli components in each hypothesis h of the MBM density is increased by n, the number of Bernoulli components in the MB birth model. In the update step, due to the unknown correspondence between measurements and objects, the number of hypotheses H is multiplied by the number of possible associations between N Bernoulli components and $|\mathbb{Z}_t|$ measurements, which is immense in all but trivial applications. Therefore, filters using the MBM conjugate prior must resort to certain approximations for maintaining computational tractability. The MBM filter [87], for example, prunes multi-Bernoulli components with low weights and Bernoulli components with low existence probability after every update step.

Another relevant filter to this thesis is the MBM_{01} filter. It uses the same prediction and update steps as the MBM filter, but performing MBM_{01} expansion after every prediction step. This results in a super-exponential increase in the number of multi-Bernoulli components for this filter. The δ-generalized labelled multi-Bernoulli filter [10] is equivalent to the labelled (see Section 5.4) MBM_{01} filter.

Poisson Multi-Bernoulli Mixture Conjugate Prior

Similarly to the MBM density, the PMBM density is a multi-object conjugate prior to the measurement model described earlier and a dynamics model with Poisson birth. If the multi-object posterior at time $t-1$ has the form

$$f_{\mathbf{X}_{t-1}|\mathbf{Z}_{1:t-1}}(\mathbb{X}|\mathbb{Z}_{1:t-1}) = \mathcal{PMBM}(\mathbb{X};\theta_{t-1|t-1})\,, \tag{5.22}$$

then it can be shown that

$$\begin{aligned} f_{\mathbf{X}_t|\mathbf{Z}_{1:t-1}}(\mathbb{X}|\mathbb{Z}_{1:t-1}) &= \int \boldsymbol{\pi}_t(\mathbb{X}|\mathbb{X}_{t-1})\mathcal{PMBM}(\mathbb{X}_{t-1};\theta_{t-1|t-1})\delta\mathbb{X}_{t-1} \\ &= \mathcal{PMBM}(\mathbb{X};\theta_{t|t-1})\,, \end{aligned}$$

and that

$$\begin{aligned} f_{\mathbf{X}_t|\mathbf{Z}_{1:t}}(\mathbb{X}|\mathbb{Z}_{1:t}) &= \frac{\boldsymbol{\mu}_t(\mathbb{Z}_t|\mathbb{X})\mathcal{PMBM}(\mathbb{X};\theta_{t|t-1})}{\int \boldsymbol{\mu}_t(\mathbb{Z}_t|\mathbb{X})\mathcal{PMBM}(\mathbb{X};\theta_{t|t-1})\delta\mathbb{Y}} \\ &= \mathcal{PMBM}(\mathbb{X},\theta_{t|t})\,, \end{aligned}$$

i.e., the predicted and updated multi-object posterior densities are also Poisson multi-Bernoulli mixture densities. The interested reader is referred to [8], [9] for explicit equations for computing the updated parameters $\theta_{t|t-1}$ and $\theta_{t|t}$.

Besides its conjugacy property, the PMBM density is also particularly well-suited for representing MOT uncertainties. Because of the addition of the Poisson component, the PMBM density is capable of explicitly representing possible existing but undetected objects: the Poisson RFS state distribution encodes the estimate of where in the state-space it is more likely that the undetected objects are. Additionally, the initiation of new Bernoulli components is measurement-driven: instead of always adding the same number of Bernoulli components in the prediction step (like MBM filters must), Bernoullis are only added to a multi-Bernoulli component if they are associated to an existing measurement in the update step. This leads to a more efficient representation of the hypotheses, which in turn makes PMBM filters generally better than MBM filters in terms of computational cost and estimation error [87]–[89].

Similarly to the MBM filters, PMBM filters also require approximations for maintaining computational tractability, as the number of multi-Bernoullis components in the mixture also grows very quickly with time. For example,

filters such as [8] remove Bernoulli components with low probability of existence by approximating them as a Poisson RFSs and adding it to the Poisson component (recycling), remove multi-Bernoulli components with low weights (pruning), and limit the maximum number of components to a predefined maximum (capping).

5.4 MOT for Trajectories

So far we have only described MOT algorithms that perform filtering: estimating the set of object states at time t, given measurements up until that point. However, in some applications it is important to also link states over time into individual trajectories for each object. Given only the marginal distributions for $\mathbf{X}_{t-1}$ and $\mathbf{X}_t$, we do not have enough information to decide which states at time $t-1$ correspond to the ones at time t.

This section explains two common approaches for dealing with this challenge: labeling object states and considering trajectories as objects.

Labelled Object States

One way to automatically form trajectories is to add *labels* to objects, so that each object is identified over its lifetime [90], [91]. Each object state is now defined as a tuple $(\mathbf{x}, \mathbf{L})$, where $\mathbf{x} \in \mathbb{R}^{d_x}$ is what we so far have called the state of the object, and $\mathbf{L} \in \mathbb{N}$ is a label associated to it. Importantly, labels must be unique to each object to guarantee that linking states over time according to their labels generates correct trajectories.

To do so, one must extend the multi-object birth model to also label objects uniquely at the time of their birth. Although it is not mathematically possible to guarantee label uniqueness for some birth models (e.g., the Poisson birth model described in (5.17)), this is possible for the MB birth model. Doing so results in a *labeled* multi-Bernoulli birth model, for which the conjugate prior is a labeled multi-Bernoulli mixture density. One filter based on this conjugate prior is the δ-generalized labeled multi-Bernoulli mixture (δ-GLMB) filter [10], which is capable of estimating complete trajectories via the use of labels.

Although assigning labels to objects seems like a straightforward way for linking object states through time, it has some problems. First, labels by themselves do not have any physical meaning, and the association between

objects and labels is arbitrary: an object that has been assigned a label "3" could just as well be assigned a label "4" instead. Not only this increases the dimensionality of the state without good reason, but it also makes it more complicated to propose metrics with physical interpretation between sets of labeled states.

Second, tracking is usually performed by estimating the multi-object densities at each time-step, instead of the joint posterior distribution over all time steps. This, when using labels to form trajectories, can cause unrealistic track switches in settings where multiple objects are born at the same time or when objects get close to each other and then separate.

Third, the use of labels makes it challenging to fuse information in a multiple-sensor environment. Consider two independent trackers using sensors with overlapping field-of-view. Since labels do not have any physical meaning, and the association between them and objects is arbitrary, it is not straightforward to decide how to fuse estimates from these trackers.

Sets of Trajectories

An alternative way to link object states through time, which sidesteps the aforementioned problems, is to perform filtering on sets of trajectories [92], instead of sets of objects.

A trajectory is represented as a tuple of the form $(t, x_{1:i})$, where t is the initial time-step of the trajectory, i is its length, and the sequence $x_{1:i}$ contains the object states at all time steps in the window $[t, t+i-1]$. The set of all possible trajectories up to a finite time k, denoted $\mathbb{T}(k)$ is then computed as

$$\mathbb{T}(k) = \uplus_{(t,i)\in\mathbb{I}(k)} \{t\} \times \mathbb{R}^{d_x \times i}, \tag{5.23}$$

where $\mathbb{I}(k)$ denotes the set of all possible start times and durations for trajectories up until time k

$$\mathbb{I}_k \doteq \{(t,i) : 0 \leq t \leq k \text{ and } 1 \leq i \leq k-t+1\}. \tag{5.24}$$

The integral for a real-valued function $p(\cdot)$ on a the trajectory space $\mathbb{T}(k)$, henceforth referred to as a *trajectory integral* is defined as

$$\int p(x)dx \doteq \sum_{(t,i)\in\mathbb{I}(k)} \int \cdots \int p(t, x_{1:i})dx_1 \cdots dx_i. \tag{5.25}$$

Note that it sums over all possible start times t and durations i up until time k, and integrates over all possible corresponding state sequences $x_{1:i}$. Using this definition it is possible to define the set integral for sets of trajectories according to (4.1), and consequently to define RFS densities whose outcomes are sets of trajectories. By doing so, the equations for MOT prediction (5.3) and update (5.4) generalize to perform Bayesian inference on sets of trajectories, automatically yielding principled methods for tackling the problem of linking states through time.

In specific, it is possible to show that the PMBM multi-object density for sets of trajectories is a conjugate prior to the standard multi-object models with Poisson birth [11], which allows for the derivation of the trajectory-PMBM (TPMBM) filter. The TPMBM filter contains information about the entire trajectories of all objects directly in the estimated posterior [11] and generalizes the PMBM filter. Similarly to the PMBM filter, the trajectory Poisson RFS component of the TPMBM filter models the trajectories of the set of undetected objects. Correspondingly, the MBM component models the trajectories of the set of detected objects, where each multi-Bernoulli corresponds to one possible sequence of data associations. The interested reader is referred to [11] for explicit equations and implementation details.

5.5 MOT Metrics

Because of the unknown correspondence between measurements and their originating objects, evaluating performance in MOT is not trivial. Given an estimated multi-object posterior $f_{\mathbf{X}_t|\mathbf{Z}_{1:t}}(\cdot)$ and a set of ground-truth object states $\mathbb{X}_t$, how can one evaluate the quality of the estimate?

Several MOT performance measures have been proposed in the literature, such as the Hausdorff and Wasserstein metrics [93], multi-object tracking precision and accuracy (MOTP and MOTA) [94], higher-order tracking accuracy (HOTA) [95], optimal subpattern assignment (OSPA) [96], generalized OSPA (GOSPA) [97], to name a few.

All of these metrics first require a point-estimate $\hat{\mathbb{X}}$ to be extracted from the multi-object posterior. The exact details on how to do it are left to practitioners, and vary depending on the RFS distribution used to model $f_{\mathbf{X}_t|\mathbf{Z}_{1:t}}(\cdot)$. For example, one way [8] to extract such an estimate from a PMBM posterior is to use the means of the Bernoullis of the most likely MB component

which has existence probability above a predetermined threshold.

These types of estimate extraction discard most of the uncertainty information available in $f_{\mathbf{X}_t|\mathbf{Z}_{1:t}}(\cdot)$, which is unwanted in many settings. There have been efforts to extend MOT performance measures to be more uncertainty-aware, but, except for the performance measure proposed in paper B, such efforts only evaluate some of the available uncertainty [15], [16], and/or require access to ground-truth uncertainties which are not applicable to general MOT applications [16].

In the model-free multi-object setting, MOTA and MOTP are popular choices, especially in computer vision tasks. On the other hand, OSPA and its generalization GOSPA are common choices for model-based MOT. As this thesis focuses on model-based MOT, we will provide a review of the GOSPA metric and its trajectory-based counterpart, trajectory-GOSPA [98], two popular metrics in this setting.

GOSPA Metric

The GOSPA metric, introduced in [97], is a metric on the space of finite sets of object states and can take any non-negative finite value, with lower values corresponding to better performance. It deals with the unknown object-measurement correspondence by solving a minimum-cost assignment problem with a user-defined distance function, penalizing false positives and false negatives in a sound manner.

For two sets $\mathbb{X} = \{x_1, \ldots, x_{|\mathbb{X}|}\}$ and $\mathbb{Y} = \{y_1, \ldots, y_{|\mathbb{Y}|}\}$, where $|\mathbb{X}| \leq |\mathbb{Y}|$, the GOSPA metric is defined as

$$d_p^{(c,\alpha)}(\mathbb{X}, \mathbb{Y}) \doteq \left(\min_{\pi \in \Pi_{|\mathbb{Y}|}} \sum_{i=1}^{|\mathbb{X}|} d^{(c)}(x_i, y_{\pi(i)})^p + \frac{c^p}{\alpha}(|\mathbb{X}| - |\mathbb{Y}|) \right)^{\frac{1}{p}}, \tag{5.26}$$

$$d^{(c)}(x_i, y_j) \doteq \min\left(d_b(x_i, y_j), c\right), \tag{5.27}$$

where Π_n denotes the set of all permutations of $\{1, \ldots, n\}$, for any $n \in \mathbb{N}$, and elements $\pi \in \Pi_n$ are tuples of the form $\pi = (\pi(1), \ldots, \pi(n))$. The function $d_b(\cdot, \cdot)$ is a user-defined metric on the space of the elements of $\mathbb{X}$ and $\mathbb{Y}$ (e.g., $\mathbb{R}^{d_x}$ in the case of object states). Furthermore, $c \in \mathbb{R}^+$ denotes the cutoff hyperparameter, which defines the maximum allowed localization error, and $1 \leq p < \infty$ determines how heavily outliers will influence the metric value.

Lastly, $0 < \alpha \leq 2$ is another hyperparameter, which at value 2 allows the GOSPA metric to be rewritten as an optimization over assignment sets [97]

$$d_p^{(c,2)}(\mathbb{X}, \mathbb{Y}) = \left[\min_{\gamma \in \Gamma} \left(\sum_{(i,j) \in \gamma} d_b(x_i, y_j)^p + \frac{c^p}{2} (|\mathbb{X}| + |\mathbb{Y}| - 2|\gamma|) \right) \right]^{\frac{1}{p}}, \quad (5.28)$$

where an assignment γ between the sets of indices $\{1, \ldots, |\mathbb{X}|\}$ and $\{1, \ldots, |\mathbb{Y}|\}$ is a set with the following properties

$$\begin{aligned} \gamma &\subset \{1, \ldots, |\mathbb{X}|\} \times \{1, \ldots, \mathbb{Y}\}, \\ (i,j), (i,j') \in \gamma &\implies j = j', \\ (i,j), (i',j) \in \gamma &\implies i = i', \end{aligned}$$

and Γ denotes the set of all possible such assignment sets.

Trajectory-GOSPA

The trajectory-GOSPA (TGOSPA) metric is an extension of GOSPA to sets of trajectories. It handles the unknown object-measurement correspondence in a similar way as GOSPA, but it requires solving a multidimensional assignment problem instead, due to trajectory matches being allowed to change over time.

We start with a few necessary definitions. For a set of trajectories $\mathbb{X} = \{X^1, \ldots, X^{|\mathbb{X}|}\}$, each trajectory X^i is of the form $(\omega^i, x^i_{1:l(i)})$, where $\omega^i \in \mathbb{N}$ is the initial time of the trajectory, $l(i) \in \mathbb{N}$ is its length, and $x^i_{1:l(i)}$ is the sequence of object states at the $l(i)$ consecutive time steps starting from ω^i. Given a single trajectory X^i, the set $\tau_t(X^i)$ denotes the state of that trajectory at time t

$$\tau_t(X^i) \doteq \begin{cases} \{x^i_{t+1-\omega}\} & \text{if } \omega \leq t \leq \omega + l(i) - 1, \\ \emptyset & \text{otherwise.} \end{cases} \quad (5.29)$$

Additionally, for a set of trajectories $\mathbb{X} = \{X^1, \ldots, X^{|\mathbb{X}|}\}$, $\tau_t(\mathbb{X})$ denotes the set of all object states from these trajectories at time t

$$\tau_t(\mathbb{X}) \doteq \bigcup_{i=1}^{|\mathbb{X}|} \tau_t(X^i). \quad (5.30)$$

We can now define the metric. For two sets $\mathbb{X}$ and $\mathbb{Y}$ of trajectories up to time step T, the TGOSPA metric is defined as

$$d_p^{(c,\gamma)}(\mathbb{X},\mathbb{Y}) = \min_{\substack{\pi_t \in \Pi_{\mathbb{X},\mathbb{Y}} \\ t=1,\dots,T}} \left(\sum_{t=1}^{T} d^t_{\mathbb{X},\mathbb{Y}}(\mathbb{X},\mathbb{Y},\pi_t)^p + \sum_{t=1}^{T-1} s_{\mathbb{X},\mathbb{Y}}(\pi_t,\pi_{t+1})^p \right)^{\frac{1}{p}} , \quad (5.31)$$

where $\Pi_{\mathbb{X},\mathbb{Y}}$ denotes the set of all possible assignment vectors between the index sets $\{1,\dots,|\mathbb{X}|\}$ and $\{0,\dots,|\mathbb{Y}|\}$. An assignment vector $\pi_t = [\pi_t^1,\dots,\pi_t^{|\mathbb{X}|}]^T$ at time step t is a vector $\pi_t \in \{0,\dots,\mathbb{Y}\}^{|\mathbb{X}|}$ such that for its i-th component $\pi_t^i = \pi_t^{i'} = j > 0 \implies i = i'$. Also, $\pi_t^i = j \neq 0$ implies that trajectory i in $\mathbb{X}$ is assigned to trajectory j in $\mathbb{Y}$ at time step t and $\pi_t^i = 0$ implies that trajectory i in $\mathbb{X}$ is unassigned at time step t. Further, we have

$$d^t_{\mathbb{X},\mathbb{Y}}(\mathbb{X},\mathbb{Y},\pi_t)^p = \quad (5.32)$$

$$\sum_{(i,j)\in\theta_t(\pi_t)} d(\mathbb{X}_t^i,\mathbb{Y}_t^j)^p + \frac{c^p}{2}\left(|\tau_t(\mathbb{X})| + |\tau_t(\mathbb{Y})| - 2|\theta_t(\pi_t)|\right) , \quad (5.33)$$

where

$$\theta_t(\pi_t) = \left\{ (i,\pi_t^i) : i \in \{1,\dots,|\mathbb{X}|\}, |\mathbb{X}_t^i| = |\mathbb{Y}_t^{\pi_t^i}| = 1, d(\mathbb{X}_t^i,\mathbb{Y}_t^{\pi_t^i}) < c \right\} , \quad (5.34)$$

and $\mathbb{X}_t^i$ and $\mathbb{Y}_t^j$ are respectively abbreviations for $\tau_t(X^i)$ and $\tau_t(Y^j)$. Note that for $(i,j) \in \theta_t(\pi_t)$, $\mathbb{X}_t^i$ and $\mathbb{Y}_t^{\pi_t^i}$ contain exactly one element, with their distance being smaller than c. Hence, the distance $d(\mathbb{X}_t^i,\mathbb{Y}_t^j)$ coincides with $d_b(\cdot,\cdot)$ evaluated at the corresponding object states, i.e., the localization error on a user-defined distance function between object states (similar to (5.26)). Hence, (5.33) represents the sum of localization errors for properly detected objects (assignments in $\theta_t(\pi_t)$), number of missed objects ($|\tau_t(\mathbb{X})| - |\theta_t(\pi_t)|$) and false predictions ($|\tau_t(\mathbb{Y})| - |\theta_t(\pi_t)|$) at time step t.

Finally, the switching cost from time step t to $t+1$ is given by

$$s_{\mathbb{X},\mathbb{Y}}(\pi_t,\pi_{t+1})^p = \gamma^p \sum_{i=1}^{|\mathbb{X}|} s(\pi_t^i,\pi_{t+1}^i) , \quad (5.35)$$

$$s(\pi_t^i,\pi_{t+1}^i) = \begin{cases} 0 & \text{if } \pi_t^i = \pi_{t+1}^i , \\ 1 & \text{if } \pi_t^i \neq \pi_{t+1}^i, \pi_t^i \neq 0, \pi_{t+1}^i \neq 0 , \\ \frac{1}{2} & \text{otherwise.} \end{cases} \quad (5.36)$$

CHAPTER 6

Summary of included papers

This chapter provides a summary of the included papers.

6.1 Paper A

Juliano Pinto, Georg Hess, William Ljungberh, Yuxuan Xia, Lennart Svensson, Henk Wymeersch
Next Generation Multitarget Trackers: Random Finite Set Methods vs Transformer-based Deep Learning
Proceedings of the 4th International Conference on Information Fusion (FUSION), granted best student paper award
IEEE, 2021, pp. 1–8 .

This paper contains a preliminary analysis into the feasibility of training deep learning models for MOT as a tractable alternative to standard model-based methods based on the RFS formalism. It proposes the "Multi-Target Tracking Transformer" (MT3), a DL model using a Transformer-based encoder-decoder architecture similar to DETR [14], but adapted to the multi-object setting. This architecture is supervised with a combination of

the loss proposed in [14] and a novel auxiliary loss developed specifically for MOT. We evaluate the performance of MT3 and compare it to the state-of-the-art RFS-based trackers PMBM [8] and δ-GLMB [10] in tasks with a constant velocity motion model and a simple measurement model that directly measures position with added Gaussian noise. To the best of our knowledge, this is the first thorough comparison between RFS-based trackers and deep learning trackers using modern architectures like Transformers. The results show that the DL tracker is able to match performance to the benchmarks in simpler tasks, while outperforming them when the data association complexity increases.

6.2 Paper B

Juliano Pinto, Yuxuan Xia, Lennart Svensson, Henk Wymeersch
An Uncertainty-Aware Performance Measure for Multi-Object Tracking
Published in IEEE Signal Processing Letters
IEEE, 2021, vol. 28, no. 1689–1693 .

Most of the existing performance measures for MOT require practitioners to extract point-estimates from the multi-object posterior of the methods being compared, causing all the uncertainty information to be lost in the comparison. Although certain works have attempted to include more uncertainty information to MOT evaluations, these have several shortcomings that make them not ideal for most settings. This letter proposes a performance measure for MOT which is uncertainty aware, hyperparameter-free, and mathematically principled, based on the negative log-likelihood (NLL) of the multi-object posterior. Efficient algorithms for approximating this measure for several common families of MOT models are provided, which in some cases allow the performance measure to decompose into easily understandable components. Additionally, examples are provided to illustrate the benefits of using NLL as a performance measure, along with a connection between it and the popular GOSPA metric.

6.3 Paper C

Juliano Pinto, Georg Hess, William Ljungberh, Yuxuan Xia, Henk Wymeersch, Lennart Svensson
Deep Learning for Model-Based Multi-Object Tracking
Accepted for publication in IEEE Transactions on Aerospace and Electronic Systems, 2023 .

This paper extends the preliminary analysis done in paper A. It proposes several extensions to MT3, such as a redesigned selection mechanism, new loss formulations, and a new architecture capable of predicting entire kinematic states along with uncertainty estimates, resulting in the new DL-based tracker MT3v2. The comparison to state-of-the-art RFS-based trackers is considerably extended, with new tasks of varying complexity using a realistic nonlinear RADAR measurement model, and a new performance measure being included in evaluations. An error analysis is also included, depicting how the missed ratio for each of the considered trackers depends on the tracked object's position in the field of view. This paper complements and extends paper A, providing stronger evidence towards the applicability of deep learning to the model-based MOT setting, not just to the model-free counterpart.

6.4 Paper D

Juliano Pinto, Georg Hess, Yuxuan Xia, Henk Wymeersch, Lennart Svensson
Transformer-Based Multi-Object Smoothing with Decoupled Data Association and Smoothing
In review .

This paper investigates the use of deep learning in the model-based multi-object smoothing setting. The smoother "Deep Decoupled Data Association and Smoothing" (D3AS) is proposed, specifically designed for this setting with an architecture that decouples the data association and filtering subtasks by performing them in separate modules, resulting in more performant and efficient smoothing. The proposed data association module and the corresponding data association loss provide a novel learnable

architecture for learning soft associations between a variable number of objects and classes in a general fashion. The resulting smoother is thoroughly compared to the state-of-the-art model-based smoother TPMBM, with results providing evidence that DL-based smoothers can outperform traditional approaches in complicated, nonlinear smoothing tasks.

CHAPTER 7

Concluding Remarks and Future Work

This thesis studies how deep learning can be applied to model-based multi-object tracking. The available models of the environment are used to generate unlimited training data for training modern deep learning architectures, resulting in systems that are flexible and performant even in challenging tracking settings. Concluding remarks and future work of the included papers are provided here.

- **Paper A: Next Generation Multitarget Trackers: Random Finite Set Methods vs Transformer-based Deep Learning** This paper provides a preliminary investigation of the performance of deep-learning trackers for model-based MOT. A model tailored to the model-based setting is proposed, MT3, and its performance compared to two state-of-the-art RFS-based trackers. The results provide evidence for the applicability of DL in this context. Interesting future directions are to investigate the capabilities of MT3 under model mismatch, develop loss functions that are closer to GOSPA, and investigate MT3's performance in the model-free context.
- **An Uncertainty-Aware Performance Measure for Multi-Object**

Tracking This letter proposes the first principled, hyperparameter free, uncertainty-aware MOT performance measure. It provides efficient algorithms for approximating the measure for several common MOT methods, and shows a connection to GOSPA. One interesting research direction left for future work is analyzing how to use this performance measure as a loss for training DL-based MOT trackers.

- **Deep Learning for Model-Based Multi-Object Tracking** This paper extends the preliminary analysis done in paper A. It proposes MT3v2, an extension of MT3 capable of predicting entire kinematic states along with uncertainty estimates. The tracker is compared to RFS-based approaches in challenging non-linear tasks, and an error analysis is provided. Interesting future directions are adding more flexibility to the predicted state densities, adapting the architecture to predict entire trajectories, and extending MT3v2 to work with image inputs.

- **Transformer-Based Multi-Object Smoothing with Decoupled Data Association and Smoothing** This study proposes D3AS, a DL tracker designed for the model-based multi-object smoothing task. The proposed model decouples the subtasks of data association and filtering and is able to predict entire sets of trajectories over the considered time window. Interesting future directions are to extend D3AS to also predict uncertainty estimates, to compare its performance under model-mismatch to RFS-based trackers, and to adapt it to the model-free MOT setting.

References

[1] Y.-C. Yoon, D. Y. Kim, Y.-M. Song, K. Yoon, and M. Jeon, "Online multiple pedestrians tracking using deep temporal appearance matching association," *Information Sciences*, vol. 561, pp. 326–351, 2021, ISSN: 0020-0255.

[2] A. Rangesh and M. M. Trivedi, "No blind spots: Full-surround multi-object tracking for autonomous vehicles using cameras and lidars," *IEEE Transactions on Intelligent Vehicles*, vol. 4, no. 4, pp. 588–599, 2019.

[3] A. Rangesh and M. M. Trivedi, "No blind spots: Full-surround multi-object tracking for autonomous vehicles using cameras and lidars," *IEEE Transactions on Intelligent Vehicles*, vol. 4, no. 4, pp. 588–599, 2019.

[4] E. Itskovits, A. Levine, E. Cohen, and A. Zaslaver, "A multi-animal tracker for studying complex behaviors," *BMC Biology*, vol. 15, no. 1, p. 29, Apr. 2017, ISSN: 1741-7007.

[5] C. Spampinato, Y.-H. Chen-Burger, G. Nadarajan, and R. B. Fisher, "Detecting, tracking and counting fish in low quality unconstrained underwater videos.," *VISAPP (2)*, vol. 2008, no. 514-519, p. 1, 2008.

[6] C. J. Harris, A. Bailey, and T. Dodd, "Multi-sensor data fusion in defence and aerospace," *The Aeronautical Journal (1968)*, vol. 102, no. 1015, pp. 229–244, 1998.

[7] P. Nillius, J. Sullivan, and S. Carlsson, "Multi-target tracking-linking identities using bayesian network inference," in *2006 IEEE Computer*

Society Conference on Computer Vision and Pattern Recognition (CVPR'06), IEEE, vol. 2, 2006, pp. 2187–2194.

[8] Á. F. García-Fernández, J. L. Williams, K. Granström, and L. Svensson, "Poisson multi-Bernoulli mixture filter: Direct derivation and implementation," *IEEE Transactions on Aerospace Electronic Systems*, vol. 54, no. 4, pp. 1883–1901, 2018.

[9] J. L. Williams, "Marginal multi-Bernoulli filters: RFS derivation of MHT, JIPDA, and association-based MeMber," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 51, no. 3, pp. 1664–1687, 2015.

[10] B. Vo, B. Vo, and H. G. Hoang, "An efficient implementation of the generalized labeled multi-Bernoulli filter," *IEEE Transactions on Signal Processing*, vol. 65, no. 8, pp. 1975–1987, 2017.

[11] K. Granström, L. Svensson, Y. Xia, J. Williams, and Á. F. García-Fernández, "Poisson multi-bernoulli mixtures for sets of trajectories," *arXiv preprint arXiv:1912.08718*, 2019.

[12] R. P. S. Mahler, *Statistical Multisource-Multitarget Information Fusion*. USA: Artech House, Inc., 2007, ISBN: 1596930926.

[13] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," in *NIPS*, 2017, pp. 5998–6008.

[14] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *European Conference on Computer Vision*, vol. 12346, Springer, 2020, pp. 213–229.

[15] S. Nagappa, D. E. Clark, and R. Mahler, "Incorporating track uncertainty into the OSPA metric," in *14th International Conference on Information Fusion*, IEEE, 2011, pp. 1–8.

[16] X. He, R. Tharmarasa, T. Kirubarajan, and T. Thayaparan, "A track quality based metric for evaluating performance of multitarget filters," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 49, no. 1, pp. 610–616, 2013.

[17] P. Dendorfer, A. Osep, A. Milan, *et al.*, "MOTchallenge: A benchmark for single-camera multiple target tracking," *International Journal of Computer Vision*, pp. 1–37, 2020.

[18] G. Ciaparrone, F. L. Sánchez, S. Tabik, L. Troiano, R. Tagliaferri, and F. Herrera, "Deep learning in video multi-object tracking: A survey," *Neurocomputing*, vol. 381, pp. 61–88, 2020.

[19] C.-Y. Chong, "An overview of machine learning methods for multiple target tracking," in *2021 IEEE 24th International Conference on Information Fusion (FUSION)*, 2021, pp. 1–9.

[20] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," in *NIPS*, 2015, pp. 91–99.

[21] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 779–788.

[22] F. Yu, W. Li, Q. Li, Y. Liu, X. Shi, and J. Yan, "Poi: Multiple object tracking with high performance detection and appearance feature," in *European Conference on Computer Vision*, Springer, 2016, pp. 36–42.

[23] J. Shen, D. Yu, L. Deng, and X. Dong, "Fast online tracking with detection refinement," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 1, pp. 162–173, 2018.

[24] J. Chen, H. Sheng, Y. Zhang, and Z. Xiong, "Enhancing detection model for multiple hypothesis tracking," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017, pp. 18–27.

[25] J. Shen, Z. Liang, J. Liu, H. Sun, L. Shao, and D. Tao, "Multiobject tracking by submodular optimization," *IEEE Transactions on Cybernetics*, vol. 49, no. 6, pp. 1990–2001, 2019.

[26] Y. Zhang, P. Sun, Y. Jiang, *et al.*, "Bytetrack: Multi-object tracking by associating every detection box," in *ECCV (22)*, ser. Lecture Notes in Computer Science, vol. 13682, Springer, 2022, pp. 1–21.

[27] A. Milan, S. H. Rezatofighi, A. Dick, I. Reid, and K. Schindler, "Online multi-target tracking using recurrent neural networks," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[28] H. Zhou, W. Ouyang, J. Cheng, X. Wang, and H. Li, "Deep continuous conditional random fields with asymmetric inter-object constraints for online multi-object tracking," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 29, no. 4, pp. 1011–1022, 2018.

[29] J. Yin, W. Wang, Q. Meng, R. Yang, and J. Shen, "A unified object motion and affinity model for online multi-object tracking," in *CVPR*, Computer Vision Foundation / IEEE, 2020, pp. 6767–6776.

[30] P. Bergmann, T. Meinhardt, and L. Leal-Taixe, "Tracking without bells and whistles," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 941–951.

[31] Y. Zhang, C. Wang, X. Wang, W. Zeng, and W. Liu, "Fairmot: On the fairness of detection and re-identification in multiple object tracking," *International Journal of Computer Vision*, pp. 1–19, 2021.

[32] P. Voigtlaender, M. Krause, A. Osep, *et al.*, "MOTS: Multi-object tracking and segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun. 2019.

[33] X. Weng, Y. Wang, Y. Man, and K. M. Kitani, "Gnn3dmot: Graph neural network for 3d multi-object tracking with 2d-3d multi-feature learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 6499–6508.

[34] J. Li, X. Gao, and T. Jiang, "Graph networks for multiple object tracking," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2020, pp. 719–728.

[35] T. Meinhardt, A. Kirillov, L. Leal-Taixé, and C. Feichtenhofer, "Trackformer: Multi-object tracking with transformers," *CoRR*, vol. abs/2101.02702, 2021.

[36] F. Zeng, B. Dong, Y. Zhang, T. Wang, X. Zhang, and Y. Wei, "MOTR: End-to-end Multiple-Object tracking with tRansformer," in *European Conference on Computer Vision (ECCV)*, 2022.

[37] P. Sun, Y. Jiang, R. Zhang, *et al.*, "Transtrack: Multiple-object tracking with transformer," *arXiv preprint arXiv:2012.15460*, 2020.

[38] J. Cai, M. Xu, W. Li, *et al.*, "Memot: Multi-object tracking with memory," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2022, pp. 8090–8100.

[39] Y. Liu, T. Bai, Y. Tian, *et al.*, "Segdq: Segmentation assisted multi-object tracking with dynamic query-based transformers," *Neurocomputing*, vol. 481, pp. 91–101, 2022.

[40] G. Maggiolino, A. Ahmad, J. Cao, and K. Kitani, "Deep OC-SORT: multi-pedestrian tracking by adaptive re-identification," *CoRR*, vol. abs/2302.11813, 2023.

[41] Y. Du, Y. Song, B. Yang, and Y. Zhao, "Strongsort: Make deepsort great again," *CoRR*, vol. abs/2202.13514, 2022.

[42] J. Hyun, M. Kang, D. Wee, and D. Yeung, "Detection recovery in online multi-object tracking with sparse graph tracker," in *WACV*, IEEE, 2023, pp. 4839–4848.

[43] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012, pp. 1106–1114.

[44] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *CVPR*, IEEE Computer Society, 2014, pp. 580–587.

[45] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *NIPS*, 2014, pp. 3104–3112.

[46] D. Silver, J. Schrittwieser, K. Simonyan, *et al.*, "Mastering the game of go without human knowledge," *Nat.*, vol. 550, no. 7676, pp. 354–359, 2017.

[47] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, `http://www.deeplearningbook.org`.

[48] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[49] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*. Springer, 2006, vol. 4.

[50] S. Kullback and R. A. Leibler, "On Information and Sufficiency," *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[51] M. Soltanolkotabi, A. Javanmard, and J. D. Lee, "Theoretical insights into the optimization landscape of over-parameterized shallow neural networks," *IEEE Transactions on Information Theory*, vol. 65, no. 2, pp. 742–769, 2018.

[52] S. Arora, S. S. Du, W. Hu, Z. Li, and R. Wang, "Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 97, PMLR, 2019, pp. 322–332.

[53] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, "Visualizing the loss landscape of neural nets," in *NeurIPS*, 2018, pp. 6391–6401.

[54] K. Kawaguchi and J. Huang, "Gradient descent finds global minima for generalizable deep neural networks of practical sizes," in *Allerton*, IEEE, 2019, pp. 92–99.

[55] S. S. Du, X. Zhai, B. Póczos, and A. Singh, "Gradient descent provably optimizes over-parameterized neural networks," in *ICLR (Poster)*, OpenReview.net, 2019.

[56] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *SIAM Rev.*, vol. 60, no. 2, pp. 223–311, 2018.

[57] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[58] T. Tieleman and G. Hinton, *Coursera: Neural networks for machine learning, lecture 6 - RMSProp*, Course Slides, Slide 27, 2012.

[59] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR (Poster)*, 2015.

[60] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *ICLR (Poster)*, OpenReview.net, 2019.

[61] S. Ruder, "An overview of gradient descent optimization algorithms," *CoRR*, vol. abs/1609.04747, 2016.

[62] D. Choi, C. J. Shallue, Z. Nado, J. Lee, C. J. Maddison, and G. E. Dahl, "On empirical comparisons of optimizers for deep learning," *arXiv preprint arXiv:1910.05446*, 2019.

[63] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of eugenics*, vol. 7, no. 2, pp. 179–188, 1936.

[64] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth, "Learnability and the vapnik-chervonenkis dimension," *Journal of the ACM (JACM)*, vol. 36, no. 4, pp. 929–965, 1989.

[65] V. N. Vapnik and A. Y. Chervonenkis, "On the uniform convergence of relative frequencies of events to their probabilities," *Measures of complexity: festschrift for alexey chervonenkis*, pp. 11–30, 2015.

[66] V. Vapnik, *Estimation of dependences based on empirical data*. Springer Science & Business Media, 2006.

[67] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth, "Learnability and the vapnik-chervonenkis dimension," *Journal of the ACM (JACM)*, vol. 36, no. 4, pp. 929–965, 1989.

[68] V. Vapnik, *The nature of statistical learning theory*. Springer science & business media, 1999.

[69] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[70] T. Brown, B. Mann, N. Ryder, *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[71] P. Nakkiran, G. Kaplun, Y. Bansal, T. Yang, B. Barak, and I. Sutskever, "Deep double descent: Where bigger models and more data hurt," in *ICLR*, OpenReview.net, 2020.

[72] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning (still) requires rethinking generalization," *Commun. ACM*, vol. 64, no. 3, pp. 107–115, 2021.

[73] T. Poggio, A. Banburski, and Q. Liao, "Theoretical issues in deep networks," *Proceedings of the National Academy of Sciences*, vol. 117, no. 48, pp. 30 039–30 045, 2020.

[74] Z. Allen-Zhu, Y. Li, and Z. Song, "A convergence theory for deep learning via over-parameterization," in *International Conference on Machine Learning*, PMLR, 2019, pp. 242–252.

[75] A. W. Senior, R. Evans, J. Jumper, *et al.*, "Improved protein structure prediction using potentials from deep learning," *Nature*, vol. 577, no. 7792, pp. 706–710, 2020.

[76] S. Karita, N. Chen, T. Hayashi, *et al.*, "A comparative study on transformer vs rnn in speech applications," in *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, 2019, pp. 449–456.

[77] N. Parmar, A. Vaswani, J. Uszkoreit, *et al.*, "Image transformer," in *International Conference on Machine Learning*, PMLR, 2018, pp. 4055–4064.

[78] L. J. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," in *Neural Information Processing Systems Deep Learning Symposium*, 2016.

[79] D. R. So, Q. V. Le, and C. Liang, "The evolved transformer," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 97, PMLR, 2019, pp. 5877–5886.

[80] H. Le, J. Pino, C. Wang, J. Gu, D. Schwab, and L. Besacier, "Dual-decoder transformer for joint automatic speech recognition and multilingual speech translation," in *COLING*, International Committee on Computational Linguistics, 2020, pp. 3520–3533.

[81] E. D. Kaplan and C. Hegarty, *Understanding GPS/GNSS: principles and applications*. Artech house, 2017.

[82] J. D. Murray, *Mathematical biology: I. An introduction*. Springer, 2002.

[83] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*. Springer, 2006, vol. 4.

[84] N. T. Bailey *et al.*, *The mathematical theory of infectious diseases and its applications*. Charles Griffin & Company Ltd, 5a Crendon Street, High Wycombe, Bucks HP13 6LE., 1975.

[85] S. Särkkä, *Bayesian filtering and smoothing*. Cambridge University Press, 2013.

[86] X. R. Li and V. P. Jilkov, "Survey of maneuvering target tracking. part i. dynamic models," *IEEE Transactions on aerospace and electronic systems*, vol. 39, no. 4, pp. 1333–1364, 2003.

[87] Á. F. García-Fernández, Y. Xia, K. Granström, L. Svensson, and J. L. Williams, "Gaussian implementation of the multi-bernoulli mixture filter," in *2019 22th International Conference on Information Fusion (FUSION)*, IEEE, 2019, pp. 1–8.

[88] K. Granström, M. Fatemi, and L. Svensson, "Poisson multi-bernoulli mixture conjugate prior for multiple extended target filtering," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 56, no. 1, pp. 208–225, 2019.

[89] Y. Xia, K. Granstrcom, L. Svensson, and Á. F. García-Fernández, "Performance evaluation of multi-bernoulli conjugate priors for multi-target filtering," in *2017 20th International Conference on Information Fusion (Fusion)*, IEEE, 2017, pp. 1–8.

[90] B.-T. Vo and B.-N. Vo, "Labeled random finite sets and multi-object conjugate priors," *IEEE Transactions on Signal Processing*, vol. 61, no. 13, pp. 3460–3475, 2013.

[91] B.-N. Vo, B.-T. Vo, and D. Phung, "Labeled random finite sets and the bayes multi-target tracking filter," *IEEE Transactions on Signal Processing*, vol. 62, no. 24, pp. 6554–6567, 2014.

[92] Á. F. García-Fernández, L. Svensson, and M. R. Morelande, "Multiple target tracking based on sets of trajectories," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 56, no. 3, pp. 1685–1707, 2019.

[93] J. R. Hoffman and R. P. Mahler, "Multitarget miss distance via optimal assignment," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 34, no. 3, pp. 327–336, 2004.

[94] K. Bernardin and R. Stiefelhagen, "Evaluating multiple object tracking performance: The clear mot metrics," *EURASIP Journal on Image and Video Processing*, vol. 2008, pp. 1–10, 2008.

[95] J. Luiten, A. Osep, P. Dendorfer, *et al.*, "Hota: A higher order metric for evaluating multi-object tracking," *International journal of computer vision*, vol. 129, no. 2, pp. 548–578, 2021.

[96] D. Schuhmacher, B.-T. Vo, and B.-N. Vo, "A consistent metric for performance evaluation of multi-object filters," *IEEE transactions on signal processing*, vol. 56, no. 8, pp. 3447–3457, 2008.

[97] A. S. Rahmathullah, Á. F. García-Fernández, and L. Svensson, "Generalized optimal sub-pattern assignment metric," in *20th International Conference on Information Fusion (Fusion)*, IEEE, 2017, pp. 1–8.

[98] Á. F. García-Fernández, A. S. Rahmathullah, and L. Svensson, "A metric on the space of finite sets of trajectories for evaluation of multi-target tracking algorithms," *IEEE Trans. Signal Process.*, vol. 68, pp. 3917–3928, 2020.