2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) | 978-1-6654-1714-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/IROS51168.2021.9636445

FloMo: Tractable Motion Prediction with Normalizing Flows

Christoph Schöller[1] and Alois Knoll[2]

Abstract— **The future motion of traffic participants is inherently uncertain. To plan safely, therefore, an autonomous agent must take into account multiple possible trajectory outcomes and prioritize them. Recently, this problem has been addressed with generative neural networks. However, most generative models either do not learn the true underlying trajectory distribution reliably, or do not allow predictions to be associated with likelihoods. In our work, we model motion prediction directly as a density estimation problem with a normalizing flow between a noise distribution and the future motion distribution. Our model, named FloMo, allows likelihoods to be computed in a single network pass and can be trained directly with maximum likelihood estimation. Furthermore, we propose a method to stabilize training flows on trajectory datasets and a new data augmentation transformation that improves the performance and generalization of our model. Our method achieves state-of-the-art performance on three popular prediction datasets, with a significant gap to most competing models.**

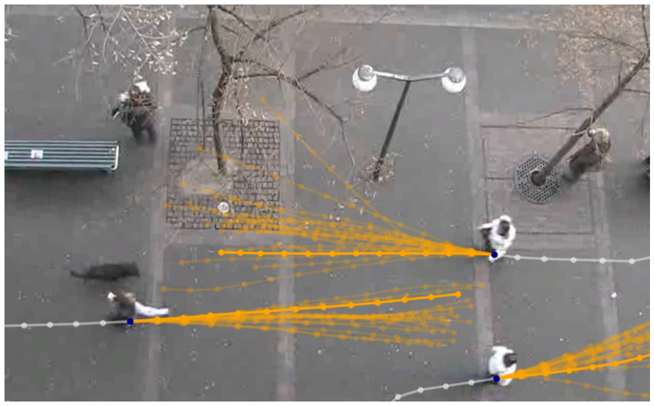


Fig. 1: Trajectory predictions of our model (orange). More likely trajectories are drawn more opaque. The distributions our model learned are highly multi-modal.

I. Introduction

For autonomous agents like vehicles and robots, it is essential to accurately predict the movement of other agents in their vicinity. Only with this ability collisions can be avoided and interactions become safe. However, trajectories can never be predicted with absolute certainty and multiple future outcomes must be taken into account.

To address this problem, research on generative models for motion prediction has recently gained attention. An ideal generative model is expressive and able to learn the true underlying trajectory distribution. Furthermore, it allows the assignment of a likelihood value to each prediction. The knowledge of how likely certain trajectories are is important to prioritize, because it is infeasible for an agent to take into account all possible future behaviors of surrounding agents.

Yet, most methods do not have all of these desirable properties. For example, Generative Adversarial Networks (GANs) have been used extensively for motion prediction [1], [2], [3], but suffer from mode collapse and are not guaranteed to learn the true distribution of the data [4], [5]. Variational Autoencoders (VAEs) are a popular type of generative models as well [6], [7], [8], [9] and approximate the true distribution with a lower bound. Unfortunately, likelihoods cannot be calculated directly with VAEs and must be estimated with computationally expensive Monte Carlo methods. Other contributions try to overcome the problem of missing likelihoods with the use of parametric density functions, most commonly normal distributions [10], [11]. However, this often requires unrealistic independence assumptions and provides only limited expressive power.

In this work, we propose a novel motion prediction model that addresses the aforementioned issues. In particular, our model FloMo is based on normalizing flows that we condition on observed motion histories. It is expressive and able to learn complex multi-modal distributions over future trajectories (see Fig. 1). With FloMo, trajectories can be efficiently sampled and likelihoods are computed in closed form. These tractable likelihoods allow us to train our model with maximum likelihood estimation, instead of a proxy loss. Because, as we show, trajectory data is prone to cause divergence of likelihoods during training, we apply a novel noise injection method that significantly stabilizes training and enables the use of our model's likelihoods in downstream tasks. Furthermore, we propose a new data augmentation transformation that helps our model to generalize better and improves its performance. We demonstrate with an extensive evaluation on three popular motion prediction datasets that our method achieves state-of-the-art performance and we show, both qualitatively and quantitatively, that the likelihoods our model produces are meaningful.

II. Related Work

Many classic approaches have been developed to make trajectory predictions [12], [13], [14], and are still relevant today [15].

Neural Networks. However, after successes on various other computer vision problems, neural networks have become popular for motion prediction as well. Alahi et al. [16] use Long Short-Term Memories (LSTMs) to predict

[1]Christoph Schöller is with fortiss GmbH, Research Institute of the Free State of Bavaria, Munich, Germany

[2]Alois Knoll is with the Technical University of Munich, Munich, Germany

pedestrian trajectories and share information between agents with a social hidden state pooling. Similarly, Pfeiffer et al. [17] provide an LSTM with an occupancy grid of static objects and an angular grid of surrounding pedestrians. But also Convolutional Neural Networks (CNNs) [18], spatio-temporal graphs [19] or state refinement modules [20] have been proposed to predict single trajectories.

Generative Models. To predict not only a single trajectory, but multiple possible outcomes, prediction methods based on generative neural networks have been developed. Sadeghian et al. [2] as well as Gupta et al. [1] utilize GANs that are provided with additional context information. To fight mode collapse, Amirian et al. [3] use an Info-GAN with an attention pooling module. The Trajectron++ model of Salzmann et al. [21] combines a conditional VAE, LSTMs and spatio-temporal graphs to produce multi-modal trajectory predictions. Inspired by BERT, Giuliari et al. [22] propose to use a transformer architecture for motion prediction. Xue et al. [9] propose the Scene Gated Social Graph that models the relations between pedestrians with a dynamic graph that is used to condition a VAE. Mohamed et al. [23] model social interactions with a spatio-temporal graph on which they apply graph convolutions and a temporal CNN to make predictions. Instead of directly predicting trajectories, Mangalam et al. [24] use a conditional VAE to first predict trajectory endpoints and a recursive social pooling module to make trajectory predictions. The prediction model of Pajouheshgar et al. [25] is fully convolutional and outputs a discrete probability distribution over image pixels.

Normalizing Flows. While originally developed for density estimation [26], normalizing flows have recently been applied to various data generation problems [27], [28]. In the area of motion prediction, normalizing flows have been rarely used. To generate trajectories for a planner, Agarwal et al. [7] sample from a conditional β-VAE [29] that uses a Neural Autoregressive Flow [30] as a flexible posterior. Bhattacharyya et al. [8] use a conditional Flow VAE with condition and posterior regularization to predict trajectories. In their recently published work [31], they use a Block Autoregressive Flow based on Haar wavelets to learn distributions for motion prediction and also adapted FlowWaveNet [27] for motion prediction. Ma et al. [32] recently showed how to find those trajectories sampled from affine flows that are both likely and diverse to make predictions.

The method we propose in this work is a flow-based generative model that can learn complex multimodal distributions. It allows tractable likelihood computation and can be trained directly with maximum likelihood estimation. Most existing generative models only possess some of these properties. In contrast to the flow-based prediction models proposed in concurrent works [31], [32], the flow we use is based on splines and hence is more flexible, which our results demonstrate. Furthermore, we propose a novel noise injection method that significantly stabilizes training and a data augmentation transformation that further improves our model's generalization and performance. In our extensive experiments we show that our model achieves state-of-the-art results on popular motion prediction datasets and that the likelihoods it produces are meaningful and can be used to modulate how concentrated our model's predictions are.

III. Problem and Notation

The motion of an agent can be defined as a finite sequence $\boldsymbol{\phi} = (\mathbf{p}^0, ..., \mathbf{p}^T)$ of positions $\mathbf{p}^t = (x^t, y^t)$ over discrete timesteps $t \in \{0, ..., T\}$. For predicting the future motion $\mathbf{x} = (\mathbf{p}^{t+1}, ..., \mathbf{p}^{t+n})$ of an agent, only a part $\mathbf{o} = (\mathbf{p}^0, ..., \mathbf{p}^t)$ of its past trajectory is observable. From the perspective of generative modeling, the goal is to learn the conditional distribution $p(\mathbf{x} \mid \mathbf{o})$. Future trajectories can then be predicted by sampling $\hat{\mathbf{x}} \sim p(\mathbf{x} \mid \mathbf{o})$.

One way to learn such a distribution is to use normalizing flows. Normalizing flows are probabilistic models that can learn complex data distributions by transforming noise samples $\mathbf{u}$ from a simple base distribution $p_u(\mathbf{u})$ into samples $\mathbf{x}$ from the target distribution:

$$\mathbf{x} = f(\mathbf{u}) \quad \text{where} \quad \mathbf{u} \sim p_u(\mathbf{u}). \tag{1}$$

By defining the transformation $f(\mathbf{u})$ such that it is invertibe and differentiable, the probability density of $\mathbf{x}$ can be obtained by a change of variables [33]:

$$p_x(\mathbf{x}) = p_u(\mathbf{u}) |\det J_f(\mathbf{u})|^{-1}. \tag{2}$$

Here $J_f(\mathbf{u})$ denotes the Jacobian matrix of the function $f(\mathbf{u})$. In the same manner, by the inverse function theorem, it is also possible to express $p_x(\mathbf{x})$ in terms of $\mathbf{x}$ and $J_{f^{-1}}$:

$$p_x(\mathbf{x}) = p_u(f^{-1}(\mathbf{x})) \left|\det J_{f^{-1}}(\mathbf{x})\right|. \tag{3}$$

For the base distribution, usually a standard normal is chosen and the invertible transformation is implemented by a neural network. To make the flow more flexible, several such transformations can be composed. It is important that the Jacobian determinant can be computed efficiently and, depending on the use case, the flow must be easy to invert. Furthermore, to represent complex distributions the transformations in the flow must be expressive.

IV. Method

The objective of our model is to learn the conditional motion distribution $p(\mathbf{x} \mid \mathbf{o})$, where $\mathbf{o}$ is an observed trajectory and $\mathbf{x}$ is the trajectory to predict (see Sec. III). We learn this distribution by utilizing normalizing flows. To then make a prediction, we sample from a standard normal base distribution $\mathbf{u} \sim \mathcal{N}(\mu = 0, \sigma = 1)$ and pass the sample through our model, which we condition with the encoded observed trajectory $\mathbf{o}$. The output of our model is a sampled trajectory prediction $\hat{\mathbf{x}}$. By evaluating Eq. 2, we can directly compute the likelihood of each sample in the same network pass. An overview of our architecture is given in Fig. 2. The main components of our model are a motion encoder and neural spline flows as proposed by Durkan et al. [34], consisting of conditional coupling layers [35] and monotonic spline transformations [36].

In this work we focus on the prediction of individual agents, because tests with integrating common interaction

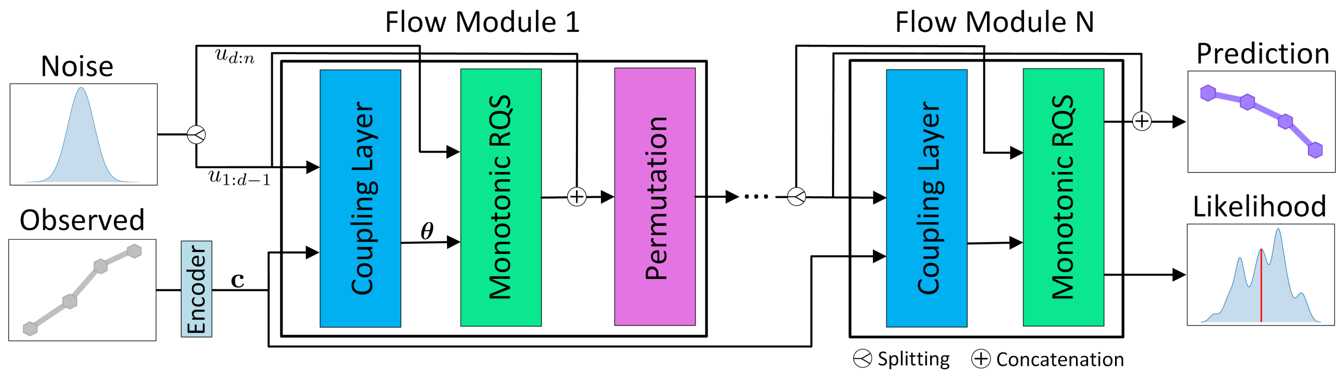


Fig. 2: Our model is composed of multiple flow modules, each containing a coupling layer for conditioning, a monotonic rational-quadratic spline (RQS) transformation and – except the last module – a permutation layer. It receives an encoded observed trajectory and a noise vector, and outputs a prediction sample along with its likelihood.

modules in our model's conditioning did not lead to relevant performance improvements. This is in line with the findings in [15], [1] and [22]. In the following sections, we explain each component of our model in detail, including how we prepare our data to achieve stable training, our objective function, and a novel trajectory augmentation transformation that we apply to increase generalization and performance.

A. *Motion Encoder*

The first module of our model is the motion encoder, which encodes the observed trajectory $\mathbf{o}$. Before we encode $\mathbf{o}$, we subtract from each position $\mathbf{p}^t \in \mathbf{o}$ its preceding position, i.e. $\mathbf{p}'^t = \mathbf{p}^t - \mathbf{p}^{t-1}$. This means instead of encoding absolute coordinates, we encode relative displacements, which has proven to be beneficial for motion prediction [37], [15]. From now on, we will denote the resulting relative observed trajectory as $\mathbf{o}'$ and its encoding as $\mathbf{c}$. We implement the encoder as a recurrent neural network with three Gated Recurrent Units (GRUs) [38] and a hidden state size of 16. Before we pass each displacement step to the encoder, we embed it with a linear layer in a 16 dimensional vector. The output of the last GRU is then passed through an Exponential Linear Unit (ELU) [39] and again linearly transformed, while keeping 16 output dimensions. We determined these hidden and embedding sizes empirically. Because the ELU function is non-zero everywhere, it helps to avoid dying neurons in the network recursion. The recurrent architecture of our encoder enables it to work with input trajectories of various lengths.

B. *Conditional Coupling Layer*

One way to design a normalizing flow is to modularize it into a transformation and a conditioner [30]. The conditioner takes the input $\mathbf{u}$ and parameterizes the transformation that in turn transforms $\mathbf{u}$ into $\mathbf{x}$. In our work, it is important that our flow is fast to evaluate both in the forward and inverse direction. For sampling trajectories, we must transform forward from $\mathbf{u}$ to $\mathbf{x}$, but during training we have to compute the likelihood of $\mathbf{x}$ in the inverse direction (Eq. 3). Furthermore, it would be desirable for our model to allow the computation of likelihoods for trajectories that an agent could possibly take, but that were not sampled. This also requires the inverse direction.

For the flow to be fast to invert, both the transformation and conditioner must be fast to invert. To achieve this for the conditioner, we use coupling layers [35], [34] to implement our flow. Coupling layers are just as fast to invert, as they are to compute forward. Our coupling layer computes the output $\mathbf{x}$ as follows ($\oplus$ denotes concatenation):

$$\begin{aligned} x_{1:d-1} &= u_{1:d-1} \\ \boldsymbol{\theta} &= \mathrm{NN}(u_{1:d-1} \oplus \mathbf{c}) \\ x_i &= \tau(u_i; \boldsymbol{\theta}_i) \text{ for i} \geq \text{d}. \end{aligned} \tag{4}$$

First, we split the input $\mathbf{u}$ in half and assign the first part $u_{1:d-1}$ directly to the output. Then we concatenate $u_{1:d-1}$ with trajectory encoding $\mathbf{c}$ (see Sec. IV-A) and feed it to the conditioner network that computes the parameters $\boldsymbol{\theta}$. Using $\boldsymbol{\theta}$ to parameterize the invertible transformation τ, we transform the second half $u_{d:n}$ of $\mathbf{u}$ element-wise to the remaining corresponding outputs. The resulting Jacobian matrix is lower triangular, and hence its determinant can be easily computed as the product of its diagonal elements [35]. By concatenating $\mathbf{c}$ to the conditioner input, we make our flow conditional on the observed trajectory, such that it learns the density $p(\mathbf{x} \,|\, \mathbf{o})$.

We implement the conditioner as a regular feed forward neural network with five hidden layers. Each layer has 32 neurons and is followed by an ELU activation. This configuration worked well empirically. Because half of the inputs are not transformed in a coupling layer, it is crucial to stack several such modules and randomly permute the input vectors between the modules. As permutations are volume-preserving, the Jacobian determinant of such a permutation layer is simply 1.

C. *Monotonic Spline Transforms*

Transformations used in normalizing flows must be expressive, invertible and differentiable. In motion prediction, expressive power is crucial to represent complex distributions and only fast invertibility allows the computation of likelihoods for query trajectories at runtime and short training times. However, most expressive flows, e.g. neural flows [30], cannot be inverted analytically and we have to resort to iterative methods like bisection search [33]. On the other hand, flows that are fast to invert often use simple transformations, e.g. affine or linear transformations, and hence are not very expressive.

However, recently Durkan et al. [34] proposed using monotonic rational-quadratic splines (RQS) [36] as flow transformations. In conjunction with coupling layers, this kind of flow becomes both expressive and fast to invert. The spline transformation described in the following corresponds to the function τ in Sec. IV-B.

The spline is defined by K different rational-quadratic functions that pass through $K+1$ knot coordinates $\left\{\left(x^k, y^k\right)\right\}_{k=0}^{K}$. These knots monotonically increase between $\left(x^0, y^0\right)=(-B,-B)$ and $\left(x^K, y^K\right)=(B, B)$. In accordance with Durkan et al., we assign the spline $K-1$ arbitrary positive derivatives $\left\{\delta^k\right\}_{k=1}^{K-1}$ for the intermediate knot connection points and set the boundary derivatives $\delta^0=\delta^K=1$ to match the linear 'tails' outside of the rational-quadratic support $[-B, B]$. This support is a hyper-parameter and is set manually. With these parameters, the spline is smooth and fully defined. The neural network that is parameterizing it can learn the knot positions and boundary derivatives during training.

The spline transformation is then applied element-wise, e.g. to a given scalar input x_{in}. If x_{in} is outside the support, the identity transformation is applied. Otherwise, the correct knot bin is determined first, and then

$$\begin{aligned} s_k &= \left(y^{k+1}-y^k\right) / \left(x^{k+1}-x^k\right) \\ \xi &= \left(x_{in}-x^k\right) / \left(x^{k+1}-x^k\right) \end{aligned} \tag{5}$$

are computed. After this, the forward transformation

$$\frac{\alpha^k(\xi)}{\beta^k(\xi)}=y^k+\frac{\left(y^{k+1}-y^k\right)\left[s^k \xi^2+\delta^k \xi(1-\xi)\right]}{s^k+\left[\delta^{k+1}+\delta^k-2 s^k\right] \xi(1-\xi)} \tag{6}$$

defined by the k^{th} bin can be evaluated. For the inverse transformation, derivatives to compute the Jacobian determinant and further details, we refer the reader to [34].

In practice, the knot coordinates and derivatives come from the conditioner network. Its output $\boldsymbol{\theta}_i=\left[\boldsymbol{\theta}_i^w, \boldsymbol{\theta}_i^h, \boldsymbol{\theta}_i^d\right]$ is simply partitioned into vectors of length K, K and $K-1$ for the knot widths and heights, as well as the knot derivatives. To compute the actual knot coordinates, $\boldsymbol{\theta}_i^w$ and $\boldsymbol{\theta}_i^h$ are softmax normalized, multiplied by $2B$ and their cumulative sums starting from $-B$ are computed.

Finally, the sampled output of our model (after the last spline transformation) represents the predicted trajectory as relative displacements. As for using relative coordinates in the motion encoding (see Sec. IV-A), this has proven to be beneficial for motion prediction [15], and it also limits the numeric range of the output. This is important to stay within the support $[-B, B]$ of the spline transformations. We denote this estimated relative displacements as $\hat{\mathbf{x}}'$. To convert it back to absolute coordinates, we compute the cumulative sum over all positions $\mathbf{p} \in \hat{\mathbf{x}}'$, starting from the last observed position $\mathbf{p}^t$ of $\mathbf{o}$. Because both the relative observed trajectory $\mathbf{o}'$ and the relative future trajectory $\mathbf{x}'$ can be unambiguously mapped back to $\mathbf{o}$ and $\mathbf{x}$, it holds that $p(\mathbf{x}' \mid \mathbf{o}')=p(\mathbf{x} \mid \mathbf{o})$. Hence we learn the target distribution.

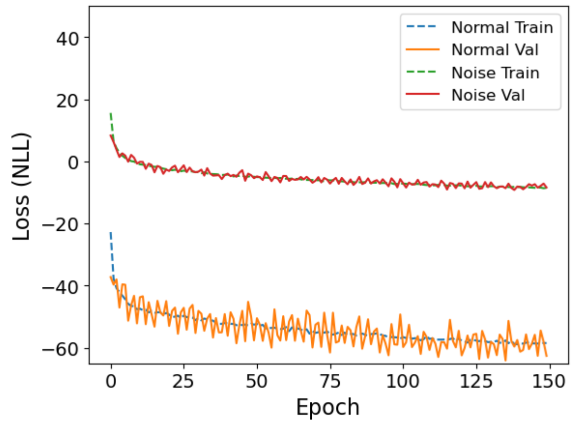


Fig. 3: Comparison between normally training our model and with our proposed noise injection. The training becomes more stable and likelihoods stay in a reasonable range.

Furthermore, like in [40], before making a prediction we rotate the trajectory of the target agent around $\mathbf{p}^t$, such that the last relative displacement $\mathbf{p}^t-\mathbf{p}^{t-1}$ is aligned with the vector $(1,0)$. After sampling from our model, we rotate the predicted trajectories back. This transformation simplifies the distribution our model must learn and makes it rotation invariant. Because rotations are volume preserving, we do not have to consider this in our flow's likelihood computation.

D. Preventing Manifolds

Whenever data is distributed such that it – or a subset of it – is residing on a lower-dimensional manifold, this leads to infinite likelihood spikes in the estimated density. Consider the two-dimensional example with joint density $p(x, y)$, where x is normally distributed and $y=x$. The distribution resides on a line and for $\int \int p(y \mid x) p(x) \mathrm{d} y \mathrm{d} x=1$ to hold, the likelihoods where y is defined must be infinite.

In practice, this problem also arises when certain dimensions in the dataset samples frequently take on equal values, or when one dimension frequently takes the same value. Because we predict relative displacements $\mathbf{x}'$ instead of absolute coordinates, this can happen if pedestrians stand still (values become zero), or if they move with constant velocity for multiple timesteps (values are equal). During training this can cause numerical instabilities, loss volatility and the overestimation of certain samples' likelihoods.

To mitigate this problem and inspired by [41], we define three hyper-parameters α, β and γ. While training, when transforming $\mathbf{x}'$ to $\mathbf{u}$ through the inverse of our flow, we augment $\mathbf{x}'$ before our first flow module as follows:

$$\begin{aligned} &\mathbf{x}''=\alpha \mathbf{x}' \\ &x_i''=x_i'+\epsilon_{\beta i} \quad \text{for all} \quad x_i'=0 \\ &x_i''=x_i'+\epsilon_{\gamma i} \quad \text{for all} \quad x_i' \neq 0. \end{aligned} \tag{7}$$

We sample noise vectors $\boldsymbol{\epsilon}_\beta$ and $\boldsymbol{\epsilon}_\gamma$ from zero-centered normal distributions with standard deviation β and γ, respectively. However, we only apply noise during the training phase and not at inference time. In the forward pass, we always compute $\mathbf{x}' = \alpha^{-1}\mathbf{x}''$ after our last flow module to normalize predicted trajectories. By adding the noise during training, we essentially lift data off potential manifolds. Generally speaking, we apply less noise to zero-valued dimensions and more to non-zero displacement vectors. Scaling $\mathbf{x}'$ with α allows us to inject more noise, while controlling the impact of the noise on the trajectory.

The lower training curves in Fig. 3 show how the loss of our model behaves when trained normally, without our noise injection. The loss is very volatile, especially for the validation dataset, and the likelihoods produced by our model are very large. Because we use the negative log likelihood loss (see Sec. IV-E), these large likelihoods lead to an artificially low overall loss. However, empirically these inflated likelihoods do not correlate with better prediction performance and are meaningless. The upper curves in Fig. 3 show how the training behaves with our noise injection. The magnitudes of the likelihoods are significantly reduced, because samples that originally lied on manifolds get smaller likelihood values assigned. Hence, they stop to dominate the training and this reduces the volatility of our validation loss. With our method, we experienced more reliable convergence during our experiments. Furthermore, it helps to avoid numerical problems during training and makes the model's likelihoods easier to use in downstream tasks (e.g. those that require normalization with softmax).

E. Objective Function

Because our model makes it easy to compute likelihoods for training examples (see Eq. 3), we simply train it with maximum likelihood estimation. In particular, we minimize the negative log likelihood

$$\mathrm{NLL} = -\frac{1}{N}\sum_{i=1}^{N} \log(p(\mathbf{x}_i \mid \mathbf{o}_i)). \tag{8}$$

F. Trajectory Augmentation

To increase the diversity of our data, we augment trajectories by randomly scaling them. In particular, for each trajectory we sample a scalar in range $[s_{\min}, s_{\max}]$ from a truncated normal distribution. Before multiplying the trajectory element-wise with the scalar, we first center the trajectory by subtracting its mean position to avoid translating it with the scaling, and then move it back. Scaling a trajectory does not influence its direction and motion pattern, but simulates varying movement speeds. It is crucial to stay within realistic limits by applying this transformation and the correct choice for the sampling interval depends on the used data.

V. Experiments

We evaluate our model with the publicly available ETH[13], UCY [42] and Stanford Drone [43] motion datasets. All datasets are based on real-world video recordings and contain complex motion patterns. The ETH/UCY datasets are evaluated jointly and focus on pedestrians that were recorded in city centers and at university campuses. They cover a total of five distinct scenes with four unique environments and 1950 individual pedestrians. The larger Stanford Drone dataset contains 10300 individual traffic participants, it covers roads and besides pedestrians it includes also other agent types like cyclists and vehicles. All datasets are heavily used in the motion prediction domain [16], [15], [21], [31], [2], [1].

We follow for all datasets the most common evaluation regimes. For the ETH/UCY datasets we always train on four scenes and evaluate on the remaining one. We slice each trajectory with a step-size of one into sequences of length 20, of which 8 timesteps are observed and 12 must be predicted. This corresponds to an observation window of $3.2\,\mathrm{s}$ and a prediction of $4.8\,\mathrm{s}$. For the Stanford Drone dataset we randomly split into training and testset but ensure that both sets do not contain parts of the same video sequences. We observe for 20 timesteps and predict the next 40 timesteps, which corresponds to $2\,\mathrm{s}$ and $4\,\mathrm{s}$, respectively. For comparability, we follow [6], [31] and scale the dataset trajectories by a factor of $1/5$.

For training our model, we only take into account trajectories of full length, because padding would cause issues as described in Sec. IV-D. However, in our evaluation we use all trajectories that have a length of at least 10 timesteps for ETH/UCY, and at least 22 timesteps for Stanford Drone, i.e. at least two timesteps to predict. Note that we also compare tractable models only based on displacement errors and not on log likelihoods. While our model's likelihoods are meaningful, as we show in Sec. V-B, the overall log likelihood for trajectory datasets is largely dominated by manifold artifacts and hence not ideal for comparison.

Training. We trained our model with the Adam Optimizer [44], learning rate 0.001, and batch size 128 for 150 epochs. We randomly split a 10% validation set for ETH/UCY and a 5% validation set for Stanford Drone from each training set to detect overfitting. Furthermore, we define the support for each spline flow as $B = 15$ and use 8 knot points. For ETH/UCY we set $\alpha = 10$, $\beta = 0.2$, $\gamma = 0.02$ and for Stanford Drone $\alpha = 3$, $\beta = 0.002$, $\gamma = 0.002$. In our scaling transformation we set $\mu = 1$ for all datasets, but for ETH/UCY $\sigma = 0.5$, $s_{\min} = 0.3$, $s_{\max} = 1.7$ and for Stanford Drone $\sigma = 0.2$, $s_{min} = 0.8$, $s_{max} = 1.2$. In total, we stack 10 flow layers in our model. All hyper-parameters described were determined empirically.

Metrics. As proposed by [1], we allow each model to predict multiple samples. For the ETH/UCY datasets we report errors in meters, and for the Stanford Drone dataset in pixels. We evaluate with the following metrics:

- *Minimum Average Displacement Error (minADE)* — Error of the sample with the smallest average L2 distance between all corresponding positions in the ground truth and the predicted trajectory.
- *Minimum Final Displacement Error (minFDE)* — Error

Model	ETH-Uni	Hotel	UCY-Uni	Zara1	Zara2	AVG
SGSG [9]	0.54 / 1.07	0.24 / 0.45	0.57 / 1.19	0.35 / 0.79	0.28 / 0.59	0.40 / 0.82
S-STGCNN [23]	0.64 / 1.11	0.49 / 0.85	0.44 / 0.79	0.34 / 0.53	0.30 / 0.48	0.44 / 0.75
S-GAN [1]	0.59 / 1.04	0.38 / 0.80	0.27 / 0.49	0.18 / 0.33	0.19 / 0.35	0.32 / 0.60
CVM-S [15]	0.44 / 0.81	0.20 / 0.35	0.34 / 0.71	0.25 / 0.49	0.22 / 0.45	0.29 / 0.56
Trajectron++ [21]	0.39 / 0.83	**0.12** / **0.21**	**0.20** / **0.44**	**0.15** / **0.33**	**0.11** / **0.25**	**0.19** / 0.41
TF$_q$ [22]	0.61 / 1.12	0.18 / 0.30	0.35 / 0.65	0.22 / 0.38	0.17 / 0.32	0.31 / 0.55
FloMo (ours)	**0.32** / **0.52**	0.15 / 0.22	0.25 / 0.46	0.20 / 0.36	0.17 / 0.31	0.22 / **0.37**

TABLE I: Displacement errors for scenes in the ETH/UCY datasets and on average. We compare our model to six state-of-the-art models. Each model predicted 20 trajectory samples and the errors are shown as minADE / minFDE.

Model	@1 s	@2 s	@3 s	@4 s
STCNN [25]	1.20	2.10	3.30	4.60
FlowWaveNet [27][31]	0.70	1.50	2.40	3.50
HBA-Flow [31]	0.70	1.40	2.30	3.20
FloMo (ours)	**0.27**	**0.56**	**0.90**	**1.27**

TABLE II: Errors for the Stanford Drone dataset, evaluated with a five-fold cross-validation, the Oracle Top 10% metric and 50 predicted trajectories. All models are tractable and allow exact likelihood computation.

Model	minADE	minFDE
SocialGAN [1]	27.23	41.44
SoPhie [2]	16.27	29.38
CF-VAE [8]	12.60	22.30
HBA-Flow [31]	10.80	19.80
PECNet [24]	9.96	15.88
FloMo (ours)	**2.60**	**4.43**

TABLE III: Evaluation results for Stanford Drone with a single dataset split, 20 predicted trajectories and the minADE / minFDE metrics. Here we also include intractable models.

of the sample with the smallest L2 distance between the last position in the ground truth and the last position in the predicted trajectory.

- *Oracle Top 10%* — Average error of the top 10% best predicted trajectories at different timesteps. It has been shown that this measure is robust to random guessing and simply increasing the number of drawn samples does not affect it [8].

Baselines. We compare our model with a variety of state-of-the-art prediction models. Except the CVM-S [15], all other models are based on neural networks. S-STGCNN [23], SGSG [9] and Trajectron++ [21] utilize neural networks in combination with graphs. TF$_q$ [22] is based on the transformer architecture. S-GAN [1], SoPhie [2] are GANs. STCNN [25], FloWaveNet and HBA-Flow are exact inference models and the latter two based on normalizing flows. Besides the Trajectron++, also CF-VAE[8] and PECNet [24] use a conditional VAE as their core network.

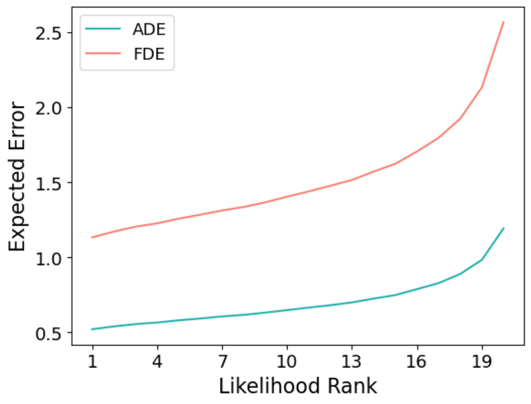
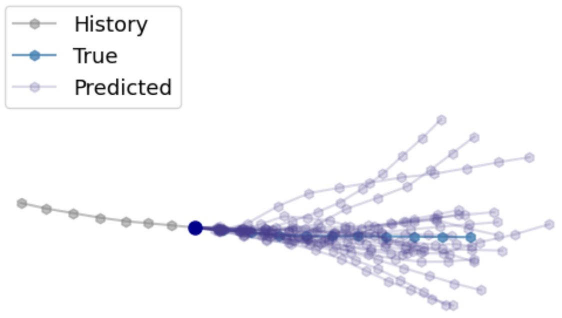


Fig. 4: Relationship between our model's prediction errors and associated likelihood ranks for the ETH/UCY datasets.

A. Displacement Errors

For the ETH/UCY datasets, we compare our model with state of the art in Tab. I. Following the standard protocol, each model was allowed to predict 20 trajectory samples in this evaluation. Except for the Trajectron++, our model significantly outperforms all other models on average errors, both in terms of minADE and minFDE. Compared to the Trajectron++, our model performs better on the ETH-Uni scene, while on the other Scenes the Trajectron++ achieves lower errors, especially for minADE. However, for the minFDE both models perform close on all scenes except ETH-Uni and Zara2. In total, the Trajectron++ achieves lower errors averaged over the whole trajectories with a minADE of 0.19, but FloMo performs better on the endpoint prediction where it achieves a minFDE of 0.37. Hence, the prediction performance of both models can be considered as approximately equivalent. However, unlike the Trajectron++ our model is tractable and allows direct likelihood computation. The close performance of both models could indicate that the noise floor for ETH/UCY predictions is approached.

On the Stanford Drone dataset we evaluated with two different protocols. For the results in Tab. II we performed a five-fold cross-validation and let each model predict 50 samples. Then we evaluated with the Oracle Top 10% metric that we described earlier. All models in this evaluation allow tractable likelihood computation, and the concurrently proposed models HBA-Flow and FlowWaveNet (applied to



(a) Regular predictions

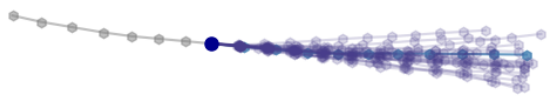

(b) Top-k predictions

Fig. 5: Comparison of our model's regular predictions with top-k predictions for the same sample.

motion prediction by [31]) are also based on normalizing flows. The displacement errors are evaluated at four different timesteps. Our model significantly outperforms all other models at each timestep with an improvement of 60% at 4 s over the second best model HBA-Flow. This results show that our model captures the true underlying distribution better than the other tractable models.

In Tab. III we performed a second evaluation on the Stanford Drone dataset with a single dataset split, 20 trajectory predictions, and the minADE and minFDE metrics. In this case we also compare to intractable models. The results on this experiment confirm those of the previous experiment. Our model significantly outperforms all compared models, with a margin of 74% in minADE and 72% in minFDE compared to the second best model PECNet.

B. Likelihoods

To verify that the likelihoods our model provides are relevant, we rank each of the 20 trajectory samples generated by our model for the ETH/UCY datasets in descending order by likelihood. Then we compute the expected ADE and FDE for each likelihood ranking position across all testsets. As for the evaluation in the previous section, for each testset evaluation we use the FloMo trained on the remaining scenes. Fig. 4 shows graphs of how the expected errors change with likelihood ranking. As expected, a higher likelihood (lower rank) corresponds to lower errors for both ADE and FDE. This proves that the likelihoods computed by our model are meaningful and can be used for decision making.

To qualitatively demonstrate how likelihoods relate to the predicted trajectories, in Fig. 5a we show 20 regularly predicted trajectories and in Fig. 5b a top-k prediction for the same example. For the top-k prediction we sample 100 trajectory candidates and only keep the 20 most likely ones. The regular predictions are much more spread out. Our model predicts sudden turns, acceleration, or deceleration. The top-k predictions are more concentrated around the true and most likely outcome of the pedestrian's movement. Furthermore, the predicted velocities are more regular. This results demonstrate that an autonomous agent can utilize the likelihoods our model provides to decide which predictions it should prioritize in its planning.

Method	ETH/UCY	Stanford Drone
No Scaling	0.27 / 0.46	2.92 / 5.02
Scaling	**0.22 / 0.37**	**2.60 / 4.43**

TABLE IV: Ablation results for our scaling transformation.

C. Ablation

To understand the impact of our scaling transformation on our model's performance, we conducted an ablation study. The results of this study for the ETH/UCY and the Stanford Drone datasets are shown in Tab. IV. Applying our transformation improved our model's performance on all datasets. By simulating varying movement speeds and thus diversifying the training data, our model learned to generalize better. We also analyzed our noise injection and found that it does not have a significant impact on average prediction performance. Most likely because the inflated density points are sparsely distributed. However, the injection's stabilizing effect on the training of our model, along with its numerical and practical advantages, make it a useful tool for training flows for motion prediction.

VI. Conclusion

In this work we proposed a motion prediction model based on spline flows that is able to learn a distribution over the future motion of agents. It makes it possible to directly compute likelihoods that are necessary for autonomous agents to prioritize predictions. Because training on trajectory data directly causes loss volatility and numerical instabilities, we proposed a method of injecting noise, such that training is stabilized, but the motion information in the trajectories is preserved. Furthermore, we suggested an augmentation transformation that improves our model's generalization.

To evaluate our model we conducted extensive experiments, in which we showed that our model achieves state-of-the-art performance in terms of displacement errors. We also showed at a quantitative and qualitative level that the likelihoods our model provides are meaningful and can be used for decision making in autonomous agents. With an ablation study, we ensured that our data augmentation transformation contributes positively to our model's performance.

Acknowledgment

This research was funded by the Federal Ministry of Transport and Digital Infrastructure of Germany in the project Providentia++.

References

[1] A. Gupta, J. Johnson, L. Fei-Fei, S. Savarese, and A. Alahi, “Social gan: Socially acceptable trajectories with generative adversarial networks,” in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[2] A. Sadeghian, V. Kosaraju, A. Sadeghian, N. Hirose, and S. Savarese, “Sophie: An attentive gan for predicting paths compliant to social and physical constraints,” *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[3] J. Amirian, J.-B. Hayet, and J. Pettre, “Social ways: Learning multi-modal distributions of pedestrian trajectories with gans,” in *Conference on Computer Vision and Pattern Recognition Workshops (CVPR Workshops)*, 2019.

[4] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, “Improved techniques for training gans,” in *Conference on Neural Information Processing Systems (NeurIPS)*, 2016.

[5] T. Che, Y. Li, A. P. Jacob, Y. Bengio, and W. Li, “Mode regularized generative adversarial networks,” in *International Conference on Learning Representations (ICLR)*, 2017.

[6] N. Lee, W. Choi, P. Vernaza, C. B. Choy, P. H. Torr, and M. Chandraker, “Desire: Distant future prediction in dynamic scenes with interacting agents,” in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[7] S. Agarwal, H. Sikchi, C. Gulino, and E. Wilkinson, “Imitative planning using conditional normalizing flow,” *arXiv preprint arXiv:2007.16162*, 2020.

[8] A. Bhattacharyya, M. Hanselmann, M. Fritz, B. Schiele, and C.-N. Straehle, “Conditional flow variational autoencoders for structured sequence prediction,” *Conference on Neural Information Processing Systems Workshops (NeurIPS Workshops)*, 2019.

[9] H. Xue, D. Q. Huynh, and M. Reynolds, “Scene gated social graph: Pedestrian trajectory prediction based on dynamic social graphs and scene constraints,” *arXiv preprint arXiv:2010.05507*, 2020.

[10] S. Casas, C. Gulino, R. Liao, and R. Urtasun, “Spatially-aware graph neural networks for relational behavior forecasting from sensor data,” *International Conference on Robotics and Automation (ICRA)*, 2020.

[11] C. Luo, L. Sun, D. Dabiri, and A. Yuille, “Probabilistic multi-modal trajectory prediction with lane attention for autonomous vehicles,” *International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[12] D. Helbing and P. Molnar, “Social force model for pedestrian dynamics,” *Physical Review E*, 1995.

[13] S. Pellegrini, A. Ess, K. Schindler, and L. Van Gool, “You’ll never walk alone: Modeling social behavior for multi-target tracking,” in *International Conference on Computer Vision (ICCV)*, 2009.

[14] S. Pellegrini, A. Ess, and L. Van Gool, “Predicting pedestrian trajectories,” in *Visual Analysis of Humans*, 2011.

[15] C. Schöller, V. Aravantinos, F. Lay, and A. Knoll, “What the constant velocity model can teach us about pedestrian motion prediction,” *Robotics and Automation Letters (RA-L)*, 2020.

[16] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, “Social lstm: Human trajectory prediction in crowded spaces,” in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[17] M. Pfeiffer, G. Paolo, H. Sommer, J. Nieto, R. Siegwart, and C. Cadena, “A data-driven model for interaction-aware pedestrian motion prediction in object cluttered environments,” in *International Conference on Robotics and Automation (ICRA)*, 2018.

[18] N. Nikhil and B. Tran Morris, “Convolutional neural network for trajectory prediction,” in *European Conference on Computer Vision (ECCV)*, 2018.

[19] A. Vemula, K. Muelling, and J. Oh, “Social attention: Modeling attention in human crowds,” in *International Conference on Robotics and Automation (ICRA)*, 2018.

[20] P. Zhang, W. Ouyang, P. Zhang, J. Xue, and N. Zheng, “Sr-lstm: State refinement for lstm towards pedestrian trajectory prediction,” in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[21] T. Salzmann, B. Ivanovic, P. Chakravarty, and M. Pavone, “Trajectron++: Dynamically-feasible trajectory forecasting with heterogeneous data,” *European Conference on Computer Vision (ECCV)*, 2020.

[22] F. Giuliari, I. Hasan, M. Cristani, and F. Galasso, “Transformer networks for trajectory forecasting,” *International Conference on Pattern Recognition (ICPR)*, 2020.

[23] A. Mohamed, K. Qian, M. Elhoseiny, and C. Claudel, “Social-stgcnn: A social spatio-temporal graph convolutional neural network for human trajectory prediction,” in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[24] K. Mangalam, H. Girase, S. Agarwal, K.-H. Lee, E. Adeli, J. Malik, and A. Gaidon, “It is not the journey but the destination: Endpoint conditioned trajectory prediction,” in *European Conference on Computer Vision (ECCV)*, 2020.

[25] E. Pajouheshgar and C. H. Lampert, “Back to square one: probabilistic trajectory forecasting without bells and whistles,” *Conference on Neural Information Processing Systems Workshops (NeurIPS Workshops)*, 2018.

[26] E. G. Tabak and C. V. Turner, “A family of nonparametric density estimation algorithms,” *Communications on Pure and Applied Mathematics*, 2013.

[27] S. Kim, S.-G. Lee, J. Song, J. Kim, and S. Yoon, “Flowavenet: A generative flow for raw audio,” *International Conference on Machine Learning (ICML)*, 2018.

[28] D. P. Kingma and P. Dhariwal, “Glow: Generative flow with invertible 1x1 convolutions,” in *Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

[29] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, “beta-vae: Learning basic visual concepts with a constrained variational framework,” in *International Conference on Learning Representations (ICLR)*, 2017.

[30] C.-W. Huang, D. Krueger, A. Lacoste, and A. Courville, “Neural autoregressive flows,” *International Conference on Machine Learning (ICML)*, 2018.

[31] A. Bhattacharyya, C.-N. Straehle, M. Fritz, and B. Schiele, “Haar wavelet based block autoregressive flows for trajectories,” *German Conference on Pattern Recognition (GCPR)*, 2020.

[32] Y. J. Ma, J. P. Inala, D. Jayaraman, and O. Bastani, “Diverse sampling for normalizing flow based trajectory forecasting,” *arXiv preprint arXiv:2011.15084*, 2020.

[33] G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan, “Normalizing flows for probabilistic modeling and inference,” *Journal of Machine Learning Research (JMLR)*, 2019.

[34] C. Durkan, A. Bekasov, I. Murray, and G. Papamakarios, “Neural spline flows,” in *Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

[35] L. Dinh, D. Krueger, and Y. Bengio, “Nice: Non-linear independent components estimation,” *International Conference on Learning Representations Workshops (ICLR Workshops)*, 2015.

[36] J. Gregory and R. Delbourgo, “Piecewise rational quadratic interpolation to monotonic data,” *IMA Journal of Numerical Analysis (IMAJNA)*, 1982.

[37] S. Becker, R. Hug, W. Hubner, and M. Arens, “Red: A simple but effective baseline predictor for the trajnet benchmark,” in *European Conference on Computer Vision Workshops (ECCV Workshops)*, 2018.

[38] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using rnn encoder-decoder for statistical machine translation,” *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[39] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, “Fast and accurate deep network learning by exponential linear units (elus),” *International Conference on Learning Representations (ICLR)*, 2016.

[40] M.-F. Chang, J. Lambert, P. Sangkloy, J. Singh, S. Bak, A. Hartnett, D. Wang, P. Carr, S. Lucey, D. Ramanan, *et al.*, “Argoverse: 3d tracking and forecasting with rich maps,” in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[41] H. Kim, H. Lee, W. H. Kang, J. Y. Lee, and N. S. Kim, “Softflow: Probabilistic framework for normalizing flow on manifolds,” *Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[42] A. Lerner, Y. Chrysanthou, and D. Lischinski, “Crowds by example,” in *Computer Graphics Forum*, 2007.

[43] A. Robicquet, A. Sadeghian, A. Alahi, and S. Savarese, “Learning social etiquette: Human trajectory understanding in crowded scenes,” in *European Conference on Computer Vision (ECCV)*, 2016.

[44] D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *International Conference for Learning Representations (ICLR)*, 2015.