Technical University of Munich
Guided Research

Winter Semester 2020/2021

Normalizing Flows for Density Estimation on Graphs

Supervisor: Prof. Dr. Stephan Günnemann
Advisor: Marin Biloš
Student: Andreea-Alexandra Muşat

Abstract

Spatial point processes are usually modelled on unrestricted domains. However, it is often the case that the events lie on a graph embedded in the two-dimensional plane, for example in the case of traffic events. We propose several methods based on normalizing flows for modelling the intensity of a spatial point process whose domain is a graph. In our approaches, we leverage both the edge proximity and possible point interactions. We also experiment with temporal models, predicting the likelihood given span of previous point realizations.

1 Introduction

Analysis of real life datasets consisting of spatial locations of points is required in sciences such as geoscience, crime research, epidemiology and ecology [1] and has relevant and challenging applications, such as discovering patterns in crimes in a neighbourhood [2], modelling earthquakes locations [3], observing trends in the tree density in an area [4] etc. However, in some situations, the data actually lives on a graph embedded in the 2D plane, for example in the case of street crimes, traffic accidents [5] or spines on dendrite networks [6].

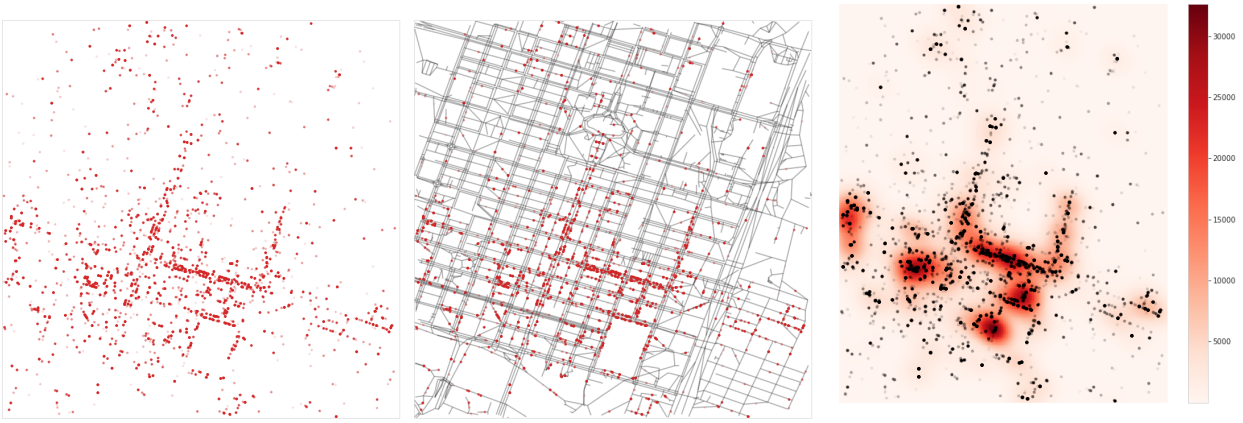

Figure 1: **Left:** A 2D spatial point process representing check-in data. **Middle:** The true data domain of the spatial point process is not the entire 2D plane, but it is restricted to a street network. **Right:** Assigning density on the whole plane results in non-zero density in areas where events might not even be possible.

However, in this case, modelling the density on the whole plane implies that density can also be assigned to locations where it is apriori known that events cannot happen (Fig. 1). Motivated by this issue, our goal is to develop a method for both sampling and density estimation for spatial points on graphs.

This report is structured as follows: in the next section we will review several tools necessary for the development of the proposed method. We start by introducing point processes in Section 2.1 and density estimation tools (normalizing flows in Section 2.2, neural spline flows in Section 2.3 and variational auto-encoders in Section 2.4). Lastly, we review graph convolutional networks in Section 2.5. After introducing the notation, the proposed method is detailed in Section 3 and related work in Section 4. The experimental setup and results follow in Section 5. Final conclusions and possible future work directions can be found in Section 6.

2 Background

2.1 Point Processes

A finite simple point process on set $\mathcal{B}$ represents a random variables whose values are finite sets of interchangeable, non-overlapping points $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$. In order to sample a new realization of a point process, we need to first sample an integer, n, representing the number of constituent points, then sample the points $\boldsymbol{x_i}$. Using this generative perspective, we can write the likelihood function as follows:

$$p(\boldsymbol{X}) = n! p(n) p(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n) \tag{1}$$

where $\boldsymbol{x}_i \in \mathcal{B}, \forall i$ and $p(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n)$ needs to be a permutation invariant probability density function.

In general, point processes are modelled using an intensity function $\lambda(\boldsymbol{x})$, which represents the expected number of points per unit length. In this framework, we can compute the total expected number of events (or total intensity) as $\Lambda = \int_{\mathcal{B}} \lambda(\boldsymbol{x}) d\boldsymbol{x}$ and thus express $p(\boldsymbol{x}) = \frac{\lambda(\boldsymbol{x})}{\Lambda}$. In an inhomogeneous Poisson process, n is Poisson distributed with rate Λ and in order to achieve invariance in Eq. 1, the distribution is factorized:

$$p(\boldsymbol{X}) = n! \frac{\Lambda^n \exp(-\Lambda)}{n!} \prod_{i=1}^{n} p(\boldsymbol{x}_i) = \exp\Big(- \int_{\mathcal{B}} \lambda(\boldsymbol{x}) d\boldsymbol{x} \Big) \prod_{i=1}^{n} \lambda(\boldsymbol{x}_i) \tag{2}$$

2.2 Normalizing Flows

Normalizing flows [7] represent a method for modelling complex continuous distributions, having both sampling and density estimation capabilities. The flow transforms a simple base distribution p_u, usually normal or uniform, into a target, more flexible distribution p_x by applying a transformation T_ϕ. More precisely:

$$\boldsymbol{u} \sim p_u(\boldsymbol{u}), \boldsymbol{x} = T_\phi(\boldsymbol{u}) \implies p_x(\boldsymbol{x}) \overset{(*)}{=} p_u(\boldsymbol{u}) \big| \det(J_{T_\phi}(\boldsymbol{u})) \big|^{-1} \overset{(\circ)}{=} p_u(T_\phi^{-1}(\boldsymbol{x})) \big| \det J_{T_\phi{}^{-1}}(\boldsymbol{x}) \big| \tag{3}$$

where $\boldsymbol{x}, \boldsymbol{u} \in \mathbb{R}^D$ are samples from the target space and base space, respectively; ϕ represents the parameters of the transformation, which are learned; J_{T_ϕ} denotes the Jacobian of T_ϕ, $(*)$ is obtained using the change of variables formula and $(\circ)$ is a result of the inverse function theorem, i.e. $J_{T_\phi{}^{-1}}(\boldsymbol{x}) = \big(J_{T_\phi}(\boldsymbol{u})\big)^{-1}$.

As it can be seen in Eq. 3, we require T_ϕ to be invertible and both T_ϕ and ${T_\phi}^{-1}$ to be differentiable and, as compositions of such functions will still be invertible and diferentiable, in practice T_ϕ can be a sequence of transformations $T_\phi = T_\phi^{(1)} \circ T_\phi^{(2)} \circ \cdots \circ T_\phi^{(n)}$. As the general computation of the Jacobian determinant has $O(D^3)$ complexity, T is usually chosen such that the resulting Jacobian is triangular, for example an autoregressive function, such that the complexity is reduced to to $O(D)$. Furthermore, if the task requires both estimating densities in the target space and sampling as well, T_ϕ needs to be easily invertible.

Under certain conditions [7], it can be shown that a normalizing flow based model can express any distribution, regardless of the choice of base distribution. Given a dataset $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^N$, the model is trained by maximizing its likelihood $p(\mathcal{D}) = \prod_{i=1}^N p_x(\boldsymbol{x}_i)$ with respect to the parameters of the transformations. Alternatively, this can be interpreted as minimizing the KL divergence between the true target distribution $p_x^*(\boldsymbol{x})$ and the flow-predicted one, $p_x(\boldsymbol{x})$, i.e. $\mathrm{KL}(p_x^*(\boldsymbol{x}) || p_x(\boldsymbol{x}))$. Most often, the target density is not available, so the expectation term in the KL divergence is approximated using a Monte Carlo method.

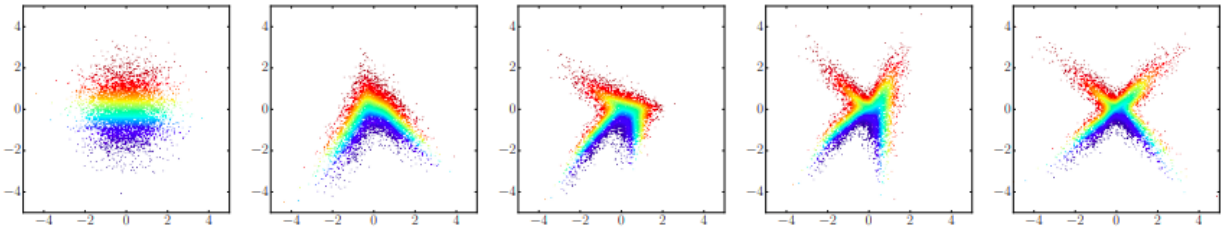

Figure 2: Transforming a centered Gaussian distribution (left) into a complex cross-shaped distribution (right) using a normalizing flow consisting of 4 stacked functions. Source: [7]

2.3 Neural Spline Flows

[8] introduced neural spline flows, which originally models an increasing function from $[-B, B]$ to $[-B, B]$ by using K different *monotonic* rational quadratic (RQ) functions (Fig. 3). The RQ functions are completely specified by a set of $K+1$ *knots*, through which it passes, $\{(x^{(k)}, y^{(k)})\}_{k=0}^{K}$ with $x^{(0)} = y^{(0)} = -B$, $x^{(K)} = y^{(K)} = B$ and $x^{(k)} < x^{(k+1)}$ and $y^{(k)} < y^{(k+1)}, \forall k$ s.t. $0 \leq k \leq K-1$, together with the *shared derivatives* at the internal points, $\{\delta^{(k)}\}_{k=1}^{K-1}, \delta^{(k)} \geq 0, \forall k$ and the boundary derivatives set to 1, i.e. $\delta^{(0)} = \delta^{(K)} = 1$. Given its expression, the neural spline has both an easily computable log Jacobian determinant, as well as an analytic inverse [8].

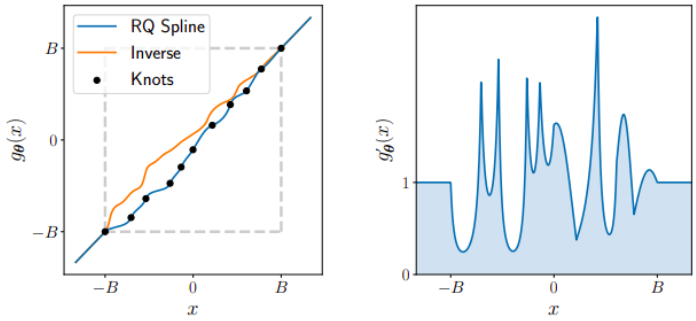

Figure 3: **Left:** A monotonic RQ spline parametrized by 11 knots and 9 shared derivatives (corresponding to the internal knots). **Right:** Derivative of the spline w.r.t x. Source: [8]

2.4 Variational Auto-Encoders

Latent variable models assume that the intrinsic dimensionality of data is much smaller than the observed one and that the data can be generated by first sampling a representation in the smaller, latent space, $\boldsymbol{z}$, which is then mapped to data space, $\boldsymbol{x}$, by sampling from a conditional distribution $p(\boldsymbol{x}|\boldsymbol{z})$. Following this generative process, the likelihood of the data can be written as:

$$p(\boldsymbol{x}) = \int p(\boldsymbol{z})p(\boldsymbol{x}|\boldsymbol{z})d\boldsymbol{z} \tag{4}$$

In cases where the above likelihood is untractable, variational inference is used to define a complex distribution $p_\theta(\boldsymbol{x})$ by using a latent variable $\boldsymbol{z}$ and maximizing the evidence lower bound (ELBO):

$$\mathop{\mathbb{E}}_{z\sim q_\phi(z)}\big[\log p_\theta(\boldsymbol{x},\boldsymbol{z})-\log q_\phi(\boldsymbol{z}|\boldsymbol{x})\big]=\mathbb{E}\Big[\log\frac{p_\theta(\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}+\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\Big]=\mathbb{E}\big[\log(p_\theta(\boldsymbol{x}|\boldsymbol{z}))\big]-\mathrm{KL}[q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z})] \tag{5}$$

where $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ is the approximate posterior with variational parameter ϕ and $p_\theta(\boldsymbol{z})$ is the prior on the latent variable.

Variational Auto-Encoders [9] parametrize both $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ and $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ using neural networks, while the prior $p(\boldsymbol{z})$ is a centered isotropic Gaussian $\mathcal{N}(\boldsymbol{z}|\boldsymbol{0},\boldsymbol{I})$. More precisely, the variational distribution is often chosen to be a multivariate Gaussian whose parameters $\phi=\{\mu,\sigma\}$ are the output of the decoder neural network which takes as input x. Similarly, the parameters of the conditional likelihood, θ, are the output of the decoder network which takes as input z. Depending on the data x, several choices such as Bernoulli or Gaussian are possible for modelling $p_\theta(\boldsymbol{x}|\boldsymbol{z})$. However, for increased flexibility, a normalizing flow can also be used.

2.5 Graph Convolutional Networks

Given a graph $G=(V,E)$ and input node features $\{x_v\}_{v\in V}$, the goal is to learn node representations which can then be used for mainstream tasks, such as node classification or link prediction. Graph Convolutional Networks (GCN) ([10], [11]) leverage the graph structure of the data by propagating information from each node to its neighbours. Denoting the adjacency matrix as A, a GCN building block is written as:

$$H^{(l+1)}=\phi\big(D^{-\frac{1}{2}}(A+I)D^{-\frac{1}{2}}H^{(l)}W^{(l)}\big) \tag{6}$$

where D is the diagonal degree matrix, $H^{(l)}$ are the node representations from layer l, $W^{(l)}$ are the weights of layer l and ϕ is a non-linearity. Intuitively, $D^{-\frac{1}{2}}(A+I)D^{-\frac{1}{2}}$ ensures message passing between neighbor nodes, including self connections due to the addition of I, while $H^{(l)}W^{(l)}$ performs a feature transformation.

This formulation encapsulates both the node features and graph structure and is efficient at the same time. Furthermore, edge embeddings can be obtained, for example, by aggregating the corresponding node embeddings (eg: via concatenation or element-wise mean or max).

3 Proposed Method

For a spatial point process on a graph $G=(V,E)$ with vertices V and edges E, the point domain $\mathcal{B}$ is represented by the reunion of line segments representing the edges. Assuming that x_u represents the position of node u and an edge from node u to node v is denoted by $uv\in E$, then this can be written as:

$$\mathcal{B}=\bigcup_{uv\in E}s(u,v),\ \text{where } s(u,v)=\{x|x=\alpha x_u+(1-\alpha)x_v,\alpha\in[0,1]\} \tag{7}$$

Thus, the position of each point can be uniquely described by the edge on which it is found and by the relative position on this (oriented) edge. Without loss of generality, we define the intensity on each edge on $[0,1]$ and later scale the edge intensities using the edge length. As the cumulative intensity function on a single edge, $\Lambda(x)$, is increasing, we model it using a neural spline with $\Lambda(0)=0$. The intensity function is then its derivative, i.e. $\lambda(x)=\frac{d}{dx}\Lambda(x)$, which can easily represent multi-model distributions (Fig. 3, right) and we can also write the total intensity of the process by summing up the individual edge intensities:

$$\Lambda=\sum_{uv\in E}\lambda(v)=\sum_{uv\in E}\left.\frac{d\Lambda(x)}{dx}\right|_{x=v} \tag{8}$$

Note that as the spline has a closed form inverse, it is also easy to evaluate $\Lambda(x)^{-1}$ and thus, to sample new points. Compared to the original formulation, however, we don't set the edge boundary derivatives (i.e. node derivatives) to 1, but rather consider them to be shared parameters. That is, we require all edges incident to a node v to have consistent densities at v.

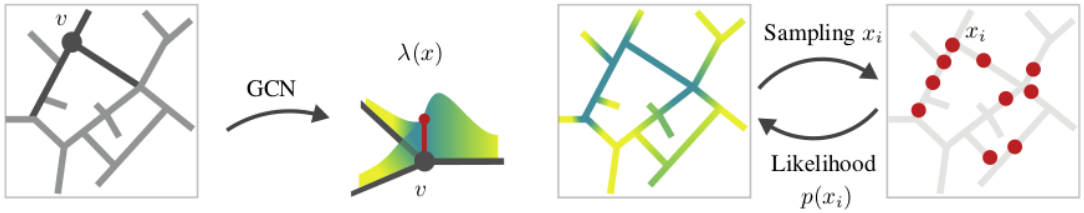

Figure 4: Sketch of our proposed method. For all nodes, representations are learned using a GCN, which are then used for obtaining the spline flow parameters. Some of the parameters are shared in order to achieve density consistency at nodes. The model can be used for sampling new point realizations. Source: [12]

3.1 Inhomogeneous spline flow

The base model defines a flow spline on each edge of a given, fixed graph G and it learns its parameters in a straightforward manner by minimizing the negative log likelihood based on Eq. 1. The detailed algorithms is presented below. However, using this approach, the parameters for each edge will be independent, whereas the data can present certain local interactions (eg: density on one edge might depend on densities of neighbouring edges). As a consequence, instead of modelling the flow parameters directly, we leverage the power of deep walk embeddings [13] and GCNs [11] on top of these embeddings, which output learnable node embeddings that are then used as inputs to simple feedforward networks in order to obtain the final flow parameters.

Algorithm 1 Basic inhomogeneous spline flow

Require: $\mathcal{D} = \{(r, e)\}$, points on a graph represented by an edge e and the relative position r on that edge.
Require: $G = (V, E)$, input graph underlying the data $\mathcal{D}$.
Require: K, number of knots per edge.
Require: ϵ, parameter used for random initializations.
Require: s, spline flow implementation
Require: o, optimizer
  {Initialize learnable parameters}
  Internal knot positions: $x(k, e) \sim \text{Uniform}(-\epsilon, \epsilon), y(k, e) \sim \text{Uniform}(-\epsilon, \epsilon) \forall e \in E, \forall k \in 1, \dots, K-1$
  Internal knot derivatives: $d_i(k, e) \sim \text{Uniform}(-\epsilon, \epsilon), \forall e \in E, \forall k \in 1, \dots, K-1$
  Node derivatives: $d_n(v) \sim \text{Uniform}(-\epsilon, \epsilon), \forall v \in V$.
  Edge heights: $y(K, e) \sim \text{Uniform}(-\epsilon, \epsilon), \forall e \in E$
  while not converged **do**
    for $(r, e) \in \mathcal{D}$ **do**
      nodes_from $= \{u | (u, v) \in e\}$ (gather all start nodes for edges in the batch)
      nodes_to $= \{v | (u, v) \in e\}$ (gather all end nodes for edges in the batch)
      $x, y, h = x[:, e], y[:, e], y[K, e]$ (fetch the knot positions)
      $d = [d_n(\text{nodes\_from}), d_i(:, e), d_n(\text{nodes\_to})]$ (gather all derivatives in order)
      $\Lambda, \log \lambda(r) = s(r, x, y, h, d)$ (get log intensities and total intensity)
      $\text{NLL} = \sum_r \log \lambda(r) - \Lambda$ (negative log-likelihoog)
      o.optimize(x, y, d_i, d_n) (update parameters to minimize NLL)
  end while

3.2 VAE

Real-life datasets might be a result of self-exciting processes, meaning that points are more probable in the future in places where there was a higher point density in the past. While the basic model with GCNs accounted for a possible dependency between edges, it does not take into account the interaction between points. Thus, we adapted the previous model into a VAE, as the latent variable can capture high-level information about the clustering of points.

From a batch of points on specific edges, the encoder produces a point-on-edge embedding for each point and aggregates them across all points in the batch via max pooling. For an edge $e = (u, v)$ and a point on e having relative position r, the point-on-edge embedding is obtained by applying a feed-forward network on the concatenation of the embeddings of node u and node v, together with r. As in the classical VAE implementation, the encoder output is used for obtaining a mean and covariance matrix that parametrize the Gaussian variational distribution. The decoder part remains the same as in the previous model, except that now the input also contains the latent variable. The ELBO can then be easily evaluated, as the KL term in Eq. 4 has a closed form, due to both distributions being Gaussian, while the expectation can be approximated by sampling from the latent space and averaging the resulting likelihoods.

3.3 Per-edge models

The models above learned different spline parameters for each edge which, besides having the disadvantage of increasing the model size, cannot generalize at all, as the graph structure is apriori fixed. As a consequence, we propose a new, 'per-edge' model, which contains a single spline that acts on each edge. Before, a single pass through the network was enough for obtaining all the parameters, even for the edges that were not part of the mini-batch. Now, one pass is required for each edge (but in practice they are not done sequentially, but rather batched). Because of this, it is more computationally expensive, as the model requires all edge intensities for computing the total intensity, but at the same time the number of parameters is greatly reduced (eg: 3.2M vs 9k parameters for base vs per-edge model, given the same values for hyperparameters, i.e. size 32 for all embeddings and a single hidden layer of size 32 for all networks)

3.4 Temporal model

As the datasets also contain time data, we also propose a temporal model which considers sequences of 14 days and then predicts the density for the next one, given the history. The model inherits its characteristics from the VAE encoder, as at each step it builds a points-on-graph embedding by aggregating point-on-edge embeddings for the corresponding day. These embeddings represent the sequential input for a recurrent model, such as a GRU or an LSTM. The hidden state at the last time step is then used as the input for the spline flow model. We reuse the flow implementation from the per-edge models.

4 Related Work

Kernel density estimation tools are the classical, non-parametric method for modelling the probability density of spatial point processes. In an unrestricted 2D plane, the kernel is a Euclidean distance, whereas for graph data, it is the graph distance induced by the shortest paths [14], [15], which can lead to biased estimates [16]. To overcome the boundary effects and the locality of the kernel methods, [16] extend the Euclidean penalized spline smoothing to graphs. [17] propose an auto-regressive parametrization, requiring the points to be ordered. Another line of work is represented by methods which do not produce continuous density estimates, for example [18], where a total variation regularized maximum-likelihood density estimation problem.

Most generally, modelling spatial point processes is done in an unrestricted setting. [19] propose a continuous equivariant normalizing flow, thus defining permutation invariant densities. Building on the kernel methods, [20] use VAEs to learn the kernel parameters, but the model is unable to sample new points. In order to deal with uncertainty around the intensity function, [21] propose a regularized method which provides robust out-of-sample coverage.

5 Experiments

5.1 Datasets

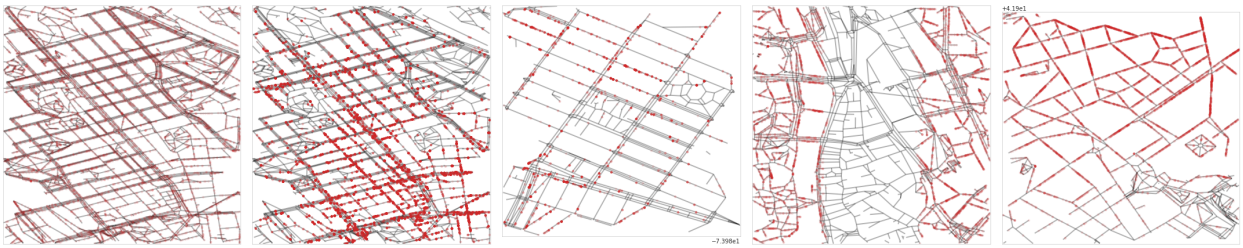

Figure 5: Various examples of datasets. From left to right: 1) Synthetic dataset of uniformly distributed points on the street network of Stockholm. 2) Check-in data Stockholm. 3) Check-in data New York. 4) Mixture dataset Zürich. 5) Inhomogeneous dataset Rome.

We use real-life graphs corresponding to the street networks of several cities, obtained using the [1]http://overpass-api.de. The size of the graphs is reduced by eliminating nodes with zero degree and by merging pairs of connected edges which don't intersect other edges. In terms of point processes, we utilized a real-life dataset of users' locations [22], which from now we will denote as the 'check-in dataset'. We also generated synthetic datasets, such as a uniform dataset, an inhomogeneous dataset, where points $(x, y) \in \mathbb{R}^2$ are sampled as follows: $(u_1, u_2^2) \sim \mathcal{U}(0, 1)$ and a mixture dataset, where points are uniformly sampled from the left and right vertical strips of the graph. When the generated points are not on the graph, they are projected to the closest edge. We refer to Fig. 5 for diverse examples.

5.2 Setup

We train all models with the Adam optimizer with learning rate in {1e-4, 1e-3} and L_2 regularizer in $\{0, 1\text{e-}3\}$ for 1000 epochs, using early stopping when the results do not improve after 30 epochs. We use mini-batches of size 64, as well as initial node embeddings and GCN embeddings of size 32 or 64. All the networks have either 1 or 2 hidden layers of size 64. Finally, the number of spline knots can be either $1, 3$ or 5. The data is split into train, validation and test (60%-20%-20%) and the best model is selected using the validation set. For the main results with confidence intervals, we report the test loss with 5 random parameter initializations.

5.3 Results

We present a comprehensive table containing negative log likelihood results on multiple cities and datasets for the base models in Table 1, the edge models in Table 2, for a 2D unrestricted spatial VAE flow in Table 3 and for the temporal model in Table 4. For the base models, the VAE achieves better results than the flow, as it are capable of capturing the interactions between points via the points-on-graph embedding. For the per-edge models, however, the benefits of the VAE model are outweighted by the small number of parameters (and maybe incapacity of fitting all edges).

[1]Overpass API

	Flow	VAE
New York	-34.48 ± 1.51	**-413.06 ± 5.65**
New York (synth)	-11.73 ± 0.03	**-113.74 ± 1.98**
Austin (synth)	-2.05 ± 0.05	**-19.64 ± 0.92**
Stockholm	-1.58 ± 0.02	**-281.51 ± 1.07**

Table 1: Test set negative log-likelihood for the simple flow and VAE models

	Edge-Flow	Edge-VAE
New York	**-33.29 ± 1.90**	-28.68 ± 0.32
New York (synth)	**-13.80 ± 0.43**	-13.97 ± 0.36
Austin (synth)	-897.6 ± 73.26	**-932.02 ± 21.21**
Stockholm	**-2.69 ± 0.11**	-2.53 ± 0.12

Table 2: Test set negative log-likelihood for the per-edge models

We also present visual results (Figure 6) and also show that the models require a large number of points for successful learning (Figure 8). Furthermore, in Figure 7 we show how density learning can sometimes fail. [23] offer a possible explanation for this, showing how a normalizing flow trained on ImageNet data assigns higher density to images from the CelebA or SVHN datasets and how they are not reliable when it comes to out of distribution detection.

	Spatial-VAE
New York	24.62
New York (synth)	20.31
Austin (synth)	9.61
Stockholm	486.8

Table 3: Test set negative log-likelihood for a spatial VAE model

	Temporal
New York	-7.67
New York (synth)	-10.35
Austin (synth)	-1.90
Stockholm	-430.61

Table 4: Test set negative log-likelihood for the temporal model

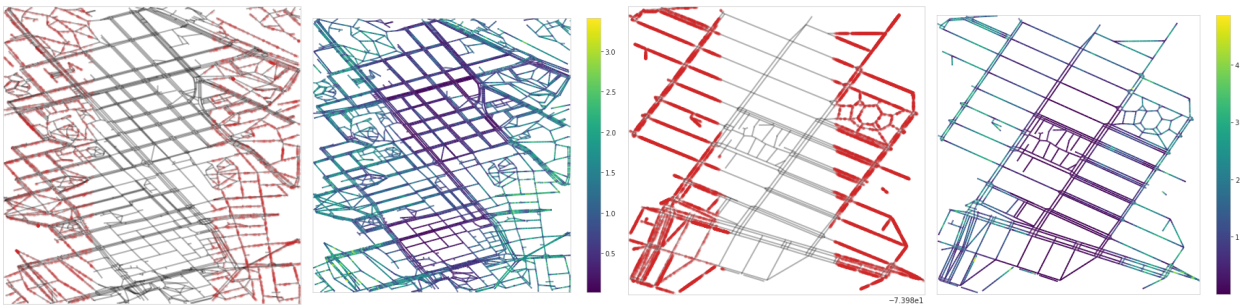

Figure 6: Mixture dataset on with the Edge-VAE successfully learned density on Stockholm (left) and New York (right). The smoothness of the spline prevents the line separating the regions from being perfect, but there is a clear difference in assigned density.

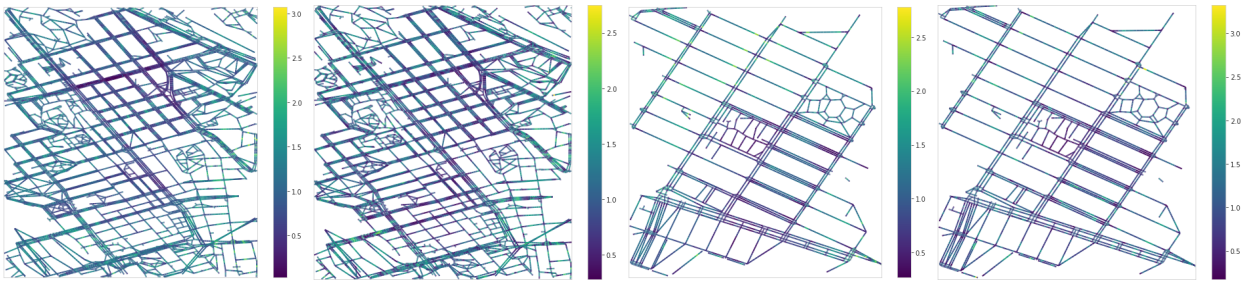

Figure 7: Unsuccessful density realizations for mixture datasets.

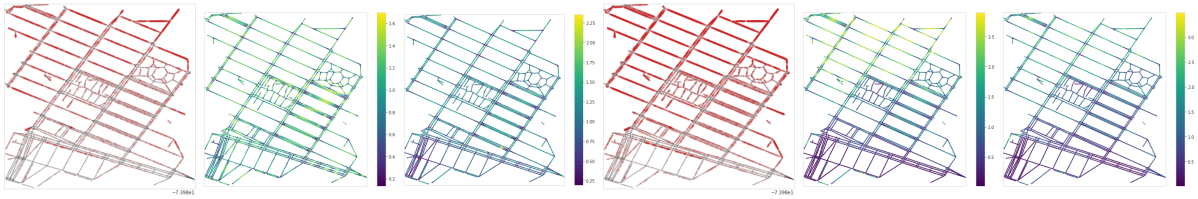

Figure 8: The impact of larger datasets, illustrated on synthetic datasets on the street network of New York. First three pictures, from left to right: inhomogeneous dataset with 2k samples, Edge-Flow density, Edge-VAE density. Last three plots, from left to right: inhomogeneous dataset with 9k samples, Edge-Flow density, Edge-VAE density. Note how the learned densities for the smaller dataset are almost uniform and also present arbitrary regions of higher density. For the larger dataset, the separation of the high and low density regions is more pronounced and respects the underlying data pattern.

In order to assess the generalization properties of the models, we also looked at the multi-graph setting (Figure 9). However, we found that it is too difficult to predict the density for unseen graphs, even if the points are distributed according to the same law, especially given our rotation and permutation equivariant setting.

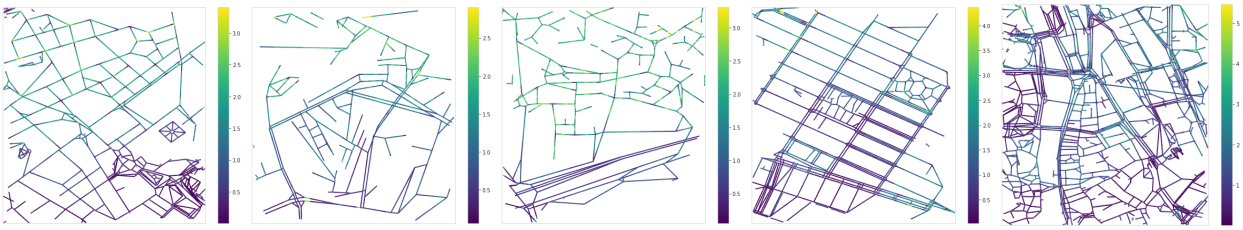

Figure 9: From left to right: 3 training graphs, one validation graph and one test graph. All datasets are inhomogeneous (i.e. high density in the upper part, low density in the lower part). The model correctly learns the underlying pattern of the data for the training graphs and although the same tendency can be seen for the validation and test graphs, there are also high density patches in the south and low density patches in the north (whereas we would expect a smooth transition from high density to low density regions)

6 Conclusion and Future Work

We proposed several models for estimating density on graphs by learning point processes which are restricted to the graph domain. Our comprehensive experiments showed that in the full-graph setting our methods successfully learned the patterns underlying the data, while the single-edge methods require more data for a correct fit. Our proposed methods leverage both the interaction between graph edges, as well as the interaction between points and besides density estimation, it can also be used for sampling new points, which might be useful for applications such as predictive policing or geo-spatial environmental applications.

As future work, it would be interesting to extend the model to marked point processes, i.e. model conditional distributions depending on the labels of the points. Another line of work would be to find efficient and effective methods for approximating the total intensity, which wouldn't require a forward pass for each graph edge. Finally, in order to make our models readily usable with real-life datasets which might be much smaller, less parameters are necessary. To this end, an ablation study might be necessary in order to evaluate the impact of reducing the size or removing different network components.

References

[1] Adrian Baddeley, Ege Rubak, and Rolf Turner. *Spatial point patterns: methodology and applications with R*. CRC press, 2015.

[2] Yongmei Lu and Xuwei Chen. On the false alarm of planar k-function when analyzing urban crime distributed along streets. *Social science research*, 36(2):611–632, 2007.

[3] Andrew Bray and Frederic Paik Schoenberg. Assessment of point process models for earthquake forecasting. *Statistical science*, pages 510–520, 2013.

[4] Guillaume Perrin, Xavier Descombes, and Josiane Zerubia. *Point processes in forestry: an application to tree crown detection*. PhD thesis, INRIA, 2006.

[5] Suman Rakshit, Tilman Davies, M Mehdi Moradi, Greg McSwiggan, Gopalan Nair, Jorge Mateu, and Adrian Baddeley. Fast kernel smoothing of point patterns on a large network using two-dimensional convolution. *International Statistical Review*, 87(3):531–556, 2019.

[6] Adrian Baddeley, Aruna Jammalamadaka, and Gopalan Nair. Multitype point process analysis of spines on the dendrite network of a neuron. *Journal of the Royal Statistical Society: Series C: Applied Statistics*, pages 673–694, 2014.

[7] George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *arXiv preprint arXiv:1912.02762*, 2019.

[8] Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural spline flows. *arXiv preprint arXiv:1906.04032*, 2019.

[9] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[10] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[11] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[12] Marin Biloš, Andreea Muşat, and Stephan Günneman. Point processes on graphs with normalizing flows. In *Deep Learning for Graphs at AAAI Conference on Artificial Intelligence 2021, AAAI workshop 2021*, 2021.

[13] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.

[14] Zhixiao Xie and Jun Yan. Kernel density estimation of traffic accidents in a network space. *Computers, environment and urban systems*, 32(5):396–406, 2008.

[15] Atsuyuki Okabe and Kei-ichi Okunuki. A computational method for estimating the demand of retail stores on a street network and its implementation in gis. *Transactions in GIS*, 5(3):209–220, 2001.

[16] Marc Schneble and Göran Kauermann. Intensity estimation on geometric networks with penalized splines. *arXiv preprint arXiv:2002.10270*, 2020.

[17] Jakob G Rasmussen and Heidi S Christensen. Point processes on directed linear networks. *Methodology and Computing in Applied Probability*, pages 1–21, 2020.

[18] Robert Bassett and James Sharpnack. Fused density estimation: theory and methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 81(5):839–860, 2019.

[19] Marin Biloš and Stephan Günnemann. Equivariant normalizing flows for point processes and sets. *arXiv preprint arXiv:2010.03242*, 2020.

[20] Baichuan Yuan, Xiaowei Wang, Jianxin Ma, Chang Zhou, Andrea L Bertozzi, and Hongxia Yang. Variational autoencoders for highly multivariate spatial point processes intensities. In *International Conference on Learning Representations*, 2019.

[21] Muhammad Osama, Dave Zachariah, and Petre Stoica. Prediction of spatial point processes: Regularized method with out-of-sample guarantees. *arXiv preprint arXiv:2007.01592*, 2020.

[22] Eunjoon Cho, Seth A Myers, and Jure Leskovec. Friendship and mobility: user movement in location-based social networks. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1082–1090, 2011.

[23] Polina Kirichenko, Pavel Izmailov, and Andrew Gordon Wilson. Why normalizing flows fail to detect out-of-distribution data. *arXiv preprint arXiv:2006.08545*, 2020.