# Spatiotemporal Modeling using Recurrent Neural Processes

Sumit Kumar

CMU-RI-TR-19-67
July 2019

Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

**Thesis Committee:**
Katia Sycara, Chair
George Kantor
Wenhao Luo

*Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.*

*For my parents and sisters*

## Abstract

Spatiotemporal processes, such as temperature in an area, motion of a vehicle, etc., depend on the spatial features of the underlying phenomena as well as time. Developing models that can estimate both mean and uncertainty associated with the prediction is important for building robust systems capable of performing such tasks. Although, Gaussian Process [27] models can estimate predictive distributions over the target data points, they are not scalable to large data regimes due to their non-parametric nature. Moreover, the explicit functional form of kernels limits the modeling capabilities to only smoothing and interpolation of data. In this work, we propose a deep neural network model, called *Recurrent Neural Processes*, adept for modeling spatiotemporal data. Being parametric, our model easily scales to large data regimes. Additionally, the use of neural networks enables the model to overcome functional design restrictions and learn implicit kernels from the data directly. Our proposed model is a latent variable model, i.e., the predicted output for a given input depends on sample drawn from a learned latent distribution. The model's predictive distribution is empirically estimated from the multiple output values obtained as a result of drawing different samples from the latent distribution. We compare our proposed model with the existent ones in the literature and show that our model can make accurate future time step predictions and can also provide meaningful structured uncertainty estimates for spatiotemporal data.

## Acknowledgments

I would like to express my sincerest gratitude towards my research advisor, Prof. Katia Sycara for her continuous support and guidance over the last two years. This work would not have been possible without her invaluable technical assistance and moral support. Her expertise in the subject matter, immense patience and trust in students, and friendliness make her an ideal mentor. I would also like to thank my committee members, George Kantor and Wenhao Luo, for their consistent guidance and feedback which have proved to be of immense help in this research. Their encouragements and constructive criticisms have helped me in becoming a better researcher and in successful completion of this project.

I would also like to thank my wonderful labmates and friends for helping me out whenever I felt stranded in my work. I am especially grateful towards Akshat Agarwal and Swaminathan Gurumurthy, for their collaboration on various projects and numerous insightful discussions that have kept the projects running and have personally helped in improving my research skills. Special thanks to my dear friend, Ishika Soni, for always supporting me and cheering me up especially in times of need.

Finally, I am deeply grateful towards my incredible family for motivating me to pursue an academic career, supporting me in all my decisions and always being there for me.

# Contents

# List of Figures

# Chapter 1

# Introduction

Spatiotemporal processes depend on both the spatial features of the underlying phenomena and time. They are encountered in numerous domains such as video prediction [7], active monitoring [18], self-driving [6], etc. For instance, the motion of any vehicle on the road depends on various spatial and temporal factors like presence of other vehicles, its past motion or trajectory, road connectivity network, etc. A self-driving vehicle needs to model the complex dependencies among these factors in order to accurately forecast the motion of all other vehicles in its vicinity. Hence, for building robust artificial intelligence (AI) systems that can perform such tasks, it is important to develop machine learning models capable of making accurate predictions for such spatiotemporal phenomena.

Additionally, AI systems should be able to express their model's uncertainty along with the predicted value. In other words, instead of just producing an output from a black box, the model should also quantity its confidence in that prediction. This is important because such uncertainty estimates can serve as a safety mechanism enforcing the system's behavior to be more cautious, safe and conservative especially when its confidence levels are low. On the contrary, in the absence of such confidence estimates, the system may take certain undesirable or harmful decisions, for instance, an over-confident but incorrect prediction made by a self-driving car regarding the motion of other vehicle can have catastrophic outcomes. An example demonstrating the benefit of predicting uncertainty in autonomous driving is shown in fig. 1.1. The uncertainty estimates can also be useful in active exploration or adaptive sampling tasks where a robot or a team of robots monitoring an area can sample locations where they have high uncertainty regarding the target distribution in order to gain more information about the underlying environmental process as shown in fig. 1.2. Similarly, in the human-robot collaborative tasks or imitation learning setup, where an AI agent is trying to learn an optimal policy from the behavior of an expert human, the agent can select to query the expert for only those data points or experiences on which it is most uncertain about the optimal behavior, instead of querying on each and every sample, thus reducing the input from the human and increasing the speed of learning (see fig. 1.3).

Gaussian Process (GP) [27] is one of the most popular machine learning models that can express uncertainty along with its predictions. A GP defines a prior distribution over functions, which can be updated to a posterior on observing some data. However, being non-parametric models, GPs are not scalable to large datasets. We review Gaussian Processes and some of the recent advancements in this field in section 3.1.

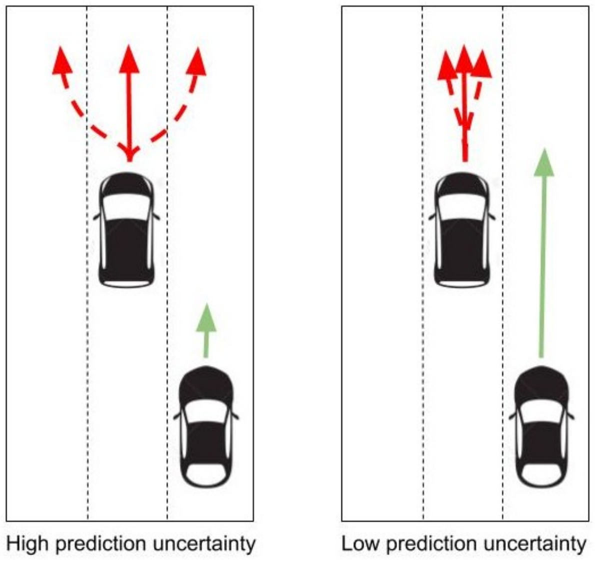


Figure 1.1: An autonomous vehicle in the rightmost lane predicts the motion of another vehicle in the middle lane. If its prediction (shown in red) has high uncertainty, the vehicle becomes extra careful in its approach as shown on the left. Otherwise, the vehicle can continue on its trajectory as shown on the right.

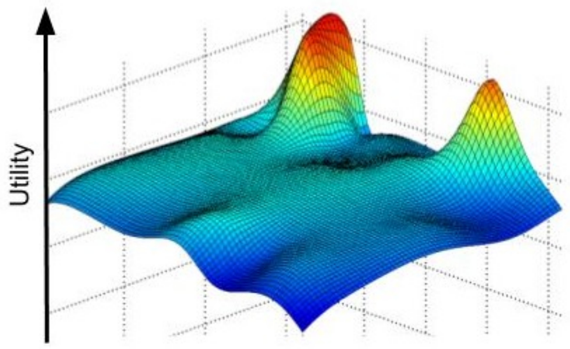


Figure 1.2: The utility distribution for a scalar field such as temperature over a 2D field is shown. The utility of collecting data/readings from any point is proportional to the model's uncertainty over the true value. An agent can select places with high uncertainty or equivalently with high utility, to collect samples and gain information regarding the distribution of the scalar field.

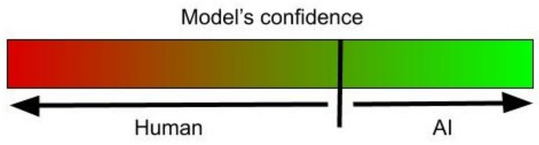


Figure 1.3: In a collaborate human-AI setup, the decision making can be shared between the human and the agent. If the agent has low confidence or high uncertainty in its prediction, it can handover the control to the human, otherwise, it can execute the appropriate actions.

Neural Network (NN) models have been highly successful in automatically capturing the underlying complex patterns of the data and have achieved state-of-the-art performance in numerous applications [12, 19, 31]. Given enough data, these models can learn the complex dependencies between features and have achieved human level or even superhuman performance results in various fields [23, 29]. Being parametric models, they do not suffer from scalability issues in the large data regime, unlike GPs. Their powerful representation capability have made them most popular model in many fields these days. However, deep networks, in their original formulation, do not provide any uncertainty estimate. In other words, they output a point estimate rather than a distribution. Recently, some neural networks models that can provide uncertainty estimates too have been proposed. We discuss these in section 3.2.

In this work, we propose a deep neural network model, called *Recurrent Neural Processes*, that, like Gaussian Processes, can estimate both mean and uncertainty of its predictions of a spatiotemporal process. Being parametric, our proposed model easily scales to large datasets.

Spatiotemporal modeling of data that exists in the form of images is a well-studied topic in the research community and popular deep Convolutional Recurrent Neural Network models [21, 28] have achieved impressive results on many datasets. Similarly, if the data is in the form of a fixed graph, then Graph Recurrent Neural Network models [14] can be used. In this work, however, we are interested in developing models for data that do not necessarily have any such pre-defined structure (images/graphs). Such cases are prevalent, for instance, a robot equipped with the task of environment monitoring takes readings of a physical phenomenon of interest such as temperature from a subset of all possible locations at each time step, instead of exhaustively sampling the entire area, thus the accumulated data is a set of locations and the corresponding readings for each time step. Similarly, in motion prediction task, the input for the model is a set of detected objects, extracted from the camera image, and their corresponding state at discrete time steps. Hence, it is important to develop models that can make accurate predictions of a spatiotemporal process given only individual samples from different time steps instead of images or graphs.

The rest of the document is organized as follows:

- We formally describe the problem we are addressing in this work in Chapter 2.
- We review popular models widely used for estimating predictive distributions in Chapter 3.
- We describe our proposed model in Chapter 4.
- The datasets used in this work, experimental setup and results are described in Chapter 5.
- We finally conclude and discuss future directions in Chapter 6.

# Chapter 2

# Problem Formulation

In simple words, the objective of this work is:

**Given some observations of a spatiotemporal process for $t = 1, \dots, T$; predict mean and uncertainty values for $T' > T$.**

We now give a mathematical formulation of this problem. Let, $(x_i^t, y_i^t)$ denote an input-output pair where $x_i^t \in \mathcal{X}$ is a data point in the input space $\mathcal{X}$ and $y_i^t \in \mathcal{Y}$ be the corresponding output value at time $t$. Additionally, let $\mathcal{F} : \mathcal{X} \to \mathcal{Y}$ be the mapping from the input space to the output space.

We are given a context set $C = \{\{x, y\}_{1:N_1}^{1}, \dots, \{x, y\}_{1:N_T}^{T}\}$ comprising of some input-output pairs where $\{x, y\}_{1:N_t}^{t} = \{(x_1^t, y_1^t), \dots, (x_{N_t}^t, y_{N_t}^t)\}$. We will use short hand notations $x_\mathcal{C}$ and $y_\mathcal{C}$ for input and output of context set $C$ respectively, i.e., $x_\mathcal{C} = \{x_{1:N_1}^1, \dots, x_{1:N_T}^T\}$ and $y_\mathcal{C} = \{y_{1:N_1}^1, \dots, y_{1:N_T}^T\}$. Additionally, we are provided with a target input set $x_\mathcal{T} \equiv x_{1:M}^{T'} = \{x_1^{T'}, x_2^{T'}, \dots, x_M^{T'}\}$ belonging to time step $T' > T$. We want to estimate a conditional probability distribution $P(y_\mathcal{T} \mid x_\mathcal{T}, x_\mathcal{C}, y_\mathcal{C})$ over the corresponding output value $y_\mathcal{T} \equiv y_{1:M}^{T'} = \{y_1^{T'}, y_2^{T'}, \dots, y_M^{T'}\}$ of the target input set $x_\mathcal{T}$. Note that the context data points span from $t = 1$ to $t = T$ whereas the target data points belong to $t = T' > T$ time step.

Figure 2.1 shows a spatiotemporal process - the motion of a rigid body (MNIST digit) in an area. In the first row, the individual time steps from this process are shown. For each of the first 3 time steps, we know the intensity value of some pixels or cells which are shown in non-blue color in the bottom row. The context input $x_\mathcal{C}$ is the 2D coordinate of all of these non-blue pixels. The context output $y_\mathcal{C}$ is the corresponding intensity value of all the input pixels at that time step. The target input $x_\mathcal{T}$ is the set of all the cells/pixels at $t = 4$.

The target distribution $P(y_\mathcal{T} \mid x_\mathcal{T}, x_\mathcal{C}, y_\mathcal{C})$ is a Normal distribution:

$$P(y_\mathcal{T} \mid x_\mathcal{T}, x_\mathcal{C}, y_\mathcal{C}) = \mathcal{N}\left(\mu_{\mathcal{T}|\mathcal{C}}, \Sigma_{\mathcal{T}|\mathcal{C}}\right)$$

where the mean $\mu_{\mathcal{T}|\mathcal{C}}$ is a vector of size $M$ and the covariance matrix $\Sigma_{\mathcal{T}|\mathcal{C}}$ is an $M \times M$ matrix.

Estimating model's uncertainty along with modeling both spatial and temporal patterns in the data is a challenging task. We will first discuss some models that can estimate both mean and the associated uncertainty with their predictions for static spatial processes in Chap. 3. Then, we

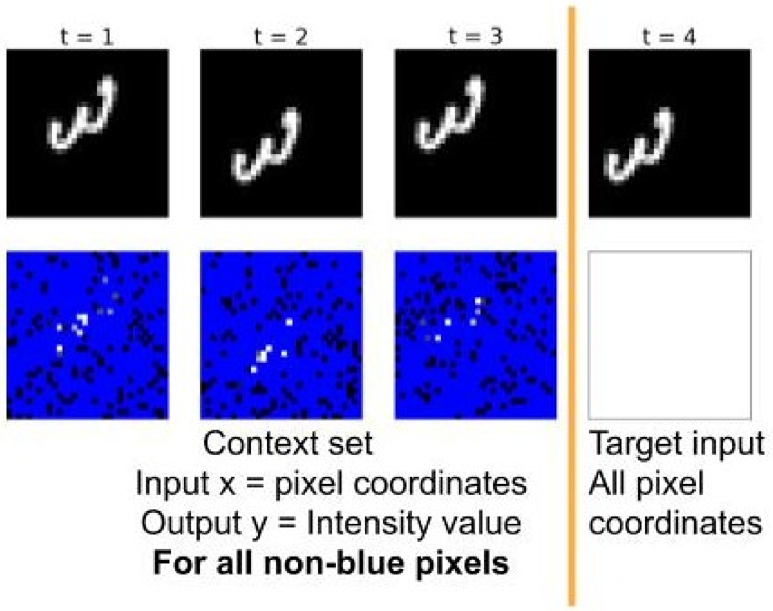


Figure 2.1: The figure displays the motion of a rigid body (MNIST digit) in a box as a spatiotemporal process. The context set consists of input-outputs pair where input is the 2D location of each of the non-blue pixels in the first 3 time steps and the output is its corresponding intensity value. The target input consists of all the grid cells at $t = 4$.

will describe our proposed model that can model temporal patterns too and estimate predictive distribution over the target data conditioned on the context data in Chap. 4.

# Chapter 3

# Related Works

We discuss two different classes of models that can estimate both mean and the associated uncertainty with their predictions for static spatial processes.

## 3.1 Gaussian Process

Gaussian Process (GP) [27] is a widely used probabilistic model that defines distributions over functions. This is different from other popular models, such as linear regression models, polynomial regression models, support vector machines [4], etc. that parameterise a single function and try to optimize the parameters in order to maximize the likelihood of training data or minimize some loss function on the training dataset. Gaussian Process is a stochastic process, which is a collection of random variables, such that every finite collection of those random variables has a multivariate normal distribution. The joint probability distribution over the output value of a set of points $x_1, \ldots, x_n$ can be written as:

$$\begin{bmatrix} f(x_1) \\ \vdots \\ f(x_n) \end{bmatrix} \sim \mathcal{N}\left( \begin{bmatrix} m(x_1) \\ \vdots \\ m(x_n) \end{bmatrix}, \begin{bmatrix} k(x_1, x_1) & \ldots & k(x_1, x_n) \\ \vdots & \ddots & \vdots \\ k(x_n, x_1) & \ldots & k(x_n, x_n) \end{bmatrix} \right)$$

$$f(X) \sim \mathcal{N}\left(\mu_X, K_{XX}\right)$$

where,

$$X = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \mu_X = \begin{bmatrix} m(x_1) \\ \vdots \\ m(x_n) \end{bmatrix} K_{XX} = \begin{bmatrix} k(x_1, x_1) & \ldots & k(x_1, x_n) \\ \vdots & \ddots & \vdots \\ k(x_n, x_1) & \ldots & k(x_n, x_n) \end{bmatrix}$$

A GP is fully defined by a mean function $m$ and a covariance function $k$:

$$f(\cdot) \sim GP\left(m(\cdot), k(\cdot, \cdot)\right)$$

The mean function $m$ is typically assumed to be zero without any loss of generality. The covariance function $k$, also known as kernel, encodes assumptions about the structure of the latent function $f$. It describes the relation between two data points and typically has some free hyperparameters to control this relation. A popular choice of kernel function is radial basis function or squared exponential kernel defined as:

$$k_{RBF}(x, x') = \sigma^2 \exp\left(-\frac{\|x - x'\|^2}{l^2}\right)$$

where $l$ is the lengthscale hyperparameter and $\sigma$ is the scale hyperparameter. If $l$ is large, then two data points $x$ and $x'$ which are far from each other in the input space are also correlated, implying that the true function $f$ is smooth. On the other hand, if $l$ is small then $f$ is likely to be a non-smooth function with rapid variations.

Given a training dataset of size $N$ with input $X$ and output $Y$, the log marginal likelihood of the data can be written as:

$$\mathcal{L} = -\frac{N}{2}\log(2\pi) - \frac{1}{2}\log\det(K_{XX}) - \frac{1}{2}Y^\top K_{XX}^{-1}Y \tag{3.1}$$

where $K_{XX}$ is the pairwise covariance matrix whose $(i,j)^{th}$ element is given by $k(X_i, X_j)$ with $X_i$ and $X_j$ being the $i^{th}$ and $j^{th}$ elements in $X$ respectively. The hyperparameters of the GP model ($\sigma$ and $l$ for RBF kernel) can be estimated by maximizing the log marginal likelihood $\mathcal{L}$ in eq. 3.1. In other words, the prior distribution is updated, by maximizing the likelihood of the observed training data with respect to the model's hyperparameters, into a posterior distribution.

After optimizing the model's hyperparameters, the posterior target distribution of a target set of samples $X^*$ conditioned on the training set $(X, Y)$ can be computed as a normal distribution:

$$\begin{aligned} f(X^*) \mid X^*, X, Y &\sim \mathcal{N}(\mu_{X^*}, K_{X^*}) \\ \mu_{X^*} &= K_{X^*X}K_{XX}^{-1}Y \\ K_{X^*} &= K_{X^*X^*} - K_{X^*X}K_{XX}^{-1}K_{X^*X}^\top \end{aligned}$$

where $K_{X^*X}$ is the cross-covariance matrix between $X^*$ and $X$.

GPs are non-parametric models, i.e., the number of parameters grow with the data. They are computationally very intensive as they incur $O(N^2)$ storage cost and $O(N^3)$ inference cost for a dataset of size $N$. Although, state-of-the-art approximations [25, 26, 34] reduce the inference cost to $O(N^2)$, the non-parametric nature of the model makes it ill-suited for large datasets. Especially, in processes with temporal variations, where the amount of observed data increases with time, these models become computationally intractable. Furthermore, identifying the most suitable kernel $k$ for the model requires domain knowledge regarding the nature and distribution of data, thus limiting the applicability of the model to small datasets only.

Gaussian processes model distributions over functions providing a Bayesian non-parametric approach to smoothing and interpolation of data. Wilson and Nickisch [34] proposed Spectral Mixture Kernels suitable for pattern discovery and achieving long range extrapolation behaviors. These kernels are derived by modelling a spectral density, which is the Fourier transform of a kernel, with a Gaussian mixture.

Recently, many approaches have tried to combine the rich representation power of deep neural networks with the probabilistic non-probabilistic modeling capabilities of Gaussian Processes. Damianou and Lawrence [5] proposed Deep GPs which are a deep belief network based on Gaussian process mappings. They model the observed data as the output of a multivariate GP and the inputs to that Gaussian process are then governed by another GP. Wilson et al. [35] proposed Deep Kernel GPs where the input data is first transformed into a representation space by using neural networks. The obtained representation are then fed into standard GP kernels.

## 3.2 Deep Neural Networks

Uncertainty estimation with deep neural network models is an active area of research. In their original formulation, neural networks define a single deterministic function parameterised by weights and biases and hence are not capable of expressing uncertainty associated with their predictions. Considered to be one of the pioneer works, Blundell et al. [3] proposed a backpropagation compatible algorithm, called *Bayes by Backprop*, for learning normal distributions over the parameters of a neural network. By assigning distribution over the parameters, the model's output for a given input is also a distribution instead of just a single point estimate. Fortunato et al. [8] extended this model for Recurrent Neural Networks by using an adapted version of truncated backpropagation through time. Due to the large number of parameters of the deep network model, closed form posterior analysis becomes computationally intractable. As a result, Monte Carlo samples are drawn from the parameter distribution to approximate the posterior distribution over the target value.

### 3.2.1 Neural Processes

Recently, Garnelo et al. [10] proposed Neural Process (NP) - a deep neural network model that, like GP model, can estimate distribution over the output value of a target set of points conditioned on some input or context set of points. However, unlike GP, NP is a parametric model with architecture similar to that of Variational Autoencoder [17].

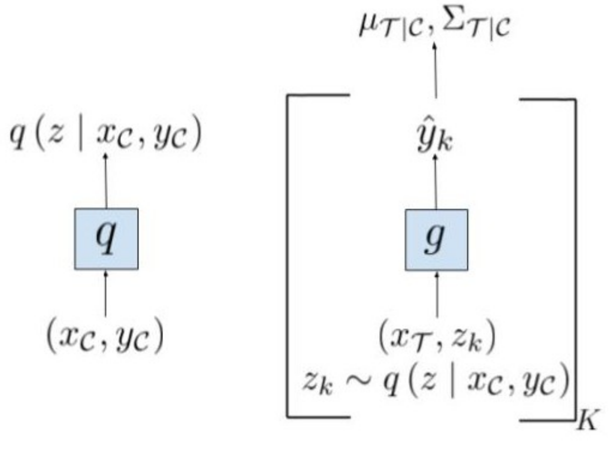

$\mu_{\mathcal{T}|\mathcal{C}}, \Sigma_{\mathcal{T}|\mathcal{C}}$
$q(z \mid x_\mathcal{C}, y_\mathcal{C})$
$\hat{y}_k$
$q$
$g$
$(x_\mathcal{C}, y_\mathcal{C})$
$(x_\mathcal{T}, z_k)$
$z_k \sim q(z \mid x_\mathcal{C}, y_\mathcal{C})$
$K$

Figure 3.1: The figure shows the schematic of Neural Process model. Left: The encoder computes the latent distribution $q(z \mid x_\mathcal{C}, y_\mathcal{C})$ from the context set $(x_\mathcal{C}, y_\mathcal{C})$. Right: The decoder makes different predictions for the target input $x_\mathcal{T}$ by drawing different $z_k$ from the latent distribution. The predictive mean and covariance are then empirically estimated from the multiple predictions.

Figure 3.1 shows the overall functionality of the model. An encoder module first computes a latent distribution $q\left(z \mid x_{\mathcal{C}}, y_{\mathcal{C}}\right)$ from the context set $\left(x_{\mathcal{C}}, y_{\mathcal{C}}\right)$ of input-output pairs. The decoder then takes as input the target input $x_{\mathcal{T}}$ and a sample $z_k$ drawn from the latent distribution to make prediction $\hat{y}_k$. By drawing $K$ different latent samples $z_1, \ldots, z_K$, the model predicts $K$ different values $\hat{y}_1, \ldots, \hat{y}_K$, from which it can then estimate the predictive mean $\mu_{\mathcal{T}|\mathcal{C}}$ as:

$$\mu_{\mathcal{T}|\mathcal{C}} = \frac{1}{K} \sum_{k=1}^{K} \hat{y}_k$$

Similarly, the predictive covariance matrix $\Sigma_{\mathcal{T}|\mathcal{C}}$ can be computed as:

$$\Sigma_{\mathcal{T}|\mathcal{C}} = \frac{1}{K-1} \sum_{k=1}^{K} \left(\hat{y}_k - \mu_{\mathcal{T}|\mathcal{C}}\right) \left(\hat{y}_k - \mu_{\mathcal{T}|\mathcal{C}}\right)^{\top}$$

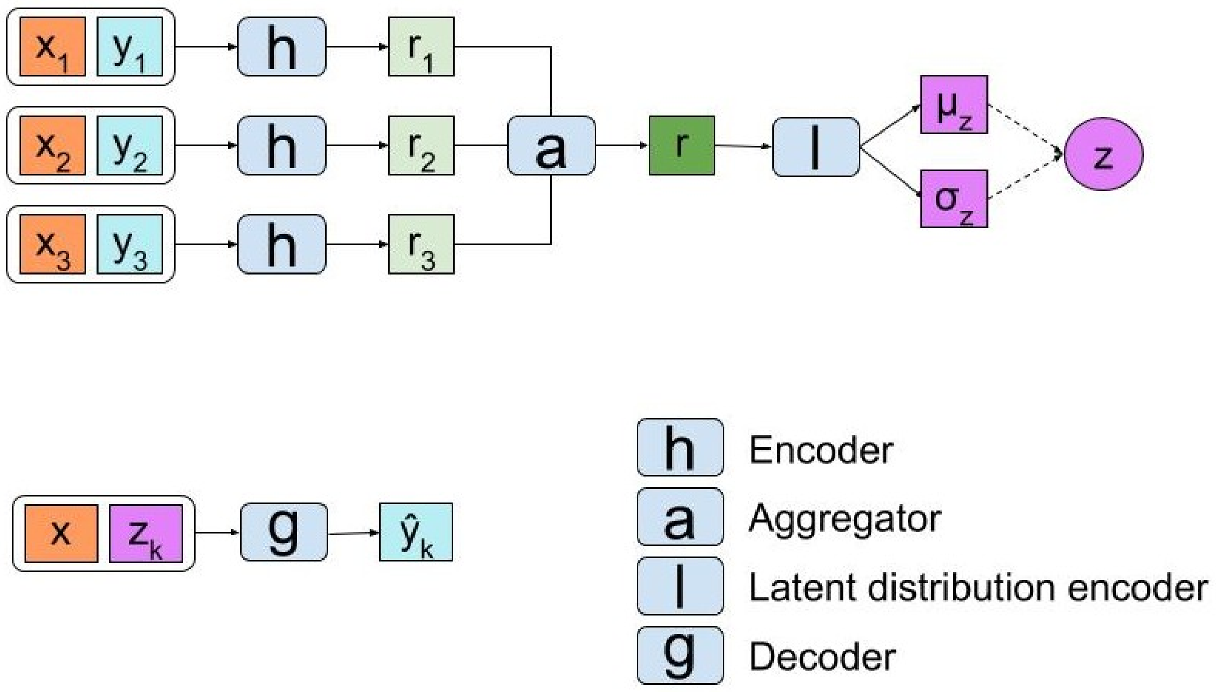


Figure 3.2: The figure shows the Neural Process model. Encoder $h$ computes a representation vector $r_i$ for each context data pair $(x_i, y_i)$. The aggregator module $a$ then combines them all into a state representation $r$. The latent distribution encoder $l$ then transforms this into a latent distribution $\mathcal{N}\left(\mu_z, \sigma_z I\right)$. Finally, the decoder $g$ takes as input the target data point $x$ and a latent sample $z_k$ and outputs a predicted value $\hat{y}_k$.

The Neural Processes model is shown in fig. 3.2. It consists of the following modules:

- An encoder $h$ computes representation vector $r_i = h\left(x_i, y_i\right)$ for each input-output pair $\left(x_i, y_i\right)$ in the context set. The encoder is parameterised as a neural network.
- The aggregator module $a$ then combines all the representation vectors $\{r_i\}_{i=1}^N$ into a state representation vector $r$, i.e., $r = a\left(\{r_i\}_{i=1}^N\right)$. The aggregator $a(\cdot)$ needs to be invariant to the number and order of context data points. The simplest choice is the mean aggregation function, i.e. $r = a\left(\{r_i\}_{i=1}^N\right) = \frac{1}{N} \sum_{i=1}^N r_i$.

- A latent distribution encoder module $l$ then transforms the state representation $r$ into a latent distribution $q\left(z \mid x_{\mathcal{C}}, y_{\mathcal{C}}\right)$, which is a factorized Normal distribution $\mathcal{N}\left(\mu_z, \sigma_z I\right)$ where $\mu_z, \sigma_z = l\left(r\right)$. This module is also a neural network with two output heads, one for mean and one for variance.
- The decoder takes as input the target input $x_{\mathcal{T}}$ and $z_k \sim \mathcal{N}\left(\mu_z, \sigma_z I\right)$ and outputs a predicted value $\hat{y}_k$. The decoder is also parameterised as a neural network.

Like GPs, NPs model distribution over functions and can estimate conditional target distribution. However, unlike GPs, where closed-from posterior analysis is carried out to compute the predictive distribution, NPs draw Monte Carlo samples from the latent distribution to empirically estimate the target distribution. The inference cost per target data point for this model is $O(N)$ where $N$ is the size of the context set.

However, NP model does not take into account any temporal dependencies present in the data and hence, is not applicable for spatiotemporal datasets. In this work, we build on top of this model and propose *Recurrent Neural Processes* which can estimate conditional target distribution for spatiotemporal datasets while inheriting all the favorable properties of the former.

# Chapter 4

# Method

## 4.1 Modeling Temporal Dependencies

We saw in section 3.2.1 that Neural Process is a promising framework for modeling static spatial processes with favorable properties like linear inference time per target data point and the ability to learn implicit kernel from the data directly, however, the model is incapable of capturing temporal dependencies. We now describe our proposed approach for incorporating temporal dependencies too in the framework.

In the Neural Processes model for static spatial processes, the state representation $r$ summarizes all the contextual information in a fixed size embedding as shown in fig. 4.1. The latent distribution $q\left(z^{T'} \mid x_{\mathcal{C}}, y_{\mathcal{C}}\right)$ captures the important information while filtering out the noise.

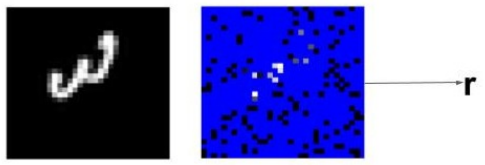


Figure 4.1: For static spatial processes, a single state representation vector $r$ summarizes all the contextual information.

An spatiotemporal process can be considered as a *related* sequence of static spatial processes. At each time step, we have a unique spatial process, however, these are not independent of each other but form a temporally correlated sequence of spatial processes as shown in fig. 4.2. We propose to compute time-dependent state representation $r^t$ which summarizes all the context information for time step $t$. Effectively, this $r^t$ summarizes the spatial process at time step $t$.

To make predictions for time step $T'$, we need to compute the corresponding latent distribution $q\left(z^{T'} \mid x_{\mathcal{C}}, y_{\mathcal{C}}\right)$. Note that this latent distribution $q\left(z^{T'} \mid x_{\mathcal{C}}, y_{\mathcal{C}}\right)$ corresponds to the spatial process at time step $T'$ for which we do not have any context data pairs. To estimate $q\left(z^{T'} \mid x_{\mathcal{C}}, y_{\mathcal{C}}\right)$, we need to determine the state representation for time step $T'$, i.e., $r^{T'}$. We propose to estimate the desired target state representation $r^{T'}$ from all the contextual state representations $r^1, r^2, \ldots, r^T$ using a forecaster module $f$, i.e., $r^{T'} = f\left(r^1, r^2, \ldots, r^T\right)$. This forecaster module is an auto-regressive model that takes as input the contextual state representations $r^1, r^2, \ldots, r^T$ and predicts $r^{T'}$. If $T'$ is the immediate next time step, i.e., $T' = T+1$, the module outputs $r^{T+1}$ directly, otherwise, it first computes $r^{T+1}$, then $r^{T+2}$ which depends on the pre-

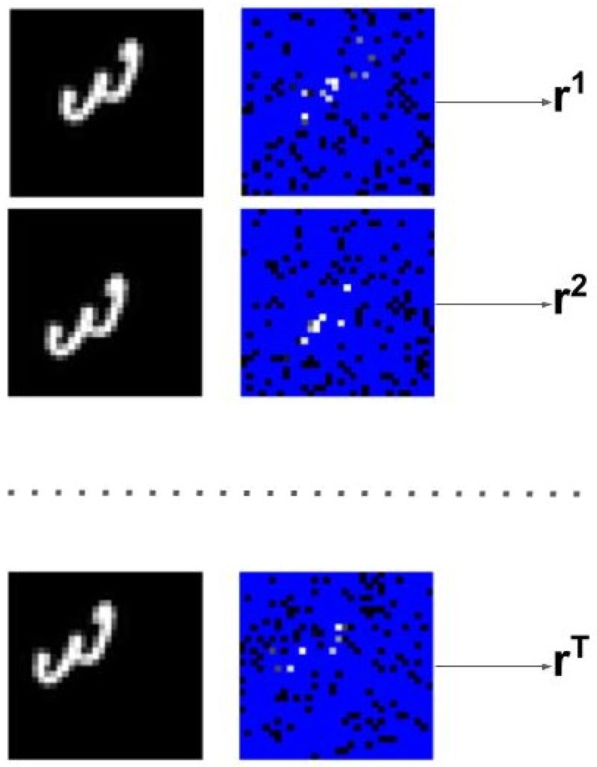


Figure 4.2: Spatiotemporal process can be considered a sequence of static spatial processes. Here, state representation $r^t$ summarizes the contextual information for time step $t$.

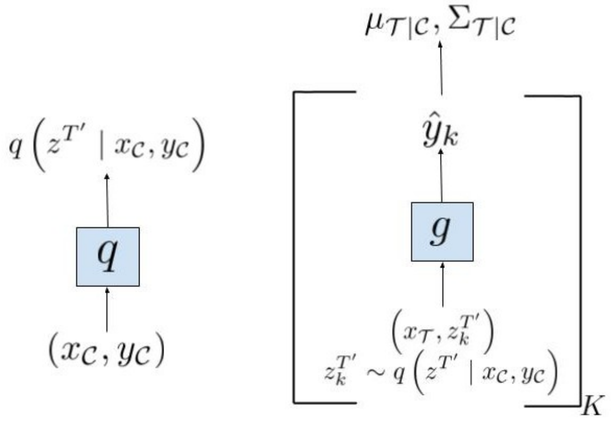
dicted $r^{T+1}$ along with all the contextual state representations. This process is repeated until the desired time step $T'$ and finally this module estimates $r^{T'}$.

## 4.2 Model

Figure 4.3 shows the overall functionality of our proposed model, Recurrent Neural Process. First, the latent distribution $q\left(z^{T'} \mid x_{\mathcal{C}}, y_{\mathcal{C}}\right)$ corresponding to the target time step $T'$ is computed from the context set. This involves forecasting the state representation for target time step from all the context time steps. After this, a decoder module makes predictions for the target input after drawing samples from the latent distribution and finally we empirically estimate the predictive mean and covariance.

We now describe our model that given context data $(x_{\mathcal{C}}, y_{\mathcal{C}})$ which spans from $t = 1$ to $t = T$ and a target set $x_{\mathcal{T}}$ with $t = T' > T$, computes the conditional distribution $P\left(y_{\mathcal{T}} \mid x_{\mathcal{T}}, x_{\mathcal{C}}, y_{\mathcal{C}}\right)$. Our proposed model is shown in fig. 4.4 and consists of the following modules:

- **Encoder**: An encoder $h$ transforms each context pair $(x_i^t, y_i^t)\ \forall\ t \in \{1, \ldots, T\}$ and $i \in \{1, \ldots, N_t\}$ into a context representation $r_i^t = h\left(x_i^t, y_i^t\right)$. We parameterise $h$ as a feedforward neural network. This encoder module is shared across all the time steps, however, one can have a different module for each time step too.
- **Aggregator**: After computing representation $r_i^t$ for each context pair, we combine all the



$q\left(z^{T'} \mid x_{\mathcal{C}}, y_{\mathcal{C}}\right)$
$q$
$(x_{\mathcal{C}}, y_{\mathcal{C}})$
$\mu_{\mathcal{T}|\mathcal{C}}, \Sigma_{\mathcal{T}|\mathcal{C}}$
$\hat{y}_k$
$g$
$\left(x_{\mathcal{T}}, z_k^{T'}\right)$
$z_k^{T'} \sim q\left(z^{T'} \mid x_{\mathcal{C}}, y_{\mathcal{C}}\right)$
$K$

Figure 4.3: The figure shows the schematic of Recurrent Neural Process model. Left: The latent distribution $q\left(z^{T'} \mid x_{\mathcal{C}}, y_{\mathcal{C}}\right)$ corresponding to the target time step $T'$ is computed from the context set $(x_{\mathcal{C}}, y_{\mathcal{C}})$. Right: The decoder makes predictions for the target input $x_{\mathcal{T}}$ by drawing $z_k^{T'}$ from the latent distribution. The predictive mean and covariance are then empirically estimated from the multiple predictions.

context representations for the same $t$ together into a state representation $r^t = a\left(\{r_i^t\}_{i=1}^{N_i}\right)$. This $r^t$ essentially summarises the state of the environment at time $t$ as can be determined from the corresponding context data pairs. This module is a simple mean aggregation function, i.e., $r^t = a\left(\{r_i^t\}_{i=1}^{N_i}\right) = \frac{1}{N_i}\sum_{i=1}^{N_i} r_i^t$.

- **Forecaster**: Given state representations $r^t$ for $t = 1, \ldots, T$, a forecaster module predicts the state representation for the target time step $T'$ as $r^{T'} = f\left(r^1, r^2, \ldots, r^T\right)$. We parameterise $f$ as a recurrent neural network.
- **Latent distribution encoder**: The state representation $r^{T'}$ computed by the forecaster module is used to parameterise the latent distribution $q\left(z^{T'} \mid x_{\mathcal{C}}, y_{\mathcal{C}}\right) = \mathcal{N}\left(\mu^{T'}, \sigma^{T'} I\right)$. The mean and standard deviation of this factorized Normal distribution are computed as learnable transformations of $r^{T'}$. Specifically, $\mu^{T'} = l_\mu\left(r^{T'}\right)$ and $\sigma^{T'} = l_\sigma\left(r^{T'}\right)$ where both $l_\mu$ and $l_\sigma$ are parameterised as feedforward neural networks.
- **Decoder**: The decoder $g$ takes an input the latent variable $z^{T'} \sim q\left(z^{T'} \mid x_{\mathcal{C}}, y_{\mathcal{C}}\right)$ and the target data point $x \in x_{\mathcal{T}}$ and outputs $y = g\left(x, z^{T'}\right)$. The latent distribution and the target data belongs to the same time step $T'$. It is also represented as a feedforward neural network.

## 4.3 Training

Our entire model comprises of some neural network modules and can be trained in an end-to-end manner. The model parameters are optimized to maximize the likelihood of target data

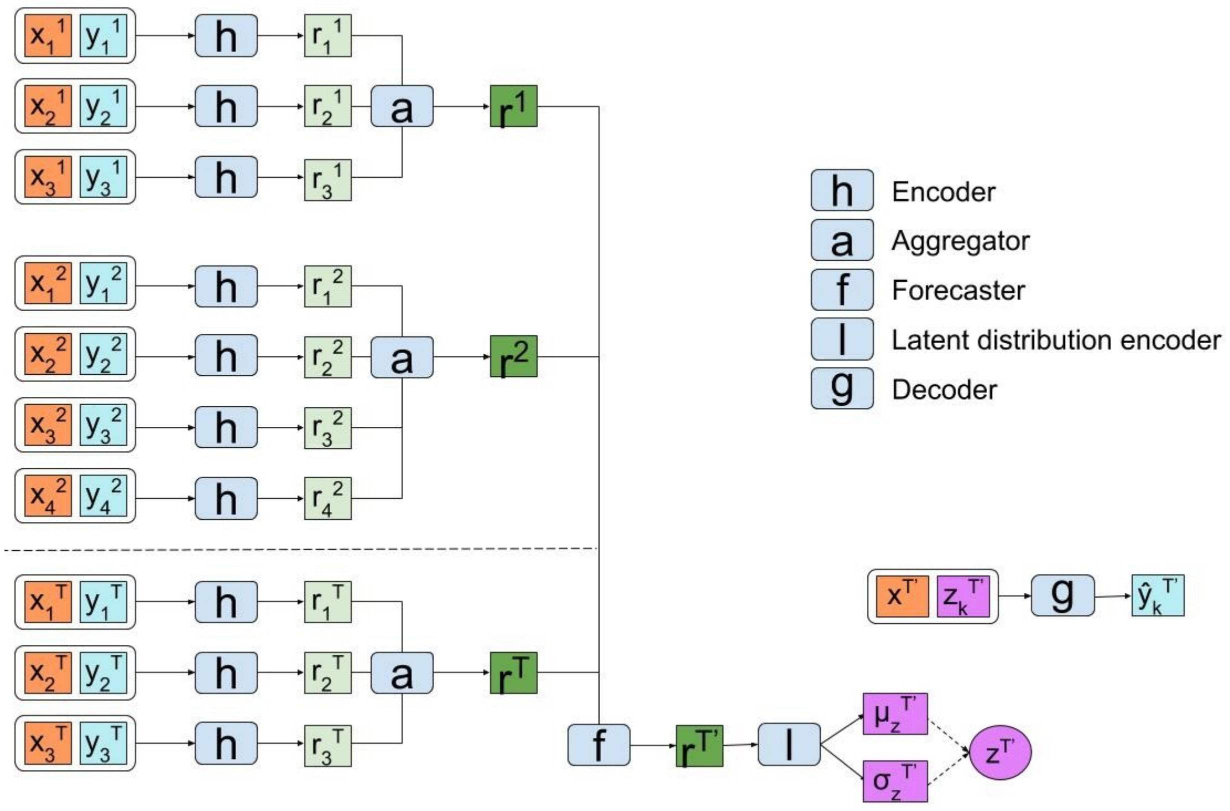


Figure 4.4: The figure describes our model architecture. The encoder $h$ forms a representation vector for each context input-output pair. The aggregator $a$ then combines representation vectors belonging to the same time step to form a state representation for each time step. The forecaster module $f$ computes the representation vector for the next time step $T'$. The latent distribution encoder $l$ takes as input the forecasted state representation and outputs the mean and variance vectors for a factorized normal latent distribution. Finally, the decoder $g$ takes as input a random vector sampled from the latent distribution and the target data point and predicts an output value for that data point.

conditioned on the context data set. We can write the conditional likelihood as:

$$\begin{aligned} P\left(y_{\mathcal{T}} \mid x_{\mathcal{T}}, x_{\mathcal{C}}, y_{\mathcal{C}}\right) &= \int_{z^{T'}} P\left(y_{\mathcal{T}} \mid x_{\mathcal{T}}, x_{\mathcal{C}}, y_{\mathcal{C}}, z^{T'}\right) P\left(z^{T'} \mid x_{\mathcal{T}}, x_{\mathcal{C}}, y_{\mathcal{C}}\right)\, dz^{T'} \\ &= \mathbb{E}_{z^{T'} \sim P\left(z^{T'} \mid x_{\mathcal{T}}, x_{\mathcal{C}}, y_{\mathcal{C}}\right)}\left[P\left(y_{\mathcal{T}} \mid x_{\mathcal{T}}, x_{\mathcal{C}}, y_{\mathcal{C}}, z^{T'}\right)\right] \end{aligned}$$

Since $P\left(z^{T'} \mid x_{\mathcal{T}}, x_{\mathcal{C}}, y_{\mathcal{C}}\right)$ is an intractable probability distribution, we used variational inference [2] to optimize the model parameters. We introduce a proposal distribution $q\left(z^{T'} \mid x_{\mathcal{C}}, y_{\mathcal{C}}\right)$ which depends only on the target set. This proposal distribution is a simple normal distribution with diagonal covariance matrix. The mean and variance of this distribution can be computed as a differentiable non-linear function of the context data pairs by using a neural network.

Now, instead of drawing samples from $P\left(z^{T'} \mid x_{\mathcal{T}}, x_{\mathcal{C}}, y_{\mathcal{C}}\right)$, we perform importance sampling with $q\left(z^{T'} \mid x_{\mathcal{C}}, y_{\mathcal{C}}\right)$ as shown:

$$P\left(y_{\mathcal{T}} \mid x_{\mathcal{T}}, x_{\mathcal{C}}, y_{\mathcal{C}}\right)=\mathbb{E}_{z^{T'} \sim q\left(z^{T'} \mid x_{\mathcal{C}}, y_{\mathcal{C}}\right)}\left[\frac{P\left(z^{T'} \mid x_{\mathcal{T}}, x_{\mathcal{C}}, y_{\mathcal{C}}\right)}{q\left(z^{T'} \mid x_{\mathcal{C}}, y_{\mathcal{C}}\right)} P\left(y_{\mathcal{T}} \mid x_{\mathcal{T}}, x_{\mathcal{C}}, y_{\mathcal{C}}, z^{T'}\right)\right]$$

Using Jensen's inequality, we can write:

$$\log P\left(y_{\mathcal{T}} \mid x_{\mathcal{T}}, x_{\mathcal{C}}, y_{\mathcal{C}}\right) \geq \mathbb{E}_{z^{T'} \sim q\left(z^{T'} \mid x_{\mathcal{C}}, y_{\mathcal{C}}\right)}\left[\log P\left(y_{\mathcal{T}} \mid x_{\mathcal{T}}, x_{\mathcal{C}}, y_{\mathcal{C}}, z^{T'}\right)+\log \frac{P\left(z^{T'} \mid x_{\mathcal{T}}, x_{\mathcal{C}}, y_{\mathcal{C}}\right)}{q\left(z^{T'} \mid x_{\mathcal{C}}, y_{\mathcal{C}}\right)}\right]$$

Even though we avoided sampling from $P\left(z^{T'} \mid x_{\mathcal{T}}, x_{\mathcal{C}}, y_{\mathcal{C}}\right)$ by introducing a proposal distribution and performing importance sampling, the computation of second term in the above equation is still infeasible. Since the conditional prior $P\left(z^{T'} \mid x_{\mathcal{T}}, x_{\mathcal{C}}, y_{\mathcal{C}}\right)$ is intractable, we approximate it with a variational posterior $q\left(z^{T'} \mid x_{\mathcal{T}}, y_{\mathcal{T}}\right)$ as shown:

$$\log P\left(y_{\mathcal{T}} \mid x_{\mathcal{T}}, x_{\mathcal{C}}, y_{\mathcal{C}}\right) \geq \mathbb{E}_{z^{T'} \sim q\left(z^{T'} \mid x_{\mathcal{C}}, y_{\mathcal{C}}\right)}\left[\log P\left(y_{\mathcal{T}} \mid x_{\mathcal{T}}, x_{\mathcal{C}}, y_{\mathcal{C}}, z^{T'}\right)+\log \frac{q\left(z^{T'} \mid x_{\mathcal{T}}, y_{\mathcal{T}}\right)}{q\left(z^{T'} \mid x_{\mathcal{C}}, y_{\mathcal{C}}\right)}\right]$$

Hence, the objective function $J$ can be written as:

$$\begin{aligned}
J &= \mathbb{E}_{q\left(z^{T'} \mid x_{\mathcal{C}}, y_{\mathcal{C}}\right)}\left[\sum_{i=1}^{M} \log P\left(y_i^{T'} \mid x_i^{T'}, z^{T'}\right)+\log \frac{q\left(z^{T'} \mid x_{\mathcal{T}}, y_{\mathcal{T}}\right)}{q\left(z^{T'} \mid x_{\mathcal{C}}, y_{\mathcal{C}}\right)}\right] \\
&= \mathbb{E}_{q\left(z^{T'} \mid x_{\mathcal{C}}, y_{\mathcal{C}}\right)}\left[\sum_{i=1}^{M} \log P\left(y_i^{T'} \mid x_i^{T'}, z^{T'}\right)\right]-D_{KL}\left[q\left(z^{T'} \mid x_{\mathcal{C}}, y_{\mathcal{C}}\right) \| q\left(z^{T'} \mid x_{\mathcal{T}}, y_{\mathcal{T}}\right)\right]
\end{aligned} \tag{4.1}$$

For computing the variational posterior $q\left(z^{T'} \mid x_{\mathcal{T}}, y_{\mathcal{T}}\right)$, the encoder first computes the representation for each data pair $(x_i^{T'}, y_i^{T'})$ where $1 \leq i \leq N$. The aggregator module then combines these together into a state representation for time $T'$. Finally, the latent distribution encoder computes the mean and variance of the variational posterior distribution.

Note that the first term in eq. 4.1 is the expected likelihood of target data given context data pairs and latent distribution sample $z^{T'}$ whereas the second term is the kl-divergence between the forecasted latent distribution computed from the context set and the latent distribution obtained from the target set. During training, the model tries to maximize the conditional likelihood while minimizing the divergence between the two latent distributions.

# Chapter 5

# Experiments

## 5.1 Datasets

We use three spatiotemporal datasets to evaluate the performance of our model:

- **Moving Ellipses**: An ellipse in a bounding box of size $20 \times 20$ moves in a window of size $32 \times 32$. The dimensions and brightness intensity of ellipse are randomly selected and kept same throughout the motion sequence. The ellipse is given a random direction at the start and it bounces off the walls on colliding. An example sequence of $10$ time steps is shown in figure 5.1.

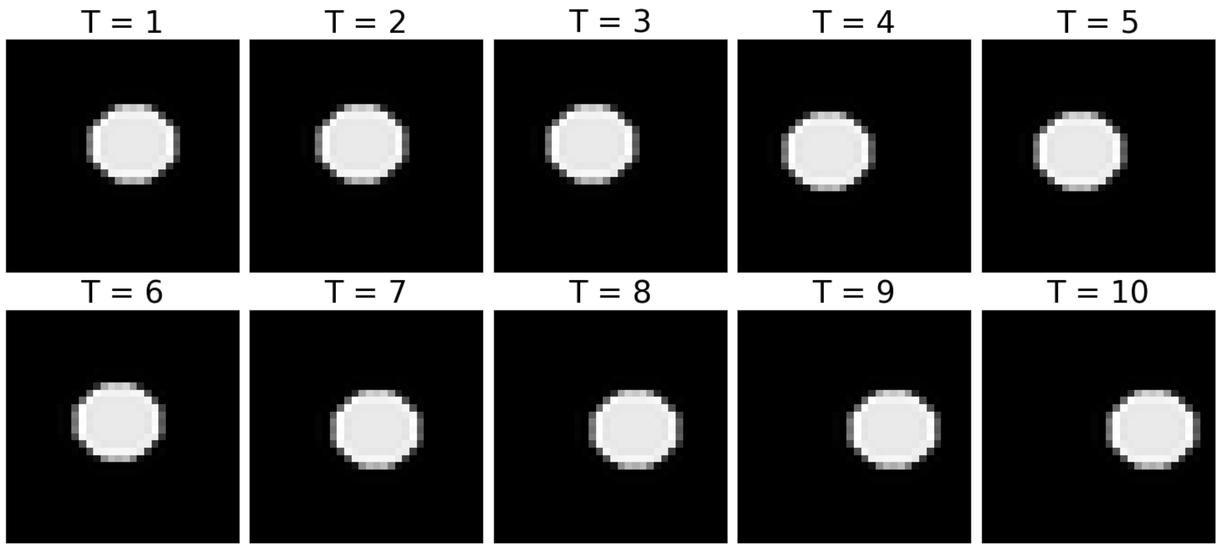


Figure 5.1: The figure shows $10$ consecutive time steps of one sequence from Moving Ellipses dataset.

- **Moving MNIST**: This dataset is similar to the last one except that instead of an ellipse, we have handwritten images from the MNIST [20] dataset. Unlike the originally proposed Moving MNIST dataset [30], here we have only one digit in a $32 \times 32$ image. We show one sequence from the dataset in figure 5.2.

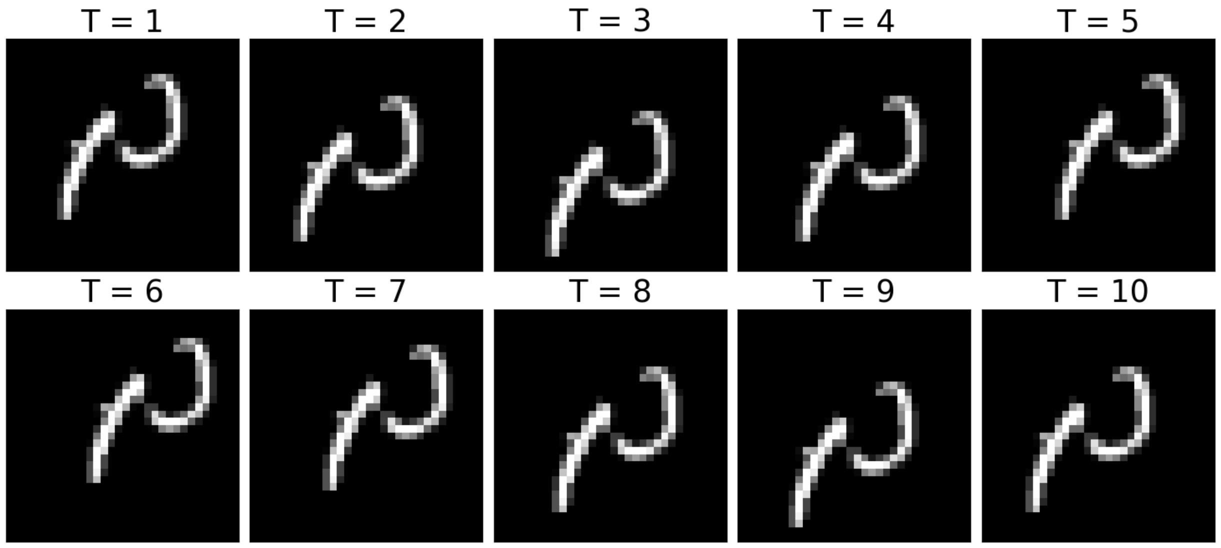


Figure 5.2: The figure shows 10 consecutive time steps of one sequence from Moving MNIST dataset.

- **Temperature Dataset** [1]: This dataset contains monthly average temperature readings across the whole world recorded since 1948. The data has an spatial coverage of 2.5° latitude ×2.5° longitude. For this work, we selected the area between 10°S - 70°N and 60° W - 140° W (North America region) forming a square grid of size $32 \times 32$. We show one sequence from this dataset in figure 5.3.

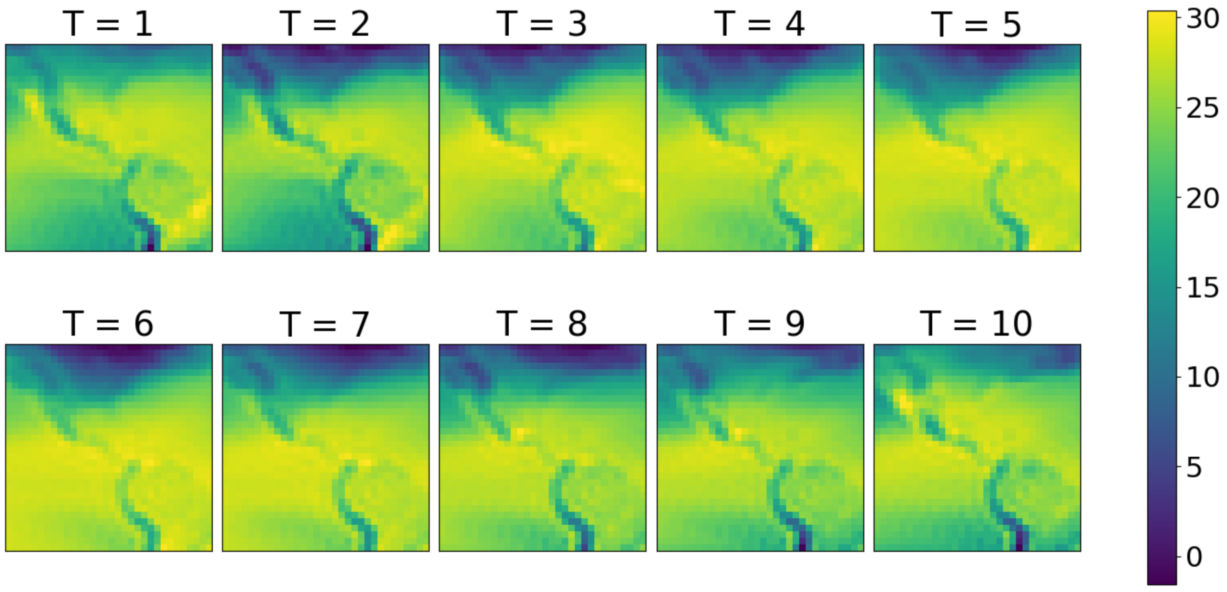


Figure 5.3: The figure shows 10 consecutive time steps of one sequence from Temperature dataset. The colorbar on the right shows the temperature in degree Celsius. Here, one time step represents one month as each image shows the average temperature for a month.

[1] `https://www.esrl.noaa.gov/psd/data/gridded/data.ncep.reanalysis.derived.surface.html`

The first two datasets are synthetic and consists of an object moving around in an area. The Moving Ellipses dataset has a simple regular shape (ellipses) whereas the Moving MNIST one consists of complex shapes (handwritten digits). In order to make accurate predictions, the model needs to learn both the spatial and temporal dependencies from the observed data.

For the Moving Ellipses and Moving MNIST datasets, we generated $20000$ sequences of $20$ time steps each. We used $15000$ sequences for training and the rest for evaluation. For the temperature dataset, we used the first three quarters of the data for training and the rest for evaluation purposes. We also normalized this dataset in the range of $0$ to $1$.

## 5.2 Implementation Details

We now describe the architecture of our model. The encoder $h$ is a 3 layer fully connected (FC) neural network with $256$ neurons in each layer. The forecaster module $f$ is a single LSTM [13] cell with $256$ neurons. The latent distribution encoder networks, $l_\mu$ and $l_\sigma$, are single FC layers with $256$ neurons. The decoder $g$ is a 2 layer FC network with $256$ neurons in each layer. All the networks use leaky ReLU non-linear activation function with a negative slope of $0.01$. All the model parameters follow Xavier [11] normal initialization. We used Adam [16] optimizer with a learning rate of $0.001$. Our entire model is built in PyTorch [24]. We train the model for $200$ epochs and after $100$ epochs, we reduce the learning rate to $0.0001$.

The model is trained for $T = 10$ time steps to maximize the objective function in eq. 4.1 with a batch of $64$ sequences. For each sequence, we select $N$ cells from the grid for each $1 \leq t \leq T$ as the context set and the entire $32 \times 32$ grid, i.e., $1024$ data points at time $T + 1$ as the target set, i.e., $T' = T + 1$. For each data pair, the input is the 2D coordinates of the pixel/location and the output is the intensity value at that location.

We illustrate the context set and the target set for an example sequence from Moving MNIST dataset in figure 5.4. The non-blue pixels' coordinates with their corresponding intensity values in the first $10$ frames form the context set whereas the entire grid at $11^{th}$ time step is the target set. Note that the context set locations are selected randomly from the entire grid. The blue cells mean that the model is oblivious regarding the true value at those locations for that time step. We kept $N_t = N \ \forall \ t$ just to facilitate batch-wise parallel matrix operations on GPU, however, the model can also be trained with different number of samples at each time step. The mean and standard deviation plots are computed from $20$ outputs obtained from sampling different latent variable $z_k^{T'} \sim q\left(z^{T'} \mid x_C, y_C\right)$ and then feeding to the decoder.

## 5.3 Baselines

We compare our proposed model, Recurrent Neural Processes (RNP), against the following baseline methods:

- RBF Kernel GP (GP): This is a simple GP model with radial basis function (RBF) kernel. We use Automatic Relevance Determination (ARD) for the kernel, i.e., each data

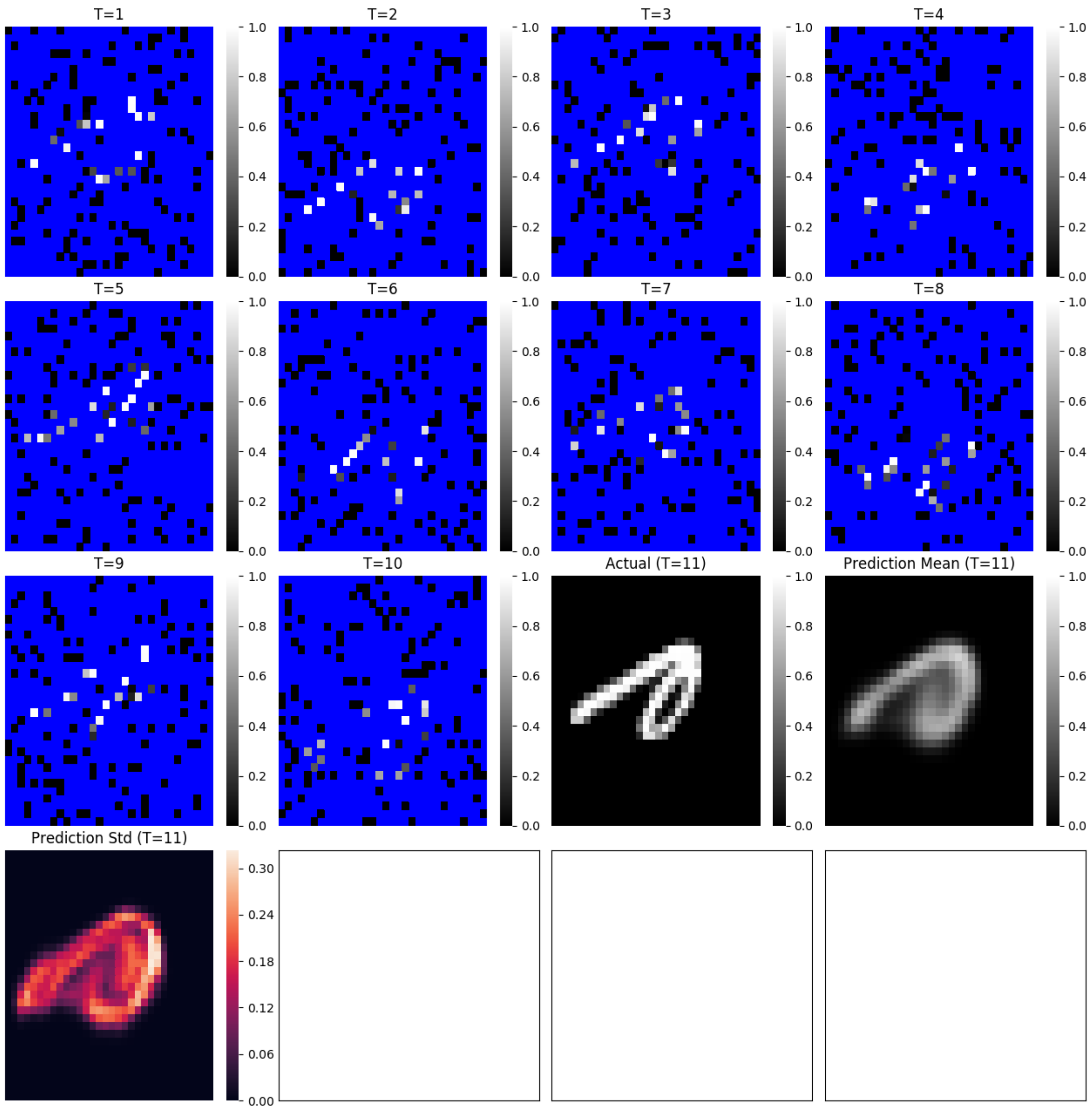


Figure 5.4: In the first 10 images, the coordinates of non-blue pixels and their corresponding intensity values form the context set. This set consists of $N = 128$ input-output pairs for $1 \leq t \leq 10$. The target set consists of the entire grid at $t = 11$. The model's predicted mean and variance plots are generated from 20 outputs, each obtained as a result of sampling different random vector $z$ from the latent distribution.

dimension has a separate lengthscale as shown:

$$k_{RBF}(\tau) = \exp\left(-\sum_{d=1}^{D}\left(\frac{\tau_d}{l^{(d)}}\right)^2\right)$$

where $\tau_d$ is the $d^{th}$ component of $D$ dimensional vector $\tau = x - x'$ and the lengthscale $l = \left(l^{(1)}, \ldots, l^{(D)}\right)$ is a vector of size as input $x \in \mathbb{R}^D$.

- Spectral Mixture Kernel GP (SMGP): This kernel was proposed in [33] to be suitable for spatiotemporal data as it can discover patterns and achieve extrapolation behavior. The functional form of the kernel can be written as:

$$k_{SM}(\tau) = \sum_{q=1}^{Q} w_q \prod_{d=1}^{D} \exp\left(-2\pi^2\tau_d^2 v_q^{(d)}\right)\cos\left(2\pi\tau_d\mu_q^{(d)}\right)$$

  Essentially, the kernel is defined as a mixture of $Q$ Gaussians on $\mathbb{R}^D$. Each component has a mean $\mu_q = \left(\mu_q^{(1)}, \ldots, \mu_q^{(D)}\right)$ and diagonal covariance $v_q = \left(v_q^{(1)}, \ldots, v_q^{(D)}\right)$. The weights $w_q$ specify the relative contribution of each mixture component. We set the number of mixture components $Q$ to be $4$.

- Deep RBF Kernel GP (DKGP): An input data point $x \in \mathbb{R}^D$ is transformed into a representation space $\mathbb{R}^P$ by using a non-linear function $g : \mathbb{R}^D \to \mathbb{R}^P$. This function is parameterised as a 2 layer FC neural network with $16$ neurons in each layer. As before, we use leaky ReLU activation function with a negative slope of $0.01$. The $16$ dimensional representation vectors are then fed into a standard RBF kernel with ARD.

$$k_{DRBF}(x, x') = k_{RBF}\left(g(x), g(x')\right) = \exp\left(-\sum_{p=1}^{P}\left(\frac{\gamma_p}{l^{(p)}}\right)^2\right)$$

  where, $\gamma = g(x) - g(x')$ and $\gamma \in \mathbb{R}^P$.

- Deep GP (DGP): This model was proposed by [5] to incorporate the rich representation power of neural networks in Gaussian Processes. We used a single hidden layer with the number of latent dimensions as 12. Here also, we used RBF kernel with ARD.

Since, the data is not a sequence of inputs but rather samples from a 3D spatio-temporal process, recurrent GP models [1, 22], that take an input sequence, are not applicable for comparison. The design specifications of all the GP models are determined by the memory requirements of storing all the data and parameters on a GPU. We used publicly available implementation of Deep GPs[2] and the other GP models are build using the open-source GPyTorch[3] [9] library. For all the GP methods, the input is a set of 3D (2D grid coordinates and time $t$) vectors.

Our model RNP has been trained on the entire training set and then evaluated on a held-out test set. On the other hand, the parameters of GP models have been optimized to maximize the likelihood of the context set of points from a given data sequence. The learned model is then evaluated on the target set of points from that sequence only. This means that the GP models learn a different set of parameters for each example sequence.

[2]`https://github.com/SheffieldML/PyDeepGP`

[3]`https://github.com/cornellius-gp/gpytorch`

## 5.4 Results

### 5.4.1 Comparison with Baselines

To compare all the models, we compute the mean absolute error (MAE) between model's prediction and the ground truth on $20$ sequences. The corresponding results are shown in Table 5.1.

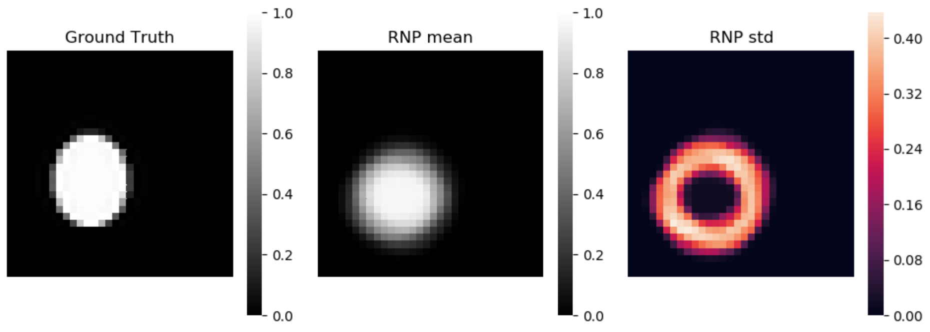
Table 5.1: Mean Absolute Error between ground truth and predictions obtained by different models evaluated on a batch of $20$ sequences. The number reported is the average MAE obtained across the batch whereas the associated standard deviation is mentioned in the parentheses.

| DATASET | N | GP | SMGP | DKGP | DGP | RNP |
|---|---|---|---|---|---|---|
| ELLIPSES | 64 | 0.22 | 0.15 | 0.16 | 0.17 | **0.04** |
| ELLIPSES | 128 | 0.21 | 0.18 | 0.15 | 0.18 | **0.03** |
| ELLIPSES | 256 | 0.86 | 0.18 | 0.63 | 0.13 | **0.03** |
| ELLIPSES | 512 | 1.51 | 0.17 | 1.20 | 0.29 | **0.02** |
| MOVING MNIST | 64 | 0.18 | 0.15 | 0.21 | 0.16 | **0.10** |
| MOVING MNIST | 128 | 0.26 | 0.15 | 0.24 | 0.16 | **0.08** |
| MOVING MNIST | 256 | 0.23 | 0.18 | 0.19 | 0.24 | **0.07** |
| MOVING MNIST | 512 | 0.29 | 0.16 | 0.19 | 0.34 | **0.06** |
| TEMPERATURE | 64 | 0.79 | 0.31 | 1.06 | 0.10 | **0.025** |
| TEMPERATURE | 128 | 0.95 | 0.80 | 0.92 | 0.16 | **0.024** |
| TEMPERATURE | 256 | 1.37 | 1.06 | 1.16 | 0.20 | **0.024** |
| TEMPERATURE | 512 | 2.00 | 1.57 | 1.42 | 0.20 | **0.024** |

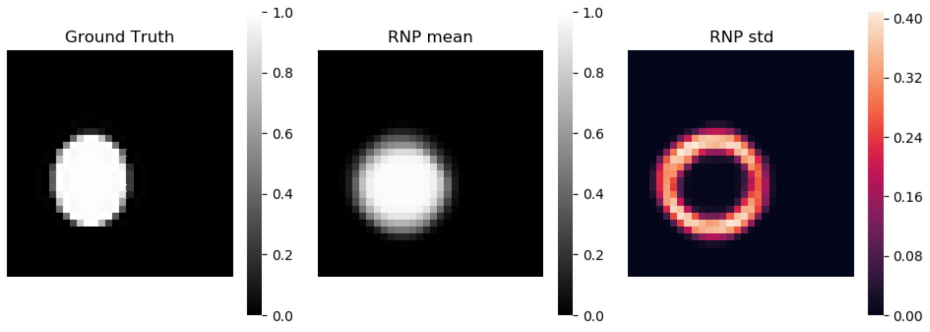
All the GP models showed poor results in general and their predictive performance varied a lot across different sequences as evident from the high standard deviation in their error metric. This stems from the limited functional form of the covariance function which is incapable of modelling the complex spatial and temporal dependencies together. With deep kernels, some improvement in MAE is seen, however even that is comparatively quite high than the ones obtained by our model.

Our model, RNP, achieved the lowest MAE across all the three datasets and for all the values of $N$. Our model learns implicit kernel or covariance function from the training data itself and hence are able to capture the complex spatial and temporal dependencies.

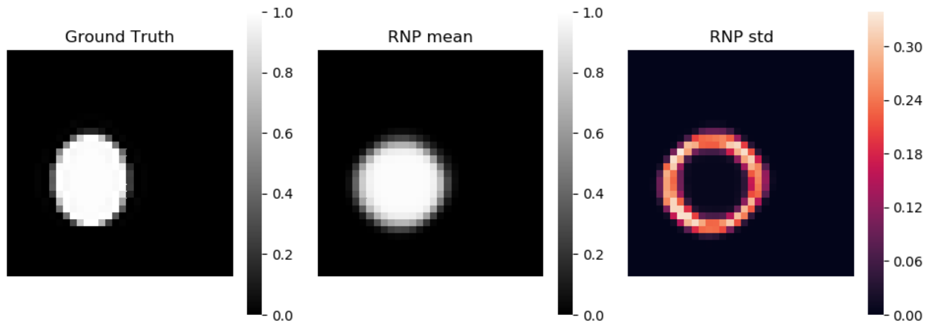
As stated before, the aim of this work is to build model that, in addition to a mean estimate, can also provide meaningful uncertainty estimates. We show some graphic results of predictions made by RNP on sequences from Moving Ellipses dataset in figures 5.5 and 5.6. Similarly, a couple of plots of predictions made by the two models on sequences from Moving MNIST dataset are shown in figures 5.7 and 5.8. The mean and standard deviation plots for RNP are computed from $20$ outputs, each obtained by sampling different latent variable.



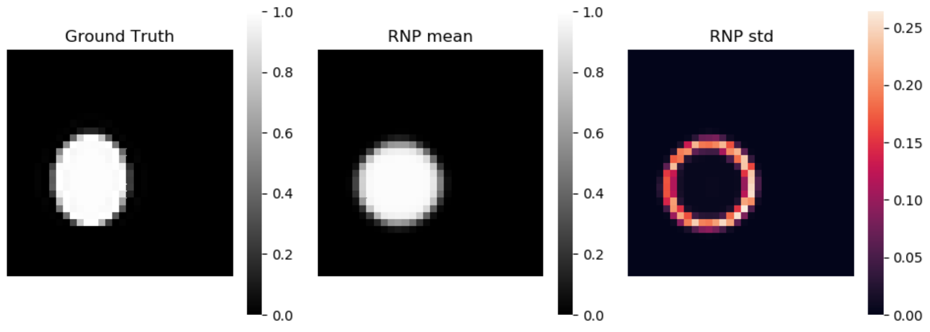
(a) $N = 64$



(b) $N = 128$

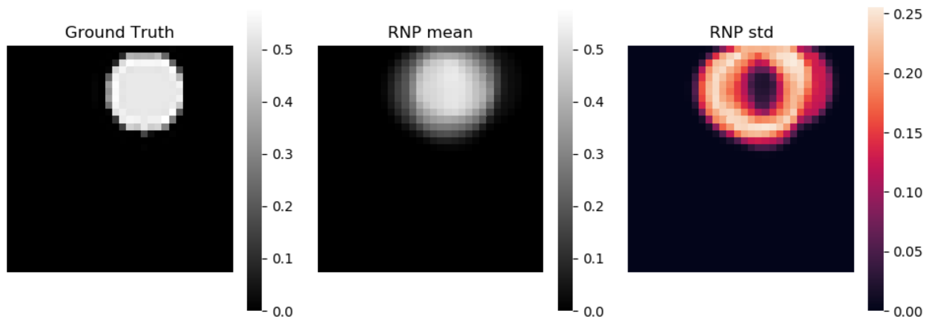


(c) $N = 256$

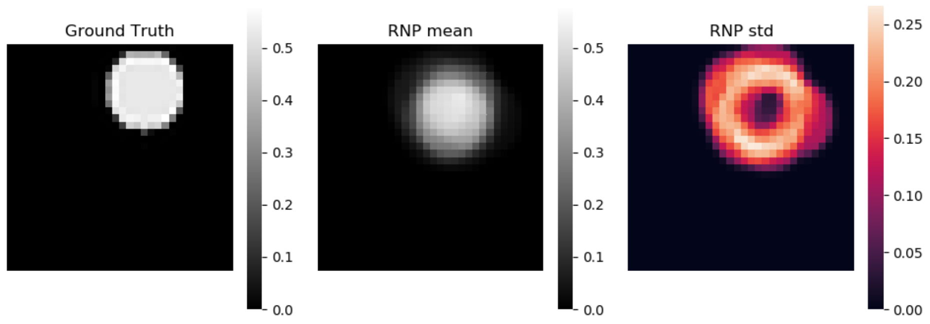


(d) $N = 512$

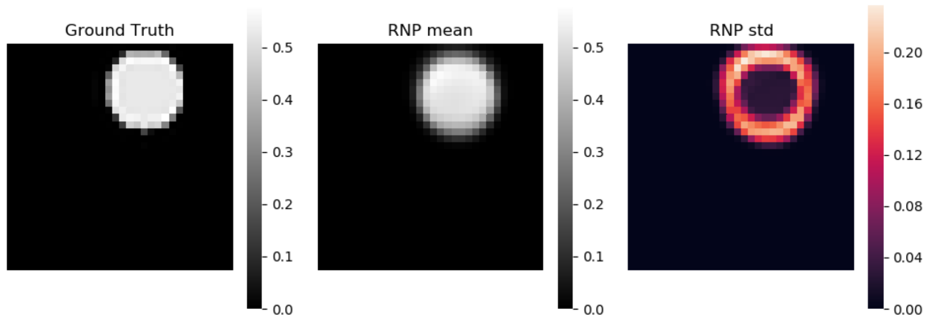
Figure 5.5: Mean and standard deviation of predictions made by RNP model for different values of $N$ on a sequence from Moving Ellipse dataset.



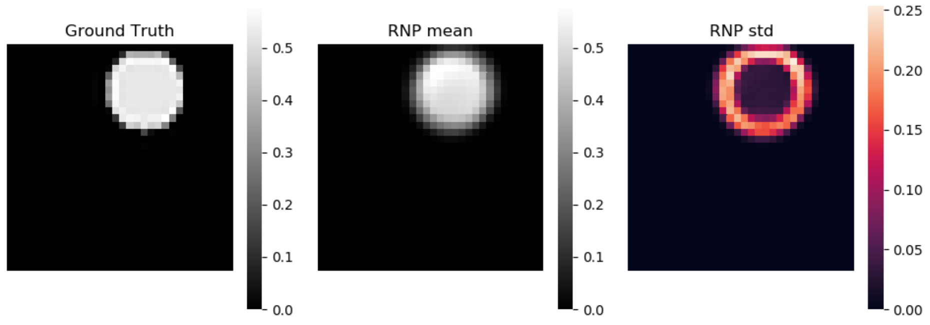
(a) $N = 64$



(b) $N = 128$

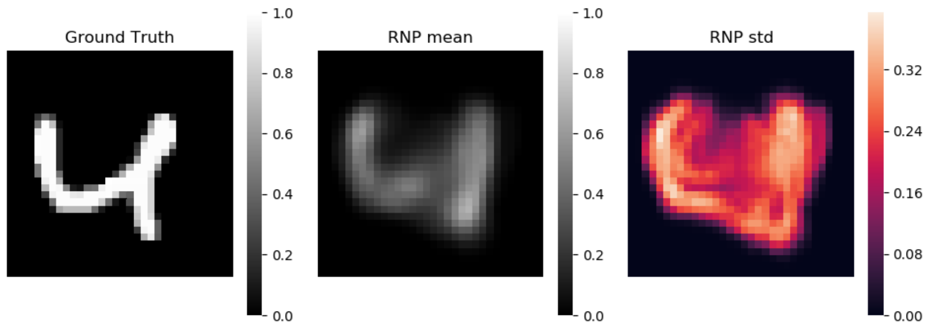


(c) $N = 256$

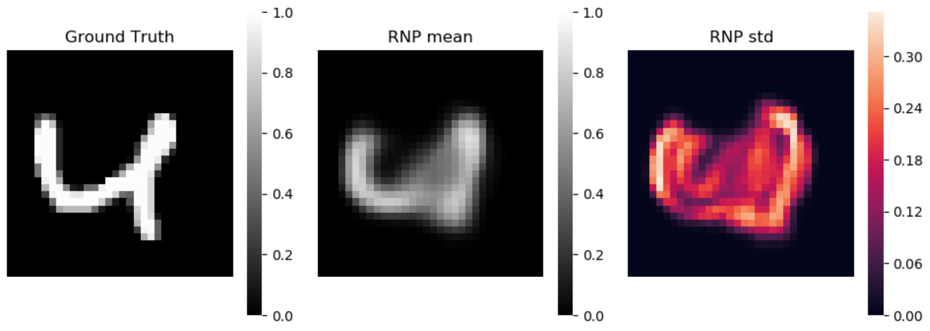


(d) $N = 512$

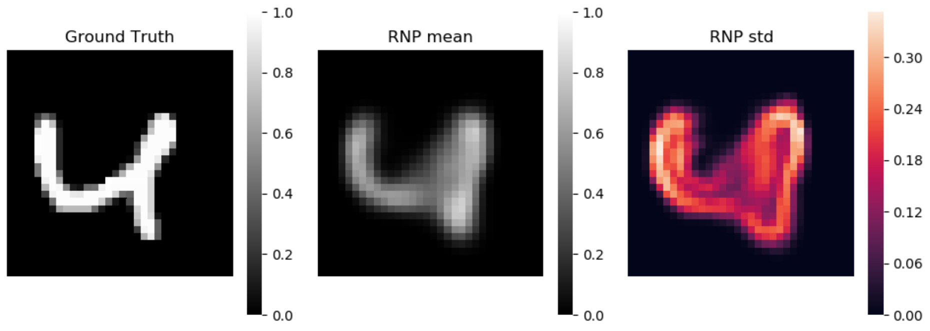
Figure 5.6: Mean and standard deviation of predictions made by RNP model for different values of $N$ on a sequence from Moving Ellipse dataset.



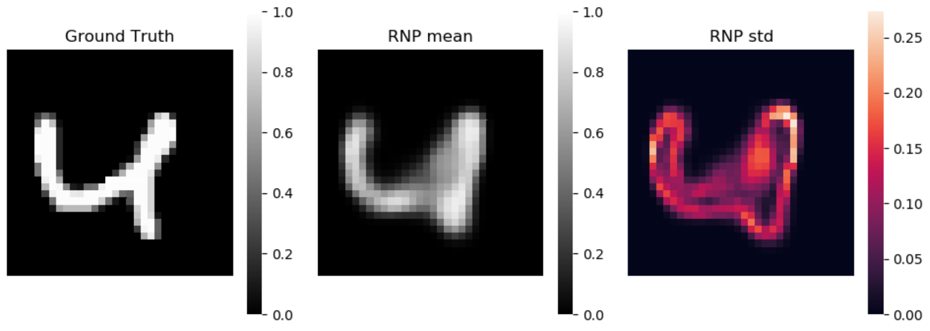
(a) $N = 64$



(b) $N = 128$

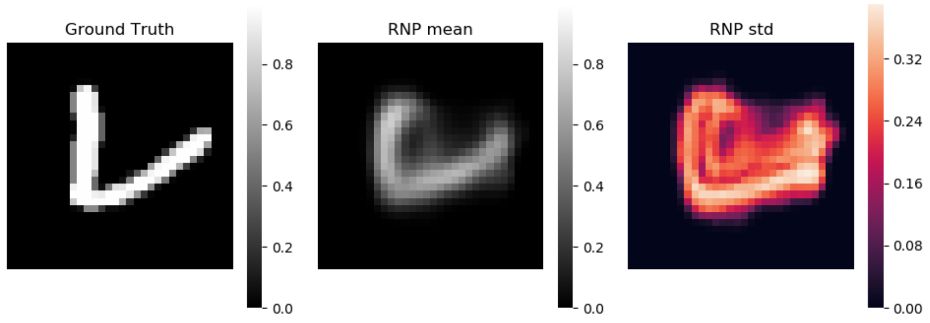


(c) $N = 256$

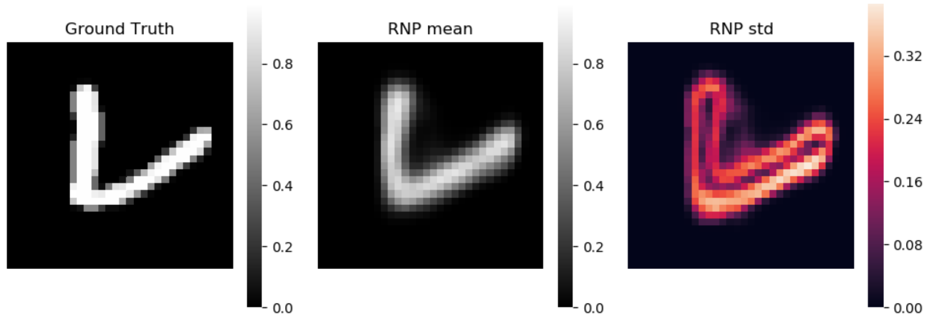


(d) $N = 512$

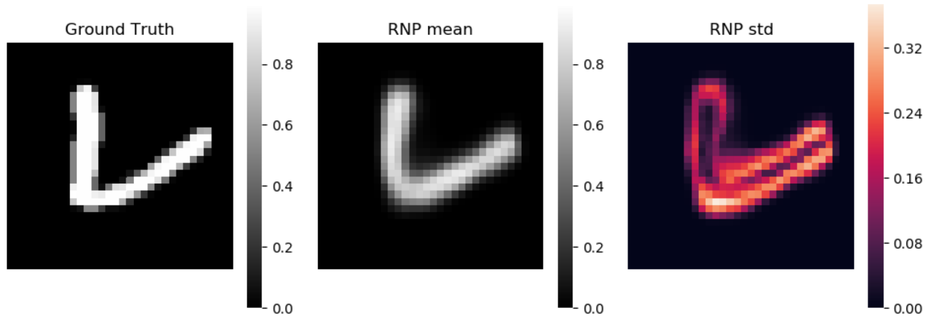
Figure 5.7: Mean and standard deviation of predictions made by RNP model for different values of $N$ on a sequence from Moving MNIST dataset.



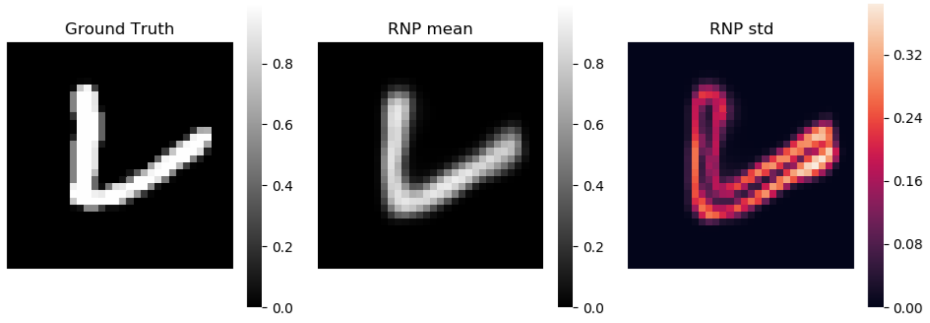
(a) $N = 64$



(b) $N = 128$



(c) $N = 256$



(d) $N = 512$

Figure 5.8: Mean and standard deviation of predictions made by RNP model for different values of $N$ on a sequence from Moving MNIST dataset.

As can be seen from the plots, the mean value of RNP predictions closely resembles the ground truth. Furthermore, RNP model gives coherent structured uncertainty estimates which improves (decreases) with the increase in the number of context data points ($N$). The model's uncertainty is concentrated in regions with sharp change in brightness, i.e., near the edges. In regions far from the object (digit/ellipse), the model is confident of its predictions and does not show any uncertainty. Also, the model is also confident of its predictions in the center areas of the object.

### 5.4.2 Uncertainty prediction analysis

We observed in the last section, that with increase in context set size, the model's uncertainty estimate gets better qualitatively. We now do a quantitative analysis of the same. For Moving MNIST dataset, we computed the number of pixels with model's uncertainty (standard deviation of prediction) more than some threshold like $0.1$, $0.2$ and $0.3$. The experimental setup remains the same as before. For each of $10$ context time steps, we randomly sample $N$ locations from the grid and form the context set. The model then predicts mean and uncertainty for all the locations at $t = 11$. Note that we have a total of $1024$ locations/pixels. The corresponding result is shown in fig. 5.9.

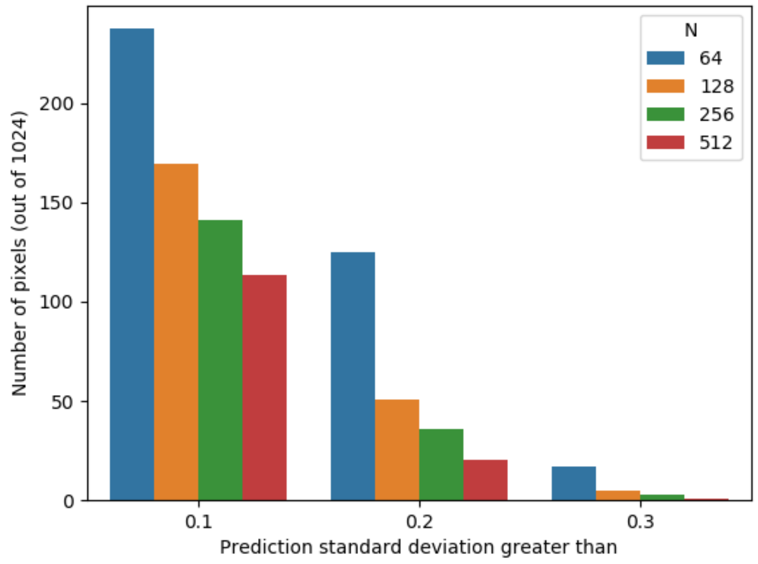


Figure 5.9: With increase in $N$, the number of grid locations with uncertainty greater than any threshold decreases.

With increase in the context set size, the number of target inputs for which the model's uncertainty is greater than a threshold decreases. This indicates that the model becomes more and more confident in its predictions and its uncertainty gets concentrated in regions near the object as we saw previously.

### 5.4.3 Model's performance against context set size

In the last section, we observed that the model's prediction (both mean and uncertainty) improves with increase in number of context set data points per time step, so a natural question to ask is how much can the performance improve with increase in $N$ ?

We evaluate the mean absolute error of the model's prediction as a function of $N$. The corresponding plot for the Moving Ellipses dataset can be found in figure 5.10. Similarly, the plots for the Moving MNIST and the Temperature datasets are shown in figures 5.11 and 5.12 respectively. All of these plots are obtained from 20 test example sequences evaluated

On a grid with 1024 cells, the model's performance seems to saturate after sampling about half of all the cells for Moving Ellipses and Moving MNIST datasets. In contrast, for the Temperature dataset, the peak performance is achieved bit earlier, only with 400 samples per time step. This is due to the simplicity of this dataset compared to the two synthetic ones as the average monthly temperature distribution will not show as many variations as differently shaped objects moving in an area, thus making it easier for the deep network to capture all the dependencies.

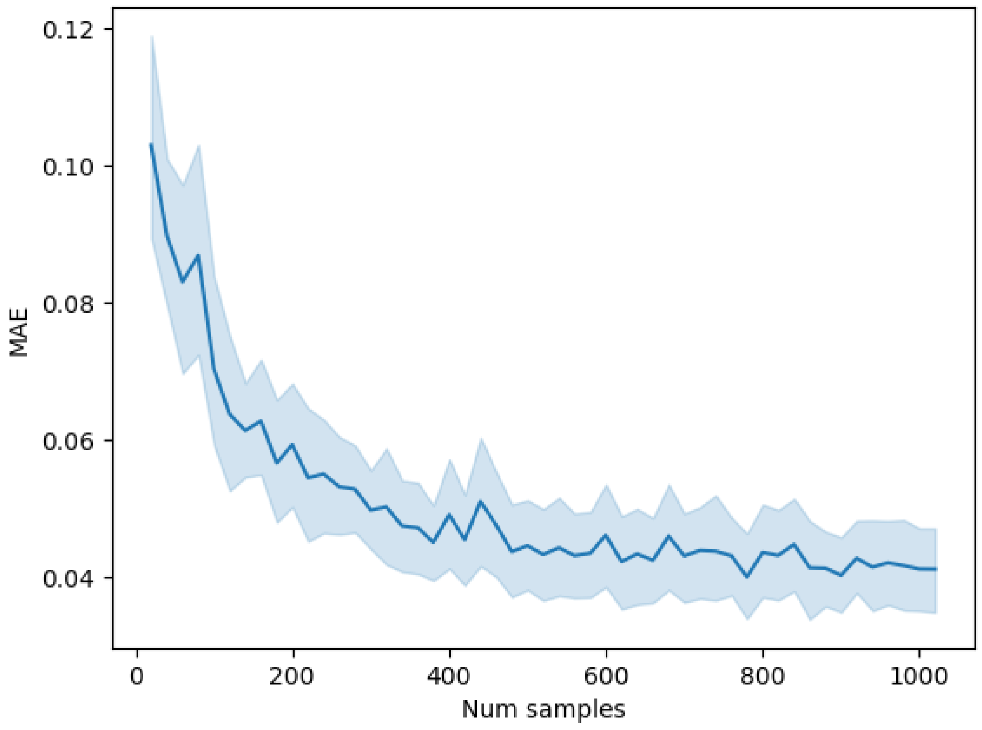


Figure 5.10: The plot of MAE of model's prediction against $N$ averaged over 20 test sequences from Moving Ellipses dataset.

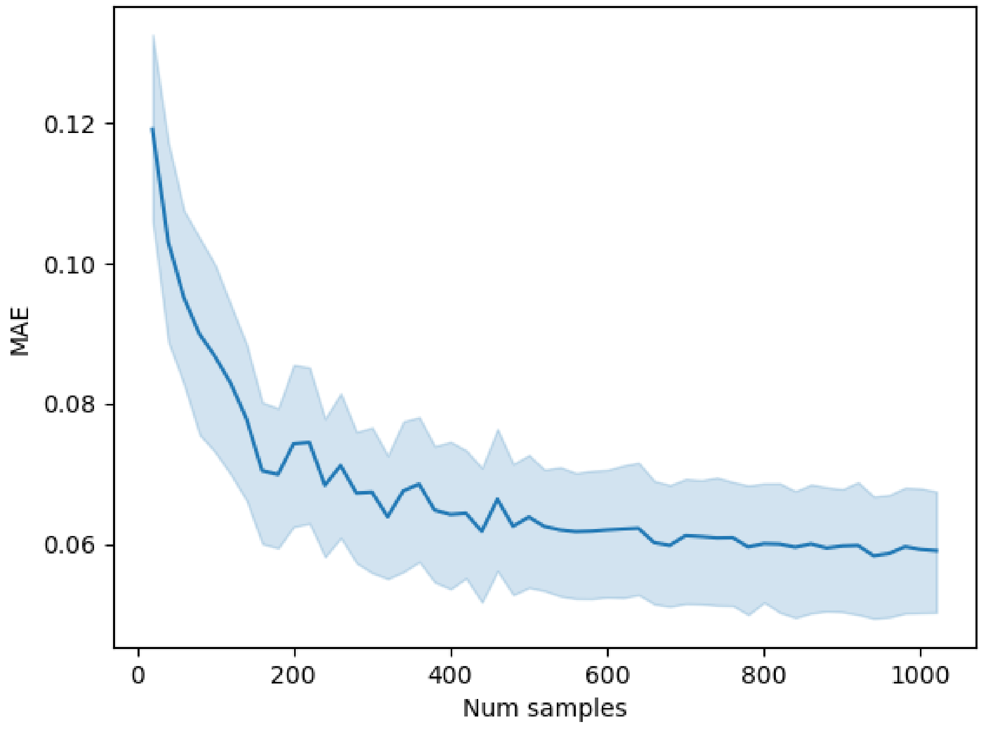


Figure 5.11: The plot of MAE of model's prediction against $N$ averaged over 20 test sequences from Moving MNIST dataset.

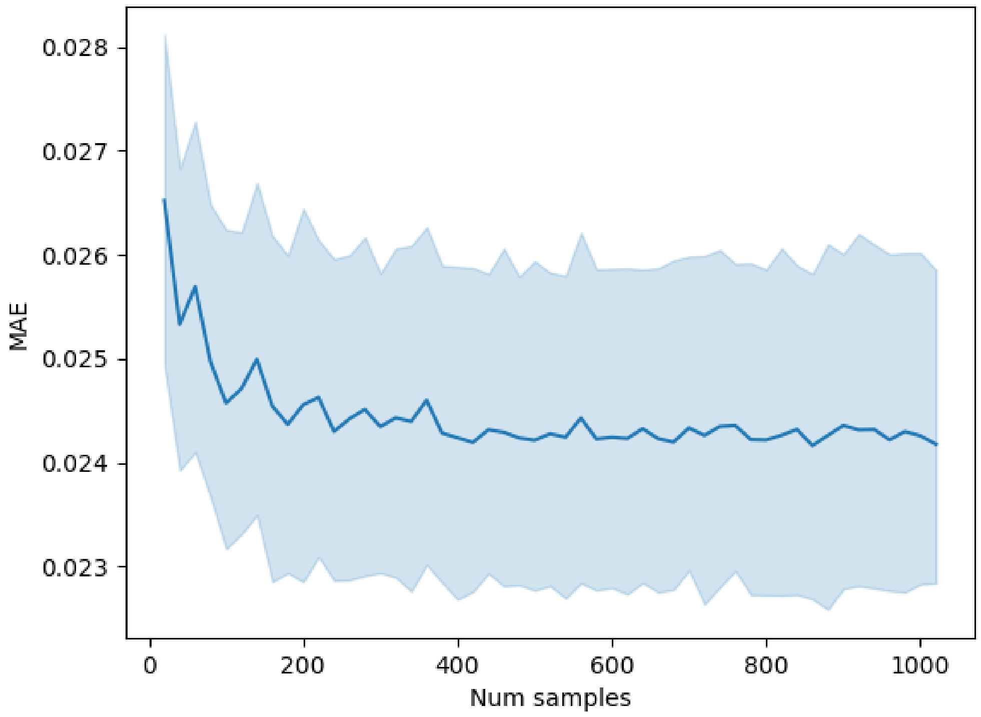


Figure 5.12: The plot of MAE of model's prediction against $N$ averaged over 20 test sequences from Temperature dataset.

# Chapter 6

# Conclusion and Future Work

We proposed a deep latent variable model, Recurrent Neural Processes, suitable for modeling spatiotemporal data where we only have access to individual samples from some time steps instead of complete images or feature maps for each time step. Our proposed model outperformed GP-based models and dropout-based deep network models by achieving low absolute errors in predictions on all three datasets. Additionally, our model produced meaningful structured uncertainty estimates.

Recently, Kim et al. [15] proposed Attentive Neural Processes where the state representation is computed by using self-attention [32], instead of simple averaging, among the context representations. Furthermore, the decoder module used cross-attention between target and context set to allow each target data point to attend to relevant context points for making predictions. However, since in our case, context set and target set belong to different time steps, simple cross-attention mechanism can not be used. Our attempts to use only self-attention for computing state representations did not yield any benefits. Applying cross-attention over time (spatiotemporal cross-attention) where context points from different time steps can attend to the target set is an interesting future direction in our work.

We also tested our model on the original Moving MNIST dataset [30] where there are $2$ digits in a $64 \times 64$ frame. Due to the presence of large empty spaces and the use of mean aggregation function for computing state representations, which acts as a representational bottleneck, the model produced blurred outputs. Although, the locations of two digits were correctly identified as blobs, the fine features were missing. Extending the model to large spatial state space is another interesting research direction.

We also intend to utilize the uncertainty estimates in downstream tasks such as adaptive sampling of a spatiotemporal phenomena where an autonomous agent will select the places to gather data from in order to actively learn the underlying process.

# Bibliography

[1] Maruan Al-Shedivat, Andrew Gordon Wilson, Yunus Saatchi, Zhiting Hu, and Eric P Xing. Learning scalable deep kernels with recurrent structure. *arXiv preprint arXiv:1610.08936*, 2016. 5.3

[2] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, 2017. 4.3

[3] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015. 3.2

[4] Christopher JC Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167, 1998. 3.1

[5] Andreas Damianou and Neil Lawrence. Deep gaussian processes. In *Artificial Intelligence and Statistics*, pages 207–215, 2013. 3.1, 5.3

[6] Nemanja Djuric, Vladan Radosavljevic, Henggang Cui, Thi Nguyen, Fang-Chieh Chou, Tsung-Han Lin, and Jeff Schneider. Motion prediction of traffic actors for autonomous driving using deep convolutional networks. *arXiv preprint arXiv:1808.05819*, 2018. 1

[7] Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. In *Advances in neural information processing systems*, pages 64–72, 2016. 1

[8] Meire Fortunato, Charles Blundell, and Oriol Vinyals. Bayesian recurrent neural networks. *arXiv preprint arXiv:1704.02798*, 2017. 3.2

[9] Jacob Gardner, Geoff Pleiss, Kilian Q Weinberger, David Bindel, and Andrew G Wilson. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. In *Advances in Neural Information Processing Systems*, pages 7576–7586, 2018. 5.3

[10] Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J Rezende, SM Eslami, and Yee Whye Teh. Neural processes. *arXiv preprint arXiv:1807.01622*, 2018. 3.2.1

[11] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010. 5.2

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1

[13] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 5.2

[14] Ashesh Jain, Amir R Zamir, Silvio Savarese, and Ashutosh Saxena. Structural-rnn: Deep learning on spatio-temporal graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5308–5317, 2016. 1

[15] Hyunjik Kim, Andriy Mnih, Jonathan Schwarz, Marta Garnelo, Ali Eslami, Dan Rosenbaum, Oriol Vinyals, and Yee Whye Teh. Attentive neural processes. *arXiv preprint arXiv:1901.05761*, 2019. 6

[16] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 5.2

[17] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013. 3.2.1

[18] Andreas Krause, Ajit Singh, and Carlos Guestrin. Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research*, 9(Feb):235–284, 2008. 1

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 1

[20] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 5.1

[21] William Lotter, Gabriel Kreiman, and David Cox. Deep predictive coding networks for video prediction and unsupervised learning. *arXiv preprint arXiv:1605.08104*, 2016. 1

[22] César Lincoln C Mattos, Zhenwen Dai, Andreas Damianou, Jeremy Forth, Guilherme A Barreto, and Neil D Lawrence. Recurrent gaussian processes. *arXiv preprint arXiv:1511.06644*, 2015. 5.3

[23] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015. 1

[24] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017. 5.2

[25] Geoff Pleiss, Jacob R Gardner, Kilian Q Weinberger, and Andrew Gordon Wilson. Constant-time predictive distributions for gaussian processes. *arXiv preprint arXiv:1803.06058*, 2018. 3.1

[26] Joaquin Quiñonero-Candela and Carl Edward Rasmussen. A unifying view of sparse approximate gaussian process regression. *Journal of Machine Learning Research*, 6(Dec): 1939–1959, 2005. 3.1

[27] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pages 63–71. Springer, 2003. (document), 1, 3.1

[28] Xingjian SHI, Zhourong Chen, Hao Wang, Dit-Yan Yeung, and Wai-kin Wong. W.-c. woo. convolutional lstm network: A machine learning approach for precipitation nowcasting. *Advances in Neural Information Processing Systems*, 28:802–810, 2015. 1

[29] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016. 1

[30] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. Unsupervised learning of video representations using lstms. In *International conference on machine learning*, pages 843–852, 2015. 5.1, 6

[31] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015. URL `http://arxiv.org/abs/1409.4842`. 1

[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017. 6

[33] Andrew Wilson and Ryan Adams. Gaussian process kernels for pattern discovery and extrapolation. In *International Conference on Machine Learning*, pages 1067–1075, 2013. 5.3

[34] Andrew Wilson and Hannes Nickisch. Kernel interpolation for scalable structured gaussian processes (kiss-gp). In *International Conference on Machine Learning*, pages 1775–1784, 2015. 3.1

[35] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. Deep kernel learning. In *Artificial Intelligence and Statistics*, pages 370–378, 2016. 3.1