INDIVIDUAL PROJECT

DEPARTMENT OF COMPUTING

IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

Deep learning for rain now-casting

Author:
Szymon Zmyslony

Supervisor:
Julie McCann

June 15, 2019

Submitted in partial fulfillment of the requirements for the MEng Computing of Imperial College London

Abstract

This paper develops an end-to-end rain now-casting model leveraging on a network of 51 rain sensors in Singapore. We first provide a statistical analysis of temporal and spatial distribution of rainfall events on the island. That investigation later guides our research into LSTM based encoder-decoder models for short-term precipitation forecasting. We first review current state-of-the-art approaches based on 2-D radar maps of clouds and then apply those findings to the specific domain of weather stations. We propose and evaluate four different neural network architectures, of which the two most important leverage graph convolutions to extract spatial relations between sensors. We then offer a comparative comparison of our approach with the radar map based one and conclude that one-dimensional rainfall data coming from rain sensors are lacking information for our models and cannot be considered useful in the real life. However, we also point out that obtaining datasets with more environmental features (such as solar radiation, temperature) will likely lead to better performance of graph-based memory models and suggest future research directions in that area.

Acknowledgments

I would like to thank my supervisor Professor Julie A. McCann for her unfailing support over the course of this project as well as Dr. Cong Zhao for his guidance and many insightful questions. My discussions with both have been instrumental to developing new ideas as well as evaluating this project. Thank you.

Contents

Chapter 1

Introduction

Water's growing significance for the world was recognised by the United Nations in 2010 when the organization's General Assemble declared ”Human Right to Water and Sanitation” to be one of human rights [1]. UN Secretary Ban Ki-Moon stated that safe drinking water and adequate sanitation are crucial for poverty reduction, crucial for sustainable development, and crucial for achieving any and every one of the Millennium Development Goals [2]. What is more, water is also important for other humanitarian reasons such as sustainable precision farming [3], flood warning systems [4], or sewage management. As fresh water reservoirs are not always readily available across the world and the existing ones become increasingly more polluted, rainfall prediction has become a crucial way of alleviating global warming effects and providing drinking water for the people.
In this project, we are interested in predicting future precipitation amount over the area of Singapore based on measurements of a network of environmental sensors (capturing rainfall amount, temperature, wind-speed etc.) distributed quite evenly across the country - but not regular-grid fashion (see 3.1). Our models forecasts could be applied at some test real estates over the island which would be further used as variable in another algorithm for managing drainage and run-off in the estate. If accurate, such model will enhance utilisation of recycled rainfall water likely leading to reduced costs as well as higher sustainability of estates. This is a challenging problem because rain in Singapore is a highly variable phenomena with a short (less than 1hour) average duration of a rainfall event.
Predicting rainfall intensity at a high spatial resolution over a short period of time (2-6 hours) is called precipitation now-casting and modelling perspective it can be regarded as spatio-temporal sequence generation. We adopt the same formulae as [5] and define sequence forecasting problem as predicting the most likely length-K sequence in the future given the previous J observations which include the current one:

$$\tilde{\mathcal{X}}_{t+1}, \ldots, \tilde{\mathcal{X}}_{t+K} = \arg\max_{\mathcal{X}_{t+1},\ldots,\mathcal{X}_{t+K}} p(\mathcal{X}_{t+1}, \ldots, \mathcal{X}_{t+K} \mid \hat{\mathcal{X}}_{t-J+1}, \hat{\mathcal{X}}_{t-J+2}, \ldots, \hat{\mathcal{X}}_{t}) \tag{1.1}$$

In this project, the $\hat{\mathcal{X}}$ observation at every timestamp J will consists of n i-dimensional vectors where n is the of sensors of the network and i is the number of variables they measure and predicted sequence will consist of measurements $\tilde{\mathcal{X}}$ of rainfall only for

every timestamp K.
For simplification, we assume that all sensors collect the same type of data.
Current state-of-art methods for precipitation nowcasting involve spatio-temporal series analysis of 2-D radar maps of the clouds [5]. It is achieved with the use of a specialised version of recurrent neural networks (RNN) called long-short term memory networks (LSTM) for time-series generation combined with convolutions layers responsible for extracting spatial features. Output of such models is a sequence of 2-D radar maps that are later fed into a physical equation mapping pixel intensities into actual rain intensities.
Here, we take an alternative approach and build an end-to-end now-casting model from weather station data directly to rainfall intensity. **This is a novel approach worth exploring as recent popularisation of remote sensing techniques like weather stations provides us with ample data to experiment with and does not require expensive cloud radars in contrast to previous studies. What is more, we believe that research on spatio-temporal forecasting of non-grid data could be applied in other domains.** For this reason, we decided to follow existing state-of-the-art techniques for predicting rainfall based on 2-D radar maps and apply the same principles to data from a network of distributed sensors. This project makes following contributions in that area:

- We provide a statistical analysis of temporal (in 3.2.3) and spatial (in 3.2.2) distribution of rainfall across Singapore and describe how what implications it has for developing a now-casting system

- In section 2.2.2 we theoretically explain empirical success of rainfall prediction models based on 2-D radar maps convolution (ConvLSTM - [5]) and propose a graph-convolution memory layer (6.5). We then apply it to the problem of now-casting from sensors network data

- We also propose three other memory models and evaluate its performance based on standard forecasting metrics. The best performing model is a Graph-Concat network in section 6.4 and it achieves MSE value of 0.66

Chapter 2

Background

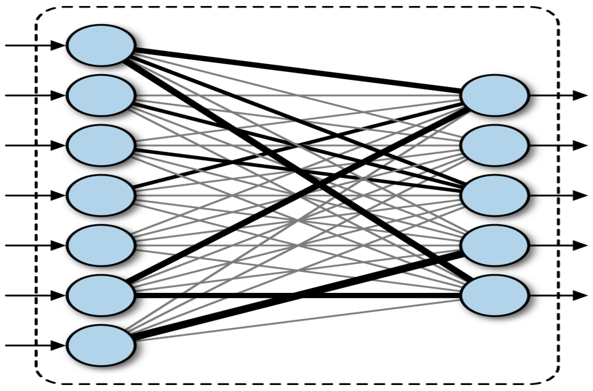
In this section, we introduce necessary background knowledge for understanding the project. Firstly, we discuss concept of a fully connected feed-forward neural network: what is the basic concept of a layer, present detailed view of a single neuron as well quickly recap simple activation functions. Secondly, we briefly introduce convolution and pooling operations on a regular grid domain as it will ease the understanding of existing state of the art techniques which later motivate the graph research. Further on, we discuss convolution properties from a geometric perspective and point out how they help to make convolution neural networks very effective in capturing spatial relations. Finally, we look explore the concept of a recurrent neural network (RNN) and long-short term memory network (LSTM). Those types of models will form the main prediction frameworks used in the project.

2.1 Artificial neural network

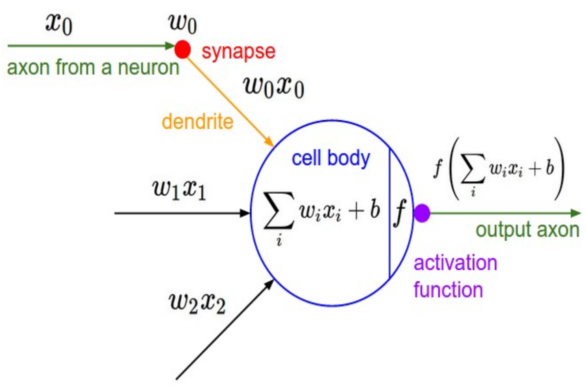
Feed forward artificial neural network is a fundamental building block for many deep learning models. It aims to learn the mapping $\boldsymbol{Y} = f(\boldsymbol{X}, \boldsymbol{\theta})$ given input-output pairs $\boldsymbol{X}$, $\boldsymbol{Y}$ by fine-tuning $\boldsymbol{\theta}$ so it results in the best approximation of the original function f. Neural network consists of number of consecutively applied functions called layers. We usually have at least one input and output layer and some hidden layers. An example of simple two layer neural network is shown in figure 2.1(a). Now, left layer represents an input layer for $\boldsymbol{X}$, in this case $\boldsymbol{X} \in \mathbb{R}^7$ as we have seven input nodes (neurons). For each neuron i in the left layer and each neuron j in the right layer, there is a corresponding weighed connection $w_{i,j}$ - there is different affine transformation $layer_{\boldsymbol{W},\boldsymbol{b}}$ for each layer - so that

$$y_j = g(\sum_i (w_{i,j} x_i + b_i)) \tag{2.1}$$

where b_i represents a bias and g is activation function introducing non-linearity in the neural network. This is show in more detail in Figure 2.1(b)

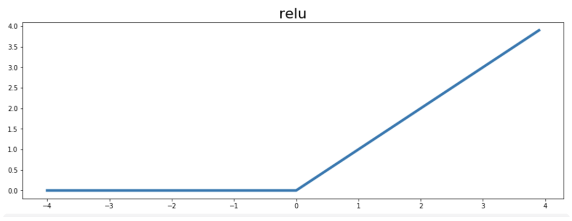


(a) fully-connencted network

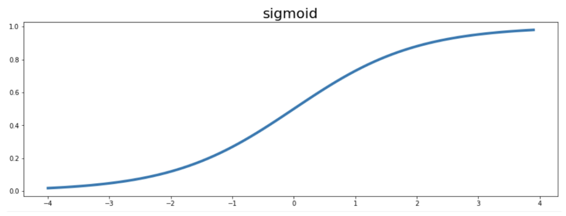


(b) Neuron model

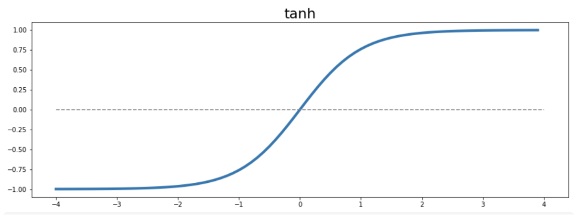
Figure 2.1: Neural networks basics



(a) relu



(b) sigmoid



(c) TanH

Figure 2.2: Activation functions

Popular activation functions are - they will all be used in the project:

- The rectified linear activation function $g(z) = \max(0, z)$
- Logistic sigmoid function: $g(z) = \frac{1}{1+e^{-z}}$
- Tanh function: $g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

Neural network is therefore an ordered collection of layers (consisting of neurons), weights and biases for each of the connection, and activation function applied after calculating weighed sum. It is important to note that in feed-forward network, there are no feedback connections - meaning output of the network is not used as its input again which will be the case in recurrent neural network. Before further discussion of more advanced neural networks, we will explore how a neural network can learn its parameters $\boldsymbol{\theta} = \boldsymbol{W}, \boldsymbol{b}$.

2.1.1 Neural network learning

After calculating output layer in 2.1(a) with values $\boldsymbol{Y}^{network} = \begin{bmatrix} y_1 \\ \vdots \\ y_5 \end{bmatrix}$, we conclude forward propagation of the network. We use output values to calculate final loss value L, using mean squared error function for $n = 5$:

$$L = \frac{1}{n} \sum_{i}^{n} (\boldsymbol{Y}_i^{data} - \boldsymbol{Y}_i^{network})^2 \tag{2.2}$$

The back-propagation algorithm [6] allows the information from the loss function to then flow backwards through the network and compute gradients which allows learning of the network's parameters $\boldsymbol{\theta} = \boldsymbol{W}, \boldsymbol{b}$ with another algorithm, for example stochastic gradient descent [6]. Neural networks usually operate on batched data - we do not complete froward pass with only an instance data, rather we do it for multiple examples and then average back propagated gradients. There are more parametes in the learning algorithm that can be fine-tuned depending on the nature of the task - we discuss this in more detail in 5.4. After finishing learning (there are many methods to evaluate when to stop), we are able to predict output on unseen data $\boldsymbol{X}^{unseen}$ by performing single forward pass:

$$Y^{predicted} = net_{\boldsymbol{\theta}}(\boldsymbol{X}^{unseen}) \tag{2.3}$$

2.1.2 Over-fitting

As neural networks can fit approximate any given function arbitrary well, we usually partition our data into train and test sets and train the model on the first set of data and evaluate its performance on the latter one. It is often the case that training results do not generalize well for unseen data. Over-fitting happens when neural

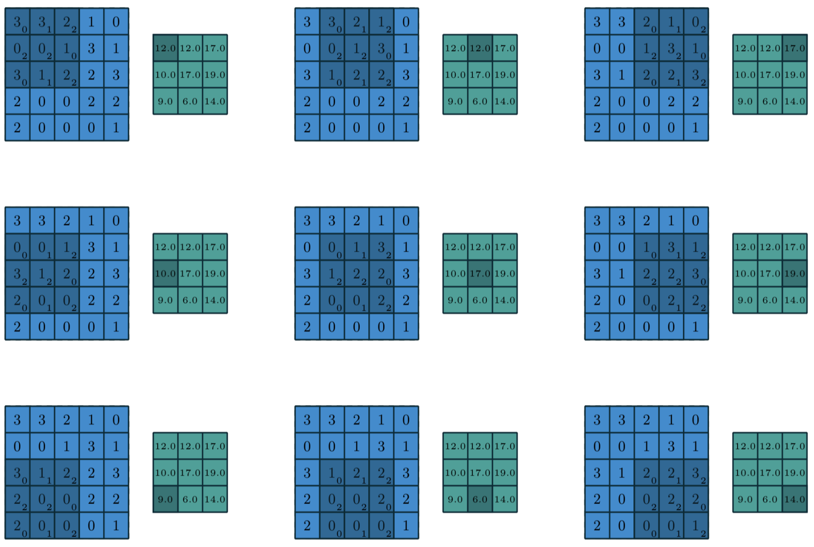


Figure 2.3: Convolution operation on 2-D image [7]

network works well on training data but achieve poor results for test data. It can be overcome with obtaining a better-bigger data-set or some means of regularization such as penalizing big weight values or dropping random neurons when learning. This is also discussed in more detail in 5.4.

2.2 Convolution neural network

2.2.1 Intuitive understanding

Convolution

Convolution operation can be understood as a very efficient way of extracting features from 2-D maps by calculating weighed sum of pixels with the use kernel matrix. It leverages topological ordering of pixels, unlike affine transformations in feed-forward networks, **reuses same kernel for all image segments** - meaning it is using the same patch function across the whole domain, and is sparse as only some input pixels contribute to given output pixel. Figure 2.3 shows application of kernel

$$K = \begin{bmatrix} 0 & 1 & 2 \\ 2 & 2 & 0 \\ 0 & 1 & 2 \end{bmatrix}$$

to the underlying input feature map.
We can see that output of the convolution operation (output feature map) is smaller in size than input (input feature map) as the kernel matrix (patch operator) can not average pixels that are outside of the border. Should the same size of output be needed, padding on borders could be applied and in that case output feature

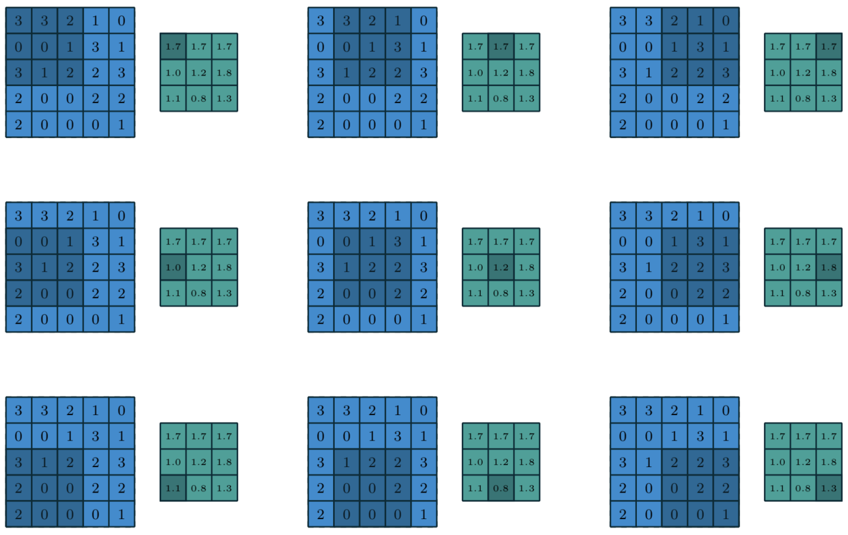


Figure 2.4: 5x5 image reduced to 3x3 by averaging neighbouring pixels [7]

dimension can be calculated as: $w = W - W_k + 2P + 1$ where w is the output length, W is input feature map length, W_k is the kernel length and P is length of padding. Kernels work across the depth dimension, meaning that single kernel is used at each channel, see 2.5 where input feature map has 3 channels - red, green, and blue, and number of kernels corresponds to number of output feature maps which then later can combined into final output and back-propagation algorithm can be used for learning kernel values.

Pooling

Pooling operation is another important component of convolution neural networks. It can be thought of as a down-sampling of a feature map in order to reduce its size (thus calculation complexity) as well as provide some degree of equivariance for local transformation. In essence, pooling is like applying another patch operator on a feature map that summarizing local characterise of the input. Figure 2.4 shows how an averaging kernel filter produces output feature map, other patch operators (sum of pixel values, max of pixels values, learnable kernel) are possible and they largely depend on the task at hand.

2.2.2 Why does convolution really work?

Having intuitively understood how two main ingredients of convolution neural networks work, we now describe geometric properties of regular grids (such as squares) that make pooling and convolution so effective in extracting spatial information. This section is largely based on [9].
We consider squared integrable function $f \in [0,1]^2$ and we analyze an unknown function $y : f \rightarrow \mathcal{Y}$ where $\mathcal{Y}$ has the dimensionality of the target space - in this

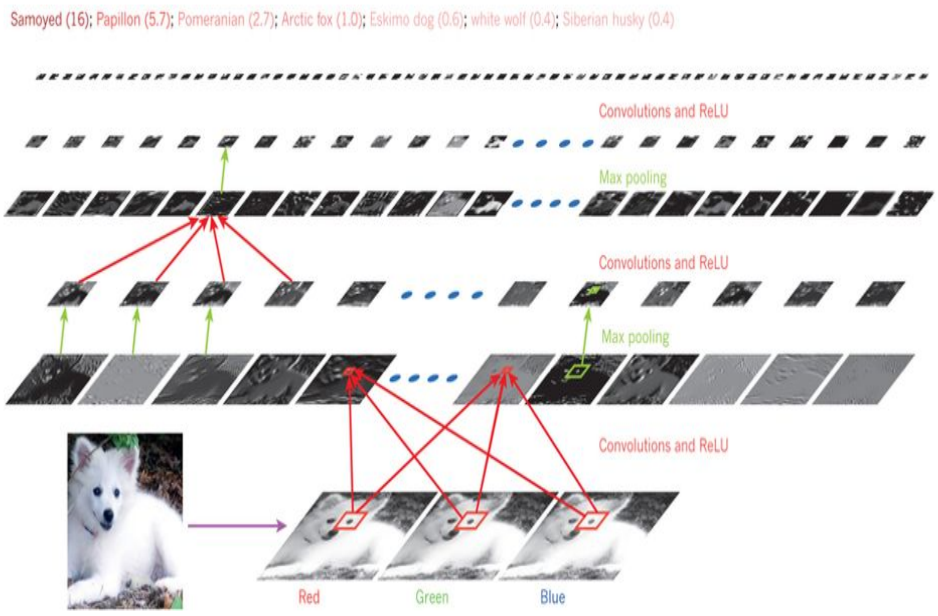

Samoyed (16); Papillon (5.7); Pomeranian (2.7); Arctic fox (1.0); Eskimo dog (0.6); white wolf (0.4); Siberian husky (0.4)
Convolutions and ReLU
Max pooling
Convolutions and ReLU
Max pooling
Convolutions and ReLU
Red
Green
Blue

Figure 2.5: LeCun's convolutional neural net [8]

case, it will be further parameterized by timestamp k and correspond to model's predictions. In most of image analysis tasks (such as [5] and [10]), we can put some reasonable priors on the function y

Stationarity

Firstly, we assume that function y is equivariant to translations which can be defined as an operator τ working on function square integrable f for $x, v \in [0,1]^2$:

$$\tau_v f(x) = f(x - v)$$

Now for the equivariance property to hold, we must have

$$y(\tau_v f(x)) = \tau_v(y(f(x))) \tag{2.4}$$

Above is well defined given that output space is such that translation is possible - for example output is a picture.

Deformations

We also assume that the function y is equivariant with respect to local deformations $mathcalL$ defined as:

$$\mathcal{L}_\tau f(x) = f(x - \tau(x))$$

where τ is an operator from $[0,1]^2$ to $[0,1]^2$. We again assume equivariance such that:

$$|y(\mathcal{L}_\tau f) - \mathcal{L}_\tau y(f)| \approx ||\delta\tau|| \tag{2.5}$$

where $||\delta\tau||$ is the smoothness of τ meaning that our function y does not change much when input image is slightly deformed.

Convolution neural networks (CNN)

Now, assuming that 2.4 and 2.5 holds true, we define 2-d convolutional neural network to consist of multiple layers [11]: $\mathbf{g} = C_\tau(\mathbf{f})$, acting on a p-dimensional input $\mathbf{f}(x) = (f_1(x), \dots, f_p(x))$ by applying a bank of filters $\tau = (\tau_{l,l'})$, $l = 1, \dots, q, l' = 1, \dots, p$ and point-wise non-linearity ξ functions such as in 2.2.

$$g_l(x) = \xi\left(\sum_{l'=1}^{p}(f_{l'} \star \tau_{l,l'})(x)\right) \tag{2.6}$$

where convolution $(f \star \tau)(x)$ is defined as

$$(f \star \tau)(x) = \int_\Omega f(x - x')\tau(x')\delta x' \tag{2.7}$$

Such a layer produces a q-dimensional output $\mathbf{g}(x) = (g_1(x), \dots, g_q(x))$ which is referred to as feature maps. In 2.2.1 we pointed out learnable kernel parameters are shared across width and height of the image. This corresponds to learning observed function y defined in 2.2.2, which we assumed is translation equivariant. From 2.5, we know that convulutions can be composed so that important spatial information is captured at different scales. Thanks to that, we have a constant number of parameters per layer (kernel size) independent of the input size (unlike as in fully connected nets) and hierarchical properties of convolutions allow to achieve sub-linear total learning complexity. Therefore, we conclude that placing geometric priors on imagines in $[0, 1]^2$ makes patch convolution operators underpins their experimental success. Figure 2.5 demonstrates whole working CNN in practice. Notion of stability to local translations and applying same patch operator across the spatial domain will prove critical to research in 6.

2.3 Memory networks

Deep neural networks with convolution layers were discovered for processing grid structured data, but they are not suited (at least until recently) for dealing witih temporal sequences. Recurrent neural networks (RNN), in contrast, were specially designed for variable-length sequence processing. In this section, we give a short introduction of the general class of memory networks (RNN) and then expand it to long-short term memory networks (LSTM) which are better in handling long-term dependencies encountered in nowcasting problem.

2.3.1 RNN

Let's take a simplest temporal sequence analysis problem. We would like to extract information of when it is gonna rain from two sentences: ”It will be raining today” and ”Today, it is going to rain”. Should feed-forward neural network was to be trained to solve this problem, separate parameters for each input feature (for example, word embedding) in each place in the sentence of fixed length. That has proven

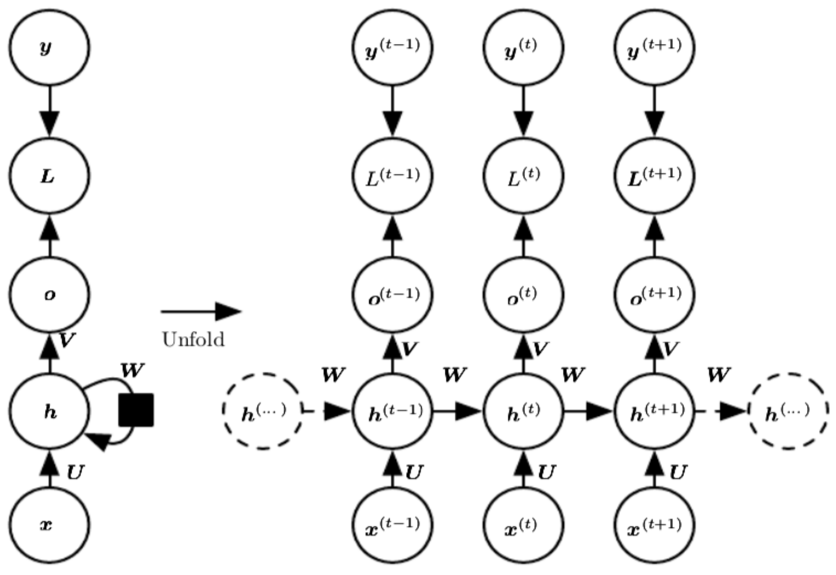


Figure 2.6: Unfolding of recurrent neural network [6]

to result in bad results and training efficiency. Recurrent neural networks use parameter sharing across parts of the sequence (in our case, timestamps) and enable future predictions based on varied-length input. This makes them particularly suitable for now-casting problem being considered. Such a formulation of the recurrent network can be understood as a folded feed-forward network where $\boldsymbol{s}^t$ is state of the system at time t. Evolution dynamics of that can be recurrently defined by [6]:

$$\boldsymbol{s}^t = f(\boldsymbol{s}^{t-1}, \boldsymbol{\theta}) \tag{2.8}$$

which can be later unfolded. For $t = 3$, we have:

$$\boldsymbol{s}^3 = f(\boldsymbol{s}^2, \boldsymbol{\theta}) = \ = f(f(\boldsymbol{s}^1, \boldsymbol{\theta}), \boldsymbol{\theta})$$

Recurrent equation 2.8 is often used to calculate hidden state h^t for each timestamp t in the network:

$$\boldsymbol{h}^t = f(\boldsymbol{h^{t-1}}, \boldsymbol{x}^t, \boldsymbol{\theta}) \tag{2.9}$$

In this setting, $\boldsymbol{h}^t$ represent a lossy summary of history and is used to make prediction on the next output $\boldsymbol{o}^t$. Unfolding of recurrent neural network and calculating the loss function on the next output $\boldsymbol{o}$ is shown in figure 2.6. It uses recurrent connection between hidden state h^{t-1} and h^t to provide the model with a prediction context. RNN is a significant improvement on CNN and regular feed-forward network in case of sequential problems, but the concept still suffers from problem of long-term dependencies. Weights given to distant past are exponentially smaller than parameters for short-term past and as such they make network learning very difficult. This has been explored in [6] to a great detail. We now omit learning methods for recurrent networks because they are not crucial for understanding this project and are discussed extensively in other sources. We also omit internal mathematics of recurrent layers and move on to describe structure of more advanced LSTM unit which is able to deal with long-term dependencies in an efficient way.

2.3.2 LSTM unit cell

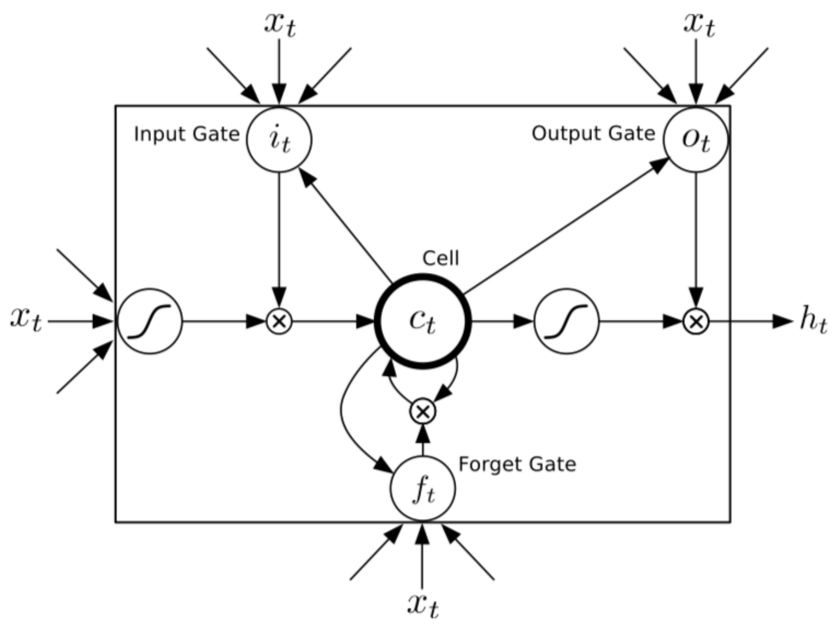


Figure 2.7: Standard LSTM unit - subscript notation (c_t instead of c^t) [12]

Long short term memory units form a crucial way of networks dealing with sequential future prediction. We adopt a formulation from [12] of the LSTM unit cell (figure 2.7) where it has cell state c_t at time t. Memory access is controlled by input i_t, output o_t, and forget f_t gates which all use $sigm$ as activation function. All of the 3 gates and input (left of the 2.7 receive, at time t, input $\boldsymbol{x}^t$ and previous states summary $\boldsymbol{h}^{t-1}$ and compute final output $\boldsymbol{h}^t$ by following equations in 2.10. $\circ$ is Hadamard product.

$$\begin{aligned}
i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci} \circ c_{t-1} + b_i) \\
f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf} \circ c_{t-1} + b_f) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \\
o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co} \circ c_t + b_o) \\
h_t &= o_t \circ \tanh(c_t)
\end{aligned} \tag{2.10}$$

Each weight matrix has indices explaining which part of the system it corresponds to, for example: W_{hi} is the hidden-input gate matrix, W_{xo} is the input-output gate matrix etc. Above cell idea became a dominant design for LSTM allowing internal cell state of the unit to trap the previous information making learning long-term features easier. LSTM architectures are considered state-of-the-art approach for time-series analysis and yield superior results when applied to nowcasting problem as their are able to capture long-term dependenciess in dynamical weather system.

2.3.3 Encoder-decoder and future predictions

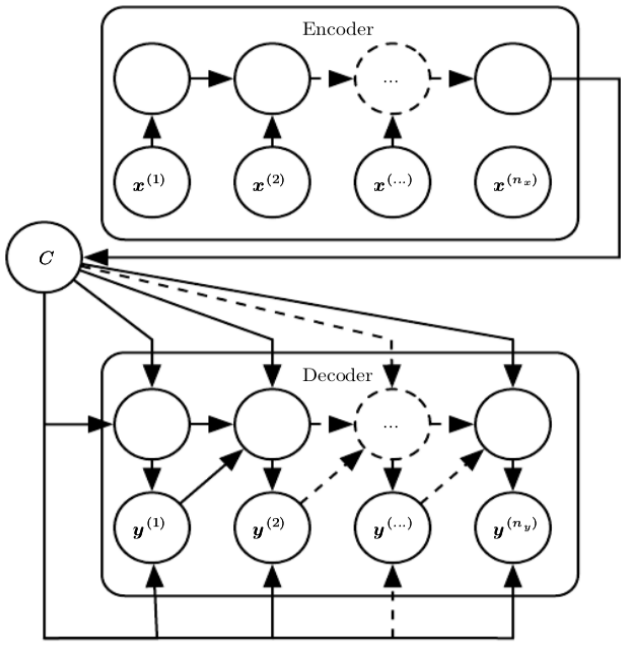


Figure 2.8: Encode-decode structure of LSTM system [6]

We now move onto describing encoder-decoder structure made of multiple LSTMs that allows mapping input sequence of length n to output forecast of length m. The goal is to predict distribution $P(y_1, y_2, \cdots, y_n | x_1, x_2, \cdots, x_m)$. This is reformulation of 1.1 where y corresponds to future predictions.
In the simplest formulation, we have one LSTM layer that acts as encoder and one LSTM layer in the decoder - see figure 2.8 (this can be extended to multiple stacked layers) - see 5. We initially fed input sequence of x into the encoder network and obtain a fixed length context summary $\boldsymbol{C}$. This is a concatenation of the last hidden state h_t and the last cell state c_t. We then use that vector as initial hidden state for the decoder, which sequentially produces future predictions and uses them as an input for following calculations.
There are many factors determining how well encoder-decoder model performs. One of the most important is the dimensionality of hidden states (h, c). If they are much larger than dimensionality of input x, we can expect that over-fitting will happen as model will have enough parameters to mirror training-data so well that it will probably not generalize for unseen sequences. In 2.8, we see that actual outputs of the network y_t are re-fed into decoder to generate following predictions. It has been discovered that during training time, this can be improved by using actual values (from target data) instead. So rather than inputting y_t and predicting y_{t+1} we would input y_t^{data} and produce y_{t+1}. This is called teacher forcing and has proved to accelerate training of the network. It is worth to point out that teacher forcing can be used only during training time as at evaluation time we do not now real targets. More architectural choices in the encoder-decoder structures are discussed in 5.

Chapter 3

Data domain and its challenges

Data being modeled in this project comes from environmental sensors spread across Singapore. There are 55 sensors measuring rain intensities at 5 min intervals and 16 of those are co-located with more advanced devices that measure wind speed, wind direction, relative humidity, and dry bulb temperature. Thee data collected comes from a 1 and half year long period starting in December 2016. In this section, we emphasise important characteristics of that data that will later motivate design of machine learning algorithms.

3.1 Sensors and available data

All environmental sensors are shown in the figure 3.1. Most of them are located in the centre of the island what will result in predictions for that region being more accurate than for outer parts of the country. Red dots represent sensors measuring all 5 environmental factors while yellow dots correspond to rainfall only devices. Singapore position in the map is between 1.15° and 1.47° north in vertical direction and ranges from 103.6° to 104.1° east in horizontal direction. Geo-coordinates of sensors locations must be projected from 3-D earth surface onto 2-D flat surface of a computer screen. This is important as it allows us to calculate accurate distances (in horizontal and vertical directions) between points on the map. Map projection is very common operation when dealing with geographical and many geo-spatial libraries in Python can handle when provided with a coordinate reference system describing describing how x, y points relate to real-world positions. Here we will use 24500 projection from a spatial reference library [13] that projects from bounds of

$$103.6200, 1.1200, 104.1600, 1.4600$$

to

$$4260.3246, 11468.9542, 64361.4025, 49065.6777$$

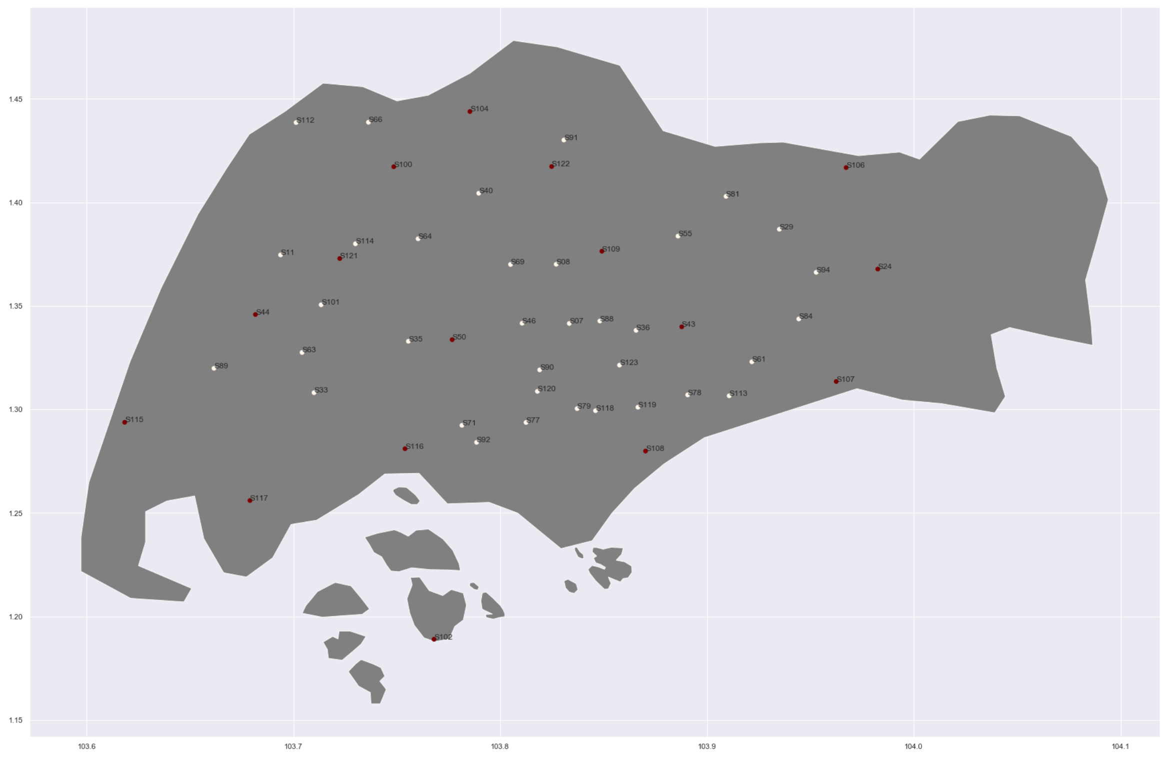


Figure 3.1: Geographical distribution of environmental sensors across Singapore. [not projected view, axis geo-coordinates of points]

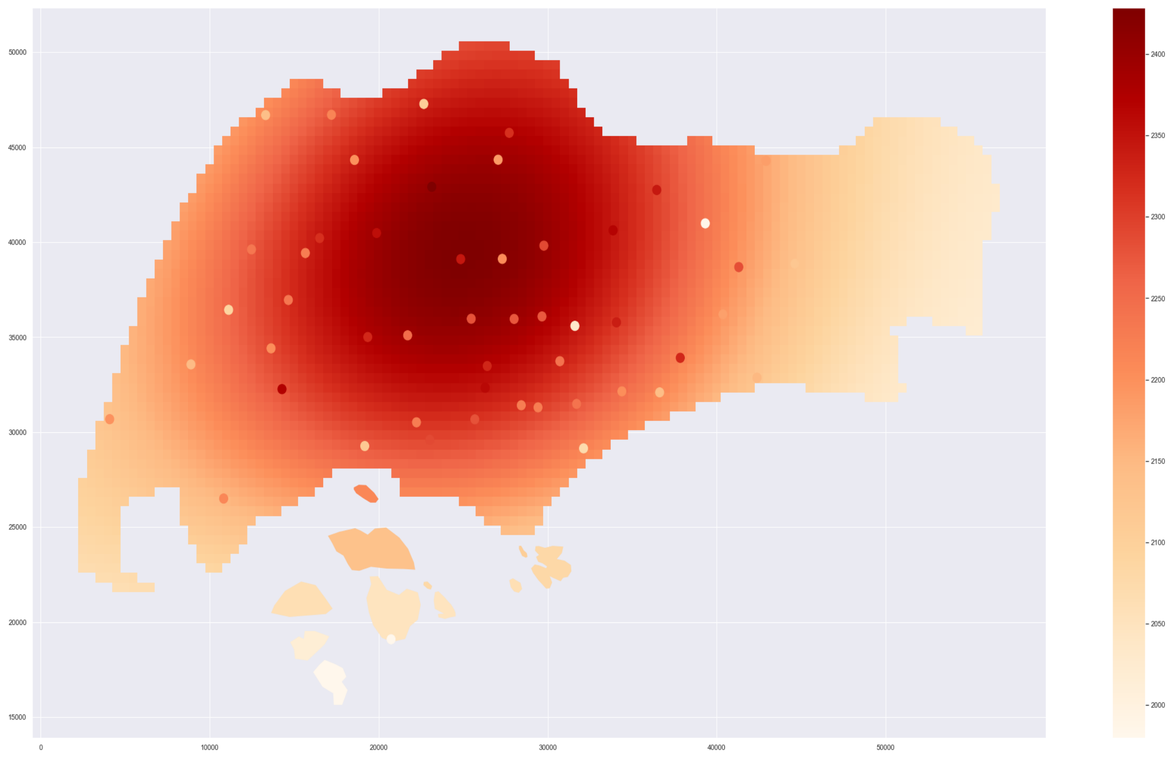


Figure 3.2: Interpolated sums of rainfall during one year period projected onto flat Cartesian coordinates

Figure 3.2 shows how sum of yearly rainfall varies across Singapore with red-white color map representing higher and lower intensities of rain. That figure also demonstrates how x and y values of geo-points changed after the projection. Gradient color on the map is an interpolated value based on method of ordinary Krigging [14] which leverages spatial correlation between data points to produce best interpolated estimating of points between sensors (points further away from ground truth sensors have less certainty. It can be seen that yearly sum varies by around 20% despite the island being only around 50km long in east-west direction and 20km long in the north-south ax. This might suggests that rainfall events is very localised and there might be low correlation of precipitation values between different sensors. That conclusion is important to this project as methods considered[5, 6] assume some degree of correlation between different data stations. We explore spatial correspondence in a greater detail in 3.2.2.

3.1.1 Missing data

Both datasets provide values for each environmental variable every 5 minutes. Timestamps' values range from mid December of 2016 to February of 2018. There is a lot missing data resulting from two main reasons: repeated bad gate-away calls to the data server access points or sensors malfunctioning at some points of time. As identified in 3.2.3, rain intensity varies greatly on diurnal and annual basis implying that the predictive model must take into the account time of the year and of the day. For that reason, wee decided to find sensors for which there is more than 10% of days missing from any given year and exclude those devices from our data-set. In rain-only data-set, there are 5 sensors with records from only 100 days in 2018 and in environmental data, we have 4 such sensors. Furthermore, models explored in this project also require that for a given time-stamp, there is available data from all sensors. Those two main limitations considerably decrease the size of data-sets for which more detailed description is given in 7

3.2 Local characteristics

The whole field of nowcasting assumes that there is a short-term dependence between forecasted values and history of measurements and that events are correlated in spatial domain for a limited radius distance. For that reason we provide, short statistical analysis of both spatial and temporal correlations between weather in Singapore. This analysis is limited to rainfall only dataset but later extended to correlation between different variables at a single sensor station for different time-lags.

3.2.1 Pearson Correlation

In following section we adopt Pearson correlation coefficient as means of evaluating thee strength and direction of linear relationships between pairs of variables [15].

That coefficient can be computed as:

$$r_{i,j} = \frac{\mathrm{Cov}(i,j)}{\sqrt{\mathrm{Var}(i)}\sqrt{\mathrm{Var}(j)}} \tag{3.1}$$

where $\mathrm{Cov}(i,j)$ is the sample co-variance of i and j; $\mathrm{Var}(i)$ is the sample variance of i; and $\mathrm{Var}(j)$ is the sample variance of y. Value of 1 means perfectly positive linear relationship, -1 is perfectly negative while 0 means no correlation at all.

3.2.2 Spatial distribution

With an eye to use past sensors readings from all geo-locations to predict future rainfall events in all sensor locations, we are investigating how correlated the lagged values are based on their horizontal and vertical distance. We adopt similar approach to [16] in which we compute pairwise correlation coefficient between stations on different time lags. In such setting $\mathrm{Cov}(i,j)$ for timelag δt is defined as:

$$\mathrm{Cov}(i,j) = \frac{1}{n-1}\Sigma_{t=1}^{n}(x_i(t) - \bar{x_i})(x_j(t+\delta t) - \bar{x_j}) \tag{3.2}$$

where n is number of commoon timesteps btween statiions i and j. We calculate that for all rainfall sensors which results 51x50 coefficient for every time-lag (1-4) investigated. In order to see correspondences between stations, we calculate horizontal and vertical distances between each station (limiting ourselves to 15km pairwise distance) and plot average values of correlation coefficient between different timestamps. One time-lag considered here is 12 minutes as this is how we re-sampled data to achieve better performance in 5 and 6. Figure 3.3 and 3.4 show that spatial extent of rainfall events is quite limited as there the correlation coefficient is quite small for most of distances and decreases even further with greater time-lag. What is more, there is greater correlation in horizontal direction than the vertical one and it can be explained by Singapore specific west-east winds in [16]. [16] also points out that both spatial and temporal correlation varies from season to season but we do not explore that variability any further because we are aiming to build a general end to end model that is not tailored for any specific time period. We now move onto investigating temporal variability of rainfall events in diurnal and yearly perspective.

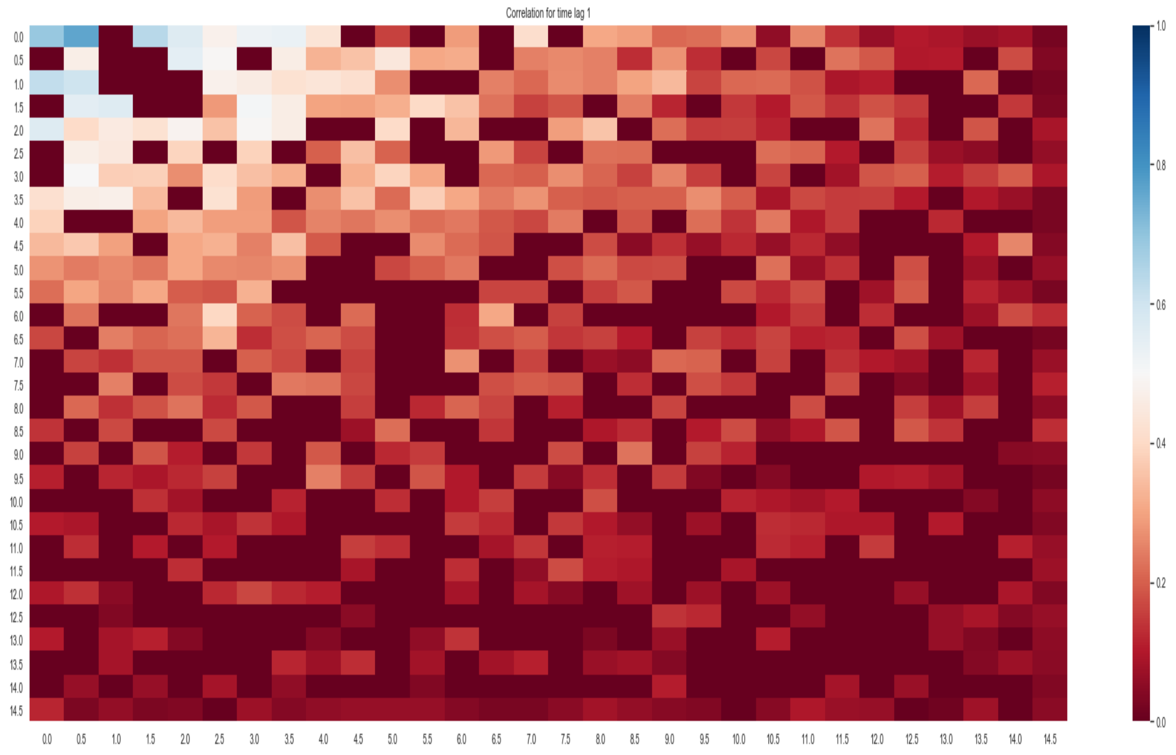


Figure 3.3: Spatial correlation of measurements

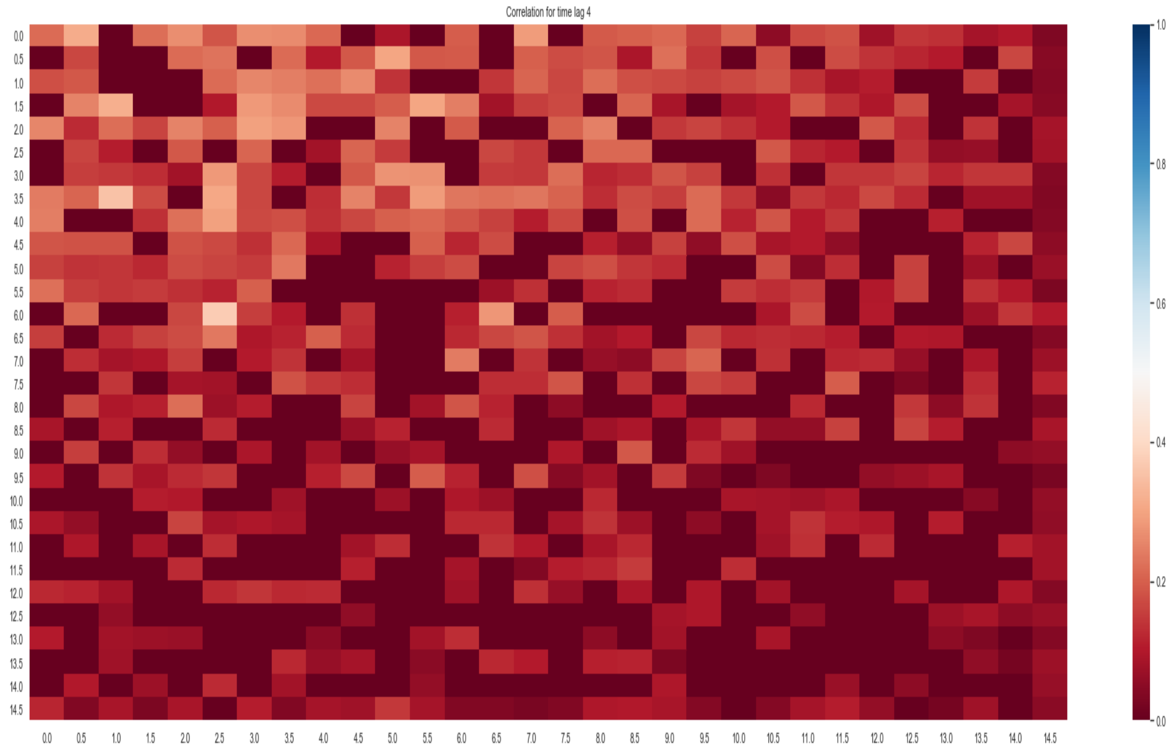


Figure 3.4: Spatial correlation of measurements lagged by one timestamp

3.2.3 Temporal distribution

Upon discovering that spatial extent of rainy events is very small, we now aim to explore how rainfall changes with time. [16] proved that Singaporean rain usually lasts less than an hour which will already pose quite a significant problem for machine learning models as they will have to be able to predict different outcomes (no-rain or rain events) from the history of rains (for example no rain in the past hour). This will not be possible without taking leveraging spatial relations between sensors, but also taken into account inter-seasonal variability of rain intensities. For that reason we plot rainfall intensity as well as its frequency depending on the time of the day and year. Figures 3.5 and 3.6 show that distribution for one randomly sampled sensor from the centre of the island with time of the year plotted on horizontal axis and hour of the day on vertical.

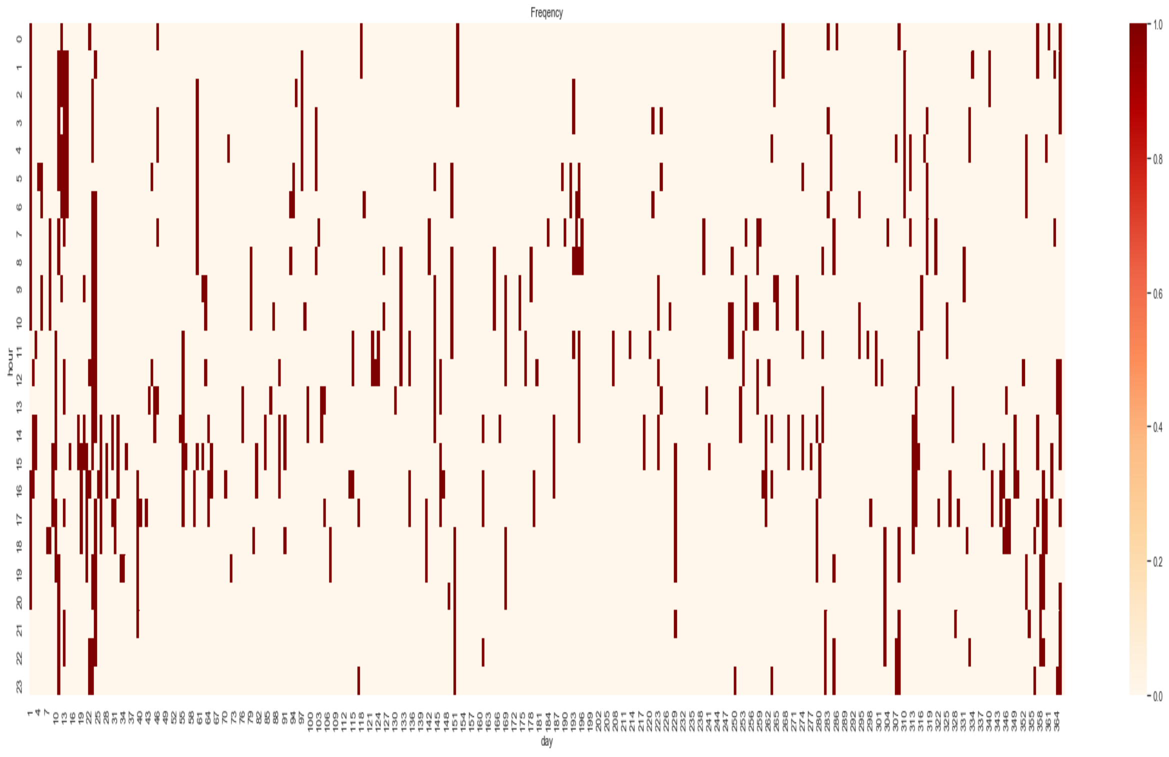


Figure 3.5

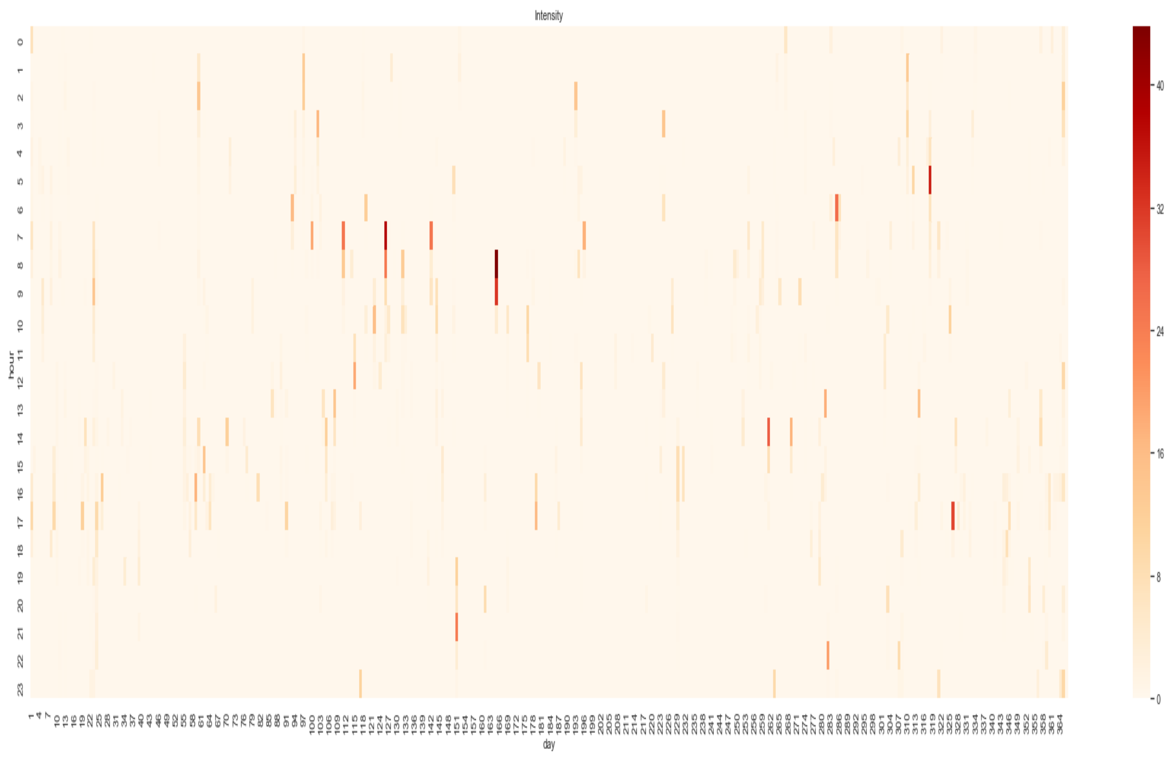


Figure 3.6

It is apparent from those figures that rainfall in Singapore a is highly variable phenomena and its behaviour changes depending on both hour of the day and time of the year. An excellent discussion (with analysis done on a larger dataset) of that is provided in [16], but for the case of machine learning models, it is clear that both information have to be somehow incorporated into other model flow. This can be done by simply adding that temporal information to other input variables, coming up with a new normalisation technique that is dependent on the season and date-time. We further analyse the consequences of that high variability in later sections.

3.2.4 Within sensor correlation

In this final data exploration section, we calculate correlation between different lagged variables from environmental dataset within one station. This is very important as based on findings from 3.2.2 and 3.2.3 we suspect that highly variable nature of rainfall data and small values for spatial correlation will make rainfall forecasting very challenging when the input data is only one dimensional time-series of rainfall values. For that reason, we believe that environmental data with 4 additional variables can prove to result in better performance of the model. We plot Pearson's coefficient co-variance between different vartiables for 6 different time-lags and present the result in figure 3.7. These are example values plotted only for station no. 116 but they are representative of other sensors. From that, we can see that rainfall in lag 6 is positively correlated with all previous recorded rainfalls as well as air temperature. It also highly negatively correlated with the previous temperature. Wind direction and wind speed seems to have no measurable statistical effect on the the actual rainfall intensities (but they may affect how they change from one timestamp to the next one). That confirms our suspicion that inclusion of more environmental data will improve reliability of rainfall forecast.

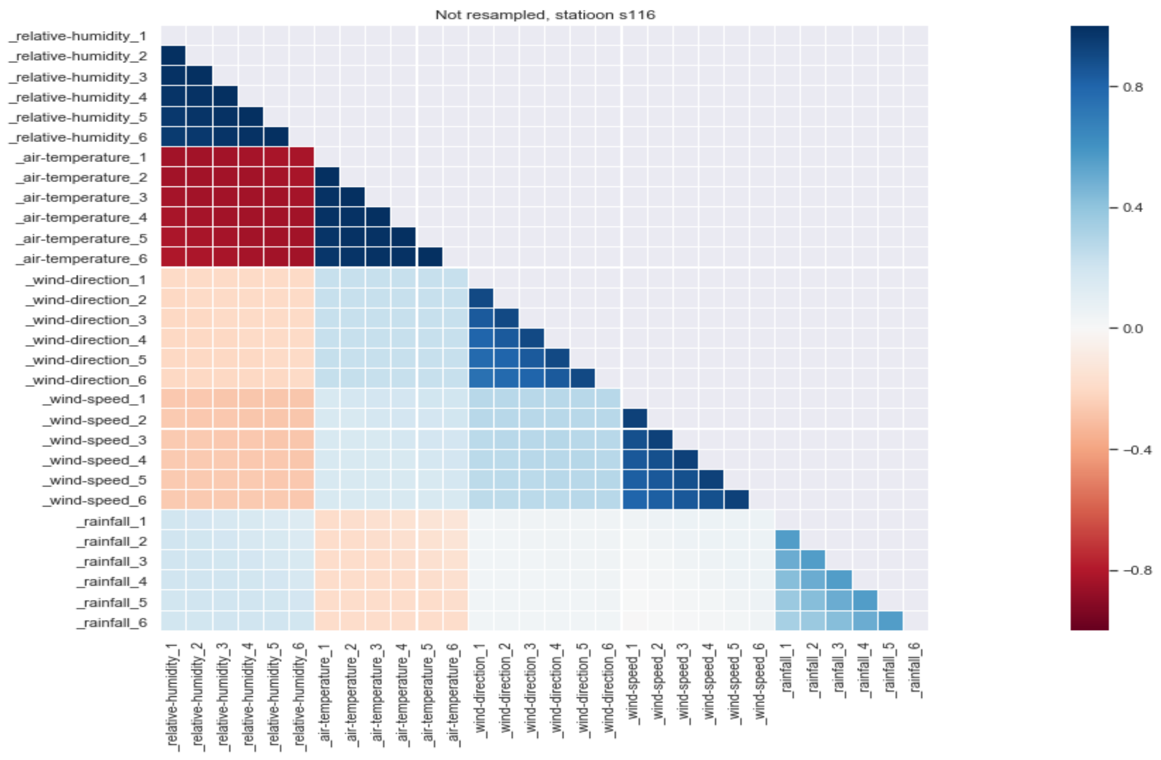


Figure 3.7: Correlation between different environmental factors

3.3 Summary

We showed how highly variable rainfall data is. It exhibits different properties depending on time of the year and day as well as that rain is usually short-lived event with small spatial extent. We then described major challenges the nature of the data will pose for machine learning models and pointed out that inclusion of more environmental data might alieavaate this problem.

Chapter 4

Evaluation plan

Main objective of the project is to create and end-to-end model being able to predict rainfall intensity over short-time period in a small area. Our approach is quite different from existing ones as we will operate on low dimension time-series signal coming from multiple sensors rather than a single 2-d map of clouds coming from a radar. From that reason, the first objective of the project is to evaluate how suitable memory network really are for this type of rainfall nowcasting, identify problem with training such models and point out advantages and disadvantages of the two approaches. We are also interested in satisfying Singaporean estates requirements described in 1. For that reason, we introduce a formal way of evaluating model's performance that combines both objectives.

4.1 Understanding performance of sensors network nowcasting

As defined in 1.1 the forecaster model's output will consist of a sequence of predictions for next n consecutive timestamps. In order to train the model, we will be minimizing squared sum of differences between predicted rainfall amount and actual reading from the sensor at corresponding timestamp. This can be expressed as:

$$L(\tilde{\mathcal{X}}_{t+1}, \ldots, \tilde{\mathcal{X}}_{t+K}) = \frac{1}{K}\Sigma_{k=0}^{K}(\mathcal{Y}_k - \tilde{\mathcal{X}}_k)^2 \tag{4.1}$$

where $\tilde{\mathcal{X}}_{t+K}$ are predicted values up to Kth timestamp and $\mathcal{Y}_{t+k}$ are target values. Such sum is then averaged over all samples in the batch. The best model will have the smallest value of MSE loss function. Unfortunately, average squared sum of errors is not a meaningful evaluation metric describing how well the model performs in different settings. In the case of Singapore that invested in big tanks able to store large amounts of water for local use, we know that main goal is to create a control trigger for drainage if predicted sum of rainfall over next four hours will exceeded maximum capacity of the tank. This is a reasonable evaluation goal because it redefines the sequence generation problem as a simple binary classification problem which is easy to measure and directly corresponds to the real objective of the system. However, evaluating model's performance at only one time horizon ($k = 4$ hours) and at one rainfall threshold τ (volume of the tank) is not sufficient. First of all, it does not allow to test tank's optimal utilisation which requires only draining the tank by however MUCH rainfall is predicted (plus overhead for uncertainty). This would give the maximum usage of tank resources and provide maximum rainwater for local usage. Secondly, the state-of-the-art nowcasting study [10] (uses 2-D radar maps rather than single sensor points) also adapts more general evaluation framework testing model's performance at different rainfall threshold levels. With an eye to evaluate difference between radar map and sensors network approach, we will indeed evaluate our nowcasting algorithm as a multiple binary classification problem at different prediction windows k and at multiple rainfall thresholds τ. We now move onto describing formal metrics used for describing effectiveness of the system

4.2 Quantifying success

4.2.1 Confusion matrix

We will consider thee event to take place if the actual rainfall amount exceeded given threshold at a given timestamp. So we may be interested in detecting heavy rainstorms - threshold for that will be 30mm/h of rain which translates to 6mm/12min (out timestamp length). The model will output a forecast which will be later classified as a hit - when it correctly predicted the event (in our example, timestamp output is greater than 6m/12min), incorrect forecast of occurrence, incorrect forecast of non-occurrence or, finally correct forecasts of non-occurrence. For multiple outputs, this information can be succinctly summarised in the contingency table which can be seen below:

$$\begin{pmatrix} \text{hits} & \text{false alarms} \\ \text{misses} & \text{correct negatives} \end{pmatrix} = \begin{array}{r c} & \begin{array}{cc} \text{Positive Event} & \text{Negative event} \end{array} \\ \begin{array}{r} \text{Positive forecast} \\ \text{Negative forecast} \end{array} & \begin{pmatrix} a & b \\ c & d \end{pmatrix} \end{array}$$

4.2.2 Metrics

Nowcasting is a still a fast developing emerging field with no established way of evaluating results, but we believe that in the case of that particular project, evaluation methodology adopted by [5] and [10] is a sensible choice to make. We will evaluate model's performance based on four different metrics: Probability of detection (POD), false alarm radio (FAR), critical success ratio (CSI), and Heidke skill score (Heidke). All of them can be computed directly from the confusion matrix and the formulas for them are given below:

$$\begin{aligned}
\text{POD} &= \frac{a}{a+c} \\
\text{FAR} &= \frac{b}{a+b} \\
\text{CSI} &= \frac{a}{a+b+c} \\
\text{Heidke} &= \frac{2*(a*d-c*b)}{(a+c)*(c+d)+(a+b)*(b+d)}
\end{aligned} \tag{4.2}$$

It is worth noting that in traditional machine learning perspective, POD = Recall and FAR = 1-Precision. We are interested maximizing all of the metrics but minimizing false alarm ratio. We will evaluate those metric at 3 different timestamps $k = 1, 5, 10$ and use [10] thresholds for binary classification: $\tau = 0.5, 2, 5, 10, 30$.

Chapter 5

Machine learning for Nowcasting

Having established what data is available and what evaluation methodology we will follow, we describe current state-of-the art model for nowcasting 2-D radar maps and point to its most important features that could be applied to our problem of nowcasting sensors network readings. We then implement those findings into two rather simple encoder-decoder memory models and try to improve on basic results by fine-tuning the model with the use Microsoft Neural Network Intelligence toolkit and applying simple regularisation techniques. Finally, we identify non-leveraging spatial information as model's major shortcoming what motives introducing geometric deep learning in the next section.

5.1 State of the art approach

5.1.1 ConvLSTM model

The existing state of the art for rainfall now-casting is largely based on work in [5], where convolution LSTM (ConvLstm) was proposed. It uses the principle as in 2.10 of input, forget, update, and output gates controlling cell state on the conveyor belt going through the whole cell. The update equations are introduced below:

$$\begin{aligned}
i_t &= \sigma(W_{xi} * \mathcal{X}_t + W_{hi} * \mathcal{H}_{t-1} + W_{ci} \circ \mathcal{C}_{t-1} + b_i) \\
f_t &= \sigma(W_{xf} * \mathcal{X}_t + W_{hf} * \mathcal{H}_{t-1} + W_{cf} \circ \mathcal{C}_{t-1} + b_f) \\
\mathcal{C}_t &= f_t \circ \mathcal{C}_{t-1} + i_t \circ \tanh(W_{xc} * \mathcal{X}_t + W_{hc} * \mathcal{H}_{t-1} + b_c) \\
o_t &= \sigma(W_{xo} * \mathcal{X}_t + W_{ho} * \mathcal{H}_{t-1} + W_{co} \circ \mathcal{C}_t + b_o) \\
\mathcal{H}_t &= o_t \circ \tanh(\mathcal{C}_t)
\end{aligned} \tag{5.1}$$

Hidden states in 5.1 are 3-D tensors with width, height and channels as dimensions. If we use n kernels W, then output of the layer will n dimensional features map of 2-D tensors. This is represented in 5.1. Convolutions connections between inputs and hidden state reduce number of parameters in comparison with a fully-connected LSTM and by leveraging property of translation in-variance achieve better results in spatial-temporal analysis. Now, [5] used multiple stacked ConvLSTM layers to create a decoder similar to the one in 2.8. This time, instead of feeding the fixed size

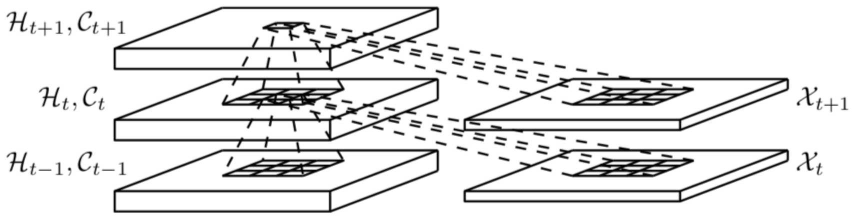

$\mathcal{H}_{t+1}, \mathcal{C}_{t+1}$
$\mathcal{H}_t, \mathcal{C}_t$
$\mathcal{H}_{t-1}, \mathcal{C}_{t-1}$
$\mathcal{X}_{t+1}$
$\mathcal{X}_t$

Figure 5.1: Convolutional connections inside of ConvLSTM [5]

context vector into each cell inside of unrolled decoder, we only use that vector $\boldsymbol{C}$ as initial hidden state for the first cell. If multiple stacked layers are used for encoder-decoder structure, we must make sure that there is the same number of memory layers in both sub-nets as well as use $\boldsymbol{C}_i$ from ith encoder layer as initial state of ith decoder layer. This reasoning is presented in more formal way in 5.2

$$\begin{aligned}\tilde{\mathcal{X}}_{t+1}, \ldots, \tilde{\mathcal{X}}_{t+K} &= \arg\max_{\mathcal{X}_{t+1},\ldots,\mathcal{X}_{t+K}} p(\mathcal{X}_{t+1}, \ldots, \mathcal{X}_{t+K} \mid \hat{\mathcal{X}}_{t-J+1}, \hat{\mathcal{X}}_{t-J+2}, \ldots, \hat{\mathcal{X}}_t) \\ &\approx \arg\max_{\mathcal{X}_{t+1},\ldots,\mathcal{X}_{t+K}} p(\mathcal{X}_{t+1}, \ldots, \mathcal{X}_{t+K} \mid f_{encoding}(\hat{\mathcal{X}}_{t-J+1}, \hat{\mathcal{X}}_{t-J+2}, \ldots, \hat{\mathcal{X}}_t)) \\ &\approx g_{forecasting}(f_{encoding}(\hat{\mathcal{X}}_{t-J+1}, \hat{\mathcal{X}}_{t-J+2}, \ldots, \hat{\mathcal{X}}_t))\end{aligned} \tag{5.2}$$

where we try to maximzie the probability of predicted frames in the decoder given encoder output which corresponds to simple function composition of two sub-nets. Figure 5.2 shows the actual model from the paper where two stacked ConvLSTM layers were used.

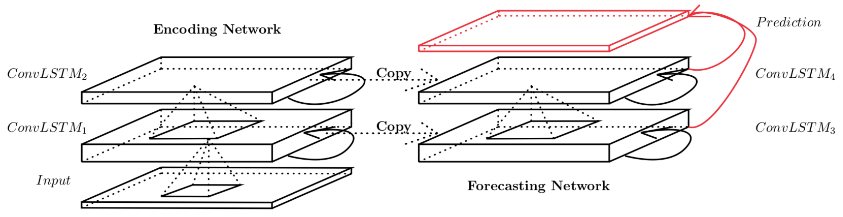


Figure 5.2: Encoder decoder structure for radar maps nowcasting [5]

5.1.2 Data splitting-strategy and training

[5] trained and evaluated his model on 97 rainiest days from the dataset and decided to randomly select $\frac{4}{6}$ of any given day for training and left out $\frac{1}{6}$ for testing and evaluating the model. As rainfall events in their data-set were also very short-lived, they decided to use 5 timestamps measurements as input to the encoder and tried to predict 15 future frames. What is more, they were interested in predicting whether

it will be raining only over 0.5mm/h and therefore they could use a cross-entropy loss function. This seems to be quite a limited approach for now-casting as pointed out in subsequent work by [10] as in the world, we are often interested in more fin-grained predictions. We now move onto explaining [10] formulation of a loss function as it will be more suitable for our problem.

5.1.3 Balanced loss function

[10] improved ConvLSTM model and one of the major innovations in that work was to use a different loss function for calculating the error on predicted frames. Their first idea was to use a mean squared error function similar to 4.1 and calculate error for each pixel in the frame and then average that over all pixels in the frame. However, data-set used in that study was much larger than in the original paper and had a highly imbalanced distribution of precipitation with majority of data being no-rain or very low-rain frames. Their work's focus was on detecting heavy rainfall events as their are more dangerous to people and harder to deal with. For that reason, they introduced balanced mean squared error function (B-MSE) and balanced absolute error (B-MAE) functions.
Specifically, they assigned a weight $w(x)$ to each pixel according to its rainfall intensity x:

$$w(x) = \begin{cases} 1, & x < 2 \\ 2, & 2 \leq x < 5 \\ 5, & 5 \leq x < 10 \\ 10, & 10 \leq x < 30 \\ 30, & x \geq 30 \end{cases}$$

The resulting B-MSE and B-MAE scores were then computed as

$$\text{B-MSE} = \frac{1}{N} \sum_{n=1}^{N} \sum_{i=1}^{480} \sum_{j=1}^{480} w_{n,i,j} (x_{n,i,j} - \hat{x}_{n,i,j})^2$$

$$\text{B-MAE} = \frac{1}{N} \sum_{n=1}^{N} \sum_{i=1}^{480} \sum_{j=1}^{480} w_{n,i,j} |x_{n,i,j} - \hat{x}_{n,i,j}|$$

where N is the total number of frames and $w_{n,i,j}$ is the weight corresponding to the (i, j)th pixel in the nth frame - each frame is 480 pixels high and wide.

5.2 Our data-set limitations

In section 3 we described structure and origin of the data. We said models explored in this project require that for a given time-stamp, there is available data from all sensors. This is because for model in 6.4 and 5.3.4 length on input tensor must be equal to 51 as it determines size of matrices inside of LSTM layer. We also want to offer a fair comparison of models that differ only by the architecture not amount of data. Furthermore, after initial experiments, we saw that 15 forecasted timestamps (as [5] did) given 5 input measurements does not yield satisfying results and decided to focus on predicting only 10 steps into the future. Finally as most of rain-data is zero valued, we decided to resample 5 minutes measurements and sum them over 12 minutes intervals as well as restrict the dataset to times where there is a rainfall event (defined as rainfall exceeding 0.5mm/h) at at least one of the sensors over 15 considered timestamps (5 for input, 10 for output). Thus, at the end we had 12471 data points (where each data point is a sequence of 15 measurements) and put 70% of that in training set and leave out 15% for both validation and testing. This is true for the data that includes rainfall measuremnets only so that all of our input feature maps to the model are 1-D. Distribution of dataset obtained by the procedure described above is shown in figure 5.3.

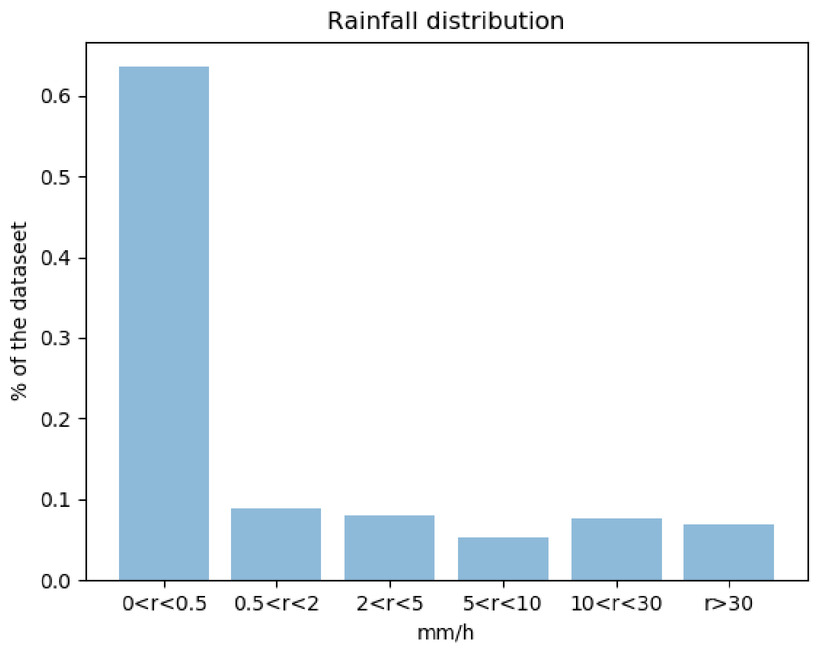


Figure 5.3

5.3 Simple memory models

As Singaporean sensors network presented in 3.1 do not have grid-like structure, we are unable to put priors of translation and local deformation on our data and therefore can not use standard convolution for extracting spatial information. For this reason, we first propose a different way of including geo-relationships data in the forecaster model and then present how two simple encoder-decoder structures can be defined for the Singaporean data.

5.3.1 Normalization

Before going into memory models we note that all our data in this project is normalised using the following equation:

$$X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{5.3}$$

where X_{min} and X_{max} are minimal and maximal amount of rainfall in the training set and X_{scaled} is normalized measurement.

5.3.2 Inclusion of auxiliary information

Liao [17] presents an extensive research on the inclusion of different types of auxiliary information in memory models. Main focus of his study is on predicting traffic conditions from online route queries but it can be extended for nowcasting models as well. In our model, one type of auxiliary information will be globally normalized geo-coordinates of weather stations. They will x, y values after the projection and we will call them $aux_{\text{x}}, aux_{\text{y}}$. Furthermore, as discussed in 3.2.3, we noticed that rainfall in Singapore varies widely on yearly and diurnal basis. It fluctuates differently based on the time of the day but this is further determined by the season of the year (whether it is a monsum season or not). Thus, we add two more pieces of information to the model, namely:

$$aux_{\text{year}} = \frac{\text{day of the year}}{\text{number of days in the year}}$$

$$aux_{\text{day}} = \frac{\text{minute of the day}}{\text{number of minutes in the day} = 60 * 24}$$

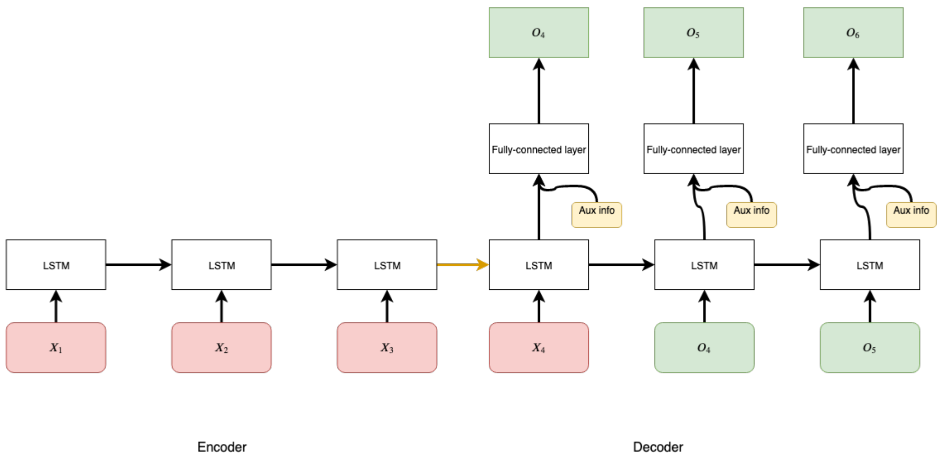


Figure 5.4: Auxiliary info inclusion for encoder-decoder structure - idea based on [17]

Both models in this section will leverage same encoder-decoder structure which is presented in figure 5.4 and use Relu activation function on the last layer as rainfall cannot really have negaative values. In this section, we focus on describing models features and explaining some fundamental assumptions that they make on the data. We discuss models performance in section 7, but it is worth noting that models in this section under-perform in comparison with those introduced in chapter 6.

5.3.3 Single sensor LSTM

The simplest model tested in the course of the project assumed that there is no local spatial correlation between sensors in the network. Instead, it is trained on single sensor points only and the geo-coordinates of the sensor are included in the auxiliary information tensor which gets appended to the outputs of the decoder and gets passed through the final fully-connected layer. As it is natural that rainfall intensity can not be negative, final FC layer is activated by the RELU function. Having said that, each X_i of encoder's input is a one-dimensional tensor containing measurements from a single sensor at a given time-stamp. Dimensional of a hidden state H on each level of the architecture is fine-tuned in 5.4, but given low number of dimensions in the input, we expect that the best value will not exceed 8.

5.3.4 Fully connected LSTM

This kind of model assumes equal local correspondence between each sensor in the network. For this reason, each X_i in the encoder is a tensor of length 51 (number of sensors in the network) and has a linear connection to all other 50 sensors in the network. Hidden state dimensions are also determined by the procedure described in 5.4. Global geo-coordinates and time auxilary information are included in the model's in the same manner as in 5.4.

5.4 Fine-tuning the model

As now-casting is largely undeveloped area of study [10], there is no obvious way of choosing hyper parameters such as dimensionality of a hidden state, number of layers, or a choice of a loss function (between B-MSE and regular MSE). For this reason we decided to use Neural Network Intelligence (NNI) [18] toll created by Microsoft as part of auto-ml project. In later sections we briefly describe what regularisation technique we used in order to achieve a better performance of the model.

5.4.1 Neural Network Intelligence (NNI)

Neural Network Intelligence (NNI) is an open-source toolkit created by Microsoft for tuning machine learning systems parameters. It allows a developer to define a hyper-parameter search space from which NNI tuner will sample values in a way such that some a single pre-defined final metric is optimized - we talk about specific tuner for this task later in 5.4.2. NNI has also a concept of an assessor that can

interrupt any given trial before it finishes if the model performs badly on some predefined intermediate metrics in comparison to previously trained models. We used a validation loss as as an intermediate metric and the test loss as a final metric. We run two separate training regimes - one with regular MSE loss and the other one with Balanced MSE with same weights at the same thresholds as in [10]. We then pick the best performing model from each regime and compare four metrics defined in 4. The results for all models are presented in 7.
Figure 5.5 presents an architecture of the NNI tool. The whole package comes with an interactive WebUI allowing to display hyper-parameters chosen by the tuner as well as models performance measured by metrics. The whole training and fine-tuning process run on GeForce GTX TITAN X kindly provided by the Department of Computing at Imperial College.

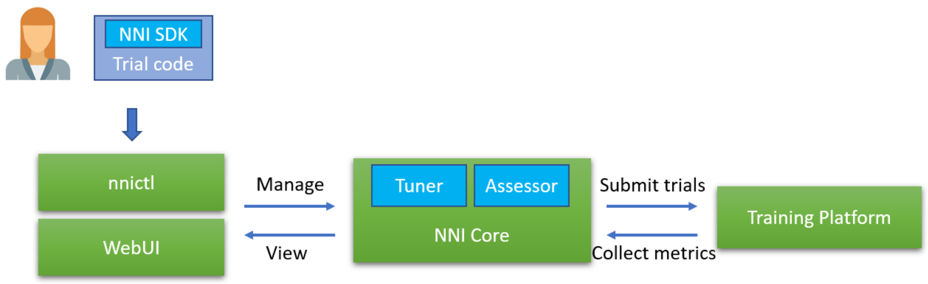


Figure 5.5: NNI architecture

5.4.2 Tuner: Tree-structured Parzen Estimator (TPE)

The tuner used in this project is of the more basic ones and it is called Tree-structured Parzen Estimator [19]. TPE sequentially construct models to approximate the performance of hyper-parameters based on historical measurements, and then subsequently choose new hyper-parameters to test based on this model. The TPE approach models $P(x|y)$ and $P(y)$ where x represents hyper-parameters and y the associated metric (in this, case validation loss). $P(x|y)$ is modeled by transforming the generative process of hyper-parameters, replacing the distributions of the configuration prior with non-parametric densities [18]. This is further described in [19] and in this project we use NNI's default implementation of that algorithm as it has proven to work much better than a random search, especially given limited computational resources [18].

5.4.3 Improving validation and test sets performance

Universal theorem of approximation [20] states that a feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of R^n, under mild assumptions on the activation function. Our models (containing multiple layers) could therefore achieve 0 as a value of loss function. However, we are interested in models performance on the test set. This motivates introduction of a dropout in the recurrent layer. Dropout trains the ensemble consisting of all sub-networks that can be formed by removing non-output units from an underlying base network [6]. This process is presented in the figure 5.6. It prevents over-fitting because individual neurons become less co-dependent on each other which curbs the individual power of each neuron leading to best performance on the unseen data. It is also more efficient than training multiple ensemble models as it does not require more computing power than training a single model. In our setting, we introduce a drop-out between hidden states only 2.10 and place a probability p that each neuron will not get activated (will be dropped) during the training phase. In the test and validation stages, we do not dropout any neuron as shown in the figure 5.7.

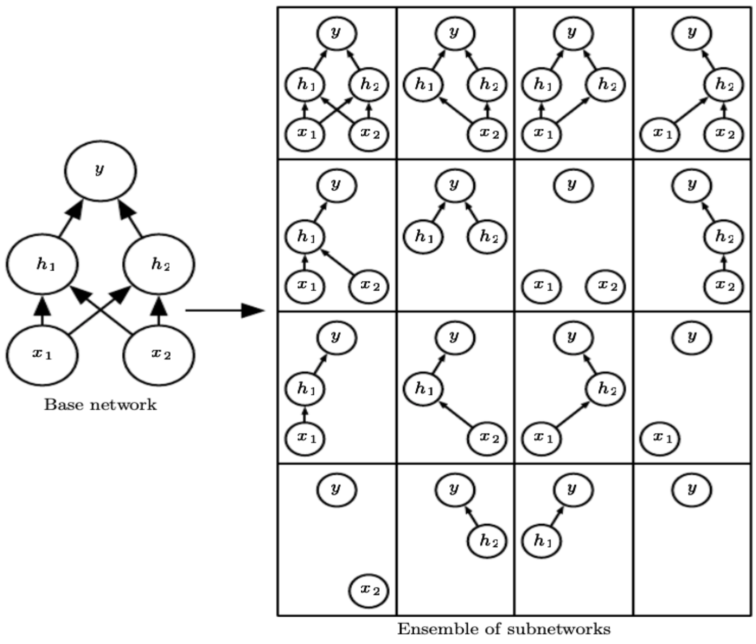
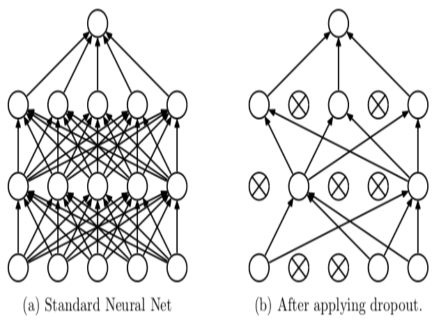
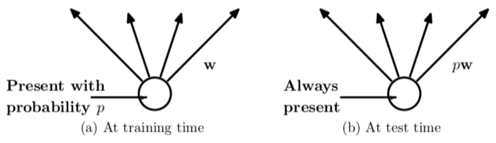


Figure 5.6: One base network (left) and all possible combinations of networks with dropped neurons (right) [6].

We fine-tune values of drop-out ($0.1 < p < 0.9 =$) using NNI toolkit and present best values for each model 7.





(a) neural network with dropped versus the same net with dropout applied

(b) Training vs testing of a layer with a dropout

Figure 5.7: Dropout basics [21]

Chapter 6

Going beyond Euclidean data

Models introduced in 5 do not leverage at all (single sensor model) or leverage in a very inefficient manner (fully connected model with 51 sensors). For that reason, in this section we introduce the concept of geometric machine learning - meaning deep learning on non-Euclidean domains. We will model sensors network as a connected directed graph with node sensors as vertices and introduce convolution and pooling operators in such domain. We begin this section by explaining related graph theory and then move onto translating 2.2.1 to the graphs domain and introducing Gaussian Mixture Model [9] convolution and TopKPooling operators [22]. We conclude by showcasing two models leveraging local spatial information of sensors network in Singapore.

6.1 Graphs

We are interested in analyzing signals defined on directed, connected, weighted graph $G = (V, E, W)$, which consists of a set of vertices V with $|V| = n$, a set of edges E, and a weighted adjacency matrix W. If there is an edge $e = (i, j)$ connecting vertices from i to j, the entry $W_{i,j}$ represents the weight of the edge; otherwise, $W_{i,j} = 0$ [23] . We also need to define the functions on nodes and vertices of the graph. Let $f : V \to R^n$ and $F : E \to R^n$ real functions where ith component represents functions values at each vertex and edge respectiely. [11].

6.1.1 Generalisation of convolution properties for graphs

There exists two main aproaches for defining convolution operations on graphs:

- Spectral methods on Euclidean domains interpret convolutions as linear operators that commute with the Laplacian operator. Therefore, in the graphs domain, spectral convolutions focus on linear operators that commute with the graph Laplacian. This property, in turn, implies operating on the spectrum of the graph weights, given by the eigenvectors of the graph Laplacian [11]. However, that class of methods iis able to capture local positional relations as well as unable to adapt to a changing domain [23]. These are severe limitations from our perspective as intend to extract local geo-spatial information from the sensors network. Moreover, if any of the sensors becomes unavailable at any points in time, the model would be able to adjust for that as it operated on fixed domain. For this reason, we will not analyze spectral methods in this project and instead focus on spatial definition of convolution.

- Spatial methods, in contrast, leverage geometric prior of translation invariance and parameter sharing across the whole Euclidean domain and therefore can be thought of as passing a template at each point of the domain and recording the correlation of the template with the function at that point [11]. This boils down to extracting an image patch, convoluting it with the kernel filter and moving onto the next window of the picture. Irregular domain of graphs and typical lack of a global coordinate system require us to represent the patch extraction in some local intrinsic system of coordinates. We define that system as d-dimensional pseudo-coordinates $\boldsymbol{u}(i, j) \in R^d$ for each vertex i and j. If there exists an edge between two vertices its value $u(i, j)$ is represented by an edge attribute - this can be simple Cartesian distance between vertices, some form of polar coordinates or vertex degree. If there is no edge between given vertices $u(i, j) = 0$. Now, that local pseudo-coordinate system is used to define a set of weighting functions $v_1(\boldsymbol{u}), \dots, v_j(\boldsymbol{u})$ localized to positions near x [11]. We thus define discrete graph convolution of signal f with kernel g to be:

$$(f \star g)(x) \quad = \quad \sum_j g_j D_j(x) f, \tag{6.1}$$

where D_j is a patch operator specified by previously mentioned weighting functions $\boldsymbol{v}(\boldsymbol{u})$:

$$D_j(x) f \quad = \quad \int_{\mathcal{X}} f(x') v_j(x, x') \delta x', \quad j = 1, \dots, J, \tag{6.2}$$

Such formation shares two important characteristic with convolution 2.2.1 on Euclidean domains. It has constant number of parameters equal to the number of weighting functions and is localized by construction [11]. In simpler terms it can be thought of as a simple non-linear neighbourhood information aggregation which is presented in the figure 6.1. Simple kernel convolution on 1-D graph signal with a single kernel is shown in 6.2. In general, all spatial graph convolution frameworks amount to differnet choice of kernel functions. In this project, we consider the most general one called Gaussina Mixture Model (MoNet) which is discussed in the subsequent section.

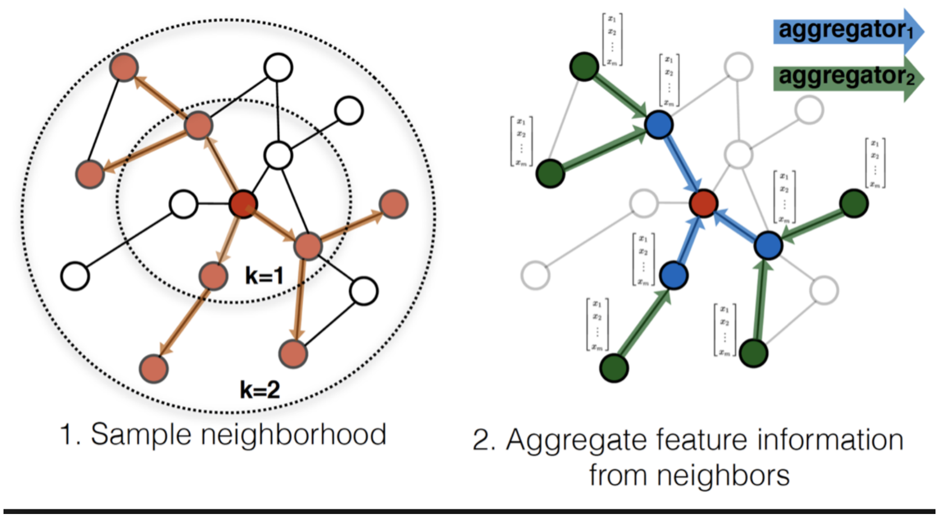


Figure 6.1: Graph convolution operation as neighbouring information aggregation [24]

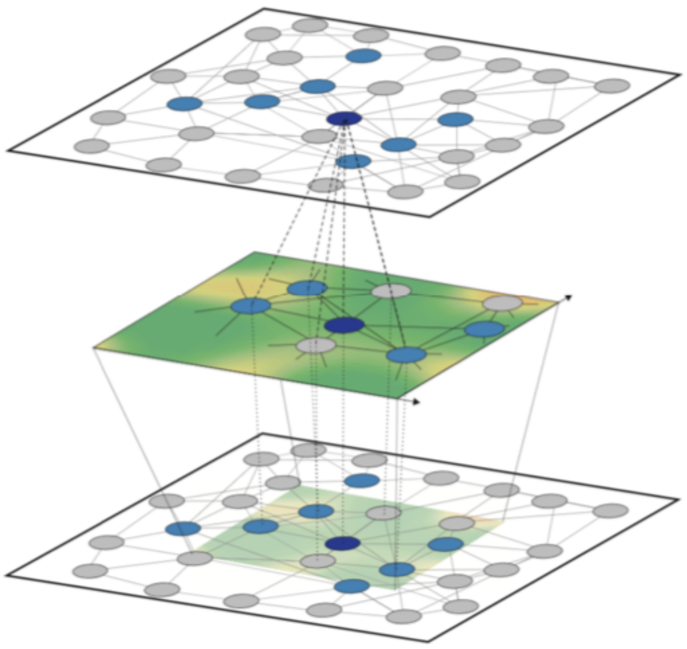


Figure 6.2: Graph convolution operation as neighbouring information aggregation [24]

6.2 MoNet Convolution

Most of graph convolution frameworks propose using a fixed kernel specifically tailored to the underlying application. Monti [9] came up with a different approach where not only kernels g_i are learn-able but so is the set of weighting functions $\boldsymbol{v}$ so that:

$$v_j(\mathbf{u}) = \exp\left(-\tfrac{1}{2}(\mathbf{u} - \boldsymbol{\mu}_j)^\top \boldsymbol{\Sigma}_j^{-1}(\mathbf{u} - \boldsymbol{\mu}_j)\right) \tag{6.3}$$

where covariance matrices $\boldsymbol{\Sigma}_1, \dots, \boldsymbol{\Sigma}_J$ and mean vectors $\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_J$ are learned. As this choice allow interpreting convolution as a mixture of Gaussians, this aproach is called Gaussian Mixture Modell (MoNet). In this project, we adopt [25] choice of local pseudo-coordinates and define them to be:

$$d_{i,j} = e^{\frac{-\frac{1}{2}||\boldsymbol{x}_i - \boldsymbol{x}_j||^2}{\sigma^2}} \tag{6.4}$$

$$u_{i,j} = \frac{e^{d_{i,j}}}{\Sigma_k e^{d_{i,k}}} \tag{6.5}$$

where $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are geo-coordinates of vertices (sensors) and σ is a learnable scalar parameter controlling the locality of the kernel and how fast the information is spread across distant vertices.

6.2.1 Top-k Pooling

Although none of Euclidean models in 5 has relied upon pooling operation (even though some approaches such as [10] have leveraged that technique), the network introduced in 6.4 relies on graph pooling in order to reduce number of parameters. Thus, we introduce top-k pooling operation [22] which reduces the number of nodes n in the input so that the output graph has kN vertices where $k \in (0, 1]$.

A learneable *projection score* vector $\boldsymbol{p}$ determines which nodes are dropped so that the operation for input signal on vertices $\mathbf{X}$ with an adjacency matrix $\mathbf{A}$ results in the output nodes $\mathbf{X}'$ and transformed adjacency matrix $\mathbf{A}'$:

$$\boldsymbol{y} = \frac{\mathbf{X}\boldsymbol{p}}{||\boldsymbol{p}||} \qquad \boldsymbol{i} = \text{top-}k(\boldsymbol{y}, k) \qquad \mathbf{X}' = (\mathbf{X} \odot \tanh(\boldsymbol{y}))_{\boldsymbol{i}} \qquad \mathbf{A}' = \mathbf{A}_{\boldsymbol{i},\boldsymbol{i}} \tag{6.6}$$

where top-k selects the top-k indices from a given input vector and $\odot$ is Hadamard product, and $\cdot_{\boldsymbol{i}}$ is an indexing operation which takes slices at indices specified by $\boldsymbol{i}$.

6.3 Sensors network graph

51 rain sensors spread across Singapore are nodes in the graph and initially there exists a connection between each node so there are $51 \times 50 = 2550$ edges. However, we know that rainfall spatial relations 3.2.2 have a limited radius and therefore we use K-NN [CITE KNN] to select only k closest sensors for each node. k is a hyperparameter fine-tuned with the use of NNI toolkit. For connected nodes, we calculate an edge attribute as shown in 6.6 and measured rainfall amount at each timestamp repsents signal function at each vertex.

6.4 GraphConcat

First model leveraging graph convolution is largely based on the work of [17] that used graph CNN (spectral methods) to embed traffic of neighbouring road segments. We follow similar formulation but instead rely upon MoNet convolution layer as well as Top-k Pooling layer. Input to the encoder is a concatenation of two tensors:

- Same tensor as in 5.3.4 consisting of rain measuremets from all 51 sensors at a given timestamp
- Tensor resulting from passing the graph construction from 6.3 through a single Gaausina Mixture Model layer and a Top-k Pooling layer with $k = 0.5$

This momodel is show in Figue 6.3. We still include auxiliary info of global geo-coordinates and time on the final filly connected layer. The activation on that layer is Relu (same as in 5).

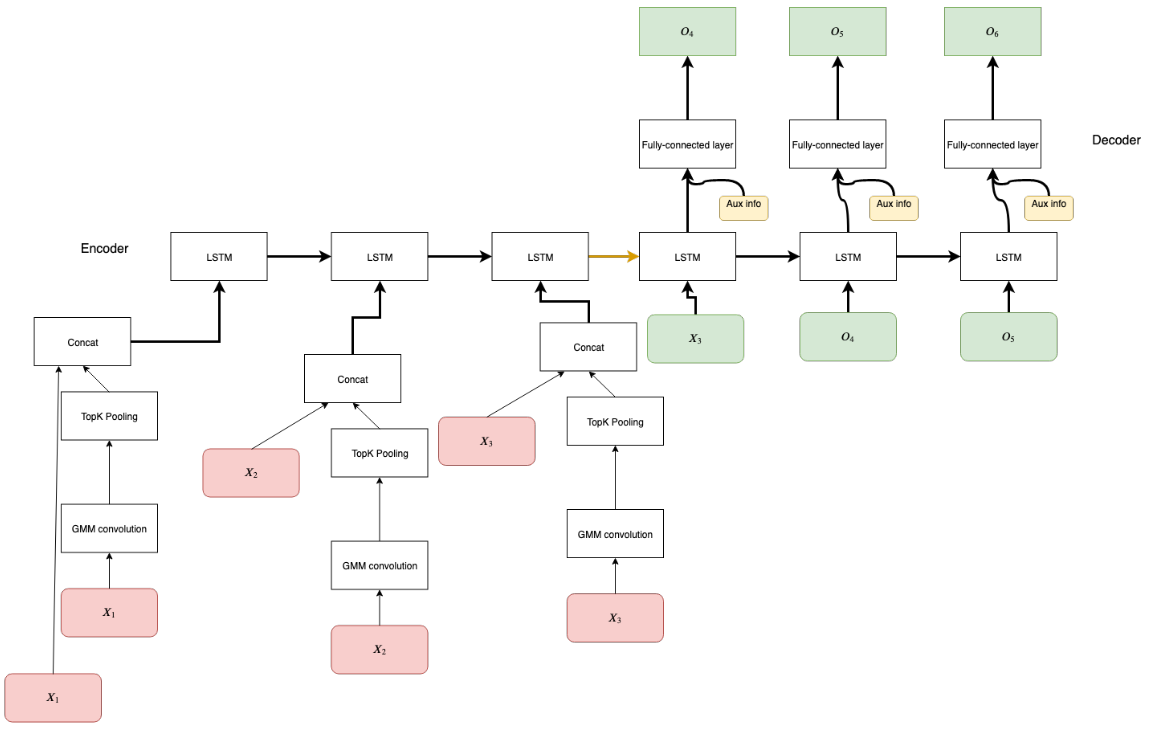


Figure 6.3: GraphConcat model

6.5 UmbrellaNet

Having seen that convolution based models outperform fully-connected memory networks on 2-D radar maps, it is natural to try to define a convolution memory model operating directly on graphs. We take the same approach as [5] and introduce graph convolutional connections in the state-to-state and input-to-state transitions so that all inputs $\mathcal{X}_1 \dots X_t$, cell outputs $\mathcal{C}_1 \dots C_t$ and hidden states $\mathcal{H}_1 \dots H_t$ are values of

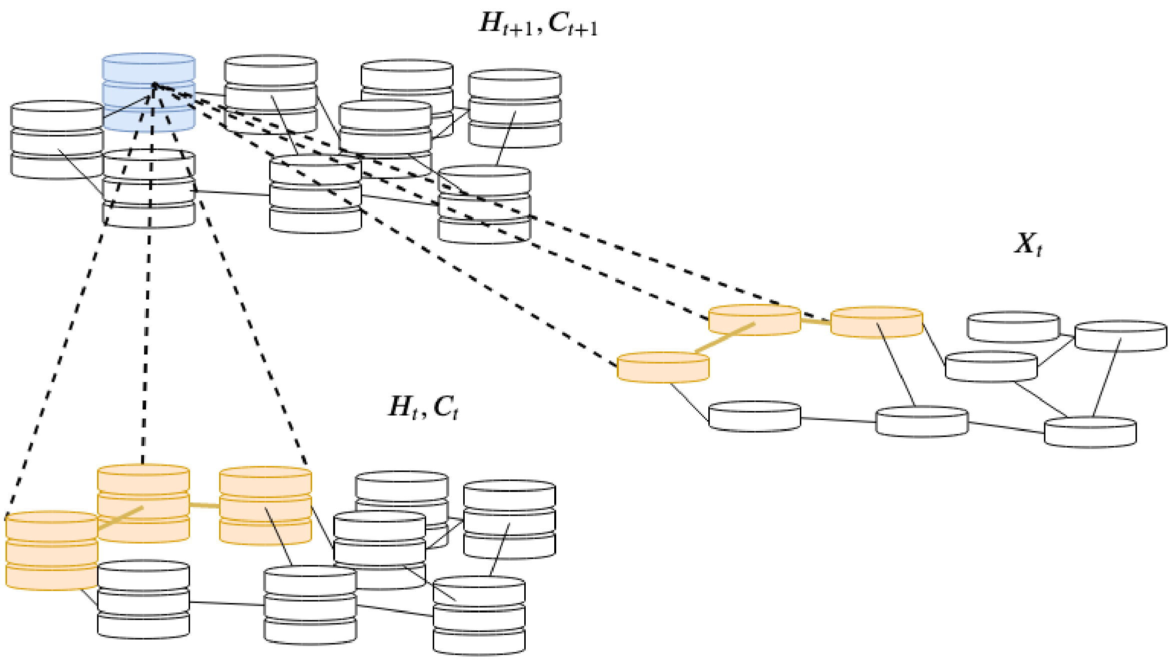


Figure 6.4: MoConvLSTM model model

graph vertices with the equal number of nodes and same pseudo-coordinates $\boldsymbol{u}(i,j)$ for all vertices i, j n the graph. Key equations of MoConvLstmLayer are shown below:

$$\begin{aligned}
i_t &= \sigma(GMM_{xi} * \mathcal{X}_t + GMM_{hi} * \mathcal{H}_{t-1} + W_{ci} \circ \mathcal{C}_{t-1} + b_i) \\
f_t &= \sigma(GMM_{xf} * \mathcal{X}_t + GMM_{hf} * \mathcal{H}_{t-1} + W_{cf} \circ \mathcal{C}_{t-1} + b_f) \\
\mathcal{C}_t &= f_t \circ \mathcal{C}_{t-1} + i_t \circ \tanh(GMM_{xc} * \mathcal{X}_t + GMM_{hc} * \mathcal{H}_{t-1} + b_c) \\
o_t &= \sigma(GMM_{xo} * \mathcal{X}_t + GMM_{ho} * \mathcal{H}_{t-1} + W_{co} \circ \mathcal{C}_t + b_o) \\
\mathcal{H}_t &= o_t \circ \tanh(\mathcal{C}_t)
\end{aligned} \tag{6.7}$$

where GMM_ij represents corresponding Gaussian Mixture Model convolution. Figure 6.4 shows graph convolutional connections between hidden states and inputs. One dimensional signal on each vertex (input feature maps) produces 3 output features maps in hidden states and cell outputs. This formulation corresponds to multi-channel CNN on Euclidian Domain. Now, UmbrellaNet is an encoder-decoder memory model as the one defined in 5.4. However, as the main focus of this project is on proposing a new graph convolutional LSTM, it does not include a dropout between hidden states as well as no auxiliary information and its final activation function is Relu. Number of kernels used for GMM convolution as well as number of layer is determined by hyper-parameter fine-tuning with NNI.

Chapter 7

Evaluation

In this section, we evaluate project's objectives, namely creating an end-to-end now-casting model predicting future precipitation in the area of Singapore. We begin by pointing out that there is a lot of missing data in the environmental dataset and emphaise importantce of this for developing well performing graph model. Then, we move onto a quantitative analysis of 4 developed models performance and contrast that with state-of-the art approach based on 2-D radar maps. Finally, we speculate why UmbrellaNet and MoConvLSTM layer under-perform, even though we expected it to achieve best results. We conclude by pointing out future directions of work for sensors now-casting models as well as graph convolutional memory models.

7.0.1 Aside on environmental dataset problem

In section 3.2.4, we saw that other environmental factors such as temperature or humidity are very likely to impact future rainfall. Unfortunately, after applying the data splitting strategy as describe above, the environmental data-set has only 343 data-points and 13 sensors which is not enough for our machine learning models to properly train and evaluate results. This constitutes a major disadvantage of this approach as we were not able to achieve state-of-the art results for rain only data but predicted that inclusion of more environmental data would lead to significant improvements due to high correlations values.

7.1 Quantitative comparison

Here, we first provide a list of hyper-parameters fine-tuned by NNI for each model considered. Then, we show evaluation metrics (only for timestamp $k = 10$) for best configuration of each model and explore final network in in more detail. We have used 50 epochs for training if batch size was bigger than 8 and 25 if it was smaller. It is worth noting that we tested models with two different loss functions (regular MSE and B-MSE with same weights [10]) and a vanilla mean squared erroor function outperform B-MSE for each model. One of the reasons it could have happened is that weights were themselves non-learnable and just the values were copied from the other paper that used a different dataset.

7.1.1 Models introduces in this project

We begin by introducing a hyper-parameter search space used for fine-tuning models by NNI and showing best configuration and reporting the value of final loss evaluated on the test set.

- Single sensor network defined in 5.3.3:

  1. encoder and decoder dropout values between hidden connection in LSTM controlling power of regularisation for better performance on the validation and test sets: $dropout \in \{0, 0.3, 0.4, 0.5, 0.6, 0.7, 0.9\}$
  2. number of layers in encoder and decoder controlling complexity and expressiveness of the memory layer $no \in \{2, 3, 4, 5, 6, 7\}$
  3. hidden dimensions controlling complexity and expressiveness of the memory layer $h_{dim} \in \{1, 2, 4, 8, 16, 32\}$
  4. batch batch controlling GPU utilisation and sensitiveness for local minimas of the loss function $batch \in \{4, 16, 32, 64, 128, 256, 512\}$
  5. learning rate controlling speed of learning and sensitiveness for local minimas of the loss function $lr \in \{0.01, 0.001, 0.0001, 0.00001\}$

  Best configuration is $dropout_{encoder} = 0.4$, $dropout_{decoder} = 0.7$, $no = 4$, $h_{dim} = 32$, $batch = 32$, $lr = 0.0001$. Final values of MSE was 3.66.

- Fully Connected model defined in 5.3.4

  1. encoder and decoder dropout values: $dropout \in \{0, 0.3, 0.5, 0.6, 0.7, 0.9\}$
  2. number of layers in encoder and decoder: $no \in \{2, 3, 4, 5, 6, 7\}$
  3. hidden dimensions controlling complexity and expressiveness of the memory layer $h_{dim} \in \{16, 32, 64, 128, 256, 512\}$
  4. batch batch controlling GPU utilisation and sensitiveness for local minimas of the loss function $batch \in \{2, 4, 16, 64, 256, 512, 1024\}$
  5. learning rate: $lr \in \{0.01, 0.001, 0.0001, \}$

  Best configuration is $dropout_{encoder} = 0.9$, $dropout_{decoder} = 0$, $no = 5$, $h_{dim} = 32$, $batch = 32$, $lr = 0.001$. Final values of MSE was 1.14.

- GraphConcat defined in 6.4 same parameter space as for Fully-connected model plus number of neighbours used to determine extent of a single GMM convolution in graph layer: $k_{neighbours} \in \{3, 6, 10, 20, 40\}$

  Best configuration is $dropout_{encoder} = 0.5$, $dropout_{decoder} = 0.3$, $no = 4$, $h_{dim} = 64$, $batch = 1028$, $lr = 0.01$, $k_{neighbours} = 10$,. Final values of MSE was 0.66.

- UmbrellaNet defined in 6.5 same parameter space as for GraphConcat but with no dropout. Best configuration is $no = 3$, $batch = 4$, $lr = 0.01$, $k_{neighbours} = 6$,. Final values of MSE was 1.04. The best performing model is GraphConcat network. We will now discuss its training regime and then move onto a comparative analysis.

7.1.2 Training GraphConcat

Figures 7.1 and 7.2 present loss functions on training and validation datasets. Training loss has a smaller final value, but it is more volatile for fluctuation over 50 epochs. This is somehow surprising as we would expect a much smoother monotonically decreasing curve. One explanation for that is that size of training set is 5 times larger and the during sampling from it we shuffle the data (as it is a standard procedure for neural network training) and for the validation set we do not shuffle the data. Another explanation for that phenomena is different dropout behaviour during training and testing phases. This is a thing that needs further research as it is bit unusual and could impact the final results. On the other hand, we had expected that the final validation loss be bigger than the training one. This is very expected as evaluation happens on unseen data.

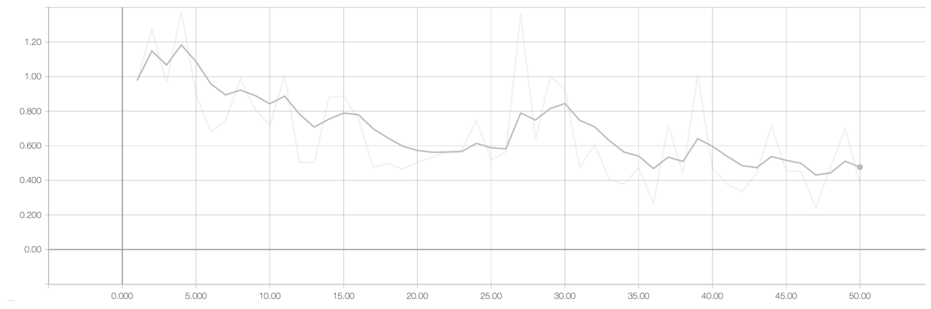


Figure 7.1: Train loss values vs number of epochs

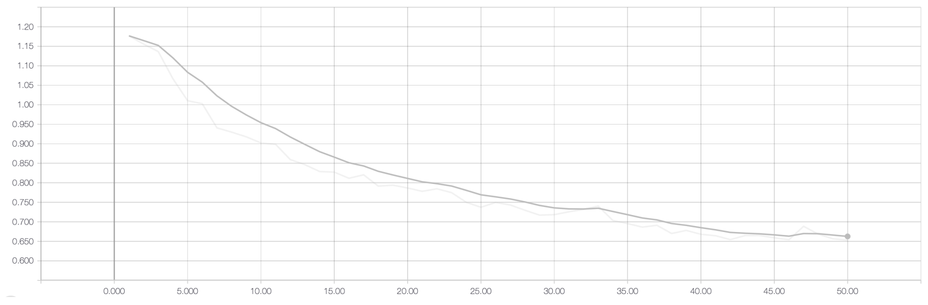


Figure 7.2: Validation loss values vs number of epochs

7.1.3 Comparative analysis

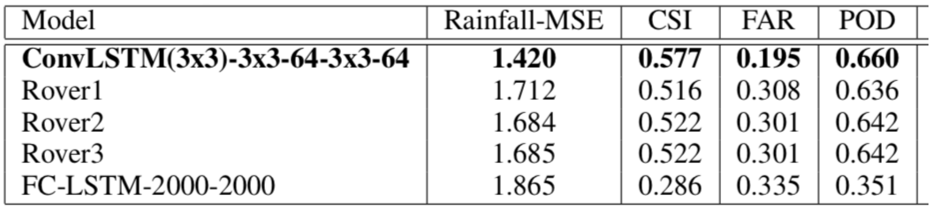
Here we compare our probability of detection and false alarm ratio metrics with [5] and [10] use results to assess GraphConcat skills for Heidki and CSI. As discussed in the evaluation methodology section, we calculate all metrics for 5 threshold values τ and at 3 time-horizons k. This is a good compromise between computational costs of calculation those metrics (CPU involved) and informative results. $\uparrow$ means that higher values of metric is better while $\downarrow$ the opposite. Both [5] and use [10] are interested in 4 hour horizon. Our model is evaluated only up to 2 hour horizon (15 timestamps of 12 minutes). Our models perform very poorly when trained with error function optimizing for 4 hour long prediction window. This makes the direct comparative analysis impossible but is enough to observe that our models underperform in comparison to 2-D convolutional models. Probability of detection should be interpreted as the capability of the the system to detect τ rainfall amount at a given time horizon k and is a measure of discrimination. Similarly, false alarm ratio measures reliability and is the number of false alarms divided by the total number of events forecast. They are both inequitable measures [26] meaning they do not account for chance occurrence of forecast (when there is a class in-balance, our random guess might have a good value of the metric) and are therefore imperfect scores. Nevertheless they allow us to evaluate our models comparatively to [5]. Figure 7.1 and 7.2 show our results evaluated at multiple time horizons$= k * 12$min. [5] used a single time horizon of 240min which is double of longest one at a single threshold of 0.5mm/h and achieved 0.66 and 0.2 for POD and FAR respectively. This is much better than the our models scores (0.54 and 0.68) at the same threshold at the time horizon of 120min. High value of false alarm ratio means that we are often predicting rain over the threshold of 0.5mm/h when it is not there but a small value POD suggest we are also not very good at detecting. That is the first sign of disadvantage of the GraphConcat.

Table 7.1: POD ↑

τ/k	1	5	10
0.5	0.81	0.67	0.54
2	0.58	0.41	0.29
5	0.60	0.44	0.32
10	0.64	0.49	0.34
30	0.74	0.58	0.38

Table 7.2: FAR ↓

τ/k	1	5	10
0.5	0.51	0.58	0.68
2	0.27	0.29	0.38
5	0.15	0.16	0.24
10	0.08	0.09	0.10
30	0.05	0.06	0.09



Model	Rainfall-MSE	CSI	FAR	POD
ConvLSTM(3x3)-3x3-64-3x3-64	**1.420**	**0.577**	**0.195**	**0.660**
Rover1	1.712	0.516	0.308	0.636
Rover2	1.684	0.522	0.301	0.642
Rover3	1.685	0.522	0.301	0.642
FC-LSTM-2000-2000	1.865	0.286	0.335	0.351

Figure 7.3: Results of [5]

We now evaluate our models on chance corrected metrics [26] of CSI and Heidki and compare our performance to [10]. Note that CSI is still not truly equitable measure because it has the disadvantage that equally unskillful forecasting systems, e.g. those that always predict occurrence and those that always predict non-occurrence yield a different score. Heidki score, in contrast is equitable, and therefore should be a primary source of comparison between different nowcasting methods.
In tables 7.3 and 7.4 we see evaluation metrics given our models and Figure 7.4 presents state-of-the-art results from [10]. The first trend worth noticing is that our models perform better at higher threshold than at lower ones. For example CSI at $\tau = 30$ for $k = 10$ is 0.37 while for $\tau = 0.5$ it is just 0.22. This in a direct compare to [10] and holds true for Heidke skill alike. One possible explanation for that is we built and to end-to-end directly predicting rainfall values whereas **??** used a mathematical expression to transform predicted pixel values of radar maps to actual rain. Their expression might not be as robust at higher threshold. Another possible reason is that our dataset distribution is different from theirs and has more higher intensity's values. In general, our model behave considerably worse at lower threshold (we analyze results for $k = 10$) τ but is better than radar approach for higher ones. It is important to remember that we compare four hour forecast window from the research paper with out 2 hour long prediction so we cannot directly say that our model is actually better. However, it gives an indication, that sensor network based nowcasting exhibits different characteristics from the radar map based.

Table 7.3: CSI ↑

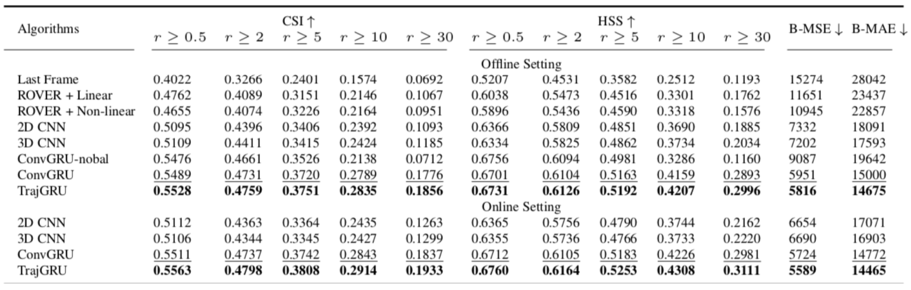
τ/k	1	5	10
0.5	0.45	0.3	0.22
2	0.48	0.35	0.26
5	0.54	0.41	0.29
10	0.61	0.47	0.33
30	0.72	0.51	0.37

Table 7.4: Heidke ↑

τ/k	1	5	10
0.5	0.51	0.47	0.26
2	0.62	0.49	0.38
5	0.69	0.64	0.44
10	0.79	0.63	0.49
30	0.83	0.68	0.54

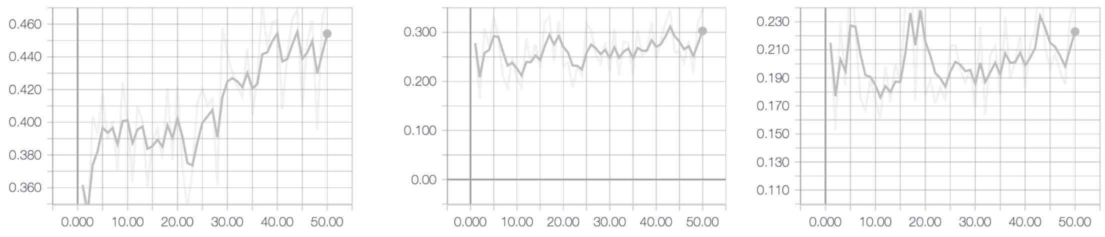
7.1.4 Discussion

Radar map approaches consistently outperform our approach. We believe that the major reason behind this is not having enough information about weather characteristics at a given sensor location. Having only 1 hour of precipitation history is

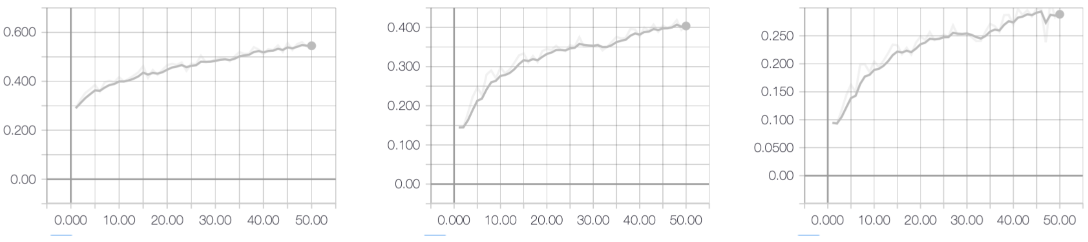


Algorithms	CSI ↑					HSS ↑					B-MSE ↓	B-MAE ↓
	$r \geq 0.5$	$r \geq 2$	$r \geq 5$	$r \geq 10$	$r \geq 30$	$r \geq 0.5$	$r \geq 2$	$r \geq 5$	$r \geq 10$	$r \geq 30$		
						Offline Setting						
Last Frame	0.4022	0.3266	0.2401	0.1574	0.0692	0.5207	0.4531	0.3582	0.2512	0.1193	15274	28042
ROVER + Linear	0.4762	0.4089	0.3151	0.2146	0.1067	0.6038	0.5473	0.4516	0.3301	0.1762	11651	23437
ROVER + Non-linear	0.4655	0.4074	0.3226	0.2164	0.0951	0.5896	0.5436	0.4590	0.3318	0.1576	10945	22857
2D CNN	0.5095	0.4396	0.3406	0.2392	0.1093	0.6366	0.5809	0.4851	0.3690	0.1885	7332	18091
3D CNN	0.5109	0.4411	0.3415	0.2424	0.1185	0.6334	0.5825	0.4862	0.3734	0.2034	7202	17593
ConvGRU-nobal	0.5476	0.4661	0.3526	0.2138	0.0712	0.6756	0.6094	0.4981	0.3286	0.1160	9087	19642
ConvGRU	0.5489	0.4731	0.3720	0.2789	0.1776	0.6701	0.6104	0.5163	0.4159	0.2893	5951	15000
TrajGRU	**0.5528**	**0.4759**	**0.3751**	**0.2835**	**0.1856**	**0.6731**	**0.6126**	**0.5192**	**0.4207**	**0.2996**	**5816**	**14675**
						Online Setting						
2D CNN	0.5112	0.4363	0.3364	0.2435	0.1263	0.6365	0.5756	0.4790	0.3744	0.2162	6654	17071
3D CNN	0.5106	0.4344	0.3345	0.2427	0.1299	0.6355	0.5736	0.4766	0.3733	0.2220	6690	16903
ConvGRU	0.5511	0.4737	0.3742	0.2843	0.1837	0.6712	0.6105	0.5183	0.4226	0.2981	5724	14772
TrajGRU	**0.5563**	**0.4798**	**0.3808**	**0.2914**	**0.1933**	**0.6760**	**0.6164**	**0.5253**	**0.4308**	**0.3111**	**5589**	**14465**

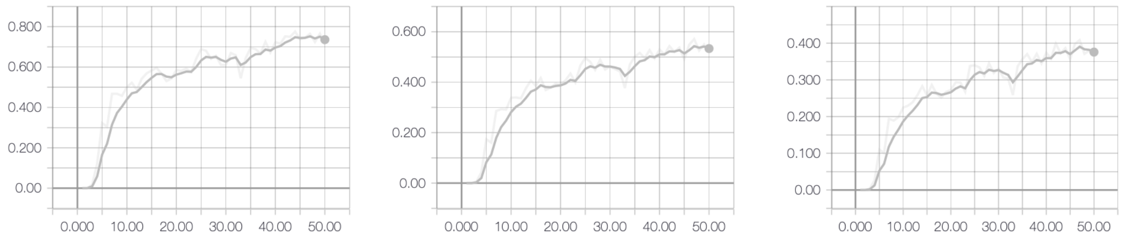
Figure 7.4: Results of [10]



(a) CSI at 0.5mm/h



(b) CSI at 5mm/h

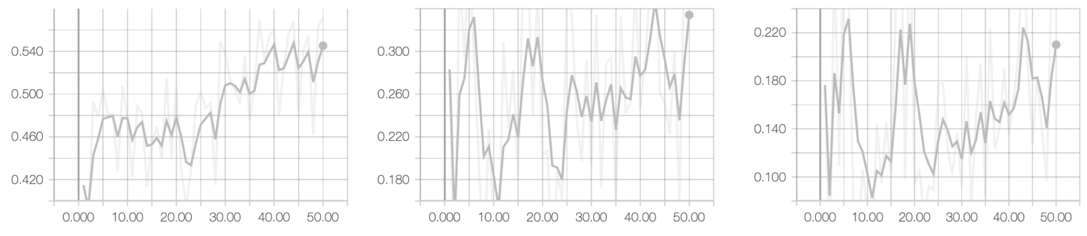


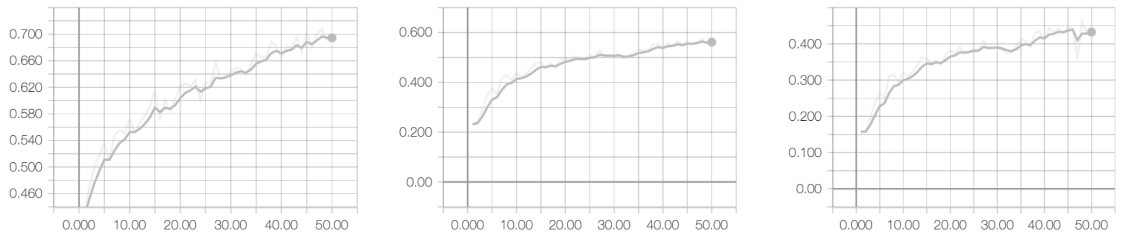
(c) CSI at 30mm/h

Figure 7.5: CSI during training

not enough to predict future rainfall in 2 or 4 hour window. Section 3.2.4, however, strongly suggested that inclusion of more environmental data such as humidity or temperature could increase the performance of rainfall nowcasting models.

What is more, we had believed that UmbrellaNet (Section 6.5) would be the best of all models as it is strongly modelled on existing state-of-the art techniques. One of possible explanations for that is one-dimensional signal (as most graph convolutions work on multi dimensional data) at graph vertices, not enough of nodes in the graph, or wrong choice of parameters (this would surprising as fine-tuned the network with NNI). Figure 7.7 show the variation training and validation losses over 50 epochs of


(a) HEIDKE at 0.5mm/h


(b) HEIDKE at 5mm/h

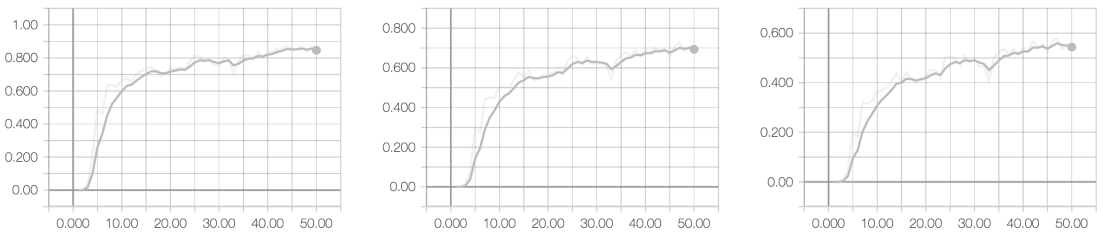

(c) HEIDKE at 30mm/h

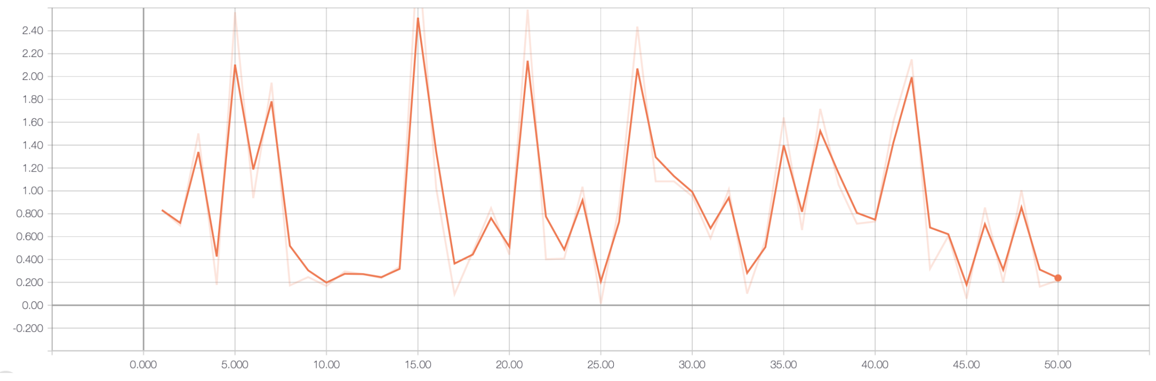
Figure 7.6: Heidke during training

training UmbrellaNet and should be noted that that high volatility of the training loss and non-improving of the validation is another symptom of problems with the MoConvLSTM layer and UmbrellaNet.

7.2 Qualitative analyis

7.2.1 Future work

Further research into now-casting models based on sensors network is needed because the ever increasing adoption of remote sensing techniques will provide researches with a more granular spatial data about environmental factors which lead to better predictions at smaller radius of forecast. This, in turn, applies that we will be able to forecast rainfall with a greater spatial precision enabling a whole wave of new sustainability focused applications (such as managing tanks of water in Singapore). The focus of further research should lie in exploring spatial convolution on graph strutted data and applying that to memory models. This enables to leverage spatial local patters in weather as well as not rely on constant domain assumption (in contrast to spectral methods) which will increase the utility of models as larger number of sensors will likely cause that some of that are unavailable at any times.


(a) Training loss for UmbrellaNet

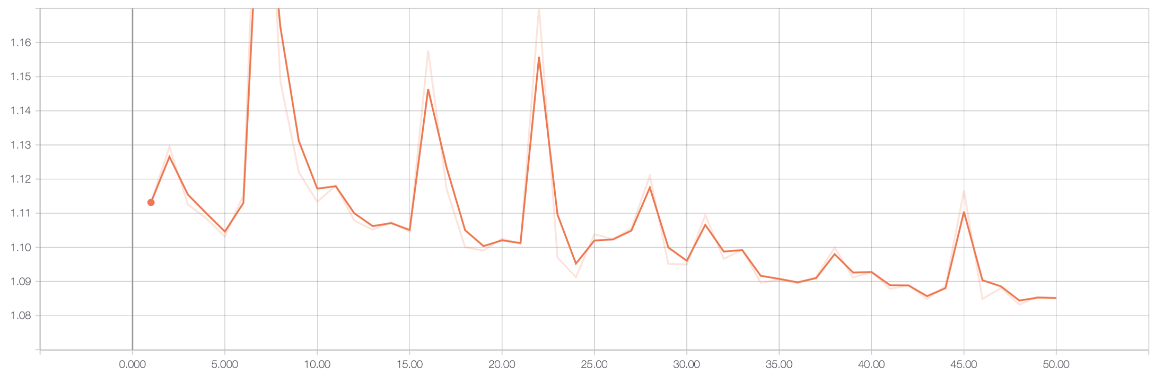

(b) Validation loss for UmbrellaNet

Figure 7.7: UmbrellaNet mean squared error functions

7.3 Conclusion

To the best of our knowledge, our project is a first attempt of creating an to end-to-end nowcasting models for rainfall prediction. We have demonstrated that one dimensional rain data is not sufficient for achieving satisfying results but provided strong evidence that inclusion of more features could be very beneficial. We also designed first spatial convolutional memory layer that is able to capture spatial relations between nodes signals changing over time but failed to properly train it. Instead, the best performing model for nowcasting we managed to train is Graph-Concat and it achieves MSE value of 0.66 for 2 hour long prediction window.

Bibliography

[1] U. General Assembly, "Resolution 64/292 - The human right to water and sanitation," 2010. pages 2

[2] N. UN, "Ban Ki-moon urges greater efforts to tackle silent crisis of safe water for all," 2007. pages 2

[3] School of Mathematics, Statistics and Computer Science, University of KwaZulu-Natal, Durban (South Africa), R. Chifurira, Department of Mathematical Statistics and Actuarial Science, University of the Free State, South Africa, D. Chikobvu, Unit of Applied Economics and Real Estate, Cape Peninsula University of Technology (South Africa), and D. Dubihlela, "Rainfall prediction for sustainable economic growth," *Environmental Economics*, vol. 7, pp. 120–129, Mar. 2017. pages 2

[4] E. Toth, A. Brath, and A. Montanari, "Comparison of short-term rainfall prediction models for real-time flood forecasting," *Journal of Hydrology*, vol. 239, pp. 132–147, Dec. 2000. pages 2

[5] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. Woo, "Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, (Cambridge, MA, USA), pp. 802–810, MIT Press, 2015. event-place: Montreal, Canada. pages 2, 3, 9, 23, 25, 26, 28, 38, 42, 43

[6] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. Adaptive computation and machine learning, Cambridge, Massachusetts: The MIT Press, 2016. pages 6, 11, 13, 32

[7] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *arXiv:1603.07285 [cs, stat]*, Mar. 2016. arXiv: 1603.07285. pages 7, 8

[8] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, May 2015. pages 9

[9] F. Monti, D. Boscaini, J. Masci, E. Rodol, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model CNNs," *arXiv:1611.08402 [cs]*, Nov. 2016. arXiv: 1611.08402. pages 8, 34, 36

[10] X. Shi, Z. Gao, L. Lausen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. Woo, "Deep Learning for Precipitation Nowcasting: A Benchmark and A New Model," *arXiv:1706.03458 [cs]*, June 2017. arXiv: 1706.03458. pages 9, 23, 24, 27, 30, 31, 37, 40, 42, 44, 45

[11] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: going beyond Euclidean data," *IEEE Signal Processing Magazine*, vol. 34, pp. 18–42, July 2017. arXiv: 1611.08097. pages 10, 34, 35

[12] N. Srivastava, E. Mansimov, and R. Salakhutdinov, "Unsupervised Learning of Video Representations using LSTMs," *arXiv:1502.04681 [cs]*, Feb. 2015. arXiv: 1502.04681. pages 12

[13] S. reference library, "EPSG:24500 Projection." pages 14

[14] P. K. Kitanidis, *Introduction to Geostatistics: Applications to Hydrogeology*. Cambridge: Cambridge University Press, 1997. pages 16

[15] K. University, "SPSS Tutorials: Pearson Correlation." pages 16

[16] F. Beck, A. Brdossy, J. Seidel, T. Mller, E. Fernandez Sanchis, and A. Hauser, "Statistical analysis of sub-daily precipitation extremes in Singapore," *Journal of Hydrology: Regional Studies*, vol. 3, pp. 337–358, Mar. 2015. pages 17, 19, 20

[17] B. Liao, J. Zhang, C. Wu, D. McIlwraith, T. Chen, S. Yang, Y. Guo, and F. Wu, "Deep Sequence Learning with Auxiliary Information for Traffic Prediction," *arXiv:1806.07380 [cs]*, June 2018. arXiv: 1806.07380. pages 29, 38

[18] C. Microsoft, "Neural Network Intelligence." pages 30, 31

[19] J. Shawe-Taylor and N. I. P. S. Foundation, eds., *Advances in neural information processing systems 24: 25th Annual Conference on Neural Information Processing Systems 2011 ; December 12 - 15, 2011, Granada, Spain*. Red Hook, NY: Curran, 2012. pages 31

[20] B. C. Csji, *Approximation with Artificial Neural Networks*. PhD thesis, Etvs Lornd University, Faculty of Sciences. pages 31

[21] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. pages 33

[22] C. Cangea, P. Velikovi, N. Jovanovi, T. Kipf, and P. Li, "Towards Sparse Hierarchical Graph Classifiers," *arXiv:1811.01287 [cs, stat]*, Nov. 2018. arXiv: 1811.01287. pages 34, 37

[23] G. Shunwang, *Geometric Deep Learning*. PhD thesis, IMPERIAL COLLEGE LONDON, DEPARTMENT OF COMPUTING. pages 34

[24] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive Representation Learning on Large Graphs," *arXiv:1706.02216 [cs, stat]*, June 2017. arXiv: 1706.02216. pages 36

[25] N. Choma, F. Monti, L. Gerhardt, T. Palczewski, Z. Ronaghi, Prabhat, W. Bhimji, M. M. Bronstein, S. R. Klein, and J. Bruna, "Graph Neural Networks for IceCube Signal Classification," *arXiv:1809.06166 [astro-ph, stat]*, Sept. 2018. arXiv: 1809.06166. pages 37

[26] R. J. Hogan, C. A. T. Ferro, I. T. Jolliffe, and D. B. Stephenson, "Equitability Revisited: Why the Equitable Threat Score Is Not Equitable," *Weather and Forecasting*, vol. 25, pp. 710–726, Apr. 2010. pages 43, 44