Demand prediction at GreenMobility using Spatial Neural Attention Models

Gustav Hatrtz (S174315) & Simon Jacobsen (s152655)
Technical University of Denmark, 02456 Deep Learning

Introduction

Our goal is to predict the demand of shared vehicles in the urban area of Copenhagen using data of where and when cars have previously booked.

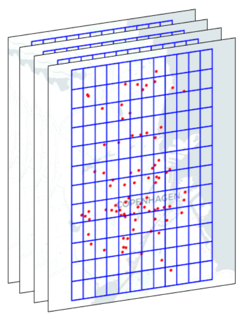
Information about adjacent areas could assist in improving the estimates of future demand across different neighbourhoods, hence allowing, in this case GreenMobility, a mobility sharing service to take action in order to meet customers behavioural patterns.

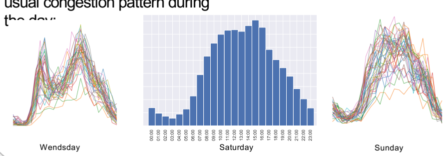
The **ConvLSTM** is an architecture which have been shown to give good result with in the domain of spatial and temporal dependent data. This is done by combining the two most popular model for these respective fields: The Convolutional Neural Network (CNN) & Long-Short Term Memory NN (LSTM).

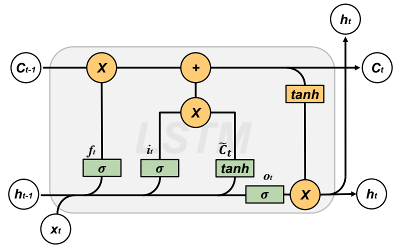
Data.

In order to model the problem the data have been discretized both spatial and temporal. The space have been divided into grid, so that the modelling is of this 'grid matrix'. The time have been sliced every half hour.

Looking at the data the demand showed several pattern relating the specific time of day, week etc. and that the demand followed usual congestion pattern during the day:







$$i_t = \sigma(W_{xi} * x_t + W_{hi} * h_{t-1} + W_{ci})$$ *Input gate*

$$f_t = \sigma(W_{xf} * x_t + W_{hf} * h_{t-1} + W_{cf})$$ *Forget gate*

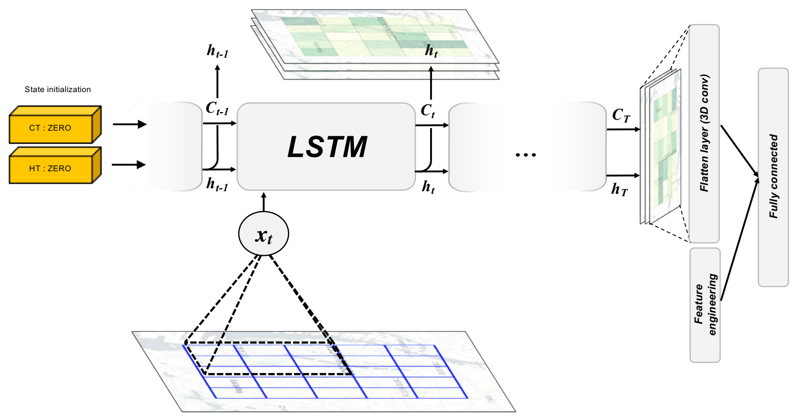
$$\tilde{C}_t = \tanh(W_{xc} * x_t + W_{hc} * h_{t-1} + b_c)$$ *Cell updates*

$$C_t = i_t * \tilde{C}_t + f_t * C_{t-1}$$ *Cell state*

$$o_t = \sigma(W_{xo} * X_t + W_{ho} * h_{t-1} + +o)$$ *Output gate*

$$h_t = o_t \circ \tanh(C_t)$$ *Hidden state*

'$*$' = the convolution operator
'$\circ$' = the Hadamard product



ConvLSTM model

```python
class VanillaConvLSTMFC_FE(nn.Module):
    def __init__(self, hidden_channels, in_chan, features_fc):
        super(VanillaConvLSTMFC_FE, self).__init__()

        self.convlstm1 = ConvLSTMCell(input_dim=in_chan,
                                  hidden_dim=hidden_channels, kernel_size=(3, 3), bias=True)
        self.convlstm2 = ConvLSTMCell(input_dim=hidden_channels,
                                  hidden_dim=hidden_channels, kernel_size=(3, 3), bias=True)
        self.output_CNN = nn.Conv3d(in_channels=hidden_channels,
                                  out_channels=1, kernel_size=(1, 3, 3), padding=(0, 1, 1))

        self.FC_CNN = nn.Linear(14*14+features_fc,14*14)
        self.activation = nn.ReLU()
        self.FC_OUT = nn.Linear(14*14,14*14)

    def convSEQ(self, x, seq_len, h_t, c_t, h_t2, c_t2):
        outputs = []

        for t in range(seq_len):
            h_t, c_t = self.convlstm1(input_tensor=x[:, t, :, :],
                                                  cur_state=[h_t, c_t])
            h_t2, c_t2 = self.convlstm2(input_tensor=h_t,
                                                  cur_state=[h_t2, c_t2])
        outputs.append(h_t2)
        outputs = torch.stack(outputs, 1)
        outputs = outputs.permute(0, 2, 1, 3, 4)

        outputs = self.output_CNN(outputs)
        return outputs

    def forward(self, x, y, hidden_state=None):
        b, seq_len, _, h, w = x.size()

        h_t, c_t = self.convlstm1.init_hidden(batch_size=b, image_size=(h, w))
        h_t2, c_t2 = self.convlstm2.init_hidden(batch_size=b, image_size=(h, w))

        outputs = self.convlst_sec(x, seq_len, future_seq, h_t, c_t, h_t2, c_t2)

        feature_cat_output = torch.cat((outputs.view(-1),y.view(-1)))

        outputs = self.FC_CNN(feature_cat_output)
        outputs = self.activation(outputs)
        outputs = self.FC_OUT(outputs)

        return outputs.view(1,1,1,h,w)
```

Conclusion & Further work:

As soon as the model has proven viable within our specific domain we will start to adjust, tune and add the following:

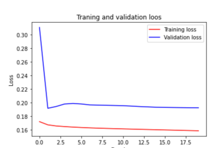
- Optimizer: Though Adam seem to be best from literature)
- Activation function: Yet ReLU seems to be the most frequently used.
- Poisson Loss: The Poisson distribution is often deployable in domains with continuous time and discrete events.
- Regularization: Try adding a weight decay to mitigate overfitting and a dropout which can give the robustness.
- Batch Normalization: To combat the ongoing change in the feature distribution during training know as the *internal covariate shift*..

Hyperparameters & Models

Model	Seq. size	Slice min	Grid size	Hidden size	Layers	Epochs
Vanilla LSTM –Baseline 1	12	30	14x14	30	2	20
Conv-LSTM [2]	12	30	14x14	30	2	10
Encoder-Forcaster Conv-LSTM – Feature engineering	24	30	6x6	64	4	30
Mean predictor - Baseline 2	1	30	14x14	x	x	x

Performance*

Performance Moving MNIST	Performance our model: abs/rel
x	38.31 / 28.95
Testing	Mean
Testing	80.40 / 58.86
x	62.76 / 38.72



Accuracy Measures:
At each timestep
Absoulte:
$\sum_i^H \sum_j^W |t_{ij} - p_{ij}|$
Relative:
$\sum_i^H \sum_j^W |t_{ij} - p_{ij}| * \frac{1}{1+mean(t_{ij})}$

Predictions

References

[1] Kumar, Ashutosh. "Convcast: An embedded convolutional LSTM based architecture for precipitation nowcasting using satellite data" Martz 2020, https://doi.org/10.1371/journal.pone.0230114.
[2] S. Xingjian, Z. Chen, H. Wang, D. Yeung, W. Wong, and W. Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In Neural Information Processing Systems, 2015.
[3] Benjamin Sautermeister. "Deep Learning Approaches to Predict Future Frames in Videos" 17 October 2016 https://bsautermeister.de/research/docs/msc_thesis.pdf
[4] Wang, D.; Yang, Y.; Ning, S. DeepSTCL: A Deep Spatio-Temporal ConvLSTM for Travel Demand Prediction. 8–13 July 2018
[5] " Convolutional Neural Networks for Visual Recognition" CS 231N, Stanford University, 2020, https://cs231n.github.io/convolutional-networks/
[6]