Time series prediction & generation from disentangled latent factors: new opportunities for smart cities

Perrine Cribier-Delande[1,2], Raphael Puget[1], Camille Noûs[3] Vincent Guigue[2], Ludovic Denoyer[2]

***Abstract*— The acceleration of urbanisation has brought many new challenges to cities around the world. Application range is wide, from air pollution to public transportation modelling. The availability of data pertaining to these issues has been growing fast in the last years, offering many opportunities to tackle those applications with machine learning algorithms. We propose an elegant and general architecture that is able to provide state of the art forecasting in several different domains. Our idea is the following: for many time-series, a number of factors, that often relate to the context they were created in, can influence the observed values, such as day or location. In this paper, we present a machine learning model that learns to represent and disentangle such factors. Our contribution is to provide an approach that works at different scales: on a short term basis (30 minutes to few hours) our deep neural network architecture delivers competitive forecasting in a classical setting; at the day/week/month level, we show that we can generate relevant time series associated with unknown contexts. To the best of our knowledge, this ambitious application has not been investigated until now.**

I. Introduction

Environmental stakes and massive urbanisation have become in the last few years some of the major challenges large cities have to face. High pollution level, heavy traffic congestion, public transportation optimization, energy management are among the principal concerns of stackholders. Many cities have been forced to take policies to restrict personal car usage in order to keep their air clean of pollution, especially during some specific period of the year (when the air is particularly dry for example). One solution to make more efficient policies is to be able to predict passengers flows and pollution levels for instance. Data, associated with machine learning, can be the key to develop efficient predictive algorithm. Thus, transportation authorities are generalising automated fare collection (AFC) via smart cards. Many cities have elaborated an open data strategy and encourage research in data sciences. These new tools can allow policy makers reduce pollution peaks, traffic jams while optimising public transportation in the meantime.

Modelling users behaviours and training predictive algorithms from various data sources require to combine different research fields: representation learning to extract relevant aspects associated with contextual factors [1] and time series analysis to deliver proper forecastings. Statistical methods such as ARIMA and Box-Jenkins [2] have been used with great success in time series forecasting applications. However, they have shown some weaknesses when tackling large amounts of non-linear data. In these cases, deep learning is increasingly being used. As they are specifically designed to handle sequences, Recurrent Neural Networks (RNN), and especially LSTM and GRU, have been used extensively to model time series [3], [4]. Their ability to keep both past trend and recent changes in memory allows them to be particularly suited to forecast series at multiple scales. In parallel, deep learning is offering powerful tools to learn representations of discrete concepts such as word in NLP [5], users in recommender systems [6] and contextual factors of time series [7].

Classical approaches in time series forecasting can roughly be split into three main categories: short-term predictors, often relying on historical tools [2], mid-term predictors exploiting deep learning to extract automatically robust features from a large period of time [8] and, finally, long-term predictors that correspond mainly to seasonality [9], [10]. Assuming this distribution, our contribution represents a significant technological break: not only to we provide state of the art forecasting at short and mid-term scales, but we also redefine the task corresponding to long-term prediction by turning it into a generative issue. Indeed, we propose a deep learning architecture that is able to disentangle different latent factors from a time series, for instance the location and the day type (week day, Sunday, ...). Then our approach is able to generate a new signal from any combination of factors. As a consequence, we are able to generate the time series corresponding to a Monday at place A even if place A has never been observed on Monday in the past: we just need to have learnt what the representation of location A is and what the representation of a Monday is.

From the technical point of view, our contribution is a deep learning model based on an encoder-decoder architecture. The encoder component learns to disentangle the salient features from time series that gave rise to them. In our examples, there are two factors: time and location. However, our approach can intrinsically deal with as many factors as requested by an application. The decoder takes as input a representation for each factor and, optionally, the past values associated with the series to forecast. The strength of our model is that it can both be used for predicting the future but also to generate a time series corresponding to a specific couple (location, day) even if it has not been seen in the dataset. Thus, it is a powerful tool to deal with the missing value issue. In an extreme setting, our approach may even able to predict time series associated to a new factor,

[1]Renault, DEA-IR, Technocentre, 1 avenue du Golf 78084 Guyancourt , France
[2]MLIA, Sorbonne Université, CNRS, LIP6, F-75005 Paris, France
[3]Laboratoire Cogitamus, http://www.cogitamus.fr/, France

observed only once.

After giving all details associated with our architecture, we provide some experimental results demonstrating our ability to perform state of the art forecasting as well as time series generation in various contexts.

II. Notations and tasks

Our dataset $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ consists in a set of n univariate time series. In this article, we focus on time series of fixed length T, even if our recurrent architecture works on signals of arbitrary length without modifications. In the training set, each signal $\mathbf{x}_i \in \mathbb{R}^T$ is associated to a couple of discrete spatio-temporal labels (s,t) ($s \in [1..N]$ and $t \in [1..M]$). In the following, we regularly denote $\mathbf{x}_{s,t}$ the series associated with this couple. The location s takes on different meanings depending on the dataset. On the contrary, for the temporal context t all datasets used in this article operate on daily time-series described on a hourly basis (leading to a constant value of T in every domain). The observed series are indexed through a mask m such that $m_{s,t} = 1$ if $\mathbf{x}_{s,t}$ is observed (and is in the training set), and $m_{s,t} = 0$ if $\mathbf{x}_{s,t}$ is not observed[1].

We tackle three distinct tasks in this article. The first one is the generation of a time series matching specific spatial and temporal context. These contexts are present in the training set but not seen together in one example. As in recommender systems, we seek to estimate missing values from spatial and temporal contexts modelled by embeddings; but in our setting, values are time-series. Our second task corresponds to an extreme setting: generating a time series matching a spatial or temporal context not present in the training set. Continuing the analogy with recommender systems, it would correspond to the cold start. These two tasks are represented in figure 1. The third task is the classical forecasting setting. In our architecture, it is a specific use of the generative process combined with a recurrent neural network (RNN). We start the generation process from a general context and we refine the forecasting by adding ground truth values from the past at the entry of the RNN.

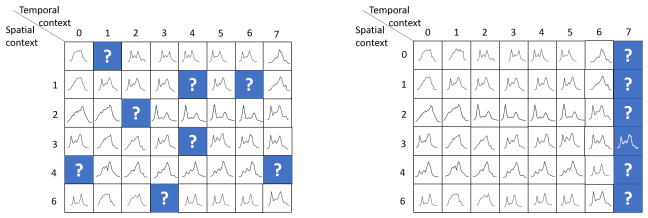

Fig. 1. A representation of our two first tasks on a dataset represented as matrices. White background means $m_{i,j} = 1$. Dark blue is $m_{i,j} = 0$ [Left] The first task, matrix completion [Right] The second task, addition of new contexts

A. Transductive setting: a matrix completion approach

Our first task is to generate a time series for any (s,t) couple. We can see our dataset $\mathcal{X}$ as an incomplete matrix of spatial context by temporal context, each cell of the matrix represents one time series of the dataset (Fig. 1, top). The cell on the s^{th} row and t^{th} columns corresponds to $\mathbf{x}_{s,t}$. Our goal is to generate this missing data, similarly to what is usually done in matrix factorisation. This can be useful when this matrix is sparse because data collection is expensive.

We introduce two matrices of learned embeddings denoted: $Z_s = \{\mathbf{s}_1, \dots, \mathbf{s}_N\}$ and $Z_t = \{\mathbf{t}_1, \dots, \mathbf{t}_M\}$ with $\mathbf{s}$ and $\mathbf{t}$ being Z-dimensional vectors encoding respectively space and time contexts[2]. To simplify notation in the following, we denote $\mathbf{s}$ the embedding related to location s (instead of $\mathbf{s}_s$).

The embeddings are fed to a decoder d_θ (parametrized by a set of parameters θ) that learns to reconstruct the series corresponding to these context. We choose a classical Mean Square Error (MSE) metrics to train a decoding function d_θ, leading to the following cost over the n labelled training samples:

$$L_{MSE} = \sum_{i=1}^{n} \|\mathbf{x}_i - \hat{\mathbf{x}}_\mathbf{i}\|^2, \text{with: } \hat{\mathbf{x}}_\mathbf{i} = d_\theta(\mathbf{s}, \mathbf{t}) \qquad (1)$$

where the i^{th} sample is associated to the couple of factors (s,t). This loss allows us to learn the two matrices of embeddings and the decoder jointly (end-to-end).

B. Inductive setting: generating time-series from unseen factors

In this second setting, $\mathbf{s}$ and $\mathbf{t}$ are no longer directly optimised, they are encoded on the fly using an encoding function e_γ (parametrized by a set of parameters γ). In order to generate all time series corresponding to a new location s' (respectively to a new day t') from a single observation $\mathbf{x}_{s',t} \in \mathbb{R}^T$, we propose:

1) to compute the embedding $(\mathbf{s}', \mathbf{t}) = e_\gamma(\mathbf{x}_{s',t})$
2) to generate all series associated to s':

$$\forall t^\dagger \in \{1, \dots, M\}, \hat{\mathbf{x}}_{s',t^\dagger} = d_\theta(\mathbf{s}', \mathbf{t}^\dagger)$$

Our cost function is still : $L_{MSE} = \sum_{i=1}^{n} \|\mathbf{x}_i - \hat{\mathbf{x}}_\mathbf{i}\|^2$ however this time : $\hat{\mathbf{x}}_\mathbf{i} = d_\theta(\mathbf{s}, \mathbf{t}) = d_\theta(e_\gamma(\mathbf{x}_{s,\cdot}), e_\gamma(\mathbf{x}_{\cdot,t}))$ With respect to the previous example, we point out that s' has never been seen during the training step. The embedding $\mathbf{s}'$ is obtained in the inference step by using the trained encoder on a new signal.

C. Classical forecasting setting

For both tasks presented above, we assumed that no data on the series to forecast are available. We now investigate the more classical forecasting issue where the aim is to predict future values with respect to past observations. This problem corresponds to a specific use of the previous approach: we still rely on $d_\theta(\mathbf{s}, \mathbf{t})$ to tackle the prediction task associated with context factors (s,t), but we now assume that the decoder d relies on a recurrent architecture. As explained in detailed in the next section, such an architecture combines an

[1]Note that in the experimental section, this split will also include a validation set for hyper-parameters tuning.

[2]Please note that embedding corresponding to each factor may have different dimensions. We also remind that our architecture can deal with as many factors as requested by the application.

input value with a latent representation encoding the whole past of the sequence to generate a new latent representation from which a prediction can be made. Whereas our generative approaches used the last prediction to feed the network and obtain the next value estimation, the forecasting architecture simply feed the network with ground truth values from the past.

The training step (and the loss functions) are the same as previously. The only difference resides in the way to exploit the recurrent decoder d. Using past ground truth values transforms the generative architecture into an efficient forecasting one.

III. Model & Learning procedure

As explained above, our general architecture can rely on different neural network implementations. The properties associated with a standard multilayer perceptron or a recurrent model are different: we give some details about the formulations to provide a better understanding of these properties.

From a general point of view, the architecture is made of a decoder that generates sequences from context factors and, for the inductive approach only, an encoder that extracts factor representations from a given time series. In theory, we can combine any neural network for the encoder with any other one for the decoder. In practice, we only consider homogeneous architectures.

The subsection on MLP describes this particular approach but it also gives the tricks associated with the inference procedure. Those tricks are also used with the others architectures. The recurrent architecture is the only one able to tackle the forecasting task; thus, the forecasting procedure is given in this subsection.

A. Multi-Layer Perceptron

The most classical neural architecture is the MLP. The idea is to map the input into a latent space and then to map the latent representation $\mathbf{h} \in \mathbb{R}^d$ to the output domain. Mapping operations are often non linear: they combine an affine transformation and a non-linear activation function.

a) Decoder: First, we concatenate the couple of context embeddings $\mathbf{s}, \mathbf{t}$ at the input: $[\mathbf{s}, \mathbf{t}] \in \mathbb{R}^{2Z}$. The intermediate representation is obtained as follow: $\mathbf{h} = g_1(W_1 \cdot [\mathbf{s}, \mathbf{t}])$ with $W_1 \in \mathbb{R}^{2Z \times d}$ and g_1 an activation (e.g. a sigmoid or RELU function). Estimating the time series from the latent representation is done by: $\hat{\mathbf{x}}_{s,t} = g_2(W_2 \cdot \mathbf{h})$, with $W_2 \in \mathbb{R}^{d \times T}$. As a consequence, we can only deal with fixed length time series (T) and we are not able to implement the forecasting framework with this approach. To sum up this formulation and bridge with previous notations, in this case:

$$\hat{\mathbf{x}}_{s,t} = d_\theta(\mathbf{s}, \mathbf{t}) = g_2(W_2 \cdot g_1(W_1 \cdot [\mathbf{s}, \mathbf{t}])), \quad \hat{\mathbf{x}}_{s,t} \in \mathbb{R}^T \quad (2)$$

The training procedure relies on the back-propagation algorithm which minimises the loss (1). This procedure optimises both the MLP parameters ($\theta = (W_1, W_2)$) and the embeddings (Z_s, Z_t). The decoding procedure is illustrated in Fig. 2.

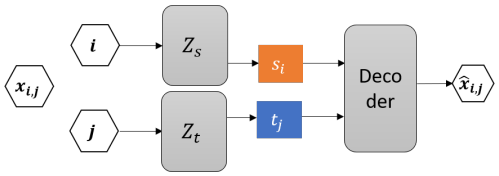

Fig. 2. Transductive model relying on discrete embeddings to generate a time-series.

The nature of the activation function, the number of layers, and their dimension was optimised through the validation set and depends on the dataset.

b) Encoder: To tackle the inductive setting and being able to predict time series associated with a context never seen during the training step, we need to optimise an encoder function that can extract representations $\mathbf{s}, \mathbf{t}$ from a signal $\mathbf{x}_{s,t}$. We rely on the same MLP architecture as above, except that we have to define an encoder per factor. For the spatial factor, the encoder is:

$$\mathbf{s} = e_{\gamma,s}(\mathbf{x}_{s,t}) = g_3(W_{3,s} \cdot g_4(W_{4,s} \cdot \mathbf{x}_{s,t})), \quad \mathbf{s} \in \mathbb{R}^Z \quad (3)$$

The symmetric function $e_{\gamma,t}$ relying on $(W_{3,t}, W_{4,t})$ is defined and optimised in parallel.

Once again, the MLP consider the time series as a block and we are limited to fixed length sequences. The learning procedure is end-to-end, namely starting from a signal, we optimise our capacity to reconstruct it by encoding and decoding it. In this process, the encoding parameters $\gamma = (W_3, W_4)$ and the decoding parameters θ are optimised at the same time.

c) Enforcing disentanglement: Taking a closer look at this architecture reveals a clear weakness: in the auto-encoding procedure, the MLPs are going to use $\mathbf{s}$ and $\mathbf{t}$ indistinctly to encode the information contained in the signal. Even if the recurrent patterns associated to each factor will be separated, we are probably going to mix up lower energy phenomena.

During the learning step, the paradigm used to enforce disentanglement is described in Fig. 3. The idea is to encode a pair of sequences: one containing the location factor s and the other the time factor t. Then, we are going to reconstruct a third time series corresponding to the pair (s, t). This particular procedure enables us to ensure a better disentangling of the context factors. To sum up the learning step, we perform a stochastic gradient descent based on the sampling of sequence triplets:

1) $\mathbf{x}_{s,t'}$ and $\mathbf{x}_{s',t}$ are encoded using e_γ, leading to four representations $\mathbf{s}, \mathbf{s}', \mathbf{t}, \mathbf{t}'$.
2) Only $\mathbf{s}, \mathbf{t}$ are used in d_θ to build $\hat{\mathbf{x}}_{s,t}$.
3) The loss $\mathcal{L}_{s,t} = \|\mathbf{x}_{s,t} - \hat{\mathbf{x}}_{s,t}\|^2$ enables us to optimise $\gamma, \theta, \mathbf{s}$ and $\mathbf{t}$.

d) Inference on unseen factors: One important thing to notice is that there is no guarantee as to the unity of the representation $\mathbf{s}$. Indeed, it is not mandatory that our function e_γ would lead to the same representation for each example corresponding to a context. That is why, during inference, we must choose how to represent the "known" context. For

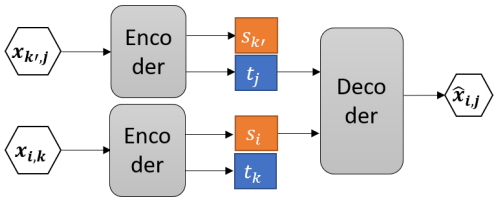

Fig. 3. Inductive model learning a continuous encoder & enforcing disentanglement between context factors.

example, if the new example is $\mathbf{x}_{s^\star,t}$ where $s^\star$ is unknown, we aim at predicting the behaviour of this location for every existing timestamp existing in the learning set. In particular, we are going to estimate $\hat{\mathbf{x}}_{s^\star,t_1}$... But how are we supposed to choose the latent representation $\mathbf{t}_1$ associated with $\mathbf{x}_{s_1,t_1}$ or $\mathbf{x}_{s_2,t_1}$?

We investigate two options to tackle this issue. For sake of clarity, we focus on a new location $s^\star$ and on a particular timestamp t_1, the aim being to estimate $\hat{\mathbf{x}}_{s^\star,t_1}$. Obviously, we can also work with new timestamps.

- **Embedding averaging:** We wish to obtain a unique representation $\mathbf{t}_1$. Given $\mathcal{T}$ the set of sequences corresponding to the targeted factor t_1. We compute the average of the encoded embeddings: $\overline{\mathbf{t}}_\mathbf{1} = \frac{1}{|\mathcal{T}|}\sum_{i\in\mathcal{T}} e_{\gamma,t}(\mathbf{x}_i)$ and we estimate a new series as: $\hat{\mathbf{x}}_{s^\star,t_1} = d_\theta(e_{\gamma,s}(\mathbf{x}_{s^\star,t}), \overline{\mathbf{t}}_\mathbf{1})$
- **Series averaging:** With this method, we compute an estimated time series associated with every factors extracted from the training set. The resulting time-series is computed by averaging all theses series: $\hat{\mathbf{x}}_{s^\star,t_1} = \frac{1}{|\mathcal{T}|}\sum_{i\in\mathcal{T}} d_\theta(e_{\gamma,s}(\mathbf{x}_{s^\star,t}), e_{\gamma,t}(\mathbf{x}_i))$

B. Convolutional Neural Network (CNN)

CNN architectures learn d filters f_i to extract local features from sequences. The weight of the filter f_i is denoted $\mathbf{w_i}$. Each filter is convoluted with the input, resulting on d feature maps. An activation function g is then applied to these feature maps. So for an input $\mathbf{x}$, the output of one CNN layer can be written as: $\mathbf{h} = [\forall i\ g(\mathbf{x} \star \mathbf{w_i})]$ where $\star$ is the operator for 1-dimension convolution.

The resulting feature maps can then be used as input for the next layer. Our encoding framework relies on a single layer in this article.

It appears that no pooling operations are performed on the temporal: as a consequence intermediate representations have a proportional dimension to the input. As for the MLP, we are limited to fixed length sequences with this architecture and we can not tackle the forecasting task.

Decoding operations are symmetric to the encoding ones: we learn filters that associate each coefficient with a local partial reconstruction of the sequence. This process enables us to reconstruct an estimate signal from $[\mathbf{s}, \mathbf{t}]$.

C. Recurrent Neural Networks (RNN) & Forecasting

To be able to perform forecasting step by step, we used a RNN architecture. RNN are called recurrent because of their ability to remember the past by maintaining a continuous internal state. This internal state is modified with each new input to add this new information while keeping memory of the past one. A RNN cell takes as input the last internal state $\mathbf{h}_{a-1}$[3] and the new input x_a, and outputs the new internal state $\mathbf{h}_a$. Depending on the type of RNN used, $\mathbf{h}_a$ can be computed using several functions. The simplest one is: $\mathbf{h}_a = g(W \cdot [\mathbf{h}_{a-1}, x_a])$ where g is the activation function and W are the parameters to learn.

a) Decoder & Forecasting: The RNN architecture is intrinsically a forecasting operator. Parameters are learnt so as to predict the next value of the series: $x_{a+1} = g_2(W_2 \cdot \mathbf{h_a})$. We denote f_{RNN} the global function outputting both the next value prediction and the next state. In this article, we aim at predicting future values based on past values and context factor representations: at each time step, we feed the RNN cell with the factor representations: $\mathbf{h}_a, \hat{x_{a+1}} = f_{RNN}(\mathbf{h}_{a-1}, [\mathbf{s}, \mathbf{t}, x_a]), \quad \mathbf{h}_0 = 0$ This decoder also enables us to learn the contextual profiles in the transductive setting.

b) Decoder & Sequence generation: RNN architecture can also be used for the complete sequence generation, as previous proposals. In this case, we modify the RNN cell so that it takes only the context and the previous state: $\mathbf{h_a}, \hat{x_{a+1}} = f_{RNN}(\mathbf{h_{a-1}}, [\mathbf{s}, \mathbf{t}])$ and we use it recursively, the initial state still being 0.

c) Encoder: When used as encoder in our model, the RNN takes as input the input series step by step and it's final internal state gives us the latent representation of our context. The last internal state $\mathbf{h_T}$ is fed to a regular MLP to give us the representation of the context. Once again, two separated MLP are required to encode two factors. As for the previous architecture, the RNN encoder is learnt in an end-to-end manner.

IV. Experiment

We tested our models on three different datasets. Each is comprised of times series labelled with spatial and temporal contexts. The temporal context matches days. The measure are taken hourly giving us 24-hour long time series.

A. Datasets

a) Smart card dataset (STIF): The smart card dataset, denoted STIF, was collected by Ile-de-France Mobilites and counts more than 256 millions of smart card validations' logs on the Parisian subway network (299 stations). The raw data contains for each log the location, hour and day of the validation. We aggregated these logs by hour for each station to get 24h long time series for each day of the period. It has been collected during the last 3 months of 2015 (October, November, December), which corresponds to 91 days. Three very specific days were removed from the dataset. Thus, it counts 88 temporal and 299 spatial contexts.

[3]The notation t is already used for the date. We use a to denote the timestamp within a day. In this article $a \in [1, T]$.

b) Energy Consumption: This kaggle dataset consists of hourly consumption of energy. It comes from PJM Interconnection LLC, a regional transmission organization in the US that supplies energy to large portion of the North East of the United States. The energy grid is split over few regions, each having there own reported consumption. We consider each region as a spatial context. The data spans from the 2nd of June 2013 to the 2nd August 2018. Few days had recording errors and were dismissed. This dataset has 1877 temporal and 10 spatial contexts.

c) Air quality : This dataset is also from Kagglewas collected and opened by the City of Madrid This dataset consists of hourly measures of a pollutant in the air of Madrid. Measurement stations measured the NO_2 and NO level in $\mu g/m^3$ in the air. The measurement cover the period from the 2^{nd} of January 2011 to the 30^{th} of April 2018. Some days had missing values and were dismissed. These two datasets have 2669 possible temporal contexts and 24 possible spatial contexts.

B. Baselines

We compare our models to different baselines.

- The first baseline (BL avg) is very general and can be used for the three tasks: it predicts $\hat{\mathbf{x}}_{s,t}$ by averaging the observed time series sharing a common context. $\hat{\mathbf{x}}_{s,t} = \frac{1}{|\mathcal{T}|}\sum_{i\in\mathcal{T}}\mathbf{x}_i$ where the subset $\mathcal{T}$ gathers all sequences from the training set associated with a location s or a temporal context t. Thus the prediction is deterministic for a given context. However, we study stable applications where it makes sense.
- The second baseline (BL NN) can only be used for the second task (adding new contexts) where we are given a series corresponding to a new location $\mathbf{x}_{s^\star,k}$ and we aim at predicting all other $\{\mathbf{x}_{s^\star,\cdot}\}$.It is based on the idea of nearest neighbours (NN). The principle is to find the p series in $\{\mathbf{x}_{\cdot,k}\}$ that are the closest to the target $\mathbf{x}_{s^\star,k}$ (i.e. the other locations that behave as $s^\star$ on day k). Let's define $\mathcal{T}$ the set of locations of these series. The prediction will be an aggregation of those close locations at each time stamp: $\hat{\mathbf{x}}_{s^\star,k} = \frac{1}{p}\sum_{i\in\mathcal{T}}\mathbf{x}_{i,k}$
- Our third baseline (Oracle NN) is an oracle since it requires to access the ground truth. It is also based on the idea of nearest neighbours. In this case, $\mathbf{x}_{\hat{s},\hat{t}}$ is simply estimated by averaging its p nearest neighbours in the training set. It can't be used on the second task, because it requires access to other example of the same context.

C. Results

For each of the datasets described above, we tested four models: MLP, CNN, RNN_{gen} and RNN_{pred} (this latter relying on the exploitation of ground truth values from the past) and three baselines. All datasets are made of 24h hourly time series taken at different locations, measuring a single quantity. The datasets were normalized between 0 and 1 via min-max scaling. Because they were inherently multi-scaled (for example some subway stations are much bigger than others and the most frequented part of town have more pollutant than others), this normalisation was done location by location.

a) Task 1: sequence generation / matrix completion: The Table I shows our results for our first task (matrix completion) with the MSE between the predicted series and the ground truth. As expected, the oracle baseline achieves near perfect results. Indeed, we work on data presenting strong regularities: for any date, it exists a very similar day in the training set. The transductive settings (using embeddings matrices) outperforms the inductive settings because it encodes the exact information of the context, whereas the inductive settings does not have access to it. For instance, the transductive approach knows that it has to predict a March 8^{th} (and it has the corresponding representation from the learning step) whereas the inductive approach only knows several sequences from the same date and encodes the representation online during the inference step.

Our results are very interesting on public transportation affluence and energy consumption prediction. For NO and NO_2, the low number of spatial contexts (24) combined with a strong regularities makes the last baseline particularly efficient. It is important to note that this last baseline has more information than the inductive setting (as explained above).

b) Task 2: sequence generation / new context: Task 2, consisting in making prediction for all days from a single sequence associated with a new location (or respectively extrapolating predictions for all location from one sequence corresponding to a new day) is challenging but really valuable from an industrial point of view. It could cut down on a lot of data collection costs.

The Table II shows our results on this task (same MSE as the previous one). Our results on the new location are just as the same as our baselines for the datasets on Energy and Air quality which can be explained by the low number of location in the datasets (10 and 24). However, our proposal are very efficient on public transportation and, on all dataset, for the prediction of what is going to happen on a new day. This latter series of experiments is very promising for a wide range of applications: in particular, it could be a way to provide efficient sales forecasting at a fine grain scale (shop and/or product).

c) Task 3: Time series forecasting: We finish by the most classical task. As we aim at predicting only the next time step (and not the whole day), our forecasting are more accurate. The Table III shows our results on the third task (same MSE as the previous ones).

Our proposals work particularly well on this task. Not only do we overcome average baselines (that provide long term predictions) but we also clearly exceed the classical baseline consisting in predicting the last observed value and even the oracle baseline.

V. Conclusion

We provide a new elegant framework and demonstrate that it is able to provide state of the art forecastings on a

		STIF		Energy		NO		NO2	
Encoder	Decoder	Series avg	Emb. avg	Series avg	Emb. avg	Series avg	Emb. avg	Series avg	Emb. avg
Oracle NN (p)		0.001 (5)		0.00008 (4)		0.0007 (5)		0.0031 (5)	
BL avg (temporal)		0.0179		0.006		0.0107		0.0215	
BL avg (spatial)		0.0121		0.0089		0.0045		0.0098	
Emb	MLP	0.0028		0.002		0.0043		0.0098	
Emb	CNN	0.0053		0.0048		0.0046		0.0101	
Emb	RNN	0.0016		0.0015		0.0041		0.0102	
MLP	MLP	0.0084	0.0084	0.0046	0.0047	0.0052	0.0052	0.0106	0.0105
CNN	CNN	0.0073	0.0073	0.0059	0.0059	0.0058	0.0057	0.0104	0.0105
RNN	RNN	0.0026	0.0025	0.0036	0.0034	0.0050	0.0051	0.0107	0.0105

TABLE I

MSE ERRORS FOR TASK 1: SEQUENCE GENERATION WITH THE MATRIX COMPLETION SETTING. FROM TOP TO BOTTOM: BASELINES, TRANSDUCTIVE MODELLING & INDUCTIVE MODELLING (CONTEXT ENCODED ON THE FLY).

			STIF		Energy		NO		NO2	
	Encoder	Decoder	Series avg	Emb. avg	Series avg	Emb. avg	Series avg	Emb. avg	Series avg	Emb. avg
New loc.	BL NN (p)		0.0123 (150)		0.0081 (3)		0.0049 (5)		0.0107 (5)	
	BL avg.		0.0125		0.0087		0.0052		0.0112	
	MLP	MLP	0.0059	0.0059	0.0118	0.0118	0.0058	0.0057	0.0107	0.0107
	CNN	CNN	0.0075	0.0076	0.0099	0.0102	0.0062	0.0064	0.0101	0.0102
	RNN	RNN	0.0049	0.0049	0.0107	0.0108	0.0062	0.0065	0.0106	0.0108
New day	BL NN		0.0143 (7)		0.0093 (900)		0.0099 (70)		0.0207 (60)	
	BL avg.		0.0145		0.0089		0.0103		0.0210	
	MLP	MLP	0.0063	0.0061	0.0076	0.0076	0.0062	0.0062	0.0133	0.0133
	CNN	CNN	0.0061	0.0059	0.0065	0.0063	0.0071	0.0071	0.0129	0.0128
	RNN	RNN	0.0041	0.0040	0.0047	0.0046	0.0063	0.0063	0.0139	0.0139

TABLE II

MSE ERRORS FOR TASK 2: NEW LOCATIONS ABOVE, NEW DAYS BELOW.

			STIF		Energy		NO		NO2	
	Encoder	Decoder	Series avg	Emb. avg	Series avg	Emb. avg	Series avg	Emb. avg	Series avg	Emb. avg
	Baseline forecasting		0.0184		0.0005		0.0029		0.0049	
	Baseline NN (p)		0.001 (5)		0.00008 (4)		0.0007 (5)		0.0031 (5)	
Known contexts	Emb	RNN_pred	0.0001		0.0003		0.0001		0.0001	
	RNN	RNN_pred	0.0002	0.0002	0.00008	0.00008	0.0001	0.0001	0.0004	0.0002
New day	RNN	RNN_pred	0.0003	0.0003	0.0001	0.0001	0.0002	0.0002	0.0002	0.0002
New loc.	RNN	RNN_pred	0.0001	0.0001	0.00003	0.00003	0.0005	0.0005	0.0001	0.0001

TABLE III

MSE ERRORS FOR TASK 3: FORECASTING ON A SHORT TERM BASIS.

short term basis as well as to tackle new tasks that could be valuable in several industrial applications.

ACKNOWLEDGMENT

This work is partially supported by the European Union's Horizon 2020 Research and Innovation Program under grant agreement No 780754, "Track & Know".

REFERENCES

[1] Y. Bengio, "Deep learning of representations: Looking forward," *CoRR*, vol. abs/1305.0445, 2013. [Online]. Available: http://arxiv.org/abs/1305.0445

[2] G. E. Box and G. M. , "Some recent advances in forecasting and control," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 17, no. 2, pp. 91–109, 1968.

[3] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735

[4] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, vol. abs/1406.1078, 2014. [Online]. Available: http://arxiv.org/abs/1406.1078

[5] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013.

[6] F. Strub, R. Gaudel, and J. Mary, "Hybrid recommender system based on autoencoders," *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems - DLRS 2016*, 2016. [Online]. Available: http://dx.doi.org/10.1145/2988450.2988456

[7] P. Cribier-Delande, R. Puget, V. Guigue, and L. Denoyer, "Time series prediction using disentangled latent factors," *ESANN*, 2020.

[8] J. Franceschi, A. Dieuleveut, and M. Jaggi, "Unsupervised scalable representation learning for multivariate time series," *CoRR*, vol. abs/1901.10738, 2019. [Online]. Available: http://arxiv.org/abs/1901.10738

[9] S. Makridakis and S. Wheelwright, *Forecasting: Methods and Applications*. John Wiley & Sons Ltd., New York, 1978.

[10] L. B. Taylor SJ, "Forecasting at scale," 2017.