HAL
open science

Deep Learning applied to Road Traffic Speed forecasting

Thomas Epelbaum, Fabrice Gamboa, Jean-Michel Loubes, Jessica Martin

To cite this version:

Thomas Epelbaum, Fabrice Gamboa, Jean-Michel Loubes, Jessica Martin. Deep Learning applied to Road Traffic Speed forecasting. 2017. hal-01598905

HAL Id: hal-01598905

https://hal.science/hal-01598905v1

Preprint submitted on 30 Sep 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Deep Learning applied to Road Traffic Speed forecasting

Thomas Epelbaum[a],Fabrice Gamboa[b], Jean-Michel Loubes[b] and Jessica Martin[c1]

[a]*Mediamobile*, [b]*Institut de Mathématiques de Toulouse* and [c]*INSA*

October 2, 2017

[1]Corresponding author: thomas.epelbaum@mediamobile.com

Abstract

In this paper, we propose deep learning architectures (FNN, CNN and LSTM) to forecast a regression model for time dependent data. These algorithm's are designed to handle Floating Car Data (FCD) historic speeds to predict road traffic data. For this we aggregate the speeds into the network inputs in an innovative way. We compare the RMSE thus obtained with the results of a simpler physical model, and show that the latter achieves better RMSE accuracy. We also propose a new indicator, which evaluates the algorithms improvement when compared to a benchmark prediction. We conclude by questionning the interest of using deep learning methods for this specific regression task.

AMS subject classifications:

Keywords: Deep Learning, Feedforward Neural Network, Convolutional Neural Network, Recurrent Neural-Network, Long Short Term Memory, time series.

Contents

1 Road Traffic Speed forecasting: Modelization, Results **3**
1.1 Introduction . . . 3
1.2 Data . . . 4
1.2.1 Speeds from FCD measures . . . 4
1.2.2 Input speeds . . . 5
1.3 Model evaluation . . . 7
1.3.1 RMSE and Q-score . . . 7
1.4 The models . . . 9
1.4.1 Common properties of the Networks . . . 9
1.4.2 Specificities of each Network . . . 10
1.4.3 Recurrent Neural Networks . . . 11
1.5 Results . . . 12
1.5.1 Full input . . . 12
1.5.2 Reduced input . . . 15
1.5.3 Importance of the Q-score . . . 17
1.6 Conclusion . . . 18

2 Deep Learning: supplementary and technical description **19**
2.1 Feedforward Neural Networks . . . 19
2.1.1 Some notations and definitions . . . 19
2.1.2 FNN architecture . . . 20
2.1.3 Weight averaging . . . 21
2.1.4 Activation function . . . 21
2.1.5 Input layer . . . 22
2.1.6 Fully connected layer . . . 22
2.1.7 Output layer . . . 22
2.1.8 Loss function . . . 22
2.1.9 Batch Normalization . . . 23
2.1.10 Architecture considered in practice . . . 24
2.2 Convolutional Neural Networks . . . 25
2.2.1 CNN new specific notations and definitions . . . 25
2.2.2 CNN architecture . . . 26
2.2.3 Input layers . . . 26
2.2.4 Convolutional layers . . . 26
2.2.5 Pooling layers . . . 27
2.2.6 Architecture considered in practice . . . 27
2.3 Recurrent Neural Networks - Long Short Term Memory . . . 28
2.3.1 RNN new specific notations and definitions . . . 28

2.3.2 RNN architecture 29
2.3.3 RNN hidden layer 29
2.3.4 LSTM hidden layer 29
2.3.5 RNN-LSTM Input Layer 31
2.3.6 RNN-LSTM Output Layer 31
2.3.7 Change to the Loss function 31
2.3.8 Architecture considered in practice 31

Chapter 1

Road Traffic Speed forecasting: Modelization, Results

1.1 Introduction

Traffic congestion is one of the major downsides of our ever-growing cities. The inconvenience for individuals stuck in traffic jams can sometimes be counted in hours per day, and weeks per year. In this context, a road traffic speed forecasting algorithm could have a highly beneficial impact: it could feed a Dynamic Routing System (DRS), and allow one to anticipate the formation and the resorption of congestion. This could lead to intelligent recommandation to drivers, and point towards public measures for shifted departures to and from work (monetary incentives to companies/individuals...)

The speeds measured on the road network can be seen as a spatio-temporal time series. Although many models whether deterministic or stochastic have been created so far, these are either too time consuming to be used in real time, too demanding in terms of storage, or give too poor a result to prove valuable. Indeed the speeds present complex behaviors including seasonality, time dependency, spatial dependency and drastic quick changes of patterns, making it uneasy to forecast. Yet since the evolution of the speeds rely on a real behavior, these time series present strong correlations and causality links that must be found.

In this paper, we focus on the design and implementation of Neural Networks (NN) to handle the forecast of time series applied to the case of road traffic. Using NN allows one to include a large number of explicative variables inside the model to capture the complexity of the time series. For this we study three kinds of NN architectures – Feedforward (FNN), Convolutional (CNN) and Recurrent (RNN) – to deal with the road traffic speed forecasting task.

There has been a growing interest for deep learning in the recent years, see for instance [22] and references therein. Among all different kind of networks, CNN are mostly used to classify images as in [21] or [40], and RNN are mostly used for Natural Language Processing tasks [27], where one tries to guess the next word in a sentence given the previous ones. Yet for a regression task when dealing with time dependent data, few results exist in the statistical literature.

In this work, we propose a way of feeding the NN to enhance the causality due to the seasonality of the observations which will be compared to the usual one. In addition we propose specific designs of the neural networks to forecast the data, each adapted to the type of NN used in this study. Moreover, to evaluate the quality of the forecast, in addition to the standard Root Mean Square Error (RMSE), we propose a new Q^2 indicator, conceived so as to assess the quality of the model for time series forecast, completing the information conveyed by RMSE. Within this framework, we show that the Convolutional network severely underperforms the

other NN variants and give some reasons why this may be. We also present a simple physical model, embedding the seasonality of the road traffic, that outperforms all the deep learning techniques while using order of magnitudes less parameters.

The paper divides as follow. We present the speed data at our disposal in Section 1.2.1, and explain how we feed them to the NN in Section 1.2.2. The new criterion used to evaluate the quality of the different models is explained in Section 1.3. The models themselves -FNN, CNN, RNN-LSTM – are presented in Section 1.4. We finally present the setup of our different simulations and our numerical results in Section 1.5.

1.2 Data

1.2.1 Speeds from FCD measures

We deal here with speed forecasting on a road network. A road network is an oriented graph. On each oriented edge a speed is computed (via a technique called Floating Car Data, or FCD, see for instance in [36]). A road network has to be described by several oriented edges as the speed limit, the topology, the presence of a traffic light... change for different zones of the network. On a edge l of a road network, the speed at time τ is denoted $\tilde{v}_{l\tau}$. In our study, the speeds are available every 3 minutes, so that $\tilde{v}_{l\tau+1}$ comes 3 minutes after $\tilde{v}_{l\tau}$.

In practice, we will normalize speeds. This procedure helps to identify similar speed patterns: formation of congestion or its resorption , without being affected by the absolute speed values. Otherwise, a classification algorithm would for instance fail to cluster together two edges which never experience any traffic jams if their speed limits are respectively 50 and 130 kph. To normalize the speeds, one needs to know the so-called free flow speed $v_{l\tau}^{\mathrm{FFS}}$ as defined in [7]. The free flow speed on Paris Ring Road is roughly $65kph$. We will thus from now on deal with the series of observations

$$v_{l\tau} = \frac{\tilde{v}_{l\tau}}{v_{l\tau}^{\mathrm{FFS}}} \ .$$

In this paper, we will study the external Paris ring road: 396 oriented edges in our network.

Figure 1.1: External Paris Ring Road.

The training set will be data from the first 9 months of 2016, the test set belonging to tenth month of 2016. Note that to avoid predicting on trivial traffic states of constant speeds, we first removed the speed data from 11pm to 5 am of each day of our 10 month sample.

1.2.2 Input speeds

In addition to the freedom of design in the FNN, CNN and RNN architectures, the way the past speeds are taken into account in the input of the networks play a crucial role in the results that one can obtain. Shall we take nearby edge past speed information, or can each edge predict its future speeds knowing only its past ones ? How far into the past shall we go ? Hence the creation of the learning set determines the behaviour of the model and should be studied with care.

We will consider two kinds of input for the different NN architecture that we implement: a full input (large quantity of data) and a reduced one (supposed to capture the seasonality of the road traffic)

Full input

The full input of the NN corresponds to data on the current day (where one is trying to predict future speeds) and the immediate past $\mathcal{D}$ days. On these $\mathcal{D}+1$ days we consider N_0 contiguous edges of our graph. Starting at an arbitrary edge $N^{(t)}$ and an arbitrary time $T^{(t)}$, the full input of the network is then

$$\left(\left\{v_{N^{(t)}+l\,T^{(t)}-b}\right\},\left\{v_{N^{(t)}+l\,T^{(t)}-\delta D+p}\right\}\right) \ , \tag{1.2.1}$$

with

$$l \in [\![0, N_0-1]\!] \ , \qquad b \in [\![1, \mathcal{B}_f]\!] \ ,$$
$$\delta \in [\![1, \mathcal{D}_f]\!] \ , \qquad p \in [\![-\mathcal{P}_{1f}, \mathcal{P}_{2f}]\!]$$

where the f subscript stands for full. The t index represents the t sample of our training set, and D is a constant representing a full day in 3 minute intervals (480). We are thus considering the $3\mathcal{B}_f$ minutes of speed data on the current day (just before $T^{(t)}$), and a time window of $3\mathcal{P}_{1f}$ minutes before $T^{(t)}$ and $3\mathcal{P}_{2f}$ minutes after $T^{(t)}$ on past days. In practice we take

$$N_0 = \mathcal{B}_f = 32 \ , \qquad \mathcal{D}_f = 7 \ ,$$
$$\mathcal{P}_{1f} = 15 \ , \qquad \mathcal{P}_{2f} = 16 \ .$$

This corresponds to F_{0f} input variables with

$$F_{0f} = N_0 \times (\mathcal{B}_f + [\mathcal{P}_{1f} + \mathcal{P}_{2f} + 1]\,\mathcal{D}_f) = 8192 \ .$$

The prediction task corresponding to this full input is

$$\left\{v_{N^{(t)}+l\,T^{(t)}+h}\right\} \ ,$$

with $h \in [\![0, \mathcal{H}_f - 1]\!]$, and in practice we will take $\mathcal{H}_f = 20$, hence an output of size $N_0 \times \mathcal{H}_f = 640$. The full input and the corresponding output is illustrated in Figure 1.2 for a given t sample of the training set.

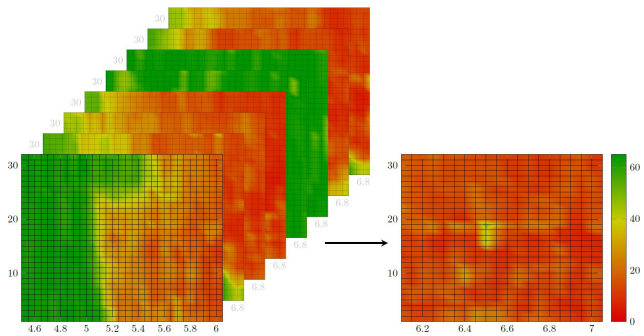

Figure 1.2: full input used for the different NN architectures (left). Prediction example (right)

Here, the output of the network (on the right part of Figure 1.2) is the subsequent 20 three minute time intervals of the N_0 links of the current instance t of the training set (time between 6:06 and 7:03 am). On the left part of Figure 1.2, the eight colored matrices correspond to the traffic on the current day (matrix in the foreground, time between 4:30 and 6:03 am) and the traffic on the seven previous days (matrices in the background, time between 5:18 and 6:51 am).

Reduced input

Starting back from our arbitrary edge $N^{(t)}$ and arbitrary time $T^{(t)}$, the reduced input deals with one edge. We thus forget the l index in this part. The reduced input reads

$$\left(\left\{v_{N^{(t)}\,T^{(t)}-b}\right\}, \left\{\bar{v}_{N^{(t)}\,T^{(t)}-\delta D+p}\right\}\right) \ . \tag{1.2.2}$$

where (the r subscript stands for reduced)

$$b \in [\![1, \mathcal{B}_r]\!] \ , \qquad \delta \in [\![1, \mathcal{D}_r]\!] \ , \qquad p \in [\![-\mathcal{P}_{1r}, \mathcal{P}_{2r}]\!] \ .$$

The signification of b and δ are unchanged (see Section 1.2.2). In practice we will take

$$\mathcal{B}_r = 4 \ , \qquad \mathcal{D}_r = 7 \ .$$

However, the way in which the previous days are taken into account is changed as follows. Instead of taking high resolution but noisy 3 minutes intervals, we will average over $\mathcal{M}$ such intervals, so that

$$\bar{v}_{N^{(t)}\,T^{(t)}-\delta D+p} = \frac{1}{\mathcal{M}} \sum_{i=0}^{\mathcal{M}-1} v_{N^{(t)}\,T^{(t)}-\delta D+\mathcal{M}p+i} \ .$$

In practice we will take

$$\mathcal{M} = 5 \ , \qquad \mathcal{P}_{1r} = 0 \ , \qquad \mathcal{P}_{2r} = 3 \ .$$

The reduced input will thus be of size

$$F_{0r} = \mathcal{B}_r + [\mathcal{P}_{1r} + \mathcal{P}_{2r} + 1]\,\mathcal{D}_f = 32$$

The corresponding prediction task will now be $\left\{v_{N^{(t)}\,T^{(t)}+h}\right\}$, with $h \in [\![0, \mathcal{H}_r - 1]\!]$, and in practice we will take $\mathcal{H}_r = 20$, and the output will be of size $\mathcal{H}_r = 20$. The reduced input and its corresponding output is illustrated in Figure 1.3

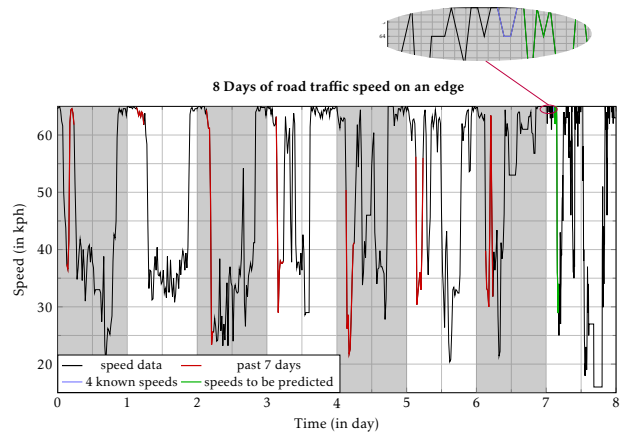

Figure 1.3: Reduced input example.

On Figure 1.3, the traffic is represented for 8 contiguous days on the $N^{(t)}$'th link of the Paris road network. The first 7 days (x axis between 0 and 7) are considered to be past days, and are pictured averaged per quarter-hours (corresponding to our $\mathcal{M}$ choice). The 8th day (x axis between 7 and 8) is depicted by 3 minute intervals. $T^{(t)}$ characterizes the starting time at which one tries to predict afterwards. In Figure 1.3, this corresponds to the green part of the curve (see main plot and inset). The $v_{N^{(t)}\,T^{(t)}-b}$ are the blue part of the curve (see both main plot and inset) while the $\bar{v}_{N^{(t)}\,T^{(t)}-\delta D+p}$ are shown in red.

1.3 Model evaluation

1.3.1 RMSE and Q-score

Generally, in the speed forecasting paradigm[5],[9],[28],[35][37],[39] one evaluates the quality of a prediction algorithm considering the Root Mean Square Error (RMSE)

$$\text{RMSE} = \sqrt{\frac{1}{LT}\sum_{h=0}^{T}\sum_{l=1}^{L}\left(v_{l\tau+h} - \hat{v}_{l\tau+h}\right)^2}$$

where $\hat{v}_{l\tau}$ is the speed predicted by the algorithm under consideration, $v_{l\tau+h}$ the ground truth speed, T (19 in our study) covering all the time intervals where the prediction is taking place, and L all the links of the road network considered. In our opinion, this indicator is not sharp enough to correctly assess the quality of a prediction algorithm. Indeed, for a constant time series, any RMSE close to 0 could fool people into thinking that the prediction algorithm is a good one, despite being worse than the prediction of taking a constant. At the other extreme, for a widely and swiftly changing time series, a large RMSE does not necessarily imply that the algorithm is a poor one.

To paliate this apparent paradox, it is necessary to introduce a new benchmark prediction. In our study we chose the real time propagation benchmark RTPB (taking $v_{l\tau}$ for any future speed $v_{l\tau+h}$), which is what is widely used in industrial DRS described for instance in [8]. An algorithm with a low RMSE is not worth much interest if it constantly predicts worse than this simple benchmark, and an algorithm with a large RMSE but constantly beating the benchmark may be worth considering; it might just be that the road network under investigation is experiencing

a large change of speed. We thus introduce

$$Q^2 = 1 - \frac{\mathrm{RMSE}^2}{\mathrm{RMSE}^2_{\mathrm{bench}}} \ . \tag{1.3.1}$$

where

$$\mathrm{RMSE}_{\mathrm{bench}} = \sqrt{\frac{1}{LT}\sum_{h=0}^{T}\sum_{l=1}^{L}\left(v_{l\tau+h} - v_{l\tau}\right)^2} \ .$$

This Q-score (or Q^2 in the following) quantifies the improvement (or deterioration) of the considered algorithm when compared with the RTPB prediction. If $Q^2 > 0$ then there is an improvement (a Q^2 equal to 1 would mean a perfect prediction), otherwise if $Q^2 < 0$ there is a deterioration. Without this Q^2 specification, any speed prediction on an almost constant speed road network could lead to a very low RMSE without nevertheless being good, and one might be tempted to choose these kind of networks to beat "state of the art" RMSE. We hasten to add that this is not what was done by the papers that we reviewed [37, 9], but it is notwithstanding hard to evaluate a traffic prediction algorithm having in mind real life applications if one solely consider the RMSE. Some studies[5, 25] use an alternative indicator: the Mean Absolute Percentage Error (MAPE).

$$\mathrm{MAPE} = \frac{100}{LT}\sum_{h=0}^{T}\sum_{l=1}^{L}\left|1 - \frac{\hat{v}_{l\tau+h}}{v_{l\tau+h}}\right| \ .$$

Not using any benchmark algorithm, the MAPE is however subject to the same drawbacks as the RMSE when used alone.

Loss function

The loss function is a critical component of a NN architecture. For a classification task, the cross-entropy loss function is a standard choice. But in this study, as we are trying to make quantitative speed predictions that could feed a DRS, we will take a quadratic loss function

$$J_t(\Theta) = \sum_{l=0}^{N_0-1}\sum_{h=0}^{\mathcal{H}-1}\left(\hat{v}_{N^{(t)}+l\,T^{(t)}+h} - v_{N^{(t)}+l\,T^{(t)}+h}\right)^2 \ ,$$

where v is the ground truth speed and $\hat{v}$ the output of the NN under consideration. See Section 1.2.2 for more details on the other notations.

Note that considering this particular loss function means that we are considering the speed forecasting problem as a regression task.

J_t corresponds to the loss error on one sample of the training set, and we still have to decide how to train the NN. We make the standard choice of mini-batch Stochastic Gradient Descent (SGD) as in [19], at each epoch $\mathcal{E}$ –one iteration of the training procedure, we pick T_{mb} samples of the training set, and compute the mini-batch loss

$$J(\Theta) = \frac{1}{2T_{\mathrm{mb}}}\sum_{t=0}^{T_{\mathrm{mb}}-1} J_t(\Theta) \ . \tag{1.3.2}$$

To achieve better accuracy, we had to regularize the Loss function. This is achieved using an $\ell^1 + \ell^2$ penalty, also known as elastic net penalty, see for instance in [41], detailed in Section 1.4.1.

In this study we picked $T_{\mathrm{mb}} = 50$ and $\mathcal{E} \sim 5.10^4$. The value of $\mathcal{E}$ is indicative, as we performed early stopping in order to improve accuracy as it is often done by users of NN as in [2] to overcome the issue of the convergence of the algorithm.

1.4 The models

We present the different NN studied, starting with their common blocks and ending with their specificities

1.4.1 Common properties of the Networks

Output function

As we deal with normalized speed, we constrain the output of our model $\hat{v}$ to be between zero and one. We hence picked an output function o such that

$$o(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } 0 \leq x \leq 1 \\ 1 & \text{if } x > 1 \end{cases}$$

Batch Normalization

Batch normalization (BN) is the most popular regularization procedure and consists in jointly normalizing the mini-batch sets per data types at each input of a NN layer, except for the input of the network itself. This is because we want to keep track of what the data represents, hence keep their mean and standard deviation untouched. It should be mentioned from the outset that BN is an empirical procedure, though it has been shown to drastically improve state of the art performance on classification tasks for instance on CIFAR/MNIST on many challenges.

In the original paper[18], the authors argued that this step should be done after the weight averaging/convolution (WA/C) operation and before the non-linear activation (NAC).

However, this choice stands on no theoretical grounds, and defeats the purpose of presenting a standardardized input to each NN layer. In addition, the back-propagation rules are a bit more cumbersome to write with the WA/C-BN-NAC convention. We therefore opted for a WA/C-NAC-BN architecture for all the layers – except of course the pooling ones where there is nor an NAC neither a BN – presented in this paper.

For the technical details on our BN implementation, see Section 1.9 of the supplementary material.

Elastic net

In practice, using equation (1.3.2) for the loss function leads to poor results: the different NN that we considered never reach a positive Q^2. We thus had to regularize the loss function. This we did by using both ℓ_1 and ℓ_2 penalties, a procedure which turns our NN to so-called elastic nets. Calling $\Theta_{f'}^{(\nu)f}$ the weight matrix between the ν'th and the $\nu+1$'th layer of our NN, this procedure amounts to add the following terms to the loss function

$$J_{\mathrm{reg}}(\Theta) = J(\Theta) + \frac{\lambda_{\ell_2}}{2} \sum_{\nu=0}^{N} \sum_{f'=0}^{F_\nu - 1} \sum_{f=0}^{F_{\nu+1}-1} \left(\Theta_{f'}^{(\nu)f}\right)^2 + \lambda_{\ell_1} \sum_{\nu=0}^{N} \sum_{f'=0}^{F_\nu - 1} \sum_{f=0}^{F_{\nu+1}-1} \left|\Theta_{f'}^{(\nu)f}\right| .$$

With F_ν ($F_{\nu+1}$) the size of the ν'th ($\nu+1$'th) layer of the NN. These new terms play a role in the update rules of the weight matrices as shown in [6].

In our numerical simulations, we took λ_{ℓ_1} and λ_{ℓ_2} to be in the range $10^{-4} - 10^{-1}$, the best value being picked by cross-validation.

No Dropout

In our study, it turned out that dropout – with the probability p of retaining a unit varying between $0.2 - 0.8$ for both the input and hidden layers – only slows down convergence without reducing the RMSE. We therefore chose to discard dropout from our architectures.

Adam Optimizer

For the mini-batch SGD used in backpropagation[24], we used the Adam optimizer proposed for instance in [20] which keeps track of both the weight gradient Δ_e^Θ ($e \in [\![0, \mathcal{E}-1]\!]$) and its square via two epoch dependent vectors m and v. For a given weight matrix Θ, we thus have

$$m_{\mathrm{e}} = \beta_1 m_{\mathrm{e}-1} + (1-\beta_1)\Delta_e^\Theta \ ,$$
$$v_{\mathrm{e}} = \beta_2 v_{\mathrm{e}} + (1-\beta_2)\left(\Delta_e^\Theta\right)^2 \ ,$$

with β_1 and β_2 parameters respectively set to 0.9 and 0.999, as advocated in the original paper[20]. We also observed that the final RMSE is poorly sensitive to the specific β values. The different weight matrices Θ are then updated thanks to ($\epsilon = 10^{-8}$)

$$\Theta_e = \Theta_{e-1} - \frac{\eta}{\sqrt{v_{\mathrm{e}} + \epsilon}} m_{\mathrm{e}} \ .$$

This is the optimization technique used for all our NN, along side a learning rate decay $\eta_e = e^{-\alpha_0}\eta_{e-1}$, where α_0 determined by cross-validation, and η_0 is usually initialized in the range $10^{-3} - 10^{-2}$. Without this weight decay, the NN perform poorly.

Activation function

We used Leaky-ReLU activation functions in all our simulations (slight improvement on the standard ReLU choice), see for instance in [1]. We also tested without success the ELU choice.

1.4.2 Specificities of each Network

Feedforward Neural Networks

For our FNN, we considered both one and three hidden layer architectures, with the WA-NLA-BN structure advocated in Section 1.4.1. We added no bias to the WA equation, as the latter is handled by the BN procedure, see the supplementary material for more details. We took a unique size F_{H} for the hidden layers, and varied it in between $8 - 256$.

We did not consider ResNet architectures[16] in this study. This might be a natural following step, but in the present state of the art techniques for regression task with NN, we are pessimistic on any improvement this might achieve. Indeed, as shown in our results, our 1 hidden layer FNN outperforms its 3 layer cousin.

Input of the FNN : To feed our FNN, we just stuck together either the full or the reduced input (see equations (1.2.1) and (1.2.2)) into a matrix $X_f^{(t)}$. Here t is the mini-batch index and $f \in F_{0(fr)}$ for respectively the full and reduced inputs.

Weight initialization : we used the standard prescription[11]

$$\left[\Theta^{(\nu)}\right]_{\text{init}} = \sqrt{\frac{6}{F_\nu + F_{\nu+1}}} \times \mathcal{N}(0,1) \ .$$

Convolutional Neural Networks

In our CNN study, we implemented a 16 weights VGG architecture [31] (ResNet could be considered in a future study). The input feature map is of size $\mathcal{D}_f + 1$, while the images composing the input are of size $\mathcal{B}_f \times N_0$ for the current day and $(\mathcal{P}_{1f} + \mathcal{P}_{2f} + 1) \times N_0$ for the previous days. In practice this corresponds to eight 32×32 images, and we therefore implemented the standard convolutions using 3×3 receptive fields with 1 strides and a padding equal to 1 on each image edge. We use pooling of 2×2 receptive fields with 1 strides. The size of the hidden feature maps has been taken in between $32 - 512$, and the size of the subsequent fully connected layers in between $256 - 512$. More details on the architecture can be found in Section 2 of the supplementary material.

Input of the CNN : For the CNN, we only considered the full input introduced in Section 1.2.2. Figure 1.2 makes complete sense for CNNs, as its left part represents the different feature maps of the input. The latter is now a four dimensional tensor $X^{(t)}_{\delta l(pb)}$ with t the mini-batch index and

$$\text{for } \delta = 0, \ X^{(t)}_{\delta lb} = v_{N^{(t)} + l\, T^{(t)} - b} \ ,$$
$$\text{for } \delta \neq 0, \ X^{(t)}_{\delta lp} = v_{N^{(t)} + l\, T^{(t)} - \delta D + p} \ ,$$

Weight initialization : We used the same prescription as for FNN.

1.4.3 Recurrent Neural Networks

For our RNN, we used the LSTM variant [38] with no peepholes in the input, forget and output gates. The latter are taken to be standard logistic σ functions, while the cell update as well as the hidden state update are taken to be tanh functions. We performed BN as in the Feedforward/Convolutional before each input of a hidden layer. We took one hidden layer in the spatial direction and $\mathcal{H}$ in the temporal one.

Input of the RNN-LSTM : an additional subtlety arises for RNN: the input depends on the temporal direction τ of the network. For the full input, we just took (f here stands for (b, p, l, δ) with $l = 0$ for the reduced input)

$$X^{(t)(\tau)}_f = \left(\left\{ v_{N^{(t)} + l\, T^{(t)} + \tau - b} \right\}, \left\{ v_{N^{(t)} + l\, T^{(t)} + \tau - \delta D + p} \right\} \right) \ ,$$

implying that the input of a current temporal layer has to be fed with the output of the previous ones. We did just so in practice. We took the same F_{H} as for FNNs.

Weight initialization : We used a diagonal prescription inspired from [32] (J is the unit matrix, $\varepsilon \sim 10^{-2}$)

$$\left[\Theta^{(\nu)}\right]_{\text{init}} = \frac{1}{2}\mathbb{I} + J\mathcal{N}(0, \varepsilon) \ .$$

No clipping : Clipping [14] did not improve our model performances. We thus removed it.

1.5 Results

With all the building blocks in place, we report here in Figures 1.4 and 1.5 the compared performance of all the algorithms.

1.5.1 Full input

The RMSE of our different NN can be found for the full input in Figure 1.4. Here one can see several particular patterns.

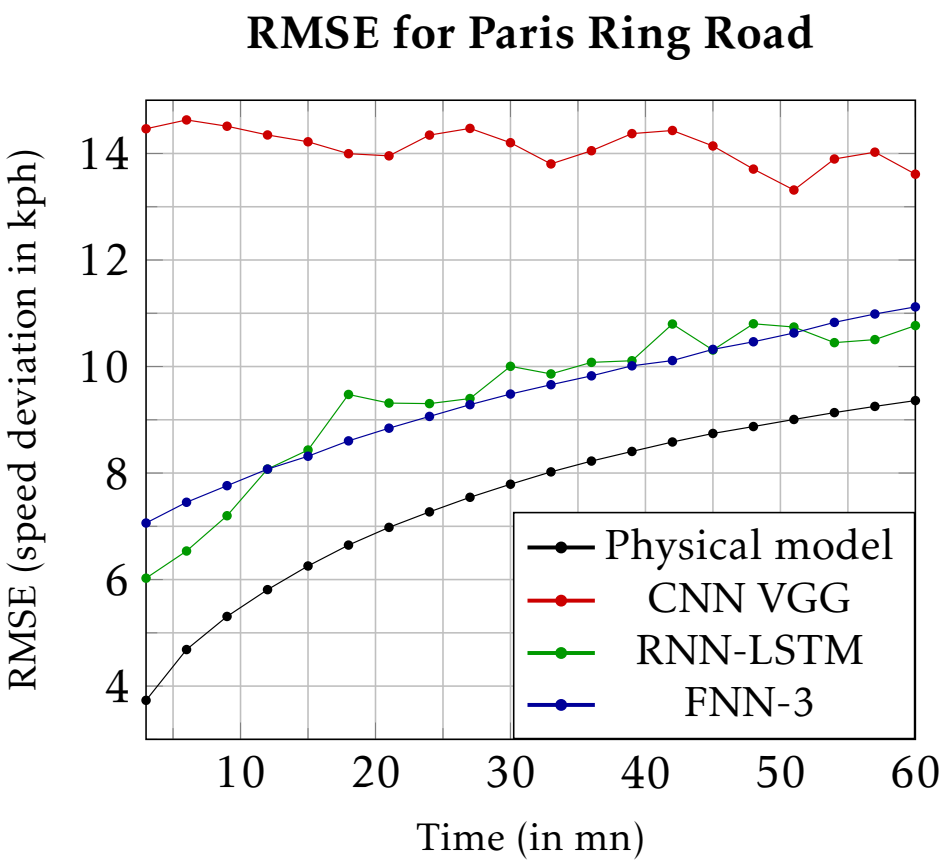

Figure 1.4: RMSE for the different models studied.

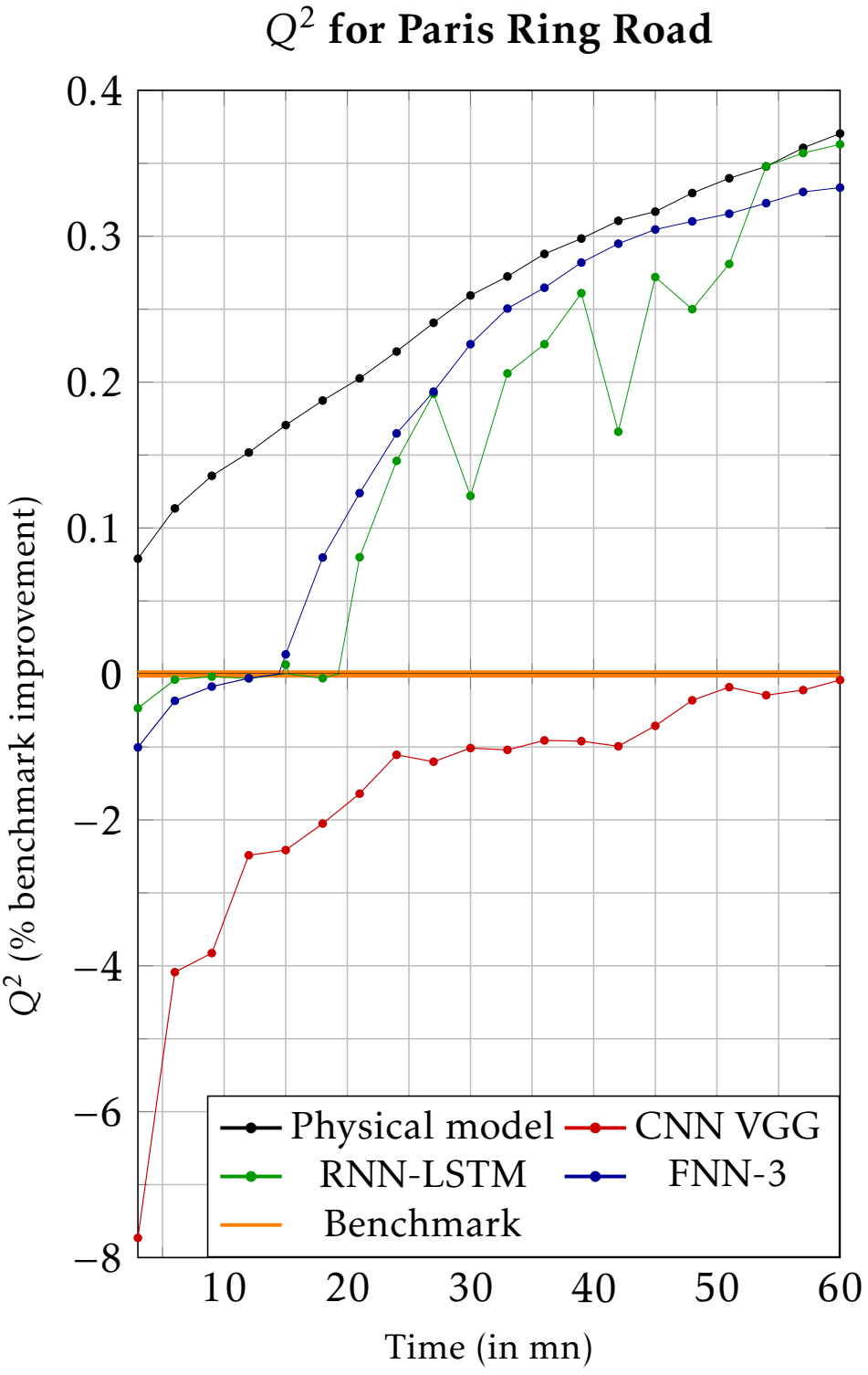

Figure 1.5: Q-score for the different models studied.

Poor CNN performance : the CNN severely underperforms the FNN and the RNN-LSTM. As the evolution of the car speeds is supposed to be a spatio-temporal field, considering the daily spatio-temporal velocities as images that characterize the traffic seemed to be a good

frame to use CNN that proved helpful in pattern recognition. Yet road traffic speed prediction is not a translationally invariant problem in the temporal direction and no matter how the CNN weight matrices are tuned, the network assumes that the input image is translational invariant. The CNN hence fails to capture the fact that the right part of the input images in Figure 1.2 play a more important role that the left part, since it represents more recent traffic states. We extend our remark to any time series problem: the CNN seems ill equipped to handle prediction task properly in the case where there is strong causality in the time direction. Note however that another study in [37] reports otherwise on some specific Chinese road networks, while only feeding the CNN with past traffic states of the current day (feature map of size 1 for the input layer). This may be due to some particular conditions in Chinese trafic. Yet the performance of NN are better than standard methods based on density models [26].

Similar FNN/RNN-LSTM performance : The two other kind of networks that we considered performed similarly, with a RMSE between $6 - 11kph$.

A simpler model performs better : We developed a simpler physical model (PM) embedding the seasonality of the traffic. This model outperforms all NN, reaching state of the art RMSE value. Note that this work will be published in a future patent.

Q^2 utility : We can assess the quality of our prediction when compared to the RTPB. We observe that FNN outperforms RNN-LSTM, despite having an almost identical RMSE. PM outperforms all NN, especially at early times.

Storage issues for real life applications : the best FNN-3/VGG/LSTM models that we obtained have respectively 284672/243712/8224 weight parameters, not counting the BN ones. Our best PM has 640 parameters in total. In addition, it has a high degree of generalizability, allowing to mutualize the parameters for large road networks.

1.5.2 Reduced input

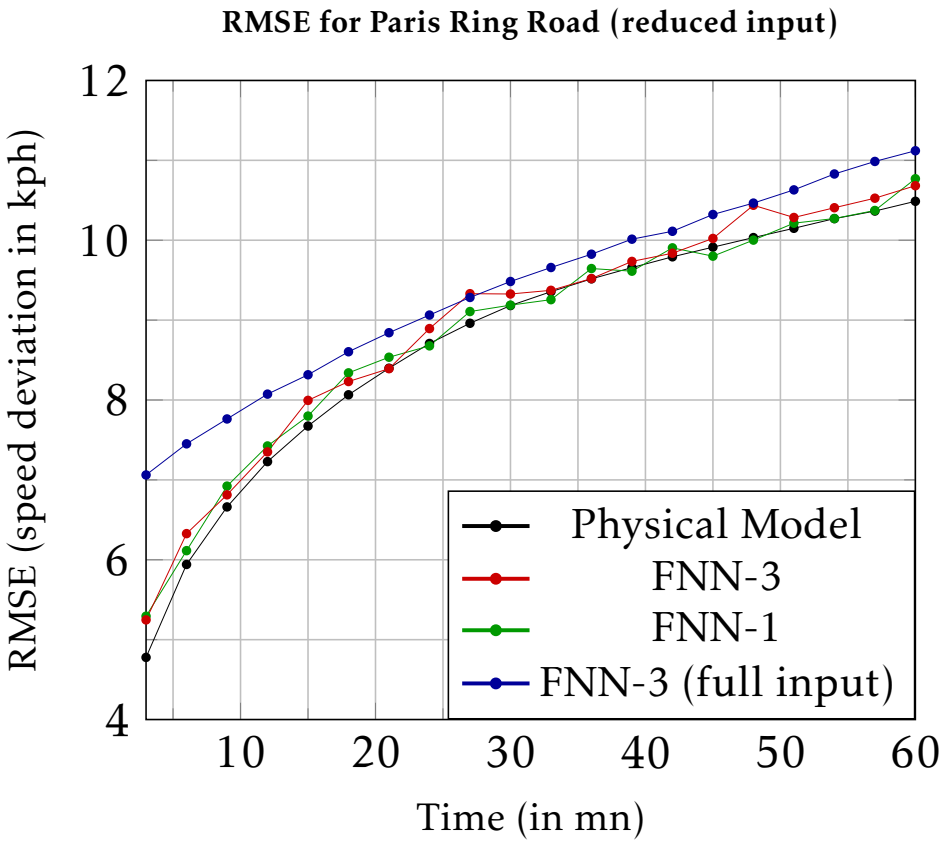

Figure 1.6: RMSE for the reduced input

We present in Figure 1.6 and 1.7 the results obtained – by two FNN architectures and the physical model – for the reduced input. From these Figures we draw

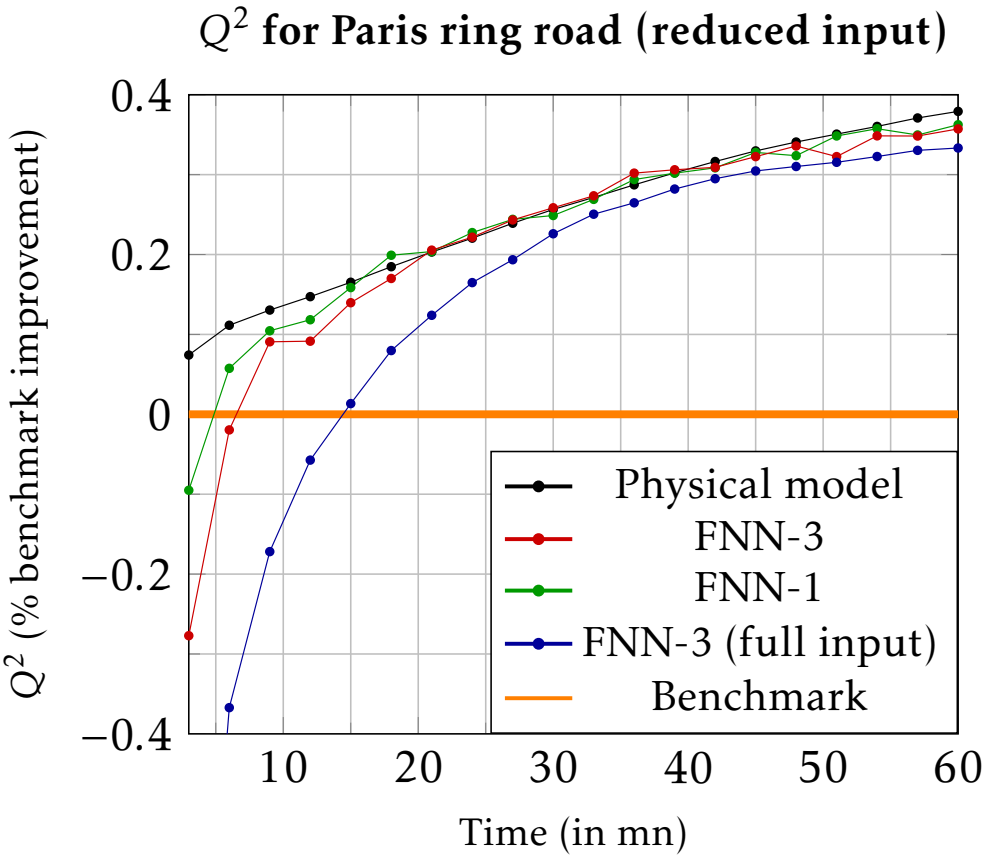

Figure 1.7: Qbench for for the reduced inputs

The reduced input outperforms the Full one : The full input, though with a large number of explanatory variables, leads to worse results in terms both of Q^2 and RMSE. This might be due to either a poor initial condition choice – we hope not – or to the regression problem specificities.

The shallow FNN outperforms the deep one : FNN-1 gives slightly better results than FNN-3.

The PM still outperforms NN: but not by much.

1.5.3 Importance of the Q-score

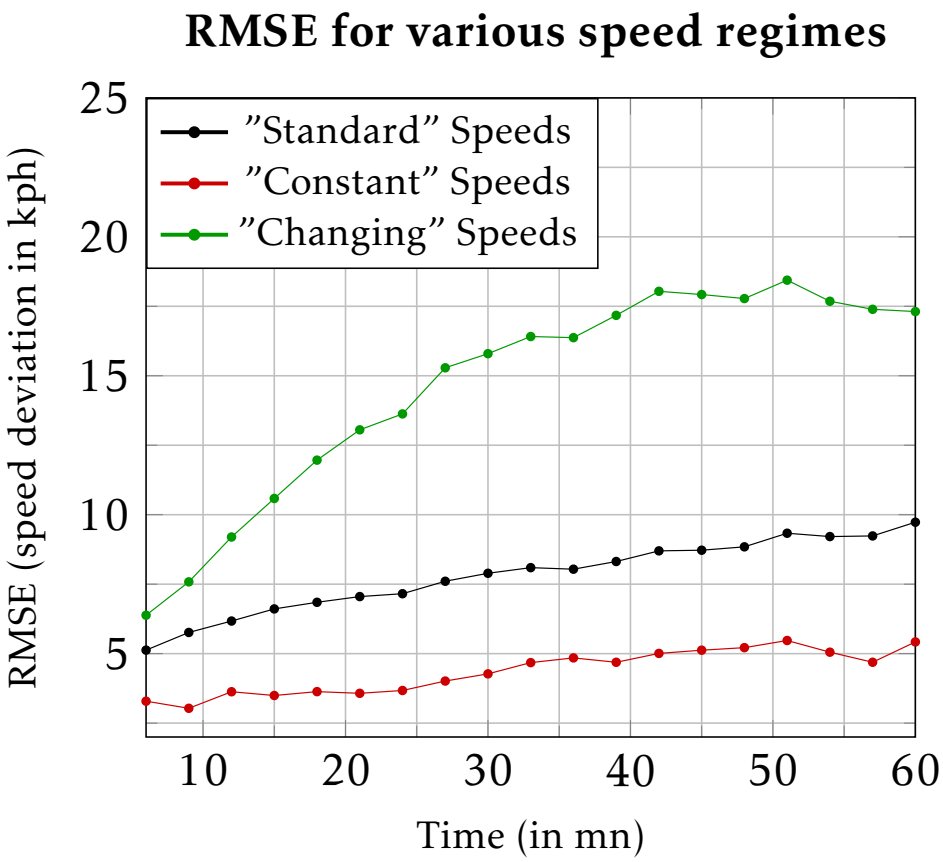

Figure 1.8: RMSE for different speed regimes

We have studied three speed regimes for the PM: the "constant" ("changing") one is the 10% part of the training set where the speeds before and after $T^{(t)}$ vary the less (more). The standard regime is the remaining 80%. Results are shown in Figures 1.8 and 1.9

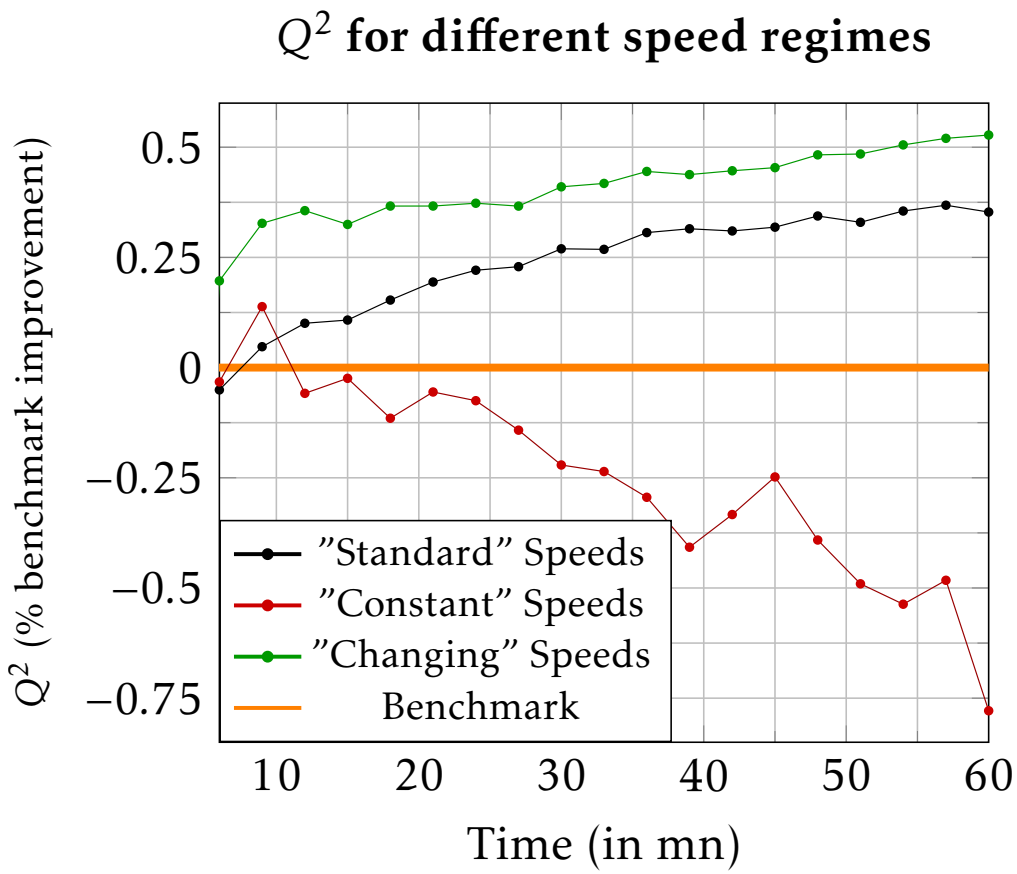

Figure 1.9: Q-score for different speed regimes

Constant speeds lead to better RMSE : The less the speed vary, the better the RMSE is, as expected.

Changing speeds lead to better Q^2 : Having low RMSE is not on its own a sign of a working model. The PM can't beat the Benchmark for "constant" speeds, while beating it by more than 50% after 50 minutes for the changing speeds. One should therefore jointly state RMSE and Q^2 in time series tasks.

1.6 Conclusion

We have implemented popular deep learning architectures, adapting their designs to the specific regression problem of predicting future road speeds, a generic example for time series presenting strong causality issues in both time and space. We showed that the CNN underperforms the other networks while we built a PM that outperforms all NN architectures. We show that feeding the NN with more data leads to worst results, as does adding more layers to the FNN. We finally designed a new indicator, the Q^2, to be used jointly with the RMSE in time series problems.

Chapter 2

Deep Learning: supplementary and technical description

2.1 Feedforward Neural Networks

Feedforward networks (FNN) are an extension of the perceptron algorithm[29]. Their architecture is simple, but training them can be a daunting task. In the next few Sections, we introduce the minimum FNN mathematical survival kit, stating what – to the best of our knowledge – are the empirical tricks and what stands on firmer mathematical ground.

2.1.1 Some notations and definitions

In the following, we call

- N the number of layers (not counting the input) in the Neural Network.
- T_{train} the number of training examples in the training set.
- T_{mb} the number of training instances in a mini-batch (more on that later).
- F_ν the number of neurons in the ν'th layer.
- $t \in [\![0, T_{\text{mb}} - 1]\!]$ the mini-batch training instance index.
- $\nu \in [\![0, N]\!]$ the index of the layer under consideration in the FNN.
- $h_f^{(t)(\nu)}$ where $f \in [\![0, F_\nu - 1]\!]$ the neurons of the ν'th layer.
- $X_f^{(t)} = h_f^{(t)(0)}$ where $f \in [\![0, F_0 - 1]\!]$ the input variables.
- $y_f^{(t)}$ where $f \in [0, F_N - 1]$ the output variables (to be predicted).
- $\hat{y}_f^{(t)} = h_f^{(t)(N)}$ where $f \in [0, F_N - 1]$ the output of the FNN.
- $\Theta_f^{(\nu)f'}$ for $f \in [0, F_\nu - 1]$, $f' \in [0, F_{\nu+1} - 1]$ and $\nu \in [0, N - 1]$ the weight matrices.
- A bias term can be included. In practice, we see when talking about the batch-normalization procedure that we can omit it, and we choose to do so in all our definitions.

2.1.2 FNN architecture

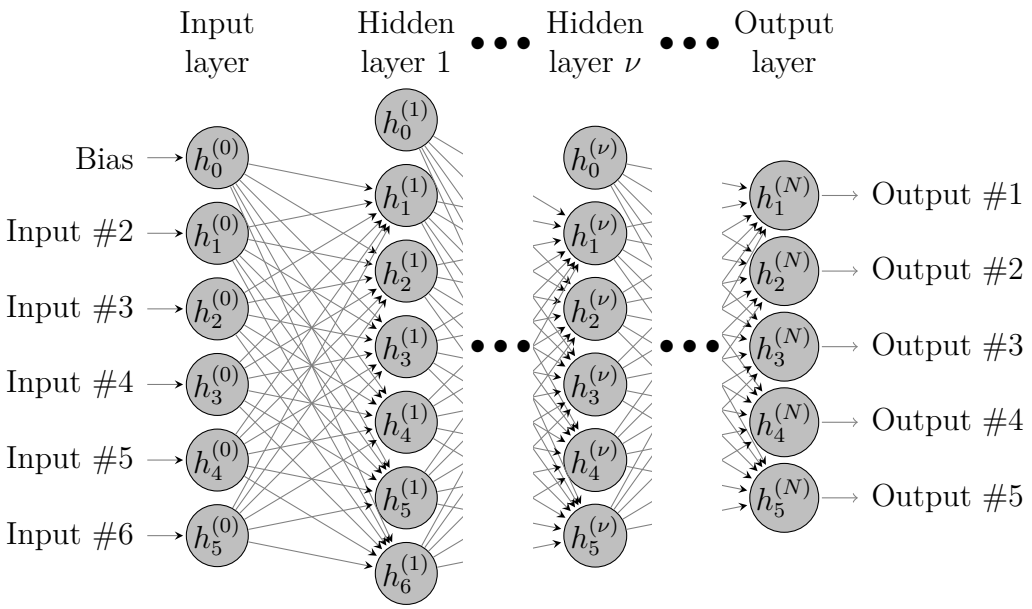

Figure 2.1: Neural Network with $N+1$ layers ($N-1$ hidden layers). For simplicity of notations, the index referencing the training set has not been indicated. Shallow architectures (considered in the core of the paper) use only one hidden layer. Deep learning amounts to take several hidden layers, usually containing the same number of hidden neurons. This number should be in the ballpark of the average of the number of input and output variables.

A FNN is made of one input layer, one (shallow network) or more (deep network, hence the name deep learning) hidden layers and one output layer. Each layer of the network (except the output one) is connected to a following layer. This connectivity is central to the FNN structure and has two main features in its simplest form: a weight averaging feature and an activation feature. We review these features extensively in the following subsections

2.1.3 Weight averaging

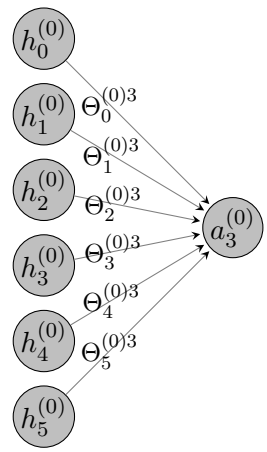

Figure 2.2: Weight averaging procedure.

One of the two main components of a FNN is a weight averaging procedure, which amounts to average the previous layer with some weight matrix to obtain the next layer. This is illustrated in Figure 2.2

Formally, the weight averaging procedure reads:

$$a_f^{(t)(\nu)} = \sum_{f'=0}^{F_\nu - 1} \Theta_{f'}^{(\nu)f} h_{f'}^{(t)(\nu)} = \left(\Theta^{(\nu)T} h^{(t)(\nu)}\right)_f \ . \tag{2.1.1}$$

Here we have delibarately ommited a potential bias term, as it is handled in Section 2.1.9 where we talk about batch normalization. In practice, for all our numerical simulations, the weights are learned using the backpropagation procedure[24] with the Adam optimizer method for gradient descent[20].

2.1.4 Activation function

The hidden neuron of each layer is defined as

$$h_f^{(t)(\nu+1)} = g\left(a_f^{(t)(\nu)}\right) \ , \tag{2.1.2}$$

where g is an activation function – the second main ingredient of a FNN – whose non-linearity allows the prediction of arbitrary output data. In practice, g is usually taken to be either a sigmoid, a tanh a Rectified Linear Unit[15] (ReLU), or its variants: Leaky ReLU, ELU...[4]. The ReLU is defined as

$$g(x) = \mathrm{ReLU}(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases} \ . \tag{2.1.3}$$

Its derivative is

$$\mathrm{ReLU}'(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases} \ . \tag{2.1.4}$$

The choice of the activation function usually follows empirical tests, though it has been formally shown [3],[30] that Neural Networks cannot converge if the activation function is too complicated. With these building blocks in place, let us consider the different layers of the network one at a time.

2.1.5 Input layer

As explained in the core of the paper, we consider a full and a reduced input. But for all purposes here, we deal with an $h_f^{(t)(0)} = X_f^{(t)}$ input layer, with $f \in [\![0, F_0 - 1]\!]$ and $t \in [\![0, T_{\mathrm{mb}} - 1]\!]$.

2.1.6 Fully connected layer

The fully connected operation is just the conjunction of the weight averaging and the activation procedure. Namely, for $\nu \in [\![0, N-1]\!]$

$$a_f^{(t)(\nu)} = \sum_{f'=0}^{F_\nu - 1} \Theta_{f'}^{(\nu)f} h_{f'}^{(t)(\nu)} \ , \tag{2.1.5}$$

and for $\nu \in [\![0, N-2]\!]$

$$h_f^{(t)(\nu+1)} = g\left(a_f^{(t)(\nu)}\right) \ . \tag{2.1.6}$$

In the case where $\nu = N - 1$, the activation function is replaced by an output function (see the next section).

2.1.7 Output layer

The output of the FNN reads

$$h_f^{(t)(N)} = o\left(a_f^{(t)(N-1)}\right) \ , \tag{2.1.7}$$

where o is called the output function. In the case of the Euclidean loss function (se the next section), the output function is just the identity. Nevertheless, in our case we know that the normalized speeds cannot take values greater than 1 and smaller than 0, we therefore impose this constraint on the output of our FNN and take

$$o(x) = \begin{cases} 0 & x < 0 \\ x & 0 \leq x \leq 1 \\ 1 & x > 1 \end{cases} \ . \tag{2.1.8}$$

2.1.8 Loss function

The loss function evaluates the error made by the FNN when it tries to estimate the data to be predicted. As explained in the core of the paper for a regression problem this is generally the mean square error (MSE)

$$J(\Theta) = \frac{1}{2T_{\mathrm{mb}}} \sum_{t=0}^{T_{\mathrm{mb}}-1} \sum_{f=0}^{F_N - 1} \left(y_f^{(t)} - h_f^{(t)(N)}\right)^2 \ . \tag{2.1.9}$$

To obtain better results, one usually regularizes the Loss function. In addition to Batch normalization (explained in the next section), we also use ℓ_1 and ℓ_2 regularization. This amounts to add the following terms to the loss function

$$J_{\text{reg}}(\Theta) = J(\Theta) + \frac{\lambda_{\ell_2}}{2} \sum_{\nu=0}^{N} \sum_{f'=0}^{F_\nu - 1} \sum_{f=0}^{F_{\nu+1}-1} \left(\Theta_{f'}^{(\nu)f}\right)^2 + \lambda_{\ell_1} \sum_{\nu=0}^{N} \sum_{f'=0}^{F_\nu - 1} \sum_{f=0}^{F_{\nu+1}-1} \left|\Theta_{f'}^{(\nu)f}\right| \ . \tag{2.1.10}$$

These new terms play a role in the update rules of the weight matrices[6].

2.1.9 Batch Normalization

As explained in the core of the paper, Batch normalization (BN) consists in jointly normalizing the mini-batch sets T_{mb} per data types F_ν at each input of a NN layer, except for the input of the network itself. In our case, we thus consider for $\nu \in [\![0, N-2]\!]$

$$\tilde{h}_f^{(t)(\nu)} = \frac{h_f^{(t)(\nu+1)} - \hat{h}_f^{(\nu)}}{\sqrt{\left(\hat{\sigma}_f^{(\nu)}\right)^2 + \epsilon}} \ . \tag{2.1.11}$$

Here, $\hat{h}_f^{(\nu)}$ and $\hat{\sigma}_f^{(\nu)}$ are respectively the mean and the standard deviation of $h_f^{(t)(\nu+1)}$ with respect to the batch index t. To make sure that the Batch Normalization operation can also represent the identity transform, we standardly add two additional parameters (γ_f, β_f) to the model (learned by backpropagation [6])

$$y_f^{(t)(\nu)} = \gamma_f^{(\nu)} \, \tilde{h}_f^{(t)(\nu)} + \beta_f^{(\nu)} \ . \tag{2.1.12}$$

The presence of the coefficient $\beta_f^{(\nu)}$ is what pushed us to get rid of the bias term. Indeed, it is now naturally included in batchnorm. During training, one must compute a running sum for the mean and the variance, that serve for the evaluation of the cross-validation and the test set. calling e the current epoch

$$\mathbb{E}\left[h_f^{(t)(\nu+1)}\right]_{e+1} = \frac{e\mathbb{E}\left[h_f^{(t)(\nu)}\right]_e + \hat{h}_f^{(\nu)}}{e+1} \ , \tag{2.1.13}$$

$$\mathbb{V}ar\left[h_f^{(t)(\nu+1)}\right]_{e+1} = \frac{e\mathbb{V}ar\left[h_f^{(t)(\nu)}\right]_e + \left(\hat{\sigma}_f^{(\nu)}\right)^2}{e+1} \tag{2.1.14}$$

and what is used at test time is

$$\mathbb{E}\left[h_f^{(t)(\nu)}\right] = \mathbb{E}\left[h_f^{(t)(\nu)}\right] \ , \qquad \mathbb{V}ar\left[h_f^{(t)(\nu)}\right] = \frac{T_{\text{mb}}}{T_{\text{mb}}-1}\mathbb{V}ar\left[h_f^{(t)(\nu)}\right] \ . \tag{2.1.15}$$

so that at test time

$$y_f^{(t)(\nu)} = \gamma_f^{(\nu)} \, \frac{h_f^{(t)(\nu)} - \mathbb{E}[h_f^{(t)(\nu)}]}{\sqrt{\mathbb{V}ar\left[h_f^{(t)(\nu)}\right] + \epsilon}} + \beta_f^{(\nu)} \ . \tag{2.1.16}$$

In practice when using BN, and as advocated in the original paper[18], one can get rid without loss of precision of the former most popular regularization technique before BN introduction: Dropout[33]. We adopt this convention in the following, as our test experienced a loss of precision with the joint use of BN and dropout. This might be due to the peculiarity of our problem which is a regression and not a classification task. To the best of our knowledge, the litterature on NN for regression task is pretty scarce. Going back to the FNN structure, this induces the following change to the weight averaging formula of Section 2.1.6: for $\nu \in [\![1, N-1]\!]$

$$a_f^{(t)(\nu)} = \sum_{f'=0}^{F_\nu - 1} \Theta_{f'}^{(\nu)f} y_{f'}^{(t)(\nu-1)} \ . \tag{2.1.17}$$

2.1.10 Architecture considered in practice

Schematically denoting a hidden fully-connected unit as (with the WA/NLA/BN order advocated in the core of the paper)

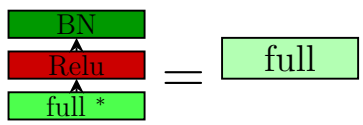

Figure 2.3: A FNN fully connected layer

and the FNN output unit as

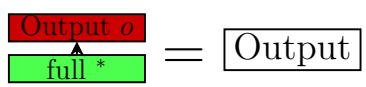

Figure 2.4: The FNN output layer

we consider in the core of our paper the following two FNN architectures. A one hidden layer FNN depicted in Figure 2.5

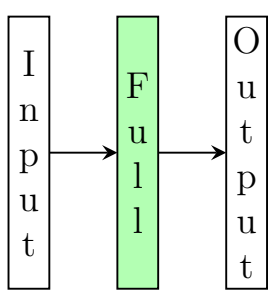

Figure 2.5: FNN with one hidden layer

and a three hidden layer FNN depicted in Figure 2.6

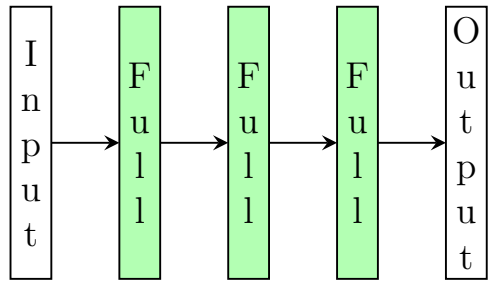

Figure 2.6: FNN with three hidden layers

In practice one could consider a lot of other FNN architectures: Resnet[16], highway Nets[34], DenseNets[17]... This could be the object of future studies.

2.2 Convolutional Neural Networks

Convolutional Neural Network (CNN) is a kind of network architecture particularly adapted to image classification, be it numbers or animal/car/... category. In this Section we review the novelty involved when dealing with CNN when compared to FNN as introduced in Section 2.1, and do so for our regression task at hand. The most fundamental novelties are the two building blocks of CNN: convolution and pooling operations. Before presenting them, let us introduce some more notations specific to the CNN architectures

2.2.1 CNN new specific notations and definitions

In addition or in replacement to the notations introduced in Section 2.1.1, we denote in the following

- F_ν, the number of feature maps in the ν'th layer.
- T_ν and N_ν, respectively the width and the height of the ν'th feature map.
- $h_{flm}^{(t)(\nu)}$ where $\nu \in [\![0, N]\!]$ $f \in [\![0, F_\nu - 1]\!]$, $l \in [\![0, N_\nu - 1]\!]$ and $m \in [\![0, T_\nu - 1]\!]$, the ν'th layer components.
- $X_{flm}^{(t)} = h_{flm}^{(t)(0)}$ where $f \in [\![0, F_0 - 1]\!]$, $l \in [\![0, N_0 - 1]\!]$ and $m \in [\![0, T_0 - 1]\!]$, the input variables.
- $y_{flm}^{(t)}$ where $f \in [\![0, F_N - 1]\!]$, $l \in [\![0, N_N - 1]\!]$ and $m \in [\![0, T_N - 1]\!]$, the output variables (to be predicted).
- $\hat{y}_{flm}^{(t)} = h_{flm}^{(t)(N)}$ where $f \in [\![0, F_N - 1]\!]$, $l \in [\![0, N_N - 1]\!]$ and $m \in [\![0, T_N - 1]\!]$, the output of the CNN.
- $\Theta_{f'lm}^{(\nu)f}$ for $\nu \in [0, N-1]$, $f \in [0, F_{\nu+1} - 1]$, $f' \in [0, F_\nu - 1]$, $l \in [\![0, N_\nu - 1]\!]$ and $m \in [\![0, T_\nu - 1]\!]$, the weight matrices.
- R_C and S_C, respectively, the receptive field and the stride of the convolution operation. Unless stated otherwise, these are kept the same for all the convolutions.
- R_P and S_P, respectively the receptive field and the stride of the pooling operation. Unless stated otherwise, these are kept the same for all the poolings.

2.2.2 CNN architecture

The CNN architecture involves convolutions (see Section 2.2.4), pooling (see Section 2.2.5) as well as an input, an output and fully connected (similar to those a FNN, see Section 2.1.6) layers. Here is a possible CNN architecture : an input is convolved with a first weight matrix then pooled, convolved with a second weight matrix then pooled, then a fully connected operation occurs before the output is computed.

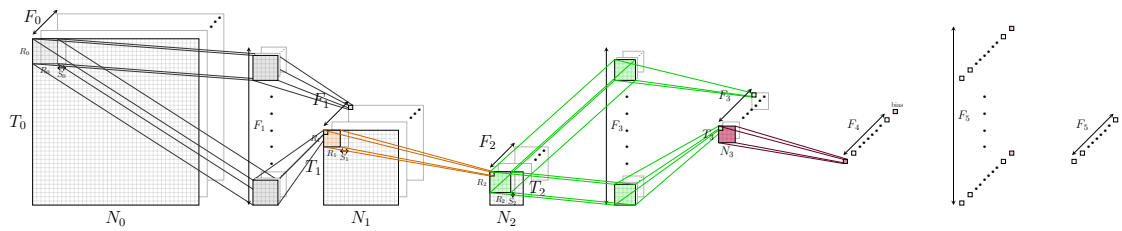

Figure 2.7: A typical CNN architecture (in this case LeNet[23] inspired): convolution operations are followed by pooling operations, until the size of each feature map is reduced to one. Fully connected layers can then be introduced.

The fully connected layers, the output layers and the loss function are unchanged (see Sections 2.1.6, 2.1.7 and 2.1.8). The batch normalization procedure is also used, and can easily be adapted from what has been presented in Section 2.1.9. Let us now see the new and modified layers.

2.2.3 Input layers

As explained in the core of the paper, we only consider the full input for CNN. But for all purposes here, we deal with an $h^{(t)(0)}_{flm} = X^{(t)}_f$ input layer, with $f \in [\![0, F_0 - 1]\!]$, $l \in [\![0, N_0 - 1]\!]$ and $m \in [\![0, T_0 - 1]\!]$.

2.2.4 Convolutional layers

The convolution operation that gives its name to the CNN is the fundamental building block of this type of network. It amounts to convolute a feature map of a hidden layer input with a weight matrix to give rise to an output feature map. The weight is really a four dimensional tensor, one dimension (F_ν) being the number of feature maps of the convolutional input layer, another ($F_{\nu+1}$) the number of feature maps of the convolutional output layer. The two others give the width and the height of the receptive field. The receptive field allows one to convolute a subset instead of the whole input image. It aims at searching for similar patterns in the input image, no matter where the pattern is (translational invariance). We saw in the core of our paper the problem that this induces. The width and the height of the output image are determined by the receptive field as well as by the stride: it is simply the number of pixels by which one slides in the vertical and/or the horizontal direction before applying the convolution operation again. The central convolution formula for the ν'th CNN layer (involving the o'th weight matrix for

the o'th convolution operation) is

$$a^{(t)(\nu)}_{f\,l\,m} = \sum_{f'=0}^{F_\nu-1}\sum_{j=0}^{R_C-1}\sum_{k=0}^{R_C-1} \Theta^{(o)f}_{f'\,j\,k} h^{(t)(\nu)}_{f'\,S_C l+j\,S_C m+k}$$
$$= \left(\Theta^{(o)T} \star h^{(t)(\nu)}\right)_{flm} \ . \tag{2.2.1}$$

Here $S_C l + j$ belongs to $[\![0, N_\nu - 1]\!]$ and $S_C m + k$ to $[\![0, T_\nu - 1]\!]$. This implies the following relation between $N_{\nu+1}$, N_ν and $T_{\nu+1}$, T_ν

$$N_{\nu+1} = \frac{N_\nu - R_C}{S_C} + 1 \ , \qquad T_{\nu+1} = \frac{T_\nu - R_C}{S_C} + 1 \ . \tag{2.2.2}$$

One then computes the hidden units via the ReLU activation function g introduced in Section 2.1.4

$$h^{(t)(\nu+1)}_{f\,l\,m} = g\left(a^{(t)(\nu)}_{f\,l\,m}\right) \ . \tag{2.2.3}$$

2.2.5 Pooling layers

The pooling operation, less and less used in the current state of the art CNN[16], is fundamentally a dimension reduction step. It amounts to take the maximum of a sub-image (characterized by the pooling receptive field R_P and a stride S_P) of the input feature map F_ν, to obtain an output feature map $F_{\nu+1} = F_\nu$ of width and height

$$N_{\nu+1} = \frac{N_\nu - R_P}{S_P} + 1 < N_\nu \ , \qquad T_{\nu+1} = \frac{T_\nu - R_P}{S_P} + 1 < T_\nu \ . \tag{2.2.4}$$

The max pooling procedure (That we use here instead of the other possible choice, the average pooling) reads for any ν

$$a^{(t)(\nu)}_{f\,l\,m} = \max_{j,k=[\![0,R_P-1]\!]} h^{(t)(\nu)}_{f\,S_P l+j\,S_P m+k} \ . \tag{2.2.5}$$

The hidden unit is then just

$$h^{(t)(\nu+1)}_{f\,l\,m} = a^{(t)(\nu)}_{f\,l\,m} \ . \tag{2.2.6}$$

2.2.6 Architecture considered in practice

Schematically denoting a hidden Convolution unit as

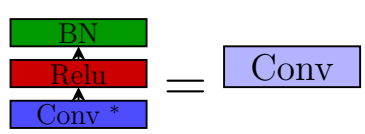

Figure 2.8: The structure of a convolution layer

we consider in the remaining of this paper the following CNN architecture: the so-called VGG CNN[31], which was the 2014 state of the art convolutional network

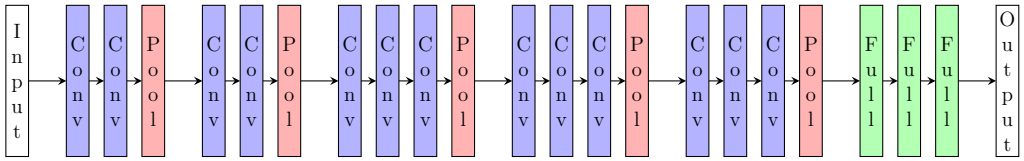

Figure 2.9: The structure of the VGG CNN.

All the other details of the model are in the core of the paper.

2.3 Recurrent Neural Networks - Long Short Term Memory

In this Section, we review the third kind of Neural Network architecture used in this paper: Recurrent Neural Networks. The specificity of this kind of network is that the time dependency of the data is built into the model. We briefly present the first Recurrent Neural Network (RNN) architecture, as well as the current most popular one: the Long Short Term Memory (LSTM) Neural Network. We use the latter in this study.

2.3.1 RNN new specific notations and definitions

In contrast to the previously discussed neural networks, where we defined

$$\begin{aligned} a_f^{(t)(\nu)} &= \text{Weight Averaging } \left(h_f^{(t)(\nu)}\right) , \\ h_f^{(t)(\nu+1)} &= \text{Activation function } \left(a_f^{(t)(\nu)}\right) , \end{aligned} \tag{2.3.1}$$

we now have hidden layers that are indexed by both a "spatial" and a "temporal" index, and the general philosophy of the RNN is (now the a is usually characterized by a c for cell state, this denotation, trivial for the basic RNN architecture will make more sense when we talk about LSTM networks)

$$\begin{aligned} c_f^{(t)(\nu\tau)} &= \text{Weight Averaging } \left(h_f^{(t)(\nu\tau-1)}, h_f^{(t)(\nu-1\tau)}\right) , \\ h_f^{(t)(\nu\tau)} &= \text{Activation function } \left(c_f^{(t)(\nu\tau)}\right) . \end{aligned} \tag{2.3.2}$$

Here, the notations are

- N, the number of layers (not counting the input) in the spatial direction.
- T, the number of layers (not counting the first one) in the temporal direction (so that $T = 0$ in a RNN corresponds to a standard FNN).
- $h_f^{(t)(\nu\tau)}$, where $\nu \in [\![0, N]\!]$, $\tau \in [\![0, T]\!]$ and $f \in [\![0, F_\nu - 1]\!]$, the $\nu\tau$'th layer components.
- $X_f^{(t)(\tau)} = h_f^{(t)(0\tau)}$, where $f \in [\![0, F_0 - 1]\!]$ and $\tau \in [\![0, T]\!]$, the input variables (more on that in Section 2.3.5).
- $y_f^{(t)(\tau)}$, where $f \in [\![0, F_N - 1]\!]$, the output variables (to be predicted).
- $\hat{y}_f^{(t)(\tau)}$, where $f \in [\![0, F_N - 1]\!]$, the output of the RNN.

- $\Theta^{\nu(\nu)f}_{f'}$ for $(\nu) \in [0, N-1]$, $f \in [0, F_\nu - 1]$, $f' \in [0, F_{\nu+1} - 1]$, the weight matrices in the "spatial" direction of the RNN.
- $\Theta^{\tau(\nu)f}_{f'}$ for $(\nu) \in [1, N-1]$, $f \in [0, F_\nu - 1]$, $f' \in [0, F_{\nu+1} - 1]$, the weight matrices in the "temporal" direction of the RNN.

2.3.2 RNN architecture

An example of a RNN architecture with $N = 4$ spatial and $T = 7$ temporal layers is depicted in Figure 2.10

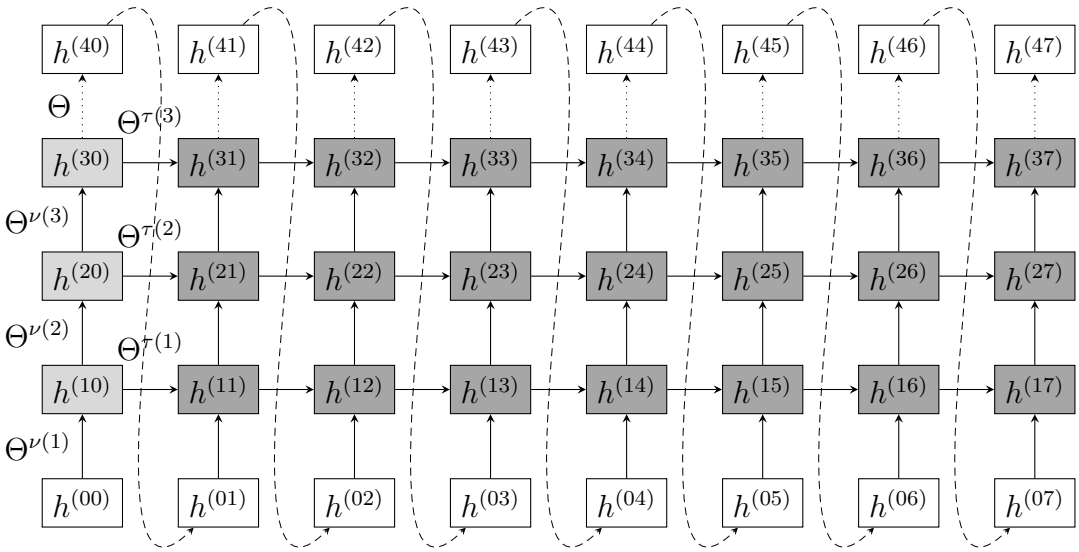

Figure 2.10: RNN architecture, with data propagating both in "space" and in "time". In our example, the temporal dimension is of size $T = 7$ while the spatial one is of size $N = 4$.

Note that the weight matrices do not vary along the temporal direction, and this RNN only has $2(N-1)+1 = 7$ weight matrices (indicated in Figure 2.10).

2.3.3 RNN hidden layer

The FNN formula of Section 2.1.6 is replaced in a RNN by (note also the change in the activation function, a standard choice in the RNN litterature[12])

$$h_f^{(t)(\nu\tau)} = \tanh\left(\sum_{f'=0}^{F_{\nu-1}-1} \Theta^{\nu(\nu)f}_{f'} h_{f'}^{(t)(\nu-1\tau)} + \sum_{f'=0}^{F_\nu-1} \Theta^{\tau(\nu)f}_{f'} h_{f'}^{(t)(\nu\tau-1)}\right) . \tag{2.3.3}$$

As this network has fallen out of favor because of vanishing gradient issues in backpropagation[XX], we now turn our attention to the LSTM RNN architecture that we use in the core of the paper.

2.3.4 LSTM hidden layer

In a Long Short Term Memory Neural Network[10], the state of a given unit is not directly determined by its immediate spatial and temporal neighbours. Instead, a cell state is updated

for each hidden unit, and the output of this unit is a probe of the cell state. Several gates are introduced in the process: the input gate $i_f^{(t)(\nu\tau)}$ determines if we allow new information $g_f^{(t)(\nu\tau)}$ to enter into the cell state. The output gate $o_f^{(t)(\nu\tau)}$ determines if we set or not the output hidden value to 0, or really probe the current cell state. Finally, the forget state $f_f^{(t)(\nu\tau)}$ determines if we forget or not the past cell state. All these concepts are illustrated in Figure 2.11

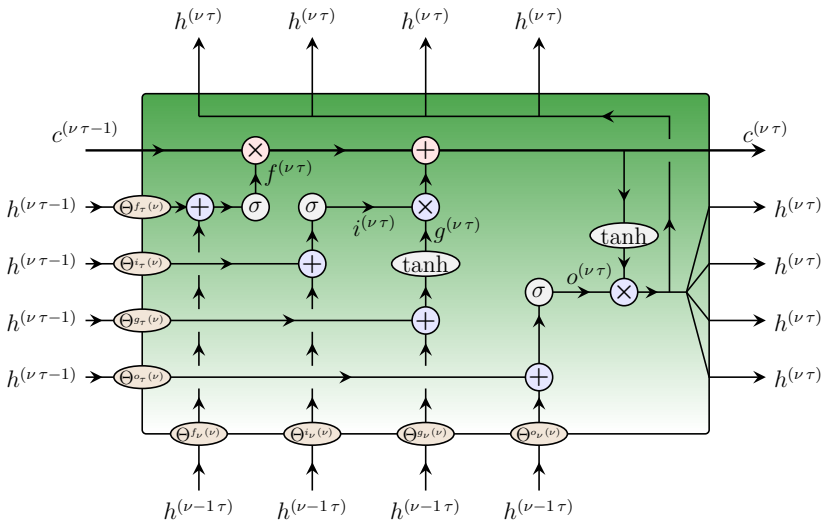

Figure 2.11: LSTM hidden unit details

Considering all the $\tau - 1$ variable values to be 0 when $\tau = 0$, we get the following formula for the input, forget and output gates:

$$(i, f, o)_f^{(t)(\nu\tau)} = \sigma\left(\sum_{f'=0}^{F_{\nu-1}-1} \Theta_{f'}^{(i,f,o)_\nu(\nu)f} h_{f'}^{(t)(\nu-1\tau)} + \sum_{f'=0}^{F_\nu-1} \Theta_{f'}^{i_\tau(\nu)f} h_{f'}^{(t)(\nu\tau-1)}\right) . \tag{2.3.4}$$

The use of the sigmoid function is the reason why the functions i, f, o are called gates. Indeed, they are valued in $]0, 1[$. Therefore, it allows or forbids information to pass through the next step. The cell state update is then performed in the following way

$$g_f^{(t)(\nu\tau)} = \tanh\left(\sum_{f'=0}^{F_{\nu-1}-1} \Theta_{f'}^{g_\nu(\nu)f} h_{f'}^{(t)(\nu-1\tau)} + \sum_{f'=0}^{F_\nu-1} \Theta_{f'}^{g_\tau(\nu)f} h_{f'}^{(t)(\nu\tau-1)}\right) , \tag{2.3.5}$$

and

$$c_f^{(t)(\nu\tau)} = f_f^{(t)(\nu\tau)} c_f^{(t)(\nu\tau-1)} + i_f^{(t)(\nu\tau)} g_f^{(t)(\nu\tau)} . \tag{2.3.6}$$

So that, as announced previously, hidden state update is just a probe of the current cell state

$$h_f^{(t)(\nu\tau)} = o_f^{(t)(\nu\tau)} \tanh\left(c_f^{(t)(\nu\tau)}\right) . \tag{2.3.7}$$

There is a whole zoology of LSTM variations[13], amounting to also include the cell state in the input (this is called peephole connections), forget and output update, removing some of the gates... In this paper we choose to stick with the standard LSTM formulation.

2.3.5 RNN-LSTM Input Layer

As explained in the core of the paper, we consider a full and a reduced input. But for all purposes here, we deal with an $h_f^{(t)(0\tau)} = X_f^{(t)(\tau)}$ input layer, with $f \in [\![0, F_0 - 1]\!]$ and $t \in [\![0, T_{\text{mb}} - 1]\!]$ and $\tau \in [\![0, T]\!]$.

2.3.6 RNN-LSTM Output Layer

The output layer of a RNN-LSTM is similar in spirit to the FNN one and reads

$$h_f^{(t)(N\tau)} = o\left(\sum_{f'=0}^{F_{N-1}-1} \Theta_{f'}^{f} h_f^{(t)(N-1\tau)}\right) , \tag{2.3.8}$$

where the output function o is the one described in Section 2.1.7.

2.3.7 Change to the Loss function

Since the structure of the data has changed compared to a FNN/CNN, here is how the loss function reads for a regression task in a RNN-LSTM framework

$$J(\Theta) = \frac{1}{2T_{\text{mb}}} \sum_{t=0}^{T_{\text{mb}}-1} \sum_{\tau=0}^{T-1} \sum_{f=0}^{F_N-1} \left(h_f^{(t)(N\tau)} - y_f^{(t)(\tau)}\right)^2 . \tag{2.3.9}$$

2.3.8 Architecture considered in practice

In the core of the paper, we consider a $N = 2$ (hence one hidden layer) RNN-LSTM architecture, adopting the $T = 20$ size to the temporal size $\mathcal{H}_{rf}$ of the full/reduced output (see the core of the paper).

Acknowledgements

We acknowledge support from Mediamobile. We thank Philippe Goudal for sparking our interest in the topic. We thank Gaspard Monge program and EDF'lab Paris Saclay for Research initiative in industrial data science's funding.

Bibliography

[1] F. Agostinelli, M. Hoffman, P. Sadowski, and P. Baldi. Learning activation functions to improve deep neural networks. *arXiv preprint arXiv:1412.6830*, 2014.

[2] R. Caruana, S. Lawrence, and C. L. Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In *Advances in neural information processing systems*, pages 402–408, 2001.

[3] A. Choromanska, Y. LeCun, and G. B. Arous. Open problem: The landscape of the loss surfaces of multilayer networks. In P. GrÃ$\frac{1}{4}$nwald, E. Hazan, and S. Kale, editors, *Proceedings of The 28th Conference on Learning Theory*, volume 40 of *Proceedings of Machine Learning Research*, pages 1756–1760, Paris, France, 03–06 Jul 2015. PMLR.

[4] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *CoRR*, abs/1511.07289, 2015.

[5] D. Deng, C. Shahabi, U. Demiryurek, L. Zhu, R. Yu, and Y. Liu. Latent space model for road networks to predict time-varying traffic. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 1525–1534, New York, NY, USA, 2016. ACM.

[6] T. Epelbaum. *Deep Learning: Technical Introduction.* arXiv, 2017.

[7] J. Fazio, B. N. Wiesner, and M. D. Deardoff. Estimation of free-flow speed. *KSCE Journal of Civil Engineering*, 18(2):646–650, Mar 2014.

[8] B. Fleischmann, S. Gnutzmann, and E. Sandvoß. Dynamic vehicle routing based on online traffic information. *Transportation science*, 38(4):420–433, 2004.

[9] M. Fouladgar, M. Parchami, R. Elmasri, and A. Ghaderi. Scalable deep traffic flow neural networks for urban traffic congestion prediction. *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2251–2258, 2017.

[10] F. A. Gers, J. A. Schmidhuber, and F. A. Cummins. Learning to forget: Continual prediction with lstm. *Neural Comput.*, 12(10):2451–2471, Oct. 2000.

[11] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh and M. Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[12] A. Graves. *Supervised Sequence Labelling with Recurrent Neural Networks.* 2011.

[13] K. Greff, R. K. Srivastava, J. Koutnık, B. R. Steunebrink, and J. Schmidhuber. LSTM: A search space odyssey.

[14] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan. Deep learning with limited numerical precision. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1737–1746, 2015.

[15] R. Hahnloser, R. Sarpeshkar, M. Mahowald, R. J. Douglas, and S. Seung. Digital selection and analog amplification co-exist in an electronic circuit inspired by neocortex. *Nature*, 405:947–951,, 06 2000.

[16] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. 7, 12 2015.

[17] G. Huang, Z. Liu, L. Van De Maaten, and K. Weinberger. Densely connected convolutional networks. 07 2017.

[18] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. 02 2015.

[19] R. Johnson and T. Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in neural information processing systems*, pages 315–323, 2013.

[20] D. Kingma and J. Ba. Adam: A method for stochastic optimization. 12 2014.

[21] Y. LeCun, Y. Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[22] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[23] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[24] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Effiicient backprop. In *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, pages 9–50, London, UK, UK, 1998. Springer-Verlag.

[25] M. Lippi, M. Bertini, and P. Frasconi. Short-term traffic flow forecasting: An experimental comparison of time-series analysis and supervised learning. *IEEE Transactions on Intelligent Transportation Systems*, 14(2):871–882, June 2013.

[26] J.-M. Loubes, É. Maza, M. Lavielle, and L. Rodriguez. Road trafficking description and short term travel time forecasting, with a classification method. *Canadian Journal of Statistics*, 34(3):475–491, 2006.

[27] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3, 2010.

[28] T. Miwa, Y. Tawada, T. Yamamoto, and T. Morikawa. En-route updating methodology of travel time prediction using accumulated probe-car data. Proc. of the 11th ITS World Congress, 2004.

[29] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

[30] L. Sagun, V. Ugur Guney, and Y. Lecun. Explorations on high dimensional landscapes. 12 2014.

[31] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[32] R. Socher, J. Bauer, C. D. Manning, and A. Y. Ng. Parsing With Compositional Vector Grammars. In *ACL*. 2013.

[33] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, Jan. 2014.

[34] R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway Networks.

[35] H. Sun, H. X. Liu, H. Xiao, and B. Ran. Short term traffic forecasting using the local linear regression model.

[36] S. Turksma. The various uses of floating car data. *IET Conference Proceedings*, pages 51–55(4), January 2000.

[37] M. Xiaolei, D. Zhuang, H. Zhengbing, M. Jihui, W. Yong, and W. Yunpeng. Learning traffic as images: A deep convolutional neural network for large-scale transportation network speed prediction. 2017.

[38] S. Xingjian, 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 *Advances in neural information processing systems*, pages 802–810, 2015.

[39] X. Yuanchang, Z. Kaiguang, S. Ying, and C. Dawei. Gaussian processes for short-term traffic volume forecasting. *Transportation Research Board of the National Academies*, 2165(-1):69–78, 2010.

[40] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[41] H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, 2005.