

Prediction of taxi destination trips based on initial partial trajectories

Francesco Brundu

cganimation@gmail.com

AA2 Final Project (A.A. 2014-15)

July 2015

1 Introduction

The taxi industry is evolving rapidly. New competitors and technologies are changing the way traditional taxi services do business.

One major shift is the widespread adoption of electronic dispatch systems that have replaced the VHF-radio dispatch systems of times past. These mobile data terminals are installed in each vehicle and typically provide information on GPS localization and taximeter state. Electronic dispatch systems make it easy to see where a taxi has been, but not necessarily where it is going. In most cases, taxi drivers operating with an electronic dispatch system do not indicate the final destination of their current ride.

The spatial trajectory of an occupied taxi could provide some hints as to where it is going.

For this challenge, we build a predictive framework exploiting the echo state networks that is able to infer the final destination of taxi rides in Porto, Portugal based on their (initial) partial trajectories. The output of such a framework will be the final trip's destination (WGS84 coordinates).

This competition is affiliated with the organization of [ECML/PKDD 2015](#).

The full description of the problem can be found at www.kaggle.com/c/pkdd-15-predict-taxi-service-trajectory-i

1.1 The Dataset

We have an accurate dataset describing a complete year (from 01/07/2013 to 30/06/2014) of the trajectories for all the 442 taxis running in the city of Porto, in Portugal (i.e. one CSV file named "train.csv").

Each data sample corresponds to one completed trip. It contains a total of 9 (nine) features, of which we use only:

1. **TRIP_ID**: (String) It contains an unique identifier for each trip;
2. **POLYLINE**: (String): It contains a list of GPS coordinates (i.e. WGS84 format) mapped as a string. The beginning and the end of the string are identified with brackets (i.e. [and], respectively). Each pair of coordinates is also identified by the same brackets as [LONGITUDE, LATITUDE]. This list contains one pair of coordinates for each 15 seconds of trip. The last list item corresponds to the trip's destination while the first one represents its start.

A distinct test-set, with no target is provided for the submission of the model for the competition.

The full description can be found at www.kaggle.com/c/pkdd-15-predict-taxi-service-trajectory-i/data

1.2 Evaluation

The evaluation metric for this competition is the **Mean Haversine Distance**. The Haversine Distance is commonly used in navigation, it measures distances between two points on a sphere based on their latitude and longitude. Between two locations it can be computed as follows:

$$a = \sin^2 \left(\frac{\phi_2 - \phi_1}{2} \right) + \cos(\phi_1) \cos(\phi_2) \sin^2 \left(\frac{\lambda_2 - \lambda_1}{2} \right) \quad (1)$$
$$d = 2 \cdot r \cdot \operatorname{atan} \left(\sqrt{\frac{a}{1-a}} \right)$$

Where ϕ is the latitude, λ is the longitude, d is the distance between two points, and r is the sphere's radius, in our case, it should be replaced by the Earth's radius in the desired metric (e.g., 6371 km).

More about evaluation can be found at www.kaggle.com/c/pkdd-15-predict-taxi-service-trajectory-i/details/evaluation

2 Development environment and the implemented model

We used Python and SciPy (<http://www.scipy.org/>) which is a Python-based ecosystem of open-source software for mathematics, science, and engineering.

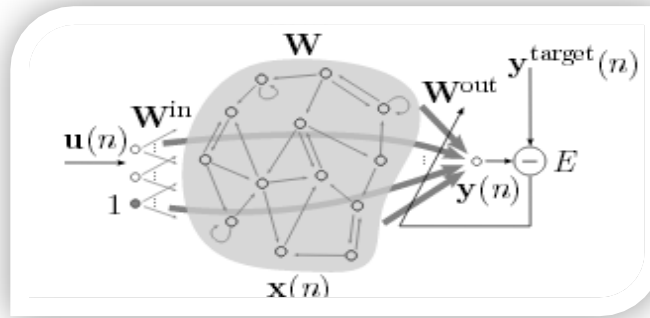
In particular, these are some of the core packages used:

- Numpy, Base N-dimensional array package.
- Scipy, for the linear algebra methods and procedures.
- Pandas, Data structures & analysis.
- Matplotlib, for the charts.

2.1 The Model

Given the intrinsic Markovianity of the trajectory where trips sharing common suffix lead to near or same destinations we can choose to use the Echo state Networks.

The specific architecture has one bias unit and direct input-to-readout connections.



ESNs use an RNN type with leaky-integrated discrete-time continuous-value units. The typical update equations are (as reported in [5])

$$\tilde{x}(n) = \tanh(W_{in} [1; u(n)] + W x(n - 1)) \quad (2)$$

$$x(n) = (1 - \alpha)x(n - 1) + \alpha \tilde{x}(n) \quad (3)$$

Where $x(n) \in R^{Nx}$ is a vector of reservoir neuron activations and \tilde{x} is its update, all at time step n , $\tanh(\cdot)$ is applied element-wise, $[\cdot; \cdot]$ stands for a vertical vector (or matrix) concatenation and 1 represents the bias input, $Win \in R^{Nx \times (1+Nu)}$ and $W \in R^{Nx \times Nx}$ are the input and recurrent weight matrices respectively, and $\alpha \in (0, 1]$ is the leaking rate. Other sigmoid wrappers can be used besides the tanh, which however is the most common choice.

The model is also sometimes used without the leaky integration, which is a special case of $\alpha = 1$ and thus $\tilde{x}(n) \equiv x(n)$.

The linear readout layer (output) is computed according to:

$$y(n) = W_{out} [1; u(n); x(n)] \quad (4)$$

Where $y(n) \in R^{Ny}$ is network output, $W_{out} \in R^{Ny \times 1+Nu+Nx}$ is the output weight matrix, and $[\cdot; \cdot]$ again stands for a vertical vector (or matrix) concatenation.

The training of W_{out} is done with the ridge regression:

$$W_{out} = Y_{target} X^T (X X^T + \lambda I)^{-1} \quad (5)$$

Given T the total number of target collected $Y_{target} \in R^{Ny \times T}$ and $X \in R^{1+Nu+Nx \times T}$, X for notational simplicity represents $[1; U; X]$, this takes into account the connections to the readout of the bias (1), the input (u) and the reservoir (x); They all contribute to the output so they must be collected as x .

Training phase

For each training sequence

- Run network with teacher input. Start with the network state $x(0)=0$, dismiss the initial transient (washout) and update the network with the training inputs $u(1) \dots u(N)$.
- Collect the reservoir states $x(n)$ into X and target values $y_{target}(n)$ into Y_{target} for each time stamp n

- The washout is to be intended as a fraction of each trajectory. I.e. $0.2 = 20\%$ of the trip points to be used for washout.

Then compute the output weights W_{out} by (5).

When dealing with a lot of data instead of collecting all the states, the matrices $Y_{target}X^T$ and XX^T can be computed incrementally, one pattern at a time. The cost increases because for every pattern needs a matrix addition and an outer product, but then when computing W_{out} we already have the matrix product computed.

$$Y_{target}X^T += y_{target}(n) \times x(n)^T \quad (6)$$

$$XX^T += x(n) \times x(n)^T \quad (7)$$

$Y_{target}X^T$ and XX^T are respectively of dimension $(N_y \times N_x)$ and $(N_x \times N_x)$. If we consider the architecture described so far and connect also the input and the bias x rewrites as $[1; u; x]$ and N_x becomes $(1+N_u+N_x)$.

Testing phase

For each test sequence

- Start with the network state $x(0)=0$, dismiss the initial transient (washout) and compute the predicted output with (4).
- Collect the predicted outputs and the target values for each sequence, if you want to track the behaviour you can collect the predicted output also as the sequence get visited.
- Recall that during the prediction the input is taken accordingly to the net prediction mode: generative/predictive.

2.2 Parameter selection and evaluation

The parameters selected to be optimized with a grid search are:

- N_r , the reservoir size
- ρ or σ , the spectral radius or the largest singular value
- α the leaking rate
- λ , the ridge regression regularization parameter

Each combination of the parameter defines a model, which in turn is evaluated on the validation set. A more accurate evaluation can be obtained doing a cross-validation (ie. with a 5-10 fold CV) and taking the model with the least mean validation error over all the folds.

Another limitation is that the random generator seed is fixed; a full grid search should additionally be done starting from different seeds, in this way different network weights can be generated.

2.3 Code structure

The code is structured into distinct files:

- `Esn_class.py`
- `Esn_loaddata.py`
- `Esn_mackleyglass.py`
- `EsnModelSelection.py`
- `esnTrainAndTestUtils.py`
- `utils.py`

The following section briefly described the main functionalities implemented.

The ***Esn_class.py*** has general purpose methods which do not rely on a given task:

- Constructor: given the net parameters it allocates and initializes the matrices W_{in} and W . It also scales W such that the largest singular value, $lsv(W)$, or $\rho(W)$ assumes the requested value.

- Fit: data fitting, it updates the net and builds the X matrix collecting the input and the states.
- Trainwout: computes Wout by (5) given X and the associated target values Ytarget.
- Predict: predicts the output of a given sequence.

Notes:

- The lsv(W) is computed with the 2norm(W) instead $\rho(W)$ is the eigenvalue of max module.
- The random generator seed is fixed, so every test is coherent, but it can also be updated.
- The connectivity factor is of 100% by default if not specified.

esnTrainAndTestUtils.py contains some methods for the taxi task, the functions use specific conventions, such as specific columns in the dataframe and predict wrt this task:

- trainEsn, build a new esn initialize it wrt the given parameters, fit the training data and compute the wout. It returns the trained esn.
- fitEsn, given the training data and a initialized esn it applies the net to the data and collect the activation states. Returns the collected activation states.
- applyEsn, given a trained esn, applies it to the test data and for each trajectory put the predicted result into two columns respectively LATITUDE and LONGITUDE appended to the dataframe.
- applyAndMhd, use applyEsn and manages the result, denormalizing the lat and long columns, extract the targets from the data trips, denormalize them and then computes the MHD score with the function provided into the utilities meanHaversineDistance
- applyAndSubmit, it loads the submission test dataset, normalize the trajectories, then similarly to applyAndMhd it apply the esn, denormalizes the results and with doSubmission generate the csv

EsnModelSelection.py is optimized for the taxi task:

- Given the search grid by means of the parameters lists, the data for the training and the data for the validation it do a grid search training only when it is necessary and evaluates the net performances on the training and the validation set.
- The heaviest operation is the training fitting which collect the activations of the net, it uses all the params except the lambda.
- The model selection is optimized by means of using the same trained net and varying the lambda parameter, this drastically reduces the cost of trying more regularization.
- Another optimization is to not build a new net and reinitialize all the matrices computing the eigenvalues which is also computational expensive, when looping in the leaking parameter, but the data fit cannot be avoided
- This procedure collects all the results of the grid search, by means of the train error and the test error for each combination of the parameters. This can be used for a graphical view of the effects induced by the parameter change on the task.

The main file is *taxitrainer.py* and the data loading and preprocessing is done in *esn_loaddata.py*

A miscellaneous of methods can be found into *utilities.py*, methods to:

- compute the mean Haversine Distance and the mean squared error
- min-max normalization and denormalization
- write the csv file with the format 'TRIP_ID', 'LATITUDE', 'LONGITUDE' for the submission.

For the benchmark task the main script file is **esn_trainer** which contains a straightforward implementation of the data loading and the parameters selection by means of the implemented esn class and plots the results.

3 Experimental Results

This section describes the benchmark task used for testing the esn implementation and then the taxi destination prediction task.

3.1 Benchmark

As a benchmarks we solve a classical task of learning to generate/predict a chaotic attractor, the Mackey-Glass ($\tau=17$) time series (mackey_glasst17.txt).

The training is done with a washout of 100 steps.

After the training the ESN is tested in a generative mode for 500 and 1000 time steps.

This is done starting from the state reached by the training (like 2000 step washout) and then letting it run freely in a self-feeding *generative* mode.

The parameter selection is done with a grid search, the grid is:

- $Nr = [1000, 500, 100, 50]$
- $\rho = [0.8, 0.90, 0.99, 1.25, 1.50]$
- $a = [0.1, 0.4, 0.7, 1]$
- $\lambda = [1e-8, 1e-6, 1e-4, 0.01]$

The parameters exploiting the best validation error are:

nr	rho	lambda	a	Conn %
500	1.25	1e-06	0.7	30

Table 1. Winning parameters.

The achieved performances with the best parameters are:

- mse: 5.087e-06 for a 500 time steps free running prediction
- mse: 9.41e-04 for a 1000 time steps free running prediction

The plot shows the prediction result, noticing as already stated by the mse that the predicted signal starts to slightly deviate after roughly 600 time steps, the net is able to maintain the learned dynamics.

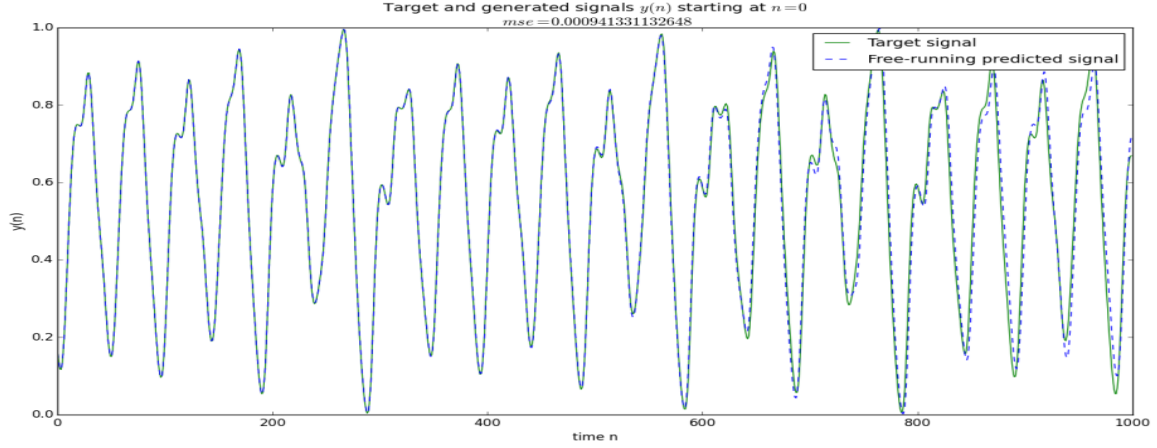


Figure 1. Target and predicted signal.

Now we can also observe the plot of the weight matrix W_{out} . It has small weights which in general is good and do not saturate the units.

Plotting also some reservoir activation levels during the prediction of 200 steps of the sequence we can see that the dynamics speed is of the same degree of the target sequence, and the units do not get saturated (values near ± 1). Not coherent dynamics can be a sign of a wrong choice of ρ .

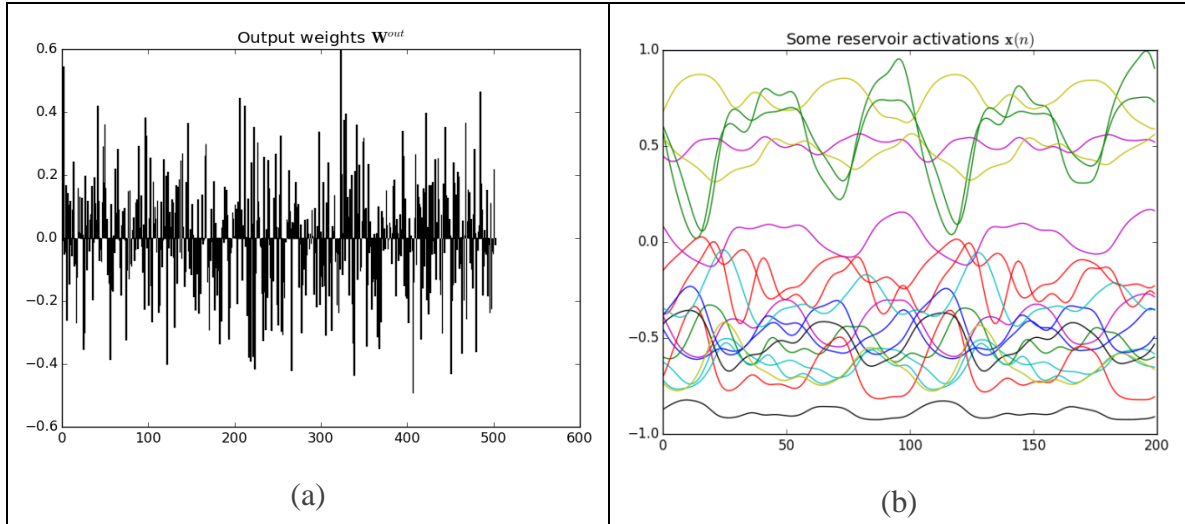


Figure 2. Output weights magnitude (a) and some activations (b)

3.2 Taxi prediction task

The taxi dataset contains 1,710,670 trips, from 01/07/2013 to 01/07/2013, some of them are empty or with missing values, this noise is handled removing the empty trajectories, some missing values does not influence the regression.

Some statistics about the trajectory length are:

Trips length stats	Min	Max	Mean	Median	Variance	Std
	0	2,516	47.9259984142	41	1,934.639	43.984

Table 3. Taxi rides length statistics.

Approximately the first 2,500 trips are relative to the first day, we use them for the training and the following 5,000 for the validation test. They are few but can give us an approximation of how much the model is able to handle them.

Figure 3.

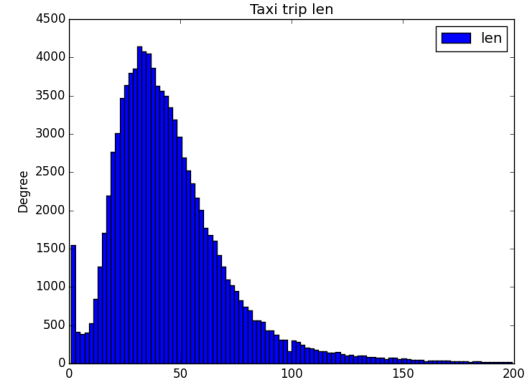


Table 2. Taxi rides length distribution.

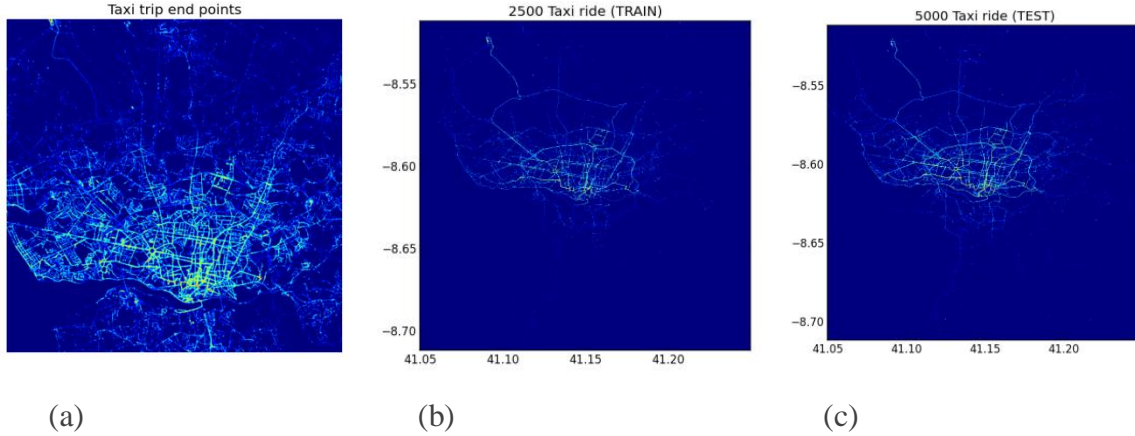


Figure 3. All the Dataset trajectories (a), the training set of 2,500 trips (b) and the following 5,000 for the validation test (c).

For the task we use only the trajectory polyline of each trip, discard the other features and the empty trajectory.

The polylines are normalized between 0-1 with a min-max Normalization; alternatively the Z-score normalization can be used.

The target is the last point of the trajectory and it is assigned as target for every point. So the net is trained to predict the destination of every prefix trajectory, this is obtained by simply collecting into X all the activations, already computed, of a give trip.

3.3 Network preparation and parameter selection

For this task the net is set up in a prediction mode. Some simple generative experiments have also been made with a negative result; the learned dynamics got the net quickly to diverge.

The training of the network for each new trajectory starts from the null state $x(0) = 0$. Initially we made some experiments with no transient and then dismissing the first 20% of the trajectory.

In this task the transient is useful to filter out the first states from the training, which are mostly not meaningful and can moreover have a negative impact in the training as shown in the experiments, indeed a slightly better performance has been achieved with the washout. The results shown below are relative to the experiment with the transient dismissed. Careful should be taken when setting the transient proportion because one too large, this choice influence the ability of the net to “recognise the destination” of a taxi ride from the very beginning, which itself is very hard.

Parameter selection

The parameters selection is done by a grid search which minimizes the mean Haversine distance over the validation set.

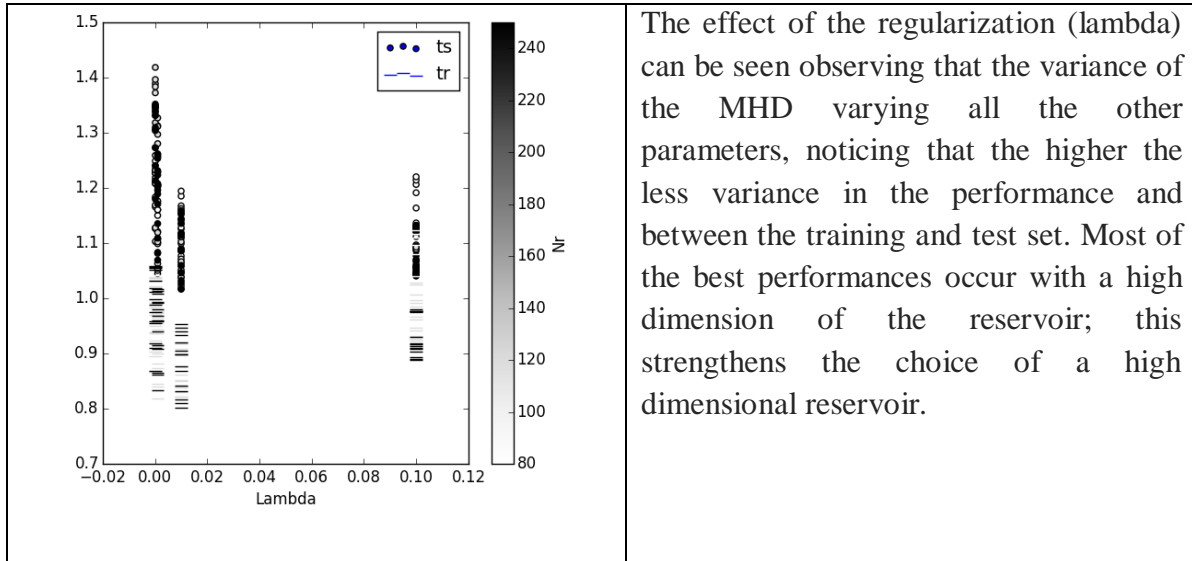
The grid search is manually tuned by adjusting the search window if a border value has been chosen or eventually increasing the granularity if a middle value is selected.

The reservoir matrix W is scaled w.r.t. the lsv, it is more efficient to be computed and into a grid search is “similar” to a shifted grid for ρ , given that for square matrices $\rho(W) < \sigma = \text{lsv}(W)$.

The final grid is:

- $N_r = [80, 120, 250]$
- $\sigma = [0.4, 0.6, 0.8, 0.9, 1]$
- $a = [0.4, 0.8, 1]$
- $\lambda = [1e-4, 1e-3, 1e-2, 0.1]$

The parameter selection results can be also analysed visually to see what’s going on and how the performance changes varying the parameters, table 4.



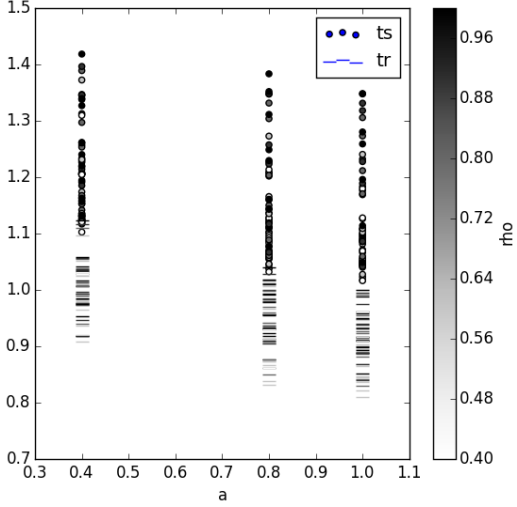
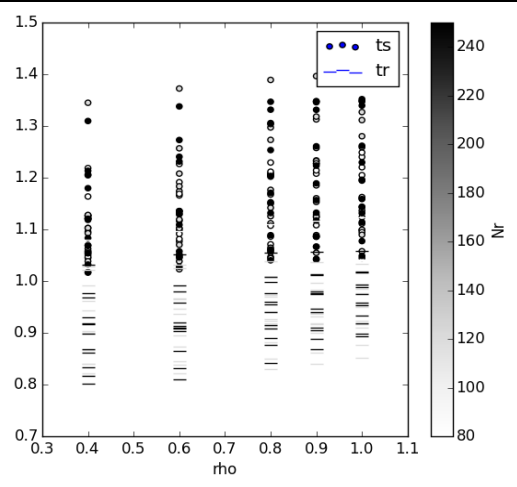
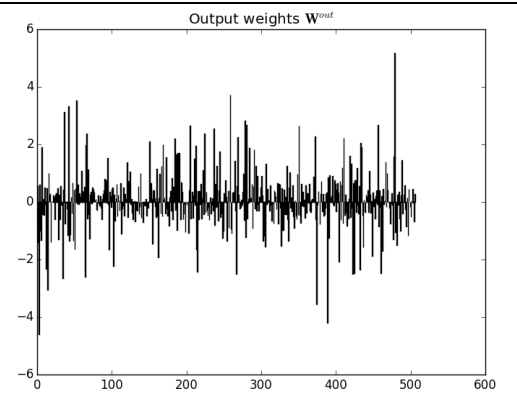
 <p>A scatter plot showing the relationship between parameter a (x-axis, ranging from 0.3 to 1.1) and ρ (y-axis, ranging from 0.7 to 1.5). The plot includes two data series: 'ts' (blue dots) and 'tr' (blue line). A color bar on the right indicates the value of ρ, ranging from 0.40 to 0.96. The data points are clustered around $a \approx 0.4$ and $a \approx 0.8$.</p>	<p>The leaking rate of the reservoir nodes can be regarded as the speed of the reservoir update dynamics discretized in time. Here is set to 1 inducing fast dynamics. This can also be interpreted by the relatively small ride length, which can be wrongly influenced from previous rides still “into the net memory”, indeed also a small ρ is chosen.</p>
 <p>A scatter plot showing the relationship between parameter ρ (x-axis, ranging from 0.3 to 1.1) and ρ (y-axis, ranging from 0.7 to 1.5). The plot includes two data series: 'ts' (blue dots) and 'tr' (blue line). A color bar on the right indicates the value of N_r, ranging from 80 to 240. The data points are clustered around $\rho \approx 0.4$ and $\rho \approx 0.8$.</p>	<p>Generally, the closer λ_{\max} is to unity, the slower is the decay of the network's response to an impulse input. So ρ is the parameter which influences the strength of the memory, the trips are relatively short so a quick changing memory is good. Indeed the downside of a long memory is to take into account the past trajectories for the current prediction.</p>
 <p>A line plot titled "Output weights W_{out}" showing the output weights over time (x-axis, ranging from 0 to 600). The y-axis ranges from -6 to 6. The plot shows a highly oscillatory signal, indicating that the output weights are relatively small and fluctuate rapidly.</p>	<p>The weights are relatively small; the solution found is correctly regularized. Indeed large weights reveal that W_{out} exploits and amplifies tiny differences among the dimensions of $x(n)$, and can be very sensitive to deviations from the exact conditions in which the network has been trained [5], which is a situation we want to avoid.</p>

Table 4. Graphical analysis of the parameter optimization solutions found. In the charts the y axis shows the MHD, the lower the better.

Final results

The best parameters found by the grid search are:

Nr	sigma	lambda	Leak rate	Connectivity %
250	0.4	0.01	1	30

Table 5. The winning parameters.

The winning model gives a

- validation error of km 1.016 (MHD), and a
- train error of km 0.801 (MHD).

It is trained with the first 2,500 trips and tested over the 5,000 following ones.

We can also see how the network predicts the destination as the taxi ride goes on, the green points are the predicted destination as we go through the real trajectory (red points). The X stands for the predicted final destination of which we evaluate the MHD.

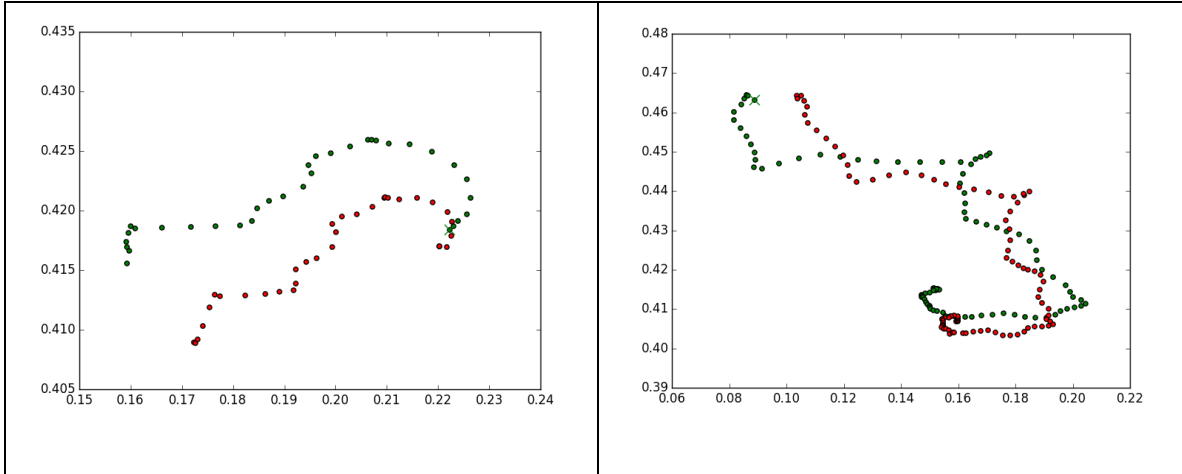


Figure 4. Partial trajectory prediction.

4 Concluding Remarks

The competition result by training with the first 10,000 trips is an error of 2.61219 **km MHD**.

Looking at the training data and the full dataset it is evident that there are areas of the map not even touched by the training samples, this can be one of the causes of such high score found in the submission.

A better solution need to address this limitation in the training taking into account all the dataset or better spatially-distributed training sample.

RNNs/ESNs handle sequential data, and have shown good performances in similar tasks, so we expect better performances with this kind of task, given the high quantity of data possibly an optimized distributed/parallel training would be needed.

It would also be better to use a more accurate estimate of the validation error in the in the model selection with a 5-10 fold CV.

The official winning solution of the competition: <https://github.com/adbrebs/taxi>

Its short report: https://github.com/adbrebs/taxi/blob/master/doc/short_report.pdf

*The official winning solution reached a score (**MHD**) of **2.01 km** with a multilayer perceptron, with a word-embeddings-like pre-processing.*

5 References

- [1] *M. Lukosevicius, H. Jaeger, Reservoir computing approaches to recurrent neural network training, Computer Science Review vol. 3(3), pag. 127-149, 2009*
- [2] *H. Jaeger, H. Haas, Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication, Science, vol.304, pag. 78-80, 2004*
- [3] *H. Jaeger, The “echo state” approach to analysing and training recurrent neural networks, GMD - German National Research Institute for Computer Science, Tech. Rep., 2001*
- [4] *C. Gallicchio, A. Micheli, Architectural and markovian factors of echo state networks, Neural Networks, vol. 24(5), pag. 440–456, 2011*
- [5] *M. Lukosevicius, A practical guide to applying echo state networks, Lecture notes in computer science, Vol. 7700 Springer, Berlin Heidelberg (2012)*
- [6] *JAEGER, Herbert. Echo state network. Scholarpedia, 2007, 2.9: 2330.*