Bike Sharing Demand

Kaggle Competition - Case Study
DMKM

Carlos López Roa
me@mr3m.me

January 15, 2017

Abstract

In the context of the DMKM master, a Kaggle competition was chosen as a case study, in this case the Bike Sharing Demand. The objective of the case study was to put in practice and to develop abilities in the Python programming language, as in libraries such as: numpy, scikit learn and pandas. In this particular approach, explorations using Artificial Neural Networks (ANN) were made, both in the Multilayer Perceptron (MLP) as in the Long Short Term Memory (LSTM) topology, the later implemented in the Deeplearning library Tensorflow. All the code was versioned using git and is available in the repository github.com/DMKM1517/BikeSharingDemand

Contents

1 Introduction **1**
1.1 Dataset 1
1.2 Objetive 3

2 Modelling **3**
2.1 Definition of the problem 3
2.2 Formal definition 4
2.3 MLP 4
2.4 LSTM 4

3 Implementation **5**
3.1 General Pipeline 5
3.2 Experiments 6

4 Results **6**
4.1 MLP 6
4.2 LSTM 7

5 Conclusions **7**

1 Introduction

Bike sharing systems are a means of renting bicycles where the process of obtaining membership, rental, and bike return is automated via a network of kiosk locations throughout a city. Using these systems, people are able rent a bike from a one location and return it to a different place on an as-needed basis. Currently, there are over 500 bike-sharing programs around the world.

The data generated by these systems makes them attractive for researchers because the duration of travel, departure location, arrival location, and time elapsed is explicitly recorded. Bike sharing systems therefore function as a sensor network, which can be used for studying mobility in a city. In this competition, participants are asked to combine historical usage patterns with weather data in order to forecast bike rental demand in the Capital Bikeshare program in Washington, D.C. [1]

The bike sharing system installed in Washington, D.C. goes by the name of capital bikeshake (CaBi) a docking station is shown in figure 1. This system began operating in September 2010 and now has more than 429 stations and 2,500 bicycles.

1.1 Dataset

The data set is composed of 17,381 observations with of 8 features sampled hourly and continuously for 2 years (2011, 2012). They are 3 response variables. A quick overlook of the different features and response variables can be seen in the figures 2, 3, 4, and a description of each one is listed below.

- **datetime:** All observations are indexed by a timestamp in the format

Figure 1: Example of a docking station of the *capital bikeshare* system in Washington D.C.

`YYYY-MM-DD HH:MM:SS`. This index spans from `2011-01-01 00:00:00` to `2012-12-19 23:00:00` with 147 missing observations.

- **season:** Categorical variable that labels the season of the year from 1 to 4, with 1 being Winter and 4 being Fall.
- **holiday:** Categorical binary variable that marks a day as a holiday.
- **workingday:** Categorical binary variable that marks a day as a working day.
- **weather:** Categorical variable that labels the weather from 1 to 4.
- **temp:** Continuous variable that represents the temperature in Celsius
- **atemp:** Continuous variable that represents the apparent temperature in Celsius.
- **humidity:** Continuous variable that represents the humidity from 0 to 100 %
- **windspeed:** Continuous variable that represents the windspeed in m/s
- **casual:** Response variable that represents the number of bike rents made by casual users (non registered) in that time step.
- **registered:** Response variable that represents the number of bike rents made by registered users in that time step.
- **count:** Response variable that represents the total number of bike rents made in that time step. Is equal to the sum of casual and registered for each time step.

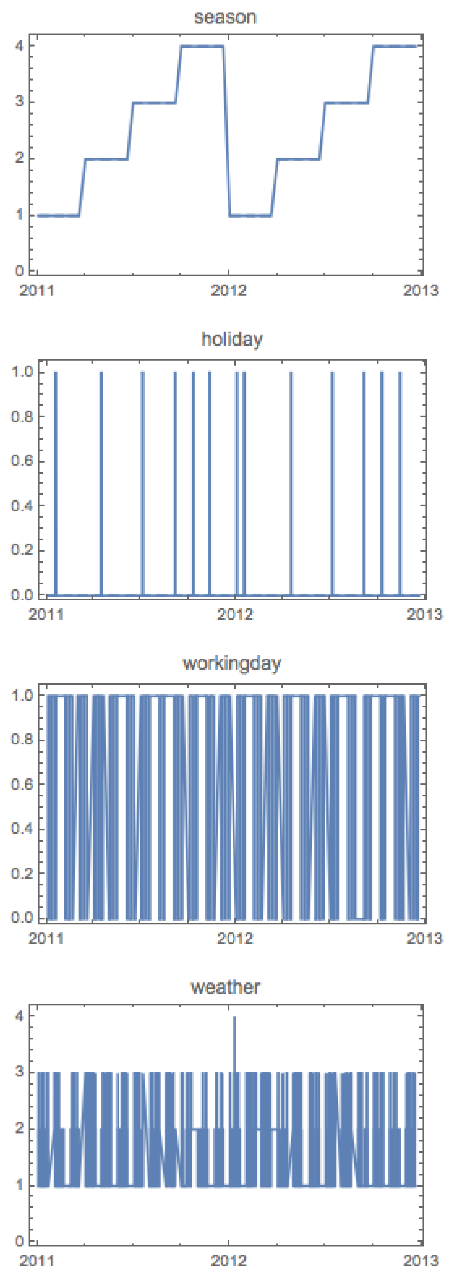

Figure 2: Overlook for the categorical variables: `season`, `holiday`, `workingday` and `weather` for the training set.

The dataset is provided in two subsets, in which one is devoted for training and one for testing. The training set is composed of the first 19 days of each month and is provided with the response variables, it has then 10,886 labeled observations. The testing set is the complement of the training set, that is, it is composed of the days from the 20th to the end of the month for each month. It has then 6,493 un-labeled obser-

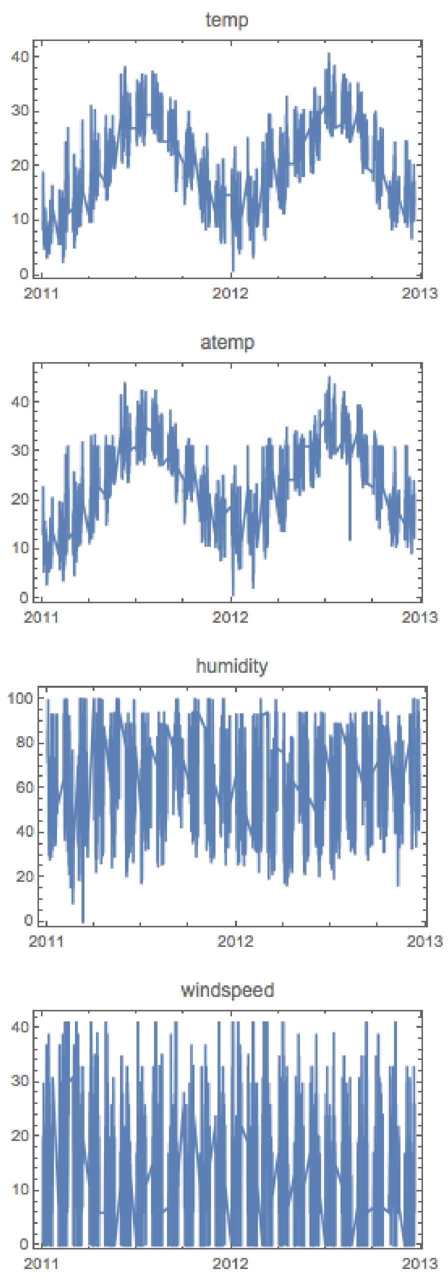

Figure 3: Overlook for the continious variables: `temp`, `atemp`, `humidity` and `windspeed` for the training set.

vations.

1.2 Objetive

Competitors are asked to submit labels for the testing set for the `count` label, that is to predict the total number of bike rentals for the testing set while minimising the RMSLE error between the prediction and the actual values.

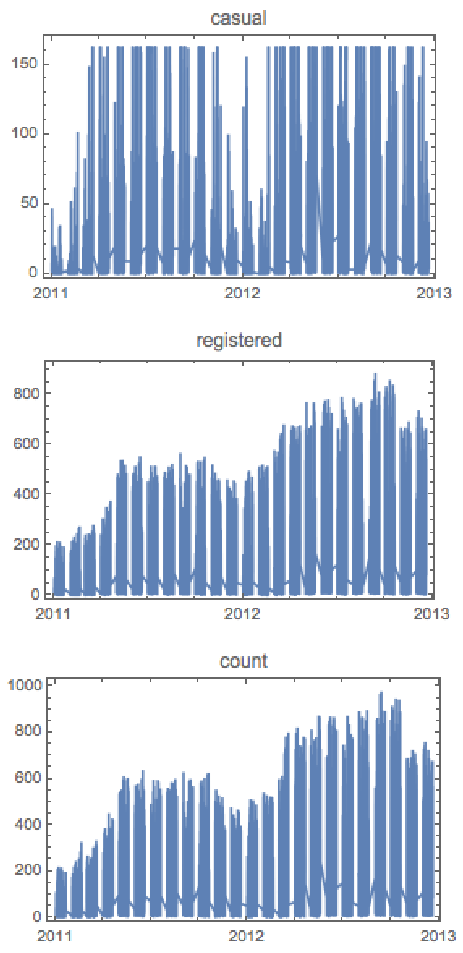

Figure 4: Overlook for the response variables: `casual`, `registered` and `count` for the training set.

The final score given by Kaggle it's computed using the following formula

$$\text{RMSLE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left[\ln(p_i+1) - \ln(a_i+1)\right]^2}, \tag{1}$$

where n is the number of hours in the test set, p_i is the predicted count and a_i is the actual count.

2 Modelling

2.1 Definition of the problem

We can look at this problem as a supervised regression problem where the feature (or predictors) are both categorical and continuous. However the time feature adds an opportunity to upgrade this problem from stationary regression to sequence learning.

A candidate to solve this kind of problem is to use Artificial Neural Networks, where the feature engineering can be automatically done by the algorithm. ANN can automatically learn the correspondence between categorical and continuous variables together, to a either categorical or continuous response variable. Automatic methods to reduce overfitting (regularisation) are also available, hence eliminating the need to do cross-validation.

2.2 Formal definition

Given a dataset D we want to find F, $\hat{y} = F(D)$ such that,

$$\min_{\hat{y}} \mathrm{RMSLE}\left(\hat{y}, y\right), \tag{2}$$

that is, find a function F that approximates the true label y and hence minimises the RMSLE error.

Thanks to the *universal approximation theorem* [2] we know that an ANN with a single hidden layer and a finite number of neurones, can approximate any Boolean functions $\mathbb{B}^n \to \mathbb{B}^m$, with 2 hidden layers they can approximate all real functions $\mathbb{R}^n \to \mathbb{R}$ and with 3 hidden layers they can approximate any function $\Omega \to \Gamma$, that is, subsets of any arbitrary metric space.

We began exploring the Multilayer Perceptron (MLP), and then upgraded the problem to Recurrent Neural Networks (RNN).

2.3 MLP

Multilayer Perceptron is a supervised learning algorithm that learns a function $F(\cdot) : \mathbb{R}^m \to \mathbb{R}^o$ by training on a dataset, where m is the number of dimensions for the input and o is the number of dimensions of the output. Given a set of featured $D = d_{ii}^m$ and a target y, it can learn a non-linear function approximator for either classification or regression. A one hidden layer MLP with scalar output is shown in figure 5 [3]

The input layer, consists of a set of neurones $\{x_i\}_i^m$ that represent the input features of the dataset. Each neuron in the hidden layer transforms the values from the previous layer with a weighted linear summation $W \cdot x$, followed by a non linear activation function $g(\cdot) : \mathbb{R} \to (-1, 1)$.

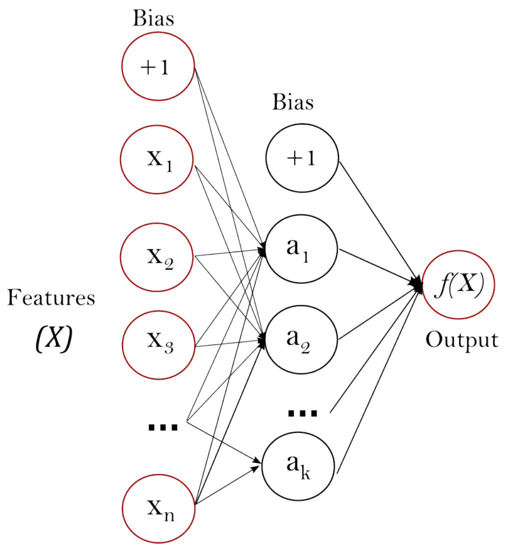

Figure 5: A one hidden layer MLP [3]

The output layer receives the values from the last hidden layer and transforms them into output values. [3]

It's worth noting that in regression problems the output layer returns $W_{n-1} \cdot x_{n-1}$ without applying the activation function. The bias term is introduced to learn constant values in the data set.

During training the optimisation objective

$$\min \Lambda(\hat{y}, y, W) = \frac{1}{2}||\hat{y} - y||_2^2 + \alpha||W||_2^2, \tag{3}$$

is set, where $\alpha||W||_2^2$ is the L_2 regularisation term. The optimisation of the cost function is done using gradient descent and the back-propagation algorithm

2.4 LSTM

Long Short Term Memory networks are a type of Recurrent Neural Networks (RNN) capable of learning long-term dependencies in the data. They are explicitly designed to avoid the problem of vanishing gradient. All recurrent neural networks have the form of a chain of repeating modules of neural network. LSTMs also have this chain like structure, but the repeating module has a different structure figure 6. [4]

LSTM cells have three gates, each gate has an activation function which allows the cell to read, write and forget information.

The equations that govern the behaviour of a cell are the following. The forgetting gate:

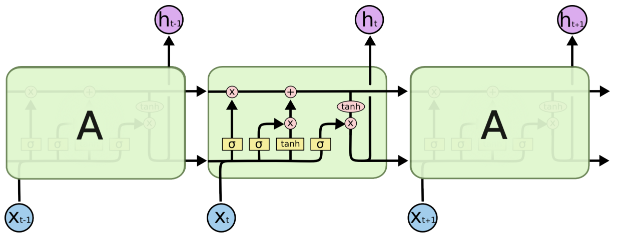

Figure 6: The repeating module in an LSTM contains four interacting layers: Each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denote its content being copied and the copies going to different locations. [4]

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f), \tag{4}$$

where σ is the activation function, W_f are the weights associated with the forgetting gate, h_{t-1} is the exit of the previous layer, x_t is the input at time t and b_f is the bias term for the forgetting gate.

For remembering a new value, a two step process is done, in which first a candidate is set by first forgetting the previous value and setting the new one with the read gate. The equation for the read gate and candidate creation:

$$\begin{aligned} i_t &= \sigma(W_i \cdot [h_{t-1}, x_t + b_i] \\ \tilde{C_t} &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C), \end{aligned} \tag{5}$$

and the actual substitution

$$C_t = f_t \times C_{t-1} + i_t \times \tilde{C_t} \tag{6}$$

The writing gate then outputs the value of the cell filtered by yet another activation function.

$$\begin{aligned} o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o) \\ h_t &= o_t \times \tanh(C_t) \end{aligned} \tag{7}$$

In this way LSTM can selectively learn, forget and retain information of the dataset. Training is done by traditional gradient descent and backpropagation algorithms

3 Implementation

The models were implemented in `Python 2.7.10` under the Anaconda distribution [5]. The MLP was implemented using the `neural-network.MLPRegressor` of `scikit-learn` [3]. The LSTM Neural Neural networks were implemented in the stack of Google's Deep Learning library `Tensorflow` [6] interfaced with the `Keras` library [?]

To reduce the computation an AWS EC2 instance with 16GB RAM and 8 cores was used, the computation time was reduced by a factor of 10.

3.1 General Pipeline

To allow a uniform response of the models the following pipeline was designed and implemented. This can be seen in figure 7

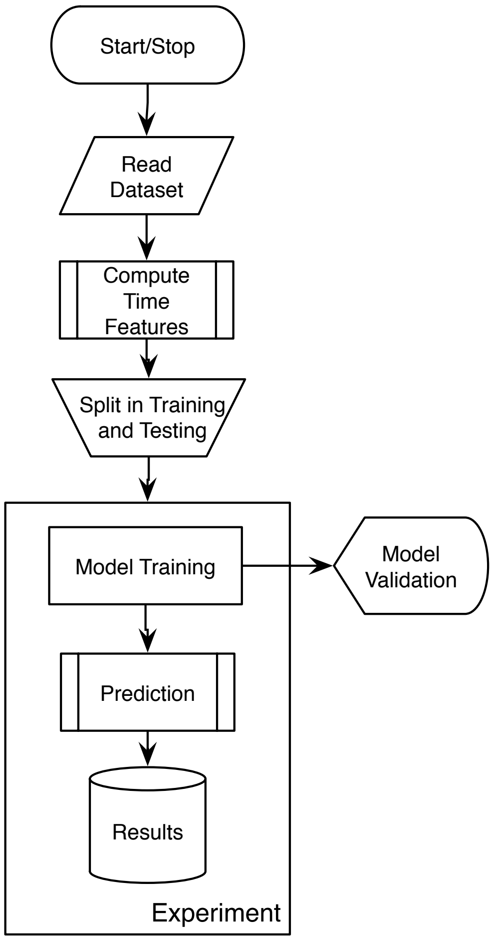

Figure 7: General Pipeline for the implementation

1. First the data sets are read into pandas dataframes.

2. Since the ANN cannot treat the `datetime` feature, and we want to extract information of the time feature, the features `month`, `dayoftheweek`, `hour` and `year` are derived from the `datetime` .
3. Then the original training set is split into training and testing. This is done by taking the first 80% of the values as training and the last 20% as testing, either by the whole dataset, by year or by month.
4. The model training is then done on the new training set by a fixed number of steps or when the gradient vanishes.
5. The prediction is done
6. The results of the training and the prediction are visualised and several measures are computing on the testing set.
7. The results are stored in disk
8. When performing an experiment (testing sets of parameters or hyper-paremeters) a cycle from step 4 until step 7

3.2 Experiments

A series of experiments were designed, both as an exploration of the models and of the parameters of the models.

An exploration over the topology of the MLP NN was made iteratively, responses in the computation time and accuracy were computed. Also, two submodels were trained independently, one predicting the `casual` response while the other predicted the `registered` response.

An exploration over the topology, the influence of regularisers, the use of submodels, the batch size and the number of epochs computed was done cross-relating it with the size of the timesteps considered in the LSTM NN.

4 Results

4.1 MLP

An overview of the submission results for the MLP tests can be seen in figure 8.

Figure showing the different performance of different MLP topologies can be seen in figures 11, 10 and ??

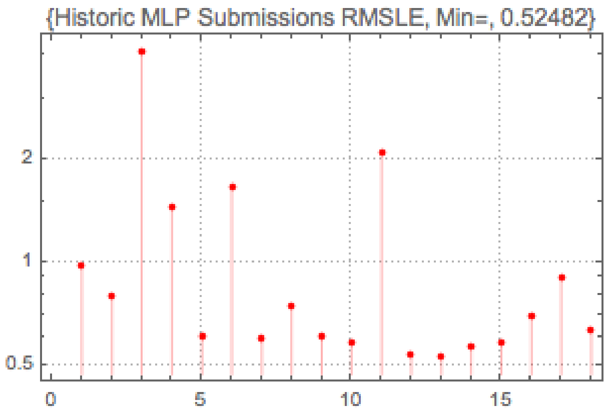

Figure 8: Historic view of the submission results in Kaggle for the MLP model: In this figure we can see the evolution of the RMSLE error until the minimum is achieved after tuning the network topology. After some other tunings are

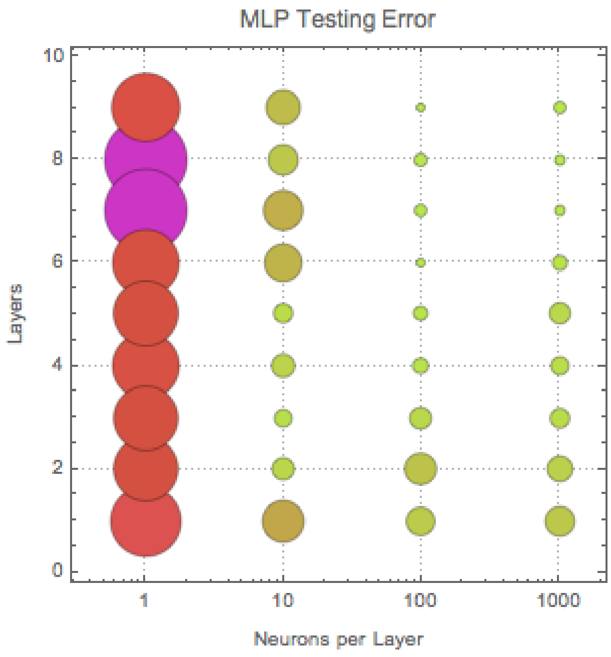

Figure 9: MLP Testing error as a function of the number of layers and the neurons per layer: Here we can see that, the bigger the network better are the results, specially when the number of layers increases. One can have the effect of overshooting the number of neurons per layer.

After revising the effect of the number of layers and the neurons per layer, the size of 100 layers per neuron was chosen and explored deeper. The results can be seen in the figures 12, 13 and 14

The best results where achieved with a MLP

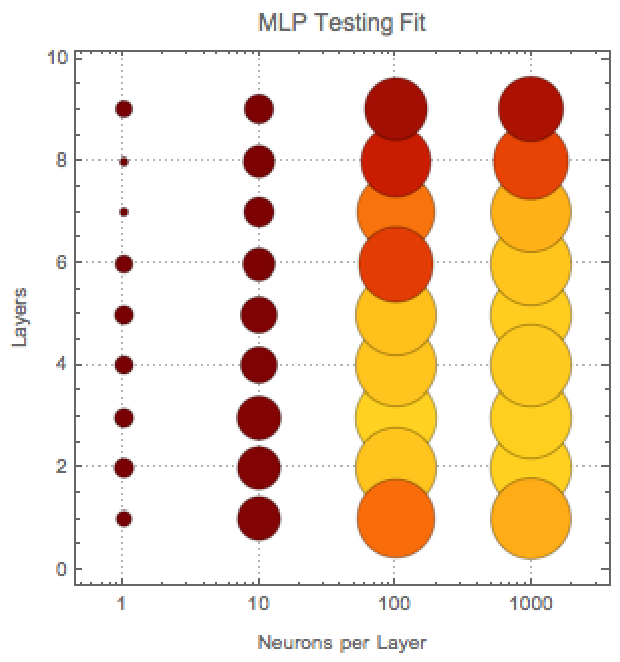

Figure 10: MLP Testing fit as a function of the number of layers and the neurons per layer: Here we can see that small NN have poor fit of the dataset, whereas bigger NN have better fit, one can see the fit diminishing when the number of layers it's too high.

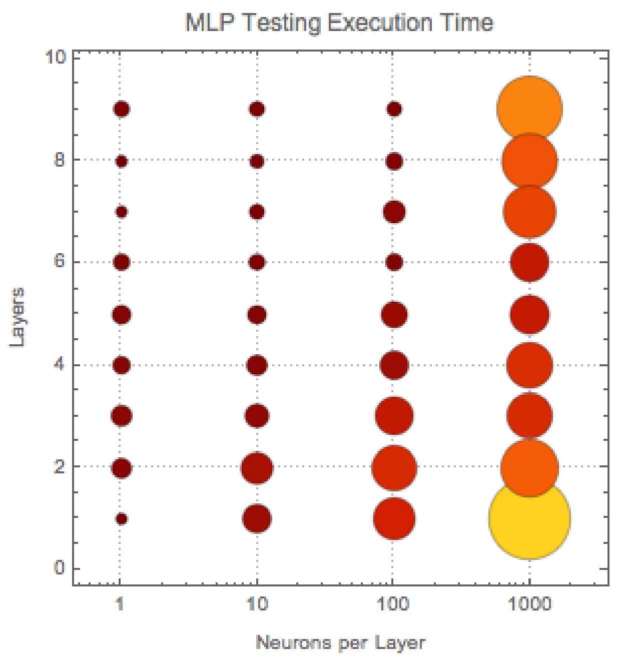

Figure 11: MLP Computation time as a function of the number of layers and the neurons per layer: The complexity of MLP grows as $O(h^k)$ where h are the number of hidden layers and k are the number of layers. In this case the stopping criteria is the vanishing of the gradient. We can see the enormous computation time that big NN can require fro training. The maximum here is aroung $8.1 * 10^3$ seconds [3]

with 10 layers and 100 neurons per layer.

When using the best setting and comparing single model and double model (one per each response variable, the RMSLE result was: $0.57953, 0.60359$ respectively. That is, single model was better

4.2 LSTM

An overview of the submission results for the MLP tests can be seen in figure 15

The amount of steps that the LSTM looks for learning a certain output is called timestep. The effect of timesteps was studied for different groups, the results can be seen in figure 16

5 Conclusions

- The MLP model cannot scale, because of numeric instability and computation cost
- The MLP model can learn the data as stationary with acceptable performance
- The MLP model fails to learn the features associated with time
- The LSTM model achieves and surpasses the results of MLP
- The LSTM model can increase it's precision by using more layers and more neurons with a triangular topology, thereby creating higher-level abstractions of the data set.
- The effect of the timesteps was not positive
- The effect of using submodels was not positive

References

[1] Fanaee-T, Hadi, and Gama, Joao, Event labeling combining ensemble detectors and background knowledge, Progress in Artificial Intelligence (2013): pp. 1-15, Springer Berlin Heidelberg.

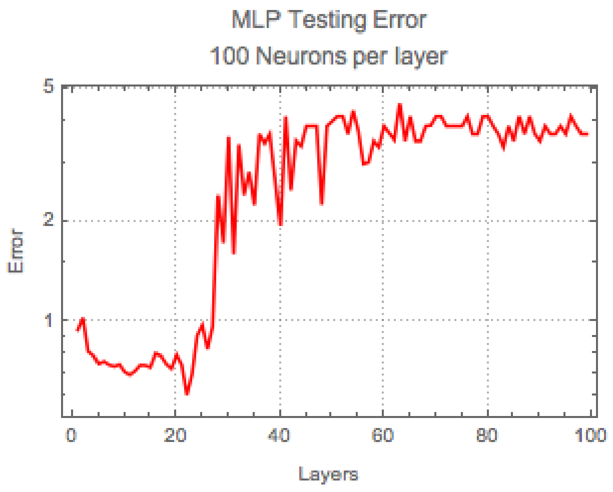

Figure 12: MLP Testing error as a function of the number of layers fixing 100 neurons per layer: We can see that the error redices until a minimum at 21 but then it keeps growing. Hence we cannot expect to reduce the error by just increasing the number of layers.

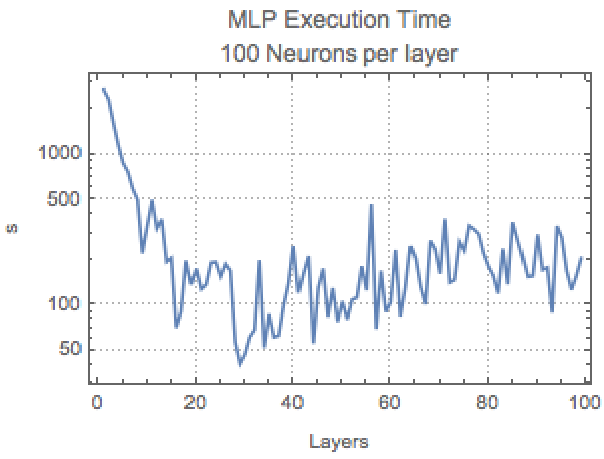

Figure 14: MLP Execution time as a function of the number of layers fixing 100 neurons per layer: We can see that in fact the execution time reduces in the first regime, due to the fact that each iteration is more costly but the number of iterations before the stopping criteria is met decreases.

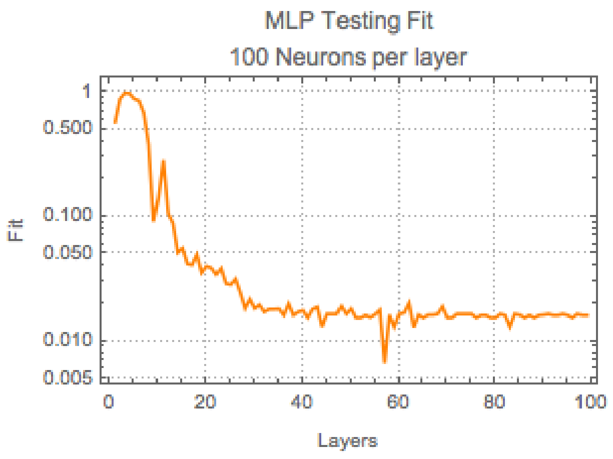

Figure 13: MLP Testing fit as a function of the number of layers fixing 100 neurons per layer: we can see that the fit in fact reduces rapidly after a maximum in the region (0-10) and gets to a stable low point.

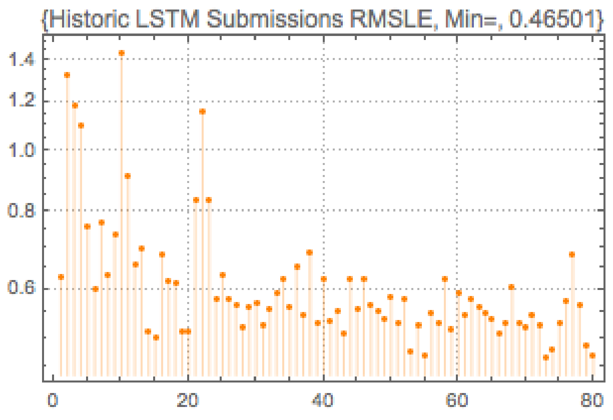

Figure 15: Historic view of the submission results in Kaggle for the LSTM model: In this figure we can see the evolution of the RMSLE error until the minimum is achieved after tuning the network topology, regulariser weight, and timesteps.

[2] Cybenko., G. (1989) "Approximations by superpositions of sigmoidal functions", Mathematics of Control, Signals, and Systems, 2 (4), 303-314

[3] Neural network models (supervised), Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[4] Understanding LSTM Networks, Christopher Olah

[5] Anaconda Software Distribution. Computer software. Vers. 2-2.4.0. Continuum Analytics, Nov. 2016. Web. <https://continuum.io>.

[6] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen,

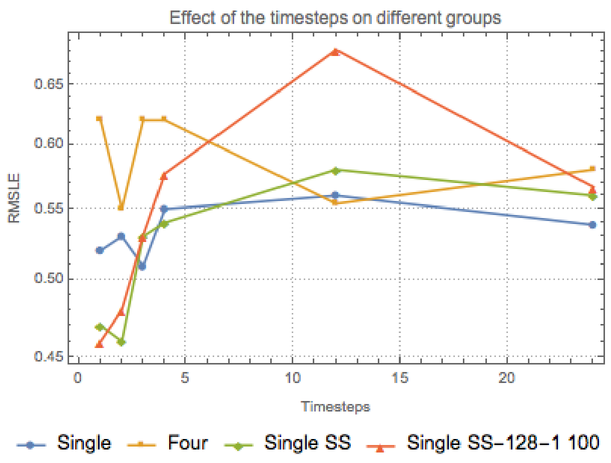

Figure 16: Effect of timesteps on different groups: The group group `single` is the single model with LSTM Layers (100,100,100,10,1), the `four` group is composed of 4 models, with the same topology, one per year and response variable, the `Single SS` is composed of LSTM Layers (100,50,25,10,1), and the `Single SS-128-1 100` is composed of LSTM Layers (128,64,32,16,8,4,2,1) topology. We can see the best response with the last one.

Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[?] KerasChollet, François, 2015, GitHub,https://github.com/fchollet/keras