Course Project
Artificial Intelligence in Industry

Computer Science Department – Science and Engineering
Artificial Intelligence

University of Bologna

Daniele Domenichelli – 954277
daniele.domenichell2@studio.unibo.it

October 2021

Index

Chapter 1 - Introduction

A very common and growing problem in many financial applications, is the capacity to forecast business time series to better predict the behaviour that a seller should employ with their prices to maximize the selling income.

The target of this project is to explore and compare different approaches to analyse and forecast time related distributions in the business time-series prediction context, by extracting and constraining temporal relations intrinsic in the selling behaviour (seasonalities).

The study was conducted to produce a simple baseline model, for this reason the models were limited to use just as little inputs as possible. In particular, only the amount of sold units and the relative month of the year were used for the training phase and to produce the output.

Chapter 2 - Dataset and Framework

The dataset used for this project was harvested and extrapolated from real-case scenarios data, then adapted and pre-processed to be suitable for the project intents.

The first step of the project is to build a framework that pre-process the dataset, feed it to the target model and then retrieve the chosen metrics to compare with the other models.

2.1 Data Overview

The dataset is represented by a table with many columns, only a few of them are selected in order to be pre-processed for further analysis.

Here an example of some rows contained in the dataset:

Date	Units	Amount	Product ID	Seller ID	…	Brand Code
2017-01-01	12	22.52	7845578	99988544	…	127145
2017-01-02	11	20.71	7845578	99988544	…	127145
2017-01-04	42	43.86	7845578	98875441	…	127145
2017-01-04	92	96.33	7845578	99988544	…	127145
…	…	…	…	…	…	…

Table 2.1.1: Dataset slice overview

Each row in the dataset represents a day in the market, for a specific product sold by a seller.

We describe shortly the shown columns:

- Date: the date when the sale was made
- Units: the quantity of product sold
- Amount: the total income (expressed in dollars)
- Product ID: an identifier of the product sold
- Seller ID: an identifier of the seller which made the transaction
- Brand Code: an identifier of the brand of the product sold

Given the high dimension and size of the dataset, a first selection was conducted to reduce the number of rows and to focus the scope of the analysis.

A reasonable selection was yielded by filtering and extracting only the most sold brand. Each brand indeed preserves a sensible amount of information relative to different sellers and their behavior, excluding in part the habits and preferences of customers in between different sellers or brands.

The most populous brand still brings a total number of 2.23M rows in the subsequent preprocessing.

After the brand selection, any unused column is dropped to facilitate the computations.

2.2 Pre-processing

A usual problem of managing real data is the presence of out-of-domain or invalid entries. As an usual practice, we should try to understand what are the causes that lead to the presence of these kind of entries, but, given the dimension of the dataset, this falls behind the project target and intents. So, the invalid rows were dropped by a filtering.

We can see which the invalid cases were encountered and filtered:

- Null or invalid dates
- Negative amount or units
- Units greater than 10'000, since some data were stored in typed database, some negative values became their very large positive two complement (65535 unsigned = -1 signed). After this filter, the greatest number of units sold in a single day is 182.

The filtered dataset counts 1.87M rows (0.36M filtered invalid cases).

The target of the project is the inspection of seasonalities in a monthly period. For this reason, the dataset was reshaped (w.r.t original shown in *Table 2.1.1*), and more informative columns were added:

Date	Month	Units	Amount	Price	Product ID
2017-01-01	1	12	22.52	1.877	7845578
2017-01-02	1	11	20.71	1.882	7845578
2017-01-04	1	42	43.86	1.044	7845578
2017-01-04	1	92	96.33	1,047	7845578
…	…	…	…	…	…

Table 2.1.2: First pre-processing

Furthermore, we exploit another known property of the dataset to get rid of the product column dependency. We know that products of the same brand falls into a similar macro-category (e.g. dairy products, meat, etc.), so their seasonalities are strictly related from one to another. The simplest solution would be to assume that all the rows are of the same product, but this does not take in account of the variation of price between different products. To overcome to this problem, an intra-class normalization is applied, such that the **Amount** column is rescaled by the average price of the product.

Date	Month	Units	Amount
2017-01-01	1	12	15.01
2017-01-02	1	11	13.81
2017-01-04	1	42	29.24
2017-01-04	1	92	64.22
…	…	…	…

Table 2.1.3: Pre-processed rescaled dataset

2.3 Framework

Once the dataset had become stable, it is needed to establish a common interface for the models to operate, such that we can easily compare the results between different models.

Each model will be fed during the training with both units and month columns (or their mean and std if the model is probabilistic), and the loss will be computed over the price column (extracted from the ratio between Amount and Units). At the test phase, the metrics are computed by feeding in the same way the model and evaluating each metric with respect of the price target column.

The final output of any model will be the forecasted price for each pair units-month, which in case of probabilistic models was sampled by a distribution over location and scale produced by the network.

The dataset has been enriched for this reason with the mean (location) and standard deviation (scale) of the price for each pair of units-month, representing the inputs for probabilistic models.

Month	Units	Price	Mean (Price)	Std (Price)
1	12	1.251	1.333	2.341
1	11	1.255	1.325	2.372
1	42	1.122	1.236	2.852
1	92	1.012	1.199	2.979
…	…	…	…	…

Table 2.1.4: Final dataset

Chapter 3 - Experiments

The exploration of different approaches was conducted to understand the peculiarity of different models, in the context of a regression task over seasonal properties.

This simple ablation study employed four different model architectures, with increasing complexity, to better highlight the effects of each single model.

At first, it was produced a simple baseline for each architecture in order to find the best performant configuration. Once the architecture structure was fixed, a search for the best hyperparameters was conducted to prevent the overfitting and test the model capacity to generalization.

The four architectures are:

- Multi-layer perceptron (MLP) regressor
- Probabilistic MLP
- Lattice Model
- Probabilistic Lattice

Every model is trained with an appropriate batch size and learning rate (256 and 10^{-3} by default), MSE for loss, Adam as optimizer ($\beta_1 = 0.9, \beta_2 = 0.999$). The dataset was shuffled before being fed into the model. The 10% of the train dataset was randomly selected to be used as a validation set.

During the training the train and loss was checked to be a valid number (different from *NaN* for example), in order to ensure the validity of the train. In case of an invalid number, the trainings were stopped and restarted with a different initialization.

3.1 MLP Model

The first employed model is a simple network composed by a variable number of layers with different sizes. The model was built with a decreasing number of neurons for deeper layers, such that it simulates a progressive embedding compression of the input toward the 1-dimensional output.

A grid search over the number and size of hidden layers was conducted to find out the most performant architecture, which lead to the choice of the baseline model.

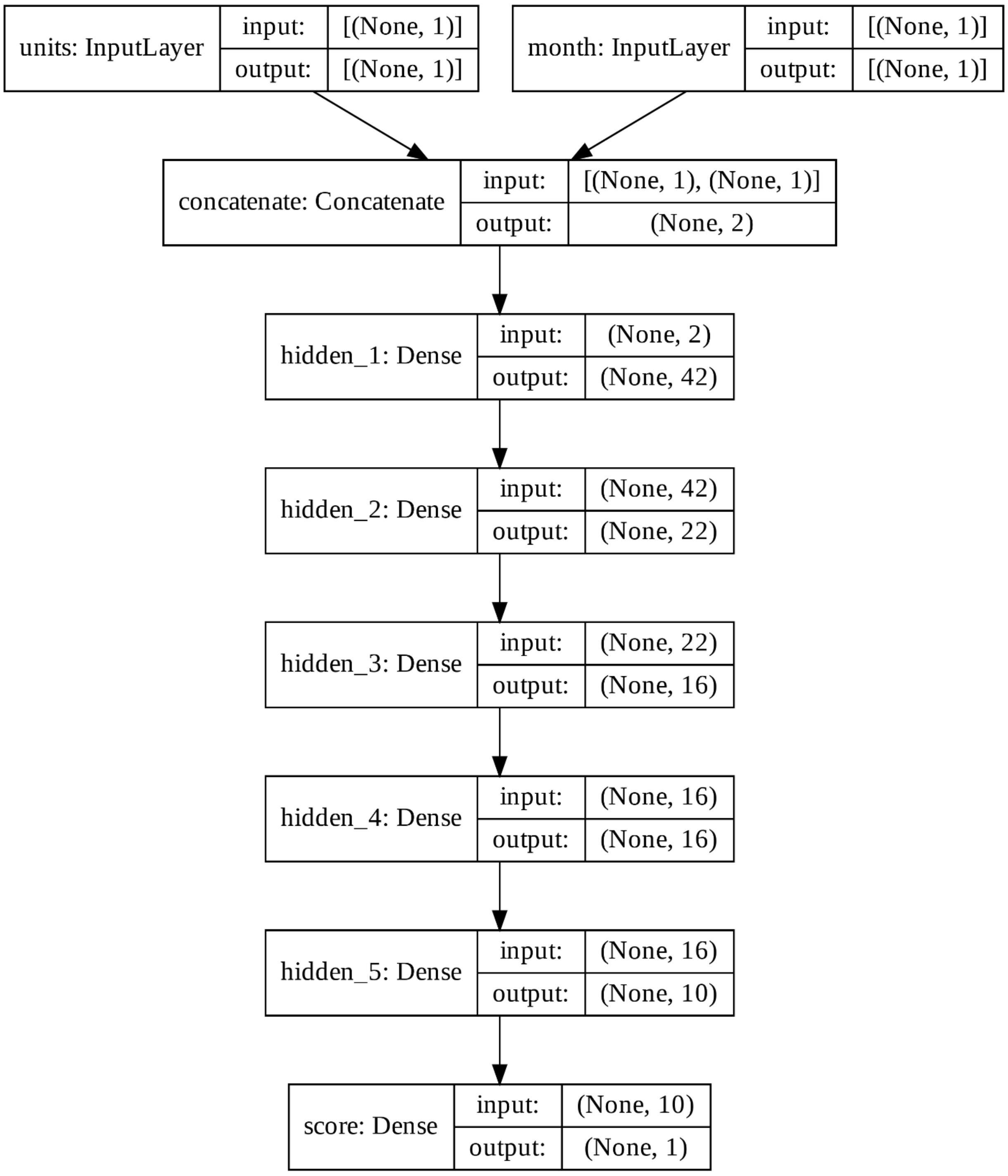


Image 3.1.1: MLP model

3.2 Probabilistic MLP Model

The financial time-series are usually aleatory and strictly dependant from the current period: we can have very different outputs from the same inputs, just one year after the other. On the other hand, there are periods in the year where people are tempted to buy more, like holidays and summertime. For this reason, it is possible to extract certain pattern by looking at the correlation between months, units, and prices. From the dataset emerges indeed the market law, where higher prices lead to lesser sales.

To overcome the unreliability of the input-output relation, it is possible to make the model learn to predict the location and scale of the input distribution. This means that we are not predicting the actual value, but we predict a guess of its position (location) and a grade of confidence (scale) given a chosen distribution.

The model was expanded from the MLP regressor, adding one dimension over the head's output to predict both location and scale, and using a distribution lambda layer to learn the actual distribution, chosen from:

- Gaussian (normal)
- Fixed-scale Normal
- Double Exponential (Laplacian)
- Poisson
- Exponential

As for the MLP model, a tiny grid search was conducted to explore the needed depth of the network.

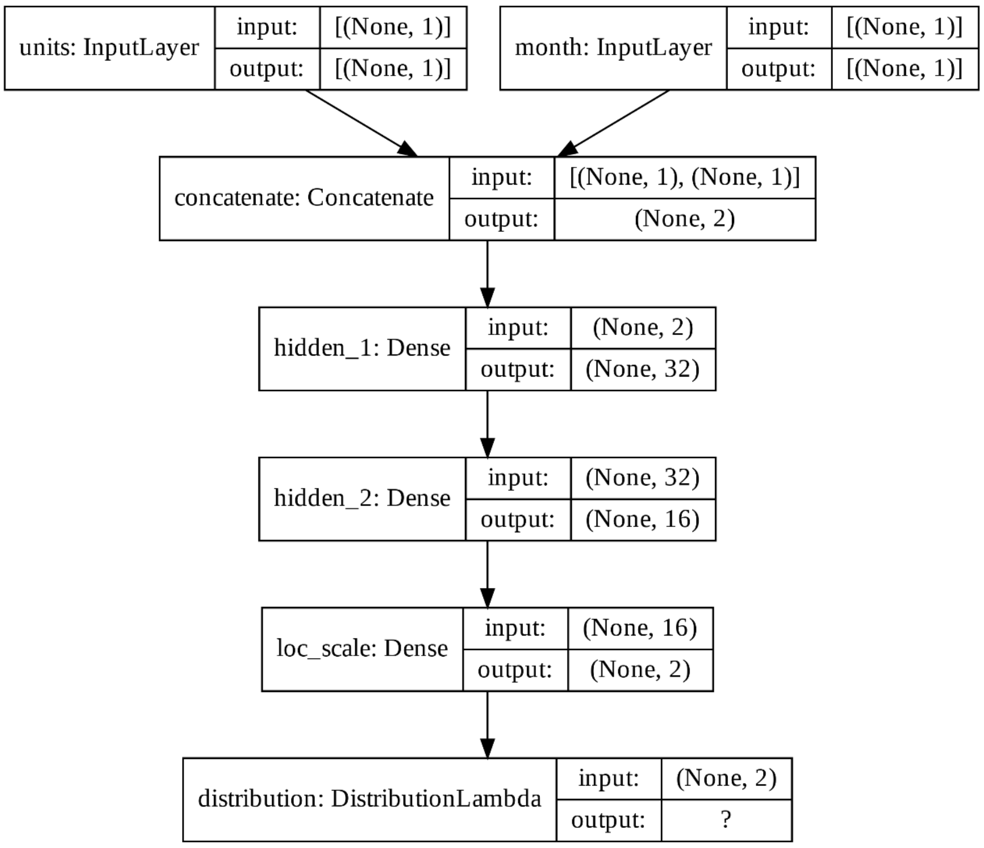


Image 3.2.1: Probabilistic MLP model

3.3 Lattice Model

As already stated, sales are strictly related to the month they happen. We could describe a transaction as a pair of month-unit inputs which produces, through the unknown selling function, a price as output. We can so distribute every transaction as a point over a surface and try to fit a grid over it. The grid is called lattice, and it can be intended as a piece-wise-linear (PWL) regression over multiple dimensions. In general, the lattice is responsible to extract and interpolate hyperplanes between a finite set of points of the input space, such that it is possible to predict a point in the output space by interpolating over the nearest hyperplane.

In our case with 2 dimensions in the input (month-units) and 1 in the output (price) we are learning a regression of a plane between each 4 input pairs of points in the domain.

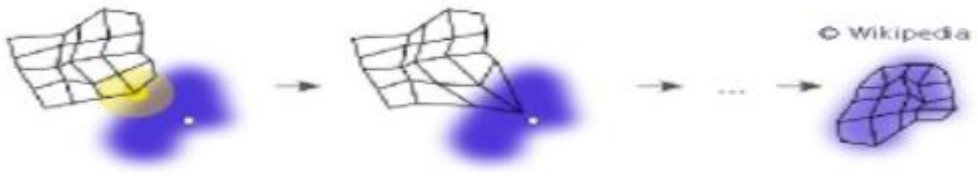


Image 3.3.1: The lattice deformation reflects the density of the observations

The lattice can be implemented as a cascade of multilinear look-up table [1], giving to the regression a better elasticity, such that it is possible to inject different constraints during the training: for example, we can explicitly force the hyperplane to maintain the sign of the first and second derivative (monotonicity and convexity/concavity) or we could impose that the grid must have a unique global minima/maxima (unimodality).

As for the MLP model, a simple baseline was built for the Lattice. The baseline takes produce a PWL grid regression over the normalized pairs month-units to produce the price output. The normalization allows the network to learn a perturbation over the identity function and have a zero-mean and unit-variance input, reducing the effect of the lattice bias over the constraints [2]. Also, the normalization is useful for the TensorFlow Lattice implementation, which requires to define an operating domain for the lattice.

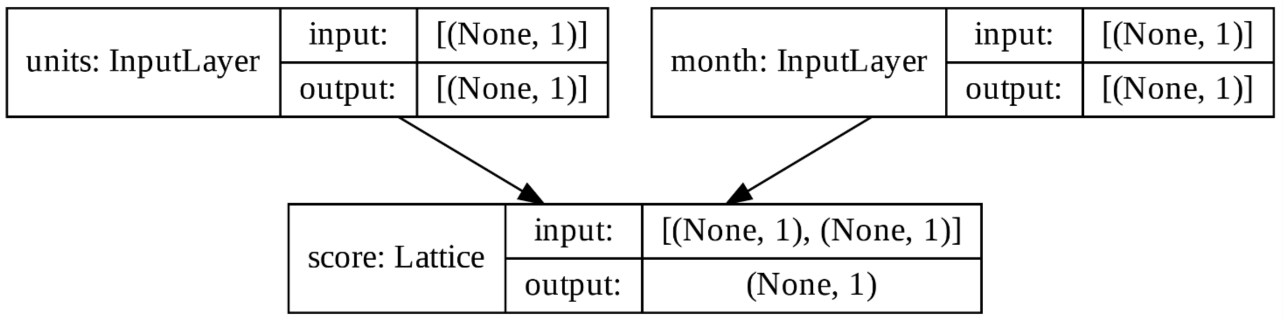


Image 3.3.2: Lattice model – baseline

Another approach similar to a learnable normalization, usually employed with PWL regressions, is the calibration of the inputs. A PWL calibration is linear layer which is intended to normalize the input at train time and clip the out-of-range samples at test time. The advantages of an embedding of the input are a better flexibility of the model during the training and a better response in the so called "Diminishing returns" cases. These are caused by a non-linear relation between an input and the effect produced in the output.

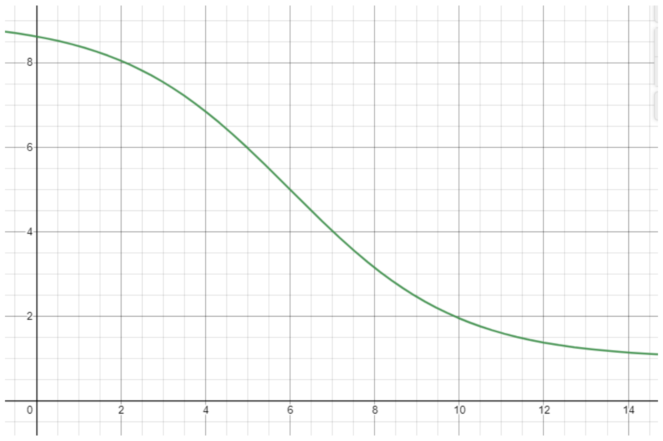


Image 3.3.2: Saturating market law

This is very true in a more practical business view called "saturating market law", where lowering or raising excessively the price does not produce a linear effect: a very low price makes the sales reach all the available target public, saturating the demand, so reducing the price will not alter the sold units; on the opposite, increasing it too much will not bring the sales to 0, because some people could be interested or have the needing to buy the product.

These changes reflect in the model with the addition of two calibration layers before the lattice.

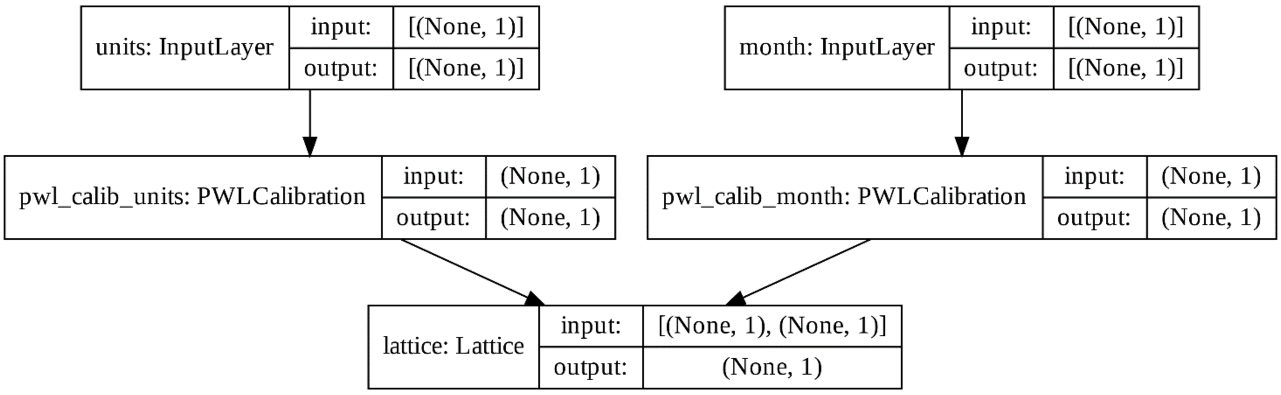


Image 3.3.3: Lattice model – calibration

3.4 Mixed Approach

Usually, in a neural network context, if each of two components increase the performance, their combination usually boosts the increment. So, the last explored model was the composition of both the Probabilistic and Lattice models, used to test the combination of their effects.

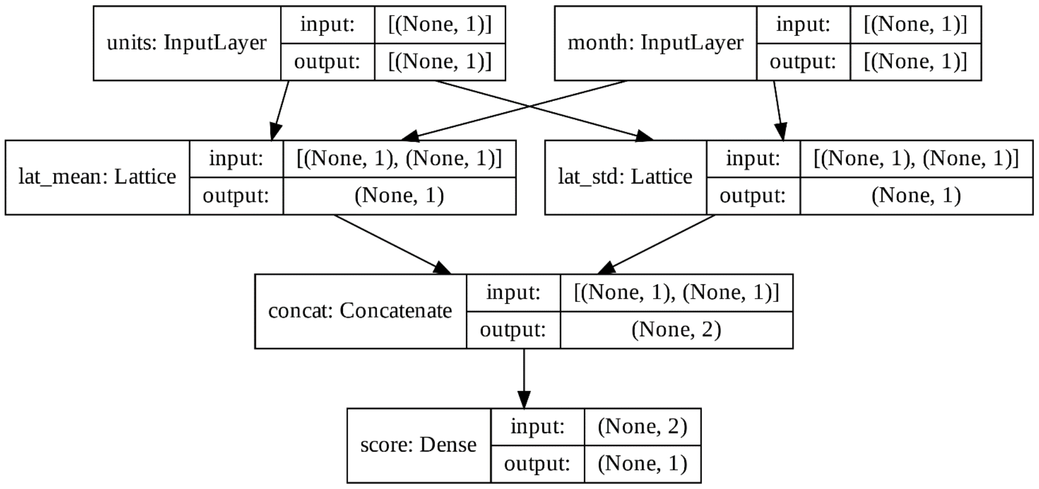


Image 3.4.1: Lattice model – dual

The combination consists in the addition of the distribution lambda layer as head of a double lattice layer. The first attempts to make the model works were not successful due to a particularity of the Lattice model: is easier to train a PWL regression in lower dimensional spaces. While exploring the model over the test set, a contrast in MSE was observed in the predictions: where the MSE of the mean was high, the one of the standard deviation was low and vice versa. By separating the lattice predictions in two different layers, a slight increment of performances was registered, with a more stable consistency between the predictions. For this reason, a dual lattice model was employed.

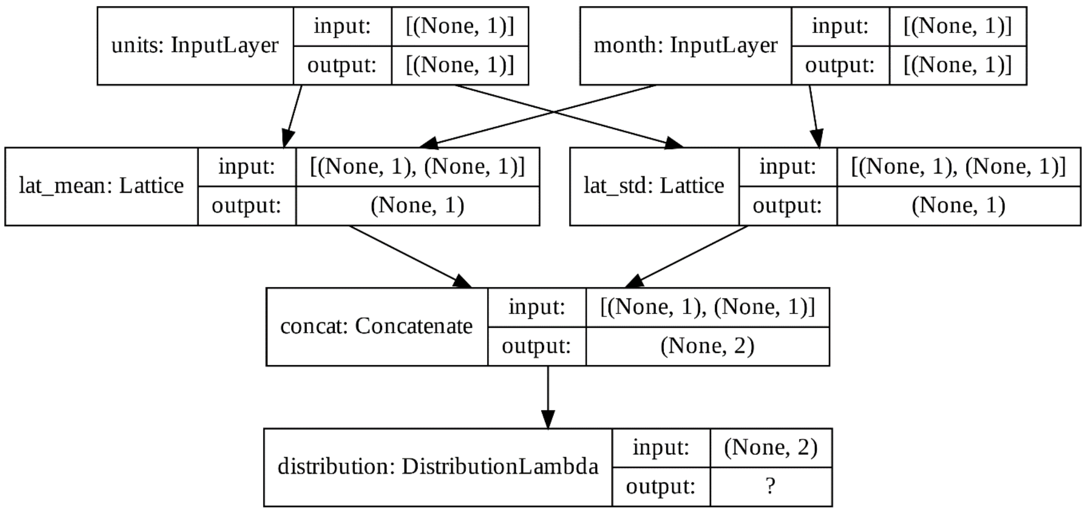


Image 3.4.2: Probabilistic lattice model - dual

Chapter 4 - Results

Synthetic data:

The first test, used to find baseline models, was conducted over a synthetic dataset. This highlights how different models behave under comfortable conditions and show that they are evaluating better than random proposal in a simple context.

The synthetic dataset was built by sampling units and months from an equally spaced grid 10x12, both in the range [0, 1]. The relative price is a composition of sine and cosine multiplied by a random factor, sampled from a normal distribution, near to one (location 1 and scale 0.01). This function makes the price range between 0 and 1.

$$x(u, m) = \frac{1}{4} normal(1, 0.01) \cdot (\sin(u\pi) + \cos(m\pi) + 1)$$

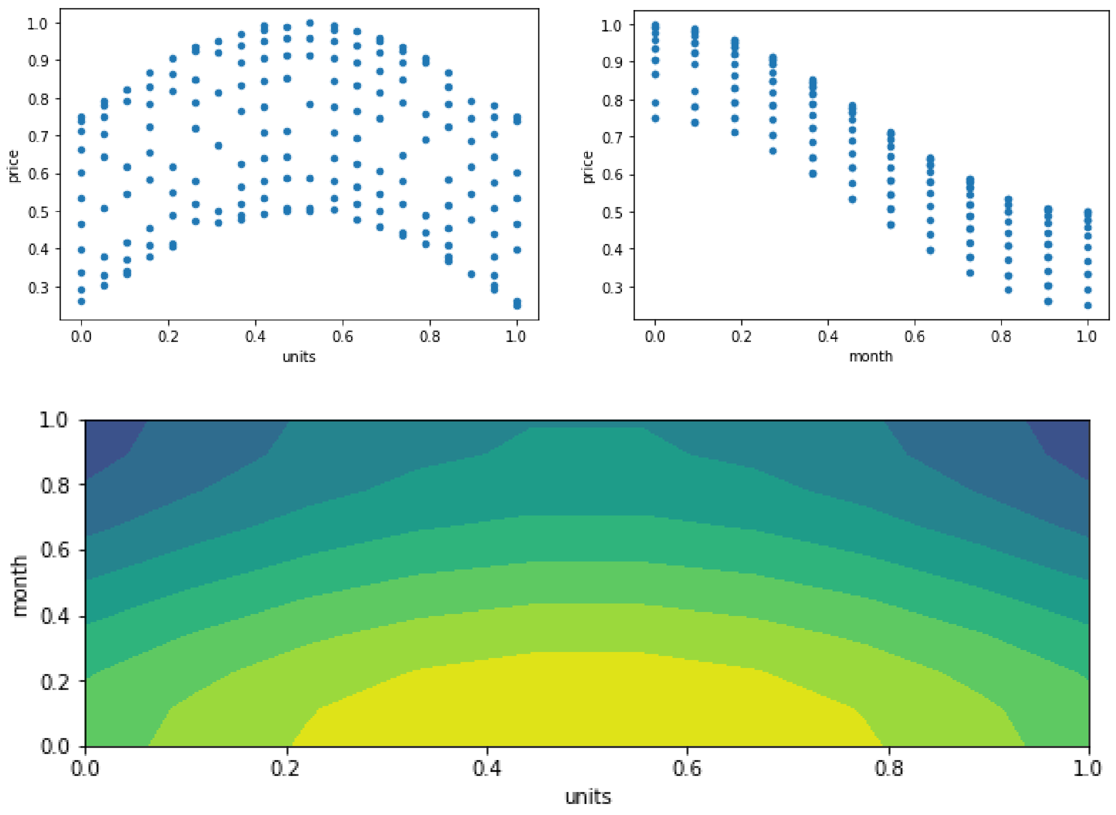


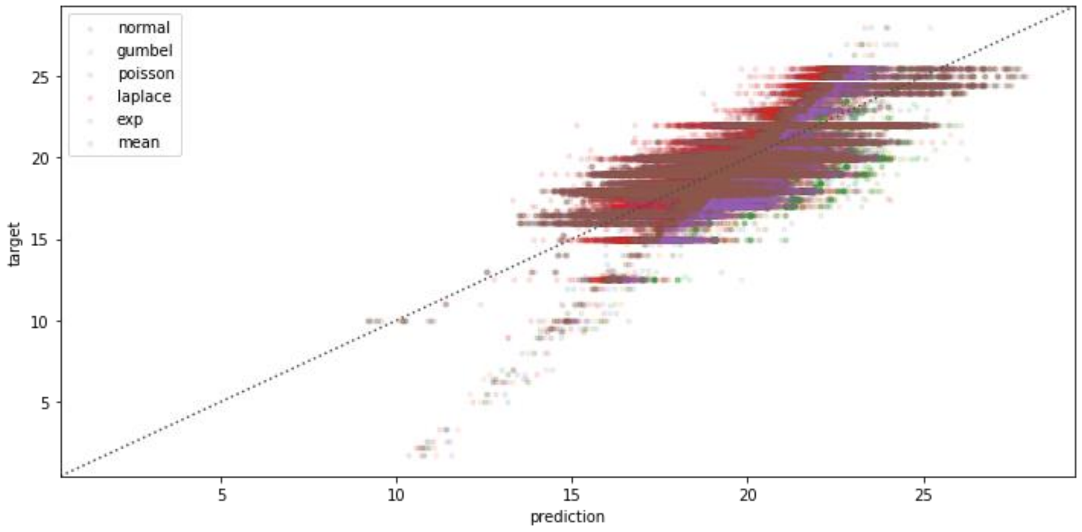
Image 4.1: Synthetic dataset

Model	MSE		R2	
	Mean	STD	Mean	STD
MLP	0.357		0.8755	
Probabilistic MLP (PMLP)	0.221	0.121	0.9755	0.9223
Lattice (Lat)	0.252		0.9711	
Calibrated Lattice (CLat)	0.224		0.9742	
Dual Lattice (DLat)	0.220	0.123	0.9322	0.9203
Probabilistic Dual Lattice (PDL)	**0.194**	**0.118**	**0.9571**	**0.9225**

Table 4.1: Synthetic data test

Probabilistic distribution study:

From the same test, in order to discover which distribution fits the best four our case, both the probabilistic models were built and trained with different distributions. Here are reported the results obtained:



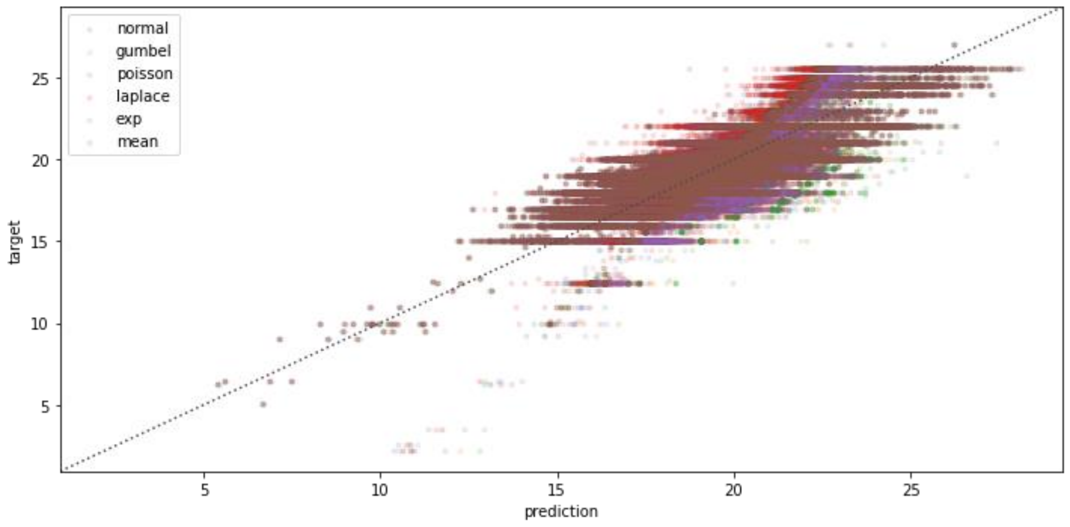


Image 4.2: Distribution comparison - scatter plot (PMLP top, PDL bottom)

Distribution	MSE		R2	
	Mean	STD	Mean	STD
Gaussian (normal)	3.689	4.055	0.498	0.335
Gumbel	3.522	4.154	0.559	0.334
Poisson	3.641	4.119	0.588	0.287
Laplace (double exp)	**3.754**	**4.120**	**0.591**	**0.348**
Exponential	3.124	4.211	0.510	0.295

Table 4.2: Distribution comparisons – table (PDL)

Real dataset:

Here are the results of all the best models tested over the real test dataset processed as already mentioned in **Chapter 2**:

Model	MSE		R2	
	Mean	STD	Mean	STD
MLP	7.994		0.3471	
Probabilistic MLP (PMLP)	3.754	4.120	0.5912	0.3484
Lattice (Lat)	3.577		0.5277	
Calibrated Lattice (CLat)	3.335		0.5841	
Dual Lattice (DLat)	3.212	3.981	0.6249	0.4951
Probabilistic Dual Lattice (PDL)	**3.091**	**3.522**	**0.6543**	**0.5108**

Table 4.3: Real scenario test set

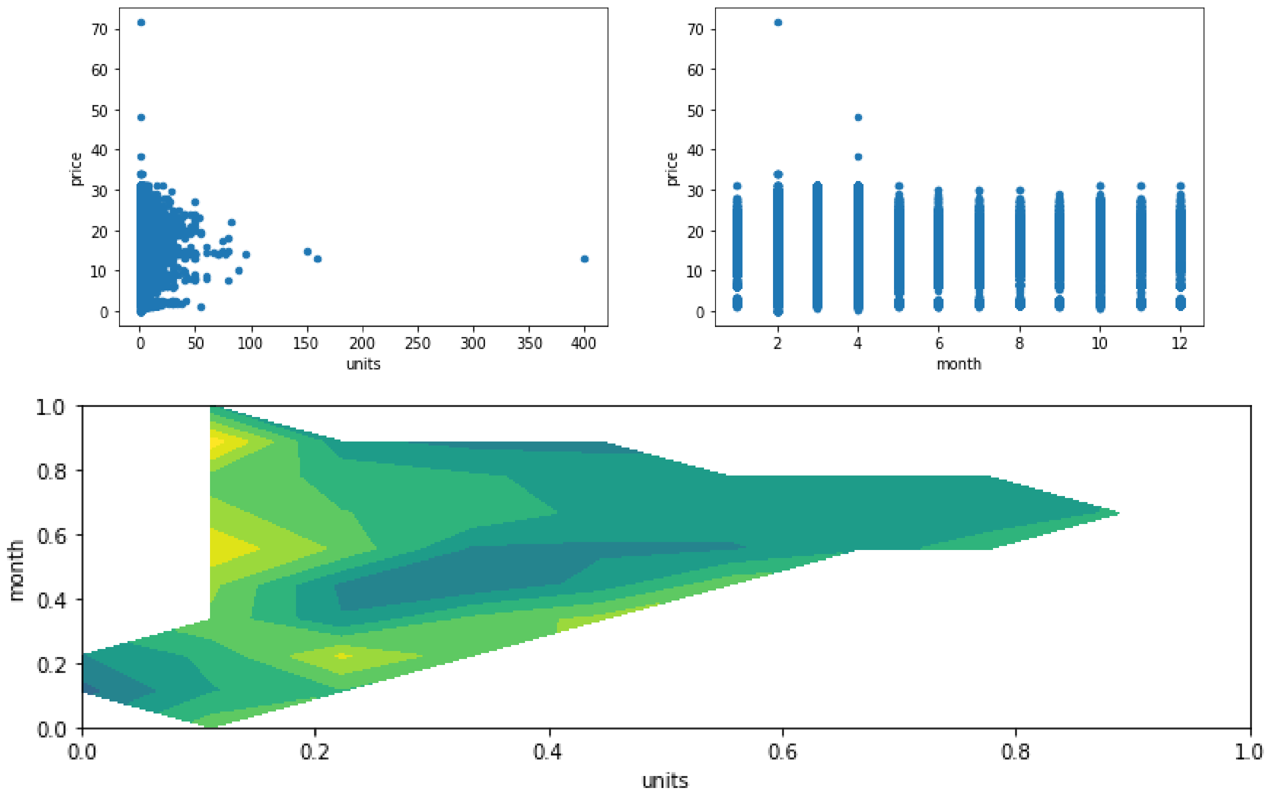


Image 4.3: Real scenario test set

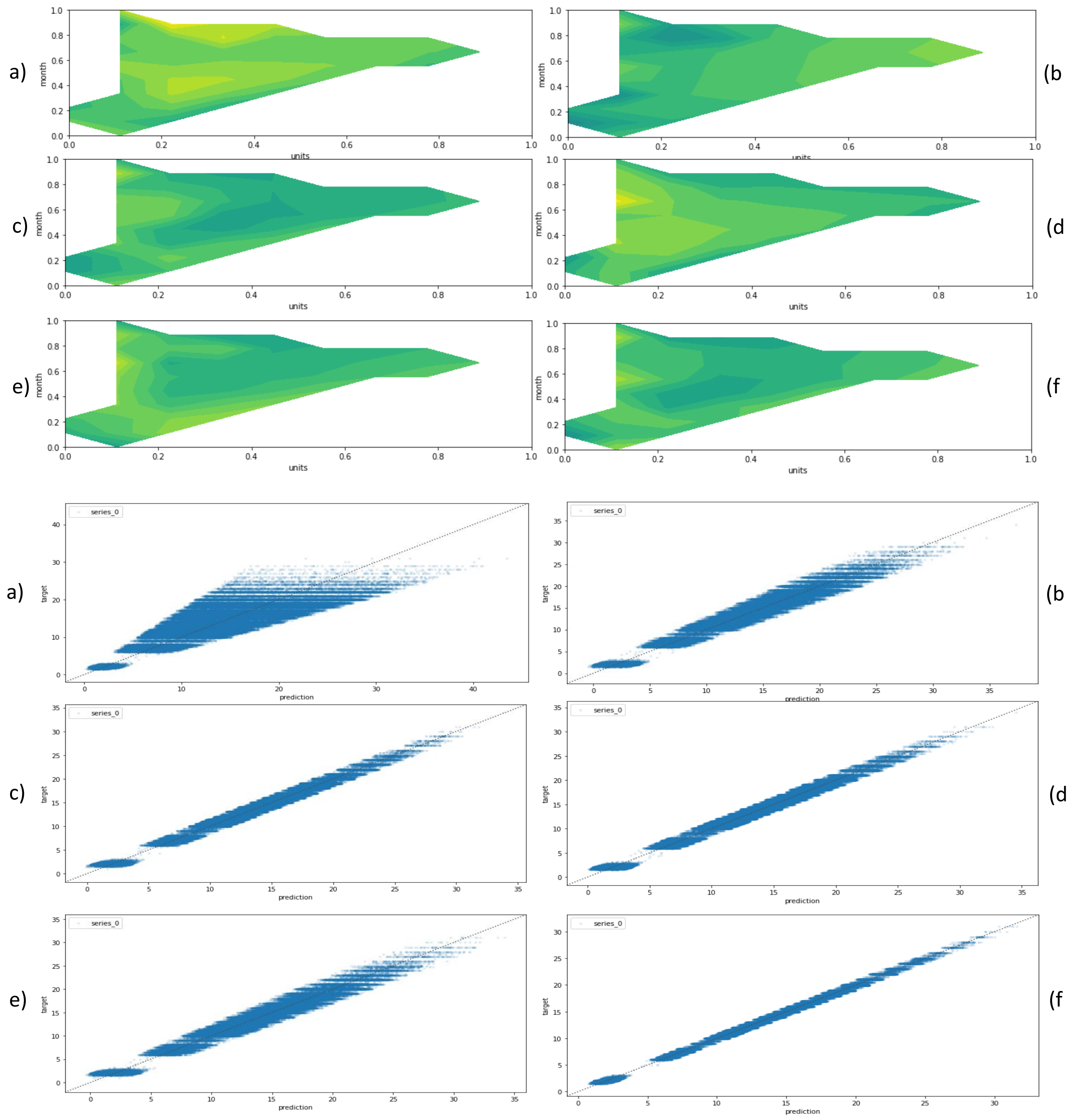


Image 4.4: Prediction comparisons, in order from top-left to bottom-right:
MLP (a), PMLP (b), Lat (c), CLat (d), DLat (e), PDL (f)

Chapter 5 - Conclusions and Remarks

The results shown that this model can be efficient in predicting mid/long time dependencies in business time series, but it still needs further studies in order to be employed in a real case scenario application, where data is not clean enough and it is subjected to many sources of errors.

Future studies could involve the implementation of an ensemble of Lattice models, such that each model could produce different kinds of errors, and overall, they could compensate one with the other. Also, it would be possible to increase the depth of the lattice architecture with some fully connected layers, in order to improve the embedding capacity of the model and learn more expressive activations. It would be possible then to add different kinds of input (e.g. Product ID, Seller ID, etc.), such that the model would have more context to work with.

Bibliography

[1] S.You, D. Ding, K. Canini, J. Pfeifer, M. Gupta – "*Deep Lattice Networks and Partial Monotonic Functions*" 2017 – you2017deep – arXiv https://arxiv.org/pdf/1709.06680.pdf

[2] A. Cotter, M. Gupta, H. Jiang, E. Louidor, J. Muller, T. Narayan, S. Wang, T. Zhu – *"Shape Constraints for Set Functions"* – PMLR – http://proceedings.mlr.press/v97/cotter19a/cotter19a.pdf