# Deep Learning Neural Networks

Enterprise Data Science and Analytics

**January, 2020**

# Online Sales Forecasting

Bruno Candeias[1], Clara Costa[2], David Oliveira[3], Henrique Pereira[4] & Manuel Oom[5]

[1]M20180313@novaims.unl.pt , [2]M20180330@novaims.unl.pt, [3]M20181430@novaims.unl.pt, [4]M20181395@novaims.unl.pt, [5]M20181431@novaims.unl.pt

## Abstract

The objective of the project is to present the prediction for the first 12 months of sales of a product after its release to the market, based on the historic sales of previous products. The typology of problem is sequence-to-sequence. Our approach is based on Recurrent Neural Networks (RNN) and different architectures will be presented with configurations using Simple RNN, Long Short-Term Memory Units (LSTM) and Gated Recurrent Units (GRU). Furthermore, in order to include the product features on the RNN, an Autoencoder layer will be included before, forcing a compressed latent representation of the original products input. The dates of product launch and its relation to other products in time launch were taken into account in the results, in order to integrate the time dimension into our analysis, enabling the comparison between products that were contemporaneous.

After a model comparison, GRU was the model with the lowest MAE and MSE. We have no evidence of overfitting.

## 1.Introduction

In the enterprise world, being able to forecast the corresponding sales accurately, supports many corporate decisions with large chances of optimizing its operations, namely: (i) having a performance benchmark, with the possibility of measuring different action impacts; (ii) enhancing its operational planning in logistics, stocks and HR departments; (iii) accurately deliver a financial plan that supports the budgets; (iv) help decide whether to pursue certain business scenario or not.

The goal of this project is to predict the monthly online sales of new products, based on its characteristics and the historical sales of similar products. This dataset belonged to a Kaggle Competition [1] where it was given a training dataset and a test dataset.

## 2.Methodology

Due to the nature of available data, we decided to use two different models: (1) one that uses static information (product features) with an Autoencoder (AE) architecture to reduce the dimension of the original 545 variables that characterize the different products and; (2) another

model with a sequence-to-sequence RNN architecture, that will learn the time series corresponding to every product, with the sales information after the release date.

The idea of making an AE-RNN hybrid architecture from the previous premisses was tempting and we, in fact, were able to construct the said model. But as we will see in further down this report, we had to abandon this promising architecture and resume other approach.

Focusing on the RNN model, we intended to train three different RNN configurations: (i) a Simple RNN; (ii) a Long Short Term Memory (LSTM) RNN; (iii) and a Gated Recurrent Unit (GRU) RNN.

To achieve this pipeline, we first trained and optimized the AE. Then we trained the 3 RNNs with a python data generator, composed by the output of the AE concatenated with the Time Series (TS). All 3 RNNs were further optimized. All these optimizations were Bayesian Optimizations, following the Bayes theorem of conditional probability, and applied to selected hyper-parameters for each model.

After constructing the above models with the best hyper-parameters selected, they were trained with the full train dataset. At this point we'll feed the test data and predict the monthly sales (outcomes).

As an extra, we wanted to implement a Baeysian Neural Network on all models in order to add to them some measure of uncertainty by using distributions as opposed to discrete numbers on the weights of the models. Unfortunately, due to technological constraints, it was not possible to deploy this neural network architecture. Nevertheless, we implemented a slightly similar approach of the Bayesian Neural Network, optimizing the models hyperparameters with Bayesian Optimization, as mentioned before.

All of our Neural Networks Models were constructed with dedicated personalised classes based on the Keras Functional API implemented by TensorFlow™. The decision to use TensorFlow™ instead of original Keras was due to: (i) ease of using GPU processing of tensors; (ii) ease of implementation of Bayesian Neural Networks with Tensorflow™.probability; (iii) ability to further customize Keras layers with functions of Tensorflow™ as needed.

Our methodology can be depicted in the following diagram (Figure 1):

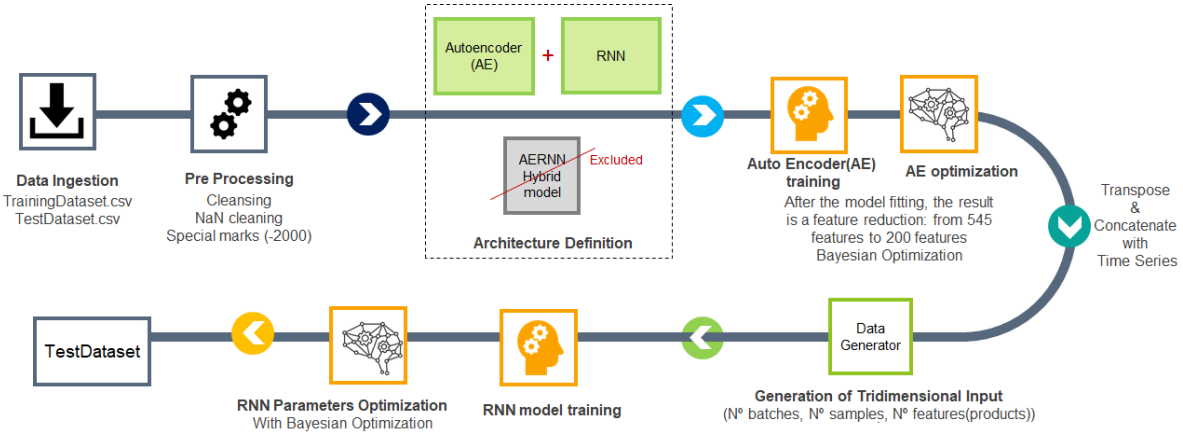


**Figure 1** - Project Methodology

## 2.1 Data ingestion

We used the files TrainingDataset.csv and TestDataset.csv. The training dataset contains four distinct groups of data, where each row represents a different product:

A. **12 features of *Outcome*** (Outcome_M1 until Outcome_M12) - reflects the monthly online sales for the first 12 months after the product launches, which will be our target features.

B. **2 features of Date type**:
   a. Date_1 : day number where the major advertising campaign began and the product was effectively launched;
   b. Date_2 : day number where the product was announced and a pre-release advertising campaign began.

C. **31 quantitative features (*Quan_x*)**

D. **513 categorical features (*Cat_x*)**

**Notes:**

1. There is no available description of the meaning of variables pointed in the points C and D;
2. Binary categorical variables are measured as “1” if the product had the feature and “0” if it did not.

## 2.2 Data Pre-Processing

Data pre-processing tasks were performed, in order to have all the data prepared to be used in our models.

***Cleansing***

- Transformation of NaN : we considered the replacement by zeros or by median values. The final scenario was to use, as model input, it was used the replacement with zeros;
- Filling the Date_2 feature (PreRelease) with Date_1 value (Release date), when there Date_2 presented Nan values;
- Conversion of date values to months (**Note:** the days were divided by 30, since the dates represent the day where the major advertising campaign began and the product

launched product - Date_1 - and the day where the product was announced and a pre-release advertising campaign began - Date_2);

- Calculation of the difference between pre-release and release date for each product.

Since the objective of the project is to predict the sales by using the corresponding product features, we proceeded with a separation of the features (A) with the monthly sales occurred (B).

***Time Series Creation***

Since the products do not have the same release date, we formulated a Time-Series with the corresponding sales months after the product release, Figure 2. On the outcomes columns we have the values of the sales (*Outcome_M01* represents the sales of the first month and *Outcome_M12* represents the sales of the 12th month). On the columns "Months with sales", we have the corresponding month of the sales in our time range (Example: for product 1, the first month of sales is the 83th month of our time range). Our time range goes from "month" 0 until the "month" 99.

| | | Outcomes | | | | | Months with sales | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Product | D_PreRelease | Outcome_ M01 | Outcome M02 | ... | Outcome M11 | Outcome M12 | 1 | 2 | ... | 11 | 12 |
| 1 | 82 | 67.0 | 10000 | ... | 500.0 | 500.0 | 83 | 84 | ... | 93 | 94 |
| 2 | 83 | 64.0 | 8000 | ... | 500.0 | 500.0 | 84 | 85 | ... | 94 | 95 |
| 3 | 59 | 49.0 | 5000 | ... | 500.0 | 500.0 | 60 | 61 | ... | 70 | 71 |

**Figure 2 -** Subset of monthly product outcome

| ... | Products | | |
|---|---|---|---|
| TimePeriod | 1 | 2 | 3 |
| 83 | 10000.0 | 0.0 | 0.0 |
| 84 | 3000.0 | 8000.0 | 0.0 |
| ... | | | |
| 93 | 500.0 | 500.0 | 0.0 |
| 94 | 500.0 | 500.0 | 0.0 |

**Figure 3 -** Time-Series table

Afterwards, we transposed the dataset, in order to obtain a "vertical Time-Series" filled with the matched product sales in that corresponding month (Figure 3). With this adjustment, we obtained the products outcome per row in the months where the products had a pre-release or sales, ordered timely. For instance, product 1 outcome on the month 83 is 10 000. In order to identify a pre-release month of each product we used a negative marking (-2.000) hoping this can be a 'tail-tell' to the Neural Network that this occurred and maybe act as an influencer of contemporaneous neighbour product's outcomes.

## 2.3 Architecture Definition

In order to define the architecture to use, we decided to test each model isolated: first the AE, then an RNN. This served as a preamble to our design of a Hybrid model (AE-RNN), where we incorporated simultaneously the output of autoencoder and the outcome time series in the same Neural Network architecture. Unfortunately, due to technical constraints, we were not able to run training sessions with on-the-fly join of the data of products and the TS. As we faced problems with the hybrid model, we did not proceed with its further implementation, even though, in our perspective would be the most appropriate architecture, theoretically. The final approach followed was: first use Autoencoder, followed by incorporating it in the RNN, in a

sequence manner. We deployed three different models, namely: Simple RNN, LSTM and GRU, as mentioned before, and optimized each component of the models with Bayesian optimization.

**RNN:** This architecture is a type of neural network architecture that performs well in sequential or contextual data, since it takes into account the previous steps of the time-series, therefore the interdependency of the data (that can be modulated to be unidirectional or bidirectional). However, these models often encounter vanishing & exploding gradient problems, but these can be overcome with the selection of appropriate hyper_parameters.

The following architecture (Figure 4) was used as the baseline for our project.

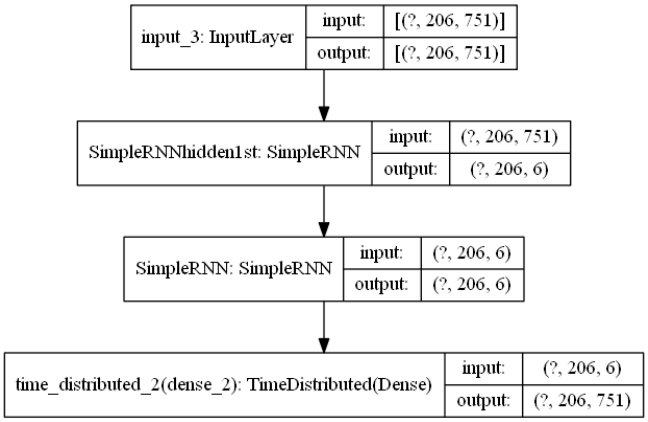


**Figure 4 -** Simple RNN model example

The last layer (TimeDistributed) will allow the Network to tackle sequence-to-sequence problems. The RNN layers (between the Input Layer and the TimeDistributed) will be interchanged with SimpleRNN, LSTM, and GRU to construct the 3 different kinds of RNNs we explained before.

**LSTM**: Long Short-Term Memory Units are another type of RNN, that is often used to solve the vanishing & exploding gradient problem by achieving memory replication effect and, therefore, keep the interdependency even if the information are further away in the corresponding sequence. In order to perform that way, LSTM uses three different gates (input, output and forget gate) and preserves the error in a cell state, which opens or blocks information dependent on its own weights and preserving the important information, which will be called whenever necessary.[4]

The architecture presented in Figure 5 represents an example of our LSTM model.

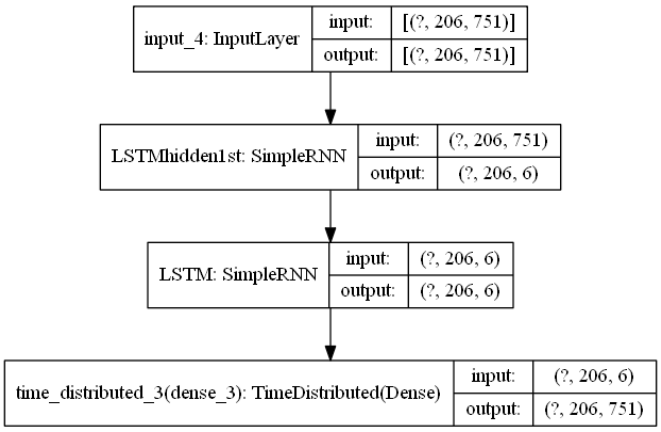


**Figure 5 -** LSTM model example

**GRU**: Gated Recurrent Units are a type of RNN that is similar to LSTM inner architecture, which is often used to solve the vanishing & exploding gradient problems. However, unlike LSTM, it consists of two gates: update gate and reset gate (Figure 6); and does not maintain an Internal Cell State, instead it uses a hidden state to transfer information [4]. This RNN type, GRU, is typically more efficient than LSTM, presenting both similar results. A GRU is basically an LSTM without an output gate, which therefore fully writes the contents from its memory cell to the larger net at each time step. Figure 7 represent a GRU model example

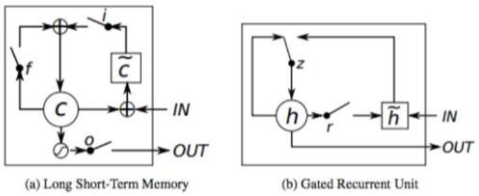


**Figure 6** - Illustration of Long Short Term Memory (LSTM) and Gated Recurrent Unit (GRU)[14]

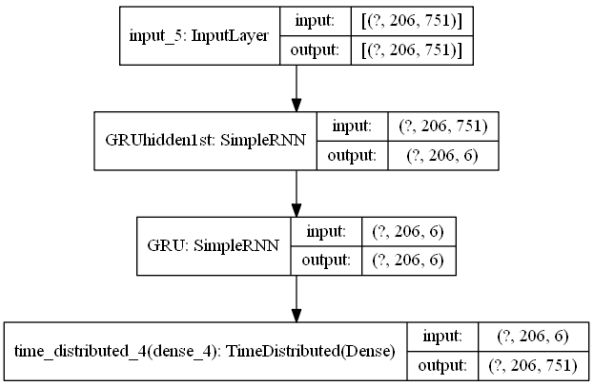


**Figure 7 -** GRU model example

**Bayesian Neural Network**: Firstly, our group decided to try to implement a Bayesian neural network, different than the previous, where weights are fixed and biased. In a Bayesian Model the weights and biases have a probability distribution attached to them. This leverages the uncertainty of the several outputs retrieved, by the possibility of being able to create confidence intervals, and probabilities of certain output being pinpointed or have uncertainty

in its value. In the current context, if the probability distribution is wide, it could mean that the model has not seen the product before and therefore it has a low confidence interval, meaning the result is not to be taken for granted. Bayesian model works best in classification problems when trained with large dataset, which, in our case would be expected to have lower accuracy metrics if compared with the other models deployed [2][3].

**Hybrid model (AE-RNN):** Using the Autoencoder and the RNN model, an integrated model was developed. The model followed the diagram presented in Figure 8. Unfortunately, we were not able to concatenate both input sources simultaneously: product sales time-series (from the data generator) and the product features reduction (encoder output).

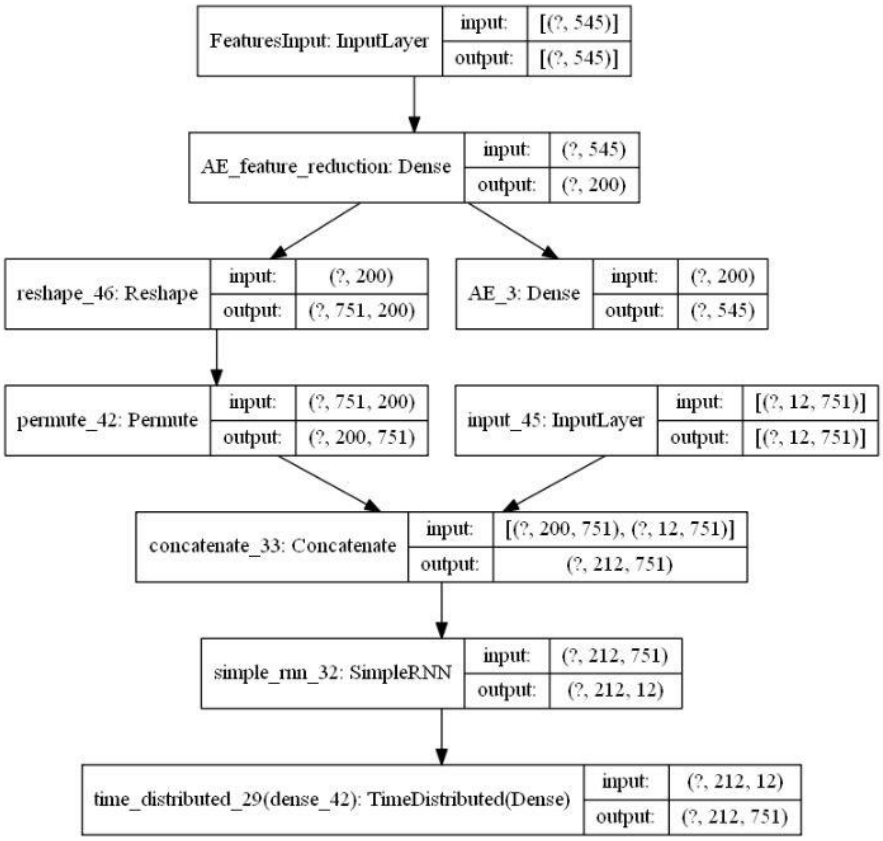


**Figure 8** - Hybrid model AE-RNN

## 2.4 Autoencoder Training and Optimization

Our dataset has 545 individual features to characterize each product, which are a computational significant number of features to work with. In order to reduce the dimension and incorporate the most significant variable relations, we proceeded to implement a neural network Autoencoder model, as shown in Figure 9.

From the baseline, we constructed an hyperparameter grid (Table 1) in order to find optimal combination of parameters that minimize the loss function using a Bayesian Optimizer.

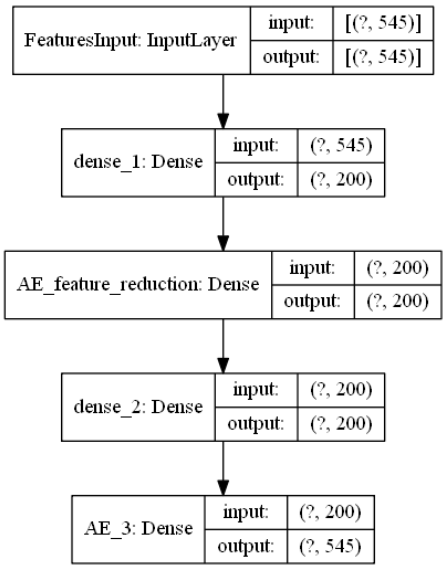


**Figure 9** - Autoencoder model

**Table 1 -** Hyperparameters

| Learning Rate | # of hidden layers | Factor of hidden layer neurons | Encoding dimension | Kernel initializer | Activation function | Optimizer | Dropout layer factor | Batch Norm. before encoder |
|---|---|---|---|---|---|---|---|---|
| **[0,001 to 0,1]** | **[0, 3]** | **[1, 2]** | **[100, 300]** | **[he_normal, he_uniform]** | **[relu, tanh, selu, linear, LeakyReLU]** | **[Adam, Nadam, RMSprop]** | **[0 to 0,49]** | **[True, False]** |

All fitting runs were implemented using essentially 2 callbacks:

I. *EarlyStopping* - prevent overfitting and stopping training to go use all epochs defined;

II. *ReduceLROnPlateau* - reduce the learning rate whenever a plateau is found in the function of loss, allowing further minimization of loss by decelerating learning (minimum lr = 0,00001) to find new local minimum on said function.

The AE for itself had 500 epochs predefined, while the Bayesian Optimization had a sequence of 60 exploratory runs and 60 iterations on best local minimas of the loss function. Although this was computationally expensive, the results were fast and satisfactory.

As it may be seen in Table 2, the metrics to evaluate the performance of the model improved significantly.

**Table 2 -** Metrics comparison for the Autoencoder, with and without Bayesian optimization

| Model/ Parameters | Validation MAE (loss) | Validation MSE | Validation Cosine Similarity |
|---|---|---|---|
| **Autoencoder baseline** | **0.0470** | **0.0414** | **0.6806** |
| **Autoencoder Bayesian Optimized** | **0.0086** | **0,0046** | **0,9698** |

The best hyper-parameters found were the following:

**Table 3 -** Best hyper-parameters

| Learning Rate | # of hidden layers | Factor of hidden layer neurons | Encoding dimension | Kernel initializer | Activation function | Optimizer | Dropout layer factor | Batch Norm. before encoder |
|---|---|---|---|---|---|---|---|---|
| **0.004768** | **0** | **1** | **280** | **he_normal** | **selu** | **Adam** | **NA** | **False** |

There was no evidence of overfitting on the AE model with best combination of parameters.

## 2.5 Data Generator

We implemented a data generator based on the code presented in class 6 ("Mystery Challenge"), which takes an array of standardized data and yields batches of data from the recent past along with the targets in the future, which in our case are the monthly outcomes. Our data generator takes as input arguments the following arguments:

- ***aux_data* -** data to be used as header for each batch of samples and targets;
- ***min_index* -** minimum index in the data array that delimits which timesteps to draw from;
- ***max_index* -** maximum index in the data array that delimits which timesteps to draw from;
- ***shuffle* -** flag that indicates whether to shuffle the samples or draw them in chronological order.

Finally, using yield the function returns a generator, instead of an array of arrays, that will be fed to an fit_generator method of our NN models.

## 2.6 RNNs Training and Optimization

The 3 RNN models were trained with the output of the encoder of the optimized AE, merged with the outcome time-series. The 3 model's hyperparameters were optimized like the AE, with a Bayesian Optimizer. To do that we constructed a new parameter grid to guide the optimization. The results of the optimization are presented in Table 4. Finally, the performance of the models were compared between them in two scenarios:

1. Baseline results;
2. Results after the Bayesian optimization.

**Table 4 -** Hyperparameters for RNNs

| Learning Rate | # of hidden layers | Factor of hidden layer neurons | Activation function | Activation function TimeDistributed | Optimizer |
|---|---|---|---|---|---|
| **[0,001 to 0,1]** | **[0, 3]** | **[1, 2]** | **[relu, tanh, selu, linear]** | **[tanh, selu, linear]** | **[Adam, Nadam, RMSprop]** |

All fitting runs were implemented using essentially 2 callbacks:

III. *EarlyStopping* - prevent overfitting and stopping training to go use all epochs defined;

IV. *ReduceLROnPlateau* - reduce the learning rate whenever a plateau is found in the function of loss, allowing further minimization of loss by decelerating learning (minimum lr = 0,00001) to find new local minimum on said function.

The RNN for itself had 5 epochs predefined (training is computationally expensive and long), while the Bayesian Optimization had a sequence of 10 exploratory runs and 10 iterations on best local minimas of the loss function. The results were satisfactory.

As it may be seen in Table 5, the metrics to evaluate the performance of each RNN model differ significantly.

**Table 5 -** Metrics comparison of models performance, with the Bayesian optimization

| | Results after Bayesian Optimization | |
|---|---|---|
| **Model/ Parameters** | **Validation MAE (loss)** | **Validation MSE** |
| SIMPLE RNN | 0,0402 | 0,0084 |
| GRU | 0,0338 | 0,0080 |
| LSTM | 0,0347 | 0,0081 |

The model with the best MAE and MSE was the GRU Model. None of the models showed signs of overfitting.

# 3. Results

The model with the best performance, the GRU model, was used to predict the sales of the TestDataSet given in Kaggle and mentioned in the data ingestion section. As stated, the model was trained on the full training dataset and then was used to predict the outcomes.

Since our test data has different dimensions (in the 'feature' axis on the TS data generator) we had to allocate the new products on the columns of best matching old products. This was done by calculating the cosine similarity score between the vector encoding the features of the new product and the same vector for the old products. We were able to predict some monthly outcomes, but not all of them, since our time-series generator was configured with a lookback of six months.

# 4. Conclusion

With this model, companies will be able to predict, at this moment, the sales of the first half year with the champion model. Additionally, companies will be able to test if a certain product characteristic increases or decreases the sales.

Taking our example to "real life" business case - a cell phone company, for example, - where the characteristics of the products are known, a company could not only predict the sales of

the first 12 months (if the time-series generator was trained with a lookback of that amount of months), but could also use its historical data to predict if a cell phone with “Face recognition unlock” (or other characteristic) will increase or decrease the sales. Our models may auxiliate a company to identify characteristics that the clients do not sought after and the ones that the clients value the most. With the sales forecast concluded, the company will be able to optimize its logistics (production, distribution and stock) and, at the same time, its HR allocations and reinforcements (temporary or not). With a sales prediction, the company can also improve its cost and projects that supports the budgets and the sales like marketing campaigns and loans for investments (if needed).

Our future steps will be the implementation of Bayesian Neural Network on our best models to better advice on uncertainty of predictions. Other variant of Bayesian Neural Network to consider, would be Variational AutoEncoders (VAE).

# References

[1] https://www.kaggle.com/c/online-sales/data

[2] Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning; Yarin Gal, Zoubin Ghahramani 4th October 2016

[3] Adapted from: https://towardsdatascience.com/making-your-neural-network-say-i-dont-know-bayesian-nns-using-pyro-and-pytorch-b1c24e6ab8cd

[4] Adapted from:https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

[5] Auto encoder model for feature reduction adapted from: https://blog.keras.io/building-autoencoders-in-keras.html

[6] Clustering layers inspired by https://www.frontiersin.org/articles/10.3389/fgene.2018.00585/full

[7] TSNE model and plot adapted from https://www.kaggle.com/jeffd23/visualizing-word-vectors-with-t-sne

[8] Based on the Mystery_Challenge_Template (lecture 9th-15th December)

[9] Combining static and dynamic data
https://dlpm2016.fbk.eu/docs/esteban_combining.pdf

[10] Combining static and dynamic data
https://stackoverflow.com/questions/52474403/keras-time-series-suggestion-for-including-static-and-dynamic-variables-in-lstm,

[11] https://blog.nirida.ai/predicting-e-commerce-consumer-behavior-using-recurrent-neural-networks-36e37f1aed22

[12] https://lilianweng.github.io/lil-log/2017/07/22/predict-stock-prices-using-RNN-part-2.html

[13] https://pathmind.com/wiki/lstm