Advanced neural networks architectures research – forecasting recommendations

Santiago Rilo Sánchez, Student of the Master´s Degree in Smart Industry at ICAI, Comillas Pontifical University,25 Alberto Aguilera street, Madrid, Spain, santiago.rilo@alu.comillas.edu)

***Resumen*—En este proyecto se ha realizado una comparación entre diversas arquitecturas de red neuronal para evaluar el efecto del tipo de arquitectura en un problema de predicción de generación de energía eólica. El trabajo incluye un estado del arte extenso sobre la materia y un caso práctico. Se concluye que la red neuronal GRU es la más adecuada, que la CNN se debe utilizar en sets de datos extensos y que existen alternativas eficaces a la función de activación ReLU.**

***Abstract*--This project contains a comparison between different neural network architectures with the goal of evaluating the effect a given architecture has in a wind power generation forecasting problem. It includes an extensive state of the art and a use case. The project concludes the GRU neural networks are successful at tackling timeseries forecasting, the CNN should be used for large datasets and that several alternative activation functions can outperform the ReLU.**

I. Introduction

The main purpose of this final thesis is to come up with a series of recommendations to select the right architecture for a neural network for a specific problem, focusing on timeseries forecasting problems. The project involves an extensive state-of-the-art section in which a review of current and past trends is carried out in order to explore the most popular neural network architectures. The second part of the project involves selecting, designing and analyzing networks that were tested using a real dataset from a wind energy forecasting competition containing wind power generation data.

II. State of the Art

In this section some of the most common neural network architectures are introduced, exposing their inner workings, most common problems and giving an overview of its most extended applications in the fields of machine learning and deep learning.

A. Multilayer Perceptron

This was the first neural network model developed. It is composed of perceptrons. A perceptron is a model for supervised learning, this means the model is given pairs of input-output couples in the training process, of binary classifiers. The neuron is given a set of training data that allows it to adjust the inner parameters to find the best fit possible.

A multilayer perceptron is made up of an input layer, hidden layers and an output layer. Each layer is composed by a previously fixed number of neurons. A hidden layer consists of neuron nodes stacked in that do not directly connect with inputs and outputs.

The dimension of the input layer depends on the dimension of the training data. The number of neurons in the hidden layer is one of the hyperparameters of the model and should be decided by the user. The dimension of the output layer depends on the application.

A multilayer perceptron is usually trained using a backpropagation algorithm [1]. It is a machine learning training iterative algorithm that consists of adjusting the weights of the model to minimize the difference between actual output and desired output. The weights of the network are updated using gradient descent computing the derivative of the error with respect to the weights.

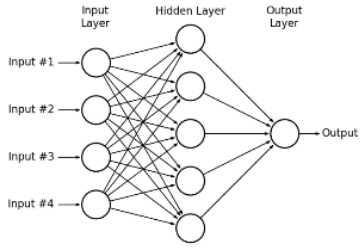


Figure 1: Multilayer `perceptron diagram

B. Convolutional Neural Networks

Convolutional neural networks are a type of feed-forward artificial neural networks that are mainly applied to machine vision. One main task of CNNs is to reduce the images into a form which is easier to process, without losing critical information to obtain an accurate prediction. This is referred as feature extraction or feature recognition. A usual requirement in CNNs is that they must be computationally viable in massive datasets.

CNNs can be broadly divided into two parts, the feature recognition part and the classification part. The classification part is usually similar to a multilayer perceptron. It connects the detected visual features to the desired output.

Some common types of layers that are present in convolutional neural networks are:

- Convolutional layers: a filter or Kernel is applied to a matrix or 2D input. This Kernel is a moving window and computes calculations that result in another matrix, usually of a smaller size, although a bigger size or the same size can also be obtained. The objective of the convolution operation is to extract high-level features

such as edges from the input image. Several features can be applied simultaneously [2].

- Pooling layer: the result of a convolutional layer is an activation map. The objective of pooling layer is to provide a non-linear downsampling for those activation maps.
- Flattened Layer: the purpose of this part of the network is to condense the information into a one-dimensional array to pass it to the Fully Connected Layer.
- Fully Connected Layer: It receives the flattened image as a column vector. This flattened output is fed to a feed-forward neural network. Over the training process this part of the network learns to distinguish between dominating and low-level features.

C. Recurrent Neural Networks

Recurrent neural networks can send information over time-steps. Recurrent neural networks can store information of previous steps to modify its behaviour in the future.

They are usually referred as systems with memory. This is due to the way they compute their output, using not only the inputs of each sample in isolation, but also some internal variables that are influenced by the values of the inputs of previous timesteps.

In order to visualize a RNN it is good practice to first observe the perpendicular vision of an MLP presented in Figure 2, comparing it with the "flattened" version shown below in Figure 4. In the last example, we can imagine one of the two perpendicular directions as normal to our vision plain. The loops represent the recurrent connections.

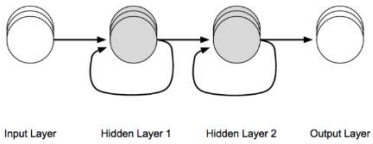


Figure 2: Flattened diagram of recurrent neural network

Recurrent neural networks commonly use backpropagation through time for the training process. Backpropagation through time is fundamentally the same idea as standard backpropagation, a chain rule is applied to calculate the derivatives based on the connection structure of the network. However, the loss is calculated in a forward motion before calculating the gradient.

There are several variants that differ on their cell structure:

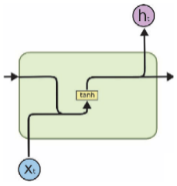


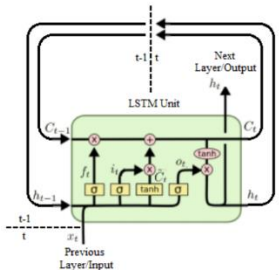
Figure 3: Inner cell structure of vanilla recurrent networks

1) Vanilla Recurrent Neural Networks

The simple RNN cell is a basic model in which there is a multiplication of the input by the previous output.

Vanilla RNNs are a prime example of a model affected by the two most prominent problems related to the training phase of recurrent neural networks.

1. Vanishing gradients: The vanishing gradient problem takes place when the derivatives are smaller than one, and a as a result the update speed of the weights turns very slow. This problem might grow in each iteration, producing an almost zero gradient and preventing the model to train correctly.
2. Exploding gradients: due to the chain calculations, if several partial derivatives turn out to be higher than one, the first weights of the network will have a large gradient which will modify greatly the weights value. It could make the computer crash if the values are greater than the maximum the datatype supports in that memory spot.



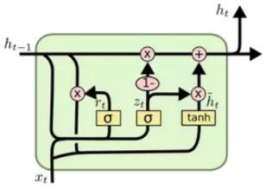
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$
$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 4: Inner cell structure of Long-Short Term Memory Cells

2) Long-Short Term Memory Cells

The LSTM cell has what is called a cell state. This is usually referred as the memory of the cell and it passes across the cell (represented as the top way). It allows information from earlier timesteps to affect the output, reducing the effect of the short-term memory. It also has a hidden state that comes from previous timesteps and that is represented with the letter h in the diagram. As the cell state goes through the cell, information is added or removed from it by the gates.

The gates are different neural networks that decide which information is allowed on the cell state. LSTM cells have a forget gate (first equation), an input gate (second and third equations) and an output gate (fifth equation). The forget gate decides whether if information from the hidden state and from the previous gate should be kept or thrown away. The input gate decides how relevant information from the current step is, it will transform it into a number between 0 and 1 (sigmoid) and multiply it for a number between -1 and 1 to regulate it (tanh in this case). The output gate decides what the next hidden state should be. The hidden state is also used for predictions. It uses information from the previous hidden state, the input and the cell state.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figure 5: Inner cell structure of Gated Recurrent Units

3) Gated Recurrent Units

They were a posterior development to the LSTM. It intended to be similarly powerful but lighter than LSTM cells. It is considered a valid alternative to LSTMs as it is more powerful than vanilla RNNs and it is comparatively less costly to train. The GRU cell has no cell state, and the gates directly modify the hidden state and use it to pass information. GRU cells have two gates: the reset gate (second equation) and the update gate (first

equation). The update gate works in a similar way as the input gate in an LSTM. It decides what information to keep or discard. The reset gate decides how much past information to forget.

The more complex the individual cell becomes, the more matrixes are involved in solving its equations and the heavier the computational needs become. Vanilla RNNs are simple but they struggle with passing information after a few timesteps due to variability in the gradients. Both GRU and LSTM are widely used nowadays. GRU is significantly lighter to compute and performs better in some small size datasets [3]. LSTM networks need to train numerous parameters, but they are robust.

4) Bi-directional RNNs and Bi-directional LSTMs

Bi-directional RNN structures are models in which the same input is introduced in order and in reverse to different first layers of the neural network.

In a Bidirectional LSTM the building blocks of the network are LSTM memory cells. Deep BLSTM networks are obtained by stacking layers in between the input and the output. InBRNN, backpropagation through time must be computed separately on both the forward andthe backward networks.

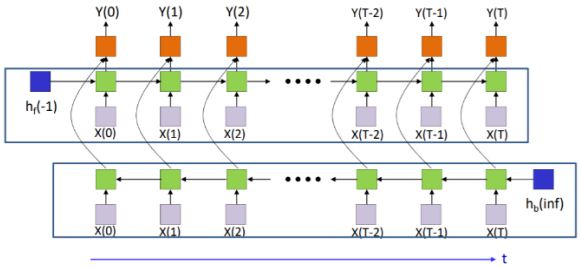


Figure 6: Simple bi-directional RNN network [4]

5) Hopfield Networks

Defined as a "loopy binary network with symmetric connections" [4]. These networks are made up by nodes and connections between the nodes. Each node is updated by a calculation that involves the value of the node (typically 1 or -1) and the values of the connections to the surrounding nodes multiplied by the values of the surrounded nodes. If the result is contrary to the sign of the node value (threshold), the value of that node is flipped (now it has the opposite value). The energy of the network is a property of Hopfield Networks that can be calculated taking into consideration all nodes and all connections [5].

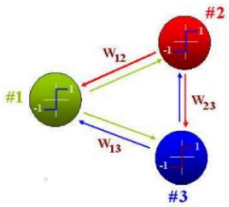


Figure 7: Three neuron Hopfield Network [5]

6) Boltzmann Machine

A Boltzmann Machine is a Hopfield Network that has N + K neurons, where N are the visible neurons (the ones that will store the actual patterns of interest and are form a HN) and K are hidden neurons.

The value of each node is not deterministic but based on a probability distribution function. Ina Boltzmann Machine the probability of generating a "visible" vector (the desired output) is defined in terms of the energies or joint configurations of the visible and hidden units.

Boltzmann machines are limited due to an extensive training time and therefore they are only suitable for small problems [6]. To solve these issues, the alternatives are Restricted Boltzmann machines [7], Deep Boltzmann Machines [8] and the Helmholtz machines [9]. Restricted Boltzmann Machines have been used in regression and forecasting applications.

7) Deep Believe Networks

A Deep Belief Networks is a generative graphical model. Generative means that it not only focusses on the distribution of our output variable given our input variables, but that it can also learn the distribution of the inputs [10].

Back-propagation is considered the standard method in artificial neural networks to calculate the error contribution of each neuron after a batch of data is processed. However, it comes with its own problems and limitations and Deep Belief Networks were a proposed solution for it. The main problems of back-propagation are that it requires labelled data (when many data sources are unlabelled), the learning time does not scale well when you had multiple hidden layers and the training process can stop in a poor local minima.

Deep Belief Networks contain many layers of hidden variables. Each layer captures high-order correlations between the activities of hidden features in the layer immediately below, performing a de facto feature extraction. The superior two layers of DBNs form a restricted Boltzmann Machine. The lower layers form a directed sigmoid belief network [11].

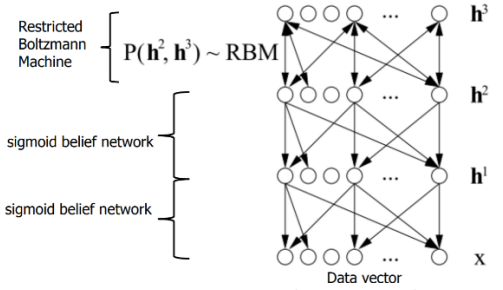


Figure 8: Representation of Deep Belief network

8) Deep Auto-encoders

An autoencoder is a type of neural network designed to learn data codings in an unsupervised manner. The input is the same as the output. Autoencoders are able to compress the input into a lower dimensional code and then reconstruct the output from this representation [12].

An autoencoder consists of three different parts, an encoder, a code and a decoder. The encoder processes the inputs and produces the code, while the decoder uses the code to reconstruct the input. In order for an autoencoder to run we need a loss function compering the output with the target. Autoencoders are trained using back-propagation.

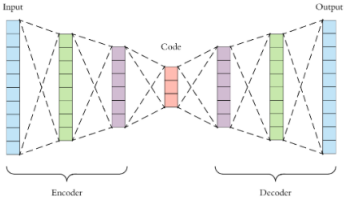


Figure 9: Diagram of an auto-encoder [13]

D. Activation functions

Activation functions play an essential role in an artificial neural network. They are the mathematical equivalent of the electrical potential that builds up in biological neurons which then fire when a given threshold is reached. Its job is determining the magnitude of the output after receiving the inputs of the neurons multiplied by the weights as an argument (potentially internal variables too, as in recurrent neural networks).

The main activation functions used in the case study are:

1) Rectified Linear Unit

The rectified linear unit is one of the simplest non-linear functions. It returns zero as an output if the input value is below zero. It returns the input when the input is above zero [14]. It is computationally lightweight, and it is very extended for deep learning applications, especially when applied to the hidden layers.

2) Exponential Linear Unit

It is very similar to the ReLU function, but it avoids the zero values when the inputs are negative.

$$R(z) = \begin{cases} z & z > 0 \\ \propto * (e^z - 1) & z \leq 0 \end{cases}$$

3) Scaled Exponential Linear Unit

Evolves on the idea of the ELU function, including an offset with an extra parameter.

$$R(z) = \gamma \begin{cases} z & z > 0 \\ \propto * (e^z - 1) & z \leq 0 \end{cases}$$

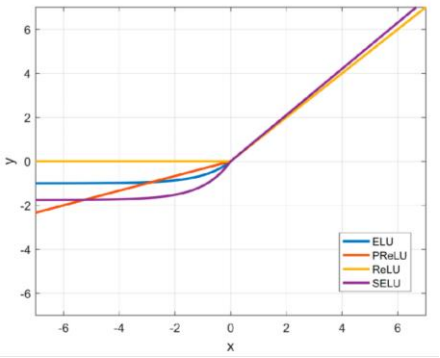


Figure 10: Visual representation of ReLU, ELU and SELU

E. Optimization Algorithms

Optimization involving neural networks means non-convex optimizations. In order to tackle this problem, there are two main approaches, the first order optimization algorithms and the second order optimization algorithms. First order optimization algorithms minimize a loss function using the values of the gradient with respect to the parameters. Second order algorithms compute the second order derivative to minimize the loss function.

The main optimization algorithms present in the use case will be highlighted:

1) Gradient Descent

It is the most common technique and the foundation to the ones developed in this section. It uses the gradient of the cost function in order to update the parameter and obtain a new configuration that leads to a reduced loss [15].

$$\theta = \theta - \eta \cdot \nabla \cdot J(\theta)$$

2) Momentum

Momentum is an added term to the optimization equation. It attempts to replicate the concept of momentum in physics, as an object slides down a slope its velocity increases and it becomes harder for it to change directions [15]. It results in faster and more consistent optimization with less oscillations.

3) Adam

It stands for adaptive moment estimation. It is one of the most popular optimization algorithms nowadays [16]. It is a combination of standard Momentum with another algorithm called RMSprop. Adam computes adaptative learning rates for each parameter. It calculates an exponential decaying average for past squared gradients, as well as an exponential decaying average of past gradients.

Estimate first momentum:

$$v_i = \rho_1 \cdot v_i + (1 - \rho_1) \cdot g_i$$

Estimate second momentum:

$$r_i = \rho_2 \cdot v_i + (1 - \rho_2) \cdot g_i^2$$

Update parameters:

$$\theta_i = \theta_i - \frac{\varepsilon}{\delta + \sqrt{r_i}} \cdot v_i$$

4) Nadam

It was proposed to improve the Adam algorithm. It is a combination of Nesterov momentum and Adam.

The only significant change is that the momentum is computed using interim parameters, instead of current parameters.

Estimate first momentum:

$$v_{i+1} = \rho_1 \cdot v_i + (1 - \rho_1) \cdot g_{NAG}$$

Estimate second momentum:

$$r_i = \rho_2 \cdot v_i + (1 - \rho_2) \cdot g_{NAG}^2$$

Update parameters:

$$\theta_{i+1} = \theta_i - \frac{\varepsilon}{\delta + \sqrt{r_i}} \cdot v_{i+1}$$

F. Loss functions

A loss function (or cost function) is a measure of how the neural network performs with respect to the ideal behaviour given the application. Therefore, it is dependent on the end use and the output of the network. It can be a function of the output, the input, the weights and biases.

The loss functions utilized for the use cases are exposed below:

1) Mean squared error

Also known as maximum likelihood or sum squared error. Common in regression and forecasting problems, it has the issue that individual extreme values can significantly taint the resulting value.

$$C_{CE}(W, B, S^r, E^r) = \frac{1}{n} \sum_j \left(a_j^L - E_j^r\right)^2$$

2) Mean Absolute Error

This measure of accuracy is frequently found in forecasting and regression problems. It measures the average magnitude of the errors in a set of predictions, without considering their direction.

$$C_{CE}(W, B, S^r, E^r) = \frac{1}{n} \sum_j \left|a_j^L - E_j^r\right|$$

III. Case Study

A. Problem Statement

The dataset to be analyzed comes from a machine learning contest and it contains recorded values of wind power generation with 90 different input variables. The available explanatory variables include wind speed, wind direction and temperatures in a given area. Overall, this problem accounts with over 35,000 samples.

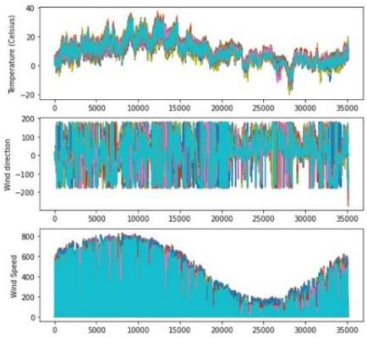


Figure 11: Different types of inputs in the forecasting problem

In Figure 12 the three different types of input variable are visible. In the top chart the temperatures are displayed in degrees Celsius. In the second chart the display shows the aggregate of the wind direction variables. This can oscillate from -180º to 180º. In the last graph the wind speed variables are shown and how they evolve through time.

As a way to test the value added with the trained models, they are compared with two benchmarking algorithms.

The first benchmark (from now one just benchmark) is a simple one-day forecasting window using the value of the previous datapoint. It is a common proxy used in several industries such as the financial services and it is an easy way to set a target for any forecasting model.

$$y_{pred}[t] = y[t-1]$$

The one-timestep lag error with outliers is shown in Table 4. The one-timestep lag error without outliers in shown in Table 6.

The second benchmark model (from now on MLP) is a simpler neural network, a multilayer perceptron. One of the objectives of the project is to show how this more complex models contribute to a better explanation of a time series forecasting problem. The parameter for batch size was selected at 50 and the number of iterations was initially set to 30. This are common initial values in similar applications. The MLP will change the structure in every experiment. In the first ones it will have the same structure as the GRU and MLP, layer and neuron wise.

The parameter for batch size was selected at 50 and the number of iterations was initially set to 30. This are common initial values in similar applications.

The training-validation split is 80% training, 20% validation. The validation set corresponds with the last part of the dataset in order, given the sequential nature of the problem.

The error measures showed in every single one of the following experiments are the errors of the validation dataset, as it shows if the models are capable of generalizing and correctly leverage the relationships between outputs and inputs. The optimal model would be the one with lowest validation error.

B. LSTM and GRU networks comparison

The cell state present in the LSTM algorithm is a powerful feature that allows it to control the information flow from past timesteps to influence the output. However, the GRU was created under the believe that a simpler algorithm could not only improve the running time, but also the performance scores of the LSTMs.

1) First LSTM-GRU Test

The first experiment was carried out using a random neural network architecture with 3 layers and 20 neurons per layer for each case. The epochs were set to 20 while the batch size was fixed at 50. These values are common initial values and will be changed in the optimized models.

The rationale behind the experiment was to check the behaviour of both algorithms with the same structure and testing out common alternatives of activation functions and optimization functions. Tanh, maxout and sigmoid were used to.

Code	Algorithm	Opt. algorithms	Activation	Average MSE	Local Ranking	Total Ranking
1	LSTM	Adam	ReLU	249.20	2	5
2	LSTM	Adam	SeLU	247.42	1	4
3	LSTM	Nadam	ReLU	351.16	4	8
4	LSTM	Nadam	SeLU	266.13	3	6
5	GRU	Adam	ReLU	237.22	3	3
6	GRU	Adam	SeLU	221.72	2	2
7	GRU	Nadam	ReLU	279.10	4	7
8	GRU	Nadam	SeLU	211.69	1	1

Table 1: Results of the first test

The average validation MSE value in Table 1 refers to the MSE score achieved by the different algorithms average over the five simulations carried out.

The main conclusion of this initial test is that the SELU activation function is a promising rival for the ReLU, outperforming it in several cases. At this level of complexity, the GRU algorithm outperformed the LSTM algorithm in 3 out of 4 cases. No definitive conclusions are driven from the comparison between the Adam and Nadam optimization algorithms.

2) Second LSTM-GRU Test

As learned in the previous section, the activation functions do impact in a notable way the performance of the algorithm. Therefore, it was decided to keep the SELU and the ReLU options and to include the ELU as an alternative as well.

Given that Adam is simpler and more common in the literature than Nadam, and that there was no significant difference in behaviour found in the first LSTM-GRU test, the Adam algorithm was selected as the only choice for the second LSTM-GRU test.

In this case, the number of simulations was 30 to make the results more reliable with a larger sample. This way there is a higher statistical certainty of the veracity of the results, as the results are an aggregate and not just a single data point. All the networks had 3 layers and 20 neurons, as in the previous experiment. Epochs where kept to 20 and batch size to 50.

Code	Algorithm	Activation	Average MSE	Local Ranking	Total Ranking
1	LSTM	ReLU	266.49	2	5
2	LSTM	eLU	256.28	1	4
3	LSTM	SeLU	310.37	3	6
4	GRU	ReLU	201.41	2	2
5	GRU	eLU	207.93	1	1
6	GRU	SeLU	226.49	3	3

Table 2: Results of the second test

The results of this test indicate that, at this level of complexity the GRU algorithm is more powerful than the LSTM algorithm. The ReLU activation function turned out to be better than the SELU, however, the simpler ELU was the best activation function referring to its average MSE validation values.

3) Optimized algorithms and analysis with outliers

In order to compare the best performing versions of the MLP benchmark, the GRU and the LSTM, a hyperparameter optimizator was used. In this case, it is a Bayesian optimizator, which creates a statistical model that evaluates the predictors as a black box. By updating the believes after each tested point, via multiple iterations, it trains the model in each iteration with the hyperparameter configuration which should have the lowest error predicted by the statistical model [17].

Using the Bayesian enables to find optimized solutions faster than by manually fixing several parameters. The tuned hyperparameters in this experiment were the model, activation function, number of layers, neurons per layer, learning rate and optimization algorithm. The optimization process used 200 iterations in each case.

Model	Activation	Number of layers	Neurons per layer	Learning rate	Optimizer
MLP	SeLU	2	34	0.0009746	Adam
LSTM	eLU	2	50	0.0010384	Adam
GRU	eLU	3	56	0.0007395	Adam

Table 3: Hyperparameters values for the models with outliers

Model	MSE	MAE
Benchmark	272.079	8.340
Optimized MLP	236.645	7.961
Optimized LSTM	210.550	7.872
Optimized GRU	178.295	7.574

Table 4: Results of prediction models with outliers

The interesting side of this experiment lays in observing how the models behave when facing an outlier. Under these circumstances some models have learned how to avoid individual outliers. This is the case for the Optimized GRU (hyperparameters shown in Table 3, results shown in Table 4) and it can be observed in Figure 13.

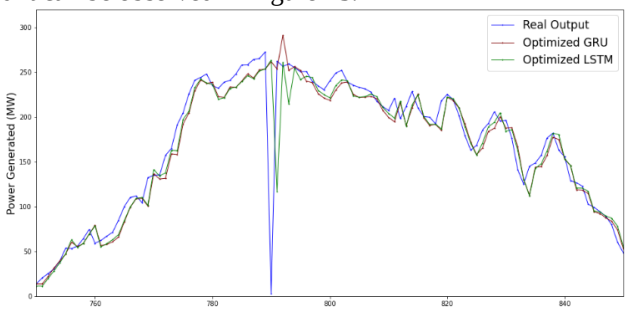


Figure 12: Optimized GRU avoiding an outlier

When analyzing the normalized residuals of the fitted optimized LSTM and GRU, it can be observed how there are multiple outliers not avoided in the validation set.

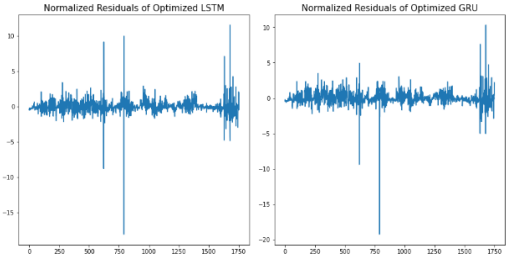


Figure 13: Normalized residuals of opt. LSTM and opt. GRU

4) Optimized algorithms and analysis without outliers

For the second experiment in this series, the power output function was cleaned to make sure no outliers were passed to the algorithms.

The timeseries was corrected both in the training and validation datasets. The number of outliers located in the validation dataset far exceeds the ones found in the training dataset. Despite being just 20% of the data, it contained 13 outliers versus 4 outliers found in the rest of the set.

The Bayesian algorithm was used again for hyperparameter optimization purposes with 200 iterations in each case.

Model	Activation	Number of layers	Neurons per layer	Learning rate	Optimizer
MLP	SeLU	3	50	0.0010145	Adam
LSTM	eLU	2	50	0.0007182	Adam
GRU	eLU	4	62	0.0000965	Adam

Table 5: Hyperparameters values for models without outliers

Model	MSE	MAE
Benchmark	115.815	7.443
Optimized MLP	125.495	7.949
Optimized LSTM	109.571	7.356
Optimized GRU	91.154	6.617

Table 6:Results of prediction models without outliers

Several relevant observations can be made from the tables above. On one hand, none of the optimized models choose the ReLU option. On the other hand, the GRU model is deeper than the rest, using 4 layers. It is also the best performer and it has a particularly small learning rate.

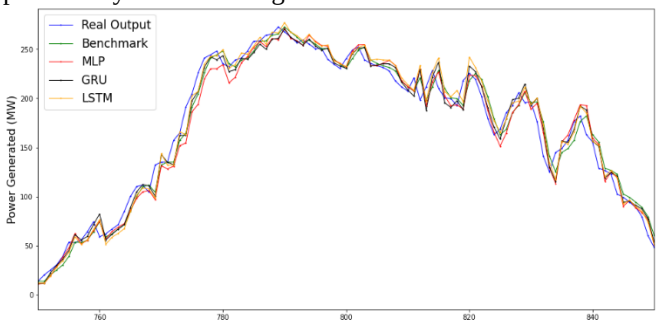


Figure 14: fitted models, output and benchmarks

In Figure 14 we observe a sample of the fitted models in the same region that used to host an outlier (See Figure 12). The behaviour of the models has improved without the outliers, it seems like interpolate the outliers was the right thing to do to improve the predictions.

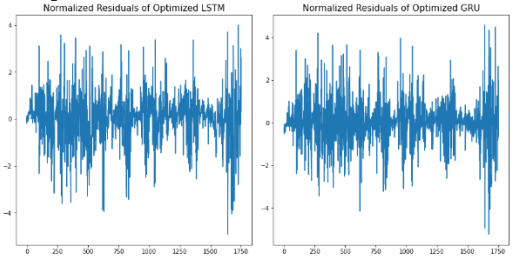


Figure 15:Normalized residuals without outliers

In Figure 15, we can see the contrast when the outliers were removed (See Figure 13). Regarding the residuals, they do not follow a normal distribution (See Figure 16). This implies that the model is not explaining all the error possible, as the result is not a white noise.

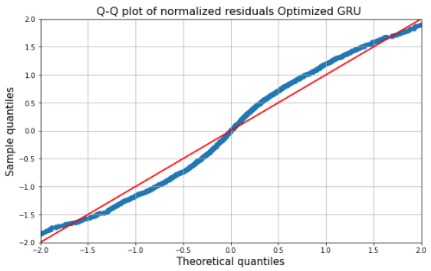


Figure 16:Q-Q plot of the optimized GRU

Despite not following a normal distribution, the residuals of the optimized GRU model are promising since it beats the MSE score of the one lag benchmark by a 22.3% and the MSE score of the MLP by a 27.4%.

Furthermore, the partial and total autocorrelation plots show the optimized GRU can explain significant information from previous lags, especially when compared with the closest algorithm, the LSTM. (Figures 18 and 19).

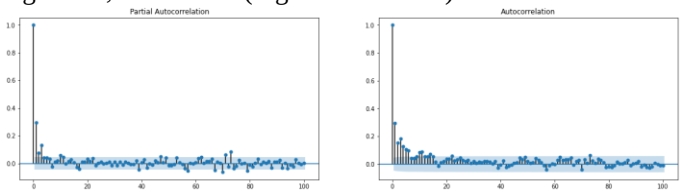


Figure 17: LSTM partial and total autocorrelation plots

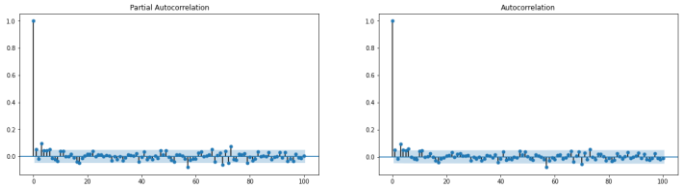


Figure 18: GRU partial and total autocorrelation plots

5) Model dynamic response

After training the models and computing the errors, the next part of the project consisted in observing the time response of the neural network architectures when a step is simulated in the output variable. The neural networks are capable of mimicking dynamics, and the step experiment is a way of showing what dynamics were learned.

As previously mentioned, the models are already trained and the steps do not affect the training data set, but only the validation one. The resulting validation dataset with the two steps implemented is shown in Figure 20.

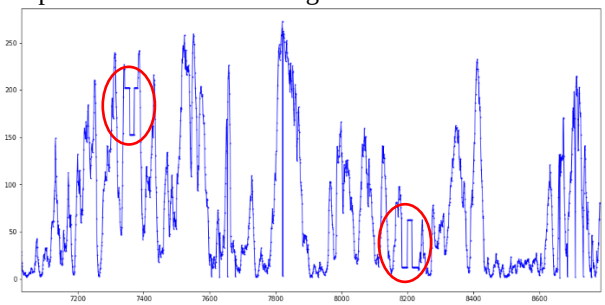


Figure 19:Validation dataset with steps implemented

In the downwards step the three models follow the one timestep lag immediately. In this sense, they all behave the same way. However, the LSTM model seems to have a positive offset. The step values are 205 MW and 155 MW. The GRU model follows the step better, and it reaches the permanent step at the end of the plain. The MLP is not able to come back to the real output level. Both the LSTM and the GRU overshooted when approximating the upwards part of the step.

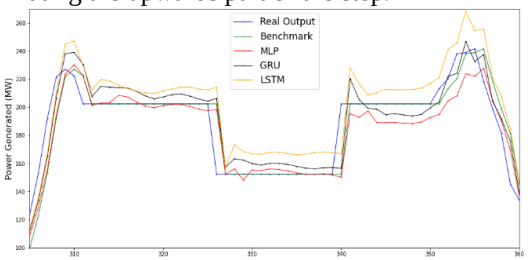


Figure 20:Time response of the downwards step

When analyzing the time response of the upwards step, the starting low value is around 12 MW and then it rises to 62 MW. In this case, the GRU model is the one showing the most overshoot. Again, the LSTM model is the slowest in reacting and it is the one with the highest error in the first timestamps. It is important to notice how none of the predictors stabilize in the high value of the upwards step. The reason behind this seems to be that the inputs are clearly indicating that the output should be significantly smaller. It also shows that the predictors rely on the one lag benchmark for big changes, but this influence wears down as the timesteps pass.

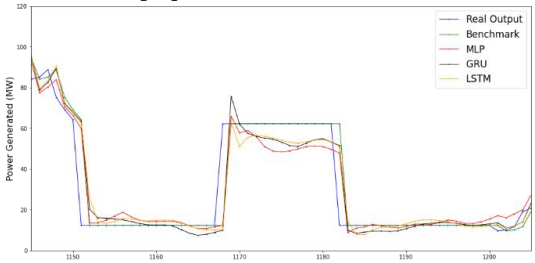


Figure 21:Time response

C. Convolutional neural network predictor

In the final experiment of the project, the one-dimensional convolutional neural network (CNN from now onwards) will be explored. The main objective is to observe differences in the predictions compared with the most successful recurrent neural network. In this model the output is modelled using pattern recognition and extraction, so an improvement over the benchmark is expected beforehand.

The way of approaching the training of the CNN was to run a Bayesian hyperparameter optimizator in two steps. The first one with 200 iterations and a wide hyperparameter space so the algorithm could explore different and diverse solutions.

After obtaining the solution, a second optimization process was launched with 50 iterations and with the space of the hyper-parameters more restricted around the values that rose in the first process. It is very relevant to find the optimized function this way, given that the CNN has a larger number of hyper-parameters, including activation function, kernel size, pool size, dropout, neuron layers, neurons per layer, number of filter layers, filters per layer and others.

Iteration	Activation	Filter layers	Filter per layer	Neuron layers	Neurons per layer
First	eLu	1	28	2	20
Second	ReLu	1	48	1	44

Figure 22: Hyperparameters values for CNNs

Model	MSE	MAE
Benchmark	115.815	7.443
Optimized GRU	91.154	6.617
Optimized CNN	98.204	6.902

Table 7: Results of CNN, benchmark and GRU

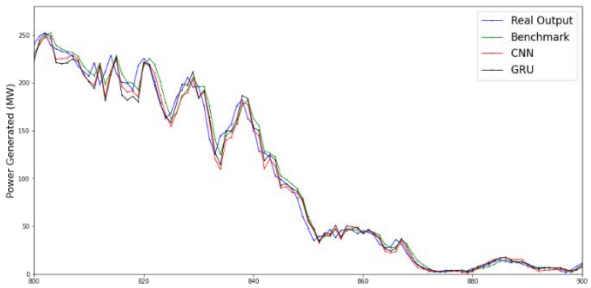


Figure 23: fitted models, output and benchmark

In Figure 24 several observations can be made. The GRU and the CNN model alternate between a close follow up of the output and a follow up of the one timestep lag. It is only when having a look at the results in Table 7 that the best neural network is shown; in this case the GRU holds the first position in both metrics.

For this experiment, the results did not follow a normal distribution and the normalized residuals were very similar to the ones of the GRU in Figure 16, reason why those figures are not attached.

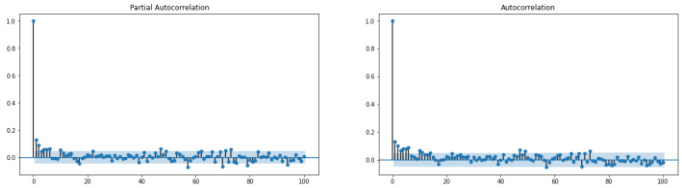


Figure 24: CNN partial and total autocorrelation plots

With respect to the autocorrelation plots, the CNN predictor obtained a good result, eliminating many significant lags. However, the GRU predictor still seems to better capture the influence of lag and inputs to reconstruct the output.

IV. Conclusions

1) Key takeaways

After carrying out an in-depth analysis of the multiple experiments in the project, several conclusions are drawn:

Recurrent neural networks are a powerful tool to address timeseries forecasting problems. Their internal memory captures information from past timesteps and allowed them to beat both benchmarks, the one-day window and the MLP. The ability to better interpret and replicate the dynamics of the output results in better predictions. A prove that the models replicate the dynamic response of the outputs is the fact that they do not follow the permanent value of an arbitrary step in the validation dataset. Better dynamics is not the only quality in favour of RNNs, they can also learn how to avoid outliers; however, it is not reliable as not all the outliers are avoided (some are just not detected). This behaviour was observed the GRU algorithm.

When talking about RNN alternatives, the LSTM is usually the first one mentioned. However, in this project the GRU algorithm outperformed to LSTM algorithm almost at all times.

Alternatively from RNNs, the optimized CNN proved to be a lightweight model that was trained in under 10 seconds each iteration. It is the model with the second most accurate result and with the fastest training time.

On the activation functions side, the ReLU was surpassed by modified versions, such as the ELU activation function. In the first experiment the ReLU activation function was outperformed by SELU. In the second ELU outperformed ReLU in all cases. The ReLU activation function was never the chose none for any Bayesian optimization except for the CNN.

2) Recommendations for forecasting problems

The GRU model should be the first one used for a timeseries forecasting problem given the good results. It provides the most accurate results with smaller datasets and its training is lighter than more complex models, like the LSTM.

Alternatives to ReLU should always be tested. ReLU is only recommended for very complex networks or CNNs.

Try convolutional neural networks when dealing with forecasting problems and very large datasets. They deliver am attractive results vs training time trade-off.

Have a look at partial and total autocorrelation plots when choosing between different neural networks. Apparently similar looking models may have quite different ACF and PACF plots.

References

[1] J. Loenel, Multilayer Perceptron - Jorge Leonel - Medium, 2018. [Online]. Available: https://medium.com/@jorgesleonel/multilayer-perceptron-6c5db6a8dfa3 (visited on 04/05/2020).

[2] M. Stewart, Intermediate Topics in Neural Networks - Towards Data Science, Jun. 2019. [Online]. Available: https://towardsdatascience.com/comprehensive-introduction-to-neural-network-architecture-c08c6d8e5d98 (visited on 04/05/2020).

[3] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling", arXiv:1412.3555 [cs].

[4] B. Raj,11-785 Deep Learning, 2019. [Online]. Available: https://www.cs.cmu.edu/~bhiksha / courses / deeplearning / Spring . 2019 / archive - f19 / www - bak11 - 22 - 2019/ (visited on 04/05/2020).

[5] S. Dennis, The Hopfield Network: Descent on an Energy Surface, 1997.

[6] J. A. Hertz, Introduction To The Theory Of Neural Computation, Mar. 2018.

[7] A. Sayantini, Restricted Boltzmann Machine Tutorial | Deep Learning Concepts, en-US, Library Catalog: www.edureka.co Section: Artificial Intelligence, Nov. 2018. [Online]. Available: https://www.edureka.co/blog/restricted-boltzmann-machinetutorial/ (visited on04/05/2020).

[8] R. Khandelwal, Deep learning — Deep Boltzmann Machine (DBM), en, Library Catalog: medium.com, Dec. 2018. [Online]. Available: del are promising sincehttps://medium.com/datadriveninvestor/deep-learning-deep-boltzmann-machine-dbm-e3253bb95d0f (visited on 04/05/2020).

[9] K. G. Kirby, "A Tutorial on Helmholtz Machines", en, p. 26, 2006.

[10] B. Póczos, Advanced Introduction to Machine Learning, CMU-10715, Sep. 2017. [Online]. Available: https://www.cs.cmu.edu/~epxing/Class/10715/lectures/DeepArchitectures.pdf

[11] Z. Gan, R. Henao, D. Carlson, and L. Carin, "Learning Deep Sigmoid Belief Networks with Data Augmentation", en, p. 9, 2015.

[12] A. Dertat, Applied Deep Learning - Part 3: Autoencoders, en, Library Catalog: towardsdata-science.com, Oct. 2017.

[13] Q. V. Le, "A Tutorial on Deep Learning Part 2: Autoencoders, Convolutional Neural Networks and Recurrent Neural Networks", en, p. 20, 2015.

[14] M. Stewart, Intermediate Topics in Neural Networks - Towards Data Science, Jun. 2019. [Online]. Available: https://towardsdatascience.com/comprehensive-introduction-to-neural-network-architecture-c08c6d8e5d98 (visited on 04/05/2020).

[15] A. Singh Walia, Types of Optimization Algorithms used in Neural Networks and Ways to Optimize Gradient Descent, Jun. 2017.

[16] A. Karpathy, A Peek at Trends in Machine Learning, en, Library Catalog: medium.com, Apr.2017. [Online]. Available: https://medium.com/@karpathy/a- peek- at- trends- in-machine-learning-ab8a1085a106 (visited on 04/05/2020).

[17] Frazier. Peter I, A Tutorial on Bayesian Optimization, July 2017. [Online]. Available: https://arxiv.org/abs/1807.02811#:~:text=Bayesian%20optimization%20is%20an%20approach,stochastic%20noise%20in%20function%20evaluations. (visited on 27/08/2020).