# A Comprehensive Benchmark of Neural Networks for System Identification

Antoine Richard, Antoine Mahé, Cedric Pradalier, Offer Rozenstein, Matthieu Geist

# A Comprehensive Benchmark of Neural Networks for System Identification

Antoine Richard[1], Antoine Mahé[2], Cédric Pradalier[3], Offer Rozenstein[4], Matthieu Geist[5]

***Abstract*— This paper compares a wide variety of neural network architectures applied in the context of black-box modeling for robotics and control. We compare six different architectural concepts and four activation functions, with over three hundred different models. Those models were applied to three robotics datasets to show the differences in performance between the architectures along with their limitations.**

## I. INTRODUCTION

Following the thriving success of machine learning, it is more and more common to use Neural Networks (NNs) for system identification [1], [2], [3]. When using such methods, the first step is to choose an architecture. This choice is critical to the learning and representation capabilities of the NN. It also determines the number of parameters in the network, which is key to estimating the amount of data required for training. There are several considerations to the NNs' design: the number of layers, their size, the activation functions they use, and other regularization techniques. The determination of these parameters relies on either theoretical insight or experimental intuition.

Training data quality poses a challenge during the learning process and several solutions have been suggested [4], [5]. On the other end, the architecture choice is often limited to the input and output layers, which are determined by the system. The black-box nature of those systems requires an in-depth-knowledge and thorough analysis of the different layers' behaviors to tune them for maximum performances, which is known to be time-consuming.

In some fields, there is an understanding that certain NN structures work well for specific applications. The use of convolutional networks for image classification [6] is such an example. However, it is difficult to select the right NN architecture to perform a dynamic regression model for a given system because there are no good tools to rank the performance of the different alternatives. This paper aims to provide such a resource for the most common structures. To this end, we have tested several structures and activation functions on a system identification task over various datasets. Those were obtained by recording the odometry of various autonomous vehicle along with the user command they received. Three different datasets are used, corresponding to the following systems : a simulated drone, a simulated boat, and a real drone in a motion capture setup.

[1]Antoine Richard is with the School of Electrical amd Computer Engineering, Georgia Institute of Technology, North Ave NW, Atlanta, GA 30332, USA. antoine.richard@gatech.edu

[2] Antoine Mahé is with CentraleSupélec, Université de Lorraine, CNRS, LORIA, France antoine-robin.mahe@centralesupelec.fr

[3] Cédric Pradalier is with UMI2958 GT-CNRS, France cedric.pradalier@georgiatech-metz.fr

[4] Offer Rozenstein is with the Institute of Soil, Water and Environmental Sciences, Agricultural Research Organization, Volcani Center, HaMaccabim Road 68, P.O.B 15159, Rishon LeZion 7528809, Israel offerr@volcani.agri.gov.il

[5] Matthieu Geist is with Google Brain, Paris, France mfgeist@google.com

## II. RELATED WORK: ARCHITECTURES & ACTIVATION FUNCTIONS

Neural networks for system identification have been around for the last thirty years [7], [8]. Recent advances in both hardware and architecture brought them back to the front of the scene. However, little work has been carried out to compare the now abundant models available for the identification task. Even though some do [9] they rely on old inadequate datasets such as DaISy [10] which are too small and contain too few samples for complex networks to learn appropriately.

The following section describes different types of NN architectures, and activation functions that are usually applied in NNs used for system identification.

### A. Architectures

System identification and Time-series forecasting are closely related from the data perspective. In both cases, one uses a sequence[1] of input to predict a sequence or a single data point. In model-identification, we often found architectures such as Multi Layer Perceptrons (MLPs), or more recently, Long Short-Term Memorys (LSTMs). However, there are many more approaches for time-series-forecasting: 1D Convolutional Neural Networks (CNNs) [11], Multi-head Convolutional Neural Networks (MH-CNNs), the large family of Recurrent Neural Networks (RNNs) [12] that includes Gated Recurrent Units (GRUs) [13] and LSTMs [14] and their variations. Bellow, we briefly outline those architectures and highlight their pros and cons.

*1) Multi-Layer Perceptron:* MLP is one of the most common NN architectures used for system identification. It relies on stacked dense layers, also called fully-connected layers. Those layers are composed of a matrix multiplication and bias addition. Often, They are completed by an activation function, a normalization function or a dropout function. While the activation function will be studied, the normalization and dropout function effects will not be reviewed. We assume that MLP networks are both shallow and simple

[1]The term sequence implies that the used data are chronologicaly ordered

enough not to require dropout or batch normalization. Hence, the investigation of different regularization techniques falls outside the scope of this paper.

*2) 1D Convolutionnal Neural Network:* 1D CNNs are less famous than their 2D counterparts, but they have a significant advantage over the simpler MLPs. By construction, a 1D CNN processes its input sequence from left to right, which means that even though it was not designed for that purpose, it processes the data in an ordered fashion. This temporal processing is compelling as it behaves a bit like a recurrent neural network but without the complexity of those networks. As will be described later, RNNs training requires careful training and tuning. Their main disadvantages lies in the extra amount of hyper-parameters, longer training time, and slower inference time when compared to the MLPs. Nevertheless, they are simple to train and use.

*3) Multi-head Convolutionnal Neural Network:* MH-CNNs are an extension of the 1D CNNs. Their main advantage is their flexibility: in most cases, those networks have one "head" per input variable. Those heads are composed of specifically tuned 1D CNN for the variable they process. The kernel, the number of channel, and the depth of the head can be adapted for each input individually. The results of the heads are then concatenated together and processed through some fully connected layers. Their main default is the high level of customization required since each head has to be tuned individually. Moreover, those networks are larger then standard CNN adding both training and inference time.

### B. Recurrent-Architectures

In recent years, recurrent architectures have been used to achieve state of the art results in Natural Language Processing. Some of them have also been used to perform time-series-forecasting or gap-filling [15]. Recurrent neural networks are a class of neural networks that explicitly depends on time. RNN uses their internal memory (also referred to as hidden-state) to store their current context. For instance, contextualization allows them to process sentences naturally. Where the next word of the sentence, is read while retaining information about previously read words. This makes them a perfect tool for Natural Language Processing but also time-series-forecasting or model identification. Figure 1[2] shows a "rolled" RNN on the right side and its "unrolled" version on the left side. As can be seen, the same network is used to iterate on the different elements of the sequence while the network retains the hidden-state. The main drawback of the recurrent architectures is their hidden state that needs to be handled expertly to maximize the performances of the networks.

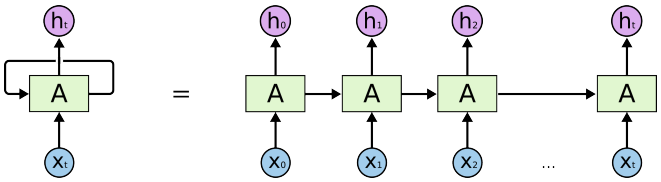


Fig. 1. Structure of a RNN

*1) Recurrent Neural Networks:* In the rest of the paper, the term RNN is used to designate a simple RNN cell that does not contain any memory mechanism but only a `tanh` layer. This simple cell is here as a control to demonstrate whether the memory mechanisms involved in the GRU and LSTM are needed.

*2) Long Short Term Memory:* The LSTMs were invented to learn long-term dependencies. They feature three gates (the input gate, the output gate, and the forget gate) used to predict the output and update their hidden state. Those gates are a form of regulation allowing more or less information to pass through them. They are composed out of a sigmoid layer and a point-wise multiplication operation. Figure 2[3] illustrates a simple LSTM cell.

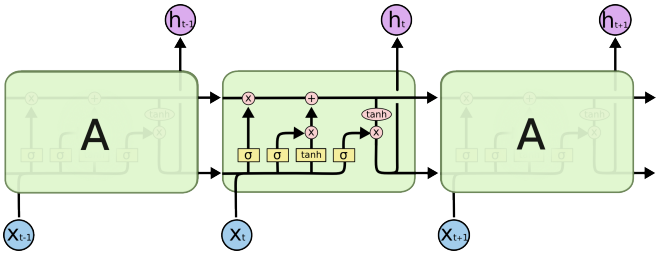


Fig. 2. Structure of a LSTM cell

*3) Gated Recurrent Unit:* GRU, as introduced by [13], is somewhat similar to the LSTM, but it combines the forget and input gates into one gate called update gate.

As we have seen, there are several architectures of networks to choose from, but another important aspect of neural network design is the selection of an activation function for each layer. This fundamental function allows the network to represent nonlinearities and triggers the activation of neurons, making the training process more efficient.

### C. Saturated Activation Functions

Saturated activation functions that were initially used in machine learning, have been slowly replaced by Rectified Linear Unit (ReLU) in most applications. Indeed, the fact that they are saturated can lead to the vanishing gradient problem: preventing the network from learning correctly. Yet, unlike the ReLU, they are strongly non-linear, which may be an appropriate choice for our application domain. Also, it is worth noting that both of the following activation functions are "dense," hence, their gradient is more expensive to compute than the ReLU gradient, which is "sparse."

*1) Sigmoid:* The sigmoid activation function is an activation function that features a smooth non-linearity.

*2) Hyperbolic Tangent:* The hyperbolic tangent, also known as `tanh` used to be a standard activation function. This function is very similar to the sigmoid. The main differences are a negative mapping and a steeper gradient.

[2]Image from `http://colah.github.io/posts/2015-08-Understanding-LSTMs/?source=post_page`

[3]Image from `http://colah.github.io/posts/2015-08-Understanding-LSTMs/?source=post_page`

### D. Unsaturated Activation functions

Unlike the sigmoids, ReLU does not suffer from the vanishing gradient problem. Its unsaturated nature prevents that. The large ReLU family is composed of three primary functions : the standard ReLU, the Leaky Rectified Linear Unit (Leaky-ReLU) and, the Parametric ReLU. We have not studied the parametric ReLU since it adds more training parameters and because it effectively acts like a trainable Leaky-ReLU, hence, if the Leaky-ReLU performs better than the ReLU, then the parametric ReLU will also perform better. In the opposite case, it means that it is probably not worth studying further.

*1) Rectified Linear Unit:* First applied to neural networks in [16] the ReLU is defined as in (1).

$$y_i = \begin{cases} x_i, & \text{if } x_i \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

*2) Leaky Rectifier Linear Unit:* Introduced in [17], the Leaky-ReLU is defined as in 2 where $\alpha$ is generally a substantial value.

$$y_i = \begin{cases} x_i, & \text{if } x_i \geq 0 \\ -\frac{x_i}{\alpha}, & \text{otherwise} \end{cases} \tag{2}$$

The Leaky-ReLU prevents some "locked" neurons from blocking the learning of the whole stack by propagating some of the gradients to the upper layers.

## III. METHOD

This section details how we evaluated the different architectures and activation functions. This resulted in a benchmark of the relative performances of the different types of architectures and evaluates how the parameters of an architecture impact its performances in the context of system identification.

### A. Neural Networks

To evaluate the performance of the different architecture types and activation functions, we designed a large number of models. In total, 332 models have been developed, in order to test different key-components of the architectures. Please note that in the following architectures, the last layer (the one that casts the output of the neural network to the size of the considered state) is not counted in the number of layers. A two-layer MLP would have three dense layers, the last layer being used to cast the correct number of output.

*1) MLP:* For the MLP, we tested the importance of the number of layers, along with the impact of their number of neurons [4]. In total, we derived 43 architectures, ranging from 1 layer to 4 layers with four possible dense layer sizes: 16, 32, 64 and 128. From 1 to 2 layers, all the combinations were covered, but for 3 to 4 layers, only some combinations were used to limit the number of tested models.

*2) Activation Functions:* We also used the MLP as a benchmark for the activation function. Nineteen MLP models were been selected to study the impact of each activation function. Only models with 1 to 3 layers were considered, with dense layer sizes smaller than 128. This left us with 19 models for each activation function. The objective here was to evaluate if some specific activation function yields better results on all the networks or if some activation function gets more of their potential as the number of parameters increased. Finally, we intended to identify which activation function is best suited for these networks.

*3) CNN:* In the case of the CNN, we were interested in three main concepts:

- The impact of the augmentation of the number of channels. The depth of the convolution ranged from one time the number of input to four times the number of input.
- The advantages of having a smaller or larger kernel size. Theoretically, larger kernels are interesting to recognize patterns that are far away. They also allow blending information on a larger scale. The kernel size ranged from 3 to 6.
- Do deeper networks, as seen in computer vision, improve performances in model identification? At most, we stacked 4 convolutions and 2 pooling layers.

To this end, we tested 40 different architectures with an increasing number of channels for a fixed number of layers and kernel size, increasing kernel size for fixed channel and layer numbers, and finally increasing the number of layers for fixed channels numbers and kernesl sizes. We also tested some VGG-like [18] architectures. The dense layers used to cast the encoder[5] outputs into the desired state are three combinations of two layers MLP with respective dense layer size 128-64, 64-64 and 128-32.

*4) MH-CNN:* MH-CNNs are attractive because of their ability to decouple each input since each head processes one input while the kernel size of the convolution (and their depth) can be adjusted to match the "rhythm" of the data. Yet, in our case, this would require tuning each branch individually for each dataset. To avoid this, one can consider two types of Networks. Networks' whose convolutions' kernels and the depth are fixed and similar on all the branches. Alternatively, networks, where each branch corresponds to a different setup in the kernel size and the whole data (as opposed to one variable), is sent through all the branches allowing the network to "choose" what to do with the input and adapt the kernel size automatically. We tested kernel sizes ranging from 3 to 6, such that the network had 4 branches. In total we evaluated 48 variations of MH-CNNs.

*5) RNN, LSTM and GRU:* In the case of the recurrent neural networks, our focus was on the size of the hidden state and the depth of the RNN. The depth of the RNN ranged from 1 hidden cell (RNN, LSTM, or GRU cell) and up to

[4]The term dense layer is similar to fully-connected. It refers to the fundamental building block of the MLP

[5]an encoder in computer vision refers to the convolutional part of the network that "encodes" the input data by bringing it onto a smaller representation space

3 hidden cells. The hidden state had three different values: 16, 32, or 64. Finally, we also evaluated the MLP that casts the RNN encoded output into the correct size output. Three configurations were tested from 0 layers to 1 layer with three possible dense layer sizes: 16, 32 or 64. Overall, this resulted in a total of 48 combinations per recurrent network type or 144 variations of recurrent neural networks.

### B. Networks Training

To train the networks, they were fed sequences of sixteen states and commands. These sequences are used to predict the next state i.e., the state that comes chronologically right after the fed sequence.

All the neural networks were trained using the same optimizer: the ADAM optimizer [19]. This optimizer reduces the need to tune the initial learning rate and its decay over time, allowing us to minimize the learning rate changes between different architectures. Furthermore, the non-recurrent architectures (MLP, CNN, MH-CNN) were trained using a regression loss: the L2 norm and a learning rate of 0.01. While the recurrent architectures (RNN, LSTM, GRU) were trained using the Huber loss and a learning rate of 0.005. The Huber loss has shown slightly better results on the recurrent neural networks over the L2 loss but did not improve the performance on the non-recurrent architectures. Finally, the results presented in the section thereafter are the results averaged over 5 runs for the same architecture on the same dataset.

### C. Evaluation

The evaluation of the models was done using the same validation set for all the models. As they were trained, we evaluated the model performances every 50 steps on two metrics:

- The single-step-accuracy. This metric measures the instantaneous accuracy of the model. It is measured by running the model on the whole validation set. The Mean Squared Error (MSE) is then used to evaluate the performance of the model at a given point during training.
- The multi-step accuracy. This trajectory metric measures the capacity of the model to iterate over its predictions. Fifty trajectories of twenty elements are randomly sampled from the evaluation set. Then the models make predictions for these trajectories of 4 seconds for the simulation datasets and 200ms for the ASCTEC drone. A per-trajectory MSE is then computed and averaged over the fifty trajectories giving us the final metric.

Additionally, we collected the size of the model in both RAM and raw-parameters number, as well as the inference execution time. These extra variables were used to assess performance and computational cost (that relates to energy consumption). All our models ran on an intel i7-4770 CPU. These metrics were received from the The profiling tool embedded in TensorFlow. Please note that we ran these models on a CPU because running them on a GPU degrades the training performances. his was most likely because the data exchange between the CPU and the GPU was too slow. That being said, when inferring large batches ($\approx 1000$ samples or more) it might be beneficial to run this on a GPU.

The reader may observe that the training times were not evaluated; this was due to the relatively short time required to train these networks. It took 2 minutes to train the smallest network and about 10 minutes to train the largest.

## IV. EXPERIMENTS

### A. Datasets

Here we introduce three datasets. They have been collected using a non-linear multiple-input multiple-output (MIMO) robotics systems. Two of them are simulated datasets, whereas the last one is a real-world robotic system.

*1) Drone Simulation: Bebop Dataset:* This dataset simulates the behavior of a Bebop 2 drone by Parrot. The simulation was done in Gazebo [20], using Robotic Operating System (ROS) [21] and a drone emulated by the ROS-package `tum simulator`[6]. This system includes the simulated drone and its low-level controller. The dataset created for this experiment was obtained using a control algorithm that moved the drone such as it uniformly explore the action space without crashing. We recorded nineteen hours of simulation with a sampling rate of 5Hz. We used 3 hours for the test set, 2 hours for the validation set, and, 14 hours for the train set. The state of the drone was composed of the linear velocity of the drone ($v_x$, $v_y$, $v_z$) and its angular velocity around z ($\omega_z$). The command consisted of four variables ($u_1$, $u_2$, $u_3$, $u_4$), the first three are linear speed command along $x$,$y$ and $z$ axis and the fourth one is the angular velocity command around the $z$ axis. Please note that these commands are input to the low-level drone controller: thus, the models based on this dataset included the joint dynamics of the drone and its low-level controller.

*2) Heron Simulation: Heron Dataset:* The Heron simulated dataset was based on a simulation package for the Heron made by Clearpath Robotics. The system was simulated in Gazebo [20], using ROS [21] and the Unmanned Surface Vessel (USV) was emulated using the UUV Simulator [22][7] along with the Heron official repository [8]. Like the drone, the USV was left to explore its action space for twenty hours. We used 3 hours for the test set, 2 hours for the validation set, and, 15 hours for the training set. The state of the Heron is comprised of the linear velocity in x and y ($v_x$, $v_y$), and the angular velocity ($\omega_z$) while the command inputs were the signals sent to the two turbines of the boat ($u_1$, $u_2$). The models based on this dataset should exhibit strong non-linearities since the turbine non-linear efficiency was taken into account by the simulator along with the resistance of the water.

*3) Drone Real-Data: AscTec Dataset:* We derived this last dataset from the EuRoC MAV Dataset [23] and its Vicon Room 2 scenes. Despite being only 2 minutes long,

[6] `http://http://wiki.ros.2org/tum_simulator`
[7] `https://uuvsimulator.github.io/`
[8] `https://github.com/heron`

these scenes feature a 100Hz Vicon, allowing for a precise measurement of the drone position. Thanks to the 100Hz sampling rate, the dataset was composed of 27000 data points. We split the dataset in a 10% test set, 10% validation set, and 80% train set. The drone was an ASCTEC Hexarotor; its state was expressed as the velocity of the drone in cartesian coordinates ($v_x$, $v_y$, $v_z$), and its orientation as a quaternion ($q_x$, $q_y$, $q_z$, $q_w$). Hence, the state was composed of seven variables while the commands were composed of six variables: one per rotor. This makes it the largest MIMO system evaluated. However, because of the very high sampling rate, the predicted system was mostly dominated by the open-loop dynamic of the system; the command sent by the user has little immediate impact on the response of the system. Please note that this dataset was too small to train the RNNs.

### B. Neural Networks

We implemented all the previously mentioned architectures in Tensorflow[24], the non-recurrent architectures were coded with high-level `tf.layers` blocks while the recurrent networks were coded using the `fixed_RNN` library embedded in Tensorflow. However, it had one main drawback: when using this framework, it was not possible to directly access the hidden-state for each element of the processed sequence. This prevented us from performing multi-step-prediction, as the hidden state after the first element of the sequence prediction was required. To get access to this variable, the "rolling" of the LSTM was cut in two parts: the first part where we only loaded the first element of the sequence and the second part where we loaded the remainder. Please note that we also tested the `dynamic_RNN` framework without significant differences in accuracy, training, or inference time.

### C. Networks Training

When training the neural networks, the dataset was shuffled at every epoch except for the recurrent networks where we preserved the continuity of the hidden state by ordering the sequences chronologically and shifting them by one time-increment at the end of every epoch. All the models were run 5 times on each dataset, and their results were averaged for each dataset individually. Due to lack of space, we only show the results of the best performing architectures for each type of network.

## V. RESULTS

This section first discusses the performances between the different architectures. We then discuss the intra-architecture parameters for each architecture. Last, the results of the different activation functions on the MLP models are analyzed.

### A. Architectures

Table I summarizes the results for different architectures. It can be seen that for all datasets, the LSTMs and GRUs performed the best in single-step-accuracy with relative improvement over the best non recurrent network of almost 50%. Still; considering the single-step-accuracy, the simple RNNs, performed better than the MLPs which yield better results than the CNNs. The multi-head version of the convolutional neural networks improved the performance over the standard CNNs, but it remained less accurate than the MLPs. In single-step-accuracy, we thus have the hierarchy shown in (3):

$$\text{LSTM} \approx \text{GRU} > \text{RNN} > \text{MLP} > \text{MHCNN} > \text{CNN} \quad (3)$$

When looking at the multi-step-accuracy, we can see that the hierarchy is not as clear. The first thing we notice is that the recurrent networks had a significantly worse accuracy on the Heron dataset. Additionally, the MLPs, performed as well or better than the CNNs. However, we believe that the implementation of the recurrent neural networks in TensorFlow may be the cause of such degradation in multi-step-MSE. For this reason, we refrain from making any strong conclusion with regards to the RNNs based on these results. Nonetheless, this clearly demonstrates that the MLPs are very easy to train and tune while achieving excellent results, making them an attractive option when the single-step-accuracy of the model is not of paramount importance.

From the computational cost and inference time requirements, it is clear that the multi-head-CNNs are slower than the rest of the other architectures while being much more massive than any of them. As for the MLPs, they are the fastest of the tested networks while achieving correct single-step-performances and reliable multi-step-results.

TABLE I
BEST RESULTS FOR EACH TYPE OF ARCHITECTURE
(LOWER IS BETTER).

| Architecture | | Single-step MSE | Multi-step MSE | Inference Time | Parameters |
|---|---|---|---|---|---|
| Bebop | MLP | 0.079 | 1.09 | 0.64 / 0.62 | 23k / 6k |
| | CNN | 0.085 | 1.07 | 0.78 / 0.79 | 57k / 11k |
| | MH-CNN | 0.072 | 1.08 | 1.57 / 2.6 | 126k / 122k |
| | RNN | 0.058 | 0.43 | 0.71 / 0.71 | 6k / 1k |
| | LSTM | **0.036** | **0.42** | 0.71 / 0.71 | 54k / 2k |
| | GRU | 0.036 | 0.42 | 0.71 / 0.71 | 39k / 15k |
| Heron | MLP | 0.0032 | 0.018 | 0.67 / 0.72 | 5k / 41k |
| | CNN | 0.0047 | **0.016** | 0.87 / 0.79 | 40k / 24k |
| | MH-CNN | 0.0031 | 0.018 | 1.34 / 1.47 | 87k / 43k |
| | RNN | 0.0022 | 0.035 | 1.02 / 1.02 | 1k / 1k |
| | LSTM | **0.0021** | 0.028 | 1.02 / 1.02 | 4k / 2k |
| | GRU | 0.0022 | 0.032 | 1.02 / 1.02 | 2k / 5k |
| AscTec | MLP | **0.010** | **0.015** | 0.40 / 0.44 | 5k / 8k |
| | CNN | 0.034 | 0.021 | 0.64 / 0.58 | 54k / 20k |
| | MH-CNN | 0.021 | **0.015** | 1.71 / 1.87 | 551k / 146k |

Time is expressed as a relative unit. There are two measurements in the parameters and the inference time; the left one is related to the best signle-step model while the right one is related to the best multi-step model.

Among the different MLP models tested, the networks with less than 2 layers performed better than the networks with 3 layers or more in single-step-accuracy, yet, it's the opposite for multi-step-accuracy. This could be due to the relative simplicity of the prediction at one step which means that simpler shallower networks are best suited for this task.

While, in multi-step the task is more complicated: it requires to account for the lag in the command and a resiliency to its own errors. This would explains why deeper and more complex models fair better on this task.

Within the CNNs, the networks with a kernel of size 3 performed better on all the datasets. The VGG like architectures did not demonstrate a particular interest. Also, similarly to the MLPs, smaller networks perform better on single-step-accuracy when larger networks perform better on multi-step-accuracy.

Hence, when it comes to building an architecture, bigger is not always better. It must be tailored to the application needs: when only considering single-step predictions, smaller models will work well enough. However, if multi-step predictions are relevant for the application, larger models will probably yield a higher accuracy.

### *B. Activation functions*

Table II, shows which activation performed best for the different dataset. It can clearly be seen by looking at these tables that the sigmoid and the tanh are outperformed by both the ReLU and Leaky-ReLU on single-step-MSE and multi-step-MSE. Between the different ReLUs, it seems that the Leaky-ReLUs are better than the normal ReLUs. This indicates that using parametric ReLU (P-ReLU) might increase the accuracy of the network.

TABLE II

BEST RESULTS FOR EACH TYPE OF ACTIVATION FUNCTION (LOWER IS BETTER).

| Activation Function | | Single-step MSE | Multi-step MSE | Inference Time | Parameters |
|---|---|---|---|---|---|
| Bebop | tanh | 0.120 | 1.17 | 0.54 / 0.63 | 6k / 3k |
| | sigmoid | 0.087 | **1.07** | 0.94 / 0.55 | 14k / 2k |
| | relu | 0.089 | 1.09 | 0.63 / 0.63 | 14k / 14k |
| | leaky-relu | **0.084** | 1.13 | 0.69 / 0.66 | 9k / 4k |
| Heron | tanh | 0.0063 | 0.028 | 0.61 / 0.86 | 4k / 5k |
| | sigmoid | 0.0048 | 0.024 | 0.71 / 0.67 | 1k / 4k |
| | relu | 0.0032 | 0.021 | 0.67 / 0.67 | 5k / 6k |
| | leaky-relu | **0.0031** | **0.0206** | 0.86 / 0.72 | 2k / 6k |
| AscTec | tanh | 0.059 | 0.060 | 0.65 / 0.65 | 3k / 3k |
| | sigmoid | 0.034 | 0.035 | 0.56 / 0.56 | 3k / 3k |
| | relu | 0.010 | 0.016 | 0.40 / 0.42 | 5k / 12k |
| | leaky-relu | **0.0093** | **0.015** | 0.42 / 0.46 | 3k / 4k |

Time is expressed as a relative unit. There are two measurements in the parameters and the inference time; the left one is related to the best single-step model while the right one is related to the best multi-step model.

## VI. CONCLUSIONS

In this paper, we compared the six main neural network architectures that are used for system identification along with the four most commonly used activation functions in this field. The results show that recurrent architectures outperform all the other architectures in single-step-accuracy, and therefore, they hold interesting potential for system identification applications. However, MLP networks also proved attractive with their simple architectural design and easy implementation. Finally, we showed that leaky ReLUs consistently improved the overall performances of the networks. Hence we recommend using these activation functions. Further work should be focus on newer architectures or more complex activation functions such as the P-ReLU.

## REFERENCES


[1] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou, "Information theoretic mpc for model-based reinforcement learning."

[2] U. Muller, J. Ben, E. Cosatto, B. Flepp, and Y. L. Cun, "Off-road obstacle avoidance through end-to-end learning," in *Advances in neural information processing systems*, 2006, pp. 739–746.

[3] A. Mahé, C. Pradalier, and M. Geist, "Trajectory-control using deep system identication and model predictive control for drone control under uncertain load." in *2018 22nd International Conference on System Theory, Control and Computing (ICSTCC)*, Oct 2018, pp. 753–758.

[4] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[5] A. Katharopoulos and F. Fleuret, "Biased importance sampling for deep neural network training," *CoRR*, vol. abs/1706.00043, 2017. [Online]. Available: http://arxiv.org/abs/1706.00043

[6] W. Rawat and Z. Wang, "Deep convolutional neural networks for image classification: A comprehensive review," *Neural computation*, vol. 29, no. 9, pp. 2352–2449, 2017.

[7] K. S. Narendra and K. Parthasarathy, "Neural networks and dynamical systems," *International Journal of Approximate Reasoning*, vol. 6, no. 2, pp. 109–131, 1992.

[8] K. S. Narendra and S. Mukhopadhyay, "Intelligent control using neural networks," *IEEE Control systems magazine*, vol. 12, no. 2, pp. 11–18, 1992.

[9] O. Ogunmolu, X. Gu, S. Jiang, and N. Gans, "Nonlinear systems identification using deep dynamic neural networks," *arXiv preprint arXiv:1610.01439*, 2016.

[10] B. De Moor, P. De Gersem, B. De Schutter, and W. Favoreel, "Daisy: A database for identification of systems," *JOURNAL A*, vol. 38, pp. 4–5, 1997.

[11] Y. LeCun, Y. Bengio, *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[12] Z. C. Lipton, J. Berkowitz, and C. Elkan, "A critical review of recurrent neural networks for sequence learning," *arXiv preprint arXiv:1506.00019*, 2015.

[13] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[15] Z. Che, S. Purushotham, K. Cho, D. Sontag, and Y. Liu, "Recurrent neural networks for multivariate time series with missing values," *Scientific reports*, vol. 8, no. 1, p. 6085, 2018.

[16] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[17] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30, no. 1, 2013, p. 3.

[18] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[19] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[20] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 3. IEEE, pp. 2149–2154.

[21] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[22] M. M. M. Manhães, S. A. Scherer, M. Voss, L. R. Douat, and T. Rauschenbach, "UUV simulator: A gazebo-based package for underwater intervention and multi-robot simulation," in *OCEANS 2016 MTS/IEEE Monterey*. IEEE, sep 2016. [Online]. Available: https://doi.org/10.1109%2Foceans.2016.7761080

[23] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, "The euroc micro aerial vehicle datasets," *The International Journal of Robotics Research*, 2016. [Online]. Available: http://ijr.sagepub.com/content/early/2016/01/21/0278364915620033.abstract

[24] M. A. et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/