engineering proceedings

Proceeding Paper

Fast Flapping Aerodynamics Prediction Using a Recurrent Neural Network [†]

João A. F. Pereira [1], Emanuel A. R. Camacho [1,2,*], Flávio D. Marques [2] and André R. R. Silva [1]

[1] Department of Aerospace Sciences, University of Beira Interior, 6201-001 Covilhã, Portugal; email1@email.com (J.A.F.P.); email2@email.com (A.R.R.S.)
[2] Department of Mechanical Engineering, Engineering School of São Carlos, University of São Paulo, São Paulo 13566-590, Brazil; email3@email.com
[*] Correspondence: earc.aerog@gmail.com; Tel.: +351-910051301
[†] Presented at the 4th International Electronic Conference on Applied Sciences, 27 October–10 November 2023; Available online: https://asec2023.sciforum.net/.

Abstract: One of the major tasks of aerodynamics is the study of the flow around airfoils. While most conventional methods deal well with steady flows, unsteady airfoils, like the ones on helicopter blades, are subject to such complex dynamic flows that their study can impose substantial difficulties. However, recent applications of machine learning, in the form of neural networks, have shown very promising results when dealing with complex dynamic aerodynamic phenomena. For this reason, this paper proposes the implementation of a recurrent neural network for the time-wise prediction of the lift, momentum, and drag coefficients for an airfoil subject to plunging motion, using the *Re*, *k*, *h*, *kh* and the time-history of the effective angle of attack as inputs. Results from early training already suggest the network's capability to approximate the desired outputs, even if with some limitations. However, the network configuration is flexible enough to be fed with either experimental or numerical data in the future.

Keywords: unsteady aerodynamics; recurrent neural network; potential flow

check for updates

Citation: Pereira, J.A.F.; Camacho, E.A.R.; Marques, F.D.; Silva, A.R.R. Fast Flapping Aerodynamics Prediction Using a Recurrent Neural Network. *Eng. Proc.* **2023**, *52*, 0. https://doi.org/

Academic Editor: Firstname Lastname

Published:

1. Introduction

Being able to study the aerodynamic loads on an airfoil is a major task in the design process of any aero-vehicle [1]. More specifically, there is a wide interest in flapping-wing dynamics. From nature, there is the example of birds, fish, and even cetaceans, which all rely on flapping-wing systems to generate lift and/or thrust [2]. Another example of a flapping wing can be found on a helicopter blade undergoing forward flight [3]. More recently, with a surge in interest in Micro Air Vehicles, the study of flapping wings has gained even greater importance [4].

Given the importance of understanding and predicting the behavior of flapping wings, many methods have been developed over the years. Traditionally, wind tunnel testing is the main way to evaluate the aerodynamic forces on the airfoils. Eventually, Computational Fluid Dynamics (CFD) became one of the most used methods for the study of flows around airfoils. However, these simulations can easily become very computationally intensive. In an attempt to reduce the computational burden of CFD simulations, several Reduced Order Models (ROM) have been developed, but these lack good generalization capabilities and struggle when dealing with dynamic, non-linear, aerodynamic phenomena [5,6].

Thus, it is clear that there is a need for the development of fast, accurate, aerodynamic prediction tools. Recently, there has been an interest in the application of machine-learning-based approaches in the field of aerodynamics, especially Neural Networks (NN).

These are of special interest for their flexibility and generalized model capability. Also, once trained, they are able to produce accurate predictions very quickly [7]. Examples of such applications include the works of Zhang et al. [5], Peng et al. [8], Wu et al. [6], Balla et al. [7] and Moin et al. [1].

In more detail, Zhang et al. [5] implemented a Recurrent Neural Network (RNN) for the prediction of the non-linear, unsteady evolution of the aerodynamic coefficients of an airfoil subjected to an oscillating motion. RNNs are a kind of neural network in which the outputs from one time step are fed as inputs for the following time step, via the so-called feedback nodes. The recurrent structure allows these networks to capture the time-dependent dynamics of a system. Because of this, RNNs have been shown to be able to process complex, non-linear mechanics [9].

For this reason, the present paper proposes the implementation of an RNN for the temporal prediction of the aerodynamic coefficients of an airfoil subject to plunging motions, used as a starting point to explore the designed neural network. The proposed network structure is comprised of a delay layer, followed by a series of fully connected hidden layers, which connect to the output nodes. The delay layer is fed with a sample of both the previous input and outputs, allowing the network to process the time history of the system. Currently, for a proof of concept, the network is being trained with data provided by an implementation of the Hess Smith Panel Method (HSPM), as it allows for a prompt generation of the large data set required for training. Once trained, the network is validated by being presented with a different set of conditions outside of the training data. In the future, the same network is intended to be trained with experimental data for the prediction of unsteady aerodynamic phenomena, such as dynamic stall.

2. Methodology

In this paper, a recurrent neural network is proposed for the time-wise prediction of aerodynamic coefficients of a plunging airfoil (with a null angle of attack). An RNN is a type of neural network that features feedback nodes, which allow it to take the previous output as an input via a delay layer. The proposed network will be used to predict the coefficients of lift, C_l, drag, C_d, and momentum C_m of an airfoil subject to plunging motion. These coefficients form the neural network's output vector $y_{net}(t) = [C_l(t)\ C_d(t)\ C_m(t)]$. The input is given by the Reynolds number, Re, reduced frequency k, nondimensional amplitude h, nondimensional velocity kh, and effective angle of attack α_{eff}.

The time-dependent inputs enter via the delay layer present at the start of the neural network. This layer features four delay nodes, one to store the time-history of α_{eff}, while the others store the history of each one of the previous outputs, which is fed via the feedback node. As one of the objectives of the delay layer is to reduce the space-dimension of the network, this layer will only store the latest q points of α_{eff} and the latest p points of y_{net}, which means the input from the delay layer, at time step t, Z_t^{-1} can be represented as

$$Z_t^{-1} = [\alpha_{\text{eff}}(t), \alpha_{\text{eff}}(t-1), ..., \alpha_{\text{eff}}(t-q), y_{net}(t), y_{net}(t-1), ..., y_{net}(t-p)]. \quad (1)$$

The combination of the external input with the delay forms the complete input of the neural network, u_{net}. Inside the network, a series of fully connected hidden layers will process the input and feed its output as the input for the following layer, until the output layer is reached. This process forms the feed-forward pass of the NN. Each layer is made of several artificial neurons, whose inner workings consist of three basic mathematical operations. Firstly, the weighted sum $w_{i,j}a_j$ of all its inputs is taken, to which the bias b_i is added, resulting in the internal state of the neuron z_i. Lastly, the neuron's internal state is passed through the activation function σ, resulting in the neuron's activation $a_i = \sigma(z_i)$.

It is now possible to describe the mapping between the inputs and the outputs of the neural network, at each time step t, as a global black-box function f

$$y_{net}(t) = f(Re, k, h, kh, \alpha(t), \alpha(t-1), ..., \alpha(t-q), y(t-1), y(t-2), ..., y(t-p), W, b), \quad (2)$$

where W and b represent all the weights and bias of the NN.

From this, it is clear that the goal of training is to find the values of weights, $w_{i,j}a_j$, and biases, b_i, that provide the best mapping between the inputs and the outputs. The training algorithm follows the one presented by Nielsen [10], in his book. For this, the network is subject to a sequential supervised learning task, in which both the inputs and respected target outputs are known for each training example. In order to measure the network's performance at each time step t, the difference between the expected value $d(t)$ and the predicted value $y_{net}(t)$, $e(t) = d(t) - y_{net}(t)$ can be taken.

Then, we can define the total error of the network over a training period $[0, \tau]$ as the sum of the square errors. This training period can be a single sample period (one epoch) or multiple epochs. The total error of the network can be used as the objective function for training

$$J^{\text{total}} = \sum_{t=1}^{\tau} J(t) = \frac{1}{2T} \sum_{t=1}^{\tau} e(t)^2, \tag{3}$$

where J^{total} is the network's objective function. The training process finds the set of values of W and b that minimize this function.

This can achieved by calculating the gradient of the cost function with respect to the parameter space. For this, a backward pass over the network is required, in which the error is obtained for the outputs and feedback across the network. This process is designated back-propagation. Since there are only target values for the outputs of the final layer, the backpropagation algorithm starts by computing the error vector of the final layer

$$\delta^L = \nabla_a J^{\text{total}} \odot \sigma'(z^L), \tag{4}$$

where δ^L is the error vector of the last layer of the network, $\odot$ is the Hadamard product, σ' is the first derivative of the activation function and z^L is the internal state vector of the last layer.

Once this is done, the error vector can be propagated across each layer $l = L-1, L-2, ..., 2$ of the neural network following the equation, in a process called error injection

$$\delta^l = ((W^{l+1})^T \delta^{l+1}) \odot \sigma'(z^L), \tag{5}$$

where δ^l is the error vector of the lth layer, $(W^{l+1})^T$ is the transpose of the weight matrix of the next layer, and δ^{l+1} is the error vector of the next layer. It is important to note that the back-propagation stops at the second layer since the first layer of the network corresponds to the input nodes.

Even though our NN is an RNN, the standard back-propagation algorithm can still be applied by unfolding in time, forming a larger, continuous FNN, to which the back-propagation can be applied. Starting at the final time step $t = \tau$, the back-propagation rewinds through the layers as shown. Then, this process repeats for $t = \tau - 1, t = \tau - 2, ..., 1$, and the parameter change suggestion is accumulated.

Finally, once one or more training batches are completed, the weights and bias can be updated according to the following rule

$$W^l_{\text{new}} = W^l_{\text{previous}} - \frac{\eta}{mN} \sum_x \delta^{x,l} (a^{x,l-1})^T \tag{6}$$

$$b^l_{\text{new}} = b^l_{\text{previous}} - \frac{\eta}{mN} \sum_x \delta^{x,l}, \tag{7}$$

where m is the batch size, such that the training period $\tau = mN$.

The batches are randomly chosen subsets of the training set, and this method helps to improve the learning process and to delay over-fitting.

Even still, the network training requires a large set of labeled training data to be generated. For this reason, an implementation of the Hess-Smith Panel Method (HSPM) used by Teng [11] is being used to produce the training data. This code follows the same assumptions of Basu and Hancock [12]. The labeled data is comprised of an oscillating period of 101 points, with its respective effective angle of attack and aerodynamic coefficients. The airfoil chosen for the data generation was the NACA0012.

In order to validate the results of the neural network, the final data set will be divided into two subsets, one for the actual training, and one for validation. This validation set is to hold distinct cases from those the network will be trained on. This is important as it allows us to check that the network has learned the behavior of the system and not just how to guess the training examples. The cost function from this validation stage is also used to assess the progress and convergence of the learning process.

The current implementation of the RNN includes a total of 3 hidden layers, with 15, 10, and 5 neurons, respectively. The input layer has 20 input fields and the output layer possesses 3 neurons, one for each of the outputs. The delay layer stores 4 points for each of its inputs: α_{eff}, C_l, C_d, and C_m for a total of 16 delay nodes.

The chosen activation function was the hyperbolic tangent since it is recommended to be used with recurrent neural networks [10]. It is also relatively fast to compute, avoids the neuron deactivation problem and its derivative is defined in the domain.

As for a proof of concept, the implemented network was already trained using an initial example training set. This is a small set of 30 training examples created using the HSPM for the conditions shown in Figure 1. This condition was set for a Reynolds number of 1×10^6. HSP was also used to generate a validation set of 5 examples, used to assess the network's performance. This validation set is for a h of 0.6 and k in the range $[0.01, 0.05]$.

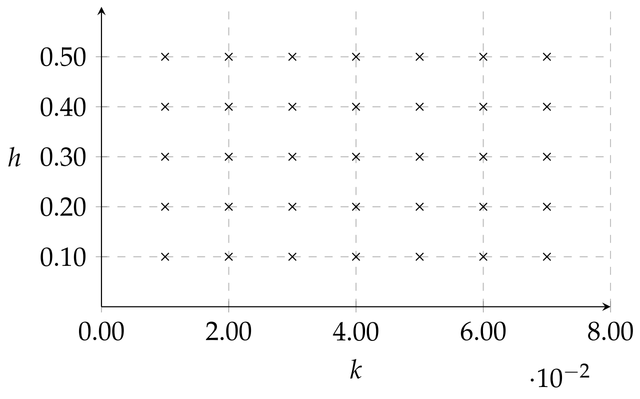

Figure 1. Conditions used for training.

3. Results and Discussion

As mentioned previously, to assess the performance of the network, a small initial training set of 30 plunging training points was built. Together with an extra set of 5 cases for validation. Some of the obtained results will be shown in this section. The network was let to run for a total of 1000 epochs with a learning rate of 1.0. The evolution of the objective function over the epochs can be seen in Figure 2.

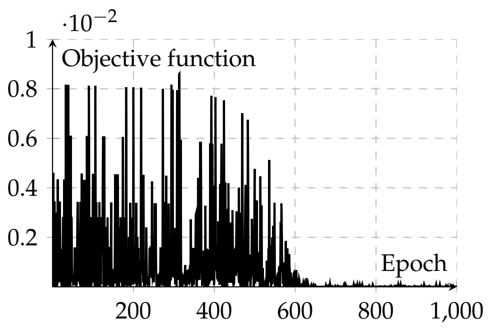

Figure 2. Evolution of the objective function with the training epochs considering a learning rate of 1.0.

From Figure 2 it is possible to see that the training started to converge after 300 epochs, and the objective function reached a minimum after just 600 training epochs. After that the convergence rate slowed down, suggesting the learning process entered a trained state, with little to no further improvement in prediction accuracy. It is important to remark that no study has been conducted yet for the optimization of the learning rate, as the current focus is to evaluate the network's functionality. It can also be noted that, at the start, the objective function exhibits a great amount of fluctuation. This can be due to the randomness of the batch selection and the fact the objective function's value is recorded for each individual case.

Let us now plot some training results from two distinct moments. The first, in Figure 3, shows an example result from the 500th epoch of training, just halfway through training. The second result, which is seen in Figure 4, corresponds to the same example regime, but it is taken from the last epoch.

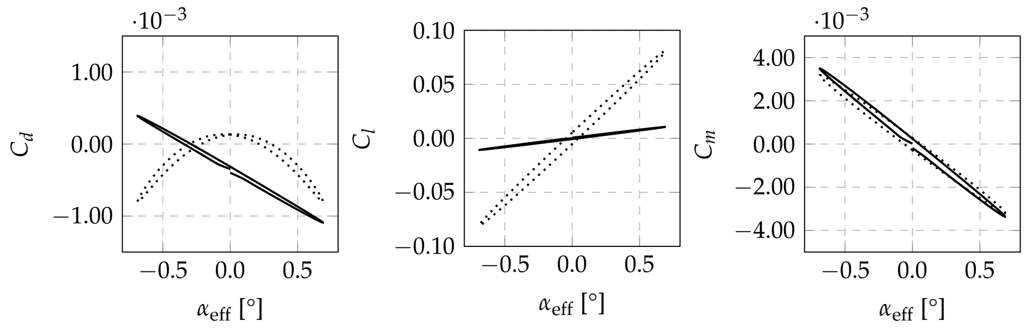

Figure 3. Comparison between the predicted outputs (full) and the target values (dotted) at epoch 500 with $h = 0.4$ and $k = 0.03$.

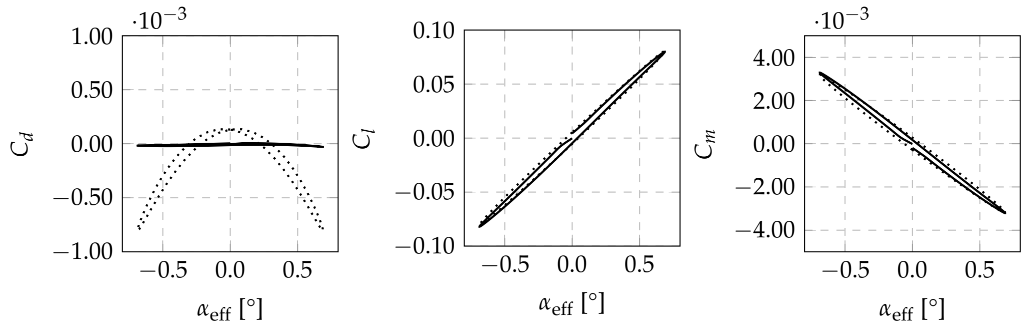

Figure 4. Comparison between the predicted outputs (full) and the target values (dotted) at epoch 1000 with $h = 0.4$ and $k = 0.03$.

Comparing both Figures 3 and 4, it is clear how the network prediction capability improved. The momentum coefficient was the first to converge, being already close to the target output at epoch 500. This was followed by the convergence of the lift coefficient. However, despite the apparent minimization of the objective function, the drag coefficient was not able to fully converge. From the training results, it has been observed that the network struggles to predict the drag coefficient.

One possible explanation is that, due to the difference in magnitude between the coefficients, the gradient of the function tends to favor the ones with greater difference and, consequently, these will be the ones with larger influence. Because of this, the network could be trapped in a local minimum where the error associated C_l and the C_m is minimized. Any further attempts from the network to optimize the C_d would produce an increased error of the remaining coefficients, causing the gradient descent to become trapped.

Looking closer at the lift and pitching momentum coefficients, it is also observed that the estimates start with a greater deviation from the target value, which steadily diminishes as the prediction approaches the end of the period. This deviation is maximum at the first 3 points of the network's estimate, which suggests that it must be a consequence of the delay initialization problem.

Finally, the trained network was run with the validation cases, outside of the training conditions. This was done in order to evaluate if the neural network had properly learned how to predict the system behavior, and not just to fit the training examples. The following plots in Figure 5 show the difference between the predicted values and the target outputs for one of the validation cases.

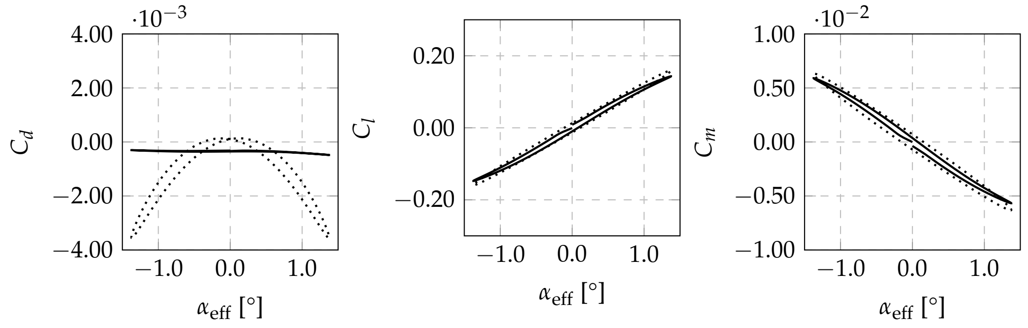

Figure 5. Comparison between the predicted outputs (full) and the target values (dotted) for a validation case with $h = 0.6$ and $k = 0.04$.

When observing the results from validation in Figure 5, the network exhibits, once again, a difficulty when predicting the drag coefficient, which is expected since it did not manage to learn it during training. However, even with some errors, it can be seen that the network was able to estimate the behavior of the lift and drag coefficients, despite the fact that it had never seen this condition during training.

It is also clear that the network struggles with the first 3 points of the period since it does not have enough information about the previous outputs. From analyzing the lift coefficient, it can be seen that the estimate does indeed improve after the first quarter of the sampling period, with one exception. When closely inspecting the plots for both the C_l and the C_m, one can notice the behavior at the ends of the ellipses, where the prediction slightly deviates from the target value.

4. Conclusions

From the results, it is possible to conclude that the network is trying to learn how to correctly predict the aerodynamic coefficients. However, this is only really the case for the C_l and C_m. It could be possible that, since these are the coefficients of the largest magnitude, their high error makes the network fall into a local minimum of the cost function. One possible solution to mitigate this problem would be to train a separate network for each of the coefficients or normalize all outputs. This would still allow for the use of the learned weight and bias matrices to quickly model the outputs.

There is still a lot of room for improvements in the network itself. One aspect to tackle is the initialization of the delays. Regarding the precision of the predictions, the current algorithm can be enhanced with methods such as momentum and weight decay. These results are also limited by the very reduced size of the current training sample. It is hoped that the results will be improved once the larger data set is used. The learning rate and mini-batch size will also have to be studied properly in order to optimize the learning process.

Author Contributions: Conceptualization, J.A.F.P. and E.A.R.C.; methodology, J.A.F.P. and E.A.R.C.; validation, J.A.F.P.; formal analysis, J.A.F.P. and E.A.R.C.; investigation, J.A.F.P.; data curation, J.A.F.P.; writing—original draft preparation, J.A.F.P. and E.A.R.C.; writing—review and editing, A.R.R.S. and F.D.M.; supervision, A.R.R.S. and F.D.M.; project administration, A.R.R.S. and F.D.M.; funding acquisition, A.R.R.S. and F.D.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Portuguese's Fundação para a Ciência e Tecnologia (FCT) under grant numbers UIDB/50022/2020, UIDP/50022/2020, LA/P/0079/2020, 2020.04648.BD, and by Brazil's National Council for Scientific and Technological Development (CNPq grant #306824/2019-1).

Institutional Review Board Statement:

Informed Consent Statement:

Data Availability Statement: The data presented in this study are available on request from the corresponding author. The data are not publicly available due to privacy.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Moin, H.; Khan, H.Z.I.; Mobeen, S.; Riaz, J. Airfoil's Aerodynamic Coefficients Prediction using Artificial Neural Network. In Proceedings of the 2022 19th International Bhurban Conference on Applied Sciences and Technology (IBCAST), Islamabad, Pakistan, 16–20 August 2022; pp. 175–182.
2. Platzer, M.F.; Jones, K.D.; Young, J.; Lai, J.C.S. Flapping Wing Aerodynamics: Progress and Challenges. *AIAA J.* **2008**, *46*, 2136–2149. https://doi.org/10.2514/1.29263.
3. Perdomo, O.; Wei, F.S. On the flapping motion of a helicopter blade. *Appl. Math. Model.* **2017**, *46*, 299–311. https://doi.org/10.1016/j.apm.2017.01.055.
4. Platzer, M.F.; Jones, K.D.; Young, J.; Lai, J.C. Flapping wing aerodynamics: progress and challenges. *AIAA J.* **2008**, *46*, 2136–2149.
5. Zhang, B.; Han, J.; Zhang, T.; Ma, R. Unsteady aerodynamic identification based on recurrent neural networks. *J. Vibroeng.* **2021**, *23*, 449–458.
6. Wu, M.Y.; Wu, Y.; Chen, Z.H.; Wu, W.T.; Aubry, N. Fast Prediction of Flow Field around Airfoils Based on Deep Convolutional Neural Network. *Appl. Sci.* **2022**, *12*, 12075. https://doi.org/10.3390/app122312075.
7. Balla, K.; Sevilla, R.; Hassan, O.; Morgan, K. An application of neural networks to the prediction of aerodynamic coefficients of aerofoils and wings. *Appl. Math. Model.* **2021**, *96*, 456–479. https://doi.org/10.1016/j.apm.2021.03.019.
8. Peng, W.; Zhang, Y.; Laurendeau, E.; Desmarais, M.C. Learning aerodynamics with neural network. *Sci. Rep.* **2022**, *12*, 6779.
9. Williams, R.J.; Zipser, D. Gradient-Based Learning Algorithms for Recurrent Networks and Their Computational Complexity. *Back-Propag. Theory Archit. Appl.* **1995**, 1–14.
10. Nielsen, M. *Neural Networks and Deep Learning*; Determination Press, 2015. Available online: http://neuralnetworksanddeeplearning.com/ (accessed on).
11. Teng, N.H. *The Development of a Computer Code (U2DIIF) for the Numerical Solution of Unsteady, Inviscid and Incompressible Flow over an Airfoil*; Technical report; Naval Postgraduate School: Monterey, CA, USA, 1987.
12. Basu, B.C.; Hancock, G.J. The unsteady motion of a two-dimensional aerofoil in incompressible inviscid flow. *J. Fluid Mech.* **1978**, *87*, 159–178. https://doi.org/10.1017/S0022112078002980.