Criteria to switch from tabulation to neural networks in computational combustion

Z. Nikolaou, L. Vervisch, Pascale Domingo

Criteria to switch from tabulation to neural networks in computational combustion

Z. Nikolaou[a], L. Vervisch[a], P. Domingo[a]

[a]*CORIA-CNRS, Normandie Université, INSA de Rouen Normandie, France.*

Abstract

Motivated by the need to reduce computational costs, look-up tables are widely used in numerical simulations of laminar and turbulent flames, for the thermodynamics of the mixture, for detailed chemistry, and for turbulent combustion closures. At the same time, there have been many studies where artificial neural networks have been trained to replace the classic tabulation approach, and their performance against tabulation typically evaluated a posteriori. In the majority of applications the focus is on accuracy, and the objective is to obtain the best network structure which minimises the inference error during training. Computational efficiency however is also important and criteria are needed to decide whether or not it is worthwhile in the first place to employ neural networks at all, and if so what the potential bounds on the compute-time and memory gains (if any) over tabulation are. This is examined analytically in this work by developing models for the computational cost of tabulation and neural networks. A framework for effective decision-making between adopting lookup tables or machine learning then emerges.

Email address: zacharias.nikolaou@insa-rouen.fr (Z. Nikolaou)

Nomenclature

t_+, t_x, t_o	Time spent for a single operation
t_a, t_l	Time spent for an activation function evaluation, a neural network layer evaluation
T_{tab}, T_{ann}	Time spent for tabulation inference, neural network inference
M_{tab}, M_{ann}	Memory used for tabulation, neural network
N_p	Number of look-up table input parameters
N_v	Number of look-up table output variables (targets)
N_i	Number of inputs to the network
N_o	Number of outputs of the network
N	Number of nodes in the hidden layers
L	Total number of layers
l_h	Number of hidden layers
S	umber of weights of the network
h	table resolution spacing
N_T^e	Number of nodes below which any network structure with N_p inputs and N_v outputs is less expensive than tabulation in terms of computational time
N_T^s	Number of nodes above which any network structure with N_p inputs and N_v outputs is more expensive than tabulation in terms of computational time
N_{MT}	Number of nodes below which any network structure with N_p inputs and N_v outputs will have the memory gain over tabulation outweigh the computational time loss
N_p^*	Number of input parameters for a given number of target variables N_v and nodes N, above which tabulation is always slower than any N-node fully-connected neural network structure with N_p^* inputs and N_v outputs.

1. Introduction

Tabulation is widely used in turbulent reacting flow simulations. In flamelet methods for instance with a presumed-pdf, to avoid solving the large number of transport equations for each individual species in the chemical mechanism, a reduced number of suitably defined variables such as the progress variable $\tilde{c}$, mixture fraction $\tilde{Z}$, and their variances are solved for instead [1]. The species mass fractions and any other unclosed terms appearing in the transport equations are then obtained from pre-built look-up tables using as inputs the reduced set of variables/parameters solved (or modelled for) during the actual Reynolds averaged Navier-Stokes (RANS) or space filtered Large Eddy Simulation (LES). In another context, pre-computational approaches have also been used to replace the expensive integration of the chemical species source terms [2, 3, 4, 5, 6].

Irrespective of the context, the pre-built tables are typically constructed using canonical 0D/1D problems (steady or unsteady). The computational cost of tabulation is composed of two main actions: (a) table look-up, and (b) interpolation. The resolution spacing of the tables for each tabulation parameter may be constant, in which case we will refer to the tables as structured, or variable, in which case we will refer to the tables as unstructured. If the table is structured, the table look-up process is very fast but more points have to be included which increases the memory requirements. If it is unstructured, bisection or some other table look-up algorithm is typically used, which increases the computational time but reduces the memory requirements. In terms of interpolation, multi-linear interpolation is almost always used which is the fastest possible interpolation scheme. Overall, tabulation methods using this classic approach have been very effective and have been applied to several LES studies for different combustion modes and flow configurations

[7, 8, 9, 10, 11, 12, 13].

Nevertheless, the memory requirements for storing even medium resolution tables can be significant. For instance, using four parameters to tabulate a single species mass fraction with 100 points in each parameter space would require 0.8Gb of double precision memory per variable. Extending this to all 55 species for example of the GRI3 mechanism [14] would require 44Gb of memory per physical core, which is well above current capabilities of most High Performance Computing (HPC) facilities (max around 5Gb/physical core). Therefore the usual remedy is to reduce as much as possible the resolution and/or the number of tabulation parameters even when using unstructured tables. Neural networks on the other hand have shown great potential for solving a wide range of regression problems both in reacting and non-reacting flows [15, 16, 17, 18, 19, 20, 20, 21] and tabulation is no exception. For instance in [22] two-layer networks were trained and applied to LES of a non-premixed piloted methane-air flame. The inputs to the network consisted of the the mixture fraction, its variance, and the scalar dissipation rate while the outputs consisted of the species mass fractions, density, viscosity and temperature (a single network was trained for each output variable). The largest network consisted of 34 neurons while the tables were constructed based on three parameters with $200\times$ 100 $\times$ 16 samples in each parameter space. As a result, the overall ANN memory requirements were about 1300 smaller in comparison to tabulation. In terms of inference time, despite the small number of neurons the authors found the networks to be about 1.3 times slower. In a later study [23] an optimisation procedure was proposed for obtaining the best network structure which minimises the training error. The same method was later applied to develop suitable networks to replace tabulation in LES of a bluff-body swirl-stabilised flame [24]. Despite similar (large) memory savings being reported, in comparison to the coarser-resolution tables the ANNs were six times

slower than tabulation, and two times slower for the finest-resolution tables [23]. In [25] a clustering pre-training processing step was proposed in an effort to reduce the overall computational cost of the ANN approach. In order to further improve the inference time of the networks, the evaluation of the exponential activation function used in the networks was approximated by a truncated series expansion. The trained networks were applied in LES of spray flames and in RANS of engine combustion with overall good results. In contrast with the previous studies, the authors reported an overall compute time speedup when using the networks: 8% for the LES case and 37% for the RANS case. In [26] a different clustering approach namely Self-Organising Maps (SOM) was used. In an effort to optimise the networks with regards to accuracy, the network structure was fixed and the ratio of neurons in each layer relative to the first layer was imposed. Then, through an iterative process the number of neurons in the first layer was increased (as those in the next layers based on the imposed rations) until sufficient accuracy was achieved. The trained network groups were then implemented in LES and RANS and compared to classic tabulation. In terms of total simulation computational time, the ANN-based approach was reported to be (slightly) slower while the authors reported a significant gain in memory savings. In [27] an alternative approach based on deep neural networks with skip connections was employed. The authors reported a memory gain of about 55 when compared to classic tabulation which is much lower than the memory gains reported in the previous studies-perhaps due the increased complexity of the network. Overall, the consensus between all studies in the literature is that the use of neural networks reduces the memory requirements and can ease the solving of problems featuring numerous degrees of freedom, for instance soot particle size distribution [28]. In terms of inference time, only the work in [25] reports a reduction in comparison to classic tabulation, as well as the work in [27], however it is not clear whether

the comparison in [27] was made on the same basis as with tabulation (the authors reported that the networks were ran on a Graphics Processing Units (GPU) but it is not clear whether the classic tabulation approach was also ran on a GPU).

The training of neural networks can be an expensive procedure as it first includes the development of a database (which typically implies running a large number of canonical simulations), data pre-processing and so on, and finally the training process itself, which for large amounts of data requires the use of high-memory GPUs. The training process may take hours to days depending on the size of the database and the complexity of the network. Therefore it is important to be able to investigate a priori whether training a neural network to replace tabulation would be beneficial or not. More importantly, it is useful to quantify a priori the maximum possible gains (if any) and under which conditions these can be achieved.

In order to do so and to effectively compare between the two approaches, an analytical model is required and to the best of the authors knowledge, to date there is no formal model-based comparison for the computational cost between the neural network approach and the classic tabulation approach. For instance it is unclear if a limit exists on the maximum achievable memory savings and/or the maximum achievable inference time savings (if any) and how these are affected by (i) the number of table parameters (look-up table dimension), (ii) the number of targets (table outputs) and (iii) the network structure. As a result, there is no computational-efficiency-based training strategy when developing networks to replace tabulation. In this work we begin in section 2 by developing such models: (a) for fully-connected neural networks, and (b) for classic structured tabulation, which is the simplest tabulation case to analyse but also the fastest. We next proceed in section 3 to investigate the effect of network structure on the computational cost. In section 4, we derive bounds for three distinct cases when training neural networks.

2. Computational cost models

2.1. Tabulation

In the case of tabulation, let N_p denote the number of tabulation parameters (i.e. the table dimensionality) and N_v the number of target variables for which a single table of dimension N_p is stored. In addition, we make the following assumptions:

(a) The tables are constructed with a fixed resolution spacing $h_i = h$ for $i \in [1, N_p]$.

(b) Multi-linear interpolation is used to infer values during run-time.

(c) The computational cost of floating point number addition and multiplication is the same i.e. $t_+ = t_x = t_o$.

Then, for a single linear interpolation of the form,

$$y_i = y_l + \frac{x_i - x_l}{h}\left(y_h - y_l\right) \ , \tag{1}$$

where (x_l, y_l), (x_h, y_h) are the low and high values within which the variable falls, in total five operations are required to interpolate at x_i and retrieve y_i. In the multi-dimensional case, it is straightforward to show that the total number of linear interpolations required is $\sum_{r=0}^{N_p-1} 2^r = (2^{N_p} - 1)$. In order to obtain the upper and lower indices for each table parameter (table look-up process) some additional operations are required. The most straightforward way of doing so for structured tables is to use $i_l = floor(x/h) = [x/h]$ and $i_h = i_l + 1$, i.e. in total three operations are required per table parameter per target variable to obtain the upper and lower bounds within which the value lies. The total compute time for all target variables for all table parameters is given by,

$$T_{tab} = \left(3N_pN_v + 5N_v\left(2^{N_p} - 1\right)\right)t_o \tag{2}$$

With regards to memory requirements this depends on the table resolution and can be calculated using,

$$M_{tab} = bN_v \prod_{r=1}^{N_p} n_r \tag{3}$$

where n_r is the number of sample points in the tabulation of parameter r and b is the memory requirement-typically this is 8 bytes (double precision).

2.2. Neural networks

Consider a fully-connected neural network. The output y_{lk} from node k in layer l is given by,

$$y_{lk} = f_{lk}\left(\sum_{j=1}^{N_{l-1}} w_{ljk}y_{l-1j} + b_{lk}\right) ,$$

where N_l is the number of nodes in layer l, and where in the most general case the activation function is node and layer specific. We assume that the activation function in the last layer is linear and that there is no bias in the last layer. Let t_a denote the time required to perform an activation function evaluation. Then, the compute time for a single layer t_l is given by,

$$t_l = \underbrace{N_l(N_{l-1} - 1)t_+}_{\text{matrix-vector additions}} + \underbrace{N_lN_{l-1}t_x}_{\text{matrix-vector multiplications}} + \underbrace{N_lt_a}_{\text{function evaluations}} + \underbrace{N_lt_+}_{\text{bias additions}}$$

and for all the layers is,

$$T_{ann} = \sum_{l=2}^{L-1} t_l + N_L(N_{L-1} - 1)t_+ + N_L N_{L-1} t_x \tag{4}$$

In order to proceed we make the following assumptions:

(a) The matrix-vector multiplication is performed with the crudest of algorithms (directly).

(b) $t_a = t_+ = t_x = t_o$.

Note that the compute time cost of the activation function depends on its formulation. The fastest and perhaps most popular activation function is the Rectified Linear Unit (RELU) for which the compute time cost is approximately equal to the cost of addition/multiplication as it involves essentially only a $max(0.0, x)$ operation where x is the input. In any case, assumption (b) ensures that the comparison with tabulation is being made for the fastest activation function. Equation 4 then reduces to,

$$T_{ann}/t_o = (N - N_o) + 2\sum_{l=2}^{L} N_l N_{l-1} = (N - N_o) + 2S \tag{5}$$

where $N = \sum_{l=2}^{L-1} N_l$ is the total number of nodes in the hidden layers, N_o the number of outputs, and $S = \sum_{l=2}^{L} N_l N_{l-1}$ is essentially the total number of trainable weights of the network, which directly relates to the network structure. In terms of memory requirements, the ANN requires storage of all the weights and biases in each layer. Therefore the memory requirements are given by,

$$M_{ann} = \sum_{l=2}^{L-1} \left(\underbrace{bN_l N_{l-1}}_{\text{weights}} + \underbrace{bN_l}_{\text{biases}} \right) + \underbrace{bN_L N_{L-1}}_{\text{weights for last layer}} = b\,(N + S) \tag{6}$$

where b is the storage size required for a single variable.

3. Effect of network structure

From Eqs. 5 and 6 we observe that for a given number of inputs N_i, number of outputs N_o, and total number of nodes in the hidden layers N, the time and memory costs depend only on S, i.e. on the network structure. Of course there are many possible network structures for N nodes: for at least p nodes to exist in each of the l_h hidden layers, the number of possible structures Q is $Q(N, l_h) = (N - pl_h + l_h - 1)!/((l_h - 1)!(N - pl_h)!)$. For example for $l_h = 2$ layers, $N = 100$ and $p = 1$, there are 99 different structures. For $l_h = 3$ and $p = 1$, there are 4851 structures and so on. In any case, the "boundary" networks consist of the "column" network i.e. having a single layer and N nodes, [N_i,N,N_o], and the "serial" network, having N layers with a single node in each one [N_i,1,1,1,1, ..., N_o]. The natural question is then which network structure is the most efficient, and which is the least efficient? The answer to this question will help to obtain bounds on the possible compute time and memory savings over tabulation. In order to gain insight into this we examine in the next section the case where the number of nodes N is a power of 2.

3.1. Illustration: the case $N = 2^p$

Consider a network with $N = \sum_{l=2}^{L-1} = 2^p$ nodes in the hidden layers, $p \in \mathbb{N}$, and all possible network structures having 2^k layers and 2^{p-k} nodes/layer where $k \in [0, k]$. For example this leads to networks with the following structures: 1 layer x 2^p nodes/layer namely the column network, 2 layers x 2^{p-1} nodes/layer and so on down to the serial network with 2^p layers and 1 node/layer. Then S for this network structure can be shown to be,

$$S = \frac{(N_i + N_o)2^p}{2^k} + 2^{2p}\left(\frac{1}{2^k} - \frac{1}{4^k}\right) = \frac{S_c}{2^k} + \frac{{S_c}^2}{(N_i + N_o)^2}\left(\frac{1}{2^k} - \frac{1}{4^k}\right) ,$$

where $S_c = N(N_i + N_o) = 2^p(N_i + N_o)$ is the cost for the column network. The above equation can be recast as,

$$\frac{S}{S_c} = -\frac{2^p}{a}\left(\frac{1}{2^k}\right)^2 + \left(1 + \frac{2^p}{a}\right)\frac{1}{2^k}\,,$$

where $a = (N_i + N_o)$ which is a second-order equation in x=$1/2^k$. Then it can be shown that $S/S_c \geq 1$ if,

$$0 \leq k \leq int\left(p - \frac{ln(a)}{ln(2)}\right) \tag{7}$$

with $S/S_c = 1$ at the two ends and S/S_c maximised in the middle of the region. The above result implies that the next most efficient network structure occurs for the next integer larger than the upper bound in the equation above. For example let $N = 32 = 2^5$ and $a = 4$. Then we have the following structures,

$$k = 0|\ 32$$
$$k = 1|\ 16{,}16$$
$$k = 2|\ 8{,}8{,}8{,}8$$
$$k = 3|\ 4{,}4{,}4{,}4{,}4{,}4{,}4{,}4$$
$$k = 4|\ 2{,}2{,}2{,}2{,}2{,}2{,}2{,}2{,}2{,}2{,}2{,}2{,}2{,}2{,}2{,}2$$
$$k = 5|\ 1{,}1{,}1{,}1{,}1{,}1{,}1{,}1{,}1{,}1{,}1{,}1{,}1{,}1{,}1{,}1{,}1{,}1{,}1{,}1{,}1{,}1{,}1{,}1{,}1{,}1{,}1{,}1{,}1{,}1{,}1{,}1{,}1$$

for k =0,1,2,3,4,5 respectively with $k = 5$ corresponding to the serial network. Let us denote ΔS the variation of the number of trainable weights when modifying the network structure. Then all networks of the above form for $0 \leq k \leq 3$ will have $\Delta S \geq 0$ with respect to $k = 0$ with $\Delta S = 0$ for $k = 3$. Only the networks corresponding to $k \geq 4$ will be computationally more efficient than the column network. This example shows that the column network is not necessarily the most

expensive network structure. The least expensive network structure in this case is the serial network, and is also the least expensive structure in general, as we show in the sections which follow. In order to examine the effect of any general network structure on S we consider the effects of the following fundamental actions one may perform on the network: (a) layer creation, and (b) shifting nodes between any two adjacent layers of the network.

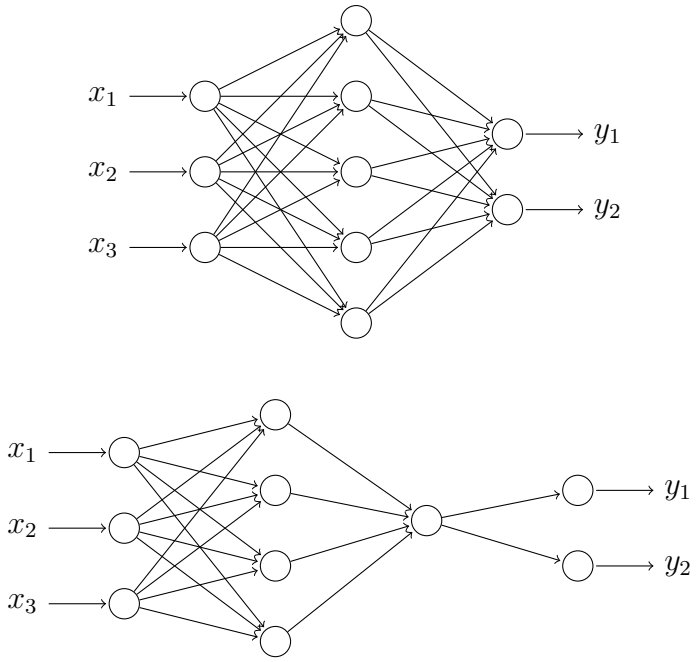

Fig. 1: Example of a single-node layer creation operation: [3,5,2] →[3,4,1,2]-such operations always reduce S (in this case from S=25 to S=18).

3.2. Layer creation:

Consider three consecutive layers of a network having x, k and z nodes. A layer-creation operation can be depicted as,

$$x, k, z \rightarrow x, k-p, p, z\ ,$$

where $k \geq 2$ and of course $x, z \geq 1$. The change in S going from left to right can be shown to be,

$$\Delta S = -p^2 + p(k + z - x) - kz = -px + (k - p)(p - z) \, .$$

Then, in the case where a single-node new layer is created, i.e. for $p = 1$, ΔS is always negative. This implies that starting from any network structure, we can create iteratively new single-node layers from layers having more than two nodes until we arrive at the (unique) serial network each time reducing S. Therefore the serial network is by construction the network with the lowest S, i.e. the most efficient network. Of course in practice experience shows that such networks do not perform well because all inputs are fed to the single node of the first layer which produces a single output. Nevertheless, having knowledge of the most efficient network structure is still useful since we now have a bound on the fastest possible network structure we could potentially train and compare with tabulation.

3.3. Node-shifting:

Consider four consecutive layers of a network having x, k, y and z nodes. A node-shifting operation can be depicted as,

$$x, k, y, z \rightarrow x, k - p, y + p, z \ \ ,$$

where $k \geq 2$ and $x, y, z \geq 1$. Then the change in S is given by,

$$\Delta S = p \, (\Gamma - p) = -p^2 + \Gamma p \, ,$$

where $\Gamma = (k + z) - (x + y)$ and $p \in [-(y - 1), (k - 1)]$. The equation above is second-order in p with a zero at $p = 0$ and at $p^* = \Gamma$. At $p^* = \Gamma/2$ it has a maximum i.e. $\Delta S_{max} = \Gamma^2/4 \geq 0$. The sign of Γ determines the behaviour of ΔS as follows,

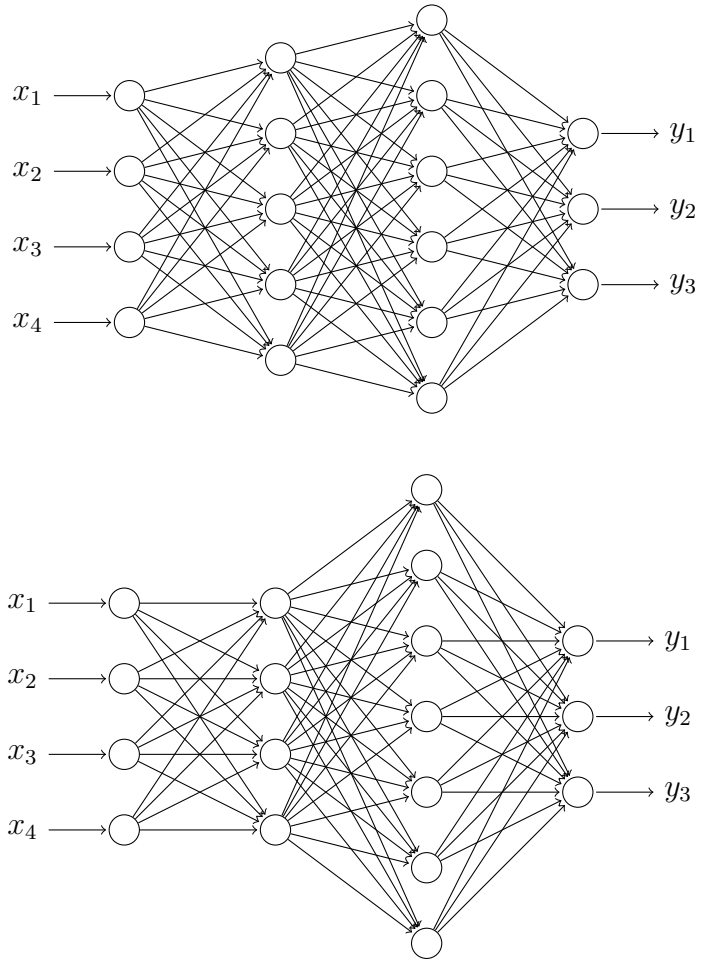

Fig. 2: Example of a single-node shift operation: [4,5,6,3] $\rightarrow$ [4,4,7,3]-for this particular case S reduces as $\Gamma = (5+3) - (4+6) = -2$. Going to [4,6,5,3] would increase S instead.

$$\Gamma \begin{cases} = 0, \forall \mathrm{p} \in [-(y-1), k-1] \text{ and } \neq 0: & \Delta S < 0 \\ > 0, \forall \mathrm{p} \in [0, \Gamma]: & \Delta S \geq 0 \\ > 0, \forall \mathrm{p} \notin [0, \Gamma]: & \Delta S < 0 \\ < 0, \forall \mathrm{p} \in [-\Gamma, 0]: & \Delta S \geq 0 \\ < 0, \forall \mathrm{p} \notin [-\Gamma, 0]: & \Delta S < 0 \end{cases}$$

In all cases we observe that provided all layers have at least two nodes, there always exists a direction to reduce S by shifting nodes left or right until no such operations are possible. On the other end, the maximum possible S is achieved by shifting nodes such that each time S is increased. At the maximum point, no further operations to increase S are possible. These operations can be applied collectively to each group of four in a multi-layer network and the total change in S will be equal to the sum of the individual changes. As we will demonstrate in the next section, the node-shifting operations can be used to construct the most expensive network structures given the number of layers.

3.4. The most expensive network structures

Consider a network with N_i inputs, N_o outputs, and N nodes in the hidden layers. In the case $L = 3$, $S^3 = NN_i + NN_o$. In the case $L = 4$ we seek the structure which maximises S^4. Let the structure be of the form $N_i, x, N - x, N_o$. Then $S^4 = -x^2 + x(N + N_i - N_o) + NN_o$ which is maximised at $x = (N + N_i - N_o)/2$ i.e. $max(S^4) = (N + N_i - N_o)^2/4 + NN_o$. Since the nodes are integer numbers two possible cases exist: (a) if the numerator is even, a single structure with maximum S exists, and (b) if the numerator is odd, there are two possible structures with maximum (and same) S. Note that if N is even, the maximum possible value is always attained while if N is odd, the maximum value is never attained. Also, in the case where $N + N_i - N_o \leq 1$, the most expensive network structure is the one which has a single node in the first hidden layer, and if $N + N_i - N_o \geq 2N - 1$ the most expensive structure is the one having $N - 1$ nodes in the second layer. In both of these cases, S is less than $max(S^4)$ i.e. less than $(N + N_i - N_o)^2/4 + NN_o$. As a result, the supremum of the set of all possible $max(S^4)$ values occurs if $N + N_i - N_o$ is even, and if $(N + N_i - N_o)/2$ lies in the interval $[1, N - 1]$. Overall, the most

expensive 4-layer structure can be written compactly as follows,

$$max(S^4)\begin{cases} \mathbf{2 \le N + N_i - No \le 2(N-1)}: \\ \quad \mathbf{(N + N_i - N_o)\ even}: \\ \quad\quad N_i, \frac{N+N_i-No}{2}, \frac{N+N_o-Ni}{2}, N_o \\ \quad \mathbf{(N + N_i - N_o)\ odd:} \\ \quad\quad N_i, [\frac{N+N_i-No}{2}], N - [\frac{(N+N_i-No}{2}], N_o \\ \quad\quad N_i, [\frac{N+N_i-No}{2}] + 1, N - [\frac{N+N_i-No}{2}] - 1, N_o \\ \mathbf{N + N_i - No \le 1}: \\ \quad N_i, 1, N-1, N_o \\ \mathbf{N + N_i - No \ge 2N - 1}: \\ \quad N_i, N-1, 1, N_o \end{cases}$$

Another interesting result is that in the cases where $max(S^4)$ is attained, then provided the following inequality holds,

$$N_i + N_o - 2\sqrt{N_i No} < N < N_i + N_o + 2\sqrt{N_i No}\,,$$

the 3-layer network is always more expensive than the 4-layer network. At the end-points of the interval above, the most expensive 3 and 4-layer structures have equal S and outside of the range above, the 4-layer network is always more expensive.

In the case of a 5-layer network the same process can be applied. Let the 5-layer structure be denoted as $N_i, x, y, N-x-y, N_o$. Then $S^5 = -y^2 + y(N - N_o) + x(N_i - N_o) + NN_o$. The relationship between x and y depends on the number of nodes in the last layer. Let $N - x - y = p$. Then $S^5 = -y^2 + y(N - N_i) + (N - p)(N_i - N_o) + NN_o$. For a given number of nodes p in the third hidden layer, S^5 is maximised for $y = (N - N_i)/2$. If $N_i > N_o$ then S^5 is decreasing in p and over all maximum values for given p, the one for $p = 1$ will be the highest of all. If $N_i = N_o$, then the choice of p (and x) does not matter and many possible maxima exist–the exact number can be found from the possible permutations of the remaining $N - (N - N_i)/2$ nodes. In

the case where $N_i < N_o$, analogous statements apply i.e. S^5 is maximised for $x = 1$, while y remains the same and swapping N_i with N_o. In the case where $N - N_i < 0$ S^5 is maximised by setting $x = N - 2$ if $N_i > N_o$ or $p = N - 2$ otherwise. Similar arguments hold for the $N - N_o$ case. In any of these cases however, as in the case of the 4-layer network, the maximum S^5 attained will be less than for the cases above. Overall we have for the 5-layer network,

$$
\max(S^5)\begin{cases}
\mathbf{N_i} > \mathbf{N_o}: \\
\quad (\mathbf{N}-\mathbf{N_i}) \leq \mathbf{1}: \\
\quad\quad N_i, N-2, 1, 1, N_o \\
\quad (\mathbf{N}-\mathbf{N_i}) \geq \mathbf{2}: \\
\quad\quad (\mathbf{N}-\mathbf{N_i})\ \mathbf{even}: \\
\quad\quad\quad N_i, \frac{N+N_i}{2}-1, \frac{N-N_i}{2}, 1, N_o \\
\quad\quad (\mathbf{N}-\mathbf{N_i})\ \mathbf{odd}: \\
\quad\quad\quad N_i, N-[\frac{N-N_i}{2}]-1, [\frac{N-N_i}{2}], 1, N_o \\
\quad\quad\quad N_i, N-[\frac{N-N_i}{2}]-2, [\frac{N-N_i}{2}]+1, 1, N_o \\
\mathbf{N_i} < \mathbf{N_o}: \\
\quad (\mathbf{N}-\mathbf{N_o}) \leq \mathbf{1}: \\
\quad\quad N_i, 1, 1, N-2, N_o \\
\quad (\mathbf{N}-\mathbf{N_o}) \geq \mathbf{2}: \\
\quad\quad (\mathbf{N}-\mathbf{N_o})\ \mathbf{even}: \\
\quad\quad\quad N_i, 1, \frac{N-N_o}{2}, \frac{N+N_o}{2}-1, N_o \\
\quad\quad (\mathbf{N}-\mathbf{N_o})\ \mathbf{odd}: \\
\quad\quad\quad N_i, 1, [\frac{N-N_o}{2}], N-[\frac{N-N_o}{2}]-1, N_o \\
\quad\quad\quad N_i, 1, [\frac{N-N_o}{2}]+1, N-[\frac{N-N_o}{2}]-2, N_o \\
\mathbf{N_i} = \mathbf{N_o}: \\
\quad (\mathbf{N}-\mathbf{N_i}) \leq \mathbf{1}: \\
\quad\quad N_i, N-2, 1, 1, N_o \\
\quad\quad N_i, 1, 1, N-2, N_o \\
\quad (\mathbf{N}-\mathbf{N_i}) \geq \mathbf{2}: \\
\quad\quad (\mathbf{N}-\mathbf{N_i})\ \mathbf{even}: \\
\quad\quad\quad N_i, a, \frac{N-N_i}{2}, b, N_o: \qquad a, b \in \{a+b = N-\frac{N-N_i}{2}\} \\
\quad\quad (\mathbf{N}-\mathbf{N_i})\ \mathbf{odd}: \\
\quad\quad\quad N_i, a, [\frac{N-N_i}{2}], b, N_o: \qquad a, b \in \{a+b = N-[\frac{N-N_i}{2}]\} \\
\quad\quad\quad N_i, c, [\frac{N-N_i}{2}]+1, d, N_o: \qquad a, b \in \{c+d = N-[\frac{N-N_i}{2}]-1\}
\end{cases}
$$

Table 1 shows a few examples for each case which may be verified either by running a few simulations and sorting the networks based on S or by using the node-shifting rules defined in the previous section. For instance if we take the first case

N_i	N_o	N	S^3	4	S^4	5	S^5
4	8	30	272	[4,13,17,8]	409	[4,1,11,18,8]	357
8	4	25	132	[8,14,11,4], [8,15,10,4]	310	[8,16,8,1,4],[8,15,9,1,4]	268
8	15	6	210	[8,1,5,15]	88	[8,1,1,4,15]	73
23	11	31	594	[23,22,9,11], [23,21,10,11]	803	[23,26,4,1,31]	717
127	43	54	7783	[127,53,1,43]	6827	[127,52,1,1,43]	6700
7	7	11	126	[7,6,5,7], [7,5,6,7]	107	[7,a,2,b,7] a+b=9	81
5	5	20	125	[5,10,10,5]	200	[5,a,7,b,5], [5,c,8,d,5] a+b=13, c+d=12	156

Table 1: Examples of most expensive 4-layer and 5-layer structures for various number of inputs, outputs and number of nodes in the hidden layers.

in the table [4,13,17,8], we have $\Delta = 13 + 8 - 4 - 17 = 0$, so any node we move left or right reduces S, therefore it is a maximum. For the five layer case [4,1,11,18,8] $\Delta_1 = 4$ and $\Delta_2 = 0$. However we can not move nodes from the single-node layer to the right, while any node movement from the second quartet will reduce S. Therefore we cannot increase S further and it is therefore a maximum.

From table 1, we observe that the most expensive 4-layer structures are always more expensive than the most expensive 5-layer structures. This is not a coincidence and applies in general as we now prove. The most expensive 5-layer structure(s) always contain at least one layer with a single node. Based on the layer-creation rules S always decreases when creating a $k+1$ layer structure from a k-layer structure by creating a single-node new layer. Therefore there always exists a 4-layer structure which is always more expensive than the most expensive 5-layer structure. As a result, the most expensive 4-layer structure is always more expensive than the most expensive 5-layer structure.

We also observe that in the case $N_i > N_o$, the most expensive network structures are the ones which have the nodes gathered closest to input side and vice versa if $N_o > N_i$. In the case $N_i = N_o$ gathering the nodes to either side amounts to the

same result since the networks obtained are reflections of one another. For instance [5,10,7,3,5] has the same maximum S as [5,3,7,10,5]. For networks with more than 5 layers some experimentation by running different test-cases, and varying N_i, N_o, N while finding the corresponding most expensive structure for a given number of layers, reveals that the same pattern applies: nodes are gathered towards the input side if $N_i > N_o$ and the output side otherwise, and in addition at least one layer exists with a single node for the most expensive structure. Therefore, based on experimentation we deduce that for $k \geq 4$ the most expensive k-layer structure is always more expensive than the $k+1$-layer structure. Practically, in order to obtain the $k+1$ most expensive structure from the k-layer structure, we may proceed as follows: if for instance $N_i > N_o$ then create a new layer from the rightmost layer of the k-layer network which contains at least two nodes, and then perform node-shifting operations until S can no longer increase. For instance in order to obtain the most expensive 5-layer network structure from the [8,15,10,5] case in Table 1 we proceed as follows,

$$\begin{aligned} 8,15,10,4 &\rightarrow 8,15,9,1,4(\Delta_1 = -1, \Delta_2 = -2) \\ &\rightarrow 8,16,8,1,4 \\ 8,16,8,1,4 &\rightarrow 8,16,7,1,1,4(\Delta_1 = 2, \Delta_2 = -9, \Delta_3 = -3) \\ &\rightarrow 8,15,8,1,1,4(\Delta_1 = 0, \Delta_2 = -7, \Delta_3 = -4) \end{aligned}$$

and we have obtained the most expensive 6-layer structure and so on. Having found the most expensive and least expensive network structures we now have bounds to compare against tabulation.

4. ANN/Tabulation comparison

The ratios T_{tab}/T_{ann} and M_{tab}/M_{ann} depend on the network structure and are bounded by the minimum and maximum S-value of the network, which as shown in

the previous section depends on the network structure. The least expensive network structure was shown to be the serial network, which is also expected to be the least accurate. The most expensive network structure is either the column network or a 4-layer network depending on the number of inputs (parameters), number of outputs (variables) and number of nodes. The compute time and memory bounds are then given by,

$$r_e = \frac{3N_pN_v + 5N_v(2^{N_p} - 1)}{T^e_{Ann}} \leq \frac{T_{tab}}{T_{ann}} \leq \frac{3N_pN_v + 5N_v(2^{N_p} - 1)}{3N + N_v + 2(N_p - 1)} = r_s \ , \tag{8}$$

$$q_e = \frac{N_v \prod_{r=1}^{N_p} n_r}{M^e_{Ann}} \leq \frac{M_{tab}}{M_{ann}} \leq \frac{N_v \prod_{r=1}^{N_p} n_r}{2N + N_v + (N_p - 1)} = q_s \ , \tag{9}$$

where S-value of the serial-network has been applied for the upper bounds. It is important to stress that r_e and r_s depend only on N, N_p and N_v, while q_e, q_s depend additionally on the table resolution. The lower bounds can be found using the results in section 3 (a small piece of code has been developed specifically for this task). Equations 8 and 9 can be used to calculate a priori the maximum possible gains/losses in compute time and memory requirements of fully-connected artificial neural networks over tabulation. For example suppose N_p=4, N_v =8, N = 50 and that the table resolution is 50x100x50x100. Then we find using the analysis in section 3 that the most expensive structure is the four-layer network [4,23,27,8]. Under these conditions we have $0.37 \leq T_{tab}/T_{ann} \leq 4.24$ and $204290.1 \leq M_{tab}/M_{ann} \leq 1801801.8$. Therefore the fastest 50-node network structure (serial network) will be at most 4.24 times faster than tabulation. At the other extreme, if the most expensive 50-layer network structure is used ([4,23,27,8]) it will be 1/0.37=2.7 times slower than tabulation.

In terms of memory, we see that for both extremes the memory gain over tabulation is quite substantial even for the most expensive network structure. At the

time of writing the maximum available memory in HPC clusters per physical core is around 5Gb. In practice, not all of this memory will be allocated for the table alone since the code itself may also requires a substantial amount of memory. If 1Gb of memory is allocated for the tabulation alone, which is a more reasonable choice for most applications, then we can calculate the bounds for a given number of nodes, input parameters, and targets. These are shown in tables 3 and 4 for the compute-time and memory bounds respectively for $N = 100$. In any case, the equations above can be used to derive case-specific bounds for conditions different than the ones in the tables. From the values in the tables, we observe that for increasing N_p and/or N_v the lower time bounds increase, which implies that ANNs become more favourable over tabulation in terms of compute time. The opposite is true for the memory gain, however the memory gains are so large that remain relatively un-affected by the increase in N_p and/or N_v. The exact number of parameters and variables for which ANNs become more favourable than tabulation in terms of compute time, depends on the total number of nodes N in the hidden layers.

Working backwards, and requiring $T_{tab}/T^e_{ann} > 1$, we can find the corresponding number of threshold values N^*_p for a given number of variables N_v, above which any network structure will be faster than tabulation. The exact values of N^*_p have been calculated for up to twenty variables/targets and for N ranging from 25-200 and are shown in Table 2. For instance, we observe that for $N_v = 5$, then provided $N_p \geq N^*_p$ =9, tabulation is always slower than any 100-node neural network. The values in Table 2 can be used as a guideline in cases where the objective is to develop networks which are faster than tabulation.

N_v/N	25	50	75	100	125	150	175
1	7	9	10	11	11	12	12
2	6	8	9	10	10	11	11
3	6	7	8	9	10	10	11
4	5	7	8	9	9	10	10
5	5	7	8	8	9	10	10
6	5	6	7	8	9	9	10
7	5	6	7	8	9	9	10
8	4	6	7	8	8	9	9
9	4	6	7	8	8	9	9
10	4	6	7	8	8	9	9
11	4	6	7	7	8	8	9
12	4	6	7	7	8	8	9
13	4	6	6	7	8	8	9
14	4	5	6	7	8	8	9
15	4	5	6	7	8	8	9
16	4	5	6	7	8	8	8
17	4	5	6	7	7	8	8
18	4	5	6	7	7	8	8
19	4	5	6	7	7	8	8
20	4	5	6	7	7	8	8

Table 2: Number of parameters N_p^* above which the compute time cost of tabulation first exceeds that of the most expensive network structure having N nodes. For $N_p \geq N_p^*$ tabulation is always slower than any N-node network structure.

N_p/N_v	1	2	3	4	5	6	7
1	(1.51e-03,2.66e-02)	(2.96e-03,5.30e-02)	(4.36e-03,7.92e-02)	(5.71e-03,1.05e-01)	(7.01e-03,1.31e-01)	(8.27e-03,1.57e-01)	(9.47e-03,1.82e-01)
2	(3.89e-03,6.93e-02)	(7.64e-03,1.38e-01)	(1.13e-02,2.07e-01)	(1.47e-02,2.75e-01)	(1.81e-02,3.42e-01)	(2.13e-02,4.09e-01)	(2.45e-02,4.76e-01)
3	(8.00e-03,1.44e-01)	(1.57e-02,2.88e-01)	(2.32e-02,4.30e-01)	(3.04e-02,5.71e-01)	(3.73e-02,7.12e-01)	(4.40e-02,8.52e-01)	(5.05e-02,9.90e-01)
4	(1.55e-02,2.83e-01)	(3.05e-02,5.65e-01)	(4.50e-02,8.45e-01)	(5.90e-02,1.12e+00)	(7.26e-02,1.40e+00)	(8.56e-02,1.67e+00)	(9.83e-02,1.95e+00)
5	(2.98e-02,5.50e-01)	(5.86e-02,1.10e+00)	(8.65e-02,1.64e+00)	(1.13e-01,2.18e+00)	(1.39e-01,2.72e+00)	(1.65e-01,3.25e+00)	(1.89e-01,3.78e+00)
6	(5.73e-02,1.07e+00)	(1.13e-01,2.13e+00)	(1.66e-01,3.19e+00)	(2.18e-01,4.24e+00)	(2.69e-01,5.29e+00)	(3.17e-01,6.32e+00)	(3.65e-01,7.35e+00)
7	(1.11e-01,2.10e+00)	(2.18e-01,4.18e+00)	(3.22e-01,6.25e+00)	(4.23e-01,8.30e+00)	(5.21e-01,1.03e+01)	(6.16e-01,1.24e+01)	(7.07e-01,1.44e+01)

Table 3: Time gain bounds T_{tab}/T_{ann}: $N = 100$, memory=1.0Gb

N_p/N_v	1	2	3	4	5	6	7
1	(4.63e+04,6.22e+05)	(4.55e+04,6.19e+05)	(4.46e+04,6.16e+05)	(4.38e+04,6.13e+05)	(4.30e+04,6.10e+05)	(4.23e+04,6.07e+05)	(4.15e+04,6.04e+05
2	(4.55e+04,6.19e+05)	(4.46e+04,6.16e+05)	(4.39e+04,6.13e+05)	(4.31e+04,6.10e+05)	(4.23e+04,6.07e+05)	(4.16e+04,6.04e+05)	(4.09e+04,6.01e+05
3	(4.46e+04,6.16e+05)	(4.39e+04,6.13e+05)	(4.31e+04,6.10e+05)	(4.24e+04,6.07e+05)	(4.17e+04,6.04e+05)	(4.10e+04,6.01e+05)	(4.03e+04,5.98e+05
4	(4.38e+04,6.13e+05)	(4.31e+04,6.10e+05)	(4.24e+04,6.07e+05)	(4.17e+04,6.04e+05)	(4.10e+04,6.01e+05)	(4.03e+04,5.98e+05)	(3.97e+04,5.95e+05
5	(4.30e+04,6.10e+05)	(4.23e+04,6.07e+05)	(4.17e+04,6.04e+05)	(4.10e+04,6.01e+05)	(4.03e+04,5.98e+05)	(3.97e+04,5.95e+05)	(3.91e+04,5.92e+05
6	(4.23e+04,6.07e+05)	(4.16e+04,6.04e+05)	(4.10e+04,6.01e+05)	(4.03e+04,5.98e+05)	(3.97e+04,5.95e+05)	(3.91e+04,5.92e+05)	(3.85e+04,5.90e+05
7	(4.15e+04,6.04e+05)	(4.09e+04,6.01e+05)	(4.03e+04,5.98e+05)	(3.97e+04,5.95e+05)	(3.91e+04,5.92e+05)	(3.85e+04,5.90e+05)	(3.79e+04,5.87e+05

Table 4: Memory gain bounds M_{tab}/M_{ann}: $N = 100$, memory/physical core=1.0Gb

N_p/N_v	1	2	3	4	5	6	7
1	(2,3,296)	(3,5,352)	(4,8,389)	(4,10,417)	(4,12,440)	(4,15,460)	(4,17,478)
2	(4,7,377)	(5,13,448)	(7,20,495)	(7,27,531)	(8,33,561)	(8,40,587)	(9,47,609)
3	(6,14,453)	(9,28,539)	(11,42,596)	(13,57,640)	(14,71,676)	(15,85,707)	(16,100,734)
4	(9,27,537)	(13,56,639)	(17,85,707)	(20,113,759)	(22,142,802)	(24,171,839)	(25,199,871)
5	(13,54,635)	(20,111,756)	(25,167,836)	(29,223,898)	(32,280,949)	(35,336,993)	(38,392,1032)
6	(19,108,752)	(29,219,894)	(36,329,990)	(42,440,1063)	(48,551,1124)	(52,661,1176)	(56,772,1222)
7	(28,215,891)	(43,433,1060)	(53,652,1173)	(62,870,1261)	(70,1088,1333)	(76,1307,1395)	(83,1525,1449)

Table 5: Node bounds (N_T^e, N_T^s, N_{MT}) assuming 1.0E09 Gb of memory/physical core is dedicated for tabulation (double precision for each variable). Any N-node network structure with $N < N_T^e$ will be faster than tabulation. Any N-node network structure with $N > N_T^s$ will be slower than tabulation. Any N-node network structure with $N < N_{MT}$ will have the memory gain over tabulation outweigh the time loss.

An alternative way to use these results is to find the corresponding number of nodes, given N_p and N_v, such that tabulation will always be slower. This can be accomplished as before if we calculate N such that $r_e \leq 1$ for the first time, call this N_T^e. Then, any network structure with $N < N_T^e$ will be faster than tabulation. At the other extreme, there is an upper bound for the least expensive network structure and a corresponding upper bound N_T^s for the number of nodes. Putting both together we have the following results:

-If $N < N_T^e$ then any N-node network structure will be faster than tabulation (from the most expensive down to the least expensive).

-If $N > N_T^s$ then no N-node network structure is faster than tabulation (including the fastest structure, i.e. the serial network).

-If $N_T^e < N < N_T^s$ then some network structures might be faster than tabulation.

In terms of memory, similar bounds, say N_M^s, above which any network structure will be more memory-intensive than tabulation, can be found. This however is not

very useful given the large memory gains observed for ANNs. A more useful perhaps bound is the point at which the memory gain outweighs the compute time loss. This puts a limit N_{MT} for the most expensive network structure. Then any network structure with $N < N_{MT}$ will have the memory gain outweigh the time loss. If this limit is exceeded, the memory gain does not outweigh the time loss and there is no benefit in using ANNs over tabulation. All of the above three bounds are shown in Table 5 in the case where 1Gb of memory is dedicated to tabulation (the corresponding resolution for each number of parameters is adjusted accordingly).

5. Conclusions

Models for the computational cost (computational time and memory) of fully-connected artificial neural networks and structured tables (fixed resolution spacing) have been developed in order to compare between the two approaches, and in order to derive theoretical bounds on the maximum possible time and memory gains/losses of neural networks over tabulation. Even though the context is in reacting flow applications, the results of this work are general and can be applied to a plethora of applications where tabulation and/or neural networks are used to solve regression problems. The main conclusions of this work are as follows:

(i) The computational time and memory requirement ratios of tabulation over neural networks T_{tab}/T_{ann} and M_{tab}/M_{ann} are bounded as given by Eqs. 8 and 9 respectively. These bounds correspond to the computationally most expensive and least expensive neural network structures. The time bounds depend on the number of nodes in the hidden layers N, the number of inputs N_p, and number of outputs N_v, while the memory bounds additionally depend on the table resolution. The upper bounds

are straightforward to calculate using these equations. In order to obtain the lower bounds (time and memory), the most expensive network structure must first be identified, and the computational time and memory requirements of that structure obtained. This can be done as follows,

(1) Calculate the number of trainable weights, $S^3 = N(N_p + N_v)$*, for the 3-layer network:* $[N_p, N, N_v]$*.*

(2) Identify the most expensive 4-layer network structure with N_p *inputs,* N *nodes and* N_v *outputs, using the rules in section 3, and then calculate the corresponding* S^4 *for the identified network structure.*

(3) Take $max(S^3, S^4)$ *to obtain the most expensive structure, and use equations 4 and 6 to calculate the lower bounds in Eqs. 8 and 9.*

(ii) There exists a number of parameters N_p^* for a given number of target variables N_v and nodes N, above which tabulation is always slower than any N-node fully-connected neural network structure with N_p^* inputs and N_v outputs. In such cases neural networks may be the preferred choice and tabulation otherwise if computational time is the only criterion.

(iii) There exists a number of nodes N_T^e, which depends on N_p and N_v, below which any N-node fully-connected neural network structure with N_p inputs and N_v outputs is faster than tabulation. This limit is relatively low, and the trained network may be of poor accuracy, however if in the event that the regression problem is such that the network is of sufficient accuracy, then the neural-network approach should be preferred over tabulation.

(iv) There exists a number of nodes N_T^s, which depends on N_p and N_v,

above which any fully-connected neural network structure with N_p inputs and N_v outputs is slower than tabulation. If the sole criterion is computational time this limit should never be exceeded.

(v) There exists a limit on the number of nodes N_{MT}, which depends on N_p, N_v N, and the table resolution, below which the memory gain of the network over tabulation will outweigh the computational time loss over tabulation. N_{MT} is not necessarily large then N_T^s, and this limit should never be exceeded.

A small piece of code for calculating the computational time and memory bounds but also for calculating N_p^*, N_T^e, N_T^s and N_{MT} is given as supplementary material. The results of this work can be used a priori in order to obtain estimates of the possible gains of neural networks over tabulation (if any) and thus to determine which approach to take (neural networks/tabulation). Furthermore, they can be used as a guideline to help choose the training strategy and/or the number of nodes and/or network structure(s) for the specific needs, which may for instance involve optimising in memory rather than compute time.

Acknowledgement

The REDAFLOW project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 101019855.

References

[1] N. Peters. Laminar diffusion flamemelt models in non-premixed turbulent combustion. *Prog. Energy Combust. Sci.*, 10:319–339, 1984.

[2] F.C. Christo, A.R. Masri, and E.M. Nebot. Artificial neural network implementation of chemistry with pdf simulation of H2/CO2 flames. *Combust. Flame*, 106:406–427, 1996.

[3] S. B. Pope. Computationally efficient implementation of combustion chemistry using in situ adaptive tabulation. *Combust. Theory Modelling*, 1:41–63, 1997.

[4] J.A. Blasco, N. Fueyo, C. Dopazo, and J. Ballester. Modelling the temporal evolution of a reduced combustion chemical system with an artificial neural network. *Combust. Flame*, 113:38–52, 1998.

[5] B.A. Sen, E.R. Hawkes, and S. Menon. Large eddy simulation of extinction and reignition with artificial neural networks based chemical kinetics. *Combust. Flame*, 157:566–578, 2010.

[6] K. Wan, C. Barnaud, L. Vervisch, and P. Domingo. Chemistry reduction using machine learning trained from non-premixed micro-mixing modeling: Application to DNS of a syngas turbulent oxy-flame with side-wall effects. *Combust. Flame*, 220:119–129, 2020.

[7] A.D. Cook, J.J. Riley, and G. Kosaly. A laminar fiamelet approach to subgrid-scale chemistry in turbulent flows. *Combust. Flame*, 109:332–341, 1997.

[8] C.D. Pierce and P. Moin. Progress-variable approach for large-eddy simulation of non-premixed turbulent combustion. *J. Fluid Mech.*, 504:73–97, 2004.

[9] B. Fiorina, O. Gicquel, L. Vervisch, S. Carpentier, and N. Darabiha. Approximating the chemical structure of partially premixed and diffusion counterflow flames using fpi flamelet tabulation. *Combust. Flame*, 140:147–160, 2005.

[10] M. Ihme, C.M. Cha, and H. Pitsch. Prediction of local extinction and re-ignition effects in non-premixed turbulent combustion using a flamelet/progress variable approach. *Combust. Flame*, 30:793–800, 2005.

[11] P. Domingo, L. Vervisch, and D. Veynante. Large-eddy simulation of a lifted methane jet flame in a vitiated coflow. *Combust. Flame*, 152:415–432, 2008.

[12] E. Knudsen, H. Kolla, E.R. Hawkes, and H. Pitsch. LES of a premixed jet flame dns using a strained flamelet model. *Combust. Flame*, 160:2911–2927, 2013.

[13] S. Popp, S. Hart, D. Butz, D. Geyer, A. Dreizler, and L. Vervisch and C. Hasse (2021). Assessing multi-regime combustion in a novel burner configuration with large eddy simulations using tabulated chemistry. *Proc. Combust. Inst.*, 38(2):2551–2558, 2021.

[14] G. P. Smith, D. M. Golden, M. Frenklach, N. W. Moriarty, B. Eiteneer, M. Goldenberg, C. T. Bowman, R. K. Hanson, S. Song, W. C. Gardiner, V. V. Lissianski, and Z. Qin. Technical report, 1999. http://www.me.berkeley.edu/gri-mech/.

[15] M. Milano and P. Koumoutsakos. Neural network modelling for near wall turbulent flow. *J. Comput. Phys.*, 182:1–26, 2002.

[16] M. Schoepplein, J. Weatheritt, R. Sandberg, M. Talei, and M. Klein. Application of an evolutionary algorithm to LES modelling of turbulent transport in premixed flames. *J. Comput. Phys.*, 374:1166 – 1179, 2018.

[17] K.T. Carlberg, A. Jameson, M.J. Kochenderfer, J. Morton, L. Peng, and F.D. Witherden. Recovering missing cfd data for high-order discretizations using deep neural networks and dynamics learning. *J. Comput. Phys.*, 395:105–124, 2019.

[18] C.J. Lapeyre, A. Misdariis, N. Cazard, D. Veynante, and T. Poinsot. Training convolutional neural networks to estimate turbulent sub-grid scale reaction rates. *Combust. Flame*, 203:255–264, 2019.

[19] A. Seltz, P. Domingo, L. Vervisch, and Z. Nikolaou. Direct mapping from LES resolved scales to filtered-flame generated manifolds using convolutional neural networks. *Combust. Flame*, 210:71–82, 2019.

[20] Z. M. Nikolaou, C. Chrysostomou, L. Vervisch, and S. Cant. Progress variable variance and filtered rate modelling using convolutional neural networks and flamelet methods. *Flow Turb. Combust.*, 103:485–501, 2019.

[21] Z. Nikolaou, Y. Minamoto, and L. Vervisch. Unresolved stress tensor modeling in turbulent premixed v-flames using iterative deconvolution: An a priori assessment. *Phys. Rev. Fluids*, 4:063202, 2019.

[22] F. Flemming, A. Sadiki, and J. Janicka. LES using artificial neural networks for chemistry representation. *Progr. Comp. Fluid Dynamics*, 5:375–385, 2005.

[23] M. Ihme, A. Marsden, and H. Pitsch. Generation of optimal artificial neural networks using a pattern search algorithm: Application to approximation of chemical systems. *Neural Comp.*, 20:573–601, 2008.

[24] M. Ihme, C. Schmitt, and H. Pitsch. Optimal artificial neural networks and tabulation methods for chemistry representation in LES of a bluff-body swirl-stabilized flame. *Proc. Combust. Inst.*, 32:1527–1535, 2009.

[25] O. Owoyele, P. Kundu, M. Ameen, T. Echekki, and S. Som. Application of deep artificial neural networks to multi-dimensional flamelet libraries and spray flames. *Intern. J. Engine Res.*, 21:151–168, 2020.

[26] R. Ranade, G. Li, S. Li, and T. Echekki. An efficient machine-learning approach for pdf tabulation in turbulent combustion closure. *Combust. Sci. Tech.*, 193 (7):1258–1277, 2021.

[27] M. Hansinger, Y. Ge, and M. Pfitzner. Deep residual networks for flamelet/progress variable tabulation with application to a piloted flame with inhomogeneous inlet. *Combust. Sci. Tech.*, 0:1–27, 2020.

[28] A. Seltz, P. Domingo, and L. Vervisch. Solving the population balance equation for non-inertial particles dynamics using pdf and neural networks: Application to a sooting flame. *Phys. Fluids.*, 33(013311), 2021.