Applied Soft Computing Journal 101 (2021) 107074

Contents lists available at ScienceDirect

Applied Soft Computing Journal

journal homepage: www.elsevier.com/locate/asoc

An integrated approach to predict scalar fields of a simulated turbulent jet diffusion flame using multiple fully connected variational autoencoders and MLP networks

Ryno Laubscher *, Pieter Rousseau

Department of Mechanical Engineering, Applied Thermal-Fluid Process Modelling Research Unit, University of Cape Town, Library Rd, Rondebosch, Cape Town, 7701, South-Africa

ARTICLE INFO

Article history:
Received 10 June 2020
Received in revised form 16 November 2020
Accepted 30 December 2020
Available online 2 January 2021

Keywords:
Combustion computational fluid dynamics
Surrogate modelling
Variational autoencoders
Deep neural networks

ABSTRACT

A novel integrated deep learning approach for data-driven surrogate modelling of combustion computational fluid dynamics (CFD) simulations is presented. It combines variational autoencoders (VAEs) with deep neural networks (DNNs) to predict detail cell-by-cell two-dimensional distributions of temperature, velocity and species mass fractions from high level inputs such as velocity and fuel and air mass fractions. The VAE model is used to generate low dimensional encodings of the CFD data and the DNN is used in turn to map boundary conditions to the encodings. The results show that regularization is required during all training phases. Sufficiently accurate results were achieved for the reproduced species mass fractions with mean average errors below 0.3 [%wt.]. The validation mean average percentage errors for the temperature and velocity fields are 1.7% and 7.1% respectively. It is therefore possible to predict detail two-dimensional contours of CFD solution data with adequate generalizability and accuracy.

1. Introduction

With increased renewable energy penetration onto electrical grids, future combustion-based gas and steam turbine power plants would be required to have 70%–100% faster-ramping rates, 35%–60% faster start-up rates and 20%–80% lower emissions than the current state of the art systems [1]. These requirements would result in significant thermal stresses induced in various components such as heat exchangers and turbine blades due to the rapid changes in plant heat loads [2,3] and require careful monitoring of emission levels at the outlet of the combustion chambers [4,5]. It is, therefore, necessary to be able to predict the local fluid flow, temperature, species concentration and emission distributions within combustion system components to effectively monitor, control, design and analyse such systems. Computational Fluid Dynamics (CFD) modelling lends itself well to the analysis of combustion systems due to the relatively accurate integrated solution of 2D/3D fluid flow, chemical reactions, as well as radiative and convective heat transfer phenomena [6]. However, due to the expensive computational overheads of these types of simulations for industrial-sized systems, they are often not employed for in-situ performance monitoring and control, such as the prediction of 2D/3D temperature, velocity and concentration fields inside a combustion chamber in real-time, or for exploration of the design space.

To circumvent the high computational burden, surrogate models with increased computational efficiency are typically used. A surrogate model is a statistical regression model of the true underlying objective function (CFD simulation model in the present work) which is typically fitted using true evaluations of the underlying objective function [7]. There are mainly three categories of surrogate models namely; multi-fidelity models, reduced-order models and data-driven models [8]. The first two categories are based on physical models and are accurate under certain conditions, but tend to be sensitive to model input parameters, which leads to insufficient model robustness [9]. Conversely, data-driven models can interpolate between simulation data points and therein learn the hidden trends in the data. The learning process whereby the data-driven models generalize to known or new trends is facilitated by machine learning algorithms such as support vector machines (SVM) [10], artificial neural networks (ANN) [11] and Kriging models [12]. CFD data-driven surrogate models that generate 2D or 3D solution fields could have a range of applications. These include visualization of processes inside equipment, deployment of virtual sensors, performing what-if analysis, exploring the design space, linking to an optimization routine to effectively control equipment, detecting anomalies, or forming part of lower-dimensional system simulation.

* Corresponding author.
E-mail address: ryno.laubscher@uct.ac.za (R. Laubscher).

https://doi.org/10.1016/j.asoc.2020.107074
1568-4946/

Of course, data-driven models could also be applied directly to experimental data to build predictive models of combustion systems [5,13,14]. However, suitable measurement locations are very limited in combustion systems and only certain global parameters can be measured such as total combustion air flow rate and fuel flow rate, in the case of gas combustion systems. Local quantities such as temperature distributions and velocity fields are typically not measured. Accurately measuring local quantities at multiple locations is difficult (due to fluctuating quantities, etc.) and expensive due to the harsh environment within the combustion system, which requires special materials. Furthermore, data-driven models trained only on experimental and operating data of an industrial combustion chamber are typically not accurate or comprehensive enough for performance monitoring [4] or design space exploration. Therefore, CFD solution data is preferred.

Various authors have looked at the development of data-driven surrogate models using data obtained from CFD solutions [15]. Shi et al. [4] augmented experimental data taken from a 660 MWe coal-fired boiler by using CFD simulation data. The hybrid experimental-CFD dataset was used to develop a predictive model which uses plant operating parameters such as load level and coal properties as inputs to predict boiler NOx emissions and thermal efficiency. The predictive model was linked to a genetic optimization algorithm that was employed to determine the airflow distribution into the boiler to minimize emissions and maximize thermal efficiency. Wang et al. [16] coupled CFD simulation results of a proton exchange membrane fuel cell model with a catalyst layer model to generate a simulated dataset. The dataset was then used to develop a surrogate model using an SVM. The SVM model was developed to predict the generated current density as the target variable. Warey et al. [17] developed a CFD model to simulate thermal comfort within the cabin of a passenger vehicle. The CFD model was used to generate 400 solution datasets which were, in turn, used to develop an ANN surrogate model. The equivalent homogeneous temperature experienced by each passenger was extracted from the CFD solution data and set as the required target variable. Various input boundary conditions such as vehicle orientation, solar load and vehicle speed were used as input variables to the ANN model.

In the work referred above [4,16,17] the CFD solution data was post-processed and global results extracted, including area-weighted temperatures, mass-weighted species mass fractions and summed current densities. These were then used as zero-dimensional target variables for the surrogate models. In contrast to this, the following authors have developed surrogate models that predict 2D/3D fields such as velocity or pressure, where each value in the target vector corresponds to a specific cell position in the CFD grid.

Ti et al. [18] developed a data-driven model that predicts 3D vector and scalar fields of velocity and turbulence intensity at the outlet of a wind turbine. Each entry in the 3D target vector corresponded to a cell in the volume mesh of the wake. The model used mass-weighted averaged velocity and turbulence intensity at the inlet of the turbine as inputs. In this work the feature space dimensionality was equal to two and the target vector dimensionality was approximately 240 000. To guard against overfitting the authors generated 2000 ANN sub-models which each trained on the same input values but predicted a subset of the original target vector. The discrepancy between the input and target dimensionality is a typical problem if one wants to develop a data-driven model using CFD data. This is because the number of input boundary conditions are orders of magnitude smaller than the solution data which has the same size as the mesh's cell count. For neural networks, if the target dimensionality is much larger than the feature space, the model tends to overfit the data.

Wu et al. [9] and Deng et al. [19] developed data-driven surrogate models which uses deep learning techniques to predict the 2D flow around aerofoils using the CFD boundary conditions as network inputs. [19] used a convolutional-autoencoder to compress the CFD solution data into latent vectors (encodings) which had a much lower dimensionality than the original CFD solution space. An additional deep feedforward neural network (DNN) model was used to predict the latent vectors using the CFD model input boundary conditions. [9] used a convolutional neural network in conjunction with a generative adversarial network (GAN) to produce a one-to-one mapping from the parameters defining the aerofoil geometry to the resultant 2D pressure field.

In the present work an integrated deep learning approach is proposed that combines variational autoencoders (VAE) with DNNs to predict the 2D temperature, velocity and species concentration fields of a turbulent jet diffusion flame using high level input conditions such as the fuel flow rate and inlet oxygen mass fraction. A steady-state CFD model of the well-known Sandia Flame D [20] experimental setup was developed and used to generate solution data for a 1000 different input conditions. The CFD model was configured to store the target datasets for each simulation run comprising of the mass fractions of six gaseous species, and temperature and velocity values for each CFD cell. VAEs are used to compress the 2D target data of each field into a lower-dimensional latent vector encoding. DNNs are then applied to learn the relationship between the high-level CFD model boundary conditions (inputs) and the encodings. Once trained the DNN model is supplied with high level physical boundary conditions such as fuel flow rate and inlet oxygen mass fraction, and then outputs a latent vector encoding. The latent vector is then fed back into the trained VAE model, which then decodes the encoding to the full 2D solution field of a specific variable such as temperature or velocity.

The novelty of the current paper is the application of a combination of VAEs and DNNs to develop an integrated deep learning approach that can predict multiple detail thermofluid solution fields (velocity, temperature, etc.) on a cell-by-cell basis for a 2D computational domain of a gaseous jet diffusion flame by using high level inlet boundary conditions as inputs to the predictive model.

2. Data generation

A steady-state CFD model was employed to generate the target data that will be used to train the VAE and DNN models. Fig. 1 shows an image of the Sandia Flame D experiment taken from [20]. The experimental flame consists of three parallel inlets namely the fuel jet, co-flow and pilot inlets.

2.1. CFD model setup

In the current study, the commercial CFD software package ANSYS® Fluent 2019 R3 was used to model the fluid flow, heat transfer and chemical reactions of the Sandia Flame D experiment. The experimental setup is modelled as a 2D axisymmetric slice. The geometry details can be seen in Fig. 2. The numerical grid was generated using unstructured quadrilateral elements and the mesh had a total of $N_{mesh} = 8586$ cells.

The main boundary conditions of the CFD model are the three velocity inlets namely the co-flow inlet, pilot inlet and the fuel inlet. The co-flow inlet supplies ambient air (N_2 and O_2) to the domain, which in turn feeds oxidizer to the combustion front. The pilot inlet feeds high-temperature flue gas (CO_2, H_2O and O_2) which is used to ignite the fuel and sustain the diffusion flame by acting as a thermal trigger. Finally, the fuel inlet injects low-temperature high-velocity unburnt hydrocarbon gas and oxidizer

Fig. 1. Image of the Sandia Flame D experiment from [20], rotated clockwise through 90 degrees.

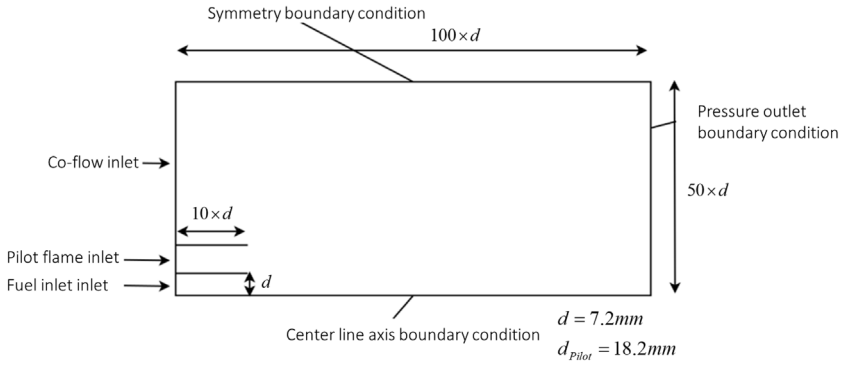

Fig. 2. Computational domain for Sandia Flame D (not to scale).

(CH_4 and O_2) into the domain. The boundary conditions used to simulate the Sandia Flame D experiment are provided in Table 1. In Table 1, V [m/s] is velocity, T[K] temperature and Y species mass fraction.

The mass (continuity), momentum, species and energy transport equations were solved using a Eulerian framework approach. The transport equations can be seen in Eq. (1) below.

$$\text{Continuity: } \frac{\partial}{\partial x_i}(\rho u_i) = 0$$

$$\text{Momentum: } \frac{\partial}{\partial x_i}\left(\rho u_i u_j\right) + \frac{\partial P}{\partial x_j} = \frac{\partial}{\partial x_i}\left[\mu\left\{\frac{\partial u_j}{\partial x_i} + \frac{\partial u_i}{\partial x_j} - \frac{2}{3}\delta_{ij}\frac{\partial u_i}{\partial x_i}\right\}\right] + \frac{\partial}{\partial x_i}\left(-\rho u_j' u_i'\right) \quad (1)$$

$$\text{Species: } \frac{\partial}{\partial x_i}\left(\rho u_j Y_k\right) = -\frac{\partial}{\partial x_j}\left(\vec{J}_k\right) + R_k$$

$$\text{Energy: } \frac{\partial}{\partial x_i}\left(u_i\left[\rho E + P\right]\right) = \frac{\partial}{\partial x_j}\left[\lambda_{eff}\frac{\partial T}{\partial x_j}\right] + S_h$$

In the above equations, x_i is the spatial dimension, u [m/s] directional velocity, P [Pa] static pressure, E [J/kg] total energy and S_h [W/m^3] energy source term. The density ρ [kg/m^3] of the gas mixture in the equations was solved using the ideal gas law and the mean molecular weight of the mixture in a specific cell, which in turn is a function of the constituent mass fractions Y_k. The transport equations were time-averaged using the Reynolds-averaging approach and the fluctuating Reynolds stress component in the momentum equation $\rho u_j' u_i'$ was approximated using the well-known Boussinesq equation [6] as seen below, where the turbulent viscosity is calculated as $\mu_t = \rho C_\mu k^2/\varepsilon$.

$$-\rho u_i' u_j' = \mu_t\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) - \frac{2}{3}\left(\rho k + \mu_t\frac{\partial u_k}{\partial x_k}\right)\delta_{ij} \quad (2)$$

To solve for the turbulent kinetic energy k[m^2/s^2] and turbulent kinetic energy dissipation rate ε [m^2/s^3] the realizable $k-\varepsilon$turbulence model was selected based on its ability to resolve swirling and spreading jet flows more accurately than the standard $k-\varepsilon$ turbulence model [21] which is due to the modification of its dissipation rate equation. The wall effects near the nozzles were modelled using standard wall functions in Fluent.

To resolve the transport of gas mixture constituents throughout the domain the species-transport approach was adopted which solves a species mass fraction Y_k transport equation for each of the k species in the gaseous mixture. The gas mixture comprised of the following components: CH_4, CO_2, CO, O_2, H_2, H_2O and N_2. In Eq. (1) above $\vec{J}_k$ is the mass diffusion term of species k and is determined by $\vec{J}_k = -\left(\rho D_{k,m} + \mu_t/Sc_t\right)\cdot\partial/\partial x_j\left(Y_k\right)$, where $D_{k,m}$ is the mass diffusion coefficient for species k into the mixture m, and in the present work is set to a constant value of $2.88\cdot10^{-5}$ [m^2/s]. The effective thermal conductivity in Eq. (1) is calculated as $\lambda_{eff} = \lambda + c_{p,m}\mu_t/\mathrm{Pr}_t$ with $c_{p,m}$[J/kg K] as the mixture specific heat, which is in turn calculated as a mass-weighted average of the individual constituent specific heats evaluated at the cell temperature. To determine the net reaction rates R_k for species k and to account for the effect of turbulence on the chemical reactions, the eddy dissipation-finite rate (EDM-FR) combustion model [22] with default model constants was used.

The purpose of the current work is to compare the integrated surrogate modelling methodology with the CFD data rather than to validate the accuracy of the CFD results. Therefore, although there are more accurate turbulence-chemistry interaction models available, such as the eddy dissipation concept model, the EDM-FR model was selected due to its lower computational requirements [23]. The combustion reactions were modelled using the modified Jones and Lindstedt (JL2) global chemical mechanism [24] shown in Eq. (3) along with the chemical kinetics parameters namely pre-exponential factor A, activation energy E_a

Table 1
CFD model boundary conditions.

Boundary condition name	Type	Values
Top wall	Symmetry	–
Centreline axis	Axis	–
Co-flow	Velocity inlet	$V_{coflow} = 0.9$ [m/s], $T_{coflow} = 300$ [K] $Y_{O_2,coflow} = 0.233$, $Y_{N_2,coflow} = 0.767$
Pilot inlet	Velocity inlet	$V_{pilot} = 11.4$ [m/s], $T_{pilot} = 1880$ [K] $Y_{O_2,pilot} = 0.056$, $Y_{H_2O,pilot} = 0.092$ $Y_{CO_2,pilot} = 0.110$, $Y_{N_2,pilot} = 0.742$
Fuel inlet	Velocity inlet	$V_{fuel} = 49.6$ [m/s], $T_{pilot} = 300$ [K] $Y_{O_2,fuel} = 0.19664$, $Y_{CH_4,fuel} = 0.15607$, $Y_{N_2,fuel} = 0.64729$

and reaction orders for each of the chemical reaction equations.

$$
\begin{aligned}
&CH_4 + 1.5O_2 \rightarrow CO + 2H_2O \\
&\quad A = 4.4 \cdot 10^{10}, E_a = 1.254 \cdot 10^8 \text{ J/kg K}, [CH_4]^{0.5}[O_2]^{1.25} \\
&CH_4 + H_2O \rightarrow CO + 3H_2 \\
&\quad A = 3 \cdot 10^8, E_a = 1.254 \cdot 10^8 \text{ J/kg K}, [CH_4][O_2] \\
&CO + H_2O \leftrightarrow CO_2 + H_2 \\
&\quad \text{Forward: } A = 2.75 \cdot 10^9, \\
&\quad E_a = 8.36 \cdot 10^7 \text{ J/kg K}, [CO][H_2O] \\
&\quad \text{Reverse: } A = 6.71 \cdot 10^{10}, \\
&\quad E_a = 1.138 \cdot 10^8 \text{ J/kg K}, [CO_2][H_2] \\
&H_2 + 0.5O_2 \leftrightarrow H_2O, A = 4.4 \cdot 10^{10} \\
&\quad \text{Forward: } E_a = 7.97 \cdot 10^{10} \text{ J/kg K}, [H_2][O_2]^{0.5} \\
&\quad \text{Reverse: } E_a = 3.48 \cdot 10^{13} \text{ J/kg K}, [H_2O]
\end{aligned}
\tag{3}
$$

The simulations were solved using the SIMPLE pressure–velocity coupling scheme. The pressure term was discretized using the PRESTO! scheme. Momentum, species and energy equations were discretized using the second-order upwind scheme and the turbulent kinetic energy and dissipation rate using the first-order upwind scheme. The convergence criteria for the simulation model was set to 1E−3 for the continuity equation, 1E−4 for the velocity equations and 1E−6 for the remaining transport equations. For more details on the modelling of the experimental setup see [25].

2.2. Simulated dataset

As stated earlier the aim of the integrated deep learning model developed in the present work is to predict the 2D gas temperature, velocity magnitude and species mass fraction fields using various high-level inlet conditions as input data. Therefore, the T, V, Y_{CH_4}, Y_{CO_2}, Y_{CO}, Y_{H_2}, Y_{H_2O} and Y_{O_2} vectors of length N_{mesh} containing the cell values are stored for each converged simulation case and are designated as the target variable vectors. The inlet boundary condition values are V_{fuel}, $Y_{O_2,fuel}$, $Y_{CH_4,fuel}$ and V_{pilot}. The input feature data vector dimensionality is therefore $d_{feature} = 4$. In order to obtain a representative set of results a design of experiments (DOE) was conducted to generate a set of 1000 simulation cases. The ranges selected for the DOE model inputs are shown in Table 2. These ranges were selected to cover a wide range of operation without causing flame blow-off or local extinction since the EDM-FR turbulence chemistry interaction model is not suited for simulating such phenomena due to its fast chemistry assumption [26].

Using the input ranges, the DOE matrix was populated using optimal-space filling with a min–max distance design type. A total of $N_{DOE} = 1000$ design points were created and the CFD simulations were performed for each design point. Once converged, the combined target data (all the target variable vectors) and input data were stored for each case. The high-level input dataset size is $\overline{X} \in \mathbb{R}^{1000 \times d_{feature}}$ where the rows of the matrix represent the different dataset entries and the columns the dataset features. The target dataset size is $\overline{Y} \in \mathbb{R}^{1000 \times N_{mesh} \times 8}$. The rows of the target dataset are entries corresponding to the input dataset row entries. The columns of the target dataset are the cell-by-cell values of each solution variable and the 3rd dimension of length 8 corresponds to each solution variable stored (temperature, velocity, methane mass fraction, etc.). 80% of the input dataset entries and corresponding target dataset entries are used for training of the deep learning models and the remaining 20% of the data is used for testing of the models. To reduce the training time of the neural networks, min–max normalization was applied on all datasets, which scales all entries to values between $0 \rightarrow 1$.

Table 2
Input ranges for the design of experiments simulations.

Input variable	Min	Max	Units
Fuel inlet velocity, V_{fuel}	30	60	m/s
Fuel inlet oxygen mass fraction, $Y_{O_2,fuel}$	0.175	0.22	kg/kg
Fuel inlet methane mass fraction, $Y_{CH_4,fuel}$	0.14	0.17	kg/kg
Pilot inlet velocity, V_{pilot}	8	15	m/s

3. Background on implemented deep learning models

The present work combines supervised and unsupervised learning to develop an integrated model configuration which can predict target variable vectors containing fluid and flow quantities using four inlet boundary conditions as high level inputs to the models. The VAE generative learning model is used to compress the original target vectors into low-dimensional encodings. The encodings will then act as the target vectors for the supervised learning DNN regression model, which uses the inlet boundary conditions (variables in DOE matrix) as inputs. The current section discusses the details of the respective deep learning models.

3.1. Deep feedforward neural networks

Deep feedforward neural networks, also known as multi-layer perceptron (MLP) networks, are inspired by the functionality of the animal brain which has billions of interconnected neurons that can process complex input signals [27]. The DNN architecture belongs to the supervised learning family of neural networks which is used to maximize the conditional probability $P(\overline{y}^i|\overline{x}^i)$ where $\overline{x}^i$ is the input row vector for the ith observation in dataset $\overline{X}$ and $\overline{y}^i$ is the ith target row vector in dataset $\overline{Y}$. In maximizing the conditional probability for all observations in the input and target datasets the DNN maps the input vectors to the target vectors. Therefore, it essentially learns a high dimensional target function that minimizes some loss function. The trained target function comprises of a network of interconnected computing units called neurons. A neural network consists of three main sections namely an input layer of neurons, several hidden layers

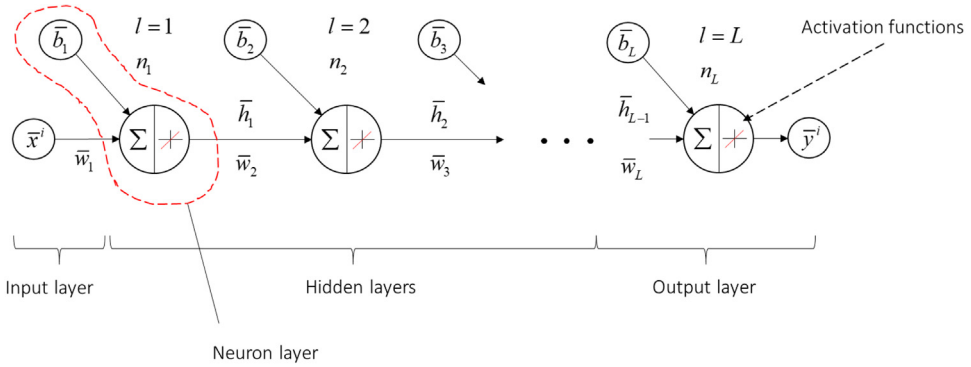

Fig. 3. Deep feedforward neural network architecture.

of neurons and an output layer [28] as shown in Fig. 3. Here l is the layer index, n_l is the number of neurons in layer l, $\bar{h}_l$ is the hidden layer output vector of size $1 \times n_l$, $\bar{w}_l$ is the weights matrix of layer l which has a size of $n_{l-1} \times n_l$ and $\bar{b}_l$ is the lth layer bias row vector of size n_l. The arrow connections between layers shown in the figure indicate a dot product of the incoming signal (either $\bar{x}^i$ or $\bar{h}_l$) and the corresponding weight matrices.

To calculate the output of a feedforward neural network the forward propagation algorithm is employed, which calculates the output from each hidden layer starting at the input layer and moving sequentially through the network until the final output is calculated. Each layer output is calculated in two steps, the first being the calculation of the summed input signal row vector $\bar{s}_l$ of size $1 \times n_l$ for layer l shown in Eq. (4).

$$\bar{s}_l = \bar{h}_{l-1} \cdot \bar{w}_l + \bar{b}_l \tag{4}$$

Once $\bar{s}_l$ is calculated it is passed into the layer's activation function f generating the layer output vector $\bar{h}_l$. Various activation functions may be used such as linear, ReLu, leaky-ReLu, sigmoid and hyperbolic-tangent. In the present work, only linear and leaky-ReLu functions are utilized as described in Eqs. (5) and (6). The output layers of regression neural networks are typically modelled using the linear activation function [28], whereas the hidden layers use the ReLu or one of its variants. Xu et al. [29] showed that leaky variants of the ReLu activation function always outperform the standard ReLu activation function, therefore the leaky variant was selected in the present work.

$$\bar{h}_l = f_{linear}(\bar{s}_l) = \bar{s}_l = \bar{h}_{l-1} \cdot \bar{w}_l + \bar{b}_l \tag{5}$$

$$\bar{h}_l = f_{leaky-ReLu}(\bar{s}_l) = \begin{cases} \bar{s}_l & \text{if} \quad \bar{s}_l > 0 \\ 0.01 \times \bar{s}_l & \text{otherwise} \end{cases} \tag{6}$$

At layer $l = L$ the output vector calculated is the target vector of the network namely $\bar{h}_L = \bar{y}^i$ for the ith dataset observation. To ensure the network accurately maps the input vectors to the target vectors the network parameters $\bar{w}_l$, $\bar{b}_l$ should be optimized to minimize some cost function $J(\bar{y}^i, \hat{y}^i)$ using the actual target values $\hat{y}^i$. For regression networks, the mean-squared error cost function is typically used as a cost function, shown in Eq. (7). Note, Eq. (7) is for a single data point. If mini-batches are used the cost function should also be averaged over the number of mini-batches.

$$J_{MSE}\left(\bar{y}^i, \hat{y}^i\right) = \frac{1}{d_{t\ \arg\ et}} \sum_{j=1}^{d_{t\ \arg\ et}} \frac{1}{2}\left(\bar{y}^i - \hat{y}^i\right)^2 \tag{7}$$

In Eq. (7), $d_{t\ \arg\ et} = n_L$ (number of neurons in the output layer). To optimize the network parameters the back-propagation algorithm [30] is implemented which calculates the gradient of the cost function with respect to each trainable parameter. The calculation methodology to determine these gradients can be seen in the algorithm shown in Eq. (8) below [31].

Algorithm 1 (*Back-propagation*).

Compute gradient of the output layer with respect to target variable:
$\bar{g} \leftarrow \nabla_{\hat{y}} J_{MSE}$
Calculate pre-activated gradient of output layer:
$\bar{g} \leftarrow \nabla_{\bar{s}_L} J_{MSE} = \bar{g} \odot f'(\bar{s}_L)$
Compute gradients on weights and biases in output layer:
$\nabla_{\bar{b}_L} J_{MSE} = \bar{g}$
$\nabla_{\bar{w}_L} J_{MSE} = \bar{g} \cdot \bar{h}_{L-1}{}^T$ (8)
Loop through hidden layers and compute gradients on weights and biases:
$\bar{g} \leftarrow \nabla_{\bar{h}_l} J_{MSE} = \bar{w}_l^T \cdot \bar{g}$
$\bar{g} \leftarrow \nabla_{\bar{s}_l} J_{MSE} = \bar{g} \odot f'(\bar{s}_l)$
$\nabla_{\bar{b}_l} J_{MSE} = \bar{g}$
$\nabla_{\bar{w}_l} J_{MSE} = \bar{g} \cdot \bar{h}_{l-1}{}^T$
Repeat till input layer reached.

Once the gradients are calculated the trainable parameters are adjusted using the gradient descent algorithm, or one of its variants such as RMSProp or Adam [31], and a new forward propagation iteration is performed. In the present work the Adam algorithm was implemented using the gradients calculated by the back-propagation algorithm. The Adam algorithm is shown in Eq. (9) below for the trainable parameter θ (weight or biases).

Algorithm 2 (*Adam*).

$$\begin{aligned} &\bar{m} \leftarrow \beta_1 \bar{m} + (1-\beta_1)\nabla_\theta J\left(\bar{\theta}\right) \\ &\bar{s} \leftarrow \beta_1 \bar{s} + (1-\beta_2)\nabla_\theta J\left(\bar{\theta}\right) \otimes \nabla_\theta J\left(\bar{\theta}\right) \\ &\bar{m} \leftarrow \frac{\bar{m}}{1-\beta_1{}^t} \\ &\bar{s} \leftarrow \frac{\bar{s}}{1-\beta_2{}^t} \\ &\bar{\theta} \leftarrow \bar{\theta} - \eta \bar{m} \otimes \left(\sqrt{\bar{s}+\varepsilon}\right)^{-1} \end{aligned} \tag{9}$$

The scaling $\bar{s}$ and momentum $\bar{m}$ matrices are initialized to 0 at the start of the training phase, t is the iteration counter, β_1 is the momentum decay hyperparameter and is set of 0.9, and β_2 is the scaling decay hyperparameter and is set to 0.999. η is the learning rate hyperparameter, which is varied to find the optimal model performance and ε is the smoothing term that is set to 10^{-8}.

The forward- and backward-propagation steps are iteratively applied so that for every successive iteration the selected cost function output should decrease monotonically.

3.2. Variational autoencoders

Whereas the DNN architecture belongs to the supervised learning faction of neural networks, the VAE architecture belongs

to the unsupervised learning faction as well as the generative learning family. Before VAEs are discussed a brief background of classical autoencoders will be provided.

Autoencoders are neural networks capable of learning lower-dimensional encodings of unlabelled data making them useful for dimensionality reduction and as powerful feature detectors [28]. These networks comprise of three parts namely (1) an encoder section, (2) a decoder section (which can be constructed from several fully connected, convolutional or recurrent neural network layers), and (3) a latent layer which outputs the lower dimensional encoding of the original input data. The encoder section has a converging structure from the input layer to the latent layer. The latent layer has l_Z number of neurons where $l_Z \ll d_{input}$. The vector $\overline{z}$ that flows out of the latent layer is the encoding vector. The encoding vector is fed into the decoder section which has the same structure as the encoder but in a diverging configuration. The output from the decoder is the reconstructed input data.

To train an autoencoder the neural network is constructed similarly to DNNs and trained using the back-propagation algorithm along with some variant of the gradient descent approach and a cost function. One of the major differences is that the target data used during training of an autoencoder is the same as its input data. Depending on the application, various cost functions can be used along with autoencoders, such as the mean-squared error loss, which for autoencoders is written as $J_{MSE}(\overline{x}^i, \hat{x}^i) = \frac{1}{d_{input}} \sum_{j=1}^{d_{input}} \frac{1}{2} \left(\overline{x}^i - \hat{x}^i\right)^2$, where $\overline{x}^i$ is the input data and $\hat{x}^i$ is the reconstructed input data. The calculated cost function value for an autoencoder is also called the reconstruction error. Fig. 4 shows an example of a stacked (multi-layer) autoencoder.

Classical autoencoders do not generate structured (normally distributed) latent spaces. This is because only the reconstruction error is minimized during training, which typically leads to overfitting of the data [32] and non-normal latent space distribution. The unstructured latent space results in classical autoencoders not performing well in generative tasks or data compression since it cannot successfully "interpolate" between different classes of input data. This implies that small deviations away from the trained non-normally distributed latent space could result in outputs with significant reconstruction errors. In the context of the current work this would imply that small errors in encodings predicted by the DNN regression model would result in large reconstruction errors in the outputs of the classical autoencoder model. For this reason, it was decided to use a VAE architecture, which during training forces the latent space to be structured (normally distributed).

The VAE architecture was developed by Kingma et al. [33] and Rezende et al. [34] in 2014 and has been successfully applied to numerous applications such as image generation [35] and dimensionality reduction [36]. The VAE learns highly structured encodings by compressing the input data into the parameters of a normal distribution, namely the means $\overline{z}_\mu$ and variances $\overline{z}_\sigma$, compared to the classical autoencoder which compresses the input data into a fixed encoding $\overline{z}$. During the training of VAEs Gaussian noise is added to the latent vector output. The added stochasticity during training improves the robustness and forces the latent space to encode meaningful representations everywhere. Therefore, every point randomly sampled from the latent space is decoded to a valid output [32].

The architecture of the VAE is very similar to the classical autoencoder and differs only at the construction of the latent layer. As mentioned, VAE architecture splits the latent layer into two parts, namely the latent space mean $\overline{z}_\mu$ and the latent space variance $\overline{z}_\sigma$. Once the latent space vectors $\overline{z}_\mu, \overline{z}_\sigma$ are calculated by forward propagation through the encoder section, the next step is to randomly sample a latent vector $\overline{Z}$ drawn from the Gaussian distribution shown in Eq. (10).

$$\overline{Z} \sim \mathrm{N}\left(\overline{z}_\mu, {\overline{z}_\sigma}^2\right) \tag{10}$$

Since the sampled vector which now feeds the decoder section is not produced by a function, but rather by a sampling process which changes every time it is queried, the back-propagation algorithm will not work. To remedy this, the sampling process is transformed into a function by using an underlying random value $\overline{\varepsilon} \sim \mathrm{N}\left(0, \overline{I}\right)$. This transformation is called the reparameterization trick and enables the backpropagation through the sampling operation [31]. The latent vector $\overline{z}$ can now be calculated using Eq. (11) and then be fed as input to the decoder section.

$$\overline{z} = \overline{z}_\mu + \overline{\varepsilon} \cdot \overline{z}_\sigma = \overline{z}_\mu + \mathrm{N}\left(0, \overline{I}\right) \cdot \overline{z}_\sigma \tag{11}$$

Fig. 5 shows the architecture of a VAE. Like classical autoencoders, the encoder hidden layers have a decreasing number of neurons per layer as the signal moves from the input to the latent layer, and vice versa for the decoder section. Note that the latent space means and variances must both have the same number of neurons as the output layer of the encoding.

To ensure that during the training phase the VAE model accurately maps the reconstructed data to the original input data while simultaneously forcing the latent space to be well structured, the cost function must be modified. For VAE training the cost function comprises of two parts, namely the reconstruction loss and the regularization loss. The former is used to ensure that the decoded samples match the initial inputs, and the latter ensures well-structured latent spaces are learned in order to reduce overfitting. For the reconstruction error, the mean-squared error loss can be used for regression tasks or the cross-entropy loss for classification tasks. To force the encodings to be normally distributed in the latent space, the Kullback–Leibler (KL) [37] divergence between a zero-centred normal distribution and the encoding distribution is minimized. The KL divergence between the underlying variable $\overline{\varepsilon}$ and the latent space distribution is calculated using Eq. (12).

$$D_{KL}(\overline{Z}(\overline{x}) \parallel \overline{\varepsilon}) = E_{x \sim Z}\left[\log\left(\overline{Z}(\overline{x})\right) - \log\left(\overline{\varepsilon}\right)\right] \tag{12}$$

After some mathematical manipulation, while recognizing that the mean and variance of the underlying random variable is 0 and 1 respectively, the KL divergence can be written as [32]

$$\begin{aligned} D_{KL}(\overline{Z} \parallel \overline{\varepsilon}) &= D_{KL}\left(\overline{Z} \parallel \mathrm{N}\left(0, \overline{I}\right)\right) \\ &= -\frac{1}{2}\left[1 + \log\left(\sigma_{\overline{Z}}\right) - \mu_{\overline{Z}}{}^2 - \exp(\sigma_{\overline{Z}})\right] \end{aligned} \tag{13}$$

In Eq. (13) $\sigma_{\overline{Z}}$, $\mu_{\overline{Z}}$ are the standard deviation and mean of the latent space distribution $\overline{Z}$.

The combined loss for the VAE using an MSE reconstruction loss and a summed regularization loss is

$$\begin{aligned} J_{VAE,loss}\left(\overline{x}^i, \hat{x}^i, \overline{z}_\mu, \overline{z}_\sigma\right) &= \sum_{j=1}^{d_{input}} \frac{1}{2}\left(\overline{x}^i - \hat{x}^i\right)^2 \\ &- \frac{1}{2}\sum_{k=1}^{l_Z}\left[1 + \log\left(\overline{z}_\sigma\right) - {\overline{z}_\mu}^2 - \exp\left(\overline{z}_\sigma\right)\right] \end{aligned} \tag{14}$$

To minimize the VAE cost function shown in Eq. (14) any back-propagation algorithm can be used. In the next section the integration of the DNN regression model and the VAE compression model is discussed along with their training and validation performance.

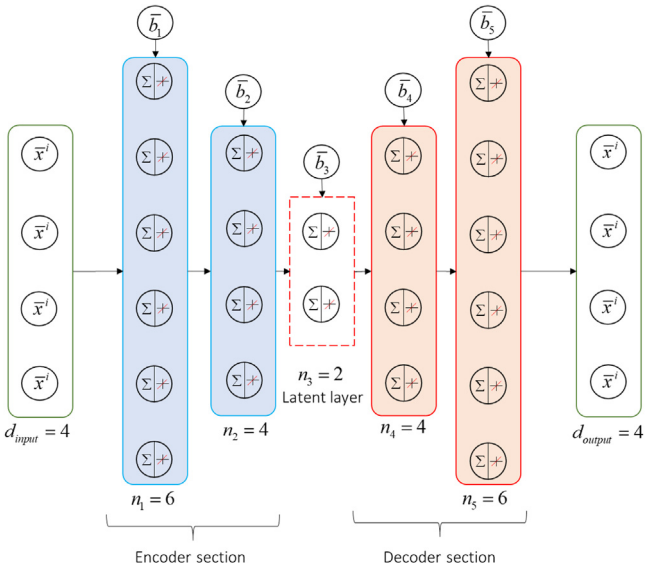

Fig. 4. Stacked autoencoder.

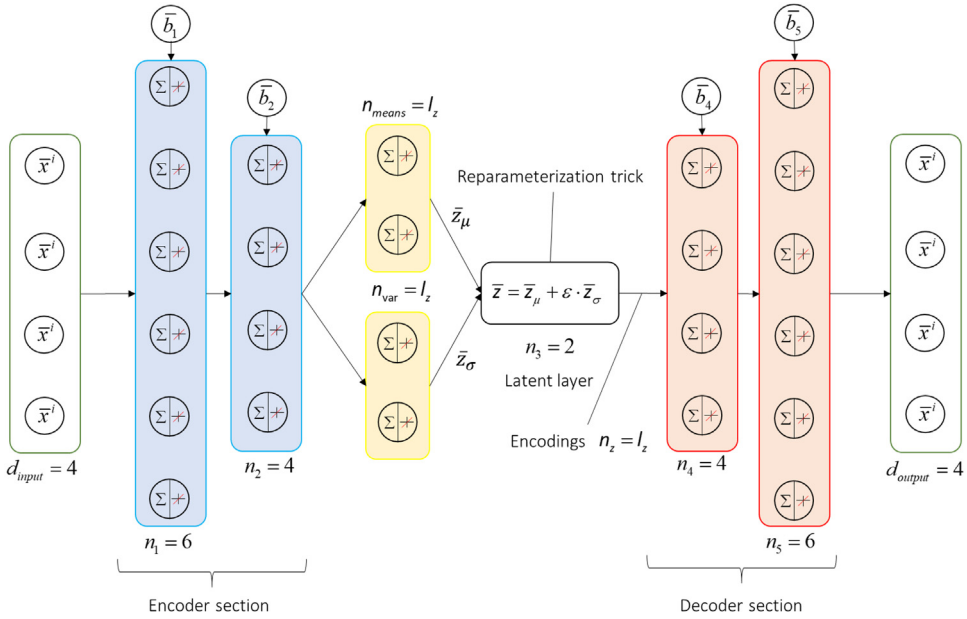

Fig. 5. Variational autoencoder architecture.

4. Model development

The goal of the present work is to develop a deep learning approach that combines VAEs with DNNs to predict the 2D temperature, velocity and species concentration fields of a turbulent jet diffusion flame using high level inputs such as the fuel flow rate and inlet oxygen mass fraction.

As mentioned in Section 2.2 there are eight target variables, namely the gas temperature, velocity and six gas mixture constituent mass fractions. Each target variable dataset has a size of $N_{DOE} \times N_{mesh}$. The input dataset which contains the varied inlet boundary conditions has a size of $N_{DOE} \times d_{feature}$. In the present work, an integrated neural network model comprising of a VAE and a DNN network is developed for each target variable. Therefore, eight sets of DNNs and VAEs are developed, each trained to predict one of the eight target variables. The training of eight sets of integrated network models was done to reduce computational requirements of training a single very large VAE model. Since the input dimension to such a large model would be $8 \times N_{mesh}$, it would result in a very large number of layer connections and therefore in very large parameter matrices. Another elegant method, to circumvent the large number of connections, would be to use multi-channel convolutional layers for the VAE network, similar to the work of [19]. The use of convolutional layers in the VAE model will be the focus of future work and the current segregated approach will then act as a benchmark for the convolutional variant.

4.1. Integrated model configuration

A graphical summary of the proposed methodology to develop and use the integrated model is provided in Fig. 6.

The first step is to generate a target data set with 1000 entries with the aid of the CFD model, based on a high-level input data set determined using DOE. These target vectors are used to simultaneously train the VAE encoders and decoders. The trained

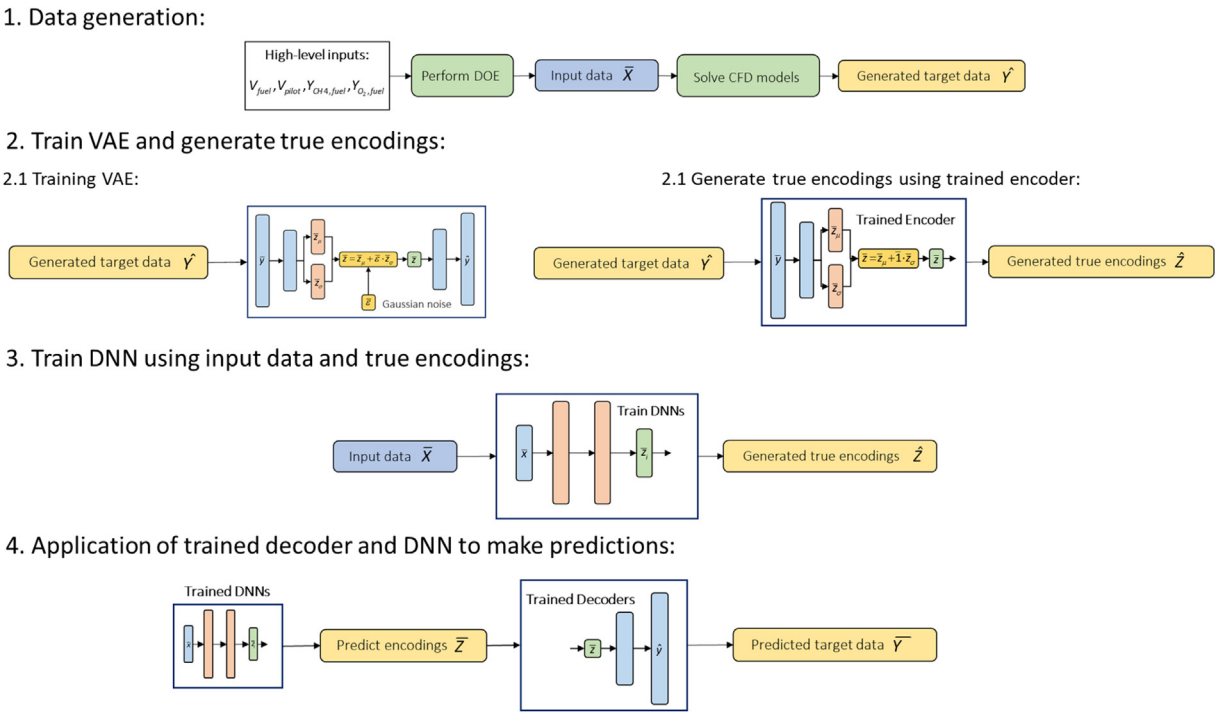

Fig. 6. Integrated model layout, showing training and application configurations.

VAE encoders are then used to generate training data for the DNNs (true encodings), which consist of latent layer encodings associated with each of the high-level data points. The DNNs are then trained separately to predict the latent layer encodings based on the high-level input data. Once trained, the DNNs can predict the latent layer encodings directly from the high-level input data, following which the trained decoders can reproduce the original target data set.

Care must be taken when generating the encodings that are used as the target vectors for the DNN regression models. In the training phase of the VAEs where backpropagation is used, Gaussian noise is added to the latent vector output to force the latent space to encode meaningful representations. When the trained VAE encoders are used to generate encodings for training the DNNs, the Gaussian noise is not added since the true encodings are required and not the statistically perturbed versions. The statistical perturbations are only useful during the VAE network training phase.

The goal is for the integrated DNN-VAE model to generate cell-by-cell species mass fraction percentages with mean absolute errors (MAE) below 1% for the training and validation datasets, and gas velocities and temperatures with mean absolute percentage errors (MAPEs) below 5%.

The integrated DNN and VAE models were developed using the PyTorch [38] library within the Python 3.6.6 programming environment. To find the best performing VAE and DNN architectures the network hyperparameters should be tuned accordingly. An elegant approach to finding the optimal hyperparameters is to use a formal optimization technique, such as a genetic algorithm [39]. However, in the present paper, a coarse grid search was employed in order to develop an understanding of the various influences of the hyperparameters on the VAE and DNN training and validation errors. In the next two subsections, the model configuration and hyperparameter grid search results for the VAEs and DNNs will be discussed.

Table 3
Hyperparameter search space for VAE models.

Parameter	Search space
Neurons per hidden layer	8,64,128,296
Number of hidden layers per VAE section	2,3
Size of latent vector	10,20,40
Learning rates	0.01,0.001,0.0001,1E−5
Mini-batch sizes	4,8,16,32

4.2. Variational autoencoder models

The number of layers per VAE section (decoder and encoder) l, number of neurons per layer n_l, batch size n_{batch}, learning rate η and latent layer number of neurons l_z were all varied to study the effect of network architectural and optimization settings on the training and validation errors and losses. Table 3 contains the hyperparameters used for each of the eight VAE models.

All the layers in the VAEs are fully connected and the activation used throughout the network is the leaky-ReLu function, except for the output layer in the decoder section which uses a sigmoid function. To overcome gradient diffusion and to combat overfitting [40] the Adam optimization algorithm was used to update model parameters during the training phase. The training epochs of each VAE model was set at 500, which proved to be sufficient since training losses remained nearly unchanged for further epochs.

The first hyperparameter search space experiment fixes the learning rate and batch sizes to $\eta = 0.001$, $n_{batch} = 16$ and varies the number of neurons per layer and the number of hidden layers for each latent vector size in Table 3. After the training of each target variable VAE, the loss (Eq. (14)) and mean absolute error (MAE) are reported. The trained model is then used to reconstruct the validation dataset (20% of original target dataset) and the MAE of the validation phase is also reported. The training and validation MAEs are calculated as $MAE = \frac{1}{d_{t\ \arg\ et}} \sum_{i=1}^{d_{t\ \arg\ et}} \left|\vec{y}^i - \hat{y}^i\right|$ with $d_{t\ \arg\ et} = N_{mesh}$. From this equation, one can see that the MAEs are only an indication of the reconstruction error. For the

Index	Architecture
1	2 x 8
2	3 x 8
3	2 x 64
4	3 x 64
5	2 x 128
6	3 x 128
7	2 x 296
8	3 x 296

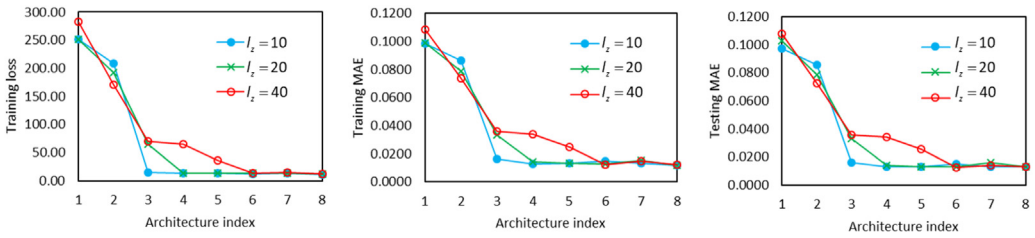

Fig. 7. Variational autoencoder losses for different number of hidden layers and neurons per layer.

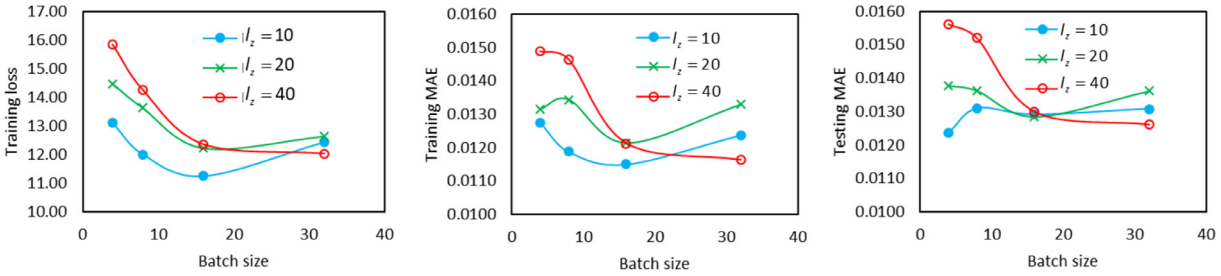

Fig. 8. Variational autoencoder losses for different mini-batch sizes.

sake of brevity, the individual values (MAEs and training losses) were averaged over all the target variables. Eq. (15) shows an example of how the averaged training loss is calculated. The same formulation was used to average all the MAEs.

$$\bar{J}_{VAE,loss} = \frac{1}{8}\left(J_{VAE,loss,T} + J_{VAE,loss,V} + J_{VAE,loss,CO2} + J_{VAE,loss,CO} + J_{VAE,loss,H2O} + J_{VAE,loss,H2} + J_{VAE,loss,O2} + J_{VAE,loss,CH4}\right) \tag{15}$$

Fig. 7 shows the training loss, training MAE and validation MAE for tested architectures indicated by the number of hidden layers x the number of neurons per hidden layer. The results clearly show that at lower capacities the networks have higher training losses and MAEs. As the network capacities increase the training losses converged at a value of approximately 12.0 whereas the training and testing MAEs converge at values of approximately 0.012 and 0.013 respectively. It is interesting to note that for a smaller latent vector size ($l_z = 10$) the network converges to a minimum loss value at smaller network capacities compared to the network architectures with larger latent vector sizes (20, 40). The reason for this is that the architecture with a latent vector size of 10 has a smaller number of trainable parameters which reduces training time of the network. Furthermore, the lower number of trainable parameters also guards against overfitting during training which results in lower testing errors for certain hyperparameter settings, as seen in the results below.

The VAE network capacity with the lowest training loss and MAEs for all three latent vector sizes is architecture 8, which has three hidden layers in the encoder and decoder sections and has 296 neurons per hidden layer. Although it cannot be claimed that architecture 8 necessarily represents the "optimum" combination, it is the "best" combination within the specified range of hyperparameters. Furthermore, the results indicate that a wider range of hyperparameters will not necessarily result in further significant improvements. Architecture 8 is therefore selected for further evaluations.

Using the selected capacities for the VAEs, the effect of mini-batch sizes on the losses and MAEs is investigated. Similar to the capacity investigation, the mini-batch size experiment is implemented using three VAE configuration variants with different latent vector sizes. Fig. 8 shows the averaged losses and MAEs for the tested mini-batch sizes.

From Fig. 8, the results show that in general, for latent vector sizes of 10 and 20, a mini-batch size of 16 yields the lowest losses and MAEs. For the architecture with a latent vector size of 40 a mini-batch size of 32 resulted in the lowest losses and MAEs. These mini-batch sizes are selected for the respected architectures for further evaluations.

Using the selected network capacities and mini-batch sizes as mentioned, the final experiment investigates the effect of learning rate on model losses. Fig. 9 shows the average losses for different learning rates. The results show that in all cases a learning rate of $\eta = 0.001$ performs best.

Table 4 summarizes the selected VAE hyperparameters and the corresponding losses. There is no discernible network configuration that out-performs the other networks. Therefore, DNN regression models will be trained using encodings generated from the three selected VAE network configurations.

4.3. Deep neural network regression models

Similar to the experiments in Section 4.2 the number of neurons per layer, number of hidden layers, mini-batch sizes and learning rates were varied within selected bounds to study its effect on training and testing errors. These hyperparameter combinations were run for DNN regression models with different output layer sizes that correspond to the latent layer sizes used in the VAE models. The target data of the DNN models are the generated encodings using the configurations shown in Table 4 and the input data from the DOE matrix features. Table 5 shows the range of hyperparameters used to find the best performing DNN models.

For the different hyperparameter cases trained in this section only the target variable averaged training and testing MAEs are reported. The DNN models were all configured using leaky-ReLu activation functions for the input and hidden layers. The output layer used a linear activation function. The mean-squared error cost function was used to evaluate the performance of the models during training and the model parameters were adjusted using the Adam optimization algorithm. The number of training epochs used was set to 1000.

For the first experiment, the mini-batch sizes and learning rates were kept constant at $n_{batch} = 16$, $\eta = 0.0001$ and the

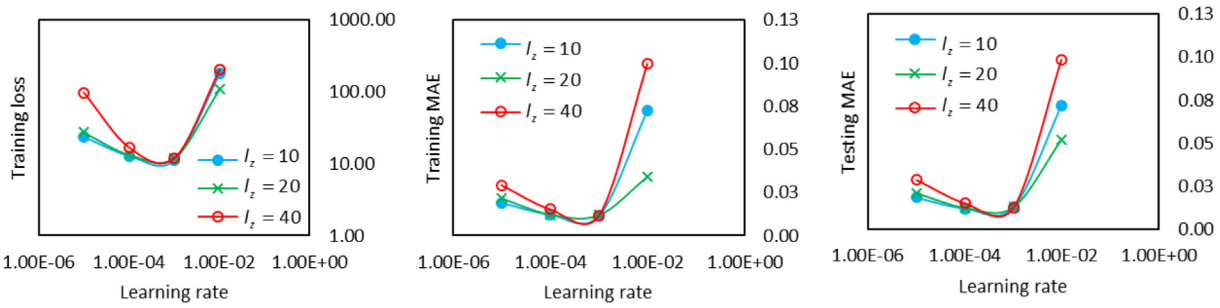

Fig. 9. Variational autoencoder losses for different learning rates.

Table 4
Summary of selected hyperparameters for variational autoencoders and model losses.

Parameter	Latent vector size model designation	$l_z = 10$ (VAE-10)	$l_z = 20$ (VAE-20)	$l_z = 40$ (VAE-40)
Selected hyperparameters				
Neurons per hidden layer		296	296	296
Number of hidden layers per section		3	3	3
Mini-batch size		16	16	32
Learning rate		0.001	0.001	0.001
Model losses				
Training loss (Eq. (14))		11.25	12.05	12.02
Training MAE		0.0115	0.0118	0.0116
Testing MAE		0.0128	0.0125	0.0126

Table 5
Hyperparameter search space for DNN models.

Parameter	Search space
Neurons per hidden layer	1024,2048,4096
Number of hidden layers per VAE section	2,3
Size of output layer	10,20,40
Learning rates	0.01,0.001,0.0001,1E−5
Mini-batch sizes	4,8,16,32

number of layers X the neurons per layer were varied for the three different output layer sizes.

Fig. 10 shows that for the three output layer size variants the models with the largest capacity (architecture 6) perform the best of the tested configurations. Using architecture 6, further model evaluations are performed to investigate the effect of learning rate magnitude on model performance. Fig. 11 shows the training and testing MAEs for the different mini-batch sizes. An interesting phenomenon is seen namely, that the MAEs increase from a batch size of 4 → 8, and then decreases as the batch size is increased. Since smaller batch sizes approximate pure stochastic decent it tends to be less stable [28]. As the batch size increases, the training parameters tend to be more regularized and therefore one would expect improved performance, at least up to a point. However, the exact reason for the initial increase in MAE followed by a decrease is not yet well understood. For the model which has an output layer size of 10 the best performance was achieved using a mini-batch size of four, and for the models with output layer sizes of 20 and 40 the best performance was achieved using a size of 32.

Fig. 12 shows that for output layer sizes of 20 and 40 a learning rate of $\eta = 0.001$ yields the lowest errors for the given number of training epochs, whereas for the model configurations using an output layer size of 10 the best model performance was achieved using $\eta = 0.0001$. Note that decreasing the learning rate further would eventually result in similar or lower errors compared to when $\eta = 0.001$ or $\eta = 0.0001$, but the number of training epochs required to reach convergence would increase significantly.

Table 6 summarizes the selected DNN regression model architectures for each output layer size used for further evaluations and their corresponding training and testing MAEs.

In the next section, the three DNN regression model architectures (for each target variable) along with their corresponding VAE models are used to evaluate the accuracy of the integrated networks.

5. Results and discussion

The best performing integrated model configurations for different latent vector sizes are now used to predict the cell-by-cell target variables. The generated target variable vectors are then compared to the actual CFD simulation results. As mentioned, 80% of the raw dataset was used for training and evaluation of the VAE and DNN models. The remaining 20% of the dataset is now used for validation.

Table 7 shows the MAEs for the training and testing datasets for each target variable, along with the difference between the training MAE and testing MAE. To establish the improvement of the proposed integrated machine learning model approach, standard deep MLP networks were used as baseline models which directly map the input boundary conditions to the output scalar fields (temperature, velocity, etc.). The standard MLP networks had three layers with 2048 neurons per hidden layer. Leaky-ReLu activation functions were used for the hidden layers. The results for these models can be seen in Table 7 under the heading "DNN (baseline)". The difference between the testing and training results are calculated using the equation shown below.

$$\%\Delta = \left(\frac{MAE_{train} - MAE_{test}}{MAE_{test}} \right) \times 100\% \tag{16}$$

In general, the VAE-20 – DNN-20 integrated networks have the lowest average training and validation MAEs across all target

Index	Architecture
1	2 x 1024
2	2 x 2048
3	2 x 4096
4	3 x 1024
5	3 x 2048
6	3 x 4096

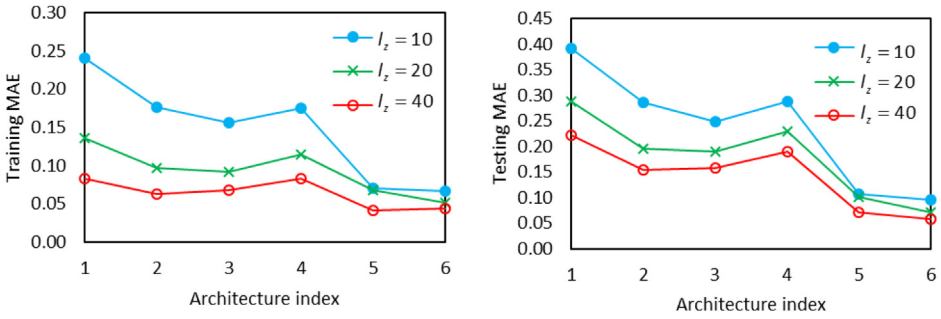

Fig. 10. Regression DNN model performance for increasing capacity.

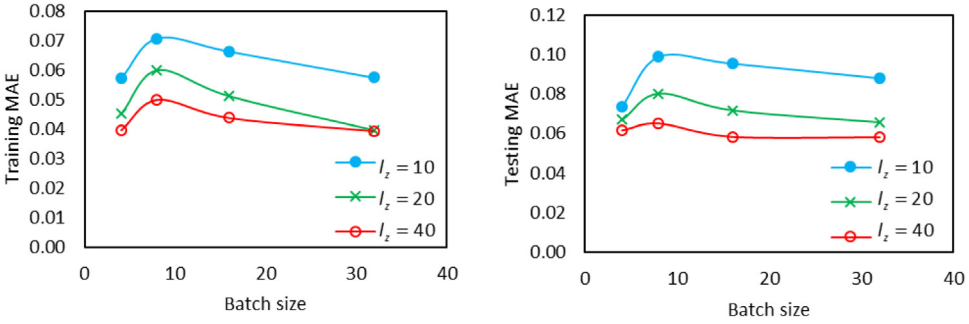

Fig. 11. Regression DNN model performance for varying mini-batch sizes.

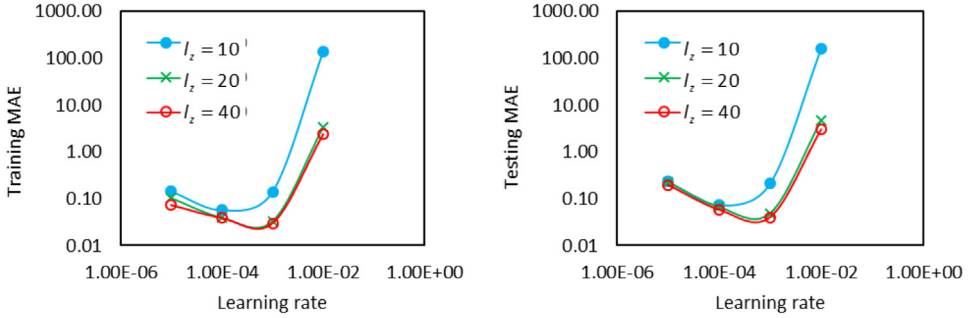

Fig. 12. Regression DNN model performance for varying learning rates.

Table 6
Summary of selected hyperparameters for regression DNN model and corresponding errors.

Parameter	Latent vector size model designation	$l_z = 10$ (DNN-10)	$l_z = 20$ (DNN-20)	$l_z = 40$ (DNN-40)
Selected hyperparameters				
Neurons per hidden layer		4096	4096	4096
Number of hidden layers (per encoder and decoder)		3	3	3
Mini-batch size		4	32	32
Learning rate		0.0001	0.001	0.001
Model losses				
Training MAE		0.0573	0.0331	0.0301
Testing MAE		0.0734	0.0487	0.0401

variables. The gas mixture constituent mass fraction MAEs are all below 0.3%, which is sufficiently low. The training MAPEs for the temperature and velocity networks are 0.43% and 1.8% respectively. The validation MAPEs for the temperature and velocity networks are 1.92% and 7.58% respectively. From the results in Table 7, one can see that the VAE-20 – DNN-20 model has on average a 52% lower training error compared to the baseline DNN model and a 24% lower average validation error.

It is noted that the differences between the training and testing errors, $\%\Delta$, are large >75%. This could be an indication of overfitting, which can be expected since no regularization method was applied during the training phase of the networks. To investigate this possibility, the DNN regression models were

Table 7
Training and testing MAEs for integrated network models.

Model designation:	DNN (baseline)		VAE-10 – DNN-10			VAE-20 – DNN-20			VAE-40 – DNN-40		
Field	Train MAE	Test MAE	Train MAE	Test MAE	%Δ	Train MAE	Test MAE	%Δ	Train MAE	Test MAE	%Δ
T [K]	8.6	20.6	4.03930	17.6360	77.10	3.3612	17.0990	80.34	3.5056	17.2907	79.73
V [m/s]	0.255	1.41	0.13630	1.0455	86.96	0.1371	0.9840	86.07	0.2654	1.0260	74.13
CO_2 [%wt]	0.04	0.16	0.02840	0.1279	77.79	0.0303	0.1283	76.41	0.0188	0.1212	84.46
CO [%wt]	0.028	0.19	0.03033	0.1684	81.99	0.0278	0.1599	82.60	0.0264	0.1609	83.60
H_2O [%wt]	0.014	0.19	0.02862	0.1575	81.82	0.0226	0.1536	85.28	0.0282	0.1618	82.59
H_2 [%wt]	0.0026	0.01	0.00134	0.0082	83.58	0.0017	0.0083	79.28	0.0014	0.0085	83.38
O_2 [%wt]	0.05	0.35	0.03872	0.2840	86.37	0.0501	0.2841	82.38	0.0432	0.2914	85.18
CH_4 [%wt]	0.018	0.11	0.01316	0.0983	86.61	0.0122	0.0948	87.13	0.0119	0.0968	87.75

retrained using drop-out regularization, with an active neuron probability set to 80% [31]. Furthermore, batch-normalization [28] was also applied at every layer except the output layer to counteract the effect of vanishing gradients. The results for the selected integrated network configurations with drop-out and batch-normalization are shown in Table 8.

The results in Table 8 show that using drop-out regularization and batch-normalization significantly reduced the difference between the training and testing errors. Although the training errors are now larger, the networks generalize much better to the testing data. The largest drop in %Δ is seen for the VAE-10 – DNN-10 networks. Nonetheless, the VAE-20 – DNN-20 networks still produce the smallest average training and testing errors. Therefore, it is selected as the best performing network configuration and will be used for further evaluations. For the VAE-20 – DNN-20 networks the training MAPEs for the temperature and velocity fields are 1.47% and 5.77% respectively, whereas the validation MAPEs are 1.7% and 7.1%.

The results obtained for the velocities are slightly worse than the targets initially set. The baseline model was also retrained with batch-normalization and dropout applied to the hidden layers. Similar to the integrated models, the generalization error between the training and validation errors of the baseline model reduced. Comparing the average errors between the baseline model and the VAE-20 – DNN-20 shows that the latter produces results with approximately 20% lower training errors and 10% lower validation errors.

These results show that the integrated approach results in lower prediction errors when compared to a standard DNN model approach. As mentioned in [15], since the number of output features is larger than the number of input features and training examples, a standard DNN model could lead to significant overfitting. Therefore, space encoding should be applied either through autoencoders or generative adversarial networks [9].

Additional insight into the selected networks' predictive performance is obtained by studying the training and testing MAE error distribution plots shown in Figs. 13 and 14. These error distributions are calculated on a cell-by-cell basis, which results in $0.8 \times N_{mesh} \times 1000 = 6868800$ entries per training target variable dataset and $0.2 \times N_{mesh} \times 1000 = 1717200$ entries in every testing target variable dataset. For the cell-by-cell temperature predictions, approximately 70% of the training and testing dataset entries have error values between 0–10 [K] when compared to the actual CFD cell values. The remaining 30% of training and testing temperature dataset entries have MAEs almost equally distributed between the indicated bin ranges. Fig. 13 also shows the MAPE temperature distribution with nearly 65% of entries in the training and validation datasets between 0–0.5% and about 10% of dataset entries between 3%–6%.

For the cell-by-cell velocity predictions, approximately 75% and 68% of training and validation dataset entries have errors between 0–0.5 [m/s], respectively. About 10% of the training and validation dataset entries have errors between 0.5–1 [m/s] and only a relatively small portion of the training (1.8%) and testing (3.5%) velocity entries have errors above 6 [m/s]. The velocity MAPE distributions show that approximately 45% of dataset entries have MAPEs between 0–0.5%. The distribution plots show an increasing trend in velocity MAPE frequency for higher MAPE ranges with approximately 15% of entries having MAPE errors between 6%–12%.

Fig. 14 shows that most of the entries in the mass fraction datasets (>90%), except for oxygen, have MAEs of between 0–0.5 [%wt]. For the oxygen dataset, 80% of the entries have MAEs between 0–0.5 [%wt] and approximately 10% of the entries have errors values between 0.5–1 [%wt]. This indicates that the species mass fraction prediction VAE-DNN networks are highly accurate with relatively low uncertainty on the actual predictions.

Figs. 15–18 show selected 2D contours of actual CFD solution data and network outputs for a single training and testing datapoint. For the training data point, the inlet conditions are: $V_{fuel} = 57.5$, $V_{pilot} = 11.82$, $Y_{fuel,CH4} = 0.143$, $Y_{fuel,O2} = 0.209$. For the testing data point, the inlet conditions are: $V_{fuel} = 43.8$, $V_{pilot} = 11.3$, $Y_{fuel,CH4} = 0.167$, $Y_{fuel,O2} = 0.179$. From Figs. 15–18, one can see that the data-driven model reproduces the qualitative velocity, temperature and species mass fraction distributions for the training and testing data points. The only noticeable difference observed is for the predicted velocity profiles in the near nozzle region.

Whether these profiles are reproduced with adequate accuracy will depend on the ultimate purpose of the reproduced profiles. For instance, if the purpose is to determine the radiation heat transfer emanating from the flue gas to surrounding surfaces, it may be different from determining the final outlet temperature profile of the gas.

Figs. 19 and 20 show line plots of the predicted and actual target variable values along the centreline of the domain for the specified training and testing data points. Fig. 19 shows that the surrogate model underpredicts the oxygen mass fractions in the region between the nozzle output and the combustion front, but otherwise the surrogate model predictions accurately replicate the CFD trends. For the validation data point, the surrogate model underpredicts the oxygen and methane in the same region as mentioned before.

To investigate the robustness of the proposed integrated modelling approach, the VAE-20 – DNN-20 was retrained using only 60% of the CFD dataset, and the remaining 40% used for validation. The results for the integrated model trained on a reduced training set can be seen in Table 9. Comparing the results below with those of the specified model in Table 8, one can see that the testing MAEs increase on average by approximately 6%, which is relatively low. This highlights the robustness of the model to accurately map and decode the underlying physics even with reduced dataset entries. This ability of the model can be ascribed

Table 8
Training and testing MAEs for integrated network models with batch-normalization and drop-out.

Model designation:	DNN (baseline)		VAE-10 – DNN-10			VAE-20 – DNN-20			VAE-40 – DNN-40		
Field	TrainMAE	TestMAE	Train MAE	Test MAE	%Δ	Train MAE	Test MAE	%Δ	Train MAE	Test MAE	%Δ
T [K]	15.1	17.0	15.636	16.465	5.03	13.0600	16.3370	20.06	14.2810	16.5540	13.73
V [m/s]	1.0	1.25	0.925	0.978	5.42	0.6700	0.9320	28.11	0.6750	0.9880	31.68
CO_2 [%wt]	0.108	0.129	0.1165	0.118884	2.00	0.0968	0.1214	20.26	0.0943	0.1131	16.61
CO [%wt]	0.14	0.16	0.14039	0.1531	8.30	0.1182	0.1524	22.44	0.1187	0.1540	22.96
H_2O [%wt]	0.133	0.15	0.13686	0.14923	8.29	0.1146	0.1453	21.15	0.1152	0.1492	22.78
H_2 [%wt]	0.0069	0.008	7.14E−03	7.84E−03	8.96	0.0060	0.0076	21.17	0.0059	0.0078	24.11
O_2 [%wt]	0.25	0.28	0.2623	0.2706	3.06	0.2138	0.2747	22.14	0.2137	0.2769	22.82
CH_4 [%wt]	0.08	0.1	0.09	0.09662	6.85	0.0729	0.0923	21.06	0.0727	0.0938	22.48

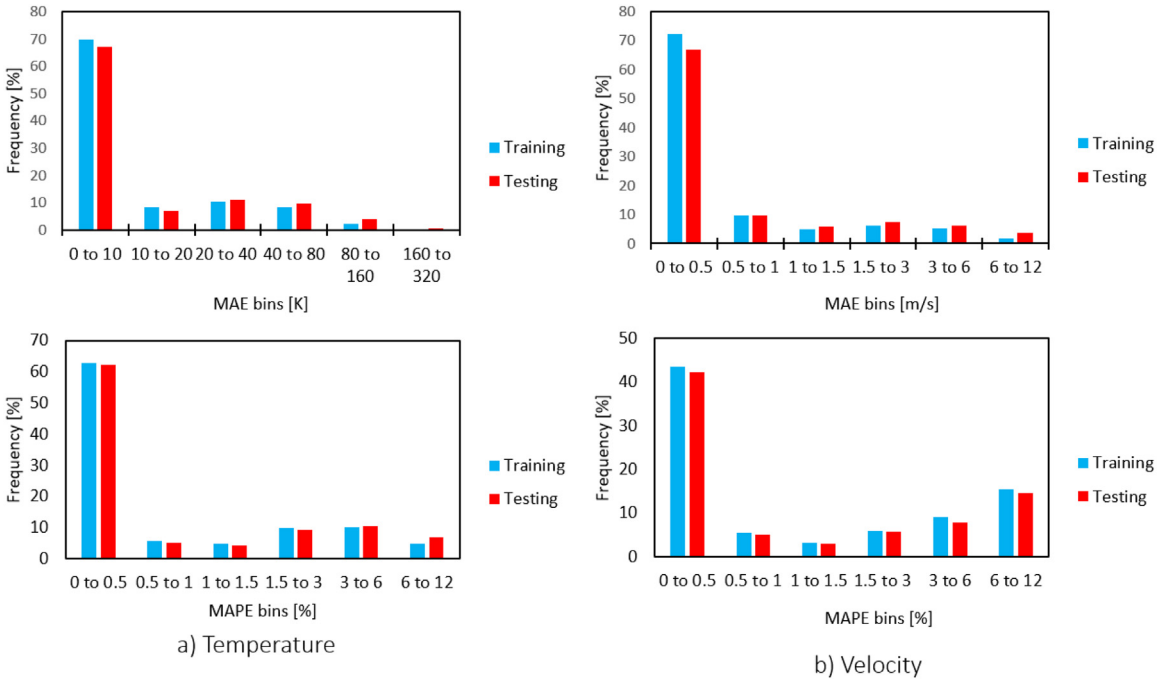

Fig. 13. Temperature and velocity MAE and MAPE error distributions.

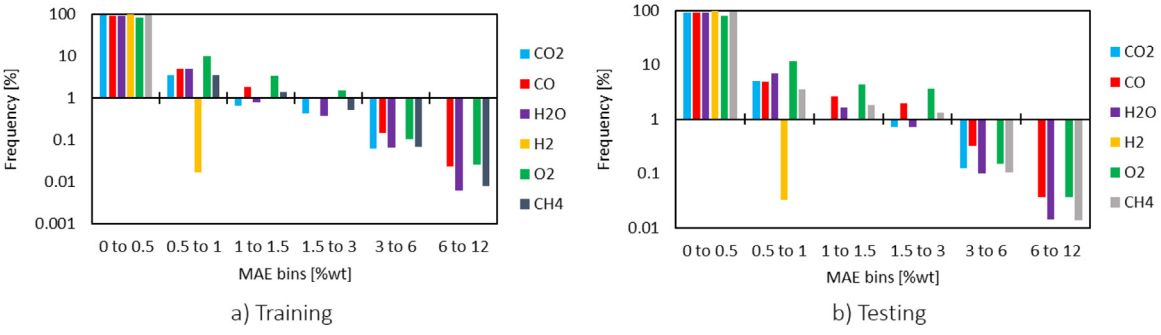

Fig. 14. Species mass fraction prediction error distributions for VAE-20 – DNN-20 model.

to the VAE, which during training learns meaningful representations across the entire features space, which results in the model always decoding into valid predictions (Section 3.2).

6. Conclusions

A novel integrated approach for data-driven surrogate modelling of combustion CFD simulations was presented that can predict various 2D fields such as temperature, velocity and species mass fractions from high level inputs such as velocity and fuel and air mass fractions. Training and validation data were generated via CFD using a 2D axisymmetric model of the Sandia Flame D experimental setup. CFD results were generated for 1000 different simulation cases, where fuel inlet and pilot inlet mass flow rates were varied along with the fuel stream methane and oxygen concentrations.

It was found that the best performing VAE-DNN configuration (VAE-20 – DNN-20) comprised of a VAE network layout with a latent vector size of 20, 3 hidden layers per encoder and decoder section and 296 neurons per hidden layer while using a learning

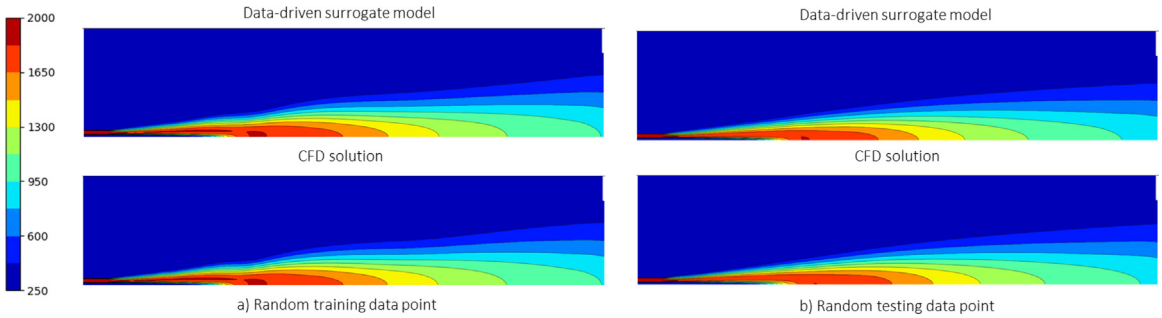

Fig. 15. Temperature [K] contour comparison between prediction and actual CFD data.

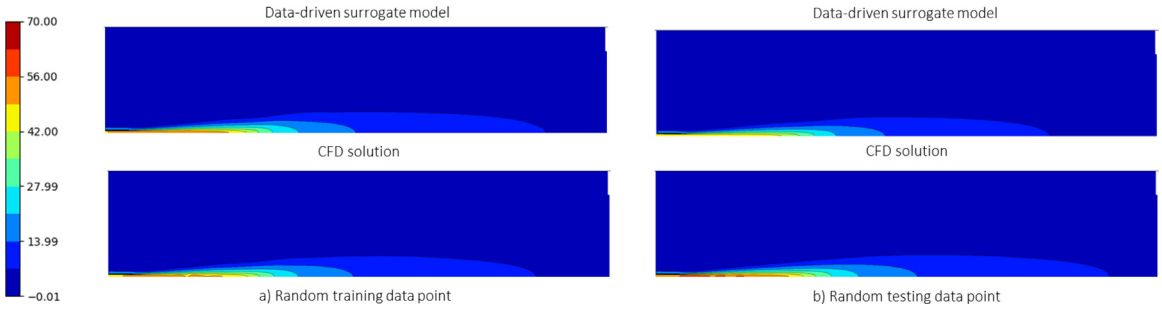

Fig. 16. Velocity [m/s] contour comparison between prediction and actual CFD data.

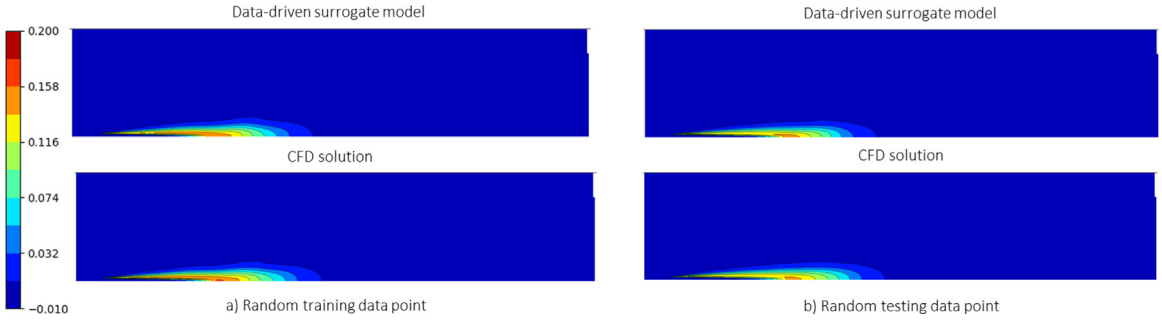

Fig. 17. CO [kg/kg] contour comparison between prediction and actual CFD data.

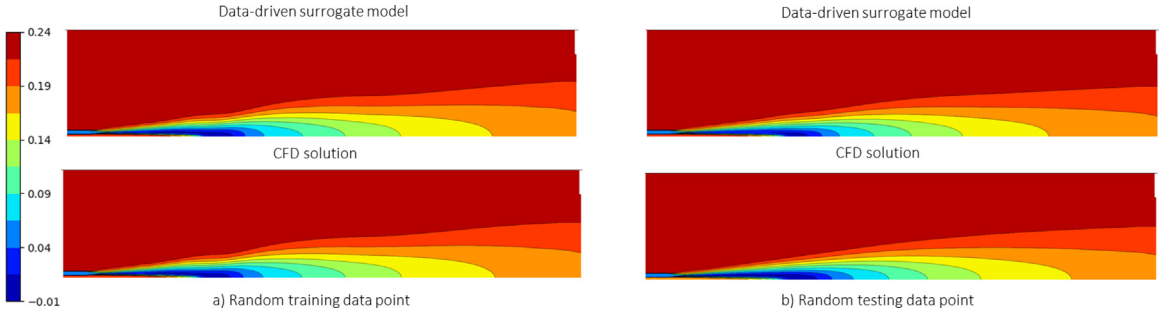

Fig. 18. O2 contour comparison between prediction and actual CFD data.

rate of 0.001 and a mini-batch size of 16 during training. The corresponding DNN network layout had 3 hidden layers and 4096 neurons per hidden layer and used a learning rate of 0.001 and mini-batch size of 32 during training.

The training and validation MAEs for the proposed layouts on temperature is 3.4 and 17.1 [K] respectively and for the velocity predictions 0.13 and 0.98 [m/s] respectively. All the species mass fraction training and validation prediction MAEs were below 0.3 [%wt]. It was noted that the differences between the training and testing error metrics were relatively large which was an indication of a measure of overfitting. The DNN-20 networks were then retrained using drop-out regularization and batch-normalization. It was seen that the training dataset MAEs increased to values closer to the validation MAEs and that the latter slightly decreased.

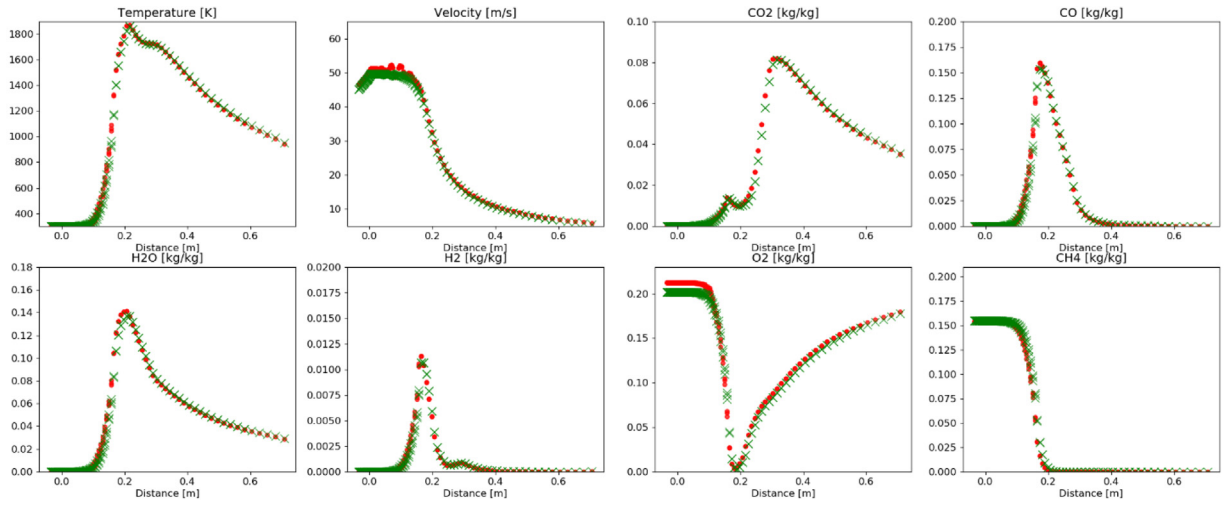

Fig. 19. Predicted and CFD values along centreline of domain for training data point (Red circles — CFD, Green cross — VAE-DNN).

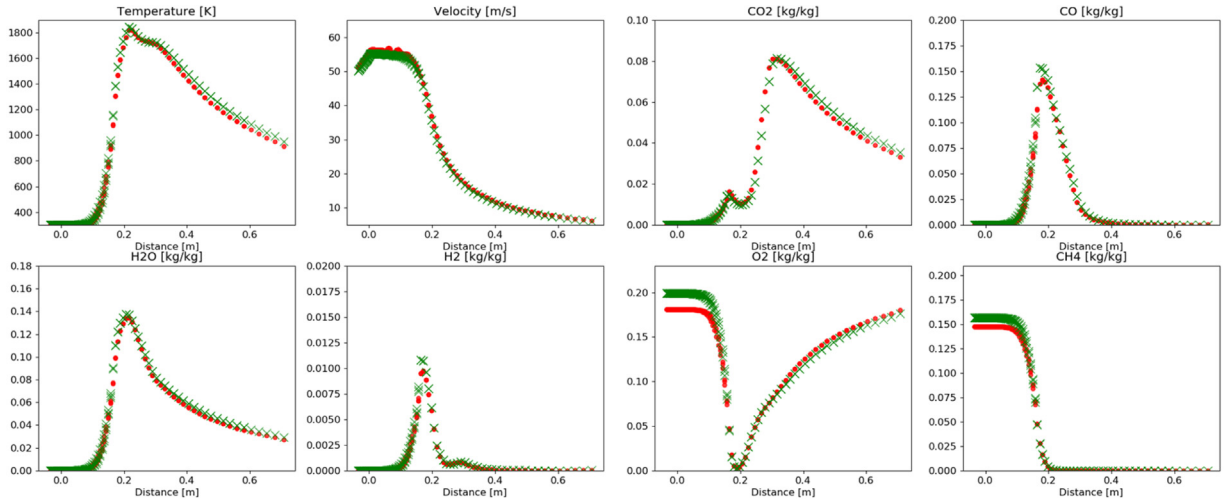

Fig. 20. Predicted and CFD values along the centreline of the domain for testing data point (Red — CFD, Green — VAE-DNN).

Table 9
Training and validation MAEs for VAE-20 – DNN-20 using reduced training dataset size.

Field	Train MAE	Test MAE
T [K]	13.87	17.09
V [m/s]	0.894	1.24
CO_2 [%wt]	0.099	0.126
CO [%wt]	0.125	0.16
H_2O [%wt]	0.121	0.15
H_2 [%wt]	0.00645	0.0079
O_2 [%wt]	0.233	0.285
CH_4 [%wt]	0.0749	0.0931

For the models using regularization and batch-normalization, the training and validation MAEs on temperature is 13.01 and 16.3 [K] respectively and for the velocity predictions 0.67 and 0.93 [m/s] respectively. The species mass fraction MAEs remained below 0.3 [%wt]. The integrated networks' accuracy goals set for the temperature and species mass fraction predictions were achieved. The networks developed to predict the velocity profiles had a training and testing MAPE above the desired value of 5%. Although the mean error values were above the desired limit, the velocity networks still showed an ability to qualitatively predict the velocity profiles.

The present work has shown that it is possible to predict 2D contours of various fluid values inside a simple jet flame combustion setup using CFD solution data and that the presented models have adequate generalizability and accuracy. The proposed model architecture and hyperparameter configurations can be used by other researchers as a starting point for the development of data-driven surrogate models of CFD combustion simulation data.

CRediT authorship contribution statement

Ryno Laubscher: conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing - original draft, Visualization. **Pieter Rousseau:** Writing - review & editing, Methodology, Resources, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

Funding

The authors would like to thank the Eskom EPPEI program for funding this project.

References

[1] S.M. Safdarnejad, J.F. Tuttle, K.M. Powell, Dynamic modeling and optimization of a coal-fired utility boiler to forecast and minimize NOx and CO emissions simultaneously, Comput. Chem. Eng. 124 (2019) 62–79.
[2] N. Modlinski, K. Szczepanek, D. Nabag, Mathematical procedure for predicting tube metal temperature in the second stage reheater of the operating fl exibly steam boiler, Appl. Therm. Eng. 146 (2018) (2019) 854–865, http://dx.doi.org/10.1016/j.applthermaleng.2018.10.063.
[3] Y. Li, G. Zhang, L. Wang, Y. Yang, Part-load performance analysis of a combined cycle with intermediate recuperated gas turbine, Energy Convers. Manag. 205 (July 2019) (2020) 112346, http://dx.doi.org/10.1016/j.enconman.2019.112346.
[4] Y. Shi, W. Zhong, X. Chen, A.B. Yu, J. Li, Combustion optimization of ultra supercritical boiler based on artificial intelligence, Energy 170 (2019) 804–817.
[5] Y. Cheng, Y. Huang, B. Pang, W. Zhang, Thermalnet: A deep reinforcement learning-based combustion optimization system for coal-fired boiler, Eng. Appl. Artif. Intell. 74 (2016) (2018) 303–311, http://dx.doi.org/10.1016/j.engappai.2018.07.003.
[6] H. Versteeg, W. Malalasekera, An Introduction to Computational Fluid Dynamics: The Finite Volume Method, first ed., Longman scientific & technical, Malaysia, 1995.
[7] M.J. Kochenderfer, T.A. Wheeler, Algorithms for Optimization, first ed., MIT Press, London, 2019.
[8] M. Eldred, D. Dunlavy, Formulations for surrogate-based optimization with data fit, multifidelity and reduced-order models, in: Proceedings of the 11th AIAA/ISSMO multidisciplinary analysis and optimization conference, 2006, p. 7117.
[9] H. Wu, X. Liu, W. An, S. Chen, H. Lyu, A deep learning approach for efficiently and accurately evaluating the flow field of supercritical airfoils, Comput. Fluids (2020) http://dx.doi.org/10.1016/j.compfluid.2019.104393.
[10] Y. Lv, F. Hong, T. Yang, F. Fang, J. Liu, A dynamic model for the bed temperature prediction of circulating fl uidized bed boilers based on least squares support vector machine with real operational data, Energy 124 (2017) 284–294, http://dx.doi.org/10.1016/j.energy.2017.02.031.
[11] E. Oko, M. Wang, J. Zhang, Neural network approach for predicting drum pressure and level in coal-fired subcritical power plant, Fuel 151 (2015) 139–145, http://dx.doi.org/10.1016/j.fuel.2015.01.091.
[12] R.A.D. Horwitz, A.G. Malan, J. Braithwaite, Aeroelastic Reduced Order Model: Kriging-Corrected Potential Flow, J. Aircr. 1–16 http://dx.doi.org/10.2514/1.C035366.
[13] R. Laubscher, Time-series forecasting of coal-fired power plant reheater metal temperatures using encoder–decoder recurrent neural networks, Energy 189 (2019) 116187, http://dx.doi.org/10.1016/j.energy.2019.116187.
[14] H. Zhou, J.P. Zhao, L.G. Zheng, C.L. Wang, K.F. Cen, Modeling NOx emissions from coal-fired utility boilers using support vector regression with ant colony optimization, Eng. Appl. Artif. Intell. 25 (1) (2012) 147–158, http://dx.doi.org/10.1016/j.engappai.2011.08.005.
[15] L. Liang, W. Mao, W. Sun, A feasibility study of deep learning for predicting hemodynamics of human thoracic aorta, 99, 2020.
[16] B. Wang, B. Xie, J. Xuan, K. Jiao, AI-based optimization of PEM fuel cell catalyst layers for maximum power density via data-driven surrogate modeling, Energy Convers. Manag. 205 (November 2019) (2020) 112460, http://dx.doi.org/10.1016/j.enconman.2019.112460.
[17] A. Warey, S. Kaushik, B. Khalighi, M. Cruse, G. Venkatesan, Data-driven prediction of vehicle cabin thermal comfort: using machine learning and high-fidelity simulation results, Int. J. Heat Mass Transfer (2020) http://dx.doi.org/10.1016/j.ijheatmasstransfer.2019.119083.
[18] Z. Ti, X.W. Deng, H. Yang, Wake modeling of wind turbines using machine learning, Appl. Energy (2020) http://dx.doi.org/10.1016/j.apenergy.2019.114025.
[19] K. Deng, H. Chen, Y. Zhang, Flow structure oriented optimization aided by deep neural network, in: 10th International Conference on Computational Fluid Dynamics, 2018, pp. 1–12.
[20] R. Barlow, J. Frank, Piloted CH4/Air flames C, D, E and F, Livermore, 2007.
[21] R. Laubscher, P. Rousseau, Numerical investigation into the effect of burner swirl direction on furnace and superheater heat absorption for a 620 MWe opposing wall-fired pulverized coal boiler, Int. J. Heat Mass Transfer (2019) 506–522, http://dx.doi.org/10.1016/j.ijheatmasstransfer.2019.03.150.
[22] ANSYS, ANSYS fluent theory guide, 2018.
[23] R. Laubscher, J.H. Hoffmann, Utilization of basic multi-layer perceptron artificial neural networks to resolve turbulent fine structure chemical kinetics applied to a CFD model of a methane/air piloted jet flame, J. Therm. Eng. 4 (2) (2018) 1828–1846.
[24] L. Wang, Z. Liu, S. Chen, C. Zheng, Comparison of different global combustion mechanisms under hot and diluted oxidation conditions, Combust. Sci. Technol. 184 (2) (2012) 259–276, http://dx.doi.org/10.1080/00102202.2011.635612.
[25] A. Shiehnejadhesar, R. Mehrabian, R. Scharler, G. Goldin, I. Obernberger, Development of a gas phase combustion model suitable for low and high turbulence conditions, Fuel 126 (2014) 177–187.
[26] G. Sankar, D.S. Kumar, K.R. Balasubramanian, Computational modeling of pulverized coal fired boilers – A review on the current position, Fuel 236 (2018) (2019) 643–665, http://dx.doi.org/10.1016/j.fuel.2018.08.154.
[27] W. McCulloch, W. Pitts, A logical calculus of the ideas immanent in nervous activity, Bull. Math. Biophys. 5 (1943) 115–133.
[28] A. Geron, Hands-on Machine Learning with Scikit-Learn & Tensorflow, first ed., O'Reilly, Sebastapol, 2017.
[29] B. Xu, N. Wang, T. Chen, Emirical evaluation of rectified activations in convulution network, 2015, arXiv:1505.00853.
[30] D. Rumelhart, G. Hinton, R. Willimans, Learning representations by back-propagating errors, Nature 323 (1986) 533–536.
[31] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, first ed., MIT Press, Chennai, 2017.
[32] F. Chollet, Deep Learning with Python, first ed., Manning Shelter Island, New York, 2018.
[33] D.P. Kingma, M. Welling, Auto-encoding variational bayes, in: 2nd Int. Conf. Learn. Represent. ICLR 2014 - Conf. Track Proc. No. Ml, 2014, pp. 1–14.
[34] D.J. Rezende, S. Mohamed, D. Wierstra, Stochastic backpropagation and approximate inference in deep generative models, in: 31st Int. Conf. Mach. Learn. ICML 2014, Vol. 4, 2014, pp. 3057–3070.
[35] X. Hou, K. Sun, L. Shen, G. Qiu, Improving variational autoencoder with deep feature consistent and generative adversarial training, Neurocomputing 341 (2019) 183–194, http://dx.doi.org/10.1016/j.neucom.2019.03.013.
[36] B. Dai, Y. Wang, J. Aston, D. Wipf, Hidden talents of the variational autoencoder, 1–45.
[37] S. Kullback, R. Leibler, On information and sufficiency, Ann. Math. Stat. 1 (1951) 79–86.
[38] I. Pointer, Programming PyTorch for Deep Learning, first ed., O'Reilly, Sabastepol, 2019.
[39] V.K. Ojha, A. Abraham, V. Snášel, Metaheuristic design of feedforward neural networks: A review of two decades of research, Eng. Appl. Artif. Intell. 60 (February) (2017) 97–116, http://dx.doi.org/10.1016/j.engappai.2017.01.013.
[40] D.P. Kingma, J.L. Ba, Adam: A method for stochastic optimization, 2015, pp. 1–15, arXiv Prepr..