Machine learning for flux regression in discrete fracture networks

GEM - International Journal on Geomathematics (2021) 12:9
https://doi.org/10.1007/s13137-021-00176-0

ORIGINAL PAPER

Machine learning for flux regression in discrete fracture networks

S. Berrone[1,3,4] · F. Della Santa[1,3,4] · S. Pieraccini[2,3] · F. Vaccarino[1,4,5]

Received: 15 July 2020 / Accepted: 15 April 2021

Abstract
In several applications concerning underground flow simulations in fractured media, the fractured rock matrix is modeled by means of the Discrete Fracture Network (DFN) model. The fractures are typically described through stochastic parameters sampled from known distributions. In this framework, it is worth considering the application of suitable complexity reduction techniques, also in view of possible uncertainty quantification analyses or other applications requiring a fast approximation of the flow through the network. Herein, we propose the application of Neural Networks to flux regression problems in a DFN characterized by stochastic trasmissivities as an approach to predict fluxes.

Keywords Discrete fracture network flow simulations · Deep learning · Uncertainty quantification

Mathematics Subject Classification 65N30 · 68T07 · 68T37 · 76-10 · 86A99 · 76U60

1 Introduction

Characterization of flow and transport in subsurface fractured media is a crucial issue in several critical and up-to-date applications concerning civil, environmental and industrial engineering: geothermal applications, enhanced oil and gas production, aquifer

✉ S. Pieraccini
sandra.pieraccini@polito.it

1 Dipartimento di Scienze Matematiche, Politecnico di Torino, Turin, Italy

2 Dipartimento di Ingegneria Meccanica e Aerospaziale, Politecnico di Torino, Turin, Italy

3 INdAM-GNCS group, Rome, Italy

4 SmartData@PoliTO, Politecnico di Torino, Turin, Italy

5 ISI Foundation, Turin, Italy

Published online: 25 April 2021

monitoring, safety assessment of CO_2 or nuclear waste geological storage, just to mention a few.

A possible approach for modelling fractured media is given by the Discrete Fracture Network (DFN) model (Adler 1999; Cammarata et al. 2007; Fidelibus et al. 2009): fractures in the rock matrix are individually described and represented as planar polygons intersecting each other along segments called *traces*; flux exchanges occur among fractures through the traces, whereas the surrounding rock matrix is assumed herein to be impervious. On each fracture the Darcy law is assumed to rule the flux; at all fracture intersections head continuity and flux balance are assumed. The present assumptions are correct for fractures that are porous media with a permeability much larger than the permeability of the surrounding rock matrix.

Flow simulations in DFNs are likely to be quite challenging problems, due to several issues: the huge size of realistic networks; the geometrical complexity of the computational domain, in which traces can intersect forming very narrow angles, or can be extremely close to each other. These features make the meshing process quite an hard task, whenever conforming meshes are needed, and in recent literature several new methods have been proposed which use different strategies for circumventing, either partially or totally, meshing problems. In Pichot et al. (2010, 2012, 2014) and de Dreuzy et al. (2013) the mortar method is used, possibly in conjunction with modifications of the geometry. In Nœtinger and Jarrige (2012), Nœtinger (2015) and Dershowitz and Fidelibus (1999) lower-dimensional problems are introduced in order to reduce the complexity: fracture connections are represented as a system of pipes modeling the flux exchange between fractures. In Berrone et al. (2013a, b, 2014, 2015, 2016b, a, 2019) the problem is reformulated as a PDE-constrained optimization, in such a way that totally non-conforming meshes are allowed, and the meshing process is therefore quite an easy task which can be independently performed on each fracture. In Fumagalli and Scotti (2013) a 2D coupling between fractures, represented as segments, and rock matrix, represented as a 2D domain, is considered. In Hyman et al. (2014) an approach with conforming finite elements is used. In Jaffré and Roberts (2012) a mixed finite elements approach is proposed, and in Karimi-Fard and Durlofsky (2014) a local adaptive upscaling method is proposed.

A deterministic knowledge of hydrogeological and geometrical parameters describing fracture position, size, orientation, aperture, etc., is typically not available; fractures in a network are usually described by sampling their geometrical and hydrogeological features from given distributions. Due to the large amount of uncertainty in the representation of DFNs, flow and transport in a real fractured medium are studied from a statistical point of view, resorting to Uncertainty Quantification (UQ) analyses. These analyses are likely to involve thousands of DFN simulations; in order to speed up these analyses, it is worth considering some sort of complexity reduction techniques (see, e.g., Canuto et al. 2019 for a multi fidelity approach, and Hyman et al. (2017) for a graph-based reduction technique).

The use of machine learning (ML) has recently attracted a lot of attention in several frameworks related to the aforementioned problems. In recent contributions (Srinivasan et al. 2018, 2019), ML techniques have been applied to DFN flow simulations in conjunction with graph-based methods. Neural Networks (NNs) have been applied in a UQ framework in Chan and Elsheikh (2018) and Tripathy and Bilionis (2018).

In Chan and Elsheikh (2018), a data-driven approach is introduced for the estimation of coarse scale basis functions within a Multiscale Finite Volume method: using a NN trained on a set of solution samples, the authors obtain a generator of subsequent basis functions at a low computational cost. In Tripathy and Bilionis (2018), in the framework of a stochastic elliptic PDE with uncertain diffusion coefficient, inspired by dimensionality reduction techniques, the authors construct a NN as surrogate model to replace the forward model solver, while performing Monte Carlo (MC) simulations for UQ. Finally, it is worth mentioning the approach in Raissi et al. (2019), in which deep learning is used both for building data-driven approximations of solutions of PDEs, and for proposing a discovery tool for incomplete models.

In this paper, we consider the application of Multi-Task Neural Networks for flux predictions in DFNs. NNs are a particular kind of machine learning algorithms based on regression; they were first introduced more than fifty years ago (see, e.g., McCulloch and Pitts 1943; Hebb 1949; Rosenblatt 1958) but only in the last decade they have started to be used in practice, due to computer hardware improvements and the increasing amount of available data (see Goodfellow et al. 2016 and references therein). In the problem addressed, we will consider a deterministic geometry of the fractures, whereas hydrogeological parameters (namely, the fracture transmissivities) will be modeled as random variables with a known distribution. Vector valued regression will be performed, through NNs, considering as inputs only the uncertain parameters and as output the flux exiting the network through some selected boundary fractures. The outcome of the NN is a function which for a given input vector (namely, for a given transmissivity value for each fracture) provides a prediction of the corresponding outgoing fluxes.

We will consider several cases, in which the number of fractures with a stochastic transmissivity is progressively increasing, starting from very few fractures and ending up with 100% of the fractures. We will discuss, in particular, the behavior of the regression quality with respect to the number of stochastic fractures and to NNs regularization and architecture, showing that NNs can be effective tools for predicting the flux values in this framework.

The paper is organized as follow. In Sect. 2 the model description, and a sketch of the numerical method used for the simulations, are presented. In Sect. 3 the main ideas and concepts behind NN are briefly recalled. A deep analysis about the application of NN to DFN flow simulation problems is presented in Sect. 4, and in Sect. 5 a sketch of the application of NNs in support of UQ analysis is given. Section 6 is devoted to show the robustness of the approach with respect to network size. Finally, some conclusions are drawn in Sect. 7.

2 Discrete fracture network

In this section we briefly recall the DFN model formulation and we sketch the numerical strategy used for the flow simulations, while referring the reader to Berrone et al. (2013a, 2014) for full details.

Let $\mathcal{F}_i$ denote the ith fracture of the network, with $i \in \mathcal{I}$, represented as a planar polygon. Fractures may intersect each other along lines called traces. Let S_m denote

the mth trace, with $m \in \mathcal{M}$. For the sake of simplicity, we assume that each trace is generated by exactly two fractures. The approach can be extended to traces generated by more than two fractures.

For each couple of intersecting fractures, say $\mathcal{F}_i$ and $\mathcal{F}_j$, generating a trace S_m, let us introduce $I_m = (i, j)$, where the couple (i, j) is ordered with the convention that $i < j$. Denoting by Ω the whole DFN, we have therefore

$$\Omega = \cup_{i \in \mathcal{I}} \mathcal{F}_i.$$

Let $\kappa_i(x_i)$ denote, for all $i \in \mathcal{I}$, a symmetric and uniformly positive definite tensor representing the fracture transmissivity, where x_i refers to a local coordinate system on $\mathcal{F}_i$. The main unknown of the problem is the hydraulic head H_i on each fracture.

Let us divide each fracture boundary $\partial\mathcal{F}_i$ in a part Γ_i^D, on which a Dirichlet boundary condition $H_i = H_i^D$ is imposed, and in a part Γ_i^N, such that $\partial\mathcal{F}_i = \Gamma_i^D \cup \Gamma_i^N$, $\Gamma_i^D \cap \Gamma_i^N = \emptyset$, on which a Neumann boundary condition $\kappa_i \frac{\partial H_i}{\partial n} = H_i^N$ is imposed, where n is an outward unit normal vector. The quantity $\frac{\partial H_i}{\partial \nu} := \kappa_i \frac{\partial H_i}{\partial n}$ is usually called the *co-normal derivative* of the hydraulic head along the unit vector n, and it represents the flux entering $\mathcal{F}_i$ through Γ_i^N.

Let us introduce the following sets:

$$\partial\Omega = \cup_{i \in \mathcal{I}} \partial\mathcal{F}_i, \quad \Gamma^D = \cup_{i \in \mathcal{I}} \Gamma_i^D, \quad \Gamma^N = \cup_{i \in \mathcal{I}} \Gamma_i^N.$$

We remark that Γ^D is required to be non-empty, whereas some of the sets Γ_i^D are allowed to be empty, see Berrone et al. (2014).

Let us collect the traces in the following sets: let

$$\mathcal{S} = \cup_{m \in \mathcal{M}} S_m$$

be the set of all traces in the network, and $\mathcal{S}_i \subset \mathcal{S}$, for $i \in \mathcal{I}$, the subset of traces belonging to $\mathcal{F}_i$.

As usual, let us denote $\mathrm{H}^1(\mathcal{F}_i)$ the Sobolev space

$$\mathrm{H}^1(\mathcal{F}_i) = \{v \in \mathrm{L}^2(\mathcal{F}_i) \,:\, \nabla v \in [\mathrm{L}^2(\mathcal{F}_i)]^2\}$$

and let us introduce $\forall i \in \mathcal{I}$ the spaces

$$V_i = \mathrm{H}^1_0(\mathcal{F}_i) = \left\{ v \in \mathrm{H}^1(\mathcal{F}_i) : v_{|_{\Gamma_i^D}} = 0 \right\}$$

with dual V_i', and

$$V_i^D = \mathrm{H}^1_{\mathrm{D}}(\mathcal{F}_i) = \left\{ v \in \mathrm{H}^1(\mathcal{F}_i) : v_{|_{\Gamma_i^D}} = H_i^D \right\}.$$

If Γ_i^D is empty, $V_i = \mathrm{H}^1(\mathcal{F}_i)$.

The hydraulic head H_i on each fracture is solution to the following problem: find $H_i \in V_i^D$ such that $\forall v \in V_i$

$$\int_{\mathcal{F}_i} \kappa_i \nabla H_i \nabla v \mathrm{d}\Omega = \int_{\mathcal{F}_i} q_i v \mathrm{d}\Omega + \int_{\Gamma_i^N} H_i^N v_{|\Gamma_i^N} \mathrm{d}\Gamma + \sum_{S \in \mathcal{S}_i} \int_S \left[\!\left[\frac{\partial H_i}{\partial \nu_S^i} \right]\!\right]_S v_{|S} \mathrm{d}\Gamma, \quad (1)$$

where $q_i \in \mathrm{L}^2(\mathcal{F}_i)$ is a source term on $\mathcal{F}_i$; H_i^N is the Neumann boundary condition imposed on Γ_i^N; $\frac{\partial H_i}{\partial \nu_S^i} = (n_S^i)^T \kappa_i \nabla H_i$ is the outward co-normal derivative of the hydraulic head along a fixed unit vector n_S^i normal to S, and $\left[\!\left[\frac{\partial H_i}{\partial \nu_S^i} \right]\!\right]_S$ denotes its jump along n_S^i. We remark that the last term in (1) represents the net flow entering/exiting the fracture through its traces. Continuity of the hydraulic head and flux balance along the traces are ensured imposing suitable matching conditions at the traces: $\forall m \in \mathcal{M}$, with $i, j \in I_m$,

$$H_{i\,|S_m} - H_{j\,|S_m} = 0, \quad (2)$$

$$\left[\!\left[\frac{\partial H_i}{\partial \nu_{S_m}^i} \right]\!\right]_{S_m} + \left[\!\left[\frac{\partial H_j}{\partial \nu_{S_m}^j} \right]\!\right]_{S_m} = 0. \quad (3)$$

A PDE-constrained optimization reformulation of problems (1)–(3) was proposed in Berrone et al. (2013a, b) and further developed in Berrone et al. (2014, 2016a, 2019). Within such reformulation, exact fullfilment of Eqs. (2) and (3) is replaced by the minimization of a suitable functional measuring flux unbalance and continuity mismatch at traces. Let us introduce the quantities

$$U_i^{S_m} := \left[\!\left[\frac{\partial H_i}{\partial \nu_{S_m}^i} \right]\!\right]_{S_m} + \alpha H_{i\,|S_m}, \qquad U_j^{S_m} := \left[\!\left[\frac{\partial H_j}{\partial \nu_{S_m}^j} \right]\!\right]_{S_m} + \alpha H_{j\,|S_m}, \quad (4)$$

for each $m \in \mathcal{M}$, with $I_m = (i, j)$, where $\alpha > 0$ a fixed parameter. Collecting the functions $U_i^{S_m}$ in the tuples

$$U_i = \prod_{S_m \in \mathcal{S}_i} U_i^{S_m} \in \mathcal{U}_i := \prod_{S_m \in \mathcal{S}_i} \mathrm{H}^{-\frac{1}{2}}(S_m), \qquad U = \prod_{i \in \mathcal{I}} U_i \in \mathcal{U} := \prod_{i \in \mathcal{I}} \mathcal{U}_i,$$

we introduce the functional

$$J(H, U) = \sum_{m \in \mathcal{M}} \Big(||H_{i|_{S_m}} - H_{j|_{S_m}}||^2_{\mathrm{H}^{\frac{1}{2}}(S_m)} + ||U_i^{S_m} + U_j^{S_m} - \alpha \left(H_{i|_{S_m}} + H_{j|_{S_m}} \right)||^2_{\mathrm{H}^{-\frac{1}{2}}(S_m)} \Big), \quad (5)$$

with $I_m = (i, j)$. Taking into account (4), we rewrite Eq. (1) as

$$\int_{\mathcal{F}_i} \kappa_i \nabla H_i \nabla v \mathrm{d}\Omega + \alpha \sum_{m \in \mathcal{M}} \int_{S_m} H_{i|_{S_m}} v_{|_{S_m}} \mathrm{d}\Gamma = \int_{\mathcal{F}_i} q_i v \mathrm{d}\Omega + \int_{\Gamma_i^N} H_i^N v_{|\Gamma_i^N} \mathrm{d}\Gamma + \sum_{m \in \mathcal{M}} \int_{S_m} U_i^{S_m} v_{|_{S_m}} \mathrm{d}\Gamma \tag{6}$$

$\forall v \in V_i, \forall i \in \mathcal{I}$. Then, Eqs. (1), (2) and (3) are equivalent to the problem

$$\begin{aligned} &\min J(H, U) \\ &\text{subject to (6).} \end{aligned} \tag{7}$$

By introducing suitable space discretizations on the fractures and on the traces (the former possibly based on non-conforming meshes), problem (7) is reformulated as a finite dimensional quadratic programming problem which can be solved via the conjugate gradient method (see Berrone et al. 2015). This approach has been used in conjunction with standard FEM, with XFEM for enriching the solution along traces (Berrone et al. 2017), and with VEM (Benedetto et al. 2014). In all cases the meshing process is independent on each fracture. An efficient parallel implementation has been produced and tested (Berrone et al. 2019, 2015). The method has also been effectively used in massive computations for uncertainty quantification analyses in a geometric Multi Level Monte Carlo framework (Berrone et al. 2018), taking advantage of the possibility of using very coarse meshes along all the network, even in presence of critical geometrical configurations.

Notwithstanding the high deal of flexibility and efficiency of the method, simulations on large scale realistic DFNs may be very costly; if uncertainty quantification analyses have to be performed, a large number of simulations comes into play, and this may be a prohibitive task. The approach proposed herein uses NNs as a model reduction tool, which may be used to speed up, e.g., uncertainty quantification computations.

In the next section we will recall the main ideas behind NNs and sketch their application to the DFN flow simulation framework; in particular, a case with deterministic geometry and random transmissivity will be addressed. The transmissivity tensor is assumed herein to be homogeneous and isotropic, so that it can be represented, on each fracture, as a scalar value.

The quite challenging case of stochastic geometry will be deferred to future work.

3 Vector valued regression methods for flux prediction

In this section we describe NNs used to address the problem of flux prediction in DFNs. We adopt a well assessed methodology for constructing NNs, which we summarize here for the reader's convenience. After a brief introduction to the mathematical background of NNs, we will describe the architecture used for tackling the problem,

sketching the corresponding algorithms, and the performance measures. In particular, for the following description, we closely follow (Goodfellow et al. 2016).

3.1 Neural networks

NN are learning algorithms described through oriented weighted graphs $N = (U, A)$, where U is the set of nodes and $A \subset U \times U$ is the set of edges, represented as ordered pairs of nodes; each edge $a_{ij} = (u_i, u_j)$ is endowed with a weight $w_{ij} \in \mathbb{R}$. Each node $u \in U$ corresponds to a *unit* (also called *neuron*): it performs some selected operations on the signals received and it sends its output to other units of the network. Among all the nodes, some *input* and *output* units are present.

The behavior of a generic NN unit can be described as follows (see Goodfellow et al. 2016, chapter 6.3). For each $u_j \in U$ let $\mathrm{Ent}(u_j) \subset U$ be the subset of nodes connected to u_j with an edge entering in it, namely

$$\mathrm{Ent}(u_j) = \{u_i \in U \mid \exists\, a_{ij} = (u_i, u_j) \in A\}\,.$$

For each $u_i \in \mathrm{Ent}(u_j)$, let $y_i \in \mathbb{R}$ be the output of unit u_i. Let $I_j = \sum_{u_i \in \mathrm{Ent}(u_j)} y_i w_{ij}$ be the input of u_j, and let $f_j : \mathbb{R} \to \mathbb{R}$ be a function (usually nonlinear, see Goodfellow et al. 2016, chapter 6.3), called *activation function*, associated to unit u_j. Then the output $y_j \in \mathbb{R}$ of unit u_j is defined as

$$y_j = f_j\left(I_j\right). \tag{8}$$

Any NN with n input units and m output units can be written as a function $\widehat{F}(\,\cdot\,;\boldsymbol{w}) : \mathbb{R}^n \to \mathbb{R}^m$ given by the composition of all activation functions f_j and all linear functions I_j, characterized by weights w_{ij}, here collected in a vector $\boldsymbol{w}$; in practice, for each $\boldsymbol{x} \in \mathbb{R}^n$ and for a given vector of weights $\boldsymbol{w}$, $\widehat{F}(\boldsymbol{x};\boldsymbol{w})$ returns a vector $\hat{\boldsymbol{z}}^{(\boldsymbol{w})}$, interpreted as the predicted output corresponding to the input $\boldsymbol{x}$:

$$\hat{\boldsymbol{z}}^{(\boldsymbol{w})} = \widehat{F}(\boldsymbol{x};\boldsymbol{w})\,. \tag{9}$$

In the present work, we only consider supervised NNs, therefore we focus on the description of their training principles.

3.1.1 Learning phase in neural networks

Given a function $F : \mathbb{R}^n \to \mathbb{R}^m$, the main target of supervised NNs (and supervised ML algorithms in general) is to build a function $\widehat{F}$ approximating F using a set $\mathcal{D}$ of pairs $(\boldsymbol{x}_k, \boldsymbol{z}_k)$, $k = 1, \ldots, D$, such that $\boldsymbol{z}_k = F(\boldsymbol{x}_k)$:

$$\mathcal{D} = \left\{(\boldsymbol{x}_k, \boldsymbol{z}_k) \in \mathbb{R}^n \times \mathbb{R}^m \mid \boldsymbol{z}_k = F(\boldsymbol{x}_k) \text{ for each } k = 1, \ldots, D\right\}, \tag{10}$$

where D is the cardinality of $\mathcal{D}$ ($D = |\mathcal{D}|$). The learning or training phase of supervised NNs corresponds to the computation of the weights $\boldsymbol{w}$ in order to minimize a given

error function. For example, one may consider the problem

$$\min_{\boldsymbol{w}} \| \widehat{F}(\boldsymbol{x}; \boldsymbol{w}) - F(\boldsymbol{x}) \|^2, \tag{11}$$

for each $\boldsymbol{x} \in \mathbb{R}^n$, where $\| \cdot \|$ denotes the ℓ^2 (i.e., Euclidean) vector norm. The selection of optimal weights $\boldsymbol{w}^*$ solving (11) is obtained as follows. Let $\mathcal{T} \subseteq \mathcal{D}$ be a so called training set. Without loss of generality, we assume that $\mathcal{T}$ is made of the first $T = |\mathcal{T}|$ elements of $\mathcal{D}$; then, the weights $\boldsymbol{w}^*$ are typically obtained using tailored versions of the gradient method (thanks to backpropagation algorithm Goodfellow et al. 2016, chapter 6.5) applied to problem

$$\min_{\boldsymbol{w}} \mathcal{E}_{\mathcal{T}}(\boldsymbol{w}) \tag{12}$$

where $\mathcal{E}_{\mathcal{T}}(\boldsymbol{w}) = T^{-1} \sum_{t=1}^{T} E_t(\boldsymbol{w})$ and $E_t(\boldsymbol{w}) = \| \hat{\boldsymbol{z}}_t^{(\boldsymbol{w})} - \boldsymbol{z}_t \|^2$, for $t = 1, \ldots, T$.

Usually, a maximum number c_{max} of iterations of the gradient method is fixed; for this reason, each single step is considered as an *epoch* of the training phase. In order to improve the learning phase of the NNs, alternative ways of training, the so-called *mini-batch* methods (Goodfellow et al. 2016, chapter 8.5), have been introduced; these methods consider training epochs as composed from more than one step of the gradient method, each one with a gradient computed with respect to a random subset $\mathcal{B} \subset \mathcal{T}$ instead of $\mathcal{T}$ itself.

3.1.2 Regularization methods

Since in supervised NNs the target is to find the best weights $\boldsymbol{w}^*$ that approximate those that minimize (11), an exact solution $\boldsymbol{w}^*_{\mathcal{T}}$ of problem (12) (or a too good approximation of it) actually is not always needed. Indeed, weights too close to $\boldsymbol{w}^*_{\mathcal{T}}$ typically lead to overfitting phenomena: very good approximation of F in training points, but poor everywhere else; on the other hand, a bad approximation of $\boldsymbol{w}^*_{\mathcal{T}}$ leads to underfitting phenomena, where the approximation of F has an extremely low accuracy for all $\boldsymbol{x} \in \mathbb{R}^n$.

To avoid both underfitting and overfitting phenomena in ML algorithms, regularization methods are introduced. Let $\mathcal{P} = \mathcal{D} \backslash \mathcal{T}$ be the test set where performance of the NN is evaluated, computing error $\mathcal{E}_{\mathcal{P}}$; in NNs the regularization methods mainly consist in favoring modifications of standard training methods, looking for weights, among those computed in the training phase, that will minimize the test error. To this aim, a validation set $\mathcal{V} \subset \mathcal{T}$, a sort of test set inside the training set, is considered and used to predict test error behavior during learning phase. Then, the NN is trained on $\mathcal{T} \backslash \mathcal{V}$, while $\mathcal{E}_{\mathcal{V}}$ is monitored; weights corresponding to low validation errors $\mathcal{E}_{\mathcal{V}}$ are likely to also correspond to low test error values and to provide a NN with good generalization skills. Following the nomenclature commonly used when validation sets come into play, from now on we redefine the training set as $\mathcal{T} = \mathcal{D} \backslash \mathcal{P} \backslash \mathcal{V}$.

A very simple regularization method (used not only for NNs but also for more general supervised ML algorithms) is the early stopping method (Goodfellow et al. 2016, chapter 7.8): the validation error $\mathcal{E}_{\mathcal{V}}$ is monitored at each epoch, and the training

phase is interrupted if the error increases for a certain number of epochs. Indeed, since test error and validation error are expected to behave similarly, stopping the training when $\mathcal{E}_{\mathcal{V}}$ is low should guarantee better performances of the model.

Early stopping regularization can be summarized as follows. Let $p^* \in \mathbb{N}$, $p^* > 0$, be a parameter (called *patience* parameter); the training is stopped if the validation error increases for p^* consecutive epochs. An additional advantage of early stopping regularization is that it speeds up the training, since many unnecessary training cycles are not computed.

Another simple regularization method, which again can be used in conjunction with any supervised ML algorithm, is the "minimum validation method" that chooses as final trained weights $\boldsymbol{w}^*$ those that have minimized the validation error among all training epochs, namely, the solution to the problem

$$\min_{\text{training } \boldsymbol{w}} \mathcal{E}_{\mathcal{V}}(\boldsymbol{w}) \,. \tag{13}$$

Herein, we used a combination of the early stopping and "minimum validation error" regularization methods. The combination of the two methods aims at speeding up the training time (early stopping method with p^* not too small) and improving the performance (minimum validation error method).

3.1.3 Fully-connected layers in neural networks

In most NN applications the graph architecture is characterized by layers, subset of units not connected to each other and connected only to units of other layers. The standard example of such a network is the fully-connected network (Fig. 1), characterized by a partition $U_0, \ldots, U_{h+1}$ of the nodes set U such that, for each $i = 1, \ldots, h$, each unit in layer U_i is connected only to all units in layers U_{i-1} and U_{i+1} with edges oriented in the direction of increasing indexes of partition sets. Layers U_0 and U_{h+1} are the sets of input and output units, respectively, and they are called *input layer* and *output layer*; all other subsets $U_1, \ldots, U_h$ are called hidden layers, since they are not directly in touch with input and output data of the problem. The parameter h is the depth of the network.

Usually, this kind of networks are preferred since they give advantages during the gradient computation with backpropagation.

3.2 Problem setting

In the test problem addressed in this paper, we consider a DFN with N fractures characterized by a fixed geometry, whereas fracture transmissivities $\kappa_1, \ldots, \kappa_N$ are either constant or modeled as random variables with log-normal distribution; more precisely, for a given $n \leq N$, we consider n fractures having

$$\log_{10} \kappa_i \sim \mathcal{N}(-5, 1/3) \,, \quad \forall\, i = 1, \ldots, n \,, \tag{14}$$

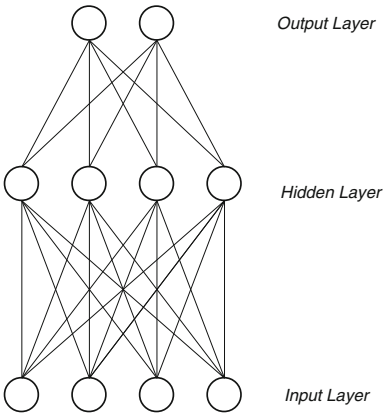

Fig. 1 Example of NN with fully-connected layers (depth 1)

whereas for the remaining $N-n$ fractures we set $\kappa_i = 10^{-5} = \bar{\kappa}$. The transmissivities values sampled range from $\kappa_{\min} \simeq 5 \cdot 10^{-7}$ to $\kappa_{\max} \simeq 10^{-4}$. Note that without loss of generality we assume that the fractures with random κ_i are the first n fractures. In the analysis several values of n will be considered. We impose boundary conditions in such a way that a number of fractures act as inlet flow fractures, whereas a number M act as outflow fractures, independently of the values of the transmissivities (see Sect. 4 for details). One of the major interests in such kind of applications relies in the computation of the total exiting flux and of its distribution among the outflow fractures. Frequently, most of the outflow occurs through a subset of the total number of exit fractures, so we focus our analysis assuming that a number $m \leq M$ of outputs is under investigation.

In the following subsection we describe the NN architecture used to address this issue.

3.2.1 Architecture

The NN architecture adopted for the flux regression problem is based on the multi-task learning structure described in Goodfellow et al. (2016), chapter 7.7. Such architectures are generally used when different tasks (the regression of the m outgoing flows, in our case) share common input variables (the n fracture transmissivities); the underlying assumption is that part of the information used for the training is shared by all the tasks, while other pieces of information are task-specific. These NNs are characterized by two typologies of layers:

- *shared layers:* layers shared by all the tasks. These layers are usually the first layers of the NN after the input layer. The weights of these layers benefit of the training effects from data associated to all tasks;
- *task-specific layers:* layers separated from other tasks. These layers are usually the last layers of the NN, ending in the output layer/unit representing one of the tasks. The weights of these layers benefit of the training effects only from data of the corresponding task.

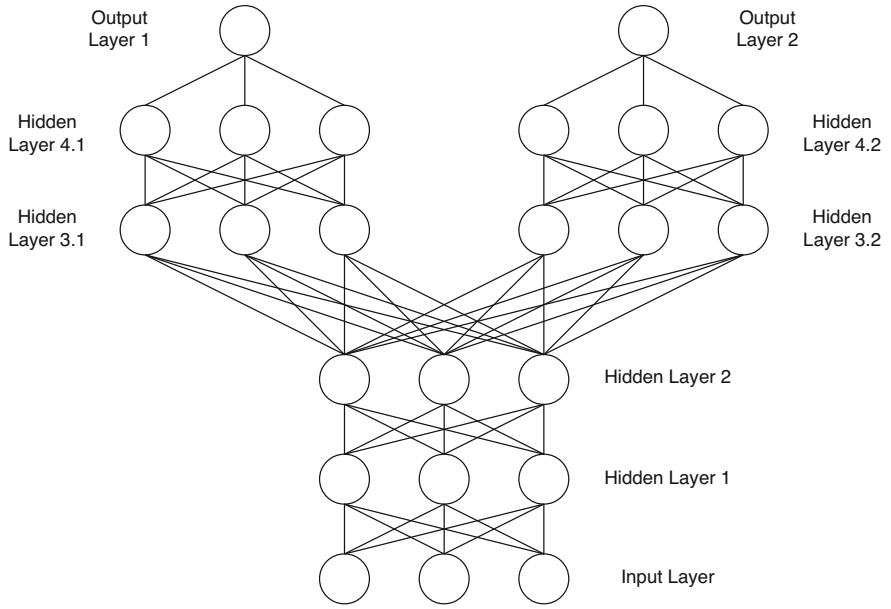

Fig. 2 Example of NN built for vector valued regression concerning flux prediction ($n = 3, m = 2, d = 2$)

The idea behind multi-task NN architectures is to build one model for all the tasks, instead of building a NN model for each task; in particular the idea is that the generalization ability of the NN can be improved thanks to the shared layers; obviously, the improvement can be achieved only if a relationship between the different tasks actually exists.

Let us consider $n \in \mathbb{N}$ (fixed) fractures with stochastic transmissivities and $m \in \mathbb{N}$ (fixed) boundary outflow fractures. Let $d \in \mathbb{N}$ be a parameter characterizing the depth of the NN and let us consider the following subnetworks:

- N_0: A fully-connected NN of depth $(d - 1)$ in which each layer has n softplus units;
- $N_1, \ldots, N_m$: m fully connected NNs of depth $(d - 1)$ in which each layer has n softplus units, with the exception of the last (output) layer which has only one linear unit, characterized by the identity activation function.

The choice of using softplus activation function was made after a preliminary investigation, comparing the performances obtained also with other activation functions.

Then, the overall multi-task NN is obtained connecting the last layer of N_0 to the first layers of $N_1, \ldots, N_m$, resulting in a $2d$-depth NN with n inputs and m outputs (see Fig. 2).

3.2.2 Dataset characterization

Let us introduce the following notation. Let $\widetilde{\boldsymbol{\kappa}} = [\kappa_1, \ldots, \kappa_N]^\top \in \mathbb{R}^N$ be the vector collecting all the transmissivities, and let $\boldsymbol{\kappa} \in \mathbb{R}^n$ be the subvector containing the

random transmissivities. Without loss of generality, we assume herein that the deterministic transmissivities are set to the common value $\bar{\kappa} = 10^{-5}$, and that the fractures with random transmissivity are the first n, so that $\widetilde{\boldsymbol{\kappa}} = [\boldsymbol{\kappa}, \bar{\kappa}, \ldots, \bar{\kappa}]^\top$.

Then, let $\widetilde{\boldsymbol{\varphi}} = [\varphi_1, \ldots, \varphi_M]^\top \in \mathbb{R}^M$ be the vector collecting all the exit flows, and $\boldsymbol{\varphi} \in \mathbb{R}^m$ the subvector containing the m components under investigation, which again we assume, for the ease of notation, to correspond to the first m components.

Let $F : \mathbb{R}^N \to \mathbb{R}^M$ be a function defined by

$$\widetilde{\boldsymbol{\varphi}} = F(\widetilde{\boldsymbol{\kappa}}), \tag{15}$$

that is, the function that provides the vector of outflows associated to the transmissivity input $\widetilde{\boldsymbol{\kappa}}$. Then, let $F_{nm} : \mathbb{R}^n \to \mathbb{R}^m$ be the function that provides the subset of exit flows under investigation $\boldsymbol{\varphi}$ as a function of the vector of random transmissivities $\boldsymbol{\kappa}$, being the other transmissivities fixed to the value $\bar{\kappa}$, namely

$$F_{nm}(\boldsymbol{\kappa}; \bar{\kappa}) = \boldsymbol{\varphi}. \tag{16}$$

Let us consider a number $D \in \mathbb{N}$ of samples $\boldsymbol{\kappa}_k \in \mathbb{R}^n$, $k = 1, \ldots, D$. The dataset $\mathcal{D}$ used for the creation of the training set, the test set and the validation set is

$$\mathcal{D} = \left\{ (\boldsymbol{\kappa}_k, \boldsymbol{\varphi}_k) \in \mathbb{R}^n \times \mathbb{R}^m \mid F_{nm}(\boldsymbol{\kappa}_k; \bar{\kappa}) = \boldsymbol{\varphi}_k \,,\ \forall\, k = 1, \ldots, D \right\}. \tag{17}$$

Starting from $\mathcal{D}$, the test set is created as a subset $\mathcal{P} \subset \mathcal{D}$ obtained by randomly picking approximately 30% of the elements in $\mathcal{D}$. The remaining elements are randomly split into two subsets $\mathcal{T}$ and $\mathcal{V}$, representing the training set and the validation set, respectively, and such that $|\mathcal{V}| \sim 20\%\, |\mathcal{D} \backslash \mathcal{P}|$.

4 Numerical results

In this section we present numerical results obtained performing vector valued regression on a given DFN. The DFN is made of fractures immersed in a 3-dimensional cubic domain $\mathbb{D}$ of edge $\ell = 1000$ m (see Figs. 3, 4). The fractures are represented as disks (modeled as octagons) and have been randomly generated sampling the geometrical parameters from given distributions, frequently used in the framework of DFN modeling (Svensk Kärnbränslehantering 2010; Hyman et al. 2016): truncated power law with cut-off for fracture radii, with exponent $\gamma = 2.5$ and upper and lower cut-off $r_u = 560$ and $r_0 = 50$, respectively; Fischer distribution for orientation, along a mean direction $\boldsymbol{\mu} = (0.0065,\ -0.0162,\ 0.9998)$ with dispersion parameter 17.8; uniform distribution for mass centers. Eight interconnected fractures have been generated with deterministic position and size, linking two opposite faces of $\mathbb{D}$ which will act as inlet and outlet faces, in order to guarantee that these faces are connected. Then, 400 fractures have been stochastically sampled from the overmentioned distributions and added to the deterministic fractures; finally, a connectivity analysis is performed in order to identify fractures disconnected from the network, which do not contribute to the flow through the network and are therefore removed. After the connectivity

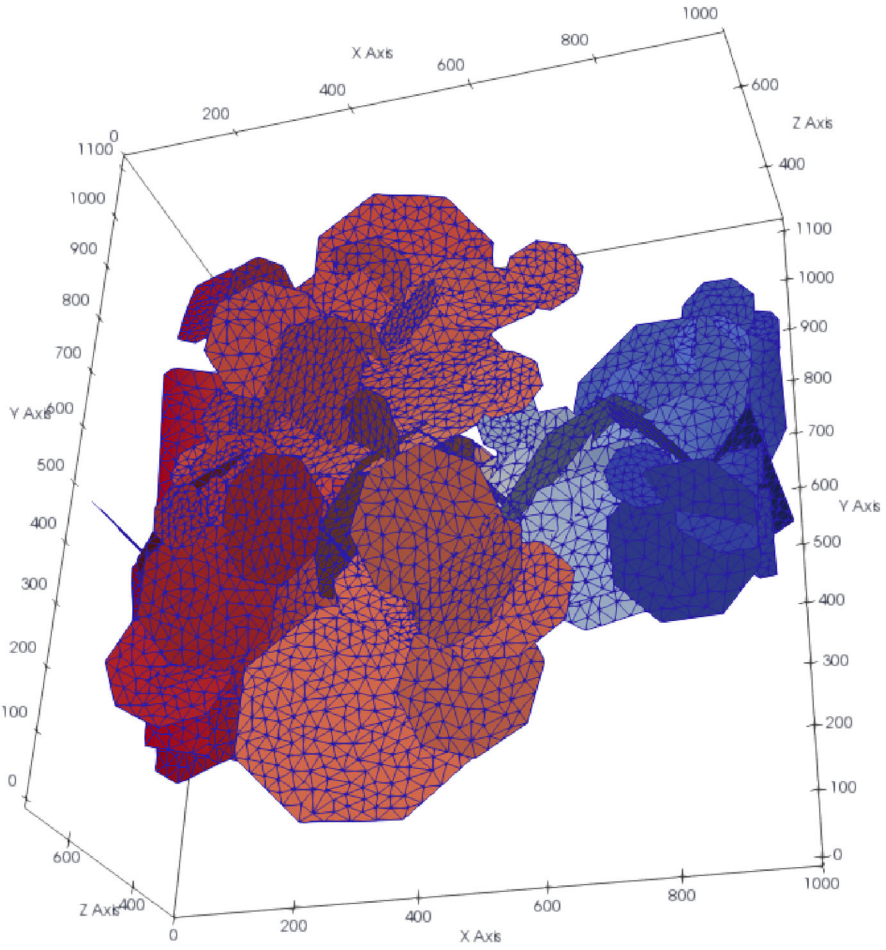

Fig. 3 DFN107. 3D view of the geometry of the network in the domain $\mathbb{D}$

analysis, the resulting main connected component is made of $N = 107$ fractures. For future reference, we label this network DFN107.

We impose boundary conditions in such a way that two opposite faces of $\mathbb{D}$ represent an inlet and outlet face, respectively; with reference to Fig. 3, we impose a Dirichlet boundary condition $H = 10$ on fracture edges created intersecting the DFN with the leftmost face of $\mathbb{D}$, corresponding to $x = 0$, and $H = 0$ on the edges obtained intersecting the DFN with the rightmost face of $\mathbb{D}$ ($x = \ell$).

In test problem DFN107 the total number of outflow fractures is $M = 7$. Here the analysis is focused on a subset of $m = \lceil M/2 \rceil = 4$ fractures, which are selected as those exhibiting the largest average exit flows among all the fracture transmissivity samplings; referring to the global indexing, they are fractures $\mathcal{F}_8$, $\mathcal{F}_{33}$, $\mathcal{F}_{62}$, $\mathcal{F}_{105}$.

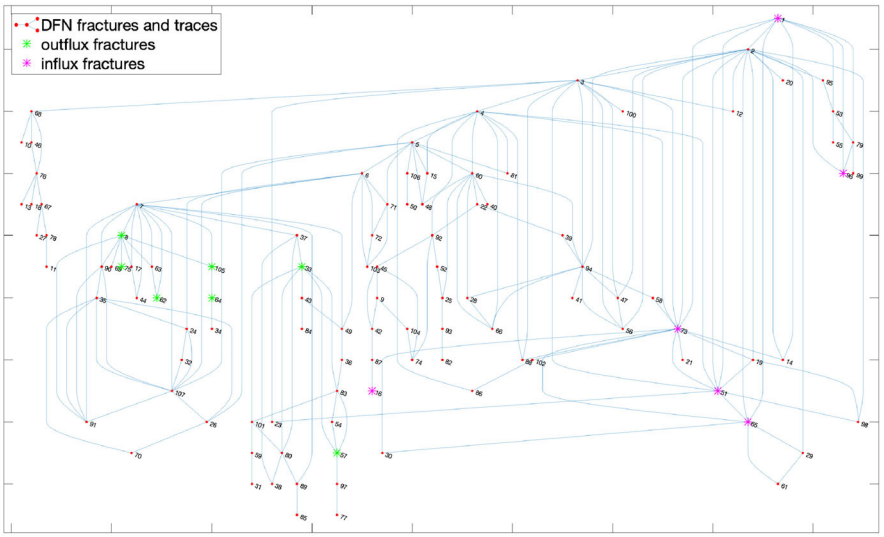

Fig. 4 DFN107. Graph visualization. Nodes correspond to fractures, and are numbered according to the fracture numbering. Edges correspond to fracture intersections. Magenta and green markers highlight inlet and outlet fractures, respectively

Since from now on m is fixed, for the ease of notation we drop the dependence from m and $\bar{\kappa}$ in (16), so we set $F_n(\boldsymbol{\kappa}) := F_{nm}(\boldsymbol{\kappa}; \bar{\kappa})$. For a fixed $n \leq N$, we will denote by $\widehat{F}_n(\boldsymbol{\kappa}) = \widehat{F}_n(\boldsymbol{\kappa}; \widehat{\boldsymbol{w}}^*)$ the NN function approximating F_n obtained with the training.

4.1 Dataset preparation

We will consider, in the following, the cases $n = 30$ ($\sim 30\%$ of the total number of fractures N), $n = 80$ ($\sim 75\%$ of N) and $n = 107 = N$. For each case a dataset $\mathcal{D}_n$ is created with $|\mathcal{D}_n| = D = 10{,}000$:

$$\mathcal{D}_n = \left\{ (\boldsymbol{\kappa}_k, \boldsymbol{\varphi}_k) \in \mathbb{R}^n \times \mathbb{R}^4 \mid F_n(\boldsymbol{\kappa}_k) = \boldsymbol{\varphi}_k \,,\ \forall k = 1, \ldots, D \right\} . \tag{18}$$

Then $\mathcal{D}_n$ is split as $\mathcal{D}_n = \mathcal{P}_n \cup \mathcal{T}_n \cup \mathcal{V}_n$ (see Sect. 3.2.2) with $|\mathcal{P}_n| = P = 3000$, $|\mathcal{V}_n| = V = 1400$ and $|\mathcal{T}_n| = T = 5600$. The values of $\boldsymbol{\varphi}_k$ have been obtained with the method described in Sect. 2. Fluxes and transmissivities here are reported in $\mathrm{mm}^2\mathrm{s}^{-1}$ instead of $\mathrm{m}^2\mathrm{s}^{-1}$, getting rid of a factor 10^{-6}.

For each $n = 30, 80, 107$, four NNs are built, characterized by the architecture described in Sect. 3.2.1, with the following parameters:

- *depth parameter* $d = 1, 3$ (then NNs have depth varying among values $2d = 2, 6$);
- *mini-batch size* $B = |\mathcal{B}_n| = 10, 30$ (Sect. 3.1.1).

The NNs have been trained using a mini-batch method with mini-batch set $\mathcal{B}_n \subset \mathcal{T}_n$ of cardinality B and characterized by a combination of the early stopping and "minimum validation error" regularization methods (Sect. 3.1.2). We fixed the maximum number

of training epochs to $c_{max} = 1000$ and the patience parameter for early stopping method to $p^* = 150$.

All the NNs have been implemented using the *keras* (Chollet et al. 2015) machine learning package in python 3 (*tensorflow* backend, Abadi et al. 2015), while for data preparation we mainly used *pandas* (McKinney 2010) and *scikit-learn* (Pedregosa 2011) packages. Training of NNs was performed by means of the ADAM optimizer implemented in *keras* package, with its default options. ADAM, originally described in Kingma and Ba (2014), is a stochastic gradient-based optimization method considered as the state of the art method for training NNs.

From now on, we will refer to these NNs with the following notation:

$$\mathcal{N}_{n,d}^{B}\,, \quad \forall\, n = 30, 80, 107\,, \quad \forall\, d = 1, 3\,, \quad \forall\, B = 10, 30\,. \tag{19}$$

After the training, the performances of all the NNs (19) have been compared (Sect. 4.3) performing a grid search method (Goodfellow et al. 2016, chapter 11.4.3) with respect to parameters n, d and B. This method is used in order to choose the hyper-parameters most suitable for the target of the problem. To this aim, in Sect. 4.2, we introduce the performance measures necessary for the comparisons.

4.2 Performance measures

We start our analysis presenting some performance measures used to evaluate flux regression models.

Let $\boldsymbol{\kappa} \in \mathbb{R}^n$ be an input vector and let $\boldsymbol{\varphi} \in \mathbb{R}^m$ be the corresponding vector of actual fluxes (computed via a DFN simulation) and $\widehat{\boldsymbol{\varphi}}$ the vector of fluxes predicted by a NN $\mathcal{N}$. We consider the following errors:

$$\begin{aligned} \boldsymbol{e}^a(\widehat{\boldsymbol{\varphi}}) &:= |\boldsymbol{\varphi} - \widehat{\boldsymbol{\varphi}}| = (|\varphi_1 - \widehat{\varphi}_1|, \ldots, |\varphi_m - \widehat{\varphi}_m|) \quad \text{(absolute errors)} \\ \boldsymbol{e}^r(\widehat{\boldsymbol{\varphi}}) &:= \boldsymbol{e}^a(\widehat{\boldsymbol{\varphi}}) \cdot \frac{1}{\sum_{i=1}^{M} \varphi_i} \qquad\qquad\qquad\quad \text{(relative errors)} \end{aligned} \tag{20}$$

Note that the relative errors are computed with respect to the total exit flow $\sum_{i=1}^{M} \varphi_i$.

In order to assess the performance of regression models, we consider a test set $\mathcal{P}$ and introduce the following error sets:

$$\begin{aligned} E^a(\mathcal{N}\,;\,\mathcal{P}) &= \left\{ \boldsymbol{e}^a(\widehat{\boldsymbol{\varphi}}_p) \in \mathbb{R}^m \,|\, \forall\, (\boldsymbol{\kappa}_p, \boldsymbol{\varphi}_p) \in \mathcal{P} \right\}\,, \\ e^a(\mathcal{N}\,;\,\mathcal{P}) &= \left\{ e_j^a(\widehat{\boldsymbol{\varphi}}_p) \in \mathbb{R} \,|\, \forall\, (\boldsymbol{\kappa}_p, \boldsymbol{\varphi}_p) \in \mathcal{P}\,,\, \forall\, j = 1, \ldots, m \right\}\,, \end{aligned} \tag{21}$$

being $e_j^a(\widehat{\boldsymbol{\varphi}}_p)$ the jth element of $\boldsymbol{e}^a(\widehat{\boldsymbol{\varphi}}_p)$; in practice, $e^a(\mathcal{N}\,;\,\mathcal{P})$ is the set of all the components of all vectors $\boldsymbol{e}^a(\widehat{\boldsymbol{\varphi}}_p) \in E^a(\mathcal{N}\,;\,\mathcal{P})$. Similar definitions hold for the relative errors.

Then, we introduce statistic information on $E^a(\mathcal{N}\,;\,\mathcal{P})$ (e.g., expected value, sample standard deviation, percentiles, etc.) componentwise; for example, the vector of

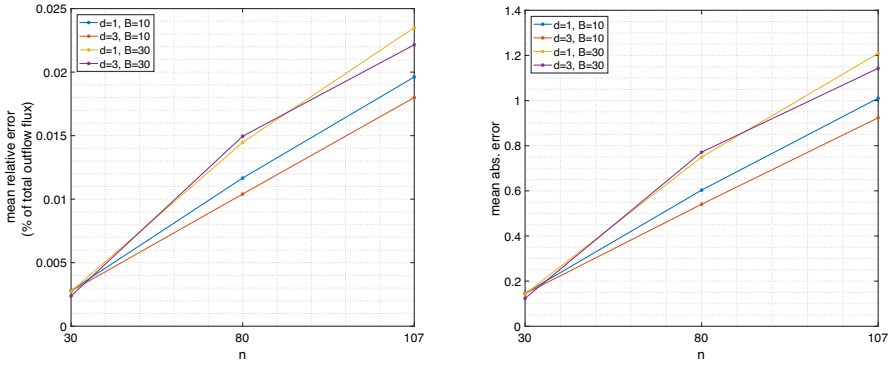

Fig. 5 DFN107. Mean errors in the flux prediction obtained with $\mathcal{N}^B_{n,d}$ versus the number of input units n, for several values of the depth parameter d and mini-batch size B. Left: mean relative error $\mathbb{E}[e^r(\mathcal{N}^B_{n,d}\,;\mathcal{P}_n)]$; right: mean absolute error $\mathbb{E}[e^a(\mathcal{N}^B_{n,d}\,;\mathcal{P}_n)]$

expected values $\mathbb{E}[E^a(\mathcal{N}\,;\mathcal{P})] \in \mathbb{R}^m$ is such that

$$\mathbb{E}[E^a(\mathcal{N}\,;\mathcal{P})] = |E^a(\mathcal{N}\,;\mathcal{P})|^{-1} \sum_{(\boldsymbol{\kappa}_p,\boldsymbol{\varphi}_p)\in\mathcal{P}} \boldsymbol{e}^a(\widehat{\boldsymbol{\varphi}}_p) = \left[\mathbb{E}[e^a_1(\widehat{\boldsymbol{\varphi}}_p)],\ldots,\mathbb{E}[e^a_m(\widehat{\boldsymbol{\varphi}}_p)]\right]^\top . \quad (22)$$

On the other hand, statistic information about set $e^a(\mathcal{N}\,;\mathcal{P})$ provides a global description of absolute errors, with respect to all fractures together; therefore, for example, expected value $\mathbb{E}[e^a(\mathcal{N}\,;\mathcal{P})]$ is the scalar value such that

$$\mathbb{E}[e^a(\mathcal{N}\,;\mathcal{P})] = |e^a(\mathcal{N}\,;\mathcal{P})|^{-1} \sum_{(\boldsymbol{\kappa}_p,\boldsymbol{\varphi}_p)\in\mathcal{P}} \sum_{j=1}^{m} e^a_j(\widehat{\boldsymbol{\varphi}}_p)\,. \quad (23)$$

Analogous definitions hold for $E^r(\mathcal{N}\,;\mathcal{P})$, $e^r(\mathcal{N}\,;\mathcal{P})$ and the corresponding statistic information.

4.3 Regression results and performance comparisons

A first comparison of the performances of the NNs listed in (19) is obtained observing the mean relative errors on the test sets for all outflow fractures (see Fig. 5), namely

$$\mathbb{E}[e^r(\mathcal{N}^B_{n,d}\,;\mathcal{P}_n)]\,, \quad \forall\, n = 30, 80, 107\,, \quad \forall\, d = 1, 3\,, \quad \forall\, B = 10, 30\,. \quad (24)$$

As seen in Fig. 5, $\mathbb{E}[e^r(\mathcal{N}^B_{n,d}\,;\mathcal{P}_n)]$ is generally increasing with respect to the input dimension n, whereas it is in general decreasing with respect to d, for fixed n and B, and increasing with respect to B, for fixed n and d, with few exceptions. However, we remark that the dependence of the errors on the parameter d is rather moderate, denoting a good stability of the approach with respect to the depth of the architecture.

As far as the training times and last training epochs are concerned, we can observe that the number of epochs and the training time generally increase with respect to the

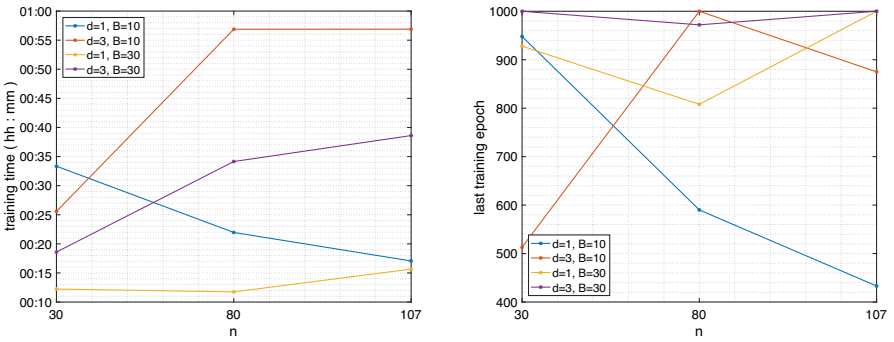

Fig. 6 DFN107. Training time (left) and training epochs (right) of $\mathcal{N}^B_{n,d}$ versus the number of input units n, for several values of the depth parameter d and mini-batch size B

depth of the network and decrease with respect to mini-batch size B; however, due to the stochastic nature of the ADAM optimizer and a not small patience parameter ($p^* = 150$), these relationships show many exceptions (Fig. 6). These results however suggest that the choice of the batch size and of the depth can impact on the efficiency, and a deeper investigation on the hyper-parameters is worth being addressed, in order to improve efficiency; such investigation is deferred to future work as the main focus here is on accuracy of the NNs.

In the next section a detailed analysis of $\mathcal{N}^* := \mathcal{N}^{10}_{107,3}$ is proposed. The choice is motivated by the fact that the case $n = 107 = N$ is the one in which all fractures are characterized by a random transmissivity, which is typically the most interesting case, also in practice; for this value of n, the previous analysis shows that the best performances are attained with $d = 3$ and $B = 10$.

4.4 Detailed analysis of $\mathcal{N}^*$ neural network

A first glance at the predicting ability of $\mathcal{N}^*$ is given by its detailed statistic information on errors reported in Table 1, both with respect to each outflow fracture and globally; in particular, in this table, the mean value $\mathbb{E}$, the sample standard deviation σ, the main percentiles and the minimum and maximum values are reported, with respect to $e^r(\mathcal{N}^*; \mathcal{P}_{107})$ and for elements of vectors $\boldsymbol{e}^r(\widehat{\boldsymbol{\varphi}}_p) \in E^r(\mathcal{N}^*; \mathcal{P}_{107})$ (see (22) and (23) for descriptions about evaluation of these statistic informations).
The results obtained show that mean values are quite satisfactory: the average error, considering all the outflow fractures, is $\sim 1.8\%$ of the total exiting flux. The good predicting ability of $\mathcal{N}^*$ can be also observed in Fig. 7, where the regression scatter plots are reported; the red lines correspond to exact prediction.

For a more detailed analysis of the results obtained with $\mathcal{N}^*$, the distributions of the actual fluxes and of the predicted fluxes are compared. In particular, the dissimilarity between the two distributions can be quantified by means of two divergence measures, the Kullback–Leibler divergence (Kullback and Leibler 1951; Kullback 1968) and the Jensen–Shannon divergence (Amari et al. 1987; Lin 1991).

Table 1 DFN107

	$e^r(\mathcal{N}^*;\mathcal{P}_{107})$	$E^r(\mathcal{N}^*;\mathcal{P}_{107})$			
	(global)	$(\mathcal{F}_8)$	$(\mathcal{F}_{33})$	$(\mathcal{F}_{62})$	$(\mathcal{F}_{105})$
Number of elements	12,000	3000	3000	3000	3000
$\mathbb{E}$	0.0180	0.0241	0.0188	0.0155	0.0135
σ	0.0219	0.0297	0.0201	0.0182	0.0154
Min	0.2e−5	1.5e−5	0.2e−5	0.2e−5	1.1e−5
50th percentile	0.0117	0.0158	0.0134	0.0102	0.0091
75th percentile	0.0230	0.0308	0.0249	0.0198	0.0174
97th percentile	0.0692	0.0952	0.0682	0.0591	0.0513
Max	0.4296	0.4296	0.2808	0.2669	0.1671

Statistic information on relative errors with respect to predictions of $\mathcal{N}^*$ ($n = N = 107$, $d = 3$, $B = 10$) on $\mathcal{P}_{107}$

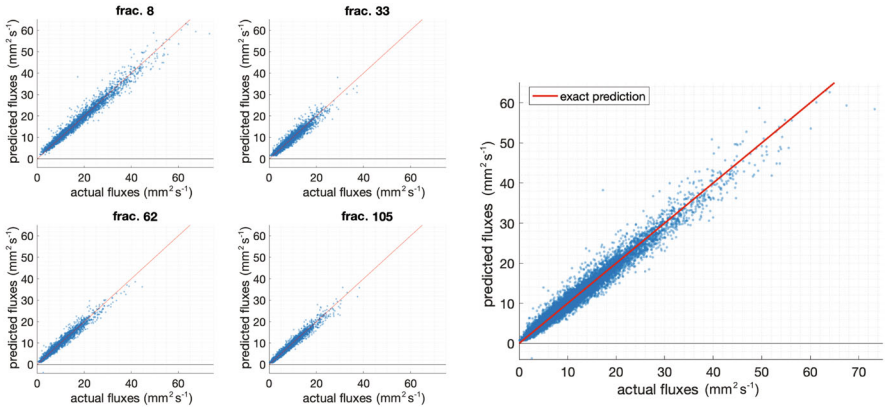

Fig. 7 DFN107. Fluxes predicted by $\mathcal{N}^*$ on inputs of $\mathcal{P}_{107}$ versus the corresponding actual fluxes. Left: scatter plots for each outflow fracture; right: cumulative plot. Red lines correspond to exact predictions

The Kullback–Liebler (KL) divergence $D_{\mathrm{KL}}(P||Q)$ between two continuous distributions P and Q, defined on the same probability space, with probability density functions p and q, respectively, is defined by the following integral (Bishop 2006):

$$D_{\mathrm{KL}}(P||Q) = \int_{-\infty}^{+\infty} p(x) \log\left(\frac{p(x)}{q(x)}\right) \mathrm{d}x. \tag{25}$$

Note that D_{KL} is always greater than or equal to zero; distributions P and Q are equal almost everywhere if and only if $D_{\mathrm{KL}}(P||Q) = D_{\mathrm{KL}}(Q||P) = 0$. KL divergence is indeed not symmetric; in order to introduce a symmetric divergence, the Jensen–Shannon (JS) divergence can be taken into account:

$$D_{\mathrm{JS}}(P||Q) = \frac{1}{2}\left(D_{\mathrm{KL}}(P||Q) + D_{\mathrm{KL}}(Q||P)\right). \tag{26}$$

Again, $D_{\mathrm{JS}}(P||Q) = 0$ if and only if $P = Q$ almost everywhere. KL and JS divergences are also defined for two discrete probability distributions P and Q defined on the same probability space, on a discrete set $\mathcal{X}$, as follows (MacKay 2002):

$$D_{\mathrm{KL}}(P||Q) = \sum_{x\in\mathcal{X}} p(x)\log\left(\frac{p(x)}{q(x)}\right), \tag{27}$$

where p and q are the probability mass functions of P and Q, respectively.

In the next subsections we will analyze differences between distributions of actual fluxes and fluxes predicted by $\mathcal{N}^*$.

4.4.1 Comparison of flux distributions for $\mathcal{N}^*$

Let $\mathcal{A} \subset \mathbb{R}^N \times \mathbb{R}^4$ be a given set (e.g. the test set $\mathcal{P}$); for each $(\boldsymbol{\kappa}, \boldsymbol{\varphi}) \in \mathcal{A}$, let $\widehat{\boldsymbol{\varphi}} \in \mathbb{R}^4$ be the prediction of $\boldsymbol{\varphi}$ made by $\mathcal{N}^*$ for a given $\boldsymbol{\kappa}$ and, for each $j = 1, \ldots, 4$, let φ_j and $\widehat{\varphi}_j$ be the jth elements of $\boldsymbol{\varphi}$ and $\widehat{\boldsymbol{\varphi}}$, respectively. For each $j = 1, \ldots, 4$, we will denote by $q_j(\mathcal{A})$ and $\widehat{q}_j(\mathcal{A}\,;\mathcal{N}^*)$ the probability density functions (pdf) of φ_j and $\widehat{\varphi}_j$, respectively, on the set $\mathcal{A}$.

Since continuous distributions of fluxes (both actual and predicted ones) are not given a priori, distributions $q_j(\mathcal{A})$ and $\widehat{q}_j(\mathcal{A}\,;\mathcal{N}^*)$ have been computed with Matlab routine *ksdensity*, which performs kernel density estimation of probability distributions from data; the functions obtained are discrete approximations of the pdf, and therefore we will refer to them as probability mass functions (pmf); correspondingly, the KL and JS divergences between two distributions characterized by the two estimated pmfs have been computed using the discrete version of KL divergence described in (27).

We can now analyze prediction performances of $\mathcal{N}^*$ comparing $q_j(\mathcal{P}_{107})$ and $\widehat{q}_j(\mathcal{P}_{107}\,;\mathcal{N}^*)$, for $j = 1, \ldots, 4$, where $\mathcal{P}_{107}$ is the test set of $\mathcal{D}_{107}$ (see (18)). A first comparison can be made observing Fig. 8 and Tables 2 and 3: for each j, not only actual and predicted distribution plots are very similar (see Fig. 8) but also expected values, sample standard deviation values and percentile values are remarkably close (compare values in Tables 2, 3).

In order to obtain a quantitative measure of similarity between the actual and predicetd pmf, we compute their KL and JS divergences, which are reported in Table 4. All divergence values are smaller than 0.01, confirming previous deductions.

These results in particular suggest that, even if some high prediction errors may occur (e.g., values in "max" row of Table 1), they are negligible from the statistical point of view and $\mathcal{N}^*$, with its predictions, reconstructs flux distributions very similar to the actual ones.

4.4.2 Running $\mathcal{N}^*$ on inputs with partially fixed transmissivities

In this section we highlight the capability of $\mathcal{N}^*$ to predict fluxes also within a framework not addressed in its training phase; namely, a case in which only a subset of transmissivities are actually random, despite $\mathcal{N}^*$ has been trained considering all transmissivites as log-normal random variables.

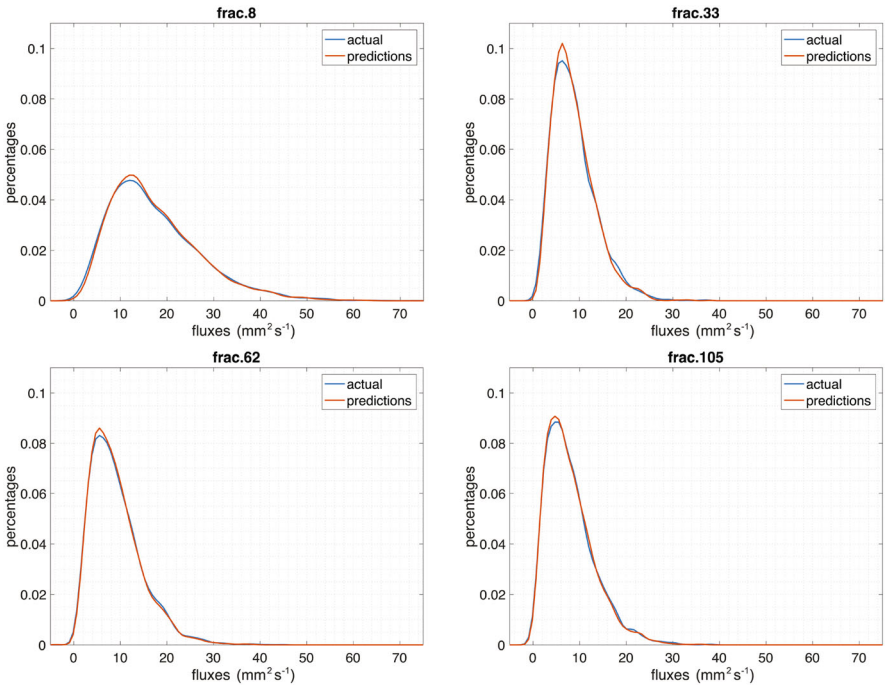

Fig. 8 DFN107. Pmf of actual fluxes in $\mathcal{P}_{107}$ ($q_j(\mathcal{P}_{107})$) compared with the corresponding pmf obtained from $\mathcal{N}^*$ flux predictions on inputs of $\mathcal{P}_{107}$ ($\widehat{q}_j(\mathcal{P}_{107}; \mathcal{N}^*)$)

Table 2 DFN107

	$q_1(\mathcal{P}_{107})$ ($\mathcal{F}_8$)	$q_2(\mathcal{P}_{107})$ ($\mathcal{F}_{33}$)	$q_3(\mathcal{P}_{107})$ ($\mathcal{F}_{62}$)	$q_4(\mathcal{P}_{107})$ ($\mathcal{F}_{105}$)
Number of elements	3000	3000	3000	3000
$\mathbb{E}$	17.4447	8.8383	9.1650	8.0058
σ	9.8431	4.7194	5.5791	5.2130
Min	1.2334	0.3559	0.4514	0.08256
3th percentile	4.2222	2.3880	2.0169	1.3648
50th percentile	15.3146	8.0007	8.0479	6.9079
75th percentile	22.8561	11.3141	12.0380	10.6672
97th percentile	40.0750	19.5170	21.4465	20.4751
Max	73.2941	37.2726	44.7325	38.2596

Statistics on actual fluxes (mm^2 s^{-1}) in $\mathcal{P}_{107}$

To this aim, let us consider datasets $\mathcal{D}_{30} \subset \mathbb{R}^{30} \times \mathbb{R}^4$ and $\mathcal{D}_{80} \subset \mathbb{R}^{80} \times \mathbb{R}^4$ used for creation of training set and test set of NNs $\mathcal{N}^B_{30,d}$ and $\mathcal{N}^B_{80,d}$ (for each $d = 1, 3$, $B = 10, 30$), respectively. Then, we create sets $\widetilde{\mathcal{D}}_{30}, \widetilde{\mathcal{D}}_{80} \in \mathbb{R}^{107} \times \mathbb{R}^4$ by extending the transmissivity vectors to vectors in $\mathbb{R}^N$, setting the additional elements to the constant value $\bar{\kappa}$. By means of sets $\widetilde{\mathcal{D}}_{30}$ and $\widetilde{\mathcal{D}}_{80}$, we can therefore test performances of $\mathcal{N}^*$

Table 3 DFN107

	$\widehat{q}_1(\mathcal{P}_{107}\,;\mathcal{N}^*)$ $(\mathcal{F}_8)$	$\widehat{q}_2(\mathcal{P}_{107}\,;\mathcal{N}^*)$ $(\mathcal{F}_{33})$	$\widehat{q}_3(\mathcal{P}_{107}\,;\mathcal{N}^*)$ $(\mathcal{F}_{62})$	$\widehat{q}_4(\mathcal{P}_{107}\,;\mathcal{N}^*)$ $(\mathcal{F}_{105})$
Number of elements	3000	3000	3000	3000
$\mathbb{E}$	17.4866	8.7957	9.0551	7.9320
σ	9.5208	4.5331	5.3992	5.0926
Min	2.0972	1.5140	-3.7443	0.8239
3rd percentile	4.7918	2.6595	1.9861	1.4126
50h percentile	15.4199	7.9420	7.9850	6.8133
75th percentile	22.6945	11.2383	11.8539	10.6492
97th percentile	39.9464	19.2921	21.1260	19.8495
Max	63.0359	37.9709	38.6921	36.0230

Statistics on $\mathcal{N}^*$ predicted fluxes ($\mathrm{mm}^2\mathrm{s}^{-1}$) obtained on inputs of $\mathcal{P}_{107}$

Table 4 DFN107

Divergence	$j=1$ $(\mathcal{F}_8)$	$j=2$ $(\mathcal{F}_{33})$	$j=3$ $(\mathcal{F}_{62})$	$j=4$ $(\mathcal{F}_{105})$
$D_{\mathrm{KL}}(\,\widehat{q}_j(\mathcal{P}_{107}\,;\mathcal{N}^*)\,\|\|\,q_j(\mathcal{P}_{107})\,)$	0.0034	0.0047	0.0021	0.0027
$D_{\mathrm{KL}}(\,q_j(\mathcal{P}_{107})\,\|\|\,\widehat{q}_j(\mathcal{P}_{107}\,;\mathcal{N}^*)\,)$	0.0074	0.0053	0.0074	0.0015
$D_{\mathrm{JS}}(\,q_j(\mathcal{P}_{107})\,\|\|\,\widehat{q}_j(\mathcal{P}_{107}\,;\mathcal{N}^*)\,)$	0.0054	0.0050	0.0048	0.0021

Kullback–Liebler and Jensen–Shannon divergences between distributions of actual fluxes in $\mathcal{P}_{107}$ ($q_j(\mathcal{P}_{107})$) and distributions of $\mathcal{N}^*$ predicted fluxes on inputs of $\mathcal{P}_{107}$ ($\widehat{q}_j(\mathcal{P}_{107}\,;\mathcal{N}^*)$)

Table 5 DFN107

Divergence	$j=1$ $(\mathcal{F}_8)$	$j=2$ $(\mathcal{F}_{33})$	$j=3$ $(\mathcal{F}_{62})$	$j=4$ $(\mathcal{F}_{105})$
$D_{\mathrm{JS}}(\,q_j(\mathcal{D}_{30})\,\|\|\,\widehat{q}_j(\widetilde{\mathcal{D}}_{30}\,;\mathcal{N}^*)\,)$	0.0070	0.0079	0.0036	0.0097
$D_{\mathrm{JS}}(\,q_j(\mathcal{D}_{80})\,\|\|\,\widehat{q}_j(\widetilde{\mathcal{D}}_{80}\,;\mathcal{N}^*)\,)$	0.0053	0.0086	0.0025	0.0150

Jensen–Shannon divergences between distributions of actual fluxes in $\mathcal{D}_{30}$, $\mathcal{D}_{80}$ ($q_j(\mathcal{D}_{30})$, $q_j(\mathcal{D}_{80})$) and distributions of $\mathcal{N}^*$ predicted fluxes on inputs of $\widetilde{\mathcal{D}}_{30}$ and $\widetilde{\mathcal{D}}_{80}$ ($\widehat{q}_j(\widetilde{\mathcal{D}}_{30}\,;\mathcal{N}^*)$, $\widehat{q}_j(\widetilde{\mathcal{D}}_{80}\,;\mathcal{N}^*)$)

on cases with random transmissivities restricted to the first 30 and 80 fractures. We remark that since $\mathcal{N}^*$ has been trained and tested on $\mathcal{D}_{107}$, the whole datasets $\widetilde{\mathcal{D}}_{30}$ and $\widetilde{\mathcal{D}}_{80}$ can now be used for testing $\mathcal{N}^*$, as none of their pairs has been used in training $\mathcal{N}^*$.

In Table 5 we report the JS divergences of $q_j(\mathcal{D}_n) \equiv q_j(\widetilde{\mathcal{D}}_n)$ and $\widehat{q}_j(\widetilde{\mathcal{D}}_n\,;\mathcal{N}^*)$, for $n = 30, 80$, and for $j = 1, \ldots, 4$. The very low divergence values (most of them are again smaller than 0.01) highlight a good approximation of pmf $q_j(\mathcal{D}_n)$ made by $\mathcal{N}^*$ predictions and therefore also a good generalization ability with respect to new inputs $\boldsymbol{\kappa} \in \mathbb{R}^N$ with several fixed transmissivities.

Table 6 DFN107

Divergence	$j=1$ $(\mathcal{F}_8)$	$j=2$ $(\mathcal{F}_{33})$	$j=3$ $(\mathcal{F}_{62})$	$j=4$ $(\mathcal{F}_{105})$
$D_{\mathrm{JS}}(\, q_j(\mathcal{P}_{30}) \,\|\| \, \widehat{q}_j(\widetilde{\mathcal{P}}_{30}\,;\mathcal{N}^*)\,)$	0.0062	0.0063	0.0047	0.0095
$D_{\mathrm{JS}}(\, q_j(\mathcal{P}_{30}) \,\|\| \, \widehat{q}_j(\mathcal{P}_{30}\,;\mathcal{N}^{10}_{30,3})\,)$	0.0015	0.0004	0.0003	0.0001
$D_{\mathrm{JS}}(\, q_j(\mathcal{P}_{80}) \,\|\| \, \widehat{q}_j(\widetilde{\mathcal{P}}_{80}\,;\mathcal{N}^*)\,)$	0.0044	0.0050	0.0029	0.0121
$D_{\mathrm{JS}}(\, q_j(\mathcal{P}_{80}) \,\|\| \, \widehat{q}_j(\mathcal{P}_{80}\,;\mathcal{N}^{10}_{80,3})\,)$	0.0009	0.0025	0.0028	0.0029

Top two rows: Jensen–Shannon divergences between distributions of actual fluxes in $\mathcal{P}_{30}$ ($q_j(\mathcal{P}_{30})$) and distributions of predicted fluxes on inputs of $\widetilde{\mathcal{P}}_{30}$ and $\mathcal{P}_{30}$, respectively ($\widehat{q}_j(\widetilde{\mathcal{P}}_{30}\,;\mathcal{N}^*)$, $\widehat{q}_j(\mathcal{P}_{30}\,;\mathcal{N}^{10}_{30,3})$). Bottom two rows: same as top two rows, but on $\widetilde{\mathcal{P}}_{80}$, $\mathcal{P}_{80}$

We end this section comparing the behavior of $\mathcal{N}^*$, in the case of partially fixed transmissivity, with those of the NNs specifically trained in the corresponding framework (namely, $\mathcal{N}^{10}_{30,3}$ and $\mathcal{N}^{10}_{80,3}$).

For all $j = 1, \ldots, 4$, let $\widehat{q}_j(\mathcal{P}_{30}\,;\mathcal{N}^{10}_{30,3})$ and $\widehat{q}_j(\mathcal{P}_{80}\,;\mathcal{N}^{10}_{80,3})$ be the pmf of predicted fluxes obtained by $\mathcal{N}^{10}_{30,3}$ and $\mathcal{N}^{10}_{80,3}$, respectively, on their corresponding test sets. We evaluate JS divergence between $q_j(\mathcal{P}_n)$ and distributions of predictions of $\mathcal{N}^*$ and $\mathcal{N}^{10}_{n,3}$ (see Table 6) on $\widetilde{\mathcal{P}}_n$ and $\mathcal{P}_n$, respectively, where $n = 30, 80$ and sets $\widetilde{\mathcal{P}}_n$ are obtained extending $\mathcal{P}_n$ as previously depicted for $\widetilde{\mathcal{D}}_n$. As expected, JS divergence values for the NN specifically trained are smaller than those for $\mathcal{N}^*$. However, $\mathcal{N}^{10}_{30,3}$ and $\mathcal{N}^{10}_{80,3}$ in general don't outperform $\mathcal{N}^*$ and therefore, in view of the possibile need to apply the method in a quite general framework, possibly with different choices of n, it seems to be worthwhile to train a single NN (e.g., $\mathcal{N}^*$) on the general case $n = N$, instead of training a NN for each n value to be considered.

5 $\mathcal{N}^*$ and uncertainty quantification

In Sect. 4.4.1 we observed that the pmf $\widehat{q}_j(\mathcal{P}_{107}\,;\mathcal{N}^*)$, predicted by $\mathcal{N}^*$, well approximates $q_j(\mathcal{P}_{107})$ for each $j = 1, \ldots, 4$. These results prove the ability of the NNs to correctly reproduce the statistical properties of the phenomenon, and suggest that NNs can also be used as a support for performing uncertainty quantification (UQ) analyses. While deferring to a further work a thorough analysis, we anticipate here some preliminary results about effectiveness of the approach. The effectiveness may of course depend on a trade-off between the number of simulations needed for the training and the number of simulations needed by the UQ analysis. In this respect, deeper investigations on the NN topology can strongly reduce the dimension of the training set.

In order to investigate these aspects we start our preliminary investigation analysing the sensitivity of NN performances with respect to the cardinality of the training and validation set $t = |\mathcal{T} \cup \mathcal{V}|$. Namely, focusing on the architecture already depicted in Sect. 3, and on the parameters d, B used for $\mathcal{N}^*$, we consider different variants of the NN, training it on sets with different cardinality and where 20% of elements are used

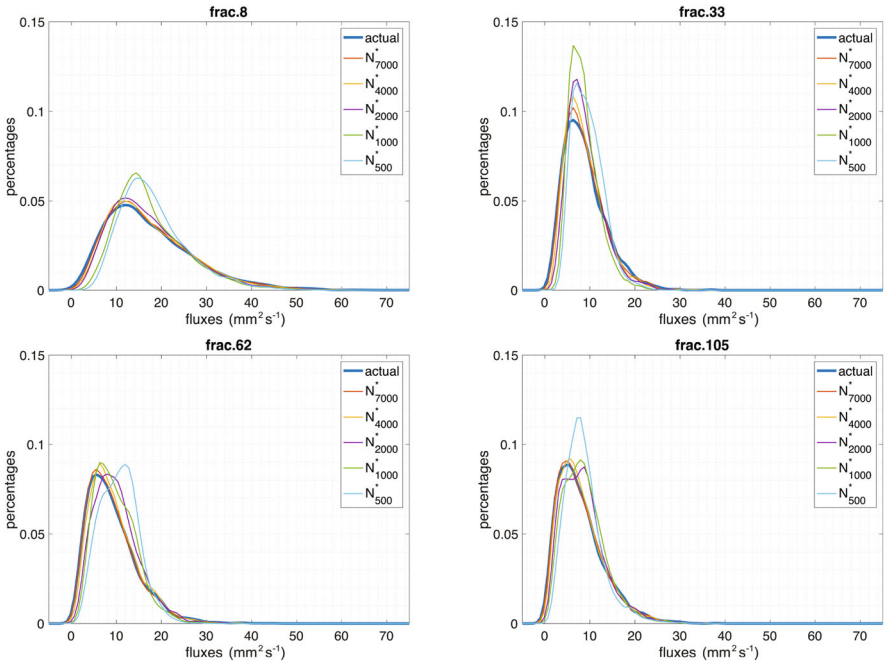

Fig. 9 DFN107. Pmf of actual fluxes in $\mathcal{P}_{107}$ ($q_j(\mathcal{P}_{107})$) compared with the corresponding pmf $\widehat{q}_j(\mathcal{P}_{107}; \mathcal{N}_t^*)$ obtained from $\mathcal{N}_t^*$ flux predictions on inputs of $\mathcal{P}_{107}$, for several values of the training and validation set size t

as validation set (i.e. $|\mathcal{V}| = 20\%\,|\mathcal{T} \cup \mathcal{V}|$); thus we introduce the NNs

$$\mathcal{N}_t^*\,, \quad \forall\, t \in \{500, 1000, 2000, 4000\,, 7000\}\,. \tag{28}$$

Hence, $\mathcal{N}_{7000}^* \equiv \mathcal{N}^*$. In order to compare the behavior of these NNs, we compare the predicted pmf with the pmf of the actual fluxes using the common test set $\mathcal{P}_{107}$ (see Fig. 9). Furthermore, we compute the JS divergences between these distributions and the ones with pmf $q_j(\mathcal{P}_{107})$, for each $j = 1, \ldots, 4$. Note that, as expected, the accuracy of the predicted pmf deteriorates while decreasing the cardinality of the training and validation set; nonetheless, a quite good similarity between the pmfs is obtained also for moderate values of t (see Table 7).

Encouraged by the good agreement between NNs predictions and actual DFNs simulations, we apply Monte Carlo method (MC) for first order moment estimation, comparing the results obtained with actual fluxes and with predicted ones, for each $t \in \{500, 1000, 2000, 4000\,, 7000\}$. The results obtained for fractures $\mathcal{F}_{33}$ and $\mathcal{F}_{105}$ are reported in Fig. 10, and they are in good agreement with the behavior highlighted in Fig. 9. Note that the two fractures are those for which the divergence values are—for fixed t-approximately the largest and the smallest, respectively (see Table 7). As far as the mean value is concerned, we note that, for fracture $\mathcal{F}_{33}$, $t = 500$ and $t = 1000$ are not enough to obtain a good approximation of the mean value using MC, as the approximation obtained with the predicted fluxes is not close to the one obtained

Table 7 DFN107

t	$j=1$ $(\mathcal{F}_8)$	$j=2$ $(\mathcal{F}_{33})$	$j=3$ $(\mathcal{F}_{62})$	$j=4$ $(\mathcal{F}_{105})$
500	0.1540	0.2357	0.1994	0.1205
1000	0.0875	0.1793	0.0812	0.0524
2000	0.0246	0.0505	0.0583	0.0264
4000	0.0146	0.0223	0.0145	0.0047
7000	0.0054	0.0050	0.0048	0.0021

Jensen–Shannon divergences between distributions of actual fluxes in $\mathcal{P}_{107}$ ($q_j(\mathcal{P}_{107})$) and distributions of $\mathcal{N}_t^*$ predicted fluxes on inputs of $\mathcal{P}_{107}$ ($\widehat{q}_j(\mathcal{P}_{107}\,;\mathcal{N}_t^*)$) for several values of the training and validation set size t

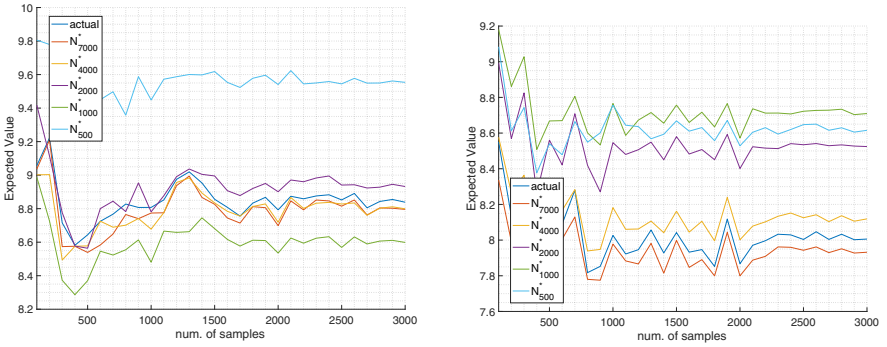

Fig. 10 DFN107. Monte Carlo mean value estimates of the exiting flux versus number of samples, using actual DFN simulations and $\mathcal{N}_t^*$ predictions, for several values of the training and validation set size t. Left: fracture $\mathcal{F}_{33}$; right: fracture $\mathcal{F}_{105}$

using actual fluxes; on the other hand, training the NN with $t = 2000$ already yields an estimate of the mean value which is quite close to the one obtained with MC on actual fluxes (the difference is approximately settled to 1%).

We end this section comparing computing times needed to apply MC method with real simulations and to apply MC with the predicted fluxes. All the simulations, the training of the NNs and the flux predictions using NNs have been computed on a workstation with two AMD Opteron Processors, Interlagos type, 12 cores, Ram 32 GB; the simulations have been performed using FEM with approximately 100 mesh elements on each fracture, and a stopping relative tolerance for the conjugate gradient method equal to 10^{-7}. Using these parameters, the average computing time of a single DFN simulation is approximately 5.5 seconds, whereas the average computing time for a prediction with $\mathcal{N}_t^*$ is approximately $0.137 \cdot 10^{-3}$ seconds; namely, as an average a prediction is approximately $4 \cdot 10^4$ times faster than a real simulation

As far as the training time is concerned, it is partially dependent on the cardinality of the training set and on the parameters that characterize the NN (see Sect. 4.3 and Fig. 6). In Table 8 we show the computing time needed for the training phase of $\mathcal{N}_t^*$, for each $t \in \{500, 1000, 2000, 4000, 7000\}$. Note that the training time abruptly increases when passing from $t = 2000$ to $t = 4000$. This behavior is probably due

Table 8 DFN107

t	Avg. time for $\mathcal{T} \cup \mathcal{V}$ creation	Training time	Total time
500	00:45:50	00:02:11	00:48:01
1000	01:31:40	00:04:54	01:36:34
2000	03:03:20	00:08:40	03:12:00
4000	06:06:40	00:53:06	06:59:46
7000	10:41:40	00:56:53	11:38:33

Total computing time for training $\mathcal{N}_t^*$ for several values of the training and validation set size t (time expressed as hh:mm:ss)

to the presence of a larger number of outliers in the training set, which are therefore more likely to be selected during the mini-batch selection for gradient evaluation (see Sect. 3.1); this results in a slower descent during the training optimization that hinders the early stopping regularization method to work well (considering the patience parameter $p^* = 150$).

As a whole, we observe that most of the computing time has been spent running DFN simulations for training and validation sets creation, being the training time typically a small fraction of the set creation time (approximately $1/6$ in the worst case, but in the other cases ranging from 0.04 to 0.09). Note that the total training cost is considerably dampened if a single NN is trained, and then applied in several frameworks, as suggested from the analysis in Sect. 4.4.2.

From the observations made in this section, we conclude that the use of NNs for UQ is promising, considering that a fine tuning of the NN architecture and hyper-parameters would improve effectiveness of the NNs.

6 Robustness with respect to network size

In this section we highlight the robustness of the method showing the results of its application to other networks with different number of total fractures and of outflow fractures. Namely, we consider three additional networks randomly sampled according to the same lines used for DFN107, but with a larger number of fractures. The three new networks, after connectivity analysis, are made of 158, 202 and 395 fractures, and are labeled DFN158, DFN202 and DFN395, respectively. In Table 9 we report, for each network, data concerning the number of traces per fracture and the trace length. The distribution are similar as the networks are obtained starting from the same distribution, nonetheless they represent quite general and realistic configurations.

The number of outflow fractures of DFN158, DFN202 and DFN395 is $M = 7$, $M = 14$ and $M = 13$, respectively; as already done for DFN107, we focus the analysis on a subset of outflow fractures carrying the largest exit flow mean values, setting therefore $m = 4$ for DFN158 and $m = 7$ for DFN202 and DFN395.

Following the observations made in Sect. 4.4.2, we focus here on the case in which all fractures have a random transmissivity, namely on the case $n = N$, performing on each new test case a grid search method limited to parameters $d = 1, 3$ and $B = 10, 30$.

Table 9 Data on traces' length and number for the DFNs considered

	Trace number			Trace length (m)		
DFN	Min	Aaverage	Max	Min	Average	Max
DFN107	1	3.59	16	1.54	94.52	504.29
DFN158	1	3.23	17	0.25	101.02	333.41
DFN202	1	3.28	18	1.85	85.84	549.31
DFN395	1	3.18	19	0.04	74.75	389.94

Table 10 Mean relative errors $\mathbb{E}[e^r(\mathcal{N}_d^B;\mathcal{P})]$ for several values of depth parameter d and mini-batch size B, for all the considered test cases ($n = N$ fixed)

	DFN107		DFN158		DFN202		DFN395	
	$d=1$	$d=3$	$d=1$	$d=3$	$d=1$	$d=3$	$d=1$	$d=3$
$B=10$	0.0196	**0.0180**	0.0221	0.0262	**0.0116**	0.0175	**0.0234**	0.0258
$B=30$	0.0235	0.0221	**0.0212**	0.0234	0.0129	0.0161	0.0245	0.0260

For each DFN the smallest errors are in bold face

Therefore, for the ease of notation, we drop in this section any dependency on n from the symbols.

6.1 DFN158: regression results and performance comparison

Following the lines of Sect. 4.3, we perform a preliminary regression analysis for DFN158 with respect to four NNs,

$$\mathcal{N}_d^B\,, \qquad \forall\, d = 1, 3\,, \quad \forall\, B = 10, 30\,, \tag{29}$$

and with respect to a dataset $\mathcal{D}$ given by 10,000 simulations that is split in a test set $\mathcal{P}$, a training set $\mathcal{T}$ and a validation set $\mathcal{V}$; the number of elements of each set is chosen according to the general description of Sect. 3.2.2, hence we have $|\mathcal{P}| = P = 3000$, $|\mathcal{T}| = T = 5600$ and $|\mathcal{V}| = V = 1400$.

In Table 10 we report the mean relative errors $\mathbb{E}[e^r(\mathcal{N}_d^B;\mathcal{P})]$ for all the outflow fractures. For the sake of comparison, we also report in the table the same errors obtained for DFN107. Also for DFN158, very small average relative errors are obtained.

Note that the behavior in terms of dependence on d and B is different with respect to DFN158, as the error increases with respect to B, for fixed d, and with respect to d, for fixed B. Nevertheless, in all the cases we observe rather small differences in the error, while varying d from 1 to 3. This is a quite interesting point, as in large realistic DFNs, made of thousands of fractures, a large depth would result in a very demanding training, by the computational point of view, due to the extremely large number of weights to be determined. Indeed, fully-connected layers are characterized by n^2 weights; in a network for vector-valued regression with depth d and m outputs

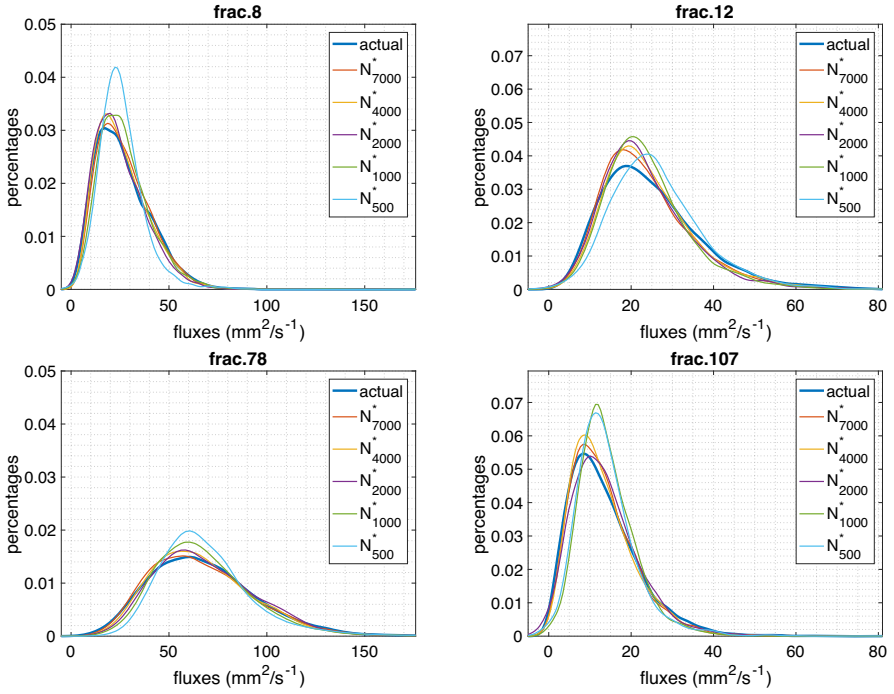

Fig. 11 DFN158. Pmf of actual fluxes in $\mathcal{P}$ ($q_j(\mathcal{P})$) compared with the corresponding pmf $\widehat{q}_j(\mathcal{P}; \mathcal{N}_t^*)$ obtained from $\mathcal{N}_t^*$ flux predictions on inputs of $\mathcal{P}$, for several values of the training and validation set size t

the number of weigths would be approximately $m \cdot d \cdot n^2$. Then, the possibility to stick to a small depth facilitates the training also in view of very large DFNs.

Following such remarks, we fix $d = 1$ and set for DFN158 $\mathcal{N}^* = \mathcal{N}_1^{30}$, namely the NN with average best performances. We repeat the analysis already performed in Sect. 5 for DFN107, comparing the distributions of actual fluxes on inputs of $\mathcal{P}$ with the corresponding distributions of predicted fluxes obtained with $\mathcal{N}_t^*$, namely the NN $\mathcal{N}^*$ trained with sets $\mathcal{T} \cup \mathcal{V}$ with increasing cardinality $t = 500\,, 1000\,, 2000\,, 4000\,, 7000$ (see Fig. 11). The values of the Jensen–Shannon divergences are reported in Table 11.

Comparing Fig. 11 and Table 11 to Fig. 9 and Table 7, respectively, a quite similar behavior of the distributions is observed; in particular, very good approximation results are obtained, and the values of D_{JS} in general decrease with t, with few exceptions.

As far as the training times of NNs $\mathcal{N}_t^*$ are concerned, they are reported in Table 11 and they appear to be quite small, especially if compared to those for DFN107 reported in Table 8. The large difference between the training times in DFN107 and DFN158 is mainly related to the different depth parameter d, since it characterizes the number of weights in the NN, and the different time of action of the early stopping method.

Table 11 DFN158

t	$j=1$ ($\mathcal{F}_8$)	$j=2$ ($\mathcal{F}_{12}$)	$j=3$ ($\mathcal{F}_{78}$)	$j=4$ ($\mathcal{F}_{107}$)	Training time
500	0.0802	0.0322	0.0759	0.0657	00:00:13
1000	0.0228	0.0293	0.0240	0.0881	00:00:21
2000	0.0092	0.0189	0.0081	0.0072	00:00:31
4000	0.0095	0.0135	0.0061	0.0117	00:01:01
7000	0.0058	0.0101	0.0025	0.0067	00:02:28

Jensen–Shannon divergences between distributions of actual fluxes in $\mathcal{P}$ ($q_j(\mathcal{P})$) and distributions of $\mathcal{N}_t^*$ flux predictions on inputs of $\mathcal{P}$ ($\widehat{q}_j(\mathcal{P}\,;\mathcal{N}_t^*)$) for several values of the training and validation set size t. Last column: training times (time expressed as hh:mm:ss)

Table 12 DFN202

t	$j=1$ ($\mathcal{F}_8$)	$j=2$ ($\mathcal{F}_{31}$)	$j=3$ ($\mathcal{F}_{61}$)	$j=4$ ($\mathcal{F}_{73}$)	$j=5$ ($\mathcal{F}_{162}$)	$j=6$ ($\mathcal{F}_{173}$)	$j=7$ ($\mathcal{F}_{176}$)	Training time
500	0.0391	0.1191	0.1681	0.1197	0.0396	0.0385	0.1419	00:00:31
1000	0.0818	0.0684	0.0610	0.1040	0.1372	0.1154	0.0836	00:01:02
2000	0.0340	0.1344	0.0738	0.0495	0.0296	0.0294	0.1859	00:02:07
4000	0.0055	0.0214	0.0206	0.0221	0.0144	0.0233	0.0247	00:04:09
7000	0.0073	0.0085	0.0082	0.0055	0.0060	0.0053	0.0132	00:13:36

Jensen–Shannon divergences between distributions of actual fluxes in $\mathcal{P}$ ($q_j(\mathcal{P})$) and distributions of $\mathcal{N}_t^*$ flux predictions on inputs of $\mathcal{P}$ ($\widehat{q}_j(\mathcal{P}\,;\mathcal{N}_t^*)$) for several values of the training and validation set size t. Last column: training times (time expressed as hh:mm:ss)

6.2 DFN202: regression results and performance comparison

The same analysis performed on DFN158 is here repeated for DFN202. The main feature of DFN202, besides doubling the number of fractures of DFN107, is to have a larger number of outflow fractures under investigation, namely $m = 7$. The mean relative errors obtained are reported in Table 10. The smaller error is in this case attained for $d = 1$ and $B = 10$, thus we set $\mathcal{N}^* := \mathcal{N}_1^{10}$. The results obtained training the networks $\mathcal{N}_t^*$ on sets $\mathcal{T} \cup \mathcal{V}$ with increasing cardinality are summarized in Fig. 12 and Table 12. For space reasons, we report in Fig. 12 the pmf of actual and predicted fluxes for four fractures only. The behavior on the omitted ones is similar. Again, good approximation behavior is attained, with D_{JS} values generally decreasing with respect to t, and brief training times also for the largest t values; it is worth noting that the larger value of $m = 7$ does not introduce significant differences in the quality of regression results obtained with the NN, if compared to cases DFN107 and DFN158 in which $m = 4$.

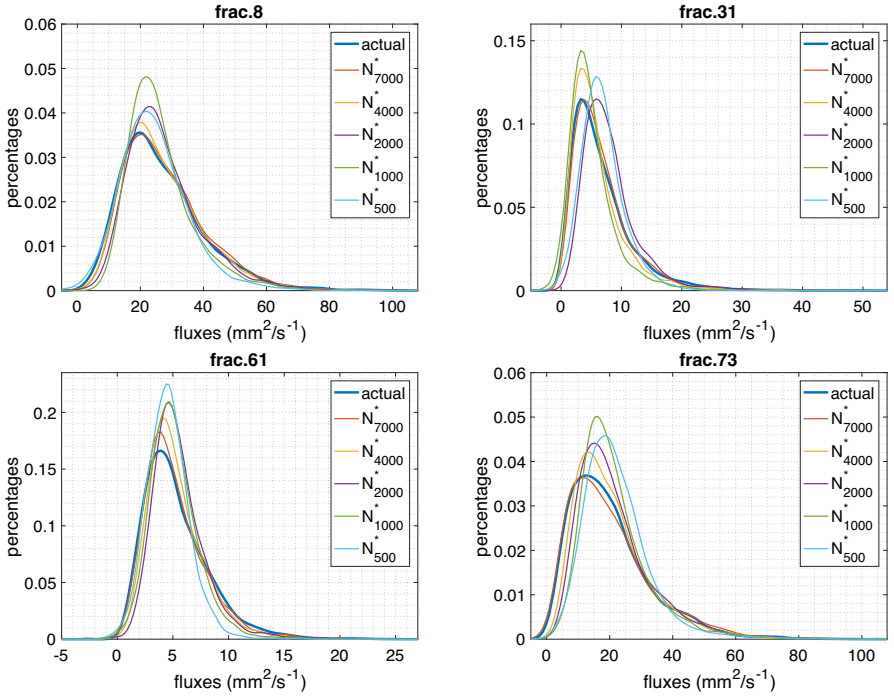

Fig. 12 DFN202. Pmf of actual fluxes in $\mathcal{P}$ ($q_j(\mathcal{P})$) compared with the corresponding pmf $\widehat{q}_j(\mathcal{P}; \mathcal{N}_t^*)$ obtained from $\mathcal{N}_t^*$ flux predictions on inputs of $\mathcal{P}$, for several values of the training and validation set size t

6.3 DFN395: regression results and performance comparison

We end this section with a similar analysis for DFN395. For this DFN, the number of fractures considerably increases, while the analyzed outflow fractures are in the same number as the DFN202 case, that is $m = 7$. The mean relative errors obtained on DFN395 are again reported in Table 10; looking at the values reported in the table, the architecture with smaller error is the one characterized by $d = 1$ and $B = 10$; then we set $\mathcal{N}^* := \mathcal{N}_1^{10}$. The results obtained training the networks $\mathcal{N}_t^*$ on sets $\mathcal{T} \cup \mathcal{V}$ with increasing cardinality, summarized in Fig. 13 and Table 13, show again good results but a general worsening of the distribution approximation with respect to the previous cases. Indeed, looking at Jensen–Shannon divergence values in Table 13, the general decreasing behavior with respect to t is conserved but the divergence values, for each fixed t, are generally higher (in general at least one order of magnitude) with respect to those reported in Tables 7, 11 and 12; the reason for these higher values relies in the so-called "curse of dimensionality" (Goodfellow et al. 2016, chapter 5.11.1), since an increment of the transmissivity space dimension causes an exponential increment of possible combinations which can be obtained during the random generation, making it more difficult to span the domain. Despite these observations, regression results obtained are still good, as seen from Table 10. Finally, we look at the training times

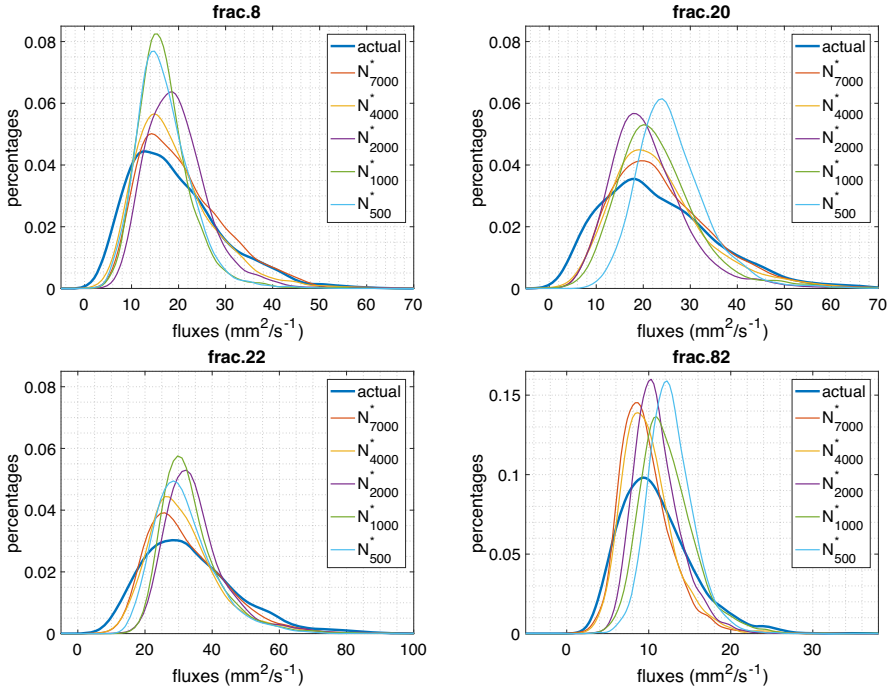

Fig. 13 DFN395. Pmf of actual fluxes in $\mathcal{P}$ ($q_j(\mathcal{P})$) compared with the corresponding pmf $\widehat{q}_j(\mathcal{P}; \mathcal{N}_t^*)$ obtained from $\mathcal{N}_t^*$ flux predictions on inputs of $\mathcal{P}$, for several values of the training and validation set size t

Table 13 DFN395

t	$j=1$ ($\mathcal{F}_8$)	$j=2$ ($\mathcal{F}_{20}$)	$j=3$ ($\mathcal{F}_{22}$)	$j=4$ ($\mathcal{F}_{82}$)	$j=5$ ($\mathcal{F}_{112}$)	$j=6$ ($\mathcal{F}_{305}$)	$j=7$ ($\mathcal{F}_{311}$)	Training time
500	0.2365	0.5372	0.2191	0.5037	2.2118	0.4591	0.2313	00:02:04
1000	0.2469	0.1499	0.3860	0.2430	0.3819	0.1531	0.2707	00:04:03
2000	0.2087	0.1539	0.3261	0.2244	0.1271	0.2159	0.1984	00:06:37
4000	0.0469	0.0618	0.0835	0.0901	0.0692	0.1105	0.0394	00:08:55
7000	0.0617	0.0581	0.0484	0.1179	0.1072	0.02177	0.0425	00:34:26

Jensen–Shannon divergences between distributions of actual fluxes in $\mathcal{P}$ ($q_j(\mathcal{P})$) and distributions of $\mathcal{N}_t^*$ flux predictions on inputs of $\mathcal{P}$ ($\widehat{q}_j(\mathcal{P}; \mathcal{N}_t^*)$), for several values of the training and validation set size t. Last column: training times (time expressed as hh:mm:ss)

for NNs $\mathcal{N}_t^*$ in Table 13; very short training times are seen, in according with the small depth of the networks ($d = 1$), with the only exception of $t = 7000$ caused by a late action of the early-stopping regularization method.

7 Conclusions

We have presented a novel model reduction approach for a massive number of DFN simulations, focusing on its application in uncertainty quantification analyses of some selected quantities of interest. The method involves vector valued regression of the QoIs performed by artificial neural networks.

Given a DFN with fixed geometry, we trained and tested several NNs by varying the number of fractures with random transmissivity, the depth of NNs and the mini-batch size; the target was to predict outflows of some boundary fractures of the given DFN.

Performances of these NNs have been measured and, using a grid search, the network $\mathcal{N}^*$ with best performance has been selected.

A more detailed analysis has been done for $\mathcal{N}^*$. We compared discrete approximations of probability density functions of predicted and actual fluxes using suitable divergence measures for probability distributions, showing a very high similarity between them.

The same distribution comparison, for evaluating performance of $\mathcal{N}^*$, has been performed with respect to other sets of data, in particular on those sets generated from inputs having a fixed transmissivity value for some given fractures; in this situation $\mathcal{N}^*$ performed very well, since it has been able to predict fluxes even if its training set was generated from inputs with no fixed values for any fracture transmissivity.

We have also analyzed the sensitivity with respect to the training set size, and we have shown the viability of the approach to produce predicted fluxes to be used in the framework of Monte Carlo method for computing statistics of the given quantity of interest. Comparing these results with those obtained using actual DFN simulations, we observed that moments computed using NNs are relatively near to values of actual moments, also for NNs trained on small training sets.

In order to show robustness with respect to the number of overall fractures, and the number of outflow fractures, the analysis has been then repeated for other DFNs with different fixed geometries, confirming the ability of the presented method in predicting fluxes essentially independently of the total number of fractures and of the number of outflow fractures.

In conclusion, all results shown in this paper confirm the viability of NNs as possible model reduction tools for DFN flow simulations, usable for example in the framework of UQ analyses, due to the good accuracy and extremely fast evaluation time. These encouraging results suggest a wider investigation, in view of a more general application of learning algorithms for UQ in DFN problems.

Acknowledgements Research performed in the framework of the Italian MIUR Award "Dipartimento di Eccellenza 2018-2022" to the Department of Mathematical Sciences, Politecnico di Torino, CUP: E11G18000350001. The research leading to these results has also been partially supported by Italian MIUR PRIN Projects 201752HKH8_003 and 201744KLJL_004, by INdAM-GNCS and by the SmartData@PoliTO center for Big Data and Machine Learning technologies. F.V. acknowledges partial support from Intesa Sanpaolo Innovation Center. The funder had no role in study design, data collection, and analysis, decision to publish, or preparation of the manuscript.

Funding Open access funding provided by Politecnico di Torino within the CRUI-CARE Agreement.

References

Abadi, M., et al.: TensorFlow: large-scale machine learning on heterogeneous systems (2015)
Adler, P.M.: Fractures and Fracture Networks. Kluwer Academic, Dordrecht (1999)
Amari, S.I., Barndorff-Nielsen, O.E., Kass, R.E., Lauritzen, S.L., Rao, C.R.: Differential geometry in statistical inference. Lect. Notes Monogr. Ser. **10**, (1987)
Benedetto, M.F., Berrone, S., Pieraccini, S., Scialò, S.: The virtual element method for discrete fracture network simulations. Comput. Methods Appl. Mech. Eng. **280**, 135–156 (2014)
Berrone, S., Pieraccini, S., Scialò, S.: A PDE-constrained optimization formulation for discrete fracture network flows. SIAM J. Sci. Comput. **35**(2), B487–B510 (2013a)
Berrone, S., Pieraccini, S., Scialò, S.: On simulations of discrete fracture network flows with an optimization-based extended finite element method. SIAM J. Sci. Comput. **35**(2), A908–A935 (2013b)
Berrone, S., Pieraccini, S., Scialò, S.: An optimization approach for large scale simulations of discrete fracture network flows. J. Comput. Phys. **256**, 838–853 (2014)
Berrone, S., Pieraccini, S., Scialò, S., Vicini, F.: A parallel solver for large scale DFN flow simulations. SIAM J. Sci. Comput. **37**(3), C285–C306 (2015)
Berrone, S., Pieraccini, S., Scialò, S.: Towards effective flow simulations in realistic discrete fracture networks. J. Comput. Phys. **310**, 181–201 (2016a)
Berrone, S., Borio, A., Scialò, S.: A posteriori error estimate for a PDE-constrained optimization formulation for the flow in DFNs. SIAM J. Numer. Anal. **54**(1), 242–261 (2016b)
Berrone, S., Pieraccini, S., Scialò, S.: Non-stationary transport phenomena in networks of fractures: effective simulations and stochastic analysis. Comput. Methods Appl. Mech. Eng. **315**, 1098–1112 (2017)
Berrone, S., Canuto, C., Pieraccini, S., Scialò, S.: Uncertainty quantification in discrete fracture network models: Stochastic geometry. Water Resour. Res. **54**(2), 1338–1352 (2018)
Berrone, S., D'Auria, A., Vicini, F.: Fast and robust flow simulations in discrete fracture networks with GPGPUs. Int. J. Geomath. **10**, 8 (2019)
Bishop, C.M.: Pattern Recognition and Machine Learning (Information Science and Statistics). Springer, Berlin (2006)
Cammarata, G., Fidelibus, C., Cravero, M., Barla, G.: The hydro-mechanically coupled response of rock fractures. Rock Mech. Rock Eng. **40**(1), 41–61 (2007)
Canuto, C., Pieraccini, S., Xiu, D.: Uncertainty quantification of discontinuous outputs via a non-intrusive bifidelity strategy. J. Comput. Phys. **398**, 108885 (2019)
Chan, S., Elsheikh, A.H.: A machine learning approach for efficient uncertainty quantification using multiscale methods. J. Comput. Phys. **354**, 493–511 (2018)
Chollet, F., et al.: Keras (2015). https://keras.io
de Dreuzy, J.R., Pichot, G., Poirriez, B., Erhel, J.: Synthetic benchmark for modeling flow in 3D fractured media. Comput. Geosci. **50**, 59–71 (2013)
Dershowitz, W.S., Fidelibus, C.: Derivation of equivalent pipe networks analogues for three-dimensional discrete fracture networks by the boundary element method. Water Resour. Res. **35**, 2685–2691 (1999)
Fidelibus, C., Cammarata, G., Cravero, M.: Hydraulic characterization of fractured rocks. In: Abbie, M., Bedford, J.S. (eds.) Rock Mechanics: New Research. Nova Science Publishers Inc., New York (2009)
Fumagalli, A., Scotti, A.: A numerical method for two-phase flow in fractured porous media with non-matching grids. Adv. Water Resour. **62**, 454–464 (2013)
Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016). http://www.deeplearningbook.org
Hebb, D.O.: The Organization of Behaviour, New York (1949)

Hyman, J.D., Gable, C.W., Painter, S.L., Makedonska, N.: Conforming Delaunay triangulation of stochastically generated three dimensional discrete fracture networks: a feature rejection algorithm for meshing strategy. SIAM J. Sci. Comput. **36**, A1871–A1894 (2014)

Hyman, J.D., Aldrich, G., Viswanathan, H., Makedonska, N., Karra, S.: Fracture size and transmissivity correlations: Implications for transport simulations in sparse three-dimensional discrete fracture networks following a truncated power law distribution of fracture size. Water Resour. Res. **52**(8), 6472–6489 (2016)

Hyman, J.D., Hagberg, A., Srinivasan, G., Mohd-Yusof, J., Viswanathan, H.: Predictions of first passage times in sparse discrete fracture networks using graph-based reductions. Phys. Rev. E **96**, 013304 (2017)

Jaffré, J., Roberts, J.E.: Modeling flow in porous media with fractures; discrete fracture models with matrix-fracture exchange. Numer. Anal. Appl. **5**(2), 162–167 (2012)

Karimi-Fard, M., Durlofsky, L.J.: Unstructured adaptive mesh refinement for flow in heterogeneous porous media. In: ECMOR XIV-14th European Conference on the Mathematics of Oil Recovery (2014)

Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. CoRR. abs/1412.6980 (2014)

Kullback, S.: Information Theory and Statistics. Dover Publications, Mineola (1968)

Kullback, S., Leibler, R.A.: On information and sufficiency. Ann. Math. Stat. **22**(1), 79–86 (1951)

Lin, J.: Divergence measures based on the Shannon entropy. IEEE Trans. Inf. Theory **37**(1), 145–151 (1991)

MacKay, D.J.C.: Information Theory. Inference and Learning Algorithms. Cambridge University Press, New York (2002)

McCulloch, W.S., Pitts, W.: A logical calculus of the ideas immanent in nervous activity. Bull. Math. Biophys. **5**, 115–133 (1943)

McKinney, W.: Data structures for statistical computing in python. In: van der Walt, S., Millman, J. (eds.) Proceedings of the 9th Python in Science Conference, pp. 51 – 56 (2010)

Nœtinger, B.: A quasi steady state method for solving transient Darcy flow in complex 3D fractured networks accounting for matrix to fracture flow. J. Comput. Phys. **283**, 205–223 (2015)

Nœtinger, B., Jarrige, N.: A quasi steady state method for solving transient Darcy flow in complex 3D fractured networks. J. Comput. Phys. **231**(1), 23–38 (2012)

Pedregosa, F., et al.: Scikit-learn: machine learning in python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)

Pichot, G., Poirriez, B., Erhel, J., de Dreuzy, J.R.: A Mortar BDD method for solving flow in stochastic discrete fracture networks. In: Domain Decomposition Methods in Science and Engineering XXI, Lecture Notes in Computational Science and Engineering, Springer, pp. 99–112 (2014)

Pichot, G., Erhel, J., de Dreuzy, J.: A mixed hybrid mortar method for solving flow in discrete fracture networks. Appl. Anal. **89**, 1629–643 (2010)

Pichot, G., Erhel, J., de Dreuzy, J.: A generalized mixed hybrid mortar method for solving flow in stochastic discrete fracture networks. SIAM J. Sci. Comput. **34**, B86–B105 (2012)

Raissi, M., Perdikaris, P., Karniadakis, G.E.: Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J. Comput. Phys. **378**, 686–707 (2019)

Rosenblatt, F.: The perceptron: a probabilistic model for information storage and organization in the brain. Psychol. Rev. 65–386 (1958)

Srinivasan, G., Hyman, J.D., Osthus, D.A., Moore, B.A., O'Malley, D., Karra, S., Rougier, E., Hagberg, A.A., Hunter, A., Viswanathan, H.S.: Quantifying topological uncertainty in fractured systems using graph theory and machine learning. Sci. Rep. 11665 (2018)

Srinivasan, S., Karra, S., Hyman, J., Viswanathan, H., Srinivasan, G.: Model reduction for fractured porous media: a machine learning approach for identifying main flow pathways. Comput. Geosci. (2019). https://doi.org/10.1007/s10596-019-9811-7

Svensk Kärnbränslehantering AB. Data report for the safety assessment, SR-site. Technical Report TR-10-52, SKB, Stockholm, Sweden (2010)

Tripathy, R.K., Bilionis, I.: Deep uq: learning deep neural network surrogate models for high dimensional uncertainty quantification. J. Comput. Phys. **375**, 565–588 (2018)