University of Stuttgart
Institute for Modelling Hydraulic and Environmental Systems
Department of Stochastic Simulation and Safety Research for Hydrosystems

Matthias Gültig

Learning a Contaminant Transport Solver for PFAS with Finite Volume Neural Networks

Bachelor's Thesis

Learning a Contaminant Transport Solver for PFAS with Finite Volume Neural Networks

Submitted by

Matthias Gültig

Matriculation Number: 3469020
Bachelor's Thesis Number: 154

Examiners: Prof. Dr.-Ing. Wolfgang Nowak
Apl. Prof. Dr.-Ing. Sergey Oladyshkin

Supervisor: M. Sc. Nils Wildt

Institute for Modeling Hydraulic and Environmental Systems
Department of Stochastic Simulation and Safety Research for Hydrosystems

Stuttgart, December 20th, 2022

Author Declaration

I declare that I have developed and written the enclosed thesis completely by myself and that I have not used sources or means without declaration in the text. Any thoughts from others or literal quotations are clearly marked.

The thesis was not used in the same or in a similar version to achieve an academic grading or is being published elsewhere.

The enclosed electronic version is identical to the printed versions.

Date	Signature

Contents

1 Propaedeuticum: Fundamental Considerations of Deep Neural Networks

To form a basis for the thesis, a toy problem is used to exemplify the conceptual idea of the functioning of an Artificial Neural Network (NN). To understand the mathematical background especially probabilistic considerations are introduced. For the technical background the use of a Python library for machine learning called PyTorch [1] is presented.

1.1 Toy Problem

The example multivariate data set [2] is taken from Python's Scikit-learn package [3]. It contains chemical analyses of two types of wine of 130 wines grown in the same region in Italy. In total 13 components were analyzed that appeared in both wines, such as the alcohol or magnesium content. In the following, these are called features. Originally, the data set contained three wine varieties, but since the toy problem was to be kept as simple as possible, the data for the third wine type was removed. The toy problem is to develop a NN that can match a wine sample to the correct variety based on the 13 features.

The distribution of the feature *alcohol content* of all samples is exemplary visualized in Fig. 1.1: Samples of the variety labeled with 1 have a mean of 12.3 vol% and standard deviation 0.5 vol%, while the samples of the wine with target 0 have a significantly higher mean value (13.8 vol%) and scatter less.

Looking at e.g. the bivariate data set of the features alcohol and magnesium content, no linear correlation is apparent (Fig. 1.2).

1.2 Definitions and Mathematical Background

In order to embed the operating principle of a NN with more than one hidden layer, which is a so-called Deep Neural Network (DNN), in a mathematical and, in particular, probabilistic context, a basic mathematical notion must first be defined, which is inspired by Goodfellow et al. [4].

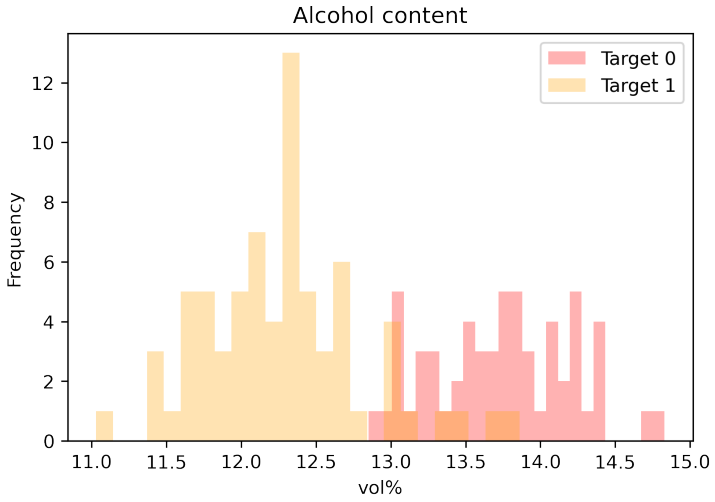

Figure 1.1: Absolute alcohol frequencies of wines in the wine data set.

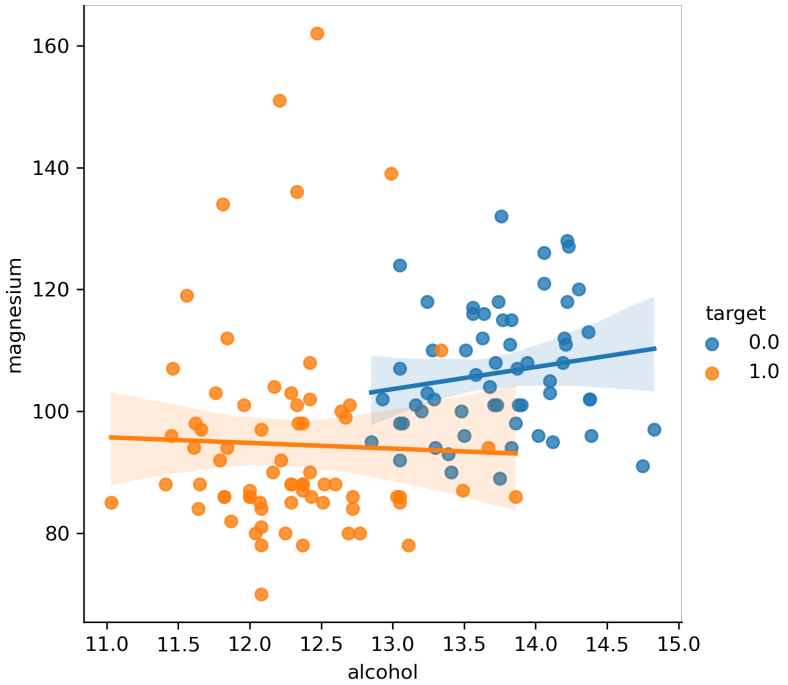

Figure 1.2: Bivariate considerations between alcohol content [vol%] and magnesium content [mg]: Linear regression with 95% confidence interval.

Fundamentals of Probability Theory
A random variable (RV) X is formally a measurable mapping that assigns elements from a probability space to elements of a measurable space. Where a measurable space, by definition consists of a basic set over which a σ - algebra is defined. A probability space is a measure space which is, compared to a measurable space provided with a probability measure. Due to inverse image and probability measure properties, a measure is also defined on the image of X, which makes the measurable image space to a measure space.

According to the definition of the counting densities, each discrete probability measure has a probability density with respect to the counting measure, this is also known as probability mass function (PMF) and provides possible values and the associated probabilities of a discrete RV.

Probability measures whose distribution functions are continuous, are absolutely continuous with respect to the Lebesgue measure, therefore, according to the Radon-Nikodym theorem, there exists a so called probability density function (PDF). The Lebesgue integral (and with corresponding theorems also the Riemann integral) of this function over a set of the probability space corresponds to the probability of the RV to assume values in this set.

In order not to go beyond the scope of this Propaedeuticum, the following simplifying assumptions for RV are made:

1. A RV $\vec{X}$ can either be discrete-valued: $\vec{X} \in \{\vec{x_1}, \vec{x_2}, ...\}$, or continuous-valued: $\vec{X} \in \mathbb{R}^d$
2. If $\vec{X}$ is discrete-valued, $\vec{X}$ has a PMF P: $\vec{X} \sim P(\vec{X} = \vec{x_i}) = P(x_i) = p_i \geq 0$, where p_i is the probability that $\vec{X} = \vec{x_i}$.
3. If $\vec{X}$ is continuous-valued $\vec{X}$ has a PDF p: $\vec{X} \sim p(\vec{x}) \geq 0$

Due to properties of the probability measure, the sum of all p_i of a RV or the integral over the PDF of a RV equals 1.

Using the Dirac function δ, a PDF $\hat{p}$ for a discrete-valued RV can also be set up:

$$\hat{p}(\vec{x}) = \sum_i p_i \delta(\vec{x} - \vec{x_i}). \tag{1.1}$$

At this place it should be pointed out for completeness that the Dirac function is formally a (regular) distribution, hence a linear, continuous functional, which assigns functions from the space of test functions, e.g. the space of infinitely often differentiable functions with compact support, to a real (or complex) number.

Let $\vec{X} \sim p(\vec{x})$, and $\vec{Y} \sim p(\vec{y})$ be RV with corresponding PDFs then the following can be defined:

Definition 1.1 (Expectation value)**.**

$$\mu := E(\vec{X}) := \int_{-\infty}^{\infty} \vec{x} p(\vec{x}) \mathrm{d}\vec{x} \tag{1.2}$$

Definition 1.2 (Joint distribution)**.** The Joint distribution is the vertically stacked PDF:

$$p(\vec{x}, \vec{y}) := p\left(\begin{pmatrix}\vec{x}\\ \vec{y}\end{pmatrix}\right) \tag{1.3}$$

Definition 1.3 (Marginal distribution)**.** The marginal distribution is the integral of the joint distribution of one RV:

$$p(\vec{x}) = \int p(\vec{x}, \vec{y}) \mathrm{d}\vec{y} \tag{1.4}$$

$$p(\vec{y}) = \int p(\vec{x}, \vec{y}) \mathrm{d}\vec{x} \tag{1.5}$$

Definition 1.4 (Conditional distribution)**.**

$$p(\vec{x}|\vec{y}) := \frac{p(\vec{x}, \vec{y})}{p(\vec{y})} \tag{1.6}$$

$$p(\vec{y}|\vec{x}) := \frac{p(\vec{x}, \vec{y})}{p(\vec{x})} \tag{1.7}$$

To measure the dissimilarity between two PDFs p and q the Kulback-Leibler divergence and cross entropy can be used.

Definition 1.5 (Kullback-Leibler divergence)**.** Let p and q be two PDFs then the Kullback-Leibler divergence (KLD) [5] is defined as:

$$D_{KL}(p \parallel q) := \int p(\vec{x}) \cdot \ln\left(\frac{p(\vec{x})}{q(\vec{x})}\right) \mathrm{d}\vec{x} = E_{\vec{X}\sim p}\left[\ln\left(\frac{p(\vec{x})}{q(\vec{x})}\right)\right], \tag{1.8}$$

where $E_{\vec{X}\sim p}$ is the expectation value according to p.

It can be shown that:

1. $D_{KL}(p \parallel q) \geq 0$
2. $D_{KL}(p \parallel q) = 0 \Leftrightarrow p = q$
3. $D_{KL}(p \parallel q) \neq D_{KL}(q \parallel p)$.

Definition 1.6 (Entropy)**.** The entropy of a probability distribution p is defined as:

$$H(p) := -\int p(\vec{x}) \ln(p(\vec{x})) \mathrm{d}\vec{x} \tag{1.9}$$

Definition 1.7 (Cross entropy)**.** The cross entrpopy (CE) between two different probability distributions p and q is defined as:

$$H(p, q) := -E_{\vec{X}\sim p}\left[\ln(q(\vec{x}))\right] = -\int_{-\infty}^{\infty} p(\vec{x}) \ln(q(\vec{x})) \mathrm{d}\vec{x} \geq 0 \tag{1.10}$$

It holds by definition

$$D_{KL}(p \parallel q) = \int p(\vec{x}) \ln\left(\frac{p(\vec{x})}{q(\vec{x})}\right) \mathrm{d}\vec{x} \overset{\text{Log. rules}}{=} \underbrace{\int p(\vec{x}) \ln(p(\vec{x})) \mathrm{d}\vec{x}}_{=-H(p)} \underbrace{- \int p(\vec{x}) \ln(q(\vec{x})) \mathrm{d}\vec{x}}_{=H(p,q)}. \tag{1.11}$$

With eq. (1.11) and properties of the KLD we have

$$H(p,q) = \underbrace{D_{KL}(p \parallel q)}_{\geq 0} + H(p) \geq H(p) \geq 0. \tag{1.12}$$

For fixed p follows

$$\min_q D_{KL}(p \parallel q) \Leftrightarrow \min_q H(p,q). \tag{1.13}$$

Thus minimizing KLD or CE results the same.

All considerations made for RV with continuous PDFs can also be applied to discrete RV, by changing the integral into a sum.

Probabilistic Framework of Supervised Learning

Definition 1.8 (Bayes rule)**.**

$$\underbrace{p(\vec{y}|\vec{x})}_{posterior} = \underbrace{p(\vec{x}|\vec{y})}_{likelihood} \cdot \frac{\overbrace{p(\vec{y})}^{prior}}{\underbrace{p(\vec{x})}_{evidence}} \tag{1.14}$$

Based on Bayes' theorem, in Bayesian estimation theory with known likelihood, prior, and evidence, a posterior density function (posterior) can be calculated.

In NNs likelihood and prior are unknown. Therefore, we try to approximate posterior $p(\vec{x}|\vec{y})$ by a PDF $q(\vec{x}|\vec{y};\vec{\theta})$. The PDF q is given by the architecture of the NN. Only the parameters $\vec{\theta}$ have to be learned using a training data set and KLD/CE as measure for dissimilarity of p and q. Finding a PDF as close as possible to p turns out into an optimization problem. The goal is to minimize CE by changing $\vec{\theta}$. First the CE should only be dependent on $\vec{\theta}$ and has to be reformulated as

$$\begin{aligned} H(p(\vec{x},\vec{y}), q(\vec{x},\vec{y};\vec{\theta})) &\overset{\text{Def.}}{=} -\int p(\vec{x},\vec{y}) \ln q(\vec{x},\vec{y};\vec{\theta}) \mathrm{d}\vec{x}\mathrm{d}\vec{y} \\ &\overset{(1.7)}{=} -\int p(\vec{x},\vec{y}) \ln q(\vec{y}|\vec{x};\vec{\theta}) \mathrm{d}\vec{x}\mathrm{d}\vec{y} - \underbrace{\int p(\vec{x},\vec{y}) \ln q(\vec{x}) \mathrm{d}\vec{x}\mathrm{d}\vec{y}}_{:=c\in\mathbb{R}} \\ &= \int p(\vec{x},\vec{y})[-\ln q(\vec{y}|\vec{x};\vec{\theta})] \mathrm{d}\vec{x}\mathrm{d}\vec{y} + c. \end{aligned} \tag{1.15}$$

Thus, the joint distribution can be represented according to eq. (1.7) as the product of conditional and marginal distribution, and the second summand in the upper equation is constant because it is independent of $\vec{\theta}$.
In the toy problem one sample consists of 13 features thus $\vec{x} \in \mathbb{R}^{13}$ and

$$\vec{y} \in \left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right\}. \tag{1.16}$$

labels the corresponding wine variety 0 or 1. Let $\vec{e_1}$ stand for variety 0 and $\vec{e_2}$ for wine variety 1. This type of labeling is also referred to as one-hot-coding and will be needed later.

The training data set contains finite samples $\vec{x}$ with corresponding wine variety $\vec{y}$ (supervised learning). Each sample j is uniformly distributed with $p_j = \frac{1}{N}$, where N is the size of the training data set. Thus, the unknown joint distribution $p(\vec{x}, \vec{y})$ is given with eq. (1.1) to

$$\hat{p}(\vec{x}, \vec{y}) = \frac{1}{N} \sum_{j=1}^{N} \delta(\vec{x} - \vec{x}(j), \vec{y} - \vec{y}(j)), \tag{1.17}$$

where the term $\delta(\vec{x} - \vec{x}(j), \vec{y} - \vec{y}(j))$ equals 1 for a corresponding sample because of properties of the Dirac function.
For the minimization problem, it follows with eqs. (1.15) and (1.17)

$$\min_{\vec{\theta}} H(\hat{p}, q) = \int \frac{1}{N} \sum_{j=1}^{N} \delta(\vec{x} - \vec{x}(j), \vec{y} - \vec{y}(j))[-\ln q(\vec{y}|\vec{x}; \vec{\theta})] \mathrm{d}\vec{x} \mathrm{d}\vec{y} + c. \tag{1.18}$$

Since in the integral, due to the Delta function only null sets are left, it gets cancelled. The constant c can be neglected at minimization of $\vec{\theta}$. Thus it holds

$$\begin{aligned} \min_{\vec{\theta}} H(\hat{p}, q) &= \frac{1}{N} \sum_{j=1}^{N} \delta(\vec{x} - \vec{x}(j), \vec{y} - \vec{y}(j))[-\ln q(\vec{y}|\vec{x}; \vec{\theta})] \\ &= \frac{1}{N} \sum_{j=1}^{N} \underbrace{[-\ln q(\vec{y}(j)|\vec{x}(j); \vec{\theta}]}_{=: l(\vec{x}(j), \vec{y}(j); \vec{\theta})}, \end{aligned} \tag{1.19}$$

since the Delta function equals 1 for each sample.

These transformations thus replace the unknown PDF p with concrete training data points. Now, the so-called cost or loss function $L(\vec{\theta})$ can be optimized using a data set

$$L(\vec{\theta}) = \frac{1}{N} \sum_{j=1}^{N} l(\vec{x}(j), \vec{y}(j); \vec{\theta}). \tag{1.20}$$

1.3 Neural Network

According to eq. (1.19), an optimal $\vec{\theta}$ must be found to minimize $H(\hat{p}, q)$. Or in other words, $p(\vec{y}|\vec{x})$ should be approximated by a model $q(\vec{y}|\vec{x};\theta)$. This is realized by creating a model or NN $f(\vec{x}, \vec{\theta})$ learning $\vec{\theta}$ from training data by solving an optimization problem. In general NNs with enough layers and nonlinear activations can be considered as universal function approximators [6] and are able to learn every continuos relation over compact input sets [7].

Preparation of Data
After deleting entries of the third wine type in the data set as described above, target $\vec{y}$ and input vector $\vec{x}$, which are stored together in one 14 dimensional sample in the data set, were separated from each other. This was realized after reading the data set with basic Pandas Dataframe operations [8]. With `train_test_split` from Sklearn the data set can randomly be split into training data and test data. In this function the parameter `train_size` can be used to set the ratio of training data to test data. To avoid tight contour lines due to different orders of magnitude of each feature in the loss function, both training and test data were normalized with functions of Sklearn which improves the optimization. A detailed description of the normalization methods used is given in the documentation [9].

Architecture of a Dense Neural Network
Derived from biological neurons, also in artificial neurons, as the smallest unit of a DNN, information should be changed and passed on under certain conditions. A linear neuron can be described by a function with d input signals $\vec{x} = [x_1, ..., x_d]^T \in \mathbb{R}^d$ with d weights $\vec{w} = [w_1, ..., w_d]^T \in \mathbb{R}^d$ and a bias or offset $b \in \mathbb{R}$. To model nonlinear relationships there can be added an additional activation function $\phi : \mathbb{R} \to \mathbb{R}$ to ultimately provide an output $y_{pred} \in \mathbb{R}$ to be calculated

$$y_{pred} = \phi(\vec{w}^T\vec{x} + b). \tag{1.21}$$

Several neurons are combined in one so-called layer. Each of the d input entries is fully connected with d neurons. $\vec{w} \in \mathbb{R}^{d\times c}$ corresponds to the weight matrix and $\vec{b} = [b_1, ...b_c]^T \in \mathbb{R}^c$ denotes the bias vector, where the layer consists of c neurons. Since each neuron has one $y_{pred} \in \mathbb{R}$, the output of the layer is vector-valued. In a layer there are no interactions between the neurons. The $c \cdot d$ weights and c biases are called parameters of a DNN, which are in the domain of the loss function and correspond to $\vec{\theta}$.

A so called feed forward DNN is the combination of several layers, where the output $\vec{y}_{pred}$ of the previous layer is used as input of the current layer.

For the toy problem, a small NN named `WineDecision` was built using PyTorch. PyTorch is an open source machine learning framework, that is particularly well suited for the creation of NNs due to its autograd system, GPU acceleration and high flexibility because of the usage of dynamic computation graphs [1]. By inheriting PyTorch's

```
1  class WineDecision(nn.Module):
2
3      def __init__(self, n_input_features: int):
4          super(WineDecision, self).__init__()
5          self.linear = nn.Linear(n_input_features, 1)
6
7      def forward(self, x: torch.tensor):
8          # constrain values (0 < y < 1)
9          y_pred = torch.sigmoid(self.linear(x))
10         return y_pred
11
12 # n_features = 13
13 model = WineDecision(n_features)
```

Listing 1.1: Structure of one layer neural network with PyTorch.

`nn.Module`, functions implemented by the library can be used for the NN. In the constructor for this, first the constructor of the `nn.Module` class is called and then one `nn.Linear` layer provided by PyTorch is added to the `WineDecision` NN. It takes a `n_input_features` - dimensional input and maps it to a 1 dimensional output. Thus $d = 13$ and $c = 1$, since only one neuron is considered (Fig. 1.3).
The input vector is first multiplied by the weights, the bias is added and then the *sigmoid* function is applied to it, to finally return a y_{pred}. Depending on whether a forward or backward pass is executed, the derivative of each operation is calculated. In line 13 the model is initialized (Listing 1.1). The *sigmoid* function

$$sig(x) = \frac{1}{1 + e^{-x}}, \tag{1.22}$$

has besides its nonlinearity another important property for the binary-classification toy problem: It only assumes values between 0 and 1. Usually higher order classification problems are realized with so-called softmax activation functions, for a binary-classification problem the simple sigmoid function is sufficient: If y_{pred} takes values ≥ 0.5, the NN decides for wine type 1 otherwise 0.

Loss Calculation
With eq. (1.20) the loss function to be optimized has already been introduced, but to be concretely applicable to the binary-classification toy problem, it still needs to be simplified.

The probability that given a sample $\vec{x}$, the wine belongs to the variety 0 can be written with the notation of eq. (1.16) as

$$p_1 = P(\vec{y}(j) = \vec{e_1}|\vec{x}(j)). \tag{1.23}$$

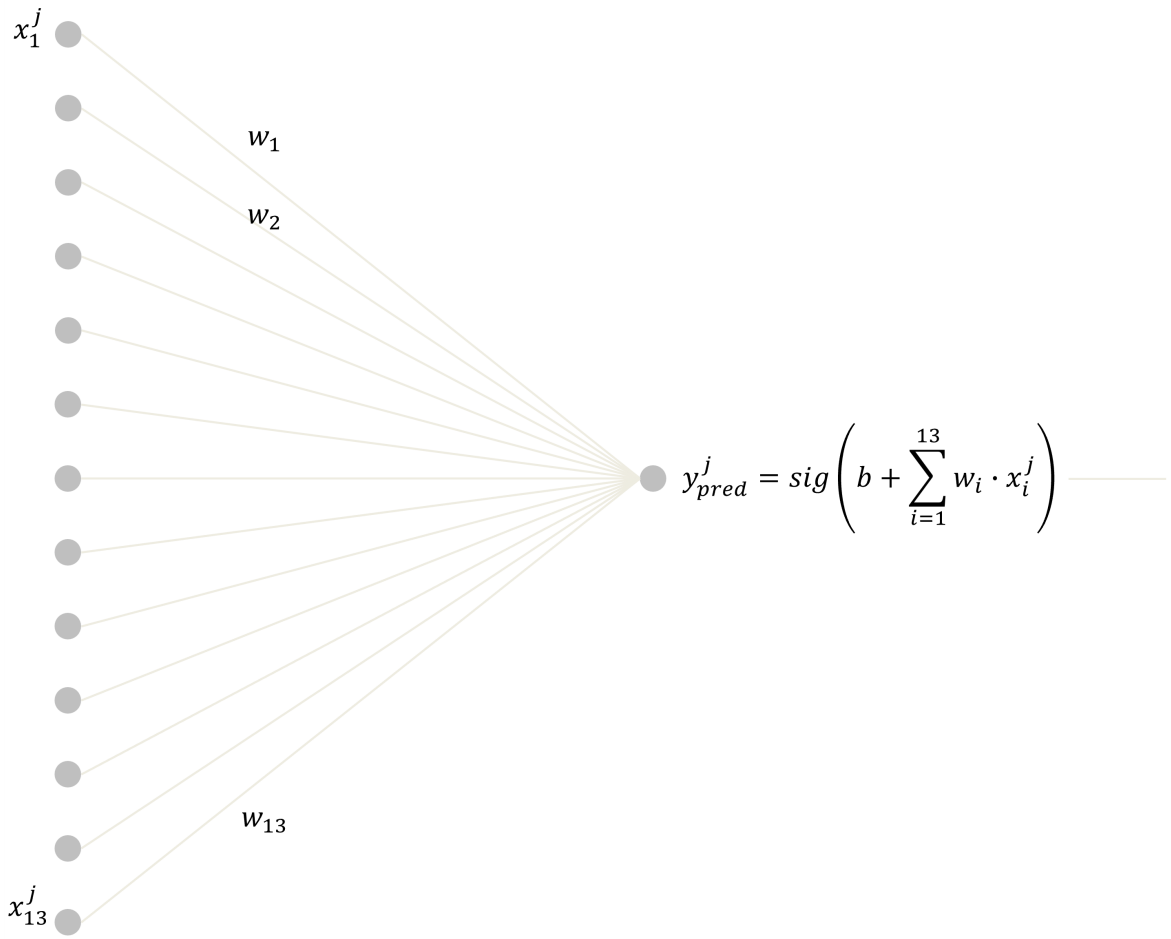

Figure 1.3: Calculations of the small implemented `WineDecision` NN for one specific wine sample j with its 13 features.

```
loss = nn.BCELoss(y_pred, y_Train)
```

Listing 1.2: Usage of PyTorchs defined BCE loss function.

P is a posterior PMF that is unknown. In general, the probability that sample j of $\vec{x}$ belongs to wine variety 0 or 1 can be written as

$$P(\vec{y}(j)|\vec{x}(j)) = p_1(j)^{y_1(j)} \cdot p_2(j)^{y_2(j)}, \tag{1.24}$$

where $y_1(j)$ is the first and $y_2(j)$ the second entry of $\vec{y}(j)$, respectively. Since a fixed wine sample j cannot belong to two wine types at the same time, $y_1 = 0$ if $y_2 = 1$ and vice versa. As mentioned at the beginning, the unknown $p_i(j)$ are approximated by $q_i(j)$ ($i \in \{1, 2\}$) with a PMF Q. While $q_i(j)$ is approximated by a forward pass of the NN $f(\vec{x}(j); \theta)$. Thus holds

$$Q(\vec{y}(j)|\vec{x}(j); \vec{\theta}) = f_1(\vec{x}(j); \vec{\theta})^{y_1(j)} \cdot f_2(\vec{x}(j); \vec{\theta})^{y_2(j)}. \tag{1.25}$$

In the binary classification problem holds

$$f_1(\vec{x}(j), \vec{\theta}) = y_{pred}(\vec{\theta}, j) \qquad f_2(\vec{x}(j), \vec{\theta}) = 1 - y_{pred}(\vec{\theta}, j), \tag{1.26}$$

where $y_{pred}(\vec{\theta}, j)$ is the output of the `WineDecision` NN of one sample j. Then l of eq. (1.20) at a given sample j is given by

$$\begin{aligned}
l(\vec{x}(j), \vec{y}(j); \vec{\theta}) &\overset{(1.19)}{=} -\ln Q(\vec{y}(j)|\vec{x}(j); \vec{\theta}) \\
&\overset{(1.25)}{=} -\ln \Big[f_1(\vec{x}(j); \vec{\theta})^{y_1(j)} \cdot f_2(\vec{x}(j); \vec{\theta})^{y_2(j)} \Big] \\
&\overset{(1.26)}{=} -\ln \Big[y_{pred}(\vec{\theta}, j))^{y_1(j)} \cdot (1 - y_{pred}(\vec{\theta}, j))^{y_2(j)} \Big] \\
&\overset{\text{Log. rules}}{=} -\Big[y_1(j) \ln y_{pred}(\vec{\theta}, j) + y_2(j) \ln (1 - y_{pred}(\vec{\theta}, j)) \Big] \\
&= -\Big[\hat{y}(j) \ln y_{pred}(\vec{\theta}, j) + (1 - \hat{y}(j)) \ln (1 - y_{pred}(\vec{\theta}, j)) \Big],
\end{aligned} \tag{1.27}$$

where in the last transformation y_1 and y_2 can be replaced by $\hat{y} \in \{0, 1\}$ because with the same argumentation as above either y_1 or y_2 equals 1 or 0, respectively for a specific sample j. The loss function to be optimized, which depends exclusively on known parameters, is with eq. (1.20) therefore

$$L(\theta) = \frac{1}{N} \sum_{j=1}^{N} -\Big[\hat{y}(j) \ln y_{pred}(\vec{\theta}, j) + (1 - \hat{y}(j)) \ln (1 - y_{pred}(\vec{\theta}, j)) \Big]. \tag{1.28}$$

For the implementation of this so-called binary cross entropy loss the predefined BCE Loss of PyTorch with the default `reduction = "mean"`, divides the calculated sum by the number of samples N and fulfills the task (Listing 1.2).

Backpropagation and Optimization
The cost function $L(\vec{\theta})$ (in the model problem $L : \mathbb{R}^{13+1} \to \mathbb{R}$, since $\vec{\theta}$ consists of 13 weights and 1 bias) must now be minimized with respect to $\vec{\theta}$. Since often for all $\vec{\theta}$, global analytic formulations for the derivative of L do not exist, numerical optimization algorithms are used.

So-called gradient descent algorithms require the first-order derivative of a function to perform an optimization and are often used in DNN. In principle, they consist of the following update equation

$$\vec{\theta}^{t+1} = \vec{\theta}^{t} - \gamma^t \vec{\nabla} L(\vec{\theta}^{t}), \tag{1.29}$$

where:

1. $t = 0, 1, ...$: iteration index,
2. $\gamma^t > 0$: step size at index t,
3. $\vec{\theta}^0$: initial guess.

γ^t is in general freely selectable, may vary with the iteration index and determines, among other things, the convergence velocity of the optimization algorithm. It can be defined as learning rate `lr` in PyTorch's optimizers (Listing 1.3).

θ^0 is determined randomly by PyTorch. In the model problem, weights and one bias have the default values

$$w_d, b \in \mathcal{U}(-\sqrt{k}, \sqrt{k}), \tag{1.30}$$

where $k = \frac{1}{13}$ and $\mathcal{U}$ describe a uniform distribution. PyTorch offers many possibilities to cleverly choose the initial guess.

In general $\vec{\theta}$ is updated after one entire loop over the training set (epoch). The steps described previously in *Architecture of a DNN* have to be done for each training sample and is performed vectorized in PyTorch. Thus one forward pass through the DNN results in a $y_{pred} \in \mathbb{R}^N$. For the calculation of $\vec{\nabla} L(\vec{\theta}^t)$ the famous backpropagation algorithm can be used [10].

Definition 1.9 (Chain rule)**.** Let $y = f(u)$ be a differentiable function with respect to u and $u = g(x)$ a differentiable function with respect to x then it holds

$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx}. \tag{1.31}$$

Applying the chain rule to the output of the model problem yields

$$\nabla^T L(\vec{\theta}) = \begin{pmatrix} \frac{\partial L}{\partial w_1} \\ \dots \\ \frac{\partial L}{\partial w_{13}} \\ \frac{\partial L}{\partial b} \end{pmatrix}^T = \frac{\partial L}{\partial \vec{\theta}} = \frac{\partial L(\vec{\theta})}{\partial y_{pred}} \cdot \frac{\partial y_{pred}}{\partial u} \cdot \frac{\partial u}{\partial \vec{\theta}}, \tag{1.32}$$

```
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)
```

Listing 1.3: Usage of PyTorchs Stochastic Gradient Descent optimization algorithm.

```
for epoch in range(num_epochs):

    # forward pass
    y_pred = model(X_Train)

    # loss calculation with BCE
    loss = nn.BCELoss()(y_pred, y_Train)

    # backward pass
    loss.backward()

    # updates
    optimizer.step()
    optimizer.zero_grad()
```

Listing 1.4: Training loop in PyTorch.

where $y_{pred} = sig(u)$, $u = \vec{w}^T\vec{x} + b$ and $\vec{\theta} = [w_1, ..., w_{13}, b]$. For one sample the first two factors are scalar-valued, the last one $\left(\frac{\partial u}{\partial \vec{\theta}}\right)$ is a $(1 \times 13 + 1)$ matrix.

After applying an iteration step of the Gradient Descent algorithm (1.29) with a corresponding step γ^t, the parameters are updated in the direction of the negative gradient of L to find optimal parameters with minimum $L(\vec{\theta})$. As termination condition a defined maximum number of epochs was chosen (`num_epochs`).

Often, the so-called Stochastic Gradient Descent (SGD) algorithm takes randomly batches, thus parts of the training data set and calculates the gradients to save computational capacity. Here, again functions of PyTorch can be used (Listing 1.3).

One of many advantages of using PyTorch is the autograd system. For each operation performed during backward pass in the signal chain of a DNN, the gradients are stored to facilitate loss calculation. However, after each training epoch they must be set back to 0 (`optimizer.zero_grad`).

A training loop (Listing 1.4) therefore consists of a forward pass, loss calculation, backward pass, optimization, and resetting the gradients.

```
with torch.no_grad():
    y_pred = model(X_Test)
```

Listing 1.5: Test loop in PyTorch.

1.4 Results

After training, the accuracy of the toy problem can be checked with other samples. For this purpose, no gradients are required, since only a forward pass is performed (Listing 1.5). As already mentioned the accuracy of the `WineDecision` was assessed by rounding the model output to 0 and 1, respectively, counting the number of samples assigned correctly and dividing the result by the total number of test samples.

As an example, a small benchmark was implemented, with the goal of determining the dependence of the accuracy on the number of training iterations and training data set sizes. The model was trained with 10% - 60 % of the data of the total data set for different number of epochs. The measurement of accuracy was always done with 40% remaining test data. To exclude statistical inaccuracies, the benchmark procedure was performed 50 times. Mean and standard deviation were calculated from each of the 50 measurements.

As expected a small number of epochs results in bad accuracy and high standard deviations, the optimization is not yet so far advanced, it is rather the coincidence that decides whether the wine types were correctly assigned. Also a small number of training samples resulted in higher standard deviations, even after 1000 epochs of training, because of overfitting. The model is too much optimized for training data and is not trained for other samples of the test data set (Fig. 1.4).

1.5 Conclusion and Outlook

In this introductory Propaedeuticum, the mathematical background and basic ideas of how NNs work were explained. In particular, the focus was on probabilistic considerations and advantages of using PyTorch to convey the basic idea. Details of results of the toy problem were deliberately not discussed, as they do not contribute to the topic of the actual bachelor thesis. A NN approximates a posterior probability density function using a training data set by optimizing weights. PyTorch offers useful concepts and functions to avoid larger implementation efforts. In order to reproduce this work, code performing the described steps of the toy problem and the benchmark are available on GitHub [11].

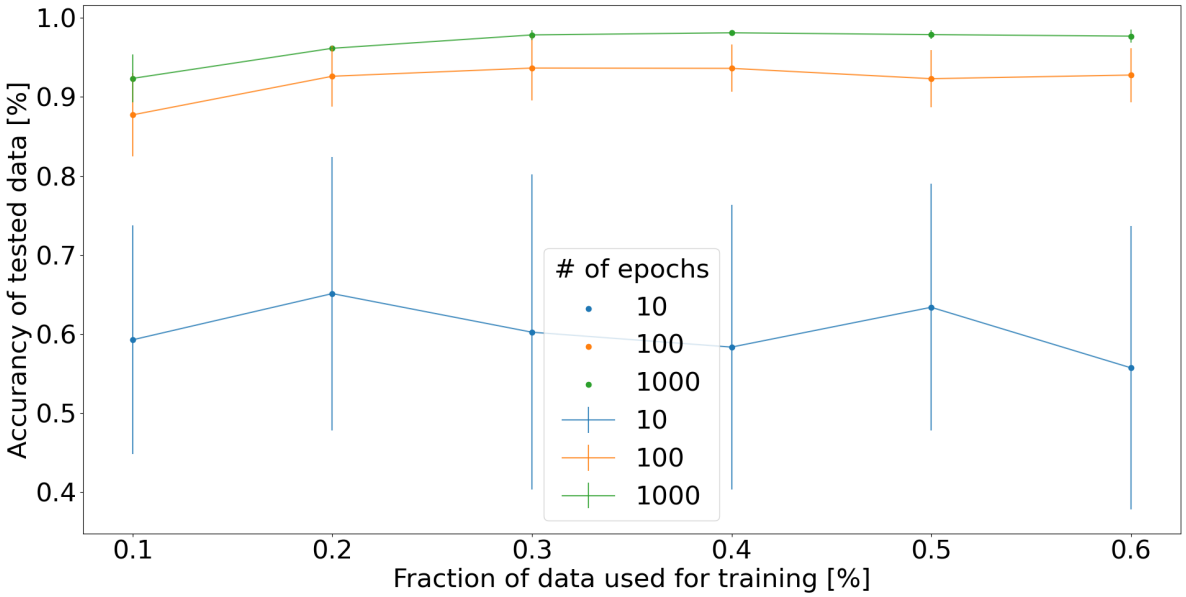

Figure 1.4: Benchmark of the `WineDecision` NN.

2 Introduction and Problem Definition

Mainly due to the wide availability of data, high computational power and versatile application possibilities the usage of Artificial Neural Networks (NNs) has increased rapidly in the last years. With more than 1.9 million publications [12] including the word *Artificial Neural Network* a hype in using NNs can be observed.
Generally NNs are considered as universal function approximators [6]. Based on input data with corresponding targets the NN learns in train passes unknown functional relations and evaluates in test passes input data the model has not seen before. If sufficient data is available, a trained NN is e.g. able to distinguish whether a dog or a cat is to be seen on an image, using the pixels of the image as input [13].
If only few or scarse data is available, the DNN starts to overfit its input and does not learn a generalized model. Physics-informed machine learning [14] is a promising way to improve the generalization ability of NNs by providing physical knowledge e.g. in the form of model equations to the NN.
In the physics-informed Finite Volume Neural Network (FINN) developed by Karlbauer et al. [15], learning abilities of NNs and knowledge about solving partial differential equations (PDEs) with the Finite Volume (FV) method, can be combined to model e.g. advection-dispersion processes of substances in contaminated soil. Models were trained and tested using both synthetic data, which are spatio-temporal solutions of processes described by PDEs, and experimental data. Experimental information were previously used, to learn constitutive relationships expressed by the retardation factor of a sorption process [16].

PFAS (per- and polyfluoroalkyl substances) are a group of approximately 4700 synthetic chemically and thermally stable, water and grease repellent chemicals that are frequently used in industry e.g. in cosmetics, tableware, for surface treatment of metals or in crop protection products. Chemically, PFAS consist of perfluorinated carbon chains, which can accumulate in various organisms due to their longevity. Humans ingest the toxic PFAS primarily from food or drinking water. PFAS can accumulate in the vadose zone, which is the area between the earth's surface and the water table, or enter groundwater. In particular, sorption and reaction processes of PFAS take place in this zone, which are still poorly understood in their entirety [17, 18].
Prior to this work, experimental data on PFAS transport through soil was generated [19] including column experiments performed under water-saturated conditions to evaluate the long-term leaching characteristics. Samples of contaminated soil were placed in a column with 2 sand layers (Dorsilit 7) on top and bottom and were flushed with water. Outflowing water was collected in a container in order to determine the concentration

of various PFAS after irregular periods of time. In addition, the mass concentration of sorbed PFAS in the soil at the end of the experiment was determined. In particular, the substance perfluorooctanesulfonic acid (PFOS) was investigated in the experiments. The perfluorinated PFAS is, in contrast to polyfluorinated compounds, difficult to degrade [20].

For modelling non-equilibrium 1D flow and transport processes in soil different models were described previously. There have been introduced different uniform transport [21] and non-equilibrium models [22, 23], which are based on the Richards [24] and Advection-Diffusion (AD) equation. A lot of these models are included in the software package Hydrus 1D of Simunek et al. [25]. Generally the non-equilibrium models distinguish between physical factors and chemical factors. Both models have also been combined to improve the description of solute transport and water flow [26].
The chemical non-equilibrium Two-Site sorption model (2SS) [27, 28], assumes two sorption sites: The first site describes sorption processes already in equilibrium and the second a first-order kinetic rate process.

Due to the lack of understanding of PFAS processes occurring in vadose zone, but the existence of various model equations to describe the physical and chemical non-equilibrium transport of substances, this topic is ideally suited to predict and learn occuring processes using a physics-informed NN based on scarse experimental data. In this work FINN was extended to learn the transport processes of PFOS through soil including the 2SS model. Using both, information about solving the PDE and experimental data of Bierbaum et al. relevant time dependent sorption processes and spatio-temporal profiles of transport quantities, which cannot be derived directly from experimental data, were learned. The core contributions of this work are:

- Implementation and validation of a FD solver for the AD equation including the 2SS model to generate synthetic training data for FINN.
- Validated embedding of the AD equation including the 2SS model into the FINN framework.
- Application of FINN on synthetic and scarse real-world experimental data in order to understand and generalize transport and sorption processes occuring in contaminated soil.

3 Methods

In order to benefit directly from the FINN code, the GitHub repository created by Karlbauer et al. [15] was forked. The methods needed in this work were implemented in the structure provided by the FINN framework. Other equations which are already implemented in the FINN framework (Allen-Cahn, Burgers, etc.) are not affected by this work.

3.1 Underlying Equations

To model the 1D transport of a substance through soil, water flow and solute transport have to be considered:
Water flow in soil can be described with the Richards equation [24]

$$\frac{\partial \theta(h)}{\partial t} = \frac{\partial}{\partial x}\left[K(h)\left(\frac{\partial h}{\partial x} + 1\right)\right] - S, \tag{3.1}$$

where x is the vertical coordinate positive upward $[L]$, t is the time $[T]$, h is the pressure head $[L]$, θ is the volumetric water content $[L^3L^{-3}]$, S is a source or sink term $[T^{-1}]$ and $K(h)$ is the unsaturated hydraulic conductivity function.

Numerical models for solute transport including sorption are based on the AD equation

$$\frac{\partial \theta c}{\partial t} + \rho \frac{\partial s}{\partial t} = \frac{\partial}{\partial x}\left(\theta D \frac{\partial c}{\partial x}\right) - \frac{\partial qc}{\partial x}, \tag{3.2}$$

where c is the concentration of the dissolved substance $[ML^{-3}]$, s is the mass specific sorbed concentration $[MM^{-1}]$, q is the Darcy flux $[LT^{-1}]$ and D is the dispersion coefficient. D is the combination of the hydrodynamic and the effective molecular diffusion coefficient [29], which is calculated by:

$$D = \alpha_l v + D_e, \tag{3.3}$$

where α_l is the longitudinal dispersivity $[L]$, D_e the effective molecular diffusion coefficient $[L^2T^{-1}]$, and v $[LT^{-1}]$ the seepage, or effective velocity. On the present Darcy scale, the effective velocity is assumed to be constant over space. However, due to the heterogeneity of the porous medium, local scattering, dilution or mixing of solutes may occur. To account for this, eq. (3.3) contains the velocity-proportional dispersion term

called hydrodynamic dispersion [29]. The effective velocity v is calculated by

$$v = \frac{q}{n_e}, \tag{3.4}$$

where q is again the Darcy flux $[LT^{-1}]$ and n_e $[-]$ the effective porosity.
During the simulations and experiments, a saturated water flow was set. Thus, the water fraction θ stayed constant for the entire simulation time and corresponds the effective porosity n_e. Therefore, the Richards equation (3.1) need not be solved and θ can be pulled out of time and space derivatives in the AD equation (3.2). Dispersion coefficient D and Darcy flux q are also spatially independent and can be pulled out. Dividing by n_e ($n_e \neq 0$) and using eq. (3.4) yields

$$\frac{\partial c}{\partial t} + \frac{\rho}{n_e}\frac{\partial s}{\partial t} = D\frac{\partial^2 c}{\partial x^2} - v\frac{\partial c}{\partial x}. \tag{3.5}$$

Using the 2SS model, the sorption process is divided into two parts

$$s = s_e + s_k, \tag{3.6}$$

where s_e $[MM^{-1}]$ is an instantaneous equilibrated sorption process, which is describable by a sorption isotherm, s_k $[MM^{-1}]$ is modeled as a first-order kinetic rate process. The term $\frac{\rho}{n_e}\frac{\partial s}{\partial t}$ in (3.5) thus expands to

$$\frac{\partial c}{\partial t} + \frac{\rho}{n_e}\frac{\partial s_e}{\partial t} + \frac{\rho}{n_e}\frac{\partial s_k}{\partial t} = D\frac{\partial^2 c}{\partial x^2} - v\frac{\partial c}{\partial x}. \tag{3.7}$$

In this work, the Freundlich sorption isotherm [30] was used for calculation of s_e

$$s_e = f k_d c^{\beta}. \tag{3.8}$$

f $[-]$ is the fraction of sites that are expected to be in equilibrium with the liquid phase, k_d $[L^3M^{-1}]$ is the Freundlich constant, and β $[-]$ the Freundlich exponent.
The first-order kinetic velocity process is modeled by

$$\frac{\partial s_k}{\partial t} = \alpha_k((1-f)k_d c^{\beta} - s_k), \tag{3.9}$$

where α_k is the first-order rate constant $[T^{-1}]$. The equation can be interpreted: A high dissolved concentration and a low kinetically sorbed concentration lead to a positive value on the right-hand side ($\alpha_k > 0$). Thus, the time derivative of s_k is positive and an increase in s_k can be observed for the next time step. The same reasoning can be done for a decrease of s_k.
Eq. (3.9) has the analytical solution

$$s_k(t) = \mathcal{K}e^{-\alpha_k t} + (1-f)k_d c(t)^{\beta} \qquad \mathcal{K} \in \mathbb{R}. \tag{3.10}$$

If equilibrium is reached with the liquid-phase concentration ($t \to \infty$), the originally kinetically sorbed concentration can then be calculated with

$$s_k(t) = (1 - f)k_d c(t)^\beta. \tag{3.11}$$

The sorption process is then describable by the sorption isotherm

$$s(t) = s_e(t) + s_k(t) = f k_d c(t)^\beta + (1 - f)k_d c(t)^\beta = k_d c(t)^\beta. \tag{3.12}$$

It holds for eq. (3.8) because of the chain rule (1.31)

$$\frac{\partial s_e}{\partial t} = f k_d \beta c^{\beta-1} \cdot \frac{\partial c}{\partial t}, \tag{3.13}$$

since f and k_d are time-independent. With eq. (3.13) it holds for eq. (3.7):

$$\begin{aligned} &\frac{\partial c}{\partial t} + \frac{\rho}{n_e} f k_d \beta c^{\beta-1} \cdot \frac{\partial c}{\partial t} + \frac{\rho}{n_e}\frac{\partial s_k}{\partial t} = D\frac{\partial^2 c}{\partial x^2} - v\frac{\partial c}{\partial x} \\ &\qquad\qquad \Leftrightarrow \\ &\frac{\partial c}{\partial t} \cdot \underbrace{\left(1 + \frac{\rho}{n_e} f k_d \beta c^{\beta-1}\right)}_{=: R(c)} + \frac{\rho}{n_e}\frac{\partial s_k}{\partial t} = D\frac{\partial^2 c}{\partial x^2} - v\frac{\partial c}{\partial x}. \end{aligned} \tag{3.14}$$

R represents the retardation factor resulting from instantaneous sorption.

3.2 Solving the Transport Equation with Finite Differences

All models in the FINN framework contain a FD solver to solve the underlying equations to generate synthetic training or test data. Thus, a FD solver for the AD equation including the 2SS model (AD2SS) was implemented too. To solve eq. (3.14) with different settings separately from the source code, in contrast to other FD solvers in the FINN framework, a `parmas` .json-file was created, where all parameters can be specified. Parameter names mentioned in the following always refer to the described `params` .json-file. For reproducibility an exemplary file can be found on GitHub [31].

In particular, on equidistant 1D grids, classical FD solvers are an intuitive way to solve PDEs. Based on Taylor's theorem, difference quotients are used to approximate derivatives of various orders.

3.2.1 Spatio-temporal Discretization

Let X_{LENGTH} $[L]$ be the length of the considered 1D domain Ω_x with $X_{STEPS} \in \mathbb{N}$ lattice points arranged equidistantly. Then for the lattice width dx $[L]$ holds

$$dx := \frac{X_{LENGTH}}{X_{STEPS} - 1}, \tag{3.15}$$

or for the discretized simulation space Ω_x

$$\Omega_x := \{0, 1 \cdot dx, 2 \cdot dx, ..., (X_{STEPS} - 2) \cdot dx, X_{LENGTH}\}. \tag{3.16}$$

Analogously, if T_{MAX} $[T]$ is the simulation time with $T_{STEPS} \in \mathbb{N}$ simulation time steps arranged equidistantly, then for the simulation time step dt $[T]$ holds

$$dt := \frac{T_{MAX}}{T_{STEPS} - 1}, \tag{3.17}$$

or for the discretized simulation time Ω_t

$$\Omega_t := \{0, 1 \cdot dt, 2 \cdot dt, ..., (T_{STEPS} - 2) \cdot dt, T_{MAX}\}. \tag{3.18}$$

Let $c : \Omega_x \times \Omega_t \to \mathbb{R}^{X_{STEPS}}$ be the unknown volume specific concentration of PFOS dissolved in water with the spatial indexing for any fixed but arbitrary $t \in \Omega_t$

$$c \begin{pmatrix} x = 0 \\ x = dx \\ ... \\ x = (X_{STEPS} - 2)dx \\ x = X_{LENGTH} \end{pmatrix}, t = \begin{pmatrix} c_1 \\ c_2 \\ ... \\ c_{X_{STEPS}-1} \\ c_{X_{STEPS}} \end{pmatrix}. \tag{3.19}$$

Let $s_k : \Omega_x \times \Omega_t \to \mathbb{R}^{X_{STEPS}}$ be the unknown mass specific concentration of PFOS kinetically sorbed in the contaminated soil, with the spatial indexing for any fixed but arbitrary $t \in \Omega_t$

$$s_k \begin{pmatrix} x = 0 \\ x = dx \\ ... \\ x = (X_{STEPS} - 2)dx \\ x = X_{LENGTH} \end{pmatrix}, t = \begin{pmatrix} s_{k,1} \\ s_{k,2} \\ ... \\ s_{k,X_{STEPS}-1} \\ s_{k,X_{STEPS}} \end{pmatrix}. \tag{3.20}$$

Homogeneous initial conditions were used

$$c(x, 0) = c_{init} \in \mathbb{R}^+ \qquad s_k(x, 0) = s_{k,init} \in \mathbb{R}^+ \quad \forall x \in \Omega_x. \tag{3.21}$$

The initial value problem

$$\frac{\partial}{\partial t}u(x,t) := \frac{\partial}{\partial t}\begin{pmatrix} c(x,t) \\ s_k(x,t) \end{pmatrix} = \begin{pmatrix} \frac{1}{R(c)}\left[D\frac{\partial^2}{\partial x^2}c(x,t) - v\frac{\partial}{\partial x}c(x,t) - \frac{\rho}{n_e}\frac{\partial}{\partial t}s_k(x,t)\right] \\ \alpha_k\left[fk_d c(x,t)^\beta - s_k(x,t)\right] \end{pmatrix} \tag{3.22}$$

$$u_0 := u(x,0) = \begin{pmatrix} c(x,0) \\ s_k(x,0) \end{pmatrix} \quad \forall x \in \Omega_x,$$

can be solved with an explicit Euler method up to the time T_{MAX}

$$u_{k+1} = u_k + dt \cdot \frac{\partial}{\partial t}u(x,t_k) \quad k \in \{0,...,T_{STEPS}-2\}. \tag{3.23}$$

It was assumed that the first-order rate process and changes of c occur at the same time step length. Therefore the two unknowns can be stacked on top of each other.

Advection and dispersion cause spatial changes of c at a time point t_k which were numerically approximated. Spatial discretization of eq. (3.14) was performed with a basic FD stencil

$$\begin{aligned} \frac{\partial c}{\partial t}R(c) + \frac{\rho}{n_e}\frac{\partial s_k}{\partial t} = {} & \frac{D}{dx^2}\begin{pmatrix} -2 & 1 & 0 & \dots & \dots & 0 \\ 1 & \ddots & \ddots & \ddots & & \vdots \\ 0 & \ddots & & & & \\ \vdots & \ddots & & & \ddots & \vdots \\ & & & & \ddots & 0 \\ \vdots & & \ddots & \ddots & \ddots & 1 \\ 0 & \dots & \dots & 0 & 1 & -2 \end{pmatrix} \cdot \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{X_{STEPS}-2} \\ c_{X_{STEPS}-1} \\ c_{X_{STEPS}} \end{pmatrix} \\ & -\frac{v}{dx}\begin{pmatrix} 1 & 0 & 0 & \dots & \dots & 0 \\ -1 & \ddots & \ddots & \ddots & & \vdots \\ 0 & \ddots & & & & \\ \vdots & \ddots & & & \ddots & \vdots \\ & & & & \ddots & 0 \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & \dots & 0 & -1 & 1 \end{pmatrix} \cdot \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{X_{STEPS}-2} \\ c_{X_{STEPS}-1} \\ c_{X_{STEPS}} \end{pmatrix}. \end{aligned} \tag{3.24}$$

Boundary conditions are required to correctly discretize the cells. At the inlet, thus the upper boundary, a Dirichlet boundary condition was used by implementing a ghost cell c_0

$$c_0 := 0 \quad \forall t \in \Omega_t. \tag{3.25}$$

At the outlet, thus the lower boundary, a Neumann boundary condition was used. Again by implementing a ghost cell $c_{X_{STEPS}+1}$

$$\frac{c_{X_{STEPS}+1} - c_{X_{STEPS}}}{dx} = 0 \quad \forall t \in \Omega_t. \tag{3.26}$$

Following the basic FD stencil as used for inner points in eq. (3.24) for c_1 would hold in

$$\frac{\partial c_1}{\partial t} R(c_1) + \frac{\rho}{n_e} \frac{\partial s_{k_1}}{\partial t} = \frac{D}{dx^2}(-2c_1 + c_2 + c_0) - \frac{v}{dx}(c_1 - c_0). \tag{3.27}$$

Since $c_0 = 0$, the discretization in eq. (3.24) remains correct.
Following the basic FD stencil as used for inner points in eq. (3.24) for $c_{X_{STEPS}}$ would hold in

$$\begin{aligned} &\frac{\partial c_{X_{STEPS}}}{\partial t} R(c_{X_{STEPS}}) + \frac{\rho}{n_e} \frac{\partial s_{k,X_{STEPS}}}{\partial t} = \\ &\frac{D}{dx^2}(-2c_{X_{STEPS}} + c_{X_{STEPS}-1} + c_{X_{STEPS}+1}) - \frac{v}{dx}(c_{X_{STEPS}-1} + c_{X_{STEPS}}). \end{aligned} \tag{3.28}$$

Using the Neumann boundary eq. (3.26) it must hold

$$\begin{aligned} &\frac{\partial c_{X_{STEPS}}}{\partial t} R(c_{X_{STEPS}}) + \frac{\rho}{n_e} \frac{\partial s_{k,X_{STEPS}}}{\partial t} = \\ &\frac{D}{dx^2}(-c_{X_{STEPS}} + c_{X_{STEPS}-1}) - \frac{v}{dx}(c_{X_{STEPS}-1} + c_{X_{STEPS}}). \end{aligned} \tag{3.29}$$

The error made in the last line for $c_{X_{STEPS}}$ is therefore compensated by a correction term

$$\begin{aligned} \frac{\partial c}{\partial t} R(c) + \frac{\rho}{n_e} \frac{\partial s_k}{\partial t} = \frac{D}{dx^2} &\begin{pmatrix} -2 & 1 & 0 & \dots & \dots & 0 \\ 1 & \ddots & \ddots & \ddots & & \vdots \\ 0 & \ddots & & & & \\ \vdots & \ddots & & & \ddots & \vdots \\ & & & & \ddots & 0 \\ \vdots & & \ddots & \ddots & \ddots & 1 \\ 0 & \dots & \dots & 0 & 1 & -2 \end{pmatrix} \cdot \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{X_{STEPS}-2} \\ c_{X_{STEPS}-1} \\ c_{X_{STEPS}} \end{pmatrix} \\ &+ \frac{D}{dx^2} \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ c_{X_{STEPS}} \end{pmatrix} \end{aligned}$$

$$-\frac{v}{dx}\begin{pmatrix} 1 & 0 & 0 & \dots & \dots & 0 \\ -1 & \ddots & \ddots & \ddots & & \vdots \\ 0 & \ddots & & & & \\ \vdots & \ddots & & & \ddots & \vdots \\ & & & & \ddots & 0 \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & \dots & 0 & -1 & 1 \end{pmatrix} \cdot \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{X_{STEPS}-2} \\ c_{X_{STEPS}-1} \\ c_{X_{STEPS}} \end{pmatrix}. \tag{3.30}$$

3.2.2 Transport through Contaminated Soil and Sand

By setting the `sandbool` - entry in the `params` .json-file as `true`, two sand layers at the upper and lower end of the column can be included in Ω_x. In the dictionary `sand` further parameters of the sand layers can be specified. The indices that describe material transitions are called `top` and `bot`. All lattice points until `top` are considered as sand counted from the beginning of the lattice. Grid points after this index have soil properties up to, and including the index `bot`. Subsequent grid points have sand properties again.
In sand no effective molecular diffusion but only hydrodynamic dispersion was considered. For this the longitudinal dispersivity of the sand is defined in `alpha_l`. The effective velocity was again calculated from the porosity in sand and q, where q stays constant for both materials due to continuity. In addition, based on experimental knowledge, in the sand layers no sorption processes occur and initially they do not contain dissolved (and no sorbed) PFOS. Thus, the discretization is done with the following initial conditions

$$c(x,0) = \begin{pmatrix} c_1 \\ \dots \\ c_{top-1} \\ c_{top} \\ c_{top+1} \\ \dots \\ c_{bot-1} \\ c_{bot} \\ c_{bot+1} \\ \dots \\ c_{X_{STEPS}} \end{pmatrix} = \begin{pmatrix} 0 \\ \dots \\ 0 \\ 0 \\ c_{init} \\ \dots \\ c_{init} \\ c_{init} \\ 0 \\ \dots \\ 0 \end{pmatrix} \qquad s_k(x,0) = \begin{pmatrix} s_{k,1} \\ \dots \\ s_{k,top-1} \\ s_{k,top} \\ s_{k,top+1} \\ \dots \\ s_{k,bot-1} \\ s_{k,bot} \\ s_{k,bot+1} \\ \dots \\ s_{k,X_{STEPS}} \end{pmatrix} = \begin{pmatrix} 0 \\ \dots \\ 0 \\ 0 \\ s_{k,init} \\ \dots \\ s_{k,init} \\ s_{k,init} \\ 0 \\ \dots \\ 0 \end{pmatrix}. \tag{3.31}$$

The vectorized treatment of the unknowns for sand and contaminated soil were adopted for all locally varying quantities occuring in the PDE. It holds for the effective velocities

and dispersion coefficients

$$v = q \begin{pmatrix} n_{e,sand}^{-1} \\ ... \\ n_{e,sand}^{-1} \\ n_{e,sand}^{-1} \\ n_{e,soil}^{-1} \\ ... \\ n_{e,soil}^{-1} \\ n_{e,soil}^{-1} \\ n_{e,sand}^{-1} \\ ... \\ n_{e,sand}^{-1} \end{pmatrix}, \qquad D = \begin{pmatrix} v_{sand}\alpha_{l,sand} \\ ... \\ v_{sand}\alpha_{l,sand} \\ v_{sand}\alpha_{l,sand} \\ v_{soil}\alpha_l + D_e \\ ... \\ v_{soil}\alpha_l + D_e \\ v_{soil}\alpha_l + D_e \\ v_{sand}\alpha_{l,sand} \\ ... \\ v_{sand}\alpha_{l,sand} \end{pmatrix}, \tag{3.32}$$

and the retardation coefficients resulting from instantaneous sorption

$$R(c) = \begin{pmatrix} 1 \\ ... \\ 1 \\ 1 \\ 1 + \frac{\rho}{n_e} f k_d \beta c_{top+1}^{\beta-1} \\ ... \\ 1 + \frac{\rho}{n_e} f k_d \beta c_{bot-1}^{\beta-1} \\ 1 + \frac{\rho}{n_e} f k_d \beta c_{bot}^{\beta-1} \\ 1 \\ ... \\ 1 \end{pmatrix}. \tag{3.33}$$

The rates of change of the dissolved concentration can be calculated as above, boundary conditions, dx and dt remain unchanged. Since no sorption processes are considered in sand, temporal derivatives of s_k are calculated with

$$\frac{\partial s_k}{\partial t} = \begin{pmatrix} \frac{\partial}{\partial t} s_{k,1} \\ ... \\ \frac{\partial}{\partial t} s_{k,top-1} \\ \frac{\partial}{\partial t} s_{k,top} \\ \frac{\partial}{\partial t} s_{k,top+1} \\ ... \\ \frac{\partial}{\partial t} s_{k,bot-1} \\ \frac{\partial}{\partial t} s_{k,bot} \\ \frac{\partial}{\partial t} s_{k,bot+1} \\ ... \\ \frac{\partial}{\partial t} s_{k,X_{STEPS}} \end{pmatrix} = \begin{pmatrix} 0 \\ ... \\ 0 \\ 0 \\ \alpha_k \left[(1-f) k_d c_{top+1}^{\beta} - s_{k,top+1} \right] \\ ... \\ \alpha_k \left[(1-f) k_d c_{bot-1}^{\beta} - s_{k,bot-1} \right] \\ \alpha_k \left[(1-f) k_d c_{bot}^{\beta} - s_{k,bot} \right] \\ 0 \\ ... \\ 0 \end{pmatrix}. \tag{3.34}$$

3.2.3 Hydrus for Data Validation

Hydrus [25] is a tool for modelling non-equilibrium flow and transport processes. In particular, the 2SS model was implemented. Hydrus can therefore be used for solving eq. (3.14) to validate the solution calculated by the described FD solver. In the following section, italicized terms denote the corresponding labels in Hydrus.

To run Hydrus with the same settings and parameters as in the FD solver, a horizontal *Standard Solute Transport* was chosen, step sizes and iteration criteria are not comparable to the explicit Euler method used in this work. $Q_s = \theta$ i.e. the saturated water content corresponds to the porosity n_e which is different in the sand and contaminated soil layer. For specifying the Darcy flux q between two pressure heads $h_b\,[L]$ and $h_c\,[L]$ with the distance $L\,[L]$ and a hydraulic conductivity $k_f\ [LT^{-1}]$ Darcy's law (1D) can be used

$$q = -k_f \frac{h_b - h_c}{L}. \tag{3.35}$$

The pressure heads, the distribution of sand and contaminated soil in the column, c_{init} and $s_{k,init}$, which were uniformly distributed, were set in Hydrus' *Soil Profile - Graphical Editor*. No water flow was simulated, therefore except of k_f other *Water Flow Parameters* were arbitrarily selectable.
The *Solute Transport Parameters* were the same as described above: *Bulk.D.* corresponds to ρ, *Disp.* corresponds to α_l, *Frac* corresponds to f, *Diffus. W.* corresponds to D_e, *Kd* corresponds to k_d. *Nu* is part of the Langmuir isotherm and was chosen to be 0 since only the Freundlich isotherm was used in this work, *Beta* corresponds to β, and *Alpha* corresponds to α_k.
In the *Solute Transport Boundary Conditions*, the *Upper Boundary Condition* was a *Concentration BC* with value 0 and the *Lower Boundary Condition* was a *Zero Concentration Gradient*. The initial concentrations were given *In Liquid Phase Concentrations*.

In Hydrus a Galerkin Finite Element (FE) method is used to solve the PDE. The spatial discretization can not be compared directly with the discretization performed in this work. Individual nodes of the FE solution are difficult to compare with individual nodes of the FD solution. In terms of time, the calculated solution can only be tapped at most every 250th time step. To compare Hydrus and FD solution the mean squared error (MSE) between Hydrus solution points at the outflow at simulation times that can be tapped, and the corresponding FD solution points was calculated. In case of sand the last cell containing contaminated soil was used to compare s_k. Knowing that it is not possible to calculate the same solution as Hydrus with numerical accuracy, the Hydrus solution was used as a reference for the accuracy of the FD solution.
The Hydrus solution was given as a `Nod_inf` .out-file. The Pyhdrus [32] library was used to transform the solution into a dictionary of Pandas Dataframes [8], which were then transformed into Numpy arrays and compared to the FD solution. Data validation occured by changing various parameters of the implemented model.

3.2.4 Stability

The described FD method for solving eq. (3.14) is an explicit forward-time forward-space differencing method (FDFS). Per time step information can only be passed from a node to neighboring nodes to stay numerically stable. Intuitively, the velocity multiplied by the time step dt should always be smaller than the node distance dx, expressed with the ratio

$$\frac{u \cdot dt}{dx} \leq 1. \tag{3.36}$$

This ratio is also called the Courant number [33], where u is the velocity occurring in the calculation. According to Biringen [34], moreover, for stability of a FDFS discretized AD equation holds

$$D\frac{dt}{dx^2} \leq \frac{1}{2}. \tag{3.37}$$

In the present case, u and D depend on values of R in the contaminated soil, since it is divided by in the discretization. In a sand-free simulation we get

$$u = \frac{v}{R^*}, \tag{3.38}$$

and

$$D = \frac{v\alpha_l + D_e}{R^*}, \tag{3.39}$$

where R^* is the minimum retardation coefficient of the entire simulation. Using eqs. (3.36), (3.15) and (3.17), for fixed T_{MAX}, u, D, and dx, we obtain

$$T_{STEPS} \geq \frac{T_{MAX}\, u}{dx} + 1 \quad \wedge \quad T_{STEPS} \geq 2\frac{T_{MAX}\, D}{dx^2} + 1, \tag{3.40}$$

as a condition for the minimal number of time steps. In simulations that include the two sand layers, v and D are usually larger due to lower porosity and $R = 1$ since no sorption processes are considered, hence v and D of the sand layer should be used to calculate the minimum time step number.

3.3 Solving the Transport Equation with FINN

The FINN framework proposed by Karlbauer et al. [15], contains different transport equations like the 2D Burgers', the Diffusion-Reaction or the Allen-Cahn equation, to provide the NN generalized physical information. For the diffusion-sorption equation so far no advection terms and only instantaneous sorption terms are implemented to learn time-independent retardation factors from synthetic data. The underlying model

equations are given by work of Nowak et al. [35]

$$\frac{\partial c}{\partial t} = \frac{D}{R(c)} \frac{\partial^2 c}{\partial x^2}, \tag{3.41}$$

$$\frac{\partial c_t}{\partial t} = D n_e \frac{\partial^2 c}{\partial x^2}, \tag{3.42}$$

$$R(c) = 1 + \frac{1 - n_e}{n_e} \rho k_d \beta c^{\beta - 1}, \tag{3.43}$$

where c and c_t $[ML^{-3}]$ are the dissolved and total concentration, respectively. D $[L^2T^{-1}]$ is the effective diffusion coefficient (without dispersivity, since no advection is considered) n_e the porosity of the soil and $R(c)$ the retardation factor, with corresponding bulk density ρ $[ML^{-3}]$, and parameters of the Freundlich isotherm, explained previously.

The FINN framework is technically structured into two main parts: In the `data` folder training and test data are generated via FD method. The FD solver described above was added to this folder. Executing the `data_generation_2ss` .py-file generates synthethic data with the parameters described in the `params` .json-file. The calculated solution of c and s_k over the entire simulation period are stored memory-efficiently as a .npy-file.

In the `models` folder there are several PINNs including FINN. In a `config` .json-file NN-specific parameters like the learning rate or the name of the model equation that should be used can be determined. In order to include this work the `"diffusion_ad2ss"` as a newly implemented model can be selected.
The high modularity of FINN is achieved by obejct orientation: All implemented models inherit from a parent class `FINN`, which inherits from PyTorch's `nn.Module` (see Propaedeuticum). Thus, a new class `FINN_DiffAD2ss` was also created to include the AD2SS equation.

Without learning functional relations, stencil values or parameters, FINN solves a PDE using a FV method starting from initial and boundary conditions, respectively. In general, similar to the FD solver, the PDE is spatially discretized for each time step and then integrated in time. As a first step the FV method for solving eq. (3.14) was inserted.

3.3.1 Finite Volume Method

In contrast to FD methods, the grid-based FV method belongs to the class of conservative methods to solve PDEs, because after discretization conservation equations remain. Considering again eq. (3.14)

$$\frac{\partial c}{\partial t} R(c) + \frac{\rho}{n_e} \frac{\partial s_k}{\partial t} = D \frac{\partial^2 c}{\partial x^2} - v \frac{\partial c}{\partial x}. \tag{3.44}$$

Control volumes $\mathcal{K}_i$, $i \in \{1, ..., X_{STEPS}\}$ are used for discretization. It holds

$$\mathcal{K}_i = [x_{i-}, x_{i+}]\,. \tag{3.45}$$

x_{i-} denotes the upper and x_{i+} the lower boundary of $\mathcal{K}_i$. Furthermore, let $x_i \in \Omega_x$ be the center of $\mathcal{K}_i$. As in the FD method, the centers of the control volumes dx are equidistant, which yields

$$dx = x_{i+} - x_{i-}. \tag{3.46}$$

It holds mass conservation of PFOS for a fixed $\mathcal{K}_i$

$$\int_{x_{i-}}^{x_{i+}} \frac{\partial c}{\partial t} \cdot R(c) + \frac{\rho}{n_e}\frac{\partial s_k}{\partial t}\mathrm{d}x = \int_{x_{i-}}^{x_{i+}} D\frac{\partial^2 c}{\partial x^2} - v\frac{\partial c}{\partial x}\mathrm{d}x. \tag{3.47}$$

$\mathcal{K}_i$ is not time-dependent, thus according to the Leibniz rule for parameter integrals, integral and time derivatives can be interchanged on the left-hand side

$$\frac{\partial}{\partial t}\int_{x_{i-}}^{x_{i+}} cR(c)\mathrm{d}x + \frac{\rho}{n_e}\frac{\partial}{\partial t}\int_{x_{i-}}^{x_{i+}} s_k\mathrm{d}x = \int_{x_{i-}}^{x_{i+}} D\frac{\partial^2 c}{\partial x^2} - v\frac{\partial c}{\partial x}\mathrm{d}x. \tag{3.48}$$

Using the midpoint rule, the integrals on the left-hand side can be approximated by multiplying the mean values of the fixed $\mathcal{K}_i$ by dx

$$\left[R(c_i)\frac{\partial c_i}{\partial t} + \frac{\rho}{n_e}\frac{\partial s_{k,i}}{\partial t}\right] \cdot dx = \int_{x_{i-}}^{x_{i+}} D\frac{\partial^2 c}{\partial x^2} - v\frac{\partial c}{\partial x}\mathrm{d}x. \tag{3.49}$$

In the 1D case, the divergence operator in the Gaussian integral theorem can be viewed as a derivative, which, taking advantage of the linearity of the integral, allows eq. (3.49) to be simplified to

$$\left[\frac{\partial c_i}{\partial t}R(c_i) + \frac{\rho}{n_e}\frac{\partial s_{k,i}}{\partial t}\right] \cdot dx = \underbrace{\oint_{x_{i-}}^{x_{i+}} D\frac{\partial c}{\partial x}\hat{n}\mathrm{d}\Gamma}_{:=A} - \underbrace{\oint_{x_{i-}}^{x_{i+}} vc\hat{n}\mathrm{d}\Gamma}_{:=B}\,. \tag{3.50}$$

Γ represents the boundary of $\mathcal{K}_i$ and $\hat{n}$ is the normal vector pointing out of $\mathcal{K}_i$. Since only the 1D case is considered, this is a scalar

$$\hat{n}_{i-} = -1 \qquad \hat{n}_{i+} = +1. \tag{3.51}$$

Evaluation of A yields

$$\begin{aligned}\oint_{x_{i-}}^{x_{i+}} D\frac{\partial c}{\partial x}\hat{n}\mathrm{d}\Gamma &= \left(D\frac{\partial c}{\partial x}\right)_{x_{i+}} - \left(D\frac{\partial c}{\partial x}\right)_{x_{i-}} \\ &= \left(D_i\frac{c_{i+1}-c_i}{dx}\right) - \left(D_i\frac{c_i - c_{i-1}}{dx}\right) = \frac{1}{dx}D_i(c_{i+1} - 2c_i + c_{i-1}).\end{aligned} \tag{3.52}$$

In eq. (3.52) D was approximated by the dispersion coefficient at node i and the derivative $\frac{\partial c}{\partial x}$ by a finite difference.

Evaluation of B yields

$$\oint_{x_{i-}}^{x_{i+}} vc\hat{n}\mathrm{d}\Gamma = (vc)_{x_{i+}} - (vc)_{x_{i-}} = v_i c_i - v_i c_{i-1} = v_i(c_i - c_{i-1}), \tag{3.53}$$

where v was approximated by the effective velocity at node i, c by the value of the node that appears before the corresponding border. If the transport equation contains a positive advection term, this method is also called upwind FV scheme.

In summary, the transport equation (3.14) can be discretized with eq. (3.49), which is basically again a FD discretization

$$R(c_i)\frac{\partial c_i}{\partial t} + \frac{\rho}{n_e}\frac{\partial s_{k,i}}{\partial t} = \frac{1}{dx^2}D_i(c_{i+1} - 2c_i + c_{i-1}) - \frac{1}{dx}v_i(c_i - c_{i-1}). \tag{3.54}$$

Karlbauer et al. developed the `flux_kernel` $\mathcal{F}_i$, which approximates the surface integral of the mass transfer over $\mathcal{K}_i$. It represents transport changes which take place at the upper $i+$ and lower $i-$ surface boundaries of $\mathcal{K}_i$, if parameters of (3.54) like the values of the differentiation stencils are unknown and should be learned with FINN

$$\mathcal{F}_i = f_{i-} + f_{i+} = \sum_{j=1}^{2} f_j \approx \oint_{x_{i-}}^{x_{i+}} \left(D\frac{\partial^2 c}{\partial x^2} - v\frac{\partial c}{\partial x}\right)\cdot\hat{n}\mathrm{d}\Gamma, \tag{3.55}$$

with

$$f_{i-} := \varphi_{\mathcal{N}_{i-}}(c, c_{i-1}) \cdot (\varphi_{\mathcal{D}}(c_i) + \mathcal{R}(\varphi_{\mathcal{A}}(c_i)) \tag{3.56}$$

$$f_{i+} := \varphi_{\mathcal{N}_{i+}}(c_i, c_{i+1}) \cdot (\varphi_{\mathcal{D}}(c_i) + \mathcal{R}(\varphi_{\mathcal{A}}(c_i)) \,. \tag{3.57}$$

Here $\varphi_{\mathcal{A}}(c_i)$ and $\varphi_{\mathcal{D}}(c_i)$ are learnable advection velocities and dispersion coefficients, respectively. $\varphi_{\mathcal{N}_{i-}}(c_i, c_{i-1})$ and $\varphi_{\mathcal{N}_{i+}}(c_i, c_{i+1})$ are linear layers that approximate derivatives which may also be learnable

$$\varphi_{\mathcal{N}_{i-}}(c_i, c_{i-1}) := \frac{\partial c_i}{\partial x} \quad \text{on } f_{i-} \tag{3.58}$$

$$\varphi_{\mathcal{N}_{i+}}(c_i, c_{i+1}) := \frac{\partial c_i}{\partial x} \quad \text{on } f_{i+}. \tag{3.59}$$

$\mathcal{R}$ was introduced to minimize numerical instability of learned advection velocities to apply an upwind differencing scheme. The first-order derivative thus can only be described either in f_{i-} or f_{i+}

$$\mathcal{R}(\varphi_{\mathcal{A}}(c_i)) = \begin{cases} ReLU(\varphi_{\mathcal{A}}(c_i)), & \text{on } f_{i-} \\ -ReLU(-\varphi_{\mathcal{A}}(c_i)), & \text{on } f_{i+} \end{cases} . \tag{3.60}$$

3.3.2 Embedding of the Transport Equation into the FINN Framework

In order to validate the implementation of the FV method in FINN with solutions of the FD method, FINN did not learn any terms or parameters firstly, but only solved eq. (3.14). In particular, dispersion coefficients, longitudinal dispersivities, Darcy flux and porosities for sand and soil were set as known, which yields

$$\frac{v}{dx} = \frac{1}{dx}\begin{pmatrix} \varphi_{\mathcal{A}}(c_1) \\ ... \\ \varphi_{\mathcal{A}}(c_{top-1}) \\ \varphi_{\mathcal{A}}(c_{top}) \\ \varphi_{\mathcal{A}}(c_{top+1}) \\ ... \\ \varphi_{\mathcal{A}}(c_{bot-1}) \\ \varphi_{\mathcal{A}}(c_{bot}) \\ \varphi_{\mathcal{A}}(c_{bot+1}) \\ ... \\ \varphi_{\mathcal{A}}(c_{X_{STEPS}}) \end{pmatrix} = \frac{q}{dx}\begin{pmatrix} n_{e,sand}^{-1} \\ ... \\ n_{e,sand}^{-1} \\ n_{e,sand}^{-1} \\ n_{e,soil}^{-1} \\ ... \\ n_{e,soil}^{-1} \\ n_{e,soil}^{-1} \\ n_{e,sand}^{-1} \\ ... \\ n_{e,sand}^{-1} \end{pmatrix} = \frac{1}{dx}\begin{pmatrix} v_s \\ ... \\ v_s \\ v_s \\ v_c \\ ... \\ v_c \\ v_c \\ v_s \\ ... \\ v_s \end{pmatrix}, \tag{3.61}$$

and

$$\frac{D}{dx^2} = \frac{1}{dx^2}\begin{pmatrix} \varphi_{\mathcal{D}}(c_1) \\ ... \\ \varphi_{\mathcal{D}}(c_{top-1}) \\ \varphi_{\mathcal{D}}(c_{top}) \\ \varphi_{\mathcal{D}}(c_{top+1}) \\ ... \\ \varphi_{\mathcal{D}}(c_{bot-1}) \\ \varphi_{\mathcal{D}}(c_{bot}) \\ \varphi_{\mathcal{D}}(c_{bot+1}) \\ ... \\ \varphi_{\mathcal{D}}(c_{X_{STEPS}}) \end{pmatrix} = \frac{1}{dx^2}\begin{pmatrix} v_{sand}\alpha_{l,sand} \\ ... \\ v_{sand}\alpha_{l,sand} \\ v_{sand}\alpha_{l,sand} \\ v_{soil}\alpha_l + D_e \\ ... \\ v_{soil}\alpha_l + D_e \\ v_{soil}\alpha_l + D_e \\ v_{sand}\alpha_{l,sand} \\ ... \\ v_{sand}\alpha_{l,sand} \end{pmatrix} = \frac{1}{dx^2}\begin{pmatrix} D_s \\ ... \\ D_s \\ D_s \\ D_c \\ ... \\ D_c \\ D_c \\ D_s \\ ... \\ D_s \end{pmatrix} \tag{3.62}$$

$\forall t \in \Omega_t$. v_s, D_s and v_c, D_c are the effective velocities for sand and contaminated soil, respectively.
In particular, v is positive and thus by definition of the ReLU function $\mathcal{R}(\varphi_{\mathcal{A}}(c_i)) = 0$ on f_{i+}. Hence with also non-learnable stencil values f_{i-} and f_{i+} can be resolved to

$$f_{i-} = \frac{D_i}{dx^2} \cdot [-1c_i + 1c_{i-1}] - \frac{v_i}{dx}[1c_i - 1c_{i-1}] \tag{3.63}$$

$$f_{i+} = \frac{D_i}{dx^2} \cdot [-1c_i + 1c_{i+1}]. \tag{3.64}$$

With the Dirichlet boundary condition eq. (3.25) at the upper end and the Neumann boundary condition eq. (3.26) at the lower end, for f_- holds

$$f_- := \begin{pmatrix} f_{1,-} \\ \dots \\ f_{top-1,-} \\ f_{top,-} \\ f_{top+1,-} \\ \dots \\ f_{bot-1,-} \\ f_{bot,-} \\ f_{bot+1,-} \\ \dots \\ f_{X_{STEPS},-} \end{pmatrix} = \begin{pmatrix} \frac{D_s}{dx^2}(-c_1 + c_0) - \frac{v_s}{dx}(c_1 - c_0) \\ \dots \\ \frac{D_s}{dx^2}(-c_{top-1} + c_{top-2}) - \frac{v_s}{dx}(c_{top-1} - c_{top-2}) \\ \frac{D_s}{dx^2}(-c_{top} + c_{top-1}) - \frac{v_s}{dx}(c_{top} - c_{top-1}) \\ \frac{D_c}{dx^2}(-c_{top+1} + c_{top}) - \frac{v_c}{dx}(c_{top+1} - c_{top}) \\ \dots \\ \frac{D_c}{dx^2}(-c_{bot-1} + c_{bot-2}) - \frac{v_c}{dx}(c_{bot-1} - c_{bot-2}) \\ \frac{D_c}{dx^2}(-c_{bot} + c_{bot-1}) - \frac{v_c}{dx}(c_{bot} - c_{bot-1}) \\ \frac{D_s}{dx^2}(-c_{bot+1} + c_{bot}) - \frac{v_s}{dx}(c_{bot+1} - c_{bot}) \\ \dots \\ \frac{D_s}{dx^2}(-c_{X_{STEPS}} + c_{X_{STEPS}-1}) - \frac{v_s}{dx}(c_{X_{STEPS}} - c_{X_{STEPS}-1}) \end{pmatrix}, \tag{3.65}$$

and for f_+ (cf. Fig. 3.1)

$$f_+ = \begin{pmatrix} f_{1,+} \\ \dots \\ f_{top-1,+} \\ f_{top,+} \\ f_{top+1,+} \\ \dots \\ f_{bot-1,+} \\ f_{bot,+} \\ f_{bot+1,+} \\ \dots \\ f_{X_{STEPS},+} \end{pmatrix} = \begin{pmatrix} \frac{D_s}{dx^2}(-c_1 + c_2) \\ \dots \\ \frac{D_s}{dx^2}(-c_{top-1} + c_{top}) \\ \frac{D_s}{dx^2}(-c_{top} + c_{top+1}) \\ \frac{D_c}{dx^2}(-c_{top+1} + c_{top+2}) \\ \dots \\ \frac{D_c}{dx^2}(-c_{bop-1} + c_{bot}) \\ \frac{D_c}{dx^2}(-c_{bot} + c_{bot+1}) \\ \frac{D_s}{dx^2}(-c_{bot+1} + c_{bot+2}) \\ \dots \\ \frac{D_s}{dx^2}(-c_{X_{STEPS}} + c_{X_{STEPS}+1}) \overset{(3.26)}{=} 0 \end{pmatrix}. \tag{3.66}$$

In summary

$$R(c)\frac{\partial c}{\partial t} + \frac{\rho}{n_e}\frac{\partial s_k}{\partial t} = f_- + f_+ =: \mathcal{F}. \tag{3.67}$$

So far the `flux_kernel` of the AD equation was embedded into the FINN framework. In a next step eq. (3.67) was reformulated and transformated to include the 2SS model and to handle a wide variety of functional relations and parameter dependencies without further implementation effort. Inspired by a general form of the AD2SS equation (Hydrus

documentation [36], eq. 3.10), eq. (3.67) was reformulated

$$\begin{aligned}\frac{\partial c}{\partial t}R(c) &= \mathcal{F} - \frac{\rho}{n_e}\frac{\partial s_k}{\partial t} \\ &= \mathcal{F} - \frac{\rho}{n_e}\alpha_k\left((1-f)k_d c^{\beta} - s_k\right) \\ &= \mathcal{F} \underbrace{-\frac{\rho}{n_e}\alpha_k(1-f)k_d c^{\beta-1}}_{=:F(c)} \cdot c + \underbrace{\frac{\rho}{n_e}\alpha_k s_k}_{=:G(s_k)},\end{aligned} \tag{3.68}$$

with the already described

$$R(c) = 1 + \frac{\rho}{n_e} f k_d \beta c^{\beta-1}. \tag{3.69}$$

Based on definitions, the time derivative of s_k can also be derived using F and G

$$\begin{aligned}\frac{\partial s_k}{\partial t} &= \begin{pmatrix} \frac{\partial}{\partial t}s_{k,1} \\ \dots \\ \frac{\partial}{\partial t}s_{k,top-1} \\ \frac{\partial}{\partial t}s_{k,top} \\ \frac{\partial}{\partial t}s_{k,top+1} \\ \dots \\ \frac{\partial}{\partial t}s_{k,bot-1} \\ \frac{\partial}{\partial t}s_{k,bot} \\ \frac{\partial}{\partial t}s_{k,bot+1} \\ \dots \\ \frac{\partial}{\partial t}s_{k,X_{STEPS}} \end{pmatrix} = \begin{pmatrix} 0 \\ \dots \\ 0 \\ 0 \\ \alpha_k\left[(1-f)k_d c_{top+1}^{\beta} - s_{k,top+1}\right] \\ \dots \\ \alpha_k\left[(1-f)k_d c_{bot-1}^{\beta} - s_{k,bot-1}\right] \\ \alpha_k\left[(1-f)k_d c_{bot}^{\beta} - s_{k,bot}\right] \\ 0 \\ \dots \\ 0 \end{pmatrix} \\ &= \begin{pmatrix} 0 \\ \dots \\ 0 \\ 0 \\ -\frac{n_e}{\rho}\left[F(c_{top+1})c_{top+1} + G(s_{k,top+1})\right] \\ \dots \\ -\frac{n_e}{\rho}\left[F(c_{bot-1})c_{bot-1} + G(s_{k,bot-1})\right] \\ -\frac{n_e}{\rho}\left[F(c_{bot})c_{bot} + G(s_{k,bot})\right] \\ 0 \\ \dots \\ 0 \end{pmatrix}.\end{aligned} \tag{3.70}$$

Again, no sorption processes occur in the sand layers. These assumptions lead ultimately to

$$\frac{\partial c_i}{\partial t} = \frac{1}{R(c_i)}\left(f_{i-} + f_{i+} + F(c_i)c_i + G(s_{k,i})\right), \tag{3.71}$$

```
flux = th.stack((flux_c, flux_sk), dim=1)
```

Listing 3.1: Stacking of calculated c and s_k fluxes.

```
def forward(self, t, u):
    """
    This function integrates du/dt through time using the
       Neural ODE method
    """
    pred = odeint(self.state_kernel, u[0], t, method="euler")
    return pred
```

Listing 3.2: Time integration with neural ODE.

and

$$\frac{\partial s_{k,i}}{\partial t} = -\frac{n_e}{\rho}\left[F(c_i)c_i + G(s_{k,i})\right], \tag{3.72}$$

for a single node i and to

$$\frac{\partial c}{\partial t} = \frac{1}{R(c)}\left(\mathcal{F} + F(c)c + G(s_k)\right), \tag{3.73}$$

and

$$\frac{\partial s_k}{\partial t} = -\frac{n_e}{\rho}\left[F(c)c + G(s_k)\right], \tag{3.74}$$

for all spatial coordinates.
Formally terms like R, F and G, can be considered as `state_kernel` introduced by Karlbauer et al. In this case it consists of 3 potentially learnable functions which are linked using basic arithmetic operations (Fig. 3.1).
By assuming the same time steps for changes of both c and s_k, stacking the approximated temporal derivatives at the current $t \in \Omega_t$ (Listing 3.1) and integrating them to a next time step in the overwritten `forward` method inherited from PyTorch's `nn.Module`, solved the PDE. For time integration a neural ODE method, called torchdiffeq [37] (Listing 3.2) was used to maintain gradients for each integration step to be able to backward pass the potentially newly learned NN parameters in the optimization step.

3.3.3 Implementation of Learnable Parameters and Functional Relations

In this work, either model parameters f, k_d, α_k, β or functional relations F, G and R were learned.

Learning Parameters

```
if not learn_f:
    self.f = th.tensor(f, dtype=th.double, device=self.device)
else:
    self.f = nn.Parameter(th.tensor(f, dtype=th.double,
        device=self.device))
```

Listing 3.3: Add (non)learnable parameter f to FINN model.

Learning only parameters required an initial guess, which was determined in the `config` .json-file. Setting the corresponding parameter at model initialization as learnable added the parameter as a degree of freedom to the NN (Listing 3.3). This parameter is optimized later in the learning process and is treated like a weight or bias of the DNN. During the forward pass, the dynamic computational graph is built, which is used to compute the gradients during the backward pass. A reinitialization of the parameters, if they are non-physical e.g. ($f \geq 1$) to simplify the optimization process is thus in principle not possible. Nevertheless, to include physical knowledge, the parameters were modified by applying functions that can be backward passed

$$\hat{f} = sig(f_{FINN}) \qquad \hat{\beta} = sig(\beta_{FINN}) \qquad \hat{k}_d = |k_{d,FINN}| \qquad \hat{\alpha}_k = |\alpha_{k,FINN}|. \tag{3.75}$$

The parameters with index $FINN$ are the DNN parameters, which were optimized, but in the `state_kernel` calculations $\hat{f}$, $\hat{\beta}$, $\hat{k}_d$ and $\hat{\alpha}_k$ were used. f can only assume values between 0 and 1. Generally the Freundlich exponent β can assume values greater than 1 but for this work β was also constrained by the sigmoid function in order to reduce the optimization space and numerical instability. The Freundlich constant k_d and the rate constant of the kinetic rate process α_k were assumed to be non-negative.

Learning Functional Relations

Functional relations were initialized with the `function_learner` implemented in the `FINN` parent class and approximated as DNN (Listing 3.4). Corresponding variables like `layer_sizes` were defined in the `config` .json-file. Learnable functions F, G and R depended on c or s_k and either implicit or explicit on time. Therefore, there are one or two `in_features` respectively. The first takes c or s_k the second, if necessary, the current time. The DNNs have one output feature. For example, the array $[2, 20, 40, 20, 1]$, in the `config` .json-file defines a NN with two input features, three hidden layers with 20-40-20 neurons each, and one output feature.
The tangent hyperbolic function was chosen as the activation function within the DNN in order to limit the activations to values between -1 and 1. If necessary also a sigmoid function can be used as activation function inside the DNN. In the output layer the sigmoid function was always used as activation function (Listing 3.4).
To allow values greater than 1, further learnable factors (f_{sc}, g_{sc}, r_{sc}, in code `f_fac`, `g_fac`, `r_fac`, respectively) were added to the DNNs and were multiplied by the outputs of the DNNs (Listing 3.5).

```
def function_learner(self):
    """
    This function constructs a feedforward NN required for
       approximating a functional relation.
    """
    layers = list()

    for layer_idx in range(len(self.layer_sizes) - 1):
        layer = nn.Linear(
            in_features=self.layer_sizes[layer_idx],
            out_features=self.layer_sizes[layer_idx + 1],
            bias=self.bias
            ).to(device=self.device)
        layers.append(layer)

        if layer_idx < len(self.layer_sizes) - 2 or not self.
           sigmoid:
            layers.append(nn.Tanh())
        else:
            # Use sigmoid function to keep the values strictly
               positive
            # (all outputs have the same sign)
            layers.append(nn.Sigmoid())
    return nn.Sequential(*nn.ModuleList(layers))
```

Listing 3.4: Usage of a DNN to learn functional relations.

```
if learn_f_hyd:
    self.func_f = self.function_learner().to(device=self.device)
    self.f_fac = nn.Parameter(th.tensor([0.1], dtype=th.double))
```

Listing 3.5: Initialize learnable F.

```
# learn F(c), functional relation only in soil layer. In sand: F(c) = 0
if not self.learn_f_hyd:
    f_hyd = th.zeros(self.Nx)
    f_hyd[self.x_start:self.x_stop] = -self.alpha*self.rho_s/self.n_e*(1-self.f)*(self.k_d*cw_soil**(self.beta-1))
else:
    time_vec = th.ones([len(cw_soil)])*t
    t_c = th.stack((cw_soil, time_vec), dim=1)
    f_hyd = th.zeros(self.Nx)
    f_hyd[self.x_start:self.x_stop] = (-self.func_f(t_c.float())*self.f_fac)[:,0]
```

Listing 3.6: Calling time-dependent F and multiply it by a learnable factor.

Analogous to the non-learnable case, the DNN approximation of the function is then called in the `state_kernel` calculation (Listing 3.6).
In order to provide the DNN physical knowledge, several constraints for the outputs of the functions were made

$$F(t,c) = \begin{cases} 0 & \text{if } c \text{ in sand} \\ -F_{FINN}(t,c) \cdot |f_{sc}| & \text{else} \end{cases} \tag{3.76}$$

$$G(t,s_k) = \begin{cases} 0 & \text{if } s_k \text{ in sand} \\ G_{FINN}(t,s_k) \cdot |g_{sc}| & \text{else} \end{cases} \tag{3.77}$$

$$R(t,c) = \begin{cases} 1 & \text{if } c \text{ in sand} \\ 1 + R_{FINN}(t,c) \cdot 10^{r_{sc}} & \text{else} \end{cases}. \tag{3.78}$$

The same assumptions apply to functional relations with only implicit time dependence.

An overview of all FINN extensions is given in Fig. 3.1.

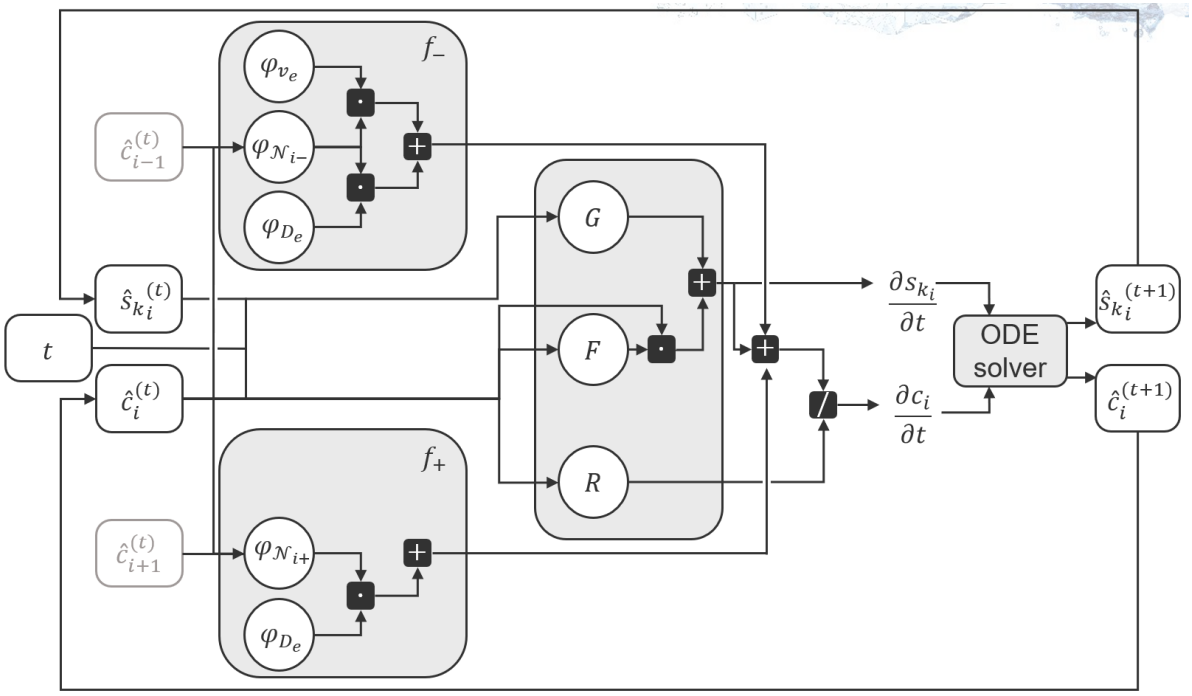

Figure 3.1: Overview of FINN extensions of this work. In this case learnable functions depended explicitly on time.

3.3.4 Training and Testing

Generally the training and testing of FINN predicted models is either based on a synthetic solution or scarce experimental data. In each of the following procedures PyTorch's `Adams` algorithm [38] was chosen as optimizer (Listing 3.7). The learning rate is also set in the `config` .json-file and was partly reduced manually with progressing training epoch number. An automated learning rate schedule was not developed.

Synthetic Data
Training with synthetic data was done either with the entire solution $c(t,x)$, $s_k(t,x) \quad \forall t,x \in \Omega_t \times \Omega_x$ or based on one quarter of the total simulation time to do so-called *in-dis-tests*, in which the learned model is tested with the remaining part of the data set, to assess the extrapolation ability of FINN.
In the first case no testing was performed and the Mean Squared Error Loss (MSE) with respect to the FD solution of the entire data set was used. With this setup, model

```
# Set up an optimizer and the criterion (loss)
optimizer = th.optim.Adam(model.parameters(), lr=config.
    training.learning_rate)
```

Listing 3.7: Using Adams algoritm for loss calculation.

```
mse = nn.MSELoss(reduction="mean")(u_hat,u)
```

Listing 3.8: Using MSE Loss in PyTorch, u: FD solution, u_hat: FINN approximation.

parameters were found.
In the second case, again the MSE with respect to the first quarter of the FD solution was used as loss function during training. Calculating the MSE of the entire data set after the test pass resulted in the accuracy of extrapolation of FINN.
Generally for MSE loss calculation PyTorch's functionality can be used (Listing. 3.8). With the keyword `mean` the loss is divided by the total number of data points N, which is

$$N = X_{STEPS} \cdot T_{STEPS} \cdot 2. \tag{3.79}$$

For each discretized location and time, N or $\frac{N}{4}$ values including c and s_k are available, respectively.

Usage of Experimental Data
In order to include experimental data, the value of the key `expdata` in `config` .json-file can be set to `true`. Using Panda's `read_excel` method, the Excel sheet provided by Bierbaum et al. [19] was read in and converted to a Pandas Dataframe. The data set contains all experimental data of the 4 column experiments (N1_1 - N1_4).
As mentioned in the introduction chapter, the column was flushed with water for approximately 150 days in each experiment. Water that flew out of the column was collected in an empty container until a specific time, then the concentrations of different PFAS were measured, the container was emptied, and the process was repeated for irregular time increments. Thus, in a measurement at one point in time, all PFAS have accumulated since the previous measurement. Hence, the measurement approximates the outflow concentration at the point in time between the current and previous measurement. To perform this half time step shift, the PFAS concentration in the outflow of the column at the beginning of the experiment was added ($0\frac{\mu g}{g}$), and then the measurement results were assigned to the mean times between two measurements (Fig. 3.2). This data was then used in the loss calculation.
In addition, the total concentrations of PFAS sorbed in the soil at the end of the experiment were investigated. They were approximately homogeneously distributed over the entire column.

Calculation of Initial Conditions based on Experimental Data
In order to solve the PDE initial conditions are needed. For an arbitrarily chosen homogeneously distributed $\tilde{c}(t=0)$ the initial condition for $\tilde{s}_k(t=0)\ \forall x \in \Omega_x$ was calculated using mass conservation

$$\tilde{m}_{tot} = \tilde{m}_{leached} + \tilde{m}_{end} \overset{!}{=} \tilde{c}(t=0)\frac{V}{m_{soil}} + \tilde{s}_k(t=0) + \tilde{s}_e(t=0). \tag{3.80}$$

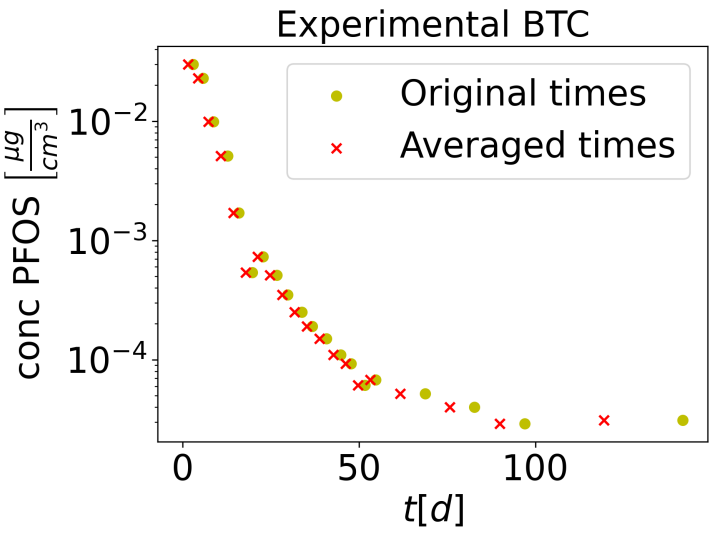

Figure 3.2: BTC of experiment N1_1, original data points (green), data points averaged over time (red).

$\tilde{m}_{tot}$ $[MM^{-1}]$ is the total initial mass concentration of PFOS in soil, $\tilde{m}_{leached}$ $[MM^{-1}]$ is the mass concentration of PFOS collected in the container at the outflow and $\tilde{m}_{end}$ $[MM^{-1}]$ the sorbed mass concentration at the end of the experiment. These values were measured by the experimentalists. Since the soil was soaked in water before the experiment started, $\tilde{m}_{tot}$ can also be calculated using $\tilde{c}(t=0)$, the cavity volume of the column V $[L^3]$, the soil mass m_{soil} $[M]$, the unknwon $\tilde{s}_k(t=0)$ and $\tilde{s}_e(t=0)$ $[MM^{-1}]$, which is the the initial instantaneously sorbed concentration. $\tilde{s}_e(t=0)$ can be calculated using the Freundlich isotherm and model parameters optimized by Bierbaum et al. V was calculated with

$$V = \pi \left(\frac{d}{2}\right)^2 n_e h, \tag{3.81}$$

h $[L]$ is the length of the column and d $[L]$ the diameter.
Since the choice of $\tilde{c}(t=0)$ is directly coupled to $\tilde{s}_e(t=0)$, $\tilde{c}(t=0)$ must not be chosen too large. Physically, it also makes sense to choose $\tilde{c}(t=0)$ in the order of magnitude of the first measured concentration at the outflow.

Loss Calculation for Experimental Data
Bierbaum et al. optimized manually model parameters for given experimental data. In order to reproduce this work with FINN, in particular the parameter f as ratio between kinetically sorbed and instantaneously sorbed concentration was learned. Several efforts were made to find an accurate loss function.
Since the BTC data is log-scaled, a basic MSE as loss function focusses on fitting data points with relatively large orders of magnitude and neglects deviations of concentrations especially at the end of the experiment, where concentrations with significantly smaller orders of magnitude occurred. An appropriate loss function turned out to be

the mean squared logarithmic loss of the data points, in order to have the same order of magnitude between the FINN calculated $\hat{c}$ and the experimental $\tilde{c}$ at experimental measurement times $\tilde{\Omega}_t$

$$L_I(\tilde{c}, \hat{c}) = \frac{1}{|\tilde{\Omega}_t| - 1} \sum_{i=1}^{|\tilde{\Omega}_t|} \left[\log \tilde{c}(t_i, x = X_{STEPS}) - \log \hat{c}(t_i, x = X_{STEPS})\right]^2, \tag{3.82}$$

with $t_i \in \tilde{\Omega}_t$. Previously the $\hat{c}$ and $\tilde{c}$ were rescaled by multiplying by 10^5 so that each value is greater than 1. Due to the initial condition at $t = 0$ (and $x = X_{STEPS}$) $\tilde{c}$ and $\hat{c}$ are the same. Thus no loss calculation is needed for this value.

When learning functional relations, $\hat{c}$ and $\hat{s}_k$ can take values of various orders of magnitude (a) as well as negative values (b). A fixed scaling and subsequent logarithmization of the values is not possible. Therefore, to solve problem (a) the averaged sum of the relative L1 error was used as the loss function

$$L_{II}(\tilde{c}, \hat{c}) = \frac{1}{|\tilde{\Omega}_t| - 1} \sum_{i=1}^{|\tilde{\Omega}_t|} \frac{|\tilde{c}(t_i, x = X_{STEPS}) - \hat{c}(t_i, x = X_{STEPS})|}{\tilde{c}(t_i, x = X_{STEPS})}. \tag{3.83}$$

To solve problem (b) a loss penalty was introduced. Let therefore $\mathbb{O}$ be a tensor containing zeros with the dimensions $X_{STEPS} \times T_{STEPS} \times 2$. And

$$\hat{u}_{neg} := ReLU(-\hat{u}), \tag{3.84}$$

where $\hat{u}$ is the previously described stacked $\hat{c}$, $\hat{s_k}$ FINN predicted solution. The loss penalty was then calculated with

$$L_{III}(\mathbb{O}, \hat{u}) = 10^5 \cdot MSE(\hat{u}_{neg}, \mathbb{O}). \tag{3.85}$$

Thus, the more negative $\hat{c}$ and $\hat{s_k}$ are, the larger the loss value becomes. The scaling factor was determined manually. Eqs. (3.83) and (3.85) lead ultimately to the applied loss function for learning functional relations on experimental data

$$L(\tilde{u}, \hat{u}) = L_{II}(\tilde{c}, \hat{c}) + L_{III}(\mathbb{O}, \hat{u}). \tag{3.86}$$

By learning functional relations for experimental data it makes also sense to asses the ability of FINN to learn relations out of initial distributions (*out-dis-tests*). Therefore a model for the functional relations is trained based on experimental data N1_1 and then tested with experimental data N1_3, where other parameters occured. In particular the Darcy flux, the mass density, and observed simulation time deviated from the experimental training data set, and were changed before forward passing the trained model. The loss was then calculated using the measurement time points of the new experiment N1_3 and eq. (3.86).

In order to reproduce this work the following repositories were created: Fork of FINN including comprehensive embedding of 2SS model and DNN models trained with synthetic data [31], source code for DNN models trained with experimental data on high-performance cluster [39] and Hydrus files for validation [40].

4 Results

The FD solver and the embedding of the FV solver into FINN were done from scratch. Therefore, in the following part, results will be presented step by step, which will build on each other. Through this approach, we tried to minimize methodological and technical errors to point out deviations and anomalies.

4.1 Validation

Solutions of the FD solver were checked for plausibility using various input conditions and were compared with Hydrus solutions to evaluate its correctness.

4.1.1 Plausibility Check

Based on an initial point concentration at $x = 28cm$ advection processes without dispersion and any sorption processes were simulated. It is plausible that at a set effective velocity of $78.9\frac{cm}{d}$ the first PFOS reaches the outflow after about $t = 0.3d$ and the mass or concentration of PFOS remains approximately constant over the time before it flows out (Fig. 4.1). Without effective velocity but with sorption and diffusion terms respec-

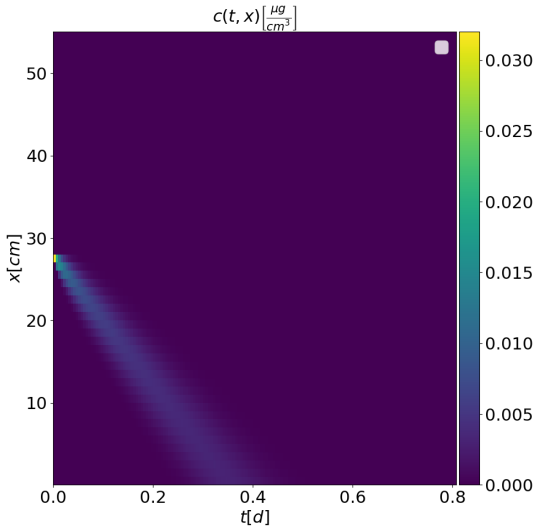

Figure 4.1: Advection process of initial point concentration without sorption processes.

tively, the funnel-shaped concentration spreading of c and s_k can be expected (Fig. 4.2). The kinetically sorbed concentration increases at the beginning, so with eq. (3.9) we have $(1-f)k_d c^\beta > s_k$, which is plausible too.

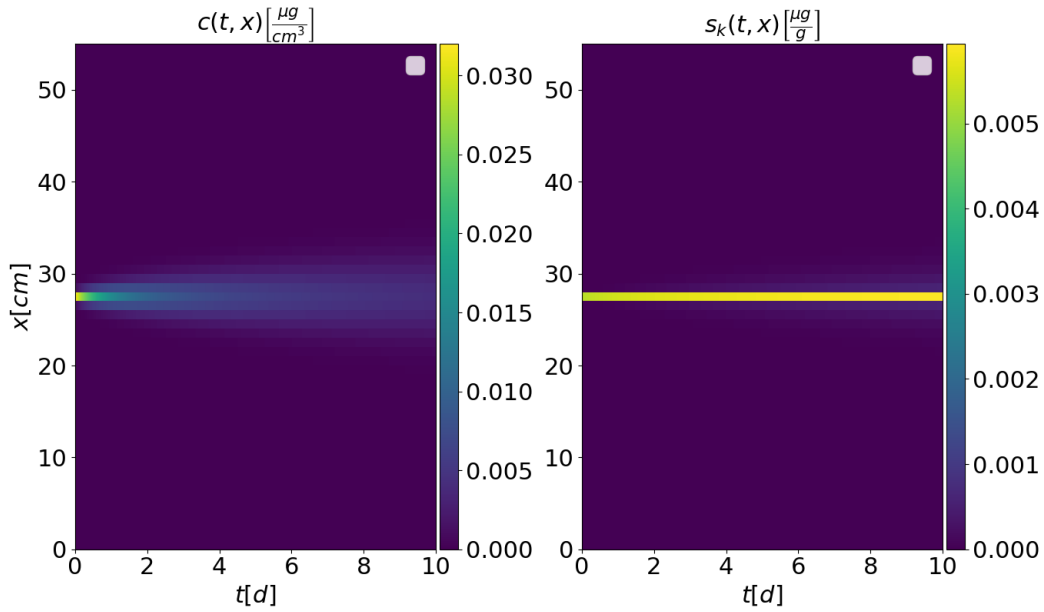

Figure 4.2: Diffusion process of initial point concentration with sorption processes.

4.1.2 Validation with Hydrus

A solution of the FD solver (Fig. 4.3, top left, bottom left) with corresponding parameters (Material parameters: Table 4.1, Model parameters: Table 4.2, Discretization parameters: Table 4.3) was used for validation with Hydrus.

$n_e\,[-]$	$\rho\left[\frac{g}{cm^3}\right]$	$D_e\left[\frac{cm^2}{d}\right]$	$\alpha_l\,[cm]$	$c_{init}\left[\frac{\mu g}{cm^3}\right]$	$s_{k,init}\left[\frac{\mu g}{g}\right]$	$q\left[\frac{cm}{d}\right]$
0.4	1.58	2.5	10	0.032	0.0053	20

Table 4.1: Material parameters used for validation of the solution with Hydrus.

$k_d\left[\frac{cm^3}{d}\right]$	$\beta\,[-]$	$f\,[-]$	$\alpha_k\left[\frac{1}{d}\right]$
4.5	0.98	0.5	0.005

Table 4.2: Model parameters used for validation of the solution with Hydrus.

The kinetic sorption process has not yet reached its equilibrium, the large changes in c affect the kinetic sorption behavior. As expected in the solution of the rate process eq.

$T_{MAX}\,[d]$	$T_{STEPS}\,[-]$	$X_{LENGTH}\,[cm]$	$X_{STEPS}\,[-]$
200	200000	40	80

Table 4.3: Discretization parameters used for validation of the solution with Hydrus.

(3.10), for large t an exponential decrease was observable (Fig. 4.4, right).
In order to asses the quality of the FD solver a Hydrus solution (Fig. 4.3, top right, bottom right) with the same parameters was produced. To receive a Darcy flux of $20\frac{cm}{d}$ a vertical flow with column length $L = 40cm$, $k_f = 20\frac{cm}{d}$, $h_b = 100cm$ and $h_c = 140cm$ with constant pressure head as boundary condition was chosen.
Hydrus values of c and s_k in the last row at given times ($t = 0d, ..., 199d$) resulted in a MSE $= 4.31 \times 10^{-9}$ for c and MSE $= 4.52 \times 10^{-10}$ for s_k, with respect to corresponding points in the FD solution (Fig. 4.4). The MSE complies with the different orders of magnitude of c and s_k.

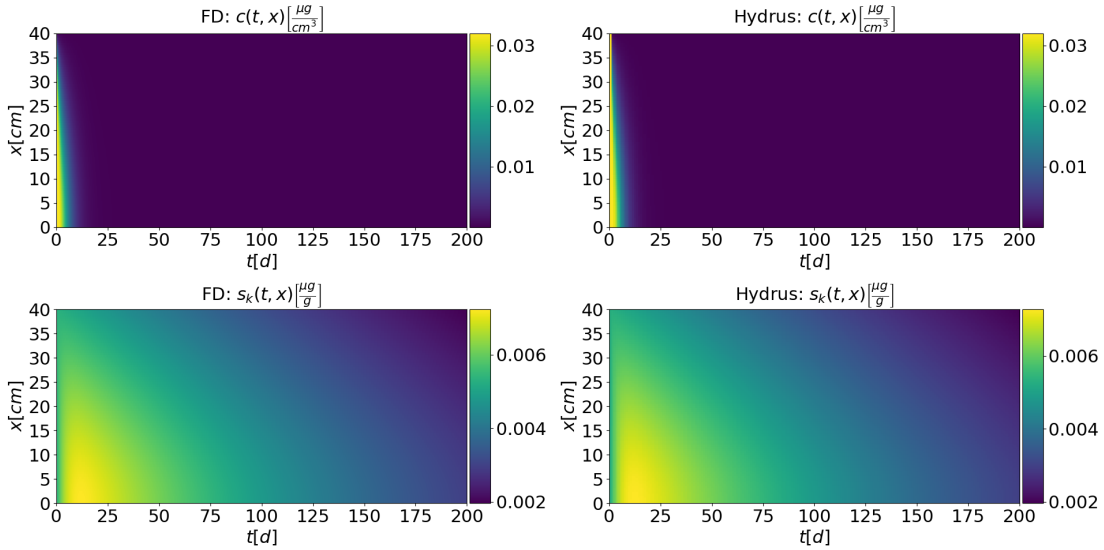

Figure 4.3: Comparison of Hydrus and FD solution without sand.

In a next step, the FD solution including sand layers was compared with a corresponding solution generated by Hydrus (Fig. 4.5). Parameters above were adopted and some were added to describe properties of the sand (Tab. 4.4). Using the assumption that

$n_{e,sand}\,[-]$	$\alpha_{l,sand}\,[cm]$	top	bot
0.31	5	10	70

Table 4.4: Sand parameters used for Hydrus validation

no sorption processes occured in the sand layers, sorption parameters or the density of

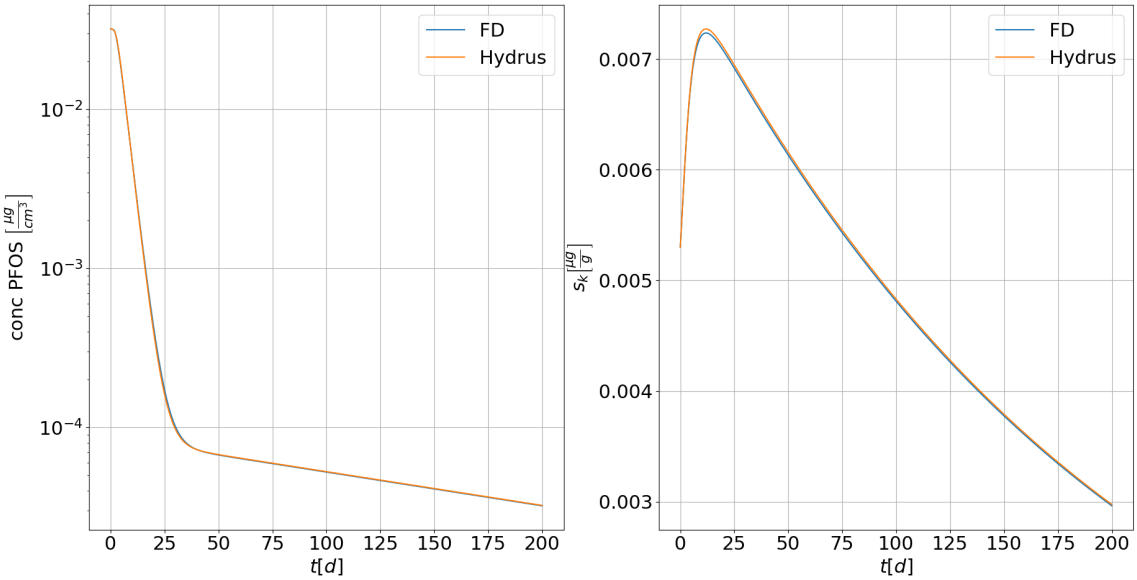

Figure 4.4: Comparison of Hydrus and FD solution without sand: c in bottom row (left), s_k in bottom row (right).

sand were not needed. Molecular diffusion was also neglected in the sand layers.
Comparing the BTCs and s_k of the bottom non-sand cell (x_{bot}) of the two solutions resulted in larger deviations: c: MSE $= 1.90 \times 10^{-5}$, s_k: MSE $= 2.71 \times 10^{-6}$.
At times close to $t = 0d$ PFOS has not yet flown out of the column, since it must first be transported through the lower sand layer (Fig. 4.6).

Further comparisons with other parameters can be found in the appendix (Figs. 6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 6.7). As already mentioned, corresponding Hydrus files are available on GitHub [40].

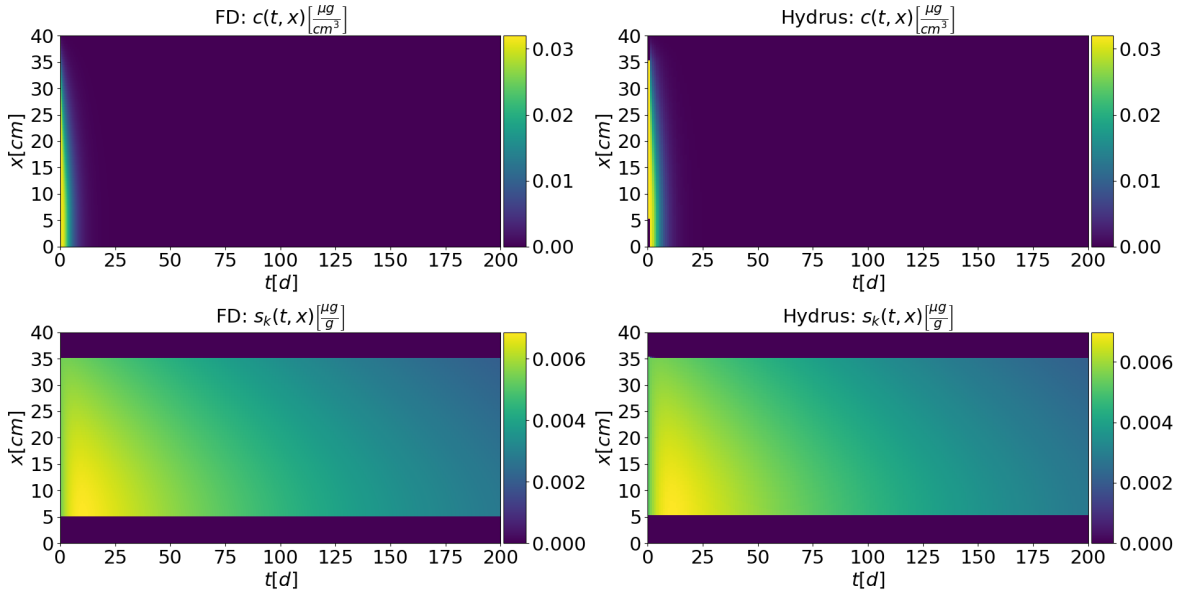

Figure 4.5: Comparison of Hydrus and FD solution of c and s_k with 2 sand layers.

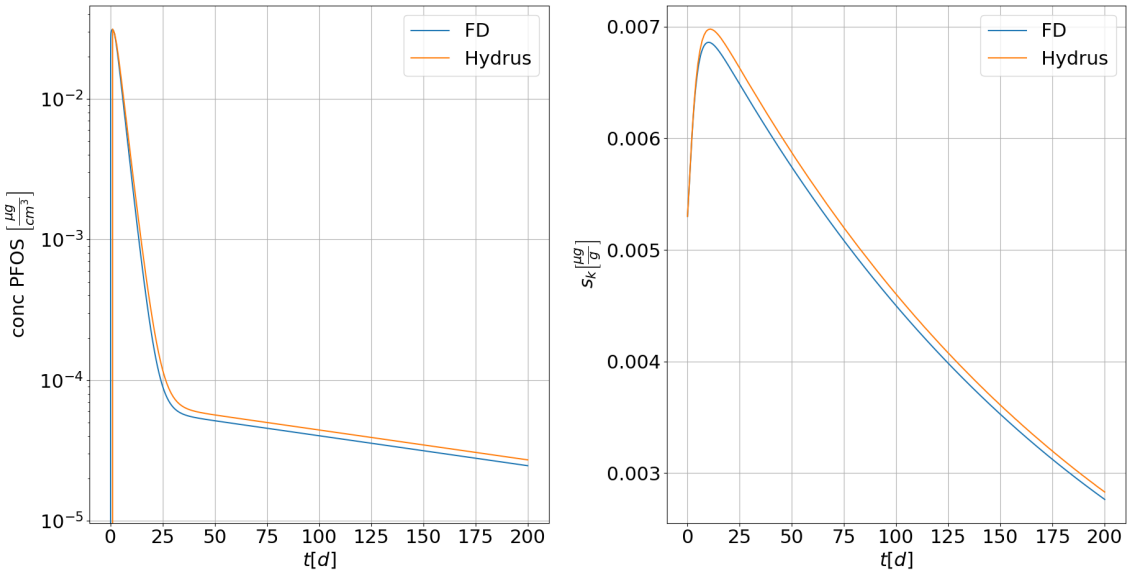

Figure 4.6: Comparison of Hydrus and FD solution with sand: c in bottom row (left), s_k in bottom contaminated soil row (right), setting with two sand layers.

4.2 Learning from Synthetic Data

With previously, with the FD solver generated synthetic data of c and s_k, the learning capability of FINN was demonstrated. The material parameters (Table 4.5) were inspired by work of Bierbaum et al. The model parameters (Table 4.6) were chosen to maximize effects of both instantaneous and kinetic sorption, as well as effects of the Freundlich

isotherm, in order to accurately measure FINN's learning ability. Only two days and a relatively rough spatial discretization was chosen to save computational effort (Table 4.7). The number of time steps used, depended on the CFL condition in the sand, was calculated with eq. (3.40)

$$T_{STEPS} \geq 2 \cdot T_{MAX} \cdot \frac{\alpha_L \cdot \frac{q}{n_e}}{R^*} \cdot \frac{1}{(\frac{X_{LENGTH}}{X_{STEPS}-1})^2} + 1 = 2 \cdot 2 \cdot \frac{5 \cdot \frac{31.56}{0.31}}{1} \cdot \frac{1}{(\frac{55}{28-1})^2} + 1 \approx 492, \quad (4.1)$$

and was chosen with a corresponding margin.

	$n_e\,[-]$	$\rho\left[\frac{g}{cm^3}\right]$	$D_e\left[\frac{cm^2}{d}\right]$	$\alpha_l\,[cm]$	$c_{init}\left[\frac{\mu g}{cm^3}\right]$	$s_{k,init}\left[\frac{\mu g}{g}\right]$	$q\left[\frac{cm}{d}\right]$
sand	0.31	NaN	0	5	0	0	31.56
soil	0.4	1.58	2.5	9	0.032	0.0053	31.56

Table 4.5: Initial material parameters of FINN, used for learning from synthetic data.

	$k_d\left[\frac{cm^3}{d}\right]$	$\beta\,[-]$	$f\,[-]$	$\alpha_k\left[\frac{1}{d}\right]$
soil	1.5	0.7	0.3	0.1

Table 4.6: Initial model parameters of FINN, used for learning from synthetic data.

$T_{MAX}\,[d]$	$T_{STEPS}\,[-]$	$X_{LENGTH}\,[cm]$	$X_{STEPS}\,[-]$	top	bot
2	600	55	28	4	24

Table 4.7: Initial discretization parameters of FINN, used for learning from synthetic data.

4.2.1 Learning Parameters

Run a: Learn Dummy Parameter
In order to validate the FV method added to FINN, all parameters and functional relations are first known and correspond to those from the FD solver. Technically, in order to not receive error messages in the optimizer in Pytorch, a dummy parameter must be added, that does not change the solution, but is considered in the dynamic computational graph. Here, a parameter with value 0 was added in the `flux_kernel` when calculating $\mathcal{R}(\varphi_{\mathcal{A}}(c_i))$ on f_{i+}, since this is also always 0 due to the positive v.
The MSE of the entire FINN solution with respect to the entire FD solution was used to calculate the loss (Tab. 4.8, Fig. 4.7). Deviations only occured at specific points, but in its entirety the differences of c_{FD} and c_{FINN}, and $s_{k,FD}$ and $s_{k,FINN}$ fluctuated around zero (Fig. 4.8). The MSE corresponds to machine accuracy. The FV method added to FINN thus produces the same results as the FD solver.

run a learn	Parameters ref	 init	 predicted	Learning schedule epochs	 lr	 MSE
dummy	0	0	0	1	0.1	5.2×10^{-19}

Table 4.8: FINN pass with dummy parameter, run a.

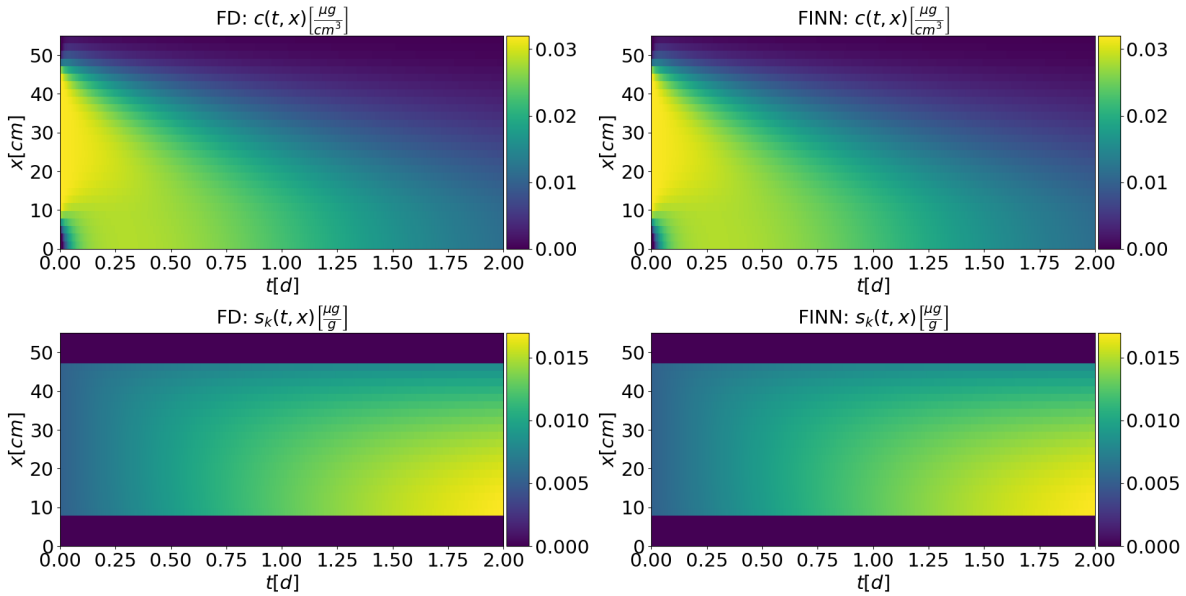

Figure 4.7: Comparison of FD and FINN solution, run a.

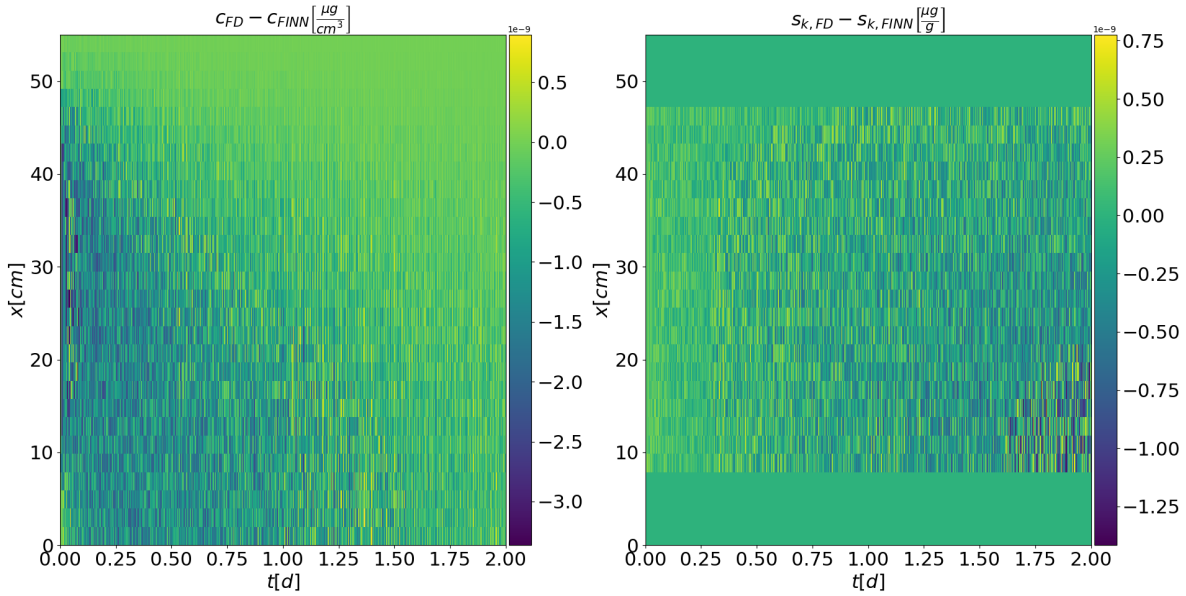

Figure 4.8: Differences of FD and FINN solution, run a.

Run b: Learn f
In order to test FINN's optimization ability the parameter f was included by an initial guess and set as learnable (Tab. 4.9, Figs. 6.8, 6.9, 6.10, 6.11). With the described learning schedule f was approximated exactly. The MSE corresponds again to machine accuracy.

run b	Parameters			Learning schedule		
learn	ref	init	predicted	epochs	lr	MSE
$f[-]$	0.3	0.7	0.3	20	0.1	4.18×10^{-7}
				20	0.01	2.61×10^{-10}
				10	0.001	2.02×10^{-13}
				10	0.0001	6.79×10^{-18}

Table 4.9: FINN pass with unknown f, ref denotes the f used in the FD solver which was used to obtain the synthetic training data, init denotes the initial guess for f of FINN, predicted denotes the value of f after optimization. In total 60 epochs were executed with the described learning schedule, run b.

Run c: Learn f, k_d, β, α_k
All model parameters f, k_d, β and α_k were learnable in run c and included by initial guesses (Tab. 4.10, Fig. 4.9). With the described learning schedule the parameters were approximated well. Using the high amount of epochs the MSE is again close to machine accuracy. Deviations in the differences occured because the approximated parameters did not yet correspond exactly to the initial parameters (Fig. 4.10). In order to visualize the deviations between the FINN and FD solution at fixed time steps, c and s_k of the entire contaminated soil layer were plotted. These c - s_k sorption isotherms were approximated well (Fig. 4.11). Additionaly the BTC (Fig. 4.12, left) and s_k at $t = 2.0d$ over the column length (Fig. 4.12, right) were visualized. Because of very close approximation of the parameters to the reference values, these courses were approximated well too. As expected with decreasing column length s_k is increasing. s_k accumulates at the bottom of the contaminated soil layer. Thus, for the entire simulation period $(1 - f)k_d c^\beta > s_k$.

run c	Parameters			Learning schedule		
learn	ref	init	predicted(*)	epochs	lr	MSE
$f[-]$	0.3	0.8	0.300096	10	0.1	5.13×10^{-5}
$k_d \left[\frac{cm^3}{d}\right]$	1.5	2	1.499331	300	0.01	2.99×10^{-8}
$\beta\,[-]$	0.7	0.5	0.699951	8000	0.001	7.52×10^{-15}
$\alpha_k \left[\frac{cm}{d}\right]$	0.1	0.6	0.100046			

Table 4.10: FINN pass with learnable f, k_d, α_k, β, (*) values rounded at 6th digit after decimal point, run c.

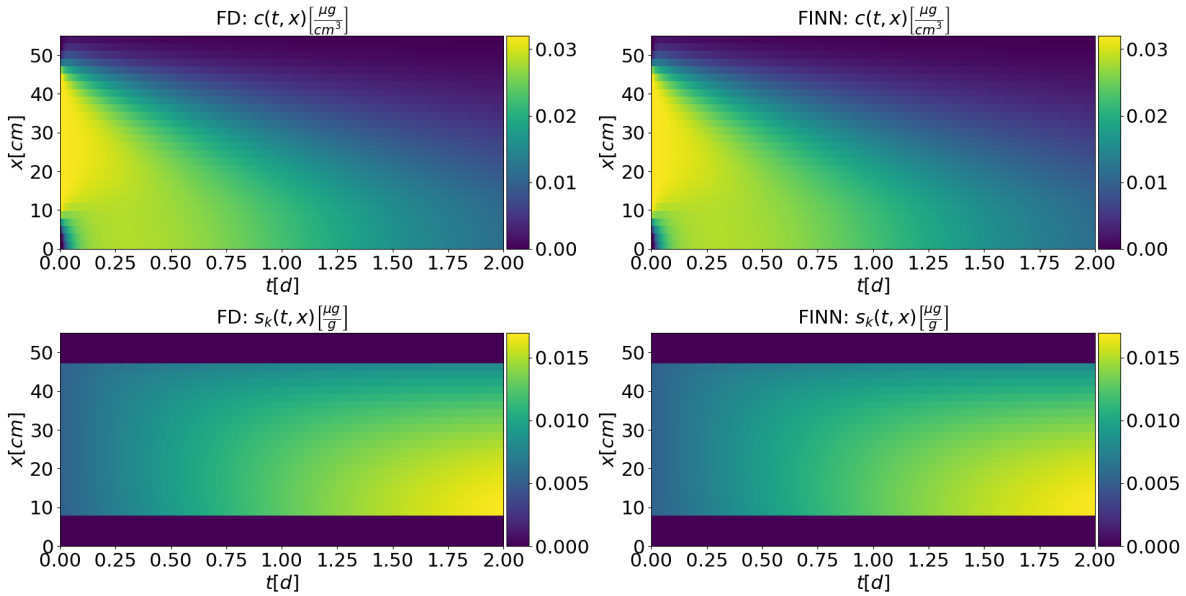

Figure 4.9: Comparison of FD and FINN solution, run c.

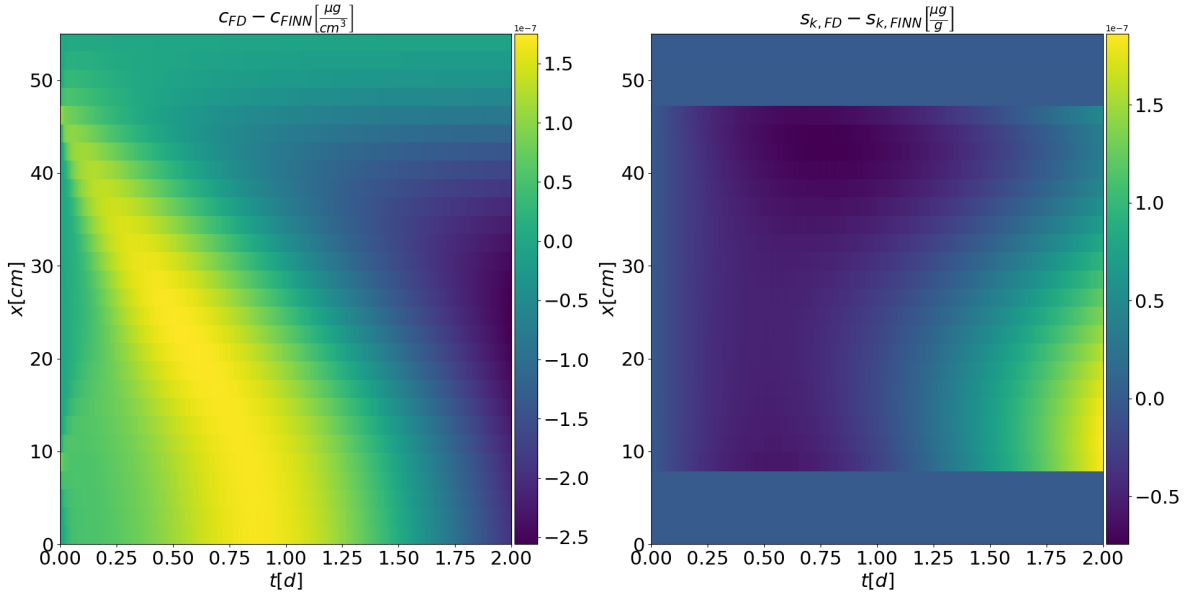

Figure 4.10: Difference of FD and FINN solution, run c.

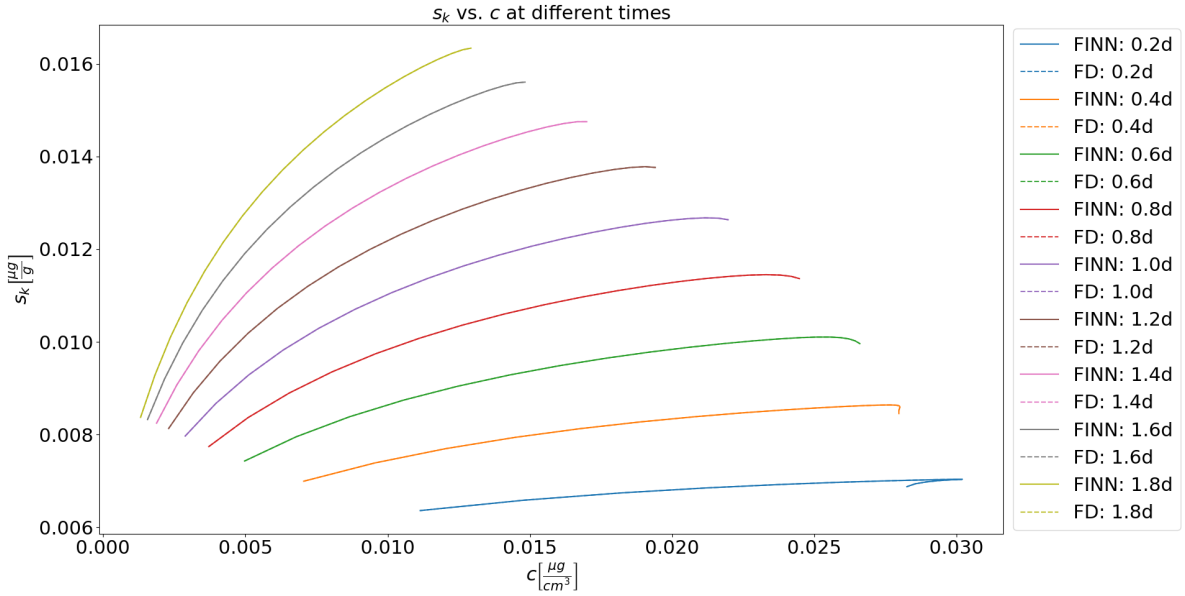

Figure 4.11: Comparison of FD and FINN solution: The unknowns s_k and c plotted at fixed time steps, run c.

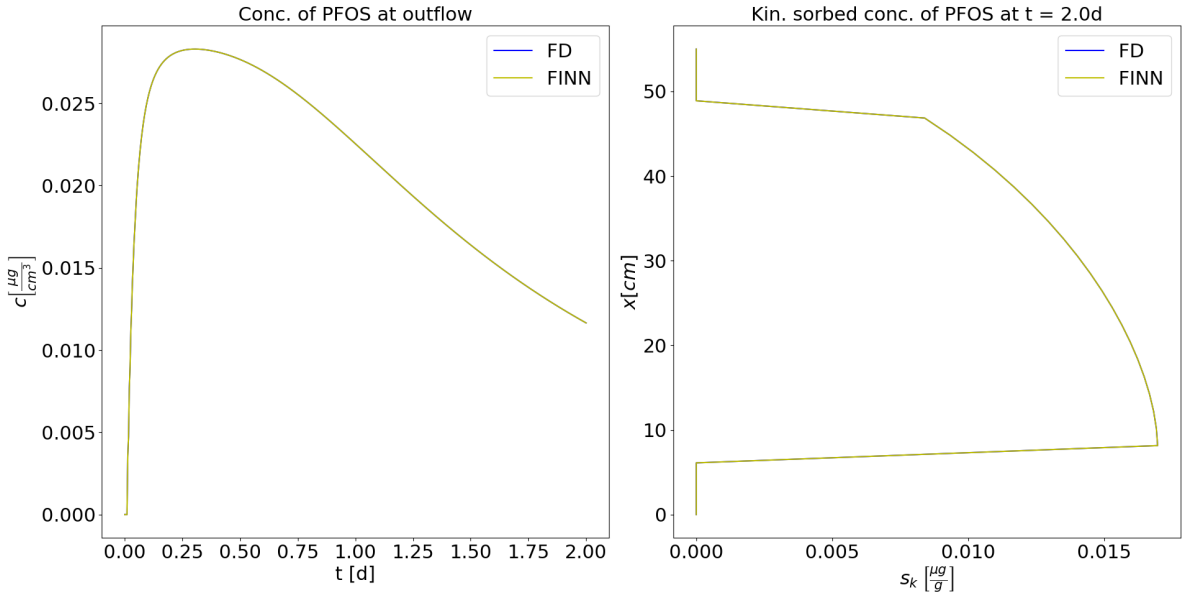

Figure 4.12: Comparison of FD and FINN solution: BTC of PFOS given by FD and approximated by FINN (left), s_k of PFOS at t_{end} (right), run c.

4.2.2 Learning Functional Relations

FINN was trained based on synthetic data for $0.5d$ and then tested with the remaining $t = 1.5d$, which is an *in-dis-test*. Each function F, G and R was approximated by a time-dependent DNN with two input features, one output feature, and three hidden layers with 20-40-20 neurons respectively. Thus, with biases and scaling term

$$\#\theta = \underbrace{2 \cdot 20 + 20 \cdot 40 + 40 \cdot 20 + 20 \cdot 1}_{weights} + \underbrace{20 + 40 + 20 + 1}_{biases} + \underbrace{1}_{scale} = 1742 \tag{4.2}$$

parameters were used per function.

Run d: Learn Time-dependent F
FINN produced good results learning only the function F described in eq. (3.76), (Tab. 4.11, Fig. 4.13). The loss value after training was two orders of magnitude smaller than the loss value of the test. At time points greater than $t = 0.5d$, the difference between the FD solution and the FINN prediction increased. $s_{k,FD}$ is greater than $s_{k,FINN}$ for the entire simulation time, thus c_{FD} is smaller than c_{FINN}, since more PFOS is in kinetically sorbed phase (Fig. 4.14).

run d learn	init scalings	Learning schedule epochs	lr	MSE_{train}	MSE_{test}
F	$f_{sc} = 0.1$	300	0.01	4.42×10^{-9}	1.94×10^{-7}

Table 4.11: FINN training and testing with unknown F, run d.

The BTC was approximated better in training than in testing. The tested model overestimates c at the outflow. Mass conservation seems to be plausible: In the FINN solution c is larger compared to the FD solution. Thus also the instantaneously sorbed concentration, calculated by the Freundlich sorption isotherm eq. (3.8) is larger. Hence, s_k predicted by FINN is smaller compared to the FD solution (Fig. 4.15).
The concave course of the c - s_k sorption isotherms in the FD solution was also reproduced by FINN. The sharp edge at $t = 0.2d$ is caused by the non-monotonous course of s_k and c over contaminated soil cells. With increasing time, especially at testing time these c - s_k sorption isotherms are predicted worse, due to less s_k in the FINN predicted solution (Fig. 4.16).

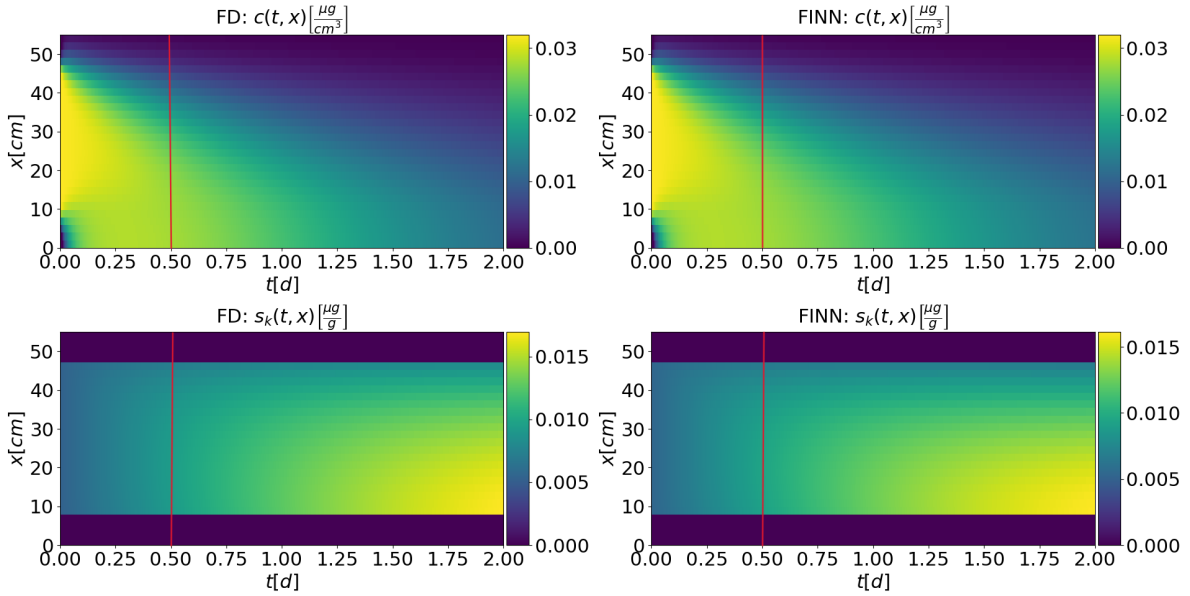

Figure 4.13: Comparison of FD and FINN solution: Red line marks the end of the training data set, run d.

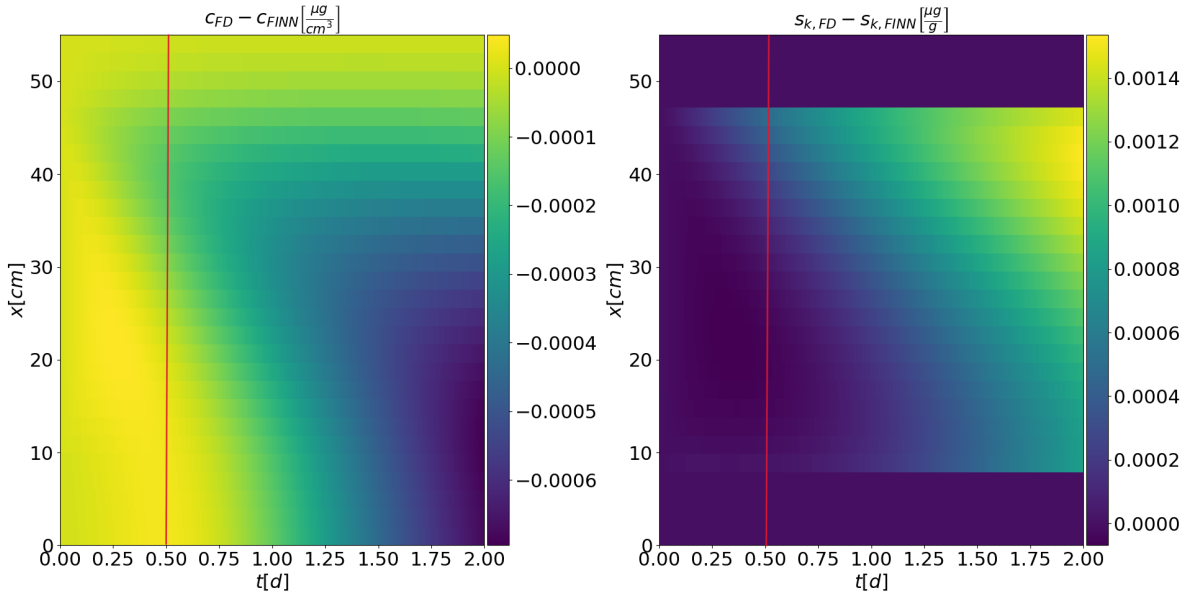

Figure 4.14: Difference of FD and FINN solution: Red line marks the end of the training data set, run d.

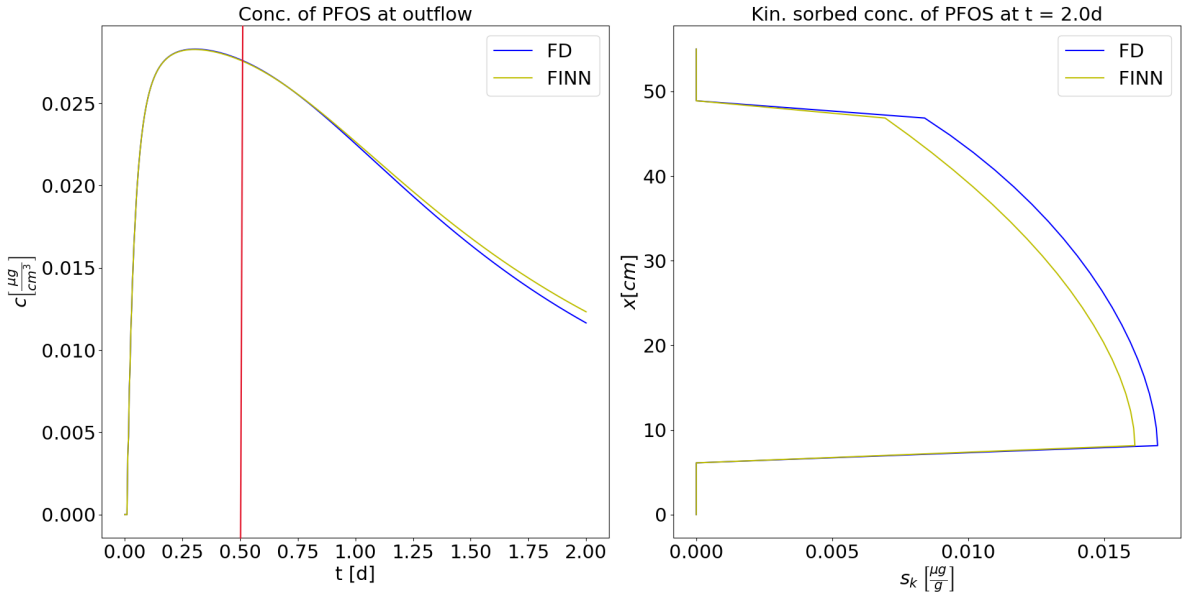

Figure 4.15: Comparison of FD and FINN solution: BTC of PFOS given by FD and approximated by FINN (left), s_k of PFOS at t_{end} (right), run d.

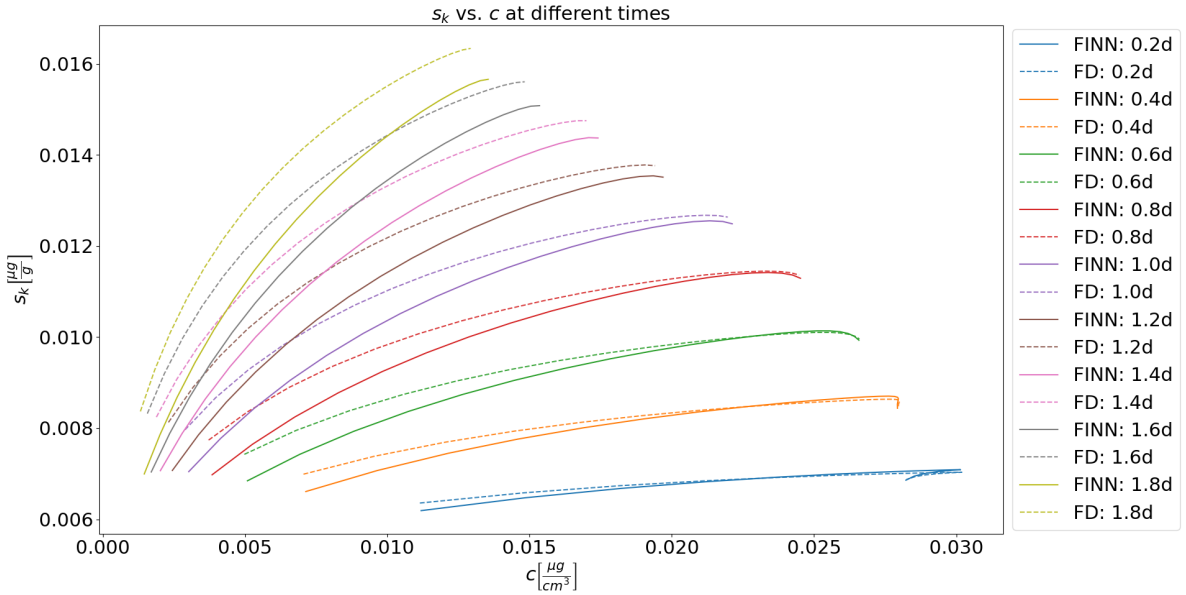

Figure 4.16: Comparison of FD and FINN solution: The unknowns s_k and c plotted at fixed time steps, run d.

Run e: Learn Time-dependent F, G and R
In a next step F eq. (3.76), G eq. (3.77) and R eq. (3.78), thus the entire sorption process were learned as explicitly time-dependent functions (Tab. 4.12, Fig. 4.17). The loss values for training and testing were in the same orders of magnitude. Compared to run d the loss after training is two orders of magnitude larger. Probably the model in run d overfitted the data.

run e learn	init scalings	Learning schedule epochs	lr	MSE_{train}	MSE_{test}
F	$f_{sc} = 0.1$	500	0.01	5.66×10^{-7}	
G	$g_{sc} = 0.0001$	100	0.001	5.49×10^{-7}	8.57×10^{-7}
R	$r_{sc} = 1$				

Table 4.12: FINN training with unknown time-dependent functions F, G, R, run e.

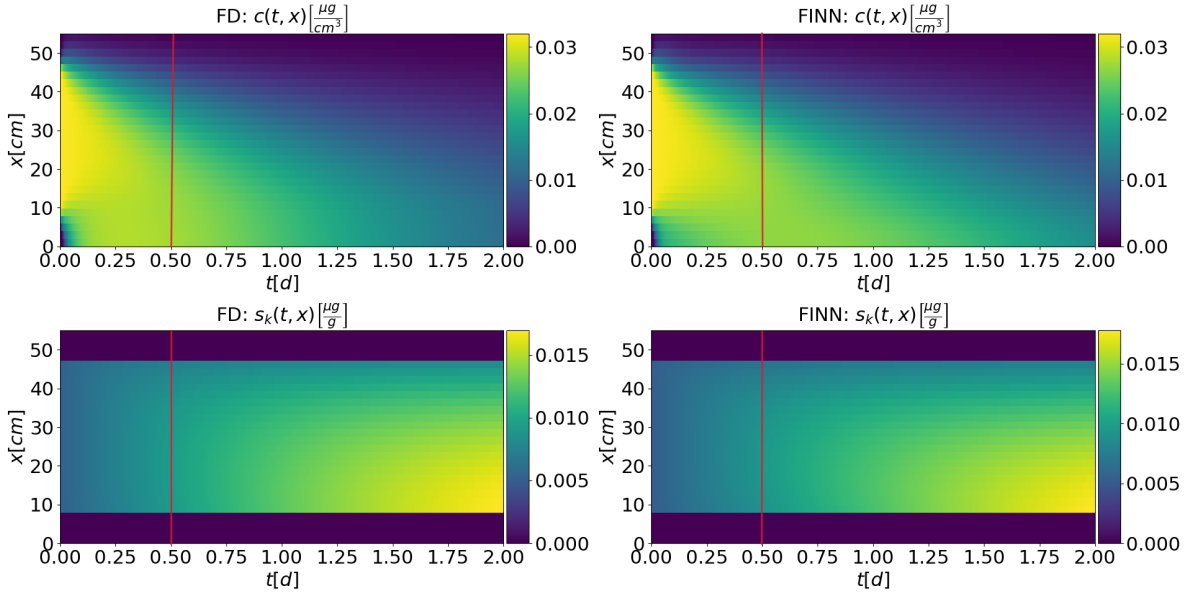

Figure 4.17: Comparison of FD and FINN solution: Red line marks end of the training data set, run e.

The model has difficulties at regions near the outflow and thus also in approximating the BTC (Fig. 4.18, left and Fig. 4.19, left). Using only the BTC data for loss calculation could improve this issue, which is done in the next steps for experimental data.
Interestingly we received for run d (Fig. 4.14, right)

$$|s_{k,FD} - s_{k,FINN}| \leq 0.0015 \qquad \forall (x,t) \in \Omega_x \times \Omega_t, \tag{4.3}$$

and for run e (Fig. 4.18, right)

$$|s_{k,FD} - s_{k,FINN}| \leq 0.0008 \qquad \forall (x,t) \in \Omega_x \times \Omega_t. \tag{4.4}$$

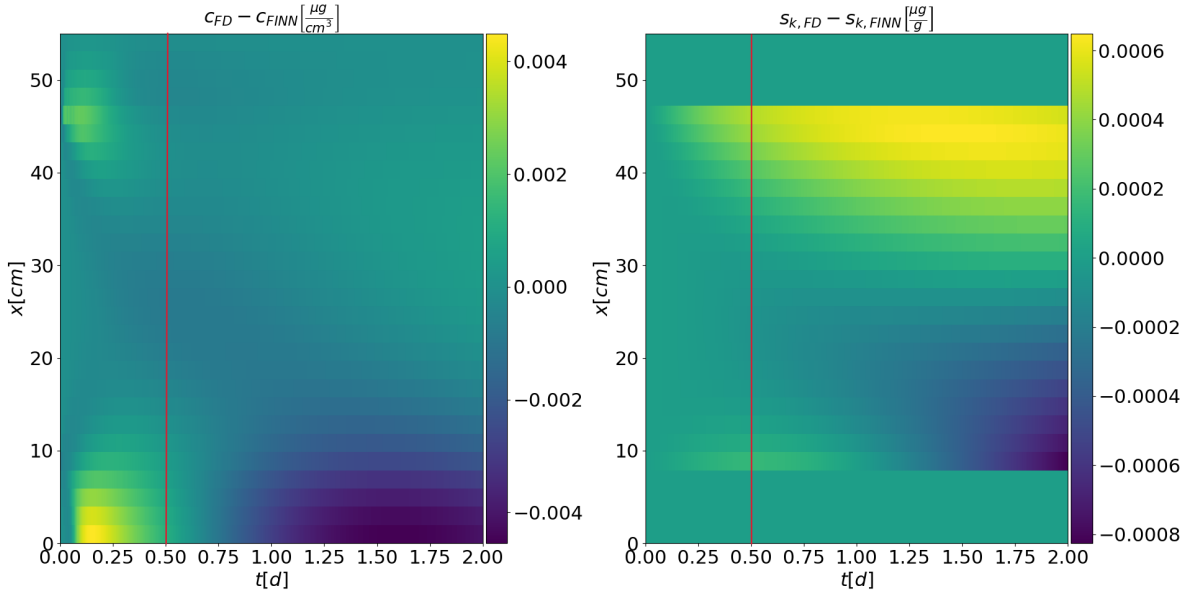

Figure 4.18: Difference of FD and FINN solution: Red line marks the end of the training data set, run e.

The absolute difference of $s_{k,FD}$ and $s_{k,FINN}$ is, compared to the model of run d smaller. The high loss value is mainly due to difficulties of the model in regions close to the outflow. Within the contaminated soil layer the model approximates the FD solution very well and better than in run d. Thus the c - s_k sorption isotherms are learned with higher accuracy in run e (Fig. 4.20).

There was also performed a run that learns functional relations with only 300 training epochs. Corresponding results are available in the appendix (Figs. 6.12, 6.13, 6.14, 6.15). Interestingly, at constant epoch numbers and learning rates, FINN seemed to perform better when all 3 functional relations F, G, and R and not only F were learned.

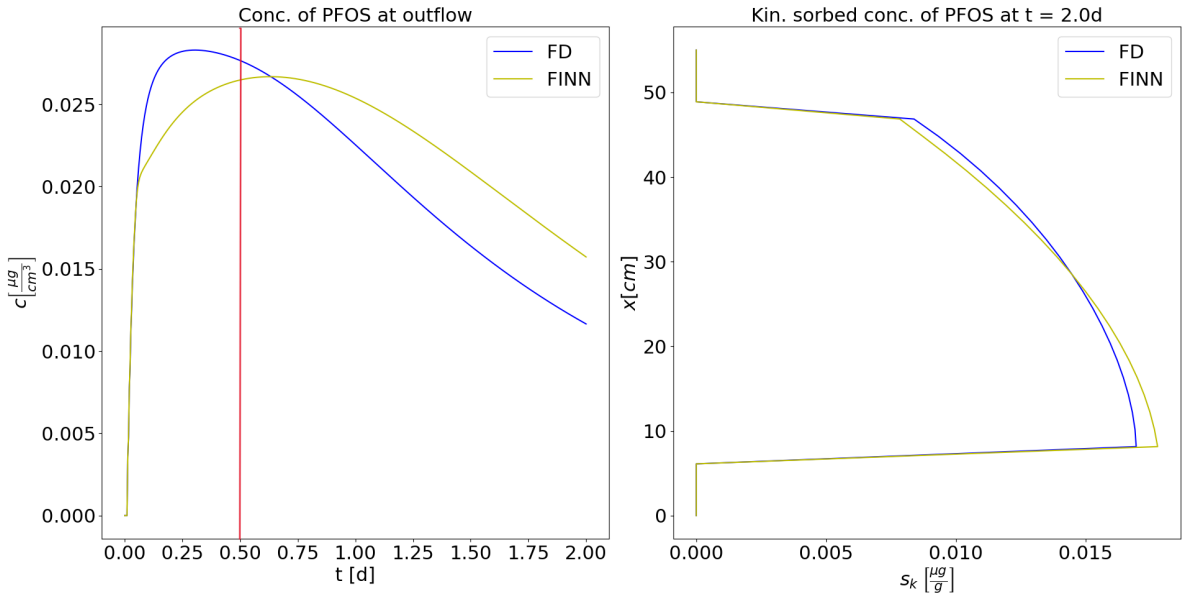

Figure 4.19: Comparison of FD and FINN solution: BTC of PFOS given by FD and approximated by FINN. Red line marks end of the training data set (left). Kin. sorbed concentration of PFOS at t_{end} (right), run e.

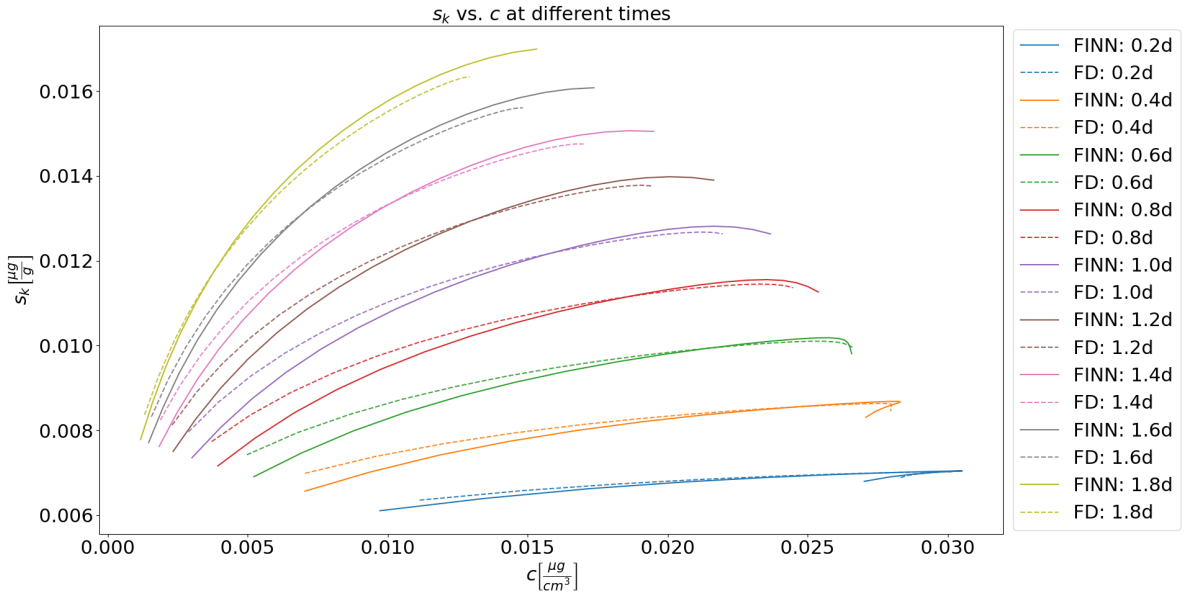

Figure 4.20: Comparison of FD and FINN solution: The unknowns s_k and c plotted at fixed time steps, run e.

4.3 Learning from Experimental Data

Performing loss calculation with data of experiment N1_1 or N1_3 of Bierbaum et al. also enabled training and testing of FINN models. In contrast to the experiment, the Darcy flux was assumed to be constant and averaged over the entire simulation time. Used material parameters for experiment N1_1 are described in Table 4.13 and discretization parameters in Table 4.14.

	$n_e\,[-]$	$\rho\left[\frac{g}{cm^3}\right]$	$D_e\left[\frac{cm^2}{d}\right]$	$\alpha_l\,[cm]$	$c_{init}\left[\frac{\mu g}{cm^3}\right]$	$s_{k,init}\left[\frac{\mu g}{g}\right]$	$q\left[\frac{cm}{d}\right]$
sand	0.31	NaN	0	5	0	0	31.68
soil	0.4	1.58	2.5	9	0.032	0.0053	31.68

Table 4.13: Material parameters of N1_1 for FINN.

$T_{MAX}\,[d]$	$T_{STEPS}\,[-]$	$X_{LENGTH}\,[cm]$	$X_{STEPS}\,[-]$	top	bot
141.76	38000	55	28	4	24

Table 4.14: Discretization parameters for FINN.

4.3.1 Learning Parameters

In order to reproduce work of Bierbaum et al. with FINN, the parameter f as ratio between kinetically sorbed and instantaneously sorbed concentration was investigated. The remaining non-learnable model parameters are given in Table 4.15.

	$k_d\left[\frac{cm^3}{d}\right]$	$\beta\,[-]$	$\alpha_k\left[\frac{1}{d}\right]$
soil	4.5	0.98	0.005

Table 4.15: Model parameters for FINN, run f.

Run f - Learn f
As initial guess the optimized f_{init} of Bierbaum et al. was used. Using the L_I loss function, an optimal $\hat{f}$ was assumed after 99 of 100 epochs (Tab. 4.16, Figs 4.21, 4.22). However, the values found for f were very similar.

In addition to BTC data, experimental data also contained the course of the totally sorbed concentration s, which is the sum of s_k and s_e at the end of the experiment, which was approximately homogeneously distributed. The $\hat{f}$ found by FINN approximated this course worse than f_{init} (Fig. 4.23, right). Since $\hat{f}$ of FINN is smaller than the f_{init} of Bierbaum et al., the share of kinetically sorbed PFOS is higher ($f = 0$ corresponds to complete kinetic sorption), which is observable in the s_k - c sorption isotherm too (Fig. 4.24).

run f	value	epochs	lr	L_I
$f_{init}[-]$	0.93	/	/	2.64×10^{-1}
$\hat{f}[-]$	0.9161	100	0.1	2.63×10^{-1}

Table 4.16: Comparison of optimal f of Bierbaum et al. (f_{init}) with FINN predicted ($\hat{f}$). L_I is the loss for f_{init} and $\hat{f}$ respectively, run f.

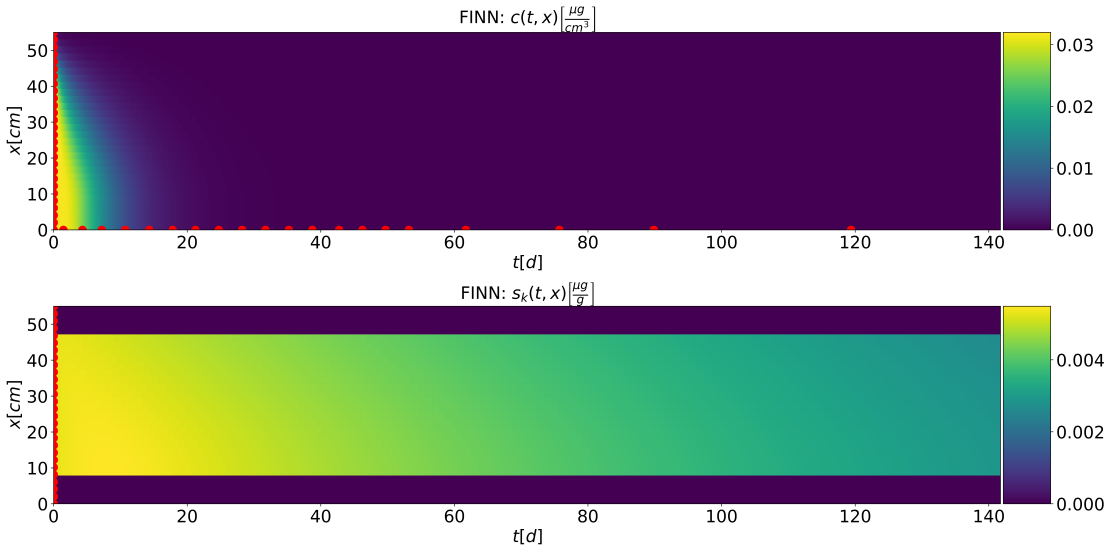

Figure 4.21: FINN prediction of c (top) and s_k (bottom) after training. Red points describe initial conditions and BTC concentrations of experiment N1_1 data, which were used for the loss calculation, run f.

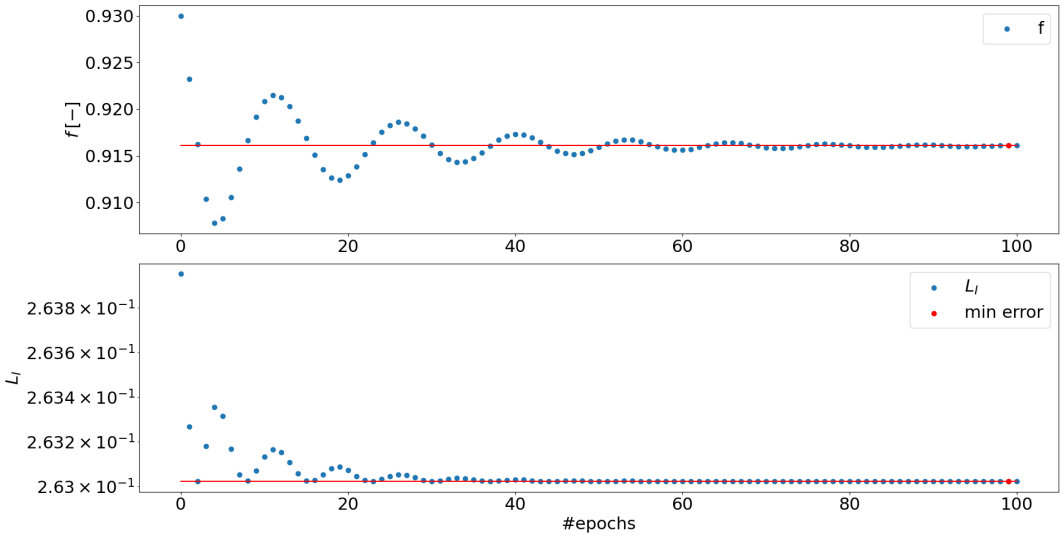

Figure 4.22: Course of predicted $\hat{f}$ (top) with corresponding L_I loss (bottom). Optimal $\hat{f}$ and minimal loss value are colored in red, run f.

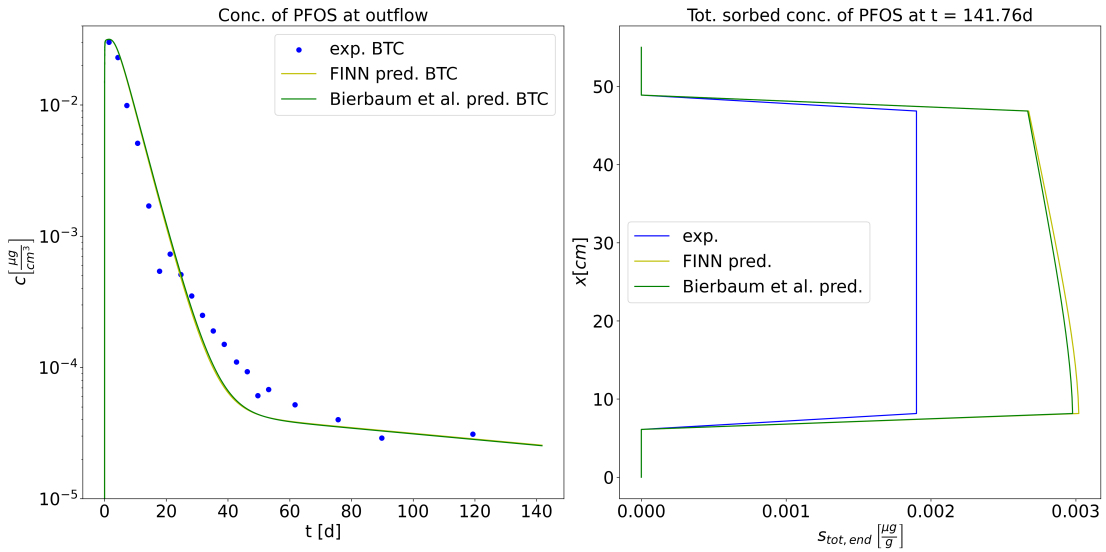

Figure 4.23: Optimizing f using experimental BTC (left). Comparison of FINN predicted parameter $\hat{f}$ and f_{init} with respect to experimental total sorbed concentration at the end of the experiment (right), run f.

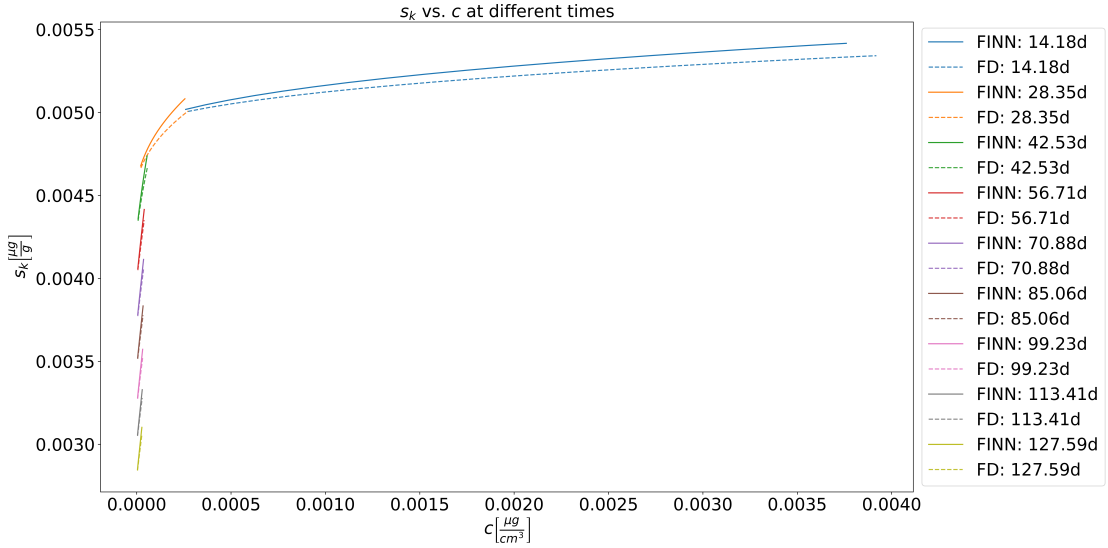

Figure 4.24: Comparison of FD solution with given f_{init} from Bierbaum et al. and FINN solution: The unknowns s_k and c plotted at fixed time steps, run f.

4.3.2 Learning Functional Relations

Run g - Learn Time-dependent F, G and R

Each function F, G and R, thus the entire sorption behavior of PFOS in experiment was approximated by a 2-100-1000-100-1 time-dependent DNN with biases and scaling term (Tab. 4.17, Fig. 4.25). Thus each function consists of

$$\#\theta = \underbrace{2 \cdot 100 + 100 \cdot 1000 + 1000 \cdot 100 + 100 \cdot 1}_{weights} + \underbrace{100 + 1000 + 100 + 1}_{biases} + \underbrace{1}_{scale} = 201502 \tag{4.5}$$

parameters. The number of parameters of the DNN for experimental data approximation was chosen larger than for synthetic data, since more time steps have to be done, which increases the number of input samples (t, c) or (t, s_k) respectively. Smaller architectures of DNNs could lead to data underfitting.

train learn	init scalings	Learning schedule epochs	lr	L_{II}	L_{III}
F	$f_{sc} = 0.1$	300	0.001	2.32×10^{-1}	0.00
G	$g_{sc} = 0.0001$				
R	$r_{sc} = 1$				

Table 4.17: FINN training with unknown time-dependent functions F, G, R, run g.

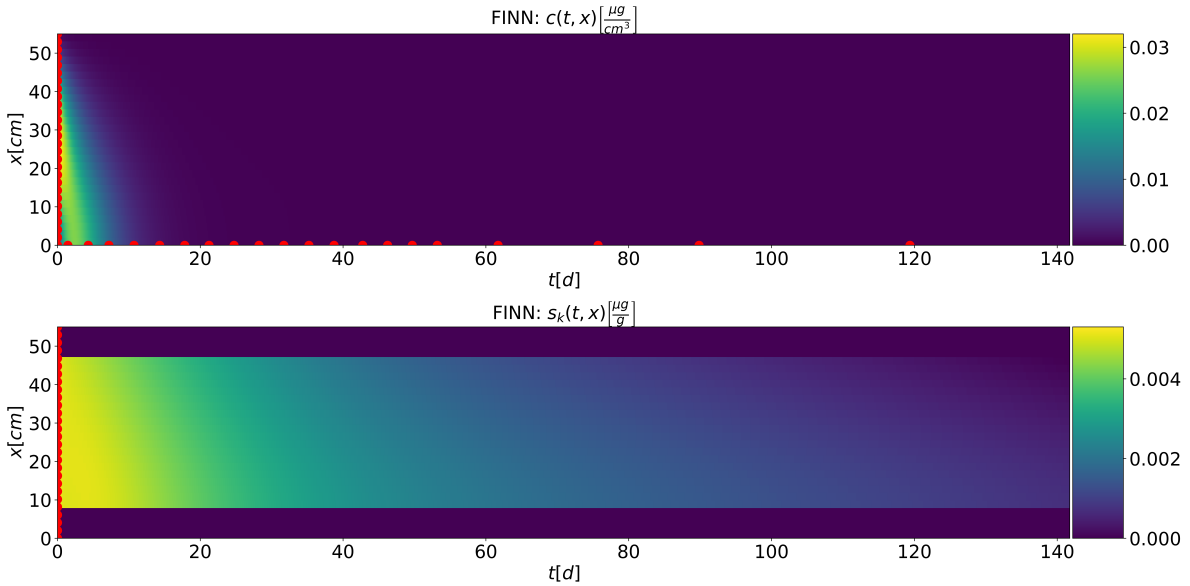

Figure 4.25: FINN prediction of c (top) and s_k (bottom) after training. Red points describe initial conditions and BTC concentrations of experiment N1_1 data, which were used for the loss calculation, run g.

Under the condition that predicted concentrations have to take positive values, the calculated combined loss value during training was lowest after 116 of 300 training epochs. For larger numbers of epochs, the scaling factor introduced as a punishment for negative learned concentration values described in eq. (3.85) was too small (Fig. 4.26).

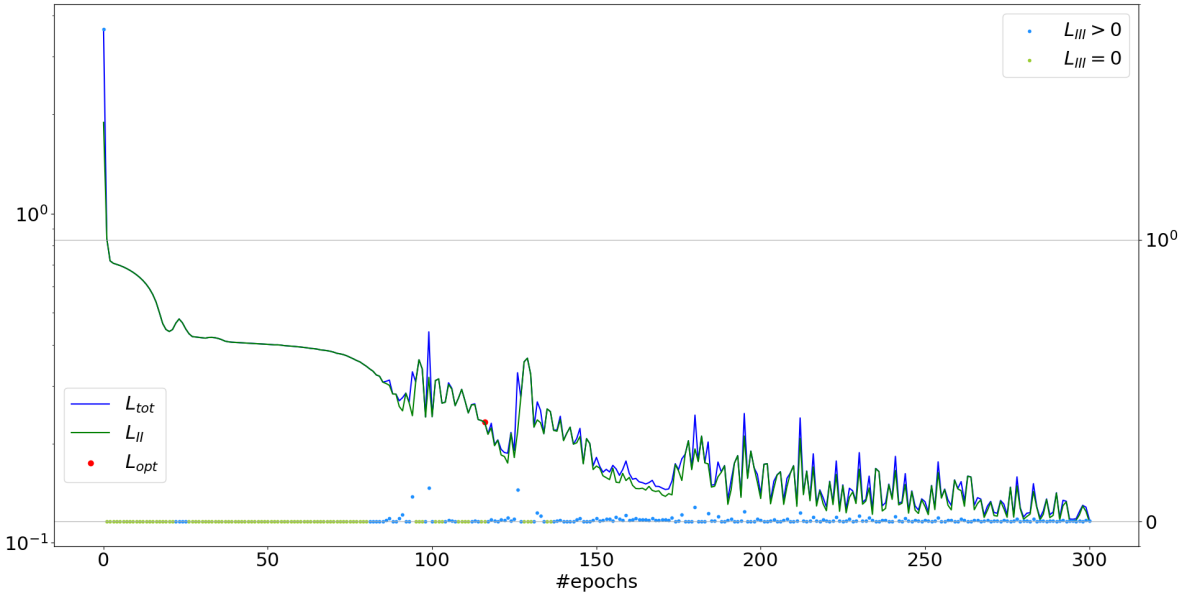

Figure 4.26: Course of L_{tot} and L_{II} during training with corresponding L_{III} loss values. The red point indicates lowest loss with $L_{III} = 0$, run g.

The course of the BTC was reproduced well. The largest deviations occured between $t = 20d$ and $t = 50d$ (Fig. 4.27, left).
FINN did not use total sorbed concentrations at the end of the experiment in the loss function because of unknown kinetical/instantaneous sorption ratios. However, the predicted course of s_k at t_{end} is reasonable, since $s = s_e + s_k$. Due to physical knowledge, e.g. the Freundlich sorption isotherm, s_e takes positive values which decrease with decreasing c. It is plausible that s_e is the remaining difference between s and s_k (Fig. 4.27, right).

Experiment N1_3 was used as *out-dist* data to test the model learned from N1_1 (Tab. 4.19, Fig. 4.28). The changing Darcy flux during experiment was again averaged, and varies from experiment N1_1, which also changed advection and dispersion quantities. The initial dissolved and kinetically sorbed concentration were guessed as described in chapter 3.3.4 (Tab. 4.18).
The tested model produced good BTC results with respect to the N1_3 experimental BTC data (Fig. 4.29, left), but reproduced non-physical results assuming experimental data at the end of the experiment as ground truth, since $s_k > s$, (Fig. 4.29, right). The predicted BTCs of trainig and testing are summarized in Fig. 4.30.

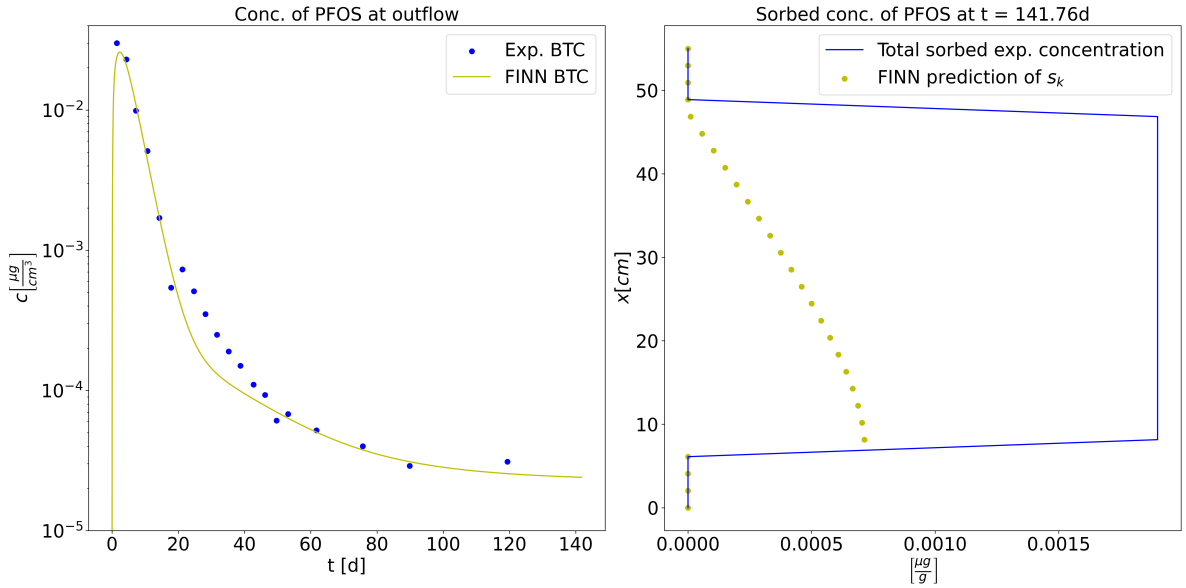

Figure 4.27: BTC approximation after training with time-dependent DNNs for approximating sorption behavior using experimental BTC data (left). Comparison of FINN predicted s_k and experimental total sorbed concentration s at the end of the experiment (right), run g.

	ρ_s	$n_e[-]$	$q\left[\frac{cm}{d}\right]$	$c_{init}\left[\frac{\mu g}{cm^3}\right]$	$s_{k,init}\left[\frac{\mu g}{g}\right]$	$s_{k,end}\left[\frac{\mu g}{g}\right]$	$T_{MAX}\,[d]$	$T_{STEPS}\,[-]$
soil	1.44	0.45	9.05	0.030	0.0098	0.0031	160	50000

Table 4.18: Adapting parameters according to experiment N1_3 for testing learned models, run g.

test	learned scalings	Learning schedule L_{II}	L_{III}
F	$f_{sc} = 0.0181$	3.60×10^{-1}	0.00
G	$g_{sc} = -0.0049$		
R	$r_{sc} = 1.0545$		

Table 4.19: FINN test pass with unknown time-dependent functions F, G, R, run g.

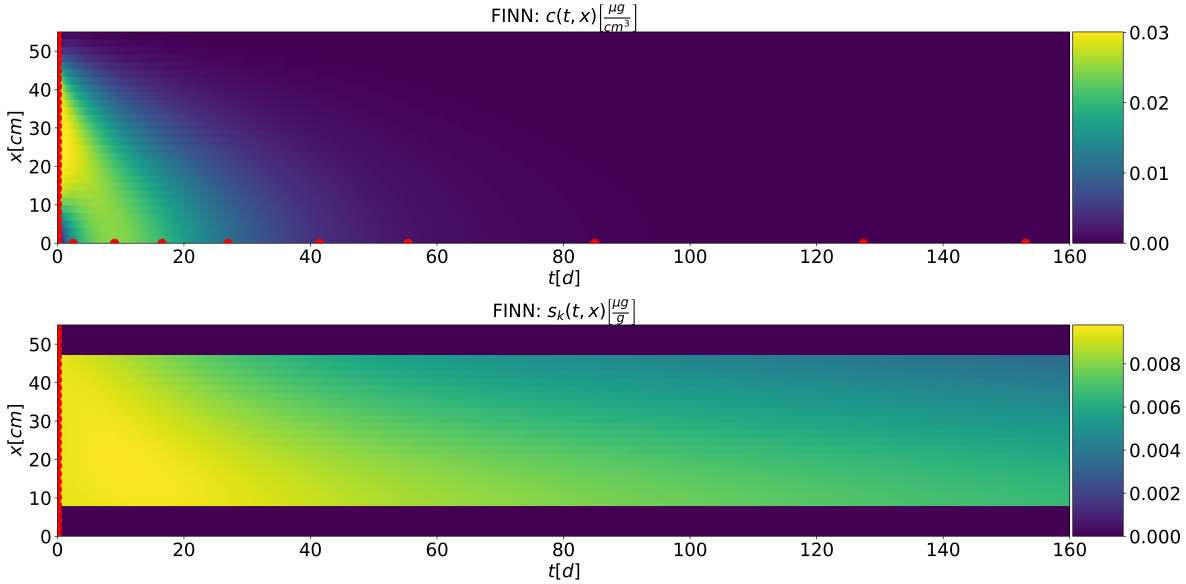

Figure 4.28: FINN prediction of c (top) and s_k (bottom) after testing. Red points describe initial conditions and BTC concentrations of experiment N1_3 data, which were used for the loss calculation, run g.

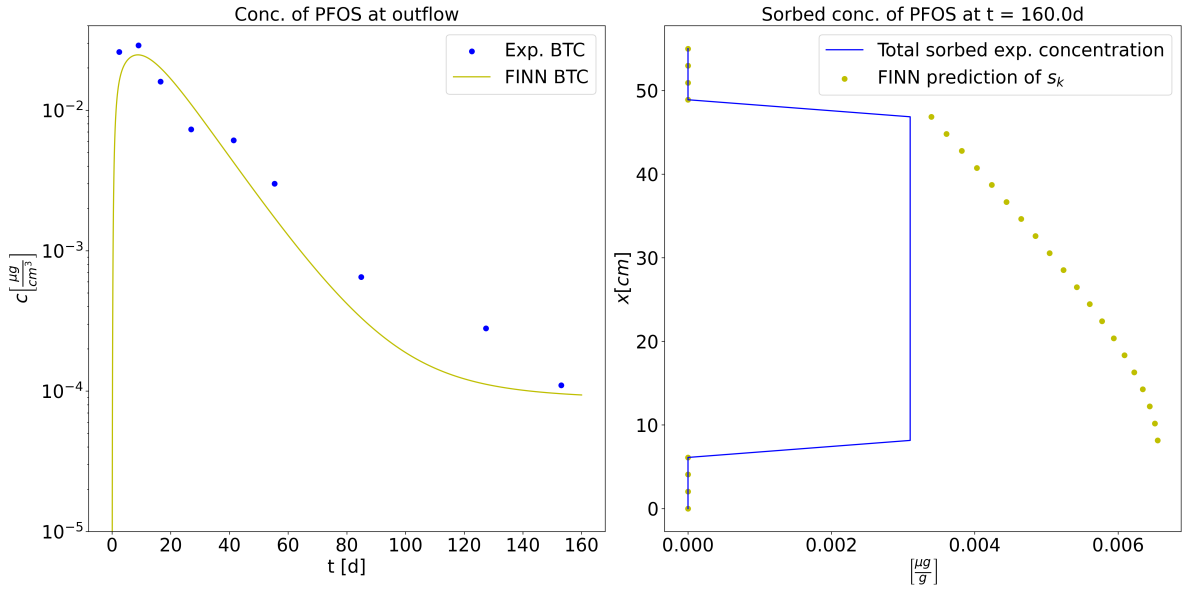

Figure 4.29: BTC approximation after *out-dis-test* (left). Comparison of *out-dis-test* FINN predicted s_k and experimental total sorbed concentration s at the end of the experiment N1_3 (right), run g.

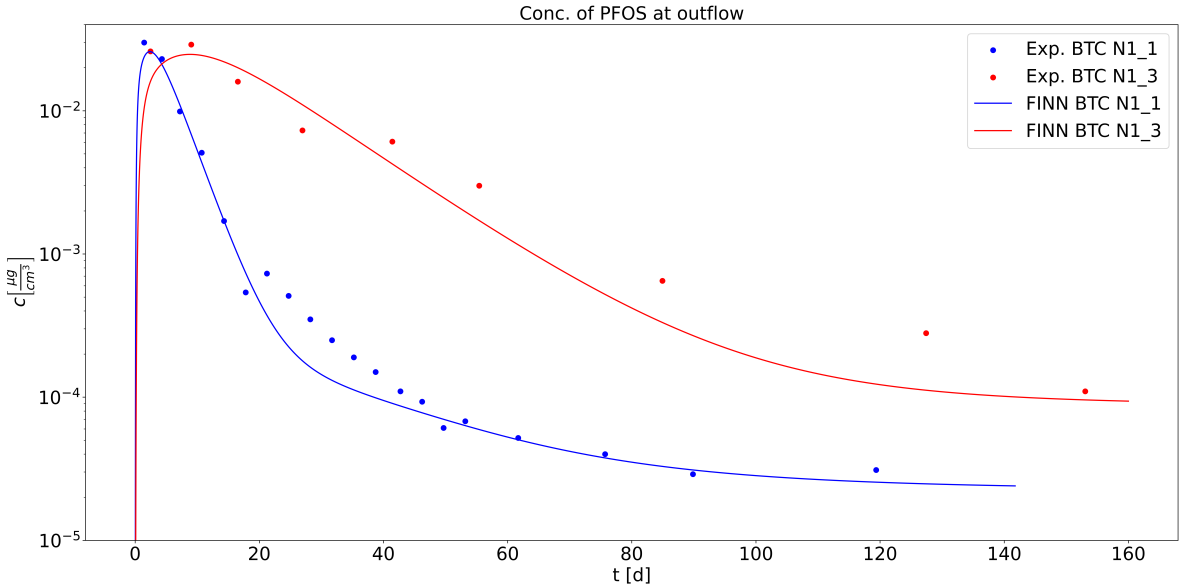

Figure 4.30: Comparison of FINN predicted BTCs: Model trained with experiment N1_1 (blue) and tested with experiment N1_3 (red), run g.

Run h - Learn Time-independent F, G and R
In a final scenario, only implicit time-dependent F, G and R were approximated (Tab. 4.20). The implicity is given because of the time dependency of c and s_k respectively.

train learn	init scalings	Learning schedule epochs	lr	L_{II}	L_{III}
F	$f_{sc} = 0.1$	300	0.001	3.90×10^{-1}	0.00
G	$g_{sc} = 0.0001$				
R	$r_{sc} = 1$				

Table 4.20: FINN training with unknown time-independent functions F, G, R, run h.

Each function was approximated by a DNN with one input feature, one output feature, and three hidden layers with 10-20-10 neurons respectively, which leads to

$$\#\theta = \underbrace{1 \cdot 10 + 10 \cdot 20 + 20 \cdot 10 + 10 \cdot 1}_{weights} + \underbrace{10 + 20 + 10 + 1}_{bias} + \underbrace{1}_{scale} = 462 \tag{4.6}$$

parameters.
Again, training was done with N1_1 experimental data (Fig. 4.31). As expected, run h gave slightly worse results in approximating the BTC than training the DNNs with explicit time-dependent models (run h: $L_{II} = 3.90 \times 10^{-1}$, run g: $L_{II} = 2.32 \times 10^{-1}$), (Fig. 4.32, left). The totally sorbed concentration at the end of the experiment was non-physically overestimated even in training (Fig. 4.32, right).
During training, negative concentrations were learned only in very few epochs (Fig. 4.33). The order of magnitude of the punishment factor included in L_{III} is reasonably chosen here. Since the loss value remained roughly constant over the last 150 epochs, a lower learning rate might have been used.

Interestingly, the *out-dis-test* with N1_3 data (Fig. 4.34) for time-independent DNNs produced good results at least for the approximation of the BTC (Fig. 4.35, left). Again, s_k was overestimated at the end of the experiment (Fig. 4.35, right).

test	learned scalings	Learning schedule L_{II}	L_{III}
F	$f_{sc} = 0.0089$	3.30×10^{-1}	0.00
G	$g_{sc} = -0.0003$		
R	$r_{sc} = 1.1673$		

Table 4.21: FINN test pass with unknown functions F, G, R, run h.

The approximated BTCs are summarized in (Fig. 4.36).

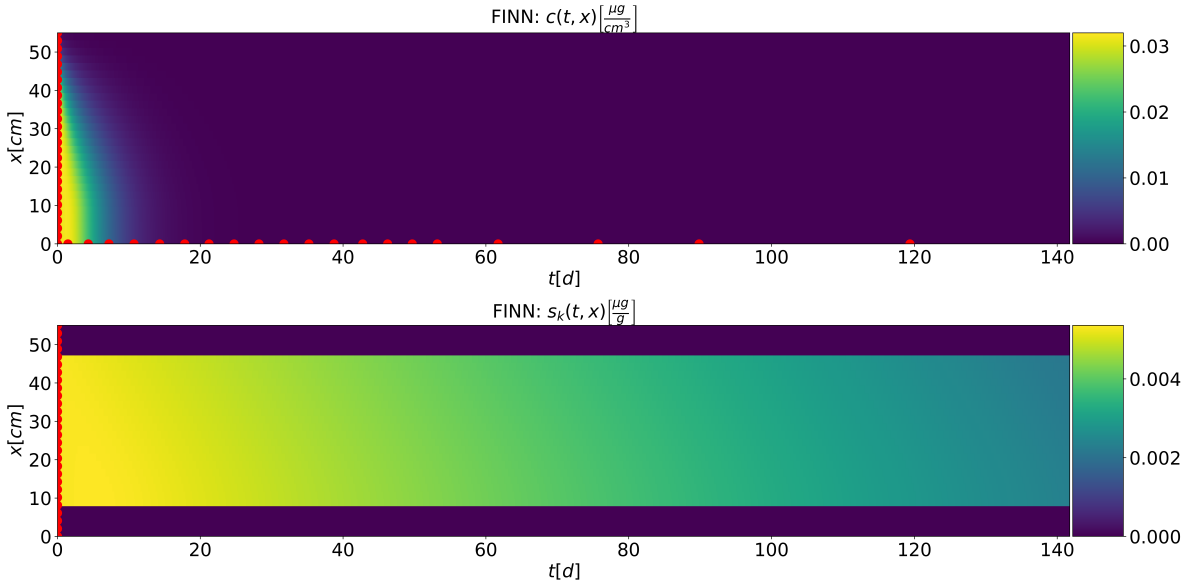

Figure 4.31: FINN prediction of c (top) and s_k (bottom) after training time-independent functional relations. Red points describe initial conditions and BTC concentrations of experiment N1_1 data, which were used for the loss calculation, run h.

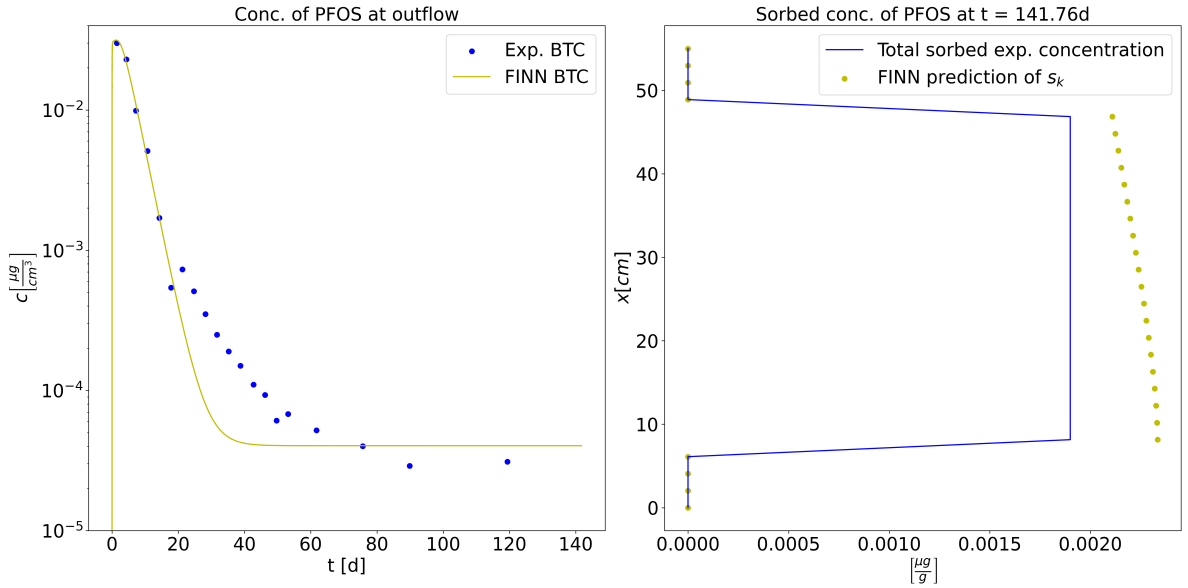

Figure 4.32: BTC approximation after training with time-independent DNNs for approximating sorption behavior using experimental BTC data (left). Comparison of FINN predicted s_k and experimental total sorbed concentration s at the end of the experiment (right), run h.

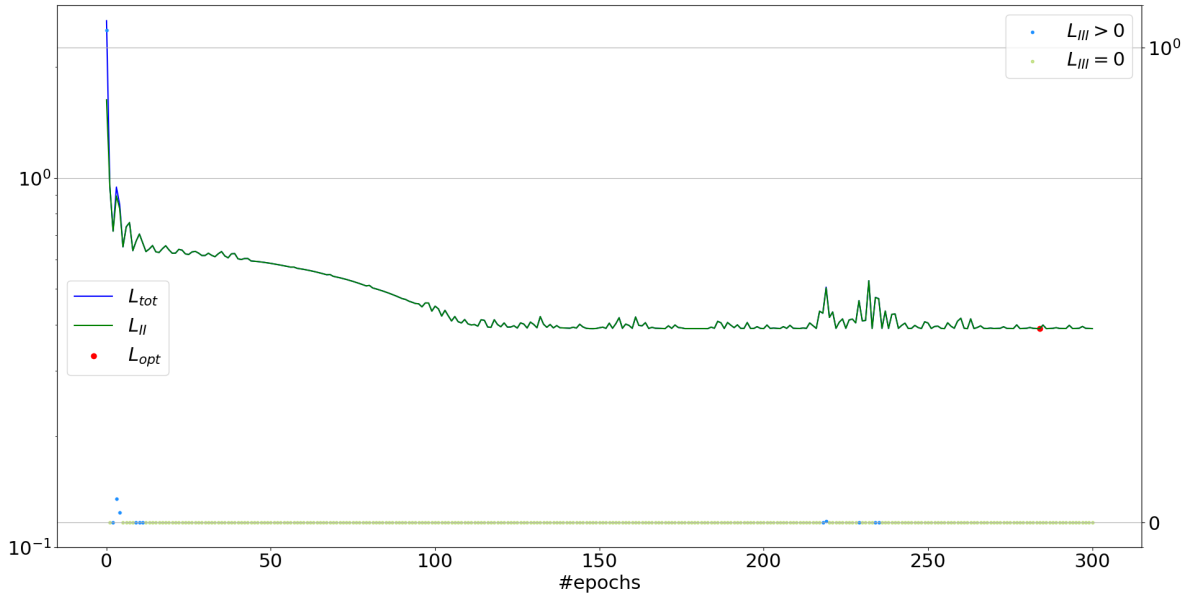

Figure 4.33: Course of L_{tot} and L_{II} during training with corresponding L_{III} loss values. The red point indicates lowest loss with $L_{III} = 0$, run h.

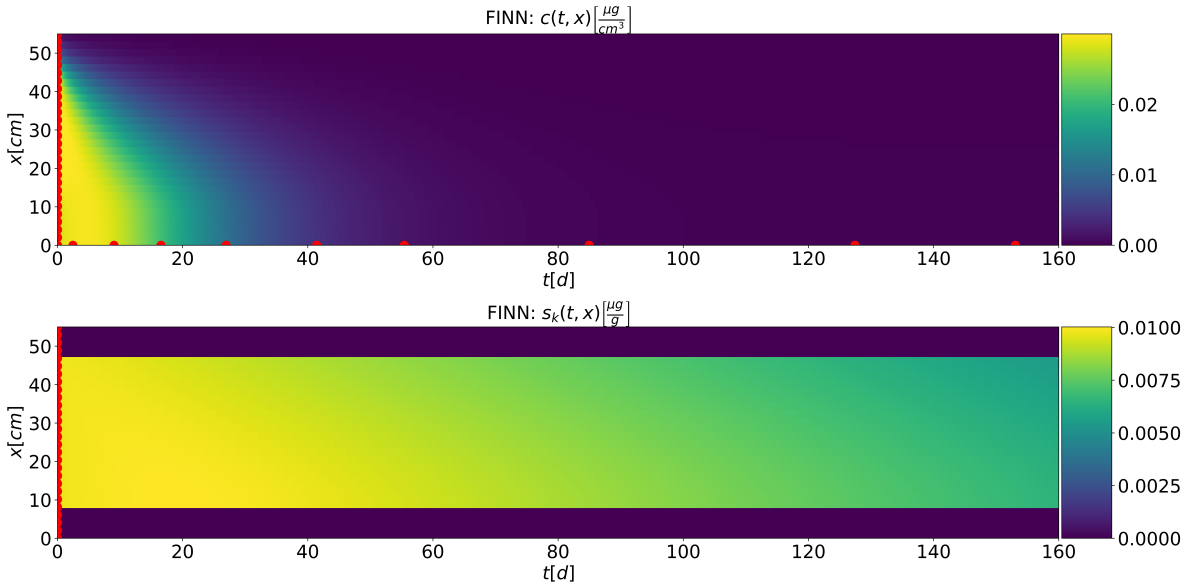

Figure 4.34: FINN prediction of c (top) and s_k (bottom) after testing time-independent functional relations. Red points describe initial conditions and BTC concentrations of experiment N1_3 data, which were used for the loss calculation, run h.

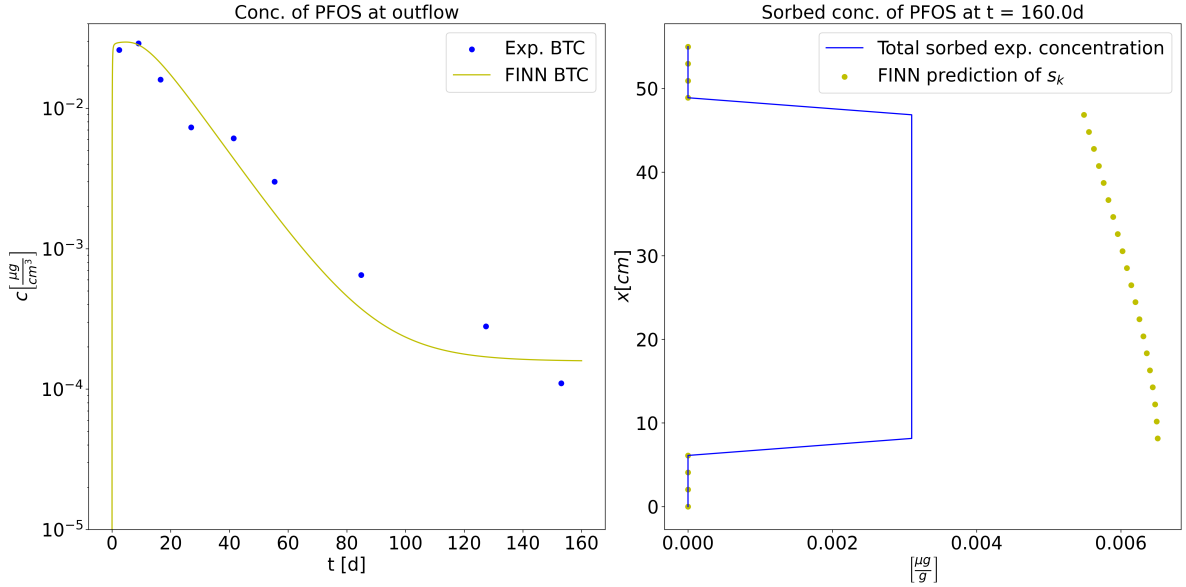

Figure 4.35: BTC approximation after *out-dis-test* (left). Comparison of *out-dis-test* FINN predicted s_k and experimental total sorbed concentration s at the end of the experiment N1_3 (right), run h.

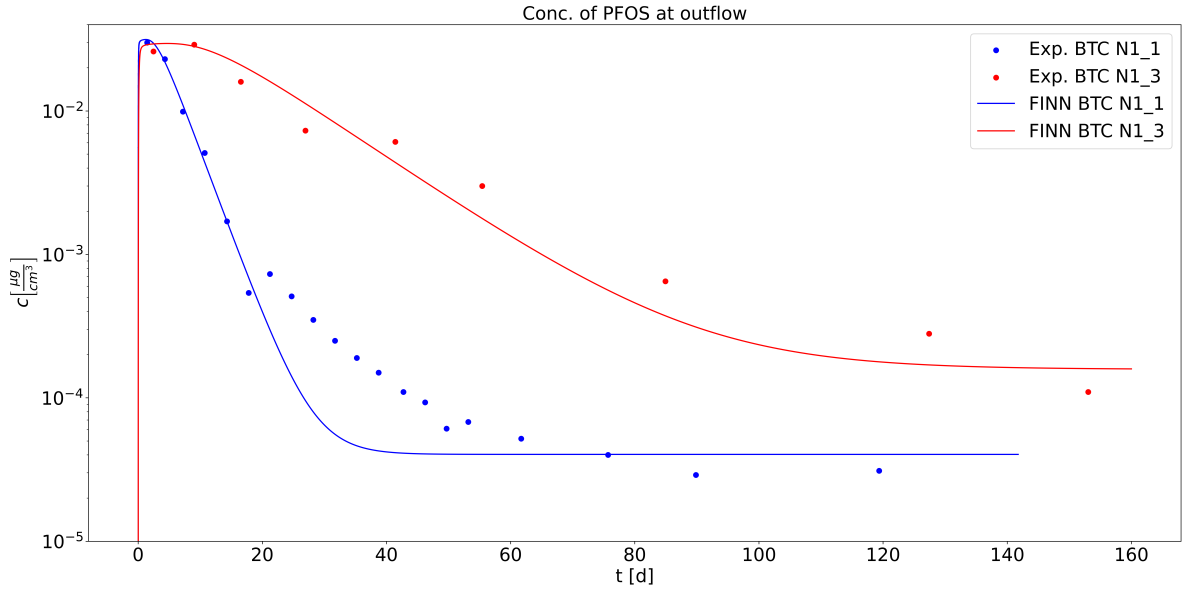

Figure 4.36: Comparison of FINN predicted BTCs: Model trained with experiment N1_1 (blue) and tested with experiment N1_3 (red), run h.

5 Discussion and Conclusion

As in the past chapters, the methods used and results produced are discussed step by step based on the three main contributions of this thesis.

5.1 Implementation of the FD Solver

First, anomalies in the implementation of the FD solver are discussed.

Non-negligibility of Sand Layers
To reasonably include the experimental data produced by Bierbaum et al. [19], the neglect of sand is not possible because of dispersion effects of PFOS into the sand layers. At the lower edge, the outward flow of PFOS is delayed due to the sand layer. This behavior was taken into account by Hydrus (Fig. 5.2) as well as in the implemented FD solver and FINN (Fig. 5.1).

Deviations between FD and Hydrus Solution
Hydrus can be used to simulate a wide variety of flow and transport processes, which can result in a large number of potential operating errors. Within the scope of this work, the Hydrus settings were extensively investigated, exchanged with the developers in the Hydrus forum and adapted to the parameters of the FD solution, but deviations still occurred.

In addition to errors in understanding, the handling of boundary conditions can be a reason for deviations. For example, in the FD solver it was assumed that in the case of no mass transfer between spatial cells, i.e. effective velocity $v = 0$ and molecular diffusion $D = 0$, no information exchange takes place across the cells, not even at the edge. Thus, each cell should describe the same process over time.
This assumption was tested using Hydrus with identical settings. In particular, the boundary conditions as described in Chapter 2 were adopted: Dirichlet boundary condition at the inlet and Neumann boundary condition at the outlet. It turned out that in Hydrus at the top cell, spatial mass transport occured (Figs. 5.3, 5.4, corresponding Hydrus file is available on GitHub [40]). This could be partly due to the fact that the treatment of boundary conditions and spatial discretization is handled differently in FE method compared to FD or FV method. However, the Fortran implementation of Hydrus was not considered further in this work.
It could be further investigated how an increased number of spatial discretization steps

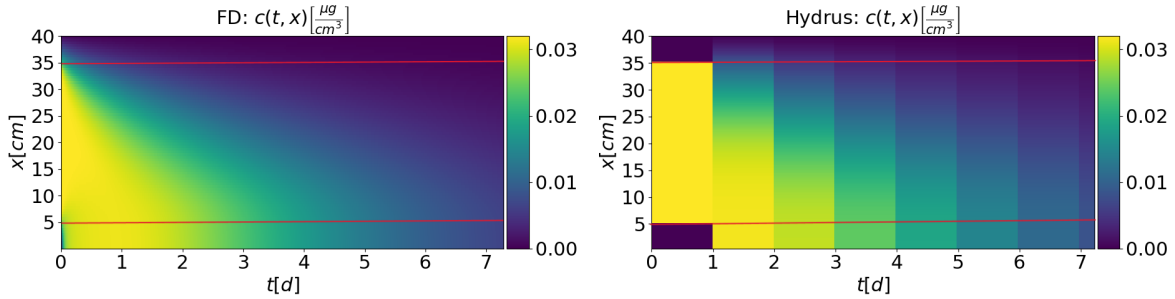

Figure 5.1: Closer look at comparison of Hydrus and FD solution with 2 sand layers. Hydrus solution (right), FD solution (left). First 7 days of 200 days total simulation time. Due to Hydrus limitations, data was only tapped at 200 points in time, which results in the rough temporal resolution. Internally, the time discretization of Hydrus is finer. Cells above the upper red line and below the lower red line are the sand layers. Dispersion of PFOS into the upper sand layer can be observed in both solutions.

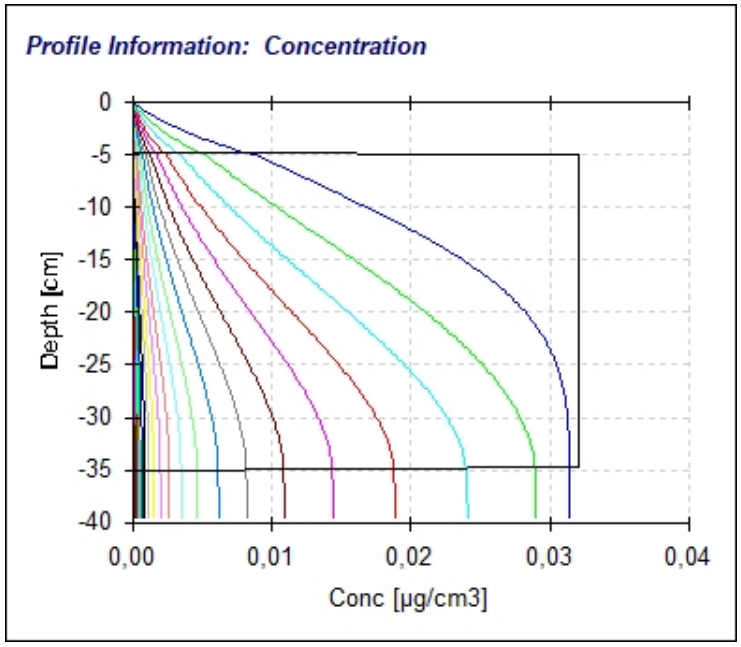

Figure 5.2: Closer look at the Hydrus solution, using Hydrus visualization tools. Colored are the courses of the dissolved concentration over the contaminated soil length at different times. Regions above and below the horizontal black line, which is the dissolved concentration at $t = 0d$, are sand layers. Dispersion of PFOS into the upper sand layer can be observed.

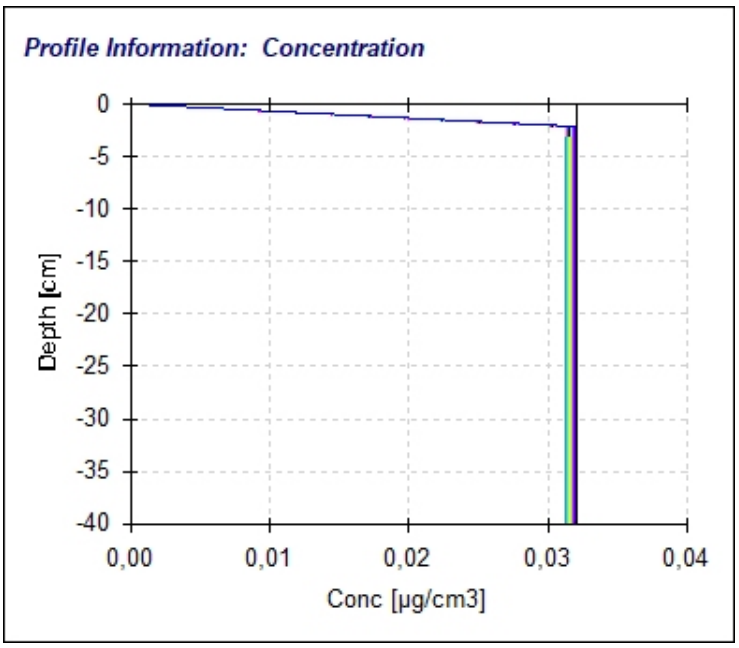

Figure 5.3: Dissolved concentration of a substance, simulated in Hydrus with no water flux. Colored: Different observation times.

affects the deviations in order to minimize effects of the different methods in spatial discretization.
In addition, Hydrus uses $f = 0.9999$ as the fraction between instantaneously and kinetically sorbed concentration in the sand layer, when in fact in the FD solver $f = 1.0$ applies, since no sorption occurs.
The MSE calculation also leads to small deviations: Due to the time discretization used in the FD method, values of the solution are not present at the exact same times as in the Hydrus solution (Fig. 5.1).

5.2 Embedding of the Transport Equation into the FINN Framework

Using DNNs to approximate F, G and R required many reflections about the architecture of the DNNs, the optimizer, or the loss function to choose, which is a fundamental challenge in Deep Learning. There is no formula with that the perfect number of hidden layers, or number of neurons can be calculated [41]. Also for the choice of activation functions, especially within the DNN, many possibilities are available, which can only be verified empirically and in a time-consuming manner. Of course, heuristics can be used to facilitate the process of the DNN design, which is done for example in DNN puring [42]. However, in the present work a very specific DNN was created, which computes unknown functions in a PDE model during time integration, where these heuristics can

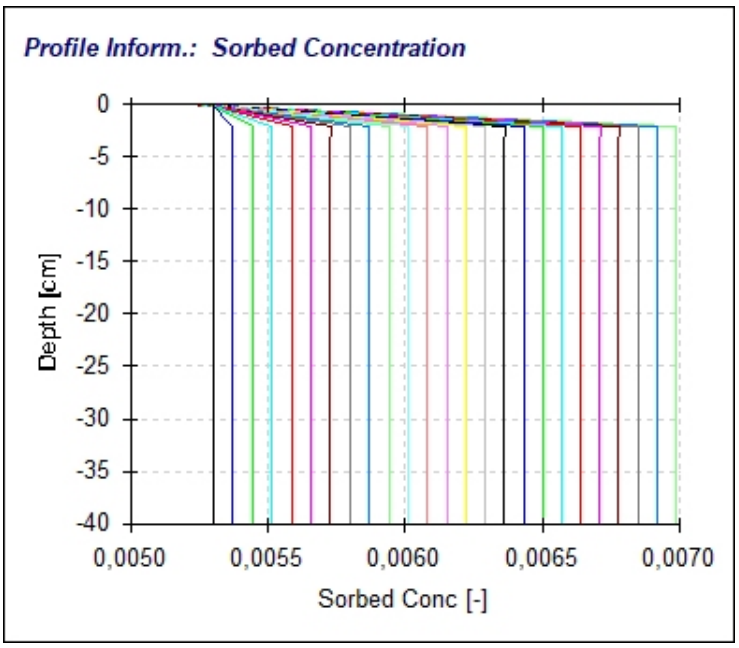

Figure 5.4: Kin. sorbed concentration with Hydrus settings as in Fig. 5.3. Colored: Different observation times.

only be applied to a limited extent. A larger number of used neurons and hidden layers, takes a longer computing time during training, as well as might overfit the training data if training occured for too many epochs, i.e. good training results but bad test results. Batch data sets as random parts of the input data set for the DNN, are commonly used during training to save computational effort when performing optimization. In this work this was also challenging, because for the loss calculation for each time step forward evaluations of the DNNs were needed. In the backward pass, time points and location points used during the integration of the spatially discretized PDE would have to be selected. For this, the torchdiffeq library [37] would have to be expanded. However, the use of no batches can prolong the training enormously, since for each of the $T_{STEPS} \cdot X_{STEPS}$ samples per DNN, corresponding gradients have to be computed. In particular, this process requires a lot of RAM.
The unusual usage of the DNNs is followed by the question of how much physical information must be made available to the DNN in order to produce useful results. In this work, the degrees of freedom of the DNN were kept rather low, in contrast to other pure machine learning NNs [43, 44]. To compensate scarce experimental data, in this work FINN was provided all physical information, in solving the PDE. No stencils were learned. Only parameters were optimized or functional relations occuring in the PDE were learned, which were even further constrained with physical information (Loss penalty for negatively learned c or s_k, $R \geq 1$, etc.). Also, the separation that each of the functional relations was approximated by its own NN can be considered as a physical information for FINN.
So-called Bayesian NNs [45] could provide a way to figure out which of the physical

information described above is necessary, leads to underfitting or overfitting, or may be incorrect by quantifying uncertainties of the predicted solution.
With PyTorch, it is also possible to move the training and testing of the DNNs to GPU without much additional effort. This leads to a significant time saving in the calculation and makes it easier to adjust the properties of the DNN [46].

5.3 Application of FINN on Experimental Data

To calculate the initial conditions from the mass balance eq. (3.80), parameters were used which, if necessary, still have to be learned, which is a circular argumentation. However, solving a PDE is not possible without initial conditions, and since the parameters f, k_d, and β according to Bierbaum et al. and run f, described experimental BTCs quite well, they were nevertheless used for the calculation of initial conditions.

It has not yet been clarified in the context of this work whether it makes sense to make the learnable functional relations explicitly time-dependent:
For the training procedures that based on synthetic data there is only given an implicit time dependancy in the model equations. Nevertheless, this approach was followed for testing purposes with the expectation, that the DNNs are still able to approximate these functions. They should learn a neglect of explicit time dependency.
Concerning training based on experimental data we knew that the experimental flow rates changed over the duration of the experiment at each sampling interval because a different pump rate was chosen or the pump tubing wore out [19]. In order to take these non-negligible fluctuations (Fig. 5.5) and also probably chemical reasons for changes of F, G and R explicitly in time into consideration, in the majority of the sessions an explicit time dependency was used. Future work could include a detailed investigation of this topic by for example learning non time-dependent functional relations but approximating the changes of experimental flow rates by another DNN as a function in time.
The overestimated s_k at the simulation end, learned in run h could be embedded in the loss function analogously to negatively learned concentrations and be punished.

5.4 Conclusion

In this work a FD solver for the Advection-Dispersion equation including the Two-Site model was implemented and validated with Hydrus. For 1D transport of PFOS, the solver produced plausible results for simulations without a sand layer and solutions close to the Hydrus values in contaminated soil sorrounded by two sand layers. The solution trajectories could be further used both to validate the FV procedure implemented in FINN and to provide synthetic training and test data.

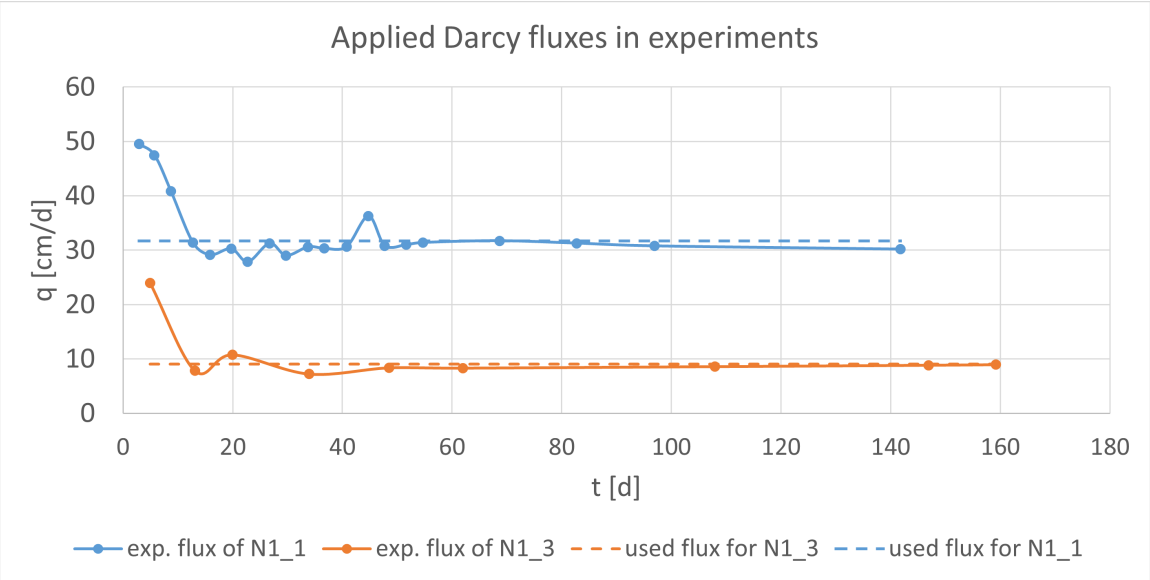

Figure 5.5: Darcy fluxes in experiments N1_1 and N1_3, with used averaged Darcy fluxes for training and testing of the experimental models.

FINN was able to reproduce the originally used model parameters of the synthetic solution to machine accuracy. Also functional relations provided good approximations for the c - s_k sorption isotherms after sufficient training epochs. With the loss function used for synthetic data, the BTC could only be approximated with errors for unknown sorption behaviors both in training and in *in-dis-tests*.
Based on scarce experimental measurements at the outflow, FINN was able to learn a model for physically reasonable sorption behaviors within the contaminated soil, that provided acceptable agreement with experimental BTCs data even under other physical conditions in *out-dis-tests*.

Besides topics mentioned in the discussion sections above, in future work the physically plausible predictions about the transport and sorption behavior of PFOS could be verified by generating further experimental data at spatial coordinates inside the column. Optimally, the dissolved concentration at the beginning of the experiment should be measured, so that values and distributions of the initial dissolved concentration cannot only be estimated, as in the present work.
In this work only PFOS was considered. In a next step, the behaviors of other non-degradable PFASs in contaminated soil could be predicted by using corresponding experimental outflow data. Further, by extending FINN to sorption models with multiple sorption sites or incorporating reaction terms, a wide variety of chemical non-equilibrium models can be learned.

6 Appendix

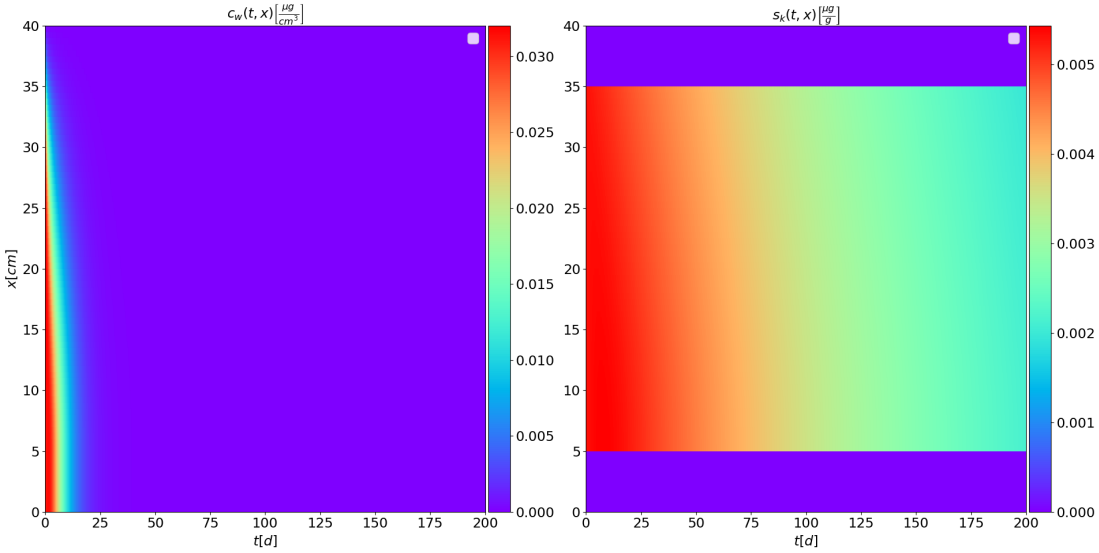

Figure 6.1: FD solution of the dissolved concentration (left) and kinetically sorbed concentration (right). Used parameters are $n_e = 0.4$, $\rho = 1.58\frac{g}{cm^3}$, $k_d = 4.5\frac{cm^3}{g}$, $\beta = 0.98$, $f = 0.93$, $\alpha_k = 0.005\frac{1}{d}$, $D_e = 2.5\frac{cm^2}{d}$, $\alpha_l = 9cm$, $v = 50\frac{cm}{d}$, $c_{init} = 0.032\frac{\mu g}{cm^3}$, $s_{k,init} = 0.0053\frac{\mu g}{g}$, $T_{MAX} = 200d$, $T_{STEPS} = 1000000$, $X_{LENGTH} = 40cm$, $X_{STEPS} = 80$, $top = 10$, $bot = 70$, $n_{e,sand} = 0.31$, $\alpha_{l,sand} = 5cm$, $v_{sand} = 64.516\frac{cm}{d}$, D was calculated as described in eq. (3.3).

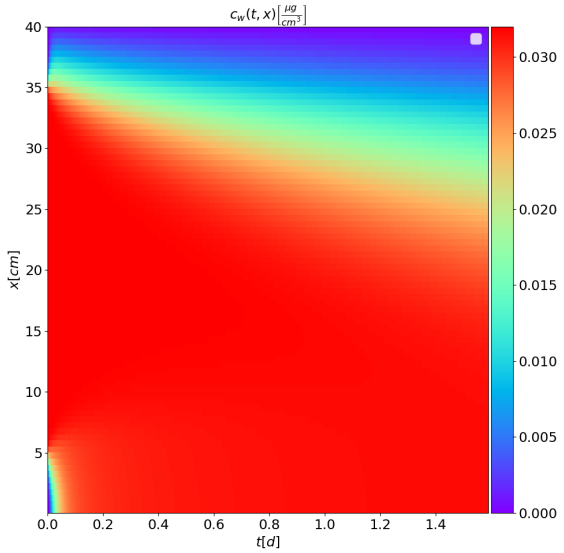

Figure 6.2: The first days of a FD solution. Parameters were the same as in Fig. 6.1.

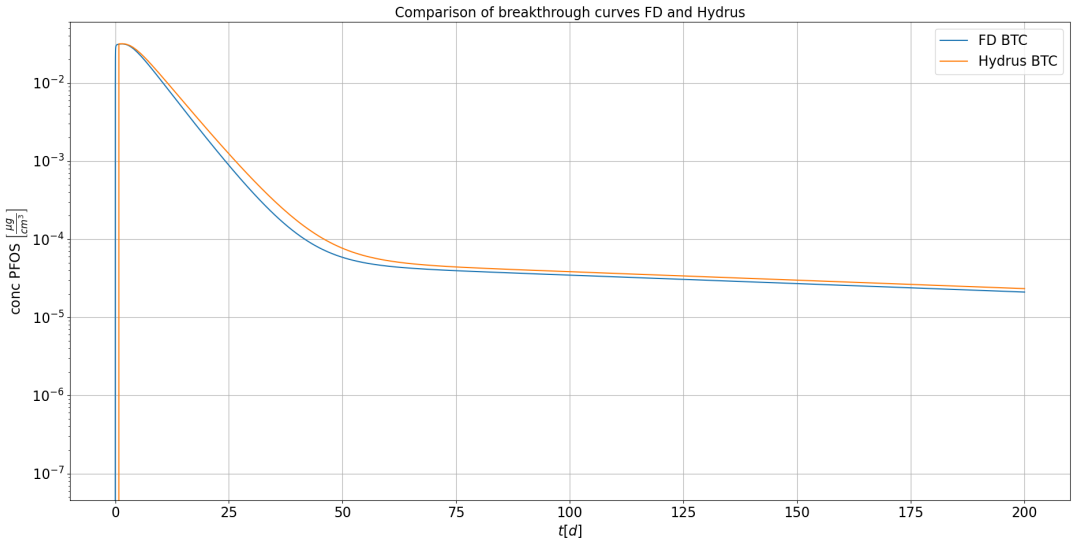

Figure 6.3: Comparison of BTCs, calculated by FD and Hydrus, sand layers included. Parameters were the same as in Fig. 6.1.

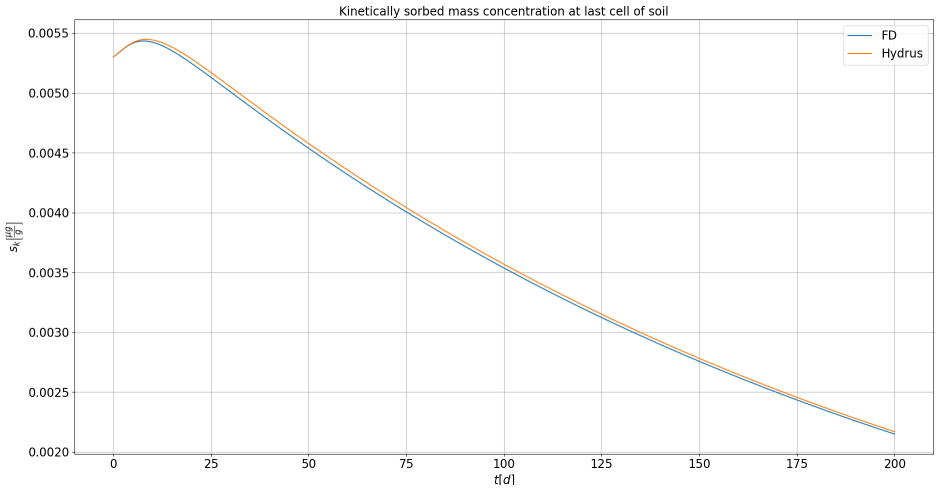

Figure 6.4: Comparison of kinetically sorbed concentration, calculated by FD and Hydrus, sand layers included. The bottom row of the contaminated soil is considered. Parameters were the same as in Fig. 6.1.

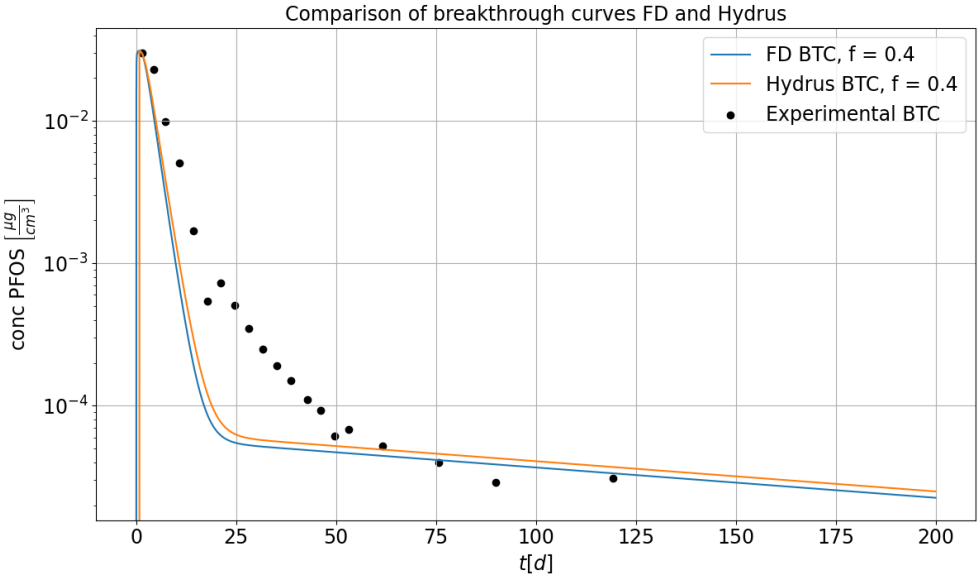

Figure 6.5: Comparison of experimental, Hydrus and FD BTC. Parameters: $n_e = 0.4$, $\rho = 1.58\frac{g}{cm^3}$, $k_d = 4.5\frac{cm^3}{g}$, $\beta = 0.98$, $f = 0.4$, $\alpha_k = 0.005\frac{1}{d}$, $D_e = 2.55\frac{cm^2}{d}$, $\alpha_l = 9cm$, $v = 78.9\frac{cm}{d}$, $c_{init} = 0.032\frac{\mu g}{cm^3}$, $s_{k,init} = 0.005337\frac{\mu g}{g}$, $T_{MAX} = 200d$, $T_{STEPS} = 230000$, $X_{LENGTH} = 55cm$, $X_{STEPS} = 56$, $top = 7$, $bot = 49$, $n_{e,sand} = 0.31$, $\alpha_{l,sand} = 5cm$, $v_{sand} = 101.80645\frac{cm}{d}$, Hydrus settings like Bierbaum et al. ($f = 0.4$).

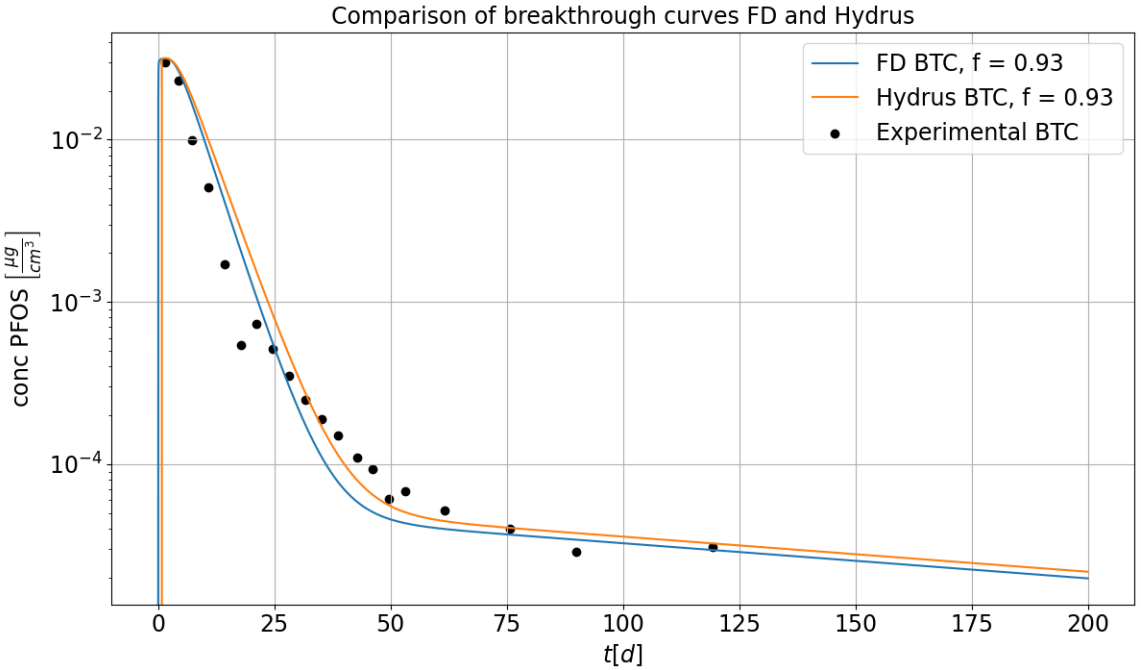

Figure 6.6: Comparison of experimental, Hydrus and FD BTCs with $f = 0.93$. Parameters were the same as in Fig. 6.5.

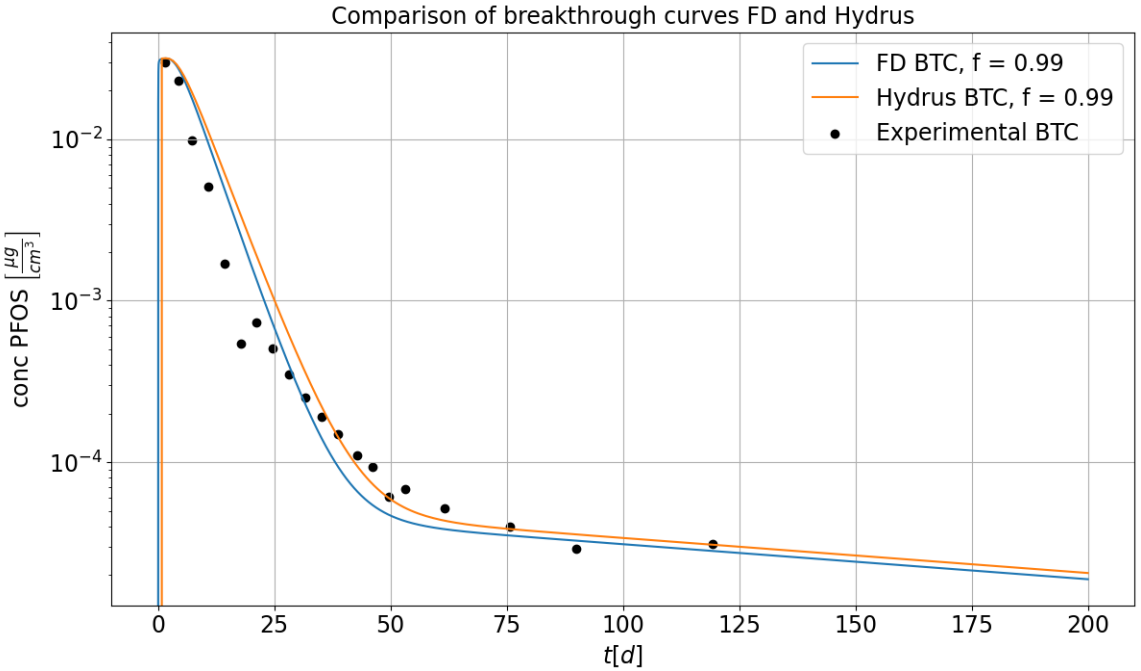

Figure 6.7: Comparison of experimental, Hydrus and FD BTCs with $f = 0.99$. Parameters were the same as in Fig. 6.5.

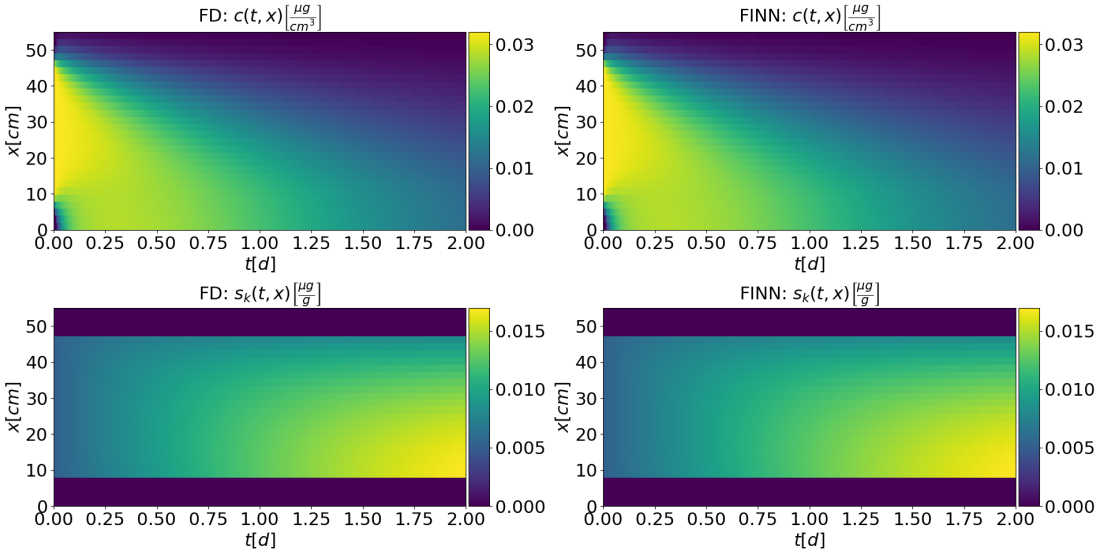

Figure 6.8: Comparison of FD and FINN predicted c and s_k, run b.

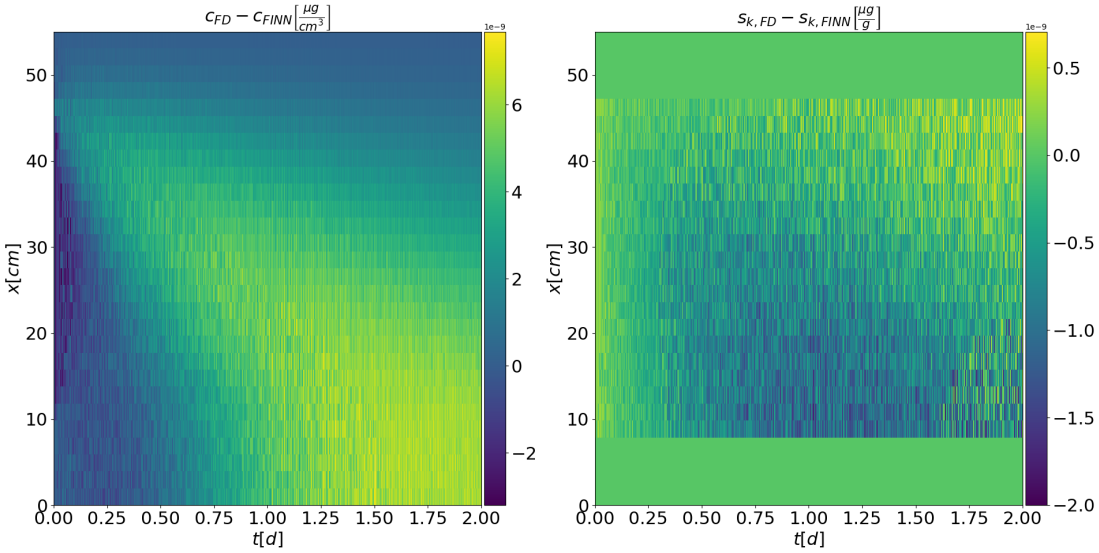

Figure 6.9: Difference of FD and FINN predicted c and s_k, run b.

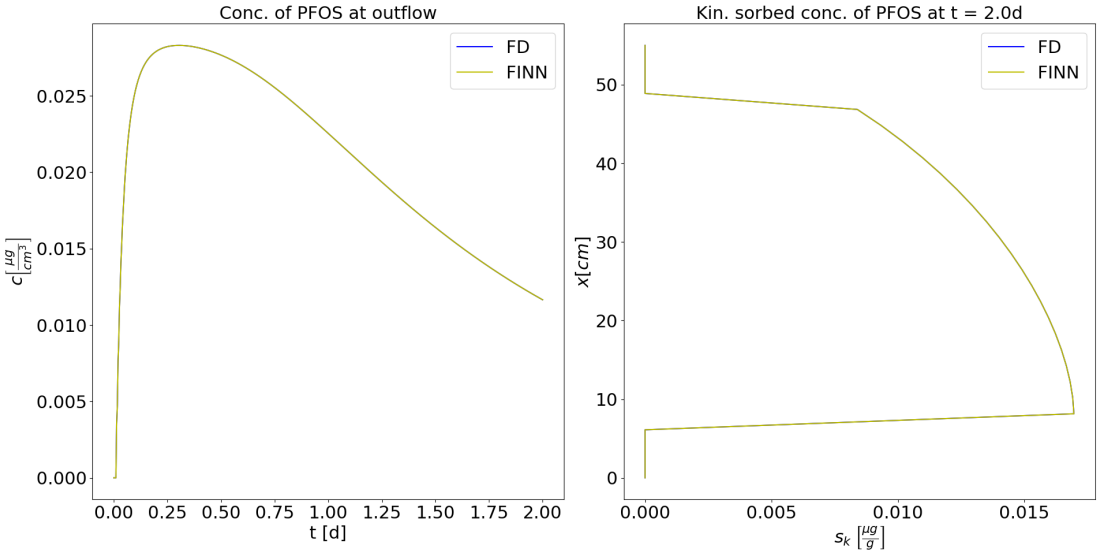

Figure 6.10: BTC of PFOS given by FD and approximated by FINN (left), Right: Kin. sorbed concentration of PFOS at t_{end} (right), run b.

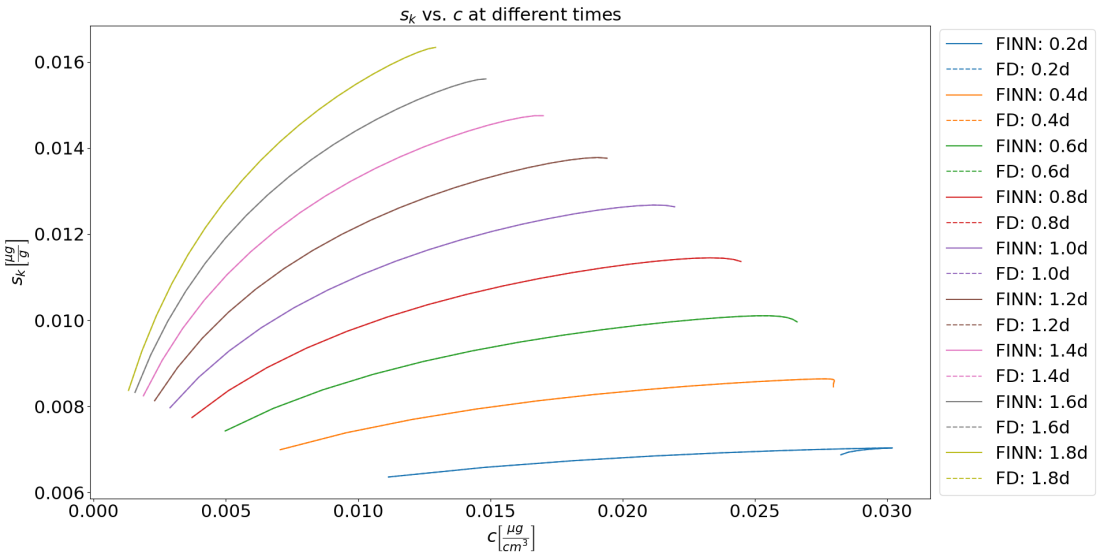

Figure 6.11: The unknowns s_k and c of the FINN solution at fixed time steps, run b.

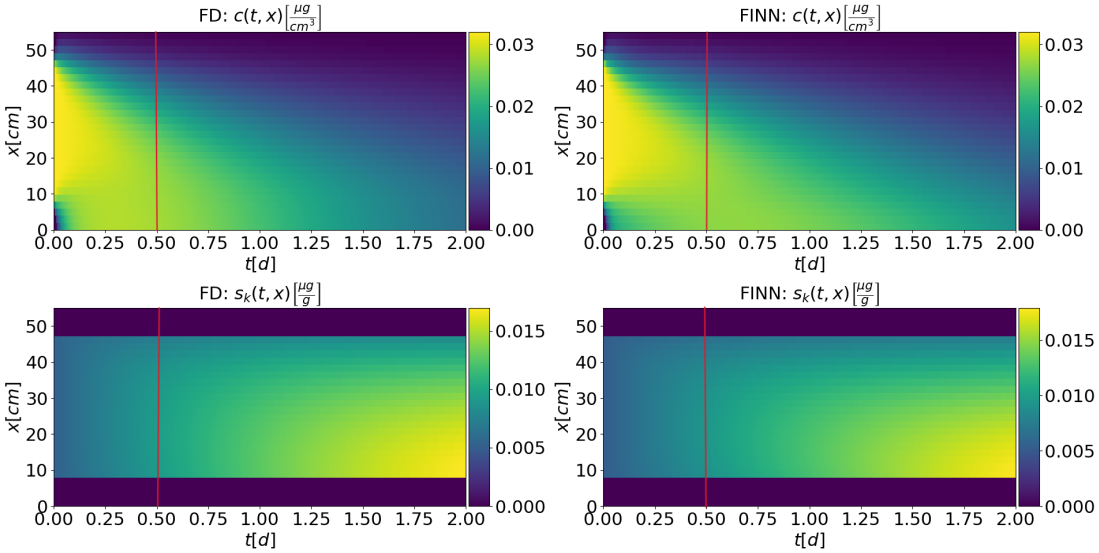

Figure 6.12: Additional comparison of c and s_k calculated by FD and FINN, red line marks end of the training data set, 300 training epochs. Other settings as in run e.

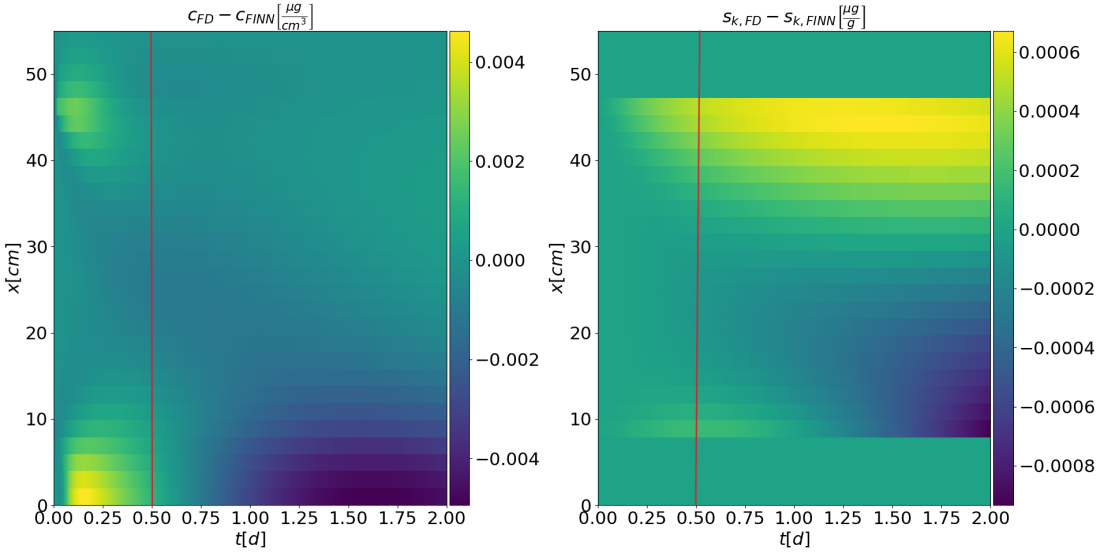

Figure 6.13: Difference FD and FINN predicted c and s_k, red line marks end of the training data set, 300 training epochs. Other settings as in run e.

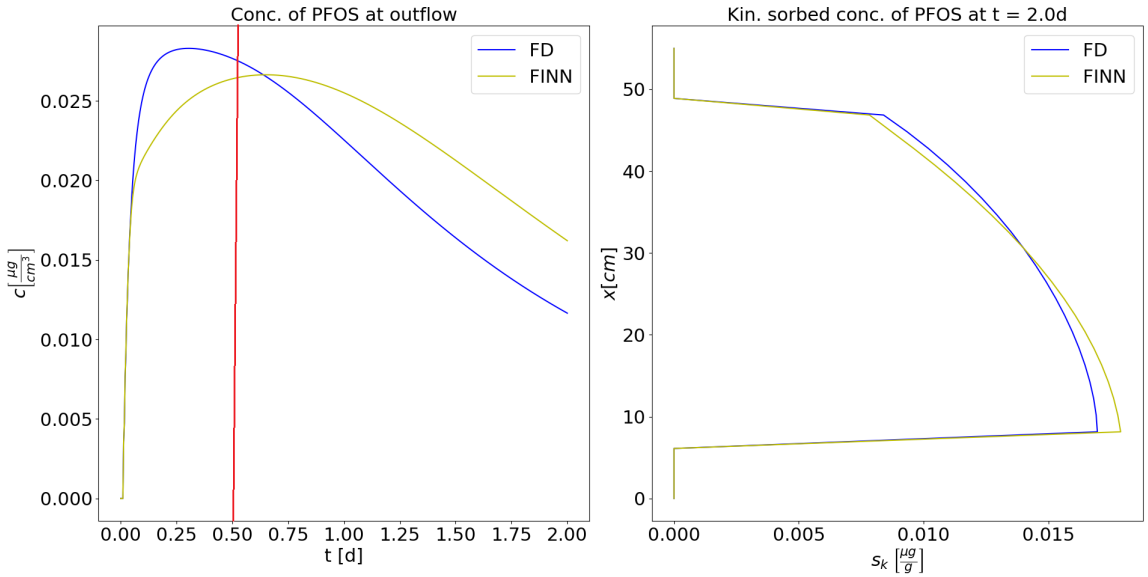

Figure 6.14: BTC of PFOS given by FD and approximated by FINN (left), Kin. sorbed concentration of PFOS at t_{end} (right). Red line marks the end of the training data set, 300 training epochs. Other settings as in run e.

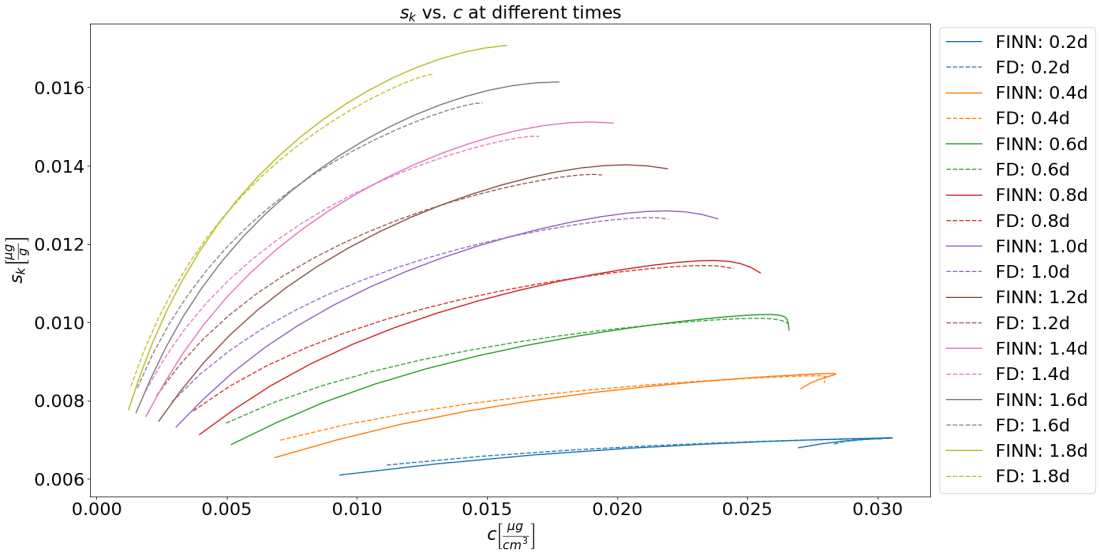

Figure 6.15: The unknowns s_k and c of the solution at fixed time steps, 300 training epochs. Other settings as in run e.

List of Figures

List of Tables

Listings

Bibliography

[1] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *arXiv*, December 2019.

[2] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[3] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[5] S. Kullback and R. A. Leibler. On Information and Sufficiency. *Ann. Math. Stat.*, 22(1):79–86, March 1951.

[6] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Math. Control Signals Systems*, 2(4):303–314, December 1989.

[7] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, January 1991.

[8] The pandas development team. pandas-dev/pandas: Pandas, February 2020.

[9] Sklearn documentation. `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html`. Accessed: 2022-10-02.

[10] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, October 1986.

[11] Benchmark for wine decision neural network. `https://github.com/matzegltg/BA/tree/main/WineDecision`. Accessed: 2022-12-12.

[12] Artificial neural network in publications - dimensions ai. `https://app.dimensions.ai/discover/publication`. Accessed: 2022-12-07.

[13] Team. Cats vs Dogs Classification (with 98.7% Accuracy) using CNN Keras - Deep Learning Project for Beginners - DataFlair. *DataFlair*, August 2021.

[14] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations. *arXiv*, November 2017.

[15] Matthias Karlbauer, Timothy Praditia, Sebastian Otte, Sergey Oladyshkin, Wolfgang Nowak, and Martin V Butz. Composing partial differential equations with physics-aware neural networks. In *Proceedings of the 39th International Conference on Machine Learning*, Proceedings of Machine Learning Research, Baltimore, USA, 16–23 Jul 2022.

[16] Timothy Praditia, Matthias Karlbauer, Sebastian Otte, Sergey Oladyshkin, Martin V. Butz, and Wolfgang Nowak. Finite Volume Neural Network: Modeling Subsurface Contaminant Transport. *arXiv*, April 2021.

[17] V. Gellrich, T. Stahl, and T. P. Knepper. Behavior of perfluorinated compounds in soils during leaching experiments. *Chemosphere*, 87(9):1052–1056, May 2012.

[18] Bo Guo, Jicai Zeng, and Mark L. Brusseau. A Mathematical Model for the Release, Transport, and Retention of Per- and Polyfluoroalkyl Substances (PFAS) in the Vadose Zone. *Water Resour. Res.*, 56(2):e2019WR026667, February 2020.

[19] T. Bierbaum, N. Klaas, C. Haslauer, J. Braun, F. T. Lange, and F. Sacher. Experimentelle Methoden zur Untersuchung der PFAS-Immobilisierung. *Mitt Umweltchem Ökotox*, March 2022.

[20] Shui Cheung Edgar Leung, Pradeep Shukla, Dechao Chen, Ehsan Eftekhari, Hongjie An, Firuz Zare, Negareh Ghasemi, Dongke Zhang, Nam-Trung Nguyen, and Qin Li. Emerging technologies for PFOS/PFOA degradation and removal: A review. *Sci. Total Environ.*, 827:153669, June 2022.

[21] P. Nkedi-Kizza, J. W. Biggar, H. M. Selim, M. Th. Van Genuchten, P. J. Wierenga, J. M. Davidson, and D. R. Nielsen. On the Equivalence of Two Conceptual Models for Describing Ion Exchange During Transport Through an Aggregated Oxisol. *Water Resour. Res.*, 20(8):1123–1130, August 1984.

[22] M. Th. Van Genuchten. Non-equilibrium transport parameters from miscible displacement experiments. *AGRIS: International Information System for the Agricultural Science and Technology*, 1981.

[23] Nobuo Toride, Feike J. Leij, and Martinus T. van Genuchten. A comprehensive set of analytical solutions for nonequilibrium solute transport with first-order decay and zero-order production. *Water Resour. Res.*, 29(7), July 1993.

[24] L. A. Richards. CAPILLARY CONDUCTION OF LIQUIDS THROUGH POROUS MEDIUMS. *Physics*, 1(5):318–333, November 1931.

[25] Jirka Šimůnek and Martinus Th. van Genuchten. Modeling Nonequilibrium Flow and Transport Processes Using HYDRUS. *Vadose Zone J.*, 7(2):782–797, May 2008.

[26] J. C. van Dam, J. Huygen, J. G. Wesseling, R. A. Feddes, P. Kabat, P. E. V. van Walsum, P. Groenendijk, and C. A. van Diepen. Theory of SWAP version 2.0;

Simulation of water flow, solute transport and plant growth in the Soil-Water-Atmosphere-Plant environment. *AGRIS: International Information System for the Agricultural Science and Technology*, 1997.

[27] M. Selim H, M. Davidson J, and S. Mansell R. Evaluation of a two-site adsorption-desorption model for describing solute transport in soils. *Proc Summer Comput Simulation Conf*, (1976):444–448, 1976.

[28] M. Th. van Genuchten and R. J. Wagenet. Two-Site/Two-Region Models for Pesticide Transport and Degradation: Theoretical Development and Analytical Solutions. *Soil Sci. Soc. Am. J.*, 53(5):1303–1310, September 1989.

[29] A. E. Scheidegger. General theory of dispersion in porous media. *J. Geophys. Res.*, 66(10):3273–3278, October 1961.

[30] Herbert Freundlich. Über die Adsorption in Lösungen. *Z. Phys. Chem.*, 57U(1):385–470, October 1907.

[31] Finn including two-site sorption. `https://github.com/matzegltg/finn-1`. Accessed: 2022-12-12.

[32] Brunetti G. Collenteur, R.A. and M. Vremec. Phydrus: Python implementation of the hydrus-1d unsaturated zone model. 2019.

[33] R. Courant, K. Friedrichs, and H. Lewy. Über die partiellen Differenzengleichungen der mathematischen Physik. *Math. Ann.*, 100(1):32–74, December 1928.

[34] Sedat Biringen. A note on the numerical stability of the convection-diffusion equation. *J. Comput. Appl. Math.*, 7(1):17–20, March 1981.

[35] Wolfgang Nowak and Anneli Guthke. Entropy-Based Experimental Design for Optimal Model Discrimination in the Geosciences. *Entropy*, 18(11):409, November 2016.

[36] J. Simunek, H. Saito, M. Sakai, and Th. M. Genuchten. The HYDRUS-1D Software Package for Simulating the One-Dimensional Movement of Water, Heat, and Multiple Solutes in Variably-Saturated Media. *ResearchGate*, January 2008.

[37] Ricky T. Q. Chen. torchdiffeq, 2018.

[38] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv*, December 2014.

[39] Workspaces for training with experimental data on high-performance cluster. `https://github.com/matzegltg/BA/tree/main/workspaces_emma`. Accessed: 2022-12-12.

[40] Used hydrus files for FD solver validation. `https://github.com/matzegltg/BA/tree/main/Hydrus_BA`. Accessed: 2022-12-12.

[41] D. Stathakis. How many hidden layers and nodes? *Int. J. Remote Sens.*, 30(8):2133–2147, April 2009.

[42] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the State of Neural Network Pruning? *arXiv*, March 2020.

[43] Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural Machine Translation in Linear Time. *arXiv*, October 2016.

[44] Matthias Karlbauer, Sebastian Otte, Hendrik P. A. Lensch, Thomas Scholten, Volker Wulfmeyer, and Martin V. Butz. A Distributed Neural Network Architecture for Robust Non-Linear Spatio-Temporal Prediction. *arXiv*, December 2019.

[45] Laurent Valentin Jospin, Hamid Laga, Farid Boussaid, Wray Buntine, and Mohammed Bennamoun. Hands-On Bayesian Neural Networks—A Tutorial for Deep Learning Users. *IEEE Comput. Intell. Mag.*, 17(2):29–48, April 2022.

[46] Da Li, Xinbo Chen, Michela Becchi, and Ziliang Zong. Evaluating the Energy Efficiency of Deep Convolutional Neural Networks on CPUs and GPUs. In *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom)*, pages 477–484. IEEE, October 2016.