PROCEEDINGS, 45th Workshop on Geothermal Reservoir Engineering
Stanford University, Stanford, California, February 10-12, 2020
SGP-TR-216

Prediction Modeling for Geothermal Reservoirs Using Deep Learning

Halldora Gudmundsdottir and Roland N. Horne

Department of Energy Resources Engineering, Stanford University, Stanford, California, USA

halldora@stanford.edu

Keywords: deep learning, machine learning, neural networks, injection, tracer

ABSTRACT

Reservoir characterization and prediction modeling are among the more challenging tasks in geothermal reservoir engineering. Because thermal breakthrough in producers can occur when injecting cold wastewater, we must understand how production is influenced by injection to sustainably manage our geothermal fields. The fractured nature of geothermal reservoirs causes the relationship between production and injection wells to be highly complex and nonlinear. In this work, we investigate models to serve as a substitute for full reservoir. We use deep learning algorithms for this purpose, and we explore two different architectures, standard feedforward neural networks and recurrent neural networks. The deep learning model maps the input to an output through series of layers, nodes and activation functions. Here, the mapping is accomplished with injection flow rates at the injectors as input and tracer concentration data at the producers as output. The models were trained on a synthetic geothermal reservoir and our analysis concluded that for this case the simple feedforward neural network outperformed the more complex recurrent neural network.

1. INTRODUCTION

Predicting thermal breakthrough is an important task in managing a sustainable geothermal reservoir. This prediction task commonly involves characterizing the reservoir and building a geological model. Characterizing fractures and flow paths in geothermal reservoirs, while challenging, gives valuable information for successful development of geothermal reservoirs. Because fractures control the mass and heat transport within the system, understanding the layout of fractures and the connectivity between them helps establish optimal injection strategies for reservoirs. Conventionally, prediction modeling involves first building a conceptual model of the reservoir based on exploration data and characterization efforts, and then converting that model to a reservoir simulation model which is subsequently calibrated using inverse analysis and production data. This conventional way of prediction modeling is often a time consuming and computationally expensive process.

Recently, there have been numerous studies published on machine learning and deep learning specifically for modeling subsurface behavior. Liu and Horne (2013) proposed a kernel-based convolution model to predict a pressure response in a single-well pressure transient analysis. In their study, input features were carefully crafted using transformations of flow rates and time. Tian and Horne (2015) extended this idea to include pressure analysis for multi-well systems. Instead of a kernel-based convolution, they used kernel-ridge regression and managed to reduce computational cost dramatically and increase prediction accuracy. Furthermore, Tian and Horne (2017) showed that deep learning techniques, such as recurrent neural networks, could identify the reservoir model and map the relationship between pressure and flow rates without manipulating the input features. More recently, Li et al. (2019) further illustrated the capabilities of recurrent neural networks by predicting downhole pressure using temperature and flow rate data as inputs. In their study, temperature was found to be the best candidate for pressure prediction, compared to flow rates and choke vale opening. A similar study by Alakeely and Horne (2020) showed that convolutional neural networks could be used to model temporal subsurface behavior and concluded that they were just as powerful as recurrent neural networks.

The goal of this study is to investigate the capabilities of neural networks to model the relationship between injectors and producers in a geothermal reservoir without relying on a physical model. Having such a model is beneficial when it comes to making fast predictions for managing the reservoir, and would allow us, if successful, to analyze and run numerous scenarios of well controls to optimize for the most sustainable exploitation of the reservoir. In this study we compare two deep learning methods, a feedforward neural network and a recurrent neural network. A synthetic geothermal reservoir is numerically simulated and used for the comparison and data from four injectors and five producers used as the input and output for the neural networks. We focus on two cases specifically, building separate models for each producer and building one model predicting all producers at the same time.

2. METHODOLOGY

2.1 Feedforward Neural Networks

In its simplest form a standard neural network, often referred to as a Multilayer Perceptron (MLP), consists of an input layer, a hidden layer and an output layer. For a neural network to be considered a deep learning model, it must contain more than one hidden layer (Goodfellow, Bengio, & Courville, 2016). An example of a feedforward neural network with one hidden layer is shown in Figure 1a. Notice that we denote this neural network as a 2-layer neural network. That is, we do not count the input layer. The layers in the network consist of neurons that perform the computations for the output. Most neural networks are fully connected, meaning that each neuron is connected to every other neuron in the layers on either side, but neurons within a single layer share no connections. The strength of the connections between neurons are represented by weights that can either be positive (amplifying effects) or negative (dampening effects). The higher the weight, the more influence a neuron has on the other neurons it is connected to.

The mapping from $\boldsymbol{x} = [x_1, x_2, x_3]$ to $\hat{\boldsymbol{y}} = [\hat{y}_1, \hat{y}_2]$ in Figure 1a can be written as

$$\boldsymbol{z}^{[1]} = \boldsymbol{W}^{[1]}\boldsymbol{x} + \boldsymbol{b}^{[1]}, \tag{1}$$

$$\boldsymbol{a}^{[1]} = g^{[1]}\left(\boldsymbol{z}^{[1]}\right), \tag{2}$$

$$\boldsymbol{z}^{[2]} = \boldsymbol{W}^{[2]}\boldsymbol{a}^{[1]} + \boldsymbol{b}^{[2]}, \tag{3}$$

$$\boldsymbol{a}^{[2]} = g^{[2]}\left(\boldsymbol{z}^{[2]}\right) = \hat{\boldsymbol{y}}, \tag{4}$$

where $\boldsymbol{W}^{[l]}$ is the weight matrix of the connection between neurons, $\boldsymbol{b}^{[l]}$ the bias term and $g^{[l]}$ the activation function, for each layer $l = 1, \dots, L$. When we train our neural network, we aim to optimize the model parameters $\boldsymbol{W}^{[l]}$ and $\boldsymbol{b}^{[l]}$ such that the predictions closely match the actual outputs for all training samples $i = 1, \dots, m$. To quantify how we are achieving this goal, we define a cost function discrepancy between the predicted value and the desired output. The objective of a neural network is to minimize the cost function which is usually accomplished through an iterative optimization algorithm such as gradient descent.

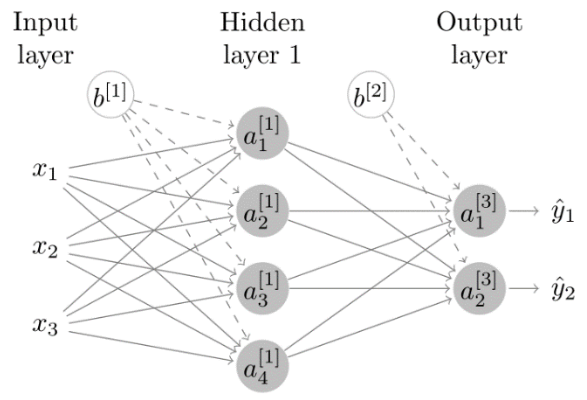


Figure 1a: A 2-layer neural network with one hidden layer. Nodes represent neurons and arrows the connection between neurons.

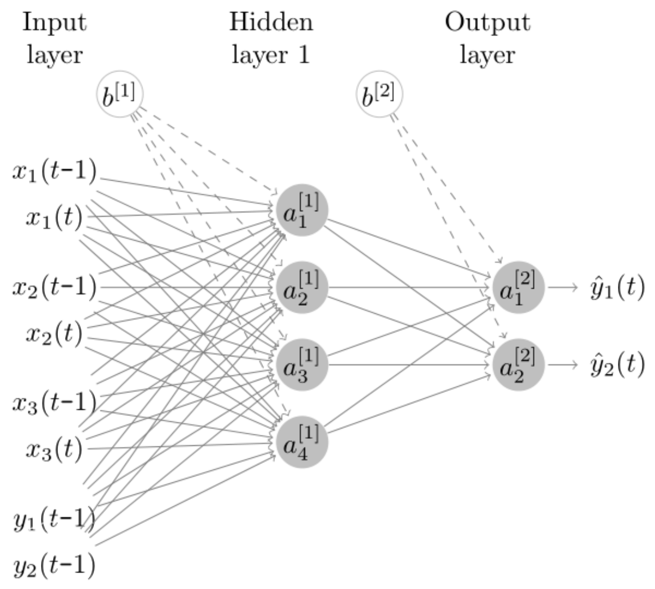


Figure 2b: A 2-layer neural network with one hidden layer. The network models time series data, with one time lag and target feedback added to the input.

In our case we want to map time series of injection flow rates to time series of production data. Also, to attempt to honor temporal dependencies, inherent in time series, we explore the option of adding time lags to our input data as well as allowing target feedback. A target feedback is when some data points of previous time of the output is added as part of the input to the model. For example, adding one time lag as well as target feedback to the input of an MLP model, the input $\boldsymbol{x}$ can be expressed in the form $\boldsymbol{x} = [x_1(t-1), x_1(t), x_2(t-1), x_2(t),\ x_3(t-1), x_3(t), y_1(t-1), y_2(t-1)]$ and the output $\hat{\boldsymbol{y}} = [\hat{y}_1(t-1), \hat{y}_2(t-1)]$. This is better explained visually in Figure 1b.

2.2 Recurrent Neural Networks

For a simple feedforward neural network, the general goal is to map $x^{(i)}$ to $y^{(i)}$ for each training sample i, $i = 1,\ldots,m$. By doing so, we assume that the output at time t only depends on the input at the same time t. More specifically, this model structure ignores the fact that for time series the order of observations is important, and that a value at time t might depend on previous values such as $t-1$, $t-2$, or $t-n$. In the previous section, an attempt to enhance the capabilities of the feedforward neural network was made by adding time lags to the input. However, another method, a recurrent neural network (RNN), allows previous computations to be fed back into the model without explicitly creating time lags.

A standard RNN with three adjacent training samples is shown in Figure 3. The difference between this model structure and the one in Figure 1a is that the hidden layers of the adjacent training samples are also connected with weights, allowing the memory of previous computations to be fed into the hidden layer together with the current input (Tian, 2018). The mathematical form of the RNN, and the equivalence of Eq. (1) of the feedforward neural network, is written as

$$\boldsymbol{z}^{(i)[1]} = \boldsymbol{W}^{[1]}\boldsymbol{x}^{(i)} + \boldsymbol{H}^{[1]}\boldsymbol{a}^{(\boldsymbol{i}-\mathbf{1})[\mathbf{1}]} + \boldsymbol{b}^{[1]}, \tag{5}$$

where $\boldsymbol{H}^{[\mathbf{1}]}$ is the weight matrix of the connection between the activation values from the previous training sample $\boldsymbol{a}^{(\boldsymbol{i}-\mathbf{1})[\mathbf{1}]}$. The additional term $\boldsymbol{H}^{[1]}\boldsymbol{a}^{(\boldsymbol{i}-\mathbf{1})[\mathbf{1}]}$ represents the memory from the hidden layer of previous training samples or time steps. A more detailed description of the workings of RNNs can be found in Tian (2018).

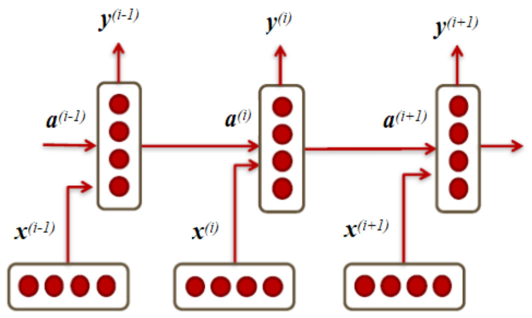


Figure 3: The structure of a recurrent neural network with one hidden layer. Adapted from Tian (2018).

3. RESULTS

The main goal of this study was to investigate the capabilities of two deep learning methods to model the relationship between wells in a geothermal reservoir. The two methods we considered were a standard feedforward neural network (MLP) and a recurrent neural network (RNN). The capabilities of the methods were analyzed and compared using two case studies. Firstly, we built a separate model for each producer in the reservoir, and secondly, we built one large model for the reservoir, including all the production wells. To perform the analysis for both cases, we used data that we obtain by simulating numerically a synthetic geothermal reservoir.

3.1 The Experimental Setup

The synthetic geothermal reservoir designed for this study is shown in Figure 4. The reservoir covers a 10x10 km area and consists of numerous discrete fractures. The fractures dominate the fluid flow in the reservoir, although, for computational expedition, some flow is allowed in the rock surrounding the fractures. The reservoir contains nine wells, four injection wells, and five production wells. Figure 4 shows a top-view of the reservoir and the location of the wells. The red lines indicate fractures that are essentially horizontal planes extending from the top to the bottom of the reservoir. For the simulation of the reservoir, we used varying injection schedules for the injectors shown in Figure 6 and 5, with a fixed injection rate and injected tracer concentration. The reservoir simulator ADGPRS, developed at Stanford University, was used to simulate the synthetic reservoir under single-phase geothermal conditions. The produced tracer concentration at each producer is illustrated in Figure 7.

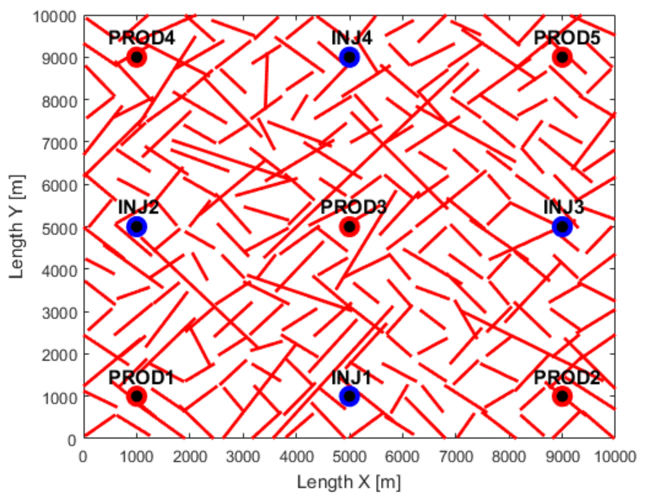


Figure 4: The synthetic geothermal reservoir used to generate data for the three case studies. Injectors are colored blue and producers are colored red.

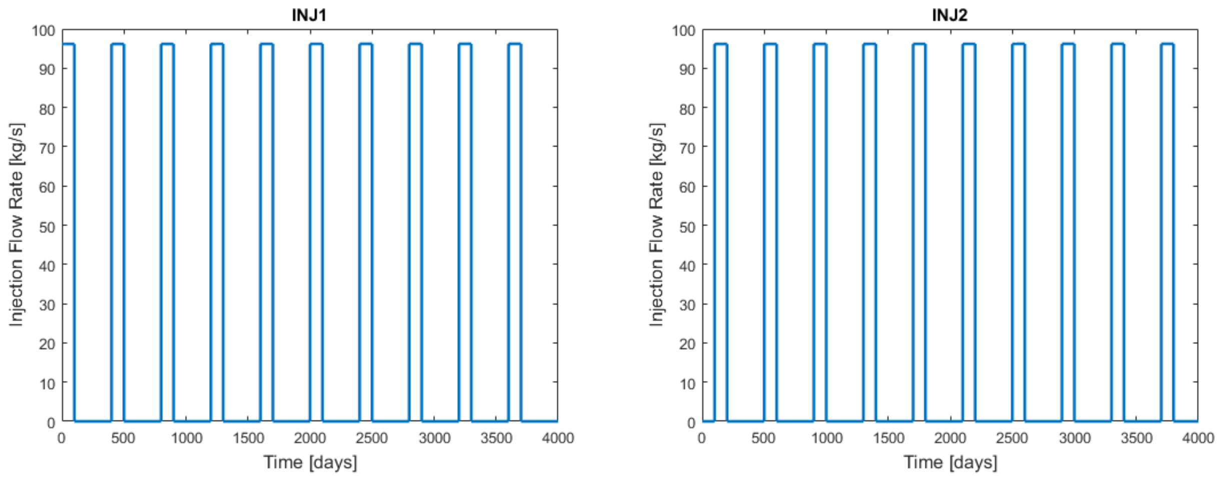


Figure 5: The injection schedule for injectors 1 and 2.

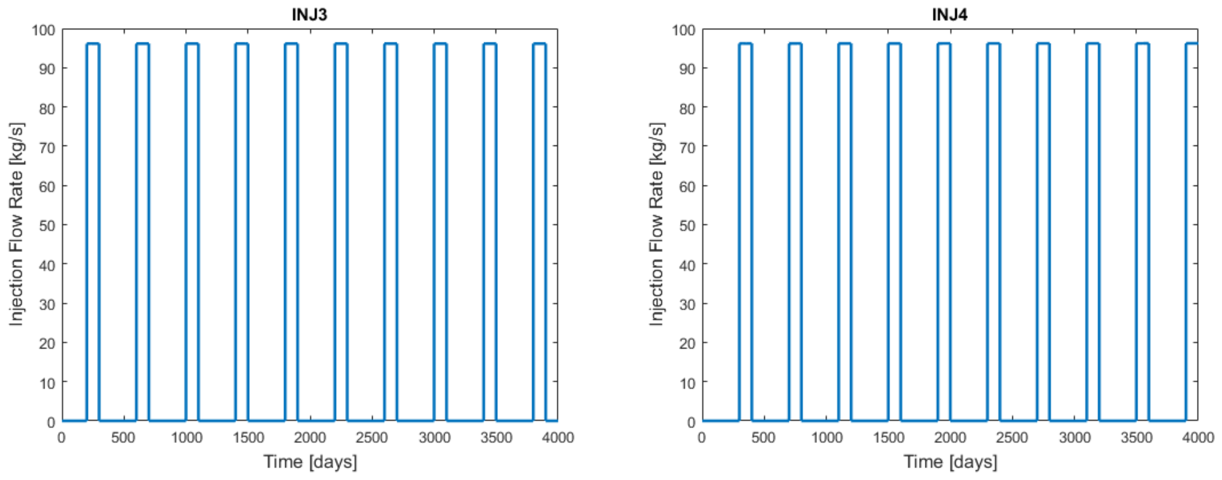


Figure 6: The injection schedule for injectors 3 and 4.

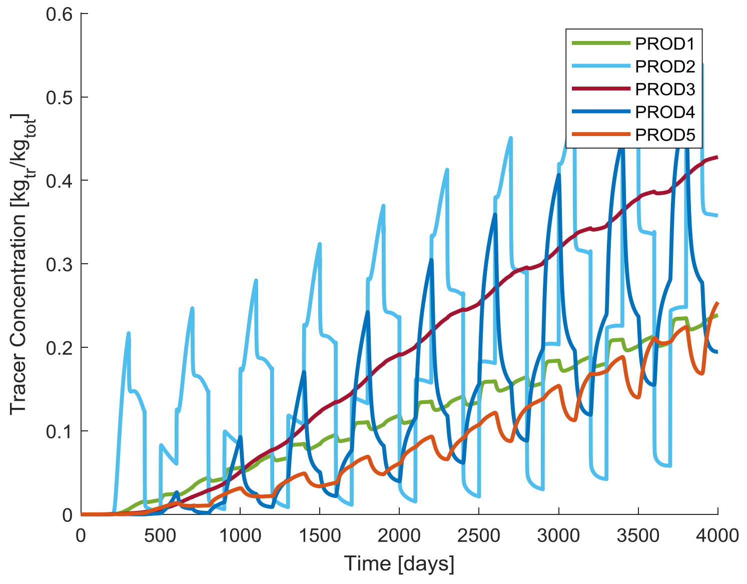


Figure 7: The produced tracer concentration from producer 1, 2, 3, 4 and 5 over the simulation time.

3.2 Case 1: Creating a Separate Neural Network Model for Each Producer

In our first case we trained a unique model for each of the five producers, given the same input from all injectors. Two types of neural networks were prepared, a feedforward neural network and a recurrent neural network. Also, for both methods, adding a target feed, where parts of the previous output are added to the input, was explored.

Without adding target feed, the input X takes the form $[Q(t)_{I1}, Q(t)_{I2}, Q(t)_{I3}, Q(t)_{I4}]$, and the output Y the form $[C(t)_{Pj}]$, where $Q(t)_{Ii}$ is the injection rate for injector i and $C(t)_{Pj}$ is the produced tracer concentration for producer j. By adding a target feed to the models, the form of the input X changes to $[Q(t)_{I1}, Q(t)_{I2}, Q(t)_{I3}, Q(t)_{I4}, C(t-1)_{Pj}]$. When building different models for each producer, the input X stays the same but the output Y changes depending on which producer is being modeled. The simulated injection and production profiles span over 10 years, with 4000 data points with each data point representing one day. For training we used 70% of the available data points, and for validation and testing we used 30% of the available data points.

To tune the hyperparameters for the model, a grid search across different numbers of neurons and layers as well as the number of time lags to include in the input was conducted. Table 1 shows the best hyperparameter configurations for the models after the tuning process. Because neural networks are stochastic, they give different results when evaluated on the same data. The results depend on the initialization of the weights in the model and thus, ideally, each model configuration should be evaluated using the average of multiple repeats of training process. In our study we initialize and train the model 20 times for each model configuration and report the average and the standard deviation of the test error. For good configurations of the model this average test error should be relatively high with a small standard deviation. A small standard deviation indicates that the configuration of the hyperparameters yield a stable model. Table 2 shows the average test error and standard deviation for the best model configurations. Notice, that we present the test error as a goodness-of-fit metric, or as the coefficient of determination R^2, where a number close to one indicates good model predictions, and a number close to zero poor model predictions.

Table 1: The best hyperparameter configurations for the models in Case 1.

	MLP	**RNN**	**MLP w/target feed**	**RNN w/target feed**
# neurons	50	20	50	20
# layers	5	3	2	3
# time lags	2	2	2	2

Table 2: The average test error (R^2) and standard deviation for the best model configurations in Case 1.

	MLP	RNN	MLP w/target feed	RNN w/target feed
Producer 1	0.92 ± 0.03	0.92 ± 0.05	0.99 ± 0.01	0.97 ± 0.05
Producer 2	0.80 ± 0.21	0.70 ± 0.28	0.990 ± 0.001	0.989 ± 0.003
Producer 3	0.89 ± 0.04	0.88 ± 0.05	0.92 ± 0.04	0.92 ± 0.09
Producer 4	0.94 ± 0.08	0.90 ± 0.12	0.999 ± 0.006	0.997 ± 0.003
Producer 5	0.87 ± 0.07	0.83 ± 0.08	0.97 ± 0.05	0.97 ± 0.03

Looking at Table 2 we see that the feedforward neural network (MLP) outperforms the recurrent neural network (RNN). However, when adding the target feed to the input, both methods perform exceptionally well, regardless of the model type. These results suggest that a simple model structure, such as the MLP, can model the relationship between the wells, and a more complex structure, such as the RNN, can potentially overfit the data and not generalize as well.

Figure 8-Figure 17 show the predicted tracer concentration using the models for each producer. Because we report the test error as an average of repeated initializations of the model, we show both the absolute best model from the pool of initializations as well as a model that has a test error close to the average.

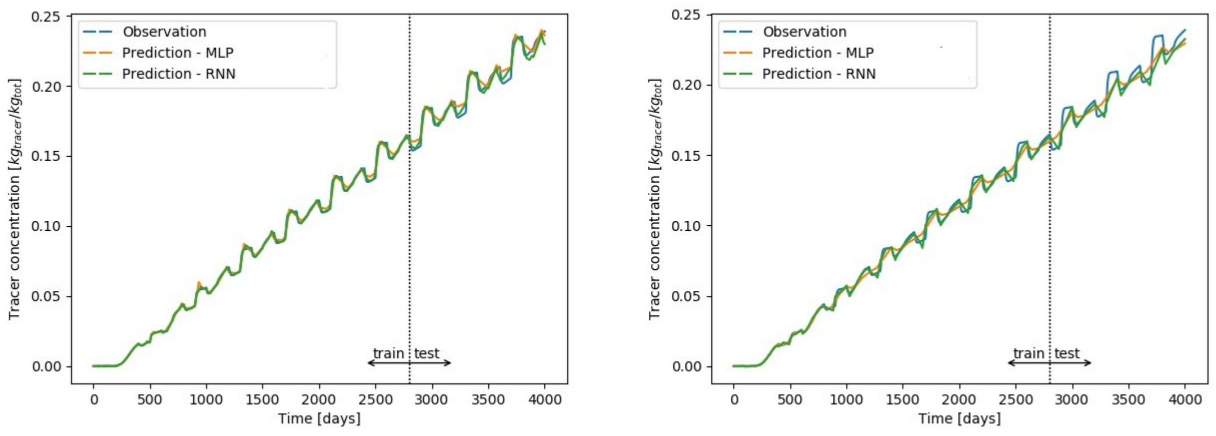


Figure 8: Predicted tracer concentration without target feed for Producer 1 with the best initialization of weights (left) and the initialization of weights closest to the average (right).

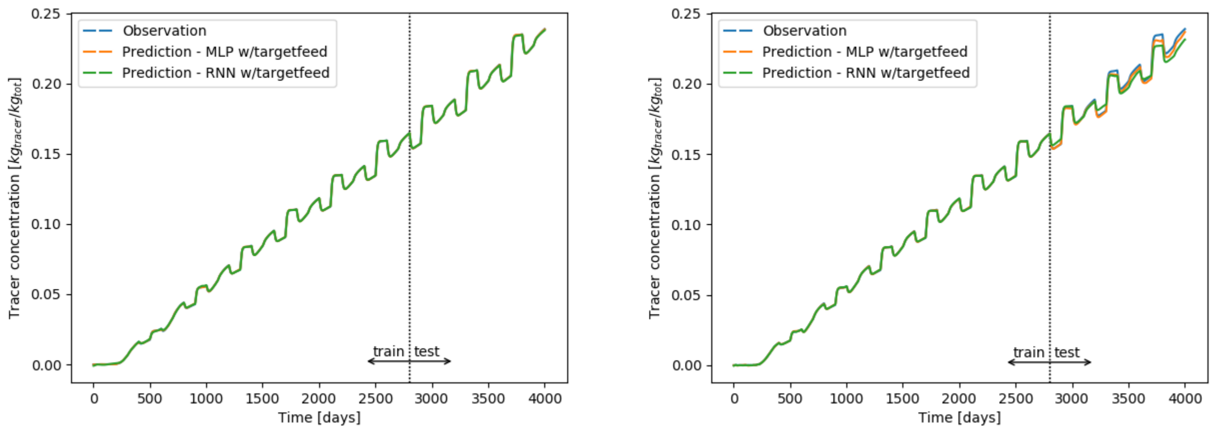


Figure 9: Predicted tracer concentration with target feed for Producer 1 with the best initialization of weights (left) and the initialization of weights closest to the average (right).

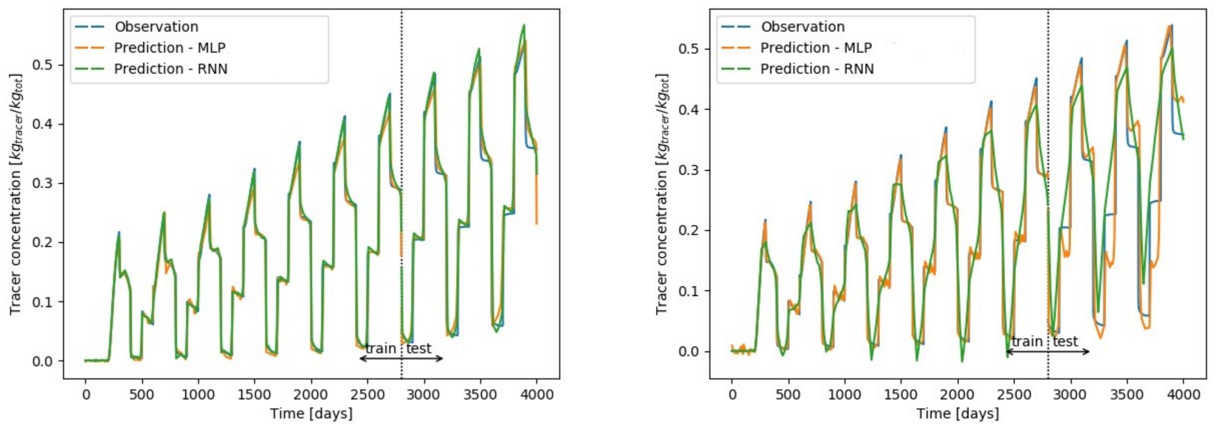


Figure 10: Predicted tracer concentration without target feed for Producer 2 with the best initialization of weights (left) and the initialization of weights closest to the average (right).

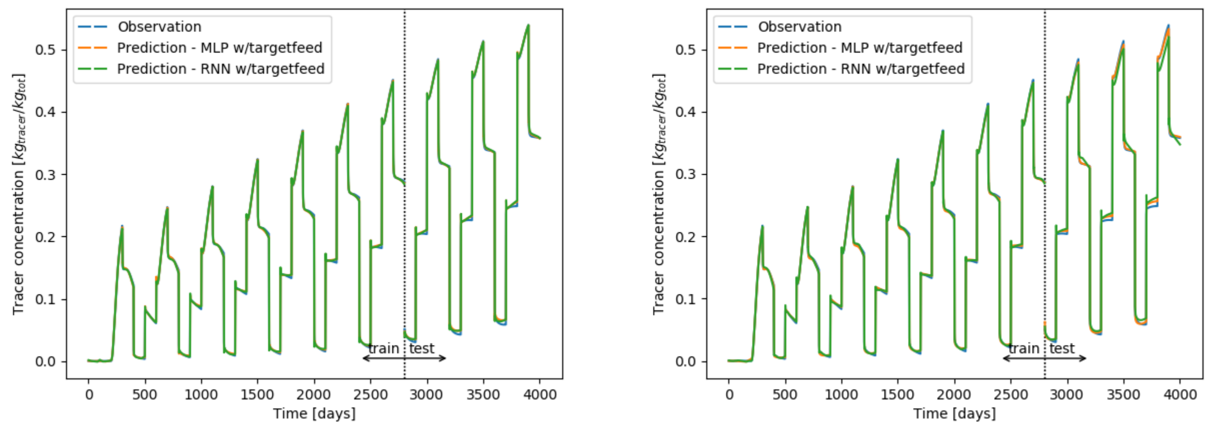


Figure 11: Predicted tracer concentration with target feed for Producer 2 with the best initialization of weights (left) and the initialization of weights closest to the average (right).

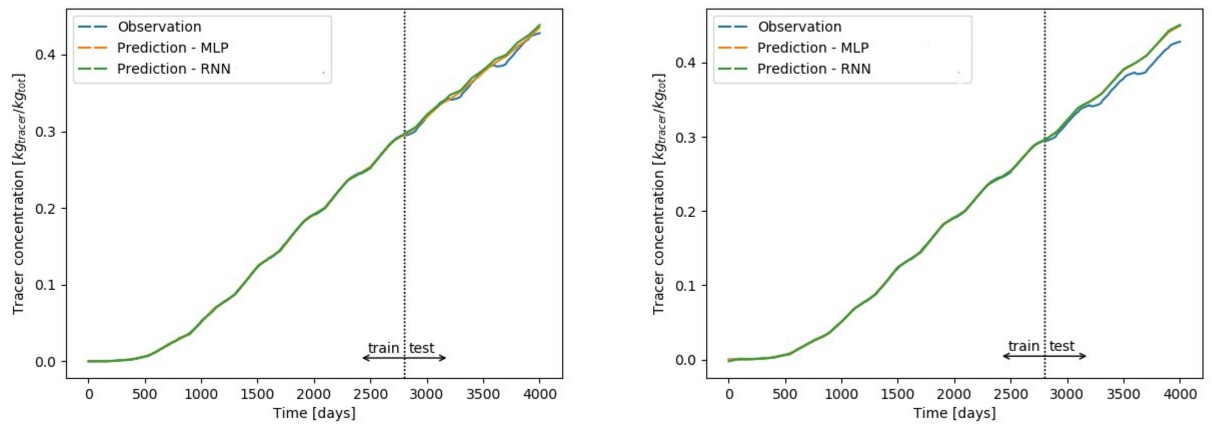


Figure 12: Predicted tracer concentration without target feed for Producer 3 with the best initialization of weights (left) and the initialization of weights closest to the average (right).

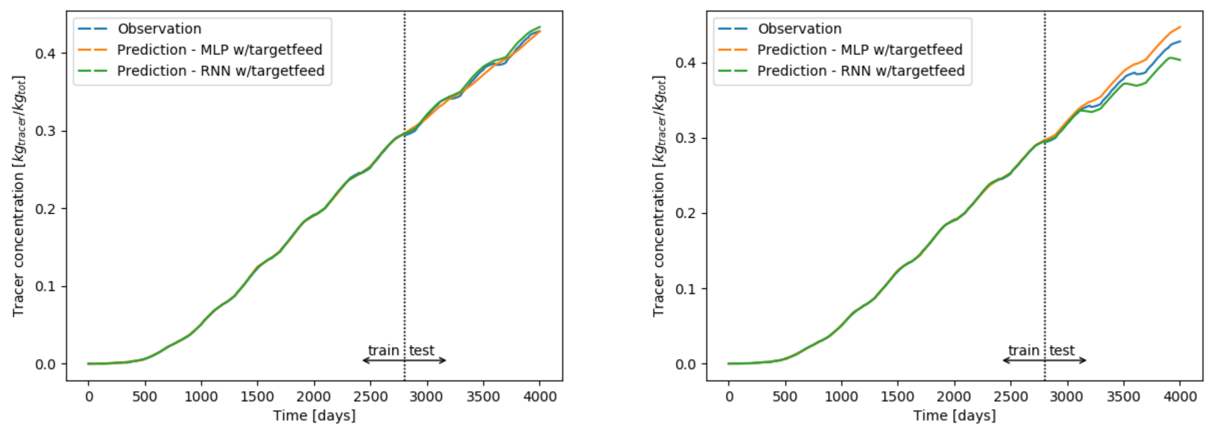


Figure 13: Predicted tracer concentration with target feed for Producer 3 with the best initialization of weights (left) and the initialization of weights closest to the average (right).

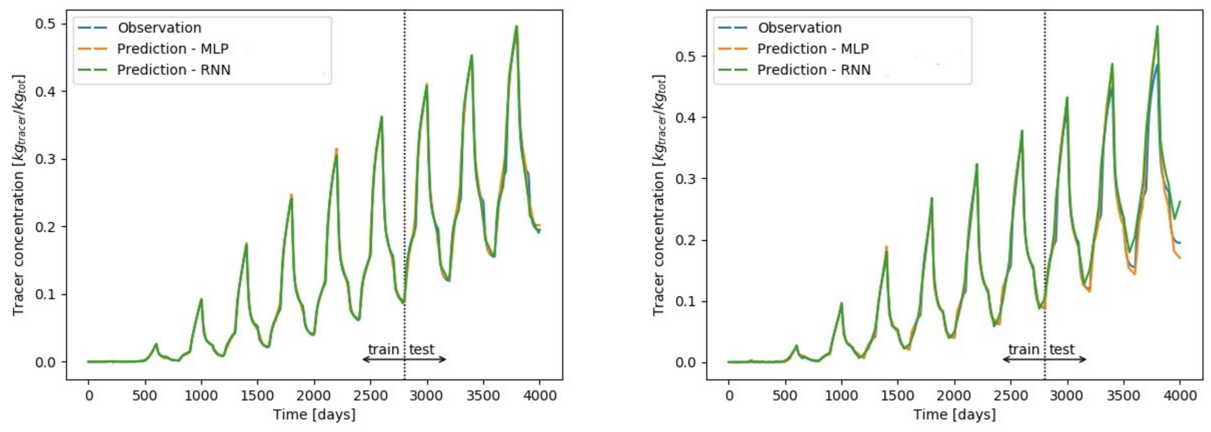


Figure 14: Predicted tracer concentration without target feed for Producer 4 with the best initialization of weights (left) and the initialization of weights closest to the average (right).

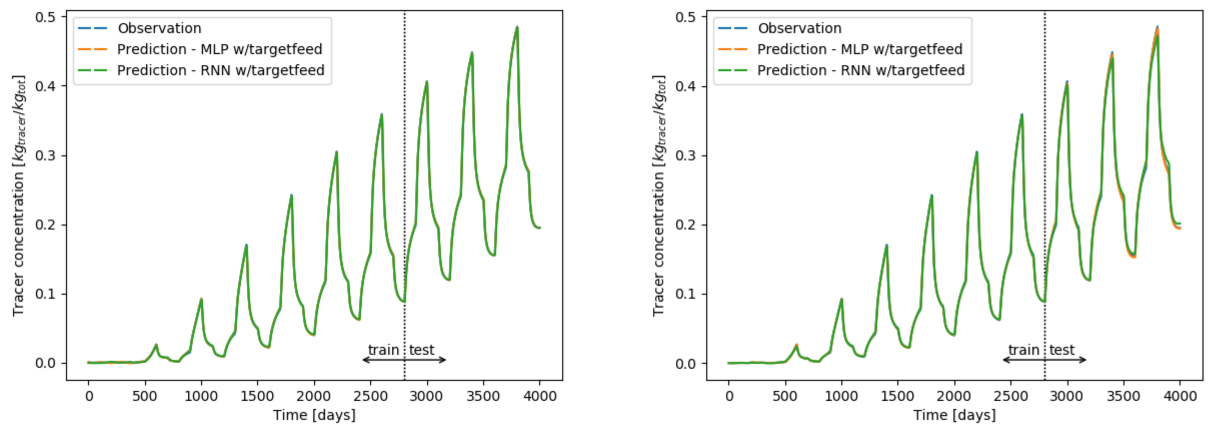
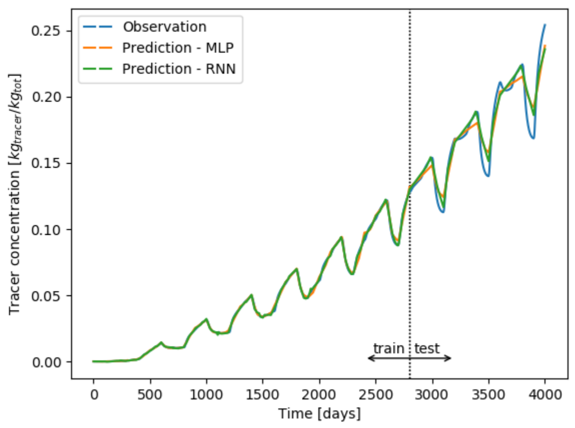


Figure 15: Predicted tracer concentration with target feed for Producer 4 with the best initialization of weights (left) and the initialization of weights closest to the average (right).



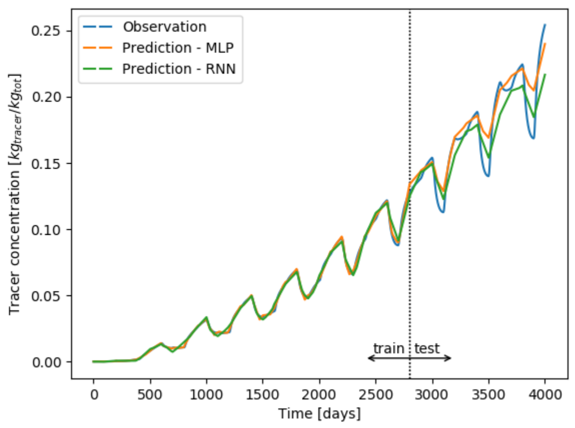
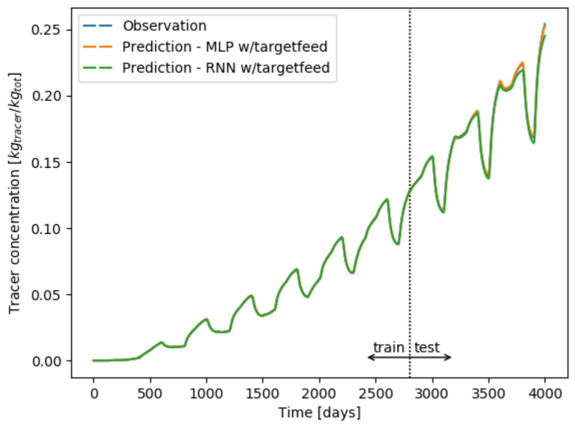


Figure 16: Predicted tracer concentration without target feed for Producer 5 with the best initialization of weights (left) and the initialization of weights closest to the average (right).



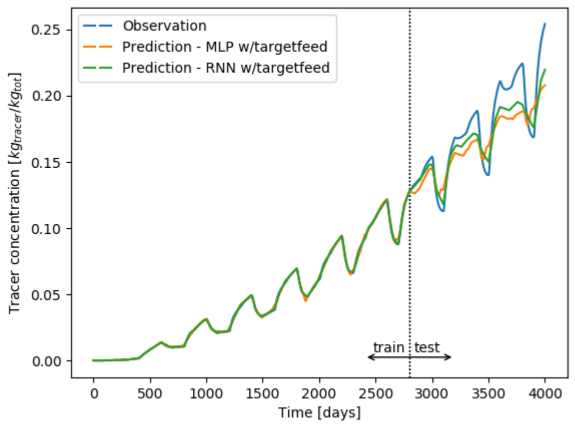


Figure 17: Predicted tracer concentration with target feed for Producer 5 with the best initialization of weights (left) and the initialization of weights closest to the average (right).

3.3 Case 2: Creating One Neural Network Model for All Producers Combined

In our second case we trained a neural network that models the relationship between all the injection wells and all the production wells. This model resembles a conventional reservoir simulator the most, where we want to match all the produced data at once. As for Case 1, without adding target feed, the form of the input X is the same, $[Q(t)_{I1}, Q(t)_{I2}, Q(t)_{I3}, Q(t)_{I4}]$, but now the output Y takes the form $[C(t)_{P1}, C(t)_{P2}, C(t)_{P3}, C(t)_{P4}, C(t)_{P5}]$. The best model configuration for this case was the same as for Case 1, and adding complexity such as more neurons or layers, did not improve the accuracy. Table 3 shows the average R^2 after initializing the model multiple times for both MLP and RNN, with and without target feed, and Figures 17 and Figure 19 show the performance of the models.

Table 3: The average test error (R^2) and standard deviation for the best model configurations in Case 1.

	MLP	RNN	MLP w/target feed	RNN w/target feed
All Producers	0.84 ± 0.06	0.69 ± 0.15	0.94 ± 0.03	0.86 ± 0.09

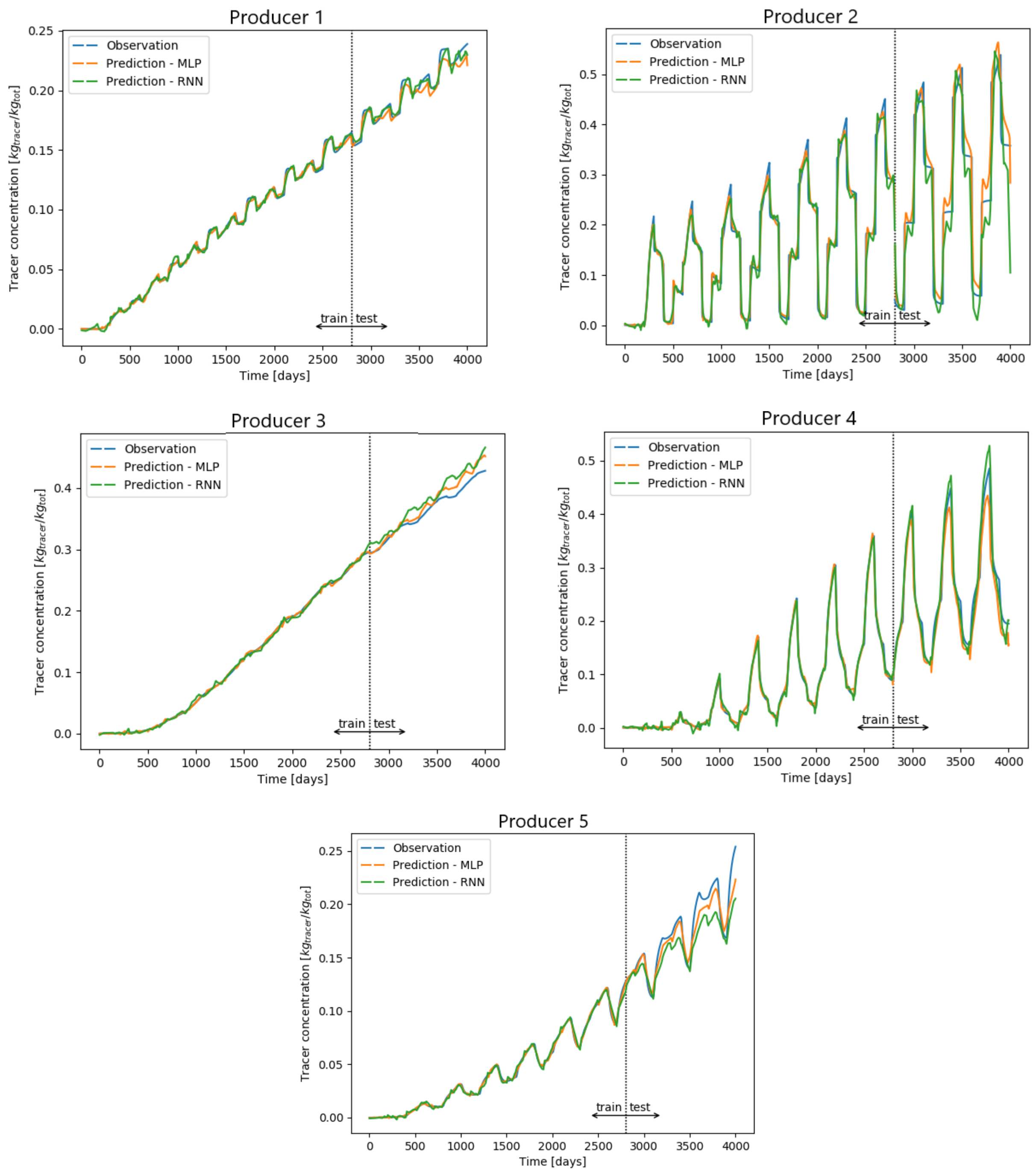


Figure 18: The tracer concentration of all producers when modeled with a single MLP and RNN without target feed.

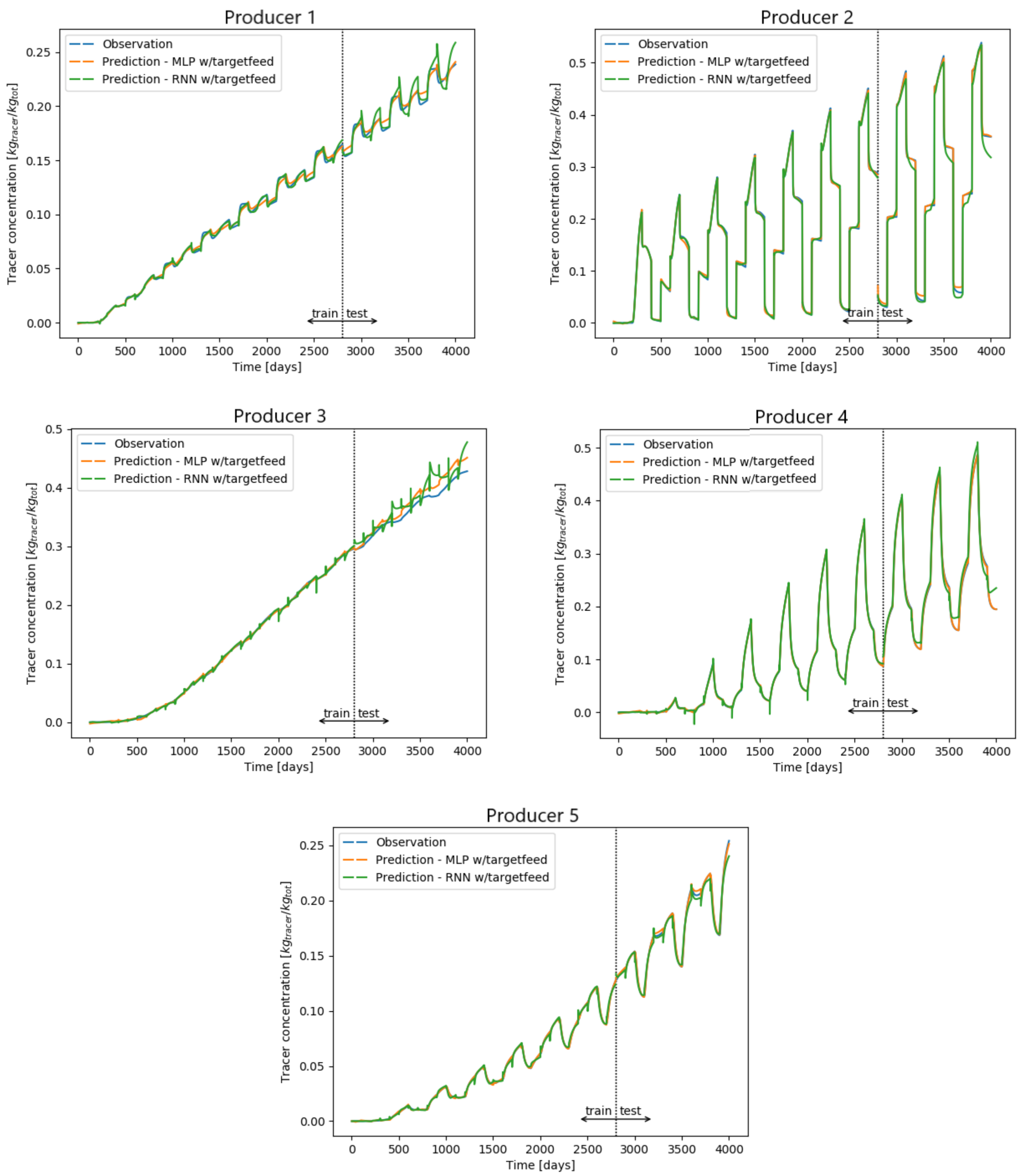


Figure 19: The tracer concentration of all producers when modeled with a single MLP and RNN with target feed.

4. DISCUSSION AND CONCLUSION

For this synthetic geothermal reservoir, the simple feedforward neural network (MLP) outperformed the more flexible structure of the recurrent neural network. Adding the target feed significantly elevated the performance of the models and in the case of MLP we were able to simplify the model architecture by reducing the number of hidden layers needed. The MLP was also much quicker to train, compared to the RNN, with fewer parameters to estimate. For example, for a neural network with 50 neurons and 5 hidden layers, a MLP has 10,901 parameters to train whereas a RNN has 23,001 parameters. In this case the training of the RNN takes five times longer compared to the training of the MLP.

When creating one model to predict for all producers, compared to having separate models, we again found that the MLP performed better than the RNN. It should though be noted that because training RNN is more computationally expensive and takes longer to train, it is possible to find model configurations that perform better than what we present here. For this case it was interesting to see that it is more difficult for the models to capture the linear behavior of Producer 1 and 3 compared to the more characteristic behavior of Producer 2, 4, and 5. This is because when training, the model will prioritize fitting the characteristic behavior because any discrepancies there have much more effect on the overall loss than for the more linear curves.

ACKNOWLEDGEMENTS

We would like to thank the Energy Resources Engineering Department at Stanford University and Landsvirkjun, National Power Company of Iceland for the financial support that made this research possible.

REFERENCES

Alakeely, A., & Horne, R. N. (2020). Simulating the Behavior of Reservoirs with Convolutional and Recurrent Neural Networks. In *International Petroleum Technology Conference, Dhahran, Saudi Arabia, January 13-15*.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. Retrieved from http://www.deeplearningbook.org

Li, Y., Sun, R., & Horne, R. (2019). Deep learning for well data history analysis. *Proceedings - SPE Annual Technical Conference and Exhibition, Calgary, Alberta, Canada, September 30 - October 2*.

Liu, Y., & Horne, R. N. (2013). Interpreting Pressure and Flow Rate Data from Permanent Downhole Gauges Using Convolution-Kernel-Based Data Mining Approaches. In *SPE Western Regional & AAPG Pacific Section Meeting, California, USA*.

Tian, C. (2018). *Machine Learning Approaches for Permanent Downhole Gauge Data Interpretation*. Stanford University.

Tian, C., & Horne, R. N. (2015). Applying Machine Learning Techniques to Interpret Flow Rate, Pressure and Temperature Data From Permanent Downhole Gauges. In *SPE Western Regional Meeting, California, USA, April 27-30.*

Tian, C., & Horne, R. N. (2017). Recurrent Neural Networks for Permanent Downhole Gauge Data Analysis. In *SPE Annual Technical Conference and Exhibition, San Antonio, Texas.*