MACHINE LEARNING APPLICATIONS IN RESERVOIRENGINEERING

Loc Luong

November 2, 2024

Contents

List of Figures

List of Tables

1 Introduction

History matching is considered as one of the most important tasks in reservoir engineering and it is defined as the process of finding an appropriate set of parameters which is the best representative of reservoir. History matching is an ill-posed problem, which means there is no unique solution to the problem. Due to the complexity and computation expense of reservoir simulation runs, there is an alternative method of coupling neural networks and particle swarm optimization to solve the history matching problem.

In this final project, we are about to design machine learning and artificial intelligence models to complete the history matching of an oil field. The oil field has been producing for 50 years.
The data contains all of the information of dynamic simulation and historical production. There are a total of 187 simulation runs with certain output, and 15 reservoir properties within the corresponding ranges.

Table 1: Range of parameters

Parameters	**Upper range**	**Lower range**
Parameter 1	149.9332	21.43146
Parameter 2	0.348973	0.051029
Parameter 3	0.349233	0.050216
Parameter 4	0.298431	0.100116
Parameter 5	0.099472	0.000123
Parameter 6	6.961201	1.132957
Parameter 7	6.98553	1.109156
Parameter 8	6.997306	1.156175
Parameter 9	6.918394	1.091107
Parameter 10	0.897132	0.203404
Parameter 11	0.999727	0.503513
Parameter 12	0.997875	0.801705
Parameter 13	0.099497	0.005689
Parameter 14	7.954186	0.138183
Parameter 15	7.919723	0.105462

2 Overview of neural network

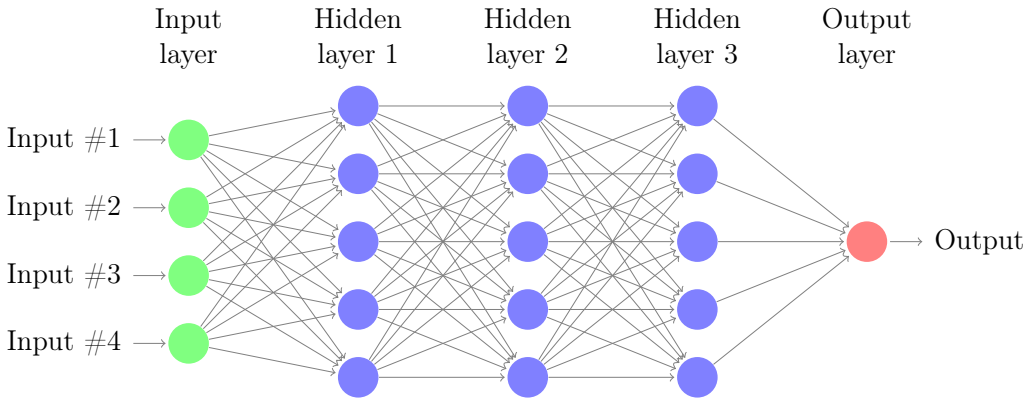


Figure 1: Multi-layer neural networks

In a typical multi-layer neural networks; beside the input and output layers, there might have many hidden layers in the middle. Figure 1 is an example of fully connect multi-layer neural network with three hidden layers. The number of layers in a neural networks are calculated by the number of hidden layers plus one.
A circle node in a layer is called a unit. Units in the input layers, hidden layers, and output layers are often called input units, hidden units, and output units, respectively. The input of the hidden layers is denoted by z, the output of each unit is usually denoted as a.
There are L matrix of coefficients for a neural networks with L layers. These matrices are often denoted by $\mathbf{W}^{(l)} \in R^{d^{(l-1)} \times d^{(l)}}$, $l = 1, 2, \ldots, L$, and $\mathbf{W}^l$ represent the connection between layer $(l-1)$ and layer (l). Biases of layer (l) usually denote by $\mathbf{b}^{(l)} \in R^{d^{(l)}}$. In optimizing the loss function of neural networks, we need to find the values of these weights $(\mathbf{W}^{(l)})$ and biases $(\mathbf{b}^{(l)})$. We can define $\boldsymbol{\theta}^{(l)} = [\mathbf{b}^{(l)}, \mathbf{W}^{(l)}]$ is the matrix of coefficients of layer (l).
The output from each layer in vector form can be defined as :

$$\mathbf{a}^{(l)} = f(\mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}) = f(\boldsymbol{\theta}^{(l)}\mathbf{a}^{*(l-1)})$$

where $\mathbf{a}^{(l-1)}$ is the output from the layer $(l-1)$ (note that $\mathbf{a}^{(0)}$ is the input data), and f is the activation functions.
We can write this in different form with respect to θ.

$$\mathbf{a}^{(l)} = f(\boldsymbol{\theta}^{(l)}\mathbf{a}^{*(l-1)})$$

where $\mathbf{a}^{*(l-1)}$ is the concatenate form of column vector of 1 and $\mathbf{a}^{(l-1)}$.
In regression problem, the loss function of neural network is usually mean

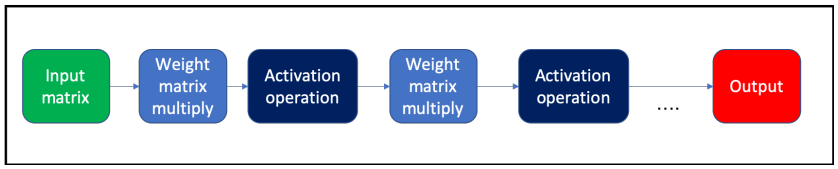
square error (MSE) which is the sum of square of the mismatch between model responses and actual data. The loss function can be defined by:

$$J(\mathbf{W}, \mathbf{b}) = \frac{1}{N} \sum_{i=1}^{N} \|y_i - \hat{y}_i\|_2^2$$

where: $\mathbf{W}, \mathbf{b}$ are the set of all matrices of coefficients and biases in each layer, y_i is the predicted output from neural network, and $\hat{y}_i$ is the actual value getting from datasets. Or it can be written by:

$$J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \|y_i - \hat{y}_i\|_2^2$$

where $\boldsymbol{\theta}$ is the set of coefficients as defined above.
The process of getting output from neural networks is shown in the figure below:



In training neural network, gradient descent and its variants are by far the most popular technique. Gradient descent is an algorithm used to optimize the objective function $J(\theta)$, with respect to the parameter $\theta \in R^d$, by gradually updating the parameter θ in the opposite direction to the gradient of the objective function. The learning rate η determines the size of each jump we take to the minimum value.
The advantages of GD are the ease of implementation and low memory cost, while the drawback is the convergence time. In machine learning or deep learning area, we use this algorithm to update parameters of models or the weights (θ) in neural networks. The general definition of gradient descent is followed by:

$$\theta = \theta - \eta \nabla J(\theta)$$

where η is defined as the learning rate and negative sign means that we will go down hill for seeking the minimum.

3 Particle swarm optimization

Particle Swarm Optimization (PSO) is a population based stochastic optimization technique that inspired by either social behavior of bird flocking or fish schooling. PSO is the global optimizer which aims to find global minimum or maximum points of an objective function.
The general update rule for PSO algorithm is followed:

1. Evaluate the fitness
2. Calculate the velocity

$$v_i^{t+1} = wv_i^t + c_1 r_1 (pbest^t - x_i^t) + c_2 r_2 (gbest^t - x_i^t)$$

3. Update the particle position (or solution)

$$x_i^{t+1} = x_i^t + v_i^{t+1}$$

The hyper-parameters used in PSO are:

- w is an inertial weight usually slightly less than 1.
- c_1 is the cognitive weight, represents the impact of personal best on particle's movement.
- c_2 is the social weight, represents the impact of global best on particle's movement.
- r_1,r_2 are two random components, represent the randomness of the process of update particle' velocity. These two values also represent the stochastic search of particle.

The intuition behind PSO algorithm is particles will communicate the information they find with others, and update their velocity based on the local and global best particles. The swarm will update best result as gbest when the new global best is found, and each particle tend to move toward the personal best and global best.
The set up values of c_1,c_2 are often 2 or in ranges of $[0, 4]$, and r_1,r_2 are between 0 and 1.

4 Methodology

4.1 Work-flow

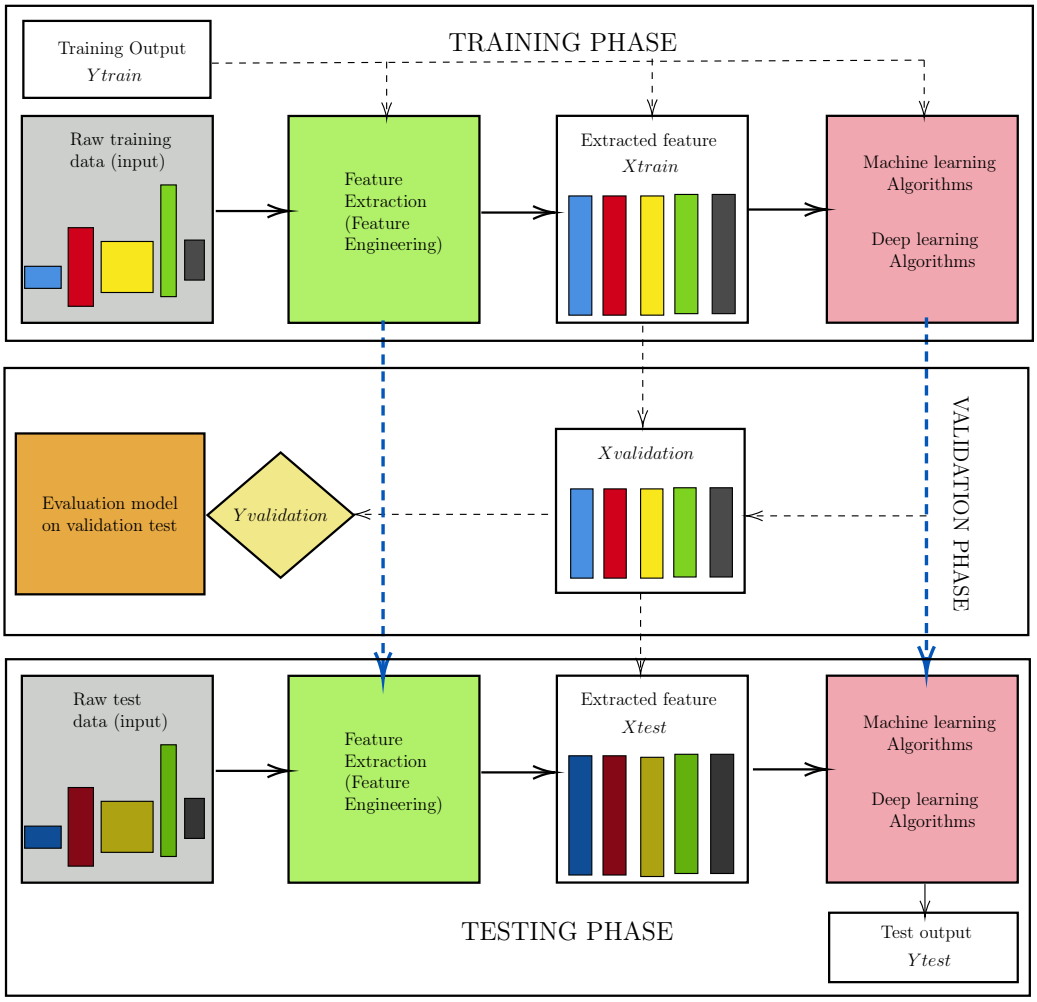


Figure 2: Work flow in training machine and deep learning algorithms

Almost every machine learning and deep learning algorithm have followed this work-flow above. There are three phases in this process which are training, validation and testing phase. In training phase, we need to design two blocks (green and pink), which are feature extractors and main algorithms. The input data is all information we know about the data. For example, for an image, it is the value of each pixel in image; for audio, it is the signal. In this project, the raw input might be all information reservoir including permeability, porosity, relative permeability, bottomhole pressure or initial

state, etc. The raw data is not usually in vector form, without the same dimensions and often contains noise. Missing values problems might encounter also. To get more usefull and clean data, we need to design the feature extractor. After feature extractor phases, we will get extracted features from raw input. These features will be used to train the machine learning or deep learning algorithms.

In main algorithms, once we have extracted features, we use this information along with (optional) training output to create appropriate models for our dataset and objective. An important point is that when building feature extractors and main algorithms, we should not use any information in the test data set. We have to assume that the information in the test data has never been seen.

In testing phase, with the new raw input, we use the feature extractor created above to create the corresponding testing feature vectors. Using these testing feature vectors and the main algorithm learned in the training phase, to predict output and evaluate the generation of our models.

4.2 Detailed work-flow in this project

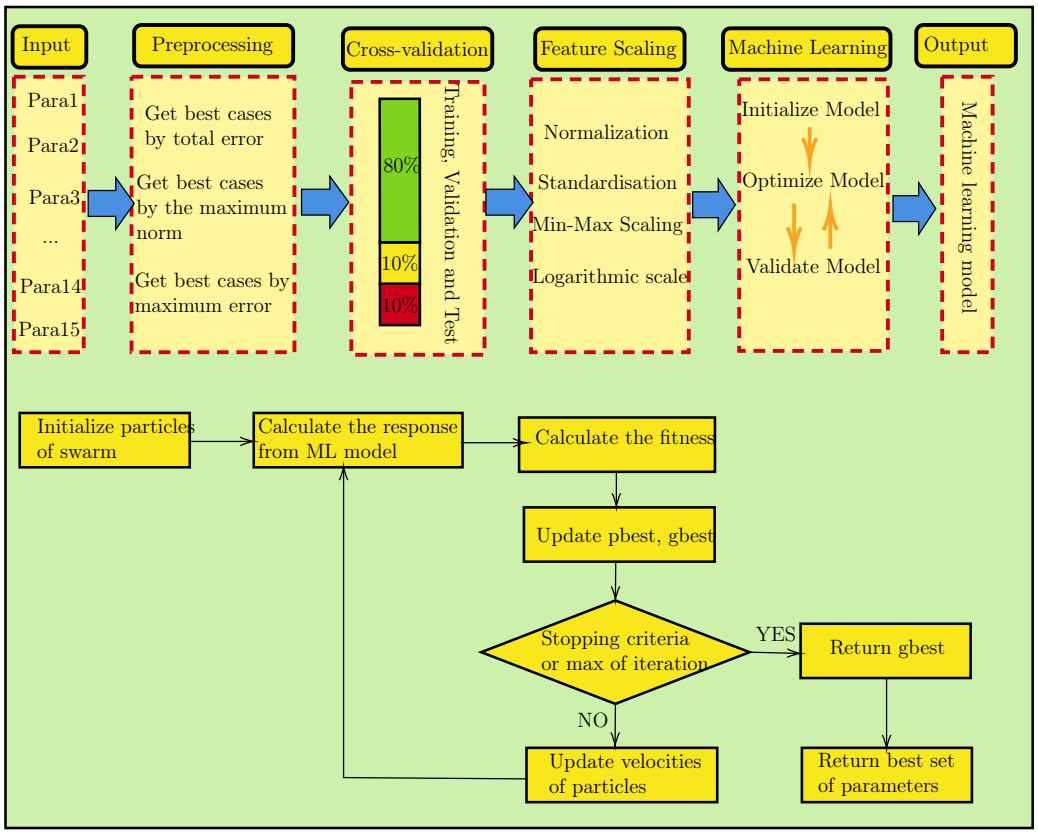


Figure 3: Detailed work-flow used in this project

The first phase is to train the neural networks with data from simulation runs. The input data is 187 parameters sets including information about reservoir. The output is production rate from simulation.
In the preprocessing section, we use three techniques to achieve the suitable numerical simulation models. The first one is to compare the overall mean square error between historical production and simulation models.

$$MSE = \sum_{i=1}^{50} (y_i - \hat{y}_i)^2 \tag{1}$$

where y_i is the historical production rate and $\hat{y}_i$ is the simulation cases. Because the overall mean square error is not reflex enough the mismatch between historical data and simulation cases, we use the infinity norm to find the minimum mismatch between these two data sets.

$$\|y - \hat{y}\|_\infty = \max(|y_i - \hat{y}_i|)$$

We will choose the simulation cases which have the minimum infinity norm. Another problem is that the maximum value of history production is about 50,000. So, we should choose the simulation cases which have the maximum production rate close to this value, not too high or not too low. Hence, we come up with the new formulate of getting best cases by comparing maximum values.

$$\begin{aligned} max\_history &= \max y_i \\ max\_cases &= \max \hat{y}_i \\ Error &= |max\_history - max\_cases| \end{aligned}$$

By setting the constraints of maximum value of production rate, the neural networks model will have more reliable results.
In cross-validation strategies, we will divide the whole data set into three chunks. We use 80% of available data for training set, a subset of 10% of data for validation, and the remaining 10% for testing set.
We also use four techniques in feature scaling including normalization, standardization, min-max scale, and logarithmic scale. By comparing these methods, we will select the logarithmic scaling (log10) due to the high reliability and accuracy of this method.
Once we get the best machine learning model from previous phase, in order to obtain the appropriate combination of reservoir parameters, particle swarm optimization (PSO) is coupled with neural networks. The detailed work-flow of training PSO algorithm has shown in figure 3.

5 Result and discussion

Table 2: History matching errors of 187 simulation cases

#	**Error**	#	**Error**	#	**Error**	#	**Error**	#	**Error**	#	**Error**
1	10.76	33	2.25	65	3.49	97	2.16	129	9.25	161	4.59
2	2.15	34	1.83	66	2.86	98	2.47	130	1.35	162	7.79
3	2.56	35	2.44	67	1.15	99	3.16	131	1.26	163	3.20
4	1.93	36	2.57	68	2.23	100	1.98	132	2.95	164	4.00
5	1.78	37	3.95	69	2.28	101	10.7	133	1.83	165	2.10
6	2.24	38	2.12	70	6.17	102	1.78	134	2.2	166	1.46
7	1.69	39	1.63	71	2.87	103	2.60	135	2.54	167	2.63
8	1.37	40	2.95	72	2.93	104	3.20	136	5.76	168	3.38
9	4.95	41	1.91	73	2.04	105	2.47	137	3.25	169	1.67
10	1.92	42	4.48	74	3.62	106	5.75	138	2.149	170	3.42
11	2.48	43	8.50	75	1.76	107	1.21	139	5.95	171	2.10
12	1.7	44	2.32	76	2.12	108	6.86	140	2.83	172	3.16
13	10.77	45	3.38	77	1.71	109	9.75	141	1.47	173	2.41
14	4.4	46	1.84	78	4.59	110	1.77	142	4.17	174	2.50
15	2.21	47	5.95	79	2.12	111	8.09	143	2.29	175	1.62
16	4.12	48	2.75	80	7.79	112	3.47	144	1.43	176	2.75
17	3.62	49	1.25	81	2.35	113	2.29	145	1.33	177	1.46
18	1.99	50	1.69	82	4.88	114	4.73	146	1.87	178	2.77
19	2.72	51	1.97	83	1.66	115	4.76	147	2.04	179	9.86
20	2.25	52	2.23	84	10.61	116	3.52	148	1.03	180	7.34
21	2.93	53	3.28	85	1.47	117	2.04	149	3.56	181	10.59
22	2.23	54	3.42	86	3.52	118	5.01	150	1.62	182	11.41
23	10.37	55	2.21	87	1.89	119	3.25	151	3.62	183	4.07
24	4.46	56	2.05	88	10.05	120	3.00	152	3.41	184	10.74
25	3.35	57	2.18	89	8.33	121	3.04	153	2.16	185	3.99
26	2.74	58	1.79	90	1.87	122	4.08	154	1.73	186	1.90
27	1.84	59	7.46	91	2.83	123	1.95	155	2.35	187	2.16
28	1.71	60	1.88	92	8.86	124	2.77	156	2.32		
29	2.04	61	3.02	93	4.04	125	8.49	157	2.00		
30	2.32	62	1.94	94	7.53	126	4.71	158	1.89		
31	3.41	63	7.75	95	2.95	127	1.37	159	1.64		
32	3.71	64	1.79	96	1.31	128	2.79	160	2.67		

We use the root mean square error for the history error between historical production and simulation production as defined by.

$$RMSE = \sqrt{\sum_{i=1}^{50}(y_i - \hat{y}_i)^2} \tag{2}$$

To calculate the history matching error, we applied the logarithmic scale to both production data and simulation cases.
The history matching errors of 187 simulation cases is shown in the table 2.

5.1 Case 1

In case 1, we use all 187 simulation runs as the input to train the model. The result has shown in the figure 4.

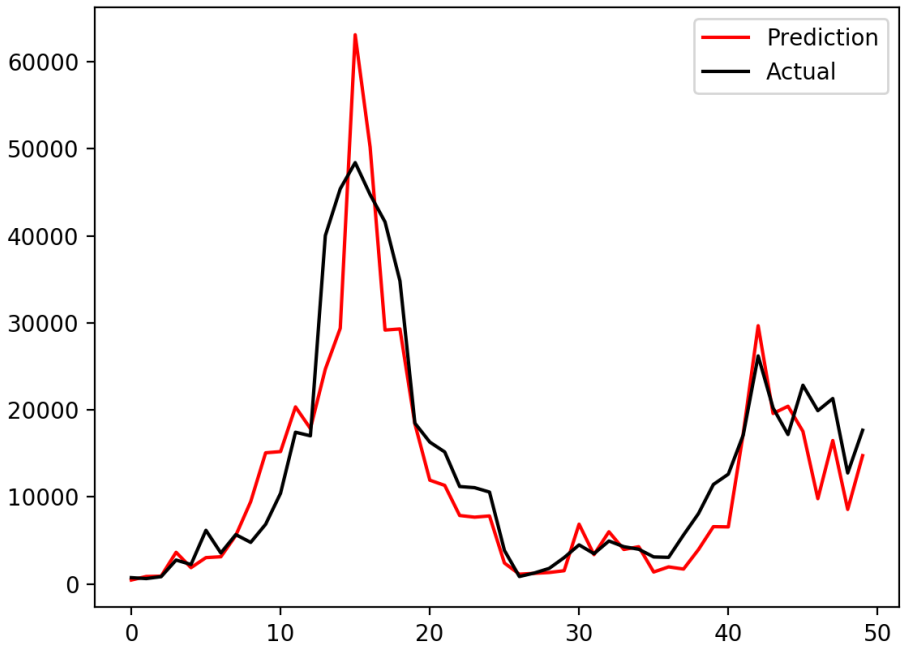


Figure 4: History matching result on case 1

The combination of 15 parameters is:

$$\begin{bmatrix} 105.68728590461802, 0.170832265211919, 0.23356042703056365, \\ 0.17439602603636412, 0.0698021099232469, 6.6414616944308085, \\ 3.863585262406055, 4.90446817444671, 6.115552955169507, \\ 0.6046652150658747, 0.7623751446615195, 0.8782655215431764, \\ 0.025625761286396886, 4.284290842722475, 5.050903830356328 \end{bmatrix}$$

There are several ways to define the objective function (or loss function) of PSO. Two of the most popular objective function is mean square error (MSE) or root mean square error (RMSE). The mean square error is defined as same as the equation 1, and root mean square error as equation 2.
In this project, we use the infinity norm to define the objective function of PSO algorithms. The objective function is follows:

$$\|y - \hat{y}\|_\infty = \max(|y_i - \hat{y}_i|)$$

We will find the maximum absolute value between production history and model, then minimize that value. If the value is small enough to ensure the minimum mismatch between production history and neural networks model, the process will be stopped. In this case the best fitness is 0.772. This best fitness indicate that the maximum mismatch between production history and neural networks is 0.772.

After finding the best case, we will change the objective function to the RMSE and it's defined as the 2 norm:

$$\|y - \hat{y}\|_2 = \sqrt{\sum_{i=1}^{50} (y_i - \hat{y}_i)^2}$$

5.2 Case 2

In case 2, we compare the maximum value of historical data and simulation cases and pick the first 20 cases that have the minimum errors. The disparity of actual production and neural networks model is shown in figure 5, figure 6 shows the comparison of cumulative production.
The parameters are:

$$\begin{bmatrix} 57.14564235917216, 0.348972798, 0.050216212, \\ 0.10011580099999999, 0.09947186, 1.132956938, \\ 6.985529649, 1.8425768315803916, 1.091106906, \\ 0.203403941, 0.50351302, 0.9978746909999999, \\ 0.09949744699999999, 0.138182724, 0.105462389 \end{bmatrix}$$

And the best fitness is : 0.4837.

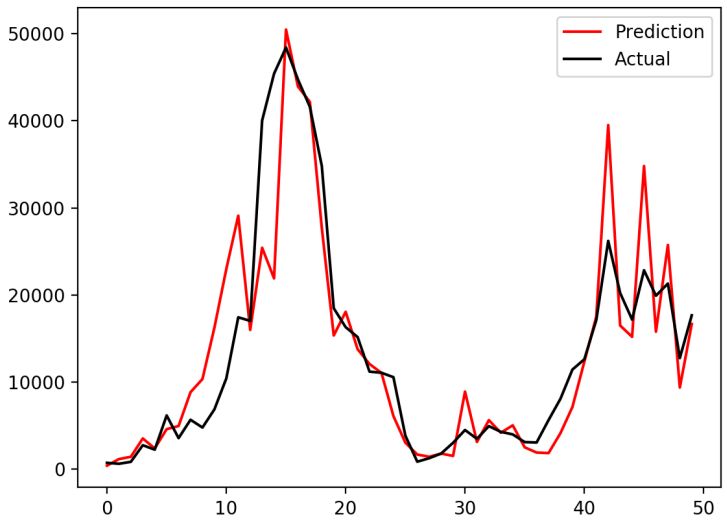


Figure 5: History matching result on case 2

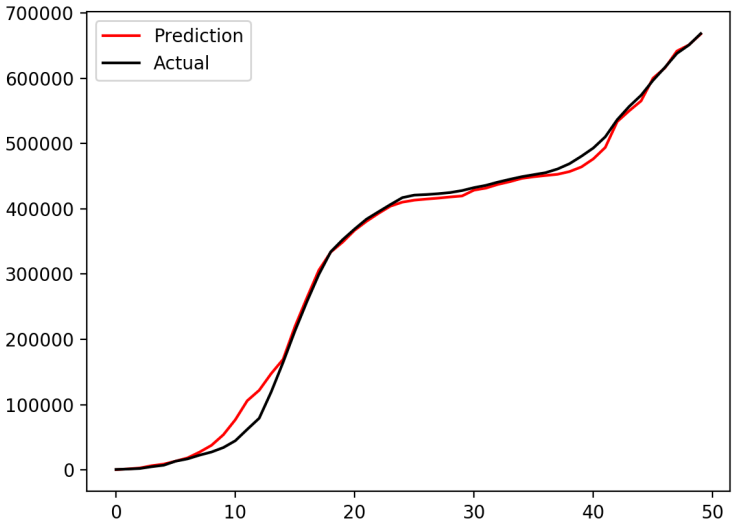


Figure 6: Cumulative production history comparison case 2

5.3 Case 3

In case 3, we compare the maximum value of historical data and simulation cases and pick the first 30 cases that have the minimum errors. Then we choose the first 20 cases which yield the minimum overall mean square error compared with production history. Finally, 20 cases from infinity norm errors are chosen. A combination of these three types of reservoir simulation runs has been fed to neural networks. The results are showed in figures 7 and 8. It can be seen that this case obtains a better history matching compared with case 2.
The parameters are:

$$\begin{bmatrix} 92.89322147276508, 0.051028982, 0.349233046, \\ 0.298431349, 0.00012343, 6.961200971, \\ 6.985529649, 1.15617542, 6.69166591254424, \\ 0.203403941, 0.999726859, 0.801704905, \\ 0.005688953, 6.297081206379236, 0.105462389 \end{bmatrix}$$

And the best fitness is : 0.445.

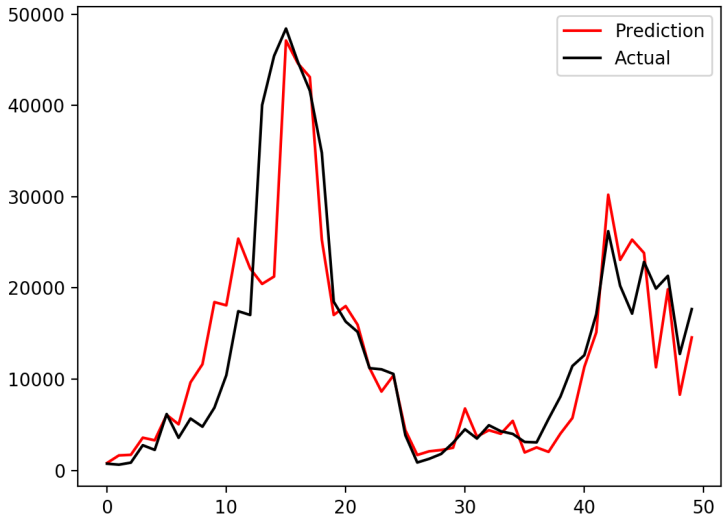


Figure 7: History matching result on case 3

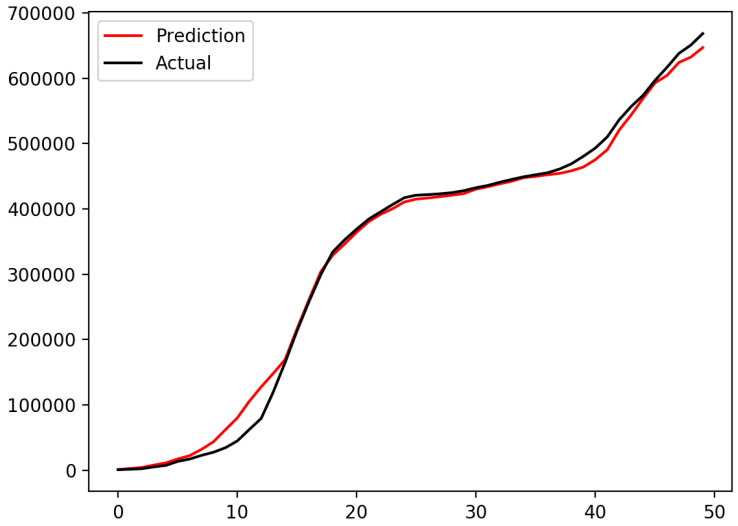


Figure 8: Cumulative production history comparison case 3

5.4 Case 4

Similarly to case 3, in this case we also compare the maximum value of historical data and simulation cases and pick the first 20 cases that have the minimum errors. Then we choose the cases from 10 to 20 which yield the overall mean square error compared with production history. Finally, cases from 10 to 20 of infinity norm errors are chosen. Although this case gives the best fitness compared with other cases, there is still a bigger gap between production history and neural networks model. The parameters are:

$$\left[\begin{array}{c} 36.11316909597768, 0.051028982, 0.050216212, \\ 0.298431349, 0.00012343, 3.222388597772492, \\ 2.7889631373832016, 2.3091741535129, 6.918394162999999, \\ 0.203403941, 0.9420679848456645, 0.9978746909999999, \\ 0.005688953, 2.788466442799089, 0.105462389] \end{array}\right]$$

And the best fitness is : 0.2877.

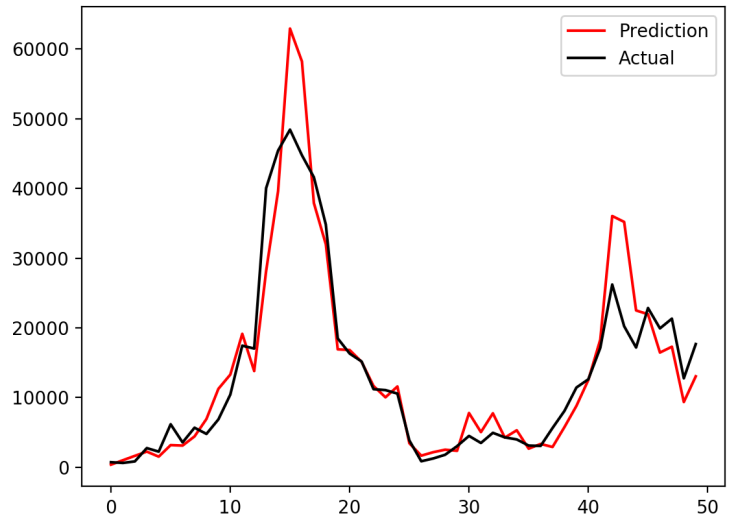


Figure 9: History matching result on case 4

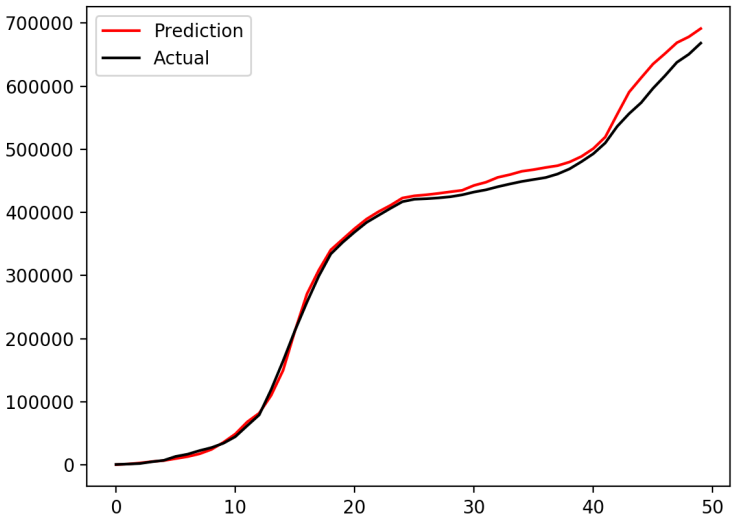


Figure 10: Cumulative production history comparison case 4

5.5 Case 5

Case 5 is derived from case 3, but with different objective function of particle swarm optimization. We use RMSE error function in order to compare with simulation results from table 2. The objective fucntion as follows:

$$RMSE = \|y - \hat{y}\|_2$$

The best fitness, in this case, is 1.124 (RMSE). This value is by far the lowest error between actual production and model, compared to errors of simulation runs from table 2. The parameters are:

$$\begin{bmatrix} 149.9332127813718, 0.051028982, 0.349233046, \\ 0.2984313463406118, 0.00012343, 6.961200441367357, \\ 6.985529649, 3.711276783748969, 6.918394162999999, \\ 0.203403941, 0.999726859, 0.9978746909999999, \\ 0.005688953, 6.619052359946135, 0.105462389 \end{bmatrix}$$

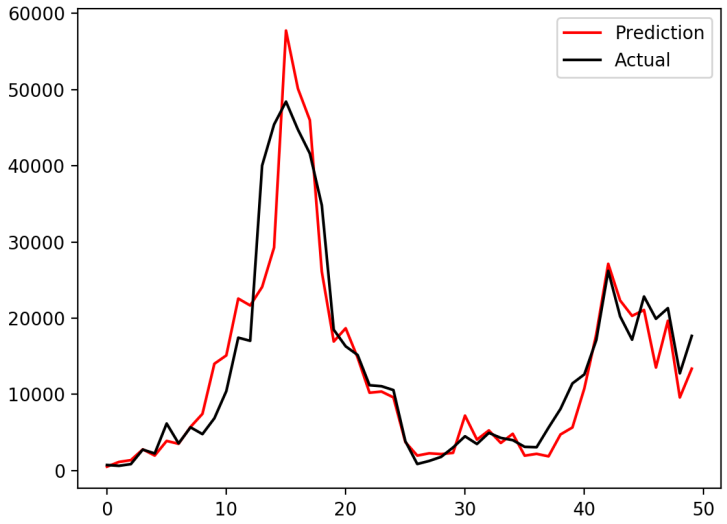


Figure 11: History matching result on case 5

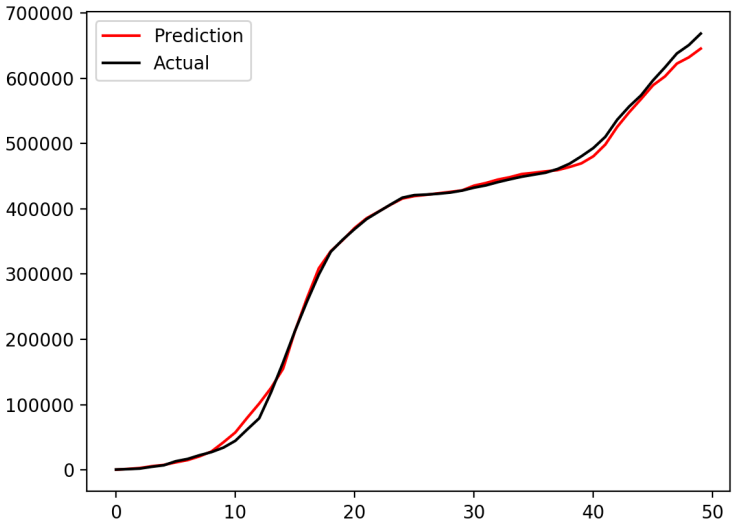


Figure 12: Cumulative production history comparison case 5

5.6 Case 6

In case 6, we use the weighted sum to define the objective function of PSO.

$$0.2 * \|y - \hat{y}\|_2 + 0.8\|y - \hat{y}\|_\infty$$

Although the root mean square error of this case is 1.149, which is slightly bigger than case 5, the result is slightly better than case 5 as shows in the figures 13 and 14. The parameters are:

$$\begin{bmatrix} 125.9601971743832, 0.051028982, 0.349233046, \\ 0.298431349, 0.00012343, 6.961200971, \\ 6.985529649, 1.15617542, 6.918394162999999, \\ 0.203403941, 0.999726859, 0.801704905, \\ 0.005688953, 7.26703499988773, 0.105462389 \end{bmatrix}$$

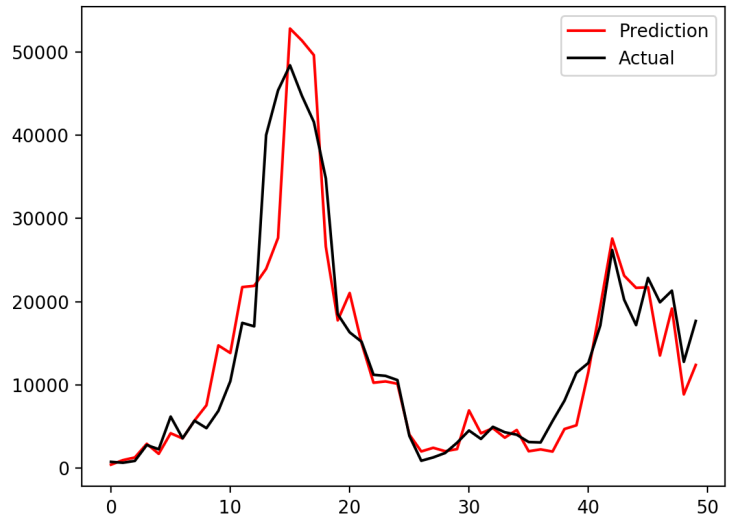


Figure 13: History matching result on case 6

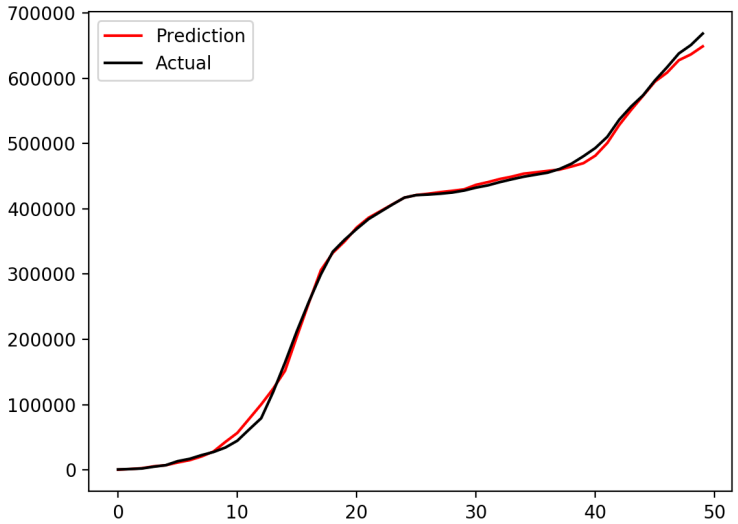


Figure 14: Cumulative production history comparison case 6

5.7 Result of training neural networks

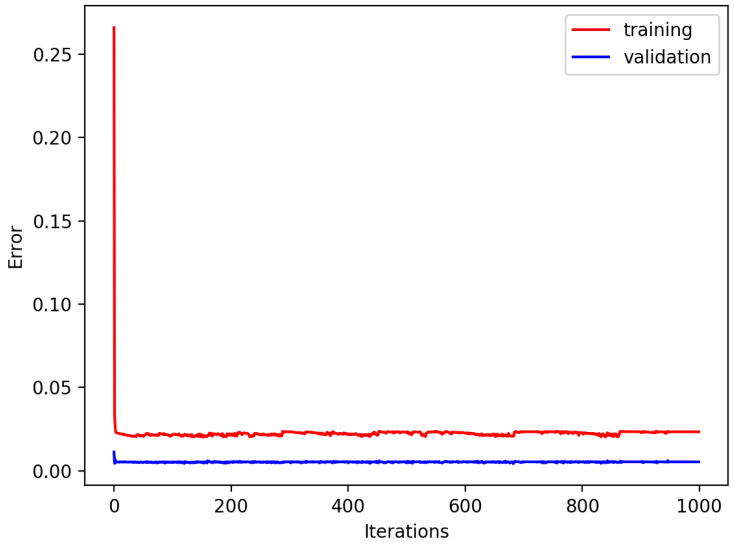


Figure 15: Error during training neural networks of case 1

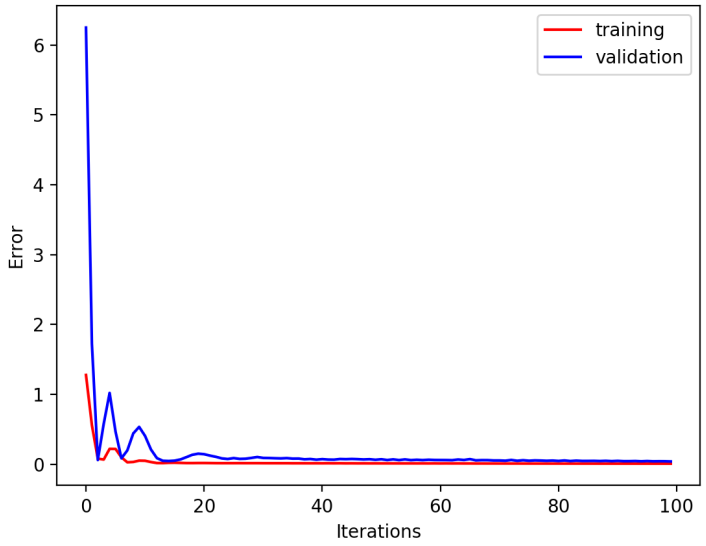


Figure 16: Error during training neural networks of case 2

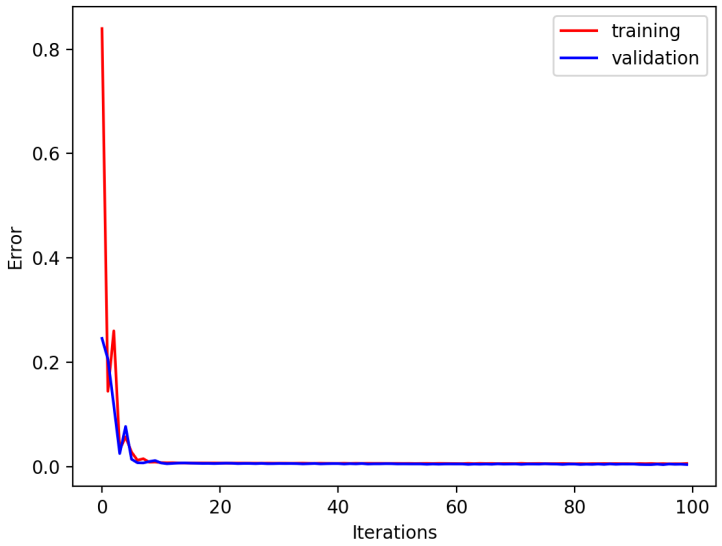


Figure 17: Error during training neural networks of case 3

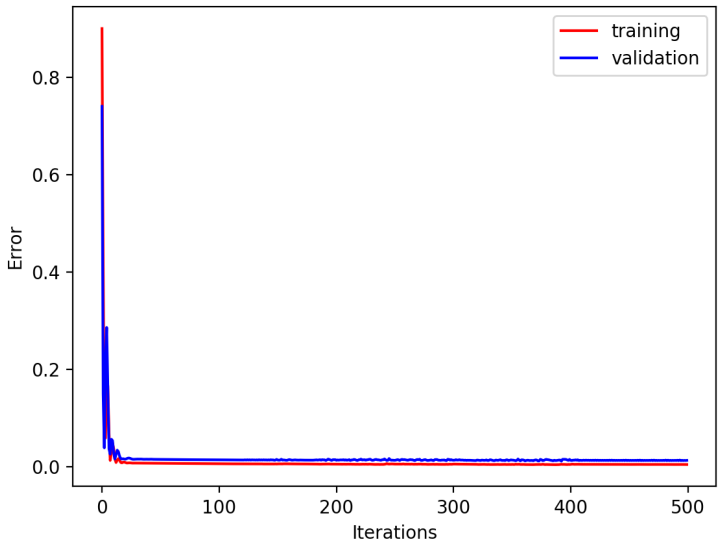


Figure 18: Error during training neural networks of case 4

6 Summary and recommendation

In this project, a machine learning model has been built coupled with particle swarm optimization to solve the history matching problem of an oil field. In terms of overall performance, the history matching case 3 as shown in the result section gives the best match compared with other cases.

The production prediction only from neural networks models. In order to get the reliable and accurate history matching results, simulation runs should be performed with different scenarios and combination of reservoir properties.

In this study, only multilayer fully connected neural networks have been studied. To get better comparison on the a wide range of neural networks on history matching problem, further study can be implemented by using other type of neural networks or deep neural networks, such as Restricted Boltzmann Machine (RBM), Long-short term memory (LSTM), or applied some other machine learning models like support vector machine, or multi-variable regression.

My implementation

I have used mainly python to solve the final project.

References

[1] Machine Learning Applications In Reservoir Engineering Slides. Dr. Qian Sun

Appendices

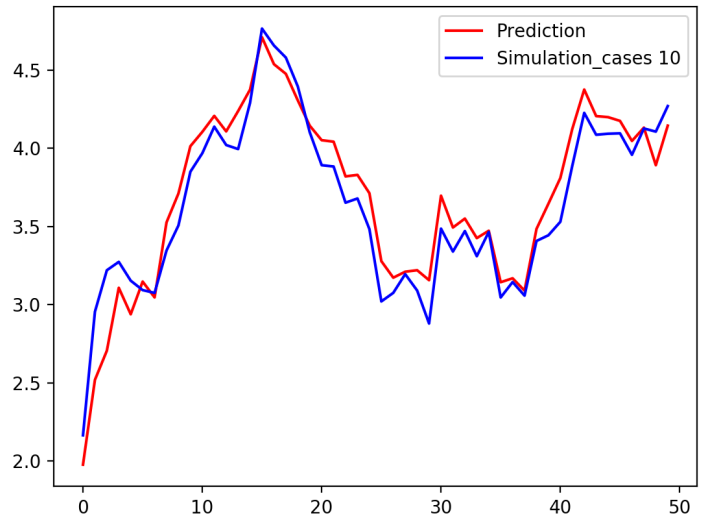


Figure 19: Result of training neural nets on simulation case 10

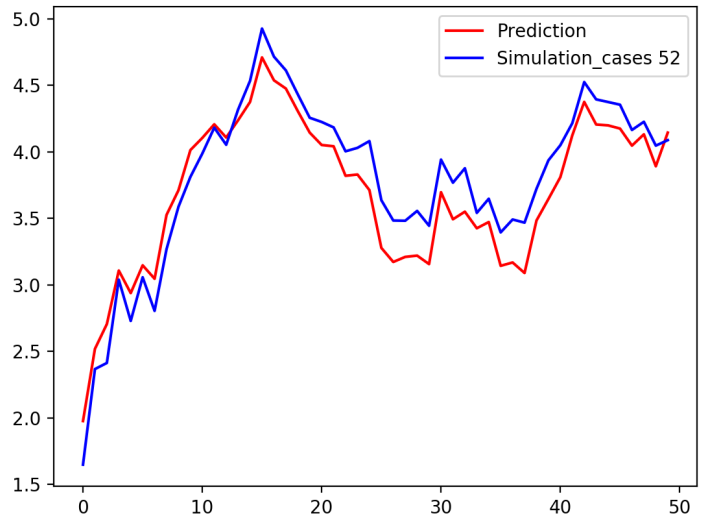


Figure 20: Result of training neural nets on simulation case 52

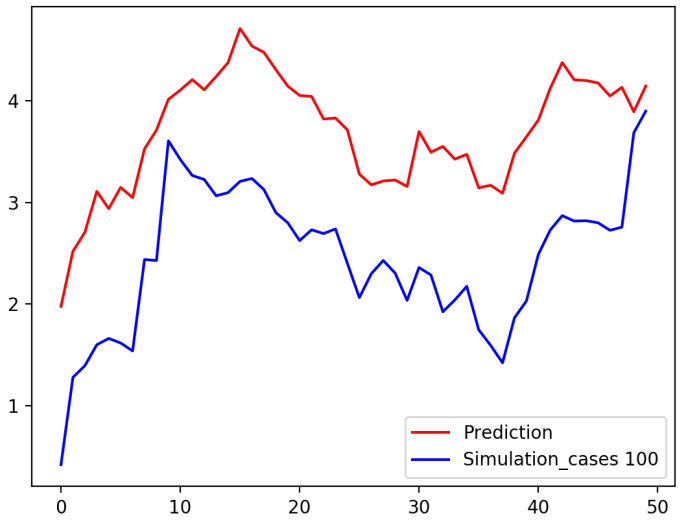


Figure 21: Result of training neural nets on simulation case 100

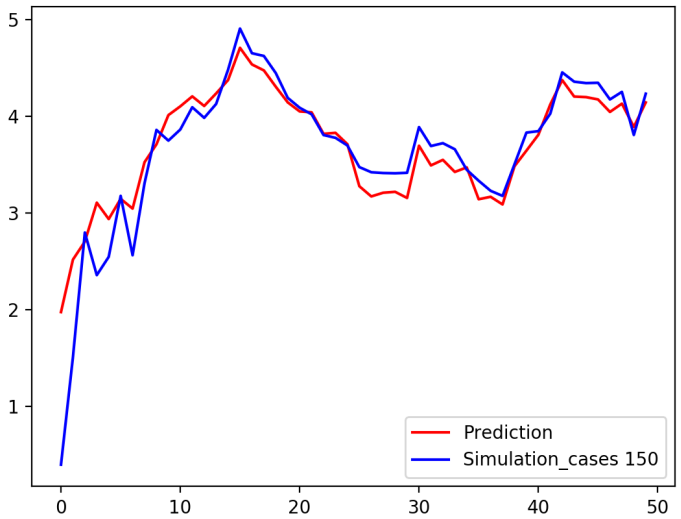


Figure 22: Result of training neural nets on simulation case 150

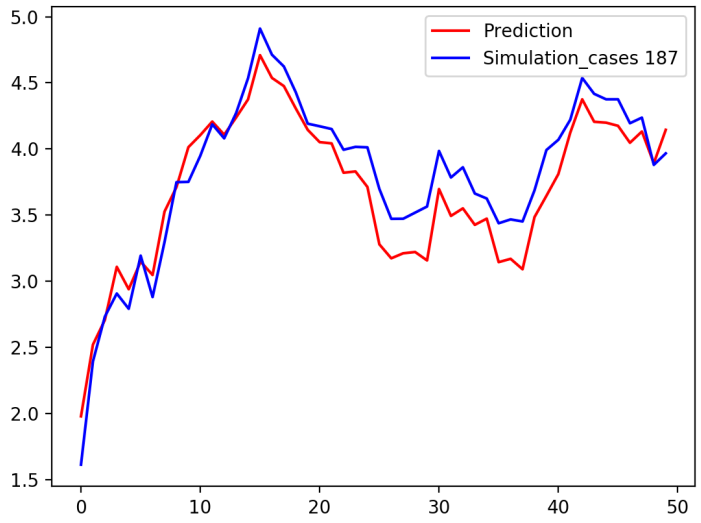


Figure 23: Result of training neural nets on simulation case 187