Machine Learning Application to A Physical System

Team Members: Abdullah Alakeely.
SUNet : alakeeaa
Category: Time series, sequence modeling.

Background:

Predicting the behavior of subsurface fields is a challenging task that require accurate knowledge of the physical properties of the field in question, and the development of a complex numerical simulation model that describe the behavior of the system (figure 1).

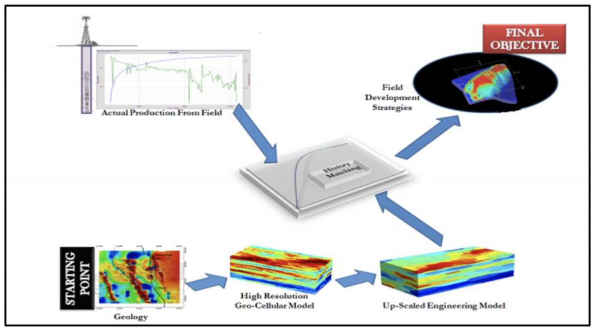

FINAL OBJECTIVE
Field Development Strategies
Actual Production From Field
History Matching
STARTING POINT
Geology
High Resolution Geo-Cellular Model
Up-Scaled Engineering Model

Figure 1: Conventional Numerical Reservoir Simulation Model Building, after Dahaghi (2009).

The development of such a model takes an enormous amount of time and effort. In addition, once the model is developed, one forward run can take from couple of minutes to a couple of weeks depending on the complexity of the underlying physical system described, Mohaghegh (2017).

Even though this modeling effort is considered the standards process describing reservoir/field behavior, the time these models take to produce results inhibit their efficient utilization when a quick answer is sought after. Not to mention, the number of forward runs in a production optimization workflow can reach a large value, and simulations runs in the order of 1000's is not uncommon, Yeten et al. (2005).

Previous work has been done to apply machine learning in similar setting in petroleum engineering to help reservoir characterization (Tian and Horne, 2015a, 2015b, 2017)

It is attractive to explore if advances in machine learning algorithms and methods, such as advances in Recurrent Neural Networks understanding, could help accelerate the process of forward molding of such simulation models with good accuracy, which is the topic of this report.

The investigation of this problem will be formulated as a supervised learning task.

Data: A simple physical model of an underground oil reservoir under water injection has been proposed. Two wells, one injects water and the other produces oil, are used in the model to generate pressure and rate profiles for 5 years. The process is repeated multiple times starting from same initial conditions, but under different production and injection rates to generate additional behaviors using the same underlying physical model, resulting in 9 different scenarios of the same reservoir. In every case/scenario, the number of changes of oil rate and water rate during 5 years period has been randomly chosen between 1 and 4. From the number of changes, a time of rate change is randomly chosen in the range [1, 1800] days.

Oil and water rates are chosen to take values randomly from the ranges of [0, 2000] bbl/day, and [-500, 0] bbl/day, respectively. Where bbl refers to the volume of production in barrels.

Task: Given rate profiles as a time series of $T\ time\ steps$, from a specific reservoir as an input feature, represented by $\left\{Q, I, \frac{dQ}{dt}, \frac{dI}{dt}\right\} \epsilon\ R^{4\ X\ T}$, with corresponding pressure profiles as an output, represented by $\{P\_Q, P\_I\} \epsilon\ R^{2\ X\ T}$.

The machine learning model should theoretically map the relationship between rate and pressure values, and re-produce the relationship in unseen scenarios produced from the same physical system.

If successful, then we can say that the machine learning model learned the reservoir behavior and able to replace the reservoir simulator to do the same takes. In practice, only one production profile from the actual reservoir through time is available, being able to reproduce the behavior of the reservoir with smallest number of training example is essential in this domain. This will be explored during training/dev/test sets discussion.

Training Strategy:

Processing Data:

The rate, and pressure vectors are scaled by maximum values then multiplied by 0.99 and offset by 0.01 to render the data in a range $\subset \{0.01, 1\}$, with the aim of speeding up convergence. Gaussian noise is also added to data to make it realistic.

Train/Dev/Test Split:

Because one of the goals is to be able to learn the reservoir behavior with the smallest number of scenarios, which will result, if successful, in learning from a low number of generated simulation runs.

Here, we start by using one sequence for training (full time series of case 2), one for dev set (case 6), and the rest are used for testing. The division could be changed based on results.

Performance Metric:

The performance of the trained model will be evaluated based on how well the model perform on training, development, and testing sets. This will be evaluated by measuring the average of sum of error of the two wells, injector and producer, where error is defined as:

$$Error = |\hat{y} - y|$$

Separating the performance this way allows for better judgment on what could be an issue in case of poor performance.

Model Architecture and Optimization Metric:

Figure 2 shows the architectures of the three models we start with.

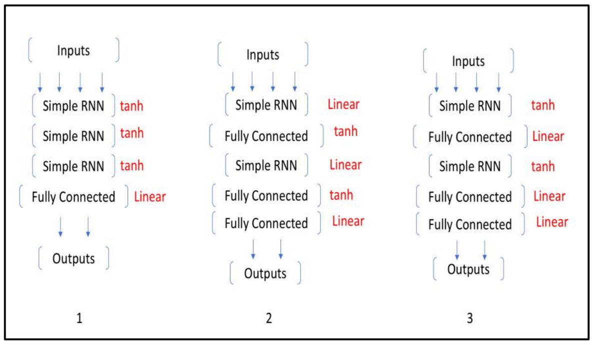


Figure 2: Recurrent Neural Network Architectures used.

Tanh activation function in some hidden layers, because we are modeling a highly non-linear system. Linear activation function is used in last layer, which is made of two neurons corresponding to two outputs. The number of hidden layers is different for different architecture ranging from 3 to 4 layers. The best values for nodes in hidden layers, n, and learning rates will be found through grid hyper-parameter search.

During training, Mean Squared Average Error between labeled pressure values, $y = \{P\_Q, P\_I\} \epsilon R^{2 X T}$, and model predicted pressure values, $\hat{y} = \{P\_Q\_predicted, P\_I\_predicted\} \epsilon R^{2 X T}$, will be used as the cost function to be optimized.

Adam with default beta values will be used as an optimizer.

Results and Discussion:

Initial Search: A grid search of the following values of learning rate = [0.0001, 0.001, 0.01, 0.1, 1], and the following values of nodes in the hidden layers, n = [5, 10, 15, 20, 25] was conducted for the proposed networks. The performance was evaluated after training for 200 epochs. Figure 3 shows the performance of architecture 1, on train/dev/test sets. A Similar trend is observed on Architecture 3, while Architecture 2 was eliminated due to producing NaN outputs, suggesting an exploding gradient issue.

This leave us with two architectures to continue with, namely: Architecture 1, and Architecture 3 (renamed architecture 2 from now on).

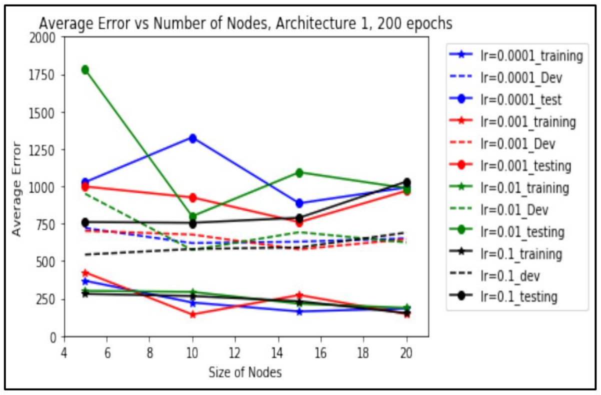


Figure 3: Architecture 1 performance.

As suggested in figure A.3, there is a clear sign of overfitting to training data, as the performance on dev and test sets are worse. However, we need to make sure the models can actually perform well on the training sets, so we perform a longer search up to 500 epochs, figure 4 shows the results of the two best performing network out of each architecture (15 nodes trained with learning rate of 0.001).

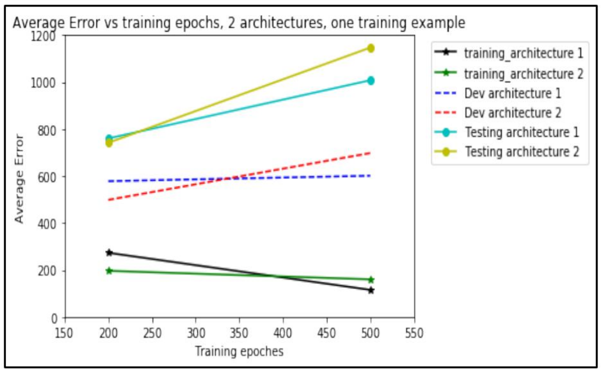


Figure 4: Best performing networks from each architecture as a function of training epochs.

As shown in the plot above, training the network longer improved the performance on training set slightly. However, the performance on both Dev and test set got worse. It is clear from the fact that we are using one scenario in both training and dev sets over fitted the network to that example.
We try now increasing the training and dev set by adding more examples to both.

Switch Sequence: As demonstrated by previous search runs, the overfitting phenomena could be reduced by adding more data. We now add more examples to both training and dev sets and we train for 500 epochs employing a switch sequence strategy, every 5 epochs as described by Horn (2009).
The training results shows that the network with architecture 1 performed better in the sense that performance on testing and dev sets is not far from training set. However, the error is still high. Figure 5 shows the performance of the best network in the search (15 nodes, and learning rate= 0.001).

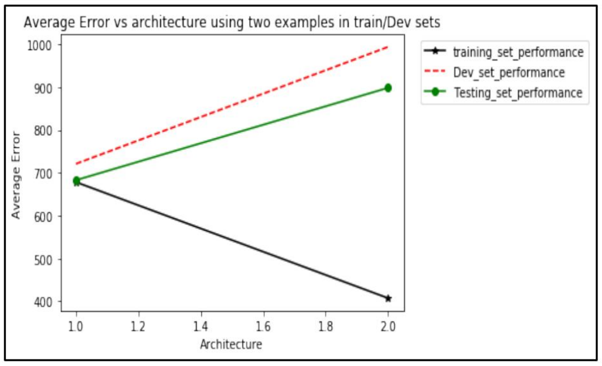


Figure 5: Architecture 1 performance.

One important observation is that the network performance is improved by utilizing switching sequence strategy.
The above results suggest that we may improve the performance by focusing on the network with architecture 1, which is what we do next.

Further Tuning: The focus now is on network with architecture 1, we repeat the experiments for 500 epochs. However, we now run them only at two node sizes, n = [15, 25], and two learning rates= [0.0001, 0.001]. In one experiment, we apply L2 regularization to the network (L2= 0.000001). In the second experiment, we apply the same regularization and we use stochastic gradient decent (SGD) optimization algorithm instead of Adam with gradient clipping set to (0.8). Table 1 shows the error results of the run for training/dev/test sets.

Table 1: Comparison of Performance of Adam (right) vs SGD (left) optimizer

Number of Nodes

Learning Rate

Training Set

	15	25		15	25
0.0001	664.532787	718.353718	**0.0001**	1013.027431	677.342989
0.0010	682.074714	667.204184	**0.0010**	716.032379	663.002491

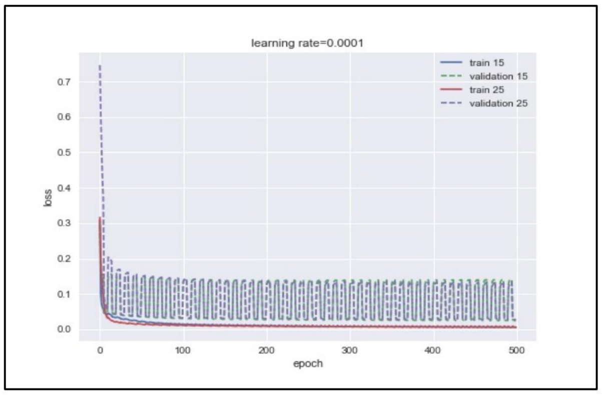
Dev Set

	15	25		15	25
0.0001	685.784764	750.764393	**0.0001**	1121.042843	708.708184
0.0010	731.097060	719.511383	**0.0010**	763.730915	741.964115

Test Set

	15	25		15	25
0.0001	683.441654	717.334051	**0.0001**	1007.185252	696.885029
0.0010	709.653391	699.205883	**0.0010**	769.911982	777.264519

Interesting to note is the outcome of this experiment. As shown in the table, the performance of regularized 15 nodes network optimized using SGD is better than the one optimized using Adam. This holds true for both learning rate values tried. The 25 nodes network results are comparable between both optimizers. However, we see that in general, SGD performed comparably or better than Adam for this task, suggesting that it might be a better optimization choice for the network and problem we are trying to model. From these experiments, network with lower learning rate (0.0001) and n=15 performed the best with SGD and below is a learning history plot comparing the learning evolution using SGD and Adam optimizers (figure 6).



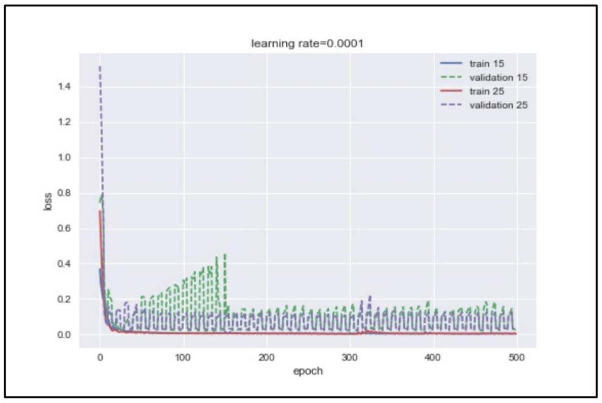
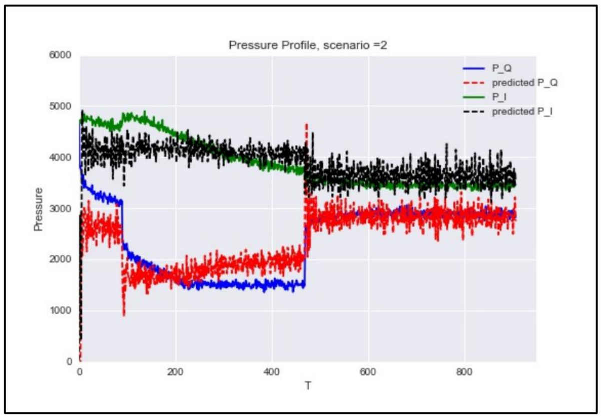


Figure 6: learning history using SGD (top) and Adam (bottom).

It is clear that the SGD optimizer had a stable learning behavior as compared to Adam where it showed an increase in loss around 150 epochs before decreasing again. However, the value it decreased to is still higher than SGD. Making SGD the better optimizer here.

Below we show some examples of the prediction results of the best performing network in our search. The network is based on architecture 1, has 15 nodes, uses SGD optimizer with lr = 0.0001, and switch sequence every 5 epochs.



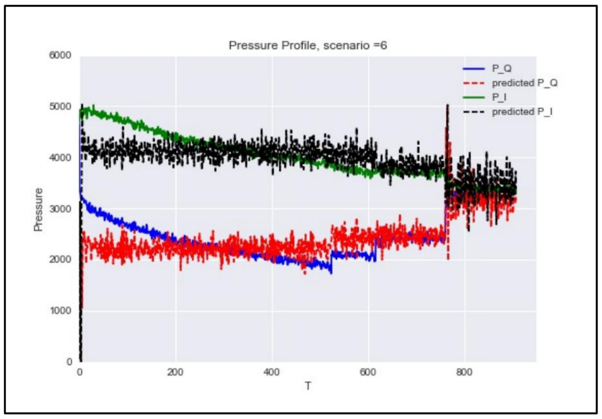


Figure 7: Examples of Prediction of trained network.

Conclusions and Future Work:

The following is observed:

1. As demonstrated, arriving at the right combination of architecture and hyper-parameter for the network is an iterative process and requires many trials. Using random search should help accelerating the process, which will be explored further.
2. There is a clear dependency between the amount and data distribution, and performance of

the network as demonstrated by improved performance by adding more data. Using the K-cross validation in a sequence switching scheme should be helpful as a hyper-parameter.
3. SGD performed better in our search, indicating that Adam might not be the best choice in every problem. More work and trials are needed to confirm the finding.
4. Other memory cells (LSTM, GRU) could be used to improve performance especially in early time dependency which was not captured properly (figure 7).

Acknowledgment: The Author take this chance to thank Professor Roland Horne for his guidance and SUPRI-D research group members for the rich discussions on AI projects.

References:

Dahaghi, A. K., & Mohaghegh, S. D. (2009, January 1). Intelligent Top-Down Reservoir Modeling of New Albany Shale. Society of Petroleum Engineers. SPE 125859-MS

Enyioha, C., & Ertekin, T. (2017, October 9). Performance Prediction for Advanced Well Structures in Unconventional Oil and Gas Reservoirs Using Artificial Intelligent Expert Systems. Society of Petroleum Engineers. SPE 187037.

Horn, J., De Jesus, O., & Hagan, M. (2009, April). Spurious Valleys in Error Surface of Recurrent Networks- Analysis and Avoidance. IEEE

Mohaghegh, S. (2017). *Data-Driven Reservoir Modeling*. Richardson, TX: Society of Petroleum Engineers.

Tian, C. and Horne, R.N., 2015a, April. Applying Machine Learning Techniques to Interpret Flow Rate, Pressure and Temperature Data from Permanent Downhole Gauges. SPE Western Regional Meeting.

Tian, C. and Horne, R.N., 2015b, September. Machine Learning Applied to Multiwell Test Analysis and Flow Rate Reconstruction. SPE ATCE.

Tian, C. and Horne, R.N., 2017, October. Recurrent Neural Networks for Permanent Downhole Gauge Data Analysis. SPE ATCE.

Yeten, B., Castellini, A., Guyaguler, B., & Chen, W. H. (2005, January 1). A Comparison Study on Experimental Design and Response Surface Methodologies. Society of Petroleum Engineers. SPE 93347-MS

Link to code:
https://github.com/alakeeaa/CS230/invitations