Applying Deep Learning to Petroleum Well Data

Janette Garcia, Akash Levy, Albert Tung, and Ruomeng (Michelle) Yang

Abstract—**In this work, we explore deep learning methods to perform time series prediction on petroleum well output. We successfully trained restricted Boltzmann machines (RBMs), fully-connected networks (FCNs), convolutional neural networks (CNNs), and recurrent neural networks (RNNs) on petroleum well data to accomplish this task. A comparison of our results indicate that neural networks are a useful tool for understanding the behavior of petroleum wells and may in many cases outperform previous methods for predicting future oil well data, both in time efficiency and accuracy.**

I. Problem Statement

OUR goal for this work is to improve upon the existing techniques used by petroleum engineers to analyze and appraise oil wells. Appraising an oil company is a lengthy investigation process; the production profiles of oil wells can be complex, driven by reservoir physics, marked by a variety of operational events and obfuscated by data noise. Petroleum engineers can usually decipher the production profiles of oil wells, understand their underlying behavior, forecast their expected production, and identify opportunities for performance improvements. However, the investigation process is time-consuming. This opens a vast portfolio of unidentified opportunities to optimize this analytic process. Thus Quantum Reservoir Impact (QRI), a petroleum reservoir management company, has proposed to explore whether deep learning algorithms, known to be effective in pattern recognition and object classification, can be used to understand and predict the behavior of oil wells. Our objective is to determine the viability of neural networks in predicting future outcomes, specifically with time series data representing oil production in petroleum wells.

II. Introduction

First introduced in the early 1980s, neural networks are a programming paradigm within deep learning that enable computers to learn from observational data. They leverage the physiology and architecture of the human brain [1]. Neural networks are generally presented as systems of interconnected neurons that send messages to each other and are used to approximate functions that depend on large amounts of input [2]. Their connections have numeric weights that are tuned based on experience, allowing them to adapt to inputs and to learn [3].

Deep learning involves relaying information from the input layer through multiple hidden layers, where representations are stored in the form of matrices and processed mathematically at each layer towards the output [4]. Various neural networks are classified by differences in the mathematical operations and layout architecture between the hidden layers. Each network is designed and optimized for jobs such as image classification, pattern recognition, and time series prediction.

A special thanks to: Verena Kaynig-Fittkau, Quantum Reservoir Impact (QRI), Institute for Applied Computational Science (IACS) at Harvard University, Harvard John A. Paulson School of Engineering and Applied Sciences.

A variety of neural networks present rich architectures to model sequential time series data of oil wells. We were particularly interested in the performance of restricted Boltzmann machines (RBMs), fully-connected networks (FCNs), convolutional neural networks (CNNs), and recurrent neural networks (RNNs). In this paper, we implement the aforementioned four deep learning models to explore the potential for neural networks to forecast this type of time series data. The task involved training the neural networks to understand oil well data and optimizing the network parameters to yield optimum results.

In the following sections, we first describe prior work relevant to the task at hand. We then discuss the neural networks we utilized to accomplish the task. After this, we explain the dataset and preprocessing methods involved in training and testing. We finally report experimental results and include discussion of our work.

III. Related Work

Deep learning models have been shown to perform well at many different tasks. Image classification is a well-known example; deep learning has been used to classify handwritten digits [5] and recognize traffic signs [6] with high accuracy. Recent research on time-series prediction has been successfully achieved on electroencephalography (EEG) data that measures cortical brain activity with temporal resolution [7] and ultra-short-term wind prediction [8]. These experiments rely on several kinds of neural networks, including deep recurrent networks, fully-connected networks, and convolutional neural networks. Work has also been conducted using recurrent neural networks on electricity demand forecasting [9]. However, problems with neural network training and time efficiency have been cited [10]. We anticipated that by implementing several neural network models and optimizing hyper-parameters on a graphics processing unit (GPU), we would be able to leverage the highly adaptive and expressive properties of neural networks and uncover useful knowledge about the models and petroleum wells.

IV. Methods

We implemented four machine learning architectures: restricted Boltzmann machines (RBMs), fully-connected networks (FCNs), convolutional neural networks (CNNs), and recurrent neural networks (RNNs). Below, we discuss the history of each model as well as the technicalities involved in applying them to our task. We then explain the data provided to us by QRI and the preprocessing steps we took to allow effective training of our models.

A. *Restricted Boltzmann Machines*

A restricted Boltzmann machine (RBM) is a type of generative neural network [11]. Generative models are parameterized families of probability distributions that extrapolate from a finite training set to a distribution over an entire input space. An RBM consists of a visible layer, hidden layer, and connections between the two layers. When the machine is given data, it will propagate the input upwards to the hidden layer, after which the hidden layer will propagate the data back down into the visible layer, yielding a reconstruction of the input [12]. Our main focus on implementing a restricted Boltzmann machine was to determine whether or not there were a discrete number of patterns that a restricted Boltzmann machine would ultimately be able to determine.

RBMs train by adjusting the weights between the visible and hidden layer [13]. We used a sigmoidal activation function since most implementations of restricted Boltzmann machines in the past have functioned well with this particular function and also because we faced issues when attempting to implement other activation functions (e.g. softmax, rectified linear units).

Several steps are involved in updating the weights of an RBM. First, the probability of a given hidden unit firing must be found by summing all of the activations over every visible unit. This probability is used to determine whether or not each hidden unit is activated. To propagate downwards to the visible units, the same process is repeated, except for the visible units. Calculating the error is simple: subtract the outputs from the inputs. The weights are then updated, and this process is repeated over all training examples, either until a certain minimum error is reached or until a maximum number of training steps is exceeded.

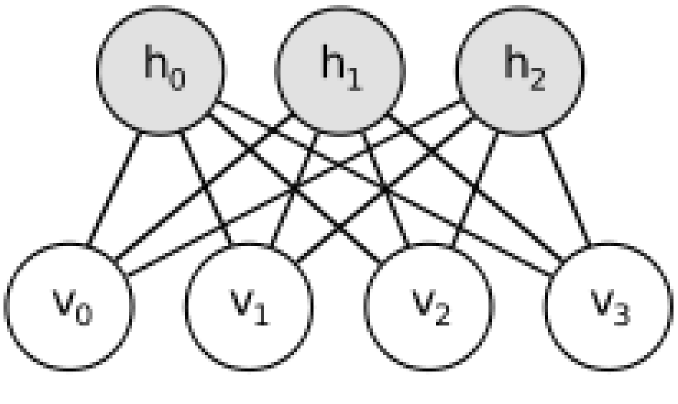

Fig. 1. **A restricted Boltzmann machine.** The visible layer (marked by the clear circles at the bottom of the figure) propogates the input to the hidden layer (marked by the shaded circles at the bottom of the figure) through the array of connections between them after which the data is propogated back down to the visible layer. Here, we observe that the connections between the two layers are undirected. Adapted from [14].

Since RBMs are only designed to probabilistically reconstruct their inputs with the same number of dimensions, adjustments needed to be made for our model. We used 48 months of input data to train our Boltzmann machine. In determining the test set, we substituted the last 12 months of the test set, and replaced them with values of 0.5 (the data set was normalized for a maximum of one and a minimum of zero); then compared the reconstruction with the actual data values for the last 12 months.

B. *Fully-Connected Networks*

Another neural network architecture we constructed was the multilayer perceptron (MLP), which is the simplest form of a deep fully connected network. A figure of this model is shown in Fig. 2. This model is feed-forward, meaning that information flow is always directed towards the output layer. MLPs operate by mapping sets of input data onto sets of output data. It has been shown that they can serve as universal function approximators [15] [16]. MLPs have an input layer whose values are determined by the input samples, at least one hidden layer whose values are derived from previous layers, and an output layer whose values are derived from the last hidden layer. Each neuron in the input and hidden layers has a forward-directed connection to each neuron in the next layer. A non-linear activation function at each neuron introduces non-linearity to the neural network.

MLPs attempt to model the human brain in a very simple fashion. Each neuron in the network receives data from every neuron in the previous layer and each of these inputs is multiplied by an independent weight. The weighted inputs are summed and are then sent through an activation function which is usually designed to scale the output to a fixed range of values. The outputs from each neuron can then be fed into the next layer of the neural network in the same fashion.

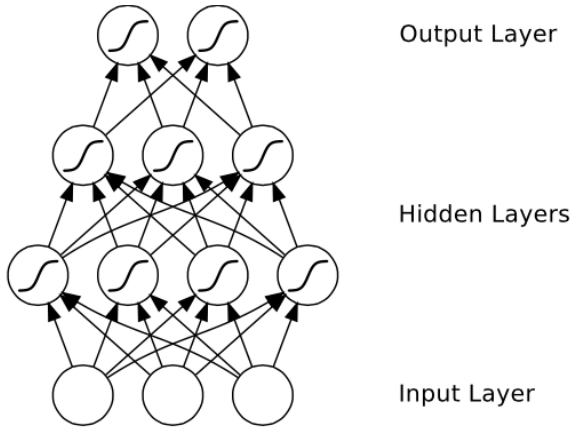

Fig. 2. **A multilayer perceptron.** The S-shaped curves in the hidden and output layer indicate the application of a sigmoid nonlinear activation function. Note that every arrow points away from the input layer, reflecting the feed-forward nature of the MLP. Adapted from [17].

This model can learn relationships between inputs and outputs through training. We implemented a supervised training mechanism in which the network takes a set of known input data and responses (i.e. predictions) and learns patterns in the data. Since the learning capacity of the neural network exists in the values of the weights between neurons, we needed a method to adjust the weights appropriately. The supervised learning algorithm that we ultimately decided to use was back propagation (BP) in conjunction with mini-batch stochastic gradient descent [18]. The way BP learns is by exposure to many examples; we had to provide a learning set that consisted of input examples and the corresponding predictions for each case. We utilized these input-output examples to show the network what type of behavior was expected.

Stochastic gradient descent (SGD) works by taking the derivative of a cost function with respect to each of the weights in the network and adjusting the weights to descend towards the minimum cost [19]. We used the mean absolute error (MAE) of the neural networks' prediction when compared to the actual prediction as our cost function. We used MAE as opposed to commonly used mean squared error (MSE) in order to give proportional weighting to the errors provide resistance to outliers. We also clipped this cost function to a maximum of 6 standard deviations in order to provide further resistance to outliers that might result from incorrectly-taken measurements. We implemented Nesterov momentum to accelerate the learning process [20]; using Nesterov's formula, we were able to use an optimized gradient which significantly reduced training times.

Since the use of a smooth, non-linear activation function is essential for use in a multi-layer network employing gradient-descent learning, we decided to use rectified linear units (ReLU) because it has been shown to train faster and more accurately than those using sigmoidal units. ReLU is faster to compute because it does not require any normalization or exponential computation. Another advantage of this activation function is that it hardly faces the vanishing gradient problem that is common with sigmoidal activation functions. Moreover, it has been demonstrated in other research papers that deep networks can be trained efficiently using ReLU without any pre-training [21].

C. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are variants of MLPs that introduce further constraints to the simple neural network model [22].

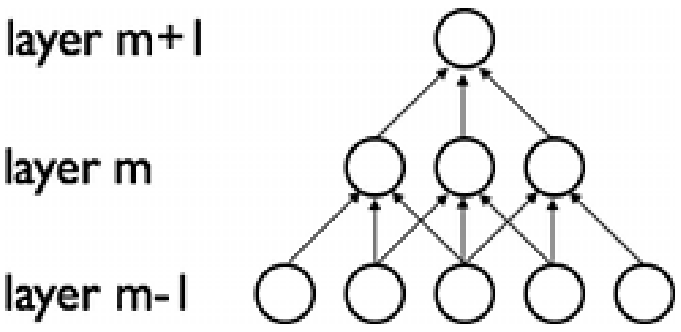

Fig. 3. **Sparse connectivity.** Neurons in a single layer of a sparse network are only connected to the neurons in the next layer that are local to them. Adapted from [23].

In particular, convolutional neural networks are sparse, meaning that they are not fully-connected, but locally-connected (see Fig. 3). In addition, the weightings between the neurons in each layer are constrained to be equal over fixed-size sets of the input (see Fig. 4). The outputs of a convolutional layer can be determined computationally by convolving the neural input with filters. Several filters can be used on a single layer to form feature maps, which can ultimately be fed into a fully-connected network where learned features are used to generate outputs.

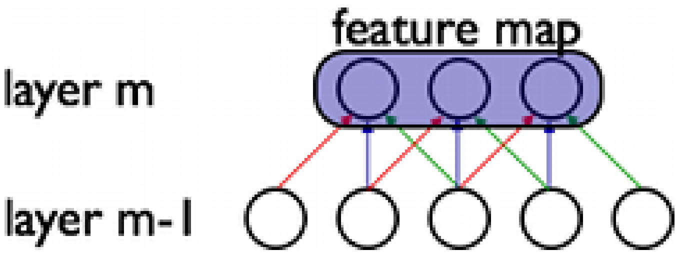

Fig. 4. **CNN architecture.** In addition to sparse connectivity, CNNs contained shared weights as indicated by the colored connections. The outputs form feature maps, which can be used as inputs to a fully-connected network. Adapted from [23].

D. Recurrent Neural Networks

Recurrent neural networks (RNNs) exploit characteristics of a feedback network, allowing the model to construct a sequential representation of input data. However, simple RNNs have been notoriously difficult to train, due to their iterative nature and the highly volatile relationship between the parameters and the hidden states [10]. This introduces the phenomenon of ”exploding” and ”vanishing” gradients. In the case of exploding gradients, error-signals exhibit exponential growth as they are back-propagated in time. Uncurbed growth causes long-term error signals to overwhelm and hide short-term ones. In the case of vanishing gradients, error-signals exhibit exponential decay during back-propagation. The decay causes short-term error signals to overwhelm long-term signals [24]. To counteract these issues, several architectures and training methods have been developed, including the Hessian-Free (HF) optimization technique, the Gated Recurrent Unit (GRU) architecture, and the Long Short-Term Memory (LSTM) architecture. We illustrate RNN models used in this research below.

1) Simple Recurrent Neural Networks: Here, we define the simple RNN architecture [25]. The network is given a sequence of inputs. Then, the network computes a sequence of hidden states and a sequence of predictions for each step in time. The sequences are computed by iterating through a series of equations that updates the model's weight matrices, hidden layers, and the output units. The RNN utilizes predefined vector-valued functions that contain a computable Jacobian and are typically non-linear and applied coordinate-wise [24].

The RNN is a feedback model that contains what are known as feedback loops. Information does not simply

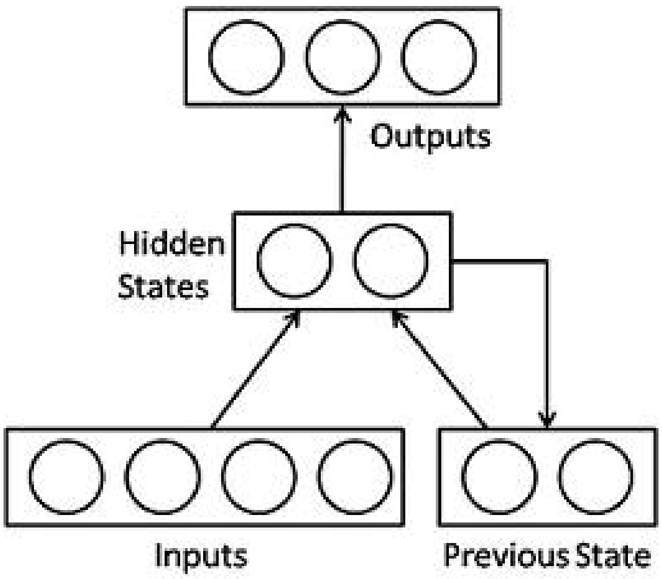

Fig. 5. **Simple RNN.** In this architecture, previous states of the system are stored and used to generate future hidden states. Adapted from [26].

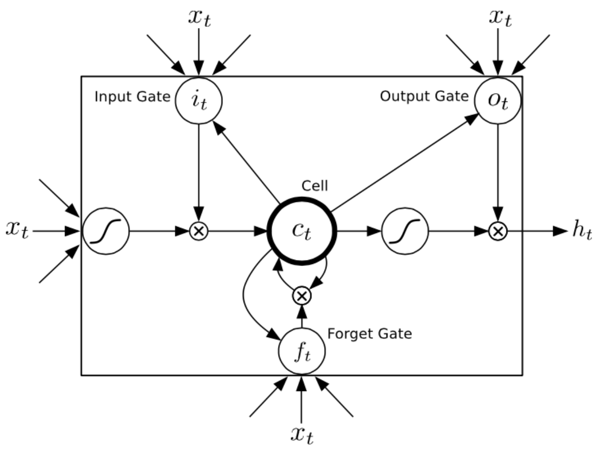

Fig. 6. **Long Short-Term Memory Cell.** Memory cells contain input, output, and forget gates [represented by i, o, and f respectively] that enable the network to store and access information over long time periods. Adapted from [28].

travel from input to output as it does in a feed-forward model. However, a purely feed-forward model ignores certain temporal structures of time series data [9]. Instead, in an RNN, the output of the model is provided as input into the nodes, along with the previous data. This helps us construct an explicit representation of our data.

2) Simple Deep Recurrent Neural Networks: The simple deep RNN is similar to the simple RNN. However, in a simple deep RNN, the output of multiple timesteps is fed back to the input. As with other deeper networks, simple deep RNNs have increased numbers of levels or layers. These layers are able to perform additional non-linear operations on the input data to learn new information and handle more complex functions [27].

3) Long Short-Term Memory: Long Short-Term Memory (LSTM) is a modified RNN architecture developed by Hochreiter and Schmidhuber in 1997 [28]. LSTM addressed the vanishing gradient problem and allows the storage information of an extended time period.

LSTM consists of a standard RNN along with additional self-connected memory cells and three multiplicative units – the input, output, and forget gates – that equivalently write, read, and reset information within the model's cells. [17] In an LSTM network, memory cells displayed in Fig. 6 [29] replace summation units in the hidden layer. These memory cells contain the three multiplicative gates, which enable the LSTM memory cells to store and access information over long time periods. The gates control the amount of information fed into the memory cell at a given time step.

By refraining from traditional RNN methods of overwriting new content at each timestep, the LSTM can effectively decipher important features and carry that information over a long distance [31].

4) Gated Recurrent Unit: The Gated Recurrent Unit (GRU) was introduced recently in 2014 by Cho et al [30]. GRU was found to outperform the LSTM on a variety of tasks by Chung et al [31]. The GRU contains gating units that modulate the flow of information within the unit similar to the LSTM. However, the GRU forgoes the separate memory cells found in LSTM.

The GRU contains two gates, the update gate and reset gate, as displayed in Figure 7. The update gate determines how much the unit updates its activation. This is similar to the LSTM, which takes the linear sum between the existing state and a newly computed state. The second gate within the GRU, the reset gate, acts to forget the previously computed state. [31]

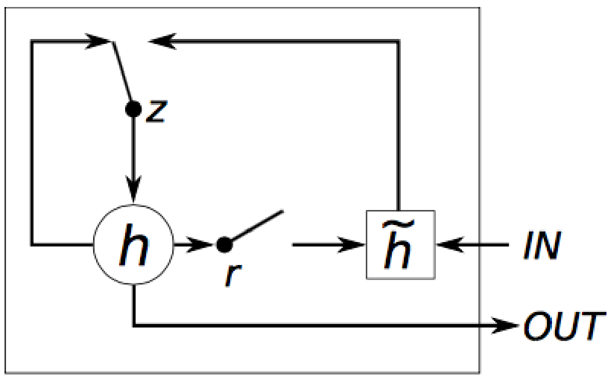

Fig. 7. **Gated Recurrent Unit.** GRU contains reset and update gates [represented by r and z respectively] that enable the network to store and access information over long time periods. Adapted from [30].

5) MUT-123: MUT-1, MUT-2, and MUT-3 are three different recurrent architectures developed by Jozefowicz et al [32]. The experiment featured aggressive optimization searches on random architectures similar to the LSTM and GRU and their hyper-parameter settings. MUT-1, MUT-2, and MUT-3 were the three best architectures on arithmetic, XML modeling, and word-level language modeling on the Penn TreeBank. The architectures differ in their mathematical implementation of their update gates.

E. Dropout

Dropout is a powerful technique that can be applied to neural networks to prevent overfitting. Overfitting is the over-training of limited training data, which results in learning complicated relationships of the sampling noise within the small set of data. Thus, overfitting occurs when a network over learns the data it has already seen, but performs poorly on new data.

Dropout works by randomly dropping out units – hidden and visible – in a neural network [33]. This is equivalent to temporarily removing those units from the networking at that certain timestep. The resulting neural network is called a thinned network. This process is pictured in Figure 8.

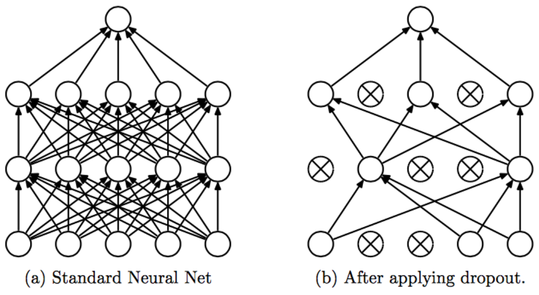

Fig. 8. **Dropout Neural Net Model.** *Left:* A standard neural network with 2 hidden layers. *Right:* A thinned network produced by applying dropout to the network on the left. Crossed units have been dropped. [33]

F. Preprocessing

We trained, validated and evaluated our models on petroleum well data provided to us by QRI. The original data was stored in a spreadsheet containing information on seven drilling sites with the date of the measurement taken, the field name, formation type, well name, oil production, water production, and gas production. Each of the seven sites contained several wells, with each well identified by its well name. The well name is generated using the first two letters of the field name, the first two letters of the formation type, and an index. Every well contains data for oil, water, and gas production for every month it has been drilled, as indicated by the date. For the purposes of this research, we considered only the oil production. However, the given dataset contained problems: each site varied in the number of wells, each well varied in the amount of data associated with it, and data points were often zeroed out (i.e. missing) or very large, probably incorrectly recorded.

We preprocessed the data from the spreadsheet to mitigate these apparent problems. We began by removing the missing, or zeroed-out, data points. Because these data points represent times when the well was turned off or when data was not recorded, they incorrectly represent oil production data for the time series and indicate times when the well would not likely change its state over time. For the time series of oil wells, these missing or zeroed points become irrelevant to training and prediction. Thus, we pushed together remaining non-zero points to create a time series without false zeros that could interfere with the training process.

Additionally, we took steps to accommodate neural network models that take only fixed-size inputs. We did this by preprocessing the data to construct time series of same-sized inputs. We use a sliding window method pictured in Fig. 9. We took each well and split it into chunks of 60 data points (months) – 48 months input, 12 months expected output. We specified a step size of 6 months, moved the window past an additional six points, and sampled another chunk of 60 data points. We continued this process until we ran out of data points for each well. Note that performing this preprocessing step is only permissible under the assumption that oil production rates are time-invariant. This is a reasonable assumption to make; according to QRI, drilling techniques have not changed significantly over the past several decades.

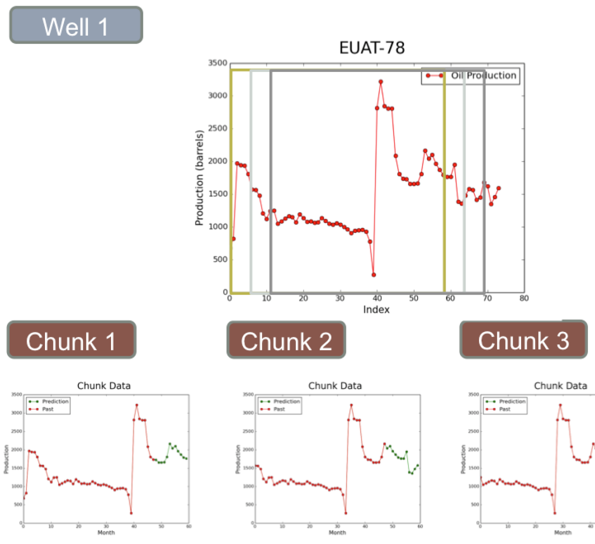

Fig. 9. **Dividing data within each well to time series of equal-sized chunks.** Every chunk contains an input of 48 months and a corresponding expected output of 12 months, totaling 60 months of data per chunk.

However, we noticed that the chunks of data still revealed large fluctuations in oil production from well to well. To maintain the same scale for our data, we normalized each chunk with respect to the 48 months of input to ease the training process.

Finally, we divided our data into training, validation, and testing sets based on the well index. We created a training set consisting of 5041 chunks, a validation set of 1026 chunks, and a test set of 1144 chunks. We used the training set to tune the hyper-parameters of our models. The validation set was used to determine when to stop training. Our final results were evaluated on our testing set.

V. Experiments

Our project task involved optimizing parameters within each of the four neural networks implemented: RBM, fully-connected network, CNN, and RNN. The specific parameters are listed within Table I.

TABLE I
Parameters to Optimize

Parameters	**Types**
Activation Functions	tanh, **ReLU**, sigmoid, hard-sigmoid
Dropout	0.1 - 0.9, **0.5**
Hidden Layers	**0 - 4**
Hidden Layer Nodes	**5 - 1000**
Loss Functions	MSE, **MAE**, RMSE, RMAE, RMSLE, RMALE
Optimization	**SGD**, Adagrad, Adadelta, RMSProp, Adam
Momentum	None, regular, **Nesterov**
Learning Rate	0.0001, **0.001**, 0.005+
Learning Rate Decay	0 - 1e-3, **1e-6**
Regularization	**None**, L1, L2
Filters (for CNNs)	5 - 500, **100**
Kernels/filter (for CNNs)	2 - 24, **13**

A. Restricted Boltzmann Machines

During the training process of the restricted Boltzmann machine, there were several problems. When first implemented, the restricted Boltzmann machine would only output the same result, with only minor fluctuations at seemingly arbitrary times. Two tests were used to determine whether or not the Boltzmann machine worked properly. First was a training set consisting of periodic step functions to determine if the RBM was even capable of learning simplistic, binary features, which was the case. Second was a sinusoidal test set to determine if the RBM could learn non-binary data. The results seem to indicate that the RBM was able to reconstruct sinusoidal waves with small periods, but not those with larger periods.

We hypothesize that the failure of restricted Boltzmann machines on oil well data is due to their inability to find any consistent patterns within the dataset that was given. Additionally, the activation function that was used, the sigmoidal function, is usually used with binary data, not time series data that was provided to us.

Because of the issues we faced with RBMs, we do not compare them to the other models in later sections.

B. Fully-Connected Networks

During the design and training process of our multilayer perceptron (MLP), we used a grid search over three hyper-parameters in an attempt to locate an optimal neural network architecture. We conducted more than 7,000 experiments on the Odyssey computer cluster at Harvard. We created scripts in which we specified the amount of memory, time, cores, and number of GPUs in order to train our models. We conducted experiments with different batch sizes on a fully connected network with 1 hidden layer in order to estimate the effect that batch size has on the overall calculated error. After seeing the results, we concluded that lower batch sizes produced better results on the testing set (see Fig. 10 below).

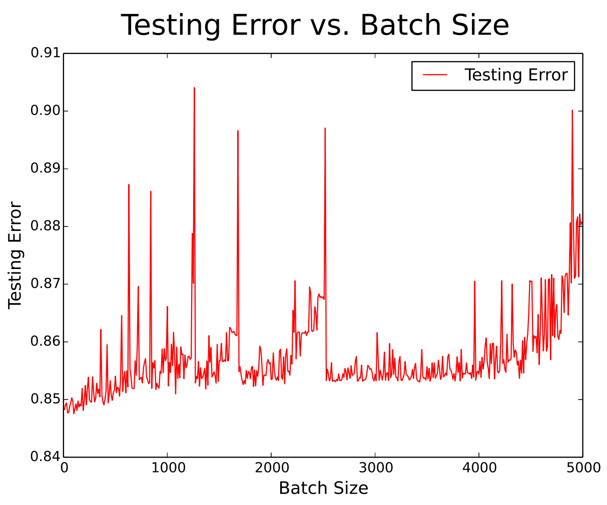

Fig. 10. **Batch size tweaking.** We see that lower batch sizes tend to produce lower error on the test set using an MLP with a single hidden layer.

We then continued training the model with different number of layers ranging from 1 to 4 as well as with different number of neurons from 10 to 500 (which we increased in steps of 10), using a fixed batch size of 10. After analyzing the results, we concluded that increasing the number of hidden layers and hidden units had little effect on testing set error (see Fig. 11).

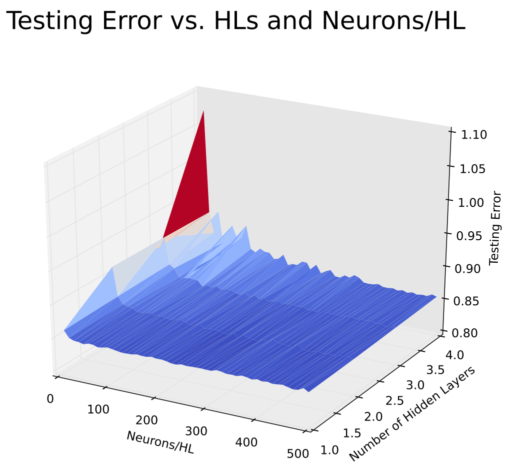

Fig. 11. **Network size tweaking.** On one axis we have the number of hidden layers in the network, on the other we have the number of neurons per hidden layer. On the third axis, we plot the cost as a function of these two independent variables. We see that one hidden layer is sufficient to minimize the error; in fact, we observe that adding more hidden layers increases the cost when the number of neurons per hidden layer is small. We conclude that one hidden layer with a small number of neurons is optimal for an FCN to predict future petroleum output.

To compare FCNs to the other types of networks, we selected two specific models which performed well. The first was a network with no hidden layers. The second was a network with one hidden layer consisting of 100 neurons, using ReLU as the activation function.

C. Convolutional Neural Networks

We tested CNNs with a few different parameters but did not perform in-depth hyper-parameter optimization on these networks. This due to the fact that these networks took a very long time to train and consistently produced results that were either worse than or comparable to FCNs. Our best model consisted of a single convolutional layer with 100 filters with 13 kernels each, fed into a fully connected layer for the output. On the CPU, these networks were slow but showed significant speedups when tested on the GPU cluster.

D. Recurrent Neural Networks

We tested RNNs on several architectures: simple RNN, simple deep RNN, GRU, LSTM, and MUT-123. We initially ran tests on the CPU, but later ran models on the GPU. We did not optimize all hyperparameters on these architectures due to time restraints and a large amount of parameters to optimize on several different architectures. Some hyperparameters we optimized included the validation frequency, epochs, and activation function on our first simple RNN model implemented in Python and Theano. We later used the Keras library and ran several different tests optimizing the number of hidden nodes within hidden layers, activation function, and architecture type. We found the best model to be a GRU with three layers using a rectified linear activation function.

VI. Results and Discussion

In Table II we record the training time taken and testing error on each of the 16 rudimentary models we formulated. Notice that all of the RNN models perform better than the FCN and CNN models. In terms of time taken to train however, the FCN 0-Hidden Layer model was the best, as expected.

TABLE II
Model Performance

Model Name	Training Time	Testing Error
GRU ReLU	148.203301 s	**0.832880**
MUT2	197.463451 s	0.835410
MUT1	267.760790 s	0.836049
GRU	284.616835 s	0.836572
LSTM	329.868082 s	0.836652
MUT2 ReLU	109.027726 s	0.837153
MUT3	137.224448 s	0.837395
Simple RNN ReLU	51.507298 s	0.837879
MUT1 ReLU	167.753578 s	0.838955
Simple Deep RNN	106.117913 s	0.838986
Simple Deep RNN ReLU	401.925844 s	0.839281
MUT3 ReLU	45.747421 s	0.839637
Simple RNN	95.576466 s	0.839948
FCN 0-Hidden Layer	**5.019959 s**	0.845515
CNN	142.188696 s	0.846481
FCN 1-Hidden Layer ReLU	9.460658 s	0.848628

All of the models exhibited similar performance and the predictions yielded by each of the models appear to be very similar (see Fig. 12 and Fig. 13 for examples). The models were good at predicting downward trends but usually failed to predict upward trends well.

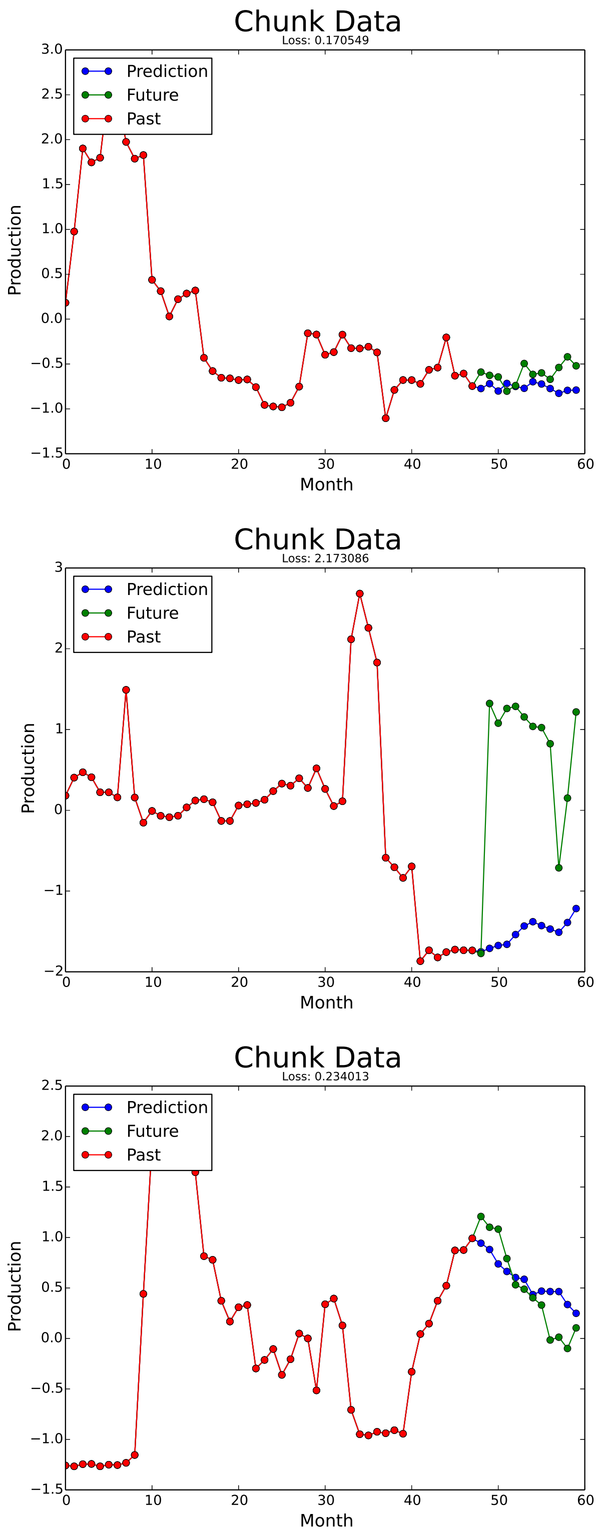

Fig. 12. **Predictions generated by FCN with no hidden layers.** These graphs depict the predictions of the fastest model (0-layer FCN) on three different chunks from the testing set. The neural network predicts downward slopes and flat lines well, but fails to detect sudden upward jumps.

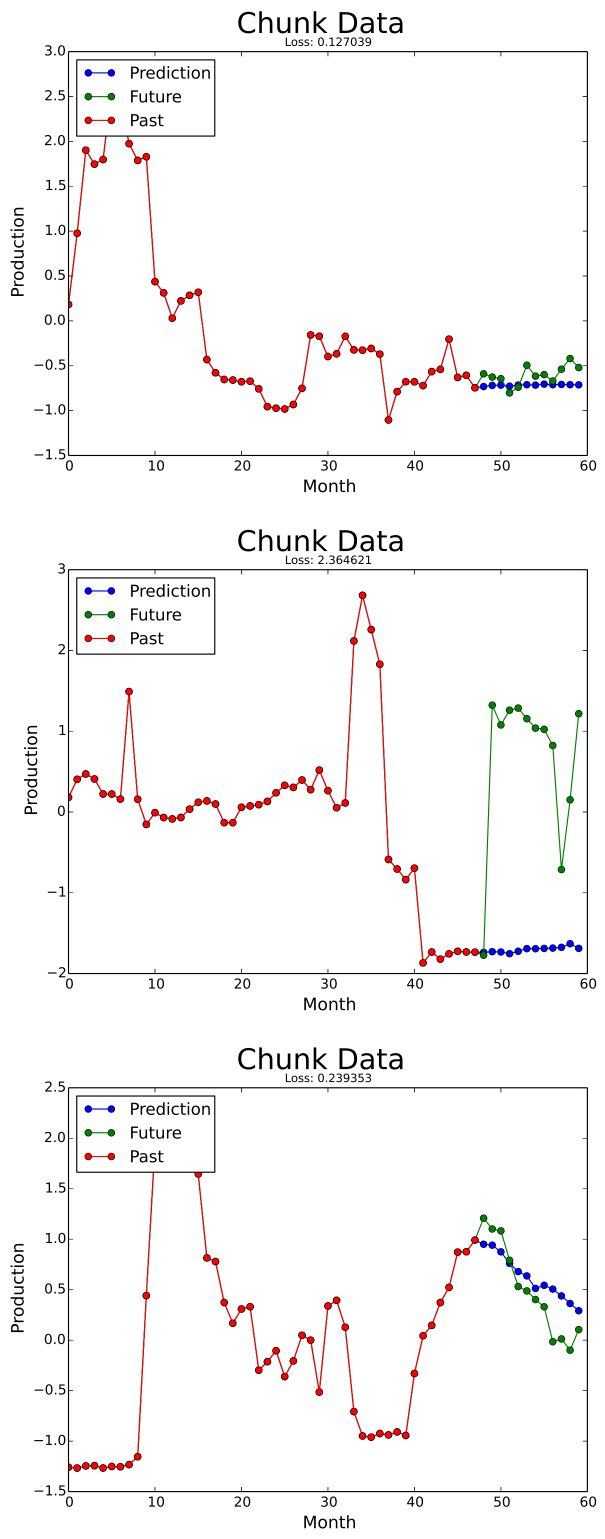

Fig. 13. **Predictions generated by GRU with ReLU.** These graphs depict the predictions of the best model (GRU with ReLU) on three different chunks from the testing set. The neural network predicts downward slopes and flat lines well, but fails to detect sudden upward jumps. Note the similarity of the predictions to those of the FCN in Fig. 12.

VII. Conclusion

All of the deep neural network models we tested were able to predict twelve months of future oil production data reasonably well. Overall we see that despite being difficult to train, with many hyper-parameters to tune, deep learning is promising for many applications in the petroleum industry. We also find that even the simplest of models are sufficient to reasonably predict the oil well productions. These models' robust nature and ability to learn patterns in unlabeled data will make them a useful tool for engineers in this sector.

VIII. Future Work

Many steps could be taken to further improve the performance of these neural networks. L1 and L2 regularization could help minimize overall error in the network [34]. Bayesian optimization techniques have been shown to yield optimal neural networks architectures [35]. Hierarchical temporal memory (HTM) mimics the human neocortex very closely, giving it potential to more accurately predict oil production rates than classical neural network architectures [36]. Additionally, we could incorporate water and gas data to supply more information to the neural network. Exposing a network to this two-dimensional data could improve predictions significantly. Since we were using a relatively small dataset, we needed to train our neural networks on every oil well we were given. But with a larger dataset, it might be viable to train multiple neural networks with each one tailored towards a specific species of oil well. Lastly, if it were known when upgrades were being undergone in a well, predictions would become much easier to make.

IX. Acknowledgments

The dataset used in this paper was provided by Quantum Reservoir Impact (QRI). The computations in this paper were run on the Odyssey cluster supported by the FAS Division of Science, Research Computing Group at Harvard University.

References

[1] D. O. Hebb. *The Organization of Behavior: A Neuropsychological Approach*. John Wiley & Sons, 1949.
[2] Bengio, Yoshua, Aaron Courville, and Pierre Vincent. "Representation learning: A review and new perspectives." *Pattern Analysis and Machine Intelligence, IEEE Transactions* on 35, no. 8 (2013): 1798-1828.
[3] Arel, Itamar, Derek C. Rose, and Thomas P. Karnowski. "Deep machine learning-a new frontier in artificial intelligence research [research frontier]." *Computational Intelligence Magazine, IEEE* 5, no. 4 (2010): 13-18.a
[4] Rosenblatt, Frank. "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review* 65, no. 6 (1958): 386.
[5] Claudiu Ciresan, Dan, Ueli Meier, Luca Maria Gambardella, and Juergen Schmidhuber. "Deep big simple neural nets excel on handwritten digit recognition." *arXiv preprint arXiv:1003.0358* (2010).
[6] Cirean, Dan, Ueli Meier, Jonathan Masci, and Jrgen Schmidhuber. "A committee of neural networks for traffic sign classification." In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pp. 1918-1921. IEEE, 2011.
[7] Prasad, Sharat C., and Piyush Prasad. "Deep Recurrent Neural Networks for Time Series Prediction." *arXiv preprint arXiv:1407.5949* (2014).
[8] Dalto, Mladen. "Deep neural networks for time series prediction with applications in ultra-short-term wind forecasting." http://centar.open.hr/_download/repository/KDI-Djalto.pdf
[9] J. W. Taylor. "Short-term electricity demand forecasting using double seasonal exponential smoothing." *Journal of the Operational Research Society* 54, no. 8 (2003): 799-805.

[10] Y. Bengio, P. Simard, and P. Frasconi. "Learning long-term dependencies with gradient descent is difficult." *Neural Networks, IEEE Transactions* on 5, no. 2 (1994): 157-166.

[11] Salakhutdinov, Ruslan, and Geoffrey E. Hinton. "Deep boltzmann machines." *International Conference on Artificial Intelligence and Statistics*. 2009.

[12] Kuremoto, Takashi, Shinsuke Kimura, Kunikazu Kobayashi, and Masanao Obayashi. "Time series forecasting using a deep belief network with restricted Boltzmann machines." *Neurocomputing* 137 (2014): 47-56.

[13] Hinton, Geoffrey. "A practical guide to training restricted Boltzmann machines." Momentum 9, no. 1 (2010): 926.

[14] Restricted Boltzmann Machines - Deep Learning 0.1 documentation http://deeplearning.net/tutorial/rbm.html

[15] Cybenko, George. "Approximation by superpositions of a sigmoidal function." *Mathematics of control, signals and systems 2, no. 4* (1989): 303-314.

[16] Hornik, Kurt. "Approximation capabilities of multilayer feedforward networks." *Neural networks* 4, no. 2 (1991): 251-257.

[17] Graves, Alex. *Supervised sequence labelling with recurrent neural networks.* Vol. 385. Heidelberg: Springer, 2012.

[18] Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors." *Cognitive modeling* 5 (1988): 3.

[19] Robbins, Herbert, and David Siegmund. "A convergence theorem for non negative almost supermartingales and some applications." *In Herbert Robbins Selected Papers*, pp. 111-135. Springer New York, 1985.

[20] Sutskever, Ilya, James Martens, George Dahl, and Geoffrey Hinton. "On the importance of initialization and momentum in deep learning." *In Proceedings of the 30th international conference on machine learning (ICML-13)*, pp. 1139-1147. 2013.

[21] A. Dalyac. Some Intuition about Activation Functions in Feed-Foward Networks. http://www.academia.edu/7826776/Mathematical_Intuition_for_Performance_of_Rectified_Linear_Unit_in_Deep_Neural_Networks

[22] Y. LeCun and L. Bottou. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 1998.

[23] Convolutional Neural Networks (LeNet) - Deep Learning 0.1 documentation http://deeplearning.net/tutorial/lenet.html

[24] Martens, James, and Ilya Sutskever. "Learning recurrent neural networks with hessian-free optimization." *In Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 1033-1040. 2011.

[25] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning Representations by Back-Propagating Errors. *Nature*, 323(6088): 533536, 1986.

[26] University of Nebraska-Lincoln CSE Wiki: Recurrent neural networks http://cse-wiki.unl.edu/wiki/index.php/Recurrent_neural_networks

[27] Bengio, Yoshua. "Learning deep architectures for AI." *Foundations and trends in Machine Learning* 2, no. 1 (2009): 1-127.

[28] Hochreiter, Sepp, and Jrgen Schmidhuber. "Long short-term memory." *Neural computation* 9, no. 8 (1997): 1735-1780.

[29] Graves, Alex. "Generating sequences with recurrent neural networks." *arXiv preprint arXiv:1308.0850* (2013).

[30] Cho, Kyunghyun, van Merrienboer, Bart, Gulcehre, Caglar, Bougares, Fethi, Schwenk, Holger, and Bengio, Yoshua. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[31] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[32] Jozefowicz, Rafal, Wojciech Zaremba, and Ilya Sutskever. "An Empirical Exploration of Recurrent Network Architectures." *In Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pp. 2342-2350. 2015.

[33] Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: A simple way to prevent neural networks from overfitting." *The Journal of Machine Learning Research* 15, no. 1 (2014): 1929-1958.

[34] Wang, Li, Michael D. Gordon, and Ji Zhu. "Regularized least absolute deviations regression and an efficient algorithm for parameter tuning." In *Data Mining, 2006. ICDM'06. Sixth International Conference on*, pp. 690-700. IEEE, 2006.

[35] Snoek, Jasper, Hugo Larochelle, and Ryan P. Adams. "Practical Bayesian optimization of machine learning algorithms." In *Advances in neural information processing systems*, pp. 2951-2959. 2012.

[36] Hawkins, Jeff, and Dileep George. *Hierarchical temporal memory: Concepts, theory and terminology. Technical report*, Numenta, 2006.