Estimating normal moveout velocity using the recurrent neural network

Reetam Biswas[1], Anthony Vassiliou[2], Rodney Stromberg[2], and Mrinal K. Sen[1]

ABSTRACT

Machine learning (ML) has recently gained immense popularity because of its successful application in complex problems. It develops an abstract relation between the input and output. We have evaluated the application of ML to the most basic seismic processing of normal moveout (NMO) correction. The arrival times of reflection events in a common midpoint (CMP) gather follow a hyperbolic trajectory; thus, they require a correction term to flatten the CMP gather before stacking. This correction term depends on an rms velocity, also referred to as the NMO velocity. In general, NMO velocity is estimated using the semblance measures and picking the peaks in the velocity panel. This process requires a lot of human intervention and computation time. We have developed a novel method using one of the tools based on an ML- approach and applied to the NMO velocity estimation problem. We use the recurrent neural network (RNN) to estimate the NMO velocity directly from the seismic data. The input to the network is a seismic gather and corresponding precalculated NMO velocity (as prelabeled data set) to flatten the gather. We first train the network to develop a relationship between the input gathers (before NMO correction) and the corresponding NMO velocities for a few CMPs as a supervised learning process. Adam optimization algorithm is used to train the RNN. The output from the network is then compared against the correct NMO velocity. The error between the two velocities is then used to update the weight of the neurons and to minimize the mean-squared error between the two velocities. After the network is trained, it can be used to calculate the NMO velocity for the rest of the seismic gathers. We evaluate our method on a noisy data set from Poland. We used only 10% of the CMPs to train the network, and then we used the trained network to predict NMO velocity for the remaining CMP locations. The stack section obtained by using RNN-generated NMO velocities is nearly identical to that obtained by the conventional semblance method.

Introduction

Machine learning (ML) was developed several decades ago, but its use in seismic processing and interpretation has been limited, mainly due to the lack of powerful computational resources. Conventional ML algorithms such as the artificial neural network (ANN) have been applied to multiple areas of science and engineering. There have been several successful attempts in the geophysics community as well. For example, first-break picking using ANN (e.g., Murat and Rudman, 1992; McCormack et al., 1993; Wang and Teng, 1997), obtaining subsurface elastic attributes (Röth and Tarantola, 1994; Calderón-Maciás et al., 2000; Moya and Irikura, 2010), studying reservoir characterization using seismic reflection data (An and Moon, 1993), velocity picking from velocity scans for velocity analysis (Schmidt and Hadsell, 1992; Fish and Kusuma, 1994; Calderón-Maciás et al., 1998), and study of S-wave splitting (Dai and MacBeth, 1994).

With the development of low-cost, powerful computers and graphics cards, there has been a steep increase in the use of various new and sophisticated ML algorithms. They are now widely accepted in almost every field of research, such as handwriting recognition, speech recognition, and signal detection (e.g., Freeman and Skapura, 1991; Cichocki and Unbehauen, 1993). In particular, the convolution neural network (CNN) (LeCun et al., 1989) has found extraordinary success in the field of computer vision. CNN outperformed various conventional methods for image classification (Krizhevsky et al., 2012), object detection (Girshick et al., 2014), and image segmentation (Ronneberger et al., 2015) by a considerable margin. Some studies report that some deep CNN-based networks have better classification success rates than humans (Russakovsky et al., 2015; He et al., 2016). There have been some applications of CNN in the geophysical field as well, for example, salt body classification (Di et al., 2018; Shi et al., 2018), fault body interpretation from seismic images (Wu et al., 2018), and first-arrival picking of microseismic events (Wu et al., 2019). Ma et al. (2018) use CNN for velocity

[1]The University of Texas at Austin, Institute for Geophysics and Department of Geological Sciences, John A. and Katherine G. Jackson School of Geosciences, Austin, Texas 78713-8924, USA. E-mail: reetam@utexas.edu (corresponding author); mrinal@utexas.edu.

[2]GeoEnergy Inc., 3100 Wilcrest Dr. #220, Houston, Texas 77042, USA. E-mail: anthony@geoenergycorp.com; rod@geoenergycorp.com.

Manuscript received by the Editor 19 December 2018; revised manuscript received 19 May 2019; published ahead of production 05 August 2019; published online 20 September 2019. This paper appears in *Interpretation*, Vol. 7, No. 4 (November 2019); p. T819–T827, 11 FIGS.

http://dx.doi.org/10.1190/INT-2018-0243.1.

picking for normal moveout (NMO) correction; however, they modify the problem into a classification problem, in which the class represents a multiplier of a constant velocity. Most CNN applications are based on a classification problem, and geophysical problems involve mainly parameter estimation-based or regression problems. However, Biswas et al. (2019) use CNN to solve an inverse problem, in which the training is guided by the physics of the forward problem.

In this paper, we consider a regression problem, in which we aim to estimate the NMO velocity directly from the seismic data. We want the network to learn a mapping from the seismic data domain to velocity such that it can flatten out the gathers while performing NMO correction. One of the vital points in the problem is that the seismic data set represents a time series, but along with it, offset information is also necessary; i.e., the spatial and temporal information are essential for the NMO velocity estimation. Another robust ML algorithm is the recurrent neural network (RNN), which works on a time series type of data. RNN has a straightforward network architecture, and it is very similar to a multilayer feedforward neural network (FNN).

Due to its flexibility, the FNN can be generalized to a variety of problems. It has been applied extensively to solve a pattern-recognition problem. In an FNN, a signal travels only in one direction from input to output, and the output of one layer does not affect that same layer; it is just a relation from input to output. Calderón-Maciás et al. (1998) attempt to estimate NMO velocity using an FNN, and Schmidt and Hadsell (1992) attempt to estimate NMO velocity using the time delay neural network. However, unlike FNN, a recurrent network provides a feedback loop in the network, and a signal can travel in both directions. In RNN, the output of the current state is dependent on the output of the previous states as well; thus, it develops a network memory. It tries to find a correlation between events separated by many moments, long-term dependencies: They are a way to share weights over time. Thus, defining a deep RNN in this manner can make the network very powerful, but it can get very complicated. This feedback loop

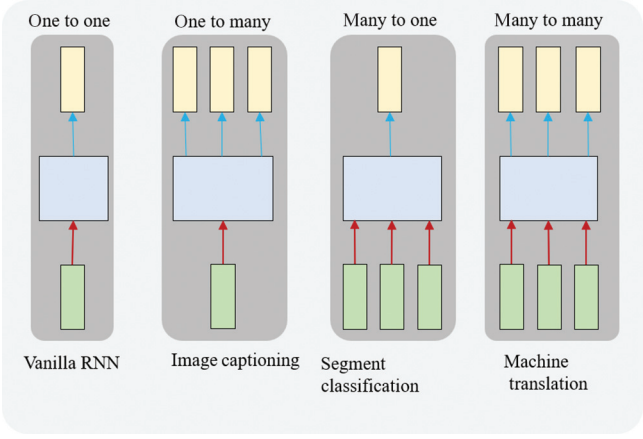

Figure 1. Different flavors of RNN.

makes RNN dynamic; the state of the RNN changes continuously until a balance is reached. Again, with new input, the network adapts to find a new equilibrium. Training the network consists of a set of training input data and corresponding correct output. The weights of the neurons are initially randomly assigned, but they are updated while training, by backpropagating through time. The network learns the abstract relationship between the input-output pattern and minimizes the error between the predicted output and correct output. After the network is trained, it is applied to a new data set for which the output is not known.

Due to the flexibility of RNN, it has been popular in a wide range of problems. It can work on any size and number of input and output pairs. There are many different flavors of RNN, as shown in Figure 1. The first kind is the vanilla RNN, which is the simplest one and maps the relationship between one input-output pair. In the second kind, one input corresponds to many outputs, and it is used in problems such as image captioning (Mao et al., 2014). The third type involves many inputs to one output, used in high-level segment classification, like guessing the emotion from a sentence (Lee and Tashev, 2015). The last kind is many-to-many classifications, used for machine translation, like translating from one language to another (Bahdanau et al., 2014).

According to the dynamical systems, RNN has been classified into two main groups (Lukoševičius and Jaeger, 2009). The first category of RNN is a stochastic system having symmetric connections with the goal to minimize the energy of the system. Some examples include Hopfield networks (Hopfield, 1982, 2007), Boltzmann machines (Hinton, 2012), and deep belief networks (Hinton and Salakhutdinov, 2006). The training for these networks is mostly done in an unsupervised fashion using statistical physics. Recently, Vamaraju and Sen (2018) apply Hopfield neural network for migration, Phan and Sen (2018) apply the Hopfield neural network for prestack seismic inversion, and Huang (1997) uses the Hopfield neural network for seismic horizon picking. In the second category of RNN, the dynamics is updated in a deterministic way and has directed connections. It transforms an input time series to an output time series using several nonlinear filters. The weights are updated in a supervised way. One other example includes long short-term memory (Hochreiter and Schmidhuber, 1997). Recently, Alfarraj and AlRegib (2018) and Alfarraj et al. (2018) use RNN to estimate petrophysical property from seismic data. Richardson (2018) uses a deep RNN to solve a seismic full-waveform inversion, and Moseley et al. (2018) use WaveNet, which is a mixed deep network of a convolutional and an RNN, to an approximate simulation of seismic waves.

The proposed approach in this paper is of the second category and can handle a nonlinear time series quite well. These networks are outstanding in generalizing the relationship in a dynamic system (Funahashi and Nakamura, 1993). In the following sections, we first describe the basic structure of the network and then

demonstrate its working on a real land data set from Poland.

Theory

This section introduces various basic concepts about the RNN and describes the way data are input in the network to calculate the NMO velocity.

Recurrent neural network

The architecture of an RNN is similar to an FNN, except that the RNN has a feedback loop, a connection pointing backward. An FNN is just a functional mapping from input to output. However, RNN is a dynamical system and can develop a self-sustained dynamics of temporal activation along the path within itself, even without any input because of the feedback loop. Figure 2 shows an example of a simple RNN. Figure 2a shows a recursive model of the RNN, and Figure 2b shows the corresponding network structure when it is unrolled through time. At any time step t, every neuron in an RNN has input vectors as $\mathbf{x}_t$ and previous time step's output vector $\mathbf{y}_{t-1}$ as shown in Figure 2b. RNN helps us to examine the functional map $f:\mathbf{x}\rightarrow\mathbf{y}$ between the given input sequence $\mathbf{x}=\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_t$ with the corresponding output sequence vector $\mathbf{y}=\mathbf{y}_1,\mathbf{y}_2,\ldots,\mathbf{y}_t$. Two weights correspond to every neuron in RNN, one for the input signal $\mathbf{x}$ represented as $\mathbf{W}_{xy}$ and the other for the previous time step's output $\mathbf{y}$ represented as $\mathbf{W}_{yy}$. We can represent the network using

$$\mathbf{y}_t=\phi(\mathbf{W}_{xy}^T\cdot\mathbf{x}_t+\mathbf{W}_{yy}^T\cdot\mathbf{y}_{t-1}+\mathbf{b}), \tag{1}$$

where ϕ represents the activation function and $\mathbf{b}$ is the bias vector. In ML, there are several choices of activation available, e.g., rectified linear unit (ReLU), tanh, and sigmoid. In our application, we have used the ReLU activation function defined by $(f(x)=\max(0,x))$ because it provides better capability than sigmoid (Krizhevsky et al., 2012). In an RNN, multiple hidden layers can be included to represent a deep neural network. Multiple layers in the RNN help to capture the different degree of features and predict a better output.

During the training of the network, in a single iteration, we update the weight for not just a single sequence but multiple sequences of data known as a minibatch. Equation 1 can be modified to compute the output of the whole minibatch in a single shot by representing it as

$$\begin{aligned}\mathbf{Y}_t&=\phi(\mathbf{X}_t\cdot\mathbf{W}_{xy}+\mathbf{Y}_t\cdot\mathbf{W}_{yy}+\mathbf{b}\\&=\phi([\mathbf{X}_t\quad\mathbf{Y}_{t-1}]\cdot\mathbf{W}+\mathbf{b}),\end{aligned} \tag{2}$$

with $\mathbf{W}=\begin{bmatrix}\mathbf{W}_{xy}\\\mathbf{W}_{yy}\end{bmatrix}$. If the minibatch has m instances of different sequences containing n_n neurons and n_i input vector size; then matrix $\mathbf{Y}_t$ has a dimension of $m\times n_n$, $\mathbf{X}_t$ has $m\times n_i$, $\mathbf{W}_{xy}$ has $n_i\times n_n$, $\mathbf{W}_{yy}$ has $n_n\times n_n$, and, finally, the bias vector $\mathbf{b}$ has a dimension of n_n. Note that $\mathbf{Y}_t$ is a function of $\mathbf{X}_t$ and $\mathbf{Y}_{t-1}$, which again is a function of $\mathbf{X}_{t-1}$ and $\mathbf{Y}_{t-2}$ and so on. Thus, $\mathbf{Y}_t$ is a function of all the inputs because time $t=0$. The value of $\mathbf{Y}_0$ is typically set to zero. Because the neurons are dependent on the previous output, the network develops a memory. The network uses the memory of the previous inputs to predict the future output. Note that the memory for a long distant time fades away because of the responsible gradient diffusing over time (Bengio et al., 1994).

The output vector $\mathbf{y}$ from RNN is a vector of size n_n, but we can modify the length to the desired length by applying a fully connected layer on top of the RNN. This can be represented as

$$\mathbf{Z}_t=FC(\mathbf{y}_t), \tag{3}$$

where the fully connected layer FC has a weight $\mathbf{W}_{fc}$ of a dimension of $n_n\times k$, where k is the number of the desired output, and here $\mathbf{Z}$ represents the NMO velocity in our implementation. Now, after a single forward pass for a minibatch, we use the mean-squared error of the predicted velocity and the given velocity as

$$E=\frac{1}{m}\sum_{i=1}^{m}(\mathbf{Z}_{\text{given}}-\mathbf{Z}_{\text{predicted}})^2. \tag{4}$$

The error calculated in equation 4 is used for updating the weights in the RNN. Note that the RNN shares the same weights over the complete time series. Using the chain rule, we can write the gradient of the weights as

$$\frac{\partial E}{\partial W}=\sum_{t=1}^{T}\sum_{k=1}^{t}\frac{\partial E_t}{\partial z_t}\frac{\partial z_t}{\partial y_t}\frac{\partial y_t}{\partial y_k}\frac{\partial y_k}{\partial W}, \tag{5}$$

where T is the total time steps in the time series. The red line in Figure 2 shows the direction of the backpropagation, and the weights are updated using a given learning rate, as a step length toward the update. We use the Adam optimization algorithm (Abadi et al., 2015) for minimizing the error between the predicted output and the

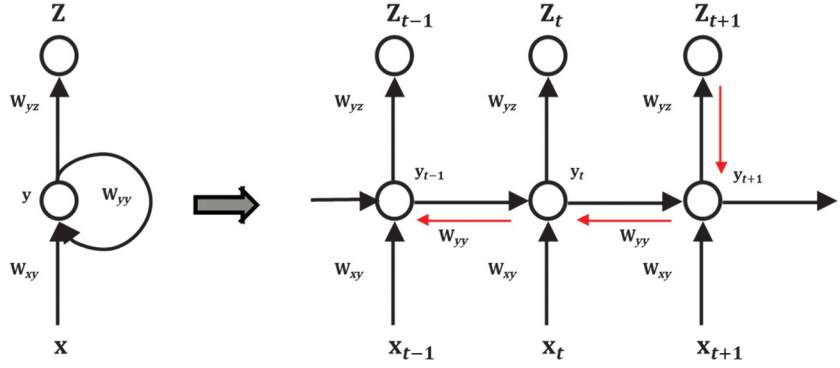

Figure 2. A simple example of an RNN: (a) the recursive form of RNN and (b) the extended form of RNN in time.

true output by updating the weights. Adam optimization uses the first- and second-order moments, and it is invariant to the diagonal rescaling of the gradient.

NMO velocity estimation

Due to the distance between the seismic source and the receiver, there is some shift in the arrival time of the reflected seismic signal, and it increases with the offset. This relationship between the arrival time and the offset is hyperbolic, and for a flat horizontal reflector, the traveltime equation can be given by

$$t^2 = t_0^2 + \frac{x^2}{v^2}, \tag{6}$$

where t_0 is the arrival time for normal incidence, x is the offset, and v is the velocity of the medium above the reflecting surface and is the rms velocity in the case of multiple interfaces. There can be multiple hyperbolas in a common-midpoint (CMP) gather spanning a large offset and in the temporal direction. Thus, estimating correct NMO velocity can be quite challenging. The NMO velocity should be able to place reflection at the correct location of the reflector, and all the energy should line up with the zero-offset reflector to add up constructively during stacking. Conventionally, NMO velocity is calculated by performing velocity scans for a range of velocities and generating semblance curve. We then pick the maximum amplitude at every time step from the semblance curve.

Because the hyperbolas are spread in time and offset, one of the crucial information for estimating velocity at a particular time step is the temporal and spatial information from the nearby time step and offset. Therefore to estimate NMO velocity at a particular time step, we have used a window on the seismic gathers, spanning the whole offset range (NX) and $2N$ in the temporal direction. Figure 3 shows the representation, in which the dimension of the window (represented in the red) used is $2N \times NX$, for estimating the NMO velocity at the center of the box represented in the magenta color and the estimated velocity represented by the green circle in the NMO velocity panel.

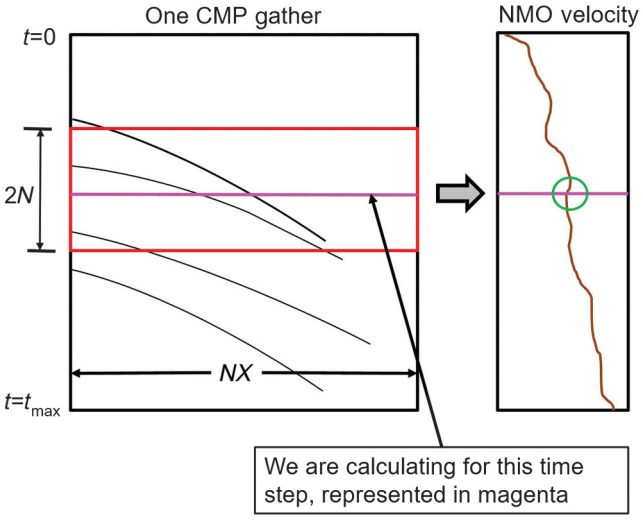

Figure 3. Representation of a CMP gather (with offset size of NX), and the blocks of data from the CMP gather used for creating a single instance of a minibatch. The magenta line represents the time at which velocity is being estimated, and the red block (of size $2N$) represents the data used for estimating the velocity at that point. The second panel on the right shows the corresponding NMO velocity.

Figure 4 shows our RNN network architecture used in the NMO velocity estimation problem. The input ($\mathbf{X}$) to the network is the seismogram in the red block shown in Figure 3 having a dimension of $[p \times n_{\text{inputs}}]$, where p is the time steps ($2N$) and n_{inputs} is the number of offsets (NX); i.e., each X_i represents a time step from the window having a dimension of $[NX \times 1]$, spanning the offset. During training, we also provide correct output ($\mathbf{Z}$), which is the preestimated NMO velocity using the semblance method having a dimension of $[1 \times 1]$. The transitional output from the RNN layer ($\mathbf{Y}$) has the dimension of $[p \times n_{\text{neurons}}]$, where we choose to have $n_{\text{neurons}} = 1000$. The output $\mathbf{Y}$ goes through the fully connected layer FC to produce the predicted output $\mathbf{Z}$. We can represent the network in Figure 4 in mathematical form as

$$\begin{aligned} Z_t &= FC(\phi(\mathbf{X}_t \times \mathbf{W}_x + \mathbf{Y}_{t-1} \times \mathbf{W}_y + \mathbf{b})), \\ &= FC\left(\phi\left([\mathbf{X}_t \quad \mathbf{Y}_{t-1}]\begin{bmatrix}\mathbf{W}_x \\ \mathbf{W}_y\end{bmatrix} + b\right)\right), \end{aligned} \tag{7}$$

where ϕ is the ReLU activation function, weight $\mathbf{W}_x$ has the dimension of $n_{\text{inputs}} \times n_{\text{neurons}}$, weight $\mathbf{W}_y$ has the dimension of $n_{\text{neurons}} \times n_{\text{neurons}}$, and bias $\mathbf{b}$ has a dimension of n_{neurons}. To initialize the network, the weight matrix is drawn from a normal distribution and bias is set to zero.

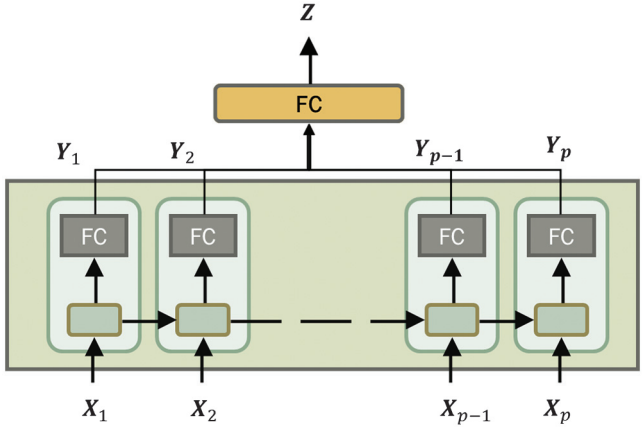

Figure 4. Plot of the RNN architecture used in the NMO velocity estimation. Here, X_i having a dimension of $1 \times NX$, represents the seismogram for a particular time step, all offsets. The term Y_i is the transient result from the network with dimension $p \times n_{\text{neurons}}$, Z is the final NMO velocity at the center of the window considered, p represents the total time window size $2N$, and FC represents a fully connected layer. Here, the RNN is represented similar to Figure 2b, unrolled in time from left to right.

Result

To demonstrate our algorithm, we applied it on prestack 2D land vibroseis data provided by Geofizyka Torun Sp. Z.o.o, Poland, available in the public domain. Figure 5 shows a stacked section. The data have a shot interval of 50 m, receiver interval of 25 m, and an offset interval of 12 m. For simplicity, we choose a small region of the data set marked in the black window. We performed some initial preprocessing and noise removal before the data are ready for velocity analysis. In our test case, we used 1000 neurons in the recurrent network, 60 offsets in each gather, a window size of 100, and a total of 700 samples in the time domain. The data have a sampling rate of 2 ms. We initially generated the NMO velocity using the semblance-based velocity analysis for a small part of the section.

To train the network and optimize the weights of the neurons in the RNN, we prepared the input seismic data before NMO, and the corresponding picked NMO velocity as the output pair. Figure 6 shows the picked NMO velocity using the semblance-based method, which is treated as the given velocity. To train the RNN weights, we picked 10% (approximately 80 gathers) of the CMPs, uniformly placed, and the rest is kept as the testing set. The location of the picked CMPs is shown with the dotted lines in Figure 6. We divide the training set in minibatches, each minibatch containing 700 time samples from a single gather. We ran multiple epochs of 2125 to train the network using Adam optimization and to minimize the mean-squared error between the output velocity and the correct provided velocity. Figure 7 shows the training error for subsequent epochs. The network maps an abstract relationship between the seismic gathers and the optimum NMO velocity. After the network is trained, it is used to predict velocities for all of the gathers (training and testing included). Figure 8 shows the predicted NMO velocity from the recurrent network. We also calculate the percentage difference between the true/given velocity and the predicted velocity for the entire section of the data set. Figure 9 shows the percentage error, having a maximum value of approximately 10%.

To validate our result, we performed a detailed analysis of our estimation at one CMP location. Figure 10 shows one CMP gather at one CMP location

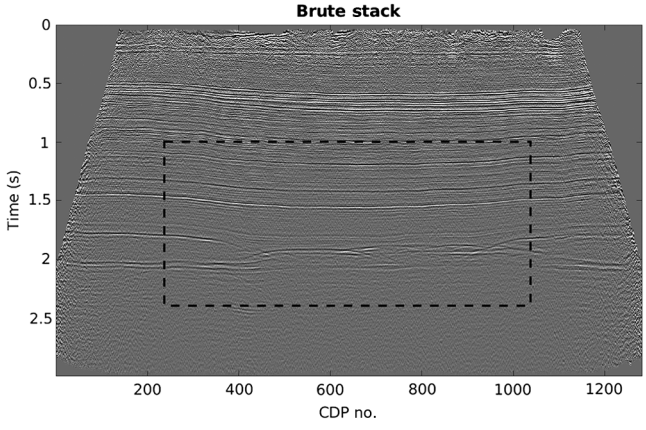

Figure 5. Plot of the stacked section of Poland data. The black window represents the region where we performed our training and testing for estimating NMO velocity.

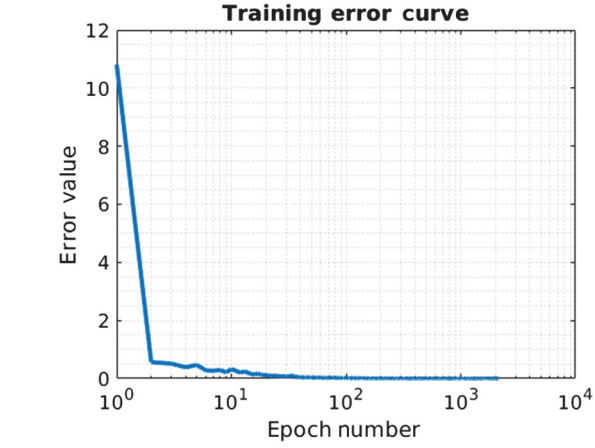

Figure 7. The average epoch error during training versus the epoch number.

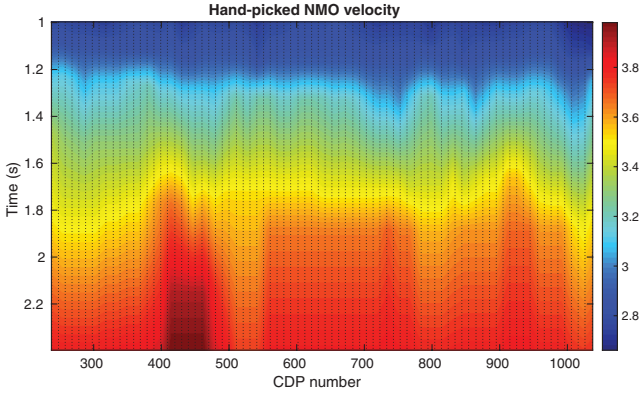

Figure 6. The given hand-picked NMO velocity. The dotted lines represent the location of the training CMPs. There is a total of 80 CMPs for training.

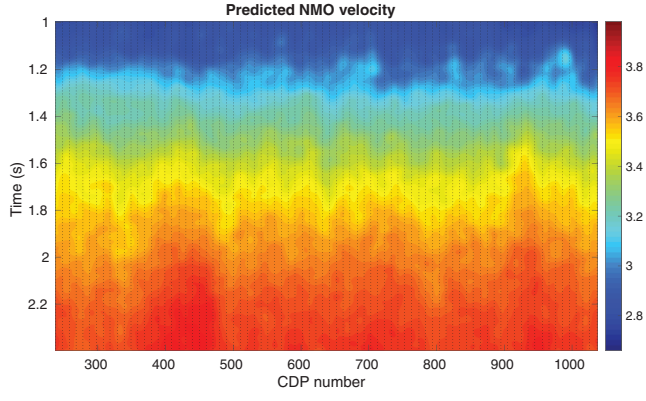

Figure 8. The estimated NMO velocity from the RNN. The dotted lines represent the location of the training CMPs. There is a total of 80 CMPs for training.

(location no. 650), which is a part of the testing data set. Figure 10a shows the noisy uncorrected CMP gather, in which we have pointed out one of the several hyperbolas using a black ellipse. Figure 10b shows the same CMP gather but corrected using the hand-picked NMO velocity from the semblance. Figure 10c shows the CMP gather but corrected using the estimated velocity from the trained RNN. Finally, in Figure 10d, we show a comparison of the NMO velocity, with the given/true velocity in the blue and the estimated velocity in the orange. To check the correctness of estimation for the complete data set, we use the predicted NMO velocity for all of the gathers, performed NMO correction, and finally stacked the gathers to produce a stacked section. We repeat the same procedure with the given hand-picked NMO velocities as well. Figure 11a and 11b shows the stacked section, in which the first stack is generated using the given hand-picked NMO velocity, and the second stack is produced using the network estimated NMO velocity. Just by observation, they seem quite similar; however, there are some differences. We have used a green box to represent regions where the reflector continuity on the stacked section is better in the RNN-predicted velocity and in the red box otherwise.

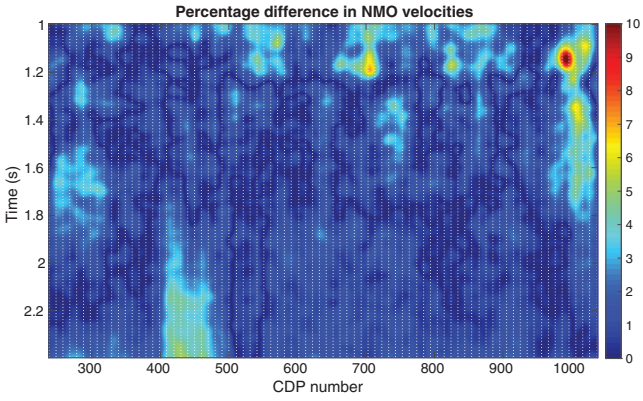

Figure 9. The percentage difference in the estimation of $V_{\text{predicted}}$ from the V_{given} NMO velocity. The dotted lines represent the location of the training CMPs. There is a total of 80 CMPs for training.

Discussion

Conventionally, the NMO velocity estimation — a routine workflow, requires semblance calculated at each CMP followed by velocity picking by an experienced processor, which can be very time consuming. The main benefit of ML comes from training for a few CMPs and its

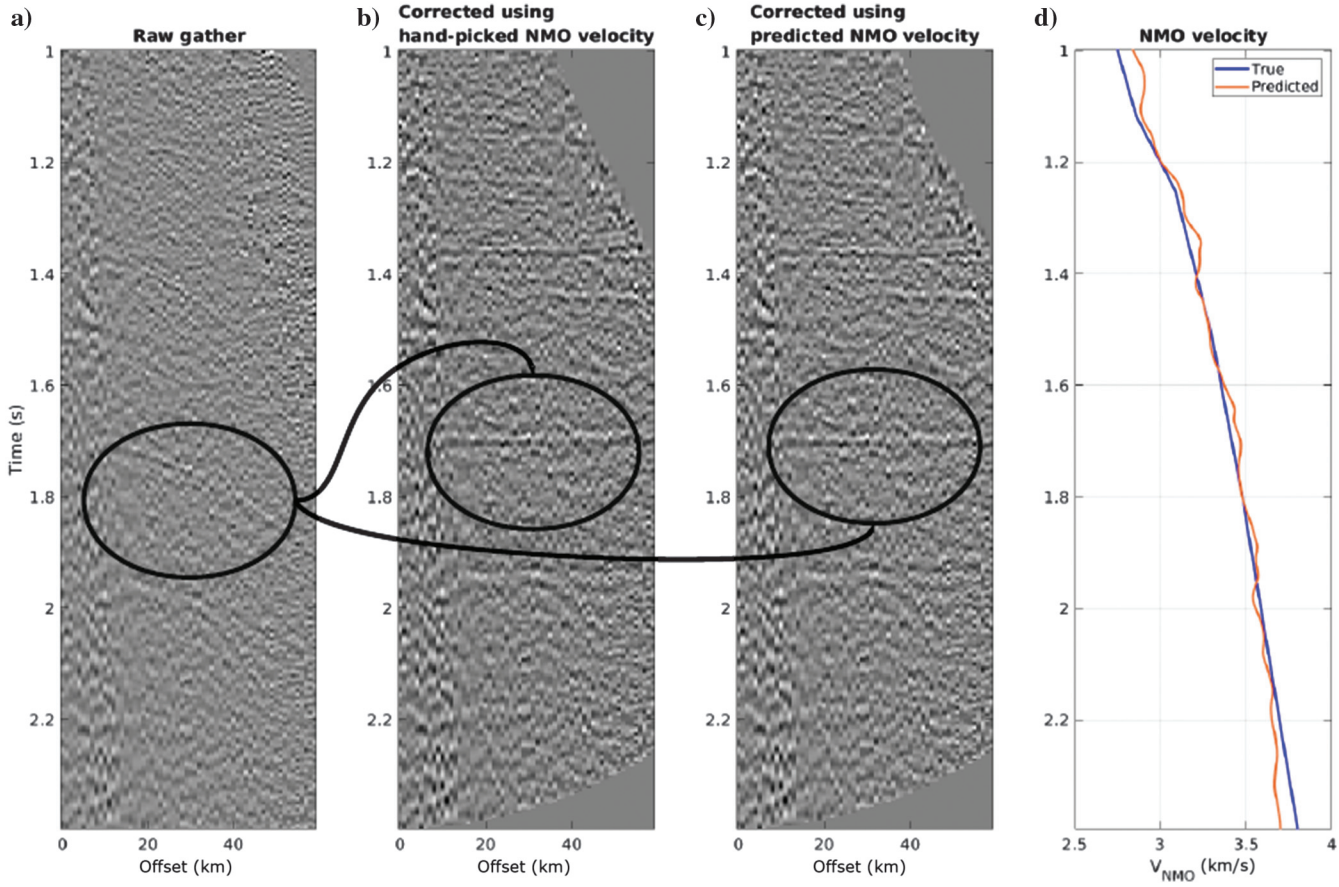

Figure 10. (a) A gather before NMO correction at CMP location 650. One of the hyperbola is highlighted using an ellipse. (b) NMO-corrected gather using the given NMO velocity, and the flatten hyperbola is marked by an ellipse. (c) NMO-corrected gather using the predicted NMO velocity, and again the flatten hyperbola is marked by an ellipse. (d) The comparison of the given NMO velocity (in blue) and the predicted NMO velocity (in orange).

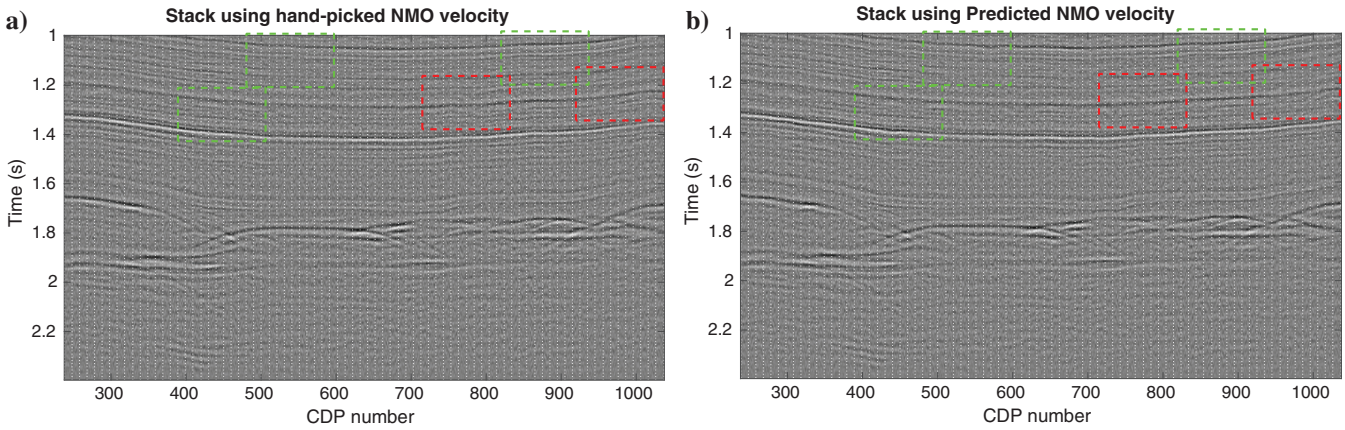

Figure 11. (a) The stacked section after NMO correction using the given NMO velocity and (b) the stacked section after NMO correction using the estimated NMO velocity from the RNN. The green box shows the region where the stack section generated from RNN-predicted velocity has better continuity and the red box otherwise.

subsequent application to the remaining data set instantly to get NMO velocity immediately. In a typical application, very few CMPs are picked for velocity, and, generally, an interpolation is performed in between them. However, performing interpolation does not take into account any input from seismic data and, thus, is prone to produce an erroneous result if any sudden irregularity comes in the seismic data. However, while applying the RNN algorithm, it uses the input data to predict the output velocity from the trained network and thus can handle these irregularities in a much better way.

We used TensorFlow (Abadi et al., 2015) to implement our workflow in a very efficient environment. To train the network, we used Nvidia K-40 GPUs and for training using 80 CMPs; 2125 epochs and with a learning rate of 0.001 to approximately 46 min. However, the training time depends on the number of time samples and offset in the data. Comparing Figures 6 and 8, it is quite evident that the velocity obtained from RNN is quite continuous, unlike the hand-picked velocity, which seems interpolated and has layers.

The RNN applied to the problem is quite flexible and is not limited to just NMO velocity estimation. With a few modifications, it can be applied to other geophysical problems, such as migration velocity analysis. In our implementation of the network, the number of unknowns is approximately 10^6. Even though we used a gradient-based local optimization, in ML, due to the high dimension, there is a great chance that our solution is close to the global minimum. In a low-dimensional problem, there exist several minima. However, in a high-dimensional optimization problem, most of the trajectory followed in optimization, i.e., the critical points, are the saddle points, which are relatively easy to escape for the algorithm (Dauphin et al., 2014).

Conclusion

In this paper, we used an ML tool to solve one of the most common seismic processing problems. We applied the RNN to estimate NMO velocities directly from seismic gathers and use those for NMO correction of the gathers. Before using the network in velocity prediction, it needs to be trained on a few gathers in a supervised fashion using input-output pairs. The input to the training requires the raw gathers before NMO correction (input) and the NMO velocity (output). These initial NMO velocities can be estimated using conventional semblance-based methods. Note that in this problem the temporal and spatial information is necessary from the data. After the network is trained, it can be used to predict the NMO velocity for the remaining gathers. The RNN learns the mapping from the data to the output velocity. Due to the memory property of the recurrent network, current output depends on the past output and the neighborhood dependency can quickly be established; hence, better estimation of the NMO velocity can be obtained. Finally, we demonstrated the method on a real data set from Poland, in which we trained the data set from just 10% of the gathers and predicted the velocity for the rest of the gathers. By comparing the RNN-predicted velocities with those from the semblance-based velocities, we find that they are in excellent agreement.

Acknowledgments

The authors would like to thank editor B. Nemeth, associate editors V. Jayaram and P. Jaiswal, and reviewers J. Walda and S. Verma, along with one anonymous reviewer for their constructive criticisms that helped to improve the manuscript. The authors would also like to thank Geo-Energy Inc. for permission to publish and Google for making TensorFlow available for a general audience, which was used in the implementation.

Data and materials availability

Data associated with this research are available and can be obtained by contacting the corresponding author.

References

Abadi, M., A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D.

Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, 2015, TensorFlow: Large-scale machine learning on heterogeneous systems (Software available from tensorflow.org).

Alfarraj, M., and G. AlRegib, 2018, Petrophysical property estimation from seismic data using recurrent neural networks: 88th Annual International Meeting, SEG, Expanded Abstracts, 2141–2146, doi: 10.1190/segam2018-2995752.1.

Alfarraj, M., N. Keni, and G. AlRegib, 2018, Property prediction from seismic attributes using a boosted ensemble machine learning scheme: SBGf/SEG Machine Learning Workshop.

An, P., and W. M. Moon, 1993, Reservoir characterization using seismic waveform and feedforward neural networks: 63rd Annual International Meeting, SEG, Expanded Abstracts, 1450–1456, doi: 10.1190/1.1487090.

Bahdanau, D., K. Cho, and Y. Bengio, 2014, Neural machine translation by jointly learning to align and translate: arXiv preprint arXiv:1409.0473.

Bengio, Y., P. Simard, and P. Frasconi, 1994, Learning long-term dependencies with gradient descent is difficult: IEEE Transactions on Neural Networks, **5**, 157–166, doi: 10.1109/TNN.72.

Biswas, R., M. K. Sen, V. Das, and T. Mukerji, 2019, Prestack and poststack inversion using a physics-guided convolutional neural network: Interpretation, **7**, no. 3, SE161–SE174, doi: 10.1190/INT-2018-0236.1.

Calderón-Maciás, C., M. K. Sen, and P. L. Stoffa, 1998, Automatic NMO correction and velocity estimation by a feedforward neural network: Geophysics, **63**, 1696–1707, doi: 10.1190/1.1444465.

Calderón-Maciás, C., M. K. Sen, and P. L. Stoffa, 2000, Artificial neural networks for parameter estimation in geophysics: Geophysical Prospecting, **48**, 21–47, doi: 10.1046/j.1365-2478.2000.00171.x.

Cichocki, A., and R. Unbehauen, 1993, Robust estimation of principal components by using neural network learning algorithms: Electronics Letters, **29**, 1869–1870, doi: 10.1049/el:19931244.

Dai, H., and C. MacBeth, 1994, Split shear-wave analysis using an artificial neural network: First Break, **12**, 605–613, doi: 10.3997/1365-2397.1994038.

Dauphin, Y. N., R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, 2014, Identifying and attacking the saddle point problem in high-dimensional non-convex optimization: Advances in Neural Information Processing Systems, 2933–2941.

Di, H., Z. Wang, and G. AlRegib, 2018, Deep convolutional neural networks for seismic salt-body delineation: Presented at theAAPG Annual Convention and Exhibition.

Fish, B. C., and T. Kusuma, 1994, A neural network approach to automate velocity picking: 64th Annual International Meeting, SEG, Expanded Abstracts, 185–188, doi: 10.1190/1.1822888.

Freeman, J. A., and D. M. Skapura, 1991, Algorithms, applications, and programming techniques: Addison-Wesley Publishing Company.

Funahashi, K.-I., and Y. Nakamura, 1993, Approximation of dynamical systems by continuous time recurrent neural networks: Neural Networks, **6**, 801–806, doi: 10.1016/S0893-6080(05)80125-X.

Girshick, R., J. Donahue, T. Darrell, and J. Malik, 2014, Rich feature hierarchies for accurate object detection and semantic segmentation: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 580–587.

He, K., X. Zhang, S. Ren, and J. Sun, 2016, Deep residual learning for image recognition: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 770–778.

Hinton, G. E., 2012, A practical guide to training restricted Boltzmann machines, *in* G. Montavon, G. B. Orr, and K. R. Müller, eds., Neural networks: Tricks of the trade: Springer, 599–619.

Hinton, G. E., and R. R. Salakhutdinov, 2006, Reducing the dimensionality of data with neural networks: Science, **313**, 504–507, doi: 10.1126/science.1127647.

Hochreiter, S., and J. Schmidhuber, 1997, Long short-term memory: Neural Computation, **9**, 1735–1780, doi: 10.1162/neco.1997.9.8.1735.

Hopfield, J. J., 1982, Neural networks and physical systems with emergent collective computational abilities: Proceedings of the National Academy of Sciences, **79**, 2554–2558, doi: 10.1073/pnas.79.8.2554.

Hopfield, J. J., 2007, Hopfield network: Scholarpedia, **2**, 1977 (revision #91363).

Huang, K.-Y., 1997, Hopfield neural network for seismic horizon picking: 67th Annual International Meeting, SEG, Expanded Abstracts, 562–565, doi: 10.1190/1.1885963.

Krizhevsky, A., I. Sutskever, and G. E. Hinton, 2012, Imagenet classification with deep convolutional neural networks: Advances in Neural Information Processing Systems, 1097–1105.

LeCun, Y., B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, 1989, Backpropagation applied to handwritten zip code recognition: Neural Computation, **1**, 541–551, doi: 10.1162/neco.1989.1.4.541.

Lee, J., and I. Tashev, 2015, High-level feature representation using recurrent neural network for speech emotion recognition: 16th Annual Conference of the International Speech Communication Association.

Lukoševičius, M., and H. Jaeger, 2009, Reservoir computing approaches to recurrent neural network training: Computer Science Review, **3**, 127–149, doi: 10.1016/j.cosrev.2009.03.005.

Ma, Y., X. Ji, T. W. Fei, and Y. Luo, 2018, Automatic velocity picking with convolutional neural networks: 88th

Annual International Meeting, SEG, Expanded Abstracts, 2066–2070, doi: 10.1190/segam2018-2987088.1.
Mao, J., W. Xu, Y. Yang, J. Wang, Z. Huang, and A. Yuille, 2014, Deep captioning with multimodal recurrent neural networks (m-RNN): arXiv preprint arXiv:1412.6632.
McCormack, M. D., D. E. Zaucha, and D. W. Dushek, 1993, First-break refraction event picking and seismic data trace editing using neural networks: Geophysics, **58**, 67–78, doi: 10.1190/1.1443352.
Moseley, B., A. Markham, and T. Nissen-Meyer, 2018, Fast approximate simulation of seismic waves with deep learning: arXiv preprint arXiv:1807.06873.
Moya, A., and K. Irikura, 2010, Inversion of a velocity model using artificial neural networks: Computers and Geosciences, **36**, 1474–1483, doi: 10.1016/j.cageo.2009.08.010.
Murat, M. E., and A. J. Rudman, 1992, Automated first arrival picking: A neural network approach 1: Geophysical Prospecting, **40**, 587–604, doi: 10.1111/j.1365-2478.1992.tb00543.x.
Phan, S., and M. K. Sen, 2018, Hopfield networks for high-resolution prestack seismic inversion: 88th Annual International Meeting, SEG, Expanded Abstracts, 526–530, doi: 10.1190/segam2018-2996244.1.
Richardson, A., 2018, Seismic full-waveform inversion using deep learning tools and techniques: arXiv preprint arXiv:1801.07232.
Ronneberger, O., P. Fischer, and T. Brox, 2015, U-net: Convolutional networks for biomedical image segmentation: International Conference on Medical Image Computing and Computer-Assisted Intervention, 234–241.
Röth, G., and A. Tarantola, 1994, Neural networks and inversion of seismic data: Journal of Geophysical Research: Solid Earth, **99**, 6753–6768, doi: 10.1029/93JB01563.
Russakovsky, O., J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, 2015, Imagenet large scale visual recognition challenge: International Journal of Computer Vision, **115**, 211–252, doi: 10.1007/s11263-015-0816-y.
Schmidt, J., and F. A. Hadsell, 1992, Neural network stacking velocity picking: 62nd Annual International Meeting, SEG, Expanded Abstracts, 18–21, doi: 10.1190/1.1822036.
Shi, Y., X. Wu, and S. Fomel, 2018, Automatic salt-body classification using a deep convolutional neural network: 88th Annual International Meeting, SEG, Expanded Abstracts, 1971–1975, doi: 10.1190/segam2018-2997304.1.
Vamaraju, J., and M. K. Sen, 2018, Mean field Boltzmann machines for high resolution Kirchhoff migration: 88th Annual International Meeting, SEG, Expanded Abstracts, 2006–2010, doi: 10.1190/segam2018-2997793.1.
Wang, J., and T.-l. Teng, 1997, Identification and picking of s phase using an artificial neural network: Bulletin of the Seismological Society of America, **87**, 1140–1149.
Wu, H., B. Zhang, F. Li, and N. Liu, 2019, Semiautomatic first-arrival picking of microseismic events by using the pixel-wise convolutional image segmentation method: Geophysics, **84**, no. 3, V143–V155, doi: 10.1190/geo2018-0389.1.
Wu, X., Y. Shi, S. Fomel, and L. Liang, 2018, Convolutional neural networks for fault interpretation in seismic images: 88th Annual International Meeting, SEG, Expanded Abstracts, 1946–1950, doi: 10.1190/segam2018-2995341.1.

Reetam Biswas received an M.S. from the Indian Institute of Technology, Kharagpur, and he is a graduate student at the University of Texas at Austin. His research interests include transdimensional seismic inversion, full-waveform inversion, and machine learning.

Anthony Vassiliou received a Ph.D. (1986) in civil engineering from the University of Calgary. He worked in the past for Mobil R&D Corporation and Amoco Production R&D. He founded GeoEnergy Inc. in 1998, of which he is currently the CEO and president. His research interests include migration velocity analysis, wave equation imaging, full-waveform inversion for 3D land seismic data, postmigration anisotropic waveform inversion, reservoir characterization, and machine learning for velocity model building.

Rodney Stromberg received a B.S. (1972) in geologic engineering from the University of Utah. He started at G.S.I. and then went on to Mobil Oil and Sohio among other seismic service companies prior to his current position beginning 2011 at GeoEnergy Inc. His research interests revolve around seismic processing, including developing methods and techniques for reducing noise and multiples embedded in 3D land seismic collections.

Mrinal K. Sen received an M.S. from IIT(ISM) Dhanbad and a Ph.D. from the University of Hawaii at Manoa, USA. He is a professor of geophysics and the holder of the Jackson Chair in applied seismology at the Department of Geological Sciences and concurrently serves as the interim director of the Institute for Geophysics at the University of Texas at Austin. During 2013 and 2014, he served as the director of the National Geophysical Research Institute, Hyderabad, India. He is an honorary member of SEG and is the recipient of SEG's 2018 Virgil Kauffman gold medal.