Seismic velocity model building using recurrent neural networks: A frequency-stepping approach

Hani Alzahrani & Jeffrey Shragge

Center for Wave Phenomena, Department of Geophysics, Colorado School of Mines, Golden CO 80401
Corresponding Email: halzahrani@mymail.mines.edu

ABSTRACT

Data-driven artificial neural networks (ANNs) demonstrably offer a number of advantages over conventional deterministic methods in a wide range of geophysical problems. For seismic velocity model building, judiciously trained ANNs offer the possibility of estimating subsurface velocity models more efficiently than deterministic full-waveform inversion (FWI) approaches with reduced sensitivity to many inherent FWI non-linearity issues; however, there are substantial challenges with effective and efficient network training. Motivated by the multi-scale approach commonly used to address FWI non-linearity challenges, we develop a frequency-stepping approach for velocity model building that uses a sequence-to-sequence recurrent neural network (RNN) with built-in long short-term memory (LSTM). The input sequences to the LSTM-RNN consist of the frequency-domain seismic data ordered by frequency from lowest available to highest usable, while the corresponding output sequences are frequency-dependent smoothed velocity models. Qualitative and quantitative analyses of the network testing results show that such a network has the potential to estimating complex velocity models starting from their smoothest components.

Key words: Recurrent Neural Networks, Seismic Velocity Inversion

1 INTRODUCTION

A major challenge in seismic exploration is building velocity models that minimize the misfit between observed and modeled seismic data. Over the past decades, this challenge has been tackled using mainly deterministic methods; however, the resurgence of neural networks in recent years largely driven by advances in GPU computing is introducing new paradigms for automated velocity model building. These methods aim to take advantage of artificial neural networks (ANNs) to bypass challenges associated with deterministic methods. ANNs also offer the potential of improved efficiency because the main cost overhead comes from training, after which predictions can be made at the cost of matrix-vector multiplications. The challenges of velocity model building using ANNs have been successfully addressed by a number of authors (Araya-Polo et al., 2018; Phan and Sen, 2018; Vamaraju and Sen, 2019; Zhang and Alkhalifah, 2019; Sun et al., 2020; He and Wang, 2021).

A recent trend in deterministic seismic velocity model building is to use full waveform inversion (FWI) to obtain a high-resolution subsurface velocity model that aims to minimize the misfit between observed and modeled data (Lailly, 1983; Tarantola, 1984). Because the governing wave equation is non-linear with respect to medium parameters, the FWI process is highly non-linear, which leads to non-convex objective functions with numerous local minima that frequently forestall the convergence of optimization approaches toward the globally minimum solution (Virieux and Operto, 2009). Such challenges are exacerbated by the oscillatory nature of seismic data, which demands using starting models that simulates seismic waveforms to within half a period of recorded data to avoid cycle skipping, which causes the inversion to converge toward a local minimum that may be far away from the true global solution. Finally, FWI is computationally expensive because each iteration usually requires (tens of) thousands of seismic wave simulations.

To mitigate the non-linearity issues, numerous FWI approaches follow a multi-scale strategy (Bunks et al., 1995) where the lowest frequencies in the data are inverted first to recover the longer wavelength model components, followed by higher frequencies to infill shorter wavelength detail. Because lower frequencies are less sensitive to cycle skipping than higher frequencies, inverting

them first helps to recover a smoother velocity model and move the inversion toward the global minimum. The smooth velocity model from the previous step can then be used as the initial model to invert the next higher frequency band. This iterative approach then continues until the final frequency band is inverted.

Motivated by this approach, we present a ANN methodology that aims to mimic the dynamics of the multi-scale FWI strategy. We achieve this by using a class of ANNs referred to as recurrent neural networks (RNNs) whose structure is ideal for sequential data. We decompose the seismic data into frequency slices that are sequentially input to the RNN along with a frequency-dependent smoothed version of the model. We gradually increase the input data frequencies from the lowest usable to the highest chosen component. Along with each frequency input, we inject the smoothed version of the velocity model appropriate for that stage. We begin by introducing RNNs and their different types including the benefits of using long short-term memory architectures. We then discuss the proposed method and detail the specific RNN used in this work. We subsequently show a number of the network testing results and discuss trends in the recovered models. Finally, we quantitatively evaluate the network performance on the ensemble of testing models in both the velocity model and data domains.

2 TRAINING DATA GENERATION

To train an ANN, we need a training data set that contains input-output pairs. Here, the input data consist of simulated surface-recorded frequency-domain seismic waveforms, while the outputs are the corresponding velocity models. We generate a total of $17,000$ velocity models that coarsely mimic subsurface geologic structures. The models are limited to five layers at most. The model layers are constructed sequentially from top to bottom, with each layer interface specified using spline interpolation between randomly generated control points. The number and locations of control points are randomly assigned. After constructing all interfaces, each layer is assigned a randomly generated velocity between 1.5 km/s and 4.0 km/s; some randomly selected layers are assigned a linear velocity gradient, while the others are assigned a constant velocity. Figure 1 shows 16 representative sample velocity models with layers of a range of dip angles and thicknesses. Some layers are curved in the shape of a syncline, while others have the form of an anticline. Velocity gradients are present in some models where the velocity is smoothly increasing with depth.

We then use the developed velocity models for numerical experimentation. A total of $10,000$ input-output pairs are used for network training, 2000 are used for training validation, and 5000 are reserved for network testing. All 2D models are of size $N_x \times N_z = 500 \times 500$ grid points at a $\Delta x = \Delta z = 4$ m discretization. Sources and receivers are placed every $\Delta s = 160$ m and $\Delta r = 48$ m on the grid, respectively. We simultate 2D seismic data in the time domain using a GPU-based constant-density acoustic wave equation solver and a 15 Hz Ricker wavelet source. We the apply a temporal FFT to transform the data set and extract data at nine frequencies ranging from 1.1 Hz to 9.9 Hz in $\Delta f = 1.1$ Hz increments. The complex waveforms are saved for network training and testing discussed below.

3 RECURRENT NEURAL NETWORKS

The most straightforward ANN architecture (feed-forward fully connected neural network) is capable of learning the connection between an input-output pair using trainable weights, the values of which are "learned" during the training process. These networks consist of an input layer, one or more hidden layer(s), and an output layer. Following the success of fully connected neural networks, more sophisticated ways of connecting network input to its hidden and output layers have been developed, among the most successful and widely used of which are convolutional neural networks (CNNs) and recurrent neural networks (RNNs). While CNNs simply replace the connective weights by a convolutional process, RNNs are more dynamic and are commonly used when input data can be decomposed into a sequence. For example, machine translation applications show that feeding the input text word by word to the ANN produces superior results than when feeding the entire text at once (Sutskever et al., 2014).

Another prominent feature of RNNs is that they have a "hidden state" vector which acts as the network's memory that helps it recall what it has learned between the first and final inputs. There are four types of RNNs depending on the number of network inputs and outputs. The most basic of these is the one-to-one RNN, which gives a single output for a single input (Figure 2a). The RNN shown in Figure 2a is similar to a fully connected network with one hidden layer. In fully connected networks, the hidden layer has one set of weights that are multiplied by the input vector, and each input is treated independently. That is, the output of a certain input does not depend on the output of the previous input. In a machine translation network, this would mean that each input word in a sentence is translated independently without looking into its context. However, a hidden layer in a one-to-one RNN has two sets of weights, one is multiplied by the input vector while the other is multiplied by the hidden state vector. The hidden state vector has information about the previous network inputs. Using the same machine translation example, the hidden-state vector contains information about the context of the input word in the sentence, and is updated at each input.

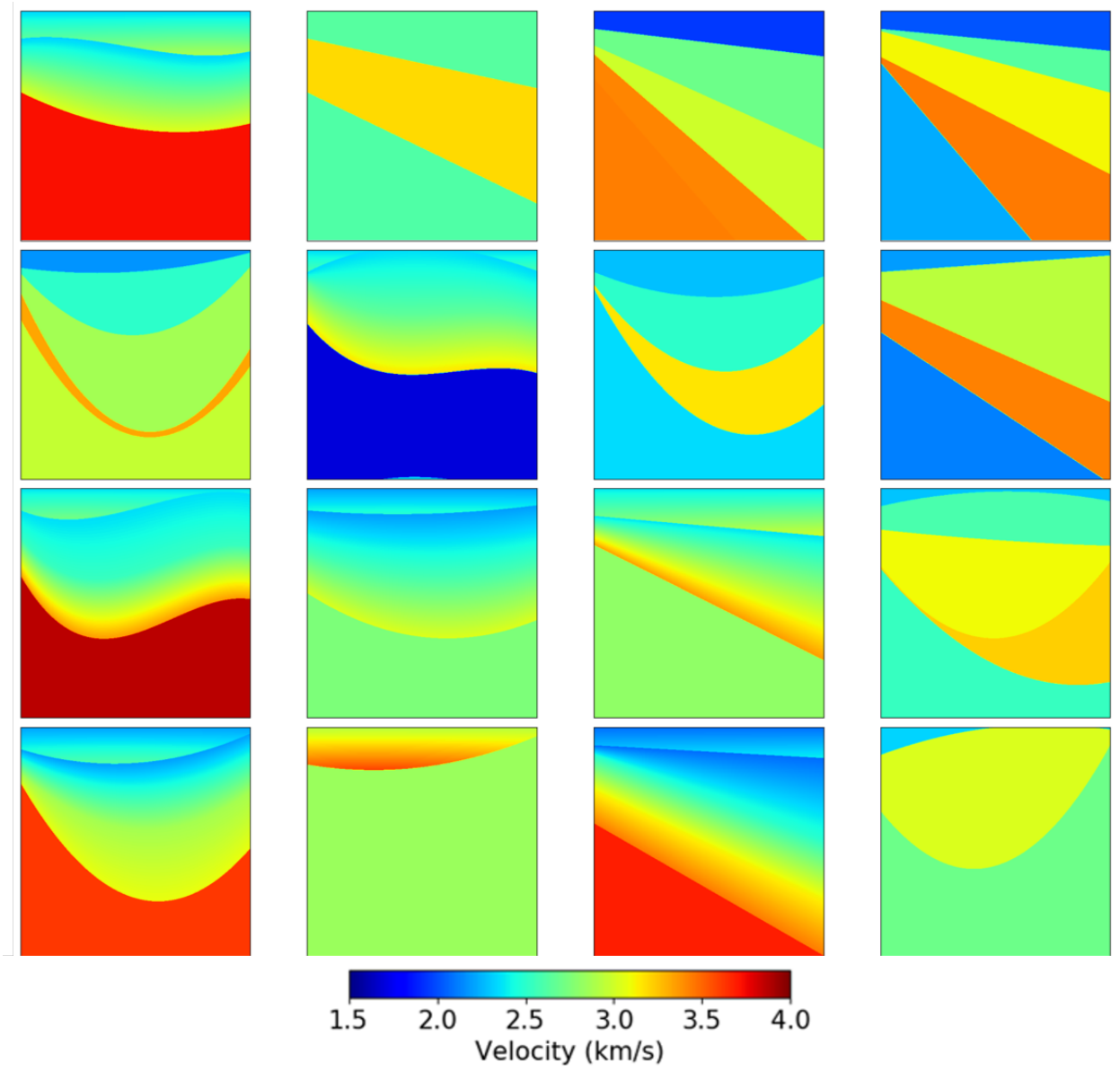

Figure 1. 16 representative 2.0 km $\times$2.0 km synthetic velocity models generated for network training.

In addition to one-to-one RNNs there is the many-to-many (or sequence-to-sequence) RNN (Figure 2b) that we use here to produce optimal results. In the sequence-to-sequence RNN the vectors $\mathbf{h_1}$ to $\mathbf{h_n}$ represent the network hidden state vectors, $\mathbf{x_1}$ to $\mathbf{x_n}$ the inputs vectors, and $\mathbf{y_1}$ to $\mathbf{y_n}$ the corresponding output vectors. The training process starts by injecting the first input $\mathbf{x_1}$ into the network, along with its corresponding output $\mathbf{y_1}$. Vector $\mathbf{h_0}$ is updated based on what the network has learned from the connection between $\mathbf{x_1}$ and $\mathbf{y_1}$ to produce $\mathbf{h_1}$. The process is repeated for the remaining input-output pairs. The final output $\mathbf{y_n}$ is computed using the continually updated state vector $\mathbf{h_n}$ which, in principle, carries the cumulative network learning along with the final input.

The network structure in Figure 2b is well suited to applying a frequency-domain multi-scale scheme for seismic velocity model building. The lowest usable frequency data are injected as the $\mathbf{x_1}$ vector, followed by sequentially higher frequencies until the highest frequency data are injected as $\mathbf{x_n}$. The output at each step, $\mathbf{y_1}$ to $\mathbf{y_n}$, ideally contains the model wavenumber components corresponding to the highest frequency used to that point. Because such velocity models are not straightforward to construct, we apply a 2D Gaussian filter of varying radius to smooth velocity models to an appropriate degree for the maximum frequency used.

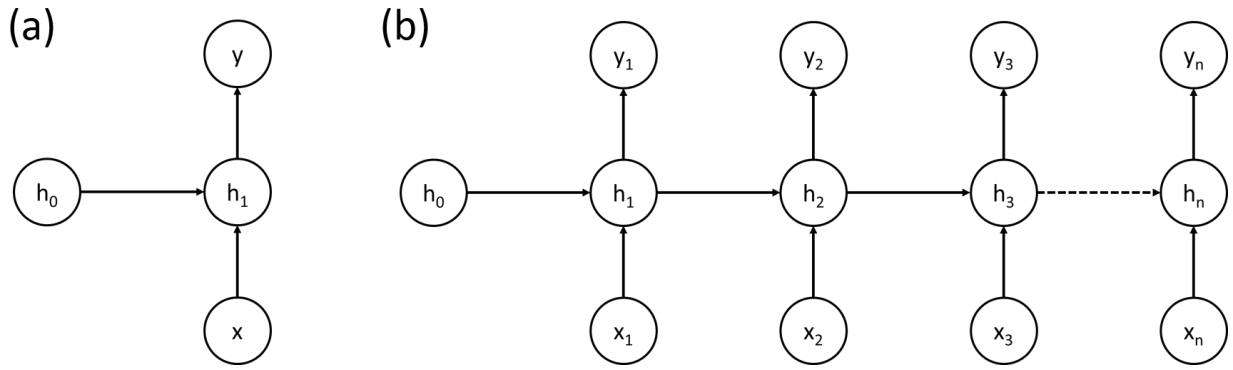

Figure 2. Diagrams illustrating the four different types of RNNs. (a) A one-to-one RNN and (b) the many-to-many RNN network used in this work.

Using this structure, the network first learns the connection between the lowest data frequency and the general velocity model trend. The learned connection is then carried along in the form of the hidden state vector to help the network learn the connection between the second frequency and a less-smoothed velocity model. The cumulative network knowledge learned in the first two steps is carried along in the same manner to the last frequency in the data. We assert that this frequency-stepping approach allows the network to reconstruct complex velocity models by recovering their components in a multi-scale fashion (i.e., from simplest to most complex).

3.1 Long short-term memory RNNs

A common drawback with standard RNNs is the vanishing or exploding gradient problem (Bengio et al., 1994). To address these issues, Hochreiter and Schmidhuber (1997) develop a particular class of RNNs called long short-term memory (LSTM). LSTMs can hold information for a longer duration by introducing a gating mechanism at each step to determine the relevant information that should be carried forward to the next step in the sequence, and the irrelevant information that should be discarded. In addition to the hidden state vector that serves as the short-term network memory, LSTM adds a cell-state vector to serve as the long-term network memory.

Figure 3b illustrates how information flows inside an LSTM cell. Each light-green box represents a layer, with its own set of weights and biases learned during training, and a predefined activation function. The text inside each square represents the layer's non-linear activation function, which introduces non-linearity into the system and can restrict output values to fall within a certain range. Layer σ represents a sigmoid function that normalizes the input between $[0, 1]$, while ReLU (rectified linear units) sets all negative values to zero while keeping the positive values untouched. As opposed to a standard single-layer RNN cell (Figure 3a), an LSTM cell (Figure 3b) consists of four trainable layers. The four layers constituting an LSTM cell are (from left to right) the forget, input, candidate-cell-state, and output layers. Each layer in an RNN cell has two inputs x_t and h_{t-1}. Consequently, each layer has two different weight matrices; one applied to the cell input x_t, the other to the hidden state of the previous cell h_{t-1}. All four layers take the same two x_t and h_{t-1} inputs. The cell-state vector is updated using the outputs of the first three layers, while the output of the fourth layer contributes to updating the hidden-state vector. The ReLU box with a white background is an activation function applied to the input vector and does not constitute a layer.

To simplify the operations of an LSTM cell (Figure 3b), consider the history of the cell-state vector as it is updated from c_{t-1} to c_t. When progressing through the LSTM cell it undergoes point-wise multiplication and addition operations with other vectors. The first operation is a multiplication with the forget vector (f_t), which is the output of a sigmoid layer (or a 'forget layer') that takes x_t and h_{t-1} as input, and produces a vector with values ranging between 0 (which means completely forget) and 1 (which means completely retain) as output. Therefore, the updated cell-state vector at this stage is given by ($c_t = f_t \times c_{t-1}$ + ...). The second more complicated addition operation involves multiple steps. First, we compute the i_t vector using the weights of the input layer and the input x_t and h_{t-1} vectors. Next, we compute $\tilde{C}_t$ using the same inputs and the weights of the candidate-cell-state layer. Then, we perform a point-wise multiplication of these two vectors (i.e., $i_t \times \tilde{C}_t$) and add the result to $f_t \times c_{t-1}$. Therefore, the full equation for computing the updated cell-state vector is $c_t = f_t \times c_{t-1} + i_t \times \tilde{C}_t$. This process first removes what the network believes to be irrelevant information, and then adds in the relevant information that the network learned at this step. After computing c_t, we use it to compute the updated hidden-state vector h_t. We first use the weights of the output layer along with the input x_t and h_{t-1} vectors to compute the output vector o_t. Next, we apply an activation function (ReLU in our experiment) to the

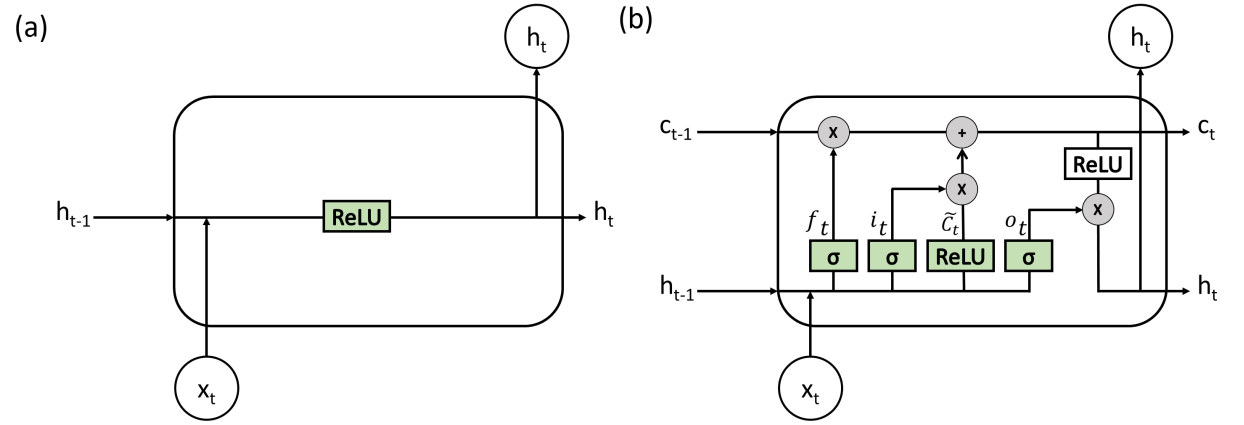

Figure 3. Diagrams showing the structure of (a) a standard RNN cell and (b) an LSTM cell. h_{t-1} and c_{t-1} are the hidden and cell state vectors coming from the previous cell, x_t is the cell's input, and h_t and c_t are the updated hidden and cell state vectors. Each green rectangle represents a layer with its own weights and biases. The text inside each square represents its activation function. While a standard RNN cell has one layer, an LSTM cell has four layers. From left to right, these are the forget layer, the input layer, the candidate-cell-state layer, and the output layer. The gray circles represent point-wise mathematical operations between two vectors.

updated cell-state vector c_t and multiply the resulting vector with the output vector (o_t). Therefore, the equation for computing the updated hidden state vector is $h_t = \text{ReLU}(c_t) \times o_t$.

Looking at the diagrams in Figure 3 and with the preceding description, it should be evident why LSTM-RNNs are better than standard RNNs at retaining relevant information for long sequences. The cell-state vector in an LSTM cell acts as a filter on what information to keep in the hidden-state vector, what information to forget, and what information to add at the current step. The absence of a similar vector in standard RNNs means that at each step every element of the hidden state vector is updated using information learned at the current step, which causes information learned at early steps to be overwritten at later steps.

Figure 4 shows the network architecture used in our experiment (starting at the bottom and moving upward). Each gray square represents an LSTM cell. An LSTM layer consists of a sequence of LSTM cells, with the sequence length being the number of frequency steps in the input data (here nine). The curved arrows show network recurrence, where the output at each step is reinjected as input at the next step (hence *recurrent* neural networks). To illustrate how information flows through the network in a forward pass (i.e., computing velocities from seismic data), Figure 5 shows the network unrolled with no recurrence arrows. As illustrated previously, computing the output of an LSTM cell requires an input vector in addition to the hidden- and cell-state vectors of the previous (frequency) step. The initial state vectors (i.e., h_{i0} and c_{i0} for $i = 1, 2, 3$) are all initialized to zero. A forward run of the network in Figure 5 begins by injecting the 1.1 Hz seismic data for all sources as the x_1 vector and initializing $h_{10} = c_{10} = 0$. Using the weight matrices of LSTM_{11} (i.e., the weight matrix for each of the four layers comprising the cell) and the process explained previously, we compute the updated output hidden-state h_{11} and cell-state c_{11} vectors.

During the next stage the network proceeds both vertically (to the next layer) and horizontally (to the next frequency). We will consider the vertical propagation of information first. The LSTM_{12} cell takes h_{11} as its input vector and the null h_{20} and c_{20} vectors and uses the layer weights to compute h_{21} and c_{21}. The same process is repeated for LSTM_{31}, where h_{21} acts as the input vector, to produce h_{31} and c_{31}. Next, we compute the velocity model at 1.1 Hz using the weights of the fully connected output layer and the vector h_{31} as the layer input. It is important to emphasize that while the network's long-term memory (i.e., c_{31}) is not used to construct the 1.1 Hz velocity model, it is used inside each LSTM cell to compute the hidden-state vector. Therefore, the information stored in c_{31} is encoded in the final hidden-state vector h_{31}, which we use to compute the velocity model for the current frequency.

Next, we go back to the LSTM_{11} cell to demonstrate the propagation of information in the same layer from one frequency to the next. LSTM_{12} takes the seismic data for all sources that correspond to 2.2 Hz as input vector x_2, along with the hidden-state h_{11} and the cell-state c_{11} vectors of the previous cell in the same layer. It then uses its weights to compute h_{12} and c_{12}. Using this procedure, the information learned from the lowest frequency is carried along in the cell-state vector to the highest usable data frequency. With this procedure, data from all frequencies as well as the memory of all layers contribute to the velocity model predicted at 9.9 Hz.

The training process performs a forward run using the network weights at that epoch (iteration) to produce nine output velocity models. We then compute the mean-squared error misfit between each predicted and true velocity models, which is used to update the weights for each layer through backpropagation. Figure 6 shows an example of the set of nine velocity models that are injected

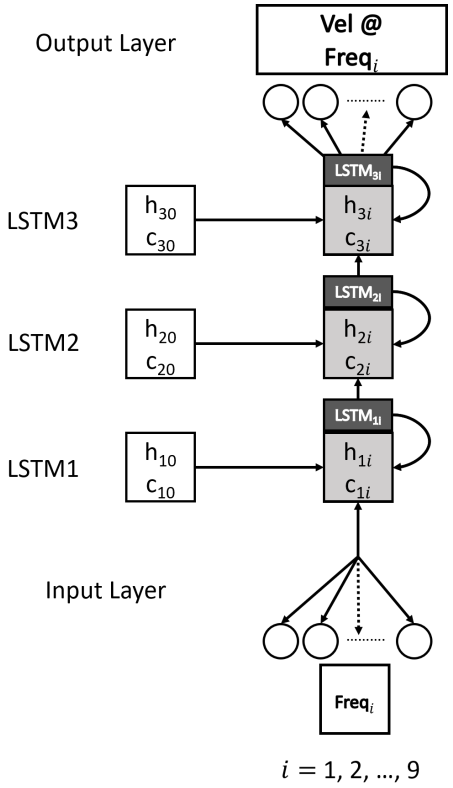

Figure 4. A sketch of the RNN used in this work consisting of three LSTM layers (LSTM1-LSTM3) with each having its own hidden state. The curved arrows indicate recurrence, with the h and c outputs at each step i are injected as input at $i + 1$. Increasingly higher frequencies are injected successively along with their corresponding filtered velocity models.

as the network output along with the data frequency to which they are attached. The 1.1 Hz model only has the general trend of the velocity values; however, higher wavenumber components are incorporated slowly as higher frequencies are introduced until reaching the 9.9 Hz velocity model (with a filter size of only 2×2 grid points, it has all the velocity structures in the true model).

Due to GPU memory limitations, we downsample the velocity models from 500×500 first to 250×250 and then to 125×125 grid points. During each downsampling step, the value at each point in the coarser grid is computed as the weighted average of nine points on the finer grid. Because of the frequency range used in this experiment, we expect the recovered models to be smooth. Therefore, this downsampling procedure has negligible effects on the resolution of the recovered models. The batch size for network training is 100 input/output pairs. We compute a total of 2000 training epochs and perform frequency-dependent velocity smoothing on the fly. Figure 7 shows the training and validation curves for the network. We use the networks weights at 1200 epochs as this is the point after which the validation curve remains nearly constant. The observed high-frequency oscillations are likely due to clipping the y-axis to a small range (i.e., $[0, 0.5]$) to more clearly show the behavior of both curves.

4 NETWORK TESTING

The trained RNN is tested using the $5,000$ models reserved for the testing phase. For each set of input seismic data, the trained network outputs nine different velocity models with different smoothing levels (i.e., one velocity model for each input frequency). Figure 8b and 8c show the nine recovered models along with their cross-sections for the testing model in Figure 8a. Following the trend of the training data shown in Figure 6, the recovered 1.1 Hz model is a very smooth representation of the true one. As the input frequency increases, the network incorporates more details to the recovered model until at 9.9 Hz it exhibits all the features of the true model. This behavior is most apparent at the vertical cross-sections (Figure 8c). The predicted velocity starts as the mean of the two layers. However, as the input frequency increases, the predicted model forms two distinct layers, with the velocity gradient of the top layer being accurately recovered by 9.9 Hz. This behavior is also observed in the thin portion of the top layer

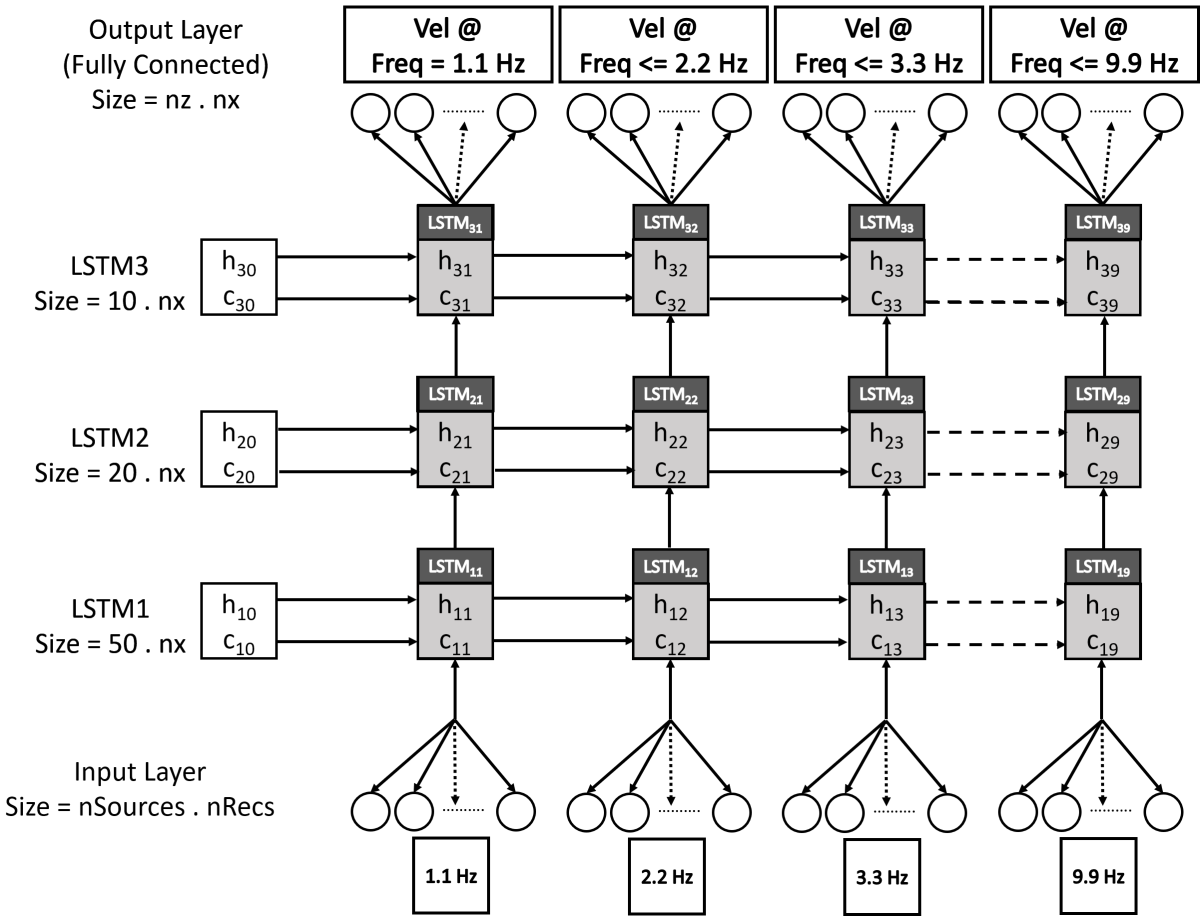

Figure 5. A sketch of the RNN used in this work consisting of three LSTM layers (rows LSTM1-LSTM3) and nine different frequencies (columns 1.1 Hz-9.9 Hz) with each having its own hidden state. The LSTM_{ij} boxes indicate recurrence, with the h_i and c_i outputs at each frequency step i are injected as input at $i+1$. Increasingly higher frequencies are injected successively along with their corresponding filtered velocity models.

toward the upper-left corner of the model. This portion is completely absent from the models recovered at lower frequencies, and it is first resolvable by the seismic data (as well as the trained RNN) at approximately 5.5 Hz. Its shape and size gets accurately reconstructed by 9.9 Hz.

We first assess the accuracy of the trained LSTM-RNN qualitatively by analyzing four testing models that show features representing most of the predicted models (Figure 9). The testing model in Figure 9a consists of a number of relatively thin layers at the shallow section and a thick deep layer, with velocities increasing with depth. The recovered model (Figure 9b and 9c) accurately captures the general velocity structure trend. Due to their thinness, the top layers are approximated with a velocity gradient. However, the deeper thick layer shape and velocity are more accurately recovered. We see a similar behavior in the second testing example (Figure 9d-9f), where the network approximates the thick shallow layers with a velocity gradient that accurately captures the low-wavenumber trend. The cross-section in Figure 9f shows that the velocity values of the two deeper layers are close, which might explain why the network seems to recover them as one thick layer with an averaged velocity value. The third testing example (Figure 9g-9i) shows a common phenomenon where the velocities of the shallower layers are well recovered, while those of the deeper layers are poorly estimated especially when they are relatively thin. However, this could be due to the lack of illumination of the deeper layer. Figure 9j-9l shows another common form of misfit between true and predicted models. When there is a strong velocity inversion (i.e., between the third and fourth layers), the recovered model trends toward the third layer velocity, but before reaching its true velocity value moves in the opposite direction to recover the slower velocity of the deeper layer.

To obtain a more robust assessment of network performance, we use the trained network to estimate the velocity models for all 5000 testing-data inputs. We then compute the normalized rms (NRMS) misfit between the true and predicted 9.9 Hz models. Because the recovered and true models have different dimensions (i.e., 125×125 and 500×500 grid points, respectively), we

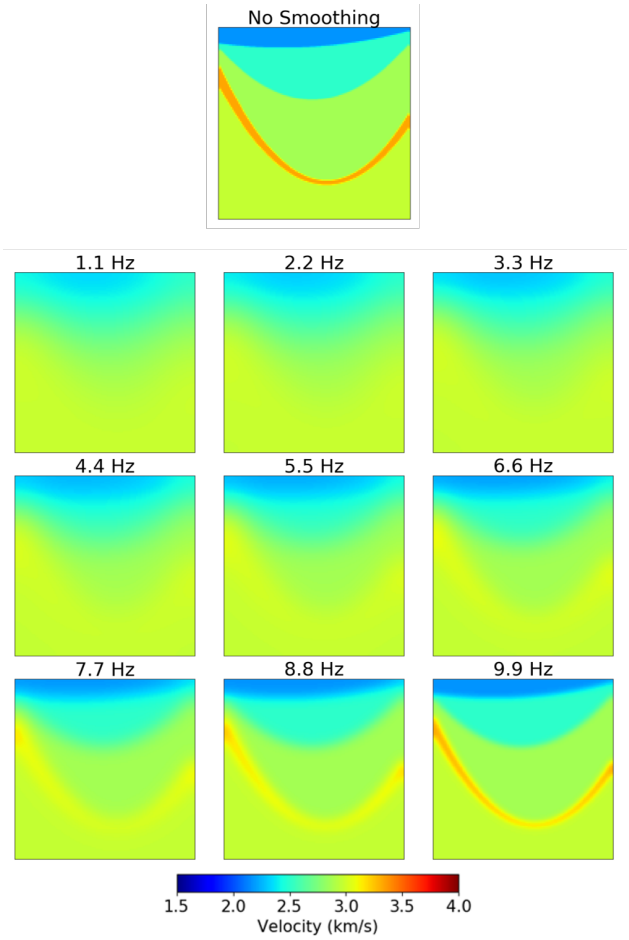

Figure 6. A training velocity model with different smoothing levels. The frequency value at the top of each model denotes the frequency value that models is attached to in the RNN.

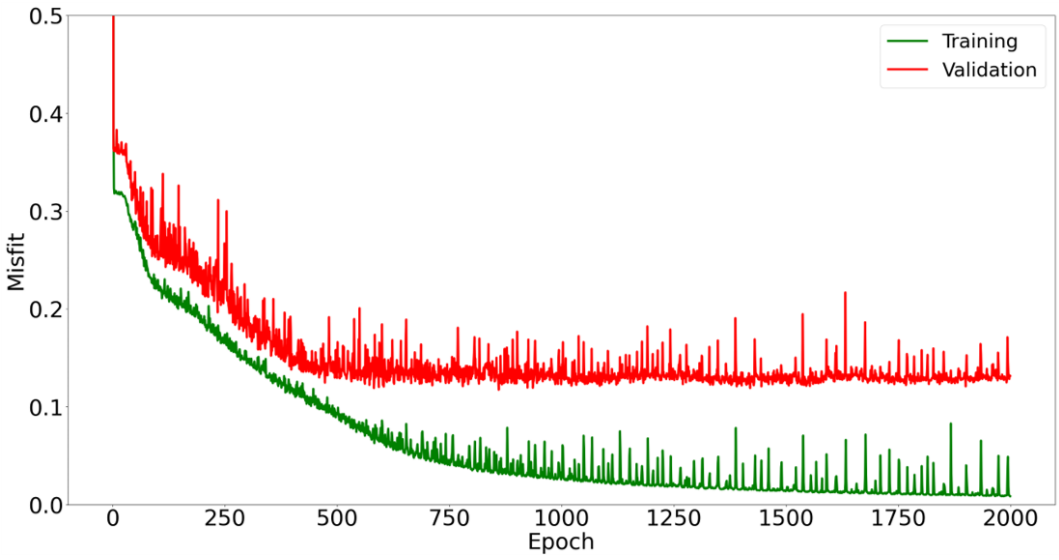

Figure 7. Training (green) and validation (red) learning curves for the RNN used in this experiment.

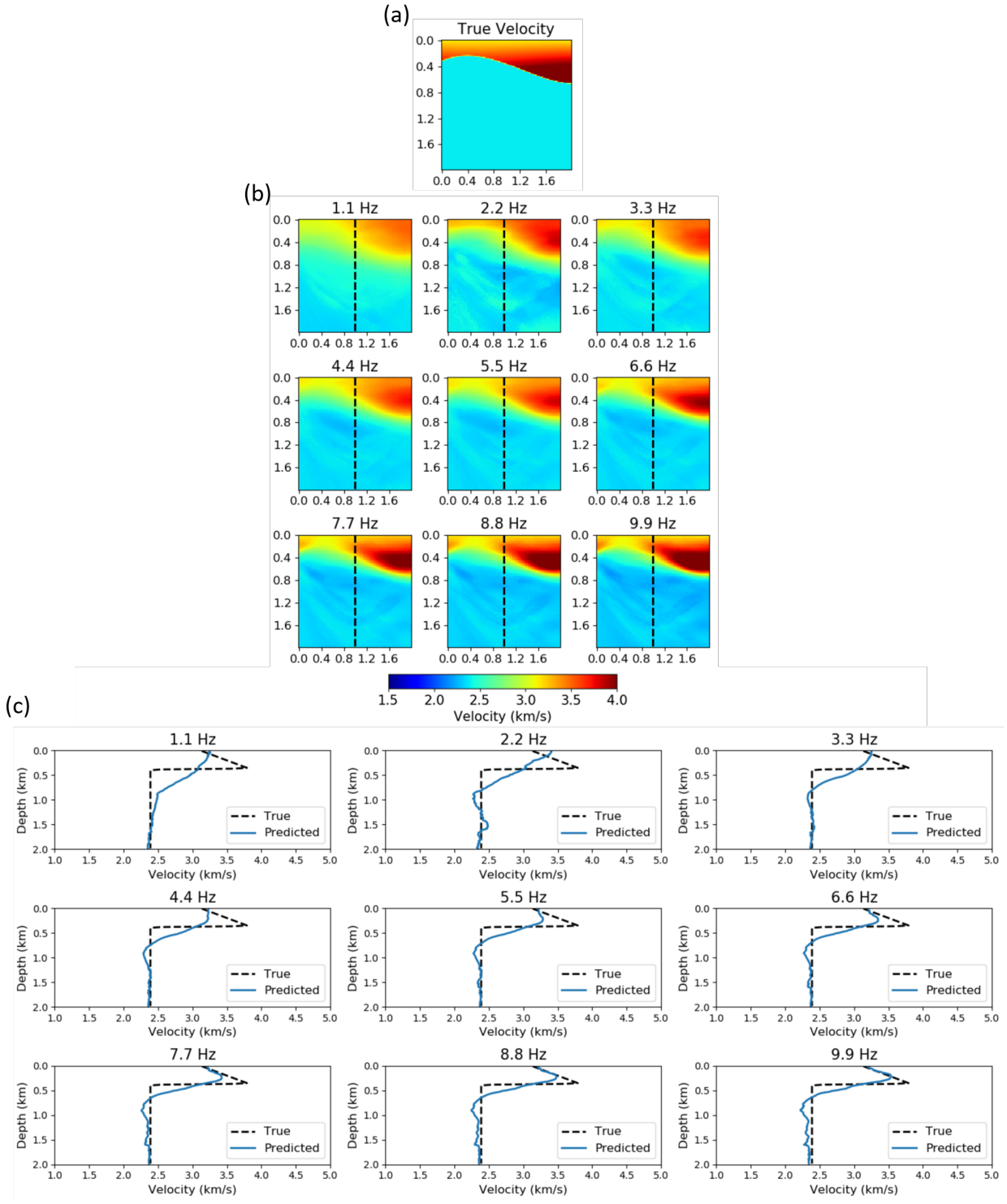

Figure 8. (a) The true testing velocity model. (b) The nine network-recovered velocity models using the trained network. (c) Vertical cross-sections extracted at the dashed line of each corresponding model in (b). The same color bar is used for all velocity models.

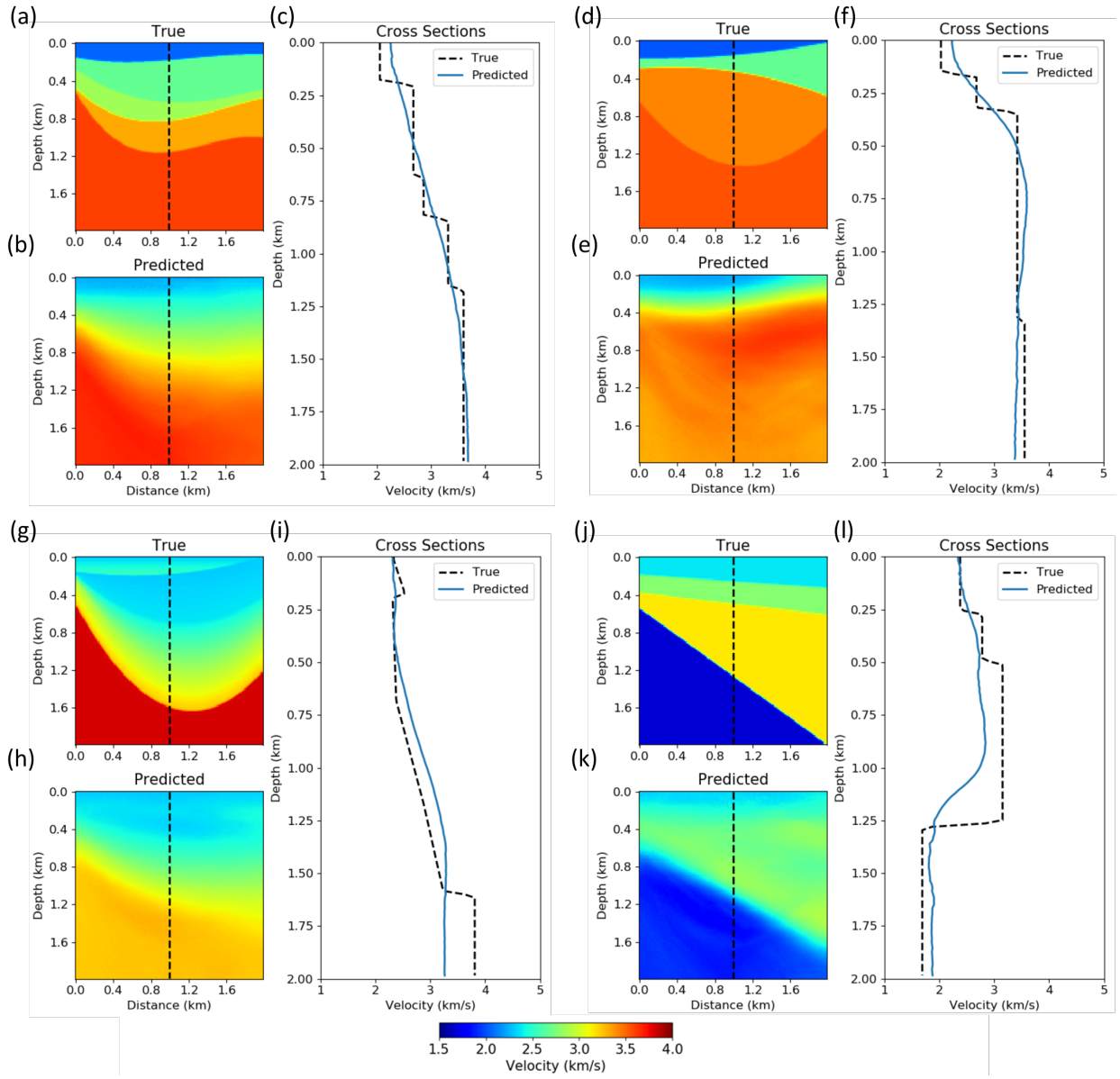

Figure 9. Four testing results of the trained network. (a), (d), (g), and (j) are the testing velocity models. (b), (e), (h) and (k) are the corresponding recovered models. (c), (f), (i), and (l) show the cross-section through each of the corresponding models at the dashed line. The same color bar is used for all velocity models.

interpolate the recovered velocities back to their original dimensions of 500×500 grid points to perform following quantitative analysis. Herein, we use an NRMS metric given by

$$NRMS = \frac{1}{2}\sqrt{\frac{\sum\left(\Phi_{True} - \Phi_{Pred}\right)^2}{\sum \Phi_{True}^2}}, \tag{1}$$

where Φ_{True} and Φ_{Pred} refer to the true and predicted data, respectively. The NRMS measure will be zero for perfectly matched data, equal to unity where $\Phi_{True} = -\Phi_{Pred}$, and be 0.5 when $\Phi_{Pred} = 0$. Figure 10 shows the distribution of the NRMS error between all 5000 true and 9.9 Hz recovered models. The NRMS misfit has a mean error of around 5.8%, which helps to quantitatively support and generalize the observations made about Figure 9 above.

To further analyze trained network performance, we use the recovered models to simulate seismic data. We interpolated the

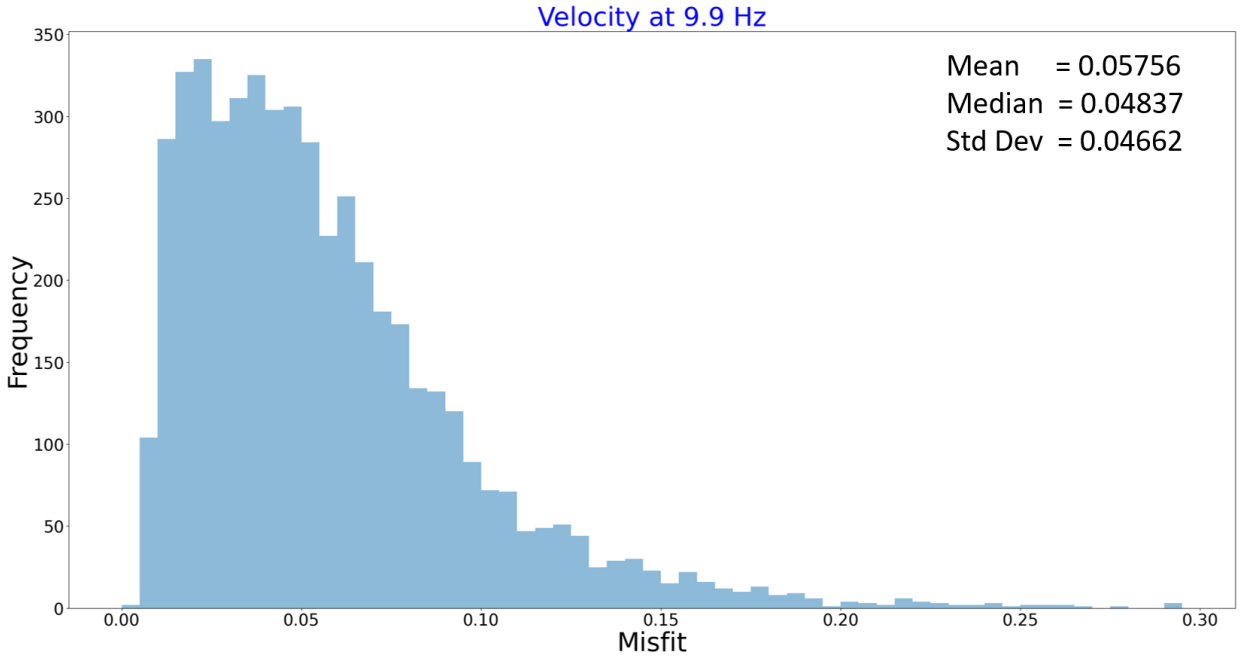

Figure 10. A histogram showing the NRMS misfit between the true and recovered 9.9 Hz velocity models calculated using equation 1.

9.9 Hz recovered velocity models from 125×125 into the original 500×500 size and then used them to synthesize seismic data using the same modeling procedure discussed above. Figure 11a and 11c compare the associated true and predicted velocity models. The recovered velocity model appears to have the correct long-wavelength components of the true model. This observation is confirmed by the fact that both the amplitude and phase curves in Figure 11d and 11d of the synthesized and true data are almost identical at 2.2 Hz. While still being very close, the two curves somewhat diverge as the frequency increases to 9.9 Hz. In the second forward modeling test (Figure 12), the recovered model accurately recovers the long-wavelength components of the true model, though it fails to estimate the velocity of the deeper layer. However, this inaccuracy seems to have little effect on the seismic data up to the highest frequency used in the experiment. This observation reinforces our assertion that the under-performance in the recovery of some models may be caused by poor illumination in the true and recovered data alike.

In the third testing example (Figure 13), the recovered model well represents the shallower model structure and heads in the right direction of the velocity inversion at the deeper layer, but it fails to estimate the true velocity. Similar to the previous example, this inaccuracy of the recovered model seems to have a negligible effects on the synthesized data in the form of a cycle-skipped phase shift at 9.9 Hz. The fourth testing example (Figure 14) captures the general trend of the true velocity model from top to bottom. However, the misfit between true and recovered data appears greater than some previous examples with a less accurate velocity fit. This could be due to the great variability in the true data, even at 2.2 Hz, which are most likely caused by factors other than the velocity structure. When using these seismograms as input to the RNN, the data simulated using the recovered model appear to be a smoother version of the true data.

To better understand how well the RNN-generated models fit the true data, we compute the ensemble NRMS misfit (equation 1) of the amplitude and unwrapped phase components for data simulated using the true and 9.9 Hz recovered models. Figure 15a and 15b show the ensemble NRMS of the amplitude and unwrapped phase misfits, respectively. The amplitude and phase error distributions have means of 7.4% and 4.7%, which are broadly consistent with the observations from Figure 10.

5 CONCLUSIONS

We develop a novel ANN technique that is inspired by the FWI multi-grid approach that uses sequence-to-sequence RNNs. This approach successively injects higher data frequencies at each step. To constrain the information carried through the network by the hidden state vector, we also inject a smoothed velocity model as the output for each input frequency. The degree of smoothness depends on the frequency being injected at that RNN step. The presented examples along with quantitative analysis demonstrates

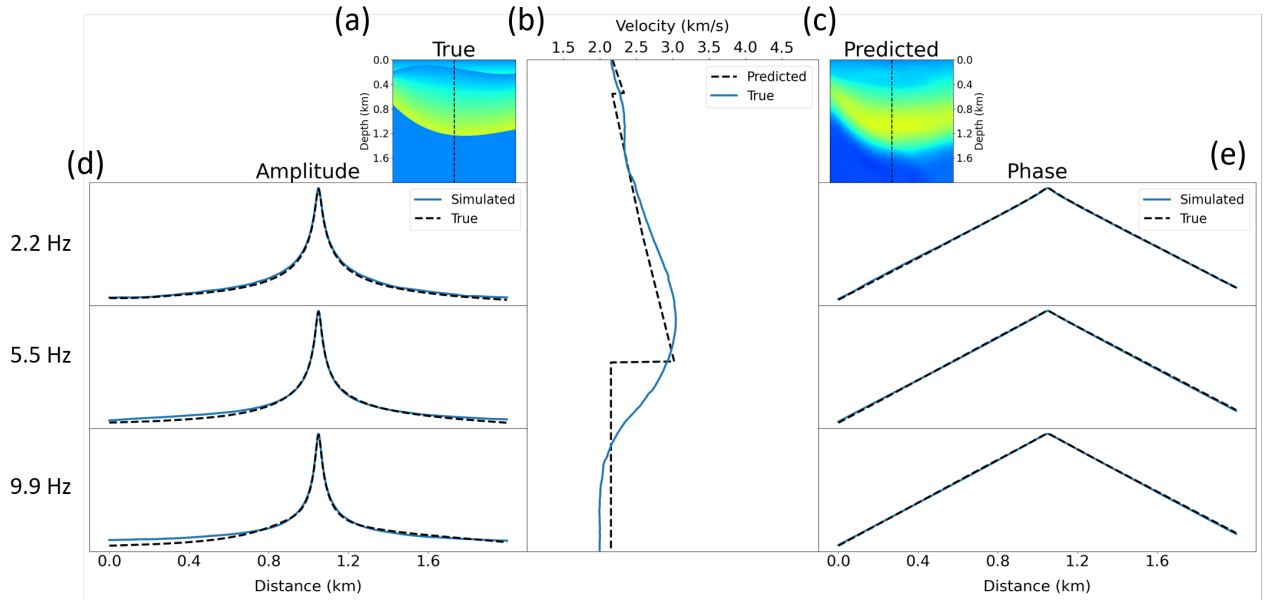

Figure 11. Comparison of the amplitude and phase components of the data simulated using the RNN-recoverd models. (a) True model. (b) Cross-sections through the true and recovered models at the dashed line. (c) Recovered model at 9.9 Hz. (d) Amplitude and (e) unwrapped phase comparisons for data modeled using the model in (c) at 2.2 Hz, 5.5 Hz and 9.9 Hz.

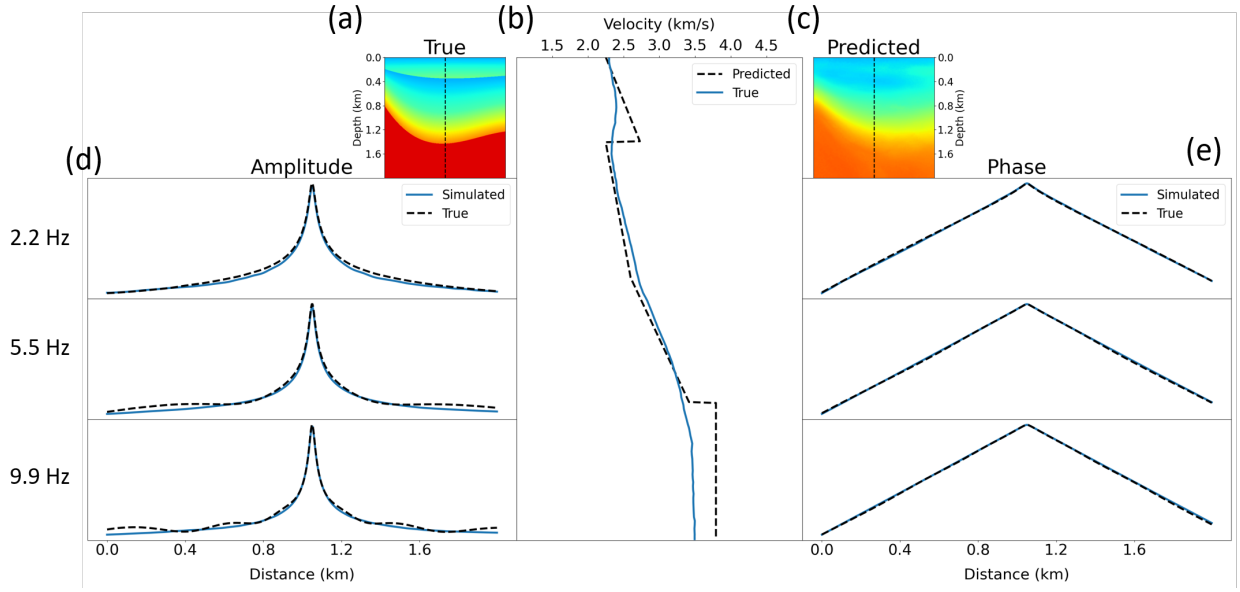

Figure 12. Comparison of the amplitude and phase components of the data simulated using the RNN-recoverd models. (a) True model. (b) Cross-sections through the true and recovered models at the dashed line. (c) Recovered model at 9.9 Hz. (d) Amplitude and (e) unwrapped phase comparisons for data modeled using the model in (c) at 2.2 Hz, 5.5 Hz and 9.9 Hz.

that the LSTM RNN network is capable of building relatively complex velocity models with high accuracy. These results are further corroborated by the forward modeling results that show data simulated with the true and 9.9 Hz recovered models match well. This holds even for models that do appear somewhat dissimilar when viewed in the velocity model domain. This means that much of the velocity differences fall within the null space of the experiment.

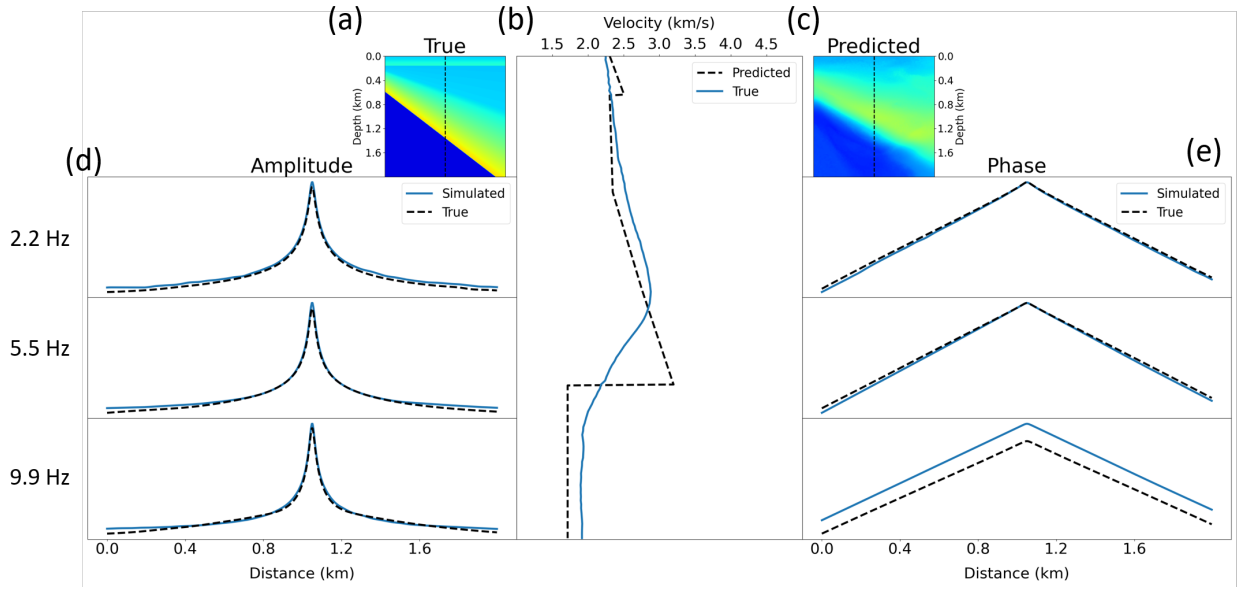

Figure 13. Comparison of the amplitude and phase components of the data simulated using the RNN-recovered models. (a) True model. (b) Cross-sections through the true and recovered models at the dashed line. (c) Recovered model at 9.9 Hz. (d) Amplitude and (e) unwrapped phase comparisons for data modeled using the model in (c) at 2.2 Hz, 5.5 Hz and 9.9 Hz. Note that the unwrapped phase shows cycle skipping at the 9.9 Hz frequency.

6 ACKNOWLEDGEMENTS

We thank Saudi Aramco (HA) and the other CWP sponsors whose support made this research possible. Computations were completed using the CSM *Wendian* facilities, the TensorFlow package and the Madagascar software environment (www.ahay.org).

REFERENCES

Araya-Polo, M., J. Jennings, A. Adler, and T. Dahlke, 2018, Deep-learning tomography: The Leading Edge, **37**, 58–66.

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

Bunks, C., F. M. Saleck, S. Zaleski, and G. Chavent, 1995, Multiscale seismic waveform inversion: Geophysics, **60**, 1457–1473.

He, Q., and Y. Wang, 2021, Reparameterized full-waveform inversion using deep neural networks: Geophysics, **86**, no. 1, V1–V13.

Hochreiter, S., and J. Schmidhuber, 1997, Long short-term memory: Neural Computation, **9**, no. 8, 1735–1780.

Lailly, P., 1983, The seismic inverse problem as a sequence of pre-stack migration, *in* Bednar, J.B. and R. Redner, E. Robinson, and A. Weglein, Eds., Conference on inverse scattering: Theory and Applications: Society of Industrial and Applied Mathematics.

Phan, S., and M. Sen, 2018, Hopfield networks for high-resolution prestack seismic inversion: SEG Technical Program Expanded Abstracts, 526–530.

Sun, J., Z. Niu, K. Innanen, J. Li, and D. Trad, 2020, A theory-guided deep-learning formulation and optimization of seismic waveform inversion: Geophysics, **85**, no. 2, R87–R99.

Sutskever, I., O. Vinyals, and Q. Le, 2014, Sequence to sequence learning with neural networks: arXiv, http://arxiv.org/abs/1409.3215.

Tarantola, A., 1984, Inversion of seismic reflection data in the acoustic approximation: Geophysics, **49**, 1259–1266.

Vamaraju, J., and M. Sen, 2019, Unsupervised physics-based neural networks for seismic migration: Interpretation, **7**, no. 3, SE189–SE200.

Virieux, J., and S. Operto, 2009, An overview of full-waveform inversion in exploration geophysics: Geophysics, **74**, no. 6, WCC1–WCC26.

Zhang, Z., and T. Alkhalifah, 2019, Regularized elastic full-waveform inversion using deep learning: Geophysics, **84**, no. 5, R741–R751.

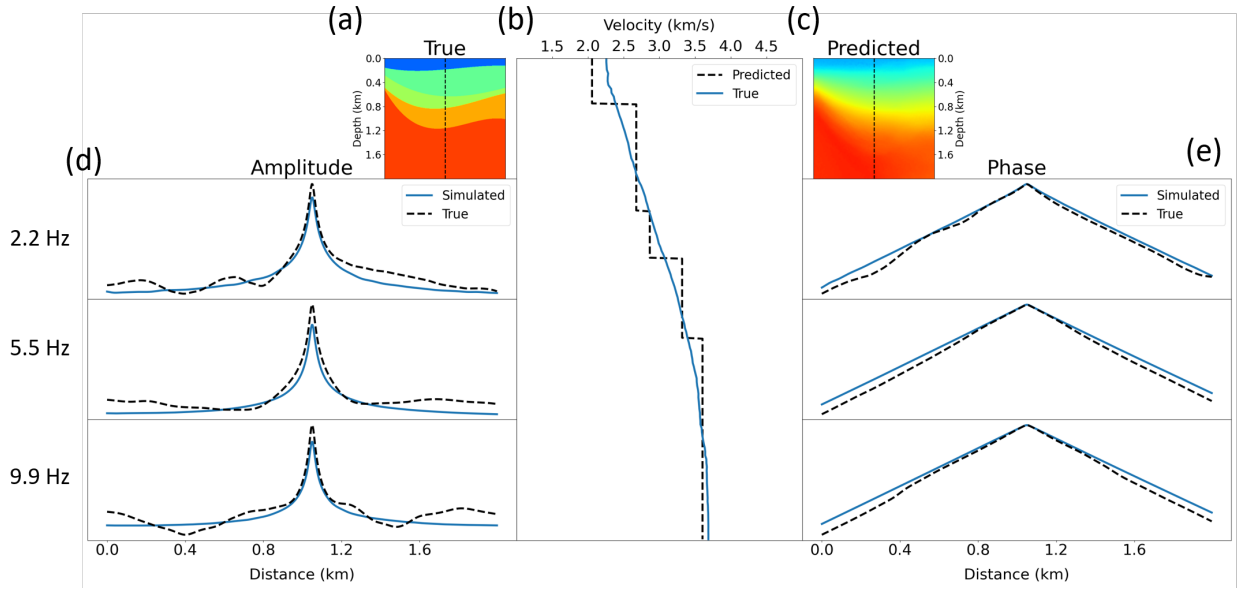

Figure 14. Comparison of the amplitude and phase components of the data simulated using the RNN-recoverd models. (a) True model. (b) Cross-sections through the true and recovered models at the dashed line. (c) Recovered model at 9.9 Hz. (d) Amplitude and (e) unwrapped phase comparisons for data modeled using the model in (c) at 2.2 Hz, 5.5 Hz and 9.9 Hz.

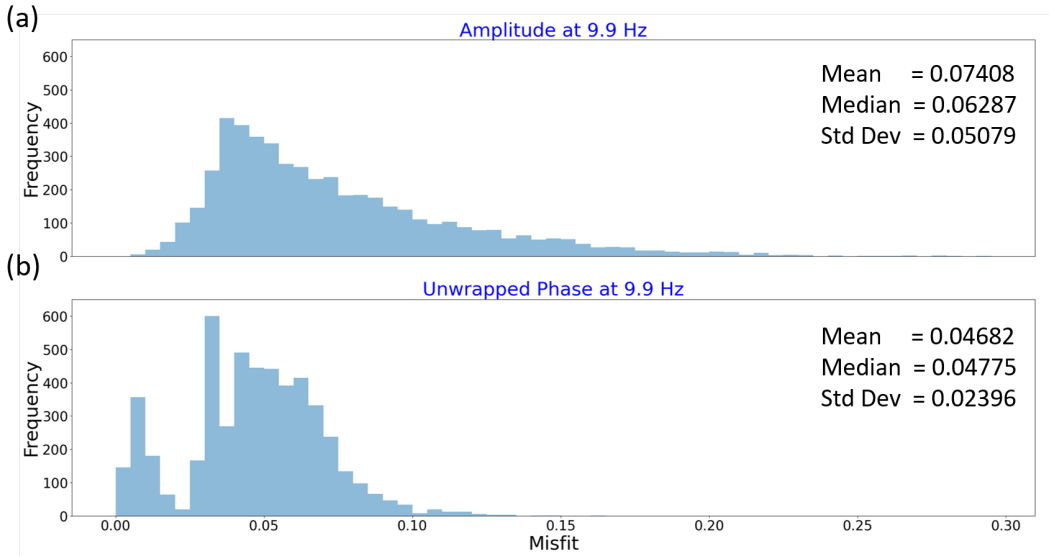

Figure 15. Histograms showing the NRMS misfit between data modeled through the true and 9.9 Hz recovered model calculated using equation 1. (a) Amplitude. (b) Unwrapped phase.